



UVM Project

HU_VM

MONAYA WAEL

MENNATU-ALLAH ABD ELFATTAH

MARWAN SAMIR EZZ-ELDIN

Verification Plan

SPI Slave

A	B	C	D	E	F
Label	Description	Stimulus Generation	Functional Coverage	Functionality Check	
SPI_1	When reset (rst_n) is asserted, all outputs (MISO, rx_valid, rx_data) should be low.	Directed sequence at the beginning of the simulation, then randomized with constraint that drives reset to be deasserted 98% of the time.	Cover reset asserted/deasserted. Cross with SS_n and MISO to ensure outputs are inactive when reset is high.	Assertion: when reset is low, outputs must remain zero.	
SPI_2	SS_n signal must be high for one cycle every 13 cycles for normal cases, and every 23 cycles for read data operations.	Constraint on SS_n inside main sequence; inline constraints switch between 13 and 23 cycle patterns.	Cover two transaction types: Normal: 1→0[13]→1, Read Data: 1→0[23]→1.	Assertion to ensure proper SS_n transaction duration per opcode.	
SPI_3	The first 3 bits of MOSI after SS_n falls must be valid command combinations (000, 001, 110, 111).	Randomized array MOSI_arr[10:0]; constraint to generate only legal command sequences for first 3 bits.	Cover valid MOSI patterns: 000, 001, 110, 111. Add illegal bin for negative testing.	Output checked against golden model; illegal combinations should result in inactive MISO.	
SPI_4	For Write Address (000), Write Data (001), Read Address (110), and Read Data (111) commands, correct response must be generated.	Main sequence generates randomized MOSI data based on command; tx_valid forced high only for read data operations.	Cross coverage between MOSI opcode bins and SS_n bins to ensure all valid command-transaction pairs are covered.	Output checked against golden model.	
SPI_5		Inline constraint in main sequence when tx_valid==111.	Coverpoint on tx_valid with cross on MOSI = 111.	Assertion that tx_valid is asserted only for read data opcode.	
SPI_6	tx_valid signal must be high during read data operation.	Use counter-based delay check in monitor or scoreboard.	Coverpoint on rx_valid latency (number of cycles after command).	Assertion: after command (000,001,110,111), rx_valid must assert exactly after 10 cycles.	
SPI_7	Ensure that the MISO output becomes valid 10 cycles after a valid command.	Randomized command sequences exercising all transitions.	Cover FSM transition bins for all legal transitions.	Assertions guarded by 'ifdef SIM': IDLE→CHK_CMD, CHK_CMD→WRITE/READ_ADDR/READ_DATA, WRITE→IDLE, READ_ADD→IDLE, READ_DATA→IDLE.	

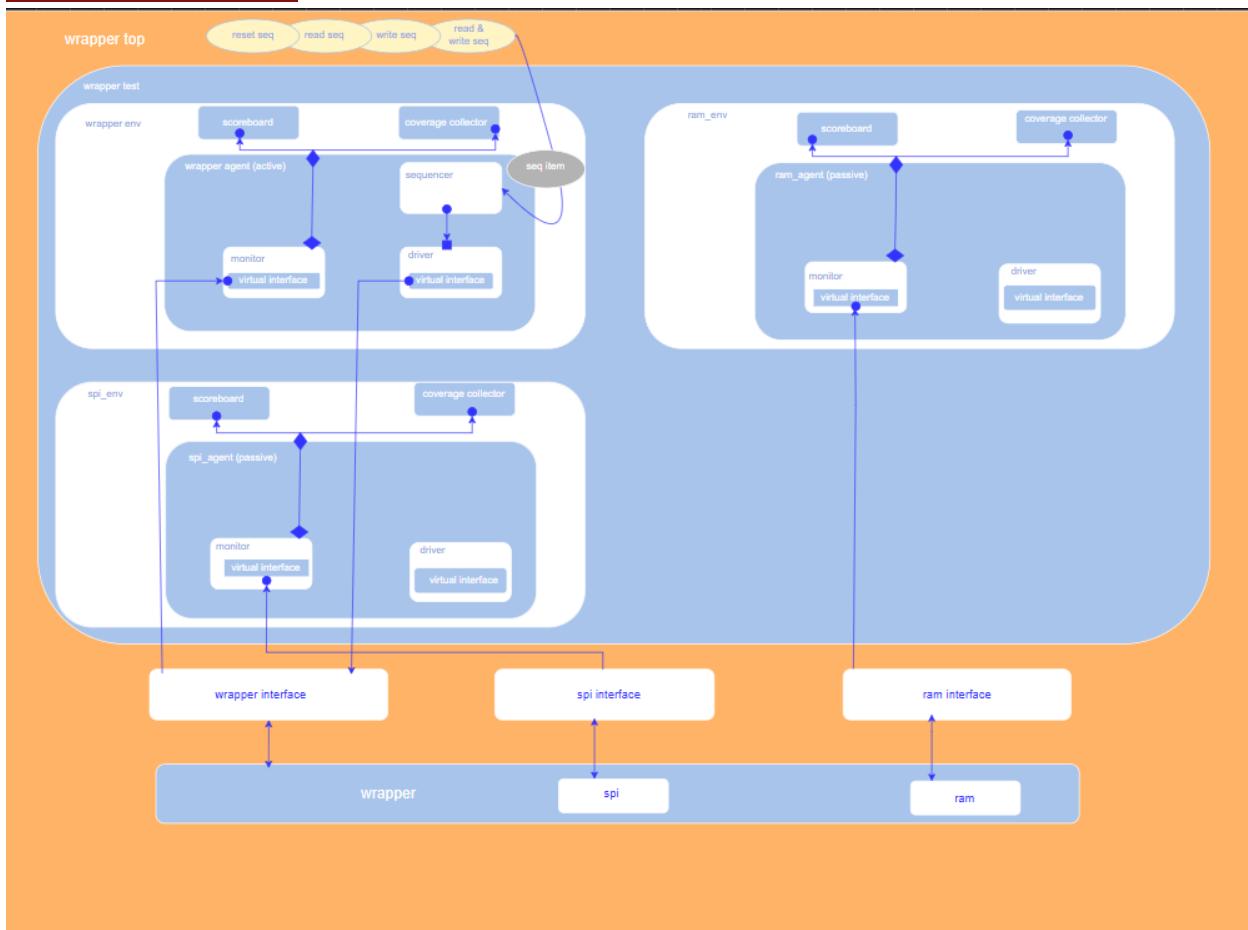
RAM

A	B	C	D	E
Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
RAM_1	When reset (rst_n) is asserted, output signals (tx_valid and dout) must be low.	Directed reset_sequence at start of sim and occasional directed reset injections. In seq_item constrain rst_n distribution to be deasserted most of the time (e.g., 95% deasserted).	Cover reset asserted / deasserted and durations. Cross reset with tx_valid and dout to ensure outputs are low during reset.	Assertion: whenever rst_n=0 then tx_valid==0 & dout==0. Checker compares golden model outputs.
RAM_2	rx_valid must be asserted most of the time (stimulus characteristic).	Constrain rx_valid distribution in seq_item (e.g., rx_valid dist [1~95,0~5]) or use pre_randomize to bias generation.	Coverpoint on rx_valid (high/low) with duration bins and cross with command bins.	Monitor that rx_valid is high per constraint and scoreboard tracks behavior vs golden model.
RAM_3	Write-only sequence ordering rule: every WRITE_ADDR must be followed by WRITE_ADDR or WRITE_DATA.	Implement WRITE_ONLY mode in sequence: use inline constraints or c_seq in seq_item to limit next command choices when seq_mode==WRITE_ONLY.	Coverpoint on command ordering: WRITE_ADDR→WRITE_ADDR and WRITE_ADDR→WRITE_DATA; bin the chains.	Assertion: every WRITE_ADDR eventually followed by WRITE_DATA (or another WRITE_ADDR) within N cycles; checker ensures memory write occurred for write data.
RAM_4	Read-only sequence ordering: every READ_ADDR must be followed by READ_DATA and after READ_DATA must be followed by READ_ADDR.	Implement READ_ONLY mode sequence and constraints to enforce alternation between READ_ADDR and READ_DATA.	Coverpoint on command ordering: READ_ADDR→READ_DATA and READ_ADDR→READ_ADDR; cover transition sequences.	Assertion: every READ_ADDR eventually followed by READ_DATA; scoreboard checks dout and tx_valid timing.
RAM_5	Randomized read/write sequence ordering rules with specified probabilities.	In READ_WRITE mode use pre_randomize/post_randomize or inline constraints to implement probabilistic next-command selection: after WRITE_DATA → 60% READ_ADDR, 40% WRITE_ADDR; after READ_DATA → 60% WRITE_ADDR, 40% READ_ADDR. Ensure READ_ADDR always followed by READ_DATA; WRITE_ADDR followed by WRITE_ADDR/WRITE_DATA.	Cover sequences and probabilities by sampling many runs; cover transitions WRITE_ADDR→WRITE_DATA, WRITE_DATA→READ_ADDR, READ_DATA→WRITE_ADDR, etc.	Checker validates ordering rules, probabilities (statistical check), and that data flows correctly between writes and reads.
RAM_6	din[9:8] data-cover: check din[9:8] takes all 4 possible values and transitions.	Randomize din payload and command fields; ensure seq_item generates all four din[9:8] values via constraints or directed sequences for edge cases.	Coverpoint on din[9:8] with bins for 00,01,10,11 and transition bins between them. Also add sequence chain bin.	Scoreboard verifies that payload used in write shows up in dout when read and that transitions follow protocol.
RAM_7	Cross coverage between din[9:8] bins and rx_valid being high.	Drive transactions with rx_valid asserted (biased) and sample din values during rx_valid windows.	Cross din[9:8] with rx_valid==1, exclude rx_valid==0. Ensure all din values are seen while rx_valid is asserted.	Checker ensures rx_valid accompanies address/data phases; assertions flag if din sampled when rx_valid==0.
RAM_8	Cross coverage between din[9:8]==READ_DATA and tx_valid==1.	Explicitly drive read_data transactions where din[9:8]==1 and observe tx_valid behavior in subsequent cycles.	Cross din[9:8]==READ_DATA with tx_valid==1 bins; ensure reads produce tx_valid high as expected.	Assertion: when READ_DATA occurs, tx_valid must pulse high (and eventually fall). Scoreboard verifies dout matches memory content.
RAM_9	tx_valid must be low during address or data input phases.	During input-phase sequences (WRITE_ADDR, WRITE_DATA, READ_ADDR) ensure tx_valid==1 and tx_valid==0 via constraints on sequences or monitor checks.	Cover that tx_valid is zero in input phases by sampling these phases and counting occurrences.	Assertion: during address/data input phases tx_valid==0. Failure report includes timestamp and sequence item context.
RAM_10	every WRITE_ADDR eventually followed by WRITE_DATA and every READ_ADDR eventually followed by READ_DATA and tx_valid pulse behavior.	Combine directed and randomized sequences to force long traces that exercise eventuality properties; use liveness checks with timeout windows.	Cover long chains such as: WRITE_ADDR→WRITE_DATA→READ_ADDR→READ_DATA A and transitions across multiple cycles.	Assertions for eventuality: WRITE_ADDR → eventually WRITE_DATA; READ_ADDR → eventually READ_DATA; READ_DATA → tx_valid rising then falling. Bind assertions in top module and run coverage.

Wrapper

A	B	C	D	E
Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
Wrapper_1	Reset behavior: when rst_n is asserted, MISO (and other outputs) must be inactive.	Directed reset_sequence at start of sim; bias rst_n distribution in seq_items to be deasserted most of the time (e.g., 95% deasserted).	Cover reset asserted/deasserted, durations, and cross reset with MISO, tx_valid (RAM) and rx_valid (SPI).	Assertion (bound in top): when rst_n==0, MISO==0. Also bind RAM and SPI assertions in top-level. Checker compares DUT to golden model during post-reset operation.
Wrapper_2	SS_n timing: normal transactions use 13-cycle SS low period; READ_DATA uses 23-cycle low period.	Main sequences drive SS_n using inline constraints: ss_period=13 or ss_period=23 depending on first-3 MOSI bits (111->23 else 13).	Cover SS_n timing bins: normal (13) and extended (23). Cross SS_n bins with MOSI opcode bins.	Assertion: when MOSI opcode indicates READ_DATA, SS_n must return high after 23 cycles; otherwise after 13 cycles.
Wrapper_3	MOSI command restriction: first 3 bits after SS_n fall must be one of (000,001,110,111).	Randomized MOSI bit-array in seq_item; use pre_randomize/post_randomize and inline constraints to force first 3 bits inside allowed set.	Cover MOSI first 3-bits with bins for each opcode and transitions between opcodes across transactions.	Checker decodes MOSI and verifies legality; illegal commands flagged. Cross MOSI with RAM command mapping.
Wrapper_4	Write-only ordering: every WRITE_ADDR must be followed by WRITE_ADDR or WRITE_DATA.	Implement write_only_sequence with seq_mode flag; use inline constraints in seq_item to restrict next command choices when seq_mode==WRITE_ONLY.	Cover bins for WRITE_ADDR→WRITE_ADDR and WRITE_ADDR→WRITE_DATA chains and longer chains.	Assertion: every WRITE_ADDR eventually followed by WRITE_DATA or another WRITE_ADDR within N cycles.
Wrapper_5	Read-only ordering: READ_ADDR↔READ_DATA alternation.	Implement read_only_sequence that toggles between READ_ADDR and READ_DATA using constraints in seq_item.	Cover READ_ADDR→READ_DATA and READ_DATA→READ_ADDR transitions and frequency.	Assertion: every READ_ADDR must be followed by READ_DATA. Scoreboard checks dout and tx_valid timing for reads.
Wrapper_6	Randomized read/write ordering with probabilities.	Implement write_read_sequence (READ_WRITE mode) with pre_randomize/post_randomize or inline constraints to implement probabilistic distributions: after WRITE_DATA → 60% READ_ADDR, 40% WRITE_ADDR; after READ_DATA → 60% WRITE_ADDR, 40% READ_ADDR. Enforce mandatory follow-ups where required.	Collect statistical coverage: frequency of transitions and long-run distribution matching target probabilities. Cover chains like WRITE_ADDR→WRITE_DATA→READ_ADDR→READ_DATA.	Checker validates enforced order and logs statistical distribution; assertions enforce mandatory immediate-follow rules (e.g., READ_ADDR→READ_DATA).
Wrapper_7	MISO stability: MISO remains stable (unchanged) unless in READ_DATA operation.	Driver/monitor must sample MISO between transactions; sequences should create idle windows to observe stability.	Coverpoint on MISO change vs command: bins for stable-during-non-read and changes-during-read. Cross with MOSI opcode.	Assertion (bound in top): if not in READ_DATA state then MISO must not change (stable value eventually).
Wrapper_8	rx_valid and tx_valid behavior across integrated DUT.	Bias rx_valid in SPI agent sequences as required; ensure RAM agent sequences use rx_valid to accept din. Pass tx agents' read monitors to check signals.	Cover rx_valid high occurrences and cross rx_valid with din[9:8] and tx_valid.	Assertion: tx_valid low during input phases; tx_valid pulses following READ_DATA and then falls. Cross-check timing between SPI and RAM outputs.
Wrapper_9	End-to-end data correctness: data written via SPI must be read back correctly through integrated RAM.	Stimulate sequences that perform WRITE_ADDR→WRITE_DATA then later READ_ADDR→READ_DATA using sequence pairs that map SPI transactions to RAM transactions.	Cover end-to-end scenarios: write-then-read success bins, multibank checks, and data integrity across different addresses and data values.	Scoreboard compares read-back dout with original write payload; raise error on mismatch.
Wrapper_10	Passive agent Integration & bindings: reuse RAM and SPI environments with passive agents in wrapper environment.	Instantiate SPI and RAM agents in the top-level UVM env; set their driver active/passive flags appropriately. Use monitors to connect to top-level scoreboard.	Cover that both agents observed expected traffic; cross agent observations to validate mapping between SPI transactions and RAM transactions.	Ensure monitors provide transactions to scoreboard; assertions for interface-level checks are bound in top module for DUT and in monitor checks.

UVM Structure



Code Snippets

SPI Slave

➤ Sequence Item package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > SPI_seq_item.sv ...
1  package spi_seq_item_pkg;
2
3  // Importing needed packages and macros
4  import uvm_pkg::*;
5  `include "uvm_macros.svh"
6
7  // Creating the class
8  class spi_seq_item extends uvm_sequence_item;
9    `uvm_object_utils(spi_seq_item)
10
11  // Constructor
12  function new(string name = "spi_seq_item");
13    super.new(name);
14  endfunction
15
16  // Randomizing stimulus inputs and outputs
17  rand logic rst_n, SS_n, tx_valid;
18  rand logic [7:0] tx_data;
19  logic MOSI;
20
21  // Outputs
22  logic [MISO: rx_valid];
23  logic [9:0] rx_data;
24
25  // Reference outputs
26  logic MISO_ref, rx_valid_ref;
27  logic [9:0] rx_data_ref;
28
29  // For constraints
30
31  // Reset constraint
32  constraint rst_c {
33    rst_n dist {0 := 2, 1 := 98};
34  }
35
36  // Constraints
37  // Reset constraint
38  constraint ss_c {
39    SS_n dist {0 := 2, 1 := 98};
40  }
41
42  function void post_randomize();
43    {MOSI_arr[0], MOSI_arr[1], MOSI_arr[2]} = cmd;
44
45    if (ss_counter == 0)
46    | SS_n = 1;
47    else
48    | SS_n = 0;
49
50    if ((normal_counter == 0) && (read_counter == 0) && (ss_counter == 0)) begin
51      MOSI_arr_drv = MOSI_arr;
52    end
53    read_data_flag = (((MOSI_arr_drv[0], MOSI_arr_drv[1], MOSI_arr_drv[2])) == READ_DATA);
54
55    if (read_data_flag) begin
56      if (ss_counter < 23)
57        ss_counter++;
58      else
59        ss_counter = 0;
60    end
61    else begin
62      if (ss_counter < 13)
63        ss_counter++;
64      else
65        ss_counter = 0;
66    end
67
68    if (read_counter < 25 && !SS_n && read_data_flag && ss_counter > 2) begin
69      if (read_counter < 11)
70        MOSI = MOSI_arr_drv[read_counter++];
71      else
72        read_counter++;
73    end
74    else begin
75      read_counter = 0;
76      read_data_flag = 0;
77    end
78
79    if (normal_counter < 15 && !SS_n && !read_data_flag && ss_counter > 2) begin
80      if (normal_counter < 11)
81        MOSI = MOSI_arr_drv[normal_counter++];
82      else
83        normal_counter++;
84    end
85    else
86      normal_counter = 0;
87
88    tx_valid = (((MOSI_arr_drv[0], MOSI_arr_drv[1], MOSI_arr_drv[2])) == READ_DATA);
89  endfunction
90
91  // Convert to string (full)
92  function string convert2string();
93    return $sformatf(
94      "%s reset = %0b, SS_n = %0b, MOSI = %0b, MISO = %0b, rx_valid = %0b, rx_data = %0b",
95      super.convert2string(),
96      rst_n, SS_n, MOSI, MISO, rx_valid, tx_valid, rx_data, tx_data
97    );
98  endfunction
99
100 // Convert to string (stimulus only)
101 function string convert2string_stimulus();
102   return $sformatf(
103     " reset = %0b, SS_n = %0b, MOSI = %0b, tx_valid = %0b, tx_data = %0b",
104     rst_n, SS_n, MOSI, tx_valid, tx_data
105   );
106 endfunction
107
108 endclass
109
110 endpackage
```

➤ Reset Sequence package

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > SPI_reset_seq.sv > ...
1 package spi_reset_seq_pkg;
2
3 // importing needed pkgs and macros
4 import uvm_pkg::*;
5 import spi_seq_item_pkg::*;
6 `include "uvm_macros.svh"
7
8 // creating the class
9 class spi_reset_seq extends uvm_sequence #(spi_seq_item);
10   `uvm_object_utils(spi_reset_seq)
11
12   // handling the seq_item
13   spi_seq_item seq_item;
14
15   // constructor
16   function new(string name = "spi_reset_seq");
17     super.new(name);
18   endfunction
19
20   // generating randomized stimulus inputs and outputs
21   task body;
22     seq_item ~ spi_seq_item::type_id::create("seq_item");
23     start_item(seq_item);
24     seq_item.rst_n = 0;
25     seq_item.SS_n = 0;
26     seq_item.MOSI = 0;
27     seq_item.tx_valid = 0;
28     seq_item.tx_data = 0;
29     finish_item(seq_item);
30   endtask
31
32 endclass
33
34 endpackage
```

➤ Main Sequence package

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > SPI_main_seq.sv > ...
1 package spi_main_seq_pkg;
2
3 // importing needed pkgs and macros
4 import uvm_pkg::*;
5 import spi_seq_item_pkg::*;
6 `include "uvm_macros.svh"
7
8 // creating the class
9 class spi_main_seq extends uvm_sequence #(spi_seq_item);
10   `uvm_object_utils(spi_main_seq)
11
12   // handling the seq_item
13   spi_seq_item seq_item;
14
15   // constructor
16   function new(string name = "spi_main_seq");
17     super.new(name);
18   endfunction
19
20   //generating randomized stimulus inputs and outputs
21   task body;
22     seq_item = spi_seq_item::type_id::create("seq_item");
23
24     repeat(10000) begin
25       start_item(seq_item);
26       assert(seq_item.randomize());
27       finish_item(seq_item);
28     end
29   endtask
30
31 endclass
32
33 endpackage
```

➤ Sequencer package

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > SPI_sequencer.sv > ...
1 package spi_sequencer_pkg;
2
3 // importing needed pkgs and macros
4 import uvm_pkg::*;
5 import spi_seq_item_pkg::*;
6 `include "uvm_macros.svh"
7
8 // creating the class
9 class spi_sequencer extends uvm_sequencer #(spi_seq_item);
10   `uvm_component_utils(spi_sequencer)
11
12   // constructor
13   function new(string name = "spi_sequencer", uvm_component parent = null);
14     super.new(name, parent);
15   endfunction
16
17 endclass
18
19 endpackage
```

➤ Design Interface

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > SPI_interface.sv > ...
1  interface SPI_IF #(
2    parameter IDLE = 3'b000,
3    parameter CHK_CMD = 3'b001,
4    parameter WRITE = 3'b010,
5    parameter READ_ADDRESS = 3'b011,
6    parameter READ_DATA = 3'b100
7
8  ) (input bit clk);
9
10 logic rst_n, SS_n, MOSI, MISO, rx_valid, tx_valid ;
11 logic [9:0] rx_data;
12 logic [7:0] tx_data;
13
14 endinterface
15 |
```

➤ Golden Model Interface

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > SPI_GM_interface.sv > ...
1  interface SPI_GM_IF #(
2    parameter IDLE = 3'b000,
3    parameter CHK_CMD = 3'b001,
4    parameter WRITE = 3'b010,
5    parameter READ_ADDRESS = 3'b011,
6    parameter READ_DATA = 3'b100
7
8  ) (input bit clk);
9
10 logic rst_n, SS_n, MOSI, MISO, rx_valid, tx_valid ;
11 logic [9:0] rx_data;
12 logic [7:0] tx_data;
13
14 endinterface
15 |
```

➤ Driver Package

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > spi_driver.sv > ...
1  package spi_driver_pkj;
2  // importing needed pkgs and macros
3  import uvm_pkg::*;
4  import spi_seq_item_pkj::*;
5  `include "uvm_macros.svh"
6
7  class spi_driver extends uvm_driver #(spi_seq_item);
8    `uvm_component_utils(spi_driver)
9
10   //giving handles
11   virtual SPI_IF SPI_vif;
12   virtual SPI_GM_IF SPI_GM_vif;
13   spi_seq_item stim_seq_item;
14
15   //constructor
16   function new(string name = "spi_driver", uvm_component parent = null);
17     super.new(name, parent);
18   endfunction //new()
19
20   task run_phase(uvm_phase phase);
21     super.run_phase(phase);
22     forever begin
23       stim_seq_item = spi_seq_item::type_id::create("stim_seq_item",this);
24       seq_item_port.get_next_item(stim_seq_item);
25       SPI_vif.rst_n= stim_seq_item.rst_n;
26       SPI_vif.SS_n= stim_seq_item.SS_n;
27       SPI_vif.tx_valid = stim_seq_item.tx_valid;
28       SPI_vif.tx_data = stim_seq_item.tx_data;
29       SPI_vif.MOSI = stim_seq_item.MOSI;
30       if (SPI_GM_vif != null) begin
31         SPI_GM_vif.rst_n= stim_seq_item.rst_n;
32         SPI_GM_vif.SS_n= stim_seq_item.SS_n;
33         SPI_GM_vif.tx_valid = stim_seq_item.tx_valid;
34         SPI_GM_vif.tx_data = stim_seq_item.tx_data;
35         SPI_GM_vif.MOSI = stim_seq_item.MOSI;
36       end
37
38       @(posedge SPI_vif.clk);
39       seq_item_port.item_done();
40
41       `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(),UVM_HIGH)
42
43     end
44   endtask: run_phase
45
46 endpackage
47 |
```

➤ Monitor Package

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > SPLmonitor.sv > ...
1 package spi_monitor_pkg;
2 // importing needed pkgs and macros
3 import uvm_pkg::*;
4 import spi_seq_item_pkg::*;
5 `include "uvm_macros.svh"
6
7 // creating the class
8 class spi_monitor extends uvm_component;
9   `uvm_component_utils(spi_monitor)
10
11   // analysis port of agent
12   uvm_analysis_port #(spi_seq_item) mon_aport;
13
14   // giving handles
15   virtual SPI_IF SPI_vif;
16   virtual SPI_GM_IF SPI_GM_vif;
17   spi_seq_item res_seq_item;
18
19   // constructor
20   function new(string name = "spi_monitor", uvm_component parent = null);
21     super.new(name, parent);
22   endfunction
23
24   //build phase
25   function void build_phase(uvm_phase phase);
26     super.build_phase(phase);
27     mon_aport = new("mon_aport", this);
28   endfunction
29
30   //run phase
31   task run_phase(uvm_phase phase);
32     super.run_phase(phase);
33     forever begin
34       res_seq_item = spi_seq_item::type_id::create("res_seq_item",this);
35       @(negedge SPI_vif.clk);
36       res_seq_item.rst_n = SPI_vif.rst_n;
37       res_seq_item.SS_n = SPI_vif.SS_n;
38       res_seq_item.MOSI = SPI_vif.MOSI;
39       res_seq_item.tx_valid = SPI_vif.tx_valid;
40       res_seq_item.tx_data = SPI_vif.tx_data;
41       res_seq_item.MISO = SPI_vif.MISO;
42       res_seq_item.rx_valid = SPI_vif.rx_valid;
43       res_seq_item.rx_data = SPI_vif.rx_data;
44
45       if (SPI_GM_vif != null) begin
46         res_seq_item.MISO_ref = SPI_GM_vif.MISO;
47         res_seq_item.rx_valid_ref = SPI_GM_vif.rx_valid;
48         res_seq_item.rx_data_ref = SPI_GM_vif.rx_data;
49       end
50
51       mon_aport.write (res_seq_item);
52       `uvm_info("run_phase", res_seq_item.convert2string(), UVM_HIGH)
53     end
54   endtask
55
56 endclass
57 endpackage
58 |
```

➤ Config Package

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > SPI_config_obj.sv > ...
1 package spi_config_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4
5 class spi_config extends uvm_object;
6   `uvm_object_utils(spi_config)
7
8   virtual SPI_IF SPI_vif;
9   virtual SPI_GM_IF SPI_GM_vif;
10  uvm_active_passive_enum is_active;
11
12  function new(string name = "spi_config");
13    super.new(name);
14    is_active = UVM_ACTIVE;
15  endfunction
16
17 endclass
18 endpackage
19 |
```

➤ Agent Package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > @ SPIagent.sv > ...
1 package spi_agent_pkg;
2   // importing needed pkgs and macros
3   import uvm_pkg::*;
4   import spi_config_pkg::*;
5   import spi_driver_pkg::*;
6   import spi_sequencer_pkg::*;
7   import spi_seq_item_pkg::*;
8   import spi_monitor_pkg::*;
9   `include "uvm_macros.svh"
10
11  // creating the class
12  class spi_agent extends uvm_agent;
13    `uvm_component_utils(spi_agent)
14    // analysis port of agent
15    uvm_analysis_port #(spi_seq_item) agent_aport;
16
17    // giving handles
18    spi_driver drv;
19    spi_sequencer sscr;
20    spi_monitor mon;
21    spi_config conf;
22
23    // constructor
24    function new(string name = "spi_agent", uvm_component parent = null);
25      super.new(name, parent);
26    endfunction
27
28    //build phase
29    function void build_phase(uvm_phase phase);
30      super.build_phase(phase);
31      if (!uvm_config_db #(spi_config)::get(this , "", "CFG" , conf)) begin
32        `uvm_fatal("build_phase", "error in getting the data");
33      end
34      if(conf.is_active == UVM_ACTIVE) begin
35        drv = spi_driver::type_id::create("drv", this);
36        sscr = spi_sequencer::type_id::create("scr", this);
37      end
38      mon = spi_monitor::type_id::create("mon", this);
39      agent_aport = new("agent_aport", this);
40    endfunction
41
42    // connect phase
43    function void connect_phase(uvm_phase phase);
44      super.connect_phase(phase);
45      if (conf.is_active == UVM_ACTIVE) begin
46        drv.SPI_vif = conf.SPI_vif;
47        drv.SPI_GM_vif = conf.SPI_GM_vif;
48        drv.seq_item_port.connect(sscr.seq_item_export);
49      end
50
51      mon.SPI_vif = conf.SPI_vif;
52      mon.SPI_GM_vif = conf.SPI_GM_vif;
53      mon.aport.connect(agent_aport);
54
55    endfunction
56
57  endclass
58 endpackage
```

➤ Scoreboard Package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > @ SPI_scoreboard.sv > ...
1 package spi_scoreboard_pkg;
2   // importing needed pkgs and macros
3   import uvm_pkg::*;
4   import spi_seq_item_pkg::*;
5   `include "uvm_macros.svh"
6
7  // creating the class
8  class spi_scoreboard extends uvm_scoreboard;
8    `uvm_component_utils(spi_scoreboard)
9
10    // analysis port of scoreboard
11    uvm_analysis_export #(spi_seq_item) sb_exp;
12    uvm_tlm_analysis_fifo #(spi_seq_item) sb_fifo;
13    int correct_count = 0, error_count = 0;
14
15    // giving handles
16    spi_seq_item seq_item_sb;
17
18    // constructor
19    function new(string name = "spi_scoreboard", uvm_component parent = null);
20      super.new(name, parent);
21    endfunction
22
23    //build phase
24    function void build_phase(uvm_phase phase);
25      super.build_phase(phase);
26      sb_exp = new("sb_exp", this);
27      sb_fifo = new("sb_fifo", this);
28
29    endfunction
30
31    // connect phase
32    function void connect_phase(uvm_phase phase);
33      super.connect_phase(phase);
34      sb_exp.connect(sb_fifo.analysis_export);
35    endfunction
36
37
```

```

37 //run phase
38 task run_phase (uvm_phase phase);
39     super.run_phase(phase);
40     forever begin
41         sb_fifo.get(seq_item_sb);
42         if (seq_item_sb.MISO != seq_item_sb.MISO_ref || seq_item_sb.rx_valid != seq_item_sb.rx_valid_ref || seq_item_sb.rx_data != seq_item_sb.rx_data_ref ) begin
43             if (seq_item_sb.MISO != seq_item_sb.MISO_ref) begin
44                 `uvm_error("run_phase", $sformatf("MISO mismatch: miso = %0b, miso_ref = %0b", seq_item_sb.MISO, seq_item_sb.MISO_ref ));
45             end
46             if (seq_item_sb.rx_valid != seq_item_sb.rx_valid_ref) begin
47                 `uvm_error("run_phase", $sformatf("RX_VALID mismatch: rx_valid = %0b, rx_valid_ref = %0b", seq_item_sb.rx_valid, seq_item_sb.rx_valid_ref ));
48             end
49             if (seq_item_sb.rx_data != seq_item_sb.rx_data_ref) begin
50                 `uvm_error("run_phase", $sformatf("RX_DATA mismatch: rx_data = %0b, rx_data_ref = %0b", seq_item_sb.rx_data, seq_item_sb.rx_data_ref ));
51             end
52             error_count++;
53         end else begin
54             `uvm_info("run_phase", $sformatf("comparison completed"),UVM_HIGH);
55             correct_count++;
56         end
57     end
58 end
59 endtask
60
61 //report phase
62 function void report_phase (uvm_phase phase);
63     super.report_phase(phase);
64     `uvm_info("report_phase", $sformatf("total correct tranxactions: %0d", correct_count),UVM_MEDIUM);
65     `uvm_info("report_phase", $sformatf("total failed tranxactions: %0d", error_count),UVM_MEDIUM);
66 endfunction
67
68 endclass
69 endpackage
70

```

➤ Coverage Package

```

D: > DigitalCourse > DigitalVerification > SYSTEM_2 > SPI_coverage.sv > ...
1 package spi_coverage_pk8;
2     // importing needed pkgs and macros
3     import uvm_pkg::*;
4     import spi_seq_item_pkg::*;
5     `include "uvm_macros.svh"
6
7     // creating the class
8     class spi_coverage extends uvm_component;
9         `UVM_COMPONENT_UTILS(spi_coverage)
10
11         // analysis port of agent
12         uvm_analysis_port #(spi_seq_item) cov_exp;
13         uvm_tlm_analysis_fifo #(spi_seq_item) cov_fifo;
14
15         // giving handles
16         spi_seq_item cov_seq_item;
17
18         // adding cover groups
19         covergroup cvr_bp;
20             // ---- rx data coverpoints -----
21             rx_data_cp: coverpoint cov_seq_item.rx_data[9:8]{
22                 bins all_vals[] = {2'b00,2'b01,2'b10,2'b11};
23                 bins wraddr_2_rdata = {2'b00 => 2'b01};
24                 bins rdaddr_2_rdata = {2'b10 => 2'b11};
25                 bins rddata_2_wraddr = {2'b11 => 2'b00};
26             }
27
28             // ---- ss_n coverpoints -----
29             ss_n_cp: coverpoint cov_seq_item.SS_n{
30                 bins normal_trans = {1 => 0 [*13] => 1};
31                 bins readdata_trans = {1 => 0 [*23] => 1};
32             }
33             SS_N_cp:coverpoint cov_seq_item.SS_n{
34                 bins normal = {1 => 0 [*4]};
35             }
36
37             MOSI_cp : coverpoint cov_seq_item.MOSI {
38                 bins write_addr = {0 => 0 => 0};
39                 bins write_data = {0 => 0 => 0};
40                 bins read_addr = {(1 => 1 => 0)};
41                 bins read_data = {(1 => 1 => 1)};
42                 bins others = default;
43             }
44
45             // ---- Crosses -----
46             SS_MOSI_c : cross SS_N_cp, MOSI_cp;
47
48         endgroup : cvr_bp;
49
50         // constructor
51         function new(string name = "spi_coverage", uvm_component parent = null);
52             super.new(name, parent);
53             cvr_bp = new();
54         endfunction
55
56         //build phase
57         function void build_phase(uvm_phase phase);
58             super.build_phase(phase);
59             cov_exp = new("cov_exp", this);
60             cov_fifo = new("cov_fifo", this);
61         endfunction
62
63         // connect phase
64         function void connect_phase(uvm_phase phase);
65             super.connect_phase(phase);
66             cov_exp.connect(cov_fifo.analysis_export);
67         endfunction
68
69         //run phase
70         task run_phase (uvm_phase phase);
71             super.run_phase(phase);
72             forever begin
73                 cov_fifo.get(cov_seq_item);
74                 cvr_bp.sample();
75             end
76         endtask
77
78     endclass
79 endpackage
80

```

➤ Env Package

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > SPI_env.sv > ...
1 package spi_env_pkg;
2
3 //importing needed pkgs
4 import uvm_pkg::*;
5 import spi_agent_pkg::*;
6 import spi_scoreboard_pkg::*;
7 import spi_coverage_pkg::*;
8 `include "uvm_macros.svh"
9
10 class spi_env extends uvm_env;
11     `uvm_component_utils(spi_env)
12
13     //giving handles
14     spi_agent agent;
15     spi_scoreboard score;
16     spi_coverage cov;
17
18     //constructor
19     function new(string name = "spi_env", uvm_component parent = null);
20         super.new(name, parent);
21     endfunction
22
23     //build_phase
24     function void build_phase (uvm_phase phase);
25         super.build_phase(phase);
26         agent = spi_agent::type_id::create("agent", this);
27         score = spi_scoreboard::type_id::create("score", this);
28         cov = spi_coverage::type_id::create("cov", this);
29     endfunction
30
31     //connect phase
32     function void connect_phase (uvm_phase phase);
33         super.connect_phase(phase);
34         agent.aport.connect(score.sb_exp);
35         agent.aport.connect(cov.cov_exp);
36     endfunction
37
38     endclass
39
40 endpackage
```

➤ Test Package

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > SPI_test.sv > ...
1 package spi_test_pkg;
2
3 //importing needed pkgs
4 import uvm_pkg::*;
5 import spi_env_pkg::*;
6 import spi_config_pkg::*;
7 import spi_reset_seq_pkg::*;
8 import spi_main_seq_pkg::*;
9 `include "uvm_macros.svh"
10
11 class spi_test extends uvm_test;
12     `uvm_component_utils(spi_test)
13
14     //giving handles
15     spi_env env;
16     spi_config conf;
17     virtual SPI_if spi_test_vif;
18     virtual SPI_GM_IF SPI_GM_vif;
19     spi_reset_seq res_seq;
20     spi_main_seq main_seq;
21
22     // constructor
23     function new(string name = "spi_test", uvm_component parent = null);
24         super.new(name, parent);
25     endfunction
26
27     //build_phase
28     function void build_phase(uvm_phase phase);
29         super.build_phase(phase);
30         env = spi_env::type_id::create("env", this);
31         conf = spi_config::type_id::create("conf", this);
32         res_seq = spi_reset_seq::type_id::create("res_seq", this);
33         main_seq = spi_main_seq::type_id::create("main_seq", this);
34         if (!uvm_config_db #(virtual SPI_IF)::get(this, "", "spi", conf.SPI_vif)) begin
35             `uvm_fatal("build_phase", "error in getting the data");
36         end
37         if (!uvm_config_db #(virtual SPI_GM_IF)::get(this, "", "spi_ref", conf.SPI_GM_vif)) begin
38             `uvm_fatal("build_phase", "error in getting the data");
39         end
40         uvm_config_db #(spi_config)::set(this, "*", "CFG", conf);
41     endfunction: build_phase
42
43     //run_phase
44     task run_phase(uvm_phase phase);
45         super.run_phase(phase);
46         phase.raise_objection(this);
47         // reset seq
48         `uvm_info("run_phase", "reset asserted", UVM_LOW);
49         res_seq.start(env.agent.sqr);
50         `uvm_info("run_phase", "reset deasserted", UVM_LOW);
51
52         // main seq
53         `uvm_info("run_phase", "main asserted", UVM_LOW);
54         main_seq.start(env.agent.sqr);
55         `uvm_info("run_phase", "main deasserted", UVM_LOW);
56         phase.drop_objection(this);
57     endtask: run_phase
58
59     endclass: spi_test
60
61 endpackage
```

➤ Design Code

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > SPI_SLAVE (SPI_IF spi_if);
1  module SPI_SLAVE (SPI_IF spi_if);
2
3    reg [3:0] counter;
4    reg      received_address;
5
6    // state registers
7    reg [2:0] cs, ns;
8
9    always @(posedge spi_if.clk) begin
10      if (~spi_if.rst_n) begin
11        cs <= spi_if.IDLE;
12      end
13      else begin
14        cs <= ns;
15      end
16    end
17
18    // next state logic
19    always @(*) begin
20      case (cs)
21        spi_if.IDLE: begin
22          if (spi_if.SS_n)
23            ns = spi_if.IDLE;
24          else
25            ns = spi_if.CHK_CMD;
26        end
27
28        spi_if.CHK_CMD: begin
29          if (spi_if.SS_n)
30            ns = spi_if.IDLE;
31          else begin
32            if (~spi_if.MOSI)
33              ns = spi_if.WRITE;
34            else begin
35              if (received_address)
36                //****fixed this bug here, from read address to read data****/
37                ns = spi_if.READ_DATA;
38              else
39                ns = spi_if.READ_ADDRESS;
40            end
41          end
42        end
43
44        spi_if.WRITE: begin
45          if (spi_if.SS_n)
46            ns = spi_if.IDLE;
47          else
48            ns = spi_if.WRITE;
49        end
50
51        spi_if.READ_ADDRESS: begin
52          if (spi_if.SS_n)
53            ns = spi_if.IDLE;
54          else
55            ns = spi_if.READ_ADDRESS;
56        end
57
58        spi_if.READ_DATA: begin
59          if (spi_if.SS_n)
60            ns = spi_if.IDLE;
61          else
62            ns = spi_if.READ_DATA;
63        end
64        //****added default case to avoid latches****
65        default: ns = spi_if.IDLE;
66      endcase
67    end
68
69    always @(posedge spi_if.clk) begin
70      if (~spi_if.rst_n) begin
71        spi_if.rx_data <= 0;
72        spi_if.rx_valid <= 0;
73        received_address <= 0;
74        spi_if.MISO <= 0;
75        spi_if.MOSI <= 0;
76        //****fixed this bug here, counter should be 0 at reset****/
77        counter <= 0;
78      end
79      else begin
80        case (cs)
81          spi_if.IDLE: begin
82            spi_if.rx_valid <= 0;
83          end
84
85          spi_if.CHK_CMD: begin
86            counter <= 10;
87          end
88
89          spi_if.WRITE: begin
90            if (counter > 0) begin
91              spi_if.rx_data[counter-1] <= spi_if.MOSI;
92              counter <= counter - 1;
93            end
94            else begin
95              spi_if.rx_valid <= 1;
96            end
97          end
98        end
99      end
100    end
101  endmodule
```

```

99      |     spi_if.READ_ADDRESS; begin
100     |       if (counter > 0) begin
101       |         spi_if.rx_data[counter-1] <= spi_if.MOSI;
102       |         counter <= counter - 1;
103     end
104     |       else begin
105       |         spi_if.rx_valid <= 1;
106       |         received_address <= 1;
107     end
108   end
109
110  |   spi_if.READ_DATA: begin
111    |     if (spi_if.tx_valid) begin
112      |       spi_if.rx_valid <= 0;
113      |       if (counter > 0) begin
114        |         spi_if.MISO <= spi_if.tx_data[counter-1];
115        |         counter <= counter - 1;
116      end
117      |      else begin
118        |        received_address <= 0;
119      end
120    end
121    |    else begin
122      |      if (counter > 0) begin
123        |        spi_if.rx_data[counter-1] <= spi_if.MOSI;
124        |        counter <= counter - 1;
125      end
126      |      else begin
127        |        spi_if.rx_valid <= 1;
128        |        /* fixed the counter reset to get all bits of miso*/
129        |        counter <= 9;
130      end
131    end
132  end
133  default : begin
134    |   spi_if.rx_data <= 0;
135    |   spi_if.rx_valid <= 0;
136    |   received_address <= 0;
137    |   spi_if.MISO <= 0;
138    |   spi_if.MOSI <= 0;
139    |   counter <= 0;
140  end
141
142  endcase
143 end
144
145
146  ifdef SIM
147
148  // sequences for command patterns
149  sequence write_addr; (#$i (spi_if.MOSI == 0) #1 ($i (spi_if.MOSI == 0) ##1 ($i (spi_if.MOSI == 0) && counter == 10); endsequence
150  sequence write_data; (#$i (spi_if.MOSI == 0) #1 ($i (spi_if.MOSI == 0) ##1 ($i (spi_if.MOSI == 1) && counter == 10); endsequence
151  sequence read_addr; (#$i (spi_if.MOSI == 1) #1 ($i (spi_if.MOSI == 1) ##1 ($i (spi_if.MOSI == 0) && counter == 10); endsequence
152  sequence read_data; (#$i (spi_if.MOSI == 1) #1 ($i (spi_if.MOSI == 1) ##1 ($i (spi_if.MOSI == 1) && counter == 9 && !received_address); endsequence
153
154  // -----
155  //          FSM PROPERTIES
156  // -----
157
158  // --- reset assertion
159  property p_rst;
160    @posedge spi_if.clk
161    (!spi_if.rst_n)
162    |=> (spi_if.MISO == 0 && spi_if.rx_valid == 0 && spi_if.rx_data == 0);
163  endproperty
164
165  property p_IDLE_to_chk;
166    @posedge spi_if.clk disable iff(!spi_if.rst_n)
167    (cs == spi_if.IDLE && !spi_if.SS_n) |=> (cs == spi_if.CHK_CMD);
168  endproperty
169
170  property p_chk_to_WRITE;
171    @posedge spi_if.clk disable iff(!spi_if.rst_n || spi_if.SS_n)
172    (cs == spi_if.CHK_CMD && !spi_if.MOSI) |=> (cs == spi_if.WRITE);
173  endproperty
174
175  property p_chk_to_READ_ADDRESS;
176    @posedge spi_if.clk disable iff(!spi_if.rst_n || spi_if.SS_n)
177    (cs == spi_if.CHK_CMD && spi_if.MOSI && !received_address) |=> (cs == spi_if.READ_ADDRESS);
178  endproperty
179
180  property p_chk_to_READ_DATA;
181    @posedge spi_if.clk disable iff(!spi_if.rst_n || spi_if.SS_n)
182    (cs == spi_if.CHK_CMD && spi_if.MOSI && received_address) |=> (cs == spi_if.READ_DATA);
183  endproperty
184
185  property p_WRITE_to_IDLE;
186    @posedge spi_if.clk disable iff(!spi_if.rst_n)
187    (cs == spi_if.WRITE && spi_if.SS_n) |=> (cs == spi_if.IDLE);
188  endproperty
189
190  property p_readdaddr_to_IDLE;
191    @posedge spi_if.clk disable iff(!spi_if.rst_n)
192    (cs == spi_if.READ_ADDRESS && spi_if.SS_n) |=> (cs == spi_if.IDLE);
193  endproperty
194
195  property p_readdata_to_IDLE;
196    @posedge spi_if.clk disable iff(!spi_if.rst_n)
197    (cs == spi_if.READ_DATA && spi_if.SS_n) |=> (cs == spi_if.IDLE);
198  endproperty

```

```

200 property p_rx_valid_wr_addr;
201   @(posedge spi_if.clk) disable iff(!spi_if.rst_n || spi_if.SS_n)
202     | (write_addr) |> ###10 (spi_if.rx_valid);
203 endproperty
204
205 property p_rx_valid_wr_data;
206   @(posedge spi_if.clk) disable iff(!spi_if.rst_n || spi_if.SS_n)
207     | (write_data) |> ###10 (spi_if.rx_valid);
208 endproperty
209
210 property p_rx_valid_rd_addr;
211   @(posedge spi_if.clk) disable iff(!spi_if.rst_n || spi_if.SS_n)
212     | (read_addr) |> ###10 (spi_if.rx_valid);
213 endproperty
214
215 property p_rx_valid_rd_data;
216   @(posedge spi_if.clk) disable iff(!spi_if.rst_n || spi_if.SS_n)
217     | (read_data) |> ###10 (spi_if.rx_valid);
218 endproperty
219
220 property p_ss_N_rd_data;
221   @(posedge spi_if.clk) disable iff(!spi_if.rst_n)
222     | (cs == spi_if.READ_DATA) |> ###23 (spi_if.SS_n[-1]);
223 endproperty
224
225 property p_ss_N_normal;
226   @(posedge spi_if.clk) disable iff(!spi_if.rst_n)
227     | (cs == spi_if.WRITE || cs == spi_if.READ_ADDRESS) |> ###13 (spi_if.SS_n[-1]);
228 endproperty
229
230 // -----
231 // FSM ASSERTIONS
232 // -----
233
234 assert property (p_rst)
235   | else $error("ASSERTION FAILED: reset didn't clear the outputs");
236
237 assert property (p_IDLE_to_chk)
238   | else $error("FSM Error: Expected spi_if.IDLE + spi_if.CHK_CMD");
239
240 assert property (p_chk_to_WRITE)
241   | else $error("FSM Error: spi_if.CHK_CMD must go to spi_if.WRITE");
242
243 assert property (p_chk_to_READ_ADDRESS)
244   | else $error("FSM Error: spi_if.CHK_CMD must go to READ_ADD");
245
246 assert property (p_chk_to_READ_DATA)
247   | else $error("FSM Error: spi_if.CHK_CMD must go to spi_if.READ_DATA");
248
249 assert property (p_WRITE_to_IDLE)
250   | else $error("FSM Error: spi_if.WRITE must return to spi_if.IDLE");
251
252 assert property (p_readdad_to_IDLE)
253   | else $error("FSM Error: READ_ADD must return to spi_if.IDLE");
254
255 assert property (p_readdata_to_IDLE)
256   | else $error("FSM Error: spi_if.READ_DATA must return to spi_if.IDLE");
257
258 assert property (p_rx_valid_wr_addr)
259   | else $error("ASSERTION FAILED: rx_valid_wr_addr did not assert after command");
260
261 assert property (p_rx_valid_wr_data)
262   | else $error("ASSERTION FAILED: rx_valid_wr_data did not assert after command");
263
264 assert property (p_rx_valid_rd_addr)
265   | else $error("ASSERTION FAILED: rx_valid_rd_addr did not assert after command");
266
267 assert property (p_rx_valid_rd_data)
268   | else $error("ASSERTION FAILED: rx_valid_rd_data did not assert after command");
269
270 assert property (p_ss_N_rd_data)
271   | else $error("ASSERTION FAILED: ss_N_rd_data did not assert after command");
272
273 assert property (p_ss_N_normal)
274   | else $error("ASSERTION FAILED: ss_N_normal did not assert after command");
275
276 // -----
277 // FSM COVERS
278 // -----
279
280 cover property (p_rst);
281 cover property (p_IDLE_to_chk);
282 cover property (p_chk_to_WRITE);
283 cover property (p_chk_to_READ_ADDRESS);
284 cover property (p_chk_to_READ_DATA);
285 cover property (p_WRITE_to_IDLE);
286 cover property (p_readdad_to_IDLE);
287 cover property (p_readdata_to_IDLE);
288 cover property (p_rx_valid_wr_addr);
289 cover property (p_rx_valid_wr_data);
290 cover property (p_rx_valid_rd_addr);
291 cover property (p_rx_valid_rd_data);
292 cover property (p_ss_N_rd_data);
293 cover property (p_ss_N_normal);
294
295 `endif
296
297 endmodule
298

```

➤ Golden Model Code

```

D:> DigitalCourse > DigitalVerification > SYSTEM_2 > @ SPI_GM.sv > ...
1  module SPI_REF (SPI_GM_IF spi_gm_if);
2
3  reg [3:0] counter;
4  reg      received_address;
5  // state registers
6  reg [2:0] cs, ns;
7  always @(posedge spi_gm_if.clk) begin
8    if (~spi_gm_if.rst_n) begin
9      | cs <- spi_gm_if.IDLE;
10     end
11    else begin
12      | cs <- ns;
13    end
14  end
15
16 // next state logic
17 always @(*) begin
18   case (cs)
19     spi_gm_if.IDLE : begin
20       if (spi_gm_if.SS_n)
21         ns = spi_gm_if.IDLE;
22       else
23         ns = spi_gm_if.CHK_CMD;
24     end
25     spi_gm_if.CHK_CMD : begin
26       if (spi_gm_if.SS_n)
27         ns = spi_gm_if.IDLE;
28       else begin
29         if (~spi_gm_if.MOSI)
30           ns = spi_gm_if.WRITE;
31         else begin
32           if (received_address)
33             //****fixed this bug here, from read address to read data*****
34             ns = spi_gm_if.READ_DATA;
35           else
36             ns = spi_gm_if.READ_ADDRESS;
37           end
38         end
39       end
40     spi_gm_if.WRITE : begin
41       if (spi_gm_if.SS_n)
42         ns = spi_gm_if.IDLE;
43       else
44         ns = spi_gm_if.WRITE;
45     end
46     spi_gm_if.READ_ADDRESS : begin
47       if (spi_gm_if.SS_n)
48         ns = spi_gm_if.IDLE;
49       else
50         ns = spi_gm_if.READ_ADDRESS;
51     end
52     spi_gm_if.READ_DATA : begin
53       if (spi_gm_if.SS_n)
54         ns = spi_gm_if.IDLE;
55       else
56         ns = spi_gm_if.READ_DATA;
57     end
58     //****added default case to avoid latches****
59     default : ns = spi_gm_if.IDLE;
60   endcase
61 end
62
63 always @(posedge spi_gm_if.clk) begin
64   if (~spi_gm_if.rst_n) begin
65     spi_gm_if.rx_data <= 0;
66     spi_gm_if.rx_valid <= 0;
67     received_address <= 0;
68     spi_gm_if.MISO <= 0;
69     //****fixed this bug here, counter should be 0 at reset*****
70     counter <= 0;
71   end
72   else begin
73     case (cs)
74       spi_gm_if.IDLE : begin
75         spi_gm_if.rx_valid <= 0;
76       end
77       spi_gm_if.CHK_CMD : begin
78         counter <= 10;
79       end
80       spi_gm_if.WRITE : begin
81         if (counter > 0) begin
82           | spi_gm_if.rx_data[counter-1] <- spi_gm_if.MOSI;
83           | counter <= counter - 1;
84         end
85         else begin
86           | spi_gm_if.rx_valid <= 1;
87         end
88       end
89       spi_gm_if.READ_ADDRESS : begin
90         if (counter > 0) begin
91           | spi_gm_if.rx_data[counter-1] <- spi_gm_if.MOSI;
92           | counter <= counter - 1;
93         end
94         else begin
95           | spi_gm_if.rx_valid <= 1;
96           | received_address <= 1;
97         end
98       end
99       spi_gm_if.READ_DATA : begin

```

```

99      `endif
100     spi_gm_if.READ_DATA : begin
101       if (spi_gm_if.tx_valid) begin
102         spi_gm_if.rx_valid <= 0;
103         if (counter > 0) begin
104           spi_gm_if.MISO <= spi_gm_if.tx_data[counter-1];
105           counter <= counter - 1;
106         end
107         else begin
108           received_address <= 0;
109         end
110       end
111       else begin
112         if (counter > 0) begin
113           spi_gm_if.rx_data[counter-1] <= spi_gm_if.MOSI;
114           counter <= counter - 1;
115         end
116         else begin
117           spi_gm_if.rx_valid <= 1;
118           counter <= 0;
119         end
120       end
121     end
122   endcase
123 end
124
125
126 endmodule
127

```

➤ Top Code

```

D:\DigitalCourse>DigitalVerification>SYSTEM_2>@ SPItop.sv > ...
1 import uvm_pkg::*;
2 `include "uvm_macros.svh"
3 import spi_test_pkg::*;
4 module TOP_MODULE();
5   bit clk;
6
7   // clock generation
8   initial begin
9     forever
10    #1 clk = ~clk;
11  end
12
13 //instantioation of DUT, TEST and interface
14 SPI_IF if_inst(clk);
15 SPI_GM_IF gm_if_inst(clk);
16
17 SPI_SLAVE_DUT (if_inst);
18 SPI_REF GM_DUT(gm_if_inst);
19
20 // run the testbench using global task run_test
21 initial begin
22
23   // Set the virtual interface for the uvm test
24   uvm_config_db#(virtual SPI_IF)::set(null , "uvm_test_top" , "spi" , if_inst);
25   uvm_config_db#(virtual SPI_GM_IF)::set(null , "uvm_test_top" , "spi_ref" , gm_if_inst);
26   run_test("spi_test");
27 end
28
29 endmodule

```

RAM

➤ Sequence Item package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > RAM_seq_item.sv > ...
1 package ram_seq_item_pkg;
2
3 import uvm_pkg::*;
4 `include "uvm_macros.svh"
5
6 typedef enum bit [1:0] { WRITE_ADDR, WRITE_DATA, READ_ADDR, READ_DATA } cmd_e ;
7 typedef enum bit [1:0] { WRITE_ONLY , READ_ONLY , READ_WRITE} seq_mode_e ;
8
9 class ram_seq_item extends uvm_sequence_item;
10
11   `uvm_object_utils(ram_seq_item)
12
13   rand logic rst_n;
14   rand logic rx_valid;
15   rand cmd_e command;
16   rand logic [7:0] data;
17
18   logic tx_valid ;
19   logic [7:0] dout ;
20
21   logic tx_valid_ref ;
22   logic [7:0] dout_ref ;
23
24   rand seq_mode_e seq_mode ;
25   cmd_e last_command ;
26
27
28   function new(string name = "ram_seq_item");
29     super.new(name);
30   endfunction
31
32
33   constraint c_reset { rst_n dist {0 := 5 , 1 := 95}; }
34   constraint c_rx_valid { rx_valid dist {0 := 5 , 1:= 95}; }
35
36   constraint c_seq {
37     if (seq_mode == WRITE_ONLY)
38     {
39       if (last_command == WRITE_ADDR)
40         command inside {WRITE_ADDR, WRITE_DATA};
41       else
42         command == WRITE_ADDR;
43     }
44     else if (seq_mode == READ_ONLY)
45     {
46       if (last_command == READ_ADDR)
47         command == READ_DATA ;
48       else if (last_command == READ_DATA)
49         command == READ_ADDR;
50     }
51     else if (seq_mode == READ_WRITE)
52     {
53       if (last_command == WRITE_ADDR)
54         command inside {WRITE_ADDR, WRITE_DATA};
55       else if (last_command == WRITE_DATA)
56         command dist {READ_ADDR := 60 , WRITE_ADDR := 40};
57       else if (last_command == READ_ADDR)
58         command inside {READ_ADDR, READ_DATA};
59       else if (last_command == READ_DATA)
60         command dist {WRITE_ADDR := 60 , READ_ADDR := 40};
61     }
62   }
63
64
65   function void post_randomize();
66     last_command = command ;
67   endfunction
68
69   function string convert2string();
70     return $sformatf("%s rx_valid = %0b%0b , command = %0b%0b , data = %0b%0b " ,
71                      super.convert2string() , rx_valid, command, data ) ;
72   endfunction
73
74   function string convert2string_stimulus();
75     return $sformatf(" rx_valid = %0b%0b , command = %0b%0b , data = %0b%0b " ,
76                      rx_valid, command, data ) ;
77   endfunction
78
79 endclass
80
81 endpackage
82
```

➤ Reset Sequence package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > RAM_RST_SEQ.SV ...  
1 package ram_reset_seq_pkg;  
2  
3 import ram_seq_item_pkg::*;  
4 import uvm_pkg::*;  
5 `include "uvm_macros.svh"  
6  
7 class ram_reset_seq extends uvm_sequence #(ram_seq_item);  
8  
9   `uvm_object_utils(ram_reset_seq)  
10  
11   ram_seq_item seq_item ;  
12  
13   function new(string name = "ram_reset_seq");  
14     super.new(name);  
15   endfunction  
16  
17   task body ;  
18  
19     seq_item = ram_seq_item::type_id::create("seq_item");  
20  
21     start_item (seq_item) ;  
22  
23     seq_item.rst_n = 0 ;  
24     seq_item.rx_valid = 0 ;  
25     seq_item.command = cmd_e'(0) ;  
26     seq_item.data = 0 ;  
27  
28     finish_item(seq_item);  
29  
30   endtask  
31  
32 endclass  
33  
34 endpackage  
35 |
```

➤ Write Only Sequence package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > RAM_WRITE_ONLY_SEQ.SV ...  
1 package ram_write_seq_pkg;  
2  
3 import ram_seq_item_pkg::*;  
4 import uvm_pkg::*;  
5 `include "uvm_macros.svh"  
6  
7 class ram_write_only_seq extends uvm_sequence #(ram_seq_item);  
8  
9   `uvm_object_utils(ram_write_only_seq)  
10  
11   ram_seq_item seq_item ;  
12  
13   function new(string name = "ram_write_only_seq");  
14     super.new(name);  
15   endfunction  
16  
17   task body ;  
18  
19     seq_item = ram_seq_item::type_id::create("seq_item");  
20     repeat (5000) begin  
21       start_item(seq_item);  
22  
23       assert(seq_item.randomize() with { seq_mode == WRITE_ONLY }) ;  
24  
25       finish_item(seq_item);  
26  
27     end  
28  
29   endtask  
30  
31 endclass  
32  
33 endpackage  
34  
35 |
```

➤ Read Only Sequence package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > RAM_READ_ONLY_SEQ.SV ...  
1 package ram_read_seq_pkg;  
2  
3 import ram_seq_item_pkg::*;  
4 import uvm_pkg::*;  
5 `include "uvm_macros.svh"  
6  
7 class ram_read_only_seq extends uvm_sequence #(ram_seq_item);  
8  
9   `uvm_object_utils(ram_read_only_seq)  
10  
11   ram_seq_item seq_item ;  
12  
13   function new(string name = "ram_read_only_seq");  
14     super.new(name);  
15   endfunction  
16  
17   task body ;  
18  
19     seq_item = ram_seq_item::type_id::create("seq_item");  
20     repeat (5000) begin  
21       start_item(seq_item);  
22  
23       assert(seq_item.randomize() with { seq_mode == READ_ONLY }) ;  
24  
25       finish_item(seq_item);  
26  
27     end  
28  
29   endtask  
30  
31 endclass  
32  
33 endpackage  
34  
35 |
```

➤ Read/Write Sequence package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > RAM_read_write_seq.sv > ...
1 package ram_read_write_seq_pkg;
2
3 import ram_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class ram_read_write_seq extends uvm_sequence #(ram_seq_item);
8
9   `uvm_object_utils(ram_read_write_seq)
10
11   ram_seq_item seq_item ;
12
13   function new(string name = "ram_read_write_seq");
14     super.new(name);
15   endfunction
16
17   task body;
18
19     seq_item = ram_seq_item::type_id::create("seq_item");
20     repeat (5000) begin
21
22       start_item(seq_item);
23
24       assert(seq_item.randomize() with { seq_mode == READ_WRITE }) ;
25
26       finish_item(seq_item);
27     end
28
29   endtask
30
31 endclass
32
33 endpackage
34 |
```

➤ Sequencer package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > RAM_sequencer.sv > ...
1 package ram_sequencer_pkg ;
2
3 import ram_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class ram_sequencer extends uvm_sequencer #(ram_seq_item) ;
8
9   `uvm_component_utils(ram_sequencer)
10
11   function new(string name = "ram_sequencer" , uvm_component parent = null );
12     super.new(name, parent);
13   endfunction
14
15 endclass
16
17 endpackage
18 |
```

➤ Design Interface

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > RAM_if.sv > ...
1 interface ram_if (clk);
2
3   input clk ;
4   logic rst_n , rx_valid , tx_valid ;
5   logic [9:0] din ;
6   logic [7:0] dout ;
7
8 endinterface : ram_if
9 |
```

➤ Golden Model Interface

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > RAM_GM_if.sv > ...
1 interface ram_GM_if (clk);
2
3   input clk ;
4   logic rst_n , rx_valid , tx_valid ;
5   logic [9:0] din ;
6   logic [7:0] dout ;
7
8 endinterface : ram_GM_if
9 |
```

➤ Driver Package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > @ RAM_driver.sv > ...
1 package ram_driver_pkg;
2
3 import ram_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class ram_driver extends uvm_driver #(ram_seq_item) ;
8
9   `uvm_component_utils(ram_driver)
10
11  virtual ram_if ram_driver_vif;
12  virtual ram_GM_if ram_GM_driver_vif;
13  ram_seq_item stim_seq_item ;
14
15  function new(string name = "ram_driver", uvm_component parent = null);
16    super.new(name, parent);
17  endfunction
18
19  task run_phase (uvm_phase phase);
20    super.run_phase(phase);
21
22    forever begin
23      stim_seq_item = ram_seq_item::type_id::create("stim_seq_item");
24      seq_item_port.get_next_item(stim_seq_item);
25
26      ram_driver_vif.rst_n = stim_seq_item.rst_n ;
27      ram_driver_vif.rx_valid = stim_seq_item.rx_valid ;
28      ram_driver_vif.din[9:8] = stim_seq_item.command ;
29      ram_driver_vif.din[7:0] = stim_seq_item.data ;
30
31      if (ram_GM_driver_vif != null) begin
32        ram_GM_driver_vif.rst_n = stim_seq_item.rst_n ;
33        ram_GM_driver_vif.rx_valid = stim_seq_item.rx_valid ;
34        ram_GM_driver_vif.din[9:8] = stim_seq_item.command ;
35        ram_GM_driver_vif.din[7:0] = stim_seq_item.data ;
36      end
37
38      @(negedge ram_driver_vif.clk );
39
40      seq_item_port.item_done();
41      `uvm_info("run_phase", stim_seq_item.convert2string(), UVM_HIGH)
42    end
43
44  endtask
45
46 endclass
47
48 endpackage
49
```

➤ Monitor Package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > @ RAM_monitor.sv > ...
1 package ram_monitor_pkg;
2
3 import ram_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class ram_monitor extends uvm_monitor;
8
9   `uvm_component_utils(ram_monitor)
10
11  virtual ram_if ram_monitor_vif;
12  virtual ram_GM_if ram_GM_monitor_vif;
13  ram_seq_item rsp_seq_item ;
14
15  uvm_analysis_port #(ram_seq_item) mon_ap ;
16
17  function new(string name = "ram_monitor", 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    mon_ap = new("mon_ap",this);
24  endfunction
25
26  task run_phase (uvm_phase phase);
27    super.run_phase(phase);
28    forever begin
29      rsp_seq_item = ram_seq_item::type_id::create("rsp_seq_item");
30
31      @(negedge ram_monitor_vif.clk );
32      rsp_seq_item.rst_n = ram_monitor_vif.rst_n ;
33      rsp_seq_item.rx_valid = ram_monitor_vif.rx_valid ;
34      rsp_seq_item.command = cmd_e'(ram_monitor_vif.din[9:8]);
35      rsp_seq_item.data = ram_monitor_vif.din[7:0];
36      rsp_seq_item.tx_valid = ram_monitor_vif.tx_valid ;
37      rsp_seq_item.dout = ram_monitor_vif.dout ;
38
39      if (ram_GM_monitor_vif != null) begin
40        rsp_seq_item.item_tx_valid_ref = ram_GM_monitor_vif.item_tx_valid ;
41        rsp_seq_item.dout_ref = ram_GM_monitor_vif.dout ;
42      end
43
44      mon_ap.write(rsp_seq_item);
45      `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH)
46    end
47
48  endtask
49
50 endclass
51
52 endpackage
53
```

➤ Config Package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > RAM_config_pkg.sv ...  
1 package ram_config_pkg;  
2  
3 import uvm_pkg::*;  
4 `include "uvm_macros.svh"  
5  
6 class ram_config extends uvm_object;  
7  
8   `uvm_object_utils(ram_config)  
9  
10  virtual ram_if ram_config_vif;  
11  virtual ram_GM_if ram_GM_config_vif;  
12  
13  uvm_active_passive_enum is_active;  
14  
15  function new(string name = "ram_config");  
16    super.new(name);  
17    is_active = UVM_ACTIVE;  
18  endfunction  
19  
20 endclass  
21  
22 endpackage  
23
```

➤ Agent Package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > RAM_agent.sv ...  
1 package ram_agent_pkg;  
2  
3 import ram_seq_item_pkg::*;  
4 import ram_sequencer_pkg::*;  
5 import ram_driver_pkg::*;  
6 import ram_monitor_pkg::*;  
7 import ram_config_pkg::*;  
8  
9 import uvm_pkg::*;  
10 `include "uvm_macros.svh"  
11  
12 class ram_agent extends uvm_agent ;  
13   `uvm_component_utils(ram_agent)  
14  
15   ram_sequencer sqr ;  
16   ram_driver drv ;  
17   ram_monitor mon ;  
18   ram_config cfg ;  
19  
20   uvm_analysis_port #(ram_seq_item) agt_ap ;  
21  
22   function new(string name = "ram_agent", uvm_component parent = null);  
23     super.new(name, parent);  
24   endfunction  
25  
26   function void build_phase (uvm_phase phase);  
27  
28     super.build_phase(phase) ;  
29     if (!uvm_config_db #(ram_config)::get(this , "" , "CFG_RAM" , cfg )) begin  
30       `uvm_fatal("build_phase", "Unable to get configuration object");  
31     end  
32  
33     if (cfg.is_active == UVM_ACTIVE) begin  
34       sqr = ram_sequencer::type_id::create("sqr", this);  
35       drv = ram_driver::type_id::create("drv", this);  
36     end  
37  
38     mon = ram_monitor::type_id::create("mon",this);  
39     agt_ap = new("agt_ap",this) ;  
40  
41   endfunction  
42  
43   function void connect_phase (uvm_phase phase);  
44     super.connect_phase(phase);  
45     if (cfg.is_active == UVM_ACTIVE) begin  
46       drv.ram_driver_vif = cfg.ram_config_vif;  
47       drv.ram_GM_driver_vif = cfg.ram_GM_config_vif;  
48       drv.seq_item_port.connect(sqr.seq_item_export);  
49     end  
50  
51     mon.ram_monitor_vif = cfg.ram_config_vif;  
52     mon.ram_GM_monitor_vif = cfg.ram_GM_config_vif;  
53     mon.mon_ap.connect(agt_ap) ;  
54  
55   endfunction  
56  
57 endclass  
58  
59 endpackage  
60
```

➤ Scoreboard Package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > @ RAM_scoreboard.sv > ...
1 package ram_scoreboard_pkg;
2
3 import ram_seq_item_pkg::*; 
4 import uvm_pkg::*; 
5 `include "uvm_macros.svh"
6
7 class ram_scoreboard extends uvm_scoreboard;
8   `uvm_component_utils(ram_scoreboard)
9
10  uvm_analysis_export #(ram_seq_item) sb_export;
11  uvm_tlm_analysis_fifo #(ram_seq_item) sb_fifo;
12  ram_seq_item seq_item_sb;
13
14  int error_count = 0;
15  int correct_count = 0;
16
17  function new(string name = "ram_scoreboard", 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    sb_export = new("sb_export", this);
24    sb_fifo = new("sb_fifo", this);
25  endfunction
26
27  function void connect_phase(uvm_phase phase);
28    super.connect_phase(phase);
29    sb_export.connect(sb_fifo.analysis_export);
30  endfunction
31
32  task run_phase(uvm_phase phase);
33    super.run_phase(phase);
34    forever begin
35      seq_item_sb = sb_fifo.get();
36      if ((seq_item_sb.dout != seq_item_sb.dout_ref) || (seq_item_sb.tx_valid != seq_item_sb.tx_valid_ref)) begin
37        `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT: %s While the reference out: 0b%b", 
38          seq_item_sb.convert2string(), seq_item_sb.dout_ref));
39        error_count++;
40      end
41      else begin
42        `uvm_info("run_phase", $sformatf("Correct ram out: %s ", seq_item_sb.convert2string()), UVM_HIGH);
43        correct_count++;
44      end
45    end
46  endtask
47
48  function void report_phase(uvm_phase phase);
49    super.report_phase(phase);
50    `uvm_info("report_phase", $sformatf("Total successful transactions: %d", correct_count), UVM_MEDIUM);
51    `uvm_info("report_phase", $sformatf("Total failed transactions: %d", error_count), UVM_MEDIUM);
52  endfunction
53
54 endclass
55
56 endpackage
57
```

➤ Coverage Package

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > @ RAM.coverage.sv > ...
1 package ram_coverage_pkg;
2
3 import ram_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class ram_coverage extends uvm_component;
8   `uvm_component_utils(ram_coverage)
9   uvm_analysis_export #(ram_seq_item) cov_export;
10  uvm_tlm_analysis_fifo #(ram_seq_item) cov_fifo;
11  ram_seq_item seq_item_cov;
12
13 // coveragegroup declaration
14 coveragegroup cvr_gp;
15
16   command_cp: coverpoint seq_item_cov.command {
17     bins write_address = {WRITE_ADDR};
18     bins write_data = {WRITE_DATA};
19     bins read_address = {READ_ADDR};
20     bins read_data = {READ_DATA};
21     bins write_data_after_address = { WRITE_ADDR => WRITE_DATA };
22     bins read_data_after_address = { READ_ADDR => READ_DATA };
23     bins transition = {WRITE_ADDR => WRITE_DATA => READ_ADDR -> READ_DATA };
24   }
25
26   rx_valid_cp: coverpoint seq_item_cov.rx_valid ;
27
28   tx_valid_cp: coverpoint seq_item_cov.tx_valid ;
29
30
31   rx_valid_with_commands : cross command_cp , rx_valid_cp {
32     ignore_bins rx_valid_zero = binsof(rx_valid_cp) intersect {0} ;
33   }
34
35   tx_valid_with_read_data : cross command_cp , tx_valid_cp {
36     option.cross_auto_bin.max = 0 ;
37     bins tx_valid_rd = binsof(command_cp.read_data) && binsof(tx_valid_cp) intersect {1} ;
38   }
39
40 endgroup
41
42 function new(string name = "ram_coverage", uvm_component parent = null);
43   super.new(name, parent);
44   cvr_gp = new();
45 endfunction
46
47 function void build_phase(uvm_phase phase);
48   super.build_phase(phase);
49   cov_export = new("cov_export", this);
50   cov_fifo = new("cov_fifo", this);
51 endfunction
52
53 function void connect_phase(uvm_phase phase);
54   super.connect_phase(phase);
55   cov_export.connect(cov_fifo.analysis_export);
56 endfunction
57
58 task run_phase(uvm_phase phase);
59   super.run_phase(phase);
60   forever begin
61     cov_fifo.get(seq_item_cov);
62     cvr_gp.sample();
63   end
64 endtask
65
66 endclass
67
68 endpackage
69 |
```

➤ Env Package

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > @ RAM_env.sv > ...
1 package ram_env_pkg;
2
3 import ram_agent_pkg::*;
4 import ram_scoreboard_pkg::*;
5 import ram_coverage_pkg::*;
6 import uvm_pkg::*;
7 `include "uvm_macros.svh"
8
9 class ram_env extends uvm_env;
10   `uvm_component_utils(ram_env)
11
12   ram_agent agt;
13   ram_scoreboard sb;
14   ram_coverage cov;
15
16   function new(string name = "ram_env", uvm_component parent = null);
17     super.new(name, parent);
18   endfunction
19
20   function void build_phase(uvm_phase phase);
21     super.build_phase(phase);
22     agt = ram_agent::type_id::create("agt",this);
23     sb = ram_scoreboard::type_id::create("sb",this);
24     cov = ram_coverage::type_id::create("cov",this);
25   endfunction: build_phase
26
27   function void connect_phase(uvm_phase phase);
28     agt.agt_ap.connect(sb.sb_export);
29     agt.agt_ap.connect(cov.cov_export);
30   endfunction
31
32 endclass
33
34 endpackage
35 |
```

➤ Test Package

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > @ RAM_test.sv > ...
1 package ram_test_pkg;
2
3 import ram_config_pkg::*;
4 import ram_env_pkg::*;
5 import ram_reset_seq_pkg::*;
6 import ram_write_seq_pkg::*;
7 import ram_read_seq_pkg::*;
8 import ram_read_write_seq_pkg::*;
9 import uvm_pkg::*;
10 `include "uvm_macros.svh"
11
12 class ram_test extends uvm_test;
13
14   `uvm_component_utils(ram_test)
15   ram_env env;
16   ram_config ram_cfg;
17   virtual ram_if ram_test_vif;
18   virtual ram_GM_if ram_GM_test_vif;
19
20   ram_reset_seq rst_seq ;
21   ram_write_only_seq write_seq ;
22   ram_read_only_seq read_seq ;
23   ram_read_write_seq read_write_seq ;
24
25   function new(string name = "ram_test", uvm_component parent = null);
26     super.new(name, parent);
27   endfunction
28
29   function void build_phase (uvm_phase phase);
30     super.build_phase(phase);
31
32     env = ram_env::type_id::create("env",this);
33     ram_cfg = ram_config::type_id::create("ram_cfg");
34
35     rst_seq = ram_reset_seq::type_id::create("rst_seq");
36     write_seq = ram_write_only_seq::type_id::create("write_seq");
37     read_seq = ram_read_only_seq::type_id::create("read_seq");
38     read_write_seq = ram_read_write_seq::type_id::create("read_write_seq");
39
40
41     if (!uvm_config_db #(virtual ram_if)::get(this , "", "RAM_IF" , ram_cfg.ram_config_vif )) begin
42       `uvm_fatal("build_phase", "Virtual interface not found ");
43     end
44
45     //FOR GOLDEN MODEL
46     if (!uvm_config_db #(virtual ram_GM_if)::get(this , "", "RAM_GM_IF" , ram_cfg.ram_GM_config_vif )) begin
47       `uvm_fatal("build_phase", "Virtual interface not found ");
48     end
49
50     uvm_config_db#(ram_config)::set(this, "", "CFG_RAM", ram_cfg);
51
52   endfunction
53
54   task run_phase(uvm_phase phase);
55     super.run_phase(phase);
56     phase.raise_objection(this);
57     //reset sequence
58     `uvm_info("run_phase", "Reset Asserted", UVM_LOW)
59     rst_seq.start(env.agt.sqr);
60     `uvm_info("run_phase", "Reset Deasserted", UVM_LOW)
61
62     //main sequence
63     `uvm_info("run_phase", "Stimulus Generation Started", UVM_LOW)
64     write_seq.start(env.agt.sqr);
65     read_seq.start(env.agt.sqr);
66     read_write_seq.start(env.agt.sqr);
67     `uvm_info("run_phase", "Stimulus Generation Ended", UVM_LOW)
68     phase.drop_objection(this);
69   endtask
70
71 endclass: ram_test
72
73 endpackage
74 |
```

➤ Design Code

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > RAM.v ...
1  module RAM (din,clk,rst_n,rx_valid,dout,tx_valid);
2
3  input      [9:0] din;
4  input      clk, rst_n, rx_valid;
5
6  output reg [7:0] dout;
7  output reg tx_valid;
8
9  reg [7:0] mem [255:0]; //mem not MEM///
10
11 reg [7:0] Rd_Addr, Wr_Addr;
12
13 always @(posedge clk) begin
14   if (!rst_n) begin
15     dout <= 0;
16     tx_valid <= 0;
17     Rd_Addr <= 0;
18     Wr_Addr <= 0;
19   end
20   else begin //begin..end//
21     if (rx_valid) begin
22       tx_valid <= 0 ;
23       case (din[9:3])
24         2'b00 : begin
25           Wr_Addr <= din[7:0];
26           tx_valid <= 0 ;
27         end
28         2'b01 : begin
29           mem [Wr_Addr] <= din[7:0];
30           tx_valid <= 0 ;
31         end
32         2'b10 : begin
33           Rd_Addr <= din[7:0];
34           tx_valid <= 0 ;
35         end
36         2'b11 : begin
37           dout <= mem [Rd_Addr]; //*****fixed this bug here, from write address to read address*****//
38           tx_valid <= 1 ; /*put the tx_valid in case statement*/
39         end
40       default : dout<= 0 ;
41     endcase
42   end
43 end
44
45 endmodule
46
47
48 |
```

➤ Golden Model Code

```
D: > DigitalCourse > DigitalVerification > SYSTEM_2 > RAM_GM.v ...
1  module ram_ref ( clk , rst_n , rx_valid , din , tx_valid , dout );
2
3  parameter MEM_DEPTH = 256 ;
4  parameter ADDR_SIZE = 8 ;
5
6  input clk , rst_n , rx_valid ;
7  input [9:0] din ;
8  output reg tx_valid ;
9  output reg [7:0] dout ;
10
11 reg [7:0] mem [MEM_DEPTH-1:0] ;
12
13 reg [ADDR_SIZE-1:] addr_wr , addr_rd ;
14
15 wire [1:8] command = din [9:8] ;
16 wire [7:0] data = din [7:0] ;
17
18
19 always @(`posedge clk ) begin
20   if (!rst_n) begin
21     addr_wr <= 0 ;
22     addr_rd <= 0 ;
23     dout <= 0 ;
24     tx_valid <= 0 ;
25   end
26   else begin
27     if (rx_valid) begin
28       tx_valid <= 0 ;
29       case (command)
30         2'b00: addr_wr <= data ;
31         2'b01: mem[addr_wr] <= data ;
32         2'b10: addr_rd <= data ;
33       default : begin
34         dout <= mem[addr_rd] ;
35         tx_valid <= 1 ;
36       end
37     endcase
38   end
39 end
40
41 endmodule
42
43 |
```

➤ Assertions Code

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > @ RAM_sva.sv > ...
1  module ram_sva( clk , rst_n , rx_valid , din , tx_valid , dout );
2
3
4  input clk , rst_n , rx_valid ;
5  input [9:0] din ;
6  input tx_valid ;
7  input [7:0] dout ;
8
9
10 property reset_outputs_low;
11   @(posedge clk)
12   (rst_n) |>> (tx_valid == 0 && dout == 0);
13 endproperty
14 assert property(reset_outputs_low)
15 | else $error("ERROR : reset operation");
16 cover property(reset_outputs_low);
17
18 property tx_valid_low ;
19   @(posedge clk) disable iff(!rst_n)
20   (rx_valid && (din[9:8]== 2'b00 || din[9:8]== 2'b01 || din[9:8]== 2'b10 ) |>> (tx_valid == 0);
21 endproperty
22 assert property(tx_valid_low)
23 | else $error("ERROR : tx valid during input command phase");
24 cover property(tx_valid_low);
25
26
27 property tx_valid_high ;
28   @(posedge clk) disable iff(!rst_n)
29   (din[9:8] == 2'b11 ) |>> $rose(tx_valid)[-1] ##1 $fell(tx_valid)[-1];
30 endproperty
31 assert property(tx_valid_high)
32 | else $error("ERROR : tx valid after read data");
33 cover property(tx_valid_high);
34
35
36 property write_data_after_address ;
37   @(posedge clk) disable iff(!rst_n)
38   (din[9:8]== 2'b00 ) |>> (din[9:8]== 2'b01 [-1] );
39 endproperty
40 assert property(write_data_after_address)
41 | else $error("ERROR : write data after address");
42 cover property(write_data_after_address);
43
44
45 property read_data_after_address ;
46   @(posedge clk) disable iff(!rst_n)
47   (din[9:8]== 2'b10 ) |>> (din[9:8]== 2'b11 [-1] );
48 endproperty
49 assert property(read_data_after_address)
50 | else $error("ERROR : read data after address");
51 cover property(read_data_after_address);
52
53 endmodule
54
```

➤ Top Code

```
D:\> DigitalCourse > DigitalVerification > SYSTEM_2 > @ RAM_top.sv > ...
1  import uvm_pkg::*;
2  `include "uvm_macros.svh"
3  import ram_test_pkg::*;
4
5  module ram_top();
6
7    bit clk, reset;
8    initial begin
9      forever
10        #1 clk = ~clk;
11    end
12
13 // Instantiate the interface and DUT
14 ram_if ramif (clk);
15 RAM DUT (.din(ramif.din) , .clk(ramif.clk) , .rst_n(ramif.rst_n) , .rx_valid(ramif.rx_valid) , .dout(ramif.dout) , .tx_valid(ramif.tx_valid) );
16
17 ram_GM_if ramGMif (clk);
18 ram_ref REF (ramGMif.clk , ramGMif.rst_n , ramGMif.rx_valid , ramGMif.din , ramGMif.tx_valid , ramGMif.dout);
19
20 //for assertion file
21 bind RAM ram_sva ram_sva_inst (ramif.clk , ramif.rst_n , ramif.rx_valid , ramif.din , ramif.tx_valid , ramif.dout) ;
22
23 // run test using run_test task
24 initial begin
25   // Set the virtual interface for the uvm test
26   uvm_config_db#(virtual ram_if)::set(null , "uvm_test_top" , "RAM_IF" , ramif );
27   uvm_config_db#(virtual ram_GM_if)::set(null , "uvm_test_top" , "RAM_GM_IF" , ramGMif );
28
29   run_test("ram_test");
30 end
31
32 endmodule
33
```

Wrapper

➤ Sequence Item package

```
D:\> DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_seq_item_pkg.sv > () wrapper_seq_item_pkg > wrapper_seq_item

1 package wrapper_seq_item_pkg;
2
3 import uvm_pkg::*;
4
5 include "uvm_macros.svh"
6
7 typedef enum bit [1:0] { WRITE_ONLY , READ_ONLY , READ_WRITE } seq_mode_e;
8
9 typedef enum bit [2:0] { WRITE_ADDR = 3'b000, WRITE_DATA = 3'b001, READ_ADDR = 3'b110, READ_DATA = 3'b111 } cmd_e;
10
11 class wrapper_seq_item extends uvm_sequence_item;
12     `uvm_object_utils(wrapper_seq_item)
13
14     rand logic SS_n;
15     rand logic rst_n;
16     rand cmd_e command;
17     rand seq_mode_e seq_mode;
18     rand bit [MOSI_arr[10:0]];
19     bit [MOSI_arrDrv[10:0]];
20     logic MOSI;
21     logic MISO;
22     cmd_e last_command;
23     logic MISO_ref;
24
25     int read_counter = 0;
26     int normal_counter = 0;
27     int ss_counter = 0;
28     bit read_data_flag = 0;
29
30     function new(string name = "wrapper_seq_item");
31         super.new(name);
32     endfunction
33
34     constraint c_reset { rst_n dist {0 := 2 , 1 := 98}; }
35
36     constraint c_seq {
37         if (seq_mode == WRITE_ONLY)
38         {
39             if (last_command == WRITE_ADDR)
40                 command inside {WRITE_ADDR, WRITE_DATA};
41             else
42                 command == WRITE_ADDR;
43         }
44         else if (seq_mode == READ_ONLY)
45         {
46             if (last_command == READ_ADDR)
47                 command == READ_DATA ;
48             else if (last_command == READ_DATA)
49                 command == READ_ADDR;
50         }
51         else if (seq_mode == READ_WRITE)
52         {
53             if (last_command == WRITE_ADDR)
54                 command inside {WRITE_ADDR, WRITE_DATA};
55             else if (last_command == WRITE_DATA)
56                 command dist (READ_ADDR := 60 , WRITE_ADDR := 40);
57             else if (last_command == READ_ADDR)
58                 command inside {READ_ADDR, READ_DATA};
59             else if (last_command == READ_DATA)
60                 command dist (WRITE_ADDR := 60 , READ_ADDR := 40);
61         }
62     }
63
64     function void post_randomize();
65         last_command = command;
66         {MOSI_arr[0], MOSI_arr[1], MOSI_arr[2]} = command;
67
68         if (ss_counter == 0)
69             SS_n = 1;
70         else
71             SS_n = 0;
72
73         if ((normal_counter == 0) && (read_counter == 0) && (ss_counter == 0)) begin
74             MOSI_arr_drv = MOSI_arr;
75         end
76
77         read_data_flag = (((MOSI_arr_drv[0], MOSI_arrDrv[1], MOSI_arrDrv[2])) == READ_DATA);
78
79         if (read_data_flag) begin
80             if (ss_counter < 23)
81                 ss_counter++;
82             else
83                 ss_counter = 0;
84         end
85         else begin
86             if (ss_counter < 13)
87                 ss_counter++;
88             else
89                 ss_counter = 0;
90         end
91
92         if (read_counter < 25 && !SS_n && read_data_flag && ss_counter > 2) begin
93             if (read_counter < 11)
94                 MOSI = MOSI_arr_drv[read_counter++];
95             else
96                 read_counter++;
97         end
98         else begin
99             read_counter = 0;
100            read_data_flag = 0;
101        end
102
103        if (normal_counter < 15 && !SS_n && !read_data_flag && ss_counter > 2) begin
104            if (normal_counter < 11)
105                MOSI = MOSI_arr_drv[normal_counter++];
106            else
107                normal_counter++;
108        end
109        else
110            normal_counter = 0;
111
112    endfunction
113
114
```

```

115
116     function string convert2string();
117         return $sformatf(
118             "%s reset = %0b, SS_n = %0b, MOSI = %0b, MISO = %0b, command = %0b%0b , MISO_ref = %0b",
119             super.convert2string(),
120             rst_n, SS_n, MOSI, MISO, command, MISO_ref
121         );
122     endfunction
123
124
125     function string convert2string_stimulus();
126         return $sformatf(
127             "%s reset = %0b, SS_n = %0b, MOSI = %0b, command = %0b, MISO_ref = %0b",
128             rst_n, SS_n, MOSI, command, MISO_ref
129         );
130     endfunction
131
132 endclass
133
134 endpackage

```

➤ Reset Sequence package

```

D:\> DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_reset_seq.sv > ...
1 package wrapper_reset_seq_pkg;
2
3 import wrapper_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class wrapper_reset_seq extends uvm_sequence #(wrapper_seq_item);
8     `uvm_object_utils(wrapper_reset_seq)
9
10    wrapper_seq_item seq_item;
11
12    function new(string name = "wrapper_reset_seq");
13        super.new(name);
14    endfunction
15
16    task body;
17        seq_item = wrapper_seq_item::type_id::create("seq_item");
18        start_item (seq_item);
19        seq_item.rst_n = 0;
20        seq_item.SS_n = 0;
21        seq_item.MOSI = 0;
22        seq_item.MISO = 0;
23        finish_item(seq_item);
24    endtask
25 endclass
26
27 endpackage

```

➤ Write Only Sequence package

```

D:\> DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_write_only_seq.sv > ...
1 package wrapper_write_only_seq_pkg;
2
3 import wrapper_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class wrapper_write_only_seq extends uvm_sequence #(wrapper_seq_item);
8     `uvm_object_utils(wrapper_write_only_seq)
9
10    wrapper_seq_item seq_item;
11
12    function new(string name = "wrapper_write_only_seq");
13        super.new(name);
14    endfunction
15
16    task body;
17        seq_item = wrapper_seq_item::type_id::create("seq_item");
18        repeat (5000) begin
19            start_item(seq_item);
20            assert(seq_item.randomize() with { seq_mode == WRITE_ONLY; });
21            finish_item(seq_item);
22        end
23    endtask
24 endclass
25
26 endpackage

```

➤ Read Only Sequence package

```

D:\> DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_read_only_seq.sv > ...
1 package wrapper_read_only_seq_pkg;
2
3 import wrapper_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class wrapper_read_only_seq extends uvm_sequence #(wrapper_seq_item);
8     `uvm_object_utils(wrapper_read_only_seq)
9
10    wrapper_seq_item seq_item;
11
12    function new(string name = "wrapper_read_only_seq");
13        super.new(name);
14    endfunction
15
16    task body;
17        seq_item = wrapper_seq_item::type_id::create("seq_item");
18        repeat (5000) begin
19            start_item(seq_item);
20            assert(seq_item.randomize() with { seq_mode == READ_ONLY; });
21            finish_item(seq_item);
22        end
23    endtask
24 endclass
25
26 endpackage

```

➤ Read/Write Sequence package

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_read_write_seq_pkg.sv > ...
1  package wrapper_read_write_seq_pkg;
2
3  import wrapper_seq_item_pkg::*;
4  import uvm_pkg::*;
5  `include "uvm_macros.svh"
6
7  class wrapper_read_write_seq extends uvm_sequence #(wrapper_seq_item);
8  `uvm_object_utils(wrapper_read_write_seq)
9
10    wrapper_seq_item seq_item;
11
12    function new(string name = "wrapper_read_write_seq");
13      super.new(name);
14    endfunction
15
16    task body;
17      seq_item = wrapper_seq_item::type_id::create("seq_item");
18      repeat (5000) begin
19        start_item(seq_item);
20        assert(seq_item.randomize() with { seq_mode == READ_WRITE; });
21        finish_item(seq_item);
22      end
23    endtask
24  endclass
25
26 endpackage
```

➤ Sequencer package

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_sequencer.sv > ...
1  package wrapper_sequencer_pkg ;
2
3  import wrapper_seq_item_pkg::*;
4  import uvm_pkg::*;
5  `include "uvm_macros.svh"
6
7  class wrapper_sequencer extends uvm_sequencer #(wrapper_seq_item) ;
8  `uvm_component_utils(wrapper_sequencer)
9
10    function new(string name = "wrapper_sequencer" , uvm_component parent = null);
11      super.new(name, parent);
12    endfunction
13
14  endclass
15
16 endpackage
```

➤ Design Interface

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_if.sv > ...
1  interface wrapper_if(clk);
2
3    input clk;
4    logic rst_n;
5    logic MOSI;
6    logic MISO;
7    logic SS_n;
8
9  endinterface
10
```

➤ Golden Model Interface

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_ref_if.sv > ...
1  interface wrapper_ref_if(clk);
2
3    input clk;
4    logic rst_n;
5    logic MOSI_ref;
6    logic MISO_ref;
7    logic SS_n_ref;
8
9  endinterface
10
```

➤ Driver Package

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_driver.sv > ...
1 package wrapper_driver_pkg;
2
3 import wrapper_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class wrapper_driver extends uvm_driver #(wrapper_seq_item);
8   `uvm_component_utils(wrapper_driver)
9
10  virtual wrapper_if wrapper_driver_vif;
11  virtual wrapper_ref_if wrapper_ref_driver_vif;
12  wrapper_seq_item stim_seq_item;
13
14  function new(string name = "wrapper_driver", uvm_component parent = null);
15    super.new(name, parent);
16  endfunction
17
18  task run_phase (uvm_phase phase);
19    super.run_phase(phase);
20
21    forever begin
22      stim_seq_item = wrapper_seq_item::type_id::create("stim_seq_item");
23      seq_item_port.get_next_item(stim_seq_item);
24
25      wrapper_driver_vif.rst_n = stim_seq_item.rst_n;
26      wrapper_driver_vif.ss_n = stim_seq_item.ss_n;
27      wrapper_driver_vif.MOSI = stim_seq_item.MOSI;
28
29      if (wrapper_ref_driver_vif != null) begin
30        wrapper_ref_driver_vif.rst_n = stim_seq_item.rst_n;
31        wrapper_ref_driver_vif.ss_n_ref = stim_seq_item.ss_n;
32        wrapper_ref_driver_vif.MOSI_ref = stim_seq_item.MOSI;
33      end
34      @(negedge wrapper_driver_vif.clk);
35      seq_item_port.item_done();
36      `uvm_info("run_phase", stim_seq_item.convert2string_stimulus() , UVM_HIGH)
37    end
38  endtask
39 endclass
40
41 endpackage
42
```

➤ Monitor Package

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_monitor.sv > ...
1 package wrapper_monitor_pkg;
2
3 import wrapper_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class wrapper_monitor extends uvm_monitor;
8   `uvm_component_utils(wrapper_monitor)
9
10  virtual wrapper_if wrapper_monitor_vif;
11  virtual wrapper_ref_if wrapper_ref_monitor_vif;
12  wrapper_seq_item seq_item;
13
14  uvm_analysis_port #(wrapper_seq_item) mon_ap;
15
16  function new(string name = "wrapper_monitor", 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    mon_ap = new("mon_ap",this);
23  endfunction
24
25  task run_phase (uvm_phase phase);
26    super.run_phase(phase);
27    forever begin
28      seq_item = wrapper_seq_item::type_id::create("seq_item");
29
30      @(negedge wrapper_monitor_vif.clk);
31      seq_item.rst_n = wrapper_monitor_vif.rst_n;
32      seq_item.ss_n = wrapper_monitor_vif.ss_n;
33      seq_item.MOSI = wrapper_monitor_vif.MOSI;
34      seq_item.MISO = wrapper_monitor_vif.MISO;
35
36      if (wrapper_ref_monitor_vif != null) begin
37        seq_item.MISO_ref = wrapper_ref_monitor_vif.MISO_ref;
38      end
39
40      mon_ap.write(seq_item);
41      `uvm_info("run_phase", seq_item.convert2string() , UVM_HIGH)
42    end
43  endtask
44 endclass
45 endpackage
46
```

➤ Config Package

```
D:\> DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_config_obj.sv > ...
1 package wrapper_config_pkg;
2
3 import uvm_pkg::*;
4 `include "uvm_macros.svh"
5
6 class wrapper_config extends uvm_object;
7 `uvm_object_utils(wrapper_config)
8
9 virtual wrapper_if wrapper_config_vif;
10 virtual wrapper_ref_if wrapper_ref_config_vif;
11
12 uvm_active_passive_enum is_active;
13
14 function new(string name = "wrapper_config");
15     super.new(name);
16     is_active = UVM_ACTIVE;
17 endfunction
18
19 endclass
20
21 endpackage
```

➤ Agent Package

```
D:\> DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_agent.sv > ...
1 package wrapper_agent_pkg;
2
3 import wrapper_seq_item_pkg::*;
4 import wrapper_sequencer_pkg::*;
5 import wrapper_driver_pkg::*;
6 import wrapper_monitor_pkg::*;
7 import wrapper_config_pkg::*;
8 import uvm_pkg::*;
9 `include "uvm_macros.svh"
10
11 class wrapper_agent extends uvm_agent;
12 `uvm_component_utils(wrapper_agent)
13
14 wrapper_sequencer sqr;
15 wrapper_driver drv;
16 wrapper_monitor mon;
17 wrapper_config cfg;
18
19 uvm_analysis_port #(wrapper_seq_item) agt_ap;
20
21 function new(string name = "wrapper_agent", uvm_component parent = null);
22     super.new(name, parent);
23 endfunction
24
25 function void build_phase (uvm_phase phase);
26     super.build_phase(phase);
27     if (!uvm_config_db #(wrapper_config)::get(this , "", "CFG" , cfg )) begin
28         `uvm_fatal("build_phase", "Unable to get configuration object");
29     end
30     if (cfg.is_active == UVM_ACTIVE) begin
31         sqr = wrapper_sequencer::type_id::create("sqr", this);
32         drv = wrapper_driver::type_id::create("drv", this);
33     end
34     mon = wrapper_monitor::type_id::create("mon",this);
35     agt_ap = new("agt_ap",this);
36 endfunction
37
38 function void connect_phase (uvm_phase phase);
39     super.connect_phase(phase);
40     if (cfg.is_active == UVM_ACTIVE) begin
41         drv.wrapper_driver_vif = cfg.wrapper_config_vif;
42         drv.wrapper_ref_driver_vif = cfg.wrapper_ref_config_vif;
43         drv.seq_item_port.connect(sqr.seq_item_export);
44     end
45     mon.wrapper_monitor_vif = cfg.wrapper_config_vif;
46     mon.wrapper_ref_monitor_vif = cfg.wrapper_ref_config_vif;
47     mon.mon_ap.connect(agt_ap);
48 endfunction
49
50 endclass
51
52 endpackage
```

➤ Scoreboard Package

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_scoreboard.sv > ...
1 package wrapper_scoreboard_pkg;
2
3 import wrapper_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class wrapper_scoreboard extends uvm_scoreboard;
8   `UVM_COMPONENT_UTILS(wrapper_scoreboard)
9
10  uvm_analysis_export #(wrapper_seq_item) sb_export;
11  uvm_tlm_analysis_fifo #(wrapper_seq_item) sb_fifo;
12  wrapper_seq_item seq_item_sb;
13
14  int error_count = 0;
15  int correct_count = 0;
16
17  function new(string name = "wrapper_scoreboard", 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    sb_export = new("sb_export", this);
24    sb_fifo = new("sb_fifo", this);
25  endfunction
26
27
28  function void connect_phase(uvm_phase phase);
29    super.connect_phase(phase);
30    sb_export.connect(sb_fifo.analysis_export);
31  endfunction
32
33
34  task run_phase(uvm_phase phase);
35    super.run_phase(phase);
36    forever begin
37      sb_fifo.get(seq_item_sb);
38      if ((seq_item_sb.MISO_ref != seq_item_sb.MISO)) begin
39        `uvm_error("run_phase", $sformatf("Comparison failed, MISO received by the DUT:%0b While the reference MISO: %0b", seq_item_sb.MISO, seq_item_sb.MISO_ref));
40        error_count++;
41      end
42      else begin
43        `uvm_info("run_phase", $sformatf("Correct MISO: %0b == MISO_Ref %0b", seq_item_sb.MISO, seq_item_sb.MISO_ref), UVM_HIGH);
44        correct_count++;
45      end
46    end
47  endtask
48
49
50  function void report_phase(uvm_phase phase);
51    super.report_phase(phase);
52    `uvm_info("report_phase", $sformatf("Total successful transactions: %0d", correct_count), UVM_MEDIUM);
53    `uvm_info("report_phase", $sformatf("Total failed transactions: %0d", error_count), UVM_MEDIUM);
54  endfunction
55
56 endclass
57
58 endpackage
```

➤ Coverage Package

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_coverage.sv > ...
1 package wrapper_coverage_pkg;
2
3 import wrapper_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class wrapper_coverage extends uvm_component;
8   `UVM_COMPONENT_UTILS(wrapper_coverage)
9
10  uvm_analysis_export #(wrapper_seq_item) cov_export;
11  uvm_tlm_analysis_fifo #(wrapper_seq_item) cov_fifo;
12  wrapper_seq_item seq_item_cov;
13
14  function new(string name = "wrapper_coverage", uvm_component parent = null);
15    super.new(name, parent);
16  endfunction
17
18  function void build_phase(uvm_phase phase);
19    super.build_phase(phase);
20    cov_export = new("cov_export", this);
21    cov_fifo = new("cov_fifo", this);
22  endfunction
23
24
25  function void connect_phase(uvm_phase phase);
26    super.connect_phase(phase);
27    cov_export.connect(cov_fifo.analysis_export);
28  endfunction
29
30
31  task run_phase(uvm_phase phase);
32    super.run_phase(phase);
33    forever begin
34      cov_fifo.get(seq_item_cov);
35    end
36  endtask
37
38 endclass
39
40 endpackage
```

➤ Env Package

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_env.sv > ...
1 package wrapper_env_pkg;
2
3 import wrapper_agent_pkg::*;
4 import wrapper_scoreboard_pkg::*;
5 import wrapper_coverage_pkg::*;
6 import uvm_pkg::*;
7 `include "uvm_macros.svh"
8
9
10 class wrapper_env extends uvm_env;
11   `uvm_component_utils(wrapper_env)
12
13   wrapper_agent      agt;
14   wrapper_scoreboard sb;
15   wrapper_coverage   cov;
16
17   function new(string name = "wrapper_env", uvm_component parent = null);
18     super.new(name, parent);
19   endfunction
20
21
22   function void build_phase(uvm_phase phase);
23     super.build_phase(phase);
24     agt = wrapper_agent::type_id::create("agt",this);
25     sb = wrapper_scoreboard::type_id::create("sb",this);
26     cov = wrapper_coverage::type_id::create("cov",this);
27   endfunction
28
29
30   function void connect_phase(uvm_phase phase);
31     super.connect_phase(phase);
32     agt.agt_ap.connect(sb.sb_export);
33     agt.agt_ap.connect(cov.cov_export);
34   endfunction
35
36
37   function void report_phase(uvm_phase phase);
38     super.report_phase(phase);
39
40     `uvm_info("WRAPPER_ENV", "Wrapper Environment completed successfully", UVM_LOW)
41   endfunction
42 endclass
43
44 endpackage
```

➤ Test Package

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_test.sv > ...
1 package wrapper_test_pkg;
2
3
4 import wrapper_config_pkg::*;
5 import wrapper_env_pkg::*;
6 import wrapper_reset_seq_pkg::*;
7 import wrapper_write_seq_pkg::*;
8 import wrapper_read_seq_pkg::*;
9 import wrapper_read_write_seq_pkg::*;
10 import spi_config_pkg::*;
11 import ram_config_pkg::*;
12 import spi_env_pkg::*;
13 import ram_env_pkg::*;
14 import uvm_pkg::*;
15 `include "uvm_macros.svh"
16
17
18 class wrapper_test extends uvm_test;
19   `uvm_component_utils(wrapper_test)
20
21   wrapper_env      env;
22   wrapper_config   wrapper_cfg;
23   virtual wrapper_if  wrapper_test_vif;
24   virtual wrapper_ref_if wrapper_ref_test_vif;
25   wrapper_reset_seq   rst_seq ;
26   wrapper_write_only_seq write_seq ;
27   wrapper_read_only_seq read_seq ;
28   wrapper_read_write_seq read_write_seq ;
29
30   ram_config        ram_conf;
31   spi_config        spi_conf;
32   spi_env           s_env;
33   ram_env           r_env;
34
35
36   function new(string name = "wrapper_test", uvm_component parent = null);
37     super.new(name, parent);
38   endfunction
39
```

```

10
11     function void build_phase(uvm_phase phase);
12         super.build_phase(phase);
13
14         env          = wrapper_env::type_id::create("env",this);
15         r_env        = ram_env::type_id::create("r_env",this);
16         s_env        = spi_env::type_id::create("s_env",this);
17         wrapper_cfg  = wrapper_config::type_id::create("wrapper_cfg");
18
19         rst_seq      = wrapper_reset_seq::type_id:: create("rst_seq");
20         write_seq    = wrapper_write_only_seq::type_id::create("write_seq");
21         read_seq     = wrapper_read_only_seq::type_id:: create("read_seq");
22         read_write_seq = wrapper_read_write_seq::type_id::create("read_write_seq");
23
24         ram_conf     = ram_config::type_id::create("ram_conf");
25         spi_conf     = spi_config::type_id::create("spi_conf");
26
27         spi_conf.is_active   = UVM_PASSIVE;
28         ram_conf.is_active   = UVM_PASSIVE;
29         wrapper_cfg.is_active = UVM_ACTIVE;
30
31
32         if (!uvm_config_db #(virtual wrapper_if)::get(this , "" , "wifv" , wrapper_cfg.wrapper_config_vif )) begin
33             `uvm_fatal("build_phase", "Virtual interface not found ");
34         end
35
36         if (!uvm_config_db #(virtual wrapper_ref_if)::get(this , "" , "wrifv" , wrapper_cfg.wrapper_ref_config_vif )) begin
37             `uvm_fatal("build_phase", "Virtual interface not found ");
38         end
39
40         uvm_config_db #(wrapper_config)::set(this, "", "CFG", wrapper_cfg);
41
42         if (!uvm_config_db #(virtual ram_if)::get(this , "" , "RAM_IF" , ram_conf.ram_config_vif )) begin
43             `uvm_fatal("build_phase", "Virtual interface not found ");
44         end
45
46
47         if (!uvm_config_db #(virtual ram_GM_if)::get(this , "" , "RAM_GM_IF" , ram_conf.ram_GM_config_vif )) begin
48             `uvm_fatal("build_phase", "Virtual interface not found ");
49         end
50
51         uvm_config_db#(ram_config)::set(this, "", "CFG_RAM", ram_conf);
52
53         if (!uvm_config_db #(virtual SPI_IF)::get(this , "" , "spi" , spi_conf.SPI_vif)) begin
54             `uvm_fatal("build_phase", "error in getting the data");
55         end
56         if (!uvm_config_db #(virtual SPI_GM_IF)::get(this , "" , "spi_ref" , spi_conf.SPI_GM_vif)) begin
57             `uvm_fatal("build_phase", "error in getting the data");
58         end
59         uvm_config_db #(spi_config)::set(this , "*" , "CFG_SPI" , spi_conf);
60     endfunction
61
62
63     task run_phase(uvm_phase phase);
64         super.run_phase(phase);
65         phase.raise_objection(this);
66
67         `uvm_info("wrapper_test", "Starting all sequences in parallel", UVM_LOW)
68
69         //reset sequence
70         `uvm_info("run_phase", "Reset Asserted", UVM_LOW)
71         rst_seq.start(env.agt.sqr);
72         `uvm_info("run_phase", "Reset Deasserted", UVM_LOW)
73
74         //main sequence
75         `uvm_info("run_phase", "Stimulus Generation Started", UVM_LOW)
76         write_seq.start(env.agt.sqr);
77         read_seq.start(env.agt.sqr);
78         read_write_seq.start(env.agt.sqr);
79         `uvm_info("run_phase", "Stimulus Generation Ended", UVM_LOW)
80         `uvm_info("wrapper_test", "All sequences completed successfully", UVM_LOW)
81
82         phase.drop_objection(this);
83     endtask
84
85     endclass
86
87     endpackage
88
89

```

➤ Design Code

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper.sv > ...
1  module WRAPPER (clk, rst_n, MOSI, MISO, SS_n);
2
3
4  input bit clk;
5  input logic rst_n;
6  input logic MOSI;
7  output logic MISO;
8  input logic SS_n;
9
10 logic tx_valid;
11 logic rx_valid;
12 logic [9:0] rx_data;
13 logic [7:0] tx_data;
14
15
16 SPI_SLAVE      SLAVE_instance (
17   .clk          (clk),
18   .MOSI         (MOSI),
19   .MISO         (MISO),
20   .SS_n         (SS_n),
21   .rst_n        (rst_n),
22   .rx_data      (rx_data),
23   .rx_valid     (rx_valid),
24   .tx_data      (tx_data),
25   .tx_valid     (tx_valid)
26 );
27
28
29 RAM           RAM_instance (
30   .clk          (clk),
31   .rst_n        (rst_n),
32   .tx_valid     (tx_valid),
33   .dout         (tx_data),
34   .din          (rx_data),
35   .rx_valid     (rx_valid)
36 );
37
38 `ifndef SIM
39
40 property reset_output_low;
41   @(posedge clk)
42     (!rst_n) |> (MISO== 0);
43 endproperty
44 assert property(reset_output_low)
45   || else $error("ERROR : reset operation");
46 cover property(reset_output_low);
47
48
49 property MISO_stable;
50   @(posedge clk) disable iff (!rst_n)
51     (!tx_valid) |=> $stable(MISO)[-1];
52 endproperty
53 assert property(MISO_stable)
54   || else $error("ERROR : MISO stability");
55 cover property(MISO_stable);
56
57
58
59
60 `endif
61
62 endmodule
63
```

➤ Golden Model Code

```
D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_ref.sv > ...
1  module wrapper_ref (clk, rst_n_ref,MOSI_ref, MISO_ref, SS_n_ref);
2
3
4  input bit clk;
5  input logic rst_n_ref;
6  input logic MOSI_ref;
7  output logic MISO_ref;
8  input logic SS_n_ref;
9
10
11 wire tx_valid_ref;
12 logic rx_valid_ref;
13 logic [9:0] rx_data_ref;
14 logic [7:0] tx_data_ref;
15
16 SPI_REF      SLAVE_ref_instance (
17   .clk          (clk),
18   .MOSI         (MOSI_ref),
19   .MISO         (MISO_ref),
20   .SS_n         (SS_n_ref),
21   .rst_n        (rst_n_ref),
22   .tx_data      (tx_data_ref),
23   .tx_valid     (tx_valid_ref),
24   .rx_data      (rx_data_ref),
25   .rx_valid     (rx_valid_ref)
26 );
27
28
29 ram_ref      RAM_ref_instance  (
30   .clk          (clk),
31   .rst_n        (rst_n_ref),
32   .din          (rx_data_ref),
33   .tx_valid     (tx_valid_ref),
34   .dout         (tx_data_ref),
35   .rx_valid     (rx_valid_ref)
36 );
37
38 endmodule
39
```

➤ Top Code

```

D: > DigitalCourse > DigitalVerification > UVM_Project > Wrapper > wrapper_top.sv > wrapper_top > if_inst

1 import uvm_pkg::*;
2 `include "ovm_macros.svh"
3 import wrapper_test_pkg::*;

4
5 module wrapper_top(clk);
6
7   input bit clk;
8   initial begin
9     forever #1 clk = ~clk;
10    end
11
12
13   wrapper_if          wif      (clk);
14   wrapper_ref_if      wrif     (clk);
15   ram_GM_if           ramGIf   (clk);
16   ram_if              ramIf    (clk);
17   SPI_IF              if_inst  (clk);
18   SPI_GM_IF           gm_if_inst (clk);
19
20   WRAPPER             dut (
21     .clk               (clk),
22     .MISO              (wif.MISO),
23     .rst_n             (wif.rst_n),
24     .SS_n              (wif.SS_n),
25     .MOST              (wif.MOST)
26   );
27
28   wrapper_ref          dut_ref (
29     .clk               (clk),
30     .MISO_ref          (wrif.MISO_ref),
31     .rst_n_ref          (wrif.rst_n),
32     .SS_n_ref          (wrif.SS_n_ref),
33     .MOST_ref          (wrif.MOST_ref)
34   );
35
36
37 /*bind WRAPPER wrapper_sva U_wrapper_sva (
38   .clk,
39   .rst_n          (wif.rst_n),
40   .MISO           (wif.MISO),
41   .rx_data        (wif.rx_data),
42   .MOSI           (wif.MOSI),
43   .SS_n           (wif.SS_n)
44 );
45
46 bind SPI_SLAVE SPI_ASSERTION assert_inst (
47   .clk            (if_inst.clk),
48   .rst_n          (if_inst.rst_n),
49   .MOSI           (if_inst.MOSI),
50   .MISO           (if_inst.MISO),
51   .rx_valid       (if_inst.rx_valid),
52   .rx_data        (if_inst.rx_data),
53   .SS_n           (if_inst.SS_n)
54 );
55 */
56 //already in spi design cause they depend on internal signals
57
58 bind RAM ram_sva          ram_sva_inst (
59   .clk            (dut.clk),
60   .rst_n          (dut.rst_n),
61   .rx_valid       (dut.rx_valid),
62   .din            (dut.din),
63   .tx_valid       (dut.tx_valid),
64   .dout           (dut.dout)
65 );
66
67 assign if_inst.rst_n          = dut.rst_n;
68 assign ramIf.rst_n          = dut.rst_n;
69 assign gm_if_inst.rst_n      = dut.ref.rst_n_ref;
70 assign ramGIf.rst_n          = dut.ref.rst_n_ref;
71
72 assign if_inst.SS_n          = dut.SS_n;
73 assign if_inst.MOSI          = dut.MOSI;
74 assign gm_if_inst.MOSI       = dut.ref.MOSI_ref;
75 assign gm_if_inst.SS_n_ref   = dut.ref.SS_n_ref;
76 assign if_inst.rx_data       = dut.rx_data;
77
78 assign if_inst.tx_valid      = dut.tx_valid;
79 assign if_inst.tx_data       = dut.tx_data;
80 assign gm_if_inst.tx_valid   = dut.tx_valid;
81 assign gm_if_inst.tx_data   = dut.tx_data;
82 assign ramIf.din             = dut.rx_valid;
83 assign ramIf.tx_valid       = dut.tx_valid;
84 assign ramGIf.din             = dut.rx_data;
85 assign ramGIf.tx_valid      = dut.tx_valid;
86
87
88 initial begin
89   uvm_config_db#(virtual wrapper_ref_if)::set(null, "uvm_test_top", "wrifv", wrif);
90   uvm_config_db#(virtual wrapper_if)::set(null, "uvm_test_top", "wifv", wif);
91   uvm_config_db#(virtual ram_if)::set(null, "uvm_test_top", "RAM_IF", ramIf);
92   uvm_config_db#(virtual ram_GM_if)::set(null, "uvm_test_top", "RAM_GM_IF", ramGIf);
93   uvm_config_db#(virtual SPI_IF)::set(null, "uvm_test_top", "spi", if_inst);
94   uvm_config_db#(virtual SPI_GM_IF)::set(null, "uvm_test_top", "spi_ref", gm_if_inst);
95   run_test("wrapper_test");
96
97 end
98 endmodule

```

Code Coverage Report

SPI Slave

```
=====
== Instance: /TOP_MODULE/DUT
== Design Unit: work.SPI_SLAVE
=====

Assertion Coverage:
 Assertions 14 14 0 100.00%
 -----
Name File(Line) Failure Count Pass Count
-----
/TOP_MODULE/DUT/assert_p_ss_N_normal SPI.sv(264) 0 1
/TOP_MODULE/DUT/assert_p_ss_N_rd_data SPI.sv(261) 0 1
/TOP_MODULE/DUT/assert_p_rx_valid_rd_data SPI.sv(258) 0 1
/TOP_MODULE/DUT/assert_p_rx_valid_rd_addr SPI.sv(255) 0 1
/TOP_MODULE/DUT/assert_p_rx_valid_wr_data SPI.sv(252) 0 1
/TOP_MODULE/DUT/assert_p_rx_valid_wr_addr SPI.sv(249) 0 1
/TOP_MODULE/DUT/assert_p_readdata_to_IDLE SPI.sv(246) 0 1
/TOP_MODULE/DUT/assert_p_readadd_to_IDLE SPI.sv(243) 0 1
/TOP_MODULE/DUT/assert_p_WRITE_to_IDLE SPI.sv(240) 0 1
/TOP_MODULE/DUT/assert_p_chk_to_READ_DATA SPI.sv(237) 0 1
/TOP_MODULE/DUT/assert_p_chk_to_READ_ADDRESS SPI.sv(234) 0 1
/TOP_MODULE/DUT/assert_p_chk_to_WRITE SPI.sv(231) 0 1
/TOP_MODULE/DUT/assert_p_IDLE_to_chk SPI.sv(228) 0 1
/TOP_MODULE/DUT/assert_p_rst SPI.sv(225) 0 1
-----
Branch Coverage:
 Enabled Coverage Bins Hits Misses Coverage
 -----
Branches 39 39 0 100.00%
=====Branch Details=====
Branch Coverage for instance /TOP_MODULE/DUT
Line Item Count Source
-----
File SPI.sv
-----IF Branch-----
10 3033 Count coming in to IF
10 1 208 if (~spi_if.rst_n) begin
13 1 2825 else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----CASE Branch-----
20 6899 Count coming in to CASE
21 1 1489 spi_if.IDLE: begin
28 1 1121 spi_if.CHK_CMD: begin
44 1 1843 spi_if.WRITE: begin
51 1 1346 spi_if.READ_ADDRESS: begin
58 1 1009 spi_if.READ_DATA: begin
65 1 1 default: ns = spi_if.IDLE;
Branch totals: 6 hits of 6 branches = 100.00%
-----IF Branch-----
22 1489 Count coming in to IF
22 1 668 if (spi_if.SS_n)
24 1 881 else
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
29 1121 Count coming in to IF
29 1 9 if (spi_if.SS_n)
31 1 1112 else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
32 1112 Count coming in to IF
32 1 534 if (~spi_if.MOSI)
34 1 578 else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
35 578 Count coming in to IF

```

```

Condition Coverage:
  Enabled Coverage      Bins   Covered   Misses   Coverage
  -----      ---      ---      ---      -----
  Conditions          4       4        0    100.00%
=====
=====Condition Details=====
Condition Coverage for instance /TOP_MODULE/DUT --
File SPI.sv
-----Focused Condition View-----
Line    90 Item   1 (counter > 0)
Condition totals: 1 of 1 input term covered = 100.00%
  Input Term   Covered   Reason for no coverage   Hint
  -----      ---      -----
  (counter > 0)      Y
  Rows:   Hits   FEC Target      Non-masking condition(s)
  -----      ---      -----
  Row  1:      1 (counter > 0)_0      -
  Row  2:      1 (counter > 0)_1      -
-----Focused Condition View-----
Line   100 Item   1 (counter > 0)
Condition totals: 1 of 1 input term covered = 100.00%
  Input Term   Covered   Reason for no coverage   Hint
  -----      ---      -----
  (counter > 0)      Y
  Rows:   Hits   FEC Target      Non-masking condition(s)
  -----      ---      -----
  Row  1:      1 (counter > 0)_0      -
  Row  2:      1 (counter > 0)_1      -
-----Focused Condition View-----
Line   113 Item   1 (counter > 0)
Condition totals: 1 of 1 input term covered = 100.00%
  Input Term   Covered   Reason for no coverage   Hint
  -----      ---      -----
  (counter > 0)      Y

Directive Coverage:
  Directives      14      14      0    100.00%
DIRECTIVE COVERAGE:
-----
Name           Design Design Lang File(Line)      Hits Status
Unit          UnitType
-----
/TOP_MODULE/DUT/cover_p_ss_N_normal   SPI_SLAVE Verilog SVA SPI.sv(293) 3965 Covered
/TOP_MODULE/DUT/cover_p_ss_N_rd_data  SPI_SLAVE Verilog SVA SPI.sv(292) 1026 Covered
/TOP_MODULE/DUT/cover_p_rx_valid_rd_data  SPI_SLAVE Verilog SVA SPI.sv(291) 63 Covered
/TOP_MODULE/DUT/cover_p_rx_valid_rd_addr  SPI_SLAVE Verilog SVA SPI.sv(290) 1 Covered
/TOP_MODULE/DUT/cover_p_rx_valid_wr_data  SPI_SLAVE Verilog SVA SPI.sv(289) 2 Covered
/TOP_MODULE/DUT/cover_p_rx_valid_wr_addr  SPI_SLAVE Verilog SVA SPI.sv(288) 2 Covered
/TOP_MODULE/DUT/cover_p_readdata_to_IDLE  SPI_SLAVE Verilog SVA SPI.sv(287) 110 Covered
/TOP_MODULE/DUT/cover_p_readdaddr_to_IDLE SPI_SLAVE Verilog SVA SPI.sv(286) 181 Covered
/TOP_MODULE/DUT/cover_p_WRITE_to_IDLE   SPI_SLAVE Verilog SVA SPI.sv(285) 266 Covered
/TOP_MODULE/DUT/cover_p_chk_to_READ_DATA  SPI_SLAVE Verilog SVA SPI.sv(284) 156 Covered
/TOP_MODULE/DUT/cover_p_chk_to_READ_ADDRESS  SPI_SLAVE Verilog SVA SPI.sv(283) 226 Covered
/TOP_MODULE/DUT/cover_p_chk_to_WRITE   SPI_SLAVE Verilog SVA SPI.sv(282) 336 Covered
/TOP_MODULE/DUT/cover_p_IDLE_to_chk   SPI_SLAVE Verilog SVA SPI.sv(281) 756 Covered
/TOP_MODULE/DUT/cover_p_RST          SPI_SLAVE Verilog SVA SPI.sv(280) 208 Covered
Statement Coverage:

```

```

=====  

Statement Coverage:  

  Enabled Coverage      Bins    Hits    Misses   Coverage  

  -----  

  Statements           41      41      0     100.00%  

=====  

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

Statement Coverage for instance /TOP_MODULE/DUT --  

Line      Item          Count    Source  

----  

File SPI.sv
  1           module SPI_SLAVE (SPI_IF spi_if);  

  2             reg [3:0] counter;  

  3             reg      received_address;  

  4             // state registers  

  5             reg [2:0] cs, ns;  

  6  

  7             always @(posedge spi_if.clk) begin  

  8               if (~spi_if.rst_n) begin  

  9                 cs <= spi_if.IDLE;  

 10                end  

 11               else begin  

 12                 cs <= ns;  

 13                end  

 14               end  

 15  

 16               // next state logic  

 17               always @(*) begin  

 18                 case (cs)  

 19                   spi_if.IDLE: begin  

 20                     if (spi_if.SS_n)  

 21                       ns = spi_if.IDLE;  

 22                     else  

 23                       ns = spi_if.CHK_CMD;  

 24                     end  

 25  

 26                     spi_if.CHK_CMD: begin  

 27                       if (spi_if.SS_n)  

 28                         ns = spi_if.IDLE;  

 29                     else begin  

 30                       if (~spi_if.MOSI)  

 31                         ns = spi_if.IDLE;  

 32  

Toggle Coverage:  

  Enabled Coverage      Bins    Hits    Misses   Coverage  

  -----  

  Toggles            22      22      0     100.00%  

=====  

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

Toggle Coverage for instance /TOP_MODULE/DUT --  

          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

```

=====  

== Instance: /\ram_top#DUT  

== Design Unit: work.RAM  

=====  

Branch Coverage:  

  Enabled Coverage      Bins    Hits    Misses   Coverage  

  -----  

  Branches            8      8      0     100.00%  

=====  

=====Branch Details=====  

Branch Coverage for instance /\ram_top#DUT  

Line      Item          Count    Source  

----  

File RAM.v
  -----IF Branch-----  

  14           15001  Count coming in to IF  

  14           764    if (~rst_n) begin  

  20           14237  else begin //begin..end  

Branch totals: 2 hits of 2 branches = 100.00%  

  -----IF Branch-----  

  22           14237  Count coming in to IF  

  22           13566  if (rx_valid) begin  

  22           671    All False Count  

Branch totals: 2 hits of 2 branches = 100.00%  

  -----CASE Branch-----  

  23           13566  Count coming in to CASE  

  24           4578   2'b00 : begin  

  27           2256   2'b01 :begin  

  30           3743   2'b10 :begin  

  33           2989   2'b11 : begin  

Branch totals: 4 hits of 4 branches = 100.00%

```

Wrapper

```

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

Toggle Coverage for instance /wrapper_top/dut --

          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)

```

Functional/Sequential Domain Coverage Report

SPI Slave

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/spi_coverage_pkg...		100.00%							
_ TYPE cvr_gp...		100.00%	100	100.00...	 ✓			auto(0)	
_ CVP cvr_g...		100.00%	100	100.00...	 ✓				
_ CVP cvr_g...		100.00%	100	100.00...	 ✓				
_ CVP cvr_g...		100.00%	100	100.00...	 ✓				
_ CVP cvr_g...		100.00%	100	100.00...	 ✓				
_ CROSS cvr...		100.00%	100	100.00...	 ✓				
_ INST Vspi...		100.00%	100	100.00...	 ✓				0

COVERGROUP COVERAGE:				
Covergroup	Metric	Goal	Bins	Status
TYPE /spi_coverage_pkg/spi_coverage/cvr_gp	100.00%	100	-	Covered
covered/total bins:	18	18	-	
missing/total bins:	0	18	-	
% Hit:	100.00%	100	-	
Coverpoint rx_data_cp	100.00%	100	-	Covered
covered/total bins:	7	7	-	
missing/total bins:	0	7	-	
% 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 SS_N_cp	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Coverpoint MOSI_cp	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Cross SS_MOSI_c	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Covergroup instance /spi_coverage_pkg::spi_coverage::cvr_gp	100.00%	100	-	Covered
covered/total bins:	18	18	-	
missing/total bins:	0	18	-	
% Hit:	100.00%	100	-	
Coverpoint rx_data_cp	100.00%	100	-	Covered
covered/total bins:	7	7	-	
missing/total bins:	0	7	-	
% Hit:	100.00%	100	-	
Coverpoint bin all_vals[0]	2854	1	-	Covered
Coverpoint bin all_vals[1]	2002	1	-	Covered
Coverpoint bin all_vals[2]	2837	1	-	Covered
Coverpoint bin all_vals[3]	2308	1	-	Covered
Coverpoint bin wraddr_2_wrdata	99	1	-	Covered
Coverpoint bin rdaddr_2_rddata	114	1	-	Covered
Coverpoint bin rddata_2_wraddr	46	1	-	Covered
Coverpoint ss_n_cp	100.00%	100	-	Covered
covered/total bins:	2	2	-	
Coverpoint ss_n_cp	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
Coverpoint bin normal_trans	458	1	-	Covered
Coverpoint bin readdata_trans	149	1	-	Covered
Coverpoint SS_N_cp	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Coverpoint bin normal	608	1	-	Covered
Coverpoint MOSI_cp	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Coverpoint bin write_addr	2417	1	-	Covered
Coverpoint bin write_data	912	1	-	Covered
Coverpoint bin read_addr	961	1	-	Covered
Coverpoint bin read_data	2555	1	-	Covered
Coverpoint default bin others	10001	-	-	Occurred
Cross SS_MOSI_c	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
Coverpoint bin <normal.read_data>	142	1	-	Covered
Coverpoint bin <normal.read_addr>	170	1	-	Covered
Coverpoint bin <normal.write_data>	142	1	-	Covered
Coverpoint bin <normal.write_addr>	143	1	-	Covered

TOTAL COVERGROUP COVERAGE: 100.00% COVERGROUP TYPES: 1

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
spi_main_seq_pkg:api_main_seq:body#ublk#2151..._immediate	SVA	on	0	0	-	-	-	-	-	0	off	assert(\$urandom >= 0)	✗
spi_main_seq_pkg:api_main_seq:body#ublk#2151..._immediate	SVA	on	0	0	-	-	-	-	-	0	off	assert(\$car(\$ro[0]))	✓
spi_main_seq_pkg:api_main_seq:body#ublk#2151..._immediate	SVA	on	0	1	-	-	-	-	-	0	off	assert(\$randomz(...))	✓
/TOP_MODULE/DUT/assert_p_rst	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (~sp_i.rst)	✓
/TOP_MODULE/DUT/assert_p_IDLE_rd...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (sp_i.rst)	✓
/TOP_MODULE/DUT/assert_p_rx_to_WRITE	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (sp_i.rst)	✓
/TOP_MODULE/DUT/assert_p_chk_to_READ_A...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (sp_i.rst)	✓
/TOP_MODULE/DUT/assert_p_chk_to_READ_D...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (sp_i.rst)	✓
/TOP_MODULE/DUT/assert_p_chk_to_IDLE...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (sp_i.rst)	✓
/TOP_MODULE/DUT/assert_p_readdata_to_IDLE...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (sp_i.rst)	✓
/TOP_MODULE/DUT/assert_p_rx_valid_rd...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (sp_i.rst)	✓
/TOP_MODULE/DUT/assert_p_rx_valid_rd_data...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (sp_i.rst)	✓
/TOP_MODULE/DUT/assert_p_rx_valid_rd_data...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (sp_i.rst)	✓
/TOP_MODULE/DUT/assert_p_ss_N_rd_data	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (sp_i.rst)	✓
/TOP_MODULE/DUT/assert_p_ss_N_normal	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge sp_i.f,dk) (sp_i.rst)	✓

ASSERTION RESULTS:

Name	File(Line)	Failure Count	Pass Count
/TOP_MODULE/DUT/assert_p_ss_N_normal	SPI.sv(273)	0	1
/TOP_MODULE/DUT/assert_p_ss_N_rd_data	SPI.sv(278)	0	1
/TOP_MODULE/DUT/assert_p_rx_valid_rd_data	SPI.sv(267)	0	1
/TOP_MODULE/DUT/assert_p_rx_valid_rd_addr	SPI.sv(264)	0	1
/TOP_MODULE/DUT/assert_p_rx_valid_wr_data	SPI.sv(261)	0	1
/TOP_MODULE/DUT/assert_p_rx_valid_wr_addr	SPI.sv(258)	0	1
/TOP_MODULE/DUT/assert_p_readdata_to_IDLE	SPI.sv(255)	0	1
/TOP_MODULE/DUT/assert_p_readadd_to_IDLE	SPI.sv(252)	0	1
/TOP_MODULE/DUT/assert_p_WRITE_to_IDLE	SPI.sv(249)	0	1
/TOP_MODULE/DUT/assert_p_chk_to_READ_DATA	SPI.sv(246)	0	1
/TOP_MODULE/DUT/assert_p_chk_to_READ_ADDRESS	SPI.sv(243)	0	1
/TOP_MODULE/DUT/assert_p_chk_to_WRITE	SPI.sv(240)	0	1
/TOP_MODULE/DUT/assert_p_IDLE_to_chk	SPI.sv(237)	0	1
/TOP_MODULE/DUT/assert_p_rst	SPI.sv(234)	0	1
/spi_main_seq_pkg:spi_main_seq:body#ublk#267846023#24#immmed_26	SPI_main_seq.sv(26)	0	1

Name	Language	Enabled	Log	Count	AtLeast	Limit	Capit %	Capit graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/TOP_MODULE/DUT/cover_p_ss...	SVA	✓	Off	3965	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_ss...	SVA	✓	Off	1026	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_rx...	SVA	✓	Off	63	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_rx...	SVA	✓	Off	1	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_rx...	SVA	✓	Off	2	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_rx...	SVA	✓	Off	2	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_re...	SVA	✓	Off	110	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_re...	SVA	✓	Off	181	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_W...	SVA	✓	Off	266	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_ch...	SVA	✓	Off	155	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_ch...	SVA	✓	Off	225	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_ch...	SVA	✓	Off	336	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_ID...	SVA	✓	Off	756	1	100%	100%	✓	0	0	0	0 ns	0
/TOP_MODULE/DUT/cover_p_rs...	SVA	✓	Off	208	1	100%	100%	✓	0	0	0	0 ns	0

DIRECTIVE COVERAGE:

Name	Design Unit	Design UnitType	Lang File(Line)	Hits	Status
/TOP_MODULE/DUT/cover_p_ss_N_normal	SPI_SLAVE	Verilog	SVA SPI.sv(293)	3965	Covered
/TOP_MODULE/DUT/cover_p_ss_N_rd_data	SPI_SLAVE	Verilog	SVA SPI.sv(292)	1026	Covered
/TOP_MODULE/DUT/cover_p_rx_valid_rd_data	SPI_SLAVE	Verilog	SVA SPI.sv(291)	63	Covered
/TOP_MODULE/DUT/cover_p_rx_valid_rd_addr	SPI_SLAVE	Verilog	SVA SPI.sv(290)	1	Covered
/TOP_MODULE/DUT/cover_p_rx_valid_wr_data	SPI_SLAVE	Verilog	SVA SPI.sv(289)	2	Covered
/TOP_MODULE/DUT/cover_p_rx_valid_wr_addr	SPI_SLAVE	Verilog	SVA SPI.sv(288)	2	Covered
/TOP_MODULE/DUT/cover_p_readdata_to_IDLE	SPI_SLAVE	Verilog	SVA SPI.sv(287)	110	Covered
/TOP_MODULE/DUT/cover_p_readadd_to_IDLE	SPI_SLAVE	Verilog	SVA SPI.sv(286)	181	Covered
/TOP_MODULE/DUT/cover_p_WRITE_to_IDLE	SPI_SLAVE	Verilog	SVA SPI.sv(285)	266	Covered
/TOP_MODULE/DUT/cover_p_chk_to_READ_DATA	SPI_SLAVE	Verilog	SVA SPI.sv(284)	156	Covered
/TOP_MODULE/DUT/cover_p_chk_to_READ_ADDRESS	SPI_SLAVE	Verilog	SVA SPI.sv(283)	226	Covered
/TOP_MODULE/DUT/cover_p_chk_to_WRITE	SPI_SLAVE	Verilog	SVA SPI.sv(282)	336	Covered
/TOP_MODULE/DUT/cover_p_IDLE_to_chk	SPI_SLAVE	Verilog	SVA SPI.sv(281)	756	Covered
/TOP_MODULE/DUT/cover_p_rst	SPI_SLAVE	Verilog	SVA SPI.sv(280)	208	Covered

TOTAL DIRECTIVE COVERAGE: 100.00% COVERS: 14

RAM

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/ram_coverage_pk...	100.00%	100	100.00...	100.00%	✓				auto(0)
TYPE cvr_gp	100.00%	100	100.00...	100.00%	✓				
CVP cvr_g...	100.00%	100	100.00...	100.00%	✓				
CVP cvr_g...	100.00%	100	100.00...	100.00%	✓				
CVP cvr_g...	100.00%	100	100.00...	100.00%	✓				
CROSS cvr...	100.00%	100	100.00...	100.00%	✓				
CROSS cvr...	100.00%	100	100.00...	100.00%	✓				
INST \ra...	100.00%	100	100.00...	100.00%	✓				0

COVERGROUP COVERAGE:				
Covergroup	Metric	Goal	Bins	Status
TYPE /ram_coverage_pkg/ram_coverage/cvr_gp	100.00%	100	-	Covered
covered/total bins:	19	19	-	
missing/total bins:	0	19	-	
% Hit:	100.00%	100	-	
Coverpoint command_cp	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 rx_valid_with_commands	100.00%	100	-	Covered
covered/total bins:	7	7	-	
missing/total bins:	0	7	-	
% Hit:	100.00%	100	-	
Cross tx_valid_with_read_data	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Covergroup instance \ram_coverage_pkg::ram_coverage::cvr_gp	100.00%	100	-	Covered
covered/total bins:	19	19	-	
missing/total bins:	0	19	-	
% Hit:	100.00%	100	-	
Coverpoint command_cp	100.00%	100	-	Covered
covered/total bins:	7	7	-	
missing/total bins:	0	7	-	
% Hit:	100.00%	100	-	
bin write_address	5035	1	-	Covered
bin write_data	2493	1	-	Covered
bin read_address	4161	1	-	Covered
bin read_data	3312	1	-	Covered
bin write_data_after_address	2492	1	-	Covered
bin read_data_after_address	3312	1	-	Covered
bin transition	235	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 auto[0]	712	1	-	Covered
bin auto[1]	14289	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 auto[0]	12012	1	-	Covered
bin auto[1]	2989	1	-	Covered
Cross rx_valid_with_commands	100.00%	100	-	Covered
covered/total bins:	7	7	-	
missing/total bins:	0	7	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <transition.auto[1]>	227	1	-	Covered
bin <read_data_after_address,auto[1]>	3162	1	-	Covered
bin <write_data_after_address,auto[1]>	2377	1	-	Covered
bin <read_data,auto[1]>	3162	1	-	Covered
bin <write_data,auto[1]>	2378	1	-	Covered
bin <read_address,auto[1]>	3956	1	-	Covered
bin <write_address,auto[1]>	4793	1	-	Covered
Illegal and Ignore Bins:				
ignore_bin rx_valid_zero	712	-	-	Occurred
Cross tx_valid_with_read_data	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 tx_valid_rd	2989	1	-	Covered

TOTAL COVERGROUP COVERAGE: 100.00% COVERGROUP TYPES: 1

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
▲ /ram_pkp/uum_reg_map:dp_write#ublk#215...	Immediate	SVA	on	0	0	-	-	-	-	off	assert(\$cast(se,o))	✗	
▲ /ram_pkp/uum_reg_map:dp_read#ublk#2151...	Immediate	SVA	on	0	0	-	-	-	-	off	assert(\$cast(se,o))	✗	
▲ /ram_read_seq_pkg:ram_read_seq...	Immediate	SVA	on	0	1	-	-	-	-	off	assert(randomize(..))	✓	
▲ /ram_read_seq_pkg:ram_read_only_seq:body...	Immediate	SVA	on	0	1	-	-	-	-	off	assert(randomize(..))	✓	
▲ /ram_write_seq_pkg:ram_write_only_seq:body...	Immediate	SVA	on	0	1	-	-	-	-	off	assert(randomize(..))	✓	
+▲ /ram_top/DUT/ram_sva_inst/assert_reset_outp...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@posedge ck) (~rst_n)==...	✓	
+▲ /ram_top/DUT/ram_sva_inst/assert_tx_valid_lo...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@posedge ck) disable iff (...)	✓	
+▲ /ram_top/DUT/ram_sva_inst/assert_tx_valid_hi...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@posedge ck) disable iff (...)	✓	
+▲ /ram_top/DUT/ram_sva_inst/assert_write_data...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@posedge ck) disable iff (...)	✓	
+▲ /ram_top/DUT/ram_sva_inst/assert_read_data...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@posedge ck) disable iff (...)	✓	

ASSERTION RESULTS:

Name	File(Line)	Failure Count	Pass Count
/ram_top/DUT/ram_sva_inst/assert_read_data_after_address	RAM_sva.sv(49)	0	1
/ram_top/DUT/ram_sva_inst/assert_write_data_after_address	RAM_sva.sv(40)	0	1
/ram_top/DUT/ram_sva_inst/assert_tx_valid_high	RAM_sva.sv(31)	0	1
/ram_top/DUT/ram_sva_inst/assert_tx_valid_low	RAM_sva.sv(22)	0	1
/ram_top/DUT/ram_sva_inst/assert_reset_outputs_low	RAM_sva.sv(14)	0	1
/ram_read_write_seq_pkg:ram_read_write_seq:body/#ublk#31146023#20/immed_24	RAM_read_write_seq.sv(24)	0	1
/ram_read_seq_pkg:ram_read_only_seq:body/#ublk#256611655#20/immed_24	RAM_read_only_seq.sv(24)	0	1
/ram_write_seq_pkg:ram_write_only_seq:body/#ublk#114248983#20/immed_24	RAM_write_only_seq.sv(24)	0	1

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
▲ /ram_top/DUT/ram_sva_inst/cover...	SVA	✓	Off	3643	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
▲ /ram_top/DUT/ram_sva_inst/cover...	SVA	✓	Off	4381	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
▲ /ram_top/DUT/ram_sva_inst/cover...	SVA	✓	Off	2781	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
▲ /ram_top/DUT/ram_sva_inst/cover...	SVA	✓	Off	10545	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
▲ /ram_top/DUT/ram_sva_inst/cover...	SVA	✓	Off	764	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0

DIRECTIVE COVERAGE:

Name	Design Unit	Design Unit	Lang	File(Line)	Hits	Status
	Design Unit	Design Unit	Lang	File(Line)	UnitType	
/ram_top/DUT/ram_sva_inst/cover_read_data_after_address	ram_sva Verilog	SVA	RAM_sva.sv(51)		3643	Covered
/ram_top/DUT/ram_sva_inst/cover_write_data_after_address	ram_sva Verilog	SVA	RAM_sva.sv(42)		4381	Covered
/ram_top/DUT/ram_sva_inst/cover_tx_valid_high	ram_sva Verilog	SVA	RAM_sva.sv(33)		2781	Covered
/ram_top/DUT/ram_sva_inst/cover_tx_valid_low	ram_sva Verilog	SVA	RAM_sva.sv(24)		10545	Covered
/ram_top/DUT/ram_sva_inst/cover_reset_outputs_low	ram_sva Verilog	SVA	RAM_sva.sv(16)		764	Covered

TOTAL DIRECTIVE COVERAGE: 100.00% COVERS: 5

Wrapper

Covergroups										
Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment	
spi_coverage_pkg...		100.00%								
+ TYPE cvr_gp		100.00%	100	100.00%	 ✓				auto(0)	
/ram_coverage_pk...		100.00%								
+ TYPE cvr_gp		100.00%	100	100.00%	 ✓				auto(0)	

```

=====
== Instance: /Wrapper_top/dut
== Design Unit: work.WRAPPER
=====

Assertion Coverage:
 Assertions 2 2 0 100.00%
-----
Name File(Line) Failure Count Pass Count
-----
/wrapper_top/dut/assert__MISO_stable
    wrapper.sv(54) 0 1
/wrapper_top/dut/assert__reset_output_low
    wrapper.sv(45) 0 1

Directive Coverage:
 Directives 2 2 0 100.00%
-----
DIRECTIVE COVERAGE:
-----
Name Design Design Lang File(Line) Hit
Unit UnitType
-----
/wrapper_top/dut/cover__MISO_stable WRAPPER Verilog SVA wrapper.sv(56) 12
/wrapper_top/dut/cover__reset_output_low WRAPPER Verilog SVA wrapper.sv(47) 3

```

A Cover Directives	Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
✓ /wrapper_top/dut/cover__MISO_stable	SVA	✓	Off	12442	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/cover__reset_output_low	SVA	✓	Off	317	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_ss...	SVA	✓	Off	6008	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_ss...	SVA	✓	Off	1484	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_rx...	SVA	✓	Off	14	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_rx...	SVA	✓	Off	1	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_rx...	SVA	✓	Off	1	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_rx...	SVA	✓	Off	210	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_re...	SVA	✓	Off	242	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_re...	SVA	✓	Off	513	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_W...	SVA	✓	Off	279	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_ch...	SVA	✓	Off	287	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_ch...	SVA	✓	Off	638	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_ID...	SVA	✓	Off	1280	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/SLAVE_instance/cover_p_rs...	SVA	✓	Off	317	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/RAM_instance/ram_sva_inst...	SVA	✓	Off	2552	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/RAM_instance/ram_sva_inst...	SVA	✓	Off	2743	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/RAM_instance/ram_sva_inst...	SVA	✓	Off	621	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/RAM_instance/ram_sva_inst...	SVA	✓	Off	1321	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	
✓ /wrapper_top/dut/RAM_instance/ram_sva_inst...	SVA	✓	Off	317	1	Unl...	1	100%	✓	✓	0	0	0 ns	0	

Bug Report

SPI Slave

1.

Design Input for Bug to Appear: we are in the check_cmd case, ss_n is low, mosi is high and the received address is high

Expected Behavior: ns should be read data

Observed Behavior: ns was read address

2.

Design Input for Bug to Appear: when reset is asserted

Expected Behavior: all signals and internal signals should be low

Observed Behavior: counter didn't change to be 0

3.

Design Input for Bug to Appear: at the read data case after entering dummy data

Expected Behavior: reset the counter to start read data on the miso to get all bits

Observed Behavior: counter didn't cover all bits

RAM

1.

Design Input for Bug to Appear: at the case of read data of the din[9:8]

Expected Behavior: read data from the place that read address refers to

Observed Behavior: read the data from the place the write address refers to

2.

Design Input for Bug to Appear: at the case of read data of the din[9:8]

Expected Behavior: spi should take bit by bit of the tx data when tx valid is high

Observed Behavior: tx valid is high at just one cycle so it takes only one bit

3.

the syntax errors: use mem instead of MEM, and add "begin ... end" in the else condition

QuestaSim Snippets

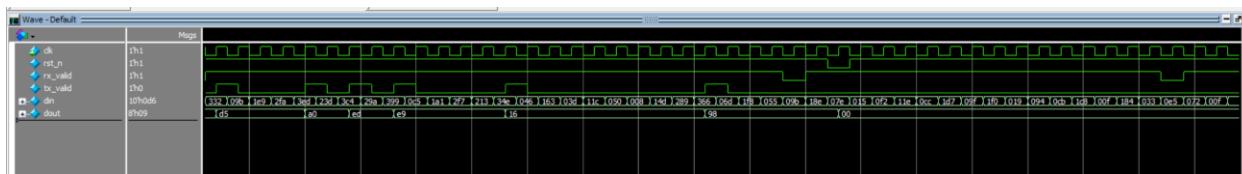
SPI Slave

```
*****
# UVM_INFO SPI_test.sv(50) @ 2: uvm_test_top [run_phase] reset deasserted
# UVM_INFO SPI_test.sv(53) @ 2: uvm_test_top [run_phase] main asserted
# UVM_INFO SPI_test.sv(55) @ 20002: uvm_test_top [run_phase] main deasserted
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(126) @ 20002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO SPI_scoreboard.sv(64) @ 20002: uvm_test_top.env.score [report_phase] total correct trannxactions: 10001
# UVM_INFO SPI_scoreboard.sv(65) @ 20002: uvm_test_top.env.score [report_phase] total failed trannxactions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 10
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
# ** Report counts by id
# [Questa UVM] 2
# [RNTST] 1
# [TEST_DONE] 1
# [report_phase] 2
# [run_phase] 4
# ** Note: $finish : C:/questasim64_2021.1/win64/..//verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
# Time: 20002 ns Iteration: 61 Instance: /TOP_MODULE
# 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
```



RAM

```
*****
# UVM_INFO RAM_test.sv(60) @ 2: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO RAM_test.sv(63) @ 2: uvm_test_top [run_phase] Stimulus Generation Started
# UVM_INFO RAM_test.sv(67) @ 30002: uvm_test_top [run_phase] Stimulus Generation Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(126) @ 30002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO RAM_scoreboard.sv(50) @ 30002: uvm_test_top.env_sb [report_phase] Total successful transactions: 15001
# UVM_INFO RAM_scoreboard.sv(51) @ 30002: uvm_test_top.env_sb [report_phase] Total failed transactions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 10
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
# ** Report counts by id
# [Questa UVM] 2
# [RNTST] 1
# [TEST_DONE] 1
# [report_phase] 2
# [run_phase] 4
# ** Note: $finish : C:/questasim64_2021.1/win64/..//verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
# Time: 30002 ns Iteration: 61 Instance: /ram_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
```



Wrapper

```
# ****
# UVM_INFO wrapper_test.sv(102) @ 2: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO wrapper_test.sv(105) @ 2: uvm_test_top [run_phase] Stimulus Generation Started
# UVM_INFO wrapper_test.sv(109) @ 30002: uvm_test_top [run_phase] Stimulus Generation Ended
# UVM_INFO wrapper_test.sv(110) @ 30002: uvm_test_top [wrapper_test] All sequences completed successfully
# UVM_INFO verilog_src/uvm-1.ld/src/base/uvm_objection.svh(1267) @ 30002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO wrapper_scoreboard.sv(53) @ 30002: uvm_test_top.env.ab [report_phase] Total successful transactions: 15001
# UVM_INFO wrapper_scoreboard.sv(54) @ 30002: uvm_test_top.env.ab [report_phase] Total failed transactions: 0
# UVM_INFO wrapper_env.sv(40) @ 30002: uvm_test_top.env [WRAPPER_ENV] Wrapper Environment completed successfully
# UVM_INFO RAM_scoreboard.sv(50) @ 30002: uvm_test_top._env.ab [report_phase] Total successful transactions: 15001
# UVM_INFO RAM_scoreboard.sv(51) @ 30002: uvm_test_top._env.ab [report_phase] Total failed transactions: 0
# UVM_INFO SPI_scoreboard.sv(64) @ 30002: uvm_test_top._env.score [report_phase] total correct transactions: 15001
# UVM_INFO SPI_scoreboard.sv(65) @ 30002: uvm_test_top._env.score [report_phase] total failed transactions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 17
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
# ** Report counts by id
# [Questasim] 2
# [RNTST] 1
# [TEST_DONE] 1
# [WRAPPER_ENV] 1
# [report_phase] 6
# [run_phase] 4
# [wrapper_test] 2
# ** Note: $finish : C:/questasim64_2021.1/win64/..//verilog_src/uvm-1.ld/src/base/uvm_root.svh(430)
# Time: 30002 ns Iteration: 61 Instance: /wrapper_top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/..//verilog_src/uvm-1.ld/src/base/uvm_root.svh line 430
```



Assertions table

SPI Slave

Feature	Assertion
When the rst is asserted, MISO , rx_valid & rx_data are low	@(posedge spi_if.clk) (!spi_if.rst_n) => (spi_if.MISO == 0 && spi_if.rx_valid == 0 && spi_if.rx_data == 0);
When the cs is idle & SS_n is low , cs is chk_cmd in next clock cycle	@(posedge spi_if.clk) disable iff(!spi_if.rst_n) (cs == spi_if.IDLE && !spi_if.SS_n) => (cs == spi_if.CHK_CMD);
When the cs is chk_cmd & MOSI is low , cs is write in next clock cycle	@(posedge spi_if.clk) disable iff(!spi_if.rst_n spi_if.SS_n) (cs == spi_if.CHK_CMD && !spi_if.MOSI) => (cs == spi_if.WRITE);
When the cs is chk_cmd , MOSI is high & received address is low , cs is read address in next clock cycle	@(posedge spi_if.clk) disable iff(!spi_if.rst_n spi_if.SS_n) (cs == spi_if.CHK_CMD && spi_if.MOSI && !received_address) => (cs == spi_if.READ_ADDRESS);
When the cs is chk_cmd , MOSI is high & received address is high , cs is read data in next clock cycle	@(posedge spi_if.clk) disable iff(!spi_if.rst_n spi_if.SS_n) (cs == spi_if.CHK_CMD && spi_if.MOSI && received_address) => (cs == spi_if.READ_DATA);
When the cs is write & SS_n is high, cs is idle in next clock cycle	@(posedge spi_if.clk) disable iff(!spi_if.rst_n) (cs == spi_if.WRITE && spi_if.SS_n) => (cs == spi_if.IDLE);
When the cs is read address & SS_n is high, cs is idle in next clock cycle	@(posedge spi_if.clk) disable iff(!spi_if.rst_n) (cs == spi_if.READ_ADDRESS && spi_if.SS_n) => (cs == spi_if.IDLE);
When the cs is read data & SS_n is high, cs is idle in next clock cycle	@(posedge spi_if.clk) disable iff(!spi_if.rst_n) (cs == spi_if.READ_DATA && spi_if.SS_n) => (cs == spi_if.IDLE);
When the write address is high ,the rx valid is high after 11 clock cycles	@(posedge spi_if.clk) disable iff(!spi_if.rst_n spi_if.SS_n) (write_addr) => ##10 (spi_if.rx_valid);
When the write data is high ,the rx valid is high after 11 clock cycles	@(posedge spi_if.clk) disable iff(!spi_if.rst_n spi_if.SS_n) (write_data) => ##10 (spi_if.rx_valid);
When the read address is high ,the rx valid is high after 11 clock cycles	@(posedge spi_if.clk) disable iff(!spi_if.rst_n spi_if.SS_n) (read_addr) => ##10 (spi_if.rx_valid);

When the read data is high ,the rx valid is high after 11 clock cycles	<pre>@(posedge spi_if.clk) disable iff(!spi_if.rst_n spi_if.SS_n) (read_data) => ##10 (spi_if.rx_valid);</pre>
When the cs is read data , the SS_n is high after 24 clock cycles	<pre>@(posedge spi_if.clk) disable iff(!spi_if.rst_n) (cs == spi_if.READ_DATA) => ##23 (spi_if.SS_n[->1]);</pre>
When the cs is write or read address , the SS_n is high after 14 clock cycles	<pre>@(posedge spi_if.clk) disable iff(!spi_if.rst_n) (cs == spi_if.WRITE cs == spi_if.READ_ADDRESS) => ##13 (spi_if.SS_n[->1]);</pre>

RAM

Feature	Assertion
When the rst is asserted, tx_valid & dout are low	<pre>@(posedge clk) (!rst_n) => (tx_valid == 0 && dout == 0);</pre>
When the rx_valid is high & din[9:8] is 0 (write address) or 1 (write data) or 2 (read address) , tx_valid is low	<pre>@(posedge clk) disable iff(!rst_n) (rx_valid && (din[9:8]== 2'b00 din[9:8]== 2'b01 din[9:8]== 2'b10)) => (tx_valid == 0);</pre>
When din[9:8] is 3 (read data) , tx_valid is rising and then it is falling	<pre>@(posedge clk) disable iff(!rst_n) (din[9:8] == 2'b11) => \$rose(tx_valid)[-1] ##1 \$fell(tx_valid)[-1];</pre>
When din[9:8] is 0 (write address) , din[9:8] is 1 (write data) in next clock cycle	<pre>@(posedge clk) disable iff(!rst_n) (din[9:8]== 2'b00) => (din[9:8]== 2'b01 [-1]);</pre>
When din[9:8] is 2 (read address) , din[9:8] is 3 (read data) in next clock cycle	<pre>@(posedge clk) disable iff(!rst_n) (din[9:8]== 2'b10) => (din[9:8]== 2'b11 [-1]);</pre>

Wrapper

Feature	Assertion
When the rst is asserted, MISO is low	<pre>@(posedge clk) (!rst_n) => (MISO== 0);</pre>
When the cs isn't read data , the MISO is stable	<pre>@(posedge clk) disable iff (!rst_n) (!tx_valid) => \$stable(MISO)[-1];</pre>