SPI master slave with a dual-port memory

Project report

December,2024

Abanob Evram Shahata

You can check the files in [Github]

Table of Contents

Verification plan of RAM:

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
rst_assert	When the reset is asserted, the output will be low	Directed at the start of the simulation	Cover possible values of rst	A checker in the testbench to make sure the output is correct. Immdiate assertion to check Reset(ASY) functionality.
write_only	This seq for write address and write data only. In this seq we will write 10000 iteration and will write in all locations	repeat block to test randomized value	Cover the number of wr_add and wr_data occurs	A checker in the testbench to make sure the output is correct. Conncurrent assertions to check the wr_addr&wr_data
read_only	This seq for read address and read data only. In this seq we will read 10000 iteration and will read in all locations	repeat block to test randomized value	Cover the number of rd_add and rd_data occurs	A checker in the testbench to make sure the output is correct. Conncurrent assertions to check the rd_addr&rd_data
write_read	Randomize the 4 operations 10000 iteration	repeat block to test randomized value	Cover all operation , din and rx_valid	A checker in the testbench to make sure the output is correct. Conncurrent assertions to check all output
rst_prob	When the reset is asserted, the output will be low	Randomize during simulation to be inactive most of the time	Cover possible values of rst	A checker in the testbench to make sure the output is correct. Immdiate assertion to check Reset(ASY) functionality.
rx_valid_prob	The signal rx_valid when be high the ram will be check the operation and do it	Randomize during simulation to be on 95 of the time	Cover the two values of rx_valid	A checker in the testbench to make sure the output is correct. Conncurrent assertions to check all output

din_prob	This signal is 10 bit the latest to bit detect the operation and the other 8 bits for data	Randomize under constraint which handle all operatinons and all cases and handel the read address befor read data	Cover all operation , din and rx_valid	A checker in the testbench to make sure the output is correct. Conncurrent assertions to check all output

Verification plan of Slave:

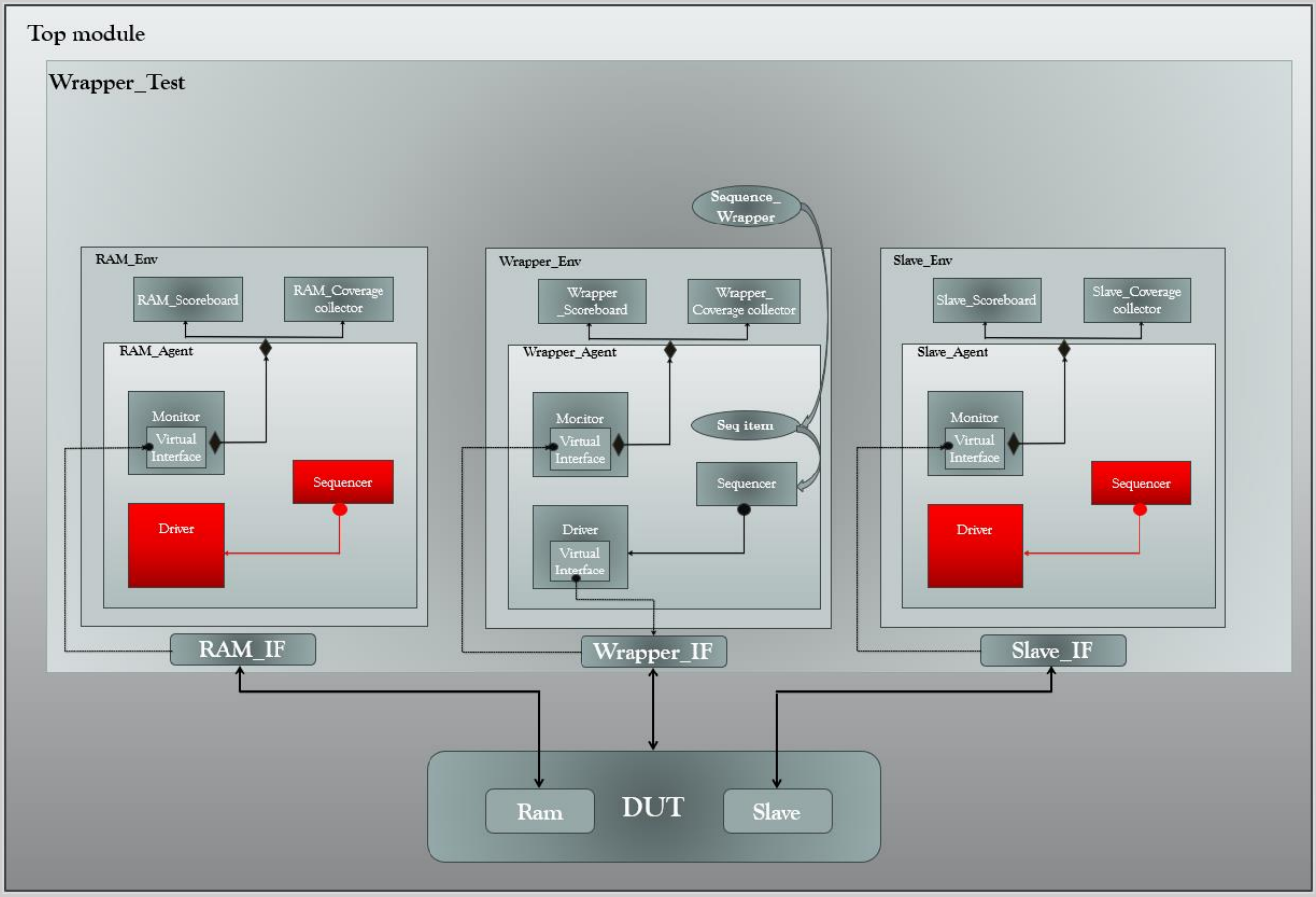
Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
rst_assert	When the reset is asserted, the output will be low	Directed at the start of the simulation	Cover possible values of rst	A checker in the testbench to make sure the output is correct. Immdiate assertion to check Reset(ASY) functionality.
main_seq	This seq will test all operation in FSM from start the communication when the SS_n active to the end of the operation. And will do this 20000 iteration	repeat block to test randomized value under some constrains	Cover the four current state 1.wr_add 2.wr_data 3.rd_add 4.rd_data	A checker in the testbench to make sure the output is correct. Conncurrent assertions to check the Current state
rand_seq	This seq will randomize all signal with out any constrains to other cases and conditons	repeat block to test randomized value with ouut any constrains	Cover the four current state 1.wr_add 2.wr_data 3.rd_add 4.rd_data	A checker in the testbench to make sure the output is correct. Conncurrent assertions to check the Current state
rst_prob	When the reset is asserted, the output will be low	Randomize during simulation to be inactive most of the time	Cover possible values of rst	A checker in the testbench to make sure the output is correct. Immdiate assertion to check Reset(ASY) functionality.
SS_N_prob	This constraint will active the SS_n at the start of communication and after end the communicationaccording to the operation	Randomize under specific functionality	Cover the four current state 1.wr_add 2.wr_data 3.rd_add 4.rd_data	A checker in the testbench to make sure the output is correct. Conncurrent assertions to check the Current state

tx_valid_c	This signal tell the ram when the operation is read_data that the slave finish sending all 10 bits and the ram will send the data from memory	Randomize to be active after sending the dummy data and before the RAM finishes sending the data	-	A checker in the testbench to make sure the output is correct. Conncurrent assertions to check the Current state
MOSI_arr_prop	This array will be used to randomize all MOSI bits, which will be used in the FSM after the end of every communication of operation.	Randomize under constraint which handle all operatinons and all cases and handel the read address befor read data	-	A checker in the testbench to make sure the output is correct. Conncurrent assertions to check the Current state
MOSI_prop	This constraint will reflect the current bit of MOSI_arr	Randomize values from MOSI_Arr	-	A checker in the testbench to make sure the output is correct. Conncurrent assertions to check the Current state

For wrapper:

It is the same plane as the slave, but the difference is that there is no rand_sequence, and we do not randomize the tx_valid signal

UVM Structure:



UVM FLOW:

❖ Initial Testing

- Started by creating standalone tests for the RAM and Slave components.

❖ Wrapper Flow Creation

1. **Create Two Shared Packages:**
    - Developed two shared packages:
        - One for the RAM files.
        - Another for the Wrapper and Slave files.
    - These packages include shared variables and parameters used across the files.
2. **Create Three Interfaces:**
    - Defined interfaces for the three designs to connect the DUT with the testbench components.

3. **Develop Assertion Files:**
    - Created three assertion files to verify the designs.
    - Bind these assertion files to the top-level module.
4. **Create Top File:**
    - Generated clocks and initialized memory with default values.
    - Passed data from the wrapper interface to the DUT (Main Design).
    - Routed data from the Slave and Wrapper interfaces to their respective golden models.
    - Bind the three assertion files to the design.
    - When the Slave and RAM were passive, they sent data from their interfaces to the DUT.
    - Registered the interfaces in the configuration database.
    - Ran tests on the Wrapper.

3. Configuration Classes

- Created configuration classes for the three designs to retrieve their virtual interfaces.

4. Sequence Item Classes

- Designed sequence item classes with the following features:
    - Included design signals and random input signals.
    - For passive designs, skipped randomization and directly transferred values from the monitor to the scoreboard and coverage collector.
    - Implemented the convert2string function to print transactions.
    - Added constraints for all three files to control randomization.

5. Wrapper Sequences

- Used two sequences to verify the Wrapper design:
    1. **Reset Sequence:** Directed reset sequence at the beginning.
    2. **Main Sequence:** Generated 50,000 constrained random iterations for testing.

6. Environment Components

1. **Sequencer:**
    - Managed sequences and drive data.
    - Not required for Slave and RAM, as they are passive designs.

2. **Driver File:**
    - Established a virtual interface between the driver and the actual interface.
    - Made the driver a producer, retrieving data and assigning it from the sequence item to interface inputs.
    - Not used for the RAM and Slave.
3. **Monitor:**
    - Set up virtual interfaces for the three designs and the real interaces.
    - Captured data from the virtual interface and passed it to the monitor object using the sequence item's inputs and outputs.
    - Sent data to the agent via analysis_port.
4. **Agent Files:**
    - Retrieved configuration objects from the database and passed them to the monitor and driver.
    - Linked the monitor to the agent to send data to the scoreboard and coverage collector.
    - Connected the sequencer and driver.
5. **Scoreboard Files:**
    - Compared monitor inputs against reference models:
        - Used a task-based reference model for RAM.
        - Used golden models for the Wrapper and Slave.
    - Logged mismatches and test failures.
    - Generated a report for each design at the end of the simulation, detailing the correct iteration count and errors.
6. **Coverage Collector Files:**
    - Monitored states and inputs for all files.
    - Ensured thorough coverage of signals and transitions.
    - Measured test completeness.

7. Environments

- Created three environments (one for each design):
    - Instantiated components like agents, scoreboards, and coverage collectors.
    - Connected the agent to the scoreboard and coverage collector.

8. Test File

- Retrieved virtual interfaces and passed them to the environment.
- Created objects for the sequences.
- Determined whether each design was active or passive.
- Signaled the start of the test using raise_objection.
- Executed the sequences during the run phase.
- Signaled test completion using drop_objection.

Designs and assertions: -

RAM Design:

```
////////////////////////////////////////////////////////////////////////////////
// Author: Abanob Evram
// Description: RAM_Design
// Date: December, 2024
////////////////////////////////////////////////////////////////////////////////
import RAM_shared_pkg::*;
module RAM(din,clk,rst_n,rx_valid,dout,tx_valid);
 input [ADDR_SIZE+1:0] din;
 input clk,rst_n,rx_valid;
 output reg [ADDR_SIZE-1:0] dout;
 output reg tx_valid;
 reg [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];
 reg [ADDR_SIZE-1:0] wr_addr,rd_addr;

 always @(posedge clk or negedge rst_n) begin
    if (~rst_n) begin
        tx_valid<=0;
        dout<=0;
        wr_addr<=0;
        rd_addr<=0;
    end

    else begin
        if(rx_valid) begin
            tx_valid<=((din[9]==1)&(din[8]==1)&(rx_valid==1))?1:0;
            case(din[9:8])
                2'b00:wr_addr<=din[ADDR_SIZE-1:0];
                2'b01:mem[wr_addr]<=din[ADDR_SIZE-1:0];
                2'b10:rd_addr<=din[ADDR_SIZE-1:0];
                2'b11:dout<=mem[rd_addr];
            endcase
        end
    end
 end
endmodule
```

SPI SLAVE Design:

```
////////////////////////////////////////////////////////////////////////////////
// Author: Abanob Evram
// Description: SLAVE_Design
// Date: December, 2024
////////////////////////////////////////////////////////////////////////////////
import wrapper_shared_pkg::*;
module SLAVE(MOSI, SS_n, clk, rst_n, tx_valid,MISO,rx_valid,rx_data,tx_data);

 input MOSI, SS_n, clk, rst_n, tx_valid;
 input [ADDR_SIZE-1:0] tx_data;
 output reg [ADDR_SIZE+1:0] rx_data;
 output reg MISO;
 output rx_valid;

(* fsm_encoding = "one_hot" *)

 STATE_e cs,ns;
 reg rd_add_signal;

 integer count_rx,count_tx;

 //MEM stage
 always @(posedge clk or negedge rst_n) begin
    if (~rst_n)
        cs<=IDLE;
    else
        cs<=ns;
 end

 //Next stage logic
 always @(*) begin
    case(cs)
        IDLE: begin
            if(~SS_n)
                ns=CHK_CMD;
            else
                ns=IDLE;
        end
        CHK_CMD: begin
            if(SS_n==0&&MOSI==0)
                ns = WRITE;
            else if(SS_n==0&&MOSI==1&&rd_add_signal==0)
                ns = READ_ADD;
            else if(SS_n==0&&MOSI==1&&rd_add_signal==1)
                ns = READ_DATA;
            else
                ns = IDLE;
        end
        WRITE: begin
            if(~SS_n)
                ns=WRITE;
            else
                ns=IDLE;
        end
        READ_ADD: begin
            if(~SS_n)
                ns=READ_ADD;
            else
                ns=IDLE;
        end
        READ_DATA: begin
            if(~SS_n)
                ns=READ_DATA;
            else
                ns=IDLE;
        end
    endcase
 end

```

```
69
70    //output logic
71    always @(posedge clk or negedge rst_n) begin
72       if (~rst_n) begin
73           rd_add_signal<=0;
74           rx_data<=0;
75           count_tx<=0;
76           count_rx<=0;
77       end
78       else begin
79           case(cs)
80               IDLE: begin
81                   MISO<=0;
82                   count_tx<=0;
83                   count_rx<=0;
84               end
85               CHK_CMD: begin
86                   count_tx<=0;
87                   count_rx<=0;
88               end
89               WRITE: begin
90                   if(count_rx<=9) begin
91                       rx_data[9-count_rx]<=MOSI;
92                       count_rx<=count_rx+1;
93                   end
94               end
95               READ_ADD: begin
96                   if(count_rx<=9) begin
97                       rx_data[9-count_rx]<= MOSI;
98                       count_rx<=count_rx+1;
99                       rd_add_signal<=1;
100                  end
101              end
102              READ_DATA: begin
103                  if(count_rx>9) begin
104                      if(tx_valid==1&&count_tx<8) begin
105                          MISO <= tx_data[7-count_tx];
106                          count_tx<=count_tx+1;
107                      end
108                      else if(tx_valid==1&&count_tx>=8)begin
109                          rd_add_signal <= 0;
110                      end
111                  end
112                  else begin
113                      rx_data[9-count_rx]<= MOSI;
114                      count_rx<=count_rx+1;
115                  end
116              end
117          endcase
118      end
119   end
120   assign rx_valid =((cs==WRITE||cs==READ_ADD||cs==READ_DATA)&&count_rx>9)?1:0 ;
121  endmodule
```

Wrapper Design:

```
1   ////////////////////////////////////////////////////////////////////////////////
2   // Author: Abanob Evram
3   // Description: Wrapper_Design
4   // Date: December, 2024
5   ////////////////////////////////////////////////////////////////////////////////
6   module Maindesign(MOSI,SS_n,clk,rst_n,MISO);
7    input MOSI,SS_n,clk,rst_n;
8    output MISO;
9    wire [9:0] rx_data;
10   wire [7:0] tx_data;
11   wire rx_valid, tx_valid;
12
13  SLAVE    spi(MOSI, SS_n, clk, rst_n, tx_valid,MISO,rx_valid,rx_data,tx_data);
14
15  RAM      ram(rx_data,clk,rst_n,rx_valid,tx_data,tx_valid);
16
17  endmodule
```

Slave Golden model:

```
import wrapper_shared_pkg::*;
module SPI_SLAVE (MOSI,MISO,SS_n,rx_data,rx_valid,tx_data,tx_valid,clk,rst_n );
input MOSI,SS_n,tx_valid,clk,rst_n;
input [7:0] tx_data;
output rx_valid;
output reg MISO;
output reg [9:0] rx_data;
reg internal_signal;

STATE_e cs,ns;
reg [4:0]counter1,counter2;
wire done_count ;
assign done_count =(counter1>=10)? 1:0;

    always @(posedge clk or negedge rst_n) begin
        if (~rst_n)
            cs <= IDLE;
        else
            cs <= ns;
    end

    always @(*) begin
        case (cs)
            IDLE: begin
                if (SS_n )
                    ns = IDLE;
                else
                    ns = CHK_CMD;
            end
            CHK_CMD: begin
                if (SS_n == 0 && MOSI == 0)
                    ns = WRITE;
                else if (SS_n == 0 && MOSI == 1 && internal_signal == 0)
                    ns = READ_ADD;
                else if (SS_n == 0 && MOSI == 1 && internal_signal == 1)
                    ns = READ_DATA;
                else
                    ns = IDLE;
            end
            WRITE: begin
                if (SS_n )
                    ns = IDLE;
                else
                    ns = WRITE;
                end
            READ_ADD: begin
                if (SS_n )
                    ns = IDLE;
                else
                    ns = READ_ADD;
                end
```

```
                end
            READ_DATA: begin
                if (SS_n )
                    ns = IDLE;
                else
                    ns = READ_DATA;
            end
            default: ns = IDLE;
        endcase
    end
    always @(posedge clk or negedge rst_n) begin
        if (~rst_n) begin
            counter1 <= 0;
            rx_data<=0;
            internal_signal <= 0;
            counter2<=0;
        end
```

```
        else begin
            case (cs)
                IDLE: begin
                    counter1 <= 0;
                    counter2<=0;
                    MISO <= 0;
                end
                CHK_CMD: begin
                    counter1<=0;
                    counter2<=0;
                end
                WRITE: begin
                    if (counter1<=9) begin
                        rx_data[9-counter1] <= MOSI;
                        counter1 <= counter1 + 1;
                    end
                end
                READ_ADD: begin
                    if (counter1<=9) begin
                        internal_signal <= 1;
                        rx_data[9-counter1] <= MOSI;
                        counter1 <= counter1 + 1;
                    end
                end
                READ_DATA : begin
                    if (counter1>9)begin
                        if (tx_valid==1&&counter2<8) begin
                            MISO <= tx_data[7-counter2];
                            counter2<=counter2+1;
                        end
                        else if (tx_valid==1&&counter2==8)
                            internal_signal <= 0;
                    end
                    else begin
                        rx_data[9-counter1] <= MOSI;
                        counter1<=counter1+1;
                    end
                end
                default: begin
                    counter1<=0;
                    internal_signal<=0;
                end
            endcase
        end
    end
    assign rx_valid =((cs==WRITE||cs==READ_ADD||cs==READ_DATA)&&done_count)?1:0;
endmodule
```

Wrapper Golden model:

```
import wrapper_shared_pkg::*;
module wrapper_golden(MOSI,SS_n,clk,rst_n,MISO);

 input MOSI, SS_n, clk, rst_n;
 reg tx_valid;
 reg [ADDR_SIZE-1:0] tx_data;
 reg [ADDR_SIZE+1:0] rx_data;
 output reg MISO;
 wire rx_valid;
 reg [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];
 reg [ADDR_SIZE-1:0] wr_addr,rd_addr;

 STATE_e cs,ns;
 reg rd_add_signal;

 integer count_rx,count_tx;

 //MEM stage
 always @(posedge clk or negedge rst_n) begin
    if (~rst_n)
        cs<=IDLE;
    else
        cs<=ns;
 end
```

```
31      //Next stage logic
32      always @(*) begin
33         case(cs)
34             IDLE: begin
35                 if(~SS_n)
36                     ns=CHK_CMD;
37                 else
38                     ns=IDLE;
39             end
40             CHK_CMD: begin
41                 if(SS_n==0&&MOSI==0)
42                     ns = WRITE;
43                 else if(SS_n==0&&MOSI==1&&rd_add_signal==0)
44                     ns = READ_ADD;
45                 else if(SS_n==0&&MOSI==1&&rd_add_signal==1)
46                     ns = READ_DATA;
47                 else
48                     ns = IDLE;
49             end
50             WRITE: begin
51                 if(~SS_n)
52                     ns=WRITE;
53                 else
54                     ns=IDLE;
55             end
56             READ_ADD: begin
57                 if(~SS_n)
58                     ns=READ_ADD;
59                 else
60                     ns=IDLE;
61             end
62             READ_DATA: begin
63                 if(~SS_n)
64                     ns=READ_DATA;
65                 else
66                     ns=IDLE;
67             end
68         endcase
69      end
70
71      //output logic
72      always @(posedge clk or negedge rst_n) begin
73         if (~rst_n) begin
74             rd_add_signal<=0;
75             count_tx<=0;
76             count_rx<=0;
77         end
78         else begin
79             case(cs)
80                 IDLE: begin
81                     MISO<=0;
82                     count_tx<=0;
83                     count_rx<=0;
84                 end
```

```
            CHK_CMD: begin
                count_tx<=0;
                count_rx<=0;
            end
            WRITE: begin
                if(count_rx<=9) begin
                    rx_data[9-count_rx]<=MOSI;
                    count_rx<=count_rx+1;
                end
            end
            READ_ADD: begin
                if(count_rx<=9) begin
                    rx_data[9-count_rx]<= MOSI;
                    count_rx<=count_rx+1;
                    rd_add_signal<=1;
                end
            end
            READ_DATA: begin
                if(count_rx>9) begin
                    if(tx_valid==1&&count_tx<8) begin
                        MISO <= tx_data[7-count_tx];
                        count_tx<=count_tx+1;
                    end
                    else if(tx_valid==0&&count_tx>=8)begin
                        rd_add_signal <= 0;
                    end
                end
                else begin
                    rx_data[9-count_rx]<= MOSI;
                    count_rx<=count_rx+1;
                end
            end
        endcase
    end
end
assign rx_valid =((cs==WRITE||cs==READ_ADD||cs==READ_DATA)&&count_rx>9)?1:0 ;
/*****************************************************************************/
```

```
//RAM
always @(posedge clk or negedge rst_n) begin
    if (~rst_n) begin
        tx_valid<=0;
        wr_addr<=0;
        rd_addr<=0;
    end

    else begin
        tx_valid<=(rx_data[9]&rx_data[8]&rx_valid)?1:0;
        if(rx_valid) begin
            case(rx_data[9:8])
                2'b00:wr_addr<=rx_data[ADDR_SIZE-1:0];
                2'b01:mem[wr_addr]<=rx_data[ADDR_SIZE-1:0];
                2'b10:rd_addr<=rx_data[ADDR_SIZE-1:0];
                2'b11:tx_data<=mem[rd_addr];
            endcase
        end
    end
end
endmodule
```

RAM Assertions:

```
6   import RAM_shared_pkg::*;
7   module RAM_sva (din,clk,rst_n,rx_valid,dout,tx_valid,wr_addr,rd_addr,mem);
8       input [ADDR_SIZE+1:0] din;
9       input clk,rst_n,rx_valid;
10      input [ADDR_SIZE-1:0] dout;
11      input tx_valid;
12      input [ADDR_SIZE-1:0] wr_addr,rd_addr;
13      input [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];
14
15      //Always comp to test asynch rst_n
16      always_comb begin
17          if(rst_n==ACTIVE_RESET) begin
18              rst_wr_addr_assert :    assert final (wr_addr  == INACTIVE);
19              rst_rd_addr_assert :    assert final (rd_addr  == INACTIVE);
20              rst_tx_valid_assert:    assert final (tx_valid == INACTIVE);
21              rst_wr_addr_cover  :    cover final  (wr_addr  == INACTIVE);
22              rst_rd_addr_cover  :    cover final  (rd_addr  == INACTIVE);
23              rst_tx_valid_cover :    cover final  (tx_valid == INACTIVE);
24          end
25      end
26
27      property wr_addr_prop;
28              @(posedge clk) disable iff(!rst_n) (rx_valid && din[ADDR_SIZE+1:ADDR_SIZE] == WRITE_ADD)|=> (wr_addr == $past(din[ADDR_SIZE-1:0]));
29      endproperty
30
31      property wr_data_prop;
32              @(posedge clk) disable iff(!rst_n) (rx_valid && din[ADDR_SIZE+ 1:ADDR_SIZE] == WRITE_DATA) |=> (mem[wr_addr] == $past(din[ADDR_SIZE-1:0]));
33      endproperty
34
35      property rd_addr_prop;
36              @(posedge clk) disable iff(!rst_n) (rx_valid && din[ADDR_SIZE+ 1:ADDR_SIZE] == READ_ADD) |=> (rd_addr == $past(din[ADDR_SIZE-1:0]));
37      endproperty
38
39      property rd_data_prop;
40              @(posedge clk) disable iff(!rst_n) (rx_valid && din[ADDR_SIZE+ 1:ADDR_SIZE] == READ_DATA) |=> (dout == $past(mem[rd_addr]));
41      endproperty
42
43      property tx_valid_prop;
44              @(posedge clk) disable iff(!rst_n) (rx_valid && din[ADDR_SIZE+ 1:ADDR_SIZE] == READ_DATA) |=> (tx_valid == 1);
45      endproperty
46
47  /*************************************************************/
48      wr_addr_assert :assert property(wr_addr_prop);
49      wr_data_assert :assert property(wr_data_prop);
50      rd_addr_assert :assert property(rd_addr_prop);
51      rd_data_assert :assert property(rd_data_prop);
52      tx_valid_assert:assert property(tx_valid_prop);
53  /*************************************************************/
54      wr_addr_cover :cover  property(wr_addr_prop);
55      wr_data_cover :cover  property(wr_data_prop);
56      rd_addr_cover :cover  property(rd_addr_prop);
57      rd_data_cover :cover  property(rd_data_prop);
58      tx_valid_cover:cover  property(tx_valid_prop);
59  endmodule
```

SLAVE Assertions:

```
import wrapper_shared_pkg::*;
module slave_sva (MOSI, SS_n, clk, rst_n, tx_valid,MISO,rx_valid,rx_data,tx_data,cs,rd_add_signal,count_rx,count_tx);

    input MOSI, SS_n, clk, rst_n, tx_valid;
    input [ADDR_SIZE-1:0] tx_data;
    input [ADDR_SIZE+1:0] rx_data;
    input MISO;
    input rx_valid;
    input STATE_e cs;
    input rd_add_signal;
    input integer count_rx,count_tx;

    always_comb begin
        if (rst_n==ACTIVE_RESET) begin
            rst_count_rx_assert      : assert final (count_rx == 0);
            rst_count_tx_assert      : assert final (count_tx == 0);
            rst_rd_add_signal_assert : assert final (rd_add_signal == INACTIVE);
            rst_cs_assert            : assert final (cs == IDLE);
            rst_count_rx_cover       : cover final  (count_rx == 0);
            rst_count_tx_cover       : cover final  (count_tx == 0);
            rst_rd_add_signal_cover  : cover final  (rd_add_signal == INACTIVE);
            rst_cs_cover             : cover final  (cs == IDLE);
        end
    end
    /*This assertinos for the cs*/
    property idel_prop;
        @(posedge clk) disable iff(!rst_n) SS_n |=> cs == IDLE;
    endproperty

    property check_prop;
        @(posedge clk) disable iff(!rst_n) (!SS_n&&cs==IDLE) |=> cs == CHK_CMD;
    endproperty

    property write_prop;
        @(posedge clk) disable iff(!rst_n) (!SS_n&&cs==CHK_CMD&&!MOSI) |=> cs == WRITE;
    endproperty

    property read_add_prop;
        @(posedge clk) disable iff(!rst_n) (!SS_n&&cs==CHK_CMD&&MOSI&&!rd_add_signal) |=> cs == READ_ADD;
    endproperty

    property read_data_prop;
        @(posedge clk) disable iff(!rst_n) (!SS_n&&cs==CHK_CMD&&MOSI&&rd_add_signal) |=> cs == READ_DATA;
    endproperty
    /*This assertions for the rd_add_signal*/
    property rd_flag_ra_pr;
            @(posedge clk) disable iff(!rst_n) ((cs == READ_ADD) && (count_rx == 10)) |=> (rd_add_signal);
    endproperty

    property rd_flag_rd_pr;
            @(posedge clk) disable iff(!rst_n) ((cs == READ_DATA) && (count_tx == 8)) |=> (!rd_add_signal);
    endproperty

/*************************************************************/
```

```
  /*************************************************************/
      idel_assert       :assert property(idel_prop);
      check_assert      :assert property(check_prop);
      write_assert      :assert property(write_prop);
      read_add_assert   :assert property(read_add_prop);
      read_data_assert  :assert property(read_data_prop);
      rd_flag_ra_assert :assert property(rd_flag_ra_pr);
      rd_flag_rd_assert :assert property(rd_flag_rd_pr);
  /*************************************************************/
      idel_cover        :cover  property(idel_prop);
      check_cover       :cover  property(check_prop);
      write_cover       :cover  property(write_prop);
      read_add_cover    :cover  property(read_add_prop);
      read_data_cover   :cover  property(read_data_prop);
      rd_flag_ra_cover  :cover  property(rd_flag_ra_pr);
      rd_flag_rd_cover  :cover  property(rd_flag_rd_pr);

  endmodule
```

Wrapper Assertions:

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

    //Always comp to test asynch rst_n
    always_comb begin
        if(rst_n==ACTIVE_RESET) begin
            rst_tx_valid_assert     :   assert final (tx_valid  == INACTIVE);
            rst_rst_rx_valid_assert :   assert final (rx_valid  == INACTIVE);
            rst_rst_tx_data_assert  :   assert final (tx_data   == INACTIVE);
            rst_rst_rx_data_assert  :   assert final (rx_data   == INACTIVE);
            rst_tx_valid_cover      :   cover final  (tx_valid  == INACTIVE);
            rst_rst_rx_valid_cover  :   cover final  (rx_valid  == INACTIVE);
            rst_rst_tx_data_cover   :   cover final  (tx_data   == INACTIVE);
            rst_rst_rx_data_cover   :   cover final  (rx_data   == INACTIVE);
        end
    end




    property rx_valid_prop;
            @(posedge clk) disable iff(!rst_n) (SS_n) ##1 (!SS_n) [*12] |=> (rx_valid);
    endproperty

    property tx_valid_prop;
            @(posedge clk) disable iff(!rst_n) (SS_n) ##1 (!SS_n) [*12] ##1 ((!SS_n) &&
       ({$past(MOSI, 10), $past(MOSI, 9)} == 2'b11)) |=> (tx_valid);
    endproperty

    property rx_data_prop;
            @(posedge clk) disable iff(!rst_n) (SS_n) ##1 (!SS_n) [*12] |=> (rx_data ==
       {$past(MOSI, 10), $past(MOSI, 9), $past(MOSI, 8), $past(MOSI, 7),
       $past(MOSI, 6), $past(MOSI, 5), $past(MOSI, 4), $past(MOSI, 3),
       $past(MOSI, 2), $past(MOSI)});
    endproperty

/*************************************************************/
    rx_valid_assert :assert property(rx_valid_prop);
    tx_valid_assert :assert property(tx_valid_prop);
    rx_data_assert  :assert property(rx_data_prop);
/*************************************************************/
    rx_valid_cover  :cover  property(rx_valid_prop);
    tx_valid_cover  :cover  property(tx_valid_prop);
    rx_data_cover   :cover  property(rx_data_prop);
endmodule
```

RAM Shared_pkg:

```
package RAM_shared_pkg;
    parameter MEM_DEPTH = 256 ;
    parameter ADDR_SIZE = $clog2(MEM_DEPTH) ;

    parameter WRITE_ADD=0;
    parameter WRITE_DATA=1;
    parameter READ_ADD=2;
    parameter READ_DATA=3;

    parameter RX_VALID_ON_DIST=90;
    parameter RESET_ON_DIST=5;

    parameter ACTIVE=1;
    parameter INACTIVE=0;

    parameter ACTIVE_RESET=0;
    parameter INACTIVE_RESET=1;
endpackage : RAM_shared_pkg
```

SLAVE&Wrapper Shared_pkg:

```
6   package wrapper_shared_pkg;
7       localparam MEM_DEPTH = 256;
8       localparam ADDR_SIZE = $clog2(MEM_DEPTH);
9
10  /*****************************************************************************/
11  //For:Slave & Wrapper
12      typedef enum bit [2:0] {IDLE,CHK_CMD,WRITE,READ_ADD,READ_DATA} STATE_e;
13
14      typedef enum bit [2 : 0] {IDLE_COV, WRITE_ADD_COV, WRITE_DATA_COV, READ_ADD_COV, READ_DATA_COV} STATE_COV_e;
15      STATE_COV_e cs_cov;
16
17      parameter RX_VALID_ON_DIST=90;
18      parameter RESET_ON_DIST=1;
19
20      parameter ACTIVE=1;
21      parameter INACTIVE=0;
22
23      parameter ACTIVE_RESET=0;
24      parameter INACTIVE_RESET=1;
25  endpackage
```

RAM Interface:

```
6   import RAM_shared_pkg::*;
7   interface RAM_if (clk);
8       logic [ADDR_SIZE+1:0] din;
9       logic rst_n,rx_valid;
10      logic [ADDR_SIZE-1:0] dout;
11      logic tx_valid;
12      input clk;
13  endinterface : RAM_if
```

SLAVE Interface:

```
6   import wrapper_shared_pkg::*;
7   interface slave_if(clk);
8       //Input signals
9       logic MOSI, SS_n, rst_n, tx_valid;
10      logic [ADDR_SIZE-1:0] tx_data;
11      input clk;
12      //Output signals
13      logic [ADDR_SIZE+1:0] rx_data,rx_data_ref;
14      logic MISO,MISO_ref;
15      logic rx_valid,rx_valid_ref;
16
17  endinterface : slave_if
```

Wrapper Interface:

```
6   import wrapper_shared_pkg::*;
7   interface wrapper_if (clk);
8       logic MOSI,SS_n,rst_n;
9       logic MISO,MISO_ref;
10      input clk;
11
12  endinterface : wrapper_if
```

Objects: -

RAM Configuration:

```
6   package RAM_config_pkg;
7       import uvm_pkg::*;
8       `include "uvm_macros.svh"
9       class RAM_config extends  uvm_object;
10          `uvm_object_utils(RAM_config)
11
12          virtual RAM_if RAM_vif;
13          uvm_active_passive_enum is_active;
14
15          function  new(string name = "RAM_config");
16              super.new(name);
17          endfunction : new
18
19      endclass : RAM_config
20  endpackage : RAM_config_pkg
```

SLAVE Configuration:

```
6   package slave_config_pkg;
7       import uvm_pkg::*;
8       `include "uvm_macros.svh"
9       class slave_config extends  uvm_object;
10          `uvm_object_utils(slave_config)
11
12          virtual slave_if slave_vif;
13          uvm_active_passive_enum is_active;
14
15          function  new(string name = "slave_config");
16              super.new(name);
17          endfunction : new
18
19      endclass : slave_config
20  endpackage : slave_config_pkg
```

Wrapper Configuration:

```
6   package wrapper_config_pkg;
7       import uvm_pkg::*;
8       `include "uvm_macros.svh"
9       class wrapper_config extends  uvm_object;
10          `uvm_object_utils(wrapper_config)
11
12          virtual wrapper_if wrapper_vif;
13          uvm_active_passive_enum is_active;
14
15          function  new(string name = "wrapper_config");
16              super.new(name);
17          endfunction : new
18
19      endclass : wrapper_config
20  endpackage : wrapper_config_pkg
```

RAM Sequence item:

```
6   package RAM_seq_item_pkg;
7       import RAM_shared_pkg::*;
8       import uvm_pkg::*;
9       `include "uvm_macros.svh"
10
11      class RAM_seq_item extends  uvm_sequence_item;
12          `uvm_object_utils(RAM_seq_item)
13
14          //Randomized properties to generate stimulus
15          rand bit rst_n,rx_valid;
16          rand bit [ADDR_SIZE+1:0] din;
17          //output signals
18          bit [ADDR_SIZE-1:0] dout;
19          bit tx_valid;
20          bit [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];
```

```
        /*Write and Read address array to handle all possible addresses based on ADDR_SIZE*/
        bit [ADDR_SIZE-1:0] wa_arr [MEM_DEPTH];
        bit [ADDR_SIZE-1:0] ra_arr [MEM_DEPTH];
        //Write & Read index for the array
        bit [ADDR_SIZE-1:0] wa_indx,ra_indx;
        //Flags for check the read and write addresses occurred
        bit wr_addr_occur, rd_addr_occur;


        function  new(string name ="RAM_seq_item");
            super.new(name);
            //loop to save all possible addresses in the array
            for (int i = 0; i < MEM_DEPTH; i++) begin
                wa_arr [i] = i;
                ra_arr [i] = i;
            end
            wa_arr.shuffle();
            ra_arr.shuffle();
        endfunction : new

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

```
/*****************************************Constraints*************************************************/

    //Constraint to randomize rst_n with a higher probability of being active
    constraint rst_prob {rst_n dist {ACTIVE_RESET:/RESET_ON_DIST,INACTIVE_RESET:/100-RESET_ON_DIST};}

    // Constraint to randomize rx_valid with a high probability of being valid
    constraint rx_valid_prob {rx_valid dist {ACTIVE:/RX_VALID_ON_DIST,INACTIVE:/100-RX_VALID_ON_DIST};}

    // Constraints on din to ensure valid address mapping
    constraint din_prob {
        din [ADDR_SIZE + 1] dist {ACTIVE:/40,INACTIVE:/ 60}; // Randomize high bit for read/write operation
        din [ADDR_SIZE]     dist {ACTIVE:/40,INACTIVE:/ 60}; // Randomize low bit for operation validity

        //Ensure proper handling of write and read operations based on flags
        if (din[ADDR_SIZE + 1] == 0) {
            if (!wr_addr_occur) //if the write address not occurr the bit 8 will be 0
                din[ADDR_SIZE] == 0;
        }
        else {
            if (!rd_addr_occur) //if the read address not occurr the bit 8 will be 0
                din[ADDR_SIZE] == 0;
        }

        //if the operation is write or read address we will send the address from the arrays
        if (din[ADDR_SIZE+1:ADDR_SIZE]==WRITE_ADD)
            din[ADDR_SIZE-1:0] == wa_arr[wa_indx];
        else if (din[ADDR_SIZE+1:ADDR_SIZE]==READ_ADD)
            din[ADDR_SIZE-1:0] == ra_arr[ra_indx];
    }


    //Function to handle updates and shuffling post-randomization
    function void post_randomize();
        if(wa_indx==MEM_DEPTH-1) //Check if all write addresses have been use
            wa_arr.shuffle();
        else if(din[ADDR_SIZE+1:ADDR_SIZE]==WRITE_ADD)
            wa_indx++;//Increment if the operation is write address

        if(ra_indx==MEM_DEPTH-1) //Check if all read addresses have been use
            ra_arr.shuffle();
        else if(din[ADDR_SIZE+1:ADDR_SIZE]==READ_ADD)
            ra_indx++;//Increment if the operation is read address

        //update the flags
        if (din[ADDR_SIZE + 1 : ADDR_SIZE] == WRITE_ADD)
            wr_addr_occur = 1;
        else if (din[ADDR_SIZE + 1 : ADDR_SIZE] == READ_ADD)
            rd_addr_occur = 1;

    endfunction : post_randomize



    endclass : RAM_seq_item
endpackage : RAM_seq_item_pkg
```

SLAVE Sequence item:

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

    class slave_seq_item extends  uvm_sequence_item;
        `uvm_object_utils(slave_seq_item)

        //Randomized properties to generate stimulus
        rand bit MOSI, SS_n, clk, rst_n, tx_valid;
        rand bit [ADDR_SIZE-1:0] tx_data;
        //output signals
        logic [ADDR_SIZE+1:0] rx_data,rx_data_ref;
        logic MISO,rx_valid,MISO_ref,rx_valid_ref;

        //Array to hold MOSI data ,The size is 21 bit for the longest way (READ_DATA) [2+2+8 dummy+8]
        rand bit MOSI_arr [20:0];
        //counter for the mosi_arr and the size is 5
        bit [4:0] count;
        //this is Read/Write flag (1 = read, 0 = write)
        bit rd_flag;

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

        function string convert2string();
            return $sformatf("%s rst_n = %0b, SS_n = %0b, MOSI = %0b, tx_valid = %0b, tx_data = %0h, rx_valid = %0b, rx_data = %0h, MISO = %0b",
             super.convert2string(), rst_n, SS_n, MOSI, tx_valid, tx_data, rx_valid, rx_data, MISO);
        endfunction

    /*****************************************************************************************************/
    /*****************************************Constraints*************************************************/

        //Constraint to randomize rst_n with a higher probability of being active
        constraint rst_prob {rst_n dist {ACTIVE_RESET:/RESET_ON_DIST,INACTIVE_RESET:/100-RESET_ON_DIST};}

        constraint SS_N_prob {
            if(MOSI_arr[2]&&MOSI_arr[3]){
                if(count==20) //when the state is read data and we finish reading the SS_n will equal 1 to end the communication
                    SS_n==1;
                else
                    SS_n==0;
            }
            else {
                if(count==12) //when we finish states the SS_n will equal 1 to end the communication
                    SS_n==1;
                else
                    SS_n==0;
            }
        }

        constraint tx_valid_c {
            //if the state is read data and we finish send the dummy bits the tx_valid should be active
            if((MOSI_arr[2]&&MOSI_arr[3])&&(count>11))
                tx_valid==1;
            else
                tx_valid==0;
        }

        constraint MOSI_arr_prop {
            MOSI_arr[1] dist {0:/50,1:/50};
            MOSI_arr[2] == MOSI_arr[1];
            if(MOSI_arr[2]){//if the bit 9 equal 1 [Read operation]
                if(rd_flag)//if we read address
                    MOSI_arr[3]==1;//We will Read date
                else
                    MOSI_arr[3]==0;//We will Read address
            }
            else//[Write operation]
                MOSI_arr[3] dist {0:/60,1:/40};
        }

        constraint MOSI_prop {
            MOSI==MOSI_arr[count];//MOSI reflects the current bit of MOSI_arr based on count
        }

        function void pre_randomize();
            if(count==0)//Enable the randomization of the array at the idle state
                MOSI_arr.rand_mode(1);
            else
                MOSI_arr.rand_mode(0);

            if((MOSI_arr[2]&&MOSI_arr[3])&&(count==20))//Enable the randomization of the tx_data at the end of the read data
                tx_data.rand_mode(1);
            else
                tx_data.rand_mode(0);

            /*befor the randomization we shold know the state of the rd_flag*/
            if(rst_n==ACTIVE_RESET)
                rd_flag=0;
            else if(count==0) begin//At the beginning when the count equal zero we will check the rd flag
                if(MOSI_arr[2]&&MOSI_arr[3])//READ_DATA
                    rd_flag=0;
                else if (MOSI_arr[2]&&!MOSI_arr[3])//READ_ADD
                    rd_flag=1;
            end

        endfunction : pre_randomize
```

```
    function void post_randomize();
        //Update the counter after every randomization
        if(SS_n||rst_n==ACTIVE_RESET)
            count=0;
        else
            count++;


    endfunction : post_randomize

    endclass : slave_seq_item
endpackage : slave_seq_item_pkg
```

Wrapper Sequence item:

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

    class wrapper_seq_item extends  uvm_sequence_item;
        `uvm_object_utils(wrapper_seq_item)

        //Randomized properties to generate stimulus
        rand bit MOSI, SS_n, rst_n;
        //output signals
        logic MISO,MISO_ref;

        //Array to hold MOSI data ,The size is 21 bit for the longest way (READ_DATA)
        rand bit MOSI_arr [(2 * ADDR_SIZE) + 6];
        //counter for the mosi_arr and the size is 5
        bit [$clog2((2 * ADDR_SIZE) + 5) - 1 : 0] count;
        //this is Read/Write flag (1 = read, 0 = write)
        bit rd_flag;

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

        function string convert2string();
            return $sformatf("%s rst_n = %0b, SS_n = %0b, MOSI = %0b, MISO = %0b",
             super.convert2string(), rst_n, SS_n, MOSI, MISO);
        endfunction
```

```
/*****************************************Constraints*****************************************************/

    //Constraint to randomize rst_n with a higher probability of being active
    constraint rst_prob {rst_n dist {ACTIVE_RESET:/RESET_ON_DIST,INACTIVE_RESET:/100-RESET_ON_DIST};}

    constraint SS_N_prob {
        if(MOSI_arr[2]&&MOSI_arr[3]){
            if(count==(2 * ADDR_SIZE) + 5) //when the state is read data and we finish reading the SS_n will equal 1 to end the communication
                SS_n==1;
            else
                SS_n==0;
        }
        else {
            if(count==12) //when we finish states the SS_n will equal 1 to end the communication
                SS_n==1;
            else
                SS_n==0;
        }
    }

    constraint MOSI_arr_prop {
        MOSI_arr[1] dist {0:/50,1:/50};
        MOSI_arr[2] == MOSI_arr[1];
        if(MOSI_arr[2]){//if the bit 9 equal 1 [Read operation]
            if(rd_flag)//if we read address
                MOSI_arr[3]==1;//We will Read date
            else
                MOSI_arr[3]==0;//We will Read address
        }
        else//[Write operation]
            MOSI_arr[3] dist {0:/60,1:/40};
    }

    constraint MOSI_prop {
        MOSI==MOSI_arr[count];//MOSI reflects the current bit of MOSI_arr based on count
    }
```

```
    function void pre_randomize();
        if(count==0)//Enable the randomization of the array at the idle state
            MOSI_arr.rand_mode(1);
        else
            MOSI_arr.rand_mode(0);



        /*befor the randomization we shold know the state of the rd_flag*/
        if(rst_n==ACTIVE_RESET)
            rd_flag=0;
        else if(count==0) begin//At the beginning when the count equal zero we will check the rd flag
            if(MOSI_arr[2]&&MOSI_arr[3])//READ_DATA
                rd_flag=0;
            else if (MOSI_arr[2]&&!MOSI_arr[3])//READ_ADD
                rd_flag=1;
        end

    endfunction : pre_randomize

    function void post_randomize();
        //Update the counter after every randomization
        if(SS_n||rst_n==ACTIVE_RESET)
            count=0;
        else
            count++;

        //Update the variable cs_cov for the coverage collector after the state comes true
        if(SS_n||rst_n==ACTIVE_RESET)
            cs_cov=IDLE_COV;
        else if (count==12)begin//for the three cases that need 12 clk cycle
            if(!MOSI_arr[2]&&!MOSI_arr[3])
                cs_cov=WRITE_ADD_COV;
            else if(!MOSI_arr[2]&&MOSI_arr[3])
                cs_cov=WRITE_DATA_COV;
            else if(MOSI_arr[2]&&!MOSI_arr[3])
                cs_cov=READ_ADD_COV;
        end
        else if(count==20)begin
            if(MOSI_arr[2]&&MOSI_arr[3])
                cs_cov=READ_DATA_COV;
        end

    endfunction : post_randomize

    endclass : wrapper_seq_item
endpackage : wrapper_seq_item_pkg
```

RAM Sequences:

```
package RAM_seq_pkg;
    import RAM_shared_pkg::*;
    import RAM_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
/*****************************************************************************************************************************************/
    class RAM_rst_assert extends  uvm_sequence #(RAM_seq_item);
        `uvm_object_utils(RAM_rst_assert)

        RAM_seq_item seq_item;

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

        task pre_body;
            seq_item = RAM_seq_item::type_id::create("seq_item");
        endtask

        task body();
            start_item(seq_item);
                seq_item.rst_n=ACTIVE_RESET;
                seq_item.rx_valid=INACTIVE;
                seq_item.din=INACTIVE;
            finish_item(seq_item);
        endtask : body
    endclass : RAM_rst_assert
/*****************************************************************************************************************************************/
    class RAM_write_only extends  uvm_sequence #(RAM_seq_item);
        `uvm_object_utils(RAM_write_only)

        RAM_seq_item seq_item;

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

        task pre_body;
            seq_item = RAM_seq_item::type_id::create("seq_item");
        endtask

        task body();
            repeat(10000) begin
                start_item(seq_item);
                    assert(seq_item.randomize() with {seq_item.din[ADDR_SIZE+1:ADDR_SIZE] inside {2'b00, 2'b01};});
                finish_item(seq_item);
            end
        endtask : body
    endclass : RAM_write_only

/*****************************************************************************************************************************************/
```

```
/*****************************************************************************************************************************************/
    class RAM_read_only extends  uvm_sequence #(RAM_seq_item);
        `uvm_object_utils(RAM_read_only)

        RAM_seq_item seq_item;

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

        task pre_body;
            seq_item = RAM_seq_item::type_id::create("seq_item");
        endtask

        task body();
            repeat(10000) begin
                start_item(seq_item);
                    assert(seq_item.randomize() with {seq_item.din[ADDR_SIZE+1:ADDR_SIZE] inside {2'b10, 2'b11};});
                finish_item(seq_item);
            end
        endtask : body
    endclass : RAM_read_only
/*****************************************************************************************************************************************/
    class RAM_write_read extends  uvm_sequence #(RAM_seq_item);
        `uvm_object_utils(RAM_write_read)

        RAM_seq_item seq_item;

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

        task pre_body;
            seq_item = RAM_seq_item::type_id::create("seq_item");
        endtask

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

SLAVE Sequences:

```
package slave_seq_pkg;
    import slave_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
/*****************************************************************************/
    class slave_rst_assert extends  uvm_sequence #(slave_seq_item);
        `uvm_object_utils(slave_rst_assert)

        slave_seq_item seq_item;

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

        task pre_body;
            seq_item = slave_seq_item::type_id::create("seq_item");
        endtask :pre_body

        task body();
            start_item(seq_item);
                seq_item.rst_n    = 0;
                seq_item.SS_n     = 0;
                seq_item.MOSI     = 0;
                seq_item.tx_valid = 0;
                seq_item.tx_data  = 0;
            finish_item(seq_item);

        endtask : body
    endclass : slave_rst_assert
/*****************************************************************************/
    class slave_main_seq extends  uvm_sequence #(slave_seq_item);
        `uvm_object_utils(slave_main_seq)

        slave_seq_item seq_item;

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

        task pre_body;
            seq_item = slave_seq_item::type_id::create("seq_item");
        endtask :pre_body

        task body();
            repeat(20000) begin
                start_item(seq_item);
                    assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask : body
    endclass : slave_main_seq
/*****************************************************************************/
```

```
/*****************************************************************************/
    class slave_rand_seq extends  uvm_sequence #(slave_seq_item);
        `uvm_object_utils(slave_rand_seq)

        slave_seq_item seq_item;

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

        task pre_body;
            seq_item = slave_seq_item::type_id::create("seq_item");
        endtask :pre_body

        task body();

            repeat(10000) begin
                start_item(seq_item);
                    seq_item.constraint_mode(0);
                    assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask : body
    endclass : slave_rand_seq
/*****************************************************************************/
endpackage : slave_seq_pkg
```

Wrapper Sequences:

```
package wrapper_sequence_pkg;
    import wrapper_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
/*****************************************************************************/
    class wrapper_rst_assert extends uvm_sequence #(wrapper_seq_item);
        `uvm_object_utils(wrapper_rst_assert);

        wrapper_seq_item seq_item;

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

        task pre_body;
            seq_item = wrapper_seq_item::type_id::create("seq_item");
        endtask : pre_body

        task body;
            start_item(seq_item);
                seq_item.rst_n = 0;
                seq_item.SS_n  = 0;
                seq_item.MOSI  = 0;
            finish_item(seq_item);

        endtask : body
    endclass : wrapper_rst_assert
/*****************************************************************************/
    class wrapper_main_seq extends  uvm_sequence #(wrapper_seq_item);
        `uvm_object_utils(wrapper_main_seq)

        wrapper_seq_item seq_item;

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

        task pre_body;
            seq_item = wrapper_seq_item::type_id::create("seq_item");
        endtask :pre_body

        task body();
            repeat(50000) begin
                start_item(seq_item);
                    assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask : body
    endclass : wrapper_main_seq
/*****************************************************************************/
endpackage
```

RAM Environment: -

RAM Driver:

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

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

        RAM_seq_item stim_seq_item;
        virtual RAM_if RAM_vif;

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

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

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

                seq_item_port.item_done();
                `uvm_info("run_phase", stim_seq_item.convert2string(), UVM_HIGH)

            end
        endtask : run_phase

    endclass : RAM_driver
endpackage : RAM_driver_pkg
```

RAM Monitor:

```
package RAM_monitor_pkg;
    import RAM_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class RAM_monitor extends  uvm_monitor;
        `uvm_component_utils(RAM_monitor)

        RAM_seq_item seq_item;
        virtual RAM_if RAM_vif;
        uvm_analysis_port #(RAM_seq_item) mon_ap;

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

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                seq_item=RAM_seq_item::type_id::create("seq_item");
                @(negedge RAM_vif.clk);
                //input
                seq_item.rst_n=RAM_vif.rst_n;
                seq_item.din=RAM_vif.din;
                seq_item.rx_valid=RAM_vif.rx_valid;
                //output
                seq_item.tx_valid=RAM_vif.tx_valid;
                seq_item.dout=RAM_vif.dout;
                //seq_item.dout_ref = RAM_vif.dout_ref;
                //seq_item.tx_valid_ref=RAM_vif.tx_valid_ref;
                mon_ap.write(seq_item);
                `uvm_info("run_phase",seq_item.convert2string(),UVM_HIGH);
            end
        endtask : run_phase

    endclass : RAM_monitor
endpackage : RAM_monitor_pkg
```

RAM Sequencer:

```
package RAM_sequencer_pkg;
    import RAM_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class RAM_sequencer extends uvm_sequencer #(RAM_seq_item);
        `uvm_component_utils(RAM_sequencer)

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

    endclass : RAM_sequencer
endpackage : RAM_sequencer_pkg
```

RAM Agent:

```
package RAM_agent_pkg;
    import RAM_seq_item_pkg::*;
    import RAM_driver_pkg::*;
    import RAM_monitor_pkg::*;
    import RAM_sequencer_pkg::*;
    import RAM_config_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class RAM_agent extends  uvm_agent;
        `uvm_component_utils(RAM_agent)

        RAM_driver driver;
        RAM_monitor monitor;
        RAM_sequencer sqr;
        RAM_config RAM_cfg;
        uvm_analysis_port #(RAM_seq_item) agt_ap;

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

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

            if(!uvm_config_db#(RAM_config)::get(this,"","CFG",RAM_cfg))
            `uvm_fatal("build_phase","Agent - unable to get configuration object");

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

        endfunction : build_phase

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            if(RAM_cfg.is_active==UVM_ACTIVE) begin
                driver.RAM_vif=RAM_cfg.RAM_vif;
                driver.seq_item_port.connect(sqr.seq_item_export);
            end
            monitor.RAM_vif=RAM_cfg.RAM_vif;
            monitor.mon_ap.connect(agt_ap);
        endfunction : connect_phase

    endclass : RAM_agent
endpackage : RAM_agent_pkg
```

RAM Scoreboard:

```
package RAM_scoreboard_pkg;
    import RAM_shared_pkg::*;
    import RAM_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class RAM_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(RAM_scoreboard)

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

        bit tx_valid_ref;
        bit [ADDR_SIZE - 1 : 0] dout_ref, wr_addr, rd_addr;
        bit [ADDR_SIZE - 1 : 0] mem_test [MEM_DEPTH];
        // We use rx_valid_reg and din_reg because the slave sends the values, and in the next clock cycle, the RAM takes these values
        bit rx_valid_reg;
        bit [ADDR_SIZE + 1 : 0] din_reg;
        //counters
        int error_count = 0;
        int correct_count = 0;

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

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

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            $readmemh("RAM_data.dat", mem_test, 0, 255);
            forever begin
                sb_fifo.get(seq_item_sb);
                ref_model(seq_item_sb);
                if ((seq_item_sb.dout !== dout_ref) || (seq_item_sb.tx_valid !== tx_valid_ref)) begin
                    `uvm_error("run_phase", $sformatf("comparsion failed, Transaction received by the DUT:%s  While the reference dout: dout_ref = %0h, tx_valid_ref = %0b",
                    seq_item_sb.convert2string(), dout_ref, tx_valid_ref))
                    error_count++;
                end
                else begin
                    `uvm_info("run_phase", $sformatf("Correct RAM Transaction: %s", seq_item_sb.convert2string()), UVM_HIGH)
                    correct_count++;
                end
            end
        endtask : run_phase
```

```
        task ref_model(RAM_seq_item seq_item_chk);
            if (!seq_item_chk.rst_n) begin
                wr_addr = 0;
                rd_addr = 0;
                dout_ref = 0;
                din_reg  = 0;
                    rx_valid_reg = 0;
            end
            else begin
                    din_reg  <= seq_item_sb.din;
                    rx_valid_reg <= seq_item_sb.rx_valid;
            end
            if (rx_valid_reg && seq_item_chk.rst_n) begin
                case (din_reg [ADDR_SIZE + 1 : ADDR_SIZE])
                    2'b00: wr_addr = din_reg [ADDR_SIZE - 1 : 0];
                    2'b01: mem_test [wr_addr] = din_reg [ADDR_SIZE - 1 : 0];
                    2'b10: rd_addr = din_reg [ADDR_SIZE - 1 : 0];
                    2'b11: dout_ref = mem_test [rd_addr];
                endcase
            end

            if (!seq_item_chk.rst_n)
                tx_valid_ref = 0;
            else if (rx_valid_reg)
                if (din_reg [ADDR_SIZE + 1 : ADDR_SIZE] == 2'b11)
                    tx_valid_ref = 1;
                else
                    tx_valid_ref = 0;
        endtask : ref_model

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

    endclass : RAM_scoreboard
endpackage : RAM_scoreboard_pkg
```

RAM Coverage Collector:

```
package RAM_coverage_pkg;
    import RAM_seq_item_pkg::*;
    import RAM_shared_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class RAM_coverage extends  uvm_component;
        `uvm_component_utils(RAM_coverage)

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

        covergroup covgrp;
            rst_cp:coverpoint seq_item_cov.rst_n {
                bins rst_active  = {ACTIVE_RESET};
                bins rst_inactive= {INACTIVE_RESET};
            }
            rx_valid_cp:coverpoint seq_item_cov.rx_valid{
                bins rx_active  = {ACTIVE};
                bins rx_inactive= {INACTIVE};
            }
            din_addr_cp:coverpoint seq_item_cov.din[ADDR_SIZE-1:0];
            din_option_cp:coverpoint seq_item_cov.din[ADDR_SIZE+1:ADDR_SIZE]{
                bins Write_add  = {WRITE_ADD};
                bins Write_data = {WRITE_DATA};
                bins Read_add   = {READ_ADD};
                bins Read_data  = {READ_DATA};
            }

            wr_addr_cr: cross rx_valid_cp, din_option_cp iff (seq_item_cov.rst_n==INACTIVE_RESET) {
                option.cross_auto_bin_max = 0;
                bins wr_addr_occur = binsof(rx_valid_cp) intersect {ACTIVE} && binsof(din_option_cp) intersect {WRITE_ADD};
            }
            wr_data_cr: cross rx_valid_cp, din_option_cp iff (seq_item_cov.rst_n==INACTIVE_RESET) {
                option.cross_auto_bin_max = 0;
                bins wr_data_occur = binsof(rx_valid_cp) intersect {ACTIVE} && binsof(din_option_cp) intersect {WRITE_DATA};
            }
            rd_addr_cr: cross rx_valid_cp, din_option_cp iff (seq_item_cov.rst_n==INACTIVE_RESET) {
                option.cross_auto_bin_max = 0;
                bins rd_addr_occur = binsof(rx_valid_cp) intersect {ACTIVE} && binsof(din_option_cp) intersect {READ_ADD};
            }
            rd_data_cr: cross rx_valid_cp, din_option_cp iff (seq_item_cov.rst_n==INACTIVE_RESET) {
                option.cross_auto_bin_max = 0;
                bins rd_data_occur = binsof(rx_valid_cp) intersect {ACTIVE} && binsof(din_option_cp) intersect {READ_DATA};
            }
        endgroup : covgrp
```

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

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

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(seq_item_cov);
                covgrp.sample();
            end
        endtask : run_phase

    endclass : RAM_coverage
endpackage : RAM_coverage_pkg
```

RAM Environment:

```
package RAM_env_pkg;
    import RAM_scoreboard_pkg::*;
    import RAM_coverage_pkg::*;
    import RAM_agent_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class RAM_env extends  uvm_env;
        `uvm_component_utils(RAM_env)

        RAM_agent agt;
        RAM_scoreboard sb;
        RAM_coverage cov;

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

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

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

    endclass : RAM_env

endpackage : RAM_env_pkg
```

SLAVE Environment: -

SLAVE Driver:

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

    class slave_driver extends  uvm_driver #(slave_seq_item);
        `uvm_component_utils(slave_driver);

        slave_seq_item stim_seq_item;
        virtual slave_if slave_vif;

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

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

                slave_vif.rst_n    = stim_seq_item.rst_n;
                slave_vif.SS_n     = stim_seq_item.SS_n;
                slave_vif.tx_valid = stim_seq_item.tx_valid;
                slave_vif.tx_data  = stim_seq_item.tx_data;
                slave_vif.MOSI     = stim_seq_item.MOSI;
                @(negedge slave_vif.clk);
                seq_item_port.item_done();
                `uvm_info("run_phase", stim_seq_item.convert2string(), UVM_HIGH);
            end
        endtask : run_phase

    endclass : slave_driver
endpackage : slave_driver_pkg
```

SLAVE Monitor:

```
package slave_monitor_pkg;
    import slave_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class slave_monitor extends uvm_monitor;
        `uvm_component_utils(slave_monitor)

        slave_seq_item seq_item;
        virtual slave_if slave_vif;
        uvm_analysis_port #(slave_seq_item) mon_ap;

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

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                seq_item = slave_seq_item::type_id::create("seq_item");
                @(negedge slave_vif.clk);
                //Input
                seq_item.rst_n    = slave_vif.rst_n;
                seq_item.SS_n     = slave_vif.SS_n;
                seq_item.MOSI     = slave_vif.MOSI;
                seq_item.tx_valid = slave_vif.tx_valid;
                seq_item.tx_data  = slave_vif.tx_data;
                //Output
                seq_item.rx_valid     = slave_vif.rx_valid;
                seq_item.rx_data      = slave_vif.rx_data;
                seq_item.MISO         = slave_vif.MISO;
                seq_item.rx_valid_ref = slave_vif.rx_valid_ref;
                seq_item.rx_data_ref  = slave_vif.rx_data_ref;
                seq_item.MISO_ref     = slave_vif.MISO_ref;
                mon_ap.write(seq_item);
                `uvm_info("run_phase", seq_item.convert2string(), UVM_HIGH)
            end
        endtask : run_phase

    endclass : slave_monitor
endpackage : slave_monitor_pkg
```

SLAVE Sequencer:

```
package slave_sequencer_pkg;
    import slave_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class slave_sequencer extends uvm_sequencer #(slave_seq_item);
        `uvm_component_utils(slave_sequencer)

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

    endclass : slave_sequencer
endpackage : slave_sequencer_pkg
```

SLAVE Agent:

```
package slave_agent_pkg;
    import slave_seq_item_pkg::*;
    import slave_driver_pkg::*;
    import slave_monitor_pkg::*;
    import slave_sequencer_pkg::*;
    import slave_config_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class slave_agent extends  uvm_agent;
        `uvm_component_utils(slave_agent)

        slave_driver driver;
        slave_monitor monitor;
        slave_sequencer sqr;
        slave_config slave_cfg;
        uvm_analysis_port #(slave_seq_item) agt_ap;

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

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

            if(!uvm_config_db#(slave_config)::get(this,"","CFG",slave_cfg))
            `uvm_fatal("build_phase","Agent - unable to get configuration object");

            if(slave_cfg.is_active==UVM_ACTIVE) begin
                driver  =slave_driver::type_id::create("driver",this);
                sqr     =slave_sequencer::type_id::create("sqr",this);
            end
            monitor =slave_monitor::type_id::create("monitor",this);
            agt_ap  =new("agt_ap",this);

        endfunction : build_phase

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            if(slave_cfg.is_active==UVM_ACTIVE) begin
                driver.slave_vif=slave_cfg.slave_vif;
                driver.seq_item_port.connect(sqr.seq_item_export);
            end
            monitor.slave_vif=slave_cfg.slave_vif;
            monitor.mon_ap.connect(agt_ap);
        endfunction : connect_phase

    endclass : slave_agent
endpackage : slave_agent_pkg
```

SLAVE Scoreboard:

```
package slave_scoreboard_pkg;
    import slave_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class slave_scoreboard extends  uvm_scoreboard;
        `uvm_component_utils(slave_scoreboard)

        uvm_analysis_export #(slave_seq_item) sb_export;
        uvm_tlm_analysis_fifo #(slave_seq_item) sb_fifo;
        slave_seq_item seq_item_sb;

        //Counters for check the result
        int rx_error_count = 0;
        int rx_correct_count = 0;
        int MISO_error_count =0;
        int MISO_correct_count = 0;
```

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

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

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(seq_item_sb);
                if ((seq_item_sb.rx_data!==seq_item_sb.rx_data_ref)||(seq_item_sb.rx_valid!==seq_item_sb.rx_valid_ref)) begin
                    `uvm_error("run_phase", $sformatf("comparsion RX failed, DUT:%s  While :rx_data_ref = %0h, rx_valid_ref = %0b",
                     seq_item_sb.convert2string(), seq_item_sb.rx_data_ref, seq_item_sb.rx_valid_ref))
                    rx_error_count++;
                end
                else begin
                    `uvm_info("run_phase", $sformatf("Correct RX: %s", seq_item_sb.convert2string()), UVM_HIGH)
                    rx_correct_count++;
                end

                if (seq_item_sb.MISO!==seq_item_sb.MISO_ref) begin
                    `uvm_error("run_phase", $sformatf("comparsion MISO failed, DUT:%s  While :MISO_ref = %0b",
                     seq_item_sb.convert2string(), seq_item_sb.MISO_ref))
                    MISO_error_count++;
                end
                else begin
                    `uvm_info("run_phase", $sformatf("Correct RX: %s", seq_item_sb.convert2string()), UVM_HIGH)
                    MISO_correct_count++;
                end
            end
        endtask : run_phase

        function void report_phase (uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("report_phase", $sformatf("Total successful[RX]: %0d", rx_correct_count), UVM_LOW)
            `uvm_info("report_phase", $sformatf("Total failed[RX]: %0d", rx_error_count), UVM_LOW)
            `uvm_info("report_phase", $sformatf("Total successful[MOSO]: %0d", MISO_correct_count), UVM_LOW)
            `uvm_info("report_phase", $sformatf("Total failed[MOSO]: %0d", MISO_error_count), UVM_LOW)
        endfunction : report_phase

    endclass : slave_scoreboard
endpackage : slave_scoreboard_pkg
```

SLAVE Coverage Collector:

```
package slave_coverage_pkg;
    import slave_seq_item_pkg::*;
    import wrapper_shared_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class slave_coverage extends  uvm_scoreboard;
        `uvm_component_utils(slave_coverage)

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

        covergroup covgrp;
            rst_cp:coverpoint seq_item_cov.rst_n{
                bins active   = {0};
                bins inactive = {1};
            }

            cs_cov_cp: coverpoint cs_cov {
                bins wr_add  = {WRITE_ADD_COV};
                bins wr_data = {WRITE_DATA_COV};
                bins rd_add  = {READ_ADD_COV};
                bins rd_data = {READ_DATA_COV};
            }
        endgroup : covgrp

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

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

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(seq_item_cov);
                covgrp.sample();
            end
        endtask : run_phase


    endclass : slave_coverage
endpackage : slave_coverage_pkg
```

SLAVE Environment:

```
package slave_env_pkg;
    import slave_scoreboard_pkg::*;
    import slave_coverage_pkg::*;
    import slave_agent_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class slave_env extends  uvm_env;
        `uvm_component_utils(slave_env)

        slave_agent agt;
        slave_scoreboard sb;
        slave_coverage cov;

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            agt=slave_agent::type_id::create("agt",this);
            sb=slave_scoreboard::type_id::create("sb",this);
            cov=slave_coverage::type_id::create("cov",this);
        endfunction : build_phase

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

    endclass : slave_env

endpackage : slave_env_pkg
```

Wrapper Environment: -

Wrapper Driver:

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

    class wrapper_driver extends  uvm_driver #(wrapper_seq_item);
        `uvm_component_utils(wrapper_driver);

        wrapper_seq_item stim_seq_item;
        virtual wrapper_if wrapper_vif;

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                stim_seq_item = wrapper_seq_item::type_id::create("stim_seq_item");
                seq_item_port.get_next_item(stim_seq_item);
                wrapper_vif.rst_n   = stim_seq_item.rst_n;
                wrapper_vif.MOSI    = stim_seq_item.MOSI;
                wrapper_vif.SS_n = stim_seq_item.SS_n;
                @(negedge wrapper_vif.clk);
                seq_item_port.item_done();
                `uvm_info("run_phase", stim_seq_item.convert2string(), UVM_HIGH);
            end
        endtask : run_phase

    endclass : wrapper_driver
endpackage : wrapper_driver_pkg
```

Wrapper Monitor:

```
package wrapper_monitor_pkg;
    import wrapper_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class wrapper_monitor extends uvm_monitor;
        `uvm_component_utils(wrapper_monitor)

        wrapper_seq_item seq_item;
        virtual wrapper_if wrapper_vif;
        uvm_analysis_port #(wrapper_seq_item) mon_ap;

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

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                seq_item = wrapper_seq_item::type_id::create("seq_item");
                @(negedge wrapper_vif.clk);
                //Input
                seq_item.rst_n    = wrapper_vif.rst_n;
                seq_item.SS_n     = wrapper_vif.SS_n;
                seq_item.MOSI     = wrapper_vif.MOSI;
                //Output
                seq_item.MISO     = wrapper_vif.MISO;
                seq_item.MISO_ref = wrapper_vif.MISO_ref;
                mon_ap.write(seq_item);
                `uvm_info("run_phase", seq_item.convert2string(), UVM_HIGH)
            end
        endtask : run_phase

    endclass : wrapper_monitor
endpackage : wrapper_monitor_pkg
```

Wrapper Sequencer:

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

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

    endclass : wrapper_sequencer
endpackage : wrapper_sequencer_pkg
```

Wrapper Agent:

```
package wrapper_agent_pkg;
    import wrapper_seq_item_pkg::*;
    import wrapper_driver_pkg::*;
    import wrapper_monitor_pkg::*;
    import wrapper_sequencer_pkg::*;
    import wrapper_config_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class wrapper_agent extends  uvm_agent;
        `uvm_component_utils(wrapper_agent)

        wrapper_driver driver;
        wrapper_monitor monitor;
        wrapper_sequencer sqr;
        wrapper_config wrapper_cfg;
        uvm_analysis_port #(wrapper_seq_item) agt_ap;

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

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

            if(!uvm_config_db#(wrapper_config)::get(this,"","CFG",wrapper_cfg))
            `uvm_fatal("build_phase","Agent - unable to get configuration object");

            if(wrapper_cfg.is_active==UVM_ACTIVE) begin
                driver  =wrapper_driver::type_id::create("driver",this);
                sqr     =wrapper_sequencer::type_id::create("sqr",this);
            end
            monitor =wrapper_monitor::type_id::create("monitor",this);
            agt_ap  =new("agt_ap",this);

        endfunction : build_phase

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            if(wrapper_cfg.is_active==UVM_ACTIVE) begin
                driver.wrapper_vif=wrapper_cfg.wrapper_vif;
                driver.seq_item_port.connect(sqr.seq_item_export);
            end
            monitor.wrapper_vif=wrapper_cfg.wrapper_vif;
            monitor.mon_ap.connect(agt_ap);
        endfunction : connect_phase

    endclass : wrapper_agent
endpackage : wrapper_agent_pkg
```

Wrapper Scoreboard:

```
package wrapper_scoreboard_pkg;
    import wrapper_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class wrapper_scoreboard extends  uvm_scoreboard;
        `uvm_component_utils(wrapper_scoreboard)

        uvm_analysis_export #(wrapper_seq_item) sb_export;
        uvm_tlm_analysis_fifo #(wrapper_seq_item) sb_fifo;
        wrapper_seq_item seq_item_sb;

        //Counters for check the result
        int MISO_error_count =0;
        int MISO_correct_count = 0;

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

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

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            //$readmemh("RAM_data.dat", mem_ref, 0, 255);
            forever begin
                sb_fifo.get(seq_item_sb);
                if (seq_item_sb.MISO!==seq_item_sb.MISO_ref) begin
                    `uvm_error("run_phase", $sformatf("comparsion MISO failed, DUT:%s  While :MISO_ref = %0b",
                     seq_item_sb.convert2string(), seq_item_sb.MISO_ref))
                    MISO_error_count++;
                end
                else begin
                    `uvm_info("run_phase", $sformatf("Correct RX: %s", seq_item_sb.convert2string()), UVM_HIGH)
                    MISO_correct_count++;
                end
            end
        endtask : run_phase
```

```
53          function void report_phase (uvm_phase phase);
54              super.report_phase(phase);
55              `uvm_info("report_phase", $sformatf("Total successful[MOSO]: %0d", MISO_correct_count), UVM_LOW)
56              `uvm_info("report_phase", $sformatf("Total failed[MOSO]: %0d", MISO_error_count), UVM_LOW)
57          endfunction : report_phase
58
59      endclass : wrapper_scoreboard
60  endpackage : wrapper_scoreboard_pkg
```

Wrapper Coverage Collector:

```
6   package wrapper_coverage_pkg;
7       import wrapper_seq_item_pkg::*;
8       import wrapper_shared_pkg::*;
9       import uvm_pkg::*;
10      `include "uvm_macros.svh"
11      class wrapper_coverage extends  uvm_scoreboard;
12          `uvm_component_utils(wrapper_coverage)
13
14          uvm_analysis_export #(wrapper_seq_item) cov_export;
15          uvm_tlm_analysis_fifo #(wrapper_seq_item) cov_fifo;
16          wrapper_seq_item seq_item_cov;
17
18          covergroup covgrp;
19              rst_cp:coverpoint seq_item_cov.rst_n{
20                  bins active   = {0};
21                  bins inactive = {1};
22              }
23
24              cs_cov_cp: coverpoint cs_cov {
25                  bins wr_add  = {WRITE_ADD_COV};
26                  bins wr_data = {WRITE_DATA_COV};
27                  bins rd_add  = {READ_ADD_COV};
28                  bins rd_data = {READ_DATA_COV};
29              }
30          endgroup : covgrp
31
32          function new(string name = "wrapper_coverage", uvm_component parent = null);
33              super.new(name, parent);
34              covgrp = new;
35          endfunction : new
36
37          function void build_phase(uvm_phase phase);
38              super.build_phase(phase);
39              cov_export = new("cov_export", this);
40              cov_fifo = new("cov_fifo", this);
41          endfunction : build_phase
42
43          function void connect_phase(uvm_phase phase);
44              super.connect_phase(phase);
45              cov_export.connect(cov_fifo.analysis_export);
46          endfunction : connect_phase
47
48          task run_phase(uvm_phase phase);
49              super.run_phase(phase);
50              forever begin
51                  cov_fifo.get(seq_item_cov);
52                  covgrp.sample();
53              end
54          endtask : run_phase
55
56
57      endclass : wrapper_coverage
58  endpackage : wrapper_coverage_pkg
```

Wrapper Environment:

```
6   package wrapper_env_pkg;
7       import wrapper_scoreboard_pkg::*;
8       import wrapper_coverage_pkg::*;
9       import wrapper_agent_pkg::*;
10      import uvm_pkg::*;
11      `include "uvm_macros.svh"
12      class wrapper_env extends  uvm_env;
13          `uvm_component_utils(wrapper_env)
14
15          wrapper_agent agt;
16          wrapper_scoreboard sb;
17          wrapper_coverage cov;
18
19          function  new(string name = "wrapper_env" , uvm_component parent = null);
20              super.new(name,parent);
21          endfunction : new
22
23          function void build_phase(uvm_phase phase);
24              super.build_phase(phase);
25              agt=wrapper_agent::type_id::create("agt",this);
26              sb=wrapper_scoreboard::type_id::create("sb",this);
27              cov=wrapper_coverage::type_id::create("cov",this);
28          endfunction : build_phase
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 : connect_phase
35
36      endclass : wrapper_env
37
38  endpackage : wrapper_env_pkg
```

Test & Top: -

Test file:

```
6   package wrapper_test_pkg;
7       import RAM_env_pkg::*;
8       import slave_env_pkg::*;
9       import wrapper_env_pkg::*;
10      import RAM_config_pkg::*;
11      import slave_config_pkg::*;
12      import wrapper_config_pkg::*;
13      import wrapper_sequence_pkg::*;
14      import uvm_pkg::*;
15      `include "uvm_macros.svh"
16
17      class wrapper_test extends  uvm_test;
18          `uvm_component_utils(wrapper_test)
19
20          RAM_env env_RAM;
21          slave_env env_slave;
22          wrapper_env env_wrapper;
23          RAM_config RAM_cfg;
24          slave_config slave_cfg;
25          wrapper_config wrapper_cfg;
26          wrapper_rst_assert rst_seq;
27          wrapper_main_seq main_seq;
28
29          function  new(string name = "wrapper_test", uvm_component parent = null);
30              super.new(name, parent);
31          endfunction : new
32
```

```
32
33          function void build_phase (uvm_phase phase);
34              super.build_phase(phase);
35              env_RAM         = RAM_env::type_id::create("env_RAM", this);
36              env_slave       = slave_env::type_id::create("env_slave", this);
37              env_wrapper     = wrapper_env::type_id::create("env_wrapper", this);
38              RAM_cfg         = RAM_config::type_id::create("RAM_cfg");
39              slave_cfg       = slave_config::type_id::create("slave_cfg");
40              wrapper_cfg     = wrapper_config::type_id::create("wrapper_cfg");
41              rst_seq         = wrapper_rst_assert::type_id::create("rst_seq");
42              main_seq        = wrapper_main_seq::type_id::create("main_seq");
43
44              if (!uvm_config_db #(virtual RAM_if)::get(this, "", "RAM_IF", RAM_cfg.RAM_vif))
45                  `uvm_fatal("build_phase", "Test-unable to get the virtual interface of the RAM from the uvm_config_db")
46              if (!uvm_config_db #(virtual slave_if)::get(this, "", "slave_IF", slave_cfg.slave_vif))
47                  `uvm_fatal("build_phase", "Test-unable to get the virtual interface of the SLAVE from the uvm_config_db")
48              if (!uvm_config_db #(virtual wrapper_if)::get(this, "", "wrapper_IF", wrapper_cfg.wrapper_vif))
49                  `uvm_fatal("build_phase", "Test-unable to get the virtual interface of the wrapper from the uvm_config_db")
50
51              RAM_cfg.is_active    =UVM_PASSIVE;
52              slave_cfg.is_active  =UVM_PASSIVE;
53              wrapper_cfg.is_active=UVM_ACTIVE;
54
55              uvm_config_db #(RAM_config)   ::set(this, "*", "CFG", RAM_cfg);
56              uvm_config_db #(slave_config)   ::set(this, "*", "CFG", slave_cfg);
57              uvm_config_db #(wrapper_config) ::set(this, "*", "CFG", wrapper_cfg);
58          endfunction : build_phase
59
60          task run_phase (uvm_phase phase);
61              super.run_phase(phase);
62              phase.raise_objection(this);
63                  `uvm_info("run_phase", "Reset Asserted", UVM_LOW);
64                  rst_seq.start(env_wrapper.agt.sqr);
65                  `uvm_info("run_phase", "Reset Deasserted", UVM_LOW);
66
67                  `uvm_info("run_phase", "Main Seq Started", UVM_LOW);
68                  main_seq.start(env_wrapper.agt.sqr);
69                  `uvm_info("run_phase", "Main Seq Ended", UVM_LOW);
70
71                  `uvm_info("run_phase", "Finish Test", UVM_LOW);
72              phase.drop_objection(this);
73          endtask : run_phase
74
75      endclass : wrapper_test
76  endpackage : wrapper_test_pkg
```

Top file:

```
import wrapper_test_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
module top ();
    bit clk;
    initial begin
        $readmemh("RAM_data.dat", DUT_WRAPPER.ram.mem, 0, 255);
        $readmemh("RAM_data.dat", GOLDEN_WRAPPER.mem, 0, 255);
        clk=0;
        forever begin
            #1 clk=~clk;
        end
    end
/*****************************************************/
//Interfaces
    slave_if s_if(clk);
    RAM_if r_if(clk);
    wrapper_if w_if(clk);
/*****************************************************/
    Maindesign DUT_WRAPPER(
        .clk  (clk),
        .rst_n(w_if.rst_n),
        .MISO (w_if.MISO),
        .MOSI (w_if.MOSI),
        .SS_n (w_if.SS_n)
    );
/*****************************************************/
    SPI_SLAVE GOLDEN_SLAVE(
        .clk     (clk),
        .MISO    (s_if.MISO_ref),
        .MOSI    (s_if.MOSI),
        .tx_valid(s_if.tx_valid),
        .tx_data (s_if.tx_data),
        .rx_valid(s_if.rx_valid_ref),
        .SS_n    (s_if.SS_n),
        .rx_data (s_if.rx_data_ref),
        .rst_n   (s_if.rst_n)
    );
    wrapper_golden GOLDEN_WRAPPER(
        .clk  (clk),
        .rst_n(w_if.rst_n),
        .MISO (w_if.MISO_ref),
        .MOSI (w_if.MOSI),
        .SS_n (w_if.SS_n)
    );
/*****************************************************/
```

```
//Assertions
    bind RAM RAM_sva RAM_SVA(
        .din(r_if.din),
        .rx_valid(r_if.rx_valid),
        .rst_n(r_if.rst_n),
        .dout(r_if.dout),
        .tx_valid(r_if.tx_valid),
        .wr_addr(DUT_WRAPPER.ram.wr_addr),
        .rd_addr(DUT_WRAPPER.ram.rd_addr),
        .mem(DUT_WRAPPER.ram.mem),
        .clk(clk)
    );

    bind SLAVE slave_sva SLAVE_SVA(
        .clk          (clk),
        .MISO         (s_if.MISO),
        .MOSI         (s_if.MOSI),
        .tx_valid     (s_if.tx_valid),
        .tx_data      (s_if.tx_data),
        .SS_n         (s_if.SS_n),
        .rx_valid     (s_if.rx_valid),
        .rst_n        (s_if.rst_n),
        .rx_data      (s_if.rx_data),
        .cs           (DUT_WRAPPER.spi.cs),
        .rd_add_signal(DUT_WRAPPER.spi.rd_add_signal),
        .count_rx     (DUT_WRAPPER.spi.count_rx),
        .count_tx     (DUT_WRAPPER.spi.count_tx)
    );

    bind Maindesign Wrapper_sva WRAPPER_SVA(
        .rst_n   (w_if.rst_n),
        .MOSI    (w_if.MOSI),
        .SS_n    (w_if.SS_n),
        .MISO    (w_if.MISO),
        .tx_valid(DUT_WRAPPER.tx_valid),
        .rx_valid(DUT_WRAPPER.rx_valid),
        .tx_data (DUT_WRAPPER.tx_data),
        .rx_data (DUT_WRAPPER.rx_data),
        .clk     (clk)
        );
/*****************************************************/
```

```
92  /*****************************************************/
93      assign r_if.din      = DUT_WRAPPER.rx_data;
94      assign r_if.rst_n    = w_if.rst_n;
95      assign r_if.rx_valid = DUT_WRAPPER.rx_valid;
96      assign r_if.dout     = DUT_WRAPPER.tx_data;
97      assign r_if.tx_valid = DUT_WRAPPER.tx_valid;
98
99      assign s_if.MOSI     = w_if.MOSI;
100     assign s_if.SS_n     = w_if.SS_n;
101     assign s_if.rst_n    = w_if.rst_n;
102     assign s_if.tx_valid = DUT_WRAPPER.tx_valid;
103     assign s_if.tx_data  = DUT_WRAPPER.tx_data;
104     assign s_if.rx_data  = DUT_WRAPPER.rx_data;
105     assign s_if.MISO     = w_if.MISO;
106     assign s_if.rx_valid = DUT_WRAPPER.rx_valid;
107 /*****************************************************/
108     initial begin
109         uvm_config_db#(virtual RAM_if)::set(null,"uvm_test_top","RAM_IF",r_if );
110         uvm_config_db#(virtual slave_if)::set(null,"uvm_test_top","slave_IF",s_if );
111         uvm_config_db#(virtual wrapper_if)::set(null,"uvm_test_top","wrapper_IF",w_if );
112
113         run_test("wrapper_test");
114     end
115 endmodule : top
```

For running the test files: -

Do file to run the RAM:

```
1  vlib work
2  vlog -f list_ram.list +cover -covercells
3  vsim -voptargs=+acc work.top -cover
4  add wave -position insertpoint sim:/top/r_if/*
5  add wave -position insertpoint  \
6  sim:/top/DUT/mem
7  run -all
```

Do file to run the SLAVE:

```
1  vlib work
2  vlog -f list_slave.list +cover -covercells
3  vsim -voptargs=+acc work.top -cover
4  add wave -position insertpoint sim:/top/s_if/*
5  coverage exclude -src ../slave/Design/SPI.sv -line 99 -code b
6  coverage exclude -src ../slave/Design/SPI.sv -line 27 -code b
7  coverage exclude -src ../slave/Design/SPI.sv -line 73 -code b
8  coverage exclude -src ../slave/Design/SPI.sv -line 98 -code b
9  coverage exclude -src ../slave/Design/SPI.sv -line 37 -code c
10 coverage exclude -src ../slave/Design/SPI.sv -line 39 -code c
11 coverage exclude -src ../slave/Design/SPI.sv -line 98 -code c
12 coverage exclude -src ../slave/Design/SPI.sv -line 102 -code c
13 run -all
```

Do file to run the Wrapper:

```
1  vlib work
2  vlog -f list_wrapper.list +cover -covercells
3  vsim -voptargs=+acc work.top -cover
4  add wave -position insertpoint sim:/top/w_if/*
5  run -all
```

List of RAM:

```
../RAM/Shared_pkg/RAM_shared_pkg.sv
../RAM/Design/RAM.sv
../RAM/Interface/RAM_if.sv
../RAM/Assertions/RAM_sva.sv

../RAM/Objects/Configrstion/RAM_config_pkg.sv
../RAM/Objects/Sequence_Item/RAM_seq_item_pkg.sv
../RAM/Objects/Sequences/RAM_seq_pkg.sv

../RAM/Top/Test/Enviroment/Agent/Driver/RAM_driver_pkg.sv
../RAM/Top/Test/Enviroment/Agent/Monitor/RAM_monitor_pkg.sv
../RAM/Top/Test/Enviroment/Agent/Sequencer/RAM_sequencer_pkg.sv
../RAM/Top/Test/Enviroment/Agent/RAM_agent_pkg.sv

../RAM/Top/Test/Enviroment/Coverage_Collector/RAM_coverage_pkg.sv
../RAM/Top/Test/Enviroment/Scoreboard/RAM_scoreboard_pkg.sv
../RAM/Top/Test/Enviroment/RAM_env_pkg.sv

../RAM/Top/Test/RAM_test_pkg.sv
../RAM/Top/RAM_top.sv
```

List of SLAVE:

```
../slave/Shared_pkg/slave_shared_pkg.sv
../slave/Design/SPI.sv
../slave/Golden/Slave.sv
../slave/Assertions/slave_sva.sv
../slave/Interface/slave_if.sv


../slave/Objects/Configrstion/slave_config_pkg.sv
../slave/Objects/Sequence_Item/slave_seq_item_pkg.sv
../slave/Objects/Sequences/slave_seq_pkg.sv

../slave/Top/Test/Enviroment/Agent/Driver/slave_driver_pkg.sv
../slave/Top/Test/Enviroment/Agent/Monitor/slave_monitor_pkg.sv
../slave/Top/Test/Enviroment/Agent/Sequencer/slave_sequencer_pkg.sv
../slave/Top/Test/Enviroment/Agent/slave_agent_pkg.sv

../slave/Top/Test/Enviroment/Coverage_Collector/slave_coverage_pkg.sv
../slave/Top/Test/Enviroment/Scoreboard/slave_scoreboard_pkg.sv
../slave/Top/Test/Enviroment/slave_env_pkg.sv

../slave/Top/Test/slave_test_pkg.sv
../slave/Top/slave_top.sv
```

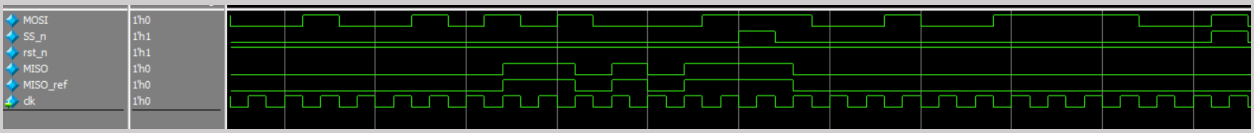
List of Wrapper:

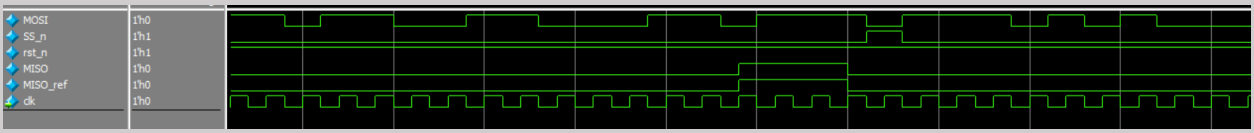
```
../wrapper/Shared_pkg/RAM_shared_pkg.sv
../wrapper/Shared_pkg/wrapper_shared_pkg.sv

../wrapper/Design/RAM.sv
../wrapper/Design/SPI.sv
../wrapper/Design/Wrapper.sv

../wrapper/Golden/Slave.sv
../wrapper/Golden/wrapper_golden.sv

../wrapper/Assertions/RAM_sva.sv
../wrapper/Assertions/slave_sva.sv
../wrapper/Assertions/Wrapper_sva.sv

../wrapper/Interface/RAM_if.sv
../wrapper/Interface/slave_if.sv
../wrapper/Interface/wrapper_if.sv


../wrapper/Objects/Configuration/RAM_config_pkg.sv
../wrapper/Objects/Configuration/slave_config_pkg.sv
../wrapper/Objects/Configuration/wrapper_config_pkg.sv

../wrapper/Objects/Sequence_Item/RAM_seq_item_pkg.sv
../wrapper/Objects/Sequence_Item/slave_seq_item_pkg.sv
../wrapper/Objects/Sequence_Item/wrapper_seq_item.sv

../wrapper/Objects/Sequences/RAM_seq_pkg.sv
../wrapper/Objects/Sequences/slave_seq_pkg.sv
../wrapper/Objects/Sequences/wrapper_sequence_pkg.sv


../wrapper/Top/Test/Enviroment_RAM/Agent/Driver/RAM_driver_pkg.sv
../wrapper/Top/Test/Enviroment_SLAVE/Agent/Driver/slave_driver_pkg.sv
../wrapper/Top/Test/Enviroment_Wrapper/Agent/Driver/wrapper_driver_pkg.sv

../wrapper/Top/Test/Enviroment_RAM/Agent/Monitor/RAM_monitor_pkg.sv
../wrapper/Top/Test/Enviroment_SLAVE/Agent/Monitor/slave_monitor_pkg.sv
../wrapper/Top/Test/Enviroment_Wrapper/Agent/Monitor/wrapper_monitor_pkg.sv

../wrapper/Top/Test/Enviroment_RAM/Agent/Sequencer/RAM_sequencer_pkg.sv
../wrapper/Top/Test/Enviroment_SLAVE/Agent/Sequencer/slave_sequencer_pkg.sv
../wrapper/Top/Test/Enviroment_Wrapper/Agent/Sequencer/wrapper_sequencer_pkg.sv

../wrapper/Top/Test/Enviroment_RAM/Agent/RAM_agent_pkg.sv
../wrapper/Top/Test/Enviroment_SLAVE/Agent/slave_agent_pkg.sv
../wrapper/Top/Test/Enviroment_Wrapper/Agent/wrapper_agent_pkg.sv

../wrapper/Top/Test/Enviroment_RAM/Coverage_Collector/RAM_coverage_pkg.sv
../wrapper/Top/Test/Enviroment_SLAVE/Coverage_Collector/slave_coverage_pkg.sv
../wrapper/Top/Test/Enviroment_Wrapper/Coverage_Collector/wrapper_coverage_pkg.sv

../wrapper/Top/Test/Enviroment_RAM/Scoreboard/RAM_scoreboard_pkg.sv
../wrapper/Top/Test/Enviroment_SLAVE/Scoreboard/slave_scoreboard_pkg.sv
../wrapper/Top/Test/Enviroment_Wrapper/Scoreboard/wrapper_scoreboard_pkg.sv
```

```
../wrapper/Top/Test/Enviroment_RAM/RAM_env_pkg.sv
../wrapper/Top/Test/Enviroment_SLAVE/slave_env_pkg.sv
../wrapper/Top/Test/Enviroment_Wrapper/wrapper_env_pkg.sv

../wrapper/Top/Test/wrapper_test_pkg.sv
../wrapper/Top/wrapper_top.sv
```

Reports: -

Questasim snippets:





Report snippets:

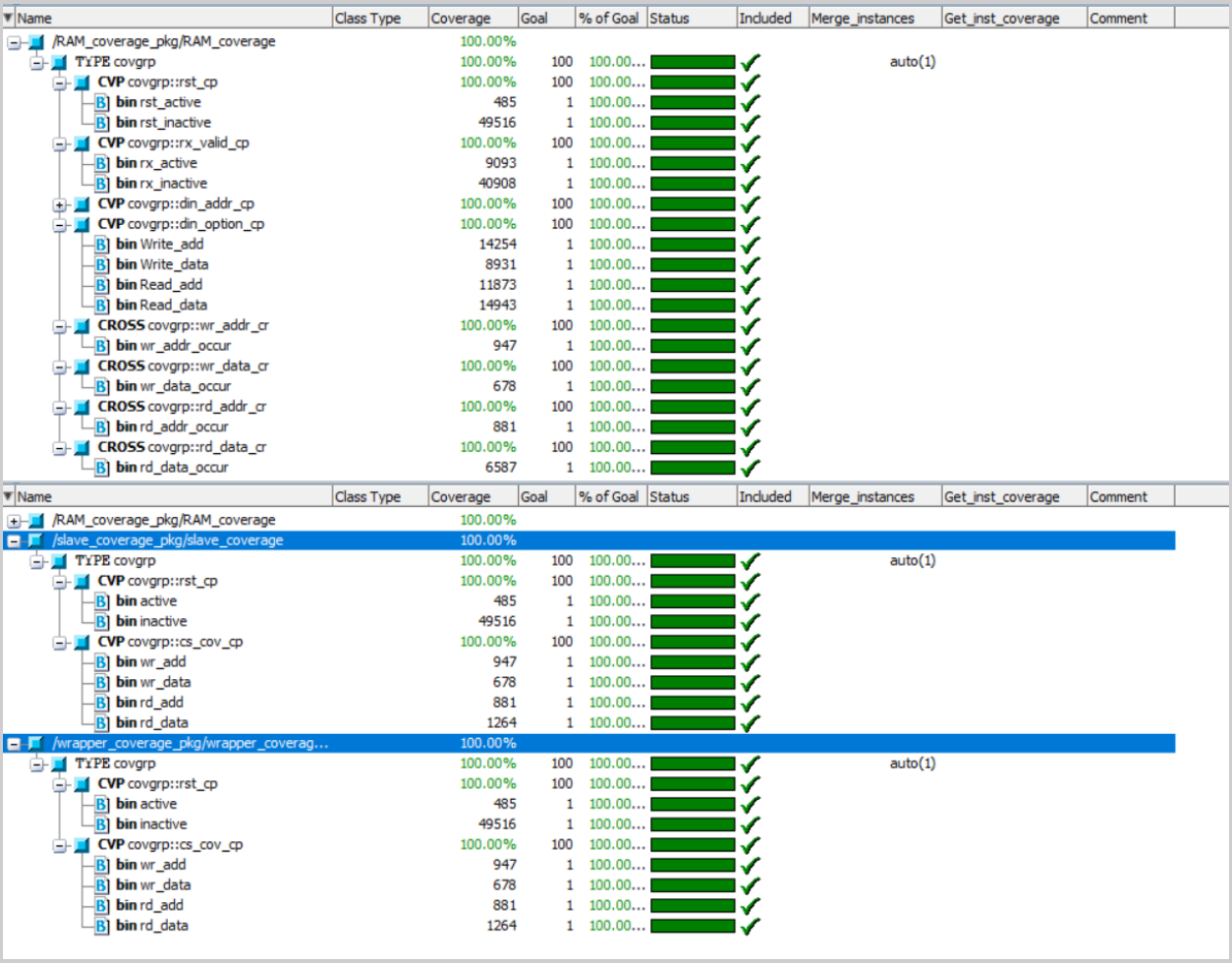
```
# UVM_INFO @ 0: reporter [RNTST] Running test wrapper_test...
# UVM_INFO ../wrapper/Top/Test/wrapper_test_pkg.sv(63) @ 0: uvm_test_top [run_phase] Reset Asserted
# UVM_INFO ../wrapper/Top/Test/wrapper_test_pkg.sv(65) @ 2: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO ../wrapper/Top/Test/wrapper_test_pkg.sv(67) @ 2: uvm_test_top [run_phase] Main Seq Started
# UVM_INFO ../wrapper/Top/Test/wrapper_test_pkg.sv(69) @ 100002: uvm_test_top [run_phase] Main Seq Ended
# UVM_INFO ../wrapper/Top/Test/wrapper_test_pkg.sv(71) @ 100002: uvm_test_top [run_phase] Finish Test
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 100002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ../wrapper/Top/Test/Enviroment_RAM/Scoreboard/RAM_scoreboard_pkg.sv(93) @ 100002: uvm_test_top.env_RAM.sb [report_phase] Total successful transactions: 50001
# UVM_INFO ../wrapper/Top/Test/Enviroment_RAM/Scoreboard/RAM_scoreboard_pkg.sv(94) @ 100002: uvm_test_top.env_RAM.sb [report_phase] Total failed transactions: 0
# UVM_INFO ../wrapper/Top/Test/Enviroment_SLAVE/Scoreboard/slave_scoreboard_pkg.sv(66) @ 100002: uvm_test_top.env_slave.sb [report_phase] Total successful[RX]: 50001
# UVM_INFO ../wrapper/Top/Test/Enviroment_SLAVE/Scoreboard/slave_scoreboard_pkg.sv(67) @ 100002: uvm_test_top.env_slave.sb [report_phase] Total failed[RX]: 0
# UVM_INFO ../wrapper/Top/Test/Enviroment_SLAVE/Scoreboard/slave_scoreboard_pkg.sv(68) @ 100002: uvm_test_top.env_slave.sb [report_phase] Total successful[MOSO]: 50001
# UVM_INFO ../wrapper/Top/Test/Enviroment_SLAVE/Scoreboard/slave_scoreboard_pkg.sv(69) @ 100002: uvm_test_top.env_slave.sb [report_phase] Total failed[MOSO]: 0
# UVM_INFO ../wrapper/Top/Test/Enviroment_Wrapper/Scoreboard/wrapper_scoreboard_pkg.sv(55) @ 100002: uvm_test_top.env_wrapper.sb [report_phase] Total successful[MOSO]: 50001
# UVM_INFO ../wrapper/Top/Test/Enviroment_Wrapper/Scoreboard/wrapper_scoreboard_pkg.sv(56) @ 100002: uvm_test_top.env_wrapper.sb [report_phase] Total failed[MOSO]: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   17
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     8
# [run_phase]     5
# ** Note: $finish    : D:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 100002 ns  Iteration: 61  Instance: /top
```

Sequential Domain Coverage report:

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Ass
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0	-	-	-	-		off	ass
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0	-	-	-	-		off	ass
/wrapper_sequence_pkg::wrapper_main_seq::body/#ublk#176326455#48/imme...	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/spi/SLAVE_SVA/rst_count_rx_assert	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/spi/SLAVE_SVA/rst_count_tx_assert	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/spi/SLAVE_SVA/rst_rd_add_signal_assert	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/spi/SLAVE_SVA/rst_cs_assert	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/spi/SLAVE_SVA/idel_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/spi/SLAVE_SVA/check_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/spi/SLAVE_SVA/write_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/spi/SLAVE_SVA/read_add_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/spi/SLAVE_SVA/read_data_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/spi/SLAVE_SVA/rd_flag_ra_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/spi/SLAVE_SVA/rd_flag_rd_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/ram/RAM_SVA/rst_wr_addr_assert	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/ram/RAM_SVA/rst_rd_addr_assert	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/ram/RAM_SVA/rst_tx_valid_assert	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/ram/RAM_SVA/wr_addr_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/ram/RAM_SVA/wr_data_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/ram/RAM_SVA/rd_addr_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/ram/RAM_SVA/rd_data_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/ram/RAM_SVA/tx_valid_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/WRAPPER_SVA/rst_tx_valid_assert	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/WRAPPER_SVA/rst_rst_rx_valid_assert	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/WRAPPER_SVA/rst_rst_tx_data_assert	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/WRAPPER_SVA/rst_rst_rx_data_assert	Immediate	SVA	on	0	1	-	-	-	-		off	ass
/top/DUT_WRAPPER/WRAPPER_SVA/rx_valid_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/WRAPPER_SVA/tx_valid_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass
/top/DUT_WRAPPER/WRAPPER_SVA/rx_data_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	ass

Name	Language	Enabled	Log	Count	AtLeas	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Thre
/top/DUT_WRAPPER/spi/SLAVE_SVA/rst_count_rx_cover	SVA	✓	Off	479	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/spi/SLAVE_SVA/rst_count_tx_cover	SVA	✓	Off	479	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/spi/SLAVE_SVA/rst_rd_add_signal_cover	SVA	✓	Off	479	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/spi/SLAVE_SVA/rst_cs_cover	SVA	✓	Off	479	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/spi/SLAVE_SVA/idel_cover	SVA	✓	Off	3062	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/spi/SLAVE_SVA/check_cover	SVA	✓	Off	3513	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/spi/SLAVE_SVA/write_cover	SVA	✓	Off	1763	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/spi/SLAVE_SVA/read_add_cover	SVA	✓	Off	960	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/spi/SLAVE_SVA/read_data_cover	SVA	✓	Off	762	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/spi/SLAVE_SVA/rd_flag_ra_cover	SVA	✓	Off	860	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/spi/SLAVE_SVA/rd_flag_rd_cover	SVA	✓	Off	620	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/ram/RAM_SVA/rst_wr_addr_cover	SVA	✓	Off	479	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/ram/RAM_SVA/rst_rd_addr_cover	SVA	✓	Off	479	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/ram/RAM_SVA/rst_tx_valid_cover	SVA	✓	Off	479	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/ram/RAM_SVA/wr_addr_cover	SVA	✓	Off	922	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/ram/RAM_SVA/wr_data_cover	SVA	✓	Off	660	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/ram/RAM_SVA/rd_addr_cover	SVA	✓	Off	860	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/ram/RAM_SVA/rd_data_cover	SVA	✓	Off	6455	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/ram/RAM_SVA/tx_valid_cover	SVA	✓	Off	6455	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/WRAPPER_SVA/rst_tx_valid_cover	SVA	✓	Off	479	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/WRAPPER_SVA/rst_rst_rx_valid_cover	SVA	✓	Off	479	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/WRAPPER_SVA/rst_rst_tx_data_cover	SVA	✓	Off	479	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/WRAPPER_SVA/rst_rst_rx_data_cover	SVA	✓	Off	479	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/WRAPPER_SVA/rx_valid_cover	SVA	✓	Off	2745	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/WRAPPER_SVA/tx_valid_cover	SVA	✓	Off	678	1	Unlimited	1	100%		✓	0	0	0 ns	
/top/DUT_WRAPPER/WRAPPER_SVA/rx_data_cover	SVA	✓	Off	2745	1	Unlimited	1	100%		✓	0	0	0 ns	

Coverage groups:



Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/RAM_coverage_pkg/RAM_coverage		100.00%							
TYPE covgrp		100.00%	100	100.00...			auto(1)		
CVP covgrp::rst_cp		100.00%	100	100.00...					
bin rst_active		485	1	100.00...					
bin rst_inactive		49516	1	100.00...					
CVP covgrp::rx_valid_cp		100.00%	100	100.00...					
bin rx_active		9093	1	100.00...					
bin rx_inactive		40908	1	100.00...					
CVP covgrp::din_addr_cp		100.00%	100	100.00...					
CVP covgrp::din_option_cp		100.00%	100	100.00...					
bin Write_add		14254	1	100.00...					
bin Write_data		8931	1	100.00...					
bin Read_add		11873	1	100.00...					
bin Read_data		14943	1	100.00...					
CROSS covgrp::wr_addr_cr		100.00%	100	100.00...					
bin wr_addr_occur		947	1	100.00...					
CROSS covgrp::wr_data_cr		100.00%	100	100.00...					
bin wr_data_occur		678	1	100.00...					
CROSS covgrp::rd_addr_cr		100.00%	100	100.00...					
bin rd_addr_occur		881	1	100.00...					
CROSS covgrp::rd_data_cr		100.00%	100	100.00...					
bin rd_data_occur		6587	1	100.00...					

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/RAM_coverage_pkg/RAM_coverage		100.00%							
/slave_coverage_pkg/slave_coverage		100.00%							
TYPE covgrp		100.00%	100	100.00...			auto(1)		
CVP covgrp::rst_cp		100.00%	100	100.00...					
bin active		485	1	100.00...					
bin inactive		49516	1	100.00...					
CVP covgrp::cs_cov_cp		100.00%	100	100.00...					
bin wr_add		947	1	100.00...					
bin wr_data		678	1	100.00...					
bin rd_add		881	1	100.00...					
bin rd_data		1264	1	100.00...					
/wrapper_coverage_pkg/wrapper_coverag...		100.00%							
TYPE covgrp		100.00%	100	100.00...			auto(1)		
CVP covgrp::rst_cp		100.00%	100	100.00...					
bin active		485	1	100.00...					
bin inactive		49516	1	100.00...					
CVP covgrp::cs_cov_cp		100.00%	100	100.00...					
bin wr_add		947	1	100.00...					
bin wr_data		678	1	100.00...					
bin rd_add		881	1	100.00...					
bin rd_data		1264	1	100.00...					

Codecoverage:

```
================================================================================
=== Instance: /\top#DUT_WRAPPER /spi
=== Design Unit: work.SLAVE
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        26        26         0   100.00%

================================Branch Details================================
```

```
Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                       9         9         0   100.00%

================================Condition Details================================

Condition Coverage for instance /\top#DUT_WRAPPER /spi --
```

```
FSM Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    FSM States                       5         5         0   100.00%
    FSM Transitions                  8         8         0   100.00%

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

FSM Coverage for instance /\top#DUT_WRAPPER /spi --
```

```
        FSM Transitions              8         8         0   100.00%
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      36        36         0   100.00%

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

Statement Coverage for instance /\top#DUT_WRAPPER /spi --
```

```
    120             1                     44102
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         52        52         0   100.00%

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

Toggle Coverage for instance /\top#DUT_WRAPPER /spi --
```

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         6         6         0   100.00%

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

Branch Coverage for instance /\top#DUT_WRAPPER /ram
```

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

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

Statement Coverage for instance /\top#DUT_WRAPPER /ram --
```

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         76        76         0   100.00%

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

Toggle Coverage for instance /\top#DUT_WRAPPER /ram --

                                                  Node      1H ->0L      0L ->1H
```

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

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

Toggle Coverage for instance /\top#DUT_WRAPPER  --
```

```
                                                         0         1

Total Coverage By Instance (filtered view): 100.00%
```


Note: I excluded the statements and conditions that will not be implemented.