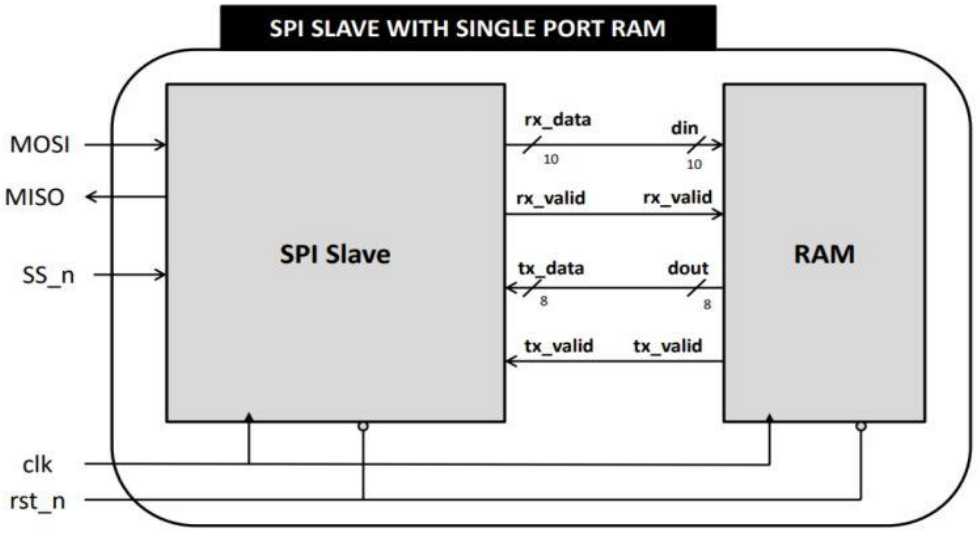
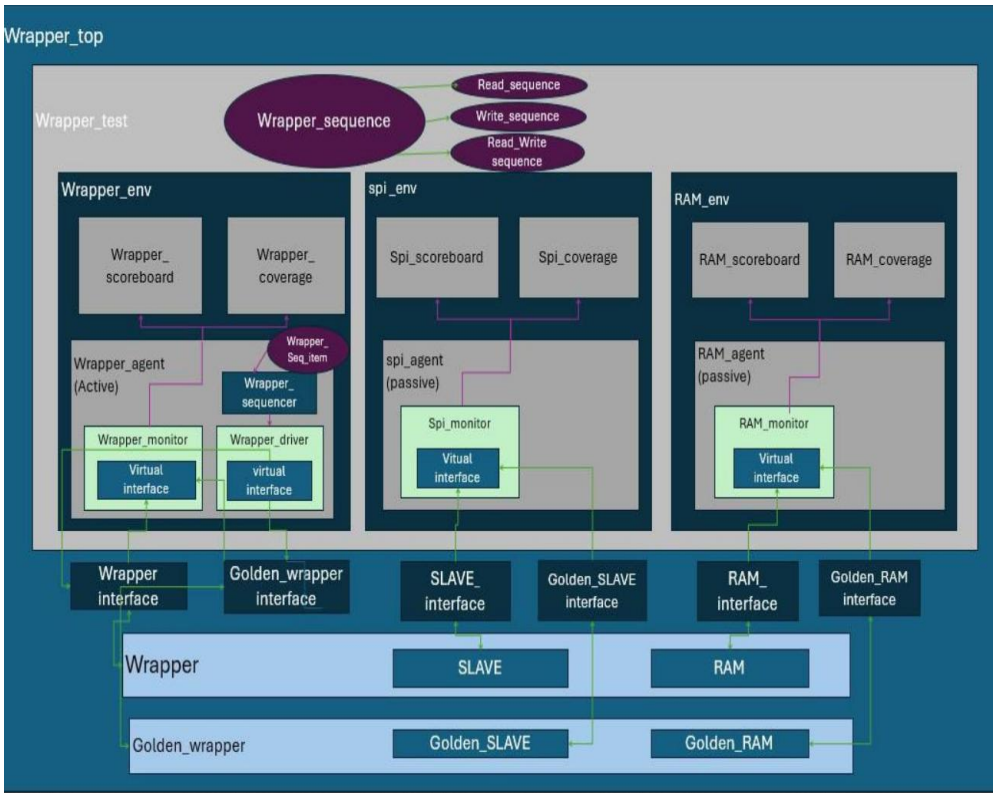
SPI Communication System Testbench

USING UVM





Part 1: UVM Environment for SPI Slave

- **Verification Plan**

Label	Description	Stimulus Generation	Functional Coverage (Later)	Functionality Check
Reset Behavior	Verify all outputs after reset.	Drive reset low for few cycles and release.	Cover reset active and inactive states.	Assertion p_reset_outputs ensures all outputs = 0
Default CS	Ensure DUT stays idle when SS_n = 1.	Keep SS_n high; no transactions applied.	Cover SS_n=1 state.	Assertion ensures FSM=IDLE when SS_n=1
Valid CMD Decode	Check FSM goes to correct state for each valid command	Send valid command sequences: 000, 001, 110, 111.	Cover transitions for all legal opcodes	Assertion p_cmd_decode ensures next state is correct
Write Addr Sequence	Verify write address command path.	Send 000 command with address data.	Cover IDLE→WRITE_ADD transitions	Assertion ensures FSM goes IDLE→WRITE_ADD→IDLE
Write Data Sequence	Verify write data command path.	Send 001 command with data bytes	Cover WRITE_ADD→WRITE_DATA path	Assertion ensures RX_VALID after 10 cycles.
Read Addr Sequence	Verify read address command path	Send 110 command sequence.	Cover IDLE→READ_ADD transitions	Assertion ensures FSM next state correct
Read Data Sequence	Verify read data command path	Send 111 command sequence.	Cover READ_ADD→READ_DATA path	Assertion checks read data match expected
RX_VALID De	Check RX_VALID asserted after fixed delay.	Monitor timing between SS_n low and RX_VALID.	Cover different delay counts.	Assertion p_rxvalid_delay checks delay = 10 cycles.
SS_n Toggle	Verify behavior when SS_n toggles mid-transfer	Toggle SS_n during transaction	Cover SS_n glitches and reassertion	Assertion ensures FSM returns to IDLE safely.
Counter Wrap	Verify internal bit counter reset.	Observe counter across full frame	Cover count=0 and max transitions.	Assertion p_counter_reset ensures reset at end of frame.
Continuous T	Verify continuous back-to-back transfers.	Send multiple commands without idle	Cover overlapping transactions.	Assertion ensures no RX_VALID overlap.
RX_VALID After Each Command	Ensure RX_VALID asserts exactly 10 cycles after any valid comman	Apply each valid sequence and monitor RX_VALID timing.	Cover all command types and RX_VALID timing bins.	Assertion checks RX_VALID asserted exactly after 10 cycles for every valid command.
RX_VALID After Each Command	Ensure RX_VALID asserts exactly 10 cycles after any valid comman	Apply each valid sequence and monitor RX_VALID timing.	Cover all command types and RX_VALID timing bins.	Assertion checks RX_VALID asserted exactly after 10 cycles for every valid command.
SS_n glitch	Verify system recovery from short SS_n pulse.	Generate very short SS_n low pulse.	Cover glitch event.	Assertion ensures FSM not corrupted.
Random Stress Test	Validate overall protocol with random data.	Randomize command, address, and da	Cross cover all command/data types.	Scoreboard ensures data integrity
RX_VALID Sta	Verify RX_VALID does not glitch.	Random data sequences with timing va	Cover stable RX_VALID bins	Assertion ensures RX_VALID stable when high.
FSM Coverag	Ensure all states & transitions covered.	Run directed & random sequences.	Cover all FSM states and transiti	Coverage report ensures 100% FSM coverage.

- **Golden Model**

SPI.sv > ...

```
module SPI_Slave(MOSI,SS_n,clk,rst_n,rx_data,tx_valid,tx_data,MISO,rx_valid);
input MOSI,SS_n,clk,rst_n,tx_valid;
input [7:0] tx_data;
output reg MISO,rx_valid;
output  reg [9:0] rx_data;
parameter IDLE=3'b000;
parameter CHK_CMD=3'b001;
parameter WRITE=3'b010;
parameter READ_ADD=3'b011;
parameter  READ_DATA=3'b100;
reg ADDRESS_read; // this is signal to increment when  reading an address
reg[3:0] counter;
reg [2:0] cs ,ns;
always@(posedge clk) 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)
    ns = IDLE;
    else begin
     if( ~MOSI) begin
        ns=WRITE;
    end
    else begin
        casex(ADDRESS_read)
        1'b0 : ns=READ_ADD;
        1'b1: ns=READ_DATA;
        1'bx: ns=READ_ADD;
        endcase
    end
   end
   end
   WRITE: begin
     if(SS_n==0 )
     ns=WRITE;
     else begin
        ns=IDLE;
     end
   end
   READ_ADD :  begin
    if(SS_n==0) begin
    ns=READ_ADD;
```

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

           end
      end
      else begin
         if(counter>0) begin
          rx_data[counter-1]<=MOSI;
          counter<=counter-1;
          rx_valid<=0;
         end
         else begin
          rx_valid<=1;
          counter<=9;
         end
      end
     end

     endcase
      end
  end
  endmodule
```

- **DUT With Design Assertions**

SPI_slave.sv > SystemVerilog - Language Support > SLAVE

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

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

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

reg [3:0] counter;
reg       received_address;

reg [2:0] cs, ns;

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

always @(*) begin
    case (cs)
        IDLE : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = CHK_CMD;
        end
        CHK_CMD : begin
            if (SS_n)
                ns = IDLE;
            else begin
                if (~MOSI)
                    ns = WRITE;
                else begin
                    if (received_address)
                        ns = READ_DATA;
                    else
                        ns = READ_ADD; /// edit 1 replace with another
                end
            end
        end
        WRITE : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = WRITE;
        end
        READ_ADD : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = READ_ADD;
        end
        READ_DATA : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = READ_DATA;
        end
```

```
    endcase
end

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

```
`ifdef SIM
 property p_IDLE_to_CHK_CMD;
     @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n)
     (cs == IDLE && spi_vif.SS_n == 1'b0) |=> (cs == CHK_CMD);
 endproperty

 property p_CHK_CMD_to_WRITE;
     @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n)
     (cs == CHK_CMD && spi_vif.MOSI == 1'b0 && spi_vif.SS_n == 1'b0) |=> (cs == WRITE);
 endproperty
 property p_CHK_CMD_to_READ_ADD;
     @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n)
     (cs == CHK_CMD && spi_vif.MOSI == 1'b1 && received_address == 1'b0 && spi_vif.SS_n == 1'b0)
     |=> (cs == READ_ADD);
 endproperty

 property p_CHK_CMD_to_READ_DATA;
     @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n)
     (cs == CHK_CMD && spi_vif.MOSI == 1'b1 && received_address == 1'b1 && spi_vif.SS_n == 1'b0)
     |=> (cs == READ_DATA);
 endproperty
 property p_WRITE_to_IDLE;
     @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n)
     (cs == WRITE && spi_vif.SS_n == 1'b1) |=> (cs == IDLE);
 endproperty

 property p_READ_ADD_to_IDLE;
     @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n)
     (cs == READ_ADD && spi_vif.SS_n == 1'b1) |=> (cs == IDLE);
 endproperty
 property p_READ_DATA_to_IDLE;
     @(posedge spi_vif.clk)
     disable iff (!spi_vif.rst_n) (cs == READ_DATA && spi_vif.SS_n == 1'b1) |=> (cs == IDLE);
 endproperty
 property p_counter_decrement;
     @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n)
     ((cs == WRITE || cs == READ_ADD|| cs==READ_DATA) && counter > 0)
     |=> (counter == $past(counter) - 1);
 endproperty

 assert property (p_IDLE_to_CHK_CMD);
 assert property (p_CHK_CMD_to_WRITE);
 assert property (p_CHK_CMD_to_READ_ADD);
 assert property (p_CHK_CMD_to_READ_DATA);
 assert property (p_WRITE_to_IDLE);
 assert property (p_READ_ADD_to_IDLE);
 assert property (p_READ_DATA_to_IDLE);
 assert property (p_counter_decrement);
 cover property (p_IDLE_to_CHK_CMD);
 cover property (p_CHK_CMD_to_WRITE);
 cover property (p_CHK_CMD_to_READ_ADD);
 cover property (p_CHK_CMD_to_READ_DATA);
 cover property (p_WRITE_to_IDLE);
 cover property (p_READ_ADD_to_IDLE);
 cover property (p_READ_DATA_to_IDLE);
 cover property (p_counter_decrement);
 `endif

 endmodule
```

DUT interface

spi_interface.sv > SystemVerilog - Language Support > spi_if

```
1   interface spi_if(clk);
2   localparam IDLE      = 3'b000;
3   localparam CHK_CMD   = 3'b001;
4   localparam WRITE     = 3'b010;
5   localparam READ_ADD  = 3'b011;
6   localparam READ_DATA = 3'b100;
7   input  clk;
8   logic rst_n;
9   logic MISO;
10  logic MOSI;
11  logic SS_n;
12  logic tx_valid;
13  logic [7:0] tx_data;
14  logic [9:0] rx_data;
15  bit rx_valid;
16  endinterface
```

Golden Interface

golden_interface.sv > SystemVerilog - Language Support > spi_gold_if

```
1   interface spi_gold_if(clk);
2   localparam IDLE      = 3'b000;
3   localparam CHK_CMD   = 3'b001;
4   localparam WRITE     = 3'b010;
5   localparam READ_ADD  = 3'b011;
6   localparam READ_DATA = 3'b100;
7   input clk;
8   logic rst_n;
9   logic MISO;
10  logic MOSI;
11  logic SS_n;
12  logic tx_valid;
13  logic [7:0] tx_data;
14  logic [9:0] rx_data;
15  bit rx_valid;
16
17  endinterface
18
```

SPI top

spi_top.sv > SystemVerilog - Language Support > spi_top

```
import spi_test_pkg::*;
import uvm_pkg::*;
import spi_shared::*;
  `include "uvm_macros.svh"

  module spi_top();
  bit clk;
 initial begin
    clk=0;
    forever begin
        #1 clk=~clk;
    end
 end

 spi_if spi_vif(clk);
 spi_gold_if gold_vif(clk);
 SLAVE DUT(spi_vif.MOSI,spi_vif.MISO,spi_vif.SS_n,spi_vif.clk,spi_vif.rst_n,spi_vif.rx_data,
          spi_vif.rx_valid,spi_vif.tx_data,spi_vif.tx_valid);

 SPI_Slave Golden(gold_vif.MOSI,gold_vif.SS_n,gold_vif.clk,gold_vif.rst_n,gold_vif.rx_data,gold_vif.tx_valid,gold_vif.tx_data,gold_vif.MISO,
 gold_vif.rx_valid);

bind SLAVE spi_sv sva_inst (
    .clk(spi_vif.clk),
    .rst_n(spi_vif.rst_n),
    .MISO(spi_vif.MISO),
    .rx_valid(spi_vif.rx_valid),
    .rx_data(spi_vif.rx_data)
);
assign cs_s = DUT.cs;
assign ns_s=DUT.ns;
assign received_address=DUT.received_address;
initial begin
    uvm_config_db #(virtual spi_if) :: set (null ,"uvm_test_top","spi_if",spi_vif);
    uvm_config_db #(virtual spi_gold_if) :: set (null , "uvm_test_top" , "spi_gold_if" , gold_vif);
    run_test("spi_test");
end
endmodule
```

SPI_TEST

spi_test.sv > SystemVerilog - Language Support > {} spi_test_pkg > spi_test > build_phase

```
1   package spi_test_pkg;
2   import uvm_pkg::*;
3   import spi_env_pkg::*;
4   import spi_config_obj::*;
5   import sequences::*;
6   `include "uvm_macros.svh"
7   class spi_test extends uvm_test;
8   `uvm_component_utils(spi_test)
9   spi_env env;
10  spi_config spi_cfg;
11  spi_config spi_gold_cfg;
12  main_sequence spi_main_seq;
13  reset_sequence spi_reset_seq;
14
15  function new(string name="spi_test",uvm_component parent = null);
16  super.new (name,parent);
17  endfunction
18
19  function void build_phase(uvm_phase phase);
20  super.build_phase(phase);
21  env =spi_env::type_id::create("env",this);
22  spi_main_seq =main_sequence::type_id::create("spi_main_seq",this);
23  spi_reset_seq= reset_sequence::type_id::create("spi_reset_seq",this);
24  spi_cfg =spi_config::type_id::create("spi_cfg");
25  spi_gold_cfg = spi_config::type_id::create("spi_gold_cfg");
26  if (!uvm_config_db#(virtual spi_if)::get(this,"","spi_if",spi_cfg.spi_config_vif))begin
27      `uvm_fatal("build_phase","test - unable to get VIF")
28  end
29  if (!uvm_config_db#(virtual spi_gold_if)::get(this,"","spi_gold_if",spi_gold_cfg.spi_gold_config_vif))begin
30      `uvm_fatal("build_phase","test - unable to get gold VIF")
31  end
32   uvm_config_db #(spi_config)::set(this,"*","CFG",spi_cfg); /// change from get to set
33   uvm_config_db #(spi_config)::set(this,"*","CFG_GOLD",spi_gold_cfg);
34  endfunction
35
36  task run_phase(uvm_phase phase);
37  super.run_phase(phase);
38  phase.raise_objection(this);
39   `uvm_info("run_phase","reset asserted",UVM_LOW);
40  spi_reset_seq.start(env.agt.sqr);
41  `uvm_info("run_phase","reset de asserted ",UVM_LOW);
42  spi_main_seq.start(env.agt.sqr);
43  `uvm_info("run_phase","main_sequence started",UVM_LOW);
44  phase.drop_objection(this);
45  `uvm_info("run_phase","main sequence finished",UVM_LOW);
46  endtask:run_phase
47  endclass
48  endpackage
```

SPI Sequences

spi_sequences.sv > SystemVerilog - Language Support > {} sequences > main_sequence > body

```
1   package sequences;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import spi_seq_item_pkg::*;
5   import spi_shared::*;
6   class reset_sequence extends uvm_sequence #(spi_seq_item);
7   `uvm_object_utils(reset_sequence)
8
9    function new (string name = "reset_sequence");
10   super.new(name);
11   endfunction
12
13  task body();
14  spi_seq_item req;
15  req = spi_seq_item::type_id::create("req");
16  start_item(req);
17  req.SS_n=1;
18  req.rst_n=0;
19  req.MOSI=0;
20  req.tx_data=0;
21  req.tx_valid=0;
22  finish_item(req);
23  endtask
24  endclass
25  ////////////////////////////
26  class main_sequence extends  uvm_sequence #(spi_seq_item);
27   `uvm_object_utils(main_sequence);
28   spi_seq_item req;
29   function new(string name= "main_sequence");
30       super.new(name);
31   endfunction
32   task body ;
33   req= spi_seq_item :: type_id :: create("req");
34   repeat(10000) begin
35   start_item(req);
36   assert (req.randomize() );
37   finish_item(req);
38   end
39   endtask
40
41   endclass
42
43
44  endpackage
```

SPI environment

spi_env.sv > SystemVerilog - Language Support > {} spi_env_pkg > spi_env

```
package spi_env_pkg;
import uvm_pkg::*;
import spi_coverage_pkg::*;
import spi_scoreboard_pkg::*;
import spi_agent_pkg::*;
`include "uvm_macros.svh"

class spi_env extends uvm_env;
`uvm_component_utils(spi_env)
spi_scoreboard sb;
spi_agent agt;
spi_coverage cov;

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

function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  agt=spi_agent::type_id::create("agt",this);
  cov=spi_coverage::type_id::create("cov",this);
  sb=spi_scoreboard::type_id::create("sb",this);
  endfunction : build_phase

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

```

SPI Shared pkg

spi_shared_pkg.sv > SystemVerilog - Language Support > {} spi_shared > received_address

```
package spi_shared;
typedef enum  {IDLE,CHK_CMD ,WRITE,READ_ADD , READ_DATA } state_e;
 state_e cs_s,ns_s;
 bit received_address;
endpackage
```

SPI Scoreboard

spi_scoreboard.sv > SystemVerilog - Language Support > {} spi_scoreboard_pkg > spi_scoreboard

```
package spi_scoreboard_pkg;
import uvm_pkg::*;
import spi_seq_item_pkg::*;
`include "uvm_macros.svh"
class spi_scoreboard extends uvm_scoreboard;
`uvm_component_utils(spi_scoreboard)

uvm_analysis_export #(spi_seq_item) sb_export;
uvm_tlm_analysis_fifo #(spi_seq_item) sb_fifo;
spi_seq_item seq_item_sb;
int error_count=0;
int correct_count=0;

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

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

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

task run_phase(uvm_phase phase);
super.run_phase(phase);
forever begin
    sb_fifo.get(seq_item_sb);
    if(seq_item_sb.MISO != seq_item_sb.MISO_ref||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",$sformatf("failed  %s ",seq_item_sb.convert2string(),UVM_HIGH));
        error_count++;
    end
    else begin
        `uvm_info("run_phase",$sformatf("by dut  :%s",seq_item_sb.convert2string()),UVM_HIGH);
        correct_count++;
    end
end
endtask
function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("SLAVE", $sformatf("===== SLAVE Scoreboard Report =====\nCorrect Results: %0d\nErrors: %0d",
            correct_count, error_count), UVM_MEDIUM);
        endfunction

endclass
endpackage
```

SPI Coverage

spi_coverage.sv > ...

```
package spi_coverage_pkg;
import uvm_pkg::*;
import spi_seq_item_pkg::*;
import spi_shared::*;
class spi_coverage extends uvm_component;
`include "uvm_macros.svh"
`uvm_component_utils(spi_coverage)
uvm_analysis_export #(spi_seq_item) cov_export;
uvm_tlm_analysis_fifo #(spi_seq_item) cov_fifo;
spi_seq_item cov_txn;

covergroup CovCode;
c1:coverpoint cov_txn.rx_data[9:8]{
bins all_values [] = {2'b00,2'b01,2'b10,2'b11}; // 00,01,10,11
bins rx_data_trans[] = ( 0 => 1) , ( 1=> 3) , (1 => 0) , ( 2 => 3) , ( 2=> 0) , (3 => 1), ( 3 => 2);
}

SS_n_c2:coverpoint cov_txn.SS_n {
    bins normal_transaction = (1 => 0[*13] => 1) ;
    bins extended_bits_transaction = (1 => 0[*23] => 1) ;
}
MOSI_c3:coverpoint cov_txn.MOSI {
    bins write_address = (0 => 0 => 0) ;//write address command(000)
    bins write_data = (0 => 0 => 1) ; //write data command(001)
    bins read_address = (1 => 1 => 0) ;//read address command(110)
    bins read_data = (1 => 1 => 1) ;//read data command(111)
}
//cross coverage between SS_n and MOSI(only legal scenarios)
 SS_n_MOSI_cross:cross SS_n_c2 , MOSI_c3 {
   // option.cross_auto_bin_max=0;
    ignore_bins wrong_normal_read_data =binsof(SS_n_c2.normal_transaction) && binsof(MOSI_c3.read_data) ;
    ignore_bins wrong_extended_write = binsof(SS_n_c2.extended_bits_transaction) && binsof(MOSI_c3.write_address) ;
    ignore_bins wrong_extended_write_data = binsof(SS_n_c2.extended_bits_transaction) && binsof(MOSI_c3.write_data);
    ignore_bins wrong_extended_read_add=binsof(SS_n_c2.extended_bits_transaction) && binsof(MOSI_c3.read_address);
      }
endgroup

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

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

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

task run_phase(uvm_phase phase);
super.run_phase(phase);
forever begin
    cov_fifo.get(cov_txn);
    CovCode.sample();
end
endtask
```

SPI Agent

spi_agent.sv > SystemVerilog - Language Support > spi_agent_pkg > spi_agent > spi_cfg

```
1   package spi_agent_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import spi_seq_item_pkg::*;
5   import spi_config_obj::*;
6   import spi_driver::*;
7   import spi_sequencer_pkg::*;
8   import spi_monitor_pkg::*;
9   class spi_agent extends uvm_agent;
10  `uvm_component_utils(spi_agent)
11
12  spi_sequencer sqr;
13  spi_driver drv;
14  spi_monitor mon;
15  spi_config spi_cfg;
16  spi_config spi_gold_cfg;
17  uvm_analysis_port #(spi_seq_item) agt_ap;
18
19  function new(string name = "spi_agent",uvm_component parent = null);
20  super.new(name,parent);
21  endfunction
22
23  function void build_phase(uvm_phase phase);
24  super.build_phase(phase);
25  if(!uvm_config_db #(spi_config)::get(this,"","CFG",spi_cfg))begin
26  `uvm_fatal("build_phase","unable to get CFG object")
27  end
28
29  if(!uvm_config_db #(spi_config)::get(this,"","CFG_GOLD",spi_gold_cfg))begin
30  `uvm_fatal("build_phase","unable to get CFG_GOLD object")
31  end
32
33  sqr = spi_sequencer::type_id::create("sqr",this);
34  drv = spi_driver::type_id::create("drv",this);
35  mon = spi_monitor::type_id::create("mon",this);
36  agt_ap = new("agt_ap",this);
37  endfunction
38
39  function void connect_phase(uvm_phase phase);
40  super.connect_phase(phase);
41  drv.spi_vif = spi_cfg.spi_config_vif;
42  mon.spi_vif = spi_cfg.spi_config_vif;
43
44  drv.spi_gold_vif = spi_gold_cfg.spi_gold_config_vif;
45  mon.spi_gold_vif = spi_gold_cfg.spi_gold_config_vif;
46
47  drv.seq_item_port.connect(sqr.seq_item_export);
48  mon.mon_ap.connect(agt_ap);
49  endfunction
50  endclass
51  endpackage
```

SPI Seq Item

spi_seq_item.sv > SystemVerilog - Language Support > spi_seq_item_pkg > spi_seq_item > post_randomize

```
package spi_seq_item_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import spi_shared::*;
class spi_seq_item extends uvm_sequence_item;
    `uvm_object_utils(spi_seq_item);
    rand logic rst_n;
    rand logic MOSI;
    rand logic SS_n;
    rand logic tx_valid;
    rand logic [7:0] tx_data;
    logic [9:0] rx_data, rx_data_ref;
    logic rx_valid, rx_valid_ref, MISO, MISO_ref;

    rand bit [10:0] mosi_bus;
    static bit [1:0] cmd_bit = 2'b00;
    int clk_count;

    function new(string name = "spi_seq_item");
        super.new(name);
        tx_data=0;
    endfunction

    constraint rst_n_c{
        rst_n dist {0:/1, 1:/99};
    }

    constraint SS_n_c{
    if(cs_s==READ_DATA){
        if(clk_count == 23) SS_n == 1;
        else SS_n == 0;
    }
    else{
        if(clk_count == 13) SS_n == 1;
        else SS_n == 0;
    }
}

    constraint tx_valid_c{
        if(cs_s==READ_DATA && clk_count > 13 && clk_count < 23) tx_valid == 1;
        else tx_valid == 0;
    }

    constraint MOSI_c{
       if (clk_count > 0 && clk_count < 12) MOSI == mosi_bus[11 - clk_count];
    }

    function void pre_randomize();
        if(clk_count==0) mosi_bus.rand_mode(1);
        else mosi_bus.rand_mode(0);

        if(mosi_bus[10:8] == 3'b111 && clk_count == 12) tx_data.rand_mode(1);
        else tx_data.rand_mode(0);

    endfunction

    constraint mosi_bus_const{
        mosi_bus[10] == cmd_bit[1];
        mosi_bus[9:8] == cmd_bit;
    }

    function void post_randomize();
        if(SS_n || !rst_n) clk_count = 0;
        else clk_count++;
        if(!rst_n) cmd_bit = 0;
        else if(clk_count == 0) cmd_bit++;
    endfunction
```

```
68    function string convert2string();
69        return $sformatf("%s rst_n = 0b%0b , MOSI = 0b%0b , SS_n = 0b%0b , tx_valid = 0b%0b , MISO = 0b%0b , tx_data = 0x%02h , rx_valid = 0b%0b , rx_data = 0x%03h",
70            super.convert2string(), rst_n, MOSI, SS_n, tx_valid, tx_data, MISO, rx_valid, rx_data);
71    endfunction
72    function string convert2string_stimulus();
73        return $sformatf("%s rst_n = 0b%0b , MOSI = 0b%0b , SS_n = 0b%0b , tx_valid = 0b%0b , tx_data = 0x%02h",
74            super.convert2string(), rst_n, MOSI, SS_n, tx_valid, tx_data);
75    endfunction
76
77  endclass
78
79
80
81  endpackage
```

SPI Sequencer

spi_sequencer.sv > ...

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

SVA Module

Assertions.sv > SystemVerilog - Language Support > spi_sv > p_rxvalid

```
1   import spi_shared::*;
2   module spi_sv (
3       input logic clk,
4       input logic rst_n,
5       input logic MISO,
6       input logic rx_valid,
7       input logic [9:0] rx_data);
8
9    property reset_low_outputs;
10     @(posedge clk)
11       (!rst_n) |=> (MISO == 1'b0 && rx_valid == 1'b0 && rx_data == 10'b0);
12   endproperty
13   assert property (reset_low_outputs);
14   cover property  (reset_low_outputs);
15
16  property p_rxvalid;
17          @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n)
18          (cs_s == CHK_CMD && ({ $past(spi_vif.MOSI,2), $past(spi_vif.MOSI,1), spi_vif.MOSI } inside {3'b000, 3'b001, 3'b110, 3'b111})
19           && spi_vif.SS_n == 1'b0)  |=> ##10 (spi_vif.rx_valid && $rose(spi_vif.SS_n)[->1]);
20  endproperty
21  assert property (p_rxvalid);
22  cover property (p_rxvalid);
23  endmodule
24
```

SPI CFG

spi_config_obj.sv > ...

```
package spi_config_obj;
import uvm_pkg::*;
`include "uvm_macros.svh"
class spi_config extends uvm_object;
`uvm_object_utils(spi_config)
virtual spi_if spi_config_vif;
virtual spi_gold_if spi_gold_config_vif;
function new(string name="spi_config");
super.new(name);
endfunction
endclass
endpackage
```

SPI Monitor

spi_monitor.sv > SystemVerilog - Language Support > {} spi_monitor_pkg > spi_monitor > run_phase

```
package spi_monitor_pkg;
import spi_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class spi_monitor extends uvm_monitor;
`uvm_component_utils(spi_monitor)

virtual spi_if spi_vif;
virtual spi_gold_if spi_gold_vif;
uvm_analysis_port#(spi_seq_item) mon_ap;
spi_seq_item rsp_seq_item;

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

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

task run_phase(uvm_phase phase);
super.run_phase(phase);
forever begin
 rsp_seq_item=spi_seq_item::type_id::create("rsp_seq_item");
 @(negedge spi_vif.clk);
 rsp_seq_item.rst_n = spi_vif.rst_n;
 rsp_seq_item.MOSI = spi_vif.MOSI;
 rsp_seq_item.MISO = spi_vif.MISO;
 rsp_seq_item.SS_n = spi_vif.SS_n;
 rsp_seq_item.rx_data = spi_vif.rx_data;
 rsp_seq_item.rx_valid = spi_vif.rx_valid;
 rsp_seq_item.tx_data = spi_vif.tx_data;
 rsp_seq_item.tx_valid = spi_vif.tx_valid;
 // gold
 rsp_seq_item.rx_data_ref=spi_gold_vif.rx_data;
 rsp_seq_item.rx_valid_ref=spi_gold_vif.rx_valid;
 rsp_seq_item.MISO_ref=spi_gold_vif.MISO;
 mon_ap.write(rsp_seq_item);
 `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH);
end
endtask
endclass
endpackage
```

SPI Driver

spi_driver.sv > SystemVerilog - Language Support > {} spi_driver > spi_driver > run_phase

```
1   package spi_driver;
2   `include "uvm_macros.svh"
3   import uvm_pkg::*;
4   import spi_seq_item_pkg::*;
5   class spi_driver extends uvm_driver #(spi_seq_item);
6   `uvm_component_utils(spi_driver)
7   spi_seq_item stim_seq_item;
8   virtual spi_if spi_vif;
9   virtual spi_gold_if spi_gold_vif;
10
11
12  function new(string name = "spi_driver",uvm_component parent =null);
13  super.new(name,parent);
14  endfunction
15
16  task run_phase (uvm_phase phase);
17   super.run_phase(phase);
18   forever begin
19   stim_seq_item=spi_seq_item::type_id::create("stim_seq_item");
20   seq_item_port.get_next_item(stim_seq_item);
21   spi_vif.MOSI = stim_seq_item.MOSI;
22   spi_vif.SS_n = stim_seq_item.SS_n;
23   spi_vif.tx_data = stim_seq_item.tx_data;
24   spi_vif.tx_valid = stim_seq_item.tx_valid;
25   spi_vif.rst_n = stim_seq_item.rst_n;
26   //for gold
27   spi_gold_vif.MOSI = stim_seq_item.MOSI;
28   spi_gold_vif.SS_n = stim_seq_item.SS_n;
29   spi_gold_vif.tx_data = stim_seq_item.tx_data;
30   spi_gold_vif.tx_valid = stim_seq_item.tx_valid;
31   spi_gold_vif.rst_n = stim_seq_item.rst_n;
32   @(negedge spi_vif.clk);
33   seq_item_port.item_done();
34   `uvm_info("run_phase",stim_seq_item.convert2string_stimulus(),UVM_HIGH);
35   end
36   endtask
37   endclass
38   endpackage
```

Do File

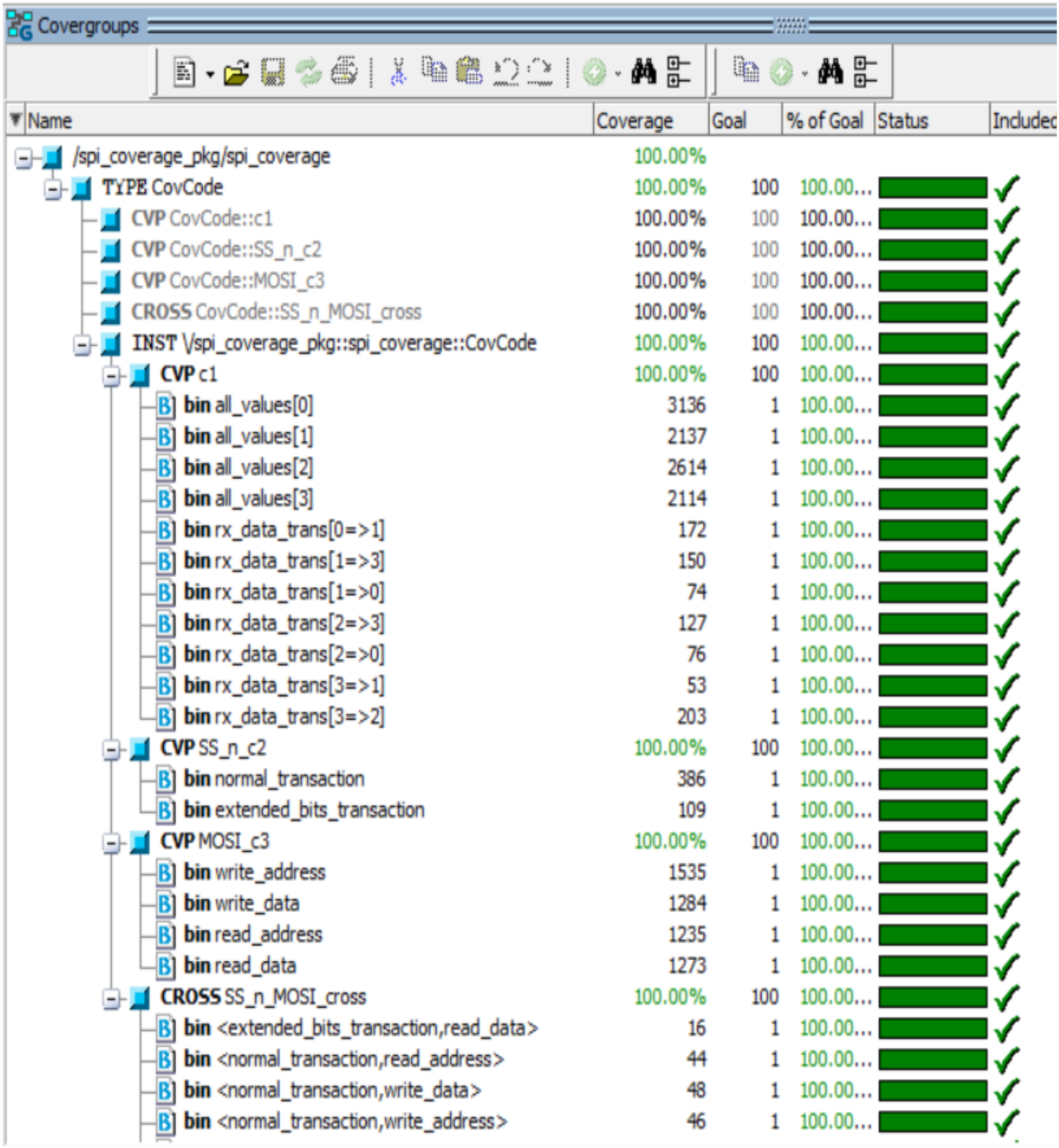
run.do

```
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.spi_top -classdebug -uvmcontrol=all -sv_seed random -cover
run 0
add wave -position insertpoint \
sim:/spi_top/Golden/MOSI \
sim:/spi_top/DUT/MOSI \
sim:/spi_top/Golden/MISO \
sim:/spi_top/DUT/MISO \
sim:/spi_top/Golden/SS_n \
sim:/spi_top/DUT/SS_n \
sim:/spi_top/Golden/tx_valid \
sim:/spi_top/DUT/tx_valid \
sim:/spi_top/Golden/rx_valid \
sim:/spi_top/DUT/rx_valid \
sim:/spi_top/Golden/rx_data \
sim:/spi_top/DUT/rx_data \
sim:/spi_top/Golden/cs \
sim:/spi_top/DUT/cs \
sim:/spi_top/Golden/ns \
sim:/spi_top/DUT/ns \
sim:/spi_top/Golden/rst_n \
sim:/spi_top/Golden/clk \
sim:/spi_top/Golden/tx_data
add wave -position insertpoint \
sim:/spi_shared::cs_s \
sim:/spi_shared::ns_s
add wave  -position insertpoint \
sim:/uvm_root/uvm_test_top/env/sb/correct_count \
sim:/uvm_root/uvm_test_top/env/sb/error_count
run -all
coverage exclude -src SPI_slave.sv -line 39 -code s
coverage exclude -src SPI_slave.sv -line 38 -code b
coverage exclude -src SPI_slave.sv -line 81 -code b
coverage save SPI.ucdb -onexit -du work.SLAVE
coverage report -detail -cvg -directive -comments -output SPI_fcover_report.txt {}
/quit -sim
vcover report SPI.ucdb -details -annotate -all -output SPI_co.txt
```

src_files.list

```
spi_interface.sv
spi_shared_pkg.sv
golden_interface.sv
+define+SIM SPI_slave.sv
SPI.sv
spi_seq_item.sv
spi_config_obj.sv
spi_driver.sv
spi_monitor.sv
spi_sequencer.sv
spi_agent.sv
spi_coverage.sv
spi_scoreboard.sv
spi_env.sv
spi_sequences.sv
spi_test.sv
Assertions.sv
spi_top.sv
```

Functional Coverage Report



All Reports Exported Will Be in Project File for Details

Assertions State

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0
/sequences::main_sequence::body/#ublk#146550787#34/immed__36	Immediate	SVA	on	0	1
/spi_top/DUT/assert__p_IDLE_to_CHK_CMD	Concurrent	SVA	on	0	1
/spi_top/DUT/assert__p_CHK_CMD_to_WRITE	Concurrent	SVA	on	0	1
/spi_top/DUT/assert__p_CHK_CMD_to_READ_ADD	Concurrent	SVA	on	0	1
/spi_top/DUT/assert__p_CHK_CMD_to_READ_DATA	Concurrent	SVA	on	0	1
/spi_top/DUT/assert__p_WRITE_to_IDLE	Concurrent	SVA	on	0	1
/spi_top/DUT/assert__p_READ_ADD_to_IDLE	Concurrent	SVA	on	0	1
/spi_top/DUT/assert__p_READ_DATA_to_IDLE	Concurrent	SVA	on	0	1
/spi_top/DUT/assert__p_counter_decrement	Concurrent	SVA	on	0	1
/spi_top/DUT/sva_inst/assert__reset_low_outputs	Concurrent	SVA	on	0	1
/spi_top/DUT/sva_inst/assert__p_rxvalid	Concurrent	SVA	on	0	1

Assertions Coverage

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included
/spi_top/DUT/cover__p_counter_decrement	SVA	✓	Off	7212	1	Unlimit...	1	100%		✓
/spi_top/DUT/cover__p_READ_DATA_to_IDLE	SVA	✓	Off	107	1	Unlimit...	1	100%		✓
/spi_top/DUT/cover__p_READ_ADD_to_IDLE	SVA	✓	Off	126	1	Unlimit...	1	100%		✓
/spi_top/DUT/cover__p_WRITE_to_IDLE	SVA	✓	Off	337	1	Unlimit...	1	100%		✓
/spi_top/DUT/cover__p_CHK_CMD_to_READ_DATA	SVA	✓	Off	122	1	Unlimit...	1	100%		✓
/spi_top/DUT/cover__p_CHK_CMD_to_READ_ADD	SVA	✓	Off	152	1	Unlimit...	1	100%		✓
/spi_top/DUT/cover__p_CHK_CMD_to_WRITE	SVA	✓	Off	383	1	Unlimit...	1	100%		✓
/spi_top/DUT/cover__p_IDLE_to_CHK_CMD	SVA	✓	Off	663	1	Unlimit...	1	100%		✓
/spi_top/DUT/sva_inst/cover__p_rxvalid	SVA	✓	Off	275	1	Unlimit...	1	100%		✓
/spi_top/DUT/sva_inst/cover__reset_low_outputs	SVA	✓	Off	105	1	Unlimit...	1	100%		✓

Code Coverage

Statements - by instance (/spi_top/DUT) | Statement | ✓ | ✗ | E

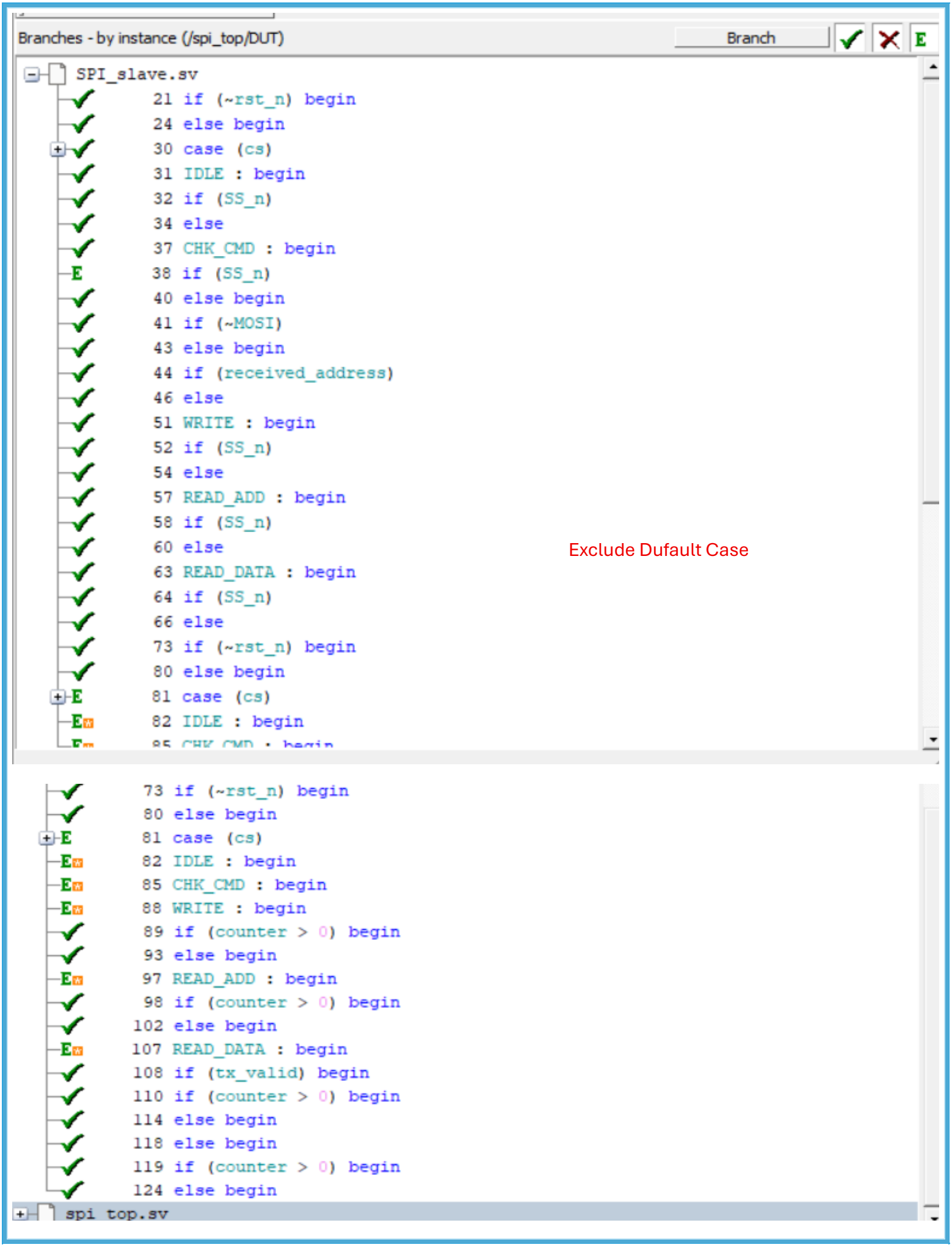
SPI_slave.sv

```
20 always @(posedge clk) begin
22 cs <= IDLE;
25 cs <= ns;
29 always @(*) begin
33 ns = IDLE;
35 ns = CHK_CMD;
E  39 ns = IDLE;
42 ns = WRITE;
45 ns = READ_DATA;
47 ns = READ_ADD; /// edit 1 replace with another
53 ns = IDLE;
55 ns = WRITE;
59 ns = IDLE;
61 ns = READ_ADD;
65 ns = IDLE;
67 ns = READ_DATA;
72 always @(posedge clk) begin
74 rx_data <= 0;
75 rx_valid <= 0;
76 counter<=0; // counter
77 received_address <= 0;
78 MISO <= 0;
83 rx_valid <= 0;
86 counter <= 10;
90 rx_data[counter-1] <= MOSI;
91 counter <= counter - 1;
```

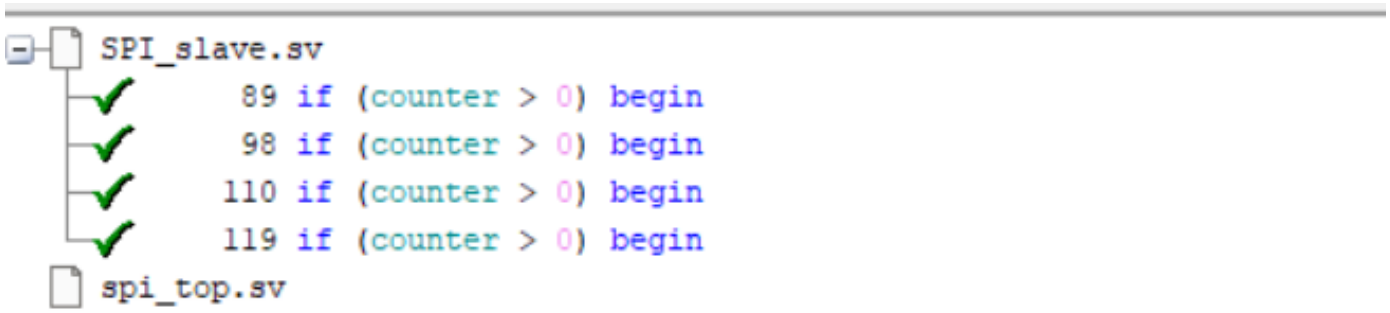
I Excluded IDEAL when cs equal CHK_CMD because Constraints Prevent That

```
91 counter <= counter - 1;
94 rx_valid <= 1;
99 rx_data[counter-1] <= MOSI;
100 counter <= counter - 1;
103 rx_valid <= 1;
104 received_address <= 1;
109 rx_valid<=0;
111 MISO <= tx_data[counter-1];
112 counter <= counter - 1;
115 received_address <= 0;
120 rx_data[counter-1] <= MOSI;
121 counter <= counter - 1;
122 rx_valid<=0;
125 rx_valid <= 1;
126 counter <= 9;
```

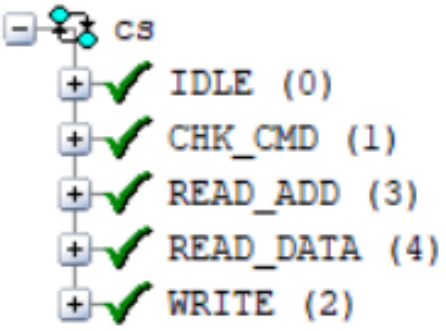
spi_top.sv



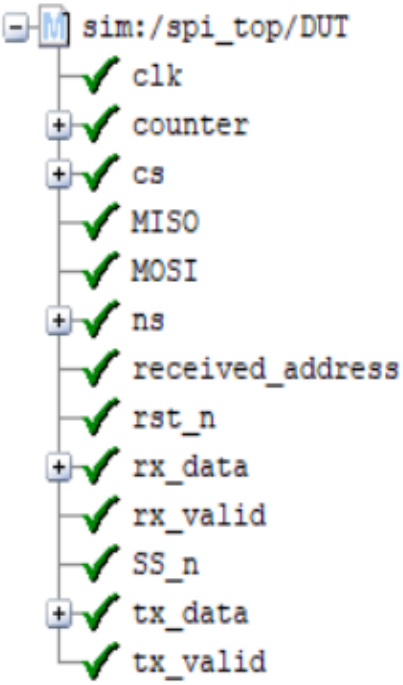
Exclude Dufault Case



FSMs - by instance (/spi_top/DUT)



Toggles - by instance (/spi_top/DUT)



Code Coverage Report Summary

```
ASSERTION RESULTS:
------------------------------------------------------------------------
Name                 File(Line)                    Failure      Pass
                                                   Count        Count
------------------------------------------------------------------------
/\spi_top#DUT /assert__p_counter_decrement
                     SPI_slave.sv(180)                   0          1
/\spi_top#DUT /assert__p_READ_DATA_to_IDLE
                     SPI_slave.sv(179)                   0          1
/\spi_top#DUT /assert__p_READ_ADD_to_IDLE
                     SPI_slave.sv(178)                   0          1
/\spi_top#DUT /assert__p_WRITE_to_IDLE
                     SPI_slave.sv(177)                   0          1
/\spi_top#DUT /assert__p_CHK_CMD_to_READ_DATA
                     SPI_slave.sv(176)                   0          1
/\spi_top#DUT /assert__p_CHK_CMD_to_READ_ADD
                     SPI_slave.sv(175)                   0          1
/\spi_top#DUT /assert__p_CHK_CMD_to_WRITE
                     SPI_slave.sv(174)                   0          1
/\spi_top#DUT /assert__p_IDLE_to_CHK_CMD
                     SPI_slave.sv(173)                   0          1
/\spi_top#DUT /sva_inst/assert__p_rxvalid
                     Assertions.sv(21)                   0          1
/\spi_top#DUT /sva_inst/assert__reset_low_outputs
                     Assertions.sv(13)                   0          1

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

Feature	Assertion
An assertion ensures that whenever reset is asserted, the outputs (MISO, rx_valid, and rx_data) are all low.	property reset_low_outputs; @(posedge clk) (!rst_n) |=> (MISO == 1'b0 && rx_valid == 1'b0 && rx_data == 10'b0); endproperty
An assertion checks that after any valid command sequence (write_add_seq (000), write_data_seq (001), read_add_seq(110), or read_data_seq(111)), the rx_valid signal must assert exactly after 10 cycles and the SS_n should eventually after the 10 cycles to close communication.	property p_rxvalid; @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n) (cs_s == CHK_CMD && ({ $past(spi_vif.MOSI,2), $past(spi_vif.MOSI,1), spi_vif.MOSI } inside {3'b000, 3'b001, 3'b110, 3'b111}) && spi_vif.SS_n == 1'b0) |=> ##10 (spi_vif.rx_valid && $rose(spi_vif.SS_n)[->1]); endproperty
IDLE to CHK_CMD	property p_IDLE_to_CHK_CMD; @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n) (cs == IDLE && spi_vif.SS_n == 1'b0) |=> (cs == CHK_CMD); endproperty
CHK_CMD to WRITE	property p_CHK_CMD_to_WRITE; @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n) (cs == CHK_CMD && spi_vif.MOSI == 1'b0 && spi_vif.SS_n == 1'b0) |=> (cs == WRITE); endproperty
CHK_CMD to READ_ADD	property p_CHK_CMD_to_READ_ADD; @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n) (cs == CHK_CMD && spi_vif.MOSI == 1'b1 && received_address == 1'b0 && spi_vif.SS_n == 1'b0) |=> (cs == READ_ADD); endproperty
CHK_CMD to READ_DATA	property p_CHK_CMD_to_READ_DATA; @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n) (cs == CHK_CMD && spi_vif.MOSI == 1'b1 && received_address == 1'b1 && spi_vif.SS_n == 1'b0) |=> (cs == READ_DATA); endproperty
READ_ADD to IDLE	property p_READ_ADD_to_IDLE; @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n) (cs == READ_ADD && spi_vif.SS_n == 1'b1) |=> (cs == IDLE); endproperty

READ_DATA to IDLE	**property p_READ_DATA_to_IDLE; @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n) (cs == READ_DATA && spi_vif.SS_n == 1'b1) |=> (cs == IDLE); endproperty**
Counter Decrement at 3 States WRITE READ_DATA AND READ_ADD	**property p_counter_decrement; @(posedge spi_vif.clk) disable iff (!spi_vif.rst_n) ((cs == WRITE || cs == READ_ADD|| cs==READ_DATA) && counter > 0) |=> (counter == $past(counter) - 1); endproperty**

Bug Report

```
CHK_CMD : begin
    if (SS_n)
        ns = IDLE;
    else begin
        if (~MOSI)
            ns = WRITE;
        else begin
            if (received_address)
                ns = READ_DATA;
            else
                ns = READ_ADD; /// edit 1 replace with another
        end
    end
end
```

1- **I make when received address is high to go to state Read data not read address**

```
always @(posedge clk) begin
    if (~rst_n) begin
        rx_data <= 0;
        rx_valid <= 0;
        counter<=0; // counter
        received_address <= 0;
        MISO <= 0;
    end
```

2-I Add Counter At reset

```
    else begin
        rx_valid <= 1;
        counter <= 9;
    end
end
```

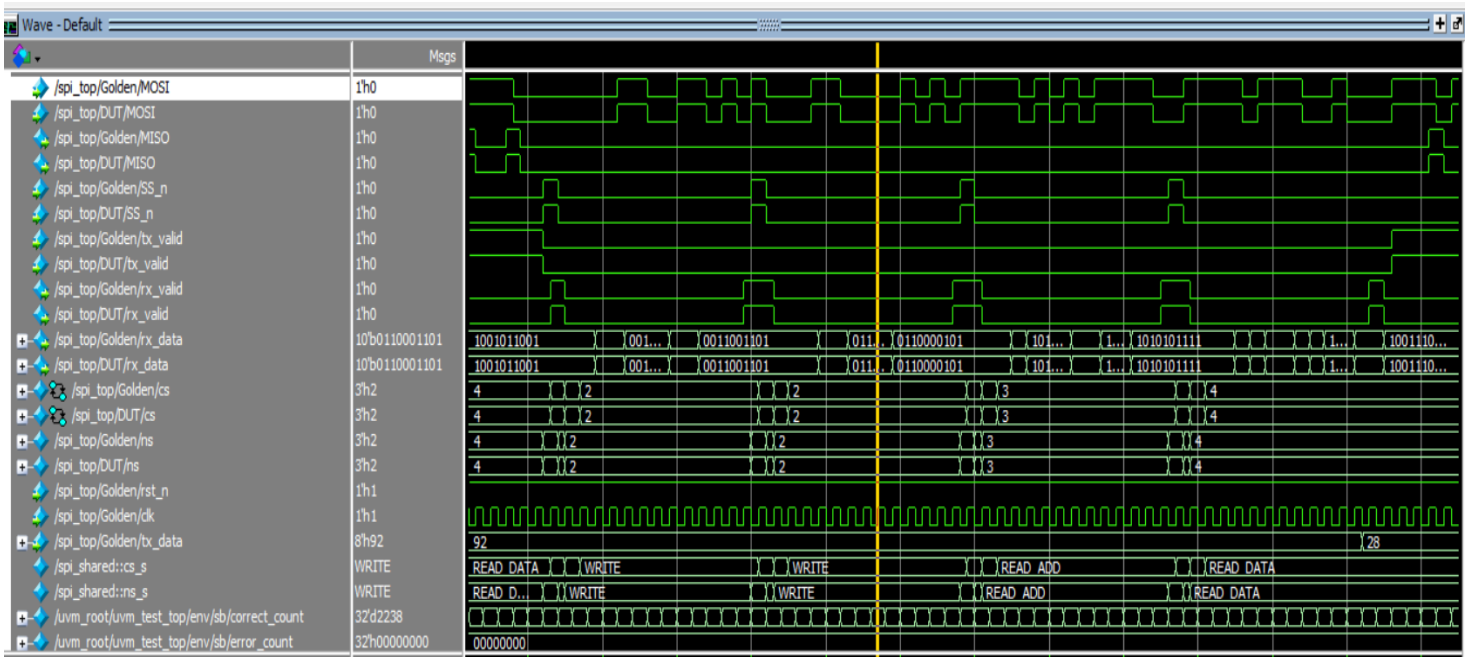
Make counter 9 not 8 before start to capture to MISO

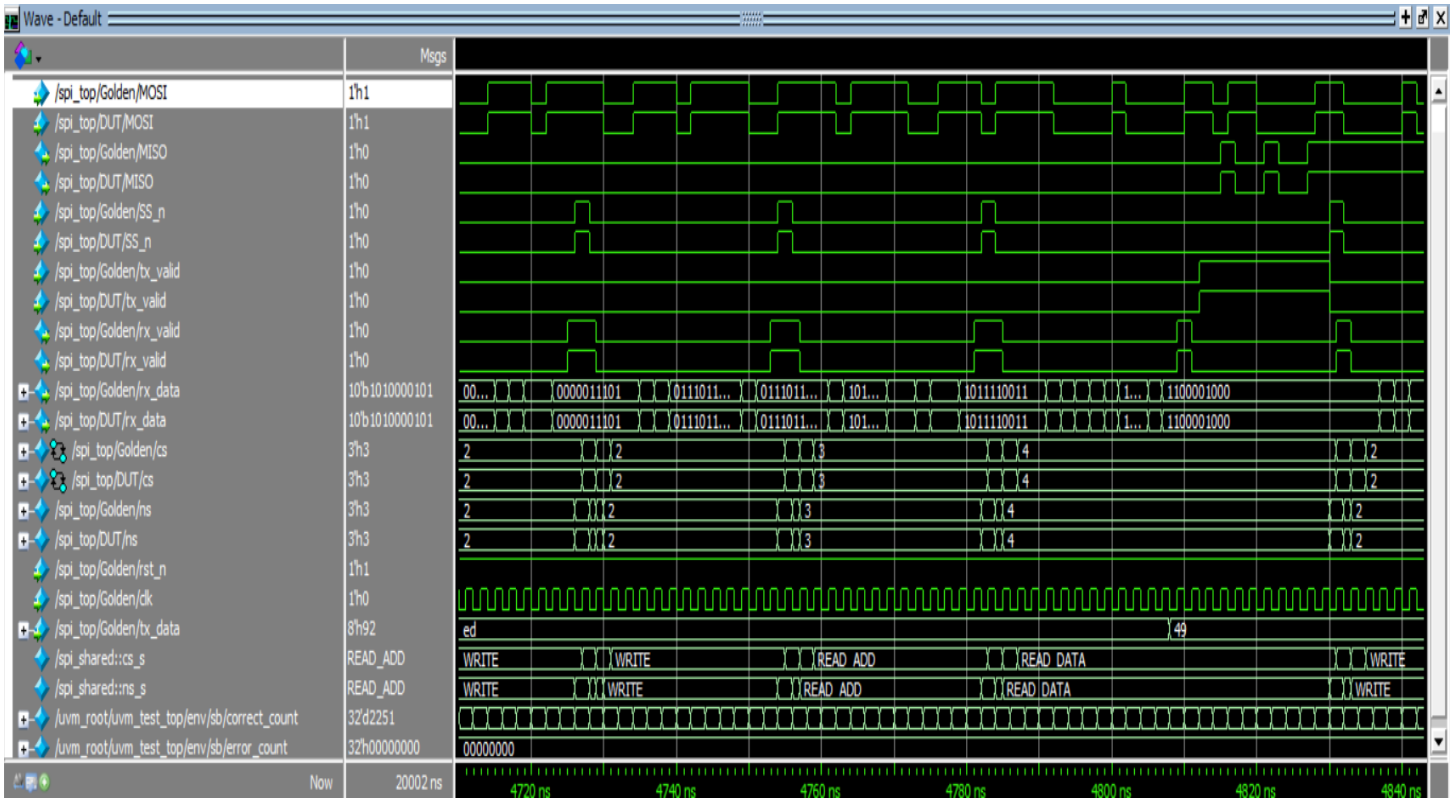
```
    end
    else begin
        if (counter > 0) begin
            rx_data[counter-1] <= MOSI;
            counter <= counter - 1;
            rx_valid<=0;
```

ADD RX_VALID EQUAL ZERO HERE TO PREVENT DUMY DATA TO CHANGE THE COMMAND

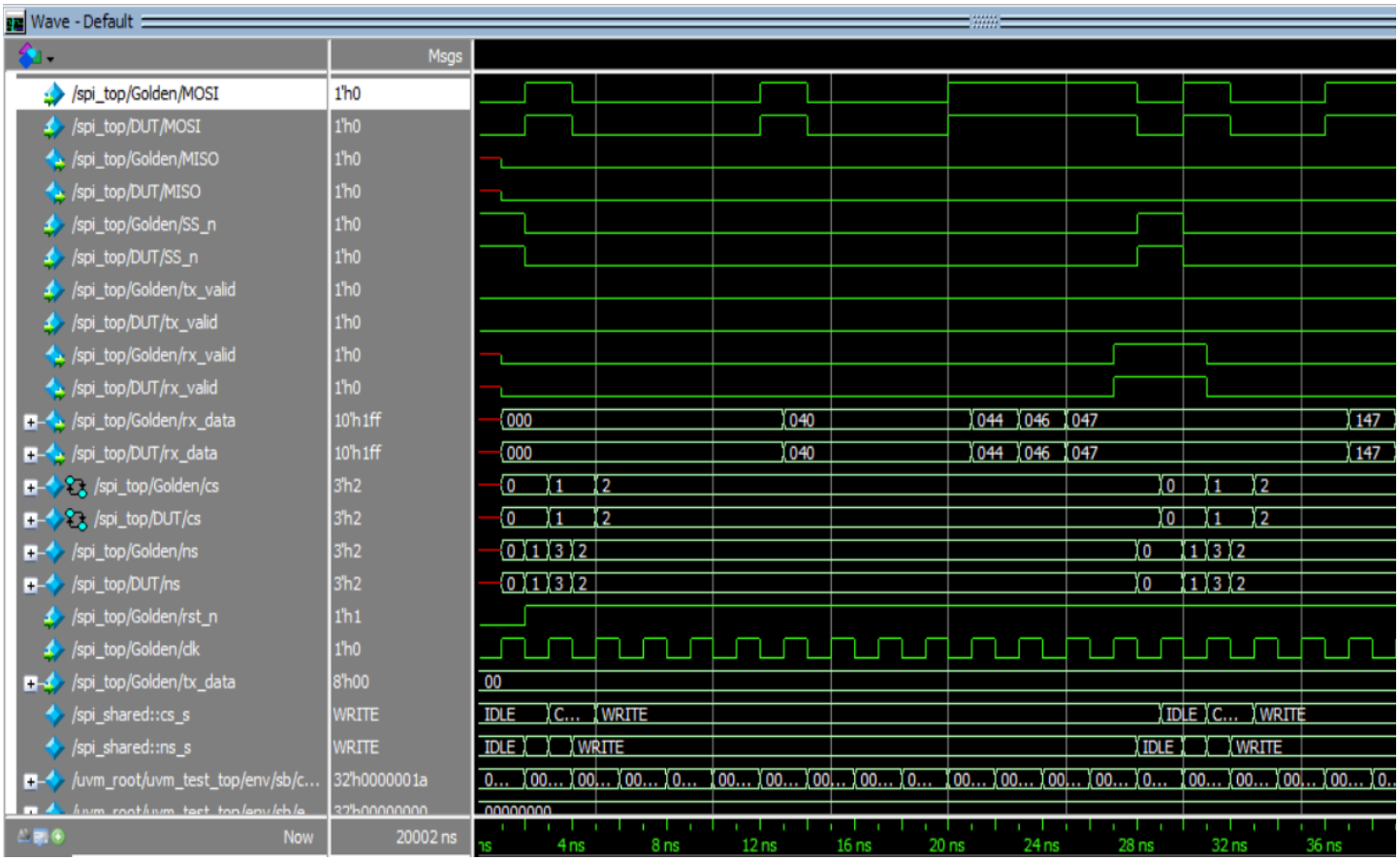
- **Waveforms and Output Summary**

Main_seq





RST_Seq



Output Summary

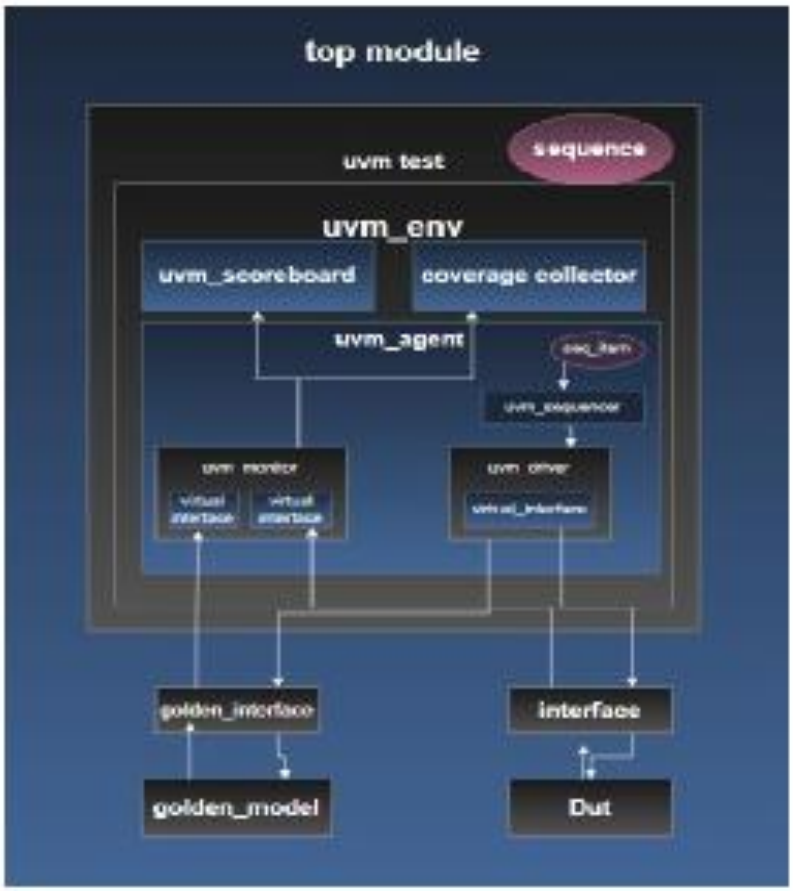
```
# UVM_INFO spi_scoreboard.sv(45) @ 20002: uvm_test_top.env.sb [SLAVE] ===== SLAVE Scoreboard Report =====
# Correct Results: 10001
# Errors: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    9
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [SLAVE]     1
# [TEST_DONE]     1
# [run_phase]     4
# ** Note: $finish    : C:/questasim64_2024.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 20002 ns  Iteration: 61  Instance: /spi_top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2024.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```

Part 2: UVM Environment for Single-Port RAM

- **Verification plan:**

	Feature/Scenario	Description	Verification Method	Implemented In	
2	Reset Functionality	Verify that asserting rst_n resets dout and tx_valid to 0	Directed (RAM_reset_sequence) + SVA (reset_low_check)	RAM_test / RAM_sva	
3	Write Operation	Verify correct behavior for write address and write data transactions	Constrained Random (RAM_write_sequence)	RAM_env / RAM_driver / RAM_monitor / RAM_scoreboard	
4	Read Operation	Verify correct behavior for read address and read data transactions	Constrained Random (RAM_read_sequence)	RAM_env / RAM_driver / RAM_monitor / RAM_scoreboard	
5	Read/Write Mixed Operation	Verify alternating read/write transactions	Constrained Random (RAM_read_write_sequence)	RAM_env / RAM_driver / RAM_monitor / RAM_scoreboard	
6	tx_valid Timing	Check tx_valid high for 8 cycles after read_data command	Assertion-based (tx_valid_high property)	RAM_sva	
7	tx_valid Low	Ensure tx_valid stays low during write_addr/write_data/read_addr	Assertion-based (tx_valid_low property)	RAM_sva	
8	Write Sequence Order	Check that write_addr is followed by write_data	Assertion-based (write_addr_followed_by_write_data)	RAM_sva	
9	Read Sequence Order	Check that read_addr is followed by read_data	Assertion-based (read_addr_followed_by_read_data)	RAM_sva	
10	Functional Coverage	Measure coverage on command types and valid signals	Covergroups in RAM_coverage_pkg	RAM_coverage	
11	Scoreboard Comparison	Verify DUT output matches Golden Model output	UVM Scoreboard	RAM_scoreboard	
12	Interface Protocol	Ensure handshake between rx_valid, din, and tx_valid	UVM Monitor + Assertions	RAM_monitor / RAM_sva	
13	Coverage Closure	Ensure all scenarios (write/read/mixed/reset) exercised	Regression with all sequences	RAM_test	
14					

D	E	F
Implemented In	Pass Criteria	Expected Result
RAM_test / RAM_sva	dout==0, tx_valid==0 after reset	Passed if all assertions and scoreboard checks succeed
RAM_env / RAM_driver / RAM_monitor / RAM_scoreboard	MEM[Wr_Addr] updated correctly	Scoreboard matches DUT and Golden Model outputs
RAM_env / RAM_driver / RAM_monitor / RAM_scoreboard	dout matches MEM[Rd_Addr]	Scoreboard comparison passes
RAM_env / RAM_driver / RAM_monitor / RAM_scoreboard	Read after write returns correct data	Scoreboard shows 0 mismatches
RAM_sva	tx_valid[*8] ##1 tx_valid==0	Assertion holds without failure
RAM_sva	tx_valid==0 when din[9:8]!=2'b11	Assertion passes for all valid rx_valid inputs
RAM_sva	din transitions from 2'b00 to 2'b01	Assertion passes; coverage achieved
RAM_sva	din transitions from 2'b10 to 2'b11	Assertion passes; coverage achieved
RAM_coverage	All bins for din[9:8] and rx_valid, tx_valid covered	Coverage goal ≥ 95%
RAM_scoreboard	dout == dout_golden, tx_valid == tx_valid_golden	0 mismatches reported
RAM_monitor / RAM_sva	Stable signals and correct transitions	No assertion or protocol errors
RAM_test	All coverpoints and crosses hit	Coverage report ≥ 95%



- **How uvm_testbench work**

Testbench Operation Description

1. **Initialization**
    - **The top module instantiates both RAM DUT and Single_Port_RAM golden model, with separate interfaces.**
    - **Interfaces are passed to the UVM test via uvm_config_db.**
2. **Build Phase**
    - **RAM_test builds the environment, agents, and configuration objects.**
    - **The environment (RAM_env) creates:**
        - **RAM_agent → containing driver, monitor, sequencer.**
        - **RAM_scoreboard and RAM_coverage.**
3. **Connect Phase**
    - **The driver and monitor connect to virtual interfaces from the configuration.**

- Monitor's analysis port connects to both scoreboard and coverage.

4. Run Phase
    - The test runs sequences:

1. reset_seq to initialize DUT and model.
2. write_seq, read_seq, read_write_seq to generate random transactions.
    - The driver drives signals to both DUT and golden model.

5. Monitor and Scoreboard
    - The monitor captures DUT and golden outputs every cycle and sends data to scoreboard.
    - The scoreboard checks for mismatches and logs results.

6. Coverage and Assertions
    - Coverage tracks all valid scenarios.
    - SVA assertions check timing and operation order correctness.

7. Report Phase
    - The scoreboard prints final pass/fail results.
    - Coverage and assertion coverage are analyzed to ensure completeness.

Refrence Model

Single_Port_RAM.sv > ...

```
1   module Single_Port_RAM (din,rx_valid,clk,rst_n,dout,tx_valid);
2   parameter MEM_DEPTH = 256;
3   parameter ADDR_SIZE = 8 ;
4   input [9:0] din;
5   input rx_valid,clk,rst_n ;
6   output reg [7:0] dout ;
7   output reg tx_valid;
8   bit [7:0] mem [MEM_DEPTH-1:0]; // memory
9   reg [ADDR_SIZE-1:0] wr_Addr,rd_Addr; //Separated two addresses for write and read operations
10  always @(posedge clk) begin
11      if(~rst_n) begin
12          dout<=0;
13          wr_Addr<=0;
14          rd_Addr<=0;
15          tx_valid<=0;
16      end
17      else
18       if(rx_valid) begin
19      case(din[9:8])
20      2'b00:  begin
21          wr_Addr<=din[7:0];
22          tx_valid<=0;
23      end
24      2'b01:  begin
25          mem[wr_Addr]<=din[7:0];
26          tx_valid<=0;
27      end
28      2'b10:  begin
29          rd_Addr<=din[7:0];
30          tx_valid<=0;
31      end
32      2'b11: begin /// forced to  wait rx_data after 10 clk
33              dout<=mem[rd_Addr]; /// tx_data
34              tx_valid<=1;
35              end
36      endcase
37      end
38  end
39  endmodule
40
```

DUT

RAM.sv > ...

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

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

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

bit [7:0] MEM [255:0];

reg [7:0] Rd_Addr, Wr_Addr;

always @(posedge clk) begin
    if (~rst_n) begin
        dout <= 0;
        tx_valid <= 0;
        Rd_Addr <= 0;
        Wr_Addr <= 0;
    end
    else
        if (rx_valid) begin
            case (din[9:8])
                2'b00 : begin
                    Wr_Addr <= din[7:0];
                end
                2'b01 : begin
                    MEM[Wr_Addr] <= din[7:0];
                end
                2'b10 : begin Rd_Addr <= din[7:0];
                end
                2'b11 : begin
                    dout <= MEM[Rd_Addr];
                end
                default : dout <= 0;  // defualt case i will never reach it
            endcase
            tx_valid<=(din[9:8]==2'b11)?1:0;
        end

    end

//// i added always block to handle cycles counter
endmodule
```

Top Module

RAM_top.sv > ...

```
1
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import RAM_test_pkg::*;
5   module top();
6       bit clk;
7
8       initial begin
9           clk = 0;
10          forever #1 clk = ~clk;
11      end
12      Gold_RAM_if gold_RAMif(clk);
13      RAM_if RAMif (clk);
14      RAM dut (
15          .din(RAMif.din), .clk(clk) ,.rst_n(RAMif.rst_n), .rx_valid( RAMif.rx_valid), .dout(RAMif.dout) , .tx_valid(RAMif.tx_valid)
16
17      );
18
19      Single_Port_RAM golden_model(.din(gold_RAMif.din),.rx_valid(gold_RAMif.rx_valid),.clk(gold_RAMif.clk),
20        .rst_n(gold_RAMif.rst_n),.dout(gold_RAMif.dout),.tx_valid(gold_RAMif.tx_valid));
21      // --- Bind the Assertions ---
22      bind RAM RAM_sva RAM_assertions_inst (
23      .clk(RAMif.clk),
24      .rst_n(RAMif.rst_n),
25      .rx_valid(RAMif.rx_valid),
26      .din(RAMif.din),
27      .dout(RAMif.dout),
28      .tx_valid(RAMif.tx_valid)
29  );
30
31
32      initial begin
33          uvm_config_db#(virtual RAM_if)::set(null, "uvm_test_top", "RAM_IF", RAMif);
34          uvm_config_db#(virtual Gold_RAM_if)::set(null,"uvm_test_top","GOLD_RAM_IF",gold_RAMif);
35          run_test("RAM_test");
36      end
37
38  endmodule
39
```

RAM Test

RAM_test.sv > ...

```
package RAM_test_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import RAM_env_pkg::*;
import RAM_config_pkg::*;
import RAM_sequences_pkg::*;
class RAM_test extends uvm_test;
    `uvm_component_utils(RAM_test)

    RAM_env env;
    RAM_config RAM_cfg;
    RAM_config RAM_gold_cfg;
    virtual RAM_if RAM_vif;
    RAM_write_sequence write_seq;
    RAM_read_sequence read_seq;
    RAM_read_write_sequence read_write_seq;
    RAM_reset_sequence reset_seq;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        env = RAM_env::type_id::create("env", this);
        RAM_cfg = RAM_config::type_id::create("RAM_cfg", this);
        RAM_gold_cfg = RAM_config::type_id::create("RAM_gold_cfg", this);
        write_seq = RAM_write_sequence::type_id::create("write_seq", this);
        reset_seq = RAM_reset_sequence::type_id::create("reset_seq", this);
        read_seq=RAM_read_sequence::type_id::create("read_seq", this);
        read_write_seq=RAM_read_write_sequence::type_id::create("read_write_seq", this);

        if (!uvm_config_db #(virtual RAM_if)::get(this, "", "RAM_IF", RAM_cfg.RAM_vif))
            `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the RAM from the uvm_config_db");
            if (!uvm_config_db #(virtual Gold_RAM_if)::get(this, "", "GOLD_RAM_IF", RAM_gold_cfg.gold_RAM_vif))
           `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the RAM from the uvm_config_db");

        uvm_config_db #(RAM_config)::set(this, "*", "CFG", RAM_cfg);
        uvm_config_db #(RAM_config)::set(this, "*", "GOLD_CFG",RAM_gold_cfg );
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);
        //uvm_top.print_topology();
        //factory.print();

        //reset sequence
        `uvm_info("run_phase", "Reset Asserted", UVM_MEDIUM)
        reset_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "Reset Deasserted", UVM_MEDIUM)

        //main sequence
        `uvm_info("run_phase", "Stimulus Generation Started", UVM_MEDIUM)
        write_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "Stimulus Generation Ended", UVM_MEDIUM)
        read_seq.start(env.agt.sqr);
        read_write_seq.start(env.agt.sqr);
        phase.drop_objection(this);
    endtask
endclass
endpackage
```

- **Read Seq**

```
42  class RAM_read_sequence extends uvm_sequence #(RAM_seq_item);
43      `uvm_object_utils(RAM_read_sequence)
44      RAM_seq_item seq_item;
45
46      function new(string name="RAM_read_sequence");
47          super.new(name);
48      endfunction
49
50      task body();
51          seq_item = RAM_seq_item::type_id::create("seq_item");
52          seq_item.read_only.constraint_mode(1);
53          seq_item.read_write.constraint_mode(0);
54          seq_item.write_only.constraint_mode(0);
55          repeat(2000) begin
56          start_item(seq_item);
57          assert(seq_item.randomize());
58          finish_item(seq_item);
59          end
60      endtask
61  endclass
```

- **Write Seq**

```
22  class RAM_write_sequence extends uvm_sequence #(RAM_seq_item);
23      `uvm_object_utils(RAM_write_sequence)
24      RAM_seq_item seq_item;
25
26      function new(string name="RAM_write_sequence");
27          super.new(name);
28      endfunction
29
30      task body();
31          seq_item = RAM_seq_item::type_id::create("seq_item");
32          seq_item.read_only.constraint_mode(0);
33          seq_item.read_write.constraint_mode(0);
34          seq_item.write_only.constraint_mode(1);
35          repeat(2000) begin
36          start_item(seq_item);
37          assert(seq_item.randomize());
38          finish_item(seq_item);
39          end
40      endtask
41  endclass
```

Read Write Seq

```
62  class RAM_read_write_sequence extends uvm_sequence #(RAM_seq_item);
63      `uvm_object_utils(RAM_read_write_sequence)
64      RAM_seq_item seq_item;
65
66      function new(string name="RAM_read_write_sequence");
67          super.new(name);
68      endfunction
69
70      task body();
71          seq_item = RAM_seq_item::type_id::create("seq_item");
72          seq_item.read_only.constraint_mode(0);
73          seq_item.read_write.constraint_mode(1);
74          seq_item.write_only.constraint_mode(0);
75          repeat(2000) begin
76          start_item(seq_item);
77          assert(seq_item.randomize());
78          finish_item(seq_item);
79          end
80      endtask
81  endclass
82  endpackage
```

- **RAM environment**

RAM_env.sv > ...

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

- **RAM Scoreborad**

RAM_scoreboard.sv > SystemVerilog - Language Support > {} RAM_scoreboard_pkg > RAM_scoreboard > run_phase

```
package RAM_scoreboard_pkg;
import uvm_pkg::*;
import RAM_seq_item_pkg::*;
`include "uvm_macros.svh"
class RAM_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(RAM_scoreboard)
    uvm_analysis_export #(RAM_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(RAM_seq_item) sb_fifo;
    RAM_seq_item seq_item_sb;
    int error_count = 0;
    int correct_count = 0;

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

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

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            sb_fifo.get(seq_item_sb);
            if (seq_item_sb.dout != seq_item_sb.dout_golden || seq_item_sb.tx_valid !== seq_item_sb.tx_valid_golden) begin
                `uvm_error("run_phase", $sformatf("Comparison failed, Transaction received by the DUT:
                %s While the reference dout: %0d, tx_valid: %0d",
                        seq_item_sb.convert2string(), seq_item_sb.dout_golden,seq_item_sb.tx_valid_golden))
                error_count++;
            end
            else begin
                `uvm_info("run_phase", $sformatf("Correct RAM dout: %s ", seq_item_sb.convert2string()), UVM_HIGH);
                correct_count++;
            end
        end

    endtask
       function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("RAM", $sformatf("===== RAM Scoreboard Report =====\nCorrect Results: %0d\nErrors: %0d",
            correct_count, error_count), UVM_MEDIUM);
        endfunction
endclass
endpackage
```

RAM Coverage

RAM_coverage_pkg.sv > ...

```
package RAM_coverage_pkg;
 import uvm_pkg::*;
 import RAM_seq_item_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;

  parameter WRITE_ADDR = 2'b00;
  parameter WRITE_DATA = 2'b01;
  parameter READ_ADDR = 2'b10;
  parameter READ_DATA = 2'b11;

  covergroup cvr_grp;
   din_cov: coverpoint seq_item_cov.din[9:8]{
    bins Bin_Write_Addr = {WRITE_ADDR};
    bins Bin_Write_Data = {WRITE_DATA};
    bins Bin_Read_Addr = {READ_ADDR};
    bins Bin_Read_data = {READ_DATA};
    bins Bin_Write = (WRITE_ADDR => WRITE_DATA);
    bins Bin_Read = (READ_ADDR => READ_DATA);
    bins read_write_bin = (0 => 1) , (1 => 3) , ( 1 => 0),(0 => 2),
                          (2 => 3) , ( 2 => 0) , (3 => 1) , ( 3 => 2);
   }
      rx_valid_cov : coverpoint seq_item_cov.rx_valid {
      bins HIGH_rx = {1'b1};
      bins LOW_rx = {1'b0};
      }

     cross_rx_valid_din: cross din_cov , rx_valid_cov {
        option.cross_auto_bin_max=0;
        bins wr_din_rx_high =  binsof(rx_valid_cov.HIGH_rx) && binsof(din_cov. Bin_Write_Addr);
        bins wrd_din_rx_high =  binsof(rx_valid_cov.HIGH_rx) && binsof(din_cov.Bin_Write_Data);
        bins rd_din_rx_high =  binsof(rx_valid_cov.HIGH_rx) && binsof(din_cov.Bin_Read_Addr);
        bins rdd_din_rx_high =  binsof(rx_valid_cov.HIGH_rx) && binsof(din_cov.Bin_Read_data);
     }

    tx_valid_cov : coverpoint seq_item_cov.tx_valid {
      bins HIGH_tx = {1'b1};
      bins LOW_tx = {1'b0};
     }

    cross_tx_valid_din: cross din_cov , tx_valid_cov {
       option.cross_auto_bin_max=0;
        bins din_rt = binsof(din_cov.Bin_Read_data) && binsof(tx_valid_cov.HIGH_tx);
     }
  endgroup

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

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

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

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

endclass
endpackage
```

RAM Agent

RAM_agent.sv > ...

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

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

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

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    if (!uvm_config_db#(RAM_config)::get(this, "", "CFG", RAM_cfg)) begin
      `uvm_fatal("build_phase", "Unable to get configuration object")
    end
     if (!uvm_config_db#(RAM_config)::get(this, "", "GOLD_CFG", RAM_gold_cfg)) begin
   `uvm_fatal("build_phase", "Unable to get GOLDEN configuration object")
  end

    sqr = RAM_sequencer::type_id::create("sqr", this);
    drv = RAM_driver::type_id::create("drv", this);
    mon = RAM_monitor::type_id::create("mon", this);
    agt_ap = new("agt_ap", this);
  endfunction

  function void connect_phase(uvm_phase phase);
    drv.RAM_vif = RAM_cfg.RAM_vif;
    mon.RAM_vif = RAM_cfg.RAM_vif;
    drv.gold_RAM_vif=RAM_gold_cfg.gold_RAM_vif;
    mon.gold_RAM_vif=RAM_gold_cfg.gold_RAM_vif;
    drv.seq_item_port.connect(sqr.seq_item_export);
    mon.mon_ap.connect(agt_ap);
  endfunction

endclass
endpackage
```

RAM CFG

RAM_config.sv > ...

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

RAM Sequencer

RAM_sequencer.sv > ...

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

RAM Seq item

RAM_seq_item.sv > SystemVerilog - Language Support > {} RAM_seq_item_pkg > RAM_seq_item

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

  class RAM_seq_item extends uvm_sequence_item;
    `uvm_object_utils(RAM_seq_item)

    rand bit clk, rst_n, rx_valid;
    rand bit [9:0] din;
    bit [7:0] dout, dout_golden;
    bit tx_valid, tx_valid_golden;
    bit [9:0] prev_din = 2'b10;

    parameter write_addr = 2'b00;
    parameter write_data = 2'b01;
    parameter read_addr  = 2'b10;
    parameter read_data  = 2'b11;

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

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

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

    // ---- Constraints ----
    constraint rst_n_c {
      rst_n dist {1 := 99, 0 := 1};
    }

    constraint rx_valid_c {
      rx_valid dist {1 := 80, 0 :=20};
    }

    // Write-only operation
    constraint write_only {
      din[9:8] inside {write_addr , write_data};
      if(prev_din[9:8] == write_addr) {
        din[9:8]==write_data ;
      }
      else if (prev_din[9:8]==write_data) {
        din[9:8]==write_addr;
      }

    }

    // Read-only operation
    constraint read_only {
      din[9:8] inside {read_addr,read_data};
      if (prev_din[9:8] == read_addr){ din[9:8] == read_data};
      else if (prev_din[9:8] == read_data) { din[9:8] == read_addr};
    }

    // Mixed read/write sequencing
    constraint read_write {

      if (prev_din[9:8] == write_addr) {
        din[9:8] ==write_data;
      }
       else if (prev_din[9:8] == write_data) {
       din[9:8] dist {read_addr := 60, write_addr := 40};
       }

      else if(prev_din[9:8] == read_addr) {
        din[9:8]==read_data;
      }
      else if (prev_din[9:8] == read_data) {
        din[9:8] dist {write_addr := 60, read_addr := 40};
      }
    }

    function void post_randomize();
      prev_din = din;
    endfunction

  endclass
endpackage
```

RAM Driver

RAM_driver.sv > ...

```
1   package RAM_driver_pkg;
2    import uvm_pkg::*;
3    import RAM_seq_item_pkg::*;
4    `include "uvm_macros.svh"
5   class RAM_driver extends uvm_driver #(RAM_seq_item);
6     `uvm_component_utils(RAM_driver)
7
8     virtual RAM_if.DRV RAM_vif;
9     virtual Gold_RAM_if.DRV gold_RAM_vif;
10    RAM_seq_item stim_seq_item;
11
12    function new(string name = "RAM_driver", uvm_component parent = null);
13      super.new(name, parent);
14    endfunction
15
16    task run_phase(uvm_phase phase);
17      super.run_phase(phase);
18      forever begin
19        stim_seq_item = RAM_seq_item::type_id::create("stim_seq_item");
20        seq_item_port.get_next_item(stim_seq_item);
21        RAM_vif.din = stim_seq_item.din;
22        RAM_vif.rst_n    = stim_seq_item.rst_n;
23        RAM_vif.rx_valid  = stim_seq_item.rx_valid;
24        //gold
25        gold_RAM_vif.din=stim_seq_item.din;
26        gold_RAM_vif.rst_n=stim_seq_item.rst_n;
27        gold_RAM_vif.rx_valid=stim_seq_item.rx_valid;
28        @(negedge RAM_vif.clk);
29        seq_item_port.item_done();
30        `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH)
31      end
32    endtask
33  endclass
34  endpackage
35
```

RAM Monitor

RAM_monitor.sv > ...

```
package RAM_monitor_pkg;
import uvm_pkg::*;
import RAM_seq_item_pkg::*;
`include "uvm_macros.svh"
class RAM_monitor extends uvm_monitor;
    `uvm_component_utils(RAM_monitor)
    virtual RAM_if.MON RAM_vif;
    virtual Gold_RAM_if.MON gold_RAM_vif;
    RAM_seq_item rsp_seq_item;
    uvm_analysis_port #(RAM_seq_item) mon_ap;

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

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            rsp_seq_item = RAM_seq_item::type_id::create("rsp_seq_item");
            @(negedge RAM_vif.clk);
            rsp_seq_item.din = RAM_vif.din;
            rsp_seq_item.rst_n= RAM_vif.rst_n;
            rsp_seq_item.rx_valid = RAM_vif.rx_valid;
            rsp_seq_item.dout = RAM_vif.dout;
            rsp_seq_item.tx_valid = RAM_vif.tx_valid;
            //gold
            rsp_seq_item.tx_valid_golden=gold_RAM_vif.tx_valid;
            rsp_seq_item.dout_golden=gold_RAM_vif.dout;
            mon_ap.write(rsp_seq_item);
            `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH)
        end
    endtask
endclass
endpackage
```

RAM Interface

RAM_if.sv > ...

```
interface RAM_if (input  clk);
    logic rst_n;
    logic rx_valid;
    logic [9:0] din;
    logic [7:0] dout;
    logic tx_valid;

    modport DRV (
        input clk,
        output rst_n, rx_valid, din
    );

    // ------------ Monitor side ------------
    modport MON (
        input clk, rst_n, rx_valid, din, dout, tx_valid
    );

endinterface

```

Golden RAM interface

Gold_RAM_if.sv > ...

```
interface Gold_RAM_if (input  clk);
    logic rst_n;
    logic rx_valid;
    logic [9:0] din;
    logic [7:0] dout;
    logic tx_valid;

    modport DRV (
        input clk,
        output rst_n, rx_valid, din
    );

    // ------------ Monitor side ------------
    modport MON (
        input clk, rst_n, rx_valid, din, dout, tx_valid
    );

endinterface

```

Ram SVA module

RAM_sva.sv > SystemVerilog - Language Support > RAM_sva > tx_valid_high

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

 property reset_low_check ;
 (@(posedge clk) !rst_n |=> (tx_valid == 0 && dout == 0));
 endproperty
 property tx_valid_low;
 @(posedge clk) disable iff(!rst_n || !rx_valid)  (din[9:8] inside {2'b00, 2'b01, 2'b10}) && !tx_valid |=> (tx_valid == 0);
 endproperty
 property tx_valid_high;
@(posedge clk) disable iff(!rst_n || !rx_valid) (din[9:8] == 2'b11 && !$past(tx_valid)) |=> ($rose(tx_valid) ##1 $fell(tx_valid));
 endproperty

property write_addr_followed_by_write_data;
 @(posedge clk) disable iff(!rst_n || !rx_valid ) ( din[9:8] == 2'b00) |-> ##1  ( din[9:8] == 2'b01)
endproperty
 property read_addr_followed_by_read_data ;
 @(posedge clk)  disable iff(!rst_n || !rx_valid) ( din[9:8] == 2'b10) |-> ##1 ( din[9:8] == 2'b11);
 endproperty
 assert property (reset_low_check);
 assert property (tx_valid_low);
 assert property (tx_valid_high);
 assert property ( write_addr_followed_by_write_data);
 assert property (read_addr_followed_by_read_data);
 cover property (reset_low_check);
 cover property (tx_valid_low);
 cover property (tx_valid_high);
 cover property ( write_addr_followed_by_write_data);
 cover property (read_addr_followed_by_read_data);
 endmodule
```

Do file

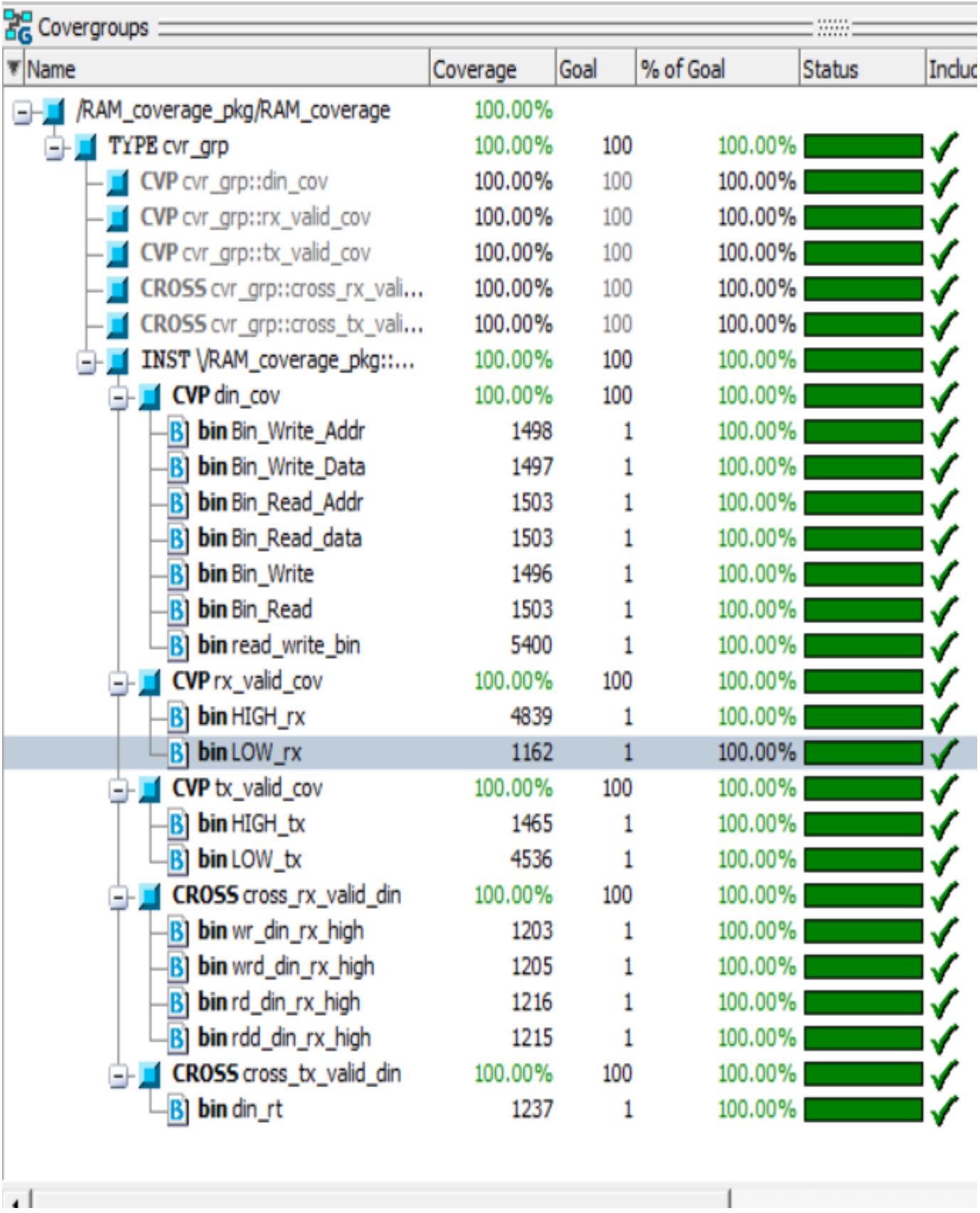
run.do

```
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
run 0
add wave -position insertpoint  \
sim:/uvm_root/uvm_test_top/env/sb/correct_count \
sim:/uvm_root/uvm_test_top/env/sb/error_count
add wave -position insertpoint  \
sim:/top/dut/din \
sim:/top/dut/rx_valid \
sim:/top/dut/dout \
sim:/top/dut/tx_valid \
sim:/top/dut/Rd_Addr \
sim:/top/dut/Wr_Addr
add wave -position insertpoint  \
sim:/top/golden_model/din \
sim:/top/golden_model/rx_valid \
sim:/top/golden_model/dout \
sim:/top/golden_model/tx_valid \
sim:/top/golden_model/wr_Addr \
sim:/top/golden_model/rd_Addr
run -all
coverage exclude -src RAM.sv -line 34 -code s
coverage exclude -src RAM.sv -line 34 -code b
coverage save RAM.ucdb -onexit -du work.RAM
coverage report -detail -cvg -directive -comments -output RAM_fcover_report.txt {}
/quit -sim
vcover report RAM.ucdb -details -annotate -all -output RAM_co.txt

```

src_files.list

```
RAM_if.sv
RAM.sv
Gold_RAM_if.sv
Single_Port_RAM.sv
RAM_sva.sv
RAM_config.sv
RAM_seq_item.sv
RAM_sequencer.sv
RAM_monitor.sv
RAM_driver.sv
RAM_agent.sv
RAM_scoreboard.sv
RAM_coverage_pkg.sv
RAM_env.sv
RAM_sequences.sv
RAM_test.sv
RAM_top.sv
```

Functional coverage Report



Covergroups

Name	Coverage	Goal	% of Goal	Status	Includ
/RAM_coverage_pkg/RAM_coverage	100.00%				
TYPE cvr_grp	100.00%	100	100.00%		
CVP cvr_grp::din_cov	100.00%	100	100.00%		
CVP cvr_grp::rx_valid_cov	100.00%	100	100.00%		
CVP cvr_grp::tx_valid_cov	100.00%	100	100.00%		
CROSS cvr_grp::cross_rx_vali...	100.00%	100	100.00%		
CROSS cvr_grp::cross_tx_vali...	100.00%	100	100.00%		
INST \/RAM_coverage_pkg::...	100.00%	100	100.00%		
CVP din_cov	100.00%	100	100.00%		
bin Bin_Write_Addr	1498	1	100.00%		
bin Bin_Write_Data	1497	1	100.00%		
bin Bin_Read_Addr	1503	1	100.00%		
bin Bin_Read_data	1503	1	100.00%		
bin Bin_Write	1496	1	100.00%		
bin Bin_Read	1503	1	100.00%		
bin read_write_bin	5400	1	100.00%		
CVP rx_valid_cov	100.00%	100	100.00%		
bin HIGH_rx	4839	1	100.00%		
bin LOW_rx	1162	1	100.00%		
CVP tx_valid_cov	100.00%	100	100.00%		
bin HIGH_tx	1465	1	100.00%		
bin LOW_tx	4536	1	100.00%		
CROSS cross_rx_valid_din	100.00%	100	100.00%		
bin wr_din_rx_high	1203	1	100.00%		
bin wrd_din_rx_high	1205	1	100.00%		
bin rd_din_rx_high	1216	1	100.00%		
bin rdd_din_rx_high	1215	1	100.00%		
CROSS cross_tx_valid_din	100.00%	100	100.00%		
bin din_rt	1237	1	100.00%		

Assertions Result

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Ac
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0	
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0	
/RAM_sequences_pkg::RAM_write_sequence::body/#ublk#5750839#35/immed__37	Immediate	SVA	on	0	1	
/RAM_sequences_pkg::RAM_read_sequence::body/#ublk#5750839#55/immed__57	Immediate	SVA	on	0	1	
/RAM_sequences_pkg::RAM_read_write_sequence::body/#ublk#5750839#75/immed__77	Immediate	SVA	on	0	1	
/top/dut/RAM_assertions_inst/assert__reset_low_check	Concurrent	SVA	on	0	1	
/top/dut/RAM_assertions_inst/assert__tx_valid_low	Concurrent	SVA	on	0	1	
/top/dut/RAM_assertions_inst/assert__tx_valid_high	Concurrent	SVA	on	0	1	
/top/dut/RAM_assertions_inst/assert__write_addr_followed_by_write_data	Concurrent	SVA	on	0	1	
/top/dut/RAM_assertions_inst/assert__read_addr_followed_by_read_data	Concurrent	SVA	on	0	1	

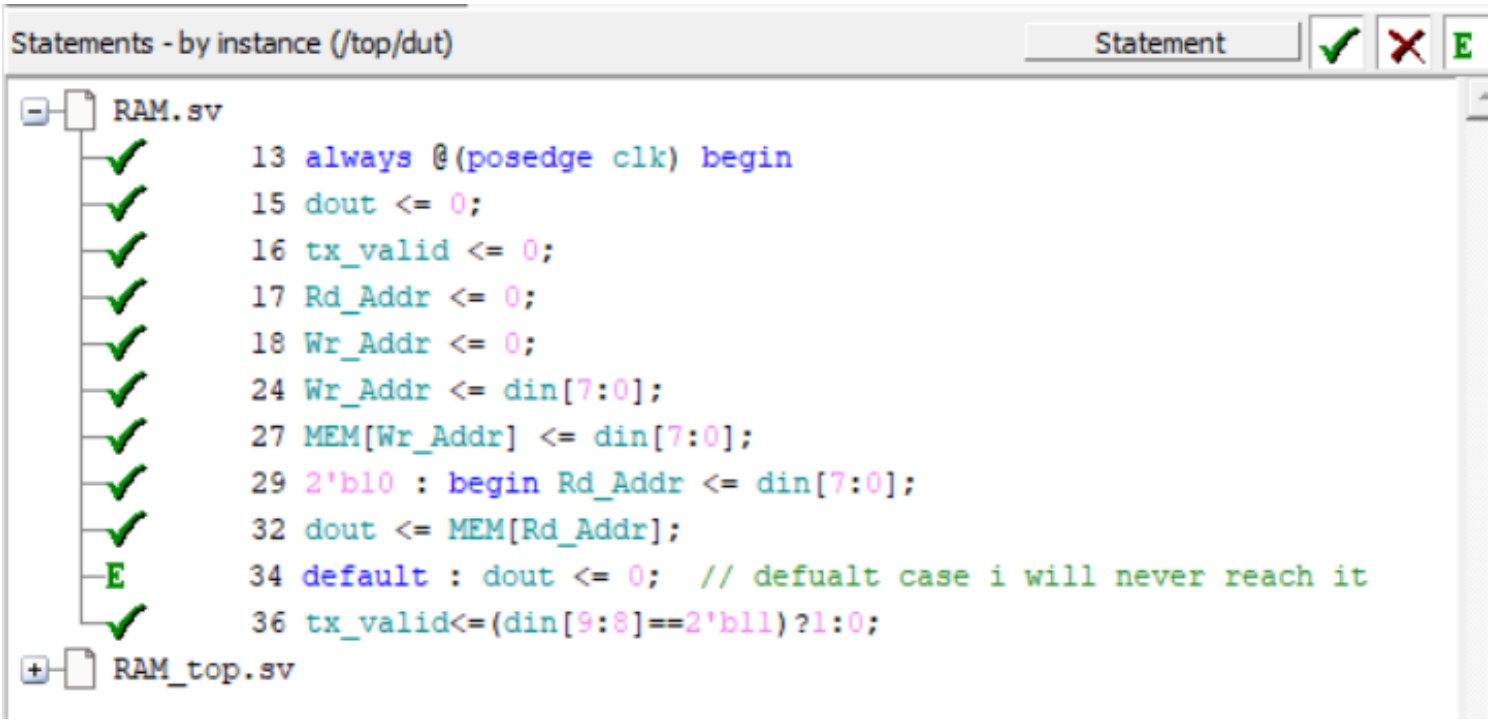
Assertions Coverage

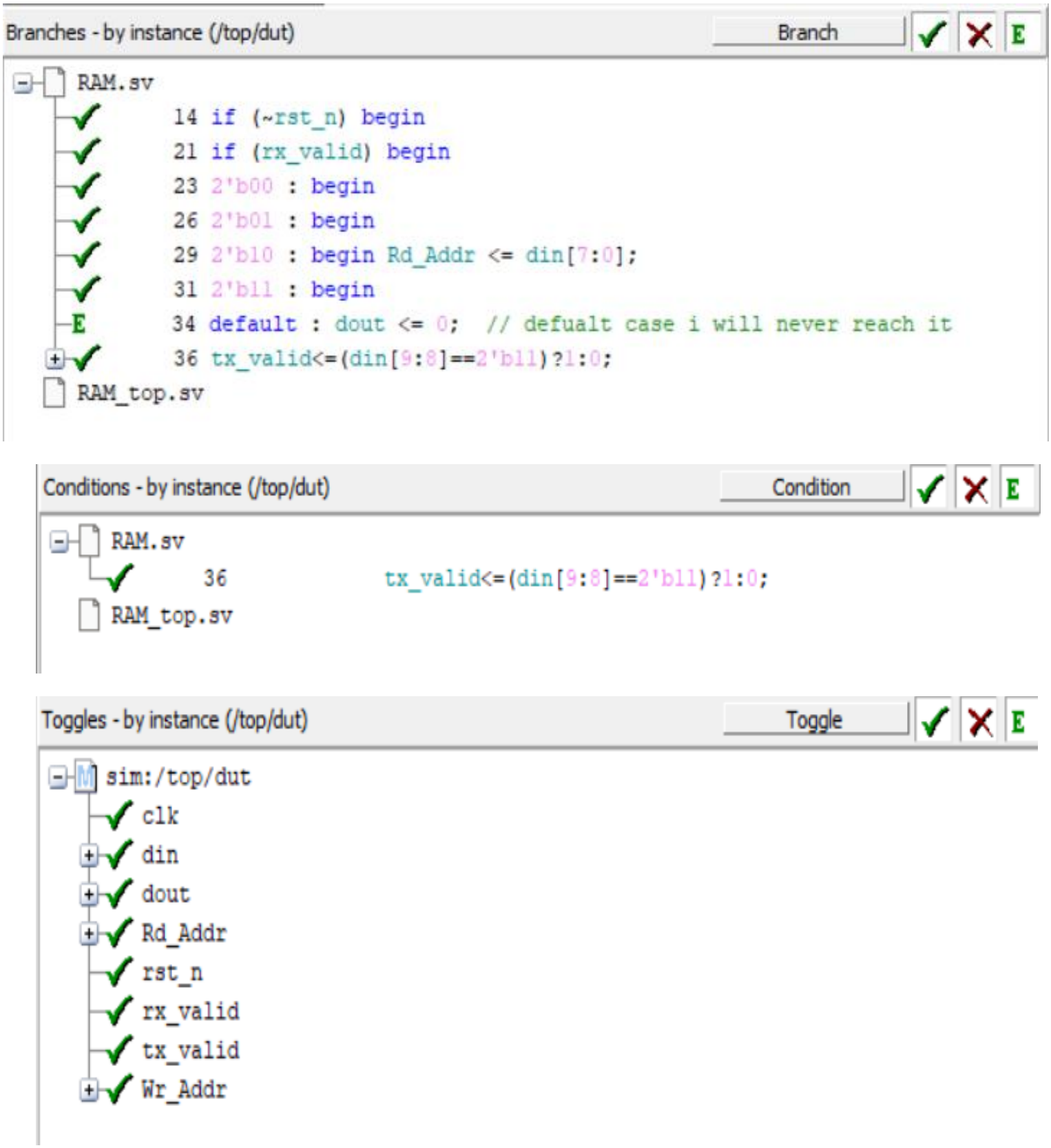
Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included
/top/dut/RAM_assertions_inst/cover__read_addr_followed_...	SVA	✓	Off	955	1	Unlimit...	1	100%		✓
/top/dut/RAM_assertions_inst/cover__write_addr_followed_...	SVA	✓	Off	943	1	Unlimit...	1	100%		✓
/top/dut/RAM_assertions_inst/cover__tx_valid_high	SVA	✓	Off	268	1	Unlimit...	1	100%		✓
/top/dut/RAM_assertions_inst/cover__tx_valid_low	SVA	✓	Off	2000	1	Unlimit...	1	100%		✓
/top/dut/RAM_assertions_inst/cover__reset_low_check	SVA	✓	Off	73	1	Unlimit...	1	100%		✓

Feature	Assertion
An assertion ensures that whenever reset is asserted, the output signals (tx_valid and dout) are low	(@(posedge clk) !rst_n |=> (tx_valid == 0 && dout == 0));
An assertion checks that during address or data input phases (write_add_seq, write_data_seq, read_add_seq), the tx_valid signal must remain deasserted.	@(posedge clk) disable iff(!rst_n || !rx_valid) (din[9:8] inside {2'b00, 2'b01, 2'b10}) && !tx_valid |=> (tx_valid == 0);
An assertion checks that after a read_data_seq occurs, the tx_valid signal must rise to indicate valid output and after it rises by one clock cycle, it should eventually fall	@(posedge clk) disable iff(!rst_n || !rx_valid) (din[9:8] == 2'b11 && !$past(tx_valid)) \|=> ($rose(tx_valid) ##1 $fell(tx_valid));
An assertion checks that every Write Address operation must be eventually followed by a Write Data operation.	@(posedge clk) disable iff(!rst_n || !rx_valid) (din[9:8] == 2'b00) |-> ##1 (din[9:8] == 2'b01)
An assertion checks that every Read Address operation must be eventually followed by a Read Data operation.	@(posedge clk) disable iff(!rst_n || !rx_valid) (din[9:8] == 2'b10) |-> ##1 (din[9:8] == 2'b11);

Code Coverage





All reports in project file for more details

Code Coverage Report Summary

```
248
249  DIRECTIVE COVERAGE:
250  ----------------------------------------------------------------------------------------
251  Name                                  Design Design   Lang File(Line)      Hits Status
252                                        Unit   UnitType
253  ----------------------------------------------------------------------------------------
254  /\top#dut /RAM_assertions_inst/cover__read_addr_followed_by_read_data
255                                        RAM_sva Verilog  SVA  RAM_sva.sv(33)   955 Covered
256  /\top#dut /RAM_assertions_inst/cover__write_addr_followed_by_write_data
257                                        RAM_sva Verilog  SVA  RAM_sva.sv(32)   943 Covered
258  /\top#dut /RAM_assertions_inst/cover__tx_valid_high
259                                        RAM_sva Verilog  SVA  RAM_sva.sv(31)   268 Covered
260  /\top#dut /RAM_assertions_inst/cover__tx_valid_low
261                                        RAM_sva Verilog  SVA  RAM_sva.sv(30)  2000 Covered
262  /\top#dut /RAM_assertions_inst/cover__reset_low_check
263                                        RAM_sva Verilog  SVA  RAM_sva.sv(29)    73 Covered
264
265  TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 5
266
267  ASSERTION RESULTS:
268  ---------------------------------------------------------------
269  Name                File(Line)                 Failure      Pass
270                                                 Count        Count
271  ---------------------------------------------------------------
272  /\top#dut /RAM_assertions_inst/assert__read_addr_followed_by_read_data
273                      RAM_sva.sv(28)                    0          1
274  /\top#dut /RAM_assertions_inst/assert__write_addr_followed_by_write_data
275                      RAM_sva.sv(27)                    0          1
276  /\top#dut /RAM_assertions_inst/assert__tx_valid_high
277                      RAM_sva.sv(26)                    0          1
278  /\top#dut /RAM_assertions_inst/assert__tx_valid_low
279                      RAM_sva.sv(25)                    0          1
280  /\top#dut /RAM_assertions_inst/assert__reset_low_check
281                      RAM_sva.sv(24)                    0          1
282
283  Total Coverage By Instance (filtered view): 100.00%
284
285
```

Bug report

C: > Users > LENOVO > Downloads > RAM.v > ...

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

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

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

reg [7:0] MEM [255:0];

reg [7:0] Rd_Addr, Wr_Addr;

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

endmodule
```

RAM.sv > ...

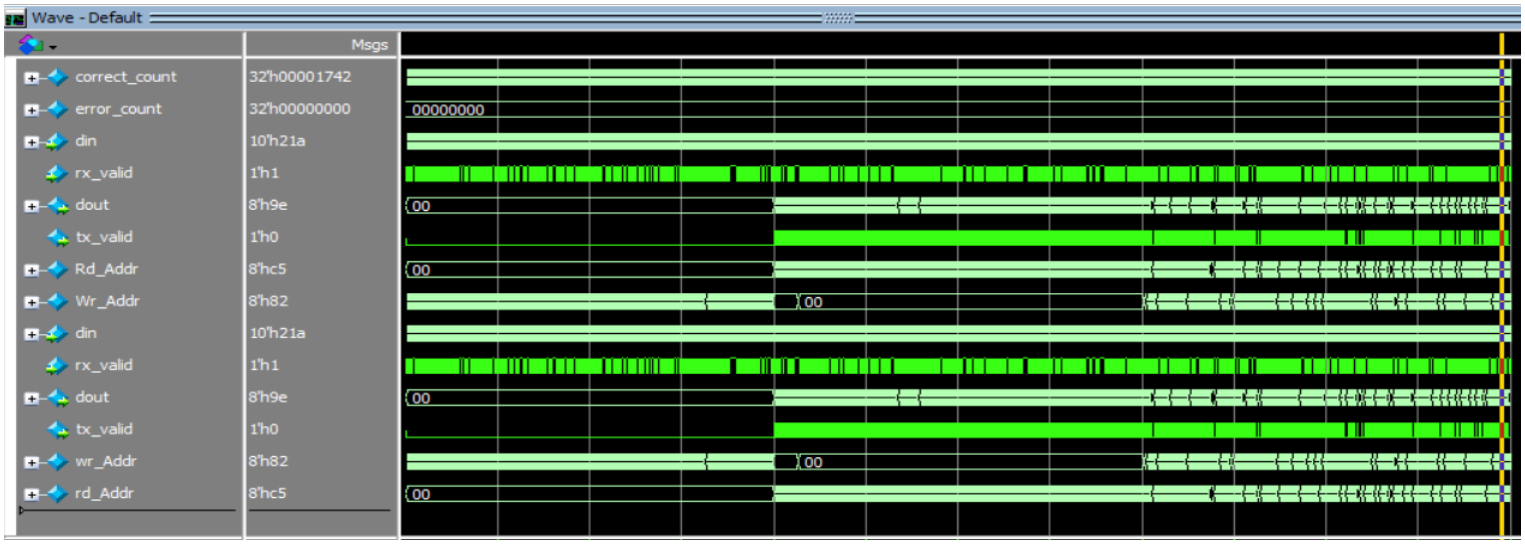
```sv
module RAM (din,clk,rst_n,rx_valid,dout,tx_valid);

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

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

bit [7:0] MEM [255:0];

reg [7:0] Rd_Addr, Wr_Addr;

always @(posedge clk) begin
    if (~rst_n) begin
        dout <= 0;
        tx_valid <= 0;
        Rd_Addr <= 0;
        Wr_Addr <= 0;
    end
    else
        if (rx_valid) begin
            case (din[9:8])
                2'b00 : begin
                    Wr_Addr <= din[7:0];
                end
                2'b01 : begin
                    MEM[Wr_Addr] <= din[7:0];
                end
                2'b10 : begin Rd_Addr <= din[7:0];
                end
                2'b11 : begin
                    dout <= MEM[Rd_Addr];
                end
                default : dout <= 0;  // defualt case i will never reach it
            endcase
            tx_valid<=(din[9:8]==2'b11)?1:0;
        end

    end

//// i added always block to handle cycles counter
endmodule
```

1- Change read data case output from dout<=mem[wr_addr] to dout<=mem[rd_addr]

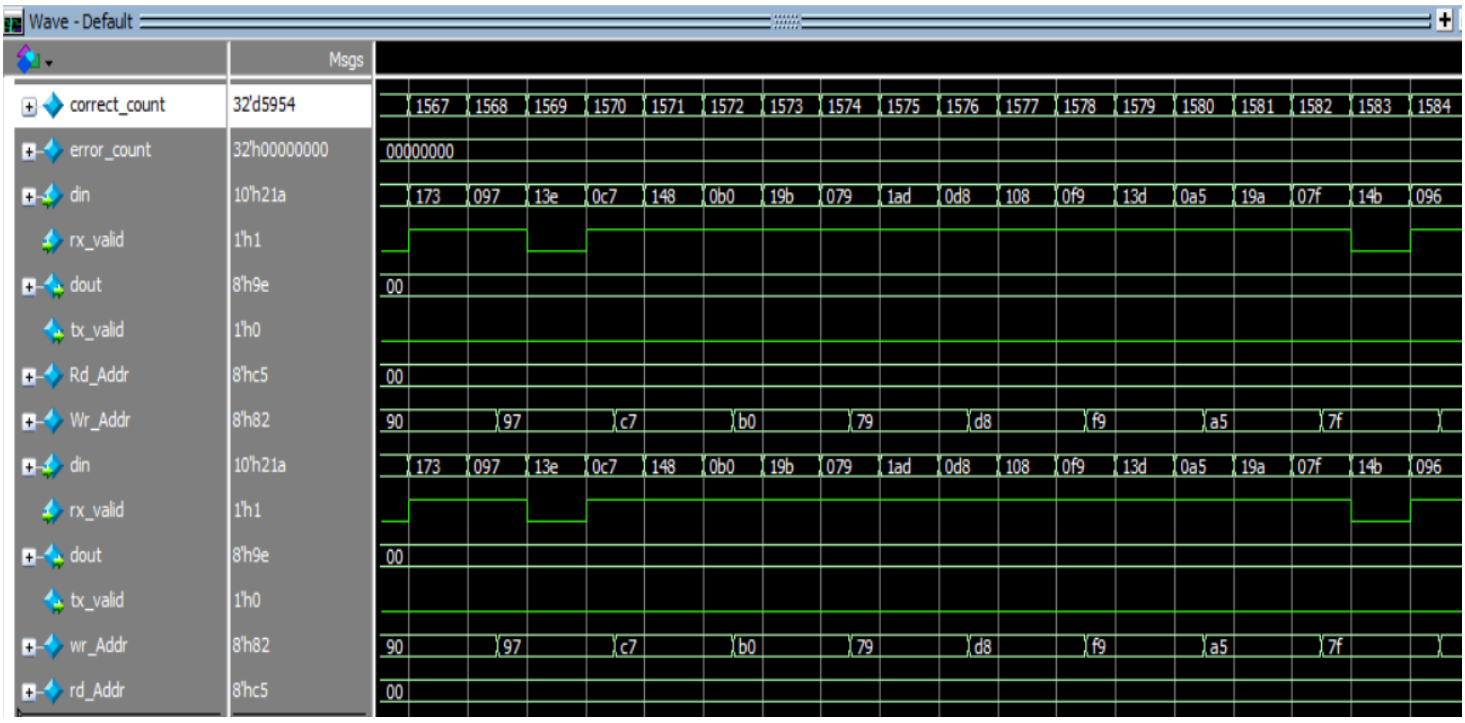
2- Its same old (ya3ny mlhash lazma)
I place tx_vlaid at the begin and end of rx_valid

Wave Forms and Output Summary

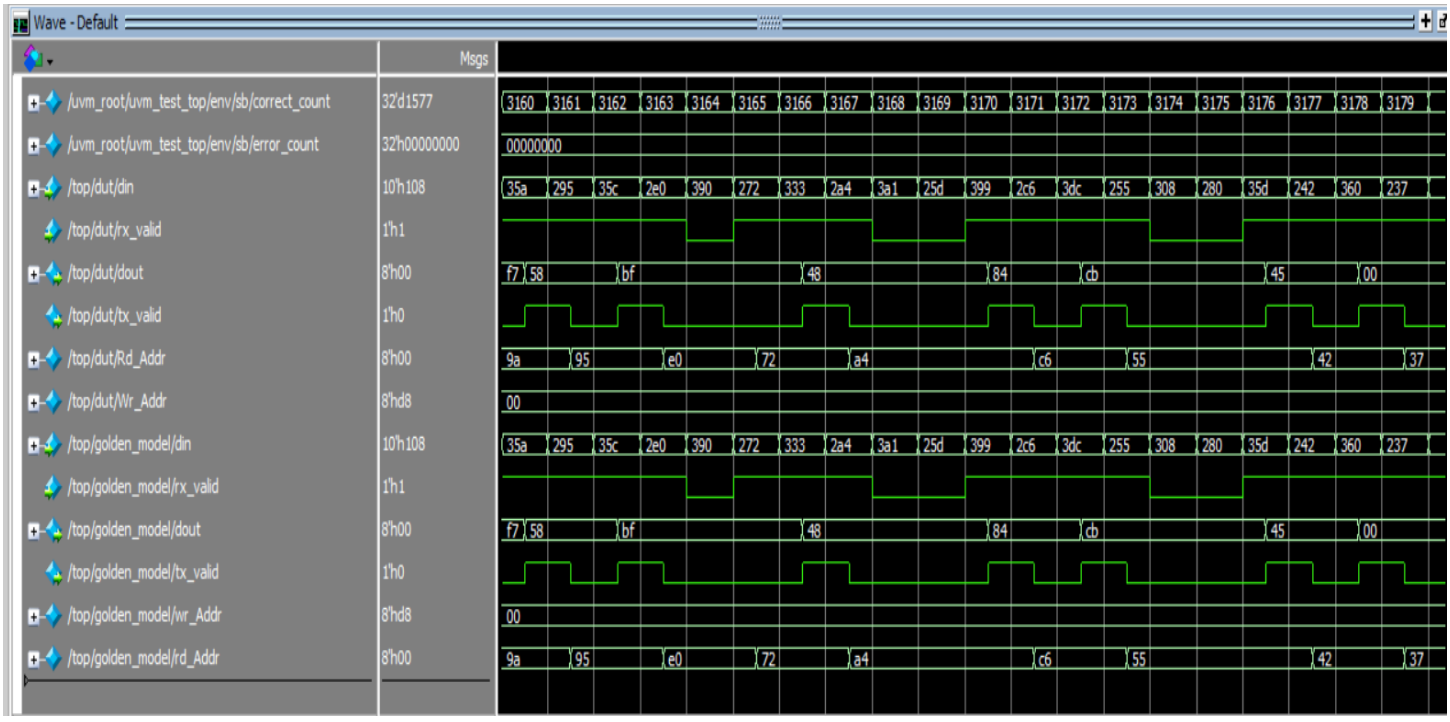
Complete Wave



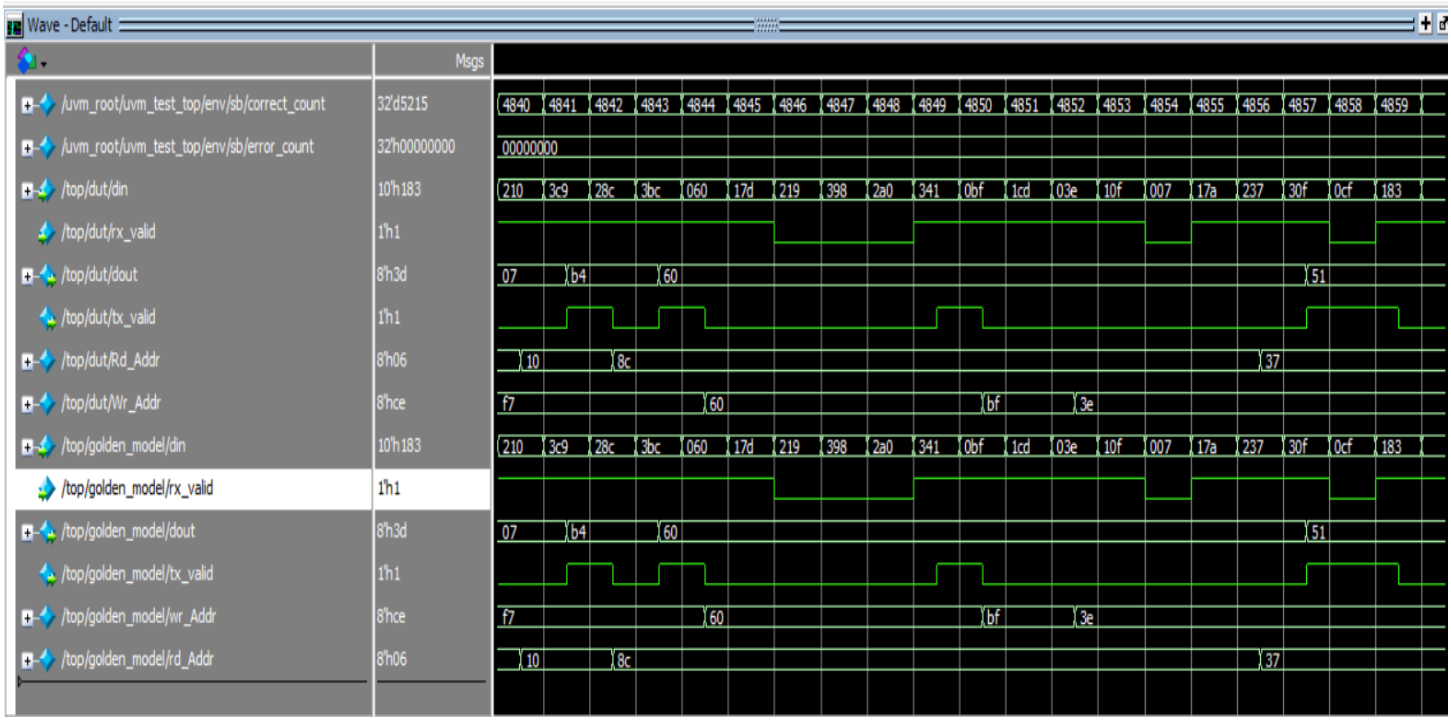
Write Seq



Read seq



Read Write seq



Output Summary

```
# UVM_INFO RAM_scoreboard.sv(47) @ 12002: uvm_test_top.env.sb [RAM] ===== RAM Scoreboard Report =====
# Correct Results: 6001
# Errors: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    9
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RAM]     1
# [RNTST]     1
# [TEST_DONE]     1
# [run_phase]     4
# ** Note: $finish    : C:/questasim64_2024.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 12002 ns  Iteration: 61  Instance: /top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2024.1/win64/../verilog_src/uvm-1.1d/src/base
```

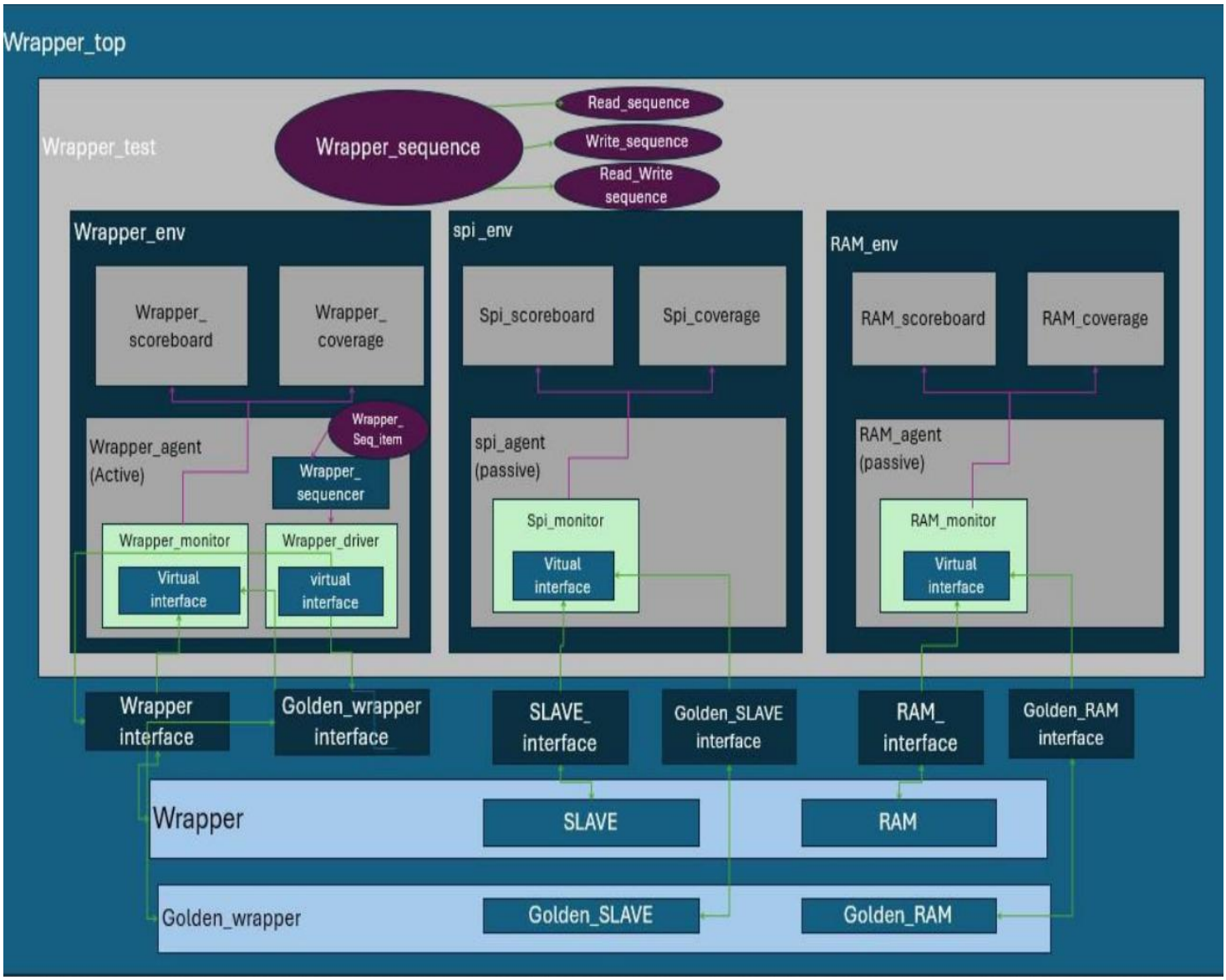
Part 3: UVM Environment for SPI Wrapper

Verification Plan

Label	Description	Stimulus Generation	Functional Coverage (Later)	Functionality Check
Reset Behavior	Verify wrapper, slave and RAM signals reset to known safe values and FSMs return to IDLE after reset.	Apply rst_n=0 at power-up and mid-transfer; then deassert.	Cover reset assertion/deassertion events and reset-during-transfer scenarios.	Check all outputs = 0, FSM returns to IDLE, rx_valid and tx_valid = 0. Scoreboard must report no transactions.
Default Idle / No Activity	Ensure wrapper remains idle when SS_n=1 (no active SPI frame).	Hold SS_n=1, apply clock, no MOSI activity	Cover long idle periods, transitior	FSM should stay in IDLE; no MISO toggling; no RAM read/write asserted
Command Decode (SPI→Wrapper)	Check correct decoding and routing of SPI commands to RAM.	Drive commands: 000, 001, 110, 111 (write_addr, write_data, read_addr, read_data	Cover all command types and opc	Wrapper must assert correct internal enables (rx_data_din, rx_valid, tx_valid); FSM transitions verified via monitor
RX Data Forwarding (SPI→RAM Write)	Confirm data received by SPI is forwarded to RAM on valid write	Apply SPI write sequences; send data to be stored.	Cover multiple data values and addresses.	Check RAM.din == rx_data_din and RAM.rx_valid asserted exactly when SPI rx_valid=1.
RAM Write Correctness	Ensure data written through wrapper matches expected RAM contents.	Perform multiple write operations to different addresses, then read back.	Cover edge addresses and repeated writes.	Scoreboard compares read-back data vs expected stored data
TX Path (RAM→SPI Read)	Validate wrapper correctly returns data from RAM via SPI.	Preload RAM with data, then issue read_data command (111).	Cover different tx_data values and tx_valid conditions.	MISO must match RAM.tx_data when tx_valid=1. Assertion checks timing alignment.
MISO Path Integrity	Ensure data output through MISO matches golden reference.	Apply identical stimuli to DUT and golden model.	Cover various MISO patterns and timing windows.	Scoreboard compares MISO vs MISO_ref bit-by-bit; mismatches increment error count.
RX_VALID Tin	Verify RX_VALID assertion delay per spec (10 cycles after frame).	Measure cycles between SS_n low and RX_VALID high	Cover bins for correct, early, late RX_VALID events.	Assertion checks RX_VALID asserted exactly after 10 cycles; any other timing fails.
SS_n	Ensure system aborts or completes transaction safely when SS_n toggles.	Deassert SS_n mid-frame or hold low for long transfer	Cover aborted and completed frame bins.	On abort, FSM resets to IDLE and no RAM write occurs; no RX_VALID must appear post-abort.
Bit Counter & Alignment	Check SPI bit counter resets and byte assembly works correctly.	Send various bit-length frames; monitor alignment	Cover counter from 0–9 and frame completion events.	Counter resets at frame end; data bits correctly grouped into bytes.

Continuous T	Verify system handles consecutive valid frames correctly.	Send sequences (000→001→110→111	Cover transitions between back-t	RX_VALID spacing correct; FSM transitions complete cleanly; no data overlap.
RX_VALID After Command	Check RX_VALID assertion after each valid command type.	Run all commands and monitor RX_VALID signal.	Cover RX_VALID after each of the 4 valid sequences	Assertion ensures RX_VALID asserted once per valid frame and deasserts before next frame
SS_n Glitch Recovery	Ensure system recovers from brief SS_n glitch.	Apply one-cycle SS_n pulse during transfer.	Cover glitch event and recovery.	FSM returns to IDLE, no invalid RAM write or read occurs.
RAM tx_valid Handling	Verify correct handling when RAM delays or misses tx_valid.	Simulate slow RAM or force delayed tx_	Cover delayed tx_valid and no-tx_valid scenarios.	Wrapper must wait until tx_valid=1 before sending MISO; no invalid output allowed.
Golden vs DU	Full comparison between DUT and golden wrapper paths.	Apply identical SPI inputs to both DUT a	Cover all command combinations	Scoreboard verifies identical MISO, rx_valid, tx_valid sequences; error counter = 0
FSM Coverag	Ensure all states and transitions are covered.	Execute mixed read/write/reset/abort	Cover FSM state transitions and c	Coverage goal: 100% FSM state and transition coverage
Monitor & TL	Ensure monitor samples all signals correctly and forwards to score	Drive known frames; observe analysis	Cover monitor sampling events a	Compare captured transaction fields vs actual signals to confirm consistency.

Wrapper structure



Detailed Description of UVM Testbench Operation

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

1. Overview

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

The UVM components are connected using **virtual interfaces**, enabling transaction-level communication between the UVM classes and the signal-level DUT interfaces.

---

2. Wrapper Environment (Active Agent)

The Wrapper_env is the main environment and acts as the active component in the testbench. It includes:

- **Wrapper_agent (Active)**: generates and drives stimulus to the DUT.
- **Wrapper_driver**: converts high-level transaction objects into signal-level activity through the Wrapper_interface.
- **Wrapper_monitor**: observes the DUT responses, reconstructs transactions, and forwards them to analysis components.
- **Wrapper_scoreboard**: compares DUT outputs with reference outputs from the Golden_wrapper.
- **Wrapper_coverage**: collects functional coverage to measure verification progress.

The agent operates in **active mode**, meaning it both drives stimulus (via the driver) and monitors DUT behavior. The **Wrapper_sequencer** acts as a mediator between the test sequences and the driver.

---

3. SPI and RAM Environments (Passive Agents)

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

- **spi_env** includes:
    - spi_agent (passive)
    - Spi_monitor
    - Spi_scoreboard
    - Spi_coverage

The Spi_monitor samples the SPI slave interface signals, converts them into SPI transaction objects, and forwards them to the scoreboard and coverage collectors.

- **RAM_env** includes:
    - RAM_agent (passive)
    - RAM_monitor
    - RAM_scoreboard
    - RAM_coverage

The RAM_monitor observes transactions on the memory interface and provides data to the scoreboard for result checking against the Golden_RAM.

Both environments validate the correctness of data exchanged between the Wrapper and the underlying modules while collecting coverage to ensure all protocol features are exercised.

---

4. Sequences and Transaction Flow

Test stimulus is generated by the **Wrapper_sequence**, which controls different operation types:

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

Each sequence creates Wrapper_Seq_item objects that represent read or write operations with parameters such as address, data, and operation type. These sequence items are sent to the **Wrapper_sequencer**, which forwards them to the **Wrapper_driver**.

The driver then drives the physical signals on the DUT interface according to the transaction details. During this process, the **Wrapper_monitor**, **Spi_monitor**, and **RAM_monitor** continuously observe activity and capture responses from the DUT.

---

5. Scoreboards and Golden Models

Each environment includes a **scoreboard** that performs transaction-level comparison between the DUT outputs and the expected outputs generated by **Golden models**.

- The **Golden models** (Golden_wrapper, Golden_SLAVE, and Golden_RAM) serve as behavioral reference models that mimic the correct functionality of each DUT component.
- When a transaction is driven to the DUT, the same transaction is also applied to the corresponding golden model.
- The scoreboard compares the DUT's observed results with the golden model's predicted results.
- Any mismatch is reported as a UVM_ERROR, ensuring functional correctness across all modules.

This comparison mechanism guarantees that the Wrapper and its connected modules perform accurately under all tested scenarios.

---

6. Coverage and Assertions

To ensure completeness of verification:

- **Functional coverage** is collected by coverage components within each environment, tracking transaction types, address ranges, data patterns, and operation combinations.
- **SystemVerilog Assertions (SVA)** are embedded to check protocol compliance, reset behavior, signal stability, and timing requirements.
  Examples include checking that outputs remain inactive during reset, or that valid handshakes occur correctly between tx_valid and rx_valid signals.

The combination of coverage and assertions provides both qualitative and quantitative confidence in the DUT's correctness.

---

7. Test Execution Flow

1. **Build Phase**: All UVM components (environments, agents, monitors, and scoreboards) are constructed and connected. Virtual interfaces are configured.
2. **Connect Phase**: Analysis ports are connected between monitors, scoreboards, and coverage components.
3. **Run Phase**:
   - The Wrapper_sequence is started on the Wrapper_sequencer.
   - Drivers translate sequence items into signal-level activity to stimulate the DUT.
   - Monitors capture responses and send observed transactions to scoreboards and coverage collectors.
   - Scoreboards compare DUT results against golden model predictions.
4. **Check and Report Phases**:
   - At the end of simulation, scoreboards summarize mismatches.
   - Coverage reports are generated to evaluate verification completeness.
   - Assertion results are reported for protocol validation.

---

8. Summary

The developed UVM testbench provides a **modular and scalable verification environment** for validating the Wrapper system.
By combining active and passive agents, golden reference models, functional coverage, and assertions, the testbench ensures complete verification of:

- Data correctness,
- Protocol adherence,
- Timing integrity,
- Functional coverage closure.

changed Files In the Two Passive Environment

RAM Cfg

RAM_config.sv > ...

```
package RAM_config_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
class RAM_config extends uvm_object;
  `uvm_object_utils(RAM_config)

  virtual RAM_if RAM_vif;
  virtual Gold_RAM_if gold_RAM_vif;
  uvm_active_passive_enum is_active;
  function new(string name = "RAM_config");
    super.new(name);
  endfunction
endclass
endpackage
```

SPI CFG

spi_config_obj.sv > SystemVerilog - Language Support > {} spi_config_obj > spi_config

```
package spi_config_obj;
import uvm_pkg::*;
`include "uvm_macros.svh"
class spi_config extends uvm_object;
`uvm_object_utils(spi_config)
virtual spi_if spi_config_vif;
virtual spi_gold_if spi_gold_config_vif;
uvm_active_passive_enum is_active;
function new(string name="spi_config");
super.new(name);
endfunction
endclass
endpackage
```

RAM agent

RAM_agent.sv > SystemVerilog - Language Support > {} RAM_agent_pkg > RAM_agent > build_phase

```
1   package RAM_agent_pkg;
2   import uvm_pkg::*;
3   import RAM_sequencer_pkg::*;
4   import RAM_driver_pkg::*;
5   import RAM_monitor_pkg::*;
6   import RAM_config_pkg::*;
7   import RAM_seq_item_pkg::*;
8   `include "uvm_macros.svh"
9   class RAM_agent extends uvm_agent;
10    `uvm_component_utils(RAM_agent)
11
12    RAM_sequencer sqr;
13    RAM_driver drv;
14    RAM_monitor mon;
15    RAM_config RAM_cfg;
16    RAM_config RAM_gold_cfg;
17    uvm_analysis_port #(RAM_seq_item) agt_ap;
18
19    function new(string name = "RAM_agent", uvm_component parent = null);
20      super.new(name, parent);
21    endfunction
22
23    function void build_phase(uvm_phase phase);
24      super.build_phase(phase);
25      if (!uvm_config_db#(RAM_config)::get(this, "", "R_CFG", RAM_cfg)) begin
26        `uvm_fatal("build_phase", "Unable to get configuration object")
27      end
28       if (!uvm_config_db#(RAM_config)::get(this, "", "R_GOLD_CFG", RAM_gold_cfg)) begin
29     `uvm_fatal("build_phase", "Unable to get GOLDEN configuration object")
30    end
31      if(RAM_cfg.is_active==UVM_ACTIVE) begin
32      sqr = RAM_sequencer::type_id::create("sqr", this);
33      drv = RAM_driver::type_id::create("drv", this);
34      end
35      mon = RAM_monitor::type_id::create("mon", this);
36      agt_ap = new("agt_ap", this);
37    endfunction
38
39    function void connect_phase(uvm_phase phase);
40      if(RAM_cfg.is_active==UVM_ACTIVE) begin
41      drv.RAM_vif = RAM_cfg.RAM_vif;
42      drv.seq_item_port.connect(sqr.seq_item_export);
43      drv.gold_RAM_vif=RAM_gold_cfg.gold_RAM_vif;
44      end
45      mon.gold_RAM_vif=RAM_gold_cfg.gold_RAM_vif;
46      mon.RAM_vif = RAM_cfg.RAM_vif;
47      mon.mon_ap.connect(agt_ap);
48    endfunction
49
50  endclass
51  endpackage
```

SPI agent

spi_agent.sv > SystemVerilog - Language Support > {} spi_agent_pkg > spi_agent > build_phase

```
package spi_agent_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import spi_seq_item_pkg::*;
import spi_config_obj::*;
import spi_driver::*;
import spi_sequencer_pkg::*;
import spi_monitor_pkg::*;
class spi_agent extends uvm_agent;
`uvm_component_utils(spi_agent)

spi_sequencer sqr;
spi_driver drv;
spi_monitor mon;
spi_config spi_cfg;
spi_config spi_gold_cfg;
uvm_analysis_port #(spi_seq_item) agt_ap;

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

function void build_phase(uvm_phase phase);
super.build_phase(phase);
if(!uvm_config_db #(spi_config)::get(this,"","S_CFG",spi_cfg))begin
`uvm_fatal("build_phase","unable to get CFG object")
end

if(!uvm_config_db #(spi_config)::get(this,"","S_CFG_GOLD",spi_gold_cfg))begin
`uvm_fatal("build_phase","unable to get CFG_GOLD object")
end
if (spi_cfg.is_active==UVM_ACTIVE) begin
sqr = spi_sequencer::type_id::create("sqr",this);
drv = spi_driver::type_id::create("drv",this);
end
mon = spi_monitor::type_id::create("mon",this);
agt_ap = new("agt_ap",this);
endfunction

function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
if(spi_cfg.is_active==UVM_ACTIVE) begin
drv.spi_vif = spi_cfg.spi_config_vif;
drv.spi_gold_vif = spi_gold_cfg.spi_gold_config_vif;
drv.seq_item_port.connect(sqr.seq_item_export);
end
mon.spi_vif = spi_cfg.spi_config_vif;
mon.spi_gold_vif = spi_gold_cfg.spi_gold_config_vif;
mon.mon_ap.connect(agt_ap);
endfunction
endclass
endpackage
```

Wrapper Files

Wrapper interface

wrapper_if.sv > SystemVerilog - Language Support > wrapper_if > SS_n

```
1   interface wrapper_if(clk);
2   input clk;
3   bit MOSI, SS_n, rst_n;
4   bit MISO;
5   endinterface
```

Golden Wrapper interface

Golden_wrapper_if.sv > SystemVerilog - Language Support > gold_wrapper_if

```
1   interface gold_wrapper_if(clk);
2   input clk;
3   bit MOSI, SS_n,  rst_n;
4   bit MISO;
5   endinterface
```

Wrapper DUT

SPI_wrapper.sv > SystemVerilog - Language Support > WRAPPER

```
1   module WRAPPER (MOSI,MISO,SS_n,clk,rst_n);
2
3   input  MOSI, SS_n, clk, rst_n;
4   output MISO;
5
6   wire [9:0] rx_data_din;
7   wire       rx_valid;
8   wire       tx_valid;
9   wire [7:0] tx_data_dout;
10
11  RAM   RAM_instance   (rx_data_din,clk,rst_n,rx_valid,tx_data_dout,tx_valid);
12  SLAVE SLAVE_instance (MOSI,MISO,SS_n,clk,rst_n,rx_data_din,rx_valid,tx_data_dout,tx_valid);
13
14
15  endmodule
```

Wrapper Golden Model

Golden_wrapper.sv > SystemVerilog - Language Support > GOLDEN_WRAPPER > [ ] tx_data

```
1   module GOLDEN_WRAPPER (MOSI,MISO,SS_n,clk,rst_n);
2   input  MOSI, SS_n, clk, rst_n;
3   output MISO;
4   wire [9:0] rx_data;
5   wire       rx_valid;
6   wire       tx_valid;
7   wire [7:0] tx_data;
8
9   Single_Port_RAM RRM(rx_data,rx_valid,clk,rst_n,tx_data,tx_valid);
10  SPI_Slave SSV(MOSI,SS_n,clk,rst_n,rx_data,tx_valid,tx_data,MISO,rx_valid);
11  endmodule
```

Wrapper shared

wrapper_shared_pkg.sv > SystemVerilog - Language Support > {} wrapper_shared > tx_valid

```
1   package wrapper_shared;
2     parameter  write_addr = 2'b00;
3     parameter  write_data = 2'b01;
4     parameter  read_addr  = 2'b10;
5     parameter  read_data  = 2'b11;
6     typedef enum  {IDLE,CHK_CMD ,WRITE,READ_ADD , READ_DATA } state_e;
7     state_e cs_s,ns_s;
8     bit received_address;
9     int  clk_count = 0;
10    bit rx_valid;
11    bit [9:0] rx_data;
12    bit tx_valid;
13  endpackage
```

Wrapper Top module

wrapper_top.sv > SystemVerilog - Language Support > wrapper_top

```
1   `include "uvm_macros.svh"
2   import
3   module
4   import
5   import wrapper_shared::*;
6   bit clk;
7   initial begin
8       clk=0;
9       forever #1 clk=~clk;
10  end
11
12  wrapper_if w_if(clk);
13  gold_wrapper_if gw_if(clk);
14  spi_gold_if gold_vif(clk);
15  spi_if spi_vif(clk);
16  Gold_RAM_if gold_RAMif(clk);
17  RAM_if RAMif (clk);
18  WRAPPER DUT(w_if.MOSI,w_if.MISO,w_if.SS_n,w_if.clk,w_if.rst_n);
19  GOLDEN_WRAPPER Golden(gw_if.MOSI,gw_if.MISO,gw_if.SS_n,gw_if.clk,gw_if.rst_n);
20  /*logic rst_n;
21  logic rx_valid;
22  logic [9:0] din;
23  logic [7:0] dout;
24  logic tx_valid;*/
25
26  assign RAMif.rst_n=w_if.rst_n;
27  assign RAMif.rx_valid=DUT.rx_valid;
28  assign RAMif.din=DUT.rx_data_din;
29  assign RAMif.dout=DUT.tx_data_dout;
30  assign RAMif.tx_valid=DUT.tx_valid;
31
32  assign gold_RAMif.rst_n=gw_if.rst_n;
33  assign gold_RAMif.rx_valid=Golden.rx_valid;
34  assign gold_RAMif.din=Golden.rx_data;
35  assign gold_RAMif.dout=Golden.tx_data;
36  assign gold_RAMif.tx_valid=Golden.tx_valid;
37
38  /*logic rst_n;
39  logic MISO;
40  logic MOSI;
41  logic SS_n;
42  logic tx_valid;
43  logic [7:0] tx_data;
44  logic [9:0] rx_data;
45  bit rx_valid;*/
46
47  assign spi_vif.rst_n=w_if.rst_n;
48  assign spi_vif.MISO=w_if.MISO;
49  assign spi_vif.MOSI=w_if.MOSI;
50  assign spi_vif.SS_n=w_if.SS_n;
51  assign spi_vif.tx_valid=DUT.tx_valid;
52  assign spi_vif.tx_data=DUT.tx_data_dout;
53  assign spi_vif.rx_data=DUT.rx_data_din;
54  assign spi_vif.rx_valid=DUT.rx_valid;
55
```

package wrapper_shared;

package wrapper_shared;

```
assign gold_vif.rst_n=gw_if.rst_n;
assign gold_vif.MISO=gw_if.MISO;
assign gold_vif.MOSI=gw_if.MOSI;
assign gold_vif.SS_n=gw_if.SS_n;
assign gold_vif.tx_valid=Golden.tx_valid;
assign gold_vif.tx_data=Golden.tx_data;
assign gold_vif.rx_data=Golden.rx_data;
assign gold_vif.rx_valid=Golden.rx_valid;

assign cs_s=state_e'(DUT.SLAVE_instance.cs);
assign ns_s=state_e'(DUT.SLAVE_instance.ns);
assign received_address=DUT.SLAVE_instance.received_address;
assign rx_valid=DUT.rx_valid;
assign rx_data=DUT.rx_data_din;
assign tx_valid=DUT.tx_valid;

bind WRAPPER wrapper_sva sva_inst(.MISO(w_if.MISO),.MOSI(w_if.MOSI),.rst_n(w_if.rst_n),.clk(w_if.clk),.SS_n(w_if.SS_n));

initial begin
    uvm_config_db#(virtual spi_if) :: set (null ,"uvm_test_top","spi_if",spi_vif);
    uvm_config_db#(virtual spi_gold_if) :: set (null , "uvm_test_top" , "spi_gold_if" , gold_vif);
    uvm_config_db#(virtual RAM_if)::set(null, "uvm_test_top", "RAM_IF", RAMif);
    uvm_config_db#(virtual Gold_RAM_if)::set(null,"uvm_test_top","GOLD_RAM_IF",gold_RAMif);
    uvm_config_db#(virtual wrapper_if)::set(null, "uvm_test_top", "WRAPPER_IF", w_if);
    uvm_config_db#(virtual gold_wrapper_if)::set(null,"uvm_test_top","GOLD_WRAPPER_IF",gw_if);
    run_test("wrapper_test");
end
endmodule
```

Wrapper Test

wrapper_test.sv > SystemVerilog - Language Support > {} wrapper_test_pkg > wrapper_test > build_phase

```
package wrapper_test_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import wrapper_env_pkg::*;
import wrapper_config_obj::*;
import wrapper_sequences_pkg::*;
import RAM_config_pkg::*;
import spi_config_obj::*;
import spi_env_pkg::*;
import RAM_env_pkg::*;
class wrapper_test extends uvm_test;
    `uvm_component_utils(wrapper_test)

    wrapper_env wr_env;
    spi_env sp_env;
    RAM_env R_env;
    wrapper_config wrapper_cfg;
    wrapper_config wrapper_gold_cfg;
    RAM_config RAM_cfg;
    RAM_config RAM_gold_cfg;
    spi_config spi_cfg;
    spi_config spi_gold_cfg;
    ////////////////////////////////
    wrapper_write_sequence write_seq;
    wrapper_read_sequence read_seq;
    wrapper_read_write_sequence read_write_seq;
    wrapper_reset_sequence reset_seq;

    function new(string name = "wrapper_test", uvm_component parent = null);
        super.new(name, parent);
    endfunction
    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        wr_env=wrapper_env::type_id::create("wr_env",this);
      sp_env =spi_env::type_id::create("sp_env",this);
      R_env = RAM_env::type_id::create("R_env", this);
     wrapper_cfg       = wrapper_config::type_id::create("wrapper_cfg");
     wrapper_gold_cfg  = wrapper_config::type_id::create("wrapper_gold_cfg");
     spi_cfg           = spi_config::type_id::create("spi_cfg");
     spi_gold_cfg      = spi_config::type_id::create("spi_gold_cfg");
     RAM_cfg           = RAM_config::type_id::create("RAM_cfg");
     RAM_gold_cfg      = RAM_config::type_id::create("RAM_gold_cfg");
     write_seq         = wrapper_write_sequence::type_id::create("write_seq",this);
     read_seq          = wrapper_read_sequence::type_id::create("read_seq",this);
     read_write_seq    = wrapper_read_write_sequence::type_id::create("read_write_seq",this);
     reset_seq         = wrapper_reset_sequence::type_id::create("reset_seq",this);
       if (!uvm_config_db #(virtual wrapper_if)::get(this, "", "WRAPPER_IF", wrapper_cfg.wrapper_config_vif))
           `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the wrap from the uvm_config_db");
          if (!uvm_config_db #(virtual gold_wrapper_if)::get(this, "", "GOLD_WRAPPER_IF",wrapper_gold_cfg.wrapper_gold_config_vif))
         `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the wrap from the uvm_config_db");
       if (!uvm_config_db#(virtual spi_if)::get(this,"","spi_if",spi_cfg.spi_config_vif))begin
          `uvm_fatal("build_phase","test - unable to get VIF")
        end
       if (!uvm_config_db#(virtual spi_gold_if)::get(this,"","spi_gold_if",spi_gold_cfg.spi_gold_config_vif))begin
           `uvm_fatal("build_phase","test - unable to get gold VIF")
       end
```

```
57              if (!uvm_config_db #(virtual RAM_if)::get(this, "", "RAM_IF", RAM_cfg.RAM_vif))
58           `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the RAM from the uvm_config_db");
59           if (!uvm_config_db #(virtual Gold_RAM_if)::get(this, "", "GOLD_RAM_IF", RAM_gold_cfg.gold_RAM_vif))
60          `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the RAM from the uvm_config_db");
61
62              uvm_config_db #(wrapper_config)::set(this, "*", "WR_CFG", wrapper_cfg);
63              uvm_config_db #(wrapper_config)::set(this, "*", "WR_GOLD_CFG",wrapper_gold_cfg );
64              uvm_config_db #(spi_config)::set(this,"*","S_CFG",spi_cfg); /// change from get to set
65              uvm_config_db #(spi_config)::set(this,"*","S_CFG_GOLD",spi_gold_cfg);
66              uvm_config_db #(RAM_config)::set(this, "*", "R_CFG", RAM_cfg);
67              uvm_config_db #(RAM_config)::set(this, "*", "R_GOLD_CFG",RAM_gold_cfg );
68              RAM_cfg.is_active=UVM_PASSIVE;
69              //RAM_gold_cfg.is_active=UVM_PASSIVE;
70              spi_cfg.is_active=UVM_PASSIVE;
71              //spi_gold_cfg.is_active=UVM_PASSIVE;
72              wrapper_cfg.is_active=UVM_ACTIVE;
73              //wrapper_gold_cfg.is_active=UVM_ACTIVE;
74      endfunction
75
76      task run_phase(uvm_phase phase);
77          super.run_phase(phase);
78          phase.raise_objection(this);
79          //reset sequence
80          `uvm_info("run_phase", "Reset Asserted", UVM_MEDIUM)
81          reset_seq.start(wr_env.agt.sqr);
82          `uvm_info("run_phase", "Reset Deasserted", UVM_MEDIUM)
83
84          //main sequence
85          `uvm_info("run_phase", "Stimulus Generation Started", UVM_MEDIUM)
86          write_seq.start(wr_env.agt.sqr);
87          `uvm_info("run_phase", "Stimulus Generation Ended", UVM_MEDIUM)
88          read_seq.start(wr_env.agt.sqr);
89          read_write_seq.start(wr_env.agt.sqr);
90          phase.drop_objection(this);
91      endtask
92  endclass
93  endpackage
```

Wrapper Sequences

wrapper_sequences.sv > SystemVerilog - Language Support > {} wrapper_sequences_pkg > wrapper_read_write_sequence > body

```
1   package wrapper_sequences_pkg;
2   import uvm_pkg::*;
3   import wrapper_seq_item_pkg::*;
4   `include "uvm_macros.svh"
5   class wrapper_reset_sequence extends uvm_sequence #(wrapper_seq_item);
6       `uvm_object_utils(wrapper_reset_sequence)
7       wrapper_seq_item seq_item;
8
9       function new(string name = "wrapper_reset_sequence");
10          super.new(name);
11      endfunction
12
13      task body;
14          seq_item = wrapper_seq_item::type_id::create("seq_item");
15          start_item(seq_item);
16          seq_item.SS_n = 0;
17          seq_item.MOSI = 0;
18          seq_item.rst_n= 0;
19          finish_item(seq_item);
20      endtask
21  endclass
22  class wrapper_write_sequence extends uvm_sequence #(wrapper_seq_item);
23      `uvm_object_utils(wrapper_write_sequence)
24      wrapper_seq_item seq_item;
25
26      function new(string name="wrapper_write_sequence");
27          super.new(name);
28      endfunction
29
30      task body();
31          seq_item = wrapper_seq_item::type_id::create("seq_item");
32          seq_item.read_only.constraint_mode(0);
33          seq_item.read_write.constraint_mode(0);
34          seq_item.write_only.constraint_mode(1);
35          repeat(10000) begin
36          start_item(seq_item);
37          assert(seq_item.randomize());
38          finish_item(seq_item);
39          end
40      endtask
41  endclass
```

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

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

    task body();
        seq_item = wrapper_seq_item::type_id::create("seq_item");
        seq_item.read_only.constraint_mode(1);
        seq_item.read_write.constraint_mode(0);
        seq_item.write_only.constraint_mode(0);
        repeat(10000) begin
        start_item(seq_item);
        assert(seq_item.randomize());
        finish_item(seq_item);
        end
    endtask
endclass
class wrapper_read_write_sequence extends uvm_sequence #(wrapper_seq_item);
```

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

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

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

Wrapper Environment

wrapper_env.sv > ...

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

Wrapper scoreboard

wrapper_scoreboard.sv > SystemVerilog - Language Support > {} wrapper_scoreboard_pkg

```
package wrapper_scoreboard_pkg;
import uvm_pkg::*;
import wrapper_seq_item_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;
int error_count=0;
int correct_count=0;

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

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

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

task run_phase(uvm_phase phase);
super.run_phase(phase);
forever begin
    sb_fifo.get(seq_item_sb);
    if(seq_item_sb.MISO != seq_item_sb.MISO_ref )begin
        `uvm_error("run_phase",$sformatf("failed  %s ",seq_item_sb.convert2string(),UVM_HIGH));
        error_count++;
    end
    else begin
        `uvm_info("run_phase",$sformatf("by dut  :%s",seq_item_sb.convert2string()),UVM_HIGH);
        correct_count++;
    end
end
endtask
function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("WRAPPER", $sformatf("===== WRAPPER Scoreboard Report =====\nCorrect Results: %0d\nErrors: %0d",
            correct_count, error_count), UVM_MEDIUM);
        endfunction

endclass
endpackage
```

Wrapper Coverage

wrapper_coverage.sv > SystemVerilog - Language Support > {} wrapper_coverage_pkg > wrapper_coverage

```
package wrapper_coverage_pkg;
import uvm_pkg::*;
import wrapper_seq_item_pkg::*;
import wrapper_shared::*;
`include "uvm_macros.svh"
class wrapper_coverage extends uvm_component;

`uvm_component_utils(wrapper_coverage)
uvm_analysis_export #(wrapper_seq_item) cov_export;
uvm_tlm_analysis_fifo #(wrapper_seq_item) cov_fifo;
wrapper_seq_item cov_txn;

covergroup CovCode;
c1:coverpoint rx_data[9:8]{   // come from shared
bins all_values [] = {2'b00,2'b01,2'b10,2'b11};
bins rx_data_trans[] = ( 0 => 1) , ( 1=> 3) , (1 => 0) , ( 2 => 3) , ( 2=> 0) , (3 => 1), ( 3 => 2);
}

SS_n_c3:coverpoint cov_txn.SS_n {
    bins ssn_13_wr = (1 => 0[*13] => 1) ;
    bins ssn_23_wr = (1 => 0[*23] => 1) ;
}
MOSI_c4:coverpoint cov_txn.MOSI {
    bins write_address = (0 => 0 => 0) ;
    bins write_data = (0 => 0 => 1) ;
    bins read_address = (1 => 1 => 0) ;
    bins read_data = (1 => 1 => 1) ;
}
//cross coverage between SS_n and MOSI(only legal scenarios)
 SS_n_MOSI_cross:cross SS_n_c3 , MOSI_c4 {
    option.cross_auto_bin_max=0;
    bins legal_write_address = binsof(SS_n_c3.ssn_13_wr) && binsof(MOSI_c4.write_address);
    bins legal_write_data = binsof(SS_n_c3.ssn_13_wr) && binsof(MOSI_c4.write_data) ;
    bins legal_read_address = binsof(SS_n_c3.ssn_13_wr) && binsof(MOSI_c4.read_address) ;
    bins legal_read_data = binsof(SS_n_c3.ssn_23_wr) && binsof(MOSI_c4.read_data) ;
    ignore_bins wrong_normal_read_data =binsof(SS_n_c3.ssn_13_wr) && binsof(MOSI_c4.read_data) ;
    ignore_bins wrong_extended_write = binsof(SS_n_c3.ssn_23_wr) && binsof(MOSI_c4.write_address) ;
    ignore_bins wrong_extended_write_data = binsof(SS_n_c3.ssn_23_wr) && binsof(MOSI_c4.write_data);
    ignore_bins wrong_extended_read_add=binsof(SS_n_c3.ssn_23_wr) && binsof(MOSI_c4.read_address);
      }
       din_cov: coverpoint rx_data[9:8]{
       bins Bin_Write_Addr = {write_addr};
       bins Bin_Write_Data = {write_data};
       bins Bin_Read_Addr = {read_addr};
       bins Bin_Read_data = {read_data};
       bins Bin_Write = (write_addr=> write_data);
       bins Bin_Read = (read_addr => read_data);
       bins read_write_bin = (0 => 1) , (1 => 3) , ( 1 => 0),(0 => 2),
                             (2 => 3) , ( 2 => 0) , (3 => 1) , ( 3 => 2);
 }
    rx_valid_cov : coverpoint rx_valid {
    bins HIGH_rx = {1'b1};
    bins LOW_rx = {1'b0};
    }
```

```
    cross_rx_valid_din: cross din_cov , rx_valid_cov {
      option.cross_auto_bin_max=0;
      bins wr_din_rx_high =  binsof(rx_valid_cov.HIGH_rx) && binsof(din_cov. Bin_Write_Addr);
      bins wrd_din_rx_high =  binsof(rx_valid_cov.HIGH_rx) && binsof(din_cov.Bin_Write_Data);
      bins rd_din_rx_high =  binsof(rx_valid_cov.HIGH_rx) && binsof(din_cov.Bin_Read_Addr);
      bins rdd_din_rx_high =  binsof(rx_valid_cov.HIGH_rx) && binsof(din_cov.Bin_Read_data);
    }

  tx_valid_cov : coverpoint tx_valid {
     bins HIGH_tx = {1'b1};
     bins LOW_tx = {1'b0};
    }

  cross_tx_valid_din: cross din_cov , tx_valid_cov {
      option.cross_auto_bin_max=0;
      bins din_rt = binsof(din_cov.Bin_Read_data) && binsof(tx_valid_cov.HIGH_tx);
    }
  endgroup


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

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

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

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

Wrapper agent

wrapper_agent.sv > SystemVerilog - Language Support > {} wrapper_agent_pkg

```
1   package wrapper_agent_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import wrapper_seq_item_pkg::*;
5   import wrapper_config_obj::*;
6   import wrapper_driver_pkg::*;
7   import wrapper_sequencer_pkg::*;
8   import wrapper_monitor_pkg::*;
9   class wrapper_agent extends uvm_agent;
10  `uvm_component_utils(wrapper_agent)
11
12  wrapper_sequencer sqr;
13  wrapper_driver drv;
14  wrapper_monitor mon;
15  wrapper_config wrapper_cfg;
16  wrapper_config wrapper_gold_cfg;
17  uvm_analysis_port #(wrapper_seq_item) agt_ap;
18
19  function new(string name = "wrapper_agent",uvm_component parent = null);
20  super.new(name,parent);
21  endfunction
22
23  function void build_phase(uvm_phase phase);
24  super.build_phase(phase);
25  if(!uvm_config_db #(wrapper_config)::get(this,"","WR_CFG",wrapper_cfg))begin
26  `uvm_fatal("build_phase","unable to get CFG object")
27  end
28
29  if(!uvm_config_db #(wrapper_config)::get(this,"","WR_GOLD_CFG",wrapper_gold_cfg))begin
30  `uvm_fatal("build_phase","unable to get CFG_GOLD object")
31  end
32  if (wrapper_cfg.is_active==UVM_ACTIVE) begin
33  sqr = wrapper_sequencer::type_id::create("sqr",this);
34  drv = wrapper_driver::type_id::create("drv",this);
35  end
36  mon = wrapper_monitor::type_id::create("mon",this);
37  agt_ap = new("agt_ap",this);
38  endfunction
39
40  function void connect_phase(uvm_phase phase);
41  super.connect_phase(phase);
42  if(wrapper_cfg.is_active==UVM_ACTIVE) begin
43  drv.w_if = wrapper_cfg.wrapper_config_vif;
44  drv.gw_if = wrapper_gold_cfg.wrapper_gold_config_vif;
45  drv.seq_item_port.connect(sqr.seq_item_export);
46  end
47  mon.w_if = wrapper_cfg.wrapper_config_vif;
48  mon.gw_if = wrapper_gold_cfg.wrapper_gold_config_vif;
49  mon.mon_ap.connect(agt_ap);
50  endfunction
51  endclass
52  endpackage
53
```

Wrapper CFG

wrapper_config_obj.sv > ...

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

Wrapper Sequencer

wrapper_sequencer.sv > SystemVerilog - Language Support > {} wrapper_sequencer_pkg > wrapper_sequencer

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

Wrapper Seq item

wrapper_seq_item.sv > ...

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

class wrapper_seq_item extends uvm_sequence_item;
    `uvm_object_utils(wrapper_seq_item);

    rand bit rst_n;
    rand bit MOSI;
    rand bit SS_n;
    bit MISO, MISO_ref;
    rand bit [10:0] mosi_bus;
    bit [1:0] prev_din ;
    int prev_count;
    bit [2:0]  prev_mosi_3;

    function new(string name = "wrapper_seq_item");
        super.new(name);
    endfunction
    //////////////////
    constraint rst_n_c {
    rst_n dist {1 := 99, 0 := 1};
    }
    ////////////////
    // Write-only operation
    constraint write_only {
        if(prev_count==1){
        mosi_bus[10]==0;
      (prev_din == write_addr) ->
        mosi_bus[9:8] dist {write_addr := 10, write_data := 90};
        (prev_din == write_data) ->
       mosi_bus[9:8] dist {write_addr := 90, write_data := 10};
        }
    }
    // Read-only operation
    constraint read_only {
        if(prev_count==1) {
            mosi_bus[9:8] inside {read_addr,read_data};
       mosi_bus[10]==1;
      ( received_address) -> mosi_bus[9:8] == read_data;
      (!received_address) -> mosi_bus[9:8] == read_addr;
        }
    }
    // Mixed read/write sequencing
    constraint read_write {
        if(prev_count==1){
            mosi_bus[10:8] inside {3'b000,3'b001,3'b110,3'b111};

      if (prev_mosi_3==3'b000) {
        mosi_bus[9:8] dist {write_addr := 10, write_data := 90};
        mosi_bus[10]==0;
      }
```

```
      else if (prev_mosi_3==3'b001){
       mosi_bus[10:8] dist {{1'b1,read_addr} := 60, {1'b0,write_addr} := 40};
      }
     else if (prev_mosi_3==3'b110) {
       if(received_address) {
       mosi_bus[9:8] ==read_data;
       mosi_bus[10]==1;
       }
       else {
          mosi_bus[9:8] ==read_addr;
          mosi_bus[10]==1;
       }
     }
     else if (prev_mosi_3==3'b111 && !received_address){
       mosi_bus[10:8] dist {{1'b0,write_addr} := 60, {1'b1,read_addr} := 40};}
   }}
   function void post_randomize();

   if(SS_n || !rst_n) begin
       clk_count = 0;
   end
   else clk_count++;
      prev_count=clk_count;
      prev_din = mosi_bus[9:8];
      prev_mosi_3=mosi_bus[10:8];
   endfunction

   //////////////////////////
   constraint SS_n_c{
   if(cs_s==READ_DATA){
       if(clk_count == 23) SS_n == 1;
       else SS_n == 0;
   }
    else if (cs_s!=READ_DATA)  {
       if(clk_count == 13) SS_n == 1;
       else SS_n == 0;
   }
  }
  ////////////////////////////
   constraint MOSI_c{
      if (clk_count > 0 && clk_count < 12 ) MOSI == mosi_bus[11 - clk_count];
   }

   function void pre_randomize();
       if(cs_s==CHK_CMD) begin
           mosi_bus.rand_mode(1);
       end
       else begin
           mosi_bus.rand_mode(0);
       end
   endfunction



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

   function string convert2string_stimulus();
       return $sformatf("rst_n=%0b SS_n=%0b MOSI=%0b",
           rst_n, SS_n, MOSI);
   endfunction

endclass
endpackage
```

Wrapper Driver

wrapper_driver.sv > SystemVerilog - Language Support > {} wrapper_driver_pkg

```
package wrapper_driver_pkg;
`include "uvm_macros.svh"
import uvm_pkg::*;
import wrapper_seq_item_pkg::*;
class wrapper_driver extends uvm_driver #(wrapper_seq_item);
`uvm_component_utils(wrapper_driver)
wrapper_seq_item stim_seq_item;
virtual wrapper_if w_if;
virtual gold_wrapper_if gw_if;


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

task run_phase (uvm_phase phase);
 super.run_phase(phase);
 forever begin
 stim_seq_item=wrapper_seq_item::type_id::create("stim_seq_item");
 seq_item_port.get_next_item(stim_seq_item);
 w_if.MOSI = stim_seq_item.MOSI;
 w_if.SS_n = stim_seq_item.SS_n;
 w_if.rst_n = stim_seq_item.rst_n;
 //for gold
 gw_if.MOSI = stim_seq_item.MOSI;
 gw_if.SS_n = stim_seq_item.SS_n;
 gw_if.rst_n = stim_seq_item.rst_n;
 @(negedge w_if.clk);
 seq_item_port.item_done();
 `uvm_info("run_phase",stim_seq_item.convert2string_stimulus(),UVM_HIGH);
 end
 endtask
 endclass
 endpackage
```

Wrapper Monitor

wrapper_monitor.sv > ...

```
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)

virtual wrapper_if w_if;
virtual gold_wrapper_if gw_if;
uvm_analysis_port#(wrapper_seq_item) mon_ap;
wrapper_seq_item rsp_seq_item;

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

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

task run_phase(uvm_phase phase);
super.run_phase(phase);
forever begin
 rsp_seq_item=wrapper_seq_item::type_id::create("rsp_seq_item");
 @(negedge w_if.clk);
 rsp_seq_item.rst_n = w_if.rst_n;
 rsp_seq_item.MOSI = w_if.MOSI;
 rsp_seq_item.MISO = w_if.MISO;
 rsp_seq_item.SS_n = w_if.SS_n;
 // gold
 rsp_seq_item.MISO_ref=gw_if.MISO;
 mon_ap.write(rsp_seq_item);
 `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH);
end
endtask
endclass
endpackage

```

Wrapper SVA

wrapper_sva.sv > SystemVerilog - Language Support > wrapper_sva > stable_miso

```
1   import wrapper_shared::*;
2   module wrapper_sva(MISO,MOSI,rst_n,SS_n,clk);
3   input MISO,MOSI,rst_n,SS_n,clk;
4
5
6   property p_reset_outputs_inactive;
7     @(posedge clk)
8     (!rst_n) |=> (MISO == 0 && rx_valid == 0 && rx_data == 0);
9   endproperty
10
11  assert property (p_reset_outputs_inactive);
12  cover property (p_reset_outputs_inactive);
13
14  property stable_miso;
15  @(posedge clk) disable iff (!rst_n) (cs_s!=READ_DATA) |=> $stable(MISO);
16  endproperty
17
18  assert property (stable_miso);
19  cover property (stable_miso);
20  endmodule
```

Do file

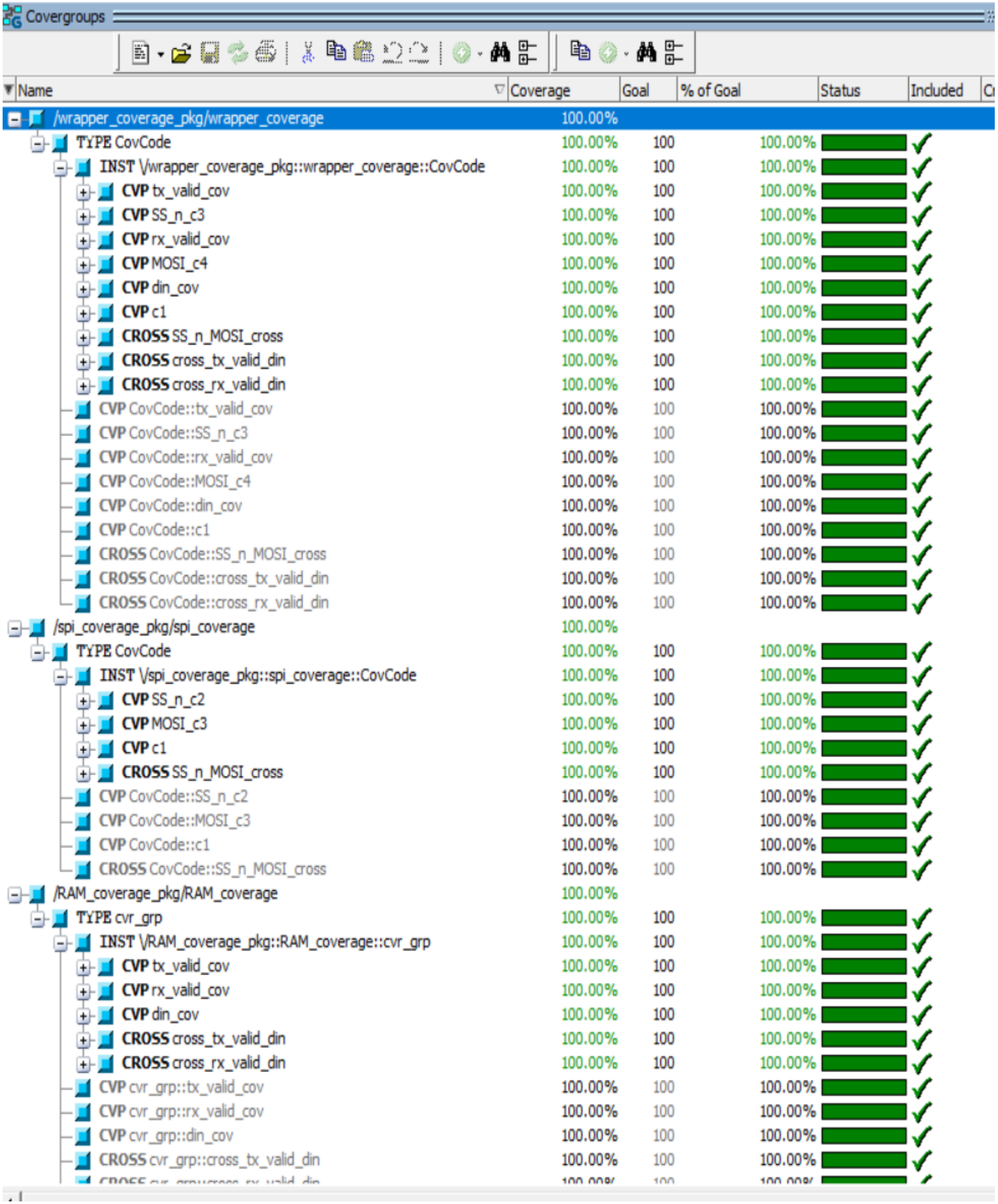
run.do

```
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.wrapper_top -classdebug -uvmcontrol=all -cover -sv_seed random
run 0
add wave -position insertpoint  \
sim:/wrapper_shared::cs_s  \
sim:/wrapper_shared::ns_s
add wave -position insertpoint  \
sim:/wrapper_top/DUT/MISO  \
sim:/wrapper_top/DUT/tx_valid \
sim:/wrapper_top/DUT/tx_data_dout \
sim:/wrapper_top/DUT/rx_valid \
sim:/wrapper_top/DUT/rx_data_din \
sim:/wrapper_top/DUT/rst_n \
sim:/wrapper_top/DUT/MOSI \
sim:/wrapper_top/DUT/SS_n \
sim:/wrapper_top/DUT/clk
add wave -position insertpoint  \
sim:/wrapper_top/Golden/MISO  \
sim:/wrapper_top/Golden/tx_valid \
sim:/wrapper_top/Golden/tx_data \
sim:/wrapper_top/Golden/rx_valid \
sim:/wrapper_top/Golden/rx_data
add wave -position insertpoint  \
sim:/uvm_root/uvm_test_top/wr_env/sb/correct_count \
sim:/uvm_root/uvm_test_top/wr_env/sb/error_count
run -all
coverage exclude -src RAM.sv -line 34 -code b
coverage exclude -src SPI_slave.sv -line 38 -code b
coverage exclude -src SPI_slave.sv -line 39 -code s
coverage exclude -src SPI_slave.sv -line 81 -code b
coverage exclude -src RAM.sv -line 34 -code s
coverage save wrapper.ucdb -onexit -du work.WRAPPER
coverage report -detail -cvg -directive -comments -output wrapper_fcover_report.txt {}
quit -sim
vcover report wrapper.ucdb -details -annotate -all -output wrapper_co.txt
```

src_files.list

```
RAM_if.sv
RAM.sv
Gold_RAM_if.sv
Single_Port_RAM.sv
RAM_config.sv
RAM_seq_item.sv
RAM_sequencer.sv
RAM_monitor.sv
RAM_driver.sv
RAM_agent.sv
RAM_scoreboard.sv
RAM_coverage_pkg.sv
RAM_env.sv
RAM_sequences.sv
spi_interface.sv
spi_shared_pkg.sv
golden_interface.sv
+define+SIM SPI_slave.sv
SPI.sv
spi_seq_item.sv
spi_config_obj.sv
spi_driver.sv
spi_monitor.sv
spi_sequencer.sv
spi_agent.sv
spi_coverage.sv
spi_scoreboard.sv
spi_env.sv
spi_sequences.sv
Golden_wrapper_if.sv
Golden_wrapper.sv
wrapper_if.sv
wrapper_shared_pkg.sv
SPI_wrapper.sv
wrapper_seq_item.sv
wrapper_sequencer.sv
wrapper_config_obj.sv
wrapper_monitor.sv
wrapper_driver.sv
wrapper_agent.sv
wrapper_coverage.sv
wrapper_scoreboard.sv
wrapper_env.sv
wrapper_sequences.sv
wrapper_test.sv
wrapper_sva.sv
wrapper_top.sv
```

Functional Coverage Report



Assertions Result

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0	
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0	
/wrapper_sequences_pkg::wrapper_write_sequence::body/#ublk#142042551#35/immed__3...	Immediate	SVA	on	0	1	
/wrapper_sequences_pkg::wrapper_read_sequence::body/#ublk#142042551#55/immed__5...	Immediate	SVA	on	0	1	
/wrapper_sequences_pkg::wrapper_read_write_sequence::body/#ublk#142042551#75/imm...	Immediate	SVA	on	0	1	
/wrapper_top/DUT/SLAVE_instance/assert__p_IDLE_to_CHK_CMD	Concurrent	SVA	on	0	1	
/wrapper_top/DUT/SLAVE_instance/assert__p_CHK_CMD_to_WRITE	Concurrent	SVA	on	0	1	
/wrapper_top/DUT/SLAVE_instance/assert__p_CHK_CMD_to_READ_ADD	Concurrent	SVA	on	0	1	
/wrapper_top/DUT/SLAVE_instance/assert__p_CHK_CMD_to_READ_DATA	Concurrent	SVA	on	0	1	
/wrapper_top/DUT/SLAVE_instance/assert__p_WRITE_to_IDLE	Concurrent	SVA	on	0	1	
/wrapper_top/DUT/SLAVE_instance/assert__p_READ_ADD_to_IDLE	Concurrent	SVA	on	0	1	
/wrapper_top/DUT/SLAVE_instance/assert__p_READ_DATA_to_IDLE	Concurrent	SVA	on	0	1	
/wrapper_top/DUT/SLAVE_instance/assert__p_counter_decrement	Concurrent	SVA	on	0	1	
/wrapper_top/DUT/sva_inst/assert__p_reset_outputs_inactive	Concurrent	SVA	on	0	1	
/wrapper_top/DUT/sva_inst/assert__stable_miso	Concurrent	SVA	on	0	1	

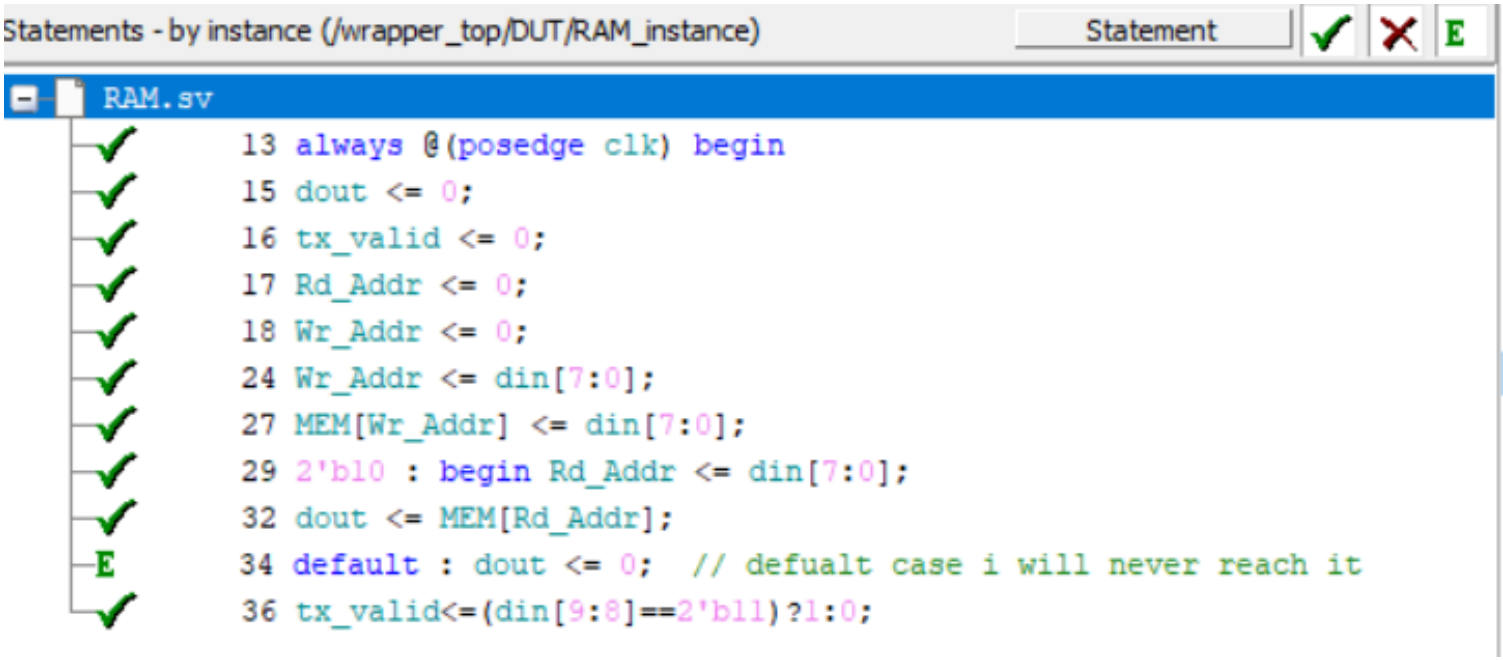
Assertions Coverage

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included
/wrapper_top/DUT/SLAVE_instance/cover__p_counter_decr...	SVA	✓	Off	21724	1	Unlimit...	1	100%		✓
/wrapper_top/DUT/SLAVE_instance/cover__p_READ_DATA_...	SVA	✓	Off	326	1	Unlimit...	1	100%		✓
/wrapper_top/DUT/SLAVE_instance/cover__p_READ_ADD_t...	SVA	✓	Off	408	1	Unlimit...	1	100%		✓
/wrapper_top/DUT/SLAVE_instance/cover__p_WRITE_to_ID...	SVA	✓	Off	979	1	Unlimit...	1	100%		✓
/wrapper_top/DUT/SLAVE_instance/cover__p_CHK_CMD_to...	SVA	✓	Off	403	1	Unlimit...	1	100%		✓
/wrapper_top/DUT/SLAVE_instance/cover__p_CHK_CMD_to...	SVA	✓	Off	464	1	Unlimit...	1	100%		✓
/wrapper_top/DUT/SLAVE_instance/cover__p_CHK_CMD_to...	SVA	✓	Off	1086	1	Unlimit...	1	100%		✓
/wrapper_top/DUT/SLAVE_instance/cover__p_IDLE_to_CHK...	SVA	✓	Off	1969	1	Unlimit...	1	100%		✓
/wrapper_top/DUT/sva_inst/cover__stable_miso	SVA	✓	Off	21488	1	Unlimit...	1	100%		✓
/wrapper_top/DUT/sva_inst/cover__p_reset_outputs_inacti...	SVA	✓	Off	275	1	Unlimit...	1	100%		✓

Feature	Assertion
An assertion ensures that whenever reset is asserted, the outputs (MISO, rx_valid, and rx_data) are all inactive.	@(posedge clk) (!rst_n) |=> (MISO == 0 && rx_valid == 0 && rx_data == 0);
An assertion to make sure that the MISO remains with a stable value as long as it is not a read data operation	@(posedge clk) disable iff (!rst_n) (cs_s!=READ_DATA) |=> $stable(MISO);

Code Coverage



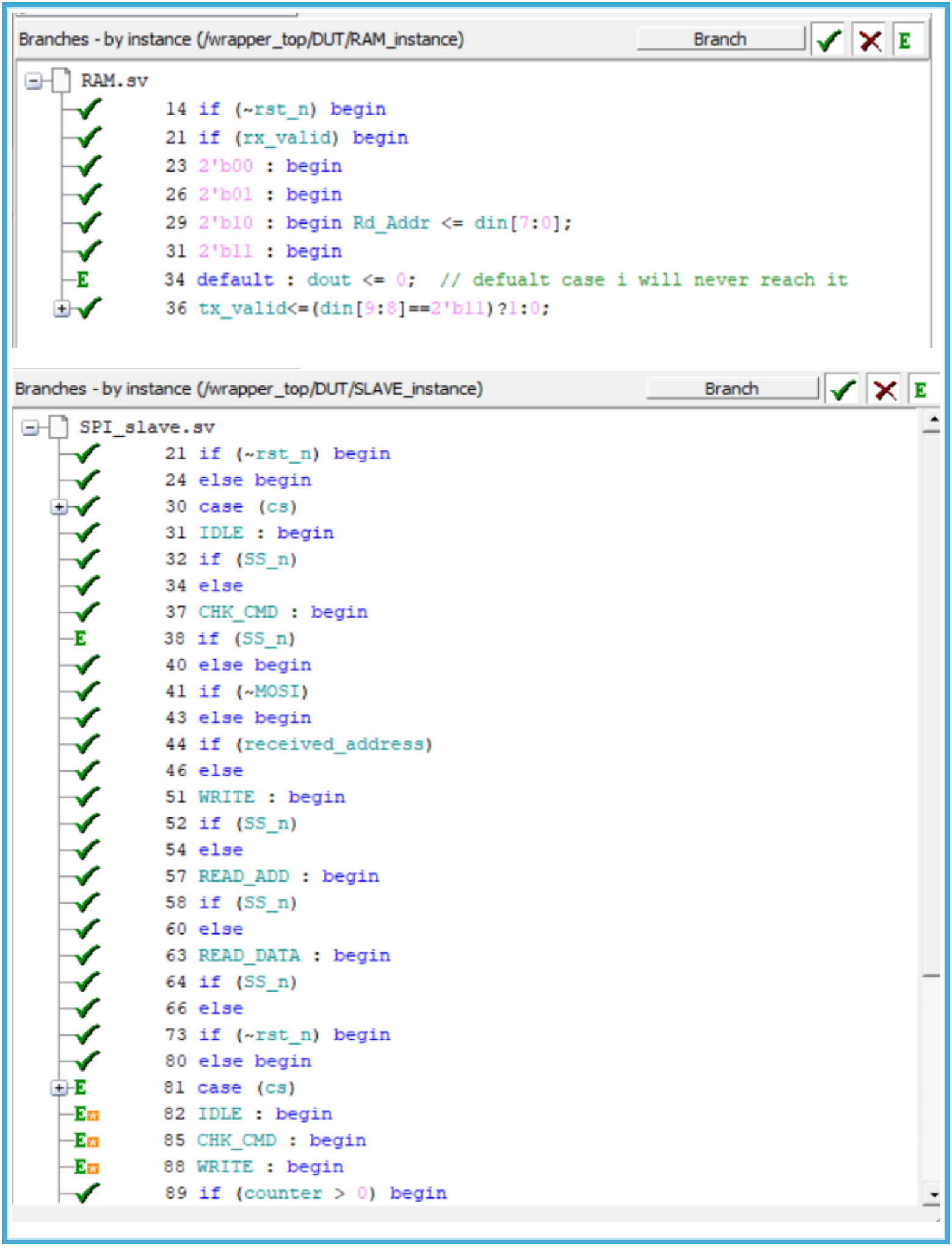
Statements - by instance (/wrapper_top/DUT/SLAVE_instance) | Statement | ✓ | ✗ | E

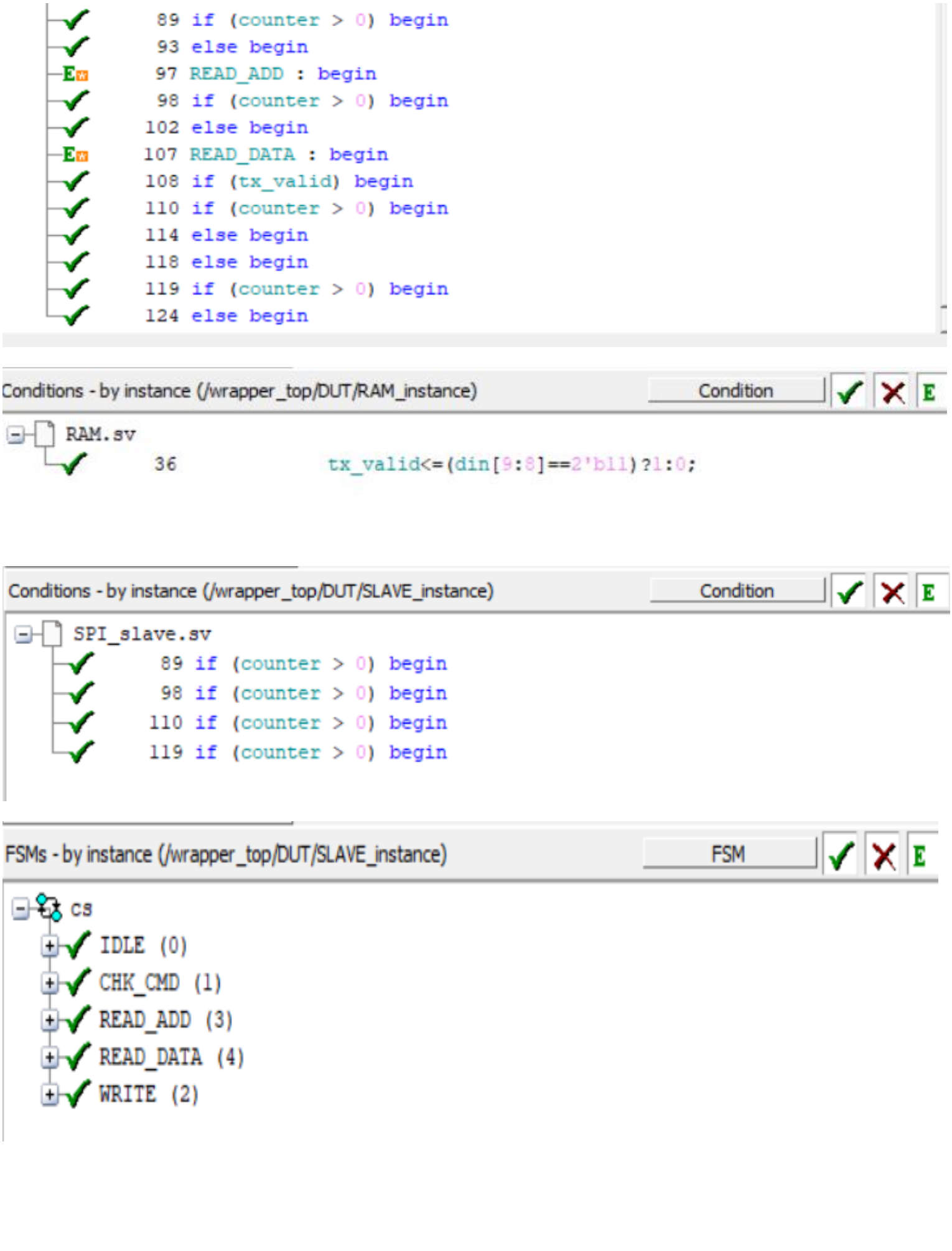
SPI_slave.sv

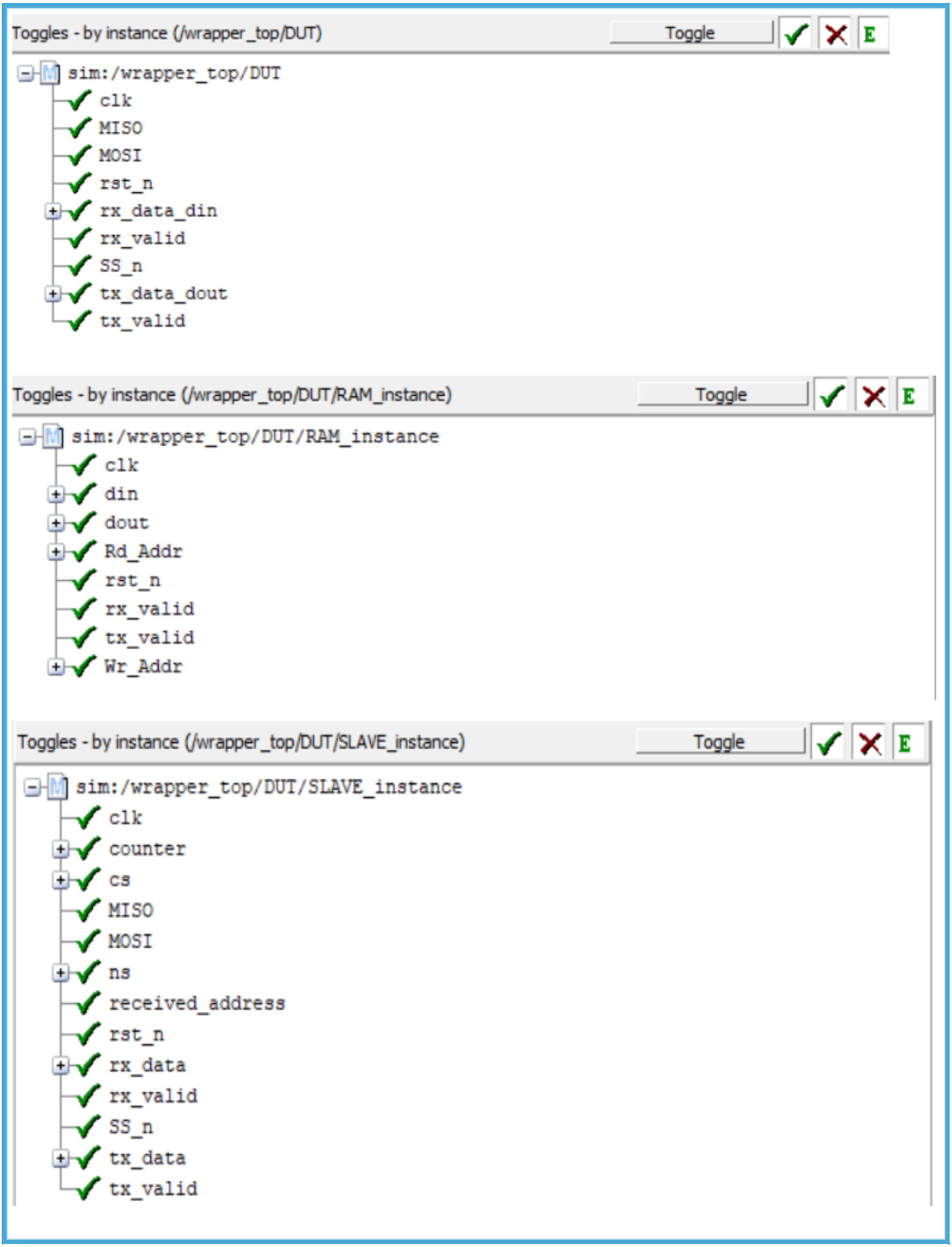
```
20 always @(posedge clk) begin
22 cs <= IDLE;
25 cs <= ns;
29 always @(*) begin
33 ns = IDLE;
35 ns = CHK_CMD;
E 39 ns = IDLE;
42 ns = WRITE;
45 ns = READ_DATA;
47 ns = READ_ADD; /// edit 1 replace with another
53 ns = IDLE;
55 ns = WRITE;
59 ns = IDLE;
61 ns = READ_ADD;
65 ns = IDLE;
67 ns = READ_DATA;
72 always @(posedge clk) begin
74 rx_data <= 0;
75 rx_valid <= 0;
76 counter<=0; // counter
77 received_address <= 0;
78 MISO <= 0;
83 rx_valid <= 0;
86 counter <= 10;
90 rx_data[counter-1] <= MOSI;
91 counter <= counter - 1;
94 rx_valid <= 1;
99 rx_data[counter-1] <= MOSI;
100 counter <= counter - 1;
```


```
100 counter <= counter - 1;
```


```
103 rx_valid <= 1;
104 received_address <= 1;
109 rx_valid<=0;
111 MISO <= tx_data[counter-1];
112 counter <= counter - 1;
115 received_address <= 0;
120 rx_data[counter-1] <= MOSI;
121 counter <= counter - 1;
122 rx_valid<=0;
125 rx_valid <= 1;
126 counter <= 9;
```


Branches - by instance (/wrapper_top/DUT/RAM_instance)
Branch
E
RAM.sv
14 if (~rst_n) begin
21 if (rx_valid) begin
23 2'b00 : begin
26 2'b01 : begin
29 2'b10 : begin Rd_Addr <= din[7:0];
31 2'b11 : begin
34 default : dout <= 0; // defualt case i will never reach it
36 tx_valid<=(din[9:8]==2'b11)?1:0;
Branches - by instance (/wrapper_top/DUT/SLAVE_instance)
Branch
E
SPI_slave.sv
21 if (~rst_n) begin
24 else begin
30 case (cs)
31 IDLE : begin
32 if (SS_n)
34 else
37 CHK_CMD : begin
38 if (SS_n)
40 else begin
41 if (~MOSI)
43 else begin
44 if (received_address)
46 else
51 WRITE : begin
52 if (SS_n)
54 else
57 READ_ADD : begin
58 if (SS_n)
60 else
63 READ_DATA : begin
64 if (SS_n)
66 else
73 if (~rst_n) begin
80 else begin
81 case (cs)
82 IDLE : begin
85 CHK_CMD : begin
88 WRITE : begin
89 if (counter > 0) begin


89 if (counter > 0) begin
93 else begin
97 READ_ADD : begin
98 if (counter > 0) begin
102 else begin
107 READ_DATA : begin
108 if (tx_valid) begin
110 if (counter > 0) begin
114 else begin
118 else begin
119 if (counter > 0) begin
124 else begin
Conditions - by instance (/wrapper_top/DUT/RAM_instance)
Condition
E
RAM.sv
36 tx_valid<=(din[9:8]==2'b11)?1:0;
Conditions - by instance (/wrapper_top/DUT/SLAVE_instance)
Condition
E
SPI_slave.sv
89 if (counter > 0) begin
98 if (counter > 0) begin
110 if (counter > 0) begin
119 if (counter > 0) begin
FSMs - by instance (/wrapper_top/DUT/SLAVE_instance)
FSM
E
cs
IDLE (0)
CHK_CMD (1)
READ_ADD (3)
READ_DATA (4)
WRITE (2)

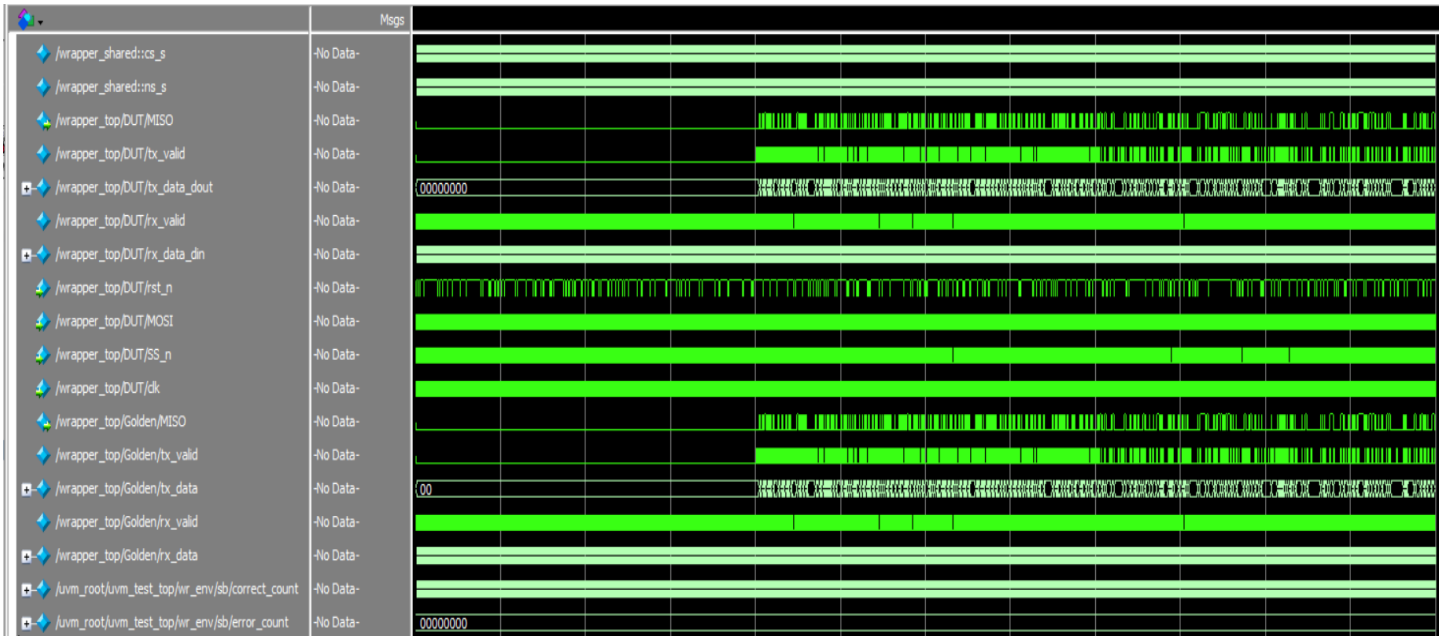

Toggles - by instance (/wrapper_top/DUT)
Toggle
E
sim:/wrapper_top/DUT
clk
MISO
MOSI
rst_n
rx_data_din
rx_valid
SS_n
tx_data_dout
tx_valid
Toggles - by instance (/wrapper_top/DUT/RAM_instance)
Toggle
E
sim:/wrapper_top/DUT/RAM_instance
clk
din
dout
Rd_Addr
rst_n
rx_valid
tx_valid
Wr_Addr
Toggles - by instance (/wrapper_top/DUT/SLAVE_instance)
Toggle
E
sim:/wrapper_top/DUT/SLAVE_instance
clk
counter
cs
MISO
MOSI
ns
received_address
rst_n
rx_data
rx_valid
SS_n
tx_data
tx_valid

Code Coverage Report Summary

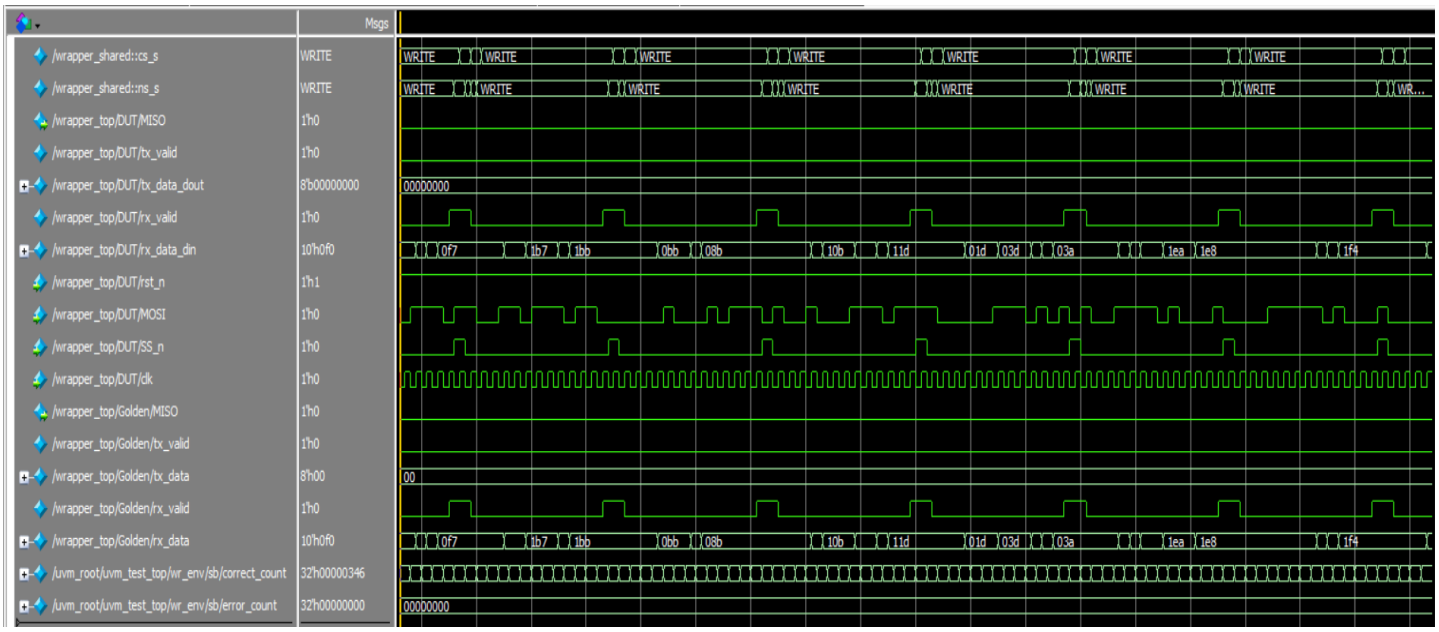
```
DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
/\wrapper_top#DUT /SLAVE_instance/cover__p_counter_decrement
                                        SLAVE  Verilog  SVA  SPI_slave.sv(188)
                                                                    21614 Covered
/\wrapper_top#DUT /SLAVE_instance/cover__p_READ_DATA_to_IDLE
                                        SLAVE  Verilog  SVA  SPI_slave.sv(187)
                                                                      312 Covered
/\wrapper_top#DUT /SLAVE_instance/cover__p_READ_ADD_to_IDLE
                                        SLAVE  Verilog  SVA  SPI_slave.sv(186)
                                                                      416 Covered
/\wrapper_top#DUT /SLAVE_instance/cover__p_WRITE_to_IDLE
                                        SLAVE  Verilog  SVA  SPI_slave.sv(185)
                                                                      964 Covered
/\wrapper_top#DUT /SLAVE_instance/cover__p_CHK_CMD_to_READ_DATA
                                        SLAVE  Verilog  SVA  SPI_slave.sv(184)
                                                                      409 Covered
/\wrapper_top#DUT /SLAVE_instance/cover__p_CHK_CMD_to_READ_ADD
                                        SLAVE  Verilog  SVA  SPI_slave.sv(183)
                                                                      490 Covered
/\wrapper_top#DUT /SLAVE_instance/cover__p_CHK_CMD_to_WRITE
                                        SLAVE  Verilog  SVA  SPI_slave.sv(182)
                                                                     1085 Covered
/\wrapper_top#DUT /SLAVE_instance/cover__p_IDLE_to_CHK_CMD
                                        SLAVE  Verilog  SVA  SPI_slave.sv(181)
                                                                     1999 Covered
/\wrapper_top#DUT /sva_inst/cover__stable_miso
                                        wrapper_sva Verilog  SVA  wrapper_sva.sv(19)
                                                                    21586 Covered
/\wrapper_top#DUT /sva_inst/cover__p_reset_outputs_inactive
                                        wrapper_sva Verilog  SVA  wrapper_sva.sv(12)
                                                                      326 Covered

TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 10

ASSERTION RESULTS:
---------------------------------------------------------------------
Name                 File(Line)                   Failure      Pass
                                                  Count        Count
---------------------------------------------------------------------
/\wrapper_top#DUT /SLAVE_instance/assert__p_counter_decrement
                     SPI_slave.sv(180)                    0          1
/\wrapper_top#DUT /SLAVE_instance/assert__p_READ_DATA_to_IDLE
                     SPI_slave.sv(179)                    0          1
/\wrapper_top#DUT /SLAVE_instance/assert__p_READ_ADD_to_IDLE
                     SPI_slave.sv(178)                    0          1
/\wrapper_top#DUT /SLAVE_instance/assert__p_WRITE_to_IDLE
                     SPI_slave.sv(177)                    0          1
/\wrapper_top#DUT /SLAVE_instance/assert__p_CHK_CMD_to_READ_DATA
                     SPI_slave.sv(176)                    0          1
/\wrapper_top#DUT /SLAVE_instance/assert__p_CHK_CMD_to_READ_ADD
                     SPI_slave.sv(175)                    0          1
/\wrapper_top#DUT /SLAVE_instance/assert__p_CHK_CMD_to_WRITE
                     SPI_slave.sv(174)                    0          1
/\wrapper_top#DUT /SLAVE_instance/assert__p_IDLE_to_CHK_CMD
                     SPI_slave.sv(173)                    0          1
/\wrapper_top#DUT /sva_inst/assert__stable_miso
                     wrapper_sva.sv(18)                   0          1
/\wrapper_top#DUT /sva_inst/assert__p_reset_outputs_inactive
                     wrapper_sva.sv(11)                   0          1

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

Wave Forms and Output Summary

- **Complete Waveform**



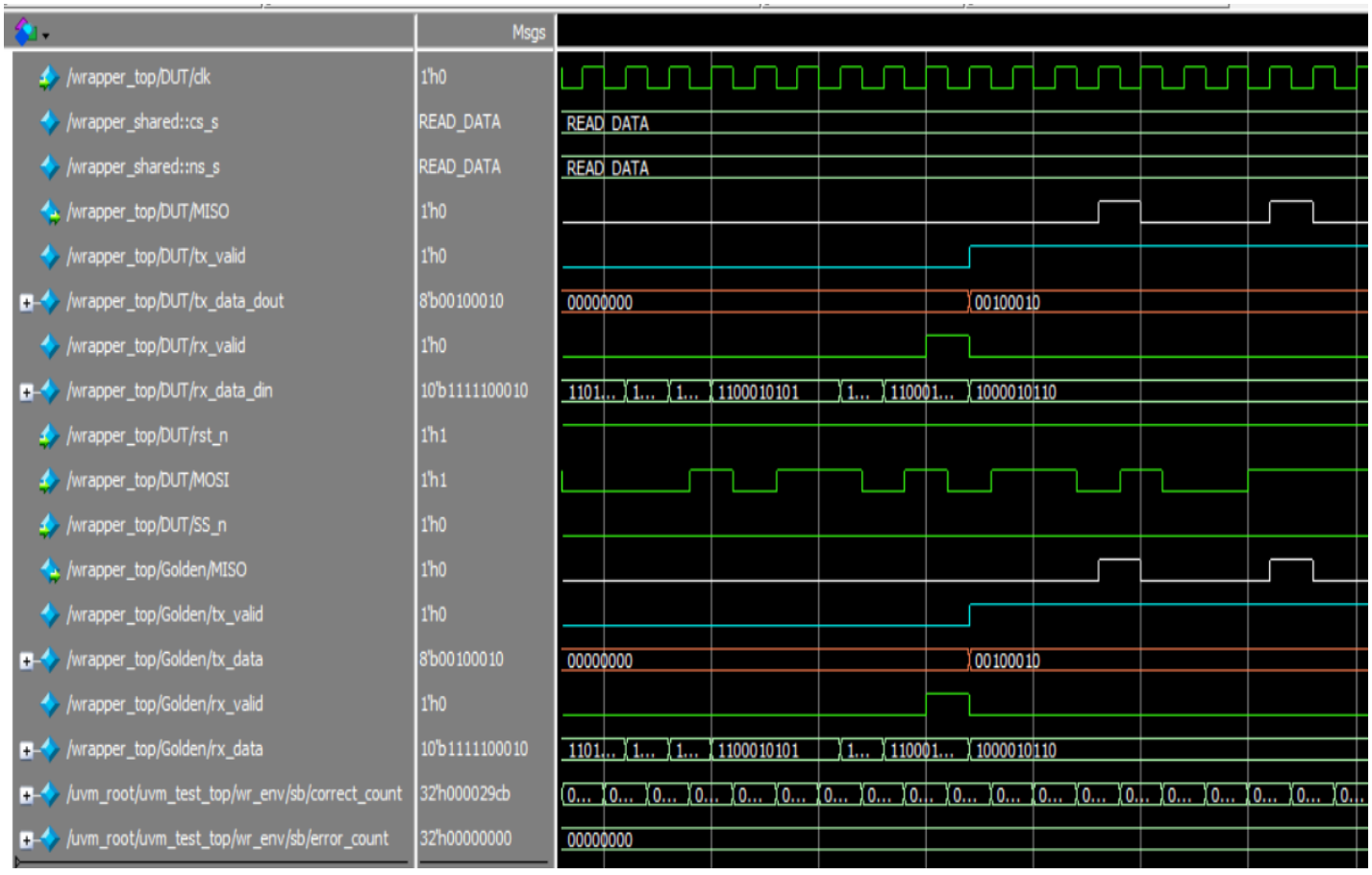
Write Seq

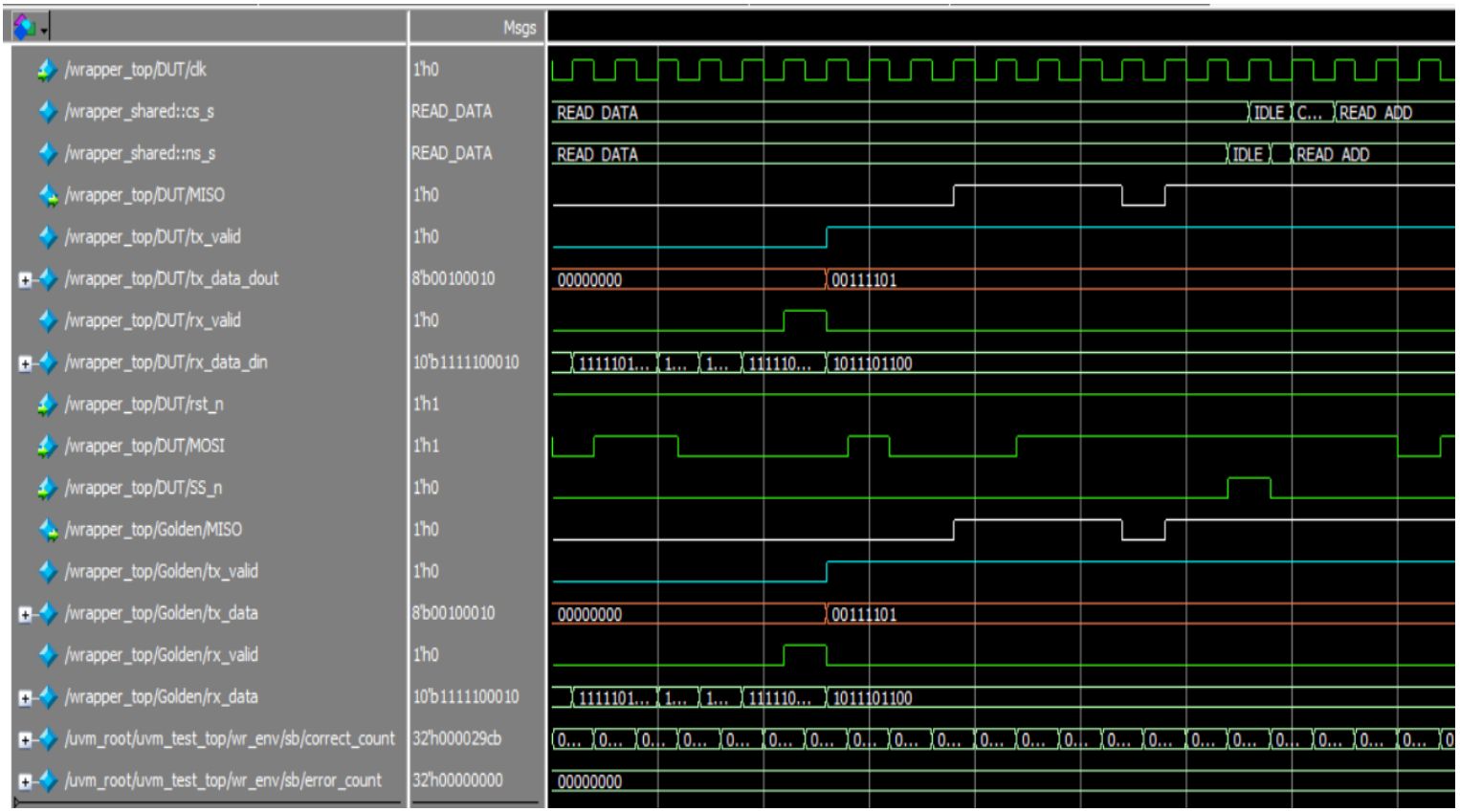


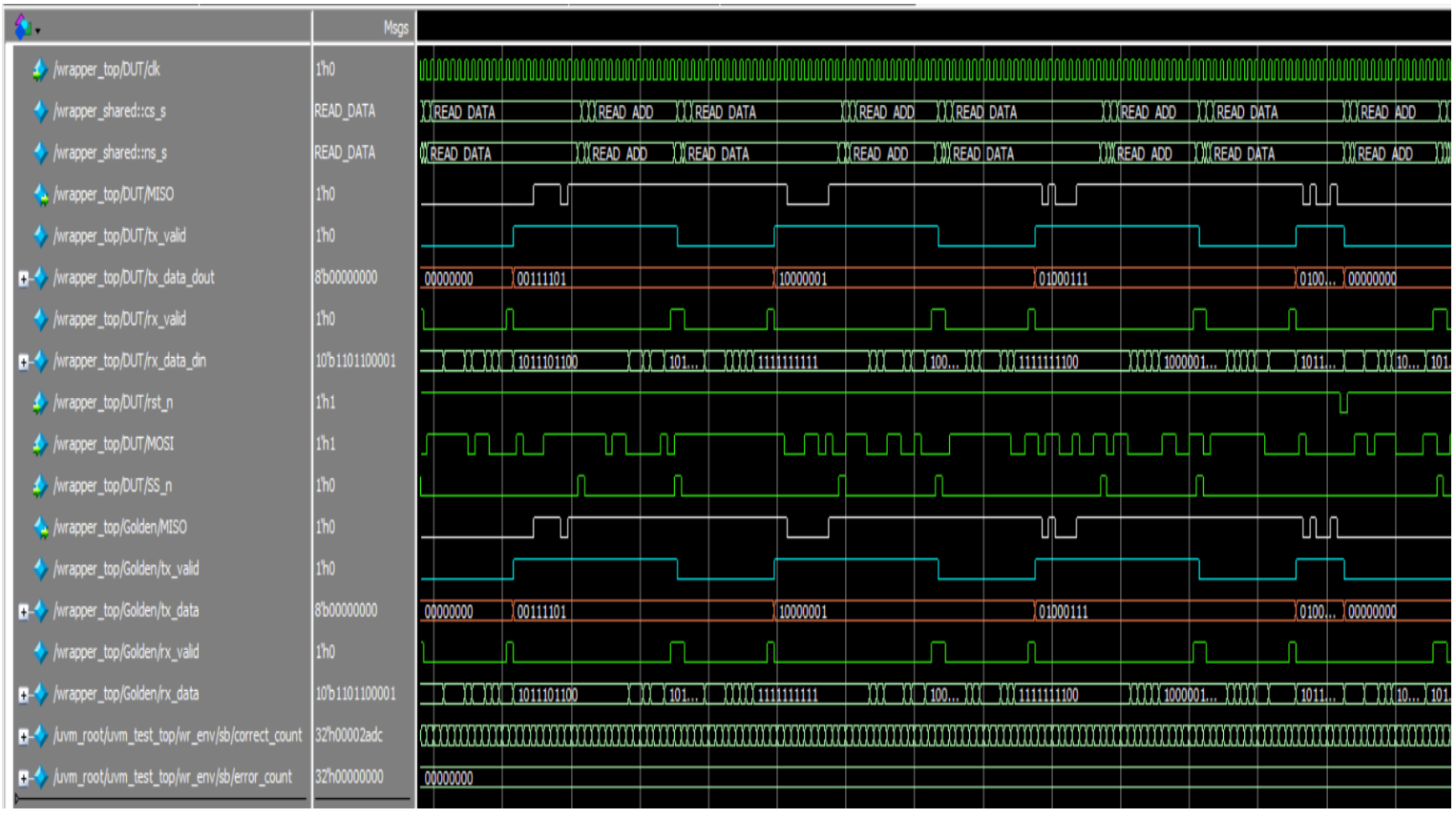
Memory Content at the end of writing seq

```
sim:/wrapper_top/DUT/RAM_instance/MEM @ 19845 ns
255 : 8'h00 8'h00 8'hba 8'hf0 8'h00 8'h00 8'hd9 8'h44 8'h0b 8'h75 8'h11 8'hcb 8'h00 8'hfe 8'h00 8'h19
239 : 8'h00 8'h00 8'hfe 8'h00 8'hca 8'h62 8'hce 8'h24 8'hc5 8'h00 8'h00 8'h00 8'h00 8'h00 8'h50 8'h72
223 : 8'h21 8'h3f 8'h00 8'h2e 8'h85 8'h00 8'h00 8'h9b 8'hbd 8'h00 8'h91 8'h15 8'h4d 8'hce 8'h37 8'h23
207 : 8'h13 8'h00 8'h4e 8'h00 8'h5c 8'hfd 8'h2c 8'h00 8'h00 8'h00 8'h0a 8'h00 8'h00 8'hc9 8'h18 8'h52
191 : 8'hc6 8'h09 8'hbf 8'hf3 8'h9e 8'h70 8'hab 8'h71 8'hd1 8'h18 8'h6c 8'hd4 8'h7c 8'hc5 8'h00 8'h00
175 : 8'h19 8'h1f 8'h78 8'he0 8'h46 8'hbf 8'h00 8'h00 8'h00 8'h5c 8'h59 8'h00 8'h2a 8'h45 8'he0 8'hc0
159 : 8'h76 8'h50 8'h00 8'h54 8'h9e 8'h00 8'hb6 8'h00 8'h00 8'had 8'h61 8'h00 8'hb5 8'hac 8'hc4 8'h00
143 : 8'h07 8'h74 8'h31 8'h00 8'hf9 8'hed 8'h00 8'h9d 8'ha6 8'h00 8'h32 8'h95 8'h9e 8'h00 8'hba 8'h7d
127 : 8'h00 8'h00 8'hed 8'h30 8'h00 8'h14 8'h1f 8'h35 8'h45 8'h00 8'h6a 8'h00 8'h00 8'h00 8'ha1 8'h9d
111 : 8'h32 8'h00 8'h00 8'h24 8'h43 8'h49 8'ha3 8'h2f 8'h00 8'h37 8'h00 8'h00 8'h00 8'h2a 8'h75 8'h00
 95 : 8'h89 8'h00 8'h54 8'h3e 8'h00 8'hab 8'h3f 8'h00 8'h39 8'h00 8'h25 8'h00 8'hca 8'h1a 8'h00 8'h00
 79 : 8'hd4 8'hb8 8'h1f 8'haa 8'h30 8'h00 8'h00 8'h00 8'hbf 8'h00 8'h31 8'h6f 8'he6 8'h23 8'hcc 8'h1b
 63 : 8'h00 8'h00 8'hb8 8'hba 8'h00 8'hf4 8'ha2 8'hd9 8'h6a 8'h00 8'hed 8'hea 8'h48 8'ha3 8'h5c 8'h00
 47 : 8'h00 8'had 8'h00 8'h00 8'hd3 8'h47 8'h2a 8'h00 8'h00 8'h00 8'h14 8'h64 8'hb2 8'h00 8'he0 8'hbc
 31 : 8'hf3 8'h11 8'hf0 8'h7b 8'h50 8'h00 8'h00 8'h00 8'h00 8'h00 8'h3f 8'h00 8'h00 8'hc1 8'h46 8'h4a
 15 : 8'hf0 8'h00 8'h4b 8'h00 8'h5d 8'had 8'h9e 8'h00 8'hab 8'h6e 8'h25 8'h58 8'h39 8'h2d 8'h6e 8'hf6
```

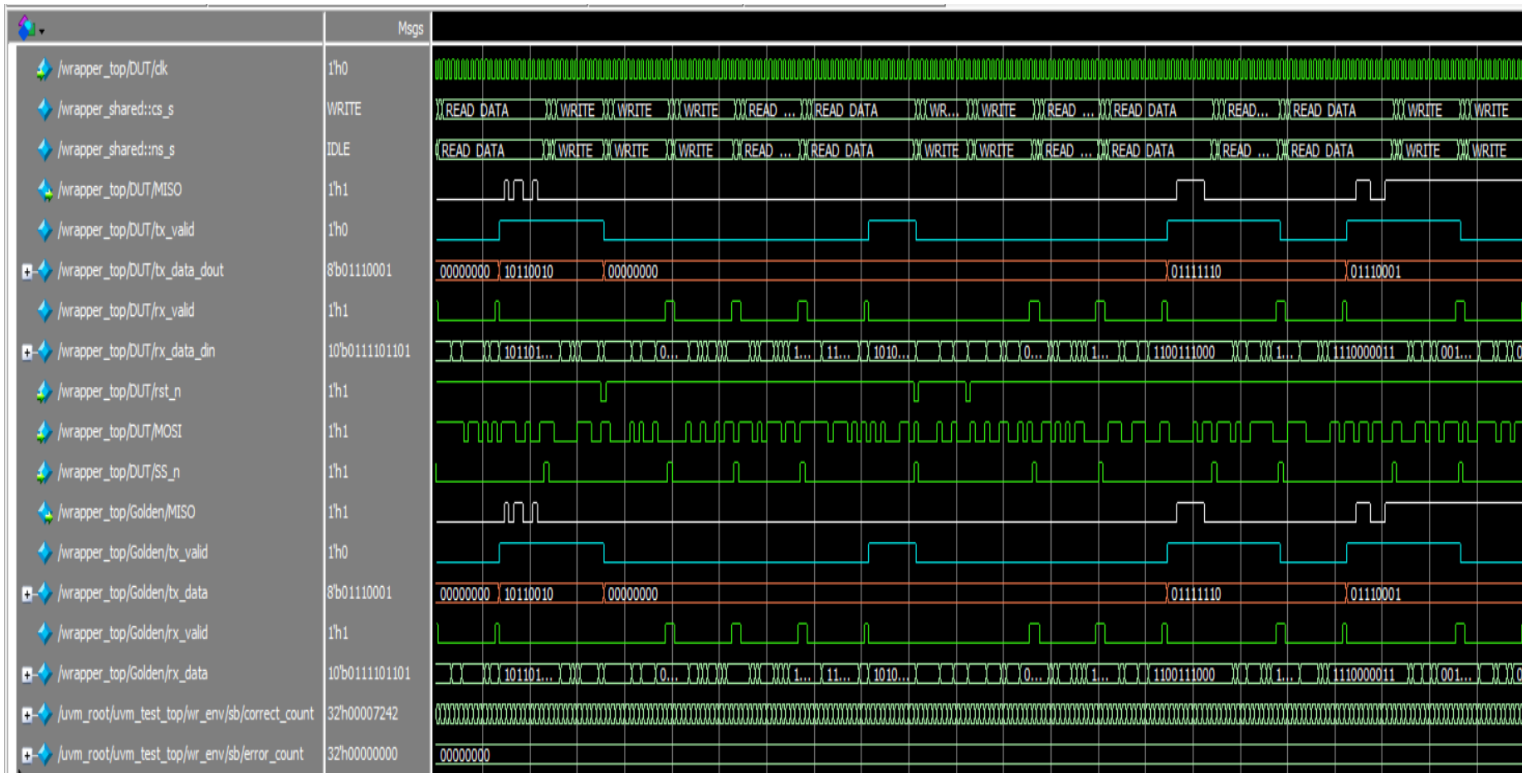
Read seq

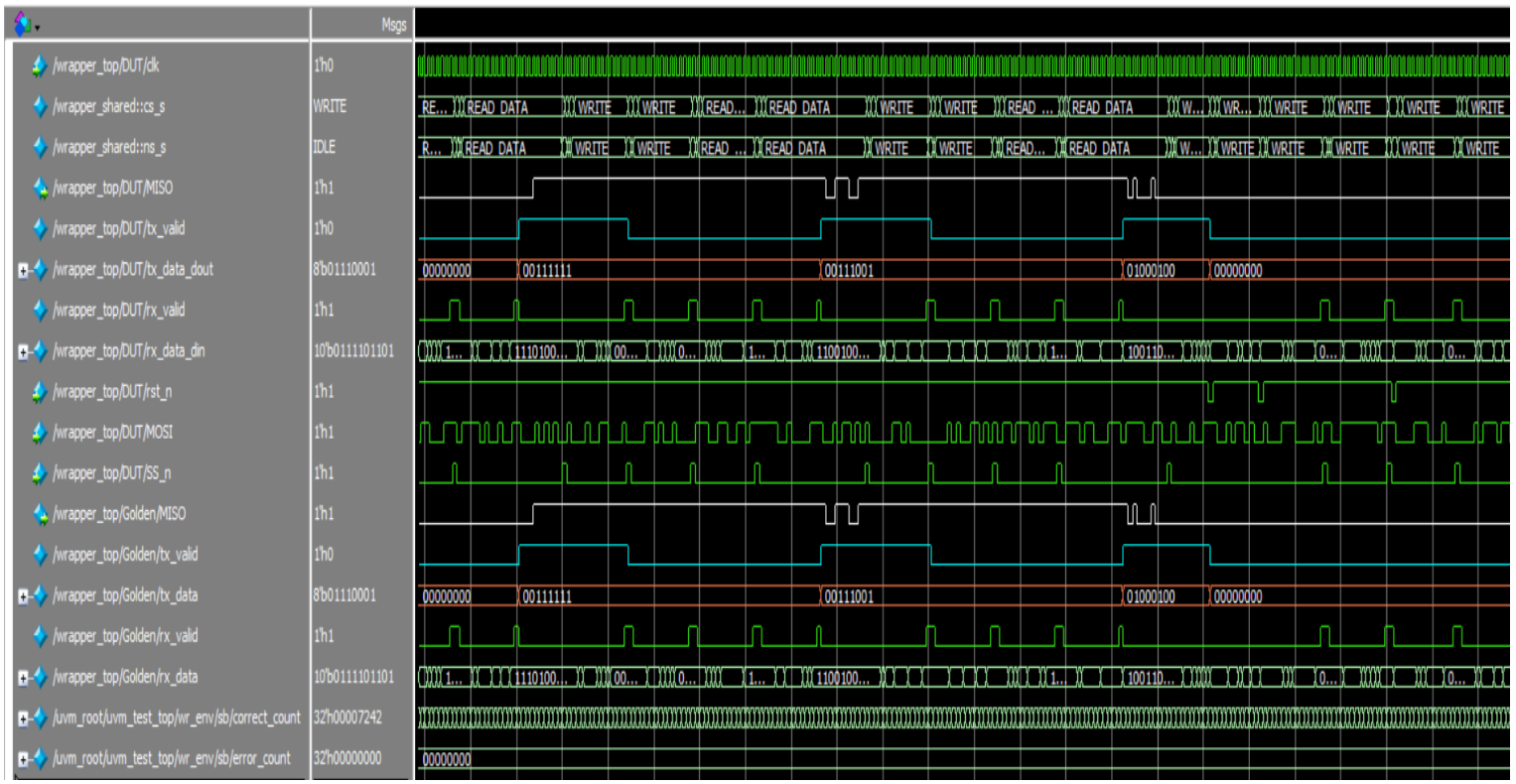



Msgs
/wrapper_top/DUT/clk
/wrapper_shared::cs_s
/wrapper_shared::ns_s
/wrapper_top/DUT/MISO
/wrapper_top/DUT/tx_valid
/wrapper_top/DUT/tx_data_dout
/wrapper_top/DUT/rx_valid
/wrapper_top/DUT/rx_data_din
/wrapper_top/DUT/rst_n
/wrapper_top/DUT/MOSI
/wrapper_top/DUT/SS_n
/wrapper_top/Golden/MISO
/wrapper_top/Golden/tx_valid
/wrapper_top/Golden/tx_data
/wrapper_top/Golden/rx_valid
/wrapper_top/Golden/rx_data
/uvm_root/uvm_test_top/wr_env/sb/correct_count
/uvm_root/uvm_test_top/wr_env/sb/error_count
1'h0
READ_DATA
READ_DATA
1'h0
1'h0
8'b00100010
1'h0
10'b1111100010
1'h1
1'h1
1'h0
1'h0
1'h0
8'b00100010
1'h0
10'b1111100010
32'h000029cb
32'h00000000
IDLE
C...
READ_ADD
IDLE
READ_ADD
00000000
00111101
1111101...
1...
1...
111110...
1011101100
00000000
00111101
1111101...
1...
1...
111110...
1011101100
00000000


Msgs
/wrapper_top/DUT/clk
/wrapper_shared::cs_s
/wrapper_shared::ns_s
/wrapper_top/DUT/MISO
/wrapper_top/DUT/tx_valid
/wrapper_top/DUT/tx_data_dout
/wrapper_top/DUT/rx_valid
/wrapper_top/DUT/rx_data_din
/wrapper_top/DUT/rst_n
/wrapper_top/DUT/MOSI
/wrapper_top/DUT/SS_n
/wrapper_top/Golden/MISO
/wrapper_top/Golden/tx_valid
/wrapper_top/Golden/tx_data
/wrapper_top/Golden/rx_valid
/wrapper_top/Golden/rx_data
/uvm_root/uvm_test_top/wr_env/sb/correct_count
/uvm_root/uvm_test_top/wr_env/sb/error_count
1'h0
READ_DATA
READ_DATA
1'h0
1'h0
8'b00000000
1'h0
10'b1101100001
1'h1
1'h1
1'h0
1'h0
1'h0
8'b00000000
1'h0
10'b1101100001
32'h00002adc
READ_ADD
READ_DATA
00000000
00111101
10000001
01000111
0100...
00000000
1011101100
101...
1111111111
100...
1111111100
1000001...
1011...
00000000

Read Write seq





Output Summary

```
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 60002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO RAM_scoreboard.sv(47) @ 60002: uvm_test_top.R_env.sb [RAM] ===== RAM Scoreboard Report =====
# Correct Results: 30001
# Errors: 0
# UVM_INFO spi_scoreboard.sv(45) @ 60002: uvm_test_top.sp_env.sb [SLAVE] ===== SLAVE Scoreboard Report =====
# Correct Results: 30001
# Errors: 0
# UVM_INFO wrapper_scoreboard.sv(45) @ 60002: uvm_test_top.wr_env.sb [WRAPPER] ===== WRAPPER Scoreboard Report =====
# Correct Results: 30001
# Errors: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   11
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RAM]     1
# [RNTST]     1
# [SLAVE]     1
# [TEST_DONE]     1
# [WRAPPER]     1
# [run_phase]     4
# ** Note: $finish    : C:/questasim64_2024.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 60002 ns  Iteration: 61  Instance: /wrapper_top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2024.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```