Project 2:

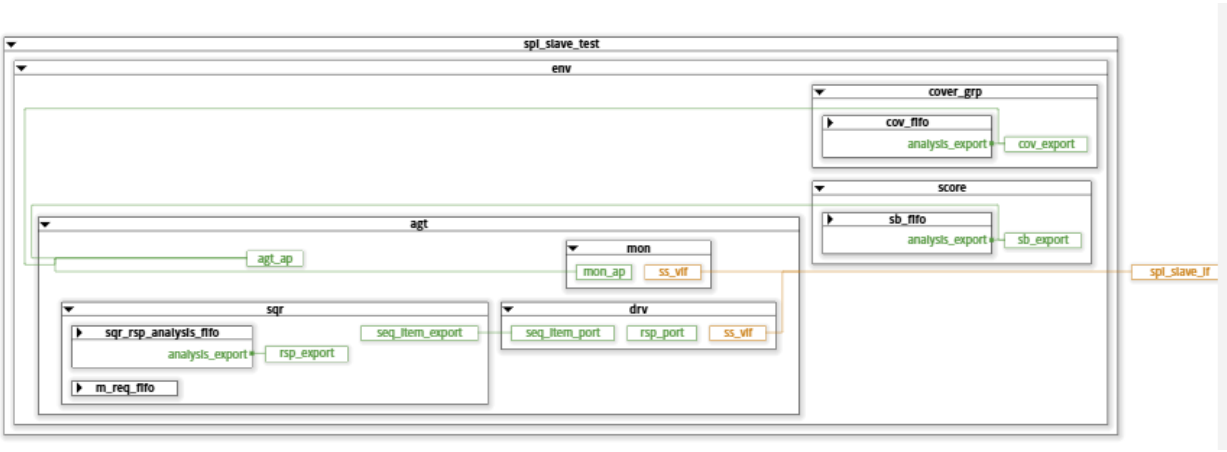
SPI with Single Port RAM Verification Project

Group number	
3	كريم أيمن محمود يحيى
2	Kareem Hassan Atif
2	Ahmed Mohamed Abdelmoneam

Submitted to: Eng. Kareem Waseem

1-SPI Slave:

1. SPI Slave UVM architecture



Comment : first top set the virtual interface in configuration data base ,then test get this virtual interface and set configuration data base with all interface then agent get this configuration object and use it to connect between driver and monitor with the interface of DUT , Driver ask sequencer if there is data from sequence item , if there is data take data and give it to interface then monitor get data from interface and give it to scoreboard to compare it with refrence model by using fifo to get sequence item and give it to scoreboard , the same way for collector get sequence item and cover if sequence done , then after the simulation done test raise object and end simulation .

2. Design file

```
module SLAVE (spi_slave_if.DUT ss_if);

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

reg [3:0] counter;
reg       received_address;

reg [2:0] cs, ns;
always @(posedge ss_if.clk) begin
    if (~ss_if.rst_n) begin
        cs <= IDLE;
    end
    else begin
        cs <= ns;
    end
end




always @(*) begin
    case (cs)
        IDLE : begin
            if (ss_if.SS_n)
                ns = IDLE;
            else
                ns = CHK_CMD;
        end
        CHK_CMD : begin
            if (ss_if.SS_n)
                ns = IDLE;
            else begin
                if (~ss_if.MOSI)
                    ns = WRITE;
                else begin
                      if (received_address)
                        ns = READ_DATA;
                    else
                        ns = READ_ADD;
```

```
                end
            end
        end
        WRITE : begin
            if (ss_if.SS_n)
                ns = IDLE;
            else
                ns = WRITE;
        end
        READ_ADD : begin
            if (ss_if.SS_n)
                ns = IDLE;
            else
                ns = READ_ADD;
        end
        READ_DATA : begin
            if (ss_if.SS_n)
                ns = IDLE;
            else
                ns = READ_DATA;
        end
    endcase
end

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

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

            READ_DATA : begin
                if (ss_if.tx_valid) begin
                    if (counter > 0) begin
                        ss_if.MISO <= ss_if.tx_data[counter-1];
                        counter <= counter - 1;
                    end
                    else begin
                        received_address <= 0;
                         ss_if.rx_valid <= 0;
                    end
                end
                else begin
                    if (counter > 0) begin // write data before going to read
data
                        ss_if.rx_data[counter-1] <= ss_if.MOSI;
                        counter <= counter - 1;
                    end
                    else begin
                        ss_if.rx_valid <= 1;
                        counter <= 8;
                    end
                end
            end
        endcase
    end
end
// READ_ADD and READ_DATA is replaced
// counter not equal 0 at reset
 `ifdef SIM
```

```
sequence write_add_seq;
  (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 0)[*3];
endsequence
sequence write_data_seq;
  (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 0)[*2]
##1(ss_if.MOSI==1);
endsequence
sequence read_add_seq;
  (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 1)[*2]
##1(ss_if.MOSI==0);
endsequence

sequence read_data_seq;
  (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 1)[*3];
endsequence


property chck_rx_valid;
  @(posedge ss_if.clk) disable iff(~ss_if.rst_n)
    (write_add_seq or write_data_seq or read_add_seq or read_data_seq) |=> ##9
($rose(ss_if.rx_valid) && $rose(ss_if.SS_n)[->1]);
endproperty

   property chck_reset ;
   @(posedge ss_if.clk)   (~ss_if.rst_n) |=>(ss_if.MISO ==0 && ss_if.rx_valid==0
&& ss_if.rx_data==0);
   endproperty

  property chck_state_idle;
   @(posedge ss_if.clk) disable iff(~ss_if.rst_n) (cs==IDLE && !ss_if.SS_n)
|=>(cs==CHK_CMD);
  endproperty
  property chck_state_write;
   @(posedge ss_if.clk) disable iff(~ss_if.rst_n) (cs==CHK_CMD && !ss_if.SS_n &&
!ss_if.MOSI) |=>  (cs==WRITE);
  endproperty
  property chck_state_read_add;
   @(posedge ss_if.clk) disable iff(~ss_if.rst_n)  (cs==CHK_CMD && !ss_if.SS_n &&
ss_if.MOSI && !received_address) |=>(cs==READ_ADD);
  endproperty
  property chck_state_read_data;
   @(posedge ss_if.clk) disable iff(~ss_if.rst_n)  (cs==CHK_CMD && !ss_if.SS_n &&
ss_if.MOSI && received_address) |=>(cs==READ_DATA);
  endproperty
   property chck_state_write_to_idle;
```

```
   @(posedge ss_if.clk)  (cs==WRITE &&(~ss_if.rst_n)) |=> (cs==IDLE);
   endproperty
   property chck_state_read_add_to_idle;
   @(posedge ss_if.clk)  (cs==READ_ADD && (~ss_if.rst_n)) |=> (cs==IDLE);
   endproperty
   property chck_state_read_datato_idle;
   @(posedge ss_if.clk)  (cs==READ_DATA && (~ss_if.rst_n)) |=> (cs==IDLE);
   endproperty


   assert property (chck_reset);
    assert property (chck_rx_valid);
     assert property (chck_state_idle);
    assert property (chck_state_write);
     assert property (chck_state_read_add);
     assert property (chck_state_read_data);
     assert property (chck_state_write_to_idle);
     assert property (chck_state_read_add_to_idle);
     assert property
(chck_state_read_datato_idle);


            cover property (chck_reset);
            cover property (chck_rx_valid);
            cover property (chck_state_idle);
            cover property (chck_state_write);
            cover property (chck_state_read_add);
            cover property (chck_state_read_data);
            cover property (chck_state_write_to_idle);
            cover property (chck_state_read_add_to_idle);
            cover property (chck_state_read_datato_idle);

 `endif
Endmodule
```

3. Design Bugs

```
1  if (received_address)
2                 ns = READ_ADD;
3             else
4                 ns = READ_DATA;
5         end
```

Figure 1 bug 1

```
1  if (received_address)
2                  ns = READ_DATA;
3              else
4                  ns = READ_ADD;
5          end
```

Figure 2 bug1 fixed

```
if (~rst_n) begin
        rx_data <= 0;
        rx_valid <= 0;
        received_address <= 0;
        MISO <= 0;
    end
```

Figure 3 bug 2

```
if (~ss_if.rst_n) begin
        ss_if.rx_data <= 0;
        ss_if.rx_valid <= 0;
        received_address <= 0;
        ss_if.MISO <= 0;
        counter<=0; // reset
    end
```

Figure 4 bug2 fixed

```
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
```

Figure 5 bug 3

```
if (ss_if.tx_valid) begin
                    if (counter > 0) begin
                        ss_if.MISO <= ss_if.tx_data[counter-1];
                        counter <= counter - 1;
                    end
                    else begin
                        received_address <= 0;
                         ss_if.rx_valid <= 0;
                    end
                end
```

Figure 6 bug 3 fixed

```
else begin
                        rx_valid <= 1;
                        counter <= 8;
                    end
```

Figure 7 bug 4

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

Figure 8 bug4 fixed

4. Refrence model file

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

   endcase
    end
end
endmodule
```

5. Interface file

```
interface spi_slave_if (clk);
  input  bit clk;
 logic           MOSI, rst_n, SS_n, tx_valid;
 logic      [7:0] tx_data;
logic [9:0] rx_data,rx_data_ref;
logic       rx_valid, MISO,MISO_ref , rx_valid_ref;
modport DUT(input clk,MOSI,rst_n,SS_n,tx_data,tx_valid, output
MISO,rx_data,rx_valid);
endinterface : spi_slave_if
```

6. Top file

```
import uvm_pkg::*;
`include "uvm_macros.svh"
import spi_slave_test_pkg::*;
import spi_slave_seq_item_package::*;
import shared_pkg::*;

module top();
    bit clk;
    initial begin
      forever #2 clk=~clk;
    end
    spi_slave_if ss_if(clk);
    SLAVE DUT(ss_if);
    SPI_Slave
dut(ss_if.MOSI,ss_if.SS_n,ss_if.clk,ss_if.rst_n,ss_if.rx_data_ref,ss_if.tx_valid,
ss_if.tx_data,ss_if.MISO_ref,ss_if.rx_valid_ref);
    initial begin
      uvm_config_db #(virtual spi_slave_if) ::
set(null,"uvm_test_top","SPI_IF",ss_if);
    run_test ("spi_slave_test");
    end
endmodule
```

7. Sequence item file

```
package spi_slave_seq_item_package;
import uvm_pkg::*;
 `include "uvm_macros.svh"
import shared_pkg::*;
class spi_slave_seq_item extends uvm_sequence_item;
  `uvm_object_utils(spi_slave_seq_item);
 rand logic [7:0] tx_data;
 logic [9:0] rx_data ,rx_data_ref;
 logic rx_valid, MISO,MISO_ref ,rx_valid_ref;
  logic MOSI;
 rand logic SS_n;
 rand logic rst_n;
 rand logic tx_valid;
 rand bit[0:10] array_rand;

  function new(string name= "spi_slave_seq_item");
     super.new(name);
 endfunction
   //reset
    constraint reset
    {
      rst_n  dist { 1:/98 , 0:/2};
    }
    //SS_n for all cases
    constraint serial_comm_all_cases
    {
      if(  array_rand[0:2] inside {3'b000, 3'b001, 3'b110} && counter_allcases%14
!=0) {
          SS_n==0;
      }
       else
        {
          SS_n==1;
        }
    }
    //SS_n for read data
    constraint serial_comm_read_data{
     if(  array_rand[0:2] inside {3'b111} && counter_read%24 !=0) {
         SS_n==0;
      }
       else
        {
          SS_n==1;
```

```
        }
    }
    //tx_valid for read data
    constraint trans_ram
    {
            if (array_rand[0:2] == 3'b111  && counter_read==23)
              {
                tx_valid==1;
              }
              if(array_rand[0:2] != 3'b111)
                {
                  tx_valid==0;
                }
    }
    // array_rand for all cases
    constraint mosi_in
    {
      if (SS_n_prev && !SS_n)
      array_rand[0:2] inside {3'b000, 3'b001, 3'b110 , 3'b111};
    }


    function void post_randomize;
      SS_n_prev=SS_n;
       if(counter_allcases<11 )
        MOSI=array_rand[counter_allcases];
    endfunction
    function update_counter_allcases;
     counter_allcases++;
    if (counter_allcases==14 )
    counter_allcases=0;
    endfunction

    function update_counter_read;
     counter_read++;
    if (counter_read==24 )
    counter_read=0;
    endfunction

 function string convert2string();
 return $sformatf("%s reset = 0b%0b , mosi=%b , miso=%b , ss_n = %b
",super.convert2string(),rst_n,MOSI,MISO,SS_n);
 endfunction
 function string convert2string_stimulus();
 return $sformatf("reset = 0b%0b , mosi=%b , ss_n = %b ",rst_n,MOSI,SS_n);
```

```
 endfunction
endclass
endpackage
```

8. Main sequence file

```
package spi_slave_main_sequence_package;
import uvm_pkg::*;
`include "uvm_macros.svh"
import spi_slave_seq_item_package::*;
import shared_pkg::*;
class spi_slave_main_sequence extends  uvm_sequence #(spi_slave_seq_item);
 `uvm_object_utils(spi_slave_main_sequence);
 spi_slave_seq_item seq_item_main;
 function new(string name= "spi_slave_main_sequence");
     super.new(name);
 endfunction
 task body ;
 seq_item_main= spi_slave_seq_item :: type_id :: create("seq_item_main");
 repeat(1000) begin
 start_item(seq_item_main);
 seq_item_main.reset.constraint_mode(1);
 seq_item_main.serial_comm_all_cases.constraint_mode(1);
 seq_item_main.mosi_in.constraint_mode(1);
 seq_item_main.serial_comm_read_data.constraint_mode(0);
 seq_item_main.trans_ram.constraint_mode(0);
 if(counter_allcases==0) begin
  seq_item_main.array_rand.rand_mode(1);
    $display("array_rand=%b , time =%t
,conter=%d",seq_item_main.array_rand,$time,counter_allcases);
 end
 else begin
  seq_item_main.array_rand.rand_mode(0);
 end
 assert (seq_item_main.randomize() with {seq_item_main.array_rand[0:2] inside
{3'b000,3'b001,3'b110};} );
 seq_item_main.update_counter_allcases;
 finish_item(seq_item_main);
 end

 repeat(1000) begin
 start_item(seq_item_main);
 seq_item_main.reset.constraint_mode(1);
 seq_item_main.serial_comm_all_cases.constraint_mode(0);
```

```
 seq_item_main.mosi_in.constraint_mode(0);
 seq_item_main.serial_comm_read_data.constraint_mode(1);
 seq_item_main.trans_ram.constraint_mode(1);
 if(counter_read==0) begin
  seq_item_main.array_rand.rand_mode(1);
    $display("array_rand=%b , time =%t
,conter=%d",seq_item_main.array_rand,$time,counter_read);
 end
 else begin
  seq_item_main.array_rand.rand_mode(0);
 end
 assert (seq_item_main.randomize() with {seq_item_main.array_rand[0:2] inside
{3'b111};} );
 seq_item_main.update_counter_read;
 finish_item(seq_item_main);

 end
 endtask

 endclass
endpackage
```

9. Reset sequence file

```
package spi_slave_reset_sequence_package;
import uvm_pkg::*;
`include "uvm_macros.svh"
import spi_slave_seq_item_package::*;
class spi_slave_reset_sequence extends  uvm_sequence #(spi_slave_seq_item);
 `uvm_object_utils(spi_slave_reset_sequence);
 spi_slave_seq_item seq_item;
 function new(string name= "spi_slave_reset_sequence");
     super.new(name);
 endfunction
 task body ;
 seq_item= spi_slave_seq_item :: type_id :: create("seq_item");
 start_item(seq_item);
        seq_item.rst_n=1;
        seq_item.rx_valid=0;
        seq_item.tx_valid=0;
        seq_item.SS_n=1;
 finish_item(seq_item);
 endtask
endclass
endpackage
```

10. Sequencer file

```
package spi_slave_sequencer_package ;
import uvm_pkg::*;
`include "uvm_macros.svh"
import spi_slave_seq_item_package::*;
class spi_slave_sequencer extends uvm_sequencer #(spi_slave_seq_item);
  `uvm_component_utils(spi_slave_sequencer)
  function new(string name= "spi_slave_sequencer",uvm_component parent=null);
    super.new(name,parent);
  endfunction
endclass
endpackage
```

11. Configuration object file

```
package spi_slave_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
class spi_slave_config extends uvm_object;
    `uvm_object_utils(spi_slave_config) // factory store name of class
   virtual spi_slave_if ss_vif; //  create interface inside configration class
 function new(string name= "spi_slave_config");
     super.new(name);
 endfunction
endclass
endpackage
```

12. Shared pkg

```
package shared_pkg;
int counter_allcases =0  ;
int counter_read=0;
 logic SS_n_prev=1;
endpackage
```

13. Driver

```
 package spi_slave_driver_pkg;
 import uvm_pkg::*;
    `include "uvm_macros.svh"
    import spi_slave_seq_item_package::*;
    import shared_pkg ::*;
class spi_slave_driver extends uvm_driver #(spi_slave_seq_item);
 `uvm_component_utils(spi_slave_driver);
 virtual spi_slave_if ss_vif;
 spi_slave_seq_item seq_item;
 function new(string name ="spi_slave_driver",uvm_component parent =null);
    super.new(name,parent);
    endfunction
    task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever begin
      seq_item =  spi_slave_seq_item :: type_id :: create ( "seq_item");
      seq_item_port.get_next_item(seq_item);
      ss_vif.rst_n=seq_item.rst_n;
      ss_vif.SS_n=seq_item.SS_n;
      ss_vif.MOSI=seq_item.MOSI;
      ss_vif.tx_valid=seq_item.tx_valid;
      ss_vif.tx_data=seq_item.tx_data;
      @(negedge ss_vif.clk);
      seq_item_port.item_done();
      `uvm_info ("run_phase",seq_item.convert2string_stimulus(),UVM_HIGH)
    end
    endtask
endclass
 endpackage
```

14. Monitor

```
package spi_slave_monitor_pkg;
 import uvm_pkg::*;
    `include "uvm_macros.svh"
    import  spi_slave_seq_item_package::*;
    import shared_pkg ::*;
class spi_slave_monitor extends uvm_monitor;
 `uvm_component_utils(spi_slave_monitor);
 virtual spi_slave_if ss_vif;
spi_slave_seq_item seq_item;
 uvm_analysis_port #(spi_slave_seq_item) mon_ap;
  function new(string name ="spi_slave_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

    task run_phase (uvm_phase phase);
    super.run_phase(phase);
    forever begin
        seq_item=spi_slave_seq_item::type_id::create("seq_item");
        @(negedge ss_vif.clk);
        seq_item.rst_n=ss_vif.rst_n;
        seq_item.MOSI=ss_vif.MOSI;
        seq_item.SS_n=ss_vif.SS_n;
        seq_item.MISO=ss_vif.MISO;
        seq_item.MISO_ref=ss_vif.MISO_ref;
        seq_item.tx_valid=ss_vif.tx_valid;
        seq_item.tx_data=ss_vif.tx_data;
        seq_item.rx_valid=ss_vif.rx_valid;
        seq_item.rx_data=ss_vif.rx_data;
        seq_item.rx_valid_ref=ss_vif.rx_valid_ref;
         seq_item.rx_data_ref=ss_vif.rx_data_ref;

        mon_ap.write(seq_item);
        `uvm_info ("run_phase",seq_item.convert2string_stimulus(),UVM_HIGH)
    end
    endtask
endclass
endpackage
```

15. Agent

```
package spi_slave_agent_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import  spi_slave_seq_item_package::*;
  import spi_slave_config_pkg::*;
  import  spi_slave_driver_pkg::*;
  import spi_slave_monitor_pkg::*;
  import spi_slave_sequencer_package::*;

  class spi_slave_agent extends uvm_agent;
    `uvm_component_utils(spi_slave_agent)

    spi_slave_seq_item seq_item;
    spi_slave_config           s_cfg;
    spi_slave_driver       drv;
   spi_slave_sequencer sqr;
   spi_slave_monitor       mon;
    uvm_analysis_port #(spi_slave_seq_item) agt_ap;

    function new(string name = "spi_slave_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_slave_config)::get(this, "", "CFG", s_cfg)) begin
        `uvm_fatal("build_phase", "unable to get configuration object")
      end
      sqr = spi_slave_sequencer::type_id::create("sqr", this);
      drv = spi_slave_driver::type_id::create("drv", this);
      mon = spi_slave_monitor::type_id::create("mon", this);
      agt_ap = new("agt_tb",this);
    endfunction

    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      drv.ss_vif = s_cfg.ss_vif;
      mon.ss_vif = s_cfg.ss_vif;
      drv.seq_item_port.connect(sqr.seq_item_export);
     mon.mon_ap.connect(agt_ap);
    endfunction
  endclass
endpackage
```

16. Scoreboard

```
package spi_slave_scoreboard_pkg;
 import uvm_pkg::*;
    `include "uvm_macros.svh"
    import  spi_slave_seq_item_package::*;
    class spi_slave_scoreboard extends uvm_scoreboard ;
     `uvm_component_utils(spi_slave_scoreboard);
     uvm_analysis_export #(spi_slave_seq_item) sb_export;
     uvm_tlm_analysis_fifo  #(spi_slave_seq_item) sb_fifo;
    spi_slave_seq_item sq_item_sb;
     int err_count=0;
     int correct_count=0;
     function new(string name ="spi_slave_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(sq_item_sb);
  if (sq_item_sb.MISO != sq_item_sb.MISO_ref || sq_item_sb.rx_valid
!=sq_item_sb.rx_valid_ref || sq_item_sb.rx_data!=sq_item_sb.rx_data_ref) begin
    `uvm_error("run_phase",$sformatf("Comparison failed, Transaction received by
the DUT: %s While the reference out:0b%0b",
sq_item_sb.convert2string(),sq_item_sb.MISO_ref))

    err_count++;
  end
  else begin
    `uvm_info("run_phase",$sformatf("Correct ALU out: %s",
sq_item_sb.convert2string()), UVM_HIGH)
    correct_count++;
  end
```

```
  end
endtask

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

17. collector

```
package  spi_slave_collector_package ;
import uvm_pkg::*;
`include "uvm_macros.svh"
 import  spi_slave_seq_item_package::*;
 import shared_pkg::*;
class spi_slave_coverage extends uvm_component;
  `uvm_component_utils(spi_slave_coverage)

  uvm_analysis_export #(spi_slave_seq_item) cov_export;
  uvm_tlm_analysis_fifo #(spi_slave_seq_item) cov_fifo;
  spi_slave_seq_item seq_item_cov;
  covergroup covcode ;
  reciver_data: coverpoint seq_item_cov.rx_data[9:8];
  ss_n_allcases : coverpoint seq_item_cov.SS_n
  {
    bins trans_all_cases= (1 => 0[*13] =>1) ;

  }
  ss_n_read_data : coverpoint seq_item_cov.SS_n
  {
     bins trans_read= (1 => 0[*23] =>1) ;
  }
  ss_n : coverpoint seq_item_cov.SS_n
  {
    bins ss_n_val={0};
  }



  mosi: coverpoint seq_item_cov.MOSI
```

```
  {
    bins write_add = (0=>0=>0);
    bins write_data = (0=>0=>1);
    bins read_add = (1=>1=>0);
    bins read_data =(1=>1=>1);
    bins mosi_val_low={0};
     bins mosi_val_high={1};
  }
  cross_mosi_ss_n : cross mosi,ss_n
  {
     option.cross_auto_bin_max=0;
     bins ss_n_low_mosi_low = binsof(ss_n.ss_n_val) && binsof(mosi.mosi_val_low);
     bins ss_n_low_mosi_high = binsof(ss_n.ss_n_val)  &&
binsof(mosi.mosi_val_high);
  }

  endgroup

  function new(string name = "spi_slave_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(seq_item_cov);
    covcode.sample();
  end
  endtask

endclass
endpackage
```

18. Environment

```
package spi_slave_env_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import spi_slave_agent_pkg::*;
import spi_slave_scoreboard_pkg::*;
import spi_slave_collector_package::*;
class spi_slave_env extends uvm_env;

      `uvm_component_utils(spi_slave_env);
      spi_slave_scoreboard score;
       spi_slave_coverage cover_grp;
     spi_slave_agent agt;
      function new(string name ="spi_slave_env",uvm_component parent =null);
    super.new(name,parent);
    endfunction
   function void build_phase (uvm_phase phase);
   super.build_phase(phase);
   agt = spi_slave_agent :: type_id ::create("agt",this);
   score =   spi_slave_scoreboard :: type_id :: create("score",this);
   cover_grp = spi_slave_coverage  ::type_id ::create("cover_grp",this);

   endfunction
   function void connect_phase (uvm_phase phase);
   agt.agt_ap.connect(score.sb_export);
   agt.agt_ap.connect(cover_grp.cov_export);
   endfunction
endclass
endpackage
```

19. Test

```
package spi_slave_test_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import spi_slave_env_pkg::*;
 import spi_slave_reset_sequence_package::*;
 import spi_slave_main_sequence_package::*;
import spi_slave_config_pkg ::*;
class spi_slave_test extends uvm_test;
    `uvm_component_utils(spi_slave_test);
   spi_slave_env  env;
   spi_slave_config   s_cfg;
    spi_slave_reset_sequence rst_seq;
   spi_slave_main_sequence main_seq;
    function new(string name ="spi_slave_test",uvm_component parent =null);
    super.new(name,parent);
    endfunction
    function void build_phase (uvm_phase phase);
    super.build_phase(phase);
    env =spi_slave_env :: type_id :: create("env",this);
    s_cfg = spi_slave_config   :: type_id :: create ("s_cfg");
    rst_seq= spi_slave_reset_sequence :: type_id :: create ("rst_seq");
    main_seq=spi_slave_main_sequence :: type_id :: create("main_seq");

    if(!uvm_config_db #(virtual spi_slave_if) ::
get(this,"","SPI_IF",s_cfg.ss_vif))
    `uvm_fatal("build_phase","Test - unable to get the virtual interface");
    uvm_config_db #(spi_slave_config) :: set(this,"*","CFG",s_cfg);
    endfunction

    task run_phase(uvm_phase phase);
    super.run_phase(phase);
    phase.raise_objection(this);
     `uvm_info("run_phase","reset_asserted",UVM_LOW)
     rst_seq.start(env.agt.sqr);
     `uvm_info("run_phase","reset_deasserted",UVM_LOW)

     `uvm_info ("run_phase","stimulus generation begin",UVM_LOW)
      main_seq.start(env.agt.sqr);
       `uvm_info ("run_phase","stimulus generation done",UVM_LOW)
    phase.drop_objection(this);
    endtask
endclass: spi_slave_test
endpackage
```

20. Src_file

```
spi_slave_if.sv
spi_slave_config.sv
SPI_slave.sv
SPi_Slave_interface.v
shared_pkg.sv
spi_slave_sequence_item.sv
spi_slave_reset_sequence.sv
spi_slave_main_sequence.sv
spi_slave_sequencer.sv
spi_slave_driver.sv
spi_slave_monitor.sv
spi_slave_agent.sv
spi_slave_scoreboard.sv
spi_slave_collector.sv
spi_slave_env.sv
spi_slave_test.sv
top.sv
```

21. Do file

```
vlib work
vlog -f src_files.list +define+SIM +cover -covercells
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
add wave -r /top/ss_if/*
run -all
coverage save spi.ucdb -onexit

```

22. Functional coverage

```
Covergroup Coverage:
    Covergroups                     1        na        na   100.00%
        Coverpoints/Crosses         6        na        na        na
            Covergroup Bins        15        15         0   100.00%
--------------------------------------------------------------------------------------------------------------
Covergroup                                          Metric       Goal       Bins    Status

--------------------------------------------------------------------------------------------------------------
TYPE /spi_slave_collector_package/spi_slave_coverage/covcode
                                                   100.00%        100          -    Covered
    covered/total bins:                                 15         15          -
    missing/total bins:                                  0         15          -
    % Hit:                                         100.00%        100          -
    Coverpoint reciver_data                        100.00%        100          -    Covered
        covered/total bins:                              4          4          -
        missing/total bins:                              0          4          -
        % Hit:                                     100.00%        100          -
    Coverpoint ss_n_allcases                       100.00%        100          -    Covered
        covered/total bins:                              1          1          -
        missing/total bins:                              0          1          -
        % Hit:                                     100.00%        100          -
    Coverpoint ss_n_read_data                      100.00%        100          -    Covered
        covered/total bins:                              1          1          -
        missing/total bins:                              0          1          -
        % Hit:                                     100.00%        100          -
    Coverpoint ss_n                                100.00%        100          -    Covered
        covered/total bins:                              1          1          -
        missing/total bins:                              0          1          -
        % Hit:                                     100.00%        100          -
    Coverpoint mosi                                100.00%        100          -    Covered
        covered/total bins:                              6          6          -
```

23. Code coverage

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         74        74         0   100.00%

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

Toggle Coverage for instance /top/ss_if --

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

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        40        40         0   100.00%

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

Branch Coverage for instance /top/DUT

    Line         Item                     Count     Source
    ----         ----                     -----     ------
  File SPI_slave.sv
------------------------------------IF Branch------------------------------------
    14                                      547     Count coming in to IF
    14              1                        36         if (~ss_if.rst_n) begin

    17              1                       511         else begin

Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------CASE Branch-----------------------------------
    23                                     1016     Count coming in to CASE
    24              1                       258             IDLE : begin

    30              1                       193             CHK_CMD : begin

    44              1                       317             WRITE : begin
```

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      39        39         0   100.00%

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

Statement Coverage for instance /top/DUT --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File SPI_slave.sv
    1                                                module SLAVE (spi_slave_if.DUT ss_if);

    2

    3                                                localparam IDLE      = 3'b000;

    4                                                localparam WRITE     = 3'b001;

    5                                                localparam CHK_CMD   = 3'b010;

    6                                                localparam READ_ADD  = 3'b011;

    7                                                localparam READ_DATA = 3'b100;

    8
```

24. Assertion coverage

```
Directive Coverage:
    Directives                       9        9        0   100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                     Design Design   Lang File(Line)      Hits Status
                                         Unit   UnitType
--------------------------------------------------------------------------------------------
/top/DUT/cover__chck_state_read_datato_idle
                                         SLAVE  Verilog  SVA  SPI_slave.sv(195)
                                                                                4 Covered
/top/DUT/cover__chck_state_read_add_to_idle
                                         SLAVE  Verilog  SVA  SPI_slave.sv(194)
                                                                               10 Covered
/top/DUT/cover__chck_state_write_to_idle SLAVE  Verilog  SVA  SPI_slave.sv(193)
                                                                               17 Covered
/top/DUT/cover__chck_state_read_data     SLAVE  Verilog  SVA  SPI_slave.sv(192)
                                                                               15 Covered
/top/DUT/cover__chck_state_read_add      SLAVE  Verilog  SVA  SPI_slave.sv(191)
                                                                               43 Covered
/top/DUT/cover__chck_state_write         SLAVE  Verilog  SVA  SPI_slave.sv(190)
                                                                               74 Covered
/top/DUT/cover__chck_state_idle          SLAVE  Verilog  SVA  SPI_slave.sv(189)
                                                                              138 Covered
/top/DUT/cover__chck_rx_valid            SLAVE  Verilog  SVA  SPI_slave.sv(188)
                                                                               28 Covered
/top/DUT/cover__chck_reset               SLAVE  Verilog  SVA  SPI_slave.sv(187)
                                                                               36 Covered
```

25. Questasim snippets

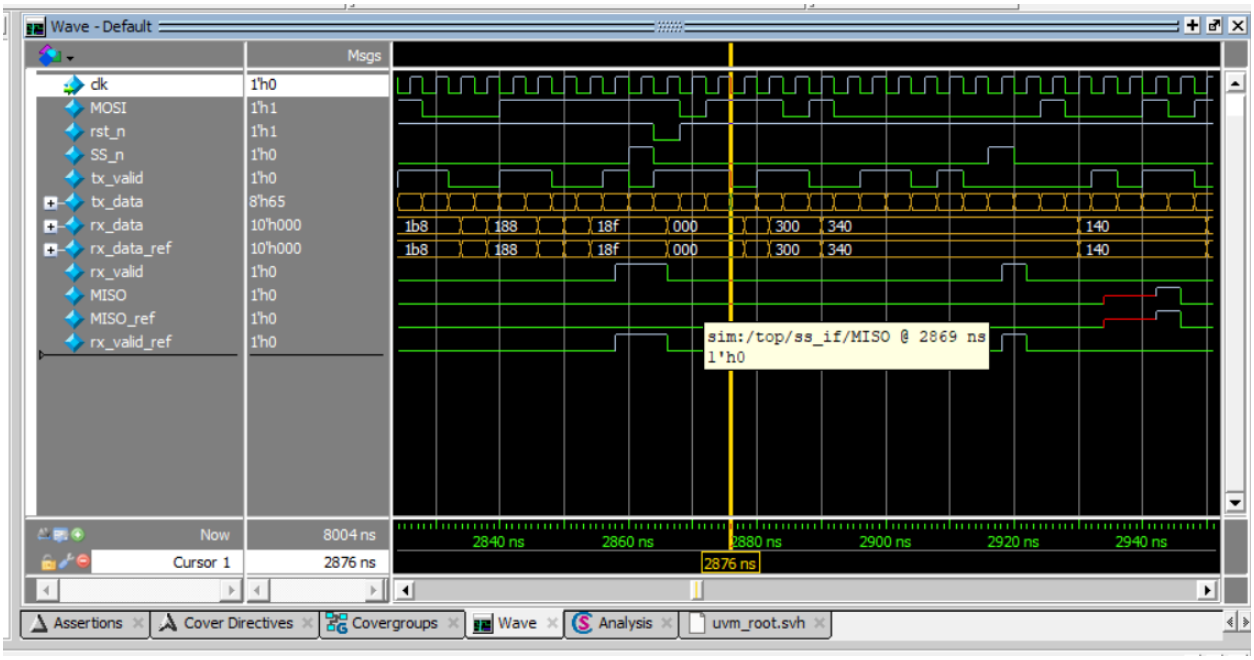


Figure 9 when ss_n is deasserted mosi read_add sequence (110)

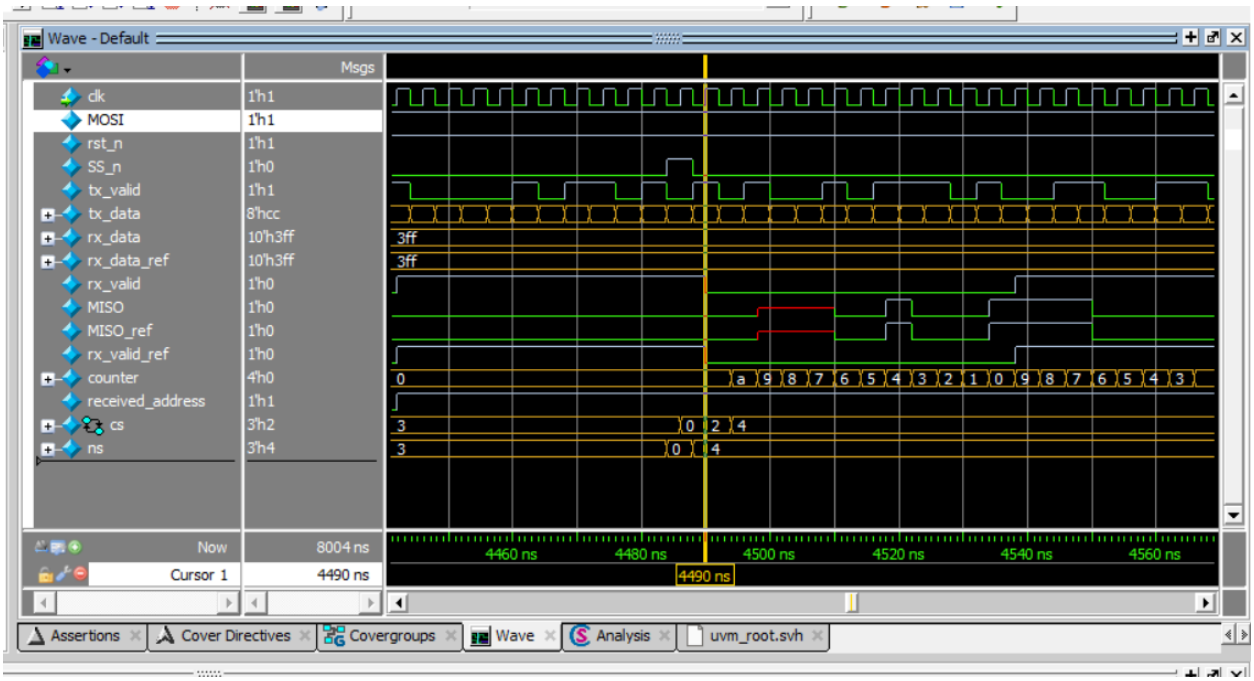


Figure 10 when ss_n is deasserted mosi read_data sequence (111)

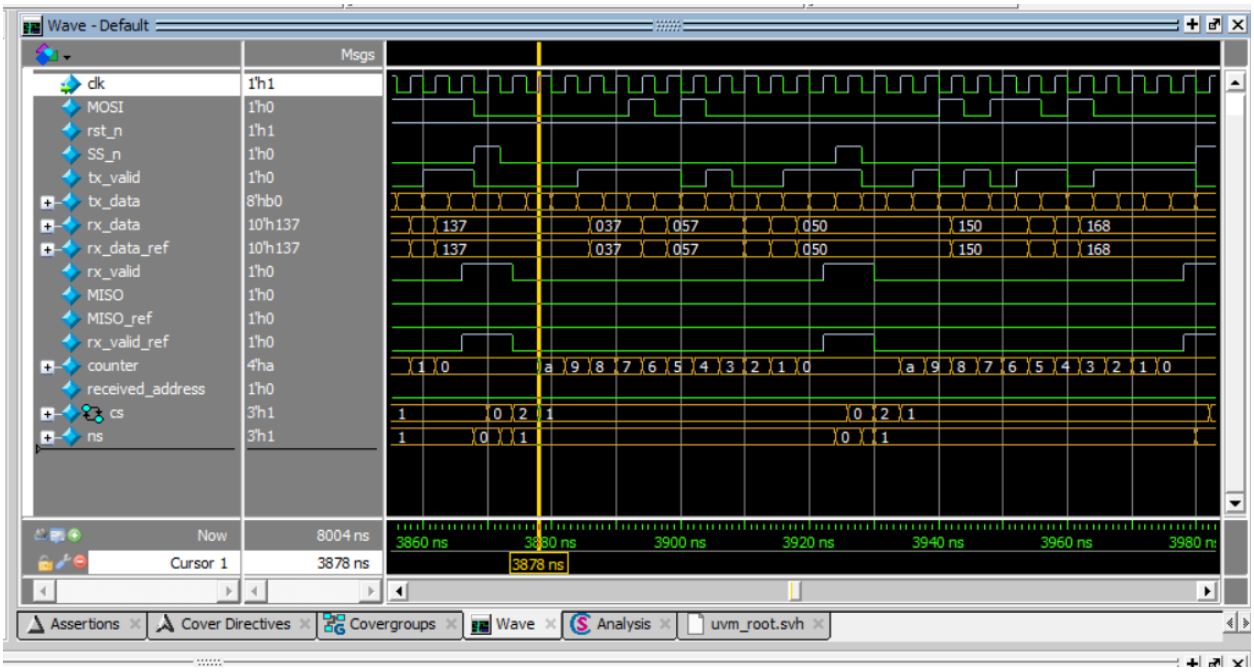


Figure 11 when ss_n is deasserted mosi write_add sequence (000)

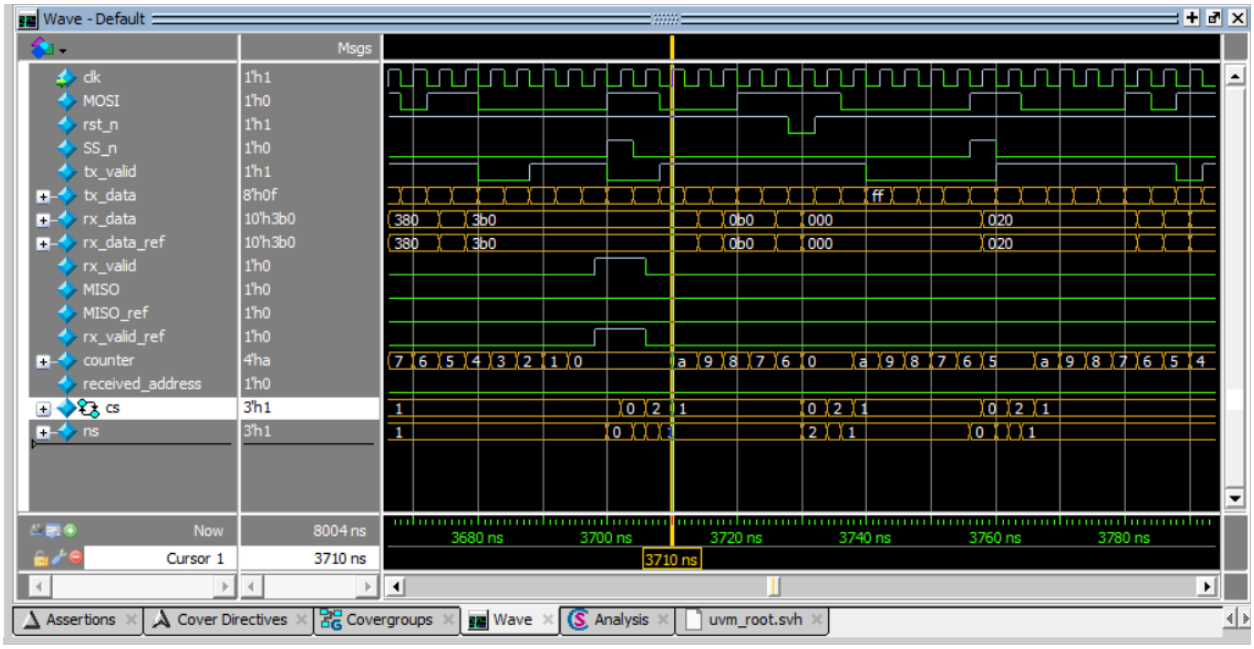


Figure 12 when ss_n is deasserted mosi write_data sequence (001)

26. Table of assertions with features

Feature	Assertion
Whenever the rst is asserted, MISO , rx_valid , rx_data is low	`@(posedge ss_if.clk) (~ss_if.rst_n) |=>(ss_if.MISO ==0 && ss_if.rx_valid==0 && ss_if.rx_data==0);`
Whenever the current state is idle and SS_n is low so current state will be CHCK_CMD in next cycle and reset is high	`@(posedge ss_if.clk) disable iff(~ss_if.rst_n) (cs==IDLE && !ss_if.SS_n) |=>(cs==CHK_CMD);`
Whenever the current state CHCK_CMD and SS_n is low and mosi is low so current state will be WRITE in next cycle and reset is high	`@(posedge ss_if.clk) disable iff(~ss_if.rst_n) (cs==CHK_CMD && !ss_if.SS_n && !ss_if.MOSI) |=> (cs==WRITE);`
Whenever the current state CHCK_CMD and SS_n is low and mosi is high and received_add is low so current state will be WRITE in next cycle and reset is high	`@(posedge ss_if.clk) disable iff(~ss_if.rst_n) (cs==CHK_CMD && !ss_if.SS_n && ss_if.MOSI && !received_address) |=>(cs==READ_ADD);`
Whenever the current state CHCK_CMD and SS_n is low and mosi is high and received_add is high so current state will be WRITE in next cycle and reset is high	`@(posedge ss_if.clk) disable iff(~ss_if.rst_n) (cs==CHK_CMD && !ss_if.SS_n && ss_if.MOSI && received_address) |=>(cs==READ_DATA);`
Whenever the rst is asserted and current state is WRITE , the current state will be IDEAL in next clock cycle	`@(posedge ss_if.clk) (cs==WRITE &&(~ss_if.rst_n)) |=> (cs==IDLE);`
Whenever the rst is asserted and current state is READ_ADD , the current state will be IDEAL in next clock cycle	`@(posedge ss_if.clk) (cs==READ_ADD && (~ss_if.rst_n)) |=> (cs==IDLE);`
Whenever the rst is asserted and current state is READ_DATA , the current state will be IDEAL in next clock cycle	`@(posedge ss_if.clk) (cs==READ_DATA && (~ss_if.rst_n)) |=> (cs==IDLE);`
Whenever SS_n is deasserted and wite_add or write_data or read_add or read_data sequence done then rx_valid will assert and SS_n will assert to end communication	`@(posedge ss_if.clk) disable iff(~ss_if.rst_n) (write_add_seq or write_data_seq or read_add_seq or read_data_seq) |=> ##9 ($rose(ss_if.rx_valid) && $rose (ss_if.SS_n)[->1]);`

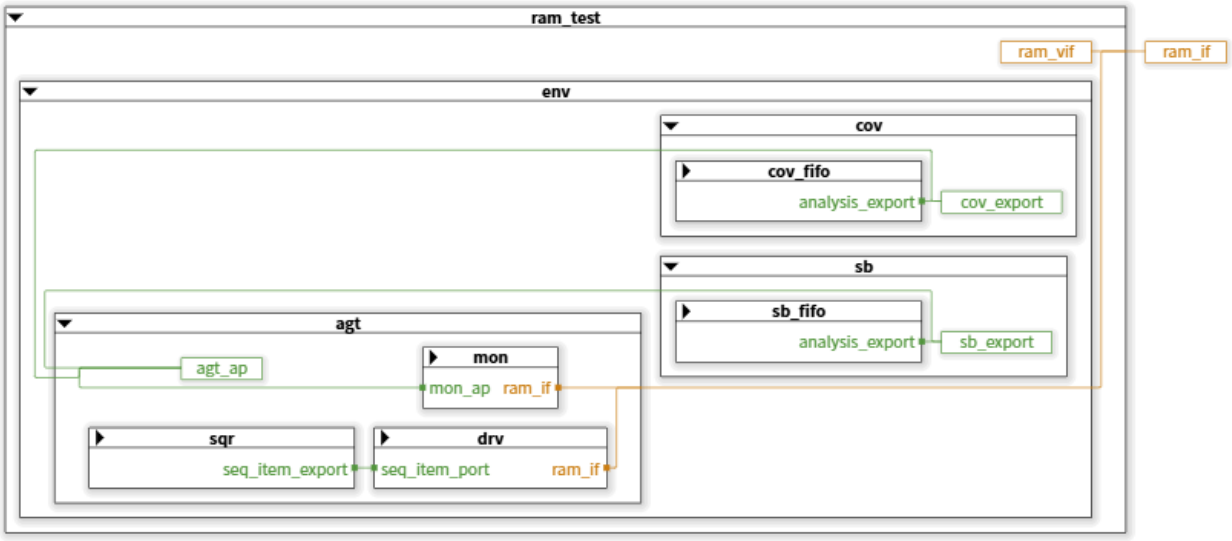
Sequence which I mentioned in last row :

```
sequence write_add_seq;
  (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 0)[*3];
endsequence
sequence write_data_seq;
  (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 0)[*2] ##1(ss_if.
endsequence
sequence read_add_seq;
  (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 1)[*2] ##1(ss_if.
endsequence

sequence read_data_seq;
  (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 1)[*3];
endsequence
```

2-RAM:

Feature	Assertion
When rst_n is asserted, dout & tx_valid are low	(@(posedge clk) (!rst_n) |=> (dout == 0) && (tx_valid == 0))
When din[9:8] = 11 and rx_valid is asserted, tx_valid should be high and should eventually fall.	@(posedge clk) disable iff(!rst_n) ((din[8] && din[9]) && rx_valid) |=> (tx_valid == 1) |=> $fell(tx_valid)[->1])
When din[9:8] = 10, tx_valid should be low.	(@(posedge clk) disable iff(!rst_n) (!(din[8] && din[9])) |=> (tx_valid == 0))
When din[9:8] = 00 which is a write address instruction, there should eventually be a write data instruction din[9:8] = 01	(@(posedge clk) disable iff(!rst_n) ((!din[8] && !din[9])) |=> (din[8] && !din[9])[->1]);
When din[9:8] = 10 which is a read address instruction, there should eventually be a read data instruction din[9:8] = 11	(@(posedge clk) disable iff(!rst_n) ((!din[8] && din[9])) |=> (din[8] && din[9])[->1]);

Ram Uvm architecture :



Top:

```
import uvm_pkg::*;
import ram_test_pkg::*;
import shared_pkg::*;
`include "uvm_macros.svh"
module ram_top();
    bit clk;
    initial begin
        forever begin
            #1 clk = ~clk;
        end
    end

    ram_if RAM_IF(clk);
    RAM DUT(RAM_IF.din,RAM_IF.clk,RAM_IF.rst_n,RAM_IF.rx_valid,RAM_IF.dout,RAM_IF.tx_valid);
    ram_ref ref_model(RAM_IF.clk, RAM_IF.rst_n, RAM_IF.din, RAM_IF.rx_valid, RAM_IF.dout_ref, RAM_IF.tx_valid_ref);
    bind RAM ram_sva sva_inst(RAM_IF.din,RAM_IF.clk,RAM_IF.rst_n,RAM_IF.rx_valid,RAM_IF.dout,RAM_IF.tx_valid);

    initial begin
        uvm_config_db #(virtual ram_if)::set(null, "uvm_test_top", "RAM_IF", RAM_IF);
        run_test("ram_test");
    end

endmodule
```

Test:

```
package ram_test_pkg;
import uvm_pkg::*;
import ram_env_pkg::*;
import ram_cfg_obj_pkg::*;
import ram_seq_item_pkg::*;
import ram_reset_seq_pkg::*;
import ram_wr_only_seq_pkg::*;
import ram_rd_only_seq_pkg::*;
import ram_wr_rd_seq_pkg::*;
`include "uvm_macros.svh"

class ram_test extends uvm_test;
    `uvm_component_utils(ram_test)
    ram_cfg_obj ram_cfg;
    virtual ram_if ram_vif;
    ram_env env;
    ram_reset_seq rst_seq;
    ram_wr_only_seq wr_only_seq;
    ram_rd_only_seq rd_only_seq;
    ram_wr_rd_seq wr_rd_seq;


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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        ram_cfg = ram_cfg_obj::type_id::create("cfg_object");
        env = ram_env::type_id::create("env", this);
        rst_seq = ram_reset_seq::type_id::create("rst_seq");
        wr_only_seq = ram_wr_only_seq::type_id::create("wr_only_seq");
        rd_only_seq = ram_rd_only_seq::type_id::create("rd_only_seq");
        wr_rd_seq = ram_wr_rd_seq::type_id::create("wr_rd_seq");

        if(!uvm_config_db #(virtual ram_if)::get(this, "", "RAM_IF", ram_cfg.ram_vif)) begin
            `uvm_fatal("build_phase", "unable to get virtual interface of ram")
        end

        uvm_config_db #(ram_cfg_obj)::set(this, "*", "RAM_CFG", ram_cfg);
    endfunction
```

```
    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);
        `uvm_info("run_phase", "rst signal asserted", UVM_LOW)
        rst_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "rst signal deasserted", UVM_LOW)
        phase.drop_objection(this);

        phase.raise_objection(this);
        `uvm_info("run_phase", "write only sequence started", UVM_LOW)
        wr_only_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "write only sequence ended", UVM_LOW)
        phase.drop_objection(this);

        phase.raise_objection(this);
        `uvm_info("run_phase", "read only sequence started", UVM_LOW)
        rd_only_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "read only sequence ended", UVM_LOW)
        phase.drop_objection(this);

        phase.raise_objection(this);
        `uvm_info("run_phase", "write & read sequence started", UVM_LOW)
        wr_rd_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "write & read sequence ended", UVM_LOW)
        phase.drop_objection(this);

    endtask
endclass
endpackage
```

Env:

```
ram_env.sv > ...
1   package ram_env_pkg;
2   import uvm_pkg::*;
3   import ram_scoreboard_pkg::*;
4   import ram_coverage_pkg::*;
5   import ram_agent_pkg::*;
6   `include "uvm_macros.svh"
7
8   class ram_env extends uvm_env;
9       `uvm_component_utils(ram_env)
10      ram_scoreboard sb;
11      ram_coverage cov;
12      ram_agent agt;
13
14      function new(string name = "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          sb = ram_scoreboard::type_id::create("sb", this);
21          cov = ram_coverage::type_id::create("cov", this);
22          agt = ram_agent::type_id::create("agt", this);
23      endfunction
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
```

Agent:

```
package ram_agent_pkg;
import uvm_pkg::*;
import ram_seq_item_pkg::*;
import ram_sqr_pkg::*;
import ram_driver_pkg::*;
import ram_monitor_pkg::*;
import ram_cfg_obj_pkg::*;
`include "uvm_macros.svh"

class ram_agent extends uvm_agent;
    `uvm_component_utils(ram_agent)
    ram_cfg_obj cfg;
    ram_driver drv;
    ram_sqr sqr;
    ram_monitor mon;
    uvm_analysis_port #(ram_seq_item) agt_ap;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        cfg = ram_cfg_obj::type_id::create("cfg");
        sqr = ram_sqr::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);

        if(!uvm_config_db #(ram_cfg_obj)::get(this, "", "RAM_CFG", cfg)) begin
            `uvm_fatal("build_phase", "unable to get cfg object")
        end

    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        drv.ram_vif = cfg.ram_vif;
        mon.ram_vif = cfg.ram_vif;
        drv.seq_item_port.connect(sqr.seq_item_export);
        mon.mon_ap.connect(agt_ap);
    endfunction
endclass
endpackage
```

Config Object, Sequence Item and Sequencer:

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

class ram_cfg_obj extends uvm_object;
    `uvm_object_utils(ram_cfg_obj)
    virtual ram_if ram_vif;

    function new(string name = "cfg_obj");
        super.new(name);
    endfunction
endclass
endpackage
```

ram_seq_item.sv > {} ram_seq_item_pkg > ram_seq_item

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

class ram_seq_item extends uvm_sequence_item;
    `uvm_object_utils(ram_seq_item)

    rand bit rst_n, rx_valid;
         bit tx_valid, tx_valid_ref;
    rand bit [9:0] din;
         bit [7:0] dout, dout_ref;
         op_e old_operation;

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

    constraint rst_rx_constraint
    {
        // RAM_RESET
        rst_n dist {0:= 2, 1:= 98};
        // RAM_RX_VALID
        rx_valid dist {0:= 2, 1:= 98};
    }

    constraint wr_only_constraint
    {
        // Always constrain to write operations (address or data)
        // RAM_WR
        din[9:8] inside {WR_ADDR, WR_DATA};
    }

```

```
    constraint rd_only_constraint
    {
        // RAM_RD
        if(old_operation == RD_ADDR)
        {
            din[9:8] == RD_DATA;
        }
        else if (old_operation == RD_DATA)
        {
            din[9:8] == RD_ADDR;
        }
        else
        {
            din[9:8] inside {RD_ADDR, RD_DATA};
        }

    }

    constraint rd_wr_constraint
    {
        if(old_operation == WR_ADDR) // wr_addr
        {
            din[9:8] inside {WR_ADDR, WR_DATA};
        }
        else if (old_operation == WR_DATA) // wr_data
        {
            din[9:8] dist {RD_ADDR:= 60, WR_ADDR:= 40}; // read_address = 60%, write_address = 40%
        }
        else if (old_operation == RD_ADDR) // rd_addr
        {
            din[9:8] inside {RD_ADDR, RD_DATA};
        }
        else // rd_data
        {
            din[9:8] dist {WR_ADDR:= 60, RD_ADDR:= 40};
        }
    }
```

```
    function void post_randomize();
        old_operation = op_e'(din[9:8]);
    endfunction

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

    function string convert2string_stimulus();
        return $sformatf("rst_n = %0b rx_valid = %0b tx_valid = %0b din = %b dout = %b", rst_n, rx_valid, tx_valid, din, dout);
    endfunction
endclass
endpackage
```

```
ram_sqr.sv > ...
1   package ram_sqr_pkg;
2   import uvm_pkg::*;
3   import ram_seq_item_pkg::*;
4   `include "uvm_macros.svh"
5
6   class ram_sqr extends uvm_sequencer #(ram_seq_item);
7       `uvm_component_utils(ram_sqr)
8
9       function new(string name = "sqr", uvm_component parent = null);
10          super.new(name, parent);
11      endfunction
12
13  endclass
14  endpackage
```

Reset, Write only sequence, Read only sequence and W&R sequence:

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

class ram_reset_seq extends uvm_sequence #(ram_seq_item);
    `uvm_object_utils(ram_reset_seq)
    ram_seq_item item;

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

    task body();
        // Assert reset
        item = ram_seq_item::type_id::create("item");
        start_item(item);
        item.rst_n = 0;
        item.rx_valid = 0;
        item.tx_valid = 0;
        item.din = 0;
        finish_item(item);

        // Hold reset for 5 cycles
        repeat(5) begin
            start_item(item);
            item.rst_n = 0;
            item.rx_valid = 0;
            item.tx_valid = 0;
            item.din = 0;
            finish_item(item);
        end

        // Deassert reset
        start_item(item);
        item.rst_n = 1;
        item.rx_valid = 0;
        item.tx_valid = 0;
        item.din = 0;
        finish_item(item);
    endtask

endclass
endpackage
```

ram_wr_rd_seq.sv > ...

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

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

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

Monitor and Driver:

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)
    ram_seq_item item;
    virtual ram_if ram_vif;
    uvm_analysis_port #(ram_seq_item) mon_ap;

    function new(string name = "mtr", 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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            item = ram_seq_item::type_id::create("item");
            @(negedge ram_vif.clk);
            item.rst_n = ram_vif.rst_n;
            item.rx_valid = ram_vif.rx_valid ;
            item.din = ram_vif.din ;
            item.tx_valid = ram_vif.tx_valid;
            item.tx_valid_ref = ram_vif.tx_valid_ref;
            item.dout = ram_vif.dout;
            item.dout_ref = ram_vif.dout_ref;
            mon_ap.write(item);
            `uvm_info("run_phase", item.convert2string_stimulus(), UVM_HIGH);
        end
    endtask
endclass
endpackage
```

```systemverilog
package ram_driver_pkg;
import uvm_pkg::*;
import ram_seq_item_pkg::*;
`include "uvm_macros.svh"

class ram_driver extends uvm_driver #(ram_seq_item);
    `uvm_component_utils(ram_driver)
    ram_seq_item item;
    virtual ram_if ram_vif;

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            item = ram_seq_item::type_id::create("item");
            seq_item_port.get_next_item(item);
            ram_vif.rst_n = item.rst_n;
            ram_vif.rx_valid = item.rx_valid;
            ram_vif.din = item.din;
            @(negedge ram_vif.clk);
            seq_item_port.item_done();
            `uvm_info("run_phase", item.convert2string_stimulus(), UVM_HIGH);
        end
    endtask
endclass
endpackage
```

Interface:

```
interface ram_if(clk);
    parameter ADDR_SIZE = 8, MEM_DEPTH = 256;
    input bit clk;
    logic rst_n, rx_valid, tx_valid, tx_valid_ref;
    logic [ADDR_SIZE+1:0] din; // default word size + extra 2 bits
    logic [ADDR_SIZE-1:0] dout, dout_ref;

endinterface
```

Coverage:

```
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 item;

    covergroup ram_cvr_gp;
    transaction_ordering_cp: coverpoint item.din[9:8]
    {
        bins all_values = {[0:3]};
        bins wr_data_after_wr_address = (0 => 1);
        bins rd_data_after_rd_address = (2 => 3);
        bins full_transition = (0 => 1 => 2 => 3);
    }

    rx_valid_cp: coverpoint item.rx_valid
    {
        bins rx_high = {1};
    }
    tx_valid_cp: coverpoint item.tx_valid
    {
        bins tx_high = {1};
    }

    cross_op_rx_cp: cross transaction_ordering_cp, rx_valid_cp;
    cross_op_tx_cp: cross transaction_ordering_cp, tx_valid_cp
    {
        // Cross bin for read data (3) when tx_valid is high (1)
        bins rd_data_tx_high = binsof(transaction_ordering_cp.all_values) intersect {3} &&
                               binsof(tx_valid_cp) intersect {1};
        option.cross_auto_bin_max = 0;
    }

    endgroup

```

```
41      function new(string name = "cov", uvm_component parent = null);
42          super.new(name, parent);
43          ram_cvr_gp = new();
44      endfunction
45
46      function void build_phase(uvm_phase phase);
47          super.build_phase(phase);
48          cov_export = new("cov_export", this);
49          cov_fifo = new("cov_fifo", this);
50      endfunction
51
52      function void connect_phase(uvm_phase phase);
53          super.connect_phase(phase);
54          cov_export.connect(cov_fifo.analysis_export);
55      endfunction
56
57      task run_phase(uvm_phase phase);
58          super.run_phase(phase);
59          forever begin
60              cov_fifo.get(item);
61              ram_cvr_gp.sample();
62          end
63      endtask
64  endclass
65  endpackage
```

Scoreboard:

```
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)
    ram_seq_item item;
    uvm_analysis_export #(ram_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(ram_seq_item) sb_fifo;
    int error_count = 0, correct_count = 0;

    function new(string name = "sb", 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(item);
            if(item.dout != item.dout_ref || item.tx_valid != item.tx_valid_ref) begin
                $display("Error, transation sent by dut is %s while dout_ref = %b and tx_ref = %b", item.convert2string_stimulus(), item.dout_ref, item.tx_valid_ref);
                error_count++;
            end
            else begin
                correct_count++;
            end
        end
    endtask
endclass
endpackage
```

Reference Model:

```
module ram_ref(clk, rst_n, din, rx_valid, dout, tx_valid);
input clk, rst_n, rx_valid;
input [9:0] din;
output reg tx_valid;
output reg [7:0] dout;

parameter MEM_DEPTH = 256, ADDR_SIZE = 8;

reg [ADDR_SIZE-1 :0] mem [MEM_DEPTH-1 :0];

reg [ADDR_SIZE - 1 : 0] wr_address, rd_address;
always @(posedge clk) begin
    if(!rst_n) begin
      dout <= 0;
      wr_address <= 0;
      rd_address <= 0;
      tx_valid <= 0;
    end
    else begin
        if(rx_valid) begin
            case ({din[9:8]})
                2'b00: begin
                  wr_address <= din[7:0];
                end
                2'b01: begin
                  mem[wr_address] <= din[7:0];
                end
                2'b10: begin
                  rd_address <= din[7:0];
                end
                2'b11: begin
                  dout <= mem[rd_address];
                end
            endcase
        end
        tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 : 1'b0;
    end
end
endmodule
```

Design:

```
1   module RAM (din,clk,rst_n,rx_valid,dout,tx_valid);
2
3   input      [9:0] din;
4   input            clk, rst_n, rx_valid;
5
6   output reg [7:0] dout;
7   output reg       tx_valid;
8
9   reg [7:0] MEM [255:0];
10
11  reg [7:0] Rd_Addr, Wr_Addr;
12
13  always @(posedge clk) begin
14      if (!rst_n) begin
15          dout <= 0;
16          tx_valid <= 0;
17          Rd_Addr <= 0;
18          Wr_Addr <= 0;
19      end
20      else begin // added begin_end to else branch
21          if (rx_valid) begin
22              case (din[9:8])
23                  2'b00 : Wr_Addr <= din[7:0];
24                  2'b01 : MEM[Wr_Addr] <= din[7:0];
25                  2'b10 : Rd_Addr <= din[7:0];
26                  // 2'b11 : dout <= MEM[Wr_Addr]; bug
27                  2'b11 : dout <= MEM[Rd_Addr];
28                  default : dout <= 0;
29              endcase
30          end
31          tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 : 1'b0;
32      end
33  end
34
35  endmodule
```

Bugs Found:

1- In READ_DATA case, output would read from Wr_addr.
2- Missing begin end in else branch

Shared_Pkg:

```
package shared_pkg;
    typedef enum bit [1:0] {WR_ADDR, WR_DATA, RD_ADDR, RD_DATA } op_e;
endpackage

```

Do File:

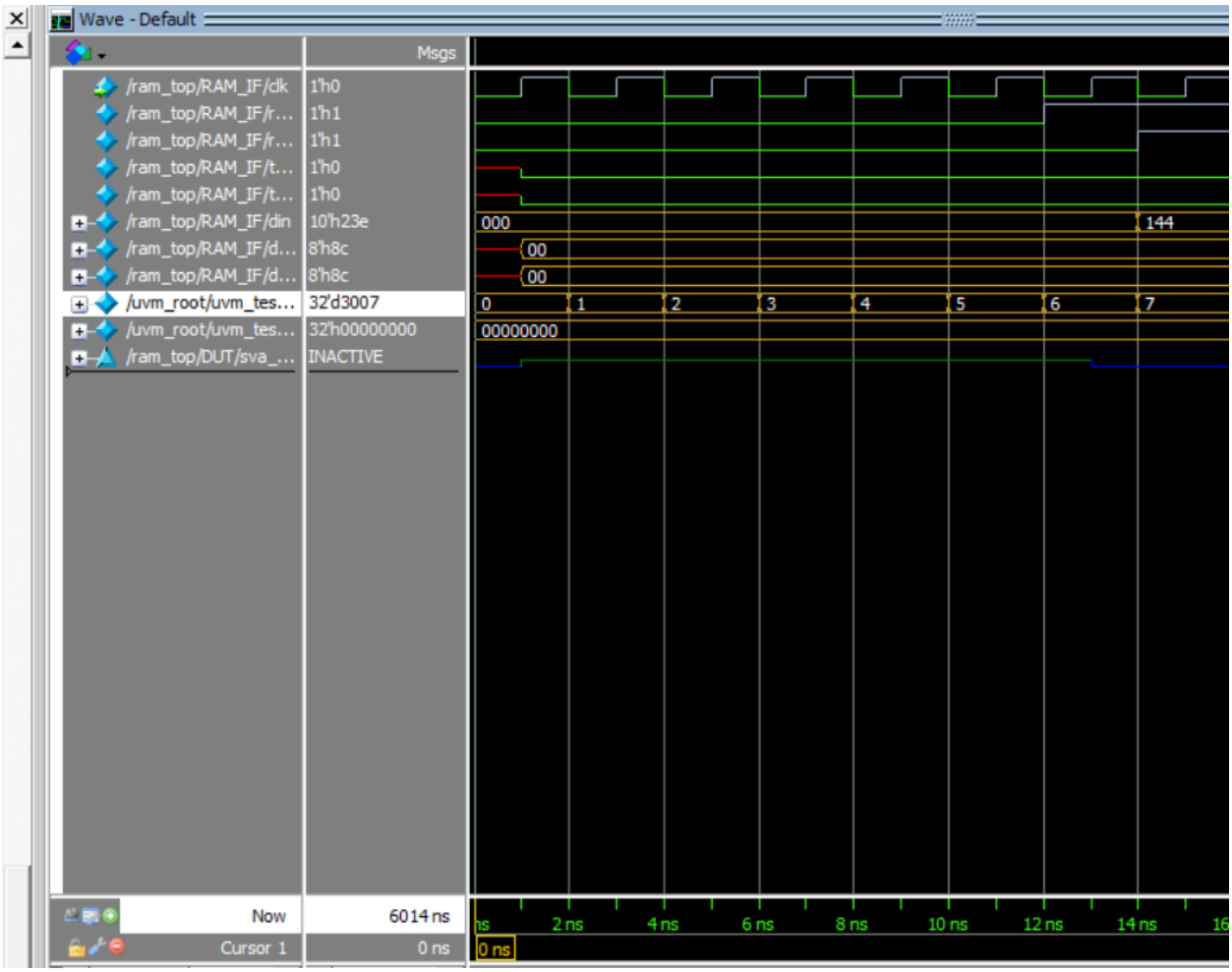
```
vlib work
vlog -f src_files.list +cover
vsim -voptargs=+acc work.ram_top -classdebug -uvmcontrol=all -cover
run 0
add wave /ram_top/RAM_IF/*
add wave -position insertpoint  \
sim:/uvm_root/uvm_test_top/env/sb/correct_count \
sim:/uvm_root/uvm_test_top/env/sb/error_count
add wave /ram_top/DUT/sva_inst/rst_n_assertion
coverage save ram_top.ucdb -onexit
run -all
```

Assertions:

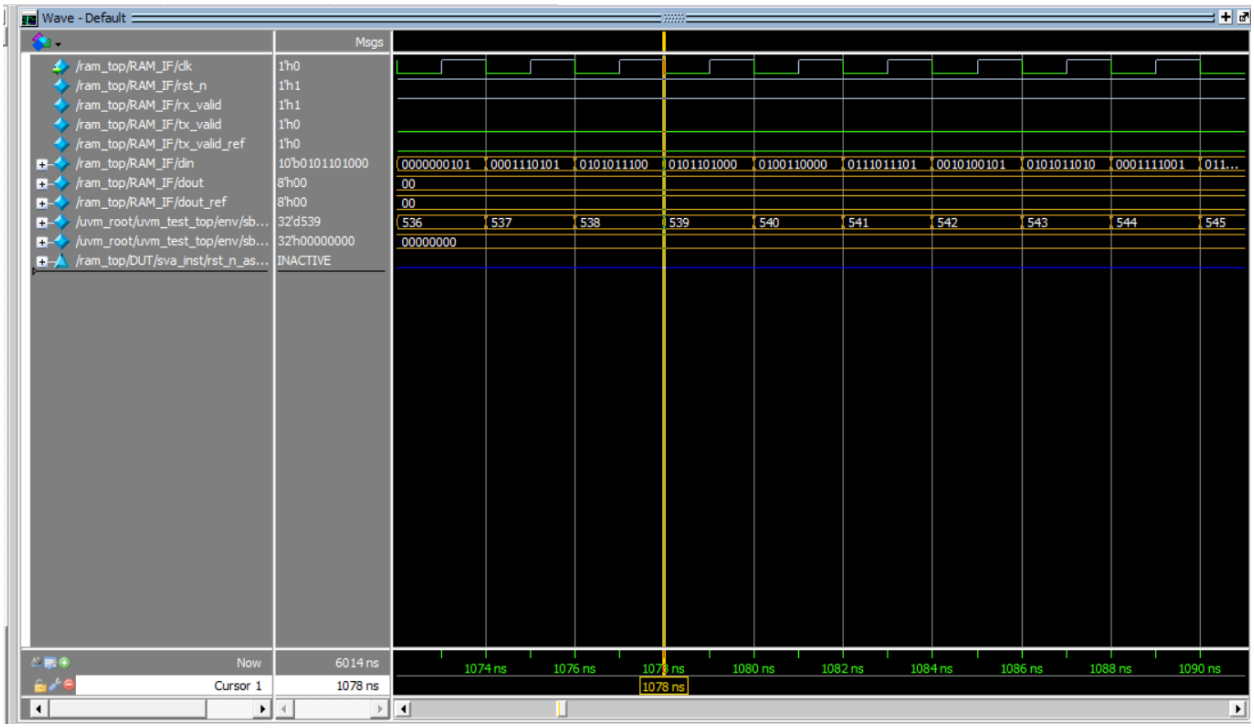
```
module ram_sva(input [9:0] din, input clk,rst_n,rx_valid, input [7:0] dout, input tx_valid);
    rst_n_assertion: assert property (@(posedge clk) (!rst_n) |=> (dout == 0) && (tx_valid == 0));
    tx_valid_0_assertion: assert property (@(posedge clk) disable iff(!rst_n) (!(din[8] && din[9])) |=> (tx_valid == 0));
    tx_valid_1_assertion: assert property (@(posedge clk) disable iff(!rst_n) ((din[8] && din[9]) && rx_valid) |=> (tx_valid == 1) |=> $fell(tx_valid)[->1]);
    wr_addr_then_wr_data_assertion: assert property (@(posedge clk) disable iff(!rst_n) ((!din[8] && !din[9])) |=> (din[8] && !din[9])[->1]);
    rd_addr_then_rd_data_assertion: assert property (@(posedge clk) disable iff(!rst_n) ((!din[8] && din[9])) |=> (din[8] && din[9])[->1]);

endmodule
```

Waveforms:

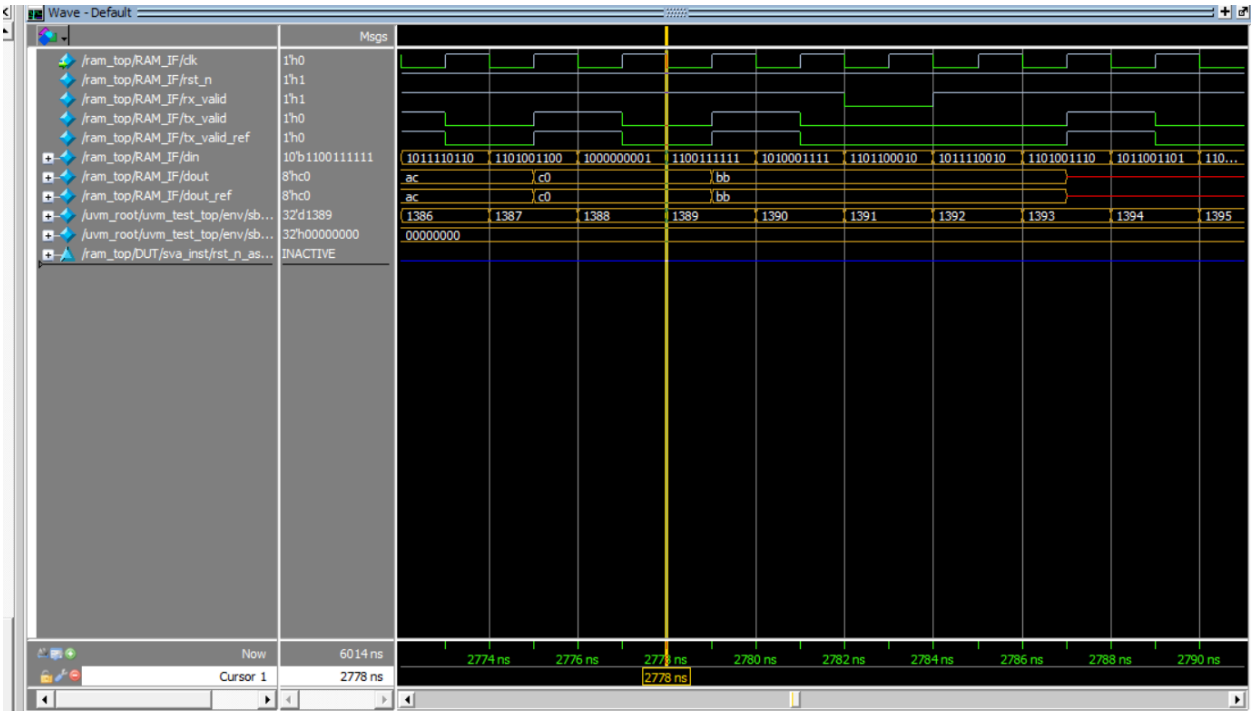
Reset Sequence:



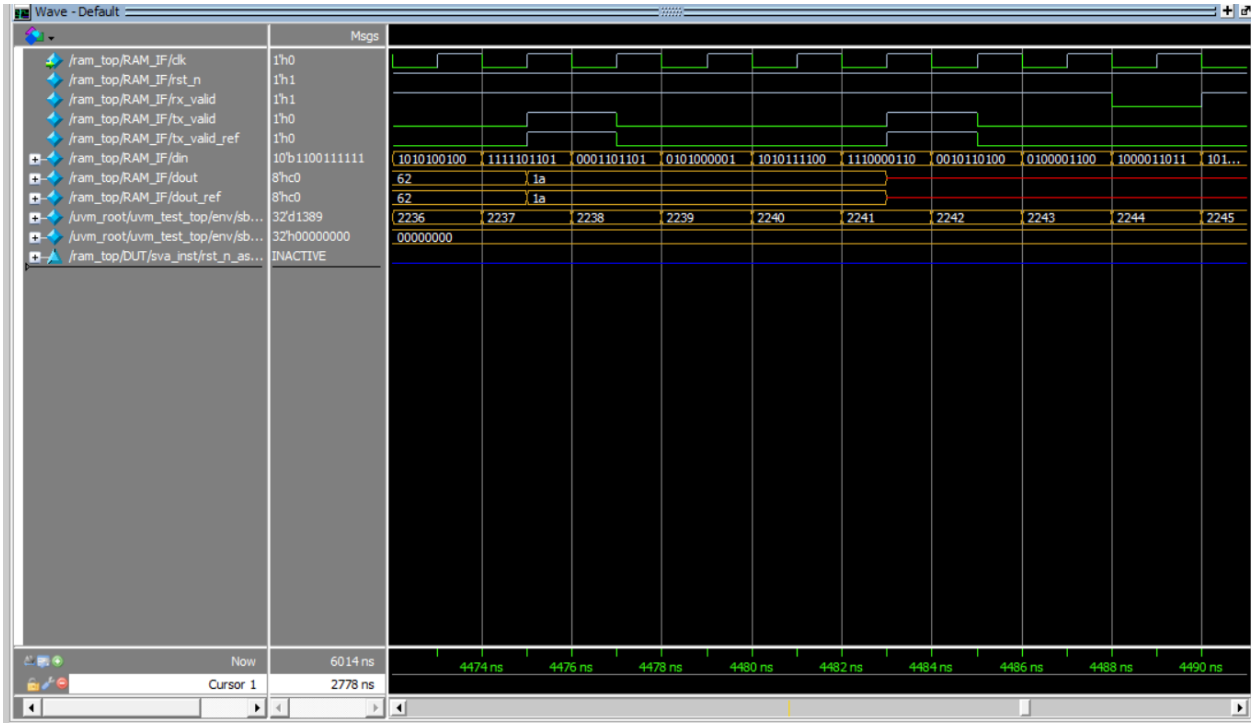
Write-Only Sequence:



Read-Only Sequence:



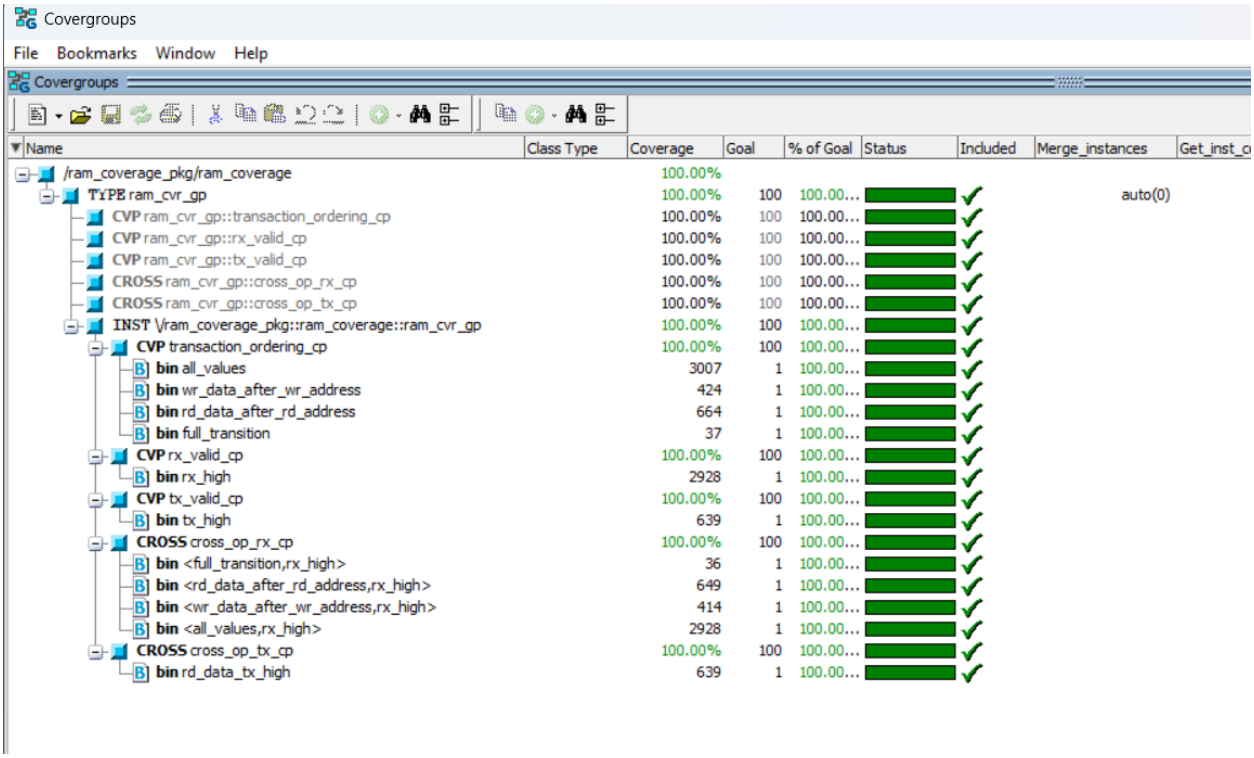
Write & Read Sequence:



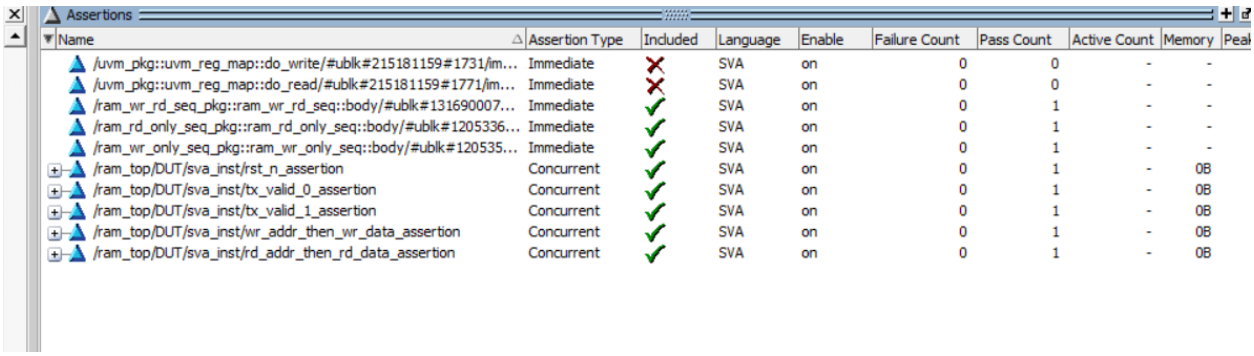
Transcript:

```
#   with 'UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(all)
# UVM_INFO @ 0: reporter [RNTST] Running test ram_test...
# UVM_INFO ram_test.sv(46) @ 0: uvm_test_top [run_phase] rst signal asserted
# ****************************************************************************
# * Questa UVM Transaction Recording Turned ON.                              *
# * recording_detail has been set.                                           *
# *  To turn off, set 'recording_detail' to off:                             *
# * uvm_config_db#(int)            ::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
# ****************************************************************************
# UVM_INFO ram_test.sv(48) @ 14: uvm_test_top [run_phase] rst signal deasserted
# UVM_INFO ram_test.sv(52) @ 14: uvm_test_top [run_phase] write only sequence started
# UVM_INFO ram_test.sv(54) @ 2014: uvm_test_top [run_phase] write only sequence ended
# UVM_INFO ram_test.sv(58) @ 2014: uvm_test_top [run_phase] read only sequence started
# UVM_INFO ram_test.sv(60) @ 4014: uvm_test_top [run_phase] read only sequence ended
# UVM_INFO ram_test.sv(64) @ 4014: uvm_test_top [run_phase] write & read sequence started
# UVM_INFO ram_test.sv(66) @ 6014: uvm_test_top [run_phase] write & read sequence ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 6014: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   12
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [run_phase]     8
# ** Note: $finish    : D:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 6014 ns  Iteration: 61  Instance: /ram_top
# Break in Task uvm_pkg/uvm_root::run_test at D:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430

VSIM 2>
```

Functional Coverage:



Assertions Coverage:



Code Coverage:

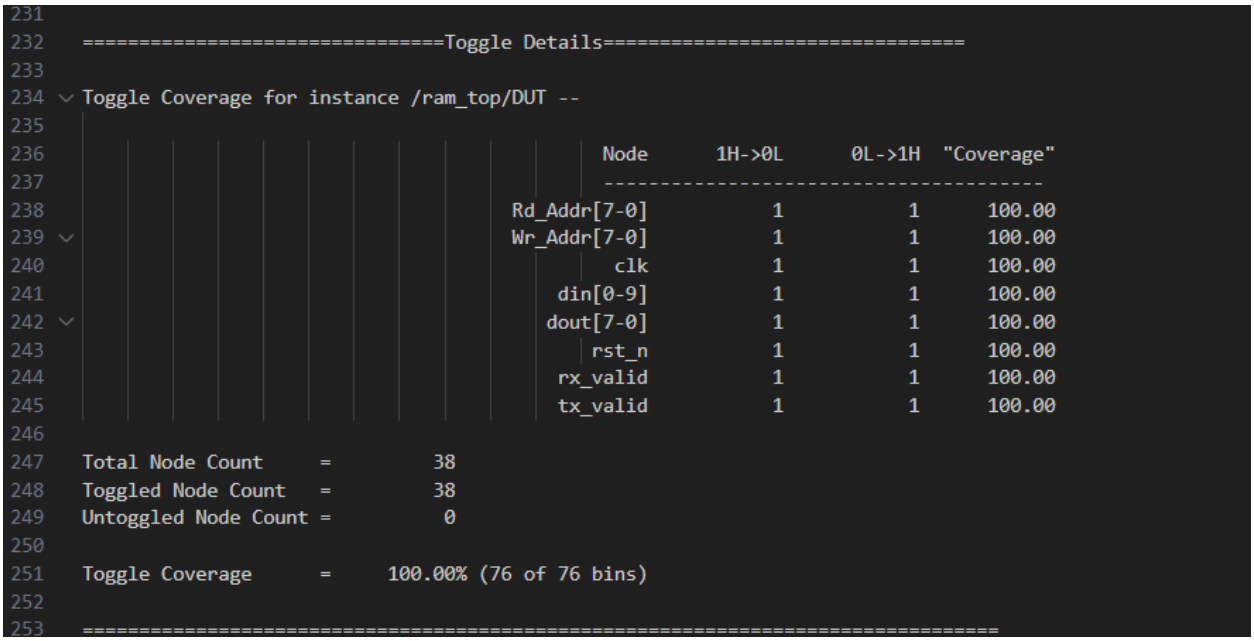
cvr_rpt.txt

```
Coverage Report by instance with details

================================================================================
=== Instance: /ram_top/RAM_IF
=== Design Unit: work.ram_if
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         62        62         0   100.00%

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

Toggle Coverage for instance /ram_top/RAM_IF --

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

Total Node Count      =         31
Toggled Node Count    =         31
Untoggled Node Count =          0

Toggle Coverage       =     100.00% (62 of 62 bins)

================================================================================
```

```
================================================================================
=== Instance: /ram_top/DUT/sva_inst
=== Design Unit: work.ram_sva
================================================================================

Assertion Coverage:
    Assertions                       5         5         0   100.00%
--------------------------------------------------------------------
Name                 File(Line)                   Failure      Pass
                                                  Count        Count
--------------------------------------------------------------------
/ram_top/DUT/sva_inst/rst_n_assertion
                     ram_sva.sv(2)                      0          1
/ram_top/DUT/sva_inst/tx_valid_0_assertion
                     ram_sva.sv(3)                      0          1
/ram_top/DUT/sva_inst/tx_valid_1_assertion
                     ram_sva.sv(4)                      0          1
/ram_top/DUT/sva_inst/wr_addr_then_wr_data_assertion
                     ram_sva.sv(5)                      0          1
/ram_top/DUT/sva_inst/rd_addr_then_rd_data_assertion
                     ram_sva.sv(6)                      0          1
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         44        44         0   100.00%

================================Toggle Details================================
```

```
================================Toggle Details================================

Toggle Coverage for instance /ram_top/DUT/sva_inst --

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

Total Node Count     =         22
Toggled Node Count   =         22
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (44 of 44 bins)

================================================================================
=== Instance: /ram_top/DUT
=== Design Unit: work.RAM
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         8         8         0   100.00%

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

```
121
122  Expression Coverage:
123      Enabled Coverage              Bins    Covered    Misses  Coverage
124      ----------------              ----       ----    ------  --------
125      Expressions                      3         3         0   100.00%
126
127  ================================Expression Details================================
128
129  Expression Coverage for instance /ram_top/DUT --
130
131    File RAM.v
132  ----------------Focused Expression View-----------------
133  Line       31 Item    1  ((din[9] && din[8]) && rx_valid)
134  Expression totals: 3 of 3 input terms covered = 100.00%
135
136    Input Term   Covered  Reason for no coverage   Hint
137   -----------  --------  ----------------------  --------------
138        din[9]         Y
139        din[8]         Y
140     rx_valid          Y
141
142       Rows:       Hits  FEC Target            Non-masking condition(s)
143   ---------  ---------  --------------------  -------------------------
144    Row   1:          1  din[9]_0              -
145    Row   2:          1  din[9]_1              (rx_valid && din[8])
146    Row   3:          1  din[8]_0              din[9]
147    Row   4:          1  din[8]_1              (rx_valid && din[9])
148    Row   5:          1  rx_valid_0           (din[9] && din[8])
149    Row   6:          1  rx_valid_1           (din[9] && din[8])
150
151
152  Statement Coverage:
153      Enabled Coverage              Bins      Hits    Misses  Coverage
154      ----------------              ----      ----    ------  --------
155      Statements                      10        10         0   100.00%
156
157  ================================Statement Details================================
```

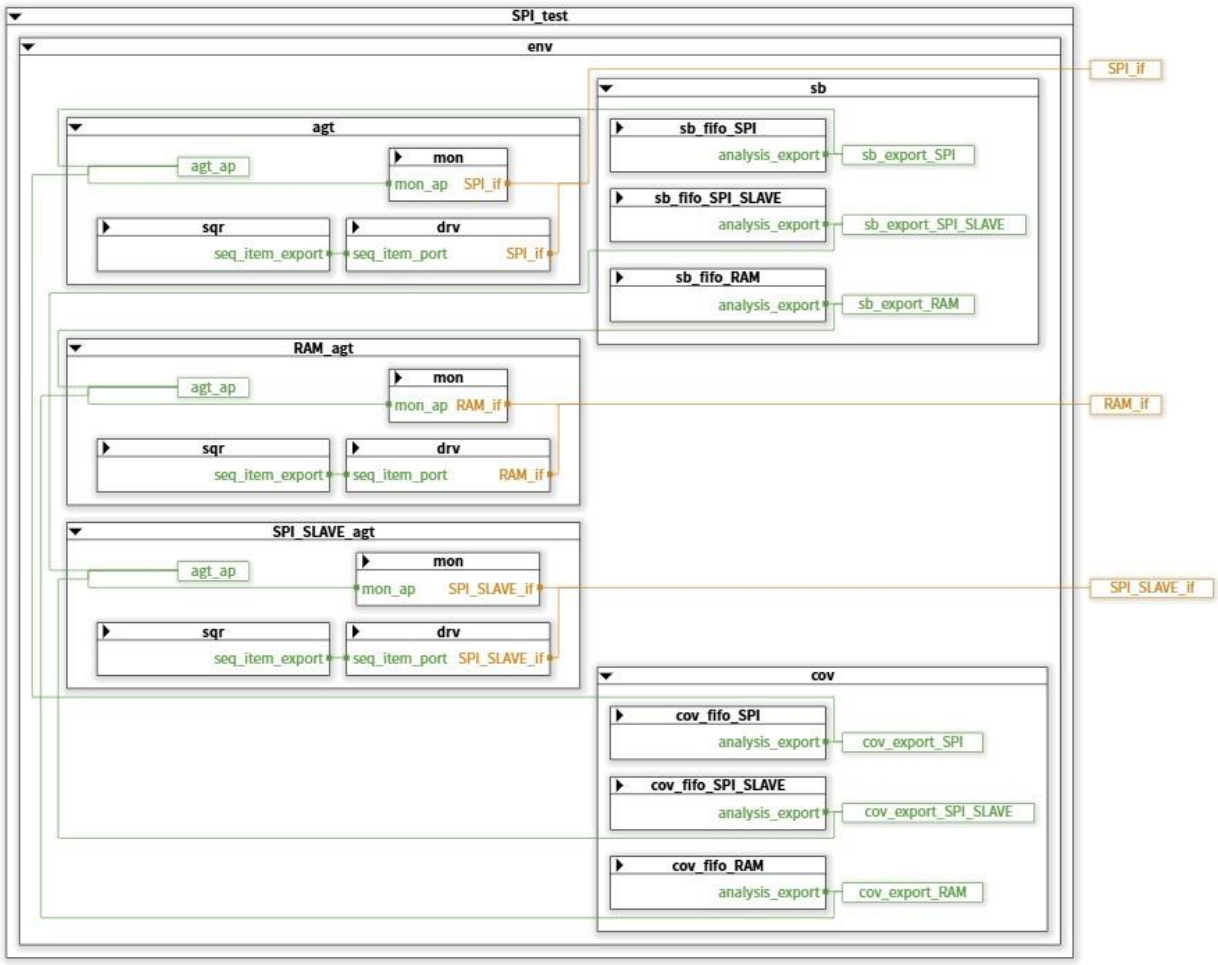


```
231
232   ================================Toggle Details================================
233
234   Toggle Coverage for instance /ram_top/DUT --
235
236                                                  Node      1H->0L      0L->1H  "Coverage"
237                                                  ----------------------------------------
238                                           Rd_Addr[7-0]           1           1      100.00
239                                           Wr_Addr[7-0]           1           1      100.00
240                                                   clk           1           1      100.00
241                                              din[0-9]           1           1      100.00
242                                             dout[7-0]           1           1      100.00
243                                                 rst_n           1           1      100.00
244                                              rx_valid           1           1      100.00
245                                              tx_valid           1           1      100.00
246
247   Total Node Count     =          38
248   Toggled Node Count   =          38
249   Untoggled Node Count =           0
250
251   Toggle Coverage      =     100.00% (76 of 76 bins)
252
253   ================================================================================
```

Default branch excluded as all cases are covered.

3-SPI Wrapper:

3.1. SPI Wrapper UVM architecture:



The environment includes a scoreboard, coverage, and three agents, one for each component (RAM, SPI Slave, and SPI Wrapper).
The SPI Wrapper agent is active, and its driver is responsible for driving the data.
The SPI Slave and RAM agents are passive; they only monitor the data driven from the SPI Wrapper to the SPI Slave and RAM.

3.2. SPI Wrapper Assertions:

Feature	Assertion
An assertion ensures that whenever reset is asserted, the output (MISO) is inactive.	@(posedge clk) !rst_n |=> !MISO;
When din[9:8] = 11 and rx_valid is asserted, tx_valid should be high and should eventually fall.	sequence READ_DATA_SEQUENCE; $fell(SS_n) ##1 (MOSI[->3] ##0 1'b1); endsequence property MISO_STABLE_NOT_READ; @(posedge clk) disable iff (!rst_n) $fell(SS_n) \|=> (not READ_DATA_SEQUENCE ##1 ($stable(MISO) throughout (!SS_n))); endproperty

3.3. Codes:

```
module SPI_Wrapper (input clk,rst_n,SS_n,MOSI, output MISO);
parameter MEM_DEPTH=256, ADDR_SIZE=8;
wire [ADDR_SIZE+1:0] rx_data_din;
wire [ADDR_SIZE-1:0] tx_data_dout;
wire rx_valid,tx_valid;

/*module SPI_SLAVE (MOSI,SS_n,clk,rst_n,rx_data,tx_valid,tx_data,MISO,rx_valid);*/

SPI_SLAVE DUT_SPI(.clk(clk),.rst_n(rst_n),.SS_n(SS_n),.MOSI(MOSI),.tx_valid(tx_valid),.tx_data(tx_data_dout),.MISO(MISO),.rx_valid(rx_valid),.rx_data(rx_data_din));


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

RAM DUT_RAM(.din(rx_data_din),.clk(clk),.rst_n(rst_n),.rx_valid(rx_valid), .dout(tx_data_dout),.tx_valid(tx_valid));

endmodule
```

Figure 1: SPI Wrapper RTL Design

```
module SPI_Wrapper_GOLDEN (input clk,rst_n,SS_n,MOSI, output MISO);
parameter MEM_DEPTH=256, ADDR_SIZE=8;
wire [ADDR_SIZE+1:0] rx_data_din;
wire [ADDR_SIZE-1:0] tx_data_dout;
wire rx_valid,tx_valid;

/*module SPI_SLAVE_GOLDEN (MOSI,SS_n,clk,rst_n,rx_data,tx_valid,tx_data,MISO,rx_valid);*/

SPI_SLAVE_GOLDEN DUT_SPI_GOLDEN(.clk(clk),.rst_n(rst_n),.SS_n(SS_n),.MOSI(MOSI),.tx_valid(tx_valid),.tx_data(tx_data_dout),.MISO(MISO),.rx_valid(rx_valid),.rx_data(rx_data_din));

/*module RAM_GOLDEN (clk, rst_n, din, rx_valid, dout, tx_valid);*/

RAM_GOLDEN #(.MEM_DEPTH(MEM_DEPTH),.ADDR_SIZE(ADDR_SIZE)) DUT_RAM_GOLDEN(.din(rx_data_din),.clk(clk),.rst_n(rst_n),.rx_valid(rx_valid), .dout(tx_data_dout),.tx_valid(tx_valid));

endmodule
```

Figure 2: SPI Wrapper Golden Model RTL Design

```
module SPI_sva(input clk, rst_n, SS_n, MOSI, MISO);

//RESET
property RESET;
    @(posedge clk) !rst_n |=> !MISO;
endproperty

// Detect a READ_DATA command pattern
sequence READ_DATA_SEQUENCE;
    $fell(SS_n) ##1 (MOSI[->3] ##0 1'b1); // detect 3 bits + some end marker
endsequence

// Property: whenever SS_n is low, if READ_DATA_SEQUENCE does NOT start,
// then MISO must stay stable while SS_n is low.
property MISO_STABLE_NOT_READ;
    @(posedge clk) disable iff (!rst_n)
        $fell(SS_n) |=>
            (not READ_DATA_SEQUENCE ##1 ($stable(MISO) throughout (!SS_n)) );
endproperty

// Assertions
RESET_assert: assert property (RESET);
READ_DATA_assert: assert property (MISO_STABLE_NOT_READ);

// Coverage
RESET_cover: cover property (RESET);
READ_DATA_cover: cover property (MISO_STABLE_NOT_READ);

endmodule
```

Figure 3: SPI Wrapper sva

```
interface RAM_if(clk);
    parameter ADDR_SIZE = 8, MEM_DEPTH = 256;
    input bit clk;
    logic rst_n, rx_valid, tx_valid, tx_valid_ref;
    logic [ADDR_SIZE+1:0] din; // default word size + extra 2 bits
    logic [ADDR_SIZE-1:0] dout, dout_ref;

endinterface
```

Figure 4: RAM if

```
interface SPI_SLAVE_if (clk);
  //Inputs
  input bit clk;
  logic MOSI, rst_n, SS_n, tx_valid;
  logic [7:0] tx_data;
  //Outputs
  logic [9:0] rx_data,rx_data_ref;
  logic       rx_valid, MISO, MISO_ref, rx_valid_ref;

  modport DUT(input clk,MOSI,rst_n,SS_n,tx_data,tx_valid, output MISO,rx_data,rx_valid);

endinterface : SPI_SLAVE_if
```

Figure 5: SPI Slave if

```
interface SPI_if (clk);
  //Parameters
  parameter MEM_DEPTH=256;
  parameter ADDR_SIZE=8;
  //Inputs
  input clk;
  logic rst_n, SS_n, MOSI;
  //Outputs
  logic MISO;
  //Outputs Golden
  logic MISO_GOLDEN;
endinterface : SPI_if
```

Figure 6: SPI Wrapper if

```
package RAM_shared_pkg;
    typedef enum bit [1:0] {WR_ADDR, WR_DATA, RD_ADDR, RD_DATA } op_e;
endpackage
```

Figure 7: RAM Package

```
package SPI_SLAVE_shared_pkg;
typedef enum  {IDLE,CHK_CMD ,WRITE,READ_ADD , READ_DATA } current_st;
int counter_allcases =0  ;
int counter_read=0;
 logic SS_n_prev=1;
endpackage
```

Figure 8: SPI Slave Package

```
package SPI_pkg;
    typedef enum {IDLE, CHK_CMD, WRITE, READ_ADD, READ_DATA} current_st;
    typedef enum bit [0:2] {WR_ADDR=3'b000, WR_DATA=3'b001, RD_ADDR=3'b110, RD_DATA=3'b111} op_e;
    parameter MEM_DEPTH=256, ADDR_SIZE=8;
    logic SS_n_prev = 1; // To detect the falling edge of SS_n
    //Counters for sequences
    int counter_allcases = 0;
    int counter_read = 0;
endpackage
```

Figure 9: SPI Wrapper Package

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

    class SPI_config extends uvm_object;
        `uvm_object_utils(SPI_config)

        virtual SPI_if SPI_vif;
        virtual SPI_SLAVE_if SPI_SLAVE_vif;
        virtual RAM_if RAM_vif;
        uvm_active_passive_enum is_active;

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

endpackage
```

Figure 10: Config

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

  class SPI_sequencer extends uvm_sequencer #(SPI_seq_item);
    `uvm_component_utils(SPI_sequencer)

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

  endclass
endpackage
```

Figure 11: SPI Wrapper Sequencer

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

    class ram_seq_item extends uvm_sequence_item;
        `uvm_object_utils(ram_seq_item)
        //Inputs
        logic rst_n, rx_valid;
        logic [9:0] din;
        //Outputs
        logic tx_valid, tx_valid_ref;
        logic [7:0] dout, dout_ref;

        function new(string name = "ram_seq_item");
            super.new(name);
        endfunction
endclass
endpackage
```

Figure 12: RAM Sequence Item

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

  class spi_slave_seq_item extends uvm_sequence_item;
    `uvm_object_utils(spi_slave_seq_item)

    //Inputs
    logic [7:0] tx_data;
    logic  MOSI, SS_n, rst_n, tx_valid;
    //Outputs
    logic [9:0] rx_data, rx_data_ref;
    logic MISO, MISO_ref, rx_valid, rx_valid_ref;

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

  endclass
  endpackage
```

Figure 13: SPI Slave Sequence Item

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

    class SPI_seq_item extends uvm_sequence_item;
        `uvm_object_utils(SPI_seq_item)

        //Inputs
        rand logic rst_n, SS_n, MOSI;
        //Outputs
        logic MISO;
        //Outputs Golden
        logic MISO_GOLDEN;
        //array to help in MOSI randomization
        rand bit [0:10] array_rand;
        //Saving the last operation
        op_e old_operation;
        //array index to drive MOSI bit by bit
        int bit_index = 0;


        //Constraints
        //RESET
        constraint reset {rst_n dist {0:/2, 1:/98};}

        //SS_n_high
        constraint SS_n_high {
            if ((array_rand[0:2] inside {WR_ADDR, WR_DATA, RD_ADDR} && counter_allcases % 13 != 0)     //SS_n = 1 every 13 cycles
             || (array_rand[0:2] inside {RD_DATA} && counter_read % 23 != 0))                          //SS_n = 1 every 23 cycles
                SS_n==0;
            else
                SS_n==1;
        }

        //Write only
        constraint wr_only_constraint {
            // Always constrain to write operations (address or data)
            array_rand[0:2] inside {WR_ADDR, WR_DATA};
        }

        //Read only
        constraint rd_only_constraint {
            // Always constrain to read operations sequentially (address then data then address then ..etc)
            if(old_operation == RD_ADDR)
                array_rand[0:2] == RD_DATA;
            else if (old_operation == RD_DATA)
                array_rand[0:2] == RD_ADDR;
            else
                array_rand[0:2] inside {RD_ADDR, RD_DATA};
        }

        //Write and Read
        constraint rd_wr_constraint {
            if(old_operation == WR_ADDR) //Write_address
                array_rand[0:2] inside {WR_ADDR, WR_DATA};         // write_address or write_data
            else if (old_operation == WR_DATA) //write_data
                array_rand[0:2] dist {RD_ADDR:/ 60, WR_ADDR:/ 40}; // read_address = 60%, write_address = 40%
            else if (old_operation == RD_ADDR) // Read_address
                array_rand[0:2] == RD_DATA; //read_data
            else // Read_data
                array_rand[0:2] dist {WR_ADDR:/ 60, RD_ADDR:/ 40}; // Write Address = 60%, Read Address = 40%
        }


        function void post_randomize;
            // Start of a new frame when SS_n goes low
            if (SS_n_prev && !SS_n)
                bit_index = -1; // will be incremented to 0 in the next step (don't drive MOSI in the same cycle SS_n goes low(IDLE -> CHK_CMD))

            // While SS_n is low -> drive MOSI bit-by-bit
            if (!SS_n) begin
                MOSI = array_rand[bit_index];
                bit_index++;
                if (bit_index > 10)
                    bit_index = 0; // wrap after full frame
            end

            // update operation tracking
            old_operation = op_e'(array_rand[0:2]);
            SS_n_prev = SS_n;

            //Only increment counter_allcases counter when valid
            if (array_rand[0:2] inside {WR_ADDR, WR_DATA, RD_ADDR})
                counter_allcases++;
            if (counter_allcases == 13)
                counter_allcases = 0;

            //Only counter_read increment counter when valid
            if (array_rand[0:2] inside {RD_DATA})
                counter_read++;
            if (counter_read == 23)
                counter_read = 0;

        endfunction

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

    endclass

endpackage
```

Figure 14: SPI Wrapper Sequence Item

```
1   package SPI_reset_sequence_pkg;
2       import uvm_pkg::*;
3       `include "uvm_macros.svh"
4       import SPI_seq_item_pkg::*;
5       import SPI_pkg::*;
6
7       class SPI_reset_sequence extends uvm_sequence #(SPI_seq_item);
8           `uvm_object_utils(SPI_reset_sequence)
9
10          SPI_seq_item seq_item;
11
12          function new(string name = "SPI_reset_sequence");
13              super.new(name);
14          endfunction
15
16          task body;
17              seq_item = SPI_seq_item::type_id::create("seq_item");
18              start_item(seq_item);
19              seq_item.rst_n = 0;
20              seq_item.SS_n=0;
21              seq_item.MOSI=0;
22              finish_item(seq_item);
23          endtask
24
25      endclass
26
27  endpackage
```

Figure 15: Reset Sequence

```
package SPI_wr_only_sequence_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_seq_item_pkg::*;
    import SPI_pkg::*;

    class SPI_wr_only_sequence extends uvm_sequence #(SPI_seq_item);
        `uvm_object_utils(SPI_wr_only_sequence)

        SPI_seq_item seq_item;

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

    task body();
        seq_item = SPI_seq_item::type_id::create("seq_item");
        //Write only
        repeat(5000) begin
            start_item(seq_item);
            //At the beginning of the operation, enable the write sequence and randomization
            if(counter_allcases == 0) begin
                seq_item.array_rand.rand_mode(1);
                seq_item.wr_only_constraint.constraint_mode(1);
                seq_item.rd_only_constraint.constraint_mode(0);
                seq_item.rd_wr_constraint.constraint_mode(0);
            end
            else begin
                //Disable all constraints and randomization because we are already in an operation
                seq_item.wr_only_constraint.constraint_mode(0);
                seq_item.rd_only_constraint.constraint_mode(0);
                seq_item.rd_wr_constraint.constraint_mode(0);
                seq_item.array_rand.rand_mode(0);
            end
            assert (seq_item.randomize());
            finish_item(seq_item);
        end

    endtask

endclass
endpackage
```

Figure 16: Write only Sequence

```
package SPI_rd_only_sequence_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_seq_item_pkg::*;
    import SPI_pkg::*;

    class SPI_rd_only_sequence extends uvm_sequence #(SPI_seq_item);
        `uvm_object_utils(SPI_rd_only_sequence)

        SPI_seq_item seq_item;

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

    task body();
        seq_item = SPI_seq_item::type_id::create("seq_item");
        //Read only
        repeat(5000) begin
            start_item(seq_item);
            //At the beginning of the operation, enable the read sequence and randomization
            if(counter_allcases == 0 && (seq_item.array_rand[0:2] inside {WR_ADDR, WR_DATA, RD_ADDR})
            || counter_read == 0 && (seq_item.array_rand[0:2] inside {RD_DATA})) begin
                seq_item.wr_only_constraint.constraint_mode(0);
                seq_item.rd_only_constraint.constraint_mode(1);
                seq_item.rd_wr_constraint.constraint_mode(0);
                seq_item.array_rand.rand_mode(1);
            end
            else begin
                //Disable all constraints and randomization because we are already in an operation
                seq_item.wr_only_constraint.constraint_mode(0);
                seq_item.rd_only_constraint.constraint_mode(0);
                seq_item.rd_wr_constraint.constraint_mode(0);
                seq_item.array_rand.rand_mode(0);
            end
            assert (seq_item.randomize());
            finish_item(seq_item);
        end

    endtask

endclass
endpackage
```

Figure 17: Read only Sequence

```
package SPI_wr_rd_sequence_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_seq_item_pkg::*;
    import SPI_pkg::*;

    class SPI_wr_rd_sequence extends uvm_sequence #(SPI_seq_item);
        `uvm_object_utils(SPI_wr_rd_sequence)

        SPI_seq_item seq_item;

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

    task body();
        seq_item = SPI_seq_item::type_id::create("seq_item");
        //Write and Read
        repeat(5000) begin
            start_item(seq_item);
            //At the beginning of the operation, enable the read/write sequence and randomization
            if(counter_allcases == 0 && (seq_item.array_rand[0:2] inside {WR_ADDR, WR_DATA, RD_ADDR})
            || counter_read == 0 && (seq_item.array_rand[0:2] inside {RD_DATA})) begin
                seq_item.wr_only_constraint.constraint_mode(0);
                seq_item.rd_only_constraint.constraint_mode(0);
                seq_item.rd_wr_constraint.constraint_mode(1);
                seq_item.array_rand.rand_mode(1);
            end
            else begin
                //Disable all constraints and randomization because we are already in an operation
                seq_item.wr_only_constraint.constraint_mode(0);
                seq_item.rd_only_constraint.constraint_mode(0);
                seq_item.rd_wr_constraint.constraint_mode(0);
                seq_item.array_rand.rand_mode(0);
            end
            assert (seq_item.randomize());
            finish_item(seq_item);
        end

    endtask

endclass
endpackage
```

Figure 18: Write/Read Sequence

```
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)

    ram_seq_item item;
    virtual RAM_if ram_vif;
    uvm_analysis_port #(ram_seq_item) mon_ap;

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

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            item = ram_seq_item::type_id::create("item");
            @(negedge ram_vif.clk);
            //input rst_n, rx_valid, din, output tx_valid, tx_valid_ref, dout, dout_ref
            item.rst_n        = ram_vif.rst_n;
            item.rx_valid     = ram_vif.rx_valid;
            item.din          = ram_vif.din;
            item.tx_valid     = ram_vif.tx_valid;
            item.tx_valid_ref = ram_vif.tx_valid_ref;
            item.dout         = ram_vif.dout;
            item.dout_ref     = ram_vif.dout_ref;

            mon_ap.write(item);
        end
    endtask
endclass
endpackage
```

Figure 19: RAM Monitor

```
package spi_slave_monitor_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import spi_slave_seq_item_package::*;
    import SPI_SLAVE_shared_pkg ::*;

    class spi_slave_monitor extends uvm_monitor;
        `uvm_component_utils(spi_slave_monitor);

        virtual SPI_SLAVE_if ss_vif;
        spi_slave_seq_item seq_item;
        uvm_analysis_port #(spi_slave_seq_item) mon_ap;

        function new(string name ="spi_slave_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

        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            forever begin
                seq_item=spi_slave_seq_item::type_id::create("seq_item");
                @(negedge ss_vif.clk);
                //input tx_data, MOSI, SS_n, rst_n, tx_valid, output rx_data, rx_data_ref, MISO, MISO_ref, rx_valid, rx_valid_ref
                seq_item.rst_n        = ss_vif.rst_n;
                seq_item.MOSI         = ss_vif.MOSI;
                seq_item.SS_n         = ss_vif.SS_n;
                seq_item.MISO         = ss_vif.MISO;
                seq_item.MISO_ref     = ss_vif.MISO_ref;
                seq_item.tx_valid     = ss_vif.tx_valid;
                seq_item.tx_data      = ss_vif.tx_data;
                seq_item.rx_valid     = ss_vif.rx_valid;
                seq_item.rx_data      = ss_vif.rx_data;
                seq_item.rx_valid_ref = ss_vif.rx_valid_ref;
                seq_item.rx_data_ref  = ss_vif.rx_data_ref;

                mon_ap.write(seq_item);
            end
        endtask
    endclass
endpackage
```

Figure 20: SPI Slave Monitor

```
package SPI_monitor_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_seq_item_pkg::*;
    import SPI_pkg::*;

    class SPI_monitor extends uvm_monitor;
        `uvm_component_utils(SPI_monitor)

        virtual SPI_if SPI_vif;
        SPI_seq_item seq_item;
        uvm_analysis_port #(SPI_seq_item) mon_ap;

        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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                seq_item = SPI_seq_item::type_id::create("seq_item");
                //input rst_n, SS_n, MOSI, output MISO
                @(negedge SPI_vif.clk);
                seq_item.rst_n       = SPI_vif.rst_n;
                seq_item.SS_n        = SPI_vif.SS_n;
                seq_item.MOSI        = SPI_vif.MOSI;
                seq_item.MISO        = SPI_vif.MISO;
                seq_item.MISO_GOLDEN = SPI_vif.MISO_GOLDEN;
                mon_ap.write(seq_item);
            end
        endtask

    endclass

endpackage
```

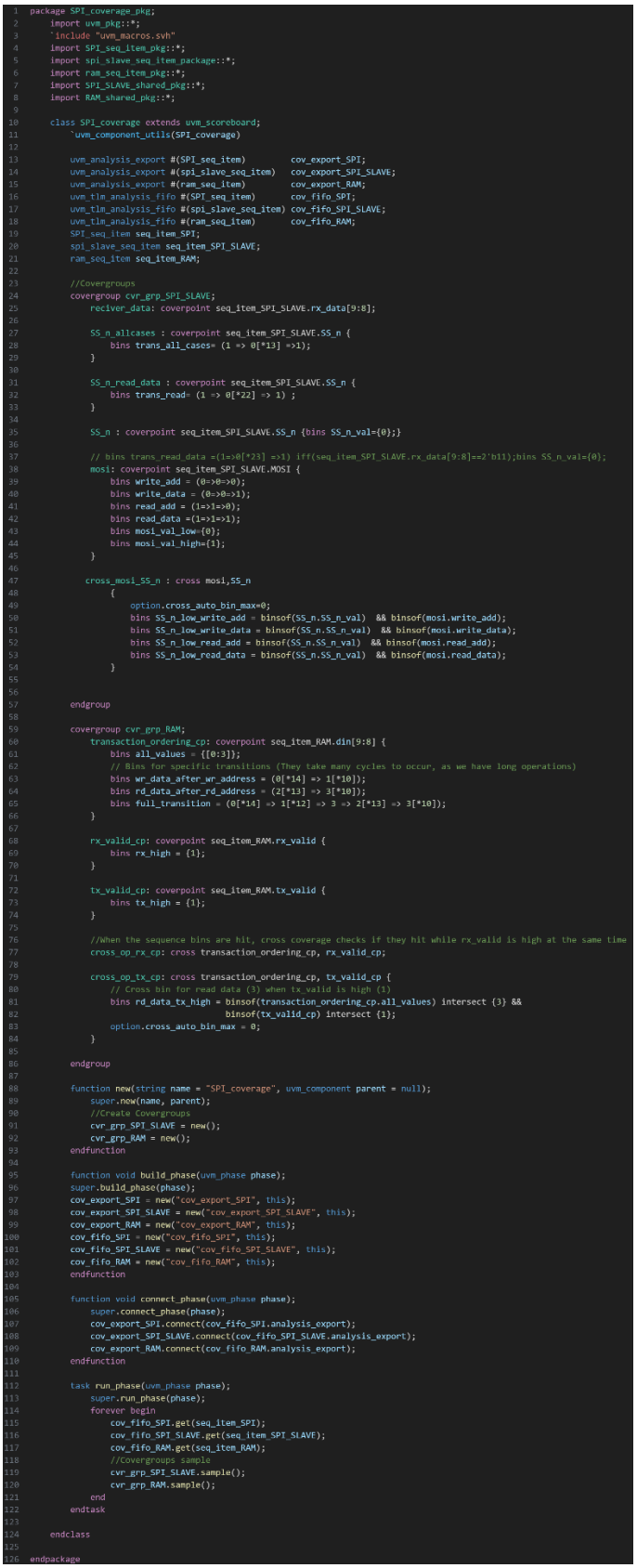
Figure 21: SPI Wrapper Monitor

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

    class SPI_driver extends uvm_driver #(SPI_seq_item);
        `uvm_component_utils(SPI_driver)

        virtual SPI_if SPI_vif;
        SPI_seq_item seq_item;

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                seq_item = SPI_seq_item::type_id::create("seq_item");
                seq_item_port.get_next_item(seq_item);
                //input rst_n, SS_n, MOSI, output MISO
                SPI_vif.rst_n = seq_item.rst_n;
                SPI_vif.SS_n  = seq_item.SS_n;
                SPI_vif.MOSI  = seq_item.MOSI;
                @(negedge SPI_vif.clk);
                seq_item_port.item_done();
            end
        endtask
    endclass
endpackage
```

Figure 22: SPI Wrapper Driver

```
package SPI_scoreboard_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_seq_item_pkg::*;
    import spi_slave_seq_item_package::*;
    import ram_seq_item_pkg::*;
    import SPI_SLAVE_shared_pkg::*;
    import RAM_shared_pkg::*;

    class SPI_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(SPI_scoreboard)

        uvm_analysis_export #(SPI_seq_item)         sb_export_SPI;
        uvm_analysis_export #(spi_slave_seq_item)   sb_export_SPI_SLAVE;
        uvm_analysis_export #(ram_seq_item)         sb_export_RAM;
        uvm_tlm_analysis_fifo #(SPI_seq_item)       sb_fifo_SPI;
        uvm_tlm_analysis_fifo #(spi_slave_seq_item) sb_fifo_SPI_SLAVE;
        uvm_tlm_analysis_fifo #(ram_seq_item)       sb_fifo_RAM;
        SPI_seq_item seq_item_SPI;
        spi_slave_seq_item seq_item_SPI_SLAVE;
        ram_seq_item seq_item_RAM;

        int correct_counter = 0;
        int error_counter_SPI = 0;
        int error_counter_SPI_SLAVE = 0;
        int error_counter_RAM = 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_SPI = new("sb_export_SPI", this);
        sb_export_SPI_SLAVE = new("sb_export_SPI_SLAVE", this);
        sb_export_RAM = new("sb_export_RAM", this);
        sb_fifo_SPI = new("sb_fifo_SPI", this);
        sb_fifo_SPI_SLAVE = new("sb_fifo_SPI_SLAVE", this);
        sb_fifo_RAM = new("sb_fifo_RAM", this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            sb_export_SPI.connect(sb_fifo_SPI.analysis_export);
            sb_export_SPI_SLAVE.connect(sb_fifo_SPI_SLAVE.analysis_export);
            sb_export_RAM.connect(sb_fifo_RAM.analysis_export);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo_SPI.get(seq_item_SPI);
                sb_fifo_SPI_SLAVE.get(seq_item_SPI_SLAVE);
                sb_fifo_RAM.get(seq_item_RAM);

                if (seq_item_SPI.MISO != seq_item_SPI.MISO_GOLDEN) begin
                    `uvm_error("ERROR", $sformatf("Error in Wrapper: rst_n=%b, SS_n=%b, MOSI=%b, MISO=%b, MISO_GOLDEN=%b", seq_item_SPI.rst_n, seq_item_SPI.SS_n, seq_item_SPI.MOSI, seq_item_SPI.MISO, seq_item_SPI.MISO_GOLDEN))
                    error_counter_SPI++;
                end

                //input tx_data, MOSI, SS_n, rst_n, tx_valid, output rx_data, rx_data_ref, MISO, MISO_ref, rx_valid, rx_valid_ref
                else if (seq_item_SPI_SLAVE.rx_data != seq_item_SPI_SLAVE.rx_data_ref || seq_item_SPI_SLAVE.rx_valid != seq_item_SPI_SLAVE.rx_valid_ref || seq_item_SPI_SLAVE.MISO != seq_item_SPI_SLAVE.MISO_ref) begin
                    `uvm_error("ERROR", $sformatf("Error in SPI_SLAVE"))
                    error_counter_SPI_SLAVE++;
                end

                //input rst_n, rx_valid, din, output tx_valid, tx_valid_ref, dout, dout_ref
                else if ((seq_item_RAM.dout != seq_item_RAM.dout_ref) || (seq_item_RAM.tx_valid != seq_item_RAM.tx_valid_ref)) begin
                    `uvm_error("ERROR", $sformatf("Error in RAM: rst_n=%b, rx_valid=%b, din=%b, tx_valid=%b, tx_valid_ref=%b, dout=%b, dout_ref=%b",
                    seq_item_RAM.rst_n, seq_item_RAM.rx_valid, seq_item_RAM.din, seq_item_RAM.tx_valid, seq_item_RAM.tx_valid_ref, seq_item_RAM.dout, seq_item_RAM.dout_ref))
                    error_counter_RAM++;
                end

                else correct_counter++;
            end
        endtask

        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("Report Phase", $sformatf("Correct = %d", correct_counter), UVM_MEDIUM)
            `uvm_info("Report Phase", $sformatf("SPI Wrapper Errors = %d", error_counter_SPI), UVM_MEDIUM)
            `uvm_info("Report Phase", $sformatf("SPI_SLAVE Errors = %d", error_counter_SPI_SLAVE), UVM_MEDIUM)
            `uvm_info("Report Phase", $sformatf("RAM Errors = %d", error_counter_RAM), UVM_MEDIUM)
        endfunction

    endclass

endpackage
```

Figure 23: Scoreboard



```
package SPI_coverage_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_seq_item_pkg::*;
    import spi_slave_seq_item_package::*;
    import ram_seq_item_pkg::*;
    import SPI_SLAVE_shared_pkg::*;
    import RAM_shared_pkg::*;

    class SPI_coverage extends uvm_scoreboard;
        `uvm_component_utils(SPI_coverage)

        uvm_analysis_export #(SPI_seq_item)         cov_export_SPI;
        uvm_analysis_export #(spi_slave_seq_item)   cov_export_SPI_SLAVE;
        uvm_analysis_export #(ram_seq_item)         cov_export_RAM;
        uvm_tlm_analysis_fifo #(SPI_seq_item)       cov_fifo_SPI;
        uvm_tlm_analysis_fifo #(spi_slave_seq_item) cov_fifo_SPI_SLAVE;
        uvm_tlm_analysis_fifo #(ram_seq_item)       cov_fifo_RAM;
        SPI_seq_item seq_item_SPI;
        spi_slave_seq_item seq_item_SPI_SLAVE;
        ram_seq_item seq_item_RAM;

        //Covergroups
        covergroup cvr_grp_SPI_SLAVE;
            reciver_data: coverpoint seq_item_SPI_SLAVE.rx_data[9:8];

            SS_n_allcases : coverpoint seq_item_SPI_SLAVE.SS_n {
                bins trans_all_cases= (1 => 0[*13] =>1);
            }

            SS_n_read_data : coverpoint seq_item_SPI_SLAVE.SS_n {
                bins trans_read= (1 => 0[*22] => 1) ;
            }

            SS_n : coverpoint seq_item_SPI_SLAVE.SS_n {bins SS_n_val={0};}

            // bins trans_read_data =(1=>0[*23] =>1) iff(seq_item_SPI_SLAVE.rx_data[9:8]==2'b11);bins SS_n_val={0};
            mosi: coverpoint seq_item_SPI_SLAVE.MOSI {
                bins write_add = (0=>0=>0);
                bins write_data = (0=>0=>1);
                bins read_add = (1=>1=>0);
                bins read_data =(1=>1=>1);
                bins mosi_val_low={0};
                bins mosi_val_high={1};
            }

           cross_mosi_SS_n : cross mosi,SS_n
                {
                    option.cross_auto_bin_max=0;
                    bins SS_n_low_write_add = binsof(SS_n.SS_n_val)  && binsof(mosi.write_add);
                    bins SS_n_low_write_data = binsof(SS_n.SS_n_val)  && binsof(mosi.write_data);
                    bins SS_n_low_read_add = binsof(SS_n.SS_n_val)  && binsof(mosi.read_add);
                    bins SS_n_low_read_data = binsof(SS_n.SS_n_val)  && binsof(mosi.read_data);
                }


        endgroup

        covergroup cvr_grp_RAM;
            transaction_ordering_cp: coverpoint seq_item_RAM.din[9:8] {
                bins all_values = {[0:3]};
                // Bins for specific transitions (They take many cycles to occur, as we have long operations)
                bins wr_data_after_wr_address = (0[*14] => 1[*10]);
                bins rd_data_after_rd_address = (2[*13] => 3[*10]);
                bins full_transition = (0[*14] => 1[*12] => 3 => 2[*13] => 3[*10]);
            }

            rx_valid_cp: coverpoint seq_item_RAM.rx_valid {
                bins rx_high = {1};
            }

            tx_valid_cp: coverpoint seq_item_RAM.tx_valid {
                bins tx_high = {1};
            }

            //When the sequence bins are hit, cross coverage checks if they hit while rx_valid is high at the same time
            cross_op_rx_cp: cross transaction_ordering_cp, rx_valid_cp;

            cross_op_tx_cp: cross transaction_ordering_cp, tx_valid_cp {
                // Cross bin for read data (3) when tx_valid is high (1)
                bins rd_data_tx_high = binsof(transaction_ordering_cp.all_values) intersect {3} &&
                                       binsof(tx_valid_cp) intersect {1};
                option.cross_auto_bin_max = 0;
            }

        endgroup

        function new(string name = "SPI_coverage", uvm_component parent = null);
            super.new(name, parent);
            //Create Covergroups
            cvr_grp_SPI_SLAVE = new();
            cvr_grp_RAM = new();
        endfunction

        function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        cov_export_SPI = new("cov_export_SPI", this);
        cov_export_SPI_SLAVE = new("cov_export_SPI_SLAVE", this);
        cov_export_RAM = new("cov_export_RAM", this);
        cov_fifo_SPI = new("cov_fifo_SPI", this);
        cov_fifo_SPI_SLAVE = new("cov_fifo_SPI_SLAVE", this);
        cov_fifo_RAM = new("cov_fifo_RAM", this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cov_export_SPI.connect(cov_fifo_SPI.analysis_export);
            cov_export_SPI_SLAVE.connect(cov_fifo_SPI_SLAVE.analysis_export);
            cov_export_RAM.connect(cov_fifo_RAM.analysis_export);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo_SPI.get(seq_item_SPI);
                cov_fifo_SPI_SLAVE.get(seq_item_SPI_SLAVE);
                cov_fifo_RAM.get(seq_item_RAM);
                //Covergroups sample
                cvr_grp_SPI_SLAVE.sample();
                cvr_grp_RAM.sample();
            end
        endtask

    endclass

endpackage
```

Figure 24: Coverage

```
package ram_agent_pkg;
import uvm_pkg::*;
import ram_seq_item_pkg::*;
import ram_sqr_pkg::*;
import ram_driver_pkg::*;
import ram_monitor_pkg::*;
import SPI_config_pkg::*;
`include "uvm_macros.svh"

class ram_agent extends uvm_agent;
    `uvm_component_utils(ram_agent)

    SPI_config cfg;
    ram_driver drv;
    ram_sqr sqr;
    ram_monitor mon;
    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 #(SPI_config)::get(this, "", "CONFIG_RAM", cfg))
            `uvm_fatal("build_phase", "unable to get configuration object for RAM agent")

        if (cfg.is_active == UVM_ACTIVE) begin
        sqr = ram_sqr::type_id::create("sqr", this);
        drv = ram_driver::type_id::create("drv", this);
        end
        mon = ram_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 (cfg.is_active == UVM_ACTIVE) begin
        drv.ram_vif = cfg.RAM_vif;
        drv.seq_item_port.connect(sqr.seq_item_export);
        end
        mon.ram_vif = cfg.RAM_vif;
        mon.mon_ap.connect(agt_ap);
    endfunction
endclass
endpackage
```

Figure 25: RAM Agent

```
package spi_slave_agent_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import spi_slave_seq_item_package::*;
  import SPI_config_pkg::*;
  import spi_slave_driver_pkg::*;
  import spi_slave_monitor_pkg::*;
  import spi_slave_sequencer_package::*;

  class spi_slave_agent extends uvm_agent;
    `uvm_component_utils(spi_slave_agent)

    spi_slave_seq_item seq_item;
    SPI_config          s_cfg;
    spi_slave_driver       drv;
    spi_slave_sequencer sqr;
    spi_slave_monitor       mon;
    uvm_analysis_port #(spi_slave_seq_item) agt_ap;

    function new(string name = "spi_slave_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, "", "CONFIG_SPI_SLAVE", s_cfg))
        `uvm_fatal("build_phase", "unable to get configuration object for SPI_SLAVE agent")

      if (s_cfg.is_active == UVM_ACTIVE) begin
        sqr = spi_slave_sequencer::type_id::create("sqr", this);
        drv = spi_slave_driver::type_id::create("drv", this);
      end
      mon = spi_slave_monitor::type_id::create("mon", this);
      agt_ap = new("agt_tb",this);
    endfunction

    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      if (s_cfg.is_active == UVM_ACTIVE) begin
      drv.ss_vif = s_cfg.SPI_SLAVE_vif;
      drv.seq_item_port.connect(sqr.seq_item_export);
      end
      mon.ss_vif = s_cfg.SPI_SLAVE_vif;
      mon.mon_ap.connect(agt_ap);
    endfunction
  endclass
endpackage
```

Figure 26: SPI Slave Agent

```
package SPI_agent_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_driver_pkg::*;
  import SPI_monitor_pkg::*;
  import SPI_sequencer_pkg::*;
  import SPI_config_pkg::*;
  import SPI_seq_item_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;
    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, "" , "CONFIG", SPI_cfg))
        `uvm_fatal("Build_Phase", "Error in Wrapper Interface in Agent");

      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_vif;
        drv.seq_item_port.connect(sqr.seq_item_export);
      end
      mon.SPI_vif = SPI_cfg.SPI_vif;
      mon.mon_ap.connect(agt_ap);
    endfunction

  endclass

endpackage
```

Figure 27: SPI Wrapper Agent

```
package SPI_env_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_agent_pkg::*;
  import spi_slave_agent_pkg::*;
  import ram_agent_pkg::*;
  import SPI_scoreboard_pkg::*;
  import SPI_coverage_pkg::*;

  class SPI_env extends uvm_env;
    `uvm_component_utils(SPI_env)

    SPI_agent agt;
    spi_slave_agent SPI_SLAVE_agt;
    ram_agent RAM_agt;
    SPI_scoreboard sb;
    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);
      SPI_SLAVE_agt = spi_slave_agent::type_id::create("SPI_SLAVE_agt", this);
      RAM_agt = ram_agent::type_id::create("RAM_agt", this);
      sb = SPI_scoreboard::type_id::create("sb", this);
      cov = SPI_coverage::type_id::create("cov", this);
    endfunction

    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      agt.agt_ap.connect(sb.sb_export_SPI);
      agt.agt_ap.connect(cov.cov_export_SPI);
      SPI_SLAVE_agt.agt_ap.connect(sb.sb_export_SPI_SLAVE);
      SPI_SLAVE_agt.agt_ap.connect(cov.cov_export_SPI_SLAVE);
      RAM_agt.agt_ap.connect(sb.sb_export_RAM);
      RAM_agt.agt_ap.connect(cov.cov_export_RAM);
    endfunction

  endclass

endpackage
```

Figure 28: Env

```
package SPI_test_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_config_pkg::*;
  import SPI_env_pkg::*;
  import SPI_sequencer_pkg::*;
  import SPI_reset_sequence_pkg::*;
  import SPI_wr_only_sequence_pkg::*;
  import SPI_rd_only_sequence_pkg::*;
  import SPI_wr_rd_sequence_pkg::*;

  class SPI_test extends uvm_test;
    `uvm_component_utils(SPI_test)

    SPI_config SPI_cfg, SPI_SLAVE_cfg, RAM_cfg;
    SPI_env env;
    SPI_reset_sequence rst_seq;
    SPI_wr_only_sequence wr_only_seq;
    SPI_rd_only_sequence rd_only_seq;
    SPI_wr_rd_sequence wr_rd_seq;

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

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      env = SPI_env::type_id::create("env",this);
      SPI_cfg = SPI_config::type_id::create("SPI_cfg");
      SPI_SLAVE_cfg = SPI_config::type_id::create("SPI_SLAVE_cfg");
      RAM_cfg = SPI_config::type_id::create("RAM_cfg");
      rst_seq = SPI_reset_sequence::type_id::create("rst_seq");
      wr_only_seq = SPI_wr_only_sequence::type_id::create("wr_only_seq");
      rd_only_seq = SPI_rd_only_sequence::type_id::create("rd_only_seq");
      wr_rd_seq = SPI_wr_rd_sequence::type_id::create("wr_rd_seq");

      if (!uvm_config_db #(virtual SPI_if)::get(this, "" , "SPI_IF", SPI_cfg.SPI_vif))
        `uvm_fatal("Build_Phase", "Error in Wrapper Interface in test");

      if (!uvm_config_db #(virtual SPI_SLAVE_if)::get(this, "" , "SPI_SLAVE_IF", SPI_SLAVE_cfg.SPI_SLAVE_vif))
        `uvm_fatal("Build_Phase", "Error in SPI_SLAVE Interface in test");

      if (!uvm_config_db #(virtual RAM_if)::get(this, "" , "RAM_IF", RAM_cfg.RAM_vif))
        `uvm_fatal("Build_Phase", "Error in RAM Interface in test");

      SPI_cfg.is_active = UVM_ACTIVE;
      SPI_SLAVE_cfg.is_active = UVM_PASSIVE;
      RAM_cfg.is_active = UVM_PASSIVE;

      uvm_config_db #(SPI_config)::set(this, "*", "CONFIG", SPI_cfg);
      uvm_config_db #(SPI_config)::set(this, "*", "CONFIG_SPI_SLAVE", SPI_SLAVE_cfg);
      uvm_config_db #(SPI_config)::set(this, "*", "CONFIG_RAM", RAM_cfg);
    endfunction

    task run_phase(uvm_phase phase);
      super.run_phase(phase);
      phase.raise_objection(this);

      //reset sequence
      `uvm_info("Run_Phase", "Reset Sequence Start.", UVM_MEDIUM)
      rst_seq.start(env.agt.sqr);
      `uvm_info("Run_Phase", "Reset Sequence End.", UVM_MEDIUM)

      //Write only sequence
      `uvm_info("Run_Phase", "Write Only Sequence Start.", UVM_MEDIUM)
      wr_only_seq.start(env.agt.sqr);
      `uvm_info("Run_Phase", "Write Only Sequence End.", UVM_MEDIUM)

      //Read only sequence
      `uvm_info("Run_Phase", "Read Only Sequence Start.", UVM_MEDIUM)
      rd_only_seq.start(env.agt.sqr);
      `uvm_info("Run_Phase", "Read Only Sequence End.", UVM_MEDIUM)

      //Write and Read sequence
      `uvm_info("Run_Phase", "Write and Read Sequence Start.", UVM_MEDIUM)
      wr_rd_seq.start(env.agt.sqr);
      `uvm_info("Run_Phase", "Write and Read Sequence End.", UVM_MEDIUM)

      phase.drop_objection(this);
    endtask

  endclass: SPI_test

endpackage
```

Figure 29: Test

```
import uvm_pkg::*;
`include "uvm_macros.svh"
import SPI_test_pkg::*;

module top();
  logic clk;
  initial begin
    clk = 0;
    forever #5 clk = ~clk;
  end

  //Interface
  SPI_if inter(clk);
  SPI_SLAVE_if inter_slave(clk);
  RAM_if inter_ram(clk);

  /*SPI Wrapper*/

  //DUT
  //module SPI_Wrapper (input clk,rst_n,SS_n,MOSI, output MISO);
  SPI_Wrapper #(inter.MEM_DEPTH, inter.ADDR_SIZE) DUT(inter.clk, inter.rst_n, inter.SS_n, inter.MOSI, inter.MISO);

  //GOLDEN
  //module SPI_Wrapper_GOLDEN (input clk,rst_n,SS_n,MOSI, output MISO);
  SPI_Wrapper_GOLDEN #(inter.MEM_DEPTH, inter.ADDR_SIZE) DUT_GOLDEN(inter.clk, inter.rst_n, inter.SS_n, inter.MOSI, inter.MISO_GOLDEN);


  /*SPI SLAVE*/

  //DUT
  //module SPI_SLAVE (MOSI,SS_n,clk,rst_n,rx_data,tx_valid,tx_data,MISO,rx_valid);
  SPI_SLAVE DUT_SPI_SLAVE(inter_slave.MOSI, inter_slave.SS_n, inter_slave.clk, inter_slave.rst_n, inter_slave.rx_data,
                          inter_slave.tx_valid, inter_slave.tx_data, inter_slave.MISO, inter_slave.rx_valid);

  //GOLDEN
  //module SPI_SLAVE_GOLDEN (MOSI,SS_n,clk,rst_n,rx_data,tx_valid,tx_data,MISO,rx_valid);
  SPI_SLAVE_GOLDEN DUT_SPI_SLAVE_GOLDEN(inter_slave.MOSI, inter_slave.SS_n, inter_slave.clk, inter_slave.rst_n, inter_slave.rx_data_ref,
                                        inter_slave.tx_valid, inter_slave.tx_data, inter_slave.MISO_ref, inter_slave.rx_valid_ref);
  assign inter_slave.rst_n    = DUT.rst_n;
  assign inter_slave.MOSI     = DUT.MOSI;
  assign inter_slave.SS_n     = DUT.SS_n;
  assign inter_slave.tx_valid = DUT.tx_valid;
  assign inter_slave.tx_data  = DUT.tx_data_dout;


  /*RAM*/

  //DUT
  //module RAM (din,clk,rst_n,rx_valid,dout,tx_valid);
  RAM DUT_RAM(inter_ram.din, inter_ram.clk, inter_ram.rst_n, inter_ram.rx_valid, inter_ram.dout, inter_ram.tx_valid);

  //GOLDEN
  //module RAM_GOLDEN (clk, rst_n, din, rx_valid, dout, tx_valid);
  RAM_GOLDEN #(inter_ram.MEM_DEPTH, inter_ram.ADDR_SIZE) DUT_RAM_GOLDEN(inter_ram.clk, inter_ram.rst_n, inter_ram.din,
              inter_ram.rx_valid, inter_ram.dout_ref, inter_ram.tx_valid_ref);
  assign inter_ram.rst_n     = DUT.rst_n;
  assign inter_ram.din       = DUT.rx_data_din;
  assign inter_ram.rx_valid  = DUT.rx_valid;


  //sva
  bind SPI_Wrapper SPI_sva Wrapper_BINDING(inter.clk, inter.rst_n, inter.SS_n, inter.MOSI, inter.MISO);
  bind RAM RAM_sva RAM_BINDING(inter_ram.din, inter_ram.clk, inter_ram.rst_n, inter_ram.rx_valid, inter_ram.dout, inter_ram.tx_valid);

  initial begin
    uvm_config_db #(virtual SPI_if)::set(null, "uvm_test_top", "SPI_IF", inter);
    uvm_config_db #(virtual SPI_SLAVE_if)::set(null, "uvm_test_top", "SPI_SLAVE_IF", inter_slave);
    uvm_config_db #(virtual RAM_if)::set(null, "uvm_test_top", "RAM_IF", inter_ram);
    run_test("SPI_test");
  end
endmodule
```

Figure 30: Top

```
vlib work
vlog -f src_files.list  +cover -covercells
vsim -voptargs=+acc work.top -cover -classdebug -uvmcontrol=all
add wave /top/inter/*
coverage save SPI.ucdb -onexit
run -all
```

Figure 31: Do File

```
RAM.v
RAM_GOLDEN.v
SPI_SLAVE.sv
SPI_SLAVE_GOLDEN.v
SPI_Wrapper.v
SPI_Wrapper_GOLDEN.v
RAM_sva.sv
SPI_sva.sv
SPI_SLAVE_if.sv
RAM_if.sv
SPI_Wrapper_if.sv
RAM_shared_pkg.sv
SPI_SLAVE_shared_pkg.sv
SPI_pkg.sv
SPI_config.sv
RAM_seq_item.sv
SPI_SLAVE_seq_item.sv
SPI_seq_item.sv
SPI_reset_sequence.sv
SPI_wr_only_sequence.sv
SPI_rd_only_sequence.sv
SPI_wr_rd_sequence.sv
RAM_sqr.sv
SPI_SLAVE_sequencer.sv
SPI_sequencer.sv
RAM_driver.sv
SPI_SLAVE_driver.sv
SPI_driver.sv
RAM_monitor.sv
SPI_SLAVE_monitor.sv
SPI_monitor.sv
SPI_scoreboard.sv
SPI_coverage.sv
RAM_agent.sv
SPI_SLAVE_agent.sv
SPI_agent.sv
SPI_env.sv
SPI_test.sv
top.sv
```

Figure 32: src File

3.4. Wave:

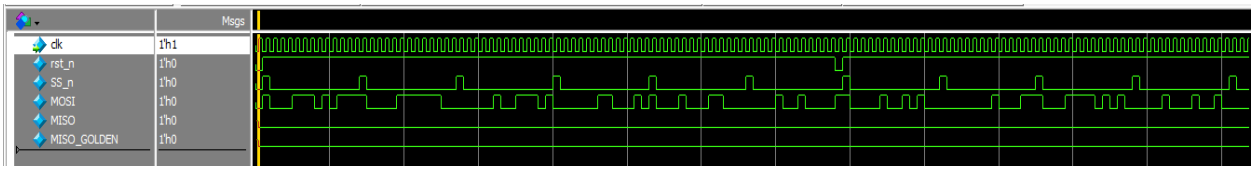


Figure 33: Reset Sequence Wave

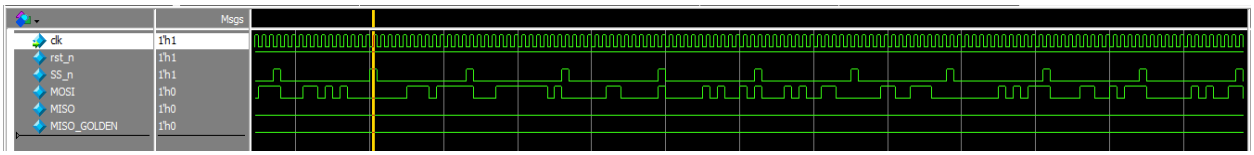


Figure 34: Write only Sequence Wave

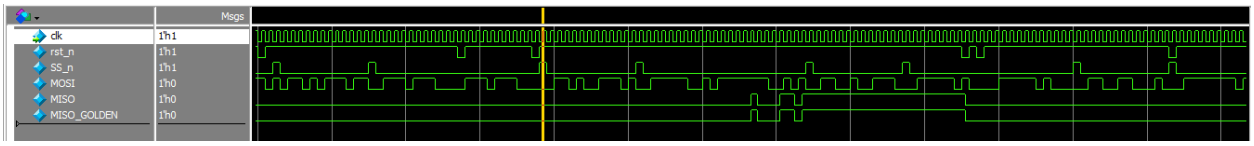


Figure 35: Read only Sequence Wave

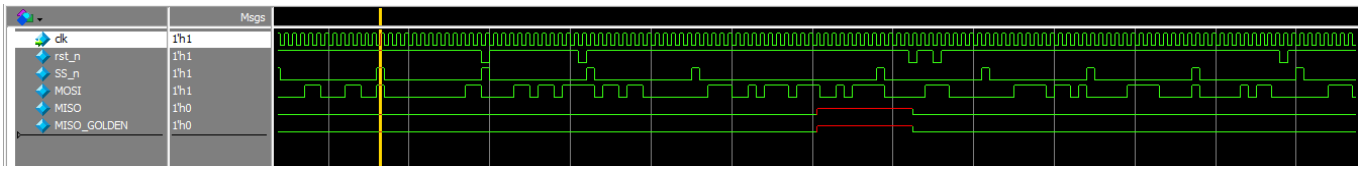


Figure 36: Write/Read Sequence Wave

Figure 37: Transcript

3.5. Coverage:

```
=== Instance: /top/inter_ram
=== Design Unit: work.RAM_if
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         62        62         0   100.00%

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

Toggle Coverage for instance /top/inter_ram --

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

Total Node Count     =         31
Toggled Node Count   =         31
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (62 of 62 bins)
```

Figure 38: Ram Toggle Coverage

```
=== Instance: /top/inter_slave
=== Design Unit: work.SPI_SLAVE_if
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         74        74         0   100.00%

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

Toggle Coverage for instance /top/inter_slave --

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

Total Node Count     =         37
Toggled Node Count   =         37
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (74 of 74 bins)
```

Figure 39: SPI Slave Toggle Coverage

```
=== Instance: /top/inter
=== Design Unit: work.SPI_if
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         12        12         0   100.00%

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

Toggle Coverage for instance /top/inter --

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

Total Node Count     =          6
Toggled Node Count   =          6
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (12 of 12 bins)
```

Figure 40: SPI Wrapper Toggle Coverage

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         8         8         0   100.00%
```

Figure 41: Ram Branch Coverage

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

Figure 42: Ram Statement Coverage

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        34        34         0   100.00%
```

Figure 43: SPI Slave Branch Coverage

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      39        39         0   100.00%
```

Figure 44: SPI Slave Statement Coverage

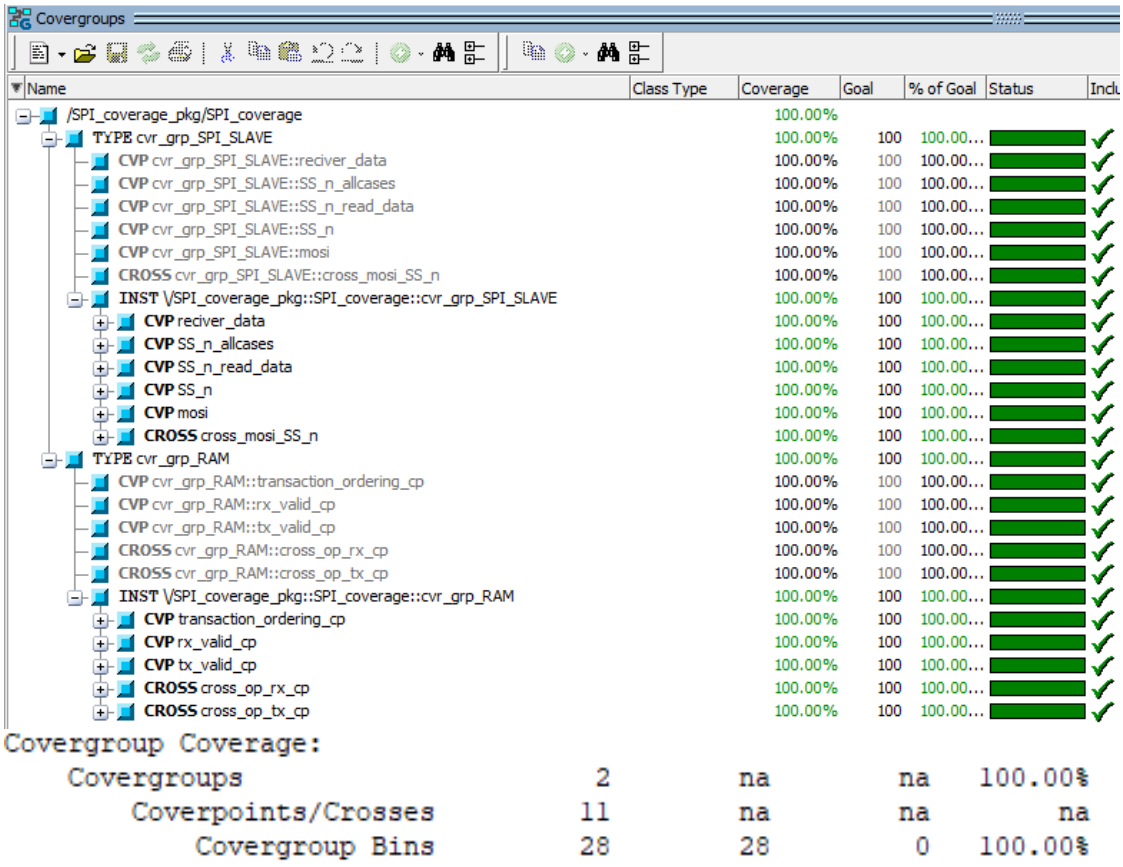
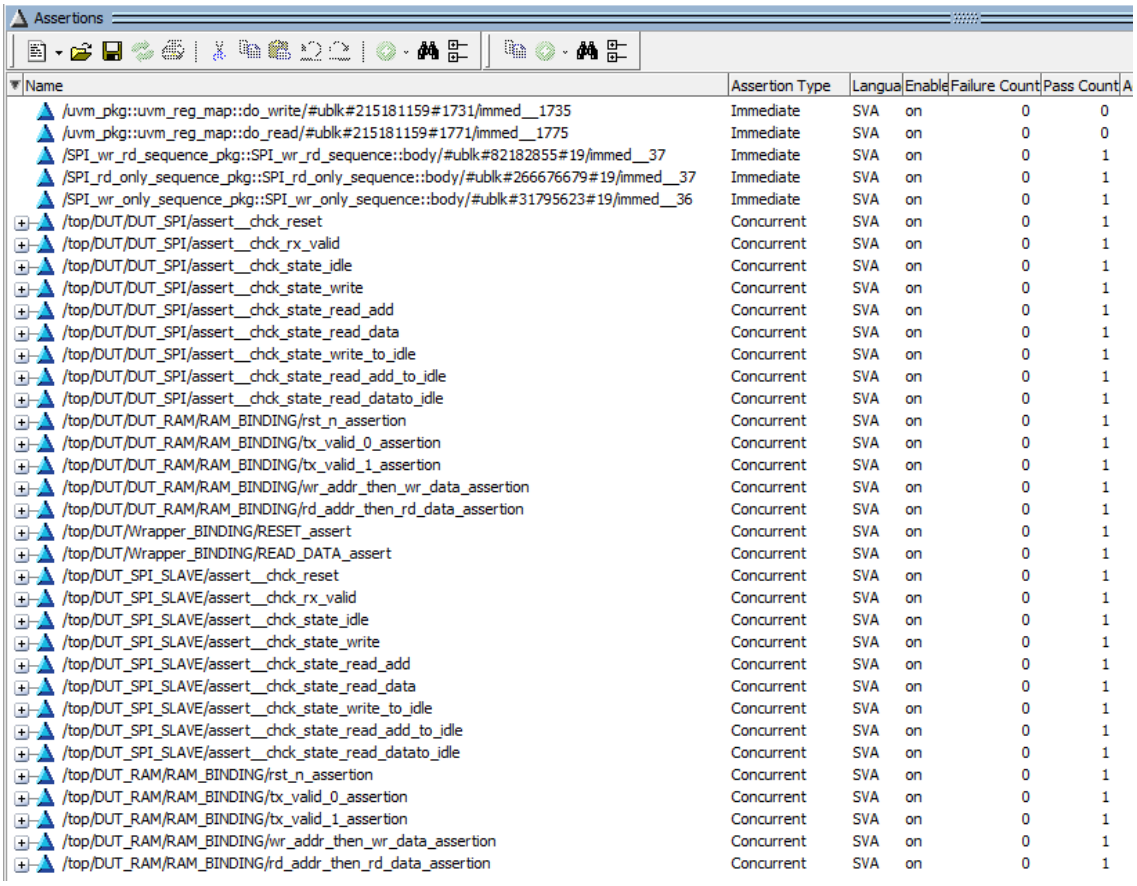
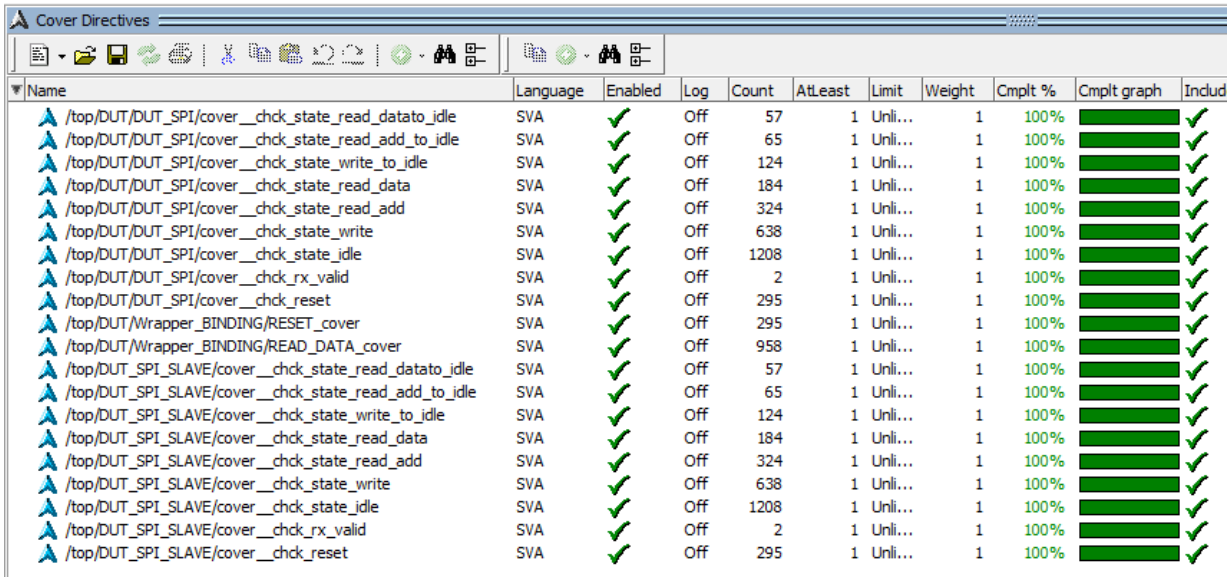


Figure 45: Functional Coverage



Name	Assertion Type	Langua	Enable	Failure Count	Pass Count	A
/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	
/SPI_wr_rd_sequence_pkg::SPI_wr_rd_sequence::body/#ublk#82182855#19/immed__37	Immediate	SVA	on	0	1	
/SPI_rd_only_sequence_pkg::SPI_rd_only_sequence::body/#ublk#266676679#19/immed__37	Immediate	SVA	on	0	1	
/SPI_wr_only_sequence_pkg::SPI_wr_only_sequence::body/#ublk#31795623#19/immed__36	Immediate	SVA	on	0	1	
/top/DUT/DUT_SPI/assert__chck_reset	Concurrent	SVA	on	0	1	
/top/DUT/DUT_SPI/assert__chck_rx_valid	Concurrent	SVA	on	0	1	
/top/DUT/DUT_SPI/assert__chck_state_idle	Concurrent	SVA	on	0	1	
/top/DUT/DUT_SPI/assert__chck_state_write	Concurrent	SVA	on	0	1	
/top/DUT/DUT_SPI/assert__chck_state_read_add	Concurrent	SVA	on	0	1	
/top/DUT/DUT_SPI/assert__chck_state_read_data	Concurrent	SVA	on	0	1	
/top/DUT/DUT_SPI/assert__chck_state_write_to_idle	Concurrent	SVA	on	0	1	
/top/DUT/DUT_SPI/assert__chck_state_read_add_to_idle	Concurrent	SVA	on	0	1	
/top/DUT/DUT_SPI/assert__chck_state_read_datato_idle	Concurrent	SVA	on	0	1	
/top/DUT/DUT_RAM/RAM_BINDING/rst_n_assertion	Concurrent	SVA	on	0	1	
/top/DUT/DUT_RAM/RAM_BINDING/tx_valid_0_assertion	Concurrent	SVA	on	0	1	
/top/DUT/DUT_RAM/RAM_BINDING/tx_valid_1_assertion	Concurrent	SVA	on	0	1	
/top/DUT/DUT_RAM/RAM_BINDING/wr_addr_then_wr_data_assertion	Concurrent	SVA	on	0	1	
/top/DUT/DUT_RAM/RAM_BINDING/rd_addr_then_rd_data_assertion	Concurrent	SVA	on	0	1	
/top/DUT/Wrapper_BINDING/RESET_assert	Concurrent	SVA	on	0	1	
/top/DUT/Wrapper_BINDING/READ_DATA_assert	Concurrent	SVA	on	0	1	
/top/DUT_SPI_SLAVE/assert__chck_reset	Concurrent	SVA	on	0	1	
/top/DUT_SPI_SLAVE/assert__chck_rx_valid	Concurrent	SVA	on	0	1	
/top/DUT_SPI_SLAVE/assert__chck_state_idle	Concurrent	SVA	on	0	1	
/top/DUT_SPI_SLAVE/assert__chck_state_write	Concurrent	SVA	on	0	1	
/top/DUT_SPI_SLAVE/assert__chck_state_read_add	Concurrent	SVA	on	0	1	
/top/DUT_SPI_SLAVE/assert__chck_state_read_data	Concurrent	SVA	on	0	1	
/top/DUT_SPI_SLAVE/assert__chck_state_write_to_idle	Concurrent	SVA	on	0	1	
/top/DUT_SPI_SLAVE/assert__chck_state_read_add_to_idle	Concurrent	SVA	on	0	1	
/top/DUT_SPI_SLAVE/assert__chck_state_read_datato_idle	Concurrent	SVA	on	0	1	
/top/DUT_RAM/RAM_BINDING/rst_n_assertion	Concurrent	SVA	on	0	1	
/top/DUT_RAM/RAM_BINDING/tx_valid_0_assertion	Concurrent	SVA	on	0	1	
/top/DUT_RAM/RAM_BINDING/tx_valid_1_assertion	Concurrent	SVA	on	0	1	
/top/DUT_RAM/RAM_BINDING/wr_addr_then_wr_data_assertion	Concurrent	SVA	on	0	1	
/top/DUT_RAM/RAM_BINDING/rd_addr_then_rd_data_assertion	Concurrent	SVA	on	0	1	

Figure 46: Assertion Coverage



Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Includ
/top/DUT/DUT_SPI/cover__chck_state_read_datato_idle	SVA	✓	Off	57	1	Unli...	1	100%		✓
/top/DUT/DUT_SPI/cover__chck_state_read_add_to_idle	SVA	✓	Off	65	1	Unli...	1	100%		✓
/top/DUT/DUT_SPI/cover__chck_state_write_to_idle	SVA	✓	Off	124	1	Unli...	1	100%		✓
/top/DUT/DUT_SPI/cover__chck_state_read_data	SVA	✓	Off	184	1	Unli...	1	100%		✓
/top/DUT/DUT_SPI/cover__chck_state_read_add	SVA	✓	Off	324	1	Unli...	1	100%		✓
/top/DUT/DUT_SPI/cover__chck_state_write	SVA	✓	Off	638	1	Unli...	1	100%		✓
/top/DUT/DUT_SPI/cover__chck_state_idle	SVA	✓	Off	1208	1	Unli...	1	100%		✓
/top/DUT/DUT_SPI/cover__chck_rx_valid	SVA	✓	Off	2	1	Unli...	1	100%		✓
/top/DUT/DUT_SPI/cover__chck_reset	SVA	✓	Off	295	1	Unli...	1	100%		✓
/top/DUT/Wrapper_BINDING/RESET_cover	SVA	✓	Off	295	1	Unli...	1	100%		✓
/top/DUT/Wrapper_BINDING/READ_DATA_cover	SVA	✓	Off	958	1	Unli...	1	100%		✓
/top/DUT_SPI_SLAVE/cover__chck_state_read_datato_idle	SVA	✓	Off	57	1	Unli...	1	100%		✓
/top/DUT_SPI_SLAVE/cover__chck_state_read_add_to_idle	SVA	✓	Off	65	1	Unli...	1	100%		✓
/top/DUT_SPI_SLAVE/cover__chck_state_write_to_idle	SVA	✓	Off	124	1	Unli...	1	100%		✓
/top/DUT_SPI_SLAVE/cover__chck_state_read_data	SVA	✓	Off	184	1	Unli...	1	100%		✓
/top/DUT_SPI_SLAVE/cover__chck_state_read_add	SVA	✓	Off	324	1	Unli...	1	100%		✓
/top/DUT_SPI_SLAVE/cover__chck_state_write	SVA	✓	Off	638	1	Unli...	1	100%		✓
/top/DUT_SPI_SLAVE/cover__chck_state_idle	SVA	✓	Off	1208	1	Unli...	1	100%		✓
/top/DUT_SPI_SLAVE/cover__chck_rx_valid	SVA	✓	Off	2	1	Unli...	1	100%		✓
/top/DUT_SPI_SLAVE/cover__chck_reset	SVA	✓	Off	295	1	Unli...	1	100%		✓

Figure 47: Cover Directives

4- Verification Plan:

	Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
2	RAM_RESET	when reset is asserted , dout, tx_valid and all internal registers should be low	Directed at the beginning of simulation then randomized	-	Reference Model, Assertions
3	Write only	If rx_valid is asserted and din[9:8] is 00, write address register should be written to din[7:0]. If din[9:8] is 01, din[7:0] should be written to the memory address of wr_address	Randomized with constraint wr_only_constraint	Cover if the write operations have occurred or not	Reference Model, Assertions
4	Read only	If rx_valid is asserted and din[9:8] is 10, read address register should be written to din[7:0]. If din[9:8] is 11, dout should be read from memory address of rd_address	Randomized with constraint rd_only_constraint	Cover if the read operations have occurred or not	Reference Model, Assertions
5	Write and Read	Verify all operations	Randomized with constraint rd_wr_constraint	Cover if all operations have occurred or not	Reference Model, Assertions
6	RESET	When reset is asserted , MISO , rx_data , rx_valid will be low	Directed at the beginning of simulation then randomized with constraint to make reset asserted by probability 2%	-	Reference Model, Assertions
7	SS_n_high	verify if SS_n is asserted every 13 cycles when it's write or read address operations, or every 23 cycles when it's read data operation	Randomized with constraint SS_n_high	Cover SS_n sequence	Reference Model, Assertions

Figure 48: Verification Plan