

Project 2:

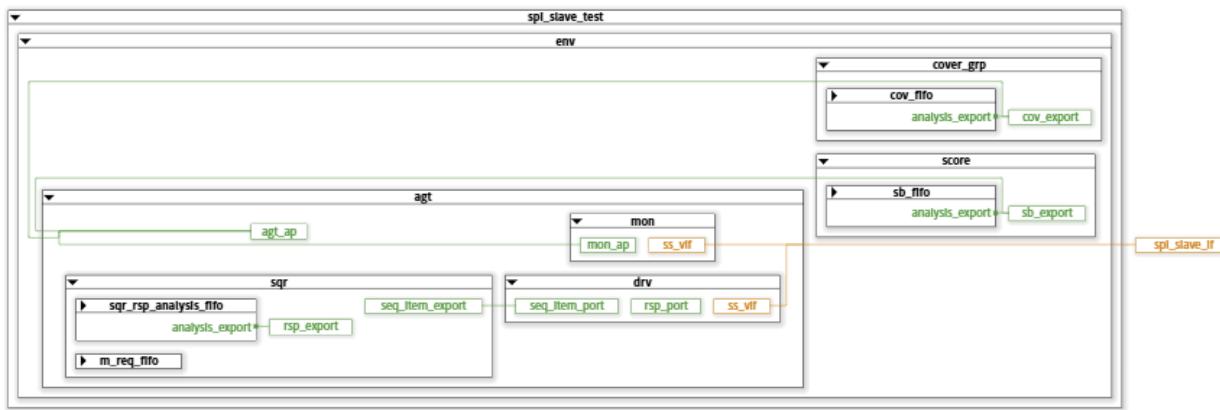
SPI with Single Port RAM Verification Project

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
    endcase
end

```

```

        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
            end
        endcase
    end
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
// 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 bug 1 fixed



```
1 if (~rst_n) begin
2     rx_data <= 0;
3     rx_valid <= 0;
4     received_address <= 0;
5     MISO <= 0;
6 end
```

Figure 3 bug 2



```
1 if (~ss_if.rst_n) begin
2     ss_if.rx_data <= 0;
3     ss_if.rx_valid <= 0;
4     received_address <= 0;
5     ss_if.MISO <= 0;
6     counter<=0; // reset
7 end
```

Figure 4 bug2 fixed



```
1 if (tx_valid) begin
2     rx_valid <= 0;
3     if (counter > 0) begin
4         MISO <= tx_data[counter-1];
5         counter <= counter - 1;
6     end
7     else begin
8         received_address <= 0;
9     end
10    end
```

Figure 5 bug 3



```
1 if (ss_if.tx_valid) begin
2         if (counter > 0) begin
3             ss_if.MISO <= ss_if.tx_data[counter-1];
4             counter <= counter - 1;
5         end
6         else begin
7             received_address <= 0;
8             ss_if.rx_valid <= 0;
9         end
10    end
```

Figure 6 bug 3 fixed



```
1 else begin
2         rx_valid <= 1;
3         counter <= 8;
4     end
```

Figure 7 bug 4



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

Figure 8 bug4 fixed

4. Reference 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
    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
            endcase
        end
    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 configuration 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
```

```

    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 ;
    receiver_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

```
1  spi_slave_if.sv
2  spi_slave_config.sv
3  SPI_slave.sv
4  SPI_Slave_interface.v
5  shared_pkg.sv
6  spi_slave_sequence_item.sv
7  spi_slave_reset_sequence.sv
8  | spi_slave_main_sequence.sv
9  | spi_slave_sequencer.sv
10 | spi_slave_driver.sv
11 | spi_slave_monitor.sv
12 | spi_slave_agent.sv
13 | spi_slave_scoreboard.sv
14 | spi_slave_collector.sv
15 | spi_slave_env.sv
16 | spi_slave_test.sv
17 | top.sv
```

21. Do file

```
1  vlib work
2  vlog -f src_files.list +define+SIM +cover -covercells
3  vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
4  add wave -r /top/ss_if/*
5  run -all
6  coverage save spi.ucdb -onexit
7
```

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
<hr/>				
TYPE /spi_slave_collector_package/spi_slave_coverage/covcode				
covered/total bins:	100.00%	100	-	Covered
missing/total bins:	15	15	-	
% Hit:	0	15	-	
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:		9	9	0	100.00%
DIRECTIVE COVERAGE:					
Name		Design Unit	Design Unit	Lang File(Line)	Hits Status

/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

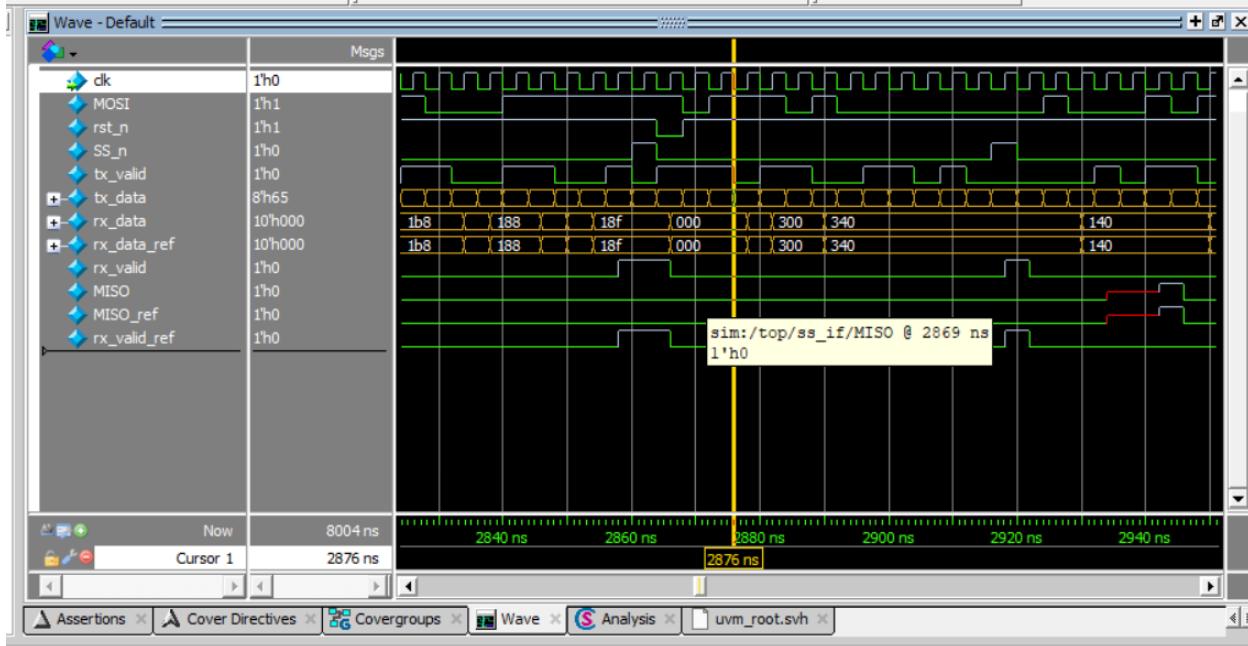


Figure 9 when ss_n is deasserted mosi read_add sequence (110)

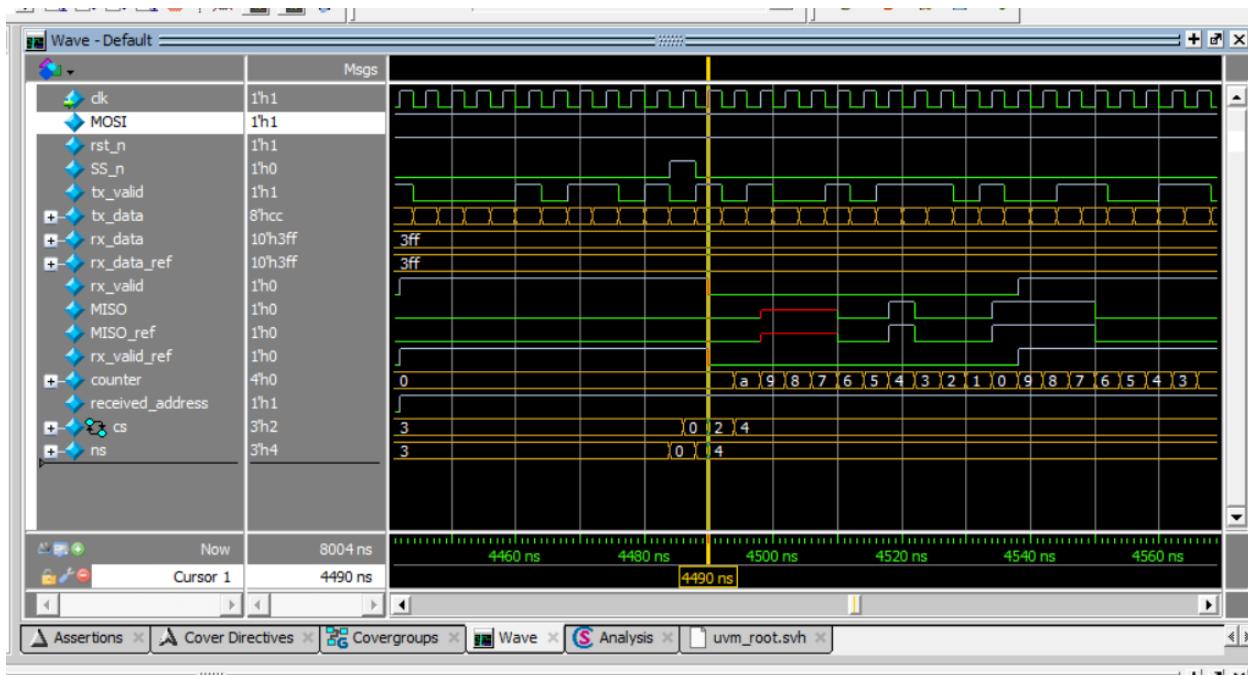


Figure 10 when ss_n is deasserted mosi read_data sequence (111)

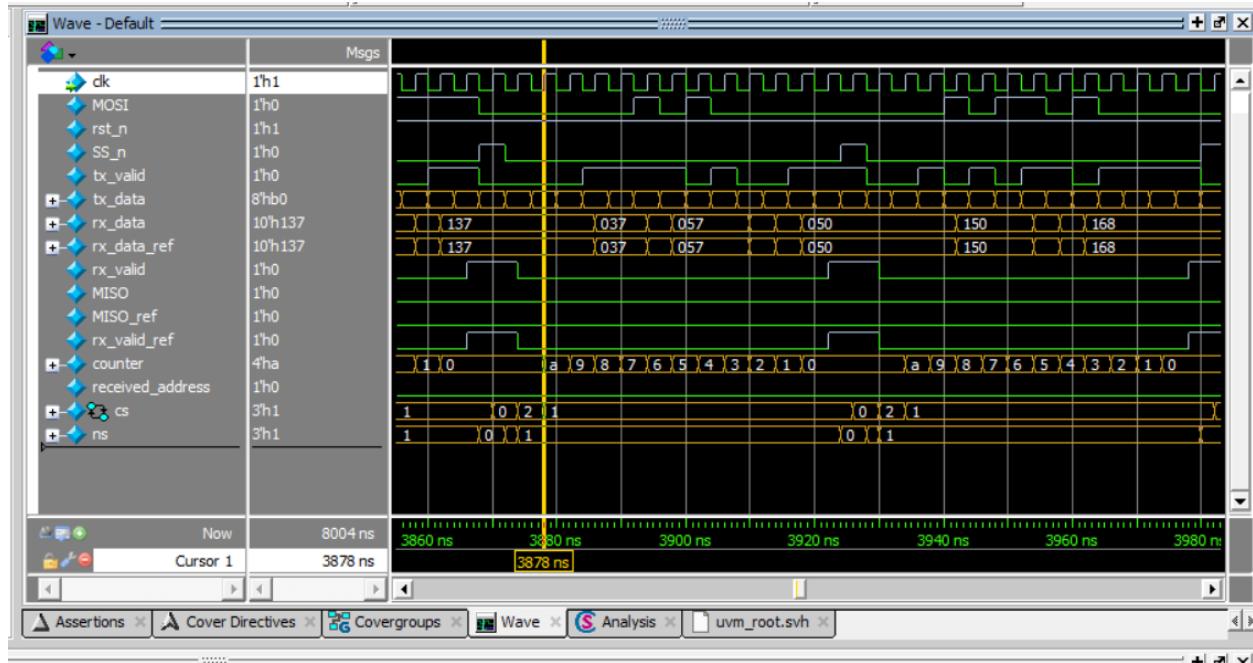


Figure 11 when `ss_n` is deasserted mosi write_add sequence (000)

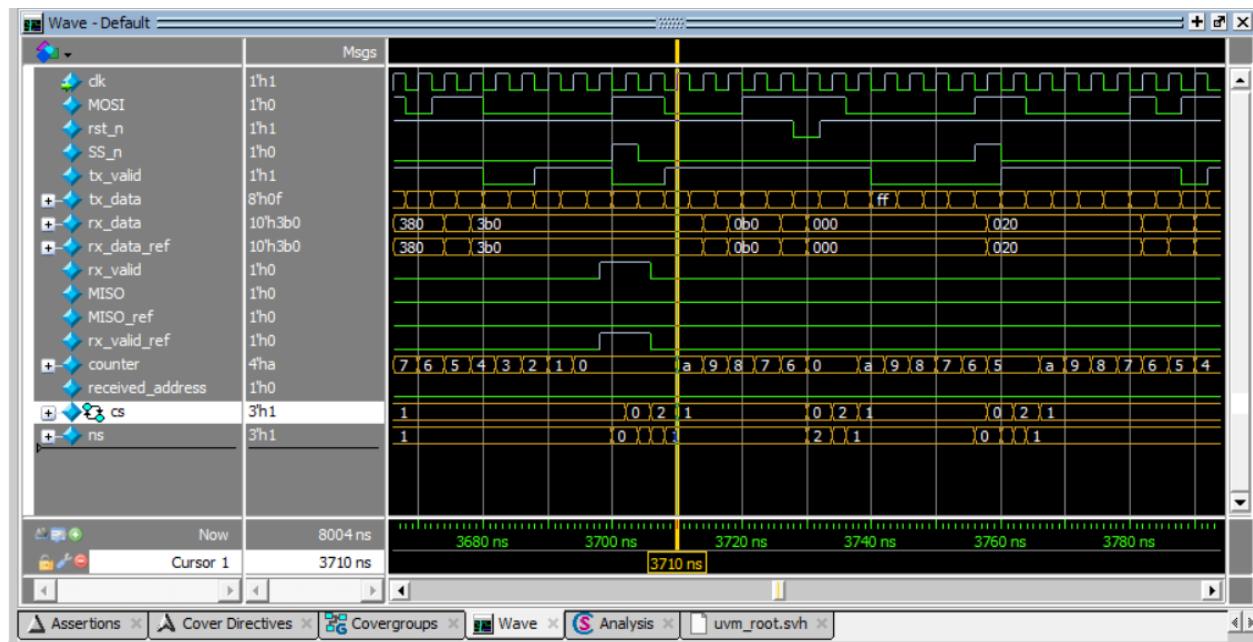


Figure 12 when `ss_n` is deasserted mosi write_data sequence (001)

26. Table of assertions with features

Feature	Assertion
Whenever the rst is asserted, MISO , rx_valid , rx_data is low	<pre>@(posedge ss_if.clk) (~ss_if.rst_n) =>(ss_if.MISO ==0 && ss_if.rx_valid==0 && ss_if.rx_data==0);</pre>
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	<pre>@(posedge ss_if.clk) disable iff(~ss_if.rst_n) (cs==IDLE && !ss_if.SS_n) =>(cs==CHK_CMD);</pre>
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	<pre>@(posedge ss_if.clk) disable iff(~ss_if.rst_n) (cs==CHK_CMD && !ss_if.SS_n && !ss_if.MOSI) => (cs==WRITE);</pre>
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	<pre>@(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);</pre>
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	<pre>@(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);</pre>
Whenever the rst is asserted and current state is WRITE , the current state will be IDEAL in next clock cycle	<pre>@(posedge ss_if.clk) (cs==WRITE &&(~ss_if.rst_n)) => (cs==IDLE);</pre>
Whenever the rst is asserted and current state is READ_ADD , the current state will be IDEAL in next clock cycle	<pre>@(posedge ss_if.clk) (cs==READ_ADD && (~ss_if.rst_n)) => (cs==IDLE);</pre>
Whenever the rst is asserted and current state is READ_DATA , the current state will be IDEAL in next clock cycle	<pre>@(posedge ss_if.clk) (cs==READ_DATA && (~ss_if.rst_n)) => (cs==IDLE);</pre>
Whenever SS_n is deasserted and write_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	<pre>@(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]);</pre>

Sequence which I mentioned in last row :

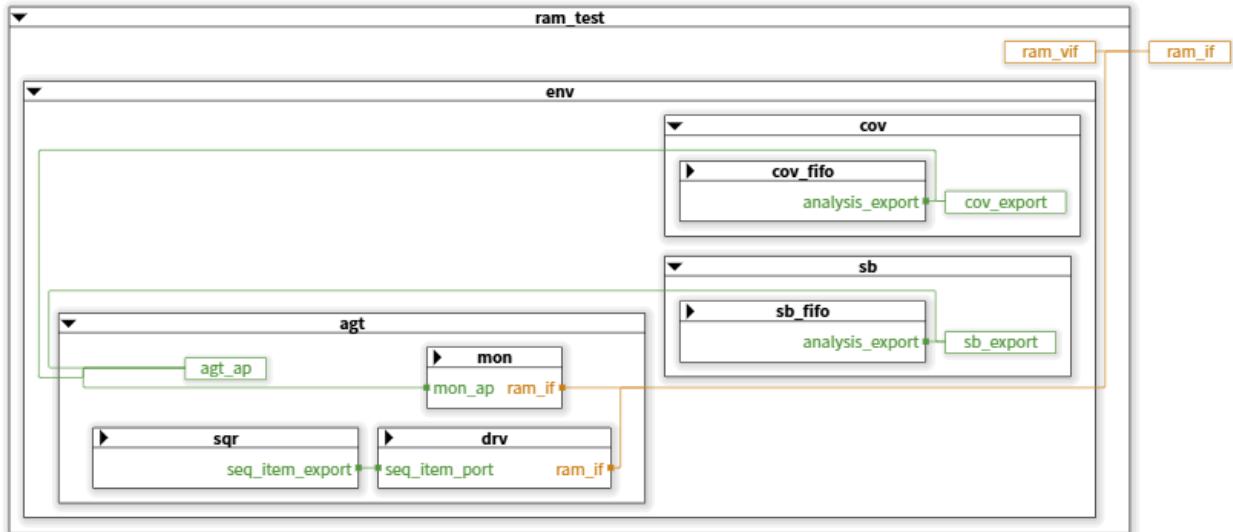


```
1 sequence write_add_seq;
2   (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 0)[*3];
3 endsequence
4 sequence write_data_seq;
5   (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 0)[*2] ##1(ss_if.
6 endsequence
7 sequence read_add_seq;
8   (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 1)[*2] ##1(ss_if.
9 endsequence
10
11 sequence read_data_seq;
12   (ss_if.SS_n==1) ##1 (ss_if.SS_n==0) ##1 (ss_if.MOSI == 1)[*3];
13 endsequence
```

2-RAM:

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

Ram Uvm architecture :



Top:

```
1  import uvm_pkg::*;
2  import ram_test_pkg::*;
3  import shared_pkg::*;
4  `include "uvm_macros.svh"
5  module ram_top();
6      bit clk;
7  initial begin
8      forever begin
9          #1 clk = ~clk;
10     end
11  end
12
13  ram_if RAM_IF(clk);
14  RAM_DUT(RAM_IF.din,RAM_IF.clk,RAM_IF.rst_n,RAM_IF.rx_valid,RAM_IF.dout,RAM_IF.tx_valid);
15  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);
16  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);
17
18  initial begin
19      uvm_config_db #(virtual ram_if)::set(null, "uvm_test_top", "RAM_IF", RAM_IF);
20      run_test("ram_test");
21  end
22
23 endmodule
```

Test:

```
1 package ram_test_pkg;
2 import uvm_pkg::*;
3 import ram_env_pkg::*;
4 import ram_cfg_obj_pkg::*;
5 import ram_seq_item_pkg::*;
6 import ram_reset_seq_pkg::*;
7 import ram_wr_only_seq_pkg::*;
8 import ram_rd_only_seq_pkg::*;
9 import ram_wr_rd_seq_pkg::*;
10 `include "uvm_macros.svh"
11
12 class ram_test extends uvm_test;
13     `uvm_component_utils(ram_test)
14     ram_cfg_obj ram_cfg;
15     virtual ram_if ram_vif;
16     ram_env env;
17     ram_reset_seq rst_seq;
18     ram_wr_only_seq wr_only_seq;
19     ram_rd_only_seq rd_only_seq;
20     ram_wr_rd_seq wr_rd_seq;
21
22
23     function new(string name = "test", uvm_component parent = null);
24         super.new(name, parent);
25     endfunction
26
27     function void build_phase(uvm_phase phase);
28         super.build_phase(phase);
29         ram_cfg = ram_cfg_obj::type_id::create("cfg_object");
30         env = ram_env::type_id::create("env", this);
31         rst_seq = ram_reset_seq::type_id::create("rst_seq");
32         wr_only_seq = ram_wr_only_seq::type_id::create("wr_only_seq");
33         rd_only_seq = ram_rd_only_seq::type_id::create("rd_only_seq");
34         wr_rd_seq = ram_wr_rd_seq::type_id::create("wr_rd_seq");
35
36         if(!uvm_config_db #(virtual ram_if)::get(this, "", "RAM_IF", ram_cfg.ram_vif)) begin
37             `uvm_fatal("build_phase", "unable to get virtual interface of ram")
38         end
39
40         uvm_config_db #(ram_cfg_obj)::set(this, "*", "RAM_CFG", ram_cfg);
41     endfunction
```

```
42
43     task run_phase(uvm_phase phase);
44         super.run_phase(phase);
45         phase.raise_objection(this);
46         `uvm_info("run_phase", "rst signal asserted", UVM_LOW)
47         rst_seq.start(env.agt.sqr);
48         `uvm_info("run_phase", "rst signal deasserted", UVM_LOW)
49         phase.drop_objection(this);
50
51         phase.raise_objection(this);
52         `uvm_info("run_phase", "write only sequence started", UVM_LOW)
53         wr_only_seq.start(env.agt.sqr);
54         `uvm_info("run_phase", "write only sequence ended", UVM_LOW)
55         phase.drop_objection(this);
56
57         phase.raise_objection(this);
58         `uvm_info("run_phase", "read only sequence started", UVM_LOW)
59         rd_only_seq.start(env.agt.sqr);
60         `uvm_info("run_phase", "read only sequence ended", UVM_LOW)
61         phase.drop_objection(this);
62
63         phase.raise_objection(this);
64         `uvm_info("run_phase", "write & read sequence started", UVM_LOW)
65         wr_rd_seq.start(env.agt.sqr);
66         `uvm_info("run_phase", "write & read sequence ended", UVM_LOW)
67         phase.drop_objection(this);
68
69     endtask
70 endclass
71 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:

```
1 package ram_agent_pkg;
2 import uvm_pkg::*;
3 import ram_seq_item_pkg::*;
4 import ram_sqr_pkg::*;
5 import ram_driver_pkg::*;
6 import ram_monitor_pkg::*;
7 import ram_cfg_obj_pkg::*;
8 `include "uvm_macros.svh"
9
10 class ram_agent extends uvm_agent;
11     `uvm_component_utils(ram_agent)
12     ram_cfg_obj cfg;
13     ram_driver drv;
14     ram_sqr sqr;
15     ram_monitor mon;
16     uvm_analysis_port #(ram_seq_item) agt_ap;
17
18 function new(string name = "agt", uvm_component parent = null);
19     super.new(name, parent);
20 endfunction
21
22 function void build_phase(uvm_phase phase);
23     super.build_phase(phase);
24     cfg = ram_cfg_obj::type_id::create("cfg");
25     sqr = ram_sqr::type_id::create("sqr", this);
26     drv = ram_driver::type_id::create("drv", this);
27     mon = ram_monitor::type_id::create("mon", this);
28     agt_ap = new("agt_ap", this);
29
30 if(!uvm_config_db #(ram_cfg_obj)::get(this, "", "RAM_CFG", cfg)) begin
31     `uvm_fatal("build_phase", "unable to get cfg object")
32 end
33
34 endfunction
35
36 function void connect_phase(uvm_phase phase);
37     super.connect_phase(phase);
38     drv.ram_vif = cfg.ram_vif;
39     mon.ram_vif = cfg.ram_vif;
40     drv.seq_item_port.connect(sqr.seq_item_export);
41     mon.mon_ap.connect(agt_ap);
42 endfunction
43 endclass
44 endpackage
```

Config Object, Sequence Item and Sequencer:

```
1 package ram_cfg_obj_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4
5 class ram_cfg_obj extends uvm_object;
6     `uvm_object_utils(ram_cfg_obj)
7     virtual ram_if ram_vif;
8
9     function new(string name = "cfg_obj");
10    super.new(name);
11    endfunction
12 endclass
13 endpackage
```

```
ram_seq_item.sv > {} ram_seq_item_pkg > ram_seq_item
1 package ram_seq_item_pkg;
2 import uvm_pkg::*;
3 import shared_pkg::*;
4 `include "uvm_macros.svh"
5
6 class ram_seq_item extends uvm_sequence_item;
7     `uvm_object_utils(ram_seq_item)
8
9     rand bit rst_n, rx_valid;
10    | bit tx_valid, tx_valid_ref;
11    rand bit [9:0] din;
12    | bit [7:0] dout, dout_ref;
13    | op_e old_operation;
14
15    function new(string name = "item");
16        super.new(name);
17    endfunction
18
19    constraint rst_rx_constraint
20    {
21        // RAM_RESET
22        rst_n dist {0:= 2, 1:= 98};
23        // RAM_RX_VALID
24        rx_valid dist {0:= 2, 1:= 98};
25    }
26
27    constraint wr_only_constraint
28    {
29        // Always constrain to write operations (address or data)
30        // RAM_WR
31        din[9:8] inside {WR_ADDR, WR_DATA};
32    }
33
```

```

33
34     constraint rd_only_constraint
35     {
36         // RAM RD
37         if(old_operation == RD_ADDR)
38         {
39             din[9:8] == RD_DATA;
40         }
41         else if (old_operation == RD_DATA)
42         {
43             din[9:8] == RD_ADDR;
44         }
45         else
46         {
47             din[9:8] inside {RD_ADDR, RD_DATA};
48         }
49     }
50 }
51
52     constraint rd_wr_constraint
53     {
54         if(old_operation == WR_ADDR) // wr_addr
55         {
56             din[9:8] inside {WR_ADDR, WR_DATA};
57         }
58         else if (old_operation == WR_DATA) // wr_data
59         {
60             din[9:8] dist {RD_ADDR:= 60, WR_ADDR:= 40}; // read_address = 60%, write_address = 40%
61         }
62         else if (old_operation == RD_ADDR) // rd_addr
63         {
64             din[9:8] inside {RD_ADDR, RD_DATA};
65         }
66         else // rd_data
67         {
68             din[9:8] dist {WR_ADDR:= 60, RD_ADDR:= 40};
69         }
70     }
71 }
72
73     function void post_randomize();
74         old_operation = op_e'(din[9:8]);
75     endfunction
76
77     function string convert2string();
78         return $sformatf("%s rst_n = %0b rx_valid = %0b tx_valid = %0b din = %b dout = %b", super.convert2string(),
79         rst_n, rx_valid, tx_valid, din, dout);
80     endfunction
81 endclass
82 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:

```
1 package ram_reset_seq_pkg;
2 import uvm_pkg::*;
3 import ram_seq_item_pkg::*;
4 `include "uvm_macros.svh"
5
6 class ram_reset_seq extends uvm_sequence #(ram_seq_item);
7     `uvm_object_utils(ram_reset_seq)
8     ram_seq_item item;
9
10    function new(string name = "item");
11        super.new(name);
12    endfunction
13
14    task body();
15        // Assert reset
16        item = ram_seq_item::type_id::create("item");
17        start_item(item);
18        item.rst_n = 0;
19        item.rx_valid = 0;
20        item.tx_valid = 0;
21        item.din = 0;
22        finish_item(item);
23
24        // Hold reset for 5 cycles
25        repeat(5) begin
26            start_item(item);
27            item.rst_n = 0;
28            item.rx_valid = 0;
29            item.tx_valid = 0;
30            item.din = 0;
31            finish_item(item);
32        end
33
34        // Deassert reset
35        start_item(item);
36        item.rst_n = 1;
37        item.rx_valid = 0;
38        item.tx_valid = 0;
39        item.din = 0;
40        finish_item(item);
41    endtask
42
43 endclass
44 endpackage
```

```
④ 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
```

```
W:\ram_wr_only_seq.vhd 100% ram_wr_only_seq_pkg 100% ram_wr_only_seq
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:

```
❶ ram_monitor.sv > ...
1 package ram_monitor_pkg;
2 import uvm_pkg::*;
3 import ram_seq_item_pkg::*;
4 `include "uvm_macros.svh"
5
6 class ram_monitor extends uvm_monitor;
7     `uvm_component_utils(ram_monitor)
8     ram_seq_item item;
9     virtual ram_if ram_vif;
10    uvm_analysis_port #(ram_seq_item) mon_ap;
11
12    function new(string name = "mtr", uvm_component parent = null);
13        super.new(name, parent);
14    endfunction
15
16    function void build_phase(uvm_phase phase);
17        super.build_phase(phase);
18        mon_ap = new("mon_ap", this);
19    endfunction
20
21    task run_phase(uvm_phase phase);
22        super.run_phase(phase);
23        forever begin
24            item = ram_seq_item::type_id::create("item");
25            @(negedge ram_vif.clk);
26            item.rst_n = ram_vif.rst_n;
27            item.rx_valid = ram_vif.rx_valid ;
28            item.din = ram_vif.din ;
29            item.tx_valid = ram_vif.tx_valid;
30            item.tx_valid_ref = ram_vif.tx_valid_ref;
31            item.dout = ram_vif.dout;
32            item.dout_ref = ram_vif.dout_ref;
33            mon_ap.write(item);
34            `uvm_info("run_phase", item.convert2string_stimulus(), UVM_HIGH);
35        end
36    endtask
37 endclass
38 endpackage
```

```
RAM_driver.sv > ...
1 package ram_driver_pkg;
2 import uvm_pkg::*;
3 import ram_seq_item_pkg::*;
4 `include "uvm_macros.svh"
5
6 class ram_driver extends uvm_driver #(ram_seq_item);
7     `uvm_component_utils(ram_driver)
8     ram_seq_item item;
9     virtual ram_if ram_vif;
10
11     function new(string name = "drv", uvm_component parent = null);
12         super.new(name, parent);
13     endfunction
14
15     task run_phase(uvm_phase phase);
16         super.run_phase(phase);
17         forever begin
18             item = ram_seq_item::type_id::create("item");
19             seq_item_port.get_next_item(item);
20             ram_vif.rst_n = item.rst_n;
21             ram_vif.rx_valid = item.rx_valid;
22             ram_vif.din = item.din;
23             @(negedge ram_vif.clk);
24             seq_item_port.item_done();
25             `uvm_info("run_phase", item.convert2string_stimulus(), UVM_HIGH);
26         end
27     endtask
28 endclass
29 endpackage
```

Interface:

```
1  interface ram_if(clk);
2      parameter ADDR_SIZE = 8, MEM_DEPTH = 256;
3      input bit clk;
4      logic rst_n, rx_valid, tx_valid, tx_valid_ref;
5      logic [ADDR_SIZE+1:0] din; // default word size + extra 2 bits
6      logic [ADDR_SIZE-1:0] dout, dout_ref;
7
8  endinterface
```

Coverage:

```
1 package ram_coverage_pkg;
2 import uvm_pkg::*;
3 import ram_seq_item_pkg::*;
4 `include "uvm_macros.svh"
5
6 class ram_coverage extends uvm_component;
7     `uvm_component_utils(ram_coverage)
8     uvm_analysis_export #(ram_seq_item) cov_export;
9     uvm_tlm_analysis_fifo #(ram_seq_item) cov_fifo;
10    ram_seq_item item;
11
12    covergroup ram_cvr_gp;
13        transaction_ordering_cp: coverpoint item.din[9:8]
14    {
15        bins all_values = {[0:3]};
16        bins wr_data_after_wr_address = (0 => 1);
17        bins rd_data_after_rd_address = (2 => 3);
18        bins full_transition = (0 => 1 => 2 => 3);
19    }
20
21    rx_valid_cp: coverpoint item.rx_valid
22    {
23        bins rx_high = {1};
24    }
25    tx_valid_cp: coverpoint item.tx_valid
26    {
27        bins tx_high = {1};
28    }
29
30    cross_op_rx_cp: cross transaction_ordering_cp, rx_valid_cp;
31    cross_op_tx_cp: cross transaction_ordering_cp, tx_valid_cp
32    {
33        // Cross bin for read data (3) when tx_valid is high (1)
34        bins rd_data_tx_high = binsof(transaction_ordering_cp.all_values) intersect {3} &&
35        | | | | | binsof(tx_valid_cp) intersect {1};
36        option.cross_auto_bin_max = 0;
37    }
38
39    endgroup
40
```

```
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:

```
1 package ram_scoreboard_pkg;
2 import uvm_pkg::*;
3 import ram_seq_item_pkg::*;
4 `include "uvm_macros.svh"
5
6 class ram_scoreboard extends uvm_scoreboard;
7   `uvm_component_utils(ram_scoreboard)
8   ram_seq_item item;
9   uvm_analysis_export #(ram_seq_item) sb_export;
10  uvm_tlm_analysis_fifo #(ram_seq_item) sb_fifo;
11  int error_count = 0, correct_count = 0;
12
13 function new(string name = "sb", uvm_component parent = null);
14   super.new(name, parent);
15 endfunction
16
17 function void build_phase(uvm_phase phase);
18   super.build_phase(phase);
19   sb_export = new("sb_export", this);
20   sb_fifo = new("sb_fifo", this);
21 endfunction
22
23 function void connect_phase(uvm_phase phase);
24   super.connect_phase(phase);
25   sb_export.connect(sb_fifo.analysis_export);
26 endfunction
27
28 task run_phase(uvm_phase phase);
29   super.run_phase(phase);
30   forever begin
31     sb_fifo.get(item);
32     if(item.dout != item.dout_ref || item.tx_valid != item.tx_valid_ref) begin
33       $display("Error, transaction sent by dut is %s while dout_ref = %b and tx_ref = %b", item.convert2string_stimulus(), item.dout_ref, item.tx_valid_ref);
34       error_count++;
35     end
36     else begin
37       correct_count++;
38     end
39   end
40 endtask
41 endclass
42 endpackage
```

Reference Model:

```
1  module ram_ref(clk, rst_n, din, rx_valid, dout, tx_valid);
2  input clk, rst_n, rx_valid;
3  input [9:0] din;
4  output reg tx_valid;
5  output reg [7:0] dout;
6
7  parameter MEM_DEPTH = 256, ADDR_SIZE = 8;
8
9  reg [ADDR_SIZE-1 :0] mem [MEM_DEPTH-1 :0];
10
11 reg [ADDR_SIZE - 1 : 0] wr_address, rd_address;
12 always @ (posedge clk) begin
13     if (!rst_n) begin
14         dout <= 0;
15         wr_address <= 0;
16         rd_address <= 0;
17         tx_valid <= 0;
18     end
19     else begin
20         if (rx_valid) begin
21             case ({din[9:8]})
22                 2'b00: begin
23                     wr_address <= din[7:0];
24                 end
25                 2'b01: begin
26                     mem[wr_address] <= din[7:0];
27                 end
28                 2'b10: begin
29                     rd_address <= din[7:0];
30                 end
31                 2'b11: begin
32                     dout <= mem[rd_address];
33                 end
34             endcase
35         end
36         tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 : 1'b0;
37     end
38 end
39 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:

```
shared_pkg.sv / ...
1 package shared_pkg;
2     typedef enum bit [1:0] {WR_ADDR, WR_DATA, RD_ADDR, RD_DATA } op_e;
3 endpackage
4
```

Do File:

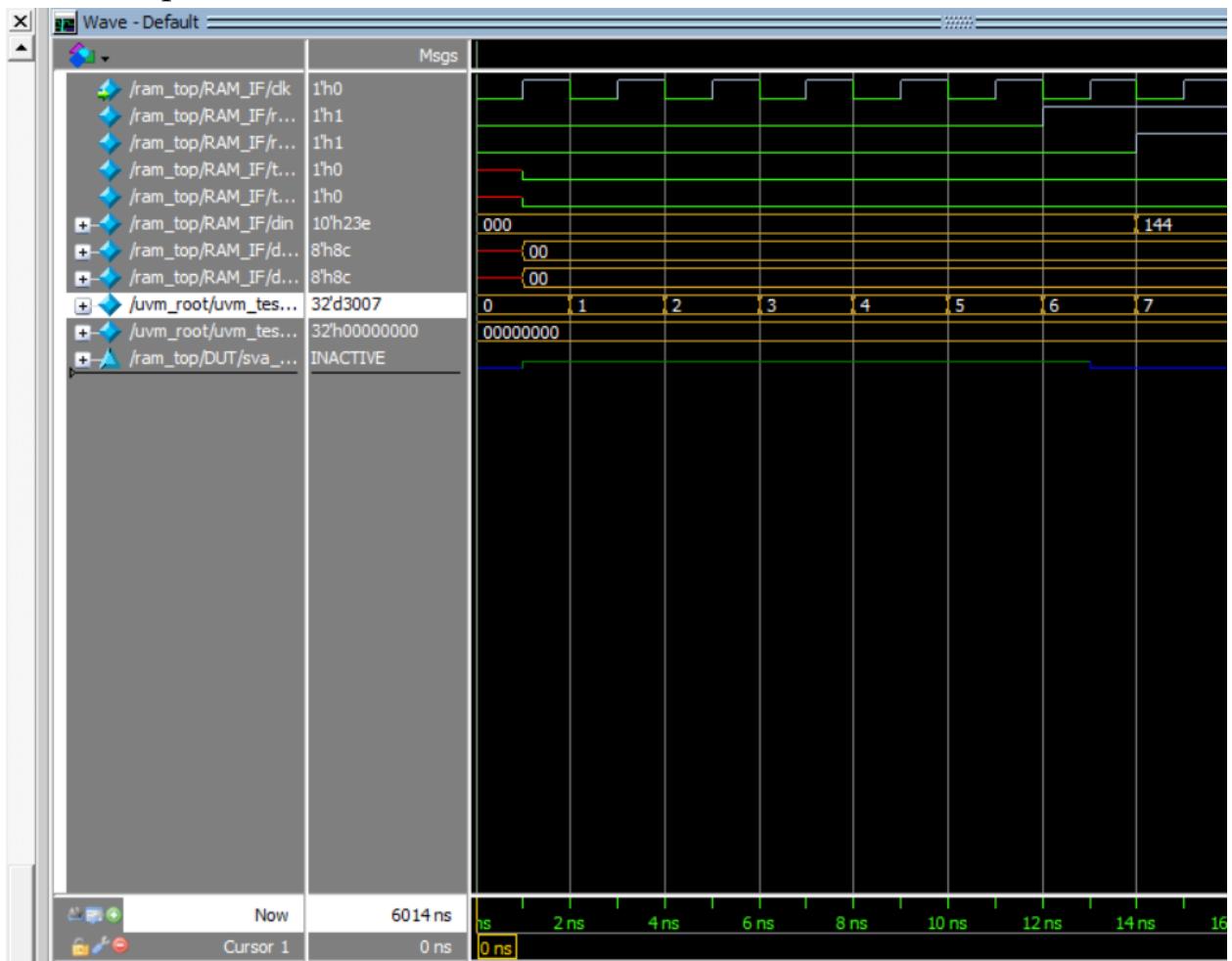
```
1 vlib work
2 vlog -f src_files.list +cover
3 vsim -voptargs=+acc work.ram_top -classdebug -uvmcontrol=all -cover
4 run 0
5 add wave /ram_top/RAM_IF/*
6 add wave -position insertpoint \
7 sim:/uvm_root/uvm_test_top/env/sb/correct_count \
8 sim:/uvm_root/uvm_test_top/env/sb/error_count
9 add wave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

Assertions:

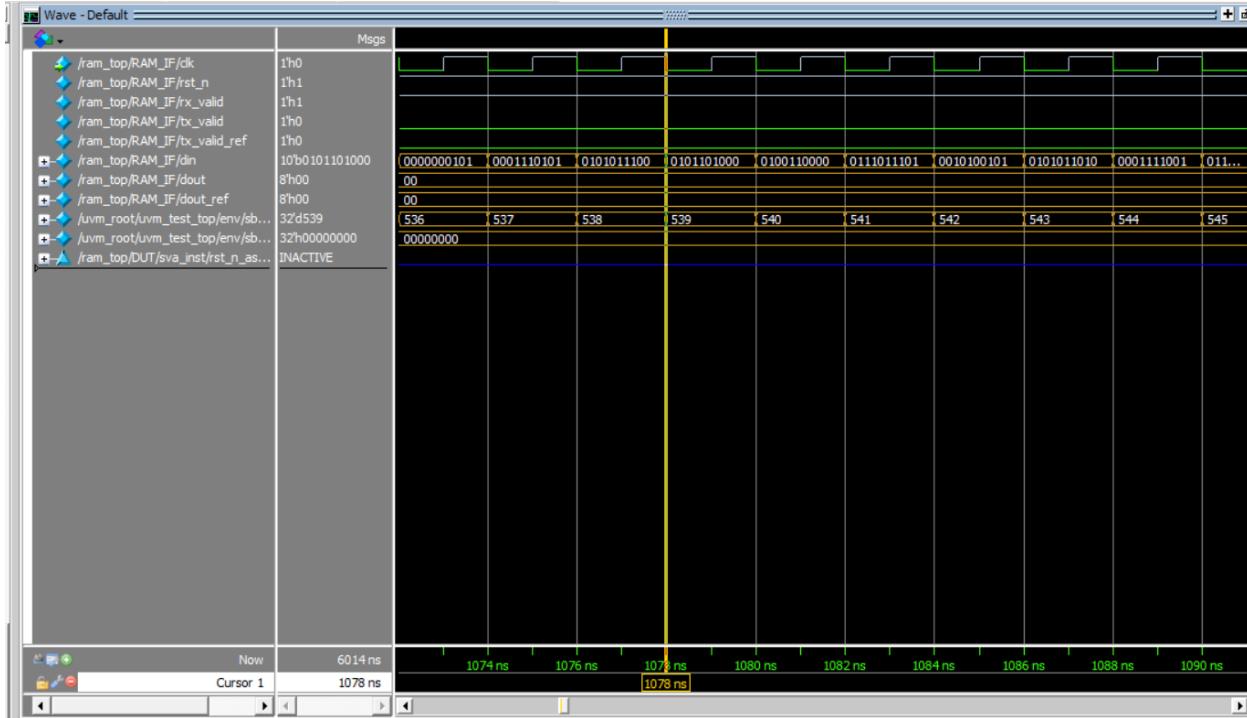
```
1 module ram_sva(input [9:0] din, input clk,rst_n,rx_valid, input [7:0] dout, input tx_valid);
2     rst_n_assertion: assert property (@(posedge clk) (!rst_n) |>> (dout == 0) && (tx_valid == 0));
3     tx_valid_0_assertion: assert property (@(posedge clk) disable iff(!rst_n) [(!din[8] && din[9])] |>> (tx_valid == 0));
4     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]);
5     wr_addr_then_wr_data_assertion: assert property (@(posedge clk) disable iff(!rst_n) ((!din[8] && !din[9])) |>> (din[8] && !din[9])[-1]);
6     rd_addr_then_rd_data_assertion: assert property (@(posedge clk) disable iff(!rst_n) ((!din[8] && din[9])) |>> (din[8] && din[9])[-1]);
7
8 endmodule
```

Waveforms:

Reset Sequence:



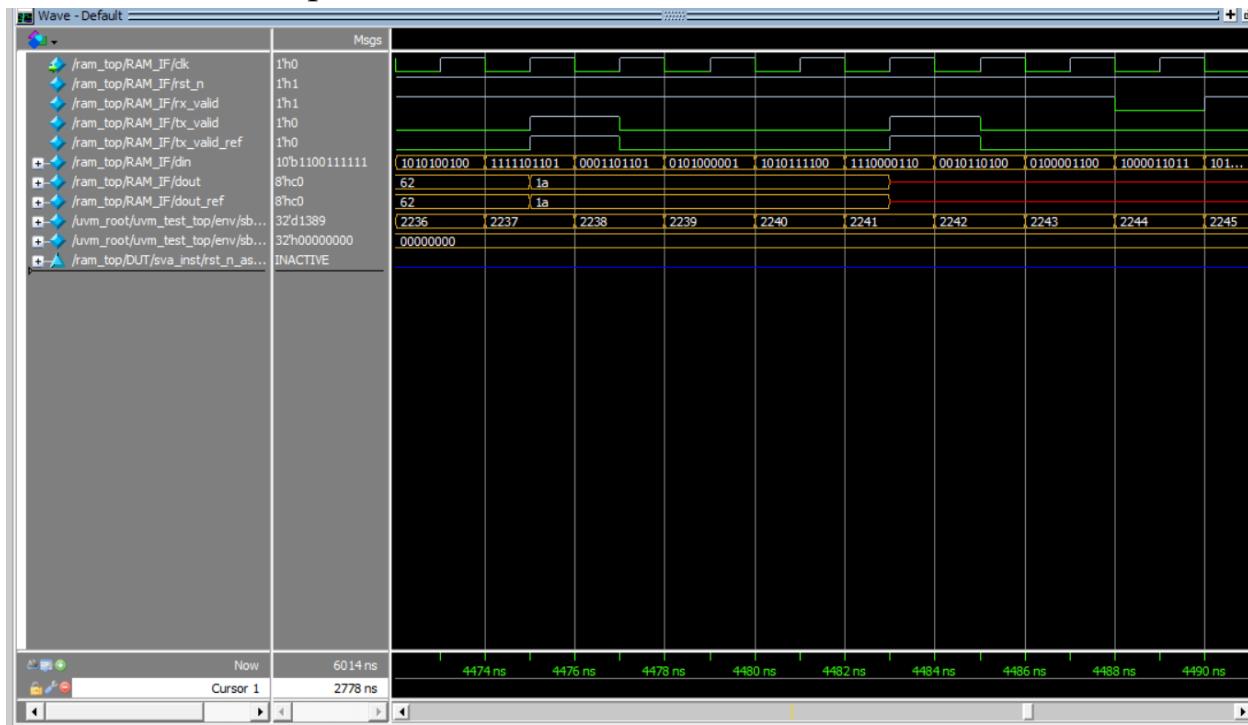
Write-Only Sequence:



Read-Only Sequence:



Write & Read Sequence:



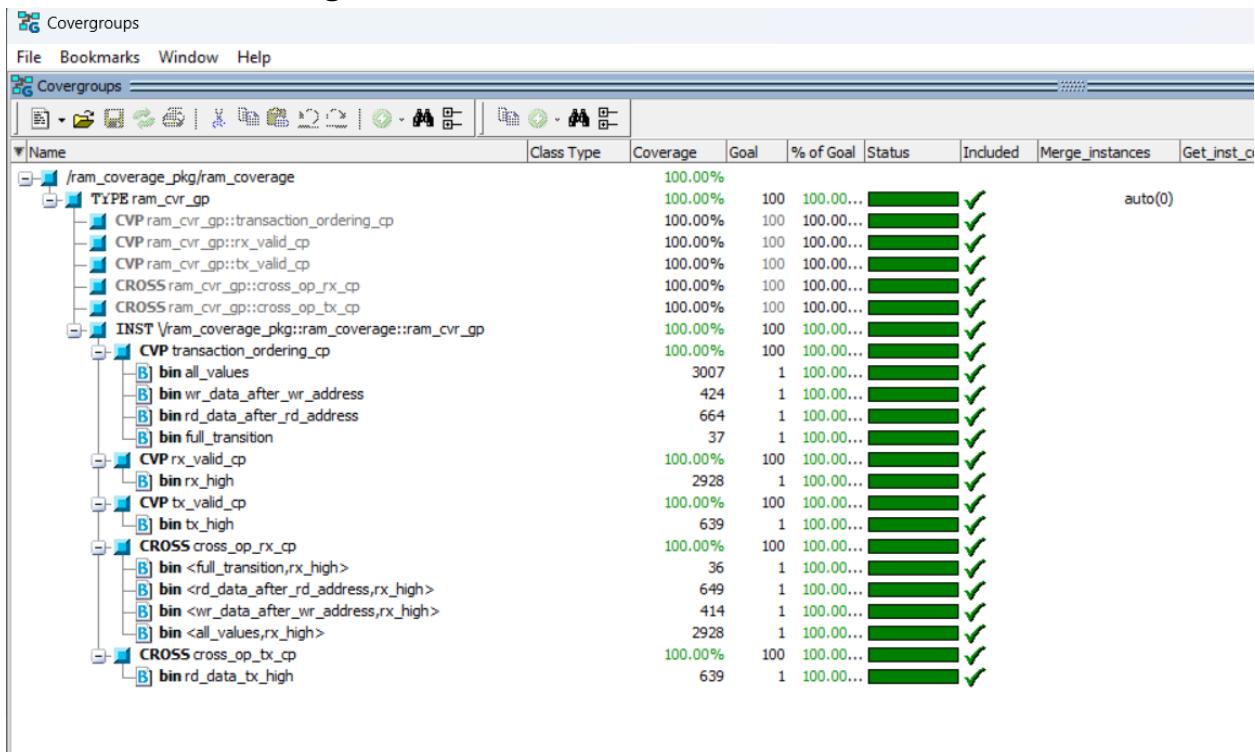
Transcript:

```

# 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] questauvm::init(all)
# UVM_INFO @ 0: reporter [RNIST] Running test ram_test...
# UVM_INFO ram_test.sv(46) @ 0: uvm_test_top [run_phase] rst signal asserted
# **** 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 Transaction Recording Turned OFF.
# 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
# [RNIST] 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:

The screenshot shows an assertions coverage analysis interface. It lists a series of assertions, each with its name, assertion type (Immediate or Concurrent), inclusion status, language (SVA), enable status, failure count, pass count, active count, memory usage, and peak memory usage. Most assertions are marked as included (green checkmark) and enabled (on). Some assertions have failure counts of 0, while others have counts of 1. Memory usage is indicated as 0B for most, except for a few which show 0B.

Name	Assertion Type	Included	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/lm...	Immediate	✗	SVA	on	0	0	-	-	-
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/lm...	Immediate	✗	SVA	on	0	0	-	-	-
/ram_wr_rd_seq_pkg::ram_wr_rd_seq::body/#ublk#131690007...	Immediate	✓	SVA	on	0	1	-	-	-
/ram_rd_only_seq_pkg::ram_rd_only_seq::body/#ublk#1205336...	Immediate	✓	SVA	on	0	1	-	-	-
/ram_wr_only_seq_pkg::ram_wr_only_seq::body/#ublk#120535...	Immediate	✓	SVA	on	0	1	-	-	-
+ /ram_top/DUT/sva_inst/rst_n_assertion	Concurrent	✓	SVA	on	0	1	-	0B	0B
+ /ram_top/DUT/sva_inst/bx_valid_0_assertion	Concurrent	✓	SVA	on	0	1	-	0B	0B
+ /ram_top/DUT/sva_inst/bx_valid_1_assertion	Concurrent	✓	SVA	on	0	1	-	0B	0B
+ /ram_top/DUT/sva_inst/wr_addr_then_wr_data_assertion	Concurrent	✓	SVA	on	0	1	-	0B	0B
+ /ram_top/DUT/sva_inst/rd_addr_then_rd_data_assertion	Concurrent	✓	SVA	on	0	1	-	0B	0B

Code Coverage:

```
cvr_rpt.txt
1  Coverage Report by instance with details
2
3  -----
4  --- Instance: /ram_top/RAM_IF
5  --- Design Unit: work.ram_if
6  -----
7  Toggle Coverage:
8    Enabled Coverage      Bins   Hits   Misses  Coverage
9    -----              ---   ---   ---   -----
10   Toggles                62     62      0  100.00%
11
12  =====Toggle Details=====
13
14  Toggle Coverage for instance /ram_top/RAM_IF --
15
16
17
18
19
20
21
22
23
24
25
26
27  Total Node Count      =      31
28  Toggled Node Count   =      31
29  Untoggled Node Count =      0
30
31  Toggle Coverage       =  100.00% (62 of 62 bins)
32
33  =====
```

```

52 -----
53 =====
54 === Instance: /ram_top/DUT/sva_inst
55 === Design Unit: work.ram_sva
56 -----
57
58 < Assertion Coverage:
59 | Assertions      5      5      0  100.00%
60 -----
61 < Name          File(Line)      Failure Count      Pass Count
62 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
63 -----
64 < /ram_top/DUT/sva_inst/rst_n_assertion
65 | | | | | ram_sva.sv(2)      0      1
66 < /ram_top/DUT/sva_inst/tx_valid_0_assertion
67 | | | | ram_sva.sv(3)      0      1
68 < /ram_top/DUT/sva_inst/tx_valid_1_assertion
69 | | | | ram_sva.sv(4)      0      1
70 < /ram_top/DUT/sva_inst/wr_addr_then_wr_data_assertion
71 | | | | ram_sva.sv(5)      0      1
72 < /ram_top/DUT/sva_inst/rd_addr_then_rd_data_assertion
73 | | | | ram_sva.sv(6)      0      1
74 < Toggle Coverage:
75 | Enabled Coverage      Bins      Hits      Misses      Coverage
76 | -----      -----      -----      -----      -----
77 | Toggles           44        44        0  100.00%
78
79 =====Toggle Details=====

```

```

58
59 =====Toggle Details=====
60
61 Toggle Coverage for instance /ram_top/DUT/sva_inst --
62
63 | | | | | Node 1H->0L 0L->1H "Coverage"
64 | | | | |
65 | | | | | clk 1 1 100.00
66 | | | | | din[0-9] 1 1 100.00
67 | | | | | dout[0-7] 1 1 100.00
68 | | | | | rst_n 1 1 100.00
69 | | | | | rx_valid 1 1 100.00
70 | | | | | tx_valid 1 1 100.00
71
72 Total Node Count = 22
73 Toggled Node Count = 22
74 Untoggled Node Count = 0
75
76 Toggle Coverage = 100.00% (44 of 44 bins)
77
78 =====
79 === Instance: /ram_top/DUT
80 === Design Unit: work.RAM
81 =====
82 Branch Coverage:
83 | Enabled Coverage Bins Hits Misses Coverage
84 | ----- --- --- -----
85 | Branches 8 8 0 100.00%
86
87 =====Branch Details=====

```

```

120
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=====

```

```

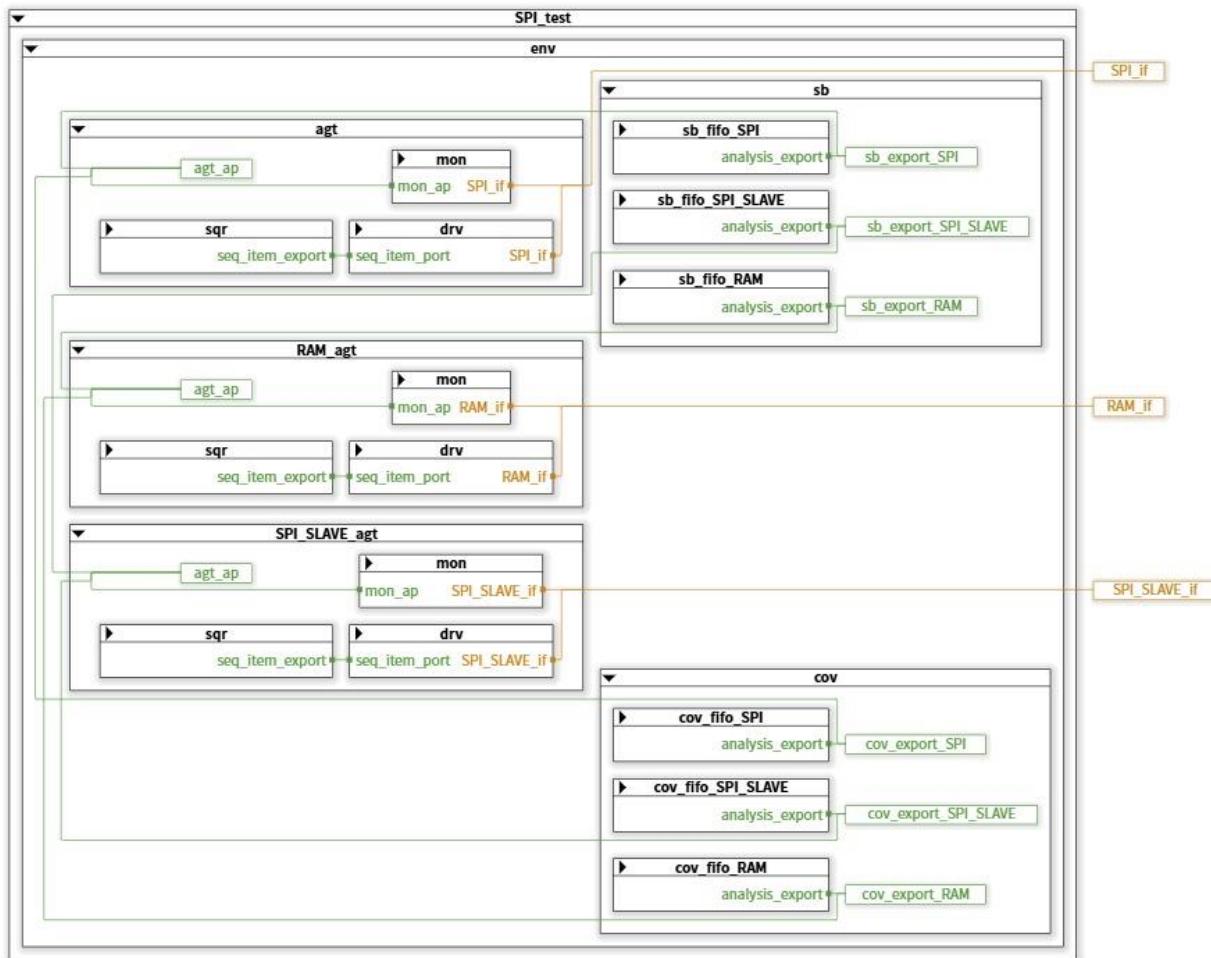
231
232 =====Toggle Details=====
233
234 ▼ Toggle Coverage for instance /ram_top/DUT --
235
236
237
238 | Node | 1H->0L | 0L->1H | "Coverage"
239 |-----|
240 | Rd_Addr[7-0] | 1 | 1 | 100.00
241 | Wr_Addr[7-0] | 1 | 1 | 100.00
242 | clk | 1 | 1 | 100.00
243 | din[0-9] | 1 | 1 | 100.00
244 | dout[7-0] | 1 | 1 | 100.00
245 | rst_n | 1 | 1 | 100.00
246 | rx_valid | 1 | 1 | 100.00
247 | tx_valid | 1 | 1 | 100.00
248
249 Total Node Count = 38
250 Toggled Node Count = 38
251 Untoggled Node Count = 0
252
253 Toggle Coverage = 100.00% (76 of 76 bins)
254
255 =====

```

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.	<pre>@(posedge clk) !rst_n => !MISO;</pre>
When din[9:8] = 11 and rx_valid is asserted, tx_valid should be high and should eventually fall.	<pre>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</pre>

3.3. Codes:

```

1 module SPI_Wrapper (input clk,rst_n,SS_n,MOSI, output MISO);
2 parameter MEM_DEPTH=256, ADDR_SIZE=8;
3 wire [ADDR_SIZE+1:0] rx_data_din;
4 wire [ADDR_SIZE-1:0] tx_data_dout;
5 wire rx_valid,tx_valid;
6
7 /*module SPI_SLAVE (.MOSI,SS_n,clk,rst_n,rx_data,tx_data,MISO,rx_valid);*/
8
9 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));
10
11 /*module RAM (din,clk,rst_n,rx_valid,dout,tx_valid);*/
12
13 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));
14
15 endmodule

```

Figure 1: SPI Wrapper RTL Design

```

1 module SPI_Wrapper_GOLDEN (input clk,rst_n,SS_n,MOSI, output MISO);
2 parameter MEM_DEPTH=256, ADDR_SIZE=8;
3 wire [ADDR_SIZE+1:0] rx_data_din;
4 wire [ADDR_SIZE-1:0] tx_data_dout;
5 wire rx_valid,tx_valid;
6
7 /*module SPI_SLAVE_GOLDEN (.MOSI,SS_n,clk,rst_n,rx_data,tx_valid,tx_data,MISO,rx_valid);*/
8
9 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));
10
11 /*module RAM_GOLDEN (clk, rst_n, din, rx_valid, dout, tx_valid);*/
12
13 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));
14
15 endmodule

```

Figure 2: SPI Wrapper Golden Model RTL Design

```

1 module SPI_sva(input clk, rst_n, SS_n, MOSI, MISO);
2
3 //RESET
4 property RESET;
5 |@(posedge clk) !rst_n |=> !MISO;
6 endproperty
7
8 // Detect a READ_DATA command pattern
9 sequence READ_DATA_SEQUENCE;
10 |$fell(SS_n) ##1 (MOSI[~3] ###0 1'b1); // detect 3 bits + some end marker
11 endsequence
12
13 // Property: whenever SS_n is low, if READ_DATA_SEQUENCE does NOT start,
14 // then MISO must stay stable while SS_n is low.
15 property MISO_STABLE_NOT_READ;
16 |@(posedge clk) disable iff (!rst_n)
17 |$fell(SS_n) |=>
18 |(not READ_DATA_SEQUENCE ##1 ($stable(MISO) throughout (!SS_n)));
19 endproperty
20
21 // Assertions
22 RESET_assert: assert property (RESET);
23 READ_DATA_assert: assert property (MISO_STABLE_NOT_READ);
24
25 // Coverage
26 RESET_cover: cover property (RESET);
27 READ_DATA_cover: cover property (MISO_STABLE_NOT_READ);
28
29 endmodule

```

Figure 3: SPI Wrapper sva

```

1  interface RAM_if(clk);
2    parameter ADDR_SIZE = 8, MEM_DEPTH = 256;
3    input bit clk;
4    logic rst_n, rx_valid, tx_valid, tx_valid_ref;
5    logic [ADDR_SIZE+1:0] din; // default word size + extra 2 bits
6    logic [ADDR_SIZE-1:0] dout, dout_ref;
7
8  endinterface

```

Figure 4: RAM if

```

1  interface SPI_SLAVE_if (clk);
2    //Inputs
3    input bit clk;
4    logic MOSI, rst_n, SS_n, tx_valid;
5    logic [7:0] tx_data;
6    //Outputs
7    logic [9:0] rx_data,rx_data_ref;
8    logic      rx_valid, MISO, MISO_ref, rx_valid_ref;
9
10   modport DUT(input clk,MOSI,rst_n,SS_n,tx_data,tx_valid, output MISO,rx_data,rx_valid);
11
12  endinterface : SPI_SLAVE_if

```

Figure 5: SPI Slave if

```

1  interface SPI_if (clk);
2    //Parameters
3    parameter MEM_DEPTH=256;
4    parameter ADDR_SIZE=8;
5    //Inputs
6    input clk;
7    logic rst_n, SS_n, MOSI;
8    //Outputs
9    logic MISO;
10   //Outputs Golden
11   logic MISO_GOLDEN;
12  endinterface : SPI_if

```

Figure 6: SPI Wrapper if

```

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

```

Figure 7: RAM Package

```

1  package SPI_SLAVE_shared_pkg;
2  typedef enum {IDLE,CHK_CMD ,WRITE,READ_ADD , READ_DATA } current_st;
3  int counter_allcases =0 ;
4  int counter_read=0;
5  | logic SS_n_prev=1;
6  endpackage

```

Figure 8: SPI Slave Package

```

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

```

Figure 9: SPI Wrapper Package

```

1  package SPI_config_pkg;
2    |   import uvm_pkg::*;
3    |   `include "uvm_macros.svh"
4
5    |   class SPI_config extends uvm_object;
6      |     `uvm_object_utils(SPI_config)
7
8      |       virtual SPI_if SPI_vif;
9      |       virtual SPI_SLAVE_if SPI_SLAVE_vif;
10     |       virtual RAM_if RAM_vif;
11     |       uvm_active_passive_enum is_active;
12
13    |       function new(string name = "SPI_config");
14      |         super.new(name);
15    |       endfunction
16    endclass
17
18  endpackage

```

Figure 10: Config

```

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

```

Figure 11: SPI Wrapper Sequencer

```

1 package ram_seq_item_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import RAM_shared_pkg::*;
5
6   class ram_seq_item extends uvm_sequence_item;
7     `uvm_object_utils(ram_seq_item)
8     //Inputs
9     logic rst_n, rx_valid;
10    logic [9:0] din;
11    //Outputs
12    logic tx_valid, tx_valid_ref;
13    logic [7:0] dout, dout_ref;
14
15    function new(string name = "ram_seq_item");
16      super.new(name);
17      endfunction
18  endclass
19 endpackage

```

Figure 12: RAM Sequence Item

```

1 package spi_slave_seq_item_package;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import SPI_SLAVE_shared_pkg::*;
5
6   class spi_slave_seq_item extends uvm_sequence_item;
7     `uvm_object_utils(spi_slave_seq_item)
8
9     //Inputs
10    logic [7:0] tx_data;
11    logic MOSI, SS_n, rst_n, tx_valid;
12    //Outputs
13    logic [9:0] rx_data, rx_data_ref;
14    logic MISO, MISO_ref, rx_valid, rx_valid_ref;
15
16    function new(string name= "spi_slave_seq_item");
17      super.new(name);
18      endfunction
19
20  endclass
21 endpackage

```

Figure 13: SPI Slave Sequence Item

```

1 package SPI_seq_item_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import SPI_pkg::*;
5
6   class SPI_seq_item extends uvm_sequence_item;
7     `uvm_object_utils(SPI_seq_item)
8
9     //Inputs
10    rand logic rst_n, SS_n, MOSI;
11    //Outputs
12    logic MISO;
13    //Outputs Golden
14    logic MISO_GOLDEN;
15    //Array to help in MOST randomization
16    rand bit [0:10] array_rand;
17    //Saving the last operation
18    op_e old_operation;
19    //Array index to drive MOSI bit by bit
20    int bit_index = 0;
21
22
23    //Constraints
24    //RESET
25    constraint reset {rst_n dist {0:/2, 1:/98};}
26
27    //SS_n_high
28    constraint SS_n_high {
29      if ((array_rand[0:2] inside {WR_ADDR, WR_DATA, RD_ADDR}) && counter_allcases % 13 != 0)           //SS_n = 1 every 13 cycles
30      || (array_rand[0:2] inside {RD_DATA} && counter_read % 23 != 0)                                     //SS_n = 1 every 23 cycles
31      SS_n==0;
32      else
33      SS_n==1;
34    }
35
36    //Write only
37    constraint wr_only_constraint {
38      // Always constrain to write operations (address or data)
39      array_rand[0:2] inside {WR_ADDR, WR_DATA};
40    }
41
42    //Read only
43    constraint rd_only_constraint {
44      // Always constrain to read operations sequentially (address then data then address then ..etc)
45      if(old_operation == RD_ADDR)
46        array_rand[0:2] == RD_DATA;
47      else if (old_operation == RD_DATA)
48        array_rand[0:2] == RD_ADDR;
49      else
50        array_rand[0:2] inside {RD_ADDR, RD_DATA};
51    }
52
53    //Write and Read
54    constraint rd_wr_constraint {
55      if(old_operation == WR_ADDR) //Write_address
56        array_rand[0:2] inside {WR_ADDR, WR_DATA};          // write_address or write_data
57      else if (old_operation == RD_DATA) //write_data
58        array_rand[0:2] dist {RD_ADDR:/ 60, WR_ADDR:/ 40}; // read_address = 60%, write_address = 40%
59      else if (old_operation == RD_ADDR) // Read_address
60        array_rand[0:2] == RD_DATA; //read_data
61      else // Read_data
62        array_rand[0:2] dist {WR_ADDR:/ 60, RD_ADDR:/ 40}; // Write Address = 60%, Read Address = 40%
63    }
64
65
66    function void post_randomize();
67      // Start of a new frame when SS_n goes low
68      if (SS_n_prev && !SS_n)
69        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))
70
71      // While SS_n is low -> drive MOSI bit-by-bit
72      if (!SS_n) begin
73        MOSI = array_rand[bit_index];
74        bit_index++;
75        if (bit_index > 10)
76          bit_index = 0; // wrap after full frame
77      end
78
79      // update operation tracking
80      old_operation = op_e'(array_rand[0:2]);
81      SS_n_prev = SS_n;
82
83      //Only increment counter_allcases counter when valid
84      if (array_rand[0:2] inside {WR_ADDR, WR_DATA, RD_ADDR})
85        counter_allcases++;
86      if (counter_allcases == 13)
87        counter_allcases = 0;
88
89      //Only counter_read increment counter when valid
90      if (array_rand[0:2] inside {RD_DATA})
91        counter_read++;
92      if (counter_read == 23)
93        counter_read = 0;
94
95    endfunction
96
97    function new(string name = "SPI_seq_item");
98      super.new(name);
99    endfunction
100
101  endclass
102
103 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

```

1 package SPI_wr_only_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_wr_only_sequence extends uvm_sequence #(SPI_seq_item);
8     `uvm_object_utils(SPI_wr_only_sequence)
9
10    SPI_seq_item seq_item;
11
12    function new(string name = "SPI_wr_only_sequence");
13      super.new(name);
14    endfunction
15
16    task body();
17      seq_item = SPI_seq_item::type_id::create("seq_item");
18      //Write only
19      repeat(5000) begin
20        start_item(seq_item);
21        //At the beginning of the operation, enable the write sequence and randomization
22        if(counter_allcases == 0) begin
23          seq_item.array_rand.rand_mode(1);
24          seq_item.wr_only_constraint.constraint_mode(1);
25          seq_item.rd_only_constraint.constraint_mode(0);
26          seq_item.rd_wr_constraint.constraint_mode(0);
27        end
28        else begin
29          //Disable all constraints and randomization because we are already in an operation
30          seq_item.wr_only_constraint.constraint_mode(0);
31          seq_item.rd_only_constraint.constraint_mode(0);
32          seq_item.rd_wr_constraint.constraint_mode(0);
33          seq_item.array_rand.rand_mode(0);
34        end
35        assert (seq_item.randomize());
36        finish_item(seq_item);
37      end
38    endtask
39
40  endclass
41 endpackage

```

Figure 16: Write only Sequence

```

1 package SPI_rd_only_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_rd_only_sequence extends uvm_sequence #(SPI_seq_item);
8     `uvm_object_utils(SPI_rd_only_sequence)
9
10    SPI_seq_item seq_item;
11
12    function new(string name = "SPI_rd_only_sequence");
13      super.new(name);
14    endfunction
15
16    task body();
17      seq_item = SPI_seq_item::type_id::create("seq_item");
18      //Read only
19      repeat(5000) begin
20        start_item(seq_item);
21        //At the beginning of the operation, enable the read sequence and randomization
22        if(counter_allcases == 0 && (seq_item.array_rand[0:2] inside {WR_ADDR, WR_DATA, RD_ADDR})
23        || counter_read == 0 && (seq_item.array_rand[0:2] inside {RD_DATA})) begin
24          seq_item.wr_only_constraint.constraint_mode(0);
25          seq_item.rd_only_constraint.constraint_mode(1);
26          seq_item.rd_wr_constraint.constraint_mode(0);
27          seq_item.array_rand.rand_mode(1);
28        end
29      end
30      else begin
31        //Disable all constraints and randomization because we are already in an operation
32        seq_item.wr_only_constraint.constraint_mode(0);
33        seq_item.rd_only_constraint.constraint_mode(0);
34        seq_item.rd_wr_constraint.constraint_mode(0);
35        seq_item.array_rand.rand_mode(0);
36      end
37      assert (seq_item.randomize());
38      finish_item(seq_item);
39    end
40  endtask
41
42 endclass
43 endpackage

```

Figure 17: Read only Sequence

```

1 package SPI_wr_rd_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_wr_rd_sequence extends uvm_sequence #(SPI_seq_item);
8     `uvm_object_utils(SPI_wr_rd_sequence)
9
10    SPI_seq_item seq_item;
11
12    function new(string name = "SPI_wr_rd_sequence");
13      super.new(name);
14    endfunction
15
16    task body();
17      seq_item = SPI_seq_item::type_id::create("seq_item");
18      //Write and Read
19      repeat(5000) begin
20        start_item(seq_item);
21        //At the beginning of the operation, enable the read/write sequence and randomization
22        if(counter_allcases == 0 && (seq_item.array_rand[0:2] inside {WR_ADDR, WR_DATA, RD_ADDR})
23        || counter_read == 0 && (seq_item.array_rand[0:2] inside {RD_DATA})) begin
24          seq_item.wr_only_constraint.constraint_mode(0);
25          seq_item.rd_only_constraint.constraint_mode(0);
26          seq_item.rd_wr_constraint.constraint_mode(1);
27          seq_item.array_rand.rand_mode(1);
28        end
29        else begin
30          //Disable all constraints and randomization because we are already in an operation
31          seq_item.wr_only_constraint.constraint_mode(0);
32          seq_item.rd_only_constraint.constraint_mode(0);
33          seq_item.rd_wr_constraint.constraint_mode(0);
34          seq_item.array_rand.rand_mode(0);
35        end
36        assert (seq_item.randomize());
37        finish_item(seq_item);
38      end
39
40    endtask
41
42  endclass
43 endpackage

```

Figure 18: Write/Read Sequence

```

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

```

Figure 19: RAM Monitor

```

1 package spi_slave_monitor_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import spi_slave_seq_item_package::*;
5   import SPI_SLAVE_shared_pkg::*;
6
7   class spi_slave_monitor extends uvm_monitor;
8     `uvm_component_utils(spi_slave_monitor);
9
10    virtual SPI_SLAVE_if ss_vif;
11    spi_slave_seq_item seq_item;
12    uvm_analysis_port #(spi_slave_seq_item) mon_ap;
13
14    function new(string name ="spi_slave_monitor",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      mon_ap = new("mon_ap", this);
21    endfunction
22
23    task run_phase (uvm_phase phase);
24      super.run_phase(phase);
25      forever begin
26        seq_item=spi_slave_seq_item::type_id::create("seq_item");
27        @(negedge ss_vif.clk);
28        //input tx_data, MOSI, SS_n, rst_n, tx_valid, output rx_data, rx_data_ref, MISO, MISO_ref, rx_valid, rx_valid_ref
29        seq_item.rst_n      = ss_vif.rst_n;
30        seq_item.MOSI       = ss_vif.MOSI;
31        seq_item.SS_n       = ss_vif.SS_n;
32        seq_item.MISO       = ss_vif.MISO;
33        seq_item.MISO_ref   = ss_vif.MISO_ref;
34        seq_item.tx_valid   = ss_vif.tx_valid;
35        seq_item.tx_data    = ss_vif.tx_data;
36        seq_item.rx_valid   = ss_vif.rx_valid;
37        seq_item.rx_data    = ss_vif.rx_data;
38        seq_item.rx_valid_ref = ss_vif.rx_valid_ref;
39        seq_item.rx_data_ref = ss_vif.rx_data_ref;
40
41        mon_ap.write(seq_item);
42      end
43    endtask
44  endclass
45 endpackage

```

Figure 20: SPI Slave Monitor

```

1 package SPI_monitor_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import SPI_seq_item_pkg::*;
5   import SPI_pkg::*;
6
7   class SPI_monitor extends uvm_monitor;
8     `uvm_component_utils(SPI_monitor)
9
10    virtual SPI_if SPI_vif;
11    SPI_seq_item seq_item;
12    uvm_analysis_port #(SPI_seq_item) mon_ap;
13
14    function new(string name = "SPI_monitor", 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      mon_ap = new("mon_ap", this);
21    endfunction
22
23    task run_phase(uvm_phase phase);
24      super.run_phase(phase);
25      forever begin
26        seq_item = SPI_seq_item::type_id::create("seq_item");
27        //input rst_n, SS_n, MOSI, output MISO
28        @(negedge SPI_vif.clk);
29        seq_item.rst_n      = SPI_vif.rst_n;
30        seq_item.SS_n       = SPI_vif.SS_n;
31        seq_item.MOSI       = SPI_vif.MOSI;
32        seq_item.MISO       = SPI_vif.MISO;
33        seq_item.MISO_GOLDEN = SPI_vif.MISO_GOLDEN;
34        mon_ap.write(seq_item);
35      end
36    endtask
37
38  endclass
39
40 endpackage

```

Figure 21: SPI Wrapper Monitor

```

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

```

Figure 22: SPI Wrapper Driver

```

1 package SPI_scoreboard_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import SPI_seq_item_pkg::*;
5   import spi_slave_seq_item_package::*;
6   import ram_seq_item_pkg::*;
7   import SPI_SLAVE_shared_pkg::*;
8   import RAM_shared_pkg::*;
9
10 class SPI_scoreboard extends uvm_scoreboard;
11   `uvm_component_utils(SPI_scoreboard)
12
13   uvm_analysis_export #(SPI_seq_item)      sb_export_SPI;
14   uvm_analysis_export #(spi_slave_seq_item) sb_export_SPI_SLAVE;
15   uvm_analysis_export #(ram_seq_item)       sb_export_RAM;
16   uvm_tlm_analysis_fifo #(SPI_seq_item)    sb_fifo_SPI;
17   uvm_tlm_analysis_fifo #(spi_slave_seq_item) sb_fifo_SPI_SLAVE;
18   uvm_tlm_analysis_fifo #(ram_seq_item)    sb_fifo_RAM;
19   SPI_seq_item seq_item_SPI;
20   spi_slave_seq_item seq_item_SPI_SLAVE;
21   ram_seq_item seq_item_RAM;
22
23   int correct_counter = 0;
24   int error_counter_SPI = 0;
25   int error_counter_SPI_SLAVE = 0;
26   int error_counter_RAM = 0;
27
28   function new(string name = "SPI_scoreboard", uvm_component parent = null);
29     super.new(name, parent);
30   endfunction
31
32   function void build_phase(uvm_phase phase);
33     super.build_phase(phase);
34     sb_export_SPI = new("sb_export_SPI", this);
35     sb_export_SPI_SLAVE = new("sb_export_SPI_SLAVE", this);
36     sb_export_RAM = new("sb_export_RAM", this);
37     sb_fifo_SPI = new("sb_fifo_SPI", this);
38     sb_fifo_SPI_SLAVE = new("sb_fifo_SPI_SLAVE", this);
39     sb_fifo_RAM = new("sb_fifo_RAM", this);
40   endfunction
41
42   function void connect_phase(uvm_phase phase);
43     super.connect_phase(phase);
44     sb_export_SPI.connect(sb_fifo_SPI.analysis_export);
45     sb_export_SPI_SLAVE.connect(sb_fifo_SPI_SLAVE.analysis_export);
46     sb_export_RAM.connect(sb_fifo_RAM.analysis_export);
47   endfunction
48
49   task run_phase(uvm_phase phase);
50     super.run_phase(phase);
51     forever begin
52       sb_fifo_SPI.get(seq_item_SPI);
53       sb_fifo_SPI_SLAVE.get(seq_item_SPI_SLAVE);
54       sb_fifo_RAM.get(seq_item_RAM);
55
56       if (seq_item_SPI.MISO != seq_item_SPI.MISO_GOLDEN) begin
57         `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))
58         error_counter_SPI++;
59       end
56
57       //input tx_data, MOSI, SS_n, rst_n, tx_valid, output rx_data, rx_data_ref, MISO, MISO_ref, rx_valid, rx_valid_ref
58       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
59         `uvm_error("ERROR", $sformatf("Error in SPI_SLAVE"))
60         error_counter_SPI_SLAVE++;
61       end
62
63       //input rst_n, rx_valid, din, output tx_valid, tx_valid_ref, dout, dout_ref
64       else if ((seq_item_RAM.dout != seq_item_RAM.dout_ref) || (seq_item_RAM.tx_valid != seq_item_RAM.tx_valid_ref)) begin
65         `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",
66         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))
67         error_counter_RAM++;
68       end
69
70       else correct_counter++;
71     end
72   endtask
73
74   function void report_phase(uvm_phase phase);
75     super.report_phase(phase);
76     `uvm_info("Report Phase", $sformatf("Correct = %d", correct_counter), UVM_MEDIUM)
77     `uvm_info("Report Phase", $sformatf("SPI Wrapper Errors = %d", error_counter_SPI), UVM_MEDIUM)
78     `uvm_info("Report Phase", $sformatf("SPI_SLAVE Errors = %d", error_counter_SPI_SLAVE), UVM_MEDIUM)
79     `uvm_info("Report Phase", $sformatf("RAM Errors = %d", error_counter_RAM), UVM_MEDIUM)
80   endfunction
81
82 endclass
83
84 endpackage

```

Figure 23: Scoreboard

```

1 package SPI_coverage_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import SPI_seq_item_pkg::*;
5   import spi_Slave.seq_item_package::*;
6   import ram_seq_item_pkg::*;
7   import SPI_SLAVE_shared_pkg::*;
8   import RAM_shared_pkg::*;
9
10  class SPI_coverage extends uvm_scoreboard;
11    `uvm_component_utils(SPI_coverage)
12
13    uvm_analysis_export #(SPI_seq_item) cov_export_SPI;
14    uvm_analysis_export #(spi_slave_seq_item) cov_export_SPI_SLAVE;
15    uvm_tlm_analysis_fifo #(ram_seq_item) cov_export_RAM;
16    uvm_tlm_analysis_fifo #(SPI_seq_item) cov_fifo_SPI;
17    uvm_tlm_analysis_fifo #(spi_slave_seq_item) cov_fifo_SPI_SLAVE;
18    uvm_tlm_analysis_fifo #(ram_seq_item) cov_fifo_RAM;
19    SPI_seq_item seq_item_SPI;
20    spi_SLAVE_seq_item seq_item_SPI_SLAVE;
21    ram_seq_item seq_item_RAM;
22
23  //Covergroups
24  covergroup cvr_grp_SPI_SLAVE;
25    receiver_data: coverpoint seq_item_SPI_SLAVE.rx_data[9:8];
26
27    SS_n_aliases : coverpoint seq_item_SPI_SLAVE.SS_n {
28      bins trans_all_cases = {1 => 0[*11] =>1};
29    }
30
31    SS_n_read_data : coverpoint seq_item_SPI_SLAVE.SS_n {
32      bins trans_read= (1 => 0[*22] => 1) ;
33    }
34
35    SS_n : coverpoint seq_item_SPI_SLAVE.SS_n {bins SS_n_val={0};}
36
37    // bins trans_read_data = {1=>0[*23]} =>1 iff(seq_item_SPI_SLAVE.rx_data[9:8]==2'b11);bins SS_n_val={0};
38    mosi: covergroup seq_item_SPI_SLAVE.MOSI {
39      bins write_addr = {0>0...0>33};
40      bins write_data = {0>0...0>33};
41      bins read_addr = {1>0...1>0};
42      bins read_data = {1>0...1>1};
43      bins mosi_val_low={0};
44      bins mosi_val_high={1};
45    }
46
47    cross_mosi_SS_n : cross mosi,SS_n
48    {
49      option.cross_auto_bin_max=0;
50      bins SS_n_low_write_addr = binsof(SS_n.SS_n_val) && binsof(mosi.write_addr);
51      bins SS_n_low_write_data = binsof(SS_n.SS_n_val) && binsof(mosi.write_data);
52      bins SS_n_low_read_addr = binsof(SS_n.SS_n_val) && binsof(mosi.read_addr);
53      bins SS_n_low_read_data = binsof(SS_n.SS_n_val) && binsof(mosi.read_data);
54    }
55
56
57 endgroup
58
59 covergroup cvr_grp_RAM;
60   transaction_ordering_cp: coverpoint seq_item_RAM.din[9:8] {
61     bins all_values = {[0:3]};
62
63     // Bins for specific transitions (They take many cycles to occur, as we have long operations)
64     bins wr_data_after_wr_address = {0[*14] => 1[*10]};
65     bins rd_data_after_rd_address = {2[*13] => 3[*10]};
66     bins full_transition = {0[*14] => 1[*12] => 3 -> 2[*13] => 3[*10]};
67   }
68
69   rx_valid_cp: coverpoint seq_item_RAM.rx_valid {
70     bins rx_high = {1};
71   }
72
73   tx_valid_cp: coverpoint seq_item_RAM.tx_valid {
74     bins tx_high = {1};
75   }
76
77   //When the sequence bins are hit, cross coverage checks if they hit while rx_valid is high at the same time
78   cross_op_rx_tx_cp: cross transaction_ordering_cp, rx_valid_cp;
79
80   cross_op_tx_cp: cross transaction_ordering_cp, tx_valid_cp {
81     // Cross bin for read data () when tx.valid is high ()
82     bins rd_data_tx_high = binsof(transaction_ordering_cp.all_values) intersect {3} &&
83       binsof(tx_valid_cp) intersect {1};
84     option.cross_auto_bin_max = 0;
85   }
86
87 endgroup
88
89 function new(string name = "SPI_coverage", uvm_component parent = null);
90   super.new(name, parent);
91   //Create Covergroups
92   cvr_grp_SPI_SLAVE = new();
93   cvr_grp_RAM = new();
94
95   function void build_phase(uvm_phase phase);
96     super.build_phase(phase);
97     cov_export_SPI = new("cov_export_SPI", this);
98     cov_export_SPI_SLAVE = new("cov_export_SPI_SLAVE", this);
99     cov_export_RAM = new("cov_export_RAM", this);
100    cov_fifo_SPI = new("cov_fifo_SPI", this);
101    cov_fifo_SPI_SLAVE = new("cov_fifo_SPI_SLAVE", this);
102    cov_fifo_RAM = new("cov_fifo_RAM", this);
103  endfunction
104
105  function void connect_phase(uvm_phase phase);
106    super.connect_phase(phase);
107    cov_export_SPI.connect(cov_fifo_SPI.analysis_export);
108    cov_export_SPI_SLAVE.connect(cov_fifo_SPI_SLAVE.analysis_export);
109    cov_export_RAM.connect(cov_fifo_RAM.analysis_export);
110  endfunction
111
112  task run_phase(uvm_phase phase);
113    super.run_phase(phase);
114    forever begin
115      cov_fifo_SPI.get(seq_item_SPI);
116      cov_fifo_SPI_SLAVE.get(seq_item_SPI_SLAVE);
117      cov_fifo_RAM.get(seq_item_RAM);
118      //Covergroups sample
119      cvr_grp_SPI_SLAVE.sample();
120      cvr_grp_RAM.sample();
121    end
122  endtask
123
124 endclass
125
126 endpackage

```

Figure 24: Coverage

```

1 package ram_agent_pkg;
2 import uvm_pkg::*;
3 import ram_seq_item_pkg::*;
4 import ram_sqr_pkg::*;
5 import ram_driver_pkg::*;
6 import ram_monitor_pkg::*;
7 import SPI_config_pkg::*;
8 `include "uvm_macros.svh"
9
10 class ram_agent extends uvm_agent;
11     `uvm_component_utils(ram_agent)
12
13     SPI_config cfg;
14     ram_driver drv;
15     ram_sqr sqr;
16     ram_monitor mon;
17     uvm_analysis_port #(ram_seq_item) agt_ap;
18
19     function new(string name = "ram_agent", uvm_component parent = null);
20         super.new(name, parent);
21     endfunction
22
23     function void build_phase(uvm_phase phase);
24         super.build_phase(phase);
25         if (!uvm_config_db #(SPI_config)::get(this, "", "CONFIG_RAM", cfg))
26             `uvm_fatal("build_phase", "unable to get configuration object for RAM agent")
27
28         if (cfg.is_active == UVM_ACTIVE) begin
29             sqr = ram_sqr::type_id::create("sqr", this);
30             drv = ram_driver::type_id::create("drv", this);
31         end
32         mon = ram_monitor::type_id::create("mon", this);
33         agt_ap = new("agt_ap", this);
34     endfunction
35
36     function void connect_phase(uvm_phase phase);
37         super.connect_phase(phase);
38         if (cfg.is_active == UVM_ACTIVE) begin
39             drv.ram_vif = cfg.RAM_vif;
40             drv.seq_item_port.connect(sqr.seq_item_export);
41         end
42         mon.ram_vif = cfg.RAM_vif;
43         mon.mon_ap.connect(agt_ap);
44     endfunction
45 endclass
46 endpackage

```

Figure 25: RAM Agent

```

1 package spi_slave_agent_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import spi_slave_seq_item_package::*;
5   import SPI_config_pkg::*;
6   import spi_slave_driver_pkg::*;
7   import spi_slave_monitor_pkg::*;
8   import spi_slave_sequencer_package::*;
9
10  class spi_slave_agent extends uvm_agent;
11    `uvm_component_utils(spi_slave_agent)
12
13    spi_slave_seq_item seq_item;
14    SPI_config         s_cfg;
15    spi_slave_driver   drv;
16    spi_slave_sequencer sqr;
17    spi_slave_monitor  mon;
18    uvm_analysis_port #(spi_slave_seq_item) agt_ap;
19
20    function new(string name = "spi_slave_agent", uvm_component parent = null);
21      super.new(name, parent);
22    endfunction
23
24    function void build_phase(uvm_phase phase);
25      super.build_phase(phase);
26      if (!uvm_config_db#(SPI_config)::get(this, "", "CONFIG_SPI_SLAVE", s_cfg))
27        `uvm_fatal("build_phase", "unable to get configuration object for SPI_SLAVE agent")
28
29      if (s_cfg.is_active == UVM_ACTIVE) begin
30        sqr = spi_slave_sequencer::type_id::create("sqr", this);
31        drv = spi_slave_driver::type_id::create("drv", this);
32      end
33      mon = spi_slave_monitor::type_id::create("mon", this);
34      agt_ap = new("agt_tb",this);
35    endfunction
36
37    function void connect_phase(uvm_phase phase);
38      super.connect_phase(phase);
39      if (s_cfg.is_active == UVM_ACTIVE) begin
40        drv.ss_vif = s_cfg.SPI_SLAVE_vif;
41        drv.seq_item_port.connect(sqr.seq_item_export);
42      end
43      mon.ss_vif = s_cfg.SPI_SLAVE_vif;
44      mon.mon_ap.connect(agt_ap);
45    endfunction
46  endclass
47 endpackage

```

Figure 26: SPI Slave Agent

```

1 package SPI_agent_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import SPI_driver_pkg::*;
5   import SPI_monitor_pkg::*;
6   import SPI_sequencer_pkg::*;
7   import SPI_config_pkg::*;
8   import SPI_seq_item_pkg::*;

10  class SPI_agent extends uvm_agent;
11    `uvm_component_utils(SPI_agent)
12
13    SPI_sequencer sqr;
14    SPI_driver drv;
15    SPI_monitor mon;
16    SPI_config SPI_cfg;
17    uvm_analysis_port #(SPI_seq_item) agt_ap;
18
19    function new(string name = "SPI_agent", uvm_component parent = null);
20      super.new(name, parent);
21    endfunction
22
23    function void build_phase(uvm_phase phase);
24      super.build_phase(phase);
25      if (!uvm_config_db #(SPI_config)::get(this, "", "CONFIG", SPI_cfg))
26        `uvm_fatal("Build_Phase", "Error in Wrapper Interface in Agent");
27
28      if (SPI_cfg.is_active == UVM_ACTIVE) begin
29        sqr = SPI_sequencer::type_id::create("sqr", this);
30        drv = SPI_driver::type_id::create("drv", this);
31      end
32      mon = SPI_monitor::type_id::create("mon", this);
33      agt_ap = new("agt_ap", this);
34
35    endfunction
36
37    function void connect_phase(uvm_phase phase);
38      super.connect_phase(phase);
39      if (SPI_cfg.is_active == UVM_ACTIVE) begin
40        drv.SPI_vif = SPI_cfg.SPI_vif;
41        drv.seq_item_port.connect(sqr.seq_item_export);
42      end
43      mon.SPI_vif = SPI_cfg.SPI_vif;
44      mon.mon_ap.connect(agt_ap);
45    endfunction
46
47  endclass
48
49 endpackage

```

Figure 27: SPI Wrapper Agent

```

1 package SPI_env_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import SPI_agent_pkg::*;
5   import spi_slave_agent_pkg::*;
6   import ram_agent_pkg::*;
7   import SPI_scoreboard_pkg::*;
8   import SPI_coverage_pkg::*;
9
10  class SPI_env extends uvm_env;
11    `uvm_component_utils(SPI_env)
12
13    SPI_agent agt;
14    spi_slave_agent SPI_SLAVE_agt;
15    ram_agent RAM_agt;
16    SPI_scoreboard sb;
17    SPI_coverage cov;
18
19    function new(string name = "SPI_env", uvm_component parent = null);
20      super.new(name, parent);
21    endfunction
22
23    function void build_phase(uvm_phase phase);
24      super.build_phase(phase);
25      agt = SPI_agent::type_id::create("agt", this);
26      SPI_SLAVE_agt = spi_slave_agent::type_id::create("SPI_SLAVE_agt", this);
27      RAM_agt = ram_agent::type_id::create("RAM_agt", this);
28      sb = SPI_scoreboard::type_id::create("sb", this);
29      cov = SPI_coverage::type_id::create("cov", this);
30    endfunction
31
32    function void connect_phase(uvm_phase phase);
33      super.connect_phase(phase);
34      agt.agt_ap.connect(sb.sb_export_SPI);
35      agt.agt_ap.connect(cov.cov_export_SPI);
36      SPI_SLAVE_agt.agt_ap.connect(sb.sb_export_SPI_SLAVE);
37      SPI_SLAVE_agt.agt_ap.connect(cov.cov_export_SPI_SLAVE);
38      RAM_agt.agt_ap.connect(sb.sb_export_RAM);
39      RAM_agt.agt_ap.connect(cov.cov_export_RAM);
40    endfunction
41
42  endclass
43
44 endpackage

```

Figure 28: Env

```

1 package SPI_test_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import SPI_config_pkg::*;
5   import SPI_env_pkg::*;
6   import SPI_sequencer_pkg::*;
7   import SPI_reset_sequence_pkg::*;
8   import SPI_wr_only_sequence_pkg::*;
9   import SPI_rd_only_sequence_pkg::*;
10  import SPI_wr_rd_sequence_pkg::*;
11
12 class SPI_test extends uvm_test;
13   `uvm_component_utils(SPI_test)
14
15   SPI_config SPI_cfg, SPI_SLAVE_cfg, RAM_cfg;
16   SPI_env env;
17   SPI_reset_sequence rst_seq;
18   SPI_wr_only_sequence wr_only_seq;
19   SPI_rd_only_sequence rd_only_seq;
20   SPI_wr_rd_sequence wr_rd_seq;
21
22   function new(string name = "SPI_test", uvm_component parent = null);
23     super.new(name, parent);
24   endfunction
25
26   function void build_phase(uvm_phase phase);
27     super.build_phase(phase);
28     env = SPI_env::type_id::create("env",this);
29     SPI_cfg = SPI_config::type_id::create("SPI_cfg");
30     SPI_SLAVE_cfg = SPI_config::type_id::create("SPI_SLAVE_cfg");
31     RAM_cfg = SPI_config::type_id::create("RAM_cfg");
32     rst_seq = SPI_reset_sequence::type_id::create("rst_seq");
33     wr_only_seq = SPI_wr_only_sequence::type_id::create("wr_only_seq");
34     rd_only_seq = SPI_rd_only_sequence::type_id::create("rd_only_seq");
35     wr_rd_seq = SPI_wr_rd_sequence::type_id::create("wr_rd_seq");
36
37     if (!uvm_config_db #(virtual SPI_if)::get(this, "", "SPI_IF", SPI_cfg.SPI_vif))
38       `uvm_fatal("Build_Phase", "Error in Wrapper Interface in test");
39
40     if (!uvm_config_db #(virtual SPI_SLAVE_if)::get(this, "", "SPI_SLAVE_IF", SPI_SLAVE_cfg.SPI_SLAVE_vif))
41       `uvm_fatal("Build_Phase", "Error in SPI_SLAVE Interface in test");
42
43     if (!uvm_config_db #(virtual RAM_if)::get(this, "", "RAM_IF", RAM_cfg.RAM_vif))
44       `uvm_fatal("Build_Phase", "Error in RAM Interface in test");
45
46     SPI_cfg.is_active = UVM_ACTIVE;
47     SPI_SLAVE_cfg.is_active = UVM_PASSIVE;
48     RAM_cfg.is_active = UVM_PASSIVE;
49
50     uvm_config_db #(SPI_config)::set(this, "*", "CONFIG", SPI_cfg);
51     uvm_config_db #(SPI_config)::set(this, "*", "CONFIG_SPI_SLAVE", SPI_SLAVE_cfg);
52     uvm_config_db #(SPI_config)::set(this, "*", "CONFIG_RAM", RAM_cfg);
53   endfunction
54
55   task run_phase(uvm_phase phase);
56     super.run_phase(phase);
57     phase.raise_objection(this);
58
59     //reset sequence
60     `uvm_info("Run_Phase", "Reset Sequence Start.", UVM_MEDIUM)
61     rst_seq.start(env.agt.sqr);
62     `uvm_info("Run_Phase", "Reset Sequence End.", UVM_MEDIUM)
63
64     //Write only sequence
65     `uvm_info("Run_Phase", "Write Only Sequence Start.", UVM_MEDIUM)
66     wr_only_seq.start(env.agt.sqr);
67     `uvm_info("Run_Phase", "Write Only Sequence End.", UVM_MEDIUM)
68
69     //Read only sequence
70     `uvm_info("Run_Phase", "Read Only Sequence Start.", UVM_MEDIUM)
71     rd_only_seq.start(env.agt.sqr);
72     `uvm_info("Run_Phase", "Read Only Sequence End.", UVM_MEDIUM)
73
74     //Write and Read sequence
75     `uvm_info("Run_Phase", "Write and Read Sequence Start.", UVM_MEDIUM)
76     wr_rd_seq.start(env.agt.sqr);
77     `uvm_info("Run_Phase", "Write and Read Sequence End.", UVM_MEDIUM)
78
79     phase.drop_objection(this);
80   endtask
81
82 endclass: SPI_test
83
84 endpackage

```

Figure 29: Test

```

1 import uvm_pkg::*;
2 `include "uvm_macros.svh"
3 import SPI_test_pkg::*;
4
5 module top();
6   logic clk;
7   initial begin
8     clk = 0;
9     forever #5 clk = ~clk;
10 end
11
12 //Interface
13 SPI_if inter(clk);
14 SPI_SLAVE_if inter_slave(clk);
15 RAM_if inter_ram(clk);
16
17 /*SPI Wrapper*/
18
19 //DUT
20 //module SPI_Wrapper (input clk,rst_n,SS_n,MOSI, output MISO);
21 SPI_Wrapper #(inter.MEM_DEPTH, inter.ADDR_SIZE) DUT(inter.clk, inter.rst_n, inter.SS_n, inter.MOSI, inter.MISO);
22
23 //GOLDEN
24 //module SPI_Wrapper_GOLDEN (input clk,rst_n,SS_n,MOSI, output MISO);
25 SPI_Wrapper_GOLDEN #(inter.MEM_DEPTH, inter.ADDR_SIZE) DUT_GOLDEN(inter.clk, inter.rst_n, inter.SS_n, inter.MOSI, inter.MISO_GOLDEN);
26
27
28 /*SPI SLAVE*/
29
30 //DUT
31 //module SPI_SLAVE (MOSI,SS_n,clk,rst_n,rx_data,tx_valid,tx_data,MISO,rx_valid);
32 SPI_SLAVE DUT_SPI_SLAVE(inter_slave.MOSI, inter_slave.SS_n, inter_slave.clk, inter_slave.rst_n, inter_slave.rx_data,
33                           inter_slave.tx_valid, inter_slave.tx_data, inter_slave.MISO, inter_slave.rx_valid);
34
35 //GOLDEN
36 //module SPI_SLAVE_GOLDEN (MOSI,SS_n,clk,rst_n,rx_data,tx_valid,tx_data,MISO,rx_valid);
37 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,
38                                         inter_slave.tx_valid, inter_slave.tx_data, inter_slave.MISO_ref, inter_slave.rx_valid_ref);
39 assign inter_slave.rst_n      = DUT.rst_n;
40 assign inter_slave.MOSI       = DUT.MOSI;
41 assign inter_slave.SS_n       = DUT.SS_n;
42 assign inter_slave.tx_valid  = DUT.tx_valid;
43 assign inter_slave.tx_data   = DUT.tx_data_dout;
44
45
46 /*RAM*/
47
48 //DUT
49 //module RAM (din,clk,rst_n,rx_valid,dout,tx_valid);
50 RAM DUT_RAM(inter_ram.din, inter_ram.clk, inter_ram.rst_n, inter_ram.rx_valid, inter_ram.dout, inter_ram.tx_valid);
51
52 //GOLDEN
53 //module RAM_GOLDEN (clk, rst_n, din, rx_valid, dout, tx_valid);
54 RAM_GOLDEN #(inter_ram.MEM_DEPTH, inter_ram.ADDR_SIZE) DUT_RAM_GOLDEN(inter_ram.clk, inter_ram.rst_n, inter_ram.din,
55                           inter_ram.rx_valid, inter_ram.dout_ref, inter_ram.tx_valid_ref);
56 assign inter_ram.rst_n      = DUT.rst_n;
57 assign inter_ram.din        = DUT.rx_data_din;
58 assign inter_ram.rx_valid   = DUT.rx_valid;
59
60
61 //sva
62 bind SPI_Wrapper SPI_sva Wrapper_BINDING(inter.clk, inter.rst_n, inter.SS_n, inter.MOSI, inter.MISO);
63 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);
64
65 initial begin
66   uvm_config_db #(virtual SPI_if)::set(null, "uvm_test_top", "SPI_IF", inter);
67   uvm_config_db #(virtual SPI_SLAVE_if)::set(null, "uvm_test_top", "SPI_SLAVE_IF", inter_slave);
68   uvm_config_db #(virtual RAM_if)::set(null, "uvm_test_top", "RAM_IF", inter_ram);
69   run_test("SPI_test");
70 end
71 endmodule

```

Figure 30: Top

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

Figure 31: Do File

```
1 RAM.v
2 RAM_GOLDEN.v
3 SPI_SLAVE.sv
4 SPI_SLAVE_GOLDEN.v
5 SPI_Wrapper.v
6 SPI_Wrapper_GOLDEN.v
7 RAM_sva.sv
8 SPI_sva.sv
9 SPI_SLAVE_if.sv
10 RAM_if.sv
11 SPI_Wrapper_if.sv
12 RAM_shared_pkg.sv
13 SPI_SLAVE_shared_pkg.sv
14 SPI_pkg.sv
15 SPI_config.sv
16 RAM_seq_item.sv
17 SPI_SLAVE_seq_item.sv
18 SPI_seq_item.sv
19 SPI_reset_sequence.sv
20 SPI_wr_only_sequence.sv
21 SPI_rd_only_sequence.sv
22 SPI_wr_rd_sequence.sv
23 RAM_sqr.sv
24 SPI_SLAVE_sequencer.sv
25 SPI_sequencer.sv
26 RAM_driver.sv
27 SPI_SLAVE_driver.sv
28 SPI_driver.sv
29 RAM_monitor.sv
30 SPI_SLAVE_monitor.sv
31 SPI_monitor.sv
32 SPI_scoreboard.sv
33 SPI_coverage.sv
34 RAM_agent.sv
35 SPI_SLAVE_agent.sv
36 SPI_agent.sv
37 SPI_env.sv
38 SPI_test.sv
39 top.sv
```

Figure 32: src File

3.4. Wave:

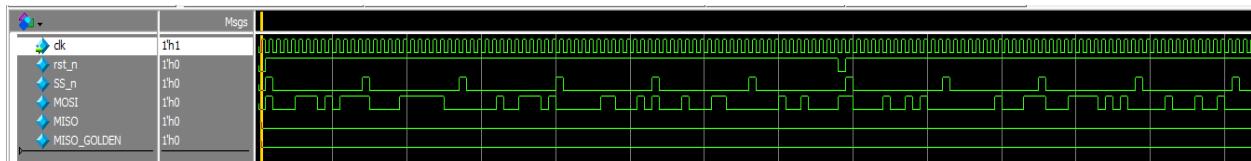


Figure 33: Reset Sequence Wave

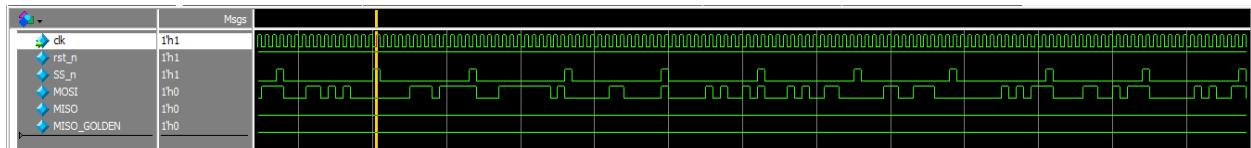


Figure 34: Write only Sequence Wave

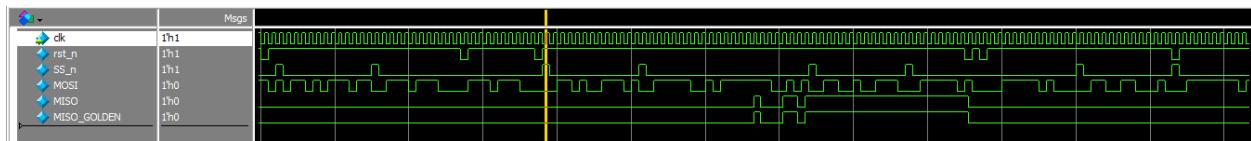


Figure 35: Read only Sequence Wave

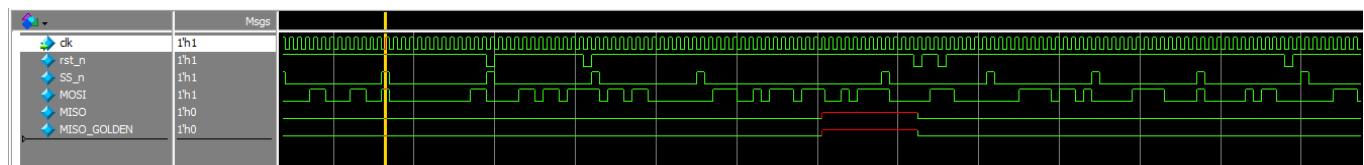


Figure 36: Write/Read Sequence Wave

```
# 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] questauvm:init(all)
# UVM_INFO SPI_top.sv(60) @ 0: uvm_test_top [Run_Phase] Reset Sequence Start.
# ***** Transaction Recording Turned ON. *****
# * 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_config_db#(uvm_sequencer_t)::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_driver_t)::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_monitor_t)::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_processor_t)::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_reporter_t)::set(null, "", "recording_detail", 0); *
# UVM_INFO SPI_top.sv(62) @ 10: uvm_test_top [Run_Phase] Reset Sequence End.
# UVM_INFO SPI_top.sv(65) @ 10: uvm_test_top [Run_Phase] Write Only Sequence Start.
# UVM_INFO SPI_top.sv(67) @ 50010: uvm_test_top [Run_Phase] Write Only Sequence End.
# UVM_INFO SPI_top.sv(70) @ 50010: uvm_test_top [Run_Phase] Read Only Sequence Start.
# UVM_INFO SPI_top.sv(72) @ 100010: uvm_test_top [Run_Phase] Read Only Sequence End.
# UVM_INFO SPI_top.sv(75) @ 100010: uvm_test_top [Run_Phase] Write and Read Sequence Start.
# UVM_INFO SPI_top.sv(77) @ 150010: uvm_test_top [Run_Phase] Write and Read Sequence End.
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 150010: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO SPI_scoreboard.sv(81) @ 150010: uvm_test_top.env.sv [Report Phase] Correct = 15001
# UVM_INFO SPI_scoreboard.sv(82) @ 150010: uvm_test_top.env.sv [Report Phase] SFR Wrapper Errors = 0
# UVM_INFO SPI_scoreboard.sv(83) @ 150010: uvm_test_top.env.sv [Report Phase] SPI_SLAVE Errors = 0
# UVM_INFO SPI_scoreboard.sv(84) @ 150010: uvm_test_top.env.sv [Report Phase] RAM Errors = 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 16
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
# ** Report counts by id
# [Questa UVM] 2
# [RNTEST] 1
# [Report Phase] 4
# [Run_Phase] 8
# [TEST_DONE] 1
# ** Note: $finish : C:/questasim64_2021.1/win64/..//verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
# Time: 150010 ns Iteration: 61 Instance: /top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/..//verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```

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

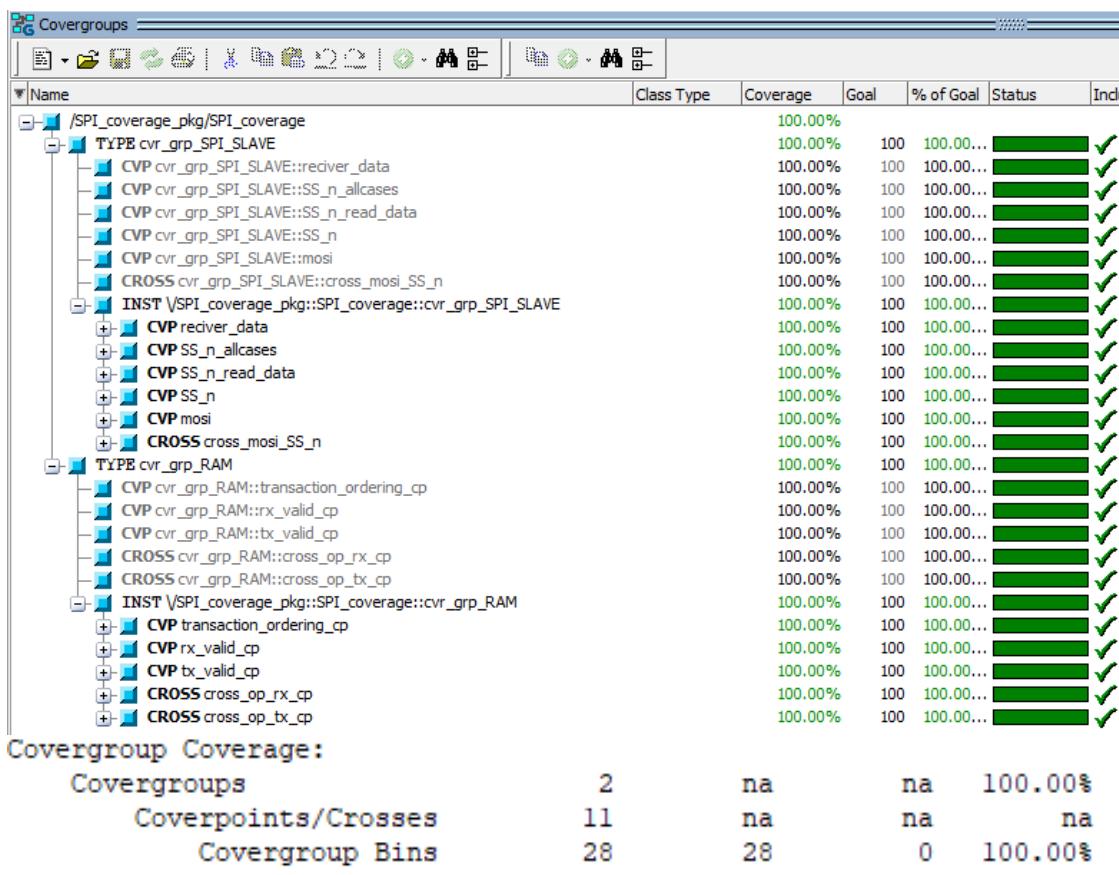


Figure 45: Functional Coverage

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed_1735	Immediate	SVA	on	0	0	
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed_1775	Immediate	SVA	on	0	0	
/SPI_wr_rd_sequence_pkg::SPI_wr_rd_sequence::body/#ublk#82182855#19/immed_37	Immediate	SVA	on	0	1	
/SPI_d_only_sequence_pkg::SPI_d_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_RAM/RAM_BINDING/rst_n_assertion	Concurrent	SVA	on	0	1	
+ /top/DUT/DUT_RAM/RAM_BINDING/bx_valid_0_assertion	Concurrent	SVA	on	0	1	
+ /top/DUT/DUT_RAM/RAM_BINDING/bx_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/bx_valid_0_assertion	Concurrent	SVA	on	0	1	
+ /top/DUT_RAM/RAM_BINDING/bx_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

Cover Directives

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%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT/DUT_SPI/cover_chck_state_write_to_idle	SVA	✓	Off	65	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT/DUT_SPI/cover_chck_state_to_idle	SVA	✓	Off	124	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT/DUT_SPI/cover_chck_state_read_data	SVA	✓	Off	184	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT/DUT_SPI/cover_chck_state_read_add	SVA	✓	Off	324	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT/DUT_SPI/cover_chck_state_write	SVA	✓	Off	638	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT/DUT_SPI/cover_chck_state_idle	SVA	✓	Off	1208	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT/DUT_SPI/cover_chck_rx_valid	SVA	✓	Off	2	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT/DUT_SPI/cover_chck_reset	SVA	✓	Off	295	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT/Wrapper_BINDING/RESET_cover	SVA	✓	Off	295	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT/Wrapper_BINDING/READ_DATA_cover	SVA	✓	Off	958	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT_SPI_SLAVE/cover_chck_state_read_datato_idle	SVA	✓	Off	57	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT_SPI_SLAVE/cover_chck_state_read_add_to_idle	SVA	✓	Off	65	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT_SPI_SLAVE/cover_chck_state_write_to_idle	SVA	✓	Off	124	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT_SPI_SLAVE/cover_chck_state_read_data	SVA	✓	Off	184	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT_SPI_SLAVE/cover_chck_state_read_add	SVA	✓	Off	324	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT_SPI_SLAVE/cover_chck_state_write	SVA	✓	Off	638	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT_SPI_SLAVE/cover_chck_state_idle	SVA	✓	Off	1208	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT_SPI_SLAVE/cover_chck_rx_valid	SVA	✓	Off	2	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓
/top/DUT_SPI_SLAVE/cover_chck_reset	SVA	✓	Off	295	1	Unli...	1	100%	<div style="width: 100%; background-color: green;"></div>	✓

Figure 47: Cover Directives

4- Verification Plan:

1	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 When reset is asserted , MISO , rx_data , rx_valid will be low	Randomized with constraint rd_wr_constraint	Cover if all operations have occurred or not	Reference Model, Assertions
6	RESET	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	Directed at the beginning of simulation then randomized with constraint to make reset asserted by probability 2%	-	Reference Model, Assertions
7	SS_n_high		Randomized with constraint SS_n_high	Cover SS_n sequence	Reference Model, Assertions

Figure 48: Verification Plan