UVM Project

Submitted by:

- Shahd Mohamed Mohamed Ramez
- Mustafa Khaled Khalifa
- Mohamed Hesham Hassan

Contents

PART 1: RAM

Verification Plan

1. Activate reset through the reset sequence at the beginning to set the program to a default state.
2. Activate the main sequence which has 3 different sequences, one for write operation only, another one for read operation only and the last for write and read operations.
3. Check that din [7:0] is written in the internal write_addr register when rx_valid is high and din [9:8] = 2'b00.
4. Check that din [7:0] is written in RAM in location of address stored in internal write_addr register when rx_valid is high din [9:8] = 2'b01.
5. Check that din [7:0] is written in the internal read_addr register when rx_valid is high and din [9:8] = 2'b10.
6. Check that data stored in RAM in location of address stored in internal read_addr register is outputted on dout bus and tx_valid is set high when rx_valid is high and din [9:8] = 2'b11.

Snippet of verification document

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
RAM_1	Incase rst_n is activated, dout and tx_valid are set to zero.	Activate rst_n at the beginning of simulation. Plus randomization under the constraints of rst to be low most of the time.	Included as coverpoint for rst_n when it is activated (when rst_n is zero).	Output Checked against golden model and assertions.
RAM_2	Incase rx_valid is high and din[9:8] is 2'b00 (write_addr), then din[7:0] are saved in the write_addr register in RAM module.	Randomization under constraints on rx_valid to be activated (high) 90% of the time, din[9:8] to be 2'b00 40% of the time if the last op was not read addr or write addr, and with less probability if the prev operation was write addr or read addr.	*Included as coverpoint of 256 bins for a variable named write_address that stores write addr value when din[9:8] = 2'b00, and is sampled when the current operation is write_data (din[9:8] = 2'b01) to make sure that all ram addresses are written to them (fully excercised). *Included as coverpoint for din[9:8] to make sure write_addr operation occured.	Output Checked against golden model and assertions.
RAM_3	Incase rx_valid is high and din[9:8] is 2'b01(write_data), then din[7:0] are written in RAM in address stored in write_addr register.	Randomization under constraints on rx_valid to be activated (high) 90% of the time, din[9:8] to be 2'b01 70% of the time if the last op was write addr, and 10% of the time if the previous operation was neither read addr or write addr.	*Included as coverpoint of 256 bins for a variable named write_address that stores write addr value when din[9:8] = 2'b00 (write_addr operation), and is sampled when the current operation is write_data (din[9:8] = 2'b01) to make sure that all ram addresses are written to them (fully excercised). *Included as coverpoint for din[9:8] to make sure write_data operation occured. Included as coverpoint for din[9:8] when transitioning from write_addr operation to write_data opeartion.	Output Checked against golden model and assertions.

Figure 1: Verification document for RAM.

RAM_4	Incase rx_valid is high and din[9:8] is 2'b10 (read_addr), then din[7:0] are saved in the read_addr register in RAM module.	Randomization under constraints on rx_valid to be activated (high) 90% of the time, din[9:8] to be 2'b11 40% of the time if the last op was not read addr or write addr, and with less probability if the previous operation was write addr or read addr.	*Included as coverpoint of 256 bins for a variable named read_address that stores write addr value when din[9:8] = 2'b10, and is sampled when the current operation is read_data (din[9:8] = 2'b11) to make sure that all ram addresses are read from them (fully excercised). *Included as coverpoint for din[9:8] to make sure read_addr operation occured.	Output Checked against golden model and assertions.
RAM_5	Incase rx_valid is high and din[9:8] is 2'b11 (read_data), then mem[read_addr] is outputed on the dout bus and tx_valid is set high.	Randomization under constraints on rx_valid to be activated (high) 90% of the time, din[9:8] to be 2'b11 70% of the time if the last op was write addr, and 10% of the time if the previous operation was neither read addr or write addr.	*Included as coverpoint of 256 bins for a variable named read_address that stores write addr value when din[9:8] = 2'b10, and is sampled when the current operation is read_data (din[9:8] = 2'b11) to make sure that all ram addresses are read from them (fully excercised). *Included as coverpoint for din[9:8] to make sure read_data operation occured. Included as cross for tx_valid with declaring tx_valid being high and din[9:8] != 2'b11 as an illegal bin and ignoring bin when tx_valid is low. *Included as coverpoint for din[9:8] when transitioning from read_addr operation to read_data opeartion.	Output Checked against golden model and assertions.

Figure 2: Verification document for RAM.

UVM Structure diagram

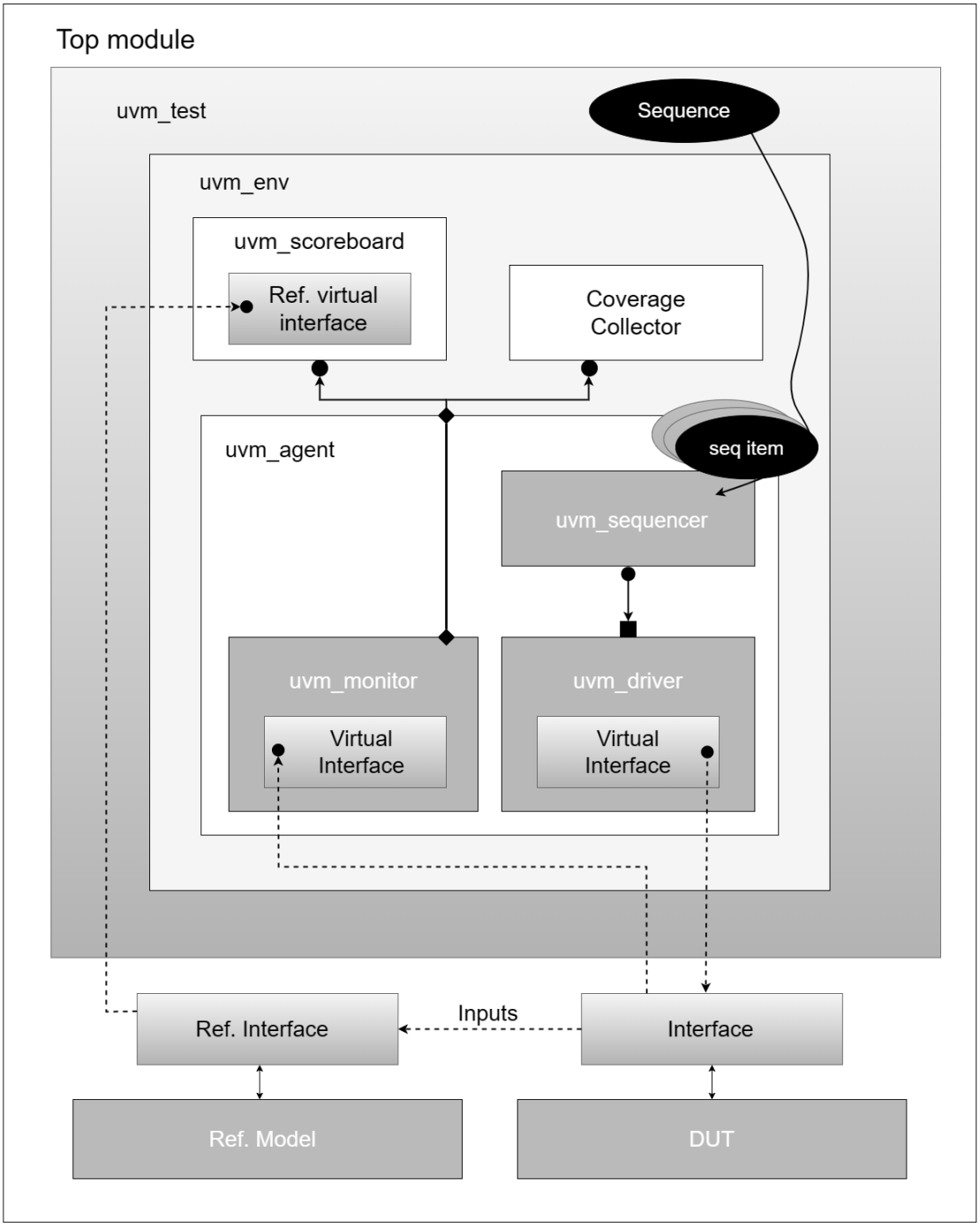


Figure 3: RAM UVM testbench structure diagram.

```
# UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
# ----------------------------------------------------------------
# Name                        Type                          Size  Value
# ----------------------------------------------------------------
# uvm_test_top                RAM_test                      -     @474
#   env                       RAM_env                       -     @481
#     agt                     RAM_agent                     -     @503
#       agt_ap                uvm_analysis_port             -     @665
#         recording_detail    integral                      32    'd1
#       driver                RAM_driver                    -     @635
#         rsp_port            uvm_analysis_port             -     @650
#           recording_detail  integral                      32    'd1
#         seq_item_port       uvm_seq_item_pull_port        -     @642
#           recording_detail  integral                      32    'd1
#         recording_detail    integral                      32    'd1
#       mon                   RAM_monitor                   -     @658
#         mon_ap              uvm_analysis_port             -     @673
#           recording_detail  integral                      32    'd1
#         recording_detail    integral                      32    'd1
#       sqr                   RAM_sequencer                 -     @526
#         rsp_export          uvm_analysis_export           -     @533
#           recording_detail  integral                      32    'd1
#         seq_item_export     uvm_seq_item_pull_imp         -     @627
#           recording_detail  integral                      32    'd1
#         recording_detail    integral                      32    'd1
#         arbitration_queue   array                         0     -
#         lock_queue          array                         0     -
#         num_last_reqs       integral                      32    'd1
#         num_last_rsps       integral                      32    'd1
#       recording_detail      integral                      32    'd1
#     cov                     RAM_coverage                  -     @517
#       cov_export            uvm_analysis_export           -     @686
#         recording_detail    integral                      32    'd1
#       cov_fifo              uvm_tlm_analysis_fifo #(T)    -     @694
#         analysis_export     uvm_analysis_imp              -     @733
#           recording_detail  integral                      32    'd1
#         get_ap              uvm_analysis_port             -     @725
#           recording_detail  integral                      32    'd1
#         get_peek_export     uvm_get_peek_imp              -     @709
#           recording_detail  integral                      32    'd1
#         put_ap              uvm_analysis_port             -     @717
#           recording_detail  integral                      32    'd1
#         put_export          uvm_put_imp                   -     @701
#           recording_detail  integral                      32    'd1
#         recording_detail    integral                      32    'd1
#       recording_detail      integral                      32    'd1
```

Figure 4: UVM testbench topology.

```
#     sb                    RAM_scoreboard              -     @510
#       sb_export           uvm_analysis_export         -     @741
#         recording_detail  integral                    32    'd1
#       sb_fifo             uvm_tlm_analysis_fifo #(T)  -     @749
#         analysis_export   uvm_analysis_imp            -     @788
#           recording_detail  integral                  32    'd1
#         get_ap            uvm_analysis_port           -     @780
#           recording_detail  integral                  32    'd1
#         get_peek_export   uvm_get_peek_imp            -     @764
#           recording_detail  integral                  32    'd1
#         put_ap            uvm_analysis_port           -     @772
#           recording_detail  integral                  32    'd1
#         put_export        uvm_put_imp                 -     @756
#           recording_detail  integral                  32    'd1
#         recording_detail  integral                    32    'd1
#       recording_detail    integral                    32    'd1
#     recording_detail      integral                    32    'd1
#   recording_detail        integral                    32    'd1
# ----------------------------------------------------------------
```

Figure 5: UVM testbench topology.

Coverage

Code coverage

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        14        14         0   100.00%

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

Branch Coverage for instance /\top#DUT
```

Figure 6: RAM branch coverage.

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      18        18         0   100.00%

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

Statement Coverage for instance /\top#DUT  --
```

Figure 7: RAM statement coverage.

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

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

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

Figure 8: RAM toggle coverage.

Functional coverage

```
============================================================================
=== Instance: /RAM_coverage_pkg
=== Design Unit: work.RAM_coverage_pkg
============================================================================

Covergroup Coverage:
    Covergroups                     1        na        na   100.00%
        Coverpoints/Crosses         8        na        na        na
            Covergroup Bins       526       526         0   100.00%
----------------------------------------------------------------------------------------------------
```

Figure 9: RAM functional coverage report.

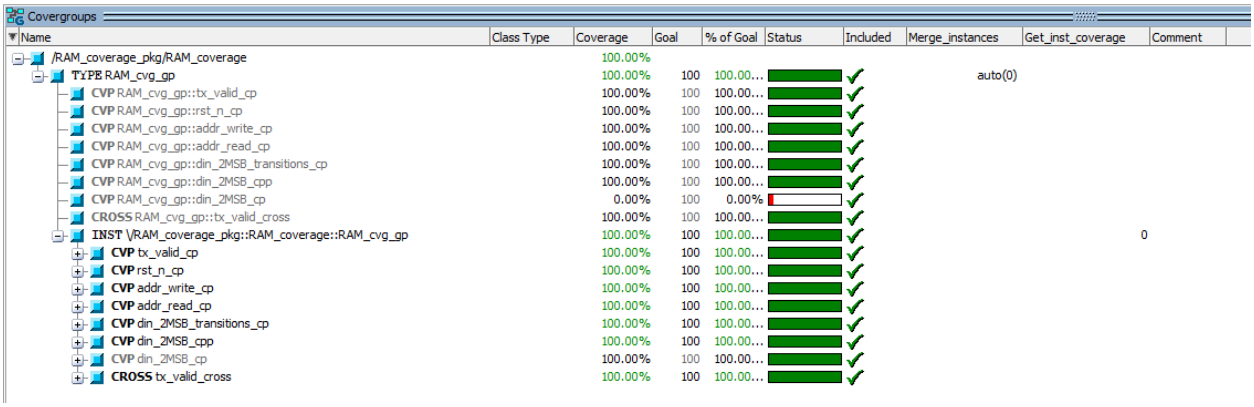


Figure 10: RAM functional coverage questasim snippet.

Assertions coverage

```
================================================================================
=== Instance: /\top#DUT /RAM_sva_inst
=== Design Unit: work.RAM_sva
================================================================================

Assertion Coverage:
    Assertions                       3         3         0   100.00%
--------------------------------------------------------------------
Name                 File(Line)                    Failure      Pass
                                                   Count        Count
--------------------------------------------------------------------
/\top#DUT /RAM_sva_inst/a_rst_dout
                     RAM_sva.sv(8)                       0          1
/\top#DUT /RAM_sva_inst/a_rst_tx_valid
                     RAM_sva.sv(9)                       0          1
/\top#DUT /RAM_sva_inst/a_tx_valid_pr
                     RAM_sva.sv(18)                      0          1
```

Figure 11: RAM assertions coverage report.

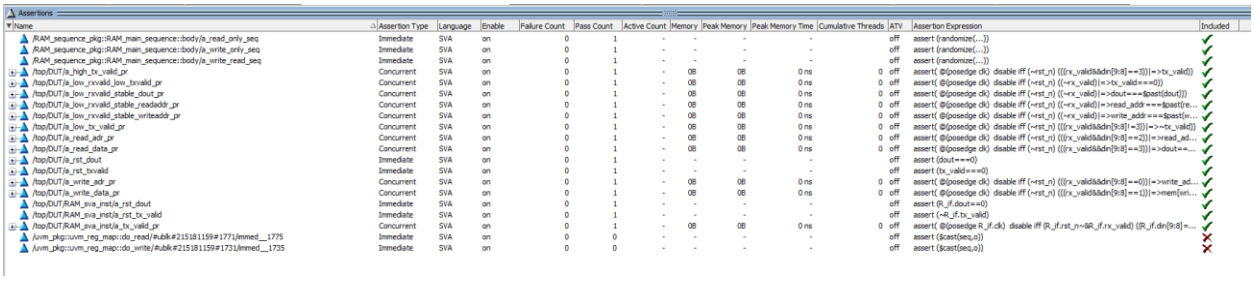


Figure 12: RAM assertions.

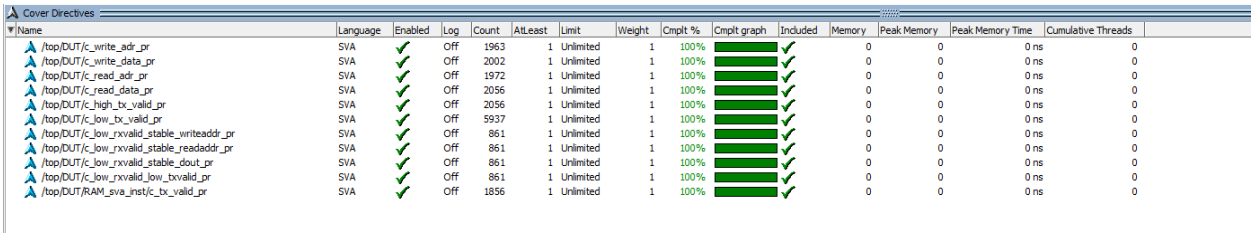


Figure 13: RAM assertions directive questa snippet.

Bugs reporting

❖ Internal addresses bug

Write address and read address should be have different registers cause the RAM can have a write address operation and then a read address operation happens before read data, or the opposite write address operation can happen before read data operation occurs to read the data at address stored in read address register, so having one register for both will cause one of them to overwrite the other in the mentioned scenarios.

```
module Dual_port_RAM(clk,rst_n,rx_valid,din,dout,tx_valid);
parameter MEM_DEPTH=256;
parameter ADDR_SIZE=8;

input [9:0]din;
input clk,rst_n,rx_valid;
output reg tx_valid;
output reg [7:0]dout;
reg [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];
reg[ADDR_SIZE-1:0] temp_adr;

always @(posedge clk or negedge rst_n) begin
    if (~rst_n) begin
    dout<=8'b0000_0000;
    tx_valid=0;
    end
    else begin
      if(rx_valid==1)begin
            if(din[9]==0)begin
              if(din[8]==0)begin
                    temp_adr<=din[7:0];//keda ha write address fel memory
                    tx_valid<=0;
               end
               else if( din[8]==1)begin
                   mem[temp_adr]<=din[7:0];// hna hy write data fel address
                   tx_valid<=0;
                end
            end
            if(din[9]==1)begin
                if (din[8]==0) begin
                     temp_adr<= din[7:0];
                      tx_valid<=0;
                end
                else if(din[8]==1)begin
                     dout[7:0]<= mem[temp_adr];
                     tx_valid<=1;

                end
          end
      end
    end
end //always
endmodule
```

Figure 14: Prefixed addresses registers bug of RAM design.

```
1  module Dual_port_RAM(RAM_if.DUT R_if);
2
3  reg [R_if.ADDR_SIZE-1:0] mem [R_if.MEM_DEPTH-1:0];
4  reg [R_if.ADDR_SIZE-1:0] write_adr,read_adr;
5  always @(posedge R_if.clk or negedge R_if.rst_n) begin
6      if (~R_if.rst_n) begin
7          R_if.dout<=8'b0000_0000;
8          R_if.tx_valid=0;
9      end
10     else begin
11         if(R_if.rx_valid==1)begin
12             if(R_if.din[9]==0)begin
13                 if(R_if.din[8]==0)begin
14                     write_adr<=R_if.din[7:0];
15                     R_if.tx_valid<=0;
16                 end
17                 else if( R_if.din[8]==1)begin
18                     mem[write_adr]<=R_if.din[7:0];
19                     R_if.tx_valid<=0;
20                 end
21             end
22
23             if(R_if.din[9]==1)begin
24                 if (R_if.din[8]==0) begin
25                     read_adr<= R_if.din[7:0];
26                     R_if.tx_valid<=0;
27                 end
28                 else if(R_if.din[8]==1)begin
29                     R_if.dout[7:0]<= mem[read_adr];
30                     R_if.tx_valid<=1;
31                 end
32             end
33         end
34     end
35 end
36
37 endmodule
```

Figure 15: Post fixed addresses registers bug of RAM design.

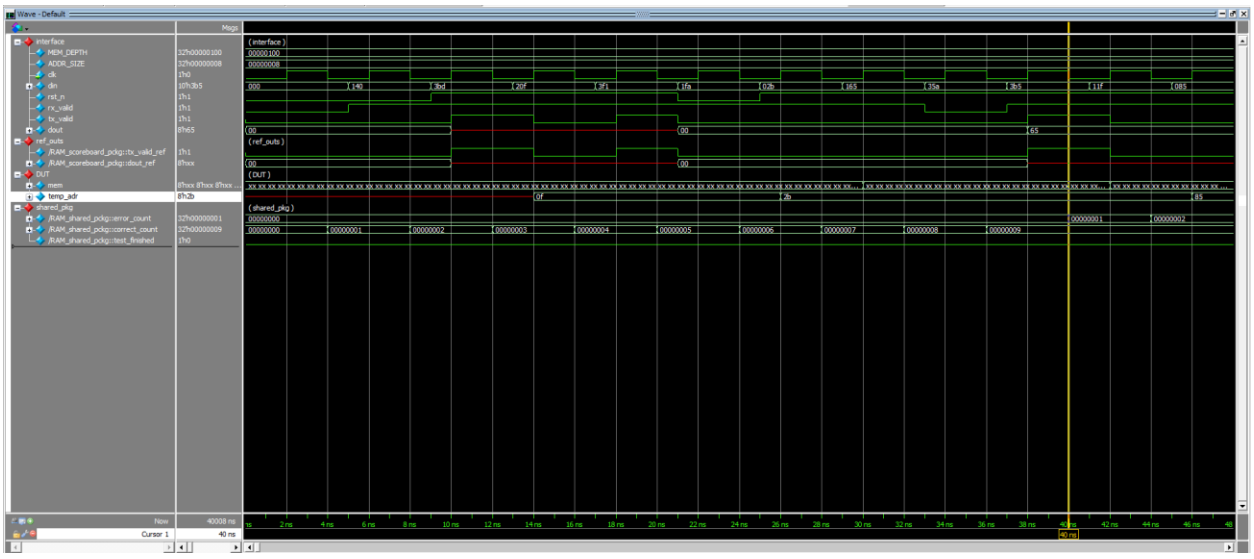


Figure 16: addresses registers bug demonstrated on waveform.

❖ Not resetting address bug

As shown in fig (50), the temp address was not assigned zero on resetting, but a better coding style is to reset it to zero on reset after splitting it to two addresses as shown in fig (53). By that on reading data without sending reading address before it (non-destructive invalid case), the reference model will be reading content of address zero while DUT will be reading content of last sent read address as demonstrated on the waveform in fig (54).

```
reg [R_if.ADDR_SIZE-1:0] mem [R_if.MEM_DEPTH-1:0];
reg [R_if.ADDR_SIZE-1:0] write_adr,read_adr;
always @(posedge R_if.clk or negedge R_if.rst_n) begin
    if (~R_if.rst_n) begin
        R_if.dout<=8'b0000_0000;
        R_if.tx_valid=0;
        write_adr <= 0;
        read_adr  <=0;
    end
    else begin
```

Figure 17: Postfixed resetting addresses registers to zero bug of RAM design.

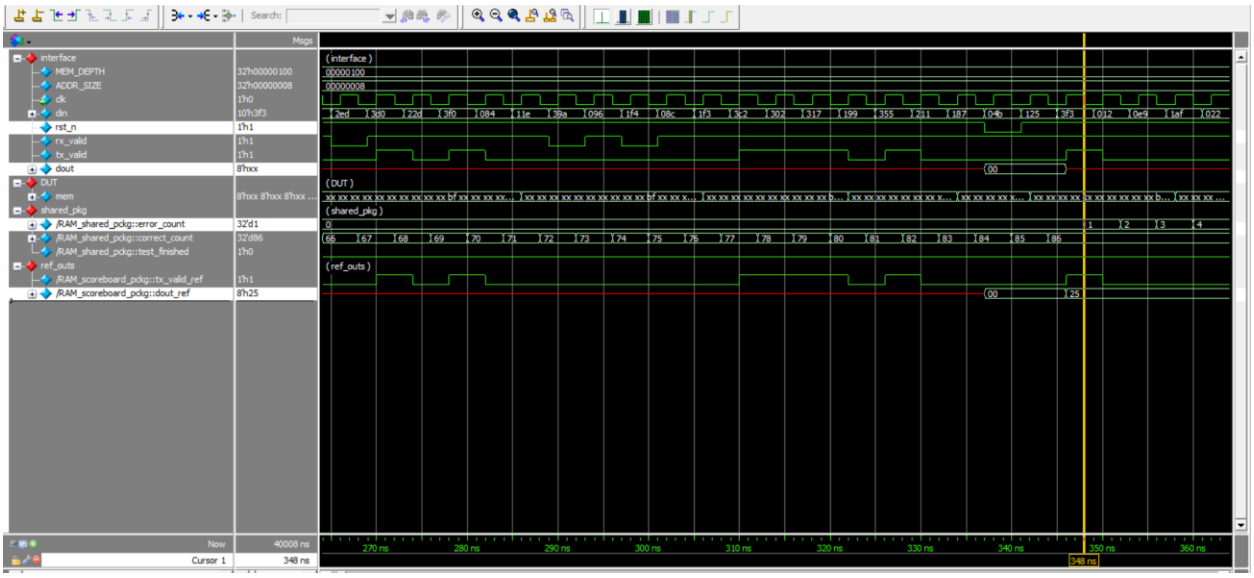


Figure 18: Not resetting addresses registers bug demonstrated on the wave.

❖ Tx valid bug

When rx_valid is low, tx_valid should be low as well, cause no operation is happening right now in the RAM, so the RAM is not outputting dout bus to the SPI.

The bug on the wave is demonstrated in fig (55), where tx_valid in reference model is set low, while that in the interface is still high although rx_valid is low.

The bugged code is shown in fig (56), and the postfixed code is shown in fig (57).

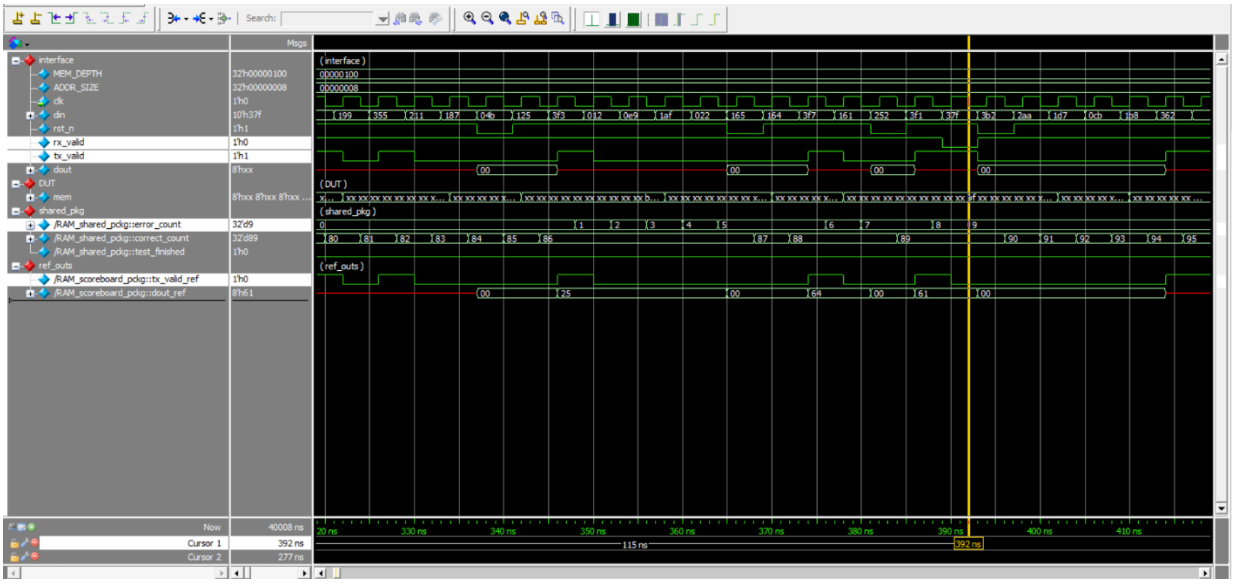


Figure 19: tx_valid bug demonstrated on the waveform.

```
module Dual_port_RAM(RAM_if.DUT R_if);
reg [R_if.ADDR_SIZE-1:0] mem [R_if.MEM_DEPTH-1:0];
reg [R_if.ADDR_SIZE-1:0] write_adr,read_adr;
always @(posedge R_if.clk or negedge R_if.rst_n) begin
    if (~R_if.rst_n) begin
        R_if.dout<=8'b0000_0000;
        R_if.tx_valid=0;
        write_adr <= 0;
        read_adr  <=0;
    end
    else begin
        if(R_if.rx_valid==1)begin
            if(R_if.din[9]==0)begin
                if(R_if.din[8]==0)begin
                    write_adr<=R_if.din[7:0];
                    R_if.tx_valid<=0;
                end
                else begin
                    mem[write_adr]<=R_if.din[7:0];
                    R_if.tx_valid<=0;
                end
            end

            if(R_if.din[9]==1)begin
                if (R_if.din[8]==0) begin
                    read_adr<= R_if.din[7:0];
                    R_if.tx_valid<=0;
                end
                else begin
                    R_if.dout[7:0]<= mem[read_adr];
                    R_if.tx_valid<=1;
                end
            end
        end
    end
end
endmodule
```

Figure 20: Prefixed tx_valid bug of RAM design.

```
module Dual_port_RAM(RAM_if.DUT R_if);
reg [R_if.ADDR_SIZE-1:0] mem [R_if.MEM_DEPTH-1:0];
reg [R_if.ADDR_SIZE-1:0] write_adr,read_adr;
always @(posedge R_if.clk or negedge R_if.rst_n) begin
    if (~R_if.rst_n) begin
        R_if.dout<=8'b0000_0000;
        R_if.tx_valid=0;
        write_adr <= 0;
        read_adr  <=0;
    end
    else begin
        if(R_if.rx_valid==1)begin
            if(R_if.din[9]==0)begin
                if(R_if.din[8]==0)begin
                    write_adr<=R_if.din[7:0];
                    R_if.tx_valid<=0;
                end
                else begin
                    mem[write_adr]<=R_if.din[7:0];
                    R_if.tx_valid<=0;
                end
            end

            if(R_if.din[9]==1)begin
                if (R_if.din[8]==0) begin
                    read_adr<= R_if.din[7:0];
                    R_if.tx_valid<=0;
                end
                else begin
                    R_if.dout[7:0]<= mem[read_adr];
                    R_if.tx_valid<=1;
                end
            end
        end
        else begin
            R_if.tx_valid <= 0;
        end
    end
end
endmodule
```

Figure 21: Postfixed tx_valid bug of RAM design.

❖ Not resetting memory bug

resetting memory content to zero when rst_n is activated was handled in reference model but not handled in the DUT. The for loop was added as shown if fig (58) to set all memory places to zero.

```
1   module Dual_port_RAM(RAM_if.DUT R_if);
2
3   reg [R_if.ADDR_SIZE-1:0] mem [R_if.MEM_DEPTH-1:0];
4   reg [R_if.ADDR_SIZE-1:0] write_addr,read_addr;
5   always @(posedge R_if.clk or negedge R_if.rst_n) begin
6       if (~R_if.rst_n) begin
7           R_if.dout<=8'b0000_0000;
8           R_if.tx_valid=0;
9           write_addr <= 0;
10          read_addr  <= 0;
11
12          for (int i = 0; i< MEM_DEPTH-1; i=i+1) begin
13              mem [i] <= 0;
14          end
15      end
16      else begin
17          if(R_if.rx_valid==1)begin
18              if(R_if.din[9]==0)begin
19                  if(R_if.din[8]==0)begin
20                      write_addr<=R_if.din[7:0];
21                      R_if.tx_valid<=0;
22                  end
23                  else begin
24                      mem[write_addr]<=R_if.din[7:0];
25                      R_if.tx_valid<=0;
26                  end
27              end
28
29              if(R_if.din[9]==1)begin
30                  if (R_if.din[8]==0) begin
31                      read_addr<= R_if.din[7:0];
32                      R_if.tx_valid<=0;
33                  end
34                  else begin
35                      R_if.dout[7:0]<= mem[read_addr];
36                      R_if.tx_valid<=1;
37                  end
38              end
39          end
40          else begin
41              R_if.tx_valid <= 0;
42          end
43      end
44  end
45
```

Figure 22: Postfixed code of resetting memory elements to zero.

Waveform

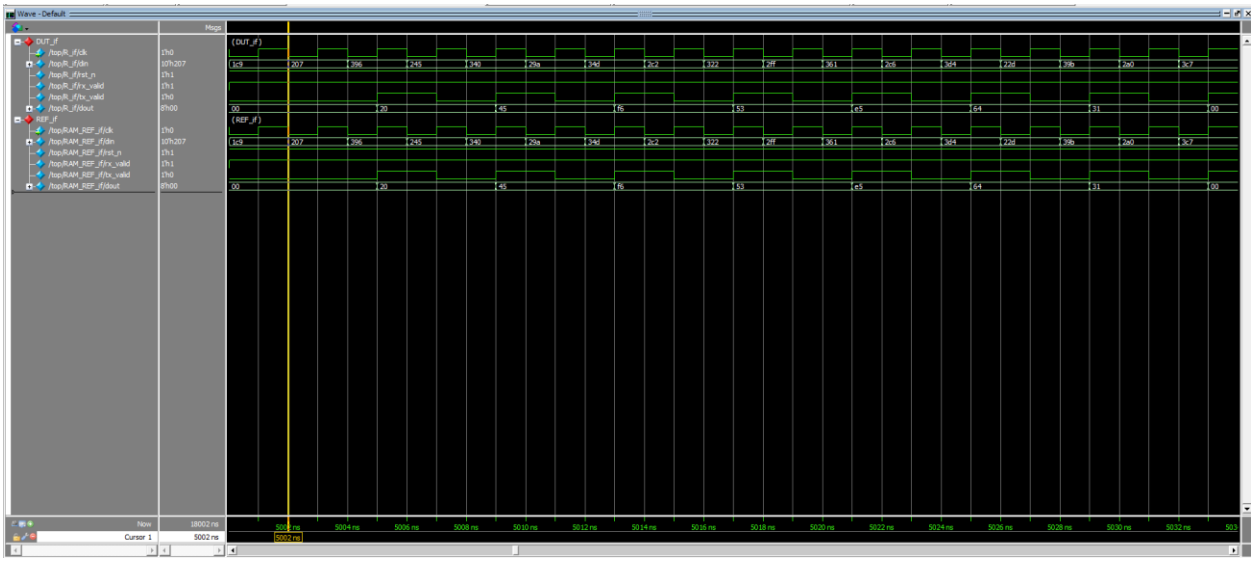


Figure 23: RAM read only sequence waveform snippet.

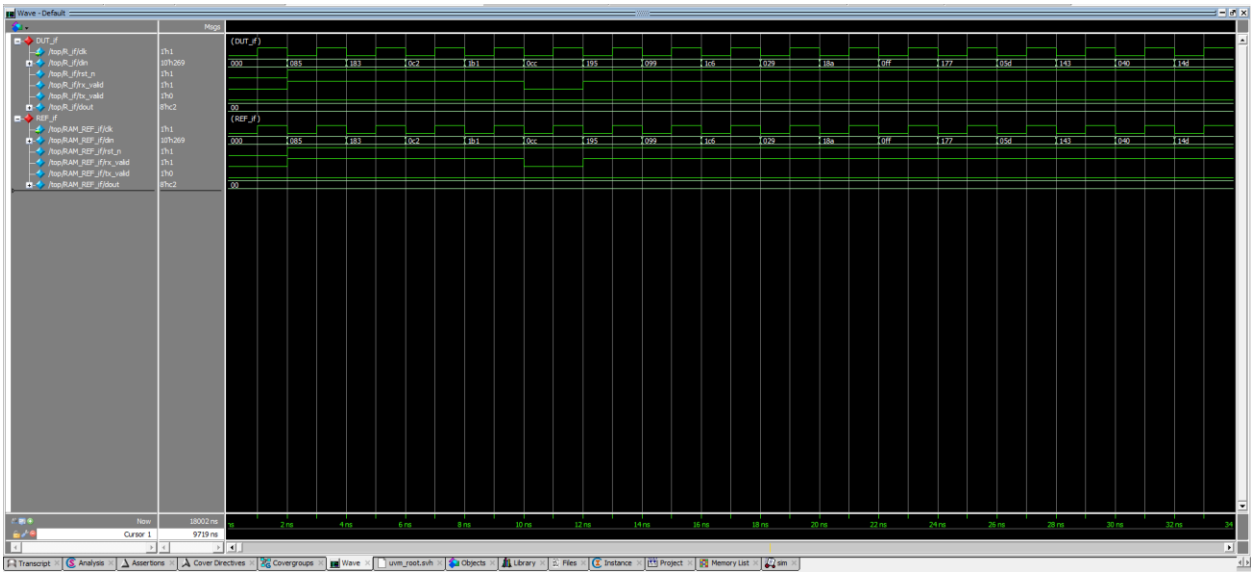


Figure 24: RAM write only sequence waveform snippet.

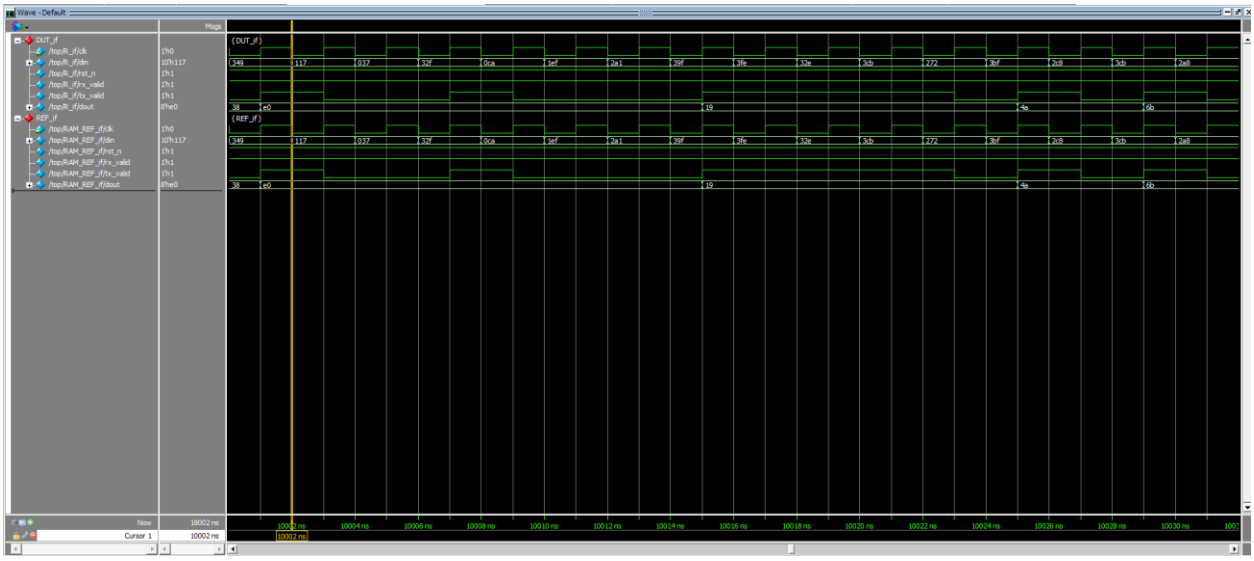


Figure 25: RAM read_write sequence waveform snippet.

Transcript

```
#
# UVM_INFO RAM_test_class.sv(50) @ 0: uvm_test_top [run_phase] Reset Asserted
# ****************************************************************************
# * Questa UVM Transaction Recording Turned ON.                              *
# * recording_detail has been set.                                           *
# *  To turn off, set 'recording_detail' to off:                             *
# * uvm_config_db#(int)            ::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
# ****************************************************************************
# UVM_INFO RAM_test_class.sv(52) @ 2: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO RAM_test_class.sv(55) @ 2: uvm_test_top [run_phase] Stimulus Generation Started
# UVM_INFO RAM_test_class.sv(57) @ 18002: uvm_test_top [run_phase] Stimulus Generation Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 18002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO RAM_scoreboard_class.sv(65) @ 18002: uvm_test_top.env.sb [report_phase] Total successful transictions: 9001
# UVM_INFO RAM_scoreboard_class.sv(66) @ 18002: uvm_test_top.env.sb [report_phase] Total failed transictions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   11
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [UVMTOP]     1
# [report_phase]     2
# [run_phase]     4
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 18002 ns  Iteration: 61  Instance: /top
# 1
# 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

VSIM 2>
```

Figure 24: RAM transcript.

PART 2: SPI

Verification Plan

1. Activate reset at the beginning to set the program to a default state which is IDLE state.
2. Randomize rst_n to be deactivated most of the time to determine SPI behavior on resetting.
3. Randomize SS_n to be activated to complete full cycle of one operation then end communication again.
4. After resetting, the transition will be from the IDLE state to CHK_CMD state when the master starts the communication (SS_n = 0).
5. After reaching the CHK_CMD state the MOSI will be randomized to different sequences of bits and stored serially at temp register to check on the different operations of the slave.
6. When the transition done from CHK_CMD state to one of the following states (WRITE, READ_ADD, READ_DATA) the slave will save the 10 bits sequence of MOSI at rx_data bus and raise rx_valid to send it to the RAM to analyze it to the required operation.
7. Check on the read_address_flag that control the transition to READ_ADD and READ_DATA such that the master should send the required address location of the data before requesting the data.
8. Check that at the READ_DATA state, after sending the bus to the memory, the memory will respond by the data required and raise tx_valid signal then slave will receive the 8 bits data from the tx_data bus and output it serially on the MISO port.
9. Check that when the master ends the communication (SS_n = 1) the slave will return to the IDLE state regardless of the past state.

Snippet of verification document

	A	B	C	D	E
1	**Label**	**Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
2	SPI_1	Incase of rst_n asserted, cs will be IDLE and MISO,rx_valid,rx_data,bus_rx,flags will be zero and counters to its initial values.	At the testbench initial block by making rst_n = 0 in 1 clock cycle then after checking the output at the monitor module, make rst_n = 1 then randomize it to be deactivated most of the time according to the constraint of rst_n.	Included in coverpoint to makesure that MISO will be zero when the rst_n is activated. Assertions included inside slave design to check that rx_valid and MISO will be zeros (outputs of the slave).	Output Checked at the monitor module with the golden model output.
3	SPI_2	Incase of rst_n deactivated, master starts the comm by asserting SS_n signal to complete full required operation passing through the states then ends comm again (SS_n = 1) to return to the IDLE state.	Randomization for SS_n under constraints to be activated limited number of cycles to give the slave the required time to complete the operation with the RAM and to be deactivated after completing the operation.	Included in assertion inside the slave design to check that when SS_n deactivated, the cs will be IDLE at the next cycle	Output Checked at the monitor module with the golden model output.
4	SPI_3	Incase of rst_n deactivated and SS_n activated, the master will send serially a required pattern of MOSI to identify the requierd operation to be done by the RAM	Randomization under constraints with counter specifier and post randomization function such that the first count at the MOSI pattern will be ignored when the transition done from IDLE to CHK_CMD, then the second count will determine the operation (write or read) the master will choose it randomly, then the third count, MOSI must equal to the past count MOSI, the fourth count, MOSI will be chosen based on flags such that the master should send an address before write or read a data, the next counts will be freely randomized to be an address or data or even ignored at the read data case.	Included in 3 different cross coverages : 1-All the different operations are coverd. 2-The counter will be zero when SS_n or rst_n asserted to send a new pattern again. 3- At the CHK_CMD the MOSI can not be (0 then 1)or(1 then 0) invalid logic. Included in 3 assertions to cover up that rx_data[9:8] takes correct values at the required operation (if READ_DATA then rx_data[9:8] = 2'b11) and so on.	Output Checked at the monitor module with the golden model output.

Figure 26: Verification document for SPI.

5	SPI_4	Incase of rst_n deactivated and SS_n activated, at the CHK_CMD state the slave will go to WRITE(which could be data or address) or READ_DATA or READ_ADD based on the MOSI and two control flags	flags values will be determined at the post randomization function according to the past operation done for example (if write address done then flag of writing operation will be asserted) and so on.	Included in 2 cover points to check that each flag has passed from high to low and vice versa. Included in 2 assertions to check that after finishing the read_add operation, it will be activated and after the read data operation, it will be deactivated.	Output Checked at the monitor module with the golden model output.
6	SPI_5	Incase of rst_n deactivated and SS_n activated, at the READ_ADD state the master should send an address which has been written before to write a data	After we saved all the written addresses at associative array, randomization under constraints with dist 90% to take an address has been written before.	No functional coverage	Output Checked at the monitor module with the golden model output.
7	SPI_6	Incase of rst_n deactivated and SS_n activated, at any state from the following (WRITE,READ_ADD,READ_DATA) the slave will save the 10 bits of MOSI serially at bus_rx then make it avialble on the rx_data bus after receiving the last bit and rx_valid will be high.	synchronized with the randomiztion of the MOSI at label 3	Included in 2 assertions to check that bus_rx has been written successfully and to check after receiving the last bit, the rx valid will be high and rx_data will be available to the RAM.	Output Checked at the monitor module with the golden model output.
8	SPI_7	Incase of rst_n deactivated and SS_n activated, at the READ_DATA state, after sending the rx_data to the RAM, the tx_data will be available to the slave to output it serially on the MISO port on 8 clock cycles.	No stimulus generation.	Included in 2 assertions to check that MISO will take all the bits of tx_data and after taking the last bit, the MISO will be set to zero (end of data).	Output Checked at the monitor module with the golden model output.

Figure 27: Verification document for SPI.

UVM Structure diagram

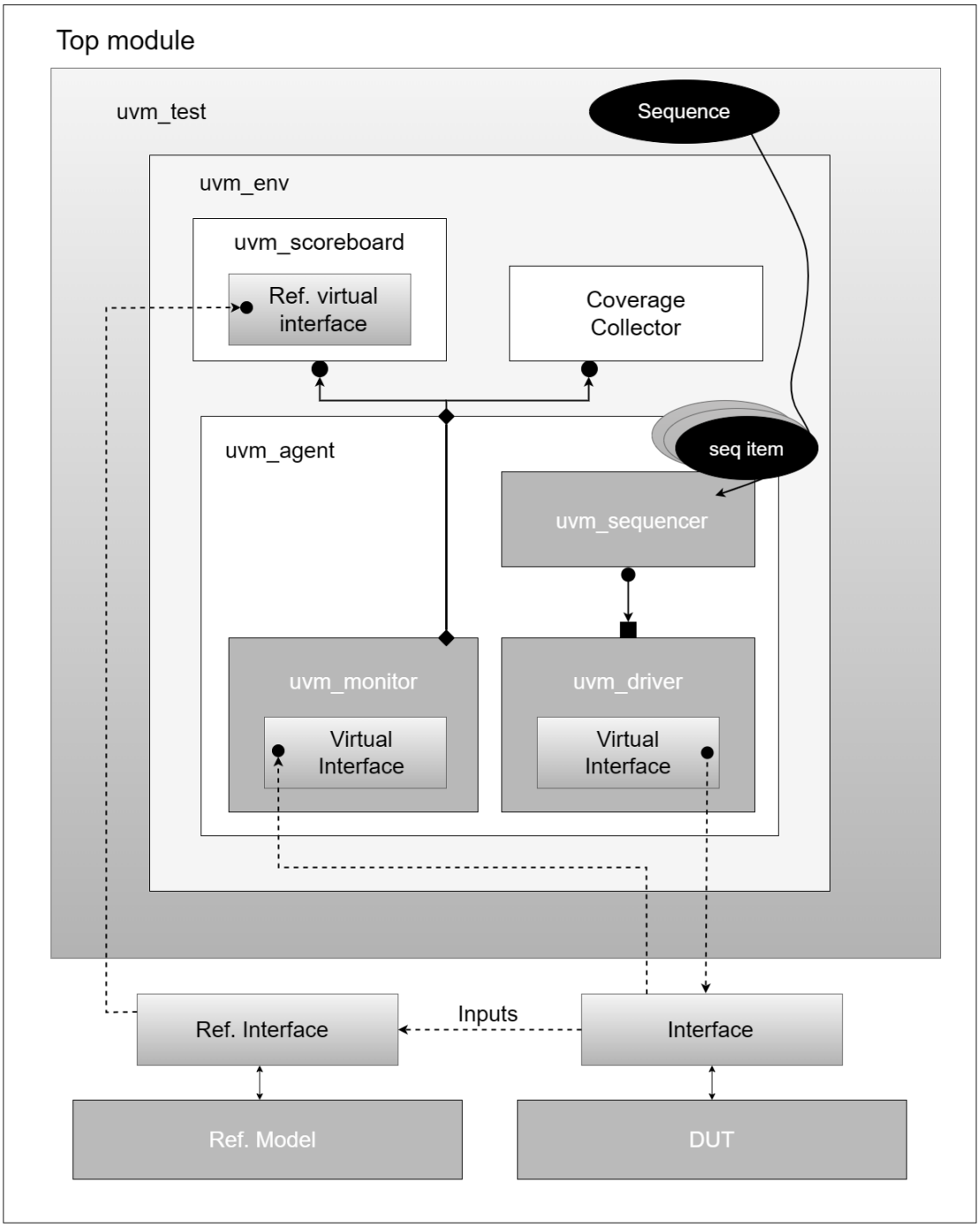


Figure 28: SPI UVM testbench structure diagram.

```
# UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
# ----------------------------------------------------------------
# Name                        Type                          Size  Value
# ----------------------------------------------------------------
# uvm_test_top                SPI_test                      -     @474
#   env                       SPI_env                       -     @481
#     agt                     SPI_agent                     -     @503
#       agt_ap                uvm_analysis_port             -     @665
#         recording_detail    integral                      32    'd1
#       driver                SPI_driver                    -     @635
#         rsp_port            uvm_analysis_port             -     @650
#           recording_detail  integral                      32    'd1
#         seq_item_port       uvm_seq_item_pull_port        -     @642
#           recording_detail  integral                      32    'd1
#         recording_detail    integral                      32    'd1
#       mon                   SPI_monitor                   -     @658
#         mon_ap              uvm_analysis_port             -     @673
#           recording_detail  integral                      32    'd1
#         recording_detail    integral                      32    'd1
#       sqr                   SPI_sequencer                 -     @526
#         rsp_export          uvm_analysis_export           -     @533
#           recording_detail  integral                      32    'd1
#         seq_item_export     uvm_seq_item_pull_imp         -     @627
#           recording_detail  integral                      32    'd1
#         recording_detail    integral                      32    'd1
#         arbitration_queue   array                         0     -
#         lock_queue          array                         0     -
#         num_last_reqs       integral                      32    'd1
#         num_last_rsps       integral                      32    'd1
#       recording_detail      integral                      32    'd1
#     cov                     SPI_coverage                  -     @517
#       cov_export            uvm_analysis_export           -     @686
#         recording_detail    integral                      32    'd1
#       cov_fifo              uvm_tlm_analysis_fifo #(T)    -     @694
#         analysis_export     uvm_analysis_imp              -     @733
#           recording_detail  integral                      32    'd1
#         get_ap              uvm_analysis_port             -     @725
#           recording_detail  integral                      32    'd1
#         get_peek_export     uvm_get_peek_imp              -     @709
#           recording_detail  integral                      32    'd1
#         put_ap              uvm_analysis_port             -     @717
#           recording_detail  integral                      32    'd1
#         put_export          uvm_put_imp                   -     @701
#           recording_detail  integral                      32    'd1
#         recording_detail    integral                      32    'd1
#       recording_detail      integral                      32    'd1
```

Figure 29: UVM testbench topology.

```
#     sb                      SPI_scoreboard                -     @510
#       sb_export             uvm_analysis_export           -     @741
#         recording_detail    integral                      32    'd1
#       sb_fifo               uvm_tlm_analysis_fifo #(T)    -     @749
#         analysis_export     uvm_analysis_imp              -     @788
#           recording_detail  integral                      32    'd1
#         get_ap              uvm_analysis_port             -     @780
#           recording_detail  integral                      32    'd1
#         get_peek_export     uvm_get_peek_imp              -     @764
#           recording_detail  integral                      32    'd1
#         put_ap              uvm_analysis_port             -     @772
#           recording_detail  integral                      32    'd1
#         put_export          uvm_put_imp                   -     @756
#           recording_detail  integral                      32    'd1
#         recording_detail    integral                      32    'd1
#       recording_detail      integral                      32    'd1
#     recording_detail        integral                      32    'd1
#   recording_detail          integral                      32    'd1
```

Figure 30: UVM testbench topology.

Coverage

Code coverage

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        43        43         0   100.00%

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

Branch Coverage for instance /\top#DUT /s1

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File SLAVE.sv
```

Figure 31: SPI wrapper branch coverage for SPI.

```
Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                       4         4         0   100.00%

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

Condition Coverage for instance /\top#DUT /s1 --
```

Figure 32: SPI wrapper condition coverage for SPI.

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

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

FSM Coverage for instance /\top#DUT /s1 --
```

Figure 33: SPI wrapper FSM coverage for SPI.

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      60        60         0   100.00%

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

Statement Coverage for instance /\top#DUT /s1 --
```

Figure 34: SPI wrapper statement coverage for SPI.

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        102       100         2    98.03%

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

Toggle Coverage for instance /\top#DUT /s1 --
```

Figure 35: SPI wrapper toggle coverage for SPI.

```
Toggle Coverage      =      98.03% (100 of 102 bins)

================================================================================
=== Instance: /\top#DUT /R1
=== Design Unit: work.Dual_port_RAM
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        11        11         0   100.00%

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

Branch Coverage for instance /\top#DUT /R1
```

Figure 36: SPI wrapper branch coverage for RAM.

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      17        17         0   100.00%

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

Statement Coverage for instance /\top#DUT /R1 --
```

Figure 37: SPI wrapper statement coverage for RAM.

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

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

Toggle Coverage for instance /\top#DUT /R1 --
```

Figure 38: SPI wrapper toggle coverage for RAM.

```
================================================================================
=== Instance: /\top#DUT
=== Design Unit: work.SPI_WRAPPER
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         50        50         0   100.00%

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

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

Figure 39: SPI wrapper toggle coverage.

Functional coverage

```
================================================================================
=== Instance: /SPI_coverage_pkg
=== Design Unit: work.SPI_coverage_pkg
================================================================================

Covergroup Coverage:
    Covergroups                     1        na        na   100.00%
        Coverpoints/Crosses        10        na        na        na
            Covergroup Bins        18        18         0   100.00%
--------------------------------------------------------------------------------------------------------------
Covergroup                                          Metric       Goal       Bins    Status

--------------------------------------------------------------------------------------------------------------
```

Figure 40: SPI functional coverage report.

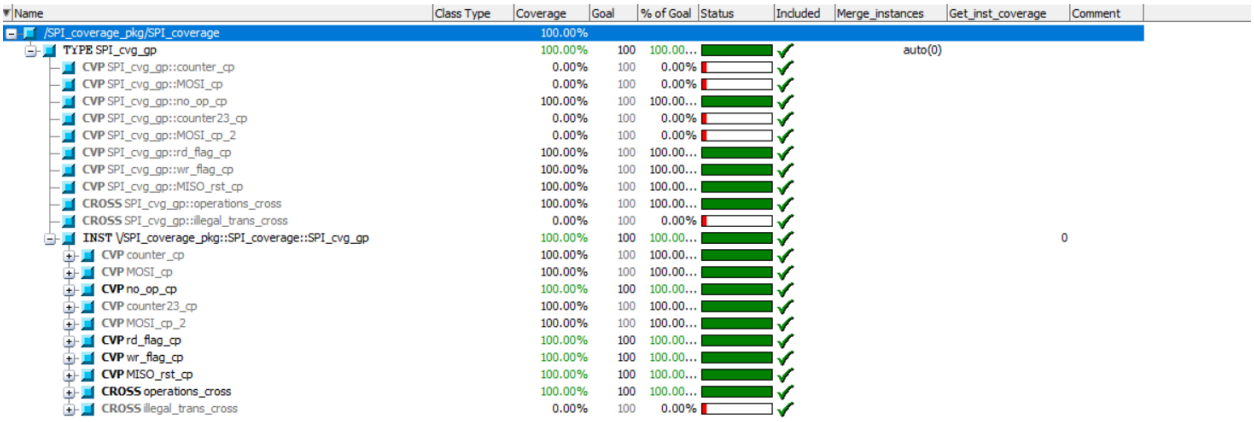


Figure 41: SPI functional coverage Questa snippet.

Assertions coverage

```
================================================================================
=== Instance: /\top#DUT /s1
=== Design Unit: work.slave
================================================================================

Assertion Coverage:
    Assertions                      11        11         0   100.00%
--------------------------------------------------------------------
Name                 File(Line)                    Failure      Pass
                                                   Count        Count
--------------------------------------------------------------------
/\top#DUT /s1/a_op_done_pr
                     SLAVE.sv(197)                       0          1
/\top#DUT /s1/a_reading_MOSI_pr
                     SLAVE.sv(207)                       0          1
/\top#DUT /s1/a_inactive_MISO_pr
                     SLAVE.sv(215)                       0          1
/\top#DUT /s1/a_active_MISO_pr
                     SLAVE.sv(224)                       0          1
/\top#DUT /s1/a_read_ad_flag_on_pr
                     SLAVE.sv(232)                       0          1
/\top#DUT /s1/a_read_ad_flag_off_pr
                     SLAVE.sv(240)                       0          1
/\top#DUT /s1/a_rxdata_read_addr_pr
                     SLAVE.sv(249)                       0          1
/\top#DUT /s1/a_rxdata_read_data_pr
```

Figure 42: SPI assertions coverage report.

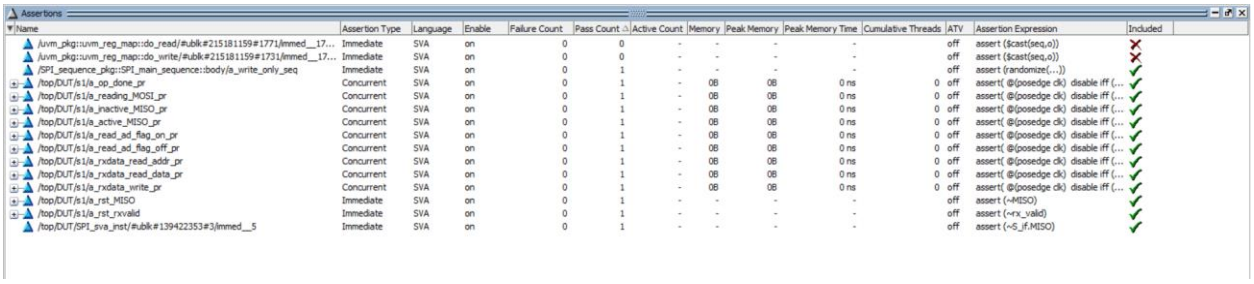


Figure 43: SPI assertions.

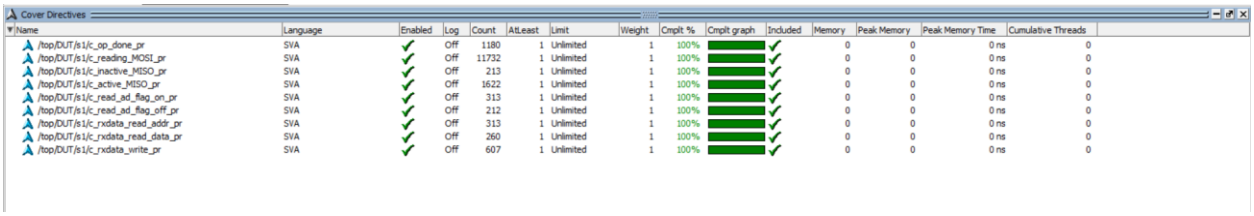


Figure 44: SPI assertions directive.

Bugs reporting

❖ Variables bugs

- read_ad_flag starts with x and does not change its value along the simulation time which causes no control between the READ_ADD and READ_DATA states.
- Write_flag is not a used signal at all (There is no statement to use or assign it).
- Cs, ns are declared as one bit reg which is not sufficient to cover up all the states.

```
18  reg read_ad_flag;//if 1 the check command will go to the read adress
19  reg write_flag;
20  reg [3:0] counter_tx;
21  reg [3:0]counter_rx=4'b1001;
22  reg [9:0] bus_rx;
23
24  reg cs,ns;
```

Figure 45: Prefixed Variables bugs of slave design.

```
16  reg read_ad_flag;//if 1 the check command will go to the read adress if Zero the heya aret 1 adress yeb2a hya hatkteb now
17  reg [3:0] counter_tx;
18  reg [3:0] counter_rx;
19  reg [9:0] bus_rx;
20
21
22  bit sending_flag;
23
24  reg [2:0] cs,ns;
25
```

Figure 46: Postfixed cs & ns bug of slave design.

```
89  //output logic
90  always @(posedge clk or negedge rst_n) begin
91
92    if(!rst_n) begin
93      MISO <= 0;
94      read_ad_flag <= 0;
95      sending_flag <= 0;
96      counter_tx   <=4'b1000;
97      counter_rx   <=4'b1010;
98      rx_valid     <=0;
99      bus_rx       <=0;
100     rx_data      <=0;
101
102   end
```

Figure 47: Postfixed non-resetting read_ad_flag bug.

❖ Next state always blocks bug

Since the MOSI is not inserted at the sensitivity list, the transition from the CHK_CMD state to (WRITE, READ_ADD, READ_DATA) states will be based on the last MOSI driven.

```
27  //next state logic
28  always @(SS_n,cs) begin //ask sensityvityy listttttt????????
29
```

Figure 48: Prefixed next state always block bug of slave design.

```
25  //next state logic
26  always @(SS_n,cs,MOSI) begin //ask sensityvityy listttttt????????
```

Figure 49: Postfixed next state always block bug of slave design.

❖ Output always block bugs

- At fig (101): the sensitivity list does not have the rst effect and it should be taken into consideration to set some of slave outputs and internal regs like flags, counters, rx_valid and rx_data.
- At fig (101) line (108): at the IDLE state MISO should be set to zero.
- At fig (101): at the CHK_CMD state the logic of given the read_ad_flag is totally wrong as at the IDLE state it sets rx_data to zero then the two conditions is always false and the value of the flag will not change across the simulation time.
- At fig (101 & 102): at the WRITE or READ_ADD states it writes at bus_rx only 9 bits as the counter equal 0, it will send the bus without overwriting the tenth bit.
- At fig (102): at the READ_DATA state the logic is totally wrong as it does not receive the bits of MOSI which will be sent to the RAM to be able to give the dout at the bus of rx_data.
- At fig (102): at the READ_DATA state it depends on reading the data from the bus is to be available as long as the tx_valid is high but it should be high for one clock cycle only.
- At fig (102): at the READ_DATA state it reads from tx_data 7 bits only as the counter equal 0, it will reset counter and output zero on MISO port at the last cycle of READ_DATA.

```
105  //output logic
106  always @(posedge clk ) begin
107     case (cs)
108        IDLE:      begin
109                      rx_data=10'b00000_00000;//ask
110                      counter_tx=4'b0111;
111                      counter_rx=4'b1001;
112                      rx_valid=0;
113                   end
114
115        CHK_CMD:
116                   if(rx_data[9:8]==2'b10) begin//read condition
117                      read_ad_flag=1;
118                   end
119                   else if (rx_data[9:8]==2'b11) begin
120                      read_ad_flag=0;
121                   end
122                   //law la dah wala dah then ns hyroh ll write
123        WRITE:
124                   if (counter_rx==0) begin//from serial to parralel
125                      rx_valid=1;
126                      rx_data=bus_rx;
127                      counter_rx=4'b1001;
128                   end
129                   else begin
130                      bus_rx[counter_rx]= MOSI;
131                      counter_rx=counter_rx-1;
132                   end
133
```

Figure 50: Prefixed output always block bugs in slave design part 1.

```
134         READ_ADD:
135
136                 if (counter_rx==0) begin//from serial to parralel
137                    rx_valid=1;
138                    rx_data=bus_rx;
139                    counter_rx=4'b1001;
140                 end
141                 else begin
142                    bus_rx[counter_rx]= MOSI;
143                    counter_rx=counter_rx-1;
144                 end
145
146         READ_DATA:
147
148                 if (tx_valid==1) begin
149
150                    if (counter_tx==0) begin//from parralel to serial
151                       counter_tx=4'b0111;
152                    end
153                    else begin
154                       MISO=tx_data[counter_tx];
155                       counter_tx=counter_tx-1;
156                    end
157                 end
158
159                    else if(tx_valid==0) begin
160                    MISO=0;
161                    end
162     endcase
163  end//end of always block
```

Figure 51: Prefixed output always block bugs in slave design part 2.

❖ Postfixed discussed bugs

```
88  //output logic
89  always @(posedge clk or negedge rst_n) begin
90
91    if(!rst_n) begin
92      MISO <= 0;
93      read_ad_flag <= 0;
94      sending_flag <= 0;
95      counter_tx   <=4'b1000;
96      counter_rx   <=4'b1010;
97      rx_valid     <=0;
98      bus_rx       <=0;
99      rx_data      <=0;
100
101   end
102
103   else begin
104     case (cs)
105       IDLE: begin
106        MISO         <= 0;
107        counter_tx   <=4'b1000;
108        counter_rx   <=4'b1010;
109        rx_valid     <=0;
110        sending_flag <=0;
111        bus_rx       <=0;
112        rx_data      <=0;
113        end
114
115           WRITE:
116       if (counter_rx==0) begin//from serial to parralel
117           rx_valid    <=1;
118           rx_data     <=bus_rx;
119           counter_rx  <=4'b1010;
120           end
121
122           else begin
123           bus_rx[counter_rx-1] <= MOSI;
124           counter_rx <= counter_rx-1;
```

Figure 52: Postfixed output always block bugs in slave design part 1.

```
125         end
126
127           READ_ADD:
128
129         if (counter_rx==0) begin//from serial to parralel
130             rx_valid   <=1;
131             rx_data    <=bus_rx;
132             counter_rx <=4'b1010;
133
134           read_ad_flag <= 1'b1;
135             end
136
137             else begin
138             bus_rx[counter_rx-1] <= MOSI;
139             counter_rx           <= counter_rx-1;
140         end
141
142         READ_DATA: begin
143
144           if (tx_valid==1) begin
145             sending_flag = 1'b1;
146             rx_valid     = 1'b0;
147           end
148
149
150           if (sending_flag) begin
151               if (counter_tx==0) begin   //from parralel to serial
152                 counter_tx <= 4'b1000;
153                 MISO <=0;
154
155                 read_ad_flag <= 1'b0;
156               end
157               else begin
158                 MISO       <= tx_data[counter_tx-1];
159                 counter_tx <= counter_tx-1;
160               end
161           end
```

Figure 53: Postfixed output always block bugs in slave design part 2.

```
162          else begin
163
164              MISO <= 0;
165
166              if (counter_rx==0) begin  //from serial to parralel
167                rx_valid   <=1;
168                rx_data    <=bus_rx;
169                counter_rx <= 4'b1010;
170              end
171
172              else if (~rx_valid) begin
173                bus_rx[counter_rx-1] <= MOSI;
174                counter_rx           <= counter_rx-1;
175          end
176          else
177            rx_valid <= 0;
178
179          end
180        end
181
182        default:
183          MISO <= 0;
184
185        endcase
186    end //else
187  end                    //end of always block
```

Figure 54: Postfixed output always block bugs in slave design part 3.

Waveform

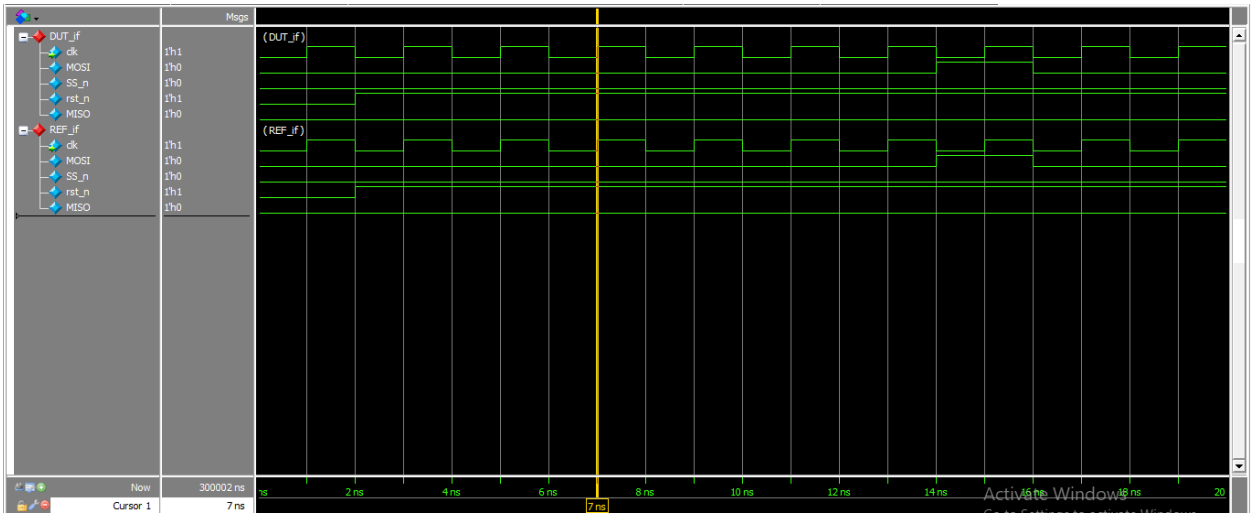


Figure 55: SPI_WRAPPER waveform snippet.

Transcript

```
#
# UVM_INFO SPI_test_class.sv(49) @ 0: uvm_test_top [run_phase] Reset Asserted
# ****************************************************************************
# * Questa UVM Transaction Recording Turned ON.                              *
# * recording_detail has been set.                                           *
# *  To turn off, set 'recording_detail' to off:                             *
# * uvm_config_db#(int)            ::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
# ****************************************************************************
# UVM_INFO SPI_test_class.sv(51) @ 2: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO SPI_test_class.sv(54) @ 2: uvm_test_top [run_phase] Stimulus Generation Started
# UVM_INFO SPI_test_class.sv(56) @ 120002: uvm_test_top [run_phase] Stimulus Generation Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 120002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO SPI_scoreboard_class.sv(65) @ 120002: uvm_test_top.env.sb [report_phase] Total successful transictions: 60001
# UVM_INFO SPI_scoreboard_class.sv(66) @ 120002: uvm_test_top.env.sb [report_phase] Total failed transictions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   11
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [UVMTOP]     1
# [report_phase]     2
# [run_phase]     4
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 120002 ns  Iteration: 61  Instance: /top
# 1
# 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 56: SPI transcript.

PART 3:

UVM Structure diagram

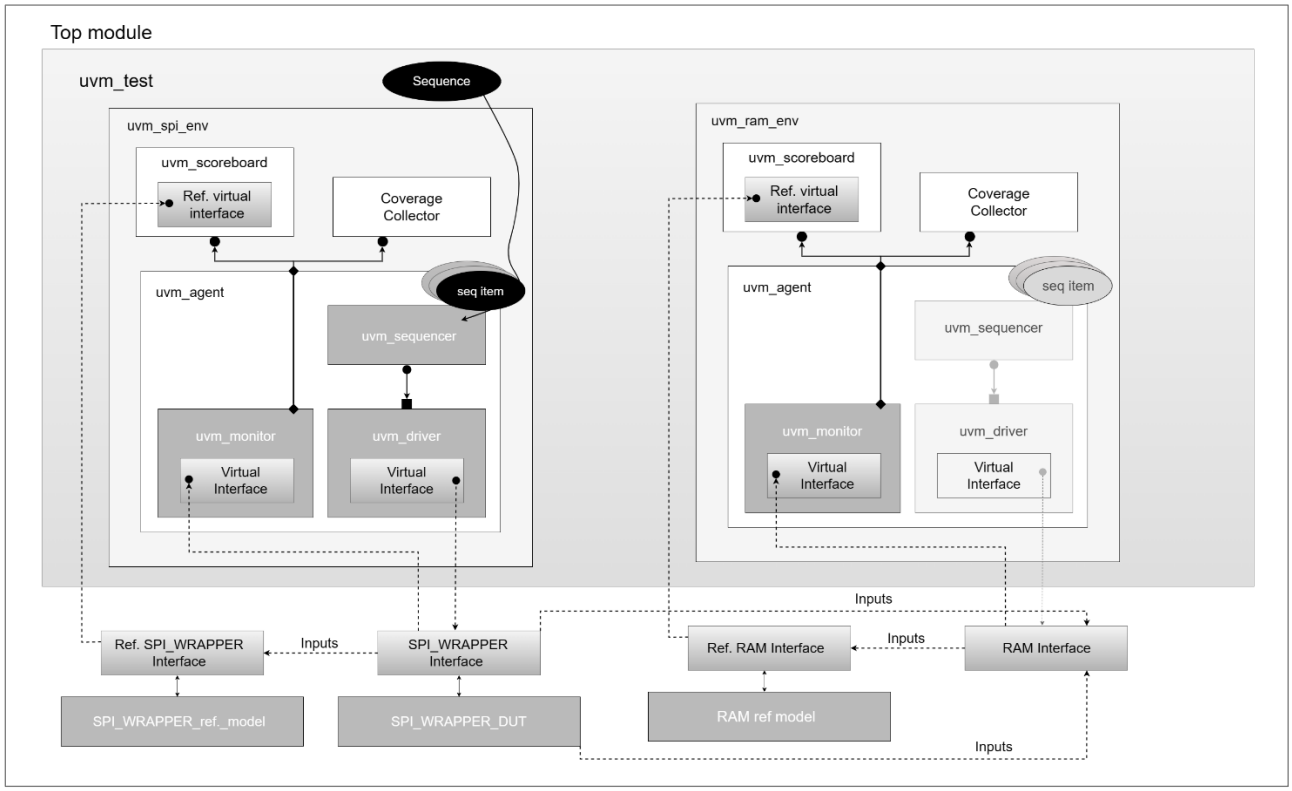


Figure 57: SPI_WRAPPER with merged RAM and SPI environments UVM testbench structure diagram.

```
# UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
# ----------------------------------------------------------------
# Name                        Type                        Size  Value
# ----------------------------------------------------------------
# uvm_test_top                SPI_test                    -     @482
#   ram_env                   RAM_env                     -     @496
#     agt                     RAM_agent                   -     @525
#       agt_ap                uvm_analysis_port           -     @555
#         recording_detail    integral                    32    'd1
#       mon                   RAM_monitor                 -     @548
#         mon_ap              uvm_analysis_port           -     @563
#           recording_detail  integral                    32    'd1
#         recording_detail    integral                    32    'd1
#       recording_detail      integral                    32    'd1
#     cov                     RAM_coverage                -     @539
#       cov_export            uvm_analysis_export         -     @571
#         recording_detail    integral                    32    'd1
#       cov_fifo              uvm_tlm_analysis_fifo #(T)  -     @579
#         analysis_export     uvm_analysis_imp            -     @618
#           recording_detail  integral                    32    'd1
#         get_ap              uvm_analysis_port           -     @610
#           recording_detail  integral                    32    'd1
#         get_peek_export     uvm_get_peek_imp            -     @594
#           recording_detail  integral                    32    'd1
#         put_ap              uvm_analysis_port           -     @602
#           recording_detail  integral                    32    'd1
#         put_export          uvm_put_imp                 -     @586
#           recording_detail  integral                    32    'd1
#         recording_detail    integral                    32    'd1
#       recording_detail      integral                    32    'd1
#     sb                      RAM_scoreboard              -     @532
#       sb_export             uvm_analysis_export         -     @626
#         recording_detail    integral                    32    'd1
#       sb_fifo               uvm_tlm_analysis_fifo #(T)  -     @634
#         analysis_export     uvm_analysis_imp            -     @673
#           recording_detail  integral                    32    'd1
#         get_ap              uvm_analysis_port           -     @665
#           recording_detail  integral                    32    'd1
#         get_peek_export     uvm_get_peek_imp            -     @649
#           recording_detail  integral                    32    'd1
#         put_ap              uvm_analysis_port           -     @657
#           recording_detail  integral                    32    'd1
#         put_export          uvm_put_imp                 -     @641
#           recording_detail  integral                    32    'd1
#         recording_detail    integral                    32    'd1
#       recording_detail      integral                    32    'd1
#     recording_detail        integral                    32    'd1
```

Figure 58: UVM testbench topology.

```
#  spi_env                  SPI_env                    -    @489
#    agt                    SPI_agent                  -    @683
#      agt_ap               uvm_analysis_port          -    @845
#        recording_detail   integral                   32   'd1
#      driver               SPI_driver                 -    @815
#        rsp_port           uvm_analysis_port          -    @830
#          recording_detail integral                   32   'd1
#        seq_item_port      uvm_seq_item_pull_port     -    @822
#          recording_detail integral                   32   'd1
#        recording_detail   integral                   32   'd1
#      mon                  SPI_monitor                -    @838
#        mon_ap             uvm_analysis_port          -    @853
#          recording_detail integral                   32   'd1
#        recording_detail   integral                   32   'd1
#      sqr                  SPI_sequencer              -    @706
#        rsp_export         uvm_analysis_export        -    @713
#          recording_detail integral                   32   'd1
#        seq_item_export    uvm_seq_item_pull_imp      -    @807
#          recording_detail integral                   32   'd1
#        recording_detail   integral                   32   'd1
#        arbitration_queue  array                      0    -
#        lock_queue         array                      0    -
#        num_last_reqs      integral                   32   'd1
#        num_last_rsps      integral                   32   'd1
#      recording_detail     integral                   32   'd1
#    cov                    SPI_coverage               -    @697
#      cov_export           uvm_analysis_export        -    @866
#        recording_detail   integral                   32   'd1
#      cov_fifo             uvm_tlm_analysis_fifo #(T) -    @874
#        analysis_export    uvm_analysis_imp           -    @913
#          recording_detail integral                   32   'd1
#        get_ap             uvm_analysis_port          -    @905
#          recording_detail integral                   32   'd1
#        get_peek_export    uvm_get_peek_imp           -    @889
#          recording_detail integral                   32   'd1
#        put_ap             uvm_analysis_port          -    @897
#          recording_detail integral                   32   'd1
#        put_export         uvm_put_imp                -    @881
#          recording_detail integral                   32   'd1
#        recording_detail   integral                   32   'd1
#      recording_detail     integral                   32   'd1
```

Figure 59: UVM testbench topology.

```
#    sb                     SPI_scoreboard             -    @690
#      sb_export            uvm_analysis_export        -    @921
#        recording_detail   integral                   32   'd1
#      sb_fifo              uvm_tlm_analysis_fifo #(T) -    @929
#        analysis_export    uvm_analysis_imp           -    @968
#          recording_detail integral                   32   'd1
#        get_ap             uvm_analysis_port          -    @960
#          recording_detail integral                   32   'd1
#        get_peek_export    uvm_get_peek_imp           -    @944
#          recording_detail integral                   32   'd1
#        put_ap             uvm_analysis_port          -    @952
#          recording_detail integral                   32   'd1
#        put_export         uvm_put_imp                -    @936
#          recording_detail integral                   32   'd1
#        recording_detail   integral                   32   'd1
#      recording_detail     integral                   32   'd1
#    recording_detail       integral                   32   'd1
#  recording_detail         integral                   32   'd1
```

Figure 60: UVM testbench topology.

Coverage

Code coverage

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        43        43         0   100.00%

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

Branch Coverage for instance /\top#DUT /s1
```

Figure 61: Branch coverage for SPI.

```
Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                       4         4         0   100.00%

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

Condition Coverage for instance /\top#DUT /s1 --
```

Figure 62: Conditional coverage for SPI.

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

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

FSM Coverage for instance /\top#DUT /s1 --
```

Figure 63: FSM coverage for SPI.

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      60        60         0   100.00%

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

Statement Coverage for instance /\top#DUT /s1 --
```

Figure 64: Statement coverage for SPI.

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        102       102         0   100.00%

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

Toggle Coverage for instance /\top#DUT /s1 --
```

Figure 65: Toggle coverage for SPI.

```
============================================================================
=== Instance: /\top#DUT /R1
=== Design Unit: work.Dual_port_RAM
============================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        11        11         0   100.00%

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

Figure 66: Branch coverage for RAM.

```
Statement Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Statements                     17        17         0   100.00%

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

Statement Coverage for instance /\top#DUT /R1 --
NOTE: The modification timestamp for source file 'Dual_port_RAM.sv' has been altered since compilation.

    Line         Item                      Count     Source
```

Figure 67: Statement coverage for RAM.

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

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

Toggle Coverage for instance /\top#DUT /R1 --
```

Figure 68: Toggle coverage for RAM.

```
============================================================================
=== Instance: /\top#DUT
=== Design Unit: work.SPI_WRAPPER
============================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         50        50         0   100.00%

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

Figure 69: Toggle coverage for SPI wrapper.

Functional coverage

```
================================================================================
=== Instance: /RAM_coverage_pkg
=== Design Unit: work.RAM_coverage_pkg
================================================================================

Covergroup Coverage:
    Covergroups                 1        na        na   100.00%
        Coverpoints/Crosses     8        na        na        na
            Covergroup Bins   526       526         0   100.00%
--------------------------------------------------------------------------------------------------------------
Covergroup                                        Metric       Goal       Bins    Status

--------------------------------------------------------------------------------------------------------------
```

Figure 70: Functional coverage for RAM.

```
================================================================================
=== Instance: /SPI_coverage_pkg
=== Design Unit: work.SPI_coverage_pkg
================================================================================

Covergroup Coverage:
    Covergroups                 1        na        na   100.00%
        Coverpoints/Crosses    10        na        na        na
            Covergroup Bins    18        18         0   100.00%
--------------------------------------------------------------------------------------------------------------
Covergroup                                       Metric       Goal       Bins    Status

--------------------------------------------------------------------------------------------------------------
```

Figure 71: Functional coverage for SPI.

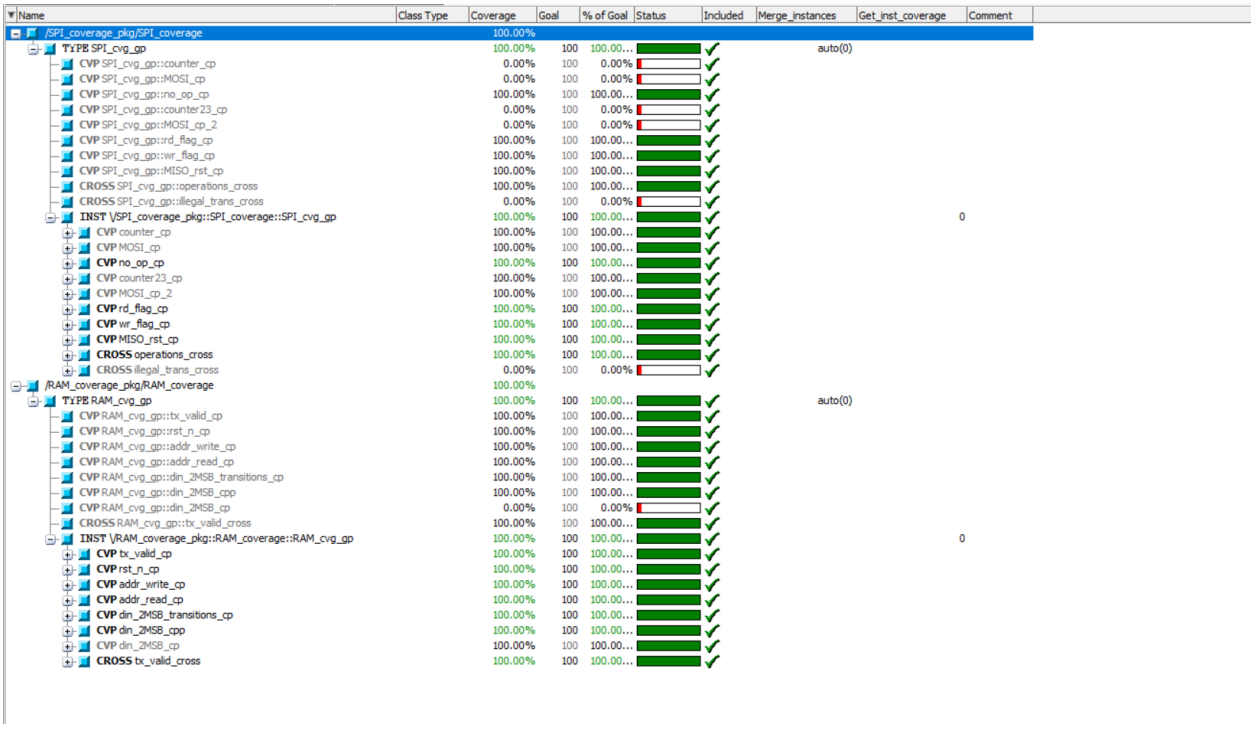
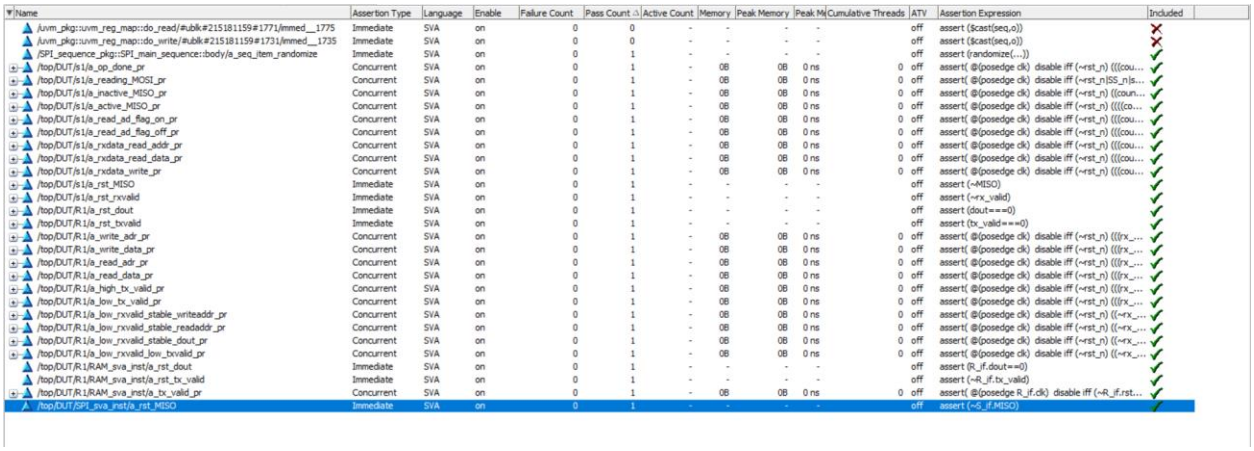


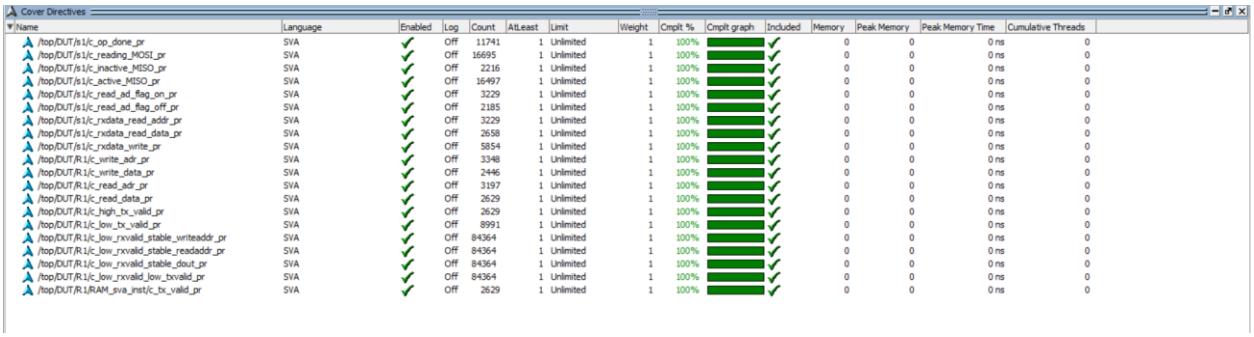
Figure 72: Functional coverage for SPI and RAM questa snippet.

Assertions coverage



Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Mi	Cumulative Threads	ATV	Assertion Expression	Included
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/SPI_sequence_pkg::SPI_main_sequence::body/a_seq_item_randomize	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/top/DUT/s1/a_op_done_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((cou...	✓
/top/DUT/s1/a_reading_MOSI_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n)SS_n)s...	✓
/top/DUT/s1/a_inactive_MISO_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((coun...	✓
/top/DUT/s1/a_active_MISO_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((co...	✓
/top/DUT/s1/a_read_ad_flag_on_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((cou...	✓
/top/DUT/s1/a_read_ad_flag_off_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((cou...	✓
/top/DUT/s1/a_rxdata_read_addr_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((cou...	✓
/top/DUT/s1/a_rxdata_read_data_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((cou...	✓
/top/DUT/s1/a_rxdata_write_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((cou...	✓
/top/DUT/s1/a_rst_MISO	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~MISO)	✓
/top/DUT/s1/a_rst_rxvalid	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~rx_valid)	✓
/top/DUT/R1/a_rst_dout	Immediate	SVA	on	0	1	-	-	-	-		off	assert (dout===0)	✓
/top/DUT/R1/a_rst_txvalid	Immediate	SVA	on	0	1	-	-	-	-		off	assert (tx_valid===0)	✓
/top/DUT/R1/a_write_adr_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((rx_...	✓
/top/DUT/R1/a_write_data_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((rx_...	✓
/top/DUT/R1/a_read_adr_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((rx_...	✓
/top/DUT/R1/a_read_data_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((rx_...	✓
/top/DUT/R1/a_high_tx_valid_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((rx_...	✓
/top/DUT/R1/a_low_tx_valid_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) ((((rx_...	✓
/top/DUT/R1/a_low_rxvalid_stable_writeaddr_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) (((~rx_...	✓
/top/DUT/R1/a_low_rxvalid_stable_readaddr_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) (((~rx_...	✓
/top/DUT/R1/a_low_rxvalid_stable_dout_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) (((~rx_...	✓
/top/DUT/R1/a_low_rxvalid_low_txvalid_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_n) (((~rx_...	✓
/top/DUT/R1/RAM_sva_inst/a_rst_dout	Immediate	SVA	on	0	1	-	-	-	-		off	assert (R_if.dout==0)	✓
/top/DUT/R1/RAM_sva_inst/a_rst_tx_valid	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~R_if.tx_valid)	✓
/top/DUT/R1/RAM_sva_inst/a_tx_valid_pr	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge R_if.clk) disable iff (~R_if.rst...	✓
/top/DUT/SPI_sva_inst/a_rst_MISO	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~S_if.MISO)	✓

Figure 73: Assertions coverage questa snippet.



Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top/DUT/s1/c_op_done_pr	SVA	✓	Off	11741	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/s1/c_reading_MOSI_pr	SVA	✓	Off	16695	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/s1/c_inactive_MISO_pr	SVA	✓	Off	2216	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/s1/c_active_MISO_pr	SVA	✓	Off	16497	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/s1/c_read_ad_flag_on_pr	SVA	✓	Off	3229	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/s1/c_read_ad_flag_off_pr	SVA	✓	Off	2185	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/s1/c_rxdata_read_addr_pr	SVA	✓	Off	3229	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/s1/c_rxdata_read_data_pr	SVA	✓	Off	2658	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/s1/c_rxdata_write_pr	SVA	✓	Off	5854	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/R1/c_write_adr_pr	SVA	✓	Off	3348	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/R1/c_write_data_pr	SVA	✓	Off	2446	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/R1/c_read_adr_pr	SVA	✓	Off	3197	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/R1/c_read_data_pr	SVA	✓	Off	2629	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/R1/c_high_tx_valid_pr	SVA	✓	Off	2629	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/R1/c_low_tx_valid_pr	SVA	✓	Off	8991	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/R1/c_low_rxvalid_stable_writeaddr_pr	SVA	✓	Off	84364	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/R1/c_low_rxvalid_stable_readaddr_pr	SVA	✓	Off	84364	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/R1/c_low_rxvalid_stable_dout_pr	SVA	✓	Off	84364	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/R1/c_low_rxvalid_low_txvalid_pr	SVA	✓	Off	84364	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/R1/RAM_sva_inst/c_tx_valid_pr	SVA	✓	Off	2629	1	Unlimited	1	100%		✓	0	0	0 ns	0

Figure 74: Assertions cover directives questa snippets.

Waveform

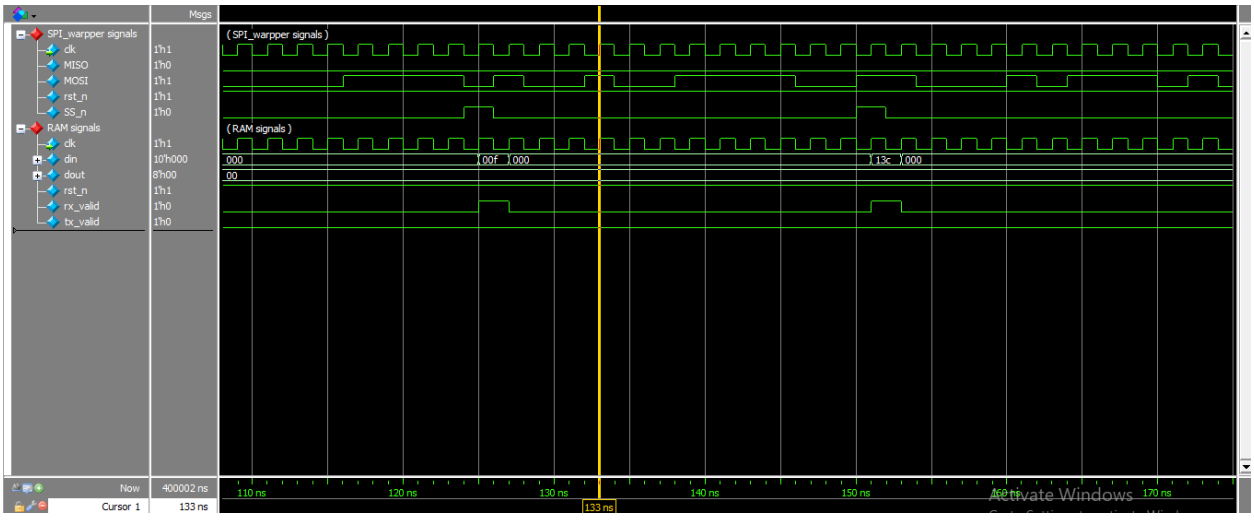


Figure 75: write address followed by write data.

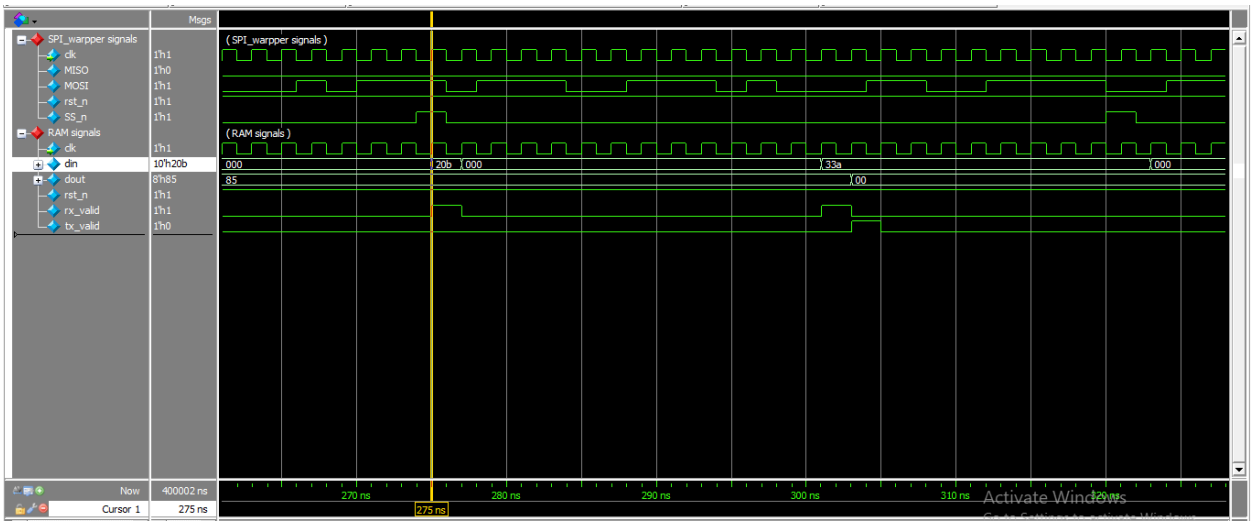


Figure 76: read address followed by read data.

Transcript

```
#
# UVM_INFO SPI_test_class.sv(73) @ 0: uvm_test_top [run_phase] Reset Asserted
# ********************************************************************************
# * Questa UVM Transaction Recording Turned ON.                                  *
# * recording_detail has been set.                                               *
# *  To turn off, set 'recording_detail' to off:                                 *
# * uvm_config_db#(int)            ::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
# ********************************************************************************
# UVM_INFO SPI_test_class.sv(75) @ 2: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO SPI_test_class.sv(78) @ 2: uvm_test_top [run_phase] Stimulus Generation Started
# UVM_INFO SPI_test_class.sv(80) @ 400002: uvm_test_top [run_phase] Stimulus Generation Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 400002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO RAM_scoreboard_class.sv(65) @ 400002: uvm_test_top.ram_env.sb [report_phase] Total successful transictions: 200001
# UVM_INFO RAM_scoreboard_class.sv(66) @ 400002: uvm_test_top.ram_env.sb [report_phase] Total failed transictions: 0
# UVM_INFO SPI_scoreboard_class.sv(65) @ 400002: uvm_test_top.spi_env.sb [report_phase] Total successful transictions: 200001
# UVM_INFO SPI_scoreboard_class.sv(66) @ 400002: uvm_test_top.spi_env.sb [report_phase] Total failed transictions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   13
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [UVMTOP]     1
# [report_phase]     4
# [run_phase]     4
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 400002 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 70: Part3 transcript.

RAM Assertions (inside DUAL_port_RAM)

Feature	Assertions
At positive edge of clock Whenever the reset is deasserted , the rx_valid is high and din[9:8]=2'b00 Check the execution of the address operation where din[7:0] is written in the write address.	@(posedge clk) disable iff (~rst_n) (rx_valid && din[9:8] == 2'b00) |=> (write_adr == $past(din[7:0]));
At positive edge of clock Whenever the reset is deasserted and the rx_valid is high and din[9:8]=2'b01 Then the write data operation is executed and din[7:0] Is written into the write address.	@(posedge clk) disable iff (~rst_n) (rx_valid && din[9:8] == 2'b01) |=> (mem[write_adr] === $past(din[7:0]));
At positive edge of clock Whenever the reset is deasserted , rx_valid is high and din[9:8] =2'b10 Then the read address operation is executed and din[7:0] Is written into the read address.	@(posedge clk) disable iff (~rst_n) (rx_valid && din[9:8] == 2'b10) |=> (read_adr === $past(din[7:0]));
At positive edge of clock Whenever the reset is deasserted ,rx_valid is high and din[9:8] =2'b11 Then the read data operation is executed and data in memory of location stored in read address Is read on dout port.	@(posedge clk) disable iff (~rst_n) (rx_valid && din[9:8] == 2'b11) |=> (dout === mem[read_adr]);
At positive edge of clock Whenever the reset is deasserted , rx_valid is high and din[9:8] =2'b11, then tx_valid should be high .	@(posedge clk) disable iff (~rst_n) (rx_valid && din[9:8] == 2'b11) |=> (tx_valid);

At positive edge of clock Whenever the reset is deasserted , rx_valid is high and din[9:8] not equal 2'b11 then tx_valid should be low.	@(posedge clk) disable iff (~rst_n) (rx_valid && din[9:8] != 2'b11) |=> (~tx_valid);
At positive edge of clock Whenever the reset is deasserted , rx_valid is low we should make sure that no operation is done and write address stays the same.	@(posedge clk) disable iff (~rst_n) (~rx_valid) |=> (write_adr === $past(write_adr));
At positive edge of clock Whenever the reset is deasserted , rx_valid is low we should make sure that no operation is done and read address stays the same.	@(posedge clk) disable iff (~rst_n) (~rx_valid) |=> (read_adr === $past(read_adr));
At positive edge of clock Whenever the reset is deasserted , rx_valid is low then output on dout port is not changed.	@(posedge clk) disable iff (~rst_n) (~rx_valid) |=> (dout === $past(dout));
At positive edge of clock Whenever the reset is deasserted , rx_valid is low then tx_valid =0	@(posedge clk) disable iff (~rst_n) (~rx_valid) |=> (tx_valid === 0);

When reset is asserted (rst_n=0), then there is no opertation, dout and tx_valid equal zero.	if (!rst_n) begin
	a_rst_dout: assert final (dout === 0);
	a_rst_txvalid: assert final (tx_valid === 0);
	end

RAM_Assertions (inside RAM_sva)

Feature	Assertions

Whenever the reset value is asserted, the output on dout port is zero and tx_valid should be low.	always_comb begin
	if(!R_if.rst_n) begin
	a_rst_dout: assert final (R_if.dout == 0);
	a_rst_tx_valid: assert final (R_if.tx_valid == 0);
	end
	end

At polsedge of clock if reset is deasserted, rx_valid is high and operation is read data then tx_valid is expected to be high in the next clock cycle.	@(posedge R_if.clk) disable iff(!R_if.rst_n) (R_if.din[9:8] == READ_DATA && R_if.rx_valid) |=> R_if.tx_valid;

Slave_assertions (Inside SlAVE file, the design file itself)

Feature	Assertions
At positive edge of clock if reset is deasserted, counter_rx=0 and current state is not IDLE then the rx_valid should be high and the input data is available on input data bus of RAM .	@(posedge clk) disable iff (~rst_n) ((counter_rx == 0 && cs != IDLE) |=> (rx_valid && (rx_data === bus_rx)));
At positive edge of clock if reset is asserted or SSn=1 or sending_flag =1 or rx_valid=1, the assertion will be disabled else if counter_rx !=0, current state is in read or write operation then the input data bus of RAM should be taking the values of MOSI.	@(posedge clk) disable iff (~rst_n || SS_n || sending_flag || rx_valid) ((counter_rx != 0)&&(cs != CHK_CMD) && (cs != IDLE)) |=> (bus_rx[$past(counter_rx) - 1] == $past(MOSI));
At positive edge of clock if reset is deasserted ,counter_tx=0 then all data is read so MISO is off (equals zero indicating no data) .	@(posedge clk) disable iff (~rst_n) (counter_tx == 0) |=> ~MISO;
At positive edge of clock if reset is deasserted if counter_rx! =0,current state is read data and sending flag equals one then the data is transmitted from parallel to serial at output port MISO.	@(posedge clk) disable iff (~rst_n) (counter_tx != 0 && cs == READ_DATA && sending_flag) |=> (MISO === tx_data [$past(counter_tx) -1]);
At positive edge of clock if reset is deasserted , When the read address operation is done Then the read address flag is asserted to make sure that the next state is read data not read address .	@(posedge clk) disable iff (~rst_n) (counter_rx == 0 && (cs == READ_ADD)) |=> read_ad_flag;
At positive edge of clock if reset is deasserted, When the read data operation is done Then the read address flag is	@(posedge clk) disable iff (~rst_n) (counter_tx == 0 && (cs == READ_DATA)) |=> ~read_ad_flag;

deasserted to make sure that the next state is read address not read data.	
At positive edge of clock if reset is deasserted, when operation read address is done then rx_data[9:8]=2'b10	@(posedge clk) disable iff (~rst_n) (counter_rx == 0 && cs == READ_ADD) |=> (rx_data[9:8] == 2'b10);
At positive edge of clock if reset is deasserted , when read data operation is done then rx_data[9:8]=2'b11	@(posedge clk) disable iff (~rst_n) (counter_rx == 0 && cs == READ_DATA) |=> (rx_data[9:8] == 2'b11);
At positive edge of clock if reset is deasserted and write operation is done then rx_data[9:8]=2'b01 Or rx_data[9:8]=2'b00.	@(posedge clk) disable iff (~rst_n) (counter_rx == 0 && cs == WRITE) |=> (rx_data[9:8] == 2'b00 || rx_data[9:8] == 2'b01);
At positive edge of clock if reset is deasserted and SSn=1 then current state returs to IDLE and the master ends communication.	@(posedge clk) disable iff (~rst_n) (SS_n |=> cs == IDLE);
When reset is asserted then reset MISO port to zero and rx_valid to zero.	always_comb begin assert final (~MISO) assert final (~rx_valid) end

Slave assertions (inside SPI sva)

Feature	Assertions
If reset is asserted, then the output on MISO port is zero	always_comb begin if (~S_if.rst_n) assert final (~S_if.MISO); end