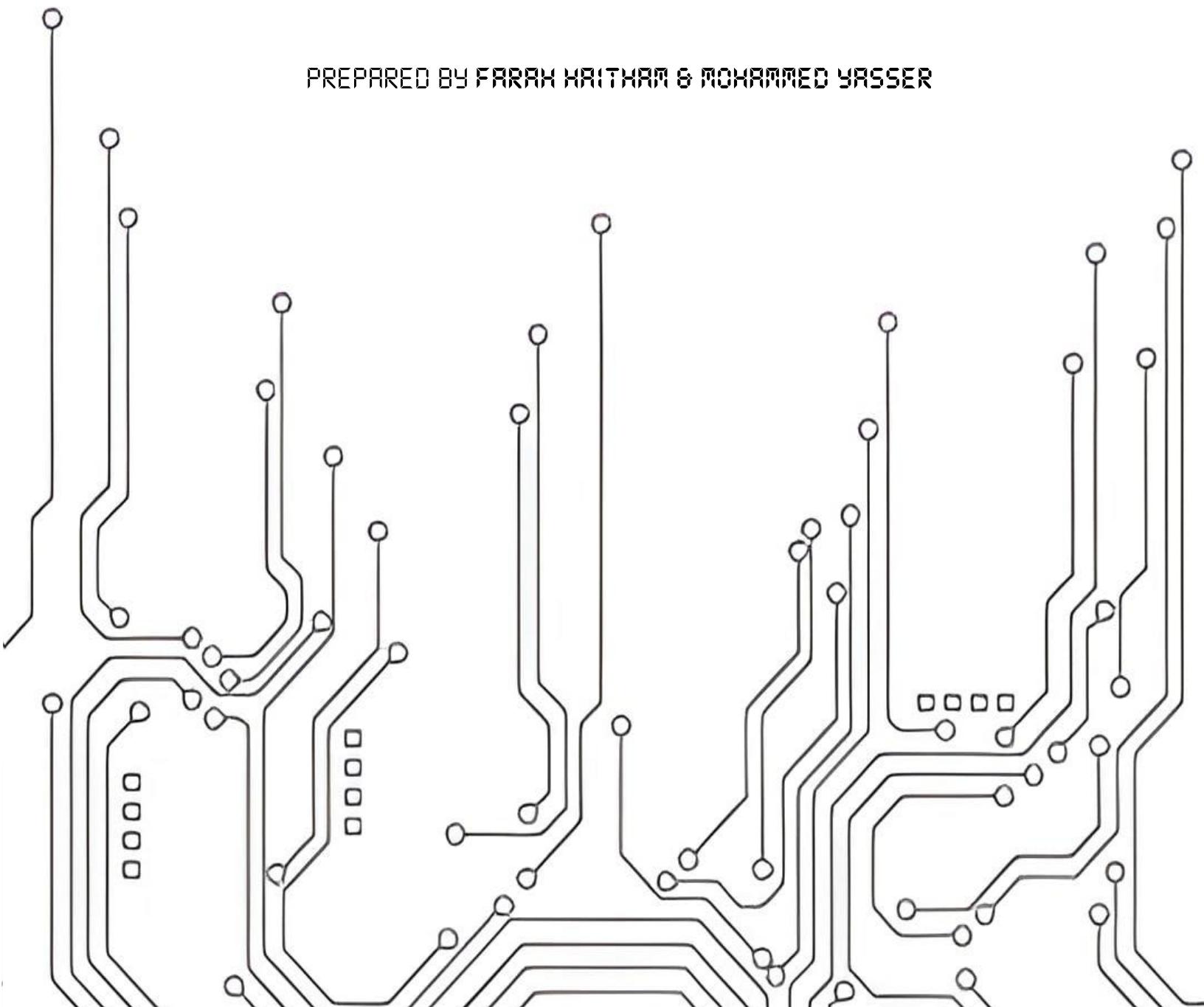


UVM-BASED VERIFICATION OF SPI SLAVE WITH SINGLE-PORT RAM

DIGITAL VERIFICATION DIPLOMA
UNDER THE SUPERVISION OF ENG. KAREEM WASSERM

PREPARED BY FARAH HAITHAM & MOHAMMED YASSER



PROJECT OVERVIEW

THIS PROJECT ENTAILS THE COMPREHENSIVE FUNCTIONAL VERIFICATION OF AN SPI SLAVE MODULE INTERFACED WITH A SINGLE PORT RAM MEMORY BLOCK USING THE UNIVERSAL VERIFICATION METHODOLOGY (UVM) FRAMEWORK. THE MAIN OBJECTIVE WAS TO ENSURE ROBUST AND RELIABLE OPERATION OF THE SPI SLAVE IN CONJUNCTION WITH MEMORY READ AND WRITE TRANSACTIONS, UNDER VARIED PROTOCOL SCENARIOS AND TIMING CONDITIONS.

THE VERIFICATION ENVIRONMENT WAS STRUCTURED MODULARLY AROUND UVM COMPONENTS INCLUDING AGENTS, DRIVERS, MONITORS, SEQUENCERS, AND SCOREBOARDS. SEPARATE AGENTS WERE DEVELOPED FOR THE SPI SLAVE AND THE RAM, FACILITATING INDEPENDENT AND COORDINATED STIMULUS GENERATION AND MONITORING. A GOLDEN REFERENCE MODEL WAS IMPLEMENTED TO VERIFY THE CORRECTNESS OF OUTPUT TRANSACTIONS BY COMPARING OUT BEHAVIOR AGAINST EXPECTED OUTPUTS.

RANDOMIZED TEST SEQUENCES WERE UTILIZED TO SIMULATE REALISTIC DATA TRANSFERS, INCLUDING WRITES AND READS VIA THE SPI INTERFACE TO RAM. FUNCTIONAL COVERAGE AND CODE COVERAGE METRICS WERE USED RIGOROUSLY TO IDENTIFY GAPS IN VERIFICATION AND ENSURE THOROUGH VALIDATION. KEY DESIGN BUGS DISCOVERED DURING SIMULATION WERE DEBUGGED AND RESOLVED, DEMONSTRATING THE EFFECTIVENESS OF THE ENVIRONMENT.

ULTIMATELY, SIMULATION RESULTS DEMONSTRATE CORRECT SPI SLAVE BEHAVIOR, ACCURATE SINGLE PORT RAM INTERFACING, AND RELIABLE DATA COMMUNICATION, CONFIRMING THE ROBUSTNESS OF THE DESIGN COMPONENTS UNDER DIVERSE OPERATIONAL CONDITIONS.

SECTION 8: VERIFICATION OF SINGLE PORT RAM

I. VERIFICATION PLAN

A	B	C	D	E
Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
1	RAM_1	When the reset is asserted, the outputs (dout & tx_valid) should be zero	Directed at the start of the simulation then randomized with constraints to be off most of the time	Coverage on the values of inputs rx_valid, din[9:8] and the value of the output tx_valid Cross coverage between din[9:8] and rx_valid. Cross coverage between din[9:8] and tx_valid.
2	RAM_2	In the write only sequence, when the rx_valid is asserted and the din[9:8] is equal to 2'b00, then write operation will happen and din[7:0] will be stored inside the wr_addr and tx_valid should be low	Randomization with constraints on rx_valid to be high most of the time and din[9:8] to always be 2'b00 or 2'b01	Coverage on the values of inputs rx_valid, din[9:8] and the value of the output tx_valid Cross coverage between din[9:8] and rx_valid. Cross coverage between din[9:8] and tx_valid.
3	RAM_3	In the write only sequence, when the rx_valid is asserted and the din[9:8] is equal to 2'b01, then write operation will happen and din[7:0] will be stored inside the memory using wr_addr as address and tx_valid should be low	Randomization with constraints on rx_valid to be high most of the time and din[9:8] to always be 2'b00 or 2'b01	Coverage on the values of inputs rx_valid, din[9:8] and the value of the output tx_valid Cross coverage between din[9:8] and rx_valid. Cross coverage between din[9:8] and tx_valid.
4	RAM_4	In the read only sequence, when the rx_valid is asserted and the din[9:8] is equal to 2'b10, then write operation will happen and din[7:0] will be stored inside rd_addr and tx_valid should be high	Randomization with constraints on rx_valid to be high most of the time and din[9:8] to always be 2'b10 or 2'b11	Coverage on the values of inputs rx_valid, din[9:8] and the value of the output tx_valid Cross coverage between din[9:8] and rx_valid. Cross coverage between din[9:8] and tx_valid.
5	RAM_5	In the read only sequence, when the rx_valid is asserted and the din[9:8] is equal to 2'b10, then read operation will happen as the dout should be the value of the memory at address equal to rd_addr and tx_valid should be low	Randomization with constraints on rx_valid to be high most of the time and din[9:8] to always be 2'b10 or 2'b12	Coverage on the values of inputs rx_valid, din[9:8] and the value of the output tx_valid Cross coverage between din[9:8] and rx_valid. Cross coverage between din[9:8] and tx_valid.
6		In the write read sequence, when the	Randomization with constraints on rx_valid to be high most of	Coverage on the values of

	A	B	C	D	E
6				din[9:8] and tx_valid.	
RAM_6	In the write read sequence, when the rx_valid is asserted and the din[9:8] is equal to 2'b00, then write operation will happen and din[7:0] will be stored inside the wr_addr and tx_valid should be low	Randomization with constraints on rx_valid to be high most of the time and din[9:8] to value based on the last operation, if the last operation was WRITE_ADDR then din[9:8] can be 2'b00 or 2'b01, if the last op was WRITE_DATA then din[9:8] can be 60% READ_ADDR or 40%WRITE_ADDR, if the last operation was READ_ADDR then din[9:8] can be 2'b00 or 2'b01, if the last op was READ_DATA then din[9:8] can be 60% WRITE_ADDR or 40%READ_ADDR	Coverage on the values of inputs rx_valid, din[9:8] and the value of the output tx_valid Cross coverage between din[9:8] and rx_valid. Cross coverage between din[9:8] and tx_valid.	The outputs are checked using both golden model and assertions	
7	RAM_7	In the write read sequence, when the rx_valid is asserted and the din[9:8] is equal to 2'b01, then write operation will happen and din[7:0] will be stored inside the memory using wr_addr as address and tx_valid should be low	Randomization with constraints on rx_valid to be high most of the time and din[9:8] to value based on the last operation, if the last operation was WRITE_ADDR then din[9:8] can be 2'b00 or 2'b01, if the last op was WRITE_DATA then din[9:8] can be 60% READ_ADDR or 40%WRITE_ADDR, if the last operation was READ_ADDR then din[9:8] can be 2'b00 or 2'b01, if the last op was READ_DATA then din[9:8] can be 60% WRITE_ADDR or 40%READ_ADDR	Coverage on the values of inputs rx_valid, din[9:8] and the value of the output tx_valid Cross coverage between din[9:8] and rx_valid. Cross coverage between din[9:8] and tx_valid.	The outputs are checked using both golden model and assertions
8	RAM_8	In the write read sequence, when the rx_valid is asserted and the din[9:8] is equal to 2'b10, then write operation will happen and din[7:0] will be stored inside rd_addr and tx_valid should be low	Randomization with constraints on rx_valid to be high most of the time and din[9:8] to value based on the last operation, if the last operation was WRITE_ADDR then din[9:8] can be 2'b00 or 2'b01, if the last op was WRITE_DATA then din[9:8] can be 60% READ_ADDR or 40%WRITE_ADDR, if the last operation was READ_ADDR then din[9:8] can be 2'b00 or 2'b01, if the last op was READ_DATA then din[9:8] can be 60% WRITE_ADDR or 40%READ_ADDR	Coverage on the values of inputs rx_valid, din[9:8] and the value of the output tx_valid Cross coverage between din[9:8] and rx_valid. Cross coverage between din[9:8] and tx_valid.	The outputs are checked using both golden model and assertions
9	RAM_9	In the write read sequence, when the rx_valid is asserted and the din[9:8] is equal to 2'b10, then read operation will happen as the dout should be the value of the memory at address equal to rd_addr and tx_valid should be high	Randomization with constraints on rx_valid to be high most of the time and din[9:8] to value based on the last operation, if the last operation was WRITE_ADDR then din[9:8] can be 2'b00 or 2'b01, if the last op was WRITE_DATA then din[9:8] can be 60% READ_ADDR or 40%WRITE_ADDR, if the last operation was READ_ADDR then din[9:8] can be 2'b00 or 2'b01, if the last op was READ_DATA then din[9:8] can be 60% WRITE_ADDR or 40%READ_ADDR	Coverage on the values of inputs rx_valid, din[9:8] and the value of the output tx_valid Cross coverage between din[9:8] and rx_valid. Cross coverage between din[9:8] and tx_valid.	The outputs are checked using both golden model and assertions
10					
11					

2. CODE SNIPPETS

INTERFACE

```
RAM_if.sv
1  interface RAM_IF(clk);
2    input      clk;
3    logic [9:0] din;
4    logic rst_n, rx_valid;
5
6    logic [7:0] dout;
7    logic tx_valid;
8
9    logic [7:0] dout_ref;
10   logic tx_valid_ref;
11 endinterface //RAM_IF
```

DESIGN

```
RAM.v
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
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[Rd_Addr]; // read addre instead of write addre //NQ
27        default : dout <= 0;
28      endcase
29      tx_valid <= (din[9] && din[8])? 1'b1 : 1'b0; //Begin + end //NQ
30    end
31  end
32 endmodule
```

GOLDEN MODEL

```
RAM.refv
1 module RAM #((
2   parameter ADDR_SIZE = 8,
3   parameter MEM_DEPTH = 256
4 )(
5   input clk, rst_n, rx_valid,
6   input[9:0] rx_data,
7   output reg tx_valid,
8   output reg[7:0] tx_data
9 );
10
11 reg[ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];
12 reg[ADDR_SIZE-1:0] wr_address, rd_address;
13
14 always @(posedge clk) begin
15   if(<rst_n>)
16     tx_valid <= 0;
17     tx_data <= 0;
18     rd_address <= 0;
19     wr_address <= 0;
20   end
21   else begin
22     if(rx_valid) begin
23       if(rx_data[9:8] == 2'b00) begin
24         wr_address <= rx_data[7:0];
25         tx_valid <= 0;
26       end
27       else if(rx_data[9:8] == 2'b01) begin
28         mem[wr_address] <= rx_data[7:0];
29         tx_valid <= 0;
30       end
31       else if(rx_data[9:8] == 2'b10) begin
32         rd_address <= rx_data[7:0];
33         tx_valid <= 0;
34       end
35       else begin
36         tx_data <= mem[rd_address];
37         tx_valid <= 1;
38       end
39     end
40   end
41 endmodule
```

SVR

```
RAM.sv.sv
1 import shared_pkg::*;
2 module RAM_sva(din,clk,rst_n,rx_valid,dout,tx_valid);
3   input [9:0] din;
4   input      clk, rst_n, rx_valid;
5
6   input [7:0] dout;
7   input      tx_valid;
8
9   op_e op;
10
11  always_comb begin
12    case (din[9:8])
13      2'b00 : op = WRITE_ADDR;
14      2'b01 : op = WRITE_DATA;
15      2'b10 : op = READ_ADDR;
16      2'b11 : op = READ_DATA;
17    endcase
18  end
19
20 //RAM_1 // RAM_2, RAM_3 //RAM_4, RAM_5 //RAM_6, RAM_7, RAM_8, RAM_9
21
22 //Assertions
23 a_reset_txValid: assert property (@(posedge clk) !rst_n |->#1 tx_valid == 1'b0);
24 a_reset_dout: assert property (@(posedge clk) !rst_n |->#1 dout == 7'b0);
25
26 a_wr_addr: assert property (@(posedge clk) disable iff (!rst_n) op == WRITE_ADDR && rx_valid |-> #1 ~tx_valid );
27 a_wr_data: assert property (@(posedge clk) disable iff (!rst_n) op == WRITE_DATA && rx_valid |-> #1 ~tx_valid );
28 a_rd_addr: assert property (@(posedge clk) disable iff (!rst_n) op == READ_ADDR && rx_valid |-> #1 ~tx_valid );
29
30 a_rd_data: assert property (@(posedge clk) disable iff (!rst_n) (op == READ_DATA && rx_valid && !is_readOnly) |->##1 $rose(tx_valid) ##[1:$] $fell(tx_valid));
31
32 a_wr_addr_to_wr_data: assert property (@(posedge clk) disable iff (!rst_n) (op == WRITE_ADDR && rx_valid) |-> (#[1:$] op == WRITE_DATA && rx_valid));
33 a_rd_addr_to_rd_data: assert property (@(posedge clk) disable iff (!rst_n) (op == READ_ADDR && rx_valid) |-> (#[1:$] op == READ_DATA && rx_valid));
34
35 //Coverage
36 a_reset_txValid_cvr: cover property (@(posedge clk) !rst_n |->#1 tx_valid == 1'b0);
37 a_reset_dout_cvr: cover property (@(posedge clk) !rst_n |->#1 dout == 7'b0);
38
39 a_wr_addr_cvr: cover property (@(posedge clk) disable iff (!rst_n) op == WRITE_ADDR |-> #1 ~tx_valid );
40 a_wr_data_cvr: cover property (@(posedge clk) disable iff (!rst_n) op == WRITE_DATA |-> #1 ~tx_valid );
41 a_rd_addr_cvr: cover property (@(posedge clk) disable iff (!rst_n) op == READ_ADDR |-> #1 ~tx_valid );
42
43 a_rd_data_cvr: cover property (@(posedge clk) disable iff (!rst_n) (op == READ_DATA && rx_valid && !is_readOnly) |->##1 $rose(tx_valid) ##[1:$] $fell(tx_valid));
44
45 a_wr_addr_to_wr_data_cvr: cover property (@(posedge clk) disable iff (!rst_n) (op == WRITE_ADDR && rx_valid) |-> (#[1:$] op == WRITE_DATA && rx_valid));
46
47
48
49
50 endmodule
```

CONFIG

```
RAM.config.sv
1 package RAM_config_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   class RAM_config extends uvm_object;
5     `uvm_object_utils(RAM_config)
6
7     virtual RAM_IF ram_if;
8     uvm_active_passive_enum is_active;
9
10    function new(string name = "RAM_config");
11      super.new(name);
12    endfunction //new()
13  endclass //RAM_config extends uvm_object
14 endpackage
```

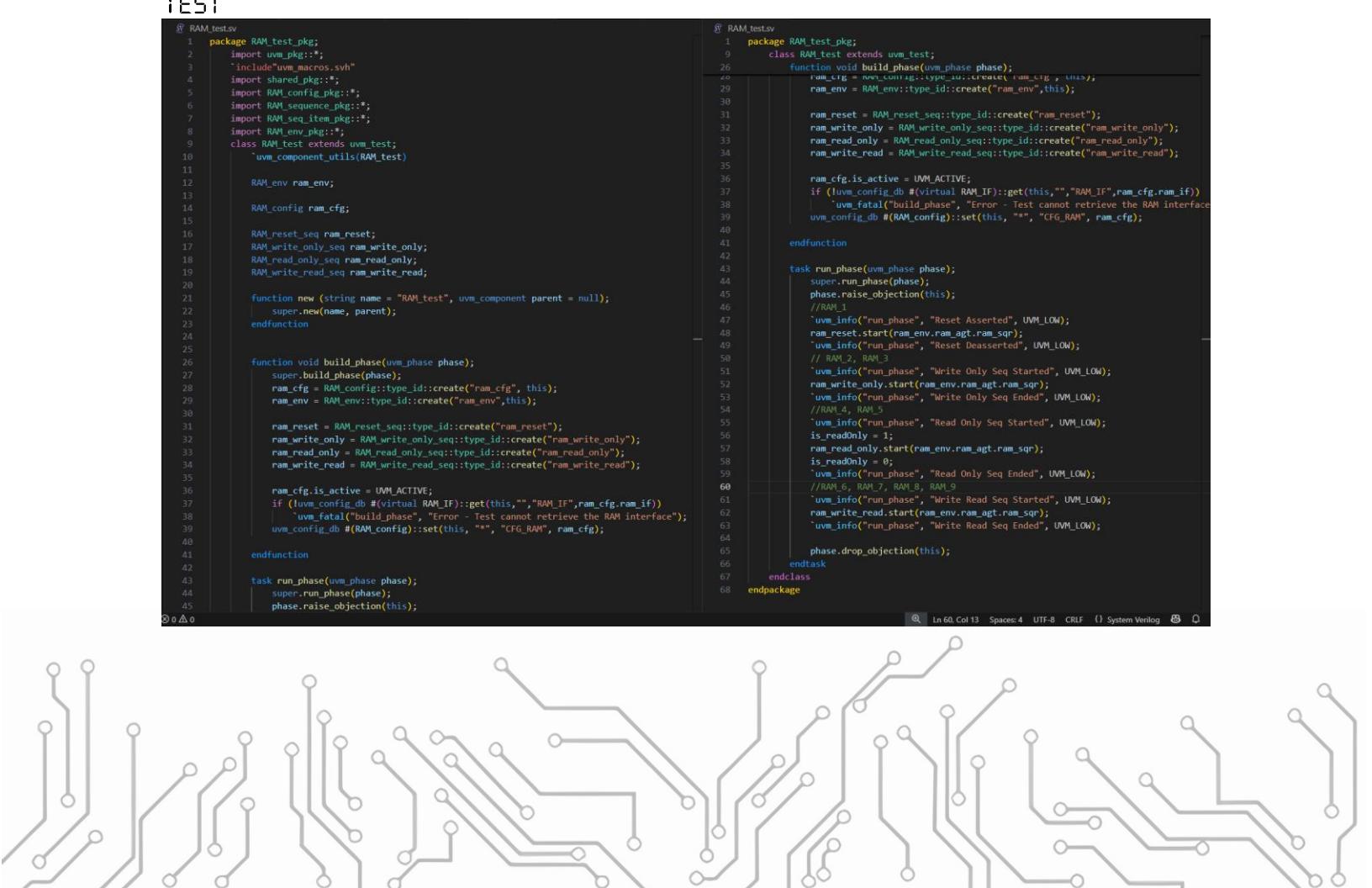
TOP



```
RAM_top.sv
1 import uvm_pkg::*;
2 `include"uvm_macros.svh"
3 import RAM_test_pkg::*;
4 module RAM_top;
5   logic clk = 0;
6   initial begin
7     $readmem("mem.dat", DUT.MEM);
8     $readmem("mem.dat", GM.mem);
9     forever #1 clk = ~clk;
10    end
11  RAM_IF ram_if (clk);
12
13  RAM_DUT (
14    .clk(ram_if.clk),
15    .rst_n(ram_if.rst_n),
16    .rx_valid(ram_if.rx_valid),
17    .din(ram_if.din),
18    .tx_valid(ram_if.tx_valid),
19    .dout(ram_if.dout)
20  );
21
22  RAM_ref GM (
23    .clk(ram_if.clk),
24    .rst_n(ram_if.rst_n),
25    .rx_valid(ram_if.rx_valid),
26    .rx_data(ram_if.din),
27    .tx_valid(ram_if.tx_valid_ref),
28    .tx_data(ram_if.dout_ref)
29  );
30
31  bind DUT RAM_SVA SVA (
32    .clk(ram_if.clk),
33    .rst_n(ram_if.rst_n),
34    .rx_valid(ram_if.rx_valid),
35    .din(ram_if.din),
36    .tx_valid(ram_if.tx_valid),
37    .dout(ram_if.dout)
38  );
39
40  initial begin
41    uvm_config_db #(virtual RAM_IF)::set(null, "uvm_test_top", "RAM_IF", ram_if);
42    run_test("RAM_test");
43  end
44
45 endmodule
```

④ 0 △ 0 In 13. Col 14 Spaces: 4 UTF-8 CRLF {} System Verilog

TEST



```
RAM_test.sv
1 package RAM_test_pkg;
2   import uvm_pkg::*;
3   `include"uvm_macros.svh"
4   import shared_pkg::*;
5   import RAM_config_pkg::*;
6   import RAM_sequence_pkg::*;
7   import RAM_seq_item_pkg::*;
8   import RAM_env_pkg::*;
9   class RAM_test extends uvm_test;
10  `uvm_component_utils(RAM_test)
11
12  RAM_env ram_env;
13
14  RAM_config ram_cfg;
15
16  RAM_reset_seq ram_reset;
17  RAM_write_only_seq ram_write_only;
18  RAM_read_only_seq ram_read_only;
19  RAM_write_read_seq ram_write_read;
20
21  function new (string name = "RAM_test", uvm_component parent = null);
22    super.new(name, parent);
23  endfunction
24
25
26  function void build_phase(uvm_phase phase);
27    super.build_phase(phase);
28    ram_cfg = RAM_config::type_id::create("ram_cfg", this);
29    ram_env = RAM_env::type_id::create("ram_env",this);
30
31    ram_reset = RAM_reset_seq::type_id::create("ram_reset");
32    ram_write_only = RAM_write_only_seq::type_id::create("ram_write_only");
33    ram_read_only = RAM_read_only_seq::type_id::create("ram_read_only");
34    ram_write_read = RAM_write_read_seq::type_id::create("ram_write_read");
35
36    ram_cfg.is_active = UVM_ACTIVE;
37    if (!uvm_config_db #(virtual RAM_IF)::get(this,"","RAM_IF",ram_cfg.ram_if))
38      `uvm_fatal("build_phase", "Error - Test cannot retrieve the RAM interface");
39    uvm_config_db #(RAM_config)::set(this, "", "CFG_RAM", ram_cfg);
40
41  endfunction
42
43  task run_phase(uvm_phase phase);
44    super.run_phase(phase);
45    phase.raise_objection(this);
46
47  endtask
48
49  endpackage
```

④ 0 △ 0 In 60. Col 13 Spaces: 4 UTF-8 CRLF {} System Verilog

ENVIRONMENT

```
RAM_env.sv
1 package RAM_env_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import RAM_agent_pkg::*;
5 import RAM_scoreboard_pkg::*;
6 import RAM_coverage_pkg::*;
7 class RAM_env extends uvm_env;
8     `uvm_component_utils(RAM_env)
9
10    RAM_agent ram_agt;
11    RAM_scoreboard ram_sb;
12    RAM_coverage ram_cvg;
13
14    function new (string name = "RAM_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        ram_agt = RAM_agent::type_id::create("ram_agt", this);
21        ram_sb = RAM_scoreboard::type_id::create("ram_sb", this);
22        ram_cvg = RAM_coverage::type_id::create("ram_cvg", this);
23    endfunction
24
25    function void connect_phase(uvm_phase phase);
26        super.connect_phase(phase);
27        ram_agt.agt_ap.connect(ram_sb.sb_exp);
28        ram_agt.agt_ap.connect(ram_cvg.cvg_exp);
29    endfunction
30
31    endclass
32
33 endpackage
```

COVERAGE

```
RAM_coverage.sv
1 package RAM_coverage_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4
5 import RAM_seq_item_pkg::*;
6 import shared_pkg::*;
7 class RAM_coverage extends uvm_component;
8     `uvm_component_utils(RAM_coverage)
9
10    uvm_analysis_export #(RAM_seq_item) cvg_exp;
11    uvm_tlm_analysis_fifo #(RAM_seq_item) cvg_fifo;
12
13    RAM_seq_item cvg_seq_item;
14 //RAM_1 // RAM_2, RAM_3 //RAM_4, RAM_5 //RAM_6, RAM_7, RAM_8, RAM_9
15    covergroup RAM_CVG;
16        rx_valid_cp: coverpoint cvg_seq_item.rx_valid{
17            bins Bin_rxValid_High = {1'b1};
18            bins Bin_rxValid_Low = {1'b0};
19        }
20        tx_valid_cp: coverpoint cvg_seq_item.tx_valid{
21            bins Bin_txValid_High = {1'b1};
22            bins Bin_txValid_Low = {1'b0};
23        }
24        operation_cp: coverpoint cvg_seq_item.din[9:8]{
25            bins Bin_Write_Addr = {WRITE_ADDR};
26            bins Bin_Write_Data = {WRITE_DATA};
27            bins Bin_Read_Addr = {READ_ADDR};
28            bins Bin_Read_Data = {READ_DATA};
29            bins Bin_Write_AddrToData = {READ_ADDR->READ_DATA};
30            bins Bin_Read_AddrToData = {READ_ADDR->READ_DATA};
31            bins Bin_Trans = {0 => 1}, {1 => 3}, {1 => 0}, {0 => 2},
32            | {2 => 3}, {2 => 0}, {3 => 1}, {3 => 2};
33        }
34
35        OP_rxValid_cross: cross operation_cp, rx_valid_cp {
36            option.cross_auto_bin_max = 0;
37            bins Bin_Write_Addr_rxValidHigh = binsof(operation_cp.Bin_Write_Addr) && binsof(rx_valid_cp.Bin_rxValid_High);
38            bins Bin_Write_Data_rxValidHigh = binsof(operation_cp.Bin_Write_Data) && binsof(rx_valid_cp.Bin_rxValid_High);
39            bins Bin_Read_Addr_rxValidHigh = binsof(operation_cp.Bin_Read_Addr) && binsof(rx_valid_cp.Bin_rxValid_High);
40            bins Bin_Read_Data_rxValidHigh = binsof(operation_cp.Bin_Read_Data) && binsof(rx_valid_cp.Bin_rxValid_High);
41        }
42        Read_OP_cross: cross operation_cp, tx_valid_cp {
43            option.cross_auto_bin_max = 0;
44            bins Bin_out = binsof(operation_cp).intersect({READ_DATA}) && binsof(tx_valid_cp).intersect({1});
45        }
    
```

In 14. Col 1 (67 selected) Spaces: 4 CRLF {} System Verilog

```


14 //RAM_1 // RAM_2, RAM_3 //RAM_4, RAM_5 //RAM_6, RAM_7, RAM_8, RAM_9
15 covergroup RAM_CVG;
16   operation_cp: coverpoint cvg_seq_item.din[9:8]{
17     bins Bin_0 = {0};
18     bins Bin_1 = {1};
19     bins Bin_2 = {2};
20     bins Bin_3 = {3};
21     bins Bin_4 = {4};
22     bins Bin_5 = {5};
23     bins Bin_6 = {6};
24     bins Bin_7 = {7};
25   }
26   OP_rxValid_cross: cross operation_cp, rx_valid_cp {
27     option.cross_auto_bin_max = 0;
28     bins Bin_Write_Addr_rxValidHigh = binsof(operation_cp.Bin_Write_Addr) && binsof(rx_valid_cp.Bin_rxValid.High);
29     bins Bin_Write_Data_rxValidHigh = binsof(operation_cp.Bin_Write_Data) && binsof(rx_valid_cp.Bin_rxValid.High);
30     bins Bin_Read_Addr_rxValidHigh = binsof(operation_cp.Bin_Read_Addr) && binsof(rx_valid_cp.Bin_rxValid.High);
31     bins Bin_Read_Data_rxValidHigh = binsof(operation_cp.Bin_Read_Data) && binsof(rx_valid_cp.Bin_rxValid.High);
32   }
33   Read_OP_cross: cross operation_cp, tx_valid_cp {
34     option.cross_auto_bin_max = 0;
35     bins Bin_out = binsof(operation_cp) intersect {READ_DATA} && binsof(tx_valid_cp) intersect {1};
36   }
37 endgroup
38 function new (string name = "RAM_coverage", uvm_component parent = null);
39   super.new(name, parent);
40   RAM_CVG = new();
41 endfunction
42
43 function void build_phase (uvm_phase phase);
44   super.build_phase(phase);
45
46   cvg_axp = new ("cvg_axp", this);
47   cvg_fifo = new ("cvg_fifo", this);
48 endfunction
49
50 function void connect_phase(uvm_phase phase);
51   super.connect_phase(phase);
52   cvg_axp.connect(cvg_fifo.analysis_export);
53 endfunction
54
55 task run_phase(uvm_phase phase);
56   super.run_phase(phase);
57   forever begin
58     cvg_fifo.get(cvg_seq_item);
59     RAM_CVG.sample();
60   end
61 endtask
62
63 endclass
64
65 endpackage


```

Ln 70, Col 16 Spaces: 4 UTF-8 CRLF ⚙ System Verilog 📁 4:36 AM 10/13/2025

SCOREBOARD

```


1 package RAM_scoreboard_pkg;
2   import uvm_pkg::*;
3   `include "vmm_macros.svh"
4   import RAM_seq_item_pkg::*;
5   class RAM_scoreboard extends uvm_scoreboard;
6     `uvm_component_utils(RAM_scoreboard)
7
8     logic [7:0] dout_ref;
9     logic       tx_valid_ref;
10
11    logic [7:0] Rd_Addr, Wr_Addr;
12
13    logic [7:0] MEM_ref [logic[7:0]];
14
15    uvm_analysis_export #(RAM_seq_item) sb_axp;
16    uvm_tlm_analysis_fifo #(RAM_seq_item) sb_fifo;
17
18    RAM_seq_item sb_seq_item;
19
20    int error_cnt = 0, correct_cnt = 0;
21
22    function new (string name = "RAM_scoreboard", 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      sb_axp = new("sb_axp", this);
29      sb_fifo = new("sb_fifo", this);
30    endfunction
31
32    function void connect_phase(uvm_phase phase);
33      super.connect_phase(phase);
34      sb_axp.connect(sb_fifo.analysis_export);
35    endfunction
36
37    task run_phase(uvm_phase phase);
38      super.run_phase(phase);
39      forever begin
40        sb_fifo.get(sb_seq_item);
41        // ref_model(sb_seq_item);
42        if (
43          sb_seq_item.tx_valid_ref != sb_seq_item.tx_valid ||
44          sb_seq_item.dout_ref != sb_seq_item.dout
45        )
46          error_cnt++;
47        else
48          correct_cnt++;
49      end
50    endtask
51
52    function void report_phase(uvm_phase phase);
53      super.report_phase(phase);
54      $display("Scoreboard: Error Count = %d, Correct Count = %d", error_cnt, correct_cnt);
55    endfunction
56
57 endclass
58
59 endpackage


```

Ln 847, Col 16 Spaces: 4 UTF-8 CRLF ⚙ System Verilog 📁 4:36 AM 10/13/2025



```


`ifndef RAM_scoreboard_sv
`define RAM_scoreboard_sv

package RAM_scoreboard_pkg;
  class RAM_scoreboard extends uvm_scoreboard;
    function new (string name = "RAM_scoreboard", uvm_component parent = null);
      super.new(name, parent);
    endfunction

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

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

    task run_phase(uvm_phase phase);
      super.run_phase(phase);
      forever begin
        sb_fifo.get(sb_seq_item);
        // ref_model(sb_seq_item);
        if (
          sb_seq_item.tx_valid_ref != sb_seq_item.tx_valid ||
          sb_seq_item.dout_ref != sb_seq_item.dout
        ) begin
          error_cnt++;
          `uvm_error("run_phase", $sformatf("Comparison Failed - Transaction Received DUT : %s, Ref Model : tx_valid = %b, dout = %h", sb_seq_item.convert2string,
                                              sb_seq_item.tx_valid_ref, sb_seq_item.dout_ref));
        end
        else begin
          correct_cnt++;
          `uvm_info("run_phase", $sformatf("Correct RAM Comparison: %s", sb_seq_item.convert2string),UVM_HIGH);
        end
      end
    endtask

    function void report_phase (uvm_phase phase);
      super.report_phase(phase);
      `uvm_info("report_phase", $sformatf("Correct Tests = %d, Error Tests = %d", correct_cnt, error_cnt),UVM_MEDIUM)
    endfunction
  endclass
endpackage


```

Ln 49, Col 73 Spaces: 4 UTF-8 CRLF ⓘ System Verilog

AGENT



```


`ifndef RAM_agent_sv
`define RAM_agent_sv

package RAM_agent_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import RAM_sequencer_pkg::*;
  import RAM_monitor_pkg::*;
  import RAM_driver_pkg::*;
  import RAM_seq_item_pkg::*;
  import RAM_config_pkg::*;
  class RAM_agent extends uvm_agent;
    `uvm_component_utils(RAM_agent)
    RAM_config ram_cfg;
    RAM_sequencer ram_sqr;
    RAM_monitor ram_mon;
    RAM_driver ram_dvr;

    uvm_analysis_port #(RAM_seq_item) agt_ap;

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

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if (!uvm_config_db #(RAM_config)::get(this, "", CFG_RAM",ram_cfg))
        `uvm_fatal("build_phase", "Error - Agent cannot retrieve the config object");
      ram_mon = RAM_monitor::type_id::create("ram_mon", this);
      if (ram_cfg.is_active == UVM_ACTIVE) begin
        ram_dvr = RAM_driver::type_id::create("ram_dvr", this);
        ram_sqr = RAM_sequencer::type_id::create("ram_sqr", this);
      end
      agt_ap = new ("agt_ap", this);
    endfunction

    function void connect_phase (uvm_phase phase);
      super.connect_phase(phase);
      if(ram_cfg.is_active == UVM_ACTIVE) begin
        ram_dvr.ram_dvr_if = ram_cfg.ram_if;
        ram_dvr.seq_item_port.connect(ram_sqr.seq_item_export);
      end
    endfunction
  endclass
endpackage


```

Ln 16, Col 15 Spaces: 4 UTF-8 CRLF ⓘ System Verilog

DRIVER

```
RAM_driver.sv
1 package RAM_driver_pkg;
2 import uvm_pkg::*;
3 `include"uvm_macros.svh"
4 import RAM_seq_item_pkg::*;
5 class RAM_driver extends uvm_driver #(RAM_seq_item);
6   `uvm_component_utils(RAM_driver)
7
8   RAM_seq_item dvr_seq_item;
9   virtual RAM_IF ram_dvr_if;
10
11   function new (string name = "RAM_driver", 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       dvr_seq_item = RAM_seq_item::type_id::create("dvr_seq_item");
19       seq_item_port.get_next_item(dvr_seq_item);
20       ram_dvr_if.rst_n = dvr_seq_item.rst_n;
21       ram_dvr_if.rx_valid = dvr_seq_item.rx_valid;
22       ram_dvr_if.din = dvr_seq_item.din;
23       repeat(1)@{negedge ram_dvr_if.clk};
24       seq_item_port.item_done();
25     `uvm_info("run_phase", dvr_seq_item.convert2string_stimulus, UVM_HIGH);
26   end
27 endtask
28 endclass
29 endpackage
```

MONITOR

```
RAM_monitor.sv
1 package RAM_monitor_pkg;
2 import uvm_pkg::*;
3 `include"uvm_macros.svh"
4 import RAM_seq_item_pkg::*;
5 class RAM_monitor extends uvm_monitor;
6   `uvm_component_utils(RAM_monitor)
7
8   virtual RAM_IF ram_mon_if;
9   RAM_seq_item mon_seq_item;
10
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       mon_seq_item = RAM_seq_item::type_id::create("mon_seq_item");
26       repeat(1)@{negedge ram_mon_if.clk};
27       mon_seq_item.rst_n = ram_mon_if.rst_n;
28       mon_seq_item.rx_valid = ram_mon_if.rx_valid;
29       mon_seq_item.din = ram_mon_if.din;
30       mon_seq_item.dout = ram_mon_if.dout;
31       mon_seq_item.tx_valid = ram_mon_if.tx_valid;
32       mon_seq_item.dout_ref = ram_mon_if.dout_ref;
33       mon_seq_item.tx_valid_ref = ram_mon_if.tx_valid_ref;
34       mon_ap.write(mon_seq_item);
35       `uvm_info("run_phase", mon_seq_item.convert2string, UVM_HIGH);
36     end
37   endtask
38
39
40 endclass
41 endpackage
```

SEQUENCER

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

SEQUENCE

```
RAM_sequence.sv
1 package RAM_sequence_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import shared_pkg::*;
5 import RAM_seq_item_pkg::*;
6 class RAM_reset_seq extends uvm_sequence #(RAM_seq_item);
7   `uvm_object_utils(RAM_reset_seq)
8
9   RAM_seq_item ram_seq;
10
11   function new(string name = "RAM_reset_seq");
12     super.new(name);
13   endfunction
14
15 // RAM_1
16   virtual task body();
17     ram_seq = RAM_seq_item::type_id::create("ram_seq");
18     start_item(ram_seq);
19     ram_seq.rst_n = 0;
20     ram_seq.rx_valid = 0;
21     ram_seq.din = 0;
22     finish_item(ram_seq);
23   endtask
24 endclass
25 class RAM_write_only_seq extends uvm_sequence #(RAM_seq_item);
26   `uvm_object_utils(RAM_write_only_seq)
27
28   RAM_seq_item ram_seq;
29
30   function new(string name = "RAM_write_only_seq");
31     super.new(name);
32   endfunction
33
34 // RAM_2, RAM_3
35   virtual task body();
36     repeat(10000)begin
37       ram_seq = RAM_seq_item::type_id::create("ram_seq");
38       start_item(ram_seq);
39       ram_seq.read_only_const.constraint_mode(0);
40       ram_seq.write_read_const.constraint_mode(0);
41       assert(ram_seq.randomize);
42       finish_item(ram_seq);
43     end
44   endtask
45 endclass
46 class RAM_read_only_seq extends uvm_sequence #(RAM_seq_item);
47   `uvm_object_utils(RAM_read_only_seq)
```

```
RAM_sequence.sv
32 // RAM_2, RAM_3
33   ...
34   endtask
35 endclass
36 class RAM_read_only_seq extends uvm_sequence #(RAM_seq_item);
37   `uvm_object_utils(RAM_read_only_seq)
38
39   RAM_seq_item ram_seq;
40
41   function new(string name = "RAM_read_only_seq");
42     super.new(name);
43   endfunction
44
45 // RAM_4, RAM_5
46   virtual task body();
47     repeat(10000)begin
48       ram_seq = RAM_seq_item::type_id::create("ram_seq");
49       start_item(ram_seq);
50       ram_seq.write_only_const.constraint_mode(0);
51       ram_seq.write_read_const.constraint_mode(0);
52       assert(ram_seq.randomize);
53       finish_item(ram_seq);
54     end
55   endtask
56 endclass
57 class RAM_write_read_seq extends uvm_sequence #(RAM_seq_item);
58   `uvm_object_utils(RAM_write_read_seq)
59
60   RAM_seq_item ram_seq;
61
62   function new(string name = "RAM_write_read_seq");
63     super.new(name);
64   endfunction
65
66 // RAM_6, RAM_7, RAM_8, RAM_9
67   virtual task body();
68     repeat(10000)begin
69       ram_seq = RAM_seq_item::type_id::create("ram_seq");
70       start_item(ram_seq);
71       ram_seq.read_only_const.constraint_mode(0);
72       ram_seq.write_only_const.constraint_mode(0);
73       assert(ram_seq.randomize);
74       finish_item(ram_seq);
75     end
76   endtask
77 endclass
78 endpackage
```

0 0 △ 0

Ln 72 Col 29 Spaces:4 UTF-8 CRLF () System Verilog

SEQUENCE ITEM

```
RAM_seq_item.sv
1 package RAM_seq_item_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import shared_pkg::*;
5 class RAM_seq_item extends uvm_sequence_item;
6   `uvm_object_utils(RAM_seq_item)
7
8   // Signals Declarations
9   rand logic [9:0] din;
10  rand logic rst_n, rx_valid;
11
12  logic [7:0] dout;
13  logic tx_valid;
14  logic [7:0] dout_ref;
15  logic tx_valid_ref;
16
17  op_e wr_op[] = '{WRITE_ADDR, WRITE_DATA};
18  op_e rd_op[] = '{READ_ADDR, READ_DATA};
19
20  bit wr_addr, wr_data, rd_addr, rd_data;
21
22  rand op_e op_rd_data, op_wr_data;
23
24  static op_e last_op;
25
26  // Functions
27  function new(string name = "RAM_seq_item");
28    super.new(name);
29  endfunction
30
31  function string convert2string();
32    return $sformatf("%s - reset = %b, rx_valid = %b, din = %h ---- tx_valid = %b, dout = %h", super.convert2string, rst_n, rx_valid, din, tx_valid, dout);
33  endfunction
34
35  function string convert2string_stimulus();
36    return $sformatf("%s - reset = %b, rx_valid = %b, din = %h", super.convert2string, rst_n, rx_valid, din);
37  endfunction
38
39  function void post_randomize();
40    last_op = op_e'(din[9:8]);
41  endfunction
42
43 // Constraints
44 // RAM_1
45 constraint reset_const{
46   rst_n dist {1:/90 , 0:/10};
47 }
48 // RAM_2, RAM_3
49 constraint rx_valid_const{
50   rx_valid dist {1:/90 , 0:/10};
51 }
```

0 0 △ 0

Ln 55 Col 33 Spaces:4 UTF-8 CRLF () System Verilog



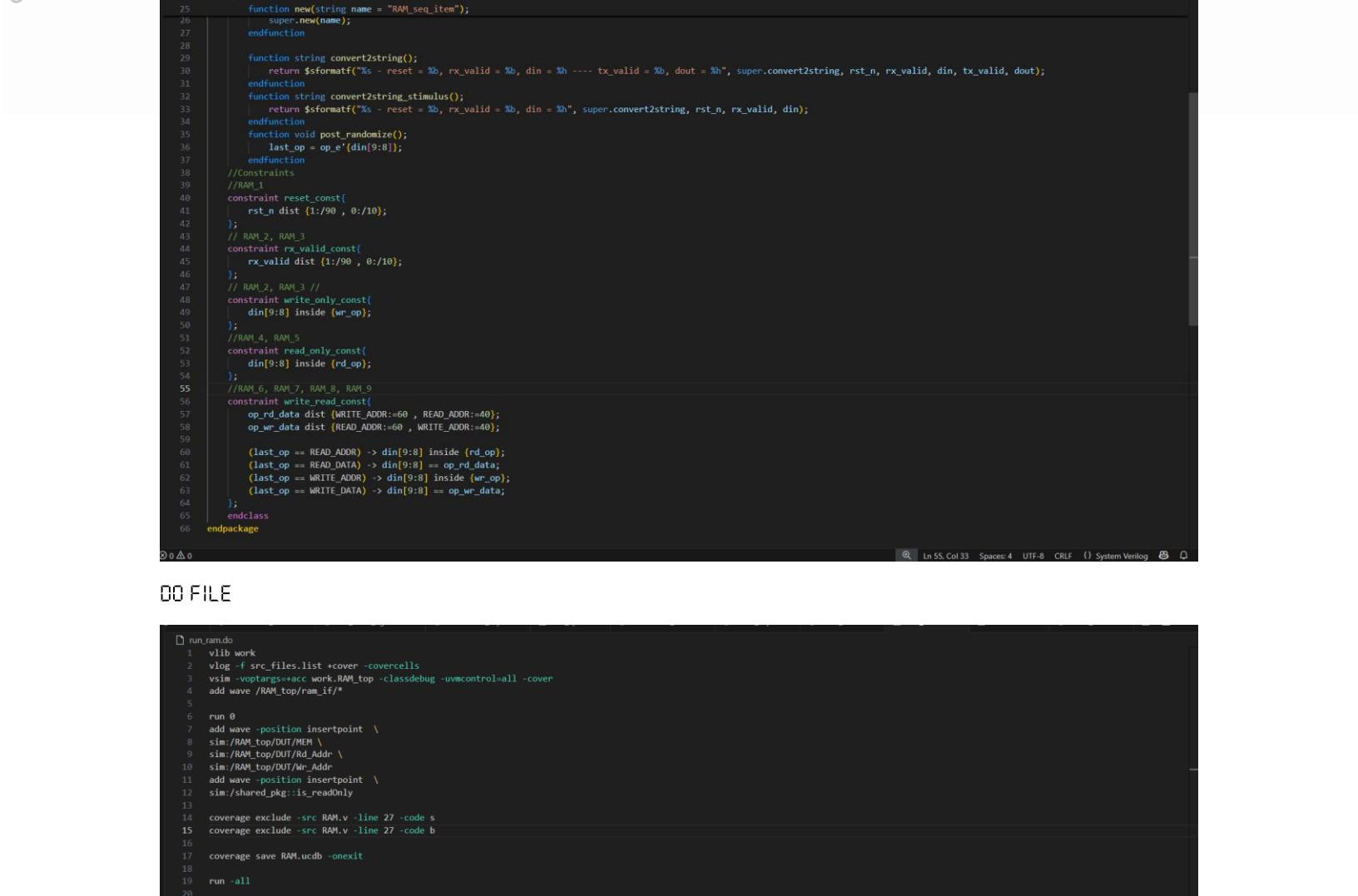
```

1 package RAM_seq_item_pkg;
2   // Functions
3   function new(string name = "RAM_seq_item");
4     super.new(name);
5   endfunction
6
7   function string convert2string();
8     return $sformatf("%s - reset = %b, rx_valid = %b, din = %h ---- tx_valid = %b, dout = %h", super.convert2string, rst_n, rx_valid, din, tx_valid, dout);
9   endfunction
10  function string convert2string_stimulus();
11    return $sformatf("%s - reset = %b, rx_valid = %b, din = %h", super.convert2string, rst_n, rx_valid, din);
12  endfunction
13
14  function void post_randomize();
15    last_op = op_e'(din[9:8]);
16  endfunction
17
18  //Constraints
19  //RAM_1
20  constraint reset_const{
21    rst_n dist {1:/90 , 0:/10};
22  };
23  // RAM_2, RAM_3
24  constraint rx_valid_const{
25    rx_valid dist {1:/98 , 0:/10};
26  };
27  // RAM_2, RAM_3 //
28  constraint write_only_const{
29    din[9:8] inside {wr_op};
30  };
31  //RAM_4, RAM_5
32  constraint read_only_const{
33    din[9:8] inside {rd_op};
34  };
35  //RAM_6, RAM_7, RAM_8, RAM_9
36  constraint write_read_const{
37    op_rd_data dist {WRITE_ADDR:>60 , READ_ADDR:<=40};
38    op_wr_data dist {READ_ADDR:>60 , WRITE_ADDR:<=40};
39
40    (last_op == READ_ADDR) -> din[9:8] inside {rd_op};
41    (last_op == READ_DATA) -> din[9:8] == op_rd_data;
42    (last_op == WRITE_ADDR) -> din[9:8] inside {wr_op};
43    (last_op == WRITE_DATA) -> din[9:8] == op_wr_data;
44  };
45  endclass
46 endpackage

```

Ln 55, Col 33 Spaces: 4 UTF-8 CRLF {} System Verilog

DO FILE



```

run_ram.do
1 vlib work
2 vlog -f src_files.list +cover -covercells
3 vsim -voptargs+=acc work.RAM_top -classdebug -uvmcontrol=all -cover
4 add wave /RAM_top/ram_if/*
5
6 run 0
7 add wave -position insertpoint \
8 sim:/RAM_top/DUT/MEM \
9 sim:/RAM_top/DUT/Rd_Addr \
10 sim:/RAM_top/DUT/Wr_Addr \
11 add wave -position insertpoint \
12 sim:/shared_pkg::is_READONLY
13
14 coverage exclude -src RAM.v -line 27 -code s
15 coverage exclude -src RAM.v -line 27 -code b
16
17 coverage save RAM.ucdb -onexit
18
19 run -all
20
21

```

3. BUG REPORT

Bug #1: AT THE CASE DIN[9:8] = 2'b1, USING WR_ADDR instead of RD_ADDR BEFORE

```
2'b10 : Rd_Addr <= din[7:0];
2'b11 : dout <= MEM[Wr_Addr];
default : dout <= 0;
```

AFTER

```
2'b10 : Rd_Addr <= din[7:0];
2'b11 : dout <= MEM[Rd_Addr]; // read addr instead of write addr //M2
default : dout <= 0;
```

Bug #2: TX_VALID SHOULD MAINTAIN ITS VALUE WHEN THE RX_VALID IS NOT ACTIVE BEFORE

```
endcase
end
tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 : 1'b0;
```

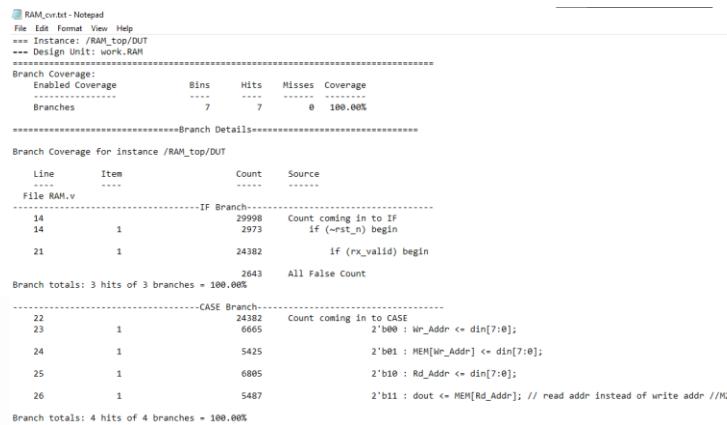
AFTER

```
else
  if (rx_valid) begin
    case (din[9:8])
      2'b00 : Wr_Addr <= din[7:0];
      2'b01 : MEM[Wr_Addr] <= din[7:0];
      2'b10 : Rd_Addr <= din[7:0];
      2'b11 : dout <= MEM[Rd_Addr]; // read addr instead of write addr //M2
      default : dout <= 0;
    endcase
    tx_valid <= (din[9] && din[8])? 1'b1 : 1'b0; //Begin - end //M1
  end
```

4. COVERAGE

CODE COVERAGE

BRANCH



RAM_coverage.txt - Notepad
File Edit Format View Help
--- Instances /RAM_top/DUT
--- Design Unit: work.RAM

Branch Coverage:
Enabled Coverage Bins Hits Misses Coverage

Branches 7 7 0 100.00%

=====Branch Details=====
Branch Coverage for instance /RAM_top/DUT
File: RAM.v
Line Item Count Source
---- ---- ----
14 IF Branch 29998 Count coming in to IF
14 1 2973 if (~rst_n) begin
21 1 24382 if (rx_valid) begin
2643 All False Count
Branch totals: 3 hits of 3 branches = 100.00%
-----CASE Branch-----
22 24382 Count coming in to CASE
23 1 6665 2'b00 : Wr_Addr <= din[7:0];
24 1 5425 2'b01 : MEM[Wr_Addr] <= din[7:0];
25 1 6805 2'b10 : Rd_Addr <= din[7:0];
26 1 5487 2'b11 : dout <= MEM[Rd_Addr]; // read addr instead of write addr //M2
Branch totals: 4 hits of 4 branches = 100.00%

STATEMENT



RAM_cv.tst - Notepad
File Edit Format View Help
*****Statement Details*****
Statement Coverage for instance /RAM_top/DUT --

Line	Item	Count	Source
1	File RAM.v	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	1 29998 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		
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[Rd_Addr]; // read addr instead of write addr //M2		
27	default : dout <= 0;		
28	endcase		
29	1 24382 tx_valid <= (din[9] && din[8])? 1'b1 : 1'b0; //Begin - end //M1		

Ln 1, Col 1 90% Windows (CRLF) UTF-8

TOGGLE



Toggle Coverage:
Enabled Coverage

Toggles Bins Hits Misses Coverage

Toggles 76 76 0 100.00%

*****Toggle Details*****
Toggle Coverage for instance /RAM_top/DUT --

Node	IH->OL	OL->IH	"Coverage"
Rd_Addr[0:7]	1	1	100.00
Wr_Addr[0:7]	1	1	100.00
clk	1	1	100.00
din[0:9]	1	1	100.00
dout[0:7]	1	1	100.00
rst_n	1	1	100.00
rx_valid	1	1	100.00
tx_valid	1	1	100.00

Total Node Count = 38
Toggled Node Count = 38
Untoggled Node Count = 0
Toggle Coverage = 100.00% (76 of 76 bins)

FUNCTIONAL COVERAGE

File	Edit	Format	View	Help
COVERGROUP COVERAGE:				
Covergroup	Metric	Goal	Bins	Status
TYPE /RAM_coverage_pkg::RAM_coverage::RAM_CVG	100.00%	100	-	Covered
covered/total bins:	16	16	-	Covered
missing/total bins:	0	16	-	
% Hit:	100.00%	100	-	
Coverpoint rx_valid_cp	100.00%	100	-	Covered
covered/total bins:	2	2	-	Covered
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
Coverpoint rx_val_low	100.00%	100	-	Covered
covered/total bins:	0	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
Coverpoint operation_cp	100.00%	100	-	Covered
covered/total bins:	7	7	-	Covered
missing/total bins:	0	7	-	
% Hit:	100.00%	100	-	
Cross OP_rxValid_cross	100.00%	100	-	Covered
covered/total bins:	4	4	-	Covered
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Covergroup instance \RAM_coverage_pkg::RAM_coverage	100.00%	100	-	Covered
covered/total bins:	16	16	-	Covered
missing/total bins:	0	16	-	
% Hit:	100.00%	100	-	
Coverpoint rx_val_low_cp	100.00%	100	-	Covered
covered/total bins:	2	2	-	Covered
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin Bin_txValid_High	100.00%	1	-	Covered
bin Bin_txValid_Low	100.00%	1	-	Covered
Coverpoint tx_val_low_cp	100.00%	100	-	Covered
covered/total bins:	2	2	-	Covered
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin Bin_txValid_High	100.00%	5979	1	Covered
bin Bin_txValid_Low	100.00%	1	-	Covered
Coverpoint operation_cp_1	100.00%	100	-	Covered
covered/total bins:	7	7	-	Covered
missing/total bins:	0	7	-	
% Hit:	100.00%	100	-	
bin Bin_Write_Addr	100.00%	1	-	Covered
bin Bin_Write_Data	100.00%	1	-	Covered
bin Bin_Read_Addr	100.00%	1	-	Covered
bin Bin_Read_Data	100.00%	1	-	Covered
bin Bin_Write_AddrToData	100.00%	1	-	Covered
bin Bin_Read_AddrToData	100.00%	1	-	Covered
bin Bin_out	100.00%	1	-	Covered
Cross OP_rxValid_cross	100.00%	100	-	Covered
covered/total bins:	4	4	-	Covered
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:	100.00%	100	-	
Auto, Bin_Write_Addr rxvalHigh	7379	1	-	Covered
bin Bin_Write_Data_rxvalHigh	6043	1	-	Covered
bin Bin_Read_Data_rxvalHigh	623	1	-	Covered
bin Bin_Read_Data_rxvalHigh	6085	1	-	Covered
Cross Read_OP_cross	100.00%	100	-	Covered
covered/total bins:	1	1	-	Covered
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:	100.00%	100	-	
bin Bin_out	5894	1	-	Covered

TOTAL COVERGROUP COVERAGE: 100.00% COVERGROUP TYPES: 1

ASSERTION COVERAGE

DIRECTIVE COVERAGE:				
Name	Design	Design	Lang	File(Line)
	Unit	Unit	File	Line
/RAM_top/DUT/SVA/a_reset_rxValid_cv	RAM_sva Verilog	SVA	RAM_sva.sv(36)	2973 Covered
/RAM_top/DUT/SVA/a_rd_addr_cv	RAM_sva Verilog	SVA	RAM_sva.sv(37)	1099 Covered
/RAM_top/DUT/SVA/a_rd_data_cv	RAM_sva Verilog	SVA	RAM_sva.sv(39)	6628 Covered
/RAM_top/DUT/SVA/a_rd_data_cv	RAM_sva Verilog	SVA	RAM_sva.sv(40)	5418 Covered
/RAM_top/DUT/SVA/a_rd_data_cv	RAM_sva Verilog	SVA	RAM_sva.sv(41)	5418 Covered
/RAM_top/DUT/SVA/a_rd_data_cv	RAM_sva Verilog	SVA	RAM_sva.sv(43)	1089 Covered
/RAM_top/DUT/SVA/a_rd_addr_cv	RAM_sva Verilog	SVA	RAM_sva.sv(45)	5267 Covered
/RAM_top/DUT/SVA/a_rd_addr_cv	RAM_sva Verilog	SVA	RAM_sva.sv(46)	5375 Covered

TOTAL DIRECTIVE COVERAGE: 100.00% COVERS: 8

ASSERTION RESULTS:				
Name	File(Line)	Failure Count	Pass Count	
/RAM_top/DUT/SVA/a_reset_rxValid	RAM_sva.sv(23)	0	1	
/RAM_top/DUT/SVA/a_rd_addr	RAM_sva.sv(24)	0	1	
/RAM_top/DUT/SVA/a_rd_data	RAM_sva.sv(26)	0	1	
/RAM_top/DUT/SVA/a_rd_data	RAM_sva.sv(27)	0	1	
/RAM_top/DUT/SVA/a_rd_data	RAM_sva.sv(28)	0	1	
/RAM_top/DUT/SVA/a_rd_data	RAM_sva.sv(30)	26	1	
/RAM_top/DUT/SVA/a_rd_addr_to_wt_data	RAM_sva.sv(32)	0	1	
/RAM_top/DUT/SVA/a_rd_addr_to_wt_data	RAM_sva.sv(33)	0	1	
/RAM_sequence_pkg@/Ram_write_only_seq/seq/loop#(ublk#(33852551#m34/imm34))	RAM_sequence.sv(39)	0	1	
/RAM_sequence_pkg@/Ram_read_only_seq/seq/loop#(ublk#(33852551#m54/imm54))	RAM_sequence.sv(59)	0	1	
/RAM_sequence_pkg@/Ram_write_only_seq/seq/loop#(ublk#(33852551#m74/imm74))	RAM_sequence.sv(74)	0	1	
Total Coverage By Instance (Filtered view): 77.20%				

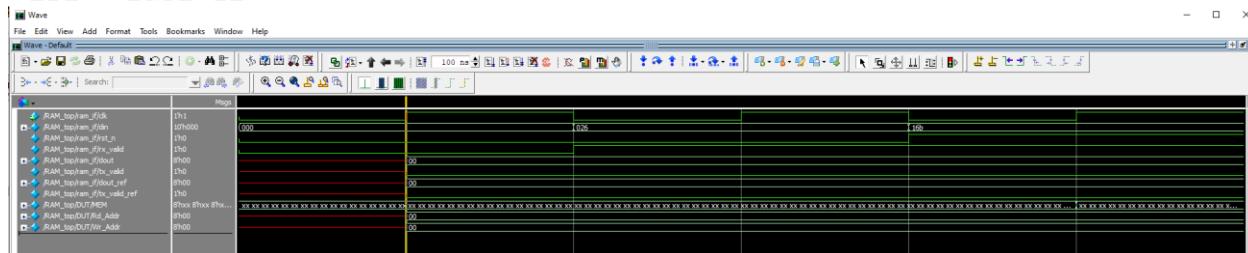
NOTE: A_RD_DATA ASSERTION WILL BE FIXED WHEN USING THE RAM INSIDE THE WRAPPER

5. ASSERTIONS TABLE

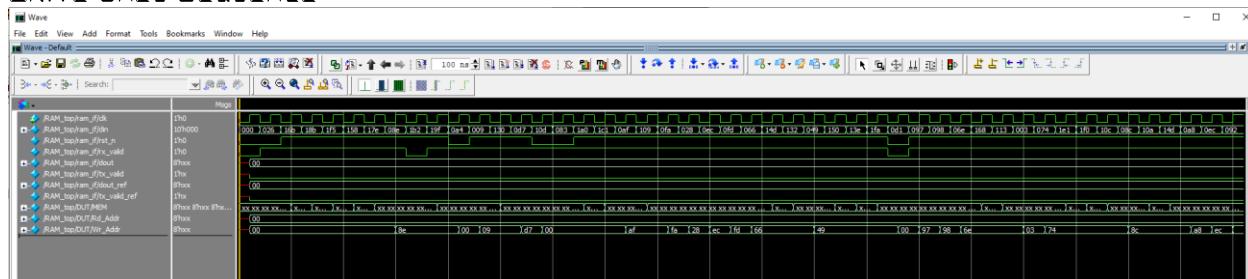
A	B	C	D	E	F	G	H	I
Feature	Assertion							
When the reset is asserted, after 1 clk cycle tx_valid is low	(@posedge clk) \$rose(tx_valid) #1 \$fell(tx_valid)							
When the reset is asserted, after 1 clk cycle dout is low	(@posedge clk) \$rose(dout) #1 \$fell(dout)							
when the reset is inactive and the din[9:8] = 2'b00 and the rx_valid is high, after 1 clk cycle tx_valid is low	(@posedge clk) disable iff (rst_n) op == WRITE_ADDR && rx_valid > #1 ~tx_valid							
when the reset is inactive and the din[9:8] = 2'b01 and the rx_valid is high, after 1 clk cycle tx_valid is low	(@posedge clk) disable iff (rst_n) op == WRITE_DATA && rx_valid > #1 ~tx_valid							
when the reset is inactive and the din[9:8] = 2'b10 and the rx_valid is high, after 1 clk cycle tx_valid is low	(@posedge clk) disable iff (rst_n) op == READ_ADDR && rx_valid > #1 ~tx_valid							
when the reset is inactive and the din[9:8] = 2'b11 and the rx_valid is high, after 1 or more clk cycles tx_valid should fall	(@posedge clk) disable iff (rst_n) (op == READ_DATA && rx_valid && isReadOnly) > #1 \$rose(tx_valid) #1 \$fell(tx_valid)							
when the reset is inactive and the din[9:8] = 2'b00 and the rx_valid is high, after 1 or more clk cycles din[9:8] should equal 2'b01 and rx_valid should be high	(@posedge clk) disable iff (rst_n) (op == WRITE_ADDR && rx_valid) > (#1\$ op == WRITE_DATA && rx_valid)							
when the reset is inactive and the din[9:8] = 2'b10 and the rx_valid is high, after 1 or more clk cycles din[9:8] should equal 2'b11 and rx_valid should be high	(@posedge clk) disable iff (rst_n) (op == READ_ADDR && rx_valid) > (#1\$ op == READ_DATA && rx_valid)							

6. SIMULATION SNIPPETS

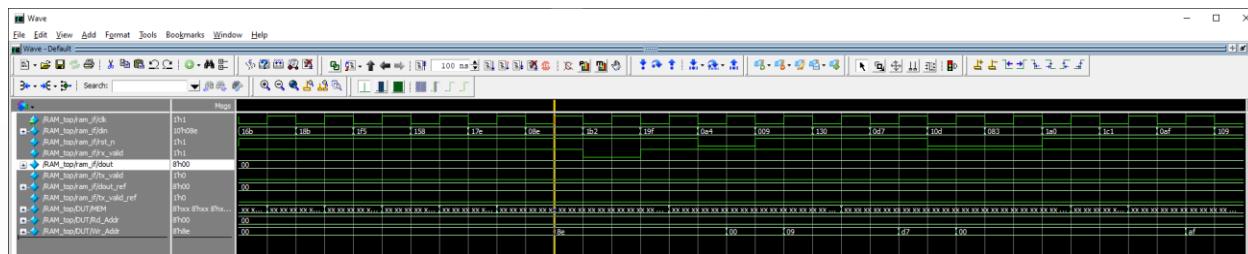
RESET SEQUENCE



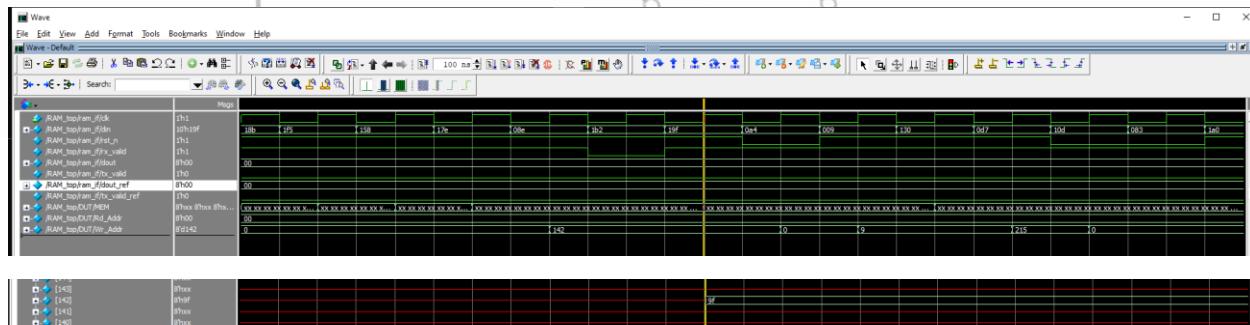
WRITE ONLY SEQUENCE



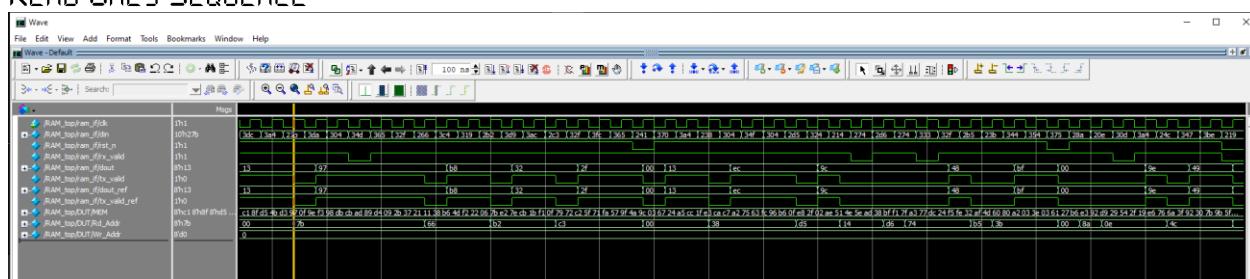
WRITE ADDRESS



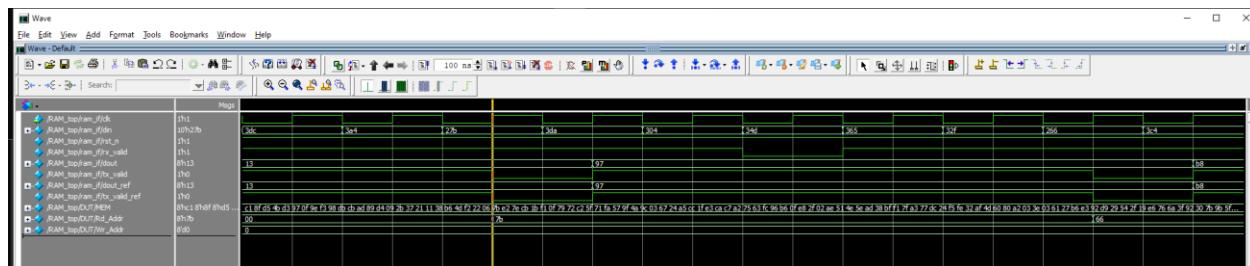
WRITE DATA



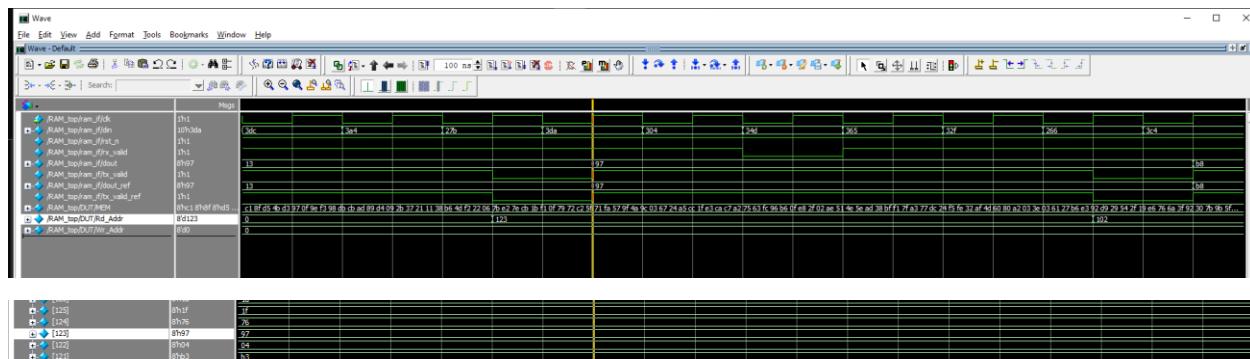
READ ONLY SEQUENCE



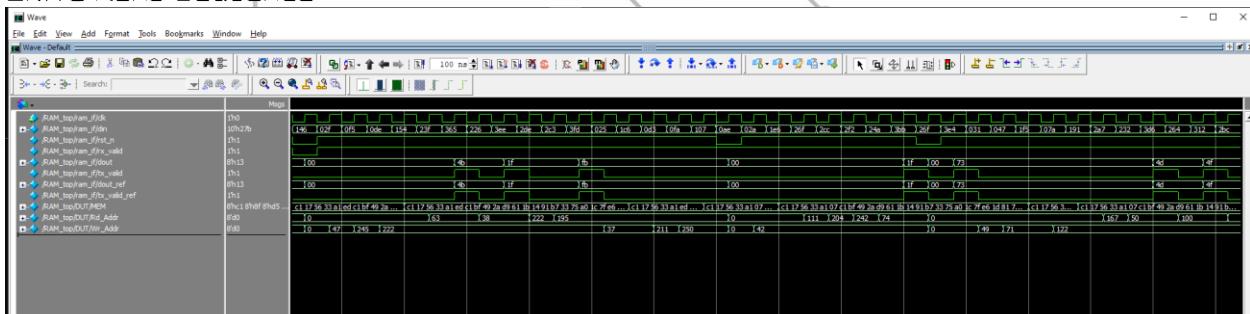
READ ADDRESS



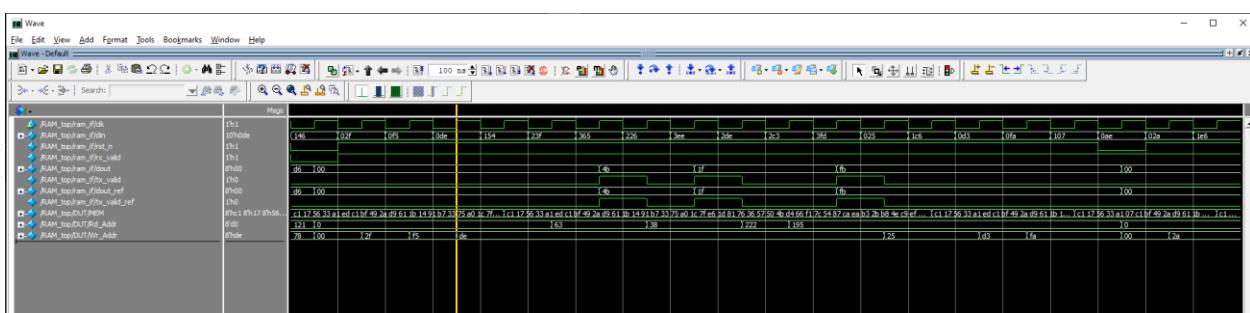
READ DATA



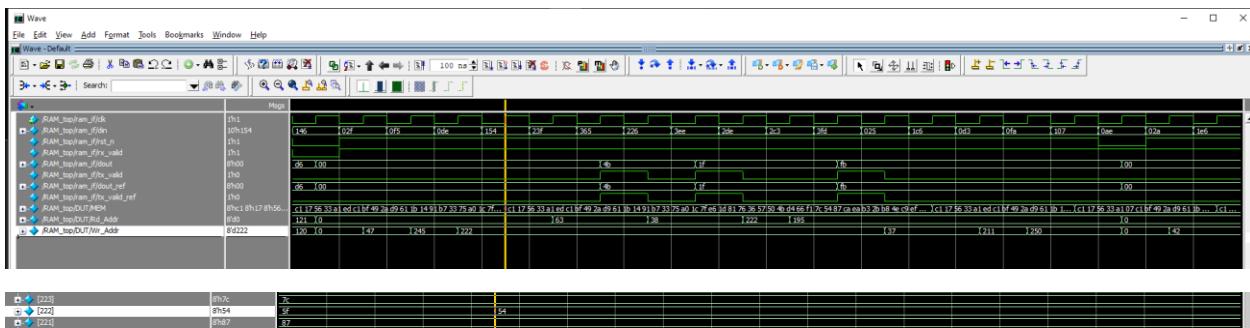
WRITE READ SEQUENCE



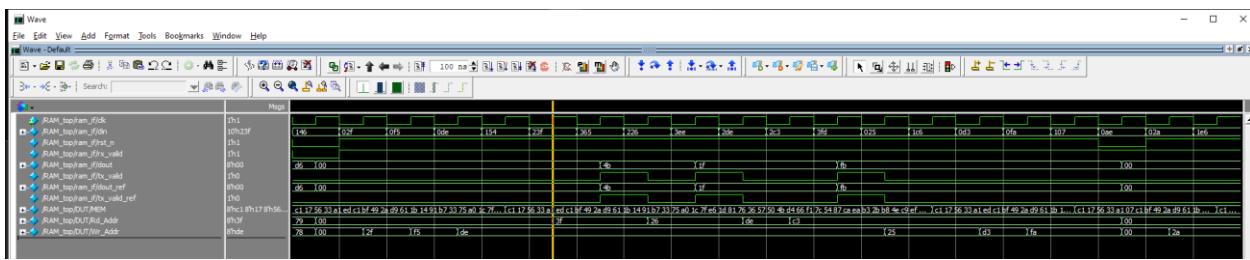
WRITE ADDR



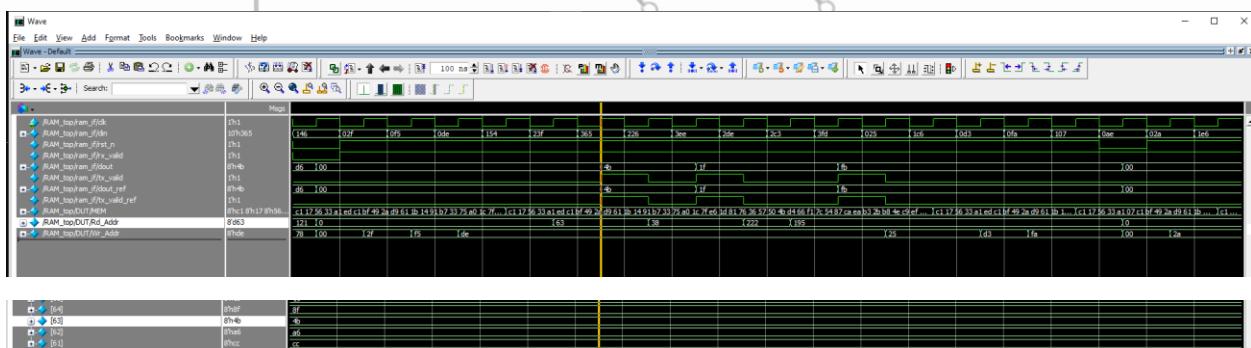
WRITE DATA



READ ADDR



READ-DATA



TRANSCRIPT SNIPPETS

```
# Time: 44257 ns Started: 44255 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 44263 ns Started: 44261 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 47141 ns Started: 47139 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 47149 ns Started: 47147 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 47979 ns Started: 47977 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 48659 ns Started: 48657 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 48711 ns Started: 48709 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 49093 ns Started: 49091 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 51139 ns Started: 51137 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 51431 ns Started: 51431 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 52081 ns Started: 52079 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 52443 ns Started: 52441 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 54723 ns Started: 54721 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 54941 ns Started: 54939 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 55009 ns Started: 55009 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 55207 ns Started: 55205 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 55406 ns Started: 55404 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 56019 ns Started: 56017 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 56555 ns Started: 56553 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 56835 ns Started: 56833 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 57735 ns Started: 57733 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 57907 ns Started: 57905 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# ** Error: Assertion error.
# Time: 58463 ns Started: 58461 ns Scope: RAM_top.DUT.SVA.a_rd_data File: RAM_gva.ev Line: 30
# UVM_INFO Report: uvm_top 8 40002: uvm_test_top.instr_phase@1 Write Read Seq Ended
# UVM_INFO verilog_src/uvm_top.1.lhd@src/base/uvm_object.vh(1247) 8 40002: reporter([TEST_DONE]) run phase is ready to proceed to the 'extract' phase
# UVM_INFO RAM_socboard.av(40) # 40002: uvm_test_top.ram_env.ram_M0 [report_phase] Correct Tests = 0
# ---- UVM Report Summary ---
# * Report counts by severity
# UVM_INFO : 13
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
# * Reports UVM :
#   (Quiescent UVM) 2
#   (UVM_Error) 1
#   (TEST_DONE) 1
#   (report_phase) 1
#   (UVM_Error) 0
# ** Note: $finish : C:/questasim4_2024.1/win64/../../../../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
# Time: 60002 ns Iterations: 61 Instance: /RAM_top
Break in Task uvm_plog/uvm_root:::run at C:/questasim4_2024.1/win64/../../../../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
# UVM_Sim
```

NOTE: A_RD_DATA ASSERTION WILL BE FIXED WHEN USING THE RAM INSIDE THE WRAPPER

SECTION #2: VERIFICATION OF SPI SLAVE

1. VERIFICATION PLAN

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
SLAVE_1	When the rst_n is asserted, the output MISO value should be low & the state cs turns back to IDLE	Directed at the start of the simulation, and then randomized with constraint that drives the reset to be off 99% of the simulation time	-	A checker in the reference model to check for the synchronous reset functionality
SLAVE_2	When the reset is deasserted, SS_n is asserted to tell the slave that the master starts communication & then deasserted to end communication	Randomized with constraint that drives the SS_n to be on every 13 clk cycles if the current state is normal operations or to be on every 23 clk cycles if the current state is read data	Covers transaction duration: 1 → 0 [*13] → 1 for normal operations & Check extended transaction: 1 → 0 [*23] → 1 for read data	A checker in the reference model to check for the functionality
SLAVE_4	When the reset is deasserted, the slave receives MOSI bit by bit & converts the data from serial "MOSI" to parallel	Randomized with a constraint that ensures that the most significant 3 bits follow the sequence of write address -> write data -> read address -> read data	Covers validate correct transition: 0->0->0 (for write address), 0->0->1 (for write data), 1->1->0 (for read address), 1->1->1 (for read data)	A checker in the reference model to check for the functionality
SLAVE_5	When the reset is deasserted, tx_valid is asserted when the slave is ready to send the parallel data to the RAM	Randomized with constraint that drives the tx_valid to be on when the current state is read data & the data is ready to be transmitted (when the counter indicates that)	-	A checker in the reference model to check for the functionality

2. CODE SNIPPETS

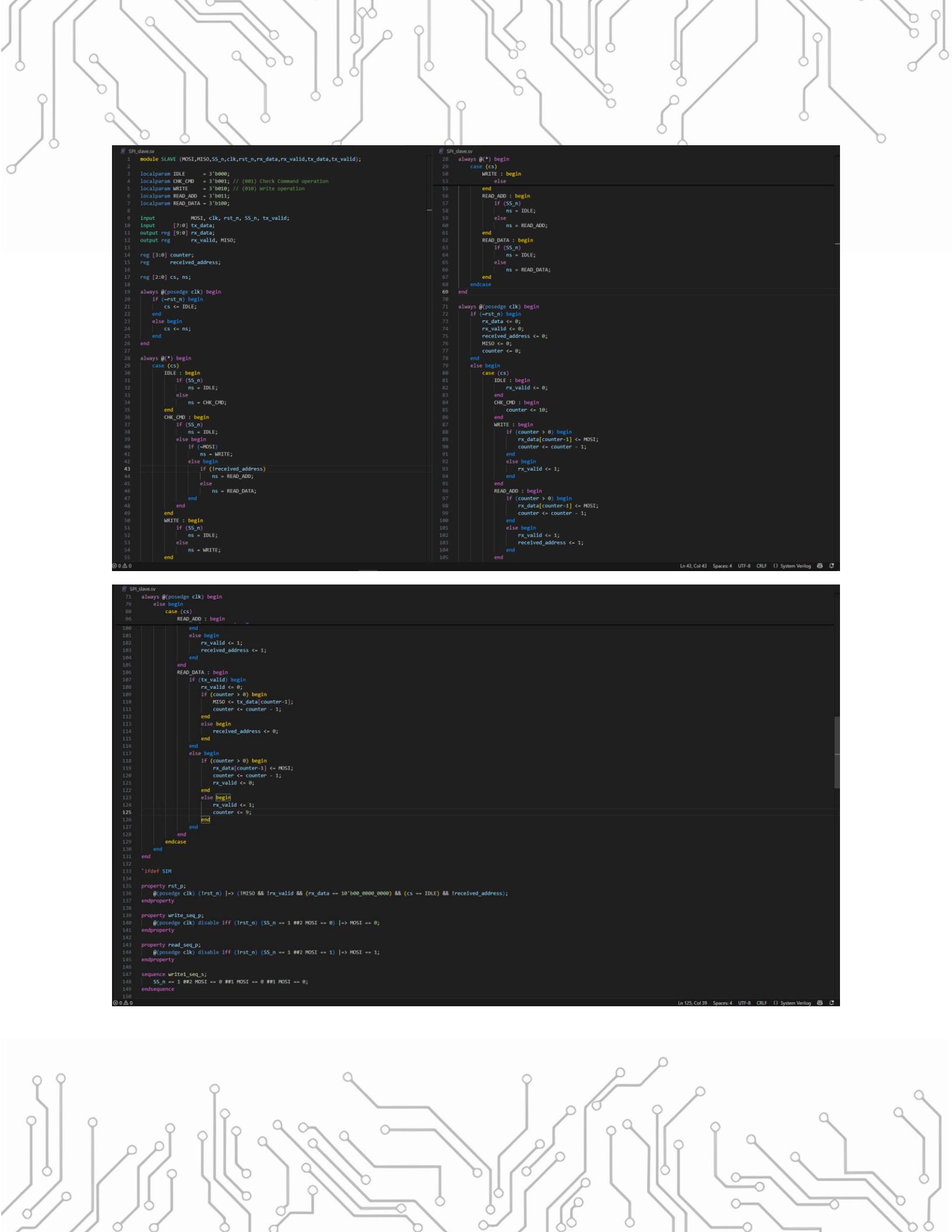
INTERFACE

```
spi_slave_if.sv
1 interface SPI_slave_if(clk);
2     input clk;
3     logic MOSI, rst_n, SS_n, tx_valid;
4     logic [7:0] tx_data;
5     logic [9:0] rx_data, rx_data_ref;
6     logic rx_valid, rx_valid_ref, MISO, MISO_ref;
7 endinterface
```

DESIGN

```
spi_slave.v
1 module SLAVE (MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
2
3 localparam IDLE = 3'b000;
4 localparam CS_OID = 3'b001;
5 localparam WRITE = 3'b010;
6 localparam READ_ADD = 3'b011;
7 localparam READ_DATA = 3'b100;
8
9 input MOSI, clk, rst_n, SS_n, tx_valid;
10 input [7:0] tx_data;
11 output reg [9:0] rx_data;
12 output reg rx_valid, MISO;
13
14 reg [3:0] Counter;
15 reg received_address;
16
17 reg [2:0] cs, ns;
18
19 always @ (posedge clk) begin
20     if (!rst_n) begin
21         cs <= IDLE;
22     end
23     else begin
24         cs <= ns;
25     end
26 end
```

```


  SPi_slave.sv
  1 module SLAVE (MOSI,MISO,SS_n,clk,rst_n,rx_data,tx_data,tx_valid);
  2
  3 localparam IDLE = 3'b000;
  4 localparam CKE_CMD = 3'b001; // (001) Check Command operation
  5 localparam WRITE = 3'b010; // (010) Write operation
  6 localparam READ_ADD = 3'b011;
  7 localparam READ_DATA = 3'b100;
  8
  9 input      MOSI,clk,rst_n,SS_n,rx_valid;
 10 input [7:0] tx_data;
 11 output reg [9:0] rx_data;
 12 output reg      rx_valid,MISO;
 13
 14 reg [3:0] counter;
 15 reg      received_address;
 16
 17 reg [2:0] cs, ns;
 18
 19 always @(posedge clk) begin
 20   if (!rst_n) begin
 21     cs <- IDLE;
 22   end
 23   else begin
 24     cs <- ns;
 25   end
 26 end
 27
 28 always @(*) begin
 29   case (cs)
 30     IDLE : begin
 31       if (SS_n)
 32         ns = IDLE;
 33       else
 34         ns = CKE_CMD;
 35     end
 36     CKE_CMD : begin
 37       if (SS_n)
 38         ns = IDLE;
 39       else begin
 40         if (!MOSI)
 41           ns = WRITE;
 42         else begin
 43           if (received_address)
 44             ns = READ_ADD;
 45           else
 46             ns = READ_DATA;
 47         end
 48       end
 49     end
 50     WRITE : begin
 51       if (SS_n)
 52         ns = IDLE;
 53       else
 54         ns = WRITE;
 55     end
 56   end
 57 end
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105

```



```

  SPi_slave.sv
  28 always @(*) begin
 29   case (cs)
 30     WRITE : begin
 31       if (SS_n)
 32         ns = IDLE;
 33       else
 34         ns = READ_ADD;
 35     end
 36     READ_DATA : begin
 37       if (SS_n)
 38         ns = IDLE;
 39       else
 40         ns = READ_DATA;
 41     end
 42     endcase
 43   end
 44
 45   always @(posedge clk) begin
 46     if (!rst_n) begin
 47       rx_data <= 10'h00_0000_0000;
 48       received_address <= 8'b0;
 49       MISO <= 8'b0;
 50       counter <= 10'b0;
 51     end
 52     else begin
 53       case (cs)
 54         IDLE : begin
 55           rx_valid <= 0;
 56         end
 57         CKE_CMD : begin
 58           Counter <= 10'b0;
 59         end
 60         WRITE : begin
 61           if (counter > 0) begin
 62             rx_data[counter-1] <- MOSI;
 63             counter <= counter - 1;
 64           end
 65           else begin
 66             rx_valid <= 1;
 67           end
 68         end
 69         READ_ADD : begin
 70           if (counter > 0) begin
 71             rx_data[counter-1] <- MOSI;
 72             counter <= counter - 1;
 73           end
 74           else begin
 75             rx_valid <= 1;
 76             received_address <= 1;
 77           end
 78         end
 79       end
 80     end
 81   end
 82 end
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105

```

Ln 43, Col 43 | Spaces: 4 | UTF-8 | CR/LF | {} System Verilog |  


```

  SPi_slave.sv
  71 always @(posedge clk) begin
 72   else begin
 73     case (cs)
 74       READ_ADD : begin
 75         end
 76         rx_valid <= 1;
 77         received_address <= 1;
 78       end
 79     end
 80     end
 81     READ_DATA : begin
 82       if (tx_valid <= 0);
 83         if (counter > 0) begin
 84           MISO <= tx_data[counter-1];
 85           counter <= counter - 1;
 86         end
 87         else begin
 88           received_address <= 0;
 89         end
 90       end
 91       else begin
 92         if (counter > 0) begin
 93           rx_data[counter-1] <- MOSI;
 94           counter <= counter - 1;
 95           rx_valid <= 0;
 96         end
 97         else begin
 98           rx_valid <= 1;
 99           counter <= 0;
 100        end
 101      end
 102    endcase
 103  end
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150

```

Ln 125, Col 39 | Spaces: 4 | UTF-8 | CR/LF | {} System Verilog |  

```

@_SPL_slave.sv
133 `ifndef SIM
134
135 property rst_p;
136   @(posedge clk) (rst_n) |> (!MISO & !rx_valid & (rx_data == 10'b00_0000_0000) && (cs == IDLE) && !received_address);
137 endproperty
138
139 property write_seq_p;
140   @(posedge clk) disable iff (rst_n) (SS_n == 1 ##2 MOSI == 0) |> MOSI == 0;
141 endproperty
142
143 property read_seq_p;
144   @(posedge clk) disable iff (rst_n) (SS_n == 1 ##2 MOSI == 1) |> MOSI == 1;
145 endproperty
146
147 sequence write1_seq_s;
148   SS_n == 1 ##2 MOSI == 0 ##1 MOSI == 0 ##1 MOSI == 0;
149 endsequence
150
151 sequence read1_seq_s;
152   SS_n == 1 ##2 MOSI == 1 ##1 MOSI == 1 ##1 MOSI == 0;
153 endsequence
154
155 sequence write2_seq_s;
156   SS_n == 1 ##2 MOSI == 0 ##1 MOSI == 0 ##1 MOSI == 1;
157 endsequence
158
159 sequence read2_seq_s;
160   SS_n == 1 ##2 MOSI == 1 ##1 MOSI == 1 ##1 MOSI == 1;
161 endsequence
162
163 sequence rx_ss_n_s;
164   ##10 rx_valid #(1:s) SS_n;
165 endsequence
166
167 property rx_valid_p;
168   @(posedge clk) disable iff (rst_n)
169     (write_seq_s or read1_seq_s or write2_seq_s or read2_seq_s) |> rx_ss_n_s;
170 endproperty
171
172 property cs_p1;
173   @(posedge clk) disable iff (rst_n) (SS_n) |> (cs == IDLE);
174 endproperty
175
176 property cs_p2;
177   @(posedge clk) disable iff (rst_n) (cs == IDLE & ISS_n) |> (cs == CHK_CMD);
178 endproperty
179
180 property cs_p3;
181   @(posedge clk) disable iff (rst_n) (cs == CHK_CMD & ISS_n & !MOSI) |> (cs == WRITE);
182 endproperty
183
184 property cs_p4;
185   @(posedge clk) disable iff (rst_n) (cs == CHK_CMD & ISS_n & MOSI & received_address) |> (cs == READ_DATA);
186 endproperty
187

```

(ln 125, Col 39 - Spaces: 4 - UTF-8 - CRLF - SystemVerilog)

```

@_SPL_slave.sv
188
189 property cs_p4;
190   @(posedge clk) disable iff (rst_n) (cs == CHK_CMD & ISS_n & MOSI & received_address) |> (cs == READ_DATA);
191 endproperty
192
193 property cs_p6;
194   @(posedge clk) disable iff (rst_n) (cs == CHK_CMD & ISS_n & MOSI & !received_address) |> (cs == READ_ADD);
195 endproperty
196
197 property cs_p7;
198   @(posedge clk) disable iff (rst_n) (cs == WRITE & ISS_n) |> (cs == WRITE);
199 endproperty
200
201 property cs_p8;
202   @(posedge clk) disable iff (rst_n) (cs == READ_ADD & ISS_n) |> (cs == READ_ADD);
203 endproperty
204
205 property counter_rst_p;
206   @(posedge clk) disable iff (rst_n) (cs == CHK_CMD) |> (counter == 4'b1010);
207 endproperty
208
209 property counter_write_p;
210   @(posedge clk) disable iff (rst_n)
211     ((cs == WRITE || cs == READ_ADD || cs == READ_DATA) && counter > 0) |> (counter == $past(counter) - 1'bl);
212 endproperty
213
214 property received_address_p1;
215   @(posedge clk) disable iff (rst_n) (cs == READ_ADD & counter == 0) |> (received_address);
216 endproperty
217
218 property received_address_p2;
219   @(posedge clk) disable iff (rst_n) (cs == READ_DATA & tx_valid & counter == 0) |> (!received_address);
220 endproperty
221
222 assert property (rst_p);
223 assert property (write_seq_p);
224 assert property (read_seq_p);
225 assert property (rx_valid_p);
226 assert property (cs_p1);
227 assert property (cs_p3);
228 assert property (cs_p4);
229 assert property (cs_p6);
230 assert property (cs_p7);
231 assert property (cs_p8);
232 assert property (cs_p9);
233 assert property (counter_rst_p);
234 assert property (counter_write_p);
235 assert property (received_address_p1);
236 assert property (received_address_p2);
237

```

(ln 125, Col 39 - Spaces: 4 - UTF-8 - CRLF - SystemVerilog)

```

238 cover property (rst_p);
239 cover property (write_seq_p);
240 cover property (read_seq_p);
241 cover property (rx_valid_p);
242 cover property (cs_p1);
243 cover property (cs_p2);
244 cover property (cs_p3);
245 cover property (cs_p4);
246 cover property (cs_p5);
247 cover property (cs_p7);
248 cover property (cs_p8);
249 cover property (cs_p9);
250 cover property (counter_rst_p);
251 cover property (counter_write_p);
252 cover property (received_address_p1);
253 cover property (received_address_p2);
254
255 `endif
256
257 endmodule

```

GOLDEN MODEL



```

spi_slave_golden_model.v
1 module SPI_slave_golden_model (
2   input clk, .rst_n, .SS_n, MOSI, tx_valid,
3   output reg MISO, rx_valid,
4   output reg[9:0] rx_data
5 );
6 parameter IDLE = 3'b000;
7 parameter CHK_CMD = 3'b001;
8 parameter WRITE = 3'b010;
9 parameter READ_ADD = 3'b011;
10 parameter READ_DATA = 3'b100;
11
12 // (* fsm_encoding = "gray" *)
13 reg[2:0] cs, ns;
14 reg rx_type;
15 reg[4:0] cnt;
16
17 always @(posedge clk) begin
18   if(rst_n) cs <= IDLE;
19   else cs <= ns;
20 end
21
22
23 always @(*) begin
24   case(cs)
25     IDLE: begin
26       if(SS_n) ns = CHK_CMD;
27       else ns = IDLE;
28     end
29     CHK_CMD: begin
30       if(SS_n) ns = IDLE;
31       else begin
32         if(~rx_type) ns = WRITE;
33         else begin
34           if(rx_type) ns = READ_DATA;
35           else ns = READ_ADD;
36         end
37       end
38     end
39     WRITE: begin
40       if(SS_n) ns = IDLE;
41       else ns = WRITE;
42     end
43     READ_ADD: begin
44       if(SS_n) ns = IDLE;
45       else ns = READ_ADD;
46     end
47     READ_DATA: begin
48       if(SS_n) ns = IDLE;
49       else ns = READ_DATA;
50     end
51     default: ns = IDLE;
52   endcase
53 end
54
55 always @(posedge clk) begin

```



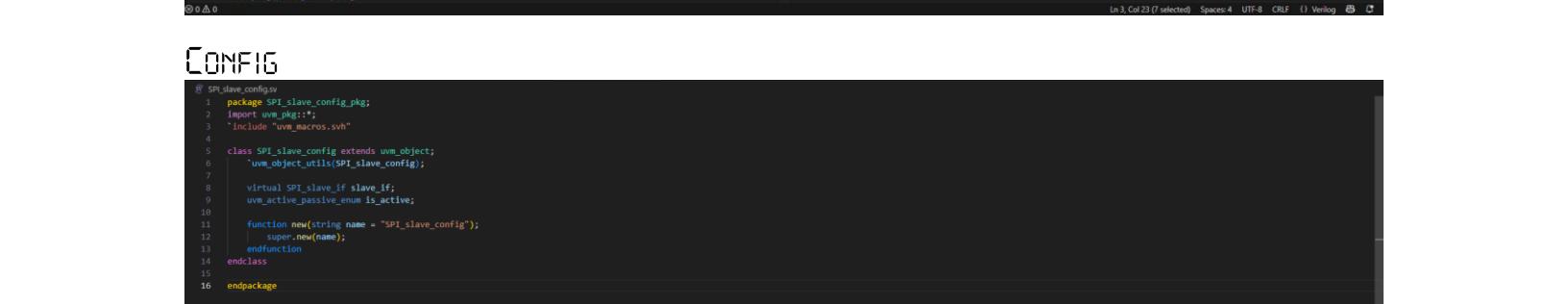
```

SPI_slave_golden_model.v
54   always @ (posedge clk) begin
55     if(rst_n) begin
56       MISO <- 0;
57       rx_type <- 0;
58       rx_valid <- 0;
59       rx_data <- 0;
60       cnt <- 0;
61     end
62     else begin
63       case(cs)
64         IDLE: rx_valid <- 0;
65         CHK_CMD: ns <- 0;
66         WRITE: begin
67           cnt <- cnt + 1;
68           if (cnt < 10) rx_data[9-cnt] <= MOSI;
69           if (cnt == 10) rx_valid <- 1;
70         end
71         READ_ADD: begin
72           cnt <- cnt + 1;
73           if (cnt < 10) rx_data[9-cnt] <= MOSI;
74           if (cnt == 10) begin
75             rx_valid <- 1;
76             rx_type <- 1;
77           end
78         end
79         READ_DATA: begin
80           if(tx_valid) begin
81             cnt <- cnt + 1;
82             if (cnt < 10) rx_data[9-cnt] <= MOSI;
83             else begin
84               rx_type <- 0;
85             end
86             if (cnt == 10) cnt <- 1;
87             rx_valid <- (cnt == 10);
88           end
89           else begin
90             rx_type <- 0;
91             cnt <- cnt + 1;
92             if (cnt >= 2 && cnt <= 9) MISO <- rx_data[9 - cnt];
93             else begin
94               rx_type <- 0;
95             end
96           end
97         end
98       endcase
99     end
100    end
101  end
102 endmodule
103
104

```

Ln 3, Col 23 (7 selected) | Spaces: 4 | UTF-8 | CRLF | Verilog |  

CONFIG

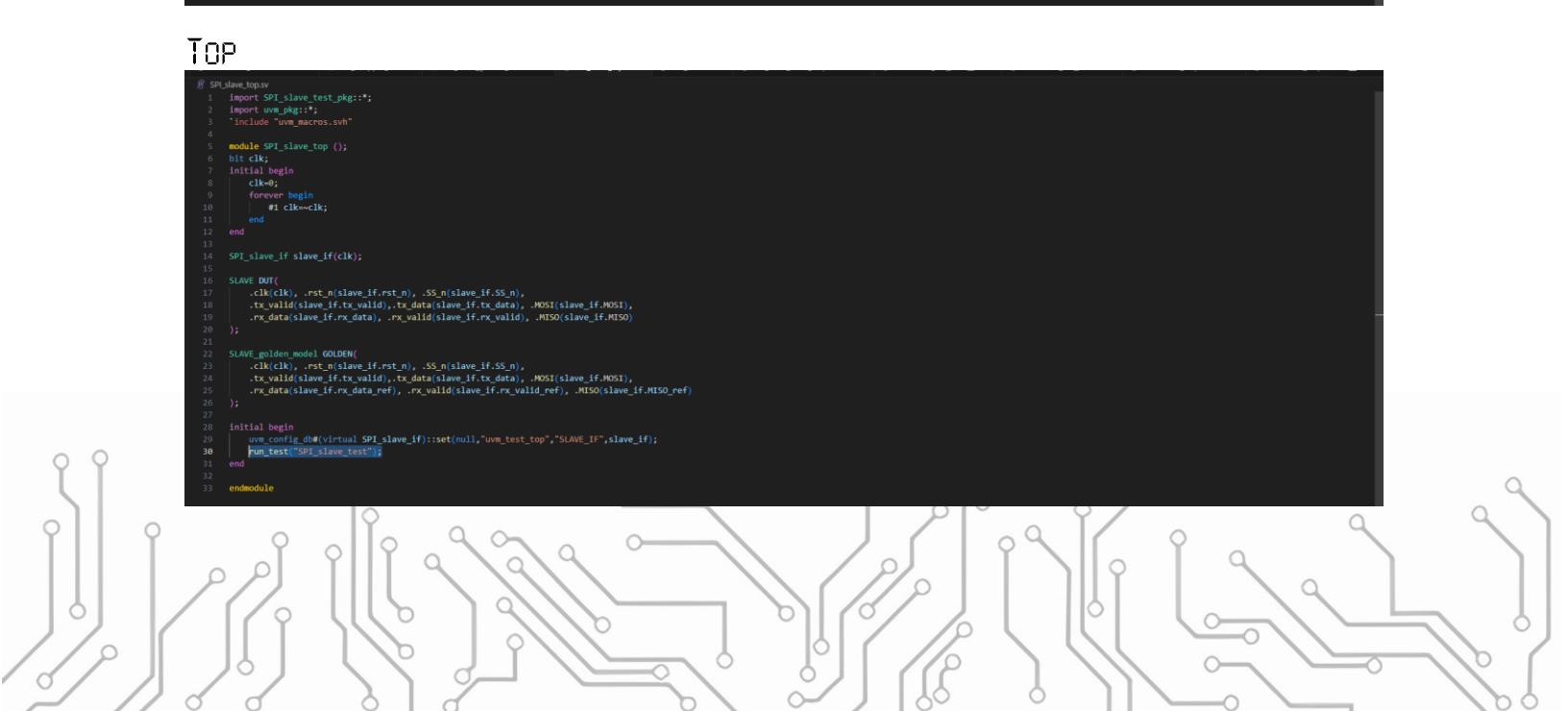


```

SPI_slave.config.v
1 package SPI_slave_config_pkg;
2 import uvm_pkg::*;
3 `include "uvr_macros.svh"
4
5 class SPI_slave_config extends uvm_object;
6   `uvm_object_utils(SPI_slave_config);
7
8   virtual SPI_slave_if slave_if;
9   uvm_active_passive_enum is_active;
10
11   function new(string name = "SPI_slave_config");
12     super.new(name);
13   endfunction
14
15 endpackage
16

```

TOP



```

spi_slave_top.sv
1 import SPI_slave_test_pkg::*;
2 import uvm_pkg::*;
3 `include "uvr_macros.svh"
4
5 module SPI_slave_top ();
6   bit clk;
7   initial begin
8     clk=0;
9     forever begin
10       #1 clk=~clk;
11     end
12   end
13
14   SPI_slave_if slave_if(clk);
15
16   SLAVE_DUT(
17     .clk(clk), .rst_n(slave_if.rst_n), .SS_n(slave_if.SS_n),
18     .tx_valid(slave_if.tx_valid), .tx_data(slave_if.tx_data), .MOSI(slave_if.MOSI),
19     .rx_data(slave_if.rx_data), .rx_valid(slave_if.rx_valid), .MISO(slave_if.MISO)
20   );
21
22   SLAVE_golden_model GOLDEN(
23     .clk(clk), .rst_n(slave_if.rst_n), .SS_n(slave_if.SS_n),
24     .tx_valid(slave_if.tx_valid), .tx_data(slave_if.tx_data), .MOSI(slave_if.MOSI),
25     .rx_data(slave_if.rx_data_ref), .rx_valid(slave_if.rx_valid_ref), .MISO(slave_if.MISO_ref)
26   );
27
28 initial begin
29   uvm_config_db#(virtual SPI_slave_if)::set(null,"uvm_test_top","SLAVE_IF",slave_if);
30   run_test("SPI_slave_test");
31 end
32
33 endmodule

```

TEST

```
8 SPI_Slave.sv
 1 package SPI_slave_test_pkg;
 2 import SPI_slave_env_pkg::*;
 3 import SPI_slave_config_pkg::*;
 4 import SPI_slave_reset_sequence_pkg::*;
 5 import SPI_slave_main_sequence_pkg::*;
 6 import uvm_pkg::*;
 7 `include "uvm_macros.svh"
 8
 9 class SPI_slave_test extends uvm_test;
10   `uvm_component_utils(SPI_slave_test)
11   SPI_slave_env env;
12   SPI_slave_config slave_cfg;
13   SPI_slave_reset_sequence rst_seq;
14   SPI_slave_main_sequence main_seq;
15
16   function new(string name = "SPI_slave_test", uvm_component parent = null);
17     super.new(name, parent);
18   endfunction
19
20   function void build_phase (uvm_phase phase);
21     super.build_phase(phase);
22     env = SPI_SLAVE_ENV::type_id::create("env", this);
23     slave_cfg = SPI_SLAVE_CONFIG::type_id::create("slave_cfg");
24     rst_seq = SPI_SLAVE_RESET_SEQUENCE::type_id::create("rst_seq");
25     main_seq = SPI_SLAVE_MAIN_SEQUENCE::type_id::create("main_seq");
26
27     if (!UVM_CONFIG_DB #(virtual SPI_SLAVE_IF)::get(this, "", "SLAVE_IF", slave_cfg.slave_if))
28       | `UVM_FATAL("build_phase", "Test-unable to get the virtual interface of the slave from the uvm_config_db")
29
30     slave_cfg.is_active = UVM_ACTIVE;
31
32     uvm_config_db #(SPI_SLAVE_CONFIG)::set(this, "", "CFG_SLAVE", slave_cfg);
33   endfunction : build_phase
34
35   task run_phase (uvm_phase phase);
36     super.run_phase(phase);
37     phase.raise_objection(this);
38
39     `UVM_INFO("run_phase", "Reset Asserted", UVM_LOW);
40     rst_seq.start(env.agt.sqr);
41     `UVM_INFO("run_phase", "Reset Deasserted", UVM_LOW);
42
43     `UVM_INFO("run_phase", "Main Sequence Started", UVM_LOW);
44     main_seq.start(env.agt.sqr);
45     `UVM_INFO("run_phase", "Main Sequence Ended", UVM_LOW);
46
47     phase.drop_objection(this);
48   endtask
49 endclass
50 endpackage
```

ENVIRONMENT

```
8 SPI_Slave_env.sv
 1 package SPI_slave_env_pkg;
 2 import SPI_SLAVE_SCOREBOARD_PKG::*;
 3 import SPI_SLAVE_COVERAGE_PKG::*;
 4 import SPI_SLAVE_AGENT_PKG::*;
 5 import UVM_PKG::*;
 6 `include "uvm_macros.svh"
 7
 8 class SPI_SLAVE_ENV extends uvm_env;
 9   `uvm_component_utils(SPI_SLAVE_ENV)
10
11   SPI_SLAVE_AGENT agt;
12   SPI_SLAVE_SCOREBOARD sb;
13   SPI_SLAVE_COVERAGE cov;
14
15   function new(string name = "SPI_SLAVE_ENV", uvm_component parent = null);
16     super.new(name, parent);
17   endfunction
18
19   function void build_phase(uvm_phase phase);
20     super.build_phase(phase);
21     agt = SPI_SLAVE_AGENT::type_id::create("agt", this);
22     sb = SPI_SLAVE_SCOREBOARD::type_id::create("sb", this);
23     cov = SPI_SLAVE_COVERAGE::type_id::create("cov", this);
24   endfunction
25
26   function void connect_phase(uvm_phase phase);
27     super.connect_phase(phase);
28     agt.agt_ap.connect(sb.sb_export);
29     agt.agt_ap.connect(cov.cov_export);
30   endfunction
31 endclass
32 endpackage
```

COVERAGE

```


@ SPI_slave_coverage.sv
1 package SPI_slave_coverage_pkg;
2 import SPI_slave_seq_item_pkg::*;
3 import uvm_pkg::*;
4 `include "uvm_macros.svh"
5
6 class SPI_slave_coverage extends uvm_scoreboard;
7   `uvm_component_utils(SPI_slave_coverage);
8
9   uvm_analysis_export #(SPI_slave_seq_item) cov_export;
10  uvm_tlm_analysis_fifo #(SPI_slave_seq_item) cov_fifo;
11  SPI_slave_seq_item seq_item_cov;
12
13  covergroup covgrp;
14
15    rx_data_cp: coverpoint seq_item_cov.rx_data[9:8] {
16      bins rx_data_all_values[] = {0:3};
17      bins rx_data_trans[] = {0->1}, {1->3}, {1->0}, {0->2},
18      {2->3}, {2->0}, {3->1}, {3->2};
19    }
20
21    SS_n_cp: coverpoint seq_item_cov.SS_n {
22      bins morsel_trans = {1->0["13"]->1};
23      bins extended_trans = {1->0["23"]->1};
24    }
25
26    MOSI_cp: coverpoint seq_item_cov.MOSI{
27      bins write_address = {0->0->0};
28      bins write_data   = {0->0->1};
29      bins read_address = {1->1->0};
30      bins read_data   = {1->1->1};
31    }
32
33    SS_n_MOSI_cr: cross SS_n_cp, MOSI_cp{
34      ignore_bins not_r_data_ext1 = binof(MOSI_cp.write_address) && binof(SS_n_cp.extended_trans);
35      ignore_bins not_r_data_ext2 = binof(MOSI_cp.write_data) && binof(SS_n_cp.extended_trans);
36      ignore_bins not_r_data_ext3 = binof(MOSI_cp.read_address) && binof(SS_n_cp.extended_trans);
37      ignore_bins not_r_data_ext4 = binof(MOSI_cp.read_data) && binof(SS_n_cp.normal_trans);
38    }
39
40  endgroup
41
42  function new(string name = "SPI_slave_coverage", uvm_component parent = null);
43    super.new(name, parent);
44    covgrp = new();
45  endfunction
46
47  function void build_phase(uvm_phase phase);
48    super.build_phase(phase);
49    cov_export = new("cov_export", this);
50    cov_fifo = new("cov_fifo", this);
51  endfunction
52
53  function void connect_phase(uvm_phase phase);
54    super.connect_phase(phase);
55    cov_export.connect(cov_fifo.analysis_export);
56  endfunction


```

In 1, Col 1 | Spaces:4 | UTF-8 | CR/LF | () System Verilog |

SCOREBOARD

```


@ SPI_slave_scoreboard.sv
1 package SPI_slave_scoreboard_pkg;
2 import SPI_slave_seq_item_pkg::*;
3 import uvm_pkg::*;
4 `include "uvm_macros.svh"
5
6 class SPI_slave_scoreboard extends uvm_scoreboard;
7   `uvm_component_utils(SPI_slave_scoreboard)
8
9   uvm_analysis_export #(SPI_slave_seq_item) sb_export;
10  uvm_tlm_analysis_fifo #(SPI_slave_seq_item) sb_fifo;
11  SPI_slave_seq_item seq_item_sb;
12
13  int rx_valid_error_count = 0;
14  int rx_data_error_count = 0;
15  int MISO_error_count = 0;
16  int rx_valid_correct_count = 0;
17  int rx_data_correct_count = 0;
18  int MISO_correct_count = 0;
19
20  function new(string name = "SPI_slave_scoreboard", 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    sb_export = new("sb_export", this);
27    sb_fifo = new("sb_fifo", this);
28  endfunction
29
30  function void connect_phase(uvm_phase phase);
31    super.connect_phase(phase);
32    sb_export.connect(sb_fifo.analysis_export);
33  endfunction
34
35  task run_phase(uvm_phase phase);
36    super.run_phase(phase);
37    forever begin
38      sb_fifo.get(seq_item_sb);
39      if (seq_item_sb.rx_data != seq_item_sb.rx_data_ref) begin
40        `uvm_error("run_phase", $formatf("Comparison of rx_data failed, DUT:%s While rx_data_ref = %h", seq_item_sb.convert2string(), seq_item_sb.rx_data_ref));
41        rx_data_error_count++;
42      end
43      else rx_data_correct_count++;
44
45      if (seq_item_sb.rx_valid != seq_item_sb.rx_valid_ref) begin
46        `uvm_error("run_phase", $formatf("comparison of rx_valid failed, DUT:%s While rx_valid_ref = %b", seq_item_sb.convert2string(), seq_item_sb.rx_valid_ref));
47        rx_valid_error_count++;
48      end
49      else rx_valid_correct_count++;
50
51      if (seq_item_sb.MISO != seq_item_sb.MISO_ref) begin
52        `uvm_error("run_phase", $formatf("comparison of MISO failed, DUT:%s While MISO_ref = %b", seq_item_sb.convert2string(), seq_item_sb.MISO_ref));
53        MISO_error_count++;
54      end
55      else MISO_correct_count++;
56
57    end
58  endtask
59
60  function void report_phase (uvm_phase phase);
61    super.report_phase(phase);
62    `uvm_info("report_phase", "SPI SLAVE SCOREBOARD", UVM_LOW);
63    `uvm_info("report_phase", $formatf("Total successful rx_data: %d", rx_data_correct_count), UVM_LOW);
64    `uvm_info("report_phase", $formatf("Total failed rx_data: %d", rx_data_error_count), UVM_LOW);
65    `uvm_info("report_phase", $formatf("Total successful rx_valid: %d", rx_valid_correct_count), UVM_LOW);
66    `uvm_info("report_phase", $formatf("Total failed rx_valid: %d", rx_valid_error_count), UVM_LOW);
67    `uvm_info("report_phase", $formatf("Total successful MISO: %d", MISO_correct_count), UVM_LOW);
68    `uvm_info("report_phase", $formatf("Total failed MISO: %d", MISO_error_count), UVM_LOW);
69  endfunction
70
71 endpackage


```

In 22, Col 16 | Spaces:4 | UTF-8 | CR/LF | () System Verilog |

AGENT

```
# SPI_slave_agent.sv
1 package SPI_slave_agent_pkg;
2 import SPI_slave_seq_item_pkg::*;
3 import SPI_slave_driver_pkg::*;
4 import SPI_slave_monitor_pkg::*;
5 import SPI_slave_sequencer_pkg::*;
6 import SPI_slave_config_pkg::*;
7 import UVM_PKG::*;
8 `include "uvm_macros.svh"
9
10 class SPI_slave_agent extends uvm_agent;
11   `uvm_component_utils(SPI_slave_agent)
12
13   SPI_slave_driver driver;
14   SPI_slave_monitor monitor;
15   SPI_slave_sequencer sqr;
16   SPI_slave_config slave_cfg;
17   uvm_analysis_port #(SPI_slave_seq_item) agt_ap;
18
19   function new(string name = "SPI_slave_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_slave_config)::get(this,"","CFG_SLAVE",slave_cfg))
26       `uvm_fatal("build_phase","agent - unable to get configuration object");
27
28     if(slave_cfg.is_active == UVM_ACTIVE) begin
29       driver = SPI_slave_driver::type_id::create("driver",this);
30       sqr = SPI_slave_sequencer::type_id::create("sqr",this);
31     end
32     monitor = SPI_slave_monitor::type_id::create("monitor",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(slave_cfg.is_active == UVM_ACTIVE) begin
39       driver.slave_if = slave_cfg.slave_if;
40       driver.seq_item_port.connect(sqr.seq_item_export);
41     end
42     monitor.slave_if = slave_cfg.slave_if;
43     monitor.mon_ap.connect(agt_ap);
44   endfunction
45
46 endclass
47 endpackage
```

DRIVER

```
# SPI_slave_driver.sv
1 package SPI_slave_driver_pkg;
2 import UVM_PKG::*;
3 import SPI_slave_seq_item_pkg::*;
4 `include "uvm_macros.svh"
5
6 class SPI_slave_driver extends uvm_driver #(SPI_slave_seq_item);
7   `uvm_component_utils(SPI_slave_driver);
8
9   virtual SPI_slave_if slave_if;
10  SPI_slave_seq_item stim_seq_item;
11
12  function new(string name = "SPI_slave_driver", uvm_component parent = null);
13    super.new(name, parent);
14  endfunction
15
16  task run_phase(uvm_phase phase);
17    super.run_phase(phase);
18    forever begin
19      stim_seq_item = SPI_slave_seq_item::type_id::create("stim_seq_item");
20      seq_item.port = seq_item.item_port.get_next_item(stim_seq_item);
21      slave_if.rst_n = stim_seq_item.rst_n;
22      slave_if.SS_n = stim_seq_item.SS_n;
23      slave_if.tx_data = stim_seq_item.tx_valid;
24      slave_if.tx_data.ref = slave_if.tx_data.item.tx_valid;
25      slave_if.MOSI = stim_seq_item.MOSI;
26      #(@negedge slave_if.clk);
27      seq_item.port.item_done();
28      `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH);
29    end
30  endtask
31 endclass
32 endpackage
```

MONITOR

```
# SPI_slave_monitor.sv
1 package SPI_slave_monitor_pkg;
2 import SPI_slave_seq_item_pkg::*;
3 import UVM_PKG::*;
4 `include "uvm_macros.svh"
5
6 class SPI_slave_monitor extends uvm_monitor;
7   `uvm_component_utils(SPI_slave_monitor);
8
9   SPI_slave_seq_item rsp_seq_item;
10  virtual SPI_slave_if slave_if;
11  uvm_analysis_port #(SPI_slave_seq_item) mon_ap;
12
13  function new(string name = "SPI_slave_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      rsp_seq_item = SPI_slave_seq_item::type_id::create("rsp_seq_item");
26      #(@edgeedge slave_if.clk);
27      rsp_seq_item.rst_n = slave_if.rst_n;
28      rsp_seq_item.SS_n = slave_if.SS_n;
29      rsp_seq_item.MOSI = slave_if.MOSI;
30      rsp_seq_item.tx_valid = slave_if.tx_valid;
31      rsp_seq_item.tx_data = slave_if.tx_data;
32      rsp_seq_item.rx_valid = slave_if.rx_valid;
33      rsp_seq_item.rx_data = slave_if.rx_data;
34      rsp_seq_item.tx_ref = slave_if.tx_valid.ref;
35      rsp_seq_item.tx_valid.ref = slave_if.tx_valid.ref;
36      rsp_seq_item.tx_data.ref = slave_if.tx_data.ref;
37      rsp_seq_item.MISO.ref = slave_if.MISO.ref;
38      mon_ap.write(rsp_seq_item);
39    end
40  endtask
41 endclass
42 endpackage
```

SEQUENCER

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

SEQUENCE RESET

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

MAIN

```
spi_slave_main_sequence.sv
1 package SPI_slave_main_sequence_pkg;
2 import uvm_pkg::*;
3 import SPI_slave_seq_item_pkg::*;
4 `include "uvm_macros.svh"
5
6 class SPI_slave_main_sequence extends uvm_sequence #(SPI_slave_seq_item);
7   `uvm_object_utils(SPI_slave_main_sequence)
8
9   SPI_slave_seq_item seq_item;
10
11   function new(string name = "SPI_slave_main_sequence");
12    super.new(name);
13   endfunction
14
15   task body;
16    seq_item = SPI_slave_seq_item::type_id::create("seq_item");
17    repeat(100) begin
18      start_item(seq_item);
19      assert(seq_item.randomize());
20      finish_item(seq_item);
21    end
22   endtask
23 endclass
24
25 endpackage
```

SEQUENCE ITEM

```

// SPI_slave_seq_item.sv
1 package SPI_slave_seq_item_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4
5 class SPI_slave_seq_item extends uvm_sequence_item;
6   `uvm_object_utils(SPI_slave_seq_item);
7   rand logic rst_n, MOSI, SS_n, tx_valid;
8   rand logic [7:0] tx_data;
9   logic [9:0] rx_data, rx_data_ref;
10  logic rx_valid, rx_valid_ref, MISO, MISO_ref;
11
12  rand bit [10:0] MOSI_array;
13  static bit [1:0] state = 2'b00;
14  int count;
15
16  function new(string name = "SPI_slave_seq_item");
17    super.new(name);
18  endfunction
19
20  function string convert2string();
21    return $format("%s rst_n = %0h , MOSI = %0h , SS_n = %0h , tx_valid = %0h , MISO = %0h , tx_data = %0h , rx_valid = %0h , rx_data = %0h",
22    super.convert2string(), rst_n, MOSI, SS_n, tx_valid, tx_data, MISO, rx_valid, rx_data);
23  endfunction
24
25  function string convert2string_stimulus();
26    return $format("%s rst_n = %0h , MOSI = %0h , SS_n = %0h , tx_valid = %0h , tx_data = %0h",
27    super.convert2string(), rst_n, MOSI, SS_n, tx_valid, tx_data);
28  endfunction
29
30 // SLAVE_1
31 constraint rst_n_const{
32   |rst_n dist {0:1, 1:99};
33 }
34
35 // SLAVE_2
36 constraint SS_n_const{
37   if(MOSI_array[10:8] == 3'b111){
38     if(count == 23) SS_n == 1;
39     else SS_n == 0;
40   }
41   else{
42     if(count == 13) SS_n == 1;
43     else SS_n == 0;
44   }
45 }
46
47 // SLAVE_3
48 constraint MOSI_array_const{
49   MOSI_array[10] == state[1];
50   MOSI_array[9:8] == state;
51 }
52
53 // SLAVE_4
54 constraint MOSI_const{
55   if (count > 0 && count < 12) MOSI == MOSI_array[11 - count];
56 }

```

Ln 63, Col 1 | Spaces: 4 | UTF-8 | CRLF | {} System Verilog

```

57
58 // SLAVE_4
59 constraint MOSI_const{
60   if (count > 0 && count < 12) MOSI == MOSI_array[11 - count];
61 }
62
63 // SLAVE_5
64 constraint tx_valid_const{
65   if(MOSI_array[10:8] == 3'b111 && count > 13 && count < 24) tx_valid == 1;
66   else tx_valid == 0;
67 }
68
69 function void pre_randomize();
70   if(count==0) MOSTI_array.rand_mode(1);
71   else MOSTI_array.rand_mode(0);
72
73   if(MOSI_array[10:8] == 3'b111 && count == 12) tx_data.rand_mode(1);
74   else tx_data.rand_mode(0);
75
76 endfunction
77
78 function void post_randomize();
79   if(SS_n || !rst_n) count = 0;
80   else count++;
81   if(!rst_n) state = 0;
82   else if(count == 0) state++;
83
84 endfunction
85
86 endclass
87
88 endpackage

```

Ln 61, Col 1 | Spaces: 4 | UTF-8 | CRLF | {} System Verilog

Do File

```

1 vlib work
2 vlog +define+SIM -f src_list.list +cover -covercells
3 vsim -voptargs+acc work.SPI_slave_top -classdebug -uvmcontrol=all -cover
4
5 add wave */SPI_slave_top/slave_if/*
6
7 add wave $position insertpoint \
8 sim:/SPI_slave_top/DUT/c \
9 sim:/SPI_slave_top/DUT/m \
sim:/SPI_slave_top/DUT/counter \
sim:/SPI_slave_top/GOLDEN/cont \
sim:/SPI_slave_top/DUT/received_address \
sim:/SPI_slave_top/GOLDEN/rx_type
10
11 coverage exclude -src SPI_slave.sv -line 38 -code s
12 coverage exclude -src SPI_slave.sv -line 37 -code b
13 coverage exclude -src SPI_slave.sv -line 128 -code b
14
15
16 run -all
17
18 coverage exclude -cvgpath [/SPI_slave_coverage_pkg/SPI_slave_coverage/covgrp//SPI_slave_coverage_pkg:SPI_slave_coverage::covgrp /rx_data_cp/rx_data_trans[0=>2]]
19
20 coverage save SLAVE.ucdb -onexit

```

Ln 13, Col 1 | Spaces: 4 | UTF-8 | CRLF | {} System Verilog

3. BUG REPORT

Bug #1: SHOULD BE '!RECEIVED_ADDRESS' INSTEAD TO ENTER READ_ADD STATE BEFORE

```
42  
43     if (received_address)  
44         ns = READ_ADD;  
45     else  
46         ns = READ_DATA;  
47     end
```

AFTER

```
43     if (!received_address)  
44         ns = READ_ADD;  
45     else  
46         ns = READ_DATA;  
47     end
```

Bug #2: COUNTER SHOULD BE 9 SINCE IT TAKES ONE EXTRA CYCLE TO RECEIVE DATA FROM RAM BEFORE

```
121  
122     else begin  
123         rx_valid <= 1;  
124         counter <= 8;  
125     end
```

AFTER

```
122     else begin  
123         rx_valid <= 1;  
124         counter <= 9;  
125     end
```

Bug #3: COUNTER SHOULD BE LOW WHEN THE RST SIGNAL IS ACTIVE BEFORE

```
71     always @ (posedge clk) begin  
72         if (~rst_n) begin  
73             rx_data <= 0;  
74             rx_valid <= 0;  
75             received_address <= 0;  
76             MISO <= 0;  
77         end
```

AFTER

```
71     always @ (posedge clk) begin  
72         if (~rst_n) begin  
73             rx_data <= 0;  
74             rx_valid <= 0;  
75             received_address <= 0;  
76             MISO <= 0;  
77             counter <= 0;  
78         end
```

4. COVERAGE

Code Coverage Branch

```
SLAVE_cvr.txt - Notepad
File Edit Format View Help
Branch Coverage:
Enabled Coverage      Bins    Hits    Misses Coverage
-----  -----  -----  -----
Branches            38     38      0  100.00%
=====Branch Details=====
Branch Coverage for instance /SPI_slave_top/OUT
Line   Item       Count   Source
----  ----  -----
File SPI_slave.sv
-----IF Branch-----
20     1          5359  Count coming in to IF
20           215   if (~rst_n) begin
23     1          5144  else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----CASE Branch-----
29     1          15029 Count coming in to CASE
30           2599   IDLE : begin
36     1          1992   CHK_CMD : begin
50     1          5062   WRITE : begin
56     1          2436   READ_ADD : begin
62     1          2939   READ_DATA : begin
1           All False Count
Branch totals: 6 hits of 6 branches = 100.00%
-----IF Branch-----
31     1          2599  Count coming in to IF
31           1157   if (SS_n)
33     1          1442   else
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
37     1          1992  Count coming in to IF
39           1992   else begin
Branch totals: 1 hit of 1 branch = 100.00%
-----IF Branch-----
40     1          1992  Count coming in to IF
40           1046   if (~MOSI)
42     1          946   else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
43     1          946   Count coming in to IF
43           678    if (!received_address)
Ln 1, Col 1  100% Windows (CRLF) UTF-8
```

```
SLAVE_cvr.txt - Notepad
File Edit Format View Help
-----IF Branch-----
43     1          946   Count coming in to IF
43           678    if (!received_address)
45     1          268   else
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
51     1          5062  Count coming in to IF
51           674    if (SS_n)
53     1          4388  else
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
57     1          2436  Count coming in to IF
57           272    if (SS_n)
59     1          2164  else
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
63     1          2939  Count coming in to IF
63           211    if (SS_n)
65     1          2728  else
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
72     1          20001 Count coming in to IF
72           215    if (~rst_n) begin
78     1          19786 else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----CASE Branch-----
79     1          19786 Count coming in to CASE
80           1345   IDLE : begin
83     1          1334   CHK_CMD : begin
86     1          8549   WRITE : begin
95     1          3404   READ_ADD : begin
105    1          5154   READ_DATA : begin
Branch totals: 5 hits of 5 branches = 100.00%
-----IF Branch-----
87     1          8549  Count coming in to IF
87           7207   if (counter > 0) begin
91     1          1342   else begin
Ln 177, Col 84  100% Windows (CRLF) UTF-8
```

SLAVE_cv.txt - Notepad

File Edit Format View Help

105 1 5154 READ_DATA : begin

Branch totals: 5 hits of 5 branches = 100.00%

----- IF Branch -----
 87 1 8549 Count coming in to IF
 if (counter > 0) begin
 91 1 1342 else begin

Branch totals: 2 hits of 2 branches = 100.00%

----- IF Branch -----
 96 1 3404 Count coming in to IF
 if (counter > 0) begin
 100 1 2863 541 else begin

Branch totals: 2 hits of 2 branches = 100.00%

----- IF Branch -----
 106 1 5154 Count coming in to IF
 if (tx_valid) begin
 116 1 2208 2954 else begin

Branch totals: 2 hits of 2 branches = 100.00%

----- IF Branch -----
 108 1 2200 Count coming in to IF
 if (counter > 0) begin
 112 1 1782 418 else begin

Branch totals: 2 hits of 2 branches = 100.00%

----- IF Branch -----
 117 1 2954 Count coming in to IF
 if (counter > 0) begin
 122 1 2718 236 else begin

Branch totals: 2 hits of 2 branches = 100.00%

STATEMENT

SLAVE_cv.txt - Notepad

File Edit Format View Help

Statement Coverage:
 Enabled Coverage
 Statements Bins Hits Misses Coverage

 Statements 38 38 0 100.00%

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

Statement Coverage for instance /SPI_slave_top/DUT --

Line	Item	Count	Source
1		-----	-----
File SPI_slave.sv			module SLAVE (MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
2			
3			localparam IDLE = 3'b000;
4			localparam CHK_CMD = 3'b001;
5			localparam WRITE = 3'b010;
6			localparam READ_ADD = 3'b011;
7			localparam READ_DATA = 3'b100;
8			
9			input MOSI, clk, rst_n, SS_n, tx_valid;
10			input [7:0] tx_data;
11			output reg [9:0] rx_data;
12			output reg rx_valid, MISO;
13			
14			reg [3:0] counter;
15			reg received_address;
16			
17			reg [2:0] cs, ns;
18			
19	1	5359	always @(posedge clk) begin
20			if (~rst_n) begin
21	1	215	cs <- IDLE;
22			end
23			else begin
24	1	5144	cs <- ns;

Ln 233, Col 89 100% Windows (CRLF) UTF-8

```

File Edit Format View Help
25
26
27
28 1 15929 always #(*) begin;
29
30   case (tx)
31     IDLE : begin
32       if (S5_o)
33         m = IDLE;
34       else
35         m = CMK_CMD;
36
37     CMK_CMD : begin
38       if (S5_o)
39         m = IDLE;
40       else begin
41         if (~MOSI)
42           m = WRITE;
43         else begin
44           if (!received_address)
45             m = READ_400;
46           else
47             m = READ_DATA;
48         end
49       end
50     WRITE : begin
51       if (S5_o)
52         m = IDLE;
53       else
54         m = WRITE;

```

Le 231, Col 89 100% Windows (CR/LF) UTF-8

```

File Edit Format View Help
5
56
57
58 1 272   READ_A00 : begin
59
60 1 2164  end
61
62   READ_DATA : begin
63     if (S5_o)
64       m = IDLE;
65     else
66       m = READ_DATA;
67
68   end
69 end
70
71 1 20001 always #(posedge clk) begin
72   if (~rst_n)
73     rx_data <= 0;
74   rx_valid <= 0;
75   received_address <= 0;
76   MISO <= 0;
77
78   end
79   else begin
80     case (tx)
81       IDLE : begin
82         rx_valid <= 0;
83       end
84       CMK_CMD : begin
85         Counter <= 10;

```

Le 231, Col 89 100% Windows (CR/LF) UTF-8

```

File Edit Format View Help
52
53 1 1334  counter <= 10;
54
55   end
56
57   WRITE : begin
58     if (Counter > 0) begin
59       rx_data[counter-1] <= MISO;
60       Counter <= Counter - 1;
61     end
62     else begin
63       rx_valid <= 1;
64     end
65   end
66
67   READ_A00 : begin
68     if (Counter > 0) begin
69       rx_data[counter-1] <= MISO;
70       Counter <= Counter - 1;
71     end
72     else begin
73       rx_valid <= 1;
74       received_address <= 1;
75     end
76
77   end
78
79   READ_DATA : begin
80     if (rx_valid <= 0)
81       if (Counter > 0) begin
82         rx_valid <= 1;
83         if (rx_data[Counter-1] == MISO)
84           rx_data[Counter-1] <= MISO;
85         Counter <= Counter - 1;
86         rx_valid <= 0;
87       end
88     else begin
89       rx_valid <= 1;
90       rx_data[Counter-1] <= MISO;
91       Counter <= Counter - 1;
92       rx_valid <= 0;
93     end
94   end
95
96   else begin
97     if (Counter > 0) begin
98       rx_data[Counter-1] <= MISO;
99       Counter <= Counter - 1;
100      rx_valid <= 0;
101    end
102    else begin
103      rx_valid <= 1;
104      Counter <= 0;
105    end
106  end
107
108  end
109  end
110  end
111
112  end
113  end
114
115
116
117 1 2730
118 1 2731
119 1 2732
120 1 2733
121
122
123 1 236
124 1 236

```

Le 231, Col 89 70% Windows (CR/LF) UTF-8

TOGGLE

Toggle Coverage:
Enabled Coverage

Toggles 72 72 0 100.0%

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

Toggle Coverage for instance /SPI_slave_top/DUT --

Node	1H->0L	0L->1H	"Coverage"
MISO	1	1	100.00
MOSI	1	1	100.00
SS_n	1	1	100.00
clk	1	1	100.00
counter[0:3]	1	1	100.00
cs[0:2]	1	1	100.00
ns[0:2]	1	1	100.00
received_address	1	1	100.00
rst_n	1	1	100.00
rx_data[0:9]	1	1	100.00
rx_valid	1	1	100.00
tx_data[0:7]	1	1	100.00
tx_valid	1	1	100.00

Total Node Count = 36
Toggled Node Count = 36
Untoggled Node Count = 0

Toggle Coverage = 100.00% (72 of 72 bins)

FSM

FSM Coverage:
Enabled Coverage

FSM States 5 5 0 100.00%
FSM Transitions 8 8 0 100.00%

-----FSM Details-----

FSM Coverage for instance /SPI_slave_top/OUT --

FSM_ID: cs

Current State Object : cs

State Value MapInfo :

Line	State Name	Value
30	IDLE	0
36	CHK_CMD	1
62	READ_DATA	4
56	READ_ADD	3
50	WRITE	2

Covered States :

State	Hit_count
IDLE	1359
CHK_CMD	1345
READ_DATA	531
READ_ADD	599
WRITE	1525

Covered Transitions :

Line	Trans_ID	Hit_count	Transition
34	0	1345	IDLE -> CHK_CMD
46	1	266	CHK_CMD -> READ_DATA
44	2	302	CHK_CMD -> READ_ADD
41	3	766	CHK_CMD -> WRITE
38	4	11	CHK_CMD -> IDLE
64	5	266	READ_DATA -> IDLE
58	6	302	READ_ADD -> IDLE
52	7	765	WRITE -> IDLE

Summary

FSM States 5 5 0 100.00%
FSM Transitions 8 8 0 100.00%

FUNCTIONAL COVERAGE

File	Edit	View	Help
COVERAGE COVERAGE			
Covergroup	Metric	Goal	Bins Status
TYPE /SPI_slave_coverage_pkg/SPI_slave_coverage/covgrp	100.00%	100	- Covered
covered/total bins:	21	21	-
missing/total bins:	0	21	-
% Hit:	100.00%	100	-
Coverpoint rx_data_cp	95.66%	100	- Uncovered
covered/total bins:	11	12	-
missing/total bins:	1	12	-
% Hit:	95.66%	100	-
Coverpoint rx_data_all_cp	100.00%	100	- Covered
covered/total bins:	2	2	-
missing/total bins:	0	2	-
% Hit:	100.00%	100	-
Coverpoint MOSI_cp	100.00%	100	- Covered
covered/total bins:	4	4	-
missing/total bins:	0	4	-
% Hit:	100.00%	100	-
Coverpoint SS_MOSI_cr	100.00%	100	- Covered
covered/total bins:	4	4	-
missing/total bins:	0	4	-
% Hit:	100.00%	100	-
Covergroup Instance V/SPI_slave_coverage_pkg::SPI_slave_coverage/covgrp	100.00%	100	- Covered
covered/total bins:	21	21	-
missing/total bins:	0	21	-
% Hit:	100.00%	100	-
Coverpoint rx_data_cp	100.00%	100	- Covered
covered/total bins:	11	11	-
missing/total bins:	0	11	-
% Hit:	100.00%	100	-
bin rx_data_all_values[0]	6202	1	- Covered
bin rx_data_all_values[1]	4298	1	- Covered
bin rx_data_all_values[2]	1114	1	- Covered
bin rx_data_all_values[3]	4187	1	- Covered
bin rx_data_trans[0]=1	344	1	- Covered
bin rx_data_trans[1]=1	327	1	- Covered
bin rx_data_trans[2]=0	149	1	- Covered
bin rx_data_trans[2]=3	259	1	- Covered
bin rx_data_trans[3]=0	158	1	- Covered
bin rx_data_trans[3]=1	102	1	- Covered
bin rx_data_trans[3]=2	409	1	- Covered
Coverpoints SS_p_cp	100.00%	100	- Covered
covered/total bins:	2	2	-
missing/total bins:	0	2	-
% Hit:	100.00%	100	-
bin_normal_trans	774	1	- Covered
bin_normal_trans	124	1	- Covered
Coverpoint MOSI_cd	100.00%	100	- Covered
covered/total bins:	4	4	-
missing/total bins:	0	4	-
% Hit:	100.00%	100	-
bin_write_address	2946	1	- Covered
bin_rdata	151	1	- Covered
bin read_address	2470	1	- Covered
bin read_data	2672	1	- Covered
Coverpoints SS_DC_cr	100.00%	100	- Covered
covered/total bins:	4	4	-
missing/total bins:	0	4	-
% Hit:	100.00%	100	-
Auto, Default and User Defined Bins:			
bin rx_data_rx_data_ext0	26	1	- Covered
bin (normal_txtrans,read_address)	107	1	- Covered
bin (normal_txtrans,write_data)	99	1	- Covered
bin (normal_txtrans,write_address)	104	1	- Covered
Illegal and Ignore Bins:			
ignore_bin not_r_data_ext4	82	1	- Occurred
ignore_bin not_r_data_ext1	27	1	- Occurred
ignore_bin not_r_data_ext2	36	1	- Occurred
ignore_bin not_r_data_ext1	22	1	- Occurred
TOTAL COVERAGE COVERAGE: 100.00% COVERGROUP TYPES: 1			

Ln 233, Col 89 80% Windows (CRLF) UTF-8

ASSERTION COVERAGE

DIRECTIVE COVERAGE:						
Name	Unit	Detail	Lang	File/Line	Hits	Status
/SPI_slave_top/DUT/cover__received_address_p1	SLAVE	Verilog	SVA	SPI_slave.sv(253)	412	Covered
/SPI_slave_top/DUT/cover__received_address_p1	SLAVE	Verilog	SVA	SPI_slave.sv(252)	537	Covered
/SPI_slave_top/DUT/cover__counter_write_p	SLAVE	Verilog	SVA	SPI_slave.sv(251)	1449	Covered
/SPI_slave_top/DUT/cover__counter_st_p	SLAVE	Verilog	SVA	SPI_slave.sv(250)	1321	Covered
/SPI_slave_top/DUT/cover__cs_p9	SLAVE	Verilog	SVA	SPI_slave.sv(249)	4889	Covered
/SPI_slave_top/DUT/cover__cs_p8	SLAVE	Verilog	SVA	SPI_slave.sv(248)	3187	Covered
/SPI_slave_top/DUT/cover__cs_p7	SLAVE	Verilog	SVA	SPI_slave.sv(247)	7780	Covered
/SPI_slave_top/DUT/cover__cs_p6	SLAVE	Verilog	SVA	SPI_slave.sv(246)	297	Covered
/SPI_slave_top/DUT/cover__cs_p4	SLAVE	Verilog	SVA	SPI_slave.sv(245)	265	Covered
/SPI_slave_top/DUT/cover__cs_p3	SLAVE	Verilog	SVA	SPI_slave.sv(244)	79	Covered
/SPI_slave_top/DUT/cover__cs_p2	SLAVE	Verilog	SVA	SPI_slave.sv(243)	1334	Covered
/SPI_slave_top/DUT/cover__cs_p1	SLAVE	Verilog	SVA	SPI_slave.sv(242)	1132	Covered
/SPI_slave_top/DUT/cover_rx_valid_p	SLAVE	Verilog	SVA	SPI_slave.sv(241)	830	Covered
/SPI_slave_top/DUT/cover__read_seq_p	SLAVE	Verilog	SVA	SPI_slave.sv(240)	562	Covered
/SPI_slave_top/DUT/cover__write_seq_p	SLAVE	Verilog	SVA	SPI_slave.sv(239)	551	Covered
/SPI_slave_top/DUT/cover__rst_p	SLAVE	Verilog	SVA	SPI_slave.sv(238)	215	Covered

TOTAL DIRECTIVE COVERAGE: 100.00% COVERS: 16

Name	File/Line	Failure Count	Pass Count
/SPI_slave_top/DUT/assert_received_address_p1	SPI_slave.sv(136)	0	1
/SPI_slave_top/DUT/assert_rxdata_p1	SPI_slave.sv(235)	0	1
/SPI_slave_top/DUT/assert_counter_write_p	SPI_slave.sv(234)	0	1
/SPI_slave_top/DUT/assert_counter_st_p	SPI_slave.sv(233)	0	1
/SPI_slave_top/DUT/assert_rxdata_p2	SPI_slave.sv(232)	0	1
/SPI_slave_top/DUT/assert_rxdata_p3	SPI_slave.sv(231)	0	1
/SPI_slave_top/DUT/assert_rxdata_p4	SPI_slave.sv(230)	0	1
/SPI_slave_top/DUT/assert_rxdata_p5	SPI_slave.sv(229)	0	1
/SPI_slave_top/DUT/assert_rxdata_p6	SPI_slave.sv(228)	0	1
/SPI_slave_top/DUT/assert_rxdata_p7	SPI_slave.sv(227)	0	1
/SPI_slave_top/DUT/assert_rxdata_p8	SPI_slave.sv(226)	0	1
/SPI_slave_top/DUT/assert_rxdata_p9	SPI_slave.sv(225)	0	1
/SPI_slave_top/DUT/assert_rx_valid_p	SPI_slave.sv(224)	0	1
/SPI_slave_top/DUT/assert_rxdata_p10	SPI_slave.sv(223)	0	1
/SPI_slave_top/DUT/assert_rxdata_p11	SPI_slave.sv(222)	0	1
/SPI_slave_top/DUT/assert_rxdata_p12	SPI_slave.sv(221)	0	1
/SPI_slave_main_sequence_pkg/SPI_slave_main_sequence/body#(Ud1lx7241691917)immed_19	SPI_slave_main_sequence.sv(19)	0	1

NOTES:

1- RODED A DEFAULT CASE THEN EXCLUDE IT TO ACHIEVE 100% CODE COVERAGE

2- EXCLUSION DONE TO THE BIN RX_DATA_TRANS (2=>0) TO ACHIEVE 100 % FUNCTIONAL COVERAGE (IT WILL BE FIXED IN THE WRAPPER)

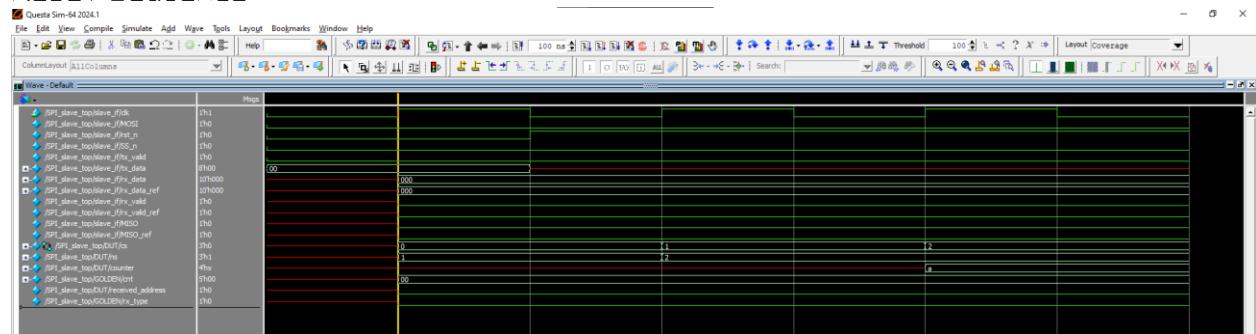
5. ASSERTIONS TABLE

Feature	Assertion
Checks that when rst_n is asserted, MISO & rx_valid & rx_data & received_address are all low and cs turns back to IDLE state	<pre>@(posedge clk) (!rst_n) => (!MISO && !rx_valid && (rx_data == 10'b00_0000_0000) && (cs == IDLE) && !received_address)</pre>
Checks that when SS_n is deasserted and after 2 clk cycles MOSI is low, the MOSI should be low after another one cycle to have a right sequence of MOSI that drives the SPI to function a write operation correctly with RAM	<pre>@(posedge clk) disable iff (!rst_n) (SS_n == 1 ##2 MOSI == 0) => MOSI == 0</pre>
Checks that when SS_n is deasserted and after 2 clk cycles MOSI is high, the MOSI should be high after another one cycle to have a right sequence of MOSI that drives the SPI to function a read operation correctly with RAM	<pre>@(posedge clk) disable iff (!rst_n) (SS_n == 1 ##2 MOSI == 1) => MOSI == 1</pre>
Checks that when SS_n is deasserted and is followed by right sequence of MOSI, the rx_valid is asserted after 10 clk cycles to transmit the data to RAM and SS_n is asserted eventually after that to end communication with the master	<pre>sequence write1_seq_s; SS_n == 1 ##2 MOSI == 0 ##1 MOSI == 0 ##1 MOSI == 0; endsequence sequence read1_seq_s; SS_n == 1 ##2 MOSI == 1 ##1 MOSI == 1 ##1 MOSI == 0; endsequence sequence write2_seq_s; SS_n == 1 ##2 MOSI == 0 ##1 MOSI == 0 ##1 MOSI == 1; endsequence sequence read2_seq_s; SS_n == 1 ##2 MOSI == 1 ##1 MOSI == 1 ##1 MOSI == 1; endsequence sequence rx_ss_n_s; ##10 rx_valid ##[1:\$] SS_n; endsequence</pre> <pre>@(posedge clk) disable iff (!rst_n) (write1_seq_s or read1_seq_s or write2_seq_s or read2_seq_s) -> rx_ss_n_s</pre>
Checks that when SS_n is deasserted, the current state turns back to IDLE state to end communication and be ready for a new one	<pre>@(posedge clk) disable iff (!rst_n) (SS_n => (cs == IDLE))</pre>
Checks that when SS_n is asserted & the current state is IDLE, the next current state is CHK_CMD	<pre>@(posedge clk) disable iff (!rst_n) (cs == IDLE && !SS_n) => (cs == CHK_CMD)</pre>

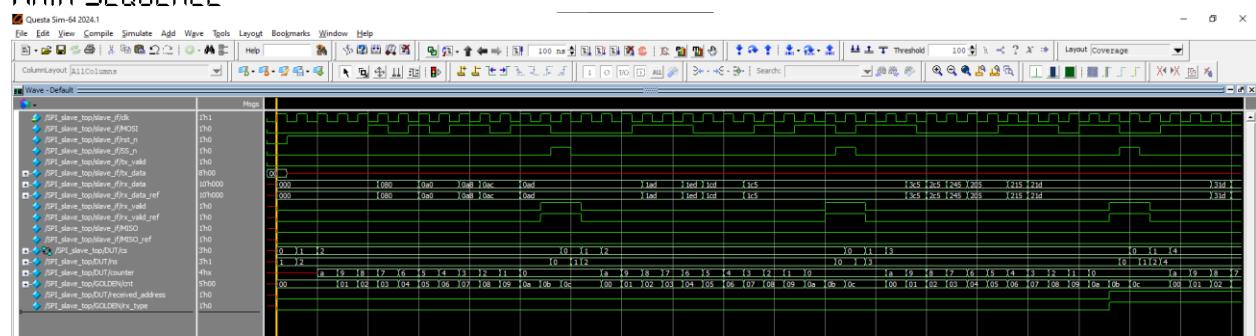
<p>Checks that when SS_n is asserted & the current state is IDLE, if the MOSI == 0 then the next current state is WRITE</p>	<pre>@(posedge clk) disable iff (!rst_n) (cs == CHK_CMD && !SS_n && !MOSI) => (cs == WRITE)</pre>
<p>Checks that when SS_n is asserted & the current state is IDLE, if the MOSI == 1 and the received_address is high then the next current state is READ_DATA</p>	<pre>@(posedge clk) disable iff (!rst_n) (cs == CHK_CMD && !SS_n && MOSI && received_address) => (cs == READ_DATA)</pre>
<p>Checks that when SS_n is asserted & the current state is IDLE, if the MOSI == 1 and the received_address is low then the next current state is READ_ADD</p>	<pre>@(posedge clk) disable iff (!rst_n) (cs == CHK_CMD && !SS_n && MOSI && !received_address) => (cs == READ_ADD)</pre>
<p>Checks that when SS_n is deasserted & the current state is WRITE, the current state stays WRITE for the next cycle</p>	<pre>@(posedge clk) disable iff (!rst_n) (cs == WRITE && !SS_n) => (cs == WRITE)</pre>
<p>Checks that when SS_n is deasserted & the current state is READ_ADD, the current state stays READ_ADD for the next cycle</p>	<pre>@(posedge clk) disable iff (!rst_n) (cs == READ_ADD && !SS_n) => (cs == READ_ADD)</pre>
<p>Checks that when SS_n is deasserted & the current state is READ_DATA, the current state stays READ_DATA for the next cycle</p>	<pre>@(posedge clk) disable iff (!rst_n) (cs == READ_DATA && !SS_n) => (cs == READ_DATA)</pre>
<p>Checks that when the current state is CHK_CMD, the counter resets to 10 to start a new operation</p>	<pre>@(posedge clk) disable iff (!rst_n) (cs == CHK_CMD) => (counter == 4'b1010)</pre>
<p>Checks that when the current state is a normal operation, the counter should decrement every clk cycle</p>	<pre>@(posedge clk) disable iff (!rst_n) ((cs == WRITE cs == READ_ADD cs == READ_DATA) && counter > 0) => (counter == \$past(counter) - 1'b1)</pre>
<p>Checks that when the current state is a READ_ADD & the counter is zero, the received_address should be high to make the next read state READ_DATA</p>	<pre>@(posedge clk) disable iff (!rst_n) (cs == READ_ADD && counter == 0) => (received_address)</pre>
<p>Checks that when the current state is a READ_DATA & the counter is zero & tx_valid is asserted, the received_address should be low to make the next read state READ_ADD</p>	<pre>@(posedge clk) disable iff (!rst_n) (cs == READ_DATA && tx_valid && counter == 0) => (!received_address)</pre>

6. SIMULATION SNIPPETS

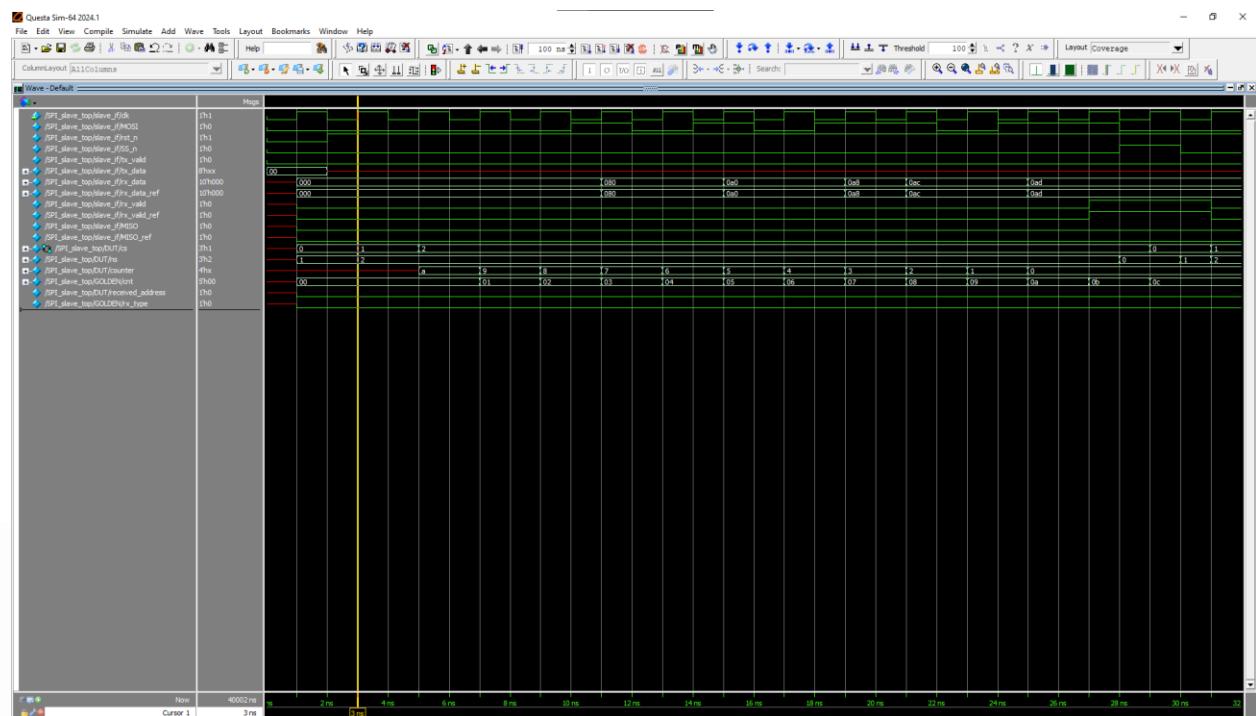
RESET SEQUENCE

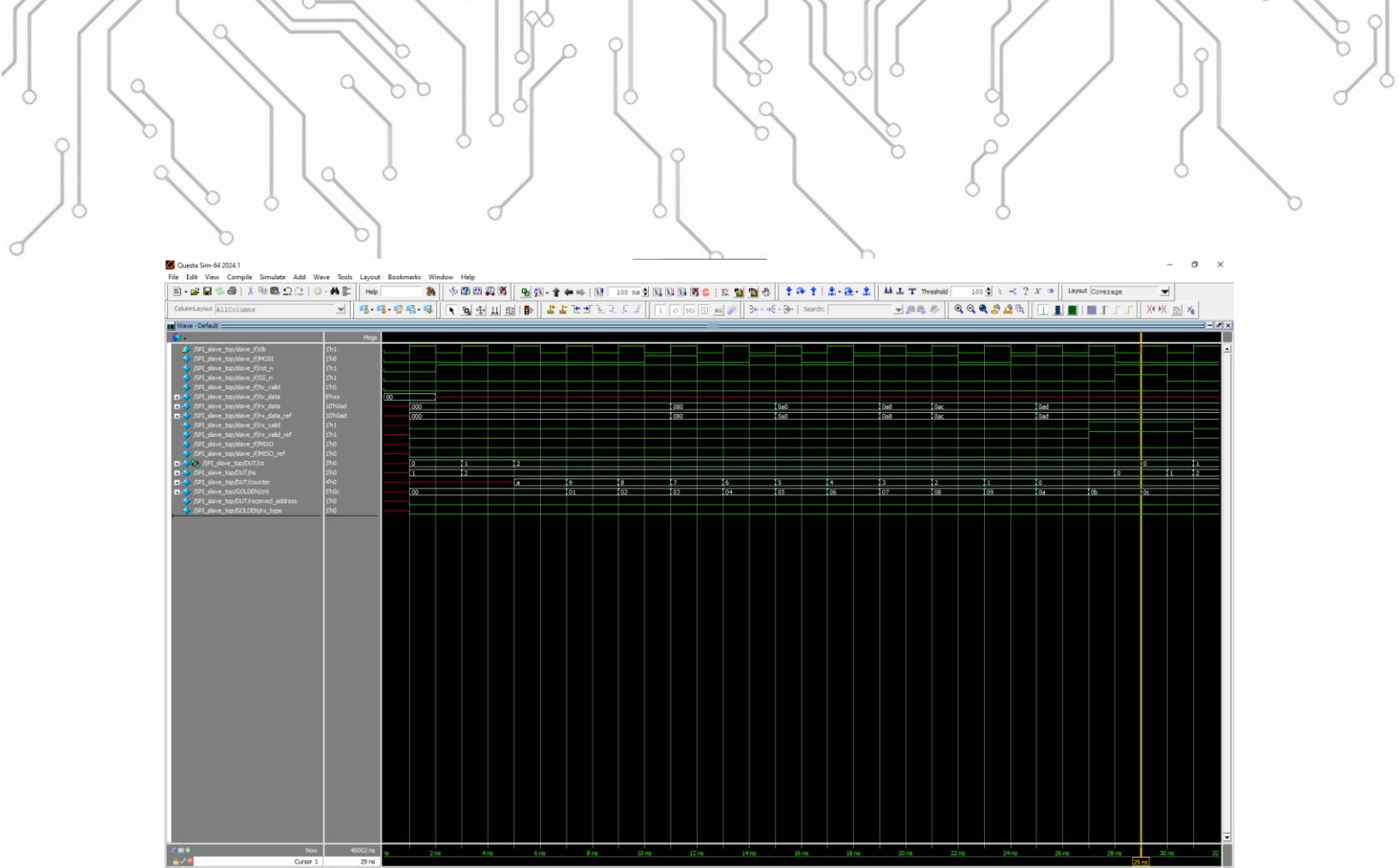


MAIN SEQUENCE

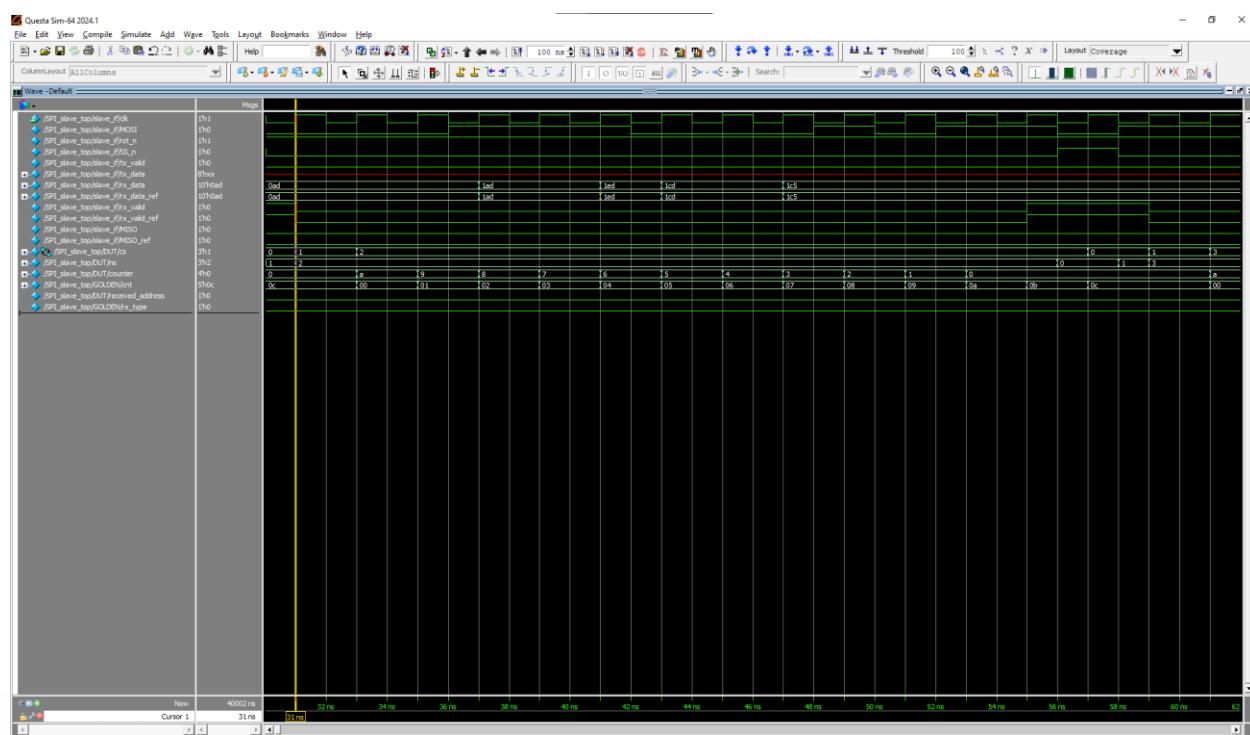


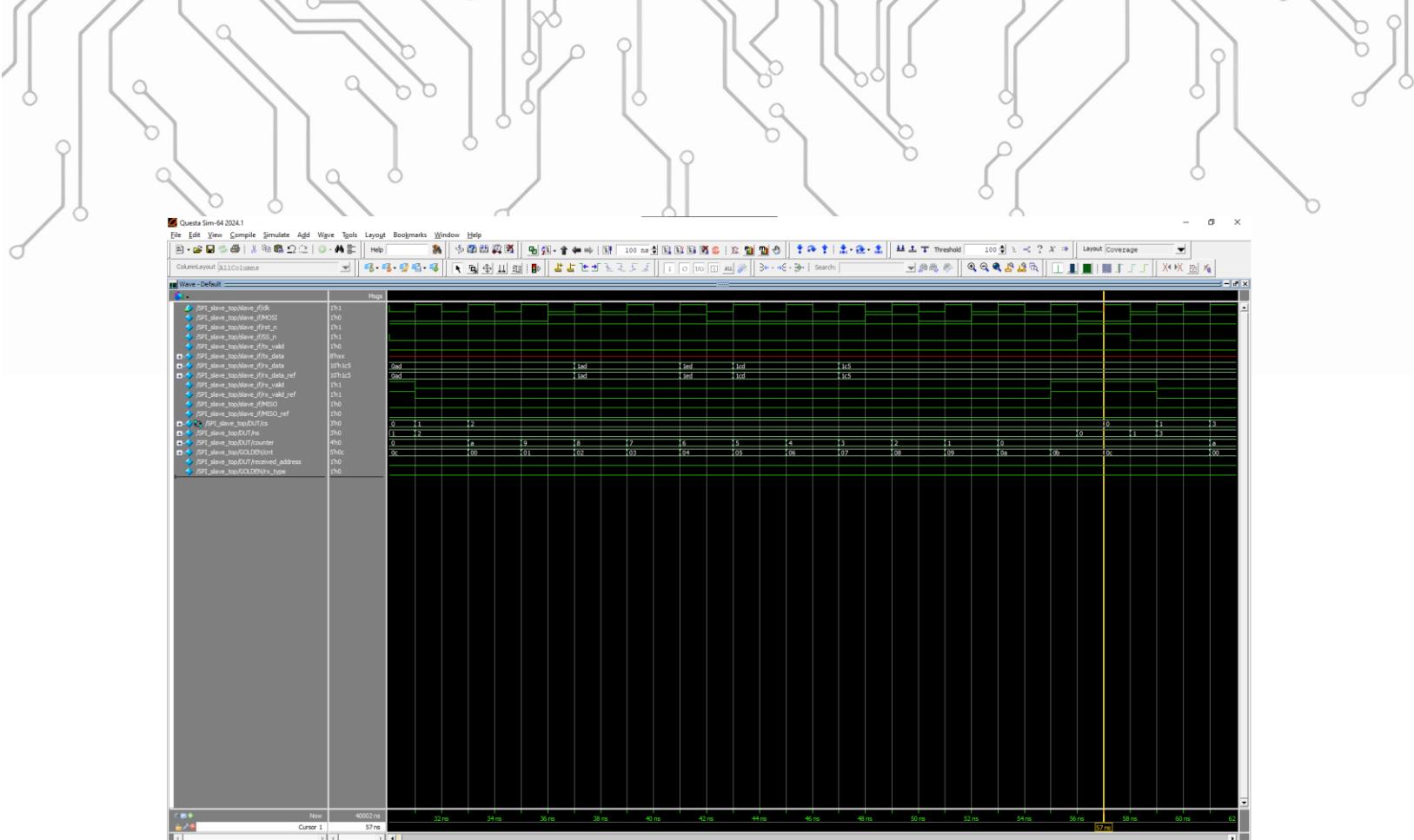
WRITE ADDRESS: (FROM 3NS TO 29 NS, 13 CLK CYCLES)



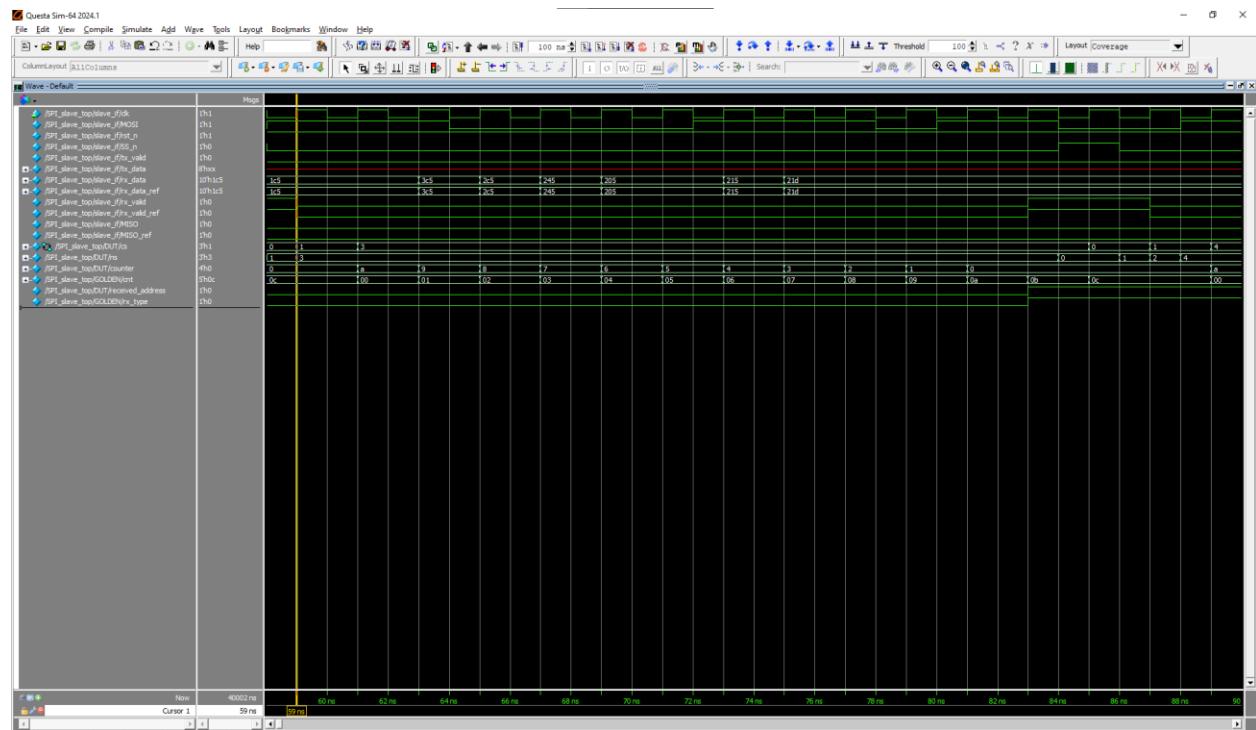


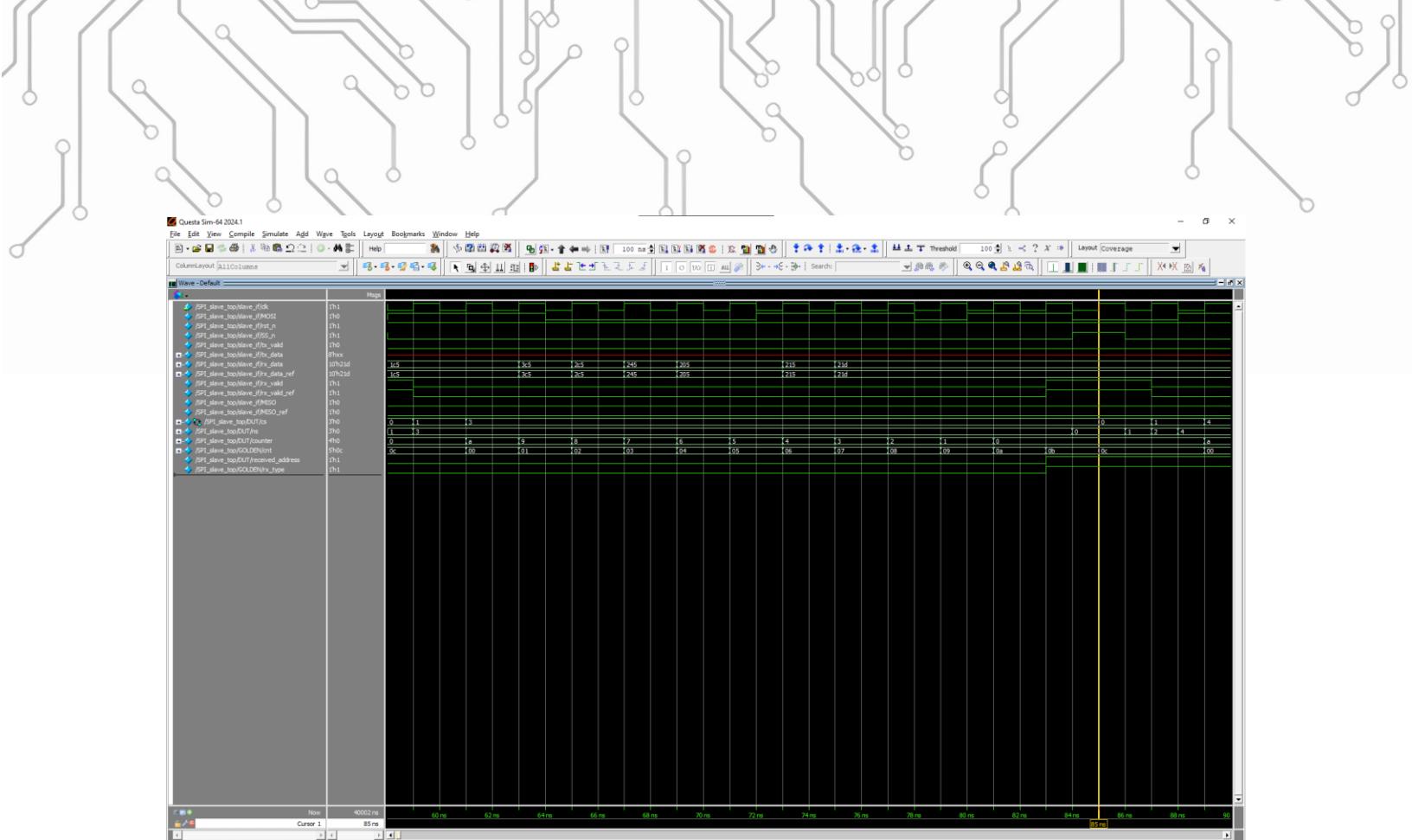
WRITE DBTB: (EROM 31NS TO 57NS) (30K CYCLES)



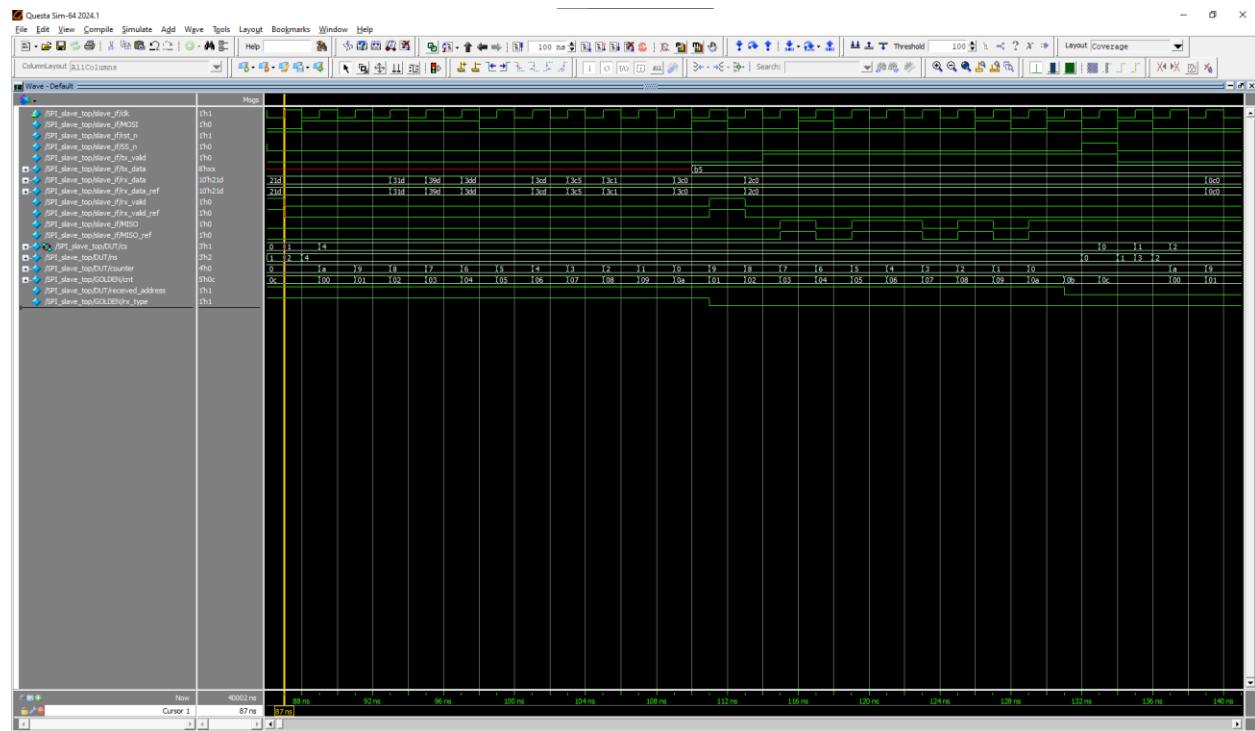


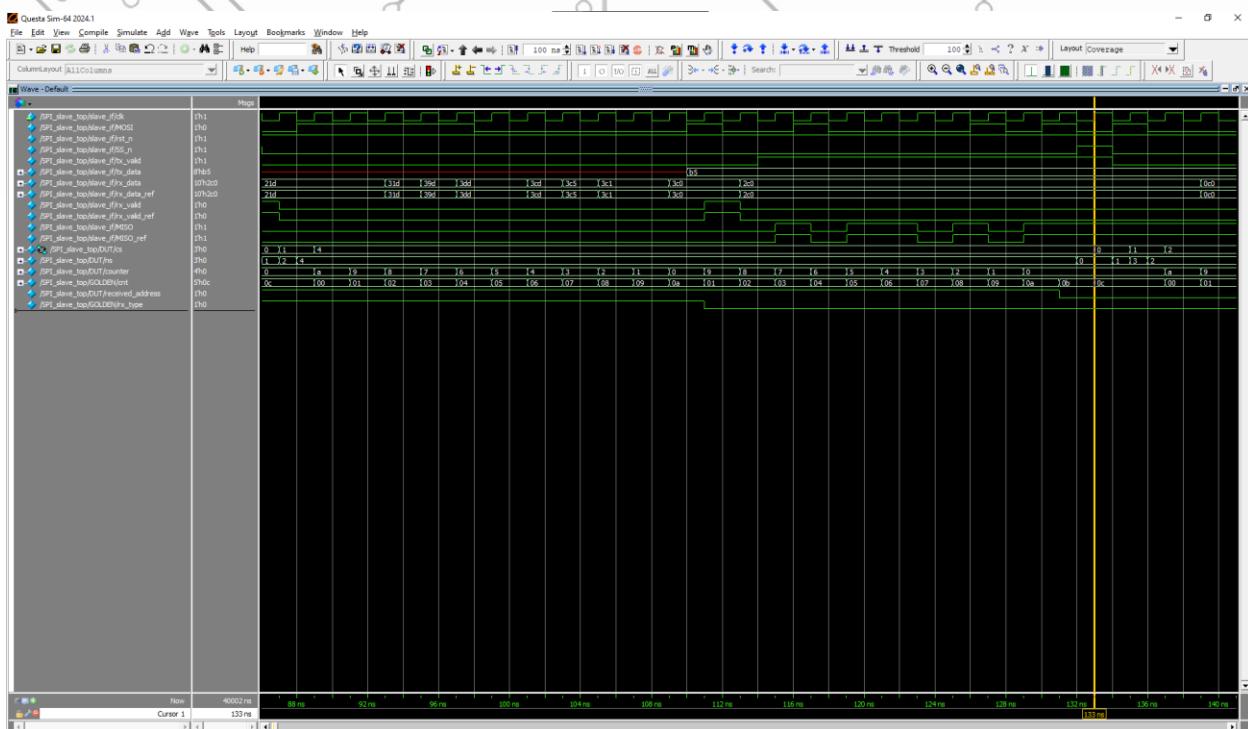
READ ADDRESS: (FROM 59 NS TO 85 NS, 13 CLK CYCLES)





RERO DATA: (FROM 87 NS 133 NS, 23 CLK CYCLES)





TRANSCRIPT SNIPPETS

```

Transport

(C) 2011-2013 Cypress Semiconductor Corp.

***** IMPORTANT RELEASE NOTES *****

You are using a version of the UVM library that has been compiled
with UVM_NO_DEPRECATED undefined.

See https://www.edn.org/web/view.pfid=3313 for more details.

You are using a version of the UVM library that has been compiled
with UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.

See https://www.edn.org/web/view.pfid=3770 for more details.

(Specify -DUM_NO_RELAXES to turn off this notice)

UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) 0: reporter [Questa UVM] questa_uvm::init(all)
UVM_INFO 0: reporter [RHTS] Running test SFI_slave_test...
UVM_INFO 0: reporter [UVM] uvm_top [run_phase] Reset Asserted
*****
* Questa UVM Transaction Recording Turned ON
* recording detail has been turned ON
* turning "recording_detail" to off:
* uvm_config_db::write( :>, null, "", "recording_detail", 0);
* uvm_config_db::write( uvm_biasstream_ni, null, "", "recording_detail", 0);
* uvm_config_db::write( uvm_biasstream_ni, null, "", "recording_detail", 0);
* uvm_config_db::write( uvm_biasstream_ni, null, "", "recording_detail", 0);

UVM_INFO SFI_slave_test.uv(42) 2: uvm_top [run_phase] Reset Deasserted
UVM_INFO SFI_slave_test.uv(44) 2: uvm_top [run_phase] Main Sequence Started
UVM_INFO SFI_slave_test.uv(45) 2: uvm_top [run_phase] Main Sequence Ended
UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_subjection.svh(1247) #40002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO SFI_slave_boardctrl.uv(4) 40002: uvm_top.env.sv [report_phase] Total successful rx_valid: 20000
UVM_INFO SFI_slave_boardctrl.uv(4) 40002: uvm_top.env.sv [report_phase] Total failed rx_data: 0
UVM_INFO SFI_slave_boardctrl.uv(4) 40002: uvm_top.env.sv [report_phase] Total failed rx_valid: 0
UVM_INFO SFI_slave_boardctrl.uv(4) 40002: uvm_top.env.sv [report_phase] Total successful rx_MISO: 20000
UVM_INFO SFI_slave_boardctrl.uv(67) 40002: uvm_top.env.sv [report_phase] Total failed MISO: 0

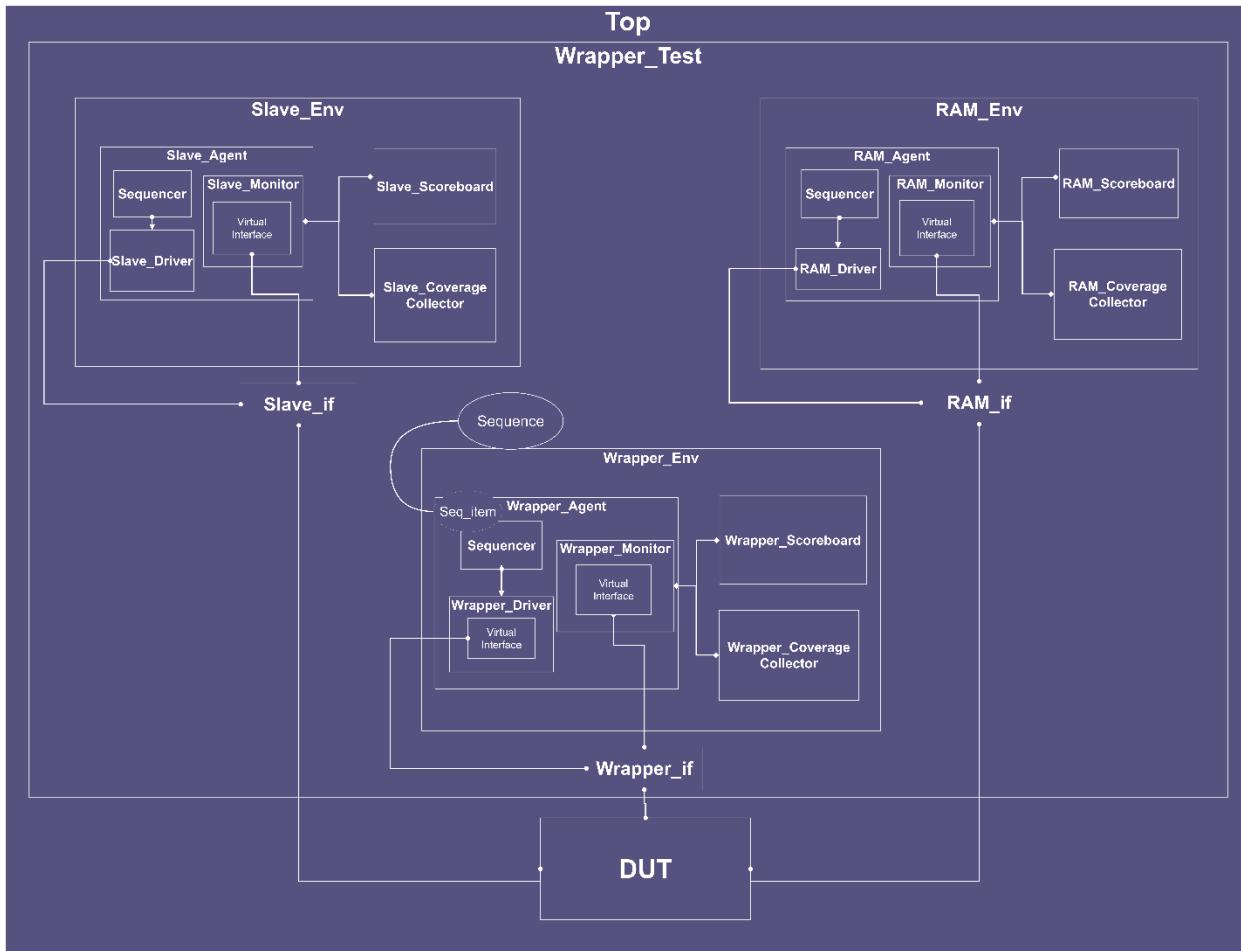
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO : 15
UVM_WARNING : 0
UVM_ERROR : 0
UVM_FATAL : 0
*** Report counts by id
(Questa UVM) 2
(UVM) 1
(TEST_DONE) 1
(report_phase) 7
(run_phase) 4
***** finish : C:\questasim4_2024.1\win64\..\verilog_src\uvm-1.1d\src\base\uvm_root.svh(430)
Time: 40002 ns Iteration: 41 Instance: SFI_slave_top

```

SECTION *3: VERIFICATION OF SPI WRAPPER

I. UVM STRUCTURE



UVM TESTBENCH OVERVIEW

THE TOP MODULE INSTRUMENTS THE DESIGN UNDER VERIFICATION (DUT) MODULES (SPI WRAPPER, SPI WRAPPER REFERENCE, SPI SLAVE, AND RAM) AND CREATES THEIR CORRESPONDING INTERFACES, WHICH ARE THEN PASSED TO THE RESPECTIVE MODULES. IT ALSO GENERATES THE SYSTEM CLOCK AND CONNECTS IT TO ALL INTERFACES. FINALLY, IT SETS THE CREATED INTERFACES INSIDE THE UVM CONFIGURATION DATABASE AND STARTS THE WRAPPER TEST.

IN THE BUILD PHASE, THE WRAPPER TEST CREATES THE ENVIRONMENTS FOR ALL DESIGN MODULES (SPI WRAPPER, SPI SLAVE, AND RAM), AS WELL AS A CONFIGURATION OBJECT FOR EACH ENVIRONMENT AND THE RANDOMIZATION SEQUENCES USED FOR VERIFICATION. THE TEST THEN RETRIEVES THE INTERFACES FROM THE TOP MODULE, ASSIGNS THEM TO THEIR RESPECTIVE CONFIGURATION OBJECTS, AND STORES THESE OBJECTS IN THE UVM CONFIGURATION DATABASE. THE SPI SLAVE AND RAM AGENTS ARE SET AS PASSIVE AGENTS.

DURING THE RUN PHASE, THE TEST STARTS EACH RANDOMIZATION SEQUENCE CREATED IN THE BUILD PHASE.

EVERY ENVIRONMENT, IN ITS BUILD PHASE, CREATES THE AGENT, SCOREBOARD, AND COVERAGE COLLECTOR. IN THE CONNECT PHASE, IT CONNECTS THE AGENT'S ANALYSIS PORTS TO BOTH THE SCOREBOARD AND THE COVERAGE COLLECTOR.

EVERY AGENT, IN ITS BUILD PHASE, CREATES ITS CONFIGURATION OBJECT, RETRIEVES THE ONE PASSED FROM THE TEST, AND THEN CREATES THE MONITOR. IF THE AGENT IS ACTIVE, IT ALSO CREATES THE DRIVER AND SEQUENCER. IN THE CONNECT PHASE, THE AGENT CONNECTS THE DRIVER AND MONITOR VIRTUAL INTERFACES TO THE ONES IN THE CONFIGURATION OBJECT (WHICH POINT TO THE ACTUAL INTERFACES IN THE TOP MODULE). IT ALSO CONNECTS ITS ANALYSIS PORT TO THE MONITOR PORT TO RECEIVE MONITORED TRANSACTIONS FOR THE SCOREBOARD AND COVERAGE COLLECTOR.

THE SEQUENCER RECEIVES RANDOMIZED SEQUENCE ITEMS FROM THE SEQUENCE STARTED IN THE TEST AND PASSES THEM TO THE DRIVER.

THE DRIVER RECEIVES THESE SEQUENCE ITEMS FROM THE SEQUENCER AND DRIVES THE DUT'S INPUT SIGNALS THROUGH THE INTERFACE.

IN THE BUILD PHASE, THE MONITOR CREATES ITS OWN SEQUENCE ITEM, SAMPLES SIGNAL VALUES FROM THE DUT VIA THE VIRTUAL INTERFACE, FILLS THE SEQUENCE ITEM WITH THESE VALUES, AND SENDS IT BACK TO THE AGENT.

AFTER THE AGENT RECEIVES THE MONITORED SEQUENCE ITEM FROM THE MONITOR, IT FORWARDS IT TO BOTH THE SCOREBOARD AND THE COVERAGE COLLECTOR.

THE SCOREBOARD RECEIVES THE MONITORED SEQUENCE ITEM, COMPARES IT WITH THE EXPECTED (GOLDEN) MODEL OUTPUT, AND REPORTS AN ERROR IF A MISMATCH OCCURS.

THE COVERAGE COLLECTOR DEFINES THE COVERGROUPS REQUIRED TO MEASURE THE FUNCTIONAL COVERAGE OF THE DESIGN. IT RECEIVES THE MONITORED SEQUENCE ITEMS AND USES THE COVERGROUP TO SAMPLE DATA AND EVALUATE DESIGN FUNCTIONALITY COVERAGE.

2. VERIFICATION PLAN

A	B	C	D	E	F	G
Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check		
WRAPPER_1	When the reset is asserted, the output (MISO) should be zero	Directed at the start of the simulation then randomized with constraints to be off most of the time	-	The outputs are checked using both golden model and assertions		
WRAPPER_2	In the write only sequence, when the SS_n is low and the value of MOSI in the following 3 cycles is 000, then the slave sends the incoming values of the MOSI to the ram then store it as write address	Randomization with constraints on SS_n to be one every 13 cycle if the internal state of the slave is not READ_DATA and to be one every 23 cycles if the internal state of the slave is READ_DATA, and constraints on MOSI value to be 000 or 001 in the next first 3 cycles after SS_n equal to zero	-	The outputs are checked using both golden model and assertions		
WRAPPER_3	In the write only sequence, when the SS_n is low and the value of MOSI in the following 3 cycles is 001, then the slave sends the incoming values of the MOSI to the ram then stores them in the memory using the write address	Randomization with constraints on SS_n to be one every 13 cycle if the internal state of the slave is not READ_DATA and to be one every 23 cycles if the internal state of the slave is READ_DATA, and constraints on MOSI value to be 000 or 001 in the next first 3 cycles after SS_n equal to zero	-	The outputs are checked using both golden model and assertions		
WRAPPER_4	In the read only sequence, when the SS_n is low and the value of MOSI in the following 3 cycles is 110, then the slave sends the incoming values of the MOSI to the ram then store it as read address	Randomization with constraints on SS_n to be one every 13 cycle if the internal state of the slave is not READ_DATA and to be one every 23 cycles if the internal state of the slave is READ_DATA, and constraints on MOSI in the next first 3 cycles after SS_n equal to zero that if its previous values was 110 then the next values will be 111 and if its previous values was 111 then the next values will be 110, if none then it will be 110	-	The outputs are checked using both golden model and assertions		
WRAPPER_5	In the read only sequence, when the SS_n is low and the value of MOSI in the following 3 cycles is 110, then the slave sends the incoming values of the MOSI to the ram then sends the data of the memory at read address to the slave then the slave output this data bit by bit on the MOSO	Randomization with constraints on SS_n to be one every 13 cycle if the internal state of the slave is not READ_DATA and to be one every 23 cycles if the internal state of the slave is READ_DATA, and constraints on MOSI in the next first 3 cycles after SS_n equal to zero that if its previous values was 110 then the next values will be 111 and if its previous values was 111 then the next values will be 110, if none then it will be 111	-	The outputs are checked using both golden model and assertions		
WRAPPER_6	In the write read sequence, when the SS_n is low and the value of MOSI in the following 3 cycles is 000, then the slave sends the incoming values of the MOSI to the ram then store it as write address	Randomization with constraints on SS_n to be one every 13 cycle if the internal state of the slave is not READ_DATA and to be one every 23 cycles if the internal state of the slave is READ_DATA, and constraints on MOSI in the next first 3 cycles after SS_n equal to zero that if its previous values was 000 then the next values will be 001 or 000 and if its previous	-	The outputs are checked using both golden model and assertions		

A	In the write read sequence, when the SS _n is low and the value of MOSI in the following 3 cycles is 000, then the slave sends the incoming values of the MOSI to the ram then store it as write address	B	Randomization with constraints on SS _n to be one every 13 cycle if the internal state of the slave is not READ_DATA and to be one every 23 cycles if the internal state of the slave is READ_DATA, and constraints on MOSI in the next first 3 cycles after SS _n equal to zero that if its previous values was 000 then the next values will be 001 or 000 and if its previous values was 001 then the next values will be 000 110 or 40% 000, if its previous values was 110 then the next values will be 111 and if its previous values was 111 then the next values will be 000 000 or 40% 110,	C	The outputs are checked using both golden model and assertions	D		E		F		G		H		I	
WRAPPER_6																	
7																	
WRAPPER_7																	
8																	
WRAPPER_8																	
9																	
WRAPPER_9																	
10																	

3. CODE SNIPPETS

INTERFACE

```
WRAPPER_if.sv
1  interface WRAPPER_IF(clk);
2    input  clk;
3    //Module Signals
4    logic  MOSI, SS_n, rst_n;
5    logic  MISO;
6    logic  MISO_ref;
7  endinterface
```

DESIGN

```
SPI_wrapper.v
1  module WRAPPER (MOSI,MISO,SS_n,clk,rst_n);
2
3  Input  MOSI, SS_n, clk, rst_n;
4  output MISO;
5
6  wire [9:0] rx_data_din;
7  wire      rx_valid;
8  wire      tx_valid;
9  wire [7:0] tx_data_dout;
10
11 RAM  RAM_Instance (
12   .din(rx_data_din),
13   .clk(clk),
14   .rst_n(rst_n),
15   .rx_valid(rx_valid),
16   .dout(tx_data_dout),
17   .tx_valid(tx_valid)
18 );
19 SLAVE SLAVE_Instance (
20   .MOSI(MOSI),
21   .MISO(MISO),
22   .SS_n(SS_n),
23   .clk(clk),
24   .rst_n(rst_n),
25   .rx_data(rx_data_din),
26   .rx_valid(rx_valid),
27   .tx_data(tx_data_dout),
28   .tx_valid(tx_valid)
29 );
30
31 endmodule
```

GOLEN Model

```
# SP_wrapper.sv
1 module SP_WRAPPER_ref (MOSI,MISO,SS_n,clk,rst_n);
2
3   Input  MOSI,SS_n,clk,rst_n;
4   output MISO;
5
6   wire [9:0] rx_data_din;
7   wire      rx_valid;
8   wire      tx_valid;
9   wire [7:0] tx_data_dout;
10
11  RAM_ref RAM_Instance (
12     .clk(clk),
13     .rst_n(rst_n),
14     .rx_data(rx_data_din),
15     .tx_data(tx_data_dout),
16     .tx_valid(tx_valid),
17     .tx_data(tx_data_dout)
18 );
19
20  SLAVE golden_model SLAVE_Instance (
21     .clk(clk),
22     .rst_n(rst_n),
23     .SS_n(SS_n),
24     .MOSI(MOSI),
25     .tx_valid(tx_valid),
26     .tx_data(tx_data_dout),
27     .rx_valid(rx_valid),
28     .rx_data(rx_data_din)
29 );
30
31 endmodule
```

SVR

```
# WRAPPER.sv
1 module WRAPPER_sva(MOSI,MISO,SS_n,clk,rst_n_rx_data_din,rx_valid,tx_valid,operation);
2
3   input MOSI;
4   input [9:0] rx_data_din;
5   input      rx_valid;
6   input      tx_valid;
7   input [8:0] SS_n;
8   input      operation;
9
10  property rst_wrapper_p;
11    @posedge clk |> (rst_n == 1'b0 & (rx_data_din == 10'h00_0000_0000) & !rx_valid);
12  endproperty
13
14  property MISO_wrapper_p;
15    @posedge clk disable iff(!rst_n) (operation != 3'b100) |> (MISO == $past(MISO));
16  endproperty
17
18  assert property (rst_wrapper_p);
19  assert property (MISO_wrapper_p);
20
21  cover property (rst_wrapper_p);
22  cover property (MISO_wrapper_p);
23 endmodule
```

CONFIG

```
# WRAPPER_config.sv
1 package WRAPPER_config_pkg;
2   import uvm_pkg::*;
3   import WRAPPER_macros.svh;
4   class WRAPPER_config extends uvm_object;
5     `uvm_object_new(WRAPPER_config)
6
7     virtual WRAPPER_IF wrapper_if;
8     uvm_active_passive_enum is_active;
9
10    function new(string name = "WRAPPER_config");
11      super.new(name);
12    endfunction
13    endclass //noknows_config extends uvm_object
14 endpackage
```

TOP

```
# WRAPPER_top.sv
1 import uvm_pkg::*;
2 `include "uvm_macros.svh"
3 import WRAPPER_macros.svh;
4 module WRAPPER_top;
5   logic CLK = 0;
6   initial begin
7     $readmemh("mem.dat", DUT.WRAPPER.RAM_Instance.MEM);
8     $readmemh("mem.dat", GM.WRAPPER.RAM_Instance.MEM);
9
10    forever #1 CLK = ~CLK;
11  end
12
13  WRAPPER_IF wrapper_IF(CLK);
14  RAM_IF ram_IF(CLK);
15  SPI_SLAVE_IF slave_IF(CLK);
16
17  WRAPPER_DUT_WRAPPER (
18    .clk(wrapper_IF.clk),
19    .rst_n(wrapper_IF.rst_n),
20    .MOSI(wrapper_IF.MOSI),
21    .SS_n(wrapper_IF.SS_n),
22    .MISO(wrapper_IF.MISO)
23  );
24
25  WRAPPER_ref GM_WRAPPER(
26    .clk(wrapper_IF.clk),
27    .rst_n(wrapper_IF.rst_n),
28    .MOSI(wrapper_IF.MOSI),
29    .SS_n(wrapper_IF.SS_n),
30    .MISO(wrapper_IF.MISO)
31  );
32
33 //RAM Inputs
34 assign ram_IF.rst_n = DUT.WRAPPER.rst_n;
35 assign ram_IF.rx_valid = DUT.WRAPPER.rx_valid;
36 assign ram_IF.din = DUT.WRAPPER.rx_data_din;
37 assign ram_IF.dout = DUT.WRAPPER.tx_data_dout;
38 assign ram_IF.tx_valid = DUT.WRAPPER.tx_valid;
39 assign ram_IF.dout_ref = GM.WRAPPER.tx_data_dout;
40 assign ram_IF.tx_valid_ref = GM.WRAPPER.tx_valid;
41
42 //Slave Inputs
43 assign slave_IF.rst_n = DUT.WRAPPER.rst_n;
44 assign slave_IF.SS_n = DUT.WRAPPER.SS_n;
45 assign slave_IF.tx_valid = DUT.WRAPPER.tx_valid;
46 assign slave_IF.data = DUT.WRAPPER.tx_data_dout;
47 assign slave_IF.MOSI = DUT.WRAPPER.MOSI;
48 assign slave_IF.rx_din = DUT.WRAPPER.rx_data_din;
49 assign slave_IF.dout = GM.WRAPPER.tx_data_din;
50 assign slave_IF.MISO_ref = DUT.WRAPPER.MISO;
51 assign slave_IF.tx_valid_ref = GM.WRAPPER.tx_valid;
52 assign slave_IF.rx_valid_ref = GM.WRAPPER.rx_valid;
53
54 //Slave Inputs
55 assign slave_IF.rst_n = DUT.WRAPPER.rst_n;
56 assign slave_IF.SS_n = DUT.WRAPPER.SS_n;
57 assign slave_IF.tx_valid = DUT.WRAPPER.tx_valid;
58 assign slave_IF.data = DUT.WRAPPER.tx_data_dout;
59 assign slave_IF.MOSI = DUT.WRAPPER.MOSI;
60 assign slave_IF.MISO_ref = DUT.WRAPPER.MISO;
61
62 //RAM Outputs
63 assign ram_IF.dout = DUT.WRAPPER.tx_data_dout;
64 assign ram_IF.tx_valid = DUT.WRAPPER.tx_valid;
65
66 bind WRAPPER WRAPPER_sva WRAPPER_sva_inst(
67   .clk(wrapper_IF.clk),
68   .rst_n(wrapper_IF.rst_n),
69   .MOSI(wrapper_IF.MOSI),
70   .SS_n(wrapper_IF.SS_n),
71   .MISO(wrapper_IF.MISO),
72   .rx_data_din(DUT.WRAPPER.rx_data_din),
73   .rx_valid(DUT.WRAPPER.rx_valid),
74   .tx_valid(DUT.WRAPPER.tx_valid),
75   .operation(GM.WRAPPER.instance.cs)
76 );
77
78 initial begin
79   uvm_config_db #(virtual WRAPPER_IF)::set(null,"uvm_test_top","WRAPPER_IF",wrapper_IF);
80   uvm_config_db #(virtual SPI_SLAVE_IF)::set(null,"uvm_test_top","SLAVE_IF",slave_IF);
81   uvm_config_db #(virtual RAM_IF)::set(null,"uvm_test_top","RAM_IF",ram_IF);
82   run_test("WRAPPER_test");
83 end
84
85 endmodule
```

In 71, Col 32 Spacing: 4 UTT-B CRLF ⓘ System Verilog ⓘ

TEST

```

# WRAPPER_test.sv
1 package WRAPPER_Env_pk;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import shared_pkg::*;
5   import WRAPPER_config_pk::*;
6   import WRAPPER_scoreboard_pk::*;
7   import WRAPPER_sequ_item_pk::*;
8   import WRAPPER_env_pk::*;
9
10  import SPI_slave_env_pk::*;
11  import SPI_slave_config_pk::*;
12
13  import RAM_env_pk::*;
14  import RAM_config_pk::*;
15
16  class WRAPPER_test extends uvm_test;
17    `uvm_component_utils(WRAPPER_test)
18
19    //ENV
20    WRAPPER_env wrapper_env;
21    Ram_env ram_env;
22    SPI_slave_env slave_env;
23
24    //Config DB
25    uvm_config_db #(sequ_item_pk*)::get(this,"$uvm_top.WRAPPER_IF", "ram_cfg");
26    ram_cfg = RAM_env::type_id::create("ram_cfg", this);
27    ram_cfg = SPI_SLAVE_CONFIG::type_id::create("slave_cfg", this);
28
29    //Sequences
30    WRAPPER_reset_seq wrapper_reset;
31    WRAPPER_write_only_seq wrapper_write_only;
32    WRAPPER_read_only_seq wrapper_read_only;
33    WRAPPER_write_read_seq wrapper_write_read;
34
35    function new (string name = "WRAPPER_test", uvm_component parent = null);
36      super.new(name, parent);
37    endfunction
38
39
40    function void build_phase(uvm_phase phase);
41      super.build_phase(phase);
42      //ENV Creation
43      wrapper_env = WRAPPER_env::type_id::create("wrapper_env", this);
44      slave_env = SPI_SLAVE_ENV::type_id::create("slave_env", this);
45      ram_env = RAM_env::type_id::create("ram_env", this);
46
47    //Config DB Creation
48    wrapper_cfg = WRAPPER_CONFIG::type_id::create("WRAPPER_CONFIG", this);
49    slave_cfg = SPI_SLAVE_CONFIG::type_id::create("slave_cfg", this);
50    ram_cfg = RAM_CONFIG::type_id::create("ram_cfg", this);
51
52    //Sequences Creation
53    wrapper_reset = WRAPPER_RESET_SEQ::type_id::create("wrapper_reset");
54    wrapper_write_only = WRAPPER_WRITE_ONLY_SEQ::type_id::create("wrapper_write_only");
55    wrapper_read_only = WRAPPER_READ_ONLY_SEQ::type_id::create("wrapper_read_only");
56    wrapper_write_read = WRAPPER_WRITE_READ_SEQ::type_id::create("wrapper_write_read");
57
58    wrapper_cfg.is_active = UVM_ACTIVE;
59    if (uvm_config_db #(virtual WRAPPER_IF)::get(this,"$WRAPPER_IF", wrapper_cfg.wrapper_if))
60      `uvm_fatal("build_phase", "Error - test cannot retrieve the WRAPPER interface");
61    else
62      uvm_config_db #(WRAPPER_CONFIG)::set(this, "", "CFG_WRAPPER", wrapper_cfg);
63
63    slave_cfg.is_active = UVM_PASSIVE;
64    if (uvm_config_db #(virtual RAM_IF)::get(this,"$RAM_IF", ram_cfg.ram_if))
65      `uvm_fatal("build_phase", "Error - test cannot retrieve the RAM interface");
66    else
67      uvm_config_db #(RAM_CONFIG)::set(this, "", "CFG_RAM", ram_cfg);
68
69    ram_cfg.is_active = UVM_PASSIVE;
70    if (uvm_config_db #(virtual RAM_IF)::get(this,"$RAM_IF", ram_cfg.ram_if))
71      `uvm_fatal("build_phase", "Error - test cannot retrieve the RAM interface");
72
73    endfunction
74
75    task run_phase(uvm_phase phase);
76      super.run_phase();
77      phase.raise_objection(this);
78
79      `uvm_info("run_phase", "Reset Asserted", UVM_LOW);
80      wrapper_reset.start(wrapper_env.wrapper_agt.wrapper_seq);
81      `uvm_info("run_phase", "Reset Deasserted", UVM_LOW);
82
83      `uvm_info("run_phase", "Write Only Seq Started", UVM_LOW);
84      wrapper_write_only.start(wrapper_env.wrapper_agt.wrapper_seq);
85      `uvm_info("run_phase", "Write Only Seq Ended", UVM_LOW);
86
87      `uvm_info("run_phase", "Reset Asserted", UVM_LOW);
88      wrapper_reset.start(wrapper_env.wrapper_agt.wrapper_seq);
89      `uvm_info("run_phase", "Reset Deasserted", UVM_LOW);
90
91      `uvm_info("run_phase", "Read Only Seq Started", UVM_LOW);
92      is_readonly = 1;
93      wrapper_read_only.start(wrapper_env.wrapper_agt.wrapper_seq);
94      is_readonly = 0;
95      `uvm_info("run_phase", "Read Only Seq Ended", UVM_LOW);
96
97      `uvm_info("run_phase", "Write Read Seq Started", UVM_LOW);
98      wrapper_write_read.start(wrapper_env.wrapper_agt.wrapper_seq);
99      `uvm_info("run_phase", "Write Read Seq Ended", UVM_LOW);
100
100    phase.drop_objection(this);
101  endtask
102
103 endclass
104
105 endpackage

```

ENVIRONMENT

```

# WRAPPER_env_pk
1 package WRAPPER_env_pk;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import WRAPPER_agent_pk::*;
5   import WRAPPER_scoreboard_pk::*;
6   import WRAPPER_coverage_pk::*;
7   class WRAPPER_env extends uvm_env;
7.1   `uvm_component_utils(WRAPPER_env)
8
9   WRAPPER_agent wrapper_agt;
10  WRAPPER_scoreboard wrapper_sb;
11  WRAPPER_coverage wrapper_cvg;
12
13  function new (string name = "WRAPPER_env", uvm_component parent = null);
14    super.new(name, parent);
15  endfunction
16
17
18  function void build_phase(uvm_phase phase);
19    super.build_phase(phase);
20    wrapper_agt = WRAPPER_AGENT::type_id::create("wrapper_agt", this);
21    wrapper_sb = WRAPPER_SCOREBOARD::type_id::create("wrapper_sb", this);
22    wrapper_cvg = WRAPPER_COVERAGE::type_id::create("wrapper_cvg", this);
23  endfunction
24
25  function void connect_phase(uvm_phase phase);
26    super.connect_phase(phase);
27    wrapper_agt.agt_ap.connect(wrapper_sb.sb_axp);
28    wrapper_agt.agt_ap.connect(wrapper_cvg.cvg_axp);
29  endfunction
30
31 endclass
32
33 endpackage

```

COVERAGE

```
8 WRAPPER_coverage.sv
 1 package WRAPPER_coverage_pkg;
 2   import uvm_pkg::*;
 3   `include UVM_MACROS.svh
 4
 5   import WRAPPER_seq_item_pkg::*;
 6   import shared_pkg::*;
 7   class WRAPPER_coverage extends uvm_component;
 8     `uvm_component_utils(WRAPPER_coverage)
 9
10    uvm_analysis_export #(WRAPPER_seq_item) cvg_axp;
11    uvm_tlm_analysis_fifo #(WRAPPER_seq_item) cvg_fifo;
12
13    WRAPPER_seq_item cvg_seq_item;
14
15    //Cover Groups
16    covergroup WRAPPER_CVG;
17      SS_n_cp: coverpoint cvg_seq_item.SS_n {
18        bins normal_trans = {1 => {[13] => 1}};
19        bins extended_trans = {1 => {*[23]} => 1};
20      }
21
22      MOSI_cp: coverpoint cvg_seq_item.MOSI{
23        bins write_address = {0 => 0 => 0};
24        bins write_data   = {0 => 0 => 1};
25        bins read_address = {1 => 1 => 0};
26        bins read_data   = {1 => 1 => 1};
27      }
28
29      SS_n_MOSI_cr: cross SS_n_cp, MOSI_cp{
30        ignore bins not_r_data_ext4 = binsof(MOSI_cp.write_address) && binsof(SS_n_cp.extended_trans);
31        ignore bins not_r_data_ext3 = binsof(MOSI_cp.write_data) && binsof(SS_n_cp.extended_trans);
32        ignore bins not_r_data_ext3 = binsof(MOSI_cp.read_address) && binsof(SS_n_cp.extended_trans);
33        ignore bins not_r_data_ext4 = binsof(MOSI_cp.read_data) && binsof(SS_n_cp.normal_trans);
34      }
35
36    endgroup
37    function new (string name = "WRAPPER_coverage", uvm_component parent = null);
38      super.new(name, parent);
39      WRAPPER_CVG = new();
40    endfunction
41
42    function void build_phase (uvm_phase phase);
43      super.build_phase(phase);
44
45      cvg_axp = new ("cvg_axp", this);
46      cvg_fifo = new ("cvg_fifo", this);
47    endfunction
48
49    function void connect_phase(uvm_phase phase);
50      super.connect_phase(phase);
51      cvg_axp.connect(cvg_fifo.analysis_export);
52    endfunction
53
54    task run_phase(uvm_phase phase);
55      super.run_phase(phase);
56      forever begin
57        cvg_fifo.get(cvg_seq_item);
58        WRAPPER_CVG.sample();
59      end
60    endtask
61
62  endclass
63
64 endpackage
```

Ln 17, Col 13 Spaces: 4 UTF-8 CRLF () System Verilog

SCOREBOARD

```
8 WRAPPER_scoreboard.sv
 1 package WRAPPER_scoreboard_pkg;
 2   import uvm_pkg::*;
 3   `include UVM_MACROS.svh
 4   import WRAPPER_seq_item_pkg::*;
 5   class WRAPPER_scoreboard extends uvm_scoreboard;
 6     `uvm_component_utils(WRAPPER_scoreboard)
 7
 8
 9     uvm_analysis_export #(WRAPPER_seq_item) sb_axp;
10     uvm_tlm_analysis_fifo #(WRAPPER_seq_item) sb_fifo;
11
12     WRAPPER_seq_item sb_seq_item;
13
14     int error_cnt = 0, correct_cnt = 0;
15
16     function new (string name = "WRAPPER_scoreboard", uvm_component parent = null);
17       super.new(name, parent);
18     endfunction
19
20     function void build_phase(uvm_phase phase);
21       super.build_phase(phase);
22       sb_axp = new("sb_axp", this);
23       sb_fifo = new("sb_fifo", this);
24     endfunction
25
26     function void connect_phase(uvm_phase phase);
27       super.connect_phase(phase);
28       sb_axp.connect(sb_fifo.analysis_export);
29     endfunction
30
31     task run_phase(uvm_phase phase);
32       super.run_phase(phase);
33       forever begin
34         sb_fifo.get(sb_seq_item);
35         if (
36           sb_seq_item.MISO != sb_seq_item.MISO_ref
37         ) begin
38           error_cnt++;
39           `uvm_error("run_phase", $format("Comparison Failed - Transaction Received DUT : %s, Ref Model : MISO = %b", sb_seq_item.convert2string, sb_seq_item.MISO_ref));
40         end
41         else begin
42           correct_cnt++;
43         end
44       end
45     endtask
46
47     function void report_phase (uvm_phase phase);
48       super.report_phase(phase);
49       `uvm_info("report_phase", $format("Correct Tests = %d, Error Tests = %d", correct_cnt, error_cnt),UVM_MEDIUM)
50     endfunction
51
52  endclass
53
54 endpackage
```

Ln 52, Col 20 Spaces: 4 UTF-8 CRLF () System Verilog

AGENT

```
WRAPPER_agent.sv
1 package WRAPPER_agent_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import WRAPPER_sequencer_pkg::*;
5   import WRAPPER_driver_pkg::*;
6   import WRAPPER_driver_if::*;
7   import WRAPPER_seq_item_pkg::*;
8   import WRAPPER_config_pkg::*;
9
10  class WRAPPER_agent extends uvm_agent;
11    `uvm_component_utils(WRAPPER_agent)
12
13    WRAPPER_config wrapper_cfg;
14
15    WRAPPER_sequencer wrapper_sqn;
16    WRAPPER_monitor wrapper_mon;
17    WRAPPER_driver wrapper_dvr;
18
19    uvm_analysis_port #(WRAPPER_seq_item) agt_ap;
20
21    function new (string name = "WRAPPER_agent", uvm_component parent = null);
22      super.new(name, parent);
23    endfunction
24
25    function void build_phase(uvm_phase phase);
26      super.build_phase(phase);
27      if (!uvm_config_db #(WRAPPER_config)::get(this,"","CFG_WRAPPER",wrapper_cfg))
28        `uvm_fatal("build_phase", "Error - Wrapper Agent cannot retrieve the config object");
29
30      wrapper_mon = WRAPPER_monitor::type_id::create("wrapper_mon", this);
31
32      if (wrapper_cfg.is_active == UVM_ACTIVE) begin
33        wrapper_dvr = WRAPPER_driver::type_id::create("wrapper_dvr", this);
34        wrapper_sqn = WRAPPER_sequencer::type_id::create("wrapper_sqn", this);
35      end
36
37      agt_ap = new ("agt_ap", this);
38    endfunction
39
40    function void connect_phase (uvm_phase phase);
41      super.connect_phase(phase);
42      if(wrapper_cfg.is_active == UVM_ACTIVE) begin
43        wrapper_dvr.wrapper_dvr_if = wrapper_cfg.wrapper_if;
44        wrapper_dvr.seq_item_port.connect(wrapper_sqn.seq_item_export);
45      end
46      wrapper_mon.wrapper_mon_if = wrapper_cfg.wrapper_if;
47      wrapper_mon.mon_ap.connect(agt_ap);
48    endfunction
49  endclass
endpackage
```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49

In 38, Col 1 | Spaces:4 | UTM-8 | CR/LF | System Verilog |

DRIVER

```
WRAPPER_driver.sv
1 package WRAPPER_driver_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import WRAPPER_seq_item_pkg::*;
5   class WRAPPER_driver extends uvm_driver #(WRAPPER_seq_item);
6     `uvm_component_utils(WRAPPER_driver)
7
8     WRAPPER_seq_item dvr_seq_item;
9     virtual WRAPPER_IF wrapper_dvr_if;
10
11    function new (string name = "WRAPPER_driver", 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        dvr_seq_item = WRAPPER_seq_item::type_id::create("dvr_seq_item");
19        seq_item_port.get_next_item(dvr_seq_item);
20        //print("Driver: %s", dvr_seq_item.signed_value);
21        wrapper_dvr_if.rst_n = dvr_seq_item.rst_n;
22        wrapper_dvr_if.MOST = dvr_seq_item.MOSI;
23        wrapper_dvr_if.SS_n = dvr_seq_item.SSI;
24        repeat(1)@{negedge wrapper_dvr_if.clk};
25        seq_item_port.item_done();
26        `uvm_info("run_phase", dvr_seq_item.convert2string_stimulus, UVM_HIGH);
27      end
28    endtask
29  endclass
30 endpackage
```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

MONITOR

```
8 WRAPPER_monitor.sv
 1 package WRAPPER_monitor_pkg;
 2   import uvm_pkg::*;
 3   `include "uvm_macros.svh"
 4   import WRAPPER_seq_item_pkg::*;
 5   class WRAPPER_monitor extends uvm_monitor;
 6     `uvm_component_utils(WRAPPER_monitor)
 7
 8     virtual WRAPPER_IF wrapper_mon_if;
 9     WRAPPER_seq_item mon_seq_item;
10
11     uvm_analysis_port #(WRAPPER_seq_item) mon_ap;
12
13     function new (string name = "WRAPPER_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@();
25         mon_seq_item = WRAPPER_seq_item::type_id::create("mon_seq_item");
26         //Monitor the signals
27         repeat(1)[@begin@];
28         mon_seq_item.rst_n = wrapper_mon_if.rst_n;
29         mon_seq_item.MOSI = wrapper_mon_if.MOSI;
30         mon_seq_item.SS_n = wrapper_mon_if.SS_n;
31         mon_seq_item.MISO = wrapper_mon_if.MISO;
32         mon_seq_item.MISO_ref = wrapper_mon_if.MISO_ref;
33         mon_ap.write(mon_seq_item);
34       | `uvm_info("run_phase", mon_seq_item.convert2string, UVM_HIGH);
35     end
36   endtask
37
38   endclass
39
40 endpackage
```

SEQUENCER

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

SEQUENCE

```
8 WRAPPER_sequence.sv
 1 package WRAPPER_sequence_pkg;
 2   import uvm_pkg::*;
 3   `include "uvm_macros.svh"
 4   import WRAPPER_seq_item_pkg::*;
 5   class WRAPPER_reset_seq extends uvm_sequence #(WRAPPER_seq_item);
 6     `uvm_object_utils(WRAPPER_reset_seq);
 7
 8     WRAPPER_seq_item wrapper_seq_item;
 9
10     function new (string name = "WRAPPER_reset_seq");
11       | super.new(name);
12     endfunction
13
14     virtual task body();
15       wrapper_seq_item = WRAPPER_seq_item::type_id::create("wrapper_seq_item");
16       start_item(wrapper_seq_item);
17       wrapper_seq_item.rst_n = 0;
18       wrapper_seq_item.MOSI = 0;
19       wrapper_seq_item.SS_n = 0;
20       finish_item(wrapper_seq_item);
21     endtask
22   endclass
23
24   class WRAPPER_write_only_seq extends uvm_sequence #(WRAPPER_seq_item);
25     `uvm_object_utils(WRAPPER_write_only_seq)
26
27     WRAPPER_seq_item wrapper_seq_item;
28
29     function new (string name = "WRAPPER_write_only_seq");
30       | super.new(name);
31     endfunction
32
33     virtual task body();
34       wrapper_seq_item = WRAPPER_seq_item::type_id::create("wrapper_seq_item");
35       wrapper_seq_item.read_only_const.constraint_mode(0);
36       wrapper_seq_item.write_read_const.constraint_mode(0);
37       repeat (1000) begin
38         start_item(wrapper_seq_item);
39         assert(wrapper_seq_item.randomize);
40         finish_item(wrapper_seq_item);
41       end
42     endtask
43   endclass
44
45   class WRAPPER_read_only_seq extends uvm_sequence #(WRAPPER_seq_item);
46     `uvm_object_utils(WRAPPER_read_only_seq)
47
48     WRAPPER_seq_item wrapper_seq_item;
49
50     function new (string name = "WRAPPER_read_only_seq");
51       | super.new(name);
52     endfunction
53
54     virtual task body();
55       wrapper_seq_item = WRAPPER_seq_item::type_id::create("wrapper_seq_item");
56       wrapper_seq_item.write_only_constraint_mode(0);
```

```
8 WRAPPER_sequence.sv
 1 package WRAPPER_sequence_pkg;
 2   class WRAPPER_write_only_seq extends uvm_sequence #(WRAPPER_seq_item);
 3     virtual task body();
 4     | end
 5   endtask
 6
 7   class WRAPPER_read_only_seq extends uvm_sequence #(WRAPPER_seq_item);
 8     `uvm_object_utils(WRAPPER_read_only_seq)
 9
10     WRAPPER_seq_item wrapper_seq_item;
11
12     function new (string name = "WRAPPER_read_only_seq");
13       | super.new(name);
14     endfunction
15
16     virtual task body();
17       wrapper_seq_item = WRAPPER_seq_item::type_id::create("wrapper_seq_item");
18       wrapper_seq_item.write_only_const.constraint_mode(0);
19       wrapper_seq_item.read_only_const.constraint_mode(0);
20       wrapper_seq_item.write_read_const.constraint_mode(0);
21       for(int i = 0; i<1000; i++) begin
22         start_item(wrapper_seq_item);
23         if(i == 1) start_read = 1;
24         else start_read = 0;
25         assert(wrapper_seq_item.randomize);
26         finish_item(wrapper_seq_item);
27       end
28     endtask
29   endclass
30
31   class WRAPPER_write_read_seq extends uvm_sequence #(WRAPPER_seq_item);
32     `uvm_object_utils(WRAPPER_write_read_seq)
33
34     WRAPPER_seq_item wrapper_seq_item;
35
36     function new (string name = "WRAPPER_write_read_seq");
37       | super.new(name);
38     endfunction
39
40     virtual task body();
41       wrapper_seq_item = WRAPPER_seq_item::type_id::create("wrapper_seq_item");
42       wrapper_seq_item.write_only_const.constraint_mode(0);
43       wrapper_seq_item.read_only_const.constraint_mode(0);
44       wrapper_seq_item.write_read_const.constraint_mode(1);
45       for(int i = 0; i<1000; i++) begin
46         start_item(wrapper_seq_item);
47         if(i == 1) start_read = 1;
48         else start_read = 0;
49         assert(wrapper_seq_item.randomize);
50         finish_item(wrapper_seq_item);
51       end
52     endtask
53   endclass
54
55 endpackage
```

In 90, Col 11 | Spacing: 4 | UTF-8 | CRLF | { } System Verilog

SEQUENCE ITEM

```

@ WRAPPER_seq_item.sv
1 package WRAPPER_seq_item_pkg;
2   import uvm_pkg::*;
3   $include "uvm-uvm.svh"
4   import shared_pkg::*;
5   class WRAPPER_seq_item extends uvm_sequence_item;
6     `uvm_object_utils(WRAPPER_seq_item)
7   // Signals Declarations
8   rand bit [1:0] state = 2'b00;
9   static bit [1:0] state_old = 2'b00;
10  int count = 0;
11  bit SS_n_HIGH = 1;
12  rand bit [10:0] MOSI_array;
13  static bit rand_ok, read_add_ok, read_type;
14
15  // Functions
16  function new(string name = "WRAPPER_seq_item");
17    super.new(name);
18  endfunction
19
20  function string convert2string();
21    return $sformatf("%s - rst_n = %b , SS_n = %b, MOSI = %b, DUT: MISO = %b", super.convert2string, rst_n, SS_n, MOSI, MISO);
22  endfunction
23
24  function string convert2string_stimulus();
25    return $sformatf("%s - rst_n = %b , SS_n = %b, MOSI = %b", super.convert2string, rst_n, SS_n, MOSI);
26  endfunction
27
28  function void pre_randomize();
29    if (count == 0 || !rst_n) begin
30      MOSI_array.rand_mode(1);
31      state.rand_mode(1);
32      rand_ok = 1;
33    end
34    else begin
35      MOSI_array.rand_mode(0);
36      state.rand_mode(0);
37      rand_ok = 0;
38    end
39  endfunction
40
41  function void post_randomize();
42    begin
43      if (!rst_n || SS_n) begin
44        count = 0;
45        SS_n_HIGH = 0;
46      end
47      else count++;
48
49      if (count > 12 && count < 23) begin
50        if (state != 2'b11) begin
51          SS_n_HIGH = 1;
52        end
53        else SS_n_HIGH = 0;
54      end
55    end
56  endfunction
57
58  constraint reset_const{
59    rst_n dist {1:-99, 0:-1};
60  };
61
62  constraint MOSI_array_const{
63    MOSI_array[10] == state[1];
64    MOSI_array[9:8] == state;
65  };
66
67  constraint SS_n_const{
68    SS_n == SS_n_HIGH;
69  };
70
71  constraint write_only_const{
72    state inside {2'b00, 2'b01};
73  };
74
75  constraint read_only_const{
76    if (start_read || read_add_ok || (state.old == 2'b11 && rand_ok)) { // Read data
77      state == 2'b10;
78    }
79    else if (state.old == 2'b00 & rand_ok) { //read Addr
80      state == 2'b11;
81    }
82  };
83
84  constraint write_read_const{
85    if (start_read || read_add_ok || (state.old == 2'b00 & rand_ok)) {
86      state inside {2'b00, 2'b01};
87    }
88    else if (state.old == 2'b01 & rand_ok) {
89      state dist {2'b10:/60, 2'b00:/40};
90    }
91    else if (state.old == 2'b11 & rand_ok) {
92      state dist {2'b00:/60, 2'b10:/40};
93    }
94    else if (state.old == 2'b10 & rand_ok) {
95      state inside {2'b11};
96    }
97  };
98
99  endclass
100 endpackage

```

In 64 Col 17 Spaces:4 UTF-8 CRLF () System Verilog

```

@ WRAPPER_seq_item.sv
1 package WRAPPER_seq_item_pkg;
2   // Functions
3   function void pre_randomize();
4   endfunction
5   function void post_randomize();
6   begin
7     if (!rst_n || SS_n) begin
8       count = 0;
9       SS_n_HIGH = 0;
10    end
11    else count++;
12
13    if (count > 12 && count < 23) begin
14      if (state != 2'b11) begin
15        SS_n_HIGH = 1;
16      end
17      else SS_n_HIGH = 0;
18    end
19    else if (count == 23) begin
20      if (state == 2'b11) begin
21        SS_n_HIGH = 1;
22      end
23      else SS_n HIGH = 0;
24    end
25  end
26
27  state.old = state;
28
29  if(state.old == 2'b01) read_type = 0;
30  else if(state.old == 2'b11) read_type = 1;
31  if(rst_n && (state.old[1] == 1)) read_add_ok = 1;
32  else read_add_ok = 0;
33
34  endfunction
35
36  constraint reset_const{
37    rst_n dist {1:-99, 0:-1};
38  };
39
40  constraint MOSI_array_const{
41    MOSI_array[10] == state[1];
42    MOSI_array[9:8] == state;
43  };
44
45  constraint SS_n_const{
46    SS_n == SS_n_HIGH;
47  };
48
49  constraint write_only_const{
50    state inside {2'b00, 2'b01};
51  };
52
53  constraint read_only_const{
54    if (start_read || read_add_ok || (state.old == 2'b11 && rand_ok)) { // Read data
55      state == 2'b10;
56    }
57    else if (state.old == 2'b00 & rand_ok) { //read Addr
58      state == 2'b11;
59    }
60  };
61
62  constraint write_read_const{
63    if (start_read || read_add_ok || (state.old == 2'b00 & rand_ok)) {
64      state inside {2'b00, 2'b01};
65    }
66    else if (state.old == 2'b01 & rand_ok) {
67      state dist {2'b10:/60, 2'b00:/40};
68    }
69    else if (state.old == 2'b11 & rand_ok) {
70      state dist {2'b00:/60, 2'b10:/40};
71    }
72    else if (state.old == 2'b10 & rand_ok) {
73      state inside {2'b11};
74    }
75  };
76
77  endclass
78
79 endpackage

```

In 64 Col 17 Spaces:4 UTF-8 CRLF () System Verilog

```

@ WRAPPER_seq_item.sv
1 package WRAPPER_seq_item_pkg;
2   // Functions
3   function void post_randomize();
4   begin
5     if(state.old == 2'b01) read_type = 0;
6     else if(state.old == 2'b11) read_type = 1;
7     if(!rst_n && (state.old[1] == 1)) read_add_ok = 1;
8     else read_add_ok = 0;
9   end
10
11  endfunction
12
13
14  constraint reset_const{
15    rst_n dist {1:-99, 0:-1};
16  };
17
18  constraint MOSI_array_const{
19    MOSI_array[10] == state[1];
20    MOSI_array[9:8] == state;
21  };
22
23  constraint SS_n_const{
24    SS_n == SS_n_HIGH;
25  };
26
27  constraint write_only_const{
28    state inside {2'b00, 2'b01};
29  };
30
31  constraint read_only_const{
32    if (start_read || read_add_ok || (state.old == 2'b11 && rand_ok)) { // Read data
33      state == 2'b10;
34    }
35    else if (state.old == 2'b00 & rand_ok) { //read Addr
36      state == 2'b11;
37    }
38  };
39
40  constraint write_read_const{
41    if (start_read || read_add_ok || (state.old == 2'b00 & rand_ok)) {
42      state inside {2'b00, 2'b01};
43    }
44    else if (state.old == 2'b01 & rand_ok) {
45      state dist {2'b10:/60, 2'b00:/40};
46    }
47    else if (state.old == 2'b11 & rand_ok) {
48      state dist {2'b00:/60, 2'b10:/40};
49    }
50    else if (state.old == 2'b10 & rand_ok) {
51      state inside {2'b11};
52    }
53  };
54
55  endclass
56
57 endpackage

```

In 64 Col 17 Spaces:4 UTF-8 CRLF () System Verilog

Do File

```
-----  
1 vlib work  
2 vlog +define+SIM -f src_files.list +cover -covercells  
3 vsim +voptargs+=acc work.WRAPPER_top -classdebug -vcontrol=all -cover  
4 add wave -position insertpoint sim:/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/*  
5 add wave -position insertpoint sim:/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/din  
6 sim:/WRAPPER_top/DUT_WRAPPER/RAM_Instance/din  
7 sim:/WRAPPER_top/DUT_WRAPPER/RAM_Instance/dout  
8 sim:/WRAPPER_top/DUT_WRAPPER/RAM_Instance/MEM  
9 sim:/WRAPPER_top/DUT_WRAPPER/RAM_Instance/Rd_Addr  
10 sim:/WRAPPER_top/DUT_WRAPPER/RAM_Instance/Wr_Addr  
11 coverage save wrapper.ucdb -onexit  
12 run -all
```

4. COVERAGE

CODE COVERAGE TOGGLE

```
=====  
Toggle Coverage:  
Enabled Coverage          Bins    Hits    Misses Coverage  
-----  
Toggles                  12      12      0    100.00%  
=====  
=====Toggle Details=====  
Toggle Coverage for instance /WRAPPER_top/wrapper_if --  
Node          1H->0L    0L->1H          "Coverage"  
---  
...  
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  
Total Node Count = 6  
Toggled Node Count = 6  
Untoggled Node Count = 0  
Toggle Coverage = 100.00% (12 of 12 bins)
```

FUNCTIONAL COVERAGE

DIRECTIVE COVERAGE:				
Name	Design	Design	Lang	File(Line)
	Unit	UnitType		Hits Status
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_reset_txValid_cvr			RAM_sva	Verilog SVA RAM_sva.sv(36) 29 Covered
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_rxValid_cvr			RAM_sva	Verilog SVA RAM_sva.sv(37) 29 Covered
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_wr_addr_cvr			RAM_sva	Verilog SVA RAM_sva.sv(39) 878 Covered
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_wr_data_cvr			RAM_sva	Verilog SVA RAM_sva.sv(40) 696 Covered
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_rd_addr_cvr			RAM_sva	Verilog SVA RAM_sva.sv(41) 38 Covered
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_rd_data_cvr			RAM_sva	Verilog SVA RAM_sva.sv(43) 9 Covered
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_wr_addr_to_wr_data_cvr			RAM_sva	Verilog SVA RAM_sva.sv(45) 78 Covered
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_rd_addr_to_rd_data_cvr			RAM_sva	Verilog SVA RAM_sva.sv(46) 72 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_received_address_p1			SLAVE	Verilog SVA SPI_slave.sv(253) 65 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_rx_valid_p1			SLAVE	Verilog SVA SPI_slave.sv(252) 73 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_counter_write_p			SLAVE	Verilog SVA SPI_slave.sv(251) 2172 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_counter_RST_p			SLAVE	Verilog SVA SPI_slave.sv(250) 195 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_CS_P9			SLAVE	Verilog SVA SPI_slave.sv(249) 730 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_CS_P8			SLAVE	Verilog SVA SPI_slave.sv(248) 427 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_CS_P7			SLAVE	Verilog SVA SPI_slave.sv(247) 1224 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_CS_P6			SLAVE	Verilog SVA SPI_slave.sv(246) 42 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_CS_P4			SLAVE	Verilog SVA SPI_slave.sv(245) 36 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_CS_P3			SLAVE	Verilog SVA SPI_slave.sv(244) 117 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_CS_P2			SLAVE	Verilog SVA SPI_slave.sv(243) 197 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_CS_P1			SLAVE	Verilog SVA SPI_slave.sv(242) 172 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_rx_valid_p			SLAVE	Verilog SVA SPI_slave.sv(241) 131 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_read_seq_p			SLAVE	Verilog SVA SPI_slave.sv(240) ..

Ln 1, Col 1 90% Windows (CRLF) UTF-8

WRAPPER_cv.txt - Notepad

```

File Edit Format View Help
RAM_sva Verilog SVA RAM_sva.sv(41) 305 Covered
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_rd_data_cvr
    RAM_sva Verilog SVA RAM_sva.sv(43) 9 Covered
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_ur_addr_to_wn_data_cvr
    RAM_sva Verilog SVA RAM_sva.sv(45) 78 Covered
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_rd_addr_to_rd_data_cvr
    RAM_sva Verilog SVA RAM_sva.sv(45) 72 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_received_address_p2
    SLAVE Verilog SVA SPI_slave.sv(253) 65 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_received_address_p1
    SLAVE Verilog SVA SPI_slave.sv(252) 73 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_counter_write_p
    SLAVE Verilog SVA SPI_slave.sv(251) 2172 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_counter_rst_p
    SLAVE Verilog SVA SPI_slave.sv(250) 195 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_cs_p9
    SLAVE Verilog SVA SPI_slave.sv(249) 730 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_cs_p8
    SLAVE Verilog SVA SPI_slave.sv(248) 427 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_cs_p7
    SLAVE Verilog SVA SPI_slave.sv(247) 1224 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_cs_p6
    SLAVE Verilog SVA SPI_slave.sv(246) 42 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_cs_p4
    SLAVE Verilog SVA SPI_slave.sv(245) 36 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_cs_p3
    SLAVE Verilog SVA SPI_slave.sv(244) 117 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_cs_p2
    SLAVE Verilog SVA SPI_slave.sv(243) 197 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_cs_p1
    SLAVE Verilog SVA SPI_slave.sv(242) 172 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_rx_valid_p
    SLAVE Verilog SVA SPI_slave.sv(241) 131 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_read_seq_p
    SLAVE Verilog SVA SPI_slave.sv(240) 68 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_write_seq_p
    SLAVE Verilog SVA SPI_slave.sv(239) 101 Covered
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/cover_rst_p
    SLAVE Verilog SVA SPI_slave.sv(238) 29 Covered
/WRAPPER_top/DUT_WRAPPER/WRAPPER_sva_inst/cover_MISO_wrapper_p
    WRAPPER_sva Verilog SVA WRAPPER_sva.sv(21) 2183 Covered
/WRAPPER_top/DUT_WRAPPER/WRAPPER_sva_Inst/cover_rst_wrapper_p
    WRAPPER_sva Verilog SVA WRAPPER_sva.sv(20) 29 Covered

TOTAL DIRECTIVE COVERAGE: 100.00% COVERS: 26

```

Ln 1, Col 1 90% Windows (CRLF) UTF-8

ASSERTION COVERAGE

WRAPPER_cv.txt - Notepad

```

File Edit Format View Help
ASSERTION RESULTS:
-----+-----+-----+-----+
Name      File(Line)        Failure Count    Pass Count
-----+-----+-----+-----+
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_releft_txvalid
    RAM_sva.sv(23)   0       1
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_reset_dout
    RAM_sva.sv(24)   0       1
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_ur_addr
    RAM_sva.sv(25)   0       1
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_ur_data
    RAM_sva.sv(27)   0       1
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_rd_addr
    RAM_sva.sv(28)   0       1
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_rd_data
    RAM_sva.sv(29)   0       1
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_ur_addr_to_wn_data
    RAM_sva.sv(32)   0       1
/WRAPPER_top/DUT_WRAPPER/RAM_Instance/RAM_sva_inst/a_rd_addr_to_rd_data
    RAM_sva.sv(33)   0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_received_address_p2
    SPI_slave.sv(236) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_received_address_p1
    SPI_slave.sv(235) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_counter_write_p
    SPI_slave.sv(234) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_counter_rst_p
    SPI_slave.sv(233) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_cs_p9
    SPI_slave.sv(232) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_cs_p8
    SPI_slave.sv(231) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_cs_p7
    SPI_slave.sv(230) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_cs_p6
    SPI_slave.sv(229) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_cs_p4
    SPI_slave.sv(228) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_cs_p3
    SPI_slave.sv(227) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_cs_p2
    SPI_slave.sv(226) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_cs_p1
    SPI_slave.sv(225) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_rx_valid_p
    SPI_slave.sv(224) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_read_seq_p
    SPI_slave.sv(223) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_write_seq_p
    SPI_slave.sv(222) 0       1
/WRAPPER_top/DUT_WRAPPER/SLAVE_Instance/assert_rst_p
    SPI_slave.sv(221) 0       1
/WRAPPER_top/DUT_WRAPPER/WRAPPER_sva_inst/assert_MISO_wrapper_p
    WRAPPER_sva.sv(18) 0       1
/WRAPPER_top/DUT_WRAPPER/WRAPPER_sva_Inst/assert_rst_wrapper_p
    WRAPPER_sva.sv(17) 0       1
/WRAPPER_sequence_pck/WRAPPER_write_read_seq/body/#anonblk#73704343#81#immmed_39
    WRAPPER_sequence.sv(39) 0       1
/WRAPPER_sequence_pck/WRAPPER_read_only_seq/body/#anonblk#73704343#58#immmed_62
    WRAPPER_sequence.sv(62) 0       1
/WRAPPER_sequence_pck/WRAPPER_write_read_seq/body/#anonblk#73704343#81##/immmed_85
    WRAPPER_sequence.sv(85) 0       1

```

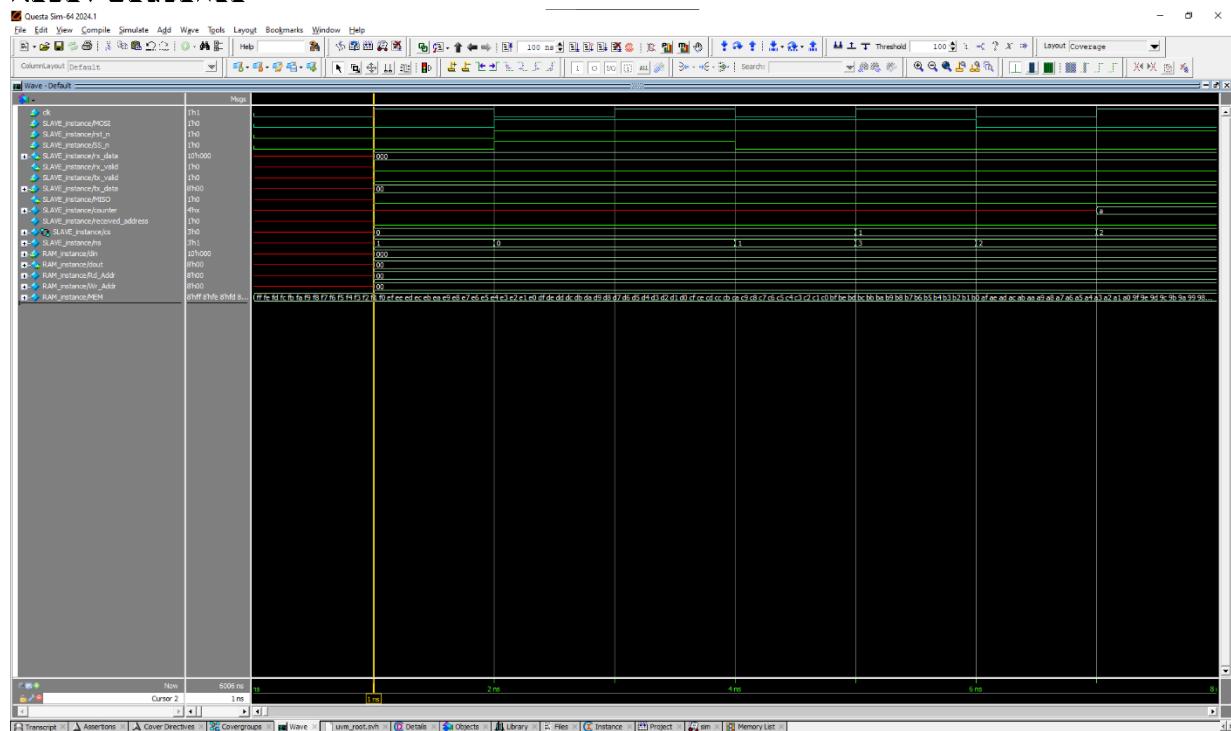
Ln 1, Col 1 90% Windows (CRLF) UTF-8

5. ASSERTIONS TABLE

A	B	C	D	E	F	G
Feature	Assertion					
1 When the reset is asserted, after 1 clk cycle MISO is low, din is low and rx_valid is low	<code>@(posedge clk) (!rst_n) => (MISO == 10'b00_0000_0000) && !rx_valid)</code>					
2 When the reset is inactive, if the internal state of the slave is not READ_DATA, then MISO is not changed	<code>@(posedge clk) disable iff(!rst_n) (operation != 3'b100) => (MISO == \$past(MISO))</code>					
3						

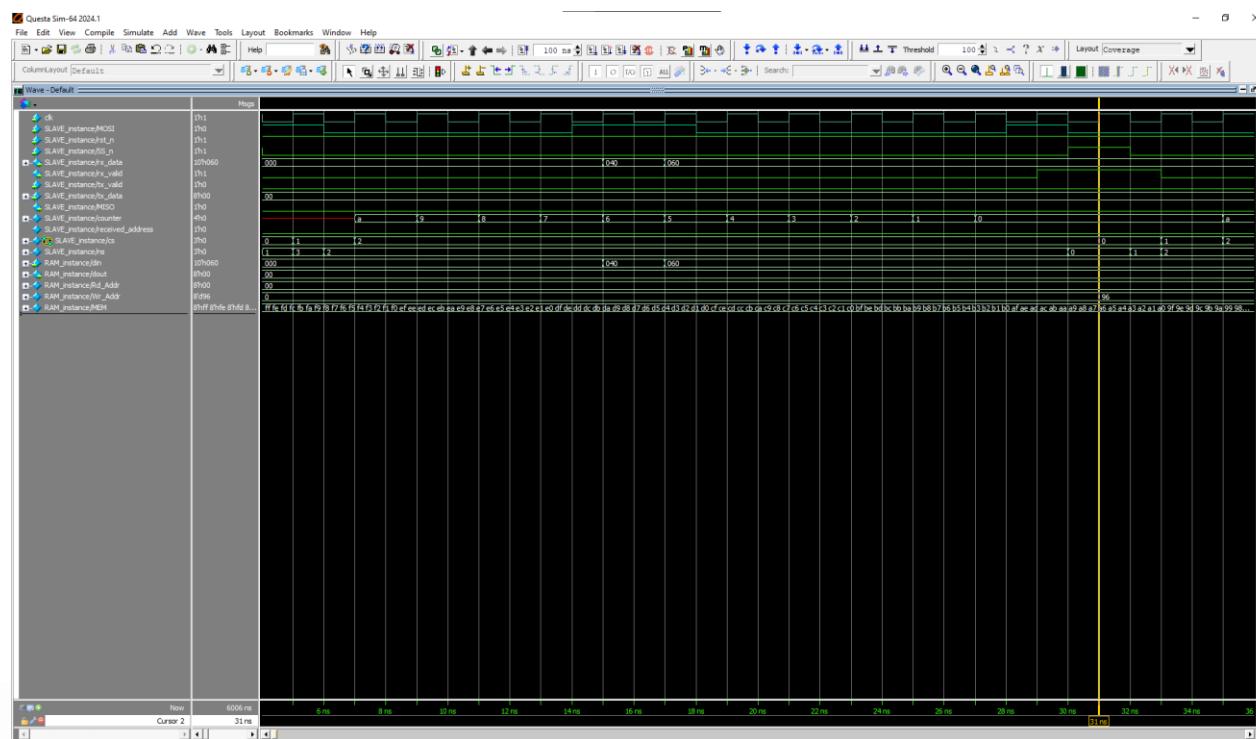
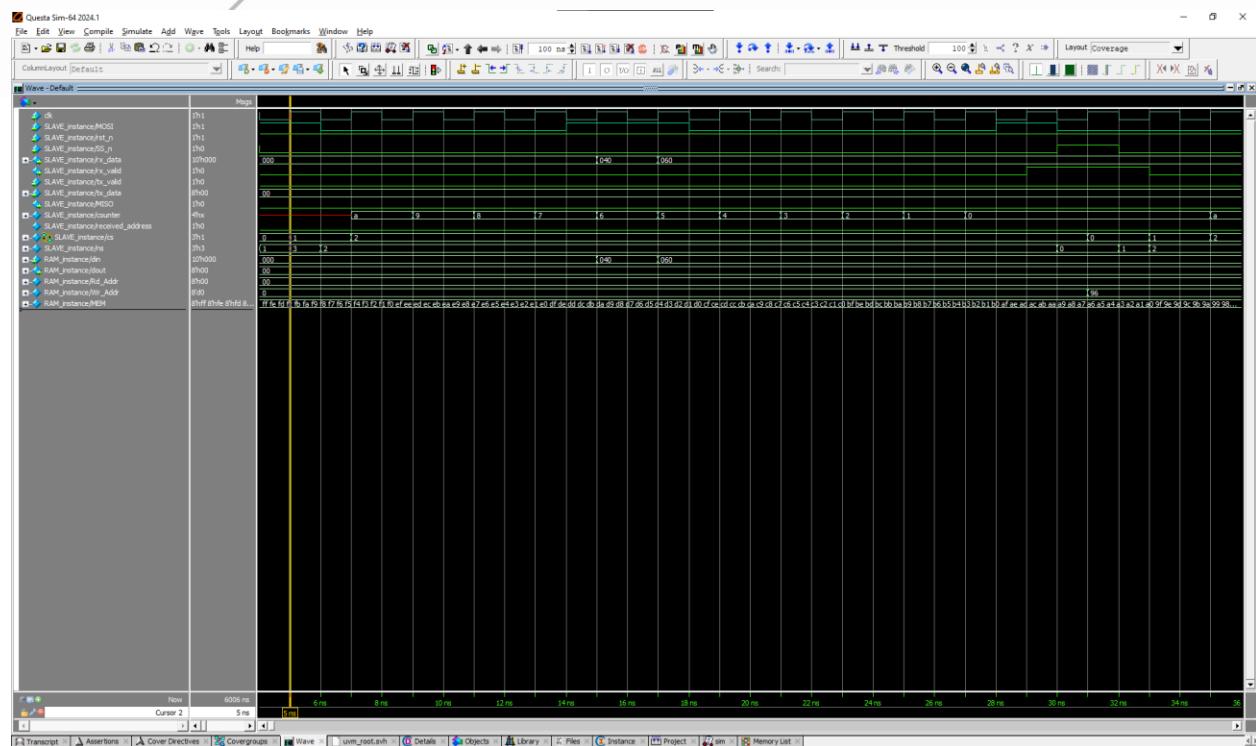
6. SIMULATION SNIPPETS

RESET SEQUENCE

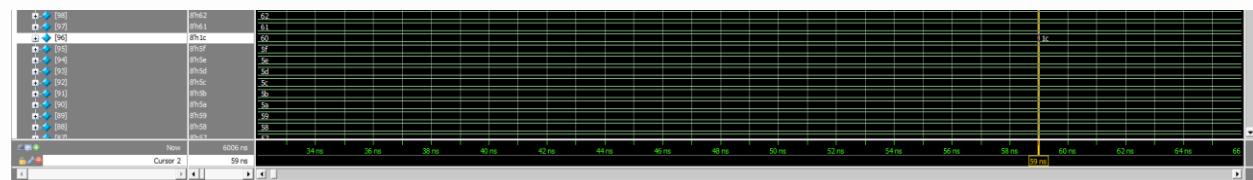
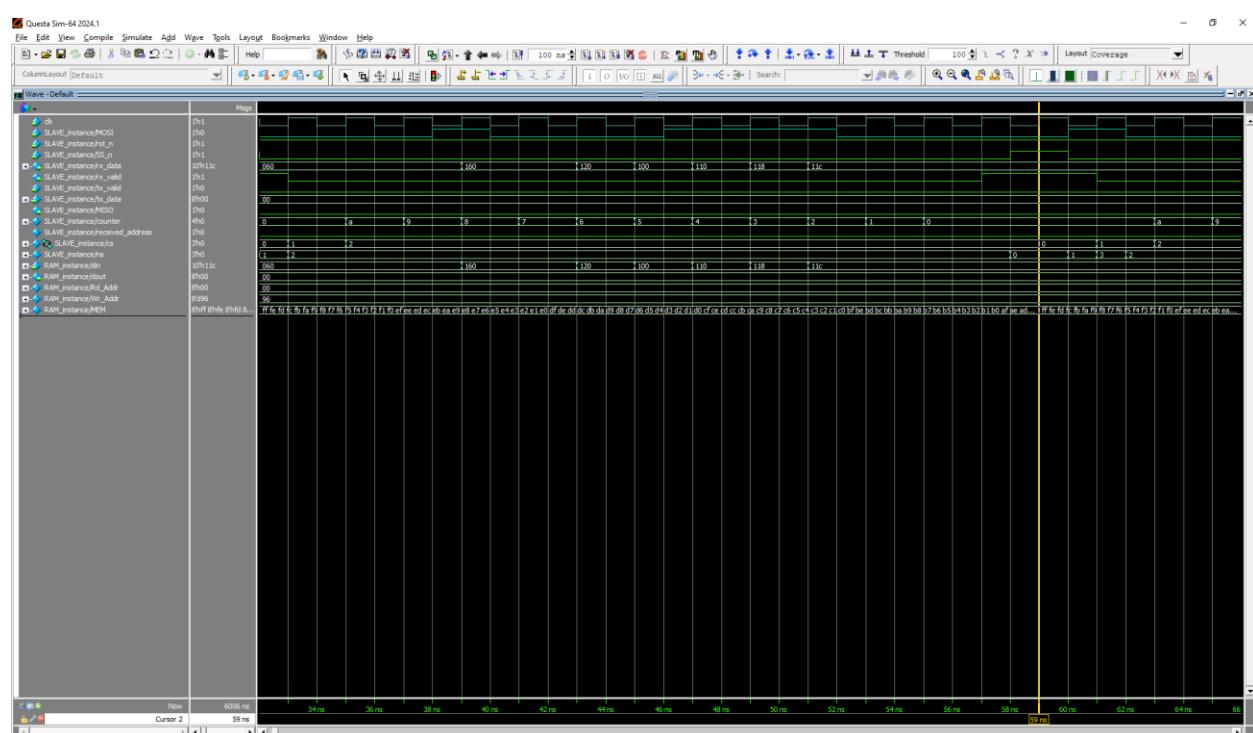
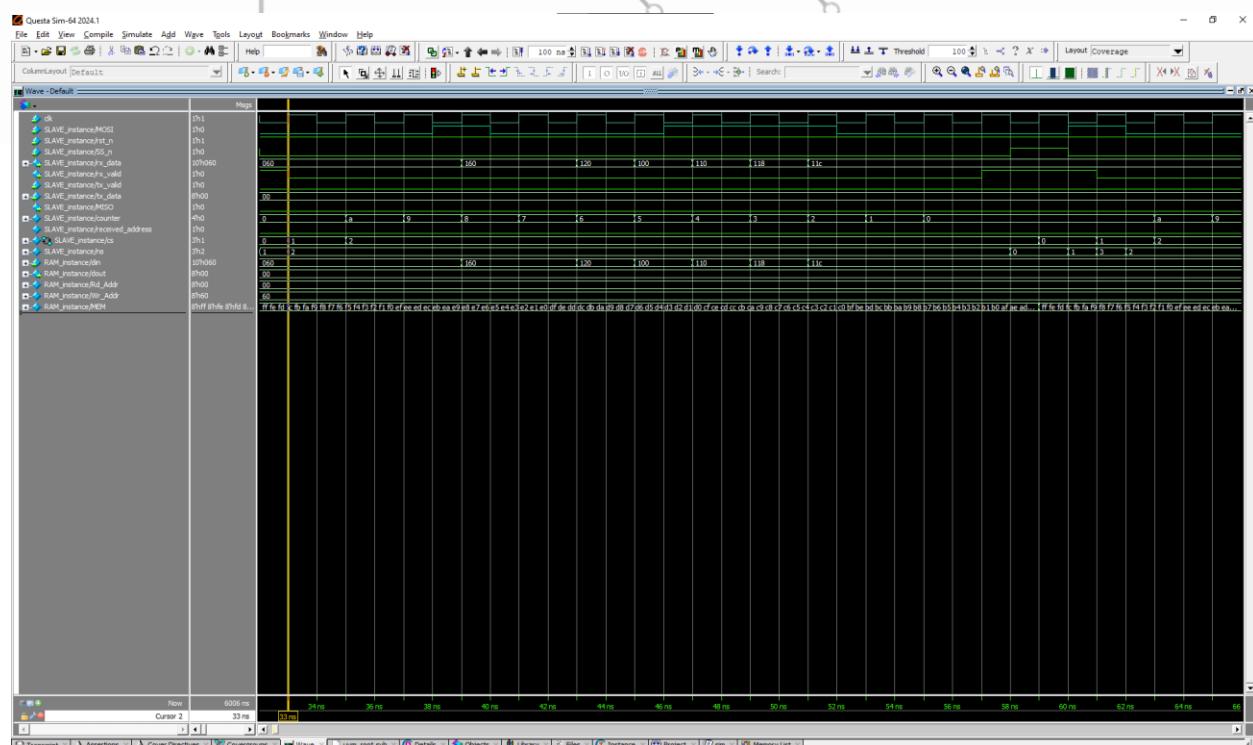


WRITE ONLY SEQUENCE

WRITE ADDR: (FROM 5 NS TO 31 NS, 13 CLK CYCLES)

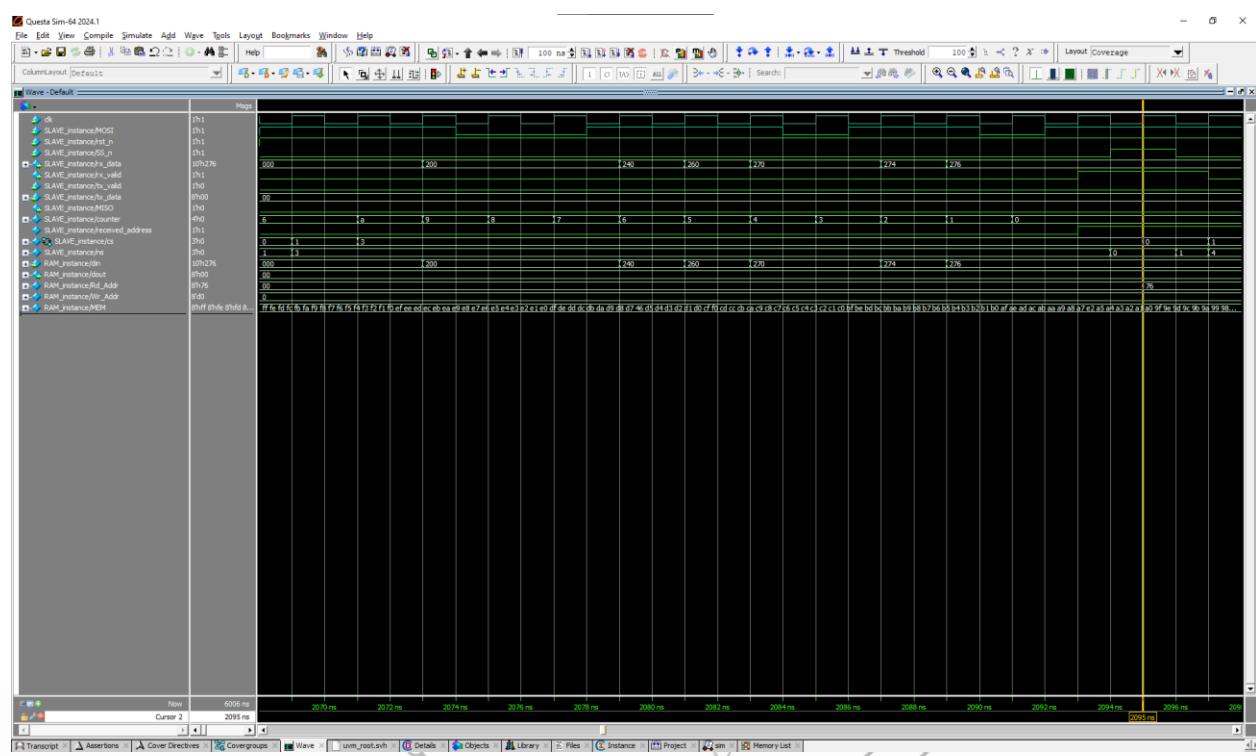
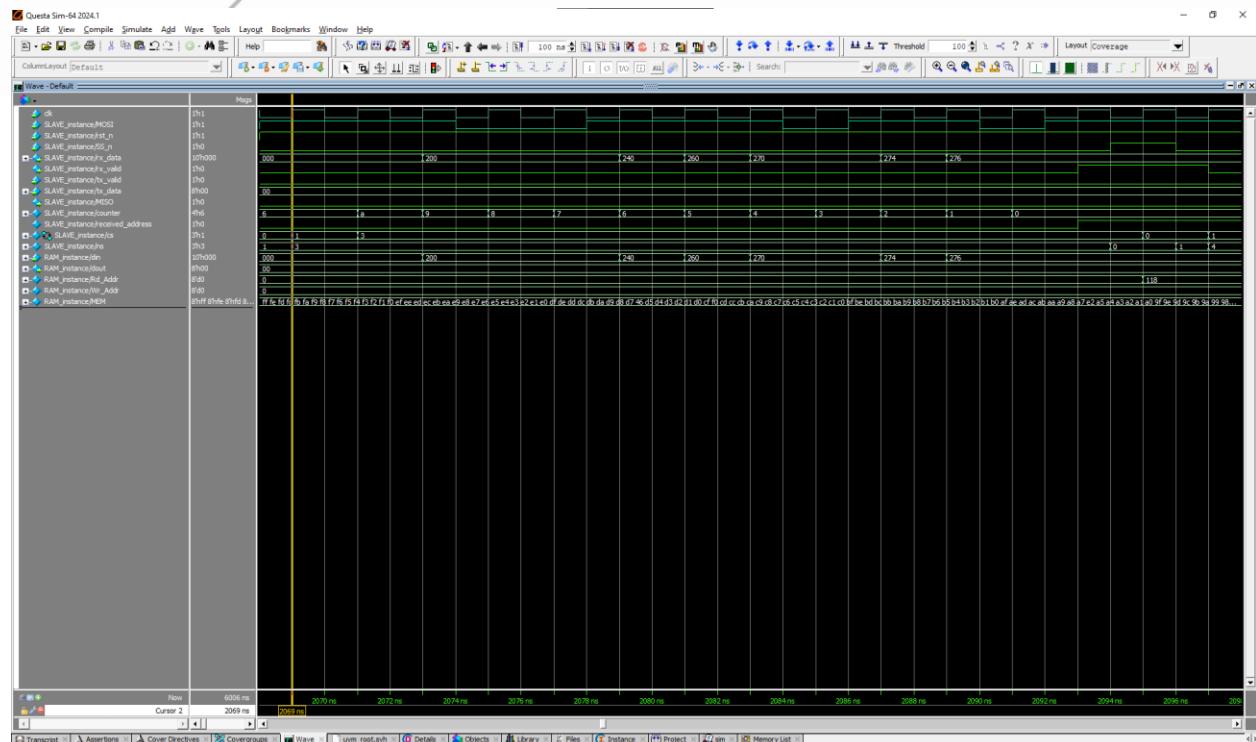


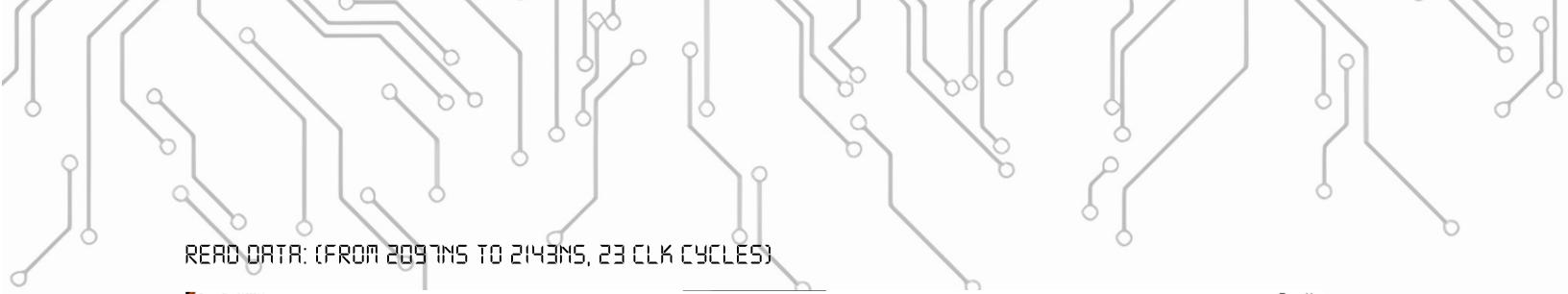
WRITE DATA (FROM 33NS TO 59NS, 13 CLK CYCLES)



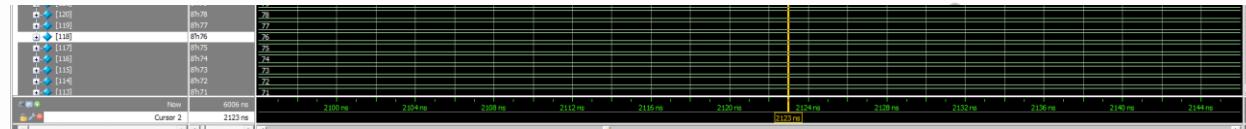
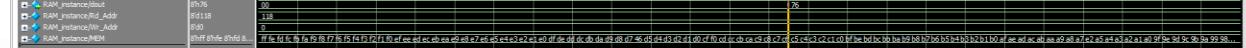
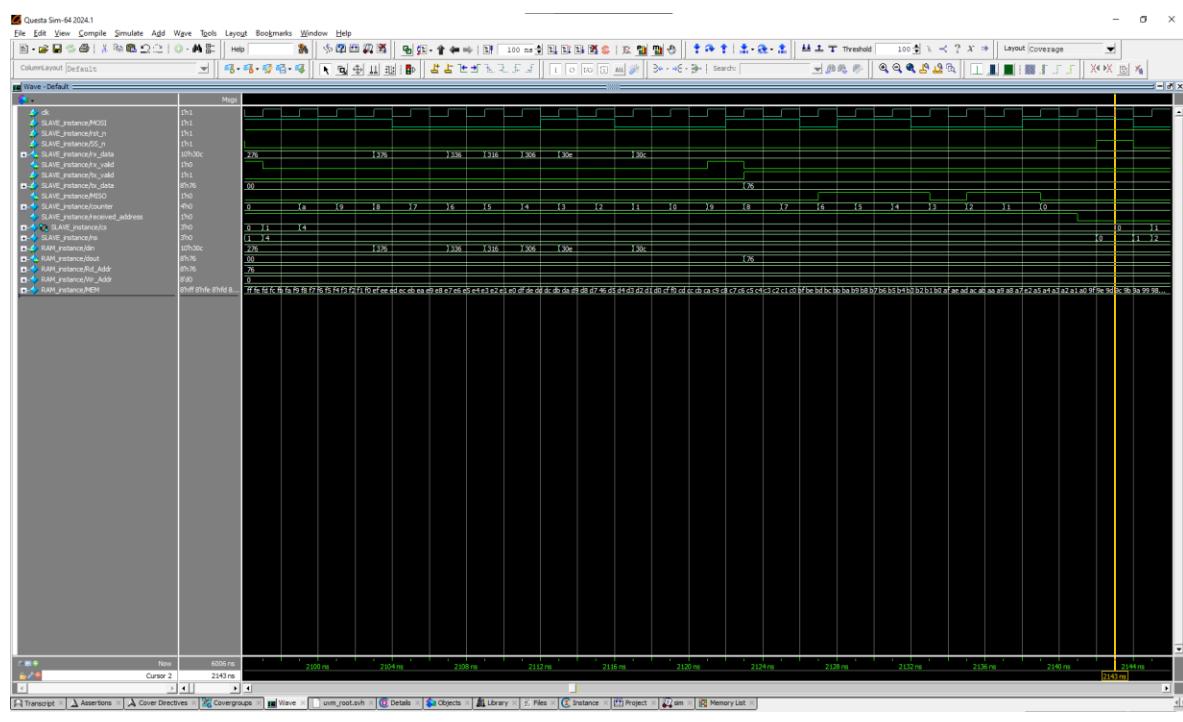
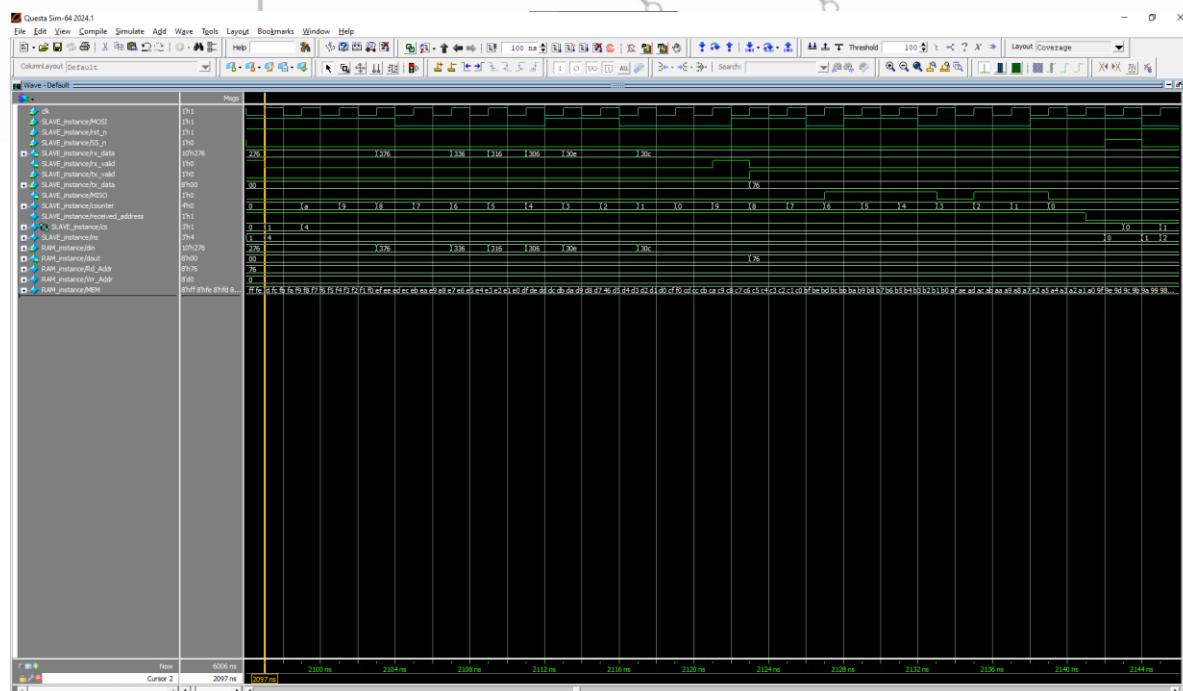
READ ONLY SEQUENCE

READ ADDRESS: (FROM 2069NS TO 2095NS, 13 CLK CYCLES)



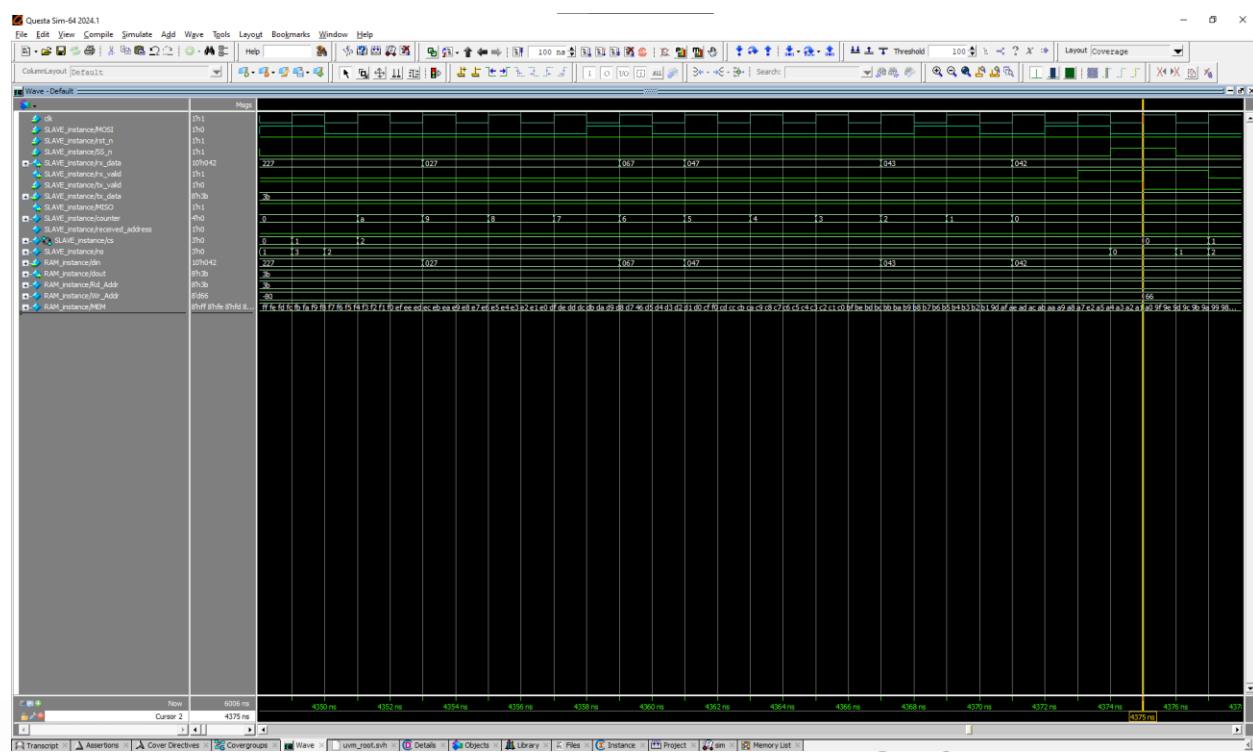
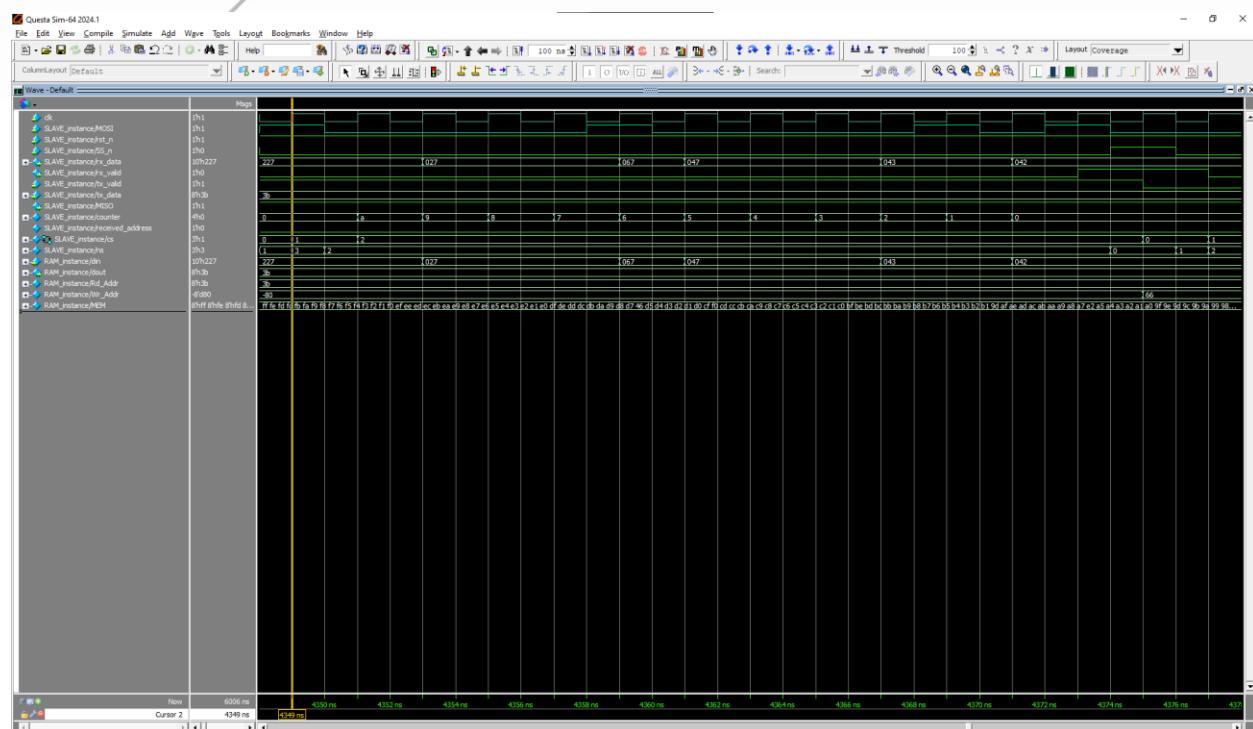


READ DATA: (FROM 2097NS TO 2143NS, 23 CLK CYCLES)

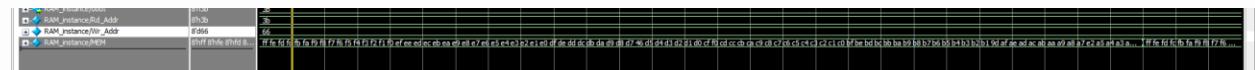
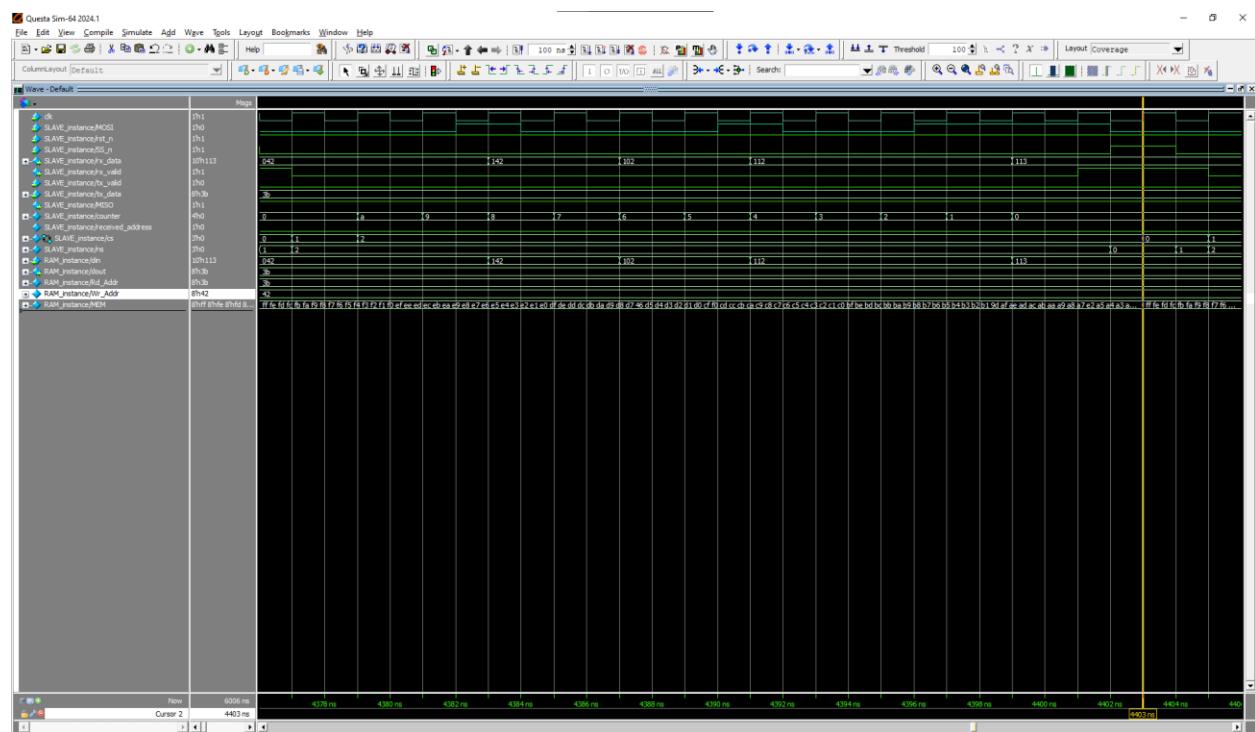
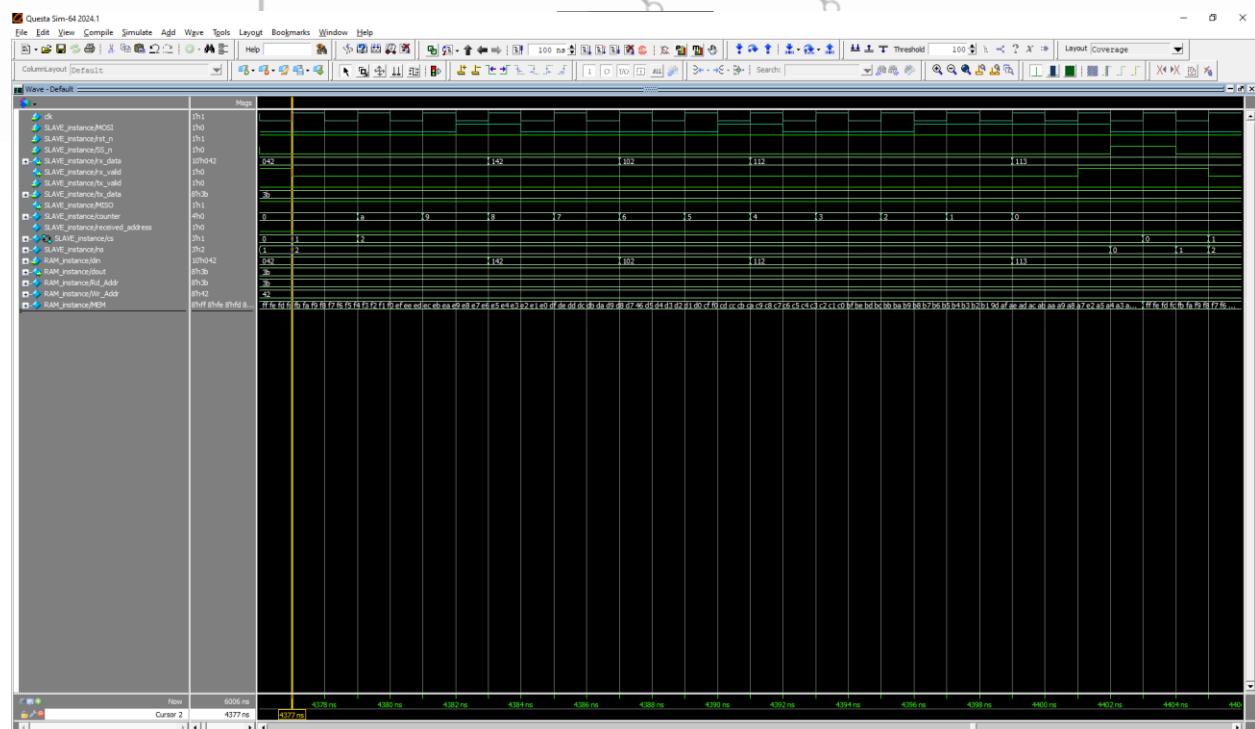


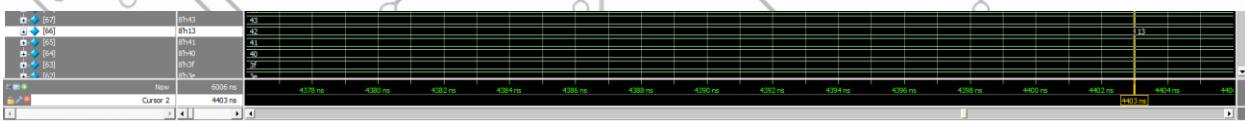
WRITE READ SEQUENCE

WRITE ADDRESS: (FROM 4349NS TO 4375NS, 13 CLK CYCLES)

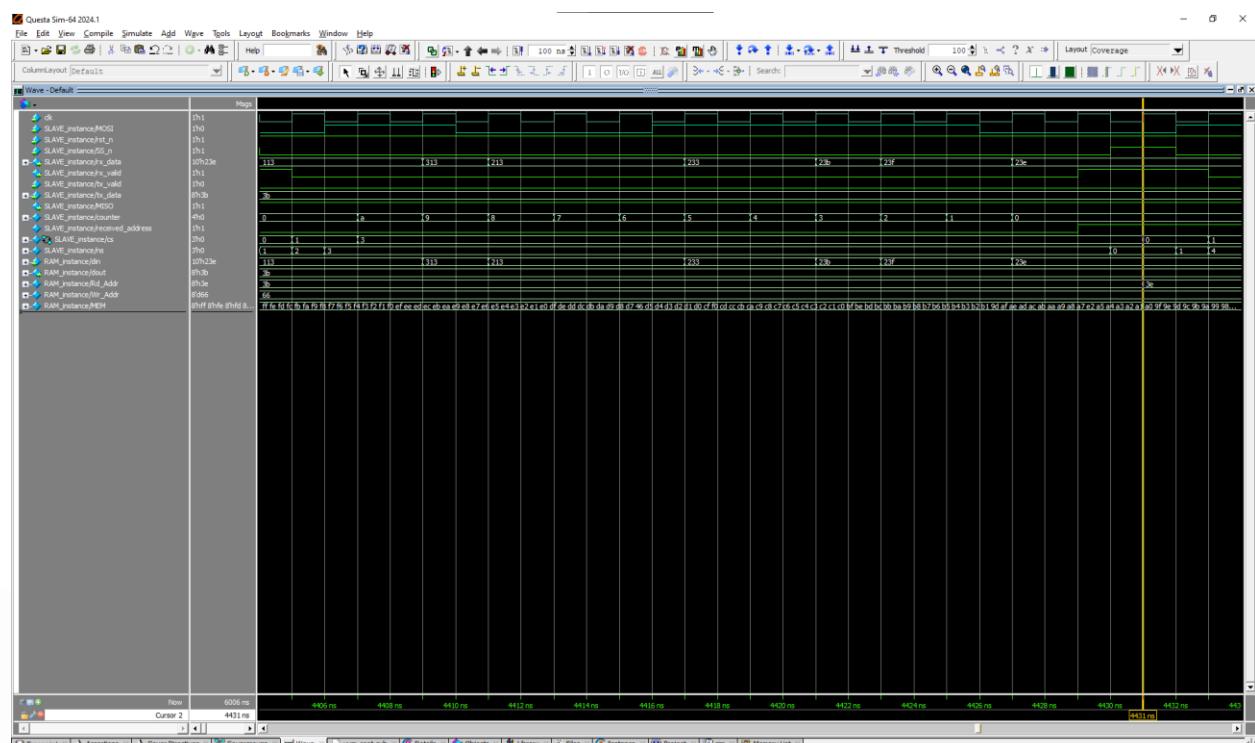
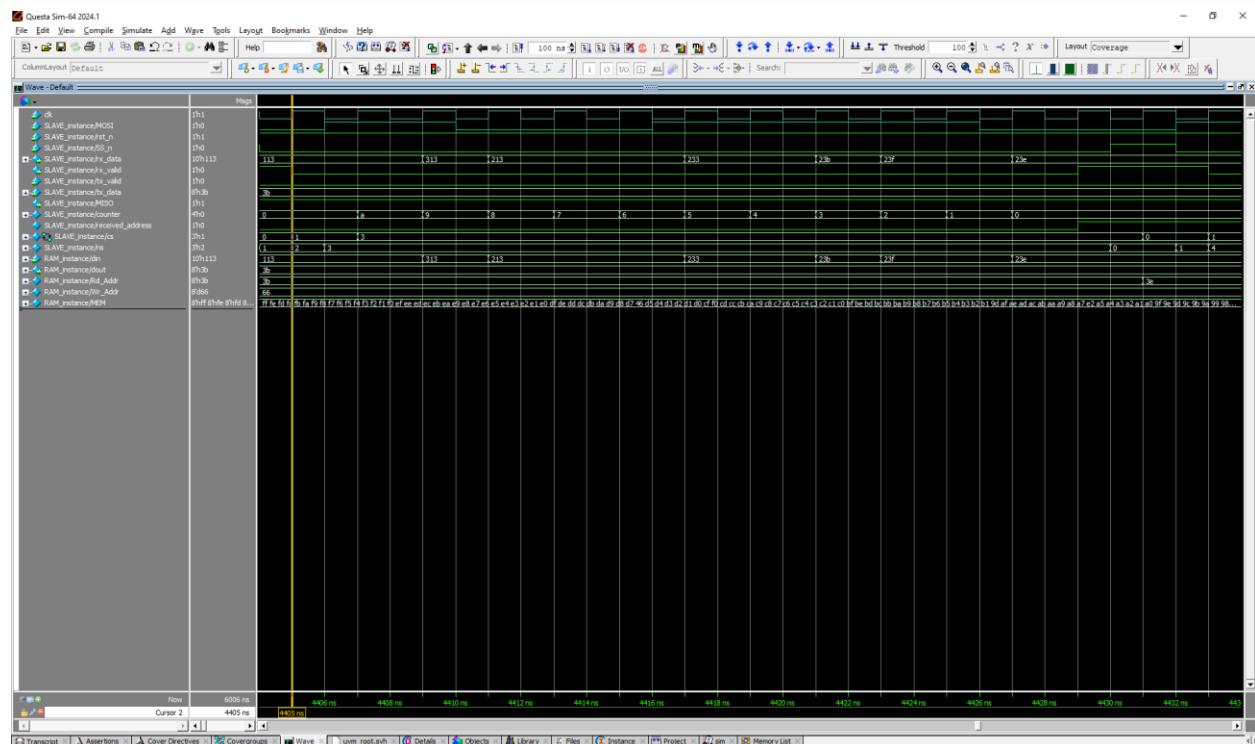


WRITE DATA: (FROM 4377NS TO 4403NS, 13 CLK CYCLES)

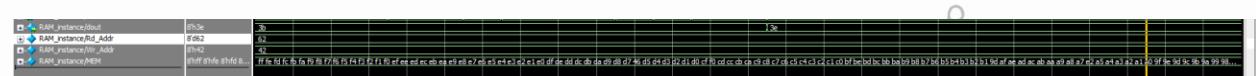
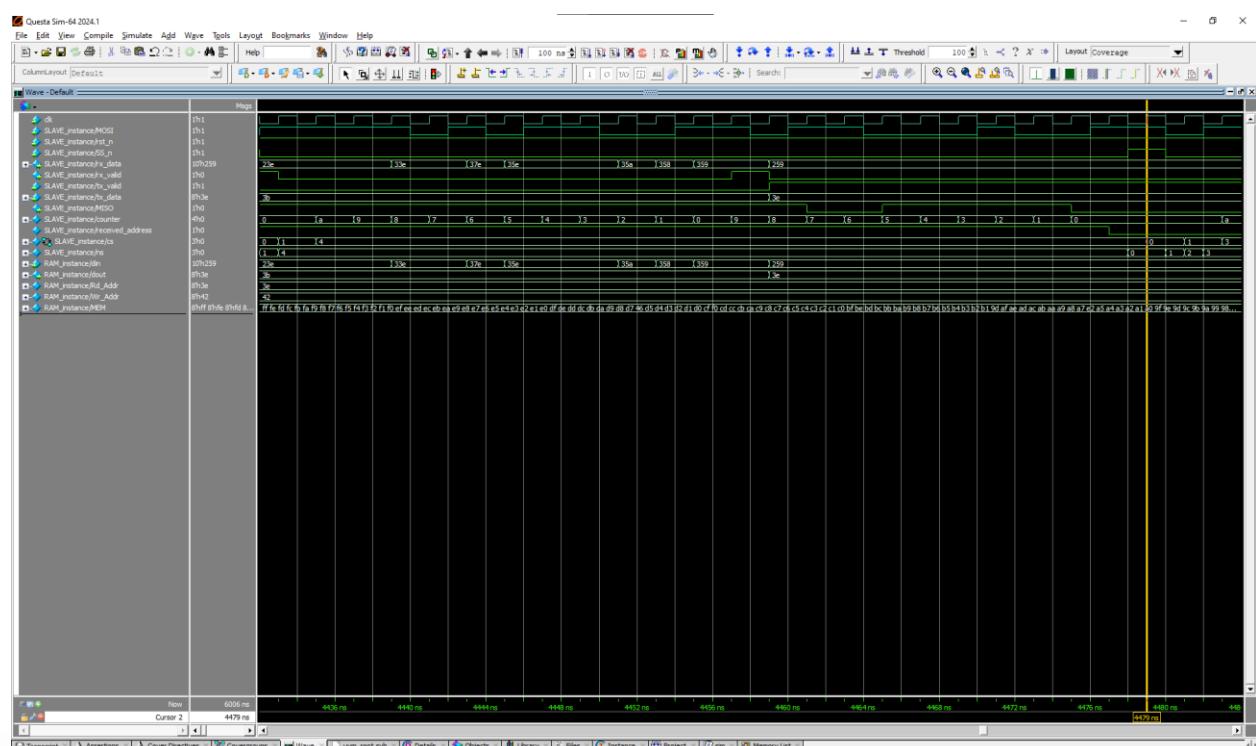
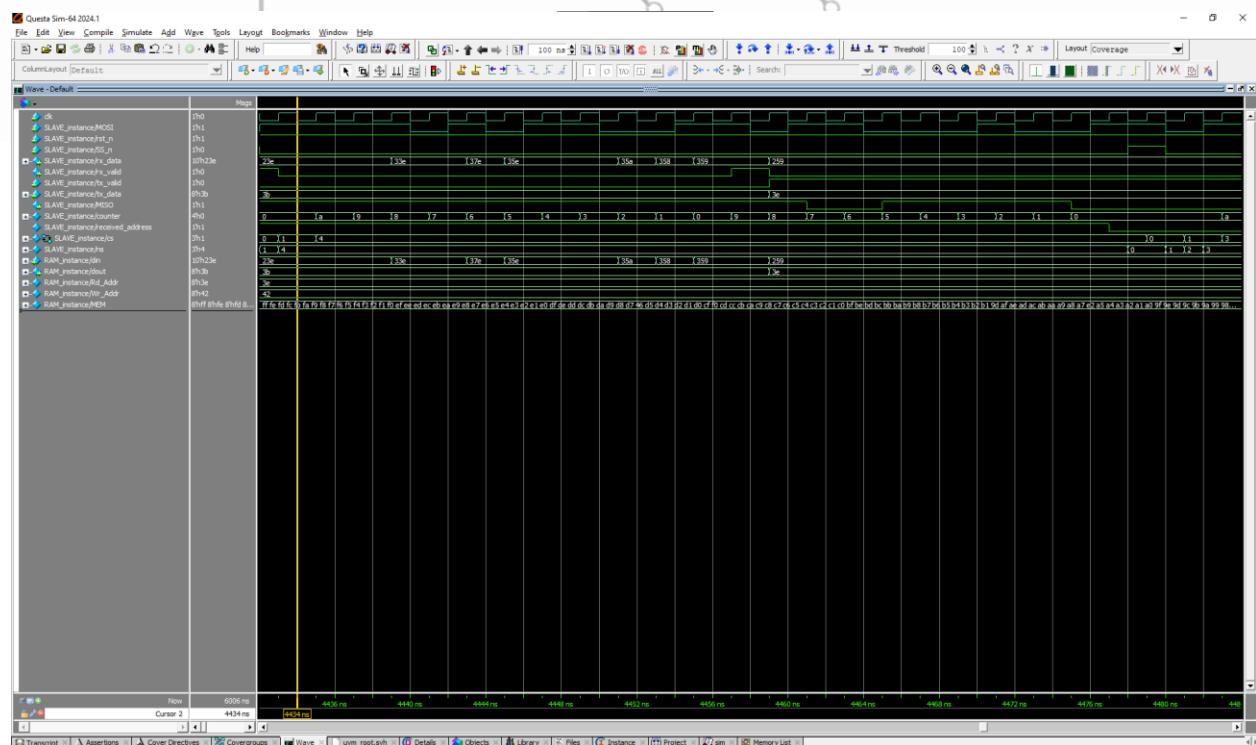


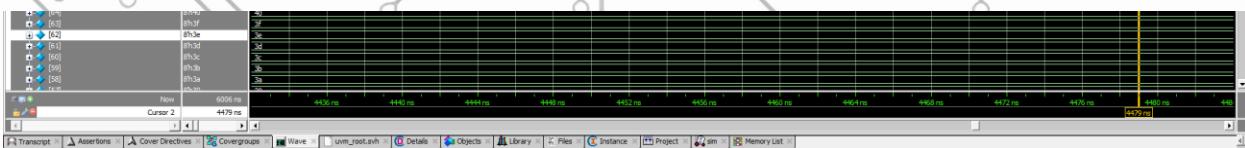


READ ADDRESS: (FROM 4405NS TO 4431NS , 13 CLK CYCLES)



READ DATA: (FROM 4433NS TO 4479NS, 23 CLK CYCLES)





TRANSCRIPT SNIPPETS

C:\QuestaSim-2024.1

File Edit View Compile Simulate Add Transcript Tools Layout Bookmarks Window Help

ColumnLayout [default] | Help | Search | Threshold 100% L A X | Layout Coverage

Transcript

```

# ***** IMPORTANT RELEASE NOTES *****
#
# You are using a version of the UVM library that has been compiled
# with UVM_NO_DEPRECATED undefined.
# See http://www.eda.org/rvbd/view.php?id=3313 for more details.
#
# You are using a version of the UVM library that has been compiled
# with UVM_OBJECT_MUST_HAVE_CONSTRUCTORS undefined.
# See http://www.eda.org/rvbd/view.php?id=3770 for more details.
#
# (Specify +UVM_NO_RELNOTES to turn off this notice)

# UVM_INFO verilog_src/questa_uvm_pkg.v(1.3)/rc/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg.v(1.3)/rc/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM] questa_uvm:init(all)
# UVM_INFO WRAPPER_top.sv(79) @ 0: uvm_test_top [run_phase] Reset Asserted
# -----
# Questa UVM framework recording Turned ON.
# -----
# +--> uvm_detail has been set to 0
# +--> To turn off, set "recording_detail" to off:
# +--> uvm_config_db#(int, uvm_object::BistreamID, string) recording_detail;
# +--> uvm_config_db#(int, uvm_object::BistreamID, string) "recording_detail", 0);
# -----
# UVM_INFO WRAPPER_top.sv(11) @ 21: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO WRAPPER_top.sv(11) @ 20001: uvm_test_top [run_phase] Write Read Seq Started
# UVM_INFO WRAPPER_top.sv(18) @ 20021: uvm_test_top [run_phase] Write Only Seq Ended
# UVM_INFO WRAPPER_top.sv(17) @ 20021: uvm_test_top [run_phase] Reset Asserted
# UVM_INFO WRAPPER_top.sv(16) @ 20041: uvm_test_top [run_phase] Read Only Seq Started
# UVM_INFO WRAPPER_top.sv(15) @ 20041: uvm_test_top [run_phase] Read Only Seq Ended
# UVM_INFO WRAPPER_top.sv(17) @ 40041: uvm_test_top [run_phase] Reset Asserted
# UVM_INFO WRAPPER_top.sv(16) @ 40041: uvm_test_top [run_phase] Read Only Seq Started
# UVM_INFO WRAPPER_top.sv(101) @ 40041: uvm_test_top [run_phase] Write Read Seq Started
# UVM_INFO WRAPPER_top.sv(100) @ 40041: uvm_test_top [run_phase] Write Read Seq Ended
# UVM_INFO WRAPPER_top.sv(100) @ 40041: uvm_test_top [run_phase] TEST_DONE: TEST_DONE phase is ready to proceed to the 'extract' phase
# UVM_INFO RAM_soccoreboard.sv(60) @ 40041: uvm_test_top.ram_env.ram_ab [report_phase] Correct Tests = 3003, Error Tests = 0
# UVM_INFO SFI_soccoreboard.sv(41) @ 40041: uvm_test_top.slv_slave_env_mb [report_phase] SFI SLAVE SCOREBOARD
# UVM_INFO SFI_soccoreboard.sv(41) @ 40041: uvm_test_top.slv_slave_env_mb [report_phase] Total failed rx_data: 3003
# UVM_INFO SFI_soccoreboard.sv(43) @ 40041: uvm_test_top.slv_slave_env_mb [report_phase] Total successful rx_valid: 3003
# UVM_INFO SFI_soccoreboard.sv(44) @ 40041: uvm_test_top.slv_slave_env_mb [report_phase] Total failed rx_valid: 0
# UVM_INFO SFI_soccoreboard.sv(45) @ 40041: uvm_test_top.slv_slave_env_mb [report_phase] Total successful rx_data: 3003
# UVM_INFO SFI_soccoreboard.sv(46) @ 40041: uvm_test_top.slv_slave_env_mb [report_phase] Total failed rx_data: 0
# UVM_INFO SFI_soccoreboard.sv(47) @ 40041: uvm_test_top.slv_slave_env_mb [report_phase] Total failed MISO: 0
# UVM_INFO WRAPPER_soccoreboard.sv(51) @ 40041: uvm_test_top.wrapper_ab [report_phase] Correct Tests = 3003, Error Tests = 0
# -----
# --- UVM Report Summary ---
# -----
# +--> Report counts by severity
# UVM_ERROR : 28
# UVM_WARNING : 0
# UVM_INFO : 0
# UVM_FATAL : 0
# +--> Report counts by id
# (Questa UVM) 2
# (UVM) 1
# (TEST_DONE) 1
# (report_phase) 9
# (UVM) 12
# +--> Hotel $finish : C:/questasim64_2024.1/win64../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
# Time: 4006 ns Iterations: 41 Instance: /WRAPPER_top

```