SPI-UVM-Verification

Digital Verification SV8

Submitted by:
Karim Walid Abdelaziz
Eiad Hassan Anwar

Submitted to:
Eng. Kareem Waseem

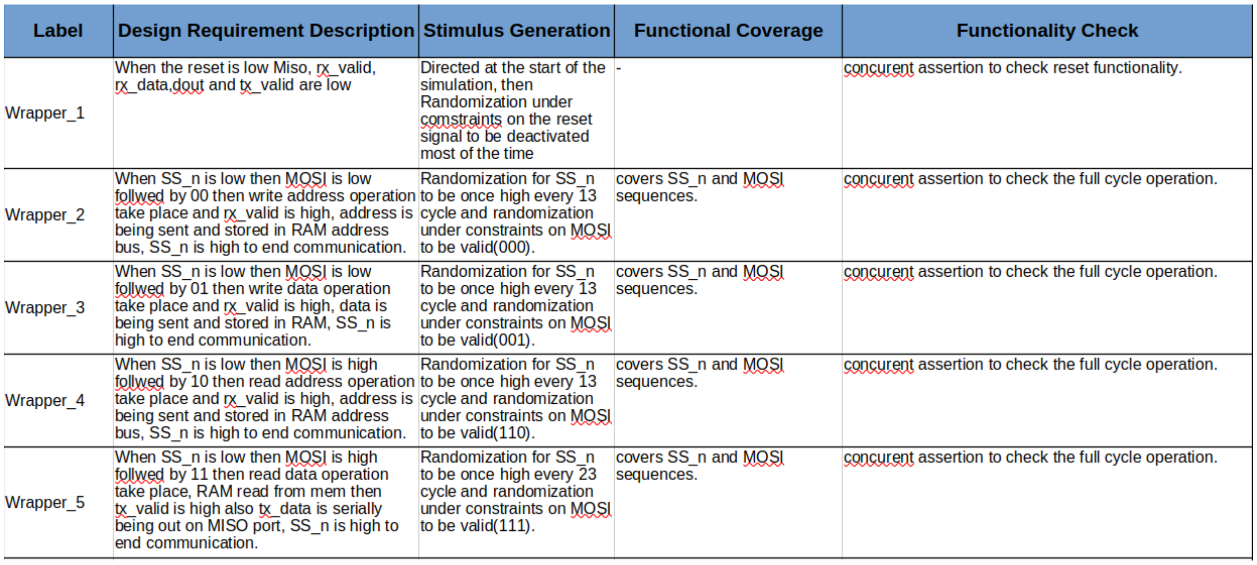
October 16, 2025

Contents

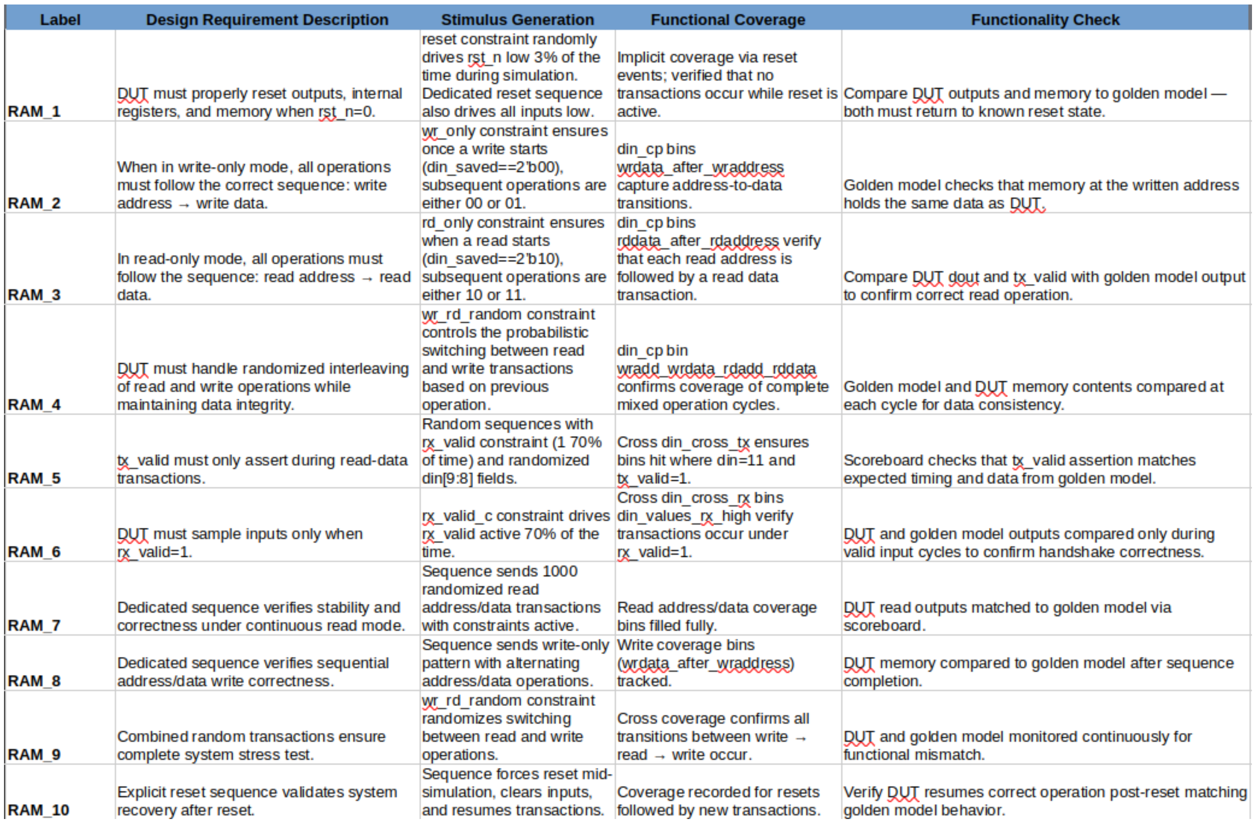
1 Project Github Link

http://github.com/Karim727/SPI-UVM-Verification/tree/main

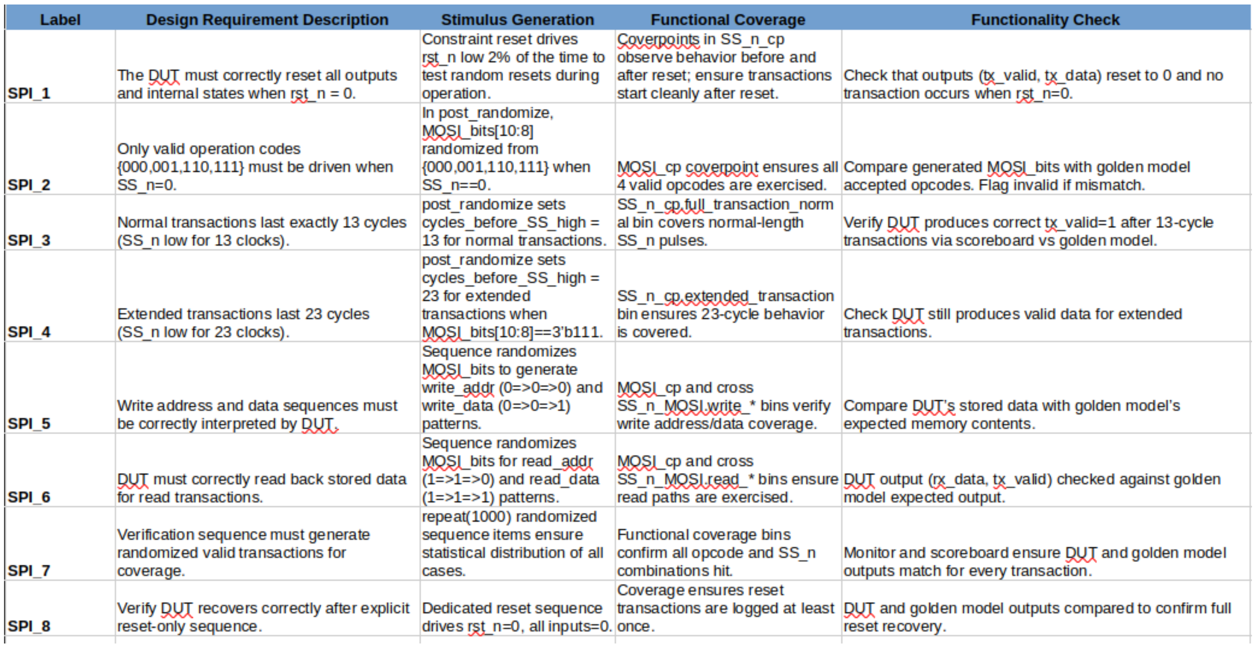
2 Verification Plan



Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
Wrapper_1	When the reset is low Miso, rx_valid, rx_data,dout and tx_valid are low	Directed at the start of the simulation, then Randomization under comstraints on the reset signal to be deactivated most of the time	-	concurent assertion to check reset functionality.
Wrapper_2	When SS_n is low then MOSI is low follwed by 00 then write address operation take place and rx_valid is high, address is being sent and stored in RAM address bus, SS_n is high to end communication.	Randomization for SS_n to be once high every 13 cycle and randomization under constraints on MOSI to be valid(000).	covers SS_n and MOSI sequences.	concurent assertion to check the full cycle operation.
Wrapper_3	When SS_n is low then MOSI is low follwed by 01 then write data operation take place and rx_valid is high, data is being sent and stored in RAM, SS_n is high to end communication.	Randomization for SS_n to be once high every 13 cycle and randomization under constraints on MOSI to be valid(001).	covers SS_n and MOSI sequences.	concurent assertion to check the full cycle operation.
Wrapper_4	When SS_n is low then MOSI is high follwed by 10 then read address operation take place and rx_valid is high, address is being sent and stored in RAM address bus, SS_n is high to end communication.	Randomization for SS_n to be once high every 13 cycle and randomization under constraints on MOSI to be valid(110).	covers SS_n and MOSI sequences.	concurent assertion to check the full cycle operation.
Wrapper_5	When SS_n is low then MOSI is high follwed by 11 then read data operation take place, RAM read from mem then tx_valid is high also tx_data is serially being out on MISO port, SS_n is high to end communication.	Randomization for SS_n to be once high every 23 cycle and randomization under constraints on MOSI to be valid(111).	covers SS_n and MOSI sequences.	concurent assertion to check the full cycle operation.



Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
RAM_1	DUT must properly reset outputs, internal registers, and memory when rst_n=0.	reset constraint randomly drives rst_n low 3% of the time during simulation. Dedicated reset sequence also drives all inputs low.	Implicit coverage via reset events; verified that no transactions occur while reset is active.	Compare DUT outputs and memory to golden model — both must return to known reset state.
RAM_2	When in write-only mode, all operations must follow the correct sequence: write address → write data.	wr_only constraint ensures once a write starts (din_saved==2'b00), subsequent operations are either 00 or 01.	din_cp bins wrdata_after_wraddress capture address-to-data transitions.	Golden model checks that memory at the written address holds the same data as DUT.
RAM_3	In read-only mode, all operations must follow the sequence: read address → read data.	rd_only constraint ensures when a read starts (din_saved==2'b10), subsequent operations are either 10 or 11.	din_cp bins rddata_after_rdaddress verify that each read address is followed by a read data transaction.	Compare DUT dout and tx_valid with golden model output to confirm correct read operation.
RAM_4	DUT must handle randomized interleaving of read and write operations while maintaining data integrity.	wr_rd_random constraint controls the probabilistic switching between read and write transactions based on previous operation.	din_cp bin wradd_wrdata_rdadd_rddata confirms coverage of complete mixed operation cycles.	Golden model and DUT memory contents compared at each cycle for data consistency.
RAM_5	tx_valid must only assert during read-data transactions.	Random sequences with rx_valid constraint (1 70% of time) and randomized din[9:8] fields.	Cross din_cross_tx ensures bins hit where din=11 and tx_valid=1.	Scoreboard checks that tx_valid assertion matches expected timing and data from golden model.
RAM_6	DUT must sample inputs only when rx_valid=1.	rx_valid_c constraint drives rx_valid active 70% of the time.	Cross din_cross_rx bins din_values_rx_high verify transactions occur under rx_valid=1.	DUT and golden model outputs compared only during valid input cycles to confirm handshake correctness.
RAM_7	Dedicated sequence verifies stability and correctness under continuous read mode.	Sequence sends 1000 randomized read address/data transactions with constraints active.	Read address/data coverage bins filled fully.	DUT read outputs matched to golden model via scoreboard.
RAM_8	Dedicated sequence verifies sequential address/data write correctness.	Sequence sends write-only pattern with alternating address/data operations.	Write coverage bins (wrdata_after_wraddress) tracked.	DUT memory compared to golden model after sequence completion.
RAM_9	Combined random transactions ensure complete system stress test.	wr_rd_random constraint randomizes switching between read and write operations.	Cross coverage confirms all transitions between write → read → write occur.	DUT and golden model monitored continuously for functional mismatch.
RAM_10	Explicit reset sequence validates system recovery after reset.	Sequence forces reset mid-simulation, clears inputs, and resumes transactions.	Coverage recorded for resets followed by new transactions.	Verify DUT resumes correct operation post-reset matching golden model behavior.



Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
SPI_1	The DUT must correctly reset all outputs and internal states when rst_n = 0.	Constraint reset drives rst_n low 2% of the time to test random resets during operation.	Coverpoints in SS_n_cp observe behavior before and after reset; ensure transactions start cleanly after reset.	Check that outputs (tx_valid, tx_data) reset to 0 and no transaction occurs when rst_n=0.
SPI_2	Only valid operation codes {000,001,110,111} must be driven when SS_n=0.	In post_randomize, MOSI_bits[10:8] randomized from {000,001,110,111} when SS_n==0.	MOSI_cp coverpoint ensures all 4 valid opcodes are exercised.	Compare generated MOSI_bits with golden model accepted opcodes. Flag invalid if mismatch.
SPI_3	Normal transactions last exactly 13 cycles (SS_n low for 13 clocks).	post_randomize sets cycles_before_SS_high = 13 for normal transactions.	SS_n_cp.full_transaction_normal bin covers normal-length SS_n pulses.	Verify DUT produces correct tx_valid=1 after 13-cycle transactions via scoreboard vs golden model.
SPI_4	Extended transactions last 23 cycles (SS_n low for 23 clocks).	post_randomize sets cycles_before_SS_high = 23 for extended transactions when MOSI_bits[10:8]==3'b111.	SS_n_cp.extended_transaction bin ensures 23-cycle behavior is covered.	Check DUT still produces valid data for extended transactions.
SPI_5	Write address and data sequences must be correctly interpreted by DUT.	Sequence randomizes MOSI_bits to generate write_addr (0=>0=>0) and write_data (0=>0=>1) patterns.	MOSI_cp and cross SS_n_MOSI.write_* bins verify write address/data coverage.	Compare DUT's stored data with golden model's expected memory contents.
SPI_6	DUT must correctly read back stored data for read transactions.	Sequence randomizes MOSI_bits for read_addr (1=>1=>0) and read_data (1=>1=>1) patterns.	MOSI_cp and cross SS_n_MOSI.read_* bins ensure read paths are exercised.	DUT output (rx_data, tx_valid) checked against golden model expected output.
SPI_7	Verification sequence must generate randomized valid transactions for coverage.	repeat(1000) randomized sequence items ensure statistical distribution of all cases.	Functional coverage bins confirm all opcode and SS_n combinations hit.	Monitor and scoreboard ensure DUT and golden model outputs match for every transaction.
SPI_8	Verify DUT recovers correctly after explicit reset-only sequence.	Dedicated reset sequence drives rst_n=0, all inputs=0.	Coverage ensures reset transactions are logged at least once.	DUT and golden model outputs compared to confirm full reset recovery.

3 Bugs

3.1 RAM

Buggy Design:

```

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

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

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

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

reg [7:0] Rd_Addr, Wr_Addr;

always @(posedge clk) begin
    if (~rst_n) begin
        dout <= 0;
        tx_valid <= 0;
        Rd_Addr <= 0;
        Wr_Addr <= 0;
    end
    else
        if (rx_valid) begin
            case (din[9:8])
```

```
                2'b00 : Wr_Addr <= din[7:0];
                2'b01 : MEM[Wr_Addr] <= din[7:0];
                2'b10 : Rd_Addr <= din[7:0];
                2'b11 : dout <= MEM[Wr_Addr];
                default : dout <= 0;
            endcase
        end
        tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 : 1'b0;
end

endmodule
```

Fixed Design:

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

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

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

reg [7:0] Rd_Addr, Wr_Addr;

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

endmodule
```

3.2 SPI

Buggy Design:

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

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

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

reg [3:0] counter;
reg       received_address;

reg [2:0] cs, ns;

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

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

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

always @(posedge clk) begin
    if (~rst_n) begin
        rx_data <= 0;
        rx_valid <= 0;
        received_address <= 0;
        MISO <= 0;
    end
    else begin
        case (cs)
            IDLE : begin
                rx_valid <= 0;
            end
            CHK_CMD : begin
                counter <= 10;
            end
            WRITE : begin
                if (counter > 0) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
                else begin
                    rx_valid <= 1;
                end
            end
            READ_ADD : begin
                if (counter > 0) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
                else begin
                    rx_valid <= 1;
                    received_address <= 1;
                end
            end
            READ_DATA : begin
                if (tx_valid) begin
                    rx_valid <= 0;
                    if (counter > 0) begin
                        MISO <= tx_data[counter-1];
                        counter <= counter - 1;
                    end
                    else begin
                        received_address <= 0;
                    end
                end
                else begin
                    if (counter > 0) begin
                        rx_data[counter-1] <= MOSI;
                        counter <= counter - 1;
                    end
                    else begin
                        rx_valid <= 1;
                        counter <= 8;
                    end
                end
            end
        endcase
    end
end

endmodule
```

Fixed Design:

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

```

```verilog
/*localparam IDLE      = 3'b000;
localparam CHK_CMD   = 3'b001;
localparam WRITE     = 3'b010;
localparam READ_ADD  = 3'b011;
localparam READ_DATA = 3'b100;*/

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

reg [3:0] counter;
reg       received_address;
reg [9:0] MOSI_reg;
reg [2:0]  ns; //cs,

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

always @(*) begin
    case (cs)
        IDLE : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = CHK_CMD;
        end
        CHK_CMD : begin
            if (SS_n)
                ns = IDLE;
            else begin
                if (~MOSI)
                    ns = WRITE;
                else if(MOSI) begin //// else only is wrong if mosi
                   is x
                    if (~received_address)// <- BUG
                        ns = READ_ADD;
                    else
                        ns = READ_DATA;
                end
            end
        end
        WRITE : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = WRITE;
        end
        READ_ADD : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = READ_ADD;
        end
        READ_DATA : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = READ_DATA;
        end
    endcase
end

always @(posedge clk) begin
    if (~rst_n) begin
        rx_data <= 0;
        rx_valid <= 0;
        received_address <= 0;
        MOSI_reg <= 0;  // MOSI_reg is not used
                        // It's used to drive rx_data directly
                        // without modifing rx_data bit by bit
                           directly
                        // using MOSI to prevent errors in the ram
                           output
        MISO <= 0;
    end
    else begin
        case (cs)
            IDLE : begin
                rx_valid <= 0;
            end
            CHK_CMD : begin
                counter <= 10;
            end
            WRITE : begin
                if (counter > 0) begin
                    MOSI_reg[counter-1] <= MOSI;
                    counter <= counter - 1;
```

```verilog
                    end
                    else begin
                        rx_data <= MOSI_reg;
                        rx_valid <= 1;
                    end
                end
                READ_ADD : begin
                    if (counter > 0) begin
                        MOSI_reg[counter-1] <= MOSI;
                        counter <= counter - 1;
                    end
                    else begin
                        rx_data <= MOSI_reg;
                        rx_valid <= 1;
                        received_address <= 1;
                    end
                end
                READ_DATA : begin
                    if (tx_valid) begin
                        rx_valid <= 0;
                        if (counter > 0) begin
                            MISO <= tx_data[counter-1];
                            counter <= counter - 1;
                        end
                        else begin
                            received_address <= 0;
                        end
                    end
                    else begin
                        if (counter > 0) begin
                            MOSI_reg[counter-1] <= MOSI;
                            counter <= counter - 1;
                        end
                        else begin
                            rx_data <= MOSI_reg;
                            rx_valid <= 1;
                            counter <= 8;
                        end
                    end
                end
            endcase
        end
end


/////////////////////////
        //ASSERTIONS//
```

```
/////////////////////////////

`ifdef SIM

property reset_check;
    @(posedge clk) disable iff (rst_n)  (~rst_n) |-> ##1
        (~MISO && ~rx_valid && rx_data == 10'd0);
endproperty
/*always_comb begin
    if(~rst_n)
    a_reset: assert final(~MISO && ~rx_valid && rx_data == 10'd0);
end*/
property valid_command;
    @(posedge clk) disable iff (~rst_n)  (cs == CHK_CMD ##1 ~MISO
       [*3]) |-> ##10
        (rx_valid && $rose(SS_n) [->1]);///eventually
endproperty
property valid_transition_1;
    @(posedge clk) disable iff (~rst_n)  (cs == IDLE && ~SS_n) |->
       ##1
        (cs == CHK_CMD);
endproperty
property valid_transition_2;
    @(posedge clk) disable iff (~rst_n)  (cs == CHK_CMD && ~SS_n) |->
        ##1
        (cs == WRITE || cs == READ_ADD || cs == READ_DATA);
endproperty
//
property valid_transition_3;
    @(posedge clk) disable iff (~rst_n)  (cs == WRITE && ~SS_n) |->
       ##[1:22]
        (cs == IDLE);
endproperty
property valid_transition_4;
    @(posedge clk) disable iff (~rst_n)  (cs == READ_ADD && ~SS_n)
       |-> ##[1:22]
        (cs == IDLE);
endproperty
property valid_transition_5;
    @(posedge clk) disable iff (~rst_n)  (cs == READ_DATA && ~SS_n)
       |-> ##[1:22]
        (cs == IDLE);
endproperty
//
a_reset:assert property (reset_check);
ap_1:assert property (valid_command);
cp_1:cover property (valid_command);
```

```
ap_2:assert property (valid_transition_1);
cp_2:cover property (valid_transition_1);
ap_3:assert property (valid_transition_2);
cp_3:cover property (valid_transition_2);
ap_4:assert property (valid_transition_3);
cp_4:cover property (valid_transition_3);
ap_5:assert property (valid_transition_4);
cp_5:cover property (valid_transition_4);
ap_6:assert property (valid_transition_5);
cp_6:cover property (valid_transition_5);
`endif

endmodule
```

3.3 Wrapper

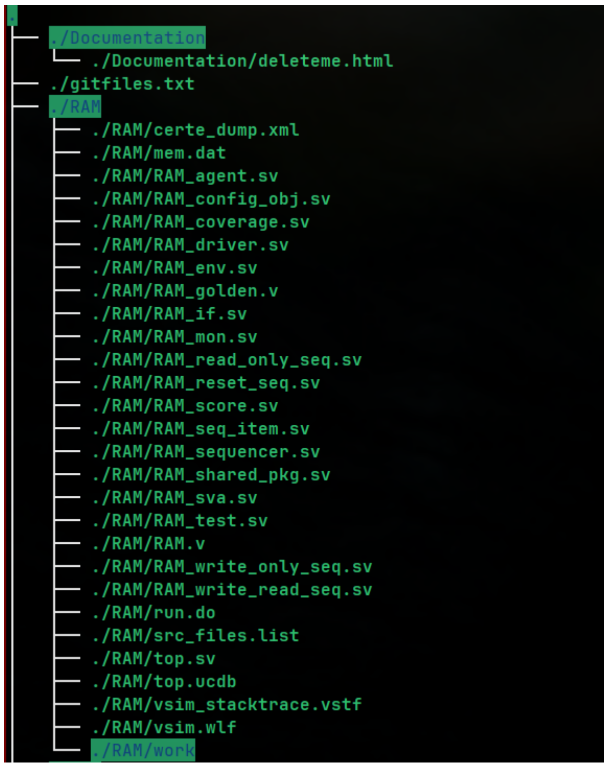
Buggy Design

```

module WRAPPER (MOSI,MISO,SS_n,clk,rst_n);

input  MOSI, SS_n, clk, rst_n;
output MISO;

wire [9:0] rx_data_din;
wire       rx_valid;
wire       tx_valid;
wire [7:0] tx_data_dout;

RAM   RAM_instance   (rx_data_din,clk,rst_n,rx_valid,tx_data_dout,
    tx_valid);
SLAVE SLAVE_instance (MOSI,MISO,SS_n,clk,rst_n,rx_data_din,rx_valid,
    tx_data_dout,tx_valid);

endmodule
```

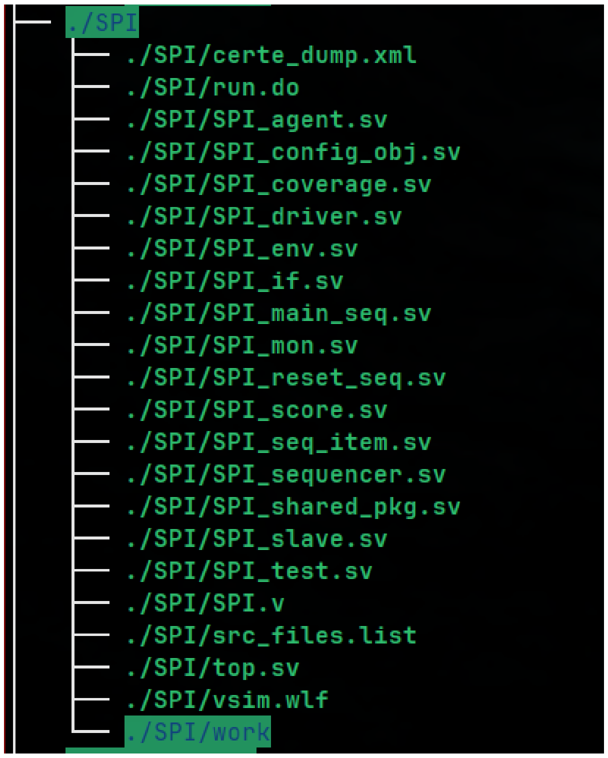
Fixed Design

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

wire [9:0] rx_data_din;
wire       rx_valid;
wire       tx_valid;
wire [7:0] tx_data_dout;

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

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

endmodule
```

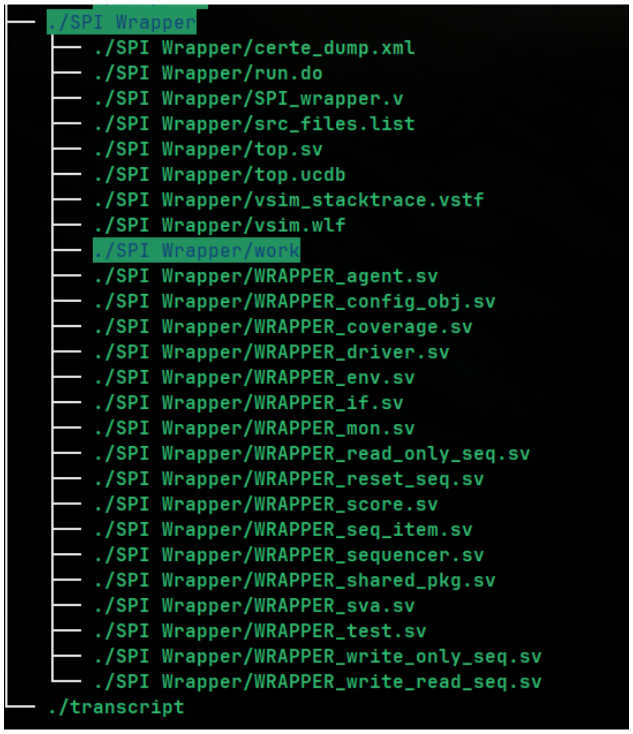
4 Files Hierarchy



```
.
├── ./Documentation
│   └── ./Documentation/deleteme.html
├── ./gitfiles.txt
├── ./RAM
│   ├── ./RAM/certe_dump.xml
│   ├── ./RAM/mem.dat
│   ├── ./RAM/RAM_agent.sv
│   ├── ./RAM/RAM_config_obj.sv
│   ├── ./RAM/RAM_coverage.sv
│   ├── ./RAM/RAM_driver.sv
│   ├── ./RAM/RAM_env.sv
│   ├── ./RAM/RAM_golden.v
│   ├── ./RAM/RAM_if.sv
│   ├── ./RAM/RAM_mon.sv
│   ├── ./RAM/RAM_read_only_seq.sv
│   ├── ./RAM/RAM_reset_seq.sv
│   ├── ./RAM/RAM_score.sv
│   ├── ./RAM/RAM_seq_item.sv
│   ├── ./RAM/RAM_sequencer.sv
│   ├── ./RAM/RAM_shared_pkg.sv
│   ├── ./RAM/RAM_sva.sv
│   ├── ./RAM/RAM_test.sv
│   ├── ./RAM/RAM.v
│   ├── ./RAM/RAM_write_only_seq.sv
│   ├── ./RAM/RAM_write_read_seq.sv
│   ├── ./RAM/run.do
│   ├── ./RAM/src_files.list
│   ├── ./RAM/top.sv
│   ├── ./RAM/top.ucdb
│   ├── ./RAM/vsim_stacktrace.vstf
│   ├── ./RAM/vsim.wlf
│   └── ./RAM/work
```

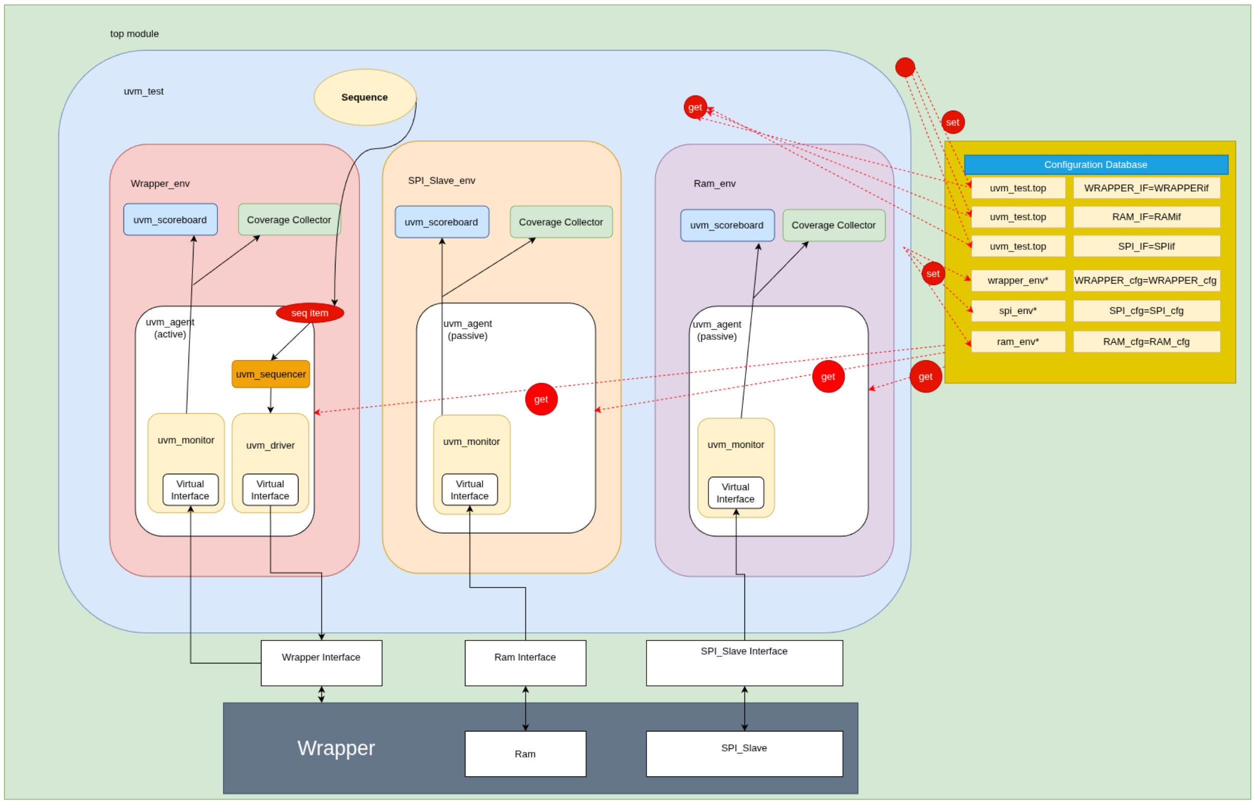


```
├── ./SPI
│   ├── ./SPI/certe_dump.xml
│   ├── ./SPI/run.do
│   ├── ./SPI/SPI_agent.sv
│   ├── ./SPI/SPI_config_obj.sv
│   ├── ./SPI/SPI_coverage.sv
│   ├── ./SPI/SPI_driver.sv
│   ├── ./SPI/SPI_env.sv
│   ├── ./SPI/SPI_if.sv
│   ├── ./SPI/SPI_main_seq.sv
│   ├── ./SPI/SPI_mon.sv
│   ├── ./SPI/SPI_reset_seq.sv
│   ├── ./SPI/SPI_score.sv
│   ├── ./SPI/SPI_seq_item.sv
│   ├── ./SPI/SPI_sequencer.sv
│   ├── ./SPI/SPI_shared_pkg.sv
│   ├── ./SPI/SPI_slave.sv
│   ├── ./SPI/SPI_test.sv
│   ├── ./SPI/SPI.v
│   ├── ./SPI/src_files.list
│   ├── ./SPI/top.sv
│   ├── ./SPI/vsim.wlf
│   └── ./SPI/work
```



```
├── ./SPI Wrapper
│   ├── ./SPI Wrapper/certe_dump.xml
│   ├── ./SPI Wrapper/run.do
│   ├── ./SPI Wrapper/SPI_wrapper.v
│   ├── ./SPI Wrapper/src_files.list
│   ├── ./SPI Wrapper/top.sv
│   ├── ./SPI Wrapper/top.ucdb
│   ├── ./SPI Wrapper/vsim_stacktrace.vstf
│   ├── ./SPI Wrapper/vsim.wlf
│   ├── ./SPI Wrapper/work
│   ├── ./SPI Wrapper/WRAPPER_agent.sv
│   ├── ./SPI Wrapper/WRAPPER_config_obj.sv
│   ├── ./SPI Wrapper/WRAPPER_coverage.sv
│   ├── ./SPI Wrapper/WRAPPER_driver.sv
│   ├── ./SPI Wrapper/WRAPPER_env.sv
│   ├── ./SPI Wrapper/WRAPPER_if.sv
│   ├── ./SPI Wrapper/WRAPPER_mon.sv
│   ├── ./SPI Wrapper/WRAPPER_read_only_seq.sv
│   ├── ./SPI Wrapper/WRAPPER_reset_seq.sv
│   ├── ./SPI Wrapper/WRAPPER_score.sv
│   ├── ./SPI Wrapper/WRAPPER_seq_item.sv
│   ├── ./SPI Wrapper/WRAPPER_sequencer.sv
│   ├── ./SPI Wrapper/WRAPPER_shared_pkg.sv
│   ├── ./SPI Wrapper/WRAPPER_sva.sv
│   ├── ./SPI Wrapper/WRAPPER_test.sv
│   ├── ./SPI Wrapper/WRAPPER_write_only_seq.sv
│   └── ./SPI Wrapper/WRAPPER_write_read_seq.sv
└── ./transcript
```

5 UVM_Structure



Testbench Workflow

1. **Top Module**: Sets three configuration objects for WRAPPER, RAM, and SPI interfaces
2. **UVM Test**: Receives configuration objects and specifies active/passive mode for each component
3. **Environment**: Configuration objects are set in the environment via UVM configuration database
4. **Agents**: Receive configuration objects and propagate them to monitors, drivers, and sequencers
5. **Active Agent**: Drives randomized sequence items through the sequencer-driver pipeline
6. **Passive Components**: Monitors collect coverage and send transactions to scoreboard for checking

Component Hierarchy

- `WRAPPER_test` (UVM Test)

  - `WRAPPER_env` (Environment)
    - `WRAPPER_agent` (Active Agent)
    - `WRAPPER_scoreboard` (Scoreboard)
    - `WRAPPER_coverage` (Coverage Collector)
  - `RAM_env` (Environment - Passive)
  - `SPI_env` (Environment - Passive)

Configuration Management

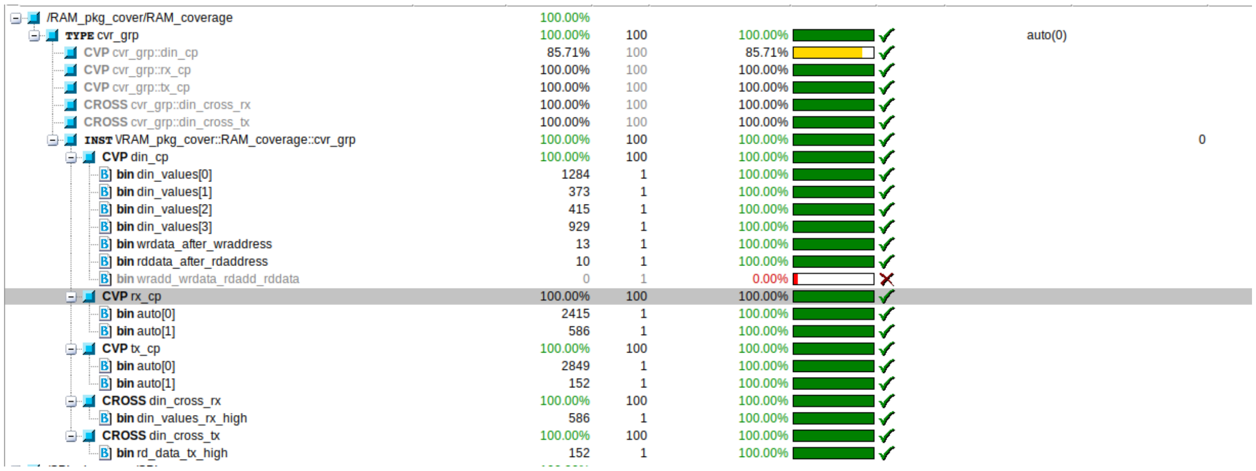
- WRAPPER_config_obj: Active configuration
- RAM_config_obj: Passive configuration
- SPI_config_obj: Passive configuration
- Virtual interfaces passed via UVM configuration database

Sequence Execution Flow

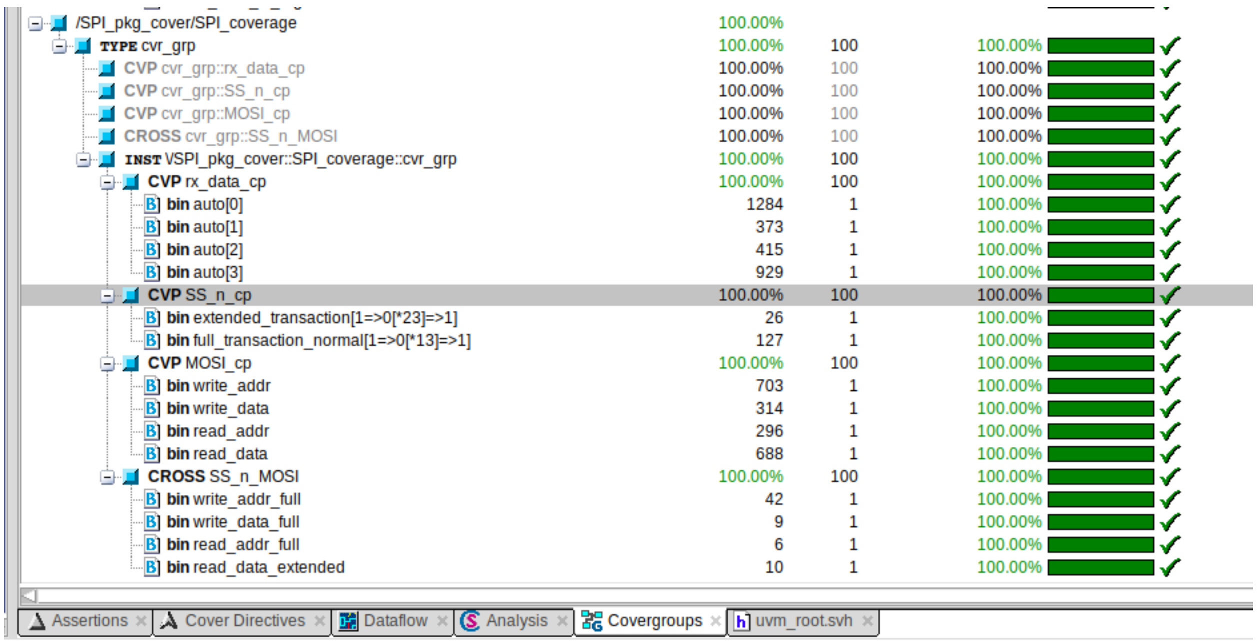
1. Reset Sequence
2. Write-Only Sequence
3. Read-Only Sequence
4. Write-Read Random Sequence

6 Covergroups

Note: the excluded coverpoint only works in when running RAM env, since the sequence of write read doesn't happen each cycle but are seperated by 13 cycles.

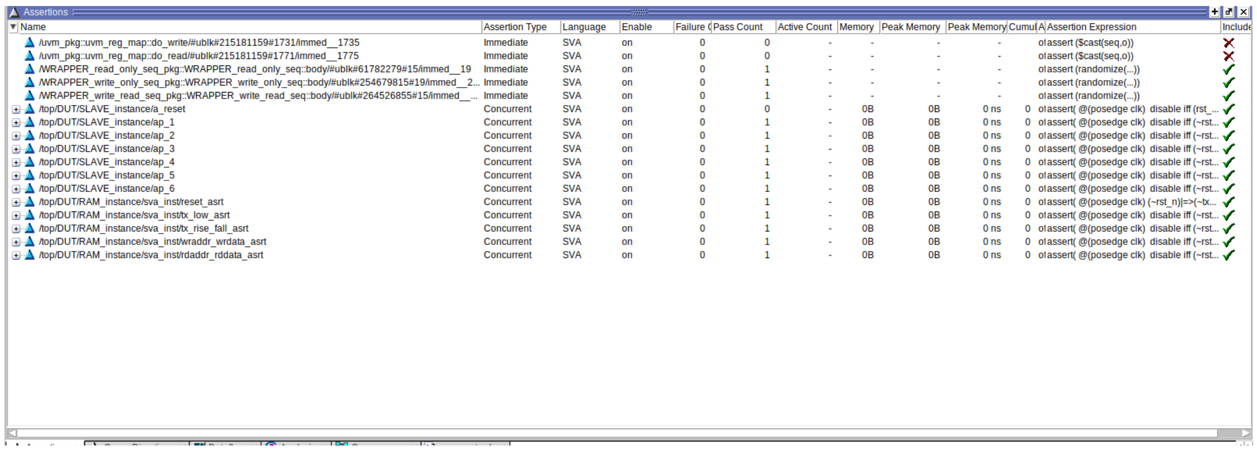


```
/RAM_pkg_cover/RAM_coverage                      100.00%
  TYPE cvr_grp                                   100.00%   100   100.00%  ✓   auto(0)
    CVP cvr_grp::din_cp                           85.71%   100    85.71%  ✓
    CVP cvr_grp::rx_cp                           100.00%   100   100.00%  ✓
    CVP cvr_grp::tx_cp                           100.00%   100   100.00%  ✓
    CROSS cvr_grp::din_cross_rx                  100.00%   100   100.00%  ✓
    CROSS cvr_grp::din_cross_tx                  100.00%   100   100.00%  ✓
    INST VRAM_pkg_cover::RAM_coverage::cvr_grp   100.00%   100   100.00%  ✓           0
      CVP din_cp                                 100.00%   100   100.00%  ✓
        bin din_values[0]                           1284     1   100.00%  ✓
        bin din_values[1]                            373     1   100.00%  ✓
        bin din_values[2]                            415     1   100.00%  ✓
        bin din_values[3]                            929     1   100.00%  ✓
        bin wrdata_after_wraddress                    13     1   100.00%  ✓
        bin rddata_after_rdaddress                    10     1   100.00%  ✓
        bin wradd_wrdata_rdadd_rddata                  0     1     0.00%  ✗
      CVP rx_cp                                  100.00%   100   100.00%  ✓
        bin auto[0]                                 2415     1   100.00%  ✓
        bin auto[1]                                  586     1   100.00%  ✓
      CVP tx_cp                                  100.00%   100   100.00%  ✓
        bin auto[0]                                 2849     1   100.00%  ✓
        bin auto[1]                                  152     1   100.00%  ✓
      CROSS din_cross_rx                         100.00%   100   100.00%  ✓
        bin din_values_rx_high                       586     1   100.00%  ✓
      CROSS din_cross_tx                         100.00%   100   100.00%  ✓
        bin rd_data_tx_high                          152     1   100.00%  ✓
```



/SPI_pkg_cover/SPI_coverage	100.00%				
TYPE cvr_grp	100.00%	100	100.00%	🟩	✓
CVP cvr_grp::rx_data_cp	100.00%	100	100.00%	🟩	✓
CVP cvr_grp::SS_n_cp	100.00%	100	100.00%	🟩	✓
CVP cvr_grp::MOSI_cp	100.00%	100	100.00%	🟩	✓
CROSS cvr_grp::SS_n_MOSI	100.00%	100	100.00%	🟩	✓
INST VSPI_pkg_cover::SPI_coverage::cvr_grp	100.00%	100	100.00%	🟩	✓
CVP rx_data_cp	100.00%	100	100.00%	🟩	✓
bin auto[0]	1284	1	100.00%	🟩	✓
bin auto[1]	373	1	100.00%	🟩	✓
bin auto[2]	415	1	100.00%	🟩	✓
bin auto[3]	929	1	100.00%	🟩	✓
CVP SS_n_cp	100.00%	100	100.00%	🟩	✓
bin extended_transaction[1=>0[*23]=>1]	26	1	100.00%	🟩	✓
bin full_transaction_normal[1=>0[*13]=>1]	127	1	100.00%	🟩	✓
CVP MOSI_cp	100.00%	100	100.00%	🟩	✓
bin write_addr	703	1	100.00%	🟩	✓
bin write_data	314	1	100.00%	🟩	✓
bin read_addr	296	1	100.00%	🟩	✓
bin read_data	688	1	100.00%	🟩	✓
CROSS SS_n_MOSI	100.00%	100	100.00%	🟩	✓
bin write_addr_full	42	1	100.00%	🟩	✓
bin write_data_full	9	1	100.00%	🟩	✓
bin read_addr_full	6	1	100.00%	🟩	✓
bin read_data_extended	10	1	100.00%	🟩	✓

7 Assertions



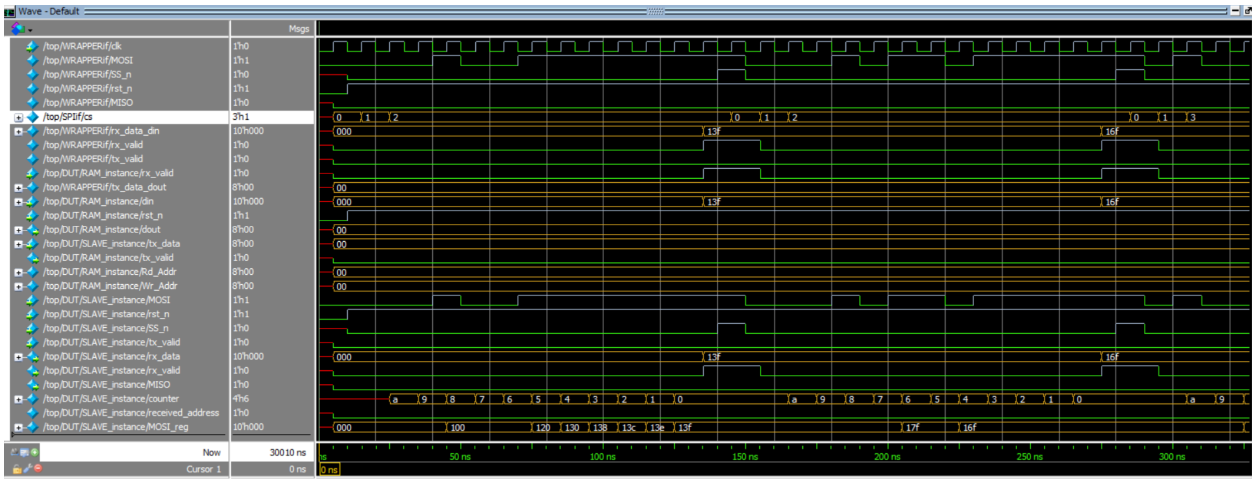
Assertions

Name	Assertion Type	Language	Enable	Failure (	Pass Count	Active Count	Memory	Peak Memory	Peak Memory	Cumul	A	Assertion Expression	Include
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0	-	-	-	-		o	assert ($cast(seq,o))	✗
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0	-	-	-	-		o	assert ($cast(seq,o))	✗
/WRAPPER_read_only_seq_pkg::WRAPPER_read_only_seq::body/#ublk#61782279#15/immed__19	Immediate	SVA	on	0	1	-	-	-	-		o	assert (randomize(...))	✓
/WRAPPER_write_only_seq_pkg::WRAPPER_write_only_seq::body/#ublk#254679815#19/immed__2...	Immediate	SVA	on	0	1	-	-	-	-		o	assert (randomize(...))	✓
/WRAPPER_write_read_seq_pkg::WRAPPER_write_read_seq::body/#ublk#264526855#15/immed__...	Immediate	SVA	on	0	1	-	-	-	-		o	assert (randomize(...))	✓
/top/DUT/SLAVE_instance/a_reset	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0	o	assert(@(posedge clk) disable iff (rst_...	✓
/top/DUT/SLAVE_instance/ap_1	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	o	assert(@(posedge clk) disable iff (~rst...	✓
/top/DUT/SLAVE_instance/ap_2	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	o	assert(@(posedge clk) disable iff (~rst...	✓
/top/DUT/SLAVE_instance/ap_3	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	o	assert(@(posedge clk) disable iff (~rst...	✓
/top/DUT/SLAVE_instance/ap_4	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	o	assert(@(posedge clk) disable iff (~rst...	✓
/top/DUT/SLAVE_instance/ap_5	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	o	assert(@(posedge clk) disable iff (~rst...	✓
/top/DUT/SLAVE_instance/ap_6	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	o	assert(@(posedge clk) disable iff (~rst...	✓
/top/DUT/RAM_instance/sva_inst/reset_asrt	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	o	assert(@(posedge clk) (~rst_n)|=>(~tx...	✓
/top/DUT/RAM_instance/sva_inst/tx_low_asrt	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	o	assert(@(posedge clk) disable iff (~rst...	✓
/top/DUT/RAM_instance/sva_inst/tx_rise_fall_asrt	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	o	assert(@(posedge clk) disable iff (~rst...	✓
/top/DUT/RAM_instance/sva_inst/wraddr_wrdata_asrt	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	o	assert(@(posedge clk) disable iff (~rst...	✓
/top/DUT/RAM_instance/sva_inst/rdaddr_rddata_asrt	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	o	assert(@(posedge clk) disable iff (~rst...	✓

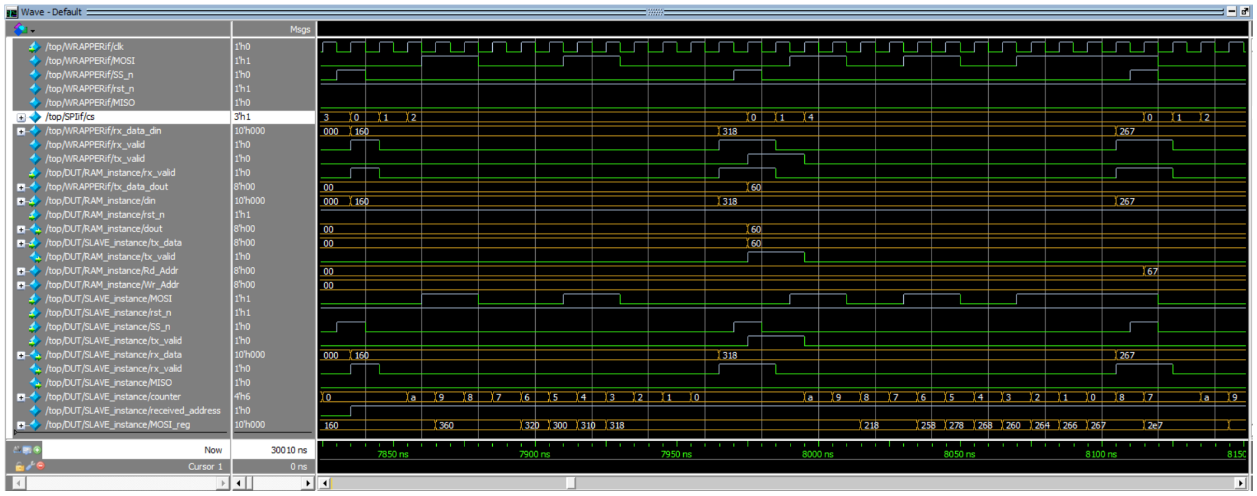
Table 1: SystemVerilog Assertions

Feature	**Assertion**
Reset: MISO low	@(posedge clk) (!rst_n |=> MISO)
Reset: rx_valid low	@(posedge clk) (!rst_n |=> rx_valid)
Reset: tx_valid low	@(posedge clk) (!rst_n |=> tx_valid)
Reset: rx_data zero	@(posedge clk) (!rst_n |=> rx_data == 0)
Command sequence: rx_valid after 10 cycles	@(posedge clk) (cs == CHK_CMD 1 MISO[*3]) |-> 10 (rx_valid && $rose(SS_n))
IDLE to CHK_CMD	@(posedge clk) (cs == IDLE && SS_n) |=> (cs == CHK_CMD)
CHK_CMD to WRITE/READ	@(posedge clk) (cs == CHK_CMD && SS_n) |=> (cs == WRITE || cs == READ_ADD || cs == READ_DATA)
WRITE to IDLE	@(posedge clk) (cs == WRITE && SS_n) |-> [1:22] (cs == IDLE)
READ_ADD to IDLE	@(posedge clk) (cs == READ_ADD && SS_n) |-> [1:22] (cs == IDLE)
READ_DATA to IDLE	@(posedge clk) (cs == READ_DATA && SS_n) |-> [1:22] (cs == IDLE)
tx_valid deasserted during input	@(posedge clk) (din[9:8] != 2'b11) |=> (tx_valid == 0)
tx_valid pulse after read data	@(posedge clk) (din[9:8] == 2'b11) |=> ($rose(tx_valid) 1 $fell(tx_valid))
Write Address followed by Write Data	@(posedge clk) (din[9:8] == 2'b00) |=> [1:5] (din[9:8] == 2'b01)
Read Address followed by Read Data	@(posedge clk) (din[9:8] == 2'b10) |=> [1:5] (din[9:8] == 2'b11)

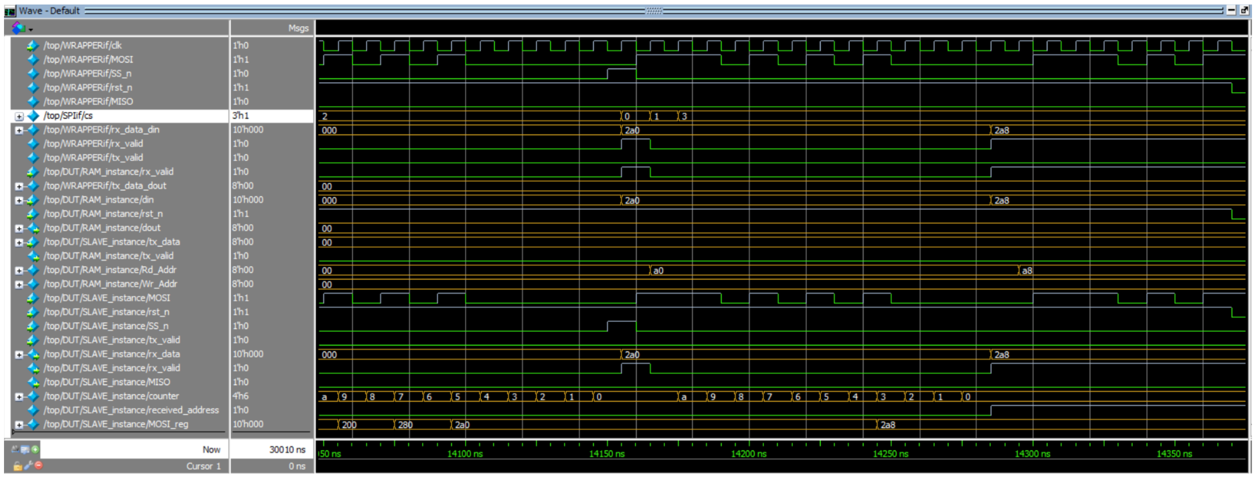
8 Waveforms



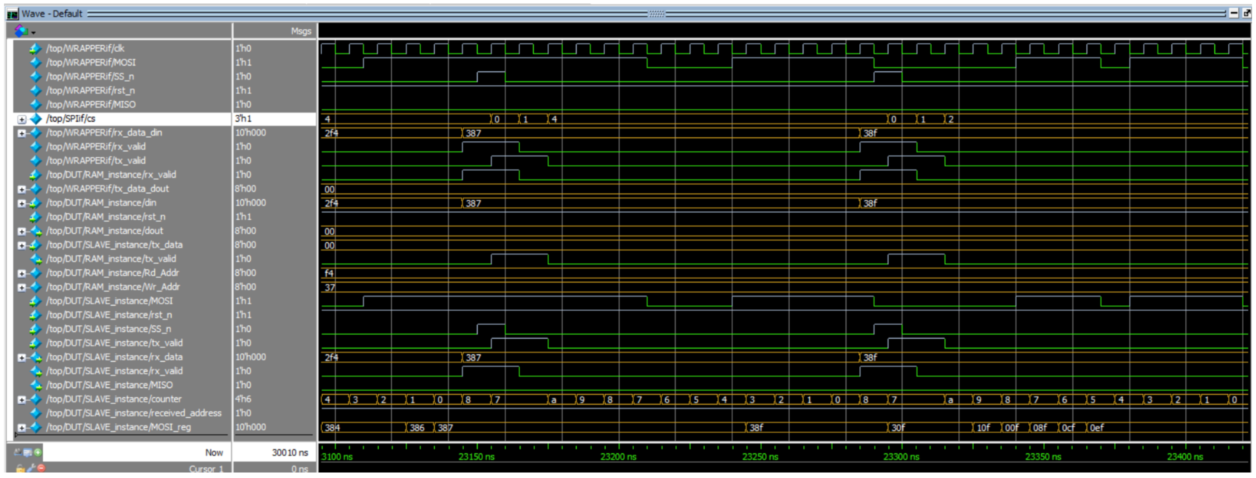
Wave - Default
Msgs
/top/WRAPPERif/clk 1'h0
/top/WRAPPERif/MOSI 1'h1
/top/WRAPPERif/SS_n 1'h0
/top/WRAPPERif/rst_n 1'h1
/top/WRAPPERif/MISO 1'h0
/top/SPIif/cs 3'h1
/top/WRAPPERif/rx_data_din 10'h000
/top/WRAPPERif/rx_valid 1'h0
/top/WRAPPERif/tx_valid 1'h0
/top/DUT/RAM_instance/rx_valid 1'h0
/top/WRAPPERif/tx_data_dout 8'h00
/top/DUT/RAM_instance/din 10'h000
/top/DUT/RAM_instance/rst_n 1'h1
/top/DUT/RAM_instance/dout 8'h00
/top/DUT/SLAVE_instance/tx_data 8'h00
/top/DUT/RAM_instance/tx_valid 1'h0
/top/DUT/RAM_instance/Rd_Addr 8'h00
/top/DUT/RAM_instance/Wr_Addr 8'h00
/top/DUT/SLAVE_instance/MOSI 1'h1
/top/DUT/SLAVE_instance/rst_n 1'h1
/top/DUT/SLAVE_instance/SS_n 1'h0
/top/DUT/SLAVE_instance/tx_valid 1'h0
/top/DUT/SLAVE_instance/rx_data 10'h000
/top/DUT/SLAVE_instance/rx_valid 1'h0
/top/DUT/SLAVE_instance/MISO 1'h0
/top/DUT/SLAVE_instance/counter 4'h6
/top/DUT/SLAVE_instance/received_address 1'h0
/top/DUT/SLAVE_instance/MOSI_reg 10'h000
Now 30010 ns
Cursor 1 0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

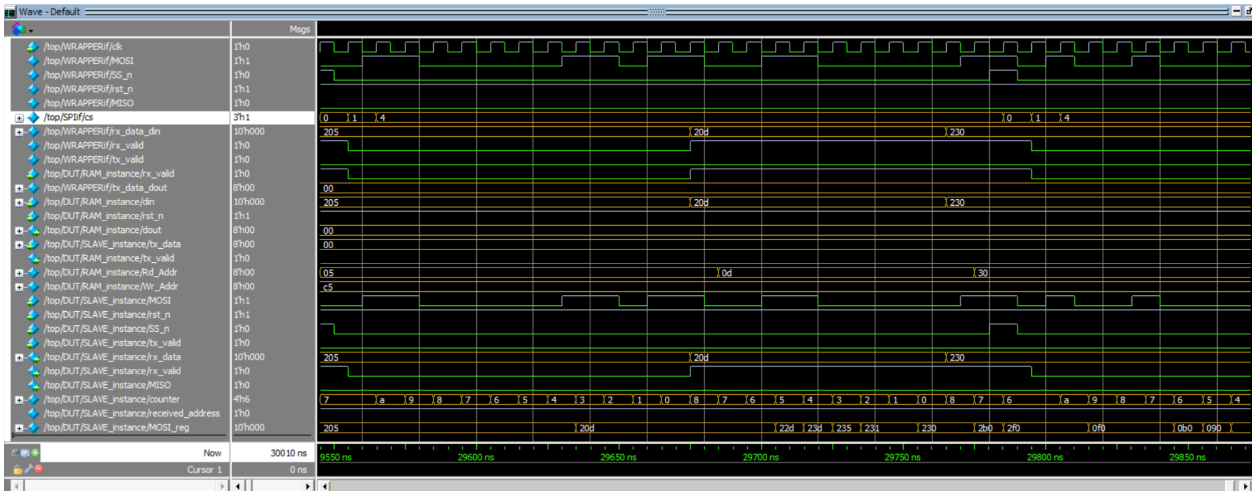


Wave - Default
Msgs
/top/WRAPPERif/clk 1'h0
/top/WRAPPERif/MOSI 1'h1
/top/WRAPPERif/SS_n 1'h0
/top/WRAPPERif/rst_n 1'h1
/top/WRAPPERif/MISO 1'h0
/top/SPIif/cs 3'h1
/top/WRAPPERif/rx_data_din 10'h000
/top/WRAPPERif/rx_valid 1'h0
/top/WRAPPERif/tx_valid 1'h0
/top/DUT/RAM_instance/rx_valid 1'h0
/top/WRAPPERif/tx_data_dout 8'h00
/top/DUT/RAM_instance/din 10'h000
/top/DUT/RAM_instance/rst_n 1'h1
/top/DUT/RAM_instance/dout 8'h00
/top/DUT/SLAVE_instance/tx_data 8'h00
/top/DUT/RAM_instance/tx_valid 1'h0
/top/DUT/RAM_instance/Rd_Addr 8'h00
/top/DUT/RAM_instance/Wr_Addr 8'h00
/top/DUT/SLAVE_instance/MOSI 1'h1
/top/DUT/SLAVE_instance/rst_n 1'h1
/top/DUT/SLAVE_instance/SS_n 1'h0
/top/DUT/SLAVE_instance/tx_valid 1'h0
/top/DUT/SLAVE_instance/rx_data 10'h000
/top/DUT/SLAVE_instance/rx_valid 1'h0
/top/DUT/SLAVE_instance/MISO 1'h0
/top/DUT/SLAVE_instance/counter 4'h6
/top/DUT/SLAVE_instance/received_address 1'h0
/top/DUT/SLAVE_instance/MOSI_reg 10'h000
Now 30010 ns
Cursor 1 0 ns
7850 ns
7900 ns
7950 ns
8000 ns
8050 ns
8100 ns

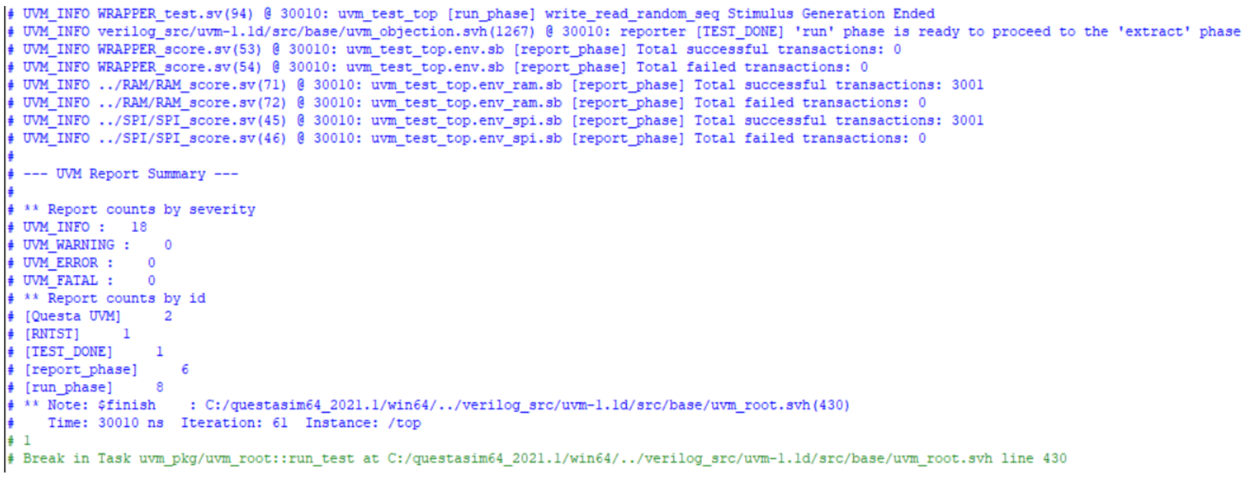


Wave - Default
Msgs
/top/WRAPPERif/clk 1'h0
/top/WRAPPERif/MOSI 1'h1
/top/WRAPPERif/SS_n 1'h0
/top/WRAPPERif/rst_n 1'h1
/top/WRAPPERif/MISO 1'h0
/top/SPIif/cs 3'h1
/top/WRAPPERif/rx_data_din 10'h000
/top/WRAPPERif/rx_valid 1'h0
/top/WRAPPERif/tx_valid 1'h0
/top/DUT/RAM_instance/rx_valid 1'h0
/top/WRAPPERif/tx_data_dout 8'h00
/top/DUT/RAM_instance/din 10'h000
/top/DUT/RAM_instance/rst_n 1'h1
/top/DUT/RAM_instance/dout 8'h00
/top/DUT/SLAVE_instance/tx_data 8'h00
/top/DUT/RAM_instance/tx_valid 1'h0
/top/DUT/RAM_instance/Rd_Addr 8'h00
/top/DUT/RAM_instance/Wr_Addr 8'h00
/top/DUT/SLAVE_instance/MOSI 1'h1
/top/DUT/SLAVE_instance/rst_n 1'h1
/top/DUT/SLAVE_instance/SS_n 1'h0
/top/DUT/SLAVE_instance/tx_valid 1'h0
/top/DUT/SLAVE_instance/rx_data 10'h000
/top/DUT/SLAVE_instance/rx_valid 1'h0
/top/DUT/SLAVE_instance/MISO 1'h0
/top/DUT/SLAVE_instance/counter 4'h6
/top/DUT/SLAVE_instance/received_address 1'h0
/top/DUT/SLAVE_instance/MOSI_reg 10'h000
Now 30010 ns
Cursor 1 0 ns
14100 ns
14150 ns
14200 ns
14250 ns
14300 ns
14350 ns


Wave - Default
Msgs
/top/WRAPPERif/clk
/top/WRAPPERif/MOSI
/top/WRAPPERif/SS_n
/top/WRAPPERif/rst_n
/top/WRAPPERif/MISO
/top/SPIif/cs
/top/WRAPPERif/rx_data_din
/top/WRAPPERif/rx_valid
/top/WRAPPERif/tx_valid
/top/DUT/RAM_instance/rx_valid
/top/WRAPPERif/tx_data_dout
/top/DUT/RAM_instance/din
/top/DUT/RAM_instance/rst_n
/top/DUT/RAM_instance/dout
/top/DUT/SLAVE_instance/tx_data
/top/DUT/RAM_instance/tx_valid
/top/DUT/RAM_instance/Rd_Addr
/top/DUT/RAM_instance/Wr_Addr
/top/DUT/SLAVE_instance/MOSI
/top/DUT/SLAVE_instance/rst_n
/top/DUT/SLAVE_instance/SS_n
/top/DUT/SLAVE_instance/tx_valid
/top/DUT/SLAVE_instance/rx_data
/top/DUT/SLAVE_instance/rx_valid
/top/DUT/SLAVE_instance/MISO
/top/DUT/SLAVE_instance/counter
/top/DUT/SLAVE_instance/received_address
/top/DUT/SLAVE_instance/MOSI_reg
Now
30010 ns
Cursor 1
0 ns
23100 ns
23150 ns
23200 ns
23250 ns
23300 ns
23350 ns
23400 ns



9 Transcript



```
# UVM_INFO WRAPPER_test.sv(94) @ 30010: uvm_test_top [run_phase] write_read_random_seq Stimulus Generation Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 30010: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO WRAPPER_score.sv(53) @ 30010: uvm_test_top.env.sb [report_phase] Total successful transactions: 0
# UVM_INFO WRAPPER_score.sv(54) @ 30010: uvm_test_top.env.sb [report_phase] Total failed transactions: 0
# UVM_INFO ../RAM/RAM_score.sv(71) @ 30010: uvm_test_top.env_ram.sb [report_phase] Total successful transactions: 3001
# UVM_INFO ../RAM/RAM_score.sv(72) @ 30010: uvm_test_top.env_ram.sb [report_phase] Total failed transactions: 0
# UVM_INFO ../SPI/SPI_score.sv(45) @ 30010: uvm_test_top.env_spi.sb [report_phase] Total successful transactions: 3001
# UVM_INFO ../SPI/SPI_score.sv(46) @ 30010: uvm_test_top.env_spi.sb [report_phase] Total failed transactions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   18
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     6
# [run_phase]     8
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 30010 ns  Iteration: 61  Instance: /top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```