REPORT

UVM Project SPI

TEAM #21

PREPARED BY
Nouran Hamdy

SUBMITTED TO
Eng Kareem Waseem

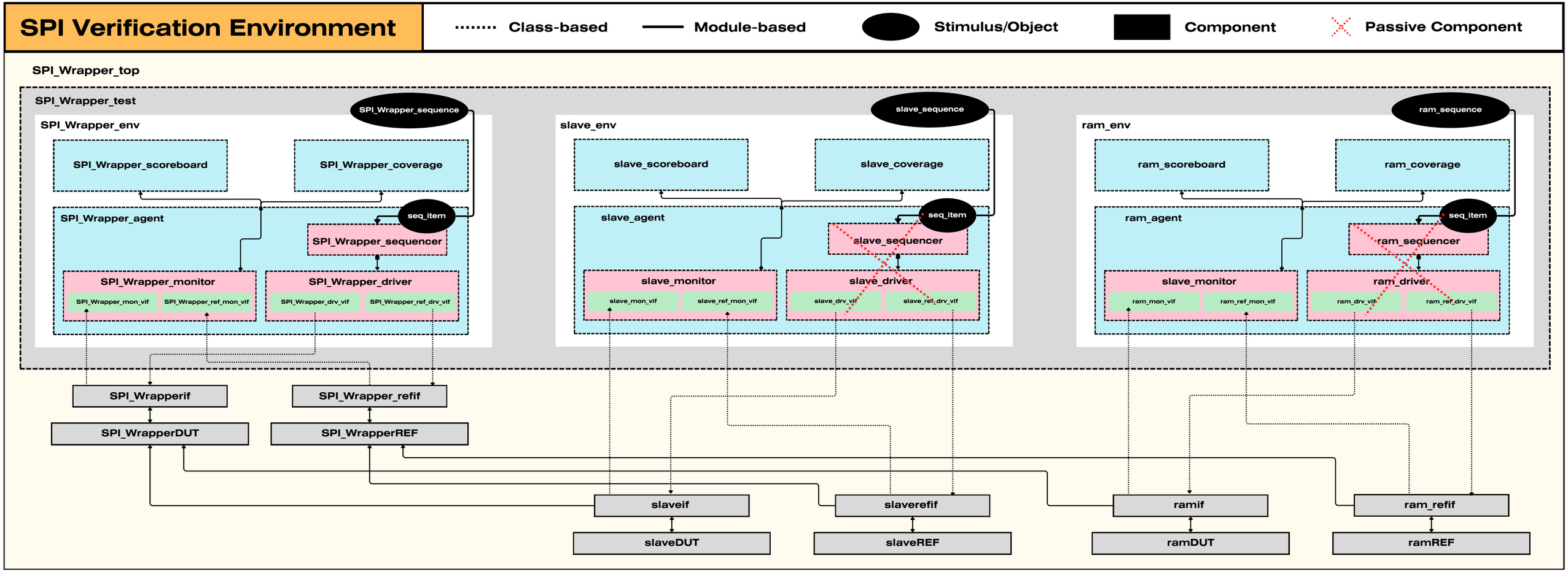

SPI Verification Environment
Class-based
Module-based
Stimulus/Object
Component
Passive Component
SPI_Wrapper_top
SPI_Wrapper_test
SPI_Wrapper_env
SPI_Wrapper_sequence
SPI_Wrapper_scoreboard
SPI_Wrapper_coverage
SPI_Wrapper_agent
seq_item
SPI_Wrapper_sequencer
SPI_Wrapper_monitor
SPI_Wrapper_mon_vif
SPI_Wrapper_ref_mon_vif
SPI_Wrapper_driver
SPI_Wrapper_drv_vif
SPI_Wrapper_ref_drv_vif
slave_env
slave_sequence
slave_scoreboard
slave_coverage
slave_agent
seq_item
slave_sequencer
slave_monitor
slave_mon_vif
slave_ref_mon_vif
slave_driver
slave_drv_vif
slave_ref_drv_vif
ram_env
ram_sequence
ram_scoreboard
ram_coverage
ram_agent
seq_item
ram_sequencer
slave_monitor
ram_mon_vif
ram_ref_mon_vif
ram_driver
ram_drv_vif
ram_ref_drv_vif
SPI_Wrapperif
SPI_Wrapper_refif
SPI_WrapperDUT
SPI_WrapperREF
slaveif
slaverefif
ramif
ram_refif
slaveDUT
slaveREF
ramDUT
ramREF

TABLE OF CONTENTS

Verification Plan

GENERAL PLAN:

• Checking Results

- Outputs' values are checked against golden model.
- Assertions are used for checking RAM behavior.

• Test Cases to Consider

- Reset Signal (rst_n) Functionality across any combination of inputs.
- Write operations in RAM – Filling up RAM with initial data.
- Read operations from RAM – Reading available data in RAM.
- Writing into, then reading from different RAM addresses.
- Completely randomized inputs to check for any unexpected behavior.

• General Constraints (For Good Usage of RAM)

- If previous signal = 2'b00 -> Next signal has to be 2'b01 (Write data in the address given previously).
- If previous signal = 2'b10 -> Next signal has to be 2'b11 (Read data from the address given previously).

• Functional Coverage Main Cover points and Cross Coverage

- First 2bits of din (Signal) has taken, and transitioned between, all values.
- Last 8bits of din (Data) has taken corner values.
- All RAM addresses have been exercised.
- Data has taken corner values with each state and transition of Signal.
- All states and transitions occurred when rx_valid was active.
- All RAM addresses have gone through all states of Signal.

• Assertions (Specific Test Cases)

- Stable cases (IDLE behavior) through the effect of control signals (rst_n and rx_valid)
- Normal Operation of RAM through different Signals (1st 2bits of din).
- tx_valid behavior, when it must (and must not) be active.

Verification Requirements

Test Item	Description and Expected Output Behavior	Stimulus Generation
Reset Signal (rst_n)	RAM outputs (dout and tx_valid) should reset to zero on reset assertion.	rst_n = 0. All other inputs are randomized under no constraints.
Control Signal (rx_valid)	RAM should accept din if rx_valid is active. RAM contents should be stable if rx_valid is inactive.	Included in all sequences where rx_valid is either active or randomized to be active most of the time.
Write_Only Opertation	rx_valid must be high for receiving an address or for a write operation to occur. Signal must be 2'b00 for saving a write address. Signal must be 2'b01 for writing data into the pre-given address. Signals (2'b00 and 2'b01) should be consecutive for a correct write operation.	rx_valid = 1. rst_n is constrained to be mostly inactive. First 2 bits of din (Signal) are randomized to be always 2'b00 or 2'b01 for writing operations only. Signal is constrained to follow any 2'b00 by 2'b01 (Signal should equal 2'b01 if the previous signal was 2'b00). Last 8 bits of din (Data) are constrained to occasionally take values of corners (All Ones, Zero, Walking Ones).
Read_Only Operation	rx_valid must be high for receiving an address or for a read operation to occur. Signal must be 2'b10 for saving a read address. Signal must be 2'b11 for reading data from the pre-given address. Signals (2'b10 and 2'b11) should be consecutive for a correct read operation.	rx_valid = 1. rst_n is constrained to be mostly inactive. First 2 bits of din (Signal) are randomized to be always 2'b10 or 2'b11 for reading operations only. Signal is constrained to follow any 2'b10 by 2'b11 (Signal should equal 2'b11 if the previous signal was 2'b10). Last 8 bits of din (Data) are constrained to occasionally take values of corners (All Ones, Zero, Walking Ones).
Normal RAM Operation (Writing and Reading from various addresses)	Same as (Write_Only Operation) and (Read_Only Operation) Description.	rst_n is randomized and constrained to be mostly inactive. rx_valid is randomized and constrained to be mostly active. Signal is constrained to follow any 2'b00 by 2'b01 (Signal should equal 2'b01 if the previous signal was 2'b00). Signal is constrained to follow any 2'b10 by 2'b11 (Signal should equal 2'b11 if the previous signal was 2'b10). Last 8 bits of din (Data) are constrained to occasionally take values of corners (All Ones, Zero, Walking Ones).
Main Sequence	Same as (Write_Only Operation) and (Read_Only Operation) Description.	All inputs are randomized under no constraints to check for unexpected behavior through different inputs combinations.

Verification Requirements

Test Item	Functional Coverage	Functionality Check	Assertions
Reset Signal (rst_n)			Property: reset_asserted
Control Signal (rx_valid)	Included in Coverpoints: signal_cp: Signal has taken, and transitioned between, all values (when rx_valid was active). data_cp: Data has taken corner values (when rx_valid was active).		Property: rx_valid_inactive
Write_Only Opertation	Included in Coverpoints: Signal_cp.WR_states: Signal has taken values needed for writing operation. data_cp: Data has taken corner values. Included in Cross Coverage: cross_signal_data.WR_data: All corners of data came along Write States. All RAM addresses have been exercised and data written into them has taken all combinations ("STORE_WR_ADDR" crossed with data_cp). ("WRITE_DATA" crossed with data_cp).		Property: save_write_addr: Checks Saving Write Address into write_addr (internal signal). wr_data: Checks if data is written in RAM at the required address.
Read_Only Operation	Included in Coverpoints: Signal_cp.RD_states: Signal has taken values needed for reading operation. data_cp: Data has taken corner values. Included in Cross Coverage: cross_signal_data.RD_data: All corners of data came along Read States. All RAM addresses have been exercised ("STORE_RD_ADDR" crossed with data_cp).	Output is checked against golden model.	Property: save_read_addr: Checks Saving Read Address into read_addr (internal signal). rd_data: Checks if the data read is the same as the data in RAM at the required address. tx_valid_active: Checks if tx_valid is active when data is ready. tx_valid_inactive: Checks tx_valid is not active when data is not ready.
Normal RAM Operation (Writing and Reading from various addresses)	Included in Coverpoints: Signal_cp: Signal has taken, and transitioned between, all values. data_cp: Data has taken corner values. Included in Cross Coverage: cross_signal_data: Data has taken corner values with each state and transition of Signal.		Refer to Write_Only and Read_Only assertions.
Main Sequence			

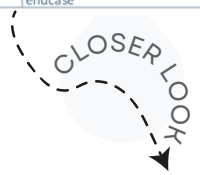
Bug Report

Bug	Original Code	Fix	Lines (Original Version)	Lines (Edited Version)
(Not a Bug) The design is converted to an sv file to access reg-defined internal signals from the sva file (mem, write_addr, read_addr) datatypes' are converted to logic instead of reg	reg [ADDR_SIZE-1:0] write_addr, read_addr; reg [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];	logic [ADDR_SIZE-1:0] write_addr, read_addr; logic [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];	[13:14]	[13:14]

Parameter (ADDR_SIZE) should be instead of the actual size to respond correctly to changes in parameters	case (din[9:8])
	write_addr <= din[7:0];
	mem [write_addr] <= din[7:0];
	read_addr <= din[7:0];
		case (din[ADDR_SIZE+1:ADDR_SIZE])
		write_addr <= din[ADDR_SIZE-1:0];
		mem [write_addr] <= din[ADDR_SIZE-1:0];
		read_addr <= din[ADDR_SIZE-1:0];
			29
			31
			35
			39
				65
				67
				71
				75

Memory elements are not supposed to reset to zero on each reset assertion, instead outputs and internal signals reset to zero	if(~rst_n) begin
	for (i=0; i < MEM_DEPTH; i=i+1) begin
	mem[i] = 0;
	end
	end
		if(~rst_n) begin
		dout <= 0;
		tx_valid <= 0;
		write_addr <= 0;
		read_addr <= 0;
		end
			[23:27]	[58:63]

(Not a Bug)
No default case
	case (din[9:8])
	2'b00: begin
	write_addr <= din[7:0];
	tx_valid <=0;
	end
	2'b01: begin
	mem [write_addr] <= din[7:0];
	tx_valid <=0;
	end
	2'b10: begin
	read_addr <= din[7:0];
	tx_valid <=0;
	end
	2'b11: begin
	dout <= mem[read_addr];
	tx_valid <=1;
	end
	endcase
		case (din[ADDR_SIZE+1:ADDR_SIZE])
		2'b00: begin
		write_addr <= din[ADDR_SIZE-1:0];
		tx_valid <=0;
		end
		2'b01: begin
		mem [write_addr] <= din[ADDR_SIZE-1:0];
		tx_valid <=0;
		end
		2'b10: begin
		read_addr <= din[ADDR_SIZE-1:0];
		tx_valid <=0;
		end
		default: begin
		dout <= mem[read_addr];
		tx_valid <=1;
		end
		endcase
			[29:46]	[65:82]



Bug	Original Code	Fix
(Not a Bug) The design is converted to an sv file to access reg-defined internal signals from the sva file (mem, write_addr, read_addr) datatypes' are converted to logic instead of reg	reg [ADDR_SIZE-1:0] write_addr, read_addr; reg [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];	logic [ADDR_SIZE-1:0] write_addr, read_addr; logic [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];

Parameter (ADDR_SIZE) should be instead of the actual size to respond correctly to changes in parameters	case (din[9:8])
	write_addr <= din[7:0];
	mem [write_addr] <= din[7:0];
	read_addr <= din[7:0];
		case (din[ADDR_SIZE+1:ADDR_SIZE])
		write_addr <= din[ADDR_SIZE-1:0];
		mem [write_addr] <= din[ADDR_SIZE-1:0];
		read_addr <= din[ADDR_SIZE-1:0];

Memory elements are not supposed to reset to zero on each reset assertion, instead outputs and internal signals reset to zero	if(~rst_n) begin
	for (i=0; i < MEM_DEPTH; i=i+1) begin
	mem[i] = 0;
	end
	end
		if(~rst_n) begin
		dout <= 0;
		tx_valid <= 0;
		write_addr <= 0;
		read_addr <= 0;
		end

(Not a Bug)
No default case
	case (din[9:8])
	2'b00: begin
	write_addr <= din[7:0];
	tx_valid <=0;
	end
	2'b01: begin
	mem [write_addr] <= din[7:0];
	tx_valid <=0;
	end
	2'b10: begin
	read_addr <= din[7:0];
	tx_valid <=0;
	end
	2'b11: begin
	dout <= mem[read_addr];
	tx_valid <=1;
	end
	endcase
		case (din[ADDR_SIZE+1:ADDR_SIZE])
		2'b00: begin
		write_addr <= din[ADDR_SIZE-1:0];
		tx_valid <=0;
		end
		2'b01: begin
		mem [write_addr] <= din[ADDR_SIZE-1:0];
		tx_valid <=0;
		end
		2'b10: begin
		read_addr <= din[ADDR_SIZE-1:0];
		tx_valid <=0;
		end
		default: begin
		dout <= mem[read_addr];
		tx_valid <=1;
		end
		endcase

Code Snippets

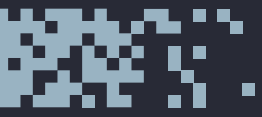
TOP MODULE

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

module ram_top;
    bit clk;

    initial begin
        forever #1 clk = ~clk;
    end

    ram_if ramif (clk);
    ram DUT (ramif);

    ram_ref_if ram_refif (clk);
    ram_ref REF (ram_refif);

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

endmodule : ram_top
```

INTERFACE

```
interface ram_if (clk);

    parameter MEM_DEPTH = 256;
    parameter ADDR_SIZE = 8;

    input clk;
    bit rst_n , rx_valid, tx_valid;
    logic [ADDR_SIZE-1:0] dout;
    logic [ADDR_SIZE+1:0] din;

    modport DUT (output tx_valid, dout,
                 input clk, rst_n, rx_valid, din);

endinterface : ram_if
```

Code Snippets



DESIGN

Before (Original)

```
17
18  ////////////////////////////////Original Code////////////////////////////////
19
20  /*
21      integer i=0;
22      always @(posedge clk,negedge rst_n) begin
23          if(~rst_n) begin
24              for (i=0; i < MEM_DEPTH; i=i+1) begin       //memory values should not equal zero following each reset, only Module outputs (and wr/rd addresses).
25                  mem[i] = 0;
26              end
27          end
28          else if(rx_valid) begin
29              case (din[9:8])                             //Should use parameter (ADDR_SIZE) instead of actual size
30                  2'b00: begin
31                      write_addr <= din[7:0];
32                      tx_valid <=0;
33                  end
34                  2'b01: begin
35                      mem [write_addr] <= din[7:0];
36                      tx_valid <=0;
37                  end
38                  2'b10: begin
39                      read_addr <= din[7:0];
40                      tx_valid <=0;
41                  end
42                  2'b11: begin                            //For Code Coverage = 100% -> This branch is moved to default
43                      dout <= mem[read_addr];
44                      tx_valid <=1;
45                  end
46              endcase
47          end
48          else
49              tx_valid =0;
50      end
51  */
```

After (Edited)

```
55  ////////////////////////////////Edited Code////////////////////////////////
56
57      always @(posedge clk,negedge rst_n) begin
58          if(~rst_n) begin
59              dout <= 0;
60              tx_valid <= 0;
61              write_addr <= 0;
62              read_addr <= 0;
63          end
64          else if(rx_valid) begin
65              case (din[ADDR_SIZE+1:ADDR_SIZE])
66                  2'b00: begin
67                      write_addr <= din[ADDR_SIZE-1:0];
68                      tx_valid <=0;
69                  end
70                  2'b01: begin
71                      mem [write_addr] <= din[ADDR_SIZE-1:0];
72                      tx_valid <=0;
73                  end
74                  2'b10: begin
75                      read_addr <= din[ADDR_SIZE-1:0];
76                      tx_valid <=0;
77                  end
78                  default: begin
79                      dout <= mem[read_addr];
80                      tx_valid <=1;
81                  end
82              endcase
83          end
84      end
```

Code Snippets

TEST

```
1   package ram_test_pkg;
2   import uvm_pkg::*;
3   import ram_env_pkg::*;
4   import ram_config_pkg::*;
5   import ram_sequence_pkg::*;
6   `include "uvm_macros.svh"
7
8       class ram_test extends uvm_test;
9           `uvm_component_utils(ram_test)
10
11          ram_env env_r;
12          ram_config ram_cfg;
13
14          ram_reset_sequence reset_seq;
15          ram_write_only_sequence write_only_seq;
16          ram_read_only_sequence read_only_seq;
17          ram_write_read_sequence write_read_seq;
18          main_sequence main_seq;
19
20          function new(string name = "ram_test", uvm_component parent = null);
21              super.new(name, parent);
22          endfunction : new
23
24          function void build_phase(uvm_phase phase);
25              super.build_phase(phase);
26              env_r = ram_env::type_id::create("env_r", this);
27              ram_cfg = ram_config::type_id::create("ram_cfg", this);
28
29              reset_seq       = ram_reset_sequence::type_id::create("reset_seq", this);
30              write_only_seq  = ram_write_only_sequence::type_id::create("write_only_seq", this);
31              read_only_seq   = ram_read_only_sequence::type_id::create("read_only_seq", this);
32              write_read_seq  = ram_write_read_sequence::type_id::create("write_read_seq", this);
33              main_seq        = main_sequence::type_id::create("main_seq", this);
34
35              ram_cfg.active = UVM_ACTIVE;
36
37              if(!uvm_config_db #(virtual ram_if)::get(this, "", "RAM_IF", ram_cfg.ram_vif))
38                  `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the RAM (DUT) from the uvm_config_db");
39
40              if(!uvm_config_db #(virtual ram_ref_if)::get(this, "", "RAM_REF_IF", ram_cfg.ram_ref_vif))
41                  `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the RAM (REF) from the uvm_config_db");
42
43              uvm_config_db #(ram_config)::set(this, "*", "RAM_CFG", ram_cfg);
44
45              //set_report_verbosity_level_hier(UVM_HIGH);
46          endfunction

48          task run_phase(uvm_phase phase);
49              super.run_phase(phase);
50              phase.raise_objection(this);
51              `uvm_info("run_phase", "Reset Asserted", UVM_LOW);
52              reset_seq.start(env_r.agt.sqr);
53              `uvm_info("run_phase", "Reset Deasserted", UVM_LOW);
54
55              `uvm_info("run_phase", "Stimulus Generation Started", UVM_LOW);
56
57              `uvm_info("run_phase", "Testing Write-Only Operations", UVM_LOW);
58              write_only_seq.start(env_r.agt.sqr);
59
60              `uvm_info("run_phase", "Testing Read-Only Operations", UVM_LOW);
61              read_only_seq.start(env_r.agt.sqr);
62
63              `uvm_info("run_phase", "Testing Write-Read Operations", UVM_LOW);
64              write_read_seq.start(env_r.agt.sqr);
65
66              `uvm_info("run_phase", "Main Sequence (Complete Randomization)", UVM_LOW);
67              write_read_seq.start(env_r.agt.sqr);
68
69              `uvm_info("run_phase", "Stimulus Generation Ended", UVM_LOW);
70              phase.drop_objection(this);
71          endtask : run_phase
72
73      endclass : ram_test
74
75  endpackage : ram_test_pkg
```

Code Snippets

CONFIGURATION OBJECT

```
1   package ram_config_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4
5       class ram_config extends uvm_object;
6           `uvm_object_utils(ram_config)
7
8           virtual ram_if ram_vif;
9           virtual ram_ref_if ram_ref_vif;
10          uvm_active_passive_enum active;
11
12          function new(string name = "ram_config");
13              super.new(name);
14          endfunction : new
15
16      endclass : ram_config
17
18  endpackage : ram_config_pkg
```

Code Snippets

ENVIRONMENT

```
1   package ram_env_pkg;
2   import uvm_pkg::*;
3   import ram_agent_pkg::*;
4   import ram_scoreboard_pkg::*;
5   import ram_coverage_pkg::*;
6   `include "uvm_macros.svh"
7
8       class ram_env extends uvm_env;
9           `uvm_component_utils(ram_env);
10
11          ram_agent agt;
12          ram_scoreboard sb;
13          ram_coverage cov;
14
15          function new(string name = "ram_env", uvm_component parent = null);
16              super.new(name, parent);
17          endfunction : new
18
19          function void build_phase(uvm_phase phase);
20              super.build_phase(phase);
21              agt = ram_agent::type_id::create("agt", this);
22              sb = ram_scoreboard::type_id::create("sb", this);
23              cov = ram_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 : connect_phase
31
32      endclass : ram_env
33  endpackage : ram_env_pkg
```

Code Snippets

AGENT

```
package ram_agent_pkg;
import uvm_pkg::*;
import ram_driver_pkg::*;
import ram_sequencer_pkg::*;
import ram_monitor_pkg::*;
import ram_config_pkg::*;
import ram_sequence_item_pkg::*;

`include "uvm_macros.svh"

    class ram_agent extends uvm_agent;
        `uvm_component_utils(ram_agent)

        ram_driver drv;
        ram_sequencer sqr;
        ram_monitor mon;
        ram_config ram_cfg;
        uvm_analysis_port #(ram_sequence_item) agt_ap;

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            ram_cfg = ram_config::type_id::create("ram_cfg", this);
            if(!uvm_config_db#(ram_config)::get(this, "", "RAM_CFG", ram_cfg))
                `uvm_fatal("build_phase", "Agent - Unable to get RAM Configuration Object.");

            if(ram_cfg.active) begin
                drv = ram_driver::type_id::create("drv", this);
                sqr = ram_sequencer::type_id::create("sqr", this);
            end
            mon = ram_monitor::type_id::create("mon", this);

            agt_ap = new("agt_ap", this);
        endfunction : build_phase

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            if(ram_cfg.active) begin
                drv.ram_drv_vif = ram_cfg.ram_vif;
                drv.ram_ref_drv_vif = ram_cfg.ram_ref_vif;
                drv.seq_item_port.connect(sqr.seq_item_export);
            end
            mon.ram_mon_vif = ram_cfg.ram_vif;
            mon.ram_ref_mon_vif = ram_cfg.ram_ref_vif;
            mon.mon_ap.connect(agt_ap);
        endfunction : connect_phase

    endclass : ram_agent

endpackage : ram_agent_pkg
```

Code Snippets

SEQUENCE ITEM

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

    class ram_sequence_item extends uvm_sequence_item;
        `uvm_object_utils(ram_sequence_item)

        rand bit rst_n, rx_valid;
        //rand logic [ADDR_SIZE+1:0] din;
        logic [ADDR_SIZE-1:0] dout;
        bit tx_valid;

        logic [ADDR_SIZE-1:0] dout_ref;
        bit tx_valid_ref;

        rand signal_e signal;
        rand logic [ADDR_SIZE-1:0] data;

        //For post randomization
        signal_e signal_old;

        //For specifying valid addresses (that have been written previously)
        //logic [ADDR_SIZE-1:0] valid_addr_q[$];

        rand bit [1:0] selector_data;

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

        function string convert2string();
            return $sformatf("%s    rst_n = 0b%0b,  rx_valid = 0b%0b,   din = 0x%0h,    tx_valid = 0b%0b,   dout = 0x%0h",
                             super.convert2string(), rst_n, rx_valid, {signal, data}, tx_valid, dout);
        endfunction : convert2string

        function string convert2string_stimulus();
            return $sformatf("rst_n = 0b%0b,    rx_valid = 0b%0b,   din = 0x%0h",
                             rst_n, rx_valid, {signal, data});
        endfunction : convert2string_stimulus


        //Constraints
        constraint c {
            rst_n           dist {0:=5,     1:=95};                     //rst_n is inactive most of the time
            rx_valid        dist {0:=30,    1:=70};                     //rx_valid is active most of the time

            selector_data   dist {[0:2]:/30, 3:=70};                    //Data is at its corner cases for some time

            if(selector_data == 0)          data == {ALL_ONES};
            else if(selector_data == 1)     data == {ZERO};
            else if(selector_data == 2)     $countones(data) == 1;

            //(signal == STORE_RD_ADDR)     -> data inside {valid_addr_q};
        }

        constraint write_op_c {
            (signal_old == STORE_WR_ADDR)   -> signal == WRITE_DATA;        //Whenever a "SAVE WR ADDRESS" signal is sent, it's followed by a "WRITE DATA" signal
        }

        constraint read_op_c {
            (signal_old == STORE_RD_ADDR)   -> signal == READ_DATA_;        //Whenever a "SAVE RD ADDRESS" signal is sent, it's followed by a "READ DATA" signal
        }

        function void post_randomize;
            //if(rst_n && rx_valid && (signal == STORE_WR_ADDR)) valid_addr_q.push_front(data);
            signal_old  = signal;
        endfunction


    endclass : ram_sequence_item

endpackage : ram_sequence_item_pkg
```

CODE SNIPPETS

SEQUENCE

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

parameter TESTS = 20000;

    class ram_reset_sequence extends uvm_sequence #(ram_sequence_item);
        `uvm_object_utils(ram_reset_sequence)

        ram_sequence_item seq_item;

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

        task body();
            seq_item = ram_sequence_item::type_id::create("seq_item");
            seq_item.constraint_mode(0);
            repeat(TESTS/10) begin
                start_item(seq_item);
                assert(seq_item.randomize() with {rst_n == 0;});
                finish_item(seq_item);
            end
        endtask : body

    endclass : ram_reset_sequence


    class ram_write_only_sequence extends uvm_sequence #(ram_sequence_item);
        `uvm_object_utils(ram_write_only_sequence)

        ram_sequence_item seq_item;

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

        task body();
            seq_item = ram_sequence_item::type_id::create("seq_item");
            seq_item.constraint_mode(1);
            seq_item.read_op_c.constraint_mode(0);
            repeat(TESTS) begin
                start_item(seq_item);
                assert(seq_item.randomize() with {signal inside {STORE_WR_ADDR, WRITE_DATA};  rx_valid == 1;});
                finish_item(seq_item);
            end
        endtask : body

    endclass : ram_write_only_sequence
```

Code Snippets

SEQUENCE

```
56      class ram_read_only_sequence extends uvm_sequence #(ram_sequence_item);
57          `uvm_object_utils(ram_read_only_sequence)
58
59          ram_sequence_item seq_item;
60
61          function new(string name = "ram_read_only_sequence");
62              super.new(name);
63          endfunction : new
64
65          task body();
66              seq_item = ram_sequence_item::type_id::create("seq_item");
67              seq_item.constraint_mode(1);
68              seq_item.write_op_c.constraint_mode(0);
69              repeat(TESTS) begin
70                  start_item(seq_item);
71                  assert(seq_item.randomize() with {signal inside {STORE_RD_ADDR, READ_DATA_};  rx_valid == 1;});
72                  finish_item(seq_item);
73              end
74          endtask : body
75
76      endclass : ram_read_only_sequence


80      class ram_write_read_sequence extends uvm_sequence #(ram_sequence_item);
81          `uvm_object_utils(ram_write_read_sequence)
82
83          ram_sequence_item seq_item;
84
85          function new(string name = "ram_write_read_sequence");
86              super.new(name);
87          endfunction : new
88
89          task body();
90              seq_item = ram_sequence_item::type_id::create("seq_item");
91              seq_item.constraint_mode(1);
92              repeat(TESTS) begin
93                  start_item(seq_item);
94                  assert(seq_item.randomize());
95                  finish_item(seq_item);
96              end
97          endtask : body
98
99      endclass : ram_write_read_sequence
100
101
102
103     class main_sequence extends uvm_sequence #(ram_sequence_item);
104         `uvm_object_utils(main_sequence)
105
106         ram_sequence_item seq_item;
107
108         function new(string name = "main_sequence");
109             super.new(name);
110         endfunction : new
111
112         task body();
113             seq_item = ram_sequence_item::type_id::create("seq_item");
114             seq_item.constraint_mode(0);
115             seq_item.c.constraint_mode(1);
116             repeat(TESTS) begin
117                 start_item(seq_item);
118                 assert(seq_item.randomize());
119                 finish_item(seq_item);
120             end
121         endtask : body
122
123     endclass : main_sequence
124
125 endpackage : ram_sequence_pkg
```

Code Snippets

SEQUENCER

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

Code Snippets

DRIVER

```
1   package ram_driver_pkg;
2   import ram_sequence_item_pkg::*;
3   import shared_pkg::*;
4   import uvm_pkg::*;
5   `include "uvm_macros.svh"
6
7       class ram_driver extends uvm_driver #(ram_sequence_item);
8           `uvm_component_utils(ram_driver)
9
10          virtual ram_if ram_drv_vif;
11          virtual ram_ref_if ram_ref_drv_vif;
12          ram_sequence_item stim_seq_item;
13
14          function new(string name = "ram_driver", uvm_component parent = null);
15              super.new(name, parent);
16          endfunction : new
17
18          task run_phase(uvm_phase phase);
19              super.run_phase(phase);
20              forever begin
21                  stim_seq_item = ram_sequence_item::type_id::create("stim_seq_item");
22                  seq_item_port.get_next_item(stim_seq_item);
23                  //DUT
24                  ram_drv_vif.rst_n                       = stim_seq_item.rst_n;
25                  ram_drv_vif.rx_valid                    = stim_seq_item.rx_valid;
26                  ram_drv_vif.din[ADDR_SIZE+1:ADDR_SIZE]  = stim_seq_item.signal;
27                  ram_drv_vif.din[ADDR_SIZE-1:0]          = stim_seq_item.data;
28
29                  //REF
30                  ram_ref_drv_vif.rst_n                       = stim_seq_item.rst_n;
31                  ram_ref_drv_vif.rx_valid                    = stim_seq_item.rx_valid;
32                  ram_ref_drv_vif.din[ADDR_SIZE+1:ADDR_SIZE]  = stim_seq_item.signal;
33                  ram_ref_drv_vif.din[ADDR_SIZE-1:0]          = stim_seq_item.data;
34
35                  @(negedge ram_drv_vif.clk);
36                  seq_item_port.item_done();
37                  `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH);
38              end
39          endtask : run_phase
40
41      endclass : ram_driver
42
43  endpackage : ram_driver_pkg
```

Code Snippets

MONITOR

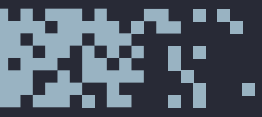
```
1   package ram_monitor_pkg;
2   import uvm_pkg::*;
3   import shared_pkg::*;
4   import ram_sequence_item_pkg::*;
5
6   `include "uvm_macros.svh"
7
8       class ram_monitor extends uvm_monitor;
9           `uvm_component_utils(ram_monitor)
10
11          ram_sequence_item rsp_seq_item;
12          virtual ram_if ram_mon_vif;
13          virtual ram_ref_if ram_ref_mon_vif;
14          uvm_analysis_port #(ram_sequence_item) mon_ap;
15
16          function new(string name = "ram_monitor", uvm_component parent = null);
17              super.new(name, parent);
18          endfunction : new
19
20          function void build_phase(uvm_phase phase);
21              super.build_phase(phase);
22              mon_ap = new("mon_ap", this);
23          endfunction : build_phase
24
25          task run_phase(uvm_phase phase);
26              super.run_phase(phase);
27
28              forever begin
29                  rsp_seq_item = ram_sequence_item::type_id::create("rsp_seq_item");
30                  @(negedge ram_mon_vif.clk);
31                  rsp_seq_item.rst_n          = ram_mon_vif.rst_n;
32                  rsp_seq_item.rx_valid       = ram_mon_vif.rx_valid;
33                  rsp_seq_item.tx_valid       = ram_mon_vif.tx_valid;
34                  rsp_seq_item.dout           = ram_mon_vif.dout;
35                  rsp_seq_item.signal         = signal_e'(ram_mon_vif.din[ADDR_SIZE+1:ADDR_SIZE]);
36                  rsp_seq_item.data           = ram_mon_vif.din[ADDR_SIZE-1:0];
37                  rsp_seq_item.dout_ref       = ram_ref_mon_vif.dout_ref;
38                  rsp_seq_item.tx_valid_ref   = ram_ref_mon_vif.tx_valid_ref;
39
40                  mon_ap.write(rsp_seq_item);
41                  `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH);
42              end
43          endtask : run_phase
44
45      endclass : ram_monitor
46
47  endpackage : ram_monitor_pkg
```

Code Snippets

SCOREBOARD

```
package ram_scoreboard_pkg;
import uvm_pkg::*;
import shared_pkg::*;
import ram_sequence_item_pkg::*;

`include "uvm_macros.svh"

    class ram_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(ram_scoreboard);

        ram_sequence_item sb_seq_item;
        uvm_analysis_export #(ram_sequence_item) sb_export;
        uvm_tlm_analysis_fifo #(ram_sequence_item) sb_fifo;

        int correct_count_dout, error_count_dout;
        int correct_count_tx, error_count_tx;


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

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

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


        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(sb_seq_item);

                if(sb_seq_item.dout !== sb_seq_item.dout_ref) begin
                    `uvm_error("run_phase", $sformatf("Comparison Failed, Transaction received by DUT: %s, while the reference output -dout-:0b%0b",
                    error_count_dout++;
                end
                else begin
                    `uvm_info("run_phase", $sformatf("Correct RAM output -dout-: %s", sb_seq_item.convert2string()), UVM_HIGH);
                    correct_count_dout++;
                end

                if(sb_seq_item.tx_valid !== sb_seq_item.tx_valid_ref) begin
                    `uvm_error("run_phase", $sformatf("Comparison Failed, Transaction received by DUT: %s, while the reference output -tx_valid-:0b%0
                    error_count_tx++;
                end
                else begin
                    `uvm_info("run_phase", $sformatf("Correct RAM output -tx_valid-: %s", sb_seq_item.convert2string()), UVM_HIGH);
                    correct_count_tx++;
                end
            end

        endtask : run_phase

        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("report_phase", $sformatf("Total Successful Transactions -dout-: %0d", correct_count_dout), UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("Total Failed Transactions -dout-: %0d", error_count_dout), UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("Total Successful Transactions -tx_valid-: %0d", correct_count_tx), UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("Total Failed Transactions -tx_valid-: %0d", error_count_tx), UVM_MEDIUM);
        endfunction : report_phase

    endclass : ram_scoreboard

endpackage : ram_scoreboard_pkg
```

Code Snippets



COVERAGE

```
package ram_coverage_pkg;
import uvm_pkg::*;
import shared_pkg::*;
import ram_sequence_item_pkg::*;

`include "uvm_macros.svh"

    class ram_coverage extends uvm_component;
        `uvm_component_utils(ram_coverage);

        uvm_analysis_export #(ram_sequence_item) cov_export;
        uvm_tlm_analysis_fifo #(ram_sequence_item) cov_fifo;

        ram_sequence_item cov_seq_item;

        covergroup cg;
            signal_cp:      coverpoint cov_seq_item.signal iff (cov_seq_item.rx_valid){
                bins WR_states[]    = {STORE_WR_ADDR, WRITE_DATA};
                bins RD_states[]    = {STORE_RD_ADDR, READ_DATA_};
                bins WR_to_RD       = (STORE_WR_ADDR => WRITE_DATA => STORE_RD_ADDR, READ_DATA_);   //Normal Operation: Read after Write
                bins RD_to_WR       = (STORE_RD_ADDR => READ_DATA_ => STORE_WR_ADDR, WRITE_DATA);   //Proposed Scenario: Write after Read
            }

            data_cp:        coverpoint cov_seq_item.data iff (cov_seq_item.rx_valid){              //Ensuring access to all bits of data and memory
                bins ALL_ones       = {ALL_ONES};
                bins ZERO           = {ZERO};
                bins Walking_ones   = {2**(ADDR_SIZE-1), 2**(ADDR_SIZE-2), 2**(ADDR_SIZE-3), 2**(ADDR_SIZE-4), 2**(ADDR_SIZE-5), 2**(ADDR_SIZE-6), 2**(ADDR_SIZE-7), 2**(ADD
                bins others         = default;
            }

            cross_signal_data:      cross signal_cp, data_cp{
                bins WR_data        = binsof(signal_cp.WR_states) && binsof(data_cp);           //All corners of data came along Write States
                bins RD_data        = binsof(signal_cp.RD_states) && binsof(data_cp);           //All corners of data came along Read States
                bins data_WR_trans  = binsof(signal_cp.WR_to_RD) && binsof(data_cp);            //All corners of data came along the transition from Write States to Read St
                bins data_RW_trans  = binsof(signal_cp.RD_to_WR) && binsof(data_cp);            //All corners of data came along the transition from Read States to Write St
            }
        endgroup


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

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

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cov_export.connect(cov_fifo.analysis_export);
        endfunction : connect_phase

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(cov_seq_item);
                cg.sample();
            end
        endtask : run_phase

    endclass : ram_coverage

endpackage : ram_coverage_pkg
```

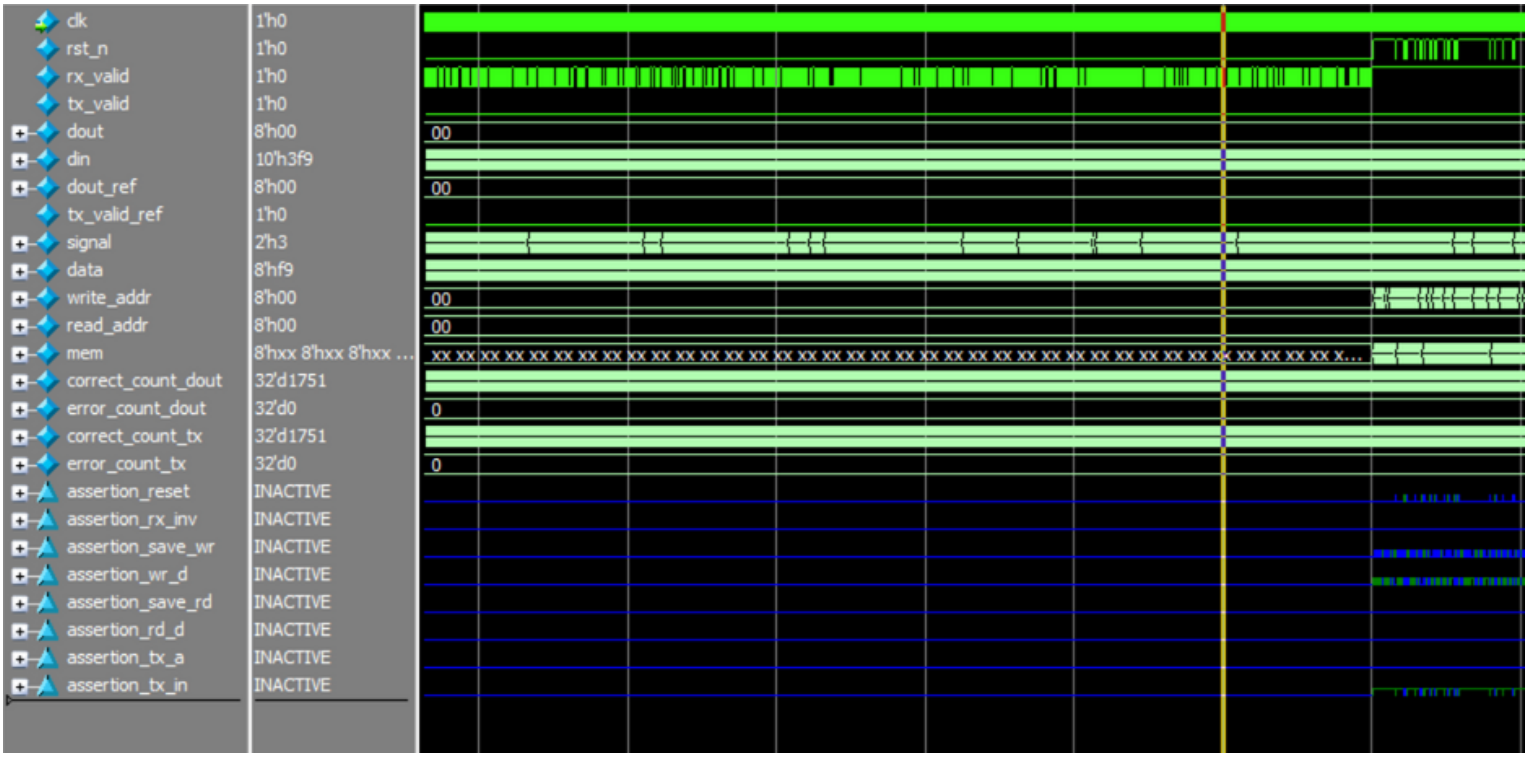
Code Snippets

ASSERTIONS

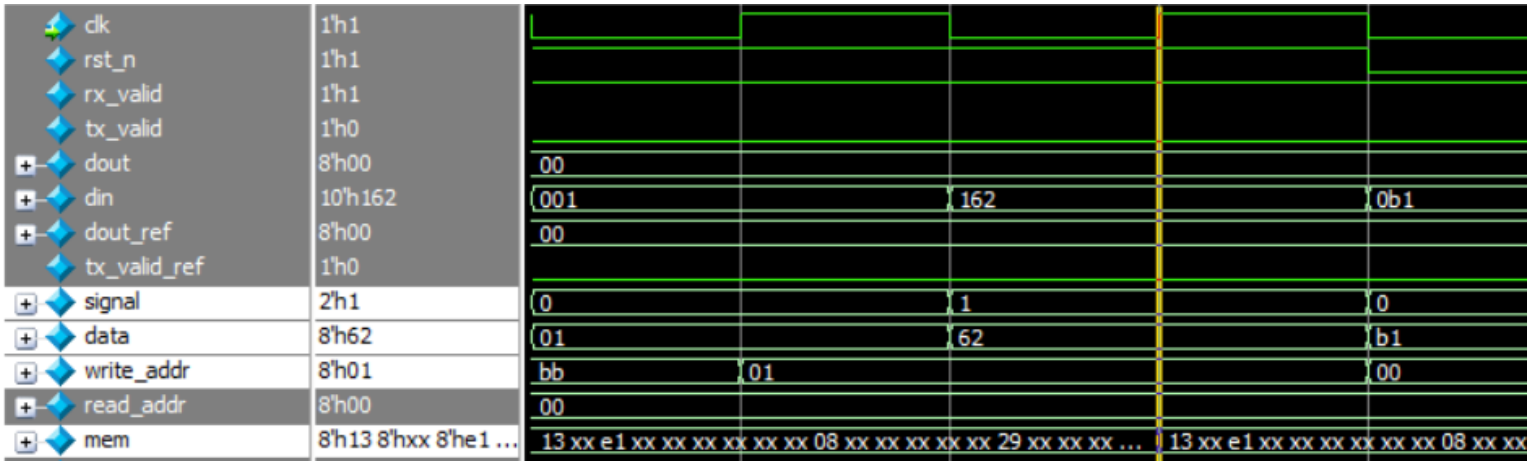
```
77  `ifdef SIM
78          //For Visual Clarity
79          logic [1:0] signal;
80          logic [intr.ADDR_SIZE-1:0] data;
81          assign signal = intr.din[intr.ADDR_SIZE+1 : intr.ADDR_SIZE];
82          assign data = intr.din[intr.ADDR_SIZE-1 : 0];
83
84
85      ////////////////////////////////////////////Checking Stable Cases (IDLE)////////////////////////////////////////////
86
87          //Checks Reset Functionality -on internal signals and outputs
88          property reset_asserted;
89              @(posedge intr.clk)
90
91              $fell(intr.rst_n) |=> !(intr.dout || intr.tx_valid || write_addr || read_addr);
92          endproperty
93
94          //Ensures No memory manipulation when rx_valid is inactive
95          property rx_valid_inactive;
96              @(posedge intr.clk) disable iff(!intr.rst_n)
97
98              $fell(intr.rx_valid) |=> $stable(mem) && $stable(write_addr) && $stable(read_addr);
99          endproperty
100
101
103     ////////////////////////////////////////////Checking Original Functionality////////////////////////////////////////////
104
105         //Checks Saving Write Address
106         property save_write_addr;
107             @(posedge intr.clk) disable iff(!intr.rst_n)
108
109             ((signal == 0) && intr.rx_valid) |=> (write_addr == $past(data));
110         endproperty
111
112         //Checks Writing Data
113         property wr_data;
114             @(posedge intr.clk) disable iff(!intr.rst_n)
115
116             ((signal == 1) && intr.rx_valid) |=> (mem[write_addr] == $past(data));
117         endproperty
118
119         //Checks Saving Read Address
120         property save_read_addr;
121             @(posedge intr.clk) disable iff(!intr.rst_n)
122
123             ((signal == 2) && intr.rx_valid) |=> (read_addr == $past(data));
124         endproperty
125
126         //Checks Reading Data
127         property rd_data;
128             @(posedge intr.clk) disable iff(!intr.rst_n)
129
130             ((signal == 3) && intr.rx_valid) |=> (intr.dout == mem[$past(read_addr)]);
131         endproperty
132
133         //Checks tx_valid is active if data is ready
134         property tx_valid_active;
135             @(posedge intr.clk) disable iff(!intr.rst_n)
136
137             ((signal == 3) && intr.rx_valid) |=> intr.tx_valid;
138         endproperty
139
140         //Checks tx_valid is not active if data is not ready (signal = 3)
141         property tx_valid_inactive;
142             @(posedge intr.clk) disable iff(!intr.rst_n)
143
144             ((signal != 3) && intr.rx_valid) |=> !intr.tx_valid;
145         endproperty
146
```

Waveform Snippets

Reset Active: Outputs equal Zero



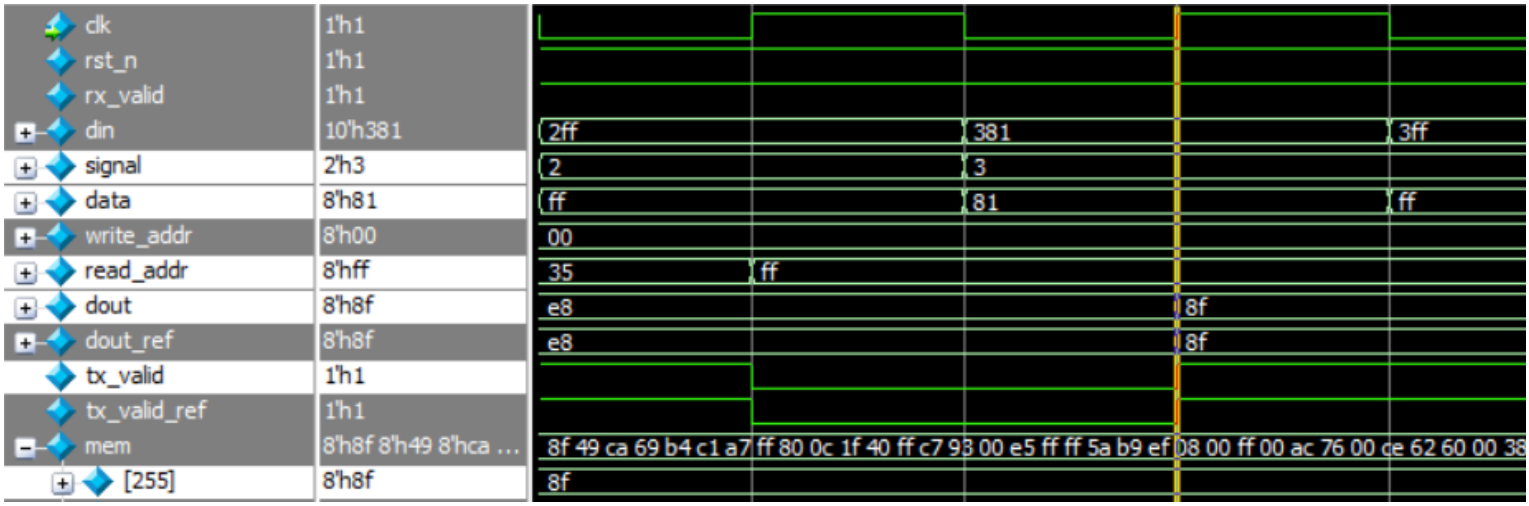
Write Operation:



Address = 8'h01, Data = 8'h62

QUESTASIM SNIPPETS

Read Operation



Transcript

```
# UVM_INFO ram_test_pkg.sv(53) @ 4000: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO ram_test_pkg.sv(55) @ 4000: uvm_test_top [run_phase] Stimulus Generation Started
# UVM_INFO ram_test_pkg.sv(57) @ 4000: uvm_test_top [run_phase] Testing Write-Only Operations
# UVM_INFO ram_test_pkg.sv(60) @ 44000: uvm_test_top [run_phase] Testing Read-Only Operations
# UVM_INFO ram_test_pkg.sv(63) @ 84000: uvm_test_top [run_phase] Testing Write-Read Operations
# UVM_INFO ram_test_pkg.sv(66) @ 124000: uvm_test_top [run_phase] Main Sequence (Complete Randomization)
# UVM_INFO ram_test_pkg.sv(69) @ 164000: uvm_test_top [run_phase] Stimulus Generation Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 164000: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ram_scoreboard_pkg.sv(62) @ 164000: uvm_test_top.env_r.sb [report_phase] Total Successful Transactions -dout-: 82000
# UVM_INFO ram_scoreboard_pkg.sv(63) @ 164000: uvm_test_top.env_r.sb [report_phase] Total Failed Transactions -dout-: 0
# UVM_INFO ram_scoreboard_pkg.sv(64) @ 164000: uvm_test_top.env_r.sb [report_phase] Total Successful Transactions -tx_valid-: 82000
# UVM_INFO ram_scoreboard_pkg.sv(65) @ 164000: uvm_test_top.env_r.sb [report_phase] Total Failed Transactions -tx_valid-: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   16
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     4
# [run_phase]     8
# ** Note: $finish    : D:/Qsim_2021/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 164 us  Iteration: 61  Instance: /ram_top
```

Coverage Report

```
31  ================================================================================
32  === Instance: /ram_top/DUT
33  === Design Unit: work.ram
34  ================================================================================
35
36  Assertion Coverage:
37      Assertions                        8        8        0   100.00%


93  Directive Coverage:
94      Directives                        8        8        0   100.00%


58  Branch Coverage:
59      Enabled Coverage              Bins      Hits    Misses  Coverage
60      ----------------              ----      ----    ------  --------
61      Branches                         7         7         0   100.00%


109 Statement Coverage:
110     Enabled Coverage              Bins      Hits    Misses  Coverage
111     ----------------              ----      ----    ------  --------
112     Statements                      15        15         0   100.00%


286 Toggle Coverage:
287     Enabled Coverage              Bins      Hits    Misses  Coverage
288     ----------------              ----      ----    ------  --------
289     Toggles                         52        52         0   100.00%


1451 Covergroup Coverage:
1452     Covergroups                      1        na        na   100.00%
1453         Coverpoints/Crosses          3        na        na        na
1454             Covergroup Bins         13        13         0   100.00%
```

Verification Plan

GENERAL PLAN:

• **Checking Results**
- Outputs' values are checked against golden model.
- Assertions are used for checking Slave's behavior.

• **Test Cases to Consider**
- Reset Signal (rst_n) Functionality across any combination of inputs.
- Normal Write Operation, 1st 2bits of MOSI = 2'b00, then 2'b01 after the next SS_n falling edge.
- Normal Read Operation, 1st 2bits of MOSI = 2'b10, then 2'b11 after the next SS_n falling edge.
- Slave's behavior on transitioning from one sequence to the other.
- Slave's behavior when SS_n stays active after transmission or goes inactive mid-transmission.

• **Functional Coverage Main Cover points and Cross Coverage**
- tx_data has taken corner values.
- tx_data has taken corner values when tx_valid was high.
- First 2bits of MOSI, following SS_n activation, have taken all possible values.
- Last 8bits of MOSI, following SS_n activation, have taken all possible values.
- First 2 bits of MOSI have cycled through all potential values within the last 8 bits of MOSI, covering the full range of possibilities.
- All combinations of control signals (rst_n, SS_n, MOSI) are covered.

• **Assertions (Specific Test Cases)**
- Stable cases (IDLE behavior) through the effect of control signals (rst_n and SS_n)
- Necessary and Illegal Transitions of Slave.
- Transitions based on 1st MOSI bit following SS_n falling edge.
- Read Transitions based on 1st 3 MOSI bits following SS_n falling edge.
- No two consecutive READ_ADD or READ_DATA states.
- 1st two bits of rx_data has to indicate the current state of the slave.
- rx_valid and tx_valid cannot be high at the same time.
- rx_valid cannot be high in any checking state.
- MISO cannot be high at any state other than READ_DATA.

Verification Requirements

Test Item	Description and Expected Output Behavior	Stimulus Generation
Reset Signal (rst_n)	Slave outputs (rx_data, rx_valid, MISO) should reset to zero on reset assertion.	rst_n = 0. All other inputs are randomized under no constraints.
Control Signal (SS_n)	Slave should stay Idle if SS_n is inactive or transition to other states (based on MOSI bits) if SS_n is active. Slave should stay stable if SS_n remained active long after data transmission. Slave should go idle if SS_n is deactivated mid data transmission.	Included in all sequences where SS_n is either active or randomized to be active most of the time.
Write_Only Operation	SS_n should equal 0 throughout the cycle -> 1 after data is collected. Cycle(1): MOSI should equal 0 (WRITE) -> 00 (Send Address) rx_data should be ready and rx_valid should be high after 8 cycles. Cycle(2): MOSI should equal 0 (WRITE) -> 01 (Send Data) rx_data should be ready and rx_valid should be high after 8 cycles.	rst_n = 1. SS_n is directed to be activated at the beginning of each sequence and deactivated following data transmission. MOSI first 3bits are set to 3'b000 following SS_n activation for sending write address. MOSI first 3bits are set to 3'b001 following SS_n activation for sending write data. MOSI last 8bits are randomized under no constraints.
Read_Only Operation	SS_n should equal 0 throughout the cycle -> 1 after data is collected. Cycle(1): MOSI should equal 1 (READ) -> 10 (Send Address) rx_data should be ready and rx_valid should be high after 8 cycles. Cycle(2): MOSI should equal 1 (READ) -> 11 (Receive Data) rx_data should be ready and rx_valid should be high after 8 cycles. when tx_valid is high, the following cycle, MISO bits start to serially output tx_data for 8 cycles.	rst_n = 1. SS_n is directed to be activated at the beginning of each sequence and deactivated following data transmission. MOSI first 3bits are set to 3'b110 following SS_n activation for sending read address. MOSI first 3bits are set to 3'b111 following SS_n activation and wait for receiving data. MOSI last 8bits are randomized under no constraints.
Write_Read_Operation	SS_n should equal 0 throughout the cycle -> 1 after data is collected. Sequences: (1): MOSI should equal 0 (WRITE) -> 00 (Send Address) rx_data should be ready and rx_valid should be high after 8 cycles. (2): MOSI should equal 0 (WRITE) -> 01 (Send Data) rx_data should be ready and rx_valid should be high after 8 cycles. (3): MOSI should equal 1 (READ) -> 10 (Send Address) rx_data should be ready and rx_valid should be high after 8 cycles. (4): MOSI should equal 1 (READ) -> 11 (Read Data) Dummy data are sent and rx_valid should be high after 8 cycles, then tx_valid becomes high and MISO reads tx_data in 8 cycles. Slave's behavior should not be affected by the order of different sequences. The slave can't go through two consecutive (READ_ADD) sequences or (READ_DATA) sequences.	rst_n = 1. SS_n is directed to be activated at the beginning of each sequence and deactivated following data transmission. MOSI bits are randomized under no constraints. Slave goes through different sequences (WRITE (Address or Data), READ_ADDR, READ_DATA) in a random manner.
Main Sequence	Same as (Write_Read_Operation) Description.	All inputs are randomized under constraints: - rst_n is inactive most of the time. - SS_n is active most of the time. to check for unexpected behavior.

Verification Requirements

Test Item	Functional Coverage	Functionality Check	Assertions
Reset Signal (rst_n)			Property: reset_asserted
Control Signal (SS_n)	Included in Cross Coverage: cross_MOSI_SS_rst: All combinations of control signals are covered.		Property: SS_inactive
Write_Only Operation	Included in Coverpoints: rx_signal_cp.signal_wr_addr: MOSI bits have taken the values required to send a write address. rx_signal_cp.signal_wr_data: MOSI bits have taken the values required to send write data. Included in Cross Coverage: rx_signal_data: MOSI bits have taken the values required to exercise all addresses and data.		Property (States): CHK_WR: Checks going from IDLE to WAIT_WR state where it waits for the 1st MOSI bit if it equals 0. MOSI_wr: Checks going to WRITE state on receiving 2 MOSI bits equal 0 following SS_n falling edge. Property (Outputs): rx_wr: Checks that 1st 2 bits of rx_data equal 00 or 01 when the slave is in WRITE state.
Read_Only Operation	Included in Coverpoints: rx_signal_cp.signal_rd_addr: MOSI bits have taken the values required to send a read address. rx_signal_cp.signal_rd_data: MOSI bits have taken the values required to receive data. tx_data_cp: tx_data has taken all values that can be read by MISO. Included in Cross Coverage: rx_signal_data: MOSI bits have taken the values required to exercise all addresses and data. tx_valid_data: All corners for to-be-read data came along active tx_valid.	Output is checked against golden model.	Property (States): CHK_RD: Checks going from IDLE to WAIT_RD state where it waits for the 1st MOSI bit if it equals 1. MOSI_rd: Checks going to WAIT_RD2 state on receiving 2 MOSI bits equal 1 following SS_n falling edge. MOSI_rd_add: Checks going to READ_ADD state on receiving 3 MOSI bits equal 110 following SS_n falling edge. MOSI_rd_data: Checks going to READ_DATA state on receiving 3 MOSI bits equal 111 following SS_n falling edge. no_add_add: Slave cannot go through two consecutive READ_ADD sequences. no_data_data: Slave cannot go through two consecutive READ_DATA sequences. Property (Outputs): rx_rd_add: Checks that 1st 2 bits of rx_data equal 10 when the slave is in READ_ADD state. rx_rd_data: Checks that 1st 2 bits of rx_data equal 11 when the slave is in READ_DATA state. rx_tx: Checks that rx_valid cannot be high when tx_valid is high.
Write_Read_Operation	Refer to (Write_Only_Operation) and (Read_Only_Operation) functional coverage.		Refer to (Write_Only_Operation) and (Read_Only_Operation) assertions. + Property (States): CHK_first: Slave must always go from IDLE to CHK_CMD. Property (Outputs): rx_chk: rx_valid cannot be high at any checking state. MISO_rd_only: MISO cannot be high at any state other than READ_DATA.
Main Sequence	Refer to (Write_Only_Operation) and (Read_Only_Operation) functional coverage. + Included in Cross Coverage: cross_MOSI_SS_rst: All combinations of control signals are covered.		

Bug Report

Bug	Original Code	Fix	Lines (Original Version)	Lines (Edited Version)
Parameter (ADDR_SIZE) should be instead of the actual size to respond correctly to changes in parameters				

Next State Logic
 	
When First MOSI bit = 0, followed by 1, slave enters WRITE state and sends a read signal to RAM.
 	
When First MOSI bit = 1, followed by 0, slave enters READ_ADD/READ_DATA states and sends a write signal to RAM.
 	
When previous first 3 MOSI bits equaled 110 directing slave to READ_ADD state. After next SS_n falling edge MOSI bits equaled 110 again, entering READ_DATA state but sends read address signal to RAM.
	READ_DATA:
	if(~SS_n && (start_to_take || start_to_give)) begin
	ns <= READ_DATA ;
	end
	else begin
	ns <= IDLE ;
	end
	CHK_CMD:
	if((~SS_n) && (MOSI == 1) && rd_addr_received) begin
	ns <= READ_DATA ;
	end
	else if((~SS_n) && (MOSI == 1)) begin
	ns <= READ_ADD ;
	end
	else if ((~SS_n) && (MOSI == 0)) begin
	ns <= WRITE ;
	end
	else if (SS_n) begin
	ns <= IDLE ;
	end
	WRITE:
	if(~SS_n && start_to_give) begin
	ns <= WRITE ;
	end
	else begin
	ns <= IDLE ;
	end
		READ_DATA:
		if(SS_n) begin
		ns <= IDLE;
		end
		else if(start_to_take) begin
		ns <= READ_DATA ;
		end
		CHK_CMD:
		if(SS_n) ns <= IDLE;
		else begin
		if(MOSI) ns <= WAIT_RD;
		else ns <= WAIT_WR;
		end
		WAIT_WR:
		if(SS_n || MOSI) ns <= IDLE;
		else ns <= WRITE;
		WAIT_RD:
		if(SS_n || ~MOSI) ns <= IDLE;
		else ns <= WAIT_RD2;
		WAIT_RD2:
		if(SS_n) ns <= IDLE;
		else begin
		if(rd_addr_received && MOSI) ns <= READ_DATA;
		else if(~rd_addr_received && ~MOSI) ns <= READ_ADD;
		else ns <= IDLE;
		end
			[42:88]	[152:204]

rx_data (after editing next state logic) does not have access to the first 2 MOSI bits, as it starts reading when in WRITE/READ_ADD states	always@(posedge clk) begin
	if (start_to_give ==1 && ~SS_n) begin
	rx_data <= {rx_data[ADDR_SIZE:0],MOSI};
	if (i==ADDR_SIZE+1) begin
	i<=0;
	rx_valid =1;
	start_to_give <= 0;
	end
	else begin
	i <= i + 1 ;
	rx_valid <= 0;
	end
	end
	else begin
	rx_valid <=0;
	if((cs == CHK_CMD) && (SS_n == 0))
	start_to_give <= 1;
	end
	end
		if (start_to_give) begin
		rx_temp <= {rx_temp[ADDR_SIZE-1:0],MOSI};
		if (i==ADDR_SIZE) begin
		i<=0;
		start_to_give <= 0;
		if(tx_valid && (cs == READ_DATA)) rx_valid <= 0;
		else rx_valid <= 1;
		end
		else begin
		i <= i + 1 ;
		rx_valid <= 0;
		end
		end
		else begin
		rx_valid <= 0;
		if(((cs == WAIT_WR) && ~MOSI) || ((cs == WAIT_RD) && MOSI))
		begin
		start_to_give <= 1;
		i <= 0;
		end
		end
		
		assign rx_data = ((cs == READ_ADD) || (cs == READ_DATA)) ? {1'b1, rx_temp} : {1'b0, rx_temp};
			[97:115]	27
				[231:249]
				293

Bug Report

Bug	Original Code	Fix	Lines (Original Version)	Lines (Edited Version)

Outputs do not reset to zero on reset assertion		always @(posedge clk or negedge rst_n) begin
		if(cs == IDLE) begin
		rx_temp <= 0;
		rx_valid <= 0;
		MISO <= 0;
		start_to_give <= 0;
		start_to_take <= 0;
		temp <= 0;
		end
		end
		
		always@(posedge clk or negedge rst_n) begin
		if(~rst_n || SS_n) begin
		rx_temp <= 0;
		rx_valid <= 0;
		i <= 0;
		end
		
		always@ (posedge tx_valid or negedge rst_n)begin
		if(~rst_n) begin
		temp <= 0;
		start_to_take <=0;
		rd_addr_received <= 0;
		end
		
		always@(posedge clk or negedge rst_n) begin
		if(~rst_n || SS_n) begin
		MISO <= 0;
				[206:215]
				[224:229]
				[253:258]
				[266:271]

MISO bits are reversed when reading tx_data
Last bit in tx_data is not read
	always@(start_to_take,posedge clk) begin
	if (start_to_take==1 && ~SS_n) begin
	MISO <= temp[0];
	temp <= {1'b0,temp[ADDR_SIZE-1:1]};
	if (j == ADDR_SIZE-1) begin
	start_to_take <= 0 ;
	j <= 0;
	end
	else begin
	j <= j + 1 ;
	end
	end
	end
		if (tx_valid && start_to_take) begin
		MISO <= temp[ADDR_SIZE-1];
		temp <= {temp[ADDR_SIZE-2:0], 1'b0};
		if (j == ADDR_SIZE) begin
		start_to_take <= 0 ;
		j <= 0;
		end
		else begin
		j <= j + 1 ;
		end
		end
		else begin
		MISO <= 0;
		start_to_take <= 0;
		j <= 0;
		end
			[124:136]	[273:288]

```
always @(posedge clk or negedge rst_n) begin
 if(cs == IDLE) begin
  rx_temp <= 0;
  rx_valid <= 0;
  MISO <= 0;
  start_to_give <= 0;
  start_to_take <= 0;
  temp <= 0;
 end
end

always@(posedge clk or negedge rst_n) begin
 if(~rst_n || SS_n) begin
  rx_temp <= 0;
  rx_valid <= 0;
  i <= 0;
 end

always@ (posedge tx_valid or negedge rst_n)begin
 if(~rst_n) begin
  temp <= 0;
  start_to_take <=0;
  rd_addr_received <= 0;
 end

always@(posedge clk or negedge rst_n) begin
 if(~rst_n || SS_n) begin
  MISO <= 0;
  start_to_take <= 0;
  j <= 0;
 end
```

CODE SNIPPETS

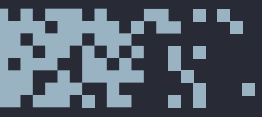
TOP MODULE

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

module slave_top;
    bit clk;

    initial begin
        forever #1 clk = ~clk;
    end

    slave_if slaveif (clk);
    slave DUT (slaveif);

    slave_ref_if slaverefif (clk);
    slave_ref REF (slaverefif);

    initial begin
        uvm_config_db#(virtual slave_if)::set(null, "uvm_test_top", "SLAVE_IF", slaveif);
        uvm_config_db#(virtual slave_ref_if)::set(null, "uvm_test_top", "SLAVE_REF_IF", slaverefif);
        run_test("slave_test");
    end

endmodule : slave_top
```

INTERFACE

```
interface slave_if (clk);

    parameter ADDR_SIZE = 8;

    input clk;
    bit MOSI, SS_n, rst_n, tx_valid, rx_valid, MISO;
    logic [ADDR_SIZE-1:0] tx_data;
    logic [ADDR_SIZE+1:0] rx_data;


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

endinterface : slave_if
```

CODE SNIPPETS



DESIGN

Before (Original)

```
30  ////////////////////////////////Original Code//////////////////////////////////////
31  /*
32
33      always@(posedge clk or negedge rst_n)   begin
34          if(~rst_n)  begin
35              cs <= IDLE ;
36          end
37          else    begin
38              cs <= ns ;
39          end
40      end
41      //Next state logic
42      always@(cs,SS_n,MOSI)   begin
43          case(cs)
44              IDLE:
45                  if(SS_n)        begin
46                      ns <= IDLE ;
47                  end
48                  else    begin
49                      ns <= CHK_CMD ;
50                  end
51              READ_DATA:
52                  if(~SS_n &&( start_to_take || start_to_give ))  begin
53                      ns <= READ_DATA ;
54                  end
55                  else    begin
56                      ns <= IDLE ;
57                  end
58              READ_ADD:
59                  if(~SS_n && start_to_give)  begin
60                      ns <= READ_ADD ;
61                  end
62                  else    begin
63                      ns <= IDLE ;
64                  end
65              CHK_CMD:
66                  if( (~SS_n) && (MOSI == 1) && rd_addr_received )    begin
67                      ns <= READ_DATA ;
68                  end
69                  else if( (~SS_n) && (MOSI == 1) )   begin
70                      ns <= READ_ADD ;
71                  end
72                  else if ( (~SS_n) && (MOSI == 0) )  begin
73                      ns <= WRITE ;
74                  end
75                  else if (SS_n)  begin
76                      ns <= IDLE ;
77                  end
```

CODE SNIPPETS

DESIGN

Before (Original)

```
78              WRITE:
79                  if(~SS_n && start_to_give)  begin
80                      ns <= WRITE ;
81                  end
82                  else    begin
83                      ns <= IDLE ;
84                  end
85
86              default: ns <= IDLE;
87          endcase
88      end
89
90      always @(posedge clk) begin
91          if (cs == READ_ADD)
92              rd_addr_received=1;
93          else if (cs == READ_DATA)
94              rd_addr_received=0;
95      end
96
97      always@(posedge clk)    begin
98          if (start_to_give ==1 && ~SS_n) begin
99              rx_data <= {rx_data[ADDR_SIZE:0],MOSI};     //param
100             if (i==ADDR_SIZE+1) begin                   //param
101                 i<=0;
102                 rx_valid =1;
103                 start_to_give <= 0;
104             end
105             else  begin
106                 i <= i + 1 ;
107                 rx_valid <= 0;
108             end
109         end
110         else begin
111             rx_valid <=0;
112             if((cs == CHK_CMD) && (SS_n == 0))
113                 start_to_give <= 1;
114         end
115     end
```

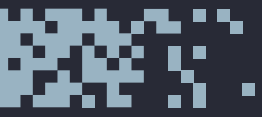
Code Snippets

DESIGN

Before (Original)

```
117
118
119         always@ (posedge tx_valid)begin
120             start_to_take <=1;
121             temp <= tx_data;
122         end
123
124         always@(start_to_take,posedge clk)  begin
125             if (start_to_take==1 && ~SS_n) begin
126                 MISO <= temp[0];
127                 temp <= {1'b0,temp[ADDR_SIZE-1:1]};         //param
128                 if (j == ADDR_SIZE-1)   begin               //param
129                     start_to_take <= 0 ;
130                     j <= 0;
131                 end
132                 else begin
133                 j <= j + 1 ;
134                 end
135             end
136         end
137     */
138     ////////////////////////////////////////////////////////////////////////////////
139
```

CODE SNIPPETS



DESIGN

After (Edited)

```
132  ////////////////////////////////Edited Code//////////////////////////////////////
133
134      always@(posedge intr.clk or negedge intr.rst_n) begin
135          if(~intr.rst_n) begin
136              cs <= IDLE ;
137          end
138          else    begin
139              cs <= ns ;
140          end
141      end
142      //Next state logic
143      always@(cs,intr.SS_n,intr.MOSI) begin
144          case(cs)
145              IDLE:
146                  if(intr.SS_n)       begin
147                      ns <= IDLE ;
148                  end
149                  else    begin
150                      ns <= CHK_CMD ;
151                  end
152              READ_DATA:
153                  if(intr.SS_n)   begin
154                      ns <= IDLE;
155                  end
156                  else if( start_to_take )    begin
157                      ns <= READ_DATA ;
158                  end
159              READ_ADD:
160                  if(~intr.SS_n && start_to_give) begin
161                      ns <= READ_ADD ;
162                  end
163                  else if(intr.SS_n)  begin
164                      ns <= IDLE ;
165                  end
166              CHK_CMD:
167                  if(intr.SS_n) ns <= IDLE;
168                  else begin
169                      if(intr.MOSI) ns <= WAIT_RD;
170                      else ns <= WAIT_WR;
171                  end
172              WAIT_WR:
173                  if(intr.SS_n || intr.MOSI) ns <= IDLE;
174                  else ns <= WRITE;
175              WAIT_RD:
176                  if(intr.SS_n || ~intr.MOSI) ns <= IDLE;
177                  else ns <= WAIT_RD2;
```

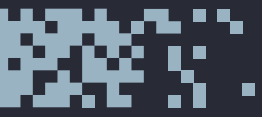
CODE SNIPPETS

DESIGN

After (Edited)

```
178                 WAIT_RD2:
179                     if(intr.SS_n) ns <= IDLE;
180                     else begin
181                         if(rd_addr_received && intr.MOSI) ns <= READ_DATA;
182                         else if(~rd_addr_received && ~intr.MOSI) ns <= READ_ADD;
183                         else ns <= IDLE;
184                     end
185                 WRITE:
186                     if(~intr.SS_n && start_to_give) begin
187                         ns <= WRITE ;
188                     end
189                     else if(intr.SS_n)  begin
190                         ns <= IDLE ;
191                     end
192 
193                 default: ns <= IDLE;
194             endcase
195         end
196 
197         always @(posedge intr.clk or negedge intr.rst_n) begin
198             if(cs == IDLE) begin
199                 rx_temp <= 0;
200                 intr.rx_valid <= 0;
201                 intr.MISO <= 0;
202                 start_to_give <= 0;
203                 start_to_take <= 0;
204                 temp <= 0;
205             end
206         end
207 
208         always @(posedge intr.clk or negedge intr.rst_n) begin
209             if(!intr.rst_n) rd_addr_received=0;
210             else begin
211                 if (cs == READ_ADD)
212                     rd_addr_received=1;
213                 else if (cs == READ_DATA)
214                     rd_addr_received=0;
215             end
216         end
```

CODE SNIPPETS



DESIGN

After (Edited)

```
218     always@(posedge intr.clk or negedge intr.rst_n) begin
219         if(~intr.rst_n || intr.SS_n) begin
220             rx_temp <= 0;
221             intr.rx_valid <= 0;
222             i <= 0;
223         end
224         else begin
225             if (start_to_give)  begin
226                 rx_temp <= {rx_temp[intr.ADDR_SIZE-1:0],intr.MOSI}; //param
227                 if (i==intr.ADDR_SIZE) begin                    //param
228                     i<=0;
229                     start_to_give <= 0;
230                     if(intr.tx_valid && (cs == READ_DATA)) intr.rx_valid <= 0;
231                     else intr.rx_valid <= 1;
232                 end
233                 else  begin
234                     i <= i + 1 ;
235                     intr.rx_valid <= 0;
236                 end
237             end
238             else begin
239                 intr.rx_valid <= 0;
240                 if( ((cs == WAIT_WR) && ~intr.MOSI) || ((cs == WAIT_RD) && intr.MOSI) ) begin
241                     start_to_give <= 1;
242                     i <= 0;
243                 end
244             end
245         end
246     end
247
248     always@ (posedge intr.tx_valid or negedge intr.rst_n)begin
249         if(~intr.rst_n) begin
250             temp <= 0;
251             start_to_take <=0;
252             rd_addr_received <= 0;
253         end
254         else if(cs == READ_DATA) begin
255             start_to_take <=1;
256             temp <= intr.tx_data;
257         end
258         else start_to_take <= 0;
259     end
```

CODE SNIPPETS

DESIGN

After (Edited)

```
261     always@(posedge intr.clk or negedge intr.rst_n) begin
262         if(~intr.rst_n || intr.SS_n) begin
263             intr.MISO <= 0;
264             start_to_take <= 0;
265             j <= 0;
266         end
267         else begin
268             if (intr.tx_valid && start_to_take) begin
269                 intr.MISO <= temp[intr.ADDR_SIZE-1];
270                 temp <= {temp[intr.ADDR_SIZE-2:0], 1'b0};   //param
271                 if (j == intr.ADDR_SIZE)    begin          //param
272                     start_to_take <= 0 ;
273                     j <= 0;
274                 end
275                 else begin
276                     j <= j + 1 ;
277                 end
278             end
279             else begin
280                 intr.MISO <= 0;
281                 start_to_take <= 0;
282                 j <= 0;
283             end
284         end
285     end
286 
287 
288     assign intr.rx_data = ((cs == READ_ADD) || (cs == READ_DATA)) ? {1'b1, rx_temp} : (cs == WRITE) ? {1'b0, rx_temp} : 0;
289 
```

CODE SNIPPETS

TEST

```
package slave_test_pkg;
import uvm_pkg::*;
import slave_env_pkg::*;
import slave_config_pkg::*;
import slave_sequence_pkg::*;
`include "uvm_macros.svh"

    class slave_test extends uvm_test;
        `uvm_component_utils(slave_test)

        slave_env env_s;
        slave_config slave_cfg;

        slave_reset_sequence reset_seq;
        slave_write_only_sequence write_only_seq;
        slave_read_only_sequence read_only_seq;
        slave_write_read_sequence write_read_seq;
        slave_main_sequence main_seq;


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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            env_s = slave_env::type_id::create("env_s", this);
            slave_cfg = slave_config::type_id::create("slave_cfg", this);

            reset_seq       = slave_reset_sequence::type_id::create("reset_seq", this);
            write_only_seq  = slave_write_only_sequence::type_id::create("write_only_seq", this);
            read_only_seq   = slave_read_only_sequence::type_id::create("read_only_seq", this);
            write_read_seq  = slave_write_read_sequence::type_id::create("write_read_seq", this);
            main_seq        = slave_main_sequence::type_id::create("main_seq", this);

            slave_cfg.active = UVM_ACTIVE;

            if(!uvm_config_db #(virtual slave_if)::get(this, "", "SLAVE_IF", slave_cfg.slave_vif))
                `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the Slave (DUT) from the uvm_config_db");

            if(!uvm_config_db #(virtual slave_ref_if)::get(this, "", "SLAVE_REF_IF", slave_cfg.slave_ref_vif))
                `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the Slave (REF) from the uvm_config_db");

            uvm_config_db #(slave_config)::set(this, "*", "SLAVE_CFG", slave_cfg);

            //set_report_verbosity_level_hier(UVM_HIGH);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            `uvm_info("run_phase", "Reset Asserted", UVM_LOW);
            reset_seq.start(env_s.agt.sqr);
            `uvm_info("run_phase", "Reset Deasserted", UVM_LOW);

            `uvm_info("run_phase", "Stimulus Generation Started", UVM_LOW);

            `uvm_info("run_phase", "Testing Write-Only Operations", UVM_LOW);
            write_only_seq.start(env_s.agt.sqr);

            `uvm_info("run_phase", "Testing Read-Only Operations", UVM_LOW);
            read_only_seq.start(env_s.agt.sqr);

            `uvm_info("run_phase", "Testing Write-Read Operations", UVM_LOW);
            write_read_seq.start(env_s.agt.sqr);

            `uvm_info("run_phase", "Main Sequence (Complete Randomization)", UVM_LOW);
            main_seq.start(env_s.agt.sqr);

            `uvm_info("run_phase", "Stimulus Generation Ended", UVM_LOW);
            phase.drop_objection(this);
        endtask : run_phase

    endclass : slave_test

endpackage : slave_test_pkg
```

CODE SNIPPETS

CONFIGURATION OBJECT

```
1   package slave_config_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4
5       class slave_config extends uvm_object;
6           `uvm_object_utils(slave_config)
7
8           virtual slave_if slave_vif;
9           virtual slave_ref_if slave_ref_vif;
10          uvm_active_passive_enum active;
11
12          function new(string name = "slave_config");
13              super.new(name);
14          endfunction : new
15
16      endclass : slave_config
17
18  endpackage : slave_config_pkg
```

CODE SNIPPETS

ENVIRONMENT

```
1   package slave_env_pkg;
2   import uvm_pkg::*;
3   import slave_agent_pkg::*;
4   import slave_scoreboard_pkg::*;
5   import slave_coverage_pkg::*;
6   `include "uvm_macros.svh"
7
8       class slave_env extends uvm_env;
9           `uvm_component_utils(slave_env);
10
11          slave_agent agt;
12          slave_scoreboard sb;
13          slave_coverage cov;
14
15          function new(string name = "slave_env", uvm_component parent = null);
16              super.new(name, parent);
17          endfunction : new
18
19          function void build_phase(uvm_phase phase);
20              super.build_phase(phase);
21              agt = slave_agent::type_id::create("agt", this);
22              sb = slave_scoreboard::type_id::create("sb", this);
23              cov = 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 : connect_phase
31
32      endclass : slave_env
33  endpackage : slave_env_pkg
```

CODE SNIPPETS

AGENT

```
package slave_agent_pkg;
import uvm_pkg::*;
import slave_driver_pkg::*;
import slave_sequencer_pkg::*;
import slave_monitor_pkg::*;
import slave_config_pkg::*;
import slave_sequence_item_pkg::*;

`include "uvm_macros.svh"

    class slave_agent extends uvm_agent;
        `uvm_component_utils(slave_agent)

        slave_driver drv;
        slave_sequencer sqr;
        slave_monitor mon;
        slave_config slave_cfg;
        uvm_analysis_port #(slave_sequence_item) agt_ap;

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            slave_cfg = slave_config::type_id::create("slave_cfg", this);
            if(!uvm_config_db#(slave_config)::get(this, "", "SLAVE_CFG", slave_cfg))
                `uvm_fatal("build_phase", "Agent - Unable to get Slave Configuration Object.");

            if(slave_cfg.active) begin
                drv = slave_driver::type_id::create("drv", this);
                sqr = slave_sequencer::type_id::create("sqr", this);
            end
            mon = slave_monitor::type_id::create("mon", this);

            agt_ap = new("agt_ap", this);
        endfunction : build_phase

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            if(slave_cfg.active) begin
                drv.slave_drv_vif = slave_cfg.slave_vif;
                drv.slave_ref_drv_vif = slave_cfg.slave_ref_vif;
                drv.seq_item_port.connect(sqr.seq_item_export);
            end
            mon.slave_mon_vif = slave_cfg.slave_vif;
            mon.slave_ref_mon_vif = slave_cfg.slave_ref_vif;
            mon.mon_ap.connect(agt_ap);
        endfunction : connect_phase

    endclass : slave_agent

endpackage : slave_agent_pkg
```

CODE SNIPPETS

SEQUENCE ITEM

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

    class slave_sequence_item extends uvm_sequence_item;
        `uvm_object_utils(slave_sequence_item)

        rand bit MOSI, SS_n, rst_n, tx_valid;
        rand logic [ADDR_SIZE-1:0] tx_data;

        bit rx_valid, rx_valid_ref, MISO, MISO_ref;
        logic [ADDR_SIZE+1:0] rx_data, rx_data_ref;

        //state_e cs, ns;

        rand bit [1:0] selector_tx;

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

        function string convert2string();
            return $sformatf("%s    rst_n = 0b%0b,  SS_n = 0b%0b,   MOSI = 0b%0b,   tx_valid = 0b%0b,   tx_data = 0x%0h,    MISO = 0b%0b,
                              super.convert2string(), rst_n, SS_n, MOSI, tx_valid, tx_data, MISO, rx_valid, rx_data);
        endfunction : convert2string

        function string convert2string_stimulus();
            return $sformatf("rst_n = 0b%0b,    SS_n = 0b%0b,   MOSI = 0b%0b,   tx_valid = 0b%0b,   tx_data = 0x%0h",
                              rst_n, SS_n, MOSI, tx_valid, tx_data);
        endfunction : convert2string_stimulus

        //Constraints
        constraint rst_c {
            rst_n           dist {0:=5,     1:=95};                         //rst_n is inactive most of the time
        }

        constraint SS_c {
            SS_n            dist {0:=90,    1:=10};                         //SS_n is active most of the time
        }

        constraint tx {
            tx_valid        dist {0:=70,    1:=30};                         //tx_valid is inactive most of the time
            selector_tx     dist {0:=30,    1:=30,      [2:3]:=40};

            if(selector_tx == 0) tx_data inside {ALL_ONES, ZERO};
            if(selector_tx == 1) $countones(tx_data) == 1;
        }


    endclass : slave_sequence_item

endpackage : slave_sequence_item_pkg
```

CODE SNIPPETS

SEQUENCE

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

parameter TESTS = 3000;

    class slave_reset_sequence extends uvm_sequence #(slave_sequence_item);
        `uvm_object_utils(slave_reset_sequence)

        slave_sequence_item seq_item;

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

        task body();
            repeat(TESTS/10) begin
                seq_item = slave_sequence_item::type_id::create("seq_item");
                start_item(seq_item);
                seq_item.constraint_mode(0);
                assert(seq_item.randomize() with {rst_n == 0;});
                finish_item(seq_item);
            end
        endtask : body

    endclass : slave_reset_sequence



    class slave_write_only_sequence extends uvm_sequence #(slave_sequence_item);
        `uvm_object_utils(slave_write_only_sequence)

        slave_sequence_item seq_item;

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

        task body();
            repeat(TESTS) begin
                write_add_seq(seq_item);
                write_data_seq(seq_item);
            end
        endtask : body
```

CODE SNIPPETS

SEQUENCE

```
48          ////////////////////////////////START & END////////////////////////////////
49          task start_seq(slave_sequence_item seq_item);
50              seq_item = slave_sequence_item::type_id::create("seq_item");
51              start_item(seq_item);
52              assert(seq_item.randomize() with {rst_n == 1; SS_n == 0;})
53              finish_item(seq_item);
54          endtask : start_seq
55
56          task end_seq(slave_sequence_item seq_item);
57              seq_item = slave_sequence_item::type_id::create("seq_item");
58              start_item(seq_item);
59              assert(seq_item.randomize() with {rst_n == 1; SS_n == 1;});
60              finish_item(seq_item);
61          endtask : end_seq
62
63          ////////////////////////////////WRITE_ADDRESS////////////////////////////////
64          task write_add_seq(slave_sequence_item seq_item);
65              start_seq(seq_item);
66              repeat(3) begin
67                  seq_item = slave_sequence_item::type_id::create("seq_item");
68                  start_item(seq_item);
69                  assert(seq_item.randomize() with {rst_n == 1; SS_n == 0; MOSI == 0;})
70                  finish_item(seq_item);
71              end
72
73              repeat(ADDR_SIZE) begin
74                  seq_item = slave_sequence_item::type_id::create("seq_item");
75                  start_item(seq_item);
76                  assert(seq_item.randomize() with {rst_n == 1; SS_n == 0;});
77                  finish_item(seq_item);
78              end
79              end_seq(seq_item);
80          endtask : write_add_seq
81
82          ////////////////////////////////WRITE_DATA////////////////////////////////
83          task write_data_seq(slave_sequence_item seq_item);
84              start_seq(seq_item);
85              repeat(2) begin
86                  seq_item = slave_sequence_item::type_id::create("seq_item");
87                  start_item(seq_item);
88                  assert(seq_item.randomize() with {rst_n == 1; SS_n == 0; MOSI == 0;})
89                  finish_item(seq_item);
90              end
91
92              seq_item = slave_sequence_item::type_id::create("seq_item");
93              start_item(seq_item);
94              assert(seq_item.randomize() with {rst_n == 1; SS_n == 0; MOSI == 1;})
95              finish_item(seq_item);
96
97              repeat(ADDR_SIZE) begin
98                  seq_item = slave_sequence_item::type_id::create("seq_item");
99                  start_item(seq_item);
100                 assert(seq_item.randomize() with {rst_n == 1; SS_n == 0;});
101                 finish_item(seq_item);
102             end
103             end_seq(seq_item);
104         endtask : write_data_seq
```

CODE SNIPPETS

SEQUENCE

```
110  class slave_read_only_sequence extends uvm_sequence #(slave_sequence_item);
111      `uvm_object_utils(slave_read_only_sequence)
112
113      slave_sequence_item seq_item;
114
115      function new(string name = "slave_read_only_sequence");
116          super.new(name);
117      endfunction : new
118
119      task body();
120          repeat(TESTS) begin
121              read_add_seq(seq_item);
122              read_data_seq(seq_item);
123          end
124      endtask : body

141      ////////////////////////////READ_ADDRESS////////////////////////////
142      task read_add_seq(slave_sequence_item seq_item);
143          start_seq(seq_item);
144          repeat(2) begin
145              seq_item = slave_sequence_item::type_id::create("seq_item");
146              start_item(seq_item);
147              assert(seq_item.randomize() with {rst_n == 1; SS_n == 0; MOSI == 1;})
148              finish_item(seq_item);
149          end
150
151          seq_item = slave_sequence_item::type_id::create("seq_item");
152          start_item(seq_item);
153          assert(seq_item.randomize() with {rst_n == 1; SS_n == 0; MOSI == 0;})
154          finish_item(seq_item);
155
156          repeat(ADDR_SIZE) begin
157              seq_item = slave_sequence_item::type_id::create("seq_item");
158              start_item(seq_item);
159              assert(seq_item.randomize() with {rst_n == 1; SS_n == 0;});
160              finish_item(seq_item);
161          end
162          end_seq(seq_item);
163      endtask : read_add_seq
164
165      ////////////////////////////READ_DATA////////////////////////////
166      task read_data_seq(slave_sequence_item seq_item);
167          start_seq(seq_item);
168          repeat(3) begin
169              seq_item = slave_sequence_item::type_id::create("seq_item");
170              start_item(seq_item);
171              assert(seq_item.randomize() with {rst_n == 1; SS_n == 0; MOSI == 1;})
172              finish_item(seq_item);
173          end
174
175          repeat(ADDR_SIZE) begin
176              seq_item = slave_sequence_item::type_id::create("seq_item");
177              start_item(seq_item);
178              assert(seq_item.randomize() with {rst_n == 1; SS_n == 0;});
179              finish_item(seq_item);
180          end
181
182          repeat(ADDR_SIZE+2) begin
183              seq_item = slave_sequence_item::type_id::create("seq_item");
184              start_item(seq_item);
185              assert(seq_item.randomize() with {rst_n == 1; SS_n == 0;});
186              finish_item(seq_item);
187          end
188          end_seq(seq_item);
189      endtask : read_data_seq
190
191  endclass : slave_read_only_sequence
```

CODE SNIPPETS

SEQUENCE

```
195     class slave_write_read_sequence extends uvm_sequence #(slave_sequence_item);
196         `uvm_object_utils(slave_write_read_sequence)
197
198         slave_sequence_item seq_item;
199
200         function new(string name = "slave_write_read_sequence");
201             super.new(name);
202         endfunction : new
203
204         task body();
205             repeat(TESTS) begin
206                 case($urandom_range(0,3))
207                     0: write_add_seq(seq_item);
208                     1: write_data_seq(seq_item);
209                     2: read_add_seq(seq_item);
210                     3: read_data_seq(seq_item);
211                 endcase //seq_item.selector_seq
212             end
213         endtask : body


327     class slave_main_sequence extends uvm_sequence #(slave_sequence_item);
328         `uvm_object_utils(slave_main_sequence)
329
330         slave_sequence_item seq_item;
331
332         function new(string name = "slave_main_sequence");
333             super.new(name);
334         endfunction : new
335
336         task body();
337             repeat(TESTS) begin
338                 seq_item = slave_sequence_item::type_id::create("seq_item");
339                 start_item(seq_item);
340                 assert(seq_item.randomize());
341                 finish_item(seq_item);
342             end
343         endtask : body
344
345     endclass : slave_main_sequence
346
347 endpackage : slave_sequence_pkg
```

CODE SNIPPETS

SEQUENCER

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

CODE SNIPPETS

DRIVER

```
1   package slave_driver_pkg;
2   import slave_sequence_item_pkg::*;
3   import shared_pkg::*;
4   import uvm_pkg::*;
5   `include "uvm_macros.svh"
6
7       class slave_driver extends uvm_driver #(slave_sequence_item);
8           `uvm_component_utils(slave_driver)
9
10          virtual slave_if slave_drv_vif;
11          virtual slave_ref_if slave_ref_drv_vif;
12
13          slave_sequence_item stim_seq_item;
14
15          function new(string name = "slave_driver", uvm_component parent = null);
16              super.new(name, parent);
17          endfunction : new
18
19          task run_phase(uvm_phase phase);
20              super.run_phase(phase);
21              forever begin
22                  stim_seq_item = slave_sequence_item::type_id::create("stim_seq_item");
23                  seq_item_port.get_next_item(stim_seq_item);
24                  //DUT
25                  slave_drv_vif.rst_n     = stim_seq_item.rst_n;
26                  slave_drv_vif.SS_n      = stim_seq_item.SS_n;
27                  slave_drv_vif.MOSI      = stim_seq_item.MOSI;
28                  slave_drv_vif.tx_valid  = stim_seq_item.tx_valid;
29                  slave_drv_vif.tx_data   = stim_seq_item.tx_data;
30
31                  //REF
32                  slave_ref_drv_vif.rst_n     = stim_seq_item.rst_n;
33                  slave_ref_drv_vif.SS_n      = stim_seq_item.SS_n;
34                  slave_ref_drv_vif.MOSI      = stim_seq_item.MOSI;
35                  slave_ref_drv_vif.tx_valid  = stim_seq_item.tx_valid;
36                  slave_ref_drv_vif.tx_data   = stim_seq_item.tx_data;
37
38                  @(negedge slave_drv_vif.clk);
39                  seq_item_port.item_done();
40                  `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH);
41              end
42          endtask : run_phase
43
44      endclass : slave_driver
45
46  endpackage : slave_driver_pkg
```

CODE SNIPPETS

MONITOR

```
package slave_monitor_pkg;
import uvm_pkg::*;
import shared_pkg::*;
import slave_sequence_item_pkg::*;

`include "uvm_macros.svh"

    class slave_monitor extends uvm_monitor;
        `uvm_component_utils(slave_monitor)

        slave_sequence_item rsp_seq_item;
        virtual slave_if slave_mon_vif;
        virtual slave_ref_if slave_ref_mon_vif;
        uvm_analysis_port #(slave_sequence_item) mon_ap;

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

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

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

            forever begin
                rsp_seq_item = slave_sequence_item::type_id::create("rsp_seq_item");
                @(negedge slave_mon_vif.clk);
                rsp_seq_item.rst_n          = slave_mon_vif.rst_n;
                rsp_seq_item.SS_n           = slave_mon_vif.SS_n;
                rsp_seq_item.MOSI           = slave_mon_vif.MOSI;
                rsp_seq_item.tx_valid       = slave_mon_vif.tx_valid;
                rsp_seq_item.tx_data        = slave_mon_vif.tx_data;
                rsp_seq_item.MISO           = slave_mon_vif.MISO;
                rsp_seq_item.rx_valid       = slave_mon_vif.rx_valid;
                rsp_seq_item.rx_data        = slave_mon_vif.rx_data;
                rsp_seq_item.MISO_ref       = slave_ref_mon_vif.MISO_ref;
                rsp_seq_item.rx_valid_ref   = slave_ref_mon_vif.rx_valid_ref;
                rsp_seq_item.rx_data_ref    = slave_ref_mon_vif.rx_data_ref;

                mon_ap.write(rsp_seq_item);
                `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH);
            end
        endtask : run_phase

    endclass : slave_monitor

endpackage : slave_monitor_pkg
```

CODE SNIPPETS

SCOREBOARD

```
package slave_scoreboard_pkg;
import uvm_pkg::*;
import shared_pkg::*;
import slave_sequence_item_pkg::*;

`include "uvm_macros.svh"

    class slave_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(slave_scoreboard);

        slave_sequence_item sb_seq_item;
        uvm_analysis_export #(slave_sequence_item) sb_export;
        uvm_tlm_analysis_fifo #(slave_sequence_item) sb_fifo;

        bit data_addr;

        int correct_count_rxdata, error_count_rxdata;
        int correct_count_rxvalid, error_count_rxvalid;
        int correct_count_MISO, error_count_MISO;


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

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

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


        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(sb_seq_item);

                if(sb_seq_item.rx_valid !== sb_seq_item.rx_valid_ref) begin
                    `uvm_error("run_phase", $sformatf("Comparison Failed, Transaction received by DUT: %s, while the reference output -rx_valid-:0b%0b", sb_seq_item.c
                    error_count_rxvalid++;
                end
                else begin
                    `uvm_info("run_phase", $sformatf("Correct slave output -rx_valid-: %s", sb_seq_item.convert2string()), UVM_HIGH);
                    correct_count_rxvalid++;
                    if(sb_seq_item.rx_valid) begin
                        if(sb_seq_item.rx_data !== sb_seq_item.rx_data_ref) begin
                            `uvm_error("run_phase", $sformatf("Comparison Failed, Transaction received by DUT: %s, while the reference output -rx_data-:0x%0h", sb_se
                            error_count_rxdata++;
                        end
                        else begin
                            `uvm_info("run_phase", $sformatf("Correct slave output -rx_data-: %s", sb_seq_item.convert2string()), UVM_HIGH);
                            correct_count_rxdata++;
                        end
                    end
                end

                if(sb_seq_item.MISO !== sb_seq_item.MISO_ref) begin
                    `uvm_error("run_phase", $sformatf("Comparison Failed, Transaction received by DUT: %s, while the reference output -MISO-:0b%0b", sb_seq_item.conv
                    error_count_MISO++;
                end
                else begin
                    `uvm_info("run_phase", $sformatf("Correct slave output -MISO-: %s", sb_seq_item.convert2string()), UVM_HIGH);
                    correct_count_MISO++;
                end
            end

        endtask : run_phase

        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("report_phase", $sformatf("Total Successful Transactions -rx_data-: %0d", correct_count_rxdata), UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("Total Failed Transactions -rx_data-: %0d", error_count_rxdata), UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("Total Successful Transactions -rx_valid-: %0d", correct_count_rxvalid), UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("Total Failed Transactions -rx_valid-: %0d", error_count_rxvalid), UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("Total Successful Transactions -MISO-: %0d", correct_count_MISO), UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("Total Failed Transactions -MISO-: %0d", error_count_MISO), UVM_MEDIUM);
        endfunction : report_phase

    endclass : slave_scoreboard

endpackage : slave_scoreboard_pkg
```

CODE SNIPPETS

COVERAGE

```
package slave_coverage_pkg;
import uvm_pkg::*;
import shared_pkg::*;
import slave_sequence_item_pkg::*;

`include "uvm_macros.svh"

    class slave_coverage extends uvm_component;
        `uvm_component_utils(slave_coverage);

        uvm_analysis_export #(slave_sequence_item) cov_export;
        uvm_tlm_analysis_fifo #(slave_sequence_item) cov_fifo;

        slave_sequence_item cov_seq_item;

        covergroup cg;

            ////////////////////////////////////////////////////////////////Coverpoints//////////////////////////////////////////////////////////////

            tx_data_cp:     coverpoint cov_seq_item.tx_data{
                bins Corners    = {ZERO, ALL_ONES, 8'haa, 8'h55};   //Needs to be edited if a different ADDER_SIZE is used
                bins others[]   = default;
            }

            rx_signal_cp:   coverpoint cov_seq_item.rx_data[ADDR_SIZE+1:ADDR_SIZE] iff(cov_seq_item.rx_valid){
                bins signal_wr_addr = {2'b00};
                bins signal_wr_data = {2'b01};
                bins signal_rd_addr = {2'b10};
                bins signal_rd_data = {2'b11};
                bins signal_trans   = (2'b00 => 2'b01 => 2'b10 => 2'b11);
            }

            rx_data_cp:     coverpoint cov_seq_item.rx_data[ADDR_SIZE-1:0] iff(cov_seq_item.rx_valid){
                bins Corners    = {ZERO, ALL_ONES, 8'haa, 8'h55};   //Needs to be edited if a different ADDER_SIZE is used
                bins others[]   = default;
            }

            rst_n_cp:       coverpoint cov_seq_item.rst_n;
            SS_n_cp:        coverpoint cov_seq_item.SS_n;
            MOSI_cp:        coverpoint cov_seq_item.MOSI;
            tx_valid_cp:    coverpoint cov_seq_item.tx_valid;

            ////////////////////////////////////////////////////////////////Cross Coverage//////////////////////////////////////////////////////////////

            cross_MOSI_SS_rst:      cross rst_n_cp, SS_n_cp, MOSI_cp;                                   //All combinations of Control Signals occured

            cross_rx_signal_data:   cross rx_signal_cp, rx_data_cp{                                     //Output has taken all the required values for expected usage
                ignore_bins trans = binsof(rx_signal_cp.signal_trans) && binsof(rx_data_cp);
            }

            cross_tx_valid_data:    cross tx_data_cp, tx_valid_cp{                                      //All corners of read data came along active tx_valid
                bins tx_val_data        = binsof(tx_data_cp) && binsof(tx_valid_cp) intersect {1};
                ignore_bins tx_inv      = binsof(tx_data_cp) && binsof(tx_valid_cp) intersect {0};
            }
        endgroup



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

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

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cov_export.connect(cov_fifo.analysis_export);
        endfunction : connect_phase

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(cov_seq_item);
                cg.sample();
            end
        endtask : run_phase

    endclass : slave_coverage

endpackage : slave_coverage_pkg
```

CODE SNIPPETS

ASSERTIONS

```
`ifdef SIM
    typedef enum logic [2:0] {IDLE_, READ_DATA_, READ_ADD_, CHK_CMD_, WRITE_, WAIT_WR_, WAIT_RD_, WAIT_RD2_} state_e;

    //For Visual Clarity
    state_e cs_sva, ns_sva;
    assign cs_sva = state_e'(cs);
    assign ns_sva = state_e'(ns);


////////////////////////////////////////////////Checking Stable Cases (IDLE)/////////////////////////////////////////////

    //Checks rst_n Functionality - Output Values and State Transitions
    property reset_asserted;
        @(posedge intr.clk)

        $fell(intr.rst_n) |-> !(intr.rx_valid || intr.rx_data || intr.MISO) && (cs_sva == IDLE_);
    endproperty

    //Checks SS_n Functionality - State Transitions
    property SS_inactive;
        @(posedge intr.clk) disable iff(!intr.rst_n)

        $rose(intr.SS_n) |=> (cs_sva == IDLE_);
    endproperty


/////////////////////////////////////////////Checking Original Functionality/////////////////////////////////////////////

   ////////////////////////////////////////Checking State Transitions

    //Slave must always go from IDLE to CHK_CMD
    property CHK_first;
        @(posedge intr.clk) disable iff(!intr.rst_n)

        $fell(intr.SS_n) |=> (cs_sva == CHK_CMD_);
    endproperty

    //Slave must always go from CHK_CMD to WAIT_WR if 1st MOSI bit = 0
    property CHK_WR;
        @(posedge intr.clk) disable iff(!intr.rst_n)

        $fell(intr.SS_n) ##1 (~intr.MOSI && ~intr.SS_n) |=> (cs_sva == WAIT_WR_);
    endproperty

    //Slave must always go from CHK_CMD to WAIT_RD if 1st MOSI bit = 1
    property CHK_RD;
        @(posedge intr.clk) disable iff(!intr.rst_n)

        $fell(intr.SS_n) ##1 (intr.MOSI && ~intr.SS_n) |=> (cs_sva == WAIT_RD_);
    endproperty



    //MOSI = 0x
    property MOSI_wr;
        @(posedge intr.clk) disable iff(!intr.rst_n)

        $fell(intr.SS_n) ##1 (~intr.MOSI && ~intr.SS_n)[*2] |=> (cs_sva == WRITE_);
    endproperty

    //MOSI = 1x
    property MOSI_rd;
        @(posedge intr.clk) disable iff(!intr.rst_n)

        $fell(intr.SS_n) ##1 (intr.MOSI && ~intr.SS_n)[*2] |=> (cs_sva == WAIT_RD2_);
    endproperty

    //MOSI = 10
    property MOSI_rd_add;
        @(posedge intr.clk) disable iff(!intr.rst_n)

        ( (cs_sva == READ_DATA_) throughout intr.SS_n[->1] ) ##[1:5] $fell(intr.SS_n) ##1 (intr.MOSI && ~intr.SS_n)[*2] ##1 (~intr.MOSI && ~intr.SS_n) |=> (cs_sva == READ_ADD_);
    endproperty

    //MOSI = 11
    property MOSI_rd_data;
        @(posedge intr.clk) disable iff(!intr.rst_n)

        ( (cs_sva == READ_ADD_) throughout intr.SS_n[->1] ) ##[1:5] $fell(intr.SS_n) ##1 (intr.MOSI && ~intr.SS_n)[*3] |=> (cs_sva == READ_DATA_);
    endproperty



    //(READ_ADD -> READ_ADD) or (READ_DATA -> READ_DATA) must not occur
    property no_add_add;
        @(posedge intr.clk) disable iff(!intr.rst_n)

        $fell(intr.SS_n) ##1 (!intr.SS_n)[*3] ##1 (cs_sva == READ_ADD_)[*1:$] ##1 (intr.SS_n) |=> ( (cs_sva != READ_ADD_) throughout (cs_sva == READ_DATA_)[->1] );
    endproperty

    property no_data_data;
        @(posedge intr.clk) disable iff(!intr.rst_n)

        $fell(intr.SS_n) ##1 (!intr.SS_n)[*3] ##1 (cs_sva == READ_DATA_)[*1:$] ##1 (intr.SS_n) |=> ( (cs_sva != READ_DATA_) throughout (cs_sva == READ_ADD_)[->1] );
    endproperty
```

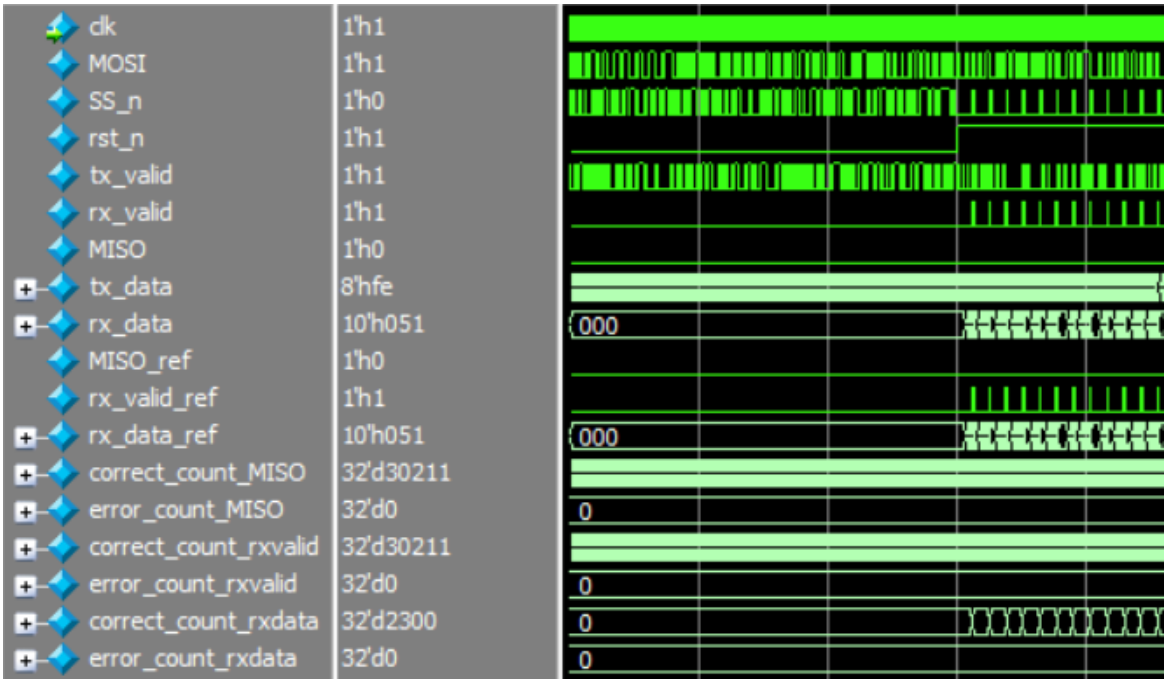
CODE SNIPPETS

ASSERTIONS

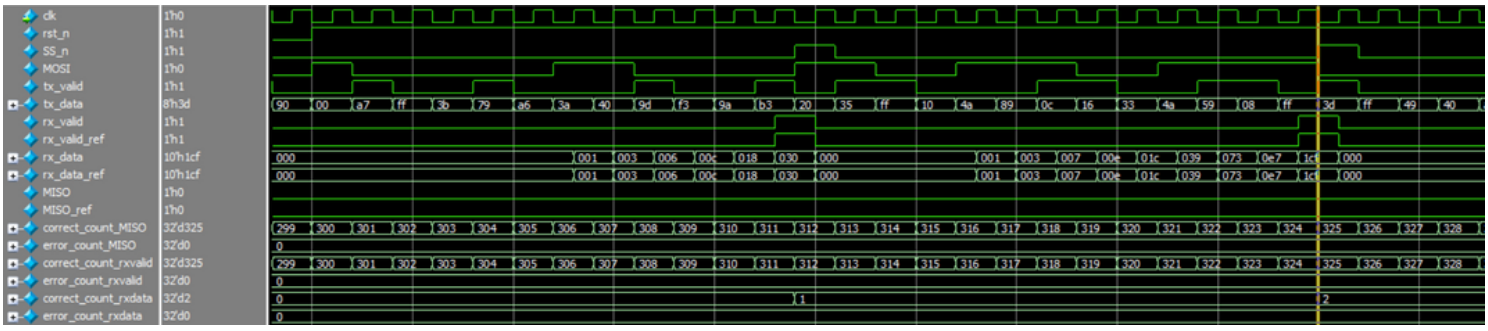
```
403     //rx_data @(rx_valid == 1) in WRITE state must start with 00 or 01
404     property rx_wr;
405         @(posedge intr.clk) disable iff(!intr.rst_n)
406
407         (intr.rx_valid && (cs_sva == WRITE_)) |-> ( (intr.rx_data[intr.ADDR_SIZE+1:intr.ADDR_SIZE] == 2'b00) || (intr.rx_data[intr.ADDR_SIZE+1:intr.ADDR_SIZE] == 2'b01) );
408     endproperty
409
410     //rx_data @(rx_valid == 1) in READ_ADD state must start with 10
411     property rx_rd_add;
412         @(posedge intr.clk) disable iff(!intr.rst_n)
413
414         (intr.rx_valid && (cs_sva == READ_ADD_)) |-> (intr.rx_data[intr.ADDR_SIZE+1:intr.ADDR_SIZE] == 2'b10);
415     endproperty
416
417     //rx_data @(rx_valid == 1) in READ_DATA state must start with 11
418     property rx_rd_data;
419         @(posedge intr.clk) disable iff(!intr.rst_n)
420
421         (intr.rx_valid && (cs_sva == READ_DATA_)) |-> (intr.rx_data[intr.ADDR_SIZE+1:intr.ADDR_SIZE] == 2'b11);
422     endproperty
423
424     //rx_valid cannot be high if tx_valid is high
425     property rx_tx;
426         @(posedge intr.clk) disable iff(!intr.rst_n)
427
428         ($rose(intr.tx_valid) && (cs == READ_DATA_)) |=> (~intr.rx_valid) throughout (~intr.tx_valid || (cs != READ_DATA))[->1];
429     endproperty
430
431     //rx_valid cannot be high at any checking state
432     property rx_chk;
433         @(posedge intr.clk) disable iff(!intr.rst_n)
434
435         ~( (cs == WRITE_) || (cs == READ_ADD_) || (cs == READ_DATA_) ) |-> (~intr.rx_valid);
436     endproperty


450     //MISO cannot be high at any state other than READ_DATA
451     property MISO_rd_only;
452         @(posedge intr.clk) disable iff(!intr.rst_n || intr.SS_n)
453
454         ~(cs_sva == READ_DATA_) |-> (~intr.MISO);
455     endproperty
```

Waveform Snippets

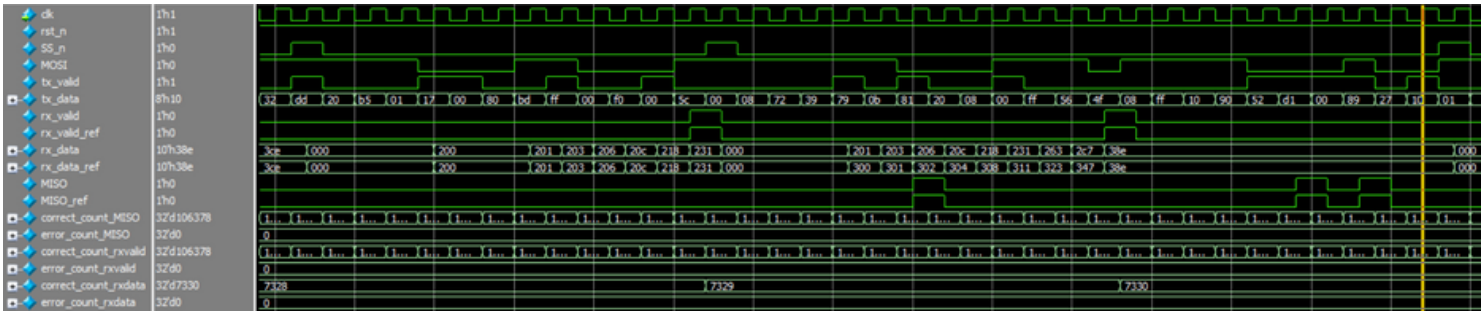
Reset Active: Outputs equal Zero



Write Operation



Read Operation



QUESTASIM SNIPPETS

Transcript

```
# UVM_INFO slave_test_pkg.sv(54) @ 600: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO slave_test_pkg.sv(56) @ 600: uvm_test_top [run_phase] Stimulus Generation Started
# UVM_INFO slave_test_pkg.sv(58) @ 600: uvm_test_top [run_phase] Testing Write-Only Operations
# UVM_INFO slave_test_pkg.sv(61) @ 156600: uvm_test_top [run_phase] Testing Read-Only Operations
# UVM_INFO slave_test_pkg.sv(64) @ 372600: uvm_test_top [run_phase] Testing Write-Read Operations
# UVM_INFO slave_test_pkg.sv(67) @ 465300: uvm_test_top [run_phase] Main Sequence (Complete Randomization)
# UVM_INFO slave_test_pkg.sv(70) @ 471300: uvm_test_top [run_phase] Stimulus Generation Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 471300: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO slave_scoreboard_pkg.sv(117) @ 471300: uvm_test_top.env_s.sb [report_phase] Total Successful Transactions -rx_data-: 13564
# UVM_INFO slave_scoreboard_pkg.sv(118) @ 471300: uvm_test_top.env_s.sb [report_phase] Total Failed Transactions -rx_data-: 0
# UVM_INFO slave_scoreboard_pkg.sv(119) @ 471300: uvm_test_top.env_s.sb [report_phase] Total Successful Transactions -rx_valid-: 235650
# UVM_INFO slave_scoreboard_pkg.sv(120) @ 471300: uvm_test_top.env_s.sb [report_phase] Total Failed Transactions -rx_valid-: 0
# UVM_INFO slave_scoreboard_pkg.sv(121) @ 471300: uvm_test_top.env_s.sb [report_phase] Total Successful Transactions -MISO-: 235650
# UVM_INFO slave_scoreboard_pkg.sv(122) @ 471300: uvm_test_top.env_s.sb [report_phase] Total Failed Transactions -MISO-: 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    : D:/Qsim_2021/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 471300 ns  Iteration: 61  Instance: /slave_top
```

Coverage Report

```
34  ================================================================================
35  === Instance: /slave_top/DUT
36  === Design Unit: work.slave
37  ================================================================================
38
39  Assertion Coverage:
40      Assertions                       17        17         0   100.00%


619 Directive Coverage:
620     Directives                       17        17         0   100.00%


79  Branch Coverage:
80      Enabled Coverage              Bins      Hits    Misses  Coverage
81      ----------------              ----      ----    ------  --------
82      Branches                        65        65         0   100.00%


320 Condition Coverage:
321     Enabled Coverage              Bins   Covered    Misses  Coverage
322     ----------------              ----      ----    ------  --------
323     Conditions                      32        32         0   100.00%


644 FSM Coverage:
645     Enabled Coverage              Bins      Hits    Misses  Coverage
646     ----------------              ----      ----    ------  --------
647     FSM States                       8         8         0   100.00%
648     FSM Transitions                 14        14         0   100.00%


705 Statement Coverage:
706     Enabled Coverage              Bins      Hits    Misses  Coverage
707     ----------------              ----      ----    ------  --------
708     Statements                      71        71         0   100.00%


1314 Toggle Coverage:
1315    Enabled Coverage              Bins      Hits    Misses  Coverage
1316    ----------------              ----      ----    ------  --------
1317    Toggles                         84        84         0   100.00%


3972 Covergroup Coverage:
3973    Covergroups                      1        na        na   100.00%
3974        Coverpoints/Crosses         10        na        na       na
3975            Covergroup Bins         28        28         0   100.00%
```

Verification Plan

GENERAL PLAN:

• **Checking Results**
- Output value is checked against golden model.
- Assertions are used for checking SPI Wrapper behavior.

• **Test Cases to Consider**
- Reset Signal (rst_n) Functionality across any combination of inputs.
- Normal Write Operation, 1st 2bits of MOSI = 2'b00, then 2'b01 after the next SS_n falling edge.
- Normal Read Operation, 1st 2bits of MOSI = 2'b10, then 2'b11 after the next SS_n falling edge.
- SPI's behavior on transitioning from one sequence to the other.
- SPI's behavior when SS_n stays active after transmission or goes inactive mid-transmission.

• **Functional Coverage Main Cover points and Cross Coverage**
- All RAM addresses are exercised.
- All Combinations of Control Signals are experienced.
- All addresses/data experienced the effect of rst_n signal.
- All addresses/data experienced the effect of SS_n signal.

• **Assertions**
- Output must equal zero on reset assertion or when SS_n is inactive.
- Output should only read the data received from RAM when tx_valid is high and should read it correctly.

Verification Requirements

Test Item	Description and Expected Output Behavior	Stimulus Generation
Reset Signal (rst_n)	SPI output (MISO) should reset to zero on reset assertion.	rst_n = 0. All other inputs are randomized under no constraints.
Control Signal (SS_n)	Slave and RAM should stay idle if SS_n is inactive and perform the intended behavior if SS_n is active. Slave and RAM should stay stable if SS_n remained active long after data transmission. Slave should go idle if SS_n is deactivated mid data transmission.	Included in all sequences where SS_n is either active or randomized to be active most of the time.
Write_Only Operation	SS_n should equal 0 throughout the cycle -> 1 after data is collected. Cycle(1): MOSI should equal 0 (WRITE) -> 00 (Send Address) Cycle(2): MOSI should equal 0 (WRITE) -> 01 (Send Data)	rst_n = 1. SS_n is directed to be activated at the beginning of each sequence and deactivated following data transmission. MOSI first 3bits are set to 3'b000 following SS_n activation for sending write address. MOSI first 3bits are set to 3'b001 following SS_n activation for sending write data. MOSI last 8bits are randomized under no constraints.
Read_Only Operation	SS_n should equal 0 throughout the cycle -> 1 after data is collected. Cycle(1): MOSI should equal 1 (READ) -> 10 (Send Address) Cycle(2): MOSI should equal 1 (READ) -> 11 (Receive Data) After [8:10] cycles, MISO bits start to serially read data from the required address in RAM.	rst_n = 1. SS_n is directed to be activated at the beginning of each sequence and deactivated following data transmission. MOSI first 3bits are set to 3'b110 following SS_n activation for sending read address. MOSI first 3bits are set to 3'b111 following SS_n activation and wait for receiving data. MOSI last 8bits are randomized under no constraints.
Write_Read_Operation	SS_n should equal 0 throughout the cycle -> 1 after data is collected. Sequences: (1): MOSI should equal 0 (WRITE) -> 00 (Send Address) (2): MOSI should equal 0 (WRITE) -> 01 (Send Data) (3): MOSI should equal 1 (READ) -> 10 (Send Address) (4): MOSI should equal 1 (READ) -> 11 (Read Data) Slave's behavior should not be affected by the order of different sequences. The slave can't go through two consecutive (READ_ADD) sequences or (READ_DATA) sequences.	rst_n = 1. SS_n is directed to be activated at the beginning of each sequence and deactivated following data transmission. MOSI bits are randomized under no constraints. Slave goes through different sequences (WRITE (Address or Data), READ_ADDR, READ_DATA) in a random manner.
Main Sequence	Same as (Write_Read_Operation) Description.	All inputs are randomized under constraints: - rst_n is inactive most of the time. - SS_n is active most of the time. to check for unexpected behavior.

Verification Requirements

Test Item	Functional Coverage	Functionality Check	Assertions
Reset Signal (rst_n)	Included in Cross Coverage: cross_rst_SS: All Combinations of Control Signals are covered. cross_rst_wr_add: All write addresses (sent by MOSI) experienced the effect of rst_n signal. cross_rst_wr_data: All write data (sent by MOSI) experienced the effect of rst_n signal. cross_rst_rd_add: All read addresses (sent by MOSI) experienced the effect of rst_n signal.		Property: reset_asserted
Control Signal (SS_n)	Included in Cross Coverage: cross_rst_SS: All Combinations of Control Signals are covered. cross_SS_wr_add: All write addresses (sent by MOSI) experienced the effect of SS_n signal. cross_SS_wr_data: All write data (sent by MOSI) experienced the effect of SS_n signal. cross_SS_rd_add: All read addresses (sent by MOSI) experienced the effect of SS_n signal.		Property: SS_inactive
Write_Only Operation	Included in Coverpoints: wr_address_cp: MOSI bits have taken the values required to target all RAM addresses in write operations. wr_data_cp: MOSI bits have taken the values required to write all combinations of data.	Output is checked against golden model.	
Read_Only Operation	Included in Coverpoints: rd_address_cp: MOSI bits have taken the values required to target all RAM addresses in read operations		Property: tx_inactive: MISO bits should equal zero when tx_valid is low MISO_out: Checks MISO reads the intended RAM address correctly.
Write_Read_Operation	Refer to (Write_Only_Operation) and (Read_Only_Operation) functional coverage.		Refer to (Reset Signal), (Control Signal) and (Read_Only_Operation) assertions.
Main Sequence	Refer to (Reset Signal) and (Control Signal) functional coverage. Refer to (Write_Only_Operation) and (Read_Only_Operation) functional coverage.		

Bug Report

Bug	Original Code	Fix	Lines (Original Version)	Lines (Edited Version)

Parameter (ADDR_SIZE) should be used instead of the actual size to respond correctly to changes in parameters	wire [7:0] tx_data1;
	wire [9:0] rx_data1;
		parameter ADDR_SIZE = 8;
		parameter MEM_DEPTH = 256;
		wire [ADDR_SIZE-1:0] tx_data1;
		wire [ADDR_SIZE+1:0] rx_data1;
			8
			10
				[18:19]
				24
				29

Code Snippets

TOP MODULE

```
1   import SPI_Wrapper_test_pkg::*;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4
5   module SPI_Wrapper_top;
6       bit clk;
7
8       initial begin
9           forever #1 clk = ~clk;
10      end
11
12      SPI_Wrapper_if SPI_Wrapperif (clk);
13      SPI_Wrapper SPI_WrapperDUT (SPI_Wrapperif);
14
15      SPI_Wrapper_ref_if SPI_Wrapper_refif (clk);
16      SPI_Wrapper_ref SPI_WrapperREF (SPI_Wrapper_refif);
17
18      slave_if slaveif (clk);
19      slave slaveDUT (slaveif);
20
21      slave_ref_if slaverefif (clk);
22      slave_ref slaveREF (slaverefif);
23
24      ram_if ramif (clk);
25      ram ramDUT (ramif);
26
27      ram_ref_if ram_refif (clk);
28      ram_ref ramREF (ram_refif);
29
30      //Design Connections
31      //Slave
32      assign slaveif.rst_n = SPI_Wrapperif.rst_n;
33      assign slaveif.SS_n = SPI_Wrapperif.SS_n;
34      assign slaveif.MOSI = SPI_Wrapperif.MOSI;
35      assign slaveif.tx_valid = SPI_WrapperDUT.tx_valid1;
36      assign slaveif.tx_data = SPI_WrapperDUT.tx_data1;
37      assign SPI_Wrapperif.MISO = slaveif.MISO;
38      assign SPI_WrapperDUT.rx_valid1 = slaveif.rx_valid;
39      assign SPI_WrapperDUT.rx_data1 = slaveif.rx_data;
40
41      //RAM
42      assign ramif.rst_n = SPI_Wrapperif.rst_n;
43      assign ramif.rx_valid = SPI_WrapperDUT.rx_valid1;
44      assign ramif.din = SPI_WrapperDUT.rx_data1;
45      assign SPI_WrapperDUT.tx_valid1 = ramif.tx_valid;
46      assign SPI_WrapperDUT.tx_data1 = ramif.dout;
```

Code Snippets

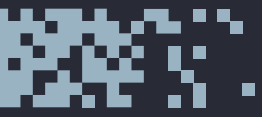
TOP MODULE

```
48      //Reference Model Connections
49      //Slave
50      assign slaverefif.rst_n = SPI_Wrapper_refif.rst_n;
51      assign slaverefif.SS_n = SPI_Wrapper_refif.SS_n;
52      assign slaverefif.MOSI = SPI_Wrapper_refif.MOSI;
53      assign slaverefif.tx_valid = SPI_WrapperREF.tx_valid;
54      assign slaverefif.tx_data = SPI_WrapperREF.tx_data;
55      assign SPI_Wrapper_refif.MISO_ref = slaverefif.MISO_ref;
56      assign SPI_WrapperREF.rx_valid = slaverefif.rx_valid_ref;
57      assign SPI_WrapperREF.rx_data = slaverefif.rx_data_ref;
58
59      //RAM
60      assign ram_refif.rst_n = SPI_Wrapper_refif.rst_n;
61      assign ram_refif.rx_valid = SPI_WrapperREF.rx_valid;
62      assign ram_refif.din = SPI_WrapperREF.rx_data;
63      assign SPI_WrapperREF.tx_valid = ram_refif.tx_valid_ref;
64      assign SPI_WrapperREF.tx_data = ram_refif.dout_ref;
65
66      initial begin
67          uvm_config_db#(virtual SPI_Wrapper_if)::set(null, "uvm_test_top", "SPI_Wrapper_IF", SPI_Wrapperif);
68          uvm_config_db#(virtual SPI_Wrapper_ref_if)::set(null, "uvm_test_top", "SPI_Wrapper_REF_IF", SPI_Wrapper_refif);
69          uvm_config_db#(virtual slave_if)::set(null, "uvm_test_top", "SLAVE_IF", slaveif);
70          uvm_config_db#(virtual slave_ref_if)::set(null, "uvm_test_top", "SLAVE_REF_IF", slaverefif);
71          uvm_config_db#(virtual ram_if)::set(null, "uvm_test_top", "RAM_IF", ramif);
72          uvm_config_db#(virtual ram_ref_if)::set(null, "uvm_test_top", "RAM_REF_IF", ram_refif);
73          run_test("SPI_Wrapper_test");
74      end
75
76  endmodule : SPI_Wrapper_top
```

INTERFACE

```
1   interface SPI_Wrapper_if (clk);
2
3       parameter ADDR_SIZE = 8;
4       parameter MEM_DEPTH = 256;
5
6       input clk;
7       bit rst_n, SS_n, MOSI, MISO;
8
9
10      modport DUT (output MISO,
11                   input clk, rst_n, SS_n, MOSI);
12
13  endinterface : SPI_Wrapper_if
```

Code Snippets



DESIGN

Before (Original)

```
module SPI_Wrapper(SPI_Wrapper_if.DUT intr);

    ////////////////////////////////////////////////////////////////////Original Code//////////////////////////////////////////////////////////////////////
/*
    input MOSI,clk,rst_n,SS_n;
    output MISO;

    wire [7:0] tx_data1;
    wire tx_valid1,rx_valid1;
    wire [9:0] rx_data1;
    slave #(ADDR_SIZE) s1(.MOSI(MOSI),.SS_n(SS_n),.clk(clk),.rst_n(rst_n),.tx_valid(tx_valid1),.tx_data(tx_data1),.rx_data(rx_data1),.rx_valid(rx_valid1),.MISO(MISO));
    ram #(.ADDR_SIZE(ADDR_SIZE), .MEM_DEPTH(MEM_DEPTH)) r1(.din(rx_data1),.clk(clk),.rst_n(rst_n),.rx_valid(rx_valid1),.dout(tx_data1),.tx_valid(tx_valid1));
*/
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

After (Edited)

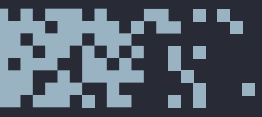
```
    ////////////////////////////////////////////////////////////////////Edited Code//////////////////////////////////////////////////////////////////////

    wire [intr.ADDR_SIZE-1:0] tx_data1;
    wire tx_valid1,rx_valid1;
    wire [intr.ADDR_SIZE+1:0] rx_data1;

    slave_if #(intr.ADDR_SIZE) s1 (intr.clk);
    ram_if #(.ADDR_SIZE(intr.ADDR_SIZE), .MEM_DEPTH(intr.MEM_DEPTH)) r1 (intr.clk);

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

CODE SNIPPETS

TEST

```
package SPI_Wrapper_test_pkg;
import uvm_pkg::*;
import SPI_Wrapper_env_pkg::*;
import slave_env_pkg::*;
import ram_env_pkg::*;
import SPI_Wrapper_config_pkg::*;
import slave_config_pkg::*;
import ram_config_pkg::*;
import SPI_Wrapper_sequence_pkg::*;
`include "uvm_macros.svh"

    class SPI_Wrapper_test extends uvm_test;
        `uvm_component_utils(SPI_Wrapper_test)

        SPI_Wrapper_env env;
        slave_env env_s;
        ram_env env_r;

        SPI_Wrapper_config SPI_Wrapper_cfg;
        slave_config slave_cfg;
        ram_config ram_cfg;

        SPI_Wrapper_reset_sequence reset_seq;
        SPI_Wrapper_write_only_sequence write_only_seq;
        SPI_Wrapper_read_only_sequence read_only_seq;
        SPI_Wrapper_write_read_sequence write_read_seq;
        SPI_Wrapper_main_sequence main_seq;


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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            env = SPI_Wrapper_env::type_id::create("env", this);
            env_s = slave_env::type_id::create("env_s", this);
            env_r = ram_env::type_id::create("env_r", this);

            SPI_Wrapper_cfg = SPI_Wrapper_config::type_id::create("SPI_Wrapper_cfg", this);
            slave_cfg = slave_config::type_id::create("slave_cfg", this);
            ram_cfg = ram_config::type_id::create("ram_cfg", this);

            reset_seq       = SPI_Wrapper_reset_sequence::type_id::create("reset_seq", this);
            write_only_seq  = SPI_Wrapper_write_only_sequence::type_id::create("write_only_seq", this);
            read_only_seq   = SPI_Wrapper_read_only_sequence::type_id::create("read_only_seq", this);
            write_read_seq  = SPI_Wrapper_write_read_sequence::type_id::create("write_read_seq", this);
            main_seq        = SPI_Wrapper_main_sequence::type_id::create("main_seq", this);

            SPI_Wrapper_cfg.active = UVM_ACTIVE;
            slave_cfg.active = UVM_PASSIVE;
            ram_cfg.active = UVM_PASSIVE;

            if(!uvm_config_db #(virtual SPI_Wrapper_if)::get(this, "", "SPI_Wrapper_IF", SPI_Wrapper_cfg.SPI_Wrapper_vif))
                `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the SPI_Wrapper (DUT) from the uvm_config_db");

            if(!uvm_config_db #(virtual SPI_Wrapper_ref_if)::get(this, "", "SPI_Wrapper_REF_IF", SPI_Wrapper_cfg.SPI_Wrapper_ref_vif))
                `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the SPI_Wrapper (REF) from the uvm_config_db");

            if(!uvm_config_db #(virtual slave_if)::get(this, "", "SLAVE_IF", slave_cfg.slave_vif))
                `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the Slave (DUT) from the uvm_config_db");

            if(!uvm_config_db #(virtual slave_ref_if)::get(this, "", "SLAVE_REF_IF", slave_cfg.slave_ref_vif))
                `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the Slave (REF) from the uvm_config_db");

            if(!uvm_config_db #(virtual ram_if)::get(this, "", "RAM_IF", ram_cfg.ram_vif))
                `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the RAM (DUT) from the uvm_config_db");

            if(!uvm_config_db #(virtual ram_ref_if)::get(this, "", "RAM_REF_IF", ram_cfg.ram_ref_vif))
                `uvm_fatal("build_phase", "Test - Unable to get the virtual interface of the RAM (REF) from the uvm_config_db");

            uvm_config_db #(SPI_Wrapper_config)::set(this, "env*", "SPI_Wrapper_CFG", SPI_Wrapper_cfg);
            uvm_config_db #(slave_config)::set(this, "env_s*", "SLAVE_CFG", slave_cfg);
            uvm_config_db #(ram_config)::set(this, "env_r*", "RAM_CFG", ram_cfg);

            //set_report_verbosity_level_hier(UVM_HIGH);
        endfunction
```

CODE SNIPPETS



TEST

```
79          task run_phase(uvm_phase phase);
80              super.run_phase(phase);
81              phase.raise_objection(this);
82              `uvm_info("run_phase", "Reset Asserted", UVM_LOW);
83              reset_seq.start(env.agt.sqr);
84              `uvm_info("run_phase", "Reset Deasserted", UVM_LOW);
85
86              `uvm_info("run_phase", "Stimulus Generation Started", UVM_LOW);
87
88              `uvm_info("run_phase", "Testing Write-Only Operations", UVM_LOW);
89              write_only_seq.start(env.agt.sqr);
90
91              `uvm_info("run_phase", "Testing Read-Only Operations", UVM_LOW);
92              read_only_seq.start(env.agt.sqr);
93
94              `uvm_info("run_phase", "Testing Write-Read Operations", UVM_LOW);
95              write_read_seq.start(env.agt.sqr);
96
97              `uvm_info("run_phase", "Main Sequence (Complete Randomization)", UVM_LOW);
98              main_seq.start(env.agt.sqr);
99
100             `uvm_info("run_phase", "Stimulus Generation Ended", UVM_LOW);
101             phase.drop_objection(this);
102         endtask : run_phase
103
104     endclass : SPI_Wrapper_test
105
106 endpackage : SPI_Wrapper_test_pkg
```

CONFIGURATION OBJECT

```
1   package SPI_Wrapper_config_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4
5       class SPI_Wrapper_config extends uvm_object;
6           `uvm_object_utils(SPI_Wrapper_config)
7
8           virtual SPI_Wrapper_if SPI_Wrapper_vif;
9           virtual SPI_Wrapper_ref_if SPI_Wrapper_ref_vif;
10          uvm_active_passive_enum active;
11
12          function new(string name = "SPI_Wrapper_config");
13              super.new(name);
14          endfunction : new
15
16      endclass : SPI_Wrapper_config
17
18  endpackage : SPI_Wrapper_config_pkg
```

Code Snippets

ENVIRONMENT

```
1   package SPI_Wrapper_env_pkg;
2   import SPI_Wrapper_agent_pkg::*;
3   import SPI_Wrapper_scoreboard_pkg::*;
4   import SPI_Wrapper_coverage_pkg::*;
5   import shared_pkg::*;
6   import uvm_pkg::*;
7   `include "uvm_macros.svh"
8
9       class SPI_Wrapper_env extends uvm_env;
10          `uvm_component_utils(SPI_Wrapper_env)
11
12          SPI_Wrapper_agent agt;
13          SPI_Wrapper_scoreboard sb;
14          SPI_Wrapper_coverage cov;
15
16          function new(string name = "SPI_Wrapper_env", uvm_component parent = null);
17              super.new(name, parent);
18          endfunction : new
19
20          function void build_phase(uvm_phase phase);
21              super.build_phase(phase);
22              agt = SPI_Wrapper_agent::type_id::create("agt", this);
23              sb = SPI_Wrapper_scoreboard::type_id::create("sb", this);
24              cov = SPI_Wrapper_coverage::type_id::create("cov", this);
25          endfunction
26
27          function void connect_phase(uvm_phase phase);
28              super.connect_phase(phase);
29              agt.agt_ap.connect(sb.sb_export);
30              agt.agt_ap.connect(cov.cov_export);
31          endfunction : connect_phase
32
33      endclass : SPI_Wrapper_env
34
35  endpackage : SPI_Wrapper_env_pkg
```

CODE SNIPPETS

AGENT

```
package SPI_Wrapper_agent_pkg;
import uvm_pkg::*;
import SPI_Wrapper_driver_pkg::*;
import SPI_Wrapper_sequencer_pkg::*;
import SPI_Wrapper_monitor_pkg::*;
import SPI_Wrapper_config_pkg::*;
import SPI_Wrapper_sequence_item_pkg::*;

`include "uvm_macros.svh"

    class SPI_Wrapper_agent extends uvm_agent;
        `uvm_component_utils(SPI_Wrapper_agent)

        SPI_Wrapper_driver drv;
        SPI_Wrapper_sequencer sqr;
        SPI_Wrapper_monitor mon;
        SPI_Wrapper_config SPI_Wrapper_cfg;
        uvm_analysis_port #(SPI_Wrapper_sequence_item) agt_ap;

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            SPI_Wrapper_cfg = SPI_Wrapper_config::type_id::create("SPI_Wrapper_cfg", this);
            if(!uvm_config_db#(SPI_Wrapper_config)::get(this, "", "SPI_Wrapper_CFG", SPI_Wrapper_cfg))
                `uvm_fatal("build_phase", "Agent - Unable to get SPI_Wrapper Configuration Object.");

            if(SPI_Wrapper_cfg.active) begin
                drv = SPI_Wrapper_driver::type_id::create("drv", this);
                sqr = SPI_Wrapper_sequencer::type_id::create("sqr", this);
            end
            mon = SPI_Wrapper_monitor::type_id::create("mon", this);

            agt_ap = new("agt_ap", this);
        endfunction : build_phase

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            if(SPI_Wrapper_cfg.active) begin
                drv.SPI_Wrapper_drv_vif = SPI_Wrapper_cfg.SPI_Wrapper_vif;
                drv.SPI_Wrapper_ref_drv_vif = SPI_Wrapper_cfg.SPI_Wrapper_ref_vif;
                drv.seq_item_port.connect(sqr.seq_item_export);
            end
            mon.SPI_Wrapper_mon_vif = SPI_Wrapper_cfg.SPI_Wrapper_vif;
            mon.SPI_Wrapper_ref_mon_vif = SPI_Wrapper_cfg.SPI_Wrapper_ref_vif;
            mon.mon_ap.connect(agt_ap);
        endfunction : connect_phase

    endclass : SPI_Wrapper_agent

endpackage : SPI_Wrapper_agent_pkg
```

CODE SNIPPETS

SEQUENCE ITEM

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

    class SPI_Wrapper_sequence_item extends uvm_sequence_item;
        `uvm_object_utils(SPI_Wrapper_sequence_item)

        rand bit rst_n, SS_n, MOSI;

        bit MISO, MISO_ref;

        randc bit [ADDR_SIZE-1:0] txn;

        bit MOSI_txn_wr_add, MOSI_txn_wr_data, MOSI_txn_rd_add, MOSI_txn_rd_data;
        int i;

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

        function string convert2string();
            return $sformatf("%s    rst_n = 0b%0b,  SS_n = 0b%0b,   MOSI = 0b%0b,   MISO = 0b%0b",
                             super.convert2string(), rst_n, SS_n, MOSI, MISO);
        endfunction : convert2string

        function string convert2string_stimulus();
            return $sformatf("rst_n = 0b%0b,    SS_n = 0b%0b,   MOSI = 0b%0b",
                             rst_n, SS_n, MOSI);
        endfunction : convert2string_stimulus

        function void pre_randomize();
            if(i == 0) begin
                MOSI_txn_wr_add = 0;
                MOSI_txn_wr_data = 0;
                MOSI_txn_rd_add = 1;
                MOSI_txn_rd_data = 1;
            end
        endfunction : pre_randomize

        //Constraints
        constraint rst_c {
            rst_n           dist {0:=2,     1:=98};             //rst_n is inactive most of the time
        }

        constraint SS_c {
            SS_n            dist {0:=95,    1:=5};              //SS_n is active most of the time
        }

        constraint MOSI_wr_add_c {
            MOSI == MOSI_txn_wr_add;
        }

        constraint MOSI_wr_data_c {
            MOSI == MOSI_txn_wr_data;
        }

        constraint MOSI_rd_add_c {
            MOSI == MOSI_txn_rd_add;
        }

        constraint MOSI_rd_data_c {
            MOSI == MOSI_txn_rd_data;
        }

        function void post_randomize();
            if(i == 0) begin
                temp_txn_wr_add = {2'b00, txn};
                temp_txn_wr_data = {2'b01, txn};
                temp_txn_rd_add = {2'b10, txn};
                temp_txn_rd_data = {2'b11, txn};
            end
            else begin
                MOSI_txn_wr_add = temp_txn_wr_add[ADDR_SIZE + 1 - i];
                MOSI_txn_wr_data = temp_txn_wr_data[ADDR_SIZE + 1 - i];
                MOSI_txn_rd_add = temp_txn_rd_add[ADDR_SIZE + 1 - i];
                MOSI_txn_rd_data = temp_txn_rd_data[ADDR_SIZE + 1 - i];
            end
            if(i > ADDR_SIZE + 1) i = 0;
            else i++;
        endfunction : post_randomize

    endclass : SPI_Wrapper_sequence_item

endpackage : SPI_Wrapper_sequence_item_pkg
```

CODE SNIPPETS

SEQUENCE

```
1   package SPI_Wrapper_sequence_pkg;
2   import uvm_pkg::*;
3   import SPI_Wrapper_sequence_item_pkg::*;
4   import shared_pkg::*;
5   `include "uvm_macros.svh"
6
7   parameter TESTS = 3000;
8
9       class SPI_Wrapper_reset_sequence extends uvm_sequence #(SPI_Wrapper_sequence_item);
10          `uvm_object_utils(SPI_Wrapper_reset_sequence)
11
12          SPI_Wrapper_sequence_item seq_item;
13
14          function new(string name = "SPI_Wrapper_reset_sequence");
15              super.new(name);
16          endfunction : new
17
18          task body();
19              seq_item = SPI_Wrapper_sequence_item::type_id::create("seq_item");
20              seq_item.constraint_mode(0);
21              repeat(TESTS) begin
22                  start_item(seq_item);
23                  assert(seq_item.randomize() with {rst_n == 0;});
24                  finish_item(seq_item);
25              end
26          endtask : body
27
28      endclass : SPI_Wrapper_reset_sequence
29
30
31
32      class SPI_Wrapper_write_only_sequence extends uvm_sequence #(SPI_Wrapper_sequence_item);
33          `uvm_object_utils(SPI_Wrapper_write_only_sequence)
34
35          SPI_Wrapper_sequence_item seq_item;
36
37          function new(string name = "SPI_Wrapper_write_only_sequence");
38              super.new(name);
39          endfunction : new
40
41          task body();
42              repeat(TESTS) begin
43                  write_add_seq(seq_item);
44                  write_data_seq(seq_item);
45              end
46          endtask : body
47
178     class SPI_Wrapper_read_only_sequence extends uvm_sequence #(SPI_Wrapper_sequence_item);
179         `uvm_object_utils(SPI_Wrapper_read_only_sequence)
180
181         SPI_Wrapper_sequence_item seq_item;
182
183         function new(string name = "SPI_Wrapper_read_only_sequence");
184             super.new(name);
185         endfunction : new
186
187         task body();
188             repeat(TESTS) begin
189                 read_add_seq(seq_item);
190                 read_data_seq(seq_item);
191             end
192         endtask : body
```

CODE SNIPPETS

SEQUENCE

```
252     class SPI_Wrapper_write_read_sequence extends uvm_sequence #(SPI_Wrapper_sequence_item);
253         `uvm_object_utils(SPI_Wrapper_write_read_sequence)
254
255         SPI_Wrapper_sequence_item seq_item;
256
257         function new(string name = "SPI_Wrapper_write_read_sequence");
258             super.new(name);
259         endfunction : new
260
261         task body();
262             repeat(TESTS) begin
263                 case($urandom_range(0,3))
264                     0: write_add_seq(seq_item);
265                     1: write_data_seq(seq_item);
266                     2: read_add_seq(seq_item);
267                     3: read_data_seq(seq_item);
268                 endcase //seq_item.selector_seq
269             end
270         endtask : body

358     class SPI_Wrapper_main_sequence extends uvm_sequence #(SPI_Wrapper_sequence_item);
359         `uvm_object_utils(SPI_Wrapper_main_sequence)
360
361         SPI_Wrapper_sequence_item seq_item;
362
363         function new(string name = "SPI_Wrapper_main_sequence");
364             super.new(name);
365         endfunction : new
366
367         task body();
368             seq_item = SPI_Wrapper_sequence_item::type_id::create("seq_item");
369             seq_item.constraint_mode(0);
370             seq_item.rst_c.constraint_mode(1);
371             seq_item.SS_c.constraint_mode(1);
372             repeat(TESTS) begin
373                 start_item(seq_item);
374                 assert(seq_item.randomize());
375                 finish_item(seq_item);
376             end
377         endtask : body
378
379     endclass : SPI_Wrapper_main_sequence
380
381 endpackage : SPI_Wrapper_sequence_pkg
```

SEQUENCER

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

Code Snippets

DRIVER

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

    class SPI_Wrapper_driver extends uvm_driver #(SPI_Wrapper_sequence_item);
        `uvm_component_utils(SPI_Wrapper_driver)

        virtual SPI_Wrapper_if SPI_Wrapper_drv_vif;
        virtual SPI_Wrapper_ref_if SPI_Wrapper_ref_drv_vif;

        SPI_Wrapper_sequence_item stim_seq_item;

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                stim_seq_item = SPI_Wrapper_sequence_item::type_id::create("stim_seq_item");
                seq_item_port.get_next_item(stim_seq_item);
                //DUT
                SPI_Wrapper_drv_vif.rst_n       = stim_seq_item.rst_n;
                SPI_Wrapper_drv_vif.SS_n        = stim_seq_item.SS_n;
                SPI_Wrapper_drv_vif.MOSI        = stim_seq_item.MOSI;

                //REF
                SPI_Wrapper_ref_drv_vif.rst_n       = stim_seq_item.rst_n;
                SPI_Wrapper_ref_drv_vif.SS_n        = stim_seq_item.SS_n;
                SPI_Wrapper_ref_drv_vif.MOSI        = stim_seq_item.MOSI;

                @(negedge SPI_Wrapper_drv_vif.clk);
                seq_item_port.item_done();
                `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH);
            end
        endtask : run_phase

    endclass : SPI_Wrapper_driver

endpackage : SPI_Wrapper_driver_pkg
```

CODE SNIPPETS

MONITOR

```
package SPI_Wrapper_monitor_pkg;
import uvm_pkg::*;
import shared_pkg::*;
import SPI_Wrapper_sequence_item_pkg::*;

`include "uvm_macros.svh"

    class SPI_Wrapper_monitor extends uvm_monitor;
        `uvm_component_utils(SPI_Wrapper_monitor)

        SPI_Wrapper_sequence_item rsp_seq_item;
        virtual SPI_Wrapper_if SPI_Wrapper_mon_vif;
        virtual SPI_Wrapper_ref_if SPI_Wrapper_ref_mon_vif;
        uvm_analysis_port #(SPI_Wrapper_sequence_item) mon_ap;

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

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

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

            forever begin
                rsp_seq_item = SPI_Wrapper_sequence_item::type_id::create("rsp_seq_item");
                @(negedge SPI_Wrapper_mon_vif.clk);
                rsp_seq_item.rst_n          = SPI_Wrapper_mon_vif.rst_n;
                rsp_seq_item.SS_n           = SPI_Wrapper_mon_vif.SS_n;
                rsp_seq_item.MOSI           = SPI_Wrapper_mon_vif.MOSI;
                rsp_seq_item.MISO           = SPI_Wrapper_mon_vif.MISO;
                rsp_seq_item.MISO_ref       = SPI_Wrapper_ref_mon_vif.MISO_ref;

                mon_ap.write(rsp_seq_item);
                `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH);
            end
        endtask : run_phase

    endclass : SPI_Wrapper_monitor

endpackage : SPI_Wrapper_monitor_pkg
```

CODE SNIPPETS

SCOREBOARD

```
package SPI_Wrapper_scoreboard_pkg;
import uvm_pkg::*;
import shared_pkg::*;
import SPI_Wrapper_sequence_item_pkg::*;

`include "uvm_macros.svh"

    class SPI_Wrapper_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(SPI_Wrapper_scoreboard);

        SPI_Wrapper_sequence_item sb_seq_item;
        uvm_analysis_export #(SPI_Wrapper_sequence_item) sb_export;
        uvm_tlm_analysis_fifo #(SPI_Wrapper_sequence_item) sb_fifo;

        int correct_count, error_count;


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

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

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


        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(sb_seq_item);

                if(sb_seq_item.MISO !== sb_seq_item.MISO_ref) begin
                    `uvm_error("run_phase", $sformatf("Comparison Failed, Transaction received by DUT: %s, while the reference output -MISO-:0b%0b", sb_seq_item.conv
                    error_count++;
                end
                else begin
                    `uvm_info("run_phase", $sformatf("Correct SPI_Wrapper output -MISO-: %s", sb_seq_item.convert2string()), UVM_HIGH);
                    correct_count++;
                end
            end

        endtask : run_phase

        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("report_phase", $sformatf("Total Successful Transactions: %0d", correct_count), UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("Total Failed Transactions: %0d", error_count), UVM_MEDIUM);
        endfunction : report_phase

    endclass : SPI_Wrapper_scoreboard

endpackage : SPI_Wrapper_scoreboard_pkg
```

CODE SNIPPETS

COVERAGE

```
package SPI_Wrapper_coverage_pkg;
import uvm_pkg::*;
import shared_pkg::*;
import SPI_Wrapper_sequence_item_pkg::*;

`include "uvm_macros.svh"

    class SPI_Wrapper_coverage extends uvm_component;
        `uvm_component_utils(SPI_Wrapper_coverage);

        uvm_analysis_export #(SPI_Wrapper_sequence_item) cov_export;
        uvm_tlm_analysis_fifo #(SPI_Wrapper_sequence_item) cov_fifo;

        SPI_Wrapper_sequence_item cov_seq_item;

        covergroup cg;

            //All addresses are exercised
            wr_address_cp:      coverpoint temp_txn_wr_add[ADDR_SIZE-1:0];
            rd_address_cp:      coverpoint temp_txn_rd_add[ADDR_SIZE-1:0];

            //All Combinations of data are covered
            wr_data_cp:         coverpoint temp_txn_wr_data[ADDR_SIZE-1:0];
            rd_data_cp:         coverpoint temp_txn_rd_data[ADDR_SIZE-1:0];

            //All Combinations of Control Signals are covered
            cross_rst_SS:       cross cov_seq_item.rst_n, cov_seq_item.SS_n, cov_seq_item.MOSI;

            //All addresses/data experienced the effect of rst_n signal
            cross_rst_wr_add:   cross cov_seq_item.rst_n, wr_address_cp;
            cross_rst_wr_data:  cross cov_seq_item.rst_n, wr_data_cp;
            cross_rst_rd_add:   cross cov_seq_item.rst_n, rd_address_cp;
            cross_rst_rd_data:  cross cov_seq_item.rst_n, rd_data_cp;

            //All addresses/data experienced the effect of SS_n signal
            cross_SS_wr_add:    cross cov_seq_item.SS_n, wr_address_cp;
            cross_SS_wr_data:   cross cov_seq_item.SS_n, wr_data_cp;
            cross_SS_rd_add:    cross cov_seq_item.SS_n, rd_address_cp;
            cross_SS_rd_data:   cross cov_seq_item.SS_n, rd_data_cp;

        endgroup

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

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

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cov_export.connect(cov_fifo.analysis_export);
        endfunction : connect_phase

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(cov_seq_item);
                cg.sample();
            end
        endtask : run_phase

    endclass : SPI_Wrapper_coverage

endpackage : SPI_Wrapper_coverage_pkg
```

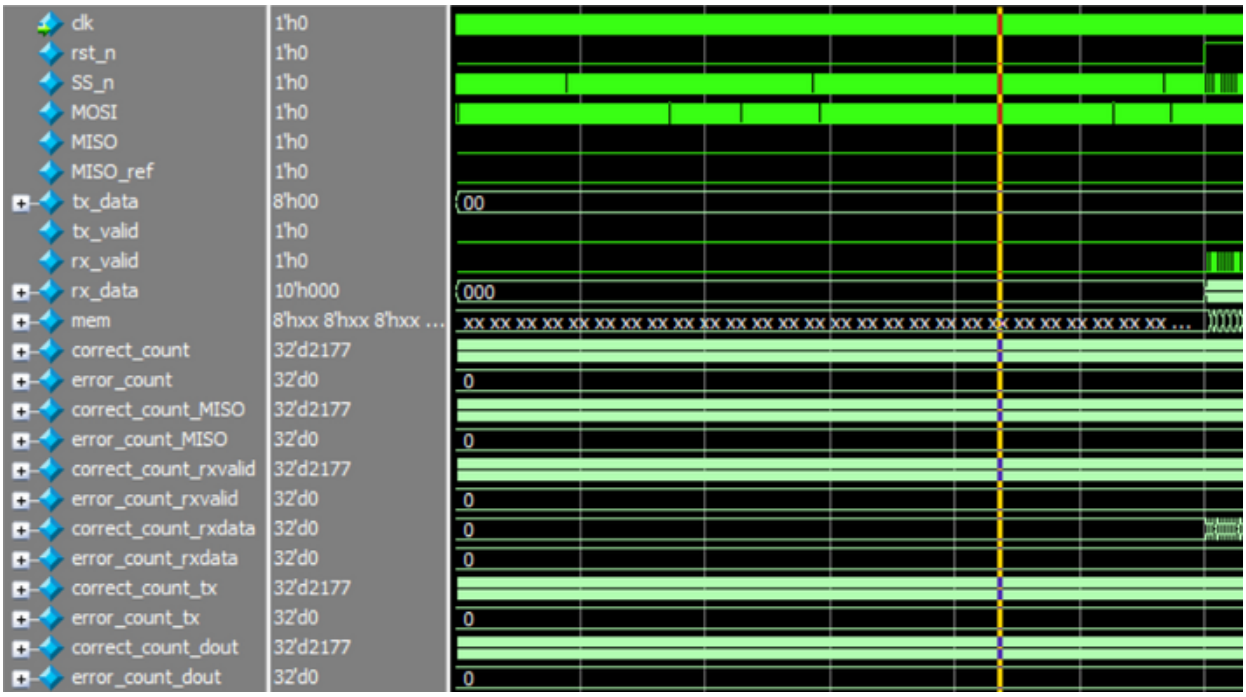
CODE SNIPPETS

ASSERTIONS

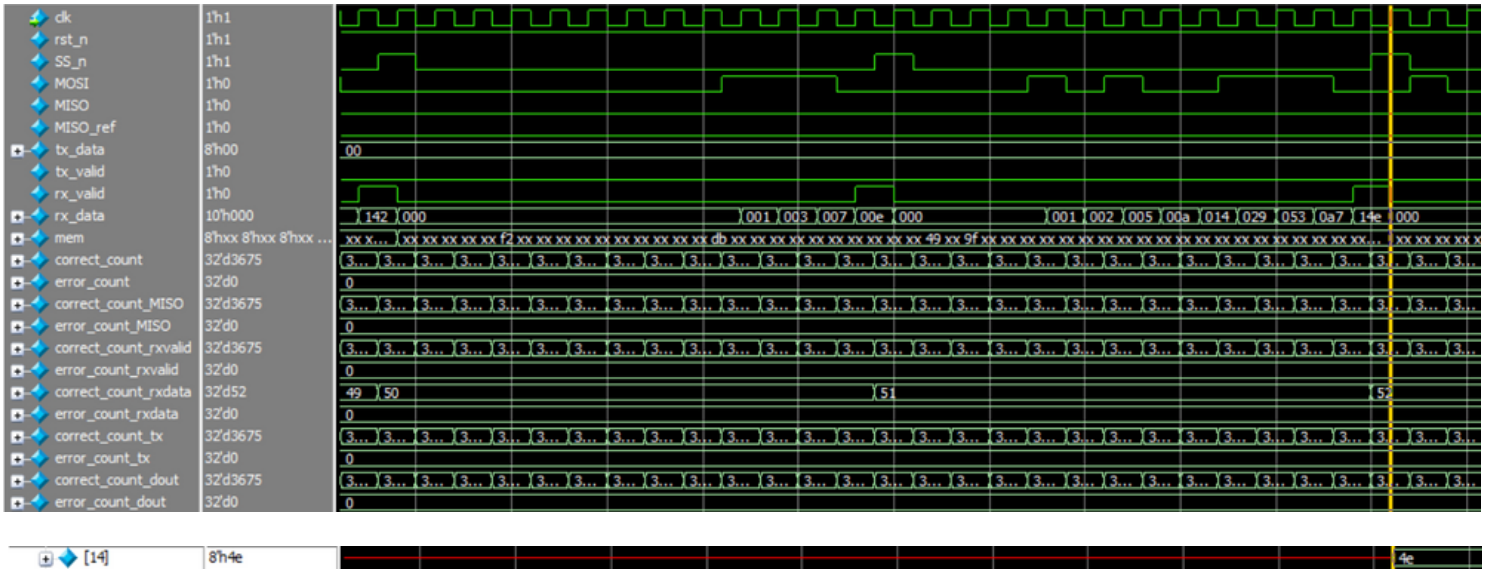
```
27      `ifdef SIM
28          property reset_asserted;
29              @(posedge intr.clk)
30              $fell(intr.rst_n) |=> ~intr.MISO;
31          endproperty
32
33          property SS_inactive;
34              @(posedge intr.clk) disable iff(!intr.rst_n)
35              $rose(intr.SS_n) |=> ~intr.MISO;
36          endproperty
37
38          property tx_inactive;
39              @(posedge intr.clk) disable iff(!intr.rst_n)
40              !tx_valid1 |=> ~intr.MISO;
41          endproperty
42
43          property MISO_out;
44              int i = 0;
45              @(posedge intr.clk) disable iff(!intr.rst_n || $rose(intr.SS_n, @(posedge intr.clk)))
46              $fell(intr.SS_n) ##[intr.ADDR_SIZE:intr.ADDR_SIZE+5] $rose(tx_valid1) |=> ((intr.MISO == tx_data1[intr.ADDR_SIZE-1-i]), i++)[*(intr.ADDR_SIZE)];
47          endproperty
```

Waveform Snippets

Reset Active: Output equal Zero



Write Operation



Waveform Snippets

Read Operation

Transcript

```
# UVM_INFO SPI_Wrapper_test_pkg.sv(84) @ 6000: uvm_test_top [run_phase] Reset Deasserted
# UVM_INFO SPI_Wrapper_test_pkg.sv(86) @ 6000: uvm_test_top [run_phase] Stimulus Generation Started
# UVM_INFO SPI_Wrapper_test_pkg.sv(88) @ 6000: uvm_test_top [run_phase] Testing Write-Only Operations
# UVM_INFO SPI_Wrapper_test_pkg.sv(91) @ 162000: uvm_test_top [run_phase] Testing Read-Only Operations
# UVM_INFO SPI_Wrapper_test_pkg.sv(94) @ 384000: uvm_test_top [run_phase] Testing Write-Read Operations
# UVM_INFO SPI_Wrapper_test_pkg.sv(97) @ 477906: uvm_test_top [run_phase] Main Sequence (Complete Randomization)
# UVM_INFO SPI_Wrapper_test_pkg.sv(100) @ 483906: uvm_test_top [run_phase] Stimulus Generation Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 483906: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO SPI_Wrapper_scoreboard_pkg.sv(52) @ 483906: uvm_test_top.env.sb [report_phase] Total Successful Transactions: 241953
# UVM_INFO SPI_Wrapper_scoreboard_pkg.sv(53) @ 483906: uvm_test_top.env.sb [report_phase] Total Failed Transactions: 0
# UVM_INFO ram_scoreboard_pkg.sv(62) @ 483906: uvm_test_top.env_r.sb [report_phase] Total Successful Transactions -dout-: 241953
# UVM_INFO ram_scoreboard_pkg.sv(63) @ 483906: uvm_test_top.env_r.sb [report_phase] Total Failed Transactions -dout-: 0
# UVM_INFO ram_scoreboard_pkg.sv(64) @ 483906: uvm_test_top.env_r.sb [report_phase] Total Successful Transactions -tx_valid-: 241953
# UVM_INFO ram_scoreboard_pkg.sv(65) @ 483906: uvm_test_top.env_r.sb [report_phase] Total Failed Transactions -tx_valid-: 0
# UVM_INFO slave_scoreboard_pkg.sv(117) @ 483906: uvm_test_top.env_s.sb [report_phase] Total Successful Transactions -rx_data-: 14596
# UVM_INFO slave_scoreboard_pkg.sv(118) @ 483906: uvm_test_top.env_s.sb [report_phase] Total Failed Transactions -rx_data-: 0
# UVM_INFO slave_scoreboard_pkg.sv(119) @ 483906: uvm_test_top.env_s.sb [report_phase] Total Successful Transactions -rx_valid-: 241953
# UVM_INFO slave_scoreboard_pkg.sv(120) @ 483906: uvm_test_top.env_s.sb [report_phase] Total Failed Transactions -rx_valid-: 0
# UVM_INFO slave_scoreboard_pkg.sv(121) @ 483906: uvm_test_top.env_s.sb [report_phase] Total Successful Transactions -MISO-: 241953
# UVM_INFO slave_scoreboard_pkg.sv(122) @ 483906: uvm_test_top.env_s.sb [report_phase] Total Failed Transactions -MISO-: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   24
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]    12
# [run_phase]     8
# ** Note: $finish    : D:/Qsim_2021/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 483906 ns  Iteration: 61  Instance: /SPI_Wrapper_top
```

Coverage Report

```
13031    TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 3

13086    TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 29
```

SPI_WRAPPER

```
89    ================================================================================
90    === Instance: /SPI_Wrapper_top/SPI_WrapperDUT
91    === Design Unit: work.SPI_Wrapper
92    ================================================================================
93
94    Assertion Coverage:
95        Assertions                        4         4         0   100.00%

129   Toggle Coverage:
130       Enabled Coverage             Bins      Hits    Misses  Coverage
131       ----------------             ----      ----    ------  --------
132       Toggles                        40        40         0   100.00%
```

Coverage Report

SLAVE

```
294  ================================================================================
295  === Instance: /SPI_Wrapper_top/slaveDUT
296  === Design Unit: work.slave
297  ================================================================================
298
299  Assertion Coverage:
300      Assertions                        17        17         0   100.00%

339  Branch Coverage:
340      Enabled Coverage             Bins      Hits    Misses  Coverage
341      ----------------             ----      ----    ------  --------
342      Branches                       64        64         0   100.00%

578  Condition Coverage:
579      Enabled Coverage             Bins   Covered    Misses  Coverage
580      ----------------             ----      ----    ------  --------
581      Conditions                     31        31         0   100.00%

899  FSM Coverage:
900      Enabled Coverage             Bins      Hits    Misses  Coverage
901      ----------------             ----      ----    ------  --------
902      FSM States                      8         8         0   100.00%
903      FSM Transitions                14        14         0   100.00%

960  Statement Coverage:
961      Enabled Coverage             Bins      Hits    Misses  Coverage
962      ----------------             ----      ----    ------  --------
963      Statements                     70        70         0   100.00%

1569  Toggle Coverage:
1570      Enabled Coverage            Bins      Hits    Misses  Coverage
1571      ----------------            ----      ----    ------  --------
1572      Toggles                       84        84         0   100.00%
```

Coverage Report

RAM

```
2608  ================================================================================
2609  === Instance: /SPI_Wrapper_top/ramDUT
2610  === Design Unit: work.ram
2611  ================================================================================
2612
2613  Assertion Coverage:
2614      Assertions                        8        8        0   100.00%

2635  Branch Coverage:
2636      Enabled Coverage              Bins      Hits    Misses  Coverage
2637      ----------------              ----      ----    ------  --------
2638      Branches                         7         7         0   100.00%

2686  Statement Coverage:
2687      Enabled Coverage              Bins      Hits    Misses  Coverage
2688      ----------------              ----      ----    ------  --------
2689      Statements                      15        15         0   100.00%

2863  Toggle Coverage:
2864      Enabled Coverage              Bins      Hits    Misses  Coverage
2865      ----------------              ----      ----    ------  --------
2866      Toggles                         52        52         0   100.00%
```