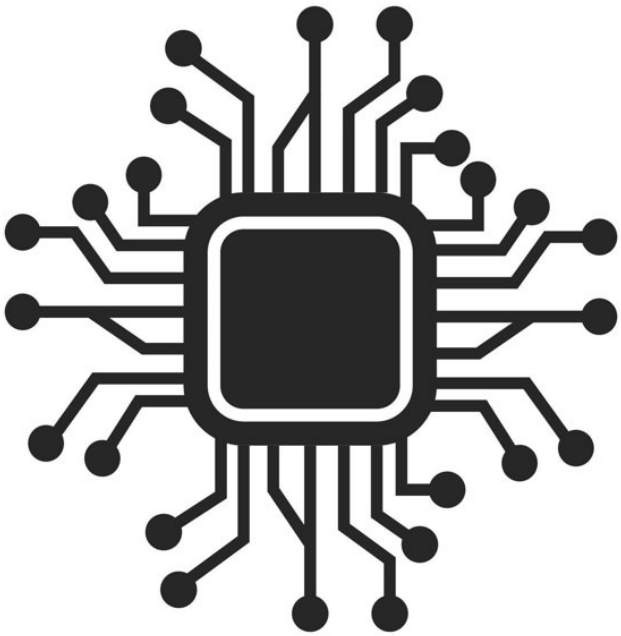
# Digital Verification using SV & UVM

## UVM Project: SPI Slave with a Single Port RAM



Prepared by

**Omar Mohamed Hussein Mostafa**

# Table of Contents

1. OVERVIEW ..... 4

2. SPI SLAVE UVM VERIFICATION ..... 5

2.1. Verification Requirements ..... 5
2.2. Verification Plan ..... 6
2.3. SPI Interface ..... 7
2.4. SPI Slave Design ..... 8
2.4.1. Design Bugs ..... 10
2.4.2. Design Assertions ..... 10
2.5. SPI Slave Design Golden Model ..... 11
2.5.1. Counter Module ..... 11
2.5.2. SPI Slave Module ..... 12
2.6. UVM Diagram ..... 15
2.7. SPI Sequence item ..... 16
2.8. SPI Sequencer ..... 18
2.9. SPI Configuration Object ..... 18
2.10. SPI Reset Sequence ..... 19
2.11. SPI Main Sequence ..... 20
2.12. SPI Driver ..... 21
2.13. SPI Monitor ..... 22
2.14. SPI Agent ..... 23
2.15. SPI Scoreboard ..... 24
2.16. SPI Cover ..... 25
2.17. SPI Environment ..... 26
2.18. SPI Test ..... 27
2.19. SPI Top ..... 28
2.20. SPI Source File ..... 29
2.21. SPI DO File ..... 30
2.22. SPI Assertions ..... 31
2.23. Simulation Results ..... 32
2.23.1. Questasim Transcript ..... 32
2.23.2. Questasim Code Coverage ..... 32
2.23.3. Questasim Functional Coverage ..... 34
2.23.4. Questasim Assertions ..... 34
2.23.5. Questasim Assertions Coverage ..... 34
2.23.6. Questasim Waveform ..... 35

3. SINGLE PORT SYNCHRONOUS RAM UVM VERIFICATION ..... 36

3.1. Verifications Requirements ..... 36
3.2. RAM Verification Plan ..... 38
3.3. RAM Interface ..... 38
3.4. RAM Design ..... 39
3.4.1. Design Bugs ..... 39

3.5. RAM Design Golden Model ..... 40
3.6. UVM Diagram ..... 41
3.7. RAM Sequence item ..... 42
3.8. RAM Sequencer ..... 43
3.9. RAM Configuration Object ..... 44
3.10. RAM Sequences ..... 45
3.10.1. RAM Reset Sequence ..... 45
3.10.2. RAM Write Sequence ..... 45
3.10.3. RAM Read Sequence ..... 46
3.10.4. RAM Read Write Sequence ..... 46
3.11. RAM Driver ..... 47
3.12. RAM Monitor ..... 48
3.13. RAM Agent ..... 49
3.14. RAM Scoreboard ..... 50
3.15. RAM Cover Collector ..... 51
3.16. RAM Environment ..... 52
3.17. RAM Test ..... 53
3.18. RAM Top ..... 54
3.19. RAM Source File ..... 55
3.20. RAM DO File ..... 55
3.21. RAM Assertions ..... 56
3.22. Simulation Results ..... 56
3.22.1. Questasim Transcript ..... 56
3.22.2. Questasim Code Coverage ..... 57
3.22.3. Questasim Functional Coverage ..... 58
3.22.4. Questasim Assertions ..... 59
3.22.5. Questasim Assertions Coverage ..... 59
3.22.6. Questasim Waveform ..... 59

4. WRAPPER UVM VERIFICATION ..... 60

4.1. Verification Requirements ..... 60
4.2. Wrapper Design ..... 61
4.3. Wrapper Interface ..... 61
4.4. Verification Plan ..... 61
4.5. UVM Diagram ..... 62
4.6. Golden Model ..... 64
4.7. Wrapper Sequence item ..... 65
4.8. Wrapper Configuration Object ..... 67
4.9. Wrapper Sequences ..... 68
4.9.1. Reset Sequence ..... 68
4.9.2. Write Only Sequence ..... 68
4.9.3. Read Only Sequence ..... 69
4.9.4. Read Write Sequence ..... 70
4.10. Wrapper sequencer ..... 70
4.11. Wrapper Driver ..... 71
4.12. Wrapper Monitor ..... 72

4.13. Wrapper Agent ..... 73
4.14. Wrapper Scoreboard ..... 74
4.15. Wrapper Environment ..... 75
4.16. SPI Sequence Item ..... 76
4.17. SPI Configuration Object ..... 77
4.18. SPI Monitor ..... 78
4.19. SPI Agent ..... 79
4.20. SPI Scoreboard ..... 80
4.21. SPI Cover Collector ..... 81
4.22. SPI Environment ..... 82
4.23. SPI Assertions ..... 83
4.24. RAM Sequence Item ..... 84
4.25. RAM Configuration Object ..... 85
4.26. RAM Monitor ..... 86
4.27. RAM Agent ..... 87
4.28. RAM Scoreboard ..... 88
4.29. RAM Cover Collector ..... 89
4.30. RAM Environment ..... 90
4.31. RAM Assertions ..... 91
4.32. Wrapper Test ..... 92
4.33. Wrapper Top Module ..... 93
4.34. Source Files ..... 94
4.35. Do File ..... 95
4.36. Wrapper Simulation Results ..... 96
4.36.1. Questasim Transcript ..... 96
4.36.2. Questasim Code Coverage ..... 96
4.36.3. Questasim Functional Coverage ..... 97
4.36.4. Questasim Assertions ..... 98
4.36.5. Questasim Assertions Coverage ..... 98
4.36.6. Questasim Waveforms ..... 99

# *1.Overview*

This project focuses on the functional verification of an SPI Slave integrated with a Single-Port Synchronous RAM using the Universal Verification Methodology (UVM). The main objective was to ensure correct communication and data integrity between the SPI interface and the RAM through a structured and reusable UVM-based verification environment. The design under verification (DUV) consisted of three main components: the SPI Slave, the RAM, and a Wrapper module. The SPI Slave implemented the SPI protocol in slave mode, receiving serial data from a master, decoding commands, and managing read and write operations. The Single-Port Synchronous RAM served as the memory element, performing synchronous read and write operations with a shared clock and control signals. The Wrapper module connected both components, translating SPI transactions into corresponding RAM operations, and served as the integration layer for the testbench.

The work was divided into three main stages: verification of the SPI Slave, verification of the RAM, and verification of the integrated Wrapper module. In the first two stages, each block was verified independently using a complete UVM environment with active agents capable of generating stimulus and driving the interfaces. The SPI Slave environment verified the correct implementation of SPI protocol operations such as read, write, and command decoding, while the RAM environment validated proper memory behavior for synchronous read and write cycles. In the final stage, both the SPI and RAM modules were integrated into a top-level Wrapper, which acted as the communication bridge between the two. For this integration, both SPI and RAM environments were instantiated as passive agents within the Wrapper verification environment to observe their transactions without driving them directly. This setup allowed end-to-end monitoring of data flow between SPI and RAM, ensuring correct command translation, data integrity, and protocol synchronization. Assertions and functional coverage were added to confirm timing accuracy, command sequencing, and full functional operation. Overall, the project demonstrated a complete, hierarchical UVM-based verification flow that transitioned from block-level active verification to system-level passive integration, reflecting professional verification practices in complex digital systems.

# *2.SPI Slave UVM Verification*

## 2.1.Verification Requirements

**Sequences Requirements:**

Split the sequences into the following sequences. You can switch on and off the constraints or use inline constraints to control the sequence item generated in the sequence

- reset_sequence
- main_sequence

**Constraint Requirements:**

1- The reset signal (rst_n) shall be deasserted most of the time.

2- The SS_n signal to be high for one cycle every 13 cycles for all cases except read data to be high for one cycle every 23 cycles

3- Declare a randomized array of 11 bits to drive bit by bit the MOSI and the first 3 bits sent serially to the MOSI when the SS_n falls are valid combinations only (000, 001, 110, 111)

4- The tx_valid signal to be high in case of read data

***(Hint: You can use post_randomize for points 2 and 3)***

**Functional Coverage Requirements:**

1- Add coverpoints on rx_data[9:8] to take all possible values and all possible transitions

2- Add coverpoints on SS_n to capture:

- Check full transaction duration: 1 → 0 [*13] → 1 for normal operations.
- Check extended transaction: 1 → 0 [*23] → 1 for READ_DATA.

3- Add coverpoints on MOSI to validate correct transitions:

- 000 (Write Address)
- 001 (Write Data)
- 110 (Read Address)
- 111 (Read Data)

4- Cross coverage between SS_n and MOSI bins (Exclude irrelevant bins to focus on legal operation scenarios)

**Assertions Requirements:**

1- An assertion ensures that whenever reset is asserted, the outputs (MISO, rx_valid, and rx_data) are all low.

2- An assertion checks that after any valid command sequence (write_add_seq (000), write_data_seq (001), read_add_seq(110), or read_data_seq(111)), the rx_valid signal must assert exactly after 10 cycles and the SS_n should eventually after the 10 cycles to close communication.

3- Add the following assertions in the design to check correct FSM transitions and guard the assertions using conditional compilation with the `ifdef directive with macro named "SIM", and then include the macro in the vlog command +define+SIM option in the vlog command of your do file. Refer to this link to learn more about conditional compilation.

- IDLE → CHK_CMD
- CHK_CMD → WRITE or READ_ADD or READ_DATA
- WRITE → IDLE
- READ_ADD → IDLE
- READ_DATA → IDLE

**Note:** You are free to add assertions, covergroups or constraints to enrich your verification.

# 2.2.Verification Plan

| Label | Design Requirment Description | Stimulus generation | Functional coverage | Functionality check |
|---|---|---|---|---|
| reset check | when the reset signal is activated , the MISO ,rx_valid,rx_data should be zero. | directed reset assertion using reset sequence | | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the spi_sva file. |
| IDLE | when the reset signal is deactivated and the SS_n is High , the cs should be IDLE | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. | Covering the Transition of the SS_n after 13 cycle except for the READ_DATA would be for 23 cycle | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the spi_sva file. |
| CHK_CMD | when the reset signal is deactivated and the SS_n is Low and the cs is IDLE, the cs should be CHK_CMD | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. | Covering the Transition of the SS_n after 13 cycle except for the READ_DATA would be for 23 cycle | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the spi_sva file. |
| WRITE_ADDR | when the reset signal is deactivated and the SS_n is Low and the cs is CHK_CMD and the MOSI is taking the following sequence 3'b000, the cs should be WRITE ADDRESS. | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. Also Randomizing variable called valid_comb to have the most significant 3 bits with valid combinations such as WRITE_ADD , WRITE_DATA , READ_ADD , WRITE_DATA. | Covering the Transition of the MOSI to cover the sequence of the WRITE_ADDR, WRITE_DATA, READ_ADDR, READ_DATA and crossing it with the SS_n coverpoint. | In the scoreboard , comparing the outputs with the expected of a Golden model. |
| WRITE_DATA | when the reset signal is deactivated and the SS_n is Low and the cs is CHK_CMD and the MOSI is taking the following sequence 3'b001, the cs should be WRITE DATA. | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. Also Randomizing variable called valid_comb to have the most significant 3 bits with valid combinations such as WRITE_ADD , WRITE_DATA , READ_ADD , WRITE_DATA. | Covering the Transition of the MOSI to cover the sequence of the WRITE_ADDR, WRITE_DATA, READ_ADDR, READ_DATA and crossing it with the SS_n coverpoint. | In the scoreboard , comparing the outputs with the expected of a Golden model. |
| READ_ADD | when the reset signal is deactivated and the SS_n is Low and the cs is CHK_CMD and the MOSI is taking the following sequence 3'b110, the cs should be WRITE ADDRESS. | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. Also Randomizing variable called valid_comb to have the most significant 3 bits with valid combinations such as WRITE_ADD , WRITE_DATA , READ_ADD , WRITE_DATA. | Covering the Transition of the MOSI to cover the sequence of the WRITE_ADDR, WRITE_DATA, READ_ADDR, READ_DATA and crossing it with the SS_n coverpoint. | In the scoreboard , comparing the outputs with the expected of a Golden model. |
| READ_DATA | when the reset signal is deactivated and the SS_n is Low and the cs is CHK_CMD and the MOSI is taking the following sequence 3'b111, the cs should be WRITE ADDRESS. | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. Also Randomizing variable called valid_comb to have the most significant 3 bits with valid combinations such as WRITE_ADD , WRITE_DATA , READ_ADD , WRITE_DATA. | Covering the Transition of the MOSI to cover the sequence of the WRITE_ADDR, WRITE_DATA, READ_ADDR, READ_DATA and crossing it with the SS_n coverpoint. | In the scoreboard , comparing the outputs with the expected of a Golden model. |

## 2.3.SPI Interface

```
interface spi_if(clk);
    input clk;
    logic MOSI, SS_n, rst_n;
    logic MISO,MISO_exp;
    logic tx_valid,rx_valid,rx_valid_exp;
    logic [9:0] rx_data,rx_data_exp;
    logic [7:0] tx_data;

endinterface
```

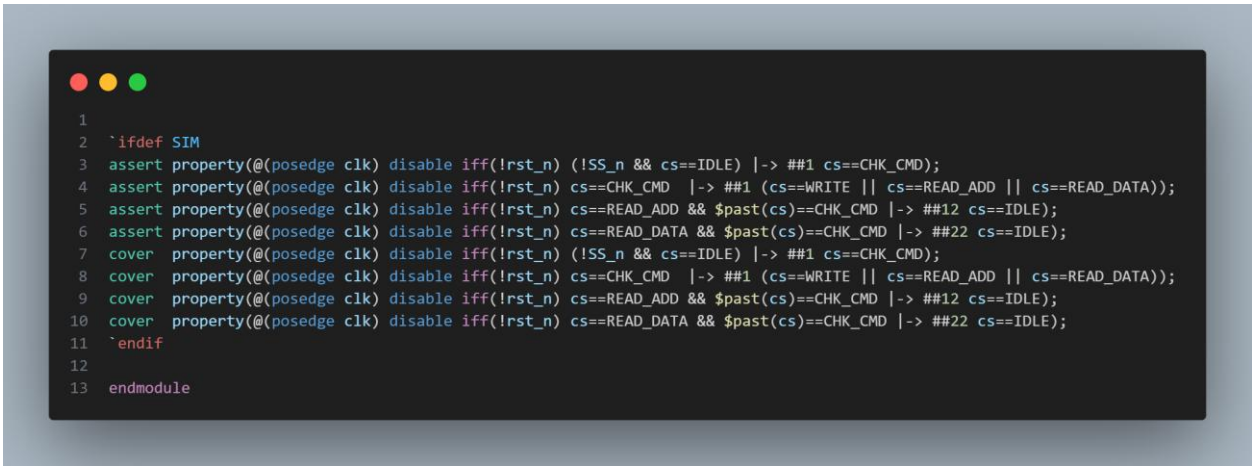
# 2.4.SPI Slave Design

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

localparam IDLE      = 3'b000;
localparam WRITE     = 3'b010;
localparam CHK_CMD   = 3'b001;     //Design Modification => making the CHK_CMD paramatrized with 2 instead of the WRITE for readiability
localparam READ_ADD  = 3'b011;
localparam READ_DATA = 3'b100;

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

reg [3:0] counter;
reg       received_address;

reg [2:0] cs, ns;

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

always @(*) begin
    case (cs)
        IDLE : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = CHK_CMD;
        end
        CHK_CMD : begin
            if (SS_n)
                ns = IDLE;
            else begin
                if (~MOSI)
                    ns = WRITE;
                else begin
                    if (!received_address)          //Design Error : should check for not finishing Read_add
                        ns = READ_ADD;
                    else
                        ns = READ_DATA;
                end
            end
        end
        WRITE : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = WRITE;
        end
        READ_ADD : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = READ_ADD;
        end

```

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

always @(posedge clk) begin
    if (~rst_n) begin
        rx_data <= 0;
        rx_valid <= 0;
        received_address <= 0;
        MISO <= 0;
    end
    else begin
        case (cs)
            IDLE : begin
                rx_valid <= 0;
                MISO     <= 0;      //Design Error: MISO must be reseted
                rx_data  <= 0;      //Design Error: should reset the rx_data

            end
            CHK_CMD : begin
                counter <= 10;
            end
            WRITE : begin
                if (counter > 0) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
                else begin
                    rx_valid <= 1;
                end
            end
            READ_ADD : begin
                if (counter > 0) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
                else begin
                    rx_valid <= 1;
                    received_address <= 1;
                end
            end
            READ_DATA : begin
                if (tx_valid) begin
                    rx_valid <= 0;
                    if (counter > 0) begin
                        MISO <= tx_data[counter-1];
                        counter <= counter - 1;
                    end
                    else begin
                        MISO<=0;
                        received_address <= 0;
                    end
                end
                else begin
                    MISO<=0;
                    if (counter > 0) begin
                        rx_data[counter-1] <= MOSI;
                        counter <= counter - 1;
                    end
                    else begin
                        rx_valid <= 1;
                        counter <= 9;     //Design Error: the counter should be reloaded to 9 as the slave would wait one cycle for the tx_valid
                    end
                end
            end
        endcase
    end
end

```

### 2.4.1.Design Bugs

- **In line 43:** The condition should check for not received_address to remain in the READ_ADDR , else should Transition to the READ_DATA.
- **In line 21:** The design should reset the MOSI in the IDLE state.
- **In line 22:** The design should reset the rx_data in the IDLE state.
- **In line 67:** The design should should reload the counter to 9 as there will be a waiting cycle before the tx_valid comes and the Data gets out starting from address 7 in the tx_data.



```
1
2  `ifdef SIM
3  assert property(@(posedge clk) disable iff(!rst_n) (!SS_n && cs==IDLE) |-> ##1 cs==CHK_CMD);
4  assert property(@(posedge clk) disable iff(!rst_n) cs==CHK_CMD  |-> ##1 (cs==WRITE || cs==READ_ADD || cs==READ_DATA));
5  assert property(@(posedge clk) disable iff(!rst_n) cs==READ_ADD && $past(cs)==CHK_CMD |-> ##12 cs==IDLE);
6  assert property(@(posedge clk) disable iff(!rst_n) cs==READ_DATA && $past(cs)==CHK_CMD |-> ##22 cs==IDLE);
7  cover  property(@(posedge clk) disable iff(!rst_n) (!SS_n && cs==IDLE) |-> ##1 cs==CHK_CMD);
8  cover  property(@(posedge clk) disable iff(!rst_n) cs==CHK_CMD  |-> ##1 (cs==WRITE || cs==READ_ADD || cs==READ_DATA));
9  cover  property(@(posedge clk) disable iff(!rst_n) cs==READ_ADD && $past(cs)==CHK_CMD |-> ##12 cs==IDLE);
10 cover  property(@(posedge clk) disable iff(!rst_n) cs==READ_DATA && $past(cs)==CHK_CMD |-> ##22 cs==IDLE);
11 `endif
12
13 endmodule
```

### 2.4.2.Design Assertions

The ‘isdef SIM is a compile time condition that protects this part of the code from synthesis.

**Assertions Table:**

| Feature | Assertions |
|---|---|
| If the reset is deactivated with the SS_n is low in the IDLE case, the cs should be CHK_CMD in the next cycle | (@(posedge clk) disable iff(!rst_n) (!SS_n && cs==IDLE) \|-> ##1 cs==CHK_CMD) |
| If the reset is deactivated with the SS_n is low in the CHK_CMD state, the ns should be WRITE or READ_ADDR or READ_DATA | @(posedge clk) disable iff(!rst_n) cs==CHK_CMD \|-> ##1 (cs==WRITE \|\| cs==READ_ADD \|\| cs==READ_DATA) |
| If the reset is deactivated with the SS_n is low in the WRITE state and the previous state was CHK_CMD , the state should be IDLE after 12 cycles. | (@(posedge clk) disable iff(!rst_n) cs==READ_ADD && $past(cs)==CHK_CMD \|-> ##12 cs==IDLE) |
| If the reset is deactivated with the SS_n is low in the READ_DATA state and the previous state was CHK_CMD , the state should be IDLE after 22 cycles. | @(posedge clk) disable iff(!rst_n) cs==READ_DATA && $past(cs)==CHK_CMD \|-> ##22 cs==IDLE) |

# 2.5.SPI Slave Design Golden Model

## 2.5.1.Counter Module

```
module down_counter#(parameter MOD=8,parameter COUNTER_BITS=3)(clk,rstn,enable,count);

input clk,rstn,enable;
output reg [COUNTER_BITS-1:0] count;

always @(posedge clk ) begin
    if(~rstn) begin
        count<=MOD-1;
    end
    else if(enable) begin
        if(count==0)begin
            count<=MOD-1;
        end
        if(count>0) begin
           count<=count-1;
        end
    end
end
endmodule
```

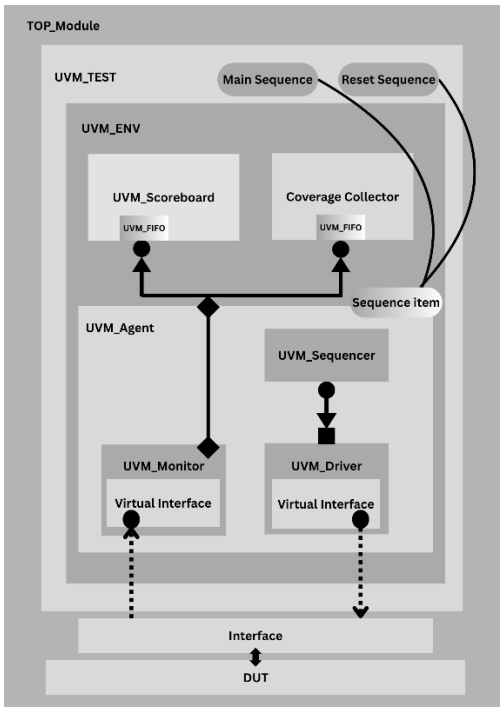
## 2.5.2.SPI Slave Module

```
module SPI_slave_op(MOSI,MISO,clk,rstn,rx_data,tx_data,rx_valid,tx_valid,SS_n);

//our state encoding;
localparam IDLE      = 3'b000;
localparam CHK_CMD   = 3'b001;
localparam WRITE     = 3'b010;
localparam READ_ADD  = 3'b011;
localparam READ_DATA = 3'b100;


//SPI slave ports
input MOSI,clk,rstn,tx_valid,SS_n;
input [7:0]tx_data;
output reg MISO,rx_valid;
output reg [9:0]rx_data;

(*fsm_encoding="sequential"*)
reg [2:0]ns,cs;                //next and current state

//internal signals
wire [3:0]count;
wire [3:0]count_10;
reg flag,Read_add_data_inx,counter_enable,counter_8_enable;

//counters
down_counter #(11,4) down_counter_10(clk,rstn && cs!=CHK_CMD ,counter_enable,count_10); //counter counts from 10 to 0 ,11 states
down_counter #(10,4)tx_to_MISO(clk,rstn&& (cs!=CHK_CMD),counter_8_enable,count);       //counter counts from 9 to 0, 10 states

//next state logic
always @(*) begin
    case (cs)
        IDLE:begin
            if(SS_n) ns=IDLE;
            else begin
                ns=CHK_CMD;
            end
        end
        CHK_CMD:begin
            if(SS_n) ns=IDLE;
            else begin
                if(MOSI) begin
                    if(!Read_add_data_inx) begin//read addr
                        ns=READ_ADD;
                    end
                    else if (Read_add_data_inx) begin
                        ns=READ_DATA;
                    end
                end
                else begin
                    ns=WRITE;
                end
            end
        end
        WRITE:begin
            if(SS_n) ns=IDLE;
            else begin
                ns=WRITE;
            end
        end
        READ_ADD:begin
            if(SS_n) ns=IDLE;
            else begin
                ns=READ_ADD;
            end
        end
        READ_DATA: begin
            if(SS_n) ns=IDLE;
            else begin
                ns=READ_DATA;
            end
        end
    endcase
end

```

```
1
2  //memory logic
3  always @(posedge clk) begin
4      if (!rstn) begin
5          cs<=3'b000;
6      end
7      else begin
8          cs<=ns;
9      end
10 end
11
12 //output logic
13 always @(posedge clk) begin
14     if(!rstn) begin
15         rx_data<=8'b0;
16         rx_valid<=0;
17         MISO<=0;
18         Read_add_data_inx<=0;
19         flag<=0;
20         counter_enable<=0;
21         counter_8_enable<=0;
22     end
23     else begin
24         case(cs)
25             WRITE: begin
26                 if(count_10>4'b0) begin
27                     rx_data[count_10-1]<=MOSI;
28                     MISO<=0;
29                     if(count_10==1) counter_enable<=0;
30                 end
31                 else if (count_10==4'b0) begin
32                    rx_valid<=1;
33                 end
34             end
35             READ_ADD: begin
36                 if(count_10>4'b0) begin
37                     rx_data[count_10-1]<=MOSI;
38                     MISO<=0;
39                     if(count_10==1) counter_enable<=0;
40                 end
41                 else if (count_10==4'b0) begin
42                    rx_valid<=1;
43                     Read_add_data_inx<=1;
44                 end
45             end
46             READ_DATA: begin
47                 if(!tx_valid) begin
48                     MISO<=0;
49                     if(count_10>4'b0) begin
50                         if(flag) begin
51                             rx_data[count_10-2]<=MOSI;
52                             MISO<=0;
53                         end
54                         else begin
55                             rx_data[count_10-1]<=MOSI;
56                             MISO<=0;
57                         end
58                     end
59                     else if (count_10==4'b0) begin
60                        rx_valid<=1;
61                        flag<=1;
62                        counter_8_enable<=1;
63                     end
64                 end
65                 else begin
66                     rx_valid<=0;
67                     if(count>0) begin
68                         MISO<=tx_data[count-1]; // getting the data out of the SPI through the MISO
69                         if(count==1) counter_8_enable<=0;
70                     end
71
```

```
            end
            CHK_CMD :begin
                rx_valid<=0;
                flag<=0;
                counter_8_enable<=0;
            end
            IDLE: begin
                rx_valid<=0; // reset all values to start communication again
                MISO<=0;
                rx_data<=0;
                counter_enable<=1;
            end
            default :begin
                rx_valid<=0;
                MISO<=0;
            end
        endcase
    end
end
endmodule

```

# 2.6.UVM Diagram

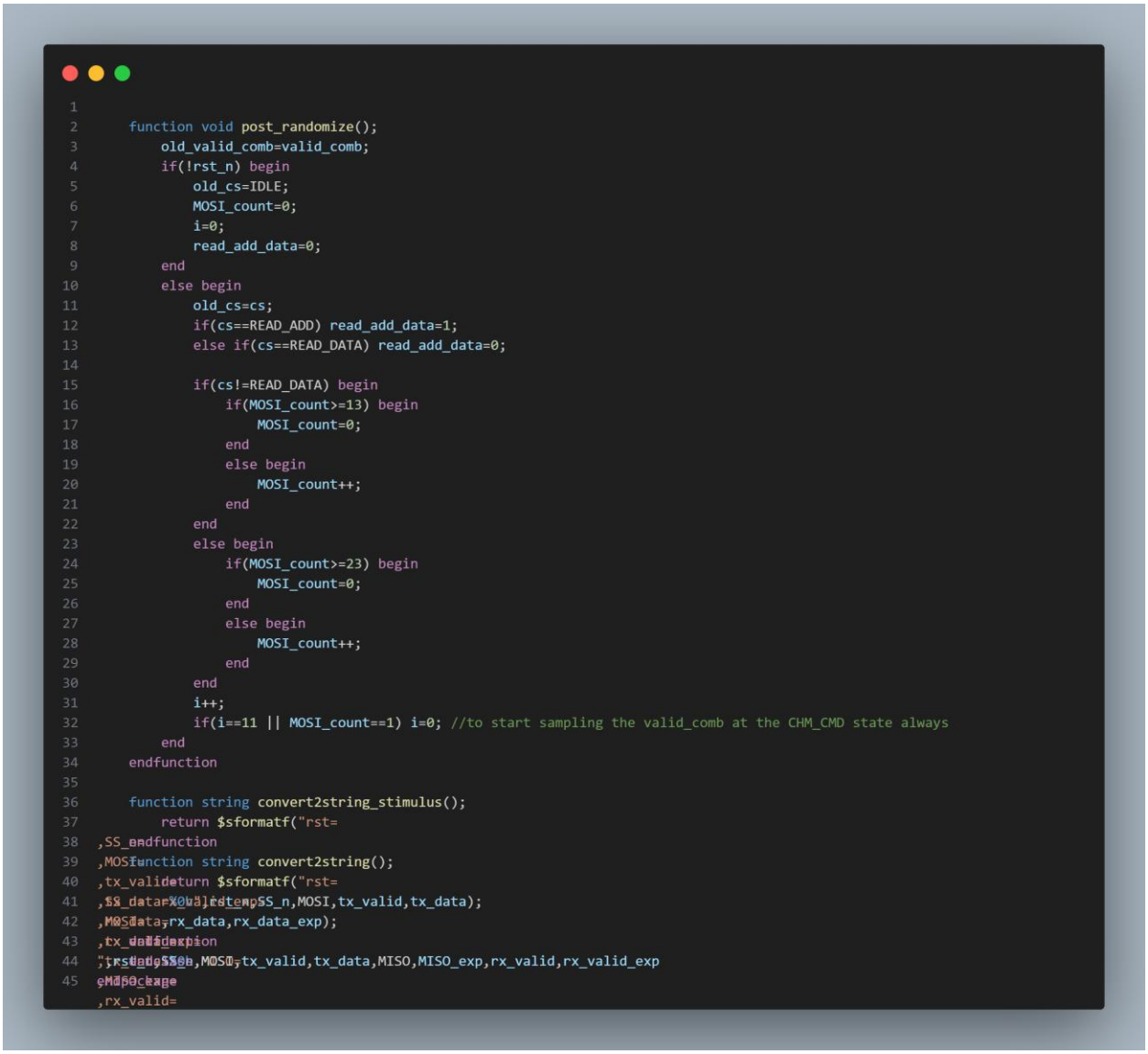



**Verification Flow:**

- TOP Module instantiates the interface, Design, Golden Model and bind assertions, then it sets the interface in the configuration object then runs the test.
- Test creates the environment and starts sequences: Main Sequence, Reset Sequence.
- Environment Creates the Agent, Scoreboard and the Cover Collector.
- Also Connecting the Agent analysis port with the Scoreboard and Cover Collector analysis export ports.
- Agent creates sequencer, driver and the monitor , connecting the sequencer port with the driver and connecting the analysis port of the monitor with the analysis port of the Agent.
- The Driver uses a virtual interface to drive DUT pins.
- The Monitor samples the same virtual interface to observe pin behavior and rebuild transaction objects.
- The UVM_Scoreboard and Coverage Collector receive transaction objects produced by monitors through TLM analysis channels and perform checking and coverage collection.

# 2.7.SPI Sequence item

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Object : Sequence Item
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

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

    class spi_seq_item extends uvm_sequence_item;
    `uvm_object_utils(spi_seq_item)
    //Constructor function
    function new(string name="spi_seq_item");
        super.new(name);
    endfunction

    typedef enum  {IDLE,CHK_CMD,WRITE,READ_ADD,READ_DATA} my_FSM;
    //Interface signals
    //Inputs
    rand logic  MOSI, rst_n, SS_n;
    rand logic  [7:0] tx_data;
    rand logic tx_valid;
    rand logic [10:0] valid_comb;
    //Outputs
    logic rx_valid,rx_valid_exp;
    logic [9:0] rx_data,rx_data_exp;
    logic MISO,MISO_exp;
    //Internal signals and sequence item counters
    logic [10:0] old_valid_comb;
    int MOSI_count,i,read_add_data;
    my_FSM  old_cs;
    rand my_FSM  cs;
    //Constrain Blocks
    constraint reset_constrains{rst_n dist{1:/95 , 0:/5};}
    constraint SS_n_constrains {
        if (old_cs != READ_DATA ) {
          (MOSI_count == 13) -> SS_n == 1;
          (MOSI_count != 13) -> SS_n == 0;
        }
        else if(old_cs==READ_DATA) {
          (MOSI_count == 23) -> SS_n == 1;
          (MOSI_count != 23) -> SS_n == 0;
        }
    }
    constraint cs_constrains{if(!rst_n || SS_n) {cs==IDLE;}
                             else if(old_cs==IDLE && !SS_n) {cs==CHK_CMD;}
                             else if(old_cs==CHK_CMD && !SS_n && !MOSI) {cs==WRITE;}
                             else if(old_cs==CHK_CMD && !SS_n && MOSI && read_add_data==0) {cs==READ_ADD;}
                             else if(old_cs==CHK_CMD && !SS_n && MOSI && read_add_data==1) {cs==READ_DATA;}
                             else if((old_cs==WRITE || old_cs==READ_ADD) && MOSI_count==13 ) {cs==IDLE;}
                             else if((old_cs==WRITE) && MOSI_count!=13 ) {cs==WRITE;}
                             else if((old_cs==READ_ADD) && MOSI_count!=13 ) {cs==READ_ADD;}
                             else if((old_cs==READ_DATA) && MOSI_count==23 ) {cs==IDLE;}
                             else if((old_cs==READ_DATA) && MOSI_count!=23 ) {cs==READ_DATA;}
    }
    constraint valid_comb_constrains{
        if(!rst_n || MOSI_count==0) valid_comb[10:8] inside{3'b000,3'b001,3'b110,3'b111};
        else valid_comb==old_valid_comb;}
    constraint MOSI_constrains {if(!SS_n && rst_n) MOSI==valid_comb[10-i] ;}
    constraint tx_valid_constrains{if(MOSI_count>=14 && MOSI_count<=22) {tx_valid==1;} else{tx_valid==0;}}

```



```
function void post_randomize();
    old_valid_comb=valid_comb;
    if(!rst_n) begin
        old_cs=IDLE;
        MOSI_count=0;
        i=0;
        read_add_data=0;
    end
    else begin
        old_cs=cs;
        if(cs==READ_ADD) read_add_data=1;
        else if(cs==READ_DATA) read_add_data=0;

        if(cs!=READ_DATA) begin
            if(MOSI_count>=13) begin
                MOSI_count=0;
            end
            else begin
                MOSI_count++;
            end
        end
        else begin
            if(MOSI_count>=23) begin
                MOSI_count=0;
            end
            else begin
                MOSI_count++;
            end
        end
        i++;
        if(i==11 || MOSI_count==1) i=0; //to start sampling the valid_comb at the CHM_CMD state always
    end
endfunction

function string convert2string_stimulus();
    return $sformatf("rst=
,SS_endfunction
,MOSIfunction string convert2string();
,tx_validreturn $sformatf("rst=
,SS_data=%0b,rst_n,SS_n,MOSI,tx_valid,tx_data);
,MOSIdata,rx_data,rx_data_exp);
,tx_valid=exp
";rst_n,SS_n,MOSI,tx_valid,tx_data,MISO,MISO_exp,rx_valid,rx_valid_exp
,MISOcxge
,rx_valid=
```

# 2.8.SPI Sequencer

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Sequencer
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package spi_sequencer_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import spi_seq_item::*;
    import shared_pkg::*;
    class spi_sqr extends uvm_sequencer #(spi_seq_item);
        `uvm_component_utils(spi_sqr)

        function new(string name="spi_sqr",uvm_component parent=null);
            super.new(name,parent);
        endfunction
    endclass
endpackage
```

# 2.9.SPI Configuration Object

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Configuration Object
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package spi_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class spi_config_obj extends uvm_object;
        `uvm_object_utils(spi_config_obj)

        virtual spi_if spi_config_vif;
        uvm_active_passive_enum is_active;
        function new(string name="alsu_config_obj");
            super.new(name);
        endfunction
    endclass
endpackage
```

# 2.10.SPI Reset Sequence

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Object : Reset Sequence
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package spi_reset_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import spi_seq_item::*;

    class spi_reset_seq extends uvm_sequence #(spi_seq_item);
    `uvm_object_utils(spi_reset_seq)

    spi_seq_item seq_item;

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

    task body;
        seq_item=spi_seq_item::type_id::create("seq_item");
        start_item(seq_item);
        seq_item.rst_n=0;
        seq_item.MOSI=$random;
        finish_item(seq_item);
    endtask
    endclass
endpackage
```

# 2.11.SPI Main Sequence

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Object : Reset Sequence
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package spi_main_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import spi_seq_item::*;

    class spi_main_seq extends uvm_sequence #(spi_seq_item);
    `uvm_object_utils(spi_main_seq)

    spi_seq_item seq_item;

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

    virtual task body();
        seq_item=spi_seq_item::type_id::create("seq_item");
        repeat(2000) begin
            start_item(seq_item);
                assert(seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
    endclass
endpackage
```

# 2.12.SPI Driver

```
////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Driver
////////////////////////////////////////////////////////////////////////////////////////////////////////

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

    import spi_config_pkg::*;
    import spi_seq_item::*;

    class spi_driver extends uvm_driver #(spi_seq_item);
        `uvm_component_utils(spi_driver)

        //Virtual interface & sequence item object
        virtual spi_if spi_driver_vif;
        spi_seq_item drv_seq_item;

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

        //Build_Phase
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
        endfunction

        //Run Phase :: Create seq_item => seq_item_port.get_next_item => drive the interface from the sequenc item => wait negative edge => seq_item_port.item_done
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                //Create seq_item
                drv_seq_item=spi_seq_item::type_id::create("drv_seq_item");
                //seq_item_port.get_next_item : get the values from the driver port connected with the sequencer to the driver seq item
                seq_item_port.get_next_item(drv_seq_item);

                //Drive the interface with the driver sequence item
                spi_driver_vif.rst_n= drv_seq_item.rst_n;
                spi_driver_vif.MOSI = drv_seq_item.MOSI;
                spi_driver_vif.tx_data= drv_seq_item.tx_data;
                spi_driver_vif.SS_n= drv_seq_item.SS_n;
                spi_driver_vif.tx_valid= drv_seq_item.tx_valid;

                @(negedge spi_driver_vif.clk);
                //Transaction Done , now get next item and so on ...
                seq_item_port.item_done();
                `uvm_info("Driver run phase",drv_seq_item.convert2string_stimulus(),UVM_HIGH)
            end

        endtask

    endclass
endpackage
```

# 2.13.SPI Monitor

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Monitor
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package spi_monitor_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import spi_seq_item::*;
    import shared_pkg::*;
    class spi_monitor extends uvm_monitor ;
        `uvm_component_utils(spi_monitor)

        //Creating Virtual interface to get the data from the interface
        virtual spi_if s_if;
        //Creating monitor sequence item to pass the interface to it to be then sent to the analysis components
        spi_seq_item mon_seq_item;
        //Creating monitor analysis port to broadcast the values to the analysis components
        uvm_analysis_port #(spi_seq_item) mon_ap;

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

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

        //Run Phase: samples the data at the negative edge and writes the analysis port with the monitor sequence item
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                mon_seq_item=spi_seq_item::type_id::create("mon_seq_item");
                @(negedge s_if.clk);

                //Drive the monitor sequence item with the interface
                mon_seq_item.rst_n=s_if.rst_n;
                mon_seq_item.SS_n=s_if.SS_n;
                mon_seq_item.MOSI=s_if.MOSI;
                mon_seq_item.MISO=s_if.MISO;
                mon_seq_item.MISO_exp=s_if.MISO_exp;
                mon_seq_item.tx_valid=s_if.tx_valid;
                mon_seq_item.tx_data=s_if.tx_data;
                mon_seq_item.rx_valid=s_if.rx_valid;
                mon_seq_item.rx_valid_exp=s_if.rx_valid_exp;
                mon_seq_item.rx_data=s_if.rx_data;
                mon_seq_item.rx_data_exp=s_if.rx_data_exp;

                //Broadcast the monitor sequence (interfance) to the analysis port
                mon_ap.write(mon_seq_item);
                `uvm_info("Monitor run phase",mon_seq_item.convert2string(),UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

# 2.14.SPI Agent

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//Name: Omar Mohamed Hussein
//UVM_Component: Agent
////////////////////////////////////////////////////////////////////////////////////////////////////
package spi_agent_pkg;

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

    import spi_seq_item::*;
    import spi_sequencer_pkg::*;
    import spi_driver_pkg::*;
    import spi_monitor_pkg::*;
    import spi_config_pkg::*;

    class spi_agent extends uvm_agent;
        `uvm_component_utils(spi_agent)

        //creating the sequencer,driver,monitor, configuration object to connect the interface,agent port
        spi_sqr sqr;
        spi_driver drv;
        spi_monitor mon;
        spi_config_obj config_obj;
        uvm_analysis_port #(spi_seq_item) agt_ap;

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

        //Build Phase
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            //Connect the agent configuration object with the configuration object in the uvm config data base
            if(!uvm_config_db #(spi_config_obj)::get(this,"","CFG",config_obj)) begin
                `uvm_fatal("agent build phase","unable to get the configuration object to the agent")
            end

            ////////////////// For Passive Agents ////////////////////////////////////////////////////
            // if(shift_reg_cfg.is_active==UVM_ACTIVE) begin
            //     sqr=sequencer::type_id::create("sqr",this);
            //     drv=shift_reg_driver::type_id::create("drv",this);
            // end
            ////////////////////////////////////////////////////////////////////////////////////////////

            //Creating the sequencer,driver,monitor with the create() and the agt port with new()
            sqr=spi_sqr    ::type_id::create("sqr",this);
            drv=spi_driver ::type_id::create("drv",this);
            mon=spi_monitor::type_id::create("mon",this);
            agt_ap=new("agt_ap",this);
        endfunction

        //connect phase
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);

            ////////////////////////////////////For Passive Agents ////////////////////////////////////
            // if(spi_cfg.is_active==UVM_ACTIVE) begin
            //     drv.spi_vif=spi_cfg.spi_vif;
            //     drv.seq_item_port.connect(sqr.seq_item_export);
            // end
            ////////////////////////////////////////////////////////////////////////////////////////////

            //Conncet the configuration object with the driver and the monitor ports
            drv.spi_driver_vif=config_obj.spi_config_vif;
            mon.s_if=config_obj.spi_config_vif;
            //Connect the driver port with the sequencer export
            drv.seq_item_port.connect(sqr.seq_item_export);
            //Connect the monitor analysis port with the agt port
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage
```

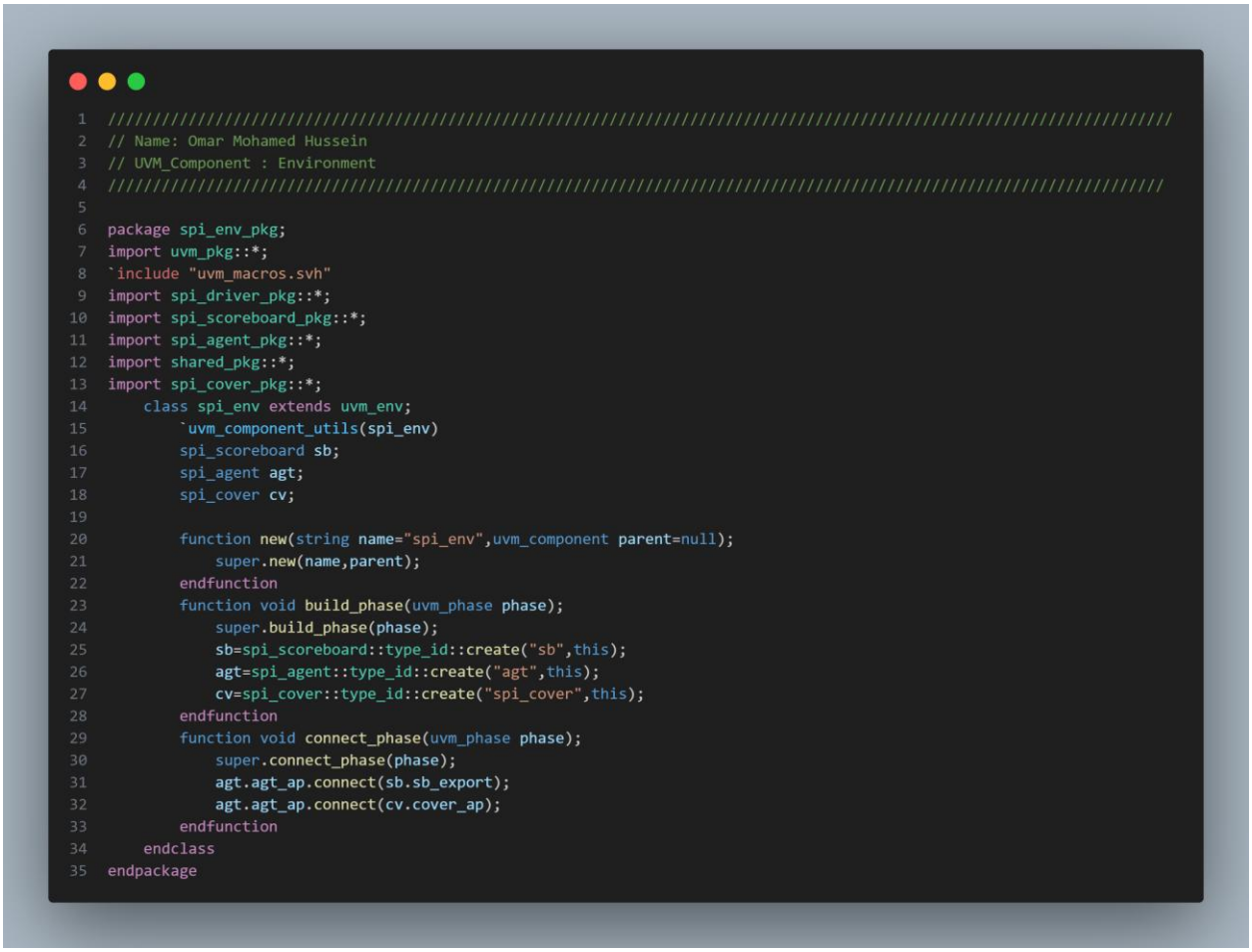
# 2.15.SPI Scoreboard

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Scoreboard
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

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

    import spi_seq_item::*;

    class spi_scoreboard extends uvm_scoreboard ;
        `uvm_component_utils(spi_scoreboard)

        //Creating sequence item to put the data in the fifo
        spi_seq_item seq_item;
        //Creating analysis export to take values from the agt port
        uvm_analysis_export #(spi_seq_item) sb_export;
        //Creating analysis fifo to take values from sb export
        uvm_tlm_analysis_fifo #(spi_seq_item) sb_fifo;

        int correct_count,error_count;

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

        //Build phase : create the export analysis
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb_export=new("sb_export",this);
            sb_fifo=new("sb_fifo",this);
        endfunction

        //Connect phase : connect the eb export with the fifo export
        function void connect_phase (uvm_phase phase);
            super.connect_phase(phase);
            sb_export.connect(sb_fifo.analysis_export);
        endfunction

        //run Task : Compare the output with the Golden model
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(seq_item);
                if(seq_item.rx_valid!=seq_item.rx_valid_exp || seq_item.rx_data!=seq_item.rx_data_exp || seq_item.MISO!=seq_item.MISO_exp) begin
                    `uvm_error("scoreboard run phase",$sformatf("Design doesn't match the Golden model => %s",seq_item.convert2string))
                    error_count++;
                end
                else begin
                    correct_count++;
                end
            end
        endtask

        //Report Phase: report the testbench with how many failed and correct trials
        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("Scoreboard Report Phase",$sformatf("Testbench Completed , Correct count=
, error count=
",correct_count,error_count),UVM_LOW)
        endfunction
    endpackage
```

# 2.16.SPI Cover

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Cover
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

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

    class spi_cover extends uvm_component;
        `uvm_component_utils(spi_cover)
        uvm_analysis_export #(spi_seq_item) cover_ap;
        uvm_tlm_analysis_fifo #(spi_seq_item) cover_fifo;
        spi_seq_item seq_item;


    //Cover Groups
        covergroup Cov_gp(ref spi_seq_item seq_item);
        //Normal Coverpoints
        rx_cp: coverpoint seq_item.rx_data[9:8]{
            bins all_values[] = {[0:3]};
            bins WRITE_ADD =(2'b00=>2'b00=>2'b00) ;
            bins WRITE_DATA=(2'b00=>2'b00=>2'b01) ;
            bins READ_ADD  =(2'b00=>2'b10=>2'b10) ;
            bins READ_DATA =(2'b00=>2'b10=>2'b11) ;
        }
        SS_n_cp: coverpoint seq_item.SS_n{
            bins read_transition=(1=> 0[*23] =>1);
            bins not_read_transition=(1=> 0[*13] =>1);
        }
        MOSI_cp:coverpoint seq_item.MOSI{
            bins b000=(0=>0=>0);
            bins b001=(0=>0=>1);
            bins b111=(1=>1=>1);
            bins b110=(1=>1=>0);
        }
        cross_cv:cross MOSI_cp,SS_n_cp{
            ignore_bins write_cv_000 =binsof(MOSI_cp.b000) && binsof(SS_n_cp.read_transition);
            ignore_bins write_cv_001 =binsof(MOSI_cp.b001) && binsof(SS_n_cp.read_transition);
            ignore_bins read_cv_111 =binsof(MOSI_cp.b111) && binsof(SS_n_cp.not_read_transition);
            ignore_bins read_cv_110 =binsof(MOSI_cp.b110) && binsof(SS_n_cp.not_read_transition);
        }
        //Cross coverage
        endgroup

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

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

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cover_ap.connect(cover_fifo.analysis_export);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                seq_item=spi_seq_item::type_id::create("seq_item");
                cover_fifo.get(seq_item);
                Cov_gp.sample();
            end
        endtask

    endclass
endpackage
```

# 2.17.SPI Environment



```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Environment
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package spi_env_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import spi_driver_pkg::*;
import spi_scoreboard_pkg::*;
import spi_agent_pkg::*;
import shared_pkg::*;
import spi_cover_pkg::*;
    class spi_env extends uvm_env;
        `uvm_component_utils(spi_env)
        spi_scoreboard sb;
        spi_agent agt;
        spi_cover cv;

        function new(string name="spi_env",uvm_component parent=null);
            super.new(name,parent);
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb=spi_scoreboard::type_id::create("sb",this);
            agt=spi_agent::type_id::create("agt",this);
            cv=spi_cover::type_id::create("spi_cover",this);
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            agt.agt_ap.connect(sb.sb_export);
            agt.agt_ap.connect(cv.cover_ap);
        endfunction
    endclass
endpackage
```

# 2.18.SPI Test

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Test
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package spi_test_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import spi_env_pkg::*;
import spi_config_pkg::*;
import spi_reset_seq_pkg::*;
import spi_main_seq_pkg::*;
    class spi_test extends uvm_test;
        `uvm_component_utils(spi_test)

        spi_env env;
        spi_config_obj spi_config_obj_test;
        spi_reset_seq reset_seq;
        spi_main_seq main_seq;
        function new(string name ="spi_test",uvm_component parent=null);
            super.new(name,parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            env=spi_env::type_id::create("env",this);
            spi_config_obj_test=spi_config_obj::type_id::create("spi_config_obj_test");
            reset_seq=spi_reset_seq::type_id::create("reset_seq");
            main_seq=spi_main_seq::type_id::create("main_seq");

            if(!uvm_config_db #(virtual spi_if)::get(this,"","spi_if",spi_config_obj_test.spi_config_vif))
                `uvm_fatal("build phase in test","unable to get interface from the database into the configuration object")
            uvm_config_db#(spi_config_obj)::set(this,"*","CFG",spi_config_obj_test);
        endfunction

        task  run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            `uvm_info("Test run phase","Reset Asserted",UVM_LOW)
            reset_seq.start(env.agt.sqr);
            `uvm_info("Test run phase","Reset Finished",UVM_LOW)

            `uvm_info("Test run phase","Main sequence Asserted",UVM_LOW)
            main_seq.start(env.agt.sqr);
            `uvm_info("Test run phase","Main sequence Finished",UVM_LOW)
            phase.drop_objection(this);
        endtask
    endclass
endpackage
```

# 2.19.SPI Top

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : TOP
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

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

module top();
  // Clock generation
  bit clk;
  initial begin
    clk=0;
    forever begin
      #10 clk=~clk;
    end
  end
  // Instantiate the interface ,Golden Model and DUT
  spi_if s_if(clk);
  SLAVE slave(s_if.MOSI,s_if.MISO,s_if.SS_n,s_if.clk,s_if.rst_n,s_if.rx_data,s_if.rx_valid,s_if.tx_data,s_if.tx_valid);
  SPI_slave_op Golden_spi(s_if.MOSI,s_if.MISO_exp,s_if.clk,s_if.rst_n,s_if.rx_data_exp,s_if.tx_data,s_if.rx_valid_exp,s_if.tx_valid,s_if.SS_n);
  bind slave spi_sva binded_assertions(s_if.MOSI,s_if.MISO,s_if.SS_n,s_if.clk,s_if.rst_n,s_if.rx_data,s_if.rx_valid,s_if.tx_data,s_if.tx_valid);

  // run test using run_test task
  initial begin
    uvm_config_db #(virtual spi_if)::set(null,"uvm_test_top","spi_if",s_if);
    run_test("spi_test");
  end
endmodule
```

# 2.20.SPI Source File

# 2.21.SPI DO File

```
vlib work
vlog +define+SIM -f src_files.list  +cover -covercells
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
add wave /top/s_if/*
add wave -position insertpoint  \
sim:/top/Golden_spi/cs
add wave -position insertpoint  \
sim:/top/slave/cs
add wave -position insertpoint  \
sim:/top/Golden_spi/count \
sim:/top/Golden_spi/count_10
add wave -position insertpoint  \
sim:/top/slave/counter
run 0
add wave -position insertpoint  \
sim:/uvm_root/uvm_test_top/env/agt/drv/drv_seq_item
coverage save priority_en_tb.ucdb -onexit
run -all
```

# 2.22.SPI Assertions

```
module spi_sva(MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
input MOSI,MISO,SS_n,clk,rst_n,rx_valid,tx_valid;
input [9:0]rx_data;
input [7:0]tx_data;

    sequence valid_comb_000;
    ($fell(SS_n) ##1 !MOSI ##1 !MOSI ##1 !MOSI);
    endsequence
    sequence valid_comb_001;
    ($fell(SS_n) ##1  !MOSI ##1 !MOSI ##1 MOSI);
    endsequence
    sequence valid_comb_110;
    ($fell(SS_n) ##1 MOSI ##1 MOSI ##1 !MOSI);
    endsequence
    sequence valid_comb_111;
    ($fell(SS_n) ##1 MOSI ##1 MOSI ##1 MOSI);
    endsequence
    assert property(@(posedge clk) disable iff(!rst_n) valid_comb_000 |-> ##10 (rx_valid ##1 SS_n[->1]));
    assert property(@(posedge clk) disable iff(!rst_n) valid_comb_001 |-> ##10 (rx_valid ##1 SS_n[->1]));
    assert property(@(posedge clk) disable iff(!rst_n) valid_comb_110 |-> ##10 (rx_valid ##1 SS_n[->1]));
    assert property(@(posedge clk) disable iff(!rst_n) valid_comb_111 |-> ##10 (rx_valid ##1 SS_n[->1]));
    assert property(@(posedge clk) !rst_n |=> !MISO && !rx_valid && rx_data==0);
    cover  property(@(posedge clk) disable iff(!rst_n) valid_comb_000 |-> ##10 (rx_valid  ##1 SS_n[->1]));
    cover  property(@(posedge clk) disable iff(!rst_n) valid_comb_001 |-> ##10 (rx_valid  ##1 SS_n[->1]));
    cover  property(@(posedge clk) disable iff(!rst_n) valid_comb_110 |-> ##10 (rx_valid  ##1 SS_n[->1]));
    cover  property(@(posedge clk) disable iff(!rst_n) valid_comb_111 |-> ##10 (rx_valid  ##1 SS_n[->1]));
    cover  property(@(posedge clk) !rst_n |=> !MISO && !rx_valid && rx_data==0);
endmodule
```

## Assertion Table:

| Feature | Assertions |
|---|---|
| If the reset signal is deactivated and there is sequence of MOSI of 3'b000 , after 10 clock cycles the rx_valid should be up and then after one cycle SS_n should high. | (@(posedge clk) disable iff(!rst_n) valid_comb_000 \|-> ##10 (rx_valid ##1 SS_n[->1])) |
| If the reset signal is deactivated and there is sequence of MOSI of 3'b001 , after 10 clock cycles the rx_valid should be up and then after one cycle SS_n should high. | (@(posedge clk) disable iff(!rst_n) valid_comb_001 \|-> ##10 (rx_valid ##1 SS_n[->1])) |
| If the reset signal is deactivated and there is sequence of MOSI of 3'b110 , after 10 clock cycles the rx_valid should be up and then after one cycle SS_n should high. | (@(posedge clk) disable iff(!rst_n) valid_comb_110 \|-> ##10 (rx_valid ##1 SS_n[->1])) |
| If the reset signal is deactivated and there is sequence of MOSI of 3'b111 , after 10 clock cycles the rx_valid should be up and then after one cycle SS_n should high. | (@(posedge clk) disable iff(!rst_n) valid_comb_111 \|-> ##10 (rx_valid ##1 SS_n[->1])) |
| If the reset signal is activated , the MISO , rx_valid and the rx_data should be low. | (@(posedge clk) !rst_n \|=> !MISO && !rx_valid && rx_data==0) |

# 2.23.Simulation Results

## 2.23.1.Questasim Transcript

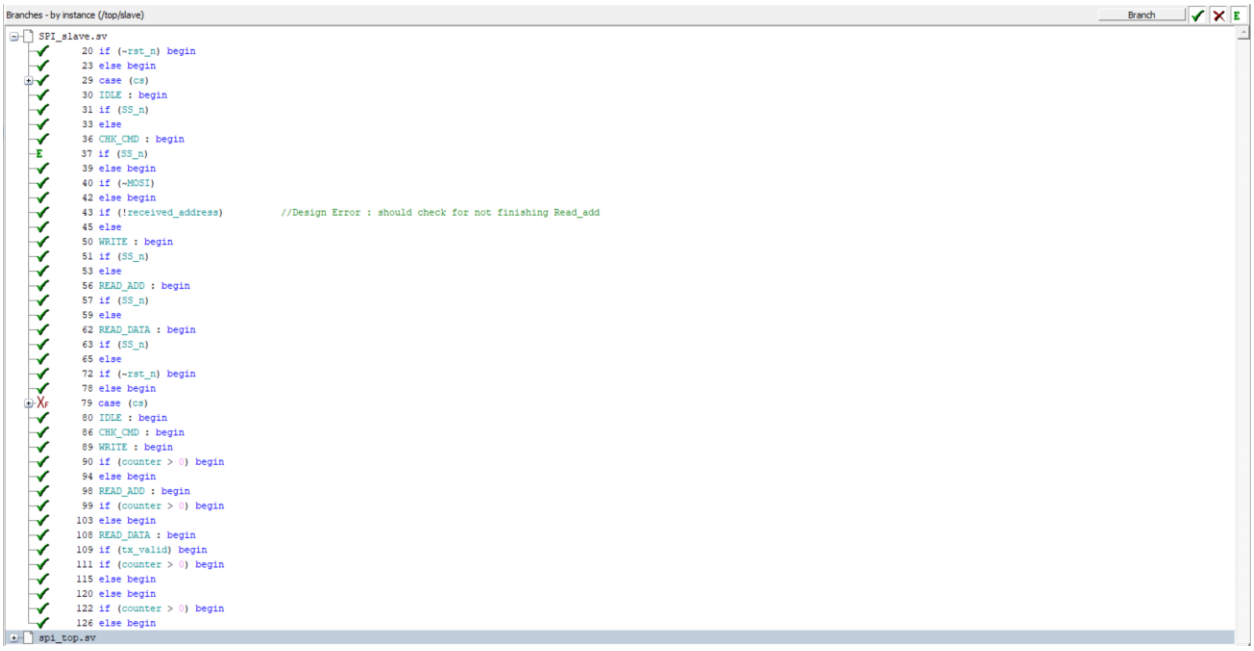
```
# ********************************************************************************
# * Questa UVM Transaction Recording Turned ON.                                  *
# * recording_detail has been set.                                               *
# *  To turn off, set 'recording_detail' to off:                                 *
# * uvm_config_db#(int)            ::set(null, "", "recording_detail", 0);       *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0);       *
# ********************************************************************************
# UVM_INFO spi_test.sv(41) @ 20: uvm_test_top [Test run phase] Reset Finished
# UVM_INFO spi_test.sv(43) @ 20: uvm_test_top [Test run phase] Main sequence Asserted
# UVM_INFO spi_test.sv(45) @ 40020: uvm_test_top [Test run phase] Main sequence Finished
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 40020: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO spi_scoreboard.sv(60) @ 40020: uvm_test_top.env.sb [Scoreboard Report Phase] Testbench Completed , Correct count=2001, error count=0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    9
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [Scoreboard Report Phase]     1
# [TEST_DONE]     1
# [Test run phase]     4
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 40020 ns  Iteration: 61  Instance: /top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430

VSIM 2>
```

## 2.23.2.Questasim Code Coverage

**Statement Coverage:**


- Note: we excluded the ns statement when the cs=CHK_CMD as our sequence will never hit this line.

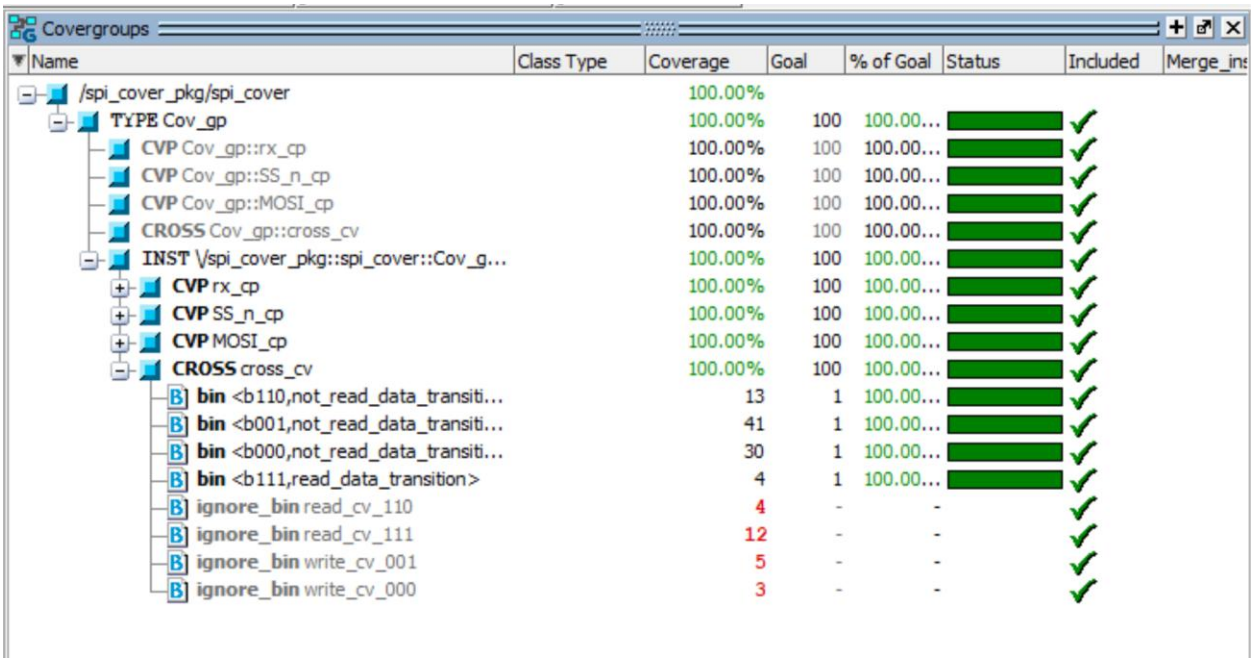
## Branch Coverage:



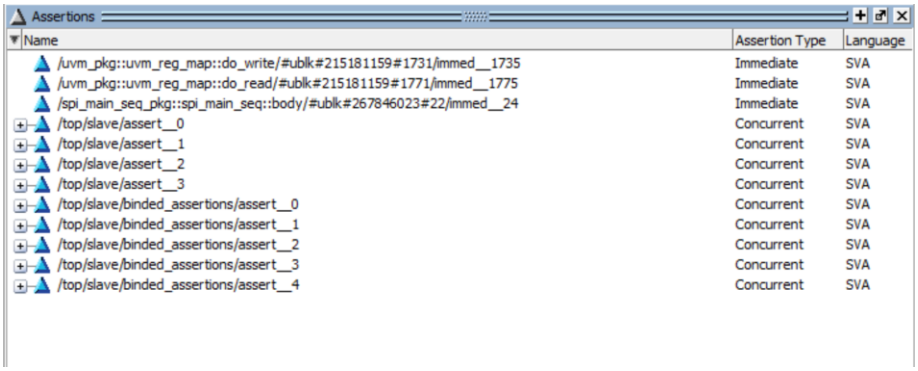

- Note: We excluded the check for the SS_n when the cs=CHK_CMD as our sequence will never hit it.

## Toggle Coverage:

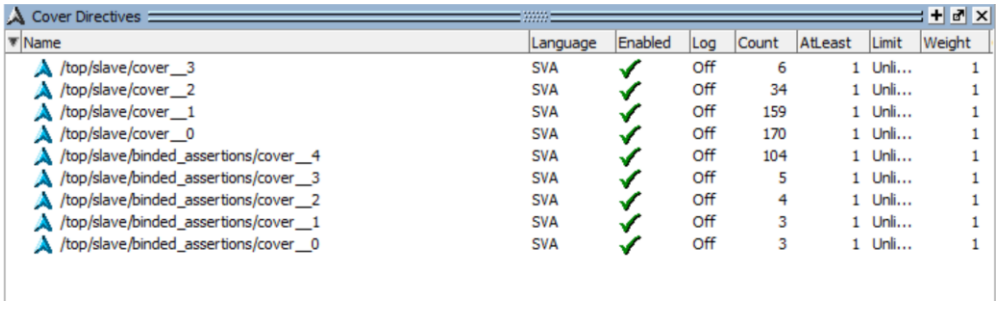
## 2.23.3.Questasim Functional Coverage




## 2.23.4.Questasim Assertions

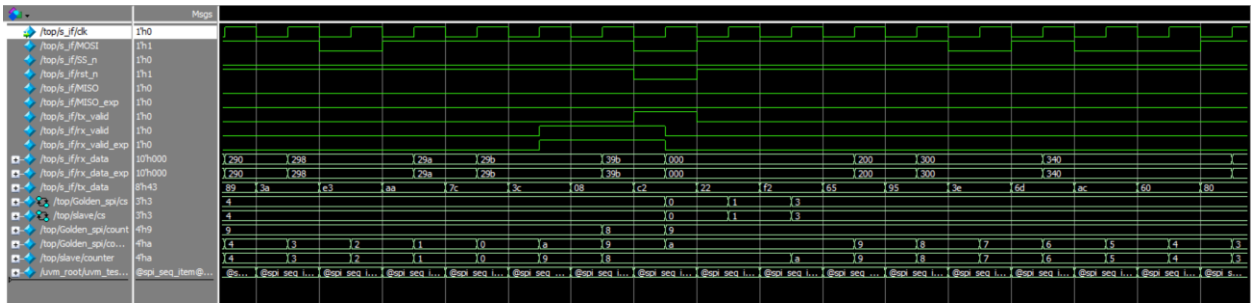



## 2.23.5.Questasim Assertions Coverage

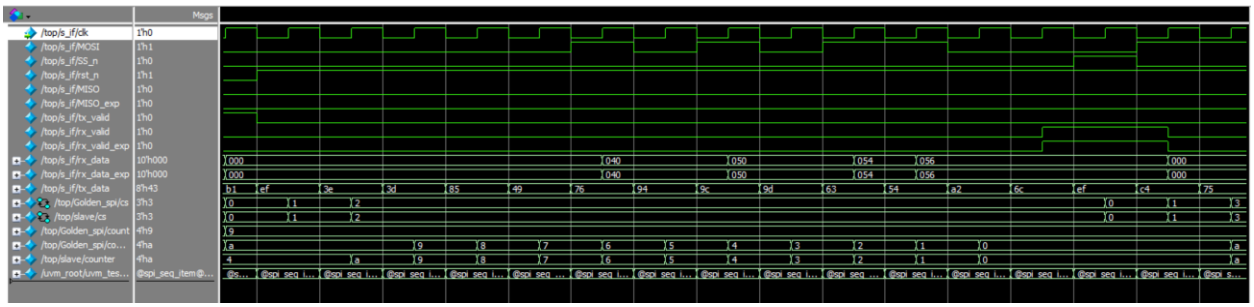


## 2.23.6.Questasim Waveform

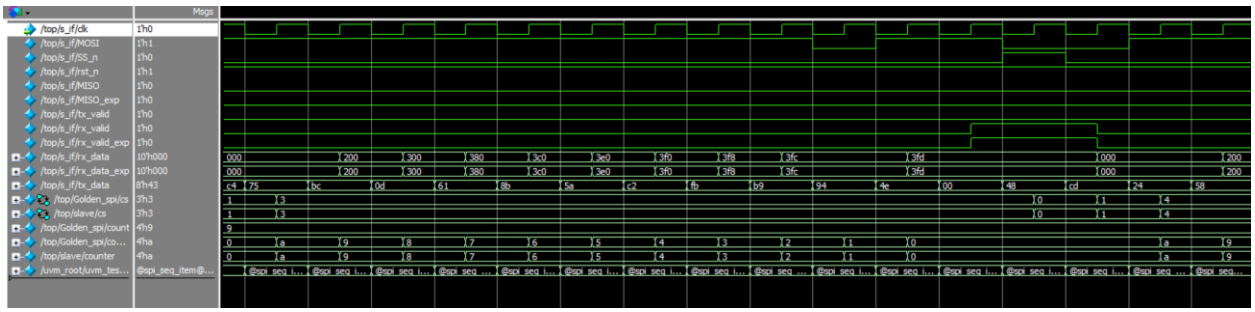
- Reset Sequence:



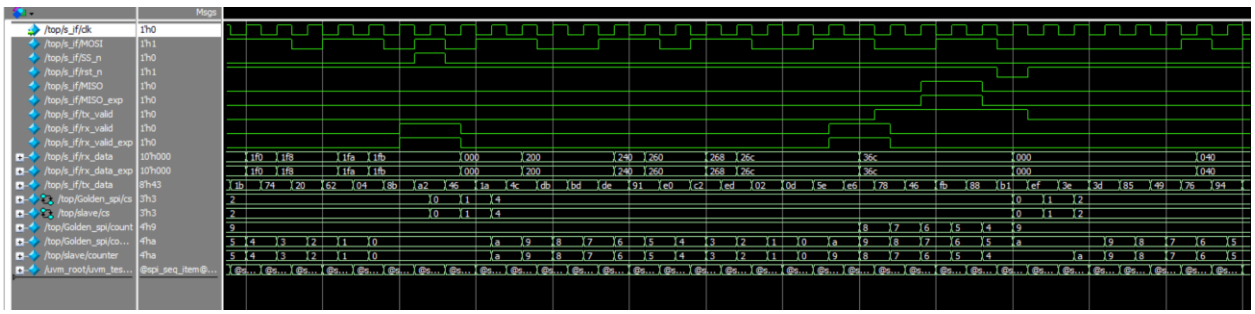
- Write Sequence:



- Read Address Sequence:



- Read Data Sequence:



# *3.Single Port Synchronous RAM UVM Verification*

## 3.1.Verifications Requirements

In this part, you will create a full UVM environment to verify the **RAM design** with default parameters (MEM_DEPTH: 256, ADDR_SIZE: 8)

### Interface:

| Name | Type | Size | Description |
|---|---|---|---|
| din | Input | 10 bits | Data Input |
| clk | | 1 bit | Clock |
| rst | | 1 bit | Active low synchronous reset |
| rx_valid | | 1 bit | If HIGH: accept din[7:0] to save the write/read address internally or write a memory word depending on the most significant 2 bits din[9:8] |
| dout | Output | 8 bits | Data Output |
| tx_valid | | 1 bit | Whenever the command is memory read the tx_valid should be HIGH |

### Sequences Requirements:

Split the RAM sequences into the following sequences. You can switch on and off the constraints or use inline constraints to control the sequence item generated in the sequence

- reset_sequence
- write_only_sequence
- read_only_sequence
- write_read_sequence

## Constraint Requirements:

1- The reset signal (rst_n) shall be deasserted most of the time.

2- The rx_valid signal shall be asserted most of time.

3- For a *write-only sequence*, every Write Address operation shall always be followed by either Write Address or Write Data operation.

4- For a *read-only sequence*, every Read Address operation shall always be followed by either Read Address or Read Data operation.

5- For a ***randomized read/write sequence***, the following ordering rules shall be enforced:

- Every Write Address operation shall always be followed by either Write Address or Write Data operation.
- After a Write Data, the next operation shall be chosen with the following probability distribution: 60% → Read Address & 40% → Write Address
- Every Read Address operation shall always be followed by either Read Address or Read Data operation.After a Read Data, the next operation shall be chosen with the following probability distribution: 60% → Write Address & 40% → Read Address

***(Hint: You can use post_randomize and inline constraints)***

## Functional Coverage Requirements:

1- Write Coverpoint to check transaction ordering for din[9:8]:

- Check din[9:8] takes 4 possible values
- Check write data after write address
- Check read data after read address
- Check write address => write data => read address => read data

3- Cross coverage:

- Between all bins of din[9:8] and rx_valid signal when it is high
- Between din[9:8] when it equals read data and tx_valid when it is high

## Assertions Requirements: (*Add the assertions file and bind it in the top module)*

1- An assertion ensures that whenever reset is asserted, the output signals (tx_valid and dout) are low

2- An assertion checks that during address or data input phases (write_add_seq, write_data_seq, read_add_seq), the tx_valid signal must remain deasserted.

3- An assertion checks that after a read_data_seq occurs, the tx_valid signal must rise to indicate valid output and after it rises by one clock cycle, it should eventually fall.

# 3.2.RAM Verification Plan

| Label | Design Requirment Description | Stimulus generation | Functional coverage | Functionality check |
|---|---|---|---|---|
| reset check | when the reset signal is activated , the tx_valid,dataout are set low | directed reset assertion using reset sequence | | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the Ram_sva file. |
| IDLE | when the reset signal is deactivated and the rx_valid is high, if the din[9:8] are 2'b00, the remaining bus should be registered as write address. | Randomizing the din and forcing din[9:8] to take write address or data only in the write sequence only | Covering the Transition from write address to write data crossing it with the rx_valid possible values | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the Ram_sva file. |
| CHK_CMD | when the reset signal is deactivated and the rx_valid is high, if the din[9:8] are 2'b01, the remaining bus should be registered as write data in the registerd address before. | Randomizing the din and forcing din[9:8] to take write address or data only in the write sequence only | Covering the Transition from write address to write data crossing it with the rx_valid possible values | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the Ram_sva file. |
| WRITE_ADDR | when the reset signal is deactivated and the rx_valid is high, if the din[9:8] are 2'b10, the remaining bus should be registered as read address. | Randomizing the din and forcing din[9:8] to take read address or data only in the write sequence only | Covering the Transition from read address to readdata crossing it with the rx_valid possible values | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the Ram_sva file. |
| WRITE_DATA | when the reset signal is deactivated and the rx_valid is high, if the din[9:8] are 2'b11, the remaining bus should be registered as read data in the registerd address before. | Randomizing the din and forcing din[9:8] to take read address or data only in the write sequence only | Covering the Transition from read address to readdata crossing it with the rx_valid possible values | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the Ram_sva file. |

# 3.3.RAM Interface

```
1  interface Ram_if(clk);
2      input clk;
3      logic     [9:0] din;
4      logic           rst_n, rx_valid;
5      logic  [7:0] dout,dout_exp;
6      logic        tx_valid,tx_valid_exp;
7
8  endinterface
```

# 3.4.RAM Design

```
1  module RAM (din,clk,rst_n,rx_valid,dout,tx_valid);
2
3  input      [9:0] din;
4  input            clk, rst_n, rx_valid;
5
6  output reg [7:0] dout;
7  output reg       tx_valid;
8
9  reg [7:0] MEM [255:0];
10
11 reg [7:0] Rd_Addr, Wr_Addr;
12
13 always @(posedge clk) begin
14     if (~rst_n) begin
15         dout <= 0;
16         tx_valid <= 0;
17         Rd_Addr <= 0;
18         Wr_Addr <= 0;
19     end
20     else  begin             //Design Error: missing begin and end
21         if (rx_valid) begin
22             case (din[9:8])
23                 2'b00 : Wr_Addr <= din[7:0];
24                 2'b01 : MEM[Wr_Addr] <= din[7:0];
25                 2'b10 : Rd_Addr <= din[7:0];
26                 2'b11 : dout <= MEM[Rd_Addr];          //Design Error : should be of Rd_addr
27                 default : dout <= 0;
28             endcase
29         end
30         if(din[9] && din[8] && rx_valid) begin
31             tx_valid<=1;
32         end
33         else if ((!din[9] || !din[8]) && rx_valid) begin
34             tx_valid<=0;
35         end
36         //tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1: 1'b0;
37     end
38 end
39
40 endmodule
```

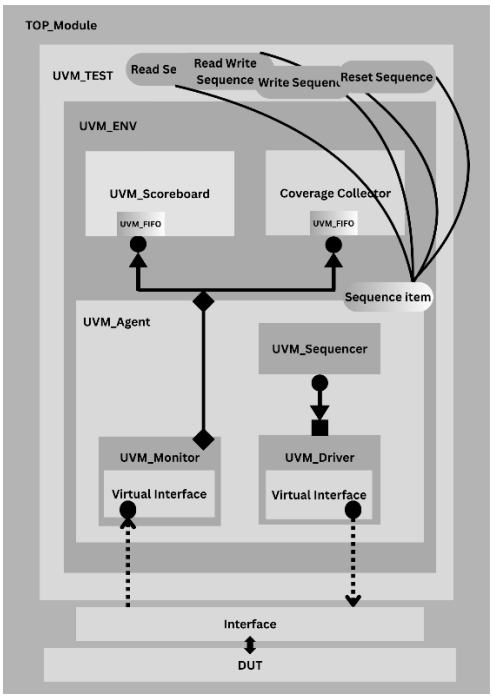
## 3.4.1.Design Bugs

- **In line 20:** Missing begin and end for the else block.
- **In line 26:** The dout should take the memory data with the address Rd_address.
- **In line 30:** should implement if else block to make sure tx_valid only updated when the rx_valid is high.

# 3.5.RAM Design Golden Model

```
module SPR#(parameter MEM_DEPTH=256,parameter ADDR_SIZE=8)(din,clk,rstn,rx_valid,tx_valid,dout);

input [9:0]din;
input clk,rstn,rx_valid;
output reg tx_valid;
output reg [7:0]dout;

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

reg [7:0]Wr_addr,Rd_addr; //internal address signal

always @(posedge clk) begin
    if(!rstn) begin
        dout<=8'b0;
        Wr_addr<=8'b0;
        Rd_addr<=8'b0;
        tx_valid<=0;
    end
    else begin
        if(rx_valid) begin
            case({din[9:8]})
                2'b00: begin
                    Wr_addr<=din[7:0];
                    tx_valid<=0;
                end
                2'b01: begin
                    MEM[Wr_addr]<=din[7:0];
                    tx_valid<=0;
                end
                2'b10: begin
                    Rd_addr<=din[7:0];
                    tx_valid<=0;
                end
                2'b11: begin
                    dout<=MEM[Rd_addr];
                    tx_valid<=1;
                end
            endcase
        end
    end
end
endmodule

```

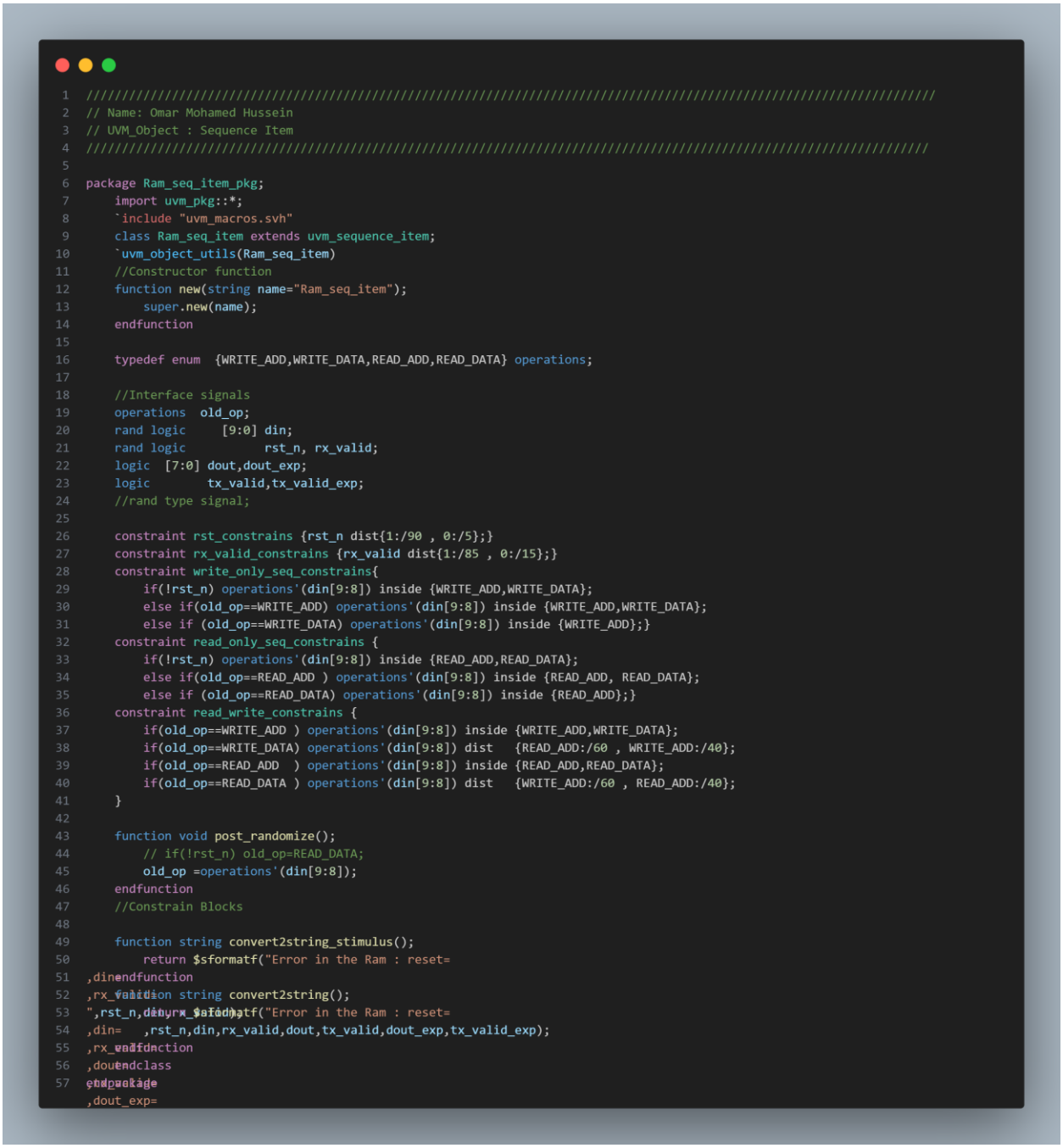
# 3.6.UVM Diagram




**Verification Flow:**

- TOP Module instantiates the interface,Ram Design, Golden Model and bind assertions, then it sets the interface in the configuration object then runs the test.
- Test creates the environment and starts sequences: Reset Sequence, Write Sequence, Read sequence and Read Write Sequence.
- Environment Creates the Agent, Scoreboard and the Cover Collector.
- Also Connecting the Agent analysis port with the Scoreboard and Cover Collector analysis export ports.
- Agent creates sequencer, driver and the monitor , connecting the sequencer port with the driver and connecting the analysis port of the monitor with the analysis port of the Agent.
- The Driver uses a virtual interface to drive DUT pins.
- The Monitor samples the same virtual interface to observe pin behavior and rebuild transaction objects.
- The RAM Scoreboard and Coverage Collector receive transaction objects produced by monitors through TLM analysis channels and perform checking and coverage collection.

# 3.7.RAM Sequence item



```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Object : Sequence Item
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_seq_item_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class Ram_seq_item extends uvm_sequence_item;
    `uvm_object_utils(Ram_seq_item)
    //Constructor function
    function new(string name="Ram_seq_item");
        super.new(name);
    endfunction

    typedef enum  {WRITE_ADD,WRITE_DATA,READ_ADD,READ_DATA} operations;

    //Interface signals
    operations  old_op;
    rand logic     [9:0] din;
    rand logic           rst_n, rx_valid;
    logic  [7:0] dout,dout_exp;
    logic        tx_valid,tx_valid_exp;
    //rand type signal;

    constraint rst_constrains {rst_n dist{1:/90 , 0:/5};}
    constraint rx_valid_constrains {rx_valid dist{1:/85 , 0:/15};}
    constraint write_only_seq_constrains{
        if(!rst_n) operations'(din[9:8]) inside {WRITE_ADD,WRITE_DATA};
        else if(old_op==WRITE_ADD) operations'(din[9:8]) inside {WRITE_ADD,WRITE_DATA};
        else if (old_op==WRITE_DATA) operations'(din[9:8]) inside {WRITE_ADD};}
    constraint read_only_seq_constrains {
        if(!rst_n) operations'(din[9:8]) inside {READ_ADD,READ_DATA};
        else if(old_op==READ_ADD ) operations'(din[9:8]) inside {READ_ADD, READ_DATA};
        else if (old_op==READ_DATA) operations'(din[9:8]) inside {READ_ADD};}
    constraint read_write_constrains {
        if(old_op==WRITE_ADD ) operations'(din[9:8]) inside {WRITE_ADD,WRITE_DATA};
        if(old_op==WRITE_DATA) operations'(din[9:8]) dist   {READ_ADD:/60 , WRITE_ADD:/40};
        if(old_op==READ_ADD  ) operations'(din[9:8]) inside {READ_ADD,READ_DATA};
        if(old_op==READ_DATA ) operations'(din[9:8]) dist   {WRITE_ADD:/60 , READ_ADD:/40};
    }

    function void post_randomize();
        // if(!rst_n) old_op=READ_DATA;
        old_op =operations'(din[9:8]);
    endfunction
    //Constrain Blocks

    function string convert2string_stimulus();
        return $sformatf("Error in the Ram : reset=
,dinendfunction
,rx_validi
",rst_n,dinrx_valid)
,din=    ,rst_n,din,rx_valid,dout,tx_valid,dout_exp,tx_valid_exp);
,rx_valfunction
,doutendclass
endpackage
,dout_exp=
```

# 3.8.RAM Sequencer

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Sequencer
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_sqr_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import Ram_seq_item_pkg::*;

    class Ram_sqr extends uvm_sequencer #(Ram_seq_item);
        `uvm_component_utils(Ram_sqr)

        function new(string name="Ram_sqr",uvm_component parent=null);
            super.new(name,parent);
        endfunction
    endclass
endpackage
```

# 3.9.RAM Configuration Object

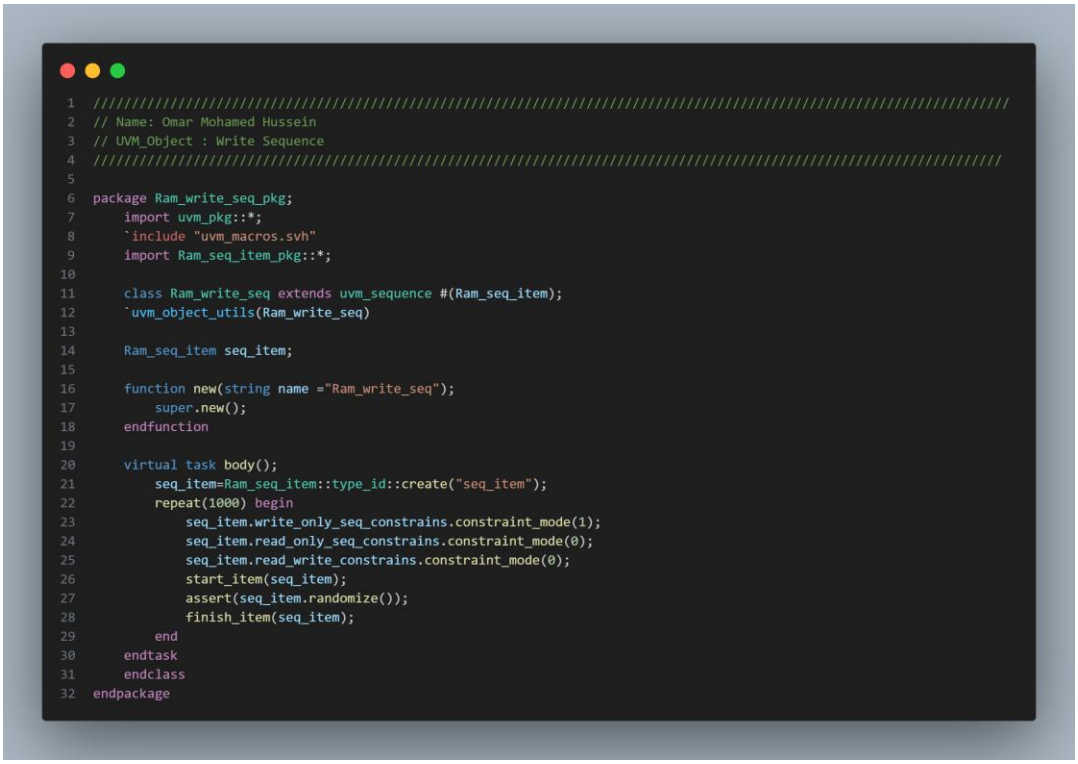
```
1  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2  // Name: Omar Mohamed Hussein
3  // UVM_Object : Configuration Object
4  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
5
6  package Ram_config_pkg;
7      import uvm_pkg::*;
8      `include "uvm_macros.svh"
9      class Ram_config_obj extends uvm_object;
10         `uvm_object_utils(Ram_config_obj)
11
12         virtual Ram_if Ram_config_vif;
13         uvm_active_passive_enum is_active;
14         function new(string name="Ram_config_obj");
15             super.new(name);
16         endfunction
17
18     endclass
19 endpackage
```

# 3.10.RAM Sequences

## 3.10.1.RAM Reset Sequence

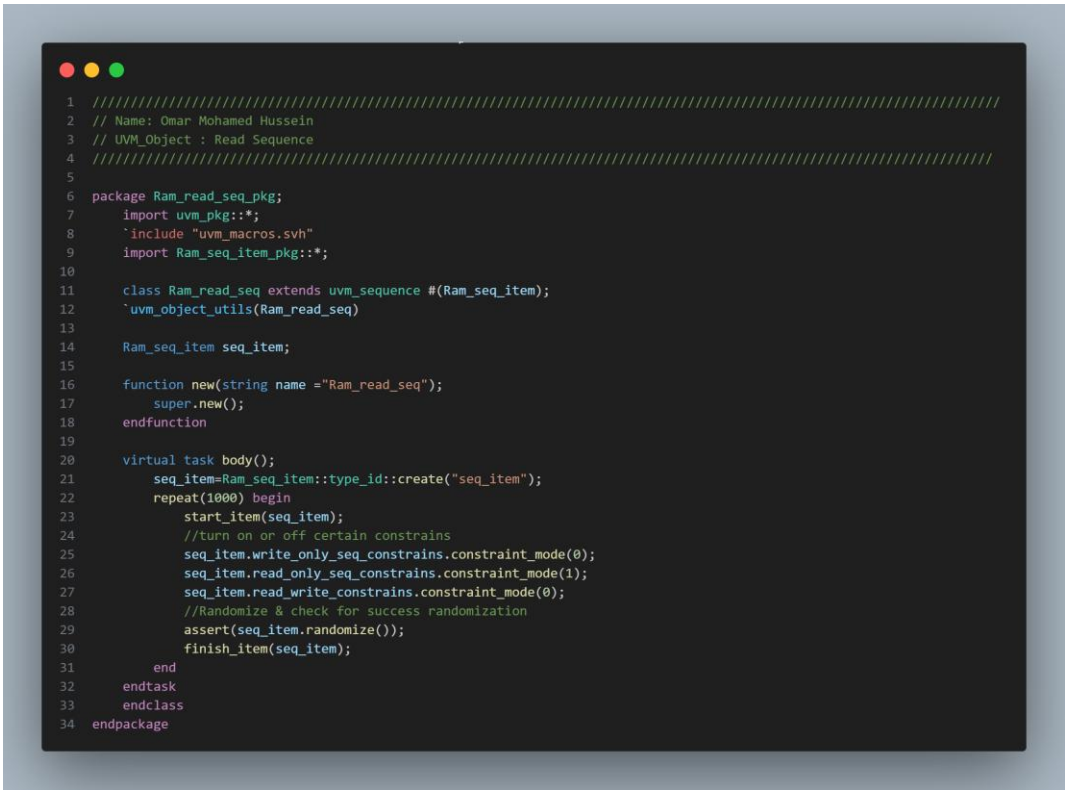
```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Object : Reset Sequence
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_reset_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import Ram_seq_item_pkg::*;

    class Ram_reset_seq extends uvm_sequence #(Ram_seq_item);
    `uvm_object_utils(Ram_reset_seq)

    Ram_seq_item seq_item;

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

    task body;
        seq_item=Ram_seq_item::type_id::create("seq_item");
        start_item(seq_item);
        seq_item.rst_n=0;
        finish_item(seq_item);
    endtask

    endclass
endpackage
```

## 3.10.2.RAM Write Sequence



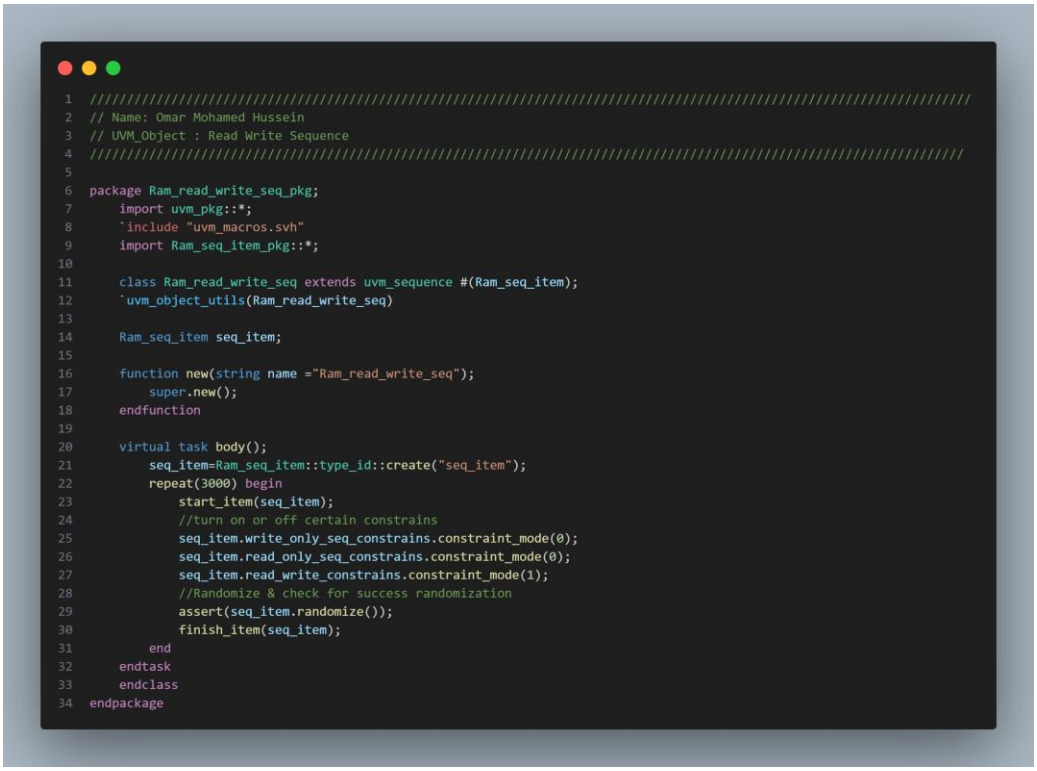
```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Object : Write Sequence
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_write_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import Ram_seq_item_pkg::*;

    class Ram_write_seq extends uvm_sequence #(Ram_seq_item);
    `uvm_object_utils(Ram_write_seq)

    Ram_seq_item seq_item;

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

    virtual task body();
        seq_item=Ram_seq_item::type_id::create("seq_item");
        repeat(1000) begin
            seq_item.write_only_seq_constrains.constraint_mode(1);
            seq_item.read_only_seq_constrains.constraint_mode(0);
            seq_item.read_write_constrains.constraint_mode(0);
            start_item(seq_item);
            assert(seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
    endclass
endpackage
```

## 3.10.3.RAM Read Sequence



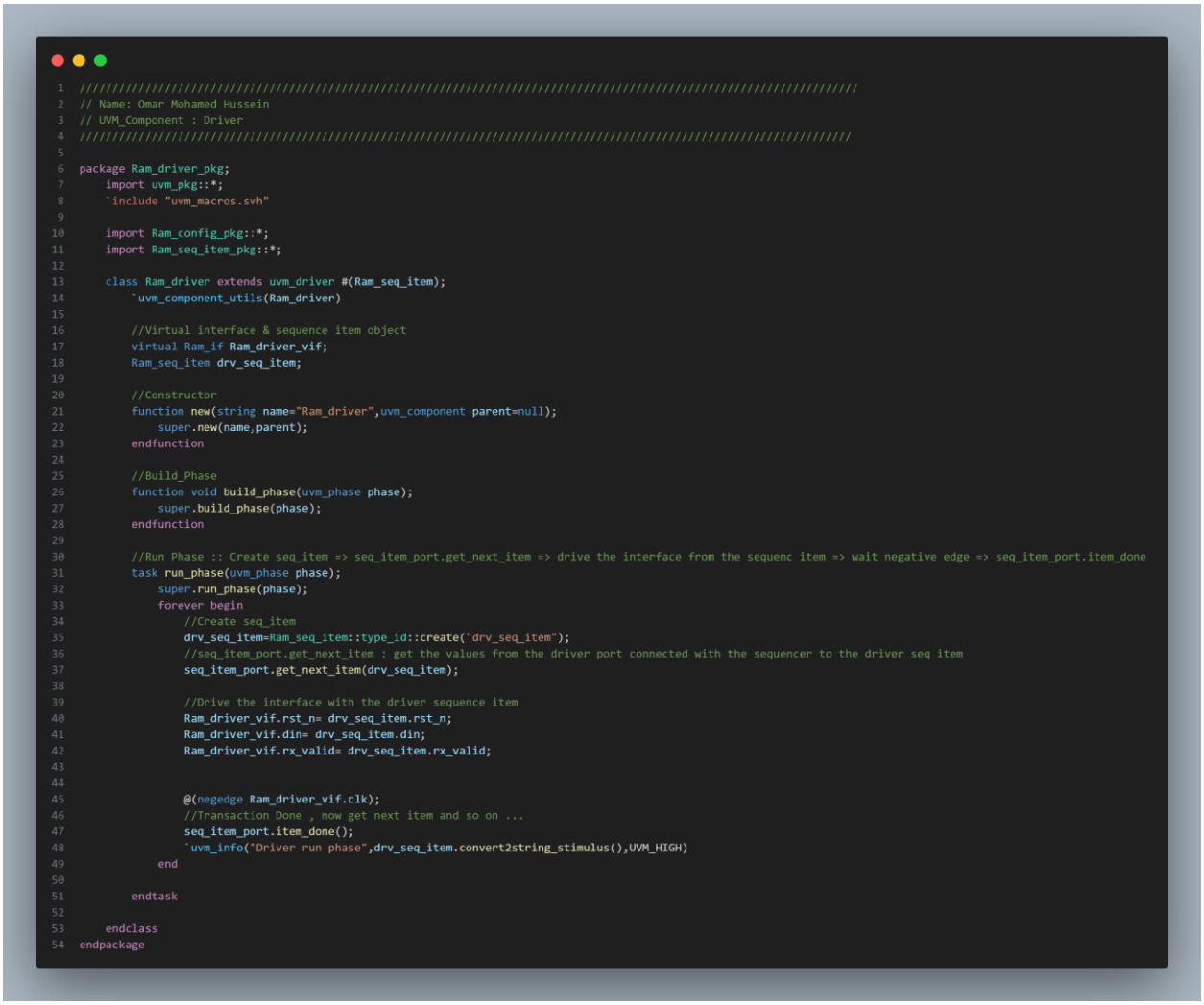
```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Object : Read Sequence
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_read_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import Ram_seq_item_pkg::*;

    class Ram_read_seq extends uvm_sequence #(Ram_seq_item);
    `uvm_object_utils(Ram_read_seq)

    Ram_seq_item seq_item;

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

    virtual task body();
        seq_item=Ram_seq_item::type_id::create("seq_item");
        repeat(1000) begin
            start_item(seq_item);
            //turn on or off certain constrains
            seq_item.write_only_seq_constrains.constraint_mode(0);
            seq_item.read_only_seq_constrains.constraint_mode(1);
            seq_item.read_write_constrains.constraint_mode(0);
            //Randomize & check for success randomization
            assert(seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
    endclass
endpackage
```

## 3.10.4.RAM Read Write Sequence



```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Object : Read Write Sequence
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_read_write_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import Ram_seq_item_pkg::*;

    class Ram_read_write_seq extends uvm_sequence #(Ram_seq_item);
    `uvm_object_utils(Ram_read_write_seq)

    Ram_seq_item seq_item;

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

    virtual task body();
        seq_item=Ram_seq_item::type_id::create("seq_item");
        repeat(3000) begin
            start_item(seq_item);
            //turn on or off certain constrains
            seq_item.write_only_seq_constrains.constraint_mode(0);
            seq_item.read_only_seq_constrains.constraint_mode(0);
            seq_item.read_write_constrains.constraint_mode(1);
            //Randomize & check for success randomization
            assert(seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
    endclass
endpackage
```

# 3.11.RAM Driver



```
////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Driver
////////////////////////////////////////////////////////////////////////////////////////////////////////

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

    import Ram_config_pkg::*;
    import Ram_seq_item_pkg::*;

    class Ram_driver extends uvm_driver #(Ram_seq_item);
        `uvm_component_utils(Ram_driver)

        //Virtual interface & sequence item object
        virtual Ram_if Ram_driver_vif;
        Ram_seq_item drv_seq_item;

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

        //Build_Phase
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
        endfunction

        //Run Phase :: Create seq_item => seq_item_port.get_next_item => drive the interface from the sequenc item => wait negative edge => seq_item_port.item_done
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                //Create seq_item
                drv_seq_item=Ram_seq_item::type_id::create("drv_seq_item");
                //seq_item_port.get_next_item : get the values from the driver port connected with the sequencer to the driver seq item
                seq_item_port.get_next_item(drv_seq_item);

                //Drive the interface with the driver sequence item
                Ram_driver_vif.rst_n= drv_seq_item.rst_n;
                Ram_driver_vif.din= drv_seq_item.din;
                Ram_driver_vif.rx_valid= drv_seq_item.rx_valid;


                @(negedge Ram_driver_vif.clk);
                //Transaction Done , now get next item and so on ...
                seq_item_port.item_done();
                `uvm_info("Driver run phase",drv_seq_item.convert2string_stimulus(),UVM_HIGH)
            end

        endtask

    endclass
endpackage
```

# 3.12.RAM Monitor

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Monitor
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_monitor_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import Ram_seq_item_pkg::*;
    class Ram_monitor extends uvm_monitor ;
        `uvm_component_utils(Ram_monitor)

        //Creating Virtual interface to get the data from the interface
        virtual Ram_if r_if;
        //Creating monitor sequence item to pass the interface to it to be then sent to the analysis components
        Ram_seq_item mon_seq_item;
        //Creating monitor analysis port to broadcast the values to the analysis components
        uvm_analysis_port #(Ram_seq_item) mon_ap;

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

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

        //Run Phase: samples the data at the negative edge and writes the analysis port with the monitor sequence item
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                mon_seq_item=Ram_seq_item::type_id::create("mon_seq_item");
                @(negedge r_if.clk);

                //Drive the monitor sequence item with the interface
                mon_seq_item.din         =r_if.din;
                mon_seq_item.rst_n       =r_if.rst_n;
                mon_seq_item.rx_valid    =r_if.rx_valid;
                mon_seq_item.dout        =r_if.dout;
                mon_seq_item.tx_valid    =r_if.tx_valid;
                mon_seq_item.tx_valid_exp=r_if.tx_valid_exp;
                mon_seq_item.dout_exp    =r_if.dout_exp;

                //Broadcast the monitor sequence (interfance) to the analysis port
                mon_ap.write(mon_seq_item);
                `uvm_info("Monitor run phase",mon_seq_item.convert2string(),UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

# 3.13.RAM Agent

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//Name: Omar Mohamed Hussein
//UVM_Component: Agent
////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_agent_pkg;

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

    import Ram_seq_item_pkg::*;
    import Ram_sqr_pkg::*;
    import Ram_driver_pkg::*;
    import Ram_monitor_pkg::*;
    import Ram_config_pkg::*;

    class Ram_agent extends uvm_agent;
        `uvm_component_utils(Ram_agent)

        //creating the sequencer,driver,monitor, configuration object to connect the interface,agent port
        Ram_sqr sqr;
        Ram_driver drv;
        Ram_monitor mon;
        Ram_config_obj config_obj;
        uvm_analysis_port #(Ram_seq_item) agt_ap;

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

        //Build Phase
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            //Connect the agent configuration object with the configuration object in the uvm config data base
            if(!uvm_config_db #(Ram_config_obj)::get(this,"","CFG",config_obj)) begin
                `uvm_fatal("agent build phase","unable to get the configuration object to the agent")
            end

            //Creating the sequencer,driver,monitor with the create() and the agt port with new()
            sqr=Ram_sqr    ::type_id::create("sqr",this);
            drv=Ram_driver ::type_id::create("drv",this);
            mon=Ram_monitor::type_id::create("mon",this);
            agt_ap=new("agt_ap",this);
        endfunction

        //connect phase
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            //Conncet the configuration object with the driver and the monitor ports
            drv.Ram_driver_vif=config_obj.Ram_config_vif;
            mon.r_if=config_obj.Ram_config_vif;
            //Connect the driver port with the sequencer export
            drv.seq_item_port.connect(sqr.seq_item_export);
            //Connect the monitor analysis port with the agt port
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage
```

# 3.14.RAM Scoreboard

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//Name: Omar Mohamed Hussein
//UVM_Component: Agent
////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_scoreboard_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import Ram_seq_item_pkg::*;

    class Ram_scoreboard extends uvm_scoreboard ;
        `uvm_component_utils(Ram_scoreboard)

        //Creating sequence item to get the data from fifo
        Ram_seq_item seq_item;
        //Creating analysis export to take values from the fifo
        uvm_analysis_export #(Ram_seq_item) sb_export;
        //Creating analysis fifo to take values from the agt port
        uvm_tlm_analysis_fifo #(Ram_seq_item) sb_fifo;

        int correct_count,error_count;

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

        //Build phase : create the export analysis
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb_export=new("sb_export",this);
            sb_fifo=new("sb_fifo",this);
        endfunction

        //Connect phase : connect the eb export with the fifo export
        function void connect_phase (uvm_phase phase);
            super.connect_phase(phase);
            sb_export.connect(sb_fifo.analysis_export);
        endfunction

        //run Task : Compare the output with the Golden model
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(seq_item);
                if(seq_item.dout!=seq_item.dout_exp || seq_item.tx_valid!=seq_item.tx_valid_exp) begin
                    `uvm_error("scoreboard run phase",$sformatf("Design doesn't match the Golden model => %s",seq_item.convert2string))
                    error_count++;
                end
                else begin
                    correct_count++;
                end
            end
        endtask

        //Report Phase: report the testbench with how many failed and correct trials
        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("Scoreboard Report Phase",$sformatf("Testbench Completed , Correct count=
, error count=
",correct_count,error_count),UVM_LOW)
        endfunction
endpackage
```

# 3.15.RAM Cover Collector

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Cover Collector
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_cover_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import Ram_seq_item_pkg::*;

    class Ram_cover extends uvm_component;
        `uvm_component_utils(Ram_cover)
        uvm_analysis_export #(Ram_seq_item) cover_ap;
        uvm_tlm_analysis_fifo #(Ram_seq_item) cover_fifo;
        Ram_seq_item seq_item;


    //Cover Groups
        covergroup Cov_gp(ref Ram_seq_item seq_item);
        //Normal Coverpoints
        operations_cp: coverpoint seq_item.din[9:8] {
            bins all_vals[]={[0:3]};
            bins Write_address_data=(2'b00=>2'b01);
            bins Read_address_data =(2'b10=>2'b11);
            bins all_operations =(2'b00=>2'b01=>2'b10=>2'b11);
        }
        rx_valid_cp:coverpoint seq_item.rx_valid{
            bins high={1};
            bins low={0};
        }
        //Cross coverage
        rx_valid_ops:cross operations_cp,rx_valid_cp {
            bins allvals_rx_valid = binsof(operations_cp.all_vals) && binsof(rx_valid_cp.high);
            bins read_data_tx_valid = binsof(operations_cp.all_vals) intersect {2'b11} && binsof(rx_valid_cp.high);
        }
        endgroup
        function new(string name="Ram_cover",uvm_component parent=null);
            super.new(name,parent);
            Cov_gp=new(seq_item);
        endfunction

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

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cover_ap.connect(cover_fifo.analysis_export);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                seq_item=Ram_seq_item::type_id::create("seq_item");
                cover_fifo.get(seq_item);
                Cov_gp.sample();
            end
        endtask

    endclass
endpackage
```

# 3.16.RAM Environment

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Cover Collector
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_env_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import Ram_driver_pkg::*;
import Ram_scoreboard_pkg::*;
import Ram_agent_pkg::*;
import Ram_cover_pkg::*;
    class Ram_env extends uvm_env;
        `uvm_component_utils(Ram_env)
        Ram_scoreboard sb;
        Ram_agent agt;
        Ram_cover cv;

        function new(string name="Ram_env",uvm_component parent=null);
            super.new(name,parent);
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb=Ram_scoreboard::type_id::create("sb",this);
            agt=Ram_agent::type_id::create("agt",this);
            cv=Ram_cover::type_id::create("cv",this);
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            agt.agt_ap.connect(sb.sb_export);
            agt.agt_ap.connect(cv.cover_ap);
        endfunction
    endclass
endpackage
```

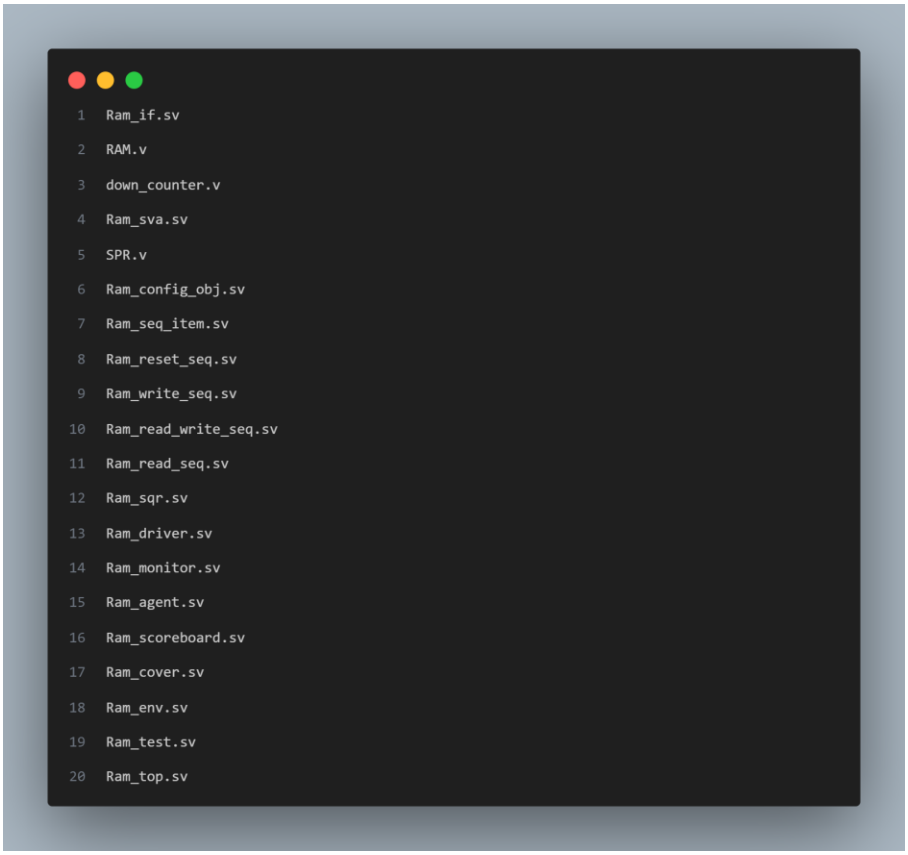
# 3.17.RAM Test

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Test
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_test_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import Ram_env_pkg::*;
import Ram_config_pkg::*;
import Ram_reset_seq_pkg::*;
import Ram_write_seq_pkg::*;
import Ram_read_write_seq_pkg::*;
import Ram_read_seq_pkg::*;
    class Ram_test extends uvm_test;
        `uvm_component_utils(Ram_test)

        Ram_env env;
        Ram_config_obj Ram_config_obj_test;
        Ram_reset_seq reset_seq;
        Ram_write_seq write_seq;
        Ram_read_seq read_seq;
        Ram_read_write_seq write_read_seq;
        function new(string name ="Ram_test",uvm_component parent=null);
            super.new(name,parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            env=Ram_env::type_id::create("env",this);
            Ram_config_obj_test=Ram_config_obj::type_id::create("Ram_config_obj_test");
            reset_seq=Ram_reset_seq::type_id::create("reset_seq");
            write_seq=Ram_write_seq::type_id::create("write_seq");
            read_seq=Ram_read_seq::type_id::create("read_seq");
            write_read_seq=Ram_read_write_seq::type_id::create("write_read_seq");

            if(!uvm_config_db #(virtual Ram_if)::get(this,"","RAM",Ram_config_obj_test.Ram_config_vif))
                `uvm_fatal("build phase in test","unable to get interface from the database into the configuration object")
            uvm_config_db#(Ram_config_obj)::set(this,"*","CFG",Ram_config_obj_test);
        endfunction

        task  run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            `uvm_info("Test run phase","Reset Asserted",UVM_LOW)
            reset_seq.start(env.agt.sqr);
            `uvm_info("Test run phase","Reset Finished",UVM_LOW)

            `uvm_info("Test run phase","Write sequence Asserted",UVM_LOW)
            write_seq.start(env.agt.sqr);
            `uvm_info("Test run phase","Write sequence Finished",UVM_LOW)

            `uvm_info("Test run phase","Read sequence Asserted",UVM_LOW)
            read_seq.start(env.agt.sqr);
            `uvm_info("Test run phase","Read sequence Finished",UVM_LOW)

            `uvm_info("Test run phase","Write Read sequence Asserted",UVM_LOW)
            write_read_seq.start(env.agt.sqr);
            `uvm_info("Test run phase","Write Read sequence Finished",UVM_LOW)
            phase.drop_objection(this);
        endtask
    endclass
endpackage
```

# 3.18.RAM Top

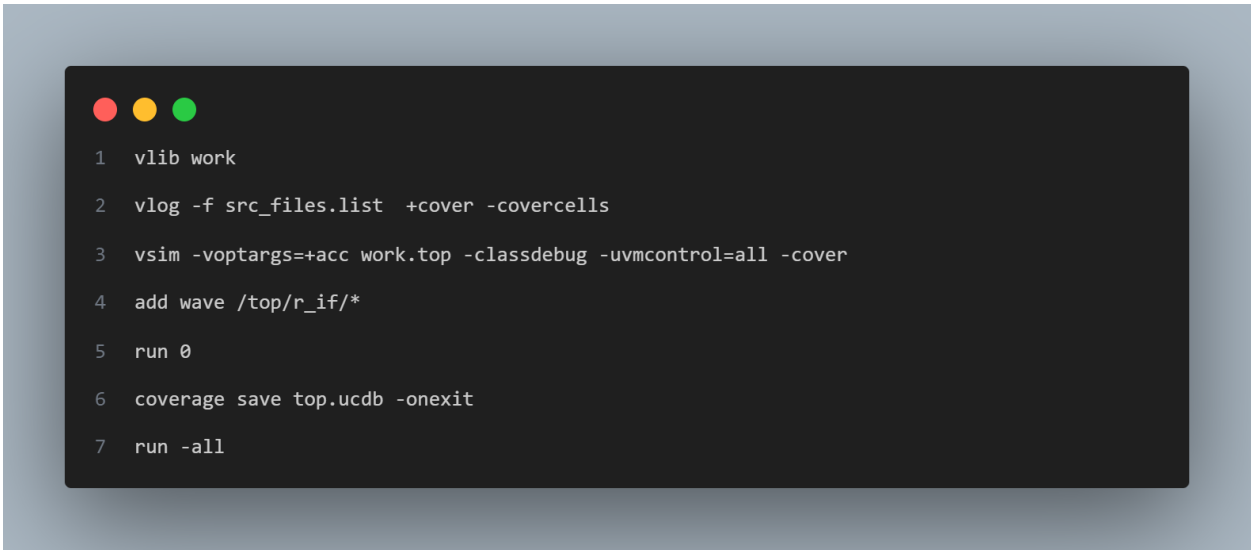
```
import uvm_pkg::*;
`include "uvm_macros.svh"
import Ram_test_pkg::*;

module top();
  // Clock generation
  bit clk;
  initial begin
    clk=0;
    forever begin
      #10 clk=~clk;
    end
  end
  // Instantiate the interface ,Golden Model and DUT
  Ram_if r_if(clk);
  RAM DUT(r_if.din,r_if.clk,r_if.rst_n,r_if.rx_valid,r_if.dout,r_if.tx_valid);
  SPR #(256,8) Golden (r_if.din,r_if.clk,r_if.rst_n,r_if.rx_valid,r_if.tx_valid_exp,r_if.dout_exp);
  bind RAM Ram_sva binded_assertions (r_if.din,r_if.clk,r_if.rst_n,r_if.rx_valid,r_if.dout,r_if.tx_valid);
  // run test using run_test task
  initial begin
    uvm_config_db #(virtual Ram_if)::set(null,"uvm_test_top","RAM",r_if);
    run_test("Ram_test");
  end
endmodule
```

# 3.19.RAM Source File



```
Ram_if.sv
RAM.v
down_counter.v
Ram_sva.sv
SPR.v
Ram_config_obj.sv
Ram_seq_item.sv
Ram_reset_seq.sv
Ram_write_seq.sv
Ram_read_write_seq.sv
Ram_read_seq.sv
Ram_sqr.sv
Ram_driver.sv
Ram_monitor.sv
Ram_agent.sv
Ram_scoreboard.sv
Ram_cover.sv
Ram_env.sv
Ram_test.sv
Ram_top.sv
```

# 3.20.RAM DO File



```
vlib work
vlog -f src_files.list  +cover -covercells
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
add wave /top/r_if/*
run 0
coverage save top.ucdb -onexit
run -all
```

# 3.21.RAM Assertions

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

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

sequence ops;
    (din[9:8]==2'b00 || din[9:8]==2'b01 || din[9:8]==2'b10) && rx_valid;
endsequence
assert property(@(posedge clk) !rst_n |=> dout==0 && !tx_valid);
assert property(@(posedge clk) disable iff(!rst_n) ops |=> !tx_valid );
assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b11 |=> $rose(tx_valid) |=> $fell(tx_valid)[->1]);
assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b00 |=>  (rx_valid && din[9:8]== 2'b01)[->1]);
assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b10 |=>  (rx_valid && din[9:8]== 2'b11)[->1]);

cover  property(@(posedge clk) !rst_n |=> dout==0 && !tx_valid);
cover  property(@(posedge clk) disable iff(!rst_n) ops |=> !tx_valid && dout==0);
cover  property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b11 |=> $rose(tx_valid) |=> $fell(tx_valid)[->1]);
cover  property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b00 |=>  (rx_valid && din[9:8]== 2'b01)[->1]);
cover  property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b10 |=>  (rx_valid && din[9:8]== 2'b11)[->1]);

endmodule
```

## Assertion Table:

| Feature | Assertions |
|---|---|
| If the reset is asserted, the dout and tx_valid should be zeros | (@(posedge clk) !rst_n \|=> dout==0 && !tx_valid) |
| IF the reset is deasserted , in case of all operations except for read_data the tx_valid should be low. | (@(posedge clk) disable iff(!rst_n) ops \|=> !tx_valid ) |
| IF the reset is deasserted , in case for read data , the tx_valid should rise and then eventually falls. | (@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b11 \|=> $rose(tx_valid) \|=> $fell(tx_valid)[->1]) |
| IF the reset is deasserted , incase for write address ,eventually write data will come. | (@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b00 \|=>  (rx_valid && din[9:8]== 2'b01)[->1]) |
| IF the reset is deasserted , incase for read address ,eventually read data will come. | (@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b10 \|=>  (rx_valid && din[9:8]== 2'b11)[->1]) |

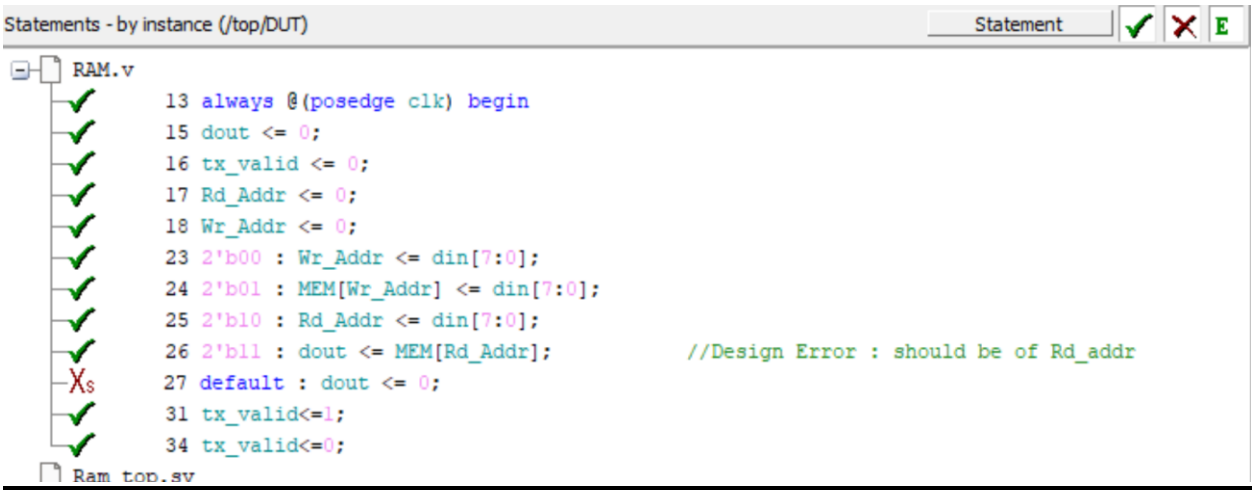
# 3.22.Simulation Results

## 3.22.1.Questasim Transcript

```
# ****************************************************************
# UVM_INFO Ram_test.sv(42) @ 20: uvm_test_top [Test run phase] Reset Finished
# UVM_INFO Ram_test.sv(44) @ 20: uvm_test_top [Test run phase] Write sequence Asserted
# UVM_INFO Ram_test.sv(46) @ 20020: uvm_test_top [Test run phase] Write sequence Finished
# UVM_INFO Ram_test.sv(48) @ 20020: uvm_test_top [Test run phase] Read sequence Asserted
# UVM_INFO Ram_test.sv(50) @ 40020: uvm_test_top [Test run phase] Read sequence Finished
# UVM_INFO Ram_test.sv(52) @ 40020: uvm_test_top [Test run phase] Write Read sequence Asserted
# UVM_INFO Ram_test.sv(54) @ 100020: uvm_test_top [Test run phase] Write Read sequence Finished
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 100020: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO Ram_scoreboard.sv(54) @ 100020: uvm_test_top.env.sb [Scoreboard Report Phase] Testbench Completed , Correct count=5001, error count=0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   13
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [Scoreboard Report Phase]     1
# [TEST_DONE]     1
# [Test run phase]     8
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 100020 ns  Iteration: 61  Instance: /top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```

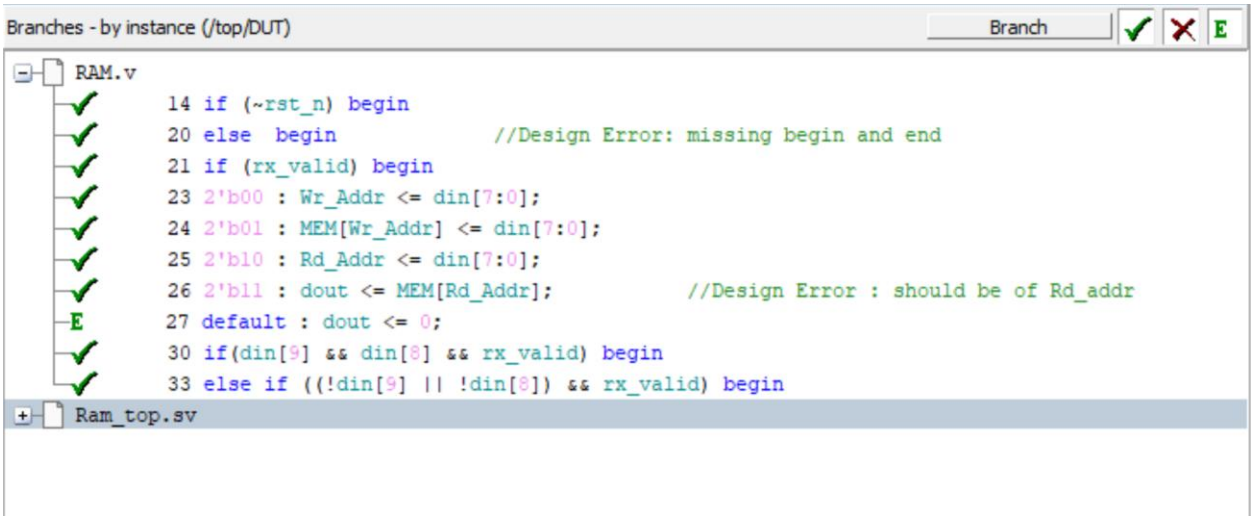
## 3.22.2.Questasim Code Coverage

### Statement Coverage:




- Note: We excluded the default statement as we will never hit it.

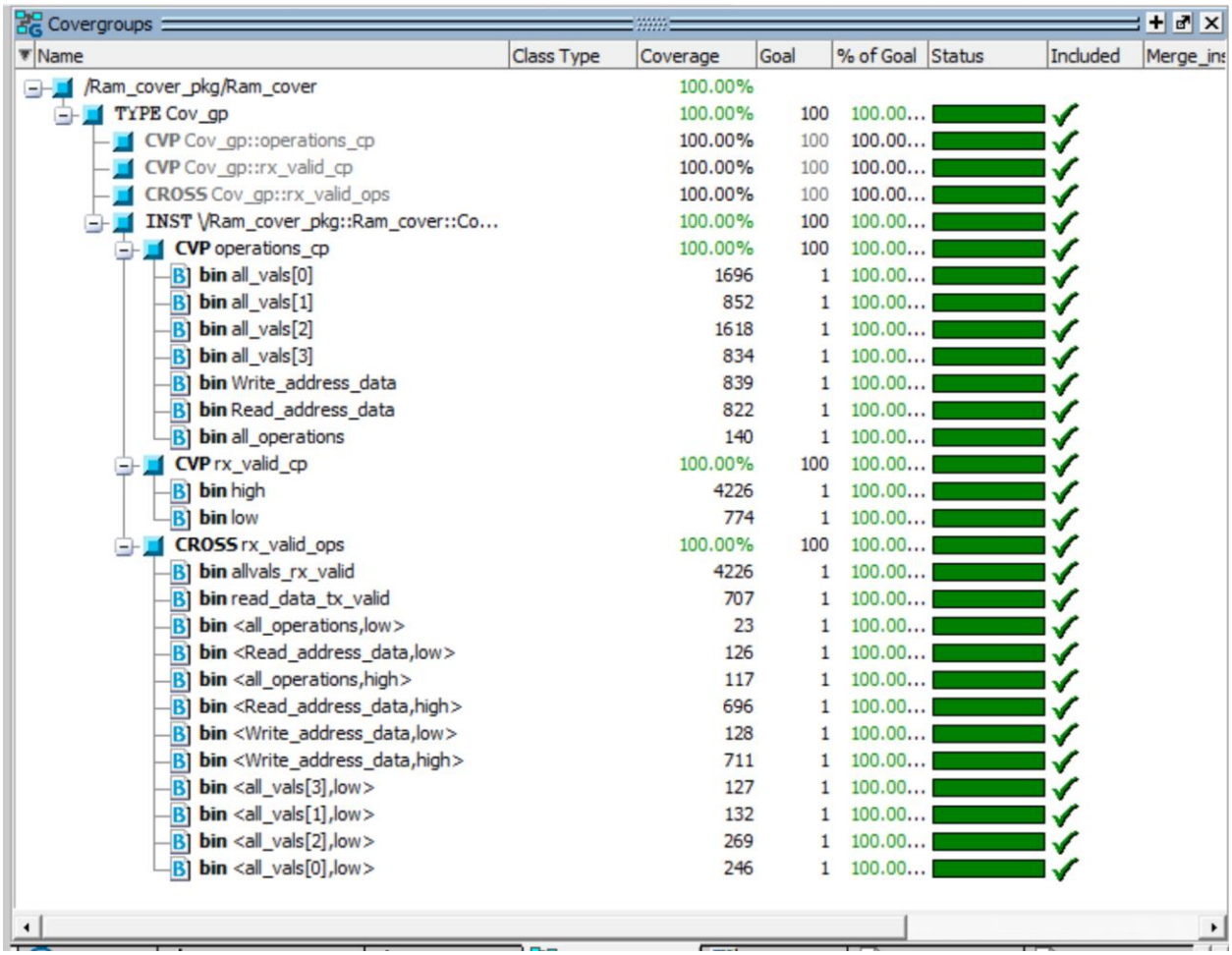
### Branch Coverage:



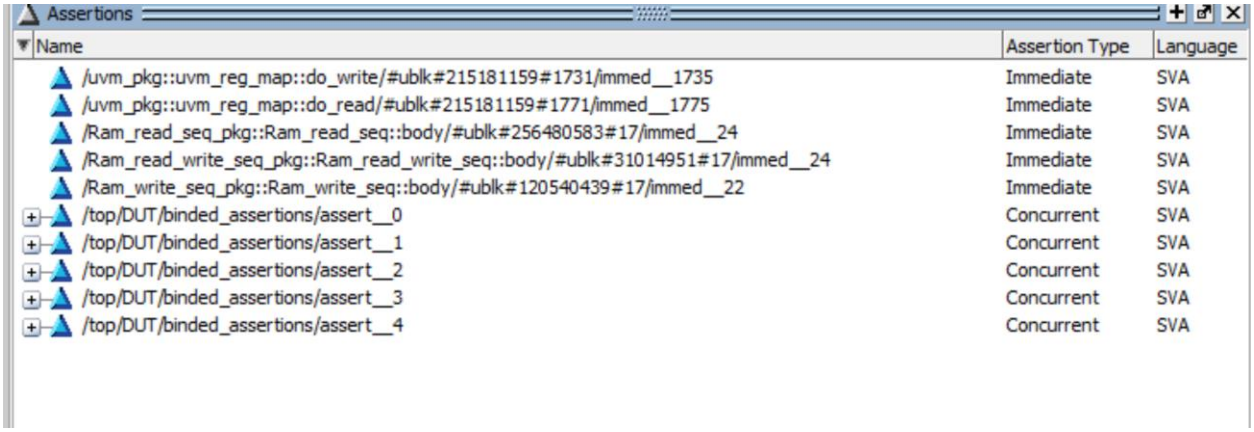

- Note: We excluded the default statement as we will never hit it.

## Toggle Coverage:

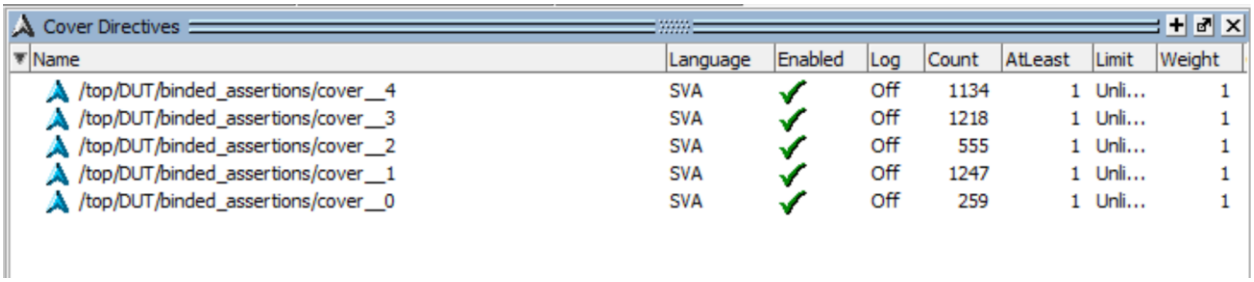

## 3.22.3.Questasim Functional Coverage



## 3.22.4.Questasim Assertions

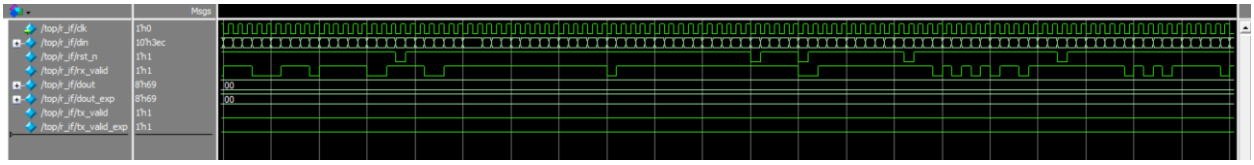



## 3.22.5.Questasim Assertions Coverage

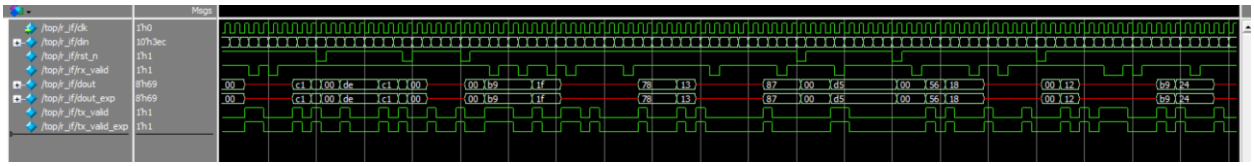



## 3.22.6.Questasim Waveform

- Write Sequence:



- Read Sequence:



# *4.Wrapper UVM Verification*

## 4.1.Verification Requirements

In this part, you will build a complete UVM environment to verify the end to end functionality of the **SPI wrapper** design. The environment should integrate both the **Single-Port RAM** and the **SPI Slave** environments developed previously to reuse them but with **passive** agents (check the last extra assignment to learn how to do it).

**Constraint Requirements:**

1- The reset signal (rst_n) shall be deasserted most of the time.

2- The SS_n signal to be high for one cycle every 13 cycles for all cases except read data to be high for one cycle every 23 cycles

3- Declare a randomized array of 11 bits to drive bit by bit the MOSI and the first 3 bits sent serially to the MOSI when the SS_n falls are valid combinations only (000, 001, 110, 111)

4- For a *write-only sequence*, every Write Address operation shall always be followed by either Write Address or Write Data operation.

5- For a *read-only sequence*, every Read Address operation shall always be followed by either Read Address or Read Data operation.

6- For a ***randomized read/write sequence***, the following ordering rules shall be enforced:

- Every Write Address operation shall always be followed by either Write Address or Write Data operation.
- After a Write Data, the next operation shall be chosen with the following probability distribution: 60% → Read Address & 40% → Write Address
- Every Read Address operation shall always be followed by either Read Address or Read Data operation.After a Read Data, the next operation shall be chosen with the following probability distribution: 60% → Write Address & 40% → Read Address

***(Hint: You can use post_randomize)***

**Assertions Requirements:**

1- An assertion ensures that whenever reset is asserted, the outputs (MISO, rx_valid, and rx_data) are all inactive.

2- An assertion to make sure that the MISO remains with a stable value as long as it is not a read data operation

>> Bind the assertion modules for the RAM and SPI Slave and SPI wrapper in the top module

**Sequences Requirements:**

Split the sequences into the following sequences. You can switch on and off the constraints or use inline constraints to control the sequence item generated in the sequence

- reset_sequence
- write_only_sequence
- read_only_sequence
- write_read_sequence

**Note:** You are free to add assertions, covergroups or constraints to enrich your verification.

# 4.2.Wrapper Design

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

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

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

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

endmodule


```

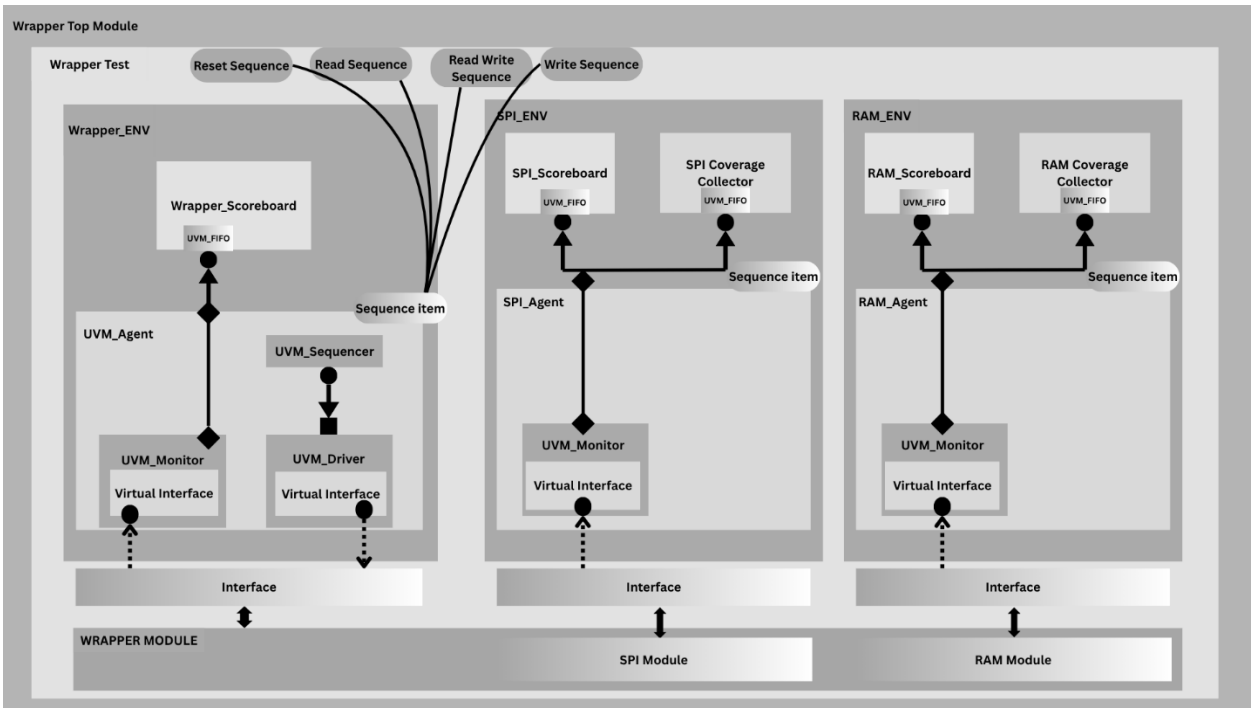
# 4.3.Wrapper Interface

```
interface wrapper_if(clk);
    input clk;
    logic SS_n,rst_n,MOSI,MISO,MISO_exp;
endinterface
```

# 4.4.Verification Plan

| Label | Design Requirment Description | Stimulus generation | Functional coverage | Functionality check |
|---|---|---|---|---|
| reset check | when the reset signal is activated , the MISO ,rx_valid,rx_data should be zero. | directed reset assertion using reset sequence | | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the spi_sva file. |
| IDLE | when the reset signal is deactivated and the SS_n is High , the cs should be IDLE | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. | Covering the Transition of the SS_n after 13 cycle except for the READ_DATA would be for 23 cycle | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the spi_sva file. |
| CHK_CMD | when the reset signal is deactivated and the SS_n is Low and the cs is IDLE, the cs should be CHK_CMD | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. | Covering the Transition of the SS_n after 13 cycle except for the READ_DATA would be for 23 cycle | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the spi_sva file. |
| WRITE_ADDR | when the reset signal is deactivated and the SS_n is Low and the cs is CHK_CMD and the MOSI is taking the following sequence 3'b000, the cs should be WRITE ADDRESS. | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. Also Randomizing variable called valid_comb to have the most significant 3 bits with valid combinations such as WRITE_ADD , WRITE_DATA , READ_ADD , WRITE_DATA. | Covering the Transition of the MOSI to cover the sequence of the WRITE_ADDR, WRITE_DATA, READ_ADDR, READ_DATA and crossing it with the SS_n coverpoint. | In the scoreboard , comparing the outputs with the expected of a Golden model. |
| WRITE_DATA | when the reset signal is deactivated and the SS_n is Low and the cs is CHK_CMD and the MOSI is taking the following sequence 3'b001, the cs should be WRITE DATA. | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. Also Randomizing variable called valid_comb to have the most significant 3 bits with valid combinations such as WRITE_ADD , WRITE_DATA , READ_ADD , WRITE_DATA. | Covering the Transition of the MOSI to cover the sequence of the WRITE_ADDR, WRITE_DATA, READ_ADDR, READ_DATA and crossing it with the SS_n coverpoint. | In the scoreboard , comparing the outputs with the expected of a Golden model. |
| READ_ADD | when the reset signal is deactivated and the SS_n is Low and the cs is CHK_CMD and the MOSI is taking the following sequence 3'b110, the cs should be WRITE ADDRESS. | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. Also Randomizing variable called valid_comb to have the most significant 3 bits with valid combinations such as WRITE_ADD , WRITE_DATA , READ_ADD , WRITE_DATA. | Covering the Transition of the MOSI to cover the sequence of the WRITE_ADDR, WRITE_DATA, READ_ADDR, READ_DATA and crossing it with the SS_n coverpoint. | In the scoreboard , comparing the outputs with the expected of a Golden model. |
| READ_DATA | when the reset signal is deactivated and the SS_n is Low and the cs is CHK_CMD and the MOSI is taking the following sequence 3'b111, the cs should be WRITE ADDRESS. | Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA the SS_n would be low for 23 cycle. Also Randomizing variable called valid_comb to have the most significant 3 bits with valid combinations such as WRITE_ADD , WRITE_DATA , READ_ADD , WRITE_DATA. | Covering the Transition of the MOSI to cover the sequence of the WRITE_ADDR, WRITE_DATA, READ_ADDR, READ_DATA and crossing it with the SS_n coverpoint. | In the scoreboard , comparing the outputs with the expected of a Golden model. |
| reset check | when the reset signal is activated , the tx_valid,dataout are set low | directed reset assertion using reset sequence | | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the Ram_sva file. |
| IDLE | when the reset signal is deactivated and the rx_valid is high, if the din[9:8] are 2'b00, the remaining bus should be registered as write address. | Randomizing the din and forcing din[9:8] to take write address or data only in the write sequence only | Covering the Transition from write address to write data crossing it with the rx_valid possible values | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the Ram_sva file. |
| CHK_CMD | when the reset signal is deactivated and the rx_valid is high, if the din[9:8] are 2'b01, the remaining bus should be registered as write data in the registerd address before. | Randomizing the din and forcing din[9:8] to take write address or data only in the write sequence only | Covering the Transition from write address to write data crossing it with the rx_valid possible values | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the Ram_sva file. |
| WRITE_ADDR | when the reset signal is deactivated and the rx_valid is high, if the din[9:8] are 2'b10, the remaining bus should be registered as read address. | Randomizing the din and forcing din[9:8] to take read address or data only in the write sequence only | Covering the Transition from read address to readdata crossing it with the rx_valid possible values | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the Ram_sva file. |
| WRITE_DATA | when the reset signal is deactivated and the rx_valid is high, if the din[9:8] are 2'b11, the remaining bus should be registered as read data in the registerd address before. | Randomizing the din and forcing din[9:8] to take read address or data only in the write sequence only | Covering the Transition from read address to readdata crossing it with the rx_valid possible values | In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the Ram_sva file. |

# 4.5.UVM Diagram

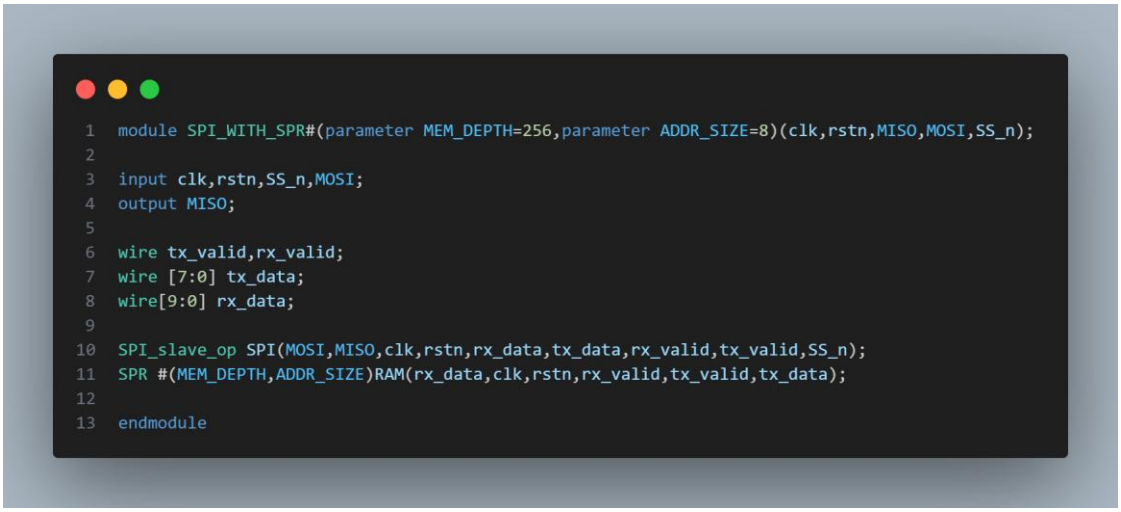



**Verification Flow:**

- TOP Module instantiates the interface, Wrapper Design, Golden Model and bind assertions Of Both the Ram and the SPI, then connect the SPI and Ram interfaces with there modules inside the Wrapper, Setting the three interfaces in the configuration object and run test.
- Test creates the three environments: Wrapper , SPI and Ram environments and starts sequences: Reset Sequence, Write Sequence, Read sequence and Read Write Sequence.
- Environments Creates the Agent, Scoreboard and the Cover Collector.
- Also Connecting the Agent analysis port with the Scoreboard and Cover Collector analysis export ports.
- Wrapper Agent is active agent where it creates sequencer, driver and the monitor , connecting the sequencer port with the driver and connecting the analysis port of the monitor with the analysis port of the Agent.
- SPI and RAM Agents are passive agents where it creates only the monitor , connecting the analysis port of the monitor with the analysis port of the Agent and monitoring the interface signals inside the sequence item then broadcast it to the agent port.

- Wrapper Driver uses a virtual interface to drive Wrapper Design pins.
- Both Scoreboard and Coverage Collector for each agent receive transaction objects produced by monitors through TLM analysis channels and perform checking and coverage collection.

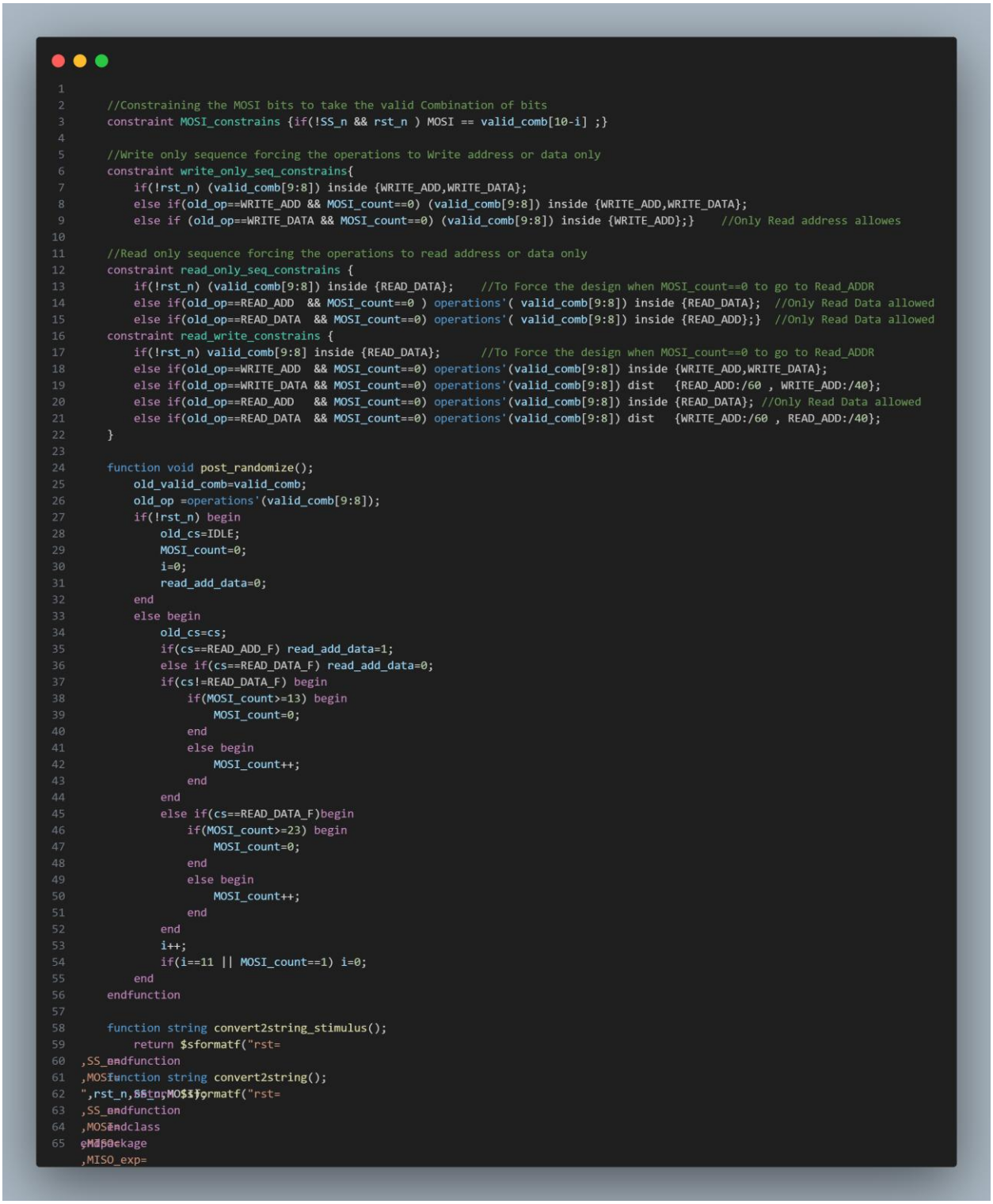
# 4.6.Golden Model



```
1  module SPI_WITH_SPR#(parameter MEM_DEPTH=256,parameter ADDR_SIZE=8)(clk,rstn,MISO,MOSI,SS_n);
2
3  input clk,rstn,SS_n,MOSI;
4  output MISO;
5
6  wire tx_valid,rx_valid;
7  wire [7:0] tx_data;
8  wire[9:0] rx_data;
9
10 SPI_slave_op SPI(MOSI,MISO,clk,rstn,rx_data,tx_data,rx_valid,tx_valid,SS_n);
11 SPR #(MEM_DEPTH,ADDR_SIZE)RAM(rx_data,clk,rstn,rx_valid,tx_valid,tx_data);
12
13 endmodule
```

# 4.7.Wrapper Sequence item

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Object : Wrapper Sequence Item
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

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

    class wrapper_seq_item extends uvm_sequence_item;
    `uvm_object_utils(wrapper_seq_item)

    //Constructor function
    function new(string name="wrapper_seq_item");
        super.new(name);
    endfunction

    //Type define
    typedef enum  {IDLE,CHK_CMD,WRITE,READ_ADD_F,READ_DATA_F} my_FSM;
    typedef enum  {WRITE_ADD,WRITE_DATA,READ_ADD,READ_DATA} operations;

    //inputs rand signals
    rand logic     [9:0] din;
    rand logic  MOSI, rst_n, SS_n;
    //output signals
    logic MISO,MISO_exp;
    //Internal signals and sequence item counters
    rand logic [10:0] valid_comb;
    logic [10:0] old_valid_comb;
    int MOSI_count,i,read_add_data;
    my_FSM  old_cs;
    rand my_FSM  cs;
    operations  old_op;

    constraint rst_constrains {rst_n dist{1:/95 , 0:/5};}
    constraint SS_n_constrains {
        if (old_cs != READ_DATA_F ) {
          (MOSI_count == 13) -> SS_n == 1;
          (MOSI_count != 13) -> SS_n == 0;
        }
        else if(old_cs==READ_DATA_F) {
          (MOSI_count == 23) -> SS_n == 1;
          (MOSI_count != 23) -> SS_n == 0;
        }
    }

    //Creating a FSM Enviroment inside the Sequence item
    constraint cs_constrains{if(!rst_n || SS_n) {cs==IDLE;}
                             else if(old_cs==IDLE && !SS_n) {cs==CHK_CMD;}
                             else if(old_cs==CHK_CMD && !SS_n && !MOSI) {cs==WRITE;}
                             else if(old_cs==CHK_CMD && !SS_n && MOSI && read_add_data==0) {cs==READ_ADD_F;}
                             else if(old_cs==CHK_CMD && !SS_n && MOSI && read_add_data==1) {cs==READ_DATA_F;}
                             else if((old_cs==WRITE || old_cs==READ_ADD_F) && MOSI_count==13 ) {cs==IDLE;}
                             else if((old_cs==WRITE) && MOSI_count!=13 ) {cs==WRITE;}
                             else if((old_cs==READ_ADD_F) && MOSI_count!=13 ) {cs==READ_ADD_F;}
                             else if((old_cs==READ_DATA_F) && MOSI_count==23 ) {cs==IDLE;}
                             else if((old_cs==READ_DATA_F) && MOSI_count!=23 ) {cs==READ_DATA_F;}
    }
    //Randomizing a 11 bit bus forcing it to start with 3 valid bits, each Randomization will be at the start of each sequence
    constraint valid_comb_constrains{
        if(!rst_n || MOSI_count==0) valid_comb[10:8] inside{3'b000,3'b001,3'b110,3'b111};
        else valid_comb==old_valid_comb;}

```



```
1
2       //Constraining the MOSI bits to take the valid Combination of bits
3       constraint MOSI_constrains {if(!SS_n && rst_n ) MOSI == valid_comb[10-i] ;}
4
5       //Write only sequence forcing the operations to Write address or data only
6       constraint write_only_seq_constrains{
7           if(!rst_n) (valid_comb[9:8]) inside {WRITE_ADD,WRITE_DATA};
8           else if(old_op==WRITE_ADD && MOSI_count==0) (valid_comb[9:8]) inside {WRITE_ADD,WRITE_DATA};
9           else if (old_op==WRITE_DATA && MOSI_count==0) (valid_comb[9:8]) inside {WRITE_ADD};}    //Only Read address allowes
10
11      //Read only sequence forcing the operations to read address or data only
12      constraint read_only_seq_constrains {
13          if(!rst_n) (valid_comb[9:8]) inside {READ_DATA};    //To Force the design when MOSI_count==0 to go to Read_ADDR
14          else if(old_op==READ_ADD  && MOSI_count==0 ) operations'( valid_comb[9:8]) inside {READ_DATA};  //Only Read Data allowed
15          else if(old_op==READ_DATA  && MOSI_count==0) operations'( valid_comb[9:8]) inside {READ_ADD};}  //Only Read Data allowed
16      constraint read_write_constrains {
17          if(!rst_n) valid_comb[9:8] inside {READ_DATA};      //To Force the design when MOSI_count==0 to go to Read_ADDR
18          else if(old_op==WRITE_ADD  && MOSI_count==0) operations'(valid_comb[9:8]) inside {WRITE_ADD,WRITE_DATA};
19          else if(old_op==WRITE_DATA && MOSI_count==0) operations'(valid_comb[9:8]) dist   {READ_ADD:/60 , WRITE_ADD:/40};
20          else if(old_op==READ_ADD   && MOSI_count==0) operations'(valid_comb[9:8]) inside {READ_DATA}; //Only Read Data allowed
21          else if(old_op==READ_DATA  && MOSI_count==0) operations'(valid_comb[9:8]) dist   {WRITE_ADD:/60 , READ_ADD:/40};
22      }
23
24      function void post_randomize();
25          old_valid_comb=valid_comb;
26          old_op =operations'(valid_comb[9:8]);
27          if(!rst_n) begin
28              old_cs=IDLE;
29              MOSI_count=0;
30              i=0;
31              read_add_data=0;
32          end
33          else begin
34              old_cs=cs;
35              if(cs==READ_ADD_F) read_add_data=1;
36              else if(cs==READ_DATA_F) read_add_data=0;
37              if(cs!=READ_DATA_F) begin
38                  if(MOSI_count>=13) begin
39                      MOSI_count=0;
40                  end
41                  else begin
42                      MOSI_count++;
43                  end
44              end
45              else if(cs==READ_DATA_F)begin
46                  if(MOSI_count>=23) begin
47                      MOSI_count=0;
48                  end
49                  else begin
50                      MOSI_count++;
51                  end
52              end
53              i++;
54              if(i==11 || MOSI_count==1) i=0;
55          end
56      endfunction
57
58      function string convert2string_stimulus();
59          return $sformatf("rst=
60  ,SS_endfunction
61  ,MOSIunction string convert2string();
62  ",rst_n,SS_n,MOSI$$formatf("rst=
63  ,SS_endfunction
64  ,MOSIendclass
65  ,MISO_exp=
```

# 4.8.Wrapper Configuration Object

```
package wrapper_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class wrapper_config_obj extends uvm_object;
        `uvm_object_utils(wrapper_config_obj)

        virtual wrapper_if wrapper_config_vif;
        uvm_active_passive_enum is_active;
        function new(string name="alsu_config_obj");
            super.new(name);
        endfunction

    endclass
endpackage
```

# 4.9.Wrapper Sequences

## 4.9.1.Reset Sequence

```
package wrapper_reset_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import wrapper_seq_item::*;

    class wrapper_reset_seq extends uvm_sequence #(wrapper_seq_item);
    `uvm_object_utils(wrapper_reset_seq)

    wrapper_seq_item seq_item;

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

    task body;
        seq_item=wrapper_seq_item::type_id::create("seq_item");
        start_item(seq_item);
        seq_item.rst_n=0;
        finish_item(seq_item);
    endtask

    endclass
endpackage
```

## 4.9.2.Write Only Sequence

```
package wrapper_write_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import wrapper_seq_item::*;

    class wrapper_write_seq extends uvm_sequence #(wrapper_seq_item);
    `uvm_object_utils(wrapper_write_seq)

    wrapper_seq_item seq_item;

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

    virtual task body();
        seq_item=wrapper_seq_item::type_id::create("seq_item");
        seq_item.write_only_seq_constrains.constraint_mode(1);
        seq_item.read_only_seq_constrains.constraint_mode(0);
        seq_item.read_write_constrains.constraint_mode(0);
        repeat(500) begin
            start_item(seq_item);
            assert(seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
    endclass
endpackage
```

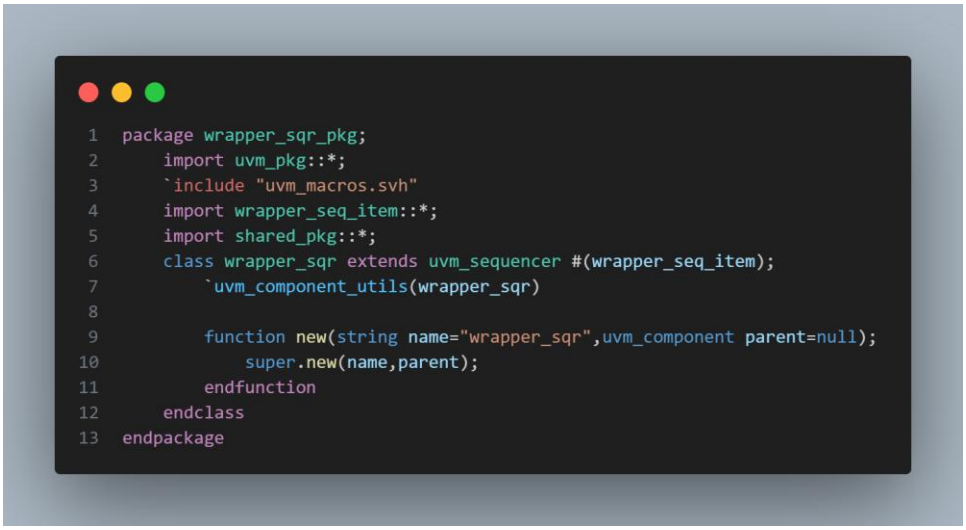
## 4.9.3.Read Only Sequence

```
package wrapper_read_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import wrapper_seq_item::*;

    class wrapper_read_seq extends uvm_sequence #(wrapper_seq_item);
    `uvm_object_utils(wrapper_read_seq)

    wrapper_seq_item seq_item;

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

    virtual task body();
        seq_item=wrapper_seq_item::type_id::create("seq_item");
        seq_item.write_only_seq_constrains.constraint_mode(0);
        seq_item.read_only_seq_constrains.constraint_mode(1);
        seq_item.read_write_constrains.constraint_mode(0);
        repeat(1000) begin
            start_item(seq_item);
            assert(seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
    endclass
endpackage
```

### 4.9.4.Read Write Sequence

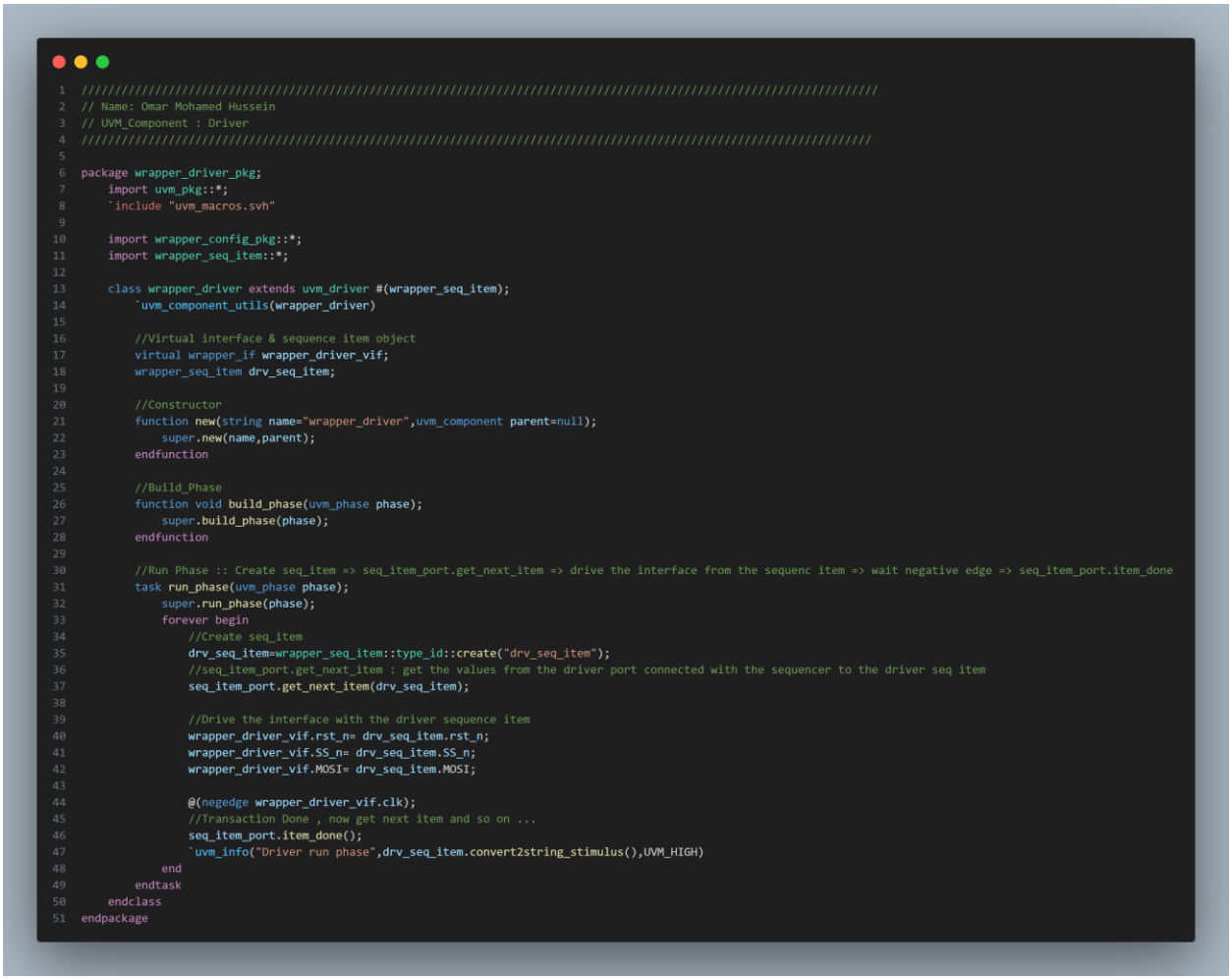
```
package wrapper_read_write_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import wrapper_seq_item::*;

    class wrapper_read_write_seq extends uvm_sequence #(wrapper_seq_item);
    `uvm_object_utils(wrapper_read_write_seq)

    wrapper_seq_item seq_item;

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

    virtual task body();
        seq_item=wrapper_seq_item::type_id::create("seq_item");
        seq_item.write_only_seq_constrains.constraint_mode(0);
        seq_item.read_only_seq_constrains.constraint_mode(0);
        seq_item.read_write_constrains.constraint_mode(1);
        repeat(500) begin
            start_item(seq_item);
            assert(seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
    endclass
endpackage
```

## 4.10.Wrapper sequencer



```
package wrapper_sqr_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import wrapper_seq_item::*;
    import shared_pkg::*;
    class wrapper_sqr extends uvm_sequencer #(wrapper_seq_item);
        `uvm_component_utils(wrapper_sqr)

        function new(string name="wrapper_sqr",uvm_component parent=null);
            super.new(name,parent);
        endfunction
    endclass
endpackage
```

# 4.11.Wrapper Driver



```
////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Component : Driver
////////////////////////////////////////////////////////////////////////////////////////////////////////

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

    import wrapper_config_pkg::*;
    import wrapper_seq_item::*;

    class wrapper_driver extends uvm_driver #(wrapper_seq_item);
        `uvm_component_utils(wrapper_driver)

        //Virtual interface & sequence item object
        virtual wrapper_if wrapper_driver_vif;
        wrapper_seq_item drv_seq_item;

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

        //Build_Phase
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
        endfunction

        //Run Phase :: Create seq_item => seq_item_port.get_next_item => drive the interface from the sequenc item => wait negative edge => seq_item_port.item_done
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                //Create seq_item
                drv_seq_item=wrapper_seq_item::type_id::create("drv_seq_item");
                //seq_item_port.get_next_item : get the values from the driver port connected with the sequencer to the driver seq item
                seq_item_port.get_next_item(drv_seq_item);

                //Drive the interface with the driver sequence item
                wrapper_driver_vif.rst_n= drv_seq_item.rst_n;
                wrapper_driver_vif.SS_n= drv_seq_item.SS_n;
                wrapper_driver_vif.MOSI= drv_seq_item.MOSI;

                @(negedge wrapper_driver_vif.clk);
                //Transaction Done , now get next item and so on ...
                seq_item_port.item_done();
                `uvm_info("Driver run phase",drv_seq_item.convert2string_stimulus(),UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

# 4.12.Wrapper Monitor

```
package wrapper_monitor_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import wrapper_seq_item::*;
    import shared_pkg::*;
    class wrapper_monitor extends uvm_monitor ;
        `uvm_component_utils(wrapper_monitor)

        //Creating Virtual interface to get the data from the interface
        virtual wrapper_if w_if;
        //Creating monitor sequence item to pass the interface to it to be then sent to the analysis components
        wrapper_seq_item mon_seq_item;
        //Creating monitor analysis port to broadcast the values to the analysis components
        uvm_analysis_port #(wrapper_seq_item) mon_ap;

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

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

        //Run Phase: samples the data at the negative edge and writes the analysis port with the monitor sequence item
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                mon_seq_item=wrapper_seq_item::type_id::create("mon_seq_item");
                @(negedge w_if.clk);
                mon_seq_item.rst_n=w_if.rst_n;
                mon_seq_item.SS_n=w_if.SS_n;
                mon_seq_item.MOSI=w_if.MOSI;
                mon_seq_item.MISO=w_if.MISO;
                mon_seq_item.MISO_exp=w_if.MISO_exp;
                //Drive the monitor sequence item with the interface

                //Broadcast the monitor sequence (interfance) to the analysis port
                mon_ap.write(mon_seq_item);
                `uvm_info("Monitor run phase",mon_seq_item.convert2string(),UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

# 4.13.Wrapper Agent

```
////////////////////////////////////////////////////////////////////////////////////////////////////////
//Name: Omar Mohamed Hussein
//UVM_Component:Agent Package
////////////////////////////////////////////////////////////////////////////////////////////////////////
package wrapper_agent_pkg;

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

    import wrapper_seq_item::*;
    import wrapper_sqr_pkg::*;
    import wrapper_driver_pkg::*;
    import wrapper_monitor_pkg::*;
    import wrapper_config_pkg::*;

    class wrapper_agent extends uvm_agent;
        `uvm_component_utils(wrapper_agent)

        //creating the sequencer,driver,monitor, configuration object to connect the interface,agent port
        wrapper_sqr sqr;
        wrapper_driver drv;
        wrapper_monitor mon;
        wrapper_config_obj config_obj;
        uvm_analysis_port #(wrapper_seq_item) agt_ap;

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

        //Build Phase
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            //Connect the agent configuration object with the configuration object in the uvm config data base
            if(!uvm_config_db #(wrapper_config_obj)::get(this,"","CFG_WRAPPER",config_obj)) begin
                `uvm_fatal("agent build phase","unable to get the configuration object to the agent")
            end

            //Creating the sequencer,driver,monitor with the create() and the agt port with new()
            sqr=wrapper_sqr    ::type_id::create("sqr",this);
            drv=wrapper_driver ::type_id::create("drv",this);
            mon=wrapper_monitor::type_id::create("mon",this);
            agt_ap=new("agt_ap",this);
        endfunction

        //connect phase
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            //Conncet the configuration object with the driver and the monitor ports
            drv.wrapper_driver_vif=config_obj.wrapper_config_vif;
            mon.w_if=config_obj.wrapper_config_vif;
            //Connect the driver port with the sequencer export
            drv.seq_item_port.connect(sqr.seq_item_export);
            //Connect the monitor analysis port with the agt port
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage
```

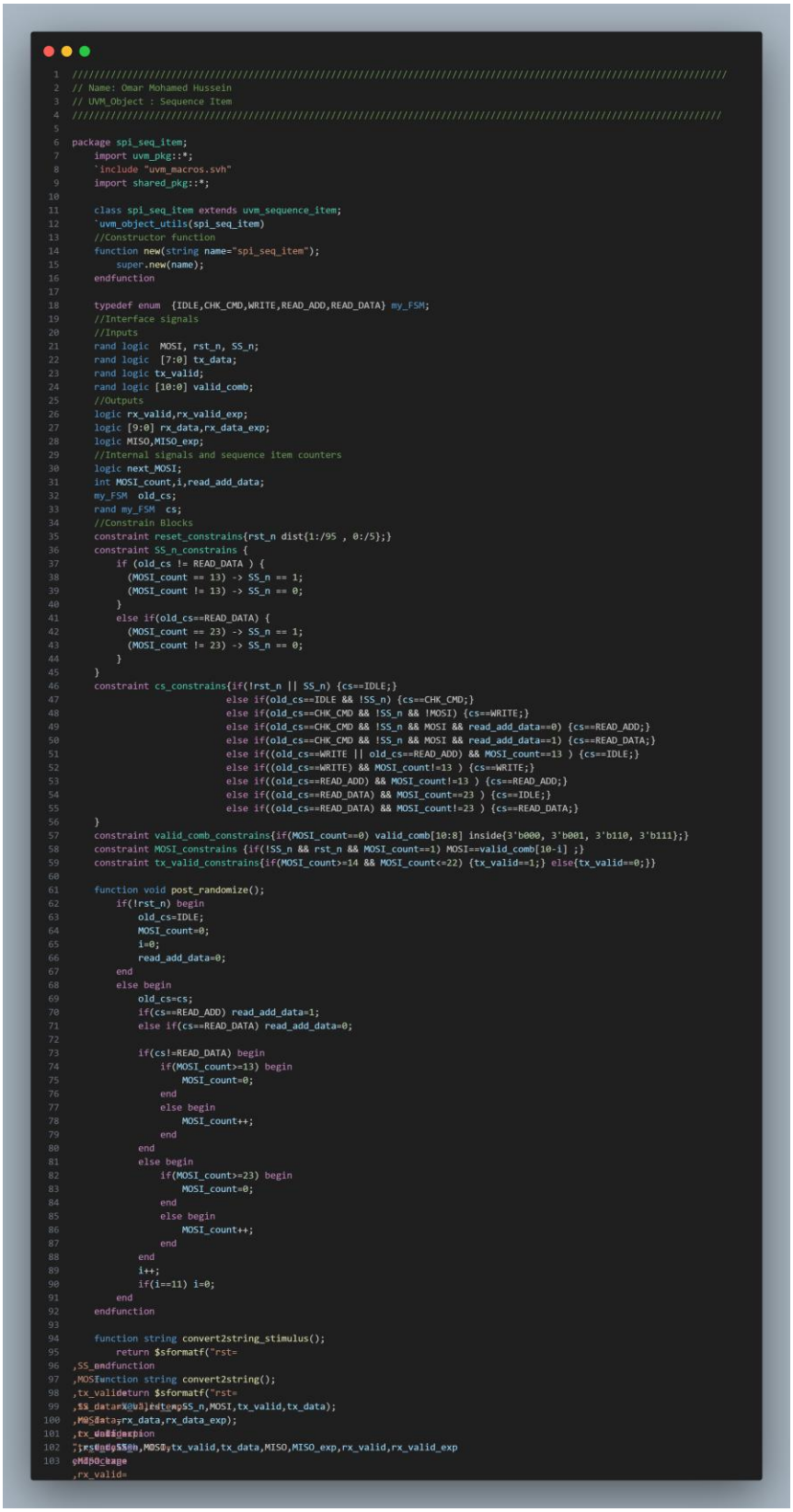
# 4.14.Wrapper Scoreboard

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

    class wrapper_scoreboard extends uvm_scoreboard ;
        `uvm_component_utils(wrapper_scoreboard)

        //Creating sequence item to get the data from fifo
        wrapper_seq_item seq_item;
        //Creating analysis export to take values from the fifo
        uvm_analysis_export #(wrapper_seq_item) sb_export;
        //Creating analysis fifo to take values from the agt port
        uvm_tlm_analysis_fifo #(wrapper_seq_item) sb_fifo;

        int correct_count,error_count;

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

        //Build phase : create the export analysis
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb_export=new("sb_export",this);
            sb_fifo=new("sb_fifo",this);
        endfunction

        //Connect phase : connect the eb export with the fifo export
        function void connect_phase (uvm_phase phase);
            super.connect_phase(phase);
            sb_export.connect(sb_fifo.analysis_export);
        endfunction

        //run Task : Compare the output with the Golden model
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(seq_item);
                if(seq_item.MISO!=seq_item.MISO_exp) begin
                    `uvm_error("scoreboard run phase",$sformatf("Design doesn't match the Golden model => %s",seq_item.convert2string))
                    error_count++;
                end
                else begin
                    correct_count++;
                end
            end
        endtask

        //Report Phase: report the testbench with how many failed and correct trials
        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("Scoreboard Report Phase",$sformatf("Testbench Completed , Correct count=
, error count=
",correct_count,error_count),UVM_LOW)
        endfunction
endpackage
```

# 4.15.Wrapper Environment

```
package wrapper_env_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import wrapper_driver_pkg::*;
import wrapper_scoreboard_pkg::*;
import wrapper_agent_pkg::*;
import shared_pkg::*;
//import wrapper_cover_pkg::*;
    class wrapper_env extends uvm_env;
        `uvm_component_utils(wrapper_env)
        wrapper_scoreboard sb;
        wrapper_agent agt;
        // wrapper_cover cv;

        function new(string name="wrapper_env",uvm_component parent=null);
            super.new(name,parent);
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb=wrapper_scoreboard::type_id::create("sb",this);
            agt=wrapper_agent::type_id::create("agt",this);
            //cv=wrapper_cover::type_id::create("wrapper_cover",this);
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            agt.agt_ap.connect(sb.sb_export);
            //agt.agt_ap.connect(cv.cover_ap);
        endfunction
    endclass
endpackage
```

# 4.16.SPI Sequence Item



```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Object : Sequence Item
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

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

    class spi_seq_item extends uvm_sequence_item;
    `uvm_object_utils(spi_seq_item)
    //Constructor function
    function new(string name="spi_seq_item");
        super.new(name);
    endfunction

    typedef enum  {IDLE,CHK_CMD,WRITE,READ_ADD,READ_DATA} my_FSM;
    //Interface signals
    //Inputs
    rand logic  MOSI, rst_n, SS_n;
    rand logic  [7:0] tx_data;
    rand logic tx_valid;
    rand logic [10:0] valid_comb;
    //Outputs
    logic rx_valid,rx_valid_exp;
    logic [9:0] rx_data,rx_data_exp;
    logic MISO,MISO_exp;
    //Internal signals and sequence item counters
    logic next_MOSI;
    int MOSI_count,i,read_add_data;
    my_FSM  old_cs;
    rand my_FSM  cs;
    //Constrain Blocks
    constraint reset_constrains{rst_n dist{1:/95 , 0:/5};}
    constraint SS_n_constrains {
        if (old_cs != READ_DATA ) {
          (MOSI_count == 13) -> SS_n == 1;
          (MOSI_count != 13) -> SS_n == 0;
        }
        else if(old_cs==READ_DATA) {
          (MOSI_count == 23) -> SS_n == 1;
          (MOSI_count != 23) -> SS_n == 0;
        }
    }
    constraint cs_constrains{if(!rst_n || SS_n) {cs==IDLE;}
                             else if(old_cs==IDLE && !SS_n) {cs==CHK_CMD;}
                             else if(old_cs==CHK_CMD && !SS_n && !MOSI) {cs==WRITE;}
                             else if(old_cs==CHK_CMD && !SS_n && MOSI && read_add_data==0) {cs==READ_ADD;}
                             else if(old_cs==CHK_CMD && !SS_n && MOSI && read_add_data==1) {cs==READ_DATA;}
                             else if((old_cs==WRITE || old_cs==READ_ADD) && MOSI_count==13 ) {cs==IDLE;}
                             else if((old_cs==WRITE) && MOSI_count!=13 ) {cs==WRITE;}
                             else if((old_cs==READ_ADD) && MOSI_count!=13 ) {cs==READ_ADD;}
                             else if((old_cs==READ_DATA) && MOSI_count==23 ) {cs==IDLE;}
                             else if((old_cs==READ_DATA) && MOSI_count!=23 ) {cs==READ_DATA;}
    }
    constraint valid_comb_constrains{if(MOSI_count==0) valid_comb[10:8] inside{3'b000, 3'b001, 3'b110, 3'b111};}
    constraint MOSI_constrains {if(!SS_n && rst_n && MOSI_count==1) MOSI==valid_comb[10-i] ;}
    constraint tx_valid_constrains{if(MOSI_count>=14 && MOSI_count<=22) {tx_valid==1;} else{tx_valid==0;}}

    function void post_randomize();
        if(!rst_n) begin
            old_cs=IDLE;
            MOSI_count=0;
            i=0;
            read_add_data=0;
        end
        else begin
            old_cs=cs;
            if(cs==READ_ADD) read_add_data=1;
            else if(cs==READ_DATA) read_add_data=0;

            if(cs!=READ_DATA) begin
                if(MOSI_count>=13) begin
                    MOSI_count=0;
                end
                else begin
                    MOSI_count++;
                end
            end
            else begin
                if(MOSI_count>=23) begin
                    MOSI_count=0;
                end
                else begin
                    MOSI_count++;
                end
            end
            i++;
            if(i==11) i=0;
        end
    endfunction

    function string convert2string_stimulus();
        return $sformatf("rst=
,SS_endfunction
,MOSIunction string convert2string();
,tx_validreturn $sformatf("rst=
,$s_data=%0d,rst_n=%0d,SS_n,MOSI,tx_valid,tx_data);
,MOSIata,rx_data,rx_data_exp);
,tx_valid=%0d
,rst_n=%0d,SS_n,MOSI,tx_valid,tx_data,MISO,MISO_exp,rx_valid,rx_valid_exp
,MISO=%0d
,rx_valid=
```

# 4.17.SPI Configuration Object

```
package spi_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class spi_config_obj extends uvm_object;
        `uvm_object_utils(spi_config_obj)

        virtual spi_if spi_config_vif;
        uvm_active_passive_enum is_active;
        function new(string name="alsu_config_obj");
            super.new(name);
        endfunction
    endclass
endpackage
```

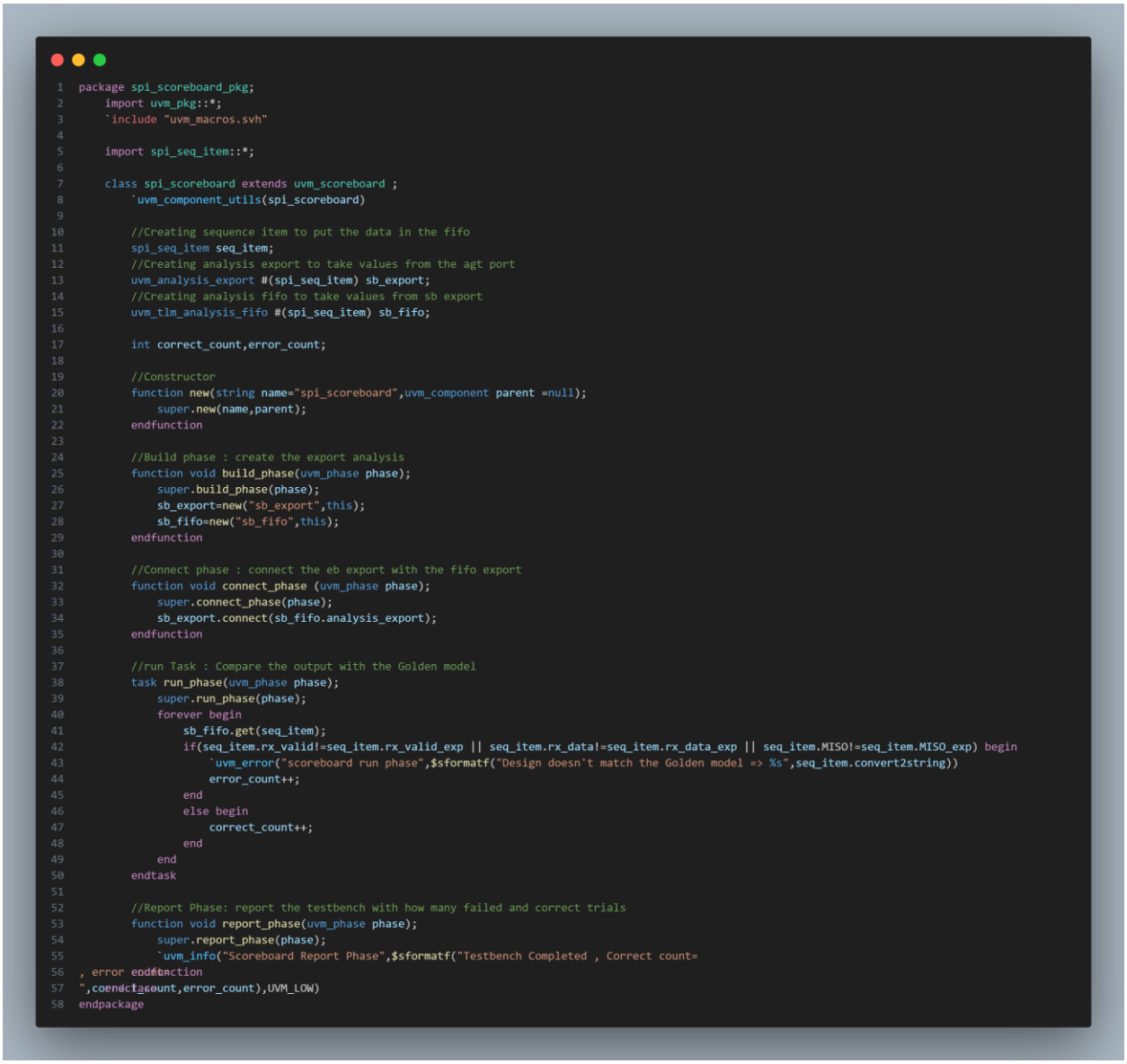
# 4.18.SPI Monitor

```
package spi_monitor_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import spi_seq_item::*;
    import shared_pkg::*;
    class spi_monitor extends uvm_monitor ;
        `uvm_component_utils(spi_monitor)

        //Creating Virtual interface to get the data from the interface
        virtual spi_if s_if;
        //Creating monitor sequence item to pass the interface to it to be then sent to the analysis components
        spi_seq_item mon_seq_item;
        //Creating monitor analysis port to broadcast the values to the analysis components
        uvm_analysis_port #(spi_seq_item) mon_ap;

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

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

        //Run Phase: samples the data at the negative edge and writes the analysis port with the monitor sequence item
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                mon_seq_item=spi_seq_item::type_id::create("mon_seq_item");
                @(negedge s_if.clk);

                //Drive the monitor sequence item with the interface
                mon_seq_item.rst_n=s_if.rst_n;
                mon_seq_item.SS_n=s_if.SS_n;
                mon_seq_item.MOSI=s_if.MOSI;
                mon_seq_item.MISO=s_if.MISO;
                mon_seq_item.MISO_exp=s_if.MISO_exp;
                mon_seq_item.tx_valid=s_if.tx_valid;
                mon_seq_item.tx_data=s_if.tx_data;
                mon_seq_item.rx_valid=s_if.rx_valid;
                mon_seq_item.rx_valid_exp=s_if.rx_valid_exp;
                mon_seq_item.rx_data=s_if.rx_data;
                mon_seq_item.rx_data_exp=s_if.rx_data_exp;

                //Broadcast the monitor sequence (interfance) to the analysis port
                mon_ap.write(mon_seq_item);
                `uvm_info("Monitor run phase",mon_seq_item.convert2string(),UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

# 4.19.SPI Agent

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//Name: Omar Mohamed Hussein
//UVM_Component:Agent Package
////////////////////////////////////////////////////////////////////////////////////////////////////
package spi_agent_pkg;

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

    import spi_seq_item::*;
    import spi_sqr_pkg::*;
    import spi_driver_pkg::*;
    import spi_monitor_pkg::*;
    import spi_config_pkg::*;

    class spi_agent extends uvm_agent;
        `uvm_component_utils(spi_agent)

        //creating the sequencer,driver,monitor, configuration object to connect the interface,agent port
        spi_sqr sqr;
        spi_driver drv;
        spi_monitor mon;
        spi_config_obj config_obj;
        uvm_analysis_port #(spi_seq_item) agt_ap;

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

        //Build Phase
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            //Connect the agent configuration object with the configuration object in the uvm config data base
            if(!uvm_config_db #(spi_config_obj)::get(this,"","CFG_SPI",config_obj)) begin
                `uvm_fatal("agent build phase","unable to get the configuration object to the agent")
            end

            ////////////////////////////////// For Passive Agents ////////////////////////////////////////
            if(config_obj.is_active==UVM_ACTIVE) begin
                sqr=spi_sqr::type_id::create("sqr",this);
                drv=spi_driver::type_id::create("drv",this);
            end
            ////////////////////////////////////////////////////////////////////////////////////////////

            //Creating the sequencer,driver,monitor with the create() and the agt port with new()
            mon=spi_monitor::type_id::create("mon",this);
            agt_ap=new("agt_ap",this);
        endfunction

        //connect phase
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);

            ////////////////////////////////////For Passive Agents ////////////////////////////////////////
            if(config_obj.is_active==UVM_ACTIVE) begin
                drv.spi_driver_vif=config_obj.spi_config_vif;
                drv.seq_item_port.connect(sqr.seq_item_export);
            end
            ////////////////////////////////////////////////////////////////////////////////////////////
            mon.s_if=config_obj.spi_config_vif;
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage
```

# 4.20.SPI Scoreboard



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

    import spi_seq_item::*;

    class spi_scoreboard extends uvm_scoreboard ;
        `uvm_component_utils(spi_scoreboard)

        //Creating sequence item to put the data in the fifo
        spi_seq_item seq_item;
        //Creating analysis export to take values from the agt port
        uvm_analysis_export #(spi_seq_item) sb_export;
        //Creating analysis fifo to take values from sb export
        uvm_tlm_analysis_fifo #(spi_seq_item) sb_fifo;

        int correct_count,error_count;

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

        //Build phase : create the export analysis
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb_export=new("sb_export",this);
            sb_fifo=new("sb_fifo",this);
        endfunction

        //Connect phase : connect the eb export with the fifo export
        function void connect_phase (uvm_phase phase);
            super.connect_phase(phase);
            sb_export.connect(sb_fifo.analysis_export);
        endfunction

        //run Task : Compare the output with the Golden model
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(seq_item);
                if(seq_item.rx_valid!=seq_item.rx_valid_exp || seq_item.rx_data!=seq_item.rx_data_exp || seq_item.MISO!=seq_item.MISO_exp) begin
                    `uvm_error("scoreboard run phase",$sformatf("Design doesn't match the Golden model => %s",seq_item.convert2string))
                    error_count++;
                end
                else begin
                    correct_count++;
                end
            end
        endtask

        //Report Phase: report the testbench with how many failed and correct trials
        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("Scoreboard Report Phase",$sformatf("Testbench Completed , Correct count=
, error count=
",correct_count,error_count),UVM_LOW)
        endfunction
    endpackage
```

# 4.21.SPI Cover Collector

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

    class spi_cover extends uvm_component;
        `uvm_component_utils(spi_cover)
        uvm_analysis_export #(spi_seq_item) cover_ap;
        uvm_tlm_analysis_fifo #(spi_seq_item) cover_fifo;
        spi_seq_item seq_item;


    //Cover Groups
        covergroup Cov_gp(ref spi_seq_item seq_item);
        //Normal Coverpoints
        rx_cp: coverpoint seq_item.rx_data[9:8]{
            bins all_values[] = {[0:3]};
            bins WRITE_ADD =(2'b00=>2'b00=>2'b00) ;
            bins WRITE_DATA=(2'b00=>2'b00=>2'b01) ;
            bins READ_ADD  =(2'b00=>2'b10=>2'b10) ;
            bins READ_DATA =(2'b00=>2'b10=>2'b11) ;
        }
        SS_n_cp: coverpoint seq_item.SS_n{
            bins read_transition=(1=> 0[*23] =>1);
            bins not_read_transition=(1=> 0[*13] =>1);
        }
        MOSI_cp:coverpoint seq_item.MOSI{
            bins b000=(0=>0=>0);
            bins b001=(0=>0=>1);
            bins b111=(1=>1=>1);
            bins b110=(1=>1=>0);
        }
        cross_cv:cross MOSI_cp,SS_n_cp{
            ignore_bins write_cv_000 =binsof(MOSI_cp.b000) && binsof(SS_n_cp.read_trans
ition);
            ignore_bins write_cv_001 =binsof(MOSI_cp.b001) && binsof(SS_n_cp.read_trans
ition);
            ignore_bins read_cv_111 =binsof(MOSI_cp.b111) && binsof(SS_n_cp.not_read_tr
ansition);
            ignore_bins read_cv_110 =binsof(MOSI_cp.b110) && binsof(SS_n_cp.not_read_tr
ansition);
        }
        //Cross coverage
        endgroup

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

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

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cover_ap.connect(cover_fifo.analysis_export);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                seq_item=spi_seq_item::type_id::create("seq_item");
                cover_fifo.get(seq_item);
                Cov_gp.sample();
            end
        endtask

    endclass
endpackage
```

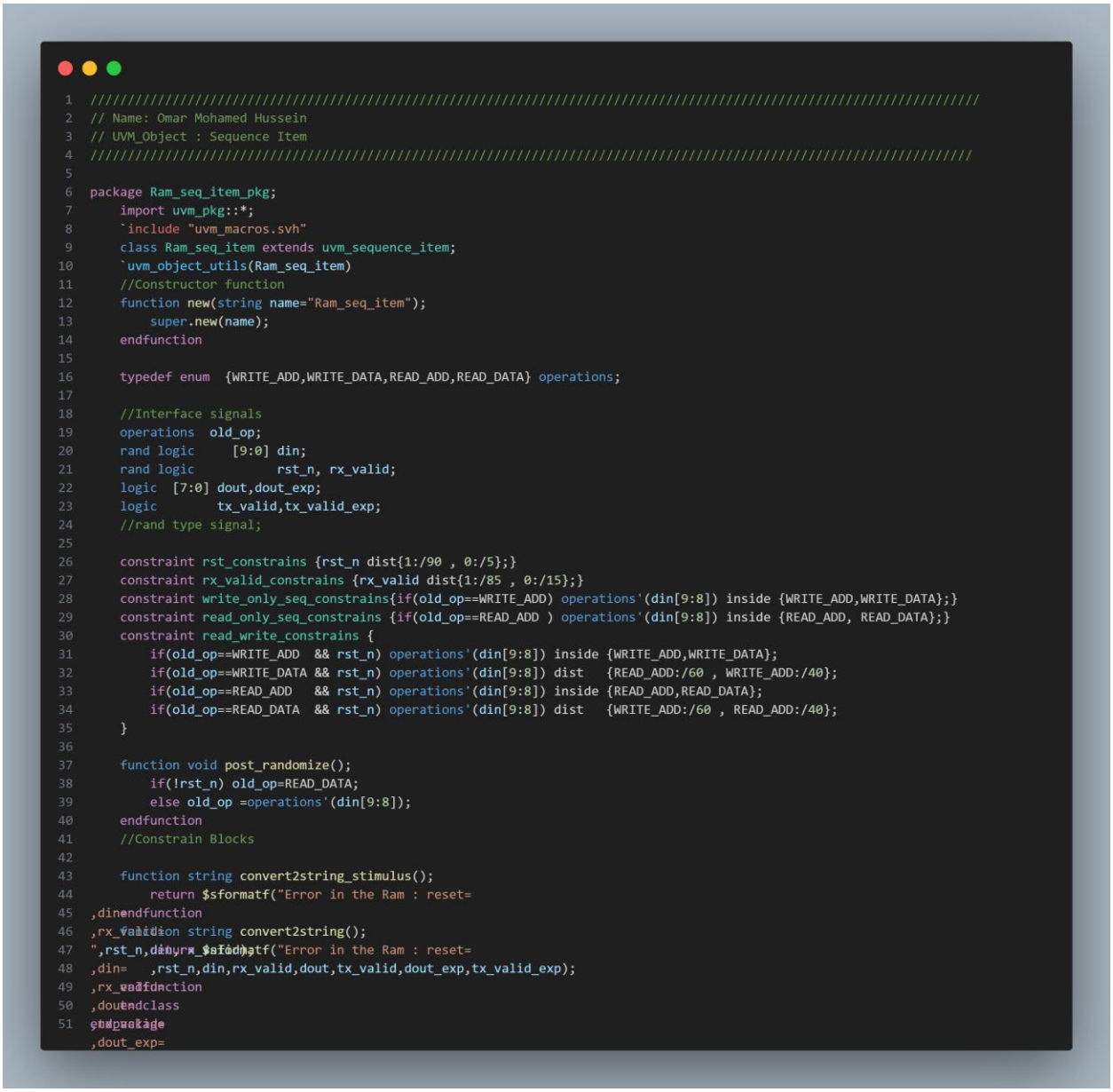
# 4.22.SPI Environment

```
package spi_env_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import spi_driver_pkg::*;
import spi_scoreboard_pkg::*;
import spi_agent_pkg::*;
import shared_pkg::*;
import spi_cover_pkg::*;
    class spi_env extends uvm_env;
        `uvm_component_utils(spi_env)
        spi_scoreboard sb;
        spi_agent agt;
        spi_cover cv;

        function new(string name="spi_env",uvm_component parent=null);
            super.new(name,parent);
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb=spi_scoreboard::type_id::create("sb",this);
            agt=spi_agent::type_id::create("agt",this);
            cv=spi_cover::type_id::create("spi_cover",this);
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            agt.agt_ap.connect(sb.sb_export);
            agt.agt_ap.connect(cv.cover_ap);
        endfunction
    endclass
endpackage
```

# 4.23.SPI Assertions

```
module spi_sva(MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
input MOSI,MISO,SS_n,clk,rst_n,rx_valid,tx_valid;
input [9:0]rx_data;
input [7:0]tx_data;

    sequence valid_comb_000;
    ($fell(SS_n) ##1 !MOSI ##1 !MOSI ##1 !MOSI);
    endsequence
    sequence valid_comb_001;
    ($fell(SS_n) ##1  !MOSI ##1 !MOSI ##1 MOSI);
    endsequence
    sequence valid_comb_110;
    ($fell(SS_n) ##1 MOSI ##1 MOSI ##1 !MOSI);
    endsequence
    sequence valid_comb_111;
    ($fell(SS_n) ##1 MOSI ##1 MOSI ##1 MOSI);
    endsequence
    assert property(@(posedge clk) disable iff(!rst_n) valid_comb_000 |-> ##10 (rx_valid ##1 SS_n[->1]));
    assert property(@(posedge clk) disable iff(!rst_n) valid_comb_001 |-> ##10 (rx_valid ##1 SS_n[->1]));
    assert property(@(posedge clk) disable iff(!rst_n) valid_comb_110 |-> ##10 (rx_valid ##1 SS_n[->1]));
    assert property(@(posedge clk) disable iff(!rst_n) valid_comb_111 |-> ##10 (rx_valid ##1 SS_n[->1]));
    assert property(@(posedge clk) !rst_n |=> !MISO && !rx_valid && rx_data==0);
    cover  property(@(posedge clk) disable iff(!rst_n) valid_comb_000 |-> ##10 (rx_valid  ##1 SS_n[->1]));
    cover  property(@(posedge clk) disable iff(!rst_n) valid_comb_001 |-> ##10 (rx_valid  ##1 SS_n[->1]));
    cover  property(@(posedge clk) disable iff(!rst_n) valid_comb_110 |-> ##10 (rx_valid  ##1 SS_n[->1]));
    cover  property(@(posedge clk) disable iff(!rst_n) valid_comb_111 |-> ##10 (rx_valid  ##1 SS_n[->1]));
    cover  property(@(posedge clk) !rst_n |=> !MISO && !rx_valid && rx_data==0);
endmodule
```

# 4.24.RAM Sequence Item



```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Name: Omar Mohamed Hussein
// UVM_Object : Sequence Item
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

package Ram_seq_item_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class Ram_seq_item extends uvm_sequence_item;
    `uvm_object_utils(Ram_seq_item)
    //Constructor function
    function new(string name="Ram_seq_item");
        super.new(name);
    endfunction

    typedef enum  {WRITE_ADD,WRITE_DATA,READ_ADD,READ_DATA} operations;

    //Interface signals
    operations  old_op;
    rand logic     [9:0] din;
    rand logic           rst_n, rx_valid;
    logic  [7:0] dout,dout_exp;
    logic        tx_valid,tx_valid_exp;
    //rand type signal;

    constraint rst_constrains {rst_n dist{1:/90 , 0:/5};}
    constraint rx_valid_constrains {rx_valid dist{1:/85 , 0:/15};}
    constraint write_only_seq_constrains{if(old_op==WRITE_ADD) operations'(din[9:8]) inside {WRITE_ADD,WRITE_DATA};}
    constraint read_only_seq_constrains {if(old_op==READ_ADD ) operations'(din[9:8]) inside {READ_ADD, READ_DATA};}
    constraint read_write_constrains {
        if(old_op==WRITE_ADD  && rst_n) operations'(din[9:8]) inside {WRITE_ADD,WRITE_DATA};
        if(old_op==WRITE_DATA && rst_n) operations'(din[9:8]) dist   {READ_ADD:/60 , WRITE_ADD:/40};
        if(old_op==READ_ADD   && rst_n) operations'(din[9:8]) inside {READ_ADD,READ_DATA};
        if(old_op==READ_DATA  && rst_n) operations'(din[9:8]) dist   {WRITE_ADD:/60 , READ_ADD:/40};
    }

    function void post_randomize();
        if(!rst_n) old_op=READ_DATA;
        else old_op =operations'(din[9:8]);
    endfunction
    //Constrain Blocks

    function string convert2string_stimulus();
        return $sformatf("Error in the Ram : reset=
,dinendfunction
,rx_validion string convert2string();
",rst_n,dinrx_sformatf("Error in the Ram : reset=
,din=   ,rst_n,din,rx_valid,dout,tx_valid,dout_exp,tx_valid_exp);
,rx_endfunction
,doutendclass
endpackage
,dout_exp=
```

# 4.25.RAM Configuration Object

```
package Ram_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class Ram_config_obj extends uvm_object;
        `uvm_object_utils(Ram_config_obj)

        virtual Ram_if Ram_config_vif;
        uvm_active_passive_enum is_active;
        function new(string name="Ram_config_obj");
            super.new(name);
        endfunction

    endclass
endpackage
```

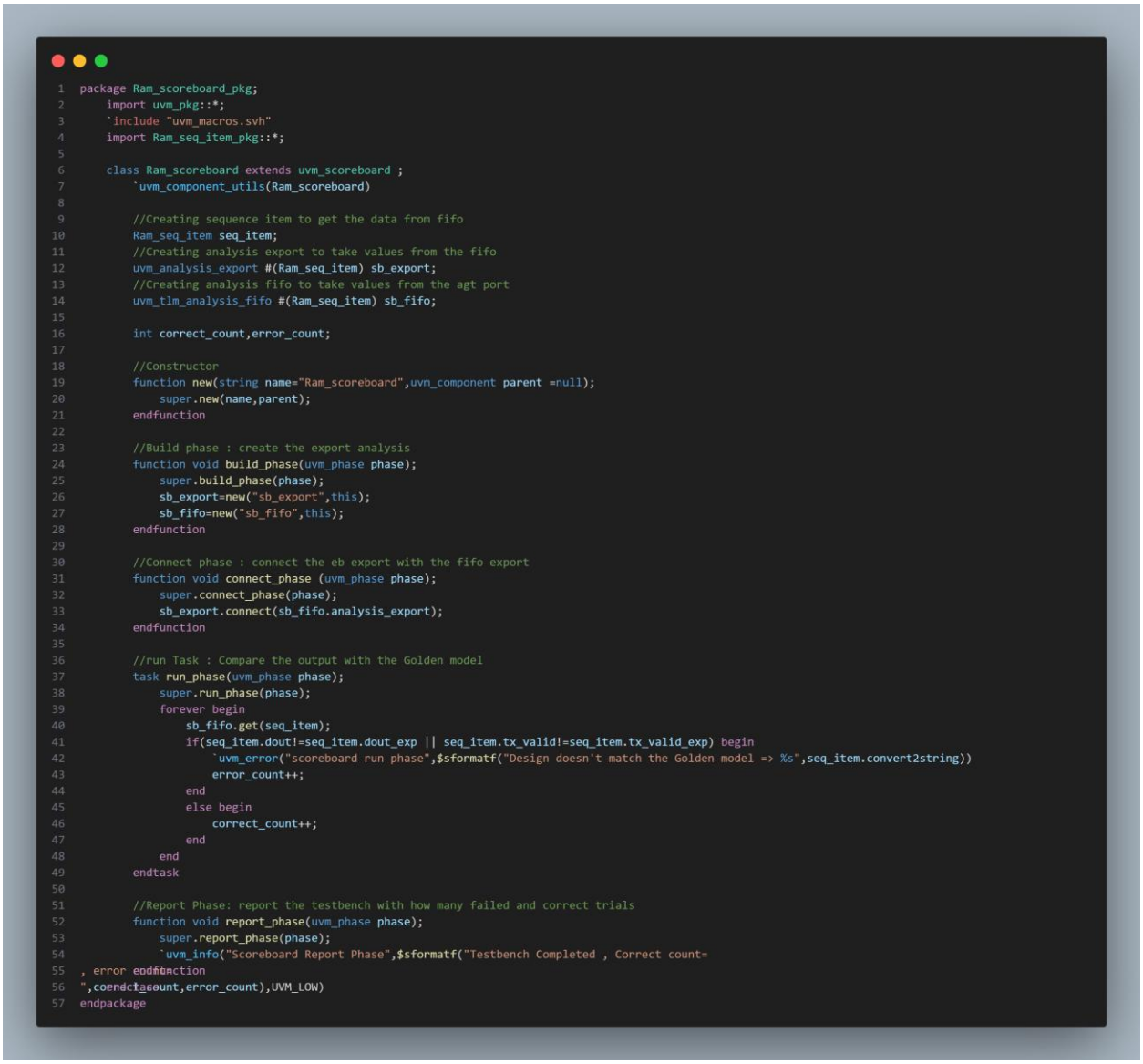
# 4.26.RAM Monitor

```
package Ram_monitor_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import Ram_seq_item_pkg::*;
    class Ram_monitor extends uvm_monitor ;
        `uvm_component_utils(Ram_monitor)

        //Creating Virtual interface to get the data from the interface
        virtual Ram_if r_if;
        //Creating monitor sequence item to pass the interface to it to be then sent to the analysis components
        Ram_seq_item mon_seq_item;
        //Creating monitor analysis port to broadcast the values to the analysis components
        uvm_analysis_port #(Ram_seq_item) mon_ap;

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

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

        //Run Phase: samples the data at the negative edge and writes the analysis port with the monitor sequence item
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                mon_seq_item=Ram_seq_item::type_id::create("mon_seq_item");
                @(negedge r_if.clk);

                //Drive the monitor sequence item with the interface
                mon_seq_item.din         =r_if.din;
                mon_seq_item.rst_n       =r_if.rst_n;
                mon_seq_item.rx_valid    =r_if.rx_valid;
                mon_seq_item.dout        =r_if.dout;
                mon_seq_item.tx_valid    =r_if.tx_valid;
                mon_seq_item.tx_valid_exp=r_if.tx_valid_exp;
                mon_seq_item.dout_exp    =r_if.dout_exp;

                //Broadcast the monitor sequence (interfance) to the analysis port
                mon_ap.write(mon_seq_item);
                `uvm_info("Monitor run phase",mon_seq_item.convert2string(),UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

# 4.27.RAM Agent

```
////////////////////////////////////////////////////////////////////////////////////////////////////////
//Name: Omar Mohamed Hussein
//UVM_Component:Agent Package
////////////////////////////////////////////////////////////////////////////////////////////////////////
package Ram_agent_pkg;

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

    import Ram_seq_item_pkg::*;
    import Ram_sqr_pkg::*;
    import Ram_driver_pkg::*;
    import Ram_monitor_pkg::*;
    import Ram_config_pkg::*;

    class Ram_agent extends uvm_agent;
        `uvm_component_utils(Ram_agent)

        //creating the sequencer,driver,monitor, configuration object to connect the interface,agent port
        Ram_sqr sqr;
        Ram_driver drv;
        Ram_monitor mon;
        Ram_config_obj config_obj;
        uvm_analysis_port #(Ram_seq_item) agt_ap;

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

        //Build Phase
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            //Connect the agent configuration object with the configuration object in the uvm config data base
            if(!uvm_config_db #(Ram_config_obj)::get(this,"","CFG_RAM",config_obj)) begin
                `uvm_fatal("agent build phase","unable to get the configuration object to the agent")
            end

            ////////////////// For Passive Agents ////////////////////////////////////////////////////////
            if(config_obj.is_active==UVM_ACTIVE) begin
                sqr=Ram_sqr::type_id::create("sqr",this);
                drv=Ram_driver::type_id::create("drv",this);
            end
            ////////////////////////////////////////////////////////////////////////////////////////////
            mon=Ram_monitor::type_id::create("mon",this);
            agt_ap=new("agt_ap",this);
        endfunction

        //connect phase
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            mon.r_if=config_obj.Ram_config_vif;
            //Connect the driver port with the sequencer export
            if(config_obj.is_active==UVM_ACTIVE) begin
                drv.seq_item_port.connect(sqr.seq_item_export);
                drv.Ram_driver_vif=config_obj.Ram_config_vif;
            end
            //Connect the monitor analysis port with the agt port
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage
```

# 4.28.RAM Scoreboard



```
package Ram_scoreboard_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import Ram_seq_item_pkg::*;

    class Ram_scoreboard extends uvm_scoreboard ;
        `uvm_component_utils(Ram_scoreboard)

        //Creating sequence item to get the data from fifo
        Ram_seq_item seq_item;
        //Creating analysis export to take values from the fifo
        uvm_analysis_export #(Ram_seq_item) sb_export;
        //Creating analysis fifo to take values from the agt port
        uvm_tlm_analysis_fifo #(Ram_seq_item) sb_fifo;

        int correct_count,error_count;

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

        //Build phase : create the export analysis
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb_export=new("sb_export",this);
            sb_fifo=new("sb_fifo",this);
        endfunction

        //Connect phase : connect the eb export with the fifo export
        function void connect_phase (uvm_phase phase);
            super.connect_phase(phase);
            sb_export.connect(sb_fifo.analysis_export);
        endfunction

        //run Task : Compare the output with the Golden model
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(seq_item);
                if(seq_item.dout!=seq_item.dout_exp || seq_item.tx_valid!=seq_item.tx_valid_exp) begin
                    `uvm_error("scoreboard run phase",$sformatf("Design doesn't match the Golden model => %s",seq_item.convert2string))
                    error_count++;
                end
                else begin
                    correct_count++;
                end
            end
        endtask

        //Report Phase: report the testbench with how many failed and correct trials
        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("Scoreboard Report Phase",$sformatf("Testbench Completed , Correct count=
, error count
",correct_count,error_count),UVM_LOW)
    endfunction
endpackage
```

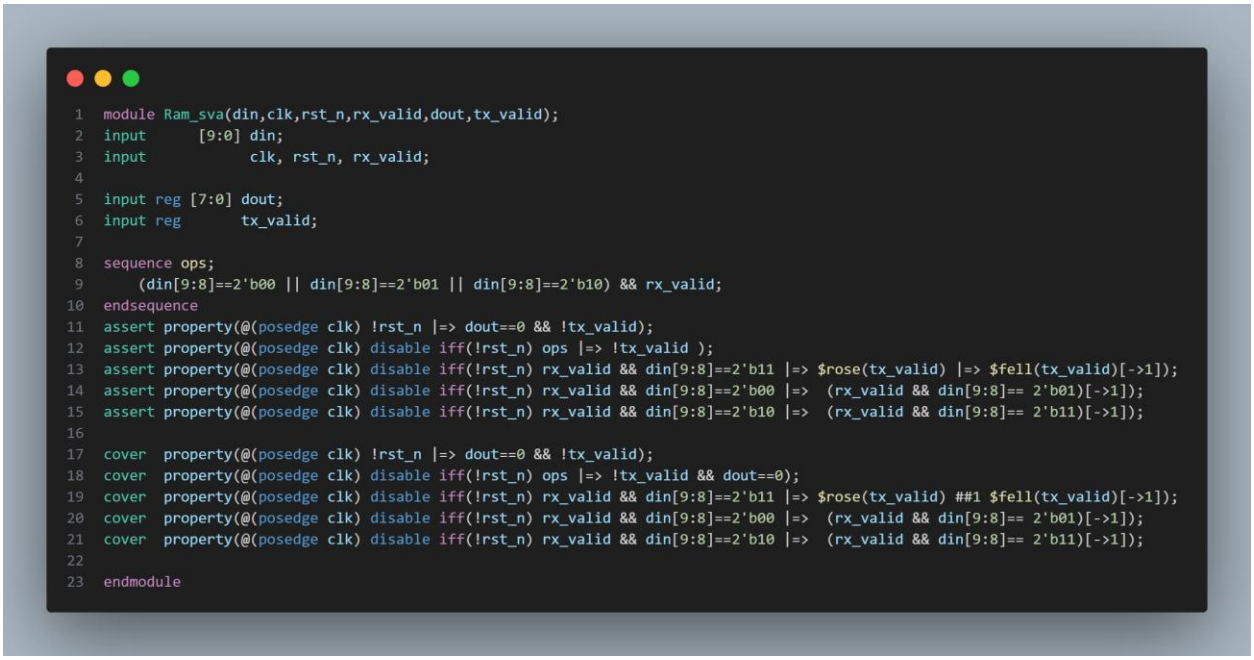
# 4.29.RAM Cover Collector

```
package Ram_cover_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import Ram_seq_item_pkg::*;

    class Ram_cover extends uvm_component;
        `uvm_component_utils(Ram_cover)
        uvm_analysis_export #(Ram_seq_item) cover_ap;
        uvm_tlm_analysis_fifo #(Ram_seq_item) cover_fifo;
        Ram_seq_item seq_item;


    //Cover Groups
        covergroup Cov_gp(ref Ram_seq_item seq_item);
        //Normal Coverpoints
        operations_cp: coverpoint seq_item.din[9:8] {
            bins all_vals[]={[0:3]};
            // bins Write_address_data=(2'b00=>2'b01);
            // bins Read_address_data =(2'b10=>2'b11);                    Impossible to hit in the Wrapper module
            // bins all_operations =(2'b00=>2'b01[*13]=>2'b10[*13]=>2'b11); Impossible to hit in the Wrapper module
        }
        rx_valid_cp:coverpoint seq_item.rx_valid{
            bins high={1};
            bins low={0};
        }
        //Cross coverage
        rx_valid_ops:cross operations_cp,rx_valid_cp {
            bins allvals_rx_valid = binsof(operations_cp.all_vals) && binsof(rx_valid_cp.high);
            bins read_data_tx_valid = binsof(operations_cp.all_vals) intersect {2'b11} && binsof(rx_valid_cp.high);
        }
        endgroup
        function new(string name="Ram_cover",uvm_component parent=null);
            super.new(name,parent);
            Cov_gp=new(seq_item);
        endfunction

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

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cover_ap.connect(cover_fifo.analysis_export);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                seq_item=Ram_seq_item::type_id::create("seq_item");
                cover_fifo.get(seq_item);
                Cov_gp.sample();
            end
        endtask

    endclass
endpackage
```

# 4.30.RAM Environment

```
package Ram_env_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import Ram_driver_pkg::*;
import Ram_scoreboard_pkg::*;
import Ram_agent_pkg::*;
import Ram_cover_pkg::*;
    class Ram_env extends uvm_env;
        `uvm_component_utils(Ram_env)
        Ram_scoreboard sb;
        Ram_agent agt;
        Ram_cover cv;

        function new(string name="Ram_env",uvm_component parent=null);
            super.new(name,parent);
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb=Ram_scoreboard::type_id::create("sb",this);
            agt=Ram_agent::type_id::create("agt",this);
            cv=Ram_cover::type_id::create("cv",this);
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            agt.agt_ap.connect(sb.sb_export);
            agt.agt_ap.connect(cv.cover_ap);
        endfunction
    endclass
endpackage
```

# 4.31.RAM Assertions



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

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

sequence ops;
    (din[9:8]==2'b00 || din[9:8]==2'b01 || din[9:8]==2'b10) && rx_valid;
endsequence
assert property(@(posedge clk) !rst_n |=> dout==0 && !tx_valid);
assert property(@(posedge clk) disable iff(!rst_n) ops |=> !tx_valid );
assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b11 |=> $rose(tx_valid) |=> $fell(tx_valid)[->1]);
assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b00 |=>  (rx_valid && din[9:8]== 2'b01)[->1]);
assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b10 |=>  (rx_valid && din[9:8]== 2'b11)[->1]);

cover  property(@(posedge clk) !rst_n |=> dout==0 && !tx_valid);
cover  property(@(posedge clk) disable iff(!rst_n) ops |=> !tx_valid && dout==0);
cover  property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b11 |=> $rose(tx_valid) ##1 $fell(tx_valid)[->1]);
cover  property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b00 |=>  (rx_valid && din[9:8]== 2'b01)[->1]);
cover  property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b10 |=>  (rx_valid && din[9:8]== 2'b11)[->1]);

endmodule
```

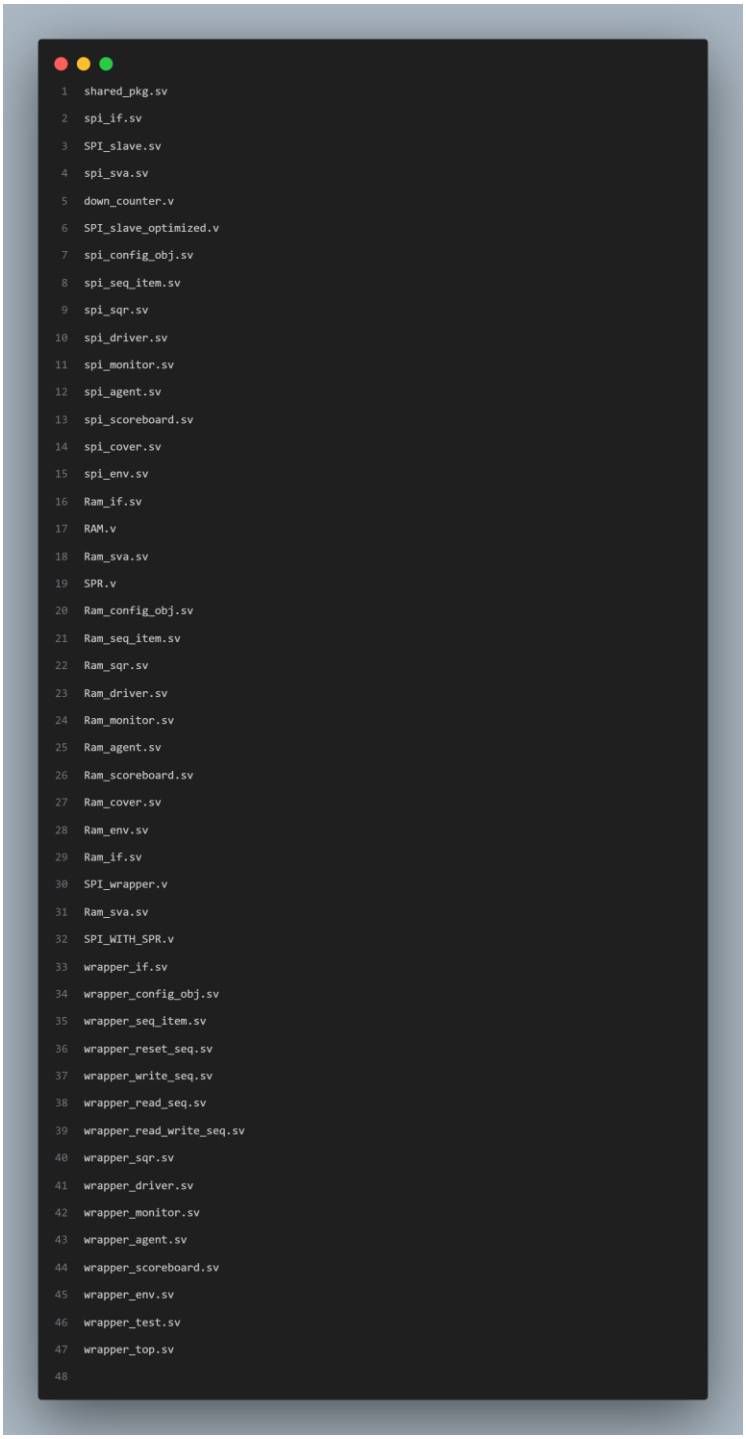
# 4.32.Wrapper Test

```
package wrapper_test_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import wrapper_env_pkg::*;
import Ram_env_pkg::*;
import spi_env_pkg::*;
import wrapper_config_pkg::*;
import spi_config_pkg::*;
import Ram_config_pkg::*;
import wrapper_reset_seq_pkg::*;
import wrapper_write_seq_pkg::*;
import wrapper_read_write_seq_pkg::*;
import wrapper_read_seq_pkg::*;
    class wrapper_test extends uvm_test;
        `uvm_component_utils(wrapper_test)

        wrapper_env w_env;
        spi_env s_env;
        Ram_env r_env;
        wrapper_config_obj wrapper_config_obj_test;
        spi_config_obj spi_config_obj_test;
        Ram_config_obj Ram_config_obj_test;
        wrapper_reset_seq reset_seq;
        wrapper_write_seq write_seq;
        wrapper_read_seq read_seq;
        wrapper_read_write_seq write_read_seq;

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            w_env=wrapper_env::type_id::create("w_env",this);
            r_env=Ram_env::type_id::create("r_env",this);
            s_env=spi_env::type_id::create("s_env",this);
            wrapper_config_obj_test=wrapper_config_obj::type_id::create("wrapper_config_obj_test");
            Ram_config_obj_test=Ram_config_obj::type_id::create("Ram_config_obj_test");
            spi_config_obj_test=spi_config_obj::type_id::create("spi_config_obj_test");
            reset_seq=wrapper_reset_seq::type_id::create("reset_seq");
            write_seq=wrapper_write_seq::type_id::create("write_seq");
            read_seq=wrapper_read_seq::type_id::create("read_seq");
            write_read_seq=wrapper_read_write_seq::type_id::create("write_read_seq");

            if(!uvm_config_db #(virtual wrapper_if)::get(this,"","WRAPPER",wrapper_config_obj_test.wrapper_config_vif))
                `uvm_fatal("build phase in test","unable to get interface from the database into the configuration object")
            uvm_config_db#(wrapper_config_obj)::set(this,"*","CFG_WRAPPER",wrapper_config_obj_test);

            if(!uvm_config_db #(virtual spi_if)::get(this,"","SPI",spi_config_obj_test.spi_config_vif))
                `uvm_fatal("build phase in test","unable to get interface from the database into the configuration object")
            uvm_config_db#(spi_config_obj)::set(this,"*","CFG_SPI",spi_config_obj_test);

            if(!uvm_config_db #(virtual Ram_if)::get(this,"","RAM",Ram_config_obj_test.Ram_config_vif))
                `uvm_fatal("build phase in test","unable to get interface from the database into the configuration object")
            uvm_config_db#(Ram_config_obj)::set(this,"*","CFG_RAM",Ram_config_obj_test);

            wrapper_config_obj_test.is_active=UVM_ACTIVE;
            spi_config_obj_test.is_active=UVM_PASSIVE;
            Ram_config_obj_test.is_active=UVM_PASSIVE;
        endfunction

        task  run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            `uvm_info("Test run phase","Reset Asserted",UVM_LOW)
            reset_seq.start(w_env.agt.sqr);
            `uvm_info("Test run phase","Reset Finished",UVM_LOW)

            `uvm_info("Test run phase","Write sequence Asserted",UVM_LOW)
            write_seq.start(w_env.agt.sqr);
            `uvm_info("Test run phase","Write sequence Finished",UVM_LOW)

            `uvm_info("Test run phase","Read sequence Asserted",UVM_LOW)
            read_seq.start(w_env.agt.sqr);
            `uvm_info("Test run phase","Read sequence Finished",UVM_LOW)

            `uvm_info("Test run phase","Write Read sequence Asserted",UVM_LOW)
            write_read_seq.start(w_env.agt.sqr);
            `uvm_info("Test run phase","Write Read sequence Finished",UVM_LOW)
            phase.drop_objection(this);
        endtask
    endclass
endpackage
```

# 4.33.Wrapper Top Module

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

module top();
  // Clock generation
  bit clk;
  initial begin
    clk=0;
    forever begin
      #10 clk=~clk;
    end
  end
  initial begin
    $readmemh("mem.dat",Golden.RAM.MEM);
    $readmemh("mem.dat",DUT.RAM_instance.MEM);
  end
  // Instantiate the interface ,Golden Model and DUT
  Ram_if r_if(clk);
  spi_if s_if(clk);
  wrapper_if w_if(clk);

  WRAPPER DUT(w_if.MOSI,w_if.MISO,w_if.SS_n,w_if.clk,w_if.rst_n);
  SPI_WITH_SPR #(256,8)Golden(w_if.clk,w_if.rst_n,w_if.MISO_exp,w_if.MOSI,w_if.SS_n);

  assign s_if.rx_data= DUT.rx_data_din;
  assign s_if.clk= DUT.clk;
  assign s_if.rst_n= DUT.rst_n;
  assign s_if.rx_valid= DUT.rx_valid;
  assign s_if.tx_data= DUT.tx_data_dout;
  assign s_if.tx_valid= DUT.tx_valid;
  assign s_if.rx_data_exp = Golden.rx_data;
  assign s_if.rx_valid_exp= Golden.rx_valid;
  assign s_if.MISO_exp= Golden.MISO;
  assign s_if.MISO= DUT.MISO;
  assign s_if.SS_n= DUT.SS_n;
  assign s_if.MOSI= DUT.MOSI;

  assign r_if.din= DUT.rx_data_din;
  assign r_if.clk= DUT.clk;
  assign r_if.rst_n= DUT.rst_n;
  assign r_if.rx_valid= DUT.rx_valid;
  assign r_if.dout= DUT.tx_data_dout;
  assign r_if.dout_exp= Golden.tx_data;
  assign r_if.tx_valid= DUT.tx_valid;
  assign r_if.tx_valid_exp= Golden.tx_valid;

  bind DUT.RAM_instance Ram_sva Ram_assertion (r_if.din,r_if.clk,r_if.rst_n,r_if.rx_valid,r_if.dout,r_if.tx_valid);
  bind DUT.SLAVE_instance spi_sva SPI_assertion (MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);

  // run test using run_test task
  initial begin
    uvm_config_db #(virtual Ram_if)::set(null,"uvm_test_top","RAM",r_if);
    uvm_config_db #(virtual spi_if)::set(null,"uvm_test_top","SPI",s_if);
    uvm_config_db #(virtual wrapper_if)::set(null,"uvm_test_top","WRAPPER",w_if);
    run_test("wrapper_test");
  end
endmodule
```

# 4.34.Source Files



# 4.35.Do File

```
vlib work
vlog +define+SIM -f src_files.list  +cover -covercells
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
add wave /top/s_if/*
add wave /top/r_if/*
add wave /top/w_if/*
add wave -position insertpoint  \
sim:/top/DUT/SLAVE_instance/cs
add wave -position insertpoint  \
sim:/top/DUT/SLAVE_instance/counter
add wave -position insertpoint  \
sim:/top/Golden/SPI/cs \
sim:/top/Golden/SPI/count \
sim:/top/Golden/SPI/count_10
run 0
coverage save top.ucdb -onexit
run -all
```

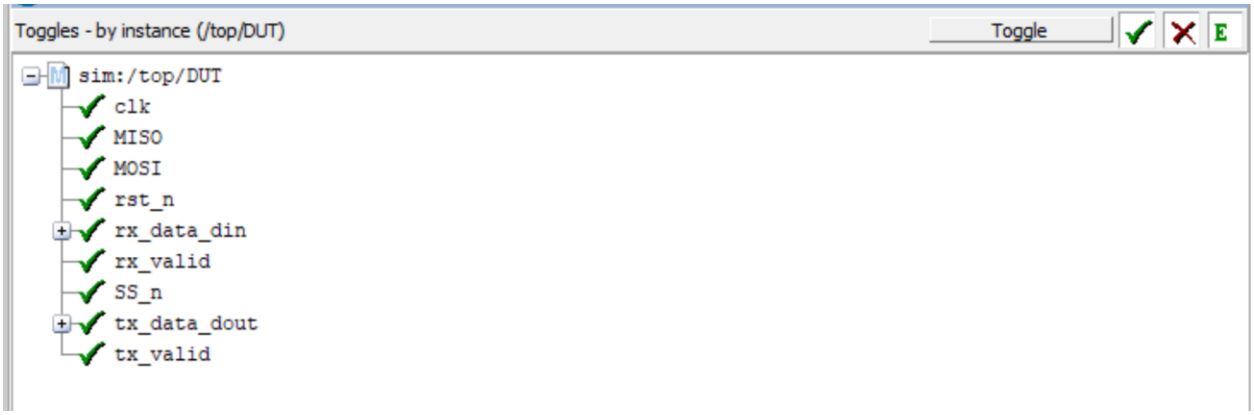
# 4.36.Wrapper Simulation Results

## 4.36.1.Questasim Transcript

```
UVM_INFO wrapper_test.sv(67) @ 20: uvm_test_top [Test run phase] Reset Finished
UVM_INFO wrapper_test.sv(69) @ 20: uvm_test_top [Test run phase] Write sequence Asserted
UVM_INFO wrapper_test.sv(71) @ 100020: uvm_test_top [Test run phase] Write sequence Finished
UVM_INFO wrapper_test.sv(73) @ 100020: uvm_test_top [Test run phase] Read sequence Asserted
UVM_INFO wrapper_test.sv(75) @ 160020: uvm_test_top [Test run phase] Read sequence Finished
UVM_INFO wrapper_test.sv(77) @ 160020: uvm_test_top [Test run phase] Write Read sequence Asserted
UVM_INFO wrapper_test.sv(79) @ 220020: uvm_test_top [Test run phase] Write Read sequence Finished
UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 220020: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO Ram_scoreboard.sv(54) @ 220020: uvm_test_top.r_env.sb [Scoreboard Report Phase] Testbench Completed , Correct count=11001, error count=0
UVM_INFO spi_scoreboard.sv(55) @ 220020: uvm_test_top.s_env.sb [Scoreboard Report Phase] Testbench Completed , Correct count=11001, error count=0
UVM_INFO wrapper_scoreboard.sv(55) @ 220020: uvm_test_top.w_env.sb [Scoreboard Report Phase] Testbench Completed , Correct count=11001, error count=0

--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :   15
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[Questa UVM]     2
[RNTST]     1
[Scoreboard Report Phase]     3
[TEST_DONE]     1
[Test run phase]     8
** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
   Time: 220020 ns  Iteration: 61  Instance: /top
Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430

SIM 3>
```

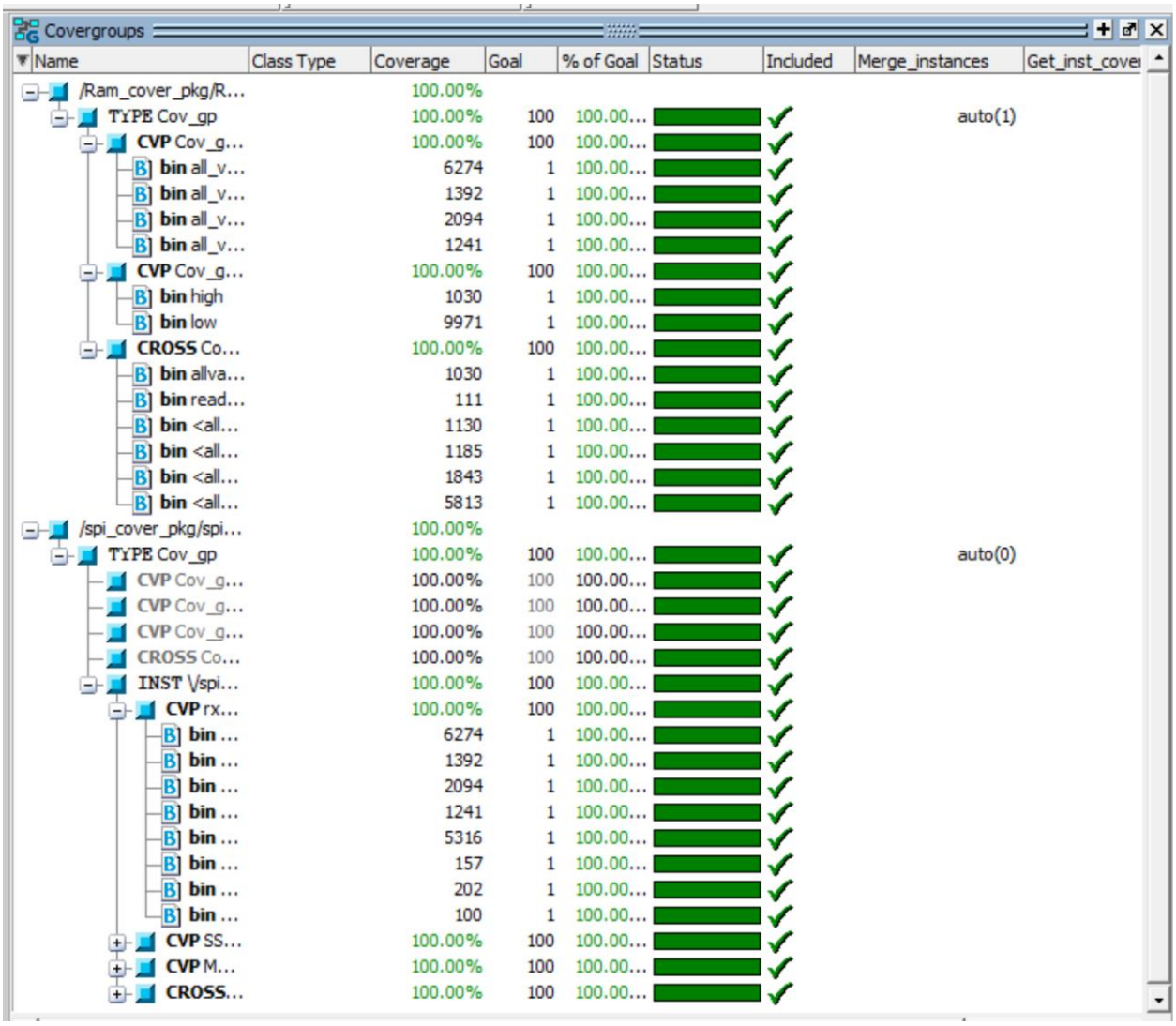
## 4.36.2.Questasim Code Coverage

**Toggle Coverage:**

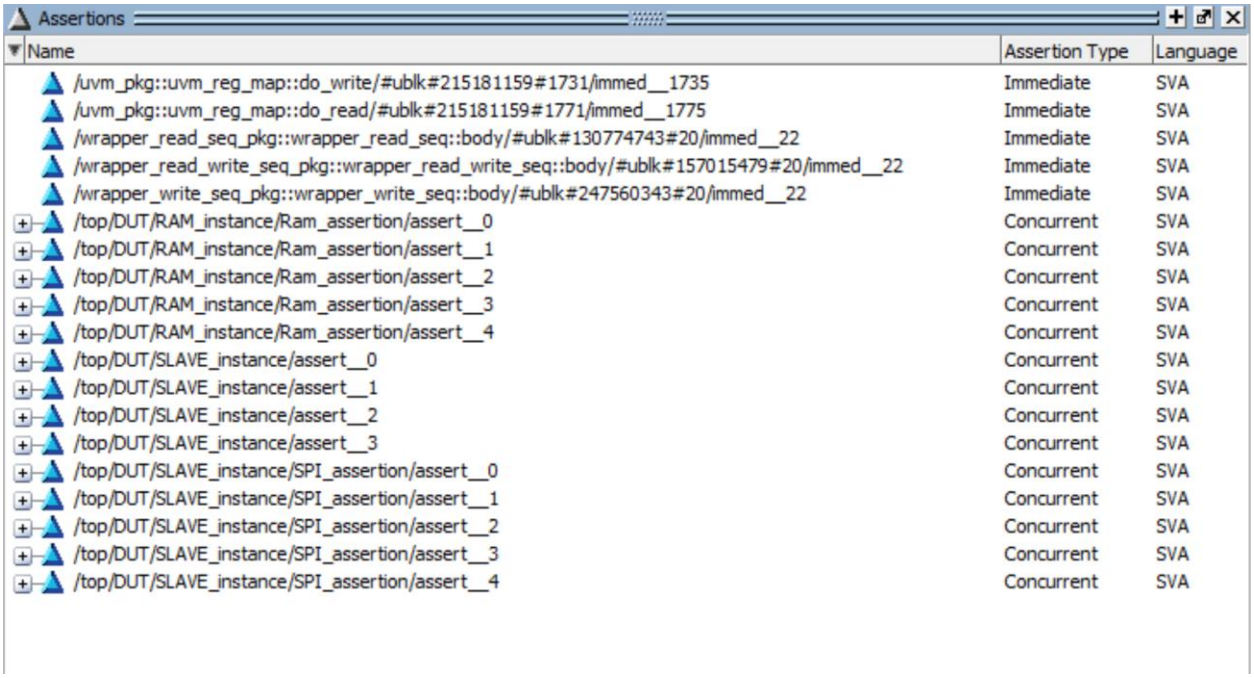



**Note:** No Statement or Branch Coverage as it is already covered in the Ram and SPI modules, while in the Wrapper no statement or branch to cover.

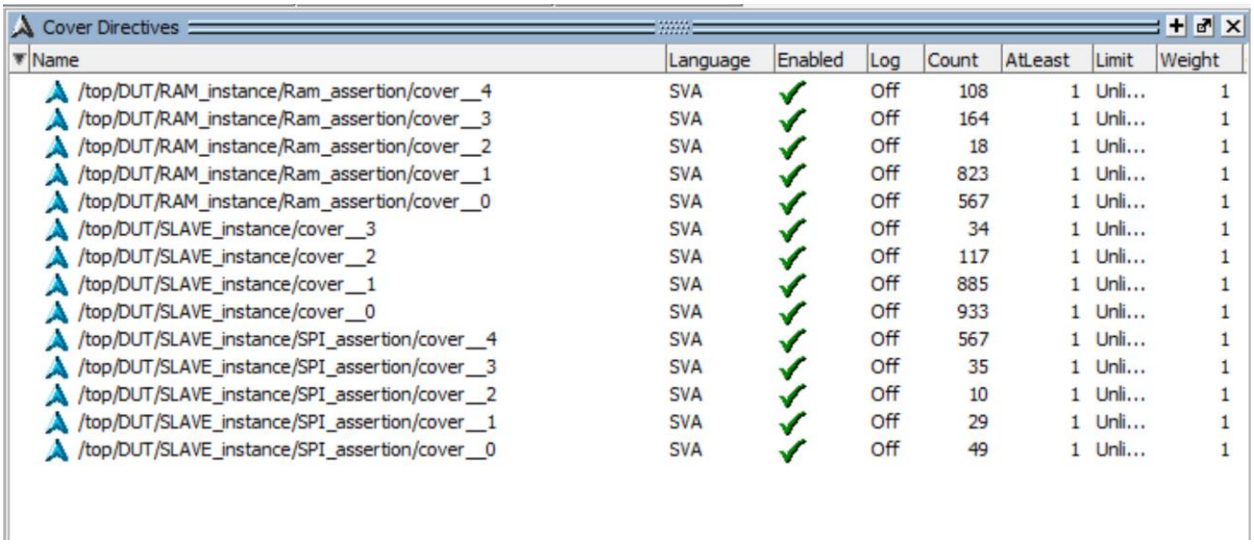
## 4.36.3.Questasim Functional Coverage



## 4.36.4.Questasim Assertions

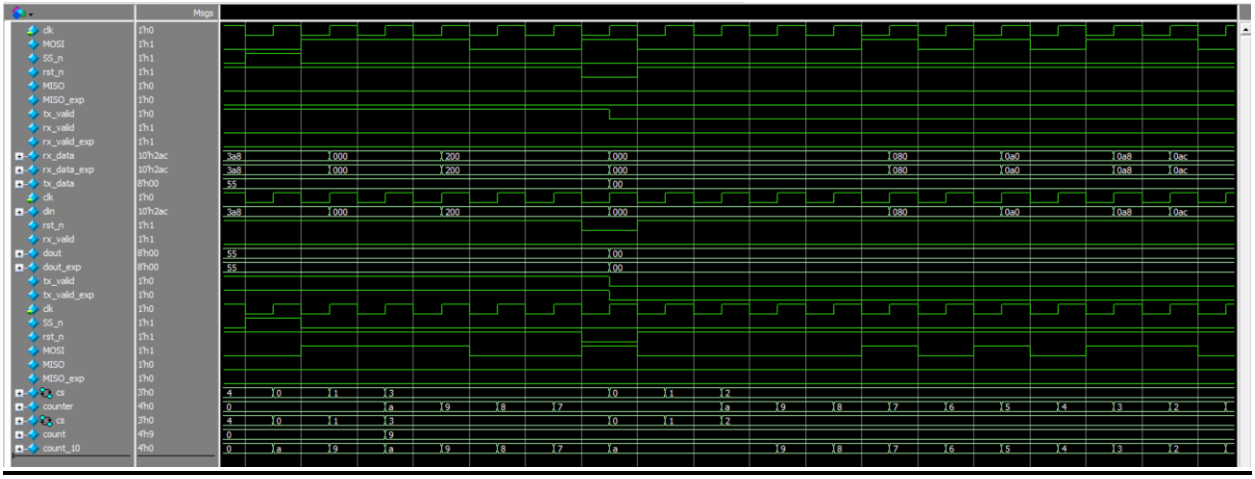



## 4.36.5.Questasim Assertions Coverage

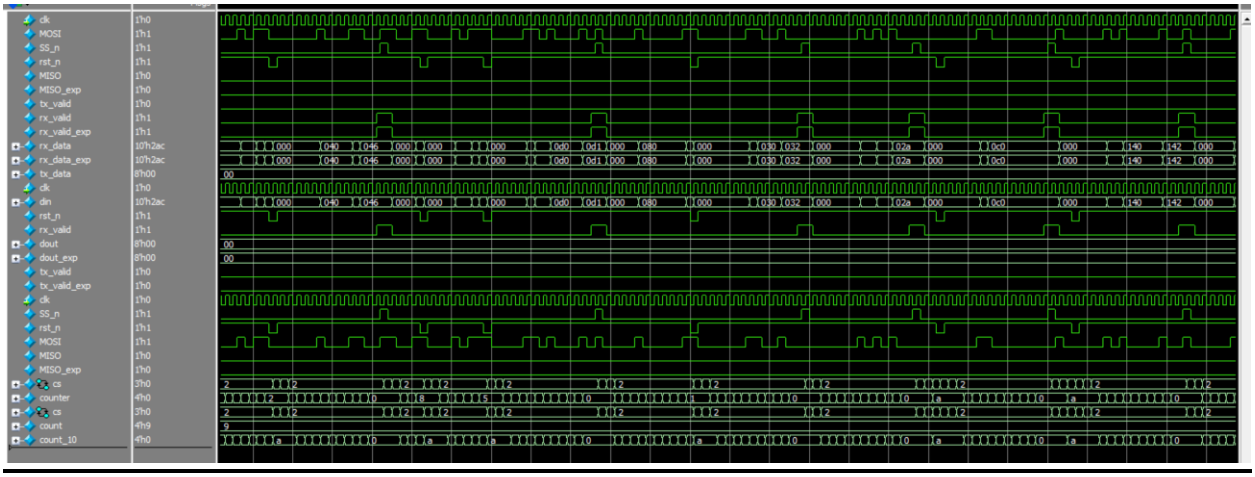


# 4.36.6.Questasim Waveforms

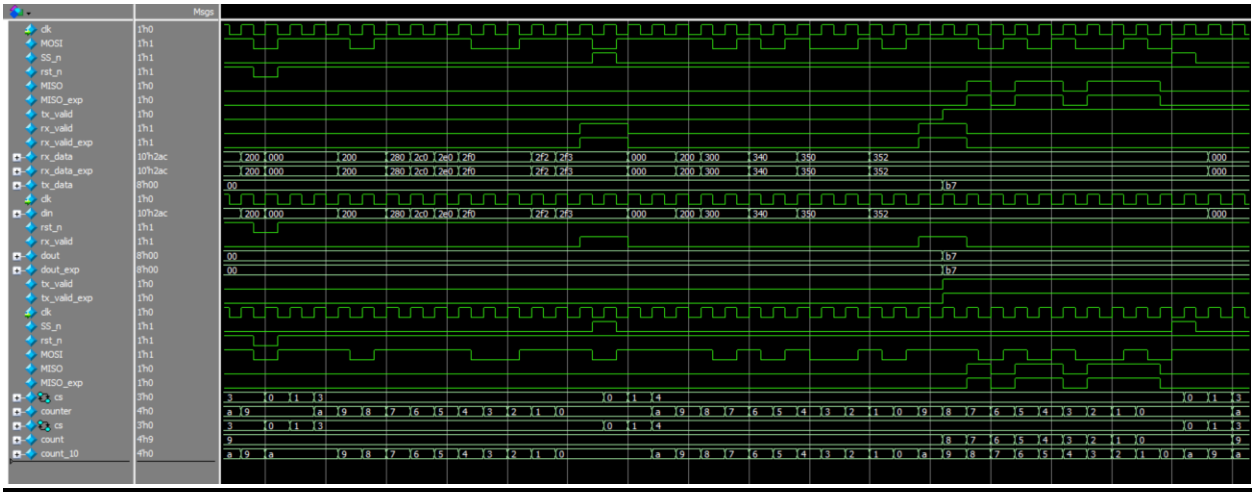
## Reset sequence:



## Write Sequence:



## Read Address Sequence:



## Read Data Sequence: