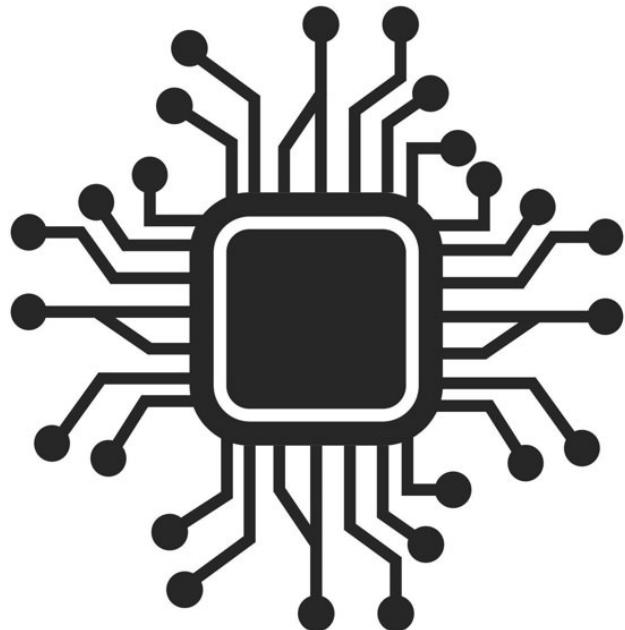


Digital Verification using SV & UVM

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 Requirement 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.
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 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 3'0000, 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 , 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 3'0001, 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 3'110, 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 3'111, 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

```
1 interface spi_if(clk);
2     input clk;
3     logic MOSI, SS_n, rst_n;
4     logic MISO,MISO_exp;
5     logic tx_valid,rx_valid,rx_valid_exp;
6     logic [9:0] rx_data,rx_data_exp;
7     logic [7:0] tx_data;
8
9 endinterface
```

2.4.SPI Slave Design

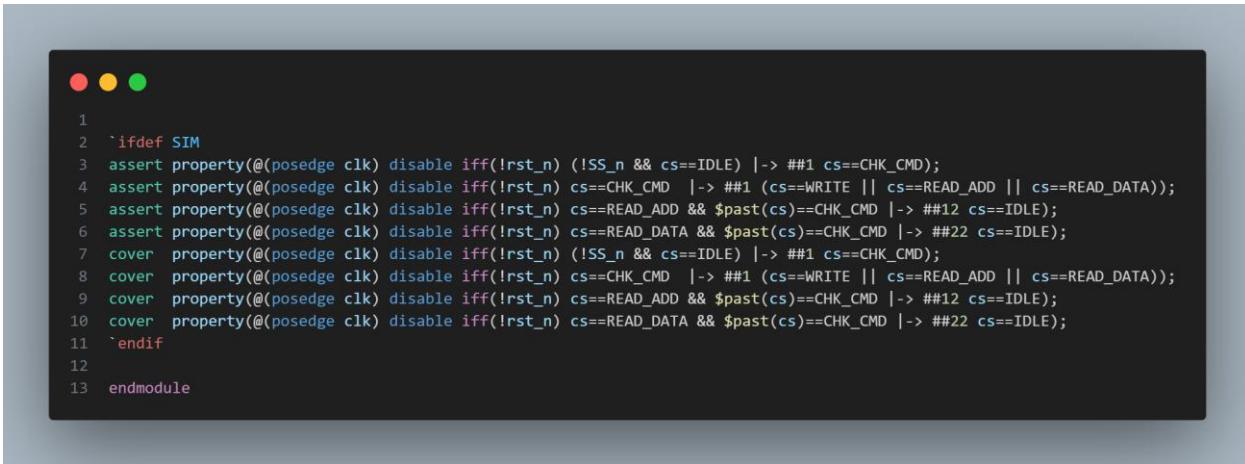
```
1 module SLAVE (MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
2
3 localparam IDLE      = 3'b000;
4 localparam WRITE     = 3'b010;
5 localparam CHK_CMD   = 3'b001;      //Design Modification => making the CHK_CMD parametrized with 2 instead of the WRITE for readability
6 localparam READ_ADD  = 3'b011;
7 localparam READ_DATA = 3'b100;
8
9 input      MOSI, clk, rst_n, SS_n, tx_valid;
10 input [7:0] tx_data;
11 output reg [9:0] rx_data;
12 output reg      rx_valid, MISO;
13
14 reg [3:0] counter;
15 reg      received_address;
16
17 reg [2:0] cs, ns;
18
19 always @(posedge clk) begin
20     if (!rst_n) begin
21         cs <= IDLE;
22     end
23     else begin
24         cs <= ns;
25     end
26 end
27
28 always @(*) begin
29     case (cs)
30         IDLE : begin
31             if (SS_n)
32                 ns = IDLE;
33             else
34                 ns = CHK_CMD;
35         end
36         CHK_CMD : begin
37             if (SS_n)
38                 ns = IDLE;
39             else begin
40                 if (!MOSI)
41                     ns = WRITE;
42                 else begin
43                     if (!received_address)      //Design Error : should check for not finishing Read_add
44                         ns = READ_ADD;
45                     else
46                         ns = READ_DATA;
47                 end
48             end
49         end
50         WRITE : begin
51             if (SS_n)
52                 ns = IDLE;
53             else
54                 ns = WRITE;
55         end
56         READ_ADD : begin
57             if (SS_n)
58                 ns = IDLE;
59             else
60                 ns = READ_ADD;
61         end
62     end
63 
```

Digital Verification using SV & UVM
UVM Project: SPI Slave with a Single Port RAM
10/10/2025

```
1      READ_DATA : begin
2          if (SS_n)
3              ns = IDLE;
4          else
5              ns = READ_DATA;
6      end
7  endcase
8 end
9
10 always @ (posedge clk) begin
11     if (~rst_n) begin
12         rx_data <= 0;
13         rx_valid <= 0;
14         received_address <= 0;
15         MISO <= 0;
16     end
17     else begin
18         case (cs)
19             IDLE : begin
20                 rx_valid <= 0;
21                 MISO <= 0;           //Design Error: MISO must be reseted
22                 rx_data <= 0;       //Design Error: should reset the rx_data
23             end
24             CHK_CMD : begin
25                 counter <= 10;
26             end
27             WRITE : begin
28                 if (counter > 0) begin
29                     rx_data[counter-1] <= MOSI;
30                     counter <= counter - 1;
31                 end
32                 else begin
33                     rx_valid <= 1;
34                 end
35             end
36             READ_ADD : begin
37                 if (counter > 0) begin
38                     rx_data[counter-1] <= MOSI;
39                     counter <= counter - 1;
40                 end
41                 else begin
42                     rx_valid <= 1;
43                     received_address <= 1;
44                 end
45             end
46             READ_DATA : begin
47                 if (tx_valid) begin
48                     rx_valid <= 0;
49                     if (counter > 0) begin
50                         MISO <= tx_data[counter-1];
51                         counter <= counter - 1;
52                     end
53                     else begin
54                         MISO<=0;
55                         received_address <= 0;
56                     end
57                 end
58             end
59             else begin
60                 MISO<=0;
61                 if (counter > 0) begin
62                     rx_data[counter-1] <= MOSI;
63                     counter <= counter - 1;
64                 end
65                 else begin
66                     rx_valid <= 1;
67                     counter <= 9;        //Design Error: the counter should be reloaded to 9 as the slave would wait one cycle for the tx_valid
68                 end
69             end
70         end
71     endcase
72 end
73 end
74
```

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 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

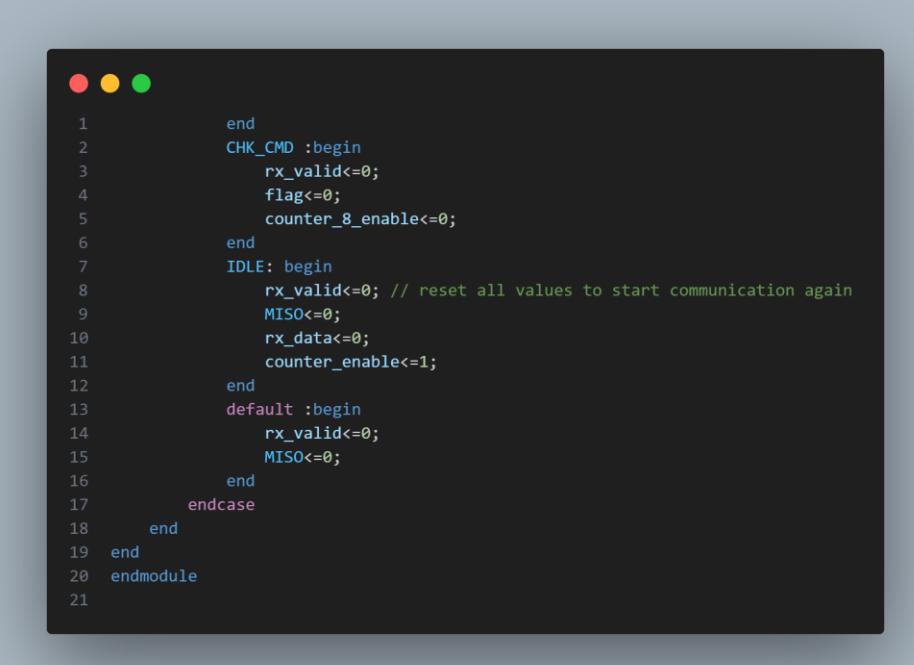
2.5.1.Counter Module

```
1 module down_counter #(parameter MOD=8,parameter COUNTER_BITS=3)(clk,rstn,enable,count);
2
3   input clk,rstn,enable;
4   output reg [COUNTER_BITS-1:0] count;
5
6   always @(posedge clk) begin
7     if(~rstn) begin
8       count<=MOD-1;
9     end
10    else if(enable) begin
11      if(count==0)begin
12        count<=MOD-1;
13      end
14      if(count>0) begin
15        count<=count-1;
16      end
17    end
18  end
19 endmodule
```

2.5.2.SPI Slave Module

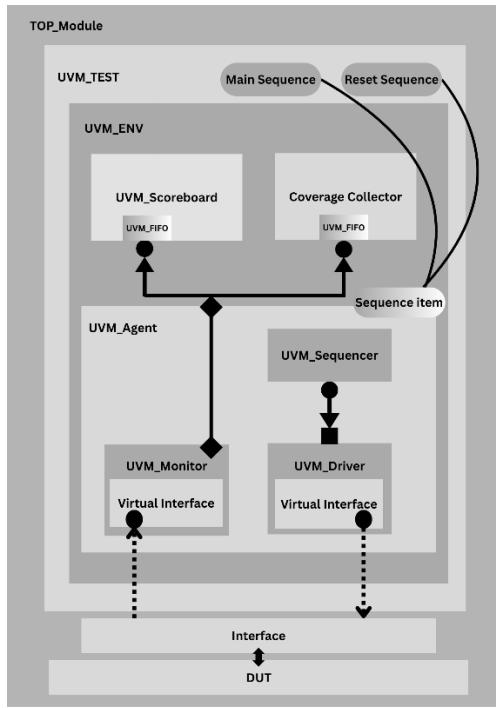
```
1 module SPI_slave_op(MOSI,MISO,clk,rstn,rx_data,tx_data,rx_valid,tx_valid,SS_n);
2
3 //our state encoding;
4 localparam IDLE      = 3'b000;
5 localparam CHK_CMD   = 3'b001;
6 localparam WRITE     = 3'b010;
7 localparam READ_ADD  = 3'b011;
8 localparam READ_DATA = 3'b100;
9
10
11 //SPI slave ports
12 input MOSI,clk,rstn,tx_valid,SS_n;
13 input [7:0]tx_data;
14 output reg MISO,rx_valid;
15 output reg [9:0]rx_data;
16
17 (* fsm_encoding="sequential"*)
18 reg [2:0]ns,cs;           //next and current state
19
20 //internal signals
21 wire [3:0]count;
22 wire [3:0]count_10;
23 reg flag,Read_add_data_inx,counter_enable,counter_8_enable;
24
25 //counters
26 down_counter #(11,4) down_counter_10(clk,rstn && cs!=CHK_CMD ,counter_enable,count_10); //counter counts from 10 to 0 ,11 states
27 down_counter #(10,4)tx_to_MISO(clk,rstn&& (cs!=CHK_CMD),counter_8_enable,count);           //counter counts from 9 to 0, 10 states
28
29 //next state logic
30 always @(*) begin
31   case (cs)
32     IDLE:begin
33       if(SS_n) ns=IDLE;
34       else begin
35         ns=CHK_CMD;
36       end
37     end
38     CHK_CMD:begin
39       if(SS_n) ns=IDLE;
40       else begin
41         if(MOSI) begin
42           if(!Read_add_data_inx) begin//read addr
43             ns=READ_ADD;
44           end
45           else if (Read_add_data_inx) begin
46             ns=READ_DATA;
47           end
48         end
49         else begin
50           ns=WRITE;
51         end
52       end
53     end
54     WRITE:begin
55       if(SS_n) ns=IDLE;
56       else begin
57         ns=WRITE;
58       end
59     end
60     READ_ADD:begin
61       if(SS_n) ns=IDLE;
62       else begin
63         ns=READ_ADD;
64       end
65     end
66     READ_DATA: begin
67       if(SS_n) ns=IDLE;
68       else begin
69         ns=READ_DATA;
70       end
71     end
72   endcase
73 end
74
```

```
1 //memory logic
2 always @(posedge clk) begin
3     if (!rstn) begin
4         cs<=3'b000;
5     end
6     else begin
7         cs<=ns;
8     end
9 end
10
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                     MOSI<=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                     MOSI<=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                     MOSI<=0;
49                     if(count_10>4'b0) begin
50                         if(flag) begin
51                             rx_data[count_10-2]<=MOSI;
52                             MOSI<=0;
53                         end
54                         else begin
55                             rx_data[count_10-1]<=MOSI;
56                             MOSI<=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
72             end
73         endcase
74     end
75 end
```



```
1      end
2      CHK_CMD :begin
3          rx_valid<=0;
4          flag<=0;
5          counter_8_enable<=0;
6      end
7      IDLE: begin
8          rx_valid<=0; // reset all values to start communication again
9          MISO<=0;
10         rx_data<=0;
11         counter_enable<=1;
12     end
13     default :begin
14         rx_valid<=0;
15         MISO<=0;
16     end
17     endcase
18 end
19 end
20 endmodule
21
```

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

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Object : Sequence Item
4 //////////////////////////////////////////////////////////////////
5
6 package spi_seq_item;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import shared_pkg::*;
10
11  class spi_seq_item extends uvm_sequence_item;
12    `uvm_object_utils(spi_seq_item)
13    //Constructor function
14    function new(string name="spi_seq_item");
15      super.new(name);
16    endfunction
17
18    typedef enum {IDLE,CHK_CMD,WRITE,READ_ADD,READ_DATA} my_FSM;
19    //Interface signals
20    //Inputs
21    rand logic MOSI, rst_n, SS_n;
22    rand logic [7:0] tx_data;
23    rand logic tx_valid;
24    rand logic [10:0] valid_comb;
25    //Outputs
26    logic rx_valid,rx_valid_exp;
27    logic [9:0] rx_data,rx_data_exp;
28    logic MISO,MISO_exp;
29    //Internal signals and sequence item counters
30    logic [10:0] old_valid_comb;
31    int MOSI_count,i,read_add_data;
32    my_FSM old_cs;
33    rand my_FSM cs;
34    //Constrain Blocks
35    constraint reset_constraints{rst_n dist{1:/95 , 0:/5};}
36    constraint SS_n_constraints {
37      if (old_cs != READ_DATA ) {
38        (MOSI_count == 13) -> SS_n == 1;
39        (MOSI_count != 13) -> SS_n == 0;
40      }
41      else if(old_cs==READ_DATA) {
42        (MOSI_count == 23) -> SS_n == 1;
43        (MOSI_count != 23) -> SS_n == 0;
44      }
45    }
46    constraint cs_constraints{if(!rst_n || SS_n) {cs==IDLE;}
47                             else if(old_cs==IDLE && !SS_n) {cs==CHK_CMD;}
48                             else if(old_cs==CHK_CMD && !SS_n && !MOSI) {cs==WRITE;}
49                             else if(old_cs==CHK_CMD && !SS_n && MOSI && read_add_data==0) {cs==READ_ADD;}
50                             else if(old_cs==WRITE || old_cs==READ_ADD) && MOSI_count==13 ) {cs==IDLE;}
51                             else if((old_cs==WRITE) && MOSI_count!=13 ) {cs==WRITE;}
52                             else if((old_cs==READ_ADD) && MOSI_count!=13 ) {cs==READ_ADD;}
53                             else if((old_cs==READ_DATA) && MOSI_count==23 ) {cs==IDLE;}
54                             else if((old_cs==READ_DATA) && MOSI_count!=23 ) {cs==READ_DATA;}
55                           }
56    constraint valid_comb_constraints{
57      if(!rst_n || MOSI_count==0) valid_comb[10:8] inside{3'b000,3'b001,3'b110,3'b111};
58      else valid_comb==old_valid_comb;
59    constraint MOSI_constraints {if(!SS_n && rst_n) MOSI==valid_comb[10-i] ;}
60    constraint tx_valid_constraints{if(MOSI_count>=14 && MOSI_count<=22) {tx_valid==1;} else{tx_valid==0;}}
61
62  
```

```
1     function void post_randomize();
2         old_valid_comb=valid_comb;
3         if(!rst_n) begin
4             old_cs=IDLE;
5             MOSI_count=0;
6             i=0;
7             read_add_data=0;
8         end
9         else begin
10            old_cs=cs;
11            if(cs==READ_ADD) read_add_data=1;
12            else if(cs==READ_DATA) read_add_data=0;
13        end
14        if(cs!=READ_DATA) begin
15            if(MOSI_count>=13) begin
16                MOSI_count=0;
17            end
18            else begin
19                MOSI_count++;
20            end
21        end
22        else begin
23            if(MOSI_count>=23) begin
24                MOSI_count=0;
25            end
26            else begin
27                MOSI_count++;
28            end
29        end
30    end
31    i++;
32    if(i==11 || MOSI_count==1) i=0; //to start sampling the valid_comb at the CHM_CMD state always
33 end
34 endfunction
35
36 function string convert2string_stimulus();
37     return $sformatf("rst=
38 ,SS_Endfunction
39 ,MOSIfunction string convert2string();
40 ,tx_validreturn $sformatf("rst=
41 ,\$s_data=\$0\$\$1\$t\$\$t\$\$n,MOSI,tx_valid,tx_data);
42 ,MOSIdata,rx_data,rx_data_exp);
43 ,tx_data\$\$dexp\$n
44 ,"\$rst\$\$t\$\$t\$\$n,MOSI,tx_valid,tx_data,MISO,MISO_exp,rx_valid,rx_valid_exp
45 \$MOSI\$tx\$\$dexp\$n
46 ,rx_valid=
```

2.8.SPI Sequencer

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Sequencer
4 //////////////////////////////////////////////////////////////////
5
6 package spi_sequencer_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import spi_seq_item::*;
10  import shared_pkg::*;
11  class spi_sqr extends uvm_sequencer #(spi_seq_item);
12    `uvm_component_utils(spi_sqr)
13
14    function new(string name="spi_sqr",uvm_component parent=null);
15      super.new(name,parent);
16    endfunction
17  endclass
18 endpackage
```

2.9.SPI Configuration Object

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

2.10.SPI Reset Sequence

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Object : Reset Sequence
4 //////////////////////////////////////////////////////////////////
5
6 package spi_reset_seq_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import spi_seq_item::*;
10
11 class spi_reset_seq extends uvm_sequence #(spi_seq_item);
12   `uvm_object_utils(spi_reset_seq)
13
14   spi_seq_item seq_item;
15
16   function new(string name ="spi_reset_seq");
17     super.new(name);
18   endfunction
19
20   task body;
21     seq_item=spi_seq_item::type_id::create("seq_item");
22     start_item(seq_item);
23     seq_item.rst_n=0;
24     seq_item.MOSI=$random;
25     finish_item(seq_item);
26   endtask
27 endclass
28 endpackage
```

2.11.SPI Main Sequence

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Object : Reset Sequence
4 //////////////////////////////////////////////////////////////////
5
6 package spi_main_seq_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import spi_seq_item::*;
10
11 class spi_main_seq extends uvm_sequence #(spi_seq_item);
12   `uvm_object_utils(spi_main_seq)
13
14   spi_seq_item seq_item;
15
16   function new(string name ="spi_main_seq");
17     super.new();
18   endfunction
19
20   virtual task body();
21     seq_item=spi_seq_item::type_id::create("seq_item");
22     repeat(2000) begin
23       start_item(seq_item);
24       assert(seq_item.randomize());
25       finish_item(seq_item);
26     end
27   endtask
28 endclass
29 endpackage
```

2.12.SPI Driver

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Driver
4 //////////////////////////////////////////////////////////////////
5
6 package spi_driver_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9
10  import spi_config_pkg::*;
11  import spi_seq_item::*;
12
13 class spi_driver extends uvm_driver #(spi_seq_item);
14   `uvm_component_utils(spi_driver)
15
16   //Virtual interface & sequence item object
17   virtual spi_if spi_driver_vif;
18   spi_seq_item drv_seq_item;
19
20   //Constructor
21   function new(string name="spi_driver",uvm_component parent=null);
22     super.new(name,parent);
23   endfunction
24
25   //Build_Phase
26   function void build_phase(uvm_phase phase);
27     super.build_phase(phase);
28   endfunction
29
30   //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
31   task run_phase(uvm_phase phase);
32     super.run_phase(phase);
33     forever begin
34       //Create seq_item
35       drv_seq_item=spi_seq_item::type_id::create("drv_seq_item");
36       //seq_item_port.get_next_item : get the values from the driver port connected with the sequencer to the driver seq item
37       seq_item_port.get_next_item(drv_seq_item);
38
39       //Drive the interface with the driver sequence item
40       spi_driver_vif.rst_n=drv_seq_item.rst_n;
41       spi_driver_vif.MOSI = drv_seq_item.MOSI;
42       spi_driver_vif.tx_data=drv_seq_item.tx_data;
43       spi_driver_vif.SS_n=drv_seq_item.SS_n;
44       spi_driver_vif.tx_valid=drv_seq_item.tx_valid;
45
46       @(negedge spi_driver_vif.clk);
47       //Transaction done , now get next item and so on ...
48       seq_item_port.item_done();
49       `uvm_info("Driver run phase",drv_seq_item.convert2string_stimulus(),UVM_HIGH)
50     end
51   endtask
52
53 endclass
54
55 endpackage
```

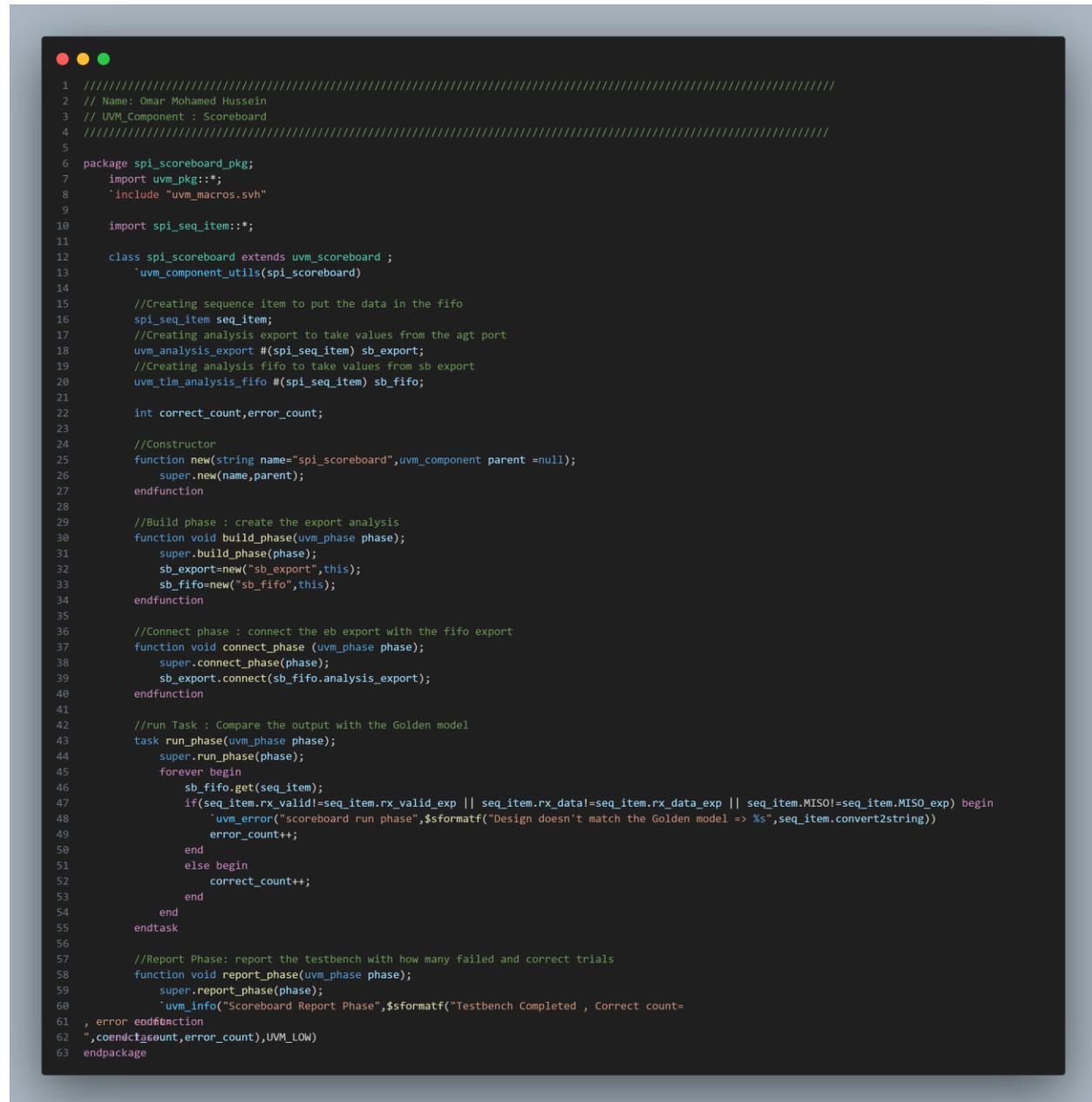
2.13.SPI Monitor

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Monitor
4 //////////////////////////////////////////////////////////////////
5
6 package spi_monitor_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import spi_seq_item::*;
10  import shared_pkg::*;
11  class spi_monitor extends uvm_monitor ;
12    `uvm_component_utils(spi_monitor)
13
14    //Creating Virtual interface to get the data from the interface
15    virtual spi_if s_if;
16    //Creating monitor sequence item to pass the interface to it to be then sent to the analysis components
17    spi_seq_item mon_seq_item;
18    //Creating monitor analysis port to broadcast the values to the analysis components
19    uvm_analysis_port #(spi_seq_item) mon_ap;
20
21    //Constructor
22    function new(string name="spi_monitor",uvm_component parent=null);
23      super.new(name,parent);
24    endfunction
25
26    //Build Phase
27    function void build_phase(uvm_phase phase);
28      super.build_phase(phase);
29      mon_ap=new("mon_ap",this);
30    endfunction
31
32    //Run Phase: samples the data at the negative edge and writes the analysis port with the monitor sequence item
33    task run_phase(uvm_phase phase);
34      super.run_phase(phase);
35      forever begin
36        mon_seq_item=spi_seq_item::type_id::create("mon_seq_item");
37        @(negedge s_if.clk);
38
39        //Drive the monitor sequence item with the interface
40        mon_seq_item.rst_n=s_if.rst_n;
41        mon_seq_item.SS_n=s_if.SS_n;
42        mon_seq_item.MOSI=s_if.MOSI;
43        mon_seq_item.MISO=s_if.MISO;
44        mon_seq_item.MISO_exp=s_if.MISO_exp;
45        mon_seq_item.tx_valid=s_if.tx_valid;
46        mon_seq_item.tx_data=s_if.tx_data;
47        mon_seq_item.rx_valid=s_if.rx_valid;
48        mon_seq_item.rx_valid_exp=s_if.rx_valid_exp;
49        mon_seq_item.rx_data=s_if.rx_data;
50        mon_seq_item.rx_data_exp=s_if.rx_data_exp;
51
52        //Broadcast the monitor sequence (interfance) to the analysis port
53        mon_ap.write(mon_seq_item);
54        `uvm_info("Monitor run phase",mon_seq_item.convert2string(),UVM_HIGH)
55      end
56    endtask
57  endclass
58 endpackage
```

2.14.SPI Agent

```
1 //////////////////////////////////////////////////////////////////
2 //Name: Omar Mohamed Hussein
3 //UVM_Component: Agent
4 //////////////////////////////////////////////////////////////////
5 package spi_agent_pkg;
6
7 import uvm_pkg::*;
8 `include "uvm_macros.svh"
9
10 import spi_seq_item::*;
11 import spi_sequencer_pkg::*;
12 import spi_driver_pkg::*;
13 import spi_monitor_pkg::*;
14 import spi_config_pkg::*;
15
16 class spi_agent extends uvm_agent;
17   `uvm_component_utils(spi_agent)
18
19   //creating the sequencer,driver,monitor, configuration object to connect the interface,agent port
20   spi_sqr sqr;
21   spi_driver drv;
22   spi_monitor mon;
23   spi_config_obj config_obj;
24   uvm_analysis_port #(spi_seq_item) agt_ap;
25
26   //Constructor
27   function new(string name="spi_agent",uvm_component parent=null);
28     super.new(name,parent);
29   endfunction
30
31   //Build Phase
32   function void build_phase(uvm_phase phase);
33     super.build_phase(phase);
34     //Connect the agent configuration object with the configuration object in the uvm config data base
35     if(!uvm_config_db #(spi_config_obj)::get(this,"","CFG",config_obj)) begin
36       `uvm_fatal("agent build phase","unable to get the configuration object to the agent")
37     end
38
39     //////////////// For Passive Agents ///////////////////
40     // if(shift_reg_cfg.is_active==UVM_ACTIVE) begin
41     //   sqr=sequencer::type_id::create("sqr",this);
42     //   drv=shift_reg_driver::type_id::create("drv",this);
43     // end
44     ////////////////
45
46     //Creating the sequencer,driver,monitor with the create() and the agt port with new()
47     sqr=spi_sqr ::type_id::create("sqr",this);
48     drv=spi_driver ::type_id::create("drv",this);
49     mon=spi_monitor::type_id::create("mon",this);
50     agt_ap=new("agt_ap",this);
51   endfunction
52
53   //connect phase
54   function void connect_phase(uvm_phase phase);
55     super.connect_phase(phase);
56
57     ////////////////For Passive Agents ///////////////////
58     // if(spi_cfg.is_active==UVM_ACTIVE) begin
59     //   drv.spi_vif=spi_cfg.spi_vif;
60     //   drv.seq_item_port.connect(sqr.seq_item_export);
61     // end
62     ////////////////
63
64     //Connect the configuration object with the driver and the monitor ports
65     drv.spi_driver_vif=config_obj.spi_config_vif;
66     mon.s_if=config_obj.spi_config_vif;
67     //Connect the driver port with the sequencer export
68     drv.seq_item_port.connect(sqr.seq_item_export);
69     //Connect the monitor analysis port with the agt port
70     mon.mon_ap.connect(agt_ap);
71   endfunction
72   endclass
73 endpackage
```

2.15.SPI Scoreboard



```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Scoreboard
4 //////////////////////////////////////////////////////////////////
5
6 package spi_scoreboard_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9
10  import spi_seq_item::*;
11
12  class spi_scoreboard extends uvm_scoreboard ;
13    `uvm_component_utils(spi_scoreboard)
14
15    //Creating sequence item to put the data in the fifo
16    spi_seq_item seq_item;
17    //Creating analysis export to take values from the agt port
18    uvm_analysis_export #(spi_seq_item) sb_export;
19    //Creating analysis fifo to take values from sb export
20    uvm_tlm_analysis_fifo #(spi_seq_item) sb_fifo;
21
22    int correct_count,error_count;
23
24    //Constructor
25    function new(string name="spi_scoreboard",uvm_component parent =null);
26      super.new(name,parent);
27    endfunction
28
29    //Build phase : create the export analysis
30    function void build_phase(uvm_phase phase);
31      super.build_phase(phase);
32      sb_export=new("sb_export",this);
33      sb_fifo=new("sb_fifo",this);
34    endfunction
35
36    //Connect phase : connect the eb export with the fifo export
37    function void connect_phase (uvm_phase phase);
38      super.connect_phase(phase);
39      sb_export.connect(sb_fifo.analysis_export);
40    endfunction
41
42    //run Task : Compare the output with the Golden model
43    task run_phase(uvm_phase phase);
44      super.run_phase(phase);
45      forever begin
46        sb_fifo.get(seq_item);
47        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
48          `uvm_error("scoreboard run phase",$sformatf("Design doesn't match the Golden model => %s",seq_item.convert2string))
49          error_count++;
50        end
51        else begin
52          correct_count++;
53        end
54      end
55    endtask
56
57    //Report Phase: report the testbench with how many failed and correct trials
58    function void report_phase(uvm_phase phase);
59      super.report_phase(phase);
60      `uvm_info("Scoreboard Report Phase",$sformatf("Testbench Completed , Correct count= %s",correct_count),UVM_LOW)
61      , error endfunction
62      ",coendif@count,error_count),UVM_LOW)
63    endpackage
```

2.16.SPI Cover

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Cover
4 //////////////////////////////////////////////////////////////////
5
6 package spi_cover_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import spi_seq_item::*;
10  import shared_pkg::*;
11
12 class spi_cover extends uvm_component;
13   `uvm_component_utils(spi_cover)
14   uvm_analysis_export #(spi_seq_item) cover_ap;
15   uvm_tlm_analysis_fifo #(spi_seq_item) cover_fifo;
16   spi_seq_item seq_item;
17
18
19 //Cover Groups
20   covergroup Cov_gp(ref spi_seq_item seq_item);
21   //Normal Coverpoints
22   RX_cp: coverpoint seq_item.rx_data[9:8];
23     bins all_values[] = {[0:3]};
24     bins WRITE_ADD =(2'b00=>2'b00=>2'b00) ;
25     bins WRITE_DATA=(2'b00=>2'b00=>2'b01) ;
26     bins READ_ADD =(2'b00=>2'b10=>2'b10) ;
27     bins READ_DATA =(2'b00=>2'b10=>2'b11) ;
28   }
29   SS_n_cp: coverpoint seq_item.SS_n{
30     bins read_transition=(1=> 0[*23] =>1);
31     bins not_read_transition=(1=> 0[*13] =>1);
32   }
33   MOSI_cp:coverpoint seq_item.MOSI{
34     bins b000=(0=>0=>0);
35     bins b001=(0=>0=>1);
36     bins b111=(1=>1=>1);
37     bins b110=(1=>1=>0);
38   }
39   cross_cv:cross MOSI_cp,SS_n_cp{
40     ignore_bins write_cv_000 =binsof(MOSI_cp.b000) && binsof(SS_n_cp.read_transition);
41     ignore_bins write_cv_001 =binsof(MOSI_cp.b001) && binsof(SS_n_cp.read_transition);
42     ignore_bins read_cv_111 =binsof(MOSI_cp.b111) && binsof(SS_n_cp.not_read_transition);
43     ignore_bins read_cv_110 =binsof(MOSI_cp.b110) && binsof(SS_n_cp.not_read_transition);
44   }
45   //Cross coverage
46 endgroup
47
48 function new(string name="spi_cover",uvm_component parent=null);
49   super.new(name,parent);
50   Cov_gp=new(seq_item);
51 endfunction
52
53 function void build_phase(uvm_phase phase);
54   super.build_phase(phase);
55   cover_ap=new("cover_ap",this);
56   cover_fifo=new("cover_fifo",this);
57 endfunction
58
59 function void connect_phase(uvm_phase phase);
60   super.connect_phase(phase);
61   cover_ap.connect(cover_fifo.analysis_export);
62 endfunction
63
64 task run_phase(uvm_phase phase);
65   super.run_phase(phase);
66   forever begin
67     seq_item=spi_seq_item::type_id::create("seq_item");
68     cover_fifo.get(seq_item);
69     Cov_gp.sample();
70   end
71 endtask
72
73 endclass
74 endpackage
```

2.17.SPI Environment

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Environment
4 //////////////////////////////////////////////////////////////////
5
6 package spi_env_pkg;
7 import uvm_pkg::*;
8 `include "uvm_macros.svh"
9 import spi_driver_pkg::*;
10 import spi_scoreboard_pkg::*;
11 import spi_agent_pkg::*;
12 import shared_pkg::*;
13 import spi_cover_pkg::*;
14 class spi_env extends uvm_env;
15   `uvm_component_utils(spi_env)
16   spi_scoreboard sb;
17   spi_agent agt;
18   spi_cover cv;
19
20   function new(string name="spi_env",uvm_component parent=null);
21     super.new(name,parent);
22   endfunction
23   function void build_phase(uvm_phase phase);
24     super.build_phase(phase);
25     sb=spi_scoreboard::type_id::create("sb",this);
26     agt=spi_agent::type_id::create("agt",this);
27     cv=spi_cover::type_id::create("spi_cover",this);
28   endfunction
29   function void connect_phase(uvm_phase phase);
30     super.connect_phase(phase);
31     agt.agt_ap.connect(sb.sb_export);
32     agt.agt_ap.connect(cv.cover_ap);
33   endfunction
34 endclass
35 endpackage
```

2.18.SPI Test

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Test
4 //////////////////////////////////////////////////////////////////
5
6 package spi_test_pkg;
7 import uvm_pkg::*;
8 `include "uvm_macros.svh"
9 import spi_env_pkg::*;
10 import spi_config_pkg::*;
11 import spi_reset_seq_pkg::*;
12 import spi_main_seq_pkg::*;
13   class spi_test extends uvm_test;
14     `uvm_component_utils(spi_test)
15
16     spi_env env;
17     spi_config_obj spi_config_obj_test;
18     spi_reset_seq reset_seq;
19     spi_main_seq main_seq;
20     function new(string name ="spi_test",uvm_component parent=null);
21       super.new(name,parent);
22     endfunction
23
24     function void build_phase(uvm_phase phase);
25       super.build_phase(phase);
26       env=spi_env::type_id::create("env",this);
27       spi_config_obj_test=spi_config_obj::type_id::create("spi_config_obj_test");
28       reset_seq=spi_reset_seq::type_id::create("reset_seq");
29       main_seq=spi_main_seq::type_id::create("main_seq");
30
31       if(!uvm_config_db #(virtual spi_if)::get(this,"","spi_if",spi_config_obj_test.spi_config_vif))
32         `uvm_fatal("build phase in test","unable to get interface from the database into the configuration object")
33       uvm_config_db#(spi_config_obj)::set(this,"*","CFG",spi_config_obj_test);
34     endfunction
35
36     task run_phase(uvm_phase phase);
37       super.run_phase(phase);
38       phase.raise_objection(this);
39       `uvm_info("Test run phase","Reset Asserted",UVM_LOW)
40       reset_seq.start(env.agt.sqr);
41       `uvm_info("Test run phase","Reset Finished",UVM_LOW)
42
43       `uvm_info("Test run phase","Main sequence Asserted",UVM_LOW)
44       main_seq.start(env.agt.sqr);
45       `uvm_info("Test run phase","Main sequence Finished",UVM_LOW)
46       phase.drop_objection(this);
47     endtask
48   endclass
49 endpackage
```

2.19.SPI Top

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM Component : TOP
4 //////////////////////////////////////////////////////////////////
5
6 import uvm_pkg::*;
7 `include "uvm_macros.svh"
8 import spi_test_pkg::*;
9
10 module top();
11   // Clock generation
12   bit clk;
13   initial begin
14     clk=0;
15     forever begin
16       #10 clk=~clk;
17     end
18   end
19   // Instantiate the interface ,Golden Model and DUT
20   spi_if s_if(clk);
21   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);
22   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);
23   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);
24
25 // run test using run_test task
26 initial begin
27   uvm_config_db #(virtual spi_if)::set(null,"uvm_test_top","spi_if",s_if);
28   run_test("spi_test");
29 end
30 endmodule
```

2.20.SPI Source File

```
1  shared_pkg.sv
2  spi_if.sv
3  SPI_slave.sv
4  spi_sva.sv
5  down_counter.v
6  SPI_slave_optimized.v
7  spi_config_obj.sv
8  spi_seq_item.sv
9  spi_reset_seq.sv
10 spi_main_seq.sv
11 spi_sqr.sv
12 spi_driver.sv
13 spi_monitor.sv
14 spi_agent.sv
15 spi_scoreboard.sv
16 spi_cover.sv
17 spi_env.sv
18 spi_test.sv
19 spi_top.sv
```

2.21.SPI DO File

```
1 vlib work
2 vlog +define+SIM -f src_files.list +cover -covercells
3 vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
4 add wave /top/s_if/*
5 add wave -position insertpoint \
6 sim:/top/Golden_spi/cs
7 add wave -position insertpoint \
8 sim:/top/slave/cs
9 add wave -position insertpoint \
10 sim:/top/Golden_spi/count \
11 sim:/top/Golden_spi/count_10
12 add wave -position insertpoint \
13 sim:/top/slave/counter
14 run 0
15 add wave -position insertpoint \
16 sim:/uvm_root/uvm_test_top/env/agt/drv/drv_seq_item
17 coverage save priority_en_tb.ucdb -onexit
18 run -all
```

2.22.SPI Assertions

```

1 module spi_sva(MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
2 input MOSI,MISO,SS_n,clk,rst_n,rx_valid,tx_valid;
3 input [9:0]rx_data;
4 input [7:0]tx_data;
5
6 sequence valid_comb_000;
7 ($fell(SS_n) ##1 !MOSI ##1 !MOSI ##1 !MOSI);
8 endsequence
9 sequence valid_comb_001;
10 ($fell(SS_n) ##1 !MOSI ##1 !MOSI ##1 MOSI);
11 endsequence
12 sequence valid_comb_110;
13 ($fell(SS_n) ##1 MOSI ##1 MOSI ##1 !MOSI);
14 endsequence
15 sequence valid_comb_111;
16 ($fell(SS_n) ##1 MOSI ##1 MOSI ##1 MOSI);
17 endsequence
18 assert property(@(posedge clk) disable iff(!rst_n) valid_comb_000 |-> ##10 (rx_valid ##1 SS_n[->1]));
19 assert property(@(posedge clk) disable iff(!rst_n) valid_comb_001 |-> ##10 (rx_valid ##1 SS_n[->1]));
20 assert property(@(posedge clk) disable iff(!rst_n) valid_comb_110 |-> ##10 (rx_valid ##1 SS_n[->1]));
21 assert property(@(posedge clk) disable iff(!rst_n) valid_comb_111 |-> ##10 (rx_valid ##1 SS_n[->1]));
22 assert property(@(posedge clk) !rst_n |=> !MISO && !rx_valid && rx_data==0);
23 cover property(@(posedge clk) disable iff(!rst_n) valid_comb_000 |-> ##10 (rx_valid ##1 SS_n[->1]));
24 cover property(@(posedge clk) disable iff(!rst_n) valid_comb_001 |-> ##10 (rx_valid ##1 SS_n[->1]));
25 cover property(@(posedge clk) disable iff(!rst_n) valid_comb_110 |-> ##10 (rx_valid ##1 SS_n[->1]));
26 cover property(@(posedge clk) disable iff(!rst_n) valid_comb_111 |-> ##10 (rx_valid ##1 SS_n[->1]));
27 cover property(@(posedge clk) !rst_n |=> !MISO && !rx_valid && rx_data==0);
28 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.ld/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.sv [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
# [RMST] 1
# [Scoreboard Report Phase] 1
# [TEST_DONE] 1
# [Test run phase] 4
# ** Note: $finish : C:/questasim64_2021.1/win64/..//verilog_src/uvm-1.ld/src/base/uvm_root.svh(430)
# Time: 40020 ns Iteration: 61 Instance: /top
# l
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/..//verilog_src/uvm-1.ld/src/base/uvm_root.svh line 430
```

VSIM 2>

2.23.2.Questasim Code Coverage

Statement Coverage:



```
spi_slave.sv
✓ 19 always @(posedge clk) begin
✓ 21 cs <= IDLE;
✓ 23 cs <= ns;
✓ 25 always @(*) begin
✓ 32 ns = IDLE;
✓ 34 ns = CHK_CMD;
✗ 36 ns = IDLE;
✓ 38 ns = WRITE;
✓ 40 ns = READ_ADD;
✓ 42 ns = READ_DATA;
✓ 44 ns = IDLE;
✓ 46 ns = IDLE;
✓ 48 ns = WRITE;
✓ 50 ns = IDLE;
✓ 52 ns = IDLE;
✓ 54 ns = IDLE;
✓ 56 ns = IDLE;
✓ 58 ns = IDLE;
✓ 60 ns = READ_ADD;
✓ 62 ns = IDLE;
✓ 64 ns = IDLE;
✓ 66 ns = READ_DATA;
✓ 71 always @(posedge clk) begin
✓ 73 rx_data <= {};
✓ 74 rx_valid <= {};
✓ 75 received_address <= {};
✓ 76 MISO <= {};
✓ 81 rx_valid <= {};
✓ 82 MISO <= {} //Design Error: MISO must be reseted
✓ 83 rx_data <= {} //Design Error: should reset the rx_data
✓ 87 counter <= {};
✓ 91 rx_data[counter-1] <= MOSI;
✓ 92 counter <= counter - 1;
✓ 95 rx_valid <= {};
✓ 100 rx_data[counter-1] <= MOSI;
✓ 101 counter <= counter - 1;
✓ 104 rx_valid <= {};
✓ 105 received_address <= {};
✓ 110 rx_valid <= {};
✓ 112 MISO <= tx_data[counter-1];
✓ 113 counter <= counter - 1;
✓ 116 MISO<= {};
✓ 117 received_address <= {};
✓ 121 MISO<= {};
✓ 123 rx_data[counter-1] <= MOSI;
✓ 124 counter <= counter - 1;
✓ 127 rx_valid <= {};
✓ 128 counter <= {} //Design Error: the counter should be reloaded to 9 as the slave would wait one cycle for the tx valid
```

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

Branch Coverage:

```

Branches - by instance (/top/slave)
Branch: SPI_slave.sv
  20 if (~rst_n) begin
  23 else begin
  29 case (cs)
  30 IDLE : begin
  31 if (SS_n)
  33 else
  36 CHK_CMD : begin
  37 if (SS_n)
  39 else begin
  40 if (~MOSI)
  42 else begin
  43 if (!received_address)      //Design Error : should check for not finishing Read_addr
  45 else
  50 WRITE : begin
  51 if (SS_n)
  53 else
  56 READ_ADD : begin
  57 if (SS_n)
  59 else
  62 READ_DATA : begin
  63 if (SS_n)
  65 else
  72 if (~rst_n) begin
  78 else begin
  79 case (cs)
  80 IDLE : begin
  86 CHK_CMD : begin
  89 WRITE : begin
  90 if (counter > 0) begin
  94 else begin
  98 READ_ADD : begin
  99 if (counter > 0) begin
  103 else begin
  108 READ_DATA : begin
  109 if (tx_valid) begin
  111 if (counter > 0) begin
  115 else begin
  120 else begin
  122 if (counter > 0) begin
  126 else begin
  
```

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

Toggle Coverage:

```

Toggles - by instance (/top/slave)
Toggle: sim:/top/slave
  clk
  counter
  cs
  MISO
  MOSI
  ns
  received_address
  rst_n
  rx_data
  rx_valid
  SS_n
  tx_data
  tx_valid
  
```

2.23.3.Questasim Functional Coverage

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_in
/spi_cover_pkg/spi_cover		100.00%					
TYPE Cov_gp		100.00%	100	100.00...		✓	
CVP Cov_gp::rx_cp		100.00%	100	100.00...		✓	
CVP Cov_gp::SS_n_cp		100.00%	100	100.00...		✓	
CVP Cov_gp::MOSI_cp		100.00%	100	100.00...		✓	
CROSS Cov_gp::cross_cv		100.00%	100	100.00...		✓	
INST \spi_cover_pkg::spi_cover::Cov_g...		100.00%	100	100.00...		✓	
CVP rx_cp		100.00%	100	100.00...		✓	
CVP SS_n_cp		100.00%	100	100.00...		✓	
CVP MOSI_cp		100.00%	100	100.00...		✓	
CROSS cross_cv		100.00%	100	100.00...		✓	
B bin <b110,not_read_data_transiti...		13	1	100.00...		✓	
B bin <b001,not_read_data_transiti...		41	1	100.00...		✓	
B bin <b000,not_read_data_transiti...		30	1	100.00...		✓	
B bin <b111,read_data_transition>		4	1	100.00...		✓	
B ignore_bin read_cv_110		4	-	-			
B ignore_bin read_cv_111		12	-	-			
B ignore_bin write_cv_001		5	-	-			
B ignore_bin write_cv_000		3	-	-			

2.23.4.Questasim Assertions

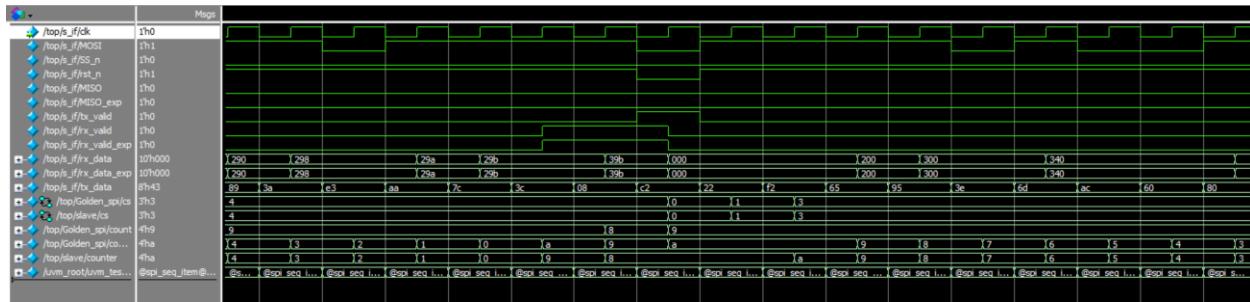
Name	Assertion Type	Language
/uvm_pkg::uvm_reg_map::do_write#(ublk#215181159#1731)immed_1735	Immediate	SVA
/uvm_pkg::uvm_reg_map::do_read#(ublk#215181159#1771)immed_1775	Immediate	SVA
/spi_main_seq_pkg::spi_main_seq::body#(ublk#267846023#22)immed_24	Immediate	SVA
+ /top/slave/assert_0	Concurrent	SVA
+ /top/slave/assert_1	Concurrent	SVA
+ /top/slave/assert_2	Concurrent	SVA
+ /top/slave/assert_3	Concurrent	SVA
+ /top/slave/binded_assertions/assert_0	Concurrent	SVA
+ /top/slave/binded_assertions/assert_1	Concurrent	SVA
+ /top/slave/binded_assertions/assert_2	Concurrent	SVA
+ /top/slave/binded_assertions/assert_3	Concurrent	SVA
+ /top/slave/binded_assertions/assert_4	Concurrent	SVA

2.23.5.Questasim Assertions Coverage

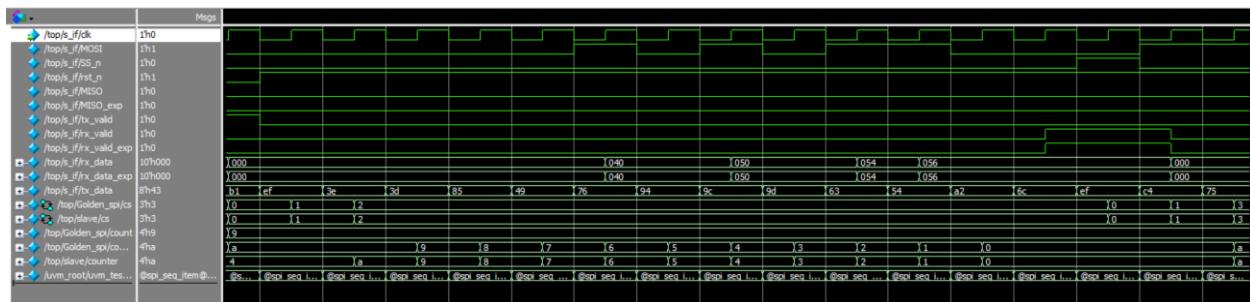
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight
/top/slave/cover __3	SVA	✓	Off	6	1	Unli...	1
/top/slave/cover __2	SVA	✓	Off	34	1	Unli...	1
/top/slave/cover __1	SVA	✓	Off	159	1	Unli...	1
/top/slave/cover __0	SVA	✓	Off	170	1	Unli...	1
/top/slave/binded_assertions/cover __4	SVA	✓	Off	104	1	Unli...	1
/top/slave/binded_assertions/cover __3	SVA	✓	Off	5	1	Unli...	1
/top/slave/binded_assertions/cover __2	SVA	✓	Off	4	1	Unli...	1
/top/slave/binded_assertions/cover __1	SVA	✓	Off	3	1	Unli...	1
/top/slave/binded_assertions/cover __0	SVA	✓	Off	3	1	Unli...	1

2.23.6.Questasim Waveform

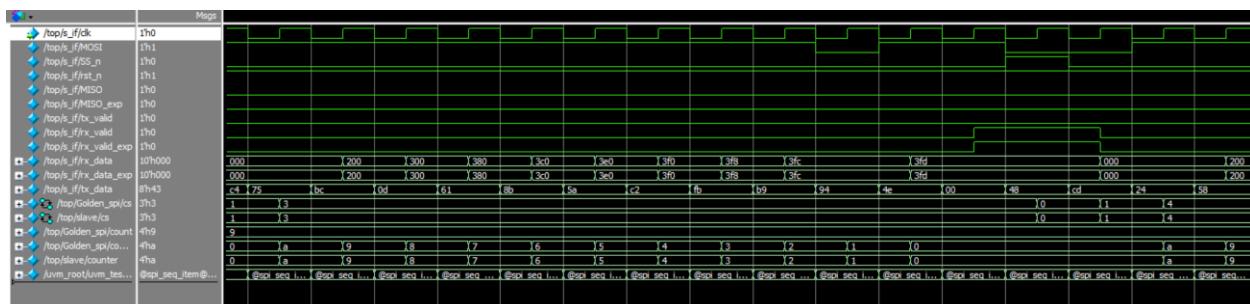
- Reset Sequence:



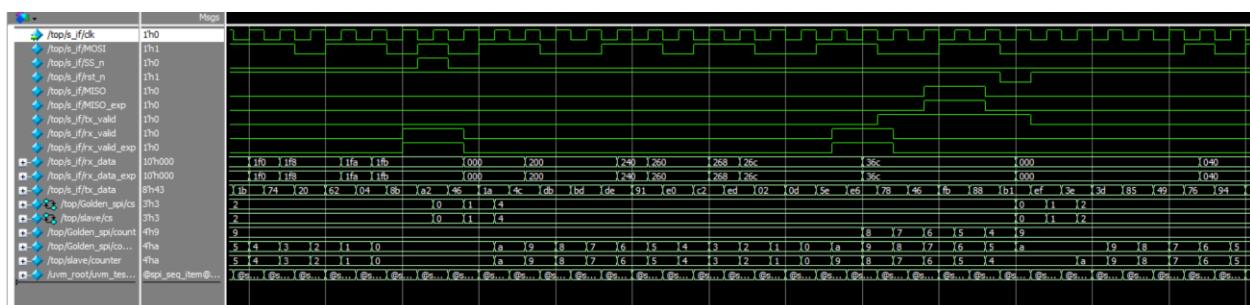
- Write Sequence:



- 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

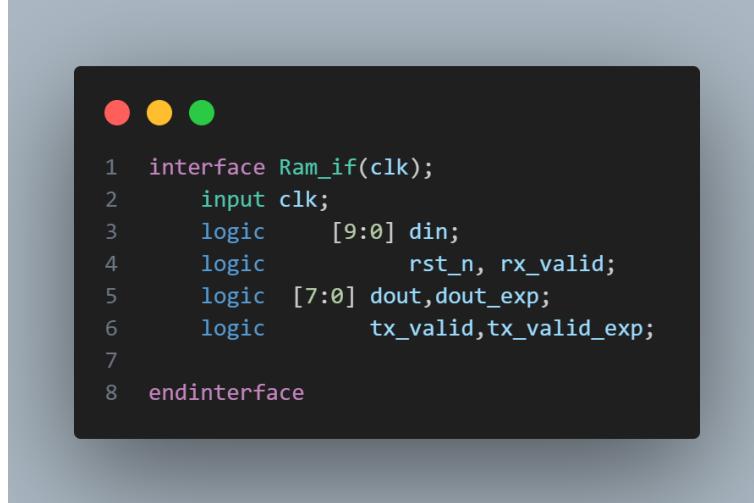
Assertions Requirements: (Add the assertions file and bind it in the top module)

- 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 Requirement 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	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.sv 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.sv 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 registered 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.sv 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.sv 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 registered 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.sv file.

3.3.RAM Interface



```

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

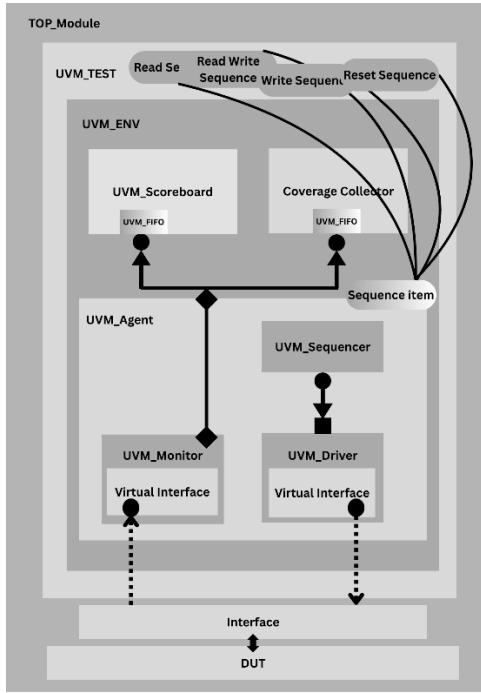
3.4.1.Design Bugs

- **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

```
● ● ●
1 module SPR#(parameter MEM_DEPTH=256,parameter ADDR_SIZE=8)(din,clk,rstn,rx_valid,tx_valid,dout);
2
3 input [9:0]din;
4 input clk,rstn,rx_valid;
5 output reg tx_valid;
6 output reg [7:0]dout;
7
8 reg [7:0] MEM[255:0];
9
10 reg [7:0]Wr_addr,Rd_addr; //internal address signal
11
12 always @(posedge clk) begin
13     if(!rstn) begin
14         dout<=8'b0;
15         Wr_addr<=8'b0;
16         Rd_addr<=8'b0;
17         tx_valid<=0;
18     end
19     else begin
20         if(rx_valid) begin
21             case({din[9:8]})
22                 2'b00: begin
23                     Wr_addr<=din[7:0];
24                     tx_valid<=0;
25                 end
26                 2'b01: begin
27                     MEM[Wr_addr]<=din[7:0];
28                     tx_valid<=0;
29                 end
30                 2'b10: begin
31                     Rd_addr<=din[7:0];
32                     tx_valid<=0;
33                 end
34                 2'b11: begin
35                     dout<=MEM[Rd_addr];
36                     tx_valid<=1;
37                 end
38             endcase
39         end
40     end
41 end
42 endmodule
43
```

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

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Object : Sequence Item
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_seq_item_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   class Ram_seq_item extends uvm_sequence_item;
10  `uvm_object_utils(Ram_seq_item)
11  //Constructor function
12  function new(string name="Ram_seq_item");
13    super.new(name);
14  endfunction
15
16  typedef enum {WRITE_ADD,WRITE_DATA,READ_ADD,READ_DATA} operations;
17
18  //Interface signals
19  operations old_op;
20  rand logic [9:0] din;
21  rand logic      rst_n, rx_valid;
22  logic [7:0] dout,dout_exp;
23  logic      tx_valid,tx_valid_exp;
24  //rand type signal;
25
26  constraint rst_constraints {rst_n dist{1:/90 , 0:/5};}
27  constraint rx_valid_constraints {rx_valid dist{1:/85 , 0:/15};}
28  constraint write_only_seq_constraints{
29    if(!rst_n) operations'(din[9:8]) inside {WRITE_ADD,WRITE_DATA};
30    else if(old_op==WRITE_ADD) operations'(din[9:8]) inside {WRITE_ADD,WRITE_DATA};
31    else if (old_op==READ_DATA) operations'(din[9:8]) inside {WRITE_ADD};
32  constraint read_only_seq_constraints {
33    if(!rst_n) operations'(din[9:8]) inside {READ_ADD,READ_DATA};
34    else if(old_op==READ_ADD) operations'(din[9:8]) inside {READ_ADD,READ_DATA};
35    else if (old_op==READ_DATA) operations'(din[9:8]) inside {READ_ADD};
36  constraint read_write_constraints {
37    if(old_op==WRITE_ADD) operations'(din[9:8]) inside {WRITE_ADD,WRITE_DATA};
38    if(old_op==WRITE_DATA) operations'(din[9:8]) dist {READ_ADD:/60 , WRITE_ADD:/40};
39    if(old_op==READ_ADD) operations'(din[9:8]) inside {READ_ADD,READ_DATA};
40    if(old_op==READ_DATA) operations'(din[9:8]) dist {WRITE_ADD:/60 , READ_ADD:/40};
41  }
42
43  function void post_randomize();
44    // if(!rst_n) old_op=READ_DATA;
45    old_op =operations'(din[9:8]);
46  endfunction
47  //Constrain Blocks
48
49  function string convert2string_stimulus();
50    return $sformatf("Error in the Ram : reset=%
51 ,din");
52  ,rx_valid;
53  ,rst_n,din,rx_valid);
54  ,din=din,rst_n,din,rx_valid,dout,tx_valid,dout_exp,tx_valid_exp);
55  ,rx_valid);
56  ,dout=dout;
57  endpackage
      dout_exp=dout_exp;
```

3.8.RAM Sequencer

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Sequencer
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_sqr_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import Ram_seq_item_pkg::*;
10
11 class Ram_sqr extends uvm_sequencer #(Ram_seq_item);
12   `uvm_component_utils(Ram_sqr)
13
14   function new(string name="Ram_sqr",uvm_component parent=null);
15     super.new(name,parent);
16   endfunction
17
18 endclass
19 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

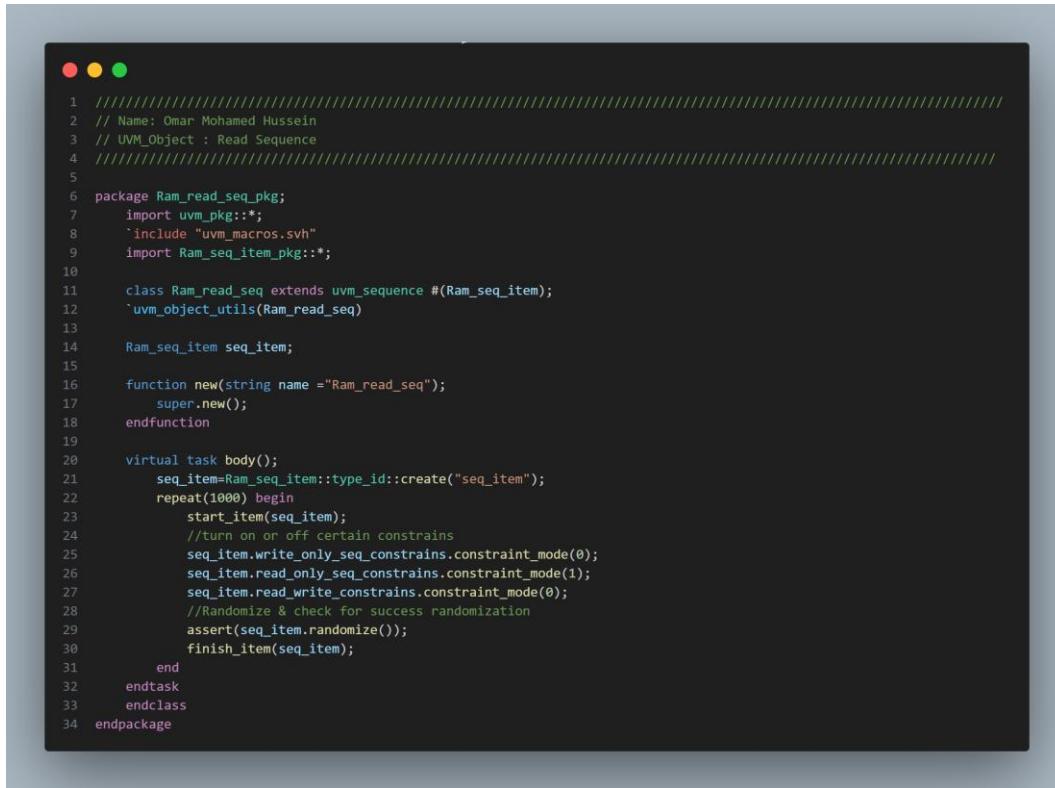
3.10.1.RAM Reset Sequence

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Object : Reset Sequence
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_reset_seq_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import Ram_seq_item_pkg::*;
10
11  class Ram_reset_seq extends uvm_sequence #(Ram_seq_item);
12    `uvm_object_utils(Ram_reset_seq)
13
14    Ram_seq_item seq_item;
15
16    function new(string name ="Ram_reset_seq");
17      super.new(name);
18    endfunction
19
20    task body;
21      seq_item=Ram_seq_item::type_id::create("seq_item");
22      start_item(seq_item);
23      seq_item.rst_n=0;
24      finish_item(seq_item);
25    endtask
26
27  endclass
28 endpackage
```

3.10.2.RAM Write Sequence

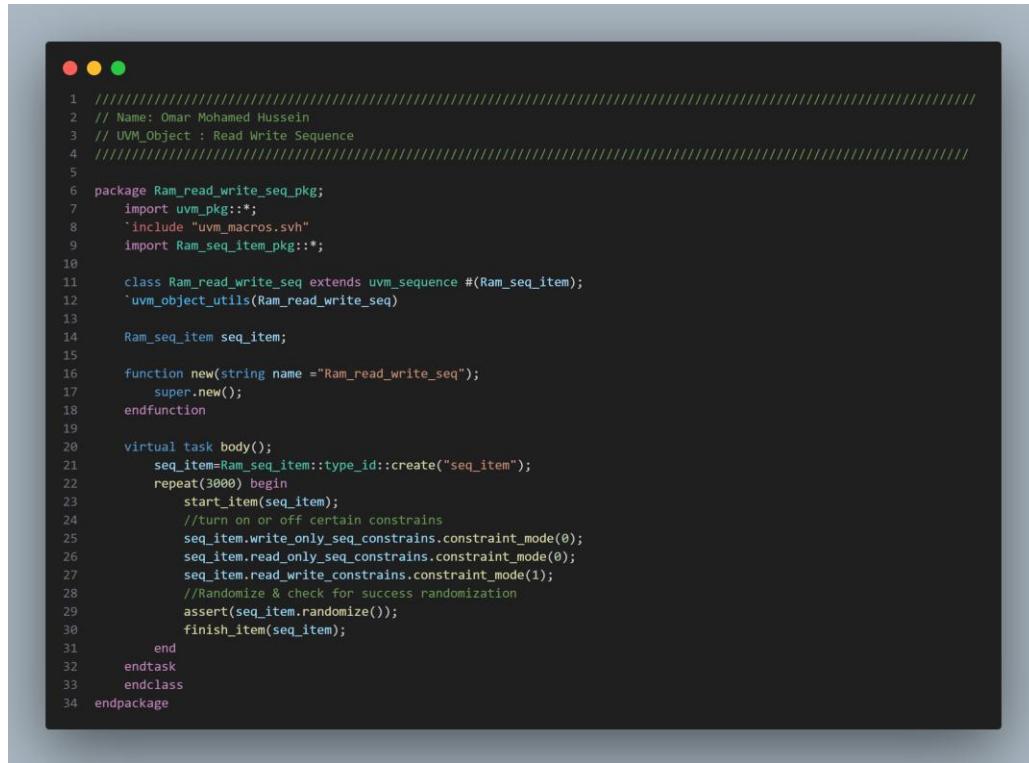
```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Object : Write Sequence
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_write_seq_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import Ram_seq_item_pkg::*;
10
11  class Ram_write_seq extends uvm_sequence #(Ram_seq_item);
12    `uvm_object_utils(Ram_write_seq)
13
14    Ram_seq_item seq_item;
15
16    function new(string name ="Ram_write_seq");
17      super.new();
18    endfunction
19
20    virtual task body();
21      seq_item=Ram_seq_item::type_id::create("seq_item");
22      repeat(1000) begin
23        seq_item.write_only_seq_constrains.constraint_mode(1);
24        seq_item.read_only_seq_constrains.constraint_mode(0);
25        seq_item.read_write_constrains.constraint_mode(0);
26        start_item(seq_item);
27        assert(seq_item.randomize());
28        finish_item(seq_item);
29      end
30    endtask
31  endclass
32 endpackage
```

3.10.3.RAM Read Sequence



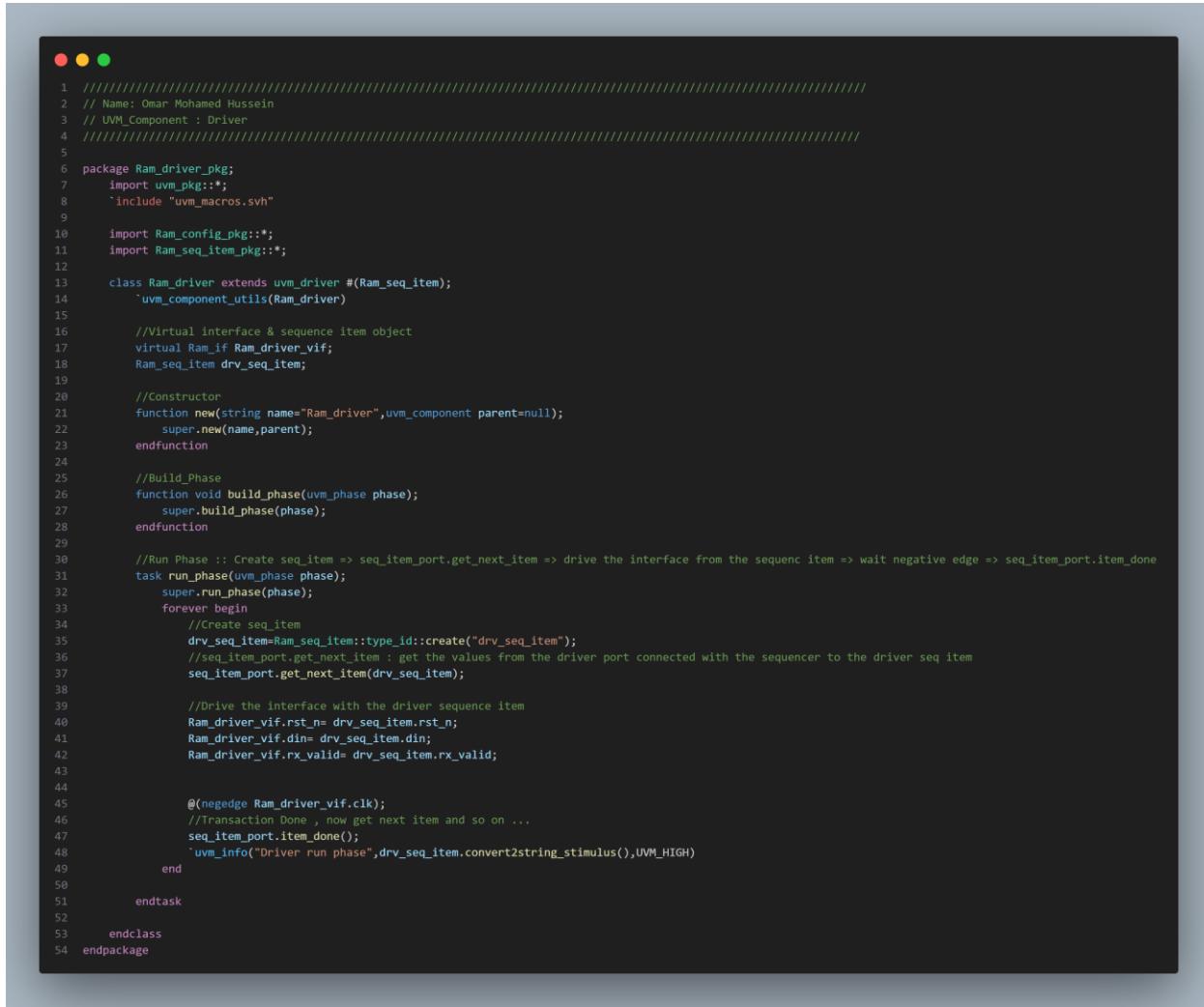
```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Object : Read Sequence
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_read_seq_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import Ram_seq_item_pkg::*;
10
11  class Ram_read_seq extends uvm_sequence #(Ram_seq_item);
12    `uvm_object_utils(Ram_read_seq)
13
14    Ram_seq_item seq_item;
15
16    function new(string name ="Ram_read_seq");
17      super.new();
18    endfunction
19
20    virtual task body();
21      seq_item=Ram_seq_item::type_id::create("seq_item");
22      repeat(1000) begin
23        start_item(seq_item);
24        //turn on or off certain constrains
25        seq_item.write_only_seq_constrains.constraint_mode(0);
26        seq_item.read_only_seq_constrains.constraint_mode(1);
27        seq_item.read_write_constrains.constraint_mode(0);
28        //Randomize & check for success randomization
29        assert(seq_item.randomize());
30        finish_item(seq_item);
31      end
32    endtask
33  endclass
34 endpackage
```

3.10.4.RAM Read Write Sequence



```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Object : Read Write Sequence
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_read_write_seq_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import Ram_seq_item_pkg::*;
10
11  class Ram_read_write_seq extends uvm_sequence #(Ram_seq_item);
12    `uvm_object_utils(Ram_read_write_seq)
13
14    Ram_seq_item seq_item;
15
16    function new(string name ="Ram_read_write_seq");
17      super.new();
18    endfunction
19
20    virtual task body();
21      seq_item=Ram_seq_item::type_id::create("seq_item");
22      repeat(3000) begin
23        start_item(seq_item);
24        //turn on or off certain constrains
25        seq_item.write_only_seq_constrains.constraint_mode(0);
26        seq_item.read_only_seq_constrains.constraint_mode(0);
27        seq_item.read_write_constrains.constraint_mode(1);
28        //Randomize & check for success randomization
29        assert(seq_item.randomize());
30        finish_item(seq_item);
31      end
32    endtask
33  endclass
34 endpackage
```

3.11.RAM Driver



```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM Component : Driver
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_driver_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9
10  import Ram_config_pkg::*;
11  import Ram_seq_item_pkg::*;
12
13 class Ram_driver extends uvm_driver #(Ram_seq_item);
14   `uvm_component_utils(Ram_driver)
15
16   //Virtual interface & sequence item object
17   virtual Ram_if Ram_driver_vif;
18   Ram_seq_item drv_seq_item;
19
20   //Constructor
21   function new(string name="Ram_driver",uvm_component parent=null);
22     super.new(name,parent);
23   endfunction
24
25   //Build_Phase
26   function void build_phase(uvm_phase phase);
27     super.build_phase(phase);
28   endfunction
29
30   //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
31   task run_phase(uvm_phase phase);
32     super.run_phase(phase);
33     forever begin
34       //Create seq_item
35       drv_seq_item=Ram_seq_item::type_id::create("drv_seq_item");
36       //seq_item_port.get_next_item : get the values from the driver port connected with the sequencer to the driver seq item
37       seq_item_port.get_next_item(drv_seq_item);
38
39       //Drive the interface with the driver sequence item
40       Ram_driver_vif.rst_n=drv_seq_item.rst_n;
41       Ram_driver_vif.din=drv_seq_item.din;
42       Ram_driver_vif.rx_valid=drv_seq_item.rx_valid;
43
44
45       @(negedge Ram_driver_vif.clk);
46       //Transaction Done , now get next item and so on ...
47       seq_item_port.item_done();
48       `uvm_info("Driver run phase",drv_seq_item.convert2string_stimulus(),UVM_HIGH)
49     end
50   endtask
51
52   endclass
53 endpackage
```

3.12.RAM Monitor

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Monitor
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_monitor_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import Ram_seq_item_pkg::*;
10  class Ram_monitor extends uvm_monitor ;
11    `uvm_component_utils(Ram_monitor)
12
13    //Creating Virtual interface to get the data from the interface
14    virtual Ram_if r_if;
15    //Creating monitor sequence item to pass the interface to it to be then sent to the analysis components
16    Ram_seq_item mon_seq_item;
17    //Creating monitor analysis port to broadcast the values to the analysis components
18    uvm_analysis_port #(Ram_seq_item) mon_ap;
19
20    //Constructor
21    function new(string name="Ram_monitor",uvm_component parent=null);
22      super.new(name,parent);
23    endfunction
24
25    //Build Phase
26    function void build_phase(uvm_phase phase);
27      super.build_phase(phase);
28      mon_ap=new("mon_ap",this);
29    endfunction
30
31    //Run Phase: samples the data at the negative edge and writes the analysis port with the monitor sequence item
32    task run_phase(uvm_phase phase);
33      super.run_phase(phase);
34      forever begin
35        mon_seq_item=Ram_seq_item::type_id::create("mon_seq_item");
36        @(negedge r_if.clk);
37
38        //Drive the monitor sequence item with the interface
39        mon_seq_item.din      =r_if.din;
40        mon_seq_item.rst_n    =r_if.rst_n;
41        mon_seq_item.rx_valid =r_if.rx_valid;
42        mon_seq_item.dout     =r_if.dout;
43        mon_seq_item.tx_valid =r_if.tx_valid;
44        mon_seq_item.tx_valid_exp=r_if.tx_valid_exp;
45        mon_seq_item.dout_exp =r_if.dout_exp;
46
47        //Broadcast the monitor sequence (interfance) to the analysis port
48        mon_ap.write(mon_seq_item);
49        `uvm_info("Monitor run phase",mon_seq_item.convert2string(),UVM_HIGH)
50      end
51    endtask
52  endclass
53 endpackage
```

3.13.RAM Agent

```
1 //////////////////////////////////////////////////////////////////
2 //Name: Omar Mohamed Hussein
3 //UVM_Component: Agent
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_agent_pkg;
7
8     import uvm_pkg::*;
9     `include "uvm_macros.svh"
10
11    import Ram_seq_item_pkg::*;
12    import Ram_sqr_pkg::*;
13    import Ram_driver_pkg::*;
14    import Ram_monitor_pkg::*;
15    import Ram_config_pkg::*;
16
17    class Ram_agent extends uvm_agent;
18        `uvm_component_utils(Ram_agent)
19
20        //creating the sequencer,driver,monitor, configuration object to connect the interface,agent port
21        Ram_sqr sqr;
22        Ram_driver drv;
23        Ram_monitor mon;
24        Ram_config_obj config_obj;
25        uvm_analysis_port #(Ram_seq_item) agt_ap;
26
27        //Constructor
28        function new(string name="Ram_agent",uvm_component parent=null);
29            super.new(name,parent);
30        endfunction
31
32        //Build Phase
33        function void build_phase(uvm_phase phase);
34            super.build_phase(phase);
35            //Connect the agent configuration object with the configuration object in the uvm config data base
36            if(!uvm_config_db #(Ram_config_obj)::get(this,"","CFG",config_obj)) begin
37                `uvm_fatal("agent build phase","unable to get the configuration object to the agent")
38            end
39
40            //Creating the sequencer,driver,monitor with the create() and the agt port with new()
41            sqr=Ram_sqr    ::type_id::create("sqr",this);
42            drv=Ram_driver ::type_id::create("drv",this);
43            mon=Ram_monitor::type_id::create("mon",this);
44            agt_ap=new("agt_ap",this);
45        endfunction
46
47        //connect phase
48        function void connect_phase(uvm_phase phase);
49            super.connect_phase(phase);
50            //Conncet the configuration object with the driver and the monitor ports
51            drv.Ram_driver_vif=config_obj.Ram_config_vif;
52            mon.r_if=config_obj.Ram_config_vif;
53            //Connect the driver port with the sequencer export
54            drv.seq_item_port.connect(sqr.seq_item_export);
55            //Connect the monitor analysis port with the agt port
56            mon.mon_ap.connect(agt_ap);
57        endfunction
58    endclass
59 endpackage
```

3.14.RAM Scoreboard

```
1 //////////////////////////////////////////////////////////////////
2 //Name: Omar Mohamed Hussein
3 //UVM_Component: Agent
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_scoreboard_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import Ram_seq_item_pkg::*;
10
11 class Ram_scoreboard extends uvm_scoreboard ;
12   `uvm_component_utils(Ram_scoreboard)
13
14   //Creating sequence item to get the data from fifo
15   Ram_seq_item seq_item;
16   //Creating analysis export to take values from the fifo
17   uvm_analysis_export #(Ram_seq_item) sb_export;
18   //Creating analysis fifo to take values from the agt port
19   uvm_tlm_analysis_fifo #(Ram_seq_item) sb_fifo;
20
21   int correct_count,error_count;
22
23   //Constructor
24   function new(string name="Ram_scoreboard",uvm_component parent =null);
25     super.new(name,parent);
26   endfunction
27
28   //Build phase : create the export analysis
29   function void build_phase(uvm_phase phase);
30     super.build_phase(phase);
31     sb_export=new("sb_export",this);
32     sb_fifo=new("sb_fifo",this);
33   endfunction
34
35   //Connect phase : connect the eb export with the fifo export
36   function void connect_phase (uvm_phase phase);
37     super.connect_phase(phase);
38     sb_export.connect(sb_fifo.analysis_export);
39   endfunction
40
41   //run Task : Compare the output with the Golden model
42   task run_phase(uvm_phase phase);
43     super.run_phase(phase);
44     forever begin
45       sb_fifo.get(seq_item);
46       if(seq_item.dout!=seq_item.dout_exp || seq_item.tx_valid!=seq_item.tx_valid_exp) begin
47         `uvm_error("scoreboard run phase",$sformatf("Design doesn't match the Golden model => %s",seq_item.convert2string))
48         error_count++;
49       end
50       else begin
51         correct_count++;
52       end
53     end
54   endtask
55
56   //Report Phase: report the testbench with how many failed and correct trials
57   function void report_phase(uvm_phase phase);
58     super.report_phase(phase);
59     `uvm_info("Scoreboard Report Phase",$sformatf("Testbench Completed , Correct count= %d",correct_count),UVM_LOW)
60   endfunction
61   `coendpackage(error_count,error_count),UVM_LOW
62 endpackage
```

3.15.RAM Cover Collector

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Cover Collector
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_cover_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   import Ram_seq_item_pkg::*;
10
11 class Ram_cover extends uvm_component;
12   `uvm_component_utils(Ram_cover)
13   uvm_analysis_export #(Ram_seq_item) cover_ap;
14   uvm_tlm_analysis_fifo #(Ram_seq_item) cover_fifo;
15   Ram_seq_item seq_item;
16
17
18 //Cover Groups
19   covergroup Cov_gp(ref Ram_seq_item seq_item);
20   //Normal Coverpoints
21   operations_cp: coverpoint seq_item.din[9:8] {
22     bins all_vals[]={[0:3]};
23     bins Write_address_data=(2'b00=>2'b01);
24     bins Read_address_data =(2'b10=>2'b11);
25     bins all_operations =(2'b00=>2'b01=>2'b10=>2'b11);
26   }
27   rx_valid_cp:coverpoint seq_item.rx_valid{
28     bins high={1};
29     bins low={0};
30   }
31 //Cross coverage
32   rx_valid_ops:cross operations_cp,rx_valid_cp {
33     bins allvals_rx_valid = binsof(operations_cp.all_vals) && binsof(rx_valid_cp.high);
34     bins read_data_tx_valid = binsof(operations_cp.all_vals) intersect {2'b11} && binsof(rx_valid_cp.high);
35   }
36 endgroup
37 function new(string name="Ram_cover",uvm_component parent=null);
38   super.new(name,parent);
39   Cov_gp=new(seq_item);
40 endfunction
41
42 function void build_phase(uvm_phase phase);
43   super.build_phase(phase);
44   cover_ap=new("cover_ap",this);
45   cover_fifo=new("cover_fifo",this);
46 endfunction
47
48 function void connect_phase(uvm_phase phase);
49   super.connect_phase(phase);
50   cover_ap.connect(cover_fifo.analysis_export);
51 endfunction
52
53 task run_phase(uvm_phase phase);
54   super.run_phase(phase);
55   forever begin
56     seq_item=Ram_seq_item::type_id::create("seq_item");
57     cover_fifo.get(seq_item);
58     Cov_gp.sample();
59   end
60   endtask
61
62 endclass
63 endpackage
```

3.16.RAM Environment

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Cover Collector
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_env_pkg;
7 import uvm_pkg::*;
8 `include "uvm_macros.svh"
9 import Ram_driver_pkg::*;
10 import Ram_scoreboard_pkg::*;
11 import Ram_agent_pkg::*;
12 import Ram_cover_pkg::*;
13   class Ram_env extends uvm_env;
14     `uvm_component_utils(Ram_env)
15     Ram_scoreboard sb;
16     Ram_agent agt;
17     Ram_cover cv;
18
19     function new(string name="Ram_env",uvm_component parent=null);
20       super.new(name,parent);
21     endfunction
22     function void build_phase(uvm_phase phase);
23       super.build_phase(phase);
24       sb=Ram_scoreboard::type_id::create("sb",this);
25       agt=Ram_agent::type_id::create("agt",this);
26       cv=Ram_cover::type_id::create("cv",this);
27     endfunction
28     function void connect_phase(uvm_phase phase);
29       super.connect_phase(phase);
30       agt.agt_ap.connect(sb.sb_export);
31       agt.agt_ap.connect(cv.cover_ap);
32     endfunction
33   endclass
34 endpackage
```

3.17.RAM Test

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Component : Test
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_test_pkg;
7 import uvm_pkg::*;
8 `include "uvm_macros.svh"
9 import Ram_env_pkg::*;
10 import Ram_config_pkg::*;
11 import Ram_reset_seq_pkg::*;
12 import Ram_write_seq_pkg::*;
13 import Ram_read_write_seq_pkg::*;
14 import Ram_read_seq_pkg::*;
15 class Ram_test extends uvm_test;
16     `uvm_component_utils(Ram_test)
17
18     Ram_env env;
19     Ram_config_obj Ram_config_obj_test;
20     Ram_reset_seq reset_seq;
21     Ram_write_seq write_seq;
22     Ram_read_seq read_seq;
23     Ram_read_write_seq write_read_seq;
24     function new(string name ="Ram_test",uvm_component parent=null);
25         super.new(name,parent);
26     endfunction
27
28     function void build_phase(uvm_phase phase);
29         super.build_phase(phase);
30         env=Ram_env::type_id::create("env",this);
31         Ram_config_obj_test=Ram_config_obj::type_id::create("Ram_config_obj_test");
32         reset_seq=Ram_reset_seq::type_id::create("reset_seq");
33         write_seq=Ram_write_seq::type_id::create("write_seq");
34         read_seq=Ram_read_seq::type_id::create("read_seq");
35         write_read_seq=Ram_read_write_seq::type_id::create("write_read_seq");
36
37         if(!uvm_config_db #(virtual Ram_if)::get(this,"","RAM",Ram_config_obj_test.Ram_config_vif))
38             `uvm_fatal("build phase in test","unable to get interface from the database into the configuration object")
39         uvm_config_db#(Ram_config_obj)::set(this,"*","CFG",Ram_config_obj_test);
40     endfunction
41
42     task run_phase(uvm_phase phase);
43         super.run_phase(phase);
44         phase.raise_objection(this);
45         `uvm_info("Test run phase","Reset Asserted",UVM_LOW)
46         reset_seq.start(env.agt.sqr);
47         `uvm_info("Test run phase","Reset Finished",UVM_LOW)
48
49         `uvm_info("Test run phase","Write sequence Asserted",UVM_LOW)
50         write_seq.start(env.agt.sqr);
51         `uvm_info("Test run phase","Write sequence Finished",UVM_LOW)
52
53         `uvm_info("Test run phase","Read sequence Asserted",UVM_LOW)
54         read_seq.start(env.agt.sqr);
55         `uvm_info("Test run phase","Read sequence Finished",UVM_LOW)
56
57         `uvm_info("Test run phase","Write Read sequence Asserted",UVM_LOW)
58         write_read_seq.start(env.agt.sqr);
59         `uvm_info("Test run phase","Write Read sequence Finished",UVM_LOW)
60         phase.drop_objection(this);
61     endtask
62     endclass
63 endpackage
```

3.18.RAM Top

```
● ● ●
1 import uvm_pkg::*;
2 `include "uvm_macros.svh"
3 import Ram_test_pkg::*;
4
5 module top();
6   // Clock generation
7   bit clk;
8   initial begin
9     clk=0;
10    forever begin
11      #10 clk=~clk;
12    end
13  end
14  // Instantiate the interface ,Golden Model and DUT
15  Ram_if r_if(clk);
16  RAM DUT(r_if.din,r_if.clk,r_if.rst_n,r_if.rx_valid,r_if.dout,r_if.tx_valid);
17  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);
18  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);
19  // run test using run_test task
20  initial begin
21    uvm_config_db #(virtual Ram_if)::set(null,"uvm_test_top","RAM",r_if);
22    run_test("Ram_test");
23  end
24 endmodule
```

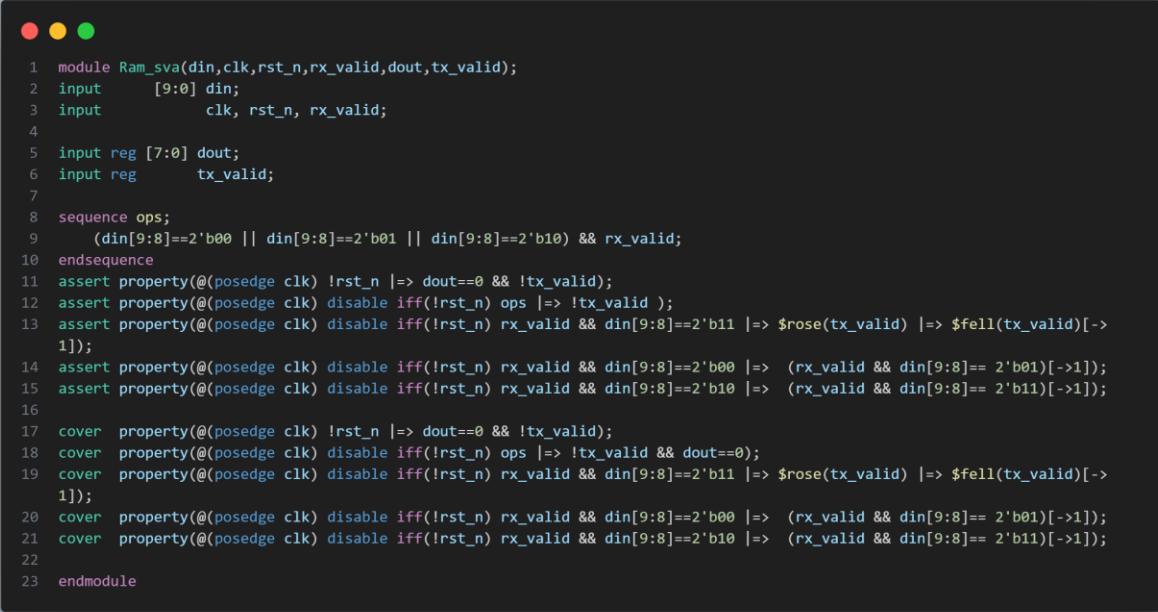
3.19.RAM Source File

```
● ● ●  
1 Ram_if.sv  
2 RAM.v  
3 down_counter.v  
4 Ram_sva.sv  
5 SPR.v  
6 Ram_config_obj.sv  
7 Ram_seq_item.sv  
8 Ram_reset_seq.sv  
9 Ram_write_seq.sv  
10 Ram_read_write_seq.sv  
11 Ram_read_seq.sv  
12 Ram_sqr.sv  
13 Ram_driver.sv  
14 Ram_monitor.sv  
15 Ram_agent.sv  
16 Ram_scoreboard.sv  
17 Ram_cover.sv  
18 Ram_env.sv  
19 Ram_test.sv  
20 Ram_top.sv
```

3.20.RAM DO File

```
● ● ●  
1 vlib work  
2 vlog -f src_files.list +cover -covercells  
3 vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover  
4 add wave /top/r_if/*  
5 run 0  
6 coverage save top.ucdb -onexit  
7 run -all
```

3.21.RAM Assertions



```

1  module Ram_sva(din,clk,rst_n,rx_valid,dout,tx_valid);
2  input      [9:0] din;
3  input      clk, rst_n, rx_valid;
4
5  input reg [7:0] dout;
6  input reg      tx_valid;
7
8  sequence ops;
9    (din[9:8]==2'b00 || din[9:8]==2'b01 || din[9:8]==2'b10) && rx_valid;
10 endsequence
11 assert property(@(posedge clk) !rst_n |=> dout==0 && !tx_valid);
12 assert property(@(posedge clk) disable iff(!rst_n) ops |=> !tx_valid );
13 assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b11 |=> $rose(tx_valid) |=> $fell(tx_valid)[-1]);
14 assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b00 |=> (rx_valid && din[9:8]== 2'b01)[-1]);
15 assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b10 |=> (rx_valid && din[9:8]== 2'b11)[-1]);
16
17 cover property(@(posedge clk) !rst_n |=> dout==0 && !tx_valid);
18 cover property(@(posedge clk) disable iff(!rst_n) ops |=> !tx_valid && dout==0);
19 cover property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b11 |=> $rose(tx_valid) |=> $fell(tx_valid)[-1]);
20 cover property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b00 |=> (rx_valid && din[9:8]== 2'b01)[-1]);
21 cover property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b10 |=> (rx_valid && din[9:8]== 2'b11)[-1]);
22
23 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 Test Phase *****
# 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) @ 200201: uvm_test_top [Test run phase] Write sequence Finished
# UVM_INFO Ram_test.sv(48) @ 200201: uvm_test_top [Test run phase] Read sequence Asserted
# UVM_INFO Ram_test.sv(50) @ 400201: uvm_test_top [Test run phase] Read sequence Finished
# UVM_INFO Ram_test.sv(52) @ 400201: 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.m [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
# [Questasim] 2
# [BIST] 1
# [Scoreboard Report Phase] 1
# [TEST_DONE] 1
# [Test run phase] 8
# ** Note: cfinish : 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:

Statements - by instance (/top/DUT)

Statement	✓	✗	E
RAM.v	✓	✓	✓
13 always @(posedge clk) begin	✓		
15 dout <= 0;	✓		
16 tx_valid <= 0;	✓		
17 Rd_Addr <= 0;	✓		
18 Wr_Addr <= 0;	✓		
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;	Xs		
31 tx_valid<=1;	✓		
34 tx_valid<=0;	✓		
Ram_top.sv			

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

Branch Coverage:

Branches - by instance (/top/DUT)

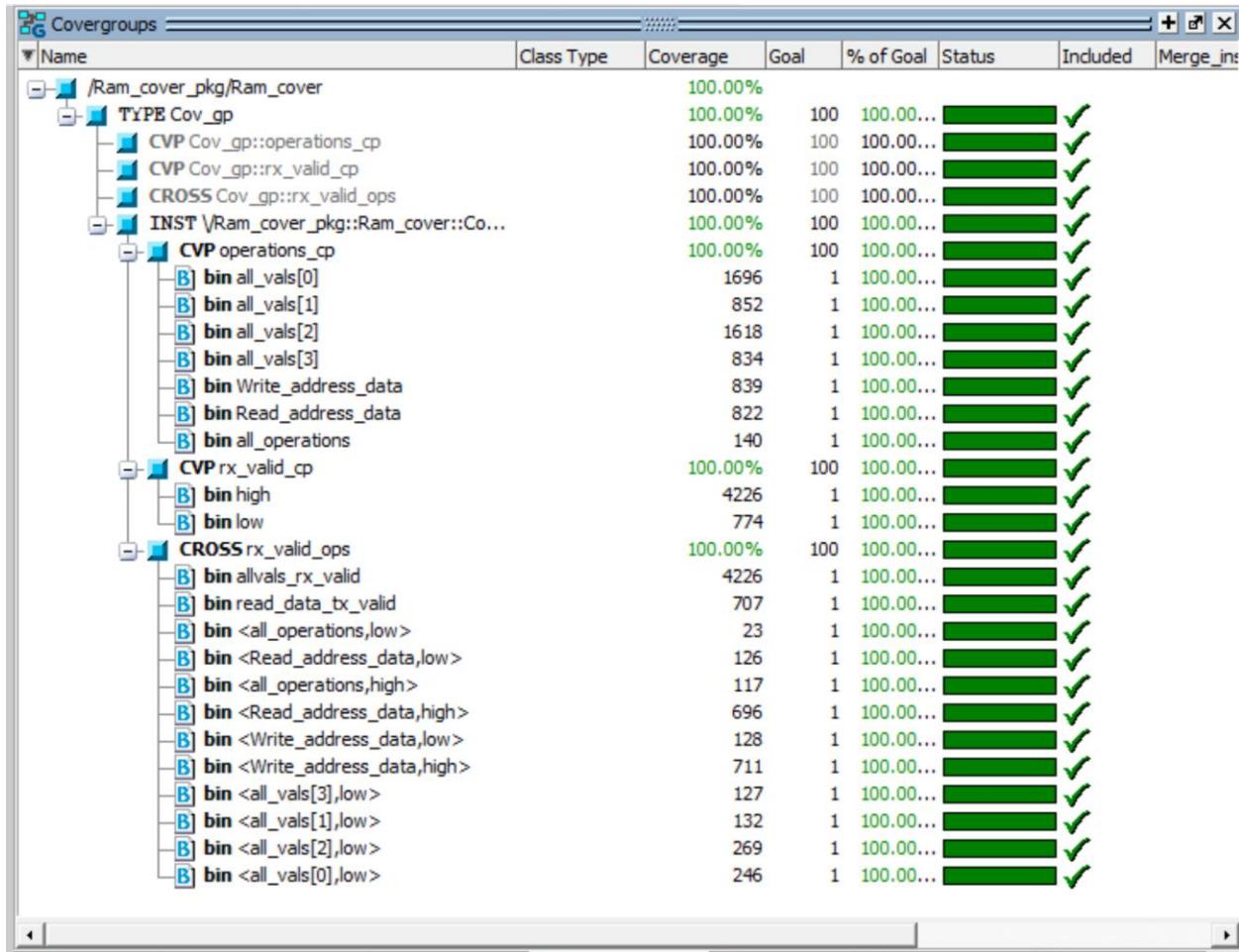
Branch	✓	✗	E
RAM.v	✓	✓	✓
14 if (~rst_n) begin	✓		
20 else begin //Design Error: missing begin and end	✓		
21 if (rx_valid) begin	✓		
23 2'b00 : Wr_Addr <= din[7:0];	✓		
24 2'b01 : MEM[Wr_Addr] <= din[7:0];	✓		
25 2'b10 : Rd_Addr <= din[7:0];	✓		
26 2'b11 : dout <= MEM[Rd_Addr]; //Design Error : should be of Rd_addr	✓		
27 default : dout <= 0;	E		
30 if(din[9] && din[8] && rx_valid) begin	✓		
33 else if ((!din[9] !din[8]) && rx_valid) begin	✓		
Ram_top.sv			

- 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

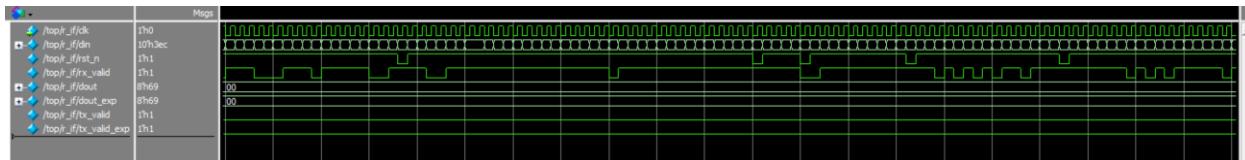
Name	Assertion Type	Language
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed_1735	Immediate	SVA
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed_1775	Immediate	SVA
/Ram_read_seq_pkg::Ram_read_seq::body/#ublk#256480583#17/immed_24	Immediate	SVA
/Ram_read_write_seq_pkg::Ram_read_write_seq::body/#ublk#31014951#17/immed_24	Immediate	SVA
/Ram_write_seq_pkg::Ram_write_seq::body/#ublk#120540439#17/immed_22	Immediate	SVA
+ /top/DUT/binded_assertions/assert_0	Concurrent	SVA
+ /top/DUT/binded_assertions/assert_1	Concurrent	SVA
+ /top/DUT/binded_assertions/assert_2	Concurrent	SVA
+ /top/DUT/binded_assertions/assert_3	Concurrent	SVA
+ /top/DUT/binded_assertions/assert_4	Concurrent	SVA

3.22.5.Questasim Assertions Coverage

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight
/top/DUT/binded_assertions/cover_4	SVA	✓	Off	1134	1	Unli...	1
/top/DUT/binded_assertions/cover_3	SVA	✓	Off	1218	1	Unli...	1
/top/DUT/binded_assertions/cover_2	SVA	✓	Off	555	1	Unli...	1
/top/DUT/binded_assertions/cover_1	SVA	✓	Off	1247	1	Unli...	1
/top/DUT/binded_assertions/cover_0	SVA	✓	Off	259	1	Unli...	1

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

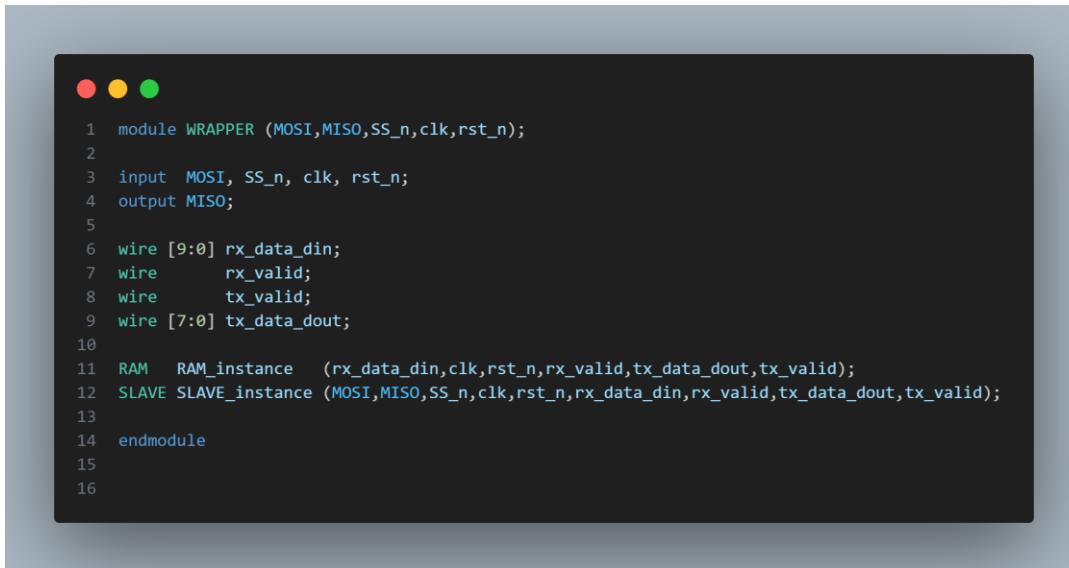
Sequences Requirements:

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

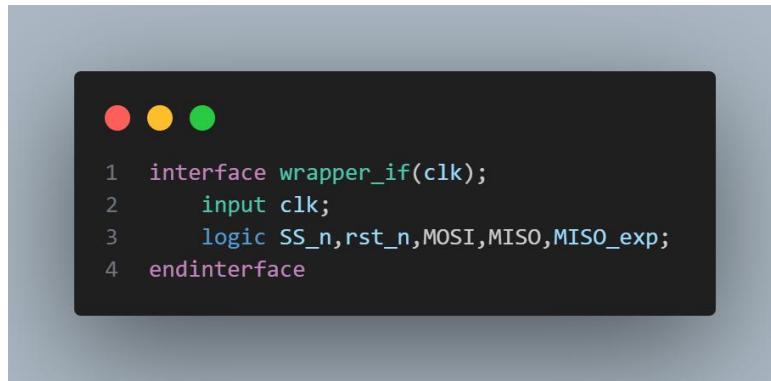


```

1 module WRAPPER (MOSI,MISO,SS_n,clk,rst_n);
2
3 input MOSI, SS_n, clk, rst_n;
4 output MISO;
5
6 wire [9:0] rx_data_din;
7 wire rx_valid;
8 wire tx_valid;
9 wire [7:0] tx_data_dout;
10
11 RAM RAM_instance (rx_data_din,clk,rst_n,rx_valid,tx_data_dout,tx_valid);
12 SLAVE SLAVE_instance (MOSI,MISO,SS_n,clk,rst_n,rx_data_din,rx_valid,tx_data_dout,tx_valid);
13
14 endmodule
15
16

```

4.3. Wrapper Interface



```

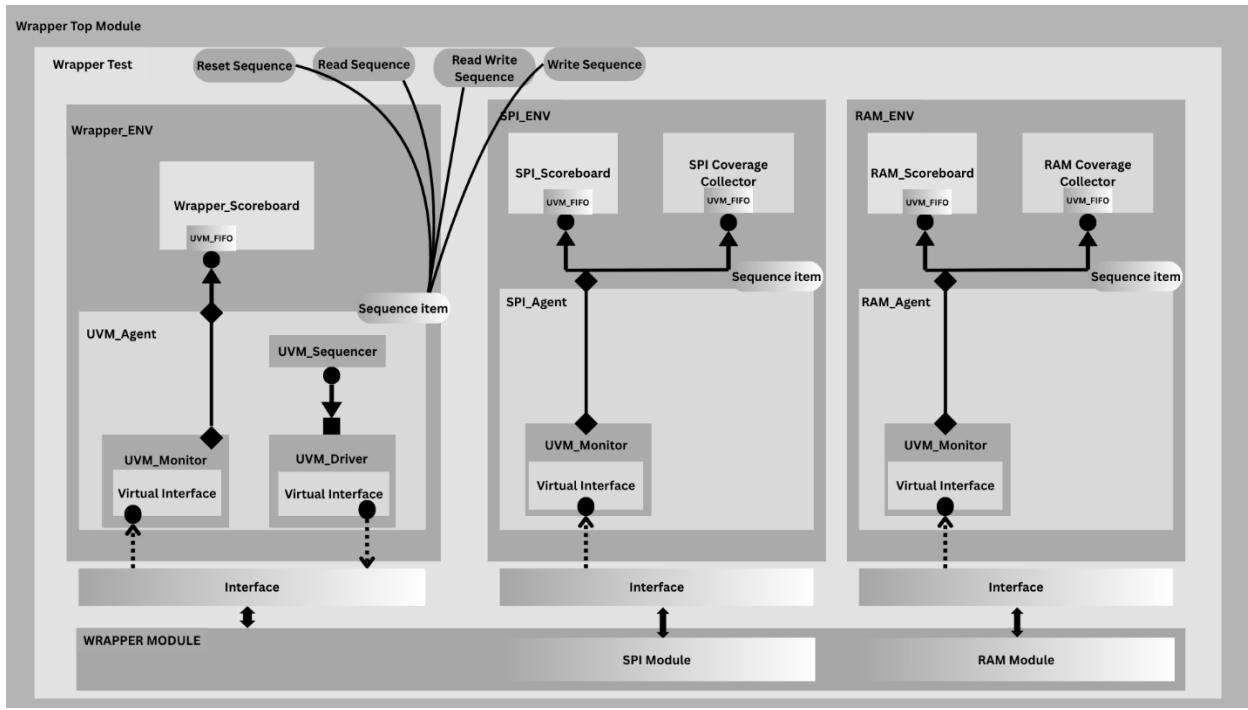
1 interface wrapper_if(clk);
2     input clk;
3     logic SS_n,rst_n,MOSI,MISO,MISO_exp;
4 endinterface

```

4.4. Verification Plan

Label	Design Requirement Description	Stimulus generation	Functional coverage	Functionality check
reset check	when the reset signal is activated , the MOSI,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 spl.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 for all cases except for the READ_DATA the SS_n would be low for 23 cycle .	In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the spl.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 for all cases except for the READ_DATA the SS_n would be low for 23 cycle .	In the scoreboard , comparing the outputs with the expected of a Golden model , also using assertions in the spl.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_com 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 , also using assertions in the spl.sva file.
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_com 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 READ_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_com 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 READ_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_com 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.
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 registered 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 registered 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

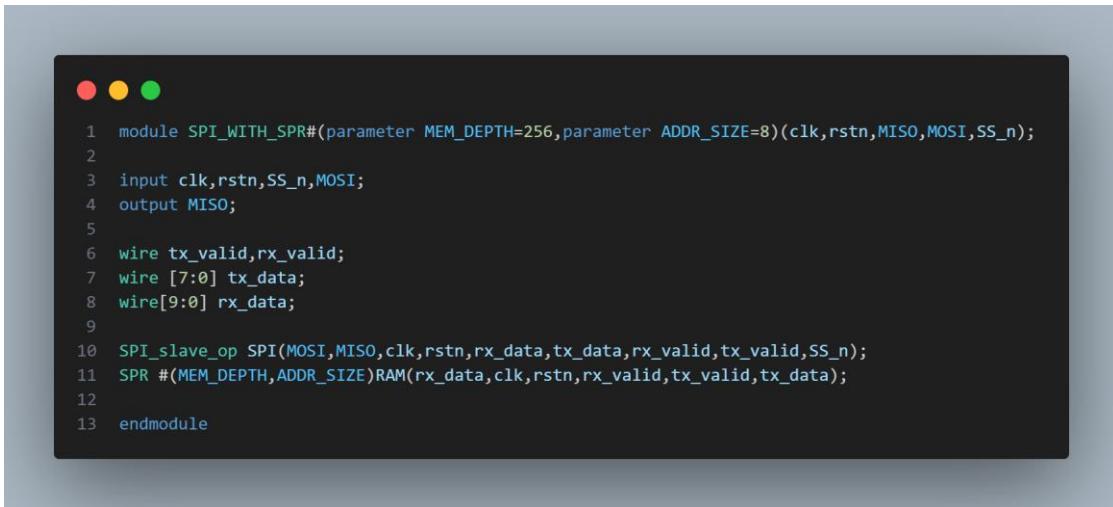


Verification Flow:

- 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

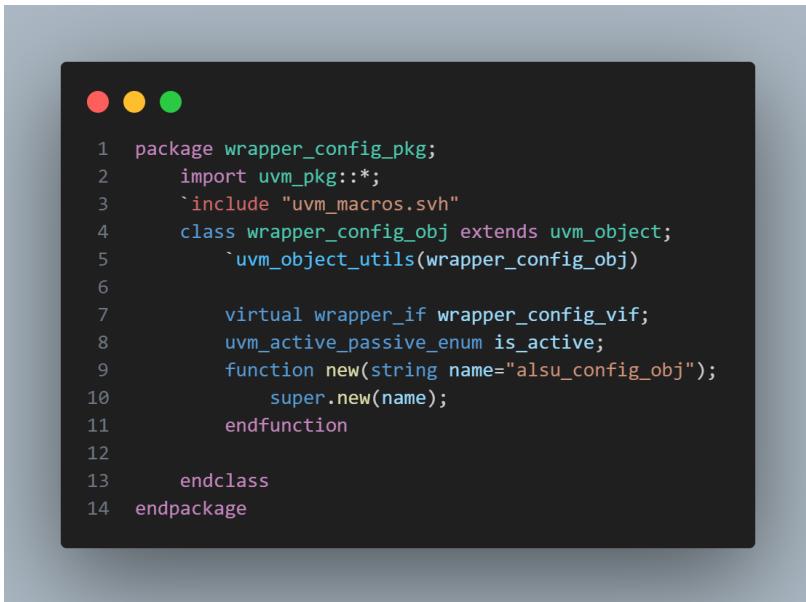
```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Object : Wrapper Sequence Item
4 //////////////////////////////////////////////////////////////////
5
6 package wrapper_seq_item;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9
10 class wrapper_seq_item extends uvm_sequence_item;
11   `uvm_object_utils(wrapper_seq_item)
12
13 //Constructor function
14 function new(string name="wrapper_seq_item");
15   super.new(name);
16 endfunction
17
18 //Type define
19 typedef enum {IDLE,CHK_CMD,WRITE,READ_ADD_F,READ_DATA_F} my_FSM;
20 typedef enum {WRITE_ADD,WRITE_DATA,READ_ADD,READ_DATA} operations;
21
22 //inputs rand signals
23 rand logic [9:0] din;
24 rand logic MOSI, rst_n, SS_n;
25 //output signals
26 logic MISO,MISO_exp;
27 //Internal signals and sequence item counters
28 rand logic [10:0] valid_comb;
29 logic [10:0] old_valid_comb;
30 int MOSI_count,i,read_add_data;
31 my_FSM old_cs;
32 rand my_FSM cs;
33 operations old_op;
34
35 constraint rst_constraints {rst_n dist{1:/95 , 0:/5};}
36 constraint SS_n_constraints {
37   if (old_cs != READ_DATA_F ) {
38     (MOSI_count == 13) -> SS_n == 1;
39     (MOSI_count != 13) -> SS_n == 0;
40   }
41   else if(old_cs==READ_DATA_F) {
42     (MOSI_count == 23) -> SS_n == 1;
43     (MOSI_count != 23) -> SS_n == 0;
44   }
45 }
46
47 //Creating a FSM Environment inside the Sequence item
48 constraint cs_constraints{if(!rst_n || SS_n) {cs==IDLE;}
49   else if(old_cs==IDLE && !SS_n) {cs==CHK_CMD;}
50   else if(old_cs==CHK_CMD && !SS_n && !MOSI) {cs==WRITE;}
51   else if(old_cs==CHK_CMD && !SS_n && MOSI && read_add_data==0) {cs==READ_ADD_F;}
52   else if(old_cs==CHK_CMD && !SS_n && MOSI && read_add_data==1) {cs==READ_DATA_F;}
53   else if((old_cs==WRITE || old_cs==READ_ADD_F) && MOSI_count==13 ) {cs==IDLE;}
54   else if((old_cs==WRITE) && MOSI_count!=13 ) {cs==WRITE;}
55   else if((old_cs==READ_ADD_F) && MOSI_count!=13 ) {cs==READ_ADD_F;}
56   else if((old_cs==READ_DATA_F) && MOSI_count==23 ) {cs==IDLE;}
57   else if((old_cs==READ_DATA_F) && MOSI_count!=23 ) {cs==READ_DATA_F;}
58 }
59 //Randomizing a 11 bit bus forcing it to start with 3 valid bits, each Randomization will be at the start of each sequence
60 constraint valid_comb_constraints{
61   if(!rst_n || MOSI_count==0) valid_comb[10:8] inside{3'b000,3'b001,3'b110,3'b111};
62   else valid_comb==old_valid_comb;
63 }
```

```

1 //Constraining the MOSI bits to take the valid Combination of bits
2 constraint MOSI_constrains {if(!SS_n && rst_n ) MOSI == valid_comb[10-i] ;}
3
4 //Write only sequence forcing the operations to Write address or data only
5 constraint write_only_seq_constraints{
6     if(!rst_n) (valid_comb[9:8]) inside {WRITE_ADD,WRITE_DATA};
7     else if(old_op==WRITE_ADD && MOSI_count==0) (valid_comb[9:8]) inside {WRITE_ADD,WRITE_DATA};
8     else if (old_op==WRITE_DATA && MOSI_count==0) (valid_comb[9:8]) inside {WRITE_ADD}; //Only Read address allowes
9
10 //Read only sequence forcing the operations to read address or data only
11 constraint read_only_seq_constraints {
12     if(!rst_n) (valid_comb[9:8]) inside {READ_DATA}; //To Force the design when MOSI_count==0 to go to Read_ADDR
13     else if(old_op==READ_ADD && MOSI_count==0) operations'( valid_comb[9:8]) inside {READ_DATA}; //Only Read Data allowed
14     else if(old_op==READ_DATA && MOSI_count==0) operations'( valid_comb[9:8]) inside {READ_ADD}; //Only Read Data allowed
15
16 constraint read_write_constraints {
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 $sformatf(`rst=
60 ,SS_`andfunction
61 ,MOSI`andfunction
62 ",rst_n,$S`andfunction
63 ,SS_`andfunction
64 ,MOSI`endclass
65 `M@D@B`okage
66 ,MISO_`exp=

```

4.8. Wrapper Configuration Object



```
● ● ●

1 package wrapper_config_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   class wrapper_config_obj extends uvm_object;
5     `uvm_object_utils(wrapper_config_obj)
6
7     virtual wrapper_if wrapper_config_vif;
8     uvm_active_passive_enum is_active;
9     function new(string name="alsu_config_obj");
10       super.new(name);
11     endfunction
12
13   endclass
14 endpackage
```

4.9. Wrapper Sequences

4.9.1. Reset Sequence



```
1 package wrapper_reset_seq_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import wrapper_seq_item::*;
5
6   class wrapper_reset_seq extends uvm_sequence #(wrapper_seq_item);
7     `uvm_object_utils(wrapper_reset_seq)
8
9     wrapper_seq_item seq_item;
10
11    function new(string name ="wrapper_reset_seq");
12      super.new(name);
13    endfunction
14
15    task body;
16      seq_item=wrapper_seq_item::type_id::create("seq_item");
17      start_item(seq_item);
18      seq_item.rst_n=0;
19      finish_item(seq_item);
20    endtask
21
22  endclass
23 endpackage
```

4.9.2. Write Only Sequence



```
1 package wrapper_write_seq_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import wrapper_seq_item::*;
5
6   class wrapper_write_seq extends uvm_sequence #(wrapper_seq_item);
7     `uvm_object_utils(wrapper_write_seq)
8
9     wrapper_seq_item seq_item;
10
11    function new(string name ="wrapper_write_seq");
12      super.new();
13    endfunction
14
15    virtual task body();
16      seq_item=wrapper_seq_item::type_id::create("seq_item");
17      seq_item.write_only_seq_constrains.constraint_mode(1);
18      seq_item.read_only_seq_constrains.constraint_mode(0);
19      seq_item.read_write_constrains.constraint_mode(0);
20      repeat(500) begin
21        start_item(seq_item);
22        assert(seq_item.randomize());
23        finish_item(seq_item);
24      end
25    endtask
26  endclass
27 endpackage
```

4.9.3.Read Only Sequence



```
1 package wrapper_read_seq_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import wrapper_seq_item::*;
5
6   class wrapper_read_seq extends uvm_sequence #(wrapper_seq_item);
7     `uvm_object_utils(wrapper_read_seq)
8
9     wrapper_seq_item seq_item;
10
11    function new(string name ="wrapper_read_seq");
12      super.new();
13    endfunction
14
15    virtual task body();
16      seq_item=wrapper_seq_item::type_id::create("seq_item");
17      seq_item.write_only_seq_constrains.constraint_mode(0);
18      seq_item.read_only_seq_constrains.constraint_mode(1);
19      seq_item.read_write_constrains.constraint_mode(0);
20      repeat(1000) begin
21        start_item(seq_item);
22        assert(seq_item.randomize());
23        finish_item(seq_item);
24      end
25    endtask
26  endclass
27 endpackage
```

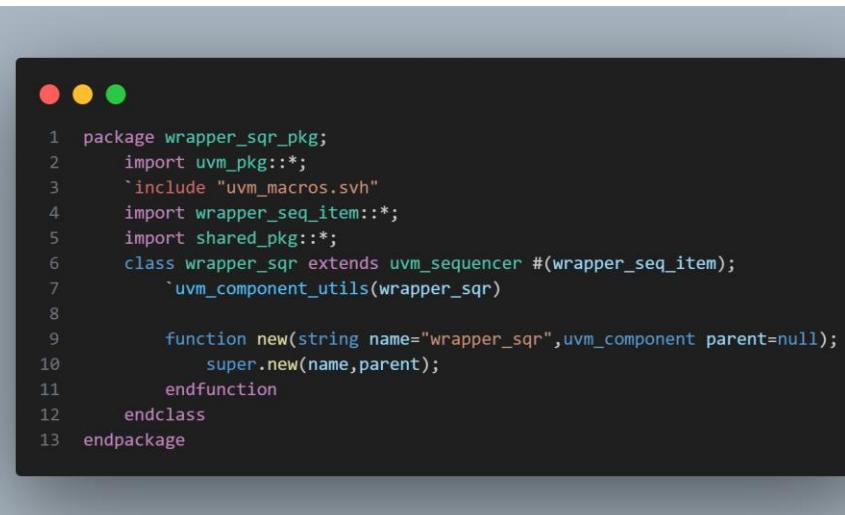
4.9.4. Read Write Sequence



```
● ● ●

1 package wrapper_read_write_seq_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import wrapper_seq_item::*;
5
6   class wrapper_read_write_seq extends uvm_sequence #(wrapper_seq_item);
7     `uvm_object_utils(wrapper_read_write_seq)
8
9     wrapper_seq_item seq_item;
10
11    function new(string name ="wrapper_read_write_seq");
12      super.new();
13    endfunction
14
15    virtual task body();
16      seq_item=wrapper_seq_item::type_id::create("seq_item");
17      seq_item.write_only_seq_constrains.constraint_mode(0);
18      seq_item.read_only_seq_constrains.constraint_mode(0);
19      seq_item.read_write_constrains.constraint_mode(1);
20      repeat(500) begin
21        start_item(seq_item);
22        assert(seq_item.randomize());
23        finish_item(seq_item);
24      end
25    endtask
26  endclass
27 endpackage
```

4.10. Wrapper sequencer



```
● ● ●

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

4.11. Wrapper Driver

```
1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM Component : Driver
4 //////////////////////////////////////////////////////////////////
5
6 package wrapper_driver_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9
10  import wrapper_config_pkg::*;
11  import wrapper_seq_item::*;
12
13 class wrapper_driver extends uvm_driver #(wrapper_seq_item);
14   `uvm_component_utils(wrapper_driver)
15
16   //Virtual interface & sequence item object
17   virtual wrapper_if wrapper_driver_vif;
18   wrapper_seq_item drv_seq_item;
19
20   //Constructor
21   function new(string name="wrapper_driver",uvm_component parent=null);
22     super.new(name,parent);
23   endfunction
24
25   //Build_Phase
26   function void build_phase(uvm_phase phase);
27     super.build_phase(phase);
28   endfunction
29
30   //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
31   task run_phase(uvm_phase phase);
32     super.run_phase(phase);
33     forever begin
34       //Create seq_item
35       drv_seq_item=wrapper_seq_item::type_id::create("drv_seq_item");
36       //seq_item_port.get_next_item : get the values from the driver port connected with the sequencer to the driver seq item
37       seq_item_port.get_next_item(drv_seq_item);
38
39       //Drive the interface with the driver sequence item
40       wrapper_driver_vif.rst_n= drv_seq_item.rst_n;
41       wrapper_driver_vif.SS_n= drv_seq_item.SS_n;
42       wrapper_driver_vif.MOSI= drv_seq_item.MOSI;
43
44       @(negedge wrapper_driver_vif.clk);
45       //Transaction done , now get next item and so on ...
46       seq_item_port.item_done();
47       uvm_info("Driver run phase",drv_seq_item.convert2string_stimulus(),UVM_HIGH)
48     end
49   endtask
50 endclass
51 endpackage
```

4.12. Wrapper Monitor

```
1 package wrapper_monitor_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import wrapper_seq_item::*;
5   import shared_pkg::*;
6   class wrapper_monitor extends uvm_monitor ;
7     `uvm_component_utils(wrapper_monitor)
8
9     //Creating Virtual interface to get the data from the interface
10    virtual wrapper_if w_if;
11    //Creating monitor sequence item to pass the interface to it to be then sent to the analysis components
12    wrapper_seq_item mon_seq_item;
13    //Creating monitor analysis port to broadcast the values to the analysis components
14    uvm_analysis_port #(wrapper_seq_item) mon_ap;
15
16    //Constructor
17    function new(string name="wrapper_monitor",uvm_component parent=null);
18      super.new(name,parent);
19    endfunction
20
21    //Build Phase
22    function void build_phase(uvm_phase phase);
23      super.build_phase(phase);
24      mon_ap=new("mon_ap",this);
25    endfunction
26
27    //Run Phase: samples the data at the negative edge and writes the analysis port with the monitor sequence item
28    task run_phase(uvm_phase phase);
29      super.run_phase(phase);
30      forever begin
31        mon_seq_item=wrapper_seq_item::type_id::create("mon_seq_item");
32        @(negedge w_if.clk);
33        mon_seq_item.rst_n=w_if.rst_n;
34        mon_seq_item.SS_n=w_if.SS_n;
35        mon_seq_item.MOSI=w_if.MOSI;
36        mon_seq_item.MISO=w_if.MISO;
37        mon_seq_item.MISO_exp=w_if.MISO_exp;
38        //Drive the monitor sequence item with the interface
39
40        //Broadcast the monitor sequence (interfance) to the analysis port
41        mon_ap.write(mon_seq_item);
42        `uvm_info("Monitor run phase",mon_seq_item.convert2string(),UVM_HIGH)
43      end
44    endtask
45  endclass
46 endpackage
```

4.13. Wrapper Agent

```
1 //////////////////////////////////////////////////////////////////
2 //Name: Omar Mohamed Hussein
3 //UVM_Component:Agent Package
4 //////////////////////////////////////////////////////////////////
5 package wrapper_agent_pkg;
6
7     import uvm_pkg::*;
8     `include "uvm_macros.svh"
9
10    import wrapper_seq_item::*;
11    import wrapper_sqr_pkg::*;
12    import wrapper_driver_pkg::*;
13    import wrapper_monitor_pkg::*;
14    import wrapper_config_pkg::*;
15
16    class wrapper_agent extends uvm_agent;
17        `uvm_component_utils(wrapper_agent)
18
19        //creating the sequencer,driver,monitor, configuration object to connect the interface,agent port
20        wrapper_sqr sqr;
21        wrapper_driver drv;
22        wrapper_monitor mon;
23        wrapper_config_obj config_obj;
24        uvm_analysis_port #(wrapper_seq_item) agt_ap;
25
26        //Constructor
27        function new(string name="wrapper_agent",uvm_component parent=null);
28            super.new(name,parent);
29        endfunction
30
31        //Build Phase
32        function void build_phase(uvm_phase phase);
33            super.build_phase(phase);
34            //Connect the agent configuration object with the configuration object in the uvm config data base
35            if(!uvm_config_db #(wrapper_config_obj)::get(this,"","CFG_WRAPPER",config_obj)) begin
36                `uvm_fatal("agent build phase","unable to get the configuration object to the agent")
37            end
38
39            //Creating the sequencer,driver,monitor with the create() and the agt port with new()
40            sqr=wrapper_sqr ::type_id::create("sqr",this);
41            drv=wrapper_driver ::type_id::create("drv",this);
42            mon=wrapper_monitor::type_id::create("mon",this);
43            agt_ap=new("agt_ap",this);
44        endfunction
45
46        //connect phase
47        function void connect_phase(uvm_phase phase);
48            super.connect_phase(phase);
49            //Connet the configuration object with the driver and the monitor ports
50            drv.wrapper_driver_vif=config_obj.wrapper_config_vif;
51            mon.w_if=config_obj.wrapper_config_vif;
52            //Connect the driver port with the sequencer export
53            drv.seq_item_port.connect(sqr.seq_item_export);
54            //Connect the monitor analysis port with the agt port
55            mon.mon_ap.connect(agt_ap);
56        endfunction
57    endclass
58 endpackage
```

4.14. Wrapper Scoreboard

```
1 package wrapper_scoreboard_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import wrapper_seq_item::*;
5   import shared_pkg::*;
6
7   class wrapper_scoreboard extends uvm_scoreboard ;
8     `uvm_component_utils(wrapper_scoreboard)
9
10    //Creating sequence item to get the data from fifo
11    wrapper_seq_item seq_item;
12    //Creating analysis export to take values from the fifo
13    uvm_analysis_export #(wrapper_seq_item) sb_export;
14    //Creating analysis fifo to take values from the agt port
15    uvm_tlm_analysis_fifo #(wrapper_seq_item) sb_fifo;
16
17    int correct_count,error_count;
18
19    //Constructor
20    function new(string name="wrapper_scoreboard",uvm_component parent =null);
21      super.new(name,parent);
22    endfunction
23
24    //Build phase : create the export analysis
25    function void build_phase(uvm_phase phase);
26      super.build_phase(phase);
27      sb_export=new("sb_export",this);
28      sb_fifo=new("sb_fifo",this);
29    endfunction
30
31    //Connect phase : connect the eb export with the fifo export
32    function void connect_phase (uvm_phase phase);
33      super.connect_phase(phase);
34      sb_export.connect(sb_fifo.analysis_export);
35    endfunction
36
37    //run Task : Compare the output with the Golden model
38    task run_phase(uvm_phase phase);
39      super.run_phase(phase);
40      forever begin
41        sb_fifo.get(seq_item);
42        if(seq_item.MISO!=seq_item.MISO_exp) begin
43          `uvm_error("scoreboard run phase",$sformatf("Design doesn't match the Golden model => %s",seq_item.convert2string))
44          error_count++;
45        end
46        else begin
47          correct_count++;
48        end
49      end
50    endtask
51
52    //Report Phase: report the testbench with how many failed and correct trials
53    function void report_phase(uvm_phase phase);
54      super.report_phase(phase);
55      `uvm_info("Scoreboard Report Phase",$sformatf("Testbench Completed , Correct count=
56      , error count=%d",correct_count,error_count),UVM_LOW)
57    endpackage
```

4.15. Wrapper Environment

```
 1 package wrapper_env_pkg;
 2 import uvm_pkg::*;
 3 `include "uvm_macros.svh"
 4 import wrapper_driver_pkg::*;
 5 import wrapper_scoreboard_pkg::*;
 6 import wrapper_agent_pkg::*;
 7 import shared_pkg::*;
 8 //import wrapper_cover_pkg::*;
 9   class wrapper_env extends uvm_env;
10     `uvm_component_utils(wrapper_env)
11     wrapper_scoreboard sb;
12     wrapper_agent agt;
13     // wrapper_cover cv;
14
15     function new(string name="wrapper_env",uvm_component parent=null);
16       super.new(name,parent);
17     endfunction
18     function void build_phase(uvm_phase phase);
19       super.build_phase(phase);
20       sb=wrapper_scoreboard::type_id::create("sb",this);
21       agt=wrapper_agent::type_id::create("agt",this);
22       //cv=wrapper_cover::type_id::create("wrapper_cover",this);
23     endfunction
24     function void connect_phase(uvm_phase phase);
25       super.connect_phase(phase);
26       agt.agt_ap.connect(sb.sb_export);
27       //agt.agt_ap.connect(cv.cover_ap);
28     endfunction
29   endclass
30 endpackage
```

4.16.SPI Sequence Item

4.17.SPI Configuration Object

```
● ● ●

1 package spi_config_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     class spi_config_obj extends uvm_object;
5         `uvm_object_utils(spi_config_obj)
6
7         virtual spi_if spi_config_vif;
8         uvm_active_passive_enum is_active;
9         function new(string name="alsu_config_obj");
10            super.new(name);
11        endfunction
12    endclass
13 endpackage
```

4.18.SPI Monitor

```
1 package spi_monitor_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import spi_seq_item::*;
5   import shared_pkg::*;
6   class spi_monitor extends uvm_monitor ;
7     `uvm_component_utils(spi_monitor)
8
9     //Creating Virtual interface to get the data from the interface
10    virtual spi_if s_if;
11    //Creating monitor sequence item to pass the interface to it to be then sent to the analysis components
12    spi_seq_item mon_seq_item;
13    //Creating monitor analysis port to broadcast the values to the analysis components
14    uvm_analysis_port #(spi_seq_item) mon_ap;
15
16    //Constructor
17    function new(string name="spi_monitor",uvm_component parent=null);
18      super.new(name,parent);
19    endfunction
20
21    //Build Phase
22    function void build_phase(uvm_phase phase);
23      super.build_phase(phase);
24      mon_ap=new("mon_ap",this);
25    endfunction
26
27    //Run Phase: samples the data at the negative edge and writes the analysis port with the monitor sequence item
28    task run_phase(uvm_phase phase);
29      super.run_phase(phase);
30      forever begin
31        mon_seq_item=spi_seq_item::type_id::create("mon_seq_item");
32        @(negedge s_if.clk);
33
34        //Drive the monitor sequence item with the interface
35        mon_seq_item.rst_n=s_if.rst_n;
36        mon_seq_item.SS_n=s_if.SS_n;
37        mon_seq_item.MOSI=s_if.MOSI;
38        mon_seq_item.MISO=s_if.MISO;
39        mon_seq_item.MISO_exp=s_if.MISO_exp;
40        mon_seq_item.tx_valid=s_if.tx_valid;
41        mon_seq_item.tx_data=s_if.tx_data;
42        mon_seq_item.rx_valid=s_if.rx_valid;
43        mon_seq_item.rx_valid_exp=s_if.rx_valid_exp;
44        mon_seq_item.rx_data=s_if.rx_data;
45        mon_seq_item.rx_data_exp=s_if.rx_data_exp;
46
47        //Broadcast the monitor sequence (interfance) to the analysis port
48        mon_ap.write(mon_seq_item);
49        `uvm_info("Monitor run phase",mon_seq_item.convert2string(),UVM_HIGH)
50      end
51    endtask
52  endclass
53 endpackage
```

4.19.SPI Agent

```
1 //////////////////////////////////////////////////////////////////
2 //Name: Omar Mohamed Hussein
3 //UVM_Component:Agent Package
4 //////////////////////////////////////////////////////////////////
5 package spi_agent_pkg;
6
7 import uvm_pkg::*;
8 `include "uvm_macros.svh"
9
10 import spi_seq_item::*;
11 import spi_sqr_pkg::*;
12 import spi_driver_pkg::*;
13 import spi_monitor_pkg::*;
14 import spi_config_pkg::*;
15
16 class spi_agent extends uvm_agent;
17   `uvm_component_utils(spi_agent)
18
19   //creating the sequencer,driver,monitor, configuration object to connect the interface,agent port
20   spi_sqr sqr;
21   spi_driver drv;
22   spi_monitor mon;
23   spi_config_obj config_obj;
24   uvm_analysis_port #(spi_seq_item) agt_ap;
25
26   //Constructor
27   function new(string name="spi_agent",uvm_component parent=null);
28     super.new(name,parent);
29   endfunction
30
31   //Build Phase
32   function void build_phase(uvm_phase phase);
33     super.build_phase(phase);
34     //Connect the agent configuration object with the configuration object in the uvm config data base
35     if(!uvm_config_db #(spi_config_obj)::get(this,"","CFG_SPI",config_obj)) begin
36       `uvm_fatal("agent build phase","unable to get the configuration object to the agent")
37     end
38
39   ////////////////////////////////////////////////////////////////// For Passive Agents //////////////////////////////////////////////////////////////////
40   if(config_obj.is_active==UM_ACTIVE) begin
41     sqr=spi_sqr::type_id::create("sqr",this);
42     drv=spi_driver::type_id::create("drv",this);
43   end
44   //////////////////////////////////////////////////////////////////
45
46   //Creating the sequencer,driver,monitor with the create() and the agt port with new()
47   mon=spi_monitor::type_id::create("mon",this);
48   agt_ap=new("agt_ap",this);
49 endfunction
50
51   //connect phase
52   function void connect_phase(uvm_phase phase);
53     super.connect_phase(phase);
54
55   ////////////////////////////////////////////////////////////////// For Passive Agents //////////////////////////////////////////////////////////////////
56   if(config_obj.is_active==UM_ACTIVE) begin
57     drv.spi_driver_vif=config_obj.spi_config_vif;
58     drv.seq_item_port.connect(sqr.seq_item_export);
59   end
60   //////////////////////////////////////////////////////////////////
61   mon.s_if=config_obj.spi_config_vif;
62   mon.mon_ap.connect(agt_ap);
63 endfunction
64 endclass
65 endpackage
```

4.20.SPI Scoreboard

```
● ○ ●
1 package spi_scoreboard_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4
5   import spi_seq_item::*;
6
7   class spi_scoreboard extends uvm_scoreboard ;
8     `uvm_component_utils(spi_scoreboard)
9
10    //Creating sequence item to put the data in the fifo
11    spi_seq_item seq_item;
12    //Creating analysis export to take values from the agt port
13    uvm_analysis_export #(spi_seq_item) sb_export;
14    //Creating analysis fifo to take values from sb export
15    uvm_tlm_analysis_fifo #(spi_seq_item) sb_fifo;
16
17    int correct_count,error_count;
18
19    //Constructor
20    function new(string name="spi_scoreboard",uvm_component parent =null);
21      super.new(name,parent);
22    endfunction
23
24    //Build phase : create the export analysis
25    function void build_phase(uvm_phase phase);
26      super.build_phase(phase);
27      sb_export=new("sb_export",this);
28      sb_fifo=new("sb_fifo",this);
29    endfunction
30
31    //Connect phase : connect the eb export with the fifo export
32    function void connect_phase (uvm_phase phase);
33      super.connect_phase(phase);
34      sb_export.connect(sb_fifo.analysis_export);
35    endfunction
36
37    //run Task : Compare the output with the Golden model
38    task run_phase(uvm_phase phase);
39      super.run_phase(phase);
40      forever begin
41        sb_fifo.get(seq_item);
42        if(seq_item.rx_valid!=seq_item.rx_data_exp || seq_item.rx_data!=seq_item.rx_data_exp || seq_item.MISO!=seq_item.MISO_exp) begin
43          `uvm_error("scoreboard run phase",$sformatf("Design doesn't match the Golden model => %s",seq_item.convert2string))
44          error_count++;
45        end
46        else begin
47          correct_count++;
48        end
49      end
50    endtask
51
52    //Report Phase: report the testbench with how many failed and correct trials
53    function void report_phase(uvm_phase phase);
54      super.report_phase(phase);
55      `uvm_info("Scoreboard Report Phase",$sformatf("Testbench Completed , Correct count=
56      , error count=%d",correct_count,error_count),UVM_LOW)
57    endfunction
58 endpackage
```

4.21.SPI Cover Collector

```
 1 package spi_cover_pkg;
 2   import uvm_pkg::*;
 3   `include "uvm_macros.svh"
 4   import spi_seq_item::*;
 5   import shared_pkg::*;
 6
 7   class spi_cover extends uvm_component;
 8     `uvm_component_utils(spi_cover)
 9     uvm_analysis_export #(spi_seq_item) cover_ap;
10     uvm_tlm_analysis_fifo #(spi_seq_item) cover_fifo;
11     spi_seq_item seq_item;
12
13
14   //Cover Groups
15   covergroup Cov_gp(ref spi_seq_item seq_item);
16   //Normal Coverpoints
17   rx_cp: coverpoint seq_item.rx_data[9:8];
18   bins all_values[] = {[0:3]};
19   bins WRITE_ADD =(2'b00=>2'b00=>2'b00) ;
20   bins WRITE_DATA=(2'b00=>2'b00=>2'b01) ;
21   bins READ_ADD  =(2'b00=>2'b10=>2'b10) ;
22   bins READ_DATA =(2'b00=>2'b10=>2'b11) ;
23 }
24 SS_n_cp: coverpoint seq_item.SS_n{
25   bins read_transition=(1=> 0[*23] =>1);
26   bins not_read_transition=(1=> 0[*13] =>1);
27 }
28 MOSI_cp:coverpoint seq_item.MOSI{
29   bins b00=(0=>0=>0);
30   bins b01=(0=>0=>1);
31   bins b11=(1=>1=>1);
32   bins b10=(1=>1=>0);
33 }
34 cross_cv:cross MOSI_cp,SS_n_cp{
35   ignore_bins write_cv_000 =binsof(MOSI_cp.b000) && binsof(SS_n_cp.read_trans
36   ition);
37   ignore_bins write_cv_001 =binsof(MOSI_cp.b001) && binsof(SS_n_cp.read_trans
38   ition);
39   ignore_bins read_cv_111 =binsof(MOSI_cp.b111) && binsof(SS_n_cp.not_read_tr
40   ansition);
41   ignore_bins read_cv_110 =binsof(MOSI_cp.b110) && binsof(SS_n_cp.not_read_tr
42   ansition);
43 }
44 //Cross coverage
45 endgroup
46
47 function new(string name="spi_cover",uvm_component parent=null);
48   super.new(name,parent);
49   Cov_gp=new(seq_item);
50 endfunction
51
52 function void build_phase(uvm_phase phase);
53   super.build_phase(phase);
54   cover_ap=new("cover_ap",this);
55   cover_fifo=new("cover_fifo",this);
56 endfunction
57
58 function void connect_phase(uvm_phase phase);
59   super.connect_phase(phase);
60   cover_ap.connect(cover_fifo.analysis_export);
61 endfunction
62
63 task run_phase(uvm_phase phase);
64   super.run_phase(phase);
65   forever begin
66     seq_item=spi_seq_item::type_id::create("seq_item");
67     cover_fifo.get(seq_item);
68     Cov_gp.sample();
69   end
70 endtask
71
72 endclass
73 endpackage
```

4.22.SPI Environment

```
● ● ●

1 package spi_env_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import spi_driver_pkg::*;
5 import spi_scoreboard_pkg::*;
6 import spi_agent_pkg::*;
7 import shared_pkg::*;
8 import spi_cover_pkg::*;
9 class spi_env extends uvm_env;
10    `uvm_component_utils(spi_env)
11    spi_scoreboard sb;
12    spi_agent agt;
13    spi_cover cv;
14
15    function new(string name="spi_env",uvm_component parent=null);
16        super.new(name,parent);
17    endfunction
18    function void build_phase(uvm_phase phase);
19        super.build_phase(phase);
20        sb=spi_scoreboard::type_id::create("sb",this);
21        agt=spi_agent::type_id::create("agt",this);
22        cv=spi_cover::type_id::create("spi_cover",this);
23    endfunction
24    function void connect_phase(uvm_phase phase);
25        super.connect_phase(phase);
26        agt.agt_ap.connect(sb.sb_export);
27        agt.agt_ap.connect(cv.cover_ap);
28    endfunction
29 endclass
30 endpackage
```

4.23.SPI Assertions

```
1 module spi_sva(MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
2 input MOSI,MISO,SS_n,clk,rst_n,rx_valid,tx_valid;
3 input [9:0]rx_data;
4 input [7:0]tx_data;
5
6 sequence valid_comb_000;
7 ($fell(SS_n) ##1 !MOSI ##1 !MOSI ##1 !MOSI);
8 endsequence
9 sequence valid_comb_001;
10 ($fell(SS_n) ##1 !MOSI ##1 !MOSI ##1 MOSI);
11 endsequence
12 sequence valid_comb_110;
13 ($fell(SS_n) ##1 MOSI ##1 MOSI ##1 !MOSI);
14 endsequence
15 sequence valid_comb_111;
16 ($fell(SS_n) ##1 MOSI ##1 MOSI ##1 MOSI);
17 endsequence
18 assert property(@(posedge clk) disable iff(!rst_n) valid_comb_000 |-> ##10 (rx_valid ##1 SS_n[->1]));
19 assert property(@(posedge clk) disable iff(!rst_n) valid_comb_001 |-> ##10 (rx_valid ##1 SS_n[->1]));
20 assert property(@(posedge clk) disable iff(!rst_n) valid_comb_110 |-> ##10 (rx_valid ##1 SS_n[->1]));
21 assert property(@(posedge clk) disable iff(!rst_n) valid_comb_111 |-> ##10 (rx_valid ##1 SS_n[->1]));
22 assert property(@(posedge clk) !rst_n |=> !MISO && !rx_valid && rx_data==0);
23 cover property(@(posedge clk) disable iff(!rst_n) valid_comb_000 |-> ##10 (rx_valid ##1 SS_n[->1]));
24 cover property(@(posedge clk) disable iff(!rst_n) valid_comb_001 |-> ##10 (rx_valid ##1 SS_n[->1]));
25 cover property(@(posedge clk) disable iff(!rst_n) valid_comb_110 |-> ##10 (rx_valid ##1 SS_n[->1]));
26 cover property(@(posedge clk) disable iff(!rst_n) valid_comb_111 |-> ##10 (rx_valid ##1 SS_n[->1]));
27 cover property(@(posedge clk) !rst_n |=> !MISO && !rx_valid && rx_data==0);
28 endmodule
```

4.24.RAM Sequence Item

```

1 //////////////////////////////////////////////////////////////////
2 // Name: Omar Mohamed Hussein
3 // UVM_Object : Sequence Item
4 //////////////////////////////////////////////////////////////////
5
6 package Ram_seq_item_pkg;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   class Ram_seq_item extends uvm_sequence_item;
10  `uvm_object_utils(Ram_seq_item)
11  //Constructor function
12  function new(string name="Ram_seq_item");
13    super.new(name);
14  endfunction
15
16  typedef enum {WRITE_ADD,WRITE_DATA,READ_ADD,READ_DATA} operations;
17
18  //Interface signals
19  operations old_op;
20  rand logic [9:0] din;
21  rand logic rst_n, rx_valid;
22  logic [7:0] dout,dout_exp;
23  logic tx_valid,tx_valid_exp;
24  //rand type signal;
25
26  constraint rst_constraints {rst_n dist{1:/90 , 0:/5};}
27  constraint rx_valid_constraints {rx_valid dist{1:/85 , 0:/15};}
28  constraint write_only_seq_constraints{if(old_op==WRITE_ADD) operations'(din[9:8]) inside {WRITE_ADD,WRITE_DATA};}
29  constraint read_only_seq_constraints {if(old_op==READ_ADD) operations'(din[9:8]) inside {READ_ADD, READ_DATA};}
30  constraint read_write_constraints {
31    if(old_op==WRITE_ADD && rst_n) operations'(din[9:8]) inside {WRITE_ADD,WRITE_DATA};
32    if(old_op==WRITE_DATA && rst_n) operations'(din[9:8]) dist {READ_ADD:/60 , WRITE_ADD:/40};
33    if(old_op==READ_ADD && rst_n) operations'(din[9:8]) inside {READ_ADD,READ_DATA};
34    if(old_op==READ_DATA && rst_n) operations'(din[9:8]) dist {WRITE_ADD:/60 , READ_ADD:/40};
35  }
36
37  function void post_randomize();
38    if(!rst_n) old_op=READ_DATA;
39    else old_op =operations'(din[9:8]);
40  endfunction
41  //Constrain Blocks
42
43  function string convert2string_stimulus();
44    return $sformatf("Error in the Ram : reset=
45 ,din");
46  endfunction
47  `uvm_object_end
48  string convert2string();
49  `uvm_object_end
50  ,dou
51  `uvm_object_end
52  ,dout_exp=

```

4.25.RAM Configuration Object

```
1 package Ram_config_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   class Ram_config_obj extends uvm_object;
5     `uvm_object_utils(Ram_config_obj)
6
7     virtual Ram_if Ram_config_vif;
8     uvm_active_passive_enum is_active;
9     function new(string name="Ram_config_obj");
10       super.new(name);
11     endfunction
12
13   endclass
14 endpackage
```

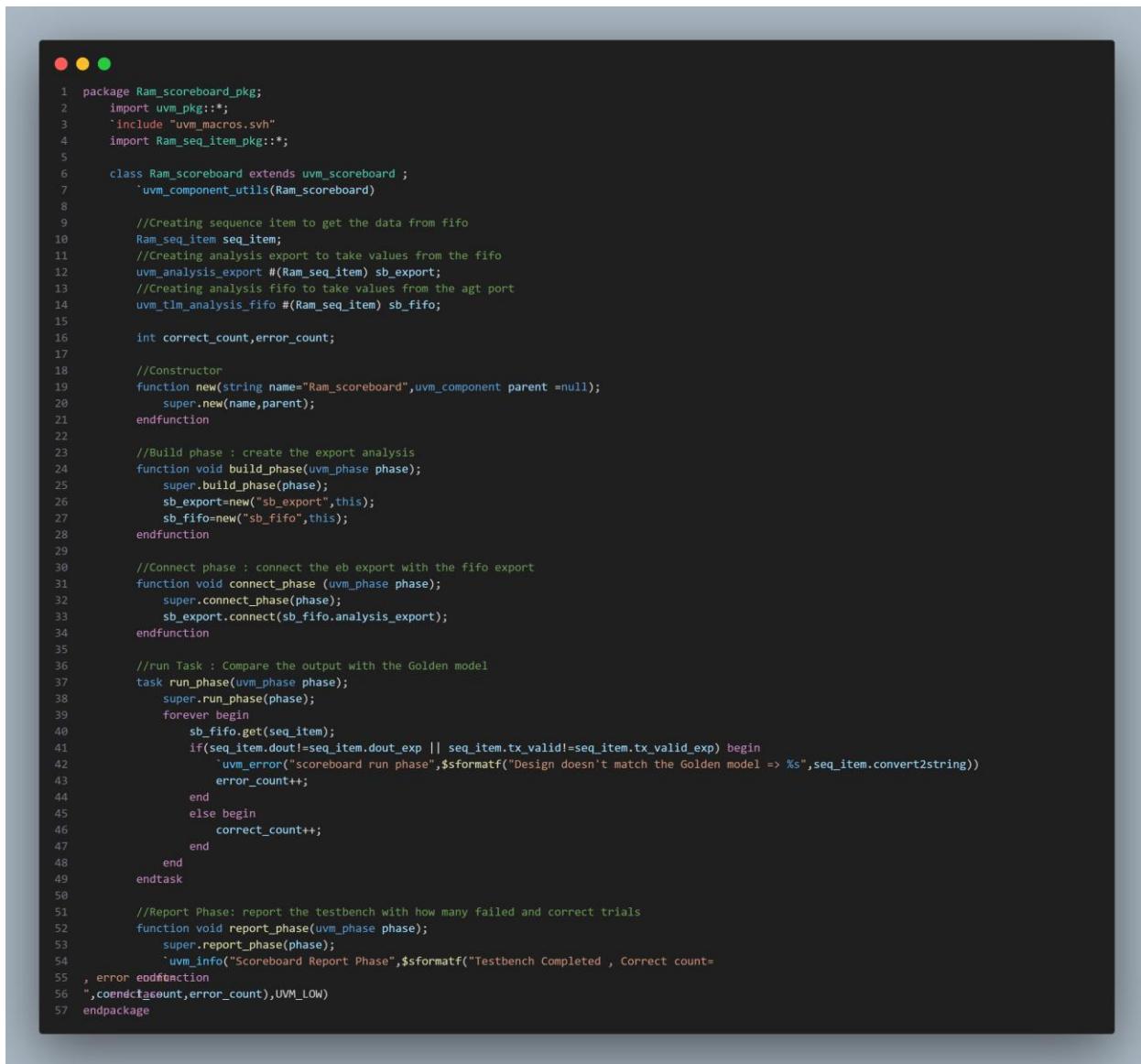
4.26.RAM Monitor

```
1 package Ram_monitor_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import Ram_seq_item_pkg::*;
5   class Ram_monitor extends uvm_monitor ;
6     `uvm_component_utils(Ram_monitor)
7
8     //Creating Virtual interface to get the data from the interface
9     virtual Ram_if r_if;
10    //Creating monitor sequence item to pass the interface to it to be then sent to the analysis components
11    Ram_seq_item mon_seq_item;
12    //Creating monitor analysis port to broadcast the values to the analysis components
13    uvm_analysis_port #(Ram_seq_item) mon_ap;
14
15    //Constructor
16    function new(string name="Ram_monitor",uvm_component parent=null);
17      super.new(name,parent);
18    endfunction
19
20    //Build Phase
21    function void build_phase(uvm_phase phase);
22      super.build_phase(phase);
23      mon_ap=new("mon_ap",this);
24    endfunction
25
26    //Run Phase: samples the data at the negative edge and writes the analysis port with the monitor sequence item
27    task run_phase(uvm_phase phase);
28      super.run_phase(phase);
29      forever begin
30        mon_seq_item=Ram_seq_item::type_id::create("mon_seq_item");
31        @ (negedge r_if.clk);
32
33        //Drive the monitor sequence item with the interface
34        mon_seq_item.din      =r_if.din;
35        mon_seq_item.rst_n    =r_if.rst_n;
36        mon_seq_item.rx_valid =r_if.rx_valid;
37        mon_seq_item.dout     =r_if.dout;
38        mon_seq_item.tx_valid =r_if.tx_valid;
39        mon_seq_item.tx_valid_exp=r_if.tx_valid_exp;
40        mon_seq_item.dout_exp =r_if.dout_exp;
41
42        //Broadcast the monitor sequence (interfance) to the analysis port
43        mon_ap.write(mon_seq_item);
44        `uvm_info("Monitor run phase",mon_seq_item.convert2string(),UVM_HIGH)
45      end
46    endtask
47  endclass
48 endpackage
```

4.27.RAM Agent

```
1 //////////////////////////////////////////////////////////////////
2 //Name: Omar Mohamed Hussein
3 //UVM_Component:Agent Package
4 //////////////////////////////////////////////////////////////////
5 package Ram_agent_pkg;
6
7     import uvm_pkg::*;
8     `include "uvm_macros.svh"
9
10    import Ram_seq_item_pkg::*;
11    import Ram_sqr_pkg::*;
12    import Ram_driver_pkg::*;
13    import Ram_monitor_pkg::*;
14    import Ram_config_pkg::*;
15
16    class Ram_agent extends uvm_agent;
17        `uvm_component_utils(Ram_agent)
18
19            //creating the sequencer,driver,monitor, configuration object to connect the interface,agent port
20            Ram_sqr sqr;
21            Ram_driver drv;
22            Ram_monitor mon;
23            Ram_config_obj config_obj;
24            uvm_analysis_port #(Ram_seq_item) agt_ap;
25
26            //Constructor
27            function new(string name="Ram_agent",uvm_component parent=null);
28                super.new(name,parent);
29            endfunction
30
31            //Build Phase
32            function void build_phase(uvm_phase phase);
33                super.build_phase(phase);
34                //Connect the agent configuration object with the configuration object in the uvm config data base
35                if(!uvm_config_db #(Ram_config_obj)::get(this,"CFG_RAM",config_obj)) begin
36                    `uvm_fatal("agent build phase","unable to get the configuration object to the agent")
37                end
38
39                //////////////// For Passive Agents ///////////////////
40                if(config_obj.is_active==UVM_ACTIVE) begin
41                    sqr=Ram_sqr::type_id::create("sqr",this);
42                    drv=Ram_driver::type_id::create("drv",this);
43                end
44                ////////////////
45                mon=Ram_monitor::type_id::create("mon",this);
46                agt_ap=new("agt_ap",this);
47            endfunction
48
49            //connect phase
50            function void connect_phase(uvm_phase phase);
51                super.connect_phase(phase);
52                mon.r_if=config_obj.Ram_config_vif;
53                //Connect the driver port with the sequencer export
54                if(config_obj.is_active==UVM_ACTIVE) begin
55                    drv.seq_item_port.connect(sqr.seq_item_export);
56                    drv.Ram_driver_vif=config_obj.Ram_config_vif;
57                end
58                //Connect the monitor analysis port with the agt port
59                mon.mon_ap.connect(agt_ap);
60            endfunction
61        endclass
62    endpackage
```

4.28.RAM Scoreboard



```
1 package Ram_scoreboard_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import Ram_seq_item_pkg::*;
5
6   class Ram_scoreboard extends uvm_scoreboard ;
7     `uvm_component_utils(Ram_scoreboard)
8
9     //Creating sequence item to get the data from fifo
10    Ram_seq_item seq_item;
11    //Creating analysis export to take values from the fifo
12    uvm_analysis_export #(Ram_seq_item) sb_export;
13    //Creating analysis fifo to take values from the agt port
14    uvm_tlm_analysis_fifo #(Ram_seq_item) sb_fifo;
15
16    int correct_count,error_count;
17
18    //Constructor
19    function new(string name="Ram_scoreboard",uvm_component parent =null);
20      super.new(name,parent);
21    endfunction
22
23    //Build phase : create the export analysis
24    function void build_phase(uvm_phase phase);
25      super.build_phase(phase);
26      sb_export=new("sb_export",this);
27      sb_fifo=new("sb_fifo",this);
28    endfunction
29
30    //Connect phase : connect the eb export with the fifo export
31    function void connect_phase (uvm_phase phase);
32      super.connect_phase(phase);
33      sb_export.connect(sb_fifo.analysis_export);
34    endfunction
35
36    //run Task : Compare the output with the Golden model
37    task run_phase(uvm_phase phase);
38      super.run_phase(phase);
39      forever begin
40        sb_fifo.get(seq_item);
41        if(seq_item.dout!=seq_item.dout_exp || seq_item.tx_valid!=seq_item.tx_valid_exp) begin
42          `uvm_error("scoreboard run phase",$sformatf("Design doesn't match the Golden model => %s",seq_item.convert2string))
43          error_count++;
44        end
45        else begin
46          correct_count++;
47        end
48      end
49    endtask
50
51    //Report Phase: report the testbench with how many failed and correct trials
52    function void report_phase(uvm_phase phase);
53      super.report_phase(phase);
54      `uvm_info("Scoreboard Report Phase",$sformatf("Testbench Completed , Correct count=
55      , error count=%d",correct_count,error_count),UVM_LOW)
56    endfunction
57 endpackage
```

4.29.RAM Cover Collector

```
1 package Ram_cover_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import Ram_seq_item_pkg::*;
5
6   class Ram_cover extends uvm_component;
7     `uvm_component_utils(Ram_cover)
8     uvm_analysis_export #(Ram_seq_item) cover_ap;
9     uvm_tlm_analysis_fifo #(Ram_seq_item) cover_fifo;
10    Ram_seq_item seq_item;
11
12
13  //Cover Groups
14  covergroup Cov_gp(ref Ram_seq_item seq_item);
15    //Normal Coverpoints
16    operations_cp: coverpoint seq_item.din[9:8] {
17      bins all_vals[]={0:3};
18      // bins Write_address_data=(2'b00=>2'b01);
19      // bins Read_address_data =(2'b10=>2'b11);           Impossible to hit in the Wrapper module
20      // bins all_operations =(2'b00=>2'b01[*13]=>2'b10[*13]=>2'b11); Impossible to hit in the Wrapper module
21    }
22    rx_valid_cp:coverpoint seq_item.rx_valid{
23      bins high={1};
24      bins low={0};
25    }
26    //Cross coverage
27    rx_valid_ops:cross operations_cp,rx_valid_cp {
28      bins allvals_rx_valid = binsof(operations_cp.all_vals) && binsof(rx_valid_cp.high);
29      bins read_data_tx_valid = binsof(operations_cp.all_vals) intersect {2'b11} && binsof(rx_valid_cp.high);
30    }
31  endgroup
32  function new(string name="Ram_cover",uvm_component parent=null);
33    super.new(name,parent);
34    Cov_gp=new(seq_item);
35  endfunction
36
37  function void build_phase(uvm_phase phase);
38    super.build_phase(phase);
39    cover_ap=new("cover_ap",this);
40    cover_fifo=new("cover_fifo",this);
41  endfunction
42
43  function void connect_phase(uvm_phase phase);
44    super.connect_phase(phase);
45    cover_ap.connect(cover_fifo.analysis_export);
46  endfunction
47
48  task run_phase(uvm_phase phase);
49    super.run_phase(phase);
50    forever begin
51      seq_item=Ram_seq_item::type_id::create("seq_item");
52      cover_fifo.get(seq_item);
53      Cov_gp.sample();
54    end
55  endtask
56
57  endclass
58 endpackage
```

4.30.RAM Environment

```
● ● ●

1 package Ram_env_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import Ram_driver_pkg::*;
5 import Ram_scoreboard_pkg::*;
6 import Ram_agent_pkg::*;
7 import Ram_cover_pkg::*;
8 class Ram_env extends uvm_env;
9   `uvm_component_utils(Ram_env)
10  Ram_scoreboard sb;
11  Ram_agent agt;
12  Ram_cover cv;
13
14  function new(string name="Ram_env",uvm_component parent=null);
15    super.new(name,parent);
16  endfunction
17  function void build_phase(uvm_phase phase);
18    super.build_phase(phase);
19    sb=Ram_scoreboard::type_id::create("sb",this);
20    agt=Ram_agent::type_id::create("agt",this);
21    cv=Ram_cover::type_id::create("cv",this);
22  endfunction
23  function void connect_phase(uvm_phase phase);
24    super.connect_phase(phase);
25    agt.agt_ap.connect(sb.sb_export);
26    agt.agt_ap.connect(cv.cover_ap);
27  endfunction
28 endclass
29 endpackage
```

4.31.RAM Assertions

```
1 module Ram_sva(din,clk,rst_n,rx_valid,dout,tx_valid);
2   input      [9:0] din;
3   input      clk, rst_n, rx_valid;
4
5   input reg [7:0] dout;
6   input reg      tx_valid;
7
8   sequence ops;
9     (din[9:8]==2'b00 || din[9:8]==2'b01 || din[9:8]==2'b10) && rx_valid;
10  endsequence
11 assert property(@(posedge clk) !rst_n |=> dout==0 && !tx_valid);
12 assert property(@(posedge clk) disable iff(!rst_n) ops |=> !tx_valid );
13 assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b11 |=> $rose(tx_valid) |=> $fell(tx_valid)[->1]);
14 assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b00 |=> (rx_valid && din[9:8]== 2'b01)[->1]);
15 assert property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b10 |=> (rx_valid && din[9:8]== 2'b11)[->1]);
16
17 cover property(@(posedge clk) !rst_n |=> dout==0 && !tx_valid);
18 cover property(@(posedge clk) disable iff(!rst_n) ops |=> !tx_valid && dout==0);
19 cover property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b11 |=> $rose(tx_valid) ##1 $fell(tx_valid)[->1]);
20 cover property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b00 |=> (rx_valid && din[9:8]== 2'b01)[->1]);
21 cover property(@(posedge clk) disable iff(!rst_n) rx_valid && din[9:8]==2'b10 |=> (rx_valid && din[9:8]== 2'b11)[->1]);
22
23 endmodule
```

4.32. Wrapper Test

```

1 package wrapper_test_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import wrapper_env_pkg::*;
5 import Ram_env_pkg::*;
6 import spi_env_pkg::*;
7 import wrapper_config_pkg::*;
8 import spi_config_pkg::*;
9 import Ram_config_pkg::*;
10 import wrapper_reset_seq_pkg::*;
11 import wrapper_write_seq_pkg::*;
12 import wrapper_read_write_seq_pkg::*;
13 import wrapper_read_seq_pkg::*;
14 class wrapper_test extends uvm_test;
15     `uvm_component_utils(wrapper_test)
16 
17     wrapper_env w_env;
18     spi_env s_env;
19     Ram_env r_env;
20     wrapper_config_obj wrapper_config_obj_test;
21     spi_config_obj spi_config_obj_test;
22     Ram_config_obj Ram_config_obj_test;
23     wrapper_reset_seq reset_seq;
24     wrapper_write_seq write_seq;
25     wrapper_read_seq read_seq;
26     wrapper_read_write_seq write_read_seq;
27 
28     function new(string name = "wrapper_test", uvm_component parent=null);
29         super.new(name, parent);
30     endfunction
31 
32     function void build_phase(uvm_phase phase);
33         super.build_phase(phase);
34         w_env=wrapper_env::type_id::create("w_env",this);
35         r_env=Ram_env::type_id::create("r_env",this);
36         s_env=spi_env::type_id::create("s_env",this);
37         wrapper_config_obj_test=wrapper_config_obj::type_id::create("wrapper_config_obj_test");
38         Ram_config_obj_test=Ram_config_obj::type_id::create("Ram_config_obj_test");
39         spi_config_obj_test=spi_config_obj::type_id::create("spi_config_obj_test");
40         reset_seq=wrapper_reset_seq::type_id::create("reset_seq");
41         write_seq=wrapper_write_seq::type_id::create("write_seq");
42         read_seq=wrapper_read_seq::type_id::create("read_seq");
43         write_read_seq=wrapper_read_write_seq::type_id::create("write_read_seq");
44 
45         if(!uvm_config_db #(virtual wrapper_if)::get(this,"","WRAPPER",wrapper_config_obj_test.wrapper_config_vif))
46             `uvm_fatal("build phase in test","unable to get interface from the database into the configuration object")
47         uvm_config_db#(wrapper_config_obj)::set(this,"*","CFG_WRAPPER",wrapper_config_obj_test);
48 
49         if(!uvm_config_db #(virtual spi_if)::get(this,"","SPI",spi_config_obj_test.spi_config_vif))
50             `uvm_fatal("build phase in test","unable to get interface from the database into the configuration object")
51         uvm_config_db#(spi_config_obj)::set(this,"*","CFG_SPI",spi_config_obj_test);
52 
53         if(!uvm_config_db #(virtual Ram_if)::get(this,"","RAM",Ram_config_obj_test.Ram_config_vif))
54             `uvm_fatal("build phase in test","unable to get interface from the database into the configuration object")
55         uvm_config_db#(Ram_config_obj)::set(this,"*","CFG_RAM",Ram_config_obj_test);
56 
57         wrapper_config_obj_test.is_active=UVM_ACTIVE;
58         spi_config_obj_test.is_active=UVM_PASSIVE;
59         Ram_config_obj_test.is_active=UVM_PASSIVE;
60     endfunction
61 
62     task run_phase(uvm_phase phase);
63         super.run_phase(phase);
64         phase.raise_objection(this);
65         `uvm_info("Test run phase","Reset Asserted",UVM_LOW)
66         reset_seq.start(w_env.agt.sqr);
67         `uvm_info("Test run phase","Reset Finished",UVM_LOW)
68 
69         `uvm_info("Test run phase","Write sequence Asserted",UVM_LOW)
70         write_seq.start(w_env.agt.sqr);
71         `uvm_info("Test run phase","Write sequence Finished",UVM_LOW)
72 
73         `uvm_info("Test run phase","Read sequence Asserted",UVM_LOW)
74         read_seq.start(w_env.agt.sqr);
75         `uvm_info("Test run phase","Read sequence Finished",UVM_LOW)
76 
77         `uvm_info("Test run phase","Write Read sequence Asserted",UVM_LOW)
78         write_read_seq.start(w_env.agt.sqr);
79         `uvm_info("Test run phase","Write Read sequence Finished",UVM_LOW)
80         phase.drop_objection(this);
81     endtask
82 endclass
83 endpackage

```

4.33. Wrapper Top Module

```
1 import uvm_pkg::*;
2 `include "uvm_macros.svh"
3 import wrapper_test_pkg::*;
4
5 module top();
6   // Clock generation
7   bit clk;
8   initial begin
9     clk=0;
10    forever begin
11      #10 clk=~clk;
12    end
13  end
14  initial begin
15    $readmemh("mem.dat",Golden.RAM.MEM);
16    $readmemh("mem.dat",DUT.RAM_instance.MEM);
17  end
18 // Instantiate the interface ,Golden Model and DUT
19 Ram_if r_if(clk);
20 spi_if s_if(clk);
21 wrapper_if w_if(clk);
22
23 WRAPPER DUT(w_if.MOSI,w_if.MISO,w_if.SS_n,w_if.clk,w_if.rst_n);
24 SPI_WITH_SPR #(256,8)Golden(w_if.clk,w_if.rst_n,w_if.MISO_exp,w_if.MOSI,w_if.SS_n);
25
26 assign s_if.rx_data= DUT.rx_data_din;
27 assign s_if.clk= DUT.clk;
28 assign s_if.rst_n= DUT.rst_n;
29 assign s_if.rx_valid= DUT.rx_valid;
30 assign s_if.tx_data= DUT.tx_data_dout;
31 assign s_if.tx_valid= DUT.tx_valid;
32 assign s_if.rx_data_exp = Golden.rx_data;
33 assign s_if.rx_valid_exp= Golden.rx_valid;
34 assign s_if.MISO_exp= Golden.MISO;
35 assign s_if.MISO= DUT.MISO;
36 assign s_if.SS_n= DUT.SS_n;
37 assign s_if.MOSI= DUT.MOSI;
38
39 assign r_if.din= DUT.rx_data_din;
40 assign r_if.clk= DUT.clk;
41 assign r_if.rst_n= DUT.rst_n;
42 assign r_if.rx_valid= DUT.rx_valid;
43 assign r_if.dout= DUT.tx_data_dout;
44 assign r_if.dout_exp= Golden.tx_data;
45 assign r_if.tx_valid= DUT.tx_valid;
46 assign r_if.tx_valid_exp= Golden.tx_valid;
47
48 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);
49 bind DUT.SLAVE_instance spi_sva SPI_assertion (MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
50
51 // run test using run_test task
52 initial begin
53   uvm_config_db #(virtual Ram_if)::set(null,"uvm_test_top","RAM",r_if);
54   uvm_config_db #(virtual spi_if)::set(null,"uvm_test_top","SPI",s_if);
55   uvm_config_db #(virtual wrapper_if)::set(null,"uvm_test_top","WRAPPER",w_if);
56   run_test("wrapper_test");
57 end
58 endmodule
```

4.34.Source Files

```
1 shared_pkg.sv
2 spi_if.sv
3 SPI_slave.sv
4 spi_sva.sv
5 down_counter.v
6 SPI_slave_optimized.v
7 spi_config_obj.sv
8 spi_seq_item.sv
9 spi_sqr.sv
10 spi_driver.sv
11 spi_monitor.sv
12 spi_agent.sv
13 spi_scoreboard.sv
14 spi_cover.sv
15 spi_env.sv
16 Ram_if.sv
17 RAM.v
18 Ram_sva.sv
19 SPR.v
20 Ram_config_obj.sv
21 Ram_seq_item.sv
22 Ram_sqr.sv
23 Ram_driver.sv
24 Ram_monitor.sv
25 Ram_agent.sv
26 Ram_scoreboard.sv
27 Ram_cover.sv
28 Ram_env.sv
29 Ram_if.sv
30 SPI_wrapper.v
31 Ram_sva.sv
32 SPI_WITH_SPR.v
33 wrapper_if.sv
34 wrapper_config_obj.sv
35 wrapper_seq_item.sv
36 wrapper_reset_seq.sv
37 wrapper_write_seq.sv
38 wrapper_read_seq.sv
39 wrapper_read_write_seq.sv
40 wrapper_sqr.sv
41 wrapper_driver.sv
42 wrapper_monitor.sv
43 wrapper_agent.sv
44 wrapper_scoreboard.sv
45 wrapper_env.sv
46 wrapper_test.sv
47 wrapper_top.sv
48
```

4.35.Do File

```
1 vlib work
2 vlog +define+SIM -f src_files.list +cover -covercells
3 vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
4 add wave /top/s_if/*
5 add wave /top/r_if/*
6 add wave /top/w_if/*
7 add wave -position insertpoint \
8 sim:/top/DUT/SLAVE_instance/cs
9 add wave -position insertpoint \
10 sim:/top/DUT/SLAVE_instance/counter
11 add wave -position insertpoint \
12 sim:/top/Golden/SPI/cs \
13 sim:/top/Golden/SPI/count \
14 sim:/top/Golden/SPI/count_10
15 run 0
16 coverage save top.ucdb -onexit
17 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(56) @ 220020: uvm_test_top.w_env.sb [Scoreboard Report Phase] Testbench Completed , Correct count=11001, error count=0

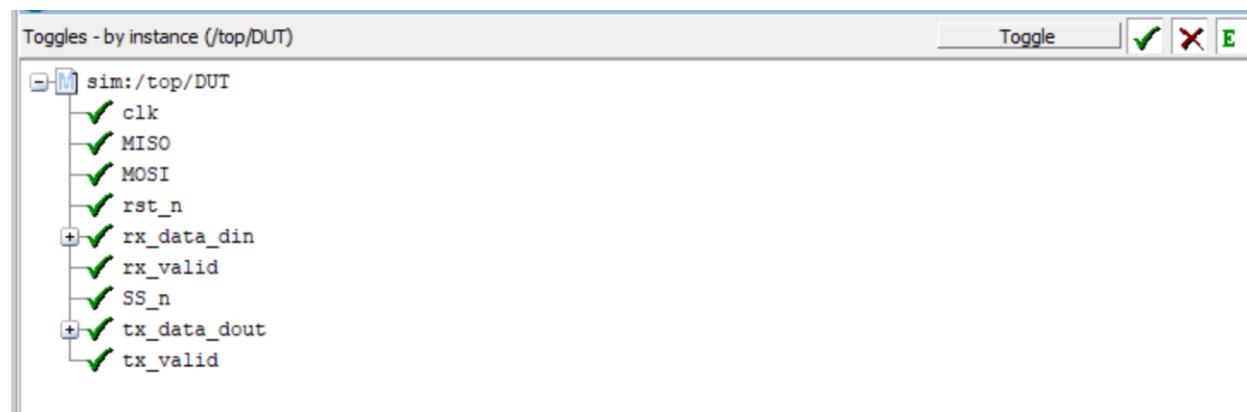
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO : 15
UVM_WARNING : 0
UVM_ERROR : 0
UVM_FATAL : 0
** Report counts by id
[Questa UVM] 2
[RNST] 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>
```

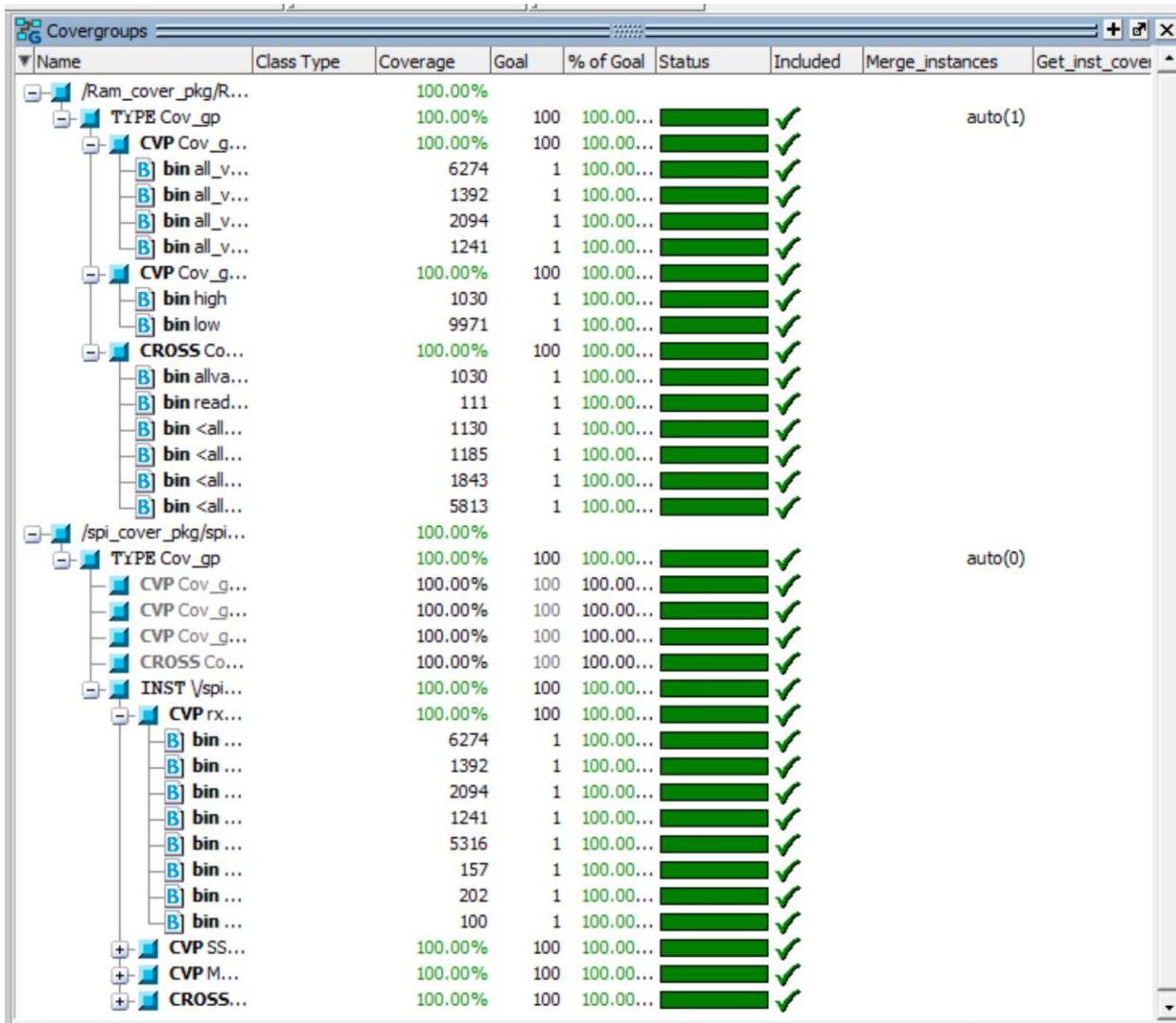
4.36.2. Questasim Code Coverage

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

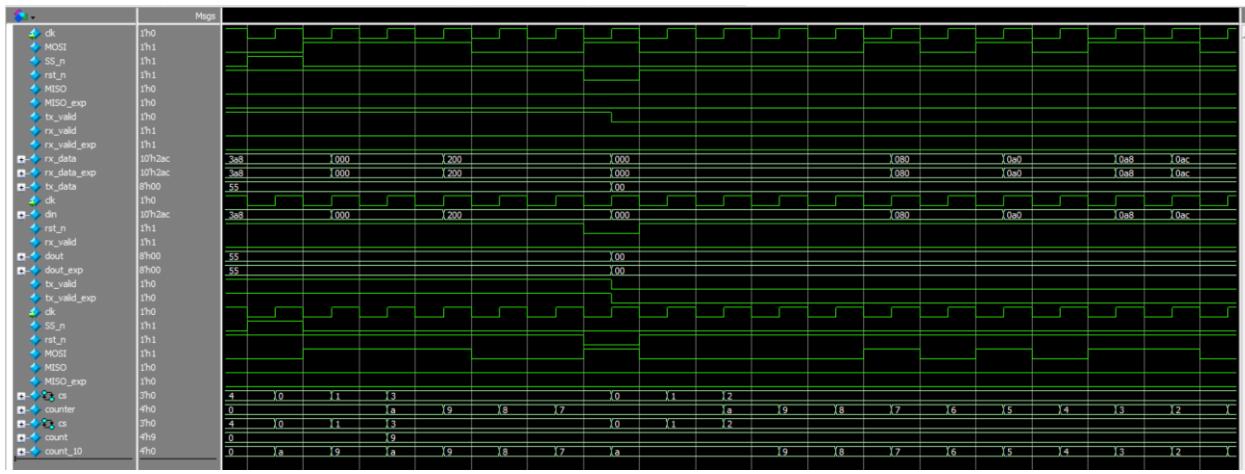
Name	Assertion Type	Language
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed_1735	Immediate	SVA
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed_1775	Immediate	SVA
/wrapper_read_seq_pkg::wrapper_read_seq::body/#ublk#130774743#20/immed_22	Immediate	SVA
/wrapper_read_write_seq_pkg::wrapper_read_write_seq::body/#ublk#157015479#20/immed_22	Immediate	SVA
/wrapper_write_seq_pkg::wrapper_write_seq::body/#ublk#247560343#20/immed_22	Immediate	SVA
+ /top/DUT/RAM_instance/Ram_assertion/assert_0	Concurrent	SVA
+ /top/DUT/RAM_instance/Ram_assertion/assert_1	Concurrent	SVA
+ /top/DUT/RAM_instance/Ram_assertion/assert_2	Concurrent	SVA
+ /top/DUT/RAM_instance/Ram_assertion/assert_3	Concurrent	SVA
+ /top/DUT/RAM_instance/Ram_assertion/assert_4	Concurrent	SVA
+ /top/DUT/SLAVE_instance/assert_0	Concurrent	SVA
+ /top/DUT/SLAVE_instance/assert_1	Concurrent	SVA
+ /top/DUT/SLAVE_instance/assert_2	Concurrent	SVA
+ /top/DUT/SLAVE_instance/assert_3	Concurrent	SVA
+ /top/DUT/SLAVE_instance/SPI_assertion/assert_0	Concurrent	SVA
+ /top/DUT/SLAVE_instance/SPI_assertion/assert_1	Concurrent	SVA
+ /top/DUT/SLAVE_instance/SPI_assertion/assert_2	Concurrent	SVA
+ /top/DUT/SLAVE_instance/SPI_assertion/assert_3	Concurrent	SVA
+ /top/DUT/SLAVE_instance/SPI_assertion/assert_4	Concurrent	SVA

4.36.5.Questasim Assertions Coverage

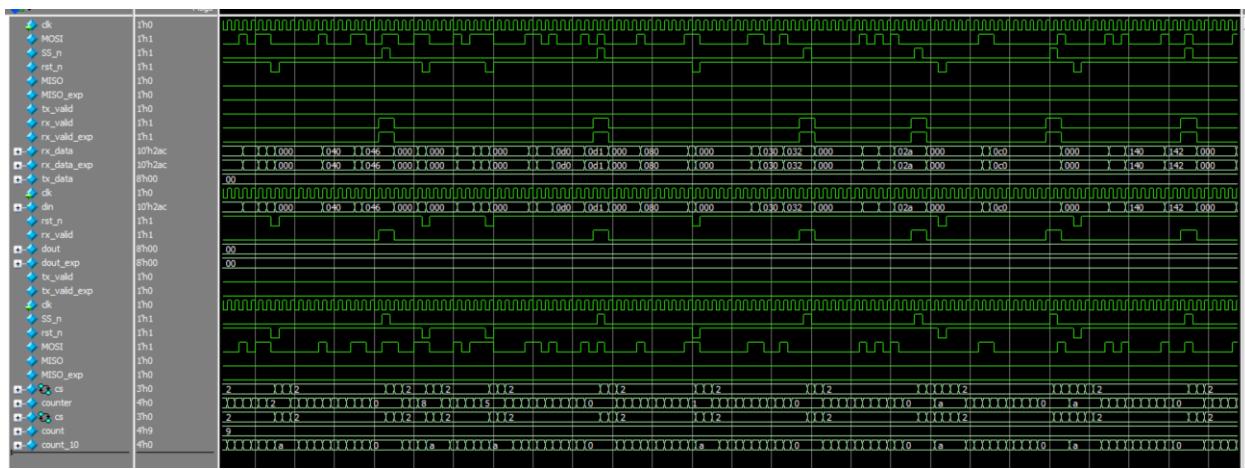
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight
/top/DUT/RAM_instance/Ram_assertion/cover_4	SVA	✓	Off	108	1	Unli...	1
/top/DUT/RAM_instance/Ram_assertion/cover_3	SVA	✓	Off	164	1	Unli...	1
/top/DUT/RAM_instance/Ram_assertion/cover_2	SVA	✓	Off	18	1	Unli...	1
/top/DUT/RAM_instance/Ram_assertion/cover_1	SVA	✓	Off	823	1	Unli...	1
/top/DUT/RAM_instance/Ram_assertion/cover_0	SVA	✓	Off	567	1	Unli...	1
/top/DUT/SLAVE_instance/cover_3	SVA	✓	Off	34	1	Unli...	1
/top/DUT/SLAVE_instance/cover_2	SVA	✓	Off	117	1	Unli...	1
/top/DUT/SLAVE_instance/cover_1	SVA	✓	Off	885	1	Unli...	1
/top/DUT/SLAVE_instance/cover_0	SVA	✓	Off	933	1	Unli...	1
/top/DUT/SLAVE_instance/SPI_assertion/cover_4	SVA	✓	Off	567	1	Unli...	1
/top/DUT/SLAVE_instance/SPI_assertion/cover_3	SVA	✓	Off	35	1	Unli...	1
/top/DUT/SLAVE_instance/SPI_assertion/cover_2	SVA	✓	Off	10	1	Unli...	1
/top/DUT/SLAVE_instance/SPI_assertion/cover_1	SVA	✓	Off	29	1	Unli...	1
/top/DUT/SLAVE_instance/SPI_assertion/cover_0	SVA	✓	Off	49	1	Unli...	1

4.36.6.Questasim Waveforms

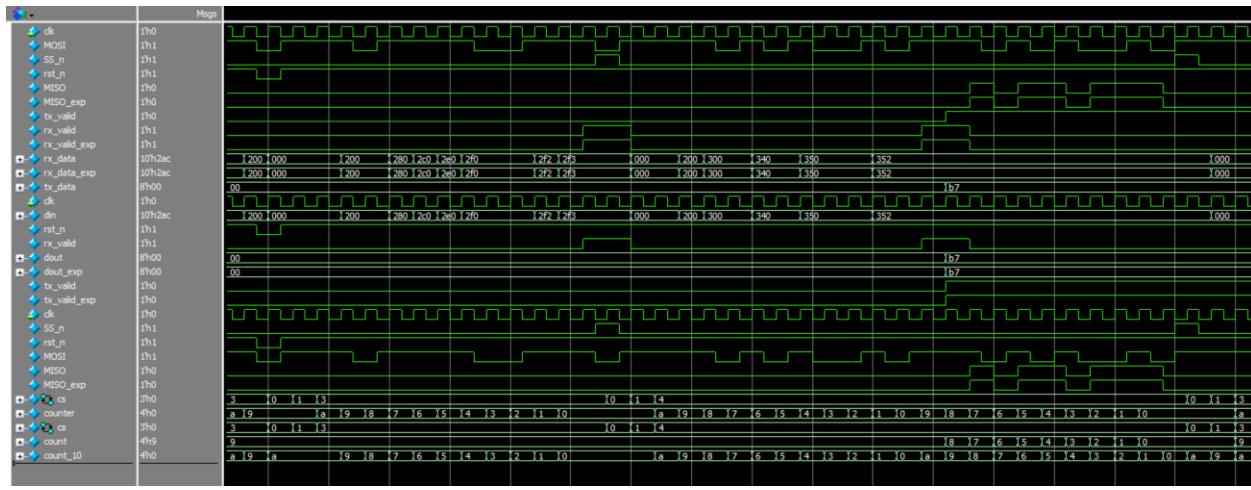
Reset sequence:



Write Sequence:



Read Address Sequence:



Read Data Sequence:

