Ministry of Communications and Information Technology

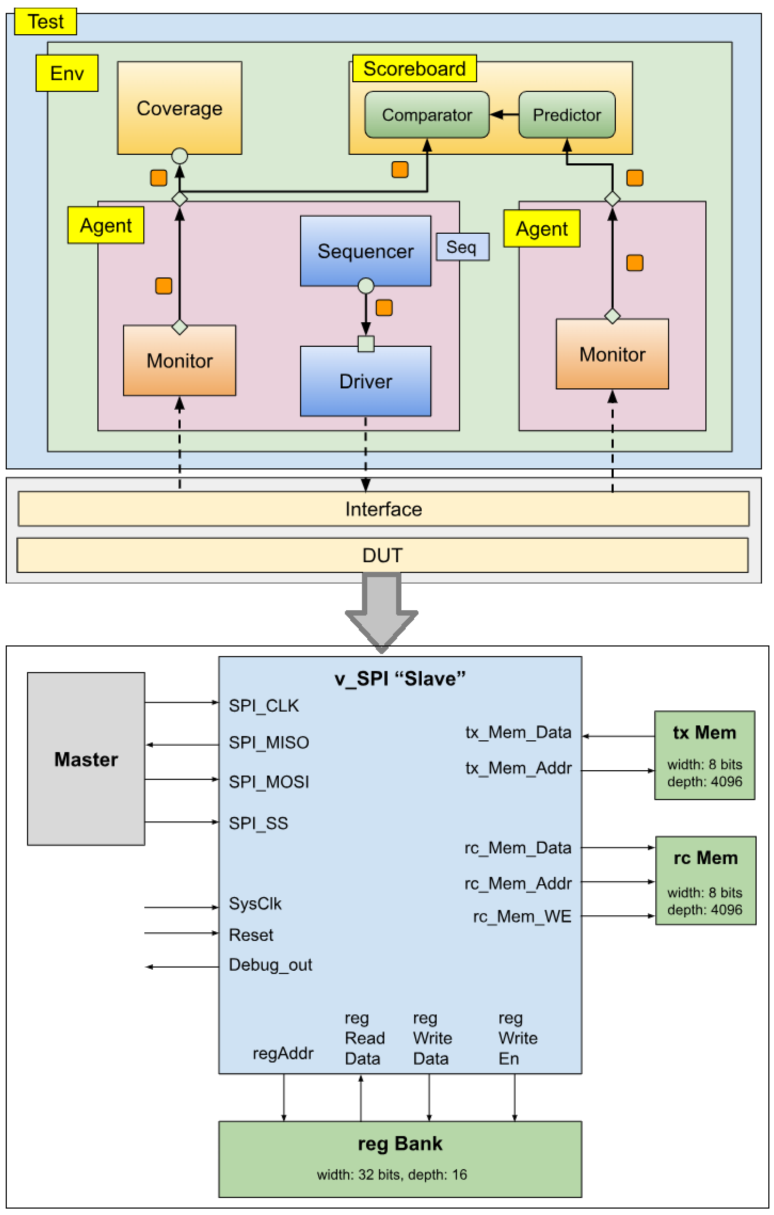
eme egypt makes electronics

The American University in Cairo

CENTER OF NANOELECTRONICS & DEVICES

Functional Verification of SPI Slave IP

using SVA and UVM



This is the submission of the Final project of the subject:
CND-212 (Digital Testing and Verification),
of the ”Advanced Digital IC Design and Verification” training program,
organized by the Center of Nanoelectronics and Devices (CND),
at the American University in Cairo (AUC),
in collaboration with the Ministry of Communications and Information Technology in Egypt (MCIT).

Submitted to:
Dr. Omar El-Dash
Eng. Amr Abd-ElAzim

Submitted By:

Name	**ID**
Atef Amin Itani	V23010377
Mostafa Ahmed Gaber	V23010491
Omar Wael Ahmed	V23009980
Ahmed Hossam Eldin	V23010393
Ahmed Kamal	V23010450
Manar Kamel Hussein	V23010646

Acknowledgments

We want to thank

Dr. Omar ElDash, Eng. Amr Abd-ElAzim, thank you for your efforts and guidance during the lectures and labs.

Mr. Roushdy, Mr. Ehab, and Mr. Ramy thank you for your overwhelming hospitality and help at the labs.

TABLE OF CONTENTS

List of Figures

List of Tables

1. DUT

1.1. About the Project

In this project, it is intended to test an SPI slave IP using systemVerilog incorporating features such as interfaces, classes, constrained-random stimulus generation, and coverage-driven verification methodologies. This involved using an actual IP or an abstract model as the test assumed a black box approach. It is required to show different test scenarios for the Slave IP to verify the functionality of the design, and to show the modularity and the reusability of the test environment. Simulation Results and coverage metrics need to be reported. We chose to verify the IP using UVM also and compare between the two methods: Assertion based and UVM.

1.2. RTL Choice

We had to choose one of three different SPI slave IPs, after assessing the RTLs, we chose the IP of: ”mjlyons” vSPI [`https://github.com/mjlyons/vSPI`], as it was hardware proved on FPGA, the code was clean, it had three different built testbenches for three different scenarios.

1.3. SPI Architecture

A good knowledge of the RTL is essential for functional verification, so we concluded this block diagram to represent the architecture of the SPI slave IP as shown below.

The RTL represents the slave block only, hence the memories, regbank and master should be provided by the user.

The reader could refer to the ReadMe in the author’s repo to understand the protocol, however we added an FSM state diagram and breif protocol description.

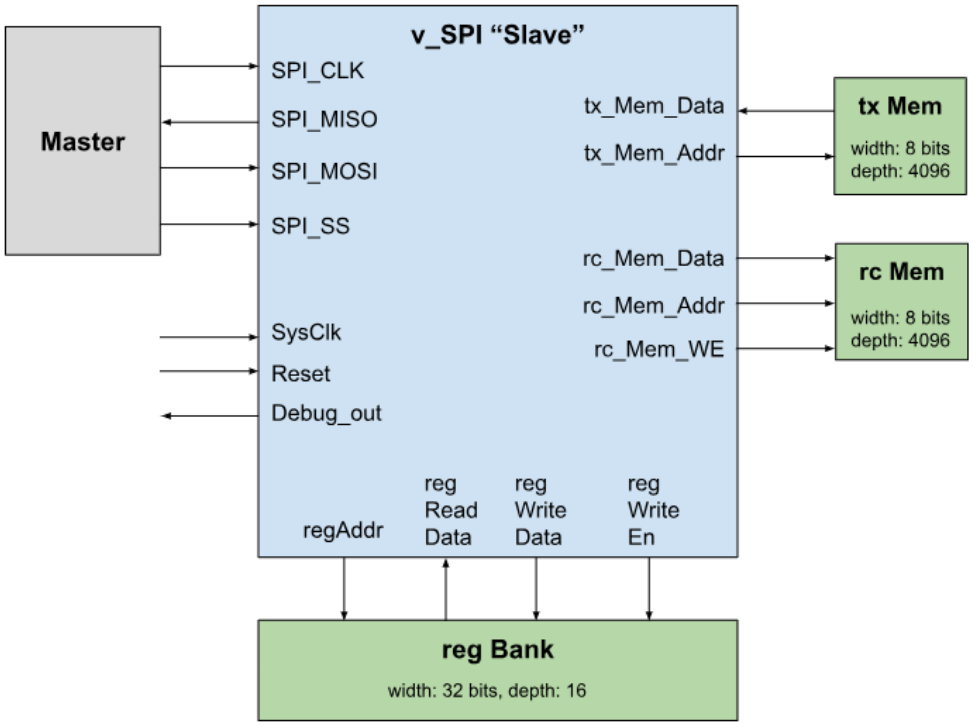


Figure 1.1: Architecture

List of DUT inputs/outputs:

#	Category	DUT Signal	Direction
1	System	Reset	Input
2		SysClk	Input
3		debug_out	Output
4	SPI main signal	SPI_CLK	Input
5		SPI_MISO	Output
6		SPI_MOSI	Input
7		SPI_SS	Input
8	TX Buffer	txMemAddr	Output
9		txMemData	Input
10	RC Buffer	rcMemAddr	Output
11		rcMemData	Output
12		rcMemWE	Output
13	Cmd Bank of Registers	regAddr	Output
14		regReadData	Input
15		regWriteEn	Output
16		regWriteData	Output

Table 1.1: DUT Signals

1.4. Finite State Machine

Further understanding of the FSM is essential for assertions and scoreboard design. It is detailed in the table below and the state diagram.

State	**Description**	**Trigger**	**Address**
Get CMD	Read first byte on MOSI and save it to cmd	at Reset or packet start	N/A
Reading	pass every 8 bits from MOSI to rcMemData	cmd = read_start or read_more	reset to zero at Get_CMD state and incremented at Reading state
Writing	pass each byte on txMemData to MISO, bit by bit	cmd = write_start or write_more	reset to zero at Get_CMD state and incremented at Writing state
Build word	pass 32 bits from MOSI to regWriteData then return to Get_CMD state	cmd[7:6] = 2'b11	last 4 bits in cmd
Send word	pass the 4 bytes in regReadData to MISO, bit by bit, then return to Get_CMD state	cmd[7:6] = 2'b10	last 4 bits in cmd

Table 1.2: FSM

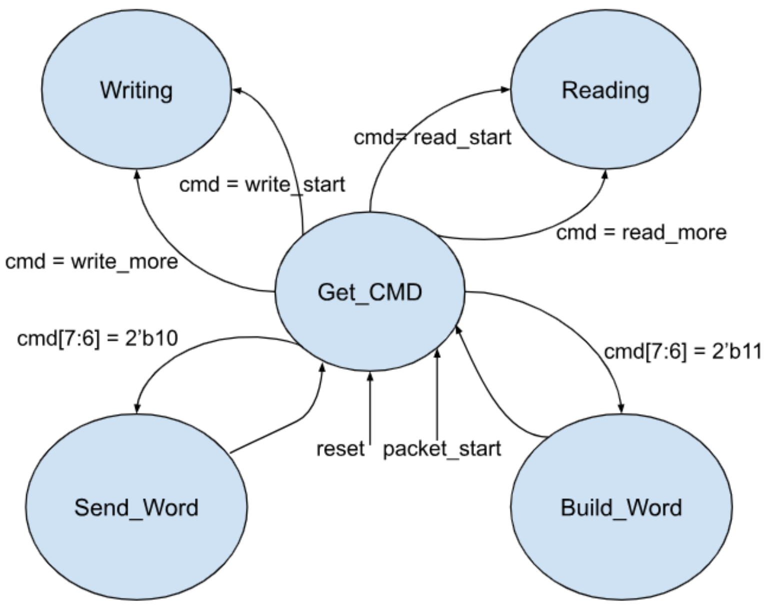


Figure 1.2: FSM

1.5. SPI Protocol

After inspecting the block and state diagrams we add the following notes:

- Master, regbank, rc & tx SRAMs are not provided with the RTL, they should be provided from the user.
- Communication is simplex as the addresses change per one state only, meaning at reading state MISO has garbage data and at writing MOSI has garbage data.
- MOSI/MISO will keep reading/writing forever unless system is reset or slave select drops.
- User should note that the regbank will read/write one word then go to initial state (get_cmd), hence the first received byte on MOSI, after each word, should be of known value as it represents the command.
- Interrupt state is found in RTL but is not implemented.
- The system is single slave.
- Sysclk should always be faster than SPI_CLK.
- The clock phase is with the rising edge and clock polarity is zero.

1.6. The Tool

We used Questa Sim 2021.1 for compilation and simulation.
The version of UVM package: uvm-1.1b You can download it from here: `https://www.accellera.org/downloads/standards/uvm`

2. SVA

The first test we will build is assertion based, using system verilog features like: assert property, sequences, constrained randomization and coverage. First, we will design the test plan that will guide us in writing the assertions and coverage parts. Also, we will need to build an interface and test bench including constrained randomization for the stimulus data. Lastly, we will connect all these with the dut on the top module.

2.1. Test Plan

After looking at the signals, we categorized them into 4 main categories: rc Mem, tx Mem, register bank, System signals. Hence we will build three test scenarios covering all the signals. The test items will be of different types, like: immediate assertions, concurrent assertions, coverpoints and covergroups, as shown in the table below:

#	Type of test	Name in code	Test Item	Description	Associated Signals
rc_Buffer					
1	Immediate Assertion	rcMem_Data	tb currByte to rcMem Data check	checks if each byte in the test bench is passed correctly to rcMemData every 8 SPI_CLKs, when SPI_SS = 0	rcMemData, SPI_SS, SPI_CLK
2	Concurrent Assertion	rcMem_Addr	rcMemAddr increment check	checks if rcMemAddr is incremented every 8 SPI_CLKs when rcMemWE is high	rcMemAddr, rcMemWE, SPI_CLK
3	Concurrent Assertion	rcMem_WE	rcMem write enable check	checks if rc_Wr_en is high when first byte in packet = CMD_READ_START	rcMemWE, SPI_CLK
tx_Buffer					
4	Concurrent Assertion	tx_Mem_Data	txMemData to MISO check	checks if each bit in tx_Mem_Data is passed correctly to MISO, when the state is SEND or WRITE	txMemData, SPI_SS, SPI_CLK
5	Immediate Assertion	txMem_Addr	txMemAddr increment check	checks if address is incremented every 8 SPI_CLKs when the state is SEND or WRITE	txMemAddr, SPI_SS, SPI_CLK
6	Concurrent Assertion	reg_Read_Data	regReadData check	at Send state, checks if regReadData is passed to 4 bytes of MISO	regReadData, SPI_SS, SPI_CLK, SPI_MISO
7	Concurrent Assertion	reg_Write_Data	regWriteData check	at regWriteEn, checks if regWriteData = last 4 bytes from MOSI	regWriteData, SPI_SS, SPI_CLK

Register bank					
8	Concurrent Assertion	reg_Write_En	regWriteEn check	at Build state, checks if regWriteEn = 1 after four consecutive bytes	regWriteEn, SPI_CLK, rcMemData
9	Immediate Assertion	reg_Addr	regAddr check	checks if address in regWriteData is passed correctly to regAddr when SPI_SS drops	regAddr, rcMemData
System					
10	Concurrent Assertion	reset_addr	Buffers addresses reset check	When Reset = 1 : rcMemAddr = 0 txMemAddr = 0	Reset, rcMemAddr, txMemAddr
11	Concurrent Assertion	debug	debug_out check	checks if byte to MOSI is passed correctly to debug_out every 8 SPI_CLKs	debug_out, SPI_CLK, SPI_SS
Coverages					
12	cover point	cover_rc_ addresses	rc_Bufer addresses	values:0:4095	rcMemAddr
13	cover point	cover_tx_ addresses	tx_Bufer addresses	values:0:4095	txMemAddr
14	cover group	cg_reg_addr	register addresses	values:0:15	regAddr
15	cover property	bit7_is_one, bit7_is_zero	7th Bit in Command	values: 0,1	SPI_SS, SPI_MOSI
16	cover property	bit6_is_one, bit6_is_zero	6th bit in Command	values: 0,1	SPI_SS, SPI_MOSI
17	cover group	cg_cmd	Cover all commands	values: 1,2,3,4,5	SPI_SS, SPI_CLK

Table 2.1: Test Plan

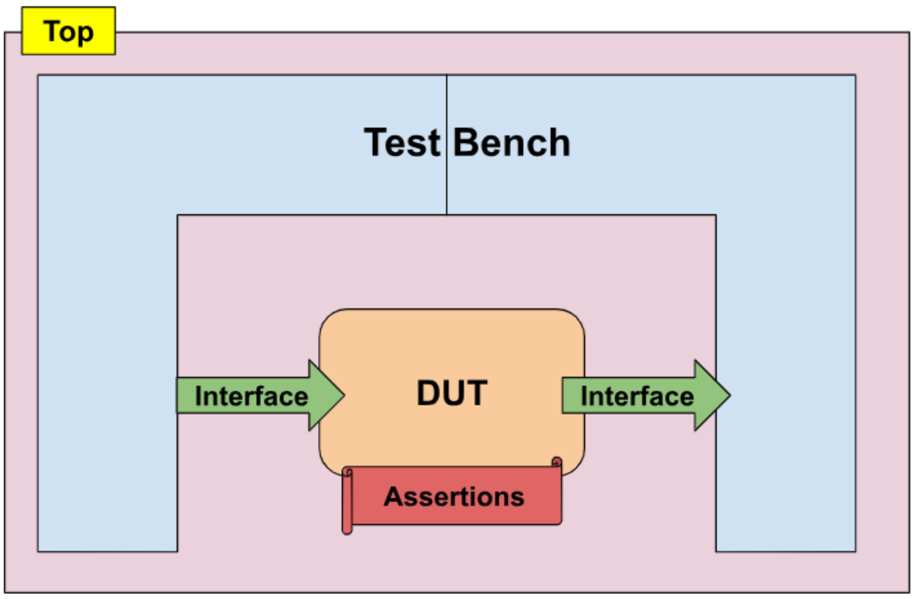


Figure 2.1: SVA top view

2.2. Top module

```
module top();

//Main clock
bit SysClk;
always begin
    #20 SysClk = ~SysClk;
  end

 Intf intf(SysClk);

 spiifc uut (
    .Reset      (intf.Reset),
    .SysClk     (SysClk),
    .SPI_CLK    (intf.SPI_CLK),
    .SPI_MISO   (intf.SPI_MISO),
    .SPI_MOSI   (intf.SPI_MOSI),
    .SPI_SS     (intf.SPI_SS),
    .txMemAddr    (intf.txMemAddr),
    .txMemData    (intf.txMemData),
    .rcMemAddr    (intf.rcMemAddr),
    .rcMemData    (intf.rcMemData),
    .rcMemWE    (intf.rcMemWE),
    .regAddr    (intf.regAddr),
    .regReadData  (intf.regReadData),
    .regWriteEn   (intf.regWriteEn),
    .regWriteData(intf.regWriteData),
    .debug_out    (intf.debug_out)
    );

 tb tb1(intf.TB);

 bind uut assertions SVA_inst (Reset,
   SysClk,
   SPI_CLK,
   SPI_MOSI,
   SPI_SS,
   rcMemWE);

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end

endmodule : top
```

Listing 1: Top module

2.3. Test Bench design

We will test four scenarios:

- Byte sent from MOSI to rcMemData with rcMemAddr incrementing.
- Byte sent from txMemData to MISO with txMemAddr incrementing.
- Word (4 bytes) sent from MOSI to regWriteData with regAddr changing.
- Word (4 bytes) sent from regReadData to MISO with regAddr changing.

We built on the test benches of the author found here: `https://github.com/mjlyons/vSPI/tree/master/test/spi_base`

Data will be randomized with constraints to fulfill the commands needed with the testing scenarios, as shown in the code below:

2.3.1. Randomization

```
//Data Randomization
  class ByteRandomizer;

  rand bit [7:0] data_in[bytes_count] ;
  rand bit [7:0] cmnd[pkts_count];
  rand bit [7:0] reg_cmnd[40];
  rand bit [31:0] regReadData[32];

  constraint read_first {foreach (cmnd[i])
                (i<10) -> cmnd[i] == 8'h01;} // let first ten packets read
   data only.
  constraint cmnd_value{foreach (cmnd[i])
      cmnd[i] dist{8'h01:=10, //CMD_READ_START 40
             8'h02:=10, //CMD_READ_MORE 10
             8'h03:=10, //CMD_WRITE_START 20
             8'h04:=10  //CMD_WRITE_MORE 20
             }; }

  constraint reg_cmnds {foreach (reg_cmnd[i])
                reg_cmnd[i] dist{[8'hC0:8'hCF]:=16,  //CMD_BUILD_WORD
                         [8'h80:8'h8F]:=16};}//CMD_SEND_WORD

  endclass
```

Listing 2: Randomization

2.3.2. Data Stimulus

```
module tb  #(parameter
        AddrBits = 12,
        RegAddrBits = 4
      )(Intf.TB tb_if);

  parameter bytes_count = 2000;//200k for full cov
  parameter pkts_count = 50; //50 for full cov
  reg [7:0] currRcByte;
  reg [7:0] pastRcByte;
  reg [7:0] currCmnd;
  reg [7:0] pastCmnd;
  reg [7:0] currTxByte;
  int index;
  int pkt_length;

  // Data Randomization goes here

  ByteRandomizer byteRandomizer;

  //task to send currRcByte bit by bit
  task recvByte;
    input   [7:0] rcByte;
    integer       rcBitIndex;
    begin
    //$display("%g - spiifc receiving byte '0x%h'", $time, rcByte);
      for (rcBitIndex = 0; rcBitIndex < 8; rcBitIndex = rcBitIndex + 1)
 begin
        tb_if.cb.SPI_MOSI <= rcByte[7 - rcBitIndex];
        #100;
      end
    end
  endtask

  //Data Stimulus
  initial begin
  // Initialize Inputs
  tb_if.cb.Reset <= 0;

  tb_if.cb.SPI_MOSI <= 0;
  tb_if.cb.SPI_SS <= 1;
  tb_if.cb.txMemData <= 0;

  // Wait for global reset to finish
  #100;
  tb_if.cb.Reset <= 1;
```

```
#100;
tb_if.cb.Reset <= 0;
#100;

// Initialize the randomizer
      byteRandomizer = new();
void'(byteRandomizer.randomize());

pkt_length = bytes_count / pkts_count;

//Testing rc & tx Buffers
for (int i=0; i<pkts_count ; i++) begin

  //Start of Packet
  tb_if.cb.SPI_SS <= 0;

  //Sending the Command
  $display("%g - start of packet - command[%0d] = 0x%h",$time,i,
 byteRandomizer.cmnd[i]);
  pastCmnd = currCmnd;
  currCmnd = byteRandomizer.cmnd[i];
  recvByte(currCmnd);

  //Sending the Data
  for (int j=0 ; j<pkt_length; j++) begin
  index = (i*pkt_length)+j;
  $display("%g - randomized data_in[%0d] = 0x%h",$time,index,
 byteRandomizer.data_in[index]);
  pastRcByte = currRcByte;
  currRcByte = byteRandomizer.data_in[index];
  if(currCmnd == 8'h03 || currCmnd == 8'h04) begin
    recvByte(currRcByte);
    tb_if.cb.txMemData  <= currRcByte; end
  else recvByte(currRcByte);
  end

  //End of Packet
  tb_if.cb.SPI_SS <= 1;
  #1000;

end

//Testing regbank
for (int i=0; i<32 ; i++) begin

  //Start of Packet
  tb_if.cb.SPI_SS <= 0;

```

```
    //Sending the Command
    $display("%g - start of packet - command[%0d] = 0x%h",$time,i,
   byteRandomizer.reg_cmnd[i]);
    pastCmnd = currCmnd;
    currCmnd = byteRandomizer.reg_cmnd[i];
    recvByte(currCmnd);

    //Sending the Data
    for (int j=0 ; j<43; j++) begin
    index = (i*4)+j;
    $display("%g - randomized reg_data_in[%0d] = 0x%h",$time,index,
   byteRandomizer.data_in[index]);
    pastRcByte = currRcByte;
    currRcByte = byteRandomizer.data_in[index];
    recvByte(currRcByte);
    if(currCmnd[7:6] == 2'b10) tb_if.cb.regReadData <= byteRandomizer.
   regReadData[i];
    end

    //End of Packet
    tb_if.cb.SPI_SS <= 1;
    #1000;

  end
  $finish;
  end
endmodule
```

Listing 3: Test Bench

2.4. Interface

The interface will link the signals between the test bench and the DUT, define the directivity of the signals,include the cover groups and points, and include the SPI_CLK stimulus, as shown in the code below:

```
interface Intf #(parameter
        AddrBits = 12,
        RegAddrBits = 4
  )(input bit SysClk);

    // Inputs
    bit Reset;
    bit SPI_CLK;
    bit SPI_MOSI;
    bit SPI_SS;
    logic [7:0] txMemData;
    logic [31:0] regReadData;
    logic [7:0] cmd_recvd; //saves firstbyte of the packet (command)

    // Outputs
    logic SPI_MISO;
    logic [AddrBits-1:0] txMemAddr;
    logic [AddrBits-1:0] rcMemAddr;
    logic [7:0] rcMemData;
    logic rcMemWE;
    logic [7:0] debug_out;
    logic [RegAddrBits-1:0] regAddr;
    logic regWriteEn;
    logic [31:0] regWriteData;

    reg SPI_CLK_en;

    initial begin
        #310
        SPI_CLK_en = 1;
    end

    initial begin
  SPI_CLK = 0;
  SPI_CLK_en = 0;
    end

    always begin
        #10 if (SPI_CLK_en) #40 SPI_CLK = ~SPI_CLK;
    end

    clocking cb @(posedge SysClk);
        output Reset;
  output SPI_CLK;
  output SPI_MOSI;
  output SPI_SS;
  output txMemData;
  output regReadData;

  input SPI_MISO;
  input txMemAddr;
  input rcMemAddr;
  input rcMemData;
  input rcMemWE;
  input debug_out;
  input regAddr;
  input regWriteEn;
  input regWriteData;

    endclocking

    modport TB(clocking cb);
```

Listing 4: Interface

2.5. Coverage

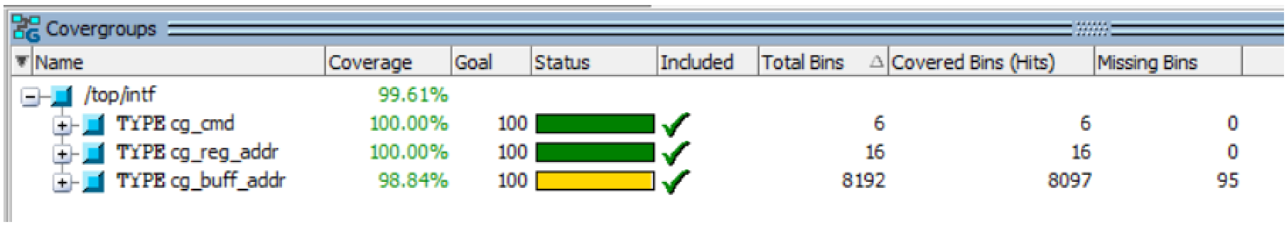
```
//=====================================
//========== Coverage =================
//=====================================

//covering Buffers addresses
 covergroup cg_buff_addr @(posedge SysClk);
  option.per_instance = 1;
  type_option.strobe = 1;
  cover_rc_addresses:  coverpoint uut.rcMemAddr
  { option.auto_bin_max = 4096; }
  cover_tx_addresses: coverpoint uut.txMemAddr
  { option.auto_bin_max = 4096; }
endgroup

  cg_buff_addr cvg_buf_addr = new();

//covering register addresses
 covergroup cg_reg_addr @(posedge SysClk);
  option.per_instance = 1;
  type_option.strobe = 1;
  cover_reg_addresses: coverpoint uut.regAddr
  { option.auto_bin_max = 16; }
endgroup

  cg_reg_addr cvg_reg_addr = new();

//covering CMD_REG_BIT
bit7_is_zero : cover property ( @(posedge SysClk)
  $fell(SPI_SS) |=> first_match($rose(SPI_CLK)) |-> (uut.SPI_MOSI === 1'b0)
   );
bit7_is_one : cover property ( @(posedge SysClk)
  $fell(SPI_SS) |=> first_match($rose(SPI_CLK)) |-> (uut.SPI_MOSI === 1'b1)
   );

//covering CMD_REG_WE_BIT
bit6_is_zero : cover property ( @(posedge SysClk)
  $fell(SPI_SS) |=> first_match($rose(SPI_CLK) [=2]) |-> (uut.SPI_MOSI ===
   1'b0) );
bit6_is_one  : cover property ( @(posedge SysClk)
  $fell(SPI_SS) |=> first_match($rose(SPI_CLK) [=2]) |-> (uut.SPI_MOSI ===
   1'b1) );

//covering all commands
sequence cmd_received;
 @(posedge SysClk)
```

```
  $fell(SPI_SS) ##1 (first_match ($rose(SPI_CLK) [=8])) ;
endsequence: cmd_received

covergroup cg_cmd @(posedge SysClk);
  option.per_instance = 1;
  type_option.strobe = 1;
  cover_commands:  coverpoint cmd_recvd
  { bins CMD_READ_START  = {8'd1};
  bins CMD_READ_MORE    = {8'd2};
  bins CMD_WRITE_START = {8'd3};
  bins CMD_WRITE_MORE   = {8'd4};
  bins CMD_BUILD_WORD   = {[8'hC0:8'hCF]};
  bins CMD_SEND_WORD    = {[8'h80:8'h8F]};
}
 endgroup

 cg_cmd cvg_cmd = new();

always @ (cmd_received, posedge SPI_SS) begin
  if(SPI_SS) cmd_recvd <= 0;
  else begin
  cmd_recvd <= uut.rcMemData ;
  cvg_cmd.sample();
  end
end

//======================================
//========= End of Coverages ==========
//======================================

endinterface
```

Listing 5: Coverage

2.5.1. Coverage result

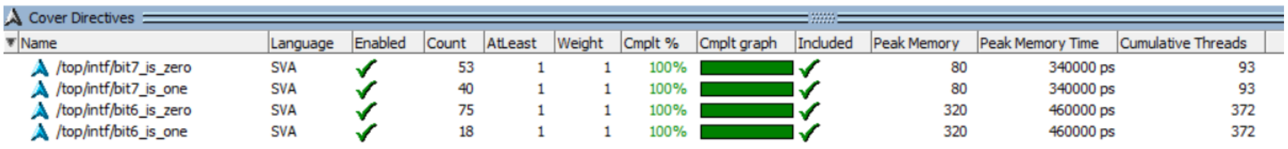
As shown previously in the test bench, we stimulated 200k bytes divided on 50 packets in order to reach high coverage values as shown in the figure below, this large data is mainly to cover the SRAMs addresses, 4096 value per SRAM. And for the register bank we stimulated 40 words.



Covergroups

Name	Coverage	Goal	Status	Included	Total Bins	Covered Bins (Hits)	Missing Bins
/top/intf	99.61%						
TYPE cg_cmd	100.00%	100	🟩	✓	6	6	0
TYPE cg_reg_addr	100.00%	100	🟩	✓	16	16	0
TYPE cg_buff_addr	98.84%	100	🟨	✓	8192	8097	95

Figure 2.2: Cover Groups



Cover Directives

Name	Language	Enabled	Count	AtLeast	Weight	Cmplt %	Cmplt graph	Included	Peak Memory	Peak Memory Time	Cumulative Threads
/top/intf/bit7_is_zero	SVA	✓	53	1	1	100%	🟩	✓	80	340000 ps	93
/top/intf/bit7_is_one	SVA	✓	40	1	1	100%	🟩	✓	80	340000 ps	93
/top/intf/bit6_is_zero	SVA	✓	75	1	1	100%	🟩	✓	320	460000 ps	372
/top/intf/bit6_is_one	SVA	✓	18	1	1	100%	🟩	✓	320	460000 ps	372

Figure 2.3: Cover Points

2.6. Assertions

First we will set multiple sequences and variables for later use in different assertions:

```
module assertions(
  input Reset,
  input SysClk,
  input SPI_CLK,
  input SPI_MOSI,
  input SPI_SS,
  input rcMemWE
  );

parameter AddrBits = 12;
parameter RegAddrBits = 4;

`define CMD_READ_START    8'd1
`define CMD_READ_MORE     8'd2
`define CMD_WRITE_START   8'd3
`define CMD_WRITE_MORE    8'd4
`define CMD_SEND          8'h83


reg state_writing;
reg state_send;
reg state_build;
reg [AddrBits-1:0] pastTxMemAddr;
reg [31:0] currRcWord;
reg [RegAddrBits-1:0]  reg_addr;
reg validByte;

//reg for state build, will be used in assertions
always @(SysClk) begin
    if(intf.cmd_recvd[7] && !SPI_SS) begin// & uut.rcMemData[7] & uut.
    rcMemData[6]) begin
    state_build <= (intf.cmd_recvd[6] ? 1 : 0);
    state_send  <= (~intf.cmd_recvd[6] ? 1 : 0);
    end else begin
    state_build <= 0;
    state_send  <= 0;
    end
end

//setting sequences for multiple usage
sequence valid_Byte;
@(posedge SysClk)
    !SPI_SS throughout (first_match ($rose(SPI_CLK) [=8])) ;
```

```
endsequence:valid_Byte

sequence building_word;
@(posedge SysClk)
   // (!SPI_SS && state_build) throughout (first_match ($rose(SPI_CLK)
   [*8])) ;
   $fell(SPI_SS) ##1 valid_Byte ##1 (intf.cmd_recvd[7:6] == 2'b11);
endsequence:building_word
```

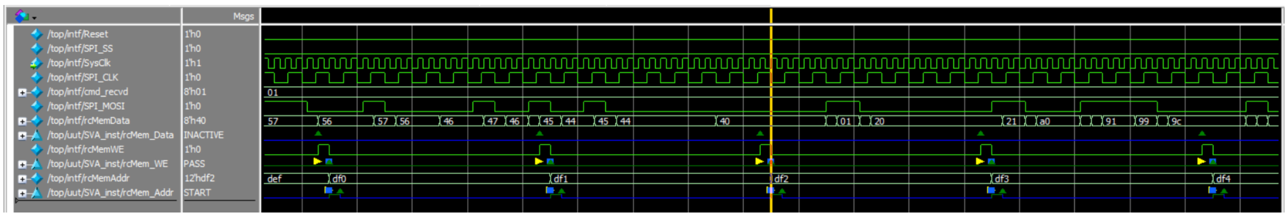
Listing 6: Assertions

2.6.1. rc Buffer Assertions

```
//====================================
//======= rc Buffer Assertions ========
//====================================
//-1 ---- tb byte to rcMem check ----
always @ (valid_Byte) begin
   rcMem_Data:  assert  (uut.rcMemData == tb1.currRcByte || uut.rcMemData
   == tb1.currCmnd)  else $error("passing currByte to rcMem");
   #800;
  end
//----------------------------------

//-2 ----- rcMemAddr inc check -----
property rcMemAddr_incr;
 @(posedge SysClk) disable iff (Reset)
  rcMemWE |=> (uut.rcMemAddr == ($past(uut.rcMemAddr) + 1));
endproperty: rcMemAddr_incr

rcMem_Addr:  assert property (rcMemAddr_incr) else $error("incrementing
   rcMemAddr");
//----------------------------------

//-3 -------- rcMemWE check --------
property rcMemWE_check;
 @(posedge SysClk)
  rcMemWE |=> valid_Byte |=> (intf.cmd_recvd === `CMD_READ_START) || (intf.
   cmd_recvd === `CMD_READ_MORE) || (intf.cmd_recvd === `CMD_SEND);
endproperty: rcMemWE_check

rcMem_WE:  assert property (rcMemWE_check)  else $error("in enabling rcMemWE
   ");
//----------------------------------
```

Listing 7: rc Buffer Assertions

Random screenshot from the waveform of 5 bytes with assertions labeled:



```
                                   Msgs
/top/intf/Reset                    1'h0
/top/intf/SPI_SS                   1'h0
/top/intf/SysClk                   1'h1
/top/intf/SPI_CLK                  1'h0
/top/intf/cmd_recvd                8'h01
/top/intf/SPI_MOSI                 1'h0
/top/intf/rcMemData                8'h40
/top/uut/SVA_inst/rcMem_Data       INACTIVE
/top/intf/rcMemWE                  1'h0
/top/uut/SVA_inst/rcMem_WE         PASS
/top/intf/rcMemAddr                12'hdf2
/top/uut/SVA_inst/rcMem_Addr       START
```

Figure 2.4: rc Buffer Assertions - waveform

2.6.2. tx Buffer Assertions

```
//======================================
//======= tx Buffer Assertions ========
//======================================

//preparing data for following assertions
always @(posedge SPI_CLK, valid_Byte) begin
  pastTxMemAddr <= $past(uut.txMemAddr);
end

always @(SysClk) begin
  state_writing <= (intf.cmd_recvd == `CMD_WRITE_START) || (intf.cmd_recvd
    == `CMD_WRITE_MORE);
end

//-1 -------- txMem_Data ----------
reg [2:0]i = 3'd7;
always @(valid_Byte) begin
  if(intf.cmd_recvd == 8'h03 || intf.cmd_recvd == 8'h04) begin
    for(int j=0; j<8; j++) begin
    i <= i-1;
    #100;
    end
  end else i <= 7;
  end

property txMem_Data; @(posedge SysClk)
  $rose(SPI_CLK) and (!SPI_SS) and (state_writing == 1'b1) |-> (uut.SPI_MISO
     === uut.txMemData[i]);
endproperty: txMem_Data

tx_Mem_Data: assert property (txMem_Data);
//----------------------------------

//-2 -------- txMem_Addr ----------
property txMemAddr; @(posedge SysClk)
```

```
  $fell(SPI_SS) |=> valid_Byte |=> (state_writing == 1'b1) |=> valid_Byte
   |=> (uut.txMemAddr == ($past(uut.txMemAddr) + 1));
endproperty: txMemAddr

txMem_Addr: assert property (txMemAddr);

//--------------------------------
```

Listing 8: tx Buffer Assertions

Random screenshot from the waveform of 5 bytes with assertions labeled:

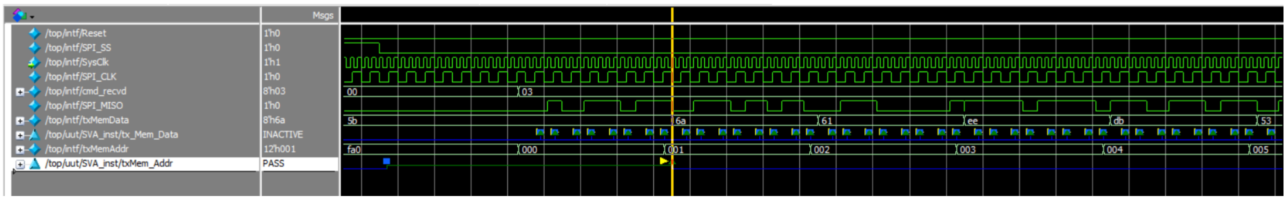

Msgs
/top/intf/Reset 1'h0
/top/intf/SPI_SS 1'h0
/top/intf/SysClk 1'h1
/top/intf/SPI_CLK 1'h0
/top/intf/cmd_recvd 8'h03
/top/intf/SPI_MISO 1'h0
/top/intf/txMemData 8'h6a
/top/uut/SVA_inst/tx_Mem_Data INACTIVE
/top/intf/txMemAddr 12'h001
/top/uut/SVA_inst/txMem_Addr PASS
00
03
5b
6a
61
ee
db
53
fa0
000
001
002
003
004
005

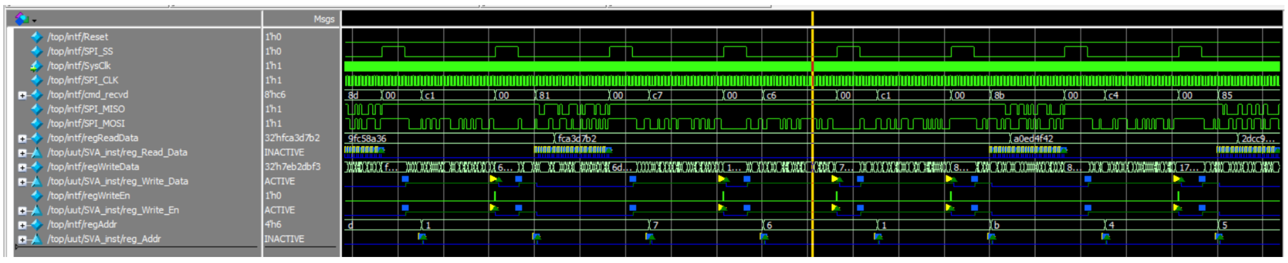
Figure 2.5: tx Buffer Assertions - waveform

2.6.3. reg bank Assertions

```
//====================================
//======= register Assertions ========
//====================================
//-1 -------- regRead_Data ----------
reg [4:0] index = 5'd31;
always @(posedge SPI_CLK) begin
  if(state_send & (!SPI_SS) ) index <= index-1;
  else index <= 5'd32;
end

property regRead_Data; @(posedge SysClk)
  state_send and $rose(SPI_CLK)  |-> uut.SPI_MISO === uut.regReadData[index
   ];
endproperty: regRead_Data

reg_Read_Data: assert property (regRead_Data);
//-----------------------------------

//-2 -------- regWrite_Data ----------
always @(building_word) begin
  for(int i=0; i<4; i++) begin
  #800;
  case(i)
    0: currRcWord[31:24] <= intf.rcMemData;
    1: currRcWord[23:16] <= intf.rcMemData;
    2: currRcWord[15:8]  <= intf.rcMemData;
    3: currRcWord[7:0]   <= intf.rcMemData;
  endcase
  end
end

property regWrite_Data; @(posedge SysClk)
  $fell(SPI_SS)|=> valid_Byte |=> (intf.cmd_recvd[7:6] == 2'b11) |=> valid_Byte [*4] |=> ##3 (uut.regWriteData == currRcWord);
endproperty: regWrite_Data

reg_Write_Data: assert property (regWrite_Data) else $error("reading RegWriteData");
//---------------------------------

//-3 -------- regWrite_En ----------
property regWrite_En;
 @(posedge SysClk)
   $fell(SPI_SS)|=> valid_Byte |=> (intf.cmd_recvd[7:6] == 2'b11) |=> valid_Byte [*4] |=> (uut.regWriteEn == 1'b1);
endproperty: regWrite_En

reg_Write_En: assert property (regWrite_En) else $error("in enabling reg regWriteEn");
//---------------------------------

//-4 -------- regAddr ---------
property regAddr;
 @(posedge SysClk)
  (!uut.state) && uut.rcByteValid && uut.rcByte[7] |=> (uut.regAddr == uut.rcByte[3:0]);
endproperty: regAddr

reg_Addr: assert property (regAddr) else $error("in passing reg addr");
//---------------------------------
```

Listing 9: reg bank Assertions

Random screenshot from the waveform of 7 words with assertions labeled:



```
                                       Msgs
/top/intf/Reset                        1'h0
/top/intf/SPI_SS                       1'h0
/top/intf/SysClk                       1'h1
/top/intf/SPI_CLK                      1'h1
/top/intf/cmd_recvd                    8'hc6
/top/intf/SPI_MISO                     1'h1
/top/intf/SPI_MOSI                     1'h1
/top/intf/regReadData                  32'hfca3d7b2
/top/uut/SVA_inst/reg_Read_Data        INACTIVE
/top/intf/regWriteData                 32'h7eb2dbf3
/top/uut/SVA_inst/reg_Write_Data       ACTIVE
/top/intf/regWriteEn                   1'h0
/top/uut/SVA_inst/reg_Write_En         ACTIVE
/top/intf/regAddr                      4'h6
/top/uut/SVA_inst/reg_Addr             INACTIVE
```

Figure 2.6: reg bank Assertions - waveform

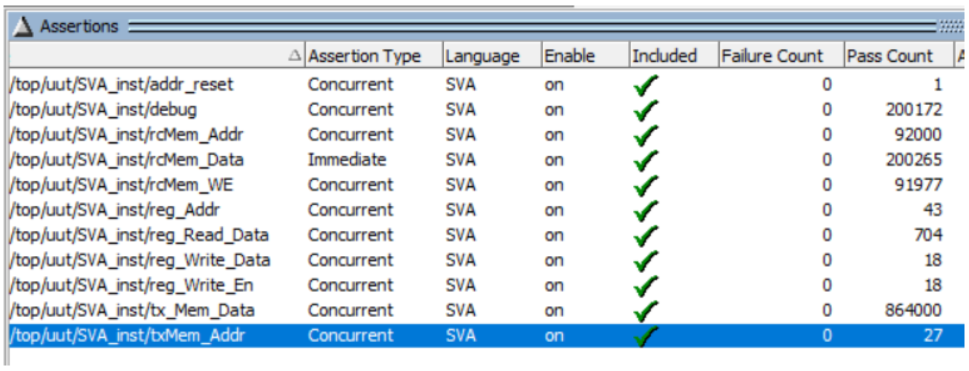
2.6.4. System Assertions

```
//=====================================
//======== System Assertions ==========
//=====================================
//-1 ------ Buffs Addr check -------
property addr_is_reset;
 @(posedge SysClk) $rose(Reset) |=>  (uut.txMemAddr === 12'h000) and (uut.rcMemAddr === 12'h000);
endproperty: addr_is_reset

  addr_reset: assert property (addr_is_reset)  else $error("Buffers Addresses are not reset!");
//---------------------------------


// -2 -------- debug_out check -------
property debug_check;
 @(posedge SysClk) disable iff (Reset)
  (!SPI_SS) |=> valid_Byte   |-> (uut.debug_out === tb1.pastRcByte || uut.debug_out === tb1.currCmnd) ;
endproperty: debug_check

always @ (posedge SPI_CLK) begin
debug:    assert property (debug_check)    else $error("reading debug");
#800;
end
//---------------------------------
```

Listing 10: System Assertions

2.6.5. Assertions Results

It took many hours to reach below result of 100% pass rate as shown below, assertions design required very good knowledge of the DUT signals and flow.



Assertions

	Assertion Type	Language	Enable	Included	Failure Count	Pass Count
/top/uut/SVA_inst/addr_reset	Concurrent	SVA	on	✓	0	1
/top/uut/SVA_inst/debug	Concurrent	SVA	on	✓	0	200172
/top/uut/SVA_inst/rcMem_Addr	Concurrent	SVA	on	✓	0	92000
/top/uut/SVA_inst/rcMem_Data	Immediate	SVA	on	✓	0	200265
/top/uut/SVA_inst/rcMem_WE	Concurrent	SVA	on	✓	0	91977
/top/uut/SVA_inst/reg_Addr	Concurrent	SVA	on	✓	0	43
/top/uut/SVA_inst/reg_Read_Data	Concurrent	SVA	on	✓	0	704
/top/uut/SVA_inst/reg_Write_Data	Concurrent	SVA	on	✓	0	18
/top/uut/SVA_inst/reg_Write_En	Concurrent	SVA	on	✓	0	18
/top/uut/SVA_inst/tx_Mem_Data	Concurrent	SVA	on	✓	0	864000
/top/uut/SVA_inst/txMem_Addr	Concurrent	SVA	on	✓	0	27

Figure 2.7: Assertions results

3. UVM

3.1. Top view

We designed the UVM environment as shown in the figure 3.1, the design took the following main considerations:

- Two agents were used, an active one to drive the interface and a passive one to monitor the inputs of the interface/outputs of the driver, which will be fed to the predictor.
- Coverages were migrated from past code with little edit to fit in the subscriber class.

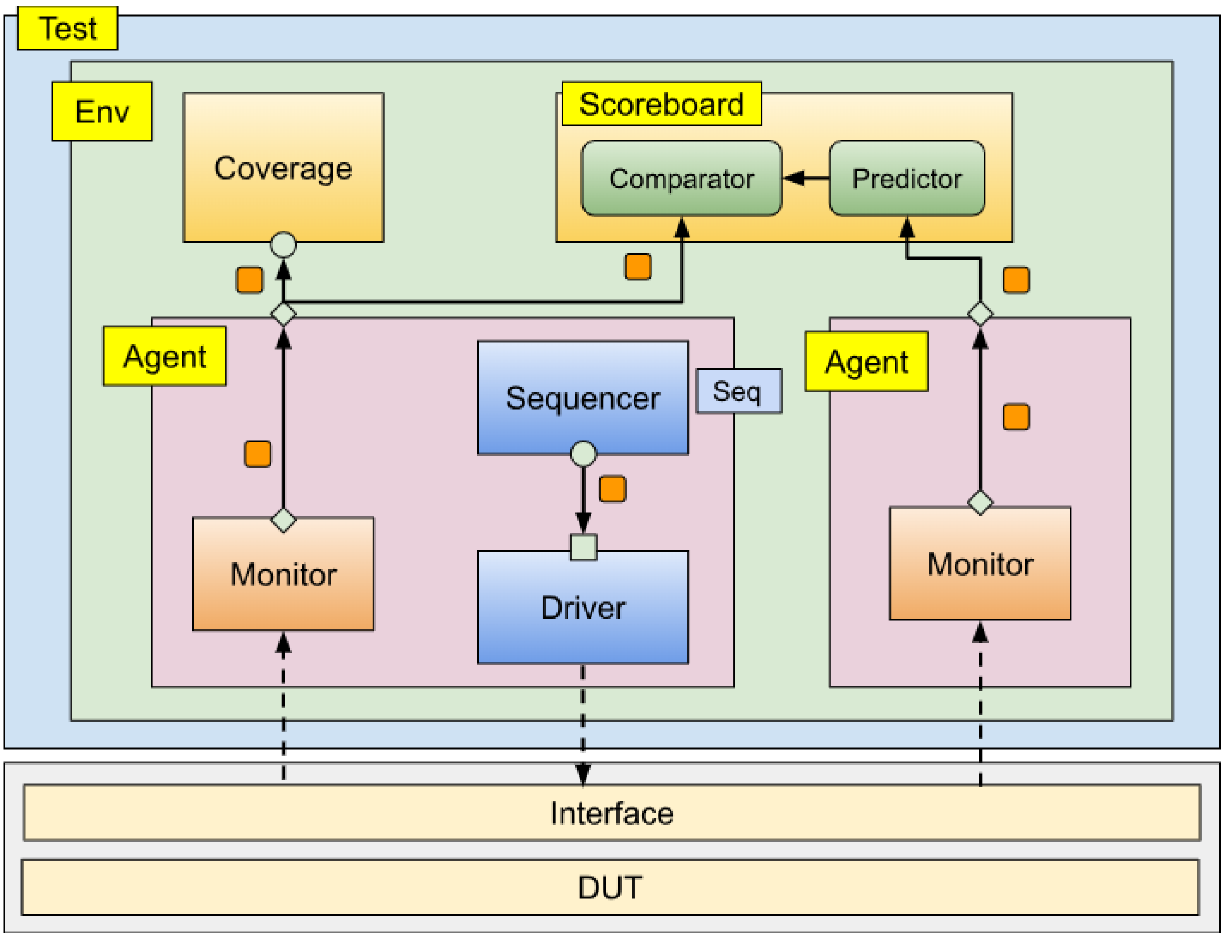


Figure 3.1: UVM top view

- Assertions from past code are not used and a scoreboard is used instead.

- The scoreboard has no reference model, it will predict each output from the input monitor and compare it with the actual output (from the output monitor) to give a PASS/FAIL result per transaction.
- Transactions are sampled on the monitor/interface connection every one valid byte (slave select is low, 8 SPI_CLKs passed).
- TLM connection rules were taken into consideration, between the agents and monitors we used analysis ports, between the driver, sequencer and subscriber we used ports and exports.
- Driver/Sequence handshake protocol was considered at the design of the driver.
- The constrained randomization was put in the Transaction class.
- The main uvm environment was built on the doulos example found in [1]
- For the scoreboard we used the ready-to-plug scoreboard of Sunburst Design found in [2]

3.2. Top Module

```
`timescale 1ns/1ps

`include "uvm_macros.svh"

module top;

  import uvm_pkg::*;
  import my_pkg::*;

  //interface instatiation
  Intf intf();

  //dut instatiation
  spiifc uut(
    .Reset(intf.Reset),
    .SysClk(intf.SysClk),
    .SPI_CLK(intf.SPI_CLK),
    .SPI_MISO(intf.SPI_MISO),
    .SPI_MOSI(intf.SPI_MOSI),
    .SPI_SS(intf.SPI_SS),
    .txMemAddr(intf.txMemAddr),
    .txMemData(intf.txMemData),
    .rcMemAddr(intf.rcMemAddr),
    .rcMemData(intf.rcMemData),
    .rcMemWE(intf.rcMemWE),
    .regAddr(intf.regAddr),
    .regReadData(intf.regReadData),
    .regWriteEn(intf.regWriteEn),
    .regWriteData(intf.regWriteData),
    .debug_out(intf.debug_out)
  );

    initial begin
    uvm_config_db #(virtual Intf)::set(null, "*", "Intf", intf);
    uvm_top.finish_on_completion = 1;
    run_test("my_test");
    $finish;
  end

endmodule: top
```

Listing 11: Top module

3.3. Interface

```
`timescale 1ns / 1ps

interface Intf  ();//input bit SysClk);

  parameter AddrBits = 12;
  parameter RegAddrBits = 4;

  // Inputs
  bit Reset;
  bit SPI_CLK;
  bit SysClk;
  bit SPI_MOSI;
  bit SPI_SS;
  logic [7:0] txMemData;
  logic [7:0] cmnd; //saves firstbyte of the packet (command)
  static logic [7:0] cmd_recvd; //calculates current command
  static logic [7:0] cmd_recvd_temp;
  logic [7:0] data_in_temp; //saves current byte of the array data_in[
     pktlength] from the driver's transaction
  logic get_cmnd; //Flag to indicate the state Get Command
  bit   [7:0] MISO_reg;
  int i;

  // Outputs
  bit SPI_MISO;
  logic [AddrBits-1:0] txMemAddr;
  logic [AddrBits-1:0] rcMemAddr;
  logic [7:0] rcMemData;
  bit rcMemWE;
    logic [7:0] debug_out;
  logic [RegAddrBits-1:0] regAddr;
  bit regWriteEn;
  logic [31:0] regReadData;
  logic [31:0] regWriteData;
    reg SPI_CLK_en;

  //Clocks
  always begin
    #20 SysClk = ~SysClk;
  end

  always begin
    #10
    if (SPI_CLK_en) #40 SPI_CLK <= ~SPI_CLK;
  end

  initial begin
    SPI_CLK <= 0;
    SPI_CLK_en = 0;
    #310
    SPI_CLK_en = 1;
  end

  // Multiple sequences and variables will be used later in the scoreboard predictor.
  sequence receiving_cmnd; @(posedge SysClk)
    $fell(SPI_SS) ;
  endsequence: receiving_cmnd

  sequence cmd_received; @(posedge SysClk)
    $fell(SPI_SS) ##1 (first_match ($rose(SPI_CLK) [=8])) ;
  endsequence: cmd_received

  always @ (cmd_received) begin
    cmd_recvd <= uut.rcMemData ;
  end

  always @ (receiving_cmnd) begin
    get_cmnd <= 1;
    #800;
    get_cmnd <= 0;
  end

  always@(posedge SysClk) begin
    if(cmnd !== 8'h03 && cmnd !== 8'h04 && cmnd[7:6] !== 2'b10) begin
       if(get_cmnd) MISO_reg = (SPI_MISO == 1 ? 8'hff : 8'h00);
       else MISO_reg = 8'h00;
    end
    else if((cmnd == 8'h03 || cmnd == 8'h04 || cmnd[7:6] == 2'b10) && !SPI_SS) begin
          for(i=0; i<8; i++) begin
           @(posedge SPI_CLK);
           MISO_reg[7-i] = SPI_MISO;
          end
    end
  end

endinterface
```

Listing 12: Interface

3.4. Transaction

```
class my_transaction extends uvm_sequence_item;

  `uvm_object_utils(my_transaction)

parameter      pkt_length = 4000;//for full coverage generate 60 pkts of 4
 k bytes length

  //Inputs
  bit          Reset;
  bit            SPI_MOSI;
  bit            SPI_SS;
  rand bit [7:0] txMemData;
  rand bit [7:0] data_in[pkt_length];
  rand bit [7:0] cmnd;
  rand bit [7:0] reg_cmnd;
  rand bit [31:0]regReadData;
  bit      [7:0] data_in_temp;

  //Outputs
  bit            SPI_MISO;
  bit      [7:0] rcMemData;
  bit      [7:0] MISO_reg;//saves MISO output for checking
  bit      [11:0]rcMemAddr;
  bit      [11:0]txMemAddr;
  bit      [3:0] regAddr;
  bit      [7:0] cmd_recvd;
  bit            get_cmnd;
  bit            SPI_CLK;
  bit      [31:0]regWriteData;

  constraint cmnd_value{cmnd
        dist{
           8'h01:=30,  //CMD_READ_START
           8'h02:=20,  //CMD_READ_MORE
           8'h03:=30,  //CMD_WRITE_START
           8'h04:=20}; //CMD_WRITE_MORE
            }

constraint reg_cmnd_value{reg_cmnd
        dist{[8'hC0:8'hCF]:=16,  //CMD_BUILD_WORD
           [8'h80:8'h8F]:=16}; //CMD_SEND_WORD
            }

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

  function string convert2string;//Prints Inputs
    return $sformatf("rcMemData=8'h%0h, rcMemAddr=8'h%0h, cmnd=8'h%0h,
 MISO_reg=8'h%0h, txMemAddr=8‘h%0h, regWriteData=8‘h%0h, regAddr=8‘h%0h",
          rcMemData       , rcMemAddr       , cmnd       , MISO_reg
 , txMemAddr       , regWriteData       , regAddr);
  endfunction

function string output2string;//Prints Outputs
    return $sformatf("rcMemData=8'h%0h, rcMemAddr=8'h%0h, cmnd=8'h%0h,
 MISO_reg=8'h%0h, txMemAddr=8‘h%0h, regWriteData=8‘h%0h, regAddr=8‘h%0h",
                  rcMemData       , rcMemAddr       , cmd_recvd   ,
 MISO_reg       , txMemAddr       , regWriteData      , regAddr);
  endfunction

  function void do_copy(uvm_object rhs);
    my_transaction tx;
    $cast(tx, rhs);
    Reset         = tx.Reset
    SPI_MOSI      = tx.SPI_MOSI;
    SPI_SS        = tx.SPI_SS;
    txMemData     = tx.txMemData;
    data_in       = tx.data_in;
    data_in_temp = tx.data_in_temp;
    cmnd          = tx.cmnd;
    reg_cmnd      = tx.reg_cmnd;
    regReadData   = tx.regReadData;
    rcMemAddr     = tx.rcMemAddr;
    txMemAddr     = tx.txMemAddr;
    cmd_recvd     = tx.cmd_recvd;
    regAddr       = tx.regAddr;
    rcMemData     = tx.rcMemData;
    get_cmnd      = tx.get_cmnd;
    MISO_reg      = tx.MISO_reg;
    SPI_MISO      = tx.SPI_MISO;
    SPI_CLK       = tx.SPI_CLK;
    regWriteData = tx.regWriteData;
  endfunction

  function bit do_compare(uvm_object rhs, uvm_comparer comparer);//called
 in the comparator class in the scoreboard
    my_transaction tx;
    bit status = 1;
    $cast(tx, rhs);
    status &= (rcMemData == tx.rcMemData);
    status &= (rcMemAddr == tx.rcMemAddr);
    status &= (MISO_reg  == tx.MISO_reg);
    status &= (txMemAddr == tx.txMemAddr);
    status &= (regWriteData == tx.regWriteData);
    status &= (regAddr   == tx.regAddr);
    return status;
  endfunction

endclass: my_transaction
```

Listing 13: Transaction

3.5. Sequence

```
class my_sequence    extends uvm_sequence  #(my_transaction);

    `uvm_object_utils(my_sequence)

  my_transaction pkt;
  parameter pkts_count = 60;

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

    task body;
        if (starting_phase != null)
      starting_phase.raise_objection(this,"Start of sequence");

    repeat(pkts_count) begin
    /*1*/ pkt = my_transaction::type_id::create("pkt");
    /*2*/ start_item(pkt);
    /*3*/ if(!pkt.randomize())
        `uvm_error("", $sformatf("Randomization for packet[%0d] failed",
   pkts_count))
    /*4*/ finish_item(pkt);
      end

      if (starting_phase != null)
        starting_phase.drop_objection(this,"End of sequence");
    endtask: body

  endclass: my_sequence
```

Listing 14: Sequence

3.6. Driver

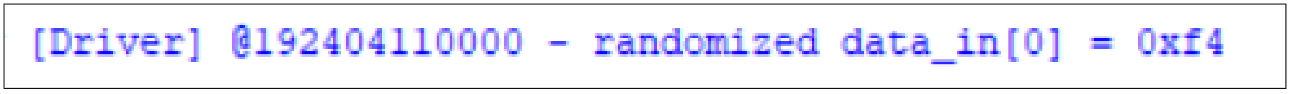
```
class my_driver      extends uvm_driver    #(my_transaction);

    `uvm_component_utils(my_driver)

  my_transaction pkt;
    virtual Intf dut_vi;

  function new(string name, uvm_component parent);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      if(! uvm_config_db #(virtual Intf)::get(this, "", "Intf", dut_vi) )
        `uvm_error("", "uvm_config_db::get failed")
    endfunction

    task run_phase(uvm_phase phase);
      begin

    //Initialize Inputs
    dut_vi.Reset     <= 0;
      dut_vi.SPI_MOSI  <= 0;
      dut_vi.SPI_SS    <= 1;
      dut_vi.txMemData <= 0;

      //Global Reset
      #100;
      dut_vi.Reset  <= 1;
      #100;
      dut_vi.Reset  <= 0;
      #100;

    forever
      begin
      seq_item_port.get_next_item(pkt);//Unblock start_item(start
   randomization,driver is ready to get a trans)

      //Packet start
      dut_vi.SPI_SS    <= 0;

      /*** Testing rc & tx Buffers ***/
      //------------------------------
      //Sending the Command
      `uvm_info("Driver",$sformatf("@%0t *** Start of Packet *** command = 0
   x%h \n",$time,pkt.cmnd),UVM_LOW);
```

```
    dut_vi.cmnd = pkt.cmnd;
    recvByte(pkt.cmnd);

    //Sending the Data
    for (int j=0 ; j<pkt.pkt_length; j++) begin
      `uvm_info("Driver",$sformatf("@%0t - randomized data_in[%0d] = 0x%h\n",$time,j,pkt.data_in[j]),UVM_LOW);
      dut_vi.data_in_temp = pkt.data_in[j];
      if(pkt.cmnd == 8'h03 || pkt.cmnd == 8'h04) dut_vi.txMemData = pkt.data_in[j];
      recvByte(pkt.data_in[j]);
    end

    dut_vi.SPI_SS    <= 1;
    #1000;
    dut_vi.SPI_SS    <= 0;

    /*** Testing register bank ***/
    //-----------------------------
    //Sending the Command
    `uvm_info("Driver",$sformatf("@%0t *** Start of Packet *** command = 0x%h \n",$time,pkt.reg_cmnd),UVM_LOW);
    dut_vi.cmnd = pkt.reg_cmnd;
    recvByte(pkt.reg_cmnd);

    //Sending the Data
    for (int j=0 ; j<4; j++) begin
      `uvm_info("Driver",$sformatf("@%0t - randomized data_in[%0d] = 0x%h\n",$time,j,pkt.data_in[j]),UVM_LOW);
      dut_vi.data_in_temp = pkt.data_in[j];
      if(pkt.reg_cmnd == 8'h80) dut_vi.regReadData = pkt.regReadData;
      recvByte(pkt.data_in[j]);
    end

    //Packet finish
    dut_vi.SPI_SS    <= 1;
    #1000;

    seq_item_port.item_done();//Unblock finish_item(go for next seq, driver has finished wiggling the intf)
    end

  end
  endtask: run_phase

  task recvByte;//Send the byte on MOSI bit by bit
  input    [7:0] rcByte;
  integer        rcBitIndex;
```

```
    begin
      // `uvm_info("Driver", $sformatf("%0t - spiifc receiving byte '0x%h'",
    $time, rcByte),UVM_LOW)
      for (rcBitIndex = 0; rcBitIndex < 8; rcBitIndex = rcBitIndex + 1)
   begin
        dut_vi.SPI_MOSI <= rcByte[7 - rcBitIndex];
        #100;
      end
    end
  endtask: recvByte

endclass: my_driver
```

Listing 15: Driver

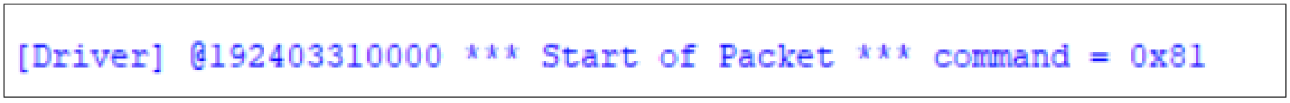
Data_in sample:



```
[Driver] @192404110000 - randomized data_in[0] = 0xf4
```

Figure 3.2: Driver sample

Command sample:



```
[Driver] @192403310000 *** Start of Packet *** command = 0x81
```

Figure 3.3: Driver sample

3.7. Output Monitor

```
class output_monitor extends uvm_monitor;

  `uvm_component_utils(output_monitor)

  virtual Intf dut_vi;
  uvm_analysis_port #(my_transaction) analysis_port;
  my_transaction m_trans;

  function new(string name, uvm_component parent);
    super.new(name, parent);
    analysis_port = new("analysis_port",this);
    m_trans = new();
  endfunction

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);

  if( !uvm_config_db #(virtual Intf)::get(this, "", "Intf", dut_vi) )
        `uvm_fatal("OUTPUT MON", "uvm_config_db::get failed")
    endfunction

  task run_phase(uvm_phase phase);
    m_trans = my_transaction::type_id::create("m_trans");

    forever begin
    @(posedge dut_vi.SysClk && !dut_vi.SPI_SS); //Synchronized with the Input Monitor
      repeat (8) @(posedge dut_vi.SPI_CLK && !dut_vi.SPI_SS);//Sample every one Valid Byte
      @(posedge dut_vi.SysClk);
      m_trans.SPI_CLK   = dut_vi.SPI_CLK;
      m_trans.rcMemData = dut_vi.rcMemData;
      m_trans.rcMemAddr = dut_vi.rcMemAddr;
      m_trans.txMemData = dut_vi.txMemData;
      m_trans.txMemAddr = dut_vi.txMemAddr;
      m_trans.MISO_reg  = dut_vi.MISO_reg;
      m_trans.SPI_MISO  = dut_vi.SPI_MISO;
      m_trans.cmd_recvd = dut_vi.cmd_recvd;
      m_trans.cmnd      = dut_vi.cmnd;
      m_trans.get_cmnd  = dut_vi.get_cmnd;
      m_trans.regReadData = dut_vi.regReadData;
      m_trans.regWriteData = dut_vi.regWriteData;
      m_trans.regAddr   = dut_vi.regAddr;

      `uvm_info("OUTPUT MON", $sformatf("OUTPUTS: rcMemData=8'h%0h,
rcMemAddr=8'h%0h, MISO_reg=8'h%0h, txMemAddr=8'h%0h",
                              m_trans.rcMemData, m_trans.rcMemAddr,
m_trans.MISO_reg, m_trans.txMemAddr), UVM_LOW)
      analysis_port.write(m_trans);
    end
  endtask

  endclass: output_monitor
```

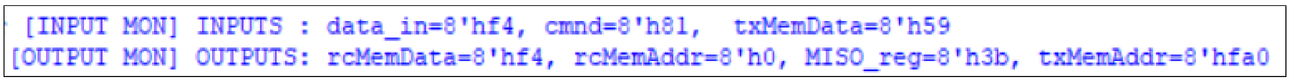
Listing 16: Output Monitor

3.8. Input Monitor

```
class input_monitor  extends uvm_monitor;

  `uvm_component_utils(input_monitor)

  virtual Intf dut_vi;
  uvm_analysis_port #(my_transaction) analysis_port;
  my_transaction m_trans;

  function new(string name, uvm_component parent);
    super.new(name, parent);
    analysis_port = new("analysis_port",this);
    m_trans = new();
  endfunction

  function void build_phase(uvm_phase phase);
      super.build_phase(phase);
    if( !uvm_config_db #(virtual Intf)::get(this, "", "Intf", dut_vi) )
        `uvm_fatal("INPUT MON", "uvm_config_db::get failed")
    endfunction

  task run_phase(uvm_phase phase);
    m_trans = my_transaction::type_id::create("m_trans");

    forever begin
    @ (posedge dut_vi.SysClk && !dut_vi.SPI_SS); //Synchronized with Output
Monitor
      repeat (8) @(posedge dut_vi.SPI_CLK && !dut_vi.SPI_SS);//Sample every
one Valid Byte
      @(posedge dut_vi.SysClk);
      m_trans.SPI_CLK      = dut_vi.SPI_CLK;
      m_trans.rcMemData    = dut_vi.rcMemData;
      m_trans.rcMemAddr    = dut_vi.rcMemAddr;
      m_trans.data_in_temp = dut_vi.data_in_temp;
      m_trans.txMemData     = dut_vi.txMemData;
      m_trans.cmd_recvd     = dut_vi.cmd_recvd;
      m_trans.cmnd          = dut_vi.cmnd;
      m_trans.get_cmnd      = dut_vi.get_cmnd;
      m_trans.MISO_reg      = dut_vi.MISO_reg;
      m_trans.SPI_MISO      = dut_vi.SPI_MISO;
      m_trans.regReadData   = dut_vi.regReadData;
      m_trans.regWriteData  = dut_vi.regWriteData;
      m_trans.regAddr       = dut_vi.regAddr;

      `uvm_info("INPUT MON", $sformatf("INPUTS : data_in=8'h%0h, cmnd=8'h%0h
    , txMemData=8'h%0h", m_trans.data_in_temp, m_trans.cmnd, m_trans.
    txMemData), UVM_LOW)
      analysis_port.write(m_trans);
    end
  endtask

  endclass: input_monitor
```

Listing 17: Input Monitor

Sample of transaction read on Input/Output Monitors:



```
 [INPUT MON] INPUTS : data_in=8'hf4, cmnd=8'h81,  txMemData=8'h59
[OUTPUT MON] OUTPUTS: rcMemData=8'hf4, rcMemAddr=8'h0, MISO_reg=8'h3b, txMemAddr=8'hfa0
```

Figure 3.4: Monitor sample

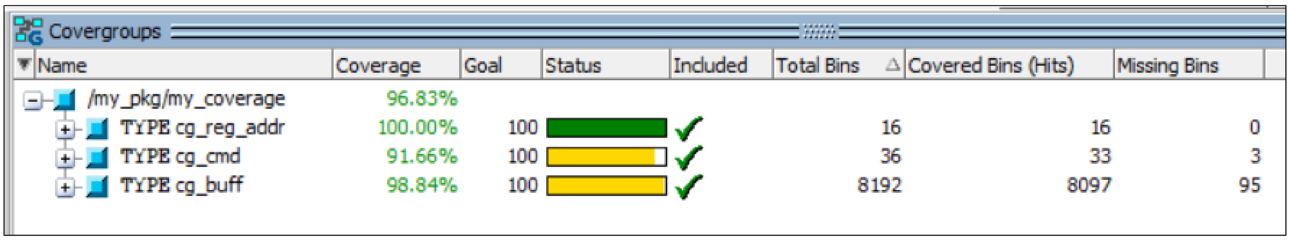
3.9. Coverage

```
class my_coverage    extends uvm_subscriber  #(my_transaction);

 `uvm_component_utils(my_coverage)

 my_transaction pkt;

 //Covering Buffers addresses
 covergroup cg_buff;
 option.per_instance = 1;
    cover_rc_addresses:  coverpoint pkt.rcMemAddr {option.auto_bin_max =
  4096;}
    cover_tx_addresses:  coverpoint pkt.txMemAddr {option.auto_bin_max =
  4096;}
 endgroup

 //Covering all commands
 covergroup cg_cmd;
 option.per_instance = 1;
 cover_commands:  coverpoint pkt.cmd_recvd{
      bins CMD_READ_START   = {8'd1};
      bins CMD_READ_MORE    = {8'd2};
      bins CMD_WRITE_START  = {8'd3};
      bins CMD_WRITE_MORE   = {8'd4};
      bins CMD_BUILD_WORD[] = {[8'hC0:8'hCF]};
      bins CMD_SEND_WORD[]  = {[8'h80:8'h8F]};}
 endgroup

 //Covering register addresses
 covergroup cg_reg_addr;
 option.per_instance = 1;
 cover_reg_addresses: coverpoint pkt.regAddr
 {option.auto_bin_max = 16;}
 endgroup

   function new(string name, uvm_component parent);
     super.new(name, parent);
     cg_buff  = new();
     cg_cmd   = new();
     cg_reg_addr = new();
   endfunction

   function void write(input my_transaction t);
     pkt = t;
     cg_buff.sample();
     cg_cmd.sample();
```

```
      cg_reg_addr.sample();
    endfunction

    function void report_phase(uvm_phase phase);
      `uvm_info(get_type_name(),$sformatf("Coverage of Commands = %3.1f%%",
cg_cmd.get_inst_coverage()),UVM_MEDIUM)
      `uvm_info(get_type_name(),$sformatf("Coverage of Buffer Addresses =
%3.1f%%",cg_buff.get_inst_coverage()),UVM_MEDIUM)
      `uvm_info(get_type_name(),$sformatf("Coverage of Register Addresses =
%3.1f%%",cg_reg_addr.get_inst_coverage()),UVM_MEDIUM)
    endfunction

endclass: my_coverage
```

Listing 18: Coverage

Coverage result:



Covergroups

Name	Coverage	Goal	Status	Included	Total Bins	Covered Bins (Hits)	Missing Bins
/my_pkg/my_coverage	96.83%						
TYPE cg_reg_addr	100.00%	100	🟩	✓	16	16	0
TYPE cg_cmd	91.66%	100	🟨	✓	36	33	3
TYPE cg_buff	98.84%	100	🟨	✓	8192	8097	95

Figure 3.5: Coverage result

3.10. Scoreboard

We used the scoreboard provided by Sunburst Design, below is a block diagram from the paper [2], you can review it for in-depth explanation. All blocks are not edited except for the prediction function in the Predictor class.

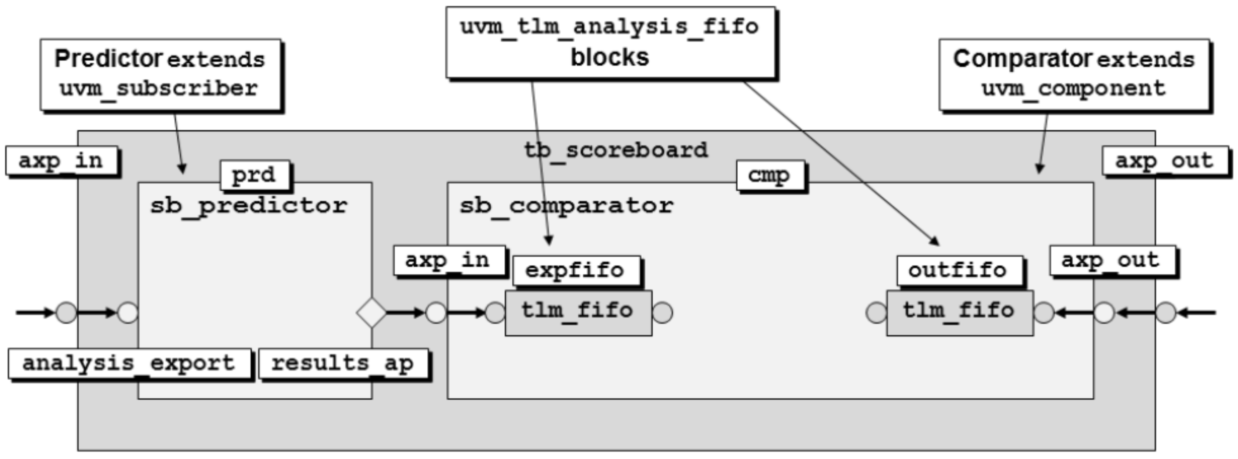


Figure 3.6: Sunburst Scoreboard Architecture

3.10.1. Comparator

```
class my_comparator extends uvm_component;

  `uvm_component_utils(my_comparator)

  uvm_analysis_export #(my_transaction) axp_in;
  uvm_analysis_export #(my_transaction) axp_out;
  uvm_tlm_analysis_fifo #(my_transaction) expfifo;
  uvm_tlm_analysis_fifo #(my_transaction) outfifo;
  my_transaction exp_tr, out_tr;


  function new (string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    axp_in  = new("axp_in", this);
    axp_out = new("axp_out", this);
    expfifo = new("expfifo", this);
    outfifo = new("outfifo", this);

```

```
endfunction

function void connect_phase(uvm_phase phase);
  super.connect_phase(phase);
  axp_in.connect (expfifo.analysis_export);
  axp_out.connect(outfifo.analysis_export);
endfunction

task run_phase(uvm_phase phase);
  my_transaction exp_tr, out_tr;
  forever begin
    `uvm_info("my_comparator run task", "WAITING for expected output", UVM_DEBUG)
    expfifo.get(exp_tr);
    `uvm_info("my_comparator run task", "WAITING for actual output", UVM_DEBUG)
    outfifo.get(out_tr);
    if (out_tr.compare(exp_tr)) begin
      PASS();
      `uvm_info ("PASS", $sformatf("\n Actual   : %s  \n Expected: %s \n", out_tr.output2string(), exp_tr.convert2string()), UVM_LOW)
    end
    else begin
      ERROR();
      `uvm_error("ERROR", $sformatf("\n Actual   : %s \n Expected: %s \n", out_tr.output2string(), exp_tr.convert2string()))
    end
  end
endtask

int VECT_CNT, PASS_CNT, ERROR_CNT;

function void report_phase(uvm_phase phase);
  super.report_phase(phase);
  if (VECT_CNT && !ERROR_CNT)
  `uvm_info(get_type_name(),$sformatf("\n\n\n*** ALL PASSED! - %0d vectors ran, %0d vectors passed ***\n",VECT_CNT, PASS_CNT), UVM_LOW)
  else
  `uvm_info(get_type_name(),$sformatf("\n\n\n*** TEST RESULT - %0d vectors ran, %0d vectors passed, %0d vectors failed ***\n",VECT_CNT, PASS_CNT, ERROR_CNT), UVM_LOW)
endfunction

function void PASS();
  VECT_CNT++;
  PASS_CNT++;
endfunction

```

```
  function void ERROR();
    VECT_CNT++;
    ERROR_CNT++;
  endfunction

  endclass: my_comparator
```

Listing 19: Comparator

3.10.2. Predictor

```
class my_predictor extends uvm_subscriber #(my_transaction);
  `uvm_component_utils(my_predictor)

  uvm_analysis_port #(my_transaction) results_ap;

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

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

  function void write(my_transaction t);
    my_transaction exp_tr;
    exp_tr = my_calc_exp(t);
    results_ap.write(exp_tr);
  endfunction

  extern function my_transaction my_calc_exp(my_transaction t);

  endclass: my_predictor
```

Listing 20: Predictor

3.10.3. Predictor function

```
function my_transaction my_predictor::my_calc_exp(my_transaction t);

  logic [7:0]  rcMemData_out;
  bit   [7:0]  MISO_reg_out;
  static logic [11:0] next_rcMemAddr_out;
  static logic [11:0] next_txMemAddr_out=0;
```

```
logic [11:0] txMemAddr_out;
logic [11:0] rcMemAddr_out;
static bit   [1:0] reg_index=0;
static logic [31:0]reg_Write_Data;
static logic [3:0] reg_Addr_out;

my_transaction tr = my_transaction::type_id::create("tr");
`uvm_info(get_type_name(), t.convert2string(), UVM_HIGH)

txMemAddr_out = next_txMemAddr_out;
rcMemAddr_out = next_rcMemAddr_out;
///*** Outputs Prediction ***///
//STATE_GET_CMD
if(t.get_cmnd) begin
  rcMemData_out = t.cmnd;
  tr.rcMemAddr = rcMemAddr_out;
  next_rcMemAddr_out = 0;
  if(t.cmnd == 8'h03 || t.cmnd[7:6] == 2'b11) next_txMemAddr_out = 0;
  MISO_reg_out = (t.SPI_MISO == 1 ? 8'hff : 8'h00);

//STATE_READING
end else if(t.cmnd == 8'h01 || t.cmnd == 8'h02) begin
  rcMemData_out = t.data_in_temp;
  next_rcMemAddr_out++;
  tr.rcMemAddr = next_rcMemAddr_out;
  MISO_reg_out = 8'h00;
  next_txMemAddr_out = txMemAddr_out;

//STATE_WRITING
end else if(t.cmnd == 8'h03 || t.cmnd == 8'h04) begin
  rcMemData_out = t.data_in_temp;
  next_txMemAddr_out++;
  if(t.get_cmnd) MISO_reg_out = (t.SPI_MISO == 1 ? 8'hff : 8'h00);
  else MISO_reg_out = t.txMemData;

//STATE_BUILD_WORD
end else if (t.cmnd[7:6] == 2'b11) begin
  rcMemData_out = t.data_in_temp;
  reg_Addr_out = t.cmnd & 8'h0F;
  if (reg_index == 0) begin
    reg_Write_Data[31:24] = t.data_in_temp;
    reg_index++;
    end else if (reg_index == 1) begin
    reg_Write_Data[23:16] = t.data_in_temp;
    reg_index++;
    end else if (reg_index == 2) begin
    reg_Write_Data[15:8] = t.data_in_temp;
    reg_index++;
```

```
      end else if (reg_index == 3) begin
      reg_Write_Data[7:0] = t.data_in_temp;
      reg_index = 0;
      tr.regWriteData = reg_Write_Data;
      end

  //STATE_SEND_WORD
  end else if(t.cmnd[7:6] == 2'b10) begin
    rcMemData_out = t.data_in_temp;
    next_txMemAddr_out++;
    reg_Addr_out = t.cmnd & 8'h0F;
      if (reg_index == 0) begin
      MISO_reg_out = t.regReadData[31:24];
      reg_index++;
      end else if (reg_index == 1) begin
      MISO_reg_out = t.regReadData[23:16];
      reg_index++;
      end else if (reg_index == 2) begin
      MISO_reg_out = t.regReadData[15:8];
      reg_index++;
      end else if (reg_index == 3) begin
      MISO_reg_out = t.regReadData[7:0];
      reg_index = 0;
      end
  end

  //Copy all sampled inputs & outputs
  tr.copy(t);
  //Overwrite output values with the calculated ones
  tr.rcMemData = rcMemData_out;
  tr.MISO_reg  = MISO_reg_out;
  tr.txMemAddr = txMemAddr_out;
  tr.regAddr   = reg_Addr_out;
  return(tr);
  endfunction
```

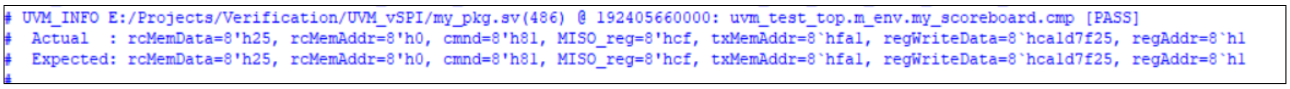
Listing 21: Predictor function

3.10.4. Scoreboard

```
class my_scoreboard  extends uvm_scoreboard;
    `uvm_component_utils(my_scoreboard)
    uvm_analysis_export #(my_transaction) axp_in;
    uvm_analysis_export #(my_transaction) axp_out;
    my_predictor prd;
    my_comparator cmp;

    function new(string name, uvm_component parent);
      super.new( name, parent );
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      axp_in = new("axp_in", this);
      axp_out = new("axp_out", this);
      prd = my_predictor::type_id::create("prd", this);
      cmp = my_comparator::type_id::create("cmp", this);
    endfunction

    function void connect_phase( uvm_phase phase );
      // Connect predictor & comparator to respective analysis exports
      axp_in.connect(prd.analysis_export);
      axp_out.connect(cmp.axp_out);
      // Connect predictor to comparator
      prd.results_ap.connect(cmp.axp_in);
    endfunction
endclass: my_scoreboard
```

Listing 22: Scoreboard

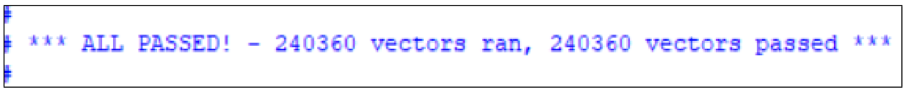
Scoreboard sample per transaction:



```
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(486) @ 192405660000: uvm_test_top.m_env.my_scoreboard.cmp [PASS]
#  Actual  : rcMemData=8'h25, rcMemAddr=8'h0, cmnd=8'h81, MISO_reg=8'hcf, txMemAddr=8`hfa1, regWriteData=8`hca1d7f25, regAddr=8`h1
#  Expected: rcMemData=8'h25, rcMemAddr=8'h0, cmnd=8'h81, MISO_reg=8'hcf, txMemAddr=8`hfa1, regWriteData=8`hca1d7f25, regAddr=8`h1
#
```

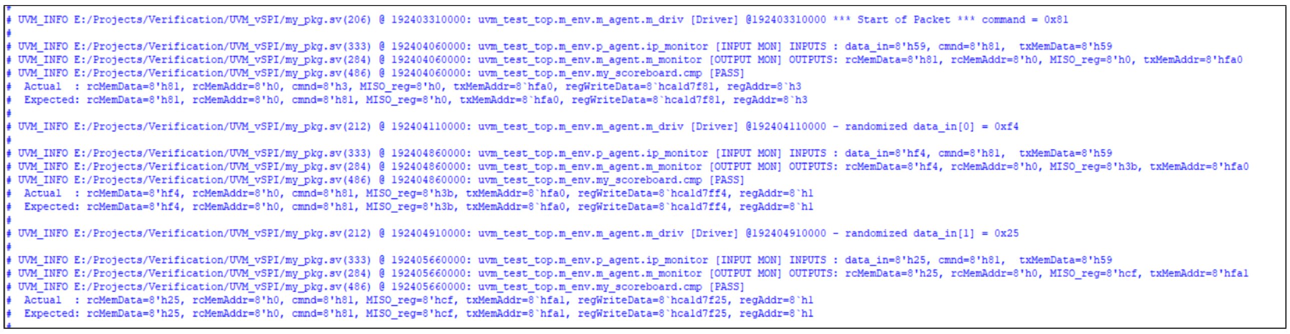
Figure 3.7: Scoreboard result

Scoreboard final result:



```
#
# *** ALL PASSED! - 240360 vectors ran, 240360 vectors passed ***
#
```

Figure 3.8: Scoreboard result



```
#
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(206) @ 192403310000: uvm_test_top.m_env.m_agent.m_driv [Driver] @192403310000 *** Start of Packet *** command = 0x81
#
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(333) @ 192404060000: uvm_test_top.m_env.p_agent.ip_monitor [INPUT MON] INPUTS : data_in=8'h59, cmnd=8'h81,  txMemData=8'h59
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(284) @ 192404060000: uvm_test_top.m_env.m_agent.m_monitor [OUTPUT MON] OUTPUTS: rcMemData=8'h81, rcMemAddr=8'h0, MISO_reg=8'h0, txMemAddr=8'hfa0
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(486) @ 192404060000: uvm_test_top.m_env.my_scoreboard.cmp [PASS]
#  Actual  : rcMemData=8'h81, rcMemAddr=8'h0, cmnd=8'h3, MISO_reg=8'h0, txMemAddr=8`hfa0, regWriteData=8`hca1d7f81, regAddr=8`h3
#  Expected: rcMemData=8'h81, rcMemAddr=8'h0, cmnd=8'h81, MISO_reg=8'h0, txMemAddr=8`hfa0, regWriteData=8`hca1d7f81, regAddr=8`h3
#
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(212) @ 192404110000: uvm_test_top.m_env.m_agent.m_driv [Driver] @192404110000 - randomized data_in[0] = 0xf4
#
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(333) @ 192404860000: uvm_test_top.m_env.p_agent.ip_monitor [INPUT MON] INPUTS : data_in=8'hf4, cmnd=8'h81,  txMemData=8'h59
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(284) @ 192404860000: uvm_test_top.m_env.m_agent.m_monitor [OUTPUT MON] OUTPUTS: rcMemData=8'hf4, rcMemAddr=8'h0, MISO_reg=8'h3b, txMemAddr=8'hfa0
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(486) @ 192404860000: uvm_test_top.m_env.my_scoreboard.cmp [PASS]
#  Actual  : rcMemData=8'hf4, rcMemAddr=8'h0, cmnd=8'h81, MISO_reg=8'h3b, txMemAddr=8`hfa0, regWriteData=8`hca1d7ff4, regAddr=8`h1
#  Expected: rcMemData=8'hf4, rcMemAddr=8'h0, cmnd=8'h81, MISO_reg=8'h3b, txMemAddr=8`hfa0, regWriteData=8`hca1d7ff4, regAddr=8`h1
#
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(212) @ 192404910000: uvm_test_top.m_env.m_agent.m_driv [Driver] @192404910000 - randomized data_in[1] = 0x25
#
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(333) @ 192405660000: uvm_test_top.m_env.p_agent.ip_monitor [INPUT MON] INPUTS : data_in=8'h25, cmnd=8'h81,  txMemData=8'h59
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(284) @ 192405660000: uvm_test_top.m_env.m_agent.m_monitor [OUTPUT MON] OUTPUTS: rcMemData=8'h25, rcMemAddr=8'h0, MISO_reg=8'hcf, txMemAddr=8'hfa1
# UVM_INFO E:/Projects/Verification/UVM_vSPI/my_pkg.sv(486) @ 192405660000: uvm_test_top.m_env.my_scoreboard.cmp [PASS]
#  Actual  : rcMemData=8'h25, rcMemAddr=8'h0, cmnd=8'h81, MISO_reg=8'hcf, txMemAddr=8`hfa1, regWriteData=8`hca1d7f25, regAddr=8`h1
#  Expected: rcMemData=8'h25, rcMemAddr=8'h0, cmnd=8'h81, MISO_reg=8'hcf, txMemAddr=8`hfa1, regWriteData=8`hca1d7f25, regAddr=8`h1
#
```

Figure 3.9: Driver-Monitor-Scoreboard sample

3.11. Other Classes

Below are the other classes of the UVM architecture which has mostly boilerplate code, so we put it lastly

3.11.1. Agent

This is the main agent driving the transactions and monitoring the interface outputs.

```
class my_agent        extends uvm_agent;

    `uvm_component_utils(my_agent)

    my_sequencer   m_seqr;
    my_driver      m_driv;
    output_monitor op_monitor;
    uvm_analysis_port #(my_transaction) analysis_port;

    function new(string name, uvm_component parent);
      super.new(name, parent);
      analysis_port = new("analysis_port",this);
    endfunction

     function void build_phase(uvm_phase phase);
       m_seqr     = my_sequencer::type_id::create("m_seqr", this);
       m_driv     = my_driver::type_id::create("m_driv", this);
       op_monitor = output_monitor::type_id::create("m_monitor", this);
    endfunction

    function void connect_phase(uvm_phase phase);
       m_driv.seq_item_port.connect(m_seqr.seq_item_export);
       op_monitor.analysis_port.connect(analysis_port);
    endfunction

endclass: my_agent
```

Listing 23: Agent

3.11.2. Passive Agent

Used to monitor the driver output (interface input), then pass this transaction through analysis port to the predictor class.

```
class passive_agent  extends uvm_agent;

    `uvm_component_utils(passive_agent)

    input_monitor   ip_monitor;
    uvm_analysis_port #(my_transaction) p_analysis_port;

    function new(string name, uvm_component parent);
      super.new(name, parent);
      p_analysis_port = new("p_analysis_port",this);
    endfunction

    function void build_phase(uvm_phase phase);
      ip_monitor = input_monitor::type_id::create("ip_monitor", this);
    endfunction

    function void connect_phase(uvm_phase phase);
      ip_monitor.analysis_port.connect(p_analysis_port);
    endfunction

endclass: passive_agent
```

Listing 24: Passive Agent

3.11.3. Environment

```
class my_env          extends uvm_env;

    `uvm_component_utils(my_env)

    my_agent      m_agent;
    passive_agent p_agent;
    my_coverage   m_coverage;
    my_scoreboard m_scoreboard;

    function new(string name, uvm_component parent);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      m_agent      = my_agent::type_id::create("m_agent", this);
      p_agent      = passive_agent::type_id::create("p_agent", this);
      m_coverage   = my_coverage::type_id::create("m_coverage", this);
      m_scoreboard = my_scoreboard::type_id::create("my_scoreboard", this);
    endfunction

    function void connect_phase(uvm_phase phase);
      m_agent.analysis_port.connect(m_coverage.analysis_export);
      p_agent.p_analysis_port.connect(m_scoreboard.axp_in);
      m_agent.analysis_port.connect(m_scoreboard.axp_out);
    endfunction

endclass: my_env
```

Listing 25: Environment

3.11.4. Test

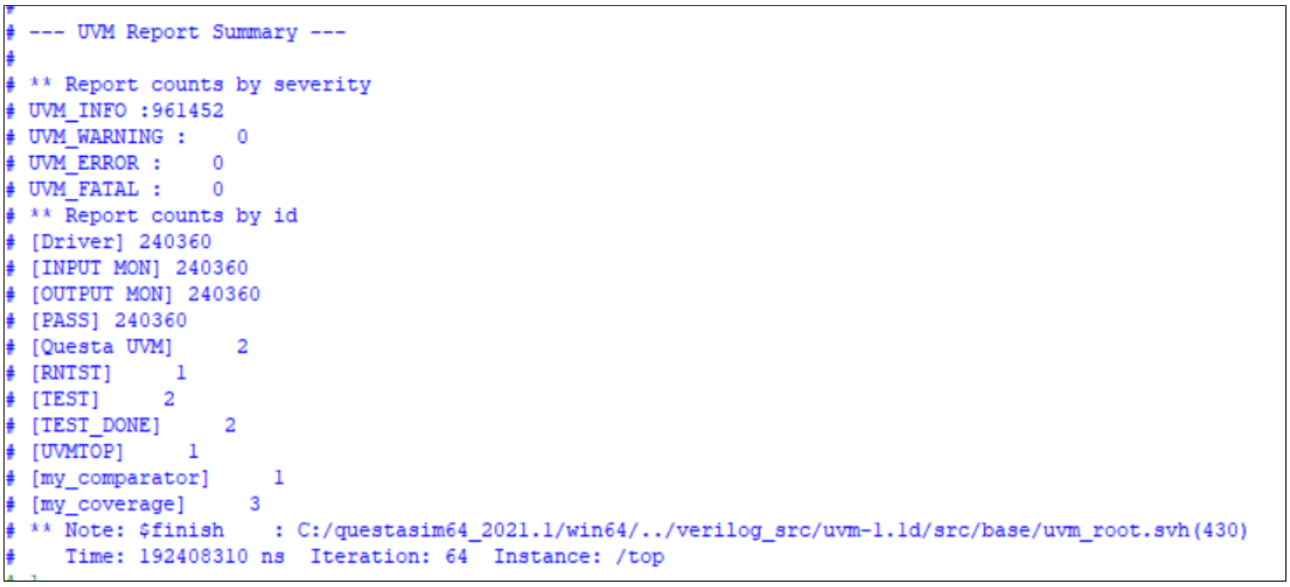
```
class my_test         extends uvm_test;

    `uvm_component_utils(my_test)

    my_env m_env;
    my_sequence seq;

    function new(string name, uvm_component parent);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      m_env = my_env::type_id::create("m_env", this);
    endfunction

    virtual function void end_of_elaboration_phase (uvm_phase phase);
      uvm_top.print_topology ();
    endfunction

    task run_phase(uvm_phase phase);
      //Initialize the system from the driver
      phase.raise_objection(this,"Start of initialization");
      `uvm_info("TEST","Start of initialization",UVM_LOW)
      #310;
      phase.drop_objection(this,"End of initialization");
      `uvm_info("TEST","End of initialization",UVM_LOW)

      //Start the sequence
      seq = my_sequence::type_id::create("seq");
      // seq.set_item_context(this,m_env.m_agent.m_seqr); //for
   randomization stability
      if(!seq.randomize())
       `uvm_error("", "Sequence Randomization failed")
      seq.starting_phase = phase;
      seq.start( m_env.m_agent.m_seqr);
    endtask

endclass: my_test
```

Listing 26: Test

3.12. Final Results

- We managed to achieve 96.8% coverage.
- The scoreboard predicted, compared, and passed all 240,360 transactions.

UVM summary report:



```
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :961452
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Driver] 240360
# [INPUT MON] 240360
# [OUTPUT MON] 240360
# [PASS] 240360
# [Questa UVM]     2
# [RNTST]     1
# [TEST]     2
# [TEST_DONE]     2
# [UVMTOP]     1
# [my_comparator]     1
# [my_coverage]     3
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 192408310 ns  Iteration: 64  Instance: /top
```

Figure 3.10: UVM Report

4. Conclusion

4.1. Comparison between the two methods

- Assertion-based verification consumed so much time compared to the UVM Scoreboard.
- However, assertions enable deep analysis of signals relations with time.
- A 100% passing assertion does not mean the design is functionally verified, as it depends on the written assertion, does it cover all the possible scenarios?
- When covering large data like the SRAMs addresses, it is good to stimulate small data range first and keep the full coverage last step to save runtime.
- Constrained randomization is easy to move from module based environment to UVM, as it is itself is a class.
- UVM seems to be easier to use and more powerful, as you can benefit from ready to use classes without editing the code, except for small functions/tasks, as we did with the Sunburst Design scoreboard.
- Reusability was clear in UVM verification instead of Module-based verification.
- Good knowledge of special classes properties is needed to write a properly working UVM environment, without the need to waste time in debugging, like: sequence/drive handshake, TLM, config database, sequence initialization options.
- After using both methods, I prefer to start my next verification project using UVM, and try not to depend on assertions in the test plan, however for assertions it is still available to run them binded to the interface or DUT.

4.2. Future enhancements

- Randomize the enabling signals like reset and slave select.
- Add more testing scenarios, like sending data to MOSI and txMemData at same time.
- Make the UVM environment parameterized, specially for the transaction length, in the sequence and transaction classes.
- Use two sequences instead of one, one for testing the SRAMs and the other for the regbank. This will save runtime and memory.
- Link actual SRAMs and register to the DUT and sample/send data from their side.
- If needed, test a wider range of the clocks frequencies, instead of fixed values.
- Merge the output and input monitor to one Agent.
- Stimulate out of range data like the reg commands: out of their range 8X and FX.

References

[1] J. Aynsley, “Easier uvm code generator example,” in *https://edaplayground.com/x/5kS8*. Doulos.

[2] C. E. Cummings, “Ovm/uvm scoreboards - fundamental architectures.” Sunburst Design Inc., 2013.

[3] C. Spear and G. Tumbush, “Systemverilog for verification.” Springer, 2012.

[4] D. Mills, “If chained implications in properties weren’t so hard, they’d be easy.” Microchip Technology Inc., 2009.

[5] “Easier uvm video tutorial,” in *https://www.doulos.com/knowhow/systemverilog/uvm/easier-uvm/*. Doulos.