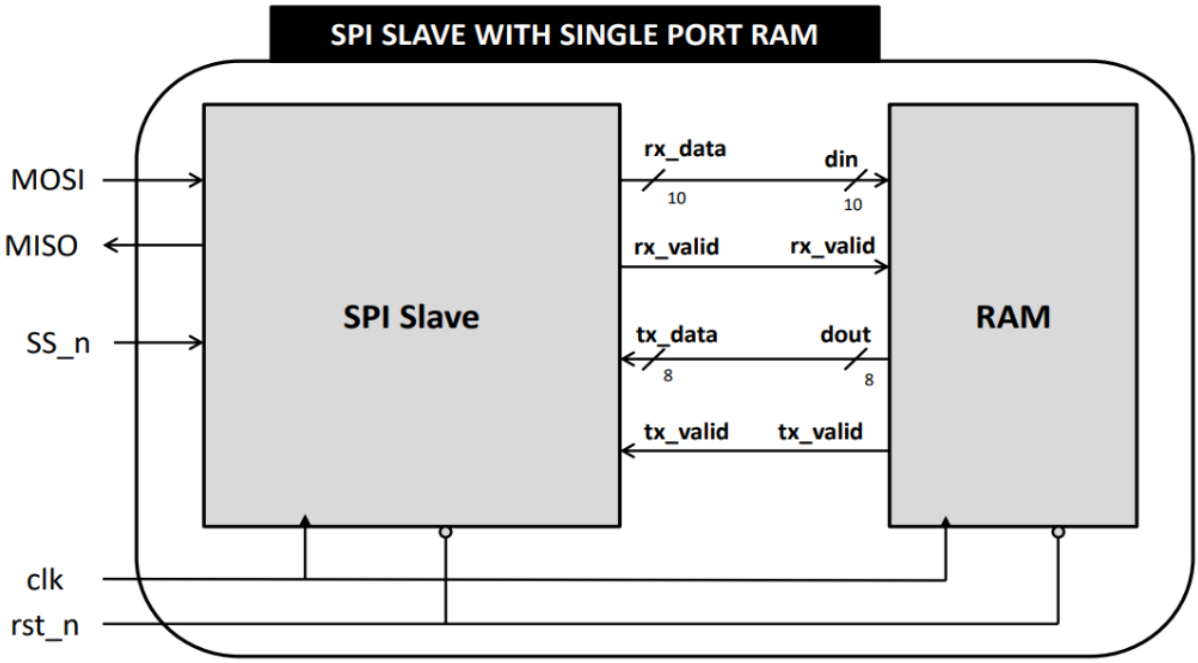
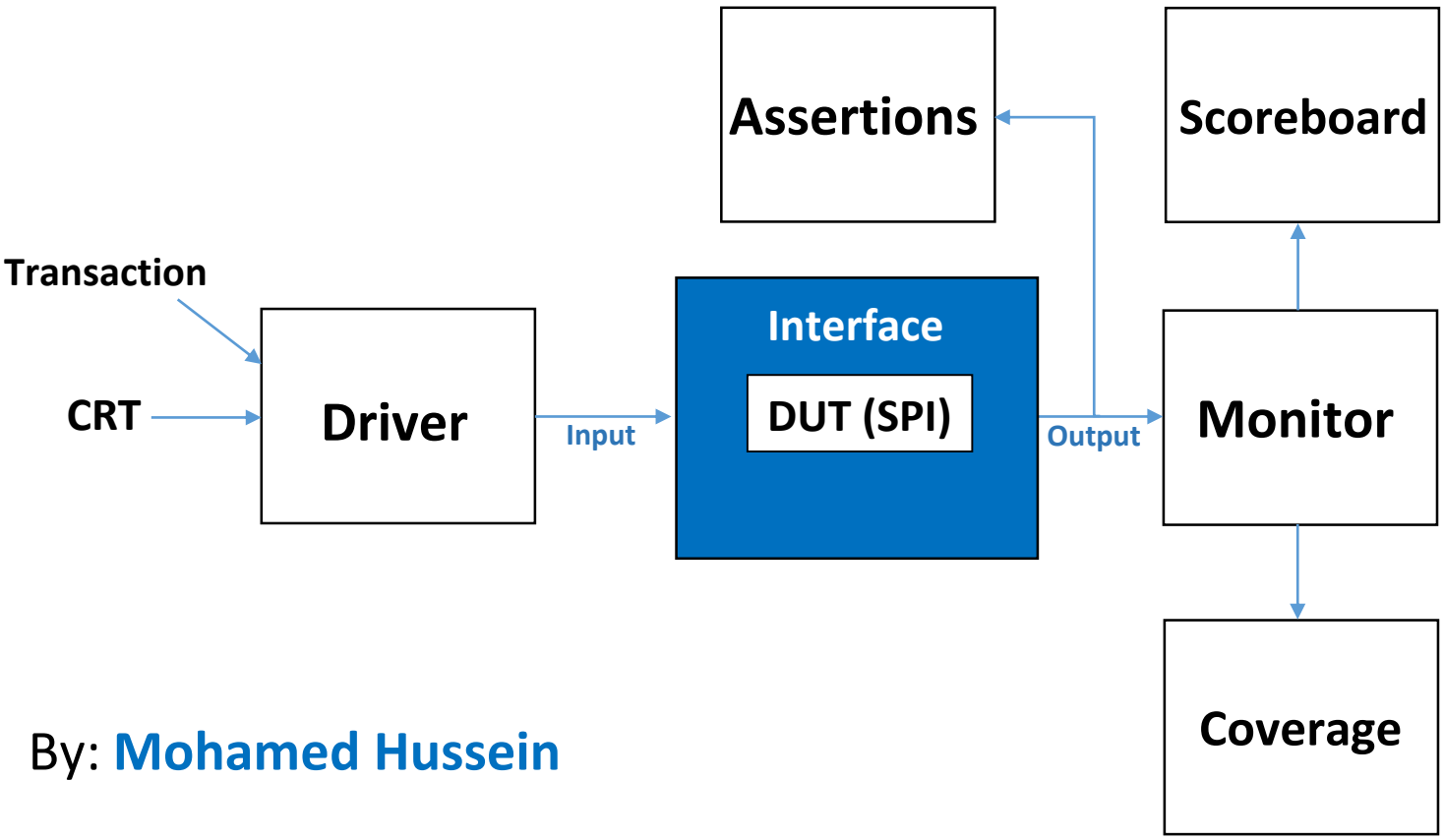
SPI Verification Using SV



Testbench Structure



By: **Mohamed Hussein**

Verification

SPI Wrapper:

- ### Code Coverage

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/spi_wrapper_coverage_pkg/SPI_WRAPPER_coverage		100.00%							
TYPE SPI_WRAPPER_Cross_Group		100.00%	100	100.00...		✓	auto(0)		
CVP SPI_WRAPPER_Cross_Group::cp_ss_n		100.00%	100	100.00...		✓			
INST \/spi_wrapper_coverage_pkg::SPI_WRAPP...		100.00%	100	100.00...		✓		0	
CVP cp_ss_n		100.00%	100	100.00...		✓			
bin state_start		76	1	100.00...		✓			
bin state_end		77	1	100.00...		✓			

- ### Statement Coverage

Statements - by instance (/spi_wrapper_top/dut/SPI)

SPI_SLAVE.v

```
20 always @(posedge clk)
23 cs <= IDLE;
25 cs <= ns ;
29 always @(*) begin
30 ns = cs ;
34 ns = IDLE;
36 ns = CHK_CMD;
E  40 ns = IDLE;
43 ns = WRITE;
45 ns = READ_ADD;
47 ns = READ_DATA;
52 ns = IDLE;
54 ns = WRITE;
58 ns = IDLE;
60 ns = READ_ADD;
64 ns = IDLE;
66 ns = READ_DATA;
72 always @(posedge clk or negedge rst_n) begin
74 counter1 <= 9; //as the first bit entered will be the MSB
75 counter2 <= 7; // as the first bit outted will be the MSB
76 ADD_DATA_checker <= 1; // as reading address first is the default
77 bus <= 0;
78 rx_data <= 0;
79 rx_valid <= 0;
80 MISO  <= 0; // making the default output is zero
85 rx_valid <= 0;
86 counter1 <= 9 ; //to start the same proccess in other states without resetting
87 counter2 <= 7 ;
88 MISO <= 0;
93 bus[counter1] <= MOSI;
94 counter1 <= counter1 - 1;   //decrement the counter to fill the whole output rx_data
97 rx_valid = 1;
98 rx_data <= bus ; //sending the parallel data to the RAM
104 bus[counter1] <= MOSI;
105 counter1 <= counter1 - 1;   //decrement the counter to fill the whole output rx_data
108 rx_valid <= 1;
109 rx_data <= bus ; //sending the parallel data to the RAM
110 ADD_DATA_checker <= 0; //(means that the read address is recieved) as when this state ends we will go to the READ_DATA state
116 bus[counter1] <= MOSI;
117 counter1 <= counter1 - 1;   //decrement the counter to fill the whole output rx_data
120 rx_valid <= 1;
121 rx_data <= bus ; //sending the parallel data to the RAM
122 counter1 <= 9 ; //only and only in this case we will reset the counter as we won't go back to the IDLE state until the process ends
124 if(rx_valid  == 1) rx_valid <= 0; //only and only in this case we will reset the rx_valid as we won't go back to the IDLE state until the process ends
126 MISO <= tx_data[counter2] ; //counter-2 as it it's an 8 bit bus not 10 bit bus
127 counter2 <= counter2 - 1 ;
130 ADD_DATA_checker <= 1; //(means that we should send another address for reading in the next time) so we will go to READ_ADD state
```

Statements - by instance (/spi_wrapper_top/dut/Ram)

RAM.v

```
20 always @(posedge clk) begin
22 dout <= 8'b0;
23 tx_valid <= 1'b0;
24 addr_wr <= 8'b0;
25 addr_re <= 8'b0;
30 addr_wr <= din[7:0]; // Write address
31 tx_valid <= 1'b0;
34 memory[addr_wr] <= din[7:0]; // Write data in the address specified earlier
35 tx_valid <= 1'b0;
38 addr_re <= din[7:0]; // Read address
39 tx_valid <= 1'b0;
42 dout <= memory[addr_re];
43 tx_valid <= 1'b1;
```

- **Branch Coverage**

Branches - by instance (/spi_wrapper_top/dut/SPI)

SPI_SLAVE.v

```
22 if(~rst_n)
24 else
31 case(cs)
32 IDLE : begin
33 if(ss_n)
35 else
38 CHK_CMD : begin
E  39 if(ss_n)
41 else begin
Em 42 if((~ss_n) && (MOSI == 0))
44 else if ((~ss_n) && (MOSI == 1) && (ADD_DATA_checker == 1))
46 else if ((~ss_n) && (MOSI == 1) && (ADD_DATA_checker == 0))
50 WRITE : begin
51 if(ss_n || counter1 == 4'b1111) //counter = -1(4'b1111 = -1) means that the whole rx_bus is completed so go to state IDLE
53 else
56 READ_ADD : begin
57 if(ss_n || counter1 == 4'b1111)
59 else
62 READ_DATA : begin
63 if(ss_n)
65 else
73 if (~rst_n) begin
83 else begin
84 if(cs == IDLE) begin
91 else if(cs == WRITE) begin
Em 92 if (counter1 >= 0)begin
96 if(counter1 == 4'b1111) begin//(4'b1111) means that the counter has the value -1 (the rx_data is completed)
102 else if (cs == READ_ADD) begin
Em 103 if (counter1 >= 0)begin
107 if(counter1 == 4'b1111) begin//(4'b1111) means that the counter has the value -1 (the rx_data is completed)
114 else if (cs == READ_DATA) begin
Em 115 if (counter1 >= 0)begin
119 if(counter1 == 4'b1111) begin//(4'b1111) means that the counter has the value -1 (the rx_data is completed)
124 if(rx_valid  == 1) rx_valid <= 0; //only and only in this case we will reset the rx_valid as we won't go back to the IDLE state until the process ends
125 if(tx_valid==1 && counter2 >=0)begin
129 if(counter2 == 3'b111)begin
```

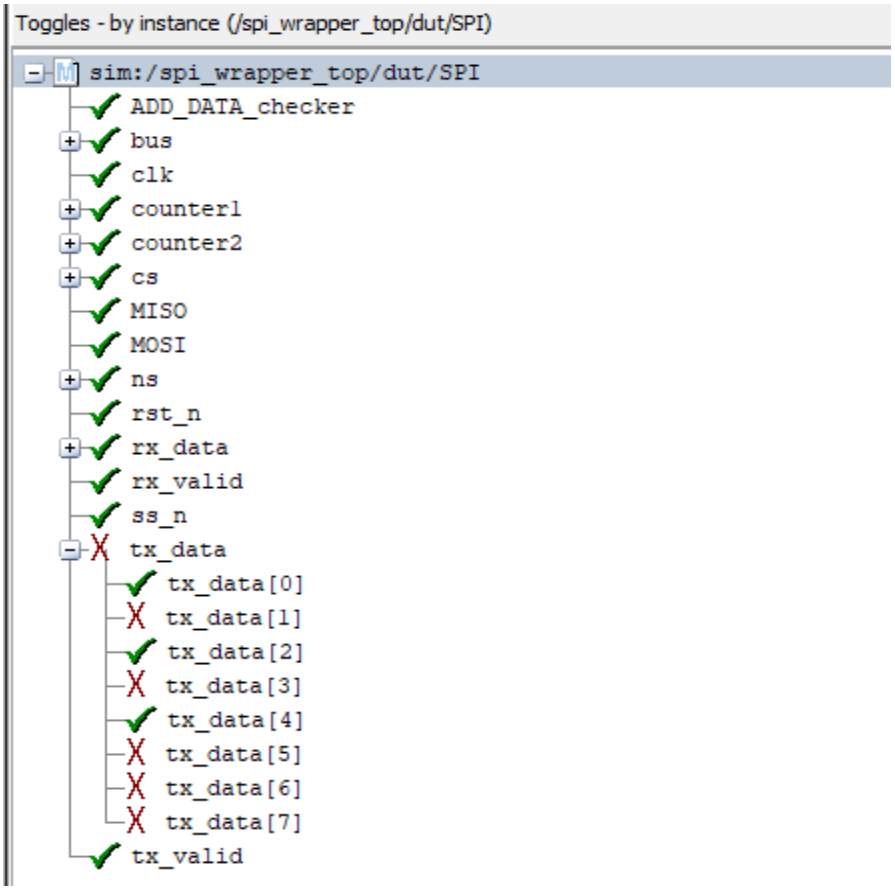
Branches - by instance (/spi_wrapper_top/dut/Ram)

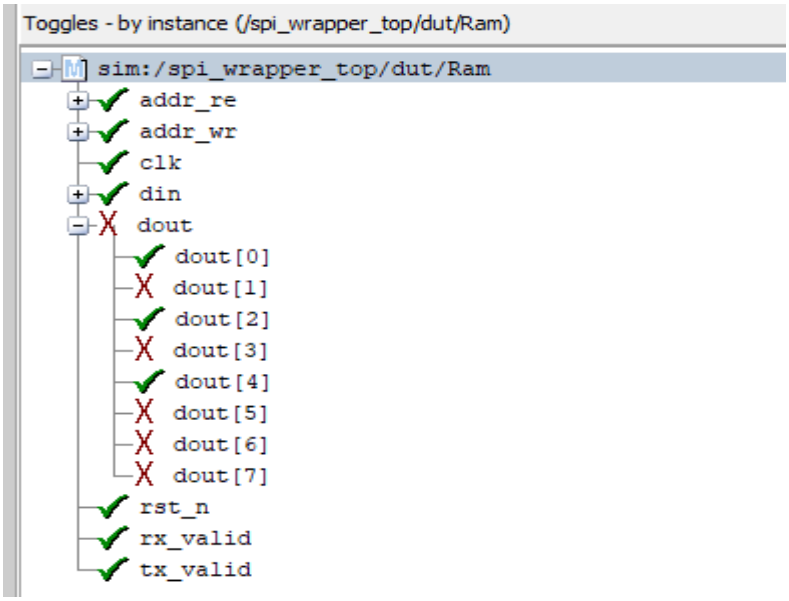
RAM.v

```
21 if (!rst_n) begin   //making the reset synchronous so the memory is synthesized as a block
26 end else begin
27 if (rx_valid) begin // Once rx_valid is high, then the din bus is completed
F  28 case (din[9:8])
B     28.1 case
29 2'b00: begin
33 2'b01: begin
37 2'b10: begin
41 2'b11: begin // Read data in the address specified earlier and then send the data in dout and make tx_valid high
```

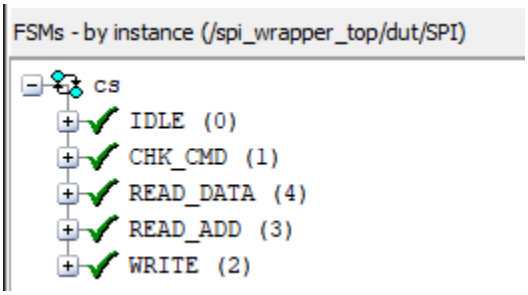
We got all cases **except for unknown din[9:8]** and as we also did not include default in the case statement **but I did this to make a RAM block when synthesizing the design**, check the design Repository for deeper understanding

- ## Toggle Coverage



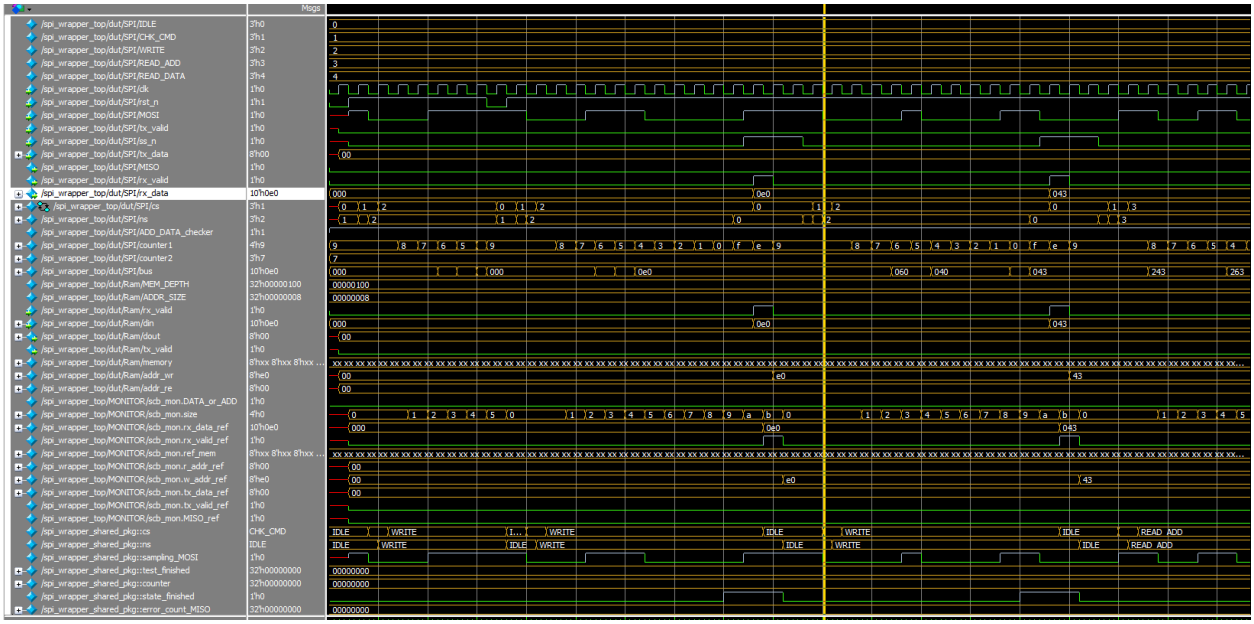


Tx_data ,(tx_data & dout are the same signal), **is randomly generated** so if we increased the tests it is expected to **get 100% toggle coverage**

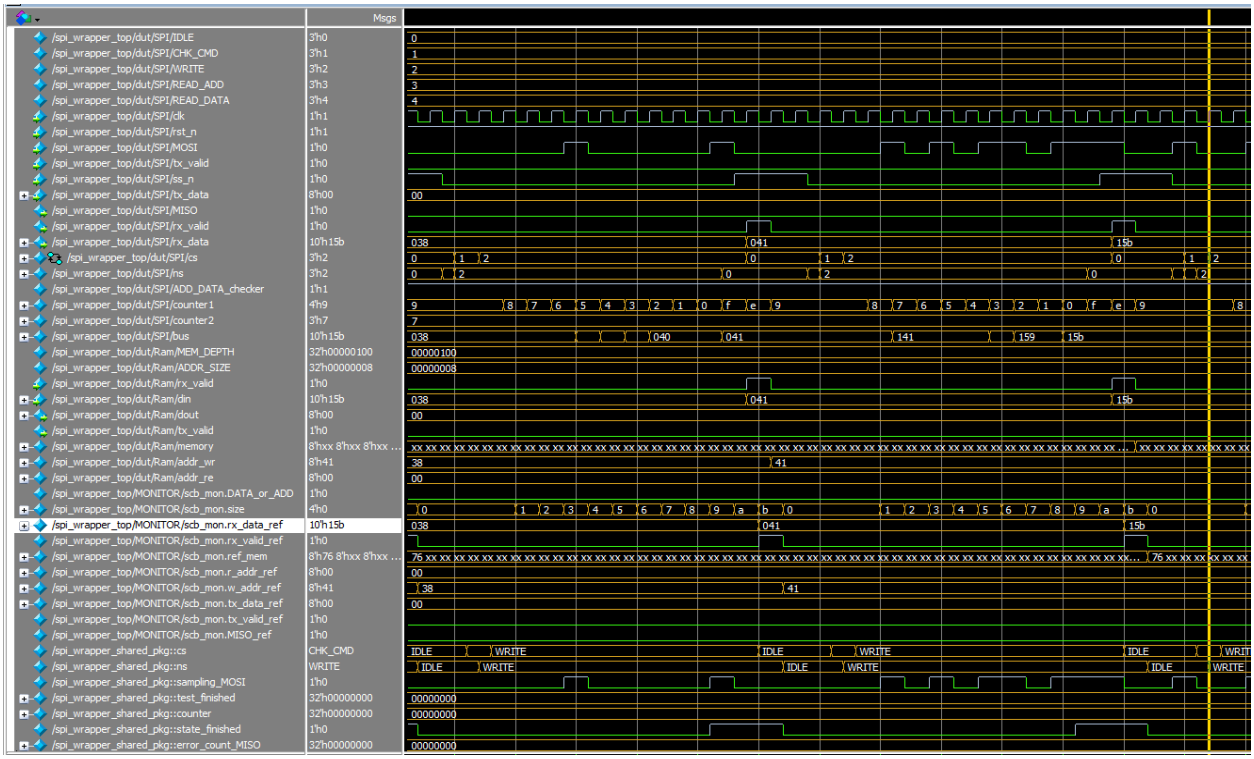


Verifying functionality

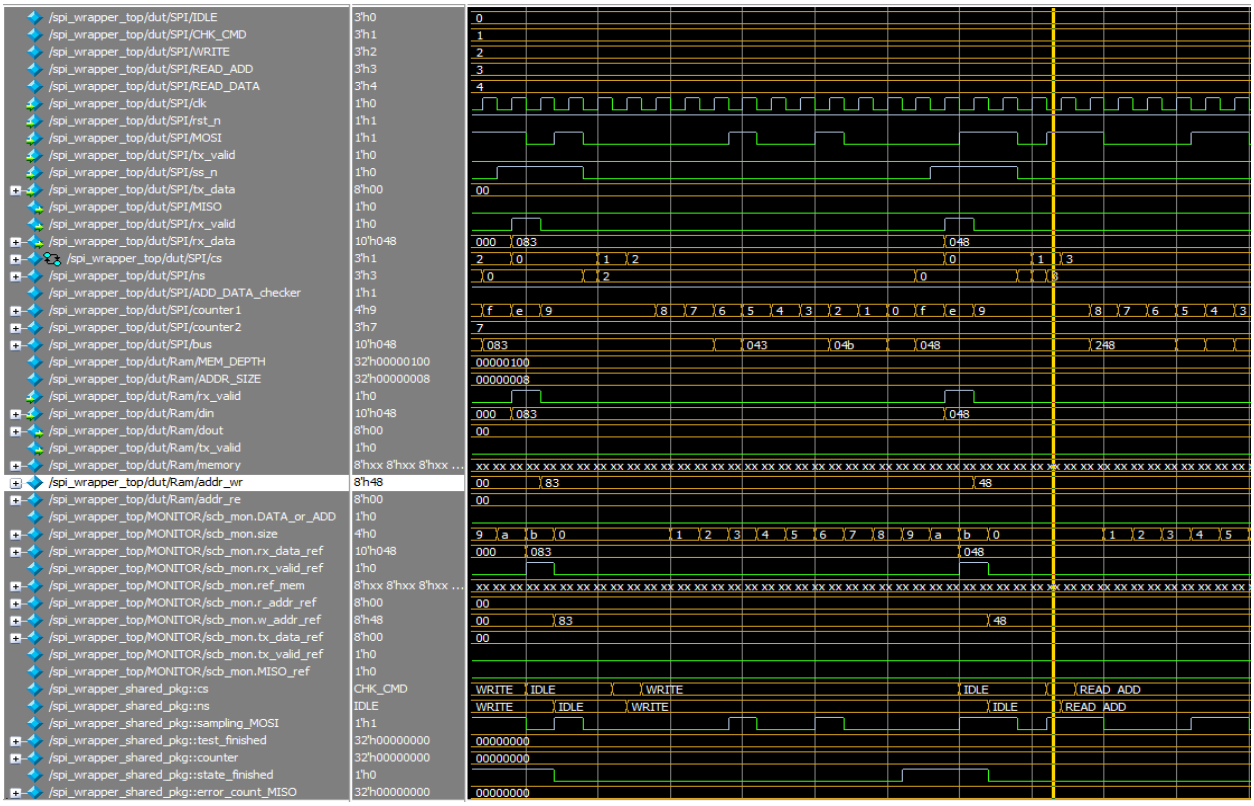
- **Write state:**



Rx_data is the same as rx_data_ref and also rx_valid is the same as rx_valid_ref, and the states are the same, notice that the Most Significant two Bits of rx_data[9:8] = 2'b00 as the state is write **address** (from Specs), so **addr_wr** has the value **0xe0**

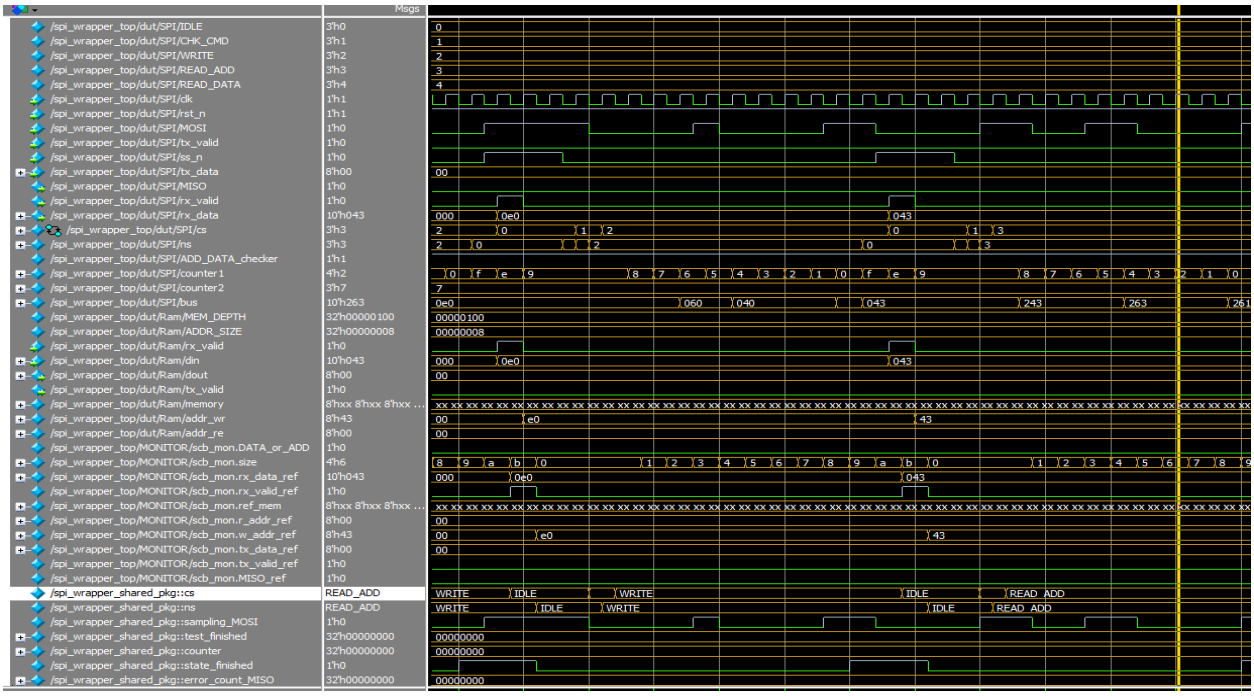


Rx_data **is the same** as rx_data_ref and also rx_valid is the same as rx_valid_ref, and the states are the same, notice that the Most Significant two Bits of rx_data[9:8] = 2'b01 as the state is write **Data** (from Specs), so we will notice that some value **(0x5b)** has been written in memory **(in address 0x41)**

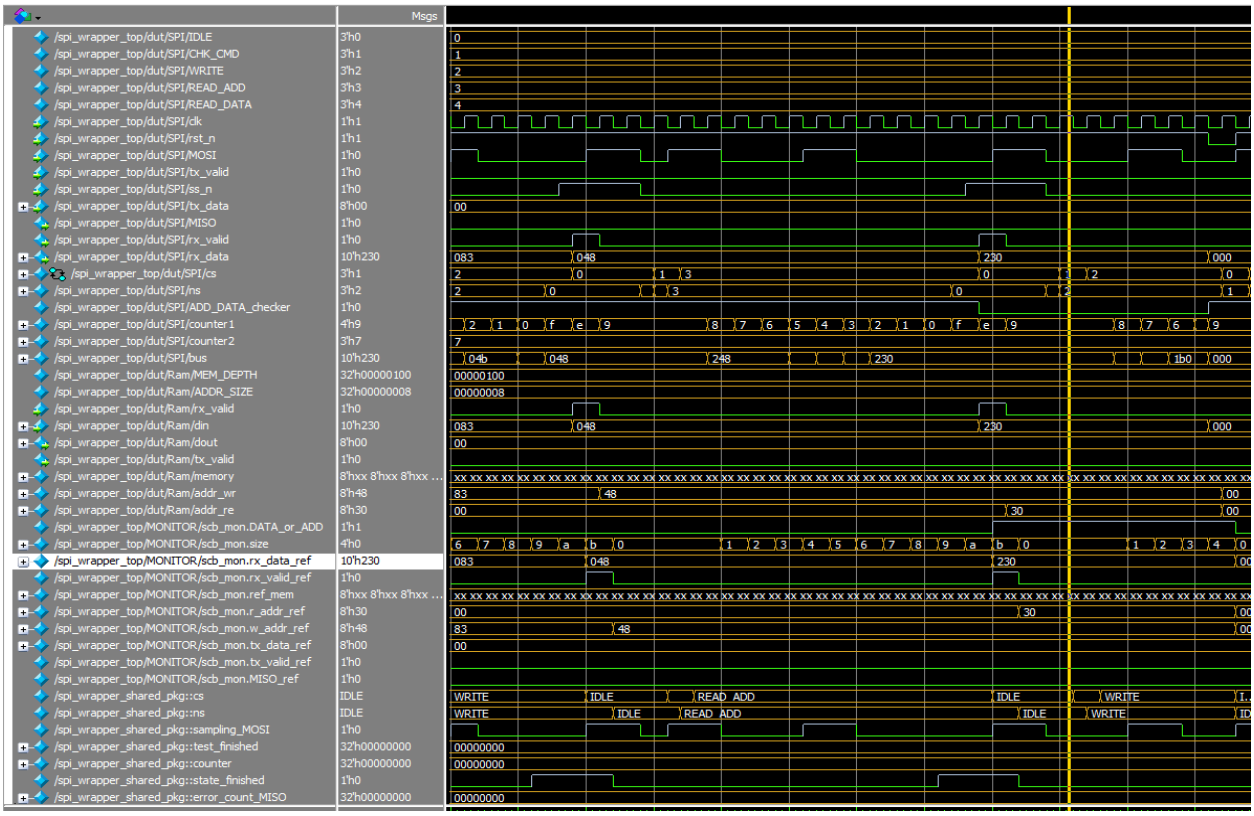


Another proof

- **READ_ADD state:**

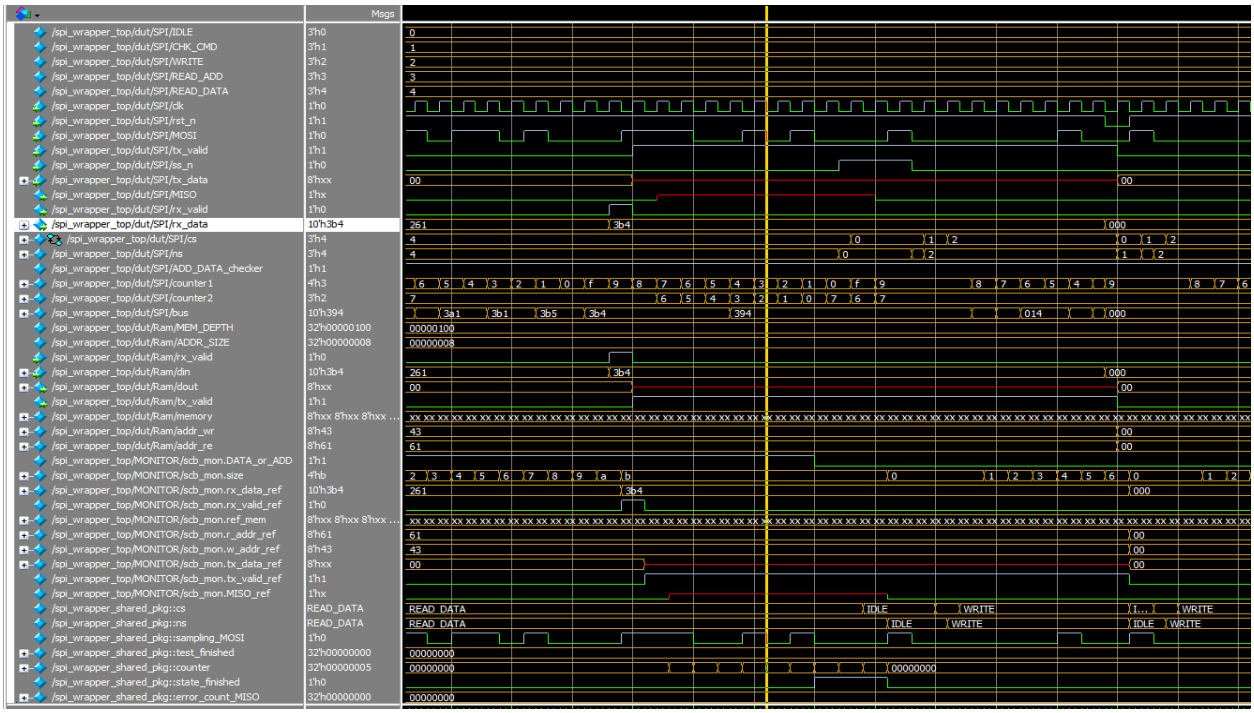


You will notice that rx_data and rx_data_ref **have the same data (0x261)**, Most Significant two Bits of rx_data[9:8] = 2'b10 as the state is read **address** (from Specs)

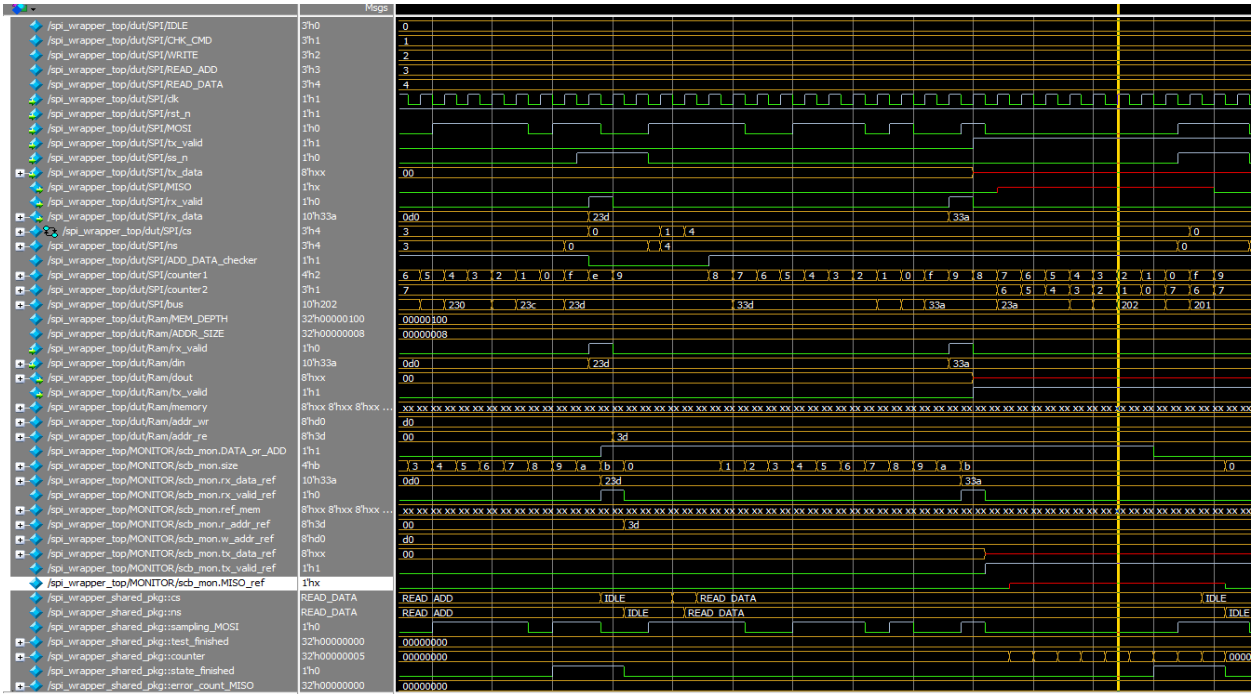


Another proof, you will find that **addr_re** as well as **r_addr_ref** having the same value (0x30)

- **READ_DATA state (Randomized test):**



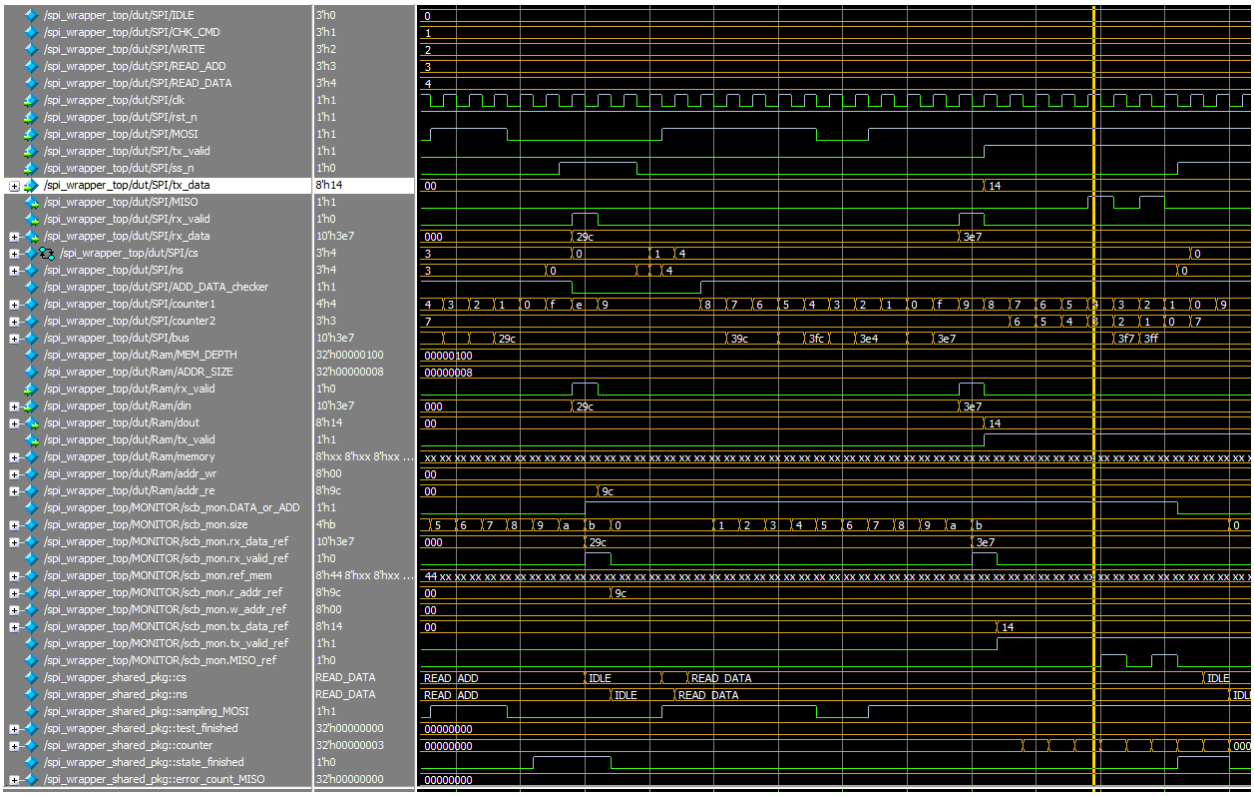
In read data state, rx_data is sent **correctly** but we care about the MISO signal in this state, Most Significant two Bits of rx_data[9:8] = 2'b11 as the state is read **Data** (from Specs), since the write address is **not the same** as read address as it is **randomized** test so the data read from the memory will often be **unknown**



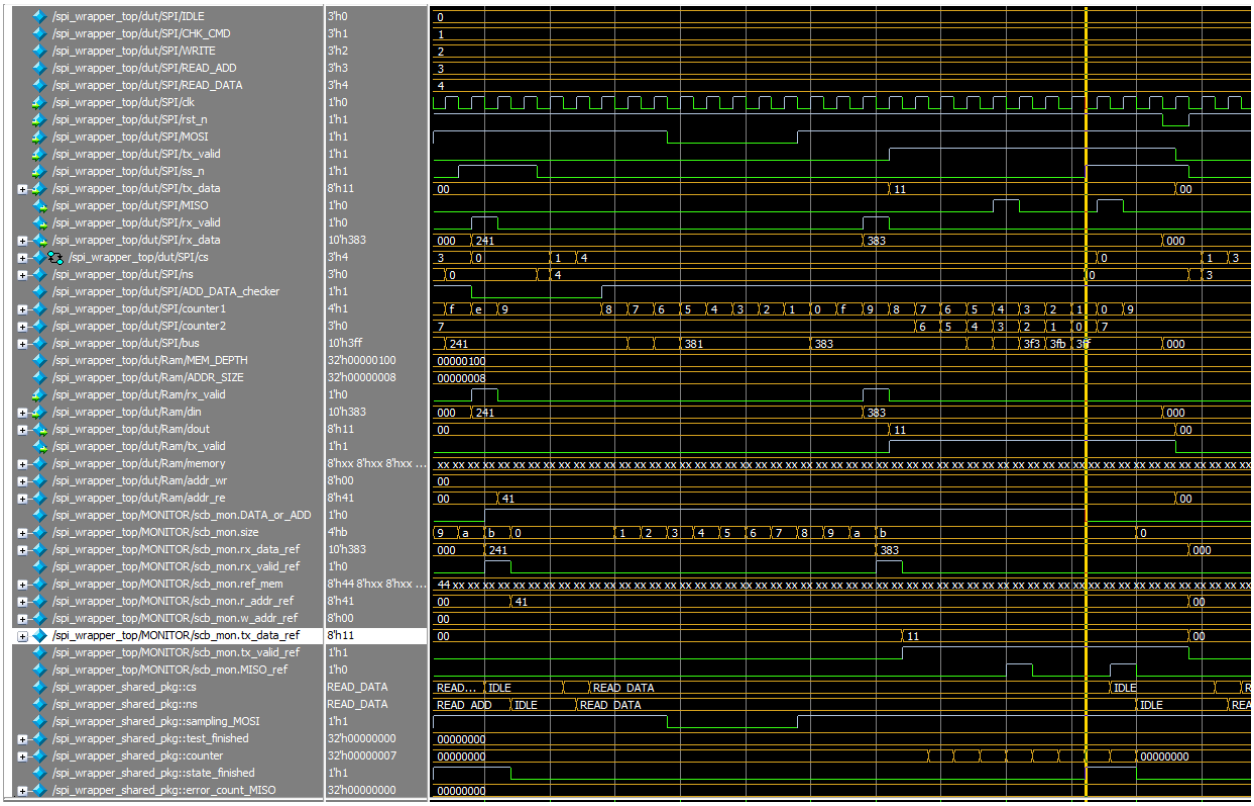
After rx_data is sent, we will notice the same behavior for **MISO signal** and **MISO_ref** signal which **indicates the correctness of the design**

- **READ_DATA state (Directed test):**

Since the read address is **not the same** as write **address so we usually expect an unknown behavior In the MISO signal (output signal)**, so I've created a nearly **directed tests** to read a real value from the memory **(make read address the same as any write address has a value in the memory)**



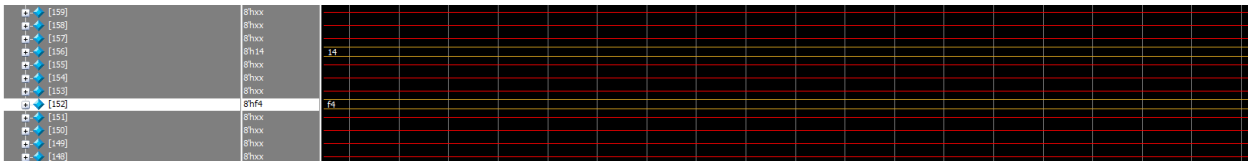
Tx_data signal now has a **real value** (0x14) which will be out on **MISO signal**



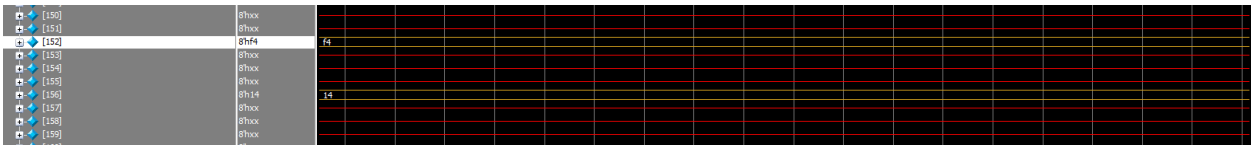
Tx_data_ref signal now has a real value (0x14) which will be out on MISO signal

- **Memories Comparison:**

We will compare the two memories values at specific addresses to see that **they are matched**

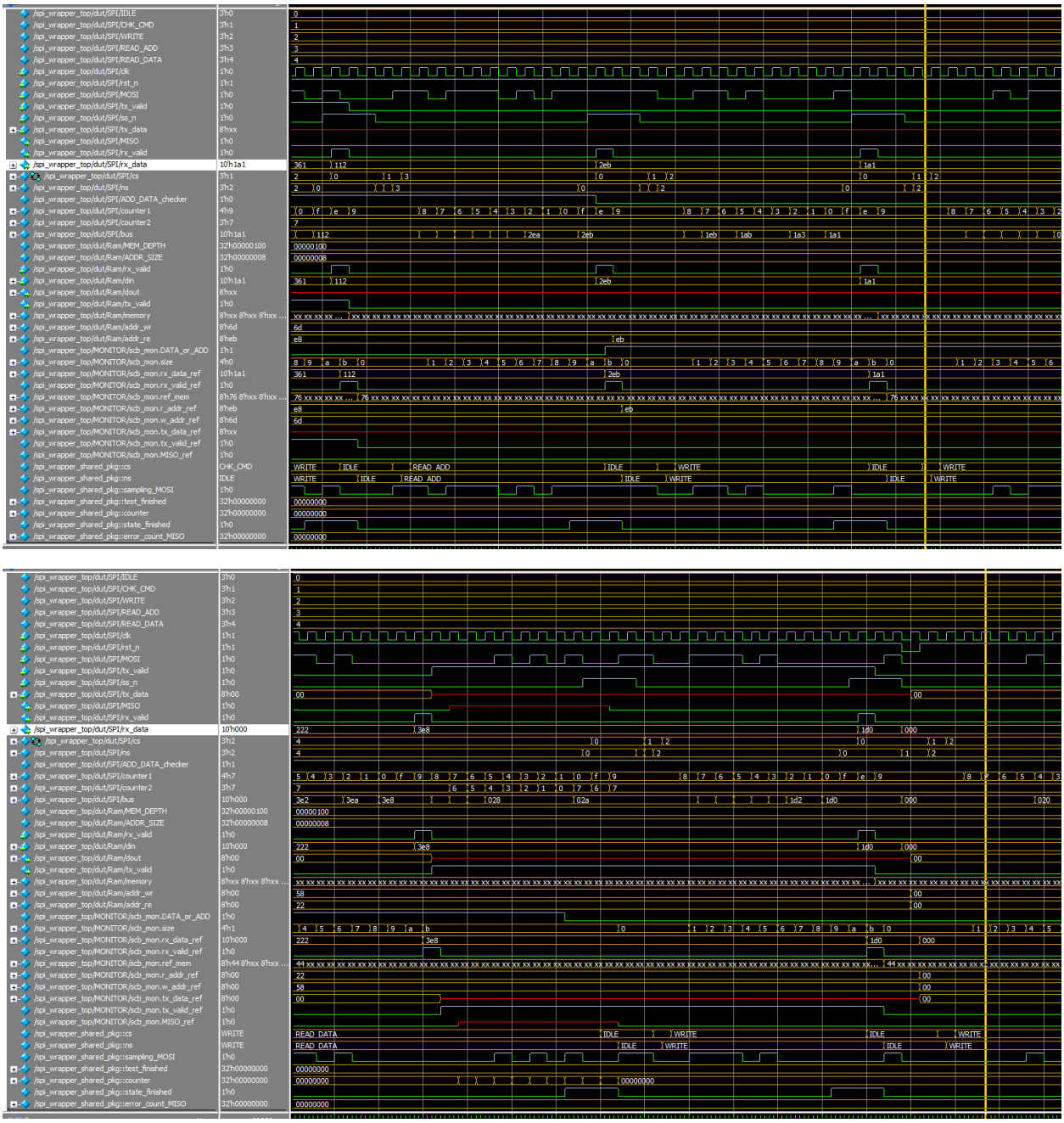


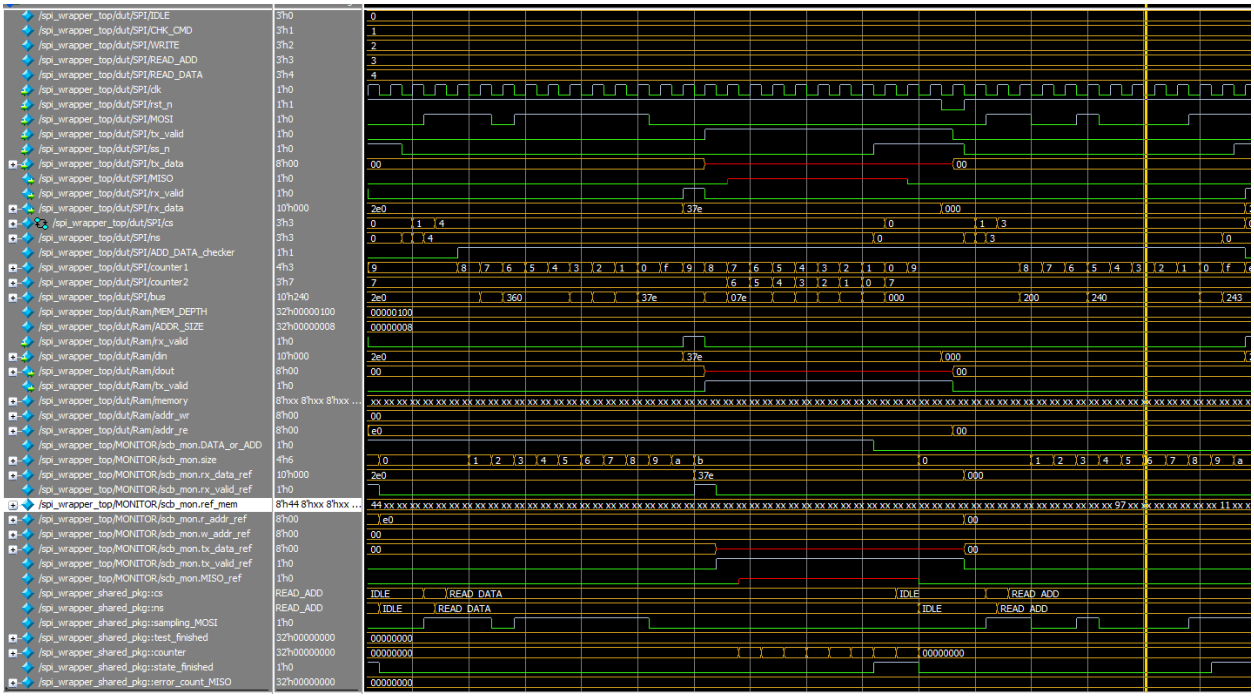
DUT memory



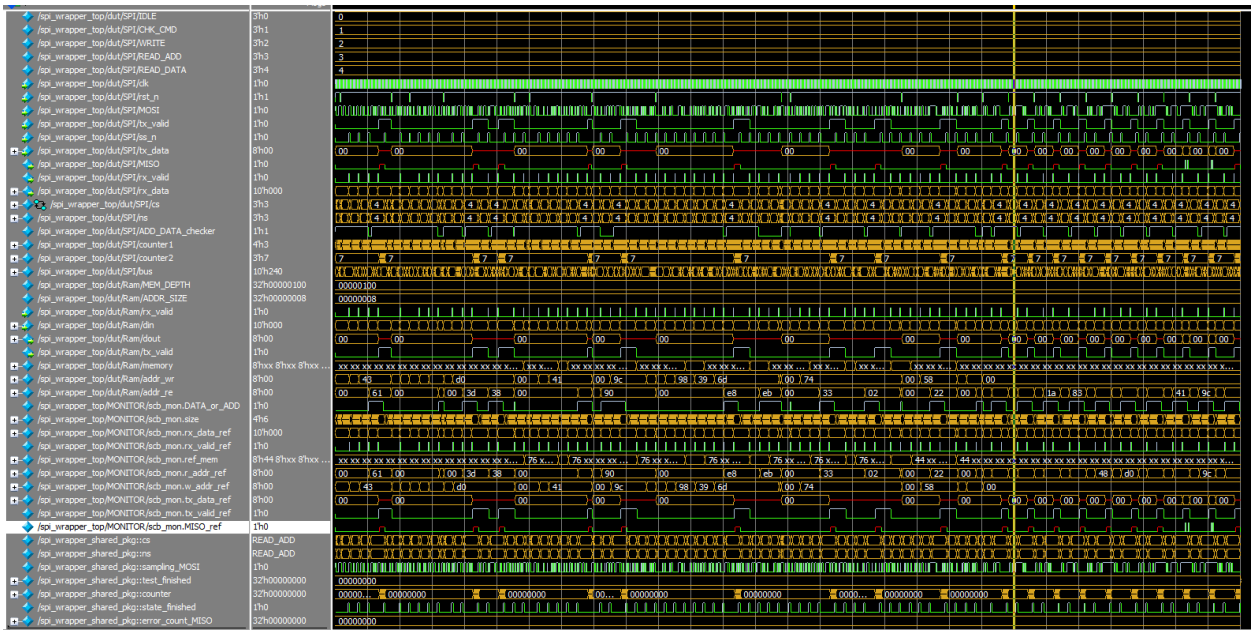
Reference memory

- **Random snapshots :**





- **Full waveform:**

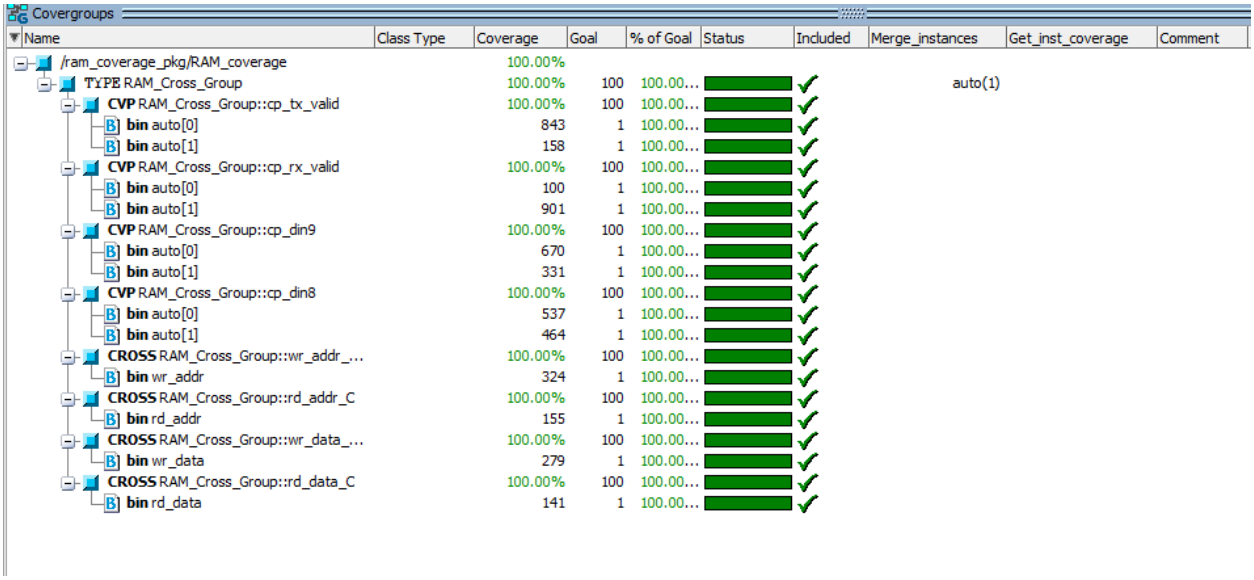


Full waveform proves MISO error counter were never executed, so we **have no output mismatch** between the original design and the **reference model** in the **scoreboard file**

```
VSIM 43> run -all
# error count = 0, correct count = 1401
# ** Note: $stop    : SPI_WRAPPER_MONITOR.sv(38)
#    Time: 28020 ns  Iteration: 0  Instance: /spi_wrapper_top/MONITOR
# Break in Module spi_wrapper_monitor at SPI_WRAPPER_MONITOR.sv line 38
```

Ram:

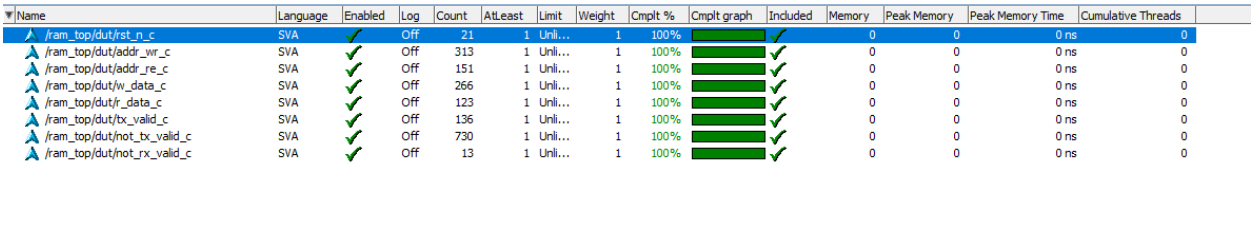
- **Functional Coverage**



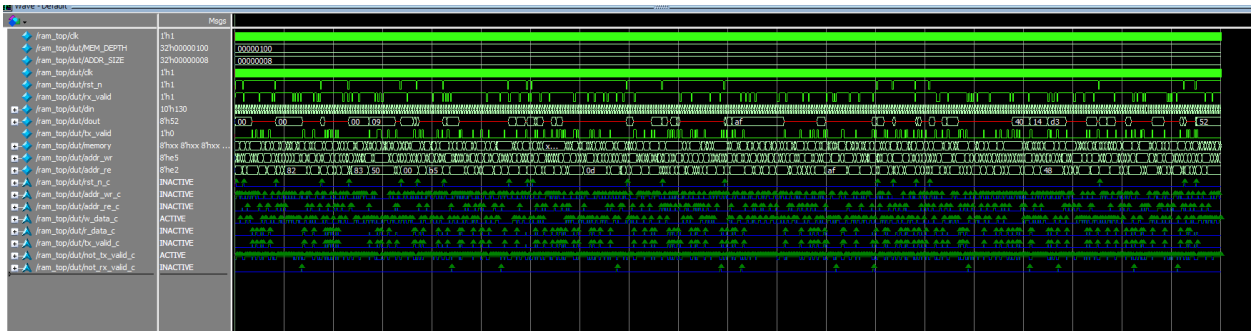
Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/ram_coverage_pkg/RAM_coverage		100.00%							
TYPE RAM_Cross_Group		100.00%	100	100.00...				auto(1)	
CVP RAM_Cross_Group::cp_tx_valid		100.00%	100	100.00...					
bin auto[0]		843	1	100.00...					
bin auto[1]		158	1	100.00...					
CVP RAM_Cross_Group::cp_rx_valid		100.00%	100	100.00...					
bin auto[0]		100	1	100.00...					
bin auto[1]		901	1	100.00...					
CVP RAM_Cross_Group::cp_din9		100.00%	100	100.00...					
bin auto[0]		670	1	100.00...					
bin auto[1]		331	1	100.00...					
CVP RAM_Cross_Group::cp_din8		100.00%	100	100.00...					
bin auto[0]		537	1	100.00...					
bin auto[1]		464	1	100.00...					
CROSS RAM_Cross_Group::wr_addr_...		100.00%	100	100.00...					
bin wr_addr		324	1	100.00...					
CROSS RAM_Cross_Group::rd_addr_C		100.00%	100	100.00...					
bin rd_addr		155	1	100.00...					
CROSS RAM_Cross_Group::wr_data_...		100.00%	100	100.00...					
bin wr_data		279	1	100.00...					
CROSS RAM_Cross_Group::rd_data_C		100.00%	100	100.00...					
bin rd_data		141	1	100.00...					

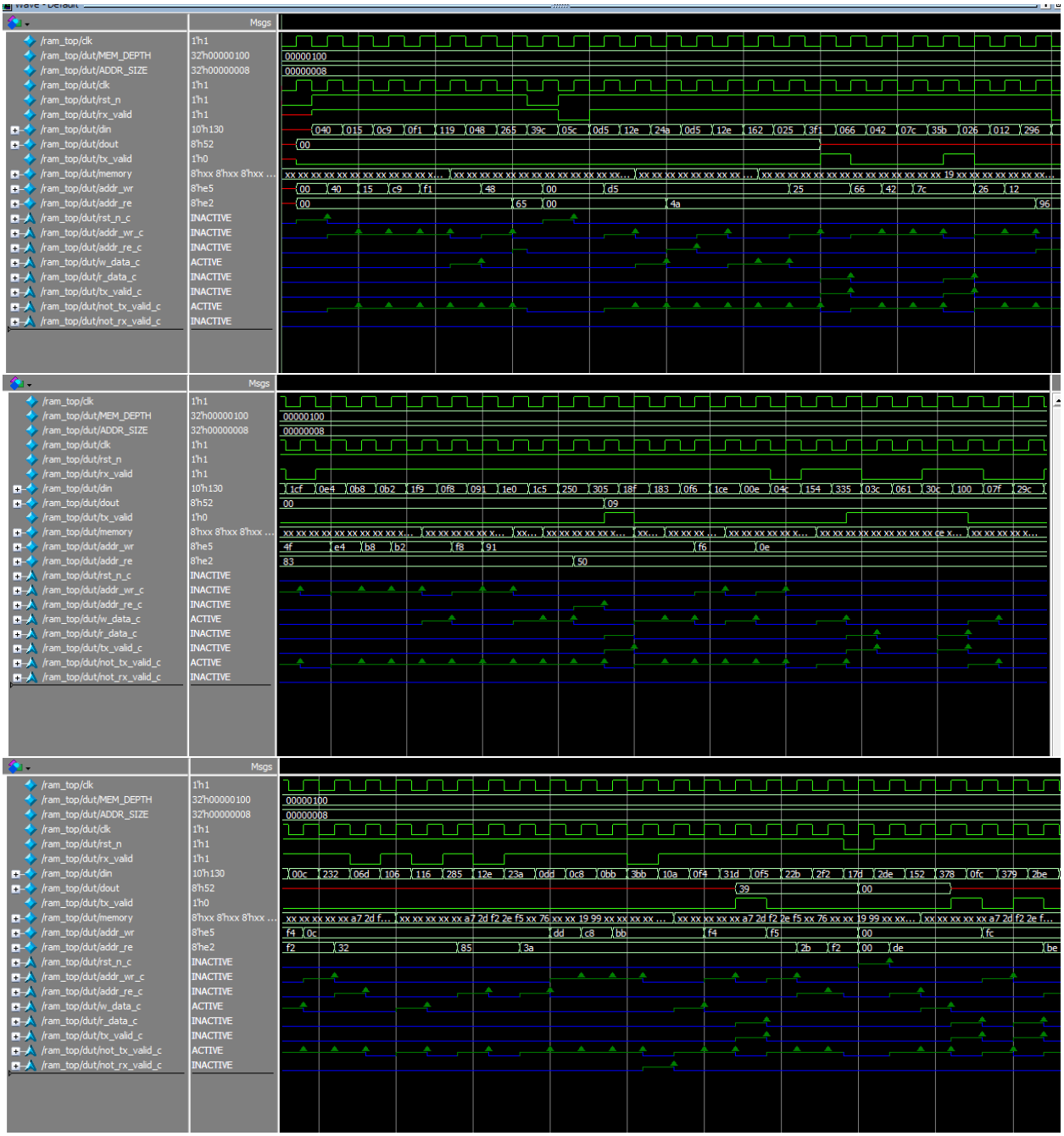
- **Assertions:**



Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/ram_top/dut/rst_n_c	SVA	✓	Off	21	1	Unli...	1	100%		✓	0	0	0 ns	0
/ram_top/dut/addr_wr_c	SVA	✓	Off	313	1	Unli...	1	100%		✓	0	0	0 ns	0
/ram_top/dut/addr_re_c	SVA	✓	Off	151	1	Unli...	1	100%		✓	0	0	0 ns	0
/ram_top/dut/w_data_c	SVA	✓	Off	266	1	Unli...	1	100%		✓	0	0	0 ns	0
/ram_top/dut/r_data_c	SVA	✓	Off	123	1	Unli...	1	100%		✓	0	0	0 ns	0
/ram_top/dut/tx_valid_c	SVA	✓	Off	136	1	Unli...	1	100%		✓	0	0	0 ns	0
/ram_top/dut/not_tx_valid_c	SVA	✓	Off	730	1	Unli...	1	100%		✓	0	0	0 ns	0
/ram_top/dut/not_rx_valid_c	SVA	✓	Off	13	1	Unli...	1	100%		✓	0	0	0 ns	0



- **Verifying Functionality:**

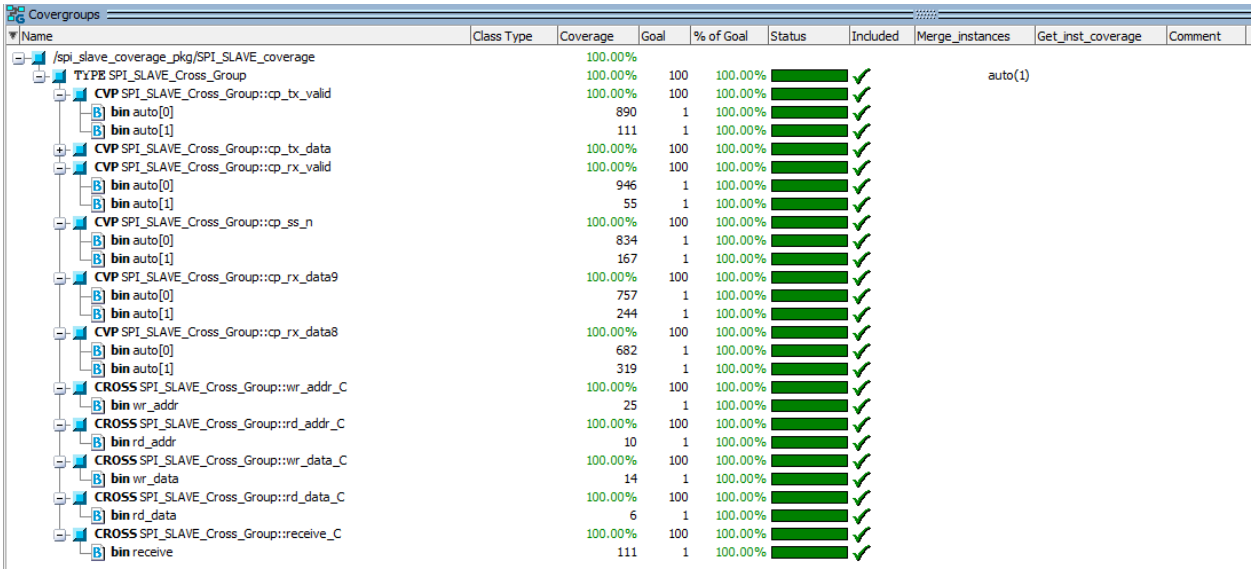


- **Transcript**

```
# Loading sv_std.std
# Loading work.ram_top(fast)
# Loading work.ram_if(fast__1)
# Loading work.RAM(fast)
# Loading work.shared_pkg(fast)
# Loading work.ram_transaction_pkg(fast)
# Loading work.RAM_TB_sv_unit(fast)
# Loading work.ram_tb(fast)
# Loading work.ram_scoreboard_pkg(fast)
# Loading work.ram_coverage_pkg(fast)
# Loading work.RAM_MONITOR_sv_unit(fast)
# Loading work.ram_monitor(fast)
VSIM 12> run -all
# error count = 0, correct count = 1001
# ** Note: $stop    : RAM_MONITOR.sv(28)
#    Time: 20020 ns  Iteration: 0  Instance: /ram_top/MONITOR
# Break in Module ram_monitor at RAM_MONITOR.sv line 28
```

SPI slave:

- **Functional Coverage**



Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/spi_slave_coverage_pkg/SPI_SLAVE_coverage		100.00%							
TYPE SPI_SLAVE_Cross_Group		100.00%	100	100.00%	✓		auto(1)		
CVP SPI_SLAVE_Cross_Group::cp_tx_valid		100.00%	100	100.00%	✓				
bin auto[0]		890	1	100.00%	✓				
bin auto[1]		111	1	100.00%	✓				
CVP SPI_SLAVE_Cross_Group::cp_tx_data		100.00%	100	100.00%	✓				
CVP SPI_SLAVE_Cross_Group::cp_rx_valid		100.00%	100	100.00%	✓				
bin auto[0]		946	1	100.00%	✓				
bin auto[1]		55	1	100.00%	✓				
CVP SPI_SLAVE_Cross_Group::cp_ss_n		100.00%	100	100.00%	✓				
bin auto[0]		834	1	100.00%	✓				
bin auto[1]		167	1	100.00%	✓				
CVP SPI_SLAVE_Cross_Group::cp_rx_data9		100.00%	100	100.00%	✓				
bin auto[0]		757	1	100.00%	✓				
bin auto[1]		244	1	100.00%	✓				
CVP SPI_SLAVE_Cross_Group::cp_rx_data8		100.00%	100	100.00%	✓				
bin auto[0]		682	1	100.00%	✓				
bin auto[1]		319	1	100.00%	✓				
CROSS SPI_SLAVE_Cross_Group::wr_addr_C		100.00%	100	100.00%	✓				
bin wr_addr		25	1	100.00%	✓				
CROSS SPI_SLAVE_Cross_Group::rd_addr_C		100.00%	100	100.00%	✓				
bin rd_addr		10	1	100.00%	✓				
CROSS SPI_SLAVE_Cross_Group::wr_data_C		100.00%	100	100.00%	✓				
bin wr_data		14	1	100.00%	✓				
CROSS SPI_SLAVE_Cross_Group::rd_data_C		100.00%	100	100.00%	✓				
bin rd_data		6	1	100.00%	✓				
CROSS SPI_SLAVE_Cross_Group::receive_C		100.00%	100	100.00%	✓				
bin receive		111	1	100.00%	✓				

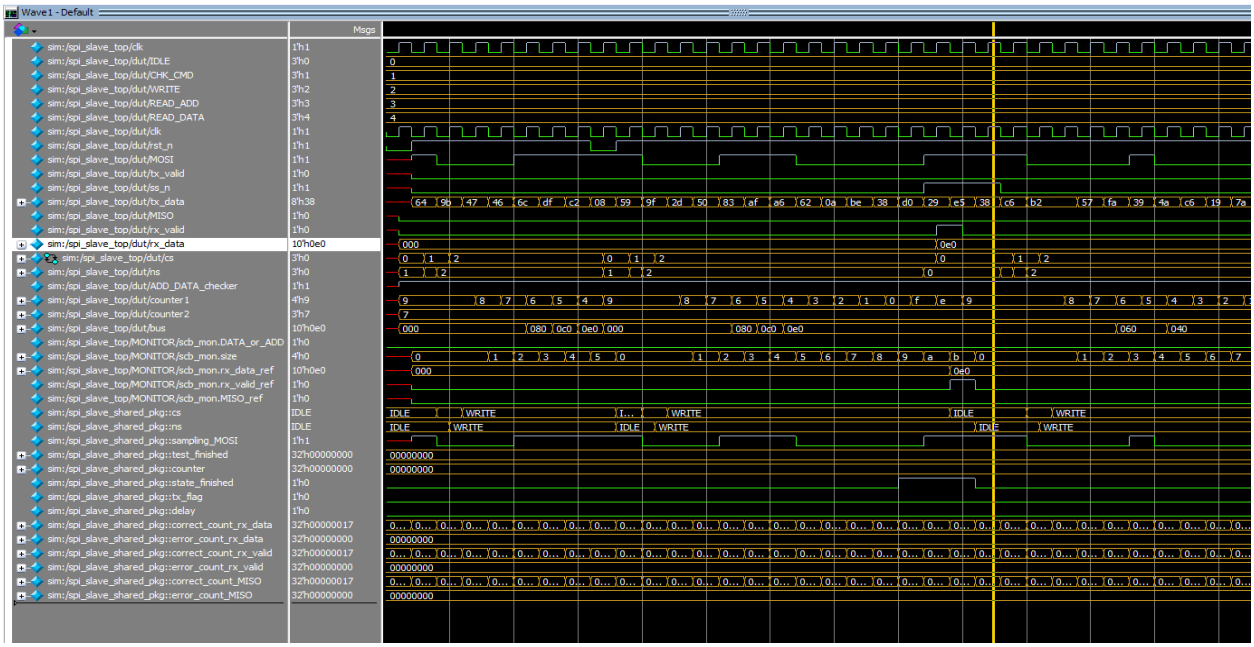
- **Assertions:**

Cover Directives

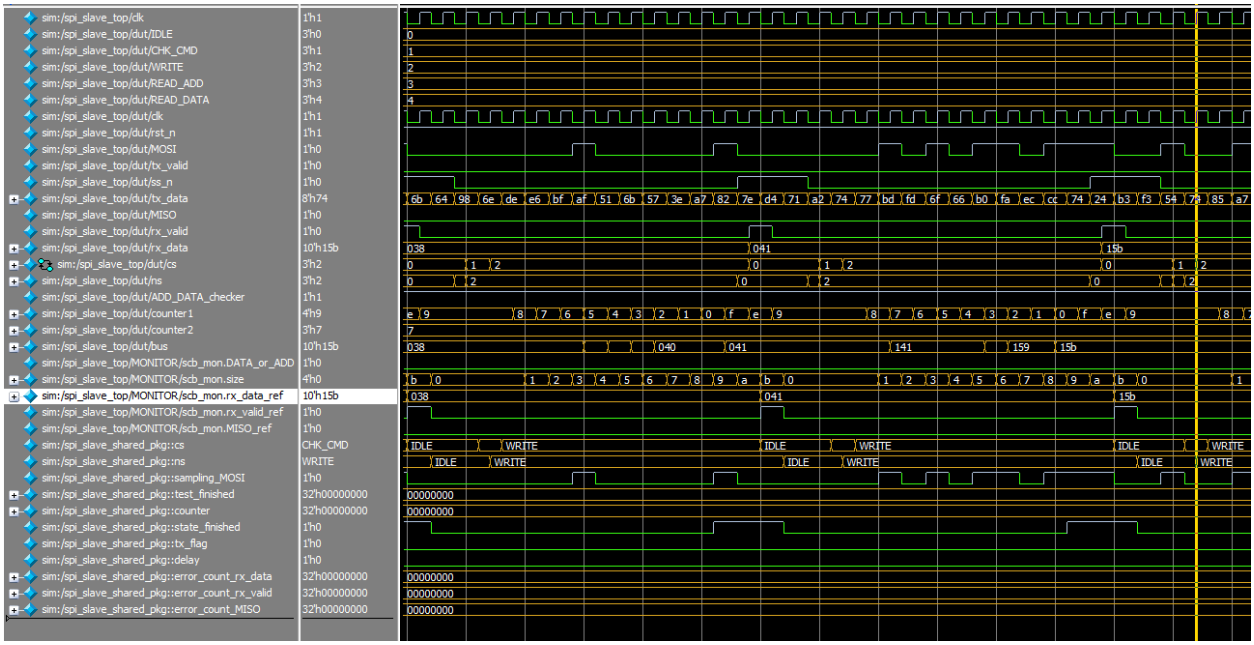
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/spi_slave_top/dut/rst_n_ns_c	SVA	✓	Off	13	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/idle_to_chk_cmd_c	SVA	✓	Off	64	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/chk_cmd_to_write_c	SVA	✓	Off	36	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/chk_cmd_to_read_add_c	SVA	✓	Off	17	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/chk_cmd_to_read_data_c	SVA	✓	Off	11	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/write_hold_c	SVA	✓	Off	326	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/write_to_idle_c	SVA	✓	Off	30	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/read_add_hold_c	SVA	✓	Off	169	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/read_add_to_idle_c	SVA	✓	Off	14	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/read_data_hold_c	SVA	✓	Off	192	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/read_data_to_idle_c	SVA	✓	Off	9	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/rst_n_o_c	SVA	✓	Off	13	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/idle_o_c	SVA	✓	Off	171	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/write_o_c	SVA	✓	Off	30	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/read_add_o_c	SVA	✓	Off	14	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/read_data_o_c_1	SVA	✓	Off	9	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/read_data_o_c_2	SVA	✓	Off	38	1	Unli...	1	100%		✓	0	0	0 ns	0
/spi_slave_top/dut/read_data_o_c_3	SVA	✓	Off	11	1	Unli...	1	100%		✓	0	0	0 ns	0

Verifying functionality

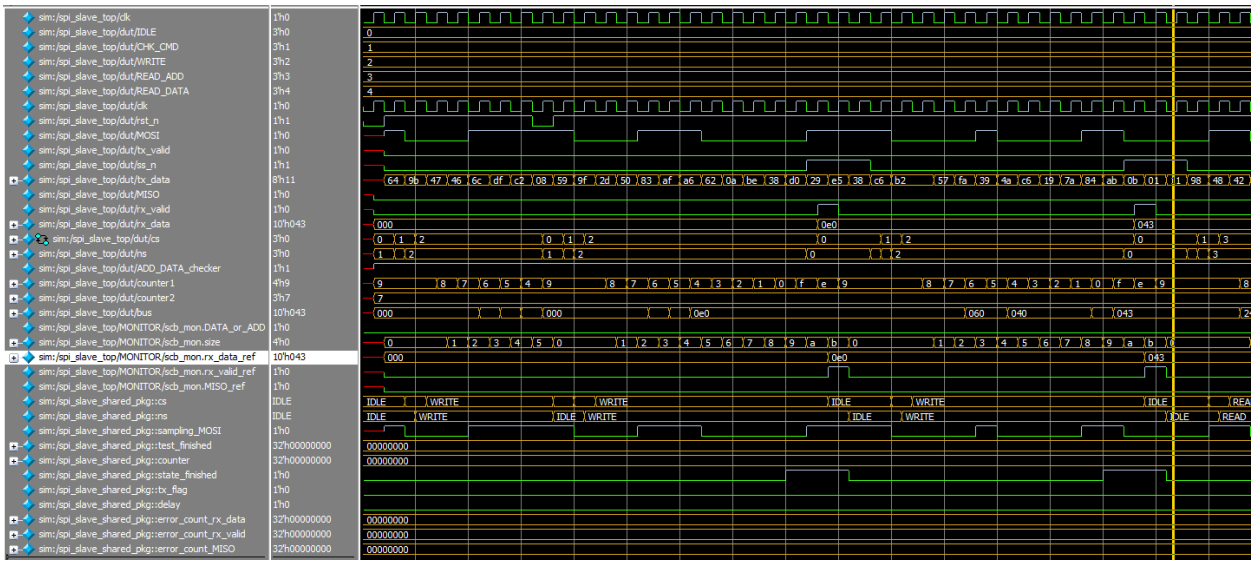
- **Write state:**



Rx_data is the same as rx_data_ref and **also rx_valid is the same as rx_valid_ref**, and the states are the same, notice that the Most Significant two Bits of rx_data[9:8] = 2'b00 as the state is write **address** (from Specs)

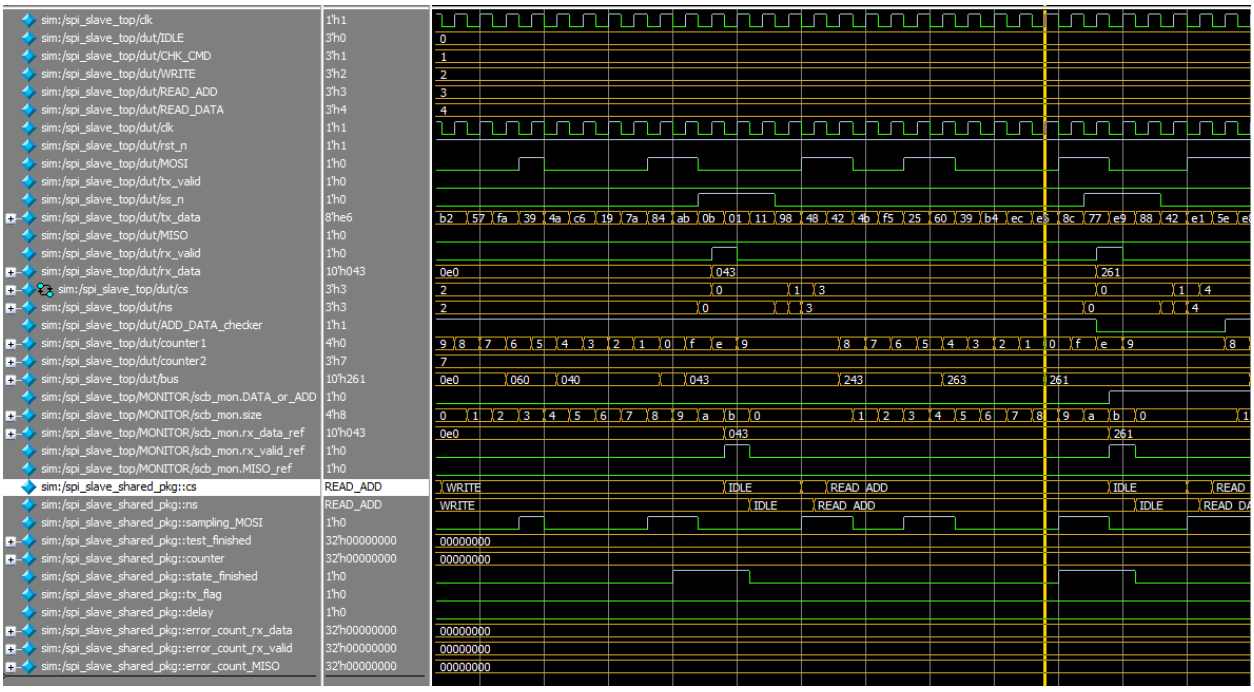


Rx_data is the same as rx_data_ref and also rx_valid is the same as rx_valid_ref, and the states are the same, notice that the Most Significant two Bits of rx_data[9:8] = 2'b01 as the state is **write Data** (from Specs)

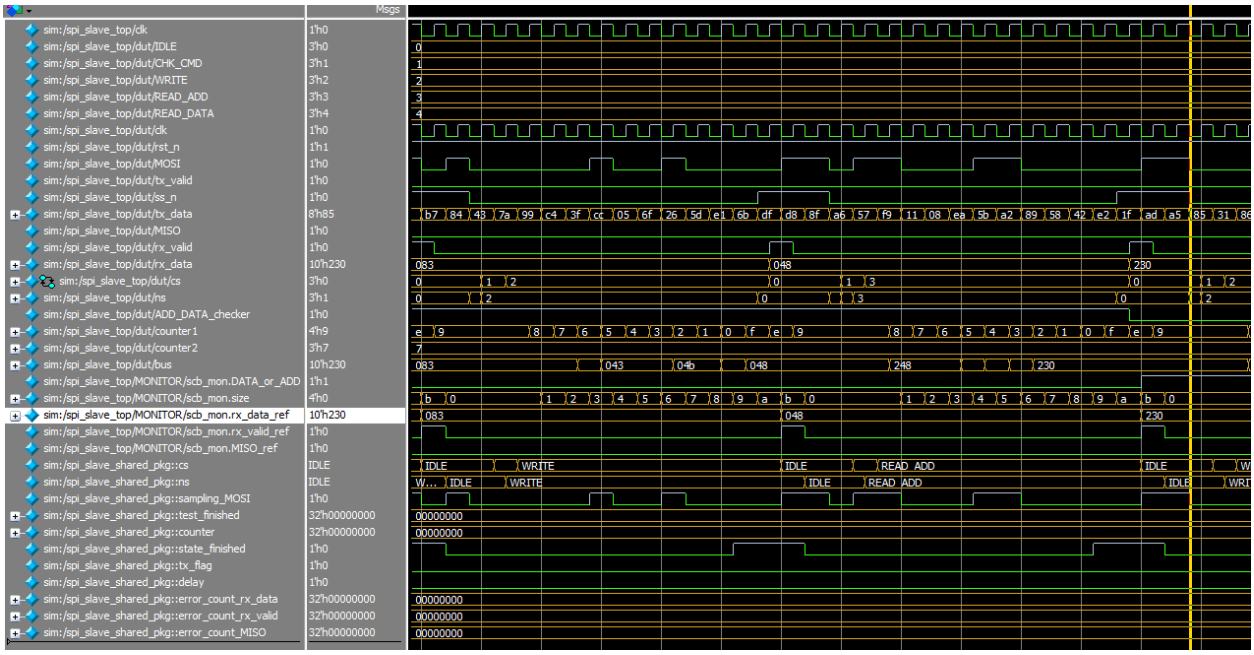


Another proof

- **READ_ADD state:**

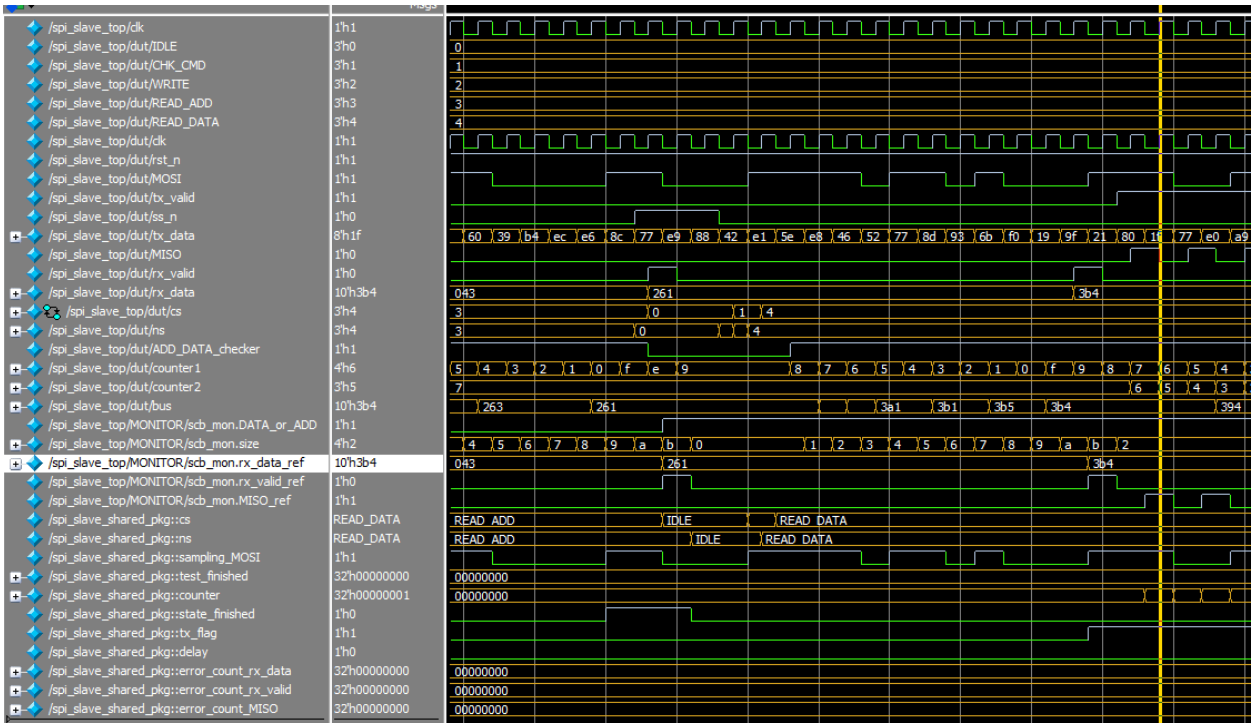


You will notice that rx_data and **rx_data_ref have the same data (0x261)**, Most Significant two Bits of rx_data[9:8] = 2'b10 as the state is **read address** (from Specs)

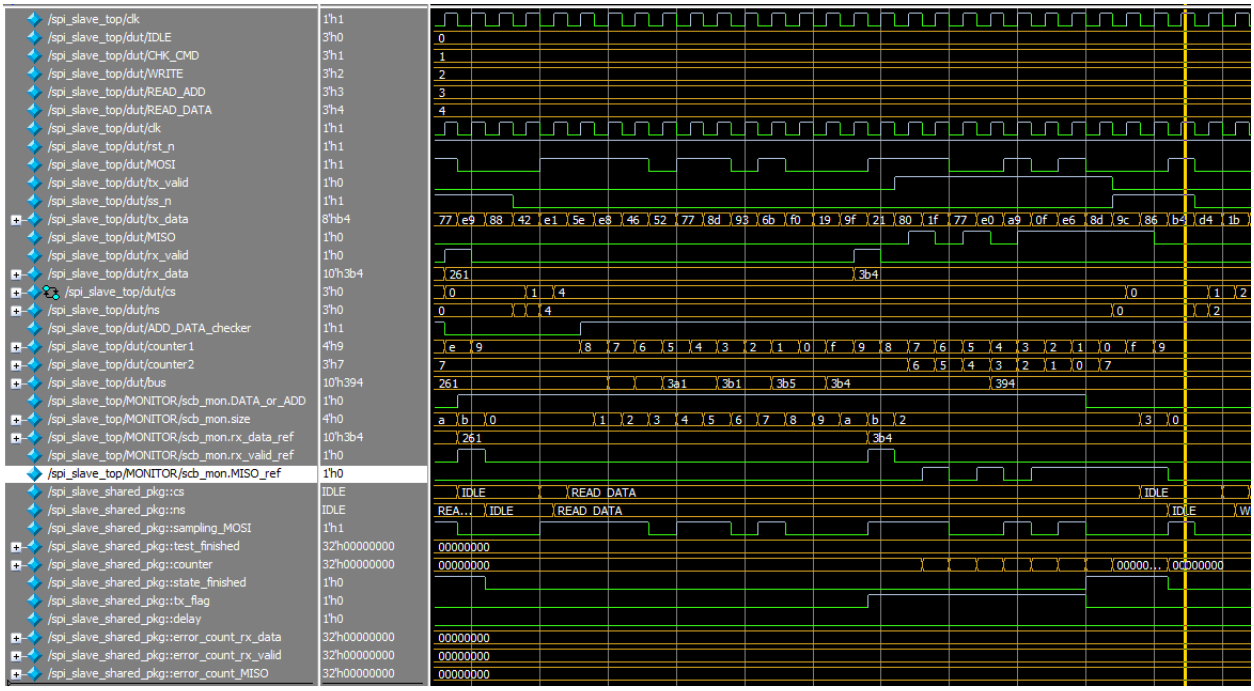


Another proof

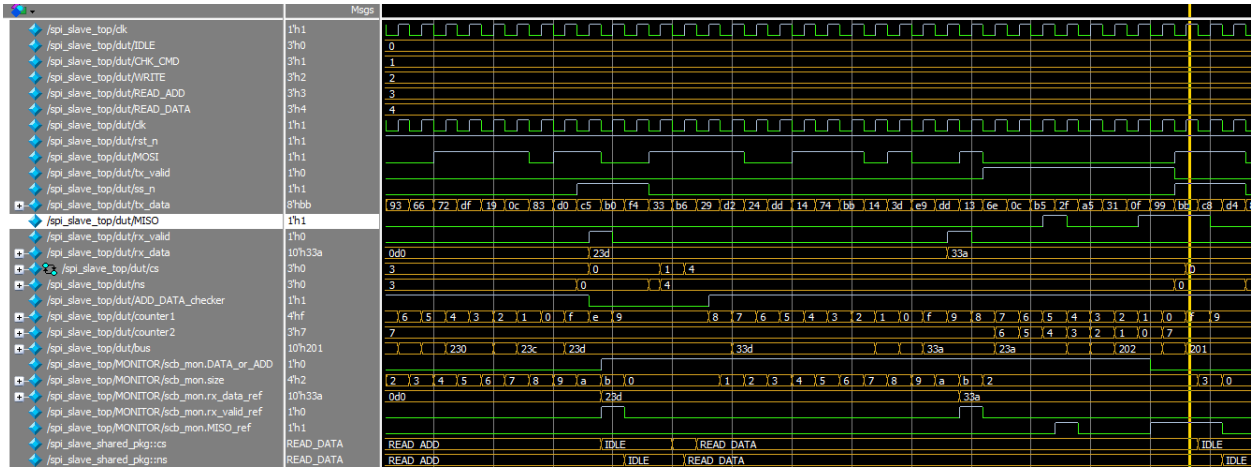
- **READ_DATA state:**



In read data state, rx_data **is sent correctly** **but we care about the MISO signal in this state**, Most Significant two Bits of rx_data[9:8] = 2'b11 as the state is **read Data** (from Specs)

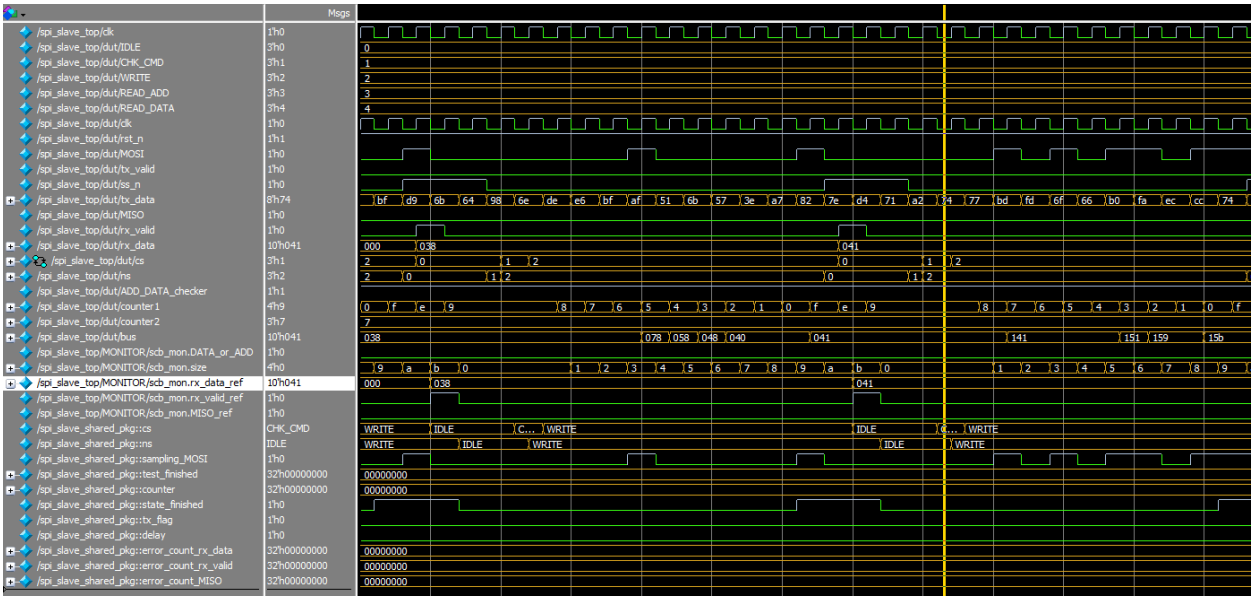


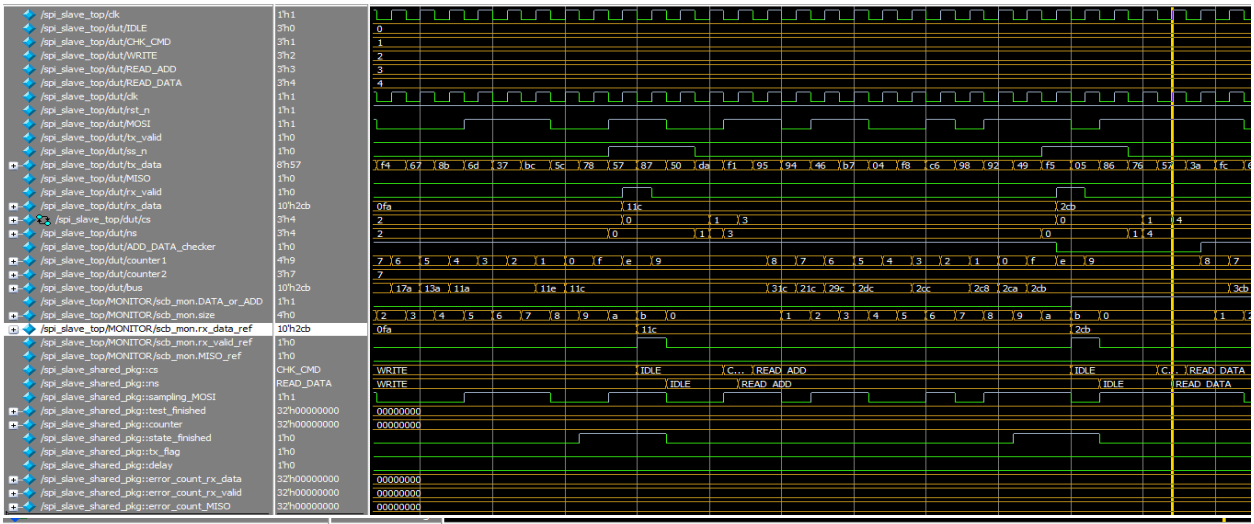
After rx_data is sent, **we will notice the same behavior for MISO signal** and MISO_ref signal which indicates **the correctness of the design**

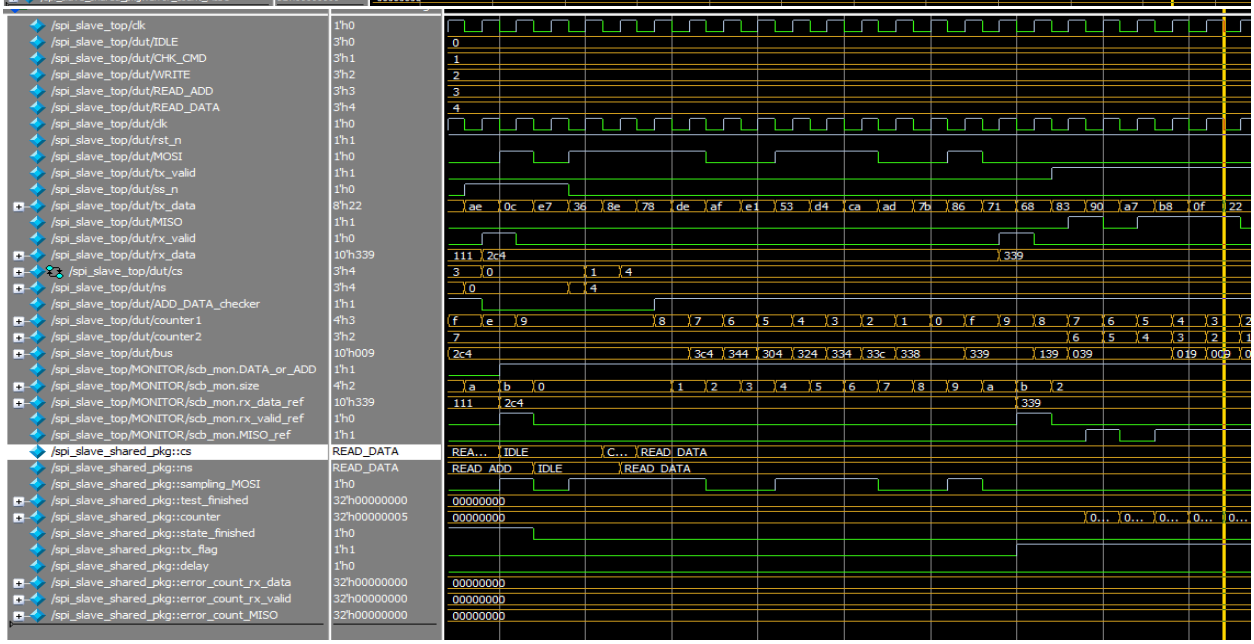


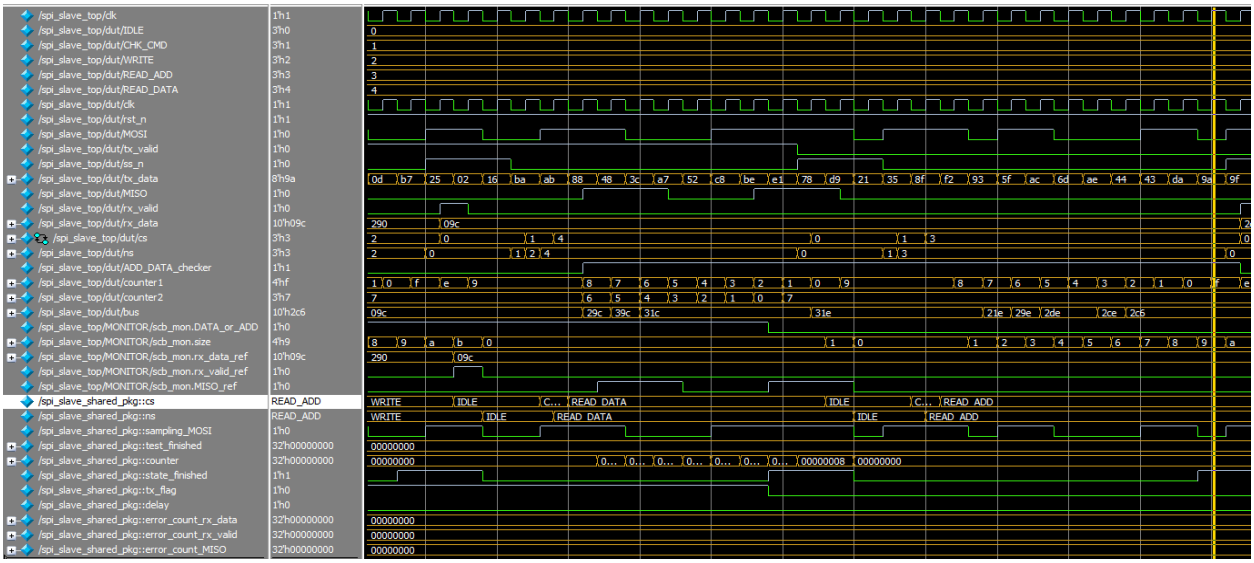
Another proof

- **Random snapshots :**

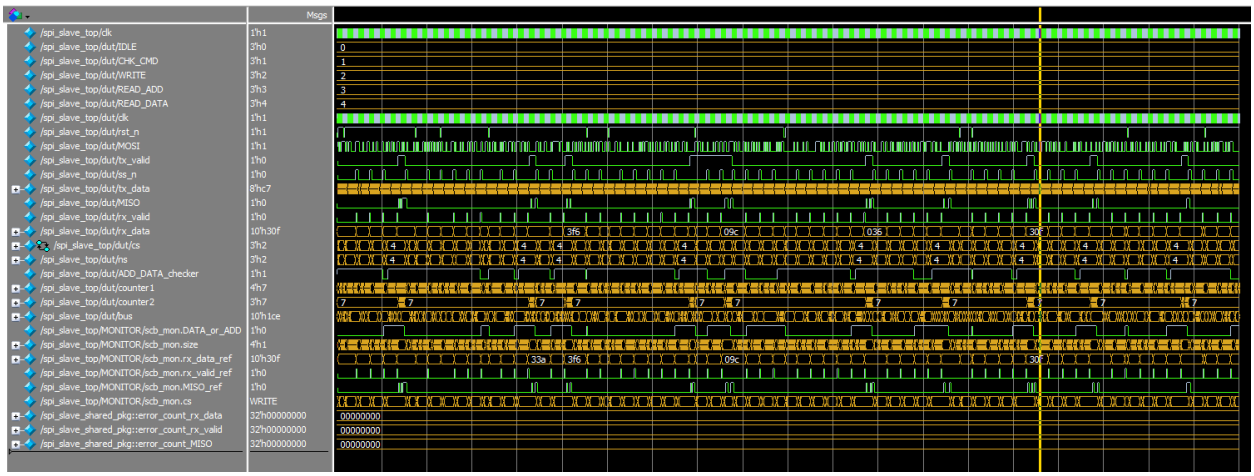



/spi_slave_top/clk
/spi_slave_top/dut/IDLE
/spi_slave_top/dut/CHK_CMD
/spi_slave_top/dut/WRITE
/spi_slave_top/dut/READ_ADD
/spi_slave_top/dut/READ_DATA
/spi_slave_top/dut/clk
/spi_slave_top/dut/rst_n
/spi_slave_top/dut/MOSI
/spi_slave_top/dut/tx_valid
/spi_slave_top/dut/ss_n
/spi_slave_top/dut/tx_data
/spi_slave_top/dut/MISO
/spi_slave_top/dut/rx_valid
/spi_slave_top/dut/rx_data
/spi_slave_top/dut/cs
/spi_slave_top/dut/ns
/spi_slave_top/dut/ADD_DATA_checker
/spi_slave_top/dut/counter1
/spi_slave_top/dut/counter2
/spi_slave_top/dut/bus
/spi_slave_top/MONITOR/scb_mon.DATA_or_ADD
/spi_slave_top/MONITOR/scb_mon.size
/spi_slave_top/MONITOR/scb_mon.rx_data_ref
/spi_slave_top/MONITOR/scb_mon.rx_valid_ref
/spi_slave_top/MONITOR/scb_mon.MISO_ref
/spi_slave_shared_pkg::cs
/spi_slave_shared_pkg::ns
/spi_slave_shared_pkg::sampling_MOSI
/spi_slave_shared_pkg::test_finished
/spi_slave_shared_pkg::counter
/spi_slave_shared_pkg::state_finished
/spi_slave_shared_pkg::tx_flag
/spi_slave_shared_pkg::delay
/spi_slave_shared_pkg::error_count_rx_data
/spi_slave_shared_pkg::error_count_rx_valid
/spi_slave_shared_pkg::error_count_MISO

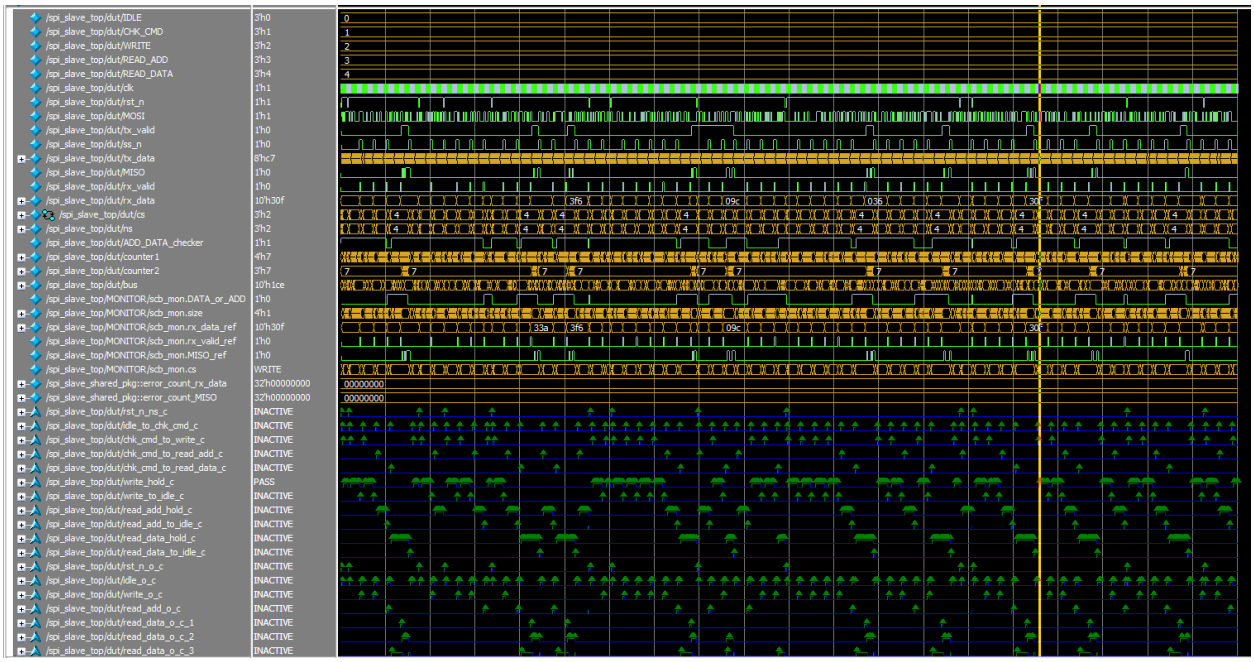

/spi_slave_top/clk
/spi_slave_top/dut/IDLE
/spi_slave_top/dut/CHK_CMD
/spi_slave_top/dut/WRITE
/spi_slave_top/dut/READ_ADD
/spi_slave_top/dut/READ_DATA
/spi_slave_top/dut/clk
/spi_slave_top/dut/rst_n
/spi_slave_top/dut/MOSI
/spi_slave_top/dut/tx_valid
/spi_slave_top/dut/ss_n
/spi_slave_top/dut/tx_data
/spi_slave_top/dut/MISO
/spi_slave_top/dut/rx_valid
/spi_slave_top/dut/rx_data
/spi_slave_top/dut/cs
/spi_slave_top/dut/ns
/spi_slave_top/dut/ADD_DATA_checker
/spi_slave_top/dut/counter1
/spi_slave_top/dut/counter2
/spi_slave_top/dut/bus
/spi_slave_top/MONITOR/scb_mon.DATA_or_ADD
/spi_slave_top/MONITOR/scb_mon.size
/spi_slave_top/MONITOR/scb_mon.rx_data_ref
/spi_slave_top/MONITOR/scb_mon.rx_valid_ref
/spi_slave_top/MONITOR/scb_mon.MISO_ref
/spi_slave_shared_pkg::cs
/spi_slave_shared_pkg::ns
/spi_slave_shared_pkg::sampling_MOSI
/spi_slave_shared_pkg::test_finished
/spi_slave_shared_pkg::counter
/spi_slave_shared_pkg::state_finished
/spi_slave_shared_pkg::tx_flag
/spi_slave_shared_pkg::delay
/spi_slave_shared_pkg::error_count_rx_data
/spi_slave_shared_pkg::error_count_rx_valid
/spi_slave_shared_pkg::error_count_MISO


/spi_slave_top/clk
/spi_slave_top/dut/IDLE
/spi_slave_top/dut/CHK_CMD
/spi_slave_top/dut/WRITE
/spi_slave_top/dut/READ_ADD
/spi_slave_top/dut/READ_DATA
/spi_slave_top/dut/clk
/spi_slave_top/dut/rst_n
/spi_slave_top/dut/MOSI
/spi_slave_top/dut/tx_valid
/spi_slave_top/dut/ss_n
/spi_slave_top/dut/tx_data
/spi_slave_top/dut/MISO
/spi_slave_top/dut/rx_valid
/spi_slave_top/dut/rx_data
/spi_slave_top/dut/cs
/spi_slave_top/dut/ns
/spi_slave_top/dut/ADD_DATA_checker
/spi_slave_top/dut/counter1
/spi_slave_top/dut/counter2
/spi_slave_top/dut/bus
/spi_slave_top/MONITOR/scb_mon.DATA_or_ADD
/spi_slave_top/MONITOR/scb_mon.size
/spi_slave_top/MONITOR/scb_mon.rx_data_ref
/spi_slave_top/MONITOR/scb_mon.rx_valid_ref
/spi_slave_top/MONITOR/scb_mon.MISO_ref
/spi_slave_shared_pkg::cs
/spi_slave_shared_pkg::ns
/spi_slave_shared_pkg::sampling_MOSI
/spi_slave_shared_pkg::test_finished
/spi_slave_shared_pkg::counter
/spi_slave_shared_pkg::state_finished
/spi_slave_shared_pkg::tx_flag
/spi_slave_shared_pkg::delay
/spi_slave_shared_pkg::error_count_rx_data
/spi_slave_shared_pkg::error_count_rx_valid
/spi_slave_shared_pkg::error_count_MISO

- **Full waveform:**



Full waveform proves **none of the error counters were executed**, so we **have no output mismatch** between the **original design** and **the reference model** in the scoreboard file



Full waveform with assertions shows that all assertions are **passed**

Note: **Tx_data is always generating random data every clock**, this is due to that we are **verifying the SPI separately from the RAM** (tx_data is a RAM output that should be received in the SPI slave), we will see **the correct data in tx_data in verifying SPI_WRAPPER section**

- **Transcript**

```
# Loading work.spi_slave_top(fast)
# Loading work.spi_slave_if(fast__1)
# Loading work.SPI_SLAVE(fast)
# Loading work.spi_slave_shared_pkg(fast)
# Loading work.spi_slave_transaction_pkg(fast)
# Loading work.SPI_SLAVE_TB_sv_unit(fast)
# Loading work.spi_slave_tb(fast)
# Loading work.spi_slave_scoreboard_pkg(fast)
# Loading work.spi_slave_coverage_pkg(fast)
# Loading work.SPI_SLAVE_MONITOR_sv_unit(fast)
# Loading work.spi_slave_monitor(fast)
VSIM 29> run -all
# error count = 0, correct count = 1001
# ** Note: $stop    : SPI_SLAVE_MONITOR.sv(31)
#    Time: 20020 ns  Iteration: 0  Instance: /spi_slave_top/MONITOR
# Break in Module spi_slave_monitor at SPI_SLAVE_MONITOR.sv line 31
```