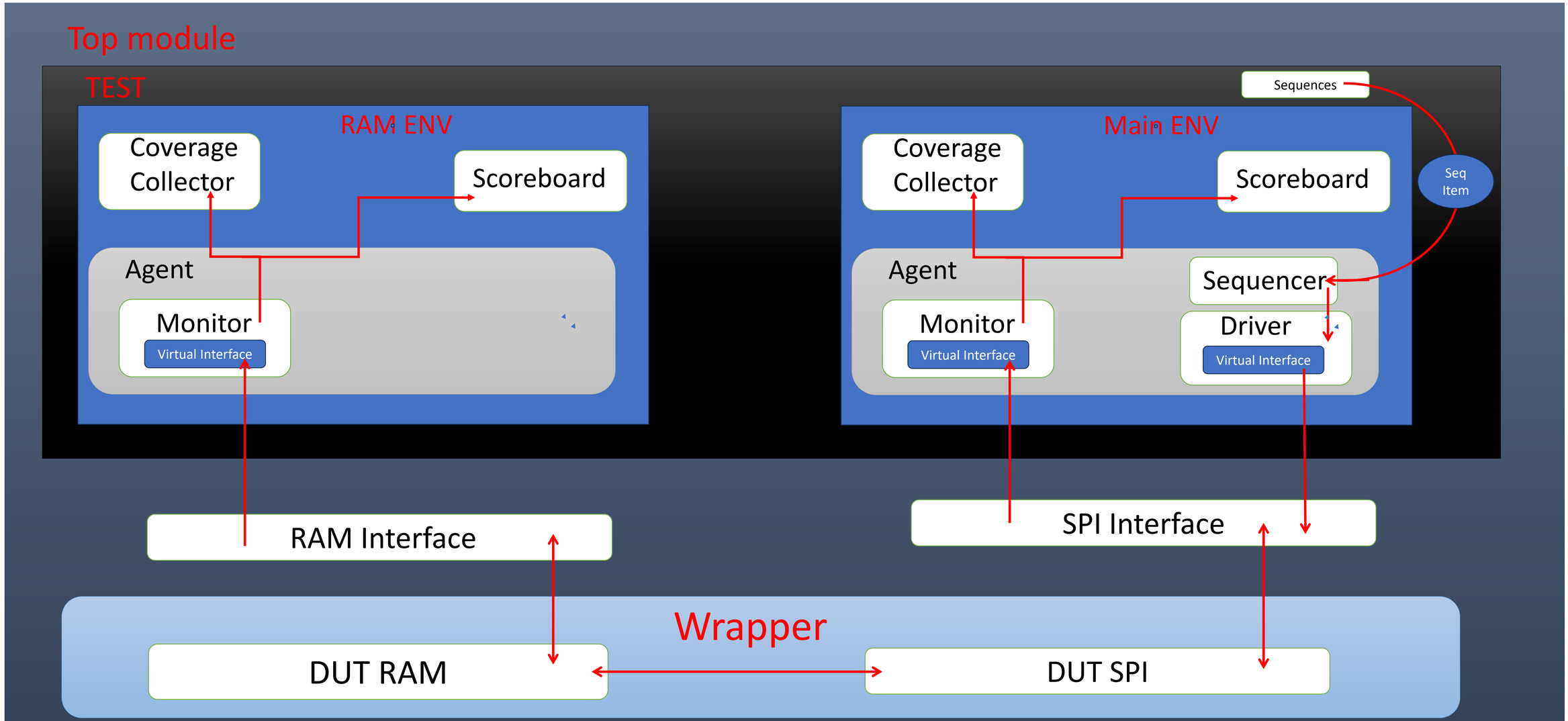
UVM Diagram


Top module
TEST
RAM ENV
Coverage Collector
Scoreboard
Agent
Monitor
Virtual Interface
Main ENV
Sequences
Seq Item
Coverage Collector
Scoreboard
Agent
Sequencer
Monitor
Driver
Virtual Interface
Virtual Interface
RAM Interface
SPI Interface
Wrapper
DUT RAM
DUT SPI

SPI wrapper

Verification plan:

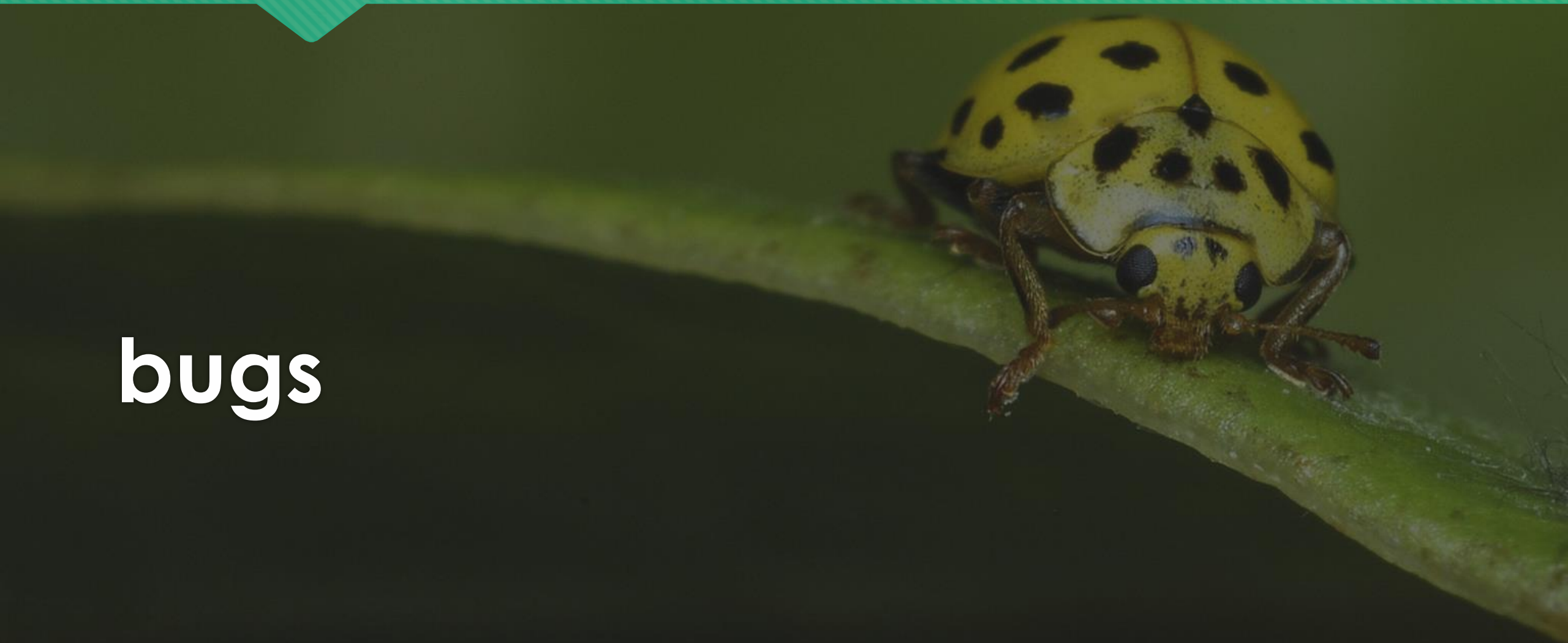
1- Check reset and the values at first state.
2- Generate all possible real scenarios like write then read address then read data or read address then write then read data ….
3- Check reset with all possible values of inputs to make sure that reset has the priority.
4- Outputs rx_valid,rx_data MISO (check functionality).
5- Accept all possible values of rx_data
6- Accept all possible values of tx_data
7- control of SS_n to generate valid scenarios.
8- control of rx_valid with constraints to generate valid scenarios.
9- control of tx_valid with constraints to generate valid scenarios
10- Check MISO with if it correct or not
11- Check FSM in SPI with all transitions.
12- Check no invalid scenarios in the design.
13- Counters for error and correct values

Verification documents

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
SPI_1	FSM IN IDEAL and ss_n=1	Randomization with no constraint	add cover_point for rst_n to come back to ideal	Output Checked against golden model
SPI_2	FSM in write address mode	randomization with constraint to mosi to have the pattern 0 0 0 in conseqtive 3 cycles and then mosi serial data is randomized	add cross bin for cross coverage of in1 & in2 & in3 representing the pattern of mosi	output checked against golden model ..assertion is used to test that when rx_valid rises which means data is ready to be sent... rx_data will be the parallel conversion of the past 10 mosi inputs…(serial to parallel conversion)
SPI_3	FSM in write data mode	Randomization under that in1 and in2 should be equal as control signal must followd by bit equal to it for example 0 for write then 00 for write address also 0 and then 01 for write data.also constraint to mosi to have the patterns of read address before read data which means he cann't override address from constraints	add cross bin for cross coverage of in1 & in2 & in3 representing the pattern of mosi	output checked against golden model ..assertion is used to test that when rx_valid which means data is ready to be sent rx_data is the parallel conversion of the past 10 mosi inputs…(serial to parallel test)
SPI_4	FSM in read address mode	randomization under that in1 and in2 should be equal also constraint to mosi to have the pattern 1 1 0 in no. other constraint	add cross bin for cross coverage of in1 & in2 & in3 representing the pattern of mosi	output checked against golden model.assertion is used to test that when rx_valid which means data is ready to be sent rx_data is the parallel conversion of the past 10 mosi inputs…(serial to parallel test)
SPI_5	FSM in read data mode	randomization under that in1 and in2 should be equal no other constraint also there is a constraint to control the mosi pattern as 1 1 0 must be followd be 1 1 1 eventually by adding flag and post randomize function .as spi memorize read address and fsm will go automatically to read data mode after read address mode unlike write.also add constraint to make spi go through 8 dummy cycles throught read data state to wait data to come back from memory	add cross bin for cross coverage of in1 & in2 & in3 representing the pattern of mosi	output checked against golden model.assertion is used to test that when tx_valid rises which means data to be read is ready in memory the miso output will be the serial conversion of tx_data (parallel to serial test)
SPI_6	CHECK reset	randomize rst_n with constraint to be 2% asserted and 98% not active	coverage point reset	output checked against golden model assertion property (check_rst_n) is used

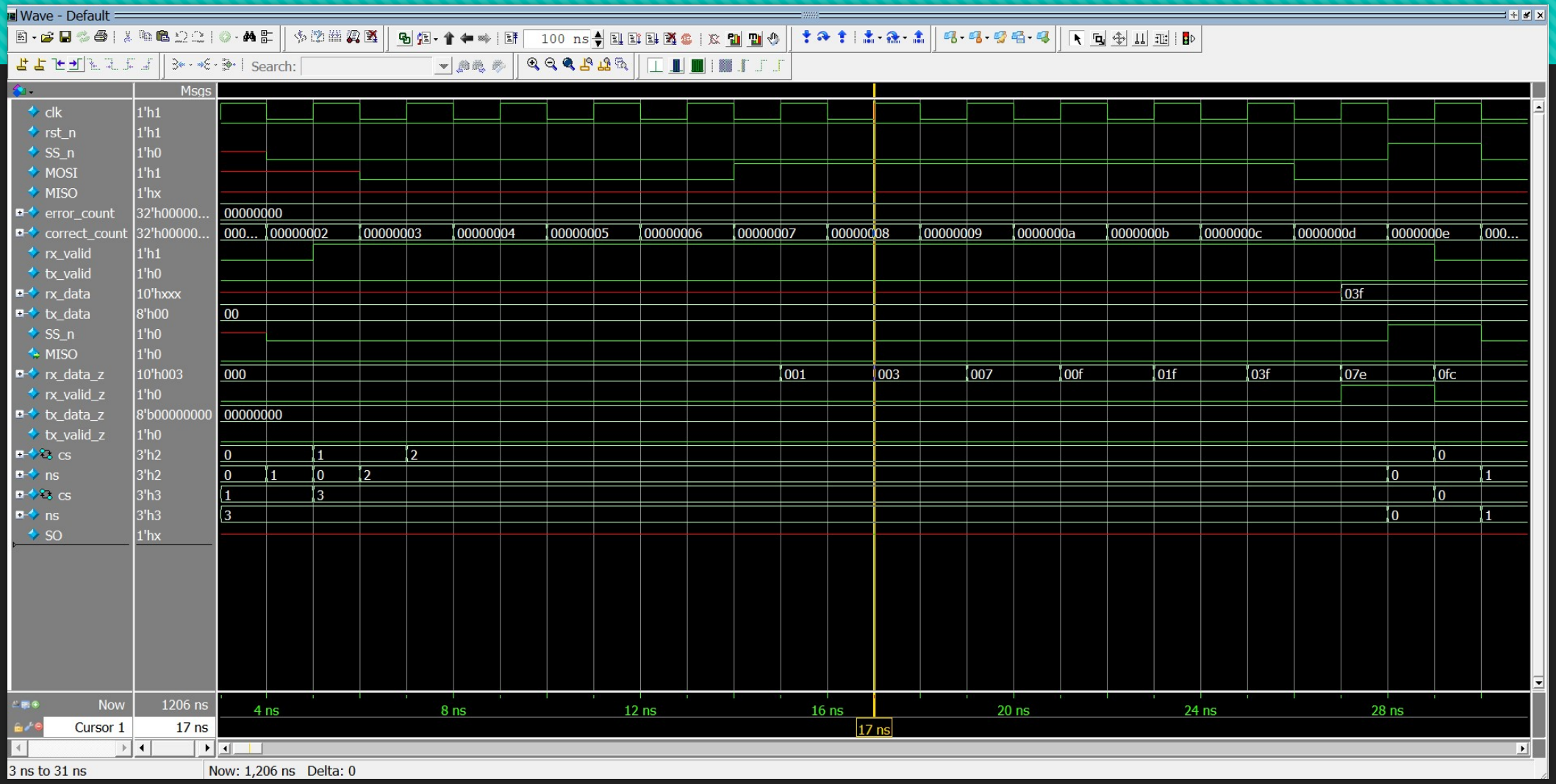
overall I made assertions to (1) check that rx_valid && tx_valid never asserted if SS_n=1 (end of communication)
//also MISO ==0 (2)we check that when SS_n=0 (start of communication) then mosi must follow these patterns
//if mosi=0 for writing then after it 00 or 01

bugs



Due to some counter problems

1. Output of design in not working at all



2- first counter bug

```
always @(posedge clk or negedge SS_n) begin
    if ((cs == WRITE) || (cs == READ_ADD) || (cs == READ_DATA)) begin
        PO <= {PO[8:0], MOSI};
        state_count <= state_count + 1;

        if (state_count == 10) begin
            rx_data <= PO;
            state_count <= 0;
        end
```

state count should stop at 9 not 10

first counter bug (after edit)

```
always @(posedge clk or negedge SS_n) begin
    if ((cs == WRITE) || (cs == READ_ADD) || (cs == READ_DATA)) begin
        rx_data <= {rx_data[8:0], MOSI};
        if (state_count == 9) begin
            state_count <= 0;
        end

         state_count <= state_count + 1;

    end
    if (rx_data[9:8] == 2'b11 && temp) begin
        MISO <= temp[7-final_count];
        final_count <= final_count + 1;
        if (final_count == 9)
        final_count <= 0;
    end
end
```

3. Second counter bug

```
66              MISO<=0;
67              if ( (tx_valid || tx_valid_flag) && (cs == READ_DATA) ) begin
68                  tx_valid_flag<=1;
69                  MISO <= temp[7-final_count];
70                  final_count <= final_count + 1;
71                  if (final_count == 9) begin
72                      final_count <= 0;
73                      tx_valid_flag<=0;
74                  end
75              end
```

Final counter should stop at 7 as it's 8 cycles

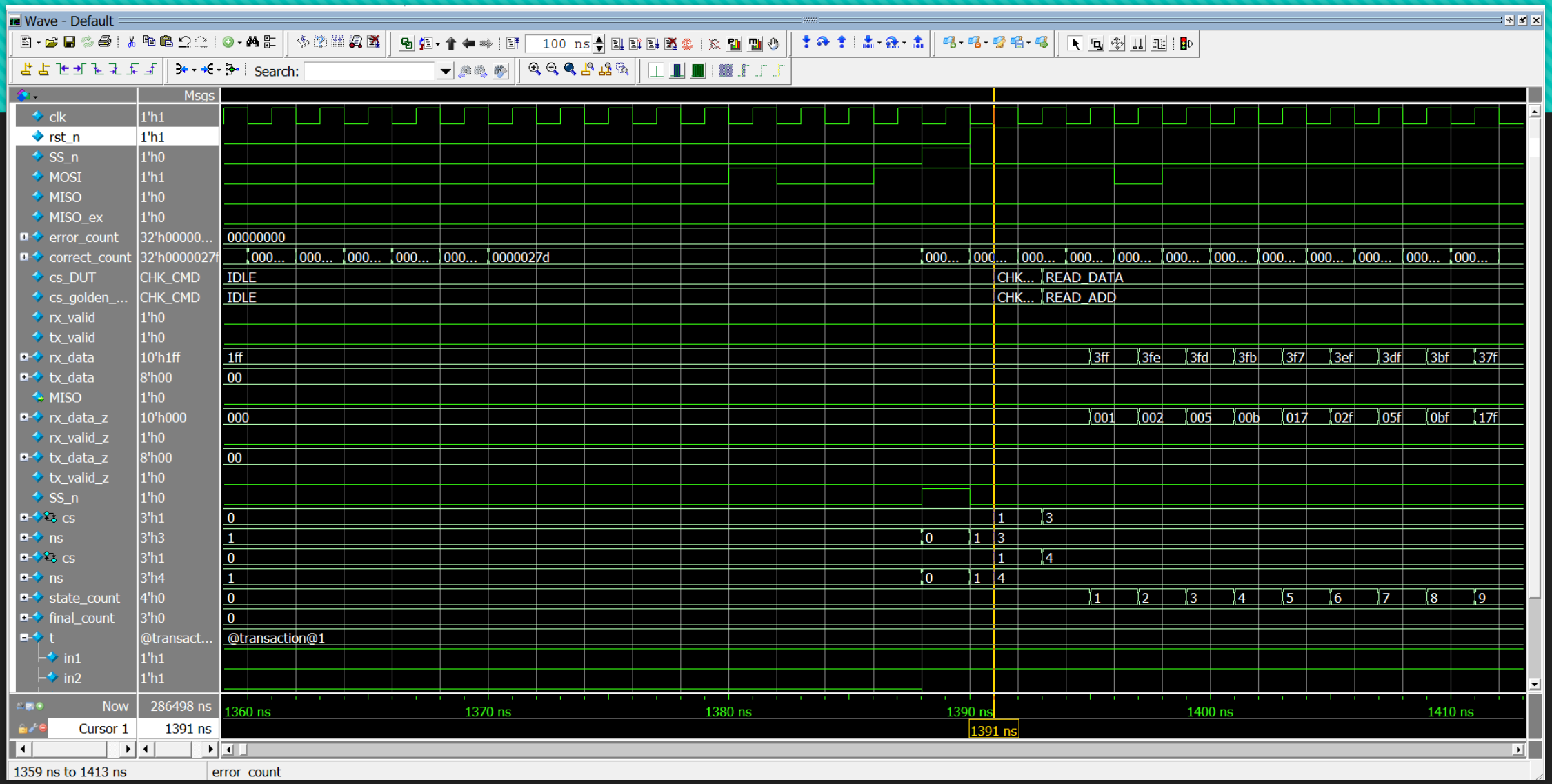
3. Second counter bug (after edit)

```
tx_valid_flag<=1;
MISO <= temp[7-final_count];
final_count <= final_count + 1;
if (final_count == 7) begin
    final_count <= 0;
    tx_valid_flag<=0;
end
```

4. PIPLING RX_data cause extreme errors

```
always @(posedge clk or negedge SS_n) begin
    if ((cs == WRITE) || (cs == READ_ADD) || (cs == READ_DATA)) begin
        PO <= {PO[8:0], MOSI}; state_count <= state_count + 1;
        if (state_count == 10) begin
        rx_data <= PO; state_count <= 0;
        end
    end
    if (rx_data[9:8] == 2'b11 && temp) begin
        SO <= temp[7-final_count];
        final_count <= final_count + 1;
        if (final_count == 10)
        final_count <= 0;
    end
end
```

Also not adding else statement to make rx_data return 0


Wave - Default
Search:
Msgs
clk 1'h1
rst_n 1'h1
SS_n 1'h0
MOSI 1'h1
MISO 1'h0
MISO_ex 1'h0
error_count 32'h00000...
correct_count 32'h0000027f
cs_DUT CHK_CMD
cs_golden_... CHK_CMD
rx_valid 1'h0
tx_valid 1'h0
rx_data 10'h1ff
tx_data 8'h00
MISO 1'h0
rx_data_z 10'h000
rx_valid_z 1'h0
tx_data_z 8'h00
tx_valid_z 1'h0
SS_n 1'h0
cs 3'h1
ns 3'h3
cs 3'h1
ns 3'h4
state_count 4'h0
final_count 3'h0
t @transact...
in1 1'h1
in2 1'h1
00000000
0000027d
IDLE
CHK...
READ_DATA
READ_ADD
1ff
3ff
3fe
3fd
3fb
3f7
3ef
3df
3bf
37f
00
000
001
002
005
00b
017
02f
05f
0bf
17f
@transaction@1
Now
286498 ns
Cursor 1
1391 ns
1360 ns
1370 ns
1380 ns
1390 ns
1400 ns
1410 ns
1359 ns to 1413 ns
error count

5. Counter continue to increase even if state is ideal as no stopping condition adder

```
always @(posedge clk or negedge SS_n) begin
    if ((cs == WRITE) || (cs == READ_ADD) || (cs == READ_DATA)) begin
        PO <= {PO[8:0], MOSI}; state_count <= state_count + 1;
        if (state_count == 10) begin
        rx_data <= PO; state_count <= 0;
        end
    end
    if (rx_data[9:8] == 2'b11 && temp) begin
        SO <= temp[7-final_count];
        final_count <= final_count + 1;
        if (final_count == 10)
        final_count <= 0;
    end
end
```

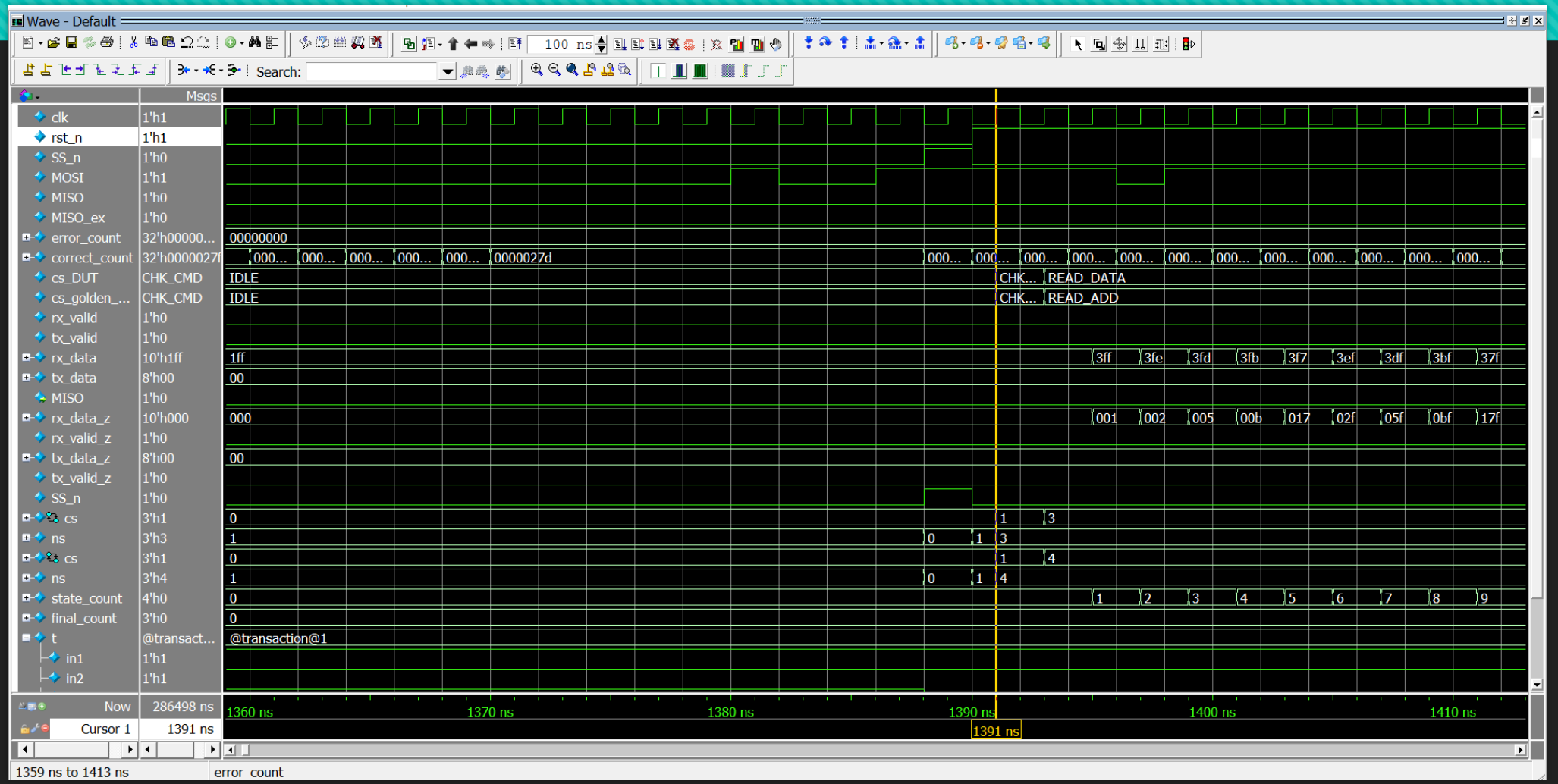
6. Sensitivity list doesn't depend on SS_n

```
always @(posedge clk or negedge SS_n) begin
    if ((cs == WRITE) || (cs == READ_ADD) || (cs == READ_DATA)) begin
        PO <= {PO[8:0], MOSI}; state_count <= state_count + 1;
        if (state_count == 10) begin
        rx_data <= PO; state_count <= 0;
        end
    end
    if (rx_data[9:8] == 2'b11 && temp) begin
        SO <= temp[7-final_count];
        final_count <= final_count + 1;
        if (final_count == 10)
        final_count <= 0;
    end
end
```

BUG 4 & 5 & 6 After Edited

```
always @(posedge clk) begin
    if ((cs == WRITE) || (cs == READ_ADD) || (cs == READ_DATA)) begin
        rx_data <= {rx_data[8:0], MOSI};
        if (state_count == 9) begin
            state_count <= 0;
            rx_valid <= 1;
        end
        else begin
            rx_valid <= 0;
            state_count <= state_count + 1;
        end
```

7- not adding flag to be zero after rst_n cause problems



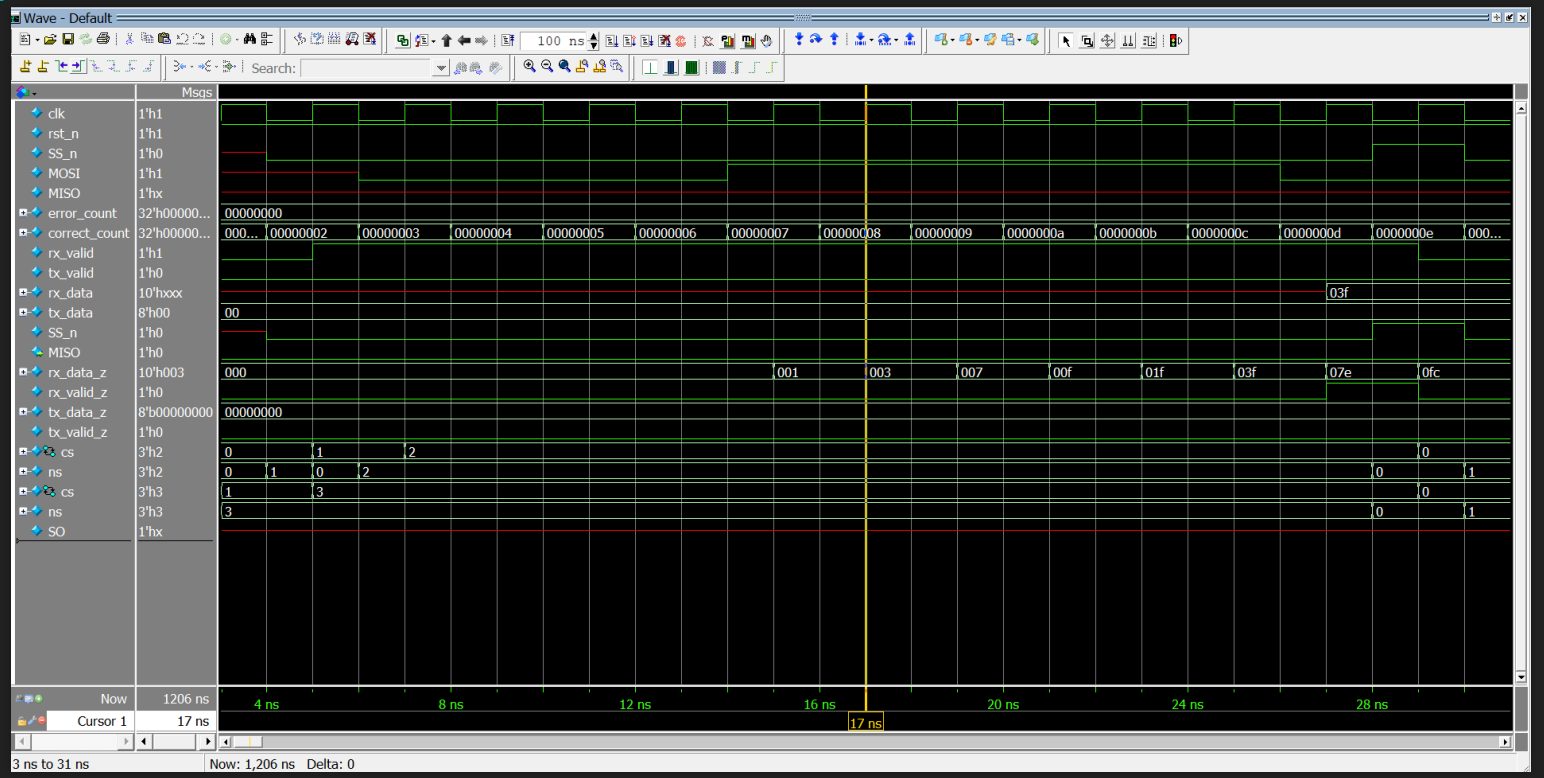
7- not adding flag to be zero after rst_n cause problems

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        cs <= IDLE;
    else
        cs <= ns;
end
```

7- not adding flag to be zero after rst_n cause problems (after edited)

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        flag_rd=0;
        cs <= IDLE;
    end
    else
        cs <= ns;
end
```

8-Serious bug (NO MISO Still)



8-Serious bug

```
14      wire [7:0] temp;
```

```
18      assign temp = (tx_valid)? tx_data: temp;
```

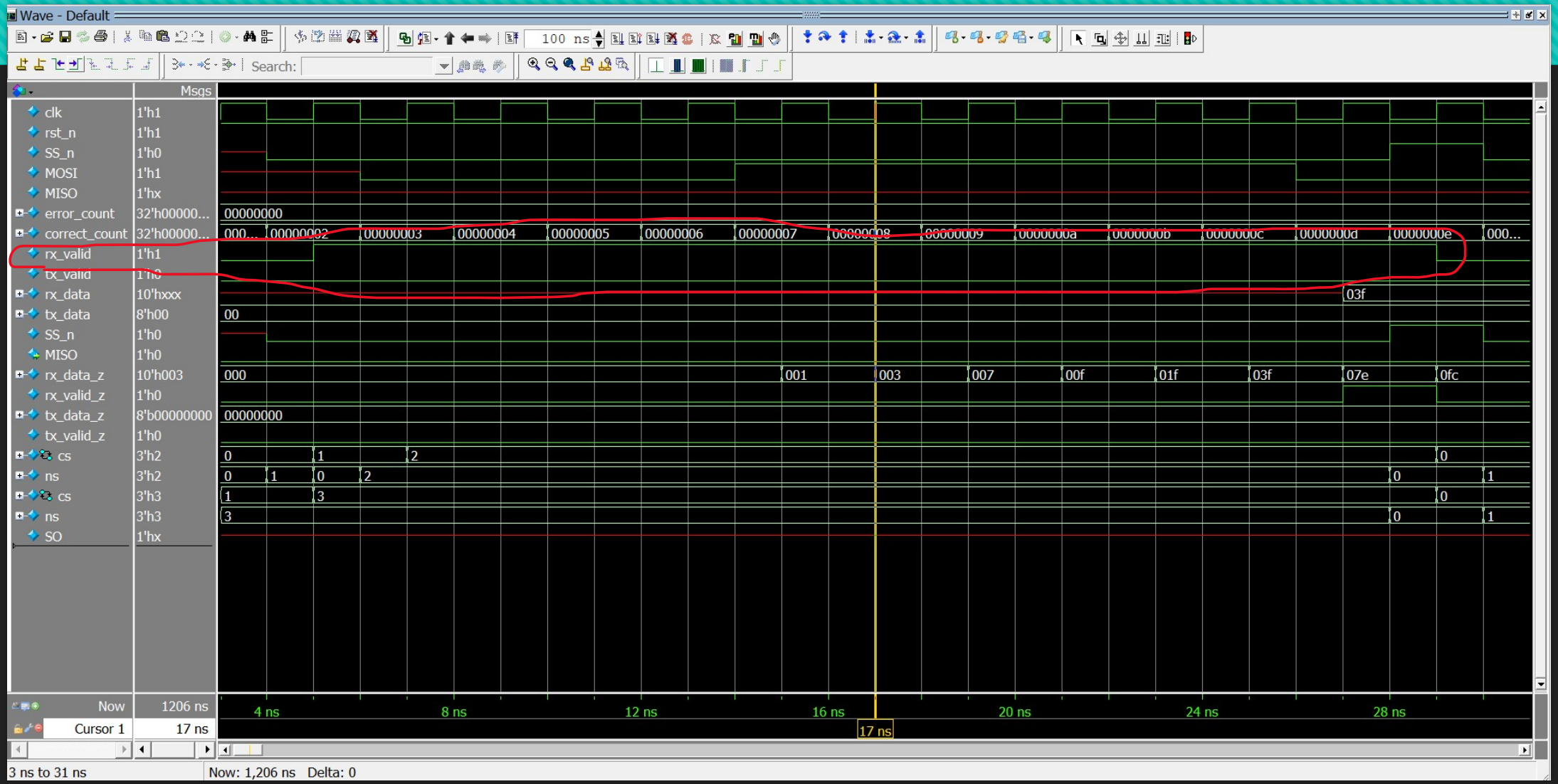
```
38      always @(posedge clk or negedge SS_n) begin
39          if ((cs == WRITE) || (cs == READ_ADD) || (cs == READ_DATA)) begin
40              PO <= {PO[8:0], MOSI}; state_count <= state_count + 1;
41              if (state_count == 10) begin
42              rx_data <= PO; state_count <= 0;
43              end
44          end
45          if (rx_data[9:8] == 2'b11 && temp) begin
46              SO <= temp[7-final_count];
47              final_count <= final_count + 1;
48              if (final_count == 10)
49              final_count <= 0;
50          end
51      end
```

After editing

```
53      always @(posedge clk) begin
54          if ((cs == WRITE) || (cs == READ_ADD) || (cs == READ_DATA)) begin
55              rx_data <= {rx_data[8:0], MOSI};
56              if (state_count == 9) begin
57                  state_count <= 0;
58                  rx_valid <= 1;
59              end
60              else begin
61                  rx_valid <= 0;
62                  state_count <= state_count + 1;
63              end
64          end
65          else begin
66              rx_valid<=0;
67              state_count <= 0;
68          end
69              MISO<=0;
70              if ( (tx_valid || tx_valid_flag)  ) begin
71                  tx_valid_flag<=1;
72                  MISO <= temp[7-final_count];
73                  final_count <= final_count + 1;
74                  if (final_count == 7) begin
75                      final_count <= 0;
76                      tx_valid_flag<=0;
77                  end
78              end
79              else begin
80                  tx_valid_flag<=0;
81                  final_count <= 0;
82              end
83      end
```

Edited part

9- Serious bug.. (rx valid always wrong)



9- Serious bug

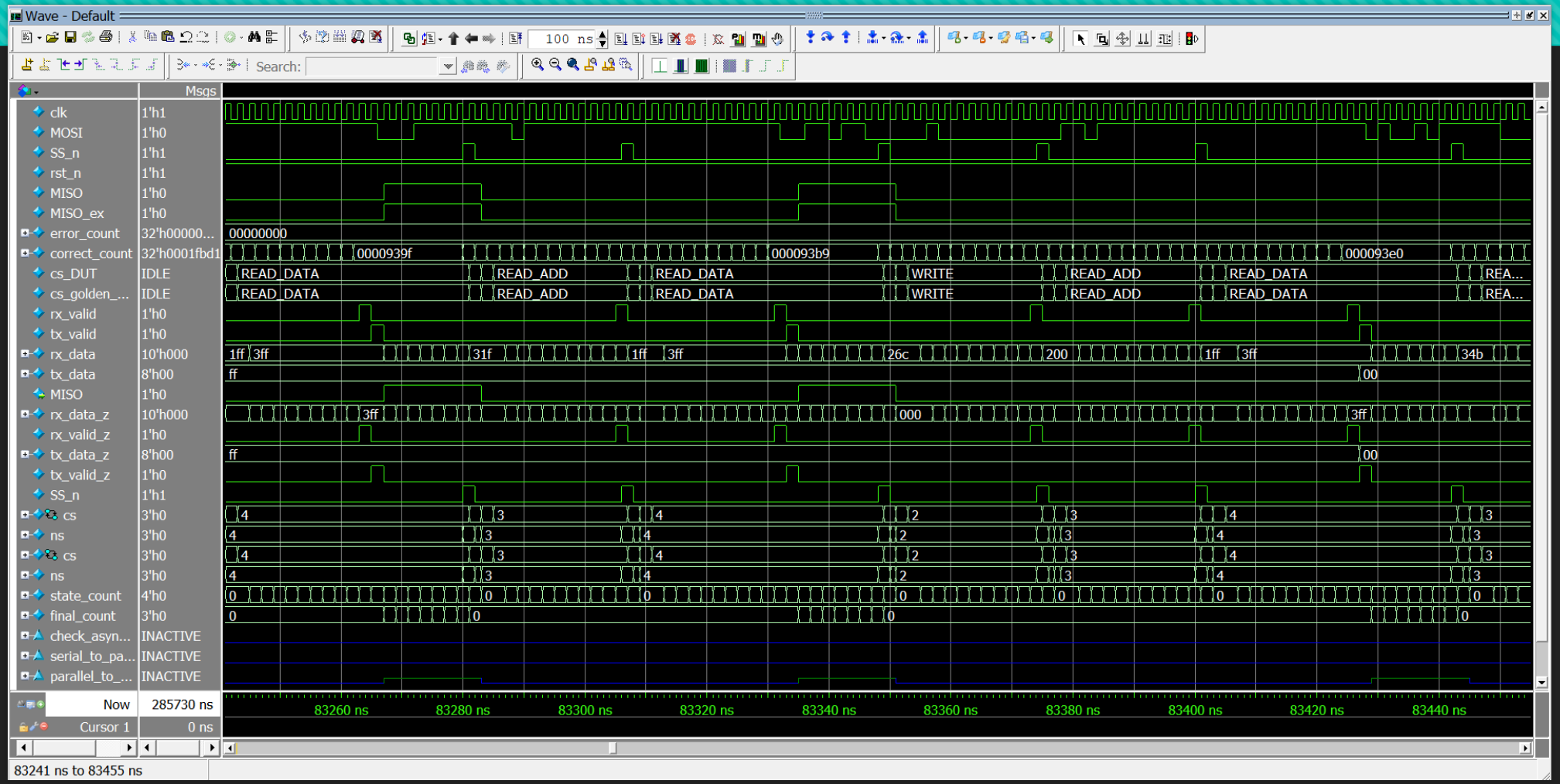
```
28      always @(cs) begin
29          case (cs)
30              IDLE: rx_valid = 0;
31              WRITE: rx_valid = 1;
32              READ_ADD: rx_valid = 1;
33              READ_DATA: rx_valid = 1;
34              default: rx_valid = 0;
35          endcase
36      end
```

After edited

```
always @(posedge clk) begin
    if ((cs == WRITE) || (cs == READ_ADD) || (cs == READ_DATA)) begin
        rx_data <= {rx_data[8:0], MOSI};
        if (state_count == 9) begin
            state_count <= 0;
            rx_valid <= 1;
        end
        else begin
            rx_valid <= 0;
            state_count <= state_count + 1;
        end
    end
    else begin
        rx_valid<=0;
        state_count <= 0;
    end
        MISO<=0;
```

We cancelled the past block and add rx_valid in always block with counters

Finally MISO Worked



SPI Wrapper Transcript

```
O my_test.sv(53) @ 30002: uvm_test_top [run_phase] welcome to uvm
O verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 30002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phas
O SPI_scoreboard.sv(53) @ 30002: uvm_test_top.env.score_board [run_phase] totall successgul transaction=      15001
O SPI_scoreboard.sv(54) @ 30002: uvm_test_top.env.score_board [run_phase] totall error transaction=          0
```

Coverage

- ASSERTION 100%
- CODE COVERAGE 100%
- Functional coverage 100%

Please see reports in zip file


END