SPI Wrapper Design

Design code

```
module SPI_Wrapper(interface_SPI.DUT_Wrapper inst_interface );
    //(MOSI, SS_n, clk, rst_n, MISO);
    logic MOSI, SS_n, clk, rst_n;
    logic MISO;
    logic rx_valid_wrapper, tx_valid_wrapper;
    logic [9:0] rx_data_wrapper;
    logic [7:0] tx_data_wrapper;

    assign clk   = inst_interface.clk ;
    assign SS_n  = inst_interface.SS_n ;
    assign rst_n = inst_interface.rst_n ;
    assign MOSI = inst_interface.MOSI ;
    assign inst_interface.MISO = MISO ;

    SPI_Slave_wrapper DUT (MOSI, SS_n, clk, rst_n, tx_data_wrapper,
tx_valid_wrapper, MISO, rx_data_wrapper, rx_valid_wrapper);
    Dp_Sync_RAM_wrapper RAM (clk, rst_n, rx_data_wrapper, rx_valid_wrapper,
tx_data_wrapper, tx_valid_wrapper);

endmodule
```

SPI slave instantiate

```
module SPI_Slave_wrapper (MOSI, SS_n, clk, rst_n, tx_data, tx_valid, MISO,
rx_data, rx_valid);
    parameter IDLE = 0;
    parameter CHK_CMD = 1;
    parameter WRITE = 2;
    parameter READ_ADD = 3;
     parameter READ_DATA = 4;

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

    reg [2:0] cs, ns;//current state and next state
    reg [9:0] PO;
    reg [7:0] temp;
    reg SO, flag_rd ;
    integer state_count = 0, final_count = 0;
    reg Act_input_output;

    // assign clk = inst_interface.clk;
```

```
    // assign rst_n=inst_interface.rst_n;
    // assign SS_n =inst_interface.SS_n;
    // assign MOSI =inst_interface.MOSI;
    // always@(*)begin
    //     tx_valid=inst_interface.tx_valid;
    //     tx_data =inst_interface.tx_data;
    //     inst_interface.MISO = MISO;
    //     inst_interface.rx_valid =rx_valid;
    //     inst_interface.rx_data  =rx_data;
    // end
    assign MISO = SO; // output of reading
    assign rx_data   = PO ;
// state Memory
    always @(posedge clk ) begin //bug : reset must be syncrouns not Async
        if (!rst_n)begin
            cs <= IDLE;
            flag_rd<=0;
            final_count <=32'hFFFF_FFFF;
            state_count <=32'hFFFF_FFFF;
            temp <= 0 ;
        end
        else
            cs <= ns;
    end
    always @ (cs)begin
        if (cs == IDLE)begin
            rx_valid = 0 ;
            PO  = 0 ;
            state_count = 0 ;
            final_count = 0 ;
            SO = 0 ;
            Act_input_output = 0;
        end
    end

    always @(posedge clk) begin//* bug

        case (cs)
              IDLE : begin

                rx_valid = 0 ;
                PO  = 0 ;
                state_count = 0 ;
                final_count = 0 ;
                SO = 0 ;
```

```
                Act_input_output = 0;
                     end
            WRITE: begin //done

              if (state_count < 10 )begin //0 ,
                  PO = {PO[8:0] , MOSI} ;
                  state_count = state_count + 1 ;
                  if (PO[9] ==1'b0 && state_count ==10)begin
                      rx_valid = 1 ;
                  end
                  else
                      rx_valid = 0 ;
              end
              else
                  rx_valid = 0 ;

                  end
            READ_ADD: begin // done

              if (state_count<10 )begin
                  PO = {PO[8:0] , MOSI} ;
                  state_count = state_count + 1 ;
                  rx_valid = 0 ;
              end
              if (PO[9:8]==2'b10 && state_count==10)begin
                  rx_valid = 1 ;
                  flag_rd = 1;
              end
              else
                  rx_valid = 0 ;
                     end
            READ_DATA: begin // done

              if (Act_input_output == 0)begin
                  if (state_count<9 )begin
                      PO = {PO[8:0] , MOSI} ;
                      state_count = state_count + 1 ;
                      rx_valid = 0 ;
                  end
                  else begin
                      PO = {PO[8:0] , MOSI} ;
                      state_count = state_count + 1 ;
                      rx_valid = 0 ;
                  end
```

```
                    if (PO[9:8]==2'b11 && state_count== 10 )begin
                        rx_valid = 1 ;
                        Act_input_output = 1;
                        flag_rd = 0;
                    end
                end
                else begin
                    state_count = state_count + 1 ;
                    rx_valid = 0 ;
                    if (tx_valid && state_count == 12 )begin
                        temp = tx_data ;
                    end
                    if (state_count >= 12 && final_count <= 7)begin
                        SO = temp [7 - final_count ];
                        rx_valid =0 ;
                        final_count = final_count + 1 ;
                    end
                end
                        end
              default: begin
                rx_valid = 0 ;
                PO  = 0 ;
                state_count = 0 ;
                final_count = 0 ;
                SO = 0 ;
                Act_input_output = 0;

                   end
             endcase
end

always @(*) begin

    case (cs)
        IDLE:
            if (SS_n)
                ns = IDLE;
            else
                ns = CHK_CMD;
        CHK_CMD:
            if (SS_n)
                ns = IDLE;
            else begin//SS_n = 0
                if (MOSI==0) //MOSI = 0
                    ns = WRITE;
```

```
                    else  begin //MOSI = 1
                        if (!flag_rd) //flag_rd = 0
                            ns = READ_ADD;
                        else  //flag_rd = 1
                            ns = READ_DATA;
                    end
                end
            WRITE:
                if (SS_n)
                    ns = IDLE;
                else
                    ns = WRITE;
            READ_ADD:
                if (SS_n)
                    ns = IDLE;
                else begin
                    ns = READ_ADD;
                    // flag_rd = 1;
                end
            READ_DATA:
                if (SS_n)
                    ns = IDLE;
                else begin
                    ns = READ_DATA;
                    // flag_rd = 0;
                end

            default: ns = IDLE;
        endcase
    end

endmodule
```

RAM Design instantiate

```
module Dp_Sync_RAM_wrapper(clk, rst_n, din, rx_valid, dout, tx_valid);
    parameter MEM_DEPTH = 256;
    parameter ADDR_SIZE = 8;

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

    reg [7:0] mem [MEM_DEPTH-1:0];
    reg [ADDR_SIZE-1:0] rd_addr,wr_addr ;
```

	`` always @(posedge clk) begin // first bug : rst must be syncrouns no Ashync if (!rst_n) begin dout <= 0; tx_valid <= 0; //second bug : tx must take zero when rst_n is asserted wr_addr <= 0 ; rd_addr <= 0 ;//3th bug : when rst_n asserted , addr and not 2 internal signals it only one end else if (rx_valid) begin tx_valid <= 0; case (din[9:8]) 2'b00: wr_addr <= din[7:0]; 2'b01: mem[wr_addr] <= din[7:0]; 2'b10: rd_addr <= din[7:0]; default: {dout, tx_valid} <= {mem[rd_addr], 1'b1}; endcase end else begin tx_valid <= 0 ; end //4th bug : when rx_valid not asserted then tx_valid must down to zero end endmodule ```
Top code	``` module top(); bit clk; always #10 clk=~clk; interface_SPI inst_interface(clk); SPI_Wrapper DUT_Wrapper (inst_interface); tb_SPI_Wrapper TESTBENCH (inst_interface); endmodule ```
Interface code	``` interface interface_SPI (clk); //(MOSI, SS_n, clk, rst_n, MISO); input bit clk ; logic MOSI, SS_n, rst_n; logic MISO; modport DUT_Wrapper (input clk , rst_n , MOSI , SS_n, output MISO); ```

```
    modport TESTBENCH  (input clk, MISO ,
                        output  rst_n, SS_n, MOSI);

endinterface
```

Packages code

```
package SPI_slave_package;
    class SPI_class ;
    logic clk;
    logic SS_n;
    logic MOSI ;
    logic MISO;
    logic [1:0] opcode ;
    logic [7:0]  data ,address , Din ,add_rd,add_wr,data_rd,data_wr ;
    logic [10:0] data_reg;   // Register to store parallel data
    logic [10:0] expected_add,expected_Data;
    logic [7:0] parallel_data [0 : 255];
    logic [7:0] parallel_address[0 : 255];
    logic [7:0] Queue [$];
    integer shift_data =0 , shift_address = 0;
    integer i =0;
    logic  rst_n;

function void Set_value();
    if (SS_n==0 && rst_n==1)begin
        case (opcode)
        2'b00: expected_add  = {expected_add[9:0],MOSI};
        2'b01: expected_Data = {expected_Data[9:0],MOSI};
        2'b10: expected_add  = {expected_add[9:0],MOSI};
        2'b11: expected_Data = {expected_Data[9:0],MOSI};
        endcase // opcode
    end
endfunction

function void Data_out();
    data = {data,MISO};
endfunction

function void Get_information();
    for (int i = 0; i < 256; i++) begin
        Queue.push_front(i);
    end
    Queue.shuffle();
    for (int i = 0; i < 256; i++) begin
        parallel_address[i]=Queue.pop_front();
```

```
    end
    for (int i = 255; i >= 0; i--) begin
        Queue.push_front(i);
    end
    Queue.shuffle();
    for (int i = 0; i < 256; i++) begin
        parallel_data[i]=Queue.pop_front();
    end
endfunction

function void ParallelToSerial();
    if (SS_n==0 && rst_n==1)begin//*
        // data_reg = {data_reg[9:0], 1'b0}; // Shift left
        // expected_add = {expected_add[9:0],MOSI};
        if (opcode==2'b00 || opcode ==2'b10)begin//*
            // Load parallel data into the data register on the rising edge
of the clock
            if (shift_address == 5'b0) begin//*
                data_reg = {opcode[1],opcode,parallel_address[i]};
                address = parallel_address[i];
                if (opcode == 2'b00)
                    add_wr = parallel_address[i];
                else
                    add_rd = parallel_address[i];
            end//*
            // Shift out data serially
            if (shift_address < 11) begin//*
                MOSI = data_reg[10];
                 // expected_add = {expected_add[9:0],MOSI};
                data_reg = {data_reg[9:0], 1'b0}; // Shift left
                shift_address = shift_address + 1'b1;
                if (shift_address == 11)
                    shift_address = 0 ;
            end //*
        end//*
        else if (opcode==2'b01 || opcode==2'b11) begin
            if (shift_data== 5'b0)begin
                data_reg = {opcode[1],opcode,parallel_data[i]};
                Din = parallel_data[i] ;
                if (opcode == 2'b01)
                    data_wr = parallel_data[i] ;
                else
                    data_rd = parallel_data[i] ;
            end
            if (shift_data < 11) begin
```

```
                MOSI = data_reg[10];
                data_reg = {data_reg[9:0], 1'b0}; // Shift left
                // expected_Data = {expected_Data[9:0],MOSI};
                shift_data = shift_data + 1'b1;
                if (shift_data == 11)
                    shift_data = 0 ;
            end
            // if (opcode==2'b11 && shift ==0)
            //     i++;
        end
    end
    else
        MOSI = 1 ;
endfunction

function void select_SS(logic selet);
        SS_n=selet;
    endfunction

function void opcode_0(logic[1:0] state);
    opcode =state;
    if (rst_n==0)
        opcode=2'b00;
    else if (opcode==2'b00)
        opcode=2'b01;
    else if (opcode==2'b01)
        opcode=2'b10;
    else if (opcode==2'b10)
        opcode=2'b11;
    else if (opcode==2'b11)
        opcode=2'b00;

endfunction

function void writing(logic[1:0] state);
 opcode =state;
 if (rst_n==0)
        opcode=2'b00;
    else if (opcode==2'b00)
        opcode=2'b01;
    else if (opcode==2'b01)
        opcode=2'b00;
    else
        opcode=2'b00;
```

```
endfunction

function void Reading(logic[1:0] state);
 opcode =state;
if (rst_n==0)
        opcode=2'b00;
    else if (opcode==2'b11)
        opcode=2'b10;
    else if (opcode==2'b10)
        opcode=2'b11;
    else
        opcode=2'b10;

endfunction

function void Increment_array();
    i=(i+1)%255 ;
    shift_address = 0 ;
    shift_data    = 0 ;
endfunction

covergroup SPI_Wrapper@(posedge clk);
    reset: coverpoint rst_n{
        bins reset_asserted ={0};
        bins reset_disable  ={1};
        }
    Address : coverpoint address ;
    Data : coverpoint Din ;
    Opcode: coverpoint opcode {
        bins writing_complete = (2'b00 => 2'b01);
        bins invalid_0 = (2'b00 => 2'b11);
        bins invalid_2 = (2'b01 => 2'b10);
        bins invalid_3 = (2'b10 => 2'b00);
        bins repeat_writing = (2'b01 => 2'b00);
        bins repeat_reading = (2'b11 => 2'b10);
        bins reading_complete = (2'b10 => 2'b11);
        }
    Send_Address_Data : coverpoint opcode{
     bins reading_add  = {2'b10};
     bins writing_add  = {2'b00};
     bins writing_data = {2'b01};
    }
    Recieving_Data :coverpoint opcode{
     bins reading_data = {2'b11};
    }
```

	Starting_Communication : coverpoint SS_n{ bins Start = {0}; bins End = {1}; bins transaction_back = (0 => 1 => 0) ; } Data_out : coverpoint data ; Sending_Address : cross Send_Address_Data , Address ; Sending_Data : cross Send_Address_Data , Data ; Receiving : cross Recieving_Data , Data_out ; endgroup SPI_Wrapper =new(); endclass endpackage
Testbench code	import SPI_slave_package::*; module tb_SPI_Wrapper (interface_SPI.TESTBENCH inst_interface); SPI_class Wrapper =new(); logic MOSI, SS_n, clk, rst_n; logic MISO; logic [1:0] state; logic [7:0] RAM [255:0]; logic [7:0] output_parallel_MISO; assign clk =inst_interface.clk; assign MISO = inst_interface.MISO ; assign inst_interface.MOSI=MOSI; assign inst_interface.SS_n=SS_n; assign inst_interface.rst_n=rst_n; integer correct_counts=0; integer error_counts =0; logic repeater=0; always@(clk)begin Wrapper.clk=clk; end task instantiate(); SS_n = Wrapper.SS_n ; rst_n = Wrapper.rst_n ; Wrapper.ParallelToSerial();

```
    MOSI    =Wrapper.MOSI;

endtask

task reset_Asserted();
    Wrapper.rst_n = 0 ;
    Wrapper.select_SS(1);
    instantiate();
    Wrapper.opcode_0(state);
    reset_check();
    Wrapper.rst_n = 1;
    rst_n = Wrapper.rst_n;

endtask

task reset_check();
    @(negedge clk)begin
                if (MISO==0 )begin
                    correct_counts++;
                end
                else begin
                    error_counts++;
                    $display("%0t ns , there is an error : MISO = %0d and
MISO_expected = %0d ",$time(),MISO, 0 );
                end
    end
endtask
task delay_cycle();
    @(negedge clk) repeater++ ;
endtask

task delay_11cycles();
    repeat(11)begin
        instantiate();
        @(negedge clk) begin
            Wrapper.Set_value();
        end
    end
    if (state == 2'b01)
        RAM[Wrapper.add_wr] = Wrapper.data_wr;
endtask
task delay_18cycles();
    delay_11cycles();
    delay_cycle();
```

```
    Wrapper.data_rd = RAM[Wrapper.add_rd] ;

    output_parallel_MISO = Wrapper.data_rd ;
    for(int i = 7 ; i >= 0 ; i--) begin
        @(negedge clk)begin
                if (MISO == output_parallel_MISO[i])begin
                    correct_counts++;
                    Wrapper.MISO =MISO;
                    Wrapper.Data_out();
                end
                else begin
                    error_counts++;
                    $display("%0t ns , there is an error : MISO = %0d and
MISO_expected = %0d ",$time(),MISO,Wrapper.Din[i] );
                end
        end
    end
endtask

always @(Wrapper.opcode)begin
        state = Wrapper.opcode;
    end

initial begin
    Wrapper.Get_information();
    reset_Asserted ();//task for reset
    for (int i = 0; i < 1300 ; i++) begin
        reset_Asserted ();//task for reset
        Wrapper.select_SS(1);
        instantiate();
        reset_check();
        for (int j = 0; j < 4; j++) begin
            Wrapper.select_SS(0);//start communication
            SS_n = 0;
            delay_cycle();
            if (rst_n == 0)
                reset_check();
            else begin
                if (state==2'b11)
                delay_18cycles();
                else
                delay_11cycles();
            end
            Wrapper.MISO =MISO;
            Wrapper.select_SS(1);//end communication
```

```
            instantiate();
            reset_check();

            if (i < 255)
            Wrapper.opcode_0(state);
            else if (i==255 || i < 555) begin
                // reset_Asserted ();
                Wrapper.writing(state);
            end
            else if (i==555 || i < 850)begin
                // reset_Asserted ();
                Wrapper.Reading(state);
            end
            else if (i==850 || i < 1300) begin
                // reset_Asserted ();
                Wrapper.opcode_0(state);
            end
        end
        Wrapper.Increment_array();
        if (i==1299)begin
            reset_Asserted ();//task for reset
            Wrapper.select_SS(1);
            instantiate();
            reset_check();
            Wrapper.select_SS(0);//start communication
            SS_n = 0;
            Wrapper.opcode = 2'b11 ;
            delay_cycle();
            repeat(11)begin
                instantiate();
                @(negedge clk) begin
                    repeater++;
                end
            end
            delay_cycle();
            repeat(8)begin
                 @(negedge clk) begin
                    if (MISO==0)
                        correct_counts++;
                    else
                        error_counts++;
                end
            end
            Wrapper.MISO =MISO;
            Wrapper.select_SS(1);//end communication
```

```
            instantiate();
            reset_check();
        end
    end
    Wrapper.select_SS(1);//end communication
    instantiate();
    delay_cycle();
    if (MISO == 0 )
        correct_counts++;
    else
        error_counts ++;
    Wrapper.select_SS(0);//end communication
    instantiate();
    delay_cycle();
    Wrapper.select_SS(1);//end communication
    instantiate();
    delay_cycle();
        if (MISO == 0 )
        correct_counts++;
    else
        error_counts ++;
    Wrapper.select_SS(0);//end communication
    instantiate();
    delay_cycle();
    $display("correct_counts =%0d and error_counts=%0d
",correct_counts ,error_counts );
    $stop;
end

endmodule
```

Do file

```
vlib work
vlog package.sv SPI_SLAVE.sv testbench.sv SPI_wrapper_sv.sv top.sv interface.sv RAM.sv Slave.sv +cover
vsim -voptargs=+acc work.top -cover
run -all
coverage save top.ucdb -du SPI_Slave -onexit
coverage report -detail -cvg -comments -output fcover_report.txt {}
quit -sim
vcover report top.ucdb -details -annotate -all -output Code_coverage_report.txt
```

Code Coverage

```
Coverage Report by instance with details

================================================================================
=== Instance: /\top#DUT_Design
=== Design Unit: work.FIFO
================================================================================
Branch Coverage:
  Enabled Coverage         Bins     Hits   Misses  Coverage
  ----------------         ----     ----   ------ --------
  Branches                   25       25        0  100.00%

================================Branch Details================================

Branch Coverage for instance /\top#DUT_Design
```

```
  Line       Item                 Count     Source
  ----       ----                 -----     ------
 File FIFO_design_sv.sv
------------------------------------IF Branch------------------------------------
  39                              6019    Count coming in to IF
  39          1                   2475                if (!rst_n) begin
  43          1                   2443                else if (wr_en ) begin
  55          1                   1101                else begin
Branch totals: 3 hits of 3 branches = 100.00%

------------------------------------IF Branch------------------------------------
  44                              2443    Count coming in to IF
  44          1                   2431                                  if (full==0)begin
  50          1                     12                                  else begin
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  62                              5407    Count coming in to IF
  62          1                   2225                if (!rst_n) begin
  66          1                   1078                else if (rd_en ) begin
  75          1                   2104                else
Branch totals: 3 hits of 3 branches = 100.00%

------------------------------------IF Branch------------------------------------
  67                              1078    Count coming in to IF
  67          1                    658                                  if (empty==0)begin
  72          1                    420                                  else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  80                              5665    Count coming in to IF
  80          1                   2327                if (!rst_n) begin
  83          1                   3338                else begin
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  84                              3338    Count coming in to IF
  84          1                   1684                                  if                  ( ({wr_en, rd_en} == 2'b10) && !full)
  86          1                    204                                  else if ( ({wr_en, rd_en} == 2'b01) && !empty)
  88          1                    299                                  else if (({wr_en, rd_en} == 2'b11) && empty)
  90          1                      6                                  else if (({wr_en, rd_en} == 2'b11) && full)
                                  1145    All False Count
Branch totals: 5 hits of 5 branches = 100.00%

------------------------------------IF Branch------------------------------------
  96                              3074    Count coming in to IF
  96          1                     14                if (count==FIFO_DEPTH )
  98          1                   3060                else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  100                             3074    Count coming in to IF
  100         1                    983                if (count==0)
  102         1                   2091                else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  104                             3074    Count coming in to IF
  104         1                     26                if (count==(FIFO_DEPTH-1))
  106         1                   3048                else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
  108                             3074    Count coming in to IF
  108         1                   1035                if (count == 1)
  110         1                   2039                else
Branch totals: 2 hits of 2 branches = 100.00%


Condition Coverage:
  Enabled Coverage          Bins   Covered   Misses  Coverage
  ----------------          ----   -------   ------  --------
  Conditions                  16        16        0  100.00%

================================Condition Details================================

Condition Coverage for instance /\top#DUT_Design  --

 File FIFO_design_sv.sv
----------------Focused Condition View-------------------
Line      84 Item    1  ((~rd_en && wr_en) && ~full)
Condition totals: 3 of 3 input terms covered = 100.00%

 Input Term   Covered  Reason for no coverage   Hint
 ----------  --------  ----------------------  -------------
    rd_en        Y
    wr_en        Y
    full         Y
```

```
    Rows:      Hits  FEC Target          Non-masking condition(s)
 ---------  ---------  --------------------  --------------------------
  Row   1:          1  rd_en_0              (~full && wr_en)
  Row   2:          1  rd_en_1              -
  Row   3:          1  wr_en_0              ~rd_en
  Row   4:          1  wr_en_1              (~full && ~rd_en)
  Row   5:          1  full_0               (~rd_en && wr_en)
  Row   6:          1  full_1               (~rd_en && wr_en)

----------------Focused Condition View-------------------
Line       86 Item    1  ((rd_en && ~wr_en) && ~empty)
Condition totals: 3 of 3 input terms covered = 100.00%

  Input Term   Covered  Reason for no coverage   Hint
 -----------  --------  -----------------------  --------------
     rd_en        Y
     wr_en        Y
     empty        Y

    Rows:      Hits  FEC Target          Non-masking condition(s)
 ---------  ---------  --------------------  --------------------------
  Row   1:          1  rd_en_0              -
  Row   2:          1  rd_en_1              (~empty && ~wr_en)
  Row   3:          1  wr_en_0              (~empty && rd_en)
  Row   4:          1  wr_en_1              rd_en
  Row   5:          1  empty_0              (rd_en && ~wr_en)
  Row   6:          1  empty_1              (rd_en && ~wr_en)

----------------Focused Condition View-------------------
Line       88 Item    1  ((rd_en && wr_en) && empty)
Condition totals: 3 of 3 input terms covered = 100.00%

  Input Term   Covered  Reason for no coverage   Hint
 -----------  --------  -----------------------  --------------
     rd_en        Y
     wr_en        Y
     empty        Y

    Rows:      Hits  FEC Target          Non-masking condition(s)
 ---------  ---------  --------------------  --------------------------
  Row   1:          1  rd_en_0              -
  Row   2:          1  rd_en_1              (empty && wr_en)
  Row   3:          1  wr_en_0              rd_en
  Row   4:          1  wr_en_1              (empty && rd_en)
  Row   5:          1  empty_0              (rd_en && wr_en)
  Row   6:          1  empty_1              (rd_en && wr_en)

----------------Focused Condition View-------------------
Line       90 Item    1  ((rd_en && wr_en) && full)
Condition totals: 3 of 3 input terms covered = 100.00%

  Input Term   Covered  Reason for no coverage   Hint
 -----------  --------  -----------------------  --------------
     rd_en        Y
     wr_en        Y
      full        Y

    Rows:      Hits  FEC Target          Non-masking condition(s)
 ---------  ---------  --------------------  --------------------------
  Row   1:          1  rd_en_0              -
  Row   2:          1  rd_en_1              (full && wr_en)
  Row   3:          1  wr_en_0              rd_en
  Row   4:          1  wr_en_1              (full && rd_en)
  Row   5:          1  full_0               (rd_en && wr_en)
  Row   6:          1  full_1               (rd_en && wr_en)

----------------Focused Condition View-------------------
Line       96 Item    1  (count == 8)
Condition totals: 1 of 1 input term covered = 100.00%

    Input Term   Covered  Reason for no coverage   Hint
  -----------  --------  -----------------------  --------------
  (count == 8)        Y

    Rows:      Hits  FEC Target          Non-masking condition(s)
 ---------  ---------  --------------------  --------------------------
  Row   1:          1  (count == 8)_0        -
  Row   2:          1  (count == 8)_1        -

----------------Focused Condition View-------------------
Line       100 Item    1  (count == 0)
Condition totals: 1 of 1 input term covered = 100.00%

    Input Term   Covered  Reason for no coverage   Hint
  -----------  --------  -----------------------  --------------
  (count == 0)        Y
```

```
    Rows:      Hits  FEC Target           Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (count == 0)_0       -
  Row   2:          1  (count == 0)_1       -

----------------Focused Condition View-------------------
Line       104 Item    1  (count == (8 - 1))
Condition totals: 1 of 1 input term covered = 100.00%

        Input Term   Covered  Reason for no coverage   Hint
       -----------  --------  ----------------------  --------------
 (count == (8 - 1))        Y

    Rows:      Hits  FEC Target           Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (count == (8 - 1))_0  -
  Row   2:          1  (count == (8 - 1))_1  -

----------------Focused Condition View-------------------
Line       108 Item    1  (count == 1)
Condition totals: 1 of 1 input term covered = 100.00%

    Input Term   Covered  Reason for no coverage   Hint
   -----------  --------  ----------------------  --------------
 (count == 1)        Y

    Rows:      Hits  FEC Target           Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (count == 1)_0       -
  Row   2:          1  (count == 1)_1       -


Statement Coverage:
    Enabled Coverage            Bins      Hits     Misses  Coverage
    ----------------            ----      ----     ------  --------
    Statements                    39        39          0  100.00%

================================Statement Details================================

Statement Coverage for instance /\top#DUT_Design  --

    Line         Item                   Count     Source
    ----         ----                   -----     ------
 File FIFO_design_sv.sv
    8                                   module FIFO #(parameter FIFO_DEPTH = 8, FIFO_WIDTH = 16)(interface_FIFO.DUT_Design inst_interface);
    9
    10                                   logic [inst_interface.FIFO_WIDTH-1:0] data_in;
    11                                   logic clk, rst_n, wr_en, rd_en;
    12                                   logic [inst_interface.FIFO_WIDTH-1:0] data_out;
    13                                   logic wr_ack, overflow;
    14                                   logic full, empty, almostfull, almostempty, underflow;
    15
    16
    17                                   assign inst_interface.wr_ack     = wr_ack;
    18                                   assign inst_interface.overflow    = overflow;
    19                                   assign inst_interface.underflow   = underflow;
    20                                   assign inst_interface.full        = full;
    21                                   assign inst_interface.empty       = empty;
    22                                   assign inst_interface.almostfull  = almostfull;
    23                                   assign inst_interface.almostempty = almostempty;
    24                                   assign inst_interface.data_out    = data_out;
    25           1                  10003    assign clk= inst_interface.clk;
    26           1                  2037     assign rst_n=inst_interface.rst_n;
    27           1                  2120     assign wr_en=inst_interface.wr_en;
    28           1                  2116     assign rd_en=inst_interface.rd_en;
    29           1                  5001     assign data_in=inst_interface.data_in;
    30
    31                                   localparam max_fifo_addr = $clog2(FIFO_DEPTH);
    32
    33                                   reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];
    34
    35                                   reg [max_fifo_addr:0] count ;
    36                                   reg [max_fifo_addr-1:0] wr_ptr,rd_ptr ;
    37
    38           1                  6019     always @(posedge clk or negedge rst_n) begin
    39                                                if (!rst_n) begin
    40           1                  2475                              wr_ptr <= 0;
    41           1                  2475                              overflow <=0;
    42                                                end
    43                                                else if (wr_en ) begin
    44                                                                  if (full==0)begin
    45           1                  2431                                mem[wr_ptr] <= data_in;
    46           1                  2431                                wr_ack <= 1;
    47           1                  2431                                wr_ptr <= wr_ptr + 1;
    48           1                  2431                                overflow <=0;
    49                                                                  end
    50                                                                  else begin
    51           1                   12                                 wr_ack  <= 0;
```

```
52      1      12                          overflow<= 1;
53                                   end
54                         end
55                         else begin
56      1      1101                  overflow  <= 0;
57      1      1101                  wr_ack    <= 0;
58                         end
59                 end
60
61      1      5407   always @(posedge clk or negedge rst_n) begin
62                         if (!rst_n) begin
63      1      2225                  rd_ptr <= 0;
64      1      2225                  underflow <=0;
65                         end
66                         else if (rd_en ) begin
67                                   if (empty==0)begin
68      1      658                             data_out <= mem[rd_ptr];
69      1      658                     rd_ptr <= rd_ptr + 1;
70      1      658                    underflow <=0;
71                                   end
72                                   else
73      1      420                             underflow <=1;
74                         end
75                         else
76      1      2104                  underflow <=0;
77                 end
78
79      1      5665   always @(posedge clk or negedge rst_n) begin
80                         if (!rst_n) begin
81      1      2327                  count <= 0;
82                         end
83                         else begin
84                                   if            ( ({wr_en, rd_en} == 2'b10) && !full)
85      1      1684                            count <= count + 1;
86                                   else if ( ({wr_en, rd_en} == 2'b01) && !empty)
87      1      204                             count <= count - 1;
88                                   else if (({wr_en, rd_en} == 2'b11) && empty)
89      1      299                             count <= count + 1;
90                                   else if (({wr_en, rd_en} == 2'b11) && full)
91      1      6                               count <= count - 1;
92                         end
93                 end
94
95      1      3074   always@(count)begin
96                         if (count==FIFO_DEPTH )
97      1      14                    full=1;
98                         else
99      1      3060                  full=0;
100                        if (count==0)
101     1      983                   empty=1;
102                        else
103     1      2091                  empty=0;
104                        if (count==(FIFO_DEPTH-1))
105     1      26                    almostfull=1;
106                        else
107     1      3048                  almostfull=0;
108                        if (count == 1)
109     1      1035                  almostempty=1;
110                        else
111     1      2039                  almostempty=0;
```

Toggle Coverage:

```
    Enabled Coverage          Bins     Hits   Misses  Coverage
    ----------------          ----     ----   ------  --------
    Toggles                    106      106        0   100.00%
```

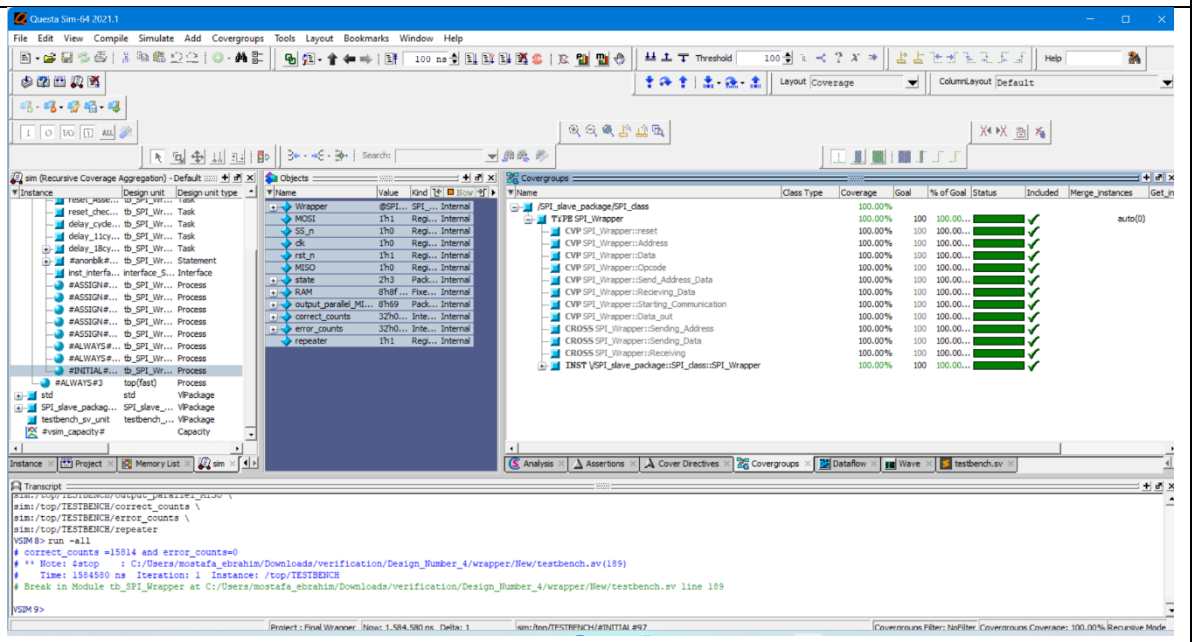
```
================================Toggle Details================================

Toggle Coverage for instance /\top#DUT_Design  --

                        Node        1H->0L    0L->1H  "Coverage"
                        --------------------------------------
                 almostempty         1         1      100.00
                  almostfull         1         1      100.00
                         clk         1         1      100.00
                  count[3-0]         1         1      100.00
              data_in[15-0]          1         1      100.00
             data_out[15-0]          1         1      100.00
                       empty         1         1      100.00
                        full         1         1      100.00
                    overflow         1         1      100.00
                       rd_en         1         1      100.00
                 rd_ptr[2-0]         1         1      100.00
                       rst_n         1         1      100.00
                   underflow         1         1      100.00
                      wr_ack         1         1      100.00
                       wr_en         1         1      100.00
                 wr_ptr[2-0]         1         1      100.00
```

Total Node Count = 53
Toggled Node Count = 53
Untoggled Node Count = 0

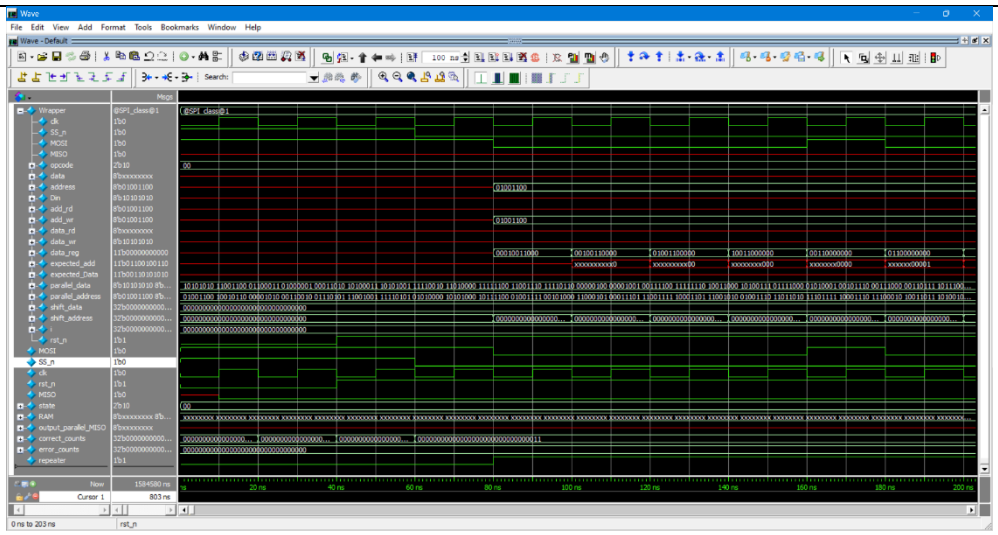
Toggle Coverage = 100.00% (106 of 106 bins)

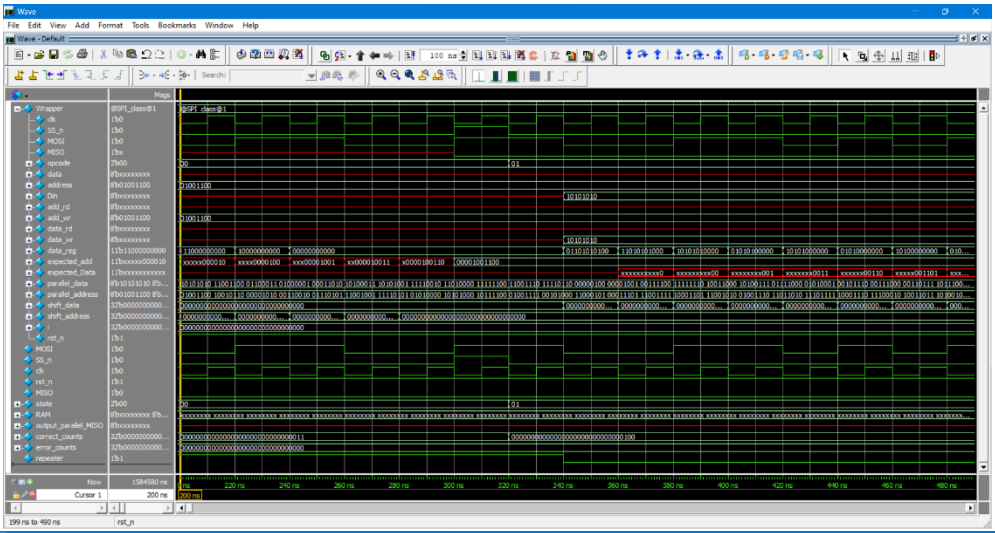
Total Coverage By Instance (filtered view): 100.00%

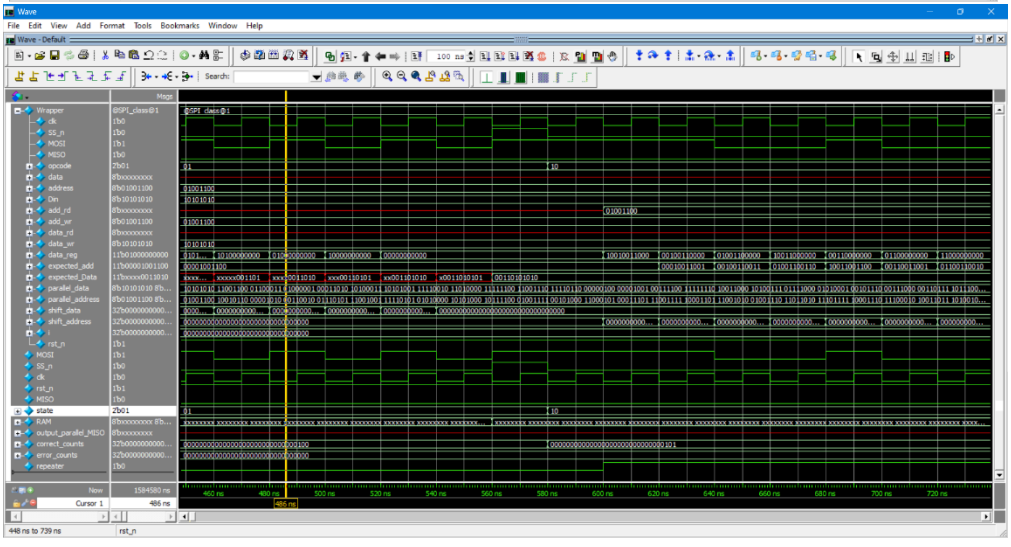
Functional coverage

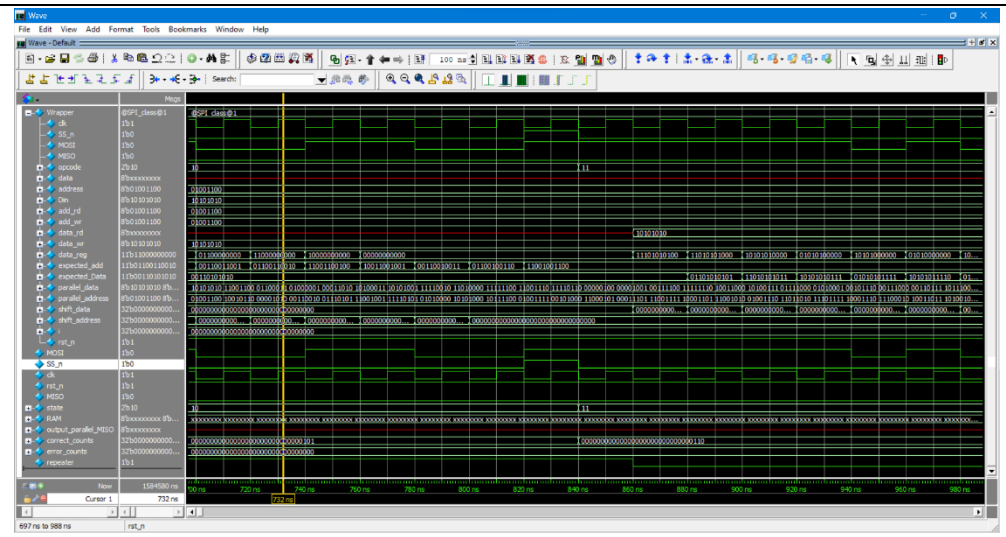


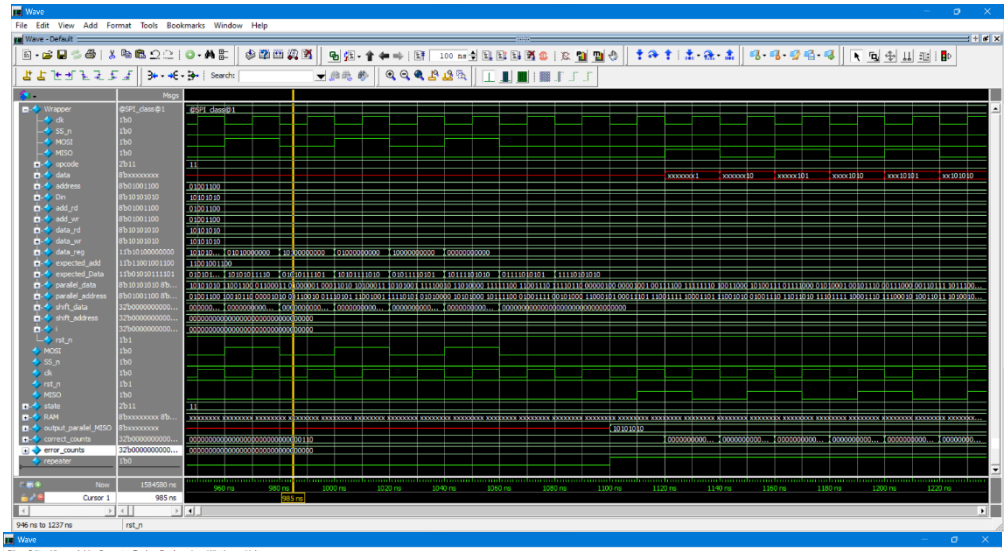
Questa Snippets	

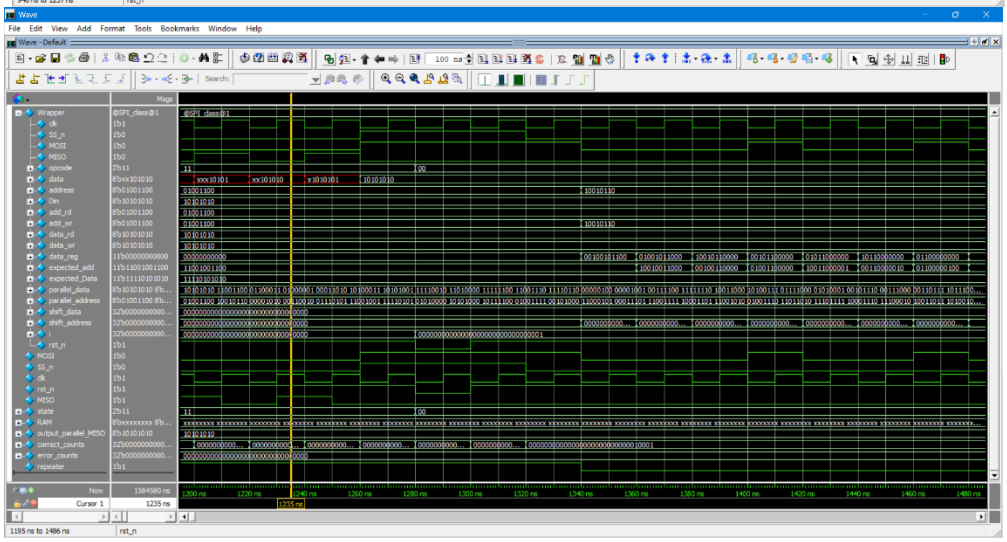












Verification req document

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
Start Communication	When the master want to start communication , SS_n equal Zero, and current state in spi was IDLE and waite for SS_n=0 for going to CHK_CMD for tring for Wrtiting or Reading	Randomized MISO must done by changing parallel data to serial data and wait in SPI slave a lot of clock cycles to get outputs , So Randomization under a complex constrain on MISO and rst_n.	Include in coverpoints for Starting communication and ending communication or transaction from State (CHK_CMD to IDLE).	There is no method for checking the internal proccess in SPI_Slave , but could Monitor the MISO and transfer it to parrallel data to comparing with Golden Model.
End Communication	this case happened when SS_n = 1 , this end communication and spi slave went to IDLE state.	communication End for Starting another proccess in SPI slave or being in waiting mode for spi , so randomization will be depend on transaction of SS_n from asserted to denyied .	include in cover points bins for starting and ending and save free transation under FSM in spi slave , bins that will be avilable for this state : bins Start = {0}; bins End = {1}; bins transaction_back = (0 => 1 => 0) ;	Output Checked against golden model
Writing_Address_Writing_operation	In this case , the communication started by sending SS_n =0 then another SS_n=0 and MOSI =1'b1 for the next posedge clk for going to Writing state inFSM and must the other cycles receive first 2 bits 2'b00 for writing addr first and continue to converted it and sending the value to RAM with Enable to activated Reading it from SPI slave	Randomization occure according sending all possible addresses First.	Included in a coverpoint for addresses of RAM : coverpoint address ; bins writing_add = {2'b00};	checking her for functionality done by test the spi alone and acomparing the output signal my Golden model to check for ideal act with RAM.
Writing_Data_for_wr_operation	this case is similar to adding addr to SPI to but changing the 2most bits to 2'b01.	Randomization occur depending on testing a lot of avilable data in for converting it to serial to match the spesific design	Included in a coverpoint for Data : bins writing_data = {2'b01};	also Writing operation Done by comparing this output with Golden model for SPI slave
Writing_Addr_rd_operation	this case is critical for SPI , because SPI must start communication then go to CHK_CMD then READ_ADDR then the first 2 bits must equal 2'b10	Randomizied the the data in in writing address for read operation and check rst and starting communication and going to read address.	included in coverpoint for address for Reading , bins reading_add = {2'b10}; bins writing_add = {2'b00}; bins writing_data = {2'b01};	also checking by comparing the output of SPI individuali before instantiate it
Reading_Data_rd_operation	Also this case when current state equal CHK_CMD and waiting to have the most 2 bit equal 2'b11 and demi 8 bits for sending	ss_n must be high until I end reading and rst_n didn't happened and wait for Rading completly.	included cover points and cross coverage to check exact values for bins as : bins repeat_reading = (2'b11 => 2'b10); bins reading_complete = (2'b10 => 2'b11); coverpoint data ; Sending_Data : cross Send_Address_Data , Data ;	Here it will be easier for comparing output one bit with one getting from Golden model to check validity
Invalid_cases	This state is Dangerious , 1) when the SPI didn't take the exection time so the output of SPI slave will not be valid but here ther is an signal outed from SPI to call the RAM this value is valid or not . 2) 2th case when u need to read data from RAM without getting Address first , also this case will handle by internal flag to call how will be in this time firstly and loock to do another instruction. but if the most 2 bit are not correct , the data will not valid for	randomizing inputs and looking forward bad sequence that didn't meet specification.	checking in this state coverage of input address and and input data and also serial data	when invalid case happened it must seen from erroring in wave form (directed) our display message tell that there is an errror for wronge usage for spi or RAM.
Reset	in this case , SPI slave still in IDLE state and output of RAM equal Zero and enable for this is Zero and MISO also equal Zero	Randomization the resert to Getting once in along period in Design system	Included in a coverpoint for Data : bins reset_asserted ={0}; bins reset_disable ={1};	this state can be checked by comparing for Golden model and Get Correct count or Error count