SPI Slave Design

Design bugs

- SPI bugs:

```
12      reg [2:0] cs, ns;
13      reg [9:0] PO;
14      wire [7:0] temp;
15      reg SO, flag_rd = 0;
16      integer state_count = 0, final_count = 0;
17
18      assign temp = (tx_valid)? tx_data: temp;
19      assign MISO = SO;
20
21      always @(posedge clk or negedge rst_n) begin
22          if (!rst_n)
23              cs <= IDLE;
24          else
25              cs <= ns;
26      end
```

1) rst_n must be Async.
2) When reset is asserted the flags and counters must flush and data_out signals must also zeros.

Handling:

```
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
37
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
```

- **Current state bugs:**

```
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
```

Handle:

```
52    always @(posedge clk) begin//* bug
53
54        case (cs)
55              IDLE : begin
56
57                  rx_valid = 0 ;
58                  PO  = 0 ;
59                  state_count = 0 ;
60                  final_count = 0 ;
61                  SO = 0 ;
62                  Act_input_output = 0;
63                      end
64              WRITE: begin //done
65
66                  if (state_count < 10 )begin //0 ,
67                      PO = {PO[8:0] , MOSI} ;
68                      state_count = state_count + 1 ;
69                      if (PO[9] ==1'b0 && state_count ==10)begin
70                          rx_valid = 1 ;
71                      end
72                      else
73                          rx_valid = 0 ;
74                  end
75                  else
76                      rx_valid = 0 ;
77
78                      end
```

```
79              READ_ADD: begin // done
80
81                if (state_count<10 )begin
82                    PO = {PO[8:0] , MOSI} ;
83                    state_count = state_count + 1 ;
84                    rx_valid = 0 ;
85                end
86                if (PO[9:8]==2'b10 && state_count==10)begin
87                    rx_valid = 1 ;
88                    flag_rd = 1;
89                end
90                else
91                    rx_valid = 0 ;
92                        end
93              READ_DATA: begin // done
94
95                if (Act_input_output == 0)begin
96                    if (state_count<9 )begin
97                        PO = {PO[8:0] , MOSI} ;
98                        state_count = state_count + 1 ;
99                        rx_valid = 0 ;
100                   end
101                   else begin
102                       PO = {PO[8:0] , MOSI} ;
103                       state_count = state_count + 1 ;
104                       rx_valid = 0 ;
105                   end
106
107                   if (PO[9:8]==2'b11 && state_count== 10 )begin
108                       rx_valid = 1 ;
109                       Act_input_output = 1;
110                       flag_rd = 0;
111                   end
112               end
113               else begin
114                   state_count = state_count + 1 ;
115                   rx_valid = 0 ;
116                   if (tx_valid && state_count == 12 )begin
117                       temp = tx_data ;
118                   end
119                   if (state_count >= 12 && final_count <= 7)begin
120                       SO = temp [7 - final_count ];
121                       rx_valid =0 ;
122                       final_count = final_count + 1 ;
123                   end
124               end
125                       end
126             default: begin
127               rx_valid = 0 ;
128               PO  = 0 ;
129               state_count = 0 ;
130               final_count = 0 ;
131               SO = 0 ;
132               Act_input_output = 0;
133
134                  end
135             endcase
136     end
```

- **Bugs flag of read operation:**

```
53      always @(MOSI, SS_n, cs) begin
54          case (cs)
55              IDLE:
56              if (SS_n)
57                  ns = IDLE;
58              else
59                  ns = CHK_CMD;
60              CHK_CMD:
61              if (SS_n)
62                  ns = IDLE;
63              else begin
64                  if (!MOSI)
65                  ns = WRITE;
66                  else begin
67                  if (!flag_rd)
68                      ns = READ_ADD;
69                  else
70                      ns = READ_DATA;
71                  end
72              end
73              WRITE:
74              if (SS_n)
75                  ns = IDLE;
76              else
77                  ns = WRITE;
78              READ_ADD:
79              if (SS_n)
80                  ns = IDLE;
81              else begin
82                  ns = READ_ADD; flag_rd = 1;
83              end
84              READ_DATA:
85              if (SS_n)
86                  ns = IDLE;
87              else begin
88                  ns = READ_DATA; flag_rd = 0;
89              end
90                default: ns = IDLE;
91          endcase
92      end
93
```

Handle:

The value of the flag changed in current state (reading data or address) block not here.

```
138     always @(*) begin
139
140         case (cs)
141             IDLE:
142                 if (SS_n)
143                     ns = IDLE;
144                 else
145                     ns = CHK_CMD;
146             CHK_CMD:
147                 if (SS_n)
148                     ns = IDLE;
149                 else begin//SS_n = 0
150                     if (MOSI==0) //MOSI = 0
151                         ns = WRITE;
152                     else  begin //MOSI = 1
153                         if (!flag_rd) //flag_rd = 0
154                             ns = READ_ADD;
155                         else  //flag_rd = 1
156                             ns = READ_DATA;
157                     end
158                 end
159             WRITE:
160                 if (SS_n)
161                     ns = IDLE;
162                 else
163                     ns = WRITE;
164             READ_ADD:
165                 if (SS_n)
166                     ns = IDLE;
167                 else begin
168                     ns = READ_ADD;
169                     // flag_rd = 1;
170                 end
171             READ_DATA:
172                 if (SS_n)
173                     ns = IDLE;
174                 else begin
175                     ns = READ_DATA;
176                     // flag_rd = 0;
177                 end
178
179             default: ns = IDLE;
180         endcase
181     end
182
```

Design code

```
module SPI_Slave #(IDLE=0 ,CHK_CMD = 1,WRITE = 2,READ_ADD = 3,READ_DATA =
4)(interface_SPI.DUT_Design inst_interface );

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

    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
    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 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
```

Golden model code

```
module SPI_Golden #(IDLE=0 ,CHK_CMD = 1,WRITE = 2,READ_ADD = 3,READ_DATA =
4)(interface_SPI.GOLDEN_REF inst_interface );

logic clk, rst_n, SS_n, MOSI, tx_valid;
logic [7:0] tx_data , temp;

logic MISO_expected , rx_valid_expected;
logic [9:0] rx_data_expected;

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_expected=MISO_expected;
     inst_interface.rx_valid_expected =rx_valid_expected;
     inst_interface.rx_data_expected  =rx_data_expected;
end
(* fsm_encoding="sequential" *)
reg [2:0] cs, ns;// current state

reg address_recieved;
integer counter_sp;
integer counter_ps;
always@(cs)begin
    if (cs == IDLE)
        rx_data_expected <= 0;
        rx_valid_expected <= 0;
        MISO_expected  <= 0;
        counter_sp <= 0;
        counter_ps <= 8;
end

//State memory
always @(posedge clk) begin
    if (~rst_n) begin
        // reset
        cs <= IDLE;
    end
    else begin
        cs <= ns;
    end
end

//Next state logic
always @(*) begin
    case(cs)
        IDLE:
            if (SS_n) begin
                ns = IDLE;
            end
```

```
            else begin
                ns = CHK_CMD;
            end
        READ_ADD:
            if (SS_n) begin
                ns = IDLE;
            end
            else begin
                ns = READ_ADD;
                // address_recieved = 1;// i received the address
            end
        CHK_CMD:
            if (SS_n) begin
                ns = IDLE;
            end
            else if (MOSI) begin
                if (address_recieved == 1) begin
                    ns = READ_DATA;
                end
                else begin
                    ns = READ_ADD;
                end
            end
            else begin
                ns = WRITE;
            end
        WRITE:
            if (SS_n) begin
                ns = IDLE;
            end
            else begin
                ns = WRITE;
            end
        READ_DATA:
            if (SS_n) begin
                ns = IDLE;
            end
            else begin
                ns = READ_DATA;
                // address_recieved = 0;
            end
        default: ns = IDLE;
    endcase
end
```

```
//Output logic
always @(posedge clk) begin
    if (~rst_n) begin
        // reset
        rx_data_expected <= 0;
        rx_valid_expected <= 0;
        MISO_expected  <= 0;
        counter_sp <= 0;
        counter_ps <= 8;
        address_recieved <= 0 ;
        temp <= 0 ;
    end
    else begin
        case(cs)
            WRITE: begin
                if (counter_sp<10) begin
                    rx_data_expected = {rx_data_expected,MOSI};//shifting
                    counter_sp = counter_sp + 1;
                    rx_valid_expected = 0;
                end
                if (counter_sp==10 && rx_data_expected[9]==1'b0) begin
                    rx_valid_expected <= 1;

                end
            end
            READ_ADD: begin
                if (counter_sp<10) begin
                    rx_data_expected = {rx_data_expected,MOSI};
                    counter_sp = counter_sp + 1;
                end
                if (counter_sp==10 && rx_data_expected[9:8]==2'b10) begin
                    rx_valid_expected = 1;
                    address_recieved = 1;
                end
            end
            READ_DATA: begin

                if (counter_sp <11 )begin
                    if (counter_sp<10) begin
                        rx_data_expected = {rx_data_expected,MOSI};
                        counter_sp = counter_sp + 1;
                        rx_valid_expected = 0 ;
                    end
                    if (counter_sp == 10 && rx_data_expected[9:8]==2'b11)begin
                        rx_valid_expected = 1 ;
```

	counter_sp = counter_sp + 1;
	address_recieved = 0;
	end
	end
	else begin
	counter_sp = counter_sp + 1;
	rx_valid_expected = 0 ;
	if (tx_valid && counter_sp == 13)begin
	temp = tx_data ;
	end
	if (counter_sp >= 13 && counter_ps > 0)begin
	MISO_expected = temp [counter_ps - 1];
	rx_valid_expected =0 ;
	counter_ps = counter_ps - 1;
	end
	end
	end
	endcase
	end
	end
	endmodule

Top code	module top();
	bit clk;
	always #10 clk=~clk;
	interface_SPI inst_interface(clk);
	SPI_Slave DUT_Design (inst_interface);
	// SPI_Wrapper DUT_Wrapper (inst_interface);
	SPI_Golden GOLDEN_REF (inst_interface);
	tb_SPI_slave TESTBENCH (inst_interface);
	endmodule

Interface code	interface interface_SPI (clk);
	
	input bit clk ;
	parameter IDLE = 0;
	parameter CHK_CMD = 1;
	parameter WRITE = 2;
	parameter READ_ADD = 3;
	parameter READ_DATA = 4;
	
	logic MOSI, SS_n, rst_n, tx_valid;
	logic [7:0] tx_data;
	logic MISO,MISO_expected;
	logic rx_valid , rx_valid_expected;

	logic [9:0] rx_data , rx_data_expected; modport DUT_Design (input clk, rst_n, SS_n, MOSI, tx_valid ,tx_data, output MISO, rx_valid , rx_data); modport TESTBENCH (input clk, MISO , rx_valid , rx_data, MISO_expected , rx_valid_expected , rx_data_expected, output rst_n, SS_n, MOSI, tx_valid ,tx_data); modport GOLDEN_REF(input clk, rst_n, SS_n, MOSI, tx_valid ,tx_data, output MISO_expected , rx_valid_expected , rx_data_expected); endinterface

Packages code	package SPI_slave_package;
	class SPI_class ;
	logic clk;
	logic SS_n;
	logic MOSI ;
	logic MISO;
	logic rx_valid;
	logic [9:0] rx_data;
	logic [1:0] opcode ;
	logic [7:0] tx_data , data ,address , Din ;
	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;
	logic tx_valid;
	
	function void Set_tx_valid(logic select);
	tx_valid =select;
	tx_data =parallel_data[i] ;
	endfunction

```
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];
            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] ;
            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 opcode_1(logic[1:0] state);
 opcode =state;
    if (rst_n==0)//00->11->01->10
        opcode=2'b00;
    else if (opcode==2'b00)
        opcode=2'b11;
    else if (opcode==2'b11)
        opcode=2'b01;
    else if (opcode==2'b01)
        opcode=2'b10;
    else if (opcode==2'b10)
        opcode=2'b00;
    else
        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_Slave_cover@(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_1 = (2'b11 => 2'b01);
        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 writing_add  = {2'b00};
        bins writing_data = {2'b01};
        bins reading_complete = (2'b10 => 2'b11);
        bins reading_add  = {2'b10};
        bins reading_data = {2'b11};
        }
    Valid_sending: coverpoint rx_valid{
        bins sending_asserted ={1};
        bins sending_disable  ={0};
        }
    Valid_receiving : coverpoint tx_valid{
        bins receiving_asserted ={1};
        bins receiving_disable  ={0};
        }
    Data_out : coverpoint data ;
    Valid_sending_add_Cross : cross Address ,Valid_sending ;
    Valid_sending_data_Cross : cross Data ,Valid_sending ;
    Receiving_Cross : cross Opcode , Valid_sending {
        ignore_bins ignore_1 = binsof (Valid_sending.sending_asserted) &&
binsof (Opcode.reading_data);
        ignore_bins ignore_2 = binsof (Valid_sending.sending_asserted) &&
binsof (Opcode.invalid_0);
        ignore_bins ignore_3 = binsof (Valid_sending.sending_asserted) &&
binsof (Opcode.invalid_1);
```

```
        ignore_bins ignore_4 = binsof (Valid_sending.sending_asserted) &&
binsof (Opcode.invalid_2);
        ignore_bins ignore_5 = binsof (Valid_sending.sending_asserted) &&
binsof (Opcode.invalid_3);
        ignore_bins ignore_6 = binsof (Valid_sending.sending_asserted) &&
binsof (Opcode.reading_complete);
        ignore_bins ignore_7 = binsof (Valid_sending.sending_asserted) &&
binsof (Opcode.writing_complete);
        ignore_bins ignore_8 = binsof (Valid_sending.sending_asserted) &&
binsof (Opcode.repeat_writing);
        ignore_bins ignore_9 = binsof (Valid_sending.sending_asserted) &&
binsof (Opcode.repeat_reading);
    }
    sending_Cross : cross Opcode , Valid_receiving ;
endgroup
SPI_Slave_cover =new();
    endclass
endpackage
```

Assertion Code

```
module RAM_Assertion_sva (interface_RAM.ASSERTION inst_interface );
    bit  clk;
    logic rst_n     ;
    logic [9:0] din ;
    logic rx_valid  ;
    logic [7:0] dout, dout_expect;
    logic tx_valid ,tx_valid_expect ;
    logic [1:0] opcode;

    assign clk      = inst_interface.clk;
    assign rst_n    = inst_interface.rst_n;
    assign rx_valid = inst_interface.rx_valid;
    assign din      = inst_interface.din;
    assign dout_expect     = inst_interface.dout_expect;
    assign tx_valid_expect = inst_interface.tx_valid_expect ;
    assign dout     = inst_interface.dout;
    assign tx_valid = inst_interface.tx_valid ;
    assign opcode   = din[9:8];


    property DATA_OUT;
    @(posedge clk) disable iff (~rst_n )
     (opcode == 2'b00)  |=> (opcode == 2'b01) |=> (opcode == 2'b10) |=> (opcode ==
2'b11) |=> (dout == dout_expect) |-> (tx_valid == tx_valid_expect);
    endproperty
```

```
 property RESET;
@(posedge clk)
 (rst_n==0) |=> (dout==0) |->(tx_valid==0);
endproperty

 property ENABLE ;
@(posedge clk) disable iff (~rst_n)
 (rx_valid==0) |=> ($past(dout)==dout) ;
endproperty

    DATA_OUT_Assertion: assert property (DATA_OUT) else $display("DATA_OUT fail");
    RESET_Assertion   : assert property (RESET)    else $display("RESET fail   ");
    ENABLE_Assertion  : assert property (ENABLE)   else $display("ENABLE fail  ");

    DATA_OUT_Cover: cover property (DATA_OUT) $display("DATA_OUT pass");
    RESET_Cover   : cover property (RESET)    $display("RESET pass   ");
    ENABLE_Cover  : cover property (ENABLE)   $display("ENABLE pass  ");
endmodule
```

Testbench code

```
import SPI_slave_package::*;
module tb_SPI_slave (interface_SPI.TESTBENCH inst_interface );
    SPI_class SPI_ports =new();
    logic MOSI, SS_n, clk, rst_n, tx_valid;
    logic [7:0] tx_data ;
    logic MISO , MISO_expected ;
    logic rx_valid, rx_valid_expected;
    logic [9:0] rx_data,rx_data_expected,data;
    logic [1:0] state;

    assign clk =inst_interface.clk;
    assign MISO=inst_interface.MISO;
    assign MISO_expected=inst_interface.MISO_expected;
    assign rx_valid=inst_interface.rx_valid;
    assign rx_valid_expected=inst_interface.rx_valid_expected;
    assign rx_data_expected=inst_interface.rx_data_expected;
    assign rx_data=inst_interface.rx_data;

    assign inst_interface.MOSI=MOSI;
    assign inst_interface.SS_n=SS_n;
    assign inst_interface.tx_valid=tx_valid;
    assign inst_interface.tx_data=tx_data;
    assign inst_interface.rst_n=rst_n;

    integer correct_counts=0;
    integer error_counts  =0;
```

```
    logic repeater;
    // SPI_Slave DUT (.*);
    // SPI_Golden  DUT_Golden (.*);

    always@(clk)begin
        SPI_ports.clk=clk;
    end

task instantiate();
    SS_n    = SPI_ports.SS_n    ;
    tx_data = SPI_ports.tx_data ;
    tx_valid= SPI_ports.tx_valid;
    rst_n   = SPI_ports.rst_n   ;
    SPI_ports.ParallelToSerial();
    MOSI    =SPI_ports.MOSI;

endtask

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

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

                end
            else begin
                error_counts++;
```

```
                $display("%0t ns , there is an error they must be zeros :
rx_valid = %0d and rx_valid_expected = %0d
",$time(),rx_valid,rx_valid_expected );
            end
        end
        else begin
            error_counts++;
            $display("%0t ns , there is an error they must be zeros :
rx_data = %0d and rx_data_expected = %0d
",$time(),rx_data,rx_data_expected );
        end
    end
endtask
task delay_cycle();
    @(negedge clk) repeater++ ;
endtask

task delay_10cycles();
    repeat(11)begin
        instantiate();
        @(negedge clk) begin
            SPI_ports.Set_value();
        end
    end

    if (state[0]==0)
        data =SPI_ports.expected_add[9:0];
    else
        data=SPI_ports.expected_Data[9:0];

    if (rx_data==data && rx_data_expected==data)begin
        if (rx_valid==rx_valid_expected)
            if (MISO== MISO_expected)begin
                    correct_counts++;
                end
                else begin
                    error_counts++;
                    $display("%0t ns , there is an error : MISO = %0d and
MISO_expected = %0d ",$time(),MISO,MISO_expected );

                end
        else begin
            error_counts++;
            $display("%0t ns , there is an error : rx_valid = %0d and
rx_valid_expected = %0d ",$time(),rx_valid,rx_valid_expected );
```

```
        end
    end
    else begin
        error_counts++;
        $display("%0t ns , there is an error : rx_data = %0d and
rx_data_expected = %0d ",$time(),rx_data,rx_data_expected );
        $display("Exact_data =%0d ",data);
    end

endtask
task delay_18cycles();

    delay_10cycles() ;
    if (rx_valid == 1 )begin
            delay_cycle();
        for(int i = 7 ; i >= 0 ; i--) begin
        @(negedge clk)begin
            if ( tx_valid == 1)begin
                if (MISO == SPI_ports.tx_data[i] && MISO_expected ==
SPI_ports.tx_data[i])begin
                    correct_counts++;
                    SPI_ports.MISO =MISO;
                    SPI_ports.Data_out();
                end
                else begin
                    error_counts++;
                    $display("%0t ns , there is an error : MISO = %0d and
MISO_expected = %0d ",$time(),MISO,MISO_expected );
                    $display("Exact_MISO =%0d ",SPI_ports.tx_data[i]);
                end
            end
            else begin
                if (MISO == MISO_expected )begin
                    correct_counts++;
                    SPI_ports.MISO =MISO;
                    SPI_ports.Data_out();
                end
                else begin
                    error_counts++;
                    $display("%0t ns , there is an error : MISO = %0d and
MISO_expected = %0d ",$time(),MISO,MISO_expected );
                end
            end
        end
    end
```

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

endtask

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

initial begin
    SPI_ports.Get_information();
    #1 SPI_ports.Set_tx_valid (0);
    reset_Asserted ();//task for reset
    for (int i = 0; i < 1600 ; i++) begin

        SPI_ports.Set_tx_valid (1);
        reset_Asserted ();//task for reset
        SPI_ports.select_SS(1);
        instantiate();
        reset_check();
        for (int j = 0; j < 4; j++) begin
            SPI_ports.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_10cycles();
            end
            SPI_ports.MISO =MISO;
            SPI_ports.rx_data=rx_data;
            SPI_ports.rx_valid=rx_valid;
            SPI_ports.select_SS(1);//end communication
            instantiate();
            reset_check();
            SPI_ports.rx_data=rx_data;
            SPI_ports.rx_valid=rx_valid;
            if (i < 255)
                SPI_ports.opcode_0(state);
            else if (i==510 || i < 765 )begin
                // reset_Asserted ();
                SPI_ports.opcode_1(state);
            end
            else if (i==765 || i < 1020) begin
                // reset_Asserted ();
                SPI_ports.writing(state);
            end
            else if (i==1020 || i < 1200)begin
                // reset_Asserted ();
                SPI_ports.Reading(state);
            end
            else if (i==1200 || i < 1500) begin
                // reset_Asserted ();
                SPI_ports.opcode_0(state);
            end
            SPI_ports.SPI_Slave_cover.sample();
        end
        SPI_ports.Increment_array();
    end

    for (int i = 0; i < 1500 ; i++) begin

        SPI_ports.Set_tx_valid (0);
        reset_Asserted ();//task for reset
        SPI_ports.select_SS(1);
        instantiate();
        reset_check();
        for (int j = 0; j < 4; j++) begin
            SPI_ports.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_10cycles();
            end
            SPI_ports.MISO =MISO;
            SPI_ports.rx_data=rx_data;
            SPI_ports.rx_valid=rx_valid;
            SPI_ports.select_SS(1);//end communication
            instantiate();
            reset_check();
            SPI_ports.rx_data=rx_data;
            SPI_ports.rx_valid=rx_valid;
            if (i < 255)
                SPI_ports.opcode_0(state);
            else if (i==510 || i < 765 )begin
                // reset_Asserted ();
                SPI_ports.opcode_1(state);
            end
            else if (i==765 || i < 1020) begin
                // reset_Asserted ();
                SPI_ports.writing(state);
            end
            else if (i==1020 || i < 1200)begin
                // reset_Asserted ();
                SPI_ports.Reading(state);
            end
            else if (i==1200 || i < 1500) begin
                // reset_Asserted ();
                SPI_ports.opcode_0(state);
            end
        end
        SPI_ports.Increment_array();
        if (i==500 || i==700 || i==1000 ||i==1300)begin
            reset_Asserted ();
        end
    end
    SPI_ports.select_SS(1);//end communication
    instantiate();
    delay_cycle();
    SPI_ports.select_SS(0);//end communication
    instantiate();
```

```
    delay_cycle();
    SPI_ports.select_SS(1);//end communication
    instantiate();
    delay_cycle();
    SPI_ports.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_Golden.sv top.sv interface.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.SPI_Slave
================================================================================
Branch Coverage:
  Enabled Coverage          Bins     Hits    Misses  Coverage
  ----------------          ----     ----    ------  --------
  Branches                    47       47         0   100.00%

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

Branch Coverage for instance /\top#DUT_Design

  Line       Item                  Count     Source
  ----       ----                  -----     ------
 File SPI_SLAVE.sv
------------------------------------IF Branch------------------------------------
   32                     181659    Count coming in to IF
   32         1             3105                if (!rst_n)begin
   39         1           178554                else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
   43                      37204    Count coming in to IF
   43         1            12402                if (cs == IDLE)begin
                           24802    All False Count
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------CASE Branch----------------------------------
   55                     181151    Count coming in to CASE
   56         1            18607                IDLE : begin
   65         1            77660                WRITE: begin //done
   80         1            43082                READ_ADD: begin // done
   94         1            29400                READ_DATA: begin // done
   127        1            12402                default: begin
Branch totals: 5 hits of 5 branches = 100.00%

------------------------------------IF Branch------------------------------------
   67                      77660    Count coming in to IF
   67         1            70600                if (state_count < 10 )begin //0 ,
   76         1             7060                else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
   70                      70600    Count coming in to IF
   70         1             7060                if (PO[9] ==1'b0 && state_count ==10)begin
   73         1            63540                  else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
   82                      43082    Count coming in to IF
```

```
  82          1              38700                                  if (state_count<10 )begin
                             4382    All False Count
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  87                         43082    Count coming in to IF
  87          1              5700                                   if (PO[9:8]==2'b10 && state_count==10)begin
  91          1              37382                                  else
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  96                         29400    Count coming in to IF
  96          1              14700                                  if (Act_input_output == 0)begin
  114         1              14700                                              else begin
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  97                         14700    Count coming in to IF
  97          1              13230                                              if (state_count<9 )begin
  102         1              1470                                   else begin
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  108                        14700    Count coming in to IF
  108         1              1470                                   if (PO[9:8]==2'b11 && state_count== 10 )begin
                             13230    All False Count
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  117                        14700    Count coming in to IF
  117         1              735                                                if (tx_valid && state_count == 12 )begin
                             13965    All False Count
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  120                        14700    Count coming in to IF
  120         1              11760                                                          if (state_count >= 12 && final_count <= 7)begin
                             2940    All False Count
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------CASE Branch----------------------------------------
  141                        130332    Count coming in to CASE
  142         1              26184                                  IDLE:
  147         1              19463                                  CHK_CMD:
  160         1              45464                                  WRITE:
  165         1              28915                                  READ_ADD:
  172         1              10305                                  READ_DATA:
  180         1              1                          default: ns = IDLE;
Branch totals: 6 hits of 6 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  143                        26184    Count coming in to IF
  143         1              13782                                              if (SS_n)
  145         1              12402                                              else
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  148                        19463    Count coming in to IF
  148         1              1                                      if (SS_n)
  150         1              19462                                              else begin//SS_n = 0
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  151                        19462    Count coming in to IF
  151         1              7062                                     if (MOSI==0) //MOSI = 0
  153         1              12400                                               else  begin //MOSI = 1
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  154                        12400    Count coming in to IF
  154         1              10930                                                            if (!flag_rd) //flag_rd = 0
  156         1              1470                                                 else  //flag_rd = 1
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  161                        45464    Count coming in to IF
  161         1              7060                                   if (SS_n)
  163         1              38404                                              else
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  166                        28915    Count coming in to IF
  166         1              3870                                   if (SS_n)
  168         1              25045                                              else begin
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------------IF Branch----------------------------------------
  173                        10305    Count coming in to IF
  173         1              1470                                   if (SS_n)
  175         1              8835                                   else begin
Branch totals: 2 hits of 2 branches = 100.00%
```

Condition Coverage:

Enabled Coverage	Bins	Covered	Misses	Coverage
Conditions	14	11	3	78.57%

```
================================Condition Details================================
```

```
Condition Coverage for instance /\top#DUT_Design  --

  File SPI_SLAVE.sv
----------------Focused Condition View-------------------
Line       43 Item    1  (cs == 0)
Condition totals: 1 of 1 input term covered = 100.00%

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

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

----------------Focused Condition View-------------------
Line       67 Item    1  (state_count < 10)
Condition totals: 1 of 1 input term covered = 100.00%

         Input Term   Covered  Reason for no coverage   Hint
        -----------  --------  ----------------------  --------------
  (state_count < 10)         Y

    Rows:       Hits  FEC Target           Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (state_count < 10)_0  -
  Row   2:          1  (state_count < 10)_1  -

----------------Focused Condition View-------------------
Line       70 Item    1  (~PO[9] && (state_count == 10))
Condition totals: 1 of 2 input terms covered = 50.00%

         Input Term   Covered  Reason for no coverage   Hint
        -----------  --------  ----------------------  --------------
            PO[9]          N  '_1' not hit            Hit '_1'
  (state_count == 10)         Y

    Rows:       Hits  FEC Target           Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  PO[9]_0              (state_count == 10)
  Row   2:    ***0***  PO[9]_1              -
  Row   3:          1  (state_count == 10)_0  ~PO[9]
  Row   4:          1  (state_count == 10)_1  ~PO[9]

----------------Focused Condition View-------------------
Line       82 Item    1  (state_count < 10)
Condition totals: 1 of 1 input term covered = 100.00%

         Input Term   Covered  Reason for no coverage   Hint
        -----------  --------  ----------------------  --------------
  (state_count < 10)         Y

    Rows:       Hits  FEC Target           Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (state_count < 10)_0  -
  Row   2:          1  (state_count < 10)_1  -

----------------Focused Condition View-------------------
Line       87 Item    1  ((PO[9:8] == 2) && (state_count == 10))
Condition totals: 1 of 2 input terms covered = 50.00%

         Input Term   Covered  Reason for no coverage   Hint
        -----------  --------  ----------------------  --------------
      (PO[9:8] == 2)         Y
  (state_count == 10)         N  '_0' not hit            Hit '_0'

    Rows:       Hits  FEC Target           Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (PO[9:8] == 2)_0      -
  Row   2:          1  (PO[9:8] == 2)_1      (state_count == 10)
  Row   3:    ***0***  (state_count == 10)_0  (PO[9:8] == 2)
  Row   4:          1  (state_count == 10)_1  (PO[9:8] == 2)

----------------Focused Condition View-------------------
Line       97 Item    1  (state_count < 9)
Condition totals: 1 of 1 input term covered = 100.00%

        Input Term   Covered  Reason for no coverage   Hint
       -----------  --------  ----------------------  --------------
  (state_count < 9)         Y

    Rows:       Hits  FEC Target           Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (state_count < 9)_0  -
  Row   2:          1  (state_count < 9)_1  -

----------------Focused Condition View-------------------
Line       108 Item   1  ((PO[9:8] == 3) && (state_count == 10))
Condition totals: 1 of 2 input terms covered = 50.00%

         Input Term   Covered  Reason for no coverage   Hint
        -----------  --------  ----------------------  --------------
      (PO[9:8] == 3)         Y
  (state_count == 10)         N  '_0' not hit            Hit '_0'

    Rows:       Hits  FEC Target           Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (PO[9:8] == 3)_0      -
  Row   2:          1  (PO[9:8] == 3)_1      (state_count == 10)
  Row   3:    ***0***  (state_count == 10)_0  (PO[9:8] == 3)
  Row   4:          1  (state_count == 10)_1  (PO[9:8] == 3)

----------------Focused Condition View-------------------
Line       117 Item   1  (tx_valid && (state_count == 12))
```

```
Condition totals: 2 of 2 input terms covered = 100.00%

         Input Term   Covered  Reason for no coverage   Hint
        -----------  --------  ----------------------  -------------
          tx_valid        Y
  (state_count == 12)       Y

    Rows:      Hits  FEC Target           Non-masking condition(s)
 ---------  ---------  --------------------  --------------------------
  Row   1:          1  tx_valid_0           -
  Row   2:          1  tx_valid_1           (state_count == 12)
  Row   3:          1  (state_count == 12)_0  tx_valid
  Row   4:          1  (state_count == 12)_1  tx_valid

----------------Focused Condition View-------------------
Line       120 Item    1  ((state_count >= 12) && (final_count <= 7))
Condition totals: 2 of 2 input terms covered = 100.00%

         Input Term   Covered  Reason for no coverage   Hint
        -----------  --------  ----------------------  -------------
  (state_count >= 12)       Y
   (final_count <= 7)       Y

    Rows:      Hits  FEC Target           Non-masking condition(s)
 ---------  ---------  --------------------  --------------------------
  Row   1:          1  (state_count >= 12)_0  -
  Row   2:          1  (state_count >= 12)_1  (final_count <= 7)
  Row   3:          1  (final_count <= 7)_0  (state_count >= 12)
  Row   4:          1  (final_count <= 7)_1  (state_count >= 12)
```

FSM Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
FSM States	5	5	0	100.00%
FSM Transitions	8	8	0	100.00%

```
================================FSM Details================================

FSM Coverage for instance /\top#DUT_Design  --

FSM_ID: cs
    Current State Object : cs
    ---------------------
    State Value MapInfo :
    ---------------------
Line        State Name            Value
----        ----------            -----
 142             IDLE                0
 147          CHK_CMD                1
 172        READ_DATA                4
 165         READ_ADD                3
 160            WRITE                2
    Covered States :
    ----------------
              State        Hit_count
              -----       ---------
               IDLE            18607
            CHK_CMD            12402
          READ_DATA            29400
           READ_ADD            43590
              WRITE            77660
    Covered Transitions :
    --------------------
Line         Trans_ID        Hit_count        Transition
----         --------        ---------        ----------
 146                0            12402        IDLE -> CHK_CMD
 157                1             1470        CHK_CMD -> READ_DATA
 155                2             3870        CHK_CMD -> READ_ADD
 152                3             7060        CHK_CMD -> WRITE
 149                4                1       CHK_CMD -> IDLE
 174                5             1470        READ_DATA -> IDLE
 167                6             3870        READ_ADD -> IDLE
 162                7             7060        WRITE -> IDLE


    Summary                  Bins     Hits   Misses  Coverage
    -------                  ----     ----   ------ --------
      FSM States                5        5        0  100.00%
      FSM Transitions           8        8        0  100.00%
```

Statement Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
Statements	77	77	0	100.00%

```
================================Statement Details================================

Statement Coverage for instance /\top#DUT_Design  --

    Line       Item                  Count     Source
    ----       ----                  -----     ------
  File SPI_SLAVE.sv
   1                                        module SPI_Slave #(IDLE=0 ,CHK_CMD = 1,WRITE = 2,READ_ADD = 3,READ_DATA = 4)(interface_SPI.DUT_Design inst_interface );
   2
   3                                            logic MOSI, SS_n, clk, rst_n, tx_valid;
   4                                            logic [7:0] tx_data;
   5                                            logic MISO;
   6                                            logic rx_valid;
   7                                            logic  [9:0] rx_data;
   8
```

```
 9                          reg [2:0] cs, ns;//current state and next state
 10                         reg [9:0] PO;
 11                         reg [7:0] temp;
 12                         reg SO, flag_rd ;
 13                         integer state_count = 0, final_count = 0;
 14                         reg Act_input_output;
 15
 16    1        377599              assign clk = inst_interface.clk;
 17    1         6201     assign rst_n=inst_interface.rst_n;
 18    1         24805    assign SS_n =inst_interface.SS_n;
 19    1         68841    assign MOSI =inst_interface.MOSI;
 20    1        136035              always@(*)begin
 21    1        136035        tx_valid=inst_interface.tx_valid;
 22    1        136035        tx_data =inst_interface.tx_data;
 23    1        136035        inst_interface.MISO = MISO;
 24    1        136035        inst_interface.rx_valid =rx_valid;
 25    1        136035        inst_interface.rx_data  =rx_data;
 26                         end
 27
 28                         assign MISO = SO; // output of reading
 29                         assign rx_data   = PO ;
 30                    // state Memory
 31    1        181659              always @(posedge clk ) begin //bug : reset must be syncrouns not Async
 32                                   if (!rst_n)begin
 33    1         3105                                cs <= IDLE;
 34    1         3105                    flag_rd<=0;
 35    1         3105                    final_count <=32'hFFFF_FFFF;
 36    1         3105                    state_count <=32'hFFFF_FFFF;
 37    1         3105                    temp <= 0 ;
 38                                   end
 39                                   else
 40    1        178554                                    cs <= ns;
 41                         end
 42    1         37204    always @ (cs)begin
 43                                   if (cs == IDLE)begin
 44    1         12402                                rx_valid = 0 ;
 45    1         12402                    PO  = 0 ;
 46    1         12402                    state_count = 0 ;
 47    1         12402                    final_count = 0 ;
 48    1         12402                    SO = 0 ;
 49    1         12402                    Act_input_output = 0;
 50                                   end
 51                         end
 52
 53    1        181151              always @(posedge clk) begin//* bug
 54
 55                                   case (cs)
 56                                                IDLE : begin
 57
 58    1         18607                                rx_valid = 0 ;
 59    1         18607                     PO  = 0 ;
 60    1         18607                     state_count = 0 ;
 61    1         18607                     final_count = 0 ;
 62    1         18607                     SO = 0 ;
 63    1         18607                     Act_input_output = 0;
 64                                           end
 65                                        WRITE: begin //done
 66
 67                                                                  if (state_count < 10 )begin //0 ,
 68    1         70600                                                       PO = {PO[8:0] , MOSI} ;
 69    1         70600                                                       state_count = state_count + 1 ;
 70                                                                        if (PO[9] ==1'b0 && state_count ==10)begin
 71    1         7060                                                          rx_valid = 1 ;
 72                                                                      end
 73                                                                      else
 74    1         63540                                                       rx_valid = 0 ;
 75                                                                  end
 76                                                                  else
 77    1         7060                                                        rx_valid = 0 ;
 78
 79                                                 end
 80                                        READ_ADD: begin // done
 81
 82                                                 if (state_count<10 )begin
 83    1         38700                                       PO = {PO[8:0] , MOSI} ;
 84    1         38700                                       state_count = state_count + 1 ;
 85    1         38700                                       rx_valid = 0 ;
 86                                                 end
 87                                                 if (PO[9:8]==2'b10 && state_count==10)begin
 88    1         5700                                        rx_valid = 1 ;
 89    1         5700                                        flag_rd = 1;
 90                                                 end
 91                                                 else
 92    1         37382                                                 rx_valid = 0 ;
 93                                                       end
 94                                        READ_DATA: begin // done
 95
 96                                                 if (Act_input_output == 0)begin
 97                                                       if (state_count<9 )begin
 98    1         13230                                       PO = {PO[8:0] , MOSI} ;
 99    1         13230                                       state_count = state_count + 1 ;
100    1         13230                                             rx_valid = 0 ;
101                                                    end
102                                                    else begin
103    1         1470                                      PO = {PO[8:0] , MOSI} ;
104    1         1470                                      state_count = state_count + 1 ;
105    1         1470                                      rx_valid = 0 ;
106                                                    end
107
108                                                    if (PO[9:8]==2'b11 && state_count== 10 )begin
109    1         1470                                      rx_valid = 1 ;
110    1         1470                                      Act_input_output = 1;
111    1         1470                                      flag_rd = 0;
112                                                    end
113                                                end
114                                                else begin
```

```
115       1          14700                                                                  state_count = state_count + 1 ;
116       1          14700                                                                  rx_valid = 0 ;
117                                                                         if (tx_valid && state_count == 12 )begin
118       1            735                                                                  temp = tx_data ;
119                                                                         end
120                                                                         if (state_count >= 12 && final_count <= 7)begin
121       1          11760                                                                                  SO = temp [7 - final_count ];
122       1          11760                                                                                  rx_valid =0 ;
123       1          11760                                                                                  final_count = final_count + 1 ;
124                                                                           end
125                                                                         end
126                                                                   end
127                                                 default: begin
128       1          12402                                                  rx_valid = 0 ;
129       1          12402                                        PO  = 0 ;
130       1          12402                                        state_count = 0 ;
131       1          12402                                        final_count = 0 ;
132       1          12402                                        SO = 0 ;
133       1          12402                                        Act_input_output = 0;
134
135                                                   end
136                                                   endcase
137                               end
138
139       1          130332                   always @(*) begin
140
141                                                   case (cs)
142                                                     IDLE:
143                                                                   if (SS_n)
144       1          13782                                                    ns = IDLE;
145                                                                   else
146       1          12402                                                    ns = CHK_CMD;
147                                                     CHK_CMD:
148                                                                   if (SS_n)
149       1            1                                                        ns = IDLE;
150                                                                   else begin//SS_n = 0
151                                                                     if (MOSI==0) //MOSI = 0
152       1          7062                                                       ns = WRITE;
153                                                                     else  begin //MOSI = 1
154                                                                                if (!flag_rd) //flag_rd = 0
155       1          10930                                                                      ns = READ_ADD;
156                                                                                else  //flag_rd = 1
157       1          1470                                                          ns = READ_DATA;
158                                                                     end
159                                                                   end
160                                                     WRITE:
161                                                                   if (SS_n)
162       1          7060                                                     ns = IDLE;
163                                                                   else
164       1          38404                                                    ns = WRITE;
165                                                     READ_ADD:
166                                                                   if (SS_n)
167       1          3870                                                     ns = IDLE;
168                                                                   else begin
169       1          25045                                                    ns = READ_ADD;
170                                                                     // flag_rd = 1;
171                                                                   end
172                                                     READ_DATA:
173                                                                   if (SS_n)
174       1          1470                                                     ns = IDLE;
175                                                                   else begin
176       1          8835                                                     ns = READ_DATA;
177                                                                     // flag_rd = 0;
178                                                                   end
179
180       1            1                    default: ns = IDLE;
```

Toggle Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
Toggles	232	232	0	100.00%

==========Toggle Details==========

Toggle Coverage for instance /\top#DUT_Design --

Node	1H->0L	0L->1H	"Coverage"
Act_input_output	1	1	100.00
MISO	1	1	100.00
MOSI	1	1	100.00
PO[9-0]	1	1	100.00
SO	1	1	100.00
SS_n	1	1	100.00
clk	1	1	100.00
cs[2-0]	1	1	100.00
final_count[31-0]	1	1	100.00
flag_rd	1	1	100.00
ns[2-0]	1	1	100.00
rst_n	1	1	100.00
rx_data[9-0]	1	1	100.00
rx_valid	1	1	100.00
state_count[31-0]	1	1	100.00
temp[7-0]	1	1	100.00
tx_data[7-0]	1	1	100.00
tx_valid	1	1	100.00

Total Node Count = 116
Toggled Node Count = 116
Untoggled Node Count = 0

Toggle Coverage = 100.00% (232 of 232 bins)

Total Coverage By Instance (filtered view): 96.42%

Branch coverage – Statement coverage – Toggle coverage ----→ are 100% but conditional coverage not 100% and I cant get 100%.

Function coverage