SPI INTERFACE Project

Verilog Implementation and Design Flow

MAHMOUD GHAMRY

AUG.2024

Contents

1. **Introduction**

The Serial Peripheral Interface (SPI) is a widely used synchronous serial communication protocol that enables the transfer of data between a microcontroller and peripheral devices such as sensors, memory chips, and display modules. Characterized by its simplicity and speed, SPI operates using a master-slave architecture where a master device controls multiple slave devices through four primary signals: MISO (Master In Slave Out), MOSI (Master Out Slave In), SCLK (Serial Clock), and SS_n (Slave Select).

In the realm of digital design, implementing SPI communication using Verilog—a hardware description language (HDL)—allows designers to create flexible and efficient custom interfaces tailored to specific requirements. Verilog, with its powerful constructs for describing and simulating digital systems, enables precise control over the timing and functionality of the SPI protocol.

The essence of SPI communication in Verilog involves the creation of modules that encapsulate the behavior of both master and slave devices. These modules manage the data flow, handle synchronization issues, and ensure that data integrity is maintained throughout the communication process. The implementation encompasses the finite state machines (FSMs) for state transitions, shift registers for serial-to-parallel and parallel-to-serial data conversion, and control logic for signal synchronization.

This document explores the fundamental principles of the SPI protocol and provides a detailed guide to implementing both SPI master and slave modules using Verilog. Through practical examples and code snippets, we illustrate the key concepts and techniques required to achieve reliable SPI communication, making it an invaluable resource for anyone looking to integrate SPI interfaces into their digital designs.

2. Design Considerations

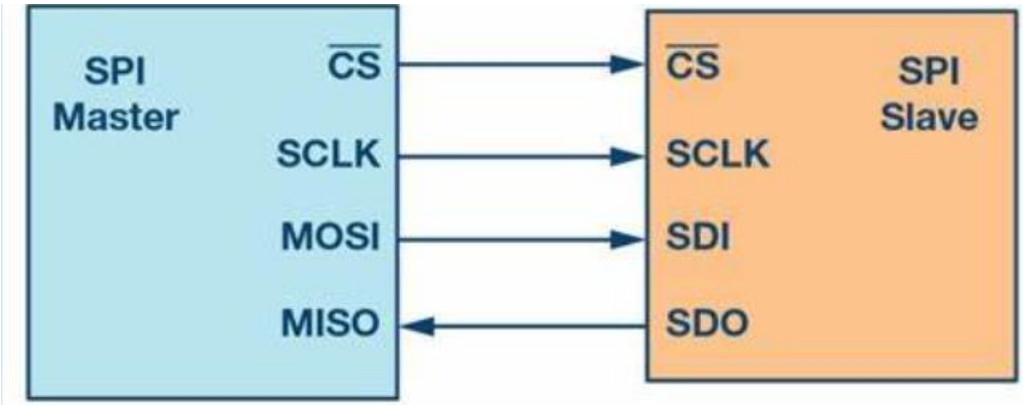


Figure 2-1: **SPI PROTOCOL**

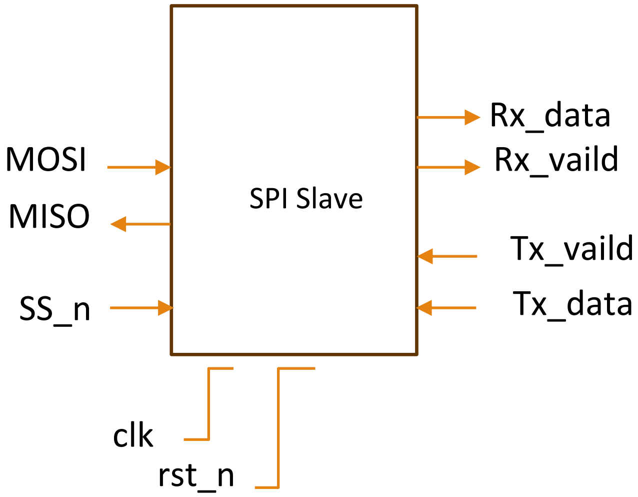


Figure 2-2: **SPI SLAVE**

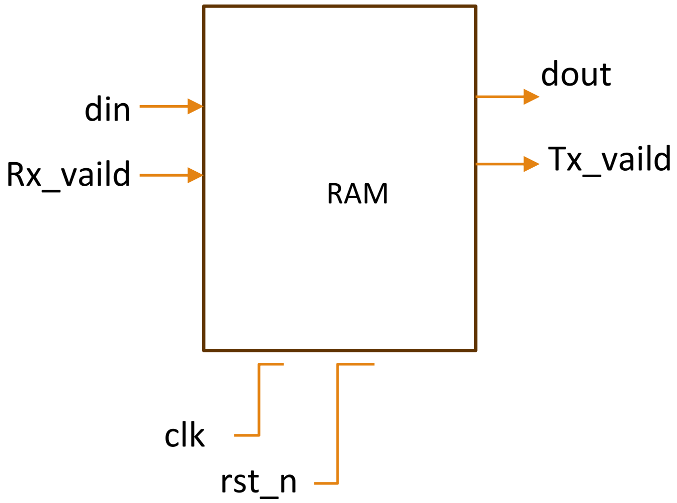


Figure 2-3: **SPI RAM**

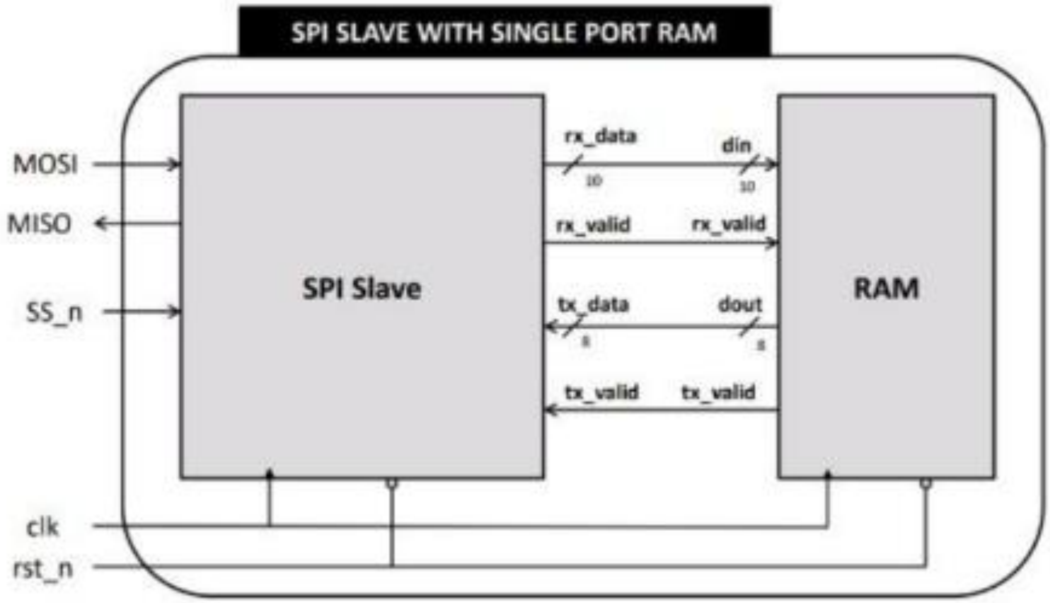
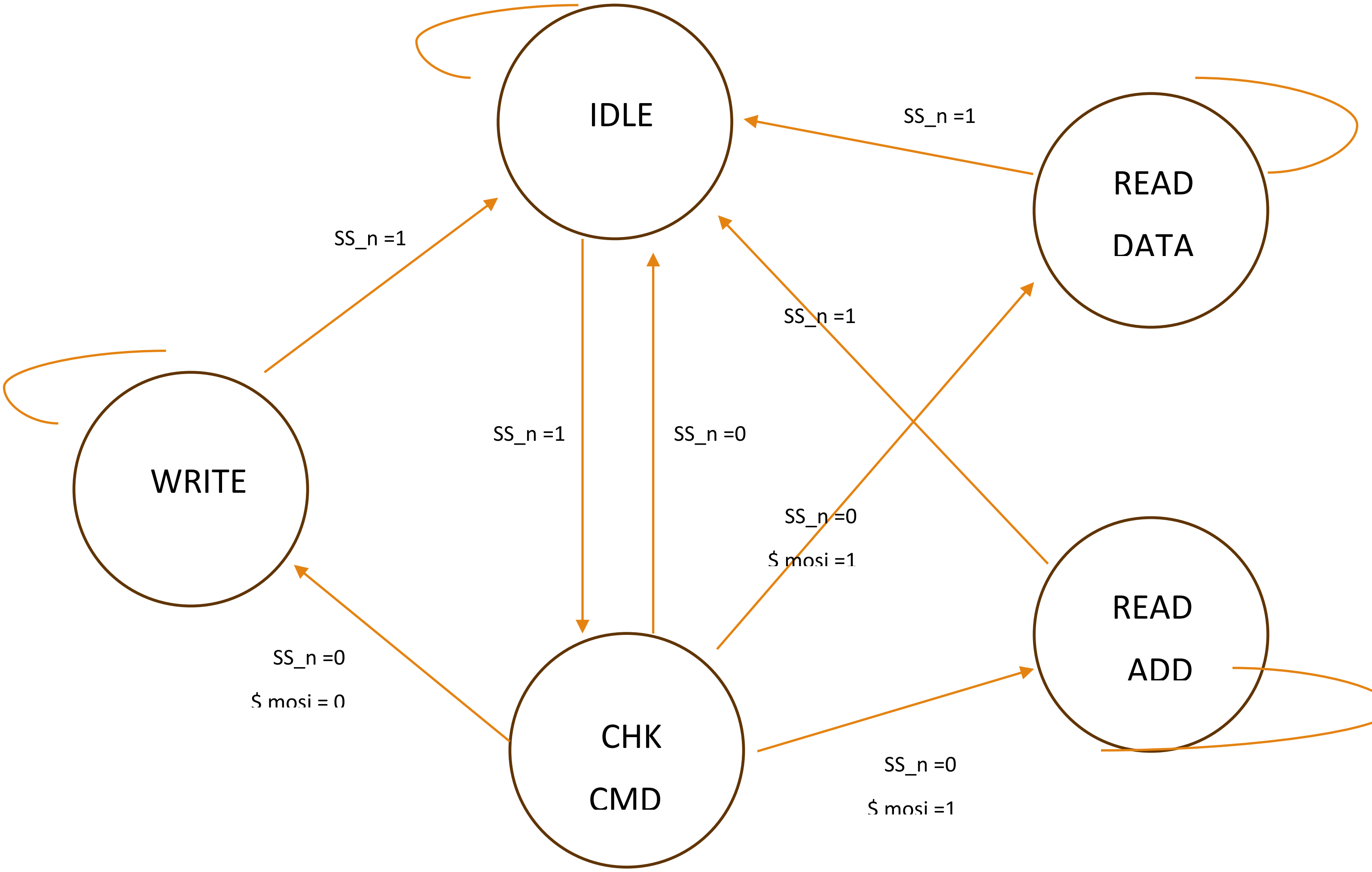


Figure 2-4: **SPI WRAPPER**

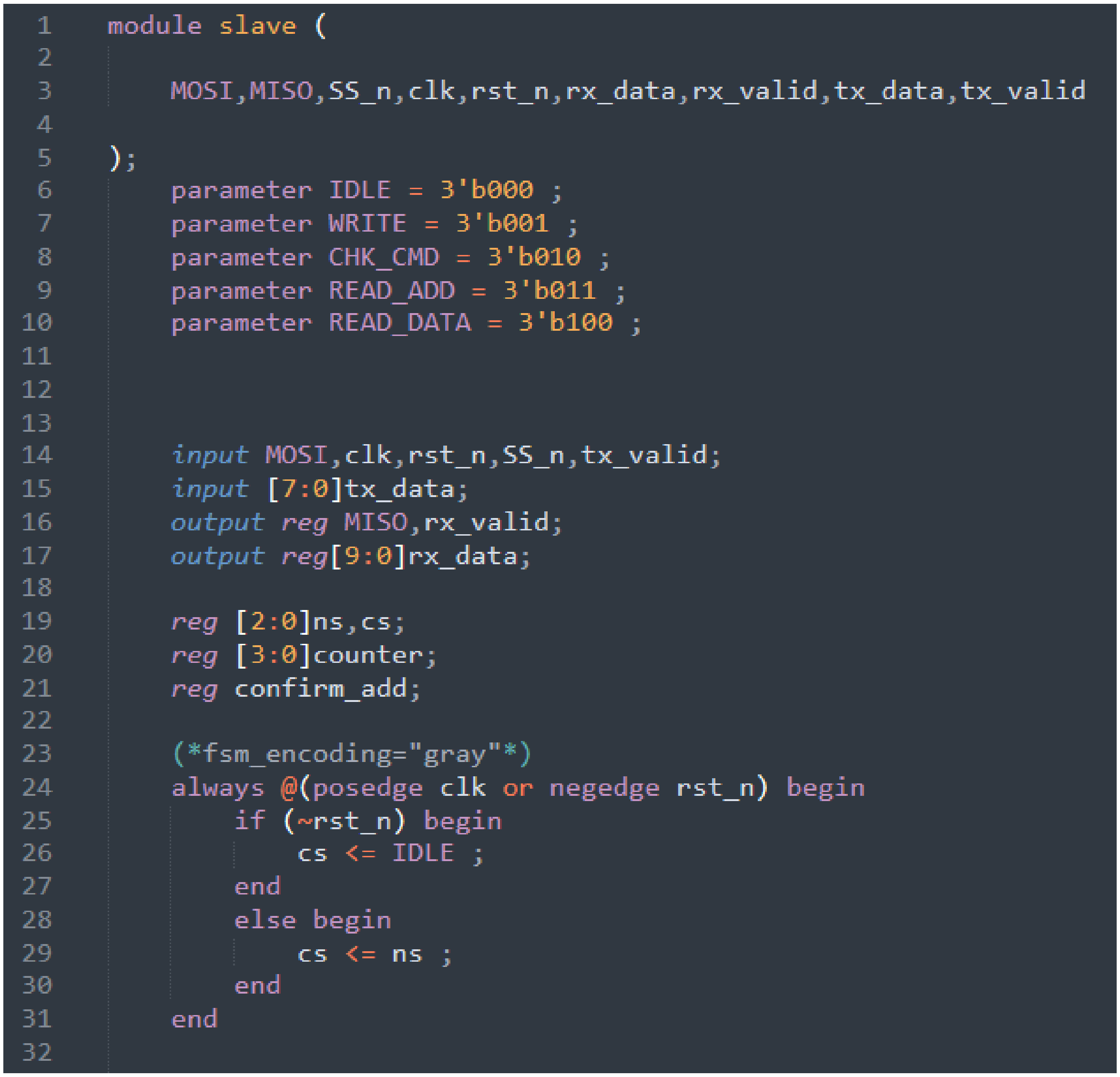
3.FSM


IDLE
SS_n =1
READ
DATA
SS_n =1
SS_n =1
WRITE
SS_n =1
SS_n =0
SS_n =0
$ mosi =1
READ
ADD
SS_n =0
$ mosi = 0
CHK
CMD
SS_n =0
$ mosi =1

3-20

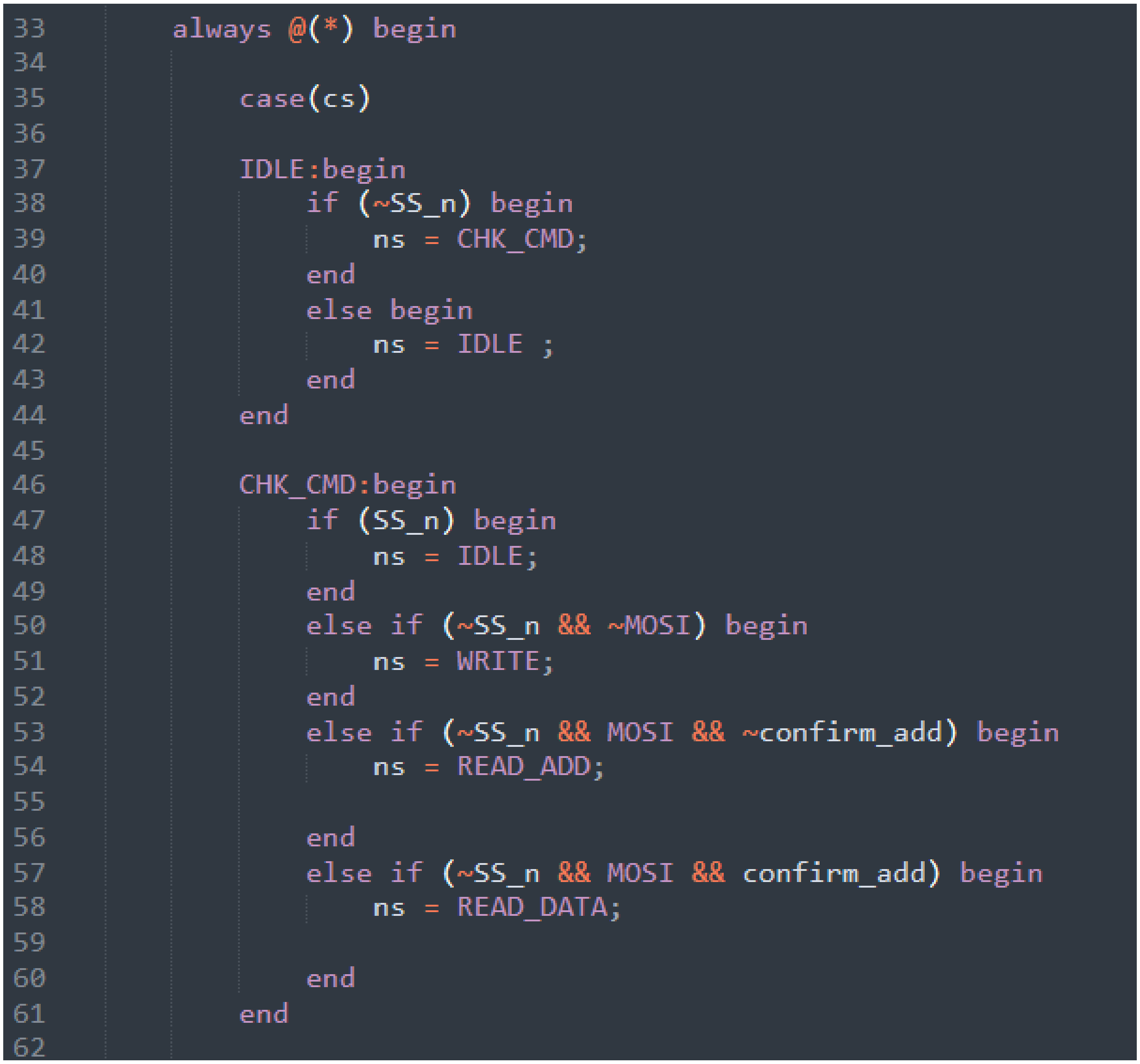
4 Coding

4.1 RTL DESIGN



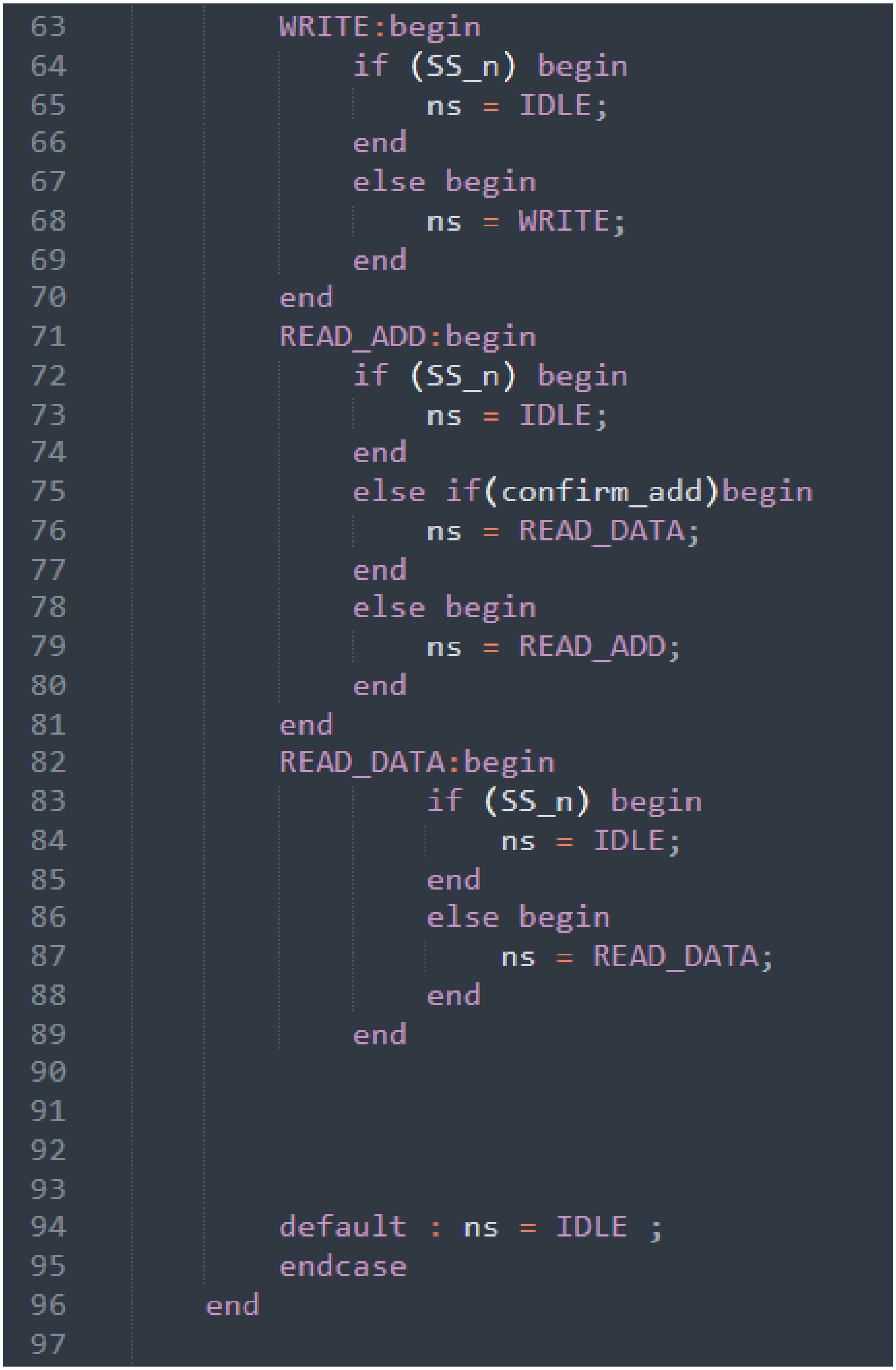
```verilog
module slave (

    MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid

);
    parameter IDLE = 3'b000 ;
    parameter WRITE = 3'b001 ;
    parameter CHK_CMD = 3'b010 ;
    parameter READ_ADD = 3'b011 ;
    parameter READ_DATA = 3'b100 ;



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

    reg [2:0]ns,cs;
    reg [3:0]counter;
    reg confirm_add;

    (*fsm_encoding="gray"*)
    always @(posedge clk or negedge rst_n) begin
        if (~rst_n) begin
            cs <= IDLE ;
        end
        else begin
            cs <= ns ;
        end
    end

```

Figure 4-1: **SPI SLAVE**



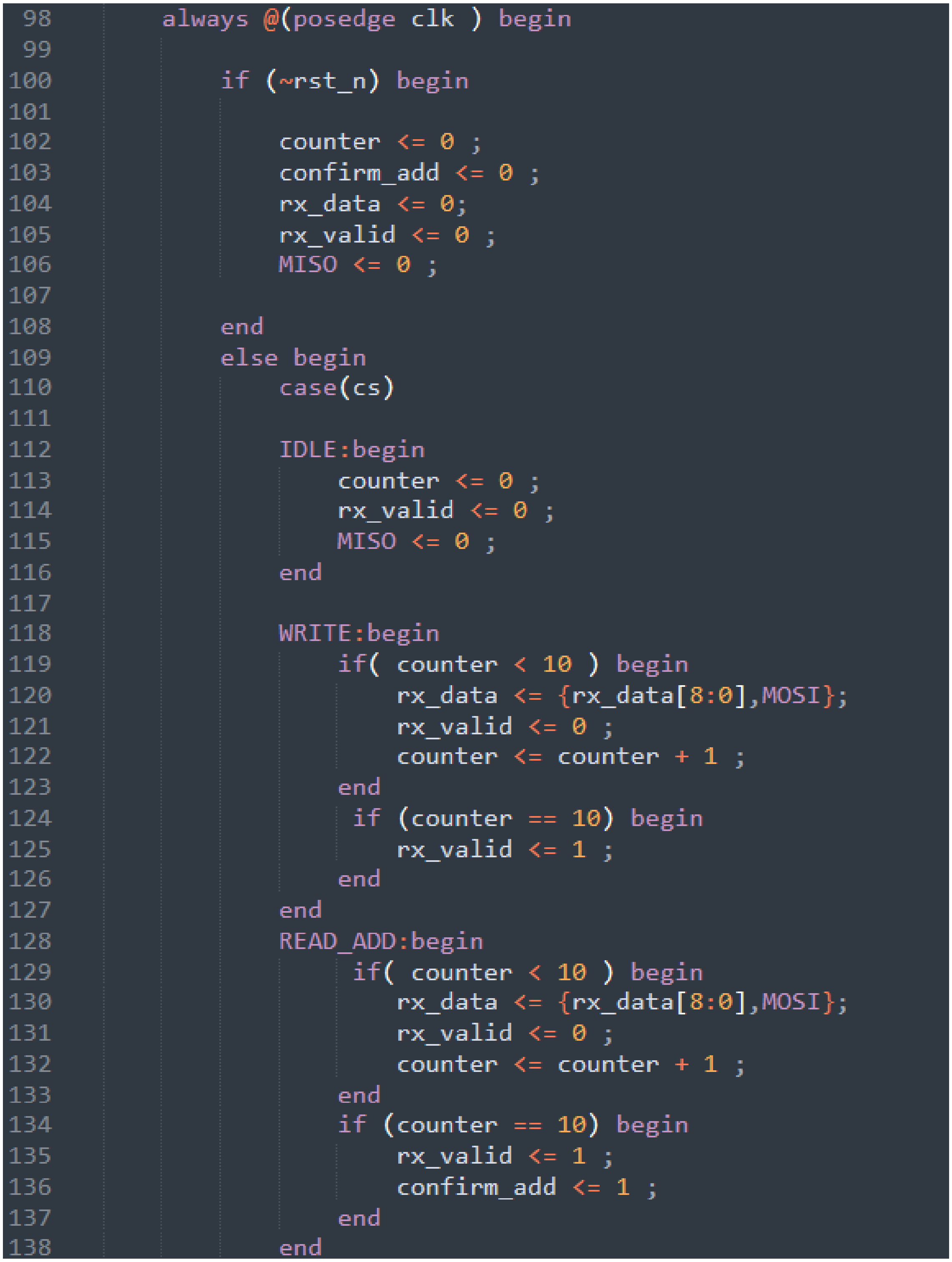
```
        always @(*) begin

            case(cs)

            IDLE:begin
                if (~SS_n) begin
                    ns = CHK_CMD;
                end
                else begin
                    ns = IDLE ;
                end
            end

            CHK_CMD:begin
                if (SS_n) begin
                    ns = IDLE;
                end
                else if (~SS_n && ~MOSI) begin
                    ns = WRITE;
                end
                else if (~SS_n && MOSI && ~confirm_add) begin
                    ns = READ_ADD;

                end
                else if (~SS_n && MOSI && confirm_add) begin
                    ns = READ_DATA;

                end
            end

```

Figure 4-2: **SPI SLAVE**



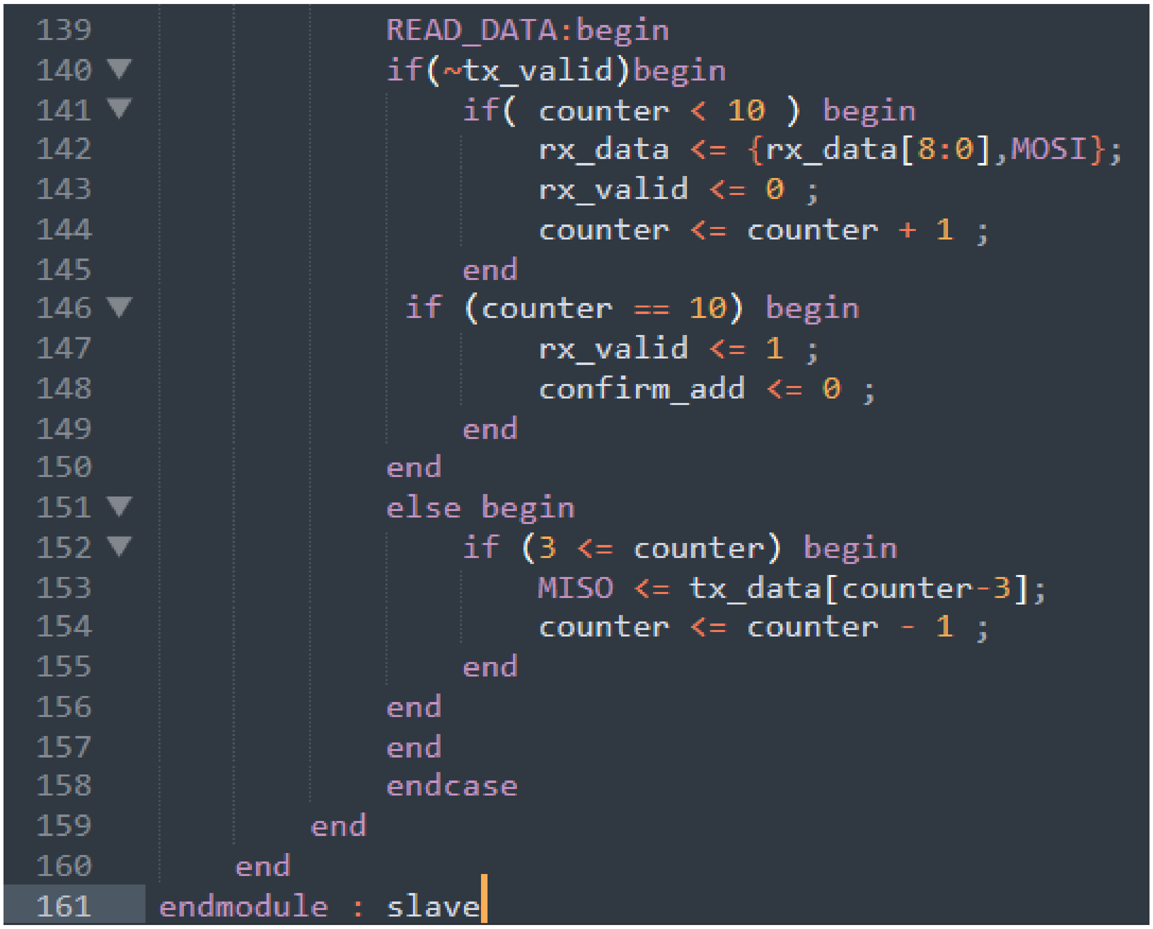
```
            WRITE:begin
                if (SS_n) begin
                    ns = IDLE;
                end
                else begin
                    ns = WRITE;
                end
            end
            READ_ADD:begin
                if (SS_n) begin
                    ns = IDLE;
                end
                else if(confirm_add)begin
                    ns = READ_DATA;
                end
                else begin
                    ns = READ_ADD;
                end
            end
            READ_DATA:begin
                    if (SS_n) begin
                        ns = IDLE;
                    end
                    else begin
                        ns = READ_DATA;
                    end
                end




            default : ns = IDLE ;
            endcase
        end

```

Figure 4-3: **SPI SLAVE**



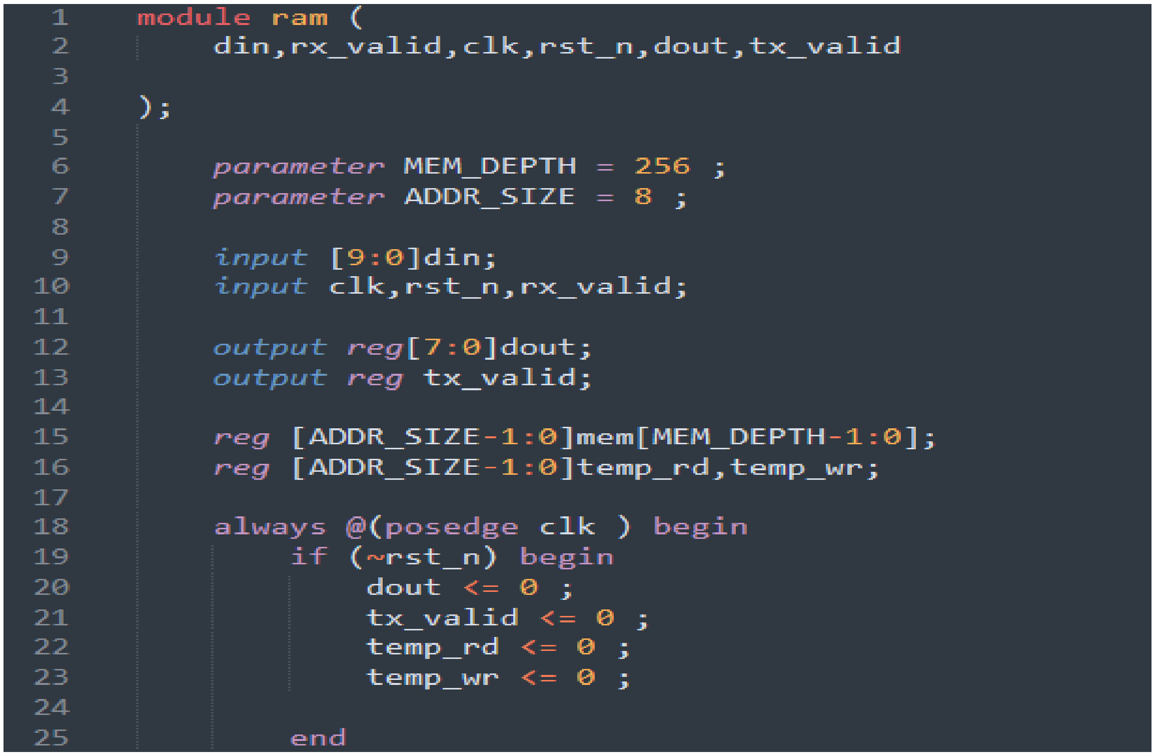
```
    always @(posedge clk ) begin

        if (~rst_n) begin

            counter <= 0 ;
            confirm_add <= 0 ;
            rx_data <= 0;
            rx_valid <= 0 ;
            MISO <= 0 ;

        end
        else begin
            case(cs)

            IDLE:begin
                counter <= 0 ;
                rx_valid <= 0 ;
                MISO <= 0 ;
            end

            WRITE:begin
                if( counter < 10 ) begin
                    rx_data <= {rx_data[8:0],MOSI};
                    rx_valid <= 0 ;
                    counter <= counter + 1 ;
                end
                 if (counter == 10) begin
                    rx_valid <= 1 ;
                end
            end
            READ_ADD:begin
                 if( counter < 10 ) begin
                    rx_data <= {rx_data[8:0],MOSI};
                    rx_valid <= 0 ;
                    counter <= counter + 1 ;
                end
                if (counter == 10) begin
                    rx_valid <= 1 ;
                    confirm_add <= 1 ;
                end
            end
```

Figure 4-4: **SPI SLAVE**



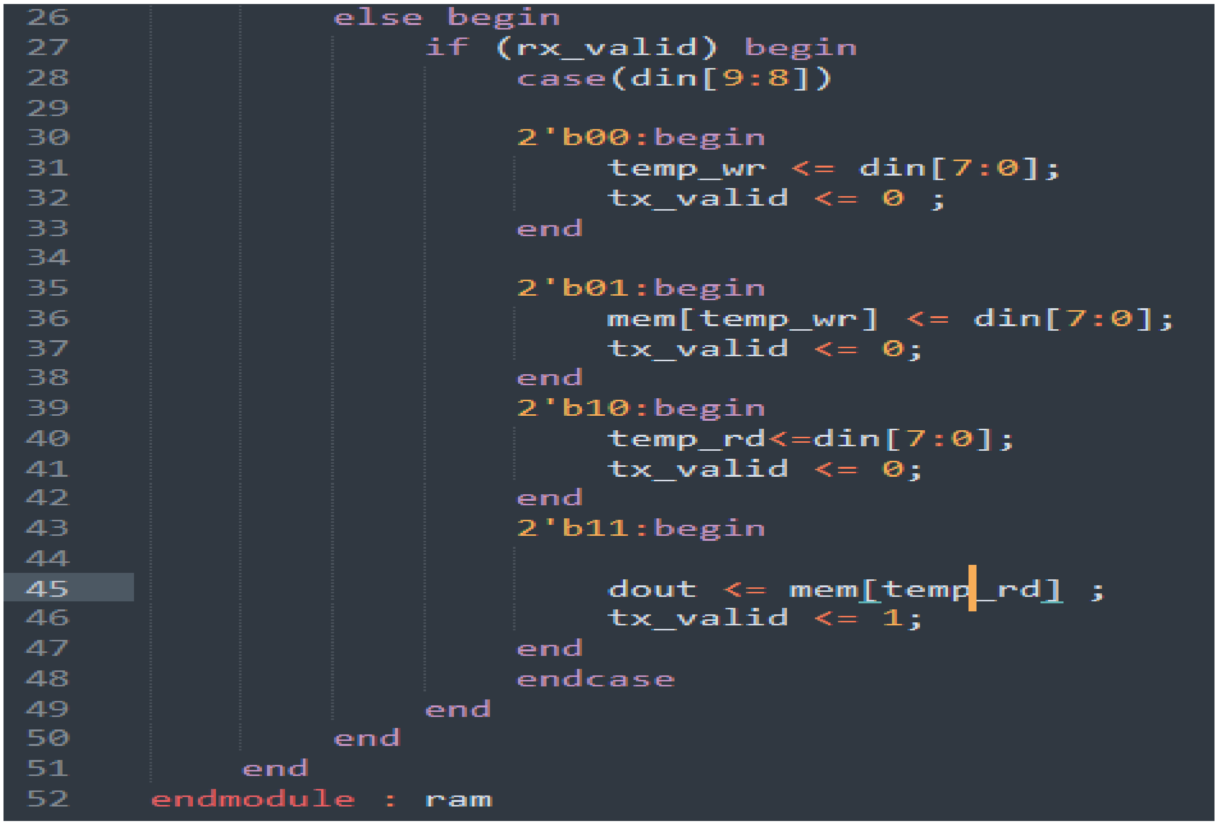
```
            READ_DATA:begin
            if(~tx_valid)begin
                if( counter < 10 ) begin
                    rx_data <= {rx_data[8:0],MOSI};
                    rx_valid <= 0 ;
                    counter <= counter + 1 ;
                end
             if (counter == 10) begin
                    rx_valid <= 1 ;
                    confirm_add <= 0 ;
                end
            end
            else begin
                if (3 <= counter) begin
                    MISO <= tx_data[counter-3];
                    counter <= counter - 1 ;
                end
            end
            end
            endcase
        end
    end
endmodule : slave
```

Figure 4-5: **SPI SLAVE**



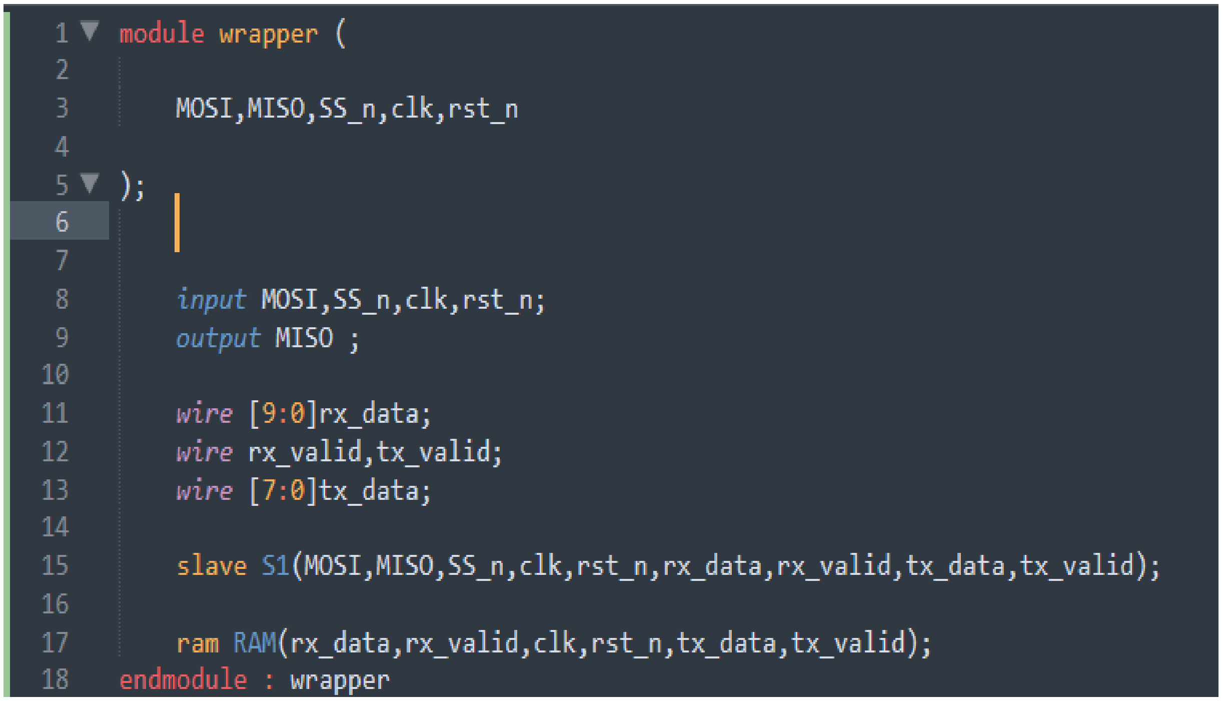
```
module ram (
    din,rx_valid,clk,rst_n,dout,tx_valid

);

    parameter MEM_DEPTH = 256 ;
    parameter ADDR_SIZE = 8 ;

    input [9:0]din;
    input clk,rst_n,rx_valid;

    output reg[7:0]dout;
    output reg tx_valid;

    reg [ADDR_SIZE-1:0]mem[MEM_DEPTH-1:0];
    reg [ADDR_SIZE-1:0]temp_rd,temp_wr;

    always @(posedge clk ) begin
        if (~rst_n) begin
            dout <= 0 ;
            tx_valid <= 0 ;
            temp_rd <= 0 ;
            temp_wr <= 0 ;

        end
```

Figure 4-6: **SPI RAM**



```
        else begin
            if (rx_valid) begin
                case(din[9:8])

                2'b00:begin
                    temp_wr <= din[7:0];
                    tx_valid <= 0 ;
                end

                2'b01:begin
                    mem[temp_wr] <= din[7:0];
                    tx_valid <= 0;
                end
                2'b10:begin
                    temp_rd<=din[7:0];
                    tx_valid <= 0;
                end
                2'b11:begin

                    dout  <= mem[temp_rd] ;
                    tx_valid <= 1;
                end
                endcase
            end
        end
    end
endmodule : ram
```

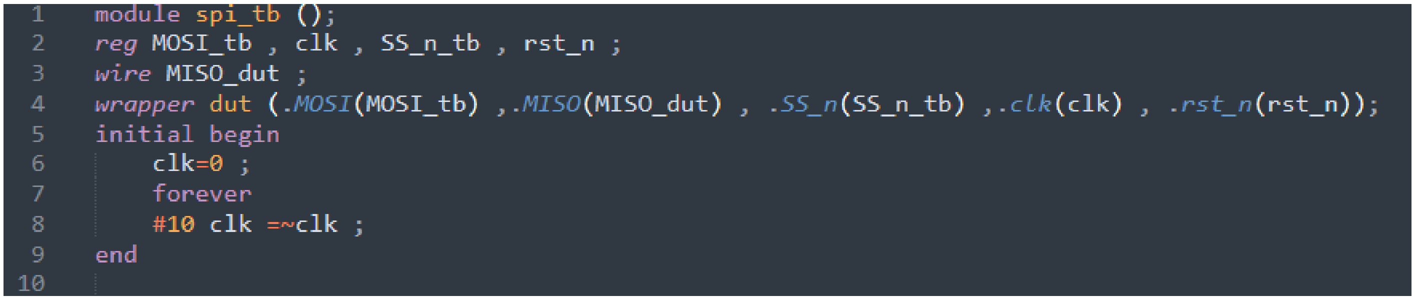
Figure 4-7: **SPI RAM**



```
module wrapper (

    MOSI,MISO,SS_n,clk,rst_n

);


    input MOSI,SS_n,clk,rst_n;
    output MISO ;

    wire [9:0]rx_data;
    wire rx_valid,tx_valid;
    wire [7:0]tx_data;

    slave S1(MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);

    ram RAM(rx_data,rx_valid,clk,rst_n,tx_data,tx_valid);
endmodule : wrapper
```

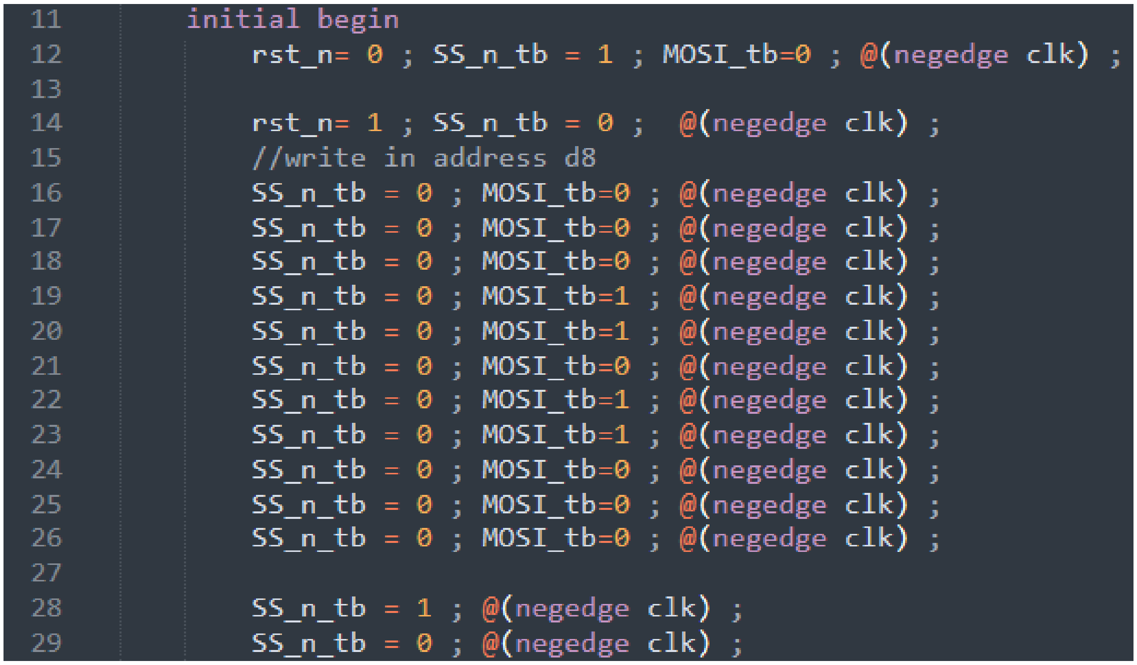
Figure 4-8: **SPI WRAPPER**

4.2 Test bench



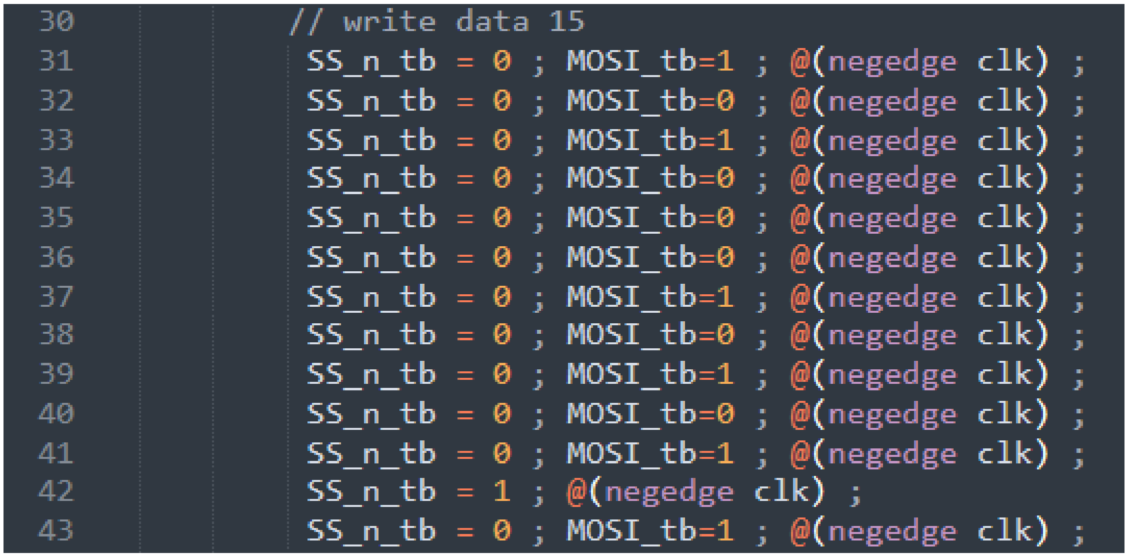
```
module spi_tb ();
reg MOSI_tb , clk , SS_n_tb , rst_n ;
wire MISO_dut ;
wrapper dut (.MOSI(MOSI_tb) ,.MISO(MISO_dut) , .SS_n(SS_n_tb) ,.clk(clk) , .rst_n(rst_n));
initial begin
    clk=0 ;
    forever
    #10 clk =~clk ;
end

```

Figure 4-2-1: **SPI Test bench**



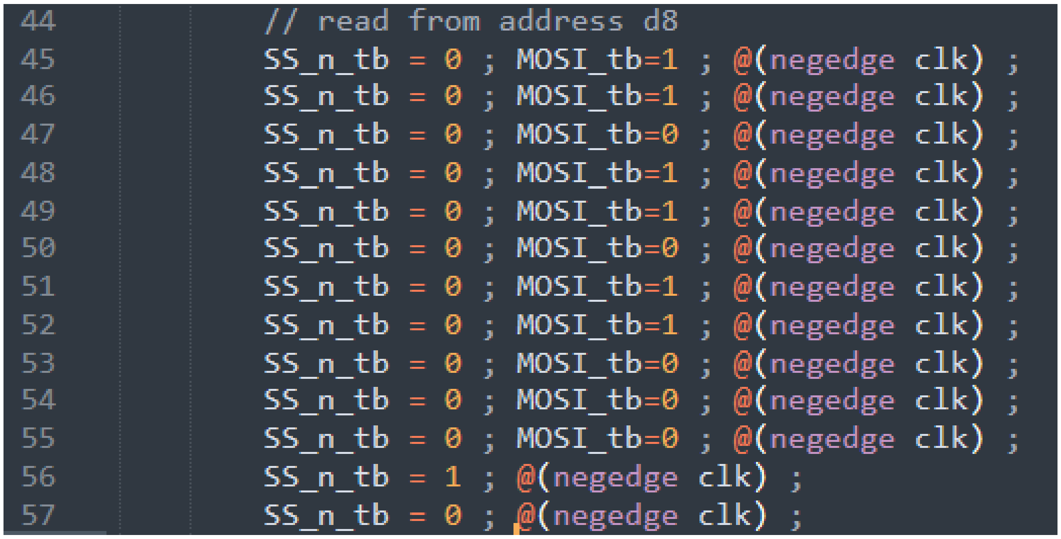
```
    initial begin
        rst_n= 0 ; SS_n_tb = 1 ; MOSI_tb=0 ; @(negedge clk) ;

        rst_n= 1 ; SS_n_tb = 0 ;  @(negedge clk) ;
        //write in address d8
        SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
        SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
        SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
        SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
        SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
        SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
        SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
        SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
        SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
        SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
        SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;

        SS_n_tb = 1 ; @(negedge clk) ;
        SS_n_tb = 0 ; @(negedge clk) ;
```

Figure 4-2-2: **SPI WRITE ADDERSS**



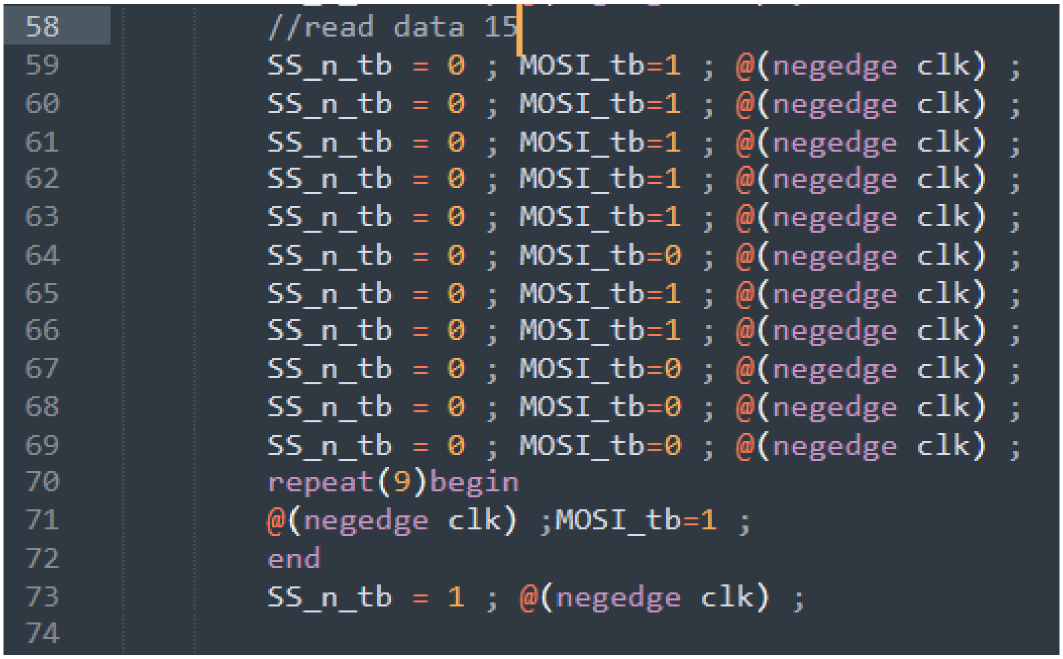
```
        // write data 15
         SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
         SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
         SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
         SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
         SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
         SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
         SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
         SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
         SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
         SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
         SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
         SS_n_tb = 1 ; @(negedge clk) ;
         SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
```

Figure 4-2-3: **SPI WRITE DATA**



```
// read from address d8
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
SS_n_tb = 1 ; @(negedge clk) ;
SS_n_tb = 0 ; @(negedge clk) ;
```

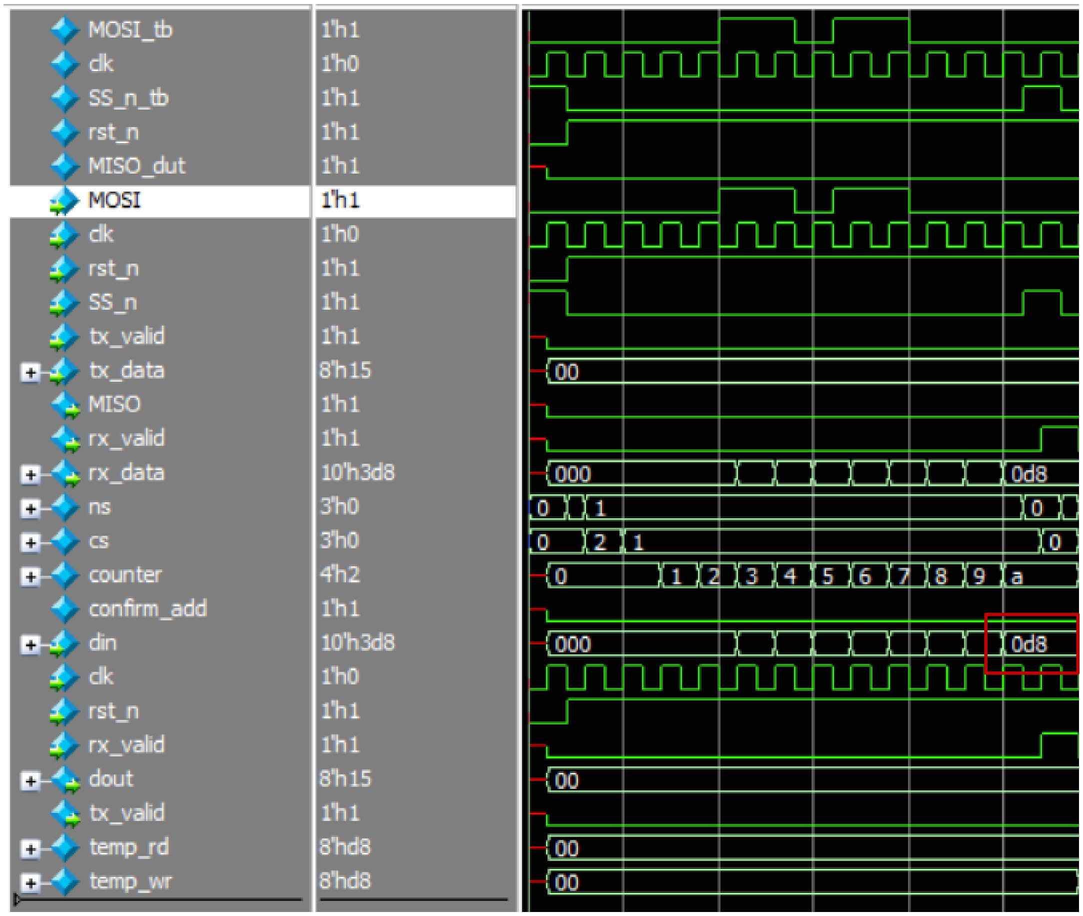
Figure 4-2-4: SPI READ ADDRESS



```
//read data 15
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=1 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
SS_n_tb = 0 ; MOSI_tb=0 ; @(negedge clk) ;
repeat(9)begin
@(negedge clk) ;MOSI_tb=1 ;
end
SS_n_tb = 1 ; @(negedge clk) ;
```

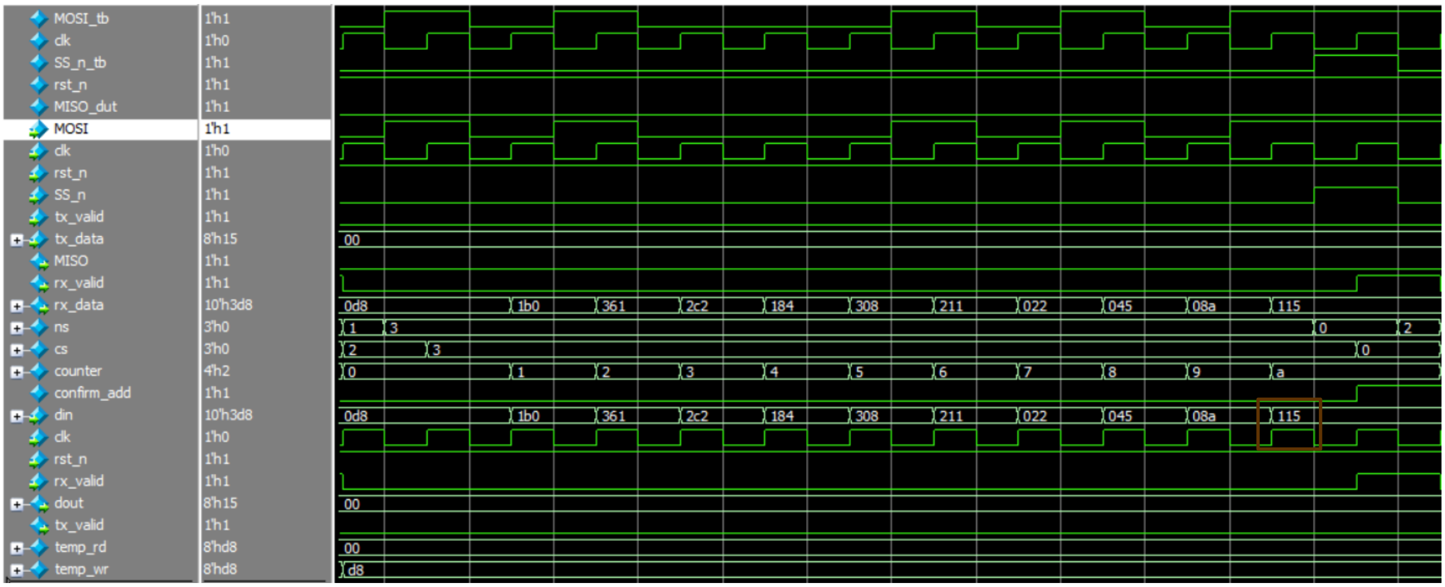
Figure 4-2-5: SPI SLAVE

4.2.1 Run Testbench in Questa sim 2021



```
MOSI_tb       1'h1
clk           1'h0
SS_n_tb       1'h1
rst_n         1'h1
MISO_dut      1'h1
MOSI          1'h1
clk           1'h0
rst_n         1'h1
SS_n          1'h1
tx_valid      1'h1
tx_data       8'h15
MISO          1'h1
rx_valid      1'h1
rx_data       10'h3d8
ns            3'h0
cs            3'h0
counter       4'h2
confirm_add   1'h1
din           10'h3d8
clk           1'h0
rst_n         1'h1
rx_valid      1'h1
dout          8'h15
tx_valid      1'h1
temp_rd       8'hd8
temp_wr       8'hd8
```

Figure 4-2-1: **WIRTE ADDRESS IN 0xd8**



```
MOSI_tb       1'h1
clk           1'h0
SS_n_tb       1'h1
rst_n         1'h1
MISO_dut      1'h1
MOSI          1'h1
clk           1'h0
rst_n         1'h1
SS_n          1'h1
tx_valid      1'h1
tx_data       8'h15
MISO          1'h1
rx_valid      1'h1
rx_data       10'h3d8
ns            3'h0
cs            3'h0
counter       4'h2
confirm_add   1'h1
din           10'h3d8
clk           1'h0
rst_n         1'h1
rx_valid      1'h1
dout          8'h15
tx_valid      1'h1
temp_rd       8'hd8
temp_wr       8'hd8
```

Figure 4-2-2: **WIRTE DATA 0x15**

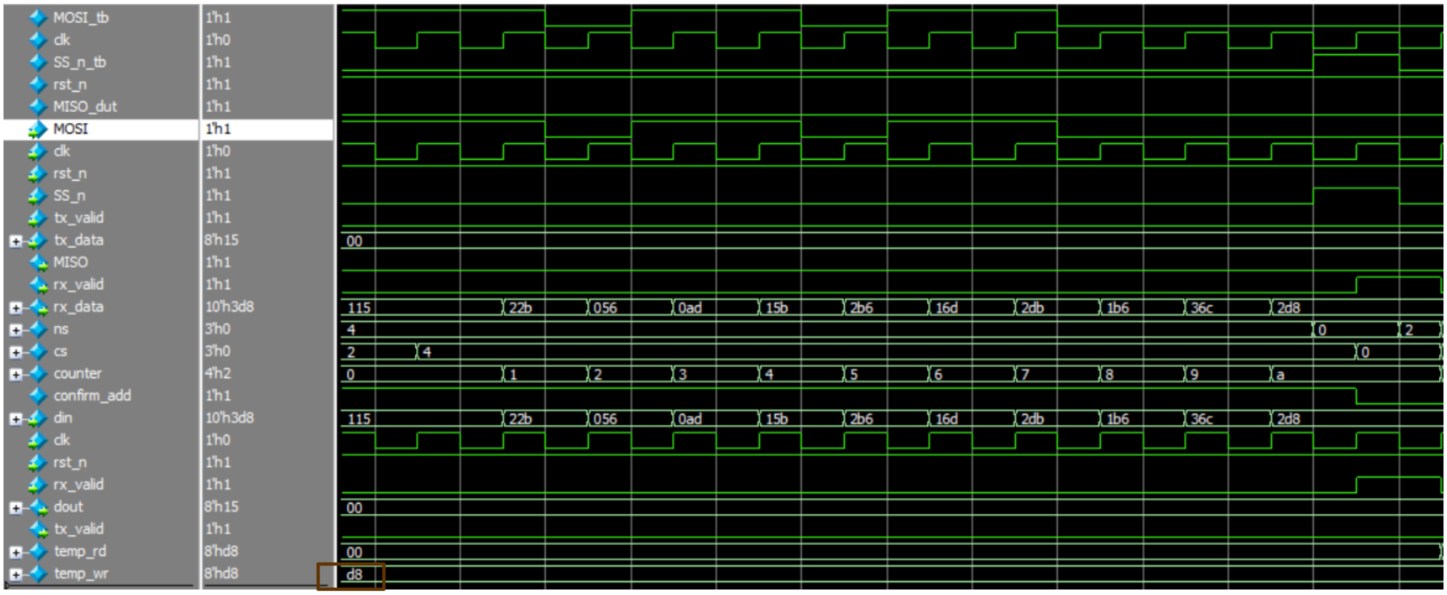


Figure 4-2-3: **READ ADDRESS FROM 0xd8**

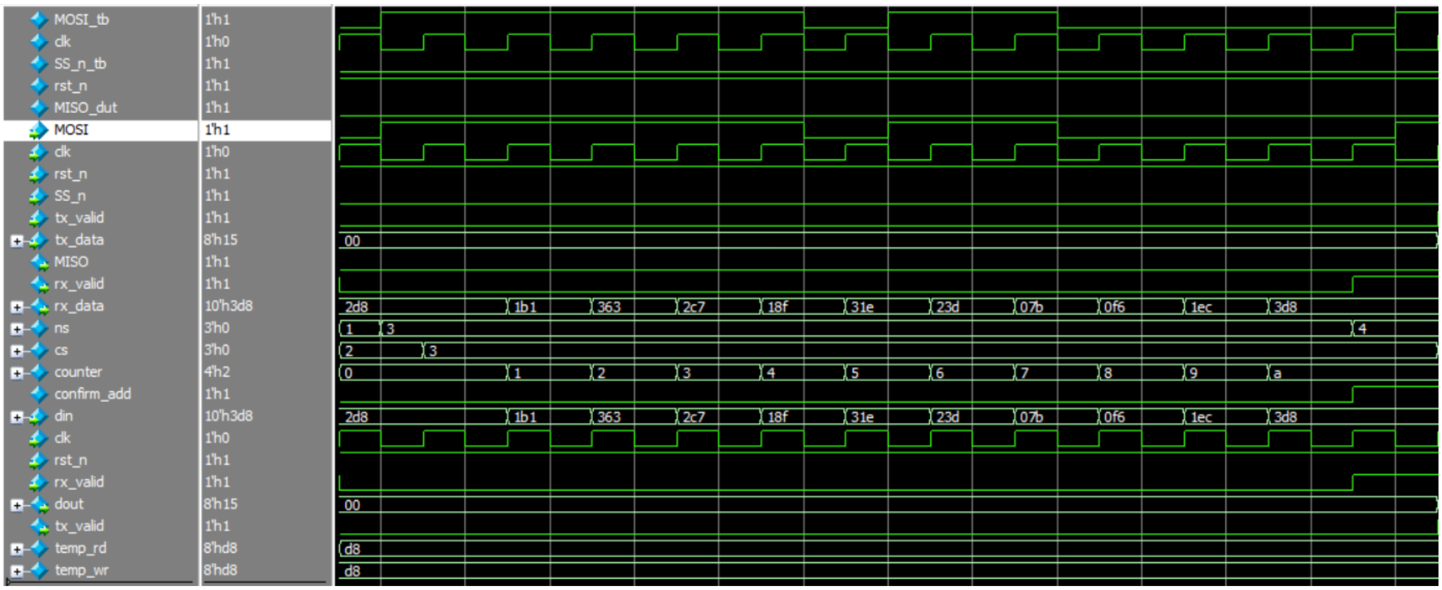


Figure 4-2-4: **READ DATA FROM MEMEORY**

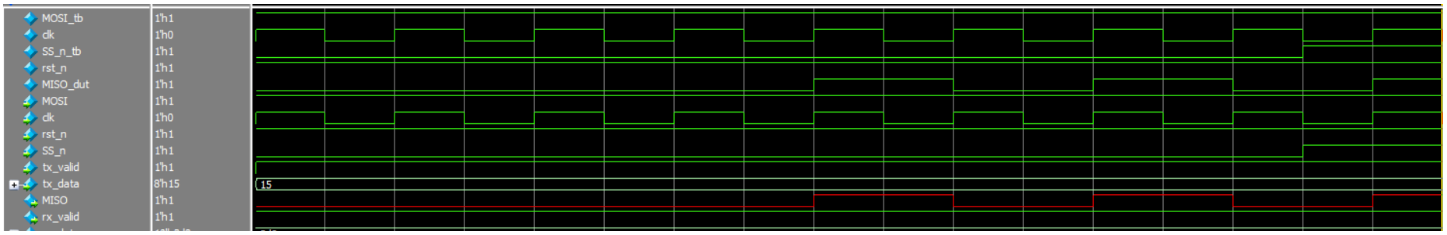


Figure 4-2-5: **MISO OUTPUT**

13-20

5.USING VIVADO 2018

5.1 SEQUENTIAL ENCODING

5.1.1 Elaborated Design

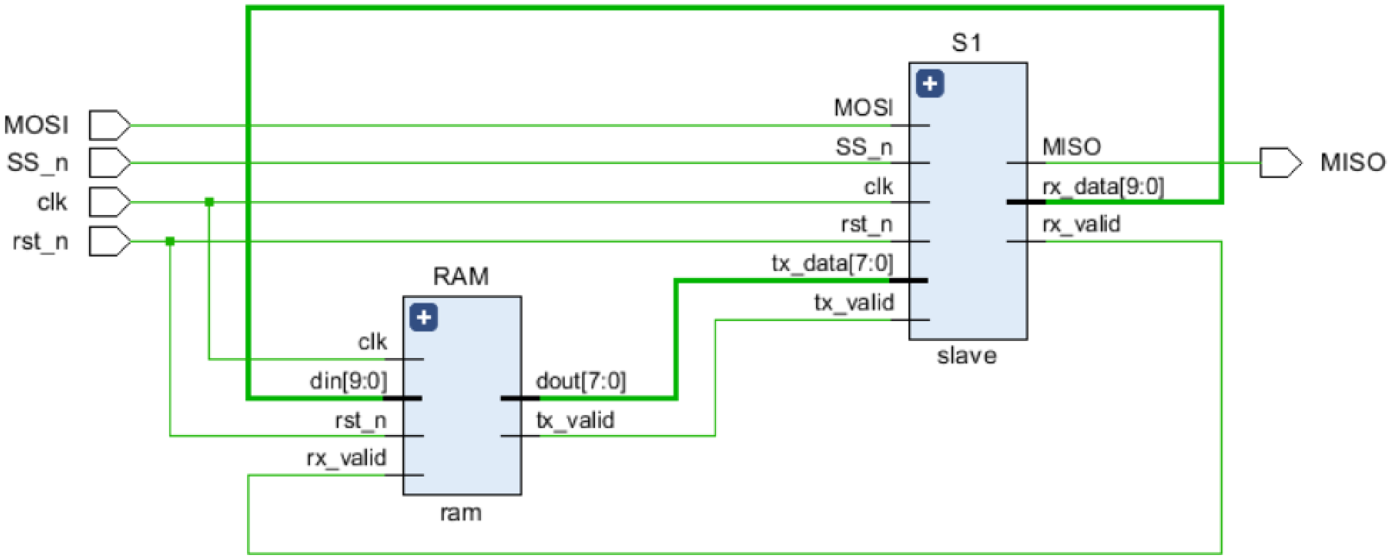


Figure 5-1-1: Elaborated Design

5.1.2. Synthesis Design

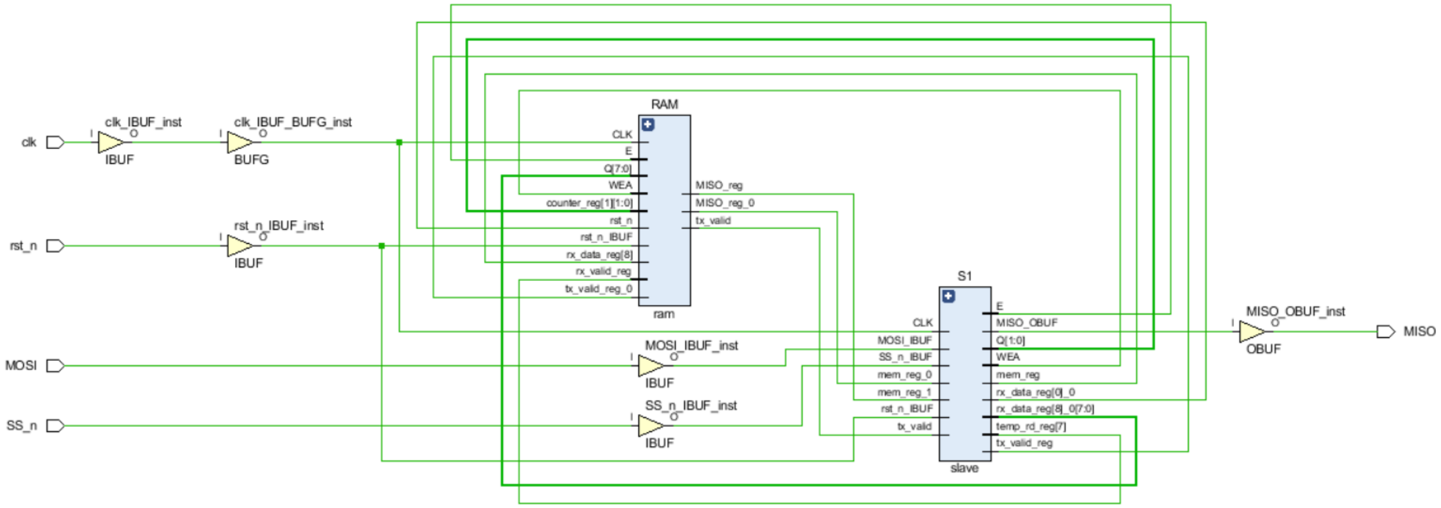


Figure 5-1-2: Synthesis Design

5.1.3. Implementation Design

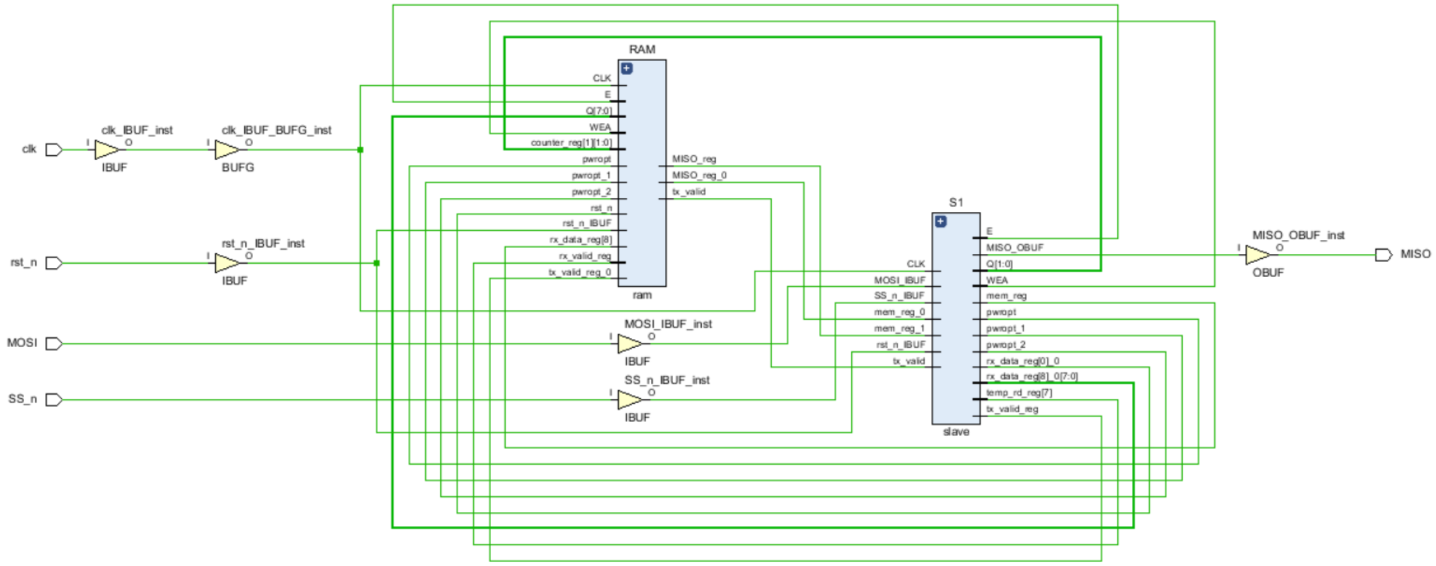


Figure 5-1-3-1: Implementation Design

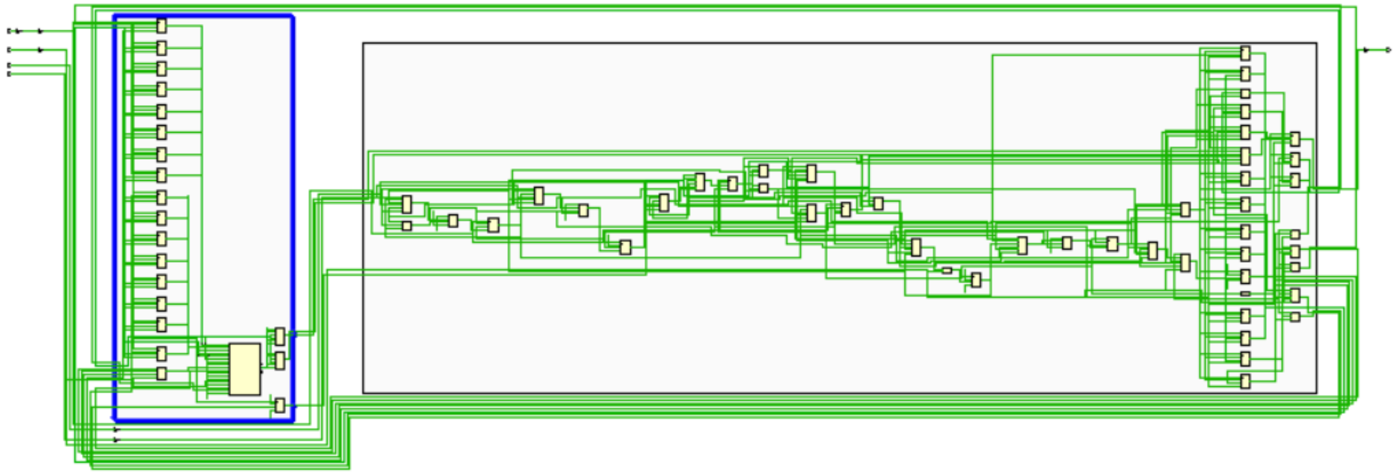
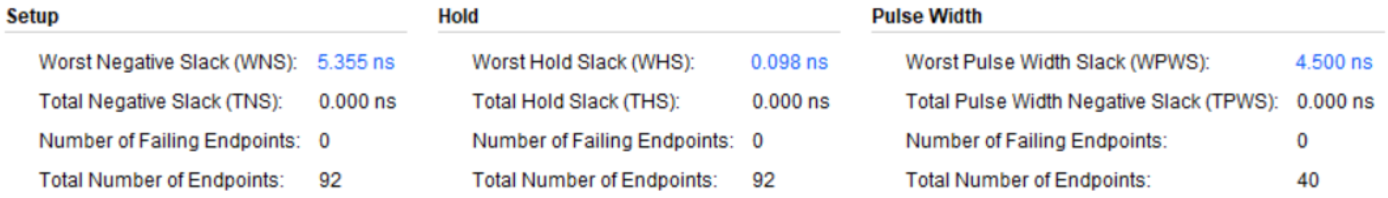


Figure 5-1-3-2: Implementation Design

5.1.4.REPORT



Setup

Worst Negative Slack (WNS):	5.355 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	92

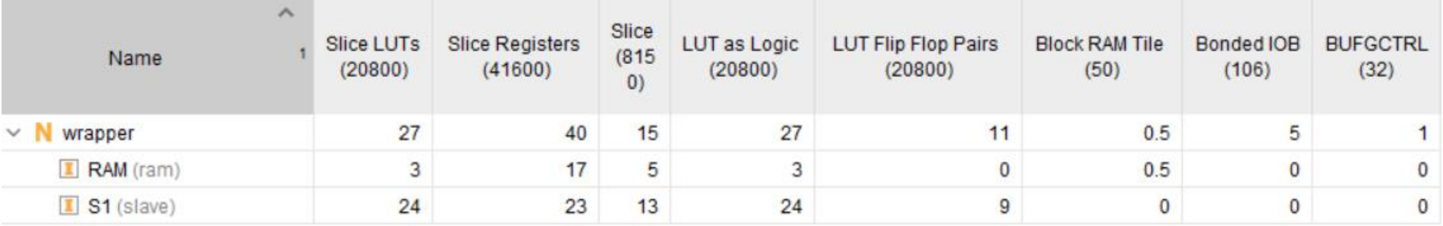
Hold

Worst Hold Slack (WHS):	0.098 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	92

Pulse Width

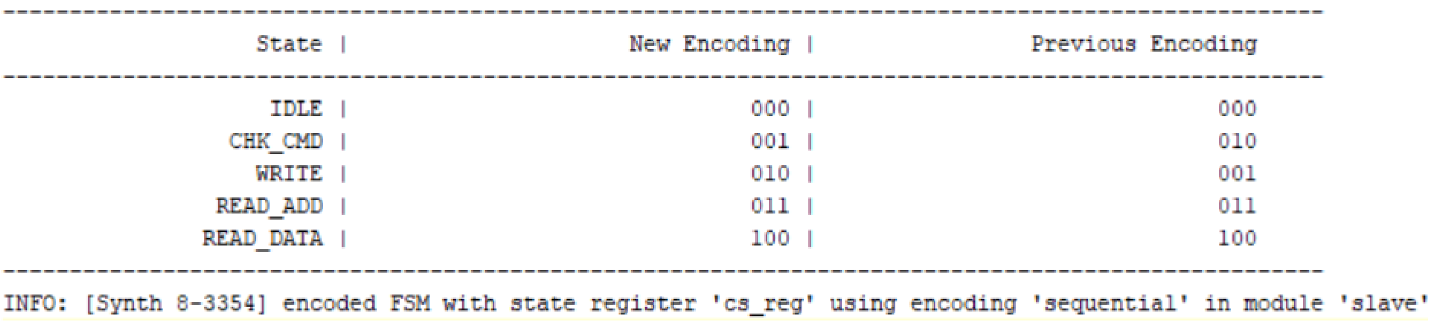
Worst Pulse Width Slack (WPWS):	4.500 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	40

Figure 5.1.4.1: TIME



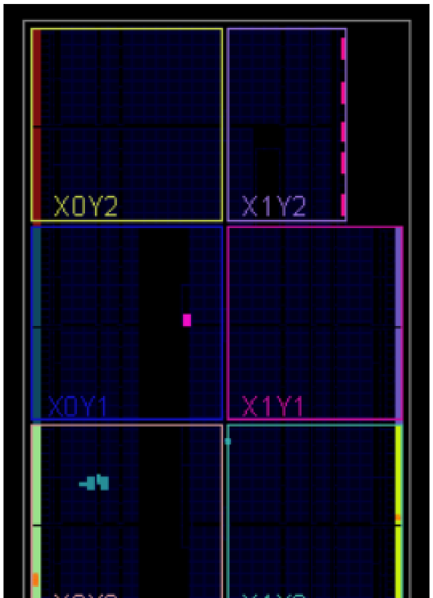
Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
wrapper	27	40	15	27	11	0.5	5	1
RAM (ram)	3	17	5	3	0	0.5	0	0
S1 (slave)	24	23	13	24	9	0	0	0

Figure 5.1.4.2: UTILIZATION



```
---------------------------------------------------------------------------------------------------
                   State |                     New Encoding |                Previous Encoding
---------------------------------------------------------------------------------------------------
                    IDLE |                              000 |                              000
                 CHK_CMD |                              001 |                              010
                   WRITE |                              010 |                              001
                READ_ADD |                              011 |                              011
               READ_DATA |                              100 |                              100
---------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'cs_reg' using encoding 'sequential' in module 'slave'
```

Figure 5.1.4.3: SEQUENTIAL ENCODING



16-20 *Figure 5.1.5* DEVICE

5.2 GRAY ENCODING

5.2.1 Elaborated Design

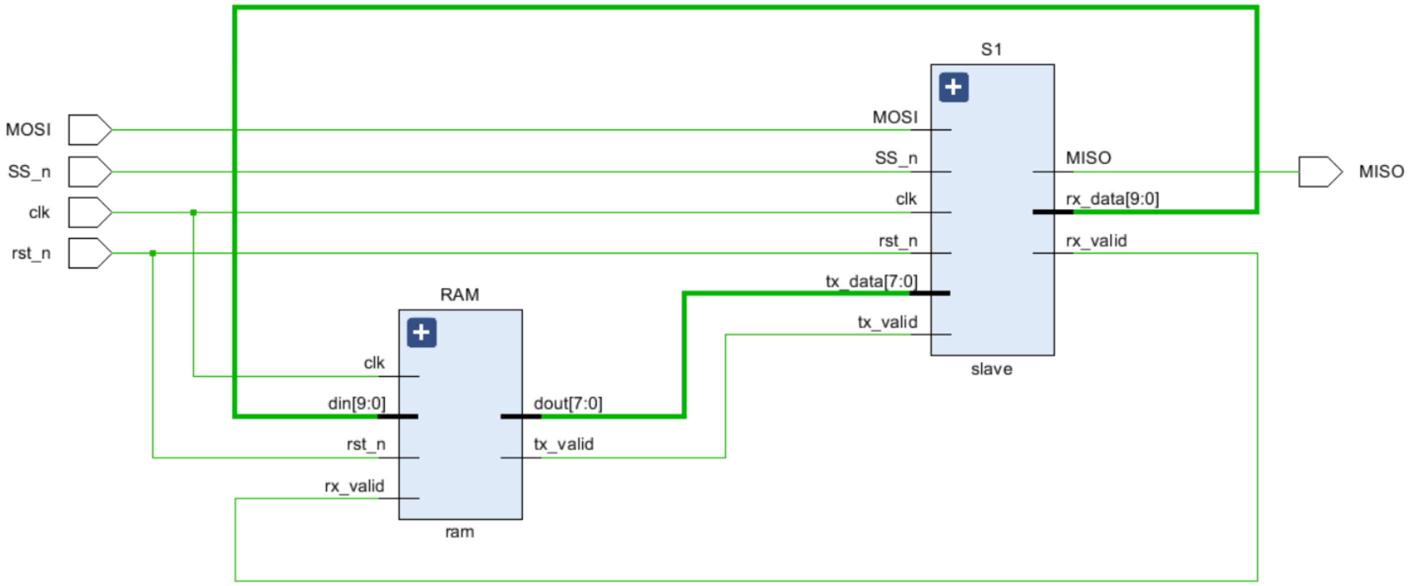


Figure 5-2-1: Elaborated Design

5.2.2 Synthesis Design

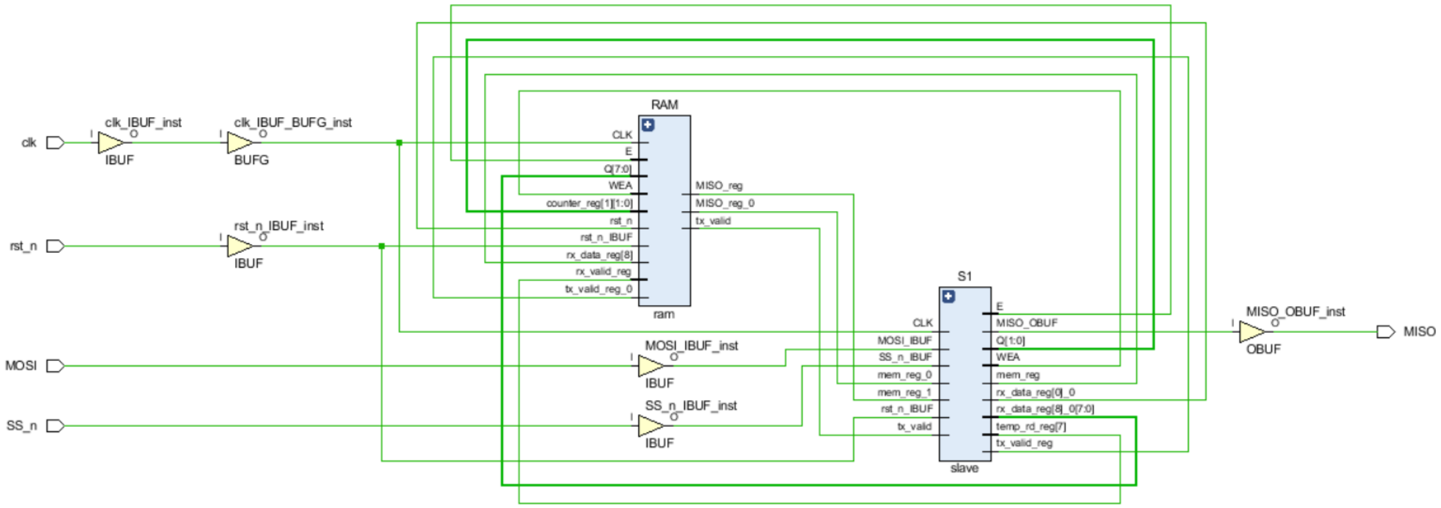


Figure 5-2-2: Synthesis Design

17-20

5.1.3. Implementation Design

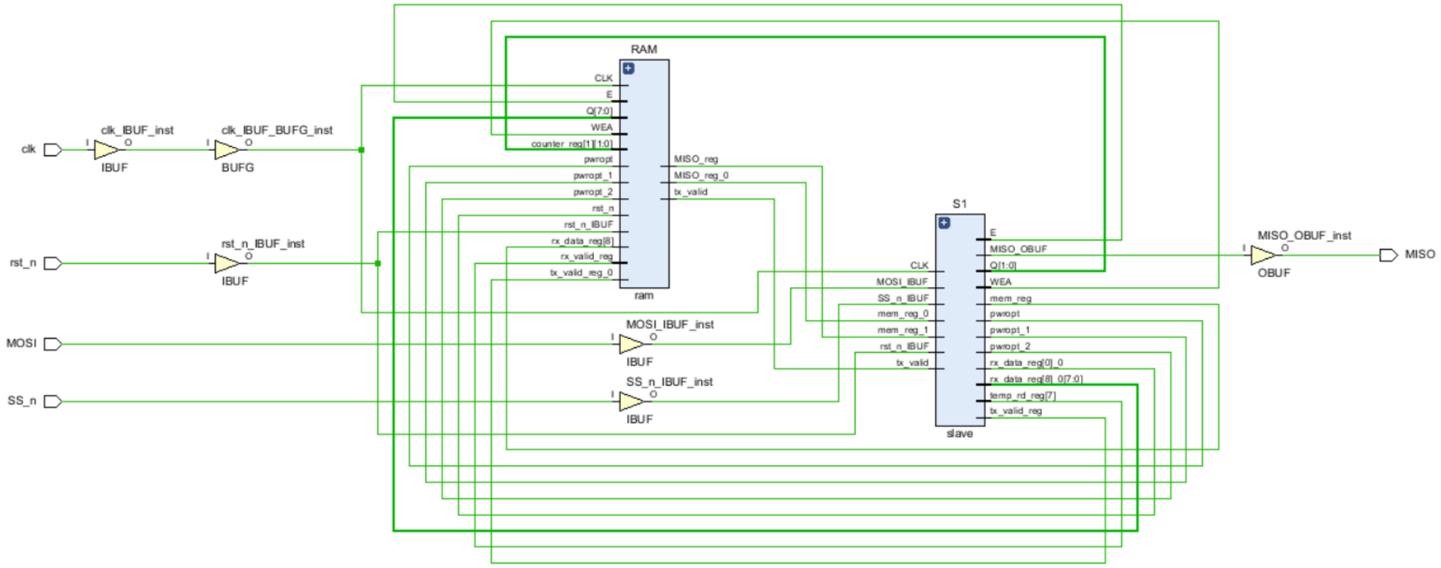


Figure 5-2-3-1: Implementation Design

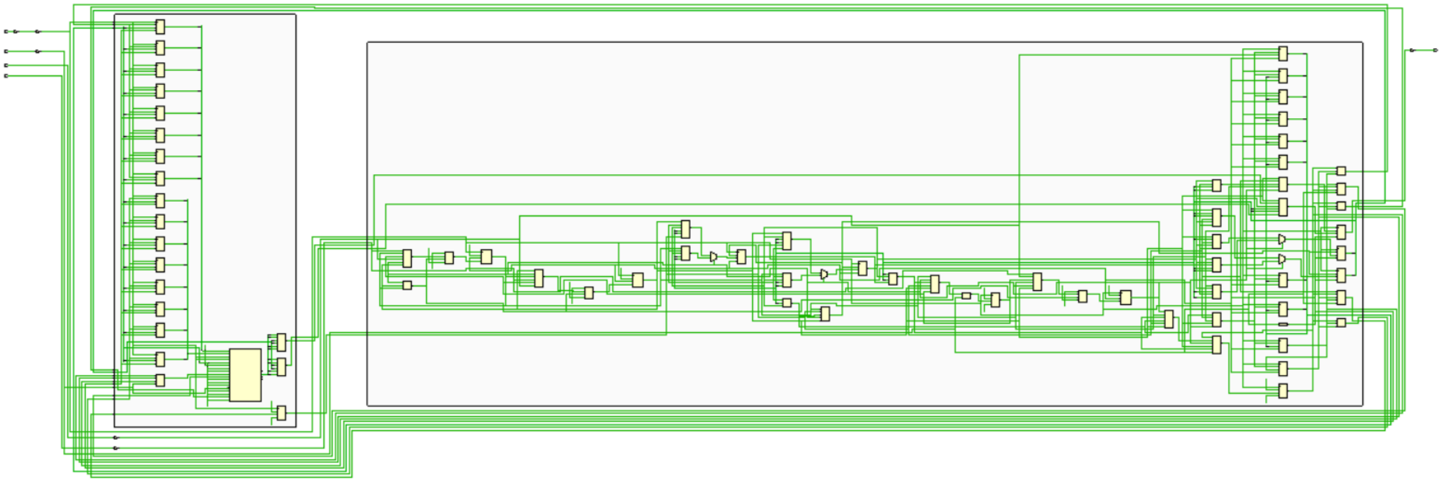
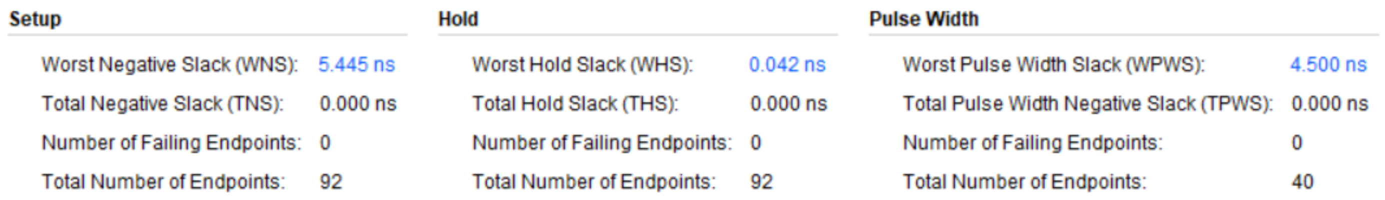


Figure 5-2-3-2: Implementation Design

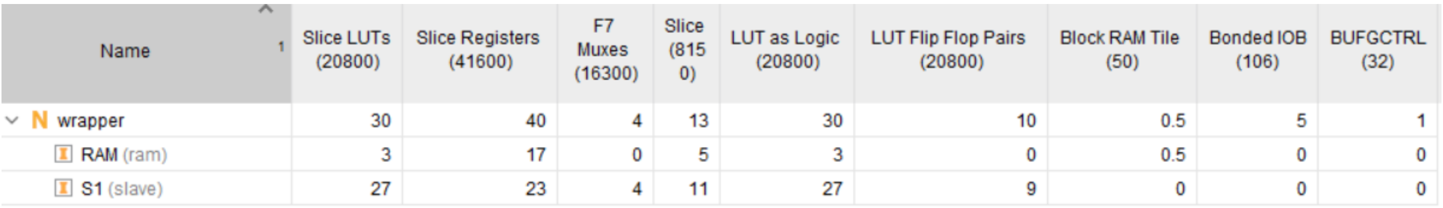
18-20

5.2.4 REPORT



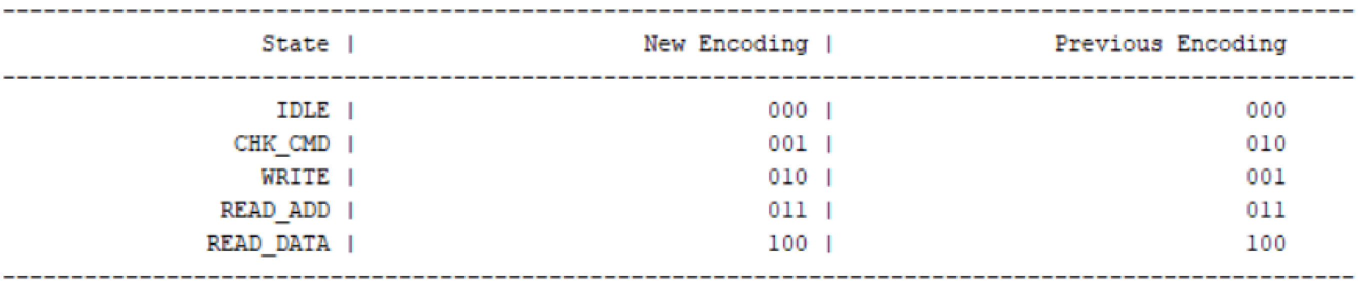
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.445 ns	Worst Hold Slack (WHS):	0.042 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	92	Total Number of Endpoints:	92	Total Number of Endpoints:	40

Figure 5-2-4-1: TIME



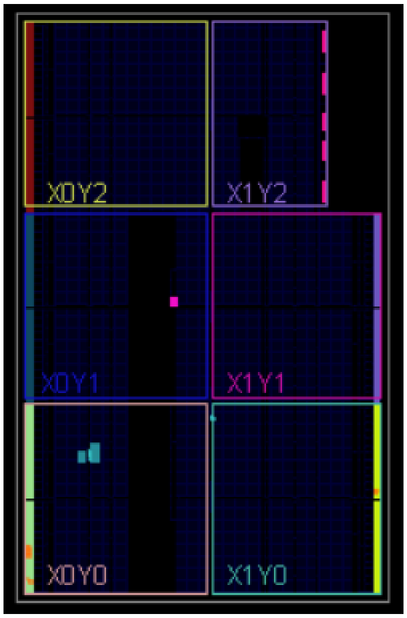
Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
wrapper	30	40	4	13	30	10	0.5	5	1
RAM (ram)	3	17	0	5	3	0	0.5	0	0
S1 (slave)	27	23	4	11	27	9	0	0	0

Figure 5.2.4.2: UTILIZATION



```
---------------------------------------------------------------------------------------------------
                  State |                     New Encoding |                 Previous Encoding
---------------------------------------------------------------------------------------------------
                   IDLE |                              000 |                               000
                CHK_CMD |                              001 |                               010
                  WRITE |                              010 |                               001
               READ_ADD |                              011 |                               011
              READ_DATA |                              100 |                               100
---------------------------------------------------------------------------------------------------
```

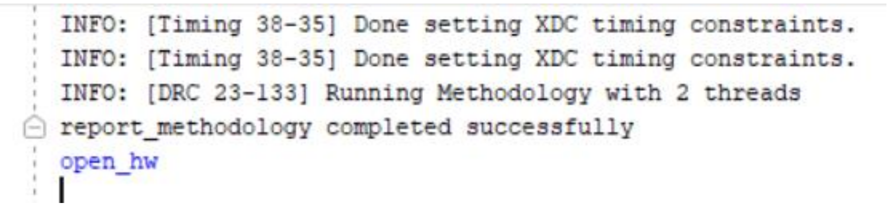
Figure 5.3.4.3: GRAY ENCODING



19-20

Figure 5.2.5 DEVICE

6 BITSTREAM CHECK

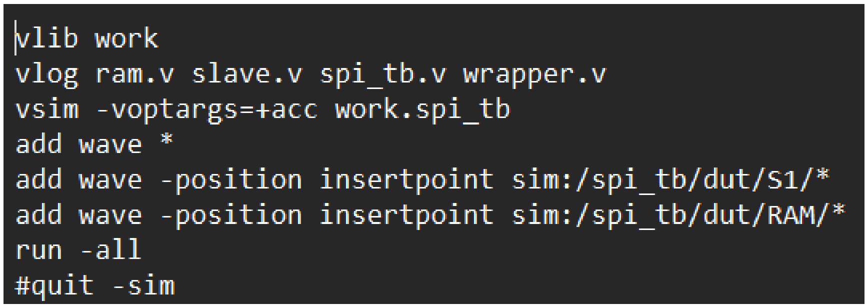


```
INFO: [Timing 38-35] Done setting XDC timing constraints.
INFO: [Timing 38-35] Done setting XDC timing constraints.
INFO: [DRC 23-133] Running Methodology with 2 threads
report_methodology completed successfully
open_hw
```

Figure 6 CHECK

7.ADDTION FILES

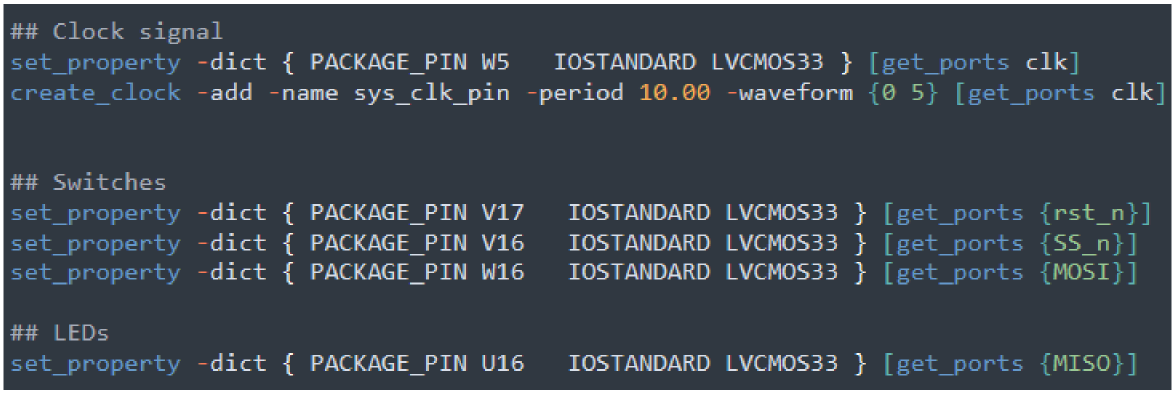
7.1 DO FILE



```
vlib work
vlog ram.v slave.v spi_tb.v wrapper.v
vsim -voptargs=+acc work.spi_tb
add wave *
add wave -position insertpoint sim:/spi_tb/dut/S1/*
add wave -position insertpoint sim:/spi_tb/dut/RAM/*
run -all
#quit -sim
```

Figure 7.1 DO FILE

7.2 Constraints



```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]


## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {rst_n}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {SS_n}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {MOSI}]

## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {MISO}]
```

Figure 7.2 CONSTRAINTS FILE