SPI SLAVE WITH SINGLE PORT RAM PROJECT

USING FPGA DESIGN FLOW

Marwan Khaled Mohamed

Contents

1 Project specifications

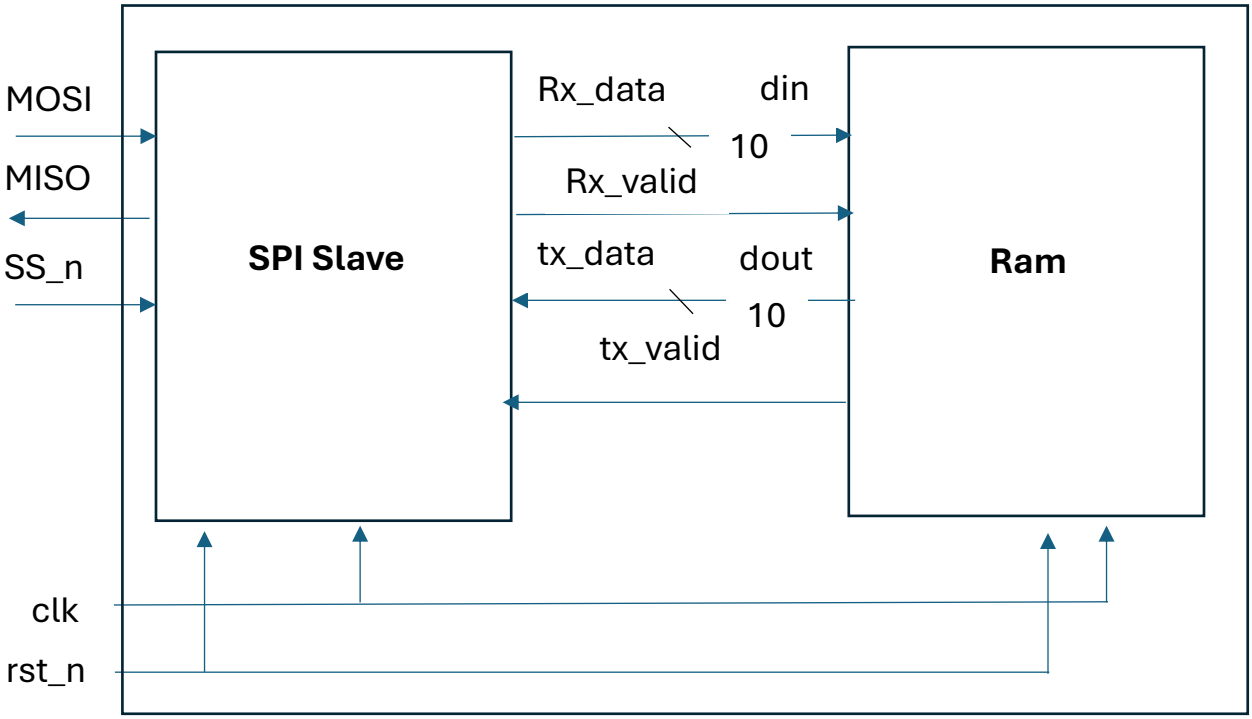

Figure 1: Block Diagram

2 Code

2.1 Design code

2.1.1 Spi code

```
1   module spi(clk,rstn,MISO,MOSI,SS_n,rx_data,rx_valid,tx_data,tx_valid);
2
3   input clk,rstn,MOSI,SS_n,tx_valid;
4   output reg [9:0] rx_data;
5   output reg MISO,rx_valid;
6   input [7:0] tx_data;
7   parameter IDLE=0;
8   parameter CHK_CMD=1;
9   parameter READ_DATA=2;
10  parameter READ_ADD=3;
11  parameter WRITE=4;
12
13  reg [2:0] cs,ns;
14  reg read_add_data;
15
16  reg [3:0] counter;
17  reg rst_counter;
18  reg [9:0] shift_reg;
19  reg dummy;
20
21  always@(SS_n,cs,MOSI)begin
22      case(cs)
23      IDLE:begin
24              if(SS_n==1)
25                  ns=IDLE;
26              else
27                  ns=CHK_CMD;
28
29          end
30      CHK_CMD:begin
31                  if(SS_n==1)
32                      ns=IDLE;
33                  else if(SS_n==0 & MOSI==0)
34                      ns=WRITE;
35                  else if(SS_n==0 & MOSI==1 )
36                      begin casex(read_add_data)
37                      0:ns=READ_ADD;
38                      1:ns=READ_DATA;
39                      1'bx:ns=READ_ADD;
40                      endcase
41                      end
42              end
43      READ_DATA:begin
44                  read_add_data=0;
45                  if(SS_n==0)
46                      ns=READ_DATA;
47                  else
48                      ns=IDLE;
49              end
50      READ_ADD:begin
51                  read_add_data=1;
```


Figure 2: SPi Code part1

```
50      READ_ADD:begin
51                  read_add_data=1;
52                  if(SS_n==0)
53                      ns=READ_ADD;
54                  else
55                      ns=IDLE;
56              end
57      WRITE:begin
58                  if(SS_n==0)
59                      ns=WRITE;
60                  else
61                      ns=IDLE;
62              end
63
64      endcase
65
66
67  end
68  always @(posedge clk or negedge rstn) begin
69      if (rstn==0) begin
70          cs<=IDLE;
71
72      end
73
74      else begin
75      cs<=ns;
76      end
77  end
78
79  always @(posedge clk)begin
80      if(cs==IDLE)
81      begin
82      rx_valid=0;
83      end
84
85
86      if(cs==CHK_CMD)
87      begin
88      counter=0;
89      shift_reg=0;
90      dummy=0;
91      end
92      case(cs)
93
94      WRITE:begin
95                  shift_reg[9-counter]=MOSI;
96                  counter=counter+1;
97                  if (counter==10)
98                      begin
99                      counter=0;
```

Figure 4: SPI Code Part2

```
94      WRITE:begin
95                  shift_reg[9-counter]=MOSI;
96                  counter=counter+1;
97                  if (counter==10)
98                      begin
99                      counter=0;
100                     rx_data=shift_reg;
101                     rx_valid=1;
102                     end
103             end
104     READ_ADD:begin
105                 shift_reg[9-counter]=MOSI;
106                 counter=counter+1;
107                 if (counter==10)
108                     begin
109                         counter=0;
110                         rx_data=shift_reg;
111                         rx_valid=1;
112                     end
113             end
114     READ_DATA:begin
115
116                     if(tx_valid==1)
117                     begin
118                     rx_valid=0;
119                     MISO=tx_data[7-counter];
120                     counter=counter+1;
121                     if (counter==8)
122                     counter=0;
123                     end
124
125
126                     if(dummy==0)
127                     begin
128
129                     shift_reg[9-counter]=MOSI;
130                     counter=counter+1;
131                     if (counter==10)
132                     begin
133                         counter=0;
134                         rx_data=shift_reg;
135                         rx_valid=1;
136                         dummy=1;
137                     end
138                     end
139
140             end
141     endcase
142 end
143 endmodule
```

Figure 3: SPI Code Part2

2.1.2 Ram Code

```
module RAM(din,rx_valid,tx_valid,clk,rstn,dout);
parameter MEM_WIDTH=8;
parameter MEM_DEPTH=256;
input clk,rstn,rx_valid;
input [9:0] din;
output reg [MEM_WIDTH-1:0] dout;
output reg tx_valid;
reg [7:0] address;
reg [MEM_WIDTH-1:0] mem [MEM_DEPTH-1:0];
always @(posedge clk ) begin
    if (!rstn) begin
        dout<=0;

    end
    else begin
            /* receive write address */
            if(rx_valid==1 && din[9:8]==2'b00 )
                address<=din[7:0];
            /* receive read address */
            else if ( rx_valid==1 && din[9:8]==2'b10)
                begin
                address<=din[7:0];
                tx_valid<=0;
                end
            /* receive data to be written in the hold address */
            else if (rx_valid==1 && din[9:8]==2'b01)
                mem[address]<=din[7:0];
            /* receive command to send data  */
            else if (rx_valid==1 && din[9:8]==2'b11)
                begin
                    dout<=mem[address];
                    tx_valid<=1;
                end

        end
end
endmodule
```

Figure 5: RAM Code

2.1.3 Top module code

```
module Spi_Ram(MOSI,MISO,SS_n,clk,rst_n);
input MOSI,clk,SS_n,rst_n;
output MISO;
wire [9:0] rx_data_internal;
wire rx_vaild_internal,tx_valid_internal;
wire [7:0] tx_data_internal;
spi DUT_spi(.clk(clk),.rstn(rst_n),.MISO(MISO),.MOSI(MOSI),.SS_n(SS_n),.rx_data(rx_data_internal),.rx_valid(rx_vaild_internal),.tx_data(tx_data_internal),.tx_valid(tx_valid_internal));
RAM DUT_RAM(.din(rx_data_internal),.rx_valid(rx_vaild_internal),.tx_valid(tx_valid_internal),.clk(clk),.rstn(rst_n),.dout(tx_data_internal));


endmodule
```

Figure 6: Top module Code

2.2 Testbench Code

```
module Spi_Ram_tb();

reg MOSI_tb,clk_tb,SS_n_tb,rstn_tb;
wire MISO_tb;
Spi_Ram DUT_tb (MOSI_tb,MISO_tb,SS_n_tb,clk_tb,rstn_tb);

initial begin
clk_tb=0;
forever
#4 clk_tb=~clk_tb;
end

/* test write address */
initial begin
$readmemh("mem.dat",DUT_tb.DUT_RAM.mem);

rstn_tb=0; MOSI_tb=0; SS_n_tb=1;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
//0011010101
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);

rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);

rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
/* return to idle */
rstn_tb=1; MOSI_tb=1; SS_n_tb=1;
@(negedge clk_tb);
```

Figure 7: Testbench Part1

```
@(negedge clk_tb);


/* test write data */
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
//01 10010101
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);

rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);

rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=1; SS_n_tb=1;
@(negedge clk_tb);
@(negedge clk_tb);

/*....... test read address where the data is 0 ..... */
rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
//1010000111
rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
@(negedge clk_tb);

rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
@(negedge clk_tb);
```

Figure 8: Testbench Part2

```
96   @(negedge clk_tb);
97   rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
98   @(negedge clk_tb);
99
100  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
101  @(negedge clk_tb);
102  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
103  @(negedge clk_tb);
104  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
105  @(negedge clk_tb);
106  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
107  @(negedge clk_tb);
108  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
109  @(negedge clk_tb);
110  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
111  @(negedge clk_tb);
112  rstn_tb=1; MOSI_tb=1; SS_n_tb=1;
113  @(negedge clk_tb);
114  @(negedge clk_tb);
115
116  /*..........test read data from zero place..... */
117  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
118  @(negedge clk_tb);
119  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
120  @(negedge clk_tb);
121  //1100000000
122  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
123  @(negedge clk_tb);
124  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
125  @(negedge clk_tb);
126
127  /* dummy data */
128  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
129  @(negedge clk_tb);
130  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
131  @(negedge clk_tb);
132  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
133  @(negedge clk_tb);
134  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
135  @(negedge clk_tb);
136  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
137  @(negedge clk_tb);
138  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
139  @(negedge clk_tb);
140  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
141  @(negedge clk_tb);
142  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
143  @(negedge clk_tb);
```

Figure 9: Testbench part3

```
143  @(negedge clk_tb);
144
145  @(negedge clk_tb); // another clock for ram to process the data because it's sequential
146
147  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
148  @(negedge clk_tb);
149  @(negedge clk_tb);
150  @(negedge clk_tb);
151  @(negedge clk_tb);
152  @(negedge clk_tb);
153  @(negedge clk_tb);
154  @(negedge clk_tb);
155  @(negedge clk_tb);
156
157  //return to idle
158  rstn_tb=1; MOSI_tb=0; SS_n_tb=1;
159  @(negedge clk_tb);
160
161  /*..................... test read address where the data is exists.......... */
162  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
163  @(negedge clk_tb);
164  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
165  @(negedge clk_tb);
166  //1011010101
167  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
168  @(negedge clk_tb);
169  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
170  @(negedge clk_tb);
171
172  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
173  @(negedge clk_tb);
174  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
175  @(negedge clk_tb);
176
177  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
178  @(negedge clk_tb);
179  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
180  @(negedge clk_tb);
181  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
182  @(negedge clk_tb);
183  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
184  @(negedge clk_tb);
185  rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
186  @(negedge clk_tb);
187  rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
188  @(negedge clk_tb);
189  /* return to idle */
```

Figure 10: Testbench part4

```
189   /* return to idle */
190   rstn_tb=1; MOSI_tb=1; SS_n_tb=1;
191   @(negedge clk_tb);
192   @(negedge clk_tb);
193
194   /*............test read data from the filled place......... */
195   rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
196   @(negedge clk_tb);
197   rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
198   @(negedge clk_tb);
199   //1100000000
200   rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
201   @(negedge clk_tb);
202   rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
203   @(negedge clk_tb);
204
205   /* dummy data */
206   rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
207   @(negedge clk_tb);
208   rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
209   @(negedge clk_tb);
210   rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
211   @(negedge clk_tb);
212   rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
213   @(negedge clk_tb);
214   rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
215   @(negedge clk_tb);
216   rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
217   @(negedge clk_tb);
218   rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
219   @(negedge clk_tb);
220   rstn_tb=1; MOSI_tb=0; SS_n_tb=0;
221   @(negedge clk_tb);
222
223   @(negedge clk_tb); // another clock for ram to process the data because it's sequential
224
225   rstn_tb=1; MOSI_tb=1; SS_n_tb=0;
226   @(negedge clk_tb);
227   @(negedge clk_tb);
228   @(negedge clk_tb);
229   @(negedge clk_tb);
230   @(negedge clk_tb);
231   @(negedge clk_tb);
232   @(negedge clk_tb);
233   @(negedge clk_tb);
234
235   //return to idle
```

Figure 11: Testbench part5

```
235   //return to idle
236   rstn_tb=1; MOSI_tb=0; SS_n_tb=1;
237   @(negedge clk_tb);
238
239
240   $stop;
241   end
242
243   endmodule
```

Figure 12: Testbench part6

3 Constraints file

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33} [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]


## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {MOSI}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {SS_n}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {sw[2]}]



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


##Buttons
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports rst_n]


## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```

Figure 13: constraints File

4 Waveform

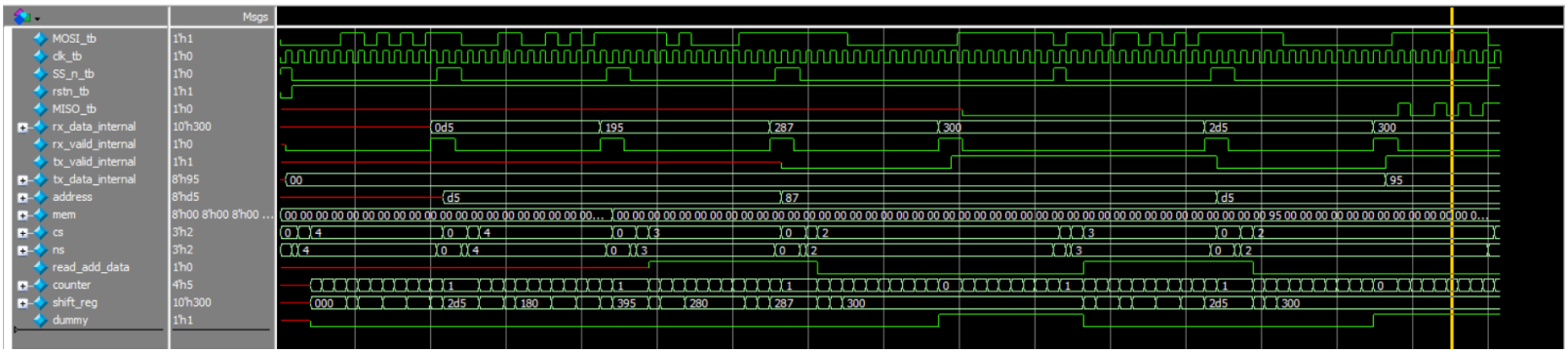

Figure 14: Waveform for entire Signals

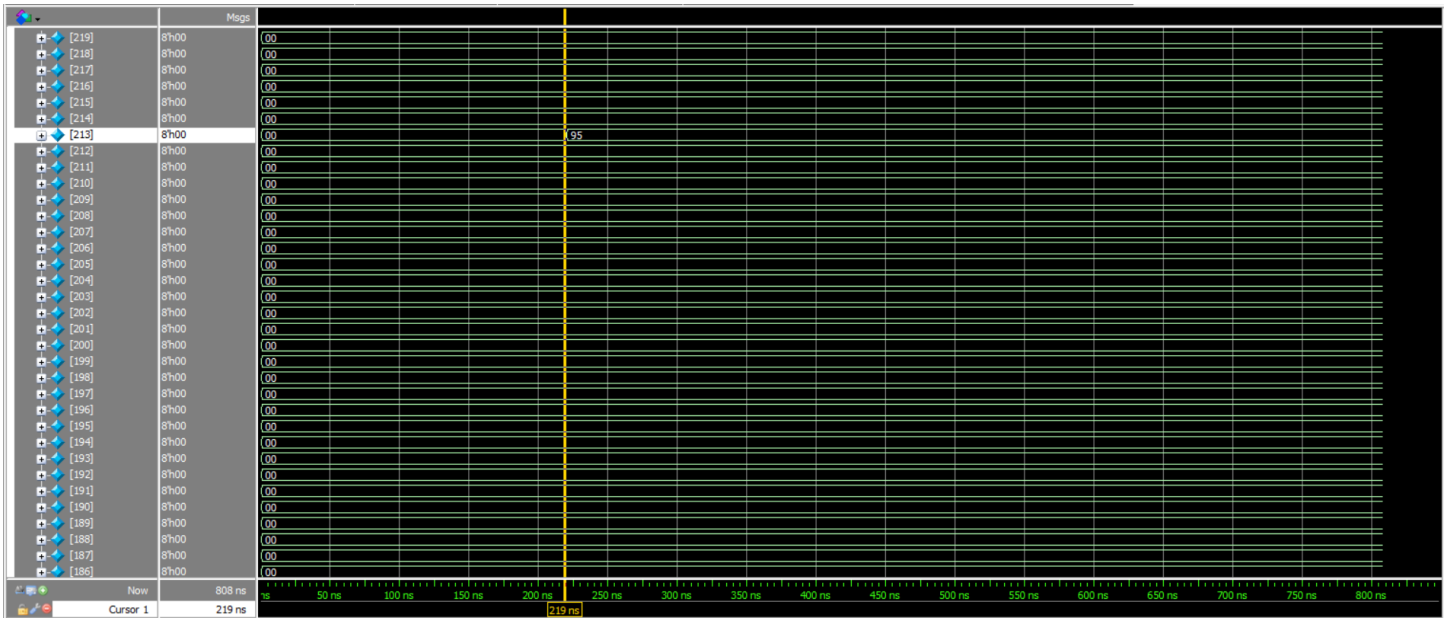

Figure 15: waveform for Memory Content

5 Do file

```
1  vlib work
2  vlog spi.v RAM.v Spi_Ram.v Spi_Ram_tb.v
3  vsim -voptargs=+acc work.Spi_Ram_tb
4  add wave *
5  run -all
6  #quit -sim
```

Figure 16: Do File

6 Vivado

6.1 Elaboration

6.1.1 Schematic

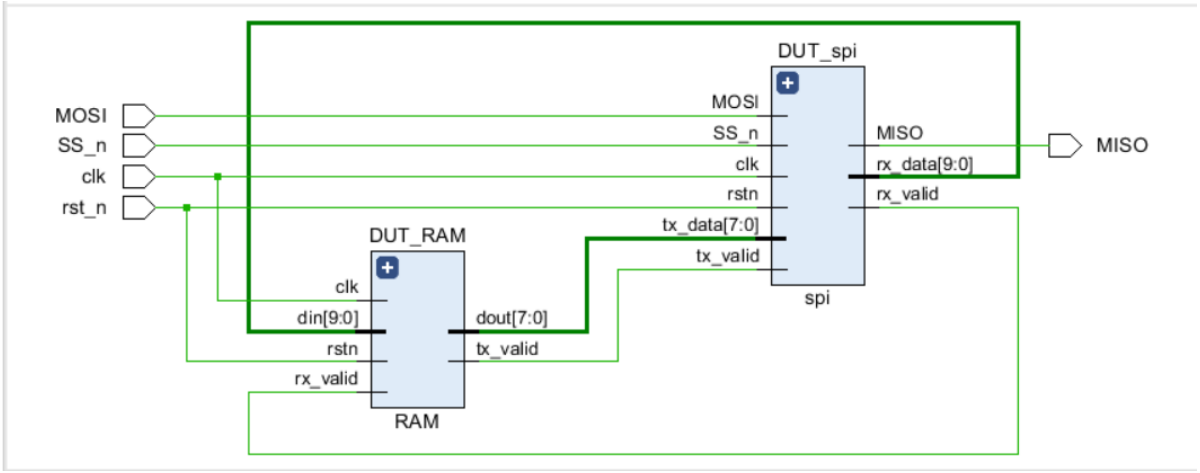

Figure 17: Elaboration schematic

6.1.2 Message

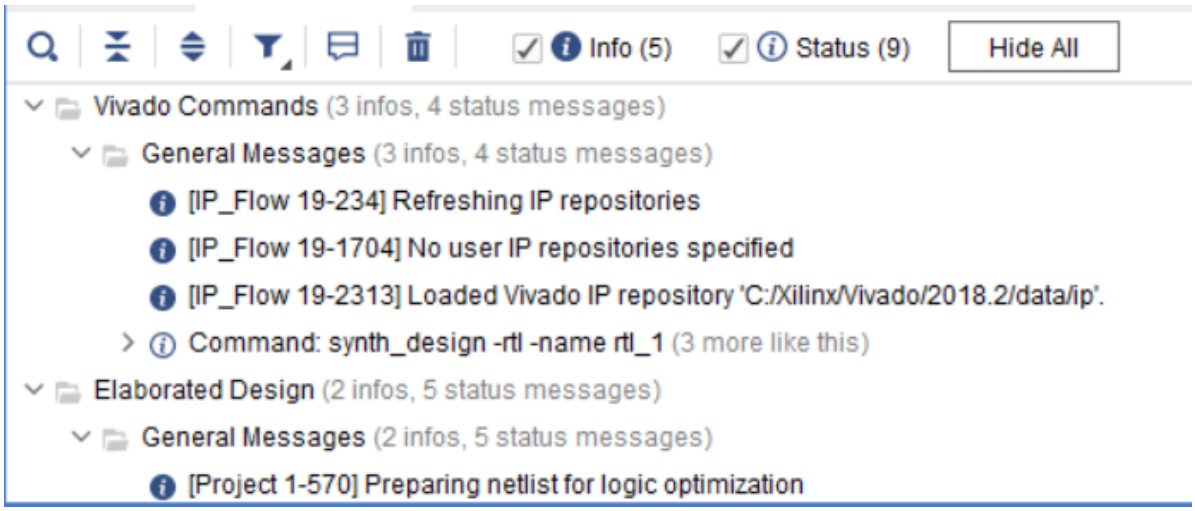

Figure 18: Elaboration Message

6.2 Synthesis

6.2.1 Schematic

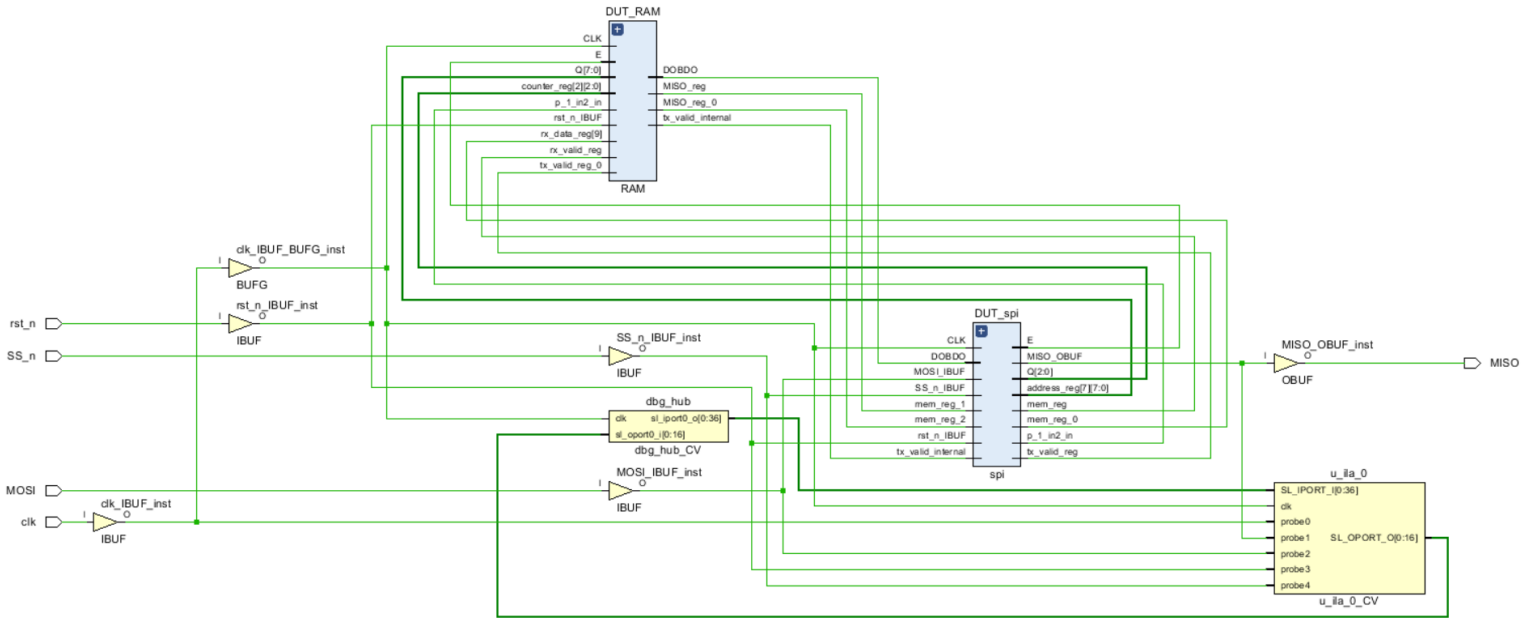

Figure 19: Schematic in Synthesis stage

6.2.2 Utilization report

Name	Slice LUTs (134600)	Slice Registers (269200)	Block RAM Tile (365)	Bonded IOB (500)	BUFGCTRL (32)
N Spi_Ram	80	43	0.5	5	1
dbg_hub (dbg_hub_CV)	0	0	0	0	0
DUT_RAM (RAM)	2	9	0.5	0	0
DUT_spi (spi)	78	34	0	0	0
u_ila_0 (u_ila_0_CV)	0	0	0	0	0

Figure 20: Utilization report in Synthesis stage

6.2.3 Timing report

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.937 ns	Worst Hold Slack (WHS):	0.147 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:	82	Total Number of Endpoints:	82	Total Number of Endpoints:	42

All user specified timing constraints are met.

Figure 21: time report in Synthesis stage

6.2.4 Message Tab

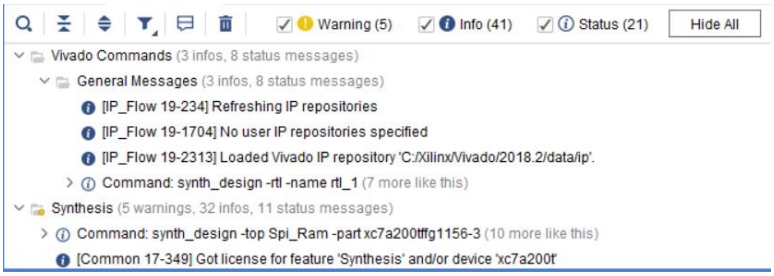

Figure 22 : Message Tab in Synthesis stage

6.3 implementation

6.3.1 Device

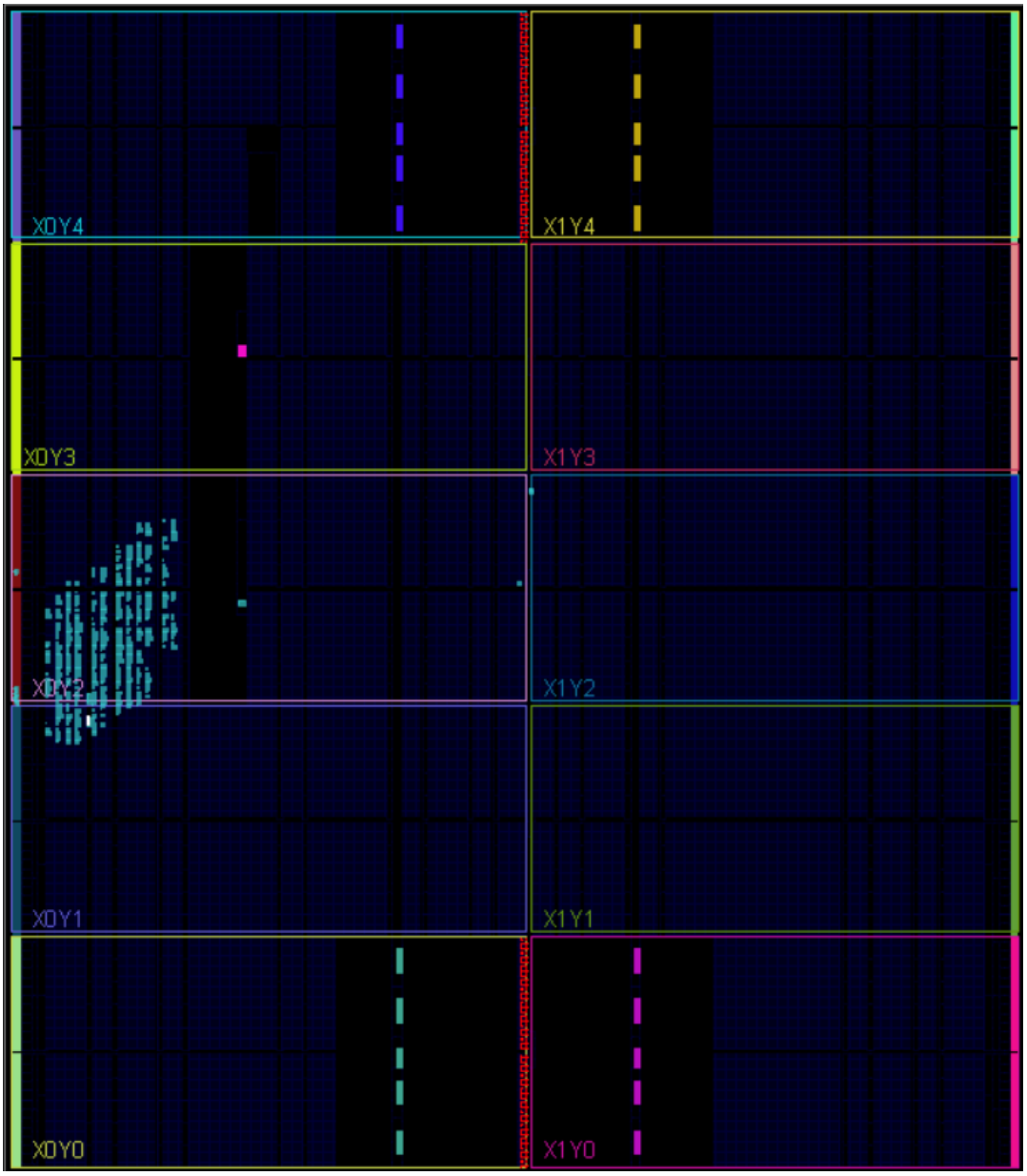

Figure 23: Device

6.3.2 Utilization report

Name	Slice LUTs (133800)	Slice Registers (267600)	F7 Muxes (66900)	Slice (33450)	LUT as Logic (133800)	LUT as Memory (46200)	LUT Flip Flop Pairs (133800)	Block RAM Tile (365)	Bonded IOB (500)	BUFGCTRL (32)	BSCANE2 (4)
Spi_Ram	1319	1959	10	636	1211	108	756	1	5	2	1
dbg_hub (dbg_hub)	475	727	0	232	451	24	306	0	0	1	1
DUT_RAM (RAM)	2	9	0	4	2	0	0	0.5	0	0	0
DUT_spi (spi)	78	34	0	26	78	0	31	0	0	0	0
u_ila_0 (u_ila_0)	764	1189	10	380	680	84	418	0.5	0	0	0

Figure 24: Utilization report in implementation stage

6.3.3 Timing report

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.773 ns	Worst Hold Slack (WHS):	0.073 ns	Worst Pulse Width Slack (WPWS):	3.950 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:	3867	Total Number of Endpoints:	3851	Total Number of Endpoints:	2142

All user specified timing constraints are met.

Figure 25: Timing Report in implementation Stage

6.3.4 Message Tab

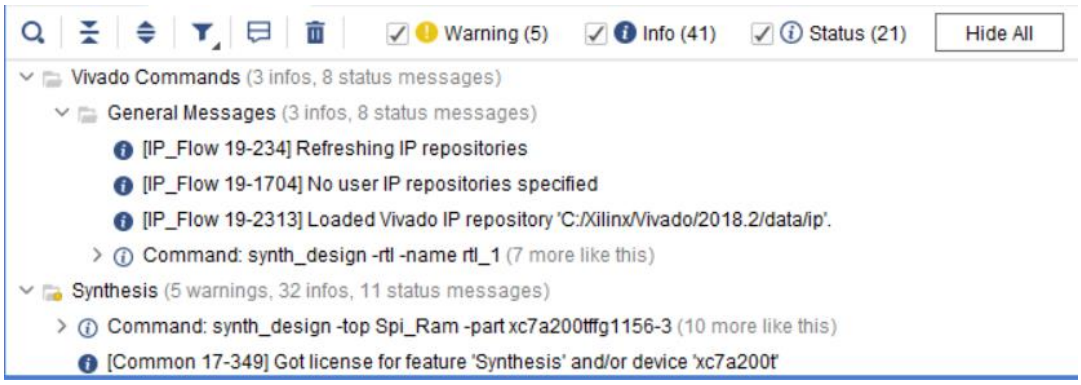

Figure 26:Message Tab in implementation Stage