

# **Digital Design**

## **SPI With RAM**

By: **Hassan Khaled**

# Design Report

spi_wrapper Project Status (02/27/2023 - 04:02:34)			
Project File:	spi_modules.xise	Parser Errors:	No Errors
Module Name:	spi_wrapper	Implementation State:	Synthesized
Target Device:	xc7a100t-3csg324	• Errors:	No Errors
Product Version:	ISE 14.7	• Warnings:	<a href="#">10 Warnings</a>
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	<a href="#">Xilinx Default (unlocked)</a>	• Timing Constraints:	
Environment:	<a href="#">System Settings</a>	• Final Timing Score:	

Device Utilization Summary (estimated values)				<a href="#">[-]</a>
Logic Utilization	Used	Available	Utilization	
Number of Slice Registers	64	126800	0%	
Number of Slice LUTs	87	63400	0%	
Number of fully used LUT-FF pairs	60	91	65%	
Number of bonded IOBs	5	210	2%	
Number of Block RAM/FIFO	1	135	0%	
Number of BUFG/BUFGCTRLs	1	32	3%	

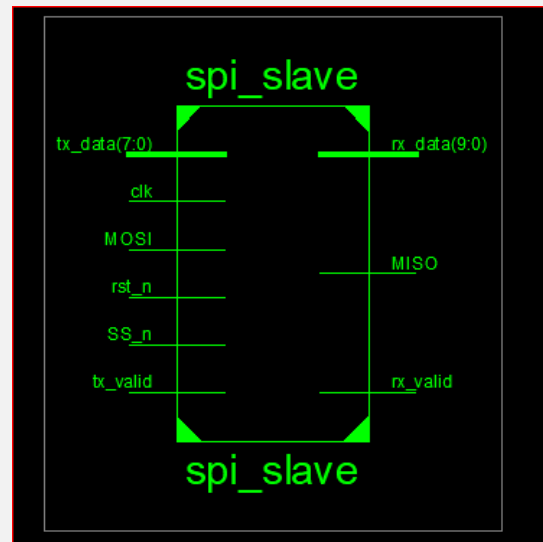
Detailed Reports						<a href="#">[-]</a>
Report Name	Status	Generated	Errors	Warnings	Infos	
<a href="#">Synthesis Report</a>	Current	Mon Feb 27 04:02:31 2023	0	<a href="#">10 Warnings (6 new)</a>	<a href="#">6 Infos (6 new)</a>	
Translation Report						
Map Report						
Place and Route Report						
Power Report						
Post-PAR Static Timing Report						
Bitgen Report						

Secondary Reports			<a href="#">[-]</a>
Report Name	Status	Generated	

Date Generated: 02/27/2023 - 04:21:59

# SPI Slave

## • Block Diagram



## • Design code

```

1  module spi_slave(
2
3  input          clk          ,
4  input          rst_n       ,
5  input          MOSI        ,
6  input          SS_n        ,
7  input          tx_valid    ,
8  input [7:0]    tx_data     ,
9
10 output reg     MISO        ,
11 output reg     rx_valid    ,
12 output reg [9:0] rx_data   ,
13 );
14
15 reg [2:0] current_state, next_state;
16
17 parameter IDLE      = 3'b000;
18 parameter CHK_CMD   = 3'b001;
19 parameter WRITE     = 3'b010;
20 parameter READ_ADD  = 3'b011;
21 parameter READ_DATA = 3'b100;
22 parameter READ_SPI  = 3'b101;
23
24 reg [3:0] counter;
25 reg add_sent , read_flag;
26 reg [7:0] temp_reg;
27
28
29
30 //-----
31
32
33 always @(posedge clk , negedge rst_n)begin
34
35     if(!rst_n)begin
36         current_state <= IDLE;
37     end
38
39     else
40         current_state <= next_state ;
41     end
42
43 //-----

```

```

43 //-----
44
45 always @(*)begin
46
47     case (current_state)
48
49     IDLE : begin
50         if(SS_n) begin
51             next_state = IDLE;
52         end
53
54         else begin
55             next_state = CHK_CMD;
56         end
57     end
58
59
60
61     CHK_CMD: begin
62         if(SS_n) begin
63             next_state = IDLE ;
64         end
65
66         else if(!SS_n && !MOSI)begin
67             next_state = WRITE ;
68         end
69
70         else begin
71             if(add_sent)
72                 next_state = READ_DATA ;
73             else
74                 next_state = READ_ADD ;
75         end
76     end
77
78
79
80     WRITE : begin
81         if(SS_n) begin
82             next_state = IDLE ;
83         end
84
85         else begin
86             if(counter != 10)begin
87                 next_state = WRITE;
88             end
89
90             else
91                 next_state = CHK_CMD;
92         end
93     end

```

```

79
80 WRITE : begin
81     if(SS_n) begin
82         next_state = IDLE ;
83     end
84
85     else begin
86         if(counter != 10)begin
87             next_state = WRITE;
88         end
89
90         else
91             next_state = CHK_CMD;
92     end
93 end
94
95 READ_ADD :begin
96     if(SS_n) begin
97         next_state = IDLE ;
98     end
99
100
101     else begin
102
103         if(counter != 'd10')begin
104             next_state = READ_ADD;
105         end
106
107         else begin
108             next_state = CHK_CMD;
109         end
110     end
111 end
112
113 READ_DATA :begin
114     if(SS_n) begin
115         next_state = IDLE ;
116     end
117
118     else begin
119         next_state = READ_SPI;
120     end
121 end
122
123 READ_SPI :begin
124     if(counter!=10)
125         next_state = READ_SPI;
126     else
127         next_state = CHK_CMD;
128     end
129 endcase
130 end

```

```

133 always @(posedge clk)begin
134
135     case (current_state)
136
137     IDLE : begin
138         MISO      <= 1'b0      ;
139         rx_valid  <= 1'b0      ;
140         rx_data   <= 0         ;
141         counter   <= 0         ;
142         add_sent  <= 1'b0      ;
143         read_flag <= 1'b0      ;
144         temp_reg  <= 1'b0      ;
145     end
146
147
148     WRITE : begin
149         if(counter !=10)begin
150             rx_data[counter] <= MOSI;
151             counter <= counter + 1;
152         end
153
154         else begin
155             counter <= 0;
156             rx_valid <= 1'b1;
157         end
158     end
159
160     READ_ADD: begin
161         if(counter !=10)begin
162             rx_data[counter] <= MOSI;
163             counter <= counter + 1;
164         end
165
166         else begin
167             counter <= 0;
168             rx_valid <= 1'b1;
169             add_sent <= 1'b1;
170         end
171     end
172
173     READ_DATA: begin
174         add_sent <= 1'b0;
175         if(tx_valid || !read_flag) begin
176             temp_reg <= tx_data;
177             read_flag <= 1'b1;
178         end
179     end
180
181     READ_SPI: begin
182         if(counter != 'd10')begin
183             MISO <= temp_reg[counter];
184             counter <= counter + 1;
185         end
186     end
187 end

```

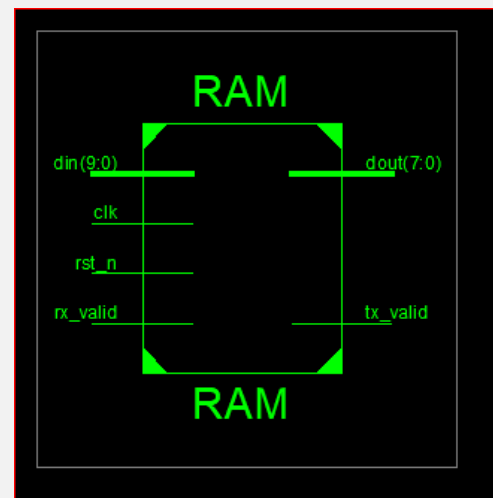
```

188
189     end
190
191     READ_SPI: begin
192         if(counter != 'd10')begin
193             MISO <= temp_reg[counter];
194             counter <= counter + 1;
195         end
196     end
197
198 endcase
199 end
200
201 endmodule
202
203

```

# RAM

## . Block Diagram



## . Design Code

```

1  module RAM #(parameter MEM_DEPTH = 'd256 , parameter ADDR_SIZE = 'd8)
2  (
3      input      clk,
4      input      rst_n,
5      input [9:0] din,
6      input      rx_valid,
7
8      output reg [7:0] dout,
9      output reg      tx_valid);
10
11     reg [2:0] current_state,next_state;
12
13     parameter IDLE =      3'b000;
14     parameter WRITE_ADDR = 3'b001;
15     parameter WRITE_DATA = 3'b010;
16     parameter READ_ADDR  = 3'b011;
17     parameter READ_DATA  = 3'b100;
18
19
20     reg [ADDR_SIZE-1:0] temp_address, read_address ,write_address;
21     reg flag , h_flag;
22
23
24     reg [7:0] ram_mem [MEM_DEPTH-1 : 0];
25
26
27     always @(posedge clk or negedge rst_n) begin
28         if (~rst_n) begin
29             current_state <= IDLE;
30         end
31         else
32             current_state <= next_state;
33     end
34
35
36
37
38     always @(*) begin
39
40         case(current_state)
41
42             IDLE: begin
43                 if(rx_valid)begin
44
45                     if (!din[9]) begin
46                         next_state = WRITE_ADDR;
47                     end
48
49                     else begin
50                         next_state = READ_ADDR;
51                     end
52                 end
53
54             else
55                 next_state = IDLE;
56             end
57
58
59             WRITE_ADDR: begin
60
61                 if(din[9:8] == 2'b01)begin
62                     next_state = WRITE_DATA;
63                 end
64
65                 end
66
67             WRITE_DATA: begin
68
69                 next_state = IDLE;
70                 end
71
72             READ_ADDR : begin
73                 if(din[9:8] == 2'b11)
74                     next_state = READ_DATA;
75
76                 else begin
77                     next_state = READ_ADDR;
78                 end
79             end
80
81             READ_DATA: begin
82
83                 next_state = IDLE;
84                 end
85
86         endcase
87     end

```

```

37     always @(*) begin
38
39         case(current_state)
40
41             IDLE: begin
42                 if(rx_valid)begin
43
44                     if (!din[9]) begin
45                         next_state = WRITE_ADDR;
46                     end
47
48                     else begin
49                         next_state = READ_ADDR;
50                     end
51                 end
52
53             else
54                 next_state = IDLE;
55             end
56
57
58             WRITE_ADDR: begin
59
60                 if(din[9:8] == 2'b01)begin
61                     next_state = WRITE_DATA;
62                 end
63
64                 end
65
66             WRITE_DATA: begin
67
68                 next_state = IDLE;
69                 end
70
71             READ_ADDR : begin
72                 if(din[9:8] == 2'b11)
73                     next_state = READ_DATA;
74
75                 else begin
76                     next_state = READ_ADDR;
77                 end
78             end
79
80             READ_DATA: begin
81
82                 next_state = IDLE;
83                 end
84
85         endcase
86     end
87

```

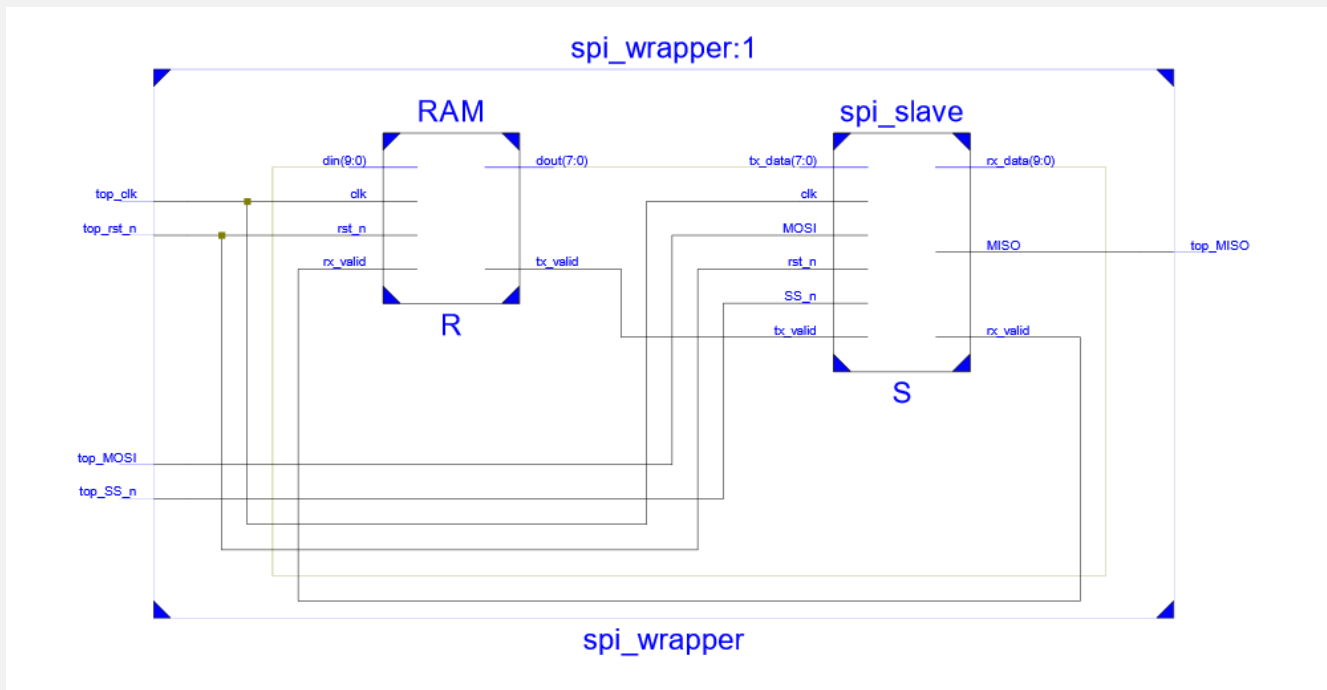
```

88     always @(posedge clk) begin
89         case(current_state)
90         IDLE: begin
91             dout      <= 8'b0;
92             tx_valid <= 1'b0;
93             temp_address <= 0;
94             read_address <= 0;
95             write_address <= 0;
96             flag <= 0;
97             h_flag <= 0;
98             end
99         WRITE_ADDR: begin
100             if(din[9:8] == 2'b00 && flag == 1'b0)begin
101                 write_address <= din[7:0];
102                 flag <= 1'b1;
103             end
104         end
105         WRITE_DATA: begin
106             ram_mem[write_address] <= din[7:0];
107         end
108         READ_ADDR: begin
109             if(din[9:8] == 2'b10 && flag == 1'b0)begin
110                 read_address <= din[7:0] ;
111                 flag <= 1;
112             end
113         end
114         READ_DATA: begin
115             if(!h_flag) begin
116                 tx_valid <= 1'b1;
117                 dout <= ram_mem[read_address];
118                 h_flag <= 1'b1;
119             end
120         end
121     endcase
122 end
123 endmodule

```

# SPI WRAPPER

## • Block Diagram



## • Design Code

```
1
2 module spi_wrapper #(parameter MEM_DEPTH = 'd256 , parameter ADDR_SIZE = 'ds)
3     input top_clk,
4     input top_rst_n,
5     input top_MOSI,
6     input top_SS_n,
7
8     output top_MISO
9 );
10
11 wire top_tx_valid, top_rx_valid ;
12 wire [7:0] top_tx_data;
13 wire [9:0] top_rx_data;
14
15
16 spi_slave S(.clk(top_clk) ,.rst_n(top_rst_n) , .MOSI(top_MOSI) , .SS_n(top_SS_n) ,
17             .tx_valid(top_tx_valid) , .tx_data(top_tx_data) , .rx_valid(top_rx_valid) , .rx_data(top_rx_data) , .MISO(top_MISO));
18
19
20
21
22 RAM #(MEM_DEPTH(MEM_DEPTH), ADDR_SIZE(ADDR_SIZE)) R(.clk(top_clk),.rst_n(top_rst_n),
23              .rx_valid(top_rx_valid), .din(top_rx_data),.dout(top_tx_data),.tx_valid(top_tx_valid));
24
25
26
27
28 endmodule
```

## . Testbench Code

```
1  module spi_wrapper_tb();
2
3  parameter top_MEM_DEPTH = 256;
4  parameter top_ADDR_SIZE = 8;
5
6  reg top_clk_tb, top_rst_n_tb, top_MOSI_tb, top_SS_n_tb ;
7  wire top_MISO_tb;
8
9
10 spi_wrapper #(.MEM_DEPTH(top_MEM_DEPTH) ,.ADDR_SIZE(top_ADDR_SIZE)) SW (.top_clk(top_clk_tb) ,
11 | | | | | | | | .top_rst_n(top_rst_n_tb) , .top_MOSI(top_MOSI_tb), .top_SS_n(top_SS_n_tb) ,.top_MISO(top_MISO_tb));
12
13 integer i;
14
15 initial begin
16 top_clk_tb = 0;
17
18 forever begin
19 #1 top_clk_tb = ~ top_clk_tb;
20 end
21 end
22
23
24
25 initial begin
26 $readmemb ("mem.dat.txt",SW.R.ram_mem);
27 end
28
29
30 initial begin
31 top_rst_n_tb = 0;
32 #3;
33 top_rst_n_tb = 1;
34 end
35
```

```
37 initial begin
38
39 // Address for the write operation (0011000110) 198
40
41 top_SS_n_tb = 1'b0;
42 top_MOSI_tb = 1'b0;
43 #8;
44 top_MOSI_tb = 1'b0;
45 #2;
46 top_MOSI_tb = 1'b1;
47 #2;
48 top_MOSI_tb = 1'b1;
49 #2;
50 top_MOSI_tb = 1'b0; //0010101010 location 170
51 #2;
52 top_MOSI_tb = 1'b0;
53 #2;
54 top_MOSI_tb = 1'b0;
55 #2;
56 top_MOSI_tb = 1'b1;
57 #2;
58 top_MOSI_tb = 1'b1;
59 #2;
60 top_MOSI_tb = 1'b0;
61 #2;
62 top_MOSI_tb = 1'b0;
63 #6;
64
65 //----- Write data 10001111 in location 198 -----
66
```



```

65 //----- Write data 10001111 in location 198 -----
66
67 top_MOSI_tb = 1'b1;
68 #2;
69 top_MOSI_tb = 1'b1;
70 #2;
71 top_MOSI_tb = 1'b1;
72 #2;
73 top_MOSI_tb = 1'b1;
74 #2;
75 top_MOSI_tb = 1'b0;
76 #2;
77 top_MOSI_tb = 1'b0;
78 #2;
79 top_MOSI_tb = 1'b0;
80 #2;
81 top_MOSI_tb = 1'b1;
82 #2;
83 top_MOSI_tb = 1'b1;
84 #2;
85 top_MOSI_tb = 1'b0;
86 #1;
87
88 //----- Read from location 190 -----
89
90 top_MOSI_tb = 1'b1;
91 #6;
92 top_MOSI_tb = 1'b0;
93 #2;
94 top_MOSI_tb = 1'b1;
95 #2;
96 top_MOSI_tb = 1'b1;           //1011000110      location priveously previously filled in write op
97 #2;
98 top_MOSI_tb = 1'b0;
99 #2;
100 top_MOSI_tb = 1'b0;
101 #2;
102 top_MOSI_tb = 1'b0;
103 #2;
104 top_MOSI_tb = 1'b1;
105 #2;
106 top_MOSI_tb = 1'b1;
107 #2;
108 top_MOSI_tb = 1'b0;
109 #2;
110 top_MOSI_tb = 1'b1;
111

```

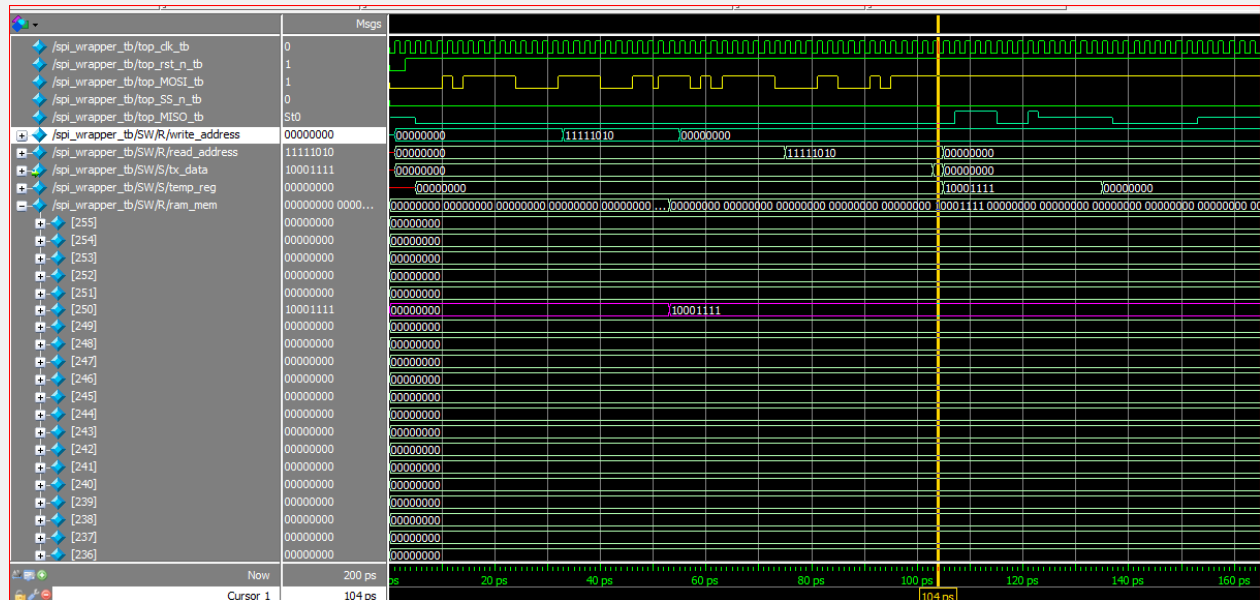
```

112
113 //-----Read Data-----
114
115 top_MOSI_tb = 1'b0;
116 #6;
117 top_MOSI_tb = 1'b1;
118 #2;
119 top_MOSI_tb = 1'b1;
120 #2;
121 top_MOSI_tb = 1'b0;
122 #2;
123 top_MOSI_tb = 1'b0;
124 #2;
125 top_MOSI_tb = 1'b0;
126 #2;
127 top_MOSI_tb = 1'b1;
128 #2;
129 top_MOSI_tb = 1'b0;
130 #2;
131 top_MOSI_tb = 1'b1;
132 #2;
133 top_MOSI_tb = 1'b1;
134 #6;
135
136 end
137 endmodule
138

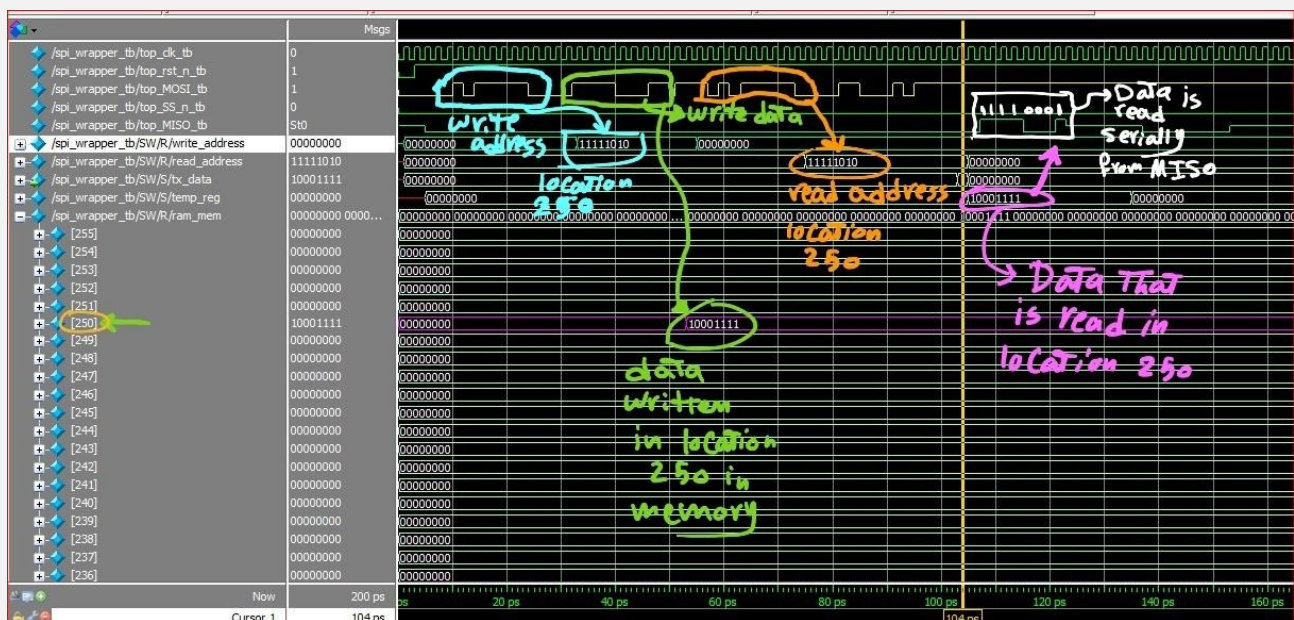
```

## • Waveform

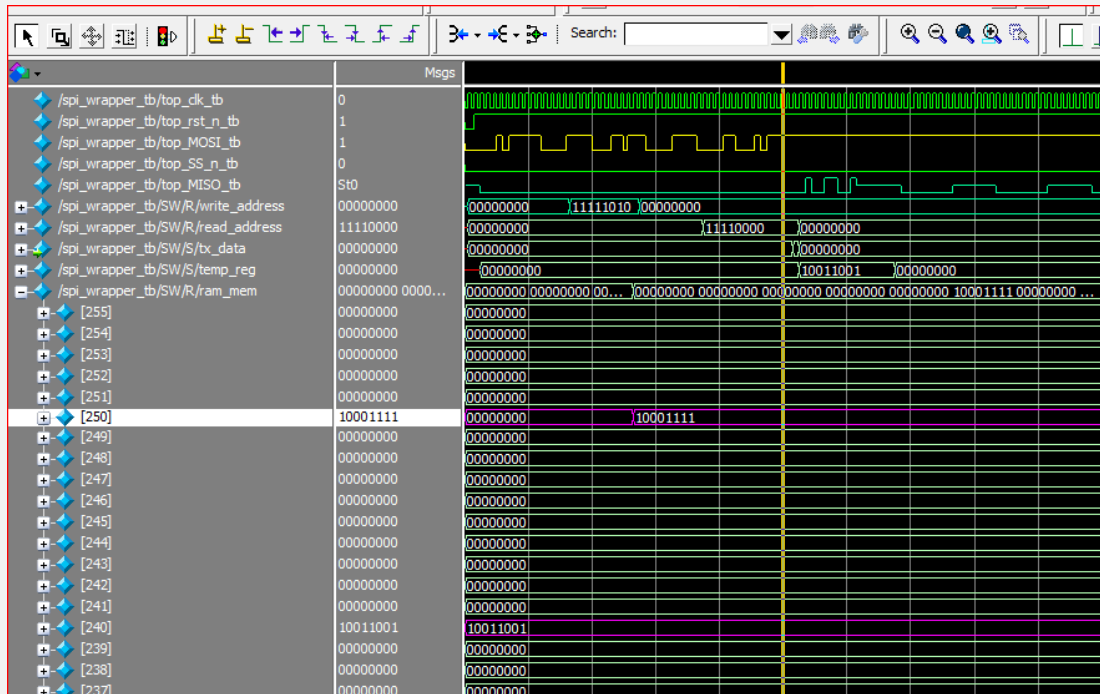
- Test Case 1 -> Write in location 250 then read from the same location



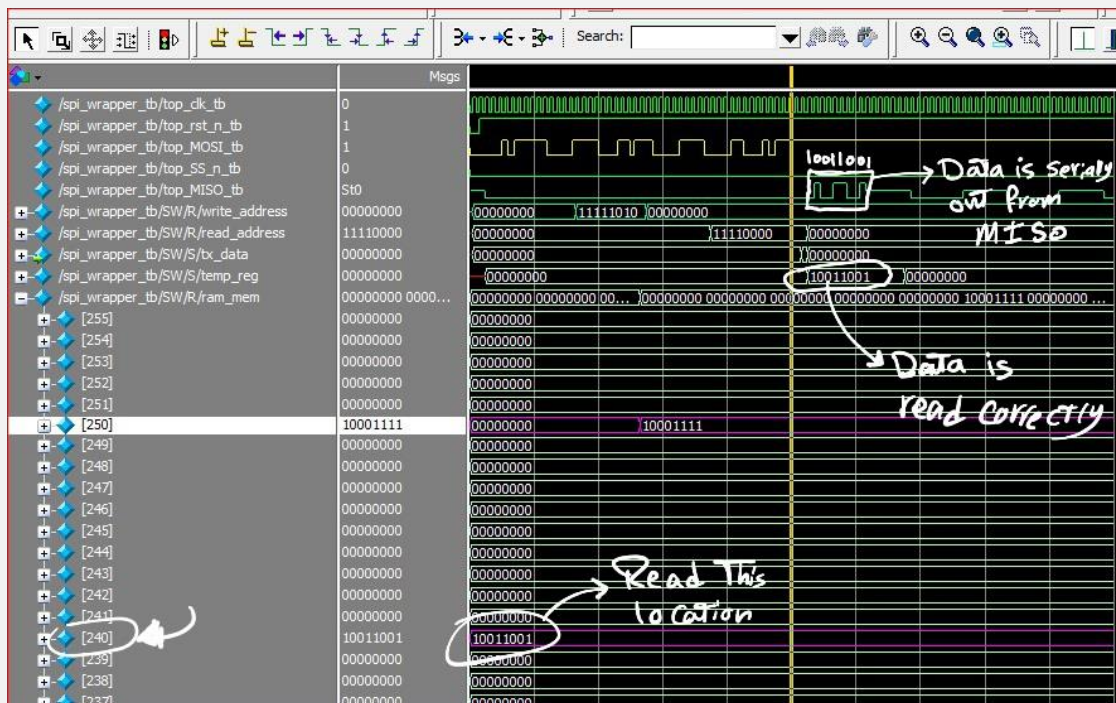
## Explained Waveform



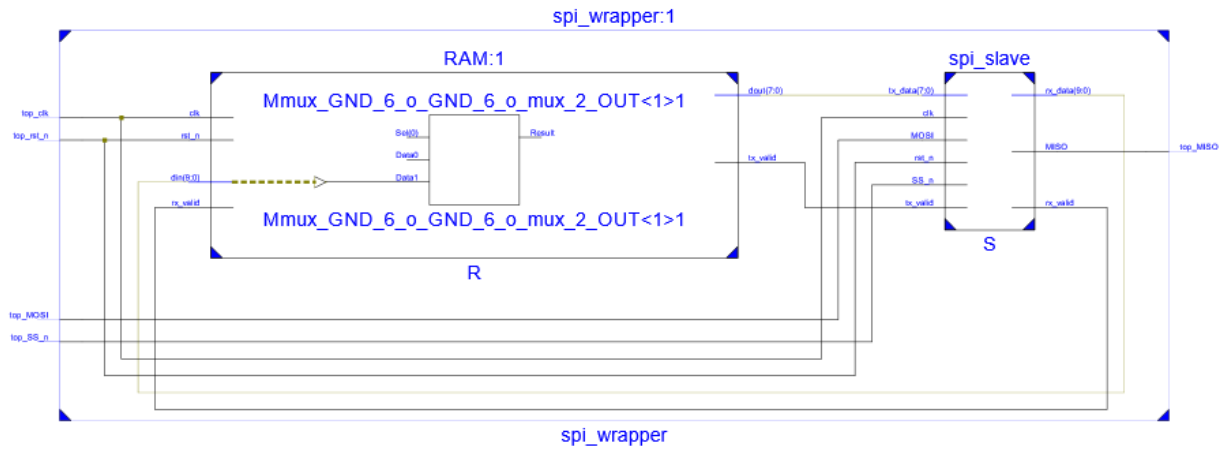
- Test Case 2 -> Write in location 250 then read from different location 240 that I have previously set manually during initialization.



## Explained Waveform

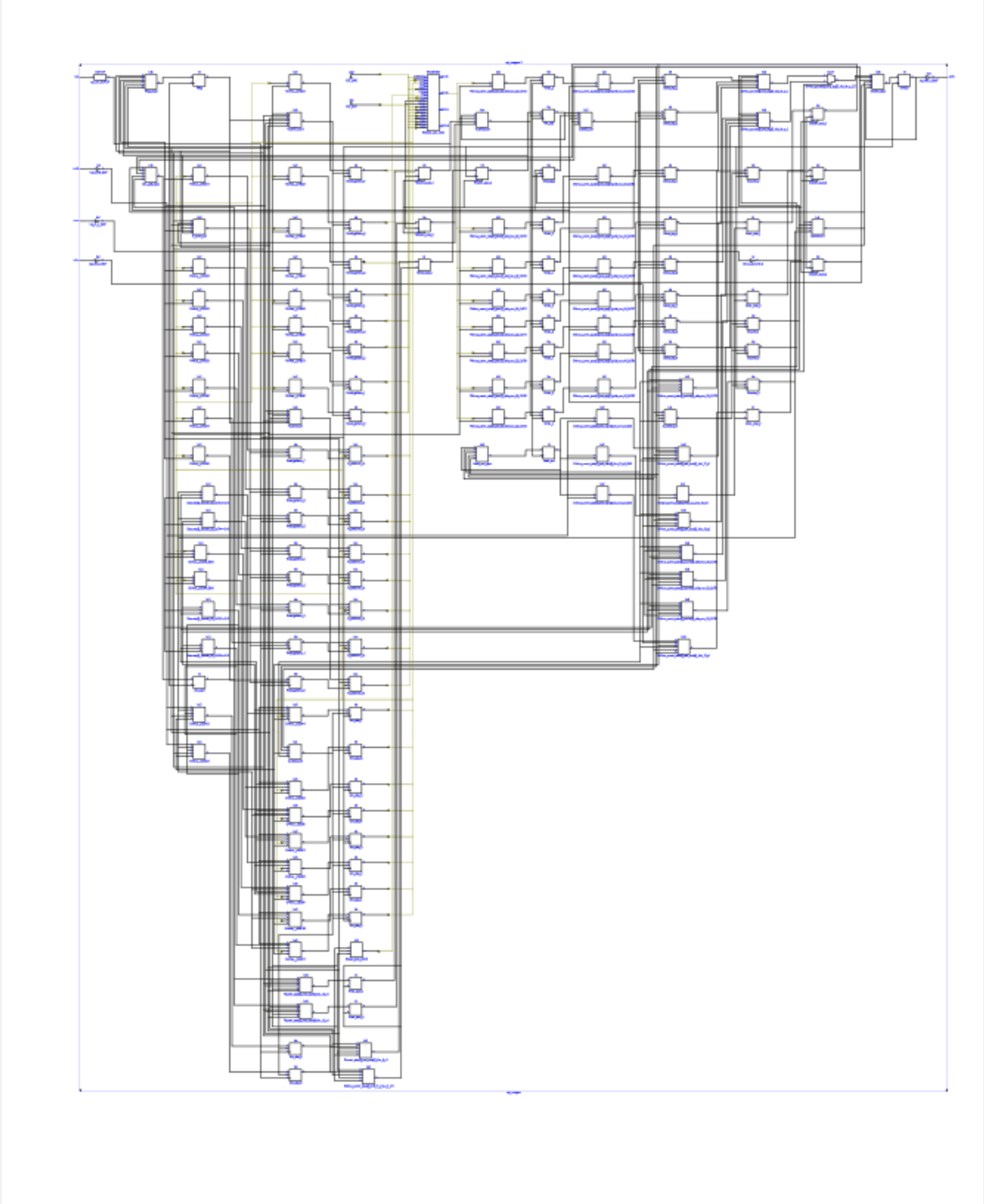


# SPI\_WRAPPER RTL Schematic



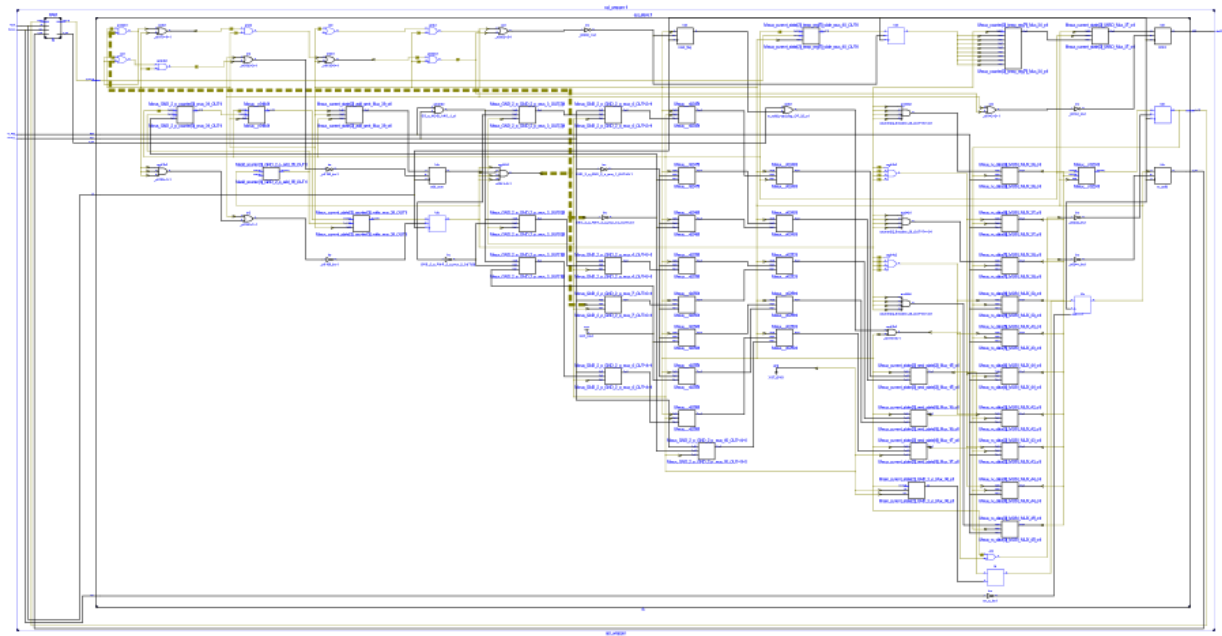
# SPI\_WRAPPER

## Technology Schematic

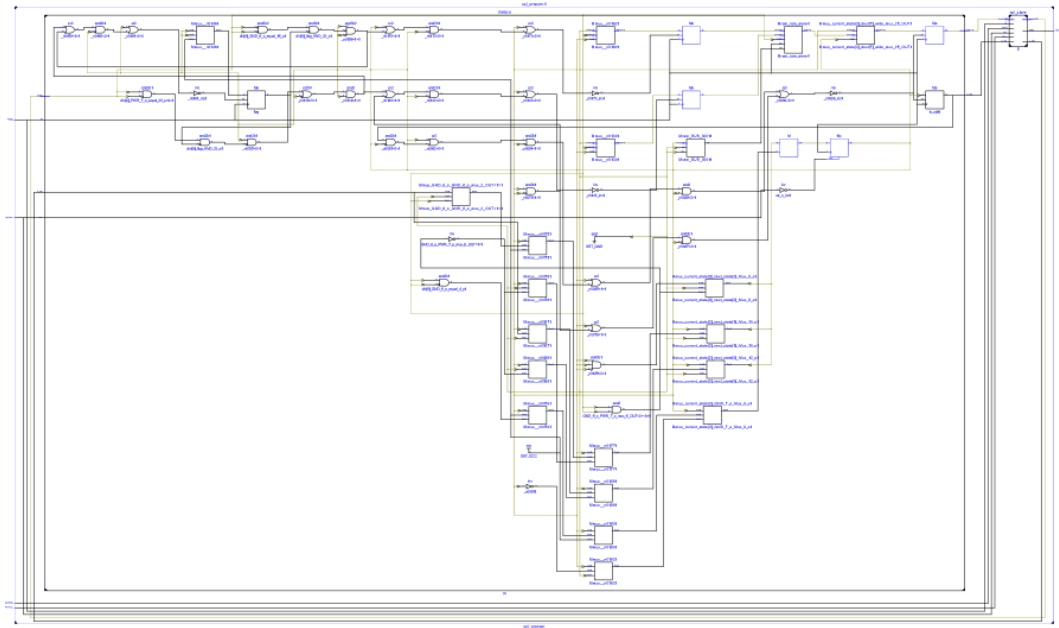


# SPI\_SLAVE

## RTL Schematic



# RAM RTL Schematic



- **Design GitHub Link :** [HassanKhaled11/SPI\\_project \(github.com\)](https://github.com/HassanKhaled11/SPI_project)
- **LinkedIn :** [Hassan Khaled | LinkedIn](#)
- **Facebook :** <https://m.facebook.com/100087870183210/>