

SPI Slave with Single Port RAM project

Abdelrhman ahmed kafsheen
NAND Ninjas group

August 2025

1 Introduction

This document presents the Verilog implementation of an SPI (Serial Peripheral Interface) memory interface system, designed for a digital design project targeting a Spartan-6 FPGA. The system includes four modules: an SPI Slave module for serial communication, a RAM module for data storage, an SPI Wrapper module that integrates the two, and a testbench for functional verification. A QuestaSim .do file automates the simulation process, with waveform captures illustrating the system's behavior during write and read operations. Vivado synthesis results, including a schematic and timing analysis, demonstrate the design's implementation feasibility. The `SPI_Slave` module interfaces with an SPI master, supporting write, read address, and read data operations using a five-state finite state machine (FSM). The `RAM` module provides a 256x8-bit synchronous memory, controlled by the SPI Slave's outputs. The `SPI_wrapper` module ensures seamless integration, and the `testbench_spi` module verifies the system by testing write and read operations. The .do file compiles and simulates the system, displaying key signals in the waveform viewer. The waveform captures, followed by Vivado schematic and timing results, confirm the design's functionality and synthesis compatibility.

2 SPI Slave Verilog Code

The following is the Verilog code for the `SPI_Slave` module, implementing SPI slave functionality with states: IDLE, CHK_CMD, WRITE, READ_ADD, and READ_DATA.

```
1 module
2     SPI_Slave(MOSI,SS_n,clk,rst_n,rx_data,tx_valid,tx_data,MISO,rx_valid);
3 input  MOSI,SS_n,clk,rst_n,tx_valid;
4 input  [7:0] tx_data;
5 output reg MISO,rx_valid;
6 output reg [9:0] rx_data;
7 parameter IDLE=3'b000;
8 parameter CHK_CMD=3'b001;
9 parameter WRITE=3'b010;
10 parameter READ_ADD=3'b011;
11 parameter READ_DATA=3'b100;
12 reg ADDRESS_read;
13 reg [3:0] counter_up;
14 reg [3:0] counter_down;
```

```

14 reg [9:0] shift_reg_parallel;
15 reg [2:0] cs ,ns;
16 always@(posedge clk) begin
17     if(~rst_n)
18         cs<=IDLE;
19     else
20         cs<=ns;
21 end
22 always@(*) begin
23     case(cs)
24     IDLE: begin
25         if(SS_n)
26             ns=IDLE;
27         else
28             ns=CHK_CMD;
29     end
30     CHK_CMD : begin
31         if (SS_n==0 && MOSI==0)
32             ns =WRITE;
33         else if(SS_n==0 && MOSI==1) begin
34             casex(ADDRESS_read)
35                 1'b0 : ns=READ_ADD;
36                 1'b1: ns=READ_DATA;
37                 1'bx: ns=READ_ADD;
38             endcase
39         end
40     end
41     WRITE: begin
42         if(SS_n==0 )
43             ns=WRITE;
44         else begin
45             ns=IDLE;
46         end
47     end
48     READ_ADD : begin
49         ADDRESS_read=1;
50         if(SS_n==0) begin
51             ns=READ_ADD;
52         end
53         else begin
54             ns=IDLE;
55         end
56     end
57     READ_DATA : begin
58         ADDRESS_read=0;
59         if(SS_n==0)
60             ns=READ_DATA;
61         else begin
62             ns=IDLE;
63         end
64     end

```

```

65     default    : ns=IDLE;
66     endcase
67 end
68 always @(posedge clk) begin
69     case(cs)
70         IDLE: rx_valid<=0;
71         CHK_CMD : begin
72             counter_up<=0;
73             counter_down<=0;
74             shift_reg_parallel<=0;
75             rx_valid<=0;
76         end
77         WRITE , READ_ADD : begin
78             shift_reg_parallel[9-counter_up]<=MOSI;
79             counter_up<=counter_up+1;
80             if(counter_up==10) begin
81                 counter_up<=0;
82                 rx_data<=shift_reg_parallel;
83                 rx_valid<=1;
84             end
85         end
86         READ_DATA : begin
87             MISO<=tx_data[7-counter_down];
88             counter_down<=counter_down+1;
89             if(counter_down==8) begin
90                 counter_down<=0;
91             end
92         end
93     endcase
94 end
95 endmodule

```

Listing 1: SPI_{slave}.v : VerilogCode for SPI Slave Module

3 RAM Verilog Code

The following is the Verilog code for the RAM module, implementing a 256x8-bit synchronous RAM. It supports write and read operations controlled by a 10-bit input, interfacing with the SPI Slave module's rx_data and rx_valid signals.

```

1 module RAM(din,rx_valid,clk,rst_n,dout,tx_valid);
2 parameter MEM_DEPTH=256;
3 parameter ADDR_SIZE=8;
4 input rx_valid ,clk,rst_n;
5 input [9:0] din;
6 output reg tx_valid;
7 output reg [7:0] dout;
8 reg [ADDR_SIZE-1:0] ADDR_read; // Address for read operation
9 reg [ADDR_SIZE-1:0] ADDR_write; // Address for write operation
10 reg [7:0] mem[MEM_DEPTH-1:0]; // array of 256 registers , each
    register its size is 8

```

```

11 always @(posedge clk ) begin
12     if(~rst_n)
13         dout<=0;
14     else begin
15         if(rx_valid) begin // to assign value of din
16             case (din[9:8])
17                 2'b00 : ADDR_write<=din[7:0]; // assign the write address
18                 2'b01: mem[ADDR_write]<=din[7:0];
19                 2'b10: begin
20                     ADDR_read<=din[7:0]; //assign read address
21                     tx_valid<=0;
22                 end
23                 2'b11: begin
24                     dout<=mem[ADDR_read];
25                     tx_valid<=1;
26                 end
27             endcase
28         end
29     end
30 end
31 endmodule

```

Listing 2: RAM.v: Verilog Code for RAM Module

4 SPI Wrapper Verilog Code

The following is the Verilog code for the SPI_wrapper module, which integrates the SPI_Slave and RAM modules to form a complete SPI memory interface system. It connects the SPI Slave's outputs to the RAM's inputs and vice versa, enabling serial communication and memory operations.

```

1 module SPI_wrapper (MOSI,MISO,SS_n,clk,rst_n);
2 input MOSI,SS_n,clk,rst_n;
3 output MISO;
4 wire rx_valid_internal ;
5 wire tx_valid_internal;
6 wire [7:0] tx_data_internal;
7 wire[9:0] rx_data_internal;
8 RAM
9     ram(.din(rx_data_internal),.rx_valid(rx_valid_internal),.clk(clk),.rst_n(rst_n));
10 SPI_Slave
11     spi(.MOSI(MOSI),.MISO(MISO),.SS_n(SS_n),.clk(clk),.rst_n(rst_n),.rx_data(rx_data_internal),.tx_valid(tx_valid_internal),.tx_data(tx_data_internal));
12 endmodule

```

Listing 3: SPI_wrapper.v : Verilog Code for SPI Wrapper Module

5 Testbench Verilog Code

The following is the Verilog code for the `testbench_spi` module, which verifies the functionality of the `SPI_wrapper` module. It tests write address, write data, read address, and read data operations by driving the SPI inputs and monitoring the outputs, with the RAM initialized from a `mem.dat` file.

```
1 module testbench_spi();
2 reg clk_tb, rst_n_tb, mosi_tb, ss_n_tb;
3 wire miso_tb;
4 SPI_wrapper tb(mosi_tb, miso_tb, ss_n_tb, clk_tb, rst_n_tb);
5 initial begin
6     clk_tb=0;
7     forever #1 clk_tb=~clk_tb;
8 end
9 initial begin
10    $readmemh("mem.dat", tb.ram.mem);
11    rst_n_tb=0; mosi_tb=1; ss_n_tb=1;
12    @(negedge clk_tb);
13    // TEST Write address
14    rst_n_tb=1; // to go to CHK_CMD
15    mosi_tb=0;
16    ss_n_tb=0;
17    @(negedge clk_tb);
18    rst_n_tb=1; // to go to write address state
19    mosi_tb=0;
20    ss_n_tb=0;
21    @(negedge clk_tb);
22    // 0010101011
23    rst_n_tb=1;
24    mosi_tb=0;
25    ss_n_tb=0;
26    @(negedge clk_tb);
27    rst_n_tb=1;
28    mosi_tb=0;
29    ss_n_tb=0;
30    @(negedge clk_tb);
31    rst_n_tb=1;
32    mosi_tb=1;
33    ss_n_tb=0;
34    @(negedge clk_tb);
35    rst_n_tb=1;
36    mosi_tb=0;
37    ss_n_tb=0;
38    @(negedge clk_tb);
39    rst_n_tb=1;
40    mosi_tb=1;
41    ss_n_tb=0;
42    @(negedge clk_tb);
43    rst_n_tb=1;
44    mosi_tb=0;
45    ss_n_tb=0;
```

```

46     @(negedge clk_tb);
47     rst_n_tb=1;
48     mosi_tb=1;
49     ss_n_tb=0;
50     @(negedge clk_tb);
51     rst_n_tb=1;
52     mosi_tb=0;
53     ss_n_tb=0;
54     @(negedge clk_tb);
55     rst_n_tb=1;
56     mosi_tb=1;
57     ss_n_tb=0;
58     @(negedge clk_tb);
59     rst_n_tb=1;
60     mosi_tb=1;
61     ss_n_tb=0;
62     @(negedge clk_tb);
63
64     rst_n_tb=1; mosi_tb=1; ss_n_tb=1;
65     @(negedge clk_tb);
66     @(negedge clk_tb); // return to IDLE
67     // TEST write data
68     rst_n_tb=1; // to go to CHK_CMD
69     mosi_tb=0;
70     ss_n_tb=0;
71     @(negedge clk_tb);
72     rst_n_tb=1; // to go to write data state
73     mosi_tb=0;
74     ss_n_tb=0;
75     @(negedge clk_tb);
76     // 0110101010
77     rst_n_tb=1;
78     mosi_tb=0;
79     ss_n_tb=0;
80     @(negedge clk_tb);
81     rst_n_tb=1;
82     mosi_tb=1;
83     ss_n_tb=0;
84     @(negedge clk_tb);
85     rst_n_tb=1;
86     mosi_tb=1;
87     ss_n_tb=0;
88     @(negedge clk_tb);
89     rst_n_tb=1;
90     mosi_tb=0;
91     ss_n_tb=0;
92     @(negedge clk_tb);
93     rst_n_tb=1;
94     mosi_tb=1;
95     ss_n_tb=0;
96     @(negedge clk_tb);

```

```

97     rst_n_tb=1;
98     mosi_tb=0;
99     ss_n_tb=0;
100    @(negedge clk_tb);
101    rst_n_tb=1;
102    mosi_tb=1;
103    ss_n_tb=0;
104    @(negedge clk_tb);
105    rst_n_tb=1;
106    mosi_tb=0;
107    ss_n_tb=0;
108    @(negedge clk_tb);
109    rst_n_tb=1;
110    mosi_tb=1;
111    ss_n_tb=0;
112    @(negedge clk_tb);
113    rst_n_tb=1;
114    mosi_tb=0;
115    ss_n_tb=0;
116    @(negedge clk_tb);
117    rst_n_tb=1; mosi_tb=1; ss_n_tb=1;
118    @(negedge clk_tb);
119    @(negedge clk_tb); // return to IDLE
120    // test read address
121    rst_n_tb=1; // to go to CHK_CMD
122    mosi_tb=1;
123    ss_n_tb=0;
124    @(negedge clk_tb);
125    rst_n_tb=1; // to go to read adress state
126    mosi_tb=1;
127    ss_n_tb=0;
128    @(negedge clk_tb);
129    // 1010101101
130    rst_n_tb=1;
131    mosi_tb=1;
132    ss_n_tb=0;
133    @(negedge clk_tb);
134    rst_n_tb=1;
135    mosi_tb=0;
136    ss_n_tb=0;
137    @(negedge clk_tb);
138    rst_n_tb=1;
139    mosi_tb=1;
140    ss_n_tb=0;
141    @(negedge clk_tb);
142    rst_n_tb=1;
143    mosi_tb=0;
144    ss_n_tb=0;
145    @(negedge clk_tb);
146    rst_n_tb=1;
147    mosi_tb=1;

```

```

148     ss_n_tb=0;
149     @(negedge clk_tb);
150     rst_n_tb=1;
151     mosi_tb=0;
152     ss_n_tb=0;
153     @(negedge clk_tb);
154     rst_n_tb=1;
155     mosi_tb=1;
156     ss_n_tb=0;
157     @(negedge clk_tb);
158     rst_n_tb=1;
159     mosi_tb=1;
160     ss_n_tb=0;
161     @(negedge clk_tb);
162     rst_n_tb=1;
163     mosi_tb=0;
164     ss_n_tb=0;
165     @(negedge clk_tb);
166     rst_n_tb=1;
167     mosi_tb=1;
168     ss_n_tb=0;
169     @(negedge clk_tb);
170
171     rst_n_tb=1; mosi_tb=1; ss_n_tb=1;
172     @(negedge clk_tb);
173     @(negedge clk_tb); // return to IDLE
174     // test read data
175     // dummy data and to make rx_valid=1
176     rst_n_tb=1; // to go to CHK_CMD
177     mosi_tb=1;
178     ss_n_tb=0;
179     @(negedge clk_tb);
180     rst_n_tb=1; // to go to write state
181     mosi_tb=0;
182     ss_n_tb=0;
183     @(negedge clk_tb);
184     // 1100000111
185     rst_n_tb=1;
186     mosi_tb=1;
187     ss_n_tb=0;
188     @(negedge clk_tb);
189     rst_n_tb=1;
190     mosi_tb=1;
191     ss_n_tb=0;
192     @(negedge clk_tb);
193     rst_n_tb=1;
194     mosi_tb=0;
195     ss_n_tb=0;
196     @(negedge clk_tb);
197     rst_n_tb=1;
198     mosi_tb=0;

```



```

199     ss_n_tb=0;
200     @(negedge clk_tb);
201     rst_n_tb=1;
202     mosi_tb=0;
203     ss_n_tb=0;
204     @(negedge clk_tb);
205     rst_n_tb=1;
206     mosi_tb=0;
207     ss_n_tb=0;
208     @(negedge clk_tb);
209     rst_n_tb=1;
210     mosi_tb=0;
211     ss_n_tb=0;
212     @(negedge clk_tb);
213     rst_n_tb=1;
214     mosi_tb=1;
215     ss_n_tb=0;
216     @(negedge clk_tb);
217     rst_n_tb=1;
218     mosi_tb=1;
219     ss_n_tb=0;
220     @(negedge clk_tb);
221     rst_n_tb=1;
222     mosi_tb=1;
223     ss_n_tb=0;
224     @(negedge clk_tb);
225     ss_n_tb=1; rst_n_tb=1; // return to idle
226     @(negedge clk_tb);
227     ss_n_tb=0; rst_n_tb=1; // go to chc-cmd
228     @(negedge clk_tb);
229     ss_n_tb=0; rst_n_tb=1; mosi_tb=1; // go to read data
230     @(negedge clk_tb);
231
232     @(negedge clk_tb); // another clock for ram to process the
        data because it's sequential
233
234     rst_n_tb=1; mosi_tb=1; ss_n_tb=0;
235     repeat (8) @(negedge clk_tb);
236     //return to idle
237     rst_n_tb=1; mosi_tb=0; ss_n_tb=1;
238     @(negedge clk_tb);
239
240     $stop;
241 end
242 endmodule

```

Listing 4: testbench_{pi.v} : VerilogCode for TestbenchModule

6 Simulation Script

The following is the QuestaSim .do file used to compile and simulate the SPI memory interface system. It compiles the Verilog modules, runs the testbench, and adds key signals to the waveform viewer for analysis. The file names have been updated to match the provided module names, and full paths are included for robustness.

```
1 vlib work
2 vmap work work
3 vlog -work work -vopt +acc "D:/digital design/project/RAM.v"
4 vlog -work work -vopt +acc "D:/digital
  design/project/SPI_Slave.v"
5 vlog -work work -vopt +acc "D:/digital
  design/project/SPI_wrapper.v"
6 vlog -work work -vopt +acc "D:/digital
  design/project/testbench_spi.v"
7 vsim -voptargs=+acc work.testbench_spi
8 add wave -position insertpoint \
9 sim:/testbench_spi/ss_n_tb \
10 sim:/testbench_spi/rst_n_tb \
11 sim:/testbench_spi/mosi_tb \
12 sim:/testbench_spi/miso_tb \
13 sim:/testbench_spi/clk_tb
14 add wave -position insertpoint \
15 sim:/testbench_spi/tb/ram/mem
16 add wave -position insertpoint \
17 sim:/testbench_spi/tb/ram/dout
18 add wave -position insertpoint \
19 sim:/testbench_spi/tb/ram/din
20 add wave -position insertpoint \
21 sim:/testbench_spi/tb/spi/tx_valid
22 add wave -position insertpoint \
23 sim:/testbench_spi/tb/spi/tx_data
24 add wave -position insertpoint \
25 sim:/testbench_spi/tb/spi/rx_valid
26 add wave -position insertpoint \
27 sim:/testbench_spi/tb/spi/rx_data
28 add wave -position insertpoint \
29 sim:/testbench_spi/tb/spi/cs
30 add wave -position insertpoint \
31 sim:/testbench_spi/tb/spi/counter_up
32 add wave -position insertpoint \
33 sim:/testbench_spi/tb/spi/counter_down
34 run -all
35 #quit -sim
```

Listing 5: simulate.do: QuestaSim Simulation Script

7 Wave forms

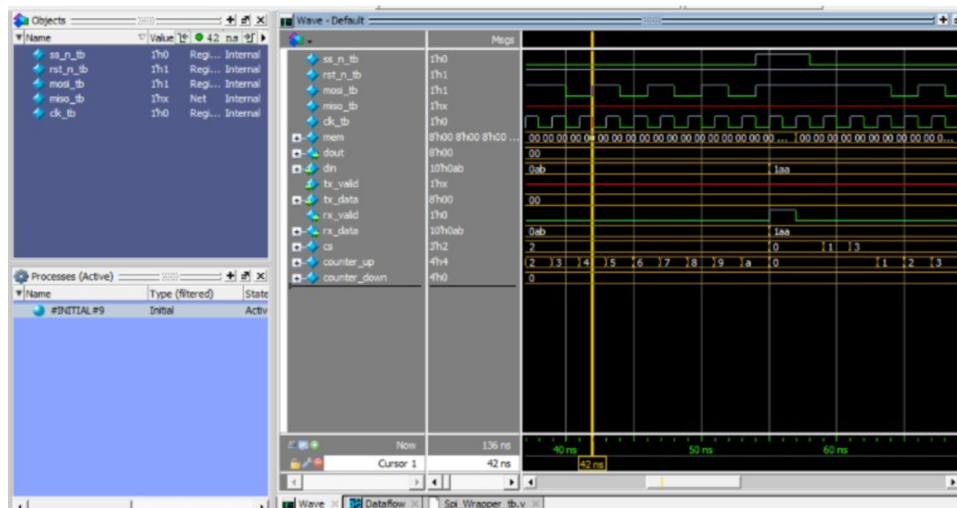


Figure 1: write address state

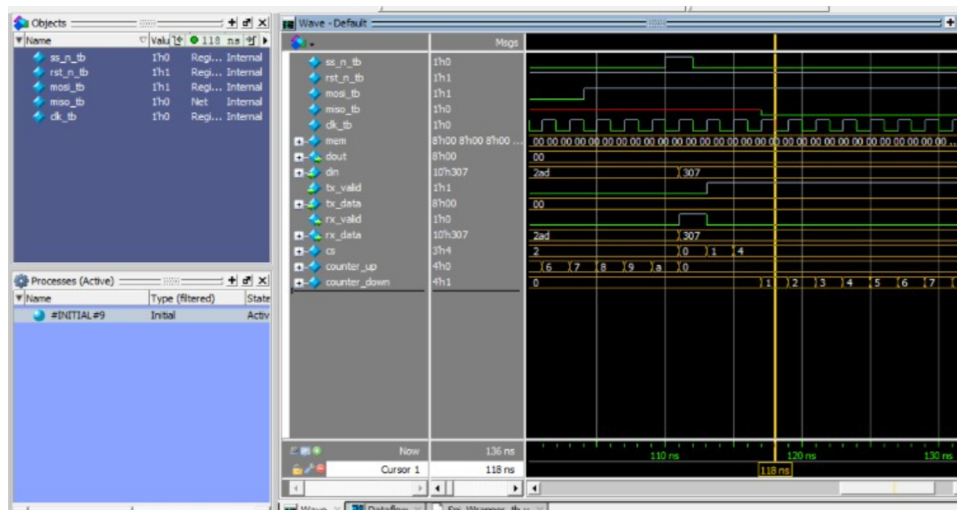


Figure 2: Write data state

8 Vivado

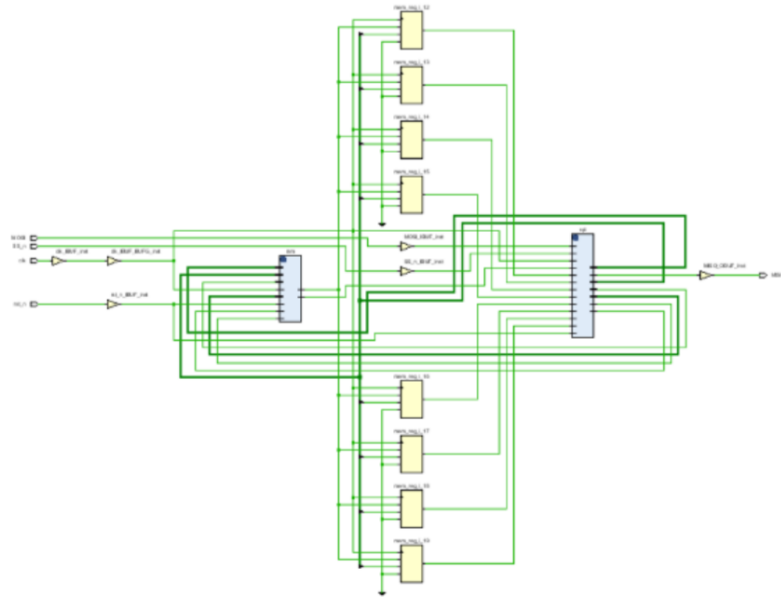


Figure 3: Vivado Schematic of the SPI Memory Interface System

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 7.006 ns	Worst Hold Slack (WHS): 0.192 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: 56	Total Number of Endpoints: 56	Total Number of Endpoints: 37

All user specified timing constraints are met.

Figure 4: Timing Analysis Report from Vivado

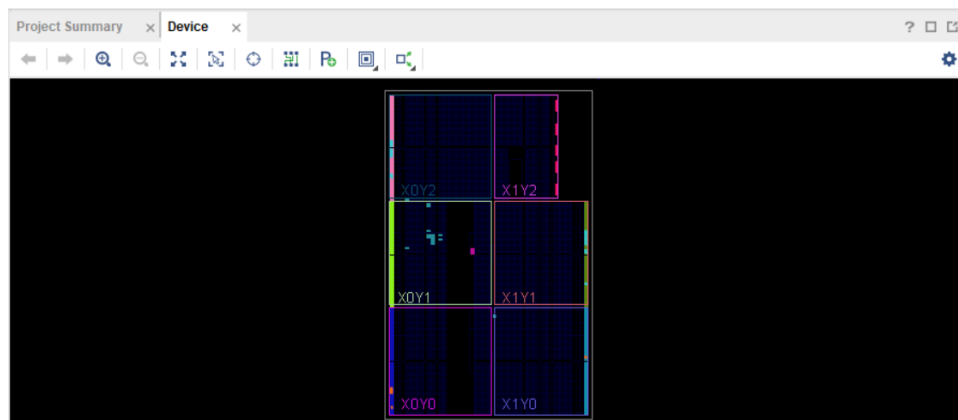


Figure 5: FPGA device