# 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.

```
module
   SPI_Slave(MOSI,SS_n,clk,rst_n,rx_data,tx_valid,tx_data,MISO,rx_valid);
input MOSI,SS_n,clk,rst_n,tx_valid;
input [7:0] tx_data;
output reg MISO,rx_valid;
output reg [9:0] rx_data;
parameter IDLE=3'b000;
parameter CHK_CMD=3'b001;
parameter WRITE=3'b010;
parameter READ_ADD=3'b011;
parameter READ_DATA=3'b100;
reg ADDRESS_read;
reg[3:0] counter_up;
reg[3:0] counter_down;
```

```verilog
reg [9:0] shift_reg_parallel;
reg [2:0] cs ,ns;
always@(posedge clk) begin
    if(~rst_n)
     cs<=IDLE;
     else
     cs<=ns;
end
always@(*) begin
    case(cs)
    IDLE: begin
        if(SS_n)
        ns=IDLE;
        else
        ns=CHK_CMD;
    end
   CHK_CMD : begin
    if (SS_n==0 && MOSI==0)
    ns =WRITE;
    else if(SS_n==0 && MOSI==1) begin
        casex(ADDRESS_read)
        1'b0 : ns=READ_ADD;
        1'b1: ns=READ_DATA;
        1'bx: ns=READ_ADD;
        endcase
    end
   end
   WRITE: begin
     if(SS_n==0 )
     ns=WRITE;
     else begin
        ns=IDLE;
     end
   end
   READ_ADD :  begin
    ADDRESS_read=1;
    if(SS_n==0) begin
    ns=READ_ADD;
    end
    else begin
        ns=IDLE;
    end
   end
   READ_DATA : begin
    ADDRESS_read=0;
    if(SS_n==0)
    ns=READ_DATA;
    else begin
        ns=IDLE;
    end
   end
   default  : ns=IDLE;
    endcase
end
always @(posedge clk) begin
   case(cs)
    IDLE: rx_valid<=0;
    CHK_CMD : begin
         counter_up<=0;
         counter_down<=0;
         shift_reg_parallel<=0;
         rx_valid<=0;
    end
    WRITE , READ_ADD : begin
        shift_reg_parallel[9-counter_up]<=MOSI;
        counter_up<=counter_up+1;
        if(counter_up==10) begin
            counter_up<=0;
            rx_data<=shift_reg_parallel;
            rx_valid<=1;
        end
    end
   READ_DATA : begin
       MISO<=tx_data[7-counter_down];
        counter_down<=counter_down+1;
        if(counter_down==8) begin
            counter_down<=0;
        end
    end
   endcase
end
endmodule
```

Listing 1: SPI$_{S}lave.v : VerilogCodeforSPISlaveModule$

## 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.

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

```
always @(posedge clk ) begin
    if(~rst_n)
    dout<=0;
   else begin
 if(rx_valid)   begin  // to assign value of din
   case (din[9:8])
   2'b00 : ADDR_write<=din[7:0]; // assign the write address
   2'b01: mem[ADDR_write]<=din[7:0];
   2'b10: begin
    ADDR_read<=din[7:0]; //assign read address
       tx_valid<=0;
   end
   2'b11: begin
    dout<=mem[ADDR_read];
    tx_valid<=1;
   end
   endcase
 end
   end
end
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.

```
module SPI_wrapper (MOSI,MISO,SS_n,clk,rst_n);
input MOSI,SS_n,clk,rst_n;
output  MISO;
wire rx_valid_internal ;
wire tx_valid_internal;
wire [7:0] tx_data_internal;
wire[9:0] rx_data_internal;
RAM
   ram(.din(rx_data_internal),.rx_valid(rx_valid_internal),.clk(clk),.rst_n(
SPI_Slave
   spi(.MOSI(MOSI),.MISO(MISO),.SS_n(SS_n),.clk(clk),.rst_n(rst_n),.rx_data(
,.tx_valid(tx_valid_internal),.tx_data(tx_data_internal));
endmodule
```

Listing 3: SPI$_{w}rapper.v : VerilogCodeforSPIWrapperModule$

## 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.

```
module testbench_spi();
reg clk_tb,rst_n_tb,mosi_tb,ss_n_tb;
wire  miso_tb;
SPI_wrapper tb(mosi_tb,miso_tb,ss_n_tb,clk_tb,rst_n_tb);
initial begin
   clk_tb=0;
    forever #1 clk_tb=~clk_tb;
end
initial begin
  $readmemh("mem.dat",tb.ram.mem);
    rst_n_tb=0; mosi_tb=1; ss_n_tb=1;
    @(negedge clk_tb);
    // TEST Write address
    rst_n_tb=1;  // to go to CHK_CMD
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
     rst_n_tb=1;  // to go to write  address state
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    // 0010101011
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
     rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);

      rst_n_tb=1; mosi_tb=1; ss_n_tb=1;
@(negedge clk_tb);
@(negedge clk_tb); // return to IDLE
    // TEST write data
     rst_n_tb=1;  // to go to CHK_CMD
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
     rst_n_tb=1;  // to go to write  data state
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    // 0110101010
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
     rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
```

```verilog
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
       rst_n_tb=1; mosi_tb=1; ss_n_tb=1;
@(negedge clk_tb);
@(negedge clk_tb); // return to IDLE
      // test read address
      rst_n_tb=1;  // to go to CHK_CMD
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
     rst_n_tb=1;  // to go to read adress state
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    // 1010101101
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
     rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);

    rst_n_tb=1; mosi_tb=1; ss_n_tb=1;
@(negedge clk_tb);
@(negedge clk_tb); // return to IDLE
   // test read data
     // dummy data and to make rx_valid=1
      rst_n_tb=1;  // to go to CHK_CMD
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
     rst_n_tb=1;  // to go to write  state
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    // 1100000111
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
     rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=0;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    rst_n_tb=1;
    mosi_tb=1;
    ss_n_tb=0;
    @(negedge clk_tb);
    ss_n_tb=1; rst_n_tb=1; // return to idle
    @(negedge clk_tb);
    ss_n_tb=0; rst_n_tb=1; // go to chc-cmd
    @(negedge clk_tb);
     ss_n_tb=0; rst_n_tb=1; mosi_tb=1; // go to read data
     @(negedge clk_tb);

   @(negedge clk_tb); // another clock for ram to process the
      data because it's sequential

rst_n_tb=1; mosi_tb=1; ss_n_tb=0;
repeat (8) @(negedge clk_tb);
//return to idle
rst_n_tb=1; mosi_tb=0; ss_n_tb=1;
@(negedge clk_tb);

    $stop;
end
endmodule
```

Listing 4: testbench$_spi.v : Verilog Code for Testbench Module$

## 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.

```
vlib work
vmap work work
vlog -work work -vopt +acc "D:/digital design/project/RAM.v"
vlog -work work -vopt +acc "D:/digital
   design/project/SPI_Slave.v"
vlog -work work -vopt +acc "D:/digital
   design/project/SPI_wrapper.v"
vlog -work work -vopt +acc "D:/digital
   design/project/testbench_spi.v"
vsim -voptargs=+acc work.testbench_spi
add wave -position insertpoint  \
sim:/testbench_spi/ss_n_tb \
sim:/testbench_spi/rst_n_tb \
sim:/testbench_spi/mosi_tb \
sim:/testbench_spi/miso_tb \
sim:/testbench_spi/clk_tb
add wave -position insertpoint  \
sim:/testbench_spi/tb/ram/mem
add wave -position insertpoint  \
sim:/testbench_spi/tb/ram/dout
add wave -position insertpoint  \
sim:/testbench_spi/tb/ram/din
add wave -position insertpoint  \
sim:/testbench_spi/tb/spi/tx_valid
add wave -position insertpoint  \
sim:/testbench_spi/tb/spi/tx_data
add wave -position insertpoint  \
sim:/testbench_spi/tb/spi/rx_valid
add wave -position insertpoint  \
sim:/testbench_spi/tb/spi/rx_data
add wave -position insertpoint  \
sim:/testbench_spi/tb/spi/cs
add wave -position insertpoint  \
sim:/testbench_spi/tb/spi/counter_up
add wave -position insertpoint  \
sim:/testbench_spi/tb/spi/counter_down
run -all
#quit -sim
```

Listing 5: simulate.do: QuestaSim Simulation Script

## 7 Wave forms

.

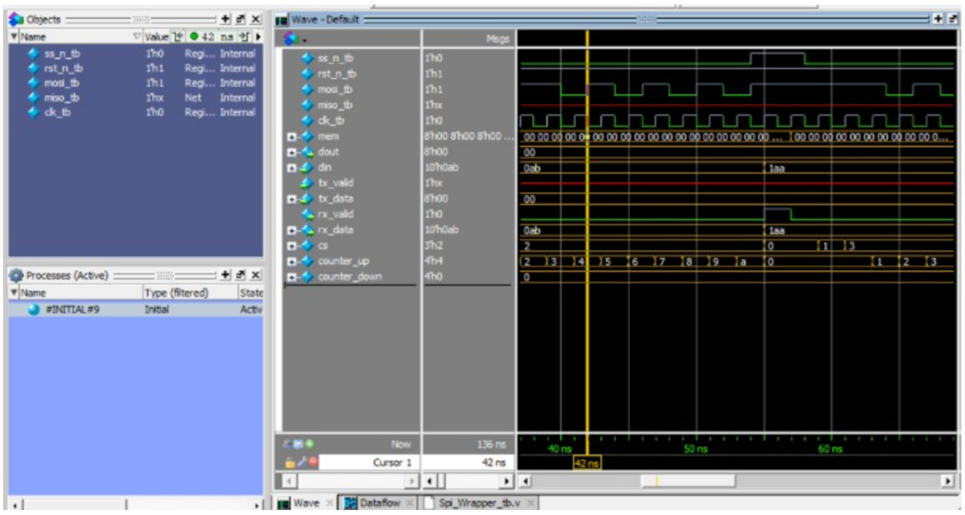


Figure 1: write address state

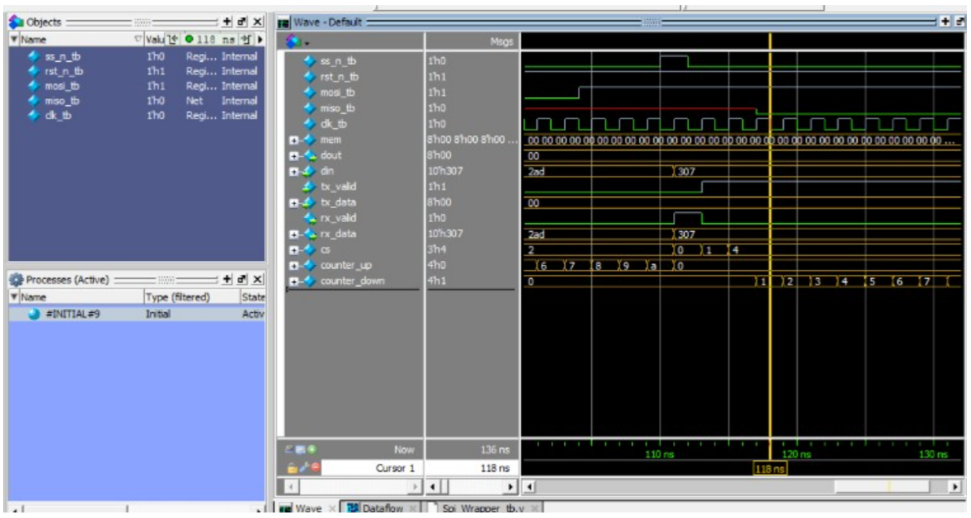


Figure 2: Write data state

## 8 Vivado

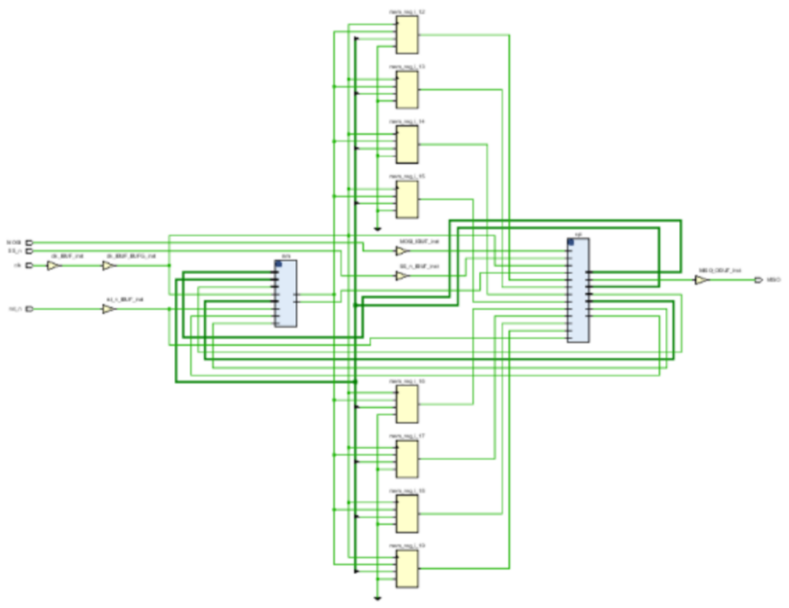
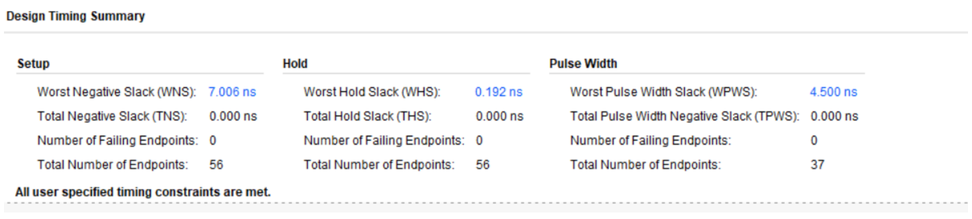


Figure 3: Vivado Schematic of the SPI Memory Interface System



**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

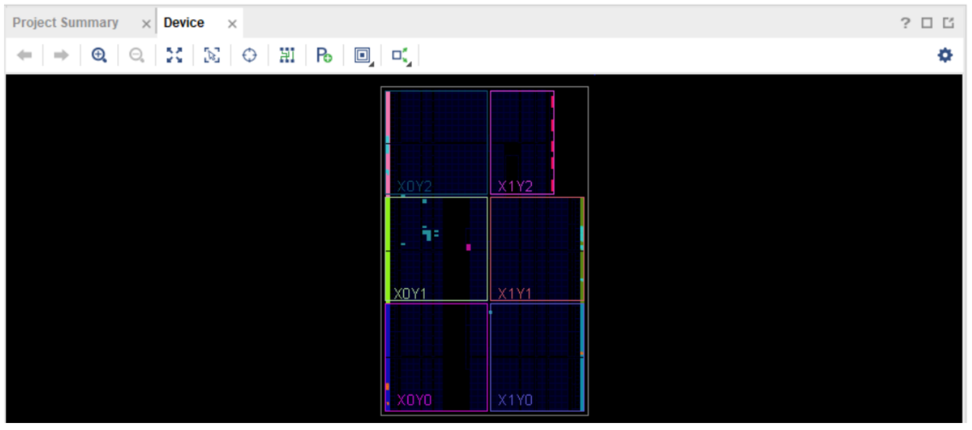


Figure 5: FPGA device