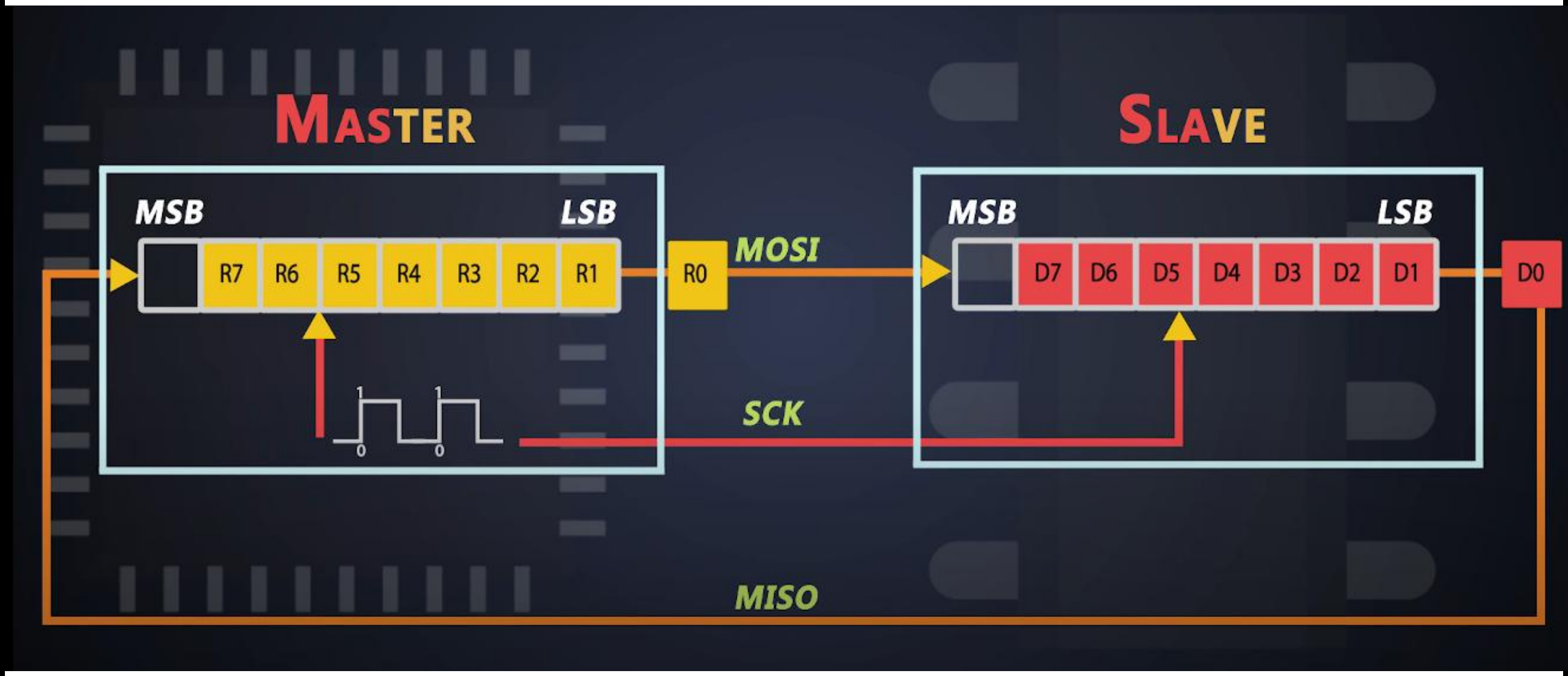
SPI_Project_version_1

Eng\Ahmed Hamed Mohamed

SPI (Serial Peripheral Interface)



SPI_Master_32bit

Click here SPI_Master

```
1
2   module SPI_Master_32bit(
3       input wire start,
4       input wire reset,
5       input wire clk,
6       input wire [31:0] data_in,
7       output reg [31:0] data_out,
8       input wire MISO,
9       output reg MOSI,
10      output reg SCLK,
11      output reg CS,
12      output reg [31:0] shifter_send,
13      output reg [31:0] shifter_recv
14  );
15      // Previous Value os SCLK
16      // detect SCLK edge
17      reg SCLK_prev;
18
19      always @(posedge clk or posedge reset) begin
20          if (reset) begin
21              SCLK_prev <= 0;
22          end
23          else begin
24              SCLK_prev <= SCLK;
25          end
26      end
27      wire SCLK_rising = (SCLK == 1 && SCLK_prev == 0);
28      wire SCLK_falling = (SCLK == 0 && SCLK_prev == 1);
29
```

SPI_Master_32bit

```
    wire SCLK_rising = (SCLK == 1 && SCLK_prev == 0);
    wire SCLK_falling = (SCLK == 0 && SCLK_prev == 1);

    // ====== New Signal ======
    reg SCLK_EN;  // to control SCLK
    // Clock Divider controlled by SCLK_EN
    always @(posedge clk or posedge reset) begin
        if (reset)
            SCLK <= 0;
        else if (SCLK_EN)
            SCLK <= ~SCLK;
        else
            SCLK <= 0;
    end


    reg [5:0] bit_cnt;              // Counts 0 to 63 (32 bits * 2 edges)
    reg [1:0] state;
    parameter IDLE = 2'b00, TRANSFER = 2'b01, DONE = 2'b10;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            CS <= 1;
            SCLK_EN <= 0;
            MOSI <= 0;
            bit_cnt <= 0;
            shifter_send <= 0;
            shifter_recv <= 0;
            data_out <= 0;
            state <= IDLE;
        end
        else begin
            case (state)
                IDLE: begin
                    if (bit_cnt < 32)begin
                        MOSI <= 0;
```

SPI_Master_32bit

```
60                  IDLE: begin
61                      if (bit_cnt < 32)begin
62                          MOSI <= 0;
63                          SCLK_EN <= 0;
64                          if (start) begin
65                              CS <= 0;
66                              SCLK_EN <= 1;
67                              bit_cnt <= 0;
68                              shifter_send <= data_in;
69                              shifter_recv <= 0;
70                              state <= TRANSFER;
71                          end
72                      end
73                  end
74
75                  TRANSFER: begin
76                      if (bit_cnt < 32)begin
77                          if (SCLK_falling) begin
78                              MOSI <= shifter_send[31];
79                          end
80                          if (SCLK_rising) begin
81                              shifter_recv <= {shifter_recv[30:0], MISO};
82                              shifter_send <= {shifter_send[30:0], 1'b0};
83                              bit_cnt <= bit_cnt + 1;
84
85                              if (bit_cnt == 31) begin
86                              state <= DONE;
87                              SCLK_EN <= 0; // close SCLK
88                              end
89                          end
90                      end
91                  end
92
93                  DONE: begin
```

SPI_Master_32bit

```
89                      end
90                  end
91              end
92
93              DONE: begin
94                  CS <= 1;
95                  SCLK_EN <= 0;
96                  MOSI <= 0;
97                  data_out <= shifter_recv;
98                  state <= IDLE;
99              end
100         endcase
101     end
102 end
103 endmodule
104
```

SPI_Slave_32bit

Click here [SPI_Slave]

```v
module SPI_Slave_32bit(
    input wire clk,
    input wire reset,
    input wire SCLK,
    input wire CS,
    input wire MOSI,
    output reg MISO,
    output reg [31:0] data_out,
    input wire [31:0] data_in,
    output reg [31:0] shifter_recv,
    output reg [31:0] shifter_send
);

    // Previous Value os SCLK
    // detect SCLK edge
    reg SCLK_prev;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            SCLK_prev <= 0;
        end
        else begin
            SCLK_prev <= SCLK;
        end
    end
    wire SCLK_rising = (SCLK == 1 && SCLK_prev == 0);
    wire SCLK_falling = (SCLK == 0 && SCLK_prev == 1);
```

SPI_Slave_32bit

```
27      wire SCLK_rising = (SCLK == 1 && SCLK_prev == 0);
28      wire SCLK_falling = (SCLK == 0 && SCLK_prev == 1);
29
30
31      reg [5:0] bit_cnt;
32      reg [1:0] state;
33      parameter IDLE = 2'b00, TRANSFER = 2'b01, DONE = 2'b10;
34
35
36      always @(posedge clk or posedge reset) begin
37          if (reset) begin
38              bit_cnt <= 0;
39              shifter_recv <= 0;
40              shifter_send <= 0;
41              data_out <= 0;
42              MISO <= 0;
43              state <= IDLE;
44          end
45          else begin
46              case (state)
47                  IDLE: begin
48                      if (!CS) begin // CS is active low
49                          bit_cnt <= 0;
50                          shifter_recv <= 0;
51                          shifter_send <= data_in;
```

SPI_Slave_32bit

```
                shifter_send <= data_in;
                state <= TRANSFER;
            end
        end

        TRANSFER: begin
            if (!CS) begin
                if (SCLK_falling) begin
                    // Prepare MISO on falling edge
                    MISO <= shifter_send[31];
                end

                if (SCLK_rising) begin
                    // Capture MOSI on rising edge
                    shifter_recv <= {shifter_recv[30:0], MOSI};
                    shifter_send <= {shifter_send[30:0], 1'b0};
                    bit_cnt <= bit_cnt + 1;

                    if (bit_cnt == 31)
                        state <= DONE;
                end
            end

            DONE: begin
                data_out <= shifter_recv;
                state <= IDLE;
            end
        endcase
    end
end

endmodule
```

SPI_Top_Module

Click here SPI_Top_Module

```
module SPI_Top_Module (
    input wire clk,
    input wire reset,
    input wire start,
    input wire [31:0] data_in,
    output wire [31:0] data_out,
    input wire MISO,
    output wire MOSI,
    output wire SCLK,
    output wire CS
);

    // Signals for connecting the Master and Slave
    wire [31:0] data_from_master;
    wire [31:0] data_to_master;

    // Instantiate the SPI Master module
    SPI_Master_32bit master (
        .clk(clk),
        .reset(reset),
        .start(start),
        .data_in(data_in),
        .data_out(data_from_master),
        .MISO(MISO),
        .MOSI(MOSI),
        .SCLK(SCLK),
        .CS(CS),
        .shifter_send(),
        .shifter_recv()
    );
```

SPI_Top_Module

```
30      );
31
32      // Instantiate the SPI Slave module
33      SPI_Slave_32bit slave (
34          .clk(clk),
35          .reset(reset),
36          .SCLK(SCLK),
37          .CS(CS),
38          .MOSI(MOSI),
39          .MISO(MISO),
40          .data_out(data_out),
41          .data_in(data_to_master),
42          .shifter_recv(),
43          .shifter_send()
44      );
45
46      // Connecting the Master data_out to the Slave data_in
47      assign data_to_master = data_from_master;
48
49  endmodule
50
```

SPI_Testbench

Click here [SPI_Testbench]

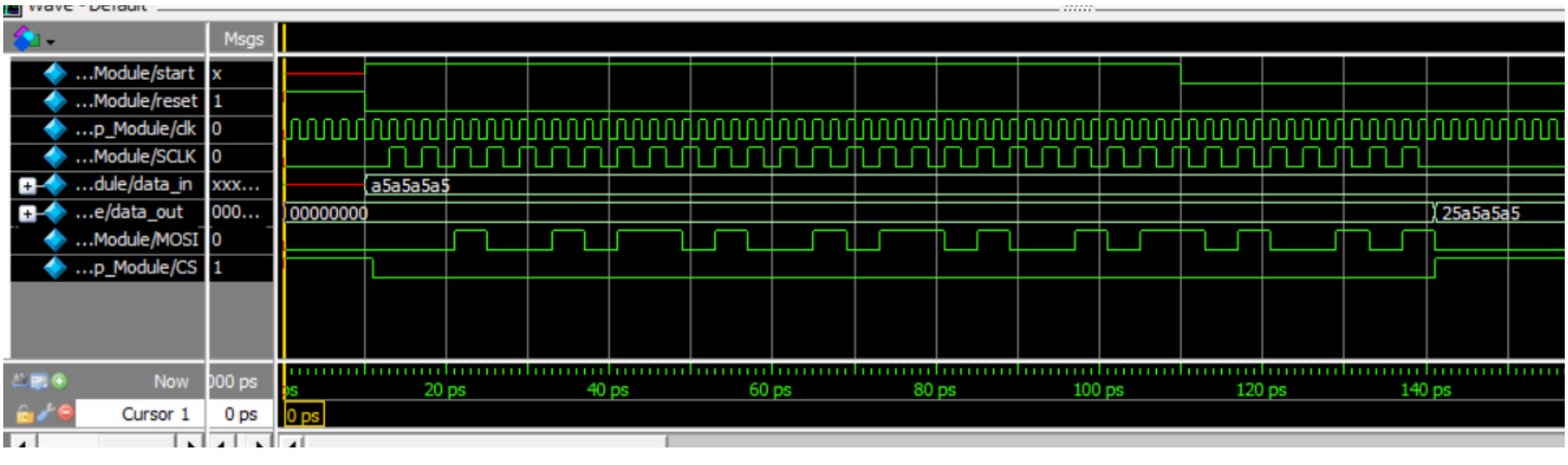
```
1
2  module tb_SPI_Top_Module;
3      reg clk;
4      reg reset;
5      reg start;
6      reg [31:0] data_in;
7      wire [31:0] data_out;
8      wire MOSI;
9      wire SCLK;
10     wire CS;
11     wire MISO;
12     // Clock generation
13     initial begin
14         clk = 0;
15         forever #1 clk = ~clk; // 100MHz clock
16     end
17
18     // Instantiate the Top Module
19     SPI_Top_Module uut (
20         .clk(clk),
21         .reset(reset),
22         .start(start),
23         .data_in(data_in),
24         .data_out(data_out),
25         .MOSI(MOSI),
26         .MISO(MISO),
27         .SCLK(SCLK),
28         .CS(CS)
29     );
30
```

SPI_Testbench

```
    );

    // Stimulus
    initial begin
        // Initial values
        reset = 1;
        #10 start=1;
        data_in = 32'hA5A5A5A5;
        reset = 0;
        start = 1;
        // Send data
        $display("Time | clk | rst | start | CS | SCLK | MOSI | MISO | Master_In | Master_Out");
        $monitor("%4t | %b   | %b   | %b     | %b  | %b    | %b    | %b    | %h       | %h",
                 $time, clk, reset, start, uut.CS, uut.SCLK, uut.MOSI, uut.MISO, uut.data_in, uut.data_out);
        #100;
        start = 0;

    end

endmodule
```

SPI_testbench

Click here [Clear_Photo]



It takes 32 SCLK Cycle then upload Output

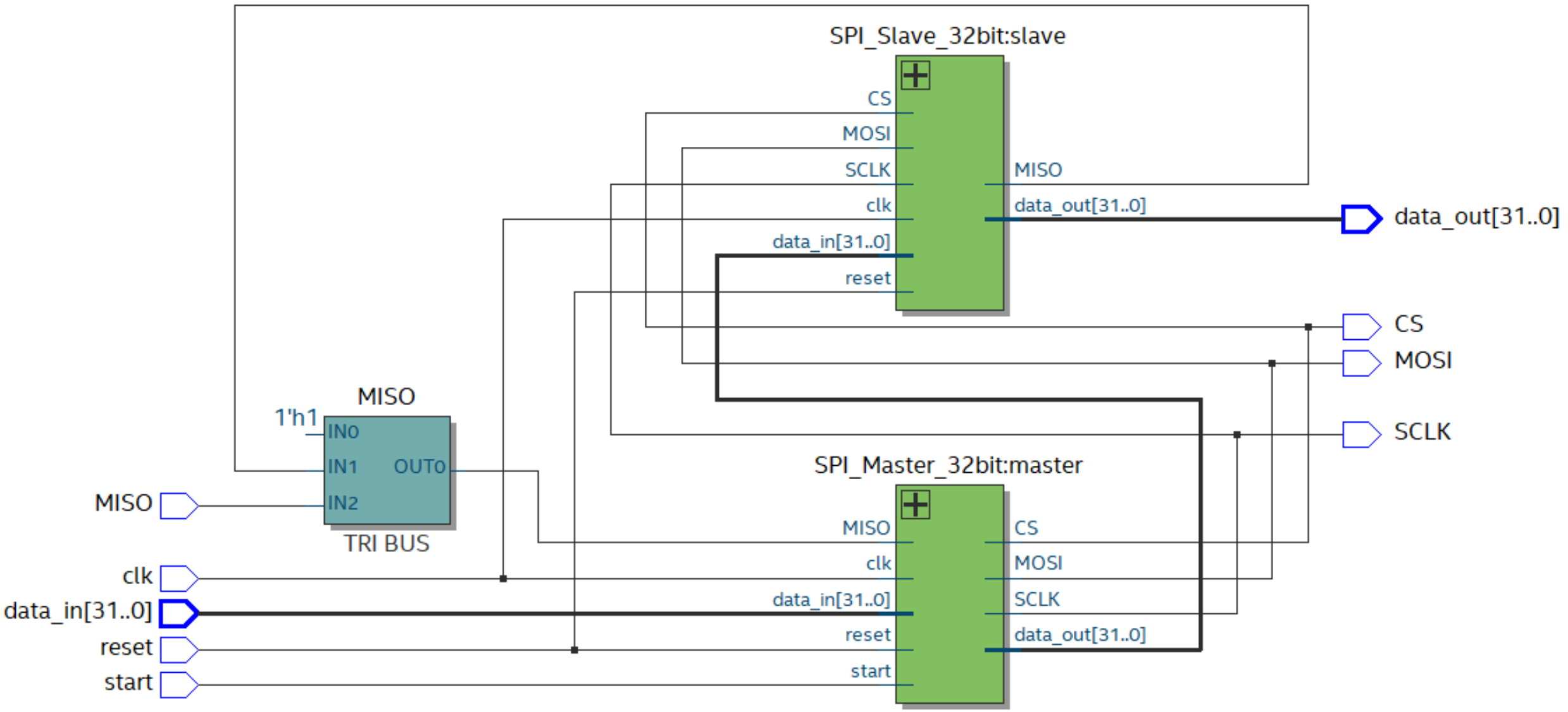
SPI_testbench

Click here [Transcript_Modelsim]

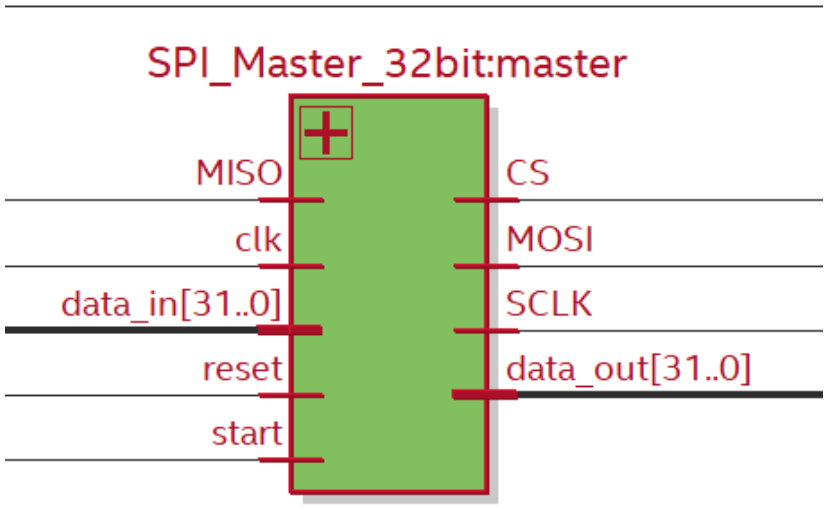
Time	clk	rst	start	CS	SCLK	MOSI	MISO	Master_In	Master_Out
10	0	0	1	1	0	0	0	a5a5a5a5	00000000
11	1	0	1	0	0	0	0	a5a5a5a5	00000000
12	0	0	1	0	0	0	0	a5a5a5a5	00000000
13	1	0	1	0	1	0	0	a5a5a5a5	00000000
14	0	0	1	0	1	0	0	a5a5a5a5	00000000
15	1	0	1	0	0	0	0	a5a5a5a5	00000000
16	0	0	1	0	0	0	0	a5a5a5a5	00000000
17	1	0	1	0	1	0	0	a5a5a5a5	00000000
18	0	0	1	0	1	0	0	a5a5a5a5	00000000
138	0	0	0	0	1	1	0	a5a5a5a5	00000000
139	1	0	0	0	0	1	0	a5a5a5a5	00000000
140	0	0	0	0	0	1	0	a5a5a5a5	00000000
141	1	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
142	0	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
143	1	0	0	1	0	0	0	a5a5a5a5	25a5a5a5

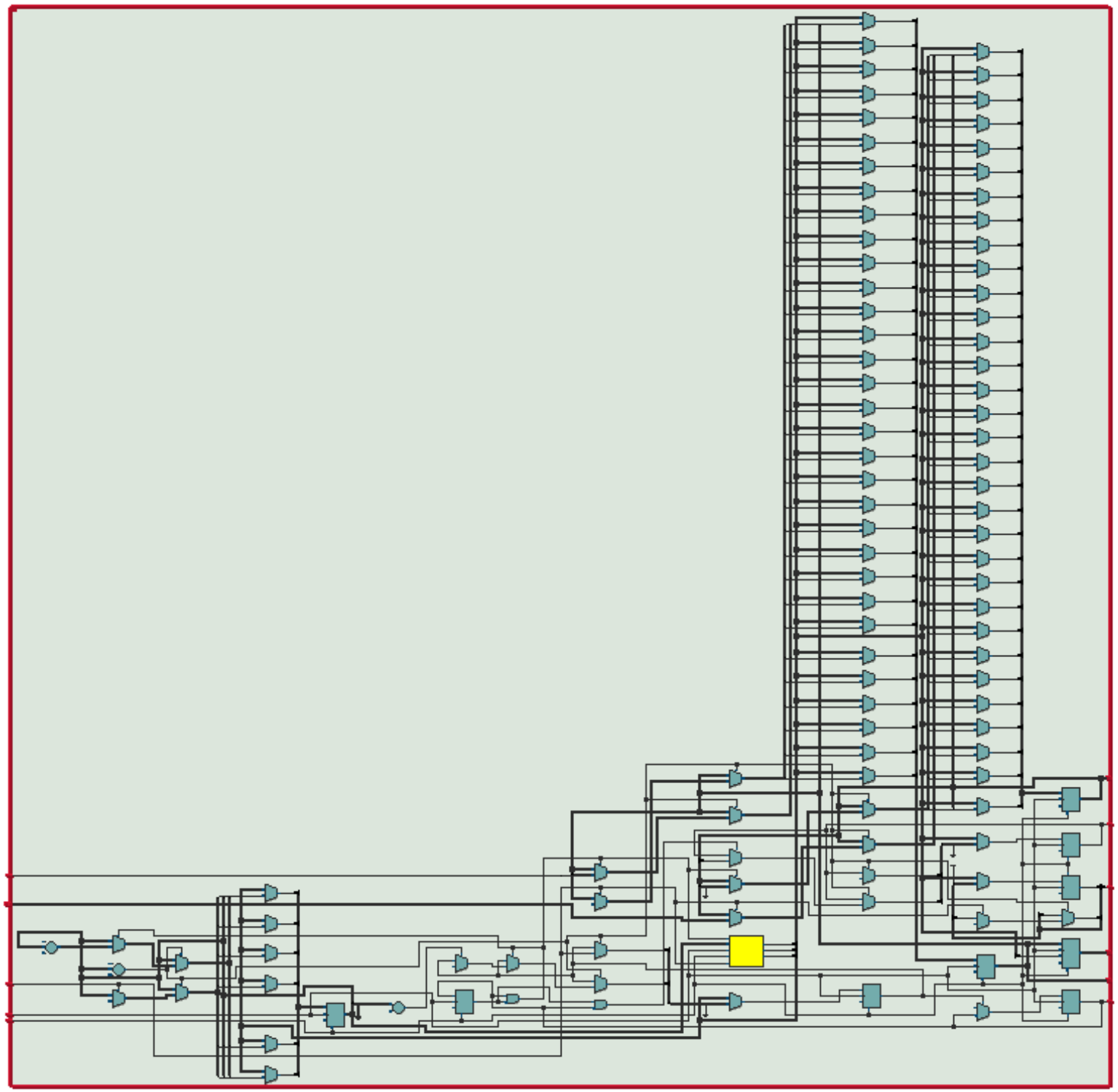
#									
# 989	1	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
# 990	0	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
# 991	1	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
# 992	0	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
# 993	1	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
# 994	0	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
# 995	1	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
# 996	0	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
# 997	1	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
# 998	0	0	0	1	0	0	0	a5a5a5a5	25a5a5a5
# 999	1	0	0	1	0	0	0	a5a5a5a5	25a5a5a5

RTL

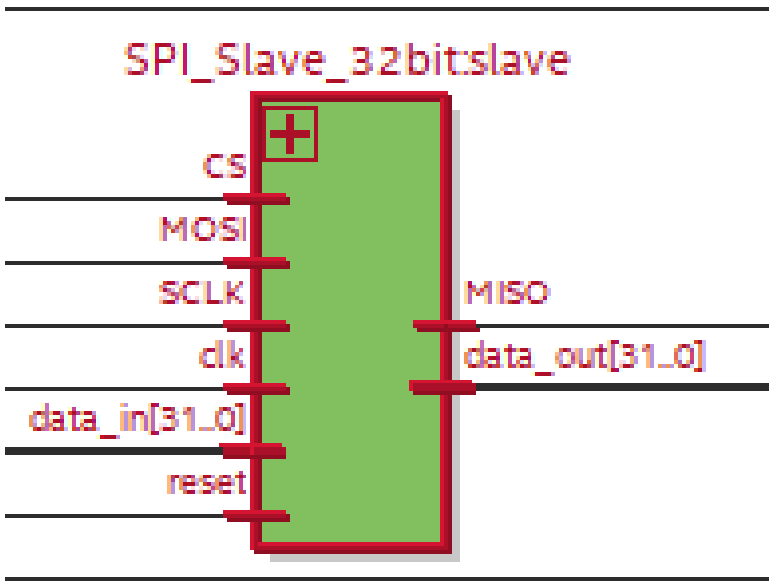

SPI_Slave_32bit:slave
CS
MOSI
SCLK
clk
data_in[31..0]
reset
MISO
data_out[31..0]
data_out[31..0]
CS
MOSI
SCLK
MISO
1'h1
IN0
IN1
IN2
OUT0
TRI BUS
MISO
clk
data_in[31..0]
reset
start
SPI_Master_32bit:master
MISO
clk
data_in[31..0]
reset
start
CS
MOSI
SCLK
data_out[31..0]

RTL


SPI_Master_32bit:master
MISO
clk
data_in[31..0]
reset
start
CS
MOSI
SCLK
data_out[31..0]



RTL


SPI_Slave_32bit:slave
CS
MOSI
SCLK
clk
data_in[31..0]
reset
MISO
data_out[31..0]