Project Report: 16-bit SPI Interface State Machine

A Digital Logic Designer

December 4, 2025

Abstract

This report details the design, implementation, and verification of a 16-bit Serial Peripheral Interface (SPI) master transmitter implemented in Verilog HDL. The design utilizes a three-state Finite State Machine (FSM) to manage the serialization of 16-bit input data and drive the standard SPI signals: Chip Select (CS), Serial Clock (SCLK), and Master Out Slave In (MOSI). The implemented FSM states are IDLE, TRANSFER, and DONE. Verification was performed using a dedicated testbench to transmit a sequence of hexadecimal data words.

1 Introduction

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification commonly used for short-distance, master-slave communication in embedded systems. This project focuses on developing the digital logic for an SPI master transmitter capable of sending 16 bits of data per transaction. The core of the implementation is a state machine that controls the timing and sequencing of the data transmission.

2 Design Specification

2.1 Interface Ports

The top-level module, SPI_state, exposes the following ports:

- clk: System clock input.
- rst: Asynchronous active-low reset.
- data_in [15:0]: The 16-bit data word to be transmitted.
- spi_cs: Output, Chip Select (active low).
- spi_sclk: Output, Serial Clock.
- spi_data: Output, Master Out Slave In (MOSI).
- counter [4:0]: Output, 5-bit counter for tracking bit position.

2.2 State Machine Logic

The FSM is defined by three local parameters: IDLE $= 0$, TRANSFER $= 1$, and DONE $= 2$. The transitions are governed by the system clock (clk) and the internal counter (count). The state transitions are as follows:

1. **IDLE State (00):**

- Initializes sclk to logic low ('0').
- cs is asserted high (1) if count is 16 (indicating the start of a new, complete transaction), otherwise it is kept low.
- The FSM transitions to the TRANSFER state.

2. **TRANSFER State (01):**
   - sclk is kept low ('0').
   - cs is asserted low ('0') to keep the slave selected.
   - The MOSI line is driven with the current data bit: data_in [count - 1].
   - The count register is decremented (count $\leftarrow$ count $-$ 1).
   - The FSM transitions to the DONE state.
3. **DONE State (10):**
   - sclk is asserted high ('1'), completing the clock cycle and latching the bit on the slave side (assuming CPOL=0, CPHA=1).
   - If count $>$ 0, there are more bits to send, and the FSM transitions back to the IDLE state to prepare the next bit.
   - If count $=$ 0, all 16 bits have been sent. The count is reset to 16, and the FSM transitions back to IDLE to wait for the next data word.

3 Implementation (Verilog HDL)

The design is implemented in the Verilog module SPI_state.v. The full code is provided in Appendix A.1.

3.1 Code Snippet: State Logic

The heart of the design is the clocked sequential logic, as shown in the case statement below:

```
    else begin
        case (state)
            IDLE: begin
                sclk  <= 1'b0;
                cs    <= count == 16;
                state <= TRANSFER;
            end

            TRANSFER: begin
                sclk  <= 1'b0;
                cs    <= 1'b0;
                MOSI  <= data_in [count - 1];
                count <= count - 1;
                state <= DONE;
            end

            DONE: begin
                sclk <= 1'b1;
                if (count > 0) state <= IDLE;
                else begin
                    count <= 16;
                    state <= IDLE;
                end
            end
```

```
25
26              default: state <= IDLE;
27          endcase
28      end
```

Listing 1: Core State Machine Logic from `SPI_state.v`

4 Verification and Testbench

4.1 Testbench Description

Verification was performed using the spi_state_tb.v testbench. The testbench instantiated the Device Under Test (dut) and generated the required stimulus signals.

- **Clock Generation:** A clk signal with a period of 10ns (5ns high, 5ns low) was generated.
- **Reset Sequence:** The rst signal was held low initially and asserted high after 10ns.
- **Data Input:** A sequence of 16-bit hexadecimal values was applied to the data_in port, with each new word applied after a period of 480ns, which is sufficient time to complete a full 16-bit transmission (16bits $\times$ 3clocks/bit $\times$ 10ns/clock $\approx$ 480ns).

The sequence of transmitted data words is: 16'hA569, 16'h2563, 16'h9B63, 16'h6A61, 16'hA265, and 16'h7564.

4.2 Testbench Code Snippet

```
1  initial begin
2      #10 rst=1;
3
4           data_in = 16'hA569;
5      #480 data_in = 16'h2563;
6      #480 data_in = 16'h9B63;
7      #480 data_in = 16'h6A61;
8      #480 data_in = 16'hA265;
9      #480 data_in = 16'h7564;
10
11     $finish;
12 end
```

Listing 2: Data Stimulus from `spi_state_tb.v`

5 Conclusion

The 16-bit SPI master transmitter was successfully designed using a three-state FSM (IDLE, TRANSFER, DONE). The design ensures that each of the 16 bits of data is correctly outputted on the spi_data (MOSI) line, synchronized with the spi_sclk signal, and contained within the assertion period of the spi_cs signal. The testbench demonstrated the transmission of multiple data packets, confirming the functional correctness of the design.

A Verilog Source Code

A.1 SPI_state.v

```
module SPI_state (
    input  wire        clk,
    input  wire        rst,
    input  wire [15:0] data_in,

    output wire        spi_cs,
    output wire        spi_sclk,
    output wire        spi_data, // MISO
    output wire [04:0] counter
);

localparam [1:0] IDLE     = 0;
localparam [1:0] TRANSFER = 1;
localparam [1:0] DONE     = 2;

//reg [15:0] MOSI;
reg        MOSI;
reg [04:0] count;
reg        cs;
reg        sclk;
reg [01:0] state;

always @ (posedge clk or negedge rst) begin
    if (~rst) begin
        MOSI  <= 1'b0;
        count <= 05'd16;
        cs    <= 01'b1;
        sclk  <= 01'b0;
        state <= IDLE;
    end

    else begin
        case (state)
            IDLE: begin
                sclk  <= 1'b0;
                cs    <= count == 16;
                state <= TRANSFER;
            end

            TRANSFER: begin
                sclk  <= 1'b0;
                cs    <= 1'b0;
                MOSI  <= data_in [count - 1];
                count <= count - 1;
                state <= DONE;
            end

            DONE: begin
                sclk <= 1'b1;
                if (count > 0) state <= IDLE;
                else begin
                    count <= 16;
                    state <= IDLE;
                end
            end

            default: state <= IDLE;
        endcase
    end
```

```
end

assign spi_cs   = cs;
assign spi_sclk = sclk;
assign spi_data = MOSI;
assign counter = count;

endmodule
```

Listing 3: SPI State Machine Module (`SPI_state.v`)

A.2 `spi_state_tb.v`

```
`timescale 1ns / 1ps

module spi_state_tb;

reg         clk;
reg         rst;
reg  [15:0] data_in;

wire        spi_cs;
wire        spi_sclk;
wire        spi_data;
wire [04:0] counter;

SPI_state dut (
    .clk (clk),
    .rst (rst),
    .counter (counter),
    .data_in (data_in),
    .spi_cs (spi_cs),
    .spi_sclk (spi_sclk),
    .spi_data (spi_data)
);

initial begin
    clk     = 0;
    rst     = 0;
    data_in = 0;
end

always #5 clk = ~clk;

initial begin
    #10 rst=1;

         data_in = 16'hA569;
    #480 data_in = 16'h2563;
    #480 data_in = 16'h9B63;
    #480 data_in = 16'h6A61;
    #480 data_in = 16'hA265;
    #480 data_in = 16'h7564;

    $finish;
end

endmodule
```

Listing 4: SPI State Machine Testbench (`spi_state_tb.v`)