SPI PROTOCOL VERILOG CODE AND TESTBENCH

SPI MASTER AND SLAVE

```
module fsm_spi(
   input wire clk,
   input wire rst,
   input wire tx_enable,
   output reg mosi,
   output reg cs,
   output wire sclk
);

   typedef enum logic [1:0] {idle = 0, start_tx = 1, tx_data = 2, end_tx = 3 } state_type;
   state_type state, next_state;



   reg [7:0] din = 8'b10101010;

   reg spi_sclk = 0;
   reg [2:0] ccount = 0;
   reg [2:0] count = 0; // 0 -7
   integer bit_count = 0;



  /////////////////generating sclk
  always@(posedge clk)
    begin
    case(next_state)
    idle:
    begin
    spi_sclk <= 0;
    end
```

```
start_tx :
begin
if(count < 3'b011 || count == 3'b111)
spi_sclk <= 1'b1;
else
spi_sclk <= 1'b0;
end

tx_data:
begin
if(count < 3'b011 || count == 3'b111)
spi_sclk <= 1'b1;
else
spi_sclk <= 1'b0;
end

end_tx:
begin
if(count < 3'b011 )
spi_sclk <= 1'b1;
else
spi_sclk <= 1'b0;
end

default : spi_sclk <= 1'b0;

endcase
end
```

```
//////////////////////sense reset

    always@(posedge clk)
    begin
     if(rst)
      state <= idle;
     else
      state <= next_state;
    end



//////////////////////next_State decoder
    always@(*)
    begin
    case(state)

    idle :
    begin
          mosi = 1'b0;
          cs   = 1'b1;
          if(tx_enable)
          next_state = start_tx;
          else
          next_state = idle;
    end

    start_tx:
    begin
        cs = 1'b0;

        if(count == 3'b111)
```

```
    next_state = tx_data;
    else
    next_state = start_tx;
end

tx_data :
begin
   mosi = din[7-bit_count];
   if(bit_count != 8) begin
   next_state = tx_data;
   end
   else
   begin
   next_state = end_tx;
   mosi  = 1'b0;
   end
end

end_tx:
begin
  cs   = 1'b1;
  mosi = 1'b0;
  if(count == 3'b111)
    next_state = idle;
  else
    next_state = end_tx;
end

default :  next_state = idle;
endcase
end
```

```
//////////////////////counter

always@(posedge clk)
begin
case(state)

idle :
begin
count <= 0;
bit_count <= 0;
end



start_tx : count <= count + 1;




tx_data:
begin
if(bit_count != 8)
      begin
        if(count < 3'b111)
          count <= count + 1;
        else
          begin
          count <= 0;
          bit_count <= bit_count + 1;
          end
      end
end
```

```
end_tx :
begin
count    <= count + 1;
bit_count <= 0;
end

default :
begin
count <= 0;
bit_count <= 0;
end

endcase
end
```

```
////////////////////////////////////////////////////////////////////////////////

 assign sclk = spi_sclk;

endmodule
```

```
//////////////////////////////////////////////////
```

```
////////////////////////////////////////////// slave

module spi_slave (
input sclk, mosi,cs,
output [7:0] dout,
output reg done
);

integer count = 0;
typedef enum logic [1:0] {idle = 0, sample = 1 } state_type;
state_type state;
reg [7:0] data = 0;


always@(negedge sclk)
begin
case (state)

idle: begin
done <= 1'b0;
```

```
if(cs == 1'b0)
state <= sample;
else
state <= idle;
end

sample:
begin
    if(count < 8)
    begin
    count <= count + 1;
    data <= {data[6:0],mosi};
    state <= sample;
    end
    else
    begin
    count <= 0;
    state <= idle;
    done  <= 1'b1;
    end
end

default : state <= idle;
endcase

end

assign dout = data;

endmodule
```

```
//////////////////////////////////////////

module top
(
input clk, rst, tx_enable,
output [7:0] dout,
output done
);

wire mosi, ss, sclk;

fsm_spi    spi_m (clk, rst, tx_enable, mosi, ss, sclk);
spi_slave  spi_s (sclk, mosi,ss, dout, done);

endmodule
```

```
//////////////////////////////////////

module tb;

   reg clk = 0;
   reg rst = 0;
   reg tx_enable = 0;
   wire [7:0] dout;
```

```
    always #5 clk = ~clk;

    initial begin
    rst = 1;
    repeat(5) @(posedge clk);
    rst = 0;
    end

    initial begin
    tx_enable = 0;
    repeat(5) @(posedge clk);
    tx_enable = 1;
    end

top dut (clk, rst, tx_enable, dout);

endmodule
```