DAY-88
#100DAYSRTL

“Aim”:-Verification of SPI Protocol using System Verilog

“TestBench Codes”:-

Design Code:-

```
module spi_master(
input clk, newd,rst,
input [11:0] din,
output reg sclk,cs,mosi
    );
  typedef enum bit [1:0] {idle = 2'b00, enable = 2'b01, send = 2'b10, comp = 2'b11 } state_type;
  state_type state = idle;
  int countc = 0;
  int count = 0;
  //////////////////////////generation of sclk
 always@(posedge clk)
  begin
    if(rst == 1'b1) begin
      countc <= 0;
      sclk <= 1'b0;
    end
    else begin
      if(countc < 10 )
          countc <= countc + 1;
      else
          begin
          countc <= 0;
          sclk <= ~sclk;
          end
    end
  end
  ///////////////////state machine
    reg [11:0] temp;
  always@(posedge sclk)
  begin
    if(rst == 1'b1) begin
      cs <= 1'b1;
      mosi <= 1'b0;
    end
    else begin
     case(state)
         idle:
             begin
               if(newd == 1'b1) begin
                 state <= send;
                 temp <= din;
                 cs <= 1'b0;
               end
               else begin
                 state <= idle;
                 temp <= 8'h00;
               end
             end
       send : begin
         if(count <= 11) begin
           mosi <= temp[count]; /////sending lsb first
           count <= count + 1;
         end
         else
             begin
               count <= 0;
               state <= idle;
               cs <= 1'b1;
               mosi <= 1'b0;
             end
       end
      default : state <= idle;
   endcase
  end
 end
endmodule
```

```
module spi_slave (
input sclk, cs, mosi,
output [11:0] dout,
output reg done
);
typedef enum bit {detect_start = 1'b0, read_data = 1'b1} state_type;
state_type state = detect_start;
reg [11:0] temp = 12'h000;
int count = 0;
always@(posedge sclk)
begin
case(state)
detect_start:
begin
done   <= 1'b0;
if(cs == 1'b0)
 state <= read_data;
 else
 state <= detect_start;
end
read_data : begin
if(count <= 11)
 begin
 count <= count + 1;
 temp  <= { mosi, temp[11:1]};
 end
 else
 begin
 count <= 0;
 done <= 1'b1;
 state <= detect_start;
 end
end
endcase
end
assign dout = temp;
endmodule
////////////////////////////////
module top (
input clk, rst, newd,
input [11:0] din,
output [11:0] dout,
output done
);
wire sclk, cs, mosi;
spi_master m1 (clk, newd, rst, din, sclk, cs, mosi);
spi_slave s1  (sclk, cs, mosi, dout, done);
endmodule
```

“Interface”:-

```
interface spi_if;
 logic clk, rst, newd;
  logic [11:0] din,dout;
  logic done;
  logic sclk;
endinterface
```

“Transaction”:-

```
class transaction;
  bit newd;                    // Flag for new transaction
  rand bit [11:0] din;         // Random 12-bit data input
  bit [11:0] dout;             // 12-bit data output
  function transaction copy();
    copy = new();              // Create a copy of the transaction
    copy.newd = this.newd;     // Copy the newd flag
    copy.din  = this.din;      // Copy the data input
    copy.dout = this.dout;     // Copy the data output
  endfunction
endclass
```

“Generator”:-

```
class generator;
  transaction tr;              // Transaction object
  mailbox #(transaction) mbx; // Mailbox for transactions
  event done;                  // Done event
  int count = 0;               // Transaction count
  event drvnext;               // Event to synchronize with driver
  event sconext;               // Event to synchronize with scoreboard
  function new(mailbox #(transaction) mbx);
    this.mbx = mbx;            // Initialize mailbox
    tr = new();                // Create a new transaction
  endfunction
  task run();
    repeat(count) begin
      assert(tr.randomize) else $error("[GEN] :Randomization Failed");
      mbx.put(tr.copy);        // Put a copy of the transaction in the mailbox
      $display("[GEN] : din : %0d", tr.din);
      @(sconext);              // Wait for the scoreboard synchronization event
    end
    -> done;              // Signal when done
  endtask
endclass
```

“Driver”:-

```
class driver;
  virtual spi_if vif;          // Virtual interface
  transaction tr;              // Transaction object
  mailbox #(transaction) mbx; // Mailbox for transactions
  mailbox #(bit [11:0]) mbxds; // Mailbox for data output to monitor
  event drvnext;               // Event to synchronize with generator
  bit [11:0] din;              // Data input
  function new(mailbox #(bit [11:0]) mbxds, mailbox #(transaction) mbx);
    this.mbx = mbx;            // Initialize mailboxes
    this.mbxds = mbxds;
  endfunction
  task reset();
    vif.rst <= 1'b1;           // Set reset signal
    vif.newd <= 1'b0;          // Clear new data flag
    vif.din <= 1'b0;           // Clear data input
    repeat(10) @(posedge vif.clk);
    vif.rst <= 1'b0;           // Clear reset signal
    repeat(5) @(posedge vif.clk);
    $display("[DRV] : RESET DONE");
    $display("----------------------------------------");
  endtask
  task run();
    forever begin
      mbx.get(tr);             // Get a transaction from the mailbox
      vif.newd <= 1'b1;        // Set new data flag
      vif.din <= tr.din;       // Set data input
      mbxds.put(tr.din);       // Put data in the mailbox for the monitor
      @(posedge vif.sclk);
      vif.newd <= 1'b0;        // Clear new data flag
      @(posedge vif.done);
      $display("[DRV] : DATA SENT TO DAC : %0d",tr.din);
      @(posedge vif.sclk);
    end
  endtask
endclass
```

“Monitor”:-

```
class monitor;
  transaction tr;              // Transaction object
  mailbox #(bit [11:0]) mbx; // Mailbox for data output
  virtual spi_if vif;          // Virtual interface
  function new(mailbox #(bit [11:0]) mbx);
    this.mbx = mbx;            // Initialize the mailbox
  endfunction
  task run();
    tr = new();                // Create a new transaction
    forever begin
      @(posedge vif.sclk);
      @(posedge vif.done);
      tr.dout = vif.dout;      // Record data output
      @(posedge vif.sclk);
      $display("[MON] : DATA SENT : %0d", tr.dout);
      mbx.put(tr.dout);        // Put data in the mailbox
    end
  endtask
endclass
```

“Scoreboard”:-

```
class scoreboard;
  mailbox #(bit [11:0]) mbxds, mbxms; // Mailboxes for data from driver and monitor
  bit [11:0] ds;                      // Data from driver
  bit [11:0] ms;                      // Data from monitor
  event sconext;                      // Event to synchronize with environment
  function new(mailbox #(bit [11:0]) mbxds, mailbox #(bit [11:0]) mbxms);
    this.mbxds = mbxds;               // Initialize mailboxes
    this.mbxms = mbxms;
  endfunction
  task run();
    forever begin
      mbxds.get(ds);                  // Get data from driver
      mbxms.get(ms);                  // Get data from monitor
      $display("[SCO] : DRV : %0d MON : %0d", ds, ms);
      if(ds == ms)
        $display("[SCO] : DATA MATCHED");
      else
        $display("[SCO] : DATA MISMATCHED");
      $display("-----------------------------------------");
      ->sconext;                       // Synchronize with the environment
    end
  endtask
endclass
```

“Environment”:-

```
class environment;
    generator gen;                    // Generator object
    driver drv;                       // Driver object
    monitor mon;                      // Monitor object
    scoreboard sco;                   // Scoreboard object
    event nextgd;                     // Event for generator to driver communication
    event nextgs;                     // Event for generator to scoreboard communication
    mailbox #(transaction) mbxgd;     // Mailbox for generator to driver communication
    mailbox #(bit [11:0]) mbxds;      // Mailbox for driver to monitor communication
    mailbox #(bit [11:0]) mbxms;      // Mailbox for monitor to scoreboard communication
    virtual spi_if vif;               // Virtual interface
  function new(virtual spi_if vif);
    mbxgd = new();                    // Initialize mailboxes
    mbxms = new();
    mbxds = new();
    gen = new(mbxgd);                 // Initialize generator
    drv = new(mbxds,mbxgd);           // Initialize driver
    mon = new(mbxms);                 // Initialize monitor
    sco = new(mbxds, mbxms);          // Initialize scoreboard
    this.vif = vif;
    drv.vif = this.vif;
    mon.vif = this.vif;
    gen.sconext = nextgs;             // Set synchronization events
    sco.sconext = nextgs;
    gen.drvnext = nextgd;
    drv.drvnext = nextgd;
  endfunction
  task pre_test();
    drv.reset();                      // Perform driver reset
  endtask
  task test();
  fork
    gen.run();                        // Run generator
    drv.run();                        // Run driver
    mon.run();                        // Run monitor
    sco.run();                        // Run scoreboard
  join_any
  endtask
  task post_test();
    wait(gen.done.triggered);         // Wait for generator to finish
    $finish();
  endtask
  task run();
    pre_test();
    test();
    post_test();
  endtask
endclass
```

“Tb_Top”:-

```
module tb;
  spi_if vif();                    // Virtual interface instance
  top dut(vif.clk,vif.rst,vif.newd,vif.din,vif.dout,vif.done);
  initial begin
    vif.clk <= 0;
  end
  always #10 vif.clk <= ~vif.clk;
  environment env;
  assign vif.sclk = dut.m1.sclk;
  initial begin
    env = new(vif);
    env.gen.count = 4;
    env.run();
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end
endmodule
```

“Result”:-

```
[GEN] : din : 935
[DRV] : DATA SENT TO DAC : 935
[MON] : DATA SENT : 935
[SCO] : DRV : 935 MON : 935
[SCO] : DATA MATCHED
----------------------------------------
[GEN] : din : 4060
[DRV] : DATA SENT TO DAC : 4060
[MON] : DATA SENT : 4060
[SCO] : DRV : 4060 MON : 4060
[SCO] : DATA MATCHED
----------------------------------------
[GEN] : din : 3576
[DRV] : DATA SENT TO DAC : 3576
[MON] : DATA SENT : 3576
[SCO] : DRV : 3576 MON : 3576
[SCO] : DATA MATCHED
```

“Reference”:-

- **System Verilog Part 2 by Kumar khandagle**

https://www.udemy.com/share/106khe3@wHUiu4HBH8vEUP7kTv4UtZCU-HXFm2GT7WqNKLFZDfS1He-3uRzOnUUjsOTEtxWBMg==/