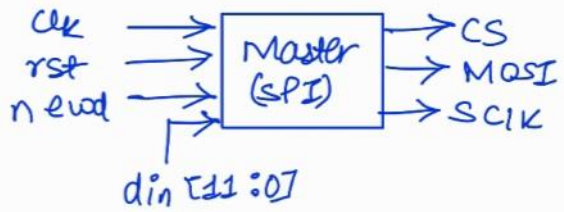
SPI Synchronous Serial communication Protocol

→Theory for SPI Master:

RITU SREE MAHAPATRA

21PRCIR04

→ SPI Design specifications:-



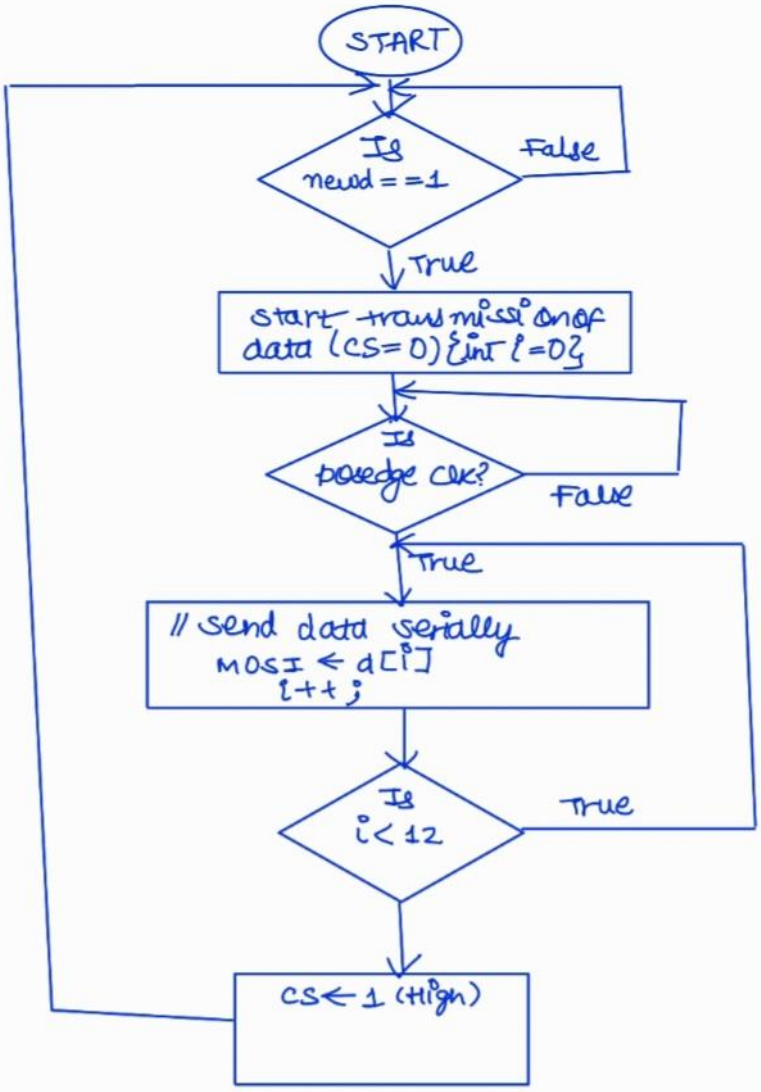
- Here input ports clk, rst, newd, din[11:0] & output ports CS, MOSI, SCLK
- Global signals: clk, rst.
- "newd" referred as new data signal wiz used to indicate when user have new data which it need to transmit to the device through an SPI transaction.

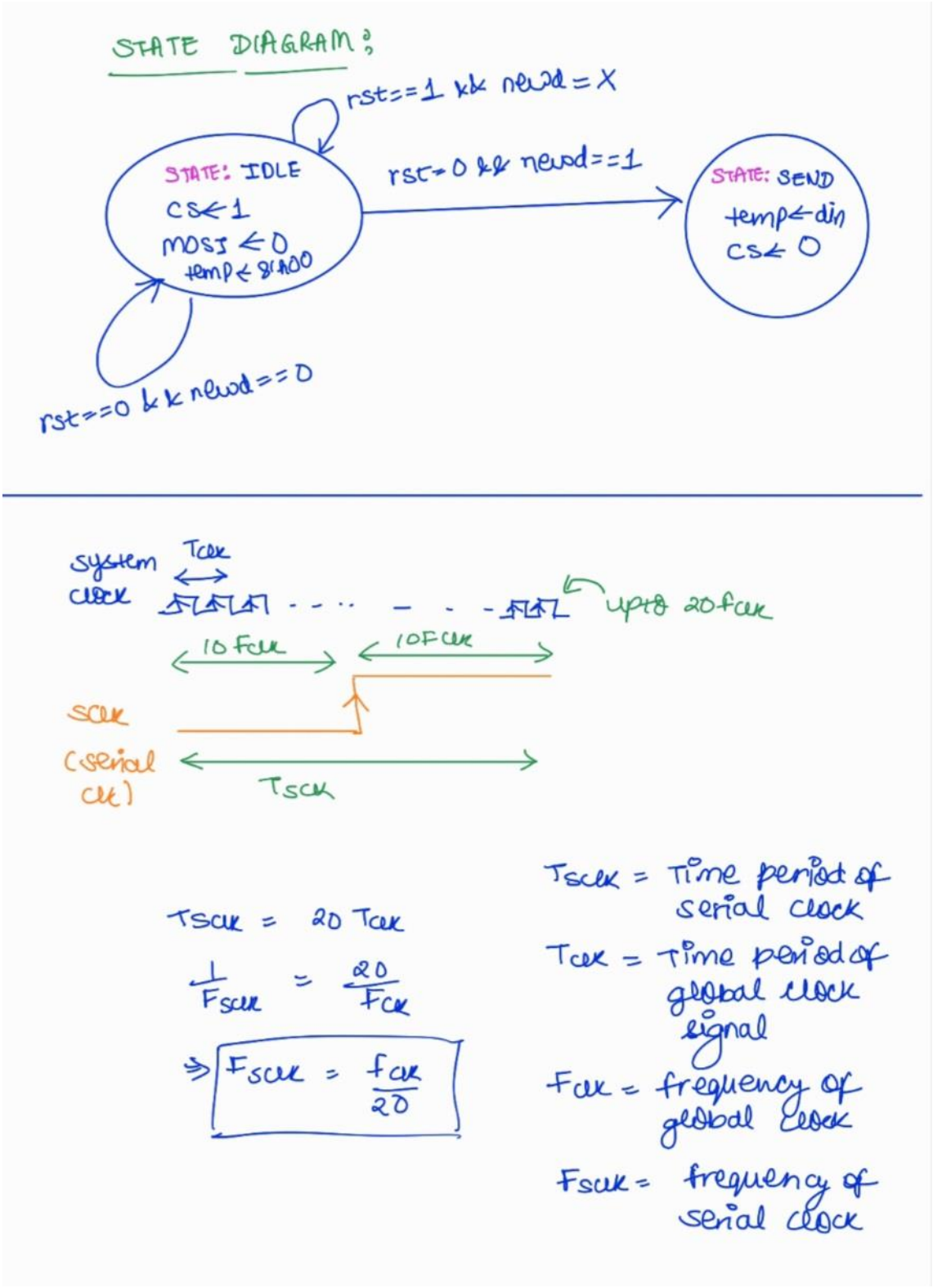
  So, as soon we have new data our device will take the data from din[11:0] bus & generate the respective transaction on an output bus.
- "CS" is used to enable our slave device. an active low on this pin will start a transaction.
- "MOSI" is the pin used to transmit the data serially from master to slave & "sclk" is the serial clock which will be going to slave for synchronization.

→ OPERATION OF THIS DEVICE :

- we will wait till user have a new data AS soon as user make "newd" signal high we will be sampling the data that we have on an din bus & start transmitting it to the slave device. IDLE value for cs is 1 (high). So when user wish to start a transaction, it conveys to the slave by making cs 0 (low). This is what is mean by starting a transaction. & from next clock tick onwards we start sending data serially one after another we wait till all the bits are sent serially as soon as we complete sending all the bits we will be ending our transaction by making cs high

FLOWCHART:—


START
Is
newd == 1
False
True
start transmission of
data (CS=0) {int i=0}
Is
posedge clk?
False
True
// send data serially
MOSI ← d[i]
i++;
Is
i < 12
True
CS ← 1 (High)



→Design Code for SPI Master:

```
1.  module spi(
2. input clk, newd,rst,
```

```
3. input [11:0] din,
4. output reg sclk,cs,mosi //since all of these are to be updated in a
   //procedural block itself hence we have used reg type
5.     );
6.
7.   typedef enum bit [1:0] {idle = 2'b00, enable = 2'b01, send = 2'b10,
8.  comp= 2'b11 } state_type;// enable state is for enabling transmission of
   //data and comp state is to represent completion of transmission of data
9.   state_type state = idle; //initial default state
10.
11.  int countc = 0;
12.  int count = 0;
13.
14.  //////////////////////////generation of sclk
15.////////sclk is usually 4X slower than the clk(global clock signal)
16. always@(posedge clk)
17.  begin
18.    if(rst == 1'b1) begin
19.      countc <= 0;
20.      sclk <= 1'b0;
21.    end
22.    else begin
23.      if(countc < 10 )   /// fclk / 20
24.          countc <= countc + 1;
25.      else
26.          begin
27.          countc <= 0;
28.          sclk <= ~sclk;// inverting sclk after 10 pulses of clk
29.          end
30.    end
31.  end
32.
33.  ////////////////////state machine
34.    reg [11:0] temp;
35.
36.  always@(posedge sclk)
37.  begin
38.    if(rst == 1'b1) begin
39.      cs <= 1'b1; //idle or default initial value of CS
40.      mosi <= 1'b0; //idle or default initial value of mosi
41.      temp<=8'h00;
42.    end
43.    else begin
44.     case(state)
45.         idle:
46.             begin
47.               if(newd == 1'b1) begin //if new data is there i.e user has
   //send new data then  sampling of data will happen
48.                 state <= send;
49.                 temp <= din; //din stored in temporary variable temp
50.                 cs <= 1'b0; //starting of transaction
51.               end
52.               else begin
53.                 state <= idle;
54.                 temp <= 8'h00;
55.               end
56.             end
57.
58.
```

```
       send : begin
//serially sending 12 bit of data on mosi
         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
//////////////////////////
interface spi_if;


  logic clk;
  logic newd;
  logic rst;
  logic [11:0] din;
  logic sclk;
  logic cs;
  logic mosi;


endinterface
```

→Verification code for SPI master:-

```
////////////////Transaction Class
class transaction;
//our target will be to verify din data that we apply to a DUT wheather we receive
//same data on mosi

  rand bit newd; //modifier rand is added for generating one bit random value for
//newd
  rand bit [11:0] din; // modifier rand is added for generating 12 bit random
values for din

```

```
8.    bit cs;
9.    bit mosi;
10.
11.   // Display function for debugging
12.   function void display (input string tag);
13.     $display("[%0s] : DATA_NEW : %0b DIN : %0d CS : %b MOSI : %0b ", tag, newd,
    din, cs, mosi);
14.   endfunction
15.
16.   // Transaction copy function
17.   function transaction copy();
18.     copy = new();
19.     copy.newd = this.newd;
20.     copy.din = this.din;
21.     copy.cs = this.cs;
22.     copy.mosi = this.mosi;
23.   endfunction
24.
25. endclass
26.
27.
28. ////////////////Generator Class
29. class generator;
30.
31.   transaction tr;
32.   mailbox #(transaction) mbx;
33.   event done;
34.   int count = 0;
35.   event drvnext;
36.   event sconext;
37.
38.   // Constructor
39.   function new(mailbox #(transaction) mbx);
40.     this.mbx = mbx;
41.     tr = new();
42.   endfunction
43.
44.   // Task to generate transactions
45.   task run();
46.     repeat(count) begin
47.       assert(tr.randomize) else $error("[GEN] :Randomization Failed");
48.       mbx.put(tr.copy);
49.       tr.display("GEN");
50.       @(drvnext);
51.       @(sconext);
52.     end
53.     -> done;
54.   endtask
55.
56. endclass
57.
58. ////////////////Driver Class
59.
60. class driver;
61.
62.   virtual spi_if vif;
63.   transaction tr;
64.   mailbox #(transaction) mbx;
65.   mailbox #(bit [11:0]) mbxds;
66.   event drvnext;
67.
68.   bit [11:0] din;
69.
70.   // Constructor
71.   function new(mailbox #(bit [11:0]) mbxds, mailbox #(transaction) mbx);
72.     this.mbx = mbx;
```

```
73.     this.mbxds = mbxds;
74.   endfunction
75.
76.   // Task to reset the driver
77.   task reset();
78.      vif.rst <= 1'b1;
79.      vif.cs <= 1'b1;
80.      vif.newd <= 1'b0;
81.      vif.din <= 1'b0;
82.      vif.mosi <= 1'b0;
83.     repeat(10) @(posedge vif.clk);
84.       vif.rst <= 1'b0;
85.     repeat(5) @(posedge vif.clk);
86.
87.     $display("[DRV] : RESET DONE");
88.     $display("----------------------------------------");
89.   endtask
90.
91.   // Task to drive transactions
92.   task run();
93.     forever begin
94.       mbx.get(tr);
95.       @(posedge vif.sclk);
96.       vif.newd <= 1'b1;
97.       vif.din <= tr.din;
98.       mbxds.put(tr.din);
99.       @(posedge vif.sclk);
100.             vif.newd <= 1'b0;
101.             wait(vif.cs == 1'b1);
102.             $display("[DRV] : DATA SENT TO DAC : %0d",tr.din);
103.             ->drvnext;
104.           end
105.         endtask
106.
107.       endclass
108.
109.       ////////////////Monitor Class
110.
111.       class monitor;
112.         transaction tr;
113.         mailbox #(bit [11:0]) mbx;
114.         bit [11:0] srx; // Received data
115.
116.         virtual spi_if vif;
117.
118.         // Constructor
119.         function new(mailbox #(bit [11:0]) mbx);
120.           this.mbx = mbx;
121.         endfunction
122.
123.         // Task to monitor the bus
124.         task run();
125.           forever begin
126.             @(posedge vif.sclk);
127.             wait(vif.cs == 1'b0); // Start of transaction
128.             @(posedge vif.sclk);
129.
130.             for (int i = 0; i <= 11; i++) begin
131.               @(posedge vif.sclk);
132.               srx[i] = vif.mosi;
133.             end
134.
135.             wait(vif.cs == 1'b1); // End of transaction
136.
137.             $display("[MON] : DATA SENT : %0d", srx);
138.             mbx.put(srx);
```

```
139.          end
140.        endtask
141.
142.      endclass
143.
144.      ////////////////Scoreboard Class
145.
146.      class scoreboard;
147.        mailbox #(bit [11:0]) mbxds, mbxms;
148.        bit [11:0] ds; // Data from driver
149.        bit [11:0] ms; // Data from monitor
150.        event sconext;
151.
152.        // Constructor
153.        function new(mailbox #(bit [11:0]) mbxds, mailbox #(bit [11:0]) mbxms);
154.          this.mbxds = mbxds;
155.          this.mbxms = mbxms;
156.        endfunction
157.
158.        // Task to compare data from driver and monitor
159.        task run();
160.          forever begin
161.            mbxds.get(ds);
162.            mbxms.get(ms);
163.            $display("[SCO] : DRV : %0d MON : %0d", ds, ms);
164.
165.            if (ds == ms)
166.              $display("[SCO] : DATA MATCHED");
167.            else
168.              $display("[SCO] : DATA MISMATCHED");
169.
170.            $display("-----------------------------------------");
171.            ->sconext;
172.          end
173.        endtask
174.      endclass
175.
176.      ////////////////Environment Class
177.
178.      class environment;
179.
180.          generator gen;
181.          driver drv;
182.          monitor mon;
183.          scoreboard sco;
184.
185.          event nextgd; // gen -> drv
186.          event nextgs; // gen -> sco
187.
188.          mailbox #(transaction) mbxgd; // gen - drv
189.          mailbox #(bit [11:0]) mbxds; // drv - mon
190.          mailbox #(bit [11:0]) mbxms; // mon - sco
191.
192.          virtual spi_if vif;
193.
194.        // Constructor
195.        function new(virtual spi_if vif);
196.          mbxgd = new();
197.          mbxms = new();
198.          mbxds = new();
199.          gen = new(mbxgd);
200.          drv = new(mbxds, mbxgd);
201.
202.          mon = new(mbxms);
203.          sco = new(mbxds, mbxms);
204.
```

```
205.          this.vif = vif;
206.          drv.vif = this.vif;
207.          mon.vif = this.vif;
208.
209.          gen.sconext = nextgs;
210.          sco.sconext = nextgs;
211.
212.          gen.drvnext = nextgd;
213.          drv.drvnext = nextgd;
214.        endfunction
215.
216.        // Task to perform pre-test actions
217.        task pre_test();
218.          drv.reset();
219.        endtask
220.
221.        // Task to run the test
222.        task test();
223.        fork
224.          gen.run();
225.          drv.run();
226.          mon.run();
227.          sco.run();
228.        join_any
229.        endtask
230.
231.        // Task to perform post-test actions
232.        task post_test();
233.          wait(gen.done.triggered);
234.          $finish();
235.        endtask
236.
237.        // Task to start the test environment
238.        task run();
239.          pre_test();
240.          test();
241.          post_test();
242.        endtask
243.      endclass
244.
245.      ////////////////Testbench Top
246.      module tb;
247.
248.        spi_if vif();
249.        spi dut(vif.clk, vif.newd, vif.rst, vif.din, vif.sclk, vif.cs, vif.mosi);
250.
251.        initial begin
252.          vif.clk <= 0;
253.        end
254.
255.        always #10 vif.clk <= ~vif.clk;
256.
257.        environment env;
258.
259.        initial begin
260.          env = new(vif);
261.          env.gen.count = 20;
262.          env.run();
263.        end
264.
265.        initial begin
266.          $dumpfile("dump.vcd");
267.          $dumpvars;
268.        end
269.      endmodule
```

→CONSOLE OUTPUT FOR TESTING MASTER DESIGN:

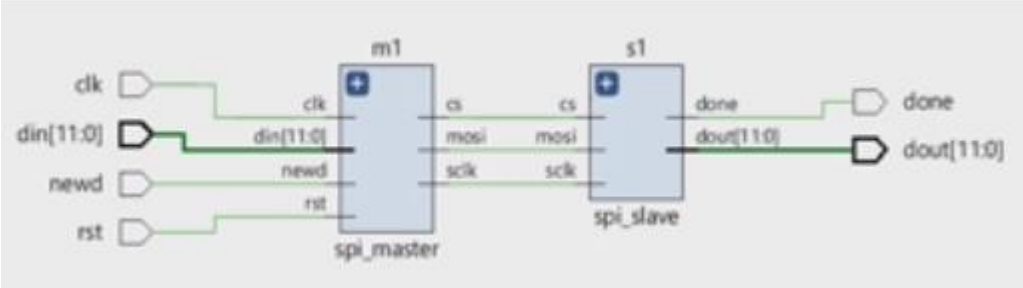
Log Share

```
# KERNEL: SLP loading done - time: 0.0 [s].
# KERNEL: Warning: You are using the Riviera-PRO EDU Edition. The performance of simulation is reduced.
# KERNEL: Warning: Contact Aldec for available upgrade options - sales@aldec.com.
# KERNEL: SLP simulation initialization done - time: 0.0 [s].
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 5580 kB (elbread=459 elab2=4957 kernel=163 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL: [DRV] : RESET DONE
# KERNEL: ----------------------------------------
# KERNEL: [GEN] : DATA_NEW : 1 DIN : 2904 CS : 0 MOSI : 0
# KERNEL: [DRV] : DATA SENT TO DAC : 2904
# KERNEL: [MON] : DATA SENT : 2904
# KERNEL: [SCO] : DRV : 2904 MON : 2904
# KERNEL: [SCO] : DATA MATCHED
# KERNEL: ----------------------------------------
# KERNEL: [GEN] : DATA_NEW : 0 DIN : 974 CS : 0 MOSI : 0
# KERNEL: [DRV] : DATA SENT TO DAC : 974
# KERNEL: [MON] : DATA SENT : 974
# KERNEL: [SCO] : DRV : 974 MON : 974
```

Log Share

```
# KERNEL: [SCO] : DATA MATCHED
# KERNEL: ----------------------------------------
# KERNEL: [GEN] : DATA_NEW : 1 DIN : 3283 CS : 0 MOSI : 0
# KERNEL: [DRV] : DATA SENT TO DAC : 3283
# KERNEL: [MON] : DATA SENT : 3283
# KERNEL: [SCO] : DRV : 3283 MON : 3283
# KERNEL: [SCO] : DATA MATCHED
# KERNEL: ----------------------------------------
# KERNEL: [GEN] : DATA_NEW : 0 DIN : 3894 CS : 0 MOSI : 0
# KERNEL: [DRV] : DATA SENT TO DAC : 3894
# KERNEL: [MON] : DATA SENT : 3894
# KERNEL: [SCO] : DRV : 3894 MON : 3894
# KERNEL: [SCO] : DATA MATCHED
# KERNEL: ----------------------------------------
# KERNEL: [GEN] : DATA_NEW : 0 DIN : 914 CS : 0 MOSI : 0
# KERNEL: [DRV] : DATA SENT TO DAC : 914
# KERNEL: [MON] : DATA SENT : 914
# KERNEL: [SCO] : DRV : 914 MON : 914
# KERNEL: [SCO] : DATA MATCHED
```

→Schematic Diagram For DUT (SPI {Master + Slave}) (i.e whole SPI Design):



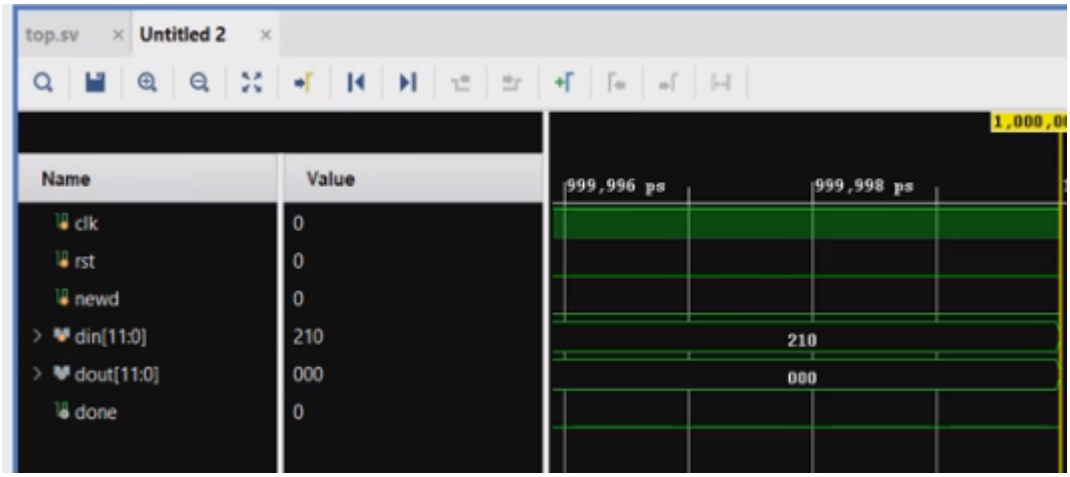
1. →Code for DUT (Master + slave SPI Design):

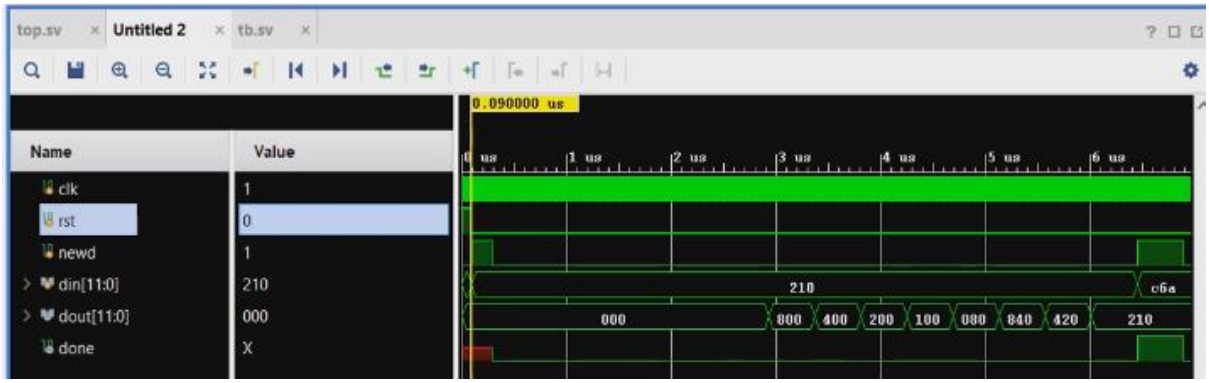
```
module spi_master(
input clk, newd,rst,
input [11:0] din,
output reg sclk,cs,mosi
    );

  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;
```

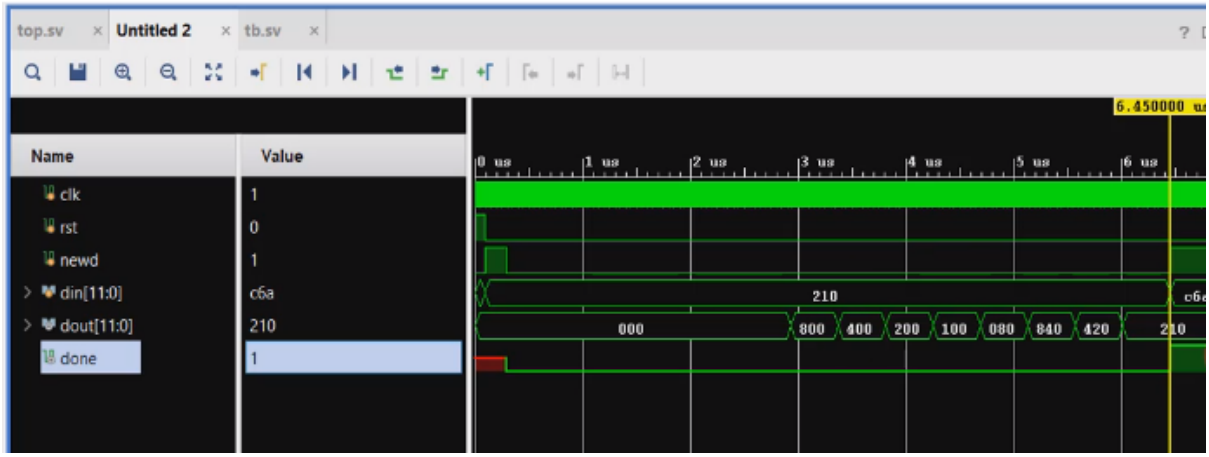
```
                  cs <= 1'b0;
                end
                else begin
                  state <= idle;
                  temp <= 8'h00;
                end
              end


        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
```

```
114.         count <= 0;
115.         done <= 1'b1;
116.         state <= detect_start;
117.         end
118.
119.       end
120.
121.       endcase
122.       end
123.       assign dout = temp;
124.
125.       endmodule
126.
127.
128.
129.       ////////////////////////////
130.       module top (
131.       input clk, rst, newd,
132.       input [11:0] din,
133.       output [11:0] dout,
134.       output done
135.       );
136.
137.       wire sclk, cs, mosi;
138.
139.       spi_master m1 (clk, newd, rst, din, sclk, cs, mosi);
140.       spi_slave s1  (sclk, cs, mosi, dout, done);
141.
142.
143.       endmodule
```

→Simulation Source:







→Testbench Code:

```
1.  ////////////////Transaction Class
2.  class transaction;
3.
4.    bit newd;                   // Flag for new transaction
5.    rand bit [11:0] din;        // Random 12-bit data input
6.    bit [11:0] dout;            // 12-bit data output
7.
8.    function transaction copy();
9.      copy = new();             // Create a copy of the transaction
10.     copy.newd = this.newd;    // Copy the newd flag
11.     copy.din  = this.din;     // Copy the data input
12.     copy.dout = this.dout;    // Copy the data output
13.   endfunction
14.
15. endclass
16.
17. ////////////////Generator Class
18. class generator;
19.
```

```
20.   transaction tr;             // Transaction object
21.   mailbox #(transaction) mbx; // Mailbox for transactions
22.   event done;                 // Done event
23.   int count = 0;              // Transaction count
24.   event drvnext;              // Event to synchronize with driver
25.   event sconext;              // Event to synchronize with scoreboard
26.
27.   function new(mailbox #(transaction) mbx);
28.     this.mbx = mbx;           // Initialize mailbox
29.     tr = new();               // Create a new transaction
30.   endfunction
31.
32.   task run();
33.     repeat(count) begin
34.       assert(tr.randomize) else $error("[GEN] :Randomization Failed");
35.       mbx.put(tr.copy);       // Put a copy of the transaction in the mailbox
36.       $display("[GEN] : din : %0d", tr.din);
37.       @(sconext);             // Wait for the scoreboard synchronization event
38.     end
39.     -> done;                  // Signal when done
40.   endtask
41.
42. endclass
43.
44. ////////////////Driver Class
45. class driver;
46.
47.   virtual spi_if vif;         // Virtual interface
48.   transaction tr;             // Transaction object
49.   mailbox #(transaction) mbx; // Mailbox for transactions
50.   mailbox #(bit [11:0]) mbxds; // Mailbox for data output to monitor
51.   event drvnext;              // Event to synchronize with generator
52.
53.   bit [11:0] din;             // Data input
54.
55.   function new(mailbox #(bit [11:0]) mbxds, mailbox #(transaction) mbx);
56.     this.mbx = mbx;           // Initialize mailboxes
57.     this.mbxds = mbxds;
58.   endfunction
59.
60.   task reset();
61.     vif.rst <= 1'b1;          // Set reset signal
62.     vif.newd <= 1'b0;         // Clear new data flag
63.     vif.din <= 1'b0;          // Clear data input
64.     repeat(10) @(posedge vif.clk);
65.     vif.rst <= 1'b0;          // Clear reset signal
66.     repeat(5) @(posedge vif.clk);
67.
68.     $display("[DRV] : RESET DONE");
69.     $display("----------------------------------------");
70.   endtask
71.
72.   task run();
73.     forever begin
74.       mbx.get(tr);            // Get a transaction from the mailbox
75.       vif.newd <= 1'b1;       // Set new data flag
76.       vif.din <= tr.din;      // Set data input
77.       mbxds.put(tr.din);      // Put data in the mailbox for the monitor
78.       @(posedge vif.sclk);
79.       vif.newd <= 1'b0;       // Clear new data flag
80.       @(posedge vif.done);
81.       $display("[DRV] : DATA SENT TO DAC : %0d",tr.din);
82.       @(posedge vif.sclk);
83.     end
84.
85.   endtask
```

```
86.
87. endclass
88.
89. ////////////////Monitor Class
90. class monitor;
91.   transaction tr;             // Transaction object
92.   mailbox #(bit [11:0]) mbx; // Mailbox for data output
93.
94.   virtual spi_if vif;         // Virtual interface
95.
96.   function new(mailbox #(bit [11:0]) mbx);
97.     this.mbx = mbx;           // Initialize the mailbox
98.   endfunction
99.
100.         task run();
101.           tr = new();               // Create a new transaction
102.           forever begin
103.             @(posedge vif.sclk);
104.             @(posedge vif.done);
105.             tr.dout = vif.dout;     // Record data output
106.             @(posedge vif.sclk);
107.             $display("[MON] : DATA SENT : %0d", tr.dout);
108.             mbx.put(tr.dout);       // Put data in the mailbox
109.           end
110.
111.         endtask
112.
113.       endclass
114.
115.       ////////////////Scoreboard Class
116.       class scoreboard;
117.         mailbox #(bit [11:0]) mbxds, mbxms; // Mailboxes for data from driver and
    monitor
118.         bit [11:0] ds;                      // Data from driver
119.         bit [11:0] ms;                      // Data from monitor
120.         event sconext;                      // Event to synchronize with
    environment
121.
122.         function new(mailbox #(bit [11:0]) mbxds, mailbox #(bit [11:0]) mbxms);
123.           this.mbxds = mbxds;               // Initialize mailboxes
124.           this.mbxms = mbxms;
125.         endfunction
126.
127.         task run();
128.           forever begin
129.             mbxds.get(ds);                  // Get data from driver
130.             mbxms.get(ms);                  // Get data from monitor
131.             $display("[SCO] : DRV : %0d MON : %0d", ds, ms);
132.
133.             if(ds == ms)
134.               $display("[SCO] : DATA MATCHED");
135.             else
136.               $display("[SCO] : DATA MISMATCHED");
137.
138.             $display("-------------------------------------");
139.             ->sconext;                       // Synchronize with the environment
140.           end
141.
142.         endtask
143.
144.       endclass
145.
146.       ////////////////Environment Class
147.       class environment;
148.           generator gen;                  // Generator object
149.           driver drv;                    // Driver object
```

```
150.           monitor mon;                    // Monitor object
151.           scoreboard sco;                 // Scoreboard object
152.
153.           event nextgd;                   // Event for generator to driver
    communication
154.           event nextgs;                   // Event for generator to scoreboard
    communication
155.
156.           mailbox #(transaction) mbxgd;   // Mailbox for generator to driver
    communication
157.           mailbox #(bit [11:0]) mbxds;    // Mailbox for driver to monitor
    communication
158.           mailbox #(bit [11:0]) mbxms;    // Mailbox for monitor to scoreboard
    communication
159.
160.           virtual spi_if vif;             // Virtual interface
161.
162.         function new(virtual spi_if vif);
163.
164.           mbxgd = new();                  // Initialize mailboxes
165.           mbxms = new();
166.           mbxds = new();
167.           gen = new(mbxgd);               // Initialize generator
168.           drv = new(mbxds,mbxgd);         // Initialize driver
169.           mon = new(mbxms);               // Initialize monitor
170.           sco = new(mbxds, mbxms);        // Initialize scoreboard
171.
172.           this.vif = vif;
173.           drv.vif = this.vif;
174.           mon.vif = this.vif;
175.
176.           gen.sconext = nextgs;           // Set synchronization events
177.           sco.sconext = nextgs;
178.
179.           gen.drvnext = nextgd;
180.           drv.drvnext = nextgd;
181.         endfunction
182.
183.         task pre_test();
184.           drv.reset();                    // Perform driver reset
185.         endtask
186.
187.         task test();
188.         fork
189.           gen.run();                      // Run generator
190.           drv.run();                      // Run driver
191.           mon.run();                      // Run monitor
192.           sco.run();                      // Run scoreboard
193.         join_any
194.         endtask
195.
196.         task post_test();
197.           wait(gen.done.triggered);       // Wait for generator to finish
198.           $finish();
199.         endtask
200.
201.         task run();
202.           pre_test();
203.           test();
204.           post_test();
205.         endtask
206.       endclass
207.
208.       ////////////////Testbench Top
209.       module tb;
210.         spi_if vif();                   // Virtual interface instance
```

```
211.
212.         top dut(vif.clk,vif.rst,vif.newd,vif.din,vif.dout,vif.done);
213.
214.         initial begin
215.           vif.clk <= 0;
216.         end
217.
218.         always #10 vif.clk <= ~vif.clk;
219.
220.         environment env;
221.
222.         assign vif.sclk = dut.m1.sclk;
223.
224.         initial begin
225.           env = new(vif);
226.           env.gen.count = 4;
227.           env.run();
228.         end
229.
230.         initial begin
231.           $dumpfile("dump.vcd");
232.           $dumpvars;
233.         end
234.       endmodule
```

→Console Output for whole SPI Design:

Log | Share

```
# KERNEL: ----------------------------------------
# KERNEL: [GEN] : din : 3576
# KERNEL: [DRV] : DATA SENT TO DAC : 3576
# KERNEL: [MON] : DATA SENT : 3576
# KERNEL: [SCO] : DRV : 3576 MON : 3576
# KERNEL: [SCO] : DATA MATCHED
# KERNEL: ----------------------------------------
# KERNEL: [GEN] : din : 3153
# KERNEL: [DRV] : DATA SENT TO DAC : 3153
# KERNEL: [MON] : DATA SENT : 3153
# KERNEL: [SCO] : DRV : 3153 MON : 3153
# KERNEL: [SCO] : DATA MATCHED
# KERNEL: ----------------------------------------
# RUNTIME: Info: RUNTIME_0068 testbench.sv (233): $finish called.
# KERNEL: Time: 28130 ns,  Iteration: 2,  Instance: /tb,  Process: @INITIAL#267_3@.
# KERNEL: stopped at time: 28130 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
```

Log Share

```
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL: [DRV] : RESET DONE
# KERNEL: ----------------------------------------
# KERNEL: [GEN] : din : 935
# KERNEL: [DRV] : DATA SENT TO DAC : 935
# KERNEL: [MON] : DATA SENT : 935
# KERNEL: [SCO] : DRV : 935 MON : 935
# KERNEL: [SCO] : DATA MATCHED
# KERNEL: ----------------------------------------
# KERNEL: [GEN] : din : 4060
# KERNEL: [DRV] : DATA SENT TO DAC : 4060
# KERNEL: [MON] : DATA SENT : 4060
# KERNEL: [SCO] : DRV : 4060 MON : 4060
# KERNEL: [SCO] : DATA MATCHED
# KERNEL: ----------------------------------------
```

→Resultant output Waveform: