CODE

→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
```

```
                  state <= send;
                  temp <= din; //din stored in temporary variable temp
                  cs <= 1'b0; //starting of transaction
                end
                else begin
                  state <= idle;
                  temp <= 8'h00;
                end
              end


        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 Environment 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

  bit cs;
  bit mosi;

  // Display function for debugging
  function void display (input string tag);
    $display("[%0s] : DATA_NEW : %0b DIN : %0d CS : %b MOSI : %0b ", tag, newd, din, cs, mosi);
  endfunction

  // Transaction copy function
  function transaction copy();
    copy = new();
    copy.newd = this.newd;
    copy.din = this.din;
    copy.cs = this.cs;
    copy.mosi = this.mosi;
  endfunction

endclass


////////////////Generator Class
class generator;

  transaction tr;
  mailbox #(transaction) mbx;
  event done;
  int count = 0;
  event drvnext;
  event sconext;

  // Constructor
  function new(mailbox #(transaction) mbx);
    this.mbx = mbx;
    tr = new();
  endfunction

  // Task to generate transactions
  task run();
    repeat(count) begin
      assert(tr.randomize) else $error("[GEN] :Randomization Failed");
      mbx.put(tr.copy);
      tr.display("GEN");
      @(drvnext);
      @(sconext);
    end
    -> done;
  endtask

endclass

////////////////Driver Class

class driver;

```

```
virtual spi_if vif;
transaction tr;
mailbox #(transaction) mbx;
mailbox #(bit [11:0]) mbxds;
event drvnext;

bit [11:0] din;

// Constructor
function new(mailbox #(bit [11:0]) mbxds, mailbox #(transaction) mbx);
  this.mbx = mbx;
  this.mbxds = mbxds;
endfunction

// Task to reset the driver
task reset();
   vif.rst <= 1'b1;
   vif.cs <= 1'b1;
   vif.newd <= 1'b0;
   vif.din <= 1'b0;
   vif.mosi <= 1'b0;
  repeat(10) @(posedge vif.clk);
    vif.rst <= 1'b0;
  repeat(5) @(posedge vif.clk);

  $display("[DRV] : RESET DONE");
  $display("----------------------------------------");
endtask

// Task to drive transactions
task run();
  forever begin
    mbx.get(tr);
    @(posedge vif.sclk);
    vif.newd <= 1'b1;
    vif.din <= tr.din;
    mbxds.put(tr.din);
    @(posedge vif.sclk);
          vif.newd <= 1'b0;
          wait(vif.cs == 1'b1);
          $display("[DRV] : DATA SENT TO DAC : %0d",tr.din);
          ->drvnext;
        end
      endtask

    endclass

    ////////////////Monitor Class

    class monitor;
      transaction tr;
      mailbox #(bit [11:0]) mbx;
      bit [11:0] srx; // Received data

      virtual spi_if vif;

      // Constructor
      function new(mailbox #(bit [11:0]) mbx);
        this.mbx = mbx;
      endfunction

      // Task to monitor the bus
      task run();
        forever begin
          @(posedge vif.sclk);
          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
```

```
        env = new(vif);
        env.gen.count = 20;
        env.run();
      end

      initial begin
        $dumpfile("dump.vcd");
        $dumpvars;
      end
    endmodule
```

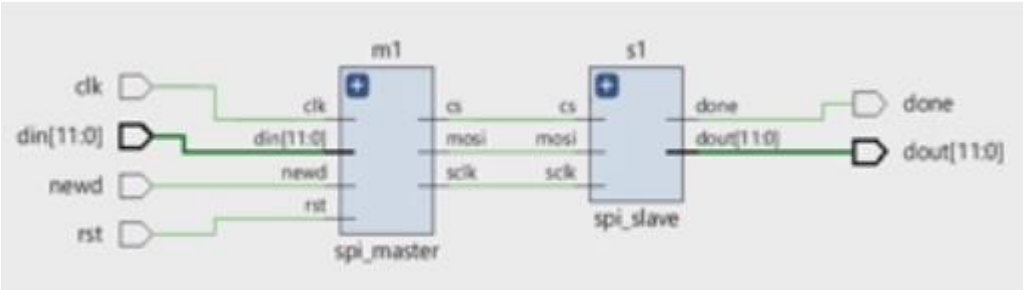
→CONSOLE OUTPUT FOR TESTING MASTER DESIGN:

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):

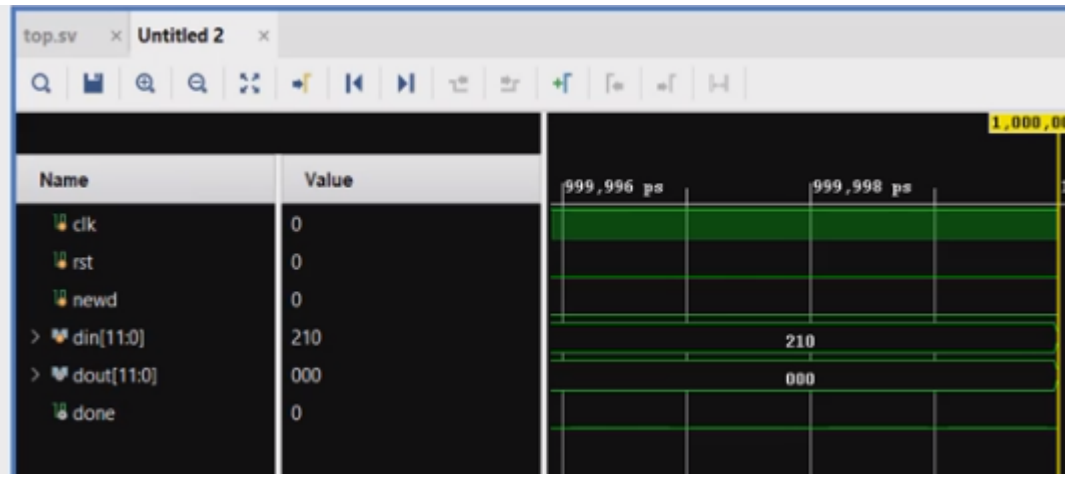
```
module spi_master(
input clk, newd,rst,
input [11:0] din,
output reg sclk,cs,mosi
    );

  typedef enum bit [1:0] {idle = 2'b00, enable = 2'b01, send = 2'b10, comp = 2'b11
} state_type;
  state_type state = idle;
```

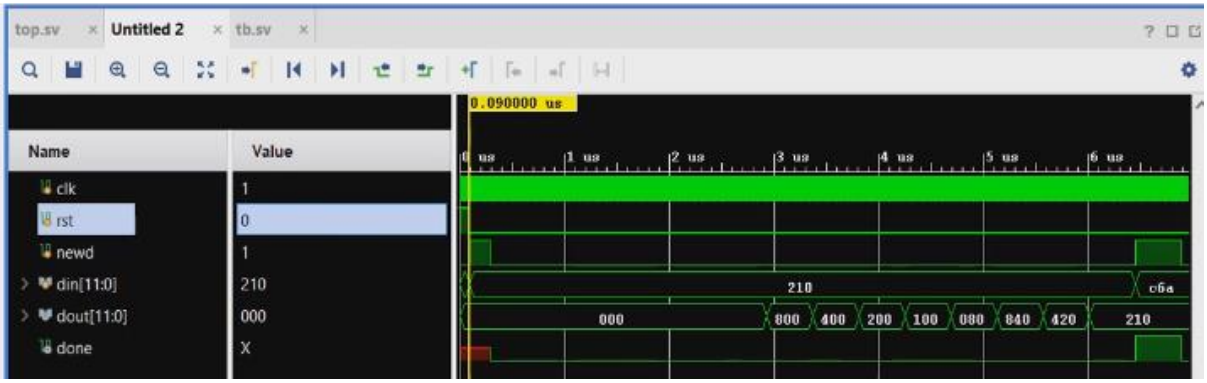
```
9.
10.   int countc = 0;
11.   int count = 0;
12.
13.   ////////////////////////generation of sclk
14.  always@(posedge clk)
15.   begin
16.     if(rst == 1'b1) begin
17.       countc <= 0;
18.       sclk <= 1'b0;
19.     end
20.     else begin
21.       if(countc < 10 )
22.           countc <= countc + 1;
23.       else
24.           begin
25.           countc <= 0;
26.           sclk <= ~sclk;
27.           end
28.     end
29.   end
30.
31.   ////////////////////state machine
32.     reg [11:0] temp;
33.
34.
35.   always@(posedge sclk)
36.   begin
37.     if(rst == 1'b1) begin
38.       cs <= 1'b1;
39.       mosi <= 1'b0;
40.     end
41.     else begin
42.      case(state)
43.          idle:
44.              begin
45.                if(newd == 1'b1) begin
46.                  state <= send;
47.                  temp <= din;
48.                  cs <= 1'b0;
49.                end
50.                else begin
51.                  state <= idle;
52.                  temp <= 8'h00;
53.                end
54.              end
55.
56.
57.        send : begin
58.          if(count <= 11) begin
59.            mosi <= temp[count]; /////sending lsb first
60.            count <= count + 1;
61.          end
62.          else
63.              begin
64.                count <= 0;
65.                state <= idle;
66.                cs <= 1'b1;
67.                mosi <= 1'b0;
68.              end
69.        end
70.
71.
72.       default : state <= idle;
73.
74.    endcase
```

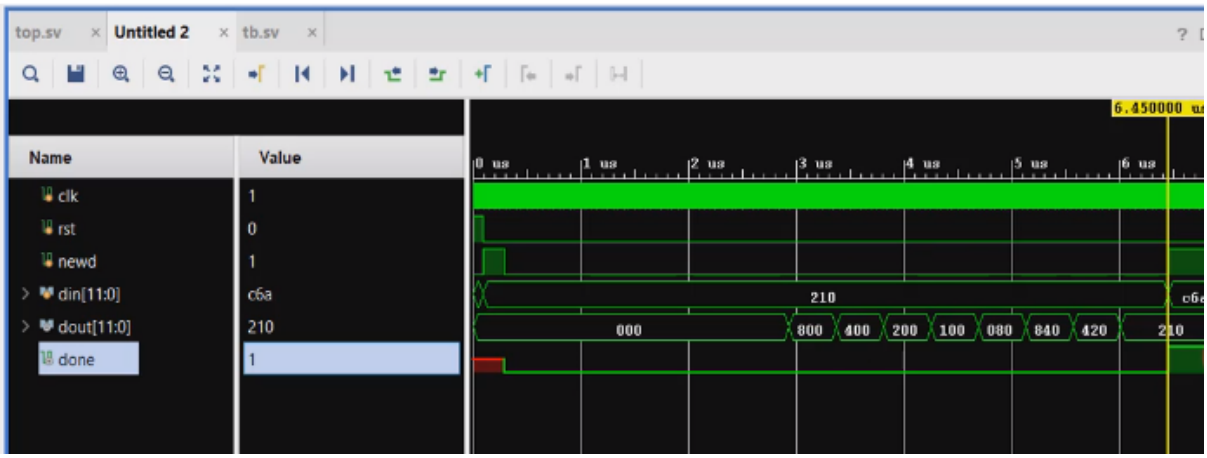
```
  end
 end

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);
```

```
141.
142.
143.        endmodule
```

→Simulation Source:







→Testbench Code:

```
////////////////Transaction Class
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
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
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
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
      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
      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
```

```
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: