```
import ovm_pkg::*;
`include "ovm_macros.svh"
`include "source/spi_interface.svh"
// uncomment following define directive to test mapped versions of master and slaves
// comment it to test source version
//`define mapped
// uncomment following directive to deliberate insert errors
`define insert_error

`timescale 1ns/10ps

module tb_spi_system#(parameter TCLK=20);

  // interconnections between master, slaves, and tb
  logic tbClkm,tbClks,Rst_n;  // master clock, slave clock, reset
  SPIbus spi();               // SPI bus between master and two slaves
  SPIctrl tb_ctrlm();         // control interfaces for SPI modules
  SPIctrl tb_ctrls[1:0]();    // array of slave control interfaces

  // variables for test vector generation and assertion checking
  integer testCountm, testCounts;
  logic [7:0] checkMXmit;       // when master strobed, save xmit data in checkMXmit
  logic [7:0] mrandToXmit;      // test data for master to transmit
  logic [7:0] srandToXmit;      // test data for slave transmit
  logic [7:0] checkSXmit[1:0];  // keep copy of slave xmit data for later checking
  logic [7:0] checkRcvds;       // keep copy of data last received by slave
  logic [1:0] slaveFull;        // slave [1:0] has data ready to transmit to master
  logic [1:0] checkSSm;         // save copy of slave select commanded to master
  logic checkSReady;            // 1 when selected slave has rcvd data ready to check
  integer randSS;               // used to tell master which slave to select
  integer randSSxmit;           // which slave tb will give value to be transmitted
  reg assertm_pass=0, assertm_fail=0; // pulsed to make it easy to find triggered assertion
  reg asserts0_pass=0, asserts0_fail=0;
  reg asserts1_pass=0, asserts1_fail=0;

  // Top level modules

`ifdef mapped  // mapped versions do not work with SV interfaces
  master MASTER(tb_ctrlm.toXmit,
    tb_ctrlm.strobe,
    tb_ctrlm.ss,
    tb_ctrlm.Rcvd,
    tb_ctrlm.Ready,
    tb_ctrlm.XmitFull,
    tb_ctrlm.busy,
    spi.miso,
    spi.mosi,
    spi.sck,
    spi.ss,
    tbClkm, Rst_n);
  slave_ID0 SLAVE0 (
    spi.mosi,
    spi.sck,
    spi.ss,
    spi.miso,
    tb_ctrls[0].toXmit,
    tb_ctrls[0].strobe,
    tb_ctrls[0].Rcvd,
    tb_ctrls[0].Ready,
    tb_ctrls[0].XmitFull,
    tb_ctrls[0].busy,
    tbClks, Rst_n);
  slave_ID1 SLAVE1 (
    spi.mosi,
    spi.sck,
    spi.ss,
    spi.miso,
    tb_ctrls[1].toXmit,
    tb_ctrls[1].strobe,
    tb_ctrls[1].Rcvd,
    tb_ctrls[1].Ready,
    tb_ctrls[1].XmitFull,
    tb_ctrls[1].busy,
    tbClks, Rst_n);
  `else  // use source version of devices
  master MASTER(.Ctrl(tb_ctrlm.Master), .Spim(spi.Master),
    .Clk_i(tbClkm), .Rst_ni(Rst_n));
  slave #(.ID(0)) SLAVE0 (.Ctrl(tb_ctrls[0].Slave), .Spis(spi.Slave),
    .Clk_i(tbClks),.Rst_ni(Rst_n));
  slave #(.ID(1)) SLAVE1 (.Ctrl(tb_ctrls[1].Slave), .Spis(spi.Slave),
    .Clk_i(tbClks),.Rst_ni(Rst_n));
  `endif

  // assertion logic checks that slaves receive correct value transmitted by master

  always @(posedge tb_ctrlm.strobe) begin
    // when master is strobed, save copy of value to be transmitted
    checkMXmit = tb_ctrlm.toXmit;
    `ifdef insert_error
      if ($dist_uniform($urandom(),0,1)) checkMXmit = ~checkMXmit;
    `endif
    // and the ID of the slave to receive that value
    checkSSm = tb_ctrlm.ss;
    end
  // whenever received value at slave changes, save a copy for checking
  always @* begin
    if (checkSSm[0]) checkRcvds = tb_ctrls[0].Rcvd;
    else if (checkSSm[1]) checkRcvds = tb_ctrls[1].Rcvd;
    end

  // when selected slave says it has newly received data ready to be read, compare
  // transmitted and received values
  assign checkSReady =
    (checkSSm[0] & tb_ctrls[0].Ready) | (checkSSm[1] & tb_ctrls[1].Ready);
  always @(posedge checkSReady) begin
    @(posedge tbClks);
    assertm_pass = (checkRcvds == checkMXmit);
    assertm_fail = ~(checkRcvds == checkMXmit);
    assert (checkRcvds == checkMXmit)
        $display("%t correct value %b received by slave",
          $time,checkRcvds);
      else
        $display("%t incorect value %b received by slave, expected %b",
          $time,checkRcvds,checkMXmit);
    #1
    assertm_pass = '0;
    assertm_fail = '0;
    end


  // assertion logic for values received by master from either slave

  // whenever slave 0 is strobed with new data to be transmitted
```

```
// save a copy of the data for checking later
always @(posedge tb_ctrls[0].strobe) begin
  // ignore new value if slave already has data to xmit
  if (!tb_ctrls[0].XmitFull & !slaveFull[0]) begin
    // save copy of data for later comparison to what master receives
    checkSXmit[0] = tb_ctrls[0].toXmit;
    `ifdef insert_error
      if ($dist_uniform($urandom(),0,1)) checkSXmit[0] = ~checkSXmit[0];
    `endif
    // wait because slave might be on verge of a data transfer
    // but doesn't know it yet
    @(posedge tb_ctrls[0].XmitFull);
    @(posedge tbClks);
    #(20*TCLK);
    // if slave is busy receiving from master, don't set full flag until done
    if (tb_ctrls[0].busy) begin
      @(negedge tb_ctrls[0].busy);
      end
    //mark slave is being ready to transmit a new value if selected
    slaveFull[0] = 1;
    end
  end

// whenever slave 1 is strobed with new data to be transmitted
// save a copy of the data for checking later
always @(posedge tb_ctrls[1].strobe) begin
  if (!tb_ctrls[1].XmitFull & !slaveFull[1]) begin
    checkSXmit[1] = tb_ctrls[1].toXmit;
    `ifdef insert_error
      if ($dist_uniform($urandom(),0,1)) checkSXmit[1] = ~checkSXmit[1];
    `endif
    @(posedge tb_ctrls[1].XmitFull);
    @(posedge tbClkm);
    #(20*TCLK);
    if (tb_ctrls[1].busy) begin
      @(negedge tb_ctrls[1].busy);
      end
    slaveFull[1] = 1;
    end
  end

// when master has reached end of a byte transfer on SPI bus
// compare received data to expected value, if anything was expected
always @(posedge tb_ctrlm.Ready) begin
  @(posedge tbClkm);
  // if slave 0 was selected and had data to transmit
  if (tb_ctrlm.ss[0] && slaveFull[0]) begin
    asserts0_pass = (tb_ctrlm.Rcvd == checkSXmit[0]);
    asserts0_fail = ~(tb_ctrlm.Rcvd == checkSXmit[0]);
    // compare received and transmitted data
    assert (tb_ctrlm.Rcvd == checkSXmit[0])
      $display("%t correct value %b received by master from slave 0",
        $time,checkSXmit[0]);
      else
      $display("%t, incorrect value %b received by master, expected %b",
        $time,tb_ctrlm.Rcvd,checkSXmit[0]);
    // mark slave as no longer having data to transmit
    slaveFull[0] = 0;
    #1
    asserts0_pass = '0;
    asserts0_fail = '0;
    end
  // if slave 1 was selected and had data to transmit
  else if (tb_ctrlm.ss[1] && slaveFull[1]) begin
    asserts1_pass = (tb_ctrlm.Rcvd == checkSXmit[1]);
    asserts1_fail = ~(tb_ctrlm.Rcvd == checkSXmit[1]);
    assert (tb_ctrlm.Rcvd == checkSXmit[1])
      $display("%t correct value %b received by master from slave 1",
        $time,checkSXmit[1]);
      else
      $display("%t, incorrect value %b received by master, expected %b",
        $time,tb_ctrlm.Rcvd,checkSXmit[1]);
    slaveFull[1] = 0;
    #1
    asserts1_pass = '0;
    asserts1_fail = '0;
    end
  end

// stimulus generation

initial begin // master Clock generation
  forever
    begin
    tbClkm = 1'b0;
    #(TCLK/2) tbClkm = 1'b1;
    #(TCLK/2);
    end
  end

initial begin // slave Clock generation
  // deliberately made asynchronouse relative to master clock
  // to make sure master & slave can work asynchrounously
  forever
    begin
    tbClks = 1'b0;
    #(TCLK/2.1) tbClks = 1'b1;
    #(TCLK/2.1);
    end
  end

// give SPI master something to transmit
initial
  begin

  // set fixed initial seed for repeatability
  mrandToXmit = $urandom(3);
  master_init();

  for (testCountm = 0; testCountm < 100; testCountm++) begin
    @(posedge tbClkm)
    mrandToXmit = $urandom();
    randSS = $dist_uniform($urandom(),0,1);
    // tell master to transmit new data to randomly selected slave
    master_xmit(mrandToXmit,randSS,50);
    end
  end

// reset SPI master
task master_init;
  tb_ctrlm.ss = 2'b0;
  tb_ctrlm.strobe = 1'b0;
  tb_ctrlm.toXmit = 8'd0;
  Rst_n = 1'b0;
  #(TCLK*0.75);
  Rst_n = 1'b1;
  #(TCLK*0.75);
endtask

// send command to master via control interface
// first provide data and slave selection, then send strobe
task master_xmit (input [7:0] datin, input [1:0] slave_index, input integer delay);
  tb_ctrlm.strobe = 1'b0;
  tb_ctrlm.toXmit = datin;
  tb_ctrlm.ss = 2'b00;
  tb_ctrlm.ss[slave_index] = 2'b01;
  @(posedge tbClkm);
  #(TCLK/2);
  tb_ctrlm.strobe = 1'b1;
  @(posedge tbClkm);
  #(TCLK/2);
  tb_ctrlm.strobe = 1'b0;
  #(TCLK*delay);
endtask

// Give the slave SPIs something to transmit
// process is similar to that used to give master data to transmit
initial
  begin
  // set fixed seed for repeatability
  // but want different sequence from master SPI
  srandToXmit = $urandom(4);
  slaveFull = 2'b00;
  tb_ctrls[0].toXmit = 'b0;
  tb_ctrls[1].toXmit = 'b0;
  tb_ctrls[0].strobe = 1'b0;
  tb_ctrls[1].strobe = 1'b0;
  for (testCounts = 0; testCounts < 100; testCounts++) begin
    @(posedge tbClkm)
    srandToXmit = $urandom();
    randSSxmit = $dist_uniform($urandom(),0,1);
    slave_xmit(srandToXmit,randSSxmit,33);
    end
  end

// send command to selected slave to have it transmit data
// process is similar to master_xmit except that we have to
// choose between one of two slaves to give the command
task slave_xmit (input [7:0] datin, input integer slave_index, input integer delay);
// Clumsy but SV won't allow variable index into array of instances
// More elegant option would be to create an array of "virtual" interfaces
  if (slave_index == 0) begin
    tb_ctrls[0].toXmit = datin;
    tb_ctrls[0].strobe = 2'b0;
    @(posedge tbClkm);
    #(TCLK/2);
    tb_ctrls[0].strobe = 1'b1;
    @(posedge tbClkm);
    #(TCLK/2);
    tb_ctrls[0].strobe = 2'b0;
    #(TCLK*delay);
    end
  else begin
    tb_ctrls[1].toXmit = datin;
    tb_ctrls[1].strobe = 2'b0;
    @(posedge tbClkm);
    #(TCLK/2);
    tb_ctrls[1].strobe = 1'b1;
    @(posedge tbClkm);
    #(TCLK/2);
    tb_ctrls[1].strobe = 2'b0;
    #(TCLK*delay);
    end
endtask

endmodule
```