```
 1 import ovm_pkg::*;
 2 `include "ovm_macros.svh"
 3 `include "source/spi_interface.svh"
 4 // uncomment following define directive to test mapped versions of master and slaves
 5 // comment it to test source version
 6 //`define mapped
 7
 8 `timescale 1ns/10ps
 9
10 module tb_spi_system#(parameter TCLK=20);
11
12   // interconnections between master, slaves, and tb
13   logic tbClkm,tbClks,Rst_n;  // master clock, slave clock, reset
14   SPIbus spi();               // SPI bus between master and two slaves
15   SPIctrl tb_ctrlm();         // control interfaces for SPI modules
16   SPIctrl tb_ctrls[1:0]();    // array of slave control interfaces
17
18   // variables for test vector generation and assertion checking
19   integer testCountm, testCounts;
20   reg [7:0] checkMXmit;       // when master strobed, save xmit data in checkMXmit
21   reg [7:0] mrandToXmit;      // test data for master to transmit
22   reg [7:0] srandToXmit;      // test data for slave transmit
23   reg [7:0] checkSXmit[1:0];  // keep copy of slave xmit data for later checking
24   reg [7:0] checkRcvds;       // keep copy of data last received by slave
25   reg [1:0] slaveFull;        // slave [1:0] has data ready to transmit to master
26   reg [1:0] checkSSm;         // save copy of slave select commanded to master
27   reg checkSReady;             // 1 when selected slave has rcvd data ready to check
28   integer randSS;             // used to tell master which slave to select
29   integer randSSxmit;         // which slave tb will give value to be transmitted
30
31   // Top level modules
32
33 `ifdef mapped  // mapped versions do not work with SV interfaces
34   master MASTER(tb_ctrlm.toXmit,
35     tb_ctrlm.strobe,
36     tb_ctrlm.ss,
37     tb_ctrlm.Rcvd,
38     tb_ctrlm.Ready,
39     tb_ctrlm.XmitFull,
40     tb_ctrlm.busy,
41     spi.miso,
42     spi.mosi,
43     spi.sck,
44     spi.ss,
45     tbClkm, Rst_n);
46   slave_ID0 SLAVE0 (
47     spi.mosi,
48     spi.sck,
49     spi.ss,
50     spi.miso,
51     tb_ctrls[0].toXmit,
52     tb_ctrls[0].strobe,
53     tb_ctrls[0].Rcvd,
54     tb_ctrls[0].Ready,
55     tb_ctrls[0].XmitFull,
56     tb_ctrls[0].busy,
57     tbClks, Rst_n);
58   slave_ID1 SLAVE1 (
59     spi.mosi,
60     spi.sck,
61     spi.ss,
62     spi.miso,
```

```
63      tb_ctrls[1].toXmit,
64      tb_ctrls[1].strobe,
65      tb_ctrls[1].Rcvd,
66      tb_ctrls[1].Ready,
67      tb_ctrls[1].XmitFull,
68      tb_ctrls[1].busy,
69      tbClks, Rst_n);
70    `else  // use source version of devices
71    master MASTER(.Ctrl(tb_ctrlm.Master), .Spim(spi.Master),
72      .Clk_i(tbClkm), .Rst_ni(Rst_n));
73    slave #(.ID(0)) SLAVE0 (.Ctrl(tb_ctrls[0].Slave), .Spis(spi.Slave),
74      .Clk_i(tbClks),.Rst_ni(Rst_n));
75    slave #(.ID(1)) SLAVE1 (.Ctrl(tb_ctrls[1].Slave), .Spis(spi.Slave),
76      .Clk_i(tbClks),.Rst_ni(Rst_n));
77    `endif
78
79    // assertion logic checks that slaves receive correct value transmitted by master
80
81    always @(posedge tb_ctrlm.strobe) begin
82      // when master is strobed, save copy of value to be transmitted
83      checkMXmit = tb_ctrlm.toXmit;
84      // and the ID of the slave to receive that value
85      checkSSm = tb_ctrlm.ss;
86      end
87    // whenever received value at slave changes, save a copy for checking
88    always @* begin
89      if (checkSSm[0]) checkRcvds = tb_ctrls[0].Rcvd;
90      else if (checkSSm[1]) checkRcvds = tb_ctrls[1].Rcvd;
91      end
92
93    // when selected slave says it has newly received data ready to be read, compare
94    // transmitted and received values
95    assign checkSReady =
96      (checkSSm[0] & tb_ctrls[0].Ready) | (checkSSm[1] & tb_ctrls[1].Ready);
97    always @(posedge checkSReady) begin
98      @(posedge tbClks);
99      assert (checkRcvds == checkMXmit)
100         $display("%t correct value %b received by slave",
101           $time,checkRcvds);
102       else $display("%t incorect value %b received by slave, expected %b",
103         $time,checkRcvds,checkMXmit);
104     end
105
106
107   // assertion logic for values received by master from either slave
108
109   // whenever slave 0 is strobed with new data to be transmitted
110   // save a copy of the data for checking later
111   always @(posedge tb_ctrls[0].strobe) begin
112     // ignore new value if slave already has data to xmit
113     if (!tb_ctrls[0].XmitFull & !slaveFull[0]) begin
114       // save copy of data for later comparison to what master receives
115       checkSXmit[0] = tb_ctrls[0].toXmit;
116       // wait because slave might be on verge of a data transfer
117       // but doesn't know it yet
118       @(posedge tb_ctrls[0].XmitFull);
119       @(posedge tbClks);
120       #(20*TCLK);
121       // if slave is busy receiving from master, don't set full flag until done
122       if (tb_ctrls[0].busy) begin
123         @(negedge tb_ctrls[0].busy);
124         end
```

```
125       //mark slave is being ready to transmit a new value if selected
126       slaveFull[0] = 1;
127       end
128     end
129
130   // whenever slave 1 is strobed with new data to be transmitted
131   // save a copy of the data for checking later
132   always @(posedge tb_ctrls[1].strobe) begin
133     if (!tb_ctrls[1].XmitFull & !slaveFull[1]) begin
134       checkSXmit[1] = tb_ctrls[1].toXmit;
135       @(posedge tb_ctrls[1].XmitFull);
136       @(posedge tbClkm);
137       #(20*TCLK);
138       if (tb_ctrls[1].busy) begin
139         @(negedge tb_ctrls[1].busy);
140         end
141       slaveFull[1] = 1;
142       end
143     end
144
145   // when master has reached end of a byte transfer on SPI bus
146   // compare received data to expected value, if anything was expected
147   always @(posedge tb_ctrlm.Ready) begin
148     @(posedge tbClkm);
149     // if slave 0 was selected and had data to transmit
150     if (tb_ctrlm.ss[0] && slaveFull[0]) begin
151       // compare received and transmitted data
152       assert (tb_ctrlm.Rcvd == checkSXmit[0])
153         $display("%t correct value %b received by master from slave 0",
154           $time,checkSXmit[0]);
155         else
156         $display("%t, incorrect value %b received by master, expected %b",
157           $time,tb_ctrlm.Rcvd,checkSXmit[0]);
158       // mark slave as no longer having data to transmit
159       slaveFull[0] = 0;
160       end
161     // if slave 1 was selected and had data to transmit
162     else if (tb_ctrlm.ss[1] && slaveFull[1]) begin
163       assert (tb_ctrlm.Rcvd == checkSXmit[1])
164         $display("%t correct value %b received by master from slave 1",
165           $time,checkSXmit[1]);
166         else
167         $display("%t, incorrect value %b received by master, expected %b",
168           $time,tb_ctrlm.Rcvd,checkSXmit[1]);
169       slaveFull[1] = 0;
170       end
171     end
172
173   // stimulus generation
174
175   initial begin // master Clock generation
176     forever
177       begin
178       tbClkm = 1'b0;
179       #(TCLK/2) tbClkm = 1'b1;
180       #(TCLK/2);
181       end
182     end
183
184   initial begin // slave Clock generation
185     // deliberately made asynchronouse relative to master clock
186     // to make sure master & slave can work asynchrounously
```

```
187     forever
188       begin
189       tbClks = 1'b0;
190       #(TCLK/2.1) tbClks = 1'b1;
191       #(TCLK/2.1);
192       end
193     end
194
195   // give SPI master something to transmit
196   initial
197     begin
198     // set fixed initial seed for repeatability
199     mrandToXmit = $urandom(3);
200     master_init();
201
202     for (testCountm = 0; testCountm < 100; testCountm++) begin
203       @(posedge tbClkm)
204       mrandToXmit = $urandom();
205       randSS = $dist_uniform($urandom(),0,1);
206       // tell master to transmit new data to randomly selected slave
207       master_xmit(mrandToXmit,randSS,50);
208       end
209     end
210
211   // reset SPI master
212   task master_init;
213     tb_ctrlm.ss = 2'b0;
214     tb_ctrlm.strobe = 1'b0;
215     tb_ctrlm.toXmit = 8'd0;
216     Rst_n = 1'b0;
217     #(TCLK*0.75);
218     Rst_n = 1'b1;
219     #(TCLK*0.75);
220   endtask
221
222   // send command to master via control interface
223   // first provide data and slave selection, then send strobe
224   task master_xmit (input [7:0] datin, input [1:0] slave_index, input integer delay);
225     tb_ctrlm.strobe = 1'b0;
226     tb_ctrlm.toXmit = datin;
227     tb_ctrlm.ss = 2'b00;
228     tb_ctrlm.ss[slave_index] = 2'b01;
229     @(posedge tbClkm);
230     #(TCLK/2);
231     tb_ctrlm.strobe = 1'b1;
232     @(posedge tbClkm);
233     #(TCLK/2);
234     tb_ctrlm.strobe = 1'b0;
235     #(TCLK*delay);
236   endtask
237
238   // Give the slave SPIs something to transmit
239   // process is similar to that used to give master data to transmit
240   initial
241     begin
242     // set fixed seed for repeatability
243     // but want different sequence from master SPI
244     srandToXmit = $urandom(4);
245     slaveFull = 2'b00;
246     tb_ctrls[0].toXmit = 'b0;
247     tb_ctrls[1].toXmit = 'b0;
248     tb_ctrls[0].strobe = 1'b0;
```

```
249     tb_ctrls[1].strobe = 1'b0;
250     for (testCounts = 0; testCounts < 100; testCounts++) begin
251       @(posedge tbClkm)
252       srandToXmit = $urandom();
253       randSSxmit = $dist_uniform($urandom(),0,1);
254       slave_xmit(srandToXmit,randSSxmit,33);
255       end
256     end
257
258   // send command to selected slave to have it transmit data
259   // process is similar to master_xmit except that we have to
260   // choose between one of two slaves to give the command
261   task slave_xmit (input [7:0] datin, input integer slave_index, input integer delay);
262   // Clumsy but SV won't allow variable index into array of instances
263   // More elegant option would be to create an array of "virtual" interfaces
264     if (slave_index == 0) begin
265       tb_ctrls[0].toXmit = datin;
266       tb_ctrls[0].strobe = 2'b0;
267       @(posedge tbClkm);
268       #(TCLK/2);
269       tb_ctrls[0].strobe = 1'b1;
270       @(posedge tbClkm);
271       #(TCLK/2);
272       tb_ctrls[0].strobe = 2'b0;
273       #(TCLK*delay);
274       end
275     else begin
276       tb_ctrls[1].toXmit = datin;
277       tb_ctrls[1].strobe = 2'b0;
278       @(posedge tbClkm);
279       #(TCLK/2);
280       tb_ctrls[1].strobe = 1'b1;
281       @(posedge tbClkm);
282       #(TCLK/2);
283       tb_ctrls[1].strobe = 2'b0;
284       #(TCLK*delay);
285       end
286   endtask
287
288 endmodule
```