FH-OÖ Hagenberg/ESD
Advanced Methods of Verification, SS 2015
Rainer Findenig © 2008

2. Übung: The Next Generation: SystemVerilog

1 BFM mit SystemVerilog

Implementieren Sie Ihr *Bus Functional Model* und die dazugehörende Testbench aus der letzten Übung in SystemVerilog. Kapseln Sie das BFM in einer Klasse, die zur Kommunikation mit dem DUV ein *Interface* mit einem *Clocking Block* verwendet. Die Testbench wird in einem eigenen Programm implementiert, die diese Klasse instantiiert. Zusätzlich benötigen Sie ein minimales Testbed, das das DUV, das Testprogramm und das *Interface* instantiiert.

1.1 Hinweise

Interfaces vereinfachen die Verwendung von Bussen in einem Hardwareentwurf [SDF06]. Sie fassen, ähnlich wie *Records* in VHDL, Signale zusammen. Darüber hinaus unterstützen sie jedoch unter anderem sogenannte *Modports* , durch die das *Interface* an die von dem Modul geforderte Sicht angepasst werden kann:

```
1  interface wishboneBus # (
2          parameter int gDataWidth = 32,
3          parameter int gAddrWidth = 8
4      )(
5          input bit clk
6      );
7
8      logic [gAddrWidth-1:0] adr;
9      logic [gDataWidth-1:0] datM;     // data coming from master
10     logic [gDataWidth-1:0] datS;     // data coming from slave
11     logic [gDataWidth/8-1:0] sel;
12     logic cyc, stb, we, ack;
13
14     modport master (     // the interface as seen from the master
15         input ack, datS,
16         output stb, cyc, we, datM, adr, sel
17     );
18     modport slave (      // the interface as seen from the slave
19         output ack, datS,
20         input stb, cyc, we, datM, adr, sel
21     );
22 endinterface
23
24 // ...
25
26 module top ();
```

```
27      logic clk, rst;
28
29      // instantiate the interface
30      wishboneBus bus(clk);
31
32      // clk generator
33      always #10 clk = ~clk;
34
35      // instantiate master and slave and connect them to the interface
36      wbMaster m(bus.master, rst);
37      wbSlave s(
38          // ...
39      );
40  endmodule
```

Zusätzlich können *Interfaces* auch *Clocking Blocks* enthalten, die zur Synchronisation der Daten zwischen Testbench und Design verwendet werden. Erweitern Sie das *Interface* aus dem obigen Codebeispiel um einen *Clocking Block* für den Master.

Anmerkung: *Interfaces* können auch Funktionen enthalten. Dies würde sich anbieten, um ein einfaches und effizientes BFM erstellen zu können. Sie unterstützen allerdings keine objektorientierten Konzepte wie zum Beispiel Vererbung oder Polymorphie, sind also Klassen unterlegen. Daher ist es sinnvoll, ein BFM als Klasse zu implementieren, die ein *Interface* als *Member* -Variable besitzt. Beachten Sie dabei, dass diese *Member* -Variable als **`virtual`** deklariert sein muss!

2 Theorie

□ Beantworten Sie folgende Fragen:

- Erklären Sie den Unterschied zwischen den SystemVerilog-Datentypen **`bit`** und **`logic`**! Wo liegen die jeweiligen Vorteile?
- Erklären Sie den Unterschied zwischen *packed* und *unpacked* Arrays! Geben Sie für beide Varianten sinnvolle Einsatzbeispiele an!

"Beware of bugs in the above code; I have only proved it correct, not tried it."

Donald Knuth

Literatur

[SDF06] Stuart Sutherland, Simon Davidmann, and Peter Flake. *SystemVerilog for Design*. Springer Science+Business Media, 2006.

1 Beantwortung der Fragen

- `bit` ist zweiwertig und `logic` ist vierwertig. `bit` braucht weniger Speicher und `logic` ist realer.
- Packed vs Unpacked
  - Packed Arrays sind wie Vektoren in VHDL und haben eine festgelegte Darstellung als Bit-Strom. Beispiel: `bit [7:0] inputByte`
  - Unpacked Arrays sind wie Arrays in C, das heißt das Werkzeug kann die Abbildung frei wählen. Beispiel: `bit inputByte [7:0]`

2 Testfälle

Es wurden die unten aufgelisteten Funktionen der BFM getestet. Dazu wurde der gesamte Speicherbereich des RAM zweimal geschrieben und gelesen. Beim ersten Mal waren auf der Datenleitung alle geraden Bit 1 und ungeraden Bit 0. Beim zweiten Mal wurde der Wert invertiert.
Zusätzlich gibt es noch einen Test der zur Kontrolle fehlschlagen soll und zwei Klassen die Zufallswerte zum Testen erzeugen. Die Zufallswerte werden mittels randc erzeugt.
Getestet und simuliert wurde auf dem Applicationserver.

- SingleRead
- SingleWrite
- BlockRead
- BlockWrite

3 Source Code

../../src/wishbone_bfm.sv

```
1  class wishbone_bfm #(parameter int gDataWidth = 32, parameter int
       gAddrWidth = 8, parameter debugMode = 0);
2          virtual wishbone_if.master sigs;
3
4          function new(virtual wishbone_if.master _sigs);
5                  sigs = _sigs;
6
7                  sigs.cb.adr <= 0;
8                  sigs.cb.datM <= 0;
9                  sigs.cb.we <= 0;
10                 sigs.cb.sel <= '0;
11                 sigs.cb.cyc <= 0;
12                 sigs.cb.stb <= 0;
13         endfunction
14
15         //
               ------------------------------------------------------------------------
16
17         virtual task singleRead(input logic [gAddrWidth-1:0] addr, output
               logic [gDataWidth-1:0] data);
```

```
18            if(debugMode)
19                    $display("singleRead @%0tns", $time);
20
21                  sigs.cb.adr <= addr;
22                  sigs.cb.we <= 0;
23                  sigs.cb.sel <= '1;
24                  sigs.cb.cyc <= 1;
25                  sigs.cb.stb <= 1;
26
27                  @(posedge sigs.cb.ack)
28                  data = sigs.cb.datS;
29                  sigs.cb.stb <= 0;
30                  sigs.cb.cyc <= 0;
31          endtask
32
33          //
               ------------------------------------------------------------------------
34
35          virtual task singleWrite(input logic [gAddrWidth-1:0] addr, logic [
               gDataWidth-1:0] data);
36            if(debugMode)
37                    $display("singleWrite @%0tns", $time);
38
39                  sigs.cb.adr <= addr;
40                  sigs.cb.datM <= data;
41                  sigs.cb.we <= 1;
42                  sigs.cb.sel <= '1;
43                  sigs.cb.cyc <= 1;
44                  sigs.cb.stb <= 1;
45
46                  @(posedge sigs.cb.ack)
47                  sigs.cb.stb <= 0;
48                  sigs.cb.cyc <= 0;
49          endtask
50
51          //
               ------------------------------------------------------------------------
52
53          virtual task blockRead(input logic [gAddrWidth-1:0] addr, ref logic
                [gDataWidth-1:0] data[]);
54            if(debugMode)
55                    $display("blockRead @%0tns", $time);
56
57                  for (int i = 0; i < $size(data); i = i + 1) begin
58                          sigs.cb.adr <= addr + i;
59                          sigs.cb.we <= 0;
60                          sigs.cb.sel <= '1;
61                          sigs.cb.cyc <= 1;
62                          sigs.cb.stb <= 1;
63
64                          @(posedge sigs.cb.ack)
65                          data[i] = sigs.cb.datS;
66                  end
67                  sigs.cb.stb <= 0;
68                  sigs.cb.cyc <= 0;
69          endtask
70
```

```
71          //
               ------------------------------------------------------------------------


72
73          virtual task blockWrite(input logic [gAddrWidth-1:0] addr, const
               ref logic [gDataWidth-1:0] data[]);
74            if(debugMode)
75                    $display("blockWrite @%0tns", $time);
76
77                  for (int i = 0; i < $size(data); i = i + 1) begin
78                          sigs.cb.adr <= addr + i;
79                          sigs.cb.datM <= data[i];
80                          sigs.cb.we <= 1;
81                          sigs.cb.sel <= '1;
82                          sigs.cb.cyc <= 1;
83                          sigs.cb.stb <= 1;
84                          @(posedge sigs.cb.ack);
85                  end
86                  sigs.cb.stb <= 0;
87                  sigs.cb.cyc <= 0;
88          endtask
89
90          //
               ------------------------------------------------------------------------


91
92          virtual task idle();
93                  $display("idle @%0tns", $time);
94
95                  @sigs.cb;
96          endtask
97  endclass
98
99  module top;
100         logic clk = 0, rst;
101         wishbone_if wb(clk);
102
103         // clk generator
104         always #10 clk = ~clk;
105
106         // RAM instantiation
107         RAM TheRam(wb.clk, rst, wb.adr, wb.datM, wb.sel, wb.cyc, wb.stb, wb
               .we, wb.datS, wb.ack);
108         // test program instantiation
109         test TheTest(wb.master, rst);
110 endmodule
111
112 class rnd_values #(parameter int gDataWidth = 32, parameter int gAddrWidth
       = 8);
113   randc logic [gAddrWidth-1:0] addr;
114   randc logic [gDataWidth-1:0] data;
115 endclass
116
117 class rnd_values_block #(parameter int gDataWidth = 32, parameter int
       gAddrWidth = 8);
118   randc logic [gAddrWidth-1:0] addr;
119   randc logic [gDataWidth-1:0] dataBlock [];
120   function new (integer gBlockSize);
121     dataBlock = new[gBlockSize];
122   endfunction
```

```
123 endclass
124
125 program test #(parameter int gDataWidth = 32, parameter int gAddrWidth = 8)
        (wishbone_if.master wb, output logic rst);
126         initial begin : stimuli
127                 wishbone_bfm#(gDataWidth, gAddrWidth) bfm = new(wb);
128                 rnd_values#(gDataWidth, gAddrWidth) rndValues = new();
129                 const integer cBlockSize = 10;
130                 rnd_values_block#(gDataWidth, gAddrWidth) rndValuesBlock =
                        new(cBlockSize);
131
132                 const integer gEndAddress = (2**gAddrWidth)-1;
133                 const logic [gDataWidth-1:0] testInput1 = 32'hAAAAAAAA;
134                 const logic [gDataWidth-1:0] testInput2 = 32'h55555555;
135                 const integer cNumRandTests = 2000;
136
137                 logic [gAddrWidth-1:0] addr = '1;
138                 logic [gDataWidth-1:0] data = '0;
139
140                 logic [gDataWidth-1:0] dataBlock[] = new [gEndAddress];
141                 logic [gDataWidth-1:0] readDataBlock[] = new [gEndAddress];
142                 logic [gDataWidth-1:0] rndDataBlock[] = new [cBlockSize];
143
144                 // generate reset
                        ----------------------------------------------------
145                 rst = 0;
146                 #10 rst = 1;
147                 #20 rst = 0;
148
149                 // stimuli
                        ------------------------------------------------------------


150                 //test Single Read Write with data 10101...
151                 for (integer i=0; i<gEndAddress; i++) begin
152                         addr = i;
153                         bfm.singleWrite(addr, testInput1);
154                         bfm.singleRead(addr, data);
155
156                         assert (data == testInput1)
157                                 else $error("SingleTest: TestInput1 is
                                        wrong" );
158                 end
159
160
161                 //test Single Read Write with data 010101...
162                 for (integer i=0; i<gEndAddress; i++) begin
163                         addr = i;
164                         bfm.singleWrite(addr, testInput2);
165                         bfm.singleRead(addr, data);
166
167                         assert (data == testInput2)
168                                 else $error("SingleTest: TestInput2 is
                                        wrong" );
169                 end
170
171
172                 //test Block Read Write with data 1010101...
173                 for (integer i=0; i<gEndAddress; i++) begin
174                         dataBlock[i]    = testInput1;
175                 end;
```

```
176
177                 bfm.blockWrite(0, dataBlock);
178                 bfm.blockRead(0, readDataBlock);
179
180                 for (integer i=0; i<gEndAddress; i++) begin
181                         assert (readDataBlock[i] == testInput1)
182                                 else $error("BlockTest: TestInput1 is wrong
                                        ");
183                 end;
184
185
186                 //test Block Read Write with data 01010101...
187                 for (integer i=0; i<gEndAddress; i++) begin
188                         dataBlock[i]    = testInput2;
189                 end;
190
191                 bfm.blockWrite(0, dataBlock);
192                 bfm.blockRead(0, readDataBlock);
193
194                 for (integer i=0; i<gEndAddress; i++) begin
195                         assert (readDataBlock[i] == testInput2)
196                                 else $error("BlockTest: TestInput2 is wrong
                                        ");
197                 end;
198
199                 //control test
200                 assert (readDataBlock[0] == testInput1)
201                         else $error("ControlTest: This test is supposed to
                                    be wrong");
202
203
204                 //random test singleWrite and singleRead
205                 for (integer i=0; i<gEndAddress; i++) begin
206                   if(rndValues.randomize())
207                     begin
208                       bfm.singleWrite(rndValues.addr, rndValues.data);
209                       bfm.singleRead(rndValues.addr, data);
210                       assert (data == rndValues.data)
211                         else $error("RandomTestSingle: Value is wrong" );
212                     end
213                   else
214                     $display("Error in randomize");
215                 end
216
217                 //random test blockWrite and blockRead
218                 for (integer i=0; i<cNumRandTests; i++) begin
219                   if(rndValuesBlock.randomize())
220                     begin
221                       bfm.blockWrite(rndValuesBlock.addr, rndValuesBlock.
                                dataBlock);
222                       bfm.blockRead(rndValuesBlock.addr, rndDataBlock);
223                       assert (rndDataBlock == rndValuesBlock.dataBlock)
224                         else $error("RandomTestBlock: Value is wrong" );
225                     end
226                   else
227                     $display("Error in randomize().");
228                 end
229
230                 $stop;
231
```

```
232         end : stimuli
233 endprogram
234
235 interface wishbone_if # (
236     parameter int gDataWidth = 32,
237     parameter int gAddrWidth = 8
238   )(
239     input bit clk
240   );
241
242   logic [gAddrWidth-1:0] adr;
243   logic [gDataWidth-1:0] datM; // data coming from master
244   logic [gDataWidth-1:0] datS; // data coming from slave
245   logic [gDataWidth/8-1:0] sel;
246   logic cyc, stb, we, ack;
247
248   clocking cb @(posedge clk);
249     input ack, datS;
250     output stb, cyc, we, datM, adr, sel;
251   endclocking
252
253   modport master ( // the interface as seen from the master
254     clocking cb
255   );
256   modport slave ( // the interface as seen from the slave
257     output ack, datS,
258     input stb, cyc, we, datM, adr, sel
259   );
260 endinterface
```

../../sim/Compile.do

```
1 vlib work
2 vlog ../src/ram.sv
3 vlog ../src/wishbone_bfm.sv
```

../../sim/Sim.do

```
1 vsim -novopt top
2 log -r *
3 run -all
```