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 Idle Test der überprüft ob alle Signale auf 'Z' gesetzt sind und einen Test der zur Kontrolle fehlschlagen soll.
Getestet wurde lokal am Rechner mit Questasim, da der Applicationserver bei beiden Gruppenmitgliedern nicht funktionierte.

- SingleRead
- SingleWrite
- BlockRead
- BlockWrite
- Idle

3 Source Code

../../src/wishbone_bfm.sv

```
1  class wishbone_bfm #(parameter int gDataWidth = 32, parameter int
       gAddrWidth = 8);
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          $display("singleRead @%0tns", $time);
19          sigs.cb.adr <= addr;
20          sigs.cb.we <= 0;
21          sigs.cb.sel <= '1;
22          sigs.cb.cyc <= 1;
23          sigs.cb.stb <= 1;
24
25          @(posedge sigs.cb.ack)
26          data = sigs.cb.datS;
27          sigs.cb.stb = 0;
28          sigs.cb.cyc = 0;
29      endtask
30
31      //
            ------------------------------------------------------------------------
32
33      virtual task singleWrite(input logic [gAddrWidth-1:0] addr, logic [
            gDataWidth-1:0] data);
34          $display("singleWrite @%0tns", $time);
35          sigs.cb.adr <= addr;
36          sigs.cb.datM <= data;
37          sigs.cb.we <= 1;
38          sigs.cb.sel <= '1;
39          sigs.cb.cyc <= 1;
40          sigs.cb.stb <= 1;
41
42          @(posedge sigs.cb.ack)
43          sigs.cb.stb = 0;
44          sigs.cb.cyc = 0;
45      endtask
46
47      //
            ------------------------------------------------------------------------
48
49      virtual task blockRead(input logic [gAddrWidth-1:0] addr, ref logic [
            gDataWidth-1:0] data[]);
50          $display("blockRead @%0tns", $time);
51          for (int i = 0; i < $size(data); i = i + 1) begin
52              sigs.cb.adr <= addr;
53              sigs.cb.we <= 0;
54              sigs.cb.sel <= '1;
55              sigs.cb.cyc <= 1;
56              sigs.cb.stb <= 1;
57
58              @(posedge sigs.cb.ack)
59              data = sigs.cb.datS;
60          end
61          sigs.cb.stb = 0;
62          sigs.cb.cyc = 0;
63      endtask
64
65      //
            ------------------------------------------------------------------------
66
```

```
67     virtual task blockWrite(input logic [gAddrWidth-1:0] addr, const ref
           logic [gDataWidth-1:0] data[]);
68        $display("blockWrite @%0tns", $time);
69        for (int i = 0; i < $size(data); i = i + 1) begin
70           sigs.cb.adr <= addr;
71           sigs.cb.datM <= data;
72           sigs.cb.we <= 1;
73           sigs.cb.sel <= '1;
74           sigs.cb.cyc <= 1;
75           sigs.cb.stb <= 1;
76           @(posedge sigs.cb.ack);
77        end
78        sigs.cb.stb = 0;
79        sigs.cb.cyc = 0;
80     endtask
81
82     //
           ------------------------------------------------------------------------


83
84     virtual task idle();
85        $display("idle @%0tns", $time);
86
87        @sigs.cb;
88     endtask
89  endclass
90
91  module top;
92     logic clk = 0, rst;
93     wishbone_if wb(clk);
94
95     // clk generator
96     always #10 clk = ~clk;
97
98     // RAM instantiation
99     RAM TheRam(wb.clk, rst, wb.adr, wb.datM, wb.sel, wb.cyc, wb.stb, wb.we,
           wb.datS, wb.ack);
100    // test program instantiation
101    test TheTest(wb.master, rst);
102 endmodule
103
104 program test #(parameter int gDataWidth = 32, parameter int gAddrWidth = 8)
       (wishbone_if.master wb, output logic rst);
105    initial begin : stimuli
106       wishbone_bfm#(gDataWidth, gAddrWidth) bfm = new(wb);
107
108
109       logic [gAddrWidth-1:0] addr = '1;
110       logic [gDataWidth-1:0] data = '0;
111
112       // generate reset
              ----------------------------------------------------
113       rst = 0;
114       #10 rst = 1;
115       #20 rst = 0;
116
117       // stimuli
              ------------------------------------------------------------
118       singleWrite(addr, '1);
119       singleRead(addr, data);
```

```
120         $display("data: %b", data);
121
122       end : stimuli
123   endprogram
124
125   interface wishbone_if # (
126       parameter int gDataWidth = 32,
127       parameter int gAddrWidth = 8
128     )(
129       input bit clk
130     );
131
132     logic [gAddrWidth-1:0] adr;
133     logic [gDataWidth-1:0] datM; // data coming from master
134     logic [gDataWidth-1:0] datS; // data coming from slave
135     logic [gDataWidth/8-1:0] sel;
136     logic cyc, stb, we, ack;
137
138     clocking cb @(posedge clk);
139       input ack, datS;
140       output stb, cyc, we, datM, adr, sel;
141     endclocking
142
143     modport master ( // the interface as seen from the master
144       clocking cb
145     );
146     modport slave ( // the interface as seen from the slave
147       output ack, datS,
148       input stb, cyc, we, datM, adr, sel
149     );
150   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  #add wave -position end  sim:/tbwishbonebfm/busIn
3  #add wave -position end  sim:/tbwishbonebfm/busOut
4  #add wave -position end  sim:/tbwishbonebfm/testInput1
5  #add wave -position end  sim:/tbwishbonebfm/testInput2
6  log -r *
7  run -all
```