

1 BFM mit SystemVerilog

Implementieren Sie Ihr *Bus Functional Model* und die dazugehörige 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.