

Übung zur Vorlesung Architekturen und Entwurf von Rechnersystemen

Prof. Dr.-Ing. A. Koch
Jaco Hofmann, MSc.



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Wintersemester 17/18
Übungsblatt 3

Der sprachliche Unterbau von Bluespec ist Haskell. Dementsprechend lassen sich viele Techniken, die aus Haskell bekannt sind, auch auf Bluespec anwenden. Eine dieser Techniken, QuickCheck, wird in dieser Übung vorgestellt. [QuickCheck](#) erlaubt es Funktionen automatisiert zu testen. Dabei kann man Eigenschaften der zu testenden Funktion abstrakt beschreiben und QuickCheck kümmert sich um das Erstellen der Testfälle.

Eine Bluespec Variante von QuickCheck heißt [BlueCheck](#). Clonen Sie das Repository von Github und kopieren Sie das Bluespec Paket `BlueCheck.bsv` in Ihr Codeverzeichnis. Einige Beispiele finden Sie in den Paketen `SimpleExample` und `StackExample`.

In dieser Übung lernen Sie die Verwendung von `BlueCheck` zum Testen von Bluespec Modulen kennen.

Aufgabe 3.1 Testen gegen Bedingungen

Bluespec erlaubt es Tests von Bedingungen durchzuführen. Dies kann zum Beispiel zum Überprüfen von arithmetischen Operationen verwendet werden (Beispiel von `BlueCheck SimpleExample`):

```
1  module [BlueCheck] mkArithSpec ();
2      function Bool addComm(Int#(4) x, Int#(4) y) =
3          x + y == y + x;
4
5      function Bool addAssoc(Int#(4) x, Int#(4) y, Int#(4) z) =
6          x + (y + z) == (x + y) + z;
7
8      function Bool subComm(Int#(4) x, Int#(4) y) =
9          x - y == y - x;
10
11     prop("addComm" , addComm);
12     prop("addAssoc" , addAssoc);
13     prop("subComm" , subComm);
14 endmodule
15
16 module [Module] mkArithChecker ();
17     blueCheck(mkArithSpec);
18 endmodule
```

Bauen Sie das Modul `mkArithChecker` und lassen Sie es im Simulator laufen. `BlueCheck` wird bei der Ausführung automatisch Parameter für die Funktionen generieren und versuchen, die Aussagen zu widerlegen.

Erweitern Sie die Testfälle um sinnvolle Tests für Multiplikation und Division.

Aufgabe 3.2 Die FIFO

Für die nächste Teilaufgabe entwickeln Sie eine FIFO auf der Basis eines [Ringspeichers](#). Die FIFO soll 16 Elemente aufnehmen können. Benutzen Sie als Basis des Stacks einen `Vector#(16, Reg#(Int#(16)))`. Verwenden Sie folgendes Interface, denken Sie an passende Guards, um das Überschreiben von Werten zu verhindern.

```
1  interface FIFO
2      method Action put(Int#(16) e); // Put Element on FIFO
3      method ActionValue#(Int#(16)) get(); // Get Element from FIFO
4  endinterface
```

Übung zur Vorlesung Architekturen und Entwurf von Rechnersystemen

Aufgabe 3.3 FIFO testen

Als weitere Möglichkeit zum Testen von Bluespec Modulen erlaubt es Bluespec, eine Referenzimplementierung (auch z.B. in C++ geschrieben) mit der Hardwareimplementierung zu vergleichen. Testen Sie die FIFO aus [Aufgabe 3.2](#) mit Hilfe von BlueCheck gegen die von Bluespec bereitgestellte FIFO:

```
1  FIFO#(Int#(16)) goldenFIFO <- mkSizedFIFO(16);
```

Sie können dafür BlueCheck anweisen, die Methoden der Module zu vergleichen. Für einen Stack könnte so ein Modul folgendermaßen aussehen:

```
1  module [BlueCheck] checkStack ();
2      /* Specification instance */
3      Stack#(8, Bit#(4)) spec <- mkStackSpec();
4
5      /* Implementation instance */
6      Stack#(8, Bit#(4)) imp <- mkBRAMStack();
7
8      equiv("pop"      , spec.pop      , imp.pop);
9      equiv("push"     , spec.push     , imp.push);
10     equiv("isEmpty"  , spec.isEmpty  , imp.isEmpty);
11     equiv("top"      , spec.top      , imp.top);
12 endmodule
```

Erstellen Sie eine randomisierte Testbench für Ihre FIFO.