Übung zur Vorlesung Architekturen und Entwurf von Rechnersystemen

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

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 17/18
Übungsblatt 1

Diese Übung bietet eine allgemeine Einführung in grundlegende Bluespec Sprachkonzepte.

Aufgabe 1.1 Hello Bluespec

Die erste Aufgabe dieser Übung führt in das Kompilieren von Bluespecmodulen sowie deren Simulation ein. Es wird vorausgesetzt, dass Bluespec vollständig eingerichtet ist und die benötigten Programme (`bsc` etc.) ausgeführt werden können. Dies sollte in den ISP-Rechnerpools der Fall sein. Bei Fragen zur Einrichtung stehen zum Beispiel das d120 Forum oder die der Bluespec Distribution beiliegenden Literatur zur Verfügung.

Zum Kennenlernen der Bluespec-Toolchain kann folgender Bluespec Code verwendet werden:

```
1 package HelloBluespec;
2   module mkHelloBluespec(Empty);
3     rule helloDisplay;
4       $display("(%0d) Hello World!", $time);
5     endrule
6   endmodule
7 endpackage
```

Der Dateiname muss mit dem Namen des Packages übereinstimmen und hat die Endung „.bsv“, in diesem Fall „HelloBluespec.bsv“. Zum Kompilieren und Simulieren mit Hilfe des Bluespec eigenen Simulators wird `bsc` verwendet. Als erster Schritt werden die Bluespec Module kompiliert. Dies geschieht mit dem Befehl

```
bsc -u -sim -g mkHelloBluespec HelloBluespec.bsv
```

Die eigentliche Simulation wird mit dem Befehl

```
bsc -sim -o out -e mkHelloBluespec
```

erstellt. Mit Hilfe des `-o` Parameters kann der Name der Ausgabedatei verändert werden. Wurden beide Befehle erfolgreich ausgeführt, kann die Simulation mit `./out` gestartet werden.

Wie zu erwarten ist, wird in jedem Taktzyklus `Hello World!` ausgegeben. Die Simulation kann mit der Tastenkombination `Strg-C` beendet werden.

Aufgabe 1.2 Blinky

Während in Programmiersprachen häufig das „Hello World“ Programm die ersten Gehversuche darstellt, findet man in hardwarenahen Umgebungen häufig LED-Blink-Programme. Dabei wird eine LED in einer festgelegten Frequenz an- und ausgeschaltet. Diese Frequenz ist typischerweise wesentlich langsamer als die Betriebsfrequenz der benutzten Hardware. Für dieses Beispiel wird angenommen, ihr Modul wird an ein 100 MHz Taktsignal angeschlossen. Um den Schaltvorgang der LED als Mensch sehen zu können, muss eine Möglichkeit gefunden werden, um aus dem schnellen Takt einen langsameren zu machen.

Aufgabe 1.2.1 Zähler

Eine einfache Möglichkeit das Taktsignal zu verlangsamen ist die Nutzung eines Zählers. Der Zähler wird mit jedem Takt um eins erhöht. Erreicht der Zähler einen Schwellenwert, wird das gewünschte Ereignis ausgeführt und der Zähler zurückgesetzt.

Ergänzen Sie das Beispiel aus Aufgabe 1.1 um einen solchen Zähler, der die Botschaft „Hello World“ alle 2^{25} Taktzyklen ausgibt. Verwenden Sie dafür ein Register des Typs `Reg#(UInt#(25))` als Zähler.

Übung zur Vorlesung Architekturen und Entwurf von Rechnersystemen

Aufgabe 1.2.2 LED

Erweitern Sie das Modul `mkHelloBluespec` aus Aufgabe 1.1 um einen Ausgang für die LED-Ansteuerung. Binden Sie dafür das Interface `HelloBluespec` ein.

```
1 interface HelloBluespec;
2   (* always_enabled, always_ready *) method Bool led();
3 endinterface
```

Die Attribute `always_enabled` und `always_ready` spezifizieren dabei, dass die folgende Methode keine Signale zum Handshake benötigt. Dies ist hier der Fall, da die LED sich ständig in einem der beiden Zustände befindet.

Erweitern Sie das Modul um ein Register für den LED Zustand vom Typ `Bool`. Dieses Register soll alle 2^{25} Taktzyklen negiert werden. Erweitern Sie das Modul um eine Implementation der Methode `led`, die den aktuellen Zustand der LED zurückgibt.

Aufgabe 1.2.3 Testbench

Aktuell wird die Simulation bis zur Unterbrechung mit `Strg-C` ausgeführt. Um die Ausführung der Simulation zu steuern und das zu testende Modul beeinflussen zu können, kann eine Testbench verwendet werden. In Bluespec wird dafür ein weiteres Modul erstellt, das eine Instanz des zu testenden Moduls enthält.

Erstellen Sie ein Modul `mkHelloTestbench` mit `Empty` als Interface. Binden Sie das Modul `mkHelloBluespec` ein. Die Testbench soll für 2 Sekunden laufen (100 MHz Takt s.o.). Die Simulation kann mit `$finish()` beendet werden.

Passen Sie die Befehle zur Kompilierung entsprechend auf das neue Topmodul `mkHelloTestbench` an (`bsc --help` listet die möglichen Parameter und deren Bedeutung auf).

Aufgabe 1.2.4 LED Ausgabe

Erweitern Sie die Testbench um eine Ausgabefunktion, die „`LED an.`“ oder „`LED aus.`“ ausgibt, wann auch immer die LED im Modul `mkHelloBluespec` an-/ausgeschaltet wird. Benutzen Sie dafür ein Register, das den letzten Zustand der LED speichert und vergleichen Sie diesen mit dem aktuellen Zustand der LED.

Aufgabe 1.2.5 Analysieren

Bluespec bietet die Möglichkeit, die Simulation auf Waveform Ebene nachzuvollziehen. Der Parameter `-V filename.vcd` veranlasst die Simulation (in unserem Fall `./out`) die Datei `filename.vcd` zu erzeugen, die alle zur Simulation gehörenden Waveforms beinhaltet. Diese Datei kann mit einem geeigneten Anzeigeprogramm wie GTKWave geöffnet werden. Standardmäßig werden bei der Kompilierung des BSV-Codes viele interne Signale wegoptimiert. Häufig ist es zur Fehlersuche hilfreich, wenn diese Signale erhalten bleiben. Mit Hilfe des `bsc` Parameters `-keep-fires` kann man die Wegoptimierung der Signale verhindern.

Vergleichen Sie die Waveforms der Simulation mit und ohne `-keep-fires`.

Achtung: Die Simulation mit BlueSim ist bei Verwendung des Parameters `-V` um ein vielfaches langsamer.

Aufgabe 1.3 Bluespec ALU

Eine ALU (Arithmetic Logic Unit) ist ein Rechenwerk, das häufig in Prozessoren zum Einsatz kommt. In dieser Aufgabe wird eine einfache ALU mit den Funktionen

- Multiplizieren
- Dividieren
- Addieren
- Subtrahieren
- Logisches Und
- Logisches Oder

implementiert.

Übung zur Vorlesung Architekturen und Entwurf von Rechnersystemen

Aufgabe 1.3.1 Das Interface

Das Bluespec Modul soll das folgende Interface besitzen:

```
interface HelloALU;
    method Action setupCalculation(AluOps op, Int#(32) a, Int#(32) b);
    method ActionValue#(Int#(32))  getResult();
endinterface
```

`AluOps` ist dabei der folgende Typ:

```
typedef enum{Mul,Div,Add,Sub,And,Or} AluOps deriving (Eq, Bits);
```

Die Anmerkung `deriving` kann in Bluespec genutzt werden, um Typklassen zu verwenden. In diesem Fall wird AluOps den Typklassen `Eq` und `Bits` zugewiesen. Der Bluespec Compiler wird automatisch dafür sorgen, dass sich der Typ AluOps als Bits darstellen lässt und verglichen werden kann. Weitere Informationen zu Typklassen finden Sie in der nächsten Übung oder in der Bluespec Referenz.

Aufgabe 1.3.2 Implementierung

Implementieren Sie auf Basis des oben vorgestellten Interfaces ein Modul `mkSimpleALU`. Die Methode `getResult` soll dabei so lange blockieren, bis das Ergebnis der Berechnung vorliegt. Erstellen Sie des Weiteren eine Testbench, die automatisch das aktuelle Ergebnis ausgibt, wenn es sich ändert, und alle Funktionen testet.

Aufgabe 1.3.3 Power

Erweitern Sie das Modul mkHelloALU um eine Möglichkeit, a^b zu berechnen. Erstellen Sie dafür ein geeignetes Modul und Interface zur sequentiellen Berechnung. Binden Sie dieses in ihrer ALU ein. Eine mögliche C Implementation sieht dabei folgendermaßen aus:

```
int pow(int a, int b) {
  int tmp = 1;
  for(int i = 0; i < b; ++i) {
    tmp *= a;
  }
  return tmp;
}
```