Vorlesung Architekturen und Entwurf von Rechnersystemen

Prof. Andreas Koch, Yannick Lavan, Johannes Wirth, Mihaela Damian

TECHNISCHE UNIVERSITÄT DARMSTADT

Bluespec-Wrap-Up Übungsblatt zur Klausurvorbereitung
Wintersemester 2022/2023

Dieses Wrap-Up Übungsblatt dient als optionales Unterstützungsangebot für die Klausurvorbereitung. Das Thema ist der Bluespec-Teil der Vorlesung. Dabei besteht kein Anspruch auf Vollständigkeit. Bei Fragen oder Problemen bei der Bearbeitung wenden Sie sich bitte an die Tutoren. Dafür können Sie entweder das Moodle-Forum oder die angebotenen Sprechstunden verwenden.

Viel Erfolg bei der Klausurvorbereitung!

1 Verständnis- und Wissensfragen

1. Woraus setzt sich ein Bluespec-Modul zusammen? Was ist die Funktion der einzelnen Bestandteile?
2. Wie unterscheiden sich Interface und Modul voneinander?
3. Wozu wird der Präfix `mk` verwendet?
4. Wie ist die Bereitschaft einer Regel (`CAN_FIRE`) definiert?
5. Was ist mit dem Ausdruck „logische Ausführungsreihenfolge“ gemeint?
6. Was sind die zentralen Unterschiede zwischen Registern und Wires?
7. Was ist eine mögliche Anwendung für den Datentyp `Maybe`?
8. Wie ist Bluespec System Verilog aufgebaut? Gibt es die Konzepte von Klassen und Typen?
9. Wie greifen Module auf (Sub-)Module zu?
10. Was ist der Unterschied zwischen Methoden und Funktionen?

11. Was muss in einem Interface deklariert werden?

12. Was sind Regeln?

13. Kann ein Modul kein Interface haben?

14. Welche Vorteile und Nachteile sind mit der strikten Trennung Modul-Interface verbunden?

15. Für was sind “int” und “bit” Abkürzungen?

1.1 Weiterführende Elemente von Bluespec System Verilog

16. Wie erhält man von einem “numeric” type den Wert als Bits?

17. Was ist der Unterschied zwischen Parallelität und Nebenläufigkeit?

18. Welche Ursachen gibt es für Konflikte?

19. Was sind die Vorteile von Pipeline-/Bypass FIFOs?

20. Wie geht der Compiler vor, wenn Konflikte auftreten?

21. Was versteht man unter nested interfaces?

22. Wobei können “tagged unions” hilfreich sein?

23. Wie werden Daten in Tupel gefasst bzw. aus Tupeln extrahiert?

24. Wozu dienen extend und truncate?

25. Wozu dienen pack und unpack?

26. Lässt sich extend mit pack und unpack ersetzen?

27. Wie lassen sich Nachrichten zwischen Regeln innerhalb eines Taktes austauschen? Welche Vorgehensweise ist zu bevorzugen?

28. Was bewirken die Attribute decending_urgency, execution_order und preempts? Wie unterscheiden sich diese voneinander?

29. Was bewirken die Attribute mutually_exclusive und synthesize?

30. Für welche Elemente einer Methode generiert der Compiler Ports?

31. Wie kann das beeinflusst werden, für was Ports generiert werden?

32. Was beeinflusst in Bluespec System Verilog den kritischen Pfad?

2 Fehlersuche

In dieser Aufgabe bekommen Sie Bluespec-Code gegeben, der syntaktische und/oder semantische Fehler enthält. Finden Sie diese Fehler, wenn möglich ohne weitere Hilfsmittel. Wie muss der Code geändert werden, um die Fehler zu beheben und warum?

2.1 Counter

```
module mkSomeCounter (Empty);
    Reg#(UInt#(16)) cnt <= mkReg(0);

    rule increment;
        cnt = cnt + 1;
        $display("%d\n", cnt);
    endrule
endmodule
```

2.2 Modulstruktur

```
package HelloBluespec;
  module mkHelloBluespec(Empty);
    Reg#(UInt#(32)) flag <- mkReg(24);

    method ActionValue#(Int#(8)) foo (int x);
      ...
    endmethod
  endmodule
endpackage
```

2.3 Scheduling

```
module mkExample(Empty);
    Reg#(int) x <- mkReg(5);

    (* execution_order = "write, read" *)
    rule read;
        $display("%d\n", x);
    endrule

    rule write;
        x <= x * -1;
    endrule
endmodule
```

3 Was ist die Ausgabe?

In den folgenden Teilaufgaben bekommen Sie korrekte Bluespec-Beispiele gegeben. Wie lautet die Ausgabe, wenn man das Modul (bzw. die jeweilige Testbench) simuliert? Versuchen Sie zuerst die Aufgabe unter Klausurbedingungen zu lösen und verzichten Sie also auch hier soweit wie möglich auf Ausprobieren, um die Funktionsweise des Moduls nachzuvollziehen.

3.1 Guards

```
module mkGuardExample(Empty);
    Reg#(UInt#(32)) counter <- mkReg(0);
    Reg#(Int#(32)) myValue <- mkReg(1);
    Reg#(Bool) flag <- mkReg(False);

    rule rule1(flag);
        myValue <= myValue + 5;
    endrule

    rule rule2(!flag);
        myValue <= myValue / 2;
    endrule

    rule switch;
        flag <= !flag;
    endrule

    rule count;
        counter <= counter + 1;
        if (counter == 8) begin
            $display("%d\n", myValue);
            $finish;
        end
    endrule
endmodule
```

3.2 Wires und Register

Zusatzfrage: In welcher Reihenfolge werden die Regeln dieses Moduls ausgeführt und wieso?

```
module mkWireExample(Empty);
    Wire#(int) myWire <- mkWire;
    Reg#(int) myReg <- mkReg(5);

    rule writeWire;
        myWire <= 5;
    endrule

    rule writeReg;
        myReg <= 6;
    endrule

    rule read;
        $display("%d\n", myWire + myReg);
        $finish;
    endrule
endmodule
```

4 Projekt

Implementieren Sie einen Stack (Kellerspeicher) als Bluespec-Modul. Nutzen Sie dabei das untenstehende Interface. Die Methoden entsprechen den gewöhnlichen Operationen[1] auf Stacks. Ihr Modul soll Platz für genau 5 Elemente bieten. Testen Sie Ihre Implementierung in einer Testbench.

```
interface Stack;
    method Action push(Int#(32) value);
    method ActionValue#(Int#(32)) pop();
    method Int#(32) peek();
endinterface
```

[1] https://en.wikipedia.org/wiki/Stack_(abstract_data_type)