Übung zur Vorlesung Einführung in Computer Microsystems

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

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 15
Übungsblatt 3 - Lösungsvorschlag

Aufgabe 3.1 Scheduling

Zu Konflikten kommt es in Bluespec unter anderem wenn mehrere Rules versuchen das selbe Register zu schreiben. In dieser Aufgabe wird eine Möglichkeit gezeigt diese Konflikte gezielt zu lösen.

Aufgabe 3.1.1 Das Design

Zur Demonstration von Scheduling-Attributen wird folgendes Interface verwendet.

```
interface Conflict;
  method Action putA(UInt#(32) val);
  method Action putB(UInt#(32) val);
  method UInt#(32) getVal();
endinterface
```

Intern sollen beide Actions in das selbe Register schreiben.
Eine mögliche Implementation mit der Hilfe von `FIFO` sieht folgendermaßen aus.

```
 import FIFO::*;

 module mkConflict(Conflict);
  FIFO#(UInt#(32)) valA <- mkFIFO();
  FIFO#(UInt#(32)) valB <- mkFIFO();

  Reg#(UInt#(32)) superVal <- mkReg(0);

  rule writeA;
    let v = valA.first(); valA.deq();
    superVal <= v;
  endrule

  rule writeB;
    let v = valB.first(); valB.deq();
    superVal <= v;
  endrule

  method Action putA(UInt#(32) val);
    valA.enq(val);
  endmethod

  method Action putB(UInt#(32) val);
    valB.enq(val);
  endmethod

  method UInt#(32) getVal();
```

```
    return superVal;
  endmethod
endmodule

module mkConflictTestbench(Empty);
  Conflict uut           <- mkConflict();
  Reg#(UInt#(7)) counter <- mkReg(0);

  rule spamConflictA (pack(counter)[0] == 1);
    uut.putA(42);
  endrule

  rule spamConflictB;
    uut.putB(125);
  endrule

  rule cntrIncr;
    counter <= counter + 1;
  endrule

  rule endSim (counter == 127);
    $finish();
  endrule
endmodule
```

Beim Versuch oben stehenden Code zu kompilieren wird eine Warnung ausgegeben. Welche Auswirkungen hat diese Warnung auf das erstellte Design? Untersuchen Sie die Waveforms der Simulation mit `-keep-fires`.

Aufgabe 3.1.2 Steuerung

Die Priorisierung der Rules kann in Bluespec unter anderem mit Hilfe von Attributen gesteuert werden. Setzen Sie das Attribut `(* preempts = "writeA, writeB" *)` vor eine der Rules des Moduls `mkConflict`. Analysieren Sie die nun entstandenen Waveforms. Tauschen Sie `writeA` und `writeB` im obigem Attribut und schauen Sie sich erneut die Waveforms an.

Aufgabe 3.1.3 Weitere Attribute

Bluespec bietet viele weitere Attribute zur feinen Steuerung der Ausführungsreihenfolge von Rules. Welche Auswirkungen hat es wenn Sie statt `preempts` die Attribute `descending_urgency` oder `excecution_order` verwenden?

Aufgabe 3.2 BRAM

Implementieren Sie auf Basis des Packages BRAM einen RAM mit vier Eingängen. Das zu verwendende Modul sieht dabei folgendermaßen aus:

```
module mkBRAM1Server #( BRAM_Configure cfg )
  ( BRAM1Port #(addr, data) )
  provisos(Bits#(addr, addr_sz),
  Bits#(data, data_sz),
  DefaultValue#(data) );
```

Das Interface ist folgendermaßen definiert:

```
typedef Server#(BRAMRequest#(addr, data), data) BRAMServer#(type addr, type data);
typedef struct {Bool write;
    Bool responseOnWrite;
    addr address;
    data datain;
```

```
  } BRAMRequest#(type addr, type data) deriving(Bits, Eq);
interface BRAM1Port#(type addr, type data);
  interface BRAMServer#(addr, data) portA;
  method Action portAClear;
endinterface: BRAM1Port
```

Eine Instanziierung könnte wie folgt aussehen:

```
BRAM_Configure cfg = defaultValue; //declare variable cfg
cfg.memorySize = 1024*32; //new value for memorySize
BRAM1Port#(UInt#(15), Bit#(16)) bram <- mkBRAM1Server (cfg);
```

Da das Interface `BRAM1Port` nur einen Port für den RAM zur Verfügung stellt müssen Sie den vorhandenen Port zwischen den angeschlossenen Geräten teilen. Erlauben Sie jedem der Eingänge nacheinander für einen Takt den BRAM zu benutzen.

Das Interface des Moduls soll dabei folgendermaßen aussehen:

```
interface RRRam;
  BRAMServer#(UInt#(15), Bit#(16)) portA;
  BRAMServer#(UInt#(15), Bit#(16)) portB;
  BRAMServer#(UInt#(15), Bit#(16)) portC;
  BRAMServer#(UInt#(15), Bit#(16)) portD;
endinterface
```

Erstellen Sie eine Testbench die die vier Eingänge des BRAM mit Testdaten versieht. PortA soll dabei jeden Takt geschrieben werden, PortB jeden zweiten Takt, PortC jeden Takt und PortD jeden achten Takt. Beenden Sie die Simulation nachdem jeder Port zehn mal geschrieben wurde.

```
package RRRam;

  import BRAM   :: *;
  import Vector :: *;
  import FIFO   :: *;

  typedef BRAMRequest#(UInt#(15), Bit#(16)) RRRamRequest;

  interface RRRam;
    interface BRAMServer#(UInt#(15), Bit#(16)) portA;
    interface BRAMServer#(UInt#(15), Bit#(16)) portB;
    interface BRAMServer#(UInt#(15), Bit#(16)) portC;
    interface BRAMServer#(UInt#(15), Bit#(16)) portD;
  endinterface

  module mkRRRam(RRRam);
    Vector#(4, FIFO#(RRRamRequest))      inFifos <- replicateM(mkFIFO());
    Vector#(4, FIFO#(Bit#(16)))         outFifos <- replicateM(mkFIFO());
    FIFO#(UInt#(2))                     readFifo <- mkSizedFIFO(16);
    Reg#(UInt#(2))                   currentPort <- mkReg(0);

    BRAM_Configure cfg = defaultValue; //declare variable cfg
    cfg.memorySize = 1024*32; //new value for memorySize
    BRAM1Port#(UInt#(15), Bit#(16)) bram <- mkBRAM1Server (cfg);

    for(Integer i = 0; i < 4; i = i + 1) begin
      rule handlePort(currentPort == fromInteger(i));
        bram.portA.request.put(inFifos[i].first());
        inFifos[i].deq();
```

```
30          if(!inFifos[i].first().write || inFifos[i].first().responseOnWrite) readFifo.enq(fromInteger(i));
31        endrule
32        rule handleReadPort(readFifo.first() == fromInteger(i));
33          let retVal <- bram.portA.response.get();
34          outFifos[i].enq(retVal);
35          readFifo.deq();
36        endrule
37      end
38
39      // Bookkeeping
40      rule roundRobin;
41        currentPort <= currentPort + 1;
42      endrule
43
44      // Interfaces
45      interface Server portA;
46        interface request  = toPut(inFifos[0]);
47        interface response = toGet(outFifos[0]);
48      endinterface
49
50      interface Server portB;
51        interface request  = toPut(inFifos[1]);
52        interface response = toGet(outFifos[1]);
53      endinterface
54
55      interface Server portC;
56        interface request  = toPut(inFifos[2]);
57        interface response = toGet(outFifos[2]);
58      endinterface
59
60      interface Server portD;
61        interface request  = toPut(inFifos[3]);
62        interface response = toGet(outFifos[3]);
63      endinterface
64    endmodule
65
66    module mkRRRamTb(Empty);
67      RRRam bram <- mkRRRam();
68
69      Vector#(4, Reg#(Bool))      portsDone   <- replicateM(mkReg(False));
70      Vector#(4, Reg#(UInt#(12))) portCounter <- replicateM(mkReg(0));
71      Reg#(UInt#(32))             counter     <- mkReg(0);
72
73      rule putPortA;
74        if(!portsDone[0]) begin
75          RRRamRequest req;
76          req.write = True;
77          req.responseOnWrite = False;
78          req.address = 0;
79          req.datain = extend(pack(portCounter[0]));
80          bram.portA.request.put(req);
81          portCounter[0] <= portCounter[0] + 1;
82          if(portCounter[0] == 9) begin
83            portsDone[0] <= True;
84          end
```

```
85        end
86      endrule
87
88      rule putPortB ((counter & 32'b1) == 0);
89        if(!portsDone[1]) begin
90          RRRamRequest req;
91          req.write = True;
92          req.responseOnWrite = False;
93          req.address = 4;
94          req.datain = extend(pack(portCounter[1]));
95          bram.portB.request.put(req);
96          portCounter[1] <= portCounter[1] + 1;
97          if(portCounter[1] == 9) begin
98            portsDone[1] <= True;
99          end
100       end
101     endrule
102
103     rule putPortC;
104       if(!portsDone[2]) begin
105         RRRamRequest req;
106         req.write = True;
107         req.responseOnWrite = False;
108         req.address = 12;
109         req.datain = extend(pack(portCounter[2]));
110         bram.portC.request.put(req);
111         portCounter[2] <= portCounter[2] + 1;
112         if(portCounter[2] == 9) begin
113           portsDone[2] <= True;
114         end
115       end
116     endrule
117
118     rule putPortD ((counter & 32'b111) == 0);
119       if(!portsDone[3]) begin
120         RRRamRequest req;
121         req.write = False;
122         req.responseOnWrite = False;
123         req.address = 12;
124         req.datain = extend(pack(portCounter[3]));
125         bram.portD.request.put(req);
126         portCounter[3] <= portCounter[3] + 1;
127         if(portCounter[3] == 9) begin
128           portsDone[3] <= True;
129         end
130       end
131     endrule
132
133     rule readPortD;
134       let v <- bram.portD.response.get();
135       $display("%d", v);
136     endrule
137
138     rule count;
139       counter <= counter + 1;
```

```
140       endrule
141
142       function testForTrue(x) = x;
143       rule endSimulation( all(testForTrue, readVReg(portsDone)) );
144         $display("Finished at time %d.", $time);
145         $finish();
146       endrule
147
148     endmodule
149
150   endpackage
```

Aufgabe 3.2.1 Scheduling

Implementieren Sie nun den obigen RAM aber benutzen Sie statt Round Robin Schedulingattribute.

Probieren Sie zunächst eine Lösung ohne Attribute.

Implementieren Sie abschließend eine möglichst schnell Implementierung mit Hilfe von Attributen. Ist Ihre Lösung auch schnell wenn sich die Datenfrequenz an den Ports ändert?

```
1   package SchedRam;
2
3     import BRAM   :: *;
4     import Vector :: *;
5     import FIFO   :: *;
6
7     typedef BRAMRequest#(UInt#(15), Bit#(16)) SchedRamRequest;
8
9     interface SchedRam;
10      interface BRAMServer#(UInt#(15), Bit#(16)) portA;
11      interface BRAMServer#(UInt#(15), Bit#(16)) portB;
12      interface BRAMServer#(UInt#(15), Bit#(16)) portC;
13      interface BRAMServer#(UInt#(15), Bit#(16)) portD;
14    endinterface
15
16    module mkSchedRam(SchedRam);
17      Vector#(4, FIFO#(SchedRamRequest))   inFifos <- replicateM(mkFIFO());
18      Vector#(4, FIFO#(Bit#(16)))         outFifos <- replicateM(mkFIFO());
19      FIFO#(UInt#(2))                     readFifo <- mkSizedFIFO(16);
20      Vector#(4, Rules)                   portRules;
21
22      BRAM_Configure cfg = defaultValue; //declare variable cfg
23      cfg.memorySize = 1024*32; //new value for memorySize
24      BRAM1Port#(UInt#(15), Bit#(16)) bram <- mkBRAM1Server (cfg);
25
26      for(Integer i = 0; i < 4; i = i + 1) begin
27        portRules[i] = rules
28          rule handlePort;
29            bram.portA.request.put(inFifos[i].first());
30            inFifos[i].deq();
31            if(!inFifos[i].first().write || inFifos[i].first().responseOnWrite) readFifo.enq(fromInteger(i));
32          endrule
33          rule handleReadPort(readFifo.first() == fromInteger(i));
34            let retVal <- bram.portA.response.get();
35            outFifos[i].enq(retVal);
36            readFifo.deq();
```

```
37          endrule
38        endrules;
39      end
40      Rules allPorts = rJoinPreempts(portRules[0],portRules[2]);
41      allPorts       = rJoinPreempts(portRules[1],allPorts);
42      addRules(rJoinPreempts(portRules[3],allPorts));
43
44      // Interfaces
45      interface Server portA;
46        interface request  = toPut(inFifos[0]);
47        interface response = toGet(outFifos[0]);
48      endinterface
49
50      interface Server portB;
51        interface request  = toPut(inFifos[1]);
52        interface response = toGet(outFifos[1]);
53      endinterface
54
55      interface Server portC;
56        interface request  = toPut(inFifos[2]);
57        interface response = toGet(outFifos[2]);
58      endinterface
59
60      interface Server portD;
61        interface request  = toPut(inFifos[3]);
62        interface response = toGet(outFifos[3]);
63      endinterface
64    endmodule
65
66    module mkSchedRamTb(Empty);
67      SchedRam bram <- mkSchedRam();
68
69      Vector#(4, Reg#(Bool))      portsDone   <- replicateM(mkReg(False));
70      Vector#(4, Reg#(UInt#(12))) portCounter <- replicateM(mkReg(0));
71      Reg#(UInt#(32))             counter     <- mkReg(0);
72
73      rule putPortA;
74        if(!portsDone[0]) begin
75          SchedRamRequest req;
76          req.write = True;
77          req.responseOnWrite = False;
78          req.address = 0;
79          req.datain = extend(pack(portCounter[0]));
80          bram.portA.request.put(req);
81          portCounter[0] <= portCounter[0] + 1;
82          if(portCounter[0] == 9) begin
83            portsDone[0] <= True;
84          end
85        end
86      endrule
87
88      rule putPortB ((counter & 32'b1) == 0);
89        if(!portsDone[1]) begin
90          SchedRamRequest req;
91          req.write = True;
```

```
92          req.responseOnWrite = False;
93          req.address = 4;
94          req.datain = extend(pack(portCounter[1]));
95          bram.portB.request.put(req);
96          portCounter[1] <= portCounter[1] + 1;
97          if(portCounter[1] == 9) begin
98            portsDone[1] <= True;
99          end
100       end
101     endrule
102
103     rule putPortC;
104       if(!portsDone[2]) begin
105         RRRamRequest req;
106         req.write = True;
107         req.responseOnWrite = False;
108         req.address = 12;
109         req.datain = extend(pack(portCounter[2]));
110         bram.portC.request.put(req);
111         portCounter[2] <= portCounter[2] + 1;
112         if(portCounter[2] == 9) begin
113           portsDone[2] <= True;
114         end
115       end
116     endrule
117
118     rule putPortD ((counter & 32'b111) == 0);
119       if(!portsDone[3]) begin
120         RRRamRequest req;
121         req.write = False;
122         req.responseOnWrite = False;
123         req.address = 12;
124         req.datain = extend(pack(portCounter[3]));
125         bram.portD.request.put(req);
126         portCounter[3] <= portCounter[3] + 1;
127         if(portCounter[3] == 9) begin
128           portsDone[3] <= True;
129         end
130       end
131     endrule
132
133     rule readPortD;
134       let v <- bram.portD.response.get();
135       $display("%d", v);
136     endrule
137
138     rule count;
139       counter <= counter + 1;
140     endrule
141
142     function testForTrue(x) = x;
143     rule endSimulation( all(testForTrue, readVReg(portsDone)) );
144       $display("Finished at time %d.", $time);
145       $finish();
146     endrule
```

```
147
148     endmodule
149
150  endpackage
```