Übung zur Vorlesung Architekturen und Entwurf von Rechnersystemen

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

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 16/17
Übungsblatt 6

In den letzten Übungen wurden vor allem streambasierte Architekturen betrachtet. Dem Gegenüber werden in System on Chips (SoCs) häufig Architekturen verwendet die auf einen gemeinsamen Speicher zugreifen und die über Busse konfiguriert werden können. Diese Übung widmet sich diesen Bussystemen durch die Entwicklung des ESA-Bus. Der ESA-Bus ist ein sehr einfach gehaltener Bus der vergleichbar ist mit AXI4-Lite. Lese- und Schreibrichtungen sind getrennt. Die Busbreite ist 8 bit und die Adressbreite sind 16 bit. Der Bus ist Byte-adressiert. Der Schreibkanal hat keine Rückrichtung. Außerdem fehlen, zur Vereinfachung, viele Signale, z.B. ein Tag, die ein Produktiv verwendeter Bus hat. Des weiteren sind keine Burst-Transfers erlaubt.

Signal	Beschreibung
raready	Slave kann Leseanfrage annehmen.
ravalid	Master sendet gültige Daten.
raaddr	Leseadresse
rrready	Master kann Antwort annehmen.
rrvalid	Slave möchte Antwort senden.
rrdata	Gelesene Daten.

Tabelle 1: Signale der Leserichtung

Signal	Beschreibung
wready	Slave kann Schreibanfrage annehmen.
wvalid	Master sendet gültige Daten.
waddr	Schreibadresse
wdata	Schreibdaten

Tabelle 2: Signale der Schreibrichtung

Aufgabe 6.1 Master und Slave

Erstellen Sie einen passenden Master und Slave. Achten Sie darauf, dass Master und Slave möglichst leicht wieder zu verwenden ist. Ein Slave kann z.B. ein Speicher sein, aber auch die Konfigurationsregister eines Bildfiltermoduls.

Überlegen Sie sich geeignete Interfaces. Bedenken Sie, dass Bluespec bei `Action/ActionValue` Methoden automatisch Handshake-Signale generiert.

Lösungsvorschlag

```
1  package ESABus;
2
3      import GetPut :: *;
4      import FIFO :: *;
```

```
import Connectable :: *;

////////////////////////////////////
// Write Direction
////////////////////////////////////

// Master

interface ESABusWr_Master_Fabric;
    method ActionValue#(Tuple2#(Bit#(16), Bit#(8))) request;
endinterface

interface ESABusWr_Master;
    interface ESABusWr_Master_Fabric fab;

    interface Put#(Tuple2#(Bit#(16), Bit#(8))) request;
endinterface

module mkESABusWr_Master(ESABusWr_Master);

    FIFO#(Tuple2#(Bit#(16), Bit#(8))) requestIn <- mkFIFO();

    interface ESABusWr_Master_Fabric fab;
        method ActionValue#(Tuple2#(Bit#(16), Bit#(8))) request;
            requestIn.deq();
            return requestIn.first();
        endmethod
    endinterface

    interface Put request = toPut(requestIn);
endmodule

// Slave

interface ESABusWr_Slave_Fabric;
    method Action write(Bit#(16) addr, Bit#(8) data);
endinterface

interface ESABusWr_Slave;
    interface ESABusWr_Slave_Fabric fab;

    interface Get#(Tuple2#(Bit#(16), Bit#(8))) request;
endinterface

module mkESABusWr_Slave(ESABusWr_Slave);
    FIFO#(Tuple2#(Bit#(16), Bit#(8))) requestIn <- mkFIFO();

    interface ESABusWr_Slave_Fabric fab;
        method Action write(Bit#(16) addr, Bit#(8) data);
            requestIn.enq(tuple2(addr, data));
        endmethod
    endinterface

    interface Get request = toGet(requestIn);
```

```
59      endmodule
60
61      // Connection
62      instance Connectable#(ESABusWr_Master_Fabric, ESABusWr_Slave_Fabric);
63          module mkConnection#(ESABusWr_Master_Fabric a, ESABusWr_Slave_Fabric b)(Empty);
64              rule forwardWrite;
65                  let d <- a.request();
66                  b.write(tpl_1(d), tpl_2(d));
67              endrule
68          endmodule
69      endinstance
70
71      ////////////////////////////////////
72      // Read Direction
73      ////////////////////////////////////
74
75      // Master
76
77      interface ESABusRd_Master_Fabric;
78          method ActionValue#(Bit#(16)) addr;
79          method Action data(Bit#(8) d);
80      endinterface
81
82      interface ESABusRd_Master;
83          interface ESABusRd_Master_Fabric fab;
84
85          interface Put#(Bit#(16)) request;
86          interface Get#(Bit#(8)) response;
87      endinterface
88
89      module mkESABusRd_Master(ESABusRd_Master);
90
91          FIFO#(Bit#(16)) requestIn <- mkFIFO();
92          FIFO#(Bit#(8)) responseOut <- mkFIFO();
93
94          interface ESABusRd_Master_Fabric fab;
95              method ActionValue#(Bit#(16)) addr;
96                  requestIn.deq();
97                  return requestIn.first();
98              endmethod
99
100             method Action data(Bit#(8) d);
101                 responseOut.enq(d);
102             endmethod
103         endinterface
104
105         interface Put request = toPut(requestIn);
106         interface Get response = toGet(responseOut);
107     endmodule
108
109     // Slave
110
111     interface ESABusRd_Slave_Fabric;
112         method Action addr(Bit#(16) addr);
```

```
113             method ActionValue#(Bit#(8)) data();
114         endinterface
115
116         interface ESABusRd_Slave;
117             interface ESABusRd_Slave_Fabric fab;
118
119             interface Get#(Bit#(16)) request;
120             interface Put#(Bit#(8)) response;
121         endinterface
122
123         module mkESABusRd_Slave(ESABusRd_Slave);
124             FIFO#(Bit#(16)) requestIn <- mkFIFO();
125             FIFO#(Bit#(8)) responseOut <- mkFIFO();
126
127             interface ESABusRd_Slave_Fabric fab;
128                 method Action addr(Bit#(16) a);
129                     requestIn.enq(a);
130                 endmethod
131
132                 method ActionValue#(Bit#(8)) data();
133                     responseOut.deq();
134                     return responseOut.first();
135                 endmethod
136             endinterface
137
138             interface Get request = toGet(requestIn);
139             interface Put response = toPut(responseOut);
140         endmodule
141
142         // Connection
143         instance Connectable#(ESABusRd_Master_Fabric, ESABusRd_Slave_Fabric);
144             module mkConnection#(ESABusRd_Master_Fabric a, ESABusRd_Slave_Fabric b)(Empty);
145                 rule forwardRequest;
146                     let d <- a.addr();
147                     b.addr(d);
148                 endrule
149
150                 rule forwardResponse;
151                     let d <- b.data();
152                     a.data(d);
153                 endrule
154             endmodule
155         endinstance
156
157     endpackage
```

Aufgabe 6.2 Master und Slave verbinden

Erstellen Sie ein Modul das einen Master und einen Slave verbinden kann. Das Interface ist dabei `Empty` und Master und Slave werden als Parameter an das Modul übergeben. Sie können die `Connectable`-Typklasse aus der AzureIP Bibliothek verwenden.

Aufgabe 6.3 Speicherslave

Erstellen Sie einen Speicher mit dem ESA-Bus Interface. Der Speicher soll 4096 Einträge vom Typ `Bit#(8)` haben. Benutzen Sie als Grundlage des Speichers das `RegFile` aus der AzureIP Bibliothek. Details zur Verwendungen finden Sie im Bluespec Reference Guide.

Testen Sie die Kommunikation über den Bus mit BlueCheck. Vergleichen Sie dazu ein lokales RegFile mit dem über ESA-Bus angebundenen Speicher.

Lösungsvorschlag

```
1  package ESAMem;
2
3      import BlueCheck :: *;
4      import FIFO :: *;
5      import GetPut :: *;
6      import Connectable :: *;
7      import RegFile :: *;
8      import Clocks :: *;
9      import StmtFSM :: *;
10
11     import ESABus :: *;
12
13     interface ESAMem;
14         interface ESABusWr_Slave_Fabric s_wr;
15         interface ESABusRd_Slave_Fabric s_rd;
16     endinterface
17
18     module mkESAMem(ESAMem);
19         ESABusWr_Slave writeSlave <- mkESABusWr_Slave();
20         ESABusRd_Slave readSlave <- mkESABusRd_Slave();
21
22         RegFile#(Bit#(12), Bit#(8)) mem <- mkRegFileFull();
23
24         rule handleWrite;
25             let r <- writeSlave.request.get();
26             mem.upd(truncate(tpl_1(r)), tpl_2(r));
27         endrule
28
29         rule handleRead;
30             let r <- readSlave.request.get();
31             readSlave.response.put(mem.sub(truncate(r)));
32         endrule
33
34         interface ESABusWr_Slave_Fabric s_wr = writeSlave.fab;
35         interface ESABusRd_Slave_Fabric s_rd = readSlave.fab;
36     endmodule
37
38     module [BlueCheck] mkESAMemSpec#(Reset r)();
39         RegFile#(Bit#(12), Bit#(8)) mem <- mkRegFileFull(reset_by r);
40         FIFO#(Bit#(8)) readRequest <- mkFIFO(reset_by r);
41         ESAMem impl <- mkESAMem(reset_by r);
42
43         ESABusWr_Master writeMaster <- mkESABusWr_Master(reset_by r);
44         ESABusRd_Master readMaster <- mkESABusRd_Master(reset_by r);
45         mkConnection(writeMaster.fab, impl.s_wr, reset_by r);
```

```
46          mkConnection(readMaster.fab, impl.s_rd, reset_by r);
47
48          function Action getMem(Bit#(16) v);
49              action
50                  readRequest.enq(mem.sub(truncate(v)));
51              endaction
52          endfunction
53
54          function Action writeMem(Tuple2#(Bit#(16), Bit#(8)) r);
55              action
56                  mem.upd(truncate(tpl_1(r)), tpl_2(r));
57              endaction
58          endfunction
59
60          equiv("upd", writeMem, writeMaster.request.put);
61          equiv("sub", getMem, readMaster.request.put);
62          equiv("read", toGet(readRequest).get, readMaster.response.get);
63      endmodule
64
65      module [Module] mkESAMemChecker ();
66          Clock clk <- exposeCurrentClock;
67          MakeResetIfc r <- mkReset(0, True, clk);
68
69          // Customise default BlueCheck parameters
70          BlueCheck_Params params = bcParamsID(r);
71          params.wedgeDetect      = True;
72          params.id.initialDepth  = 3;
73          function incr(x)        = x+1;
74          params.id.incDepth      = incr;
75          params.numIterations    = 25;
76          params.id.testsPerDepth = 100;
77
78          // Generate checker
79          Stmt s <- mkModelChecker(mkESAMemSpec(r.new_rst), params);
80          mkAutoFSM(s);
81      endmodule
82
83  endpackage
```

Aufgabe 6.4 Mehrere Master und Slaves (Schwierig)

Erstellen Sie ein Modul, dass die Kommunikation zwischen mehreren Slaves und Master regelt. Überlegen Sie sich ein geeignetes Arbitrationsverfahren (z.B. Round-Robin) um zu Entscheiden welcher Master aktiv sein darf. Jeder Slave bekommt einen 12 bit Adressraum. Die restlichen Bit der Adresse werden verwendet um den anzusprechenden Slave zu identifizieren.

Es darf immer nur ein Lesezugriff gleichzeitig aktiv sein, alle anderen Zugriffe werden pausiert. Die Zahl der Master und Slaves soll nicht vorgegeben sein. Übergeben Sie einen Vektor der Master und Slaves an das Modul.

Lösungsvorschlag

```
1   package ESAConnect;
2
3       import GetPut :: *;
4       import Vector :: *;
```

```
5       import Arbiter :: *;
6
7       import ESABus :: *;
8       import ESAMem :: *;
9
10      module mkESAWrConnector#(Vector#(nMaster, ESABusWr_Master_Fabric) master, Vector#(nSlaves,
    ↪  ESABusWr_Slave_Fabric) slaves)(Empty)
11                  provisos(Log#(nSlaves, slaveBits));
12          Arbiter_IFC#(nMaster) arbiter <- mkArbiter(False);
13
14          Rules sendRequestRules = emptyRules();
15
16          for(Integer i = 0; i < valueOf(nMaster); i = i + 1) begin
17              Wire#(Tuple2#(Bit#(16), Bit#(8))) request <- mkWire();
18              Wire#(Bit#(slaveBits)) activeSlave <- mkWire();
19
20              rule handleMaster;
21                  let r <- master[i].request();
22                  arbiter.clients[i].request();
23                  request <= r;
24
25                  Bit#(slaveBits) slaveSel = 0;
26                  if(valueOf(nSlaves) != 1)
27                      slaveSel = tpl_1(r)[15:16-valueOf(slaveBits)];
28                  activeSlave <= slaveSel;
29
30                  $display("Fetched request of master %d", i);
31              endrule
32
33              for(Integer j = 0; j < valueOf(nSlaves); j = j + 1) begin
34                  sendRequestRules = rJoinMutuallyExclusive(rules
35                      rule sendRequest if(arbiter.clients[i].grant() && (fromInteger(j) ==
    ↪  activeSlave || valueOf(nSlaves) == 1));
36                          slaves[j].write(tpl_1(request), tpl_2(request));
37                          $display("Master %d sending request to slave %d %x %x", i,
    ↪  activeSlave, tpl_1(request), tpl_2(request));
38                      endrule
39                  endrules, sendRequestRules);
40              end
41          end
42
43          addRules(sendRequestRules);
44      endmodule
45
46      module mkESARdConnector#(Vector#(nMaster, ESABusRd_Master_Fabric) master, Vector#(nSlaves,
    ↪  ESABusRd_Slave_Fabric) slaves)(Empty)
47                  provisos(Log#(nSlaves, slaveBits));
48
49          Arbiter_IFC#(nMaster) arbiter <- mkArbiter(False);
50
51          Reg#(Bool) transferActive <- mkReg(False);
52
53          Rules sendRequestRules = emptyRules();
54
```

```
55          for(Integer i = 0; i < valueOf(nMaster); i = i + 1) begin
56              Wire#(Bit#(16)) request <- mkWire();
57              Reg#(Bit#(slaveBits)) activeSlave[2] <- mkCReg(2, 0);
58
59              Reg#(Bool) iAmActive <- mkReg(False);
60
61              rule handleMaster if(!transferActive);
62                  let r <- master[i].addr();
63                  arbiter.clients[i].request();
64                  request <= r;
65
66                  Bit#(slaveBits) slaveSel = 0;
67                  if(valueOf(nSlaves) != 1)
68                      slaveSel = r[15:16-valueOf(slaveBits)];
69                  activeSlave[0] <= slaveSel;
70                  $display("Fetched request of master %d", i);
71              endrule
72
73              for(Integer j = 0; j < valueOf(nSlaves); j = j + 1) begin
74                  sendRequestRules = rJoinMutuallyExclusive(rules
75                      rule sendRequest if(!transferActive && arbiter.clients[i].grant() &&
    ↪ (fromInteger(j) == activeSlave[1] || valueOf(nSlaves) == 1));
76
77                          slaves[j].addr(request);
78
79                          $display("Master %d sending request to slave %d %x", i,
    ↪ activeSlave[1], request);
80                          transferActive <= True;
81                          iAmActive <= True;
82                      endrule
83
84                      rule fetchResponse if(transferActive && iAmActive && (fromInteger(j) ==
    ↪ activeSlave[0] || valueOf(nSlaves) == 1));
85                          transferActive <= False;
86                          iAmActive <= False;
87                          let d <- slaves[j].data();
88                          master[i].data(d);
89                          $display("Master %d got response %d", i, d);
90                      endrule
91                  endrules, sendRequestRules);
92              end
93          end
94
95          addRules(sendRequestRules);
96      endmodule
97
98      import StmtFSM :: *;
99
100     module mkMultiMasterTest(Empty);
101
102         Vector#(2, ESABusWr_Master) master_wr <- replicateM(mkESABusWr_Master);
103         Vector#(2, ESABusRd_Master) master_rd <- replicateM(mkESABusRd_Master);
104         Vector#(2, ESAMem) slave <- replicateM(mkESAMem);
105
```

```
106         Vector#(2, ESABusWr_Master_Fabric) masterIfc_Wr;
107         Vector#(2, ESABusWr_Slave_Fabric) slaveIfc_Wr;
108
109         Vector#(2, ESABusRd_Master_Fabric) masterIfc_Rd;
110         Vector#(2, ESABusRd_Slave_Fabric) slaveIfc_Rd;
111
112         for(Integer i = 0; i < 2; i = i + 1) begin
113             masterIfc_Wr[i] = master_wr[i].fab;
114             masterIfc_Rd[i] = master_rd[i].fab;
115             slaveIfc_Wr[i] = slave[i].s_wr;
116             slaveIfc_Rd[i] = slave[i].s_rd;
117         end
118
119         mkESAWrConnector(masterIfc_Wr, slaveIfc_Wr);
120         mkESARdConnector(masterIfc_Rd, slaveIfc_Rd);
121
122         Stmt fsm = {
123             seq
124                 $display("Starting test");
125                 master_wr[0].request.put(tuple2(16'hAAAB, 8'hCC));
126                 master_wr[1].request.put(tuple2(16'hAAAB, 8'hDD));
127                 master_rd[0].request.put(16'hAAAB);
128                 action
129                     let x <- master_rd[0].response.get();
130                     $display("Got response %x", x);
131                 endaction
132                 master_wr[0].request.put(tuple2(16'h0AAB, 8'hCC));
133                 master_wr[1].request.put(tuple2(16'hAAAB, 8'hCC));
134                 delay(1000);
135             endseq
136         };
137
138         mkAutoFSM(fsm);
139     endmodule
140
141     module mkSingleSlaveTest(Empty);
142
143         Vector#(2, ESABusWr_Master) master_wr <- replicateM(mkESABusWr_Master);
144         Vector#(2, ESABusRd_Master) master_rd <- replicateM(mkESABusRd_Master);
145         Vector#(1, ESAMem) slave <- replicateM(mkESAMem);
146
147         Vector#(2, ESABusWr_Master_Fabric) masterIfc_Wr;
148         Vector#(1, ESABusWr_Slave_Fabric) slaveIfc_Wr;
149
150         Vector#(2, ESABusRd_Master_Fabric) masterIfc_Rd;
151         Vector#(1, ESABusRd_Slave_Fabric) slaveIfc_Rd;
152
153         for(Integer i = 0; i < 2; i = i + 1) begin
154             masterIfc_Wr[i] = master_wr[i].fab;
155             masterIfc_Rd[i] = master_rd[i].fab;
156         end
157
158         for(Integer i = 0; i < 1; i = i + 1) begin
159             slaveIfc_Wr[i] = slave[i].s_wr;
```

```
160             slaveIfc_Rd[i] = slave[i].s_rd;
161         end
162
163         mkESAWrConnector(masterIfc_Wr, slaveIfc_Wr);
164         mkESARdConnector(masterIfc_Rd, slaveIfc_Rd);
165
166         Stmt fsm = {
167             seq
168                 $display("Starting test");
169                 master_wr[0].request.put(tuple2(16'hAAAB, 8'hCC));
170                 master_wr[1].request.put(tuple2(16'hAAAB, 8'hDD));
171                 master_rd[0].request.put(16'hAAAB);
172                 action
173                     let x <- master_rd[0].response.get();
174                     $display("Got response %x", x);
175                 endaction
176                 master_wr[0].request.put(tuple2(16'h0AAB, 8'hCC));
177                 master_wr[1].request.put(tuple2(16'hAAAB, 8'hCC));
178                 delay(1000);
179             endseq
180         };
181
182         mkAutoFSM(fsm);
183     endmodule
184
185 endpackage
```