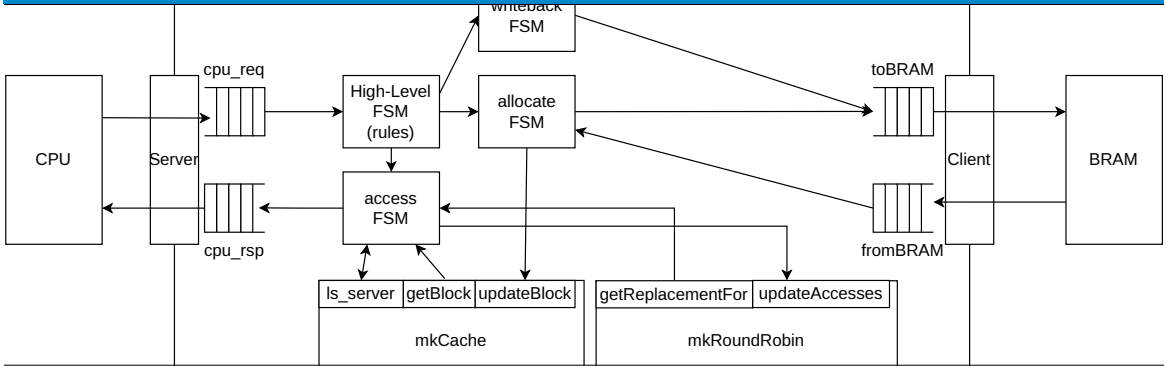
Architekturen und Entwurf von Rechnersystemen
Besprechung Übung 6

Wintersemester 2022/2023

Johannes Wirth
Fachgebiet Eingebettete Systeme und ihre Anwendungen

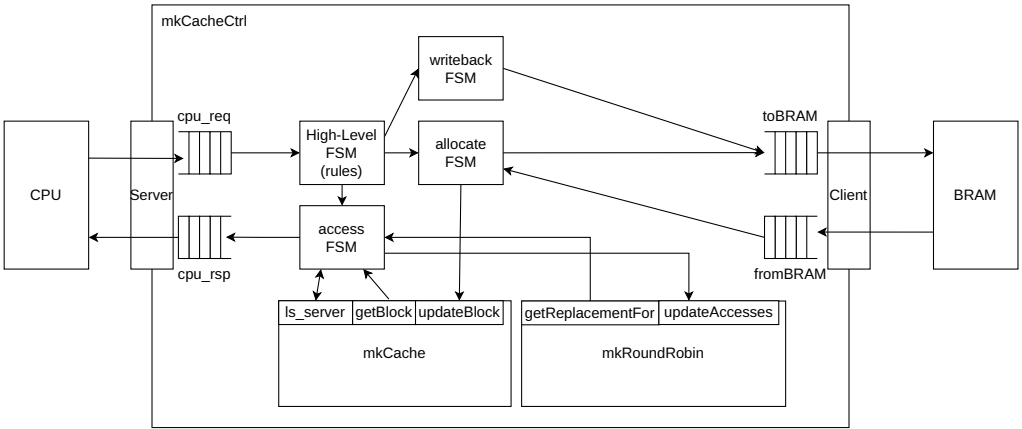


TECHNISCHE
UNIVERSITÄT
DARMSTADT

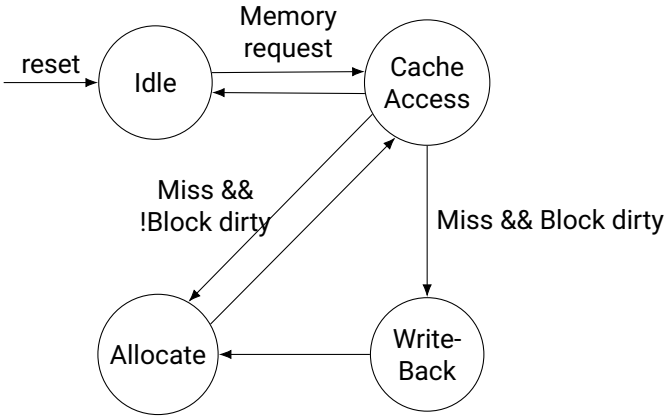
Übung 6: Cache Controller & AXI

Cache Controller

Cache Controller



High-Level FSM



Aufgabe 6.1c - IDLE-State

TECHNISCHE UNIVERSITÄT DARMSTADT

- Warten auf neuen Request von CPU, speichern in `curr_req`.
- Wechsel in Zustand `CacheAccess`.

Aufgabe 6.1c - IDLE-State

TECHNISCHE UNIVERSITÄT DARMSTADT

- Warten auf neuen Request von CPU, speichern in **`curr_req`**.
- Wechsel in Zustand **`CacheAccess`**.

```
1 rule idle(state == Idle);
2     curr_req <= cpu_req.first();
3     cpu_req.deq();
4     state <= Access;
5 endrule
```

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der aktuelle Speicherrequest liegt im Register `curr_req`.
- Die Anfrage soll zunächst an den Cache weitergeleitet werden.

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der aktuelle Speicherrequest liegt im Register **`curr_req`**.
- Die Anfrage soll zunächst an den Cache weitergeleitet werden.

```
1 action
2     DecodedAddress req_addr;
3     CacheReq to_cache;
4     if(curr_req matches tagged Read .rd) begin
5         req_addr = decode(rd.addr);
6         to_cache = tagged Load req_addr;
7     end
```

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der aktuelle Speicherrequest liegt im Register **`curr_req`**.
- Die Anfrage soll zunächst an den Cache weitergeleitet werden.

```
1  action
2      DecodedAddress req_addr;
3      CacheReq to_cache;
4      if(curr_req matches tagged Read .rd) begin
5          req_addr = decode(rd.addr);
6          to_cache = tagged Load req_addr;
7      end
8      else begin
9          WriteReq wr = curr_req.Write; // There are only two tags, so we know this is the one.
10         req_addr = decode(wr.addr);
11         Data req_data = wr.data;
12         to_cache = tagged Store StoreReq{addr: req_addr, data: req_data};
13     end
```

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der aktuelle Speicherrequest liegt im Register **`curr_req`**.
- Die Anfrage soll zunächst an den Cache weitergeleitet werden.

```
1  action
2      DecodedAddress req_addr;
3      CacheReq to_cache;
4      if(curr_req matches tagged Read .rd) begin
5          req_addr = decode(rd.addr);
6          to_cache = tagged Load req_addr;
7      end
8      else begin
9          WriteReq wr = curr_req.Write; // There are only two tags, so we know this is the one.
10         req_addr = decode(wr.addr);
11         Data req_data = wr.data;
12         to_cache = tagged Store StoreReq{addr: req_addr, data: req_data};
13     end
14     curr_addr <= req_addr;
15     cache.ls_server.request.put(to_cache);
16 endaction
```

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei einem Cache Hit wird

- die Response an den Prozessor weitergeleitet.
- der Zustand der Ersetzungsstrategie aktualisiert (`replacer.updateAccesses(Index, Way)`).
- der Controller wieder in den `Idle`-Zustand versetzt.

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei einem Cache Hit wird

- die Response an den Prozessor weitergeleitet.
- der Zustand der Ersetzungsstrategie aktualisiert (**`replacer.updateAccesses(Index, Way)`**).
- der Controller wieder in den **`Idle`**-Zustand versetzt.

```
1  action
2      let rsp <- cache.ls_server.response.get();
3      if(rsp matches tagged Valid .from_cache) begin // handle hit
4          CtrlRsp ctrl_rsp;
5          Way accessed_way;
6          // omitted: extract from rsp
10     end
```

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei einem Cache Hit wird

- die Response an den Prozessor weitergeleitet.
- der Zustand der Ersetzungsstrategie aktualisiert (**`replacer.updateAccesses(Index, Way)`**).
- der Controller wieder in den **`Idle`**-Zustand versetzt.

```
1  action
2      let rsp <- cache.ls_server.response.get();
3      if(rsp matches tagged Valid .from_cache) begin // handle hit
4          CtrlRsp ctrl_rsp;
5          Way accessed_way;
6          // omitted: extract from rsp
7          cpu_rsp.enq(ctrl_rsp); // Forward result to CPU
10     end
```

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei einem Cache Hit wird

- die Response an den Prozessor weitergeleitet.
- der Zustand der Ersetzungsstrategie aktualisiert (**`replacer.updateAccesses(Index, Way)`**).
- der Controller wieder in den **`Idle`**-Zustand versetzt.

```
1  action
2      let rsp <- cache.ls_server.response.get();
3      if(rsp matches tagged Valid .from_cache) begin // handle hit
4          CtrlRsp ctrl_rsp;
5          Way accessed_way;
6          // omitted: extract from rsp
7          cpu_rsp.enq(ctrl_rsp); // Forward result to CPU
8          replacer.updateAccesses(curr_addr.index, accessed_way);


10     end
```

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei einem Cache Hit wird

- die Response an den Prozessor weitergeleitet.
- der Zustand der Ersetzungsstrategie aktualisiert (**`replacer.updateAccesses(Index, Way)`**).
- der Controller wieder in den **`Idle`**-Zustand versetzt.

```
1  action
2      let rsp <- cache.ls_server.response.get();
3      if(rsp matches tagged Valid .from_cache) begin // handle hit
4          CtrlRsp ctrl_rsp;
5          Way accessed_way;
6          // omitted: extract from rsp
7          cpu_rsp.enq(ctrl_rsp); // Forward result to CPU
8          replacer.updateAccesses(curr_addr.index, accessed_way);
9          state <= Idle; // Ready for next request
10     end
```

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei einem Cache Miss wird

- bestimmt welcher Cache-Eintrag im passenden Set ersetzt wird
- der entsprechende Eintrag aus dem Cache ausgelesen und bestimmt, ob dieser in den Speicher zurückgeschrieben wird.
- Falls ja, Wechsel in den State **`Writeback`**.
- Falls nein, Wechsel in den State **`Allocate`**.

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei einem Cache Miss wird

- bestimmt welcher Cache-Eintrag im passenden Set ersetzt wird
- der entsprechende Eintrag aus dem Cache ausgelesen und bestimmt, ob dieser in den Speicher zurückgeschrieben wird.
- Falls ja, Wechsel in den State **`Writeback`**.
- Falls nein, Wechsel in den State **`Allocate`**.

```
1  else begin // prepare eviction
2      let new_way <- replacer.getReplacementFor(curr_addr.index);
3      evicted_way <= new_way;
11 end
```

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei einem Cache Miss wird

- bestimmt welcher Cache-Eintrag im passenden Set ersetzt wird
- der entsprechende Eintrag aus dem Cache ausgelesen und bestimmt, ob dieser in den Speicher zurückgeschrieben wird.
- Falls ja, Wechsel in den State **`Writeback`**.
- Falls nein, Wechsel in den State **`Allocate`**.

```
1  else begin // prepare eviction
2      let new_way <- replacer.getReplacementFor(curr_addr.index);
3      evicted_way <= new_way;
4      let block = cache.getBlock(curr_addr.index, new_way);
11 end
```

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei einem Cache Miss wird

- bestimmt welcher Cache-Eintrag im passenden Set ersetzt wird
- der entsprechende Eintrag aus dem Cache ausgelesen und bestimmt, ob dieser in den Speicher zurückgeschrieben wird.
- Falls ja, Wechsel in den State **`Writeback`**.
- Falls nein, Wechsel in den State **`Allocate`**.

```
1  else begin // prepare eviction
2      let new_way <- replacer.getReplacementFor(curr_addr.index);
3      evicted_way <= new_way;
4      let block = cache.getBlock(curr_addr.index, new_way);
5      if(block matches tagged Valid .vblock &&& vblock.dirty) begin
6          state <= WriteBack;
7          // omitted
8      end


11 end
```

Aufgabe 6.1d - ACCESS-State

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei einem Cache Miss wird

- bestimmt welcher Cache-Eintrag im passenden Set ersetzt wird
- der entsprechende Eintrag aus dem Cache ausgelesen und bestimmt, ob dieser in den Speicher zurückgeschrieben wird.
- Falls ja, Wechsel in den State **`Writeback`**.
- Falls nein, Wechsel in den State **`Allocate`**.

```
1  else begin // prepare eviction
2      let new_way <- replacer.getReplacementFor(curr_addr.index);
3      evicted_way <= new_way;
4      let block = cache.getBlock(curr_addr.index, new_way);
5      if(block matches tagged Valid .vblock &&& vblock.dirty) begin
6          state <= WriteBack;
7          // omitted
8      end
9      else
10         state <= Allocate;
11 end
```

Aufgabe 6.1f - WRITEBACK-State

TECHNISCHE UNIVERSITÄT DARMSTADT

- Block-Adresse im Register **`evicated_addr`**.
- Schreibe alle Wörter des Blocks in Hauptspeicher.

Aufgabe 6.1f - WRITEBACK-State

TECHNISCHE UNIVERSITÄT DARMSTADT

- Block-Adresse im Register **`evicated_addr`**.
- Schreibe alle Wörter des Blocks in Hauptspeicher.

```
1  seq
2      for(/* iterate over all words in block */) action
3          let byte_offset = (curr_bram_addr - encode(evicted_addr))
4          word_offset = byte_offset » log2(fromInteger(valueOf(BYTES_PER_WORD)));
5          requestFromBRAM(
6              curr_bram_addr,
7              True, // True => Write
8              evicted_entry.data_block[word_offset]);
9      endaction
10     state <= Allocate;
11 endseq
```

Aufgabe 6.1e - ALLOCATE-State

TECHNISCHE UNIVERSITÄT DARMSTADT

- Berechnen aller Wortadressen in einem Block.
- Alle Wörter des Blocks aus Hauptspeicher anfragen.

Aufgabe 6.1e - ALLOCATE-State

TECHNISCHE UNIVERSITÄT DARMSTADT

- Berechnen aller Wortadressen in einem Block.
- Alle Wörter des Blocks aus Hauptspeicher anfragen.

```
1  par
2      // iterate over words in block
3      for(
4        curr_bram_addr <= computeBlockAddr(curr_addr);
5        curr_bram_addr < computeEndAddr(curr_addr);
6        curr_bram_addr <= curr_bram_addr + fromInteger(valueOf(BYTES_PER_WORD)))
7      action
8          requestFromBRAM(curr_bram_addr, False, 0); // False => Read
9      endaction
10     ...
```

Aufgabe 6.1e - ALLOCATE-State

TECHNISCHE UNIVERSITÄT DARMSTADT

- Annehmen der gelesenen Wörter.
- Eintragen in neuen `CacheEntry`.
- Aktualisieren des Cache-Blocks.

Aufgabe 6.1e - ALLOCATE-State

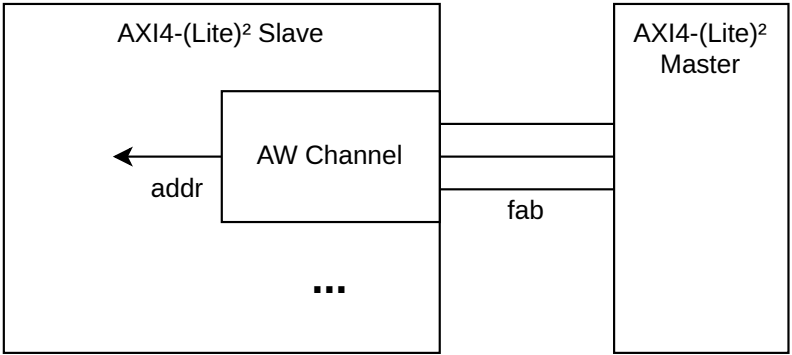
TECHNISCHE UNIVERSITÄT DARMSTADT

- Annehmen der gelesenen Wörter.
- Eintragen in neuen **`CacheEntry`**.
- Aktualisieren des Cache-Blocks.

```
1     ...
2     seq
3         line <= CacheEntry {dirty: False, tag: curr_addr.tag, data_block: unpack(0)};
4         for(/* iterate over words in block */) action
5             let bram_rsp = fromBRAM.first; fromBRAM.deq;
6             let ce = line;
7             ce.data_block[bram_rsp_count] = bram_rsp;
8             line <= ce;
9         endaction
10    endseq
11 endpar
12 action
13    cache.updateBlock(line, curr_addr.index, evicted_way);
14    state <= Access;
15 endaction
```

AXI4-(Lite)^2

Aufgabe 6.2 - Slave Architektur



Aufgabe 6.2a - Wires

TECHNISCHE UNIVERSITÄT DARMSTADT

Welche Wire-Implementierung wird benötigt?

```
1  (* always_ready, always_enabled *)
2  interface AXILiteLiteSlaveAW_Fab;
3    method Bool awready;
4    method Action awvalid(Bool valid);
5    method Action awaddr(Address addr);
6  endinterface
7
8  module mkAXILiteLiteSlaveAW(AXILiteLiteSlaveAW);
9    Wire#(Bool) validIn;
10   Wire#(Address) addrIn;
11 endmodule
```

Aufgabe 6.2a - Wires

TECHNISCHE UNIVERSITÄT DARMSTADT

Welche Wire-Implementierung wird benötigt?

```
1  (* always_ready, always_enabled *)
2  interface AXILiteLiteSlaveAW_Fab;
3    method Bool awready;
4    method Action awvalid(Bool valid);
5    method Action awaddr(Address addr);
6  endinterface
7
8  module mkAXILiteLiteSlaveAW(AXILiteLiteSlaveAW);
9    Wire#(Bool) validIn;
10   Wire#(Address) addrIn;
11 endmodule
```

Annotationen **`always_ready`** und **`always_enabled`** => **`mkBypassWire`**.

Aufgabe 6.2a - AW-Channel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface AXILiteLiteSlaveAW_Fab;
2    method Bool awready;
3    method Action awvalid(Bool valid);
4    method Action awaddr(Address addr);
5  endinterface
6
7  module mkAXILiteLiteSlaveAW(AXILiteLiteSlaveAW);
8      Wire#(Bool) validIn <- mkBypassWire();
9      Wire#(Address) addrIn <- mkBypassWire();
20 endmodule
```

Aufgabe 6.2a - AW-Channel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface AXILiteLiteSlaveAW_Fab;
2    method Bool awready;
3    method Action awvalid(Bool valid);
4    method Action awaddr(Address addr);
5  endinterface
6
7  module mkAXILiteLiteSlaveAW(AXILiteLiteSlaveAW);
8      Wire#(Bool) validIn <- mkBypassWire();
9      Wire#(Address) addrIn <- mkBypassWire();
10     FIFOF#(Address) addr_out <- mkFIFOF;
20 endmodule
```

Aufgabe 6.2a - AW-Channel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface AXILiteLiteSlaveAW_Fab;
2    method Bool awready;
3    method Action awvalid(Bool valid);
4    method Action awaddr(Address addr);
5  endinterface
6
7  module mkAXILiteLiteSlaveAW(AXILiteLiteSlaveAW);
8      Wire#(Bool) validIn <- mkBypassWire();
9      Wire#(Address) addrIn <- mkBypassWire();
10     FIFOF#(Address) addr_out <- mkFIFOF;




14     interface AXILiteLiteSlaveAW_Fab fab;
15         method awready = addr_out.notFull;
16         method awvalid = validIn._write;
17         method awaddr = addrIn._write;
18     endinterface
19     interface addr = toGet(addr_out);
20 endmodule
```

Aufgabe 6.2a - AW-Channel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface AXILiteLiteSlaveAW_Fab;
2    method Bool awready;
3    method Action awvalid(Bool valid);
4    method Action awaddr(Address addr);
5  endinterface
6
7  module mkAXILiteLiteSlaveAW(AXILiteLiteSlaveAW);
8      Wire#(Bool) validIn <- mkBypassWire();
9      Wire#(Address) addrIn <- mkBypassWire();
10     FIFOF#(Address) addr_out <- mkFIFOF;
11     rule transfer if (validIn && addr_out.notFull);
12       addr_out.enq(addrIn);
13     endrule
14     interface AXILiteLiteSlaveAW_Fab fab;
15         method awready = addr_out.notFull;
16         method awvalid = validIn._write;
17         method awaddr = addrIn._write;
18     endinterface
19     interface addr = toGet(addr_out);
20 endmodule
```

Aufgabe 6.2b - R-Channel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface AXILiteLiteSlaveR_Fab;
2      method Action rready(Bool ready);
3      method Bool rvalid;
4      method Data rdata();
5  endinterface
6
7  module mkAXILiteLiteSlaveR(AXILiteLiteSlaveR);
8      Reg#(Bool) validOut[2] <- mkCReg(2, False);
9      Reg#(Data) dataOut[2] <- mkCReg(2, 0);
10     Wire#(Bool) readyIn <- mkBypassWire();
19 endmodule
```


Embedded Systems & Applications

Aufgabe 6.2b - R-Channel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface AXILiteLiteSlaveR_Fab;
2      method Action rready(Bool ready);
3      method Bool rvalid;
4      method Data rdata();
5  endinterface
6
7  module mkAXILiteLiteSlaveR(AXILiteLiteSlaveR);
8      Reg#(Bool) validOut[2] <- mkCReg(2, False);
9      Reg#(Data) dataOut[2] <- mkCReg(2, 0);
10     Wire#(Bool) readyIn <- mkBypassWire();
11     FIFOF#(Data) data_in <- mkFIFOF;
19 endmodule
```

Aufgabe 6.2b - R-Channel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface AXILiteLiteSlaveR_Fab;
2      method Action rready(Bool ready);
3      method Bool rvalid;
4      method Data rdata();
5  endinterface
6
7  module mkAXILiteLiteSlaveR(AXILiteLiteSlaveR);
8      Reg#(Bool) validOut[2] <- mkCReg(2, False);
9      Reg#(Data) dataOut[2] <- mkCReg(2, 0);
10     Wire#(Bool) readyIn <- mkBypassWire();
11     FIFOF#(Data) data_in <- mkFIFOF;


13     interface AXILiteLiteSlaveR_Fab fab;
14         method rready = readyIn._write;
15         method rvalid = validOut[1];
16         method rdata = dataOut[1];
17     endinterface
18     interface read_data = toPut(data_in);
19 endmodule
```

Aufgabe 6.2b - R-Channel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface AXILiteLiteSlaveR_Fab;
2      method Action rready(Bool ready);
3      method Bool rvalid;
4      method Data rdata();
5  endinterface
6
7  module mkAXILiteLiteSlaveR(AXILiteLiteSlaveR);
8      Reg#(Bool) validOut[2] <- mkCReg(2, False);
9      Reg#(Data) dataOut[2] <- mkCReg(2, 0);
10     Wire#(Bool) readyIn <- mkBypassWire();
11     FIFOF#(Data) data_in <- mkFIFOF;
12     // ... Implementation
13     interface AXILiteLiteSlaveR_Fab fab;
14         method rready = readyIn._write;
15         method rvalid = validOut[1];
16         method rdata = dataOut[1];
17     endinterface
18     interface read_data = toPut(data_in);
19 endmodule
```

Aufgabe 6.2b - IDLE-Zustand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Keine Übertragung findet statt
- Wenn Datenpaket in FIFO, Wechsel nach **`TRANSFER`**

Aufgabe 6.2b - IDLE-Zustand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Keine Übertragung findet statt
- Wenn Datenpaket in FIFO, Wechsel nach **TRANSFER**

```
1    rule idle if (state == IDLE);

4    endrule
```

Aufgabe 6.2b - IDLE-Zustand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Keine Übertragung findet statt
- Wenn Datenpaket in FIFO, Wechsel nach **TRANSFER**

```
1   rule idle if (state == IDLE);
2       validOut[0] <= False;

4   endrule
```

Aufgabe 6.2b - IDLE-Zustand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Keine Übertragung findet statt
- Wenn Datenpaket in FIFO, Wechsel nach **TRANSFER**

```
1   rule idle if (state == IDLE);
2       validOut[0] <= False;
3       if (data_in.notEmpty) state <= TRANSFER;
4   endrule
```

Aufgabe 6.2b - TRANSFER-Zustand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Starte Übertragung mit erstem FIFO-Element
- Wenn **`ready`** == **`TRUE`**, Wechsel in Zustand **`IDLE`**
- Ansonsten Wechsel nach **`WAIT`**

Aufgabe 6.2b - TRANSFER-Zustand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Starte Übertragung mit erstem FIFO-Element
- Wenn **`ready`** == **`TRUE`**, Wechsel in Zustand **`IDLE`**
- Ansonsten Wechsel nach **`WAIT`**

```
1   rule transfer if (state == TRANSFER);

9   endrule
```

Aufgabe 6.2b - TRANSFER-Zustand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Starte Übertragung mit erstem FIFO-Element
- Wenn **`ready == TRUE`**, Wechsel in Zustand **`IDLE`**
- Ansonsten Wechsel nach **`WAIT`**

```
1   rule transfer if (state == TRANSFER);
2       let resp = data_in.first; data_in.deq;
3       validOut[0] <= True;
4       dataOut[0] <= resp;

9   endrule
```

Aufgabe 6.2b - TRANSFER-Zustand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Starte Übertragung mit erstem FIFO-Element
- Wenn **`ready == TRUE`**, Wechsel in Zustand **`IDLE`**
- Ansonsten Wechsel nach **`WAIT`**

```
1  rule transfer if (state == TRANSFER);
2      let resp = data_in.first; data_in.deq;
3      validOut[0] <= True;
4      dataOut[0] <= resp;
5      if (!readyIn)
6          state <= WAIT;
7      else
8          state <= IDLE;
9  endrule
```

Aufgabe 6.2b - WAIT-Zustand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Warten auf Gegenseite (Master)
- Wenn **`ready`** == **`TRUE`**, Wechsel in Zustand **`IDLE`**

Aufgabe 6.2b - WAIT-Zustand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Warten auf Gegenseite (Master)
- Wenn **`ready`** == **`TRUE`**, Wechsel in Zustand **`IDLE`**

```
1   rule wait if (state == WAIT);

3   endrule
```

Aufgabe 6.2b - WAIT-Zustand

TECHNISCHE UNIVERSITÄT DARMSTADT

- Warten auf Gegenseite (Master)
- Wenn **`ready`** == **`TRUE`**, Wechsel in Zustand **`IDLE`**

```
1   rule wait if (state == WAIT);
2       if (readyIn) state <= IDLE;
3   endrule
```

Fragen zur Übung