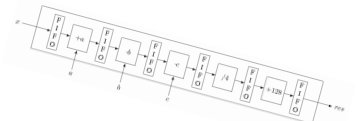
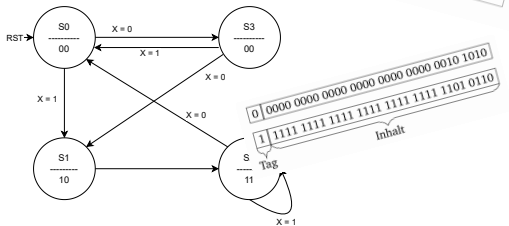
Architekturen und Entwurf von Rechnersystemen
Besprechung Übungsblatt 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 2022/2023

Johannes Wirth
Fachgebiet Eingebettete Systeme und ihre Anwendungen





Einfache Pipeline

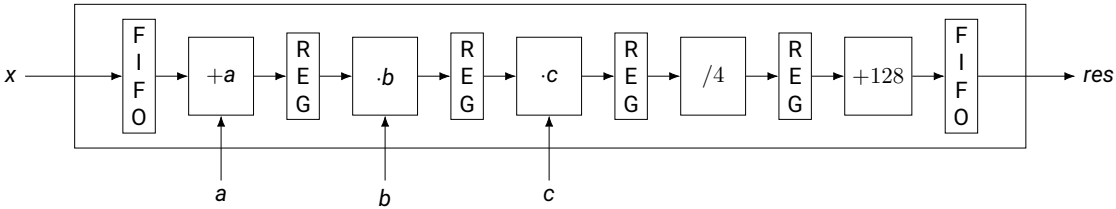


Abbildung: Pipeline aus Aufgabe 2.1.

Einfache Pipeline - Implementierung

Modul mkSimplePipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 FIFOF#(Int#(32)) fifo_in <- mkFIFOF;
2 FIFO#(Int#(32)) fifo_out <- mkFIFO;
3
4 Vector#(4, Reg#(Maybe#(Int#(32)))) regs <- replicateM(mkReg(tagged Invalid));
5
6 Vector#(3, Reg#(Int#(32))) params <- replicateM(mkRegU);
```

Einfache Pipeline - Implementierung

Modul mkSimplePipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 FIFOF#(Int#(32)) fifo_in <- mkFIFOF;
2 FIFO#(Int#(32)) fifo_out <- mkFIFO;
3
4 Vector#(4, Reg#(Maybe#(Int#(32)))) regs <- replicateM(mkReg(tagged Invalid));
5
6 Vector#(3, Reg#(Int#(32))) params <- replicateM(mkRegU);
```

Einfache Pipeline - Implementierung

Modul mkSimplePipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 FIFOF#(Int#(32)) fifo_in <- mkFIFOF;
2 FIFO#(Int#(32)) fifo_out <- mkFIFO;
3
4 Vector#(4, Reg#(Maybe#(Int#(32)))) regs <- replicateM(mkReg(tagged Invalid));
5
6 Vector#(3, Reg#(Int#(32))) params <- replicateM(mkRegU);
```

Einfache Pipeline - Implementierung

Modul mkSimplePipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 FIFOF#(Int#(32)) fifo_in <- mkFIFOF;
2 FIFO#(Int#(32)) fifo_out <- mkFIFO;
3
4 Vector#(4, Reg#(Maybe#(Int#(32)))) regs <- replicateM(mkReg(tagged Invalid));
5
6 Vector#(3, Reg#(Int#(32))) params <- replicateM(mkRegU);
```

Einfache Pipeline - Implementierung

Modul mkSimplePipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

Regel:

```
1  rule all_together;
2      // Stage 0
3      if (fifo_in.notEmpty) begin
4          let x = fifo_in.first; fifo_in.deq;
5          regs[0] <= tagged Valid (x + params[0]);
6      end else begin
7          regs[0] <= tagged Invalid;
8      end
9
10     // Stage 1
11     if (regs[0] matches tagged Valid .x) begin
12         regs[1] <= tagged Valid (x * params[1]);
13     end else begin
14         regs[1] <= tagged Invalid;
15     end
16
17     ...
```

Einfache Pipeline - Implementierung

Modul mkSimplePipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   ...
2
3     // Stage 2
4     if (regs[1] matches tagged Valid .x) begin
5         regs[2] <= tagged Valid (x * params[2]);
6     end else begin
7         regs[2] <= tagged Invalid;
8     end
9
10    // Stage 3
11    if (regs[2] matches tagged Valid .x) begin
12        regs[3] <= tagged Valid (x / 4); // alternatively » 2
13    end else begin
14        regs[3] <= tagged Invalid;
15    end
16
17    // Stage 4
18    if (regs[3] matches tagged Valid .x) begin
19        fifo_out.enq(x + 128);
20    end
21  endrule
```

Einfache Pipeline - Implementierung

Modul mkSimplePipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

Interface Definition:

```
1  method Action setParam(UInt#(2) addr, Int#(32) val);
2      params[addr] <= val;
3  endmethod
4
5  interface CalcUnit calc;
6      method Action put(Int#(32) x);
7          fifo_in.enq(x);
8      endmethod
9
10     method ActionValue#(Int#(32)) result();
11         fifo_out.deq();
12         return fifo_out.first;
13     endmethod
14 endinterface
```

Einfache Pipeline - Charakterisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Pipeline ist **statisch**

Einfache Pipeline - Charakterisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Pipeline ist **statisch**
  - Die Latenz ist immer gleich (datenunabhängig)

Einfache Pipeline - Charakterisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Pipeline ist **statisch**
  - Die Latenz ist immer gleich (datenunabhängig)
- Die Pipeline ist **starr**

Einfache Pipeline - Charakterisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Pipeline ist **statisch**
  - Die Latenz ist immer gleich (datenunabhängig)
- Die Pipeline ist **starr**
  - Die Daten bewegen sich im Gleichschritt durch die Pipeline

Einfache Pipeline - Charakterisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Pipeline ist **statisch**
  - Die Latenz ist immer gleich (datenunabhängig)
- Die Pipeline ist **starr**
  - Die Daten bewegen sich im Gleichschritt durch die Pipeline
  - Implementierung aller Stufen in einer Regel

FSMs in BSV - Basics

FSMs in BSV - Basics

TECHNISCHE UNIVERSITÄT DARMSTADT

- Package **`StmtFSM`**

FSMs in BSV - Basics

TECHNISCHE UNIVERSITÄT DARMSTADT

- Package **`StmtFSM`**
- Definition der FSMs über **`Stmt`** (eigene Sprache innerhalb von BSV)

FSMs in BSV - Basics

TECHNISCHE UNIVERSITÄT DARMSTADT

- Package **`StmtFSM`**
- Definition der FSMs über **`Stmt`** (eigene Sprache innerhalb von BSV)
  - **`seq s1; s2;endseq`**: Ablauf von **`s1`** und **`s2`** sequenziell (in verschiedenen Takten, s2 nach s1)

FSMs in BSV - Basics

TECHNISCHE UNIVERSITÄT DARMSTADT

- Package **`StmtFSM`**
- Definition der FSMs über **`Stmt`** (eigene Sprache innerhalb von BSV)
  - **`seq s1; s2;endseq`**: Ablauf von **`s1`** und **`s2`** sequenziell (in verschiedenen Takten, s2 nach s1)
  - **`par s1; s2;endpar`**: Ablauf von **`s1`** und **`s2`** nebenläufig (potenziell im gleichen Takt, unabhängig voneinander)

FSMs in BSV - Basics

TECHNISCHE UNIVERSITÄT DARMSTADT

- Package **`StmtFSM`**
- Definition der FSMs über **`Stmt`** (eigene Sprache innerhalb von BSV)
  - **`seq s1; s2;endseq`**: Ablauf von **`s1`** und **`s2`** sequenziell (in verschiedenen Takten, s2 nach s1)
  - **`par s1; s2;endpar`**: Ablauf von **`s1`** und **`s2`** nebenläufig (potenziell im gleichen Takt, unabhängig voneinander)
  - **`action`**...**`endaction`**: Atomarer Action-Block (Inhalt wird innerhalb eines Takts ausgeführt)

FSMs in BSV - Basics

TECHNISCHE UNIVERSITÄT DARMSTADT

- Package **`StmtFSM`**
- Definition der FSMs über **`Stmt`** (eigene Sprache innerhalb von BSV)
  - **`seq s1; s2;endseq`**: Ablauf von **`s1`** und **`s2`** sequenziell (in verschiedenen Takten, s2 nach s1)
  - **`par s1; s2;endpar`**: Ablauf von **`s1`** und **`s2`** nebenläufig (potenziell im gleichen Takt, unabhängig voneinander)
  - **`action`**...**`endaction`**: Atomarer Action-Block (Inhalt wird innerhalb eines Takts ausgeführt)
  - **`for(...) s1`**;/**`while(...) s1;`**: Wiederholte sequenzielle Auführung von **`s1`**

FSMs in BSV - Basics

TECHNISCHE UNIVERSITÄT DARMSTADT

- Package **`StmtFSM`**
- Definition der FSMs über **`Stmt`** (eigene Sprache innerhalb von BSV)
  - **`seq s1; s2;endseq`**: Ablauf von **`s1`** und **`s2`** sequenziell (in verschiedenen Takten, s2 nach s1)
  - **`par s1; s2;endpar`**: Ablauf von **`s1`** und **`s2`** nebenläufig (potenziell im gleichen Takt, unabhängig voneinander)
  - **`action`**...**`endaction`**: Atomarer Action-Block (Inhalt wird innerhalb eines Takts ausgeführt)
  - **`for(...) s1`**;/**`while(...) s1;`**: Wiederholte sequenzielle Auführung von **`s1`**
  - **`if(...) s1; else s2;`**: Bedingte Ausführung von Statements

Schleifen - BSV vs. FSM

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schleifen in BSV

Schleifen - BSV vs. FSM

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schleifen in BSV
  - strukturelle Verfielfältigung
  - Bsp: erzeuge mehrere “Instanzen” (von Registern, Regeln, ...)

Schleifen - BSV vs. FSM

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schleifen in BSV
  - strukturelle Verfielfältigung
  - Bsp: erzeuge mehrere “Instanzen” (von Registern, Regeln, ...)
- Schleifen in StmtFSM

Schleifen - BSV vs. FSM

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schleifen in BSV
  - strukturelle Verfielfältigung
  - Bsp: erzeuge mehrere “Instanzen” (von Registern, Regeln, ...)
- Schleifen in StmtFSM
  - zeitbasiertes Verhalten
  - Bsp: Körper wird mehrmals in verschiedenen Takten ausgeführt

FSM Testbench - Teil 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Stmt s = seq
2    dut.setParam(0, 42);
3    dut.setParam(1, 2);
4    dut.setParam(2, 13);
5    $display("Initialized parameters, starting test...");
6    par
7      seq
8        for(idx__put <= 0; idx__put < 10; idx__put <= idx__put + 1) action
9          dut.calc.put(unpack(pack(idx__put)));
10         // unpack(pack()) to cast from uint to int
11       endaction
12       dut.setParam(0, 7);
13       dut.setParam(1, 3);
14       dut.setParam(2, 17);
15       for(idx__put <= 10; idx__put < 20; idx__put <= idx__put + 1) action
16         dut.calc.put(unpack(pack(idx__put)));
17       endaction
18     endseq
19     //.....
```


Embedded Systems & Applications

FSM Testbench - Teil 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Stmt s = seq
2    dut.setParam(0, 42);
3    dut.setParam(1, 2);
4    dut.setParam(2, 13);
5    $display("Initialized parameters, starting test...");
6    par
7      seq
8        for(idx_put <= 0; idx_put < 10; idx_put <= idx_put + 1) action
9          dut.calc.put(unpack(pack(idx_put)));
10         // unpack(pack()) to cast from uint to int
11       endaction
12       dut.setParam(0, 7);
13       dut.setParam(1, 3);
14       dut.setParam(2, 17);
15       for(idx_put <= 10; idx_put < 20; idx_put <= idx_put + 1) action
16         dut.calc.put(unpack(pack(idx_put)));
17       endaction
18     endseq
19     //.....
```


Embedded Systems & Applications

FSM Testbench - Teil 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Stmt s = seq
2    dut.setParam(0, 42);
3    dut.setParam(1, 2);
4    dut.setParam(2, 13);
5    $display("Initialized parameters, starting test...");
6    par
7      seq
8        for(idx__put <= 0; idx__put < 10; idx__put <= idx__put + 1) action
9          dut.calc.put(unpack(pack(idx__put)));
10         // unpack(pack()) to cast from uint to int
11       endaction
12       dut.setParam(0, 7);
13       dut.setParam(1, 3);
14       dut.setParam(2, 17);
15       for(idx__put <= 10; idx__put < 20; idx__put <= idx__put + 1) action
16         dut.calc.put(unpack(pack(idx__put)));
17       endaction
18     endseq
19     //.....
```


Embedded Systems & Applications

FSM Testbench - Teil 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Stmt s = seq
2    dut.setParam(0, 42);
3    dut.setParam(1, 2);
4    dut.setParam(2, 13);
5    $display("Initialized parameters, starting test...");
6    par
7      seq
8        for(idx__put <= 0; idx__put < 10; idx__put <= idx__put + 1) action
9          dut.calc.put(unpack(pack(idx__put)));
10         // unpack(pack()) to cast from uint to int
11       endaction
12       dut.setParam(0, 7);
13       dut.setParam(1, 3);
14       dut.setParam(2, 17);
15       for(idx__put <= 10; idx__put < 20; idx__put <= idx__put + 1) action
16         dut.calc.put(unpack(pack(idx__put)));
17       endaction
18     endseq
19     //.....
```


Embedded Systems & Applications

FSM Testbench - Teil 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Stmt s = seq
2    dut.setParam(0, 42);
3    dut.setParam(1, 2);
4    dut.setParam(2, 13);
5    $display("Initialized parameters, starting test...");
6    par
7      seq
8        for(idx_put <= 0; idx_put < 10; idx_put <= idx_put + 1) action
9          dut.calc.put(unpack(pack(idx_put)));
10         // unpack(pack()) to cast from uint to int
11       endaction
12       dut.setParam(0, 7);
13       dut.setParam(1, 3);
14       dut.setParam(2, 17);
15       for(idx_put <= 10; idx_put < 20; idx_put <= idx_put + 1) action
16         dut.calc.put(unpack(pack(idx_put)));
17       endaction
18     endseq
19     //.....
```


Embedded Systems & Applications

FSM Testbench - Teil 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Stmt s = seq
2    dut.setParam(0, 42);
3    dut.setParam(1, 2);
4    dut.setParam(2, 13);
5    $display("Initialized parameters, starting test...");
6    par
7      seq
8        for(idx__put <= 0; idx__put < 10; idx__put <= idx__put + 1) action
9          dut.calc.put(unpack(pack(idx__put)));
10         // unpack(pack()) to cast from uint to int
11       endaction
12       dut.setParam(0, 7);
13       dut.setParam(1, 3);
14       dut.setParam(2, 17);
15       for(idx__put <= 10; idx__put < 20; idx__put <= idx__put + 1) action
16         dut.calc.put(unpack(pack(idx__put)));
17       endaction
18     endseq
19     //.....
```


Embedded Systems & Applications

FSM Testbench - Teil 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1     //.....
2     seq
3     for(idx_get <= 0; idx_get < 20; idx_get <= idx_get + 1) action
4       $display("Checking result for %d", idx_get);
5       let t <- dut.calc.result();
6       if(t == testvec[idx_get])
7           correct_tests <= correct_tests + 1;
8       else $display("Expected %d, got %d.", testvec[idx_get], t);
9     endaction
10   endseq
11 endpar
12 $display("%d of 20 tests passed.", correct_tests);
13 if(correct_tests == 20) $display("SUCCESS!");
14 else $display("FAILURE!");
15 $finish();
16 endseq;
```

FSM Testbench - Teil 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1     //.....
2     seq
3     for(idx_get <= 0; idx_get < 20; idx_get <= idx_get + 1) action
4       $display("Checking result for %d", idx_get);
5       let t <- dut.calc.result();
6       if(t == testvec[idx_get])
7           correct_tests <= correct_tests + 1;
8       else $display("Expected %d, got %d.", testvec[idx_get], t);
9     endaction
10    endseq
11  endpar
12  $display("%d of 20 tests passed.", correct_tests);
13  if(correct_tests == 20) $display("SUCCESS!");
14  else $display("FAILURE!");
15  $finish();
16  endseq;
```

FSM Testbench - Teil 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1     //.....
2     seq
3     for(idx_get <= 0; idx_get < 20; idx_get <= idx_get + 1) action
4       $display("Checking result for %d", idx_get);
5       let t <- dut.calc.result();
6       if(t == testvec[idx_get])
7           correct_tests <= correct_tests + 1;
8       else $display("Expected %d, got %d.", testvec[idx_get], t);
9     endaction
10   endseq
11 endpar
12 $display("%d of 20 tests passed.", correct_tests);
13 if(correct_tests == 20) $display("SUCCESS!");
14 else $display("FAILURE!");
15 $finish();
16 endseq;
```

FSM Testbench - Instanziierung

FSM Testbench - Instanziierung

TECHNISCHE UNIVERSITÄT DARMSTADT

1. FSM fsm <- mkFSM(stmt) - **fsm.start()**, **fsm.done()**

FSM Testbench - Instanziierung

TECHNISCHE UNIVERSITÄT DARMSTADT

1. FSM fsm <- mkFSM(stmt) - **fsm.start()**, **fsm.done()**
2. mkAutoFSM(stmt) - Direkter Start, läuft einmal

FSM Testbench - Instanziierung

TECHNISCHE UNIVERSITÄT DARMSTADT

1. FSM fsm <- mkFSM(stmt) - **fsm.start()**, **fsm.done()**
2. mkAutoFSM(stmt) - Direkter Start, läuft einmal
3. FSM fsm <- mkFSMWithPred(stmt, bool) - mkFSM, Zustandsänderung nur wenn **bool** == True

FSM Auswertung - s1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s1 = seq
4    $display("%t : %d", $time(), ctr);
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);
7    par
8      seq
9        $display("%t : %d", $time(), ctr);
10     endseq
11     seq
12       action
13         ctr <= ctr + 1;
14         $display("%t : %d", $time(), ctr);
15       endaction
16       $display("%t, %d", $time(), ctr);
17     endseq
18   endpar
19 endseq;
```

FSM Auswertung - s1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s1 = seq
4    $display("%t : %d", $time(), ctr);
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);
7    par
8      seq
9        $display("%t : %d", $time(), ctr);
10     endseq
11     seq
12       action
13         ctr <= ctr + 1;
14         $display("%t : %d", $time(), ctr);
15       endaction
16       $display("%t, %d", $time(), ctr);
17     endseq
18   endpar
19 endseq;
```

0

FSM Auswertung - s1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s1 = seq
4    $display("%t : %d", $time(), ctr);
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);
7    par
8      seq
9        $display("%t : %d", $time(), ctr);
10     endseq
11     seq
12       action
13         ctr <= ctr + 1;
14         $display("%t : %d", $time(), ctr);
15       endaction
16       $display("%t, %d", $time(), ctr);
17     endseq
18   endpar
19 endseq;
```

0

1

FSM Auswertung - s1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s1 = seq
4    $display("%t : %d", $time(), ctr);
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);
7    par
8      seq
9        $display("%t : %d", $time(), ctr);
10     endseq
11     seq
12       action
13         ctr <= ctr + 1;
14         $display("%t : %d", $time(), ctr);
15       endaction
16       $display("%t, %d", $time(), ctr);
17     endseq
18   endpar
19 endseq;
```

0

1

1

FSM Auswertung - s1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s1 = seq
4    $display("%t : %d", $time(), ctr);
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);
7    par
8      seq
9        $display("%t : %d", $time(), ctr);
10     endseq
11     seq
12       action
13         ctr <= ctr + 1;
14         $display("%t : %d", $time(), ctr);
15       endaction
16       $display("%t, %d", $time(), ctr);
17     endseq
18   endpar
19 endseq;
```

0

1

1

1

FSM Auswertung - s1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s1 = seq
4    $display("%t : %d", $time(), ctr);      0
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);      1
7    par
8      seq
9        $display("%t : %d", $time(), ctr);  1
10     endseq
11     seq
12       action
13         ctr <= ctr + 1;
14         $display("%t : %d", $time(), ctr); 1
15       endaction
16       $display("%t, %d", $time(), ctr);   2
17     endseq
18   endpar
19 endseq;
```

FSM Auswertung - s2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s2 = seq
4    $display("%t : %d", $time(), ctr);
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);
7    par
8      seq
9        $display("%t : %d", $time(), ctr);
10     endseq
11     seq
12       ctr <= ctr + 1;
13       $display("%t : %d", $time(), ctr);
14       $display("%t, %d", $time(), ctr);
15     endseq
16   endpar
17 endseq;
```

FSM Auswertung - s2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s2 = seq
4    $display("%t : %d", $time(), ctr);
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);
7    par
8      seq
9        $display("%t : %d", $time(), ctr);
10     endseq
11     seq
12       ctr <= ctr + 1;
13       $display("%t : %d", $time(), ctr);
14       $display("%t, %d", $time(), ctr);
15     endseq
16   endpar
17 endseq;
```

0

FSM Auswertung - s2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s2 = seq
4    $display("%t : %d", $time(), ctr);
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);
7    par
8      seq
9        $display("%t : %d", $time(), ctr);
10     endseq
11     seq
12       ctr <= ctr + 1;
13       $display("%t : %d", $time(), ctr);
14       $display("%t, %d", $time(), ctr);
15     endseq
16   endpar
17 endseq;
```

0

1

FSM Auswertung - s2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s2 = seq
4    $display("%t : %d", $time(), ctr);
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);
7    par
8      seq
9        $display("%t : %d", $time(), ctr);
10     endseq
11     seq
12       ctr <= ctr + 1;
13       $display("%t : %d", $time(), ctr);
14       $display("%t, %d", $time(), ctr);
15     endseq
16   endpar
17 endseq;
```

0

1

1

FSM Auswertung - s2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s2 = seq
4    $display("%t : %d", $time(), ctr);
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);
7    par
8      seq
9        $display("%t : %d", $time(), ctr);
10     endseq
11     seq
12       ctr <= ctr + 1;
13       $display("%t : %d", $time(), ctr);
14       $display("%t, %d", $time(), ctr);
15     endseq
16   endpar
17 endseq;
```

0

1

1

2

FSM Auswertung - s2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(int) ctr <- mkReg(0);
2
3  Stmt s2 = seq
4    $display("%t : %d", $time(), ctr);   0
5    ctr <= ctr + 1;
6    $display("%t : %d", $time(), ctr);   1
7    par
8      seq
9        $display("%t : %d", $time(), ctr);   1
10     endseq
11     seq
12       ctr <= ctr + 1;
13       $display("%t : %d", $time(), ctr);   2
14       $display("%t, %d", $time(), ctr);   2
15     endseq
16   endpar
17 endseq;
```

FSM Auswertung - s3

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Stmt s3 = seq
2    $display("%t : %d", $time(), ctr);
3    ctr <= ctr + 1;
4    $display("%t : %d", $time(), ctr);
5    par
6      seq
7        $display("%t : %d", $time(), ctr);
8        action
9          ctr <= ctr + 1;
10         ctr <= ctr + 1;
11       endaction
12       $display("%t : %d", $time(), ctr);
13     endseq
14     seq
15       ctr <= ctr + 1;
16       $display("%t : %d", $time(), ctr);
17       $display("%t, %d", $time(), ctr);
18     endseq
19   endpar
20 endseq;
```

FSM Auswertung - s3

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Stmt s3 = seq
2    $display("%t : %d", $time(), ctr);
3    ctr <= ctr + 1;
4    $display("%t : %d", $time(), ctr);
5    par
6      seq
7        $display("%t : %d", $time(), ctr);
8        action
9          ctr <= ctr + 1;
10         ctr <= ctr + 1;
11       endaction
12       $display("%t : %d", $time(), ctr);
13     endseq
14     seq
15       ctr <= ctr + 1;
16       $display("%t : %d", $time(), ctr);
17       $display("%t, %d", $time(), ctr);
18     endseq
19   endpar
20 endseq;
```

ER-
ROR

FSMs - Compilerfehler/-Warnungen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Warning: "ALUTest.bsv", line 96, column 10: (G0117)
2  Rule 'checkFSM_action_l110c9' shadows the effects of 'uut_pow_calc' when
3  they execute in the same clock cycle. Affected method calls:
4    uut_pow_opB.write, uut_pow_result.write
5  To silence this warning, use the '-no-warn-action-shadowing' flag.
```

TECHNISCHE UNIVERSITÄT DARMSTADT

Fragen zu StmtFSMs?

Subinterfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motivation:

Subinterfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motivation:
  - Wiederverwendbarkeit von häufigem I/O-Verhalten

Subinterfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motivation:
  - Wiederverwendbarkeit von häufigem I/O-Verhalten
  - Redundanzvermeidung

Subinterfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motivation:
  - Wiederverwendbarkeit von häufigem I/O-Verhalten
  - Redundanzvermeidung
  - Code-Strukturierung

Subinterfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motivation:
  - Wiederverwendbarkeit von häufigem I/O-Verhalten
  - Redundanzvermeidung
  - Code-Strukturierung

  $\Rightarrow$ Schnellerer Designentwurf

Modulare Pipeline - Interface Deklarationen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 interface CalcUnit;
2   method Action put(Int#(32) x);
3   method ActionValue#(Int#(32)) result();
4 endinterface
```

Modulare Pipeline - Interface Deklarationen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 interface CalcUnit;
2   method Action put(Int#(32) x);
3   method ActionValue#(Int#(32)) result();
4 endinterface
5 interface CalcUnitChangeable;
6   interface CalcUnit calc;
7   method Action setParameter(Int#(32) param);
8 endinterface
```

Pipeline Aufbau

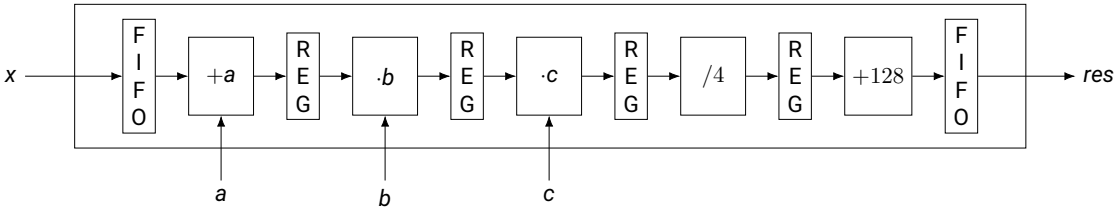


Abbildung: Aufbau der Pipeline

Modulare Pipeline - Implementierung CalcUnitChangeable

Modul mkAddA

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  module mkAddA(CalcUnitChangeable);
2    Reg#(Int#(32)) a <- mkRegU;
3    Wire#(Int#(32)) in <- mkWire;
4
5    method Action setParameter(Int#(32) param);
6      a <= param;
7    endmethod
8
9    interface CalcUnit calc;
10     method Action put(Int#(32) x);
11       in <= x;
12     endmethod
13
14     method ActionValue#(Int#(32)) result();
15       noAction;
16       return in + a;
17     endmethod
18   endinterface
19 endmodule
```

Modulare Pipeline - Implementierung CalcUnitChangeable

Modul mkAddA

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  module mkAddA(CalcUnitChangeable);
2    Reg#(Int#(32)) a <- mkRegU;
3    Wire#(Int#(32)) in <- mkWire;
4
5    method Action setParameter(Int#(32) param);
6      a <= param;
7    endmethod
8
9    interface CalcUnit calc;
10     method Action put(Int#(32) x);
11       in <= x;
12     endmethod
13
14     method ActionValue#(Int#(32)) result();
15       noAction;
16       return in + a;
17     endmethod
18   endinterface
19 endmodule
```

Modulare Pipeline - Implementierung CalcUnitChangeable

Modul mkAddA

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  module mkAddA(CalcUnitChangeable);
2    Reg#(Int#(32)) a <- mkRegU;
3    Wire#(Int#(32)) in <- mkWire;
4
5    method Action setParameter(Int#(32) param);
6      a <= param;
7    endmethod
8
9    interface CalcUnit calc;
10     method Action put(Int#(32) x);
11       in <= x;
12     endmethod
13
14     method ActionValue#(Int#(32)) result();
15       noAction;
16       return in + a;
17     endmethod
18   endinterface
19 endmodule
```

Modulare Pipeline - Implementierung CalcUnitChangeable

Modul mkAddA

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  module mkAddA(CalcUnitChangeable);
2    Reg#(Int#(32)) a <- mkRegU;
3    Wire#(Int#(32)) in <- mkWire;
4
5    method Action setParameter(Int#(32) param);
6      a <= param;
7    endmethod
8
9    interface CalcUnit calc;
10     method Action put(Int#(32) x);
11       in <= x;
12     endmethod
13
14     method ActionValue#(Int#(32)) result();
15       noAction;
16       return in + a;
17     endmethod
18   endinterface
19 endmodule
```

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  CalcUnitChangeable stage1 <- mkAddA;
2  CalcUnitChangeable stage2 <- mkMul;
3  CalcUnitChangeable stage3 <- mkMul;
4  CalcUnit stage4 <- mkDiv4;
5  CalcUnit stage5 <- mkAdd128;
6
7  Vector#(3, CalcUnitChangeable) changeables;
8  changeables[0] = stage1;
9  changeables[1] = stage2;
10 changeables[2] = stage3;
11
12 Vector#(5, CalcUnit) stages;
13 stages[0] = stage1.calc;
14 stages[1] = stage2.calc;
15 stages[2] = stage3.calc;
16 stages[3] = stage4;
17 stages[4] = stage5;
18
19 FIFOF#(Int#(32)) fifo_in <- mkFIFOF;
20 FIFO#(Int#(32)) fifo_out <- mkFIFO;
21 Vector#(4, Array#(Reg#(Maybe#(Int#(32))))) regs <- replicateM(mkCReg(2, tagged Invalid));
```


Embedded Systems & Applications

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  CalcUnitChangeable stage1 <- mkAddA;
2  CalcUnitChangeable stage2 <- mkMul;
3  CalcUnitChangeable stage3 <- mkMul;
4  CalcUnit stage4 <- mkDiv4;
5  CalcUnit stage5 <- mkAdd128;
6
7  Vector#(3, CalcUnitChangeable) changeables;
8  changeables[0] = stage1;
9  changeables[1] = stage2;
10 changeables[2] = stage3;
11
12 Vector#(5, CalcUnit) stages;
13 stages[0] = stage1.calc;
14 stages[1] = stage2.calc;
15 stages[2] = stage3.calc;
16 stages[3] = stage4;
17 stages[4] = stage5;
18
19 FIFOF#(Int#(32)) fifo_in <- mkFIFOF;
20 FIFO#(Int#(32)) fifo_out <- mkFIFO;
21 Vector#(4, Array#(Reg#(Maybe#(Int#(32))))) regs <- replicateM(mkCReg(2, tagged Invalid));
```

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  CalcUnitChangeable stage1 <- mkAddA;
2  CalcUnitChangeable stage2 <- mkMul;
3  CalcUnitChangeable stage3 <- mkMul;
4  CalcUnit stage4 <- mkDiv4;
5  CalcUnit stage5 <- mkAdd128;
6
7  Vector#(3, CalcUnitChangeable) changeables;
8  changeables[0] = stage1;
9  changeables[1] = stage2;
10 changeables[2] = stage3;
11
12 Vector#(5, CalcUnit) stages;
13 stages[0] = stage1.calc;
14 stages[1] = stage2.calc;
15 stages[2] = stage3.calc;
16 stages[3] = stage4;
17 stages[4] = stage5;
18
19 FIFOF#(Int#(32)) fifo_in <- mkFIFOF;
20 FIFO#(Int#(32)) fifo_out <- mkFIFO;
21 Vector#(4, Array#(Reg#(Maybe#(Int#(32))))) regs <- replicateM(mkCReg(2, tagged Invalid));
```


Embedded Systems & Applications

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  CalcUnitChangeable stage1 <- mkAddA;
2  CalcUnitChangeable stage2 <- mkMul;
3  CalcUnitChangeable stage3 <- mkMul;
4  CalcUnit stage4 <- mkDiv4;
5  CalcUnit stage5 <- mkAdd128;
6
7  Vector#(3, CalcUnitChangeable) changeables;
8  changeables[0] = stage1;
9  changeables[1] = stage2;
10 changeables[2] = stage3;
11
12 Vector#(5, CalcUnit) stages;
13 stages[0] = stage1.calc;
14 stages[1] = stage2.calc;
15 stages[2] = stage3.calc;
16 stages[3] = stage4;
17 stages[4] = stage5;
18
19 FIFOF#(Int#(32)) fifo_in <- mkFIFOF;
20 FIFO#(Int#(32)) fifo_out <- mkFIFO;
21 Vector#(4, Array#(Reg#(Maybe#(Int#(32))))) regs <- replicateM(mkCReg(2, tagged Invalid));
```


Embedded Systems & Applications

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  CalcUnitChangeable stage1 <- mkAddA;
2  CalcUnitChangeable stage2 <- mkMul;
3  CalcUnitChangeable stage3 <- mkMul;
4  CalcUnit stage4 <- mkDiv4;
5  CalcUnit stage5 <- mkAdd128;
6
7  Vector#(3, CalcUnitChangeable) changeables;
8  changeables[0] = stage1;
9  changeables[1] = stage2;
10 changeables[2] = stage3;
11
12 Vector#(5, CalcUnit) stages;
13 stages[0] = stage1.calc;
14 stages[1] = stage2.calc;
15 stages[2] = stage3.calc;
16 stages[3] = stage4;
17 stages[4] = stage5;
18
19 FIFOF#(Int#(32)) fifo_in <- mkFIFOF;
20 FIFO#(Int#(32)) fifo_out <- mkFIFO;
21 Vector#(4, Array#(Reg#(Maybe#(Int#(32))))) regs <- replicateM(mkCReg(2, tagged Invalid));
```

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  CalcUnitChangeable stage1 <- mkAddA;
2  CalcUnitChangeable stage2 <- mkMul;
3  CalcUnitChangeable stage3 <- mkMul;
4  CalcUnit stage4 <- mkDiv4;
5  CalcUnit stage5 <- mkAdd128;
6
7  Vector#(3, CalcUnitChangeable) changeables;
8  changeables[0] = stage1;
9  changeables[1] = stage2;
10 changeables[2] = stage3;
11
12 Vector#(5, CalcUnit) stages;
13 stages[0] = stage1.calc;
14 stages[1] = stage2.calc;
15 stages[2] = stage3.calc;
16 stages[3] = stage4;
17 stages[4] = stage5;
18
19 FIFOF#(Int#(32)) fifo_in <- mkFIFOF;
20 FIFO#(Int#(32)) fifo_out <- mkFIFO;
21 Vector#(4, Array#(Reg#(Maybe#(Int#(32))))) regs <- replicateM(mkCReg(2, tagged Invalid));
```

Pipeline Aufbau

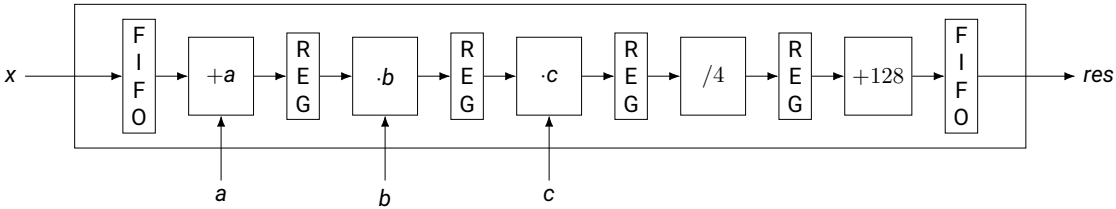


Abbildung: Aufbau der Pipeline

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

Regeln:

```
1  for(Integer i = 0; i < 5; i = i + 1) begin
2      rule push;
3          if (i == 0)
4              stages[i].put(fifo_in.first);
5          else if (regs[i-1][0] matches tagged Valid .x)
6              stages[i].put(x);
7      endrule
8  end
9
10 for(Integer i = 1; i < 4; i = i + 1) begin
11     rule pull if (regs[i][1] matches tagged Invalid &&& isValid(regs[i-1][0]));
12         regs[i-1][0] <= tagged Invalid;
13         let t <- stages[i].result();
14         regs[i][1] <= tagged Valid t;
15     endrule
16 end
```

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

Regeln:

```
1  for(Integer i = 0; i < 5; i = i + 1) begin
2      rule push;
3          if (i == 0)
4              stages[i].put(fifo_in.first);
5          else if (regs[i-1][0] matches tagged Valid .x)
6              stages[i].put(x);
7      endrule
8  end
9
10 for(Integer i = 1; i < 4; i = i + 1) begin
11     rule pull if (regs[i][1] matches tagged Invalid &&& isValid(regs[i-1][0]));
12         regs[i-1][0] <= tagged Invalid;
13         let t <- stages[i].result();
14         regs[i][1] <= tagged Valid t;
15     endrule
16 end
```

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

Regeln:

```
1  for(Integer i = 0; i < 5; i = i + 1) begin
2      rule push;
3          if (i == 0)
4              stages[i].put(fifo_in.first);
5          else if (regs[i-1][0] matches tagged Valid .x)
6              stages[i].put(x);
7      endrule
8  end
9
10 for(Integer i = 1; i < 4; i = i + 1) begin
11     rule pull if (regs[i][1] matches tagged Invalid &&& isValid(regs[i-1][0]));
12         regs[i-1][0] <= tagged Invalid;
13         let t <- stages[i].result();
14         regs[i][1] <= tagged Valid t;
15     endrule
16 end
```

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

Regeln:

```
1  for(Integer i = 0; i < 5; i = i + 1) begin
2      rule push;
3          if (i == 0)
4              stages[i].put(fifo_in.first);
5          else if (regs[i-1][0] matches tagged Valid .x)
6              stages[i].put(x);
7      endrule
8  end
9
10 for(Integer i = 1; i < 4; i = i + 1) begin
11     rule pull if (regs[i][1] matches tagged Invalid &&& isValid(regs[i-1][0]));
12         regs[i-1][0] <= tagged Invalid;
13         let t <- stages[i].result();
14         regs[i][1] <= tagged Valid t;
15     endrule
16 end
```

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

Regeln:

```
1  for(Integer i = 0; i < 5; i = i + 1) begin
2      rule push;
3          if (i == 0)
4              stages[i].put(fifo_in.first);
5          else if (regs[i-1][0] matches tagged Valid .x)
6              stages[i].put(x);
7      endrule
8  end
9
10 for(Integer i = 1; i < 4; i = i + 1) begin
11     rule pull if (regs[i][1] matches tagged Invalid &&& isValid(regs[i-1][0]));
12         regs[i-1][0] <= tagged Invalid;
13         let t <- stages[i].result();
14         regs[i][1] <= tagged Valid t;
15     endrule
16 end
```

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

Regeln:

```
1  for(Integer i = 0; i < 5; i = i + 1) begin
2      rule push;
3          if (i == 0)
4              stages[i].put(fifo_in.first);
5          else if (regs[i-1][0] matches tagged Valid .x)
6              stages[i].put(x);
7      endrule
8  end
9
10 for(Integer i = 1; i < 4; i = i + 1) begin
11     rule pull if (regs[i][1] matches tagged Invalid &&& isValid(regs[i-1][0]));
12         regs[i-1][0] <= tagged Invalid;
13         let t <- stages[i].result();
14         regs[i][1] <= tagged Valid t;
15     endrule
16 end
```

Modulare Pipeline - Pull und CReg

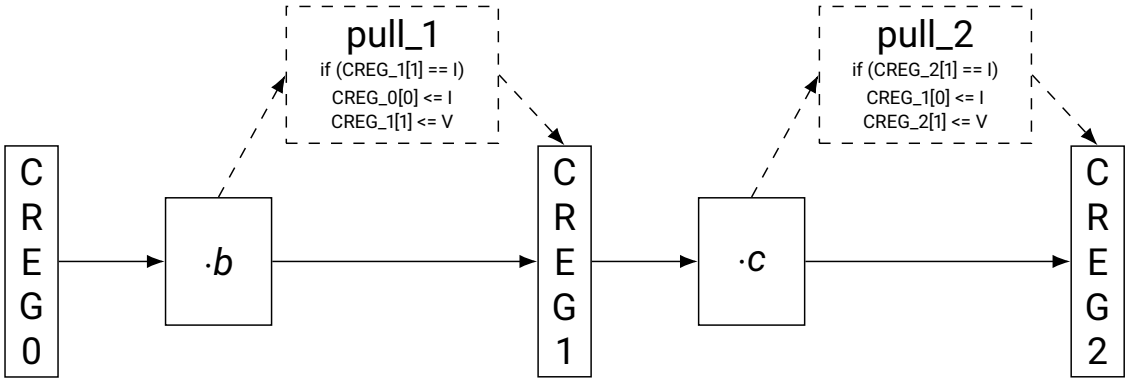


Abbildung: Pipeline Ausschnitt

Modulare Pipeline - Implementierung

Modul mkPipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

Interface Definition:

```
1  method Action setParam(UInt#(2) addr, Int#(32) val) if(setParamAllowed);
2      changeables[addr].setParameter(val);
3  endmethod
4
5  interface CalcUnit calc;
6      method Action put(Int#(32) x);
7          fifo_in.enq(x);
8      endmethod
9
10     method ActionValue#(Int#(32)) result();
11         fifo_out.deq();
12         return fifo_out.first;
13     endmethod
14 endinterface
```

Modulare Pipeline - Charakterisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Pipeline ist **statisch**

Modulare Pipeline - Charakterisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Pipeline ist **statisch**
  - Die Latenz ist immer gleich (datenunabhängig)

Modulare Pipeline - Charakterisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Pipeline ist **statisch**
  - Die Latenz ist immer gleich (datenunabhängig)
- Die Pipeline ist **elastisch**

Modulare Pipeline - Charakterisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Pipeline ist **statisch**
  - Die Latenz ist immer gleich (datenunabhängig)
- Die Pipeline ist **elastisch**
  - Die Daten können sich unabhängig voneinander durch die Pipeline bewegen

Modulare Pipeline - Charakterisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Pipeline ist **statisch**
  - Die Latenz ist immer gleich (datenunabhängig)
- Die Pipeline ist **elastisch**
  - Die Daten können sich unabhängig voneinander durch die Pipeline bewegen
  - Implementierung der Stufen in separaten Regeln

Modulare Pipeline - Charakterisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Pipeline ist **statisch**
  - Die Latenz ist immer gleich (datenunabhängig)
- Die Pipeline ist **elastisch**
  - Die Daten können sich unabhängig voneinander durch die Pipeline bewegen
  - Implementierung der Stufen in separaten Regeln
- Die Pipeline unterstützt CalcUnits mit variabler Latenz
  - Die Pipeline ist dann **dynamisch** und **elastisch**

TECHNISCHE UNIVERSITÄT DARMSTADT

Fragen zu Pipelines/Subinterfaces?

Tagged Unions - Motivation

Tagged Unions - Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bündelung verschiedener Typen in einem Obertyp

Tagged Unions - Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bündelung verschiedener Typen in einem Obertyp
- Im Gegensatz zu Structs nur ein Typ auf einmal aktiv

Tagged Unions - Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bündelung verschiedener Typen in einem Obertyp
- Im Gegensatz zu Structs nur ein Typ auf einmal aktiv
- Unterscheidung Anhand von Tag Bits

Beispiele:

Tagged Unions - Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bündelung verschiedener Typen in einem Obertyp
- Im Gegensatz zu Structs nur ein Typ auf einmal aktiv
- Unterscheidung Anhand von Tag Bits

Beispiele:

- Maybe - Valid/Invalid

Tagged Unions - Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bündelung verschiedener Typen in einem Obertyp
- Im Gegensatz zu Structs nur ein Typ auf einmal aktiv
- Unterscheidung Anhand von Tag Bits

Beispiele:

- Maybe - Valid/Invalid
- Prozessor Instruktionen - R-Typ/I-Typ/J-Typ

Tagged Unions - Übungsbeispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 typedef union tagged {
2   UInt#(32) Unsigned;
3   Int#(32)  Signed;
4 } SignedOrUnsigned deriving(Eq, Bits);
```

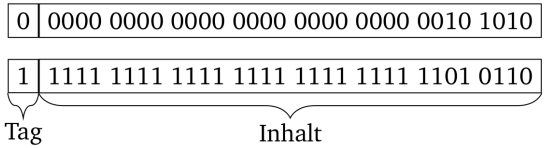


Abbildung: Bit Representation von SignedOrUnsigned

Tagged Unions - Übungsbeispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 typedef union tagged {
2   UInt#(32) Unsigned;
3   Int#(32)  Signed;
4 } SignedOrUnsigned deriving(Eq, Bits);
```

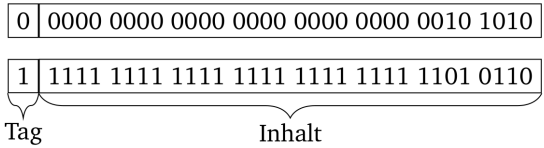


Abbildung: Bit Representation von SignedOrUnsigned

Tagged Unions - Übungsbeispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 typedef union tagged {
2   UInt#(32) Unsigned;
3   Int#(32)  Signed;
4 } SignedOrUnsigned deriving(Eq, Bits);
```

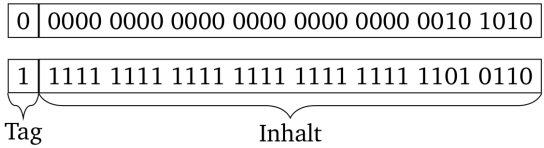


Abbildung: Bit Representation von SignedOrUnsigned

Tagged Unions - Übungsbeispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 typedef union tagged {
2   UInt#(32) Unsigned;
3   Int#(32)  Signed;
4 } SignedOrUnsigned deriving(Eq, Bits);
```

==/!=

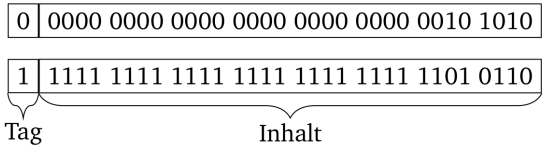


Abbildung: Bit Representation von SignedOrUnsigned

Tagged Unions - Übungsbeispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 typedef union tagged {
2   UInt#(32) Unsigned;
3   Int#(32)  Signed;
4 } SignedOrUnsigned deriving(Eq, Bits);
```

==/!=

pack/unpack

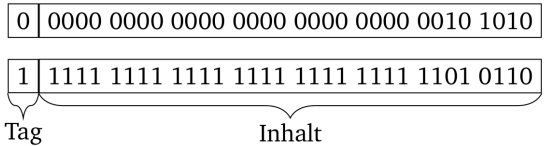


Abbildung: Bit Representation von SignedOrUnsigned

Tagged Unions - Interface Änderungen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface TU_Pipeline;
2    interface CalcUnit calc;
3    method Action setParam(UInt#(2) addr, SignedOrUnsigned val);
4  endinterface
5
6  // In Datei CalcUnits.bsv
7  interface CalcUnit;
8    method Action put(SignedOrUnsigned x);
9    method ActionValue#(SignedOrUnsigned) result();
10 endinterface
11
12 interface CalcUnitChangeable;
13   interface CalcUnit calc;
14   method Action setParameter(SignedOrUnsigned param);
15 endinterface
```

Tagged Unions - CalcUnitChangeable

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  module mkAddA(CalcUnitChangeable);
2    Reg#(SignedOrUnsigned) a <- mkRegU;
3    Wire#(SignedOrUnsigned) in <- mkWire;
4    Wire#(SignedOrUnsigned) out <- mkWire;
5
6    /* RULES AUSGELASSEN */
7
8    method Action setParameter(SignedOrUnsigned param);
9        a <= param;
10   endmethod
11
12   interface CalcUnit calc;
13     method Action put(SignedOrUnsigned x);
14         in <= x;
15     endmethod
16
17     method ActionValue#(SignedOrUnsigned) result();
18         noAction;
19         return out;
20     endmethod
21   endinterface
```

Tagged Unions - CalcUnitChangeable Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  /* Untermodule ausgelassen */
2
3  /* RULES */
4  rule computeSigned(a matches tagged Signed .sa &&&
5                     in matches tagged Signed .sin);
6      out <= tagged Signed (sa + sin);
7  endrule
8
9  rule computeUnsigned(a matches tagged Unsigned .usa &&&
10                      in matches tagged Unsigned .usin);
11     out <= tagged Unsigned (usa + usin);
12 endrule
13
14 rule crash(pack(in)[32] != pack(a)[32]); // if we input an unfitting value we crash
15     $display("ERROR: mkAddA got wrong input!");
16     $display("Tag a: %b, Tag in: %b", pack(a)[32], pack(in)[32]);
17     $finish();
18 endrule
19
20 /* Interface Definition ausgelassen */
```

Fragerunde

TECHNISCHE UNIVERSITÄT DARMSTADT

Fragen zur Vorlesung oder zur Übung?