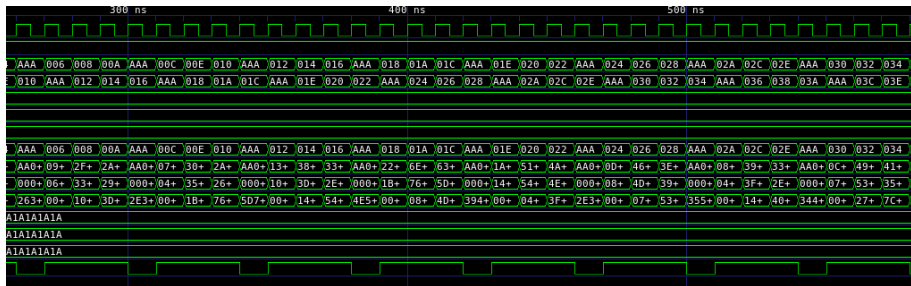


Architekturen und Entwurf von Rechnersystemen

Wintersemester 2018/19



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Dies und das



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Hörsaalübungstermine im Moodle

- ▶ Hörsaalübungstermine im Moodle
- ▶ Theorieübungsblatt Nummer 1 im Moodle
 - ▶ Einige Fragen zu Bluespec
 - ▶ z.B. Scheduling oder häufig verwendete Komponenten

- ▶ Hörsaalübungstermine im Moodle
- ▶ Theorieübungsblatt Nummer 1 im Moodle
 - ▶ Einige Fragen zu Bluespec
 - ▶ z.B. Scheduling oder häufig verwendete Komponenten
- ▶ Sprechstunden zur Vertiefung nutzen



- ▶ Hörsaalübungstermine im Moodle
- ▶ Theorieübungsblatt Nummer 1 im Moodle
 - ▶ Einige Fragen zu Bluespec
 - ▶ z.B. Scheduling oder häufig verwendete Komponenten
- ▶ Sprechstunden zur Vertiefung nutzen
- ▶ Eine Woche vor der Klausur mit Bluespec anzufangen reicht (meist) nicht...



- ▶ Vertiefung der Inhalte aus Hörsaalübung 1
- ▶ Verschiedene Einsatzzwecke von FSM
- ▶ Tagged Unions im Einsatz
- ▶ Neues Thema: Nested Interfaces



Aufgabe 2.1.1



- ▶ Eine FSM erstellen die nach 100 Taktzyklen eine Nachricht ausgibt:
 - ▶ Vertraut werden mit der Syntax von StmtFSM
 - ▶ Eigenschaften der Ausführungszeit betrachten

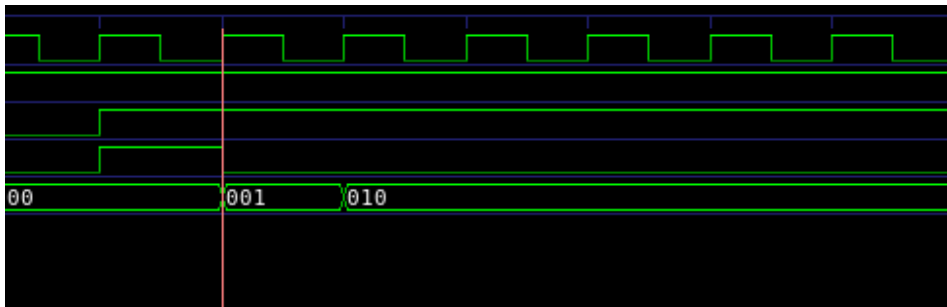


```
1  package FSMTests;
2
3  import StmtFSM :: *;
4  module mkFirstFSM(Empty);
5      Stmt firstStmt = {
6          seq
7              delay(100);
8              action
9                  $display("(%0d) Hello World!", $time);
10             endaction
11         endseq
12     };
13     mkAutoFSM(firstStmt);
14 endmodule
15 endpackage
```

Eine erste FSM

```
1 > ./out
2 (1020) Hello World!
```

▶ 10 ns Taktlänge → 102 Takte





Aufgabe 2.1.2

- ▶ StmtFSM ermöglicht nicht nur sequentielle Ausführung
 - ▶ `par`-Umgebung
- ▶ Alle Anweisungen in einer `par` Umgebung laufen parallel ab
- ▶ FSM läuft erst weiter wenn alle Teile der Umgebung fertig sind
- ▶ Erstellen Sie eine FSM mit paralleler Ausführung:
 1. Zähle 100 Taktzyklen und setze Synchronisationsregister
 2. Gib eine Nachricht 10 mal aus und wartet dann auf Synchronisationsregister



```
1  module mkSecondFSM(Empty);
2  Reg#(Bool) syncVar <- mkReg(False);
3  Stmt secondStmt = {
4      seq
5      par
6          seq
7              $display("(%0d) Part one starts.", $time);
8              delay(100);
9              syncVar <= True;
10             $display("(%0d) Part one done.", $time);
11         endseq
12         seq
13             repeat(10) $display("(%0d) Print this 10 times.", $time);
14             await(syncVar);
15             $display("(%0d) Part two done.", $time);
16         endseq
17     endpar
18     $display("(%0d) Everything is done.", $time);
19 endseq
20 };
21 mkAutoFSM(secondStmt);
22 endmodule
```



```
1  # (15) Part one starts.
2  # (15) Print this 10 times.
3  # (25) Print this 10 times.
4  # (35) Print this 10 times.
5  # (45) Print this 10 times.
6  # (55) Print this 10 times.
7  # (65) Print this 10 times.
8  # (75) Print this 10 times.
9  # (85) Print this 10 times.
10 # (95) Print this 10 times.
11 # (105) Print this 10 times.
12 # (1035) Part one done.
13 # (1045) Part two done.
14 # (1055) Everything is done.
```

- ▶ par Block wartet bis alle Teile fertig sind.



Aufgabe 2.1.3



- ▶ Häufig ist der Systemtakt zu schnell für bestimmte Aufgaben
- ▶ Ausführung der FSM kann gesteuert werden: Zusätzliche Guard
- ▶ Aufgabe:
 1. `mkFSMWithPred`: FSM mit $\frac{1}{100}$ des Systemtakts ansteuern
 2. `PulseWire` zur Synchronisation nutzen



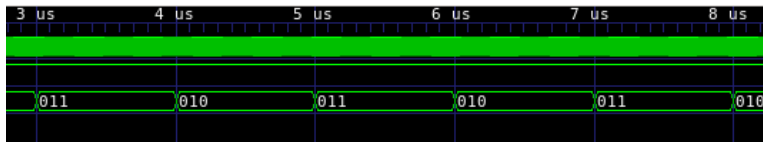
```
1  module mkThirdFSM(Empty);
2      Reg#(UInt#(12)) counter <- mkReg(0);
3      PulseWire pw <- mkPulseWire();
4      Reg#(UInt#(12)) i <- mkReg(0);
5      // Rules....
6      //////////////////////////////////////////////////
7      Stmt thirdStmt = {
8          seq
9              for(i <= 0; i < 20; i <= i + 1) seq
10                 $display("(%0d) Iteration %d.", $time, i);
11             endseq
12             $finish();
13         endseq
14     };
15     FSM myFSM <- mkFSMWithPred(thirdStmt, pw);
16     rule startFSM (myFSM.done());
17     myFSM.start();
18     endrule
19 endmodule
```



```
1  rule count (counter < 99);  
2    counter <= counter + 1;  
3  endrule  
4  
5  rule resetCount (counter == 99);  
6    counter <= 0;  
7    pw.send();  
8  endrule
```

```
1 # (1995) Iteration 0.
2 # (3995) Iteration 1.
3 # (5995) Iteration 2.
4 # (7995) Iteration 3.
5 # (9995) Iteration 4.
6 . . . . .
```

- ▶ Jede Iteration braucht zwei Zyklen. Warum?
- ▶ Waveform ansehen.





Aufgabe 2.1.5



- ▶ FSM hervorragend für Testbenches geeignet
- ▶ StmtFSM bietet einige Hilfen dafür
- ▶ Häufig genutzte Teile können ausgelagert werden



```
1  import Vector::*;
2
3  typedef struct {
4      Int#(32) opA;
5      Int#(32) opB;
6      AluOps   operator;
7      Int#(32) expectedResult;
8  } TestData deriving (Eq, Bits);
9
10
11 module mkAluFSMTB (Empty);
12     Vector#(5, TestData) myVector;
13     myVector[0] = TestData {opA: 2, opB: 4, operator: Add, expectedResult: 6};
14     myVector[1] = TestData {opA: 2, opB: 4, operator: Mul, expectedResult: 8};
15     myVector[2] = TestData {opA: 4, opB: 2, operator: Div, expectedResult: 2};
16     myVector[3] = TestData {opA: 4, opB: 0, operator: Pow, expectedResult: 1};
17     myVector[4] = TestData {opA: 4, opB: 4, operator: Pow, expectedResult: 256};
18
19     Reg#(UInt#(32)) dataPtr <- mkReg(0);
20
21     HelloALU uut <- mkHelloALU();
22     ... STMT Declarations
23 endmodule
```



```
1  Stmt checkStmt = { seq
2      action
3          let currentData = myVector[dataPtr];
4          uut.setupCalculation(currentData.operator, currentData.opA,
5                               currentData.opB);
6      endaction
7      action
8          let currentData = myVector[dataPtr];
9          let result <- uut.getResult();
10         let print = $format("Calculation: %d ", currentData.opA)
11             + fshow(currentData.operator) + $format("%d", currentData.opB);
12         $display(print);
13         if(result == currentData.expectedResult) begin
14             $display("Result correct: %d", result);
15         end else begin
16             $display("Result incorrect: %d != ", result,
17                     currentData.expectedResult);
18         end
19     endaction
20 endseq
21 };
```



```
1  FSM checkFSM <- mkFSM(checkStmt);
2  Stmt mainFSM = {
3      seq
4          for(dataPtr <= 0; dataPtr < 5; dataPtr <= dataPtr + 1) seq
5              checkFSM.start();
6              checkFSM.waitTillDone();
7          endseq
8      endseq
9  };
10 mkAutoFSM(mainFSM);
```



```
1  # Calculation:           2 Add           4
2  # Result correct:       6
3  # Calculation:           2 Mul           4
4  # Result correct:       8
5  # Calculation:           4 Div           2
6  # Result correct:       2
7  # Calculation:           4 Pow           0
8  # Result correct:       1
9  # Calculation:           8 Div           4
10 # Result incorrect:     2 !=           42
```



Aufgabe 2.2.1



▶ ALU Kompatibel mit `UInt` und `Int` machen.

▶ Einsatz von Tagged Union

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

Flexible ALU: Power generisch machen



```
1  interface Power#(type t);
2      method Action  setOperands(t a, t b);
3      method t  getResult();
4  endinterface
5
6  module mkPower(Power#(t))
7      provisos(Bits#(t, t_sz),
8              Ord#(t),
9              Arith#(t),
10             Eq#(t));
11     Reg#(Bool) resultValid <- mkReg(False);
12
13     Reg#(t) opA    <- mkReg(0);
14     Reg#(t) opB    <- mkReg(0);
15     Reg#(t) result <- mkReg(1);
16     ...
```

Flexible ALU: Welche Provisos sind nötig?



```
1 The following additional provisos are needed:
2   Eq#(t)
3     Introduced at the following locations:
4   "Alu.bsv", line 26, column 28
5   Ord#(t)
6     Introduced at the following locations:
7   "Alu.bsv", line 21, column 24
8   Bits#(t, a__)
9     Introduced at the following locations:
10  "Alu.bsv", line 19, column 27
11  "Alu.bsv", line 18, column 27
12  "Alu.bsv", line 17, column 27
13  Arith#(t)
14    Introduced at the following locations:
15  "Alu.bsv", line 23, column 30
16  "Alu.bsv", line 22, column 24
```



```
1 interface HelloALU;
2     method Action setupCalculation(AluOps op, SignedOrUnsigned a,
3                                     SignedOrUnsigned b);
4     method ActionValue#(SignedOrUnsigned) getResult();
5 endinterface
```



```
1  module mkHelloALU(HelloALU);
2      ...
3      Reg#(SignedOrUnsigned) opA    <- mkReg(tagged Signed 0);
4      Reg#(SignedOrUnsigned) opB    <- mkReg(tagged Signed 0);
5      Reg#(SignedOrUnsigned) result <- mkReg(tagged Signed 0);
6      Power#(UInt#(32)) powUInt <- mkPower();
7      Power#(Int#(32))  powInt  <- mkPower();
8      rule calculateSigned (opA matches tagged Signed .va
9                          &&&& opB matches tagged Signed .vb
10                         &&&& newOperands);
11          Int#(32) rTmp = 0;
12          case(operation)
13              Mul: rTmp = va * vb;
14          ....
15      endrule
16      rule calculateUnsigned (opA matches tagged Unsigned .va
17                             &&&& opB matches tagged Unsigned .vb
18                             &&&& newOperands);
19          UInt#(32) rTmp = 0;
20          case(operation)
21              Mul: rTmp = va * vb;
22          ...
```


Flexible ALU: Ungültige Eingaben verwerfen



```
1  function Bool isUnsigned(SignedOrUnsigned v);
2      if(v matches tagged Unsigned .va) return True;
3      else return False;
4  endfunction
5
6  rule dumpInvalid (newOperands && isUnsigned(opA) != isUnsigned(opB));
7      $display("Invalid combination of Signed and Unsigned Operands");
8      newOperands <= False;
9      resultValid <= False;
10 endrule
```

Aufgabe 2.2.2



- ▶ Maybe bereits aus der Vorlesung bekannt
- ▶ Entweder `Invalid` oder `Valid t`
- ▶ Wird hier genutzt um einen Zähler zu Implementieren
- ▶ Zähler hat `incr` und `decr` Methoden → Im gleichen Takt
- ▶ Dafür `RWire` benutzen (bzw. `CReg`)

```
1  interface RWire#(type element_type) ;
2      method Action wset(element_type datain) ;
3      method Maybe#(element_type) wget() ;
4  endinterface: RWire
```



```
1  module mkSimpleCounter(SimpleCounter);
2      RWire#(UInt#(32)) incrWire <- mkRWire();
3      RWire#(UInt#(32)) decrWire <- mkRWire();
4      Reg#(UInt#(32)) cntr <- mkReg(0);
5      // Rules...
6      //////////////
7      method Action incr(UInt#(32) v);
8          incrWire.wset(v);
9      endmethod
10     method Action decr(UInt#(32) v);
11         decrWire.wset(v);
12     endmethod
13     method UInt#(32) counterValue();
14         return cntr;
15     endmethod
16 endmodule
```

Maybe?



```
1  rule count;
2      let counterVal = cntr;
3      Maybe#(UInt#(32)) maybeIncr = incrWire.wget();
4      Maybe#(UInt#(32)) maybeDecr = decrWire.wget();
5      UInt#(32) incrVal = 0;
6      UInt#(32) decrVal = 0;
7      if(isValid(maybeIncr)) begin
8          incrVal = maybeIncr.Valid;
9      end
10     if(isValid(maybeDecr)) begin
11         decrVal = maybeDecr.Valid;
12     end
13     cntr <= cntr + incrVal - decrVal;
14 endrule
```



Aufgabe 2.2.3



- ▶ load Methode hinzufügen
- ▶ Gleichzeitig Vereinfachung der Methode

```
1  rule count;
2  let counterVal = cntr;
3  Maybe#(UInt#(32)) maybeIncr = incrWire.wget();
4  Maybe#(UInt#(32)) maybeDecr = decrWire.wget();
5  Maybe#(UInt#(32)) maybeLoad = loadWire.wget();
6
7  UInt#(32) incrVal = fromMaybe(0, maybeIncr);
8  UInt#(32) decrVal = fromMaybe(0, maybeDecr);
9  UInt#(32) baseVal = fromMaybe(cntr, maybeLoad);
10
11  cntr <= baseVal + incrVal - decrVal;
12  endrule
```

Maybe!

- ▶ $cntr + 5$
- ▶ $cntr + 5 - 6$
- ▶ $cntr - 4$
- ▶ $1024 + 42 - 48$





Aufgabe 2.3



- ▶ Möglichkeit zur Wiederverwendung von Interfaces
- ▶ Standardisierte Schnittstellen
- ▶ Aufgabe: Berechnung von $((((x + a) \times b) \times c)/4) + 128$
- ▶ Parameter a , b und c können zur Laufzeit verändert werden

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



```
1  module mkChangeableUnit#(function Int#(32) f(Int#(32) a, Int#(32) b))
2                                     (CalcUnitChangeable);
3      Reg#(Int#(32)) p <- mkReg(0);
4      Wire#(Int#(32)) a <- mkWire();
5      FIFO#(Int#(32)) r <- mkFIFO();
6      rule doCalc;
7          r.enq(f(a, p));
8      endrule
9      method Action setParameter(Int#(32) param);
10         p <= param;
11     endmethod
12     interface CalcUnit calc;
13         method Action put(Int#(32) v);
14             a <= v;
15         endmethod
16
17         method ActionValue#(Int#(32)) result;
18             r.deq();
19             return r.first();
20         endmethod
21     endinterface
22 endmodule
```



```
1  module mkCalcUnit#(function Int#(32) f(Int#(32) a))(CalcUnit);
2      Wire#(Int#(32)) a <- mkWire();
3      FIFO#(Int#(32)) r <- mkFIFO();
4
5      rule calc;
6          r.enq(f(a));
7      endrule
8
9      method Action put(Int#(32) v);
10         a <= v;
11     endmethod
12
13     method ActionValue#(Int#(32)) result;
14         r.deq();
15         return r.first();
16     endmethod
17 endmodule
```



```
1  module mkSomeCalculation (CalcUnit);
2      Reg#(Int#(32)) a <- mkReg(42);
3      Reg#(Int#(32)) b <- mkReg(2);
4      Reg#(Int#(32)) c <- mkReg(4);
5      function addFun(x,y) = x + y;
6      function timesFun(x,y) = x * y;
7      function divBy4Fun(x) = x / 4;
8      function add128Fun(x) = x + 128;
9
10     CalcUnitChangeable addA <- mkChangeableUnit(addFun);
11     CalcUnitChangeable timesB <- mkChangeableUnit(timesFun);
12     CalcUnitChangeable timesC <- mkChangeableUnit(timesFun);
13     Vector#(5,CalcUnit) calcUnits;
14     calcUnits[0] = addA.calc;
15     calcUnits[1] = timesB.calc;
16     calcUnits[2] = timesC.calc;
17     calcUnits[3] <- mkCalcUnit(divBy4Fun);
18     calcUnits[4] <- mkCalcUnit(add128Fun);
```



```
1  Reg#(Bool) initialised <- mkReg(False);
2  rule initialise (!initialised);
3      initialised <= True;
4      addA.setParameter(a);
5      timesB.setParameter(b);
6      timesC.setParameter(c);
7  endrule
8
9  rule setupCalc;
10     calcUnits[0].put(inFIFO.first());
11     inFIFO.deq();
12 endrule
13
14 rule outputResult;
15     let result <- calcUnits[4].result();
16     outFIFO.enq(result);
17 endrule
```

```
1   for(Integer i = 1; i < 5; i = i + 1) begin
2       rule calc;
3           let t <- calcUnits[i - 1].result();
4           calcUnits[i].put(t);
5       endrule
6   end
```



Übung 3: BlueCheck

Übung 3: Testen gegen Bedingungen



```
1  module [BlueCheck] mkArithSpec ();
2      function Bool addComm(Int#(4) x, Int#(4) y) =
3          x + y == y + x;
4
5      function Bool addAssoc(Int#(4) x, Int#(4) y, Int#(4) z) =
6          x + (y + z) == (x + y) + z;
7
8      function Bool subComm(Int#(4) x, Int#(4) y) =
9          x - y == y - x;
10
11     prop("addComm" , addComm);
12     prop("addAssoc" , addAssoc);
13     prop("subComm" , subComm);
14 endmodule
15
16 module [Module] mkArithChecker ();
17     blueCheck(mkArithSpec);
18 endmodule
```

Übung 3: Testen gegen Bedingungen



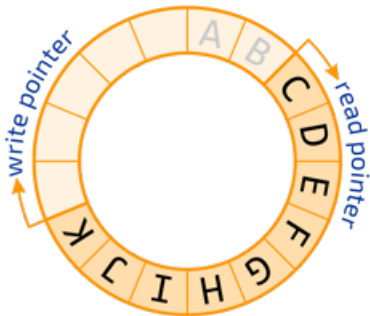
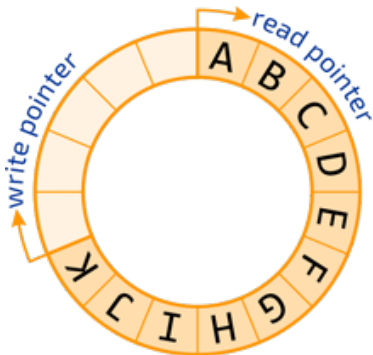
```
1  module [BlueCheck] mkArithSpec ();
2  ...
3      function Bool zeroTimesX(Int#(4) x) =
4          x * 0 == 0;
5
6      function Bool divIsMostlyNotComm(Int#(4) x, Int#(4) y) =
7          x == 0 || y == 0 || (x == y) || (-x == y) || x / y != y / x;
8
9      prop("zeroTimesX" , zeroTimesX);
10     prop("divIsMostlyNotComm" , divIsMostlyNotComm);
11     ...
12 endmodule
```



- ▶ Eine eigene FIFO entwickeln mit anderem Interface zu AzureIP
- ▶ Vergleich der eigenen Implementierung mit “Golden Sample”

```
1  interface FIFO
2      method Action          put (Int#(16) e); // Put Element on FIFO
3      method ActionValue#(Int#(16)) get (); // Get Element from FIFO
4  endinterface
```

Übung 3: FIFO als Ringspeicher



Von: <http://www.delphientwickler.de/delphi-news/178-generische-ringspeicher.html>

Übung 3: Ringspeicher in Bluespec



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Was ist nötig?

Übung 3: Ringspeicher in Bluespec

- ▶ Was ist nötig?
- ▶ Zustandselemente für Speicher

Übung 3: Ringspeicher in Bluespec

- ▶ Was ist nötig?
- ▶ Zustandselemente für Speicher
- ▶ Zustandselemente für Lese- und Schreibzeiger

```
1   Reg#(UInt#(4)) writePtr  <- mkReg(0);
2   Reg#(UInt#(4)) readPtr  <- mkReg(0);
3
4   Vector#(16, Reg#(Int#(16))) buffer <- replicateM(mkRegU());
```

Übung 3: Ringspeicher in Bluespec



```
1  module mkMyFIFO(MyFIFO);
2      Reg#(UInt#(4)) writePntr <- mkReg(0);
3      Reg#(UInt#(4)) readPntr <- mkReg(0);
4
5      Vector#(16, Reg#(Int#(16))) buffer <- replicateM(mkRegU());
6
7      method Action put(Int#(16) e) if((writePntr + 1) != readPntr);
8          writePntr <= writePntr + 1;
9          buffer[writePntr] <= e;
10     endmethod
11
12     method ActionValue#(Int#(16)) get() if(readPntr != writePntr);
13         readPntr <= readPntr + 1;
14         return buffer[readPntr];
15     endmethod
16 endmodule
```




► Beispiel aus dem BlueCheck Repository: Stacks

```
1  module [BlueCheck] checkStack ();
2      /* Specification instance */
3      Stack#(8, Bit#(4)) spec <- mkStackSpec();
4
5      /* Implmentation instance */
6      Stack#(8, Bit#(4)) imp <- mkBRAMStack();
7
8      equiv("pop"      , spec.pop      , imp.pop);
9      equiv("push"     , spec.push     , imp.push);
10     equiv("isEmpty", spec.isEmpty, imp.isEmpty);
11     equiv("top"      , spec.top      , imp.top);
12 endmodule
```

► Was ist unser "Golden Sample"?



► Hilfsfunktion um Interfaces zu Vereinheitlichen

```
1  module [BlueCheck] mkFIFOSpec ();
2      FIFO#(Int#(16)) spec <- mkSizedFIFO(16);
3      MyFIFO impl <- mkMyFIFO();
4
5      function ActionValue#(Int#(16)) pop(FIFO#(Int#(16)) e);
6          actionvalue
7              e.deq();
8              return e.first();
9          endactionvalue
10     endfunction
11
12     equiv("put", spec.enq, impl.put);
13     equiv("get", pop(spec), impl.get);
14 endmodule
```



Übung 4 bis 7: FAST Bildfilter

Übung 4: Implementierung von FAST

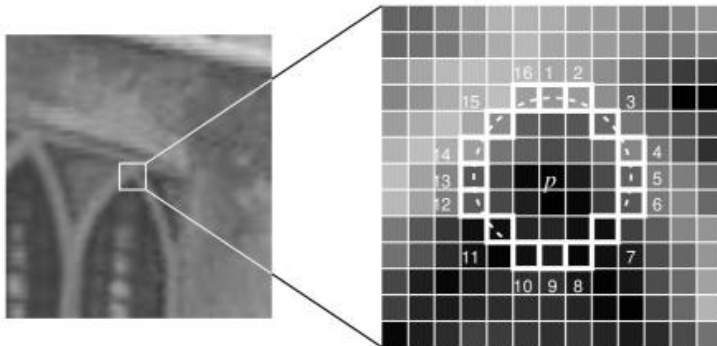


TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Features from accelerated segment test (FAST)

Übung 4: Implementierung von FAST

- ▶ Features from accelerated segment test (FAST)
- ▶ → Kantendetektieralgorithmus



Von: https://docs.opencv.org/3.0-beta/_images/fast_speedtest.jpg