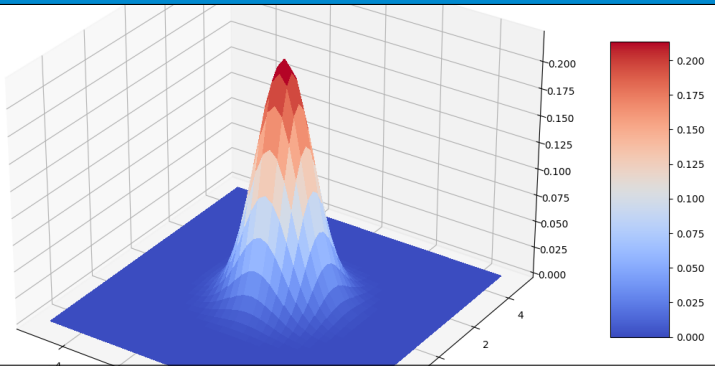
Architekturen und Entwurf von Rechnersystemen
Besprechung Übungsblatt 4 + Theorieblatt 2

Wintersemester 2022/2023

Yannick Lavan
Fachgebiet Eingebettete Systeme und ihre Anwendungen

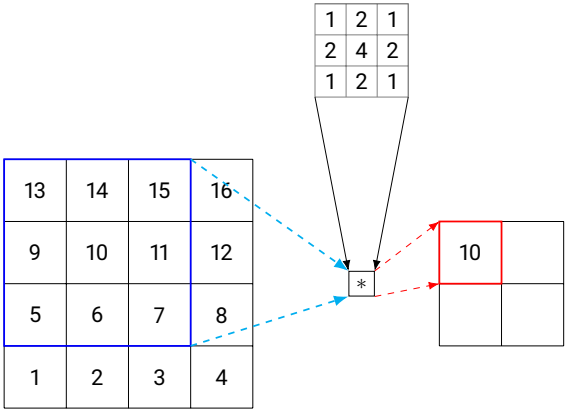
TECHNISCHE UNIVERSITÄT DARMSTADT



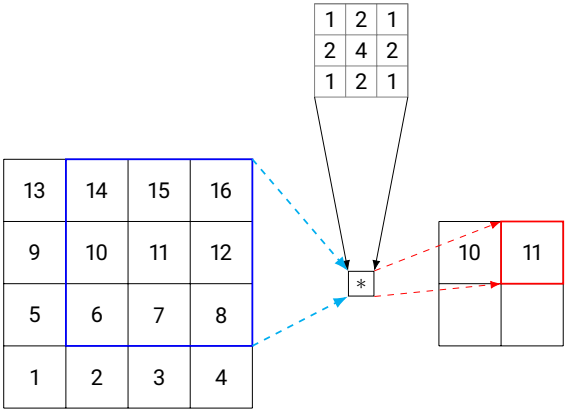
TECHNISCHE
UNIVERSITÄT
DARMSTADT

Übung 4

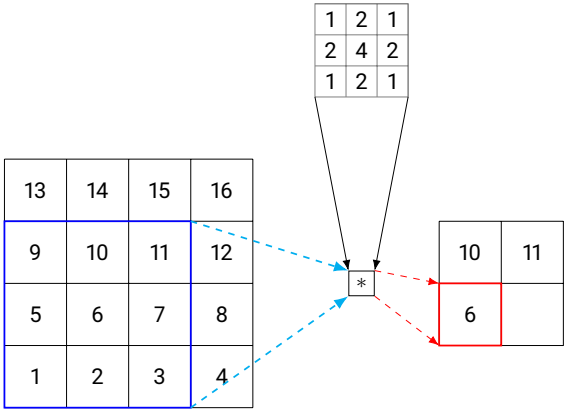
Funktionsweise



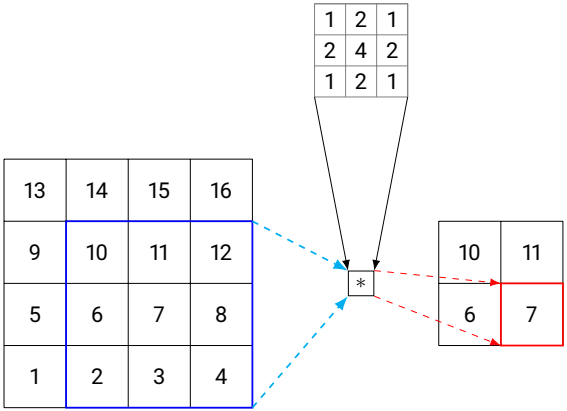
Funktionsweise



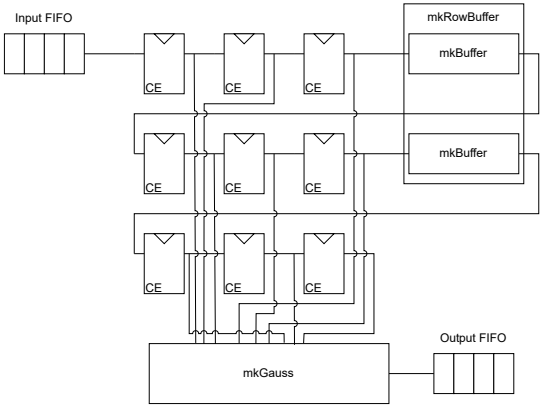
Funktionsweise



Funktionsweise



Hardware-Architektur

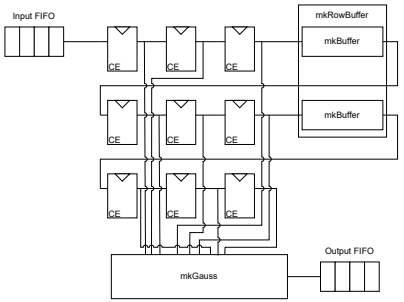


Sliding Window und Verbindungen

Boilerplate Code

TECHNISCHE UNIVERSITÄT DARMSTADT

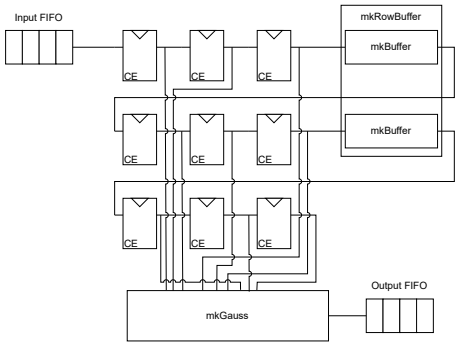
```
1  module mkGaussAccelerator(Accelerator);
2      FIFO#(GrayScale) in <- mkFIFO();
3      FIFO#(GrayScale) out <- mkFIFO();
4      Reg#(UInt#(32)) npix <- mkReg(0);
5      Reg#(Bool) started <- mkReg(False);
6
7      FilterServer filter <- mkGauss();
8      RowBufferServer rowbuffer <- mkRowBuffer();
9      Vector#(3, Vector#(3, Reg#(Maybe#(GrayScale)))) workingField
   ↪     <- replicateM(replicateM(mkReg(tagged Invalid)));
10
11     // Connections and control logic omitted...
12     method Action setRes(UInt#(32) n_pixels) if(!started);
13         npix <= n_pixels;
14         started <= True;
15     endmethod
16     interface AcceleratorServer server;
17         interface Put request = toPut(in);
18         interface Get response = toGet(out);
19     endinterface
20 endmodule
```



Verbindung Registerfeld - Buffer

TECHNISCHE UNIVERSITÄT DARMSTADT

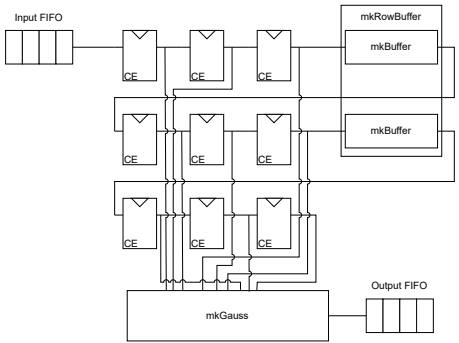
```
1  rule populate;
2      workingField[0][0] <= tagged Valid new__px;
3      // Move forward in register field
4      for(Integer x = 1; x <= 2; x = x + 1) begin
5          for(Integer y = 0; y <= 2; y = y + 1) begin
6              workingField[y][x] <= workingField[y][x-1];
7          end
8      end
9      // Populate and drain row buffers
10     Vector#(2, Maybe#(GrayScale)) nextin;
11     for(Integer y = 0; y < 2; y = y + 1) begin
12         nextin[y] = workingField[y][2];
13     end
14     rowbuffer.request.put(nextin);
15 endrule
```



Verbindung Registerfeld - Buffer

TECHNISCHE UNIVERSITÄT DARMSTADT

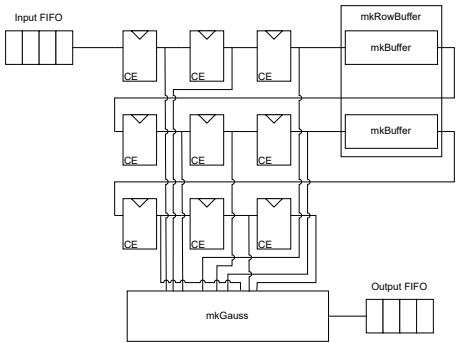
```
1  rule populate;
2      workingField[0][0] <= tagged Valid new__px;
3      // Move forward in register field
4      for(Integer x = 1; x <= 2; x = x + 1) begin
5          for(Integer y = 0; y <= 2; y = y + 1) begin
6              workingField[y][x] <= workingField[y][x-1];
7          end
8      end
9      // Populate and drain row buffers
10     Vector#(2, Maybe#(GrayScale)) nextin;
11     for(Integer y = 0; y < 2; y = y + 1) begin
12         nextin[y] = workingField[y][2];
13     end
14     rowbuffer.request.put(nextin);
15 endrule
16 rule drain(...);
17     let nextout <- rowbuffer.response.get();
18     for(Integer y = 0; y < 2; y = y + 1) begin
19         workingField[y+1][0] <= nextout[y];
20     end
21 endrule
```



Verbindung Sliding Window - Filter - Output

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  rule constructKernel (...);
2      Vector#(9, GrayScale) toGauss = replicate(0);
3      for(Integer y = 0; y < 3; y = y + 1) begin
4          for(Integer x = 0; x < 3; x = x + 1) begin
5              toGauss[3*y+x] = fromMaybe(0, workingField[y][x]);
6          end
7      end
8      filter.request.put(toGauss);
9      /*
10     ...Control logic...
11     */
12 endrule
13
14 rule forwardResult;
15     let t <- filter.response.get();
16     out.enq(t);
17 endrule
```



Kontrolllogik

Kontrolllogik - Vorüberlegungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Steuerung des Datenflusses

Kontrolllogik - Vorüberlegungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Steuerung des Datenflusses
  - Wann bewegen sich Pixel im Registerfeld?

Kontrolllogik - Vorüberlegungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Steuerung des Datenflusses
  - Wann bewegen sich Pixel im Registerfeld?
  - Wann Pixel vom Registerfeld an Row-Buffer übergeben?

Kontrolllogik - Vorüberlegungen

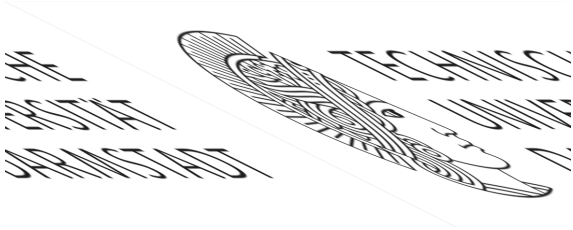
TECHNISCHE UNIVERSITÄT DARMSTADT

- Steuerung des Datenflusses
  - Wann bewegen sich Pixel im Registerfeld?
  - Wann Pixel vom Registerfeld an Row-Buffer übergeben?
  - Wann Pixel vom Row-Buffer zum Registerfeld übergeben?

Kontrolllogik - Vorüberlegungen

TECHNISCHE UNIVERSITÄT DARMSTADT

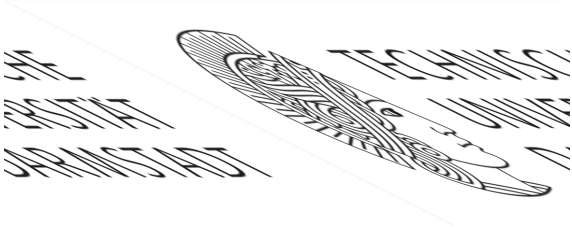
- Steuerung des Datenflusses
  - Wann bewegen sich Pixel im Registerfeld?
  - Wann Pixel vom Registerfeld an Row-Buffer übergeben?
  - Wann Pixel vom Row-Buffer zum Registerfeld übergeben?
  - Wann Pixel vom Registerfeld an Filterkernel übergeben?



Kontrolllogik - Vorüberlegungen

TECHNISCHE UNIVERSITÄT DARMSTADT

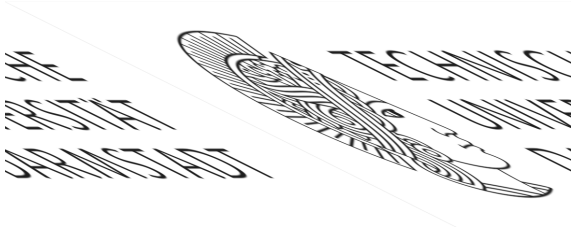
- Steuerung des Datenflusses
  - Wann bewegen sich Pixel im Registerfeld?
  - Wann Pixel vom Registerfeld an Row-Buffer übergeben?
  - Wann Pixel vom Row-Buffer zum Registerfeld übergeben?
  - Wann Pixel vom Registerfeld an Filterkernel übergeben?
- Umsetzung in BSV über explizite und implizite Guards



Kontrolllogik - Vorüberlegungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Steuerung des Datenflusses
  - Wann bewegen sich Pixel im Registerfeld?
  - Wann Pixel vom Registerfeld an Row-Buffer übergeben?
  - Wann Pixel vom Row-Buffer zum Registerfeld übergeben?
  - Wann Pixel vom Registerfeld an Filterkernel übergeben?
- Umsetzung in BSV über explizite und implizite Guards
- Aber zuerst: Betrachtung der wichtigen Szenarien



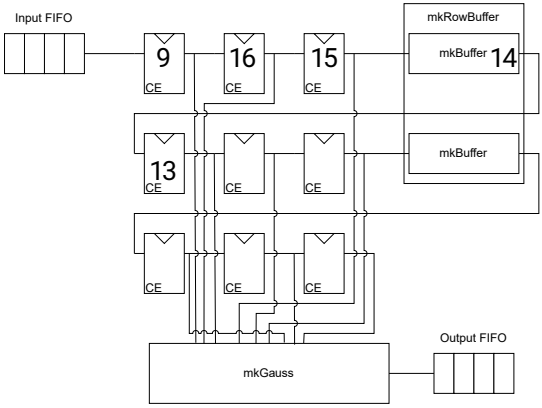
Hardware-Beispiel - Situation 1

Kein Filtern

TECHNISCHE UNIVERSITÄT DARMSTADT

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Eingabebild

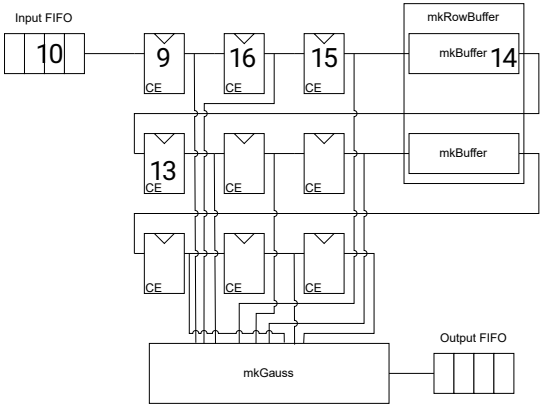


Hardware-Beispiel - Situation 2

Rotate 1

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Eingabebild

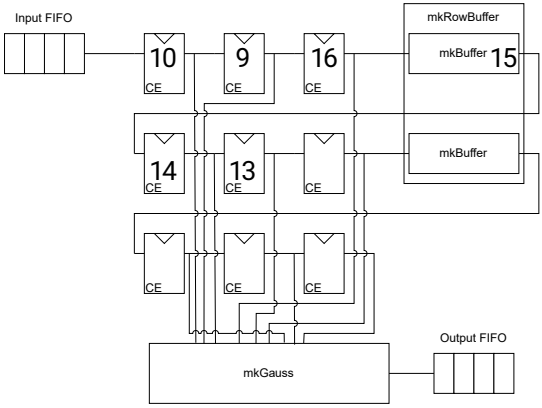


Hardware-Beispiel - Situation 2

Rotate 2

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Eingabebild



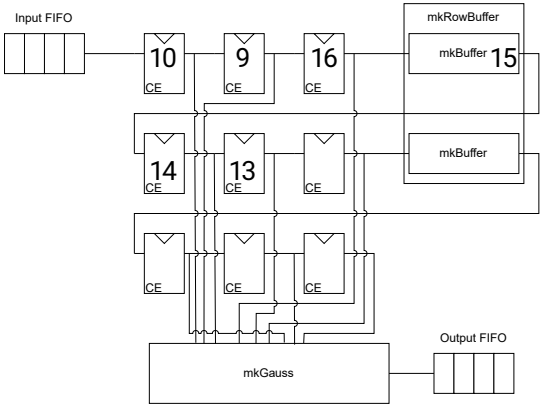
Hardware-Beispiel - Situation 3

Bad Rotate 1

TECHNISCHE UNIVERSITÄT DARMSTADT

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Eingabebild



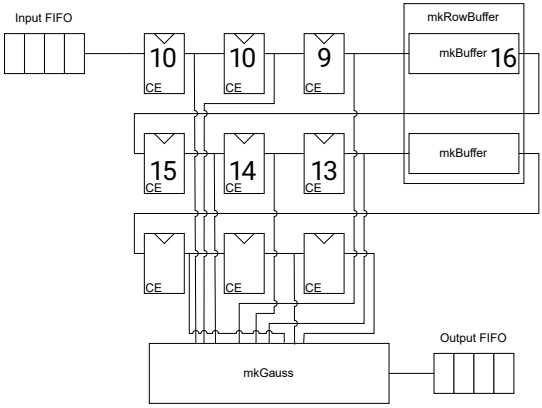
Hardware-Beispiel - Situation 3

Bad Rotate 2

TECHNISCHE UNIVERSITÄT DARMSTADT

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Eingabebild



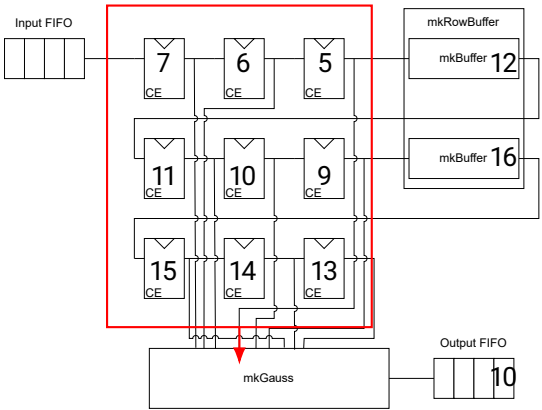
Hardware-Beispiel - Situation 4

Bad Kernel 1

TECHNISCHE UNIVERSITÄT DARMSTADT

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Eingabebild



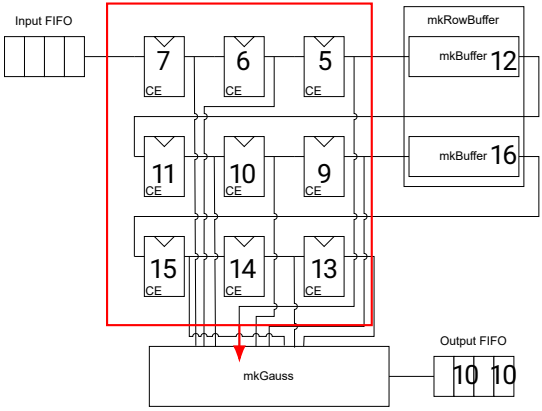
Hardware-Beispiel - Situation 4

Bad Kernel 2

TECHNISCHE UNIVERSITÄT DARMSTADT

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Eingabebild



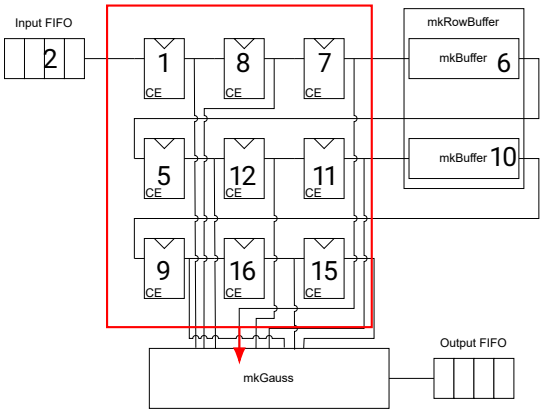
Hardware-Beispiel - Situation 5

Edge Pixel 1

TECHNISCHE UNIVERSITÄT DARMSTADT

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Eingabebild



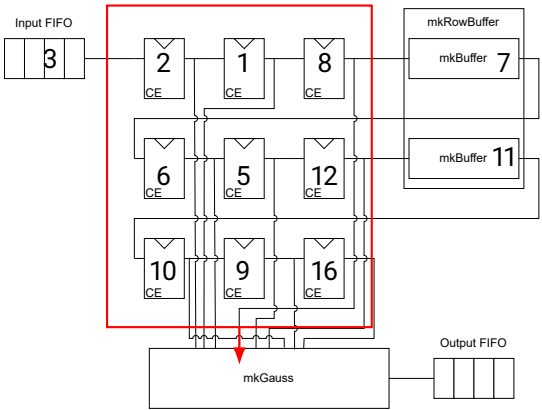
Hardware-Beispiel - Situation 5

Edge Pixel 2

TECHNISCHE UNIVERSITÄT DARMSTADT

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Eingabebild



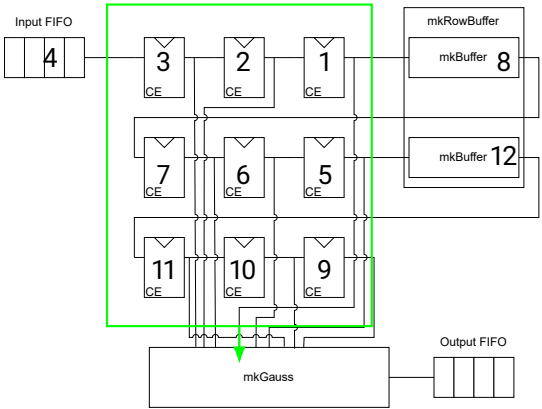
Hardware-Beispiel - Situation 5

Edge Pixel 3

TECHNISCHE UNIVERSITÄT DARMSTADT

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Eingabebild



Implementierung Rotate

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Wire#(GrayScale) new_px <- mkWire();
2  Wire#(Bool) rotate <- mkDWire(False);
3
4  rule read_in (npix > 0);
5      let p0 = in.first;
6      in.deq;
7      new_px <= p0;
8      npix <= npix - 1;
9      rotate <= True;
10     // ...more logic later
11 endrule
12
13 rule populate;
14     workingField[0][0] <= tagged Valid new_px;
15 // rest of rule on slide 10
16 endrule
```

Implementierung Rotate

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Wire#(GrayScale) new_px <- mkWire();
2  Wire#(Bool) rotate <- mkDWire(False);
3
4  rule read_in (npix > 0);
5      let p0 = in.first;
6      in.deq;
7      new_px <= p0;
8      npix <= npix - 1;
9      rotate <= True;
10     // ...more logic later
11 endrule
12
13 rule populate;
14     workingField[0][0] <= tagged Valid new_px;
15 // rest of rule on slide 10
16 endrule
```

```
1 rule drain(rotate);
2     let nextout <- rowbuffer.response.get();
3     for(Integer y = 0; y < 2; y = y + 1) begin
4         workingField[y+1][0] <= nextout[y];
5     end
6 endrule
```

Kontrolllogik Kernel 1

Pixelankunft

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  Reg#(Bool) tofilter <- mkDReg(False);
2
3  rule read_in (npix > 0);
4      let p0 = in.first;
5      in.deq;
6      new_px <= p0;
7      npix <= npix - 1;
8      rotate <= True;
9      tofilter <= True;
10 endrule
11
12 rule constructKernel (isValid(workingField[2][2]) && tofilter && ...);
13 // next slide...
```

Kontrolllogik Kernel 2

Randpixel

TECHNISCHE UNIVERSITÄT DARMSTADT

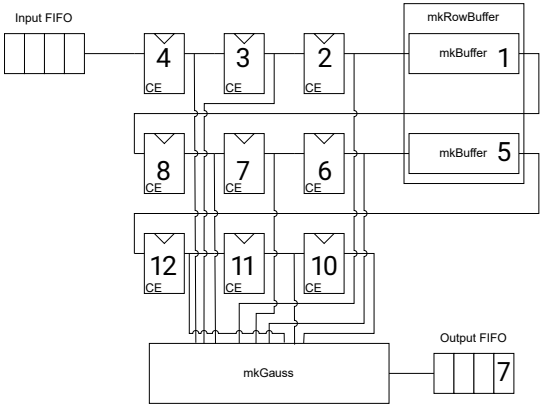
```
1  Reg#(UInt#(2)) timeout <- mkReg(0); // Used to avoid edge pixels
2  Reg#(UInt#(14)) col_cnt <- mkReg(fromInteger(width-2));
3
4  rule constructKernel (isValid(workingField[2][2]) && tofilter && timeout == 0);
5      /*
6      Feeding mkGauss omitted...
7      */
8      let t = col_cnt - 1;
9
10     if(t == 0) begin
11         timeout <= 3; // always wait 3 cycles so edge pixels don't cause computation
12         col_cnt <= fromInteger(width-2);
13     end
14     else begin
15         col_cnt <= t;
16     end
17 endrule
18
19 rule wait_timeout(timeout > 0 && rotate); // only reduce timeout if data arrived
20     timeout <= timeout - 1;
21 endrule
```

Wiederverwendbarkeit

Hardware-Beispiel - Alle Pixel verarbeitet

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Eingabebild



Erweiterte Interfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface BufferServer;
2      method Action clear();
3      interface Put#(Maybe#(GrayScale)) request;
4      interface Get#(Maybe#(GrayScale)) response;
5  endinterface: BufferServer
6
7  interface RowBufferServer;
8      method Action clear();
9      interface Put#(Vector#(2, Maybe#(GrayScale))) request;
10     interface Get#(Vector#(2, Maybe#(GrayScale))) response;
11 endinterface: RowBufferServer
```

clear-Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

mkRowBuffer:

```
1 method Action clear();
2     inputValue.clear();
3     outputValue.clear();
4     for(Integer i = 0; i < 2; i = i + 1)
5         buffer[i].clear();
6 endmethod
```

mkBuffer:

```
1 method Action clear();
2     counterInput <= 0;
3     flag <= False;
4     inputValue.clear();
5     outputValue.clear();
6     bufferedValue.clear();
7 endmethod
```

Berechnungsende

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Host liest Speicher

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Host liest Speicher
  - Wann sind die Werte im Speicher gültig?

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Host liest Speicher
  - Wann sind die Werte im Speicher gültig?
- Ansatz 1: Polling (Active waiting)

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Host liest Speicher
  - Wann sind die Werte im Speicher gültig?
- Ansatz 1: Polling (Active waiting)
  - Verschwendete Rechenzeit

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Host liest Speicher
  - Wann sind die Werte im Speicher gültig?
- Ansatz 1: Polling (Active waiting)
  - Verschwendete Rechenzeit
- Ansatz 2: Interrupts

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Host liest Speicher
  - Wann sind die Werte im Speicher gültig?
- Ansatz 1: Polling (Active waiting)
  - Verschwendete Rechenzeit
- Ansatz 2: Interrupts
  - Host arbeitet parallel an etwas anderem

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Host liest Speicher
  - Wann sind die Werte im Speicher gültig?
- Ansatz 1: Polling (Active waiting)
  - Verschwendete Rechenzeit
- Ansatz 2: Interrupts
  - Host arbeitet parallel an etwas anderem
  - Interrupt löst Wechsel zu Ergebnisverarbeitung aus

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Host liest Speicher
  - Wann sind die Werte im Speicher gültig?
- Ansatz 1: Polling (Active waiting)
  - Verschwendete Rechenzeit
- Ansatz 2: Interrupts
  - Host arbeitet parallel an etwas anderem
  - Interrupt löst Wechsel zu Ergebnisverarbeitung aus
- In unserem Beispiel:

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Host liest Speicher
  - Wann sind die Werte im Speicher gültig?
- Ansatz 1: Polling (Active waiting)
  - Verschwendete Rechenzeit
- Ansatz 2: Interrupts
  - Host arbeitet parallel an etwas anderem
  - Interrupt löst Wechsel zu Ergebnisverarbeitung aus
- In unserem Beispiel:
  - Host resetted Beschleuniger bei Interrupt

Implementierung Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  method Bool irq();
2      return started && npix == 0;
3  endmethod
4
5  method Action ack() if(started && npix == 0);
6      started <= False;
7      rowbuffer.clear();
8      for(Integer i = 0; i < 3; i = i + 1) begin
9          for(Integer j = 0; j < 3; j = j + 1) begin
10             workingField[i][j] <= tagged Invalid;
11         end
12     end
13 endmethod
```

Selbstprüfende Testbench

Ansatz

TECHNISCHE UNIVERSITÄT DARMSTADT

- Orakel bekommt Eingabebild übergeben

Ansatz

TECHNISCHE UNIVERSITÄT DARMSTADT

- Orakel bekommt Eingabebild übergeben
- Orakel berechnet gesamte Ausgabe

Ansatz

TECHNISCHE UNIVERSITÄT DARMSTADT

- Orakel bekommt Eingabebild übergeben
- Orakel berechnet gesamte Ausgabe
- Vergleiche DUT-Output mit nächstem Orakeloutput.

Boilerplate Code

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  module mkAcceleratorChecker(Empty);
2      Reg#(UInt#(32)) readCounter <- mkReg(0);
3      Reg#(UInt#(32)) writeCounter <- mkReg(0);
4      Reg#(UInt#(64)) addressRead <- mkRegU;
5      Reg#(UInt#(64)) oracle_ptr <- mkRegU;
6      Reg#(UInt#(32)) n_pixels_in <- mkRegU;
7      Reg#(UInt#(32)) n_pixels_out <- mkRegU;
8
9      Accelerator uut <- mkGaussAccelerator();
10
11     Stmt test = seq
12     /*
13     ... next slides...
14     */
15     endseq;
16
17     mkAutoFSM(test);
18 endmodule
```

Initialisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  action
2      let t1 <- readImage_create("./picture.png");
3      addressRead <= t1;
4      $display("Reading image, is at: %d", t1);
5      n_pixels_in <= fromInteger(width) * fromInteger(height);
6      n_pixels_out <= fromInteger(width-2) * fromInteger(height-2);
7      let t2 <- oracle_create(t1, fromInteger(width), fromInteger(height));
8      oracle_ptr <= t2;
9      $display("Oracle is at: 0x%h", t2);
10 endaction
11 action
12     uut.setRes(n_pixels_in);
13 endaction
```

Testdurchführung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  par
2      for(readCounter <= 0; readCounter < n_pixels_in; readCounter <= readCounter + 1) action
3          let pixel <- readImage_getPixel(addressRead);
4          uut.server.request.put(pixel);
5      endaction
6
7      for(writeCounter <= 0; writeCounter < n_pixels_out; writeCounter <= writeCounter + 1) action
8          let new_pixel <- uut.server.response.get;
9          let pixel_exp <- oracle_get_next_pixel(oracle_ptr);
10         if(new_pixel != pixel_exp) begin
11             match {.x, .y} = row_major_to_xy(writeCounter);
12             $display("Error at pixel (%d,%d). Expected %d, got %d", x, y, pixel_exp, new_pixel);
13             $finish();
14         end
15     endaction
16 endpar
```

Clean-Up

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1     /*
2     ... previous slides...
3     */
4     readImage_delete(addressRead);
5     oracle_delete(oracle_ptr);
6     $display("Test passed");
7  endseq;
8
9  mkAutoFSM(test);
```

Fragen zu Übung 4