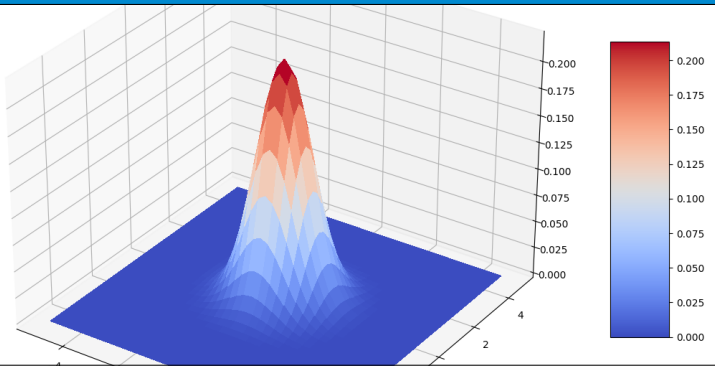
Architekturen und Entwurf von Rechnersystemen
Besprechung Übungsblatt 3

Wintersemester 2022/2023

Yannick Lavan
Fachgebiet Eingebettete Systeme und ihre Anwendungen

TECHNISCHE UNIVERSITÄT DARMSTADT



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Übung 3: Gaussfilter - Teil 1

Grundlagen

Funktionsweise

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gewichtete Summe der Pixel innerhalb eines Fensters
- Rauschunterdrückung in Bildern
- Kantenschwächung
- Gewichte anhand 2-D Gaussverteilung berechnet

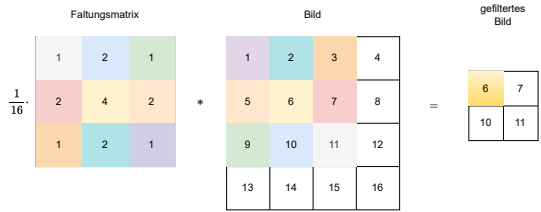


Abbildung: Einfaches Rechenbeispiel für Gaussfilterung.

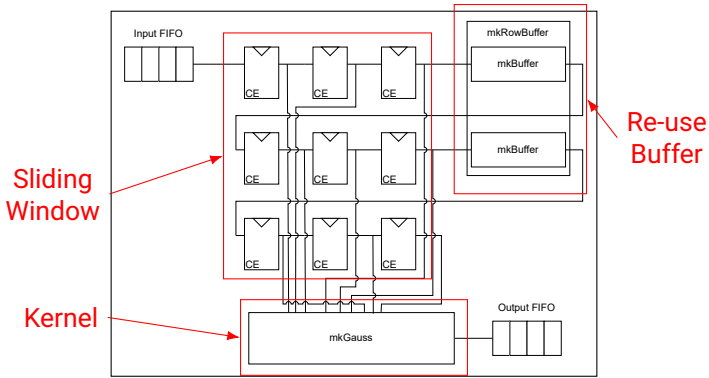
Freiheitsgrade Filter

TECHNISCHE UNIVERSITÄT DARMSTADT

- Größe der relevanten Bildregion
- Standardabweichung der Verteilung

In dieser Übung verwendet: 3×3 Region und $\sigma \approx 0.8$

System



Skizze Stream-basierte Architektur (ohne Kontrolllogik)

Implementierung - Kernel

Überlegungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eintakt/Mehrtakt/Pipeline
  - Eintakt ⇒ Latenz ↓, Fläche ↓, Frequenz ↓, Durchsatz ↓
  - Mehrtakt ⇒ Latenz ↑, Fläche ↑, Frequenz ↑, Durchsatz ↓
  - Pipeline ⇒ Latenz ↑, Fläche ↑, Frequenz ↑, Durchsatz ↑
- Umsetzung Multiplikation/Division
  - Zweierpotenzen ⇒ Shifts!

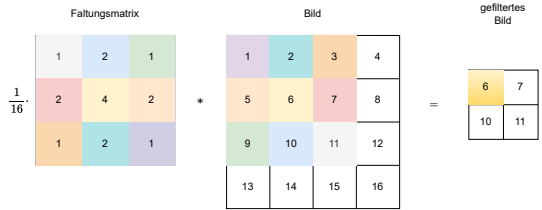


Abbildung: Einfaches Rechenbeispiel für Gaussfilterung.

Kernel - Boilerplate Code

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  typedef Server#(Vector#(9, GrayScale), GrayScale) FilterServer;
2
3  module mkGauss(FilterServer);
4      FIFO#(Vector#(9, GrayScale)) in <- mkFIFO();
5      FIFO#(GrayScale) out <- mkFIFO();
6      Vector#(9, Integer) weights; // Hard-wired kernel weights
7      weights[0] = 1;
8      weights[1] = 2;
9      weights[2] = 1;
10     weights[3] = 2;
11     weights[4] = 4;
12     weights[5] = 2;
13     weights[6] = 1;
14     weights[7] = 2;
15     weights[8] = 1;
16     /*
17     Rule on next slide...
18     */
19     interface Put request = toPut(in);
20     interface Get response = toGet(out);
21 endmodule : mkGauss
```

Kernel - Arithmetik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   rule convolve;
2       let pixels = in.first();
3       in.deq();
4       Bit#(12) mulres = 0;
5       for(Integer i = 0; i < 9; i = i + 1) begin
6           Bit#(12) px_ex = extend(pixels[i]);
7           mulres = mulres + (px_ex « log2(weights[i]));
8       end
9       out.enq(truncate(mulres » 4));
10  endrule
```

Kernel - Testbench Hilfsfunktionen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  // Bild laden/freigeben
2  import "BDPI" function ActionValue#(UInt#(64)) readImage_create(String filename);
3  import "BDPI" function Action readImage_delete(UInt#(64) addr);
4  // Pixel an bestimmter Position lesen
5  import "BDPI" function GrayScale readImage_getPixelAt(UInt#(64) addr, UInt#(32) idx);
6
7  // Gausskernelergebnis für Arbeitsfenster
8  import "BDPI" function GrayScale getGaussResult(Vector#(9, GrayScale) pixels);
9
10 // Koordinatenumrechnung
11 function UInt#(32) xy_to_row_major(UInt#(32) x, UInt#(32) y);
12 function Tuple2#(Int#(32), Int#(32)) row_major_to_xy(UInt#(32) idx);
13
14 // Pixel auslesen
15 function GrayScale get_padded_pixel(UInt#(64) addressRead, Int#(32) x, Int#(32) y);
```

Kernel - Testbench Komponenten

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  module mkGaussChecker(Empty);
2      // Unit under test (our filter)
3      FilterServer uut <- mkGauss();
4      FIFO#(GrayScale) reference_values <- mkFIFO();
5      Reg#(UInt#(32)) read_x <- mkReg(0);
6      Reg#(UInt#(32)) read_y <- mkReg(0);
7      Reg#(UInt#(32)) checkCounter <- mkReg(0);
8      Reg#(UInt#(64)) addressRead <- mkRegU;
9      Reg#(Bool) failed <- mkReg(False);
10     Reg#(UInt#(32)) n_pixels <- mkRegU;
11     Stmt checkFilter = seq
12         // On following slides...
13     endseq;
14
15     mkAutoFSM(checkFilter);
16 endmodule : mkGaussChecker
```

Kernel - Testbench Input und Referenzergebnisse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   Stmt checkFilter = seq
2       action
3           let t1 <- readImage_create("./picture.png");
4           addressRead <= t1;
5           $display("Reading image, is at: %d", t1);
6           n_pixels <= fromInteger(width-2) * fromInteger(height-2);
7       endaction
8       par
9           for(read_y <= 1; read_y < fromInteger(height-1); read_y <= read_y + 1) seq
10              for(read_x <= 1; read_x < fromInteger(width-1); read_x <= read_x + 1) action
11                  Vector#(9, GrayScale) field = replicate(0);
12                  for(Int#(32) ky = -1; ky <= 1; ky = ky + 1) begin
13                      for(Int#(32) kx = -1; kx <= 1; kx = kx + 1) begin
14                          field[3*(ky+1)+kx+1] = get_padded_pixel(addressRead, unpack(pack(read_x))+kx,
    ↪   unpack(pack(read_y))+ky);
15                      end
16                  end
17                  uut.request.put(field);
18                  reference_values.enq(getGaussResult(field));
19              endaction
20          endseq
```

Kernel - Testbench Vergleich

TECHNISCHE UNIVERSITÄT DARMSTADT

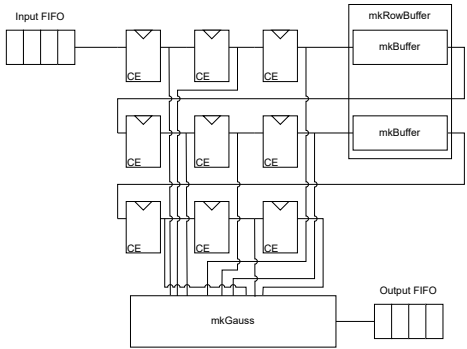
```
21          // Continued from previous slide
22          for(checkCounter <= 0; checkCounter < n_pixels; checkCounter <= checkCounter + 1) action
23              let new_pixel <- uut.response.get();
24              let ref_pixel = reference_values.first();
25              reference_values.deq();
26              if(new_pixel != ref_pixel) begin
27                  match {.x, .y} = row_major_to_xy(checkCounter);
28                  $display("Error at pixel (%d,%d). Expected %d, got %d", x, y, ref_pixel, new_pixel);
29                  failed <= True;
30              end
31          endaction
32      endpar
33      readImage_delete(addressRead);
34      action
35          if(failed)
36              $display("Test failed");
37          else
38              $display("Test successful");
39      endaction
40  endseq;
```

Fragen zur Kernelimplementierung/Testbench?

Implementierung - Buffer

Buffer - Überlegungen

- Sliding Window drei Pixel breit
  - $\Rightarrow$ Buffer speichert *Breite* $-\ 3$ Pixel
- Wenig kombinatorische Logik zwischen Registerfeld und Buffer
  - $\Rightarrow$ BypassFIFOs für Latenzminimierung i.O.



Buffer - Komponenten

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface BufferServer;
2      interface Put#(Maybe#(GrayScale)) request;
3      interface Get#(Maybe#(GrayScale)) response;
4  endinterface: BufferServer
5
6  module mkBuffer(BufferServer);
7      FIFO#(Maybe#(GrayScale)) inputValue <- mkBypassFIFO;
8      FIFO#(Maybe#(GrayScale)) outputValue <- mkBypassFIFO;
9      FIFO#(Maybe#(GrayScale)) bufferedValue <- mkSizedBRAMFIFO(width-3+1);
10     Reg#(Bit#(12)) counterInput <- mkReg(0);
11     Reg#(Bool) flag <- mkReg(False);
12     // Rules on next slide...
13
14     interface Put request = toPut(inputValue);
15     interface Get response = toGet(outputValue);
16 endmodule
```


Embedded Systems & Applications

Buffer - Implementierung Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   rule drainInput;
2       let value = inputValue.first;
3       inputValue.deq;
4       bufferedValue.enq(value);
5       if(counterInput == fromInteger(width-3 - 1))
6           flag <= True;
7       else
8           counterInput <= counterInput + 1;
9   endrule
10  rule fillOutputValid (flag);
11      let value = bufferedValue.first;
12      bufferedValue.deq;
13
14      outputValue.enq(value);
15  endrule
```

Buffer - Testbench 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  module mkBufferTb(Empty);
2      BufferServer dut <- mkBuffer();
3      Reg#(UInt#(32)) n_put <- mkReg(0); // used to check if we reached X values
4      Reg#(UInt#(32)) filled <- mkReg(0);
5      Reg#(UInt#(32)) drained <- mkReg(0);
6      Reg#(Bool) failure <- mkReg(False);
7
8      Stmt do_test = seq
9          $display("Starting test");
10         par
11             // Fill loop on next slide...
12             // ...
13             // Draining loop after next slide...
14         endpar
15         action
16             if(failure)
17                 $display("Test failed");
18             else
19                 $display("Test successful");
20         endaction
21     endseq;
22     mkAutoFSM(do_test);
```

Buffer - Testbench Fill Loop

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  seq
2      for(filled <= 0; filled < fromInteger(width) - 3; filled <= filled + 1) action
3          dut.request.put(tagged Valid pack(filled)[7:0]);
4          n_put <= n_put + 1;
5      endaction
6      $display("Put all values");
7  endseq
```

Buffer - Testbench Drain Loop

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   for(drained <= 0; drained < fromInteger(width) - 3; drained <= drained + 1) action
2       Bool fail = False;
3       if(n_put < fromInteger(width) - 3) begin
4           $display("Error: Buffer should not provide results before obtaining width-3 pixels.");
5           fail = True;
6       end
7       let t <- dut.response.get();
8       if(t matches tagged Valid .tv) begin
9           if(tv != pack(drained)[7:0]) begin
10              $display("Values differed at %d. Expected %d, got %d", drained, pack(drained)[7:0], tv);
11              fail = True;
12          end
13      end
14      else begin
15          $display("Error: We never put invalid values into the buffer, so no invalid can come back.");
16          fail = True;
17      end
18      failure <= failure || fail;
19  endaction
```

RowBuffer - Überlegungen

- Ungültige Werte einfach durchschieben
  - Einfachere Kontrolllogik
- Zwei Bildreihen zwischenspeichern
  - Dritte Reihe nach filtern nicht mehr benötigt

RowBuffer - Implementierung 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface RowBufferServer;
2      interface Put#(Vector#(2, Maybe#(GrayScale))) request;
3      interface Get#(Vector#(2, Maybe#(GrayScale))) response;
4  endinterface: RowBufferServer
5
6  module mkRowBuffer(RowBufferServer);
7      FIFO#(Vector#(2, Maybe#(GrayScale))) inputValue <- mkBypassFIFO;
8      FIFO#(Vector#(2, Maybe#(GrayScale))) outputValue <- mkBypassFIFO;
9
10     BufferServer buffer[2];
11     for(Integer i = 0; i < 2; i = i + 1)
12         buffer[i] <- mkBuffer;
13
14     // Rules on next slide...
15
16     interface Put request = toPut(inputValue);
17     interface Get response = toGet(outputValue);
18 endmodule
```

RowBuffer - Implementierung 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   rule drainInput;
2       let value = inputValue.first;
3       inputValue.deq;
4
5       for(Integer i = 0; i < 2; i = i + 1)
6           buffer[i].request.put(value[i]);
7   endrule
8
9   rule fillOutput;
10      Vector#(2, Maybe#(GrayScale)) vec;
11      for(Integer i = 0; i < 2; i = i + 1) begin
12          let t <- buffer[i].response.get;
13          vec[i] = t;
14      end
15
16      outputValue.enq(vec);
17  endrule
```

Fragen zur Übung