Architekturen und Entwurf von Rechnersystemen Wintersemester 2016/2017

TECHNISCHE UNIVERSITÄT DARMSTADT

Hörsaalübung 4:

Übungsbesprechung 3 und 4
Übungsvorstellung 5 und 6

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Übung 3

Übung 3: BlueCheck

TECHNISCHE UNIVERSITÄT DARMSTADT

- Automatisiertes Testen
- Basiert auf QuickCheck

3.1 Testen gegen Bedingungen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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
```

- Beispiel um Tests für Multiplikation und Division erweitern

3.1 Testen gegen Bedingungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kann beliebig erweitert werden.

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

Die FIFO

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eine FIFO mit 16 Elementen entwickeln.
- Wird später gegen Referenzimplementierung getestet.
- Leicht anderes Interface im Vergleich zu AzureIP FIFO.
- Ringspeicher aus `Vector` zur Speicherung der Elemente.

```
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
```

Die FIFO

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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
```

Die FIFO testen

TECHNISCHE UNIVERSITÄT DARMSTADT

- BlueCheck erlaubt es die Implementierung mit einer Referenzimplementierung zu vergleichen.
- In diesem Fall: AzureIP FIFO wird als funktionierend angenommen.

```
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
```

Die FIFO testen: Hinweis

TECHNISCHE UNIVERSITÄT DARMSTADT

- BlueCheck würde keinen Fehler finden wenn `spec` weniger (oder mehr) als 16 Elemente enthielte!
- Warum?

Die FIFO testen: Hinweis

TECHNISCHE UNIVERSITÄT DARMSTADT

- BlueCheck würde keinen Fehler finden wenn `spec` weniger (oder mehr) als 16 Elemente enthielte!
- Warum?
- BlueCheck benutzt nur Methoden die bei beiden Implementation aufgerufen werden können.

Übung 4: Einfacher Streambasierter Bildfilter

TECHNISCHE UNIVERSITÄT DARMSTADT

- RGB Farbwerte zu Graustufen umwandeln.
- Berechnung als Fixed-Point mit `(U)Q8.8` Präzision.

$$Y = 0.299R + 0.587G + 0.114B$$

- Umrechnung nach 8 Bit Nachkommastellen: `floor(n * (1 << 8))`

$$R = 76$$
$$G = 150$$
$$B = 29$$

- Multiplikation: $(a \times b) >> 8$

Übung 4: Typen und Modul

TECHNISCHE UNIVERSITÄT DARMSTADT

Datentypen:

```
1    typedef Bit#(8) Color;
2    typedef Bit#(8) GrayScale;
3
4    typedef struct {
5        Color r;
6        Color g;
7        Color b;
8        } RGB deriving(Bits, Eq, FShow);
```

Modul:

```
1    module mkGray(Server#(RGB, GrayScale));
2            FIFO#(GrayScale) outputValue <- mkFIFO;
3            FIFO#(RGB) inputValue <- mkFIFO;
4    ...
5            interface Put request = toPut(inputValue);
6            interface Get response = toGet(outputValue);
7    endmodule
```

Übung 4: Datenverarbeitung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    rule calc;
2      let color = inputValue.first; inputValue.deq;
3      let gray = toGray(color);
4
5      outputValue.enq(gray);
6    endrule
```

Übung 4: Berechnung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   function GrayScale toGray(RGB rgb);
2       UInt#(16) r = extend(unpack(rgb.r)) << 8;
3       UInt#(16) g = extend(unpack(rgb.g)) << 8;
4       UInt#(16) b = extend(unpack(rgb.b)) << 8;
5
6       UInt#(16) factorR = 76;  // floor(0.299 * (1 << 8))
7       UInt#(16) factorG = 150; // floor(0.587 * (1 << 8))
8       UInt#(16) factorB = 29;  // floor(0.114 * (1 << 8))
9       // Multiply color with factors and normalize
10      // Enough space for multiplcation result: UInt#(24)
11      UInt#(24) foo = extend(r) * extend(factorR);
12      r = truncate(foo >> 8);
13      foo = extend(g) * extend(factorG);
14      g = truncate(foo >> 8);
15      foo = extend(b) * extend(factorB);
16      b = truncate(foo >> 8);
17      UInt#(16) scale = r + g + b;
18      return pack(scale)[15:8];
19  endfunction
```

Übung 4: Testbench

TECHNISCHE UNIVERSITÄT DARMSTADT

- Konzentration auf BlueCheck Testbench

```
1    // Very simple/inefficient implementation
2    module mkSimpleGray(Server#(RGB, GrayScale));
3        FIFO#(GrayScale) outputValue <- mkFIFO;
4
5        interface Put request;
6            method Action put(RGB rgb);
7                UInt#(32) factorR = 76;  // floor(0.299 * (1 << 8))
8                UInt#(32) factorG = 150; // floor(0.587 * (1 << 8))
9                UInt#(32) factorB = 29;  // floor(0.114 * (1 << 8))
10               UInt#(32) r = extend(unpack(rgb.r)) << 8;
11               UInt#(32) g = extend(unpack(rgb.g)) << 8;
12               UInt#(32) b = extend(unpack(rgb.b)) << 8;
13               outputValue.enq(truncate(pack((r * factorR + g * factorG + b *
     ↪   factorB) >> 16)));
14           endmethod
15       endinterface
16       interface Get response = toGet(outputValue);
17   endmodule
```

Übung 4: Testbench

TECHNISCHE UNIVERSITÄT DARMSTADT

- Test gegen ineffiziente Implementierung
- Finden von Fehlern beim Runden/Überläufe

```
1    module [BlueCheck] mkColorConverterSpec ();
2        Server#(RGB, GrayScale) spec <- mkSimpleGray;
3        Server#(RGB, GrayScale) impl <- mkGray();
4
5        equiv("put", spec.request.put, impl.request.put);
6        equiv("get", spec.response.get, impl.response.get);
7    endmodule
8
9    module [Module] mkColorConverterChecker ();
10       blueCheck(mkColorConverterSpec);
11   endmodule
```

Ausblick Übung 5: Median-Filter

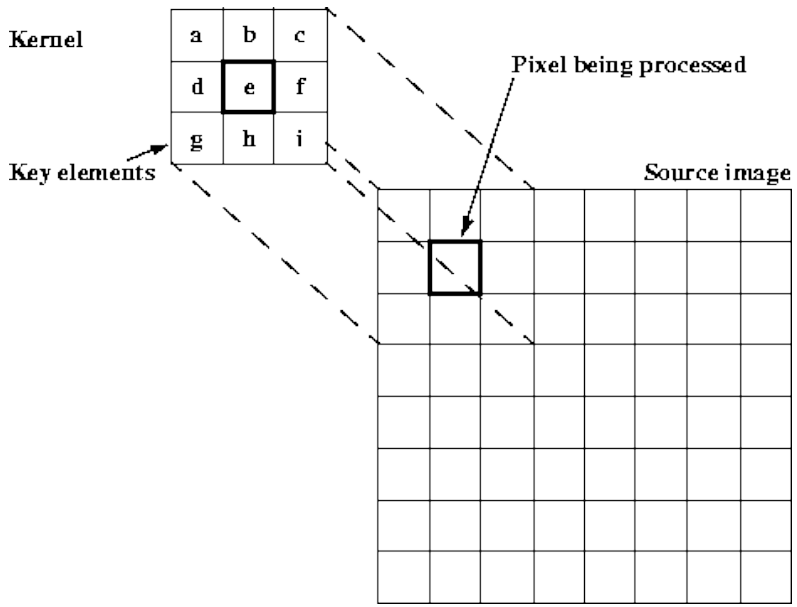
TECHNISCHE UNIVERSITÄT DARMSTADT

- Streambasierter Median-Filter
- Wähle den Median aus umliegenden Pixeln
- Arbeitet (in unserem Fall) auf Graustufen

Ausblick Übung 5: Kernel

TECHNISCHE UNIVERSITÄT DARMSTADT

- Arbeitet auf 3 $\times$ 3 großer Region um mittleren Pixel
- Median bilden aus neun Werten
- Zwei Reihen des Eingangsbildes müssen gebuffered werden

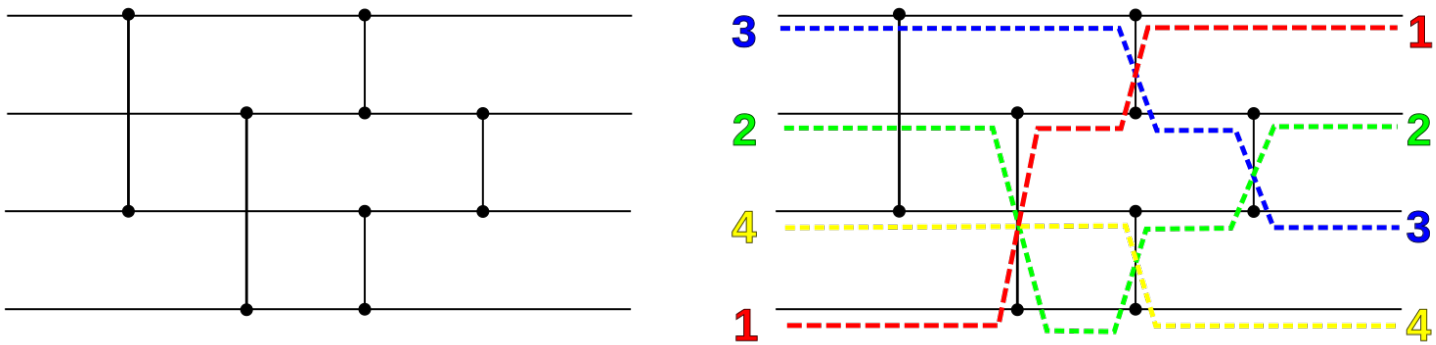


http://www2.hs-fulda.de/caelabor/inhalte/java/j3d/j3d_seminar/19/JAI%20Guide%20von%20Sun/Image-enhance.doc.html

Ausblick Übung 5: Median finden in Hardware

TECHNISCHE UNIVERSITÄT DARMSTADT

- Median kann mit Hilfe eines Sortiernetzwerks gefunden werden
- Nicht alle Elemente des Sortiernetzwerks notwendig
- Nur mittlerer Wert interessant
- Erweitert: Ränder korrekt behandeln

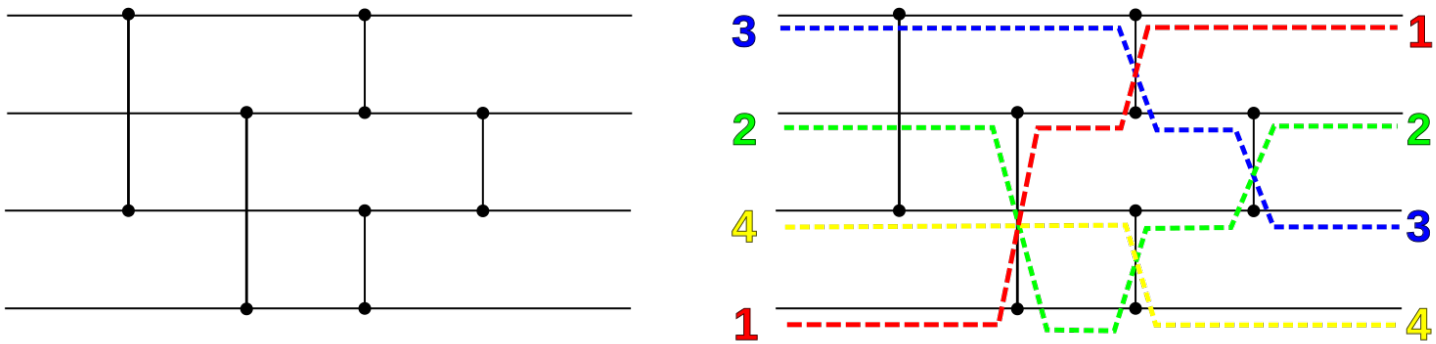


`https://en.wikipedia.org/wiki/Sorting_network#/media/File:SimpleSortingNetworkFullOperation.svg`

Ausblick Übung 5: Median finden in Hardware

TECHNISCHE UNIVERSITÄT DARMSTADT

- Median kann mit Hilfe eines Sortiernetzwerks gefunden werden
- Nicht alle Elemente des Sortiernetzwerks notwendig
- Nur mittlerer Wert interessant

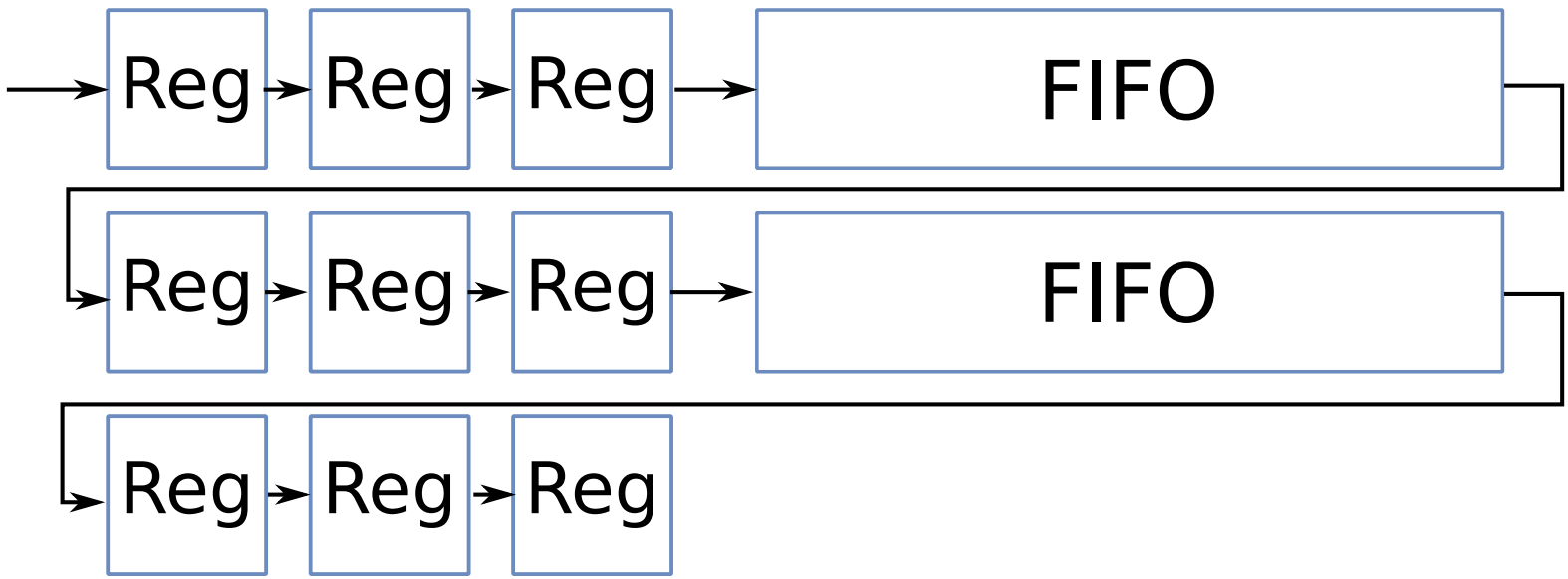


`https://en.wikipedia.org/wiki/Sorting_network#/media/File:SimpleSortingNetworkFullOperation.svg`

Ausblick Übung 5: Eingangsreihen Buffern

TECHNISCHE UNIVERSITÄT DARMSTADT

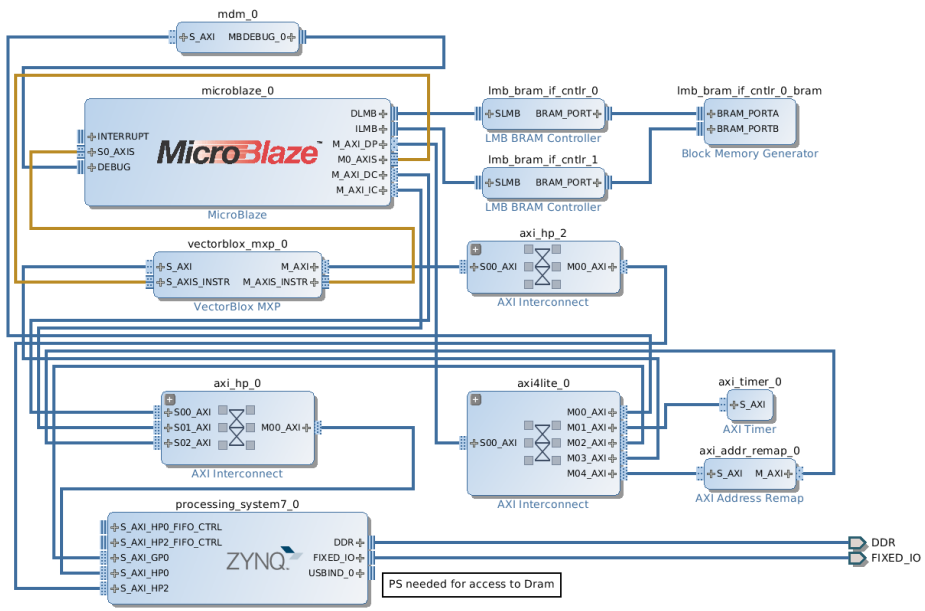
- Zwei komplette Reihen des Bildes müssen vorgehalten werden
- Effizient mit FIFO und neun Registern zu realisieren



Ausblick Übung 6: ESA-Bus

TECHNISCHE UNIVERSITÄT DARMSTADT

- System-on-Chip (SoC) benötigen effiziente Kommunikationsmöglichkeit
- System hat mehrere Master und Slave Kommunikationspartner
- Typisch: Memory-Mapped-Bus
- Kommunikation wird anhand von Adressen geroutet



`http://vectorblox.github.io/mxp/images/vivado_microblaze_no_clocks.png`

Ausblick Übung 6: ESA-Bus

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eigener einfacher Bus in Bluespec
- Getrennte Lese- und Schreibrichtungen
- Handshake
- Nur die nötigsten Signale

Ausblick Übung 6: ESA-Bus

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eigener einfacher Bus in Bluespec
- Getrennte Lese- und Schreibrichtungen
- Handshake
- Nur die nötigsten Signale

Signal	Beschreibung
raready	Slave kann Leseanfrage annehmen.
ravalid	Master sendet gültige Daten.
raaddr	Leseadresse
rrready	Master kann Antwort annehmen.
rrvalid	Slave möchte Antwort senden.
rrdata	Gelesene Daten.

Tabelle: Signale der Leserichtung

Ausblick Übung 6: ESA-Bus

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eigener einfacher Bus in Bluespec
- Getrennte Lese- und Schreibrichtungen
- Handshake
- Nur die nötigsten Signale

Signal	Beschreibung
wready	Slave kann Schreibanfrage annehmen.
wvalid	Master sendet gültige Daten.
waddr	Schreibadresse
wdata	Schreibdaten

Tabelle: Signale der Schreibrichtung

Ausblick Übung 6: Master und Slave verbinden

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bluespec hat standardisierte Typklasse für Verbindungen
- `Connectable`
- z.B. `mkConnection(foo.get, bar.put)`

```
1    typeclass Connectable#(type a, type b);
2            module mkConnection#(a x1, b x2)(Empty);
3    endtypeclass
```

Ausblick Übung 6: Eigene Instanz

TECHNISCHE UNIVERSITÄT DARMSTADT

- Eigene Instanzen der Typklasse sind leicht zu erstellen
- Beispiel AXI4-Lite:

```
1   instance Connectable#(AXI4_Lite_Master_Wr_Fab#(addrwidth, datawidth),
    ↪   AXI4_Lite_Slave_Wr_Fab#(addrwidth, datawidth));
2           module mkConnection#(AXI4_Lite_Master_Wr_Fab#(addrwidth,
    ↪   datawidth) master, AXI4_Lite_Slave_Wr_Fab#(addrwidth, datawidth)
    ↪   slave)(Empty);
3                   rule forward1; master.pawready(slave.awready); endrule
4                   rule forward2; slave.pawvalid(master.awvalid); endrule
5                   rule forward3; slave.pawaddr(master.awaddr); endrule
6                   rule forward4; slave.pawprot(master.awprot); endrule
7                   rule forward5; master.pwready(slave.wready); endrule
8                   rule forward6; slave.pwvalid(master.wvalid); endrule
9                   rule forward7; slave.pwdata(master.wdata); endrule
10                  rule forward8; slave.pwstrb(master.wstrb); endrule
11                  rule forward9; master.pbvalid(slave.bvalid); endrule
12                  rule forward10; slave.pbready(master.bready); endrule
13                  rule forward11; master.pbresp(slave.bresp); endrule
14          endmodule
15  endinstance
```