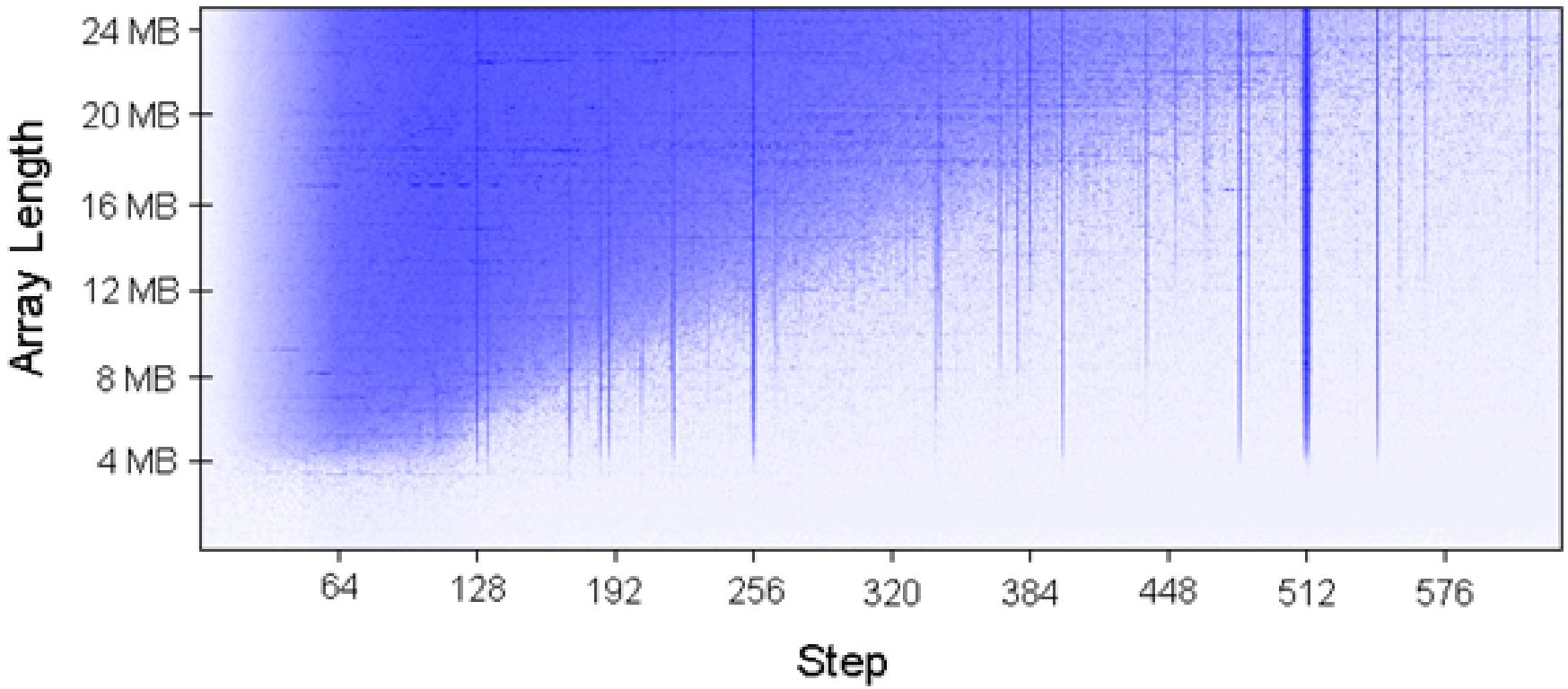
Einführung in Computer Microsystems
Sommersemester 2015
Hörsaalübung 4: Caches + HDMI

TECHNISCHE UNIVERSITÄT DARMSTADT



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Hörsaalübung 4

Hörsaalübung 4

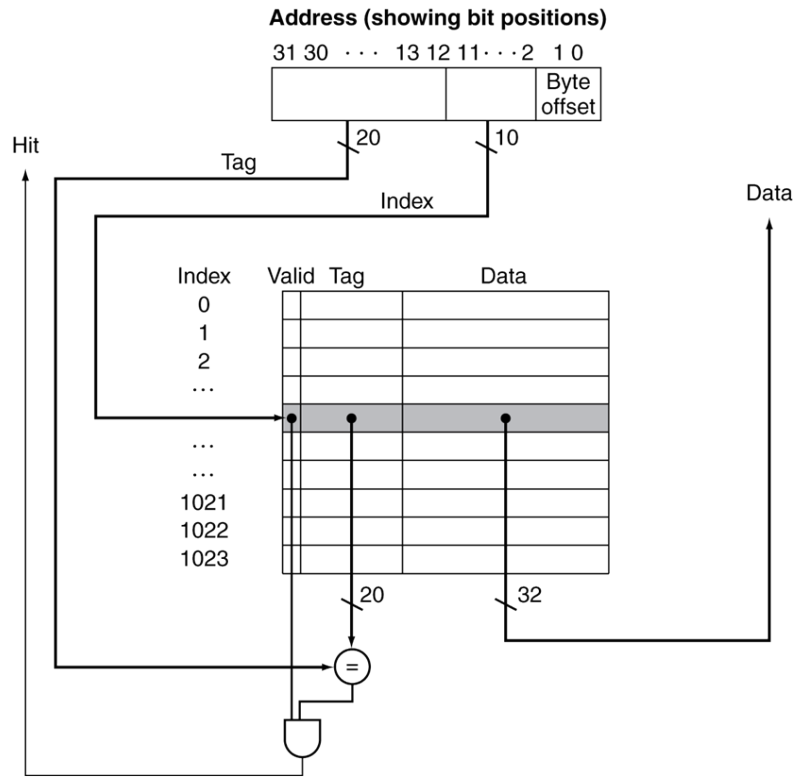
TECHNISCHE UNIVERSITÄT DARMSTADT

- Vorstellung des Caches für Übungsblatt 4
- Design eines HDMI Controllers $\rightarrow$ Von BSV zu FPGA

Cache

TECHNISCHE UNIVERSITÄT DARMSTADT

- Design eines Direct-Mapped-Cache in Bluespec
- ClientServer Interface
- Cache-Line Breite 512 bit
- Aufgeteilt in zwei Module
  - Der eigentliche Cache
  - Fetcher für Cache-Lines aus überliegendem Speicher



Cache-Line Fetcher: Typen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   typedef Client#(RAMRequest#(wordBits, cacheLineBits), Bit#(cacheLineBits))
2           RAMClient#(numeric type wordBits, numeric type cacheLineBits);
```

- Verbindung mit überliegendem RAM (dieser bietet `RAMServer`)

```
1   typedef union tagged {
2     Bit#(32) Fetch;
3     Tuple3#(Bit#(32), Bit#(32), Bit#(cacheLineBits)) FetchDirty;
4     Tuple3#(Bit#(32), Bit#(cacheLineBits), Bit#(TDiv#(cacheLineBits,8)))
5                                                          JustWrite;
6   } FetchRequest#(numeric type cacheLineBits) deriving (Bits, Eq, FShow);
```

- Definiert Operationen von Fetcher Modul
- `Fetch`: Lade neue Cache-Line aus Speicher
- `FetchDirty`: Schreibe Cache-Line und lade neue Cache-Line aus Speicher
- `JustWrite`: Schreibe Cache-Line aber hole keine Neue

Cache-Line Fetcher: Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    interface FetchCacheLineWhole#(numeric type wordBits,
2                                   numeric type cacheLineBits);
3      interface RAMClient#(wordBits, cacheLineBits)   toRAM;
4      interface Server#(FetchRequest#(cacheLineBits),
5                        Bit#(cacheLineBits)) toUser;
6    endinterface
```

- `FetchRequest` von Cache
- RAM Anfragen an überliegenden Speicher

Cache-Line Fetcher: Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    interface Client toRAM;
2      interface Get request  = toGet(requestOut);
3      interface Put response = toPut(dataIn);
4    endinterface
```

- Interface zu überliegendem Speicher
- Als BypassFIFOs implementiert
- Für Hardware: Vorsicht! Kritischer Pfad!

```
1    FIFO#(RAMRequest#(wordBits, cacheLineBits)) requestOut <- mkBypassFIFO();
2    FIFO#(Bit#(cacheLineBits))                  dataIn     <- mkBypassFIFO();
```

Cache-Line Fetcher: Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    interface Server toUser;
2      interface Put request;
3          method Action put(FetchRequest#(cacheLineBits) req)
4                                                  if(!fetchInProgress);
5              case(req) matches
6                  tagged FetchDirty .v: begin
7                      dirty <= True;
8                      ...
9                  end
10                 tagged Fetch .v: begin
11                     dirty <= False;
12                     ...
13                 end
14                 tagged JustWrite .v: begin
15                     $display("Not handled!");
16                     ...
17                 end
18             endcase
```

- Anfragen von User (Cache)
- Setze Statusvariablen und starte Fetch Prozess

Cache-Line Fetcher: Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    interface Get response;
2        method ActionValue#(Bit#(cacheLineBits)) get() if(!fetchInProgress);
3            return currentCacheLine;
4        endmethod
5    endinterface
```

- Gebe Cache-Line zurück wenn Sie gültig ist

Cache-Line Fetcher: Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   rule operation (fetchInProgress && !requestDone);
2       RAMRequest#(wordBits, cacheLineBits) req = tagged Read 0;
3       if(dirty) begin
4           Bit#(cacheLineBytes) writeEN = (1 << valueOf(cacheLineBytes)) - 1;
5           req = tagged Write tuple3(writeAddress, currentCacheLine, writeEN);
6       end else begin
7           req = tagged Read readAddress;
8       end
9       requestDone <= True;
10      requestOut.enq(req);
11  endrule
```

- Wandelt `FetcherRequest` in `RAMRequest` um
- Schreibt oder liest Cache-Line abhängig von `dirty` (`Reg#(Bool)`)

Cache-Line Fetcher: Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    rule switchOperation (fetchInProgress && dirty && requestDone);
2        dirty           <= False;
3        requestDone     <= False;
4    endrule
```

- Sorgt dafür das `rule operation` Cache-Line liest nach dem Schreiben

Cache-Line Fetcher: Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    rule operationRead (fetchInProgress && !dirty && requestDone);
2        let result <- toGet(dataIn).get();
3        currentCacheLine <= result;
4        fetchInProgress <= False;
5    endrule
```

- Leserergebnis von überliegendem Speicher wird Zwischengespeichert
- Kann über Interface von User abgerufen werden
- Fetch abgeschlossen

Cache

TECHNISCHE UNIVERSITÄT DARMSTADT

- Vergleiche Tag in Slot mit Anfrage
- Bei Hit: Gebe Cache-Line zurück
- Bei Miss: Nutze Fetcher um Cache-Line zu holen

Cache: Typen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   typedef struct {
2       Reg#(Bool) dirty;
3       Reg#(Bool) valid;
4       Reg#(Bit#(tagAddrBits)) tag;
5       Reg#(Bit#(cacheLineBits)) cacheLine;
6   } CacheLine#(numeric type tagAddrBits, numeric type cacheLineBits);
```

- Typ der eine Cache-Line abbildet
- Enthält einzelne Register
- Ermöglicht gezieltes schreiben von einzelnen Registern
  - `Reg#(CacheLine)`: Immer alle Register müssen geschrieben werden

```
1   let r = cacheLine;
2   r.dirty = True;
3   cacheLine <= r;
```

  - Hier: `cachLine.dirty <= True` möglich

Cache: Typen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    Vector#(slots, CacheLine#(tagAddrBits, cacheLineBits)) cacheLines;
2    for(Integer i = 0; i < valueOf(slots); i = i + 1) begin
3        cacheLines[i].dirty <- mkReg(False);
4        cacheLines[i].valid <- mkReg(False);
5        cacheLines[i].cacheLine <- mkRegU;
6        cacheLines[i].tag <- mkRegU;
7    end
```

- Initialisierung mit Hilfe von `for`
- Parallel nicht Sequentiell!

Cache: Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    interface Cache#(numeric type wordBits, numeric type cacheLineBits,
2                                            numeric type slots);
3        interface RAMClient#(wordBits, cacheLineBits) toRAM;
4        interface RAMServer#(wordBits, cacheLineBits) toUser;
5        method    Action clearCache();
6    endinterface
```

- Neuer Parameter: `slots`: Anzahl von Slots in Cache
- `toRAM` und `toUser` bekannt
- `clearCache` für Simulation: Alle Cache-Lines sind `valid == False` und `dirty == False`

Cache: Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    interface Client toRAM;
2        interface Get request  = toGet(requestOut);
3        interface Put response = toPut(dataIn);
4    endinterface
5    interface Server toUser;
6        interface Put request  = toPut(requestIn);
7        interface Get response = toGet(dataOut);
8    endinterface
```

- Über FIFO implementiert
- Diesmal keine `BypassFIFO`

```
1    FIFO#(RAMRequest#(wordBits, cacheLineBits))  requestIn <- mkFIFO();
2    FIFO#(Bit#(cacheLineBits))                     dataOut <- mkFIFO();
3
4    FIFO#(RAMRequest#(wordBits, cacheLineBits)) requestOut <- mkFIFO();
5    FIFO#(Bit#(cacheLineBits))                      dataIn <- mkFIFO();
```

Cache: Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    method Action clearCache() if(!actionInProgress);
2        for(Integer i = 0; i < valueOf(slots); i = i + 1) begin
3            cacheLines[i].dirty <= False;
4            cacheLines[i].valid <= False;
5        end
6    endmethod
```

- Paralleles Zurücksetzen aller Cache-Lines

Cache: Hilfsfunktionen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    function Bit#(tagAddrBits) getTag(Bit#(32) addr);
2        return addr[31:valueOf(addrBitsNoTag)];
3    endfunction
```

- Extrahiere `Tag` aus kompletter Addresse

```
1    function Bit#(slotsAddrBits) getSlot(Bit#(32) addr);
2        return addr[valueOf(addrBitsNoTag) - 1:valueOf(cacheLineAddrBits)];
3    endfunction
```

- Extrahiere `Slot` aus kompletter Addresse

```
1    function Bit#(32) toAddr(Bit#(tagAddrBits) tag, Bit#(slotsAddrBits) slot);
2        return {tag,slot,0};
3    endfunction
```

- Baue Addresse aus `Tag` und `Slot`

Cache: Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   rule handleRead (requestIn.first() matches tagged Read .v
2                    &&& !actionInProgress);
3       expectedTag                <= getTag(v);
4       expectedSlot               <= getSlot(v);
5       read                       <= True;
6       actionInProgress           <= True;
7   endrule
8
9   rule handleWriteRequest (requestIn.first() matches tagged Write .v
10                           &&& !actionInProgress);
11      ... Extract data from tuple v
12      expectedTag                <= getTag(address);
13      expectedSlot               <= getSlot(address);
14      currentWriteCacheLine      <= unpack(data);
15      currentWriteEN             <= unpack(writeen);
16      read                       <= False;
17      actionInProgress           <= True;
18  endrule
```

- Nutzeranfrage entgegennehmen und interne Variablen setzen

Cache: Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    rule fetchCorrectLine(actionInProgress
2                       && (expectedTag != cacheLines[expectedSlot].tag
3                           || !cacheLines[expectedSlot].valid)
4                       && !fetchActive);
5        ... Some statistics (Cycles for Cache miss etc)
6        FetchRequest#(cacheLineBits) req
7             = tagged Fetch toAddr(cacheLines[expectedSlot].tag, expectedSlot);
8        if(cacheLines[expectedSlot].dirty) begin
9            req = tagged FetchDirty
10                      tuple3(toAddr(expectedTag, expectedSlot),
11                             toAddr(cacheLines[expectedSlot].tag, expectedSlot),
12                             cacheLines[expectedSlot].cacheLine);
13       end
14       cacheLines[expectedSlot].valid                <= False;
15       cacheLines[expectedSlot].dirty                <= False;
16       cacheLines[expectedSlot].tag                  <= expectedTag;
17       fetchActive             <= True;
18       fetcher.toUser.request.put(req);
19   endrule
```

- Wenn Cache-Miss auftritt Fetcher aktivieren

Cache: Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    rule handleFetcherResponse (actionInProgress && fetchActive);
2        let fetchedLine  <- fetcher.toUser.response.get();
3        cacheLines[expectedSlot].valid <= True;
4        cacheLines[expectedSlot].cacheLine <= fetchedLine;
5        fetchActive      <= False;
6    endrule
```

- Ergebnis von Fetcher lesen und in Cache-Line-Vector speichern

Cache: Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    rule writeResponse(!fetchActive
2                       && cacheLines[expectedSlot].tag == expectedTag
3                       && cacheLines[expectedSlot].valid
4                       && actionInProgress);
5        if(read) begin
6            dataOut.enq(cacheLines[expectedSlot].cacheLine);
7        end else begin
8          .... Next Slide
9        end
10       actionInProgress <= False;
11   endrule
```

- Cache Hit
- Lesezugriff: Passende Cache-Line über FIFO zurückgeben
- Schreibzugriff: Passende Cache-Line bearbeiten (Nächste Folie)

Cache: Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    Bit#(cacheLineBits) writeENBits = 0;
2    for(Integer i = 0; i < valueOf(cacheLineBytes); i = i + 1) begin
3        for(Integer j = 0; j < 8; j = j + 1) begin
4            writeENBits[(i * 8) + j] = currentWriteEN[i];
5        end
6    end
7    cacheLines[expectedSlot].cacheLine <=
8                          (cacheLines[expectedSlot].cacheLine & ~writeENBits)
9                        | (currentWriteCacheLine & writeENBits);
10   cacheLines[expectedSlot].dirty <= True;
```

- Schreibzugriff
- Erstelle Bit-Enable aus Byte-Enable
- Schreibe Cache-Line mit $(l_c \wedge \neg w_e) \vee (l_w \wedge w_e)$
  - l_c momentane Cache-Line
  - l_w zu schreibende Cache-Line
  - w_e Write-Enable

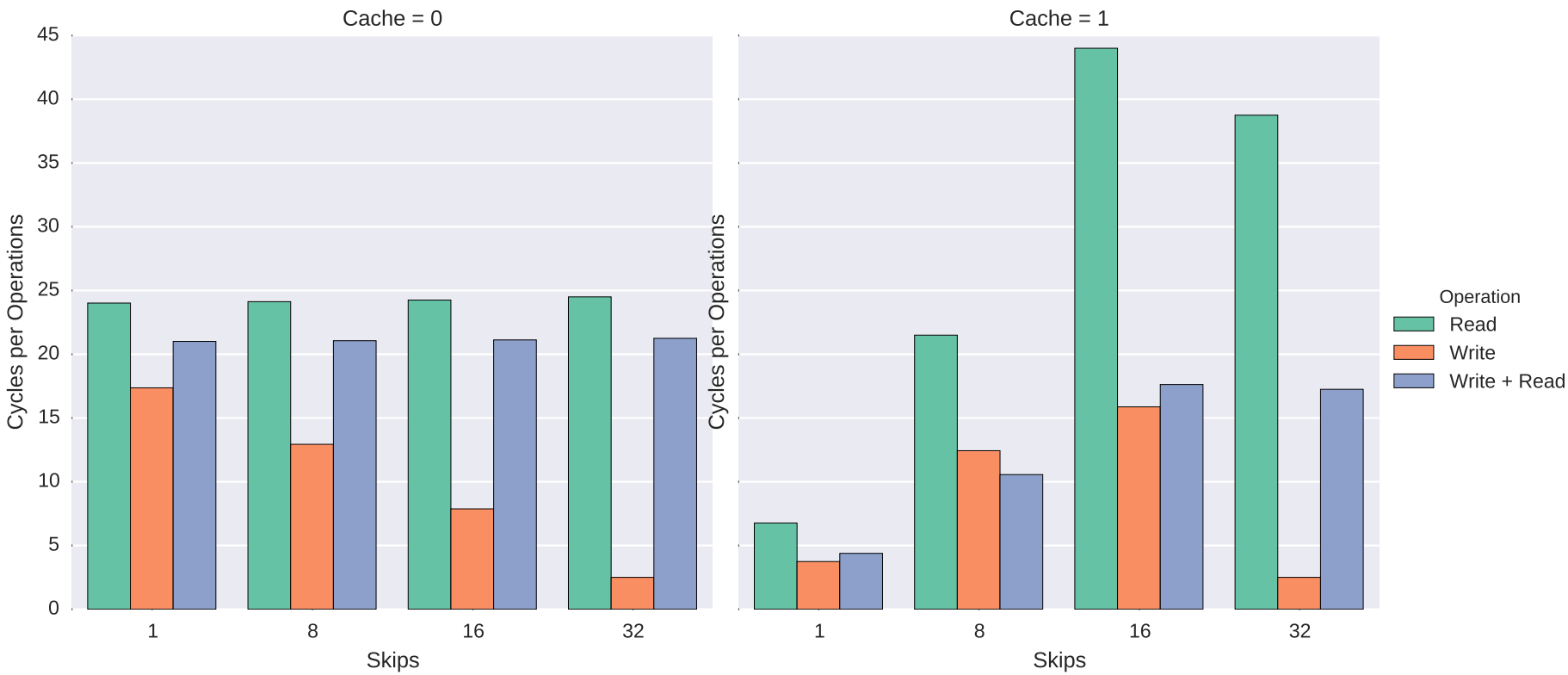
Vergleich Cache/NoCache

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verschiedene Benchmarks möglich
- Hier drei Fälle:
  - Nur Schreiben
  - Nur Lesen
  - Schreiben und dann Lesen
- Jeweils für 128, 256, 512, 1024 und 16384 Wörter
- Jeweils mit 1, 8, 16 und 32 übersprungenen Wörter zwischen den Zugriffen
- Ohne Cache und mit einem Direct-Mapped-Cache (128 Slots)
- Verglichen wird die Anzahl Zyklen pro Speicheroperation (Lesen und Schreiben ist jeweils eine Operation)
- Beliebige weitere Benchmarks denkbar

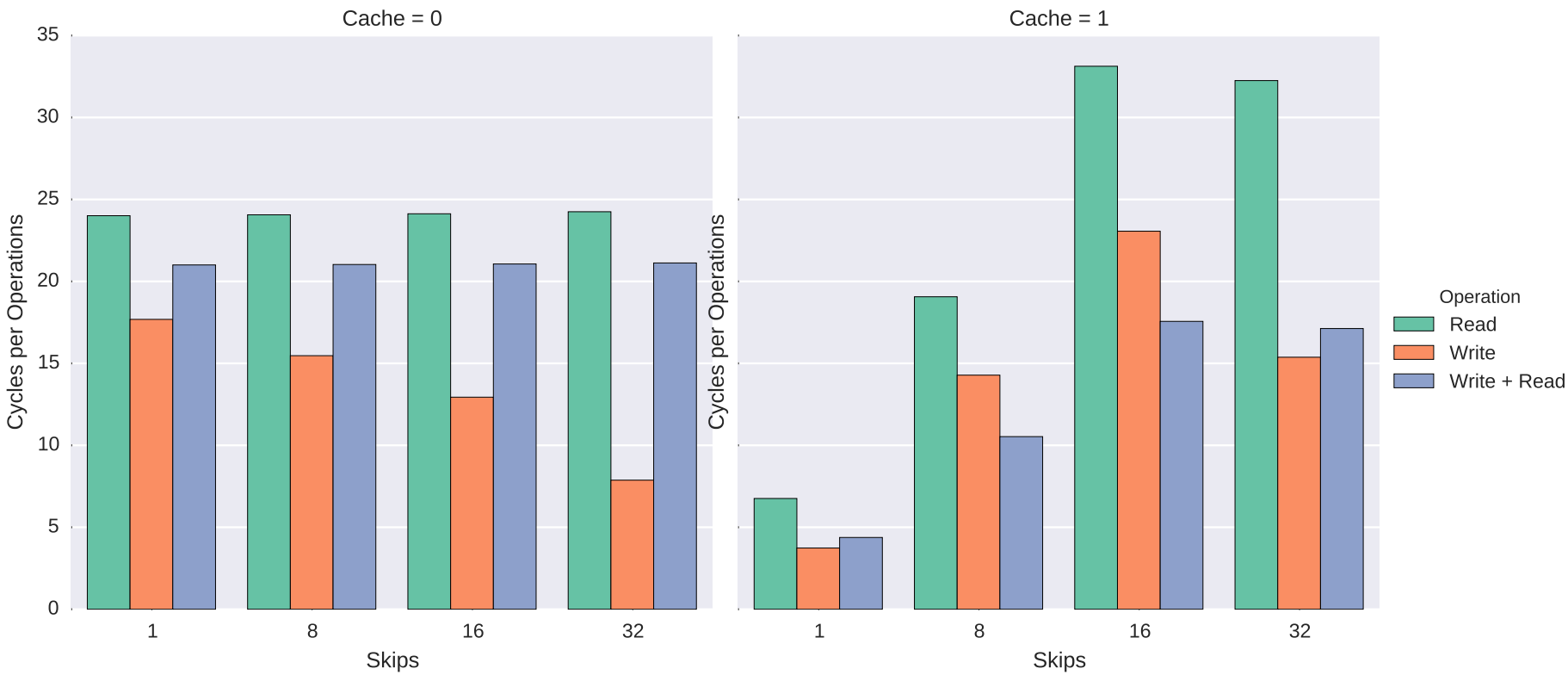
Vergleich Cache/NoCache: 128 Wörter

TECHNISCHE UNIVERSITÄT DARMSTADT



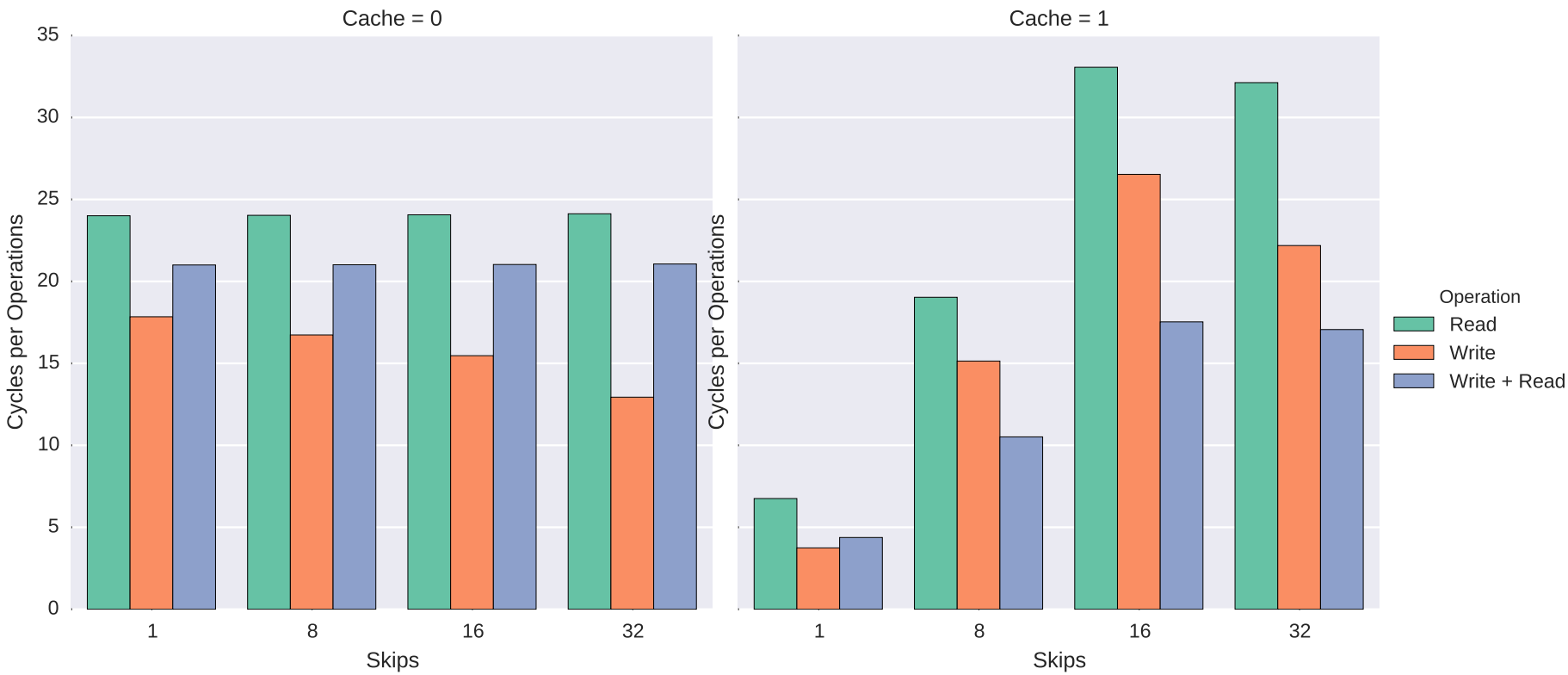
Vergleich Cache/NoCache: 256 Wörter

TECHNISCHE UNIVERSITÄT DARMSTADT



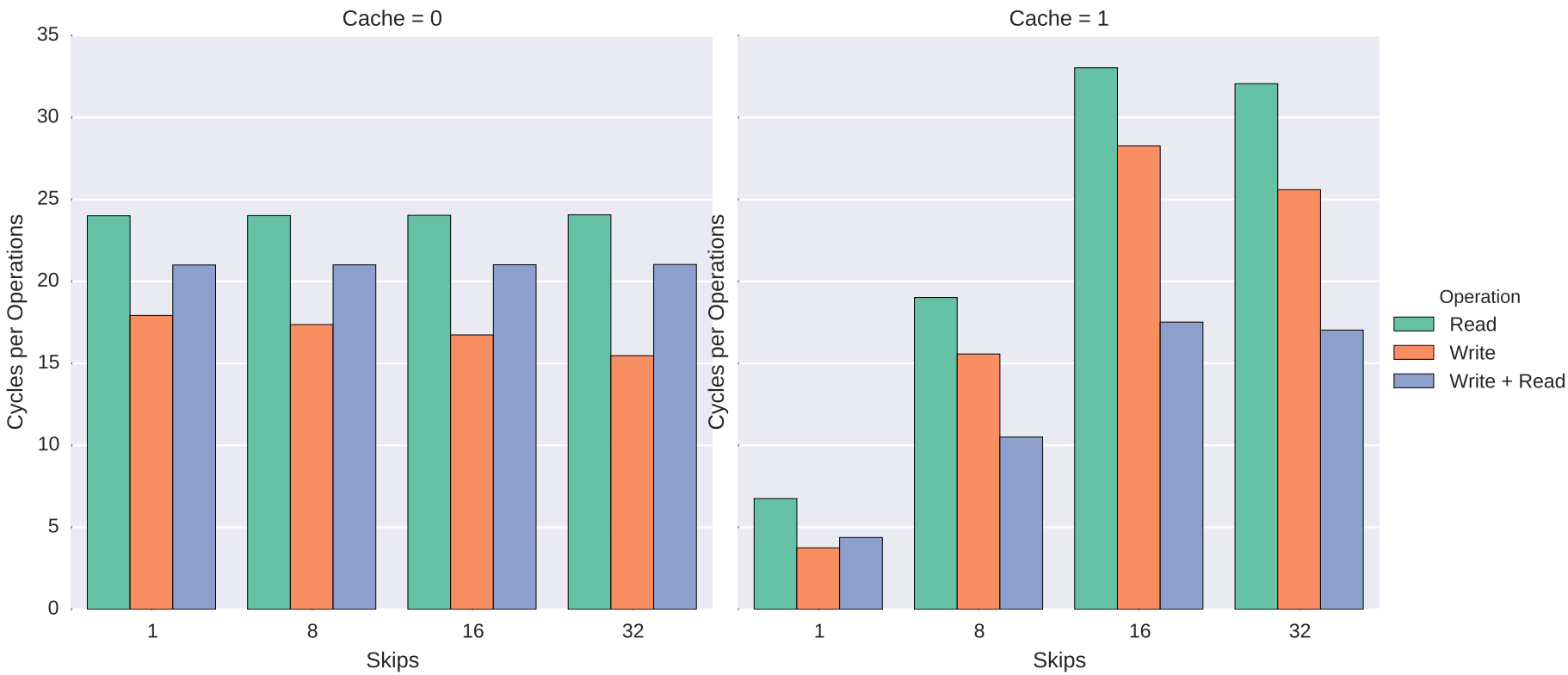
Vergleich Cache/NoCache: 512 Wörter

TECHNISCHE UNIVERSITÄT DARMSTADT



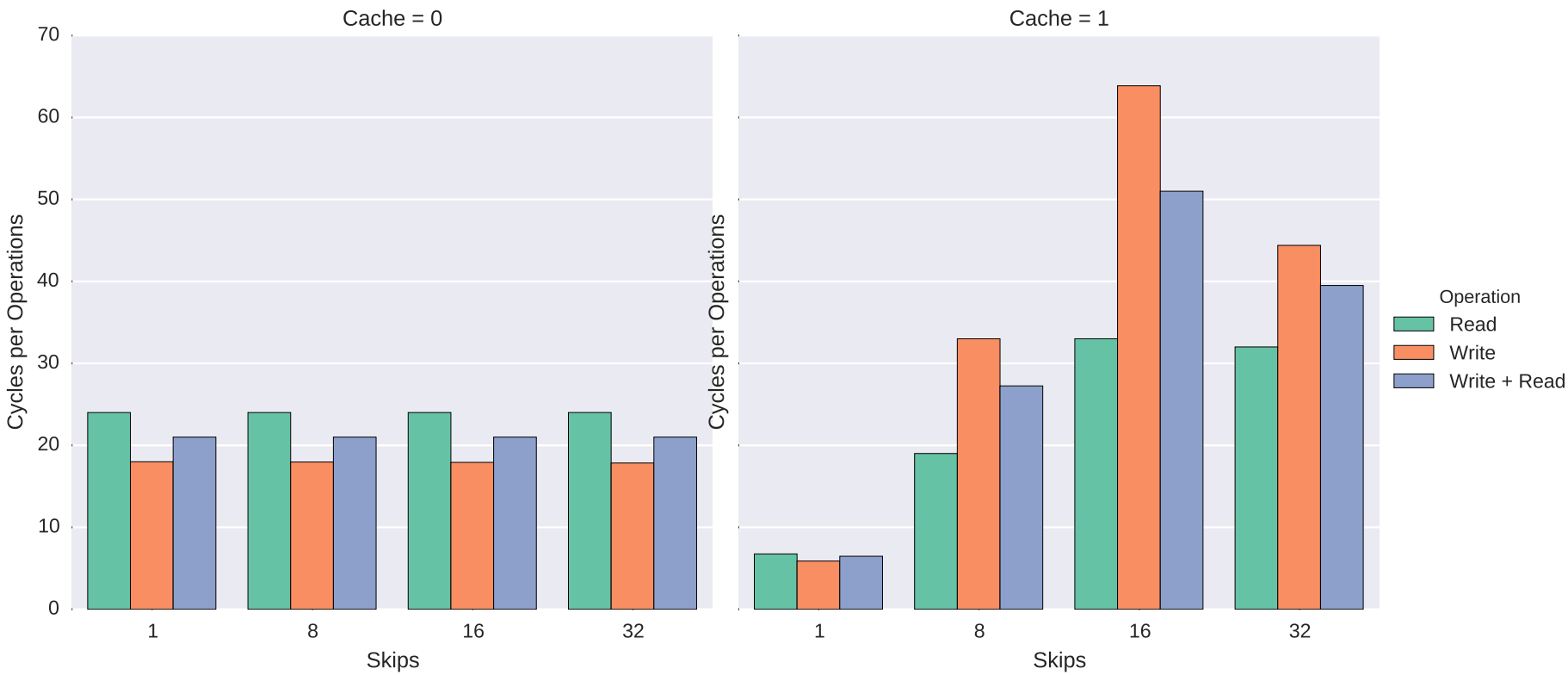
Vergleich Cache/NoCache: 1024 Wörter

TECHNISCHE UNIVERSITÄT DARMSTADT



Vergleich Cache/NoCache: 16384 Wörter

TECHNISCHE UNIVERSITÄT DARMSTADT



Vergleich Cache/NoCache

TECHNISCHE UNIVERSITÄT DARMSTADT

- Mögliche Benchmarks:
  - Matrixmultiplikation
  - Multiply-Add
  - Verhalten bei langen Rechenoperationen

„Real-Life“ Cache Effekte: i7

TECHNISCHE UNIVERSITÄT DARMSTADT

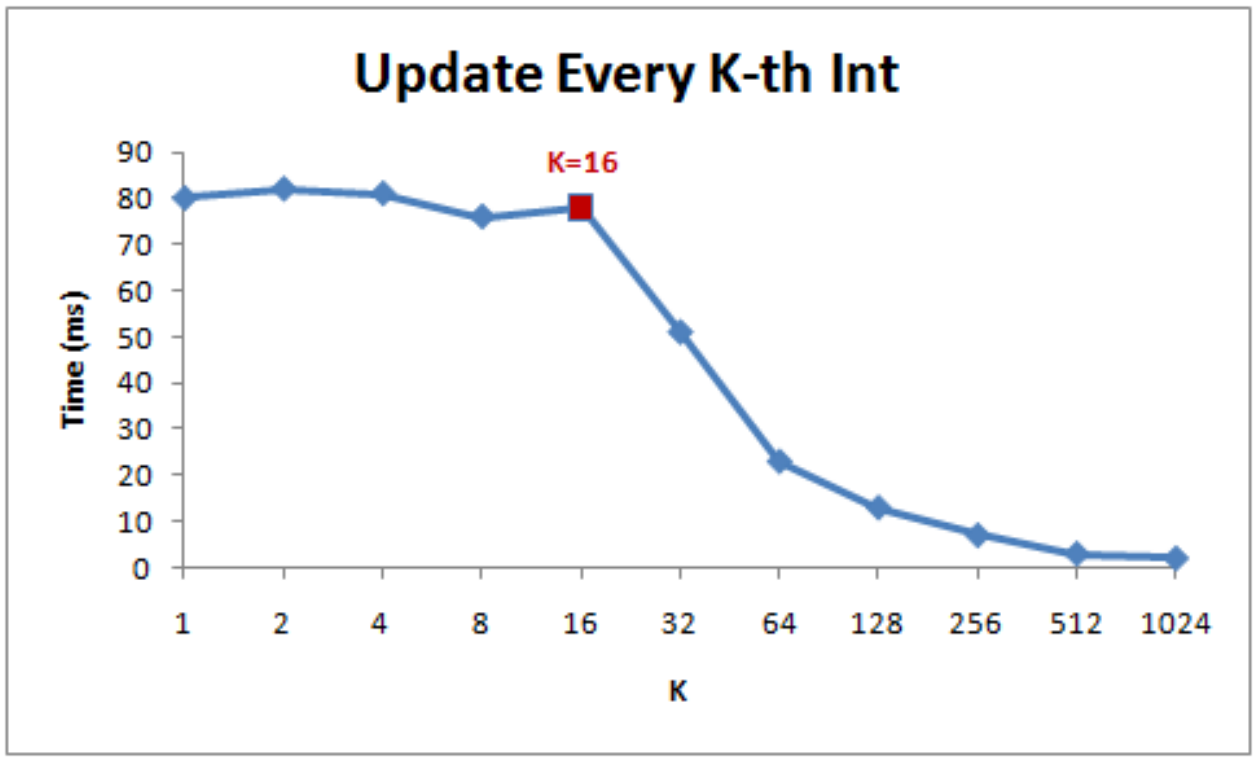
- 2 Cache Level
  - L1: 32KB 8 Fach Assoziativ
  - L2: 4MB 16 Fach Assoziativ
  - 64 Byte pro Cache-Line
- Einfache Programme um Cache Effekte zu zeigen
- Daten und Darstellungen von

  `http://igoro.com/archive/gallery-of-processor-cache-effects/`

„Real-Life“ Cache Effekte: Cache Lines

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    for (int i = 0; i < arr.Length; i += K) arr[i] *= 3;
```

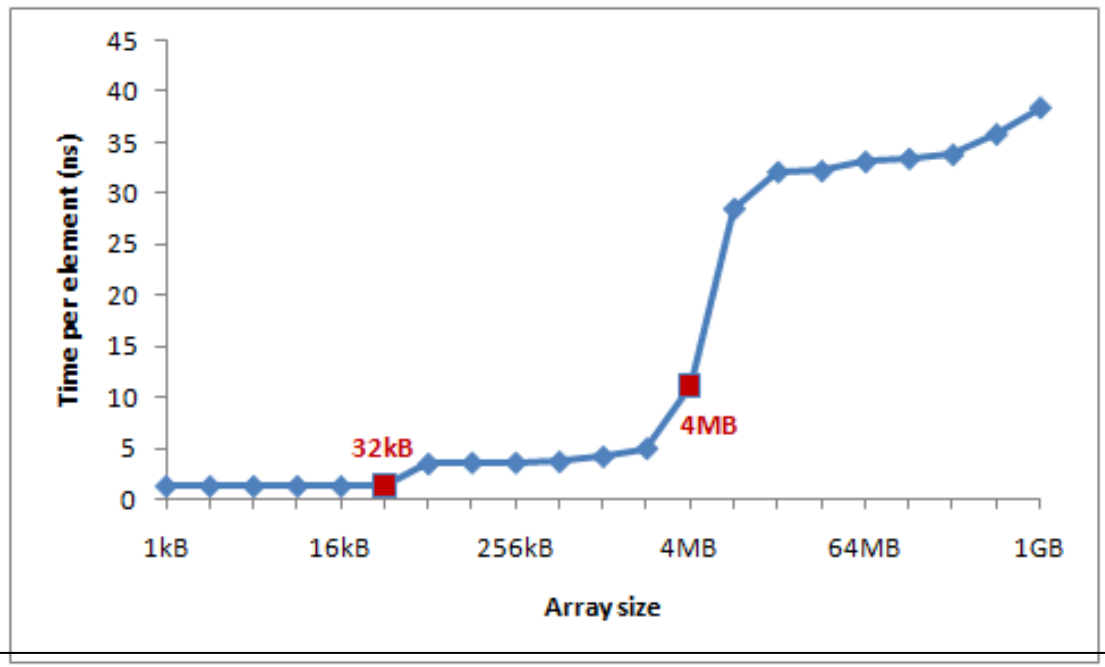


http://igoro.com/archive/gallery-of-processor-cache-effects/

„Real-Life“ Cache Effekte: L1 vs L2 Cache

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    int steps = 64 * 1024 * 1024; // Arbitrary number of steps
2    int lengthMod = arr.Length - 1;
3    for (int i = 0; i < steps; i++)
4    {
5        arr[(i * 16) & lengthMod]++;
6        // (x & lengthMod) is equal to (x % arr.Length)
7    }
```



„Real-Life“ Cache Effekte: Assozivität

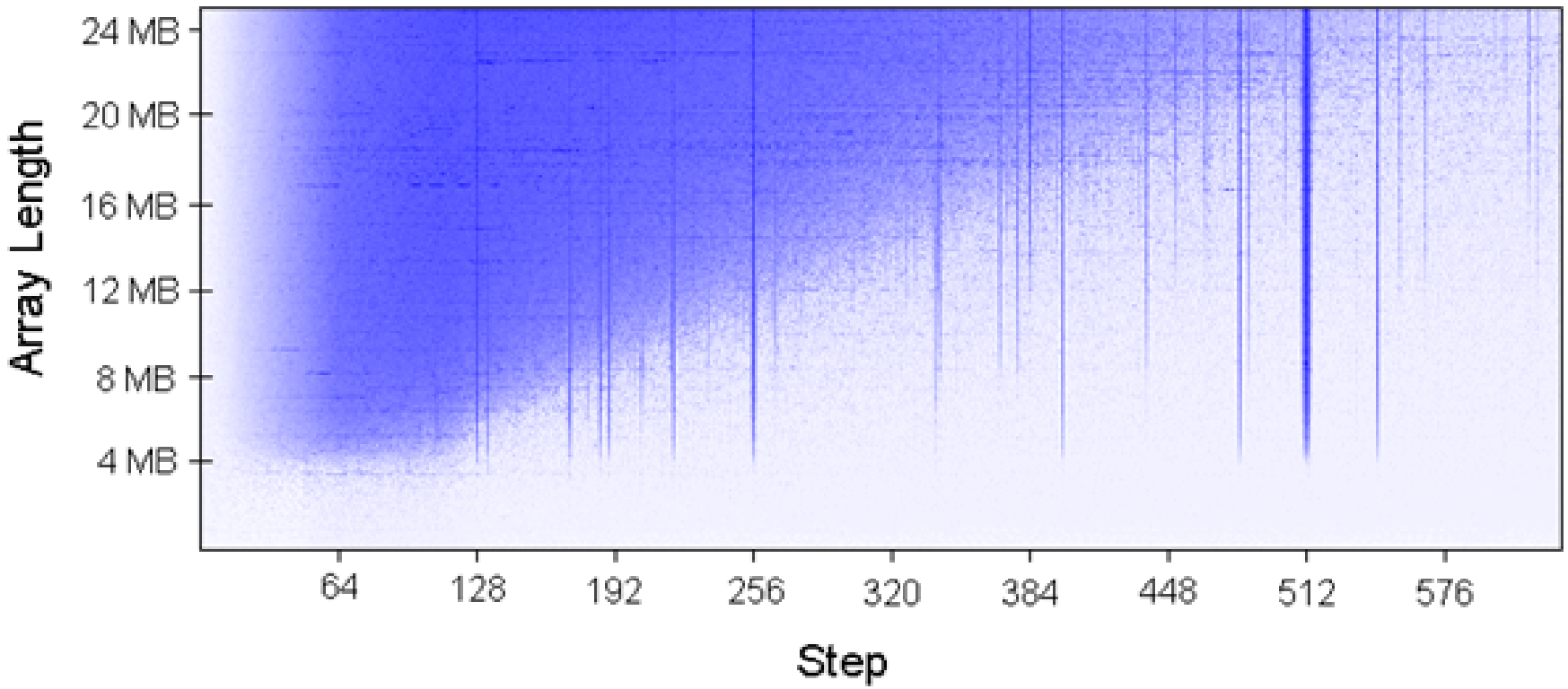
TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    public static long UpdateEveryKthByte(byte[] arr, int K)
2    {
3        Stopwatch sw = Stopwatch.StartNew();
4        const int rep = 1024*1024; // Number of iterations - arbitrary
5
6        int p = 0;
7        for (int i = 0; i < rep; i++)
8        {
9            arr[p]++;
10           p += K;
11           if (p >= arr.Length) p = 0;
12       }
13
14       sw.Stop();
15       return sw.ElapsedMilliseconds;
16   }
```

http://igoro.com/archive/gallery-of-processor-cache-effects/

Echte Cache Effekte: Assozivität

TECHNISCHE UNIVERSITÄT DARMSTADT



TECHNISCHE
UNIVERSITÄT
DARMSTADT

HDMI Controller

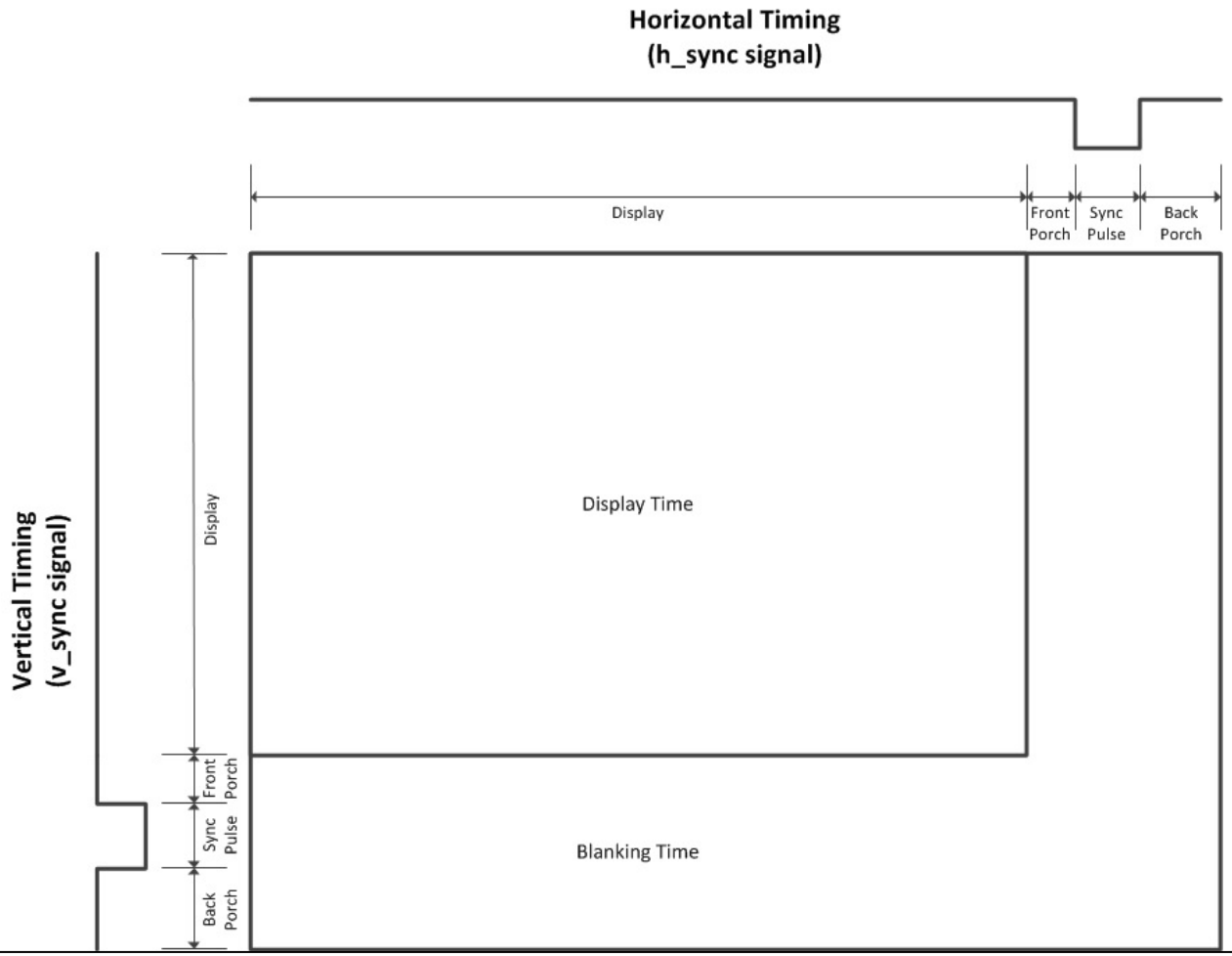
HDMI Einführung

TECHNISCHE UNIVERSITÄT DARMSTADT

- High Definition Multimedia Interface (HDMI)
- Unterstützt Video- und Audiodaten
- Kontrolle von anderen Geräten über CEC
- Mit HDMI 2.0a 14.4 Gbit/s Datenrate == 2160p mit 60 Hz
- 3D Support
- Audio Return Channel (ARC): Audio von Fernseher zu AVR
- Ethernet Channel
- Kabellänge 15 m problemlos Möglich, bis 100 m über Lichtwellenleiter
- Abwärtskompatibel zu DVI

VGA Signal

TECHNISCHE UNIVERSITÄT DARMSTADT



VGA Signal: General Timing

TECHNISCHE UNIVERSITÄT DARMSTADT

VESA Signal 1920 x 1440 @ 60 Hz (`http://tinyvga.com/vga-timing`)

Screen refresh rate	60 Hz
Vertical refresh	90 kHz
Pixel frequency	234 MHz

VGA Signal: Horizontal Timing (Line)

TECHNISCHE UNIVERSITÄT DARMSTADT

VESA Signal 1920 x 1440 @ 60 Hz (`http://tinyvga.com/vga-timing`)

Scanline part	Pixels
Visible area	1920
Front porch	128
Sync pulse	208
Back porch	344
Whole line	2600

VGA Signal: Vertical Timing (Frame)

TECHNISCHE UNIVERSITÄT DARMSTADT

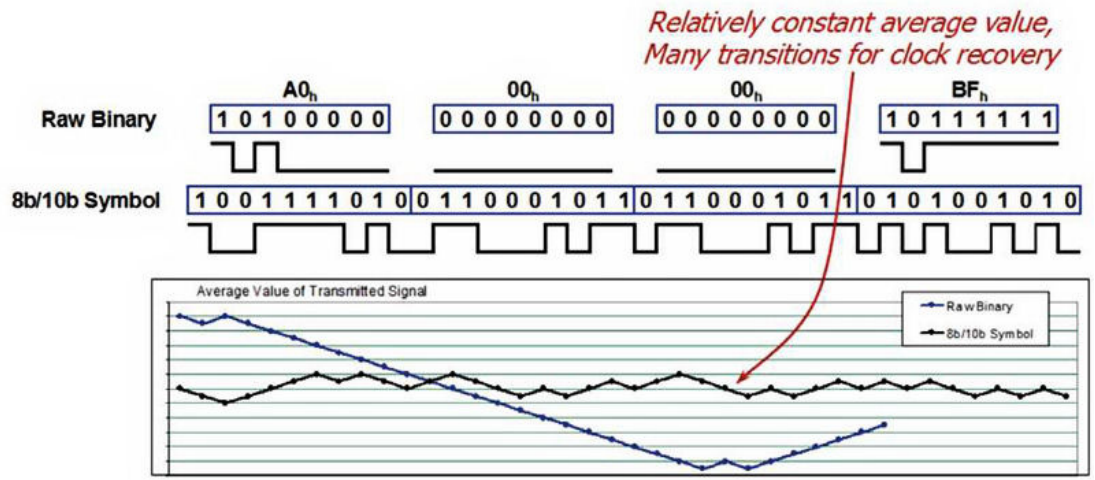
VESA Signal 1920 x 1440 @ 60 Hz (`http://tinyvga.com/vga-timing`)

Frame part	Pixels
Visible area	1440
Front porch	1
Sync pulse	3
Back porch	56
Whole line	1500

VGA Signal: Zu Digital

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bei VGA: Farbwerte als Spannung (Langsam)
- Bei HDMI: Übertragung des Farbwerts als 8 bit Wert
- Nutzung von $8b/10b$-Code (zur Vermeidung bestimmter elektronischer Probleme)
- Datenübertragung bei zehnfacher Pixelfrequenz



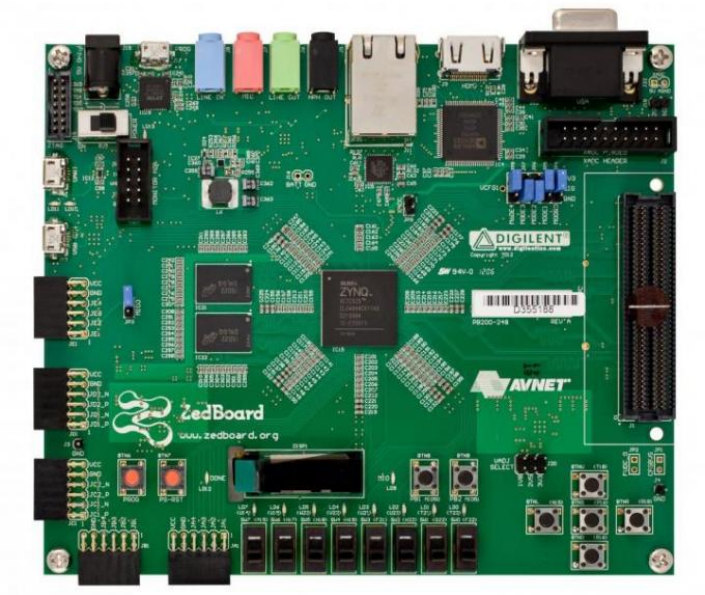
Die Platform: Xilinx Zynq

TECHNISCHE UNIVERSITÄT DARMSTADT

- Preis: $2868.75
- Aber: Günstige Alternative ZedBoard

Die Platform: Xilinx Zynq

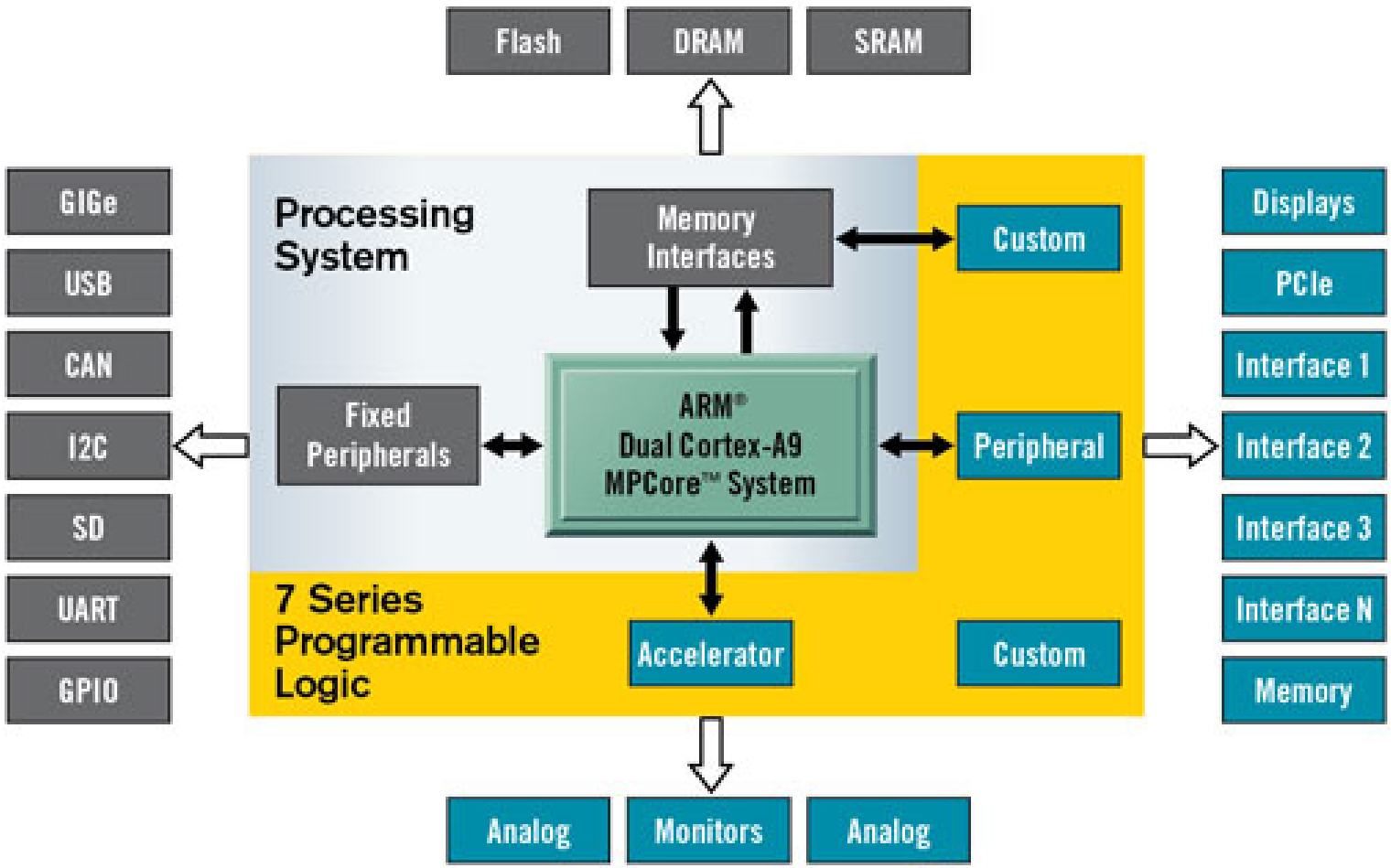
TECHNISCHE UNIVERSITÄT DARMSTADT



- Für viele Anwendungen geeignet (Ethernet, HDMI, Sound, GPIO etc.)
- Als Student $319
- Große Community `http://zedboard.org`

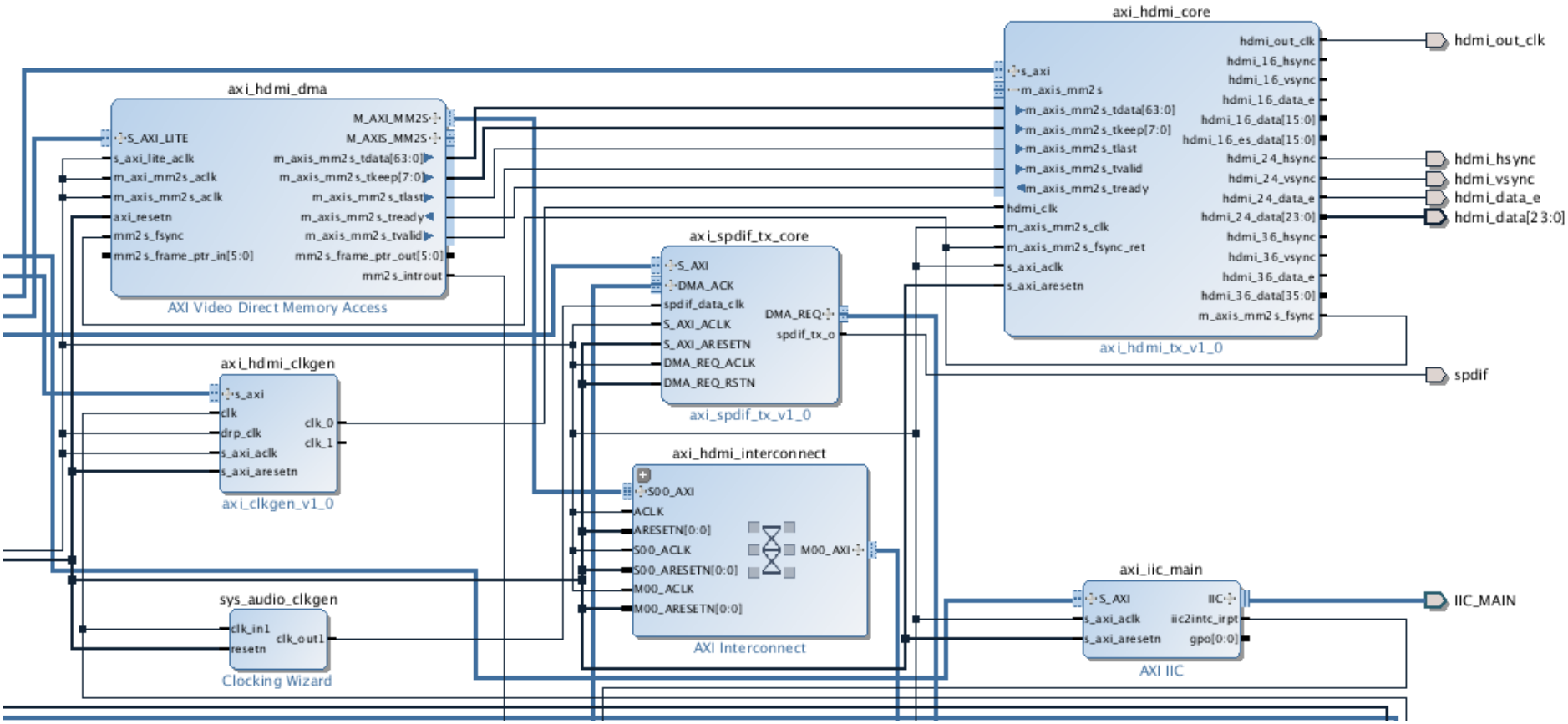
Die Platform: Xilinx Zynq

TECHNISCHE UNIVERSITÄT DARMSTADT



Ausgangspunkt: Design des Herstellers

TECHNISCHE UNIVERSITÄT DARMSTADT



Entwicklung eines eigenen Controllers

TECHNISCHE UNIVERSITÄT DARMSTADT

- Anbindung von Zynq FPGA zu HDMI Buchse → Datasheet

HDMI Video Output

[Figure 1-2, callout 18]

The ZC706 evaluation board provides a high-definition multimedia interface (HDMI®) video output using an Analog Devices ADV7511KSTZ-P HDMI transmitter at U53. The HDMI transmitter U53 is connected to the XC7Z045 AP SoC PL-side banks 12 and 13 and its output is provided on a Molex 500254-1927 HDMI type-A receptacle at P1. The ADV7511 supports 1080P 60Hz, YCbCr 4:4:4 encoding via 24-bit input data mapping.

The ZC706 evaluation board supports the following HDMI device interfaces:

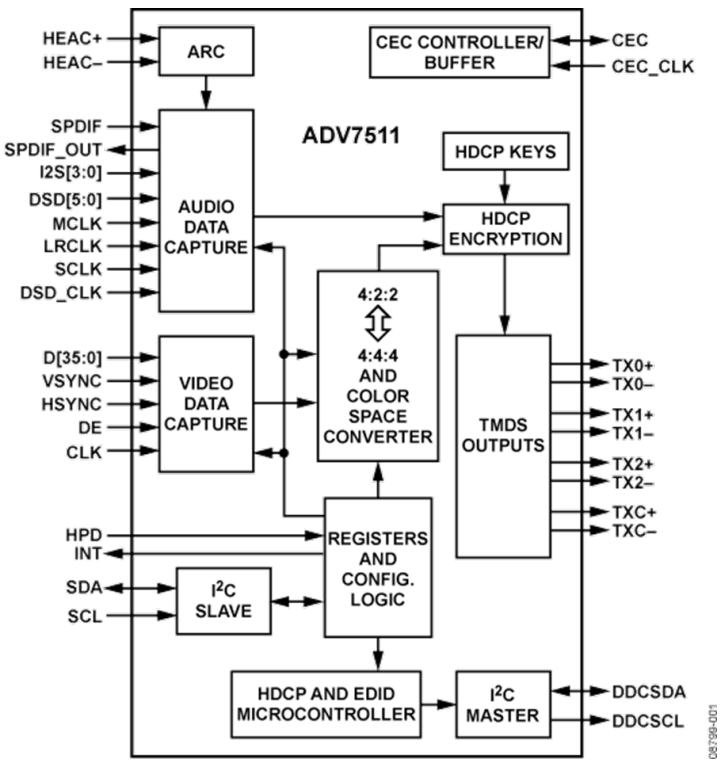
- 24 data lines
- Independent VSYNC, HSYNC
- Single-ended input CLK
- Interrupt Out pin to XC7Z045 AP SoC

- I^2C
- SPDIF

ADV7511

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kümmert sich um $8b/10b$-Code
- HDCP Kopierschutz
- Kommunikation mit Monitor



ADV7511: Firmware

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wie wird ADV7511 gesteuert? $\rightarrow I^2C$

SECTION 3 - QUICK START GUIDE

The Quick Start guide brings attention to registers that need to be configured when initially bringing up the HDMI transmitter. For detailed information, refer to the section number link on the right side of the page. Complete registers and their descriptions are listed in ▸ Section 5 -Register Maps

Power-up the Tx (HPD must be high)	
0x41[6] = 0b0 for power-up – power-down	▸ 4.8
Fixed registers that must be set on power up	
0x98 = 0x03	▸ 4.2.9
0x9A[7:5] = 0b111	▸ 4.2.9
0x9C = 0x30	▸ 4.2.9
0x9D[1:0] = 0b01	▸ 4.2.9
0xA2 = 0xA4	▸ 4.2.9
0xA3 = 0xA4	▸4.2.9
0xE0[7:0] = 0xD0	▸ 4.2.9
0xF9[7:0] = 0x00	▸ 4.2.9
Set up the video input mode	
0x15[3:0] – Video Format ID (default = 4:4:4)	▸4.3.2
0x16[5:4] – Input Color Depth for 4:2:2 (default = 12 bit)	▸4.3.2
0x16[3:2] – Video Input Style (default style = 2)	▸4.3.2
0x17[1] – Aspect ratio of input video (4x3 = 0b0, 16x9 = 0b1)	▸4.3.3
Set up the video output mode	
0x16[7:6] = 0b0 for 4:4:4 – Output Format (4:4:4 vs 4:2:2)	▸4.3.5
0x18[7] = 0b1 for YCbCr to RGB – CSC Enable	▸4.3.8
0x18[6:5] = 0b00 for YCbCr to RGB – CSC Scaling Factor	▸4.3.8
0xAF[1] = 0b1 for HDMI – Manual HDMI or DVI mode select	▸ 4.2.2
0x40[7] = 0b1 – Enable GC	▸ 4.3.6
0x4C[3:0] – Output Color Depth and General Control Color Depth (GC CD)	▸ 4.3.6
HDCP	
0xAF[7] = 0b1 for enable HDCP	▸4.7
0x97[6] – BKSV Interrupt Flag (Wait for value to be 0b1 then write 0b1)	▸4.7
Audio setup	
0x01 – 0x03 = 0x001800 for 48kHz - N Value	▸4.4.2
0x0A[6:4] – Audio Select (I2S = 0b000, SPDIF = 0b001, DSD = 0b010, HBR = 0b011, DST = 0b011)	▸ 4.4.1
Audio Mode	
0x0B[7] = 0b1 – SPDIF Enable	▸ 4.4.1.2
0x0C[5:2] = 0b1111 – I2S Enable	▸ 4.4.1.1
0x46 = 0xFF – DSD Enable	▸ 4.4.1.3
0x15[7:4] – I2S Sampling Frequency	▸ 4.4.1.1
0x0A[3:2] – Audio Mode	▸4.4.1
0x0A[3:2] – Audio Select	▸ 4.4.1

ADV7511: Linux Treiber

TECHNISCHE UNIVERSITÄT DARMSTADT

- Alternativ: Vorhandene Linux Treiber analysieren: `adv7511.c`

```
1    ...
2    /* H, V sync polarity, interpolation style */
3    val_mask = 0;
4    switch (config->out_params.up_conversion) {
5      default:
6      case 0:
7        val = 0x00;
8      break;
9      case 1:
10       val = 0x01;
11     break;
12   }
13   ...
```

- Mäßig Dokumentiert
- Für spezielle Fragen gut geeignet

ADV7511: Eingabedaten

TECHNISCHE UNIVERSITÄT DARMSTADT

- ADV7511 unterstützt verschiedene Formate für Eingabedaten
- Welches bei ZC706?

Table 5 Normal RGB or YCbCr 4:4:4 (36, 30, or 24 bits) with Separate Syncs; Input ID = 0

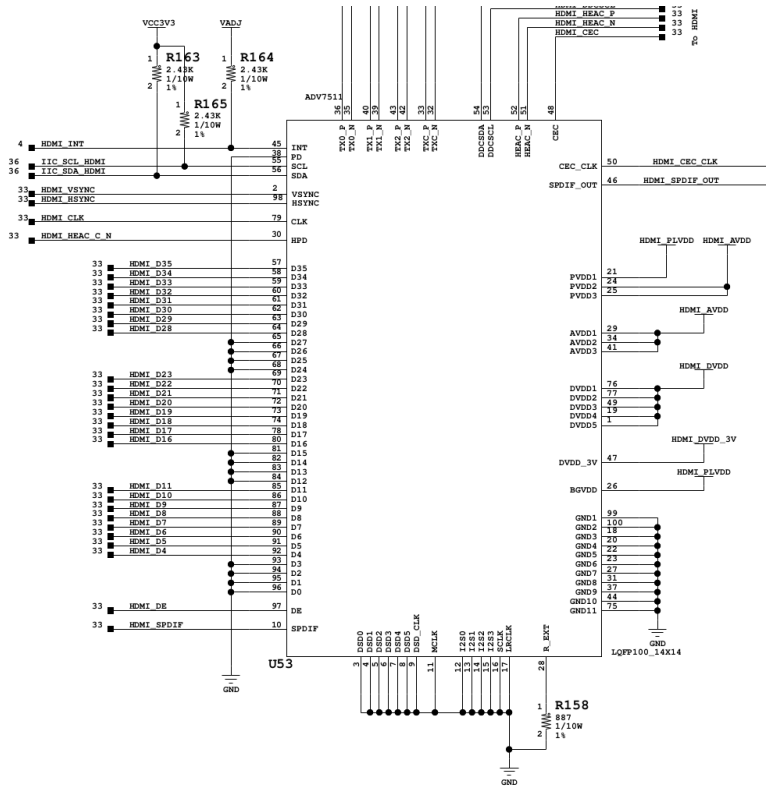
Mode	Format	Input Data D[35:0]																																			
		35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
36 bit	RGB	R[11:0]												G[11:0]												B[11:0]											
	YCrCb	Cr[11:0]												Y[11:0]												Cb[11:0]											
30 bit	RGB	R[9:0]												G[9:0]												B[9:0]											
	YCrCb	Cr[9:0]												Y[9:0]												Cb[9:0]											
24 bit	RGB	R[7:0]												G[7:0]												B[7:0]											
	YCrCb	Cr[7:0]												Y[7:0]												Cb[7:0]											
Pins D[35:0]		35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

An input format of RGB 4:4:4 or YCbCr 4:4:4 can be selected by setting the input ID (R0x15 [3:0]) to 0x0. There is no need to set the Input Style (R0x16[3:2]) or channel alignment (R0x48[4:3]). For timing details see the ▹ *ADV7511 Hardware User's Guide.*

ZC706: Schematic

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schematic bildet alle Verbindungen auf dem PCB ab



ADV7511: Eingabedaten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Für ZC706: YCbCr 4:2:2 mit 24 bit und Evenly Distributed

Table 8 **YCbCr 4:2:2 Formats (24, 20, or 16 bits) Input Data Mapping: R0x48[4:3] = '00' (evenly distributed) Input ID = 1 or 2**

Mode	Pixel	Input Data D[35:0]																																	
		35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
		Style 1																																			
24 bit	1st	Cb[11:4]												Y[11:4]												Cb[3:0]				Y[3:0]							
	2nd	Cr[11:4]												Y[11:4]												Cr[3:0]				Y[3:0]							
20 bit	1st	Cb[9:2]												Y[9:2]												Cb [1:0]				Y [1:0]							
	2nd	Cr[9:2]												Y[9:2]												Cr [1:0]				Y [1:0]							
16 bit	1st	Cb[7:0]												Y[7:0]																							
	2nd	Cr[7:0]												Y[7:0]																							
Style 2																																					

Zwischenstand

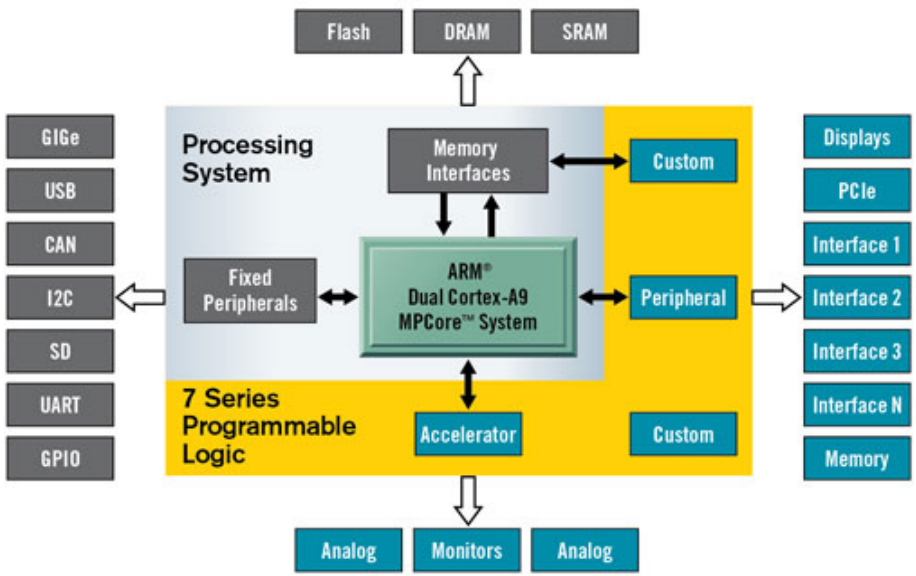
TECHNISCHE UNIVERSITÄT DARMSTADT

- Wir wissen...
  - wie der FPGA an die HDMI Buchse angeschlossen ist (ADV7511)
  - welche Daten das ADV7511 erwartet
  - wie das ADV7511 gesteuert werden kann
- Wir müssen noch herausfinden
  - wie wir mit dem ADV7511 reden
  - wie wir eine beliebige Taktfrequenz für die Pixelclock erzeugen können

I2C

TECHNISCHE UNIVERSITÄT DARMSTADT

- FPGA hat feste I^2C Controller in ARM Cores
- Alternative: I^2C Controller in FPGA

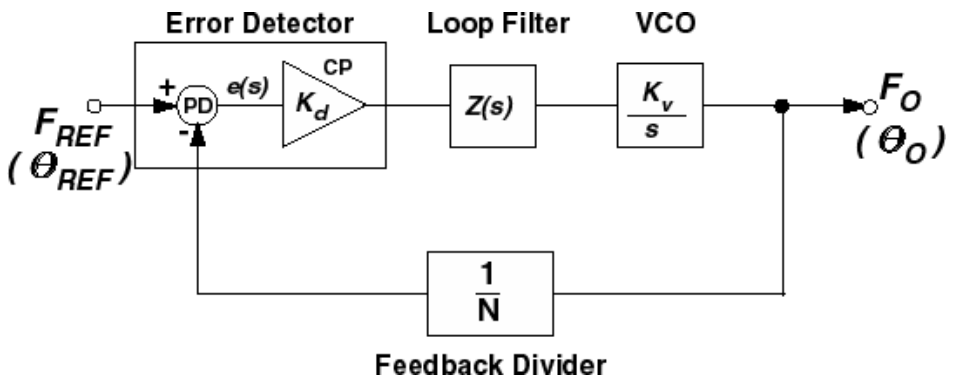


- Feste I^2C Cores werden genommen
- Das heißt aber: Design kann nicht direkt in FPGA ohne ARM Cores verwendet werden

Clock Generation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wie erzeugt man einen Takt im FPGA?
- Flexibelste Möglichkeit: Phased Locked Loop (PLL) bzw. Mixed-Mode Clock Manager (MMCM) bei Xilinx



$$F_{MUL} = \frac{F_{REF} \times m}{\frac{d}{d_{out}}} \quad (1)$$

$$F_{DIV} = \frac{F_{MUL}}{d} \quad (2)$$

$$F_O = \frac{F_{DIV}}{d_{out}} \quad (3)$$

Mehr Infos auf: `https://www.youtube.com/user/EEVblog`

Clock Generation: PLL Parameter finden

TECHNISCHE UNIVERSITÄT DARMSTADT

- PLL benötigt Parameter d, d_{out}, m (alles Ganzzahlen)
- Kleines Python Script das mögliche Kombinationen ausgibt

```
1   pllFinder.py 193.25
2   200.0 * 27 / 7 / 4 = 192.857142857
3   200.0 * 29 / 5 / 6 = 193.333333333
4   200.0 * 29 / 6 / 5 = 193.333333333
5   200.0 * 31 / 8 / 4 = 193.75
6   200.0 * 54 / 14 / 4 = 192.857142857
7   200.0 * 58 / 10 / 6 = 193.333333333
8   200.0 * 58 / 12 / 5 = 193.333333333
9   200.0 * 58 / 15 / 4 = 193.333333333
10  200.0 * 62 / 16 / 4 = 193.75
11  Best:  (29, 6, 5)
```

Clock Generation: PLL Parameter übergeben

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wie übernimmt PLL die Parameter zur Laufzeit?
- Leider kein Datasheet → Foren/Quellcode durchsuchen
- Ergebnis: PLL benötigt `filter` und `lock`
- Übergabe über `dynamic reconfiguration port (DRP)`
- Format für DRP? → Sourcecode
- Was tun ohne vorhandenen Sourcecode? Hersteller fragen
  - Im schlimmsten Fall `reverse engineering`

```
1   PLLRegs {
2   axi_clkgen_reg_clk_fb1: 911,
3   axi_clkgen_reg_clk_fb2: 128,
4   axi_clkgen_reg_clk_div: 195,
5   axi_clkgen_reg_clk_out1: 131,
6   axi_clkgen_reg_clk_out2: 128,
7   axi_clkgen_reg_lock1: 325,
8   axi_clkgen_reg_lock2: 31745,
9   axi_clkgen_reg_lock3: 32745,
10  axi_clkgen_reg_filter1: 256,
11  axi_clkgen_reg_filter2: 2192
12  };
```

Bluespec Design

TECHNISCHE UNIVERSITÄT DARMSTADT

- Welche Module benötigen wir?
  - HDMI Signal Generator: Erzeugt eigentliche Ausgabedaten
  - RGB to YCbCr 4:2:2 Converter
  - Configuration Register: Auflösung und Frame Adresse einstellen
  - PLL Handler (with DRP)
  - Video Direct Memory Access (VDMA): Pixel aus Hauptspeicher an Signal Generator

Bluespec Design: Module

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die auf der letzten Folie beschriebenen Teile in BSV
- Verschiedene Module laufen mit verschiedenem Takt
- `Globally asynchronous locally synchronous`
- Übergäng zwischen Clock-Domains behandelt Bluespec
- FIFO oder andere Strukturen zur Synchronisation zwischen Modulen

```
1    XilinxClockController pll <- mkXilinxClockController(clockParams,
2                                                         currentClk);
3    Reset hdmiRstN <- mkAsyncResetFromCR(0, pll.clkout0);
4
5    HDMIController controller <- mkHDMIController(clocked_by pll.clkout0,
6                                                   reset_by hdmiRstN);
7    AxiHandlingIfc axi <- mkAxiHandler();
8
9    AxiHPMaster master <- mkAxiHPMaster();
```

Bluespec Design: Top Level Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    interface HDMI;
2      interface AxiRdFabricMaster#(`AXI_PRM_HP) m_rd;
3      interface AxiRdFabricSlave#(`AXI_PRM_VIV) s_rd;
4      interface AxiWrFabricSlave#(`AXI_PRM_VIV) s_wr;
5      interface HDMI_Pins hdmi;
6    endinterface
```

Bluespec Design: Top Level Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1     interface s_rd = axi.s_rd;
2     interface s_wr = axi.s_wr;
3     interface m_rd = master.m_rd;
4
5     interface HDMI_Pins hdmi;
6       interface clk = pll.clkout0;
7       method data   = controller.hdmi.data;
8       method de     = controller.hdmi.de;
9       method hsync  = controller.hdmi.hsync;
10      method vsync  = controller.hdmi.vsync;
11      method spdif  = controller.hdmi.spdif;
12    endinterface
```

Bluespec Design: Auflösung einstellen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schreibanfragen auf Register werden in gewünschte Operationen umgesetzt

```
1   rule handleAxiWrites;
2     HdmiRequest req <- axi.request.get();
3     case(req.addr) matches
4       0: begin
5         let resolution = videoTimings[req.data];
6         timingFIFO.enq(resolution);
7         pllCurrent <= pllRegs[req.data];
8         pllSettingFSM.start();
9         master.setResolution(resolution.h.active, resolution.v.active);
10      end
11      1: begin
12        master.setStartAddress(req.data);
13      end
14      default: begin
15        $display("Nothing to do here.");
16      end
17    endcase
18  endrule
```

Bluespec Design: PLL anpassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    Stmt setPLLSettings = {
2    seq
3    action
4      XilinxClockRequest pllReq;
5      pllReq.rnw  = False;
6      pllReq.addr = 5'h11;
7      pllReq.data = pllCurrent.axi_clkgen_reg_clk_fb1;
8      pll.csr.request.put(pllReq);
9    endaction
10   action
11     XilinxClockRequest pllReq;
12     pllReq.rnw  = False;
13     pllReq.addr = 5'h12;
14     pllReq.data = pllCurrent.axi_clkgen_reg_clk_fb2;
15     pll.csr.request.put(pllReq);
16   endaction
17   ...
```

Bluespec Design: HDMI Signal - HSync

TECHNISCHE UNIVERSITÄT DARMSTADT

- Simpler Zähler
- HSync abhängig von Zählerstand und Timing

```
1    rule setHSync;
2      let hSyncTmp = False;
3      if((hCounter > maxDataFPorchH) && (hCounter <= maxSyncH)) begin
4         hSyncTmp = True;
5      end else begin
6         hSyncTmp = False;
7      end
8      hSyncReg <= hSyncTmp;
9    endrule
10
11   rule updateHCounter;
12     if(hCounter < maxHorizontal) begin
13        hCounter <= hCounter + 1;
14     end else begin
15        hCounter <= 0;
16     end
17   endrule
```

Bluespec Design: HDMI Signal - VSync

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wie HSync
- Zähler zählt Lines statt Pixeln

```
1    rule setVSync;
2      let vSyncTmp = False;
3      if((vCounter > maxDataFPorchV) && (vCounter <= maxSyncV)) begin
4         vSyncTmp = True;
5      end else begin
6         vSyncTmp = False;
7      end
8      vSyncReg <= vSyncTmp;
9    endrule
10
11   rule updateVCounter;
12     if(maxHorizontal == hCounter) begin
13        if(vCounter < maxVertical) begin
14           vCounter <= vCounter + 1;
15        end else begin
16           vCounter <= 0;
17        end
18     end
19   endrule
```

Bluespec Design: Erster Test

TECHNISCHE UNIVERSITÄT DARMSTADT

- Erster Test ohne variable Pixel

```
1    rule setColors (deW);
2       let pixel = pixIn.first();
3       let color = yCBCRtoADV7511(pixel.cb, pixel.cr, pixel.y1, pixel.y2);
4       if(unpack(pack(hCounter)[0])) begin
5          rDataOut <= color[47:24];
6          pixIn.deq();
7       end else begin
8          rDataOut <= color[23:0];
9       end
10   endrule
```

Bluespec Design: Erster Test

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verwandeln von YCbCr zu der korrekten Platzierung für ADV7511

```
1    function Bit#(48) yCBCRtoADV7511(Bit#(12) cb, Bit#(12) cr,
2                                     Bit#(12) y1, Bit#(12) y2);
3      Bit#(36) pix1 = {cb[11:4],4'b0,y1[11:4],4'b0,cb[3:0],y1[3:0],4'b0};
4      Bit#(24) pix1Trunc = {pix1[35:28],pix1[23:16],pix1[11:4]};
5      Bit#(36) pix2 = {cr[11:4],4'b0,y2[11:4],4'b0,cr[3:0],y2[3:0],4'b0};
6      Bit#(24) pix2Trunc = {pix2[35:28],pix2[23:16],pix2[11:4]};
7      return {pix1Trunc,pix2Trunc};
8    endfunction
```

Bluespec Design: Testbench

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bluespec erlaubt einbinden von C/C++ Funktionen
- Sehr flexibel
- Virtueller Bildschirm für Simulation (z.B. mit SDL)
- Aber: Verhält sich Simulationsumgebung wie echte Hardware?
  - In den ersten Iterationen meistens nicht
  - Testen in Hardware notwendig

Bluespec zu Vivado

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zusätzliches Modul das für die korrekte Benennung von Signalen im erzeugten Verilog sorgt

```
1   // Interface with all names in place for automatic import in Vivado
2   interface HDMIVivadoIfc;
3       (*prefix="S_AXI"*)
4       interface AxiRdSlave#(`AXI_PRM_VIV) s_rd;
5       (*prefix="S_AXI"*)
6       interface AxiWrSlave#(`AXI_PRM_VIV) s_wr;
7
8       (*prefix="M_AXI"*)
9       interface AxiRdFabricMaster#(`AXI_PRM_HP) m_rd;
10
11      interface HDMI_Pins hdmi;
12  endinterface
```

Bluespec zu Vivado: Erzeugtes Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

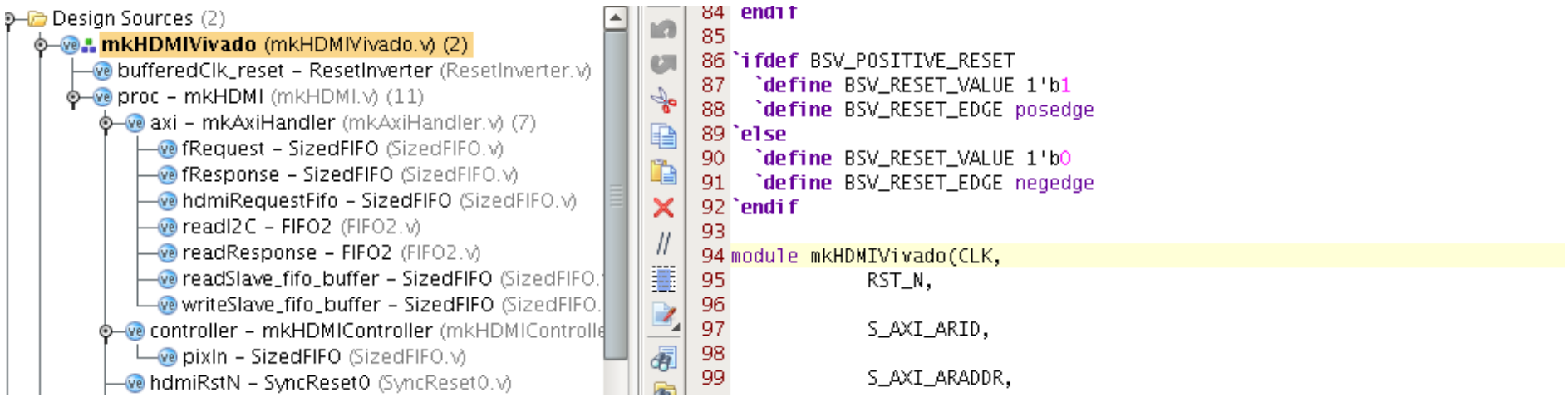
- Benennung passt
- Nächster Schritte FPGA

```
1     // M_AXI_ARADDR                 O    32
2     // M_AXI_ARLEN                  O     4
3     // M_AXI_ARSIZE                 O     3
4     // M_AXI_ARBURST                O     2
5     // M_AXI_ARLOCK                 O     2
6     // M_AXI_ARCACHE                O     4
7     // M_AXI_ARPROT                 O     3
8     // M_AXI_ARVALID                O     1
9     // M_AXI_RREADY                 O     1
10    // hdmi_data                    O    24 reg
11    // hdmi_de                      O     1 reg
12    // hdmi_hsync                   O     1 reg
13    // hdmi_vsync                   O     1 reg
14    // hdmi_spdif                   O     1 const
```

Bluespec zu Vivado: Verilog in Vivado

TECHNISCHE UNIVERSITÄT DARMSTADT

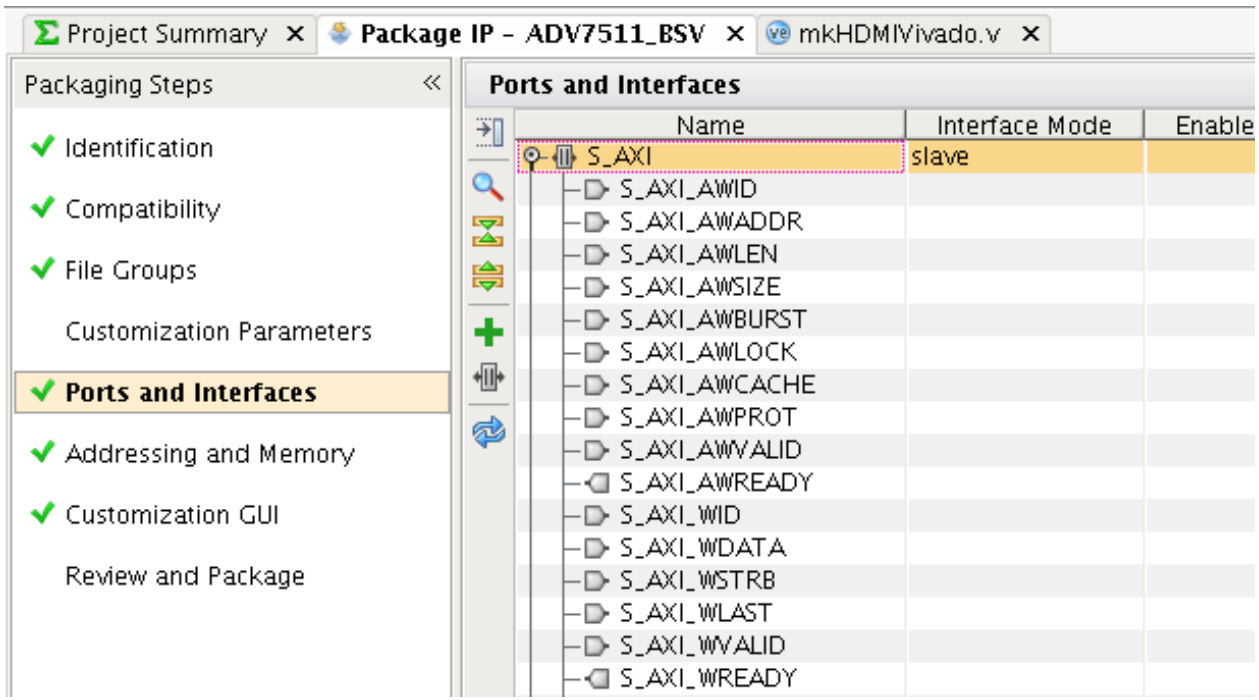
- Erzeugtes Verilog importiert in Vivado
- Verschiedene Primitiven aus `$BLUESPECDIR/Verilog`



Bluespec zu Vivado: Verilog in Vivado

TECHNISCHE UNIVERSITÄT DARMSTADT

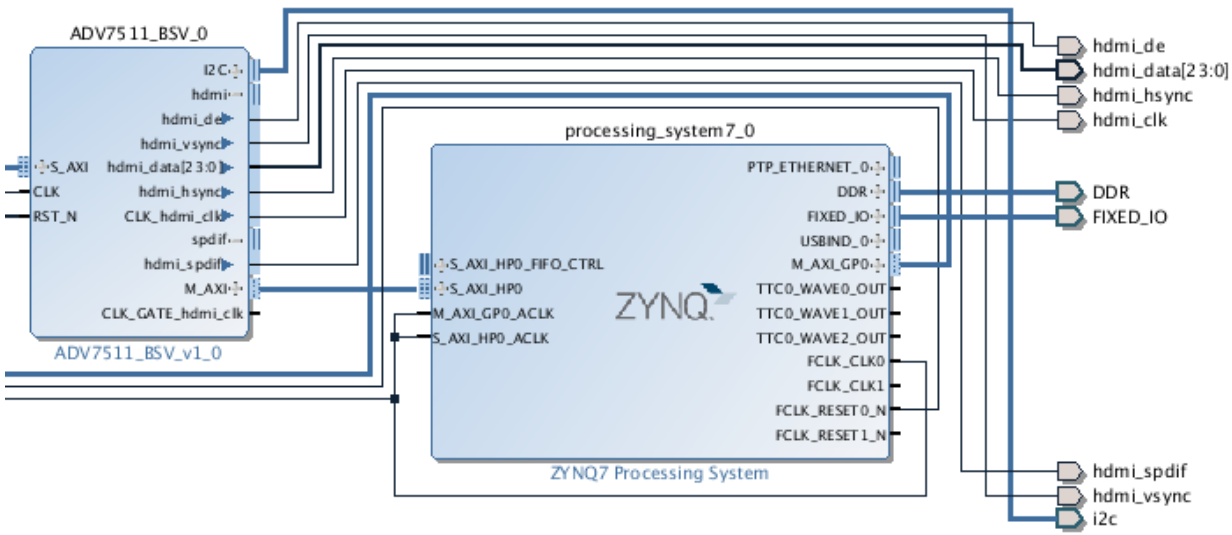
- Gepackt als Vivado Package
- Signale werden verschiedenen Interfaces zugewiesen (AXI, HDMI etc.)



Bluespec zu Vivado: Verilog in Vivado

TECHNISCHE UNIVERSITÄT DARMSTADT

- Das erzeugte Modul wird mit Zynq FPGA verschaltet
- Parameter Einstellbar (Takt der Hauptclock, aktivierte Interfaces etc.)
- Design fertig zur Synthese



Bluespec zu Vivado: Synthese

TECHNISCHE UNIVERSITÄT DARMSTADT

- Erfolgreiche Synthese generiert Bitstream für FPGA
- Außerdem umfangreiche Reports

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	2819	0	218600	1.28
LUT as Logic	2436	0	218600	1.11
LUT as Memory	383	0	70400	0.54
LUT as Distributed RAM	382	0		
LUT as Shift Register	1	0		
Slice Registers	2304	0	437200	0.52
Register as Flip Flop	2304	0	437200	0.52
Register as Latch	0	0	437200	0.00
F7 Muxes	34	0	109300	0.03
F8 Muxes	11	0	54650	0.02

Bluespec zu Vivado: Synthese

TECHNISCHE UNIVERSITÄT DARMSTADT

- Probleme bei der Synthese → Zurück zu Bluespec
  - FIFO statt BypassFIFO
  - Weniger Operationen pro Rule
  - Externer Speicher statt BRAM
  - ...

```
1   ------------------------------------------------------------------------
2   | Design Timing Summary
3   | ---------------------
4   ------------------------------------------------------------------------
5       WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints
6       -------      -------  ---------------------  -------------------
7        -2.489    -1727.268                   1633                 9300
8   Timing constraints are not met.
9   ...
10  Max Delay Paths
11  ------------------------------------------------------------------------
12  Slack (VIOLATED) : -1.746ns  (required time - arrival time)
13    Source:          zynq_i/ADV7511_BSV_0/inst/proc/axi/hdmiRequestFifo/
14                     D_OUT_reg[30]
15    Destination:     zynq_i/ADV7511_BSV_0/inst/proc/master_maxAddress_reg[31]
16  ...
```

Bluespec zu Vivado: Linux auf Zynq

TECHNISCHE UNIVERSITÄT DARMSTADT

- ARM Cores unterstützen vollständiges Linux
- Laden des Bitstreams während der Laufzeit
- Setzen der ADV7511 Firmware und der Auflösung durch C Programm

```
1    root@linaro-developer ~ # cat system.bit.bin > /dev/xdevcfg
```

Bluespec zu Vivado: ADV7511 Firmware

TECHNISCHE UNIVERSITÄT DARMSTADT

- ADV7511 Firmware setzen über I^2C

```
1    I2CBus i2c("/dev/i2c-0");
2    i2c.addressSet(0x74);
3    i2c.writeByte(0x2, 0x2);
4    i2c.addressSet(0x39);
5    i2c.writeByte(0x41, 0x50); // Disable chip
6    // Wait for HDMI connected
7    while(!(i2c.readByte(0x42) & 0x40)) {
8            std::cout << "HDMI not connected" << std::endl;
9            sleep(1);
10   }
11   ...
```

Bluespec zu Vivado: ADV7511 Firmware

TECHNISCHE UNIVERSITÄT DARMSTADT

- Auflösung in Bluespec Modul über Memory Mapped Bus

```
1    int fd = open("/dev/mem", O_RDWR);
2    volatile unsigned int *hdmi =
3                        (volatile unsigned *)mmap(NULL, 0xFFFFF,
4                         PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0x43C00000);
5    hdmi[128] = 0;
6    hdmi[129] = 536870912;
7    int input = 0;
8    while(input != -1) {
9            std::stringstream ss;
10           std::string str;
11           std::getline(std::cin, str);
12           ss << str;
13           ss >> input;
14           std::cout << "Setting mode to " << input << std::endl;
15           if(input >= 0 && input < 4) hdmi[128] = input;
16   }
```

Video: Start und Auflösung ändern

HDMI auf Zynq

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1   HDMI not connected
2   HDMI not connected
3   Device active.
4   Writing cmd 98 with data 3
5   Writing cmd 9a with data 70
6   Writing cmd 9c with data 30
7   Writing cmd 9d with data 61
8   Writing cmd a2 with data a4
9   Writing cmd a3 with data a4
10  Writing cmd e0 with data d0
11  Writing cmd f9 with data 0
12  Writing cmd 15 with data 1
13  Writing cmd 16 with data 28
14  Writing cmd 17 with data 2
15  Writing cmd 18 with data 46
16  Writing cmd af with data 16
17  Writing cmd 40 with data 0
18  Writing cmd 4c with data 4
19  Print HDMI status
20  HPD register: f0
21  DDC Control Error register: 0 Interrupt state: 0
22  PLL Lock status: 1
```

HDMI auf Zynq

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1    EDID ID Address: 7e
2    EDID Ready Interrupt: e0
3    Setting resolution to 1920x1200.
4    1
5    Setting mode to 1
6    3
7    Setting mode to 3
8    0
9    Setting mode to 0
```

Bluespec zu Vivado: Debugging

TECHNISCHE UNIVERSITÄT DARMSTADT

- Logic Analyser
- Simulation
- FPGA Features (z.B. integrierte Logic Analyser)
- Errata Sheets

TECHNISCHE UNIVERSITÄT DARMSTADT

Video: Pixel Takt auf Oscilloscope

Bluespec zu Vivado: The final Product

TECHNISCHE UNIVERSITÄT DARMSTADT

- Modul von eben erweitert um
  - VDMA
  - Linux Treiber
- Kleiner und schneller als das Modul vom Hersteller
- Mögliche Erweiterungen:
  - Textausgabe in Hardware
  - Geometrie in Hardware
  - GPGPU anbinden