TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

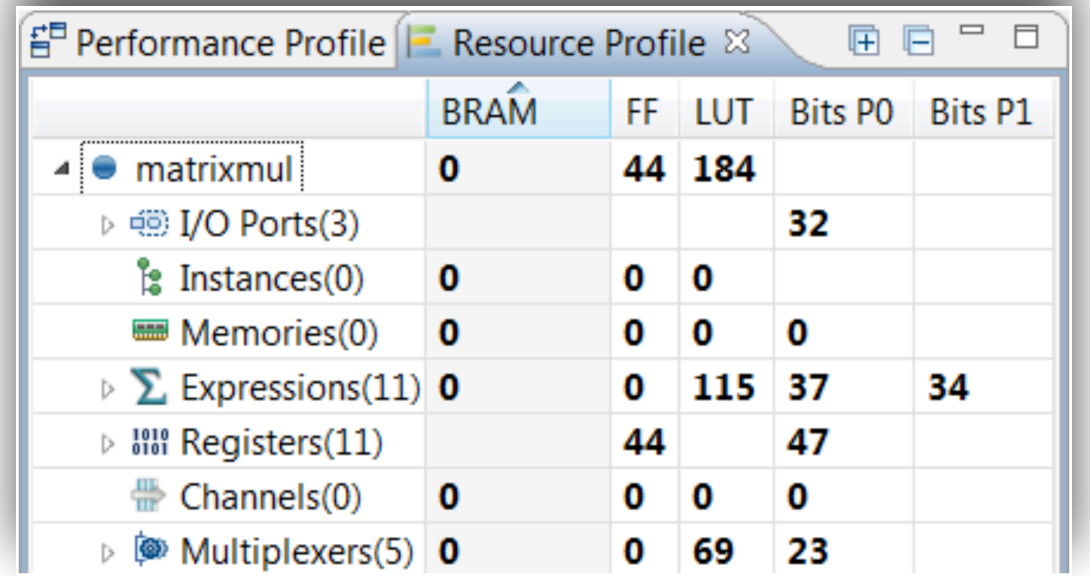
Institut für Technische Informatik
Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur

Optimierung rekonfigurierbarer Hardware mittels High-Level Synthese

Vortrag von Fabian Wendel im Rahmen des Proseminars

Dresden // 05. Juli 2018

Gliederung



	BRAM	FF	LUT	Bits P0	Bits P1
matrixmul	0	44	184		
I/O Ports(3)				32	
Instances(0)	0	0	0		
Memories(0)	0	0	0	0	
Expressions(11)	0	0	115	37	34
Registers(11)		44		47	
Channels(0)	0	0	0	0	
Multiplexers(5)	0	0	69	23	

[XILL1]

Abkürzungsverzeichnis

HLS	High-Level Synthese
FPGA	Field Programmable Gate Array
SoC	System on a Chip
AP SoC	All Programmable SoC
PL	Programmable Logic
PS	Processing System
RTL	Register Transfer Level

[AVERZ]

Einleitung

Motivation & Einleitung
Warum & Wie?

Rekonfigurierbare Hardware

- Relevanz im Architekturdesign
- Löst Grenzen auf
- Flexibilität
- Entwurf per Hand umständlich

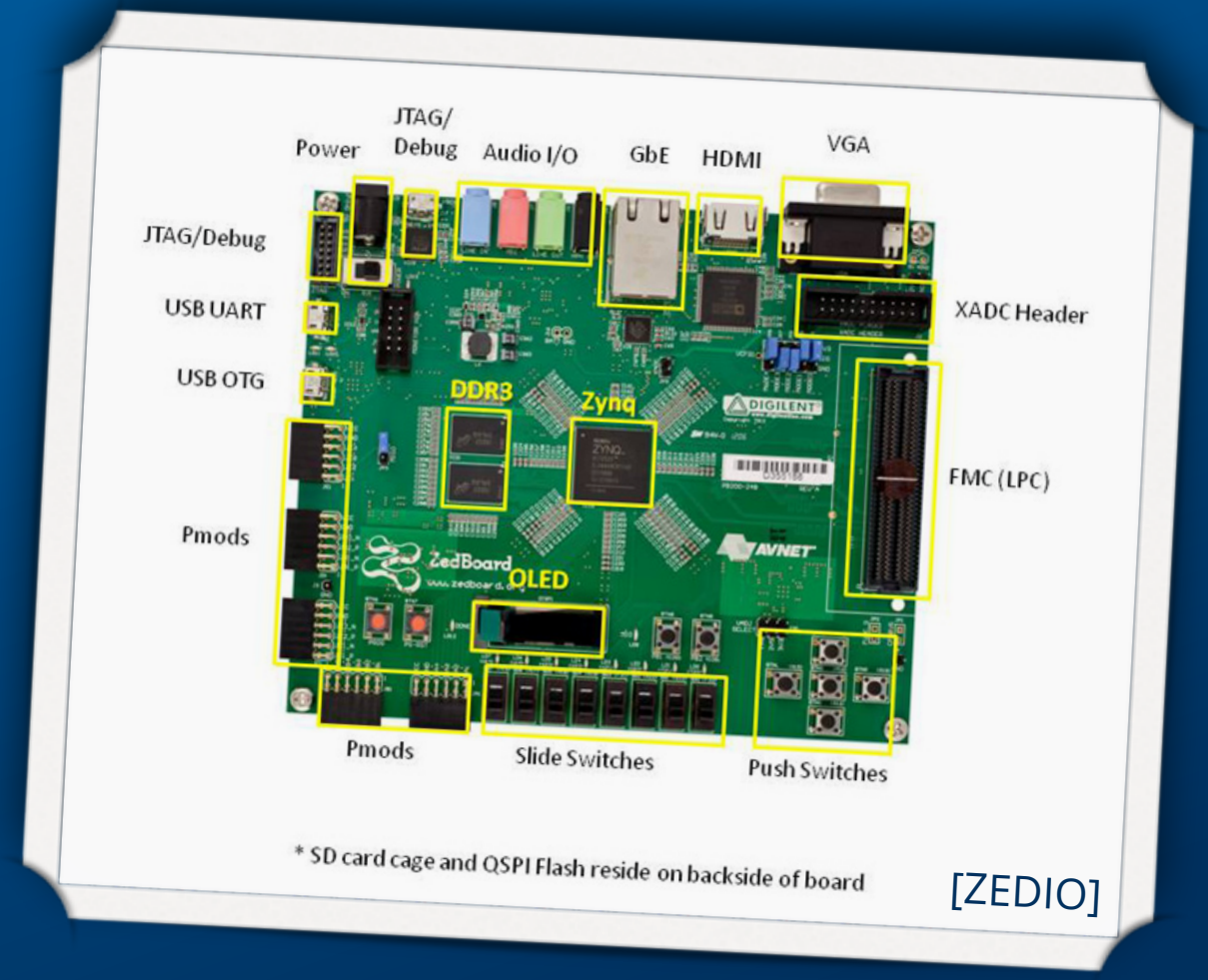
Lösung: Automatisierter Systementwurf mittels HLS: Software → Hardware

Thema dieser Arbeit

- Verschiedene Möglichkeiten der Leistungssteigerung
- Auswertung nach Effizienz

[IDEE]

Verwendete Plattform



[ZEDIO]

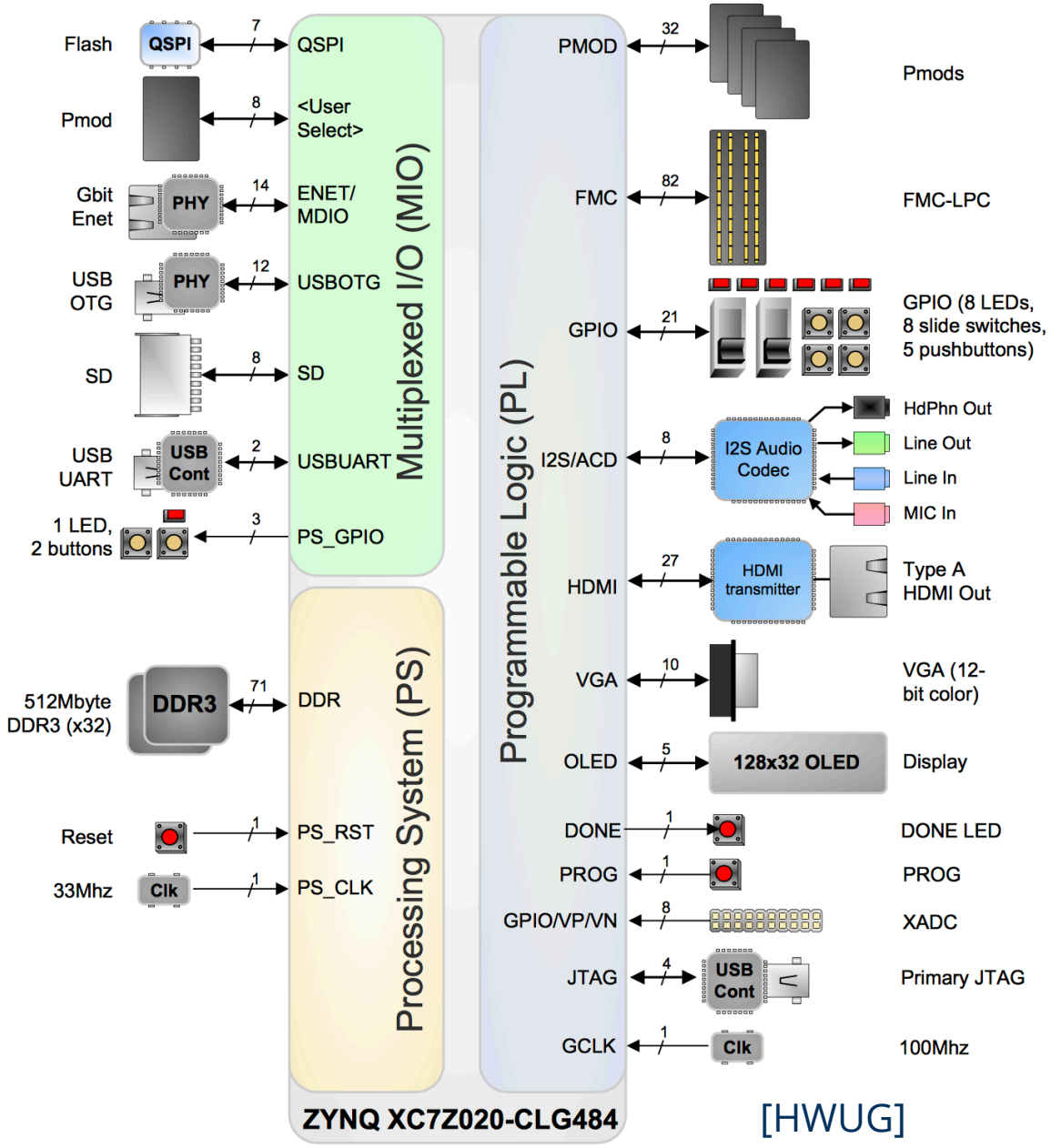
Verwendete Plattform
Überblick

Zum Board

- Boardname: „ZedBoard“
- Hersteller: AVNET
- Einsatzbereich: Schnelles Prototyping, Machbarkeit

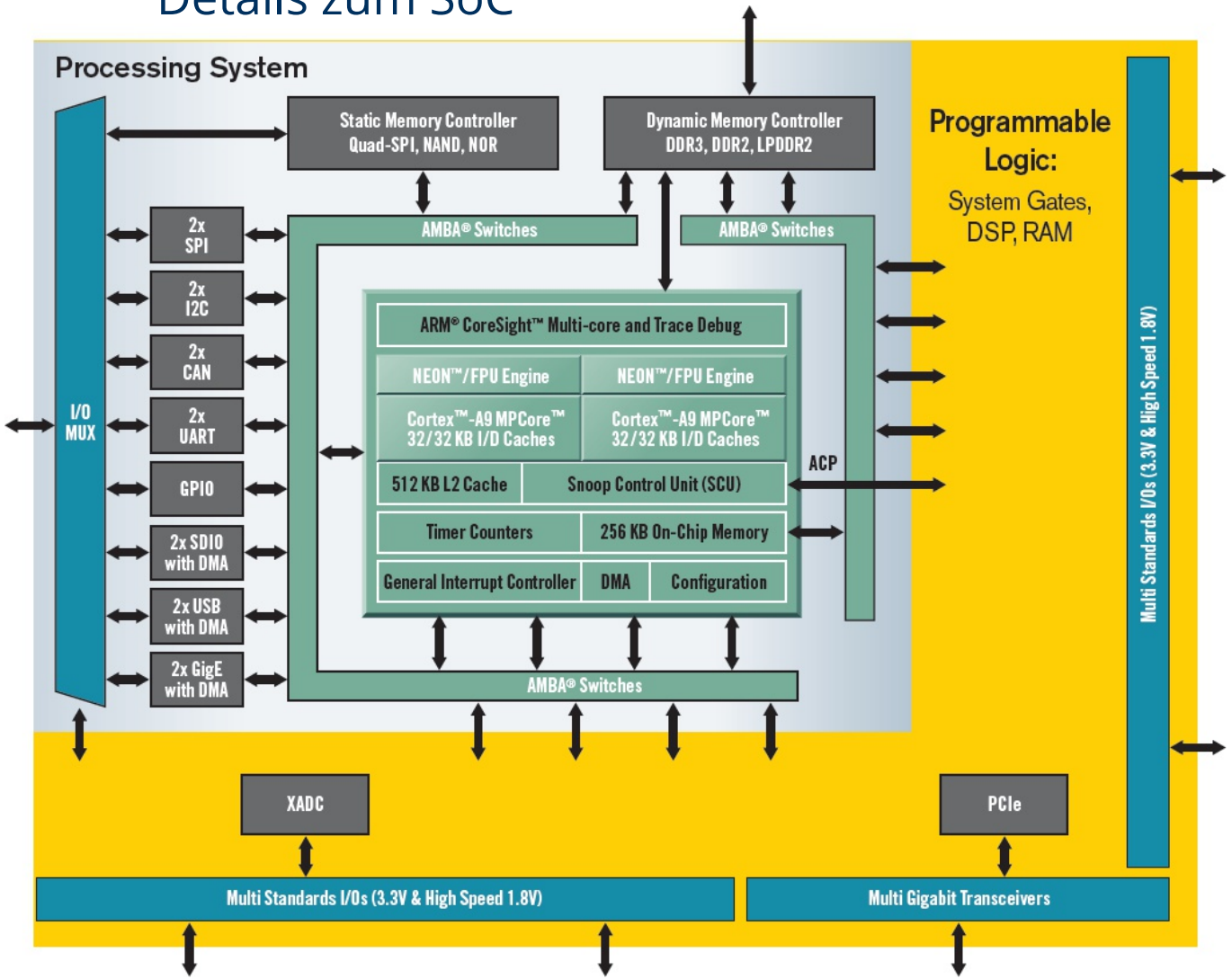
Spezifikationen

- SoC: Zynq-7000 AP SoC XC7Z020-CLG484-1
- Arbeitsspeicher: 512MB DDR3
- ROM: 4GB SD-Karte
- Kommunikation: USB-JTAG, USB-UART, Ethernet
- Anzeige: 128x32 OLED Display, VGA- & HDMI-Ausgang
- Taktgeber: 33.333 MHz Processing System (PS),
  100MHz Programmable Logic (PL)



Verwendete Plattform

Details zum SoC



Besteht aus

- Xilinx Zynq-7000 All Programmable SoC (**AP** SoC)
- Dual Cortex-A9 **PS**
- 85.000 Series-7 **PL** Zellen

[CNXS]

Verwendeter Algorithmus

Verwendeter Algorithmus
Matrixmultiplikation

Warum Matrixmultiplikation

— Bekannte Rechenoperation

— Geringe Komplexität → einfach nachzuvollziehen

```
#include "matrixmul.h"
void matrixmul(
        mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
        mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
        result_t res[MAT_A_ROWS][MAT_B_COLS])
{
  // Iterate over the rows of the A matrix
  Row: for(int i = 0; i < MAT_A_ROWS; i++) {
    // Iterate over the columns of the B matrix
    Col: for(int j = 0; j < MAT_B_COLS; j++) {
      // Do the inner product of a row of A and col of B
      res[i][j] = 0;
      Product: for(int k = 0; k < MAT_B_ROWS; k++) {
        res[i][j] += a[i][k] * b[k][j];
      }
    }
  }
}
```

Algorithmus

— Einzelne Einträge: $c_{ik} = \sum_{j=1}^{m} a_{ij} \cdot b_{jk}$

[XILL1]

High-Level Synthese

High-Level Synthese
Begriffsklärung

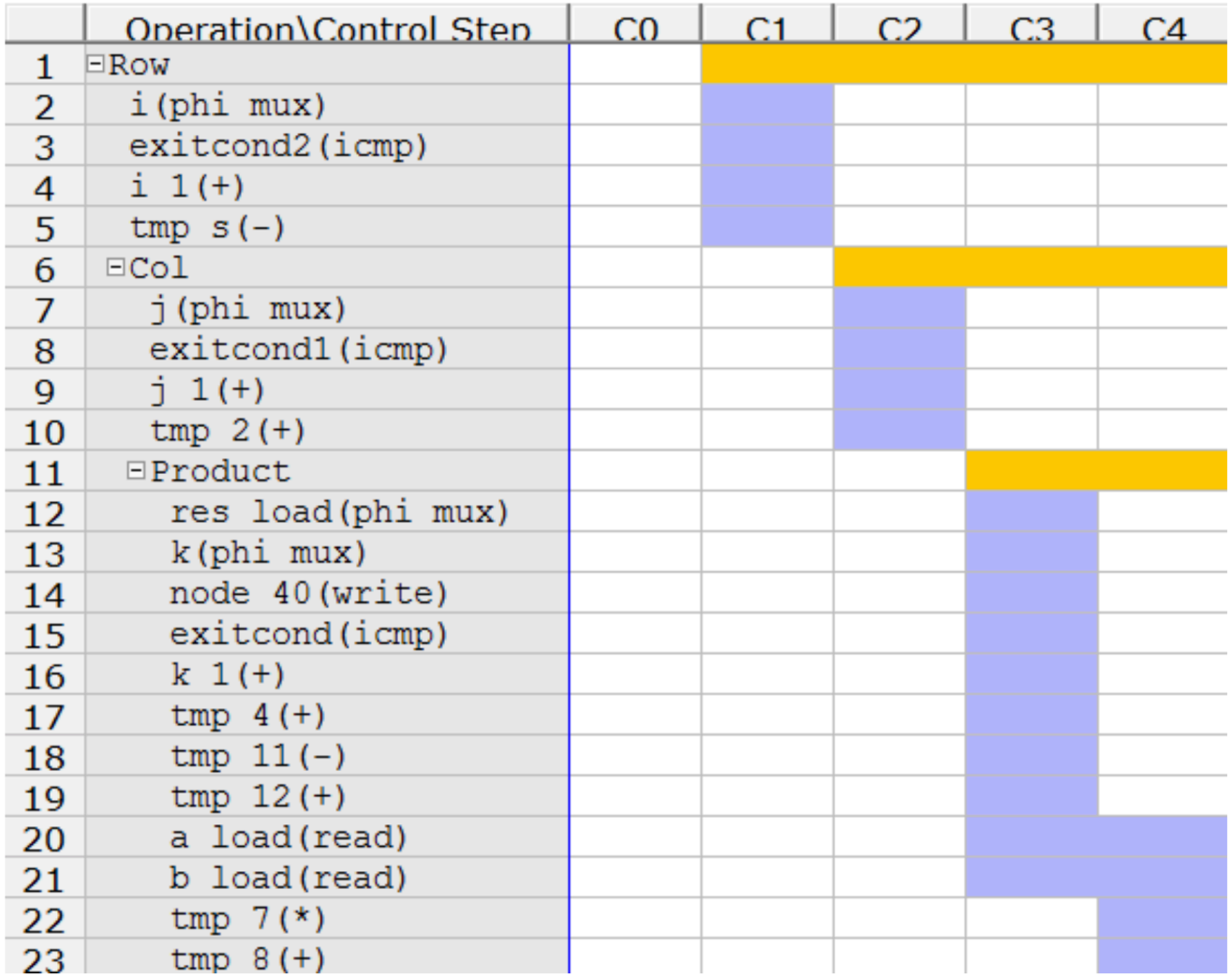
Nutzen

— Rechnergestützt

— Algorithmische Schaltungsfunktionen
→ Register-Transfer-Struktur

Funktionsweise

Operationen werden zu arithmetisch/ Logischen Funktionseinheiten zusammengefasst

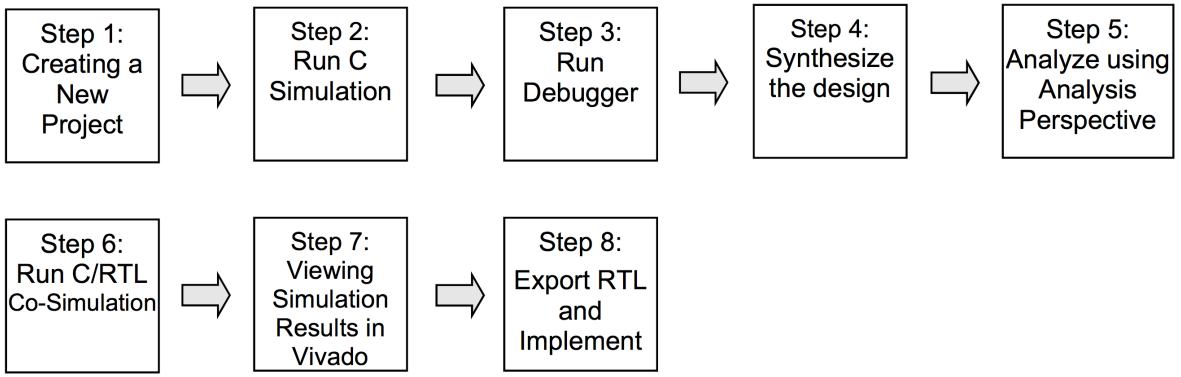


[XILL1]

High-Level Synthese
Vivado HLS

Ablaufplan des Programms

A. **C-Simulation**: Syntaxcheck des Algorithmus
B. **Debugging**: Verifikation des Testcodes
C. **C-Synthese**: Bauen des RTL-Designs
D. **Analyse**: Untersuchen des Designverhaltens
E. **C/RTL-Cosimulation**: Verifikation des RTL-Designs
F. **RTL exportieren**: Ausgabe des RTL-Codes



Für Auswertung

Fokus auf Punkte C. und D.

[XILL1]

High-Level Synthese
Pipelining zur Leistungssteigerung

Warum Pipelining

— Allgemein bekannt

— Signifikante Beschleunigung

Mehre Optionen zur Positionierung

— Äußere Schleife (Iteration über Zeilen)

— Mittlere Schleife (Iteration über Spalten)

— Innerste Schleife (Multiplikation und Summation der einzelnen Einträge)

```
7   // Iterate over the rows of the A matrix
8   Row: for(int i = 0; i < MAT_A_ROWS; i++) {
9     // Iterate over the columns of the B matrix
10    Col: for(int j = 0; j < MAT_B_COLS; j++) {
11      // Do the inner product of a row of A and col of B
12      res[i][j] = 0;
13      Product: for(int k = 0; k < MAT_B_ROWS; k++) {
14        res[i][j] += a[i][k] * b[k][j];
```

[XILL1]

High-Level Synthese
Auswertung (Zahlen)

Taktdauer

8,7 ns < 10ns ⇒ realisierbar

Latenz

— Umso weiter außen, desto geringer

— Grund: je mehr Unrolling, desto mehr Parallelisierung

Bewertungskriterium /gepipelinete Schleife	Keine	Innerste	Mittlere	Äußerste
Erreichbare Taktdauer/ns	8,7	8,7	8,7	8,7
Latenz/Taktzyklen	79	91	22	18
Gesamtbearbeitungszeit/ns	687,3	791,7	191,4	156,6
Beschleunigung	1	0,87	3,59	4,39
DSP48E	1	1	2	6
FF	44	56	85	484
LUT	184	277	325	393

Gesamtbearbeitungsdauer

— Erwartete Taktdauer * Anzahl Taktzyklen ⇒ proportional zur Latenz

— Beschleunigung: Mittlere > 3; Äußere > 4

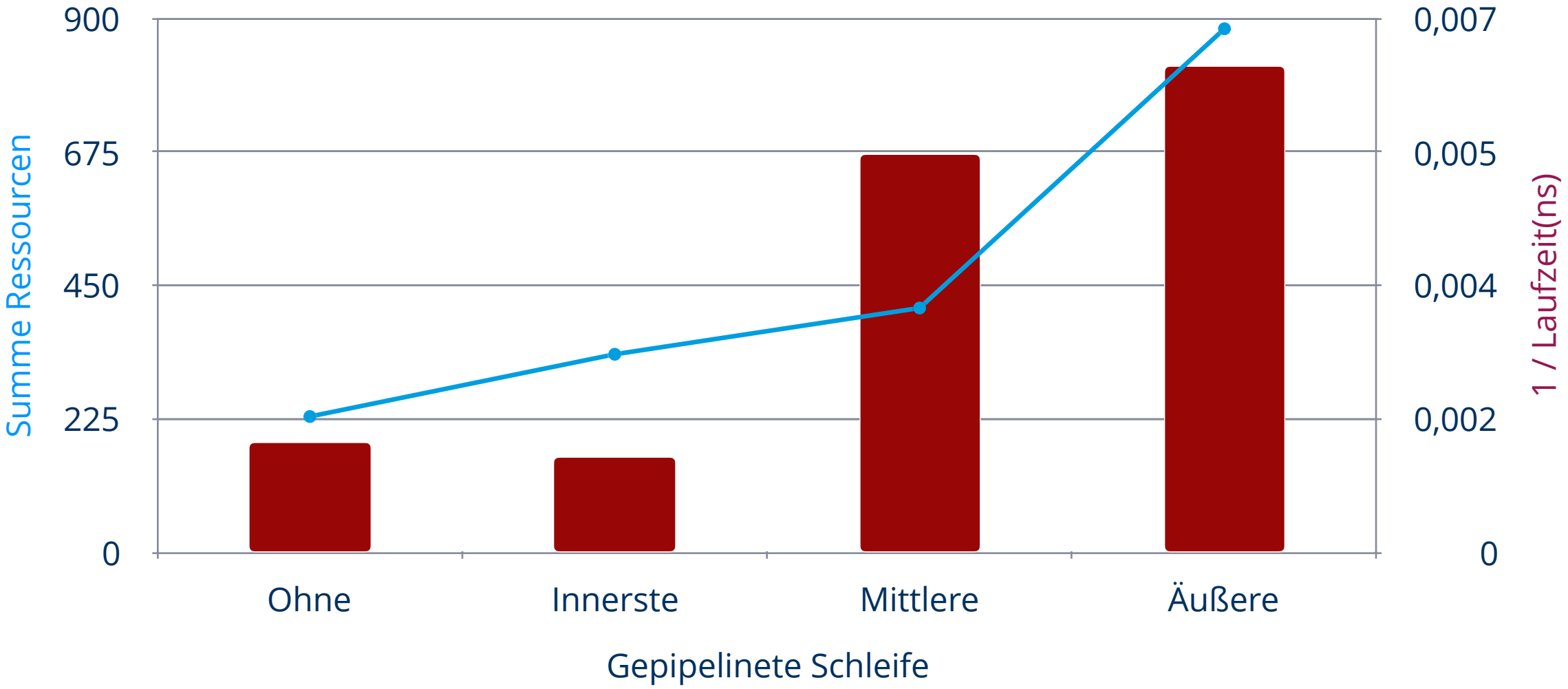
Ressourcen

— Aufwand steigt mit Bearbeitungsgeschwindigkeit

— Jedoch nie mehr als zweiprozentige Auslastung

High-Level Synthese

Auswertung (Diagramm)



Fazit & Ausblick

Fazit & Ausblick

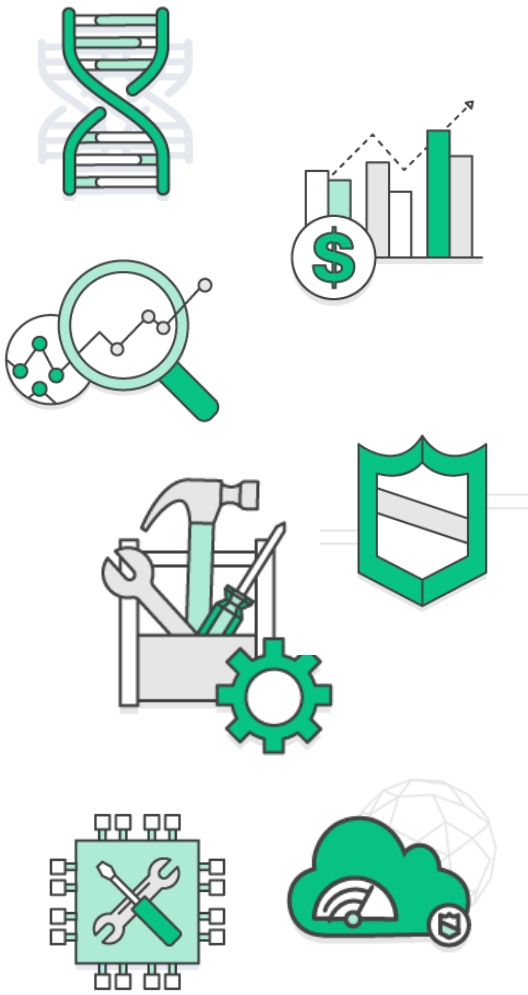
Rückschlüsse aus der Arbeit

Fazit

- HLS als effizientes Mittel im Architekturentwurf
- Entwicklungszeit wird reduziert
- Berechnung wird beschleunigt

Ausblick auf zukünftige Forschungsthemen

- Verwendung anderer Direktiven als Pipelining zur Beschleunigung?
- Ressourcenabhängige Begrenzung der Effizienz von HLS?



Amazon Web Services, Inc. [Ama18]

Quellen

Quellen

Abbildungen

[AVERZ] **mystipendium.de.** „Idee“ In: https://www.mystipendium.de/studium/einleitung-hausarbeit, *Online 04.07.18.*

[IDEE] **thesius.** „Das Abkürzungsverzeichnis in der Bachelorarbeit“ In: https://www.thesius.de/blog/articles/abkuerzungsverzeichnis/, *Online 04.07.18.*

[ZEDIO] **zedboard.org.** „Functional Overlay“ In: http://zedboard.org/product/zedboard, *Online 22.06.18.*

[CNXS] **CNXSOFT.** „Xilinx Zynq-7000 EPP Block Diagram“ In: https://www.cnx-software.com/2012/03/29/xilinx-zynq-7000-extensible-processing-platform-epp-dual-cortex-a9-fpga-soc/, *Online 05.07.18*

Dokumente

[HWUG] **Xilinx Inc.** *ZedBoard (Zynq Evaluation and Development) Hardware User’s Guide.* ZedBoard_HW_UG (v2.2), 02.07.18

[XILL1] **Xilinx Inc.** *Vivado HLS Design Flow Lab.* 01_Lab1, 13.06.18

Vielen Dank für eure Aufmerksamkeit!
Fragen?