Einführung in Computer Microsystems

3. Teil: Ein System-on-Chip am Beispiel der Xilinx Zynq-Architektur

Andreas Koch
FG Eingebettete Systeme und ihre Anwendungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Embedded Systems & Applications

Material

TECHNISCHE UNIVERSITÄT DARMSTADT

- Vorlesungsfolien basierend auf

  The Zynq Book
  von *Crockett/Elliot/Enderwitz/Stewart*
  Strathclyde Academic Media 2014

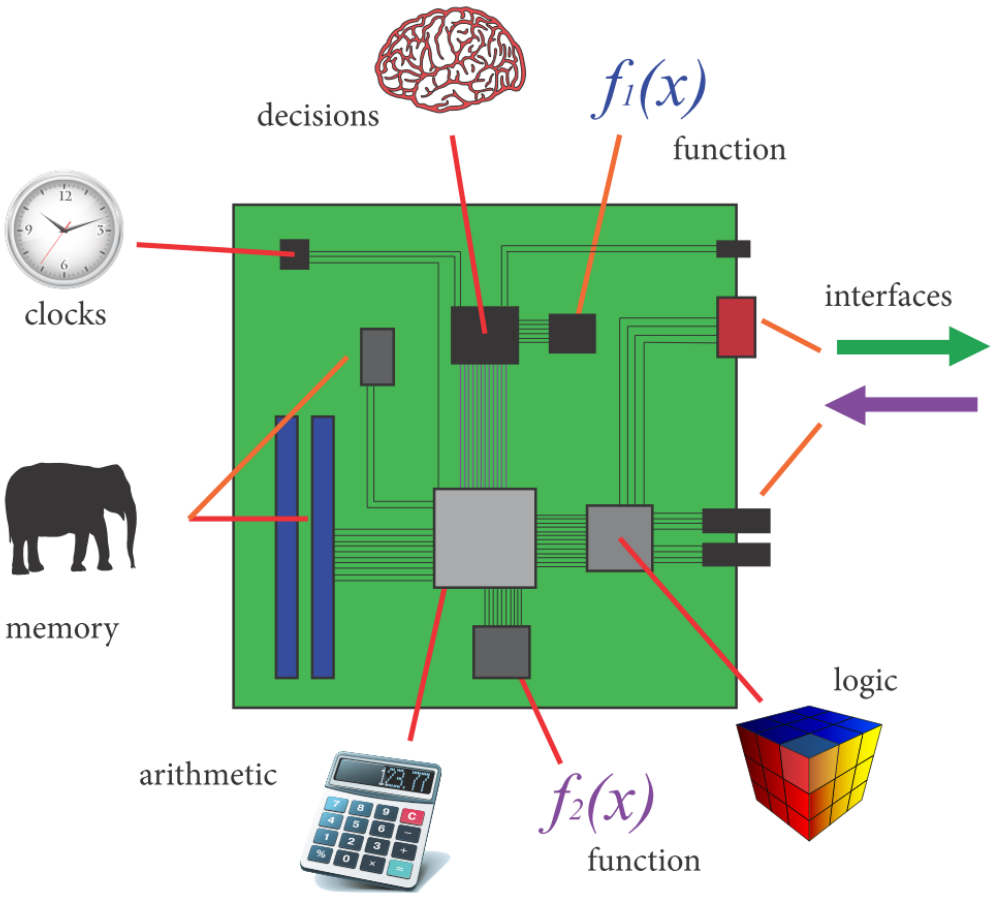
  - Frei verfügbar als Download von
    `http://www.zynqbook.com`

- Daraus auch die meisten Abbildungen in diesem Kapitel

- Weitere Abbildungen aus Dokumentation der Hersteller Xilinx und ARM

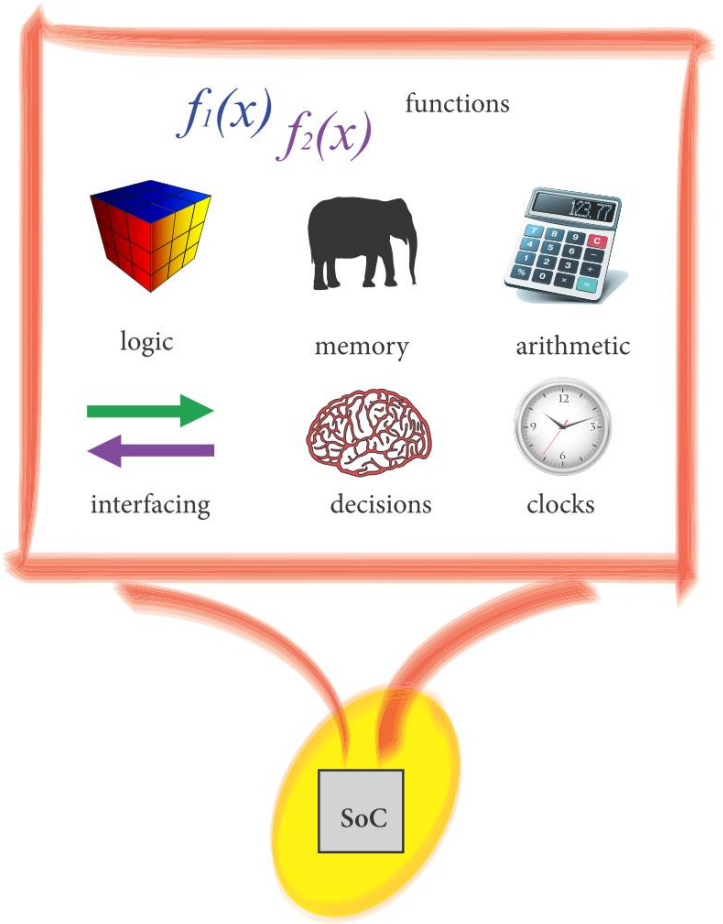
Rechnersystem auf Leiterplatte

System-on-Board



Rechnersystem auf Chip

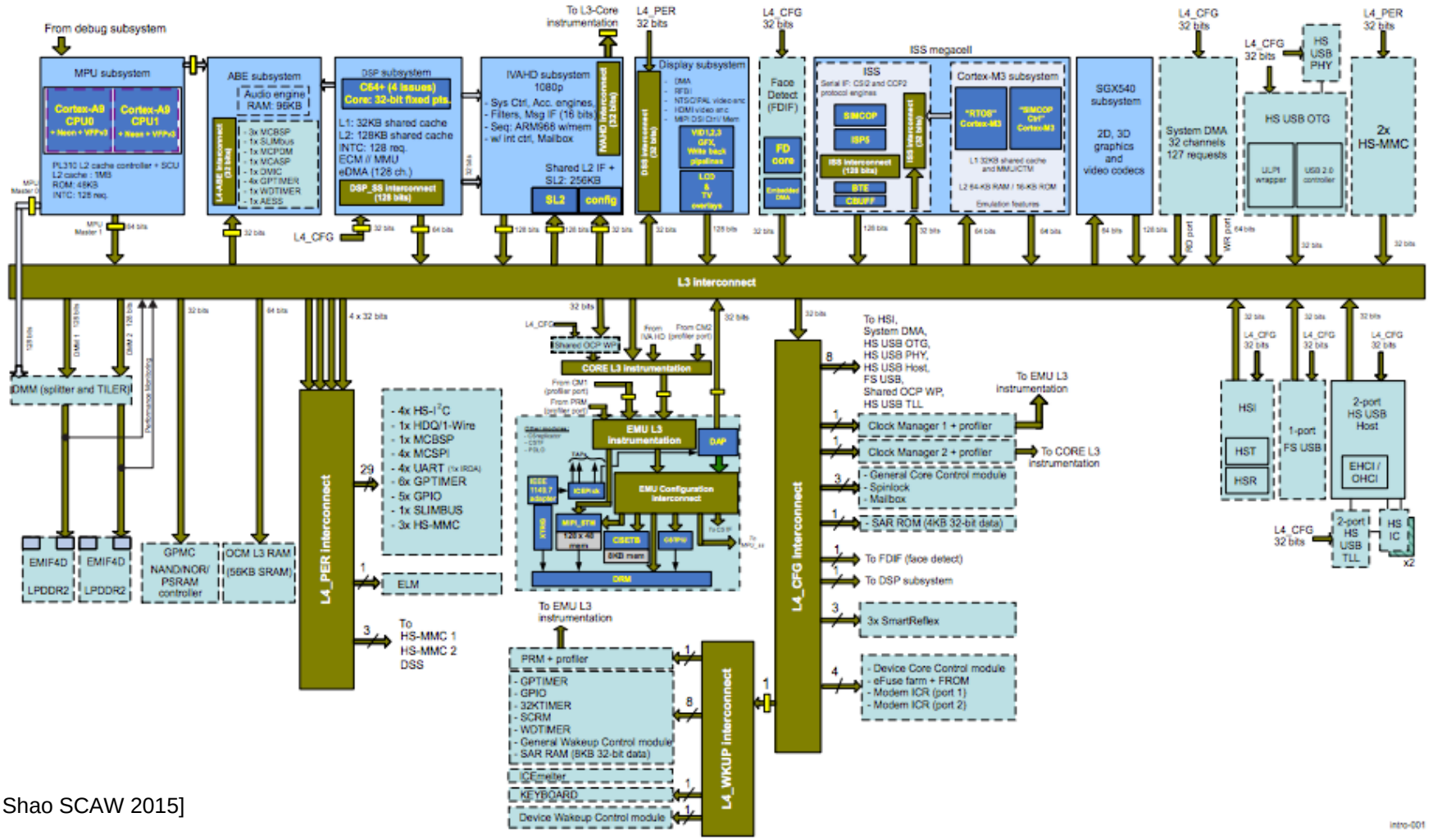
System-on-Chip



Modernes System-on-Chip

Texas Instruments OMAP 4

TECHNISCHE UNIVERSITÄT DARMSTADT

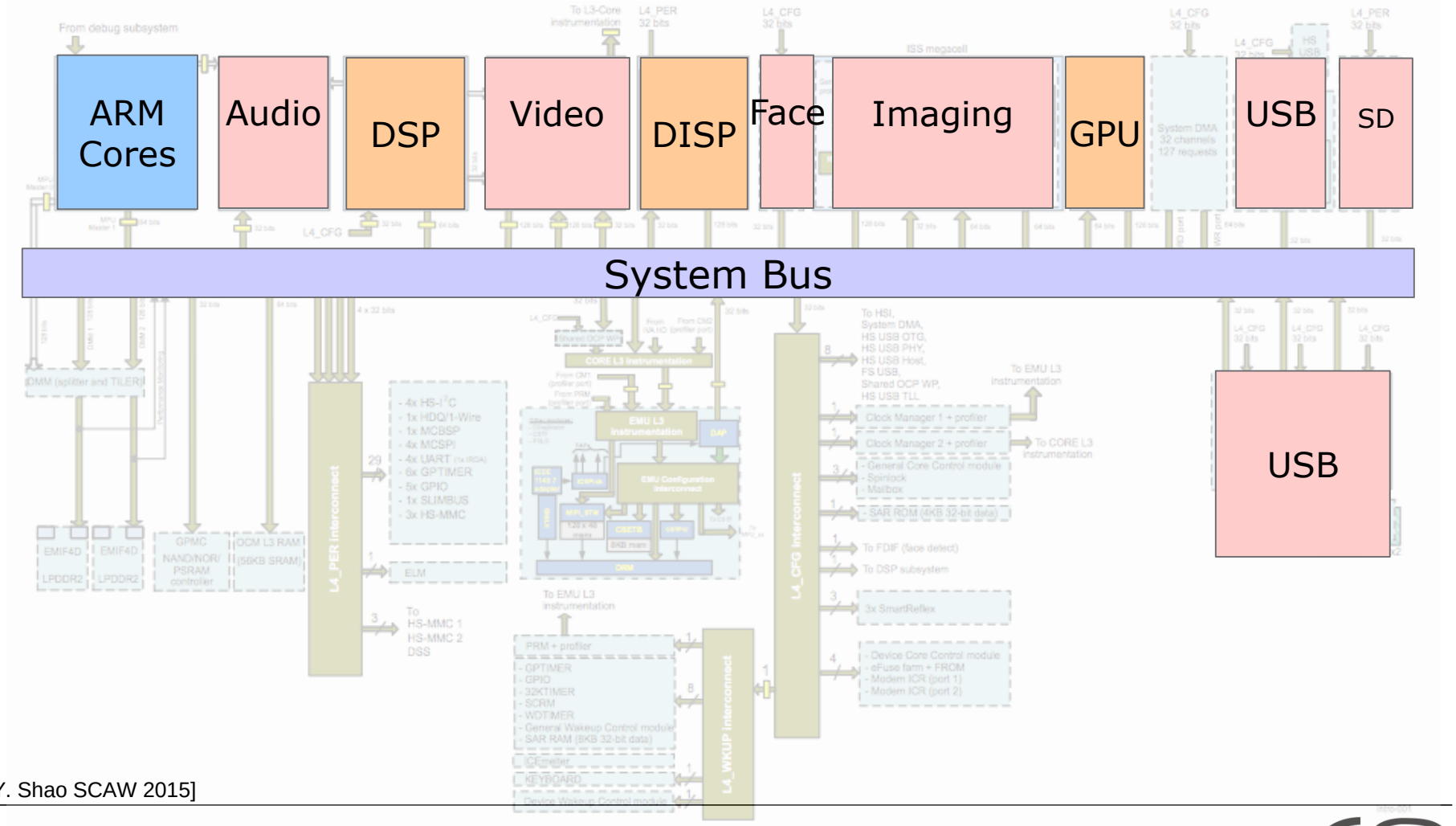


[Y. Shao SCAW 2015]

Embedded Systems & Applications

Modernes System-on-Chip

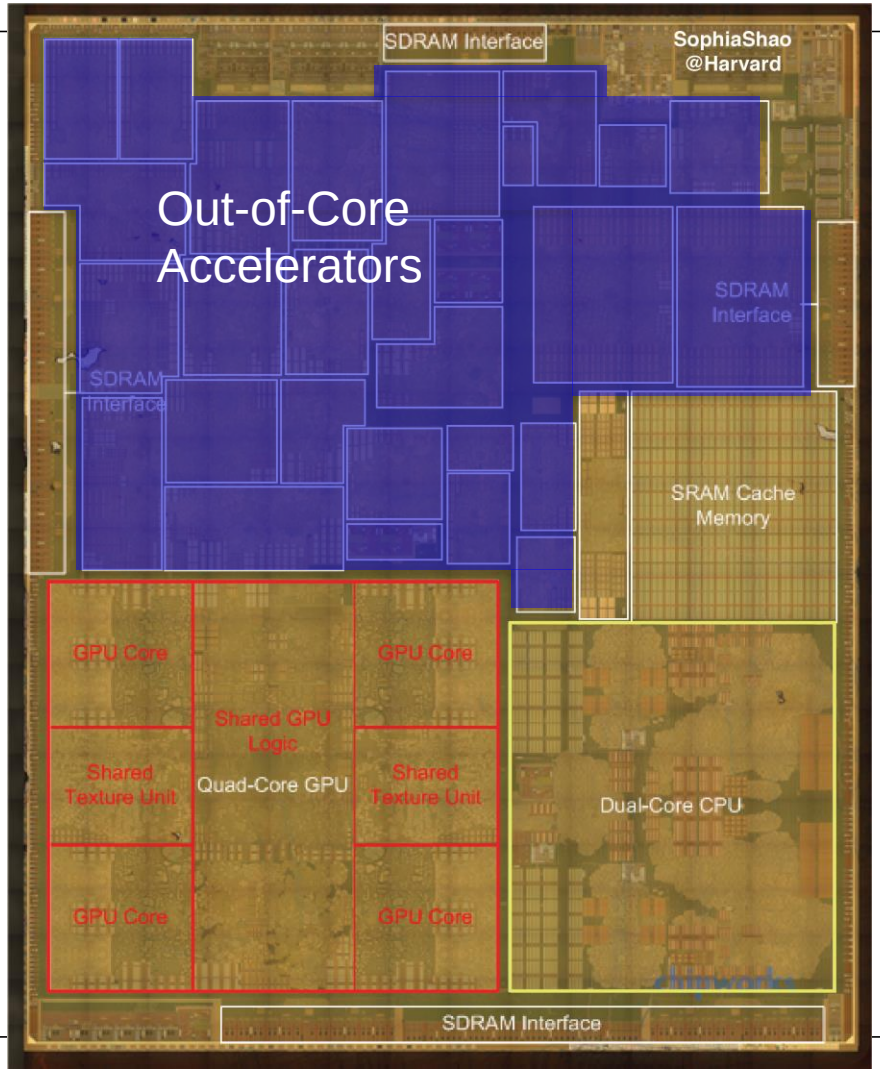
Wesentliche Komponenten

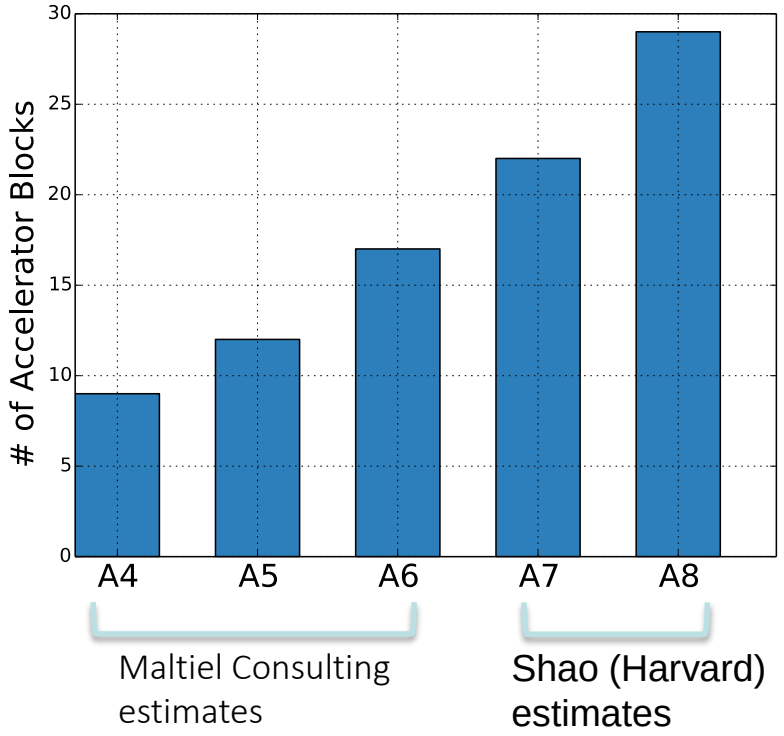


[Y. Shao SCAW 2015]

Weiteres Beispiel: Apple A8 SoC

Diverse spezialisierte Recheneinheiten





[Die photo from Chipworks]
[Accelerators annotated by Sophia Shao @ Harvard]

Rekonfigurierbares System-on-Chip

TECHNISCHE UNIVERSITÄT DARMSTADT

- System-on-Chip ist nützlich

- Problem
  - Genau welche Komponenten wie integrieren?
  - Unterschiedliche Nutzer haben unterschiedliche Anforderungen
  - Ein eigener Chip je Anforderungsprofil ist in der Regel zu teuer
    - Hohe Kosten der ASIC-Fertigung
    - Häufig nur geringe Stückzahlen

- Idee
  - Rekonfigurierbares System-on-Chip
  - Kombiniert
    - Feste Teile, hoffentlich breitflächig einsetzbar
    - Variable rekonfigurierbare Teile für Erfüllen spezieller Anforderungen
    - Damit größere Anzahlen von breitflächig einsetzbaren Chips herstellbar

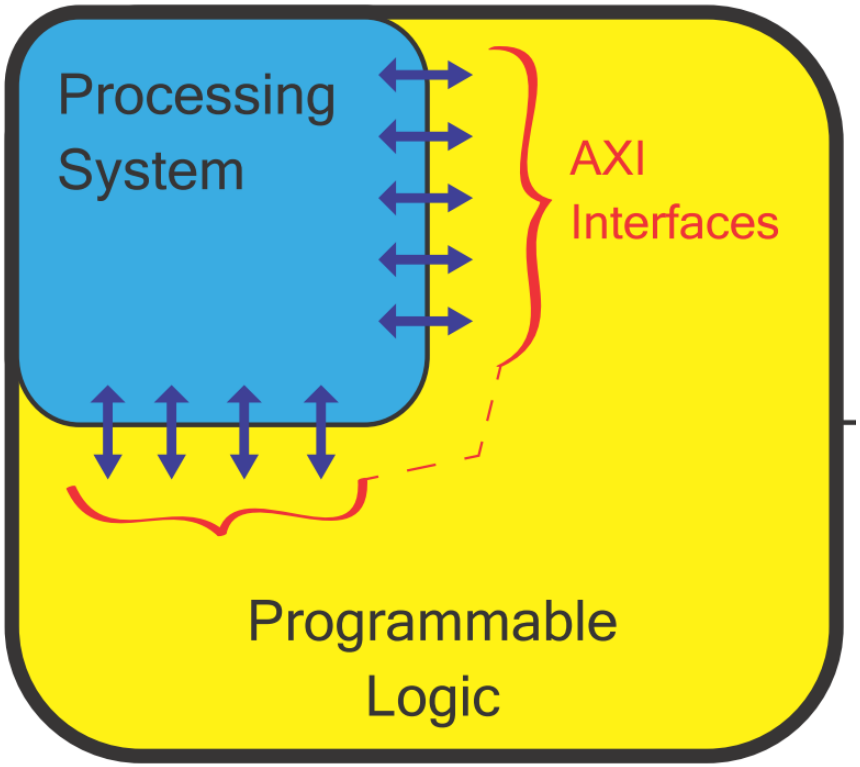
XILINX ZYNQ 7000 RSOC

Rekonfigurierbares System-on-Chip

TECHNISCHE UNIVERSITÄT DARMSTADT

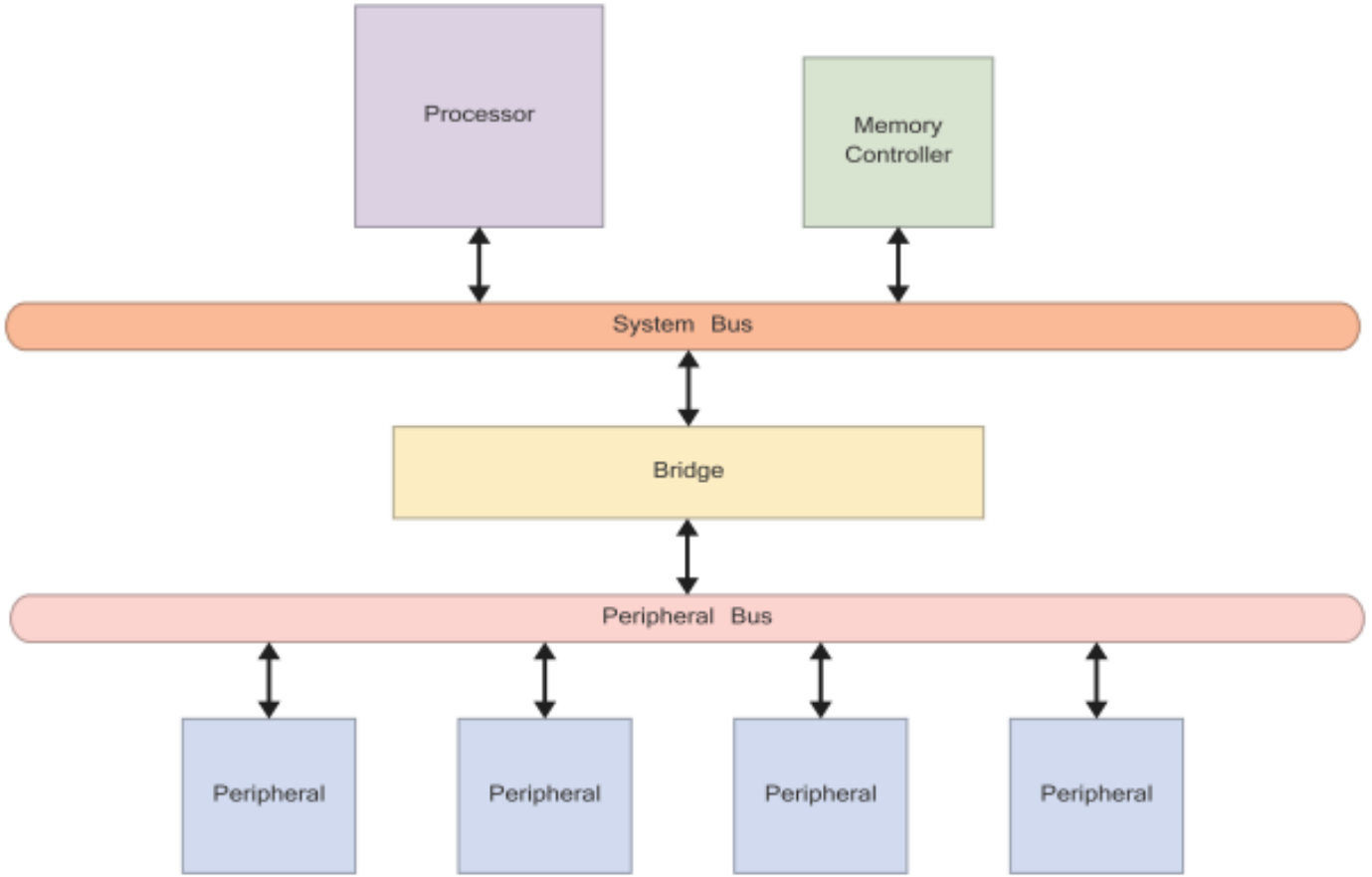
- *Reconfigurable System-on-Chip (rSoC)*

- Von verschiedenen Herstellern angeboten
  - Xilinx Zynq 7000 und UltraScale+
  - Altera Cyclone / Arria / Stratix
  - Microsemi SmartFusion2
  - …

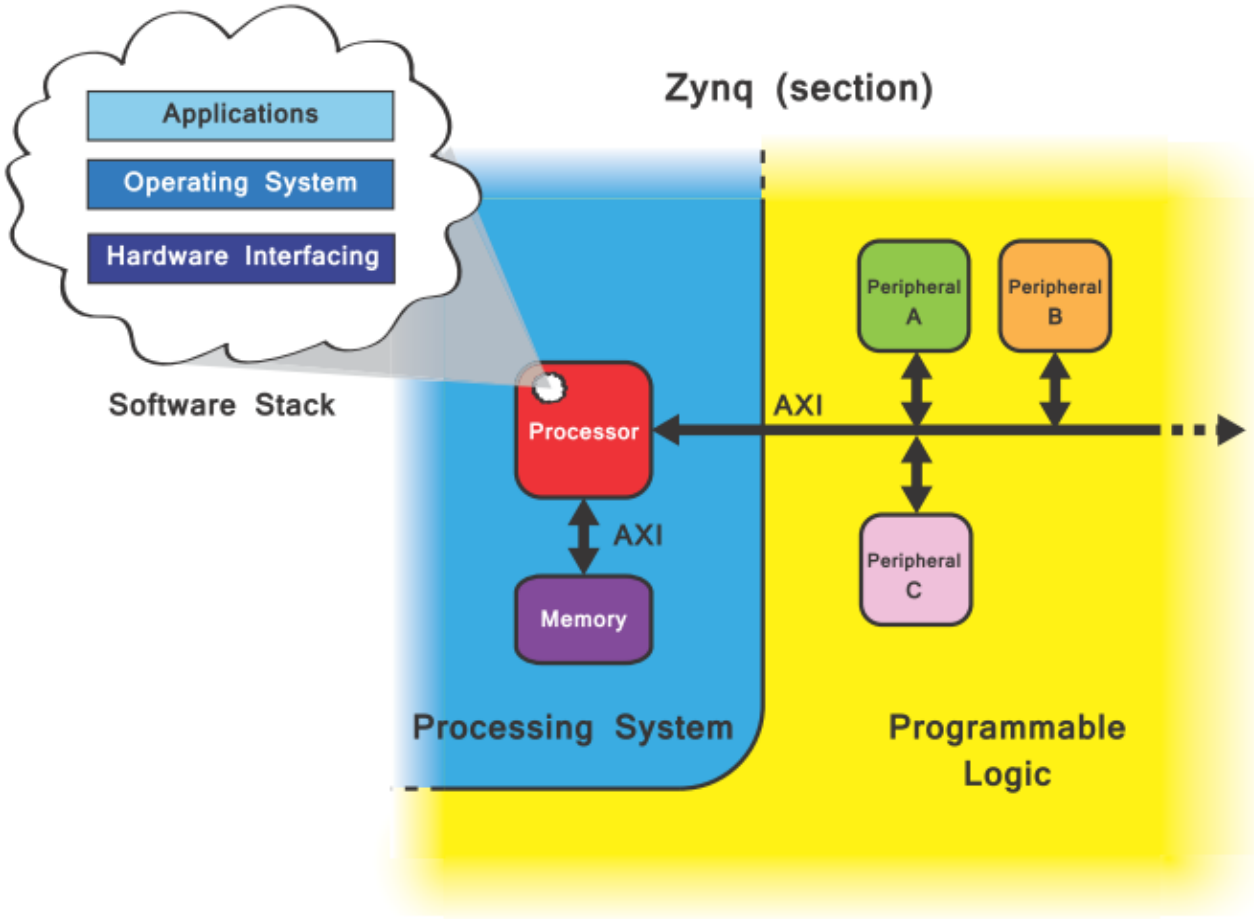


System-on-Chip: Basisarchitektur

TECHNISCHE UNIVERSITÄT DARMSTADT



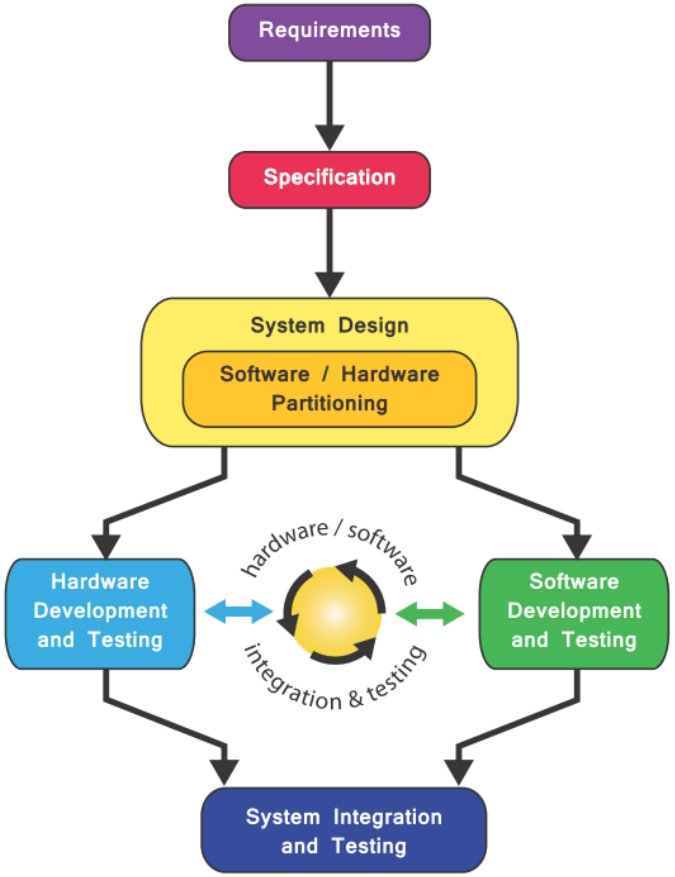
Basisarchitektur auf rSoC abgebildet



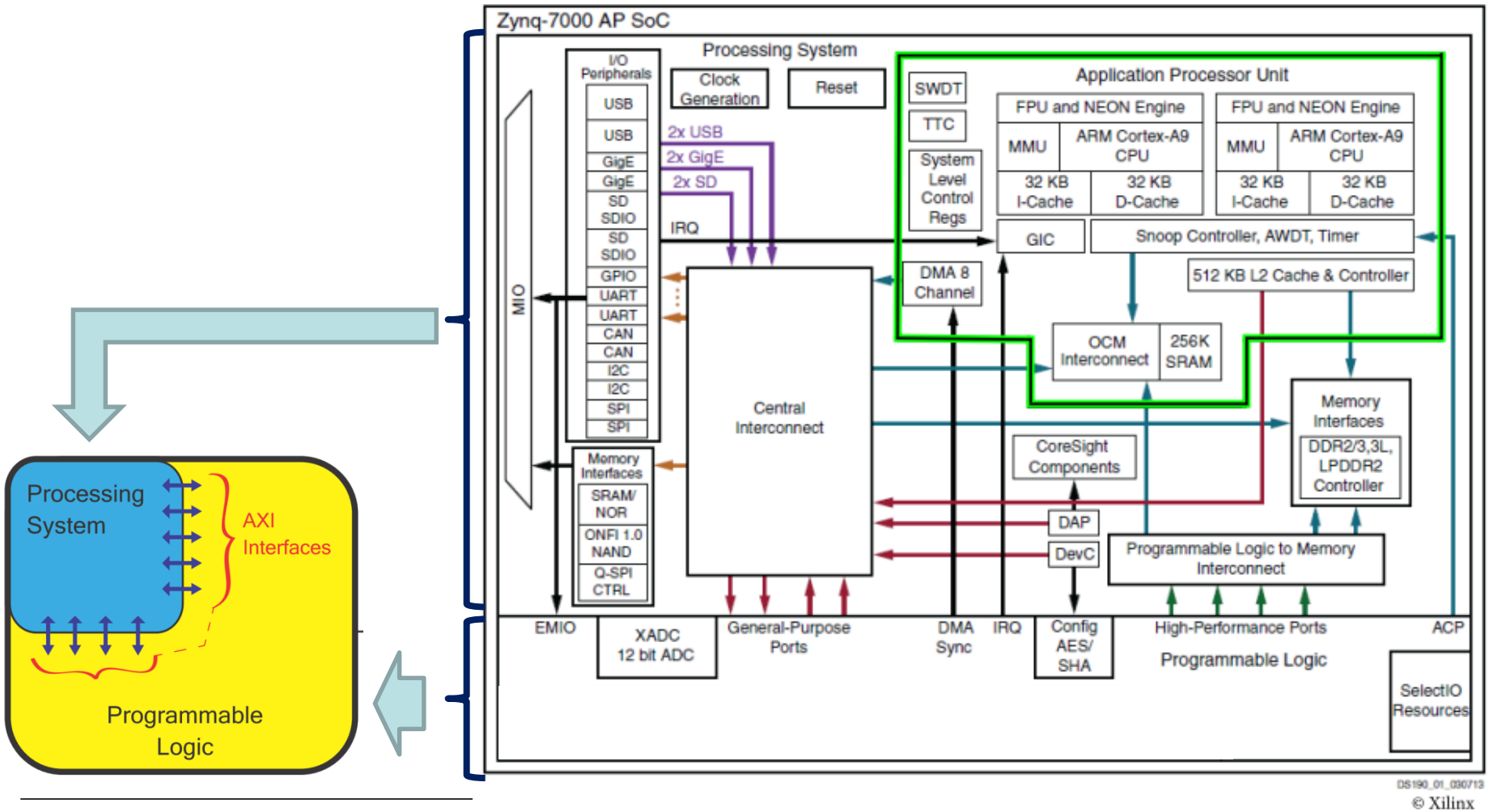
Entwicklung für rSoC

TECHNISCHE UNIVERSITÄT DARMSTADT

- Umfaßt Schritte der
  - Hardware-Entwicklung
  - Software-Entwicklung
- Häufig problematisch
  - Integration von HW/SW-Komponten
  - Debugging über HW/SW-Grenze hinweg



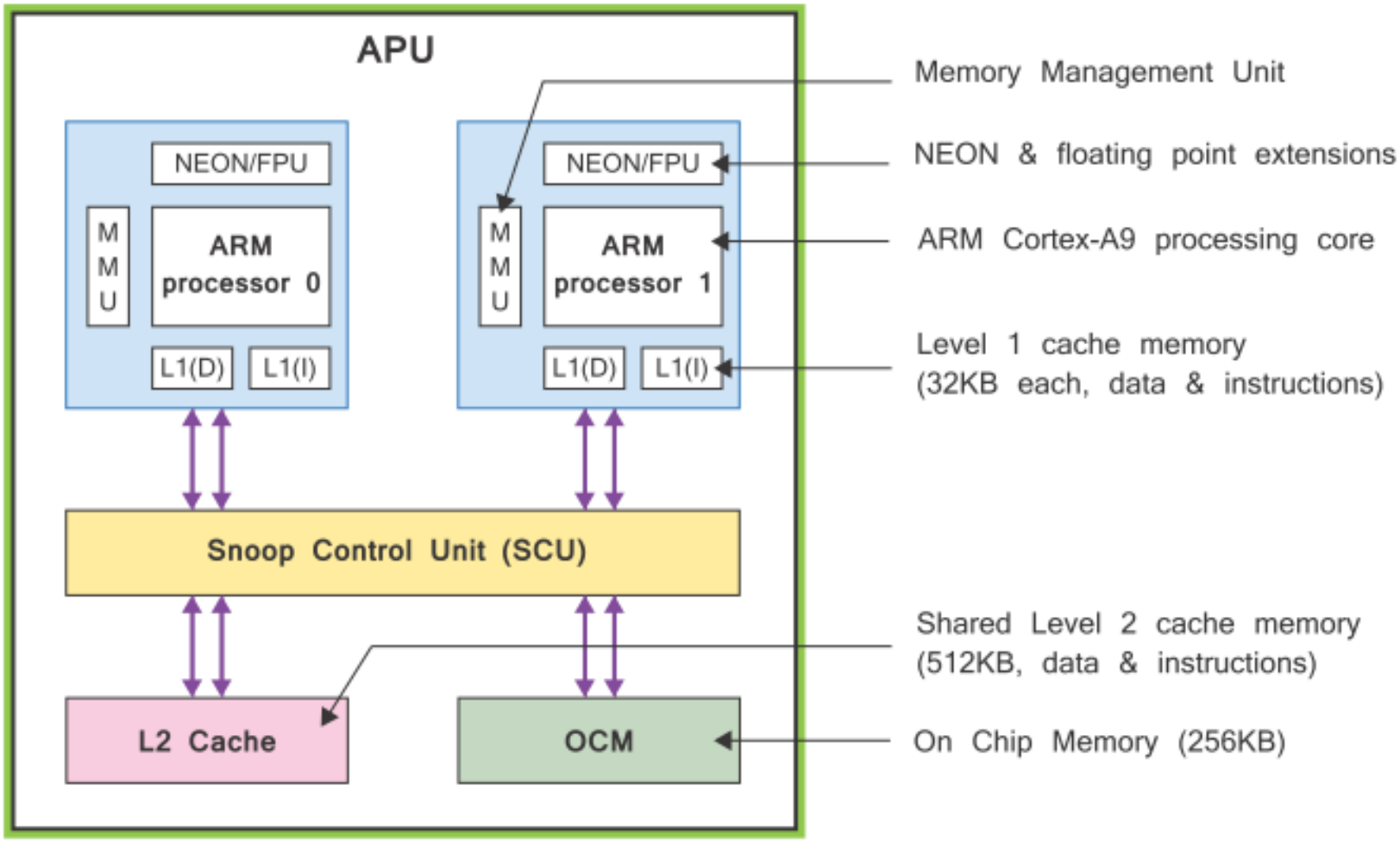
Zynq 7000 rSoC Architektur



Software-Programmierbare Prozessoren

Application Processing Unit

TECHNISCHE UNIVERSITÄT DARMSTADT



Embedded Systems & Applications

ARM Cortex-A9 Prozessorkern

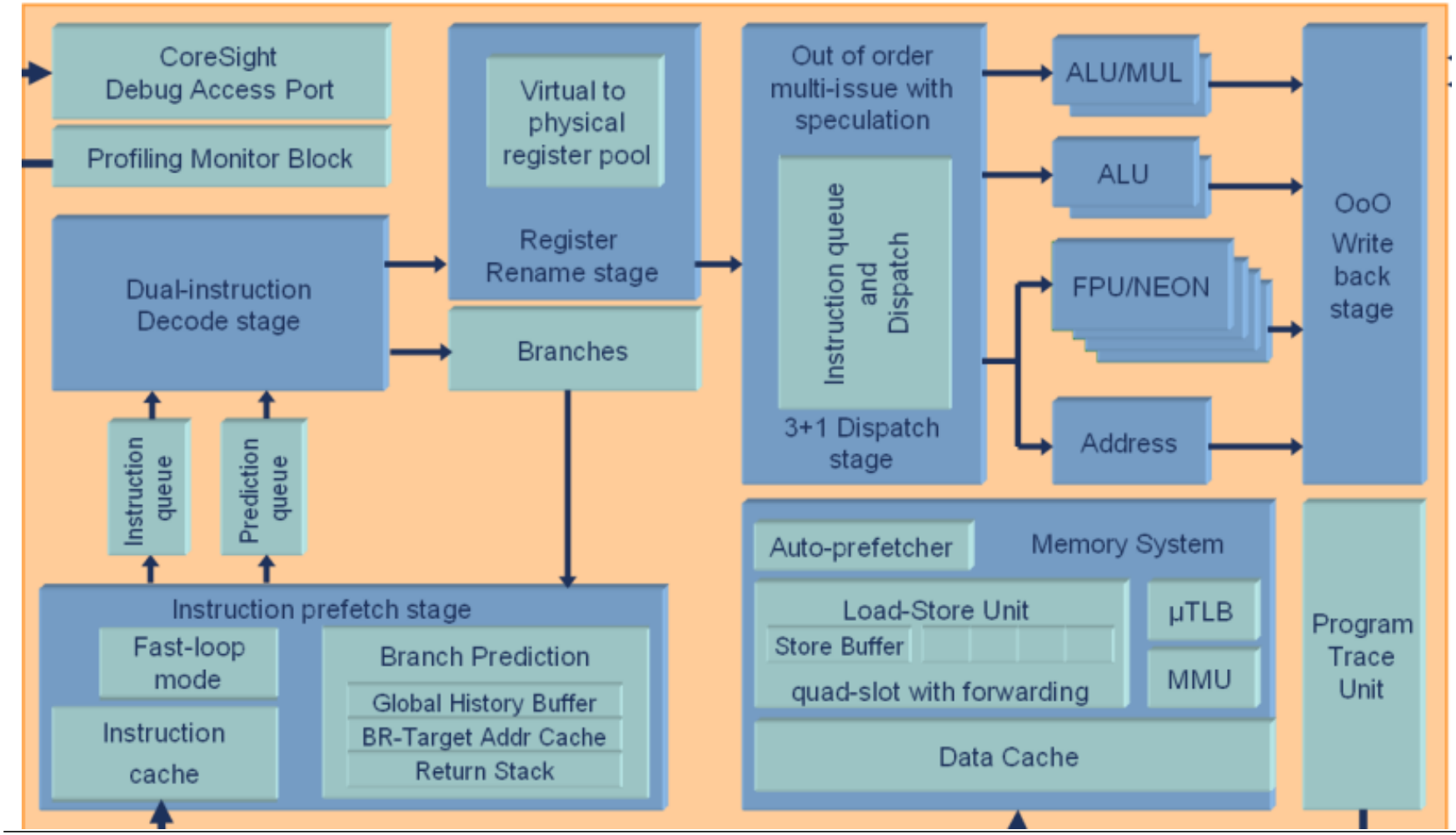
TECHNISCHE UNIVERSITÄT DARMSTADT

- Superskalar out-of-order
- Holt zwei Instruktionen je Takt
- Kann je Takt bis zu vier Instruktionen ausführen
  - ALU/MUL
  - ALU
  - FPU/SIMD
  - Load-Store

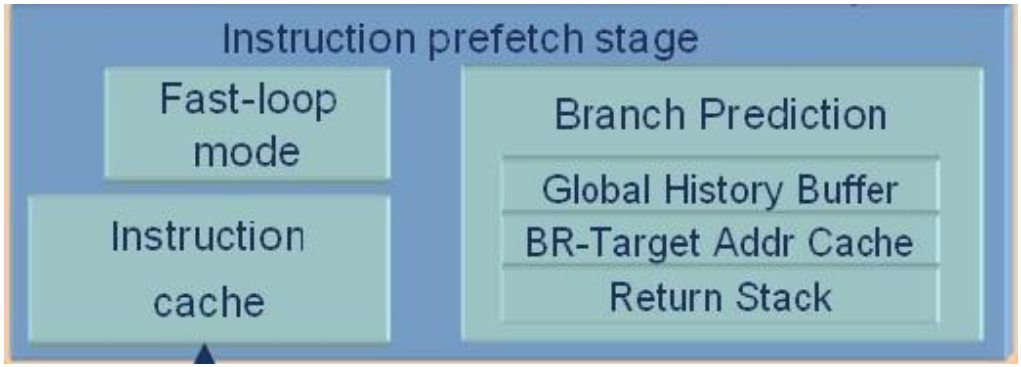
- Mehr ILP durch (→ TGDI)
  - Umbenennen von Registern (*renaming*)
  - Dynamisches Vorziehen von unabhängigen Instruktionen (*out-of-order execution*)

Interner Aufbau

8…11 Pipeline Stufen aufgeteilt in 7 Phasen: FDRIEMW



Instruction Fetch

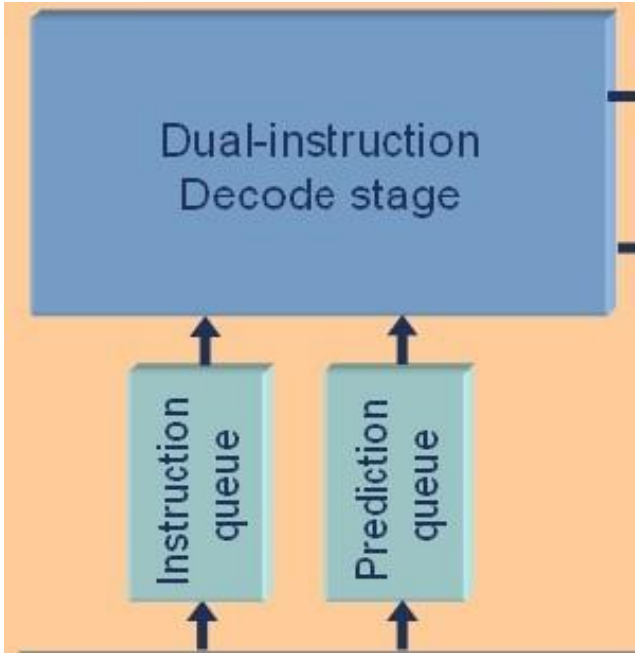


- Instruction Cache (I$): 16 KB, **32 KB (im Zynq 7000)**, 64 KB
- Sprungvorhersage
  - Branch Target Address Cache
    - 256 Einträge mit Zieladressen für Taken/Untaken; 2b Vorhersage (strongly/weakly taken/not taken)
  - Global History Buffer: 4096 Einträge, je 2b (strongly/weakly taken/not taken)
  - Return Address Stack: 8 Einträge
- Fast-Loop Mode
  - Schleifen < 64 B können oftmals ohne I$-Zugriffe ausgeführt werden, spart Energie

Instruktionsdekodierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kann bis zu zwei Instruktionen je Takt dekodieren



Umbenennen von Registern

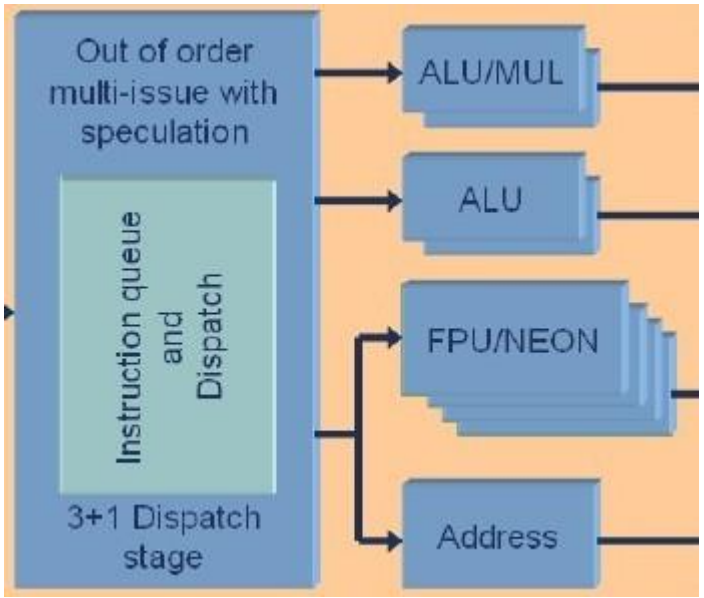
TECHNISCHE UNIVERSITÄT DARMSTADT

- Zum Auflösen von Abhängigkeiten
- Übersetzt virtuelle Registernamen in physikalische Register
  - #phys.Reg > #virt.Reg
- Beispiel kommt noch …

Instruktionen ausgeben

- Akzeptiert zwei dekodierte Instruktionen je Takt
- Kann bis zu vier Instruktionen an Ausführungseinheiten ausgeben (*dispatch*)
- Dabei umsortieren von Instruktionen

- Beispiel kommt noch …



Renaming und OoO-Ausführung

MIPS-Beispiel aus TGDI

TECHNISCHE UNIVERSITÄT DARMSTADT

```
lw  $t0, 40($s0)

add $t1, $t0, $s1

sub $t0, $s2, $s3

and $t2, $s4, $t0

or  $t3, $s5, $s6

sw  $s7, 80($t3)
```

- Im Beispiel-MIPS: 2 Takte Latenz für Load

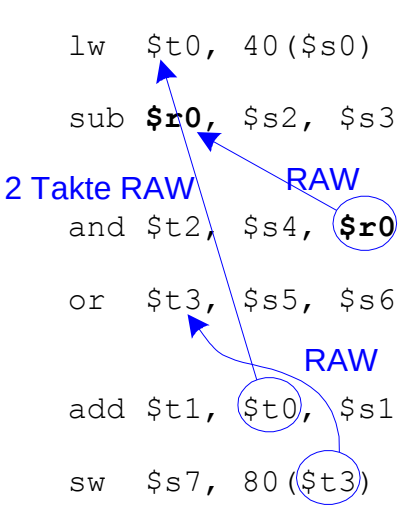
Renaming und OoO-Ausführung

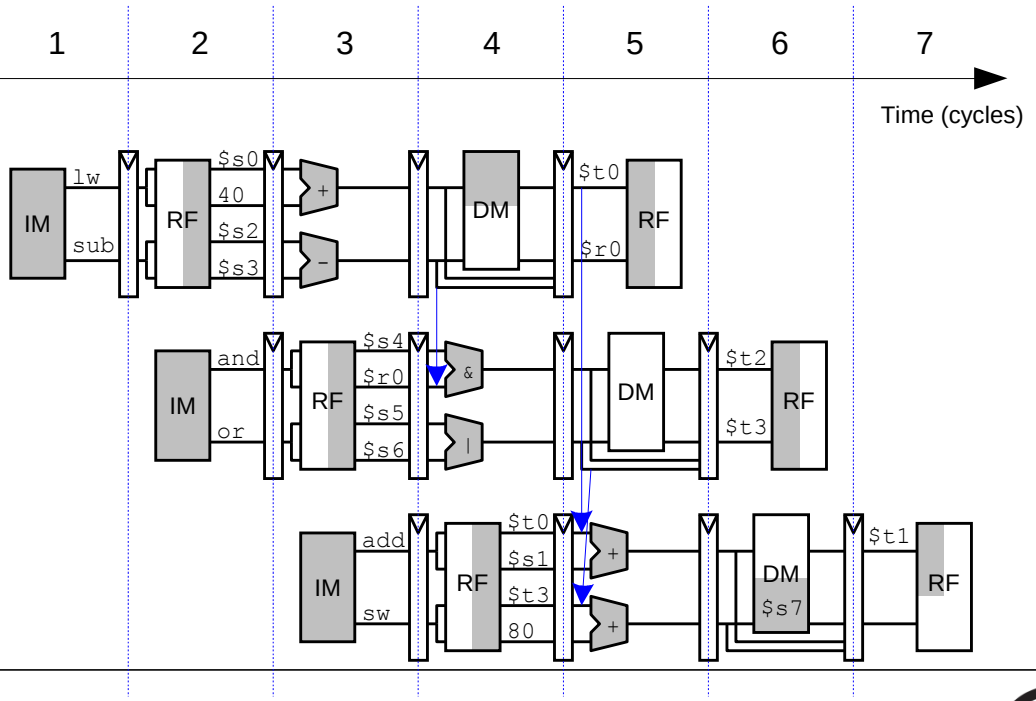
MIPS-Beispiel aus TGDI

```
lw  $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or  $t3, $s5, $s6
sw  $s7, 80($t3)
```

Idealer IPC-Wert: 2,0

Erreichter IPC-Wert: 6/3 = 2,0

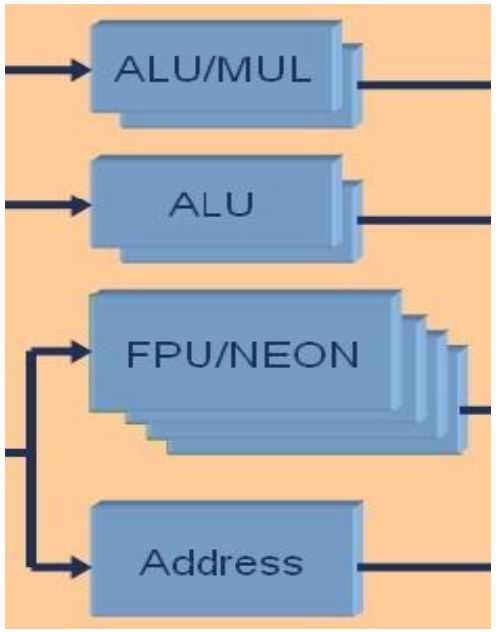




Ausführungseinheiten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Für unterschiedliche Arten von Instruktionen
  - Beispiel
    - Zwei Additionen je Takt
    - Aber nur eine Multiplikation je Takt



- ALUs haben eine Latenz von 1..3 Takten
  - Länger bei Shift/Rotate
  - **`ADD r0, r1, r2`** (1 Takt)
  - **`ADD r0, r1, r2 LSL #2`** (2 Takte)
    - r0 = r1 + (r2 << 2)
  - **`ADD r0, r1, r2 LSL r3`** (3 Takte)
    - r0 = r1 + (r2 << r3)

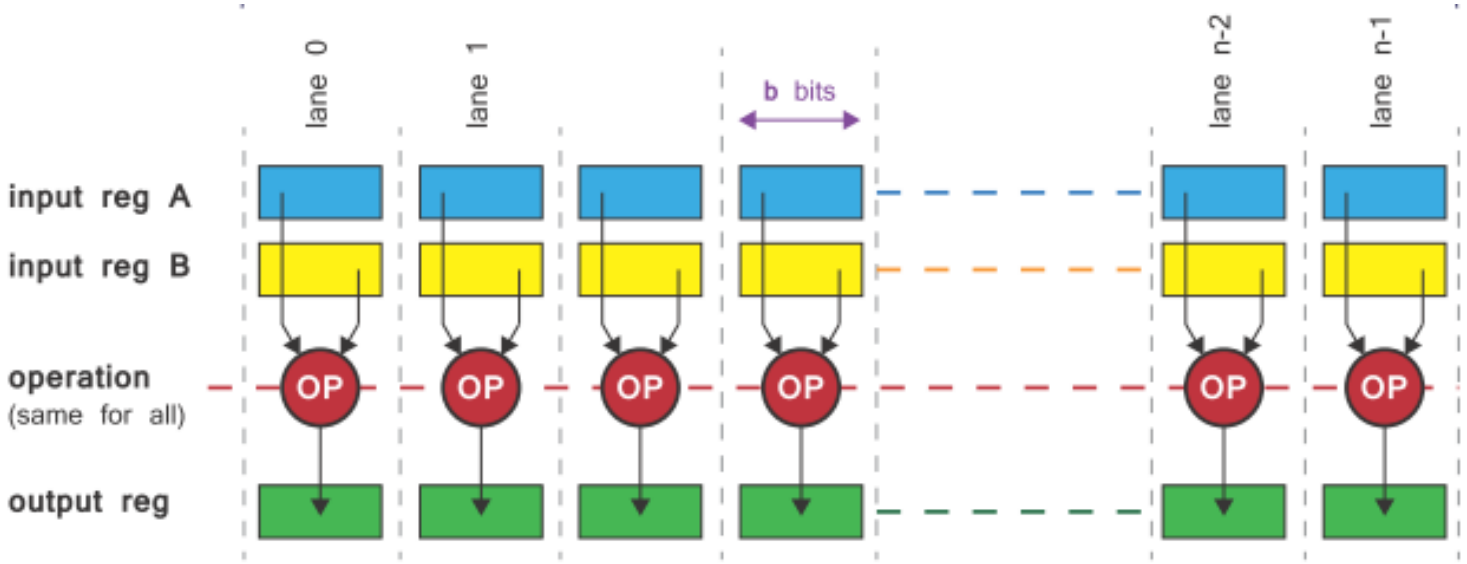
- NEON hat Latenz von 1 … 32 Takten
  - 1 Takt: VABS
  - 32 Takte: VSQRT

SIMD Rechnungen mit NEON

Vergleichbar zu SSE auf x86 CPUs

TECHNISCHE UNIVERSITÄT DARMSTADT

- *Single Instruction Multiple Data*
  - Die gleiche Operation wird auf *N* Datenelementen gleichzeitig ausgeführt



  - Skalare Datentypen: 8b/16b/32b/64b Integer, 32b Single-Precision Floating Point
  - Vektoroperationen sind insgesamt 128b breit, damit *N*=2…16
- Wird aber auch als normale FPU (auf skalaren Daten) genutzt

Speichersystem

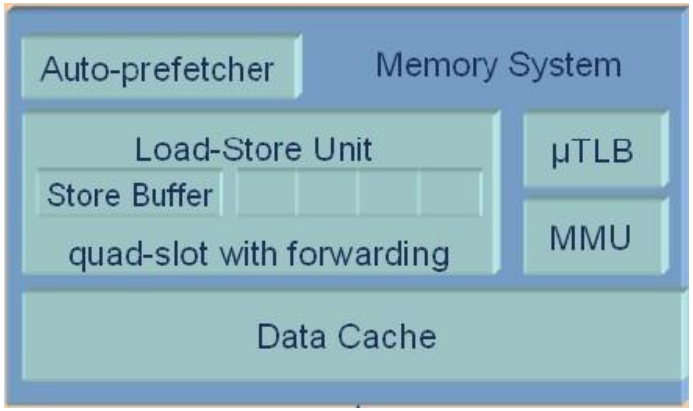
TECHNISCHE UNIVERSITÄT DARMSTADT

Adressübersetzung

- Zwei stufiges Verfahren
  - Micro TLB: 1 Takt Latenz
  - TLB in MMU: Variable Latenz

Prefetching

- Beobachtet Speicherzugriffsverhalten des Prozessors
- Holt schonmal „vorweg“ Daten in L1-D$
- Beobachtet dabei bis zu acht unabhängige Datenströme



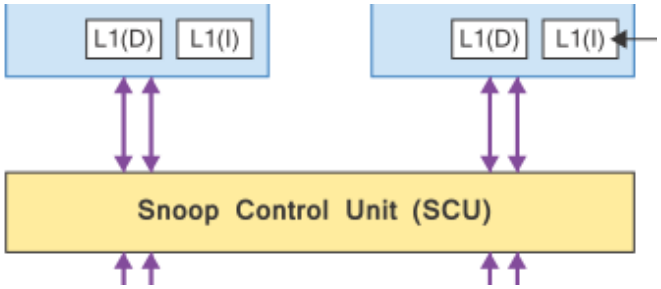
Vier aktive Speicheranfragen im D$

- Store-to-Load Forwarding

Cache-Kohärenz

TECHNISCHE UNIVERSITÄT DARMSTADT

- Problem
  - Prozessorkerne haben eigene L1-Caches
  - Greifen aber auf gemeinsamen Hauptspeicher zu
    - … und lagern Daten in ihrem lokalen L1-Cache zwischen
    - Auch eventuell veränderte!

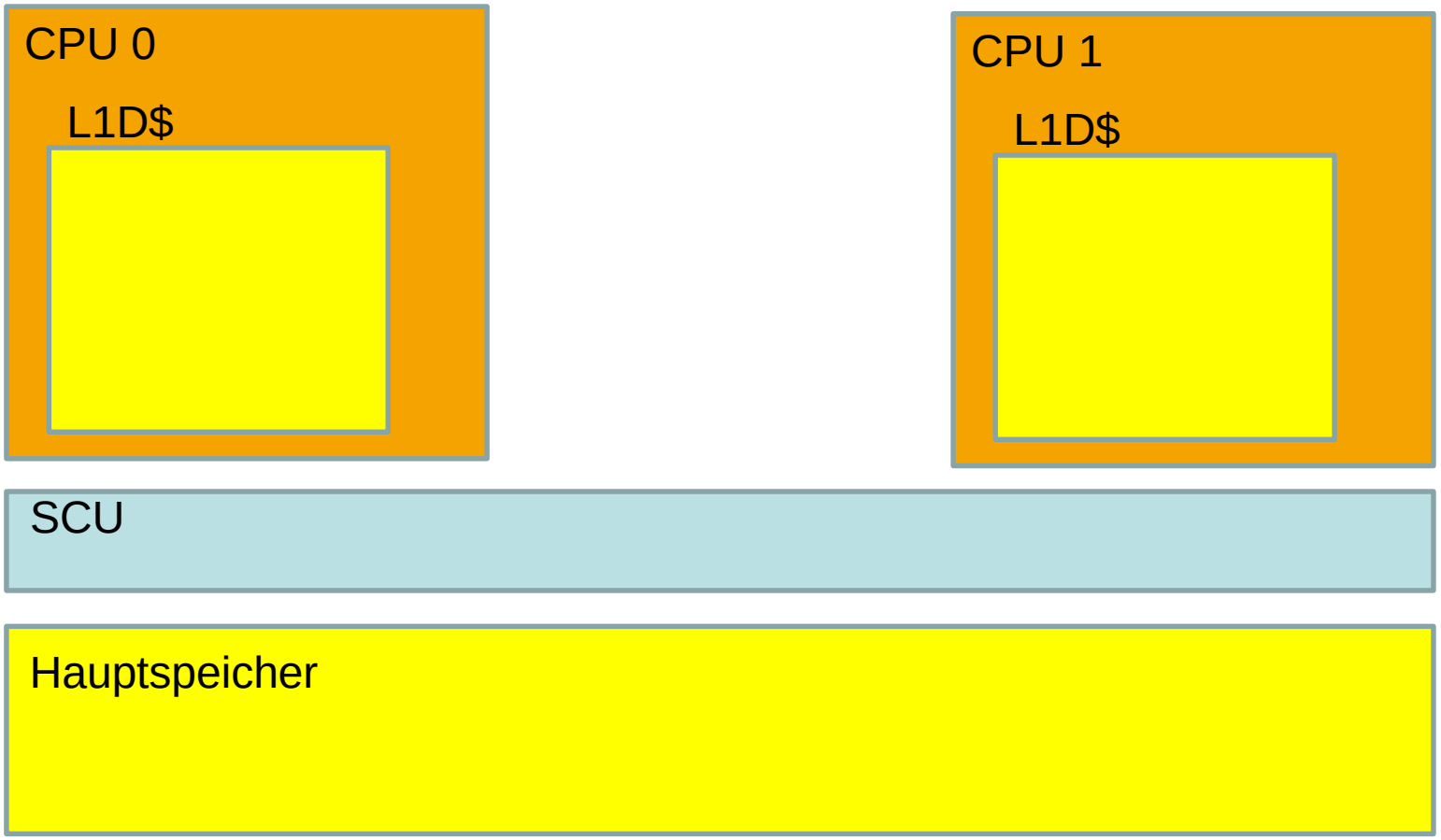


- Prozessorkerne müssen sich austauschen darüber, wo aktueller Wert liegt
  - … noch im Hauptspeicher
  - … schon in einem Cache (L1 oder L2)
  - … bereits in aktualisierter Form in einem Cache

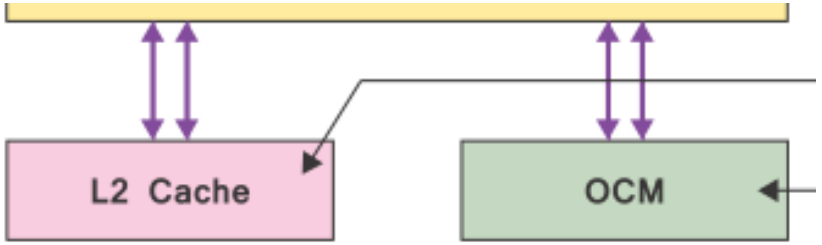
- Snoop Control Unit
  - Überwacht die Speicherzugriffe aller Prozessorkerne (und der Programmable Logic …)
  - Gibt aktuellen Wert weiter
  - Protokoll: Modified/Exclusive/Shared/Invalid (MESI)

Beispiel: Cache-Kohärenz

Hier stark vereinfacht und L2$ ignoriert



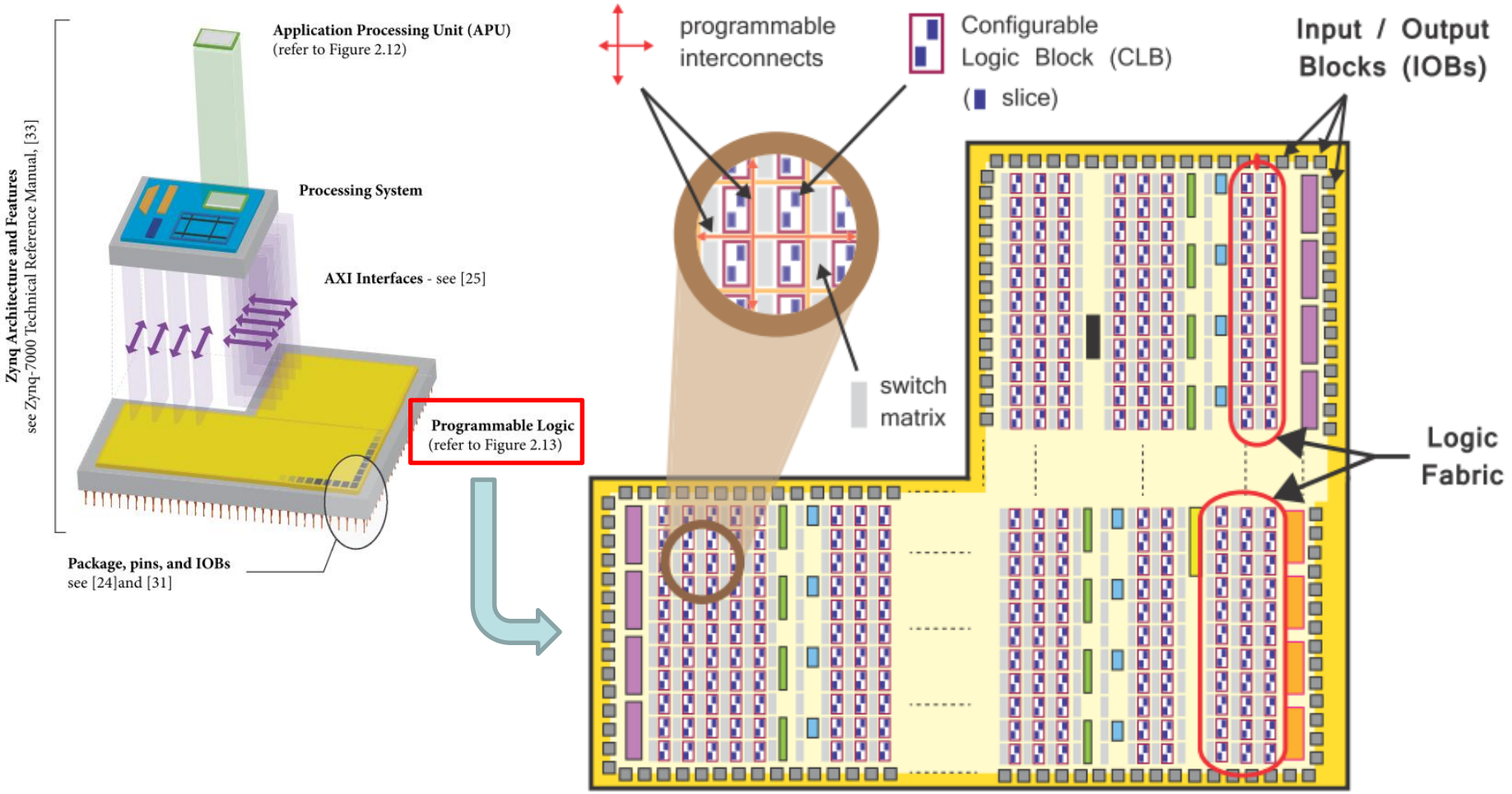
APU On-Chip Memory (OCM)



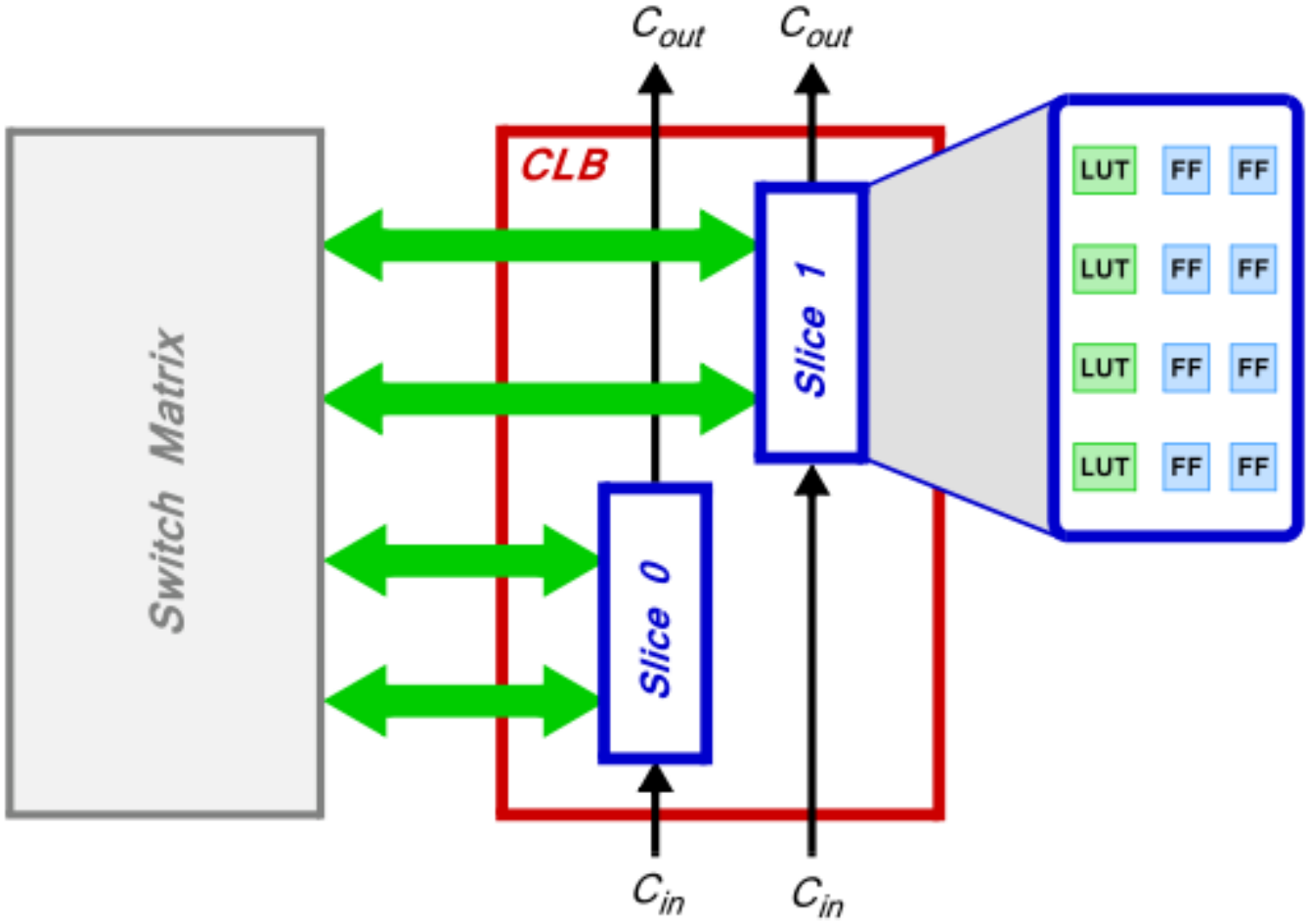
- 256 KB SRAM direkt auf dem Chip
- Nicht sonderlich schnell
  - Liest mit ca. 1.400 MB/s
  - Zum Vergleich: Externes DDR3-SDRAM 1066 liest in der Praxis mit ca. 3.700 MB/s
- OCM hat aber geringere und weniger schwankende Zugriffslatenz
  - In der Regel 32-34 Taktzyklen
  - Zum Vergleich: Externes DDR3-SDRAM hat zwischen 37-200 Zyklen

Programmierbare Logik

Integriertes FPGA (siehe TGDI)

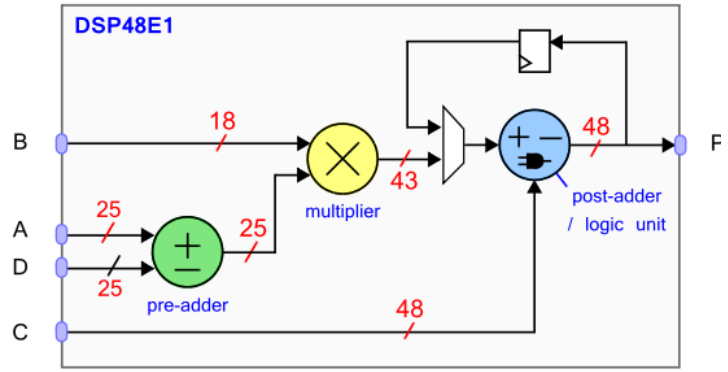


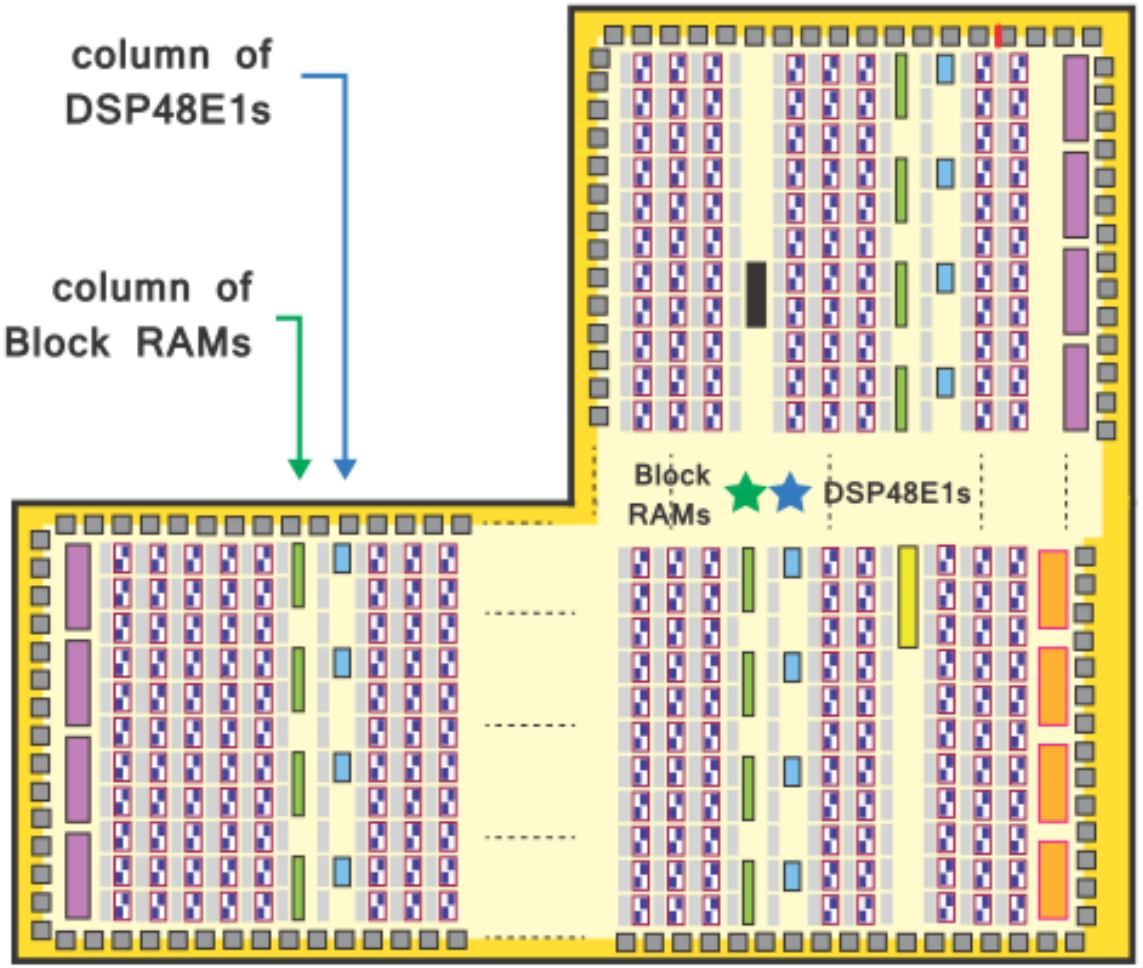
Konfigurierbarer Logikblock



Integrierte Speicher und Multiplizierer

BlockRAMs und DSP48E1-Blöcke





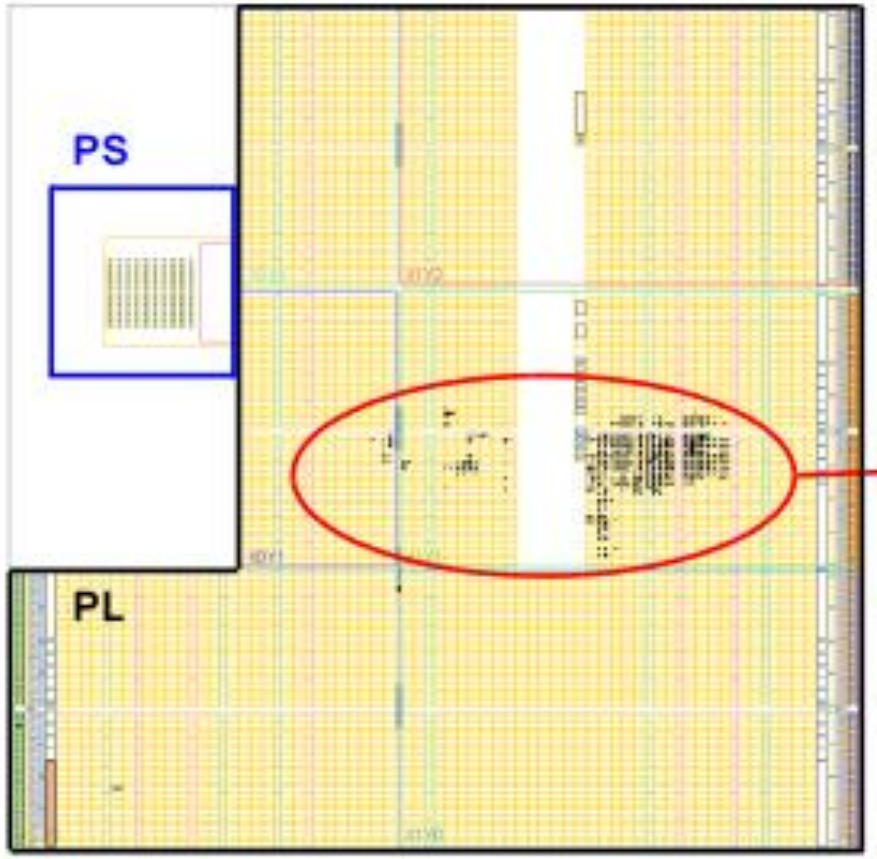
BlockRAMs: Je 36 Kb

- 1x 36 Kb oder 2x 18 Kb
- Breite konfigurierbar
  - 4K x 9b, 8K x 4b,
    1K x 36b,
    512 x 72b, …
- Alle parallel zugreifbar!

Prozessor in programmierbarer Logik

Soft Core Processor

TECHNISCHE UNIVERSITÄT DARMSTADT

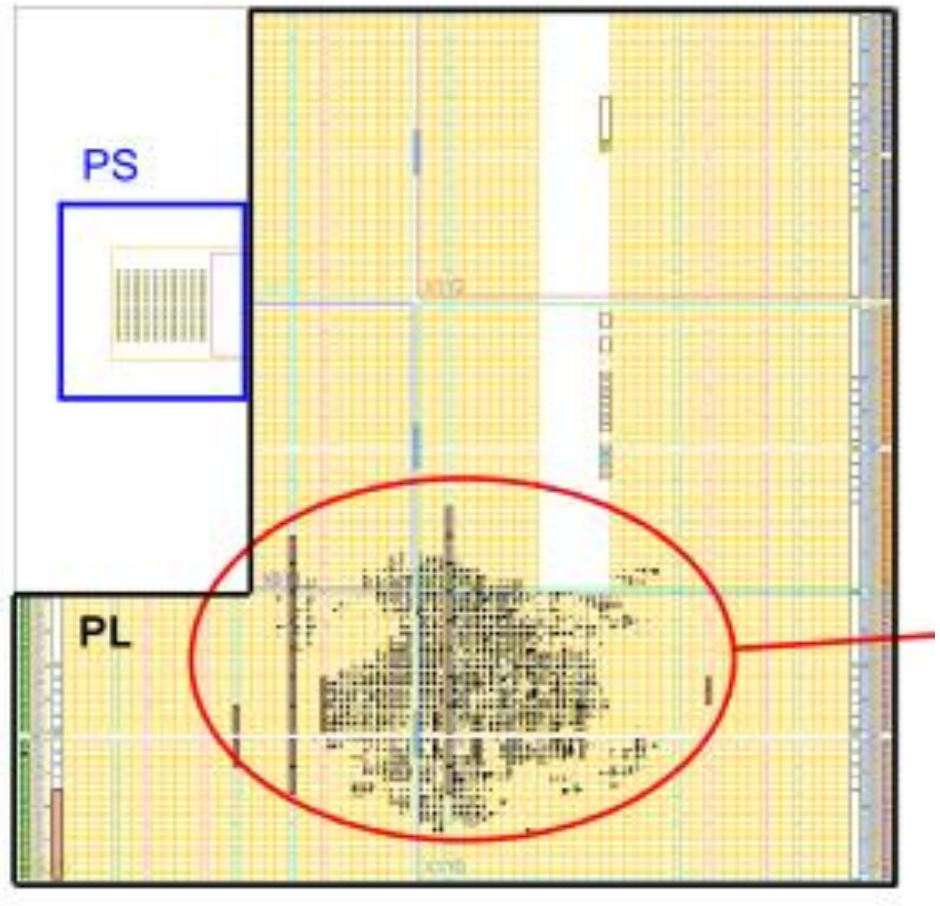


- Am Beispie des Xilinx MicroBlaze Soft Cores
  - RISC Prozessor
- Kleine Konfiguration
- Pipeline mit drei Stufen
- Maximale Taktfrequenz 298 MHz
- Rechenleistung 318 DMIPS
  - Gemessen mit Dhrystone-Benchmark

Prozessor in programmierbarer Logik

Soft Core Processor

TECHNISCHE UNIVERSITÄT DARMSTADT



- Schnelle Konfiguration
- Pipeline mit fünf Stufen
- Maximale Taktfrequenz 329 MHz
- Rechenleistung 440 DMIPS
  - Gemessen mit Dhrystone-Benchmark

Vergleich: Hard Core vs. Soft Cores

TECHNISCHE UNIVERSITÄT DARMSTADT

Name	Taktfrequenz [MHz]	Rechenleistung [DMIPS]
MicroBlaze (klein)	298	318
MicroBlaze (schnell)	329	440
Hard Cortex-A9	1.000	2.500

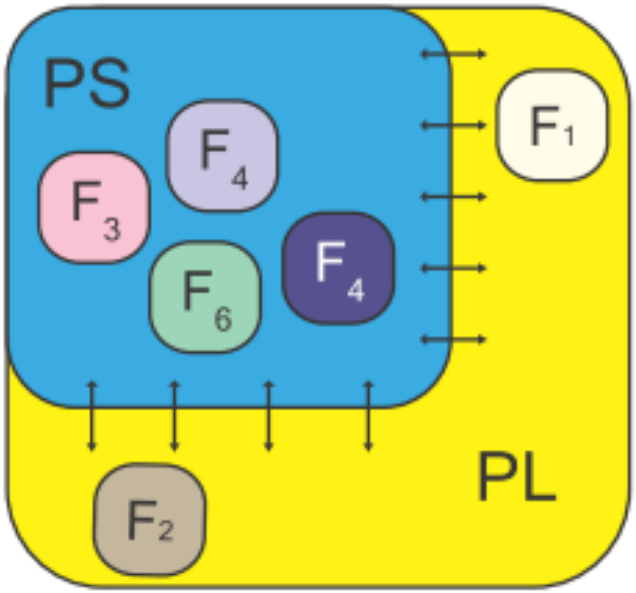
- Soft Cores in rekonfigurierbarer Logik
  - … brauchen deutlich mehr Chip-Fläche als Hard Cores
  - … laufen wesentlich langsamer

- Einsatz von Soft Cores in der Regel nicht mehr effizient, Trend geht zur Kombination von
  - Hard Core CPUs (für effiziente Realisierung von Standardfunktionen)
  - Rekonfigurierbarer Logik (für flexible Realisierung von Spezialfunktionen)

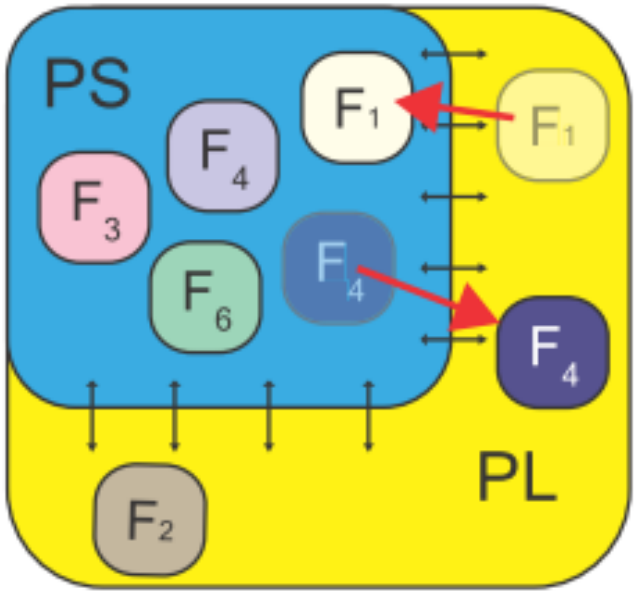
Partitionierung von Algorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Teile Ausführung auf zwischen Ausführung
  - … als Software auf Prozessor (PS)
  - … als spezialisierte Hardware-Einheiten in programmierbarer Logik (PL)



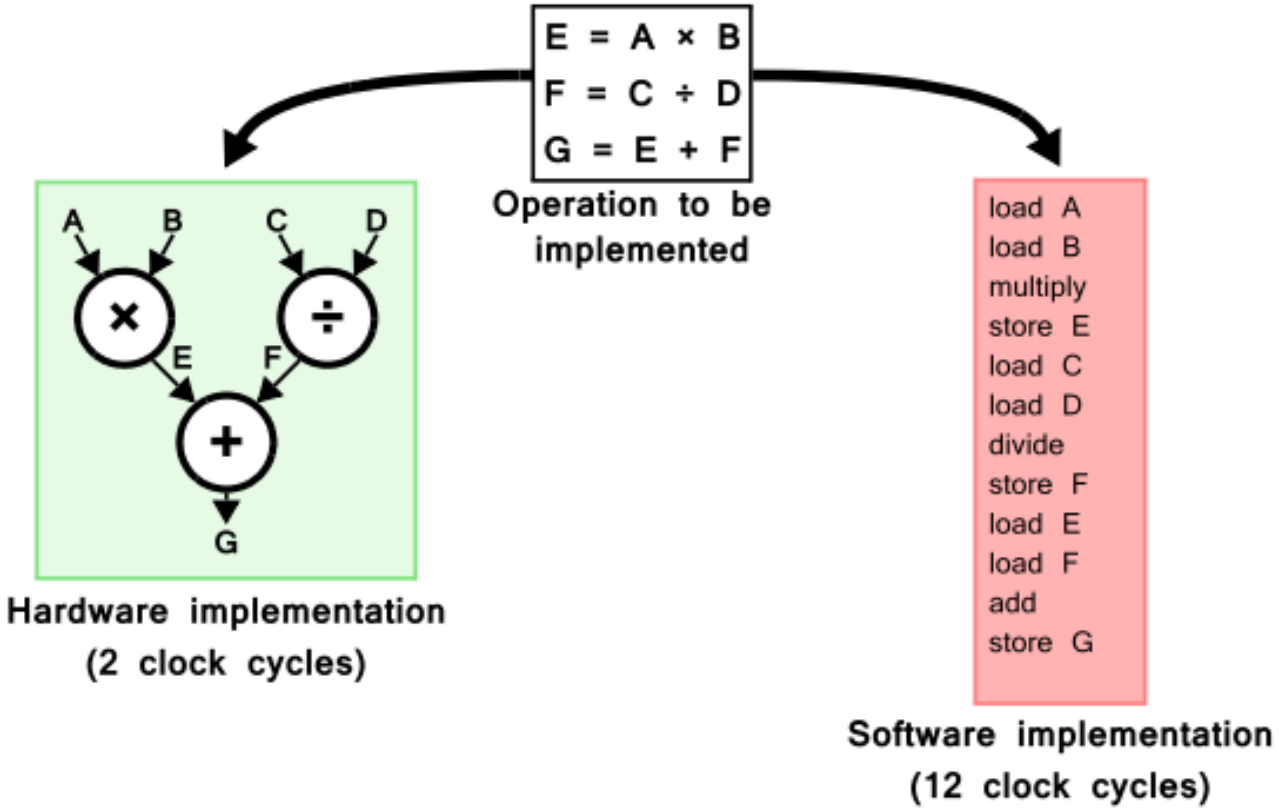
F1 und F2 in HW, Rest in SW



F1 nun in SW, F4 nun in HW

Embedded Systems & Applications

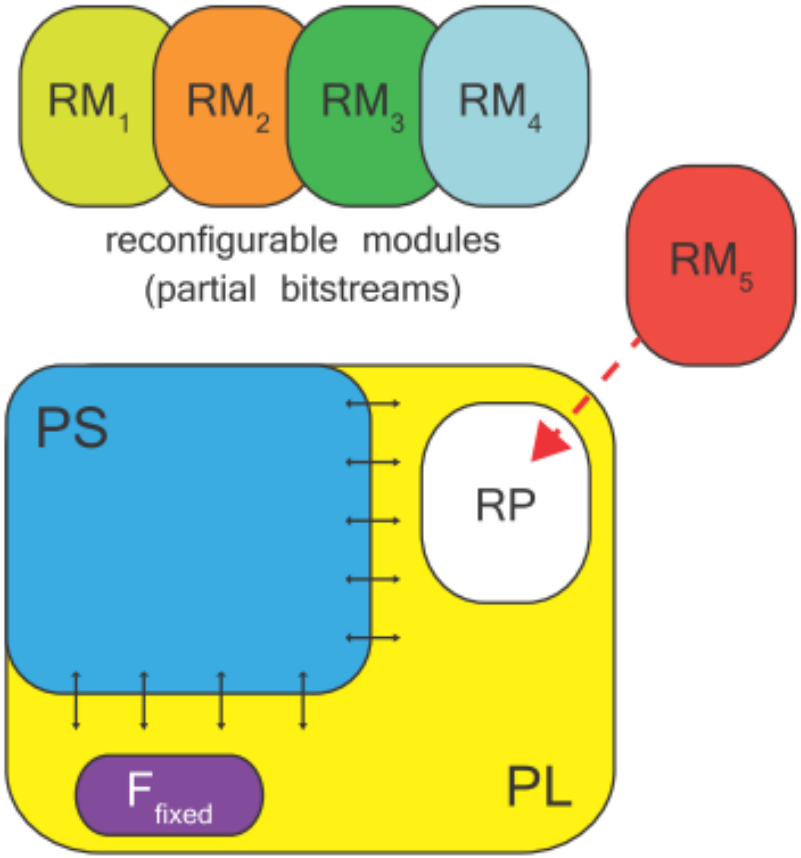
Beispiel: Schnellere Rechnung in HW



Partielle dynamische Rekonfiguration

TECHNISCHE UNIVERSITÄT DARMSTADT

- Dynamisch
  - Tausche Funktionen in der PL zur Laufzeit aus
- Partiell
  - Rekonfiguriere nur die zu ändernden Teile der PL
  - Der Rest läuft unverändert weiter
    - Auch während der Rekonfiguration!
- Erlaubt Änderung von Hardware-Funktionen während des laufenden Betriebs
  - Effiziente Ausnutzung der Chip-Fläche



Embedded Systems & Applications

TECHNISCHE UNIVERSITÄT DARMSTADT

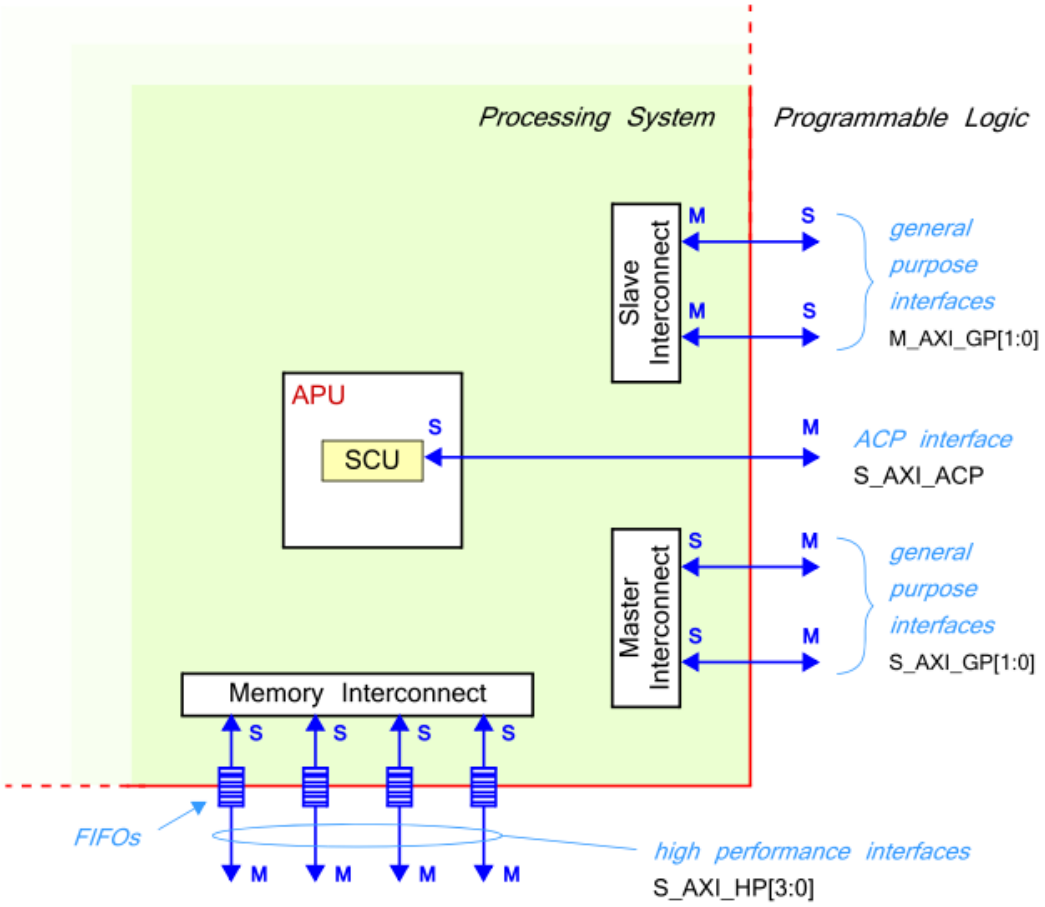
SCHNITTSTELLEN ZWISCHEN PROZESSOR UND FPGA

Schnittstellen zwischen PS und PL

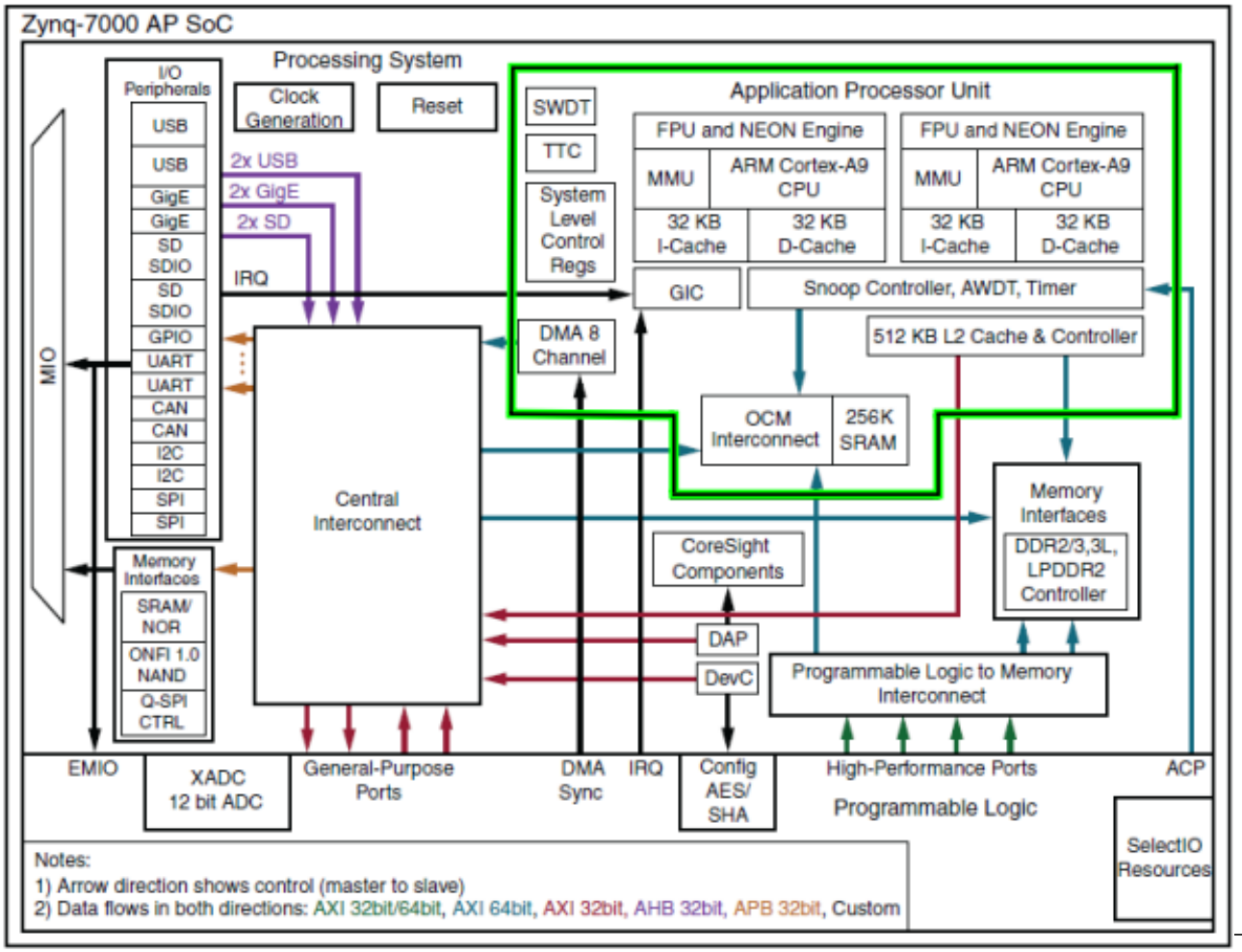
TECHNISCHE UNIVERSITÄT DARMSTADT

- Prozessor und programmierbare Logik müssen Daten austauschen
  - Eingabeparameter für Hardware-Funktionen
  - Ergebnisse von Hardware-Funktionen
  - In Zynq: Hardware-Funktionen haben keine eigene Schnittstelle zum externen Speicher
    - Greifen über PS auf gemeinsamen Hauptspeicher zu
- Kommunikation benötigt Schnittstellen zwischen PL und PS
- Alle Schnittstellen verwenden Varianten des ARM AMBA AXI4 Protokolls
  - Advanced Microcontroller Bus Architecture Advanced eXtensible Interface
  - Grundidee
    - Getrennte Signale für Lese- und Schreibrichtungen
    - Es müssen nicht immer beide Richtungen vorhanden sein

Konkrete Schnittstellen …



… für verschiedene Aufgaben

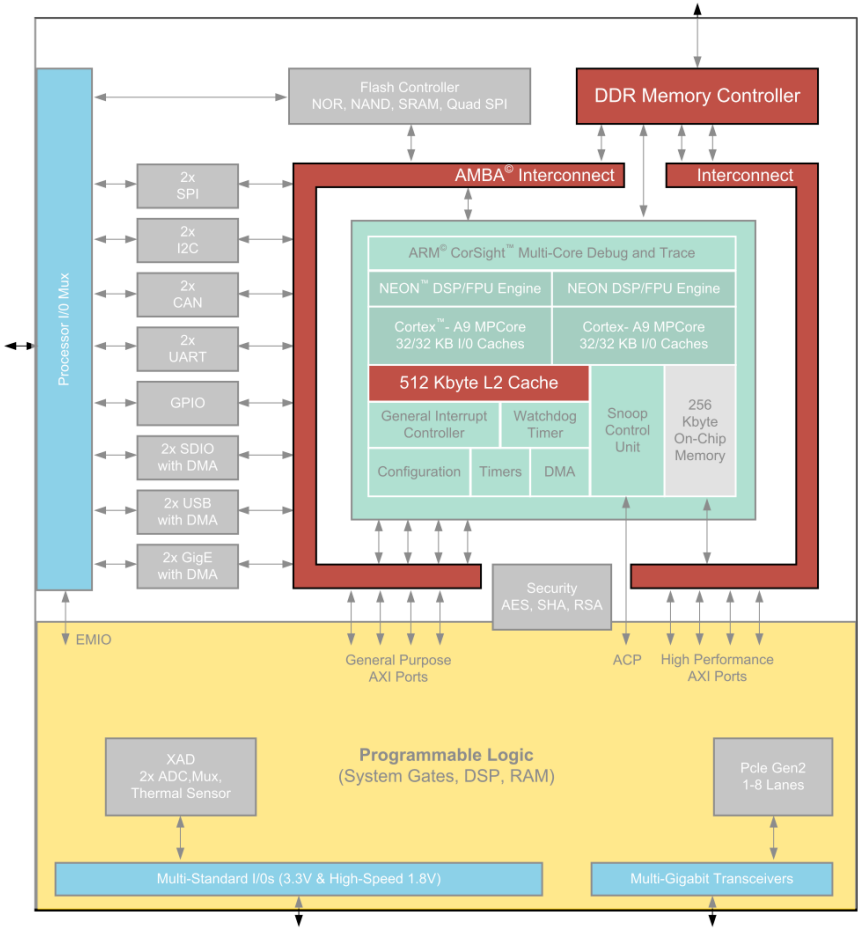


Gemeinsam genutzte Ressourcen

- Interconnects
- L2 Cache
- Speicher Controller

- Zugriffe von
  - Prozessoren
  - PL
  - Master-fähigen Peripheriegeräten

  teilen sich Übertragungsdurchsatz



[Beispiele hier aus Xilinx UG1145]

1. Beispiel: CPU und HP Ports

TECHNISCHE UNIVERSITÄT DARMSTADT

- Software (z.B. Betriebssystem) läuft auf CPU
- PL macht Videoverarbeitung von zwei Videodatenströmen (1080p60)
  - Je Strom benötigt: 376 MB/s lesen und schreiben
  - Strom 1: Liest auf HP0, schreibt auf HP1; Strom 2 auf HP3 und HP4

- Abschätzung des benötigten Durchsatzes
  - Bandbreite CPU (hier vernachlässigbar)
  - 4x 376 MB/s = 1.504 MB/s für Videodaten

- Theoretischer (!) maximaler Durchsatz zum DDR3-SDRAM Speicher
  - Beispiel: zedboard
  - 533 MHz
  - Double Data Rate (Transfer bei @posedge und @negedge)
  - 2x 16b breite DDR3-SDRAM Chips, damit 4 B pro Transfer
  - 533 MHz * 2 * 4 B = 4.264 MB/s → sollte ausreichen für Anwendung

1. Beispiel: Messungen

TECHNISCHE UNIVERSITÄT DARMSTADT

PS Performance Counters

	CPU0	CPU1
CPU Utilization(%)	100.00	0.00
CPU Instructions Per Cycle	0.38	0.00
L1 Data Cache Miss Rate(%)	8.28	0.00
L1 Data Cache Access	3466.94M	0.00
CPU Write Stall Cycles Per Instructi...	0.00	0.00
CPU Read Stall Cycles Per Instructi...	0.60	0.00

PL Performance Counters

	HP0	HP1	HP2	HP3
Write Transactions	0.00	233.93M	0.00	233.93M
Average Write Latency	0.00	19.02	0.00	19.02
Write Throughput (MB/sec)	0.00	377.42	0.00	377.43
Read Transactions	233.93M	0.00	233.93M	0.00
Average Read Latency	42.96	0.00	40.17	0.00
Read Throughput (MB/sec)	377.42	0.00	377.42	0.00

1. Beispiel: Interpretation

TECHNISCHE UNIVERSITÄT DARMSTADT

PS Performance Counters

	CPU0	CPU1
CPU Utilization(%)	100.00	0.00
CPU Instructions Per Cycle	0.38	0.00
L1 Data Cache Miss Rate(%)	8.28	0.00
L1 Data Cache Access	3466.94M	0.00
CPU Write Stall Cycles Per Instructi...	0.00	0.00
CPU Read Stall Cycles Per Instructi...	0.60	0.00

Kaum Verzögerungen für CPU-Zugriffe

PL Performance Counters

	HP0	HP1	HP2	HP3
Write Transactions	0.00	233.93M	0.00	233.93M
Average Write Latency	0.00	19.02	0.00	19.02
Write Throughput (MB/sec)	0.00	377.42	0.00	377.43
Read Transactions	233.93M	0.00	233.93M	0.00
Average Read Latency	42.96	0.00	40.17	0.00
Read Throughput (MB/sec)	377.42	0.00	377.42	0.00

Alle PL-Zugriffe haben benötigten Durchsatz

2. Beispiel: Mehr Durchsatz

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gleiches Szenario: CPU und vier HP Ports

- Nun aber höhere deutlich mehr Datentransfers aus der PL
  - Angenommen: Alle vier Ports lesen und schreiben gleichzeitig
  - Mit 512 MB/s je Port je Richtung
  - Ingesamt benötigt 4.096 MB/s

- Theoretisches (!) Maximum des Speicherdurchsatzes: 4.264 MB/s
  - … wird eng!

2. Beispiel: Interpretation

TECHNISCHE UNIVERSITÄT DARMSTADT

PS Performance Counters

	CPU0	CPU1
CPU Utilization(%)	100.00	0.00
CPU Instructions Per Cycle	0.25	0.00
L1 Data Cache Miss Rate(%)	8.30	0.00
L1 Data Cache Access	3459.83M	0.00
CPU Write Stall Cycles Per Instructi...	0.00	0.00
CPU Read Stall Cycles Per Instructi...	2.44	0.00

PL Performance Counters

	HP0	HP1	HP2	HP3
Write Transactions	262.61M	262.61M	262.51M	262.51M
Average Write Latency	19.02	19.02	19.02	19.02
Write Throughput (MB/sec)	424.22	424.22	424.06	424.06
Read Transactions	229.54M	229.54M	229.53M	229.53M
Average Read Latency	217.97	217.91	217.96	217.91
Read Throughput (MB/sec)	370.81	370.81	370.79	370.79

2. Beispiel: Messungen

TECHNISCHE UNIVERSITÄT DARMSTADT

PS Performance Counters

	CPU0	CPU1
CPU Utilization(%)	100.00	0.00
CPU Instructions Per Cycle	0.25	0.00
L1 Data Cache Miss Rate(%)	8.30	0.00
L1 Data Cache Access	3459.83M	0.00
CPU Write Stall Cycles Per Instructi...	0.00	0.00
CPU Read Stall Cycles Per Instructi...	2.44	0.00

CPU nun verlangsamt (war 0.60)

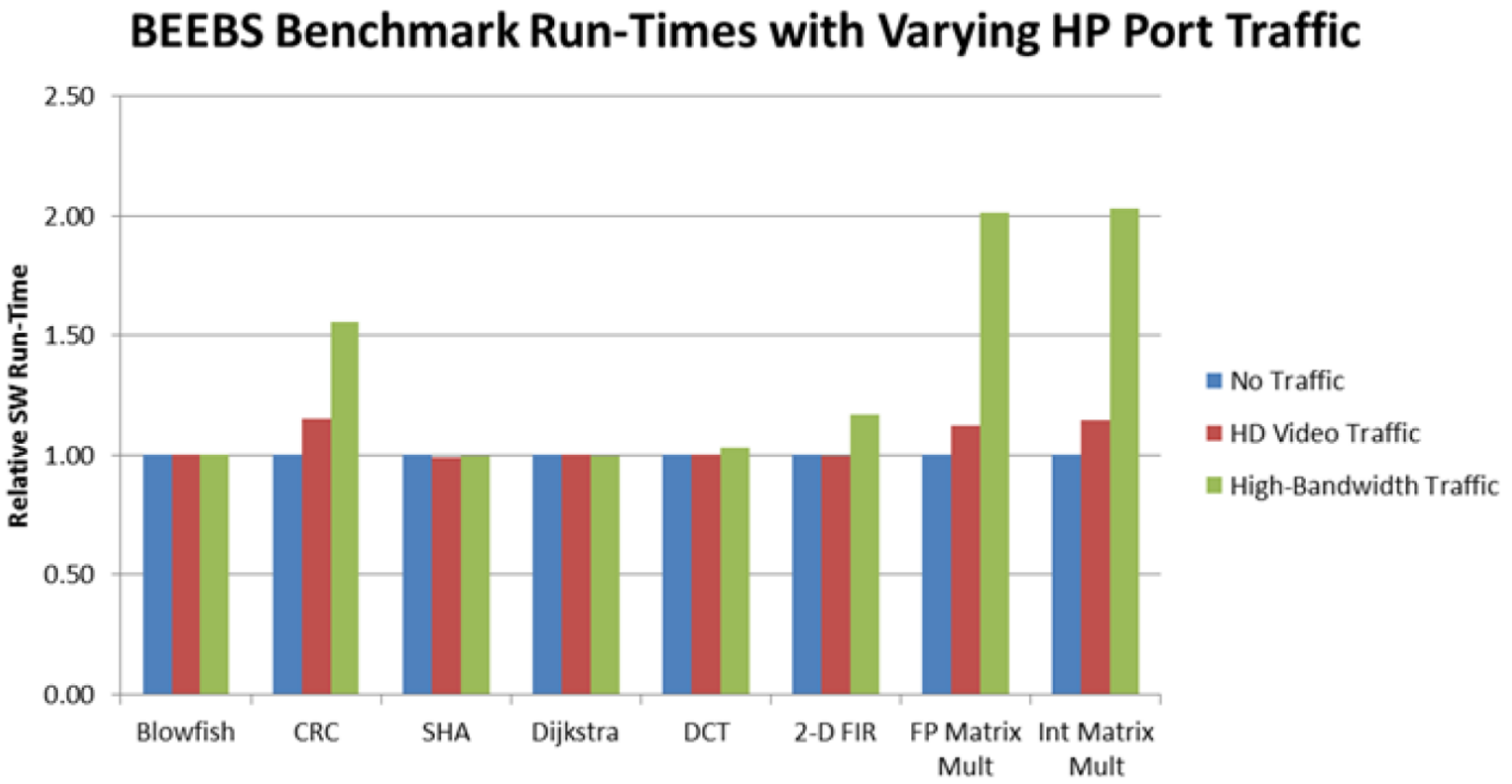
PL Performance Counters

	HP0	HP1	HP2	HP3
Write Transactions	262.61M	262.61M	262.51M	262.51M
Average Write Latency	19.02	19.02	19.02	19.02
Write Throughput (MB/sec)	424.22	424.22	424.06	424.06
Read Transactions	229.54M	229.54M	229.53M	229.53M
Average Read Latency	217.97	217.91	217.96	217.91
Read Throughput (MB/sec)	370.81	370.81	370.79	370.79

Langsamer, war 40-42

Keiner der Ports erreicht benötigten Durchsatz (gesamt 3179.76 MB/s, 74.5% DDR3 SDRAM)

Einfluß von PL auf Software-Ausführung



- Deutliche Verlangsamung von Software auf CPU bei vielen Datentransfers von PL

Diskussion HP vs ACP

TECHNISCHE UNIVERSITÄT DARMSTADT

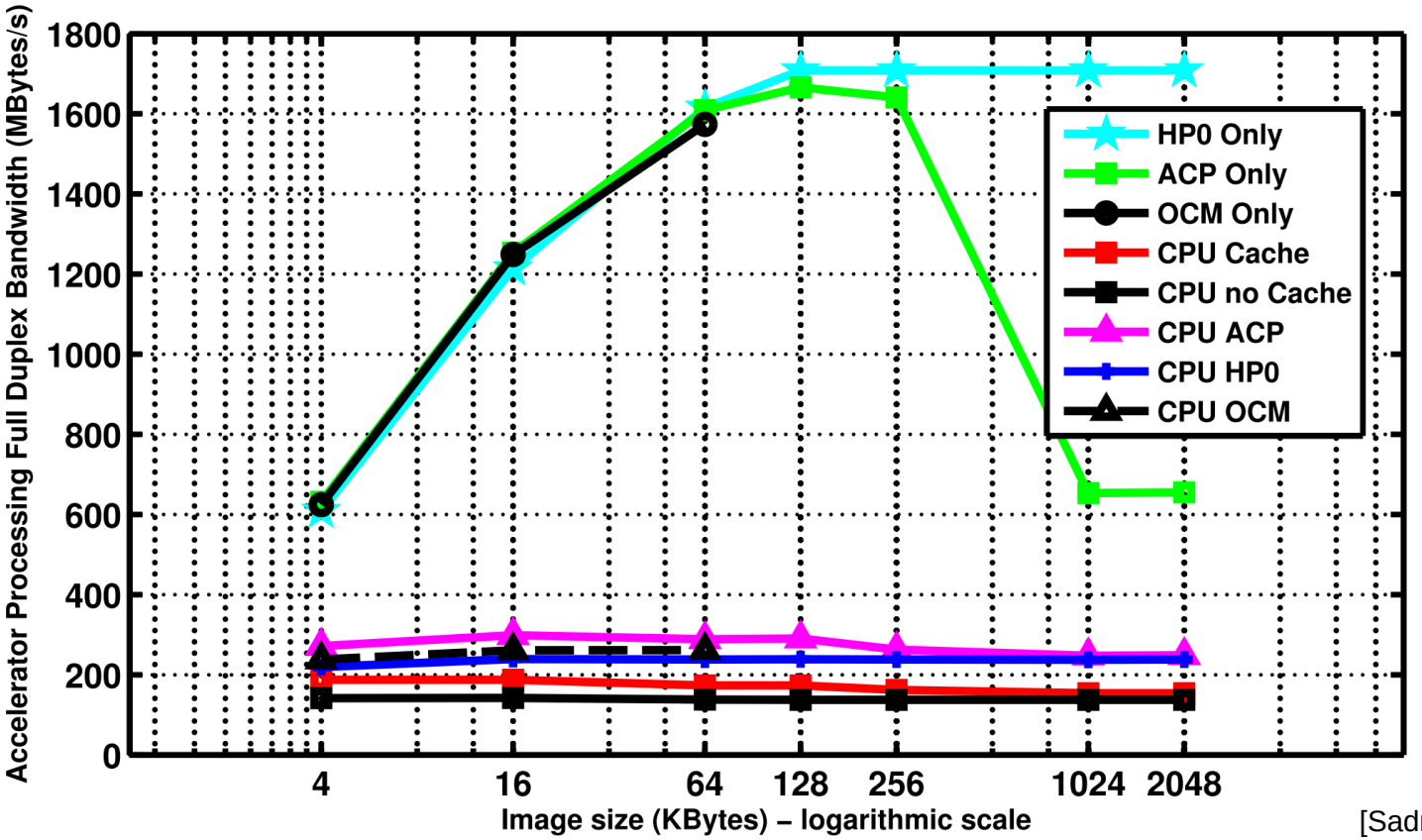
- HP und ACP greifen beide auf Speicher zu
- ACP greift aber via SCU auf L2 Cache zu

- ACP
  - Erlaubt cache-kohärente Ausführung zusammen mit CPU (*shared memory model*)
  - Liefert Daten schnell, solange sie im Cache liegen
  - Teilt sich aber L2 Cache mit CPU
  - Nach erfolglosem Zugriff auf Cache (miss) Zugriff auf externen DDR3-SDRAM
    - Cache-Verarbeitung braucht aber zusätzliche Latenz

- HP
  - Ist cache-inkohärent zu CPU
  - Greift immer auf DDR3-SDRAM zu (PL-interne Caches wären aber möglich)
  - Belastet nicht CPU L2 Cache (z.B. keine Verdrängung von Cache-Lines)

Vergleich HP vs ACP

Beispiel: Bildverarbeitung von Eingabe- nach Ausgabebild



[Sadri FPGAworld 2013]

- L2$ im Zynq: 512 KB
- Bei Bildgröße von 256 KB reicht L2$ nicht mehr aus, ACP Durchsatz bricht ein

Protokollfamilie ARM AMBA AXI4

Varianten

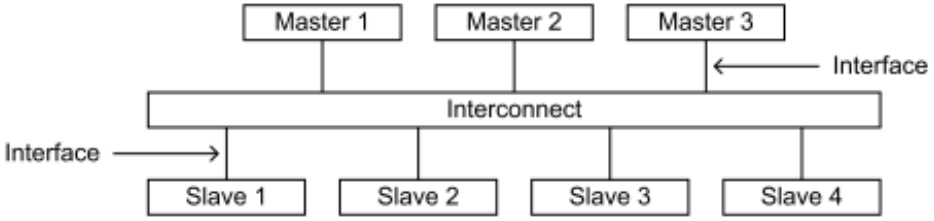
TECHNISCHE UNIVERSITÄT DARMSTADT

- **AXI4**: Mächtigste Implementierung, benötigt am meisten Chip-Fläche
  - Unterstützt memory-mapped I/O (Adressen und Daten)
  - Erlaubt effiziente Übertragung von ganzen Datengruppen (*burst transfer*)
- **AXI4-Lite**: Einfacher, braucht aber weniger Chip-Fläche
  - Unterstützt memory-mapped I/O (Adressen und Daten)
  - Aber nur noch Übertragung von Einzeldaten, keine Burst-Transfers mehr
- **AXI4-Stream**: Spezialisierte Realisierung
  - Überträgt nur reine Datenströme (keine Adressen mehr, damit ungeeignet für memory-mapped I/O)
  - Unbegrenzt lange Bursts
  - Unidirektionale Datenübertragung von Master zu Slave
    - Bidirektionale Übertragung benötigt zwei separate AXI4-Streams

AXI4 Grundkonzepte

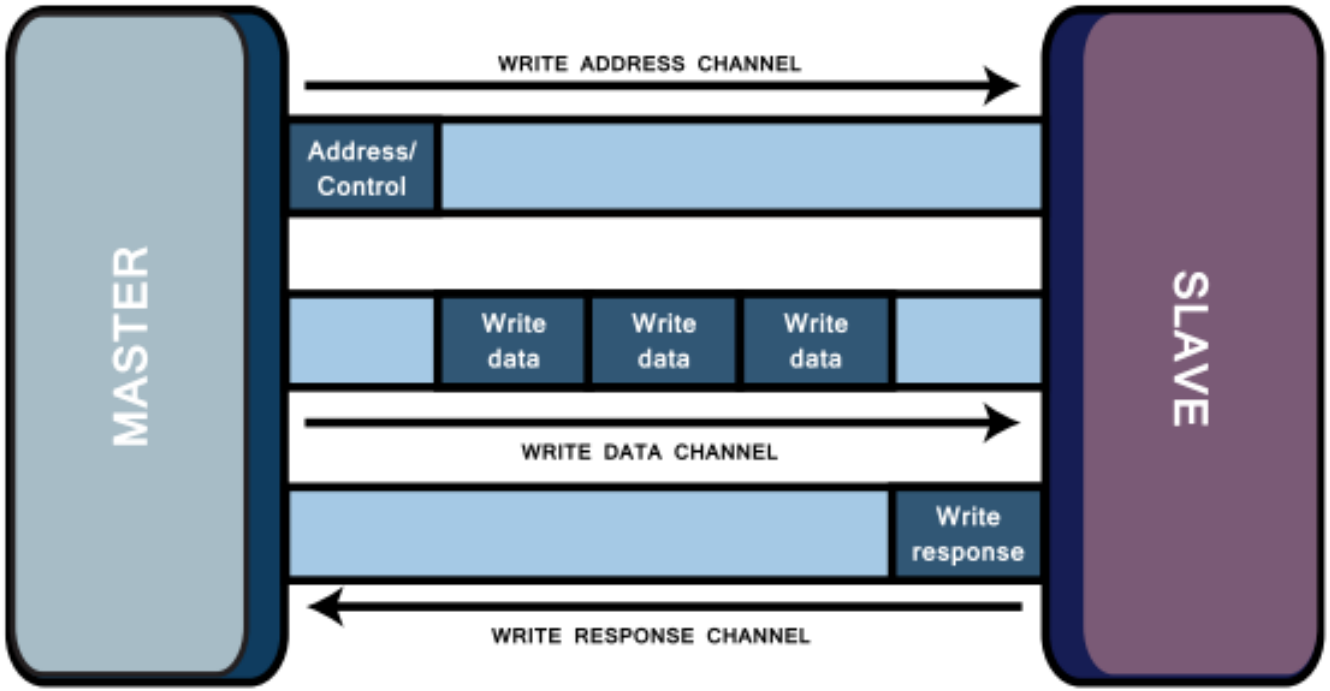
TECHNISCHE UNIVERSITÄT DARMSTADT

- **Getrennte Übertragungskanäle** für Adressen/Kommandos und Daten
  - AXI4-Stream: Verwendet einen einzelnen Kanal für Daten
- Master **löst** Übertragung aus, Slave **reagiert** auf Übertragung
  - Liefert Daten an Master bei Lesezugriff
  - Nimmt Daten vom Master entgegen bei Schreibzugriff
  - Liefert Status an Master (über Lesekanal oder extra Rückkanal für Antworten beim Schreiben)
- Mehrere AXI Kommunikationspartner kommunizieren über ein **Verbindungsnetz**
  - *Interconnect*
  - Abstrahiert, **nicht** zwangsläufig ein Bus



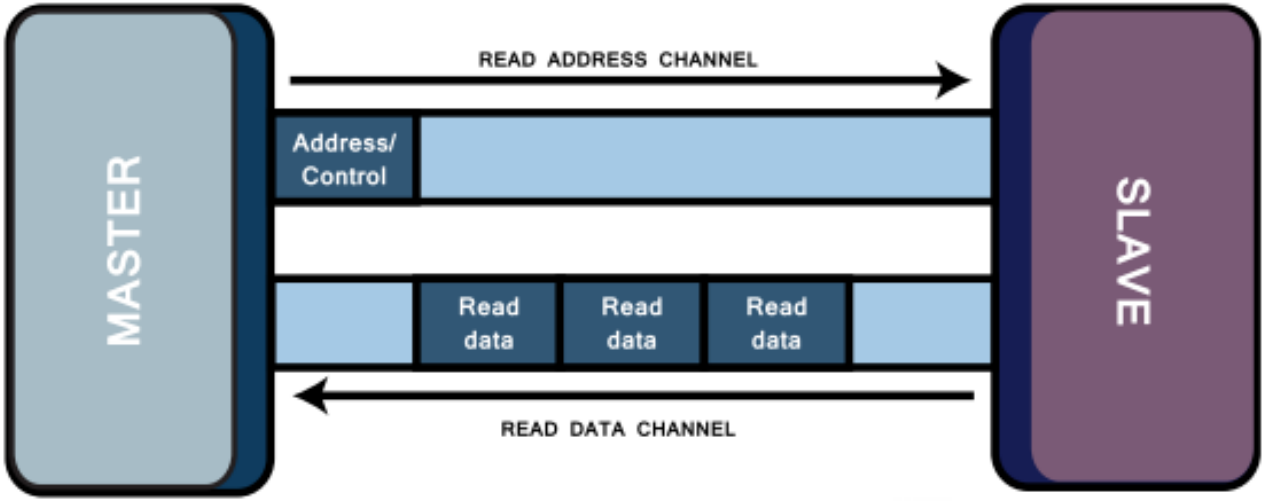
Embedded Systems & Applications

Master schreibt Daten zum Slave



- Burst-Übertragung: Master setzt nur Start-Adresse
  - Slave muss Folgeadressen selber bestimmen (z.B. Hochzählen)
  - Maximale Burstlänge sind 256 Datensätze (genannt *beats*, im Beispiel hier 3)

Master liest Daten vom Slave



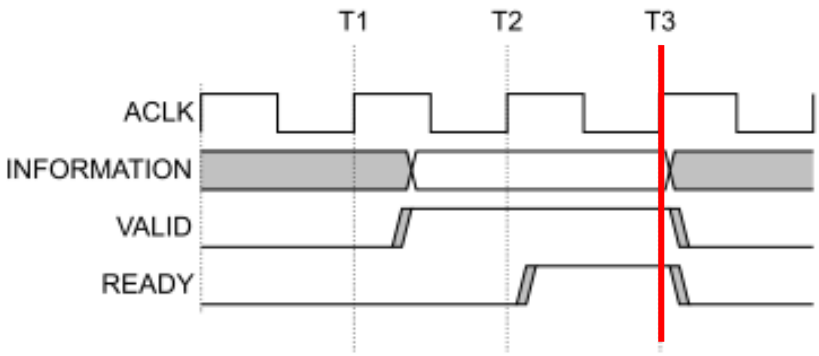
- Wieder Burst-Übertragung von 3 Beats
  - Ein einzelner Beat kann 1 bis 128 Bytes umfassen

Grundlagen der Signalisierung

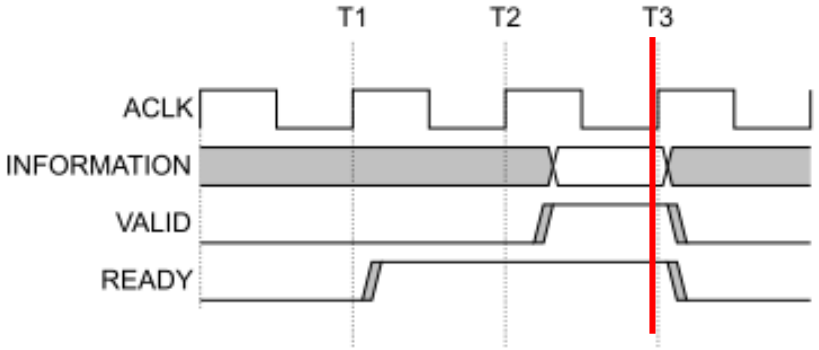
TECHNISCHE UNIVERSITÄT DARMSTADT

- Handshake zwischen Quelle und Ziel von Daten, jeweils ausgewertet @**posedge**
  - **Quelle** setzt VALID, wenn gültige Daten anliegen
  - **Ziel** setzt READY, wenn Daten übernommen werden können

- Idee
  - Wenn VALID **und** READY gleichzeitig aktiv sind, wurden Daten übernommen



VALID **vor** READY gesetzt



VALID **nach** READY gesetzt

Embedded Systems & Applications

Handshake Signale

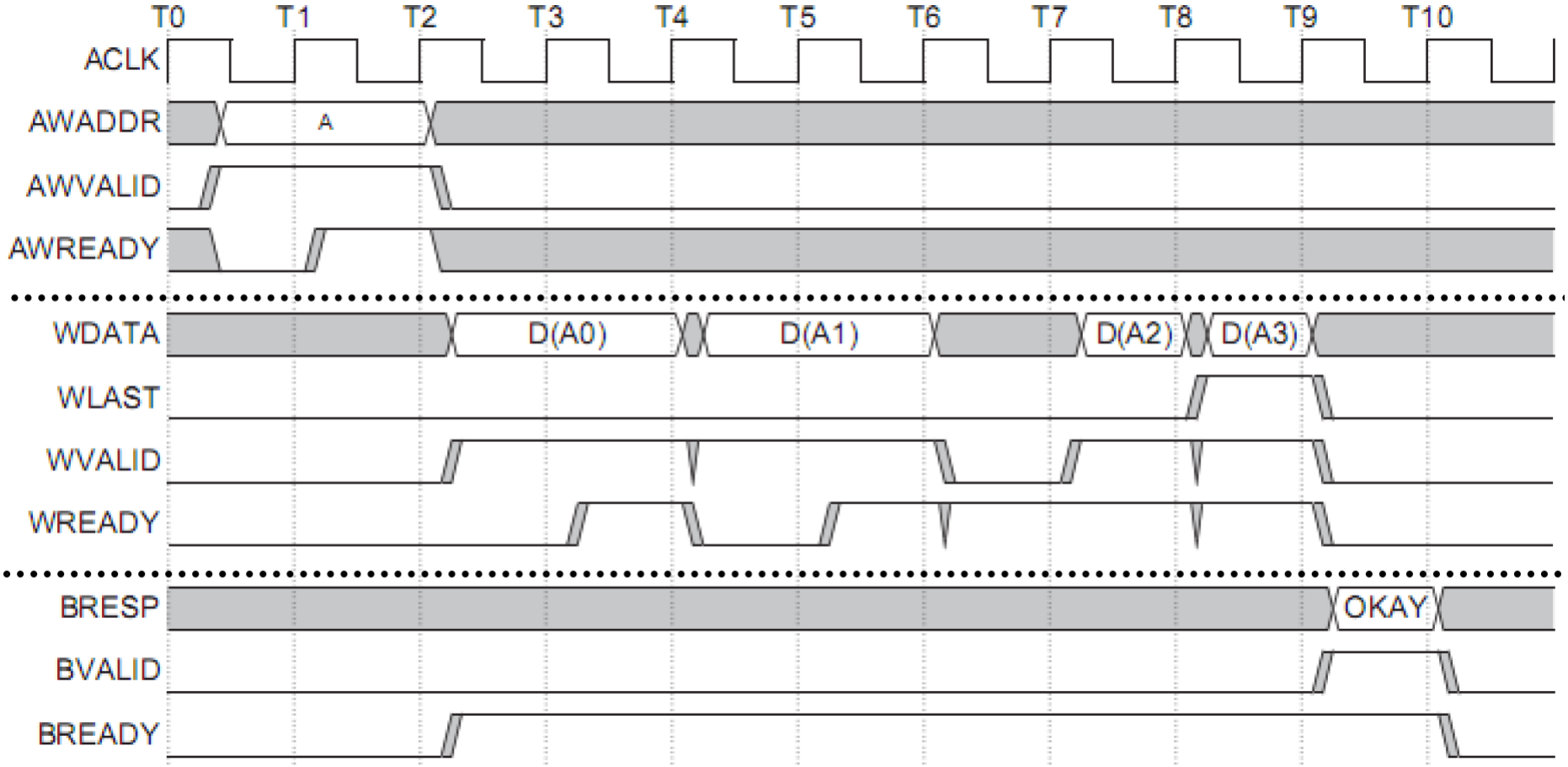
Transaction channel	Handshake pair
Write address channel	**AWVALID, AWREADY**
Write data channel	**WVALID, WREADY**
Write response channel	**BVALID, BREADY**
Read address channel	**ARVALID, ARREADY**
Read data channel	**RVALID, RREADY**

Beispiel: Master schreibt zum Slave

Vereinfacht, nicht alle Signale gezeigt

TECHNISCHE UNIVERSITÄT DARMSTADT

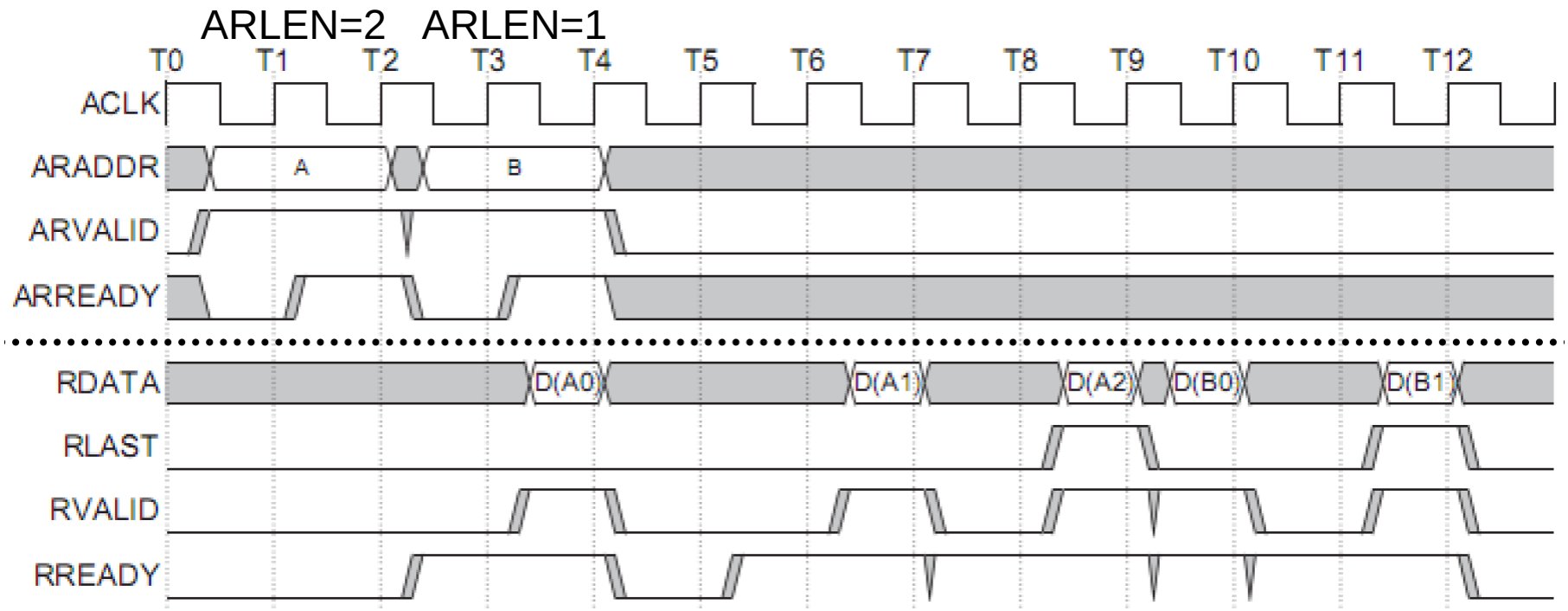
AWLEN=3 → 4 Beats



Embedded Systems & Applications

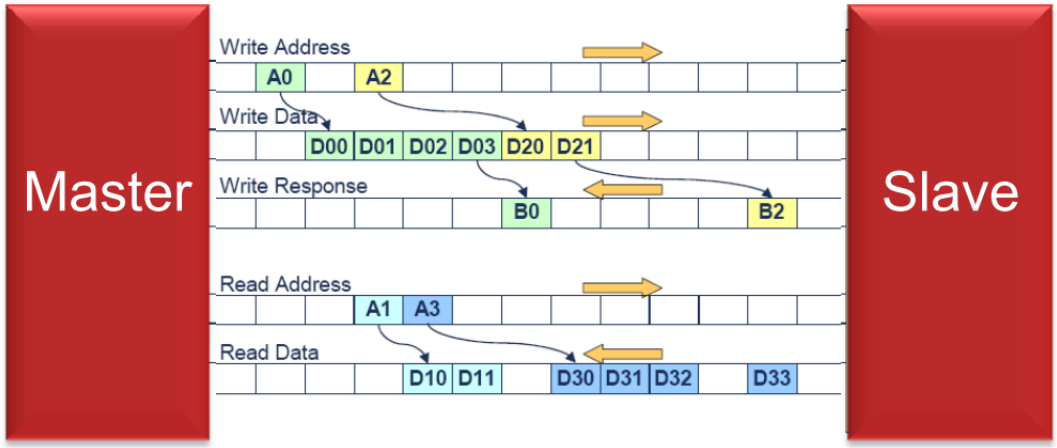
Beispiel: Master liest vom Slave

Vereinfacht, nicht alle Signale gezeigt



- Hier: Ausnutzen von Pipelining
  - Übertrage ARLEN+1=3 Datenworte an Adresse A,A+1,A+2
  - Übertrage nächste ARLEN+1=2 Datenworte an Adresse B, B+1
  - Slave beendet dann ersten Burst durch Setzen von RLAST und startet nächsten Burst

Übersicht über wesentliche AXI4 Signale



Write Address Channel

- AWID[m:0]
- **AWVALID**
- **AWREADY**
- **AWADDR[31:0]**
- AWLEN[7:0]
- AWSIZE[2:0]
- **AWPROT[2:0]**
- AWBURST[1:0]
- AWLOCK
- AWCACHE[3:0]
- AWREGION[3:0]
- AWQOS[3:0]

Write Data Channel

- **WVALID**
- **WREADY**
- **WDATA[n-1:0]**
- **WSTRB[n/8-1:0]**
- WLAST

Write Response Channel

- BID[m:0]
- **BVALID**
- **BREADY**
- **BRESP[1:0]**

Read Address Channel

- ARID[m:0]
- **ARVALID**
- **ARREADY**
- **ARADDR[31:0]**
- ARLEN[7:0]
- ARSIZE[2:0]
- ARBURST[1:0]
- **ARPROT[2:0]**
- ARLOCK
- ARCACHE[3:0]
- ARREGION[3:0]
- ARQOS[3:0]

Read Data Channel

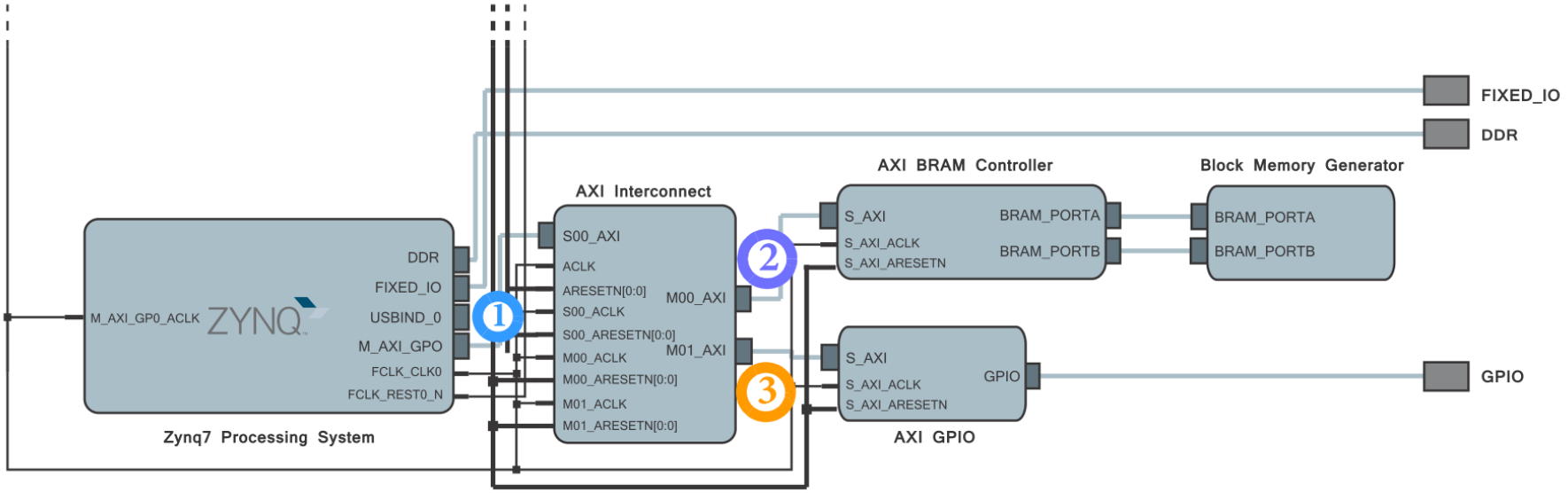
- RID[m:0]
- **RVALID**
- **RREADY**
- **RDATA[n-1:0]**
- **RRESP[1:0]**
- RLAST

ACLK
ARESETn

Fettdruck: AXI4-Lite

Beispielsystem

Prozessor soll auf BlockRAM zugreifen und LED steuern



1) PS ist Master, verwendet General Purpose-Schnittstelle
   Interconnect ist Slave
2) Interconnect agiert als Master gegenüber BRAM Schnittstelle
3) Interconnect agiert als Master gegenüber LED-Pin
   - GPIO: General Purpose I/O, allgemeinverwendbare Ein-/Ausgabepins

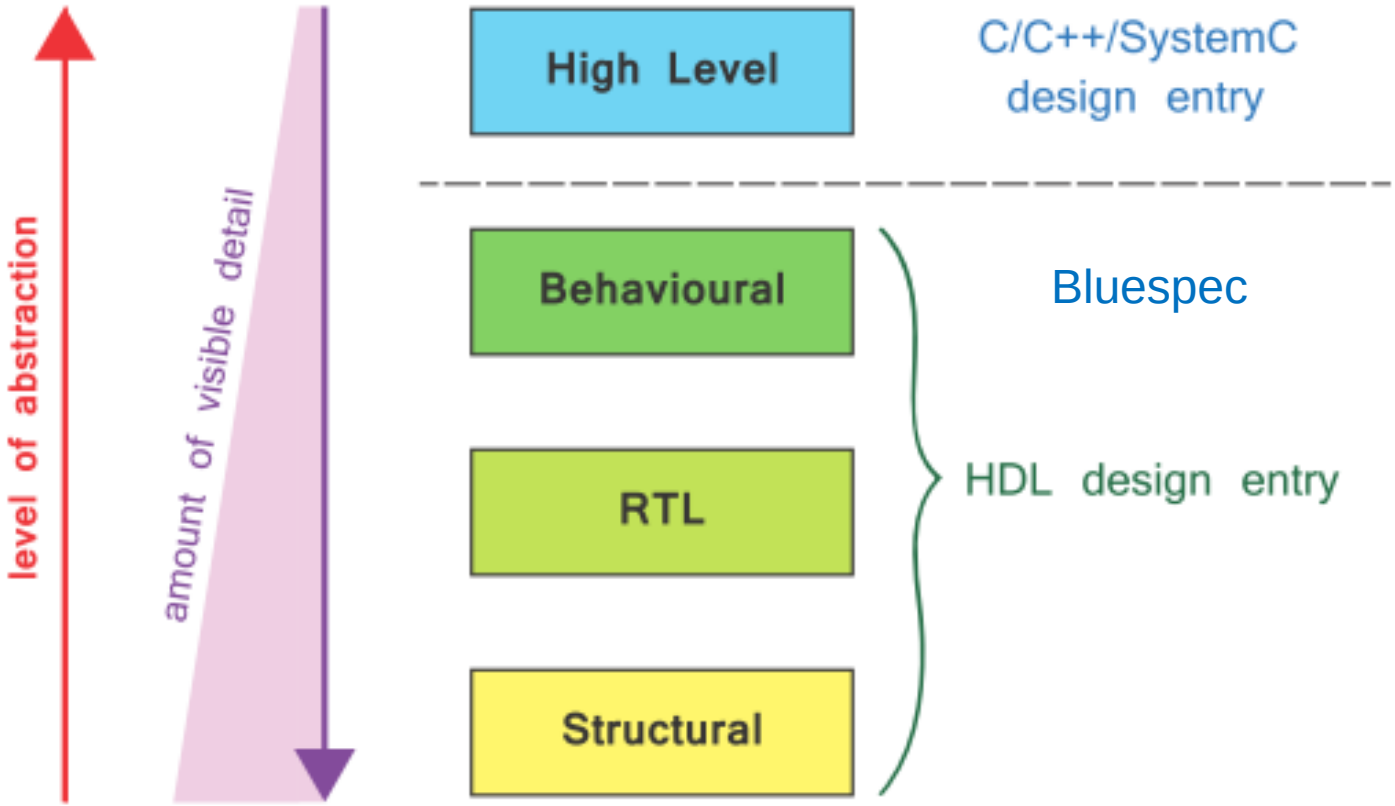
EXKURS: HIGH-LEVEL SYNTHESE

Steigerung der Entwurfsproduktivität

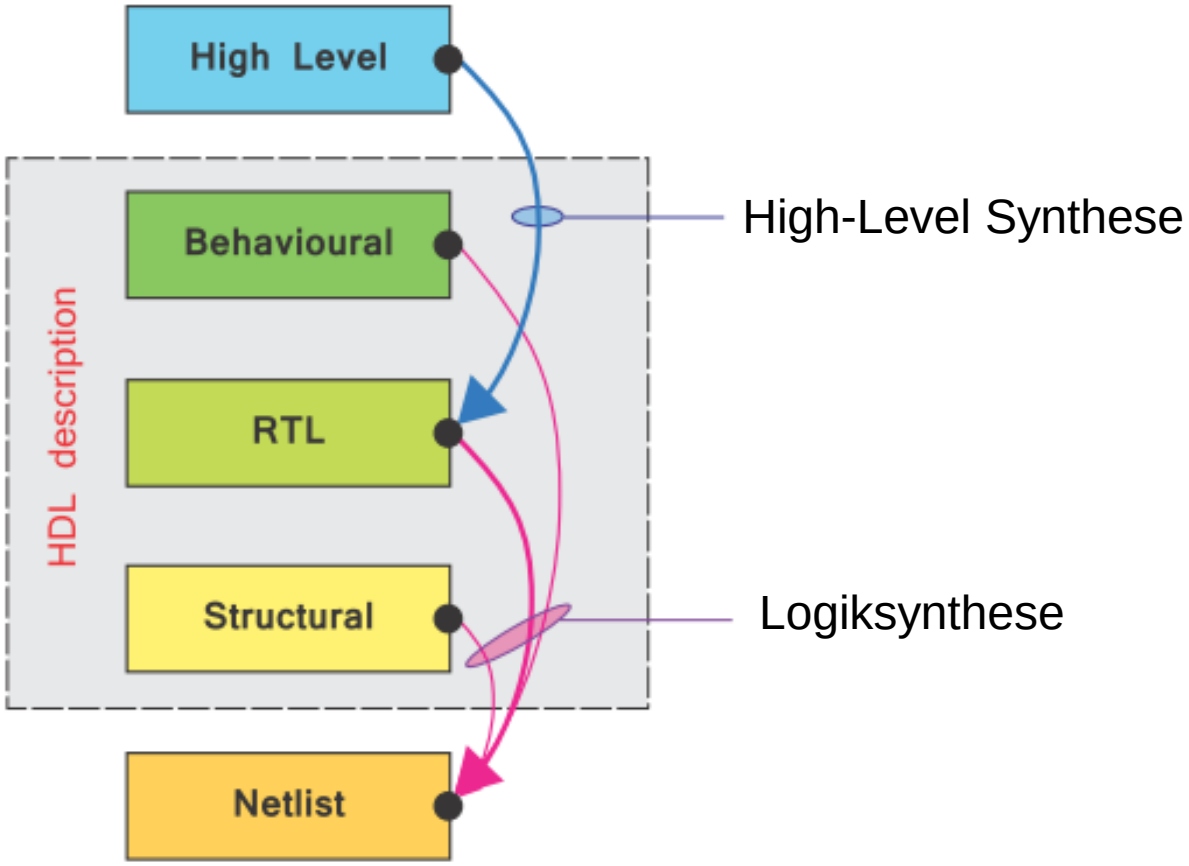
TECHNISCHE UNIVERSITÄT DARMSTADT

- rSoC ist extrem flexibel
- Aber Hardware-Entwurf immer noch deutlich aufwendiger als Software-Programmierung
  - … mit Bluespec schon produktiver als mit Verilog
  - … aber immer noch viel aufwendiger als in C oder Java
- Idee
  - Baue Übersetzer von Software-Hochsprache zu Hardware
  - Hochsprachensynthese
  - *High-Level Synthesis (HLS)*

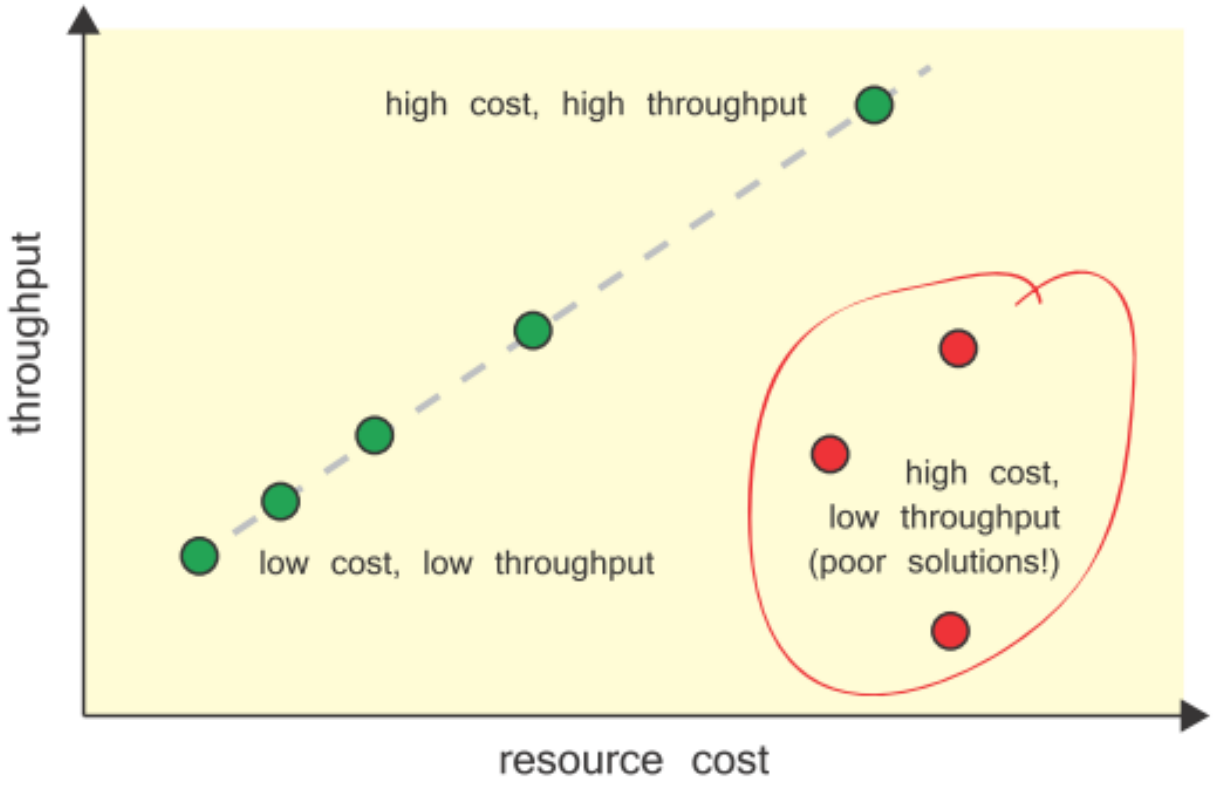
Abstraktionsebenen im HW-Entwurf



Übersetzungsschritte

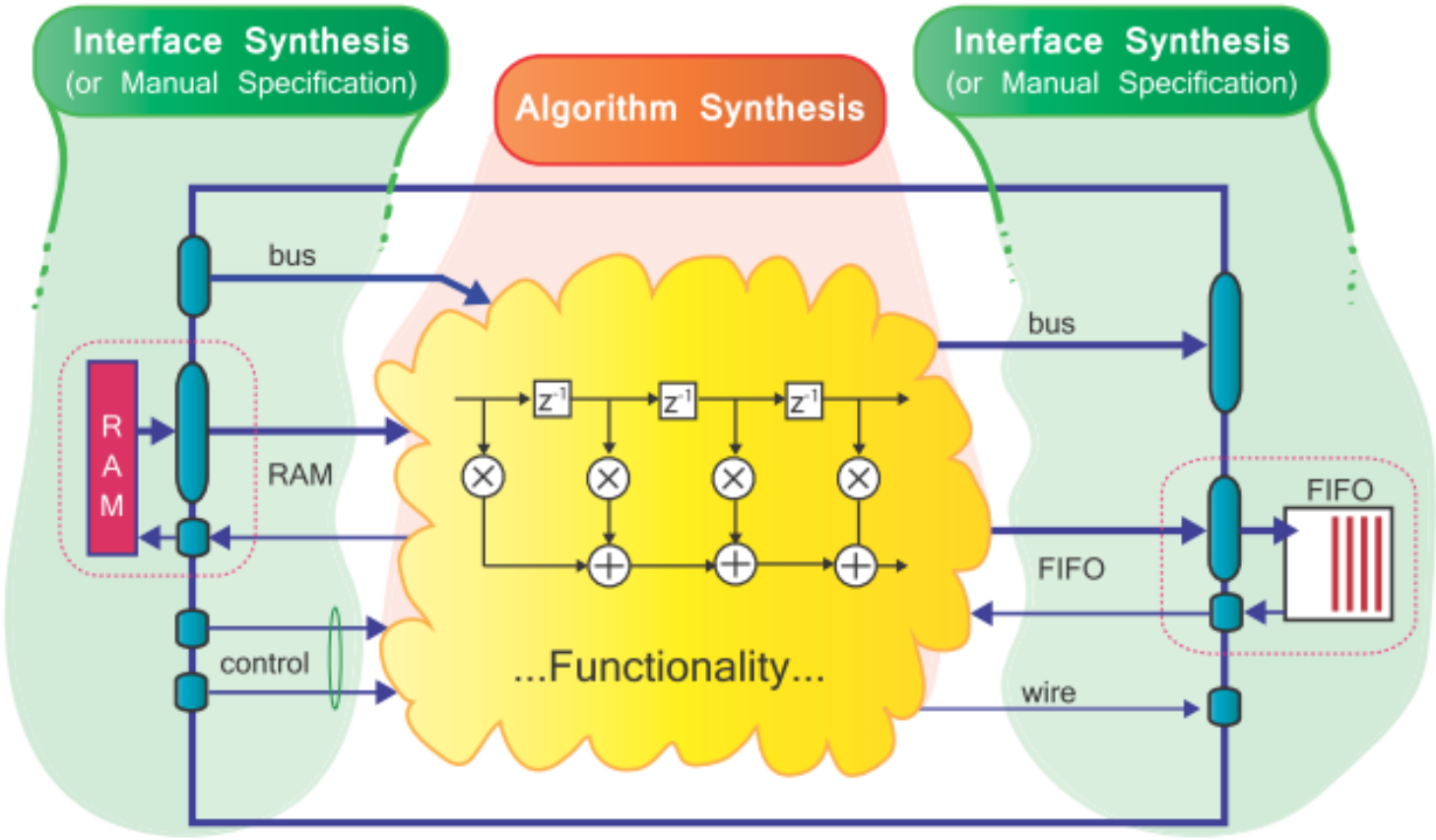


Automatische Entwurfsraumexploration



- HLS-Werkzeug probiert verschiedene Implementierungsalternativen aus
- Bietet eine an, die die Design Constraints (z.B. Fläche, Durchsatz) erfüllt

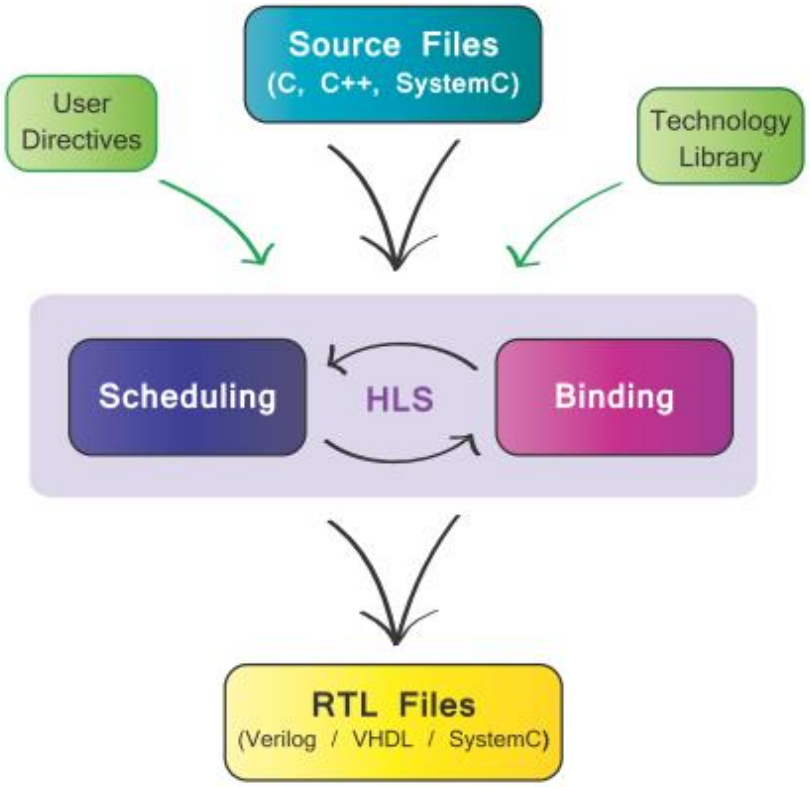
Teilaufgaben des HLS-Vorgehens



Teilaufgabe der Algorithmensynthese

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ablaufplanung (scheduling)
  - Zuordnung von Operationen an Zeitschritte
- Bindung (binding)
  - Zuordnung von ablaufgeplanten Operationen an echte Hardware-Ressourcen



Beispiel

- Bestimme Durchschnitt *d* aus 10 Zahlen *n1 … n10* als
  - *d = (n1 + n2 + n3 + n4 + n5 +n6 + n7 + n8 + n9 + n10)* * 0.1

- Einige Implementierungsvarianten
  - Verwende so wenig Ressourcen wie möglich und benutze diese wieder
    - *Resource Sharing*
  - Verwende mehr Hardware-Ressourcen und kaskadiere mehrere Operationen in einem Takt
    - *Operator Chaining* (hier kombiniert mit Resource Sharing)
    - Bedingung: Summe der Propagation Delays der Resourcen ist kleiner als Taktperiode
  - Verwende spezialisierte Hardware-Ressourcen (in der Regel schneller als allgemeine)
    - Aber davon deutlich weniger auf Chip vorhanden als allgemeine Logikblöcke

Vergleich der drei Ansätze