Karlsruher Institut für Technologie

Institut für Prozessdatenverarbeitung und Elektronik

Institut für Technik der Informationsverarbeitung

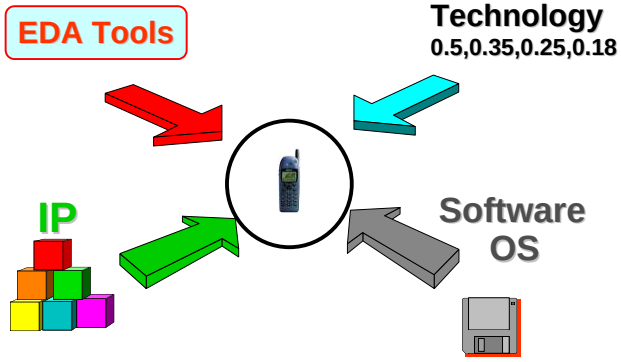
2. Zielarchitekturen

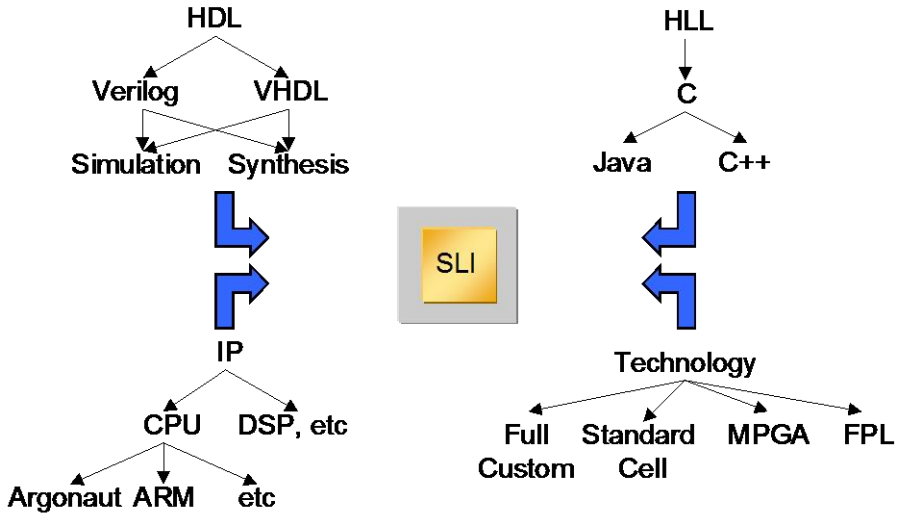
Dr.-Ing. Oliver Sander
Dipl.-Inform. Leonard Masing

Hardware/Software Co-Design

2.8.2 Komplexe SoCs: System-Level Integration

- SoC-Entwurf: Neue IP-basierte Methodik (Intellectual Property)
  - Erhöhung der Abstraktionsebene
  - Prozess-basierte nebenläufige Modelle
    - SystemC, UML-RT, CC++, SDL, JAVA
    - MATLAB, STATEMATE, COSSAP, SPW
- Gekoppelte Simulation und Integration heterogener Modelle
  - HW-SW, digital-analog, elektronisch-mechanisch (Hardware-in-the-Loop)
  - Standardisierung von Werkzeugen + Formaten Anwendungsspezifik



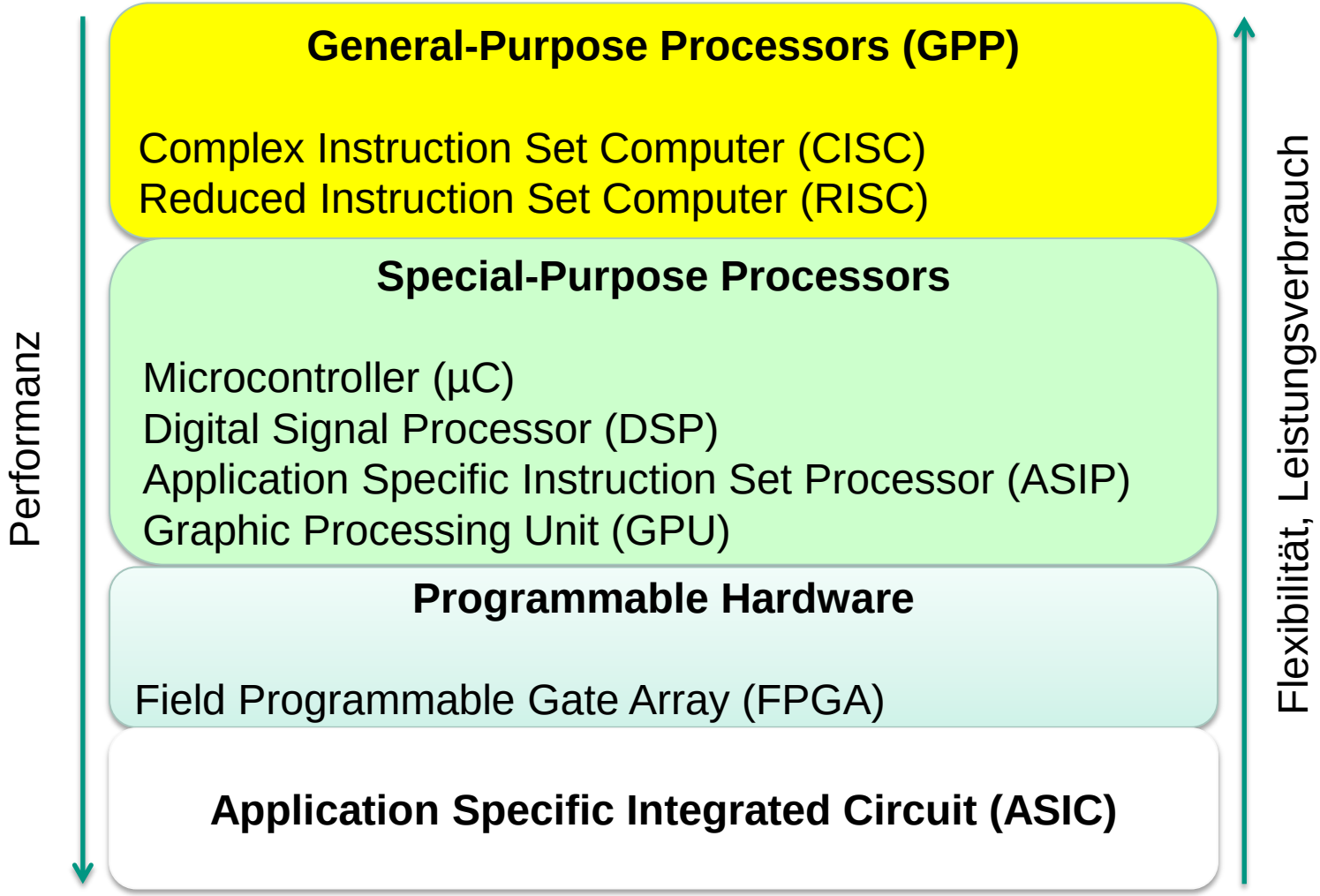


Inhalt

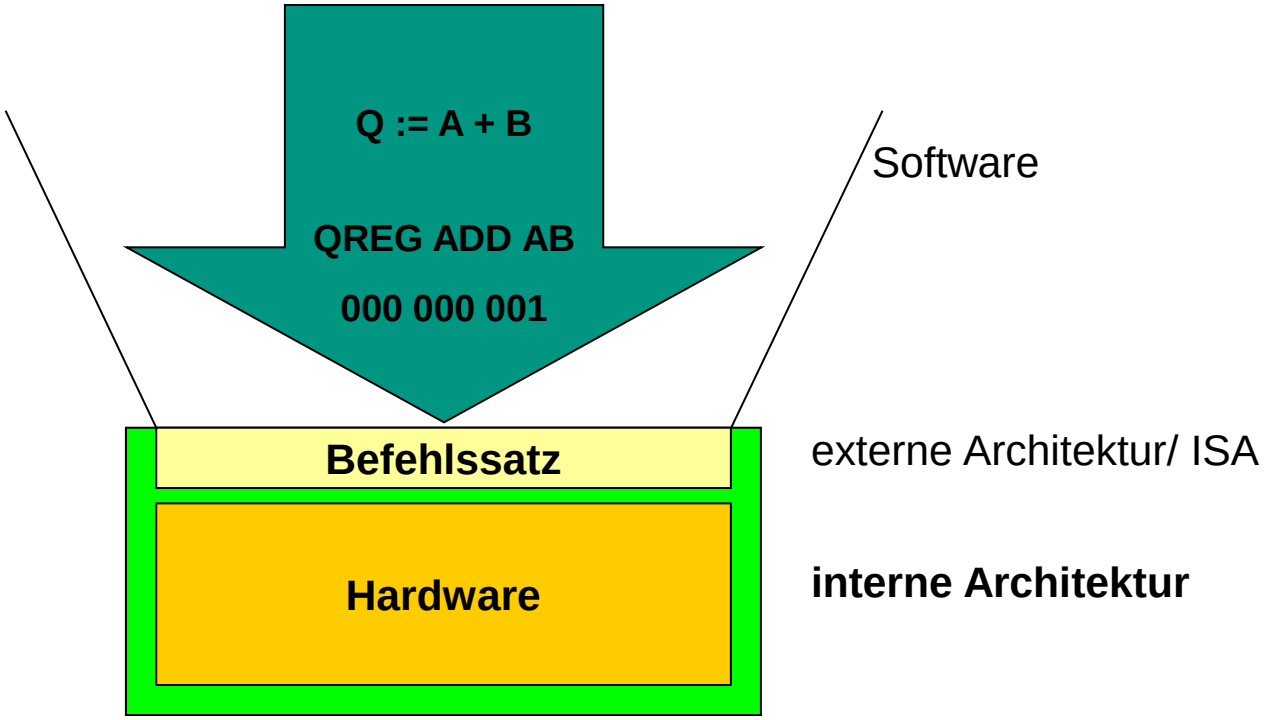
Inhalt

- 2.4 Spezialprozessoren
  - 2.4.1 Mikrocontroller (µC)
  - 2.4.2 Digitale Signalprozessoren (DSPs)
  - 2.4.3 Grafik Prozessoren (GPU)
- 2.5 Bus, Network-on-Chip (NoC) & Multicore
- 2.6 Anwendungs-spezifische Instruktionssatz Prozessoren (ASIP)
- 2.7 Field Programmable Gate Arrays (FPGA)
- 2.8 System-on-Chip (SoC)

2 Vergleich der Zielarchitekturen



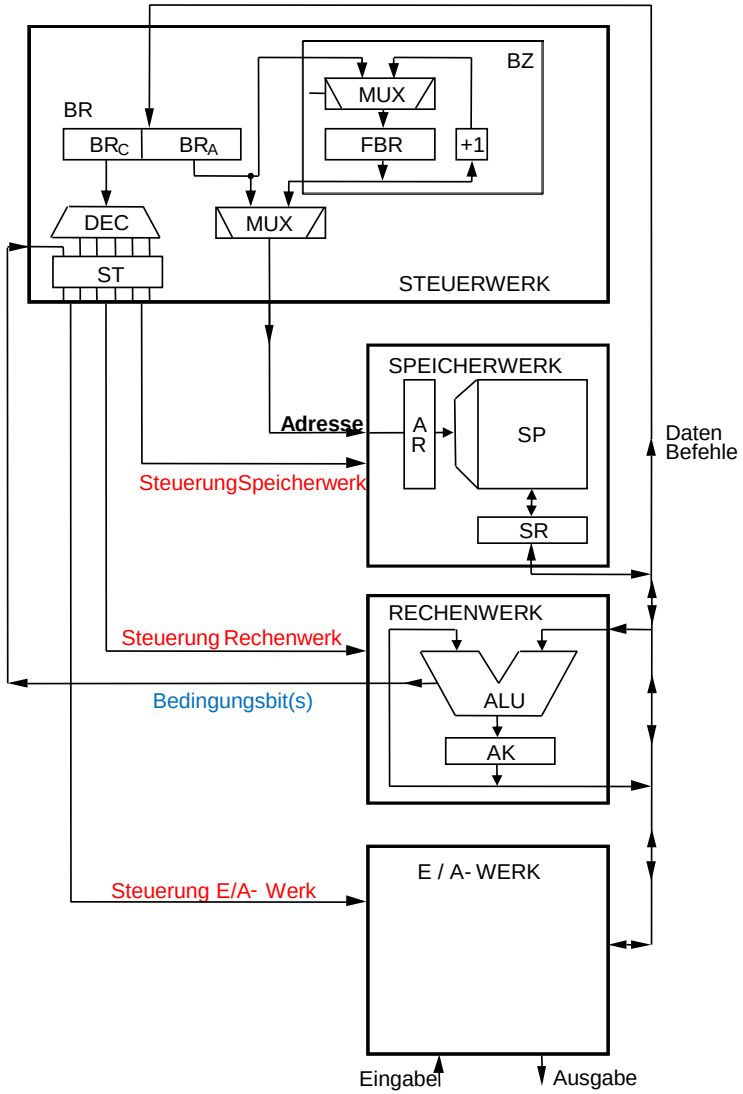
2 Rechnerarchitekturen



- Interne Architektur (Prozessor Mikroarchitektur) definiert die interne Struktur des Prozessors
  - Verschiedene interne Architekturen können dieselbe externe Architektur aufweisen
  - Computerfamilie

2.1 Aufbau eines einfachen Prozessors

- Steuerwerk
  - BR: Befehlsregister
    - BR_C: Codeteil
    - BR_A: Adressteil
  - ST: Register für Steuerbits
  - BZ: Befehlszähler
- Speicherwerk
  - AR: Adresszähler
  - SP: Speicher
  - SR: Speicherregister
- Rechenwerk
  - ALU: Arithmetische-, logische Einheit
  - AK: Akkumulator
- E/A-Werk
  - E/A: Eingabe/Ausgabe



Oliver Sander
Hardware/Software Co-Design

2.1.1 Steuerwerk

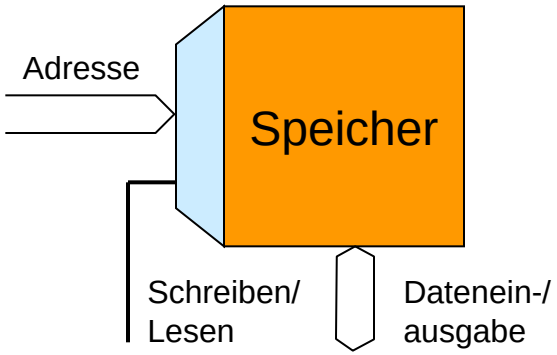
- Koordination der Komponenten des Prozessors:
  - Versorgung mit Daten und Adressen
  - Selektion von Geräten
  - Auswahl von Operanden und Operationen

- Ursprünglich Steuerung durch den Bediener, später Automatisierung durch zeitliche Abfolge von Binärinformationen

- Formalisierung der Anweisung nötig:
  - Befehl: (Maschinenlesbare) Anweisung, aus der hervorgeht, welche Operation mit welchen Operanden durchgeführt werden soll
  - Befehlsformat: Festlegung, wie die für einen Befehl notwendigen Angaben dargestellt werden

2.1.2 Speicherwerk

- Im einfachsten Fall nur eine Reihe von Registern zur Aufbewahrung von Operanden, Bedingungsvektor und Ergebnis.
- I.A. jedoch ein mehr oder weniger umfangreicher, adressierbarer Schreib/Lese-Speicher.
- Ermöglicht es, während der Verarbeitung eine große Zahl von Operanden und (Zwischen-)Ergebnissen bereitzuhalten, um so nicht für jede Operation das Ein-/ Ausgabewerk benutzen zu müssen.
- Die Zahl der verfügbaren Speicherzellen richtet sich nach der Breite des Adressvektors.



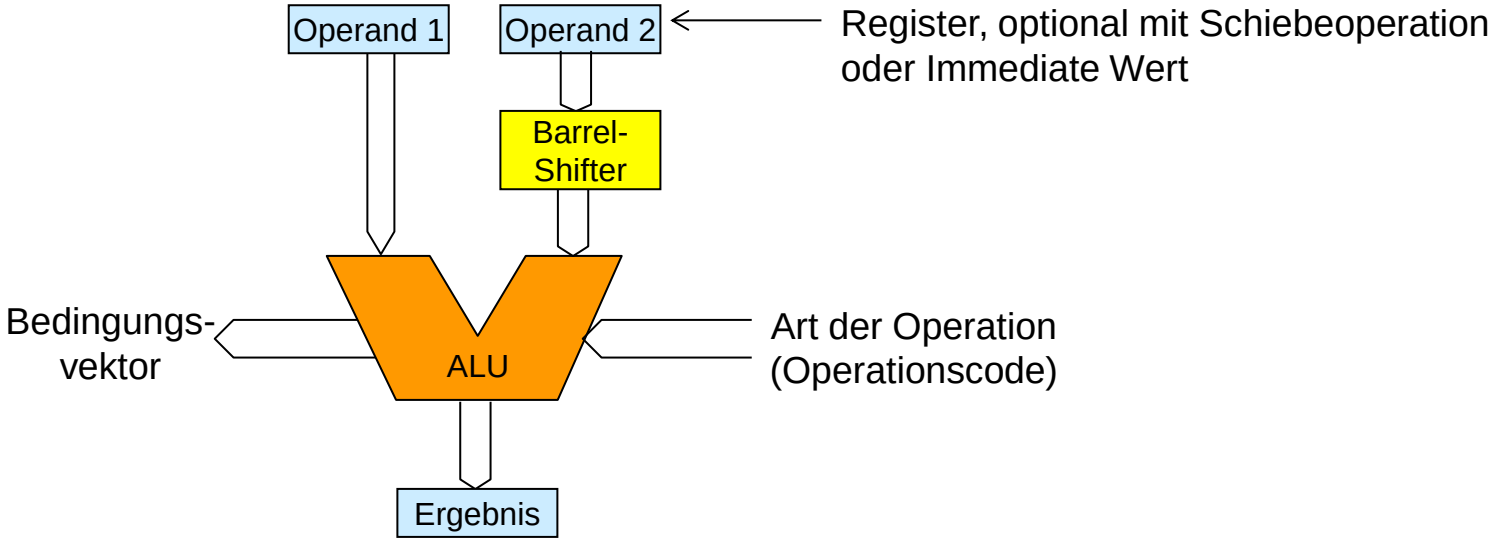
2.1.3 Rechenwerk

- Verarbeitung der Daten
- Stellt einen Satz von Basisfunktionen aus dem Bereich der numerischen und logischen Verknüpfungen zur Verfügung
- Beispiele für ein einfaches Rechenwerk:
  - Arithmetisch
    - Addition, Subtraktion
  - Logisch
    - Konjunktion, Disjunktion, Vergleich
    - Negation
  - Komplementbildung
  - Links-/Rechtsverschiebung
  - Links-/Rechtsrotation

2.1.3 Rechenwerk – Arithmetisch-logische Einheit

- Üblicherweise Beschränkung auf Funktionen mit zwei Operanden bei gegebener maximalen Stellenzahl
  - Genauigkeit (Bitbreite der Architektur)

- Einzelne Merkmale eines Operanden oder Ergebnisses (z.B. Wert gleich Null) können als Binärwerte in einem sogenannten Bedingungsvektor („flags“) zur Verfügung gestellt werden
  - Carry, Negative, Zero, Overflow

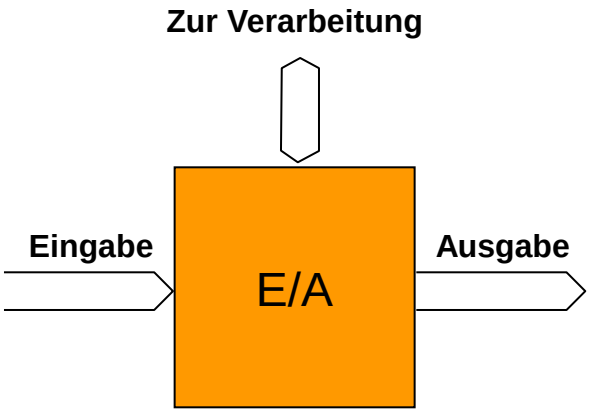


2.1.4 Eingabe-/Ausgabewerk

- Kommunikation mit dem Benutzer

- Eingabe:
  - Datenübernahme von extern angeschlossenen Geräten
  - Umsetzung in eine geeignete, normierte Darstellung zur weiteren Verarbeitung

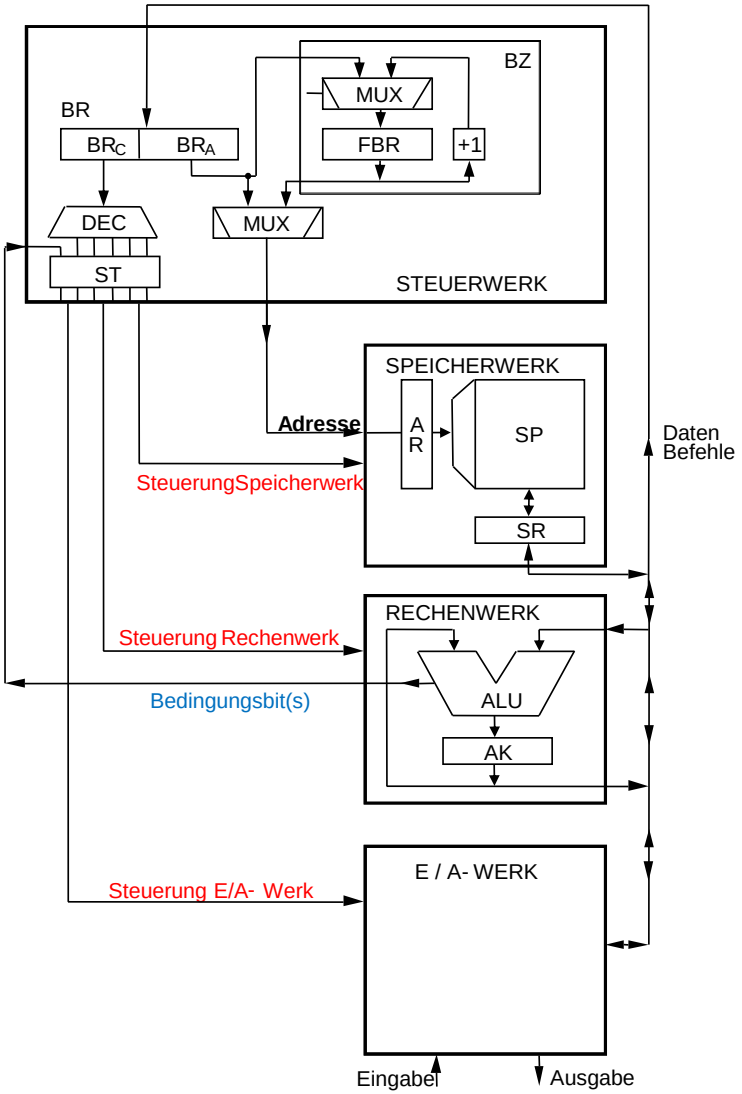
- Ausgabe:
  - Aufbereitung des Datenformats
  - Weiterleitung an externe Ausgabegeräte
  - Eventuell Auswahl des Ausgabegerätes



2.1.5 Von-Neumann-Zyklus (Arbeitsweise)

1. FETCH: Befehl aus dem Speicher in den Prozessor holen (lesender Zugriff)
2. DECODE: Befehl im Steuerwerk dekodieren
3. FETCH OPERANDS: gegebenenfalls Operanden aus dem Speicher oder der Peripherie in den Prozessor holen (lesender Zugriff)
4. EXECUTE: Befehl im Rechenwerk ausführen, ggf. Veränderung des Sprungzählers
5. WRITE BACK: gegebenenfalls Ergebnis in den Speicher oder die Peripherie schreiben (schreibender Zugriff)

Weiter bei Schritt 1



Oliver Sander
Hardware/Software Co-Design

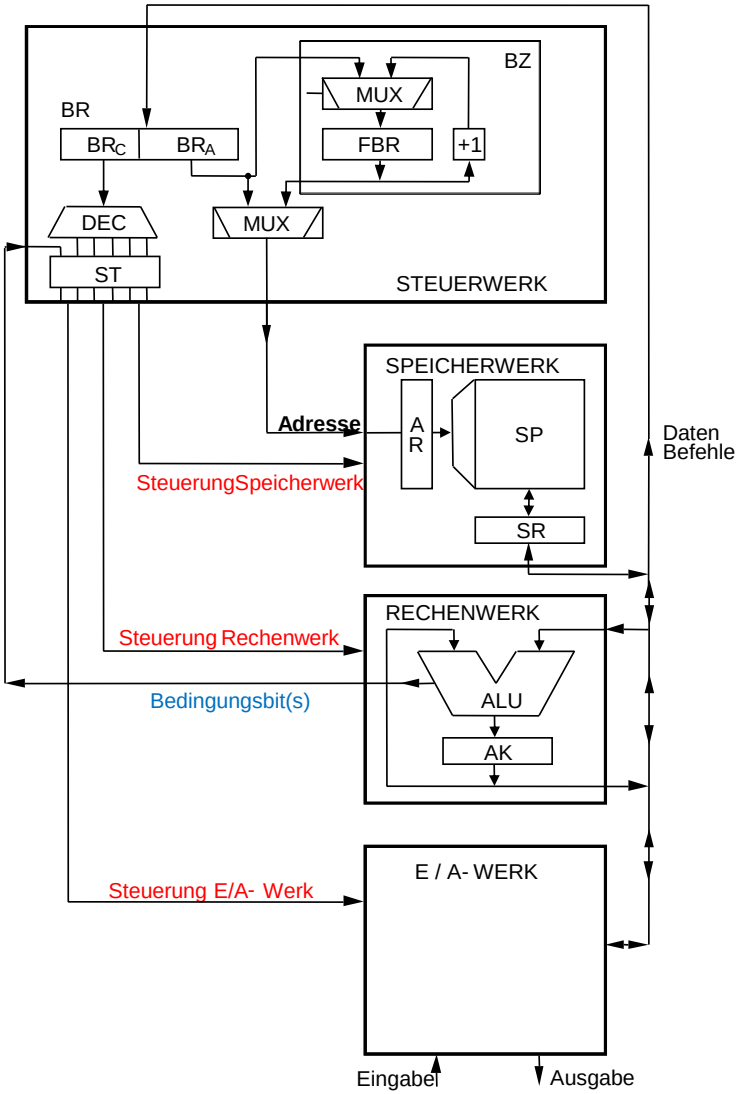
2.1.6 Befehlstypen

Befehle:

1. Transportbefehle
   - Externer Transport (Arbeitsspeicher, E/A-System)
   - Interner Transport (Register-Register)
2. Verarbeitungsbefehle
   - Arithmetische und logische Befehle
   - Schiebebefehle
   - Vergleichsbefehle
3. Sprungbefehle
4. Systembefehle (z.B. Unterbrechungsbearbeitung, Speicherverwaltung)

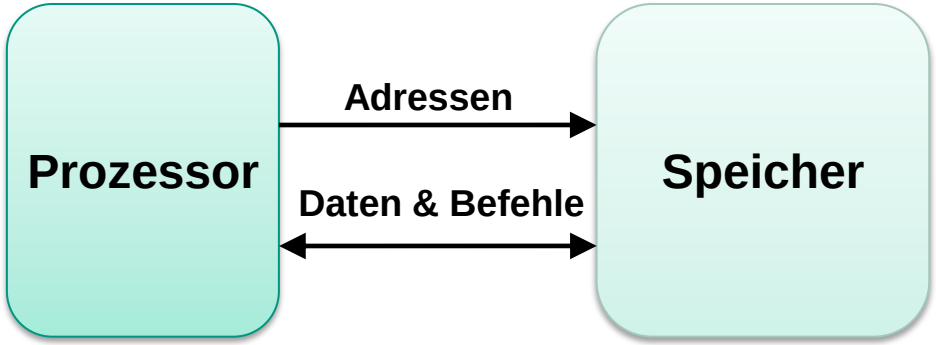
Abkürzungen:

SP	Speicher
AR	Adressregister
SR	Speicherregister
ALU	Arithmetik / Logik-Einheit
AK	Akkumulator
E/A	Ein-/Ausgabe-Werk
BR	Befehlsregister
	BR_C: Codeteil
	BR_A: Adressteil
ST	Register für Steuerbits
BZ	Befehlszähler

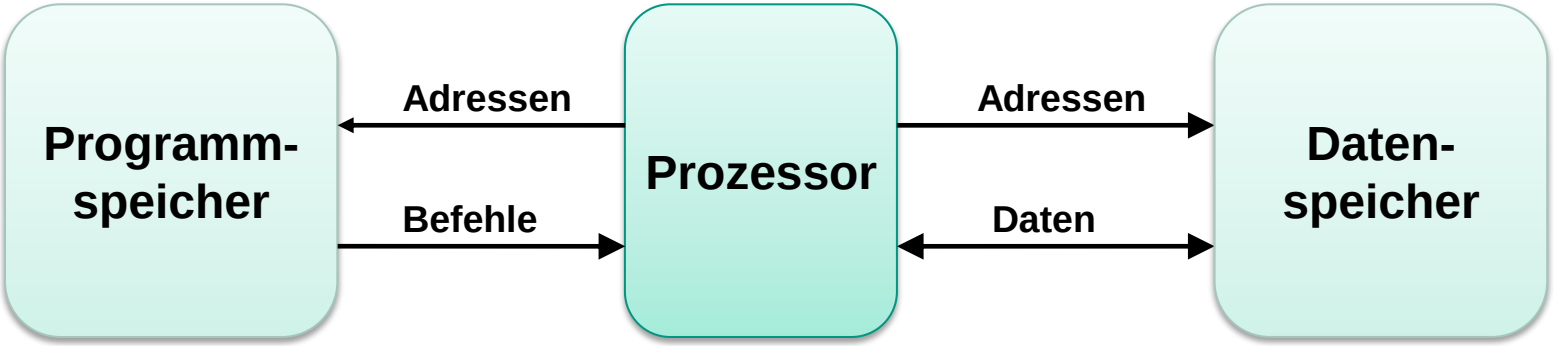


2.1.7 Speicheranbindung

- Von Neumann-Architektur:



- Harvard-Architektur:

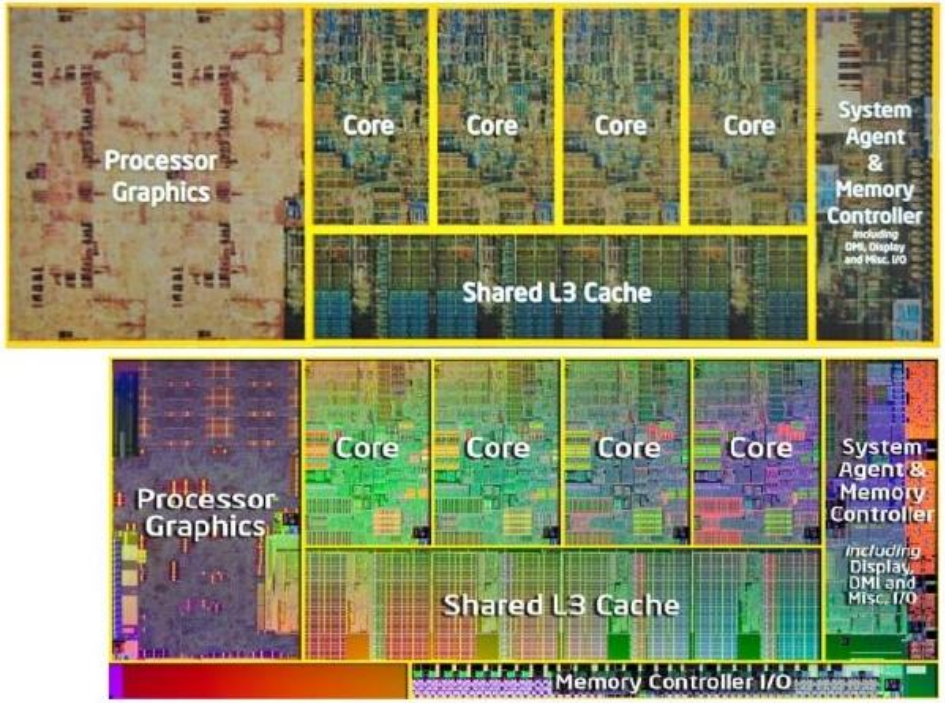


2.1.6 Beispiel: Intel Sandy Bridge vs. Ivy Bridge

- 1,16 Mrd. vs 1.4 Mrd. Transistoren
- 32nm vs. 22nm Prozess
- 212 mm^2 vs 160 mm^2 Die Größe

- QuadCore CPU
  - 3,5 GHz (3,9 GHz Turbo Mode)

- 4x32 kB I-&D-Cache (8-Wege)
- 4x256 kB L2 Cache (8-Wege)
- 8 MB L3 Cache (16-Wege)



http://www.itproportal.com/2012/04/24/intel-ivy-bridge-architecture-breakdown/

2.1.6 Beispiel: Intel Ivy Bridge

		intel inside CORE i7	intel inside CORE i7	intel inside CORE i7
Brand				
Processor Number		i7-3720QM	i7-3820QM	i7-3920XM
Price		$378	$568	$1096
TDP		45W	45W	55W
Cores/ Threads		4/8	4/8	4/8
CPU Base Freq (GHz)		2.60	2.70	2.90
Intel® Turbo Boost Technology 2.0 Max Frequency (GHz)	**SC**	3.60	3.70	3.80
	DC	3.50	3.60	3.70
	QC	3.40	3.50	3.60
DDR3/DDR3L (MHz)		1600MHz	1600MHz	1600MHz
L3 Cache		6MB	8MB	8MB
Intel® HD Graphics 4000		Yes	Yes	Yes
Gfx Base Render Frequency		650MHz	650MHz	650MHz
Graphics Max Dynamic Frequency		1250MHz	1250MHz	1300MHz
PCIe Generation 3 Support		Yes	Yes	Yes
Intel® Secure Key		Yes	Yes	Yes
Intel® OS Guard		Yes	Yes	Yes
Intel® AES/TXT/vPro Technology		Yes	Yes	Yes
Intel® Virtualization Technology		Yes	Yes	Yes
Package		rPGA/ BGA-1224*	rPGA/ BGA-1224*	rPGA

- Was ist ein Prozessor?
- Wie ist ein einfacher Prozessor aufgebaut?
  - Wie ist ein Rechenwerk aufgebaut?
- Wie kann die Speicheranbindung klassifiziert werden?
  - Ist der Bus unidirektional?
  - Was ist der Unterschied zwischen primären und sekundären Speicher?

Inhalt

2.2 Flynn’sche Taxonomie (I)

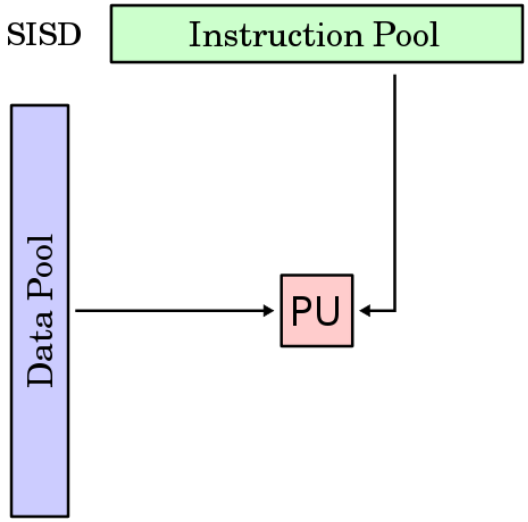
- Duale Unterscheidungsmerkmale:
  - Wie viele Instruktionen werden gleichzeitig ausgeführt?
    - Anzahl der Instruktionsströme (Hauptspeicher Prozessor)
  - Wie viele Datenworte verarbeitet eine Instruktion?
    - Anzahl der Datenströme (Hauptspeicher Prozessor Hauptspeicher)
  - Je Datenstrom und Instruktionsstrom zwei Klassen:
    - eins (single) oder viele (multiple)

Flynn’sche Klassifikation	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

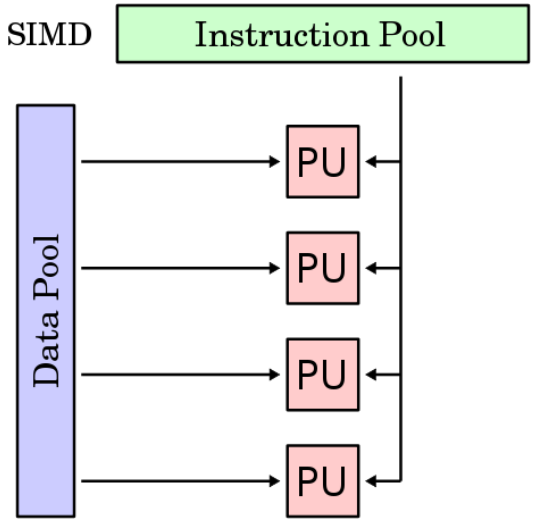
Flynn, Michael J.; , "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, vol.C-21, no.9, pp.948-960, Sept. 1972

2.2 Flynn’sche Taxonomie (II)

- Single Instruction, Single Data (SISD)
  - Von-Neumann-Architektur
    - gemeinsamer Daten- und Instruktions-Speicher
  - Harvard-Architektur
    - getrennter Daten- und Instruktions-Speicher

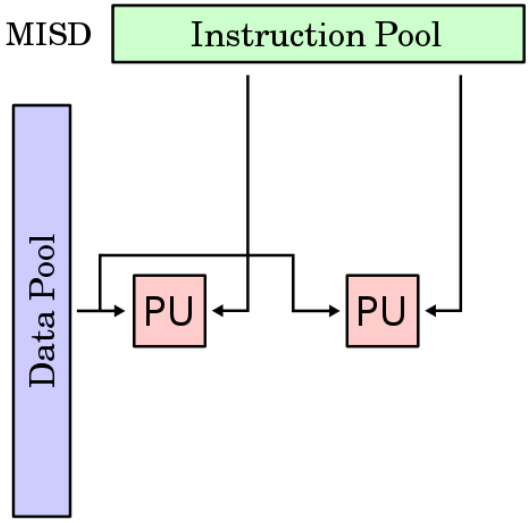


- Single Instruction, Multiple Data (SIMD)
  - Vektorprozessor
    - schnelle Ausführung gleichartiger Rechenoperationen auf mehrere gleichzeitig eintreffende oder zur Verfügung stehende Eingangsdatenströme

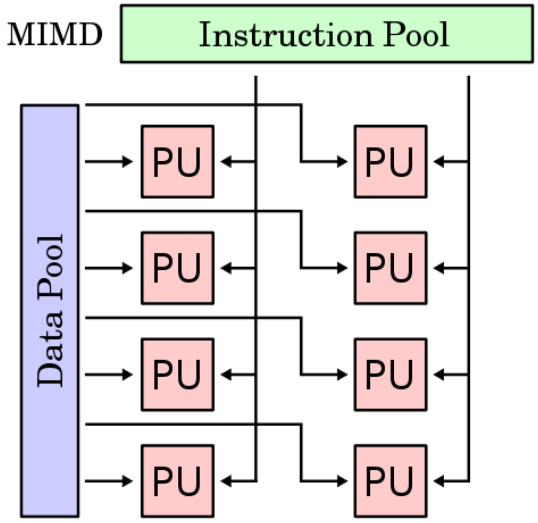


2.2 Flynn’sche Taxonomie (III)

- Multiple Instruction, Single Data (MISD)
  - Fehlertolerante, redundante Rechner
    - Redundante Datenströme zur Fehlererkennung bzw. –korrektur



- Multiple Instruction, Multiple Data (MIMD)
  - Multiprozessorsysteme
    - Führen gleichzeitig verschiedene Operationen auf verschieden gearteten Eingangsdatenströmen durch
    - Nebenläufige Kontrollflüsse
    - Kopplung über Verbindungs-Netzwerke



- Wie können Prozessoren klassifiziert werden?
- Welche Beispiele gibt es für die jeweiligen Klassen?
- Lassen sich alle heutigen Prozessoren eindeutig klassifizieren?

Inhalt

2.3 General Purpose Prozessor (GPP) (I)

- Eigenschaften
  - hohe Performanz für großes Anwendungsspektrum, nicht für eine spezielle Anwendung optimiert universell
  - große Leistungsaufnahme i.a. hohe Taktraten
- Entwicklung
  - hochoptimierte Schaltungsstrukturen Full-Custom + Standardzellen
  - Entwicklungszeit 100 Personenjahre
  - nur in sehr großen Stückzahlen rentabel
- Einsatzgebiete
  - PCs, Workstations
  - Laptops

2.3.1 GPP: Architekturen

- Klassifizierung externer Architekturen
  - Wie werden Operanden von Befehlen im Prozessor gespeichert
  - Anzahl der Operanden pro Befehl
  - Residenz von Operanden
  - Operationsvorrat
  - Typ und Länge der Operanden

- Beispielklassen:
  - Akkumulator-Maschine
  - Stack-Maschine
  - Registersatzmaschine
    - RISC-Architektur
    - CISC-Architektur

2.3.1.1 GPP-Arch: Akkumulatormaschine (I)

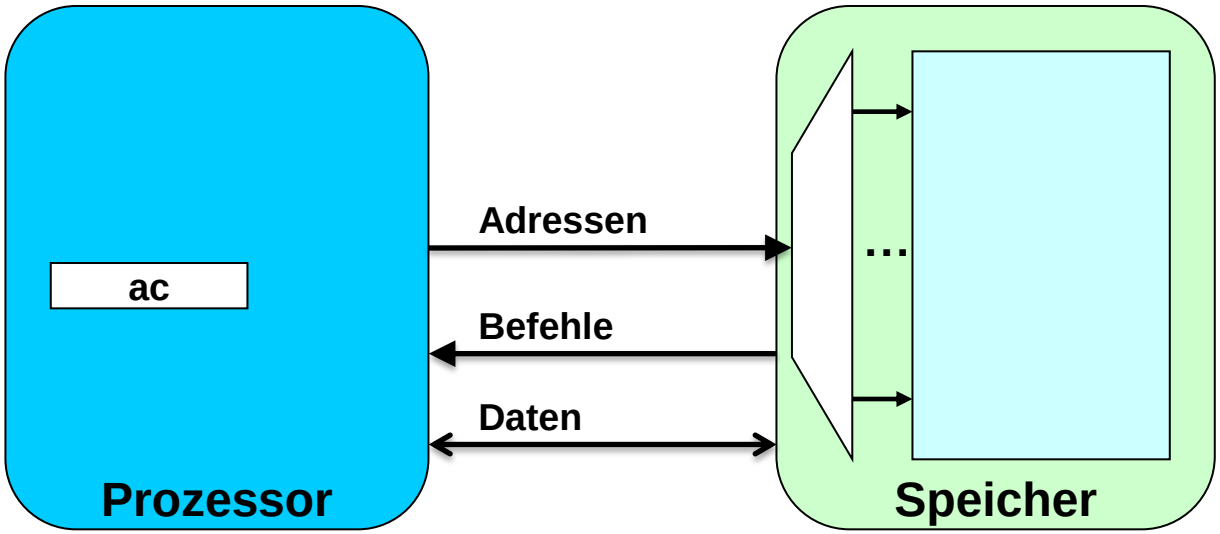
- Organisationsprinzip vieler einfacher, früherer Rechner, aber auch z.B. Motorola 6809

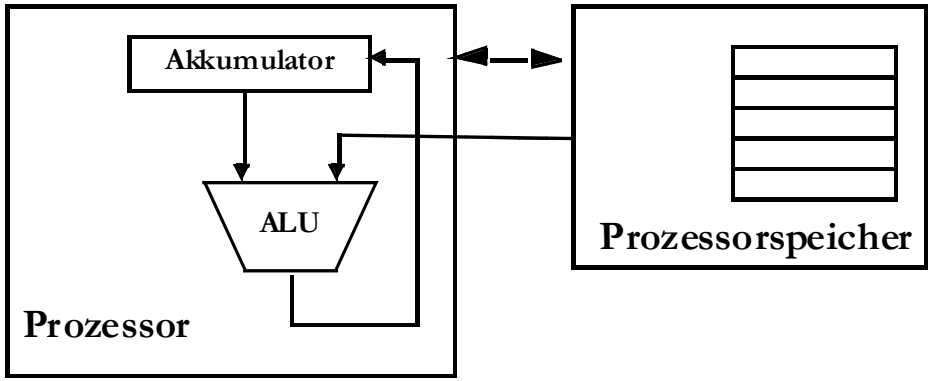
- Die Rechenergebnisse wurden in einem Register, dem Akkumulator (AC), „akkumuliert“
  - LAC m Lade Wert unter Adresse m in den AC
  - SAC m Speichere den Inhalt von AC nach Adresse m
  - ADD m Addiere den Inhalt von AC mit dem Wert unter Adresse m und speichere das Resultat nach AC

- Beispiel: c := a + b
  - LAC a
  - ADD b
  - SAC c

2.3.1.1 GPP-Arch: Akkumulatormaschine (II)

- Struktur:





2.3.1.2 GPP-Arch: Stackmaschine (I)

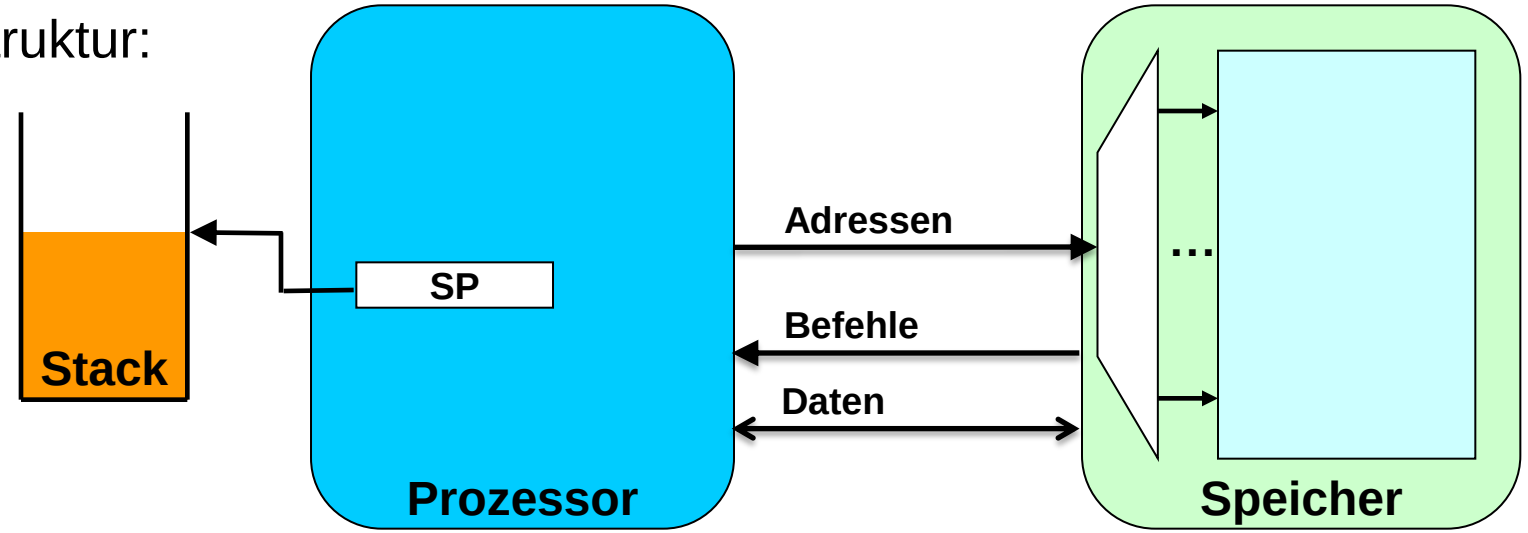
- Befehle beziehen sich auf einen Stack zur Speicherung u.a. von Zwischenergebnissen

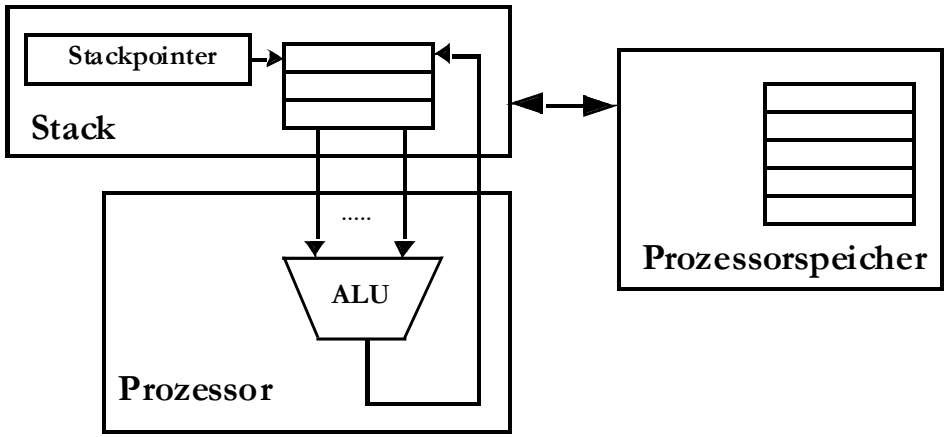
- Typische Befehle:
  - PUSH m Lege den Wert unter Adresse m auf den Stack
  - ADD Addiere die beiden obersten Werte des Stacks, entferne sie und lege die Summe als oberstes Element auf den Stack
  - STORE m Speichere das oberste Element auf dem Stack nach der Adresse m

- Beispiel: c := a + b
  - PUSH a
  - PUSH b
  - ADD
  - STORE c

2.3.1.2 GPP-Arch: Stackmaschine (II)

- Struktur:





2.3.1.3 GPP-Arch: Registersatzmaschine (I)

- Der Prozessor enthält z.B. 32 Universalregister

- Typische Befehle:
  - Datenverarbeitungs-Instruktionen
    - Arithmetic, Logisch, Compare, Move
  - Verzweigungs-Instruktionen
    - Branch
  - Register-Daten-Transfer-Instruktionen
    - Load, Store

- Üblich sind Dreiadressbefehle oder Zweiadressbefehle
  - OP DEST, SRC1, SRC2 DEST := SRC1 OP SRC2
  - OP DEST, SRC DEST := DEST OP SRC
    - Beispiel: ADD R5, R6 bedeutet R5 := R5 + R6

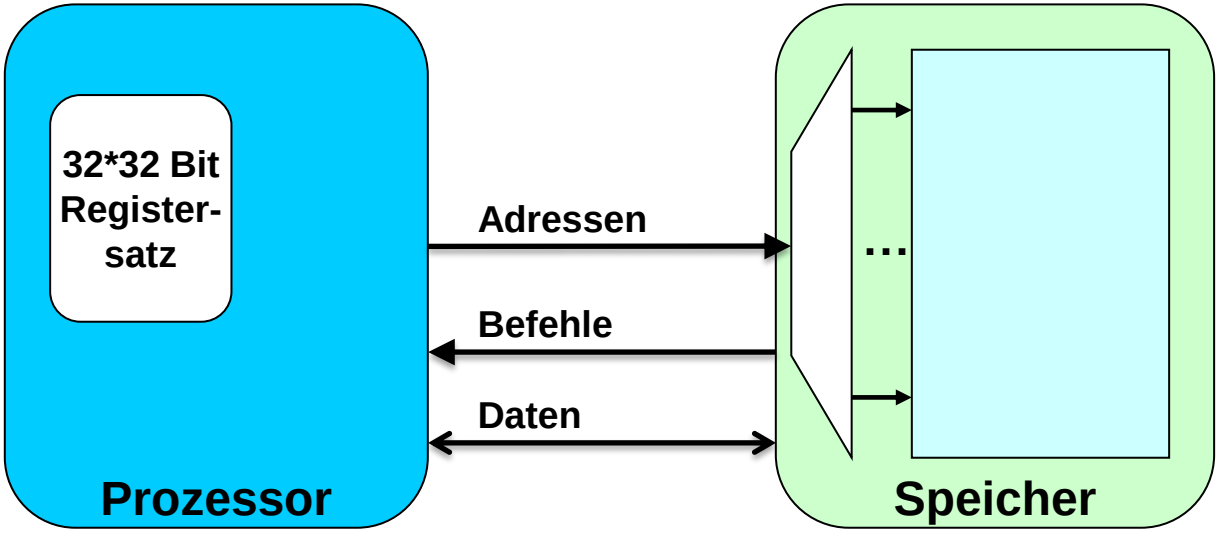
2.3.1.3 GPP-Arch: Registersatzmaschine (II)

- Complex instruction set computer
  - Schwerpunkt auf Hardware
  - Mehrere Taktzyklen pro komplexer Instruktion
  - Memory-to-Memory-Architektur
    - Load/Store sind eingebaute Instruktionen
  - Kleine Codegrößen
  - Benötigt mehr Transistoren zum Speichern komplexer Instruktionen
  - Beispiel: c := a + b
    - MOVE R1, a
    - ADD R1, b
    - MOVE c, R1
- Reduced instruction set computer
  - Schwerpunkt auf Software
  - Nur ein Taktzyklus pro reduzierter Instruktion
  - Load-Store-Architektur
    - Load-Store sind unabhängige Instruktionen
  - Große Codegrößen
  - Benötigt mehr Transistoren für Speicherregister
  - Beispiel: c := a + b
    - LOAD R1, a
    - LOAD R2, b
    - ADD R3, R2, R1
    - STORE c, R3

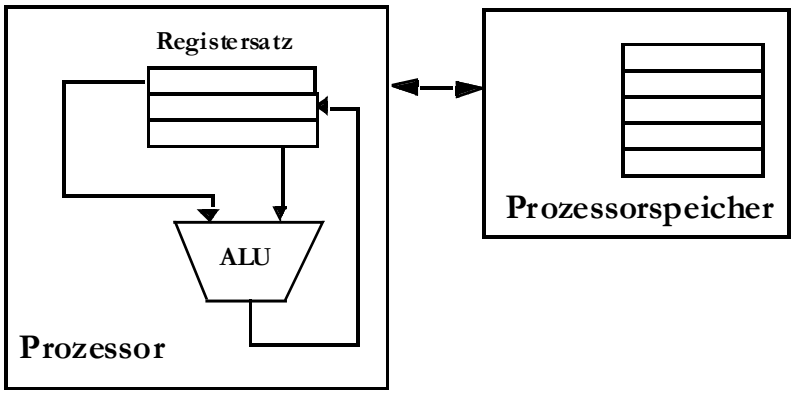
http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/risc/risccisc/

2.3.1.3 GPP-Arch: Registersatzmaschine (III)

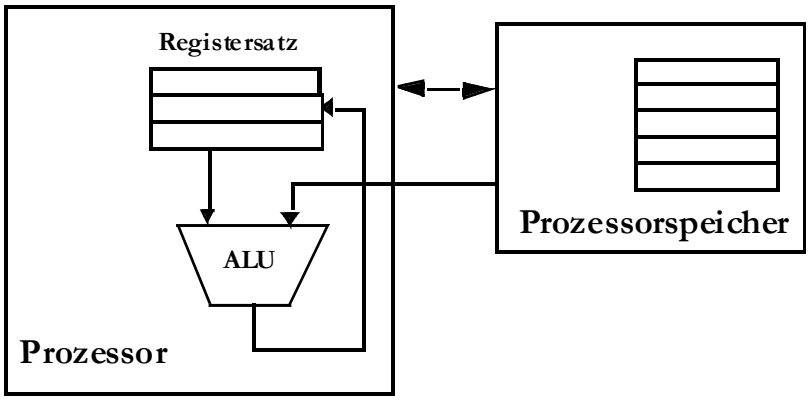
- Struktur:



- RISC



- CISC



2.3.1.3 GPP-Arch: Registersatzmaschine (IV)

- RISC vs. CISC
  - Programmgröße, transportierte Daten und Code sowie Ausführungszeiten:

$$Zeit/_{Programm} = Zeit/_{Zyklus} * Zyklen/_{Instruktion} * Instruktionen/_{Programm}$$

	Programmgröße:		Transportierte Instr. und Daten:		CPU Zeit:	
	VAX	DEC 3100	VAX	DEC 3100	VAX	DEC 3100
Gnu C Compiler	410kB	688kB	18MB	21MB	291s	90s
TeX	159kB	217kB	67MB	78MB	449s	95s
Spice	223kB	372kB	99MB	106MB	352s	94s

- RISC-Programme (DEC) dagegen sind länger bei gleichzeitig schnellerer Ausführung
- CISC-Programme (VAX) sind meist kleiner, benötigen jedoch eine längere Ausführungszeit

- Wie können General Purpose Prozessoren unterschieden werden?
- Wovon hängt die Geschwindigkeit eines GPP ab?
- Was ist der Nachteil eines RISC-Prozessors?
- Was ist ein Compiler?

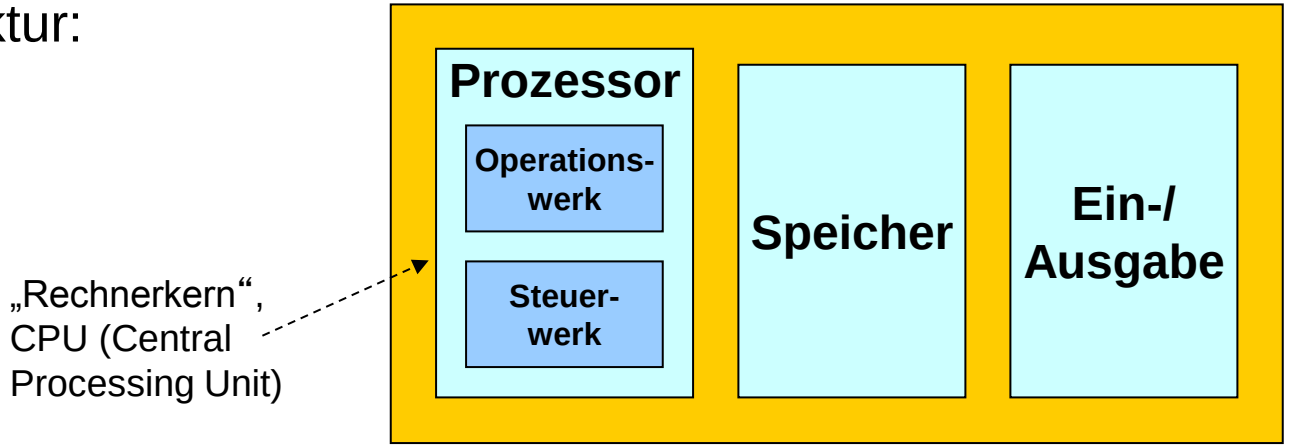
Inhalt

2.3.2 GPP: Performanzsteigerung (I)

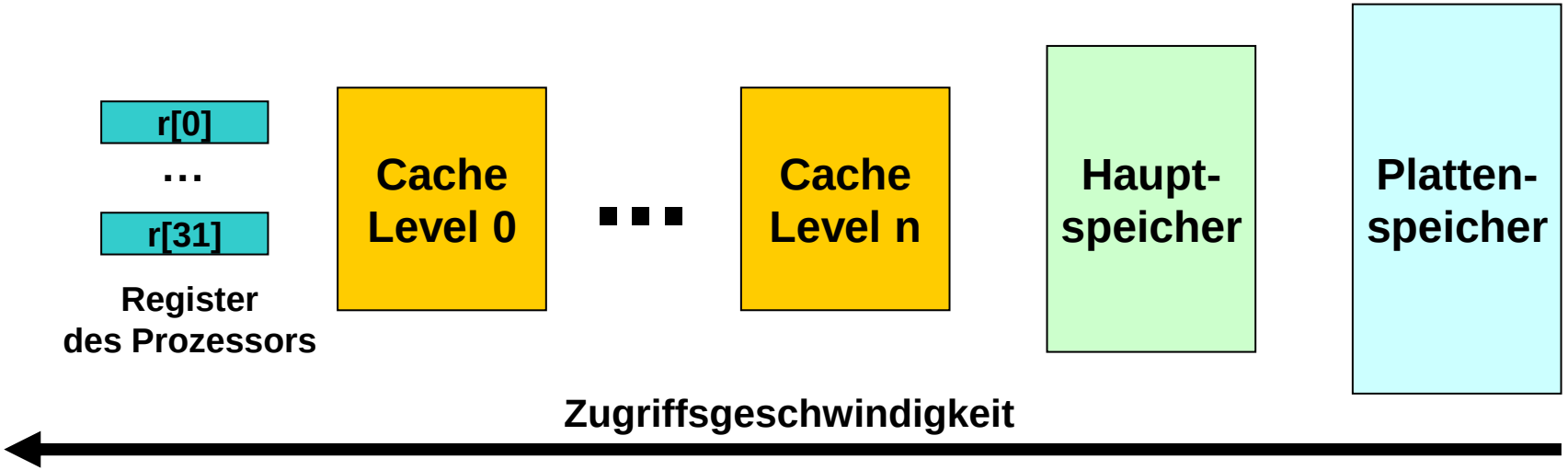
- Pipeline und tiefe Instruktions-Pipeline Superpipelining
  - z.B.: fetch | decode | read | execute | write back
  - Sprungvorhersage
- Mehrere skalare Einheiten superskalar
  - z.B.: Integer Units, Floating-Point Unit, Load/Store unit
  - Je nach skalarer Einheit verschieden tiefe Pipelines
- Dynamic Scheduling Out-of-Order Execution
  - Prozessor erkennt parallel ausführbare Instruktionen und weist sie den Einheiten zu.
  - komplexes Steuerwerk
- VLIW
  - Compiler erkennt Parallelität und plant die Abfolge der Instruktionen auf den verschiedenen Funktionseinheiten.

2.3.2 GPP: Performanzsteigerung (II)

- Interne Architektur:



- Speicherhierarchie in einem Rechner:



Inhalt

2.3.2.1 GPP: Pipelining (I)

- **Pipelining** — „Fließband-Bearbeitung“

- „Pipelines beschleunigen die Ausführungsgeschwindigkeit eines Rechners in gleicher Weise wie Henry Ford die Autoproduktion mit der Einführung des Fließbandes revolutionierte.“
  (Peter Wayner 1992)

- Befehlspipeline eines Rechners:
  Die Befehlsbearbeitung wird in n funktionelle Einheiten gegliedert.
  Ausführung geschieht zeitlich überlappend.

2.3.2.1 GPP: Pipelining (II)

- Unter dem Begriff **Pipelining** versteht man die Zerlegung einer Maschinenoperation in mehrere Phasen, die dann von hintereinander geschalteten Verarbeitungseinheiten taktsynchron bearbeitet werden, wobei jede Verarbeitungseinheit genau eine spezielle Teiloperation ausführt.

- Die Gesamtheit dieser Verarbeitungseinheiten nennt man eine **Pipeline**.

- Bei einer **Befehlspipeline / Instruction Pipeline** wird die Ausführung eines Maschinenbefehls in verschiedene Phasen unterteilt, aufeinanderfolgende Maschinenbefehle werden jeweils um einen Taktzyklus versetzt ausgeführt.

2.3.2.1 GPP: Pipelining (III)

- Jede Stufe der Pipeline heißt **Pipeline-Stufe** oder **Pipeline-Segment**.
- Pipeline-Stufen werden durch getaktete **Pipeline-Register** (auch latches genannt) getrennt.
- Ein **Pipeline-Maschinentakt** ist die Zeit, die benötigt wird, um einen Befehl eine Stufe weiter durch die Pipeline zu schieben.
- Idealerweise wird ein Befehl in einer k-stufigen Pipeline in k Takten von k Stufen ausgeführt.
- Wird in jedem Takt ein neuer Befehl geladen, dann werden zu jedem Zeitpunkt unter idealen Bedingungen k Befehle gleichzeitig behandelt und jeder Befehl benötigt k Takte, bis zum Verlassen der Pipeline.

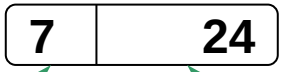
2.3.2.1 GPP: Pipelining (IV)

- Man definiert die **Latenz** als die Zeit, die ein Befehl benötigt, um alle k Pipeline-Stufen zu durchlaufen.

- Der **Durchsatz** einer Pipeline wird definiert als die Anzahl der Befehle, die eine Pipeline pro Takt verlassen können.
  Dieser Wert spiegelt die Rechenleistung einer Pipeline wieder.

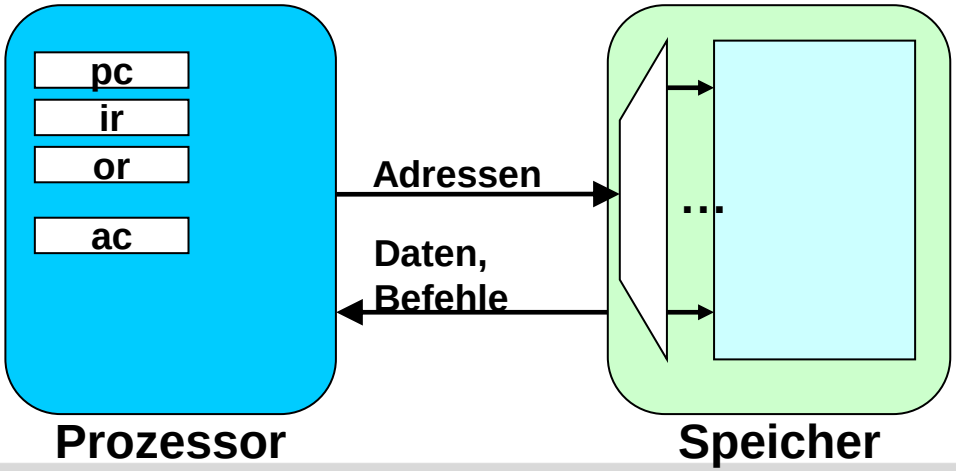
2.3.2.1 Exemplarische Befehlsausführung (I)

- Verarbeitungsphasen eines Befehls am Beispiel
  - Annahme: einfache Akkumulatormaschine
  - Befehlsformat



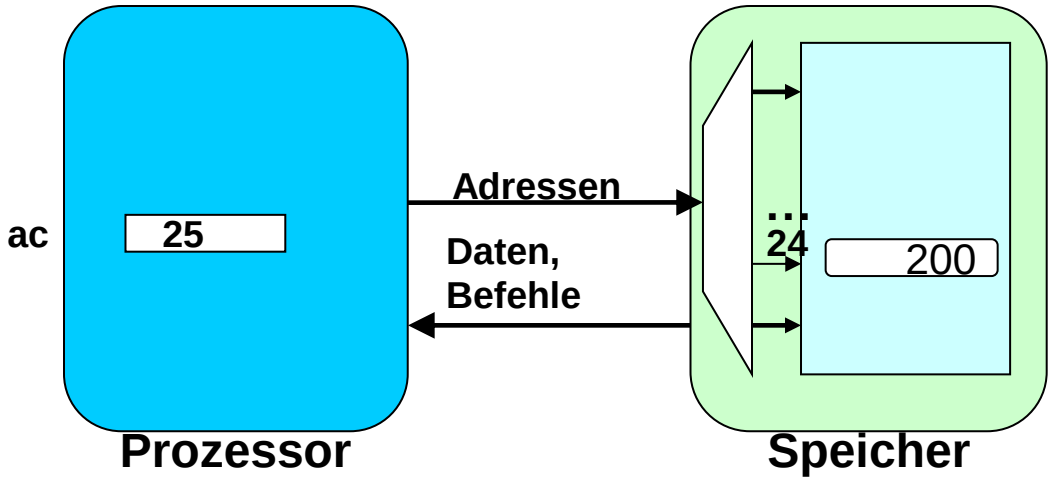
  - Befehlscode
  - Adresse 24
  - Addiere-Befehl □ 7

- Der Addiere-Befehl holt den Wert unter der im Befehl angegebenen Adresse (im Beispiel 24) und addiert ihn zum Inhalt

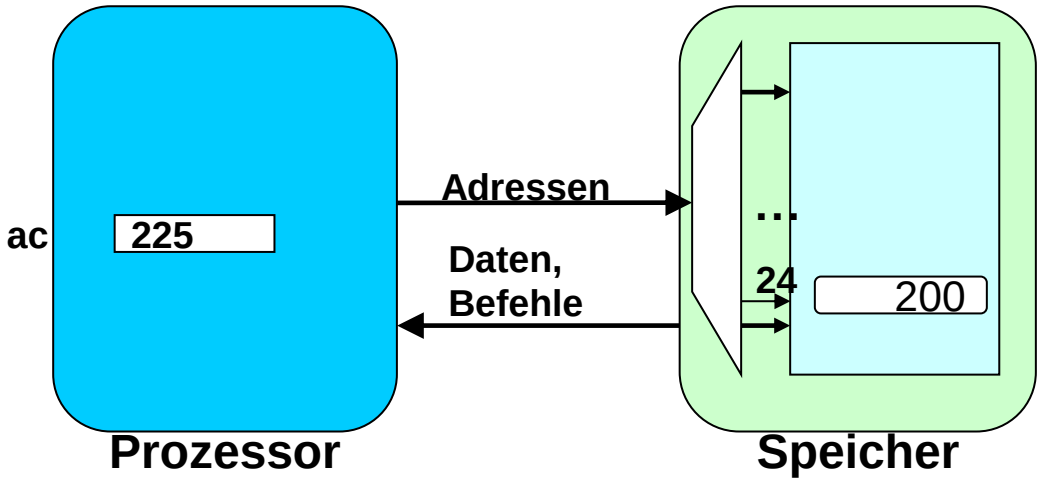


2.3.2.1 Exemplarische Befehlsausführung (II)

- Befehlscode: Addiere-Befehl □ 7 → | 7 | 24 | ← Adresse □ 7
- Vorher:

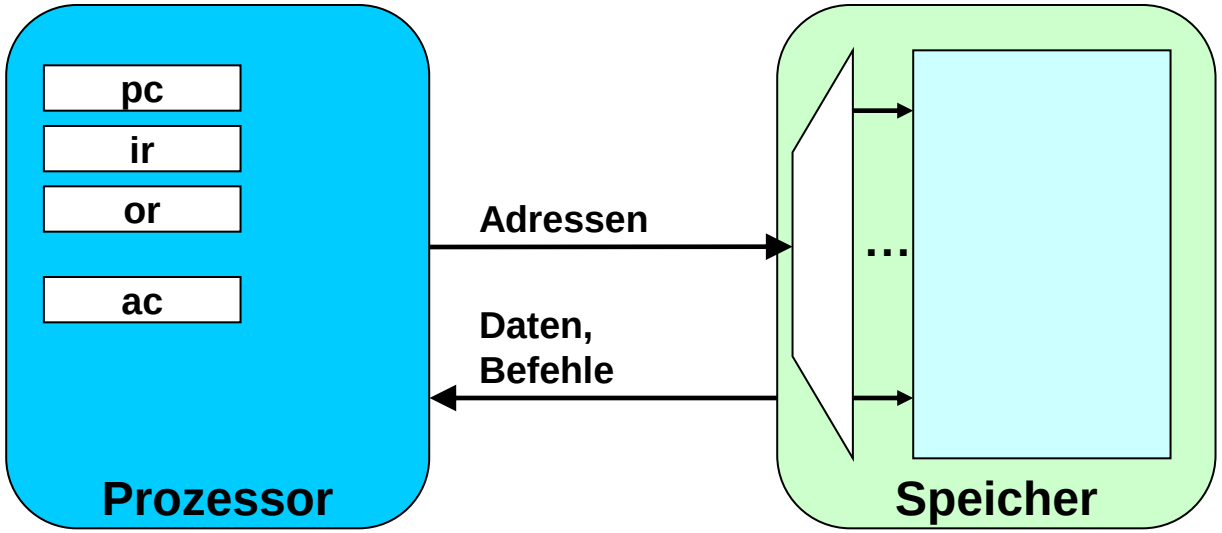


- Nachher:



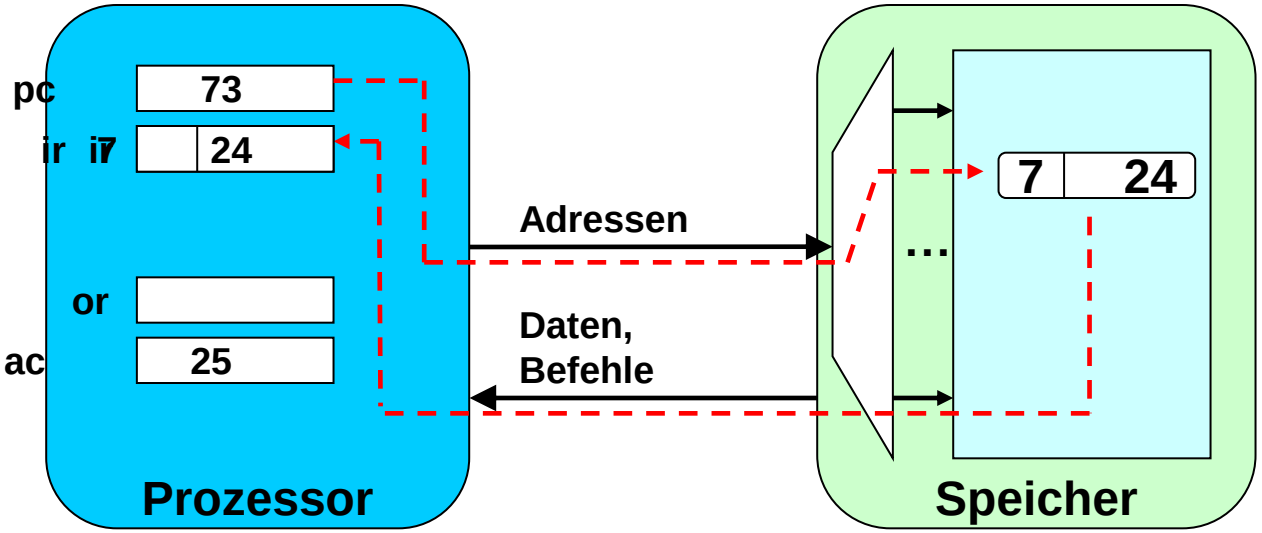
2.3.2.1 Exemplarische Befehlsausführung (III)

- Struktur:
  - pc: program counter, auch
  - ip: instruction pointer genannt
  - ir: instruction register
  - or: operand register
  - ac: accumulator



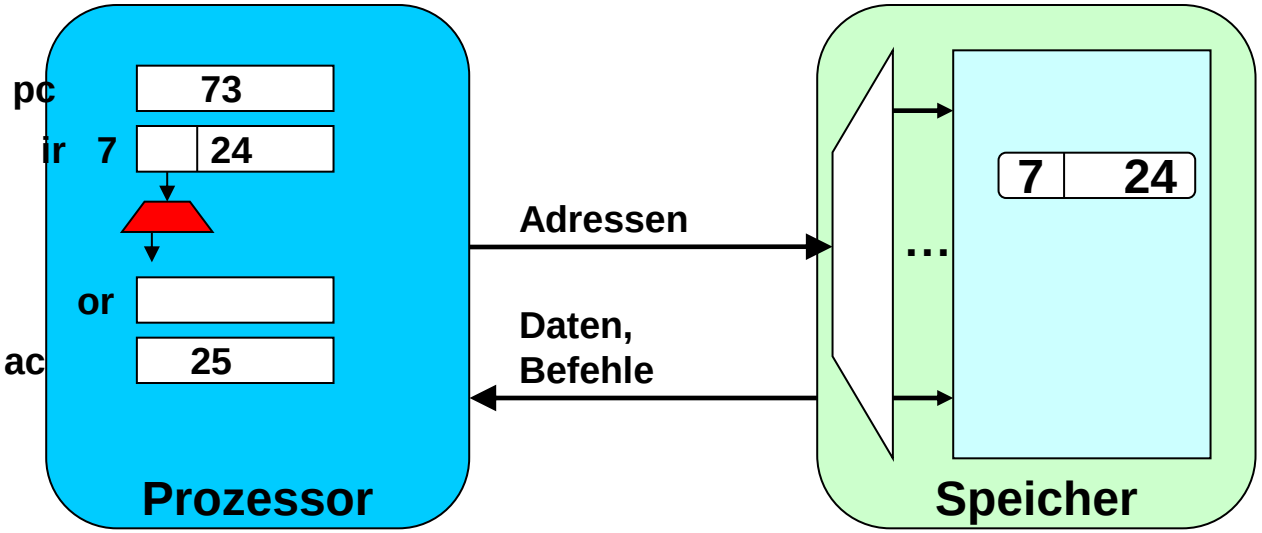
2.3.2.1 Exemplarische Befehlsausführung (IV)

- 1. Befehlsholphase (instruction fetch, IF)



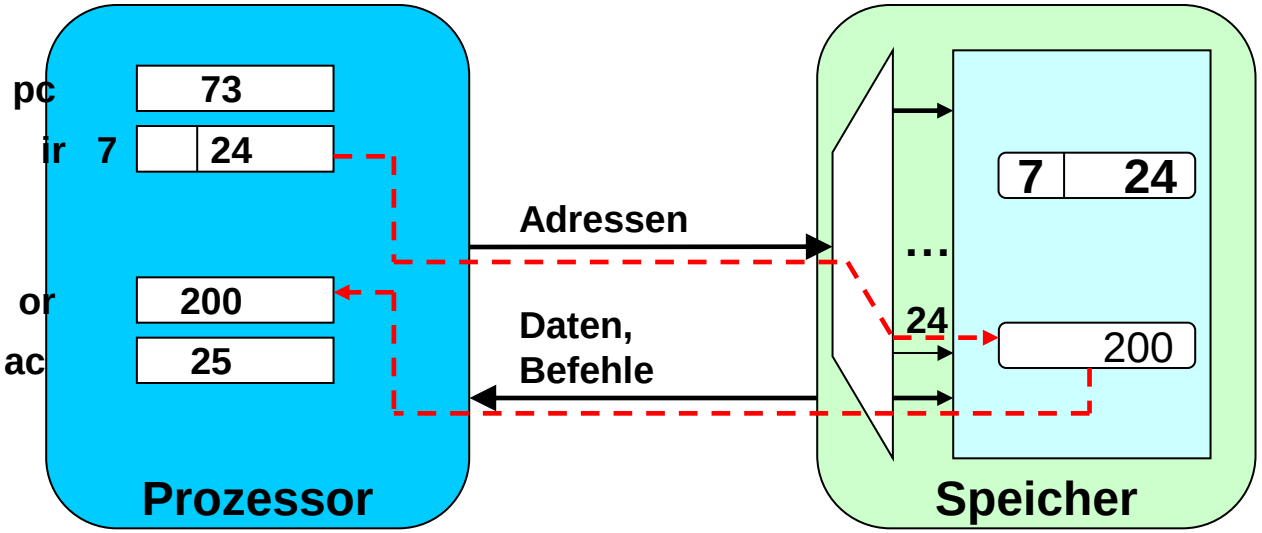
2.3.2.1 Exemplarische Befehlsausführung (V)

- 2. Befehl dekodieren (instruction decode, ID)



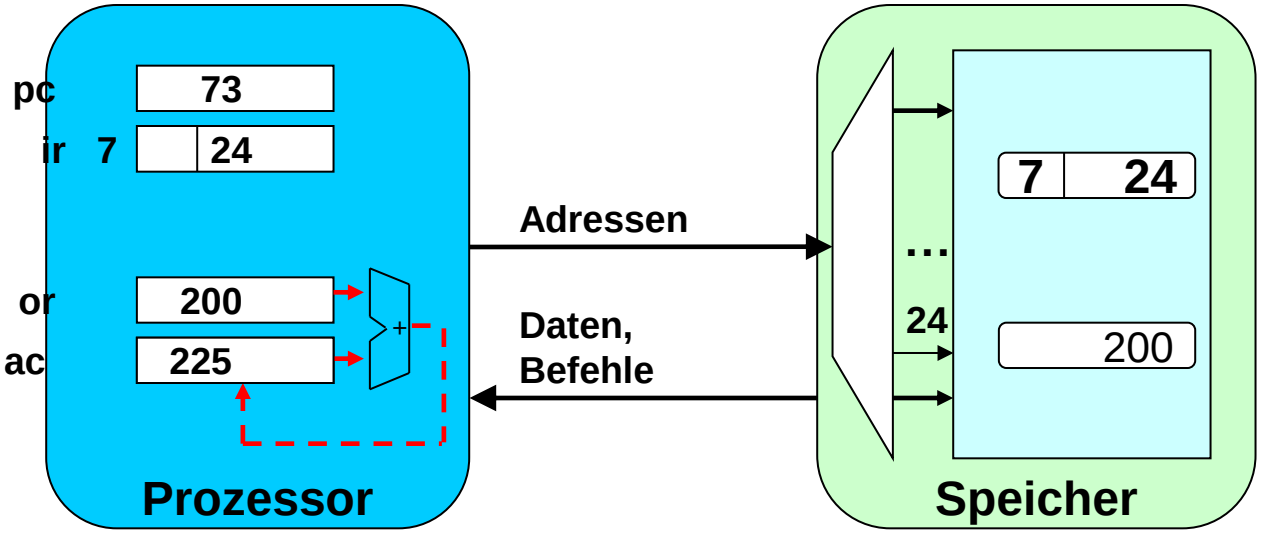
2.3.2.1 Exemplarische Befehlsausführung (VI)

- 3. Operand holen (operand fetch, OF)



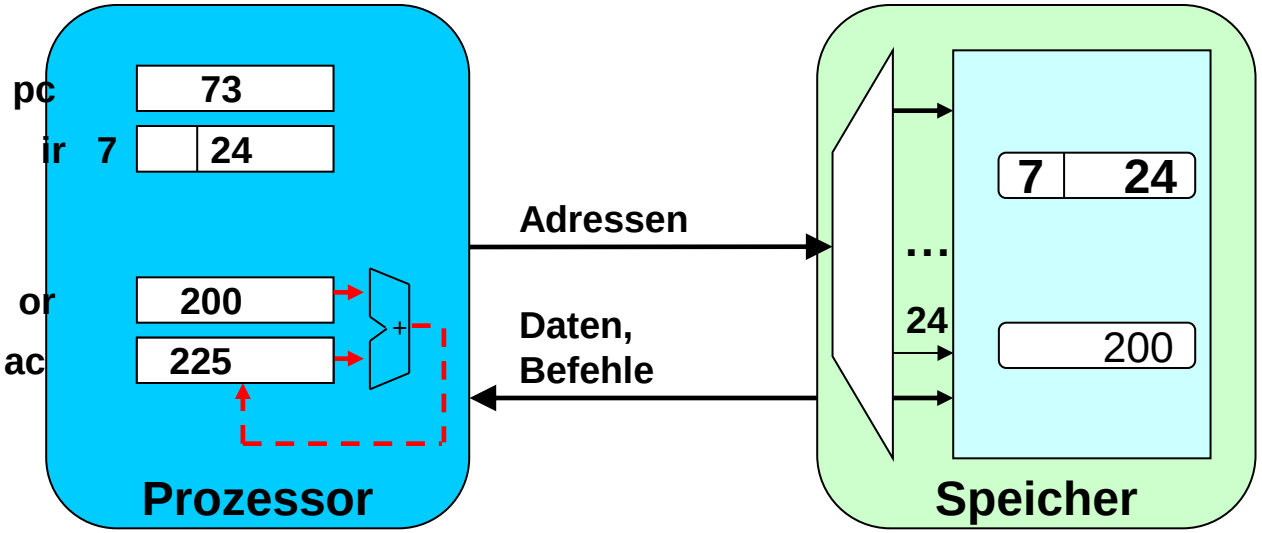
2.3.2.1 Exemplarische Befehlsausführung (VII)

- 4. Befehl ausführen (instruction execute, EX)



2.3.2.1 Exemplarische Befehlsausführung (VIII)

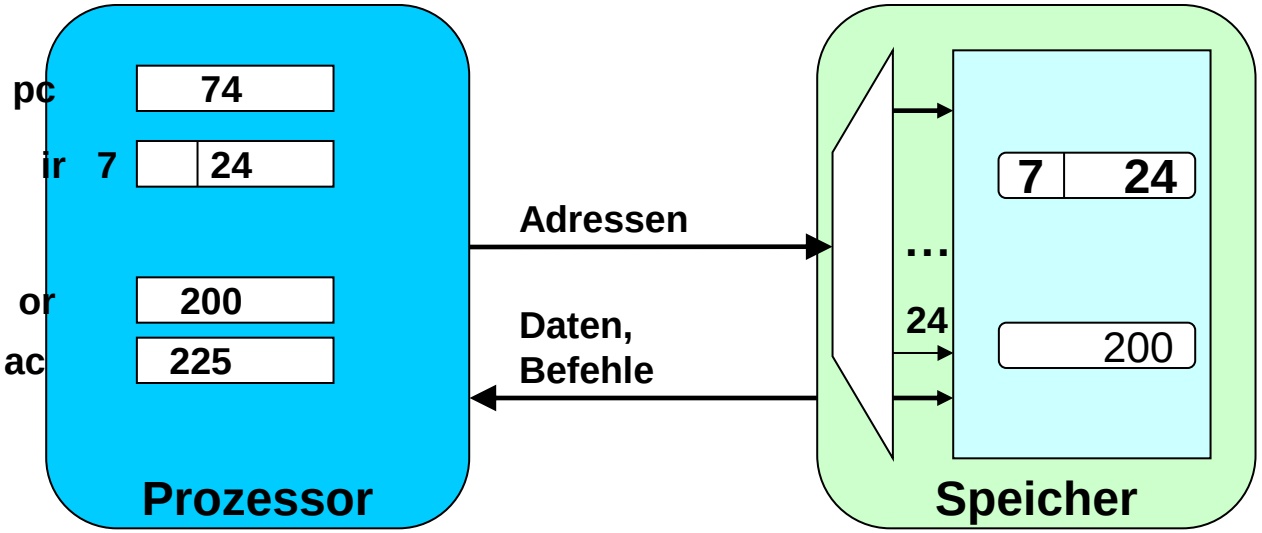
- Befehlszähler inkrementieren
  - kann als separater Schritt oder parallel erfolgen



Oliver Sander
Hardware/Software Co-Design

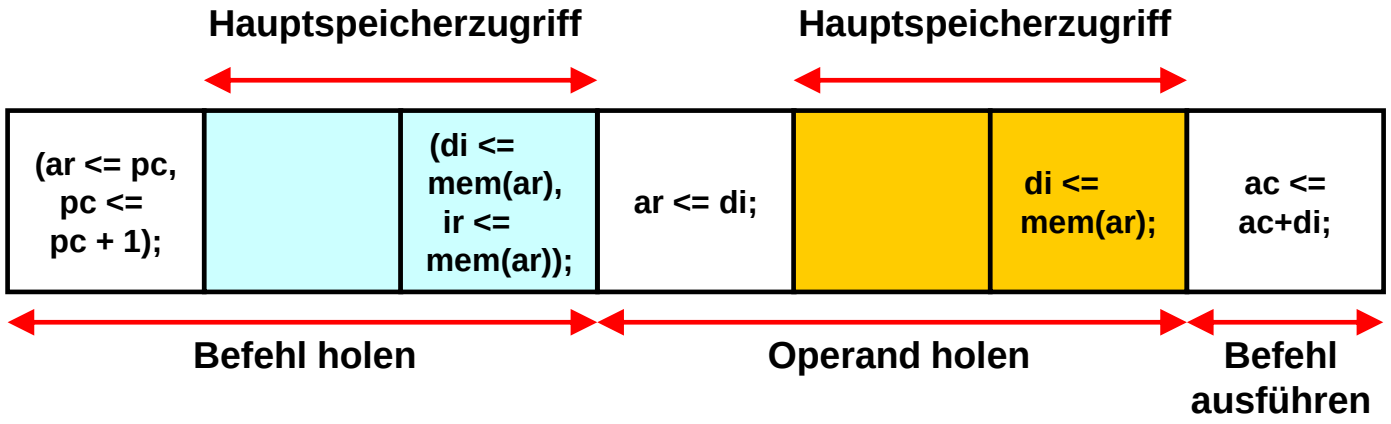
2.3.2.1 Exemplarische Befehlsausführung (IX)

- Resultat:
  - Ergebnis 225 steht im accumulator ac



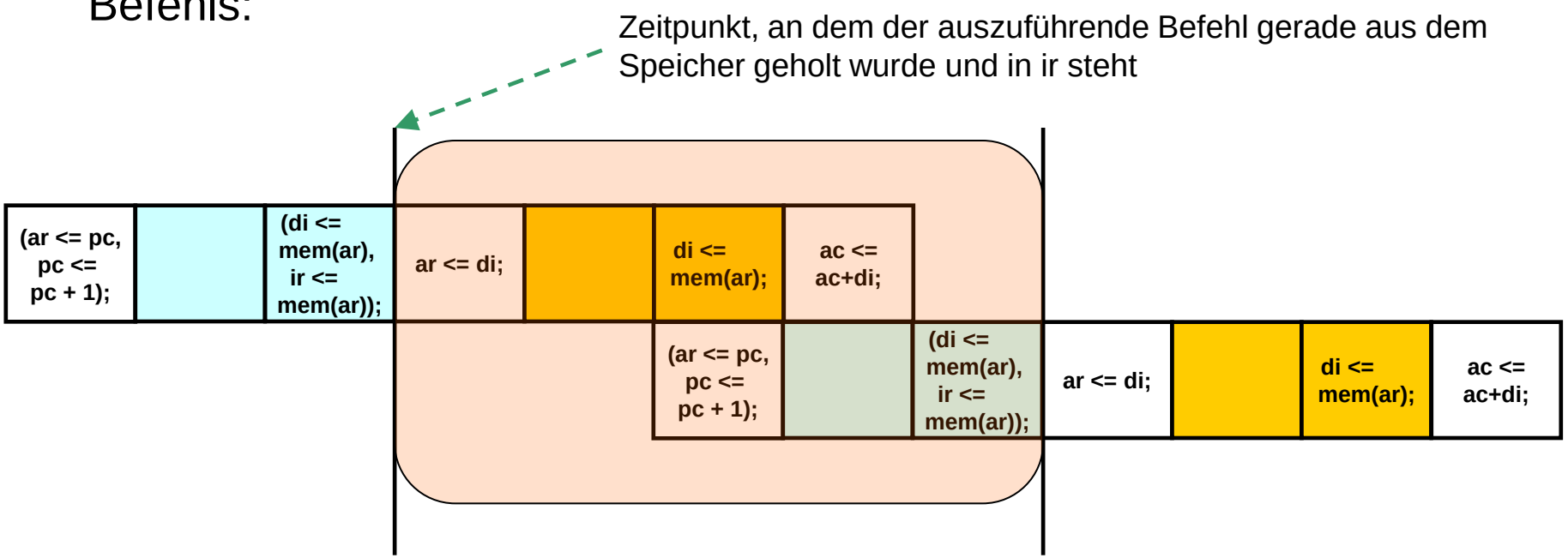
2.3.2.1 Exemplarische Befehlsausführung (X)

- Annahmen:
  - Hauptspeicherzugriffzeit = 2 * Prozessortaktzeit
  - Befehlsauslegung für größtmögliche Geschwindigkeit
  - Beispiel: Addiere-zu-Akkumulator-Befehl
  - Auslastung bei serieller Ausführung:
    - Hauptspeicher: 57%
    - CPU: 71%



2.3.2.1 Exemplarische Befehlsausführung (XI)

- Überlappung der Ausführungsphase mit dem Holen des nächsten Befehls:

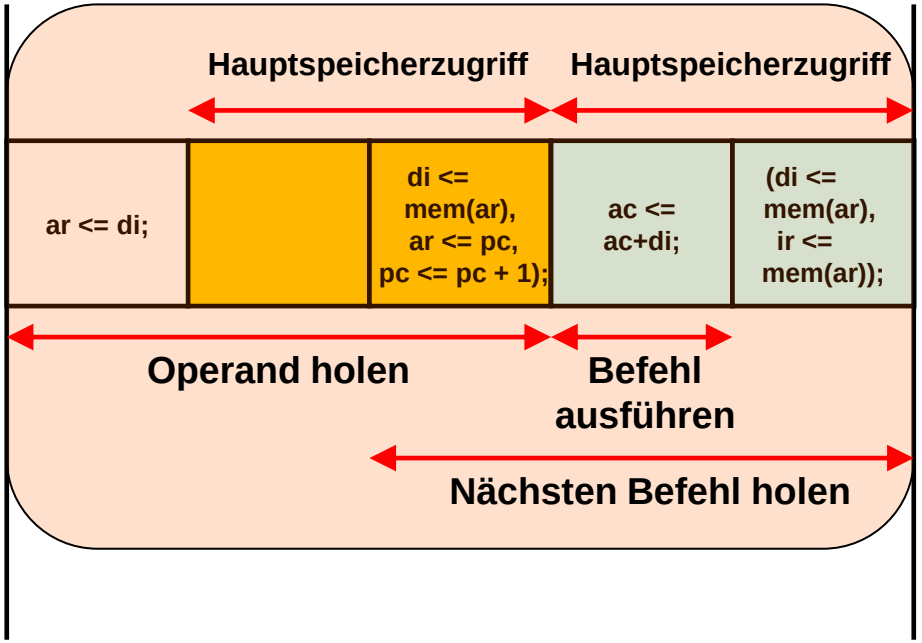


- Es werden keine zusätzlichen Ressourcen benötigt

- Eine zeitliche Einsparung von 2 Takten pro Befehl => 28,6 % Leistungssteigerung

2.3.2.1 Exemplarische Befehlsausführung (XII)

- Während eines Speicherzugriffs wird bereits der nächste Speicherzugriff vorbereitet
- Ebenfalls Parallel zum Speicherzugriff: Kalkulation des Ergebnisses
- Nur noch 5 Takte notwendig, keine zusätzlichen Kosten
- Auslastung bei überlappter Ausführung:
  - Hauptspeicher: 80%
  - CPU: 80%



2.3.2.1 Phasen der Befehlsausführung einer 5-stufigen Befehlspipeline (I)

- Befehlsbereitstellungsphase (Instruction Fetch):
  - Der Befehl, der durch den Befehlszähler adressiert ist, wird aus dem Hauptspeicher oder dem Cache-Speicher in einen Befehlspuffer geladen. Der Befehlszähler wird weitergeschaltet

- Dekodier- und Operandenbereitstellungsphase (Decode/Operand fetch):
  - Aus dem Operationscode des Maschinenbefehls werden prozessorinterne Steuersignale erzeugt (1. Takthälfte).
    Die Operanden werden aus Registern bereitgestellt (2. Takthälfte)

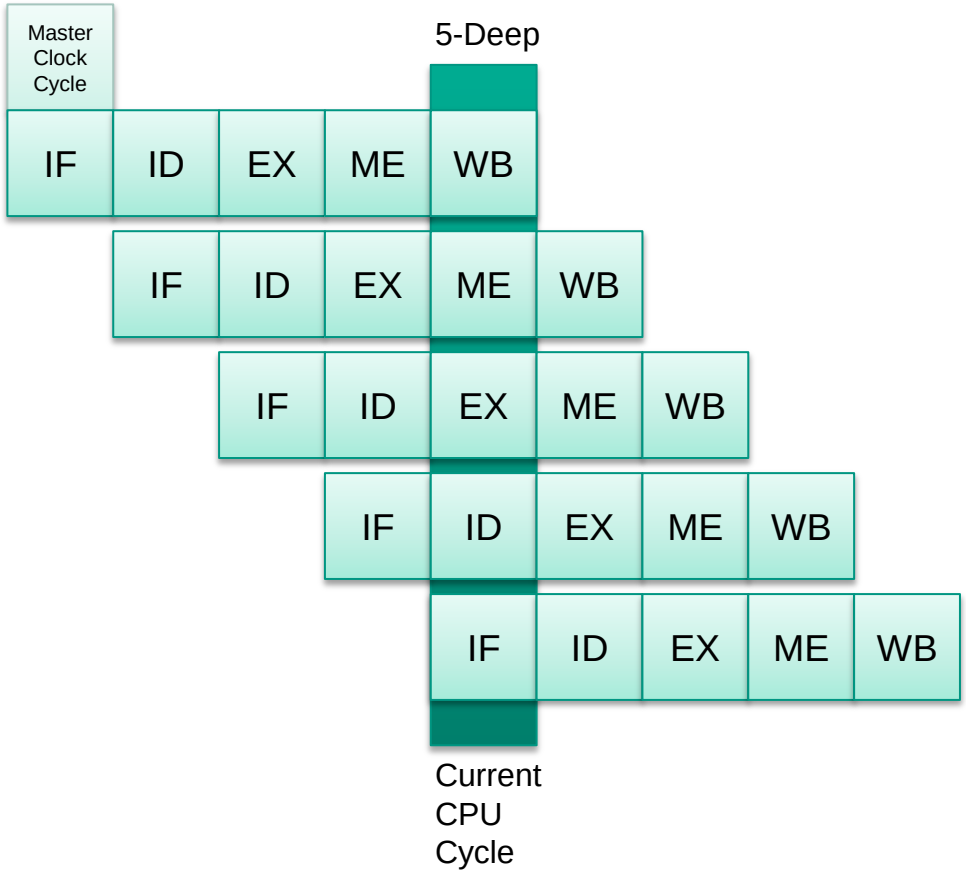
2.3.2.1 Phasen der Befehlsausführung einer 5-stufigen Befehlspipeline (II)

- Ausführungsphase (ALU Operation):
  - Die Operation wird auf den Operanden ausgeführt.
    Bei Lade-/Speicherbefehlen wird die effektive Adresse berechnet
- Speicherzugriffsphase (memory access):
  - Der Speicherzugriff[1] wird durchgeführt
- Resultatspeicherphase (Write Back):
  - Das Ergebnis wird in ein Register geschrieben (1. Takthälfte)

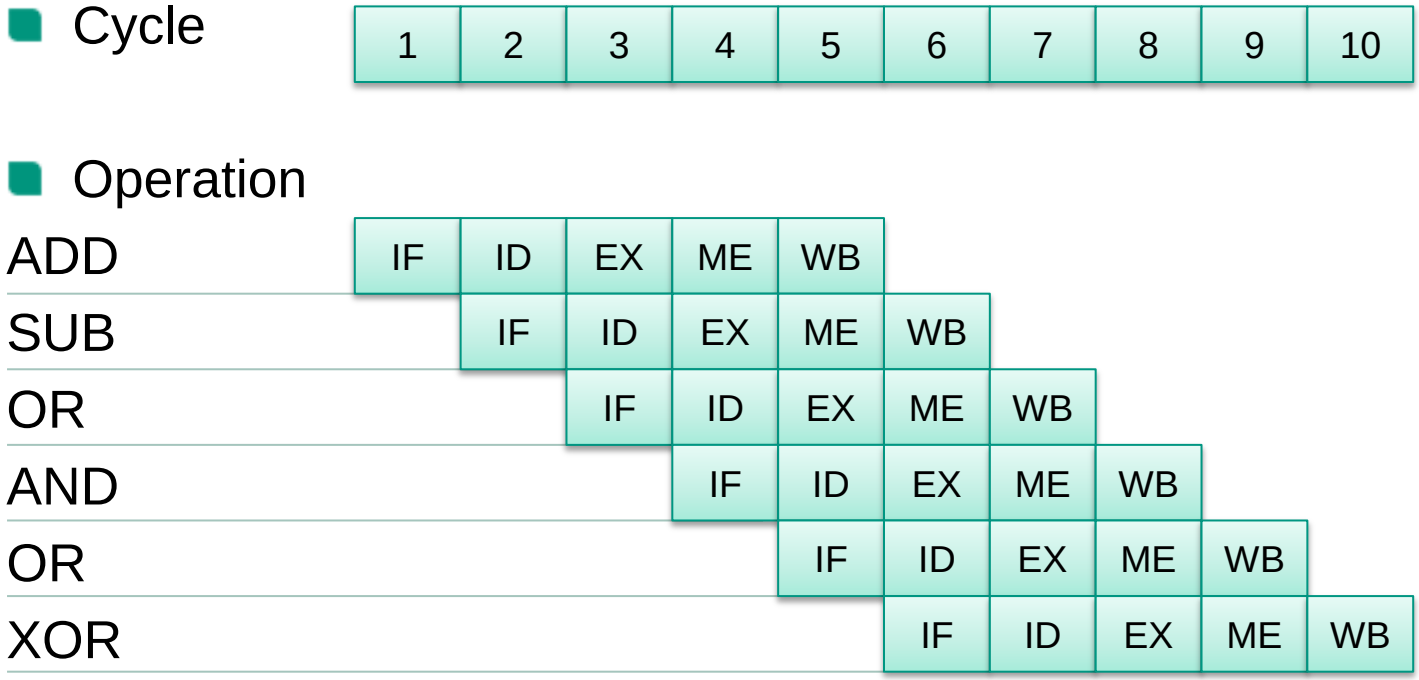
1) Anmerkung: z.B. Zugriff auf Cache oder Hauptspeicher, aber nicht (!) Register

2.3.2.1 Phasen der Befehlsausführung einer 5-stufigen Befehlspipeline (III)

- IF: Instruction Fetch
- ID: Instruction Decode
- EX: Execute
- MEM: Memory Access
- WB: Write Back



2.3.2.1 Beispiel: Optimales Pipelining



- Alle Operationen arbeiten auf Registern (single cycle execution)
- Zu jedem Taktzyklus wird eine Instruktion beendet
  - Erinnerung: Latenz bleibt identisch aber Durchsatz wird erhöht (5x)

2.3.2.1 Pipeline Konflikte (I)

- Erreicht man immer den n-fachen Durchsatz bei einer n-stufigen Pipeline? → NEIN, da es zu Pipelinekonflikten kommen kann.

D.H.:

- Die nächste Instruktion im Befehlsstrom im zugewiesenen Taktzyklus wird nicht ausgeführt
- Unterbrechung des taktsynchronen Durchlaufs durch die einzelnen Stufen der Pipeline
- Verursacht Leistungseinbußen, da Durchsatz reduziert wird

2.3.2.1 Pipeline Konflikte (II)

- Strukturkonflikte
  - Ergeben sich aus Ressourcenkonflikten
    - Die Hardware kann nicht alle mögliche Kombinationen von Befehlen unterstützen, die sich in der Pipeline befinden können
  - Beispiel: Gleichzeitiger Schreibzugriff zweier Befehle auf eine Registerdatei mit nur einem Schreibeingang

- Datenkonflikte
  - Ergeben sich aus Datenabhängigkeiten zwischen Befehlen im Programm
  - Instruktion benötigt das Ergebnis einer vorangehenden und noch nicht abgeschlossenen Instruktion in der Pipeline
  - D.h. ein Operand ist noch nicht verfügbar
    z.B. Echte Datenabhängigkeit, Namensabhängigkeiten

- Steuerkonflikte
  - Treten bei Verzweigungsbefehlen und anderen Instruktionen auf, die den Befehlszähler verändern

2.3.2.1 Auflösung von Pipeline Konflikten

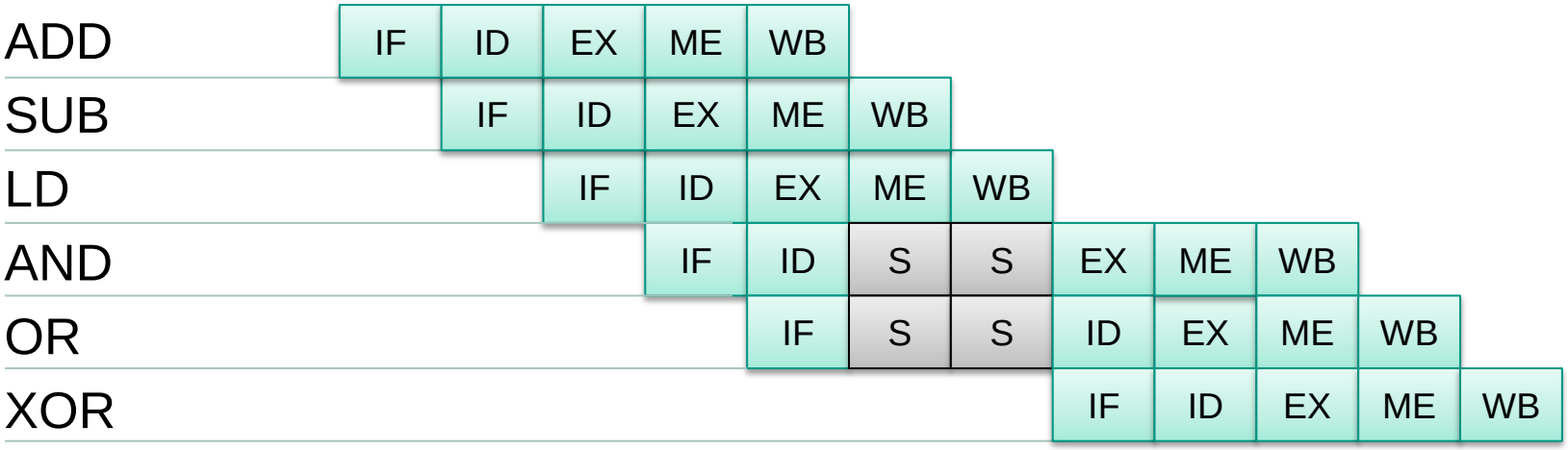
- Softwarebasierend:
  - Aufgabe des Compilers:
    - Erkennen von Datenkonflikten
    - Einfügen von Leeroperationen nach jedem Befehl, der einen Konflikt verursacht (NOP)
  - Statische Verfahren:
    - Instruction Scheduling, Pipeline Scheduling
    - Eliminieren von Leeroperationen
    - Umordnen der Befehle des Programms (Code-Optimierung)

- Hardware-Lösungen (Dynamische Verfahren)
  - Erkennen von Konflikten
    - Entsprechende Konflikterkennungslogik notwendig!
    - Techniken
      - Leerlauf der Pipeline (Stalling)
      - Forwarding

2.3.2.1 Beispiel: Load Pipeline

- Cycle

- Operation



- Stalling für zwei Taktzyklen
- 6 Taktzyklen für 4 Instruktionen
  - Clock cycles per Instruction (CPI) = 1,5

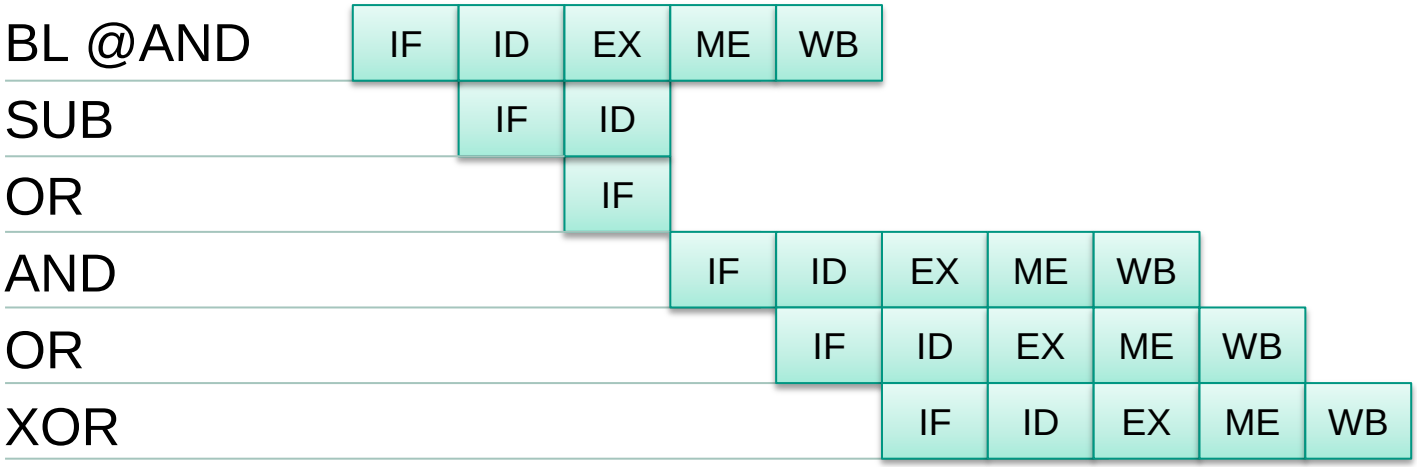
S Stall

Oliver Sander
Hardware/Software Co-Design

2.3.2.1 Beispiel: Branch and Link Pipeline

- Cycle

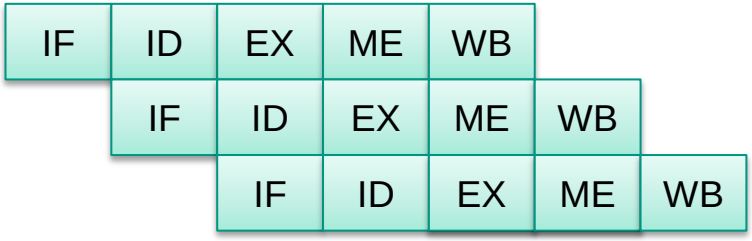
- Operation



- Aufbrechen der Pipeline

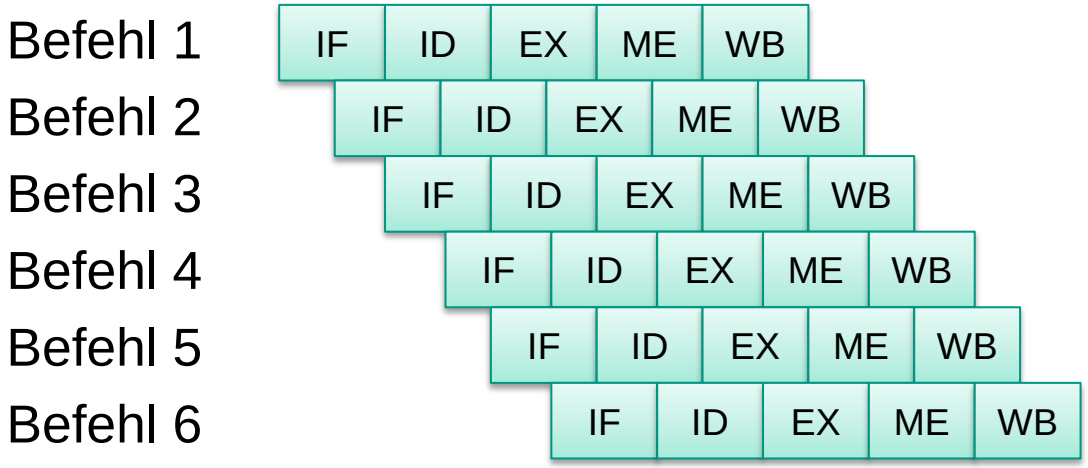
2.3.2.1 Beispiel: Pipeline - Datenkonflikte

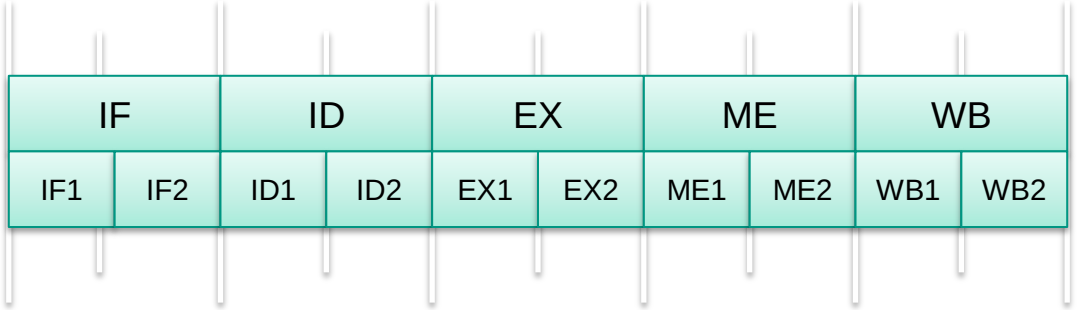
- Datenkonflikte
  - Datenabhängigkeiten (zwischen Befehlen im Programm)
  - Beispiel:
    - ADD R1, R2, R3
    - AND R6, R1, R8
    - XOR R9, R1, R11



S Stall

2.3.2.1 Superpipelining (I)

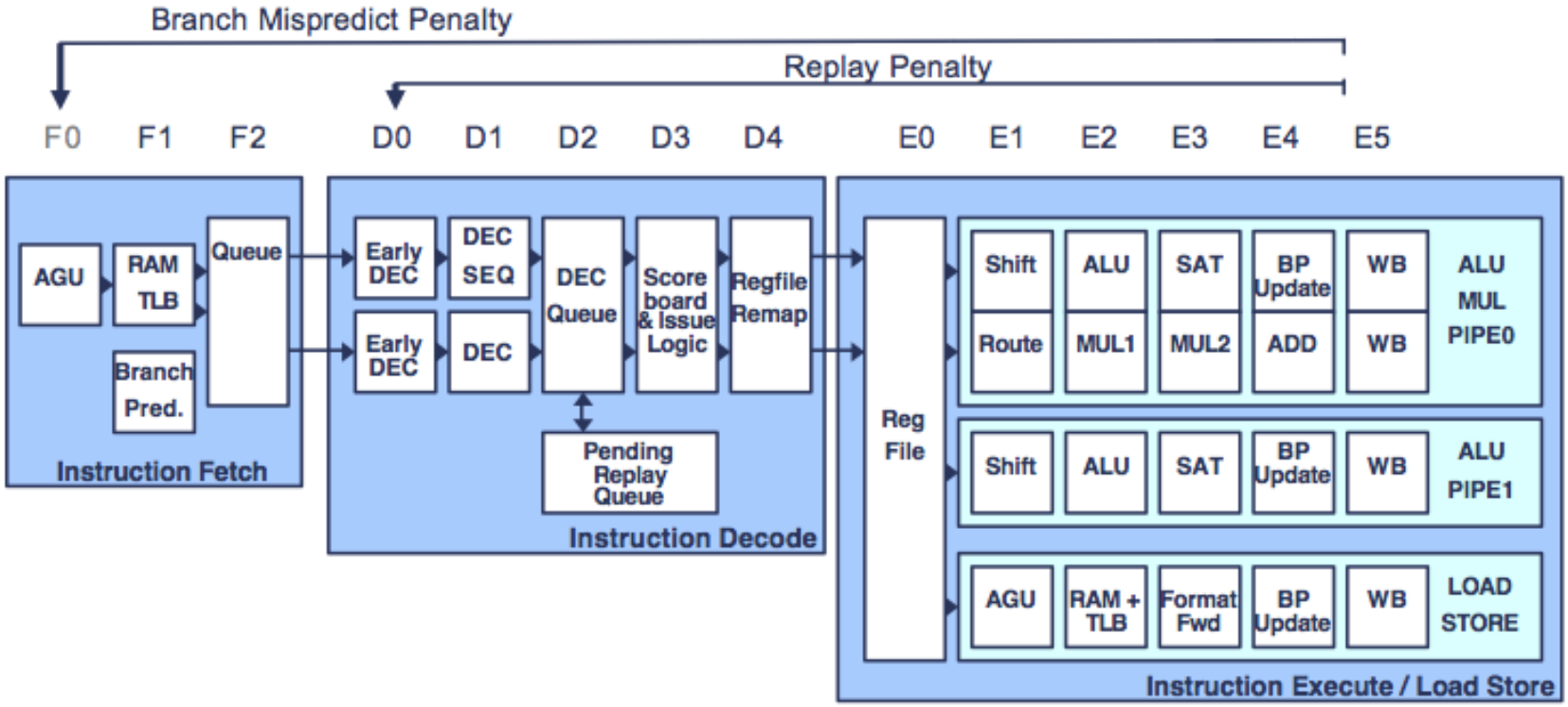




Maschinenzyklus
½ Grundtakt der
RISC-Maschine

2.3.2.1 Superpipelining (II)

- ARM Cortex A8 integer pipeline



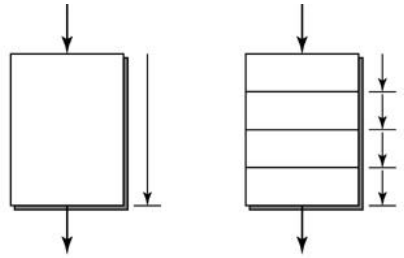
- Optimierung von Code zur Ausnutzung der Pipeline sehr schwierig
  - Compiler-Aufgabe!

- Was ist Pipelining?
- Wie ist die 5-stufige DLX Pipeline aufgebaut?
- Welche Latenz hat die Pipeline bei einem Takt pro Stufe?
- Was ist der ideale Durchsatz des Prozessors? Kann er erreicht werden?
- Welche Konflikte können auftreten?
- Wie können diese gelöst werden?
- Warum gibt es keine 50-stufigen Pipelines?

Inhalt

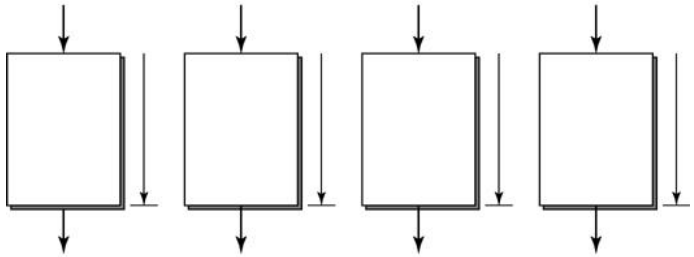
2.3.2.2 Superskalarität (I)

- a Keine Parallelität (Nonpipelined)
- b Zeitliche Parallelität (Pipelined)



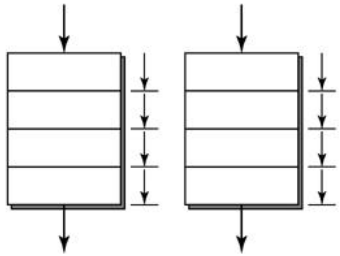
(a) No parallelism (b) Temporal parallelism

- c Räumliche Parallelität (Multiple units)



(c) Spatial parallelism

- d Zeitliche und Räumliche Parallelität

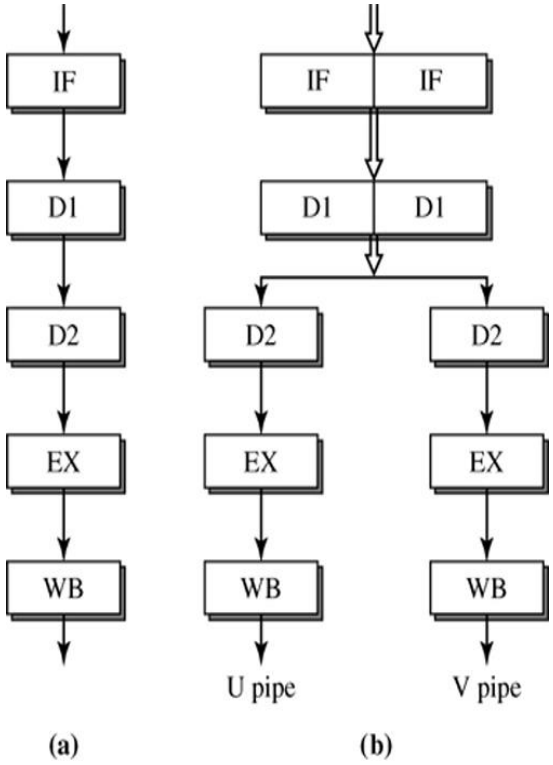


(d) Parallel pipeline

Parts from: Prof. Lizy Kurian John Univ. Austin, Texas

2.3.2.2 Superskalarität (II)

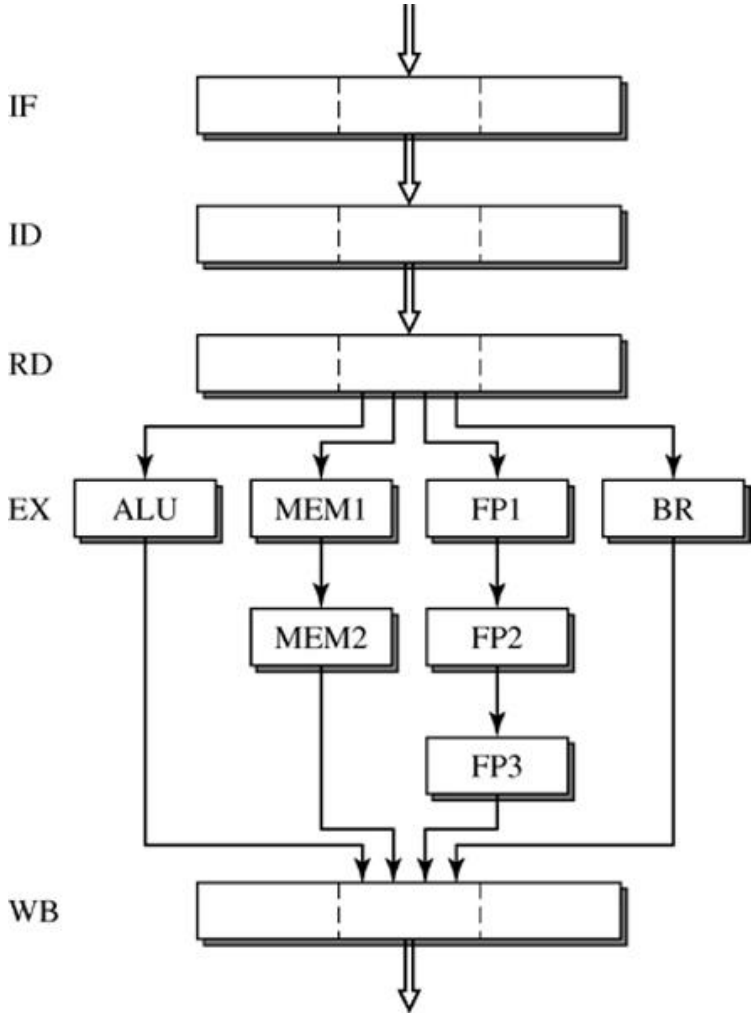
- Fetch
  - Load 16-bytes of instruction into prefetch buffer
- Decode1
  - Determine instruction length, instruction type
- Decode2
  - Compute memory address
  - Generate immediate operands
- Execute
  - Register Read
  - ALU operation
  - Memory read/write
- Write-Back
  - Update register file



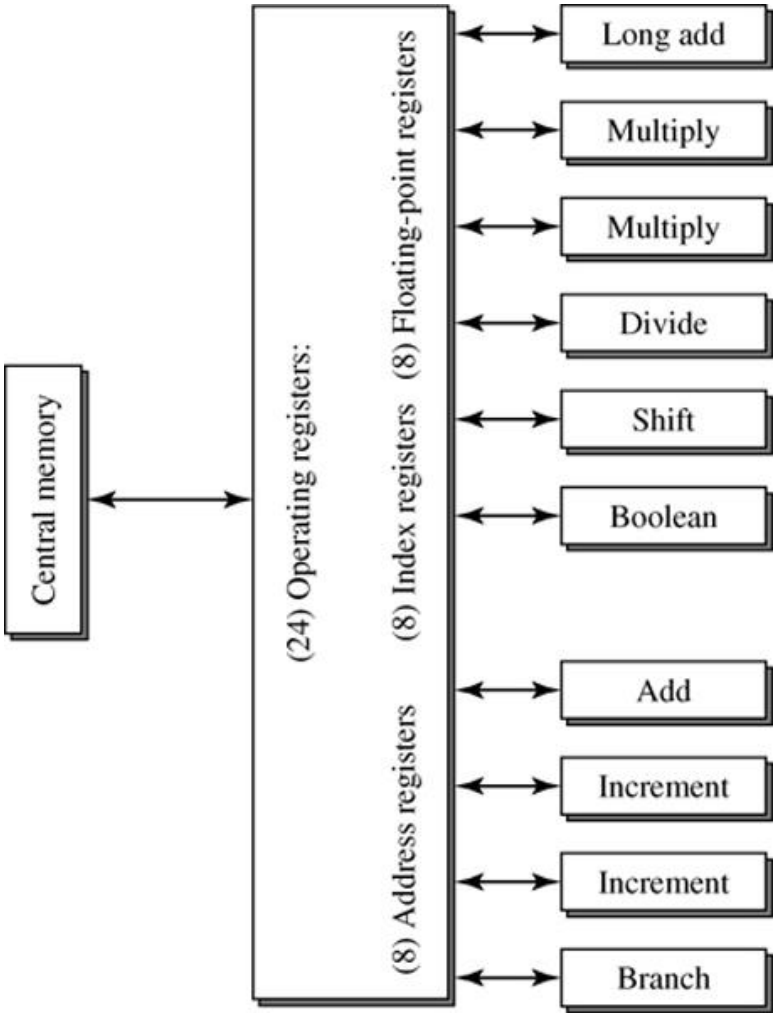
5-stufige i486 skalare Pipeline

5-stufige Pentium parallele Pipeline

2.3.2.2 Superskalarität (III)



Parts from: Prof. Lizy Kurian John Univ. Austin, Texas



CDC 6600 mit 10 verschiedenartigen funktionalen Einheiten

2.3.2.2 Superskalarität – Begiffe (I)

- Superpipeline
  - Tiefe Pipeline, aber keine superskalaren Einheiten
- Superskalar
  - Kann mehr als eine Instruktion/Zyklus ausführen
- Out-of-Order
  - Kann Instruktionen außerhalb der Programmreihenfolge ausführen
- Spekulation
  - Führt Instruktionen auch nach Branches weiter aus, die später eventuell annulliert werden müssen

2.3.2.2 Superskalarität – Begiffe (II)

- Mehrere Funktionseinheiten, parallel arbeitend
- Den Ausführungseinheiten kann mehr als ein Befehl pro Takt zugewiesen werden
- Die Befehle werden aus einem sequenziellen Strom von normalen Befehlen zugewiesen
- Die Zuweisung der Befehle erfolgt in Hardware durch einen dynamischen Scheduler (-> Dispatcher)
- Die Anzahl der zugewiesenen Befehle pro Takt wird dynamisch von der Hardware bestimmt und liegt zwischen null und der maximal möglichen Zuweisungsbandbreite

2.3.2.2 Superskalarität – Begiffe (III)

- Die dynamische Zuweisung von Befehlen führt zu einem komplexen Hardware-Scheduler.
  - Die Komplexität des Schedulers steigt mit der Größe des Befehlsfensters und mit der Anzahl der Befehle, die außerhalb der Programmreihenfolge zugewiesen werden können.

- Mehrere Ausführungseinheiten müssen verfügbar sein
  - Anzahl der Ausführungseinheiten entspricht mindestens der Zuweisungsbandbreite,
  - Häufig gibt es jedoch noch mehr Ausführungseinheiten, um potenzielle Strukturkonflikte zu umgehen.

2.3.2.2 Superskalarität – Begiffe (IV)

- Superskalartechnik ist eine Mikroarchitekturtechnik und hat keinen Einfluss auf die Befehlssatz-Architektur.

- Der Begriff „superskalar“ wird oft in unpräziser Weise benutzt, um Prozessoren mit mehreren parallelen Pipelines oder mehreren Ausführungseinheiten zu beschreiben. Gilt jedoch nur dann, wenn die Zuweisung der Instruktionen dynamisch erfolgt
  - In dieser Weise kann man nicht zwischen der Superskalar- und der VLIW-Technik zu unterscheiden.

2.3.2.2 Superskalarität – Begiffe (V)

- Befehls-Pipelining und Superskalartechnik nutzen beide feinkörnige Parallelität (fine-grain oder instruction-level parallelism), d.h. Parallelität zwischen einzelnen Befehlen.
  - Pipelining nutzt zeitliche Parallelität (temporal parallelism)
  - Superskalar nutzt räumliche Parallelität (spatial parallelism).

- Eine Leistungssteigerung durch zeitliche Parallelität kann mit einer längeren Pipeline und höherer Taktfrequenz erreicht werden.

- Falls genügend feinkörnige Parallelität vorhanden ist, kann die Leistung durch räumliche Parallelität im superskalaren Fall mit Hilfe von mehr Ausführungseinheiten und einer hohen dynamischen Zuweisungsrate erreicht werden.

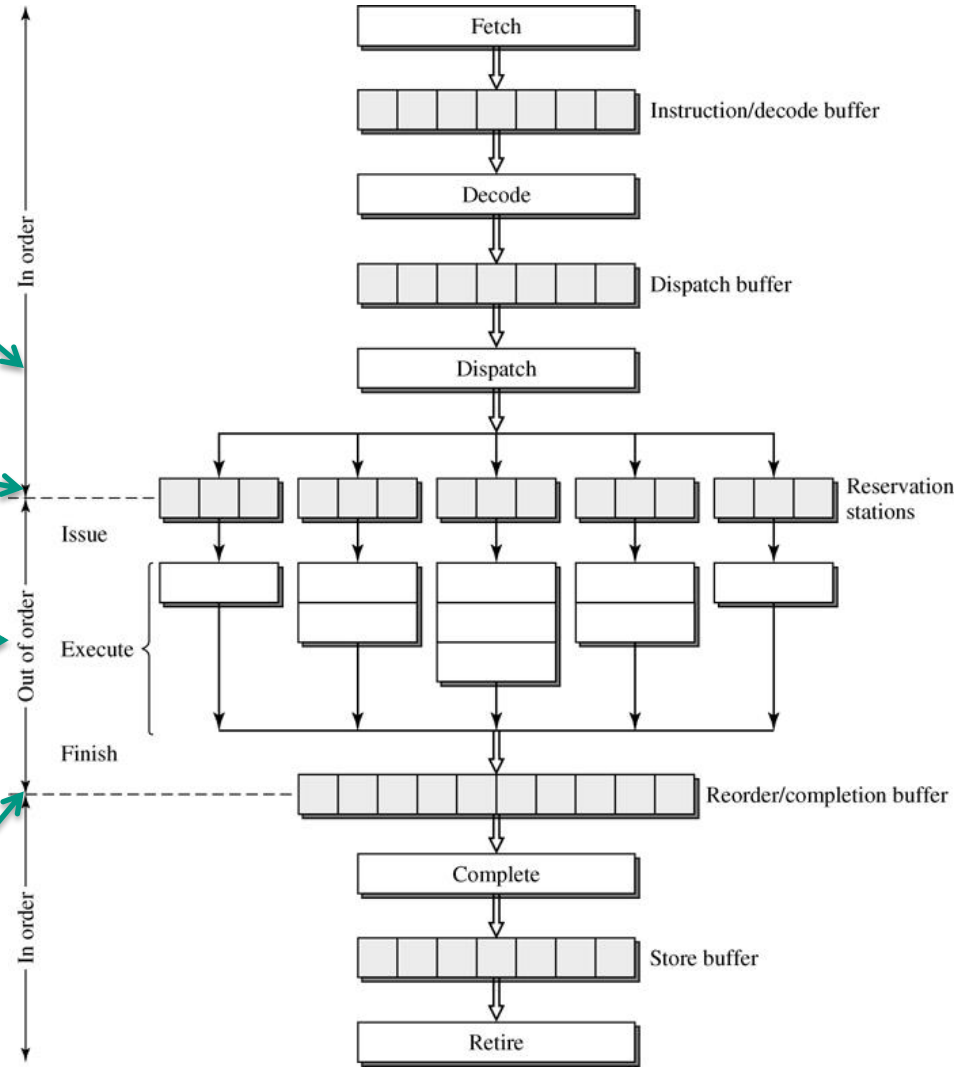
2.3.2.2 Superskalarität – Begiffe (VI)

- Zwei Befehle können parallel ausgeführt werden, wenn:
  - Beide einfache Instruktionen sind
    - z.B. keine speziellen Einheiten einer bestimmten Pipeline benötigt werden
  - Instruktion 1 kein Sprung ist
  - Wenn das Ziel von Instruktion 1 nicht die Quelle von Instruktion 2 ist
  - Wenn das Ziel von Instruktion 1 nicht das Ziel von Instruktion 2 ist

- Wenn zwei Befehle nicht parallel ausgeführt werden können:
  - Instruktion 1 in erster Pipeline abarbeiten
  - Instruktion 2 im nächsten Zyklus abarbeiten
    - Wenn möglich, zusammen mit der darauf folgenden Instruktion

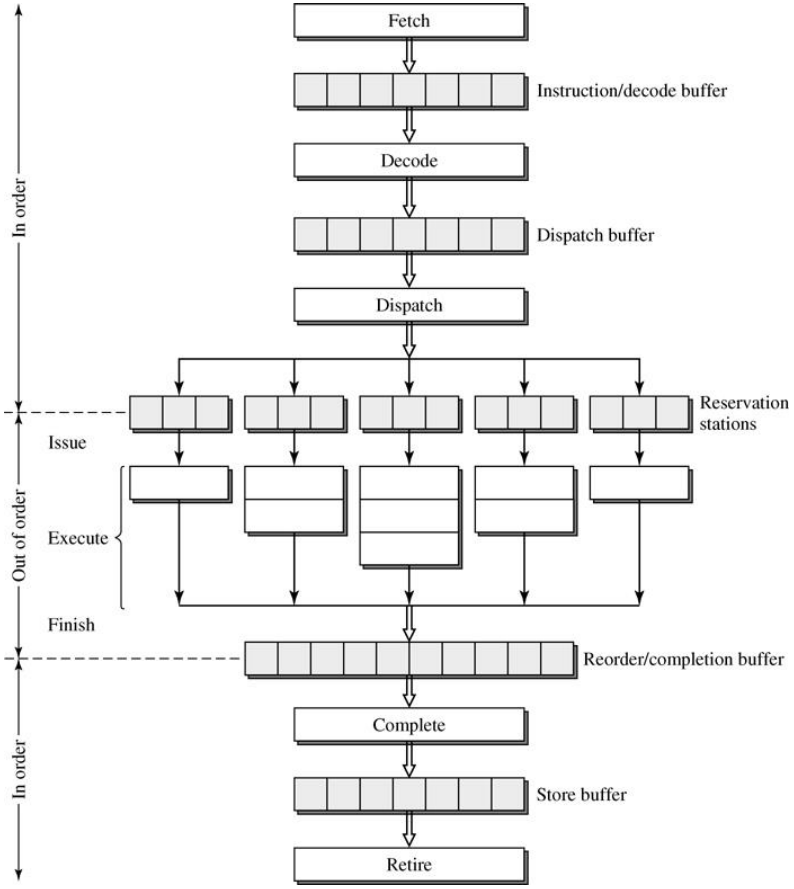
2.3.2.2 Dynamisches Scheduling - Prozessorpipeline (I)

- Prinzipielle Vorgehensweise:
  - Zuweisen von Befehlen an Ausführungseinheiten
    - In-order Issue
    - Out-of-Order Issue
  - Reservation Stations
  - Weiterleiten der Befehle an Funktionseinheiten, wenn Operanden verfügbar sind
  - Ablegen der Befehle in Programmreihenfolge in dem Reorder-Puffer



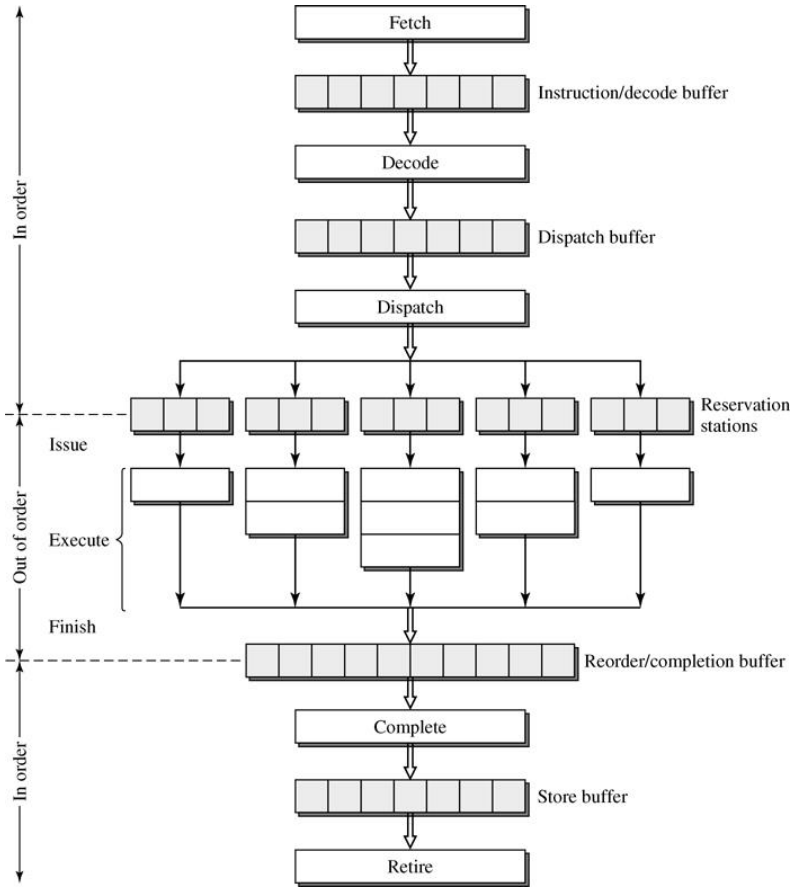
2.3.2.2 Dynamisches Scheduling - Prozessorpipeline (II)

- IF Phase
  - Holen mehrerer Befehle in den Hole-Puffer
  - Verwaltung der Komponenten für die Sprungzielvorhersage
- ID Phase
  - Dekodierung der Befehle
  - Umbenennung von Registern
  - Abbildung von logischen Registern auf physikalische Prozessorregister
  - Schreiben der Befehle ins Befehlsregister
  - Erkennen und Auflösen von Konflikten aufgrund von Daten- und Strukturabhängigkeiten



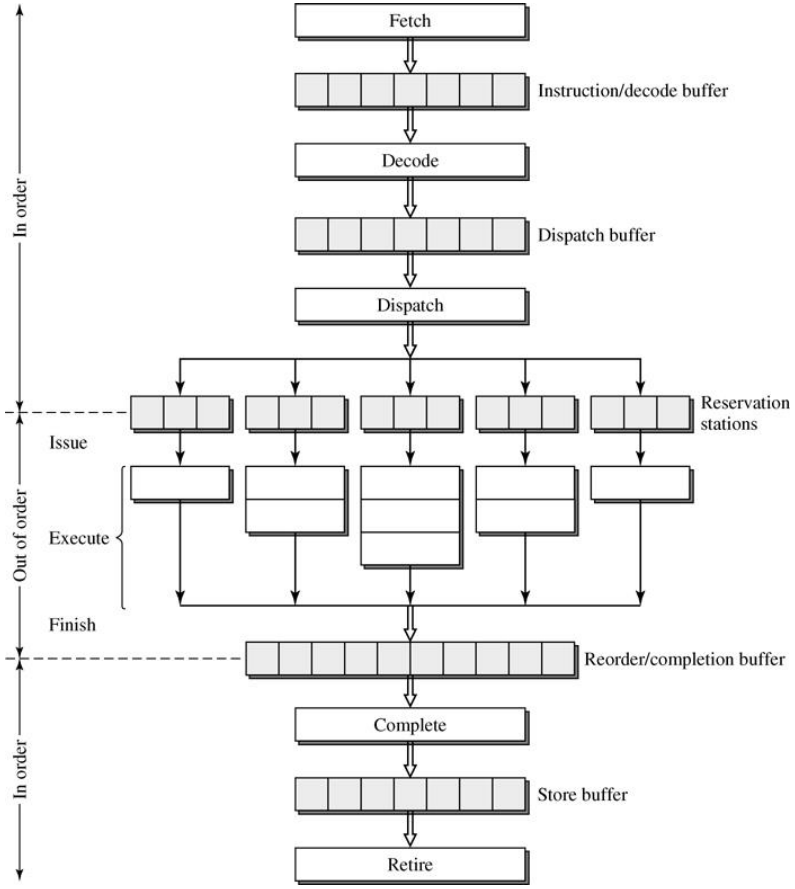
2.3.2.2 Dynamisches Scheduling – Dispatcher

- Eine Instruktion wird aus der Reservation Station an die Funktionseinheit weitergegeben, wenn alle Operanden verfügbar sind
- Der Befehl wird in der Funktionseinheit ausgeführt
- Wenn alle Operanden verfügbar sind und die Funktionseinheit nicht besetzt ist, wird die Instruktion sofort zur Ausführung angestoßen
- Eine Instruktion kann sich null oder mehrere Zyklen in der Reservation Station aufhalten
- Dispatch und Ausführen erfolgt nicht in der sequentiellen Programmordnung



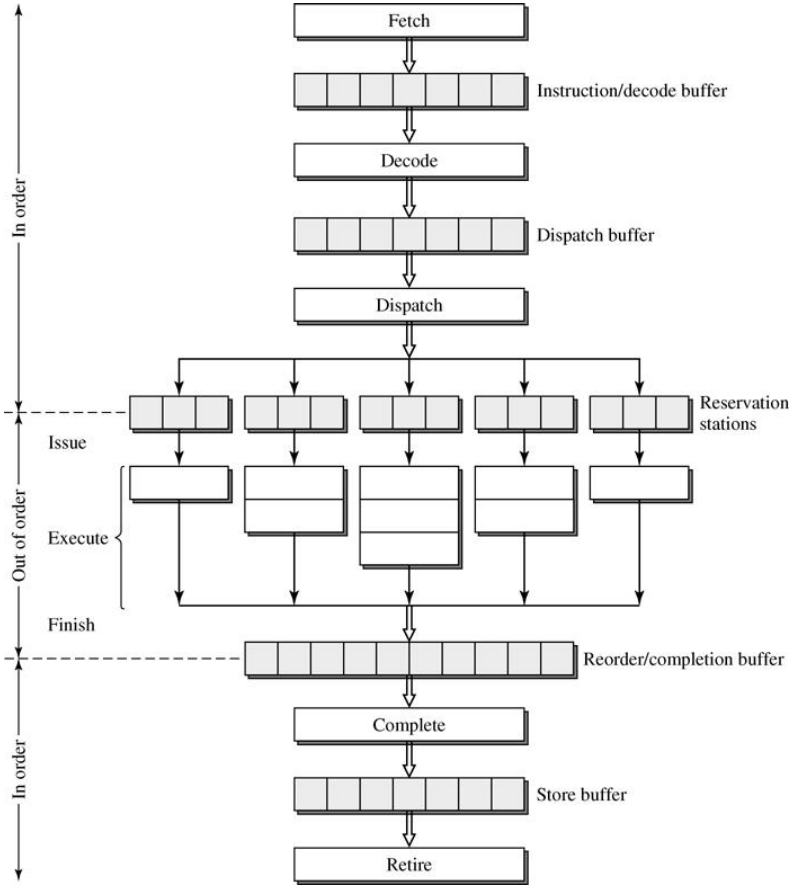
2.3.2.2 Dynamisches Scheduling - Reservation Stations

- Puffer für eine Instruktion mit ihrem Operanden
  - (siehe Tomasulo-Algorithmus, Hennessy/Patterson Buch)
- Puffer mit ein oder mehreren Einträgen. Jeder Eintrag kann eine Instruktion mit ihrem Operanden enthalten
- Konfliktauflösung mit Hilfe von Reservation Stations
- Befehle warten in Reservation Station bis Operand verfügbar ist



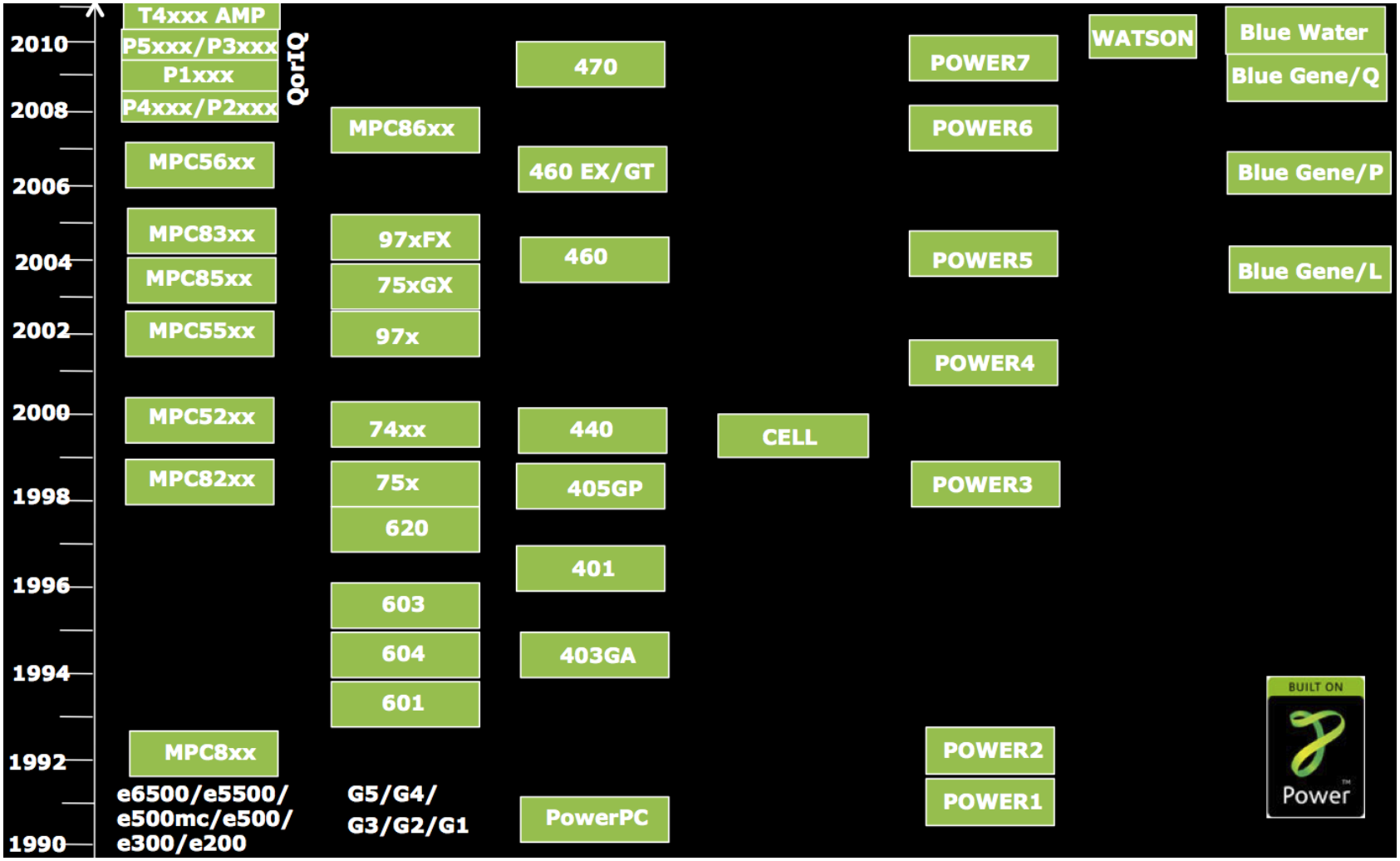
2.3.2.2 Dynamisches Scheduling - Completion-Unit

- Eine Instruktion beendet ihre Ausführung, wenn das Ergebnis für nachfolgende Befehle bereit steht (Forwarding-Puffer)
- Completion: Befehl ist vollständig
- Erfolgt unabhängig von der Programmordnung
- Aktualisierung des Zustands des Rückordnungspuffers (Reorder-Puffer)
  - Es kann eine Unterbrechung angezeigt sein
  - Es kann ein unvollständiger Befehl angezeigt werden, der von einer Spekulation abhängt

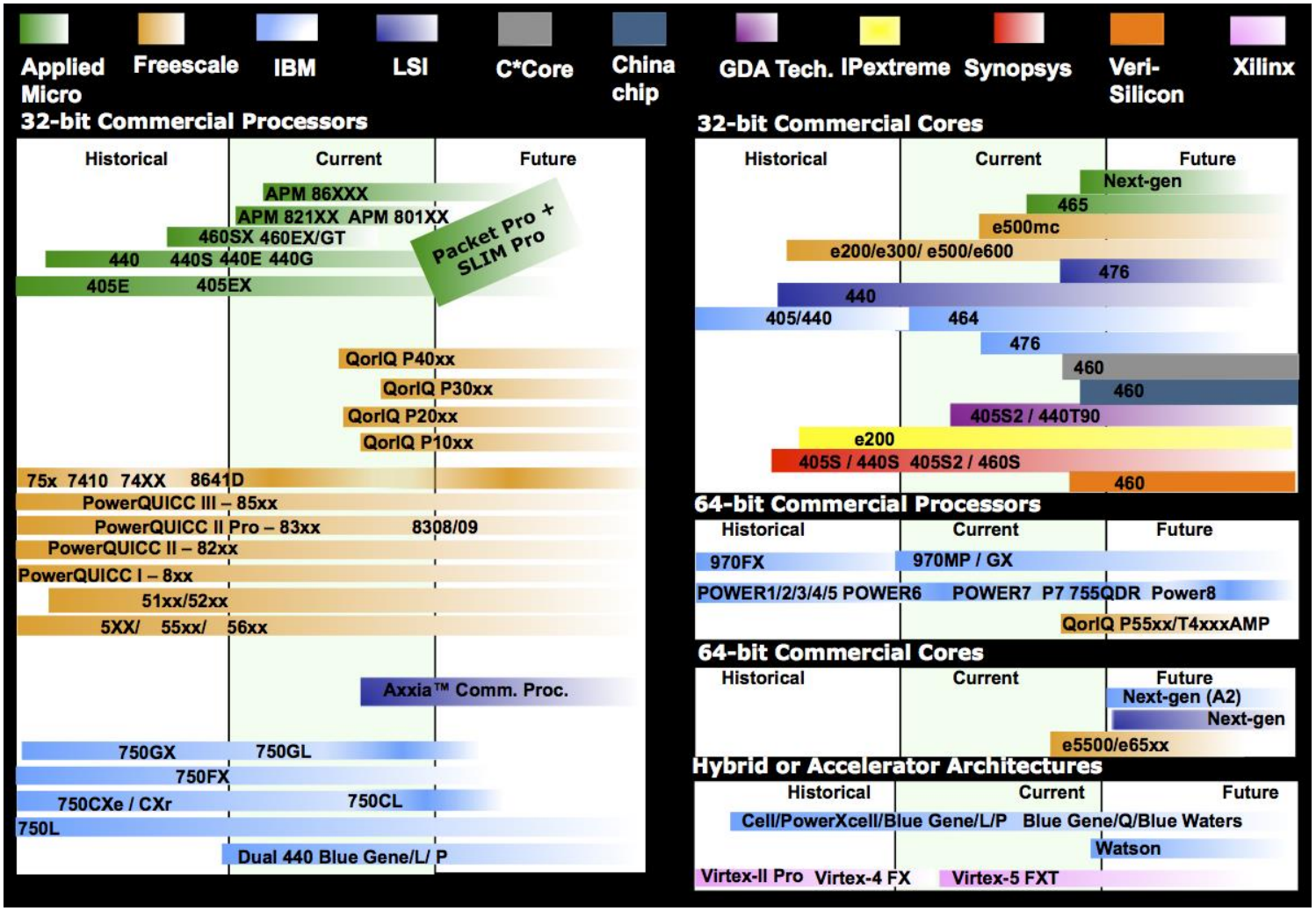


Oliver Sander
Hardware/Software Co-Design

2.3.2.2 Beispiel – Power Architektur



2.3.2.2 Beispiel – Power Architektur Roadmap

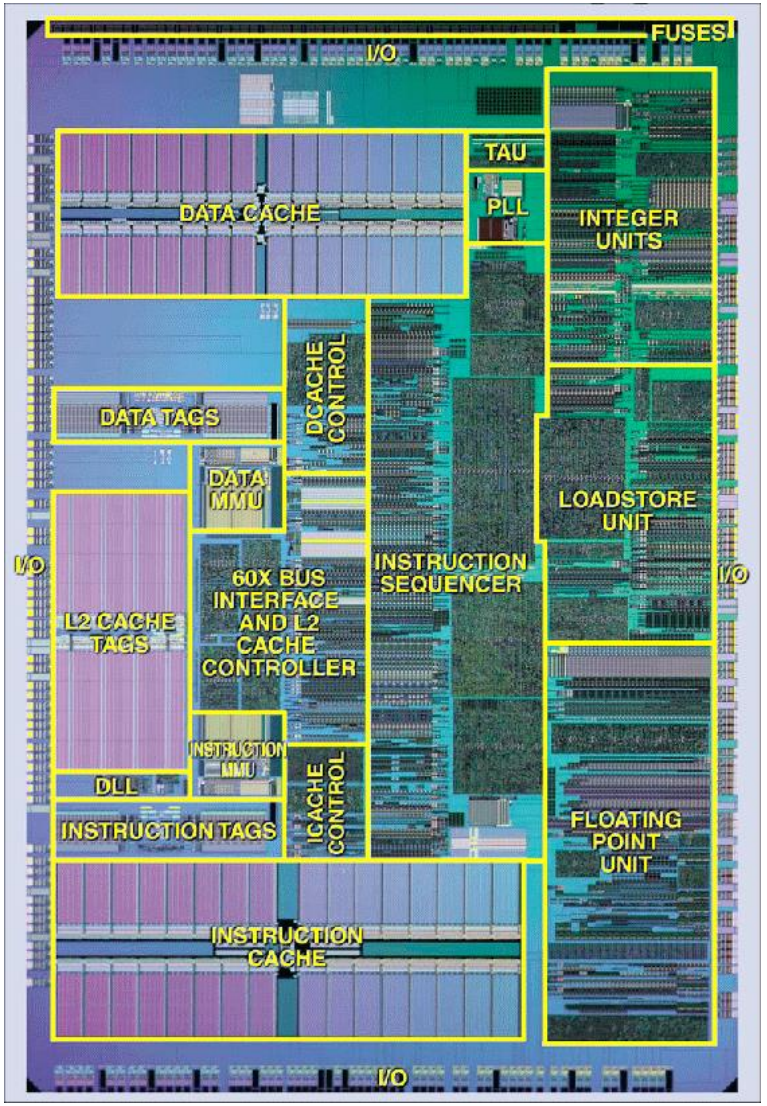


2.3.2.2 Beispiel – Power7

- 45 nm SOI process, 567 mm2
- 1.2 billion transistors
- 3.0 – 4.25 GHz clock speed
- 32+32 kB L1 instruction and data cache (per core)[13]
- 256 kB L2 Cache (per core)
- 4 MB L3 cache per core with maximum up to 32MB supported.
- max 4 chips per quad-chip module
  - 4, 6 or 8 cores per chip
    - 4 SMT threads per core
    - 12 execution units per core:
      - 2 fixed-point units
      - 2 load/store units
      - 4 double-precision floating-point units
      - 1 vector unit supporting VSX
      - 1 decimal floating-point unit
      - 1 branch unit
      - 1 condition register unit

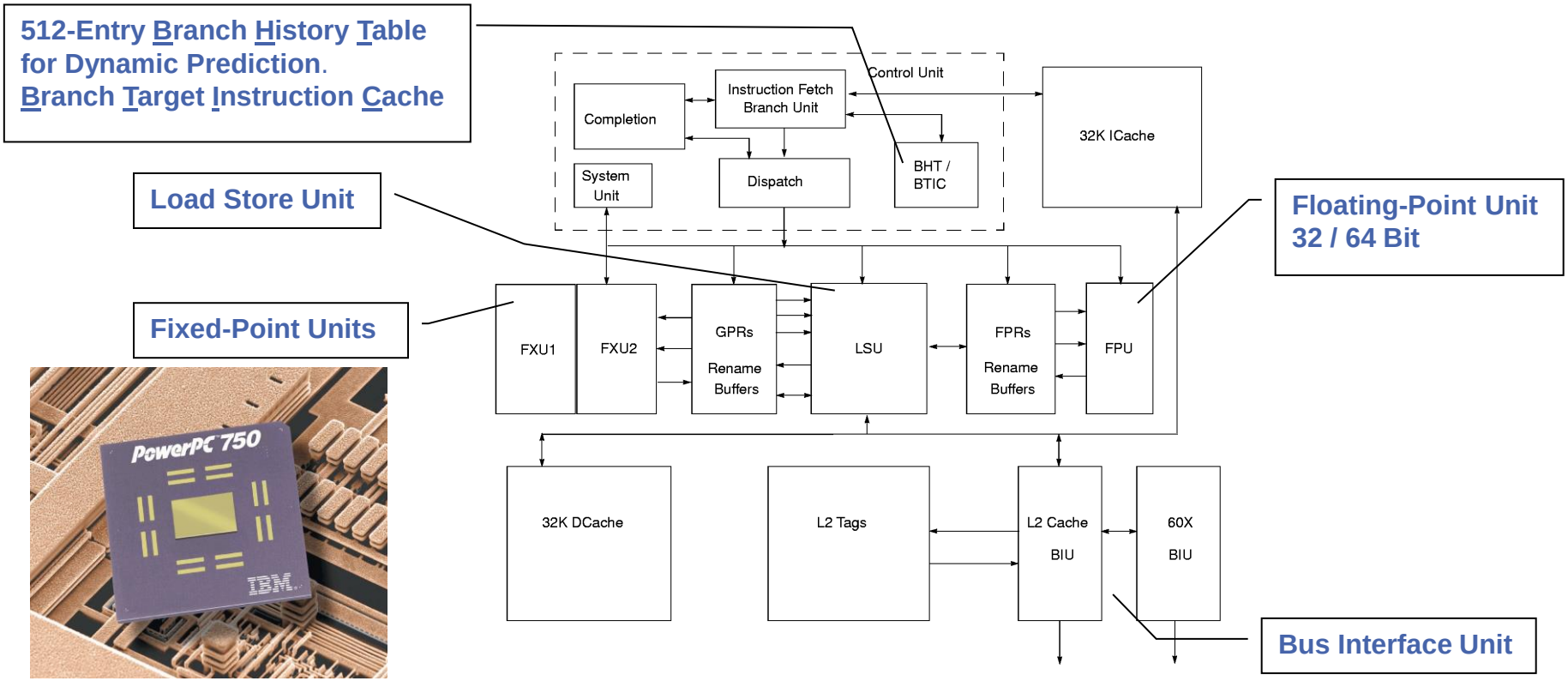
2.3.2.2 Beispiel – PowerPC 750

- Hersteller IBM
- CMOS 0.20 µm
- Kupfer Technologie PID-8p
- 6 Metall- Lagen
- Die Size: 5.14mm x 7.78mm (40mm2)
- 6.35 Millionen Transistoren
- Logic Design: vollst. statisch
- Frequenz: 300-500 MHz
- Core- Spannung: 2 V
- I/O Spannung: 3.3V 2.5V, oder 1.8V
- Package: 25x25mm, 360 Blei-Keramik Ball Grid Array
- Leistung: 4.7 W bei 400 MHz

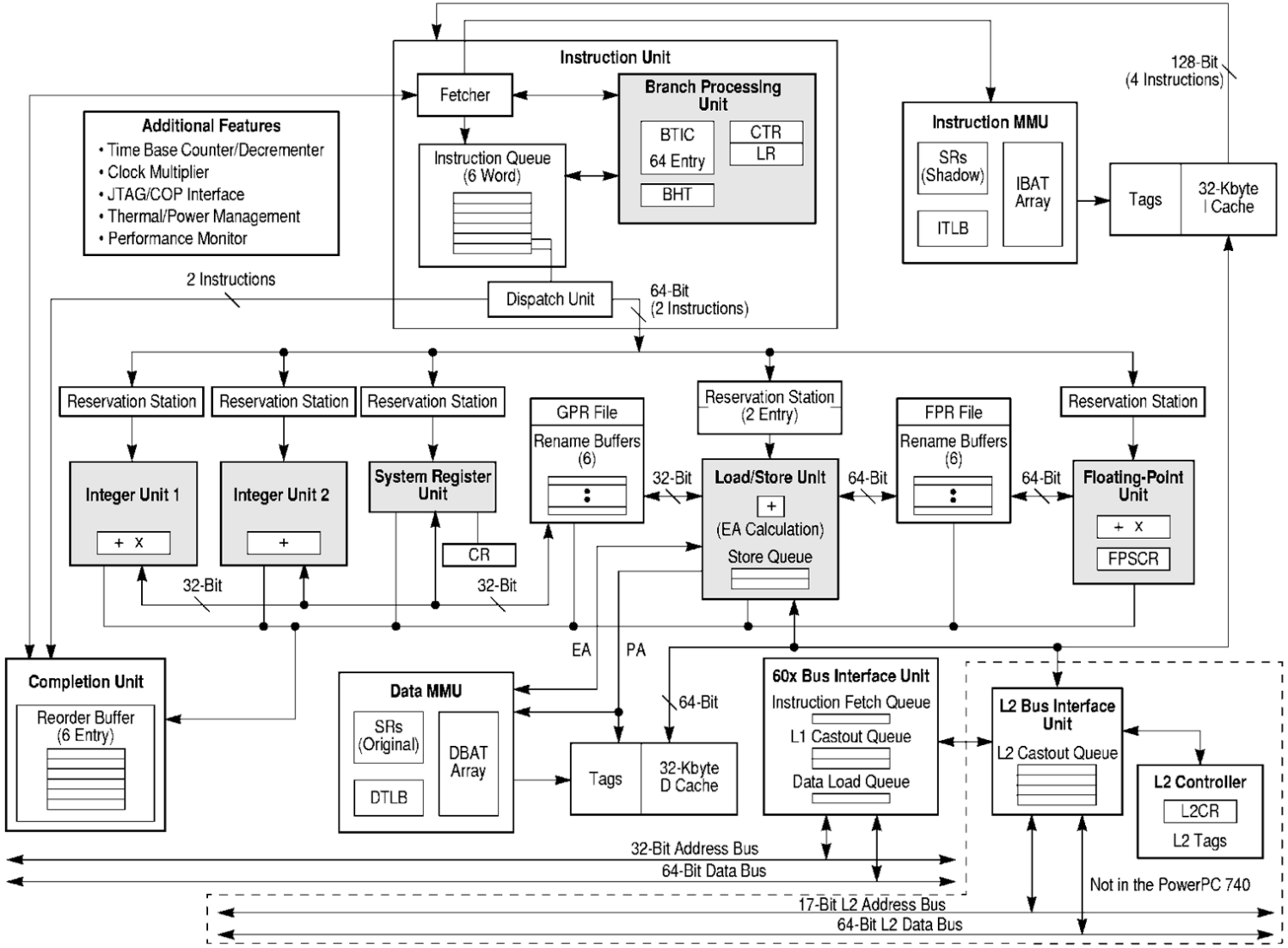


2.3.2.2 Beispiel – PowerPC 750

- Harvard- Architektur : getrennten Daten- / Instruktionsspeicher
- Superskalar Out-of-order-Execution
- Spekulative Ausführung einer Instruktionssequenz



2.3.2.2 Beispiel – PowerPC 750



2.3.2.2 Beispiel – PowerPC 750

- Power Management Unit
  - Statische Verlustleistung Sleep-Modus
  - Dynamische Verlustleistung Abschalten interner Einheiten (clock gating)
  - Integrierte Temperaturregelung Sensor + Kontrolle

- Level 2 (L2) Cache Interface
  - Interner L2-Cache Controller und 4-K entry tags
  - Unterstützt 256-Kbyte, 512-Kbyte, sowie 1-Mbyte assoziativer L2-Cache
  - Anbindung an bis zu 233 MHz externe SRAMs
  - Parity-Check im Bus-Interface

2.3.2.2 Beispiel – PowerPC 750

- Instruction Fetching & Branch Unit
  - 4 Instruktionen werden pro Taktzyklus geladen
  - 64-entry BTIC (Branch Target Instruction Cache)
  - 512-entry BHT (Branch History Table)
- Load/Store Unit (LSU)
  - alle load/store Instruktionen zwischen GPRs, FPRs und Cache/Hauptspeicher
  - effektive (logische) Adressberechnung
- Dispatch Unit
  - Hardwaredetektion von Instruktionstyp/Abhängigkeiten
  - weist 2 Instruktionen/Zyklus an 6 unabhängige Einheiten
  - 4-stufige Pipeline: fetch, dispatch, execute und complete

2.3.2.2 Beispiel – PowerPC 750

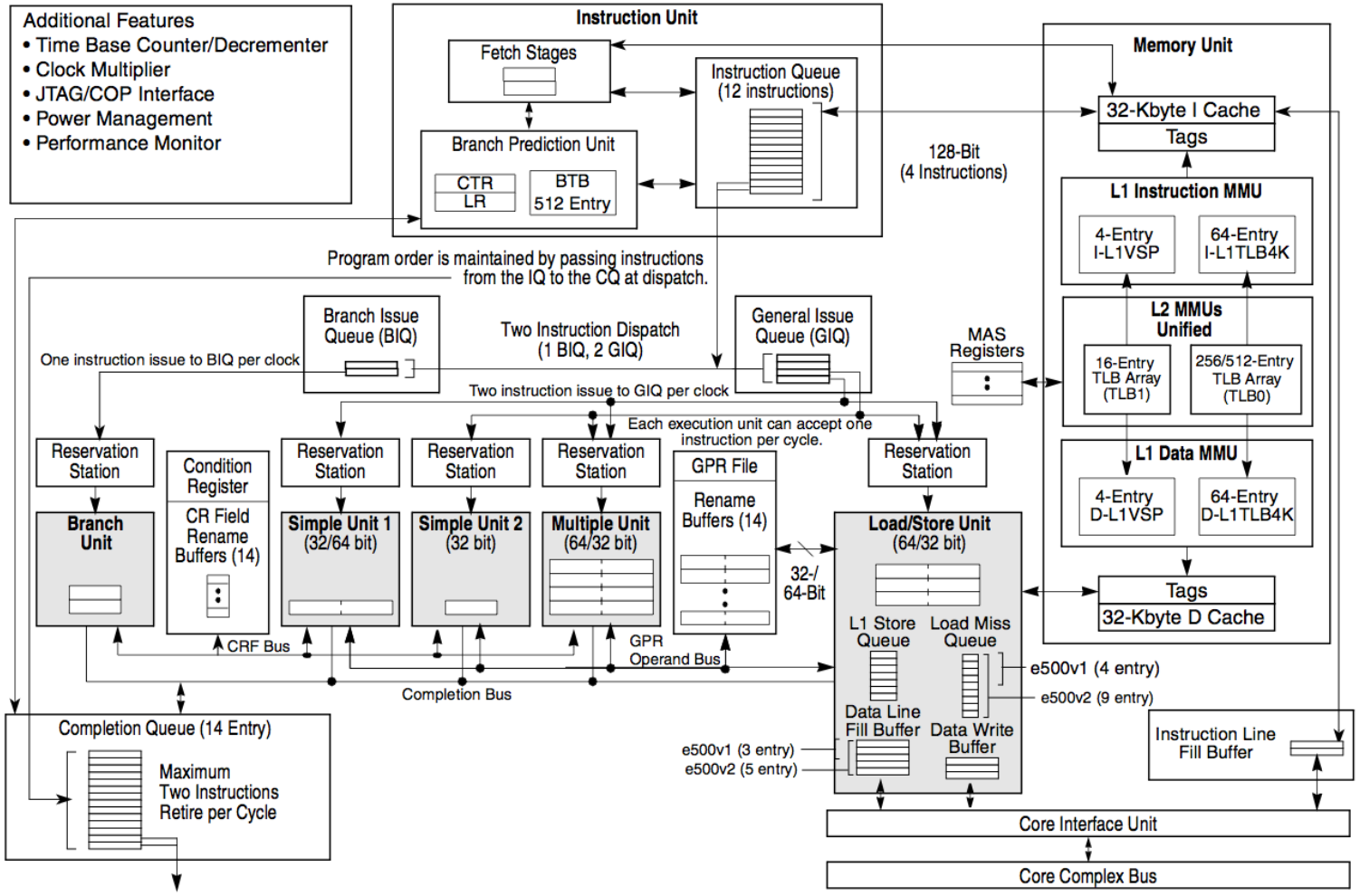
- Fixed-Point Execution Unit (FXU)
  - 1 Taktzyklus für Addition, Subtraktion, Shift, oder Rotate
  - Hardware-Multiplizierer und –Dividierer in FXU1
- Floating-Point Execution Unit (FPU)
  - optimiert für single-precision Multiplikation/Addition
  - IEEE-754 standard single- und double-precision floating-point Arithmetik
- Memory Management Unit (MMU)
  - unterstützt 52-bit virtuelle und 32-bit physikalische Adressierung
  - Hardware Adress-Translation 128-Entry Daten- und Instruktions-TLB (Translation Lookaside Buffer)

2.3.2.2 Beispiel – PowerPC 750

- Cache Units (ICache, DCache)
  - separate 32-Kbyte 8-Wege assoziative Instruktions- und Daten-Caches
  - 3-Zustands Kohärenz (Modified, Exclusive, Invalid)
  - write-back oder write-through Daten-Cache

- Bus Interface Unit (BIU)
  - allgemeines Interface für verschiedene Systemkonfigurationen
  - 32-bit Adress - und 64-bit Daten-Bus
  - JTAG-Interface Board-Test
  - Parity-Check im Bus-Interface

2.3.2.2 Beispiel - PowerPC e500 core



2.3.2.2 Superskalarprinzip und dynamisches Scheduling - Zusammenfassung

- Aus einem sequentiellen Befehlsstrom werden Befehle zur Ausführung angestoßen (zugewiesen)
- Die Zuweisung erfolgt dynamisch durch die Hardware
- Es können mehr als ein Befehl zugewiesen werden
- Die Anzahl der zugewiesenen Befehle pro Takt wird dynamisch von der Hardware bestimmt und liegt zwischen null und der maximalen Zuweisungsbreite
- Komplexe Hardware für Zuweisungslogik erforderlich
- Mehrere von einander unabhängige Funktionsanweisungen sind verfügbar.
- Mikroarchitektur bestimmt superskalare Eigenschaft

- Was ist der Unterschied zwischen Skalar und Superskalar?
- Welche zusätzlichen Einheiten werden benötigt? Welche Funktion habe diese?
- Welche Probleme ergeben sich durch parallele Pipelines?
- Was ist dynamisches Scheduling?
- Was ist spekulative Ausführung?
- Einordnung nach Flynn?

Inhalt

2.3.2.3 Very Long Instruction Word (VLIW) (I)

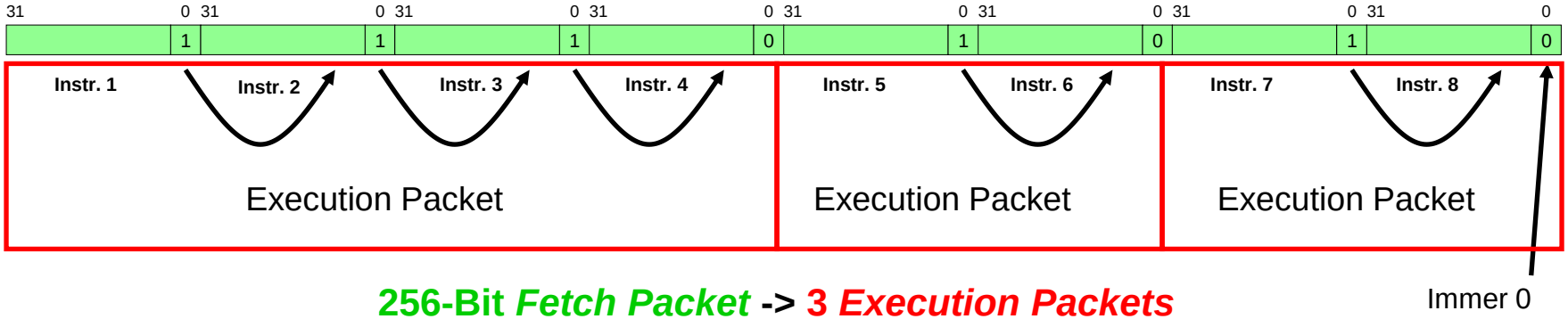
- VLIW = Very Long Instruction Word
- Architekturtechnik, bei der ein Compiler eine feste Anzahl von einfachen, voneinander unabhängigen Befehlen zu einem Befehlspaket zusammenpackt und in einem Maschinenbefehlswort meist fester Länge speichert.
- Das Maschinenbefehlsformat eines VLIW-Befehlspakets kann mehrere hundert Bits lang sein, in der Praxis sind dies zwischen 128 und 1024 Bits.
- Alle Befehle innerhalb eines VLIW-Befehlspakets müssen unabhängig voneinander sein und eigene Opcodes und Operandenbezeichner enthalten.

2.3.2.3 Very Long Instruction Word (VLIW) (II)

- Die Anzahl der Befehle in einem VLIW-Befehlspaket ist in der Regel fest.
  - Wenn die volle Bandbreite eines VLIW-Befehlspakets nicht ausgenutzt werden kann, muss es mit Leerbefehlen aufgefüllt werden.

- Problem bei VLIW-Architekturen:
  - volle Parallelität nicht immer ausnutzbar Befehlswort teilweise unnötig lang
  - Neuere VLIW-Architekturen sind in der Lage, durch ein komprimiertes Befehlsformat auf das Auffüllen mit Leerbefehlen zu verzichten.

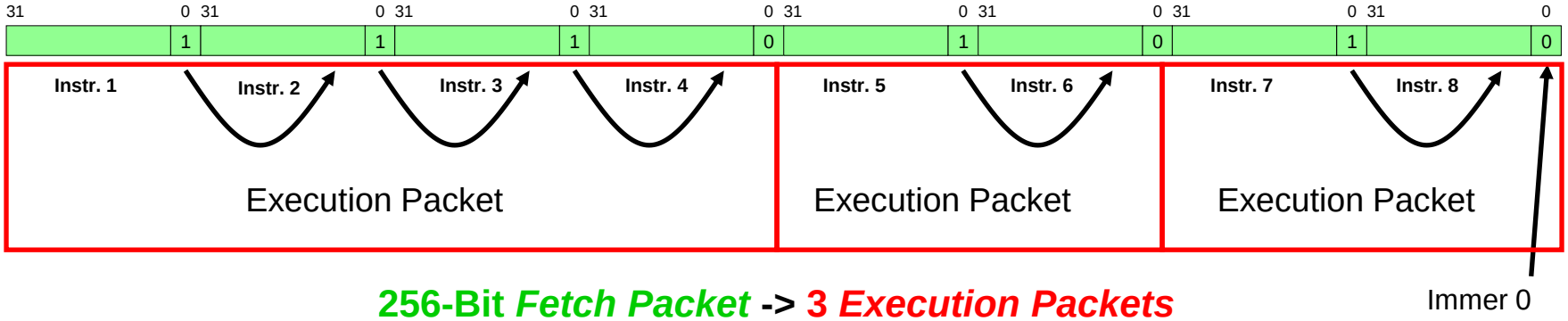
- Lösung: Instruction-Packing: Instruktionswort enthält mehrere Teil-Instruktionen



256-Bit *Fetch Packet* -> 3 *Execution Packets*

2.3.2.3 Very Long Instruction Word (VLIW) (III)

- Befehlswort enthält bis zu 8 32-Bit-Befehle (je 5 ns-Zyklus) = Befehlspaket
  - ein Bit gibt an, ob nächster Befehl zum selben Befehlspaket gehört,
  - erlaubt Befehle unterschiedlicher Bitbreite
- Teil-Instruktionen bedienen sich nebenläufiger Funktions-Einheiten.
- Komplizierte Compiler/Code-Generatoren
  - Scheduling muss konfliktfrei für die Ausführung geplant sein.



256-Bit *Fetch Packet* -> 3 *Execution Packets*

2.3.2.3 Very Long Instruction Word (VLIW) (IV)

- Vorläufer zu Superskalarität
- Geringe Codedichte
  - Compiler schwer zu entwickeln
- Vom Compiler generiertes Scheduling zur Laufzeit von Hardware durchzuführen
- VLIW vor ca. 15 Jahren mit hoher Erwartung entwickelt
  - Erreichbarer Parallelitäts-Grad für viele Anwendungen nicht konstant
  - nur für Anwendungen mit viel Parallelität sinnvoll
- Anwendungen müssen von der Struktur her geeignet sein
  - Effizienz bei hoher synchroner Parallelität auf Instruktionsebene

2.3.2.3 Very Long Instruction Word (VLIW) (V)

- Ein VLIW-Prozessor besteht aus einer Anzahl von Ausführungseinheiten, die jeweils eine Maschinenoperation taktsynchron zu den anderen Maschinenoperationen eines VLIW-Befehlspakets ausführen können, wobei ein VLIW-Befehlspaket maximal so viele einfache Befehle umfasst, wie Ausführungseinheiten in dem VLIW-Prozessor vorhanden sind.
- Der Prozessor startet im Idealfall in jedem Takt ein VLIW-Befehlspaket.
- Die Befehle in einem solchen VLIW-Befehlswort werden dann gleichzeitig geholt, decodiert, zugewiesen und ausgeführt.

2.3.2.3 Very Long Instruction Word (VLIW) (VI)

- In Abhängigkeit von der Anzahl n der Befehle, die gemeinsam durch die Pipeline fließen und entsprechend der maximalen Anzahl n von Befehlen in einem Befehlspaket spricht man von einem n-fachen VLIW-Prozessor.

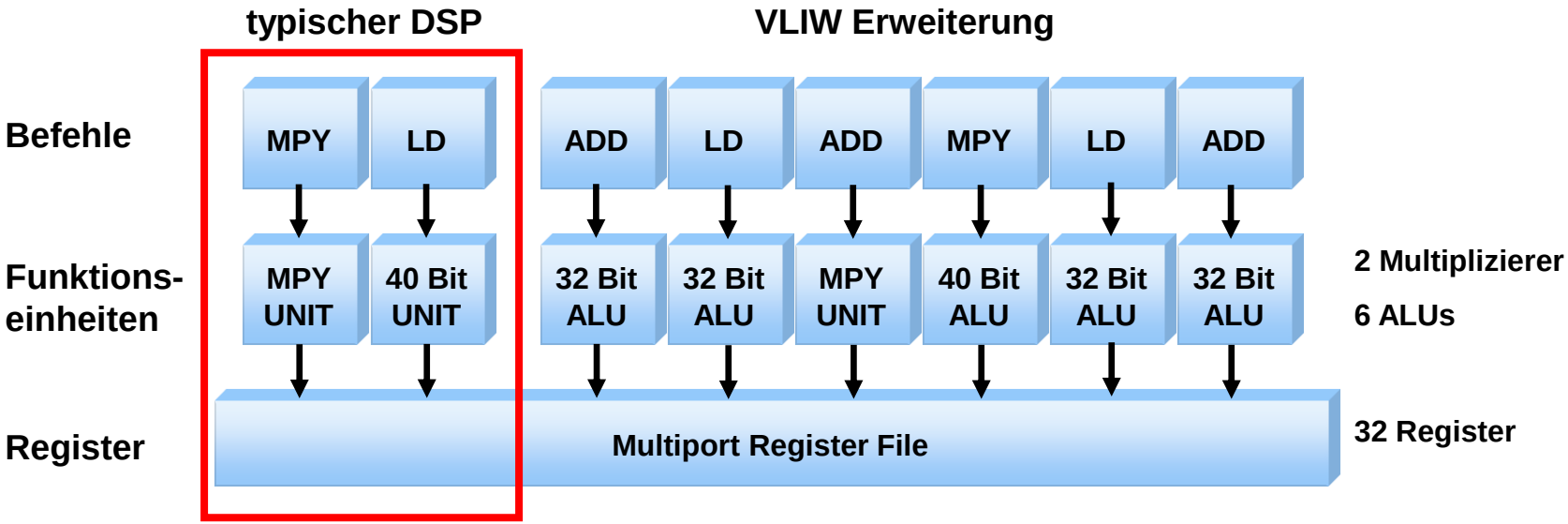
- Ablaufreihenfolge der VLIW-Befehlspakete ist fest, Ausführung auch außerhalb der Programmreihenfolge nicht möglich.

- Operanden stehen in einem allen Ausführungseinheiten zugänglichen Registersatz.

2.3.2.3 Beispiel TI TMS 320C62x

- TI DSP TMS 320C62x-Familie mit VLIW-Architektur
  - 1997/98: schnellste DSP-Familie mit ca. 1600 MIPS, 200 MHz / 5 ns Zykluszeit

- Verlagerung der Komplexität: von HW in optimierende Compilerschritte
  - zur Übersetzungszeit parallel ausführbare Rechenoperationen bestimmen
  - einfachere Prozessoren (Fläche):
    - TMS320C6201: 550.000 Transistoren « Pentium: 5.000.000 Transistoren



2.3.2.3 VLIW-Befehl vs. CISC und SIMD-Befehl

- Unterschied zu einem CISC-Befehl:
  - CISC-Befehl kann mit einem Opcode mehrere, eventuell sequenziell nacheinander ablaufende Operationen codieren

- Unterschied zu einem SIMD-Befehl:
  - SIMD-Befehle sind die Multimediabefehlen, bei denen ein Opcode eine gleichartige Operation auf einer Anzahl von Operanden(paaren) auslöst.
  - Die Operationen innerhalb eines VLIW-Befehlspakets sind in der Regel verschiedenartig.

- Was ist der Unterschied zu Skalar und Superskalar?
- Wo findet die Parallelisierung statt?
- Wie unterscheidet sich die Instruktion?
- Wie muss die Hardware aufgebaut sein?
- Einordnung nach Flynn?

Inhalt

2.3.2.4 Single Instruction Multiple Data (SIMD)

- Gleiche Instruktion auf mehreren Daten
  - Multimediaanwendungen
    - Viele Daten gleichzeitig
    - Viele Parallelisierungsmöglichkeiten
- Reduziert Zyklenzahl für gleiche Menge von Daten
- Vektorisierung von Daten
  - Parallele Berechnung der Vektoren
  - Speichern der einzelnen Daten in Register
- Schwer zu implementieren
  - Schwer automatisch Code zu finden, der SIMD gut ausnutzt
    - Code von Hand schreiben/optimieren

2.3.2.4 SIMD – Multimedia-Erweiterungen

- Multimedia-Anwendungen
  - z.B.: Audio/Video Playback, DVD, Videokonferenzen
  - 8/16-Bit Datentypen (z.B.: Pixeldarstellung in Bildverarbeitung)
  - viele arithmetische Operationen (Multiplikation, Addition)
  - große Datenmengen, große I/O Bandbreite
  - viel Datenparallelität Single Instruction Multiple Data (SIMD)

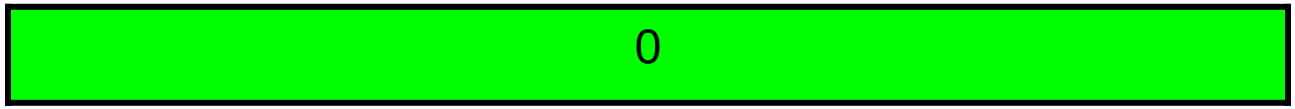
- Sub-Word Execution Model
  - 32/64-Bit Register und ALUs in kleinere Einheiten auftrennen
  - Instruktionen arbeiten parallel auf den Einheiten -> SIMD

- Beispiele:
  - MMX (Intel), SSE (Intel), VIS (UltraSparc), MDMX (MIPS), MAX-2 (HP)

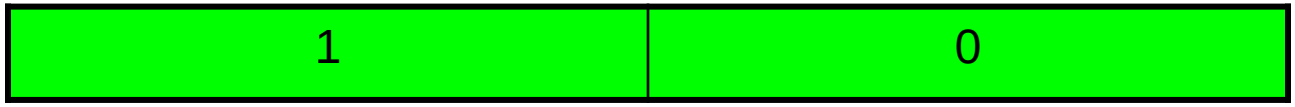
2.3.2.4 SIMD – Sub-Word Execution Model (I)

- Beispiel: 64-Bit Datentyp in kleinere Einheiten auftrennen
- Instruktionen parallel auf kleineren Ausführungseinheiten

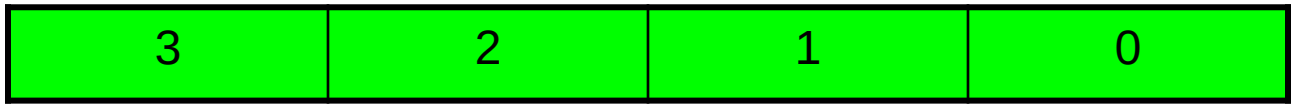
- Long Word



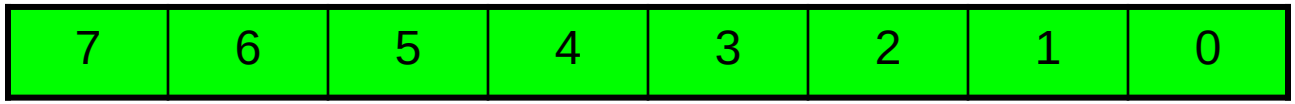
- Packed Word



- Packed Half-Word



- Packed Byte

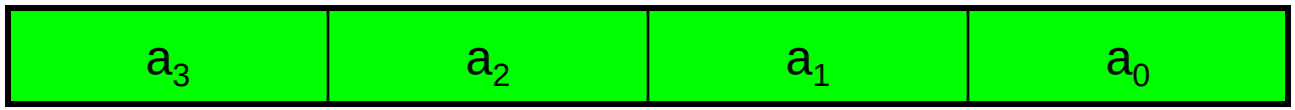


2.3.2.4 SIMD – Sub-Word Execution Model (II)

- Beispiel 1: Addition
  - Instruktionen parallel auf 4 Ausführungseinheiten

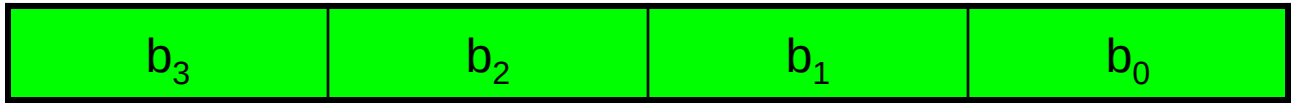
- ADD R3 := R1, R2

- R1



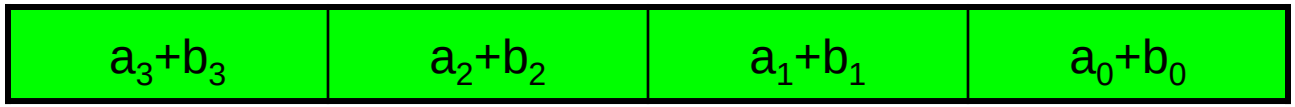
+ + + +

- R2



= = = =

- R3

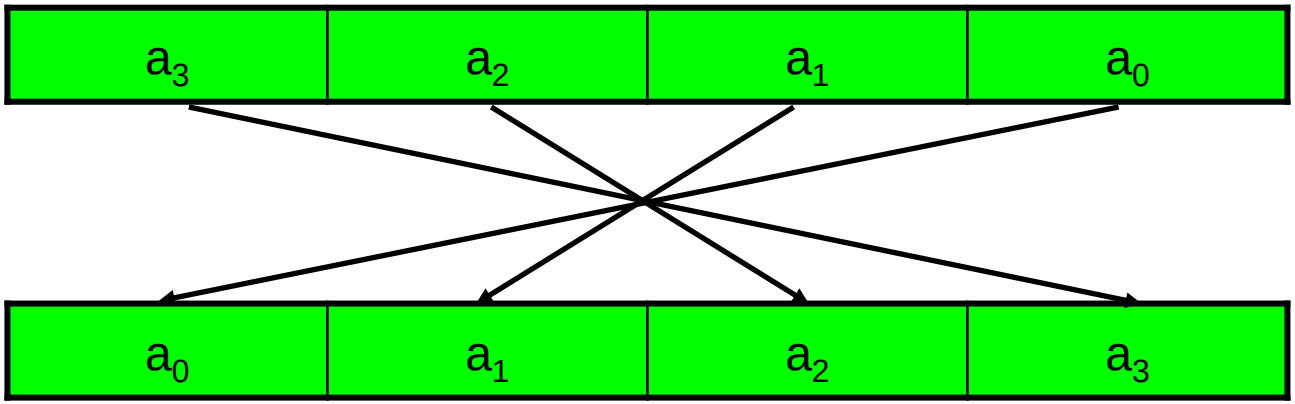


2.3.2.4 SIMD – Sub-Word Execution Model (III)

- Beispiel 2: Permutation
  - Vertauschen der Reihenfolge der Half-Words

- PERMUTE R3 := R1 (Muster 0123)

- R1



- R3

- Welche Funktion haben SIMD-instruktionen?
- Welche Parallelität wird ausgenutzt?
- In welchen Anwendungen macht SIMD Sinn?
- Welche bekannten Beispiele gibt es für SIMD-Funktions-Einheiten?

Inhalt

2.3.2.5 Caches (I) - Beispiel

- Sie schreiben einen Bericht und sitzen an einem Tisch in der Bibliothek (Bibliothek = Hard Disk). Sie haben Zugriff auf ein riesiges Repertoire, aber leider dauert es ein Buch zu suchen und zu holen…
- Der Tisch ist Ihr „Speicher“ (kleinere Kapazität als die Bibliothek) auf dem Sie sehr schnell ein Buch finden sobald es sich dort befindet. Jedoch ist der Tisch auch mal voll von Büchern…dann müssen Bücher zurück gegeben werden
- Geöffnete Bücher sind „Cache“ da Sie sehr schnell darin lesen können
- Behalten Sie so viele Bücher wie möglich auf dem Tisch
- Welche Strategie nutzen Sie Bücher aufzuschlagen und wieder zu schließen bzw. Zurückzustellen?

2.3.2.5 Caches (II)

- Caches reduzieren unperformante Hauptspeicherzugriffe, durch Einsatz schnellerer Zwischenspeicher
  (Hauptspeicher ist 15 x langsamer).

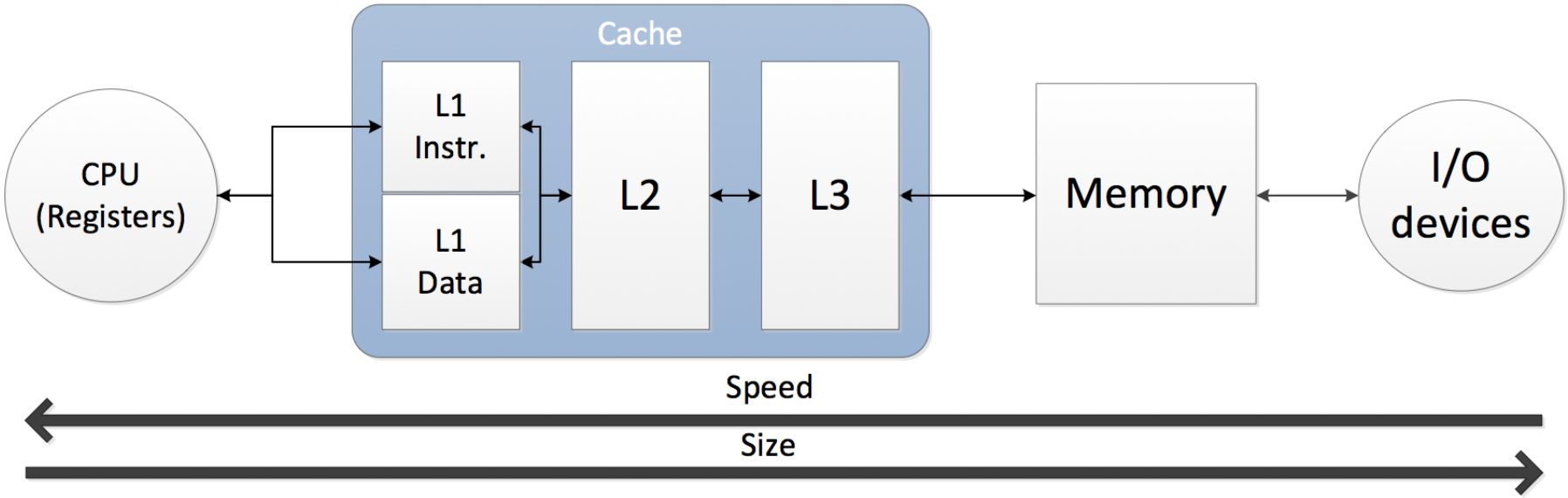
- Cache kommt aus dem Französischen: cacher (verstecken)

- Das Cache ist software-transparent, d.h. der Benutzer braucht nichts von seiner Existenz zu wissen.

2.3.2.5 Caches - Hierarchie

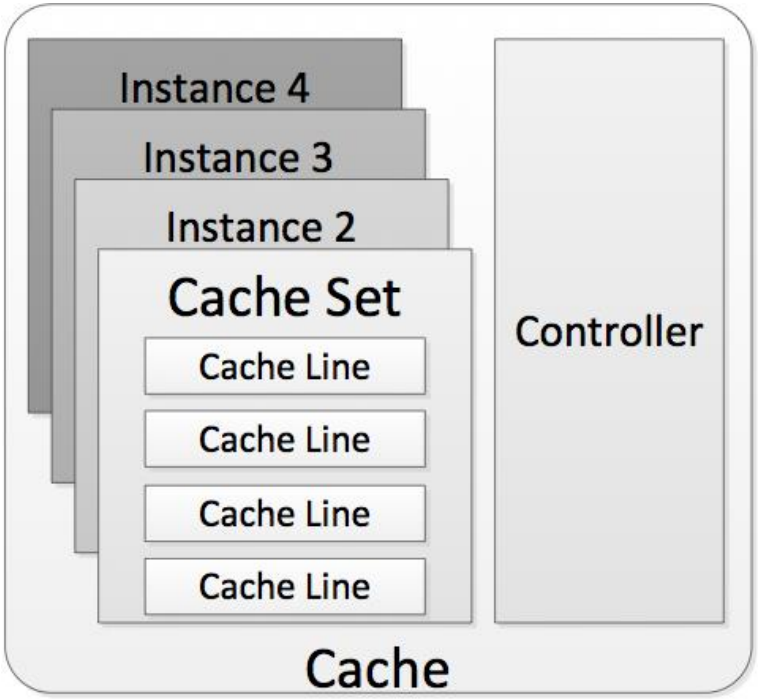
- In der Regel besitzt ein Rechner einen getrennten Cache für Instruktionen (Instruktionscache) und für Daten (Datencache)

- Cache Level: Level 1 bis Level N bezeichnet die Cache Speicher beginnend mit dem schnellsten Speicherzugriff



2.3.2.5 Caches - Aufbau (I)

- Cache besteht aus
  - Controller
  - mehreren Sets
    - Cache Lines

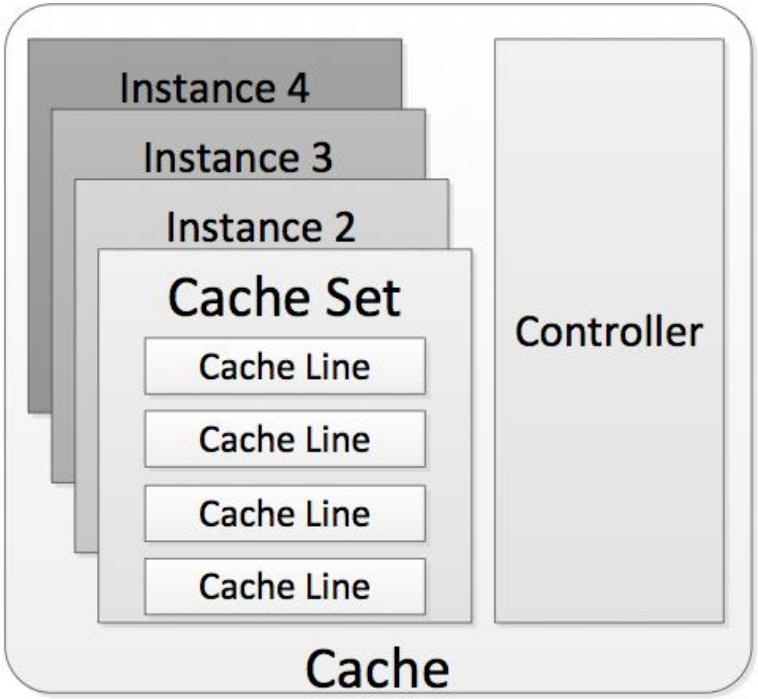


4-Wege assoziativer Cache

© Bachelorarbeit Maximilian Braun, ITIV 2012

2.3.2.5 Caches - Aufbau (II)

- Parameter:
  - Wordbreite (M)
  - Assoziativität (N)
  - Words-per-Line (WpL)
  - Linesize
    - Linesize [bit] = WpL * M
  - Anzahl Cache Sets (S)
  - Cache-Größe
    - Größe [bit] = S * N * WpL * M
- Verhaltensparameter:
  - Replacement-Strategy
  - Cache-Strategy



4-Wege assoziativer Cache

2.3.2.5 Caches - Lesezugriff

- Lese Datum aus dem Arbeitsspeicher unter Adresse address:

- CPU überprüft, ob eine Kopie der Hauptspeicherzelle address im Cache Abgelegt ist.
  - Falls ja (cache hit)
    - so entnimmt die CPU das Datum aus dem Cache. Die Überprüfung und das eigentliche Lesen aus dem Cache erfolgt in einem Zyklus, ohne ein Wartezyklus einfügen zu müssen.
  - Falls nein (cache miss)
    - so greift die CPU auf den Arbeitsspeicher zu, lädt den umgebenden Block des Datums in den Cache und lädt das Datum von dort in die CPU

2.3.2.5 Caches – Schreibzugriff / Cachestrategie

- Schreibe Datum in den Arbeitsspeicher unter Adresse address:
- CPU überprüft, ob eine Kopie der Hauptspeicherzelle address im Cache abgelegt ist

  - Write Back: (Cache hit)
    - Änderungen werden zuerst nur in das Cache geschrieben (Setzen eines Dirty-Bits). Erst bei einer Auslagerung der Cachezeile in Hauptspeicher wird diese zurück geschrieben. (Konsistenzprobleme bei Multiprozessoren)
  - Write Through: (Cache hit)
    - Jede Änderung wird zugleich in Cache und Hauptspeicher aktualisiert. (Belastet sehr den Speicherbus)

  - Write Around: (Cache miss)
    - Das Ziel ist zum Schreiben nicht im Cache. Die Änderung wird ohne Cache-Update direkt in den Hauptspeicher geschrieben
  - Write-Allocate: (Cache miss)
    - Zeile ist nicht im Cache. Block wird aus Hauptspeicher in Cache-Zeile kopiert und dann eine Schreib-Treffer-Operation (Write Back, Write Through) ausgeführt

- Cache Flush: Zurückschreiben des kompletten Caches in den Hauptspeicher

2.3.2.5 Cache miss, Heiße und kalte Caches

- Heißer Cache: Cache arbeitet optimal, nur wenige Misses
- Kalter Cache: hohe Missrate

- Es werden drei Arten von Cache Misses unterschieden:
  - **Capacity**:
    - Daten wurden verdrängt, weil der Cache zu klein ist.
      → Vergrößerung des Caches
  - **Conflict**:
    - In einem Set ist nicht ausreichend Platz obwohl andere Sets frei sind.
      → Erhöhung der Assoziativität
  - **Compulsory**:
    - Beim erstmaligen Zugriff befinden sich die Daten noch nicht im Cache.
      → Spekulatives Laden durch Prefetcher

2.3.2.5 Caches - Verdrängungsstrategien

- Szenario
  - Cache miss, alle Speicherbereiche des Caches belegt
- Ausweg
  - verdränge einen Block aus dem Cache, lade an seine Stelle den gerade benötigten Block

2.3.2.5 Caches - Verdrängungsstrategien

- FIFO (First-In-First-Out):
  - schlechte Strategie, da der Block, der zuerst geladen wurde, als erstes überschrieben wird
  - Blöcke werden überschrieben, die häufig gebraucht werden, z.B. während Initialisierung
- Optimale Ersetzungsstrategie:
  - der Block, dessen Daten in Zukunft am längsten (zeitlich) nicht mehr angesprochen werden, wird ausgelagert
  - Nachteil: diese Information steht meist nicht zur Verfügung
- LRU (Least-Recently-Used):
  - der Block, der in der Vergangenheit am längsten (zeitlich) nicht mehr gebraucht wurde, wird ausgelagert
- Random
  - Zufällige Auswahl des Blockes, der ersetzt wird

2.3.2.5 Caches - Begriffe (I)

- Blockrahmen / Block Frames
  - Anzahl von Speicherplätzen dazu ein Address Tag sowie Status bits (Gesamtzahl von Blockrahmen C)
- Cache Block / Cache Line
  - Speichermenge die in einen Blockrahmen passt
- Blocklänge
  - Anzahl der Speicherplätze im Cache Block (z.B. 16 / 32 Bytes)
- Address Tag enthält Teil der Speicheradresse

2.3.2.5 Caches - Begriffe (II)

- Sätze / Sets
  - Menge der Blockrahmen eines Cache Speichers (Anzahl Sätze S)
- Anzahl der Blockrahmen in einem Satz wird Assoziativität bezeichnet (Assoziativität N)
- Die Gesamtzahl C der Blockrahmen in einem Cache, entspricht immer dem Produkt aus der Anzahl S der Sätze und der Assoziativität
- C=S*N

Zeile 1	Adress-Tag	Datenblock	Control(Bits)
Zeile 2	Adress-Tag	Datenblock	Control(Bits)
Zeile 3	Adress-Tag	Datenblock	Control(Bits)
…	…	…	…
Zeile n	Adress-Tag	Datenblock	Control(Bits)

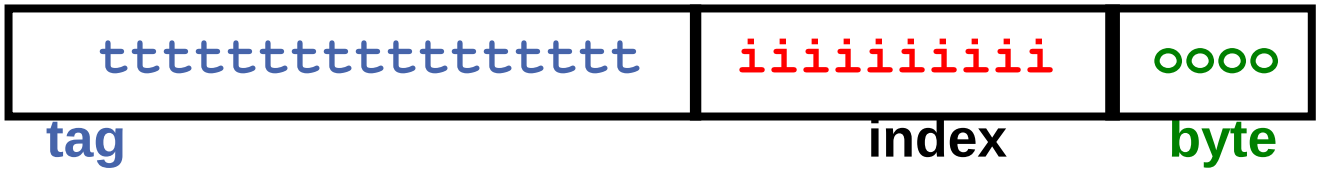
2.3.2.5 Caches - Organisation

- Direkt abgebildet - Direct-Mapped (N=1)
  - Also S=C
  - schnell
  - einfach
  - schlechte Trefferrate

- Mehrwege – Assoziativ
  - Also S=C/N
  - Vermischung von *direct-mapped* und *voll-assoziativ*

- Voll – Assoziativ (S=1)
  - Also C=N
  - langsam
  - kompliziert
  - sehr gute Trefferrate

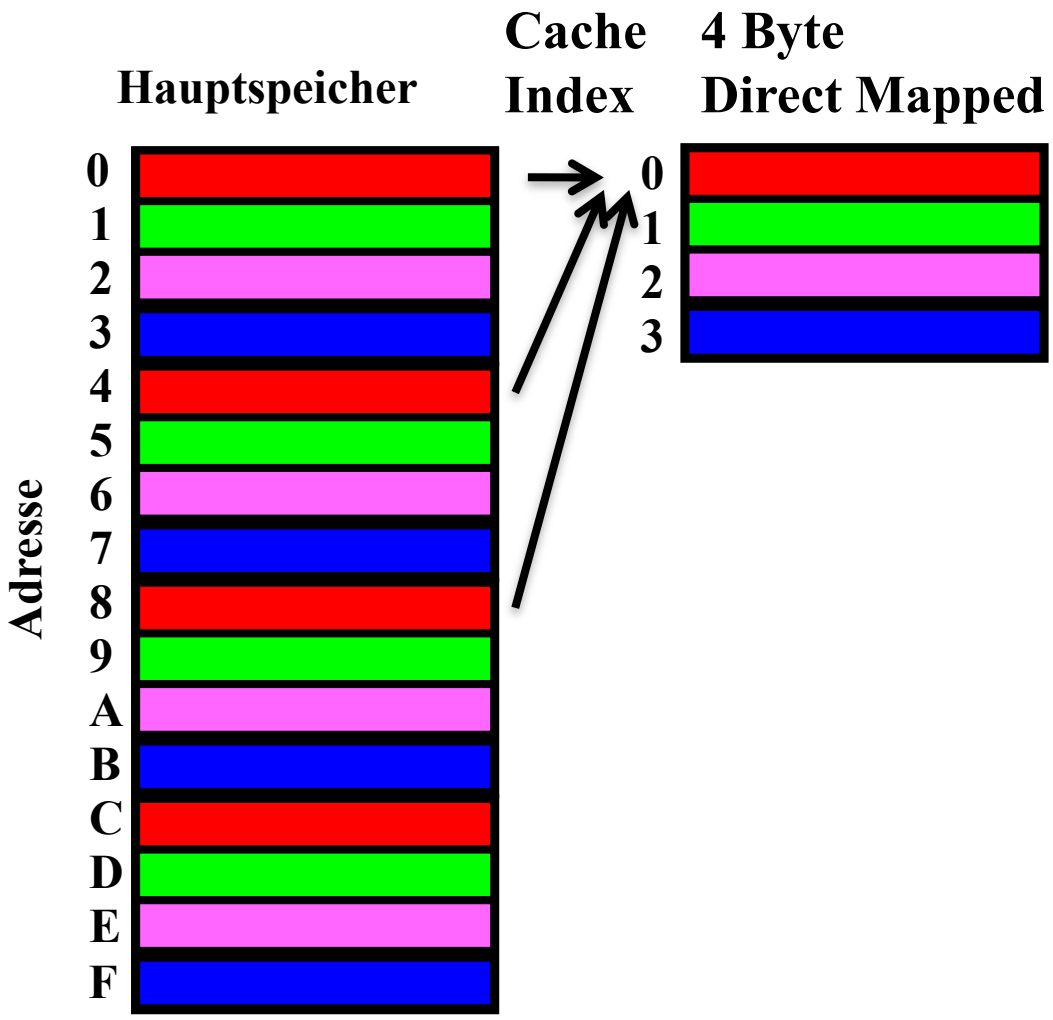
- Speicheradresse in 3 Bereiche trennen:



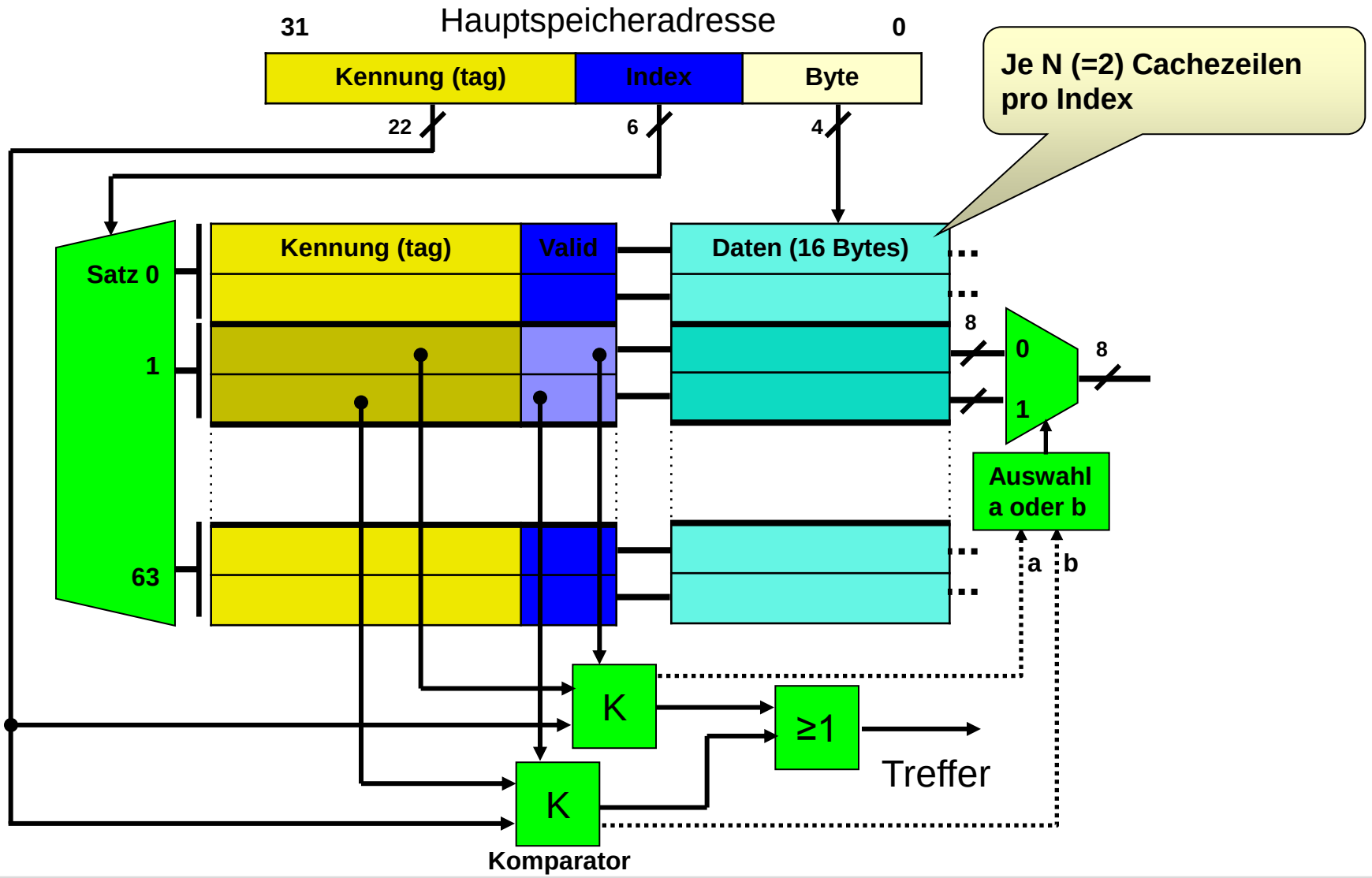
- C = Anzahl der Blockrahmen (Cacheline + Overhead)
- S = Anzahl der Sätze
- N = Assoziativität

2.3.2.5 Caches – Direct Mapped Cache

- Jede Speicheradresse (Hauptspeicher) ist einem Block im Cache Speicher zugeordnet
- Deshalb muss lediglich an einer einzigen Stelle nachgesehen werden ob die Daten dort existieren oder nicht
- “Block” ist die Einheit die zwischen Cache und Memory (also Hauptspeicher) transferiert werden muss

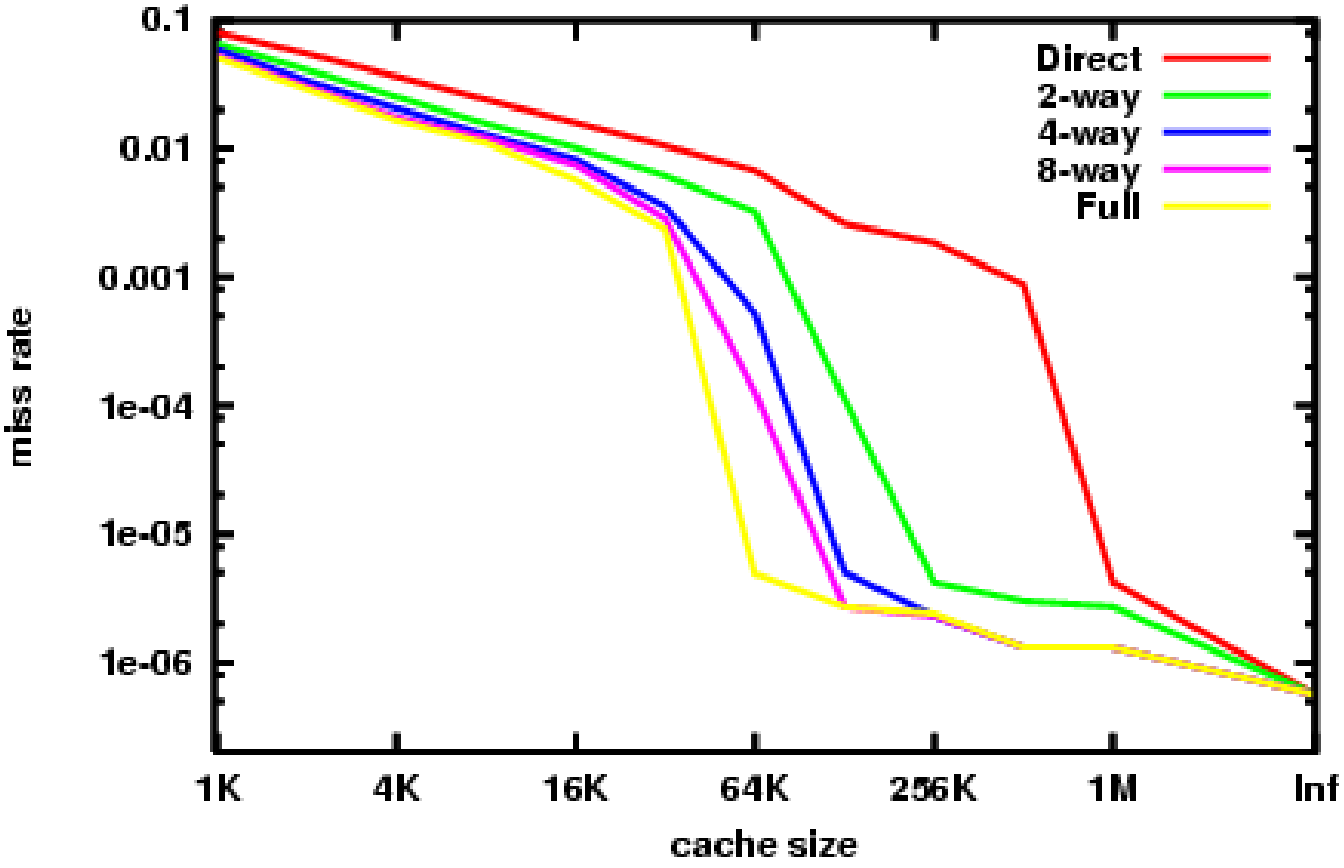


2.3.2.5 Caches - 2-Wege assoziativer Cache



2.3.2.5 Caches – Missrate Vergleich

- Cache Performance für Spec CPU2000 Benchmark



http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/

- Was ist ein Cache?
- Welche Funktion hat ein Cache in einem 1-Computer System? Welche in einem 2-Computer-System?
- Welche Cache-Stragegien gibt es? Welche Vorteile hat welche Strategie?
- Welche Schreib-Strategien gibt es? Welche machen Sinn?
- Wie sind Offset, Index & Tag aufgeteilt?
- Was ist Cache Kohärenz?

Inhalt

2.3.2.6 Multiple Instruction Multiple Data (MIMD)

- Mehrere General Purpose Prozessoren sind miteinander verbunden
- Im Gegensatz zu SIMD-Prozessoren können MIMD-Prozessoren mehrere Programme auf mehreren Prozessoren ausführen
  - Flexibilität
- In der 90ern hat SIMD an Relevanz verloren, da Mikroprozessoren sehr billig und leistungsfähig wurden
  - MIMD-Prozessoren können aus off-the-shelf Mikroprozessoren aufgebaut werden
- Als Variante kann single program multiple data streams (SPMD) das selbe Programm auf mehreren Prozessoren ausführen.
  - Vermeidet Probleme des parallelen Programmierens
  - Eher Programmiermodell als Architekturkonzept!

2.3.2.6 SIMD vs MIMD

- SIMD Computer brauchen weniger Hardware als MIMD Computer
  - Eine Steuer-Einheit
- SIMD Prozessoren sind speziell designed, sind teuer und haben lange Design-Zyklen
- Nicht alle Applikationen passen natürlicher Weise zu SIMD
- Konzeptionell überdecken MIMD Computer die SIMD Anforderungen
  - Alle Prozessoren führen das selbe Programm aus (SPMD)
  - SPMD kann mit wenig Aufwand günstig aus off-the-shelf Komponenten aufgebaut werden

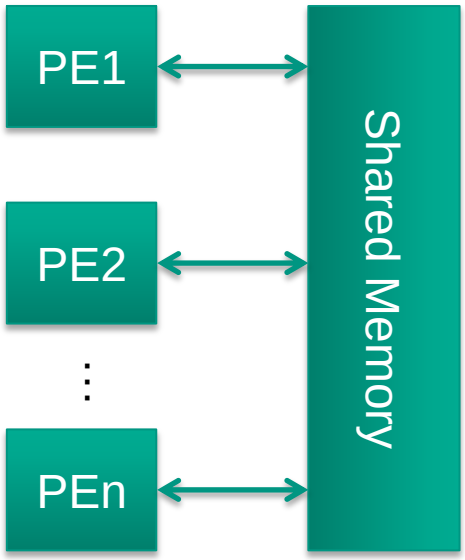
2.3.2.6 MIMD Prozessor Klassifizierung

- Shared Memory:
  - Zentraler Speicher, bestehend aus mehreren Modulen, mit der gleichen Entfernung zu den Prozessoren
    - Symmetric Multiprozessor (SMP)
    - Uniform Memory Access (UMA)

- Distributed Memory
  - Speicher ist verteilt für jeden Prozessor, um die Skalierbarkeit zu verbessern
    - Message Passing Architectures
      - Kein Prozessor kann direkt den Speicher eines anderen Prozessors ansprechen
    - Hardware Distributed Shared Memory (DSM)
      - Der Speicher ist verteilt, jedoch ist der Adressraum geteilt
    - Software Distributed Shared Memory (DSM)
      - Eine OS-Ebene gibt dem Programmierer das Bild eines geteilten Speichers

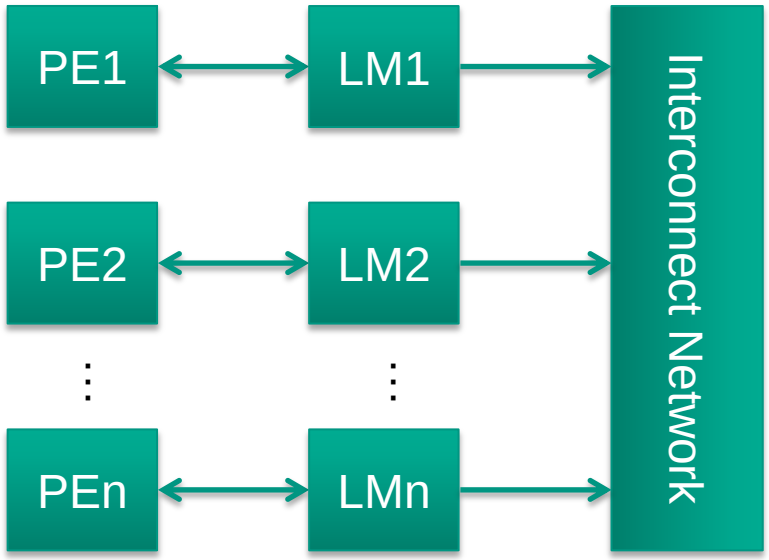
2.3.2.6 MIMD mit Shared Memory

- Typischerweise als Multi-Prozessor bezeichnet
  - Thightly coupled
  - Nicht skalierbar



2.3.2.6 MIMD mit Distributed Memory

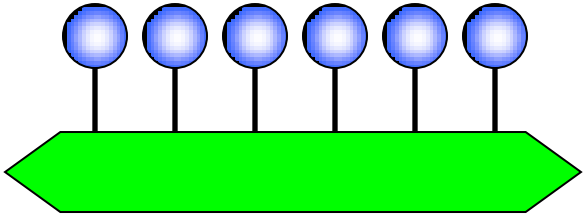
- Typischerweise als Multi-Computer bezeichnet
  - Loosely coupled mit message passing
  - Skalierbar



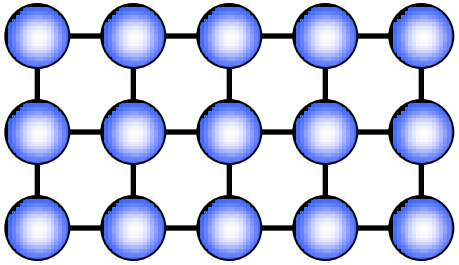
2.3.2.6 Interconnect in MIMD Architekturen

- Kommunikation über gemeinsame Variable oder Nachrichten
- Kopplung über Verbindungs-Netzwerke
- Typischerweise Parallelrechner (Einsatz z.B. Wettersimulationen, Chemie etc.)

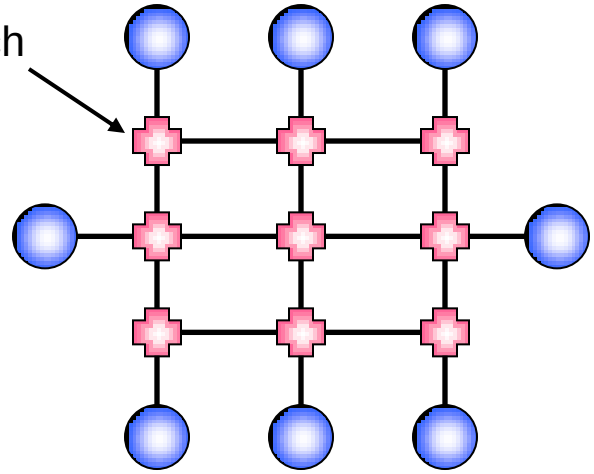
Verbindungs-Netzwerke:



Multiprozessorbus



Statisches Verbindungsnetzwerk



Dynamisches Verbindungsnetzwerk

2.3.2.6 MIMD Design Issues

- Design Issues in Bezug zu einer MIMD Maschine sind sehr komplex, da Hardware und Software Probleme involviert sind.

- Wichtige Probleme:
  - Prozessor design
  - Physikalische Organisation
  - Interconnect Struktur
  - Inter-Prozessor Kommunikationsprotokolle
  - Speicher Hierarchie
  - Cache Organisation und Kohärenz
  - Betriebssystem design
  - Parallele Programmiersprachen

2.3.2.6 Beispiel: Intel Core Architektur (I)

- Harvard-Architektur
- 14-stufige Pipeline
  - 4 Instruktions-Decode Einheiten
- Microcode ROM:
  CISC → RISC Befehle (Micro-Ops)
- Superskalar: 3 ALUs, Scheduler, Out-of-Order
- SIMD (MMX, SSE)
- uvm.

- Anm: auf den folgenden Folien sind aufgrund fehlender öffentlicher Dokumentation Bilder unterschiedlicher Core Architekturversionen.

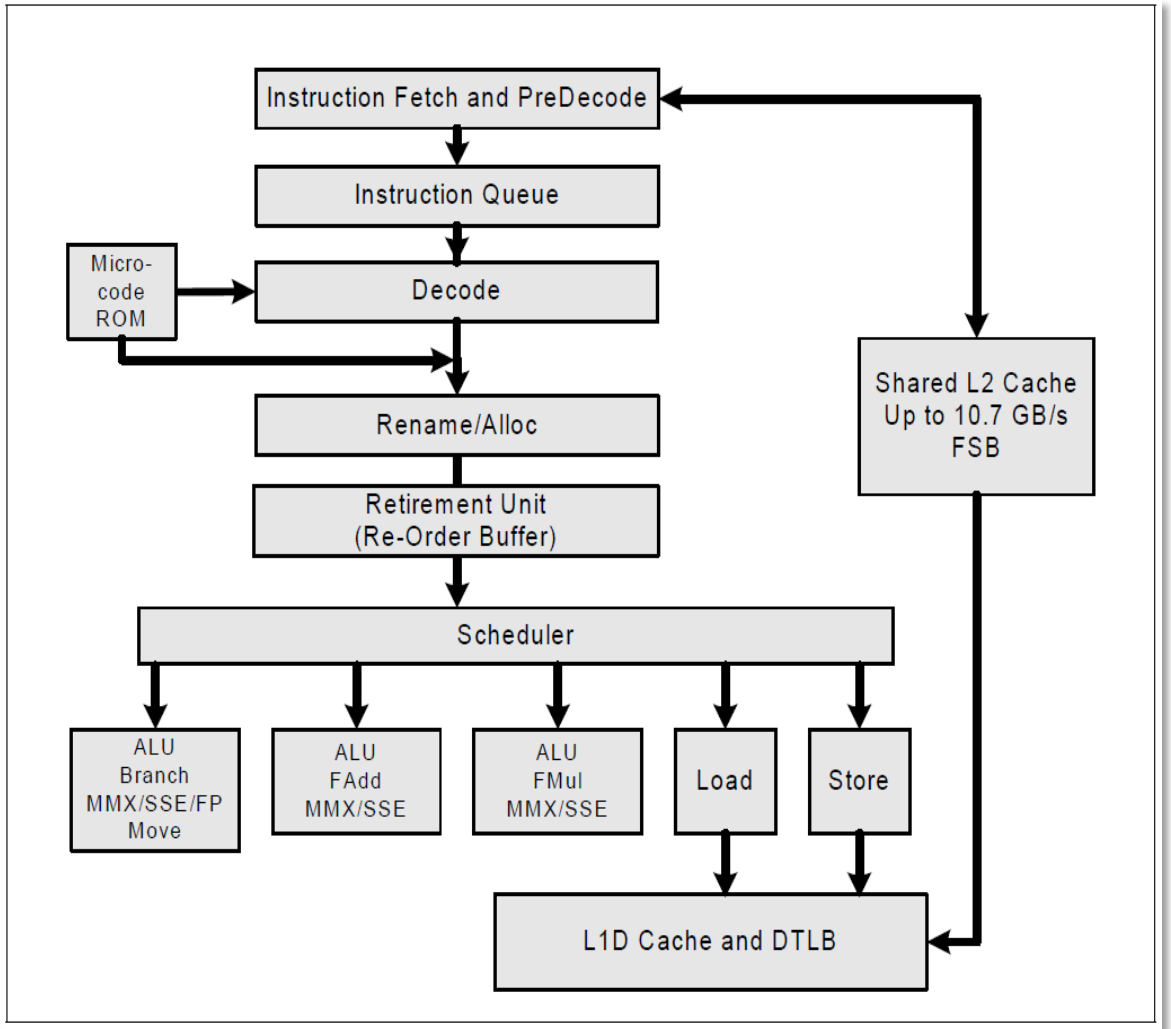
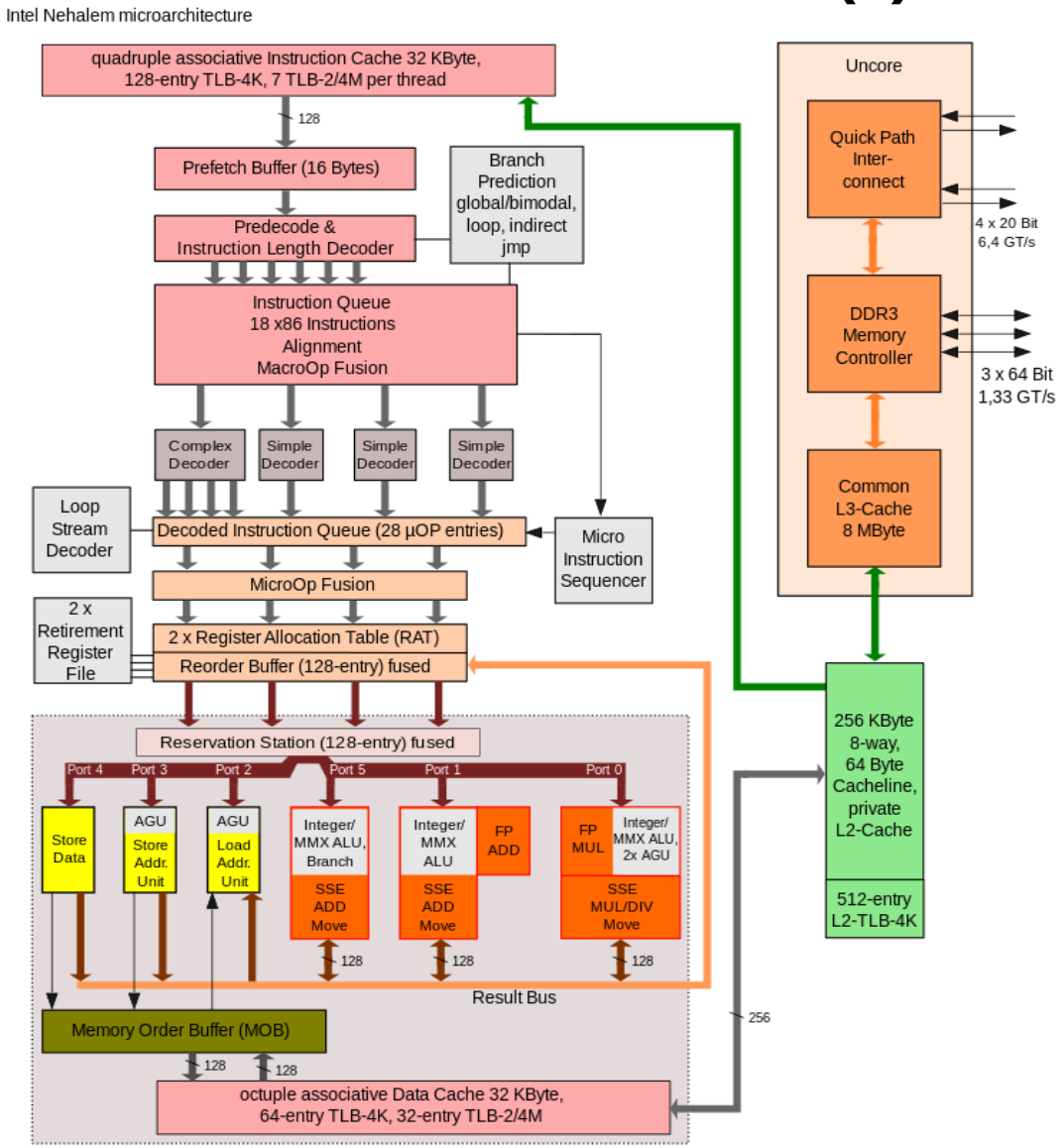


Figure 2-3. The Intel Core Microarchitecture Pipeline Functionality

Quelle: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-1-manual.pdf

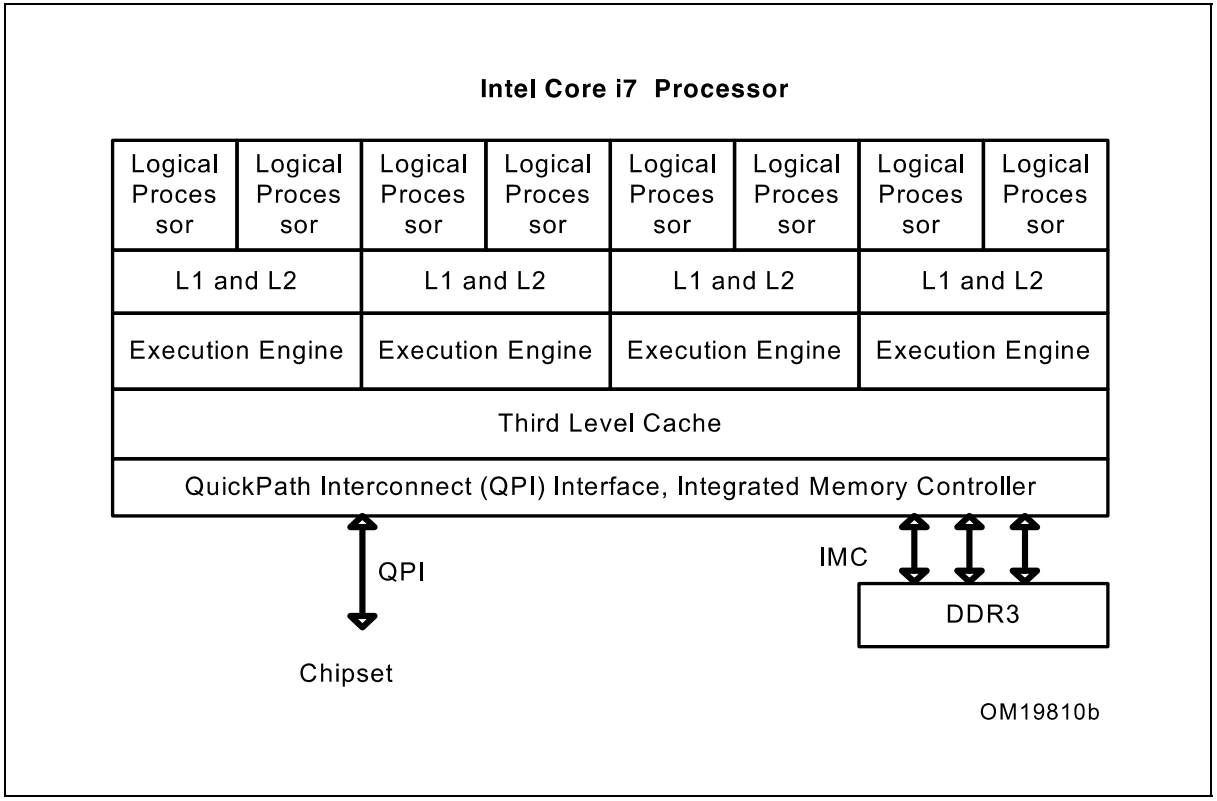
2.3.2.6 Beispiel: Intel Nehalem Architektur (II)



Quelle: Wikipedia

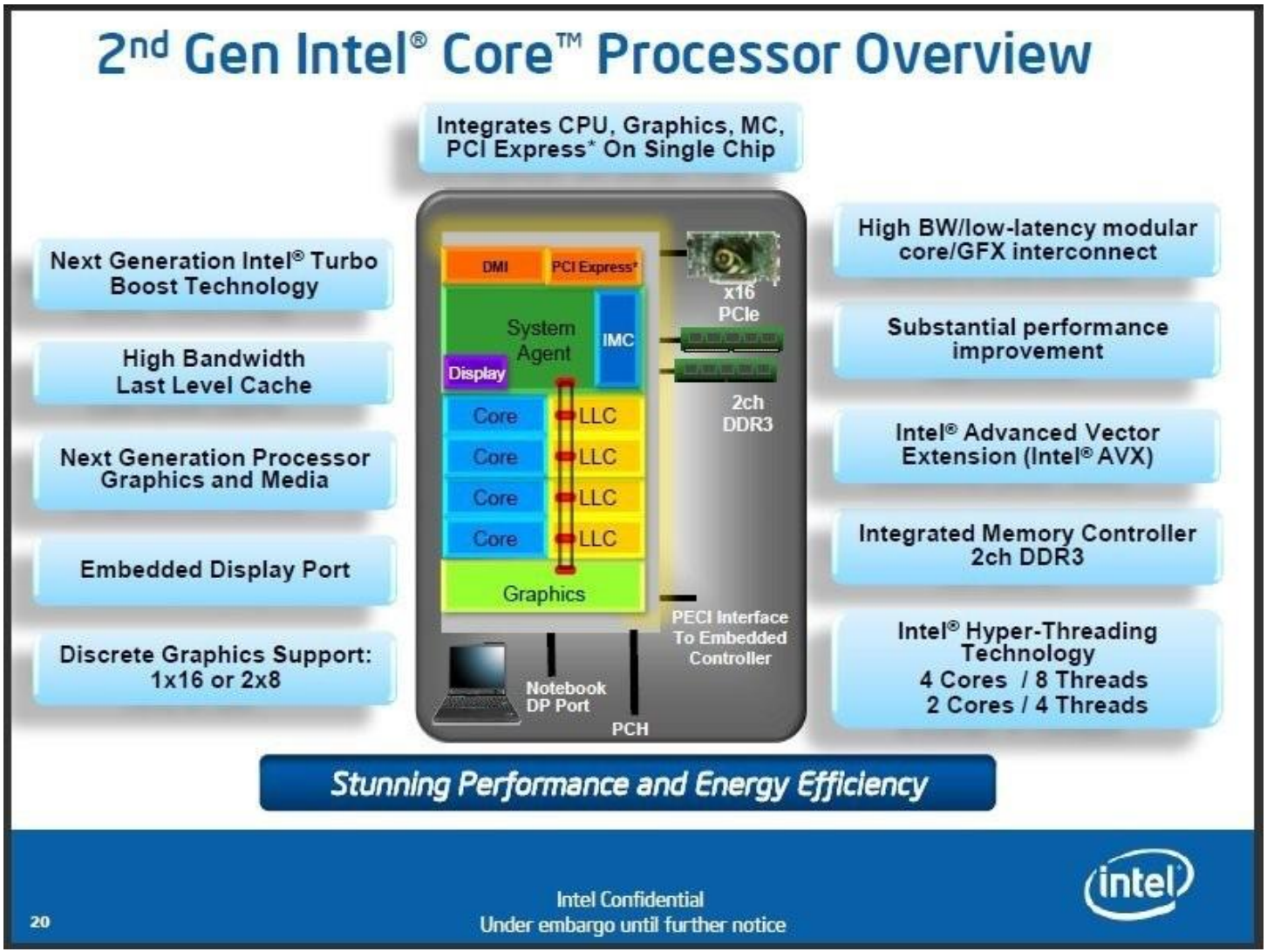
GT/s: gigatransfers per second

2.3.2.6 Beispiel: Intel Core i7 Processor

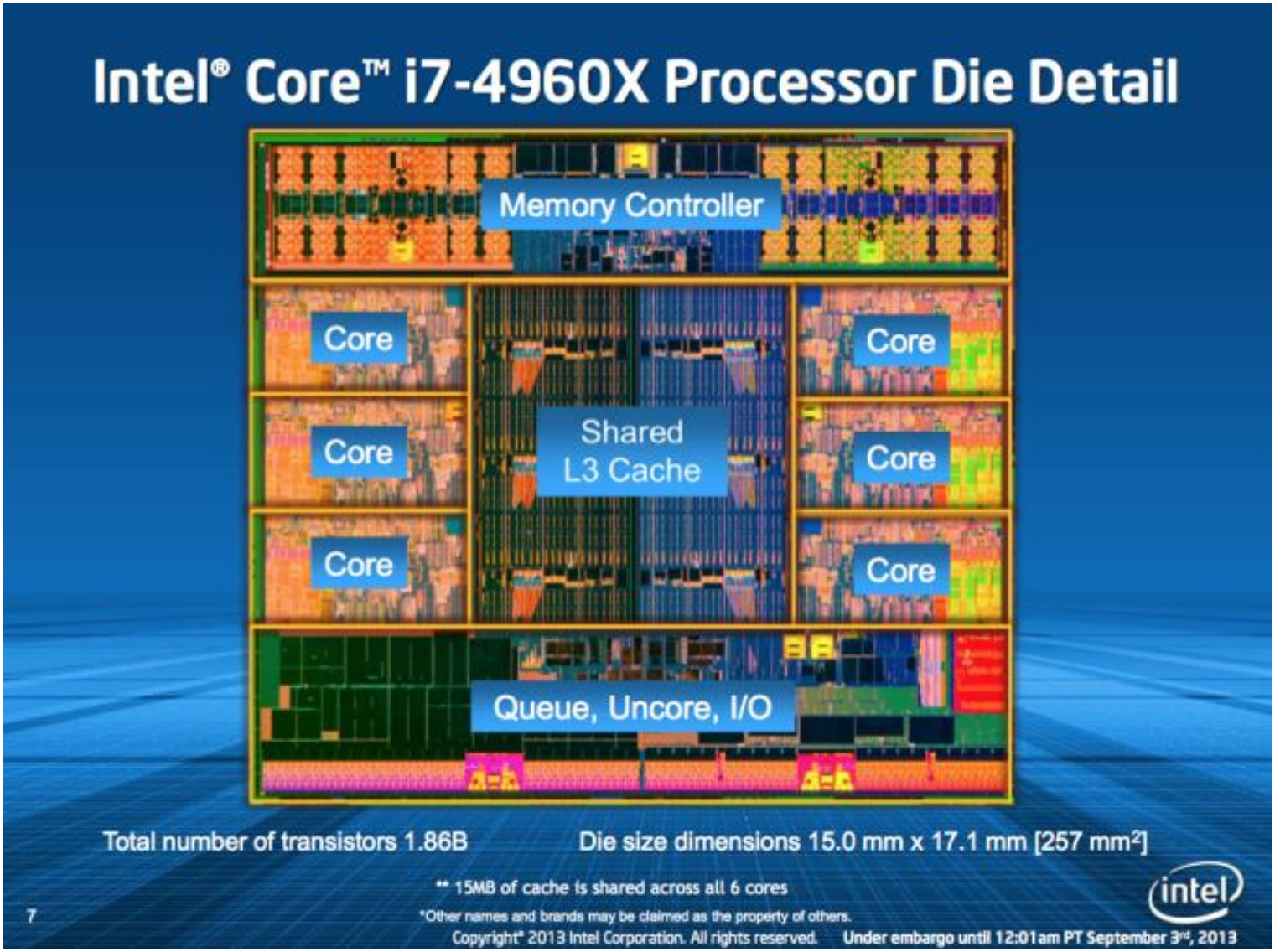


Quelle: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-1-manual.pdf

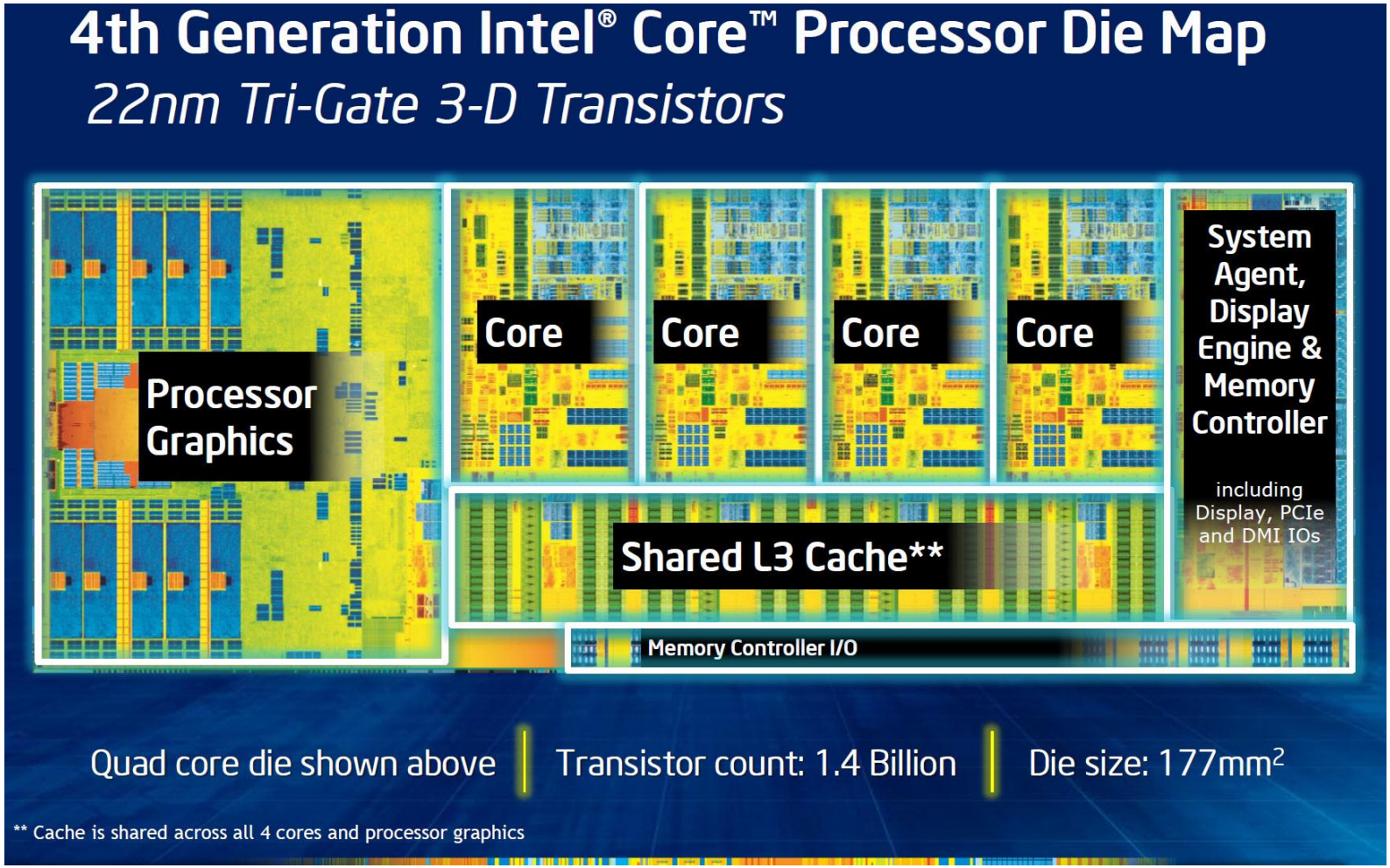
2.3.2.6 Beispiel: Intel Core



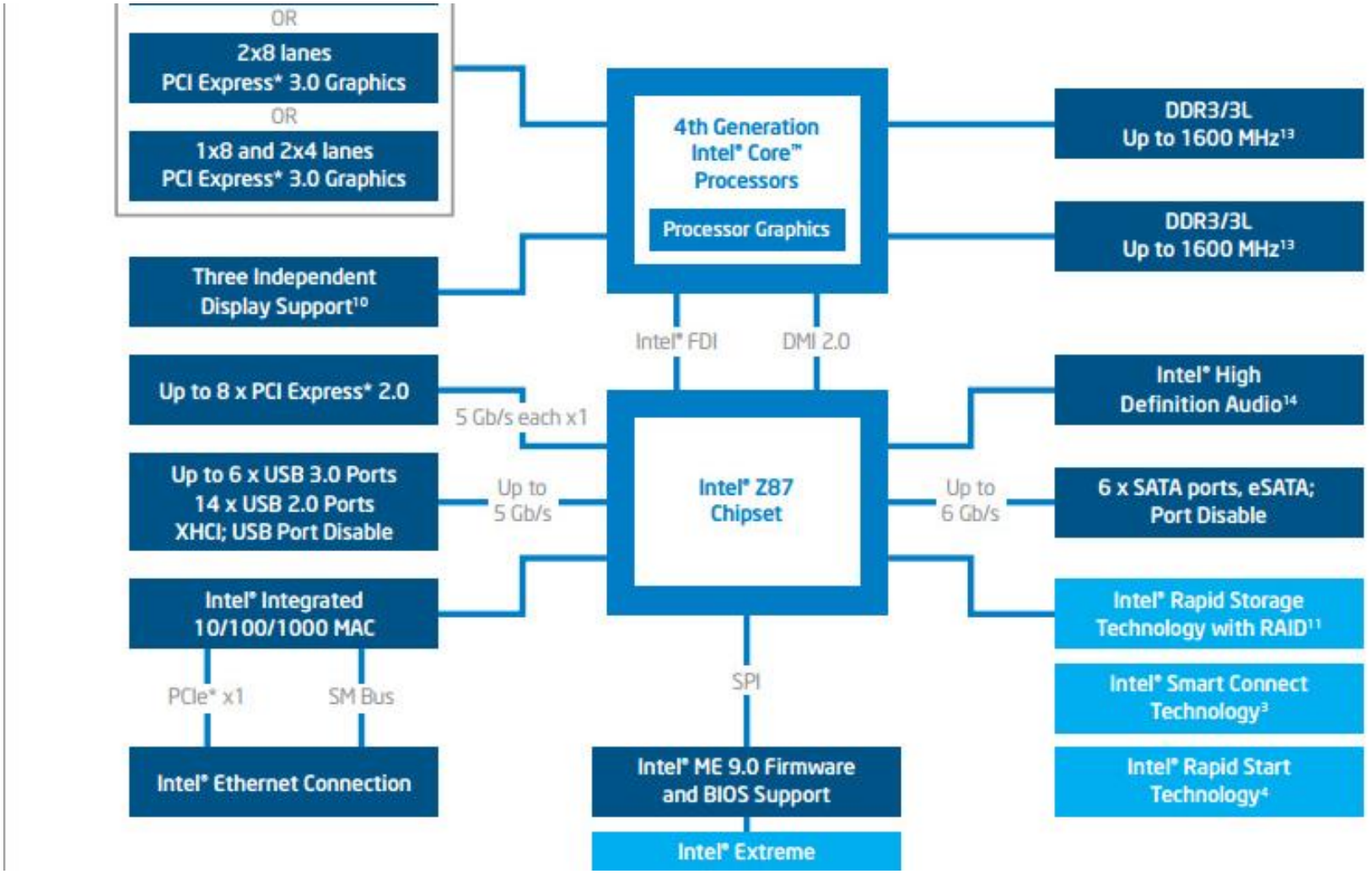
2.3.2.6 Beispiel: Intel Core i7 „Ivy-Bridge“



2.3.2.6 Beispiel: Intel iX Core „Haswell“



2.3.2.6 Beispiel: Haswell Rechnerarchitektur



- Welche Vorteile bringt MIMD?
- Welche Art der Klassifizierung gibts es bei MIMD?
- Wie grenzen sich Superskalar, SIMD, VLIW und MIMD ab?

2.3.2.7 GP Prozessoren und Echtzeit

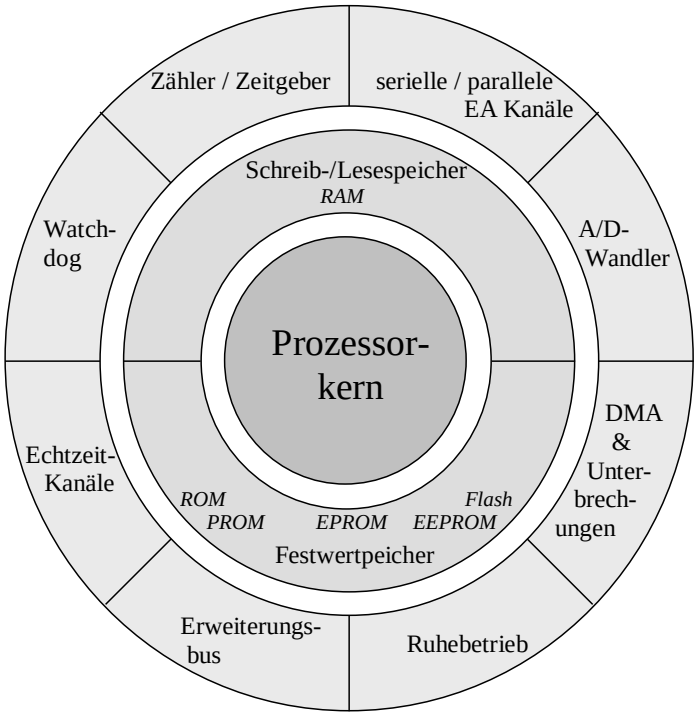
- Ausführungszeiten schlecht vorhersagbar durch dynamische Effekte:
  - Dynamisches Instruktions-Scheduling
  - Hierarchische Speicherorganisation
    - Caches
  - Branch-Prediction Mechanismen
    - Für Echtzeitsysteme mit harten Zeitfristen muß die Ausführungszeit eines Blockes bekannt bzw. beschränkt sein.
    - Insbesondere kann ein Programmstück bei mehreren Ausführungen verschiedene Ausführungszeiten haben abhängig von Eingabedaten
    - Man kann natürlich GP-Prozessoren verwenden, wenn man die Worst-Case Ausführungszeit bestimmen kann:
      - oftmals schwer abzuschätzen
      - nur unter bestimmten Annahmen: Zielarchitektur + Programmpfadanalyse
      - Cinderella-Werkzeug (Sharad Malik, Princeton University, USA)
    - GP Prozessoren sind i.a. ungeeignet für harte Echtzeitanforderungen
  - komplexe I/O- und Speicherinterfaces

Inhalt

- 2.4 Spezialprozessoren
  - **2.4.1 Mikrocontroller (µC)**
  - 2.4.2 Digitale Signalprozessoren (DSPs)
  - 2.4.3 Grafik Prozessoren (GPU)
- 2.5 Bus, Network-on-Chip (NoC) & Multicore
- 2.6 Anwendungs-spezifische Instruktionssatz Prozessoren (ASIP)
- 2.7 Field Programmable Gate Arrays (FPGA)
- 2.8 System-on-Chip (SoC)

2.4.1 Mikrocontroller: Schalenmodell

- prinzipiell kein Unterschied zum Kern eines Mikroprozessors
- Kosten spielen jedoch meist die dominante Rolle
  - einfacher als der Kern eines Mikroprozessors



2.4.1 Mikrocontroller: Prozessorkern - Varianten

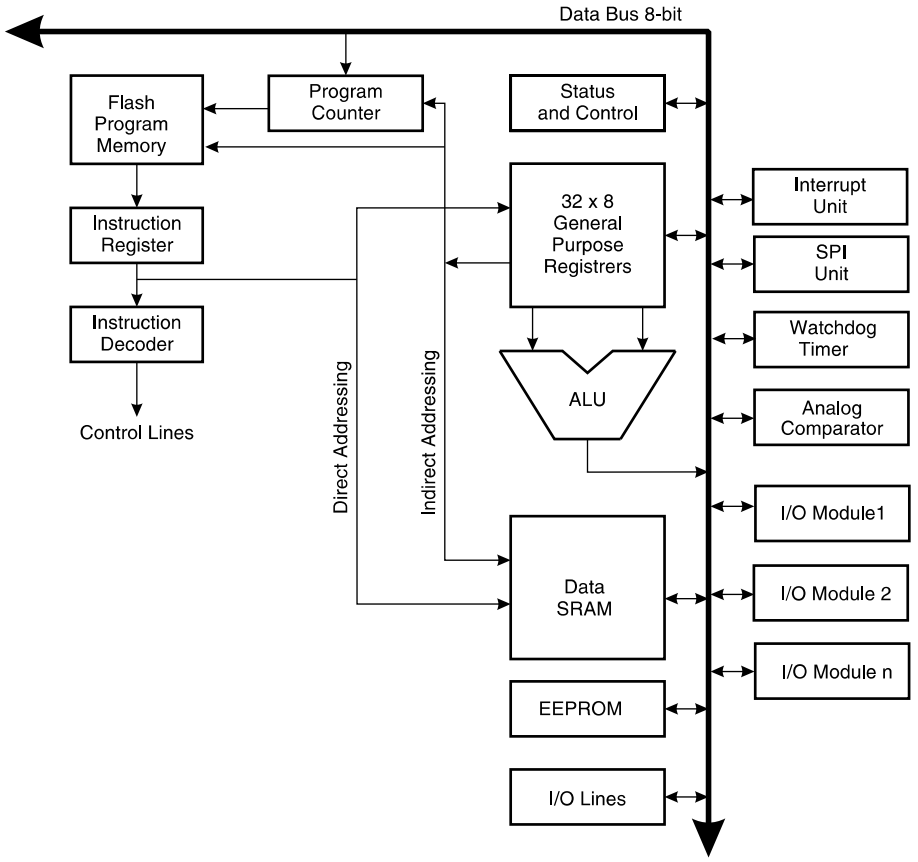
- 1. Eigens für den Mikrocontroller entwickelter einfacher Kern

- 2. Verwendung älterer Kerne von Mikroprozessoren
  - bewährte Technik, Kompatibilität, reduzierte Kosten
  - Leistungsvermögen meist ausreichend
  - Modifikationen:
    - Stromsparmodus
    - kein Cache
    - keine virtuelle Speicherverwaltung

      → Reduktion des Stromverbrauchs, Verbesserung des Echtzeitverhaltens

2.4.1 Mikrocontroller: Peripherie

- Microcontroller sind dafür optimiert
  - viele Bit- und Logikoperationen
  - Register sind oft als RAM realisiert:
    - Kontextwechsel (Contextswitch) durch Pointeroperation, dadurch minimale Interruptlatenz und schneller Kontextwechsel
  - periphere Einheiten integriert
    - Analog Digital Wandler
    - Digital Analog Wandler
    - Kommunikationsinterfaces (CAN, Seriell, SPI etc.)
    - Timer/PWM
    - …



Beispiel ATmega128

2.4.1 Mikrocontroller: Speicher

- In der Anfangszeit eingebetteter Systeme besaßen die Mikrocontroller maskenprogrammierte ROMs

- Mit der Zeit wurden verschiedene Arten von programmierbaren Speichern entwickelt
  - **PROM** (Programmable Read-Only Memory)
    - Einmal programmierbar
  - **EPROM** (Erasable Programmable Read-Only Memory)
    - Programmierbar und mit UV Licht löschbar
  - **EEPROM** (Electrically Erasable Programmable Read-Only Memory)
    - Programmierbar und elektrisch löschbar
  - **Flash-EEPROM**
    - Programmierbar und elektrisch löschbar
    - Deutlich schneller beschreibbar und löschbar als EEPROMs
    - Gelöscht werden ganze Bereiche (Pages)
    - Stand der Technik

      -> Hat heute praktisch alle anderen ROM Technologien verdrängt

2.4.1 Mikrocontroller: Zähler und Zeitgeber

- im Echtzeitbereich ein wichtiges Hilfsmittel
- für eine Vielzahl unterschiedlich komplexer Anwendungen einsetzbar
- Beispiel:
  - Zählen von Ereignissen, Messen von Zeiten kommen mit einem Zähler bzw. Zeitgeber aus
  - Pulsweitenmodulation, Frequenz- oder Drehzahlmessung, Schrittmotorsteuerungen benötigen mehrere Einheiten

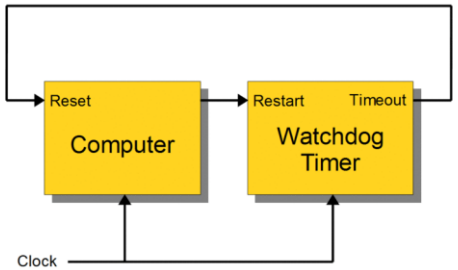
2.4.1 Mikrocontroller: Watchdog

- „Wachhund“ zur Überwachung der Programmaktivitäten eines Mikrocontrollers

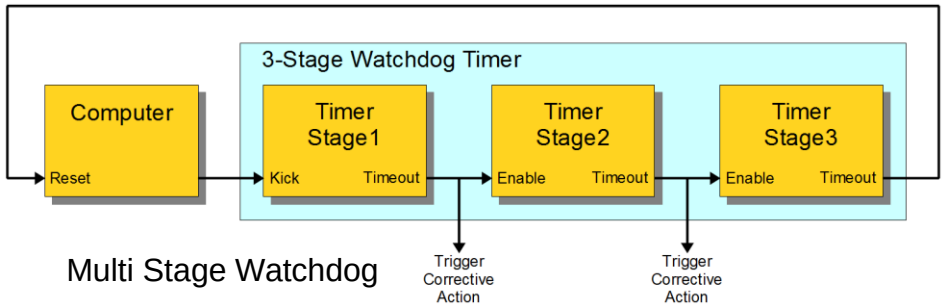
- Programm muss in regelmäßigen Abständen Lebenszeichen liefern
  - Bleiben diese aus, so nimmt der Wachhund einen Fehler im Programmablauf an => Reset

- Hardware-Watchdog
  - Time-Out-Watchdog
  - Fenster-Watchdog
  - Intelligenter Watchdog
- Software-Watchdog

- Beispiel:
  - Mars Sojourner Mission



Single Stage Watchdog



Multi Stage Watchdog

2.4.1 Mikrocontroller: Anwendungen

- Steuerungsdominante Anwendungen
  - kontrollfluss-dominanter Code
    - viele Verzweigungen und Sprünge
  - weniger komplexe arithmetische Operationen
  - geringer Datendurchsatz
  - Multitasking schnelle Kontextwechsel

2.4.1 Mikrocontroller: Low-Cost

- Systeme mit 4/8/16-Bit Prozessoren
- Codegröße dominiert die Chipfläche und damit die Kosten
- Performanzanforderungen oftmals gering
- optimiert auf bestimmte Anwendungsgebiete
  - Industriesteuerungen, Regelungstechnik, Bedienschnittstellen, …

2.4.1 Mikrocontroller: Übersicht

- **Microchip – PIC**
  - 8-Bit, relativ alt aber stark verbreitet
- **Intel – 8051**
  - 8-Bit, relativ alt aber stark verbreitet
  - An viele Halbleiterhersteller lizenziert (z.B. Atmel)
- **Atmel – AVR**
  - 8-Bit, neuere RISC Architektur
- **Texas Instruments – MSP430**
  - 16-Bit, ultra low power
- **ARM Limited – ARM**
  - 32-Bit, verschiedene Untertypen verfügbar: z.B. ARM7TDMI
  - An viele Halbleiterhersteller lizenziert (z.B. ST Microelectronics, NXP)
- Und viele mehr…

- Generell existieren von jeder Familie eine Vielzahl an Varianten mit unterschiedlicher Speicher- und Peripherieausstattung

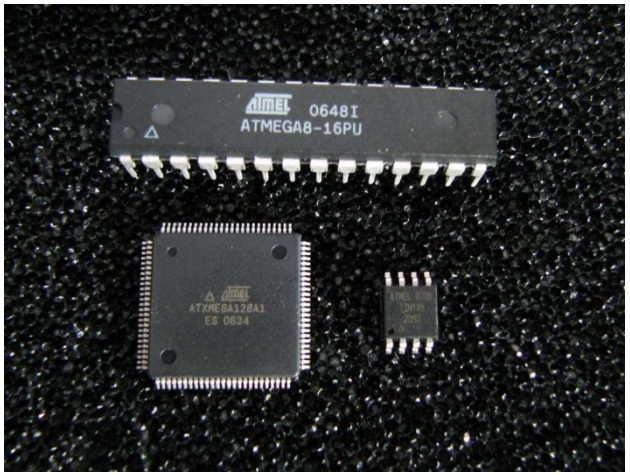
2.4.1 µC Beispiel: Intel MCS-51

- bekannter 8-Bit Microcontroller, ursprünglich 1980 von Intel für eingebettete Systeme entwickelt (MCS-51)
- sehr viele Varianten und Clones von verschiedenen Herstellern
  - Sehr verbreitet in den 1980er und frühen 1990er Jahren
  - AMD, Atmel, Dallas, Matra, Philips, Siemens etc.
- Ursprünglich für NMOS entwickelt
  - Low-Power Weiterentwicklung für CMOS (z.B. 80C51)
- Befehlssatz auf 1-Bit Operationen optimiert
- Harvard Architektur

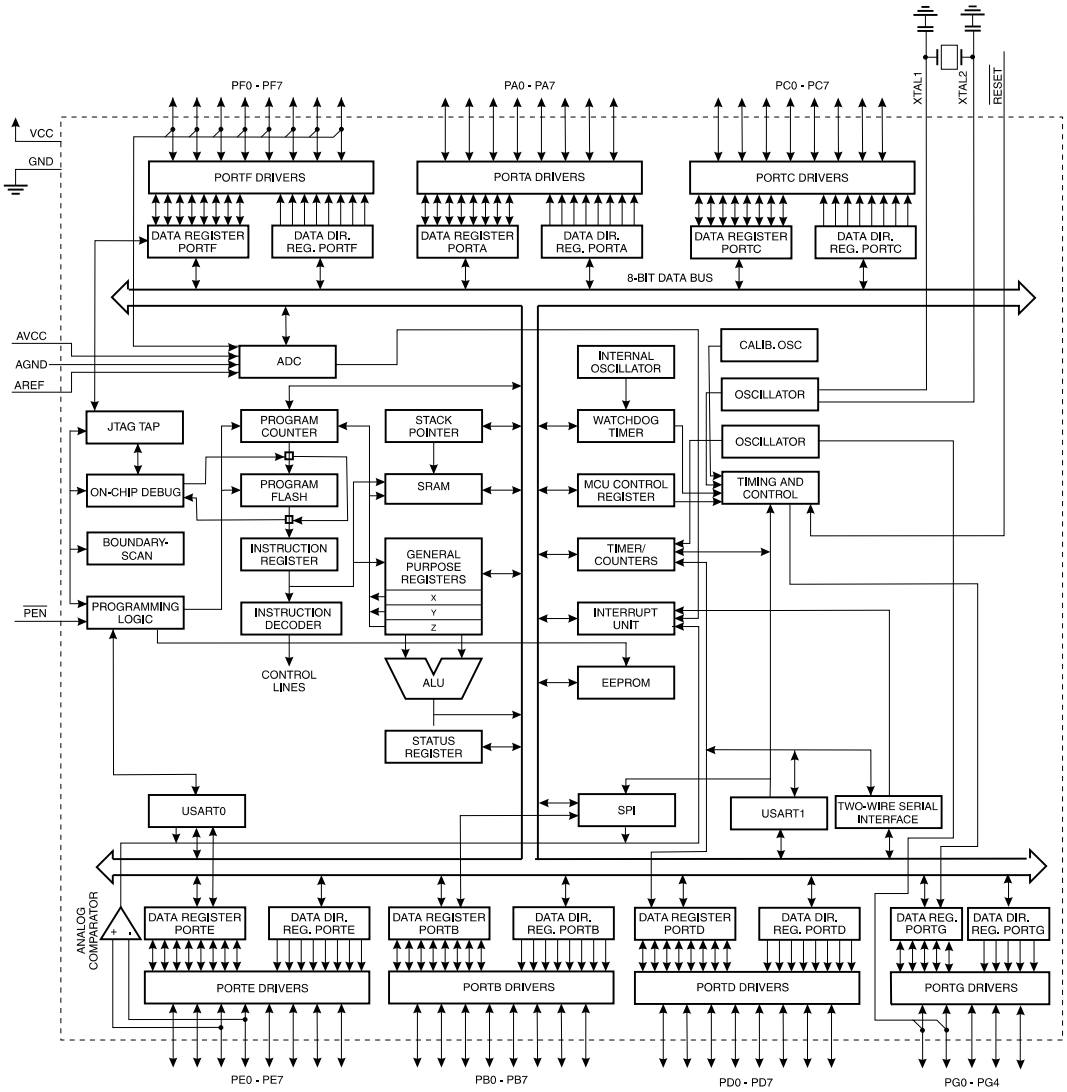
2.4.1 µC Beispiel: Atmel AVR

- 8-Bit RISC Architektur (hochsprachenoptimiertes Hardwaredesign)
- 16-20 MHz max. Taktfrequenz
  - Viele Befehle benötigen nur einen oder zwei Takte
- Unterscheidung in Tiny & Mega Serie
  - **Tiny**: Kleinere Controller bis 16K Programmspeicher, 512B RAM, 28 IO Pins
  - **Mega**: Größere Controller bis 256K Programmspeicher, 4K RAM, 86 IO Pins, Hardware Multiplizierer, AD Wandler
  - Spezielle Varianten mit Unterstützung für CAN, LIN, USB, Ethernet, usw.
- Einsatz bei einfachen Steueraufgaben
- Viele Typen als DIP verfügbar
- GCC und Hersteller Tools frei verfügbar

- Weitere Informationen:
  http://www.mikrocontroller.net/articles/AVR



2.4.1 µC Beispiel: Atmel ATmega 128



Oliver Sander
Hardware/Software Co-Design

2.4.1 µC Beispiel: MSP430

- 16-Bit RISC Architektur
- Optimiert für extrem geringe Leistungsaufnahme
- 16 MHz max. Taktfrequenz
  - Befehle benötigen einen bis sechs Takte
- Unter anderem mit Hardware Multiplizierer, DA und AD Wandler verfügbar und somit für einfache DSP Aufgaben geeignet
- Einsatz für Steueraufgaben vor allen in Bereichen mit sehr begrenzten Energievorräten (z.B. Armbanduhr,)
- GCC frei verfügbar

- Weitere Informationen:
  http://www.mikrocontroller.net/articles/MSP430

2.4.1 Mikrocontroller – High Performance

- Systeme mit 16-/32-/64-Bit Prozessoren
- Beispiel: Motorola MC683xx, Intel x196, Siemens x166

- Einsatzgebiete:
  - Systeme mit steuerungsdominanten Teilen und zusätzlich
    - hohen Datenraten (Telekommunikation, Automobiltechnik)
    - hohen Berechnungsanforderungen (Regelungstechnik, Signalverarbeitung)
  - Microcontroller-Core als Teil eines Systems-on-Chip (SoC)
    - ARM Cortex M3 auf Actel Smart Fusion Board
    - ARM Cortex A9 auf Xilinx Zynq

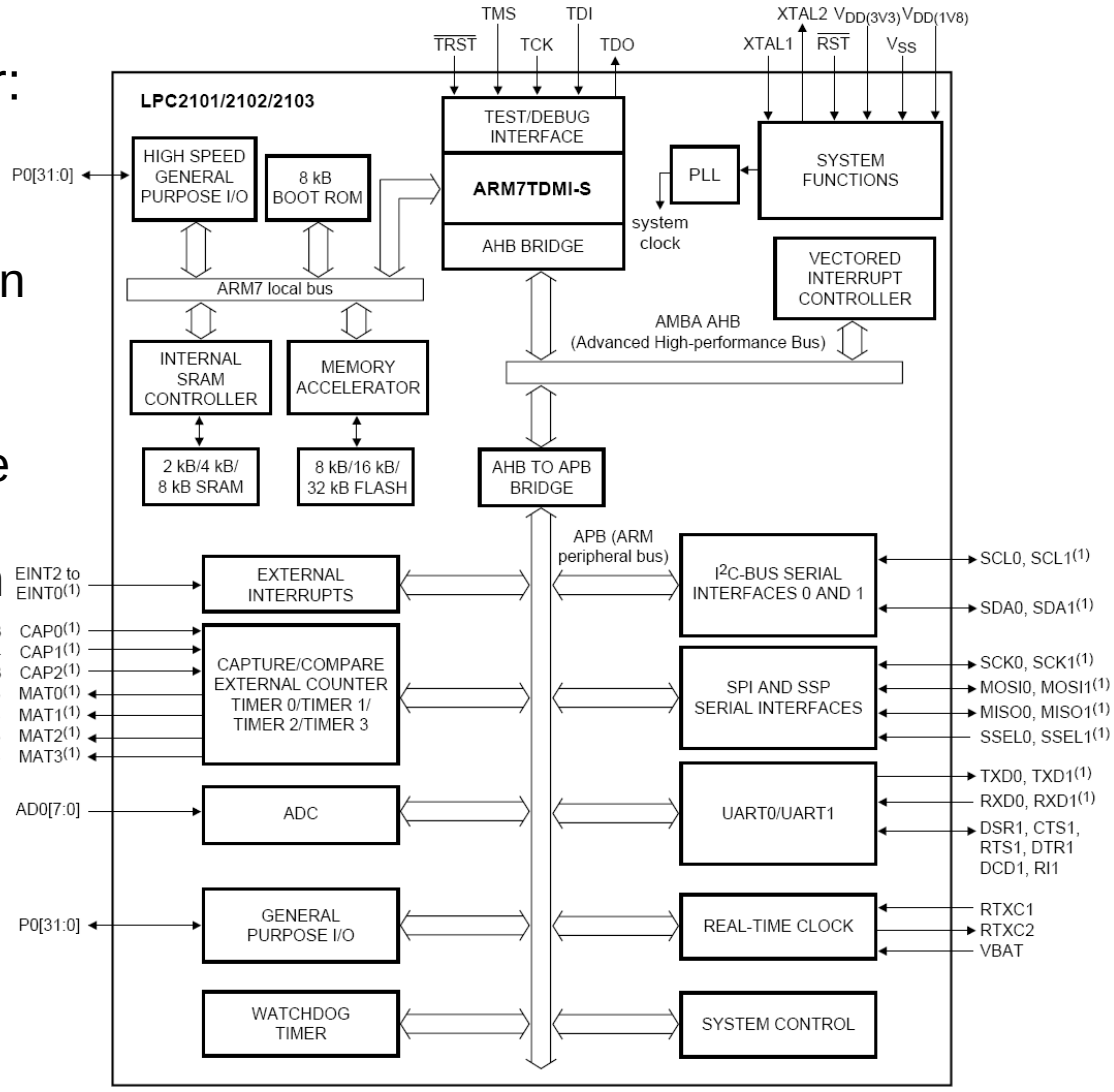
2.4.1 µC Beispiel: ARM7 TDMI

- 32-Bit RISC Architektur
- Prozessorkern wird von verschiedenen Herstellern mit Peripherie erweitert und vertrieben (z.B. ARM7TDMI-S im LPC21xx von NXP)
- 70 MHz max. Taktfrequenz
- Auch komplexere Peripherie wie USB, Ethernet, TFT Controller verfügbar
- Einsatz auch für rechenintensivere Aufgaben
- Durch hohe Stückzahlen fast so günstig wie 8-Bit Controller
- „Der neue 8051“
- GCC frei verfügbar

- Weitere Informationen:
  http://www.mikrocontroller.net/articles/ARM

2.4.1 µC Beispiel: ARM7 TDMI

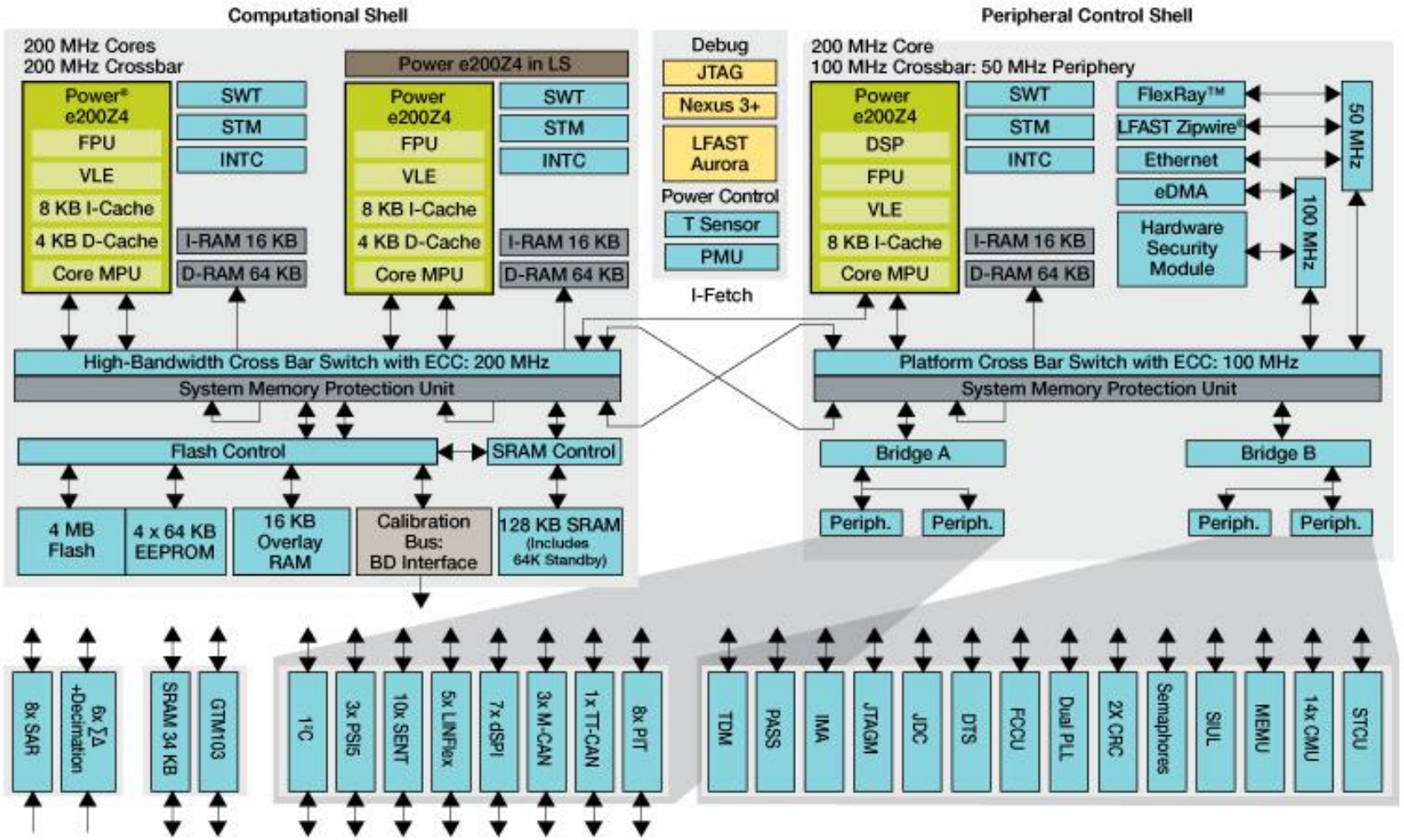
- Typisch für Mikrocontroller:
  - Es existiert eine Basis-Architektur
  - Darauf aufbauend werden viele verschiedene Untertypen angeboten
  - Jeder besitzt eine andere Kombination von Peripherie-Komponenten bzw. RAM & ROM Größen
  - Oft sind Pins mehrfach belegt



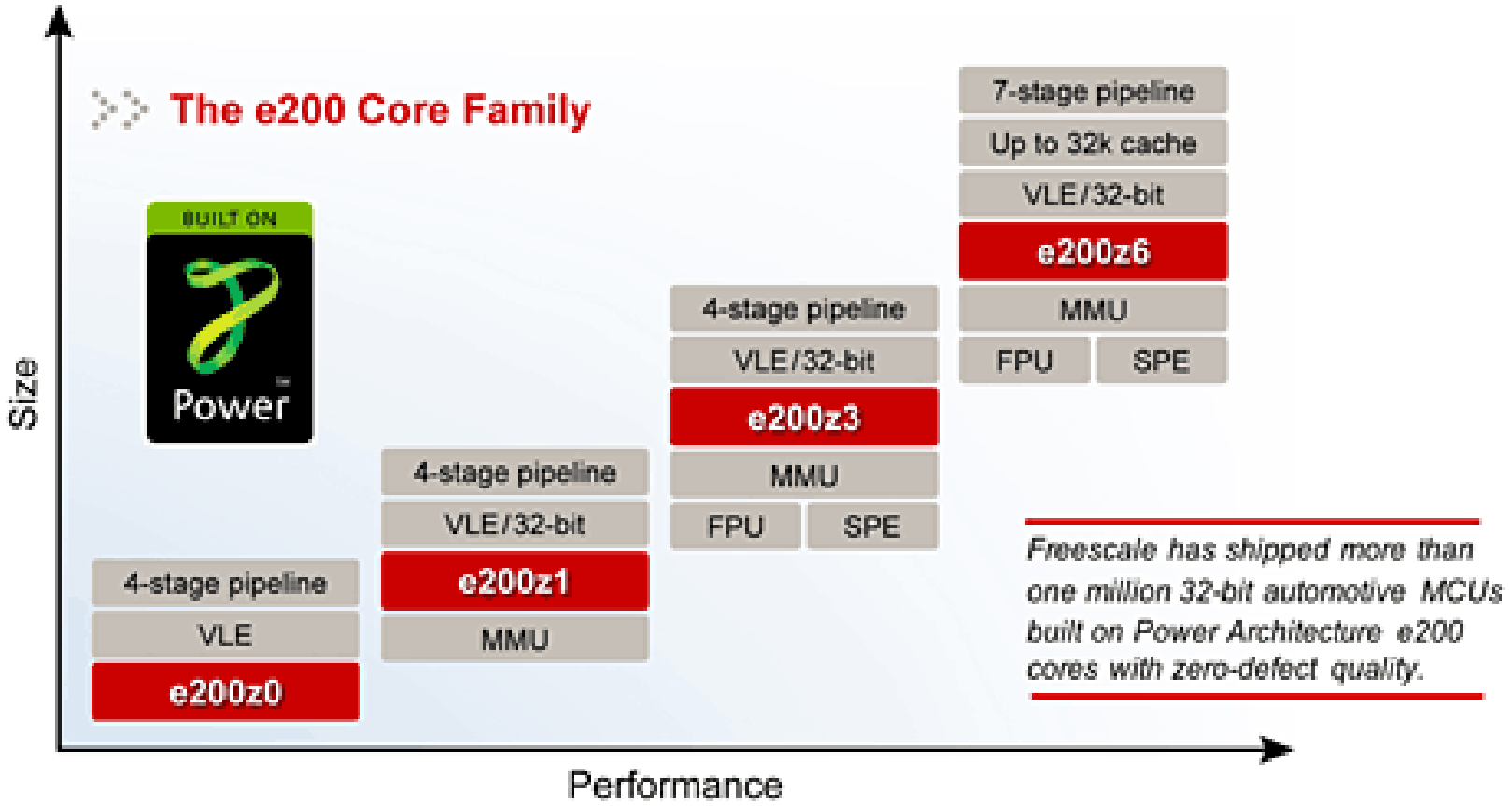
Oliver Sander
Hardware/Software Co-Design

Beispiel: Controller für Motorsteuergeräte

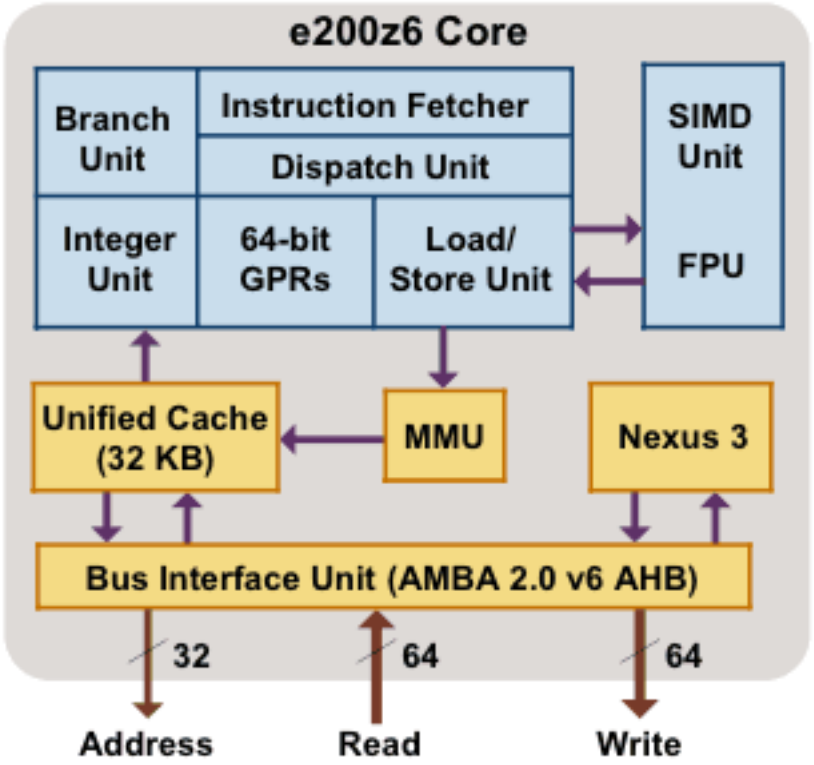
Qorivva MPC5746M Block Diagram



E200 Processor Familie



E200z6 Core



- Was ist ein Mikrocontroller?
- Gib Beispiele für 8/16/32/64 bit Mikrocontroller
- Warum ist der 8051 ein 8 bit Mikrocontroller?
- Was ist der Unterschied zwischen einem Mikroprozessor und einem Mikrocontroller?

Inhalt

- 2.4 Spezialprozessoren
  - 2.4.1 Mikrocontroller (µC)
  - **2.4.2 Digitale Signalprozessoren (DSPs)**
  - 2.4.3 Grafik Prozessoren (GPU)
- 2.5 Bus, Network-on-Chip (NoC) & Multicore
- 2.6 Anwendungs-spezifische Instruktionssatz Prozessoren (ASIP)
- 2.7 Field Programmable Gate Arrays (FPGA)
- 2.8 System-on-Chip (SoC)

2.4.2 Digitale Signal Prozessoren (DSP)

- Typischerweise viele mathematische Operationen, die schnell ausgeführt werden sollen

- Signale für Audio/Video Anwendungen
  - Analog Digital Wandlung
  - Digitale Signal Verarbeitung
    - Digitaler-Signal-Prozessor (DSP)
  - Digital Analog Wandlung

- Architektur spezialisiert auf digitale Signalverarbeitung
  - Zusätzliche zur Unterstützung von Mikrocontoller-Features

2.4.2 DSP: Signalverarbeitungs-Anwendungen

- Datenfluß-dominanter Code
- viele arithmetische Operationen
  - vorwiegend Multiplikationen und Additionen
- wenig Sprünge und Verzweigungen,
  - sehr gut vorhersagbare Sprungziele
- oftmals große Parallelisierung möglich
- regelmäßige Operationen auf mehrdimensionalen Datenfeldern
  - mehrfach geschachtelte Schleifen
  - regelmäßigen Datenabhängigkeiten (Indexfkt. der Variablen!)
- sehr große Datenmengen, hoher Datendurchsatz
- spezialisierte Peripherie für DSP-Umgebung

2.4.2 DSP: Datenpfad- / Architektur-Optimierung (I)

- Harvard-Architektur, Mehrfachzugriffe auf Operanden
  - MAC-Instruktion: Zugriff auf Instruktion und zwei Operanden in einem Zyklus
    - Speicherarchitektur mit Mehrfachzugriff notwendig
  - Architektur muss für Instruktionen und Daten getrennte Busse haben
    - Harvard-Architektur: mehrere Datenbusse für mehrfachen Operandenzugriff
  - Erste DSPs: getrennte externe Busse für Instruktionen und Daten.
  - Moderne DSPs: nur interne Harvard-Architektur, aber extern oft zwei identische Speicherschnittstellen, über die gleichzeitig auf verschiedene Off- Chip Speicherbausteine zugegriffen wird (heute oft auch On-Chip!).

2.4.2 DSP: Datenpfad- / Architektur-Optimierung (II)

- Parallel-Instruktionen (MAC - Multiply & ACcumulate)
  - MAC-Operation ist oft benötigte Operatorverkettung
    - d.h., in einem Befehlszyklus werden 2 Operanden multipliziert und das Resultat in einem Register akkumuliert (Standardprozessor braucht typischerweise 10 Zyklen)
  - Multiplikationsdatenpfad optimiert in Hardware realisiert
    - DSP-Architekturen hatten erstmals Hardware-Multiplizierer - auch für Gleitkomma

```
sum = 0.0;

for (i=0; i<N; i++)

sum = sum + a[i]*b[i];
```

2.4.2 DSP: Datenpfad- / Architektur-Optimierung (III)

- Zero-Overhead Schleifen
  - Schleifen mit bekannter Anzahl von Durchläufen enthalten speziellen Zähler
    - wird mit jedem Durchlauf dekrementiert und mit 0 verglichen,
    - ist Schleifenzähler = 0: nächsten Befehl ausführen (nach Schleife),
      sonst: an Schleifenanfang zurückspringen
  - DSPs besitzen hierzu Spezial-Register zur Hardwareunterstützung
    - mit Anfangs- und End-Adresse der Schleife sowie dem Zähler geladen
    - Während Schleifendurchlauf: Zähler wird parallel dekrementiert und die Adresse der entsprechenden Instruktion (Zurückspringen oder nicht!) gesetzt
  - Dadurch: keine Takte für die Schleifensteuerung notwendig (zero overhead)

2.4.2 DSP: Datenpfad- / Architektur-Optimierung (IV)

- spezielle DSP-Adressierungsarten (circular, bit-revers)
  - Adressgeneratoren arbeiten parallel zur eigentlichen Instruktionsabarbeitung
  - Einsparung von Prozessorzyklen für die Adressberechnung
    - Beispiel: verschiedene Formen von Autoinkrement/Autodekrement einer Adresse, um eine programmierbare Schrittweite
  - weitere wesentliche Adressierungsarten: circular-Adressierung (Modulo-M z.B. für Filter) und die bitrevers-Adressierung (z.B. für FFT).

2.4.2 DSP: Multiply and Accumulate (I)

```
sum = 0.0;
for (i=0; i<N; i++)
sum = sum + a[i]*b[i];
```

LDF: Load Float- value

Adressen: a[i], b[i] mit Incr. ++

Zero-Overhead Schleife

Wiederhole nächste Instruktion N-1 mal.

```
LDF 0, R0
LDF 0, R1
RPTS N-1
MPYF3 *(AR0)++, *(AR1)++, R0
|| ADDF3 R0, R1, R1
ADDF3 R0, R1, R1
```

Zwischenprodukt

MAC – Instruktion

|| bedeutet parallele Ausführung von MPYF3 und ADDF3

(vgl. Pipelinestruktur).

Endsumme

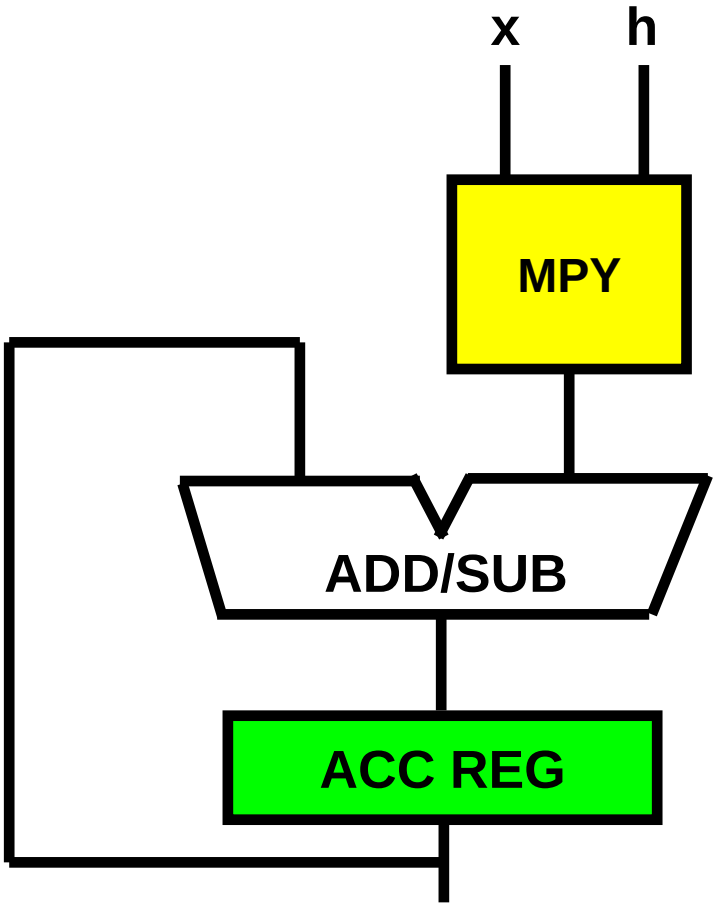
TMS320C3x Assembler

2.4.2 DSP: Multiply and Accumulate (II)

Faltungssumme für FIR-System

$$y(n) = \sum_{0}^{N-1} h(m) \cdot x(n-m)$$

Basiselement zu Berechnung der endlichen Impulsantwort:
-> nicht-optimierte Variante, kein Pipelining im Vgl. zum TMS320C3x



2.4.2 DSP: Wichtige Eigenschaften

- Eingeschränkte Parallelität
  - Viele DSPs haben nur 1-2 MAC-Einheiten
    - jedoch: in einem Takt Zuweisungen zu mehreren Registern
    - diese geringfügige Parallelität ist im Befehlssatz sichtbar:
      - Beispiel: Parallel-Moves veranlassen gleichzeitig Arithmetik-Operation und Datentransport
- Heterogene Register [sätze]
  - mehrere verschiedene Registersätze (Zugriff nur durch spezielle Funktionseinheiten)
  - Standardprozessoren verwenden dagegen meist nur einen General-Purpose-Registersatz
- Echtzeitfähigkeit
  - Oft ist Angabe der maximalen Laufzeit (nicht das Mittel) wichtig.
    - häufiger Verzicht auf Caches mit datenabhängigen Laufzeiten
  - Immer mehr verbreitet: mehrere Datenpfade und VLIW-Architekturen
- i. Allg. mehr Rechenleistung pro Watt als bei Standardprozessoren
  - wichtig bei portablen Geräten

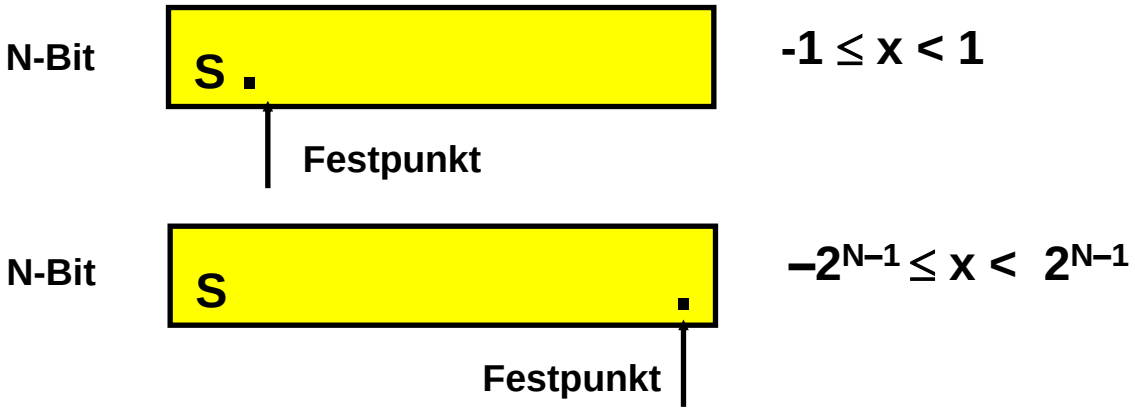
2.4.2 DSP: Zahlenformate

- Allgemein:
  - Mantisse bestimmt die Genauigkeit
  - Exponent bestimmt die Dynamik
- Fixed Point (FX)
  - gleiche Mantissenbreite: kleiner (billiger) + schneller als Floating-Point
  - Entwurf von manchen Anwendungen durch Rundungs- und Skalierungsprobleme erschwert
  - Für viele DSP-Anwendungen ausreichend
- Floating Point (FP)
  - Große Dynamik des Zahlenbereiches
  - einfacherer Entwurf von Anwendungen
  - Aber: teure Hardware (komplex, große Fläche)

2.4.2 DSP: Arithmetik / Genauigkeit (I)

- DSPs mit FP–Unit kosten 2-4 mal soviel als DSPs mit nur FX-Units
  - FP-DSPs sind oftmals langsamer als reine FX-DSPs

- DSP-Programmierer skalieren feste Zahlenbereiche im Code
  - SW-Libraries
  - getrennter expliziter Exponent -> Blocked Floating-Point

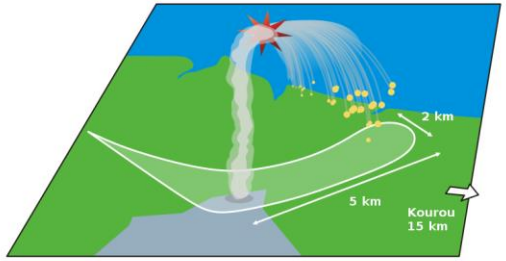


2.4.2 DSP: Arithmetik / Genauigkeit (II)

- Blocked Floating-Point: einzelner Exponent für Gruppe von Zahlen
  - Exponent wird in extra Variable gespeichert
    - für entsprechende Gruppe von Zahlen verfügbar
    - Hardware stellt ggf. besonderes Register bereit
  - Nach Operation müssen die Ergebnisse entsprechend diesem Exponenten durch Schiebeoperationen wieder angepasst werden

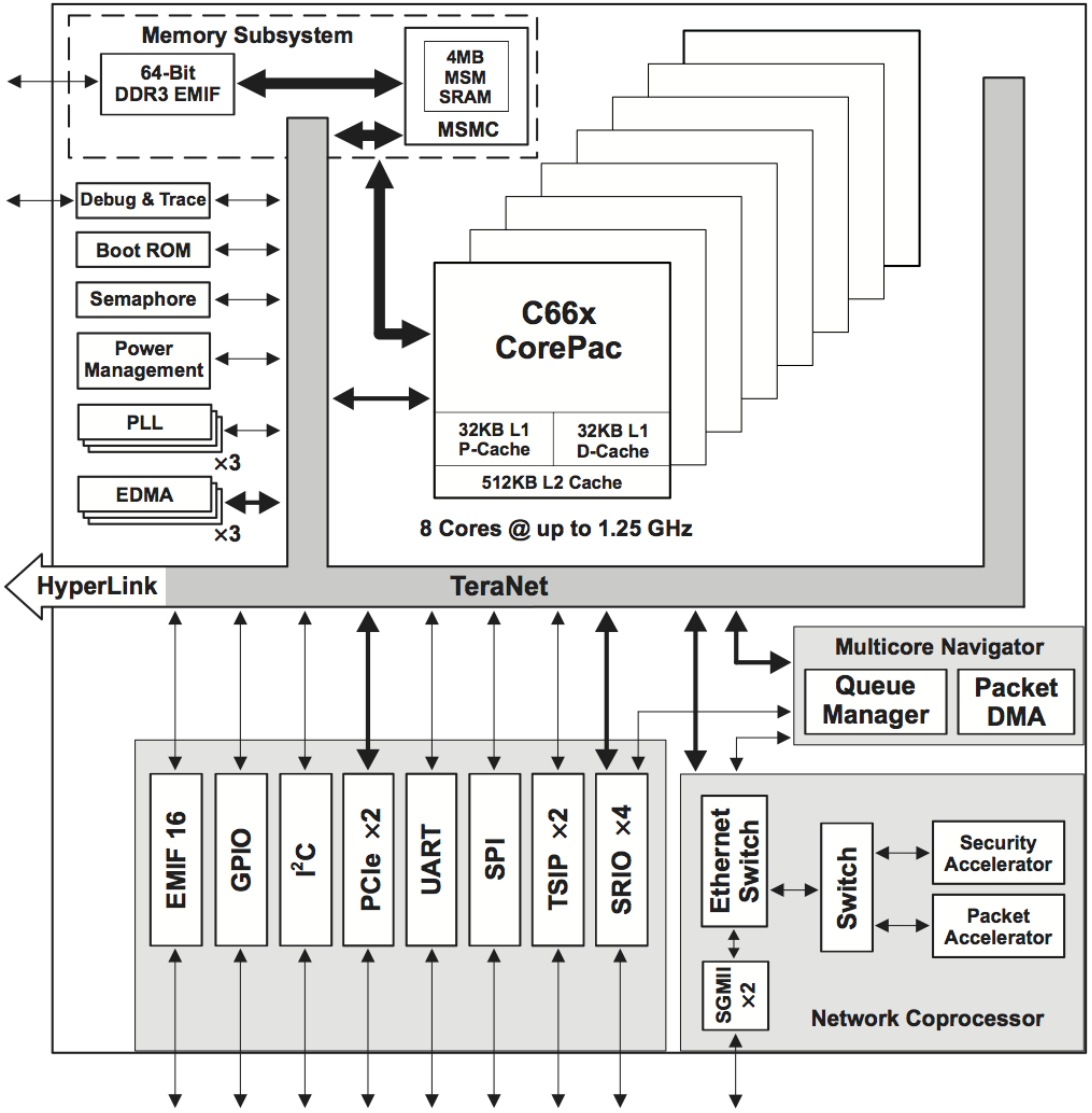
- DSPs stammen eigentlich von analogen Signalprozessoren (ASP)
  - Verwenden aber eine endliche Modulo-Arithmetik

- Begrenzung bei Arithmetiküberläufen: Sättigungs-Arithmetik
  - setze auf größte positive Zahl ($2^{N-1}-1$), oder auf kleinste negative Zahl (-2^{N-1}) → “saturation” (Sättigung)
  - viele Algorithmen werden unter Verwendung dieses Modells entwickelt



2.4.2 DSP Beispiel: TI TMS320C6678 (I)

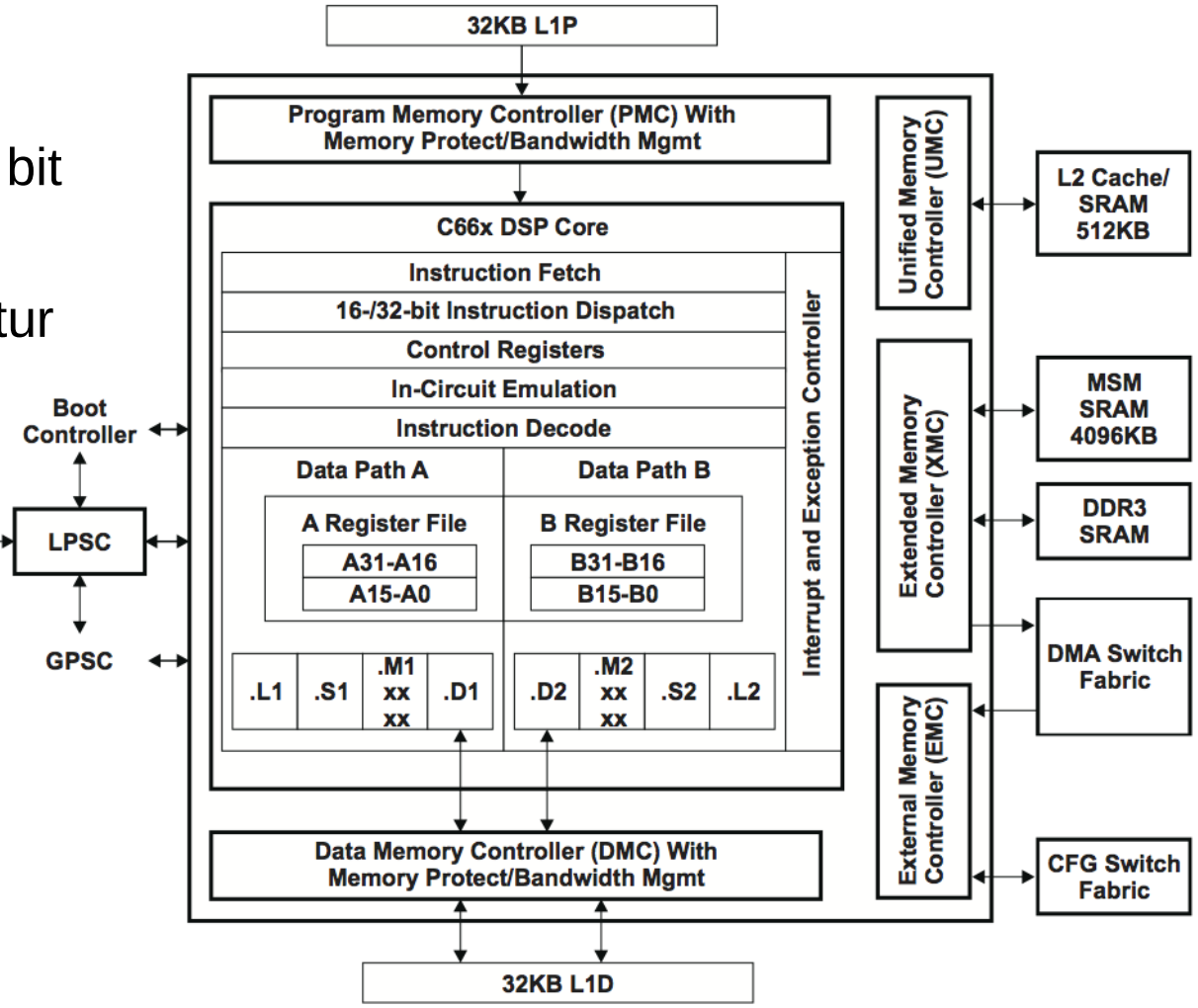
- High-End 8 Core-DSP von Texas Instruments
- Fest- und Gleitkomma-arithmetik
- 1 GHz Taktfrequenz
- 4 MB Shared Memory
- Bis zu 8 GB DDR3 adressierbar
- Umfangreiche Peripherie



Oliver Sander
Hardware/Software Co-Design

2.4.2 DSP Beispiel: TI TMS320C6678 (III)

- 2 Datenpfade
- Je 32 Register mit 32 bit
- 8-fach VLIW-Architektur
- Harvard-Architektur der L1-Speicher
- L1P, L1D und L2 als Cache konfigurierbar



2.4.2 DSP: Programmentwicklung

- Assembler, Compiler
  - Viele DSP-Eigenschaften bisher nur in Assembler nutzbar
  - Programmierung in C
    - Profiling notwendig: Monitoring ausgewählter Programmteile
    - zeitkritische Teile in Assembler programmieren
- Bibliotheken
  - Programmierung in C
    - Aufruf von handoptimierten Bibliotheksfunktionen
  - Beispiele: Bibliotheken für Bildverarbeitung, Sprachverarbeitung
- Codegeneratoren
  - Spezielle Entwicklungsumgebungen
    - Modellierung, Simulation + Codegenerierung
    - Beispiele: Matlab, Synopsys COSSAP, SPW

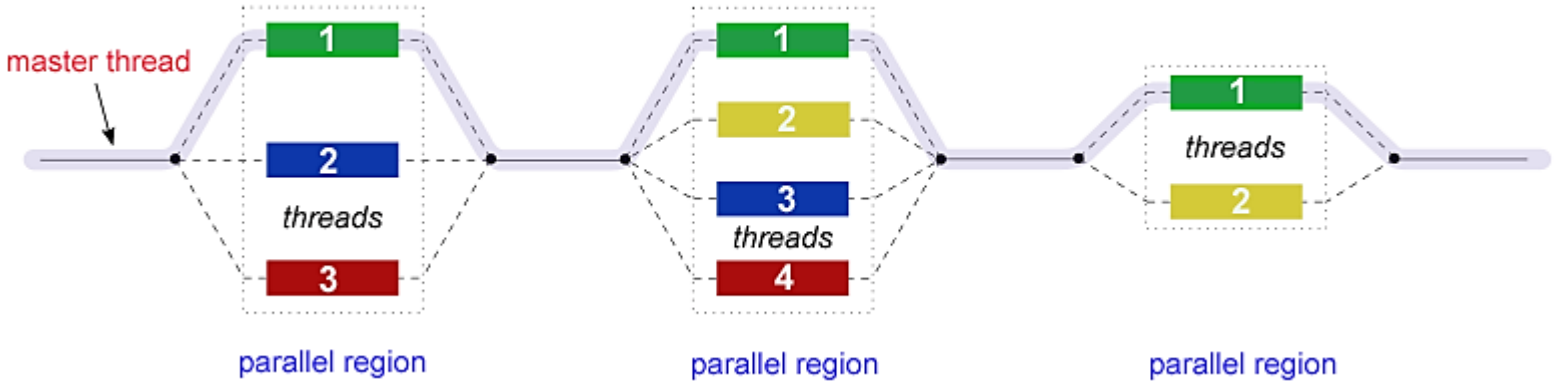
2.4.2 DSP Beispiel: TI Code Composer Studio (CCS)

- Integrated Development Environment (IDE) für Texas Instruments embedded Prozessoren

- Tool Suite für Entwicklung und Debugging
  - Quellcode Editor, Projekt-Umgebung, Debugger, Profiler und Simulator für jeden TI Prozessor
  - SYS/BIOS
    - Verwaltung von MultiCore Systemen
    - Real-time scheduling, Synchronisation und Instrumentierung
    - Preemptive Multitasking, Hardware-Abstraktion, Memory Management
    - Unterstützt OpenMP
  - System Analyzer
    - Real-time Visualisierung für Performanz und Verhalten von Applikationscode
    - Analisiert Information von Software und Hardware
    - Benchmarkung, CPU & task load monitoring, OS execution monitoring & MultiCore event correlation

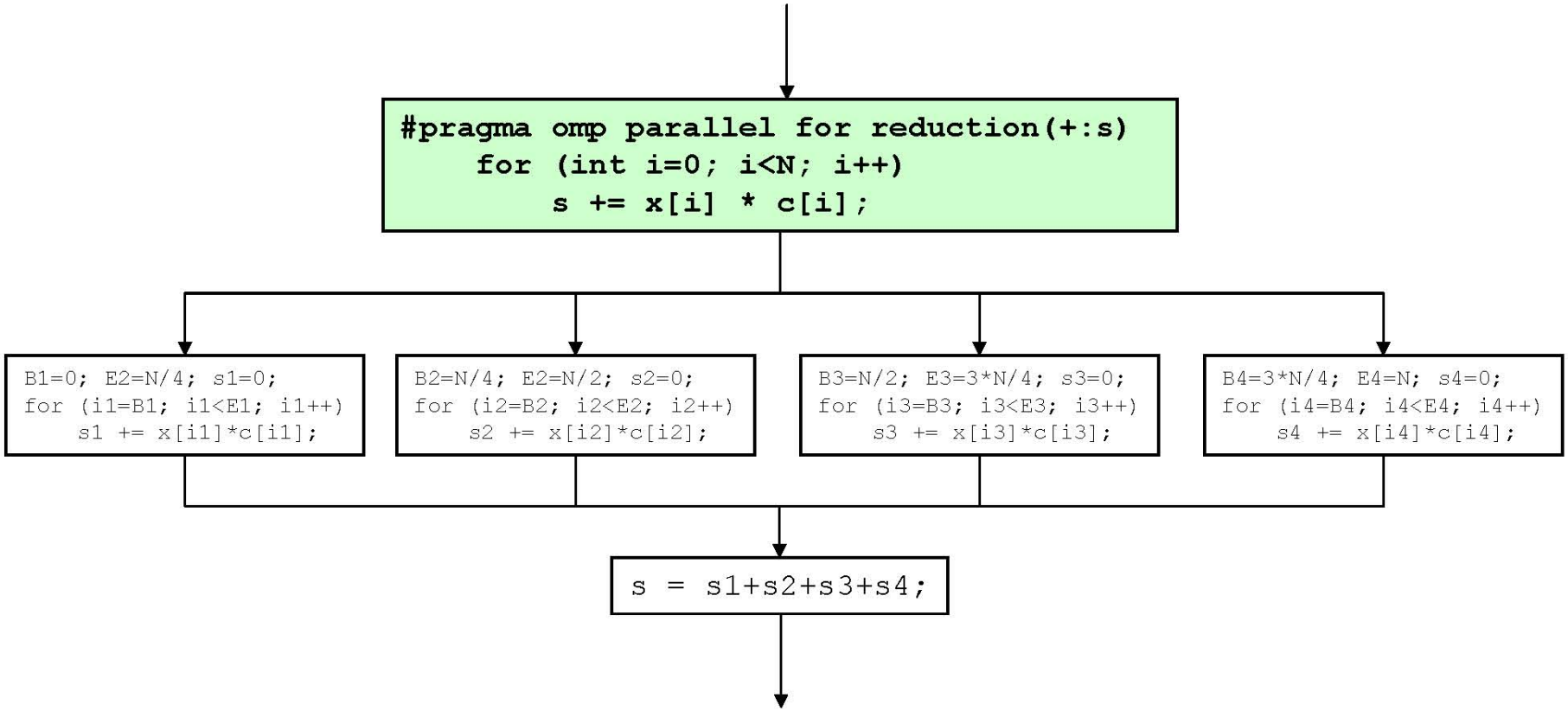
2.4.2.1 OpenMP Einführung

- API zur Programmierung von Multiprozessor-Systemen mit Shared Memory
- Programmierung in C, C++ und Fortran möglich
- Bestehend aus Compilerdirektiven, Bibliotheksfunktionen und Umgebungsvariablen
- Parallelisierung auf Thread- bzw. Schleifenebene
- Programmablauf nach fork/join-Modell:



2.4.2.1 OpenMP Beispiel

- Parallele For-Schleife mit 4 Threads:
  - Beispiel für *data parallelism*



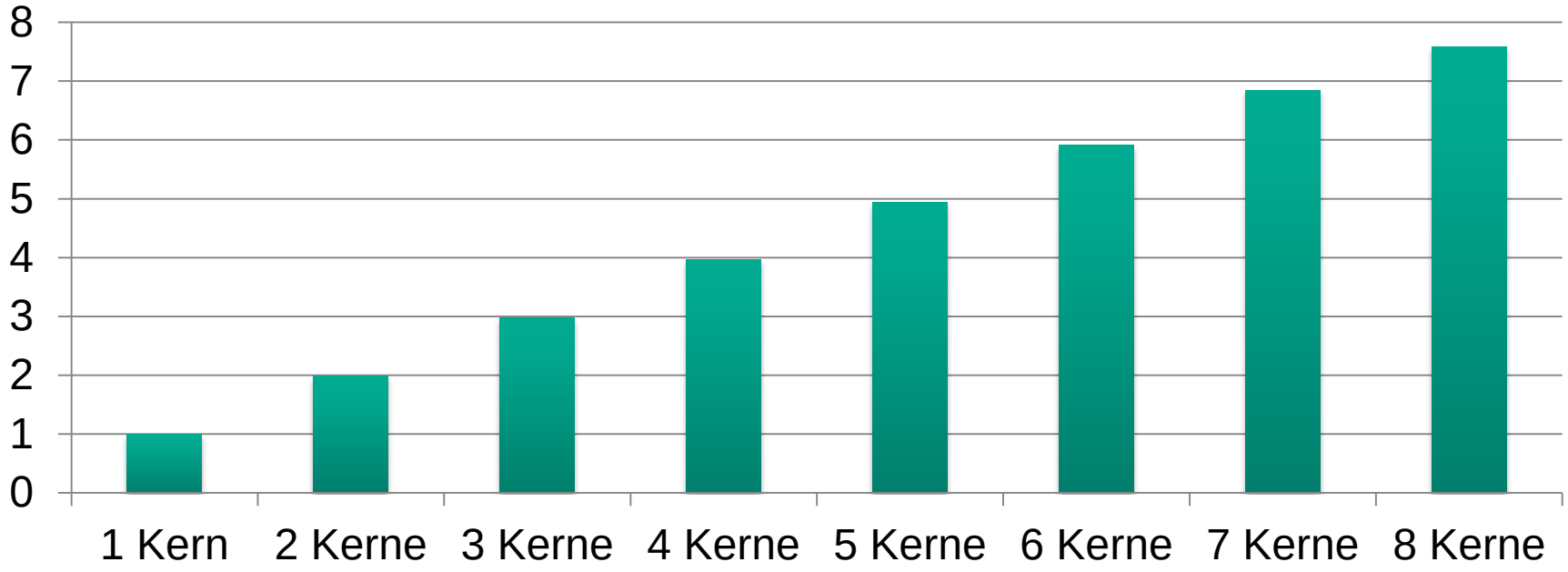
2.4.2.1 OpenMP

- Vorteile:
  - Erweiterung der C-Programmierbarkeit
  - Beschleunigter Softwareentwurf
  - Hohe Portierbarkeit, da standardisiert und plattformunabhängig
  - Wachsende Zahl an unterstützenden Compilern
  - Threadhandling läuft automatisch ab

- Nachteile:
  - Datenabhängigkeiten werden nicht erkannt
  - Synchronisation der Threads ggfs. nötig

  - Auf TI DSPs nur in Verbindung mit SYS/BIOS lauffähig
    - Großer Overhead, Performanzverlust

2.4.2.1 Parallelisierung einer Faltung



- Zielarchitektur TI TMS320C6678:
  - Unterstützung von Gleit- und Festkommaformaten
  - Begrenzter Speedup durch Overhead für Threadhandling und limitierte Speicherbandbreite
    - Shared Memory als Bottleneck

Tradowsky, Carsten, et al. "A New Approach to Model-Based Development for Audio Signal Processing." *Audio Engineering Society Convention 134*. Audio Engineering Society, 2013.

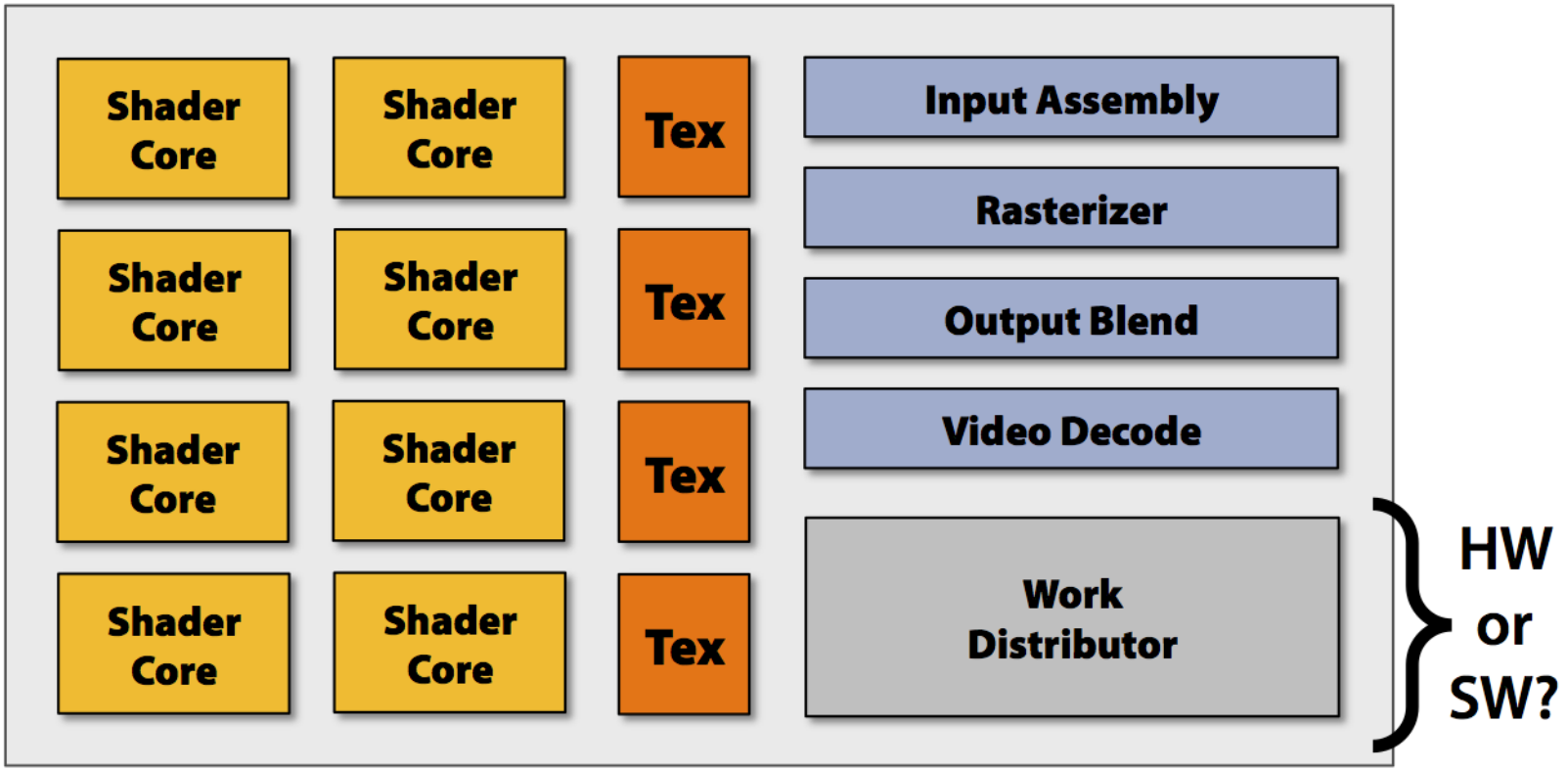
- Was ist ein digitaler Signalprozessor?
- Was ist der Unterschied zwischen einem Mikroprozessor und einem digitalen Signalprozessor?
- Wie werden DSPs heute programmiert?
- Wie kann Parallelität realisiert werden?

Inhalt

- 2.4 Spezialprozessoren
  - 2.4.1 Mikrocontroller (µC)
  - 2.4.2 Digitale Signalprozessoren (DSPs)
  - **2.4.3 Grafik Prozessoren (GPU)**
- 2.5 Bus, Network-on-Chip (NoC) & Multicore
- 2.6 Anwendungs-spezifische Instruktionssatz Prozessoren (ASIP)
- 2.7 Field Programmable Gate Arrays (FPGA)
- 2.8 System-on-Chip (SoC)

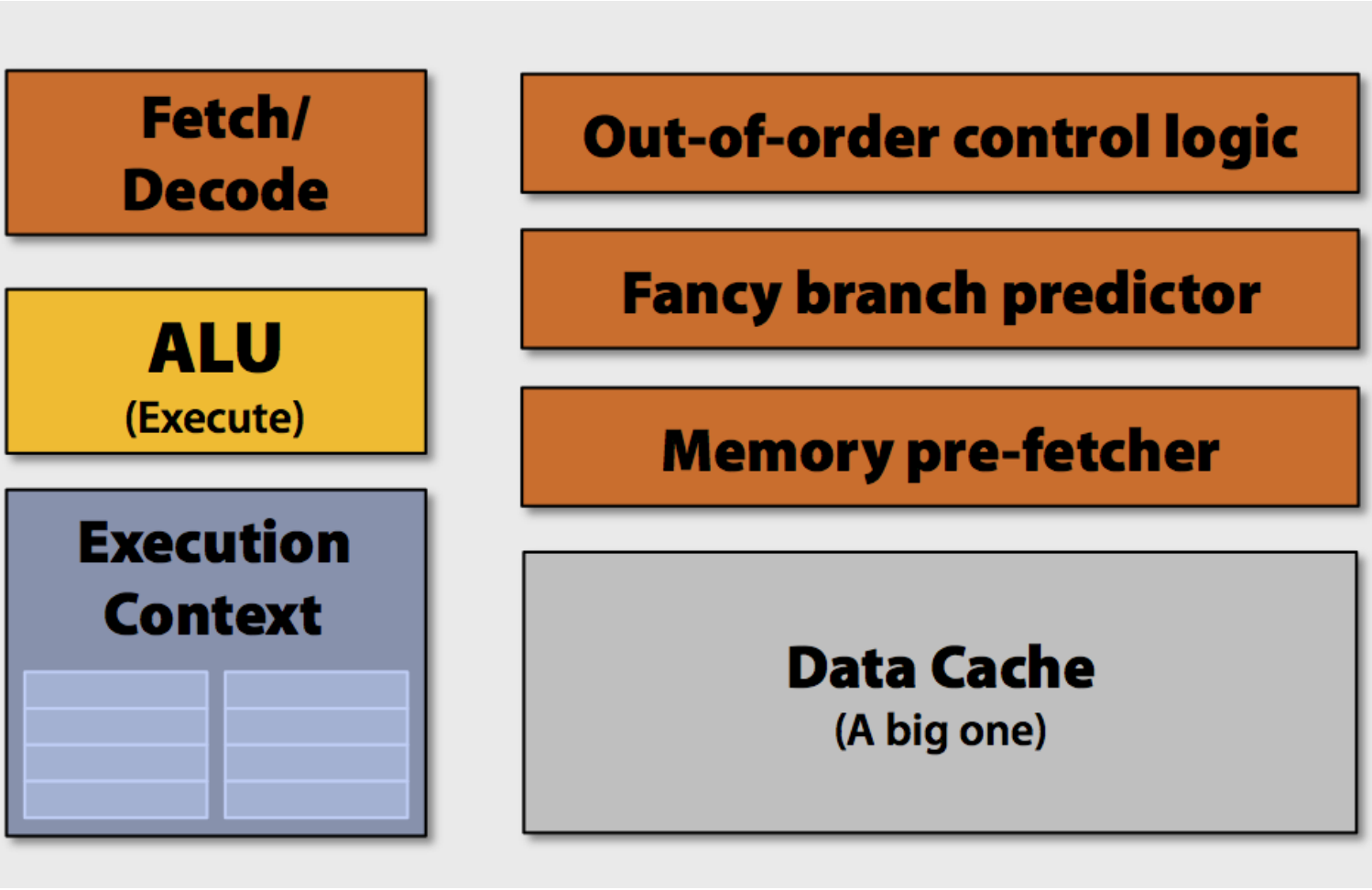
2.4.3 Was ist eine GPU?

- Heterogener Chip Multi-Prozessor
  - Speziell angepasst für Grafikanwendungen



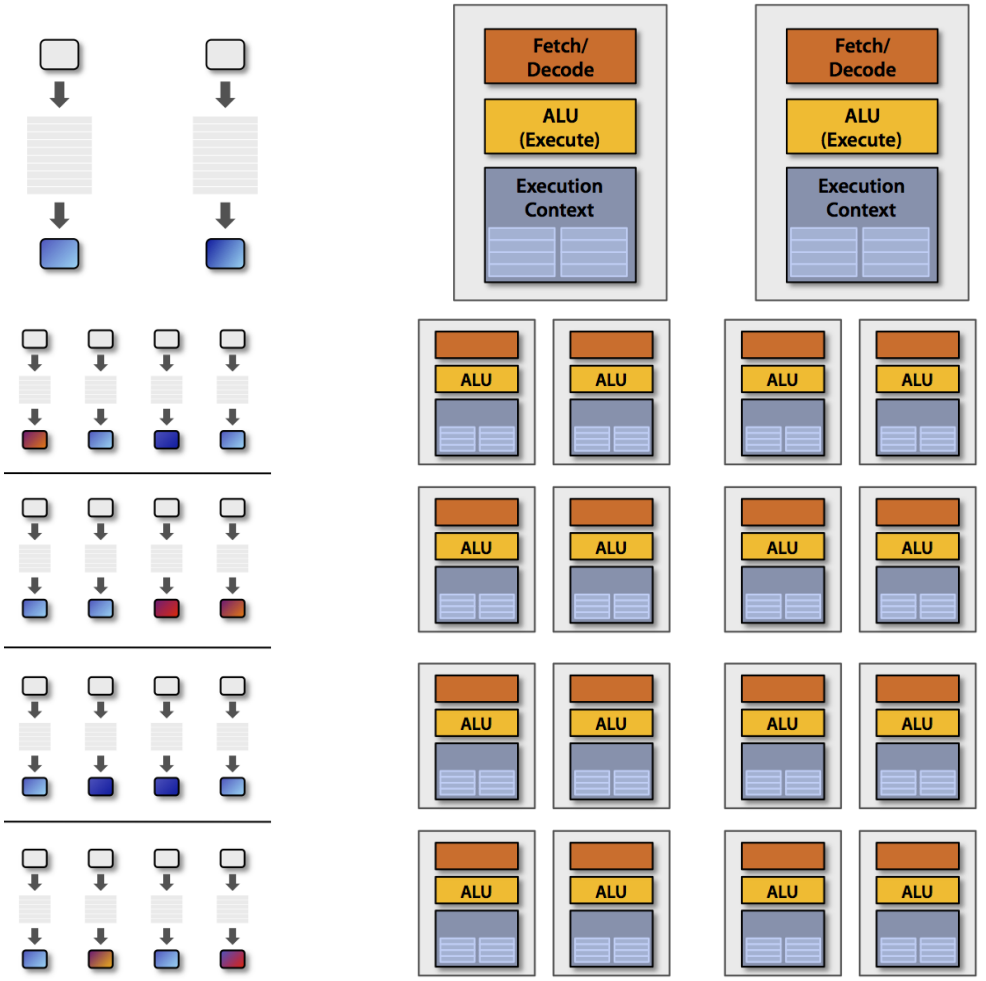
http://s08.idav.ucdavis.edu/fatahalian-gpu-architecture.pdf

2.4.3 CPU-style Kerne (I)

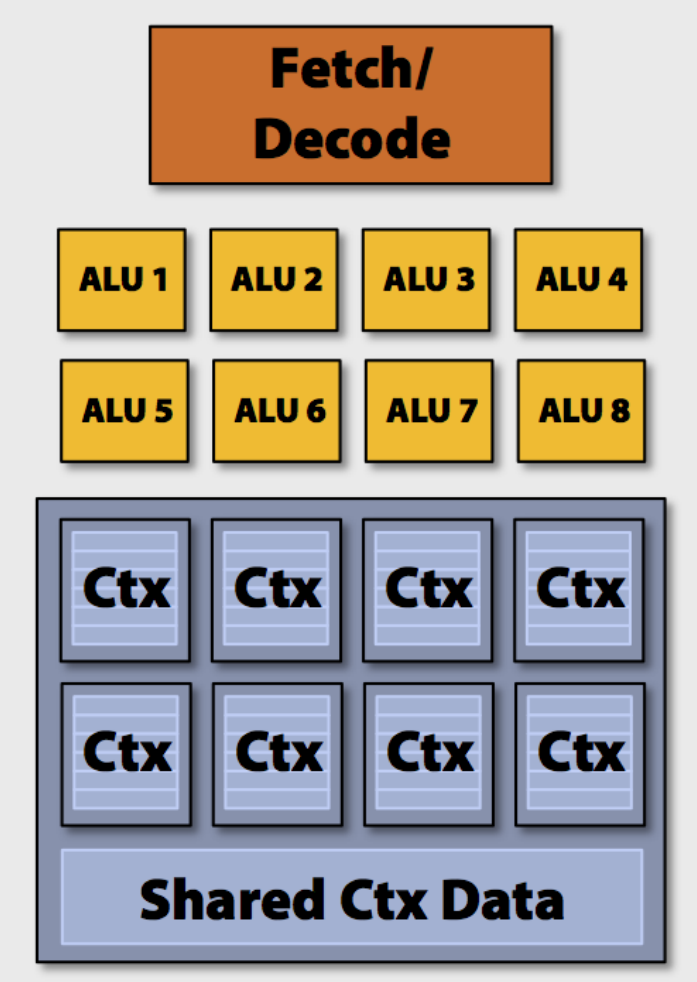


2.4.3 CPU-style Kerne (II)

- 2 Kerne
  - 2 Fragmente parallel
  - 2 simultane Instruktionsströme
- 4 Kerne
  - 4 Fragmente parallel
  - 4 simultane Instruktionsströme
- 16 Kerne
  - 16 Fragmente parallel
  - 16 simultane Instruktionsströme



2.4.3 Einfacher Prozessorkern

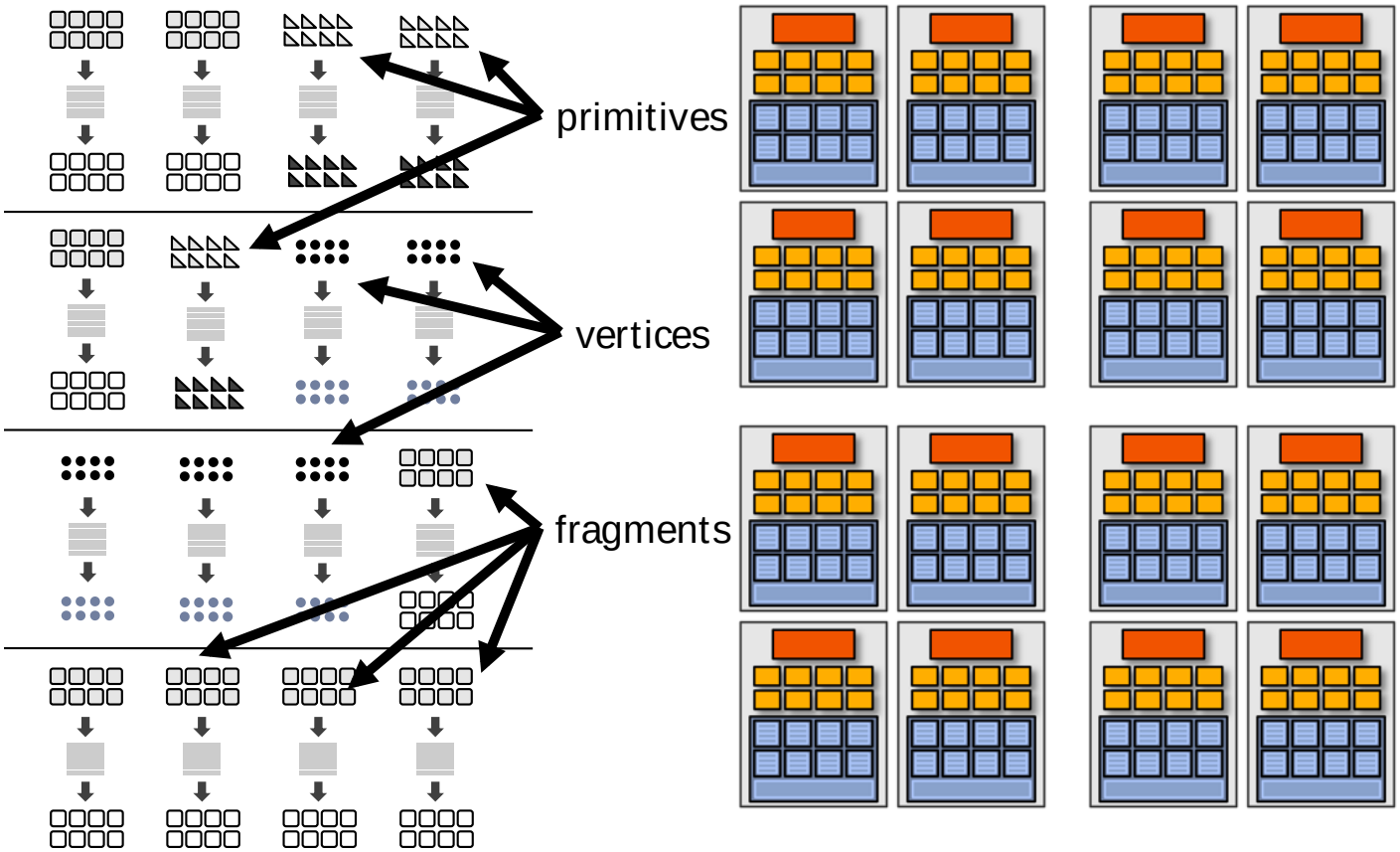


- Idee: Amortisieren von Kosten/Komplexität bei der Verwaltung von Instruktionsströmen zwischen vielen ALUs
- Hinzufügen von ALUs ➔ SIMD
  - Neu kompilierter Shader sollte auf Anzahl der im Shader Core vorhandenen ALUs angepasst sein
    - Hier: 8 parallele Fragmente für 8 ALUs

Hier: Wie viele Fragmente sollen sich einen Instruktionsstrom teilen?

2.4.3 128 [Vertices/fragments
Primitives
CUDA threads
OpenCL work items
Computer shader threads] parallel

- 16 Shader Cores mit je 8 internen Fragmenten (ALUs)
  - Insgesamt 128 parallel nutzbare Fragmente
  - Weiterhin 16 simultane Instruktionsströme



2.4.3 SIMD Prozessierung

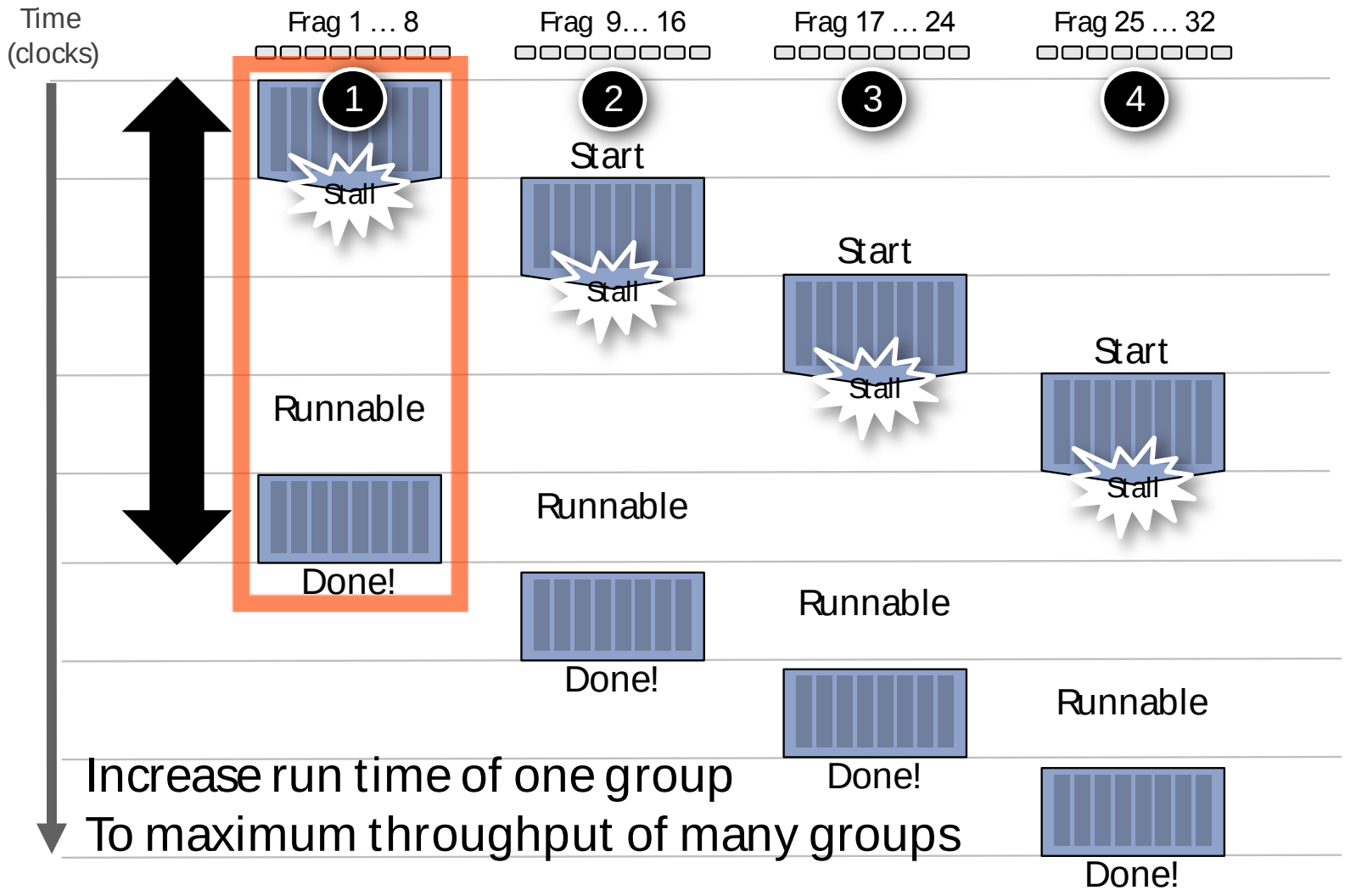
- SIMD Prozessierung impliziert nicht SIMD Instruktionen
- Option 1: Explizite Vektor Instruktionen
  - Intel/AMD x86 SSE, Intel Xeon Phi
- Option 2: Skalare Instruktionen, implizite Hardware Vektorisierung
  - Hardware entscheidet Verteilung der Instruktionsströme auf ALUs
  - NVIDIA GeForce (SIMT warps), AMD Radeon architectures

- Realität: 16 bis 64 Fragmente teilen sich einen Instruktionsstrom

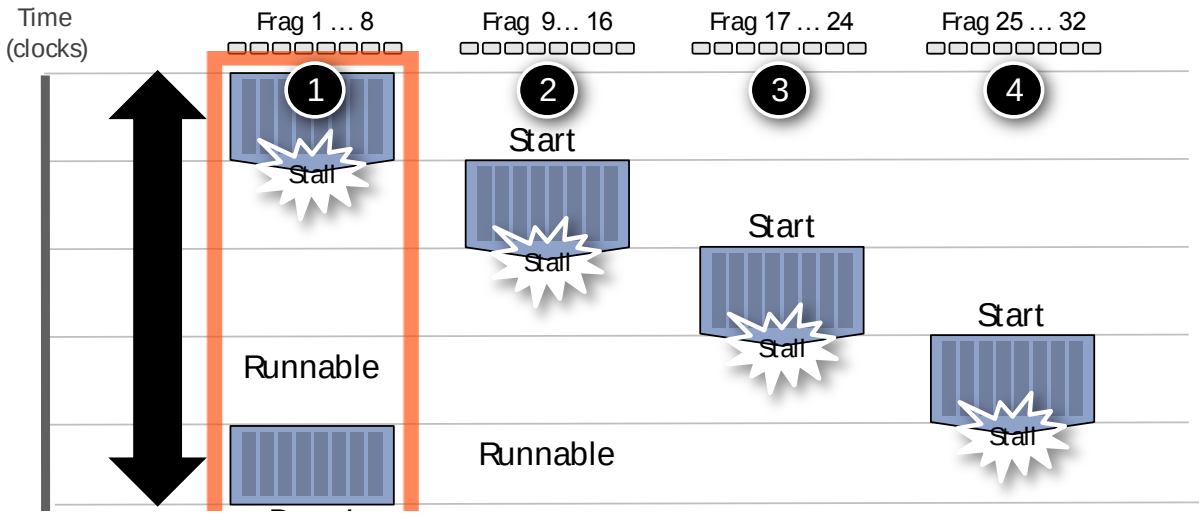
2.4.3 Stalls und Interleaved Processing

- Stalls kommen vor, wenn ein Kern aufgrund einer Abhängigkeit (z.B. Datenabhängigkeit, Speicherzugriff) die nächste Instruktion nicht ausführen kann.
- Zugriffslatenzen im Bereich von 100en bis 1000en Zyklen
- Die “Fancy Caches und Logik” die normalerweise bei Stalls hilft haben wir entfernt.
- Aber wir haben eine Menge unabhängiger Fragmente
- Idee #3:
  - Die Ausführung vielr Fragmente verschachteln um stalls durch Operationen mit hoher Latenz zu vermeiden.

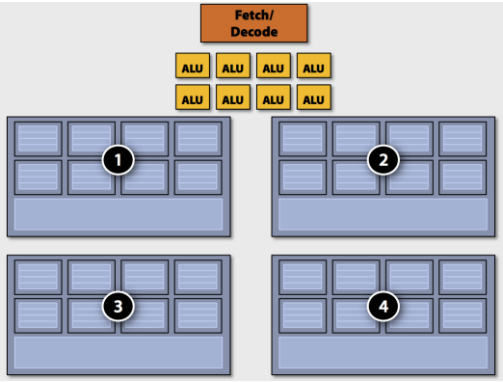
2.4.3 Durchsatz (I)



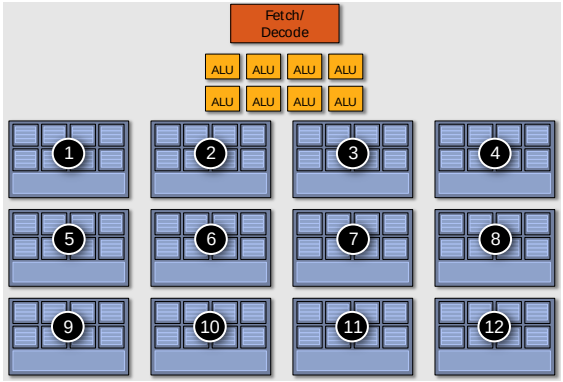
2.4.3 Durchsatz (II)



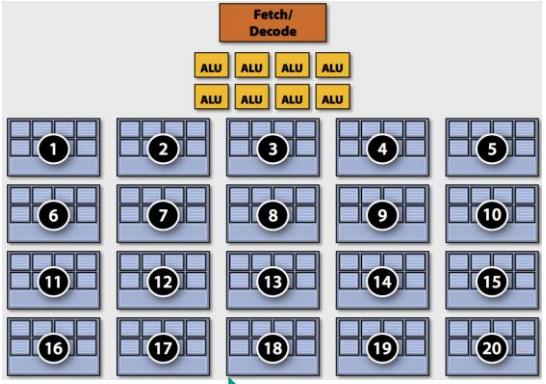
- 4 kleine Kontexte



12 mittlere Kontexte



20 große Kontexte



latency hiding ability

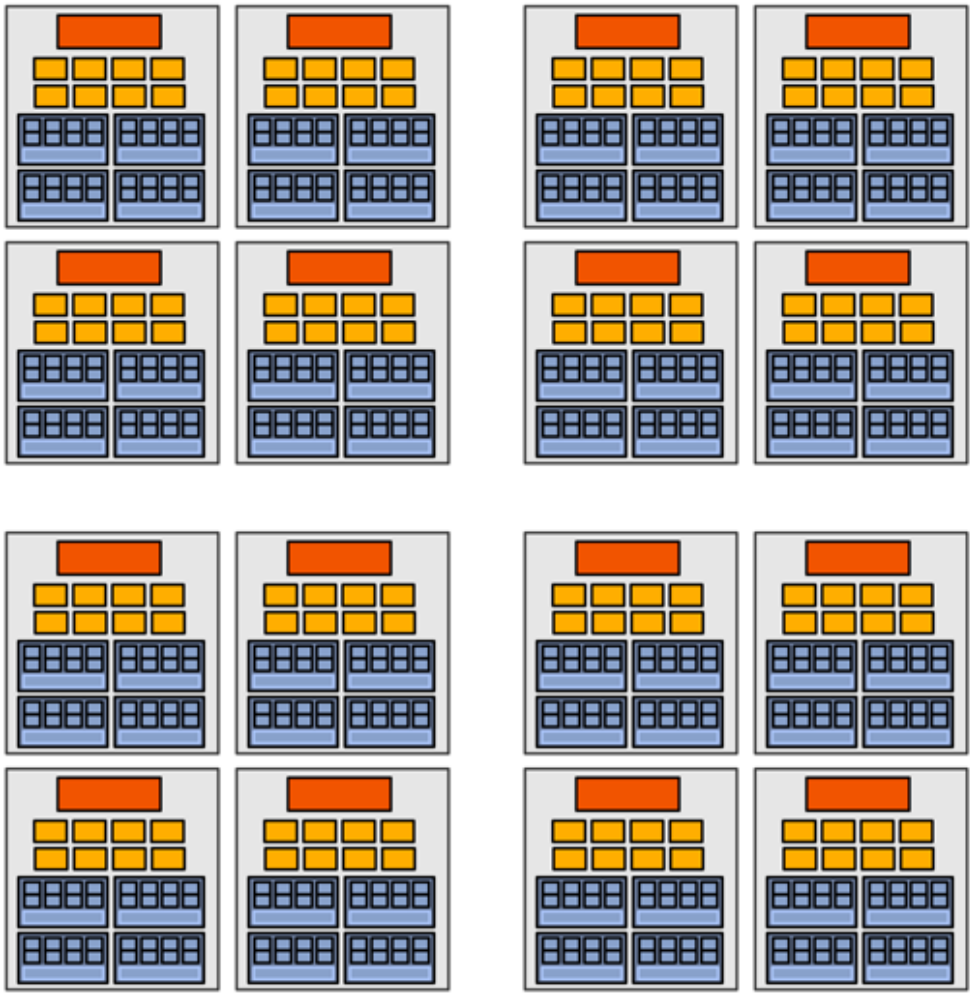
2.4.3 Beispiel

16 cores

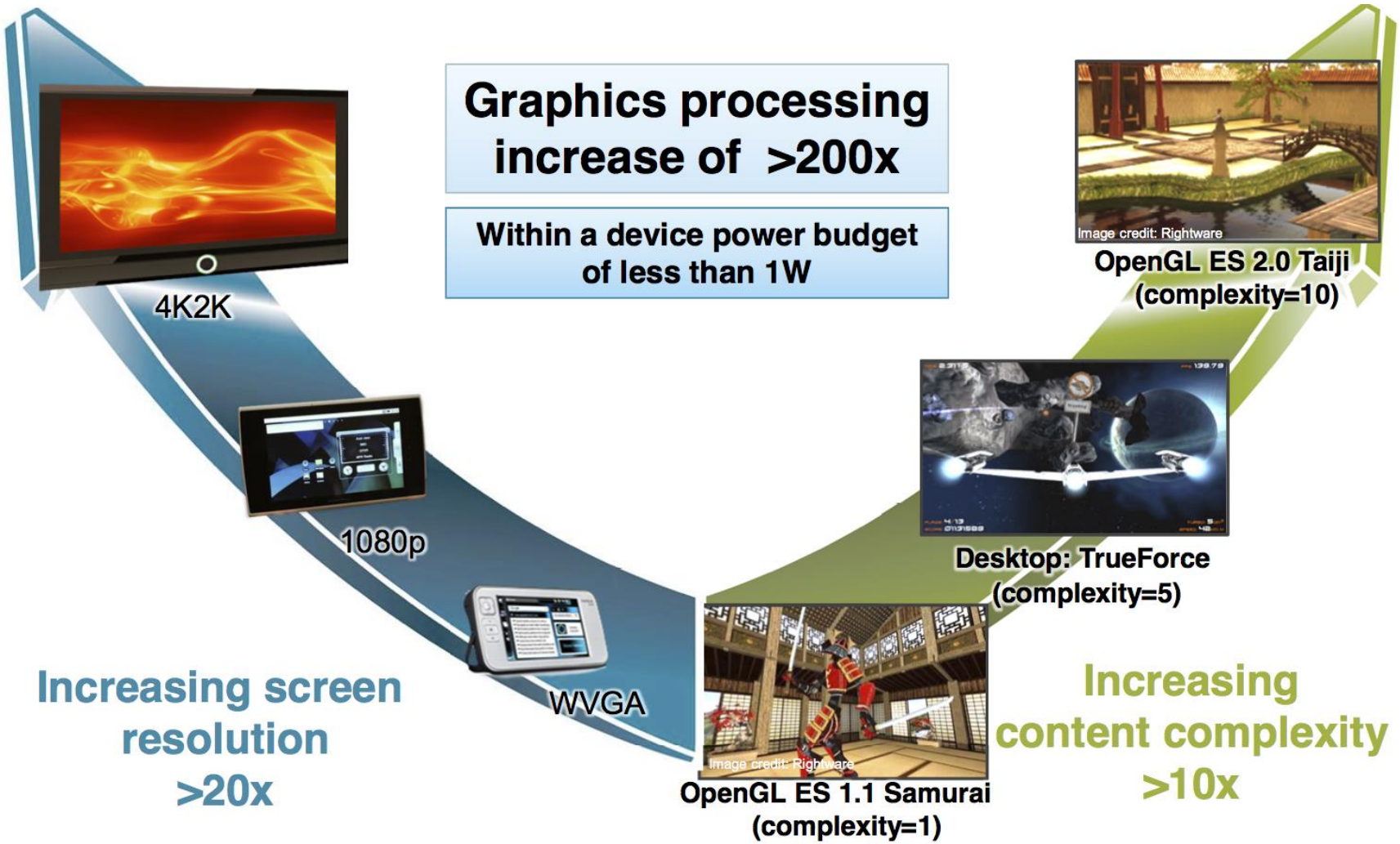
8 mul-add ALUs per core (128 total)

16 simultaneous instruction streams

64 concurrent (but interleaved) instruction streams

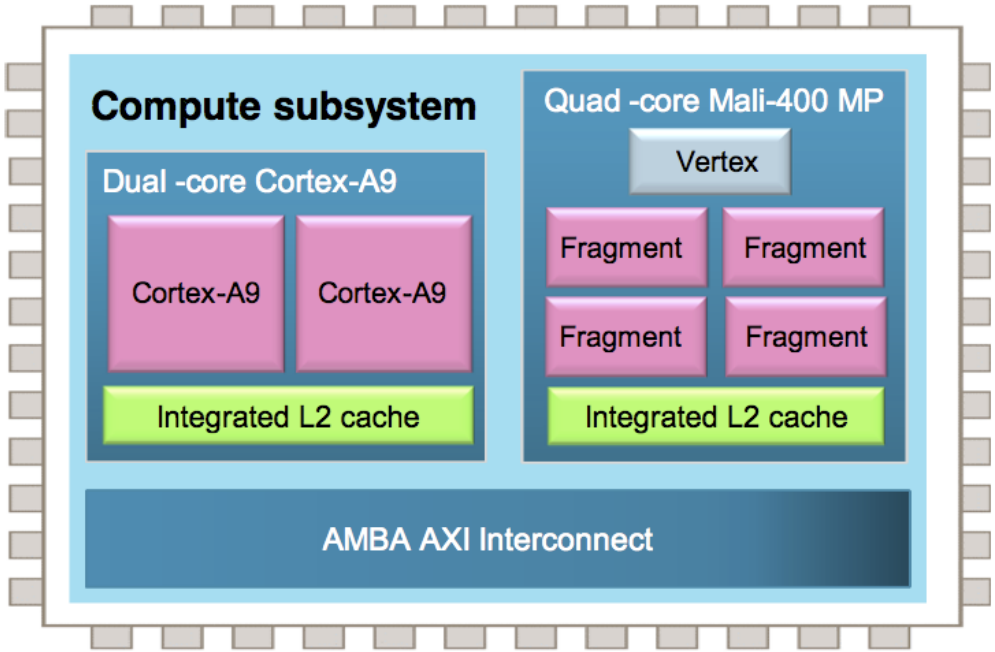


2.4.3.2 Höhere Performanz bei weniger Leistung



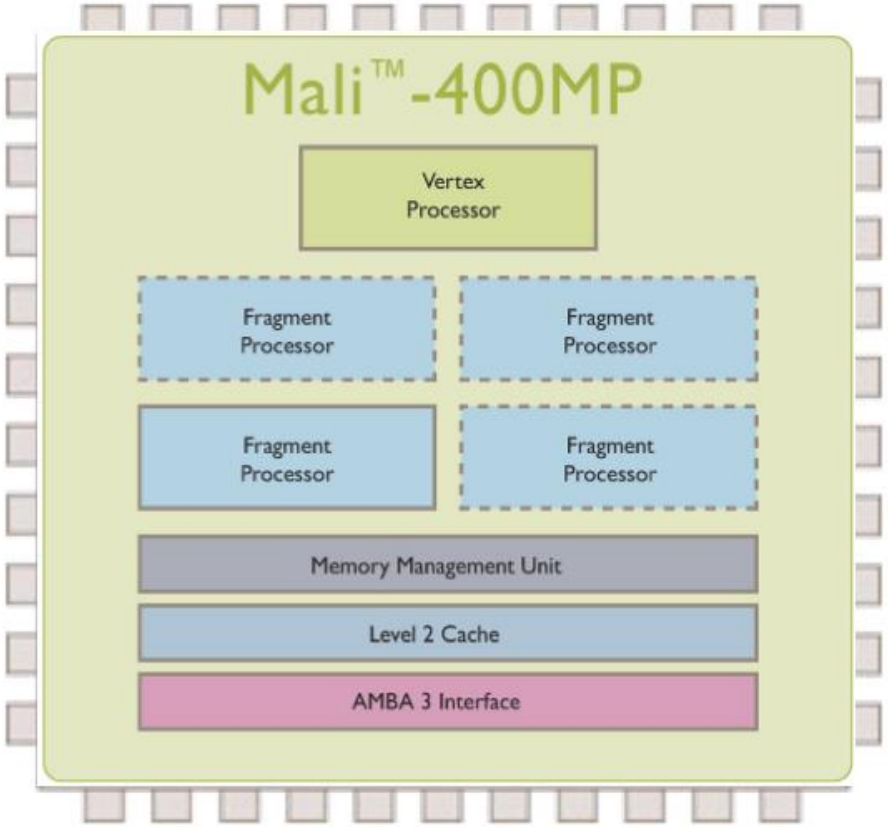
2.4.3.2 Mobile Geräte von heute

- Quad-core Mali-400 MP GPU
- Dual-core CortexTM-A9 CPU
- OpenGL ES 1.1 and 2.0
- OpenVGTM 1.1



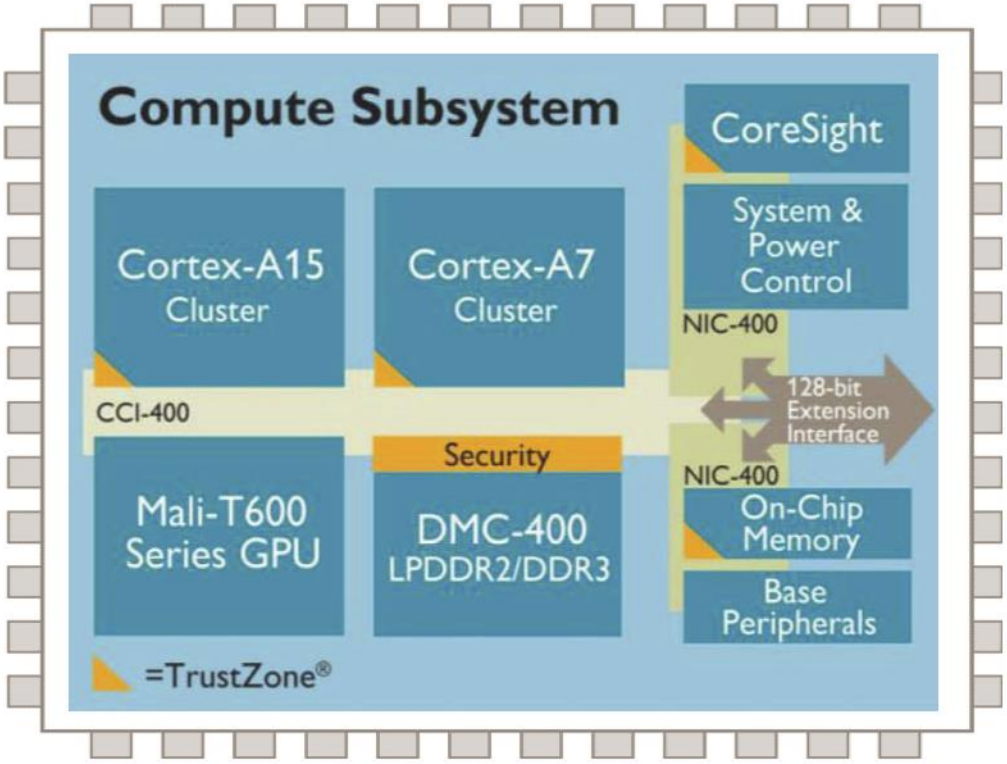
2.4.3.2 Beispiel: ARM Mali-400 MP

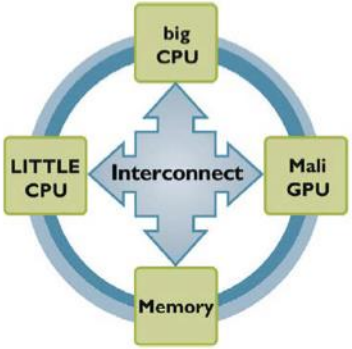
- Multi-Prozessor skalierbare Grafik Performanz
  - Bis zu 4 Fragment-Prozessoren
- 2D und 3D Grafikbeschleunigung
  - OpenGL ES 2.0 / 1.1 und OpenVG 1.1
- Hohe Performanz und Bildqualität
  - Skaliert bis 1080p Auflösungen mit Anti-Aliasing
- Effiziente Energie- und Bandbreiten-Ausnutzung
  - Integrierter konfigurierbarer L2 Cache
  - Führende Bandbreiten-Effizienz und Latenz-Toleranz
- Hoch-qualitative Treiber
  - Einziger optimierter Treiber für alle Konfigurationen
  - Multicore Skalierung transparent für Software Entwickler



2.4.3.2 Mobile Geräte von morgen

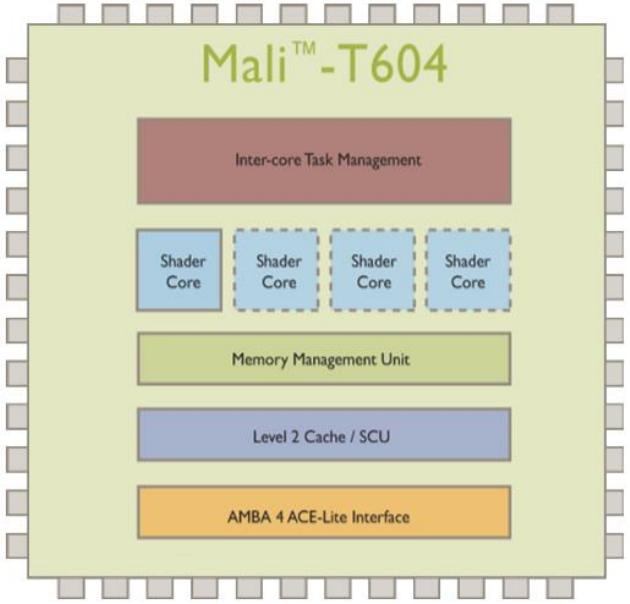
- Quad-core Mali-T604
- ARM big.LITTLE
- 4xCortex-A15 + 4xCortex-A7
- Grafik- und Rechen-API Unterstützung

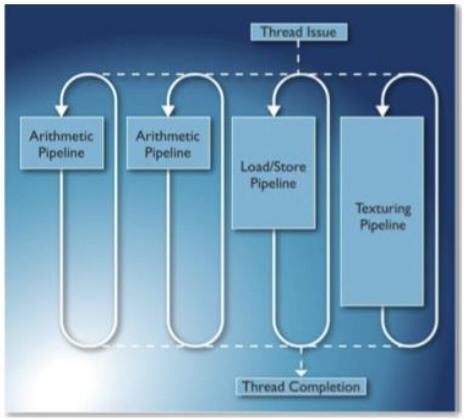




2.4.3.2 Beispiel: ARM Mali-T604

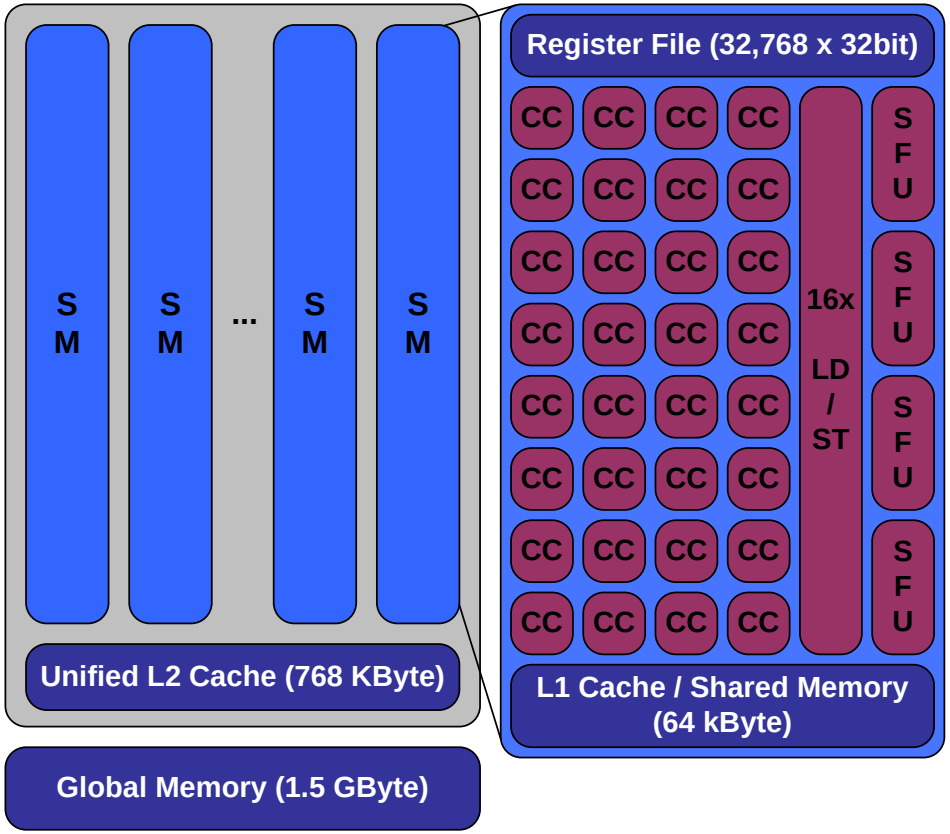
- Innovative GPU Architektur
  - Tri-pipe – für Performanz und Flexibilität
  - GPGPU Computing und Grafik
- Performanz und Skalierbarkeit
  - Bis zu 5x Mali-400 MP Performanz
  - Skalierbar bis zu 4 Kernen
- Power Effizient
  - Dynamisches power Management
- API Unterstützung
  - OpenGL ES 2.0 und 1.1, OpenVG 1.1
  - OpenCL 1.1 Full Profile für GPGPU
  - DirectX
- OS Unterstützung
  - Android, Linux, Windows Phone





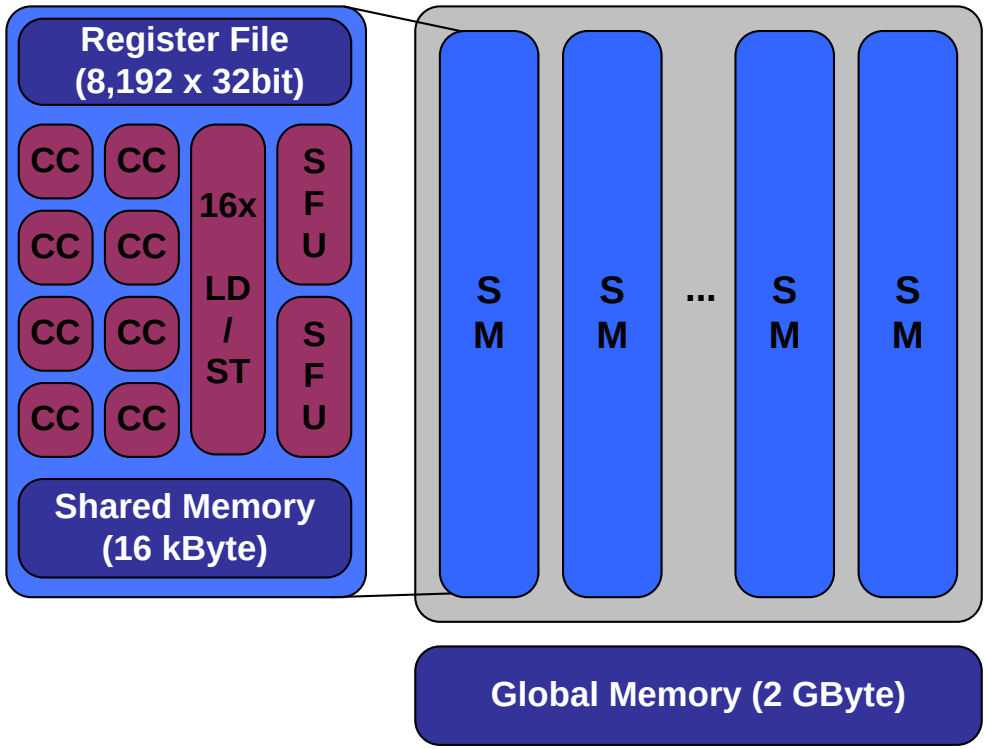
2.4.3.3 Beispiel: Nvidia Geforce GTX 580

- GF 110 – “Fermi“ Mikroarchitektur
- 16 Streaming Multiprozessoren
  - 32 CUDA Cores @ 1.54 GHz
  - 32.768 Register
  - 64 kByte lokaler Speicher (konfigurierbar in Level-1 Cache und Shared Memory)
- Global Memory
  - Unified Level-2 Cache
  - Hier: 1.5 GByte GDDR5 RAM



2.4.3.3 Beispiel: Nvidia Geforce GTX 285

- GT 200b Mikroarchitektur
- 30 Streaming Multiprozessoren
  - 8 CUDA Cores @ 1.48 GHz
  - 8,192 Register
  - 16 kByte Speicher (Shared Memory)
- Global Memory
  - Hier: 2 GByte GDDR3 RAM



- Parallel Prozessor Fähigkeiten
  - Hoher Grad an Parallelität für effiziente Nutzung
  - Implizites Daten-paralleles Programmiermodell
    - Führt skalaren Kernel auf tausenden Threads und Datenelementen aus
  - Kostspielige inter-thread Kommunikation oder Synchronisation

- Was ist eine GPU?
- Was ist ein Shader Core?
- Wieso gibts es mehrere ALUs pro Kern?
- Wie kann der Durchsatz erhöht werden?
- Wie kann hoher Durchsatz bei geringer Leistungsaufnahme erreicht werden?
- Wieso kann eine Vielfache Beschleunigung gegenüber GP-Prozessoren erreicht werden?

Inhalt

- 2.4 Spezialprozessoren
  - 2.4.1 Mikrocontroller (µC)
  - 2.4.2 Digitale Signalprozessoren (DSPs)
  - 2.4.3 Grafik Prozessoren (GPU)
- **2.5 Bus, Network-on-Chip (NoC) & Multicore**
- 2.6 Anwendungs-spezifische Instruktionssatz Prozessoren (ASIP)
- 2.7 Field Programmable Gate Arrays (FPGA)
- 2.8 System-on-Chip (SoC)

2.5.1 Bussystem (Bus)

- Den Daten und Informationsaustausch zwischen verschiedenen Systemkomponenten ermöglichende, als Linien- oder Ringnetz aufgebaute mehradrige Sammelleitung, an die alle Komponenten angeschlossen sind.
- Sie verbindet den Ausgang jeder Komponente mit den Eingängen aller übrigen.
- Der Datenaustausch zwischen den Komponenten erfolgt im Multiplex-Betrieb.
- Transportsystem für Informationen mehrerer Teilnehmer (Broadcast).

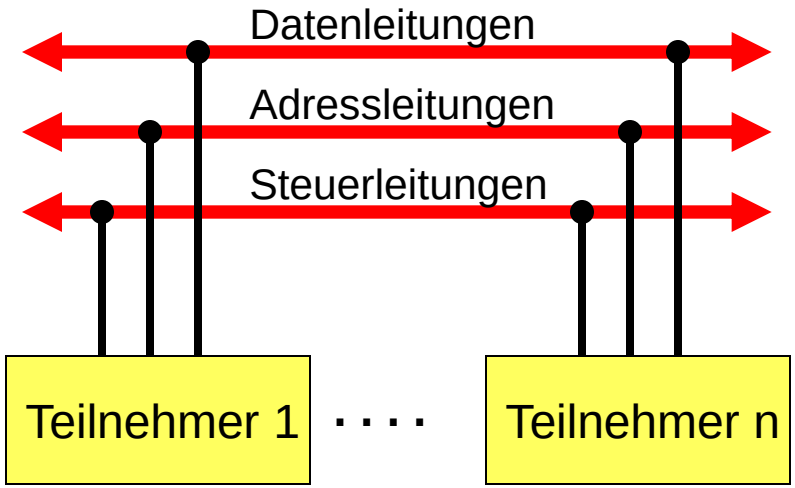
2.5.1 Bussystem (Bus) - Anwendungsbereich

- Verbindung von Komponenten auf allen Ebenen:
  - funktionale Einheiten auf Chips (on-Chip Bus)
  - Chips zum System/Rechner (Systembus)
  - Rechner zu Ein-/Ausgabegeräten (Peripheriebus)
  - Steuergeräte/Computer in Fahrzeugen, Flugzeugen (z.B. Feldbus)

- Teilnehmer sind über (Bus-)Schnittstelle an den Bus angeschlossen
  - Schnittstellendefinition ist Teil der Busdefinition
    (zusätzlich z.B. Festlegung der Buslänge, Zahl von Einschubplätzen, Terminierung)

2.5.1 Parallele Bussysteme

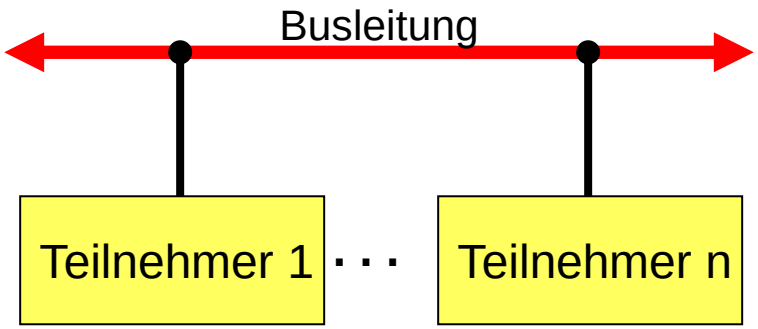
- Hardware:



- Datenbus:
  - Eigentliche Datenübertragung typ. Wortbreite: 8, 16, 32, 64, 128 Bit
- Adressbus:
  - Auswahl einzelner Geräte und Adressen innerhalb der Geräte
    - z.Teil: Daten und Adressen zeitversetzt auf den selben Leitungen (z.B. IEC-Bus)
- Steuerbus:
  - Busanforderung, Arbitrierung, Interrupts, Handshaking
  - Versorgungsbus: Stromversorgung und Taktleitung

2.5.1 Serielle Bussysteme

- Hardware:



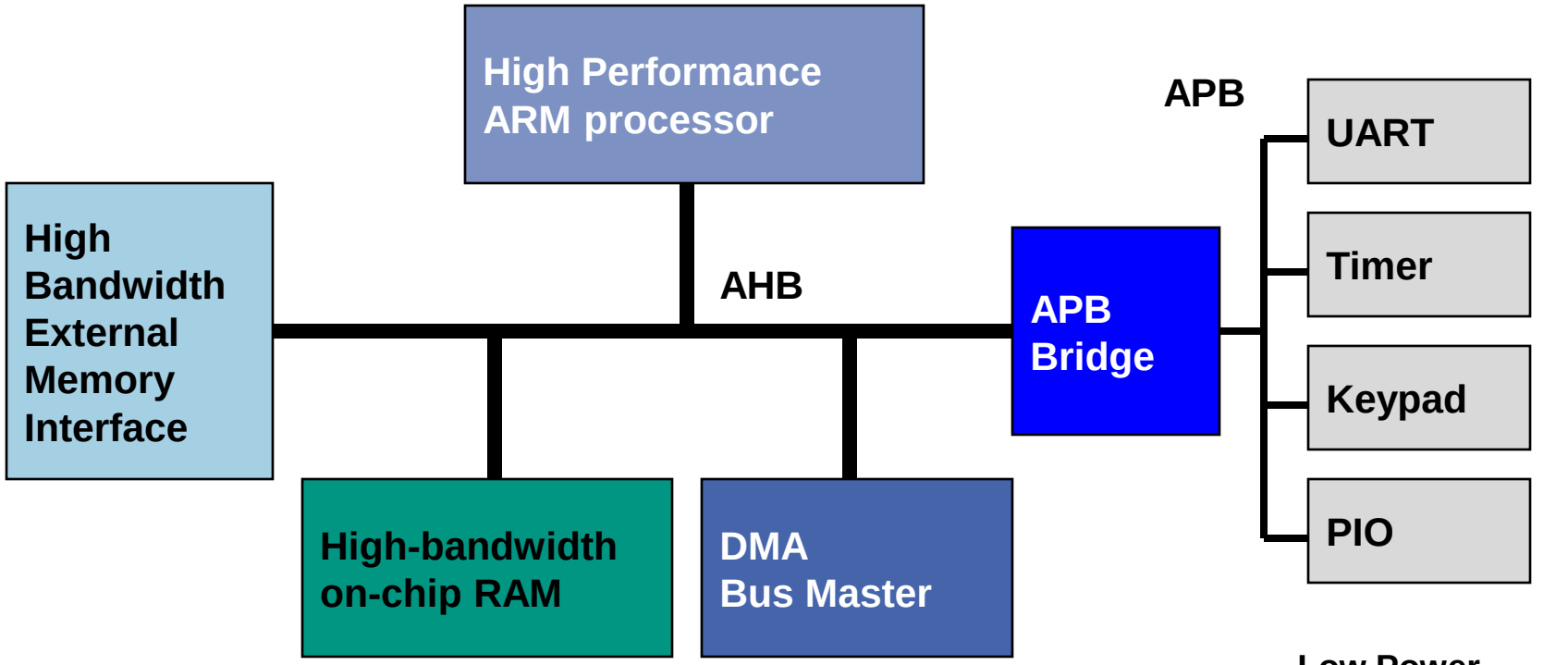
- Nur eine Leitung, die als Busstruktur ausgebildet ist.

- Funktionaler Aufbau (Busprotokoll):
  - Funktionen, die bei parallelen Bussystemen durch spezielle Leitungen übernommen werden, sind hier durch (Software-) Protokolle realisiert.
  - serielle Busse sind in Bezug auf Änderungen i.Allg. flexibler als parallele Busse.

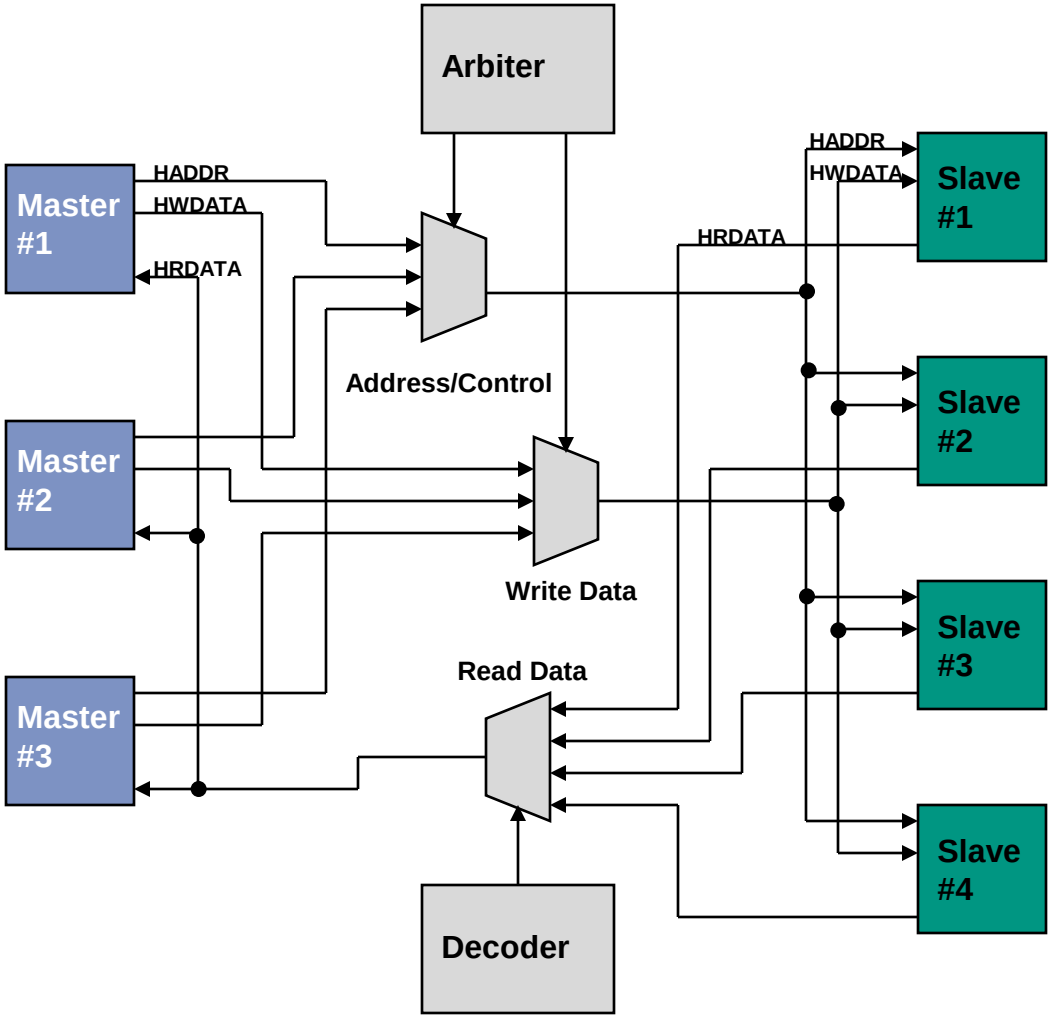
2.5.1 Bussystem Beispiel: Advanced Microcontroller Bus Architecture (AMBA)

- Offener Standard einer on-chip Bus Spezifikation für Plattform-basierte Designs (SoC).
- IP Module: eingebettete Prozessoren, Speicher, periphere Einheiten
- Wiederverwendbarkeit der Module durch gemeinsamen “backbone”.
- 2 on-chip Busse mit einer Bridge verbunden:
  - high-speed bus:
    verbindet Prozessoren, DMA controller, on-chip memory, high-performance units
  - low-speed, lower-power peripheral bus:
    einfacheres Protokoll, verbindet Timer, GPIO, SIO, ….

2.5.1 Ein AMBA Beispiel System

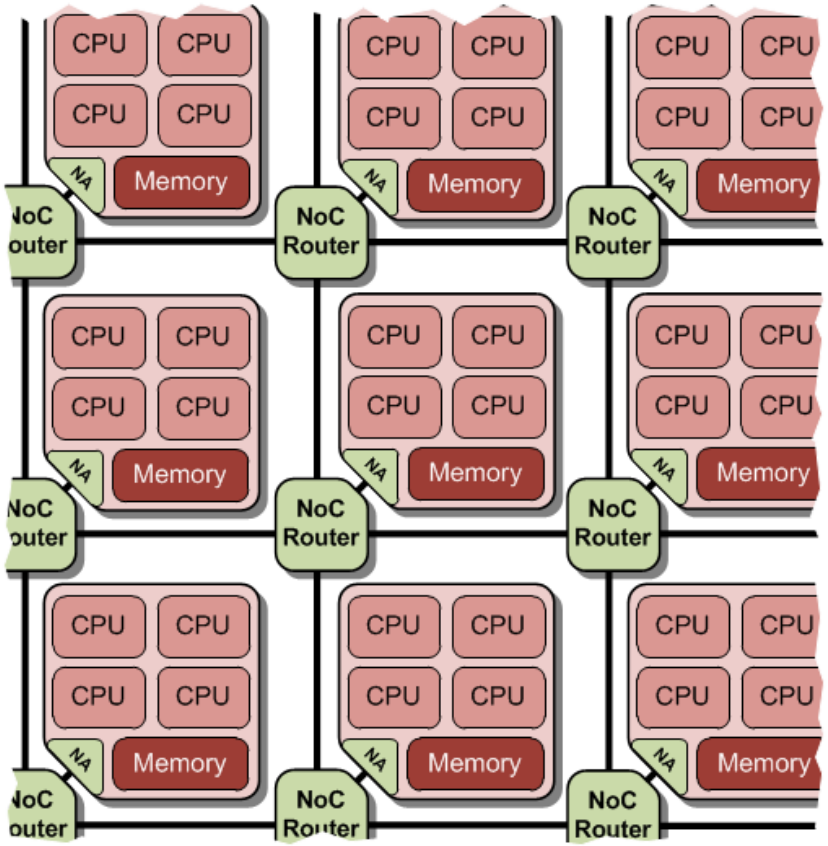


2.5.1 AHB Struktur



2.5.2 Network-on-Chip (NoC)

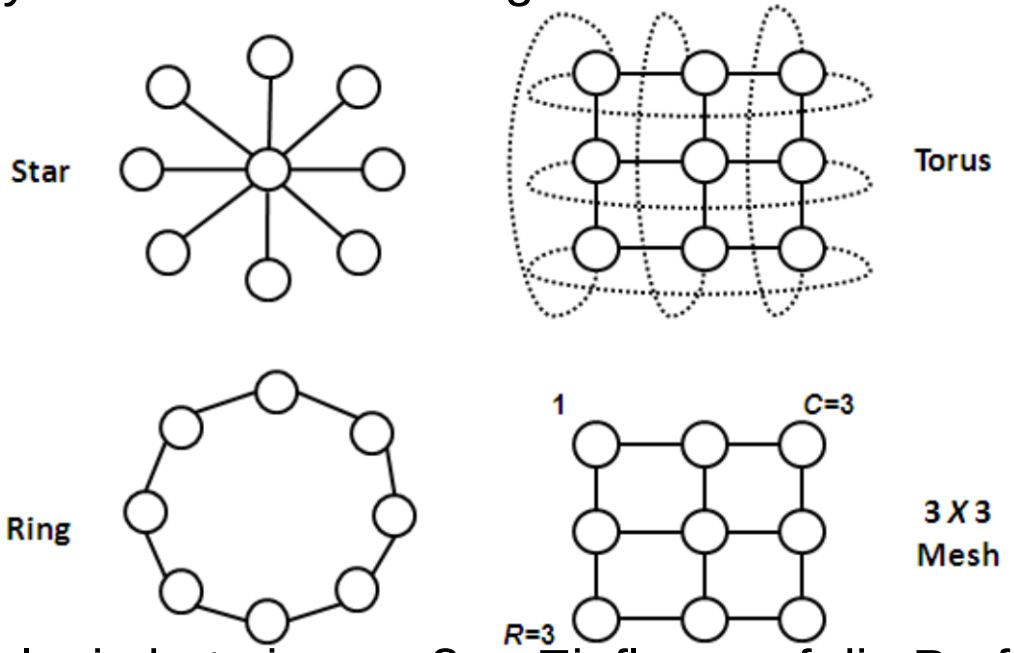
- Networks-on-Chip bieten eine skalierbare Kommunikationsinfrastruktur
- Jede Komponente bringt Kommunikationshardware mit
- Ein Kommunikationsvorgang blockiert immer nur einen Teil der Kommunikationsinfrastruktur
- Im Vergleich zu Bussystemen größere Komplexität
- Komponenten eines NoC:
  - Router
  - Routingentscheidung
  - Packete zwischenspeichern
  - Weiterleitung der Pakete
  - Network-Adapter
  - Protokollübersetzung
  - Zwischenspeichern/umsortieren ankommender Pakete



Quelle: *ITIV, Heißwolf*

2.5.2 NoC – Topologie

- Topologie: Physikalische Anordnung der Netzwerkknoten

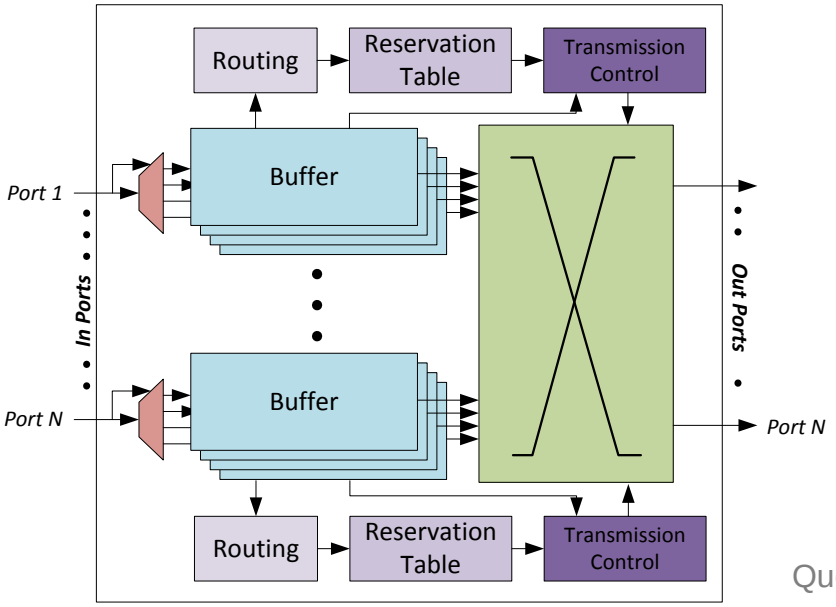


- Wahl der Topologie hat eine großen Einfluss auf die Performanz und den Flächenbedarf:
  - Komplexität des Routers ist abhängig von der Anzahl der Router-Ports
  - Performanz des Netzwerks ist abhängig von der Anzahl an direkten Nachbarn

Quelle: *Rickard Holsmark, Maurizio Palesi, and Shashi Kumar. 2006 Deadlock free routing algorithms for mesh topology NoC systems with regions*

2.5.2 Network-on-Chip – Router

- Aufbau eines Packet-Switching Routers mit virtuellen Kanälen:

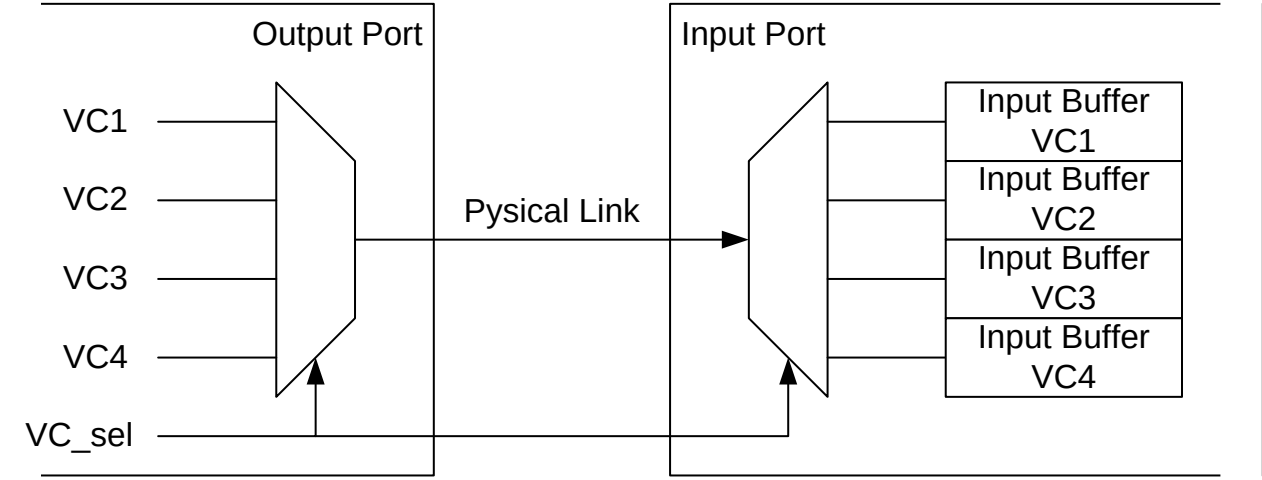


Quelle: *ITIV, Heißwolf*

- Funktionsweise:
  - Eingehende Pakete werden entsprechend der virtuellen Kanäle auf Buffer verteilt.
  - Zieladressen werden in der Routing-Einheit verarbeitet und Reservierungen des Ausgangsports vorgenommen
  - Die „Transmission Control“ regelt, welche Reservierung zum Zuge kommt und leitet die entsprechenden Daten weiter

2.5.2 Network-on-Chip – Virtuelle Kanäle

- Virtuelle Kanäle verbessern die Performanz und ermöglichen mehrere parallele Verbindungen über einen Link:

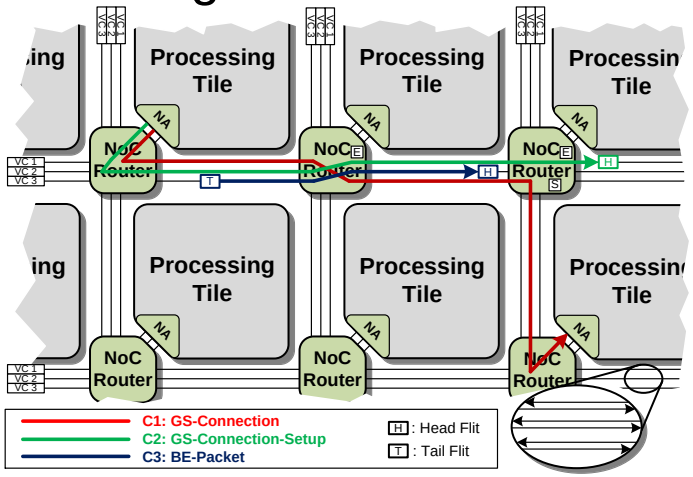


- Prinzip der virtuellen Kanäle:
  - Der physikalische Link wird im Zeitmultiplexing zwischen unterschiedlichen virtuellen Kanälen aufgeteilt
  - Performanz des Netzwerkes wird durch virtuelle Kanäle erhöht, da die Wahrscheinlichkeit von Blockaden reduziert wird
  - Die Anzahl der mögliche Verbindungen bei Verbindungsorientierter Kommunikation kann stark erhöht werden

Quelle: *ITIV, Heißwolf*

2.5.2 Network-on-Chip - Datenübertragung

- Paket- und Verbindungsorientierte Kommunikation in einem NoC:



Quelle: *ITIV, Heißwolf*

- Paketorientierte Kommunikation (Best Effort (BE)):
  - Nutzdaten werden in einem Packet zusammenhängend übertragen
- Verbindungsorientierte Kommunikation (Guaranteed Service (GS))
  - Vor der eigentlichen Übertragung der Nutzdaten wird eine Verbindung von der Quelle zum Ziel etabliert
  - Große Datenmengen können effizienter übertragen werden
  - Garantien für die Latenz und den Durchsatz sind möglich

2.5.3 Skalierung bei mehr als vier Kernen

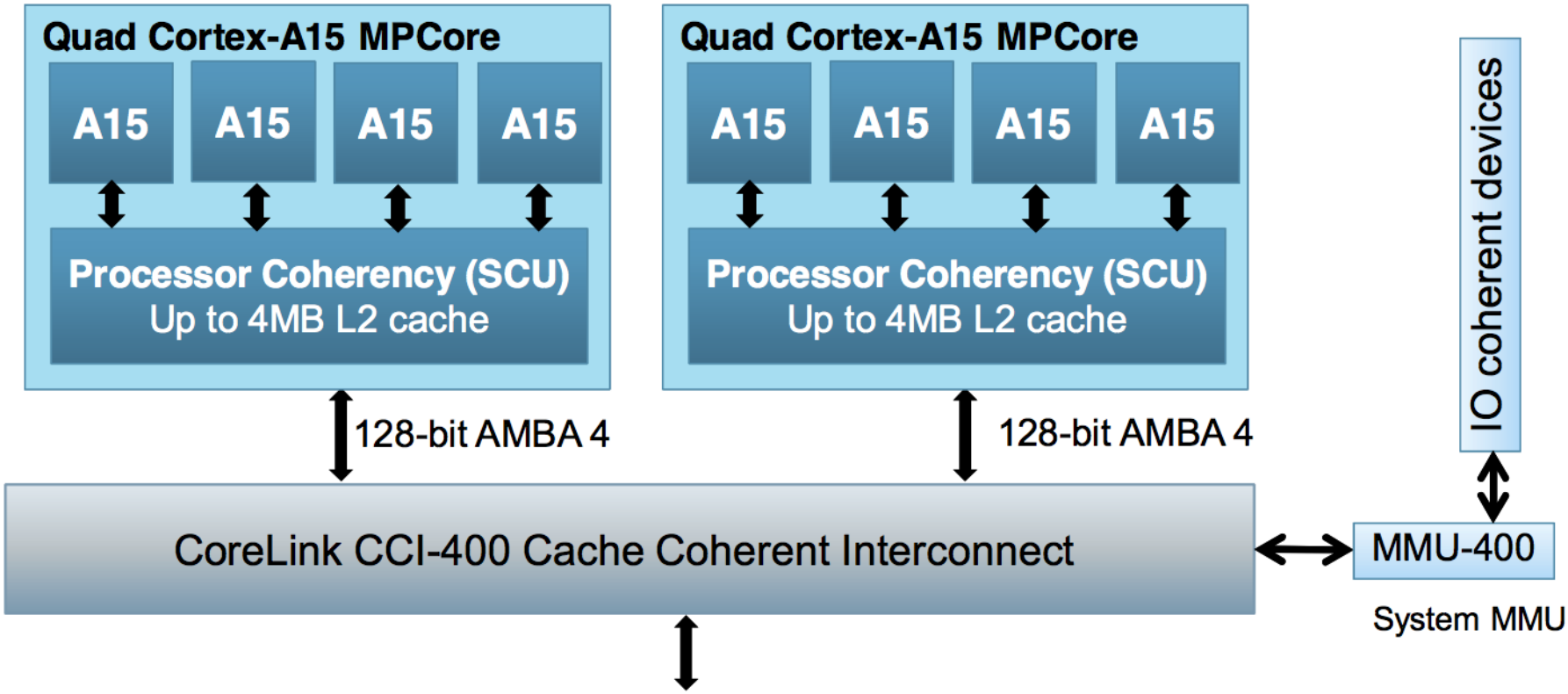
- Einführung von AMBA 4 Kohärenz Erweiterungen
  - Kohärenz, Barrier und Memory Management
- Software Implikationen
  - Hardware verwaltete Kohärenz vereinfacht Software
  - Prozessor verbraucht weniger Zeit zur Cache Verwaltung
- Kohärenz Typen
  - I/O Kohärenz
    - Geräte snoopen in Prozessor Cache
      (aber Prozessoren snoopen nicht ins Gerät)
  - Volle Cache Kohärenz
    - Cache snooping in beide Richtungen

2.5.3 Beispiel: ARM Cortex-A15

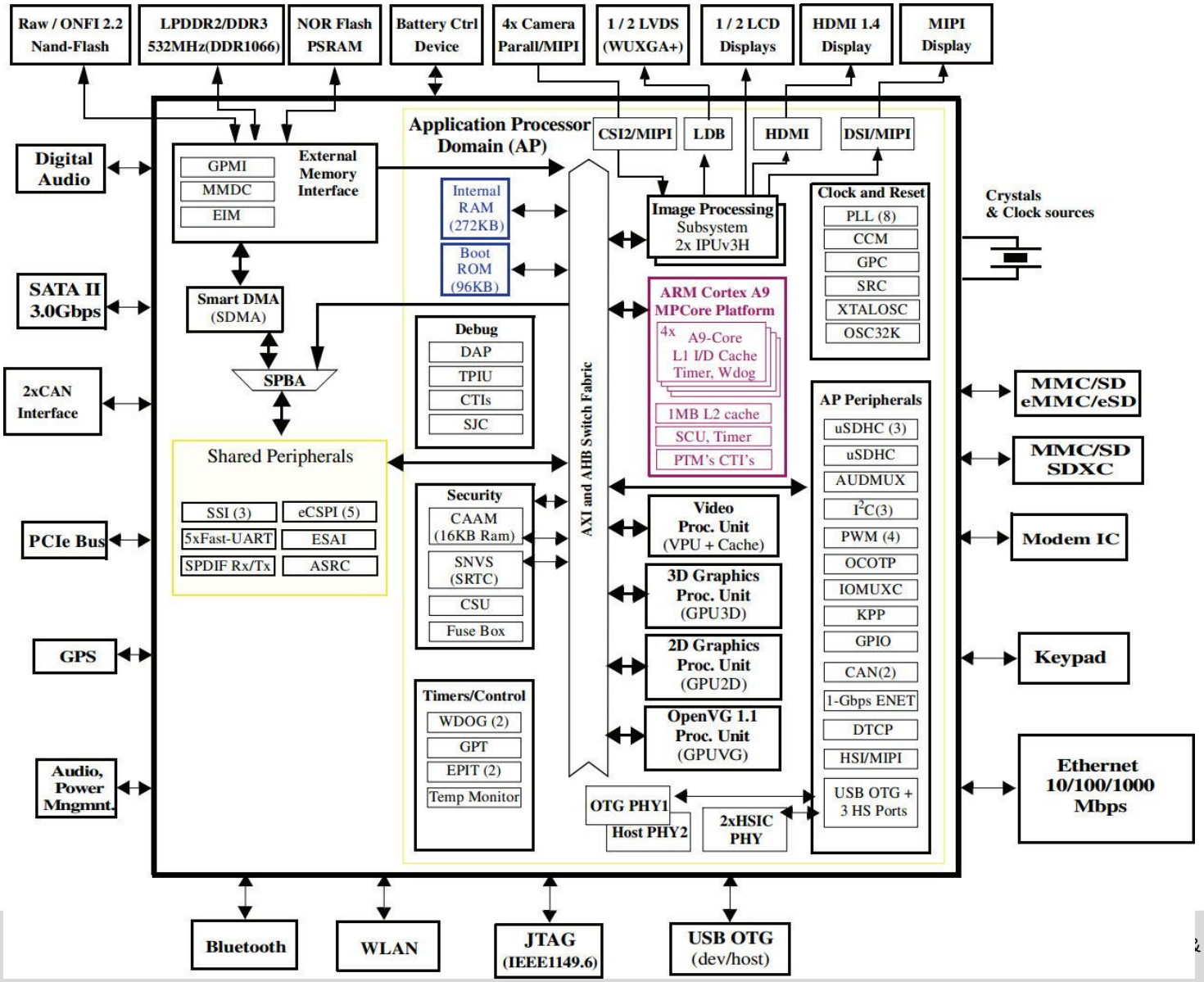
- Cortex-A class Multi-Prozessor
  - 40bit physikalische Addressierung (1TB)
  - Volle Hardware Virtualisierung
  - AMBA 4 System Kohärenz
  - ECC und Paritäts-Sicherung für alle SRAMs
- Fortschrittliches Power Management
  - Fein-granulares Abschalten der Pipeline
  - Aggressive L2 Power Reduzierungsfähigkeiten
  - Schnelles state save und restore
- Signifikante Performanz Verbesserung
  - Verbesserte single-thread und MP Performanz
- Integrierter L2 Cache mit SCU Funktionalität
- 128-bit AMBA 4 Interface mit Kohärenz Erweiterungen

2.5.3 ARM Cortex-A15 System Skalierbarkeit

- Einführung von CCI-400 Cache Coherent Interconnect
  - Prozessor zu Prozessor Kohärenz und I/O Kohärenz
  - Speicher und Synchronizations Barrieren
  - TLB und Cache Versorgung

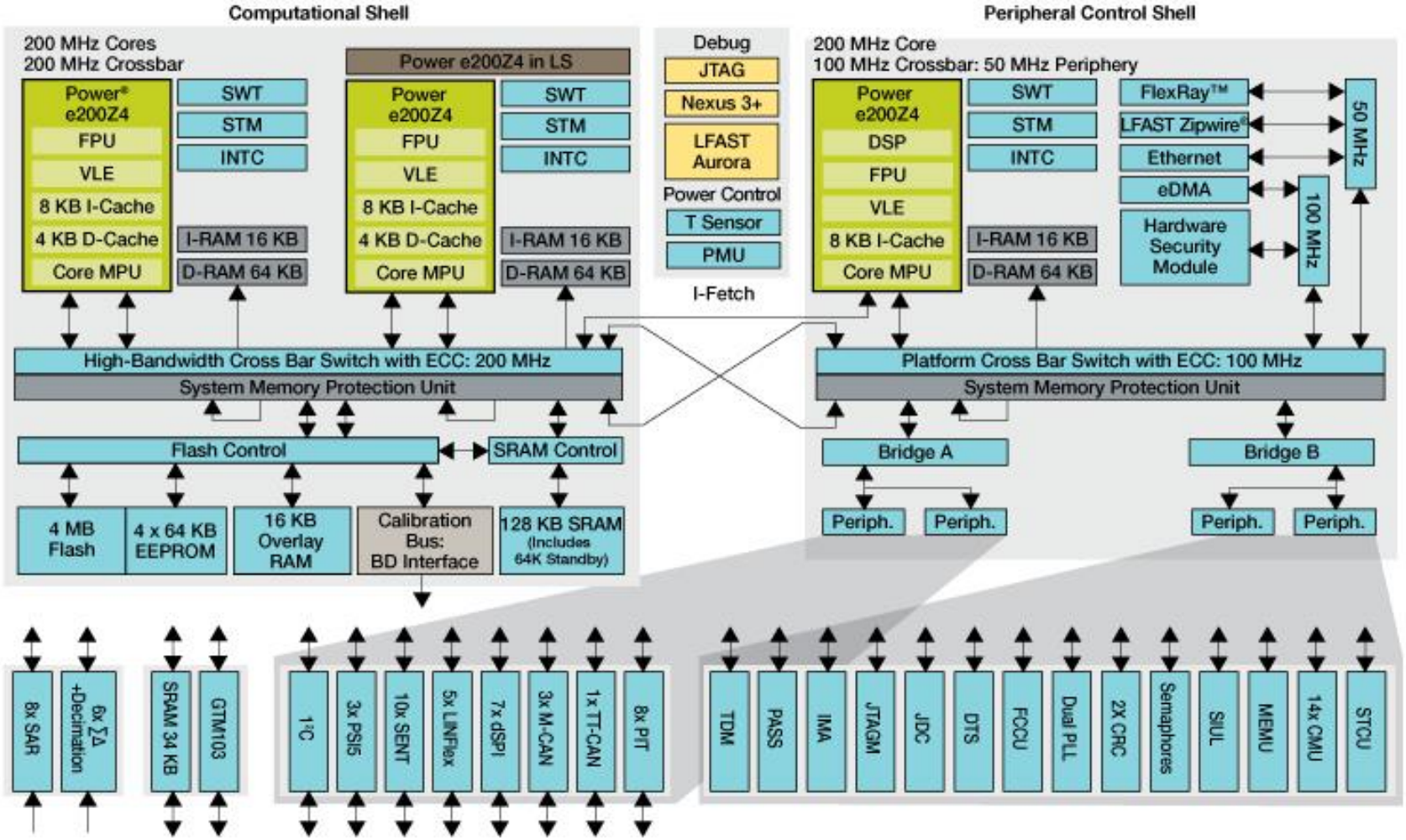


2.5.3 Beispiel Freescale i.MX6

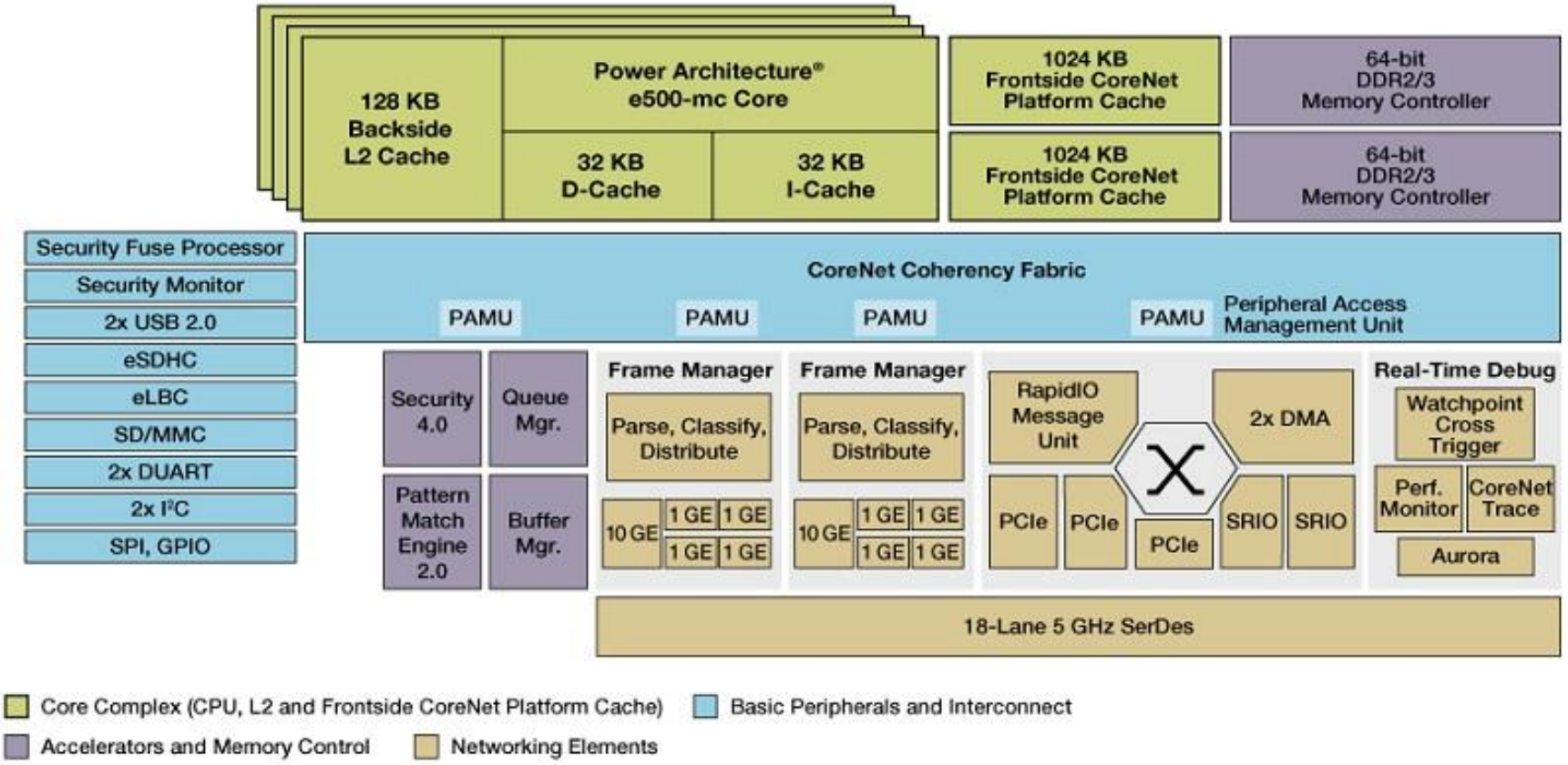


2.5.3 Beispiel: Controller für Motorsteuergeräte

Qorivva MPC5746M Block Diagram



2.5.3 Freescale P4080 – Network Processor



2.5.3 Evolution zu Many-Core

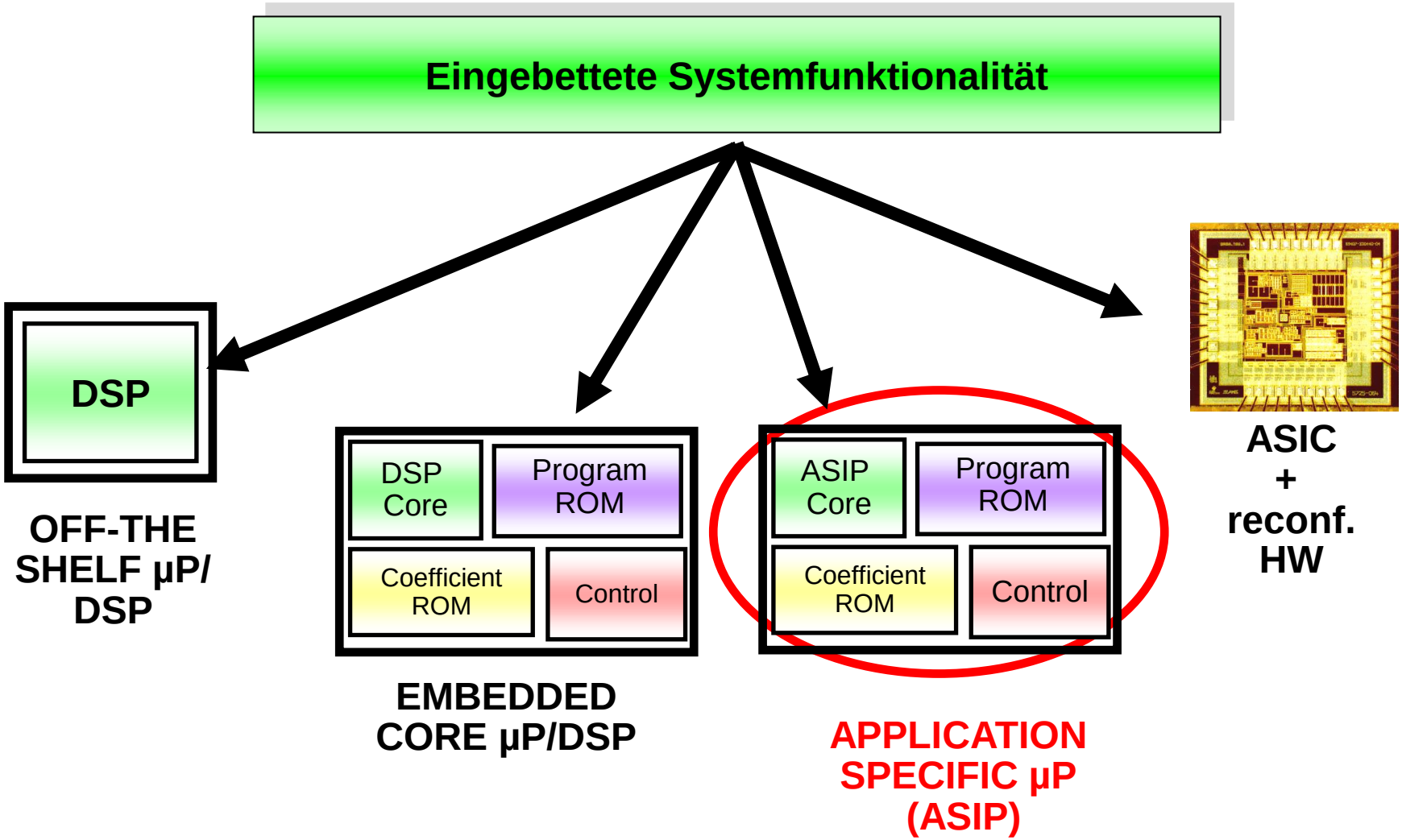
- Basis Theorem
  - Einfachere und kleinere Prozessor Designs benötigen exponentiell weniger Energie um die gleiche Anzahl von Berechnungen zu erreichen im Vergleich zu einem größeren und komplexeren Prozessor Design
- Daumenregel
  - Um die Performanz um 50% zu erhöhen, wird die doppelte Leistung und Chipfläche benötigt
  - Man erreicht schnell einen Punkt ab dem sich der Power Performance Area Trade-off nicht mehr lohnt

2.5.3 Herausforderung von Many-Core

- Kleinere Prozessoren können nicht das gleiche Single-Thread Programm in der gleichen Zeit wie ein High-Performance Prozessor ausführen
  - Vorhandene Applikationen können nicht effizient ausgeführt werden
- Viele Threads können nicht einfach auf einfachere, kleinere Tasks runtergebrochen werden, um vom Multiprocessing auf den kleineren Prozessoren profitieren zu können
- ➔ Software Entwicklungsherausforderung

- Welche Arten von Kommunikationsprotokollen gibt es?
- Aus welchen Komponenten besteht ein Bussystem? Wie funktioniert es?
- Was sind Limitierungen von Multicore Architekturen?
- Wie kann eine höhere Performanz erreicht werden?
- Was sind Herausforderungen von Manycore Systemen?
- Was ist ein Network-On-Chip? Wie ist es aufgebaut? Wie funktioniert es?

2.6 Implementierungsalternativen hochperformanter Algorithmen



2.6 Application-Specific Instruction Set Processors (ASIP)

- Anwendungs-orientierte Spezialisierung
  - des Instruktionssatzes für eingeschränkte Klasse von Anwendungen
    - Bsp.: Operatorverkettung
      - Multiply-Accummulate -> DSP
      - Filter-/Signalverarbeitungsoperationen -> Audio-/Video-Prozessoren
  - der Funktionseinheiten
    - Bsp.: Pixel-Operationen, 1/sqrt(x)
  - der Speicherarchitektur
    - Bsp.: mehrere Speicherbänke mit parallelem Zugriff

- Vorteile gegenüber anderen Prozessoren
  - höhere Performanz
  - niedrigere Kosten (kleinere Chipfläche, weniger Pins)
  - kleinere Codegröße -> mächtigere Instruktionen
  - geringere Leistungsaufnahme

2.6 ASIP: Klassifizierung nach Eigenschaften (I)

- Datentypen
  - Festkomma- oder Gleitkomma-Arithmetik, spezielle Bitbreiten
- Codetyp
  - Mikrocode, alle Instruktionen benötigen einen Maschinenzyklus
  - Makrocode, Instruktion benötigen mehrere Zyklen -> Instruktionspipelining
- Speicherorganisation
  - Load/Store-Architekturen oder Mem/Register-Architekturen
  - ASIPs besitzen häufig keinen Cache
    - Speicher (RAM, ROM, Register) werden meistens on-Chip realisiert
  - Registerstruktur entweder heterogen oder homogen
    - homogen: Register universell für alle Operationen einsetzbar
    - heterogen: verschiedene Registersätze, je nach Operation

2.6.1 ASIP: Klassifizierung nach Eigenschaften (II)

- Instruktionsformat
  - einzeln codiert
    - Felder in Abhängigkeit von Opcode interpretiert, oder
  - orthogonal codiert
    - Bsp. VLIW: Instruktionsteile voneinander unabhängig
      - erlaubt Ansteuerung von unabhängigen Funktionseinheiten

- Besonderheiten
  - spezielle arithmetische Einheiten,
  - spezielle Datentypen,
  - besondere Adressierungsarten,
  - HW-Unterstützung von Schleifenkonstrukten,
  - ...

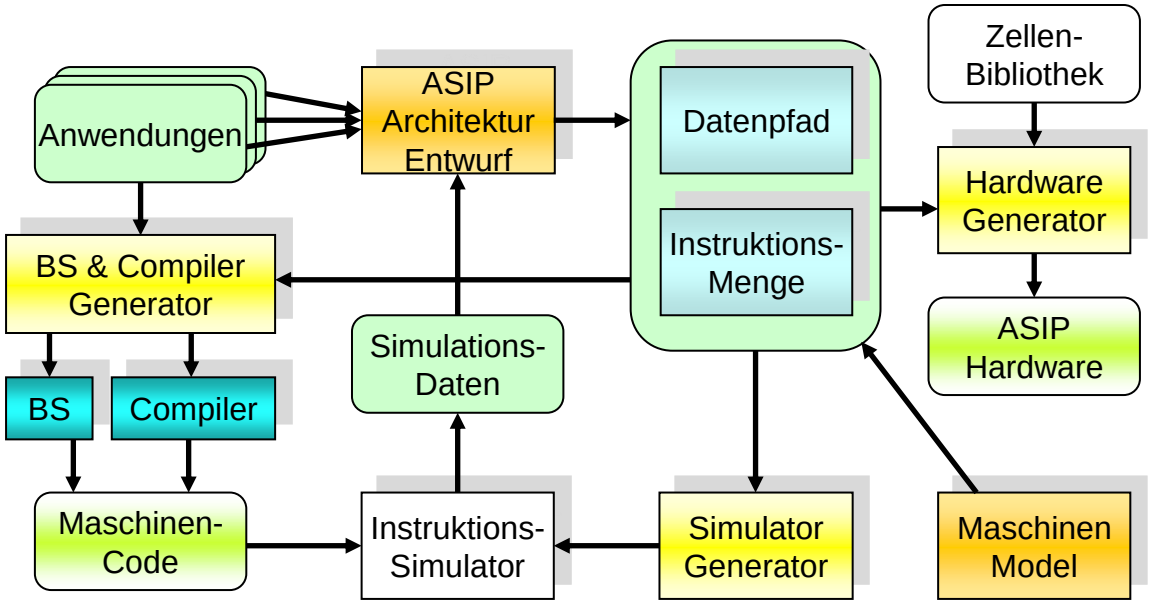
2.6 ASIP: Beispiele

- Da ASIPs sehr anwendungsspezifisch sind und oft speziell für einzelne Anwendungen entwickelt worden sind, sind diese i.a. nicht auf dem kommerziellen Markt erhältlich
- Beispiel: Network Processors -> oft firmeninterne Entwicklungen

Company	Product	RISC based	Task Specific Processor based	ASIC
Level One	*IXP1200*	✔		
IBM	*PNP*	✔		
MMC	*nP*	✔		
Maker	*MXT*	✔		
Sitera	*PRISM IQ1200*	✔		
EZChip	*NP-1*	✔		
C-Port	*C-5 DCP*	✔		
Agere	*PayloadPlus*		✔	
Fast-chip	*PolicyEdge*		✔	
Hi-fn	*7711, 7751*		✔	
Xaqti	*TeraPower-CL*		✔	
Broadcom	*StrataSwitch*		✔	
Solidum	*PAX.port 1100*		✔	
Netlogic	*Policy, CIDR*		✔	
Switchcore	*CXE*		✔	
Entridia	*Opera*			✔

2.6 ASIP: angestrebter Entwurfsablauf (I)

- Erzeugung der ASIP-Hardware
  - Wenn simulierte ASIP-Hardware allen funktionalen und zeitlichen Anforderungen genügt
    - Hardwarerealisierung wird erzeugt
  - Hardwaregeneratoren verwenden diverse Bibliotheken mit Prozessorkernen, Bus-Interfaces, Speichern, ALUs, Arithmetik in Pipelineausführung, Spezialoperatoren, Zählern, Registern usw.



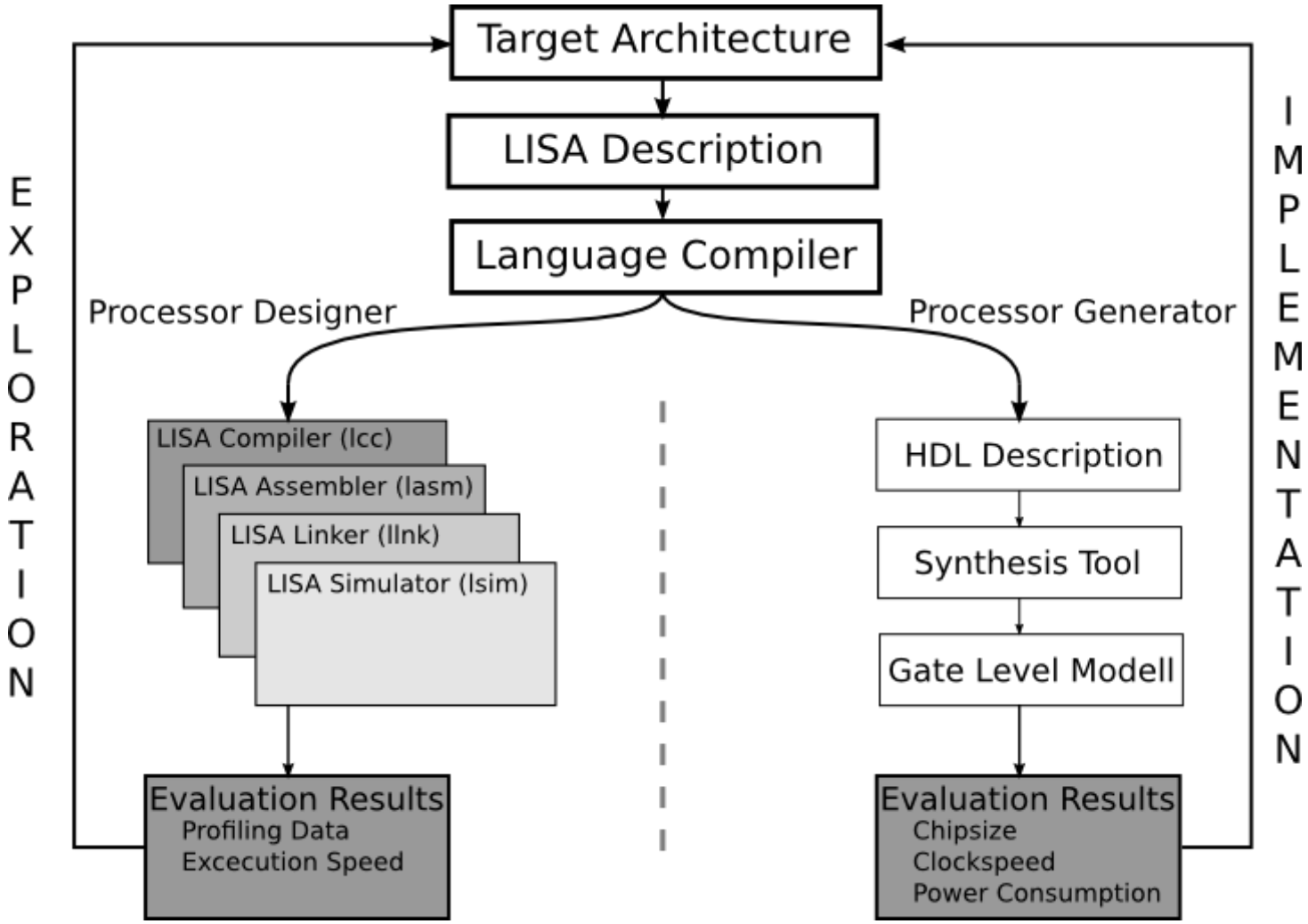
2.6 ASIP: angestrebter Entwurfsablauf (II)

- Architektur-Entwurf
  - Identifizierung neuer Instruktionssätze / Datenpfade aus Obermenge von Instruktionen für Standard-Arithmetik, Speicher- und Kontrollflußinstruktionen
  - Besondere Instruktionen verfügbar
    - z.B. digitale Filterung oder Viterbi-Dekodierung
  - Anpassung des Datenpfads + Instruktionen (+ Speicherstruktur)
    - erfolgt aus Informationen, die aus Anwendungs-Benchmarks und zu erwartenden Daten abgeleitet werden können (Nebenläufigkeit!)
    - Nebenbedingungen wie Ausführungsgeschwindigkeit, Fläche und Leistungsaufnahme sind zu berücksichtigen
  - Ein abstraktes Maschinenmodell (ggf. mit Pipelining) dient dabei als Entscheidungsgrundlage für die Hardwarearchitektur

2.6 ASIP: angestrebter Entwurfsablauf (III)

- Ziel: Erzeugung des Betriebssystems und Compilers für ASIPs
  - Compiler erzeugt den Maschinencode für abgeleitete ASIP-Architektur
    - generiert aus einer Hochsprachenbeschreibung z.B. C
  - Codeerzeugung ist auf andere Zielarchitekturen retargierbar
    - die in HDL gegeben sind (Instruktionsmuster, Ressourcen, Verbindungen)
  - Flexible Anpassung der Codeerzeugung auf neue / geänderte Zielarchitekturen
    - für den ASIP-Entwurf sehr vorteilhaft

2.6 ASIP Beispiel: Synopsys Processor Designer

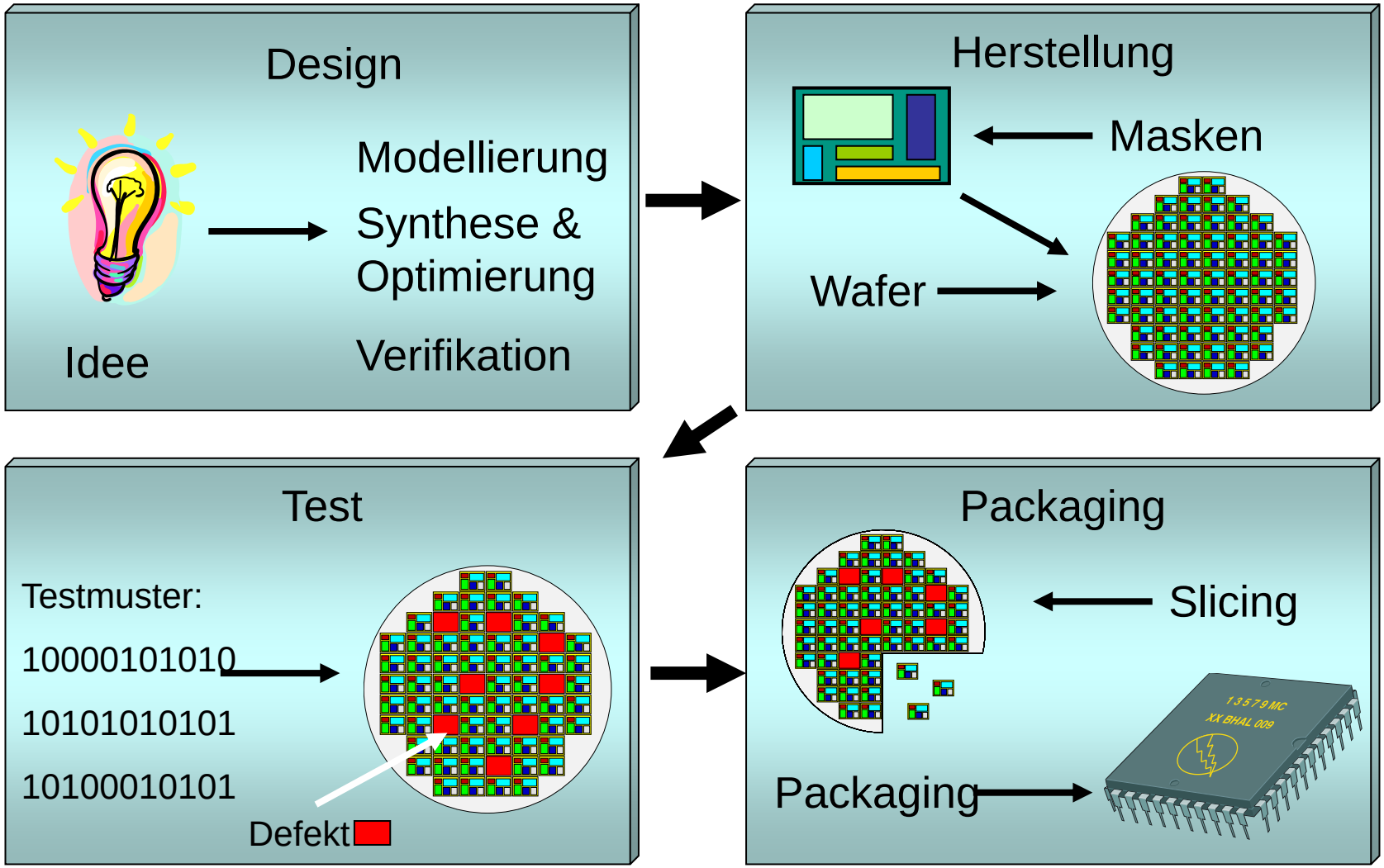


- Was ist ein ASIP?
- Was sind dessen besondere Eigenschaften?
- Welche Vor- oder Nachteile haben ASIPs gegenüber anderen Prozessoren?
- Wie wird ein ASIP entworfen?

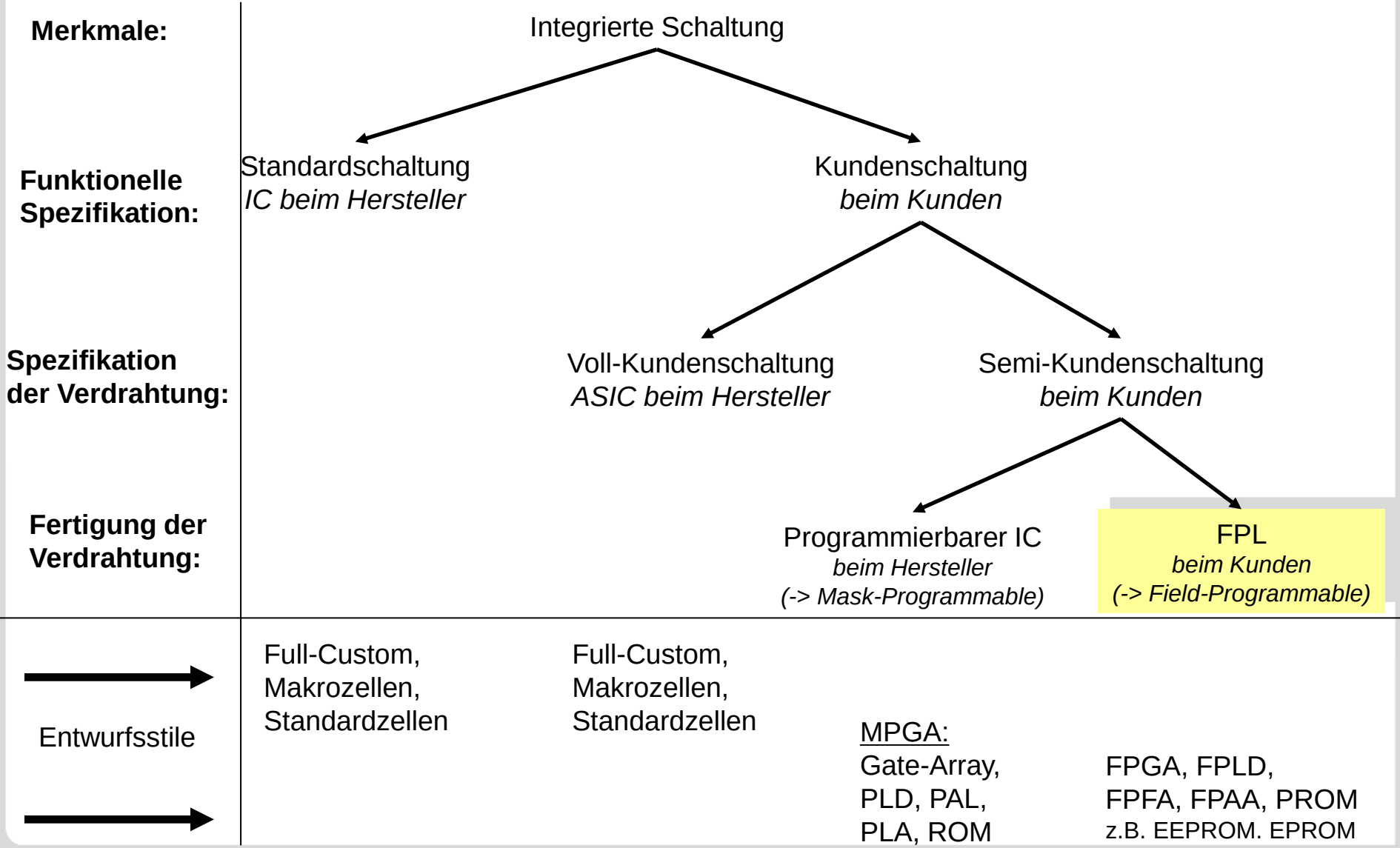
Inhalt

- 2.4 Spezialprozessoren
  - 2.4.1 Mikrocontroller (µC)
  - 2.4.2 Digitale Signalprozessoren (DSPs)
  - 2.4.3 Grafik Prozessoren (GPU)
- 2.5 Bus, Network-on-Chip (NoC) & Multicore
- 2.6 Anwendungs-spezifische Instruktionssatz Prozessoren (ASIP)
- **2.7 Field Programmable Gate Arrays (FPGA)**
- 2.8 System-on-Chip (SoC)

2.7.1 Entwurfsstile: Phasen bei Integrierten Schaltungen



2.7.1 Entwurfsstile für ICs



2.7.1 Vergleich der wichtigsten Entwurfsstile

	Custom	Cell-based	MPGA	FPGA
Dichte	Sehr hoch	Hoch	Hoch	Medium bis niedrig
Performance	Sehr hoch	Hoch	Hoch	Medium bis niedrig
Entwurfszeit	Sehr lange	Kurz	Kurz	Sehr kurz
Fabrikationszeit	Medium	Medium	Kurz	Sehr kurz
Kosten bei geringer Stückzahl	Sehr hoch	Hoch	Hoch	Niedrig
Kosten bei hoher Stückzahl	Niedrig	Niedrig	Niedrig	Hoch

2.7.1 Evolution der Implementierungstechnologien

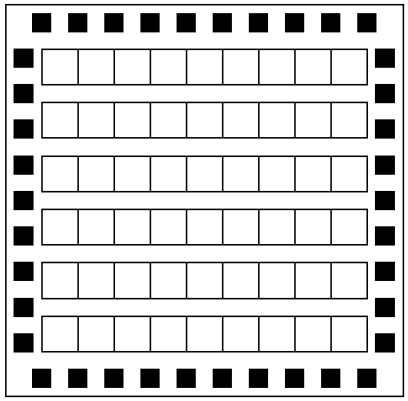
- Diskrete Bauteile: Relais, Transistoren (1940‘er -50‘er)
- Diskrete Gatter-Logik (1950‘er-60‘er)
- Integrierte Schaltungen (1960‘er-70‘er)
  - z.B. TTL (Transistor-Transistor-Logik): Datenbuch für Hunderte verschiedener Blöcke
  - Abbildung der Schaltung auf die TTL-Blöcke aus dem Katalog
- Gate-Arrays (IBM 1970‘er)
  - “Kundenspezifische” integrierte Schaltungs-Chips
  - Design unter Verwendung einer Bibliothek (TTL-Library)
  - Transistoren sind bereits auf dem Chip integriert
  - Place & Route Software personalisiert den Chip automatisch
    + Große Schaltungen auf einem Chip
    + Automatische Design-Tools (kein langwieriges Custom-Layout)
    - Nur gut bei sehr großen Stückzahlen

Trend in Richtung höherer Integration

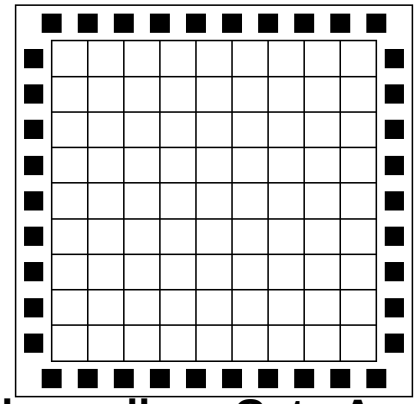
2.7.2 Gate-Array Technologie

- Einfache Logik Gatter
  - Gebrauch von Transistoren für kombinatorische und sequentielle Logik
    - Pre-Diffused Technologie, vor der Personalisierung
- Personalisierung
  - „Verdrahtung“ zur Verbindung der Ein-/Ausgänge der vorgefertigten Komponenten (Transistoren / Gatter)
- I/O-Blöcke
  - Spezielle Peripherieblöcke für externe Verbindungen
- Hinzufügen von Verdrahtung, um Verbindungen zu schaffen
  - Während der Chipherstellung
    - -> Kundenspezifische “Masken-Programmierung”
  - Erzeugung beliebiger Schaltungen durch anwendungsspezifische Metallisierung

2 Typen von Gate-Arrays:



Channeled Gate-Array



Channelless Gate-Array (*Sea-of-Gates*)

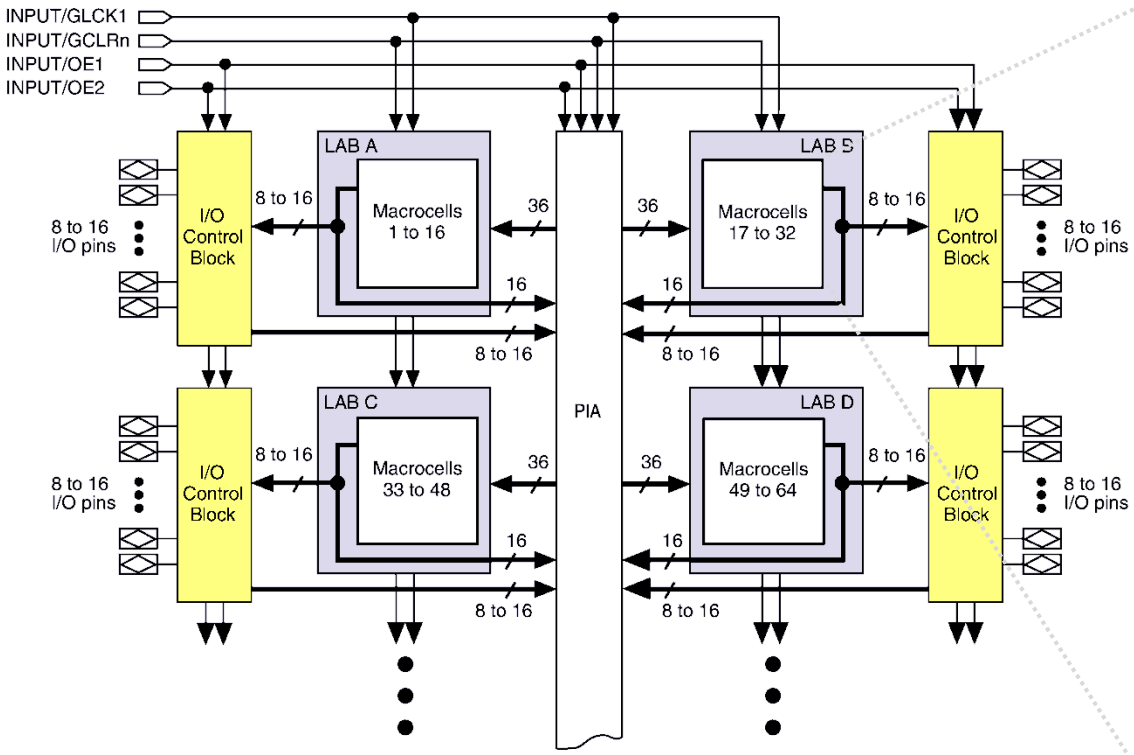
2.7.2 Einmal Programmierbare Logik – Fuse und Anti-Fuse

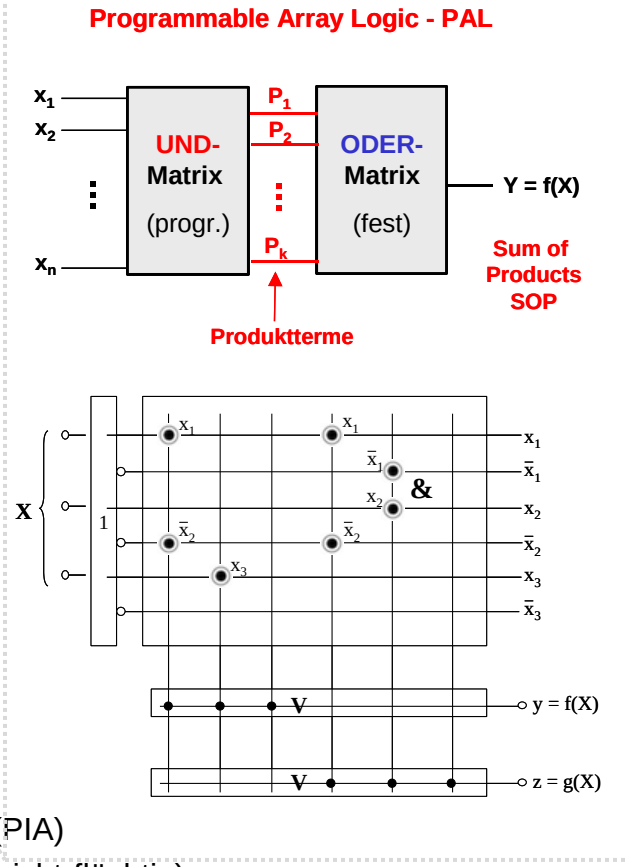
- „Schmelzsicherung“ unterbricht (-> Fuse) oder erzeugt (-> Anti-Fuse) Verbindung zwischen zwei Verdrahtungslagen
- Typische Verbindungen liegen bei 50-300 Ohm
- **Einmalige** Programmierbarkeit (Tests vor der Programmierung?)
- Erlaubt sehr hohe Integrationsdichte
- Weniger empfindlich gegenüber ionisierender Strahlung (z.B. Weltraumeinsatz)

2.7.2 Mehrfach Programmierbare Logik – RAM-basiert

- Gesetzte Bits in Look-up-Tables (LUTs) realisieren Logikfunktionen
- Gesetzte Bits in einer FlipFlop-Zelle (FF) kontrollieren Schalter, die unterschiedliche Leitungen trennen / verbinden (-> Verdrahtung)
- Typische Verbindungen liegen bei 0.5K - 1K Ohm
- Können mehrfach (re-) programmiert werden (auch teilweise dynamisch, während des Betriebs)
  - Dynamische (partielle) Rekonfiguration
- Geringe Integrationsdichte, strahlungsempfindlich (spezielle Typen verfügbar.)

2.7.3 Feinkörnig Rekonfigurierbare Hardware: Altera MAX 7000 CPLD

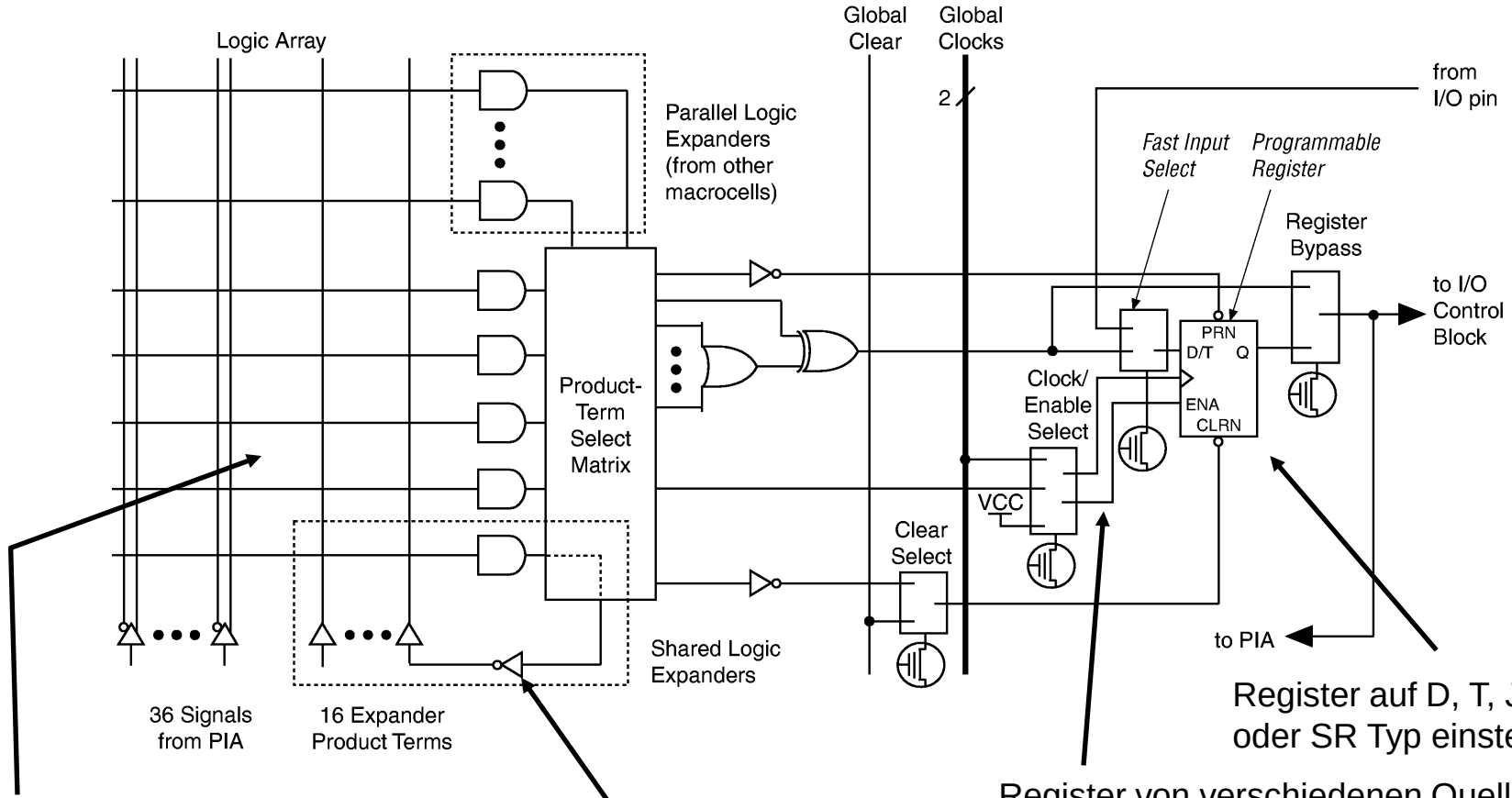




- Logic Array Blocks (LAB), I/O Blocks, und Programmable Interconnect Array (PIA)
- Programmierbarkeit basiert auf EEPROM Floating-Gate (Flash) Technology (nicht-flüchtig)
- 16 Makrozellen in jedem LAB
- Flip-Flops + kombinatorische Basis-Logik
- Programmierbares Interconnect-Array enthält Busse, welche die Eingänge/Ausgänge der Makrozellen verbinden.
- I/O Control-Blöcke verbinden alle benachbarten LABs sowie extern zu den Pins.

2.7.3 Altera programmierbare Makrozelle

- Ein LAB besteht bei der MAX 7000 aus 16 Makrozellen



Register auf D, T, JK oder SR Typ einstellbar.

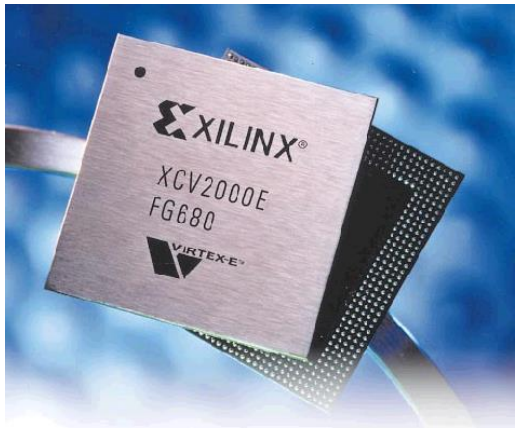
Logic Array erlaubt 5 Produktterme je Makrozelle

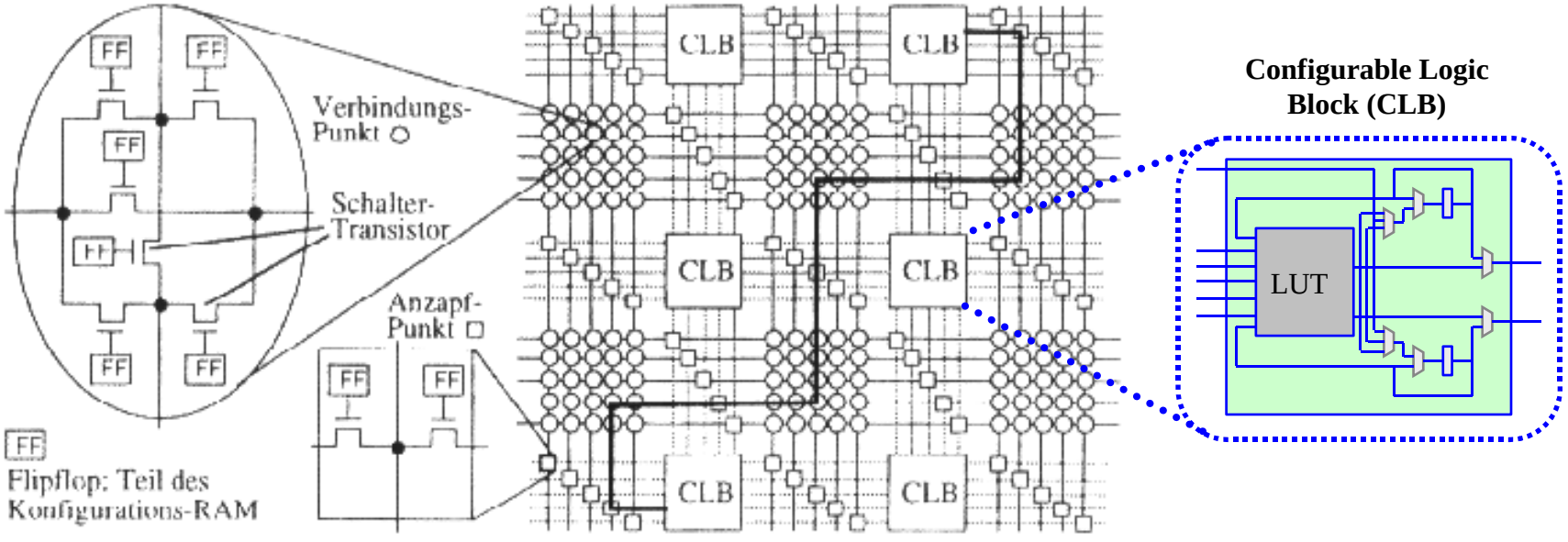
Rückkopplung auf den Eingang, auch von den anderen Makrozellen

Register von verschiedenen Quellen taktbar (lokal, global ggf. mit Enable)

2.7.3 Field Programmable Gate Array: Struktur

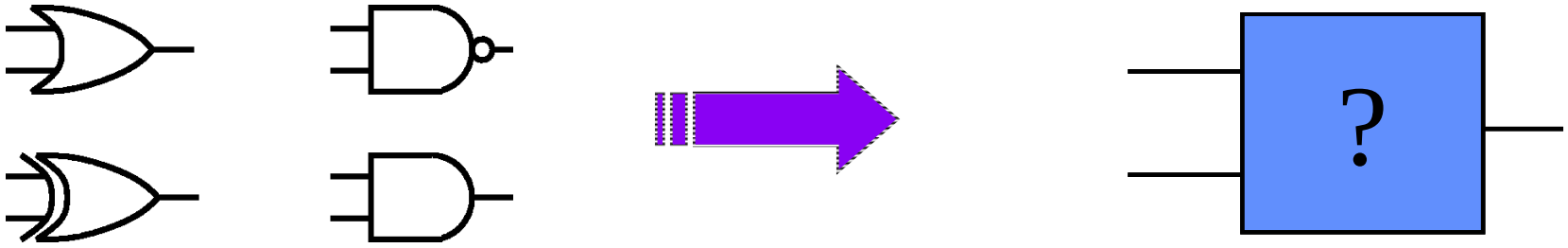
- FPGA-Architekturen
  - SRAM-basierende Look-up Tables (LUTs)
  - Probleme:
    - Verdrahtung: reduziert Performanz
    - Verhältnis: aktive / passive Elemente
- rekonfigurierbare Verbindungen (Switch Matrix)





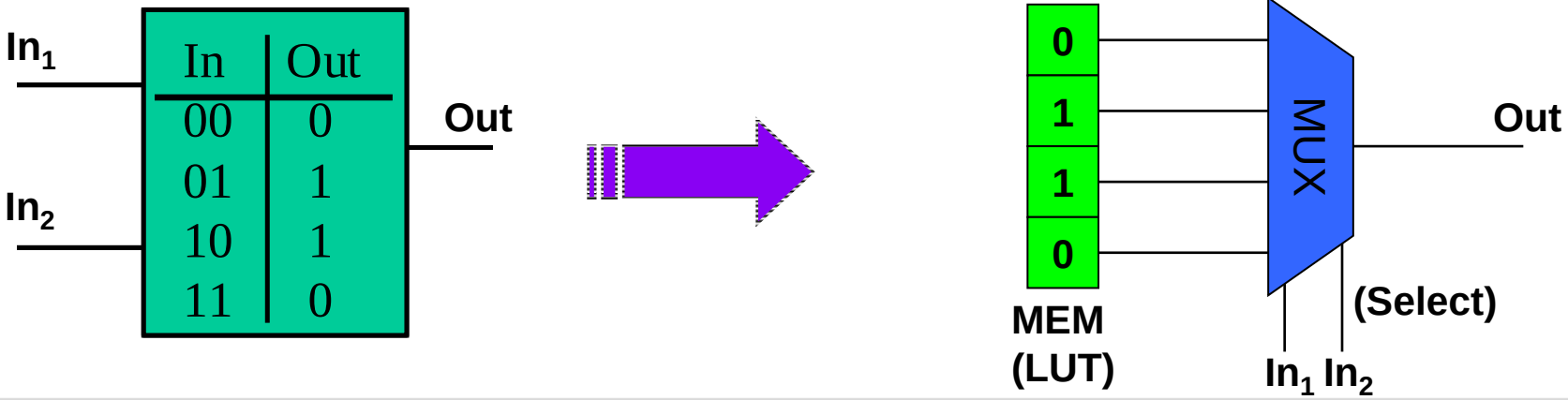
2.7.3 RAM-basierter FPGA : Logikrealisierung

- Menge programmierbarer “Logikgatter”, integriert in ein flexibles Verbindungsnetzwerk
  - eine “benutzerprogrammierbare” Alternative zu dedizierten Schaltungen

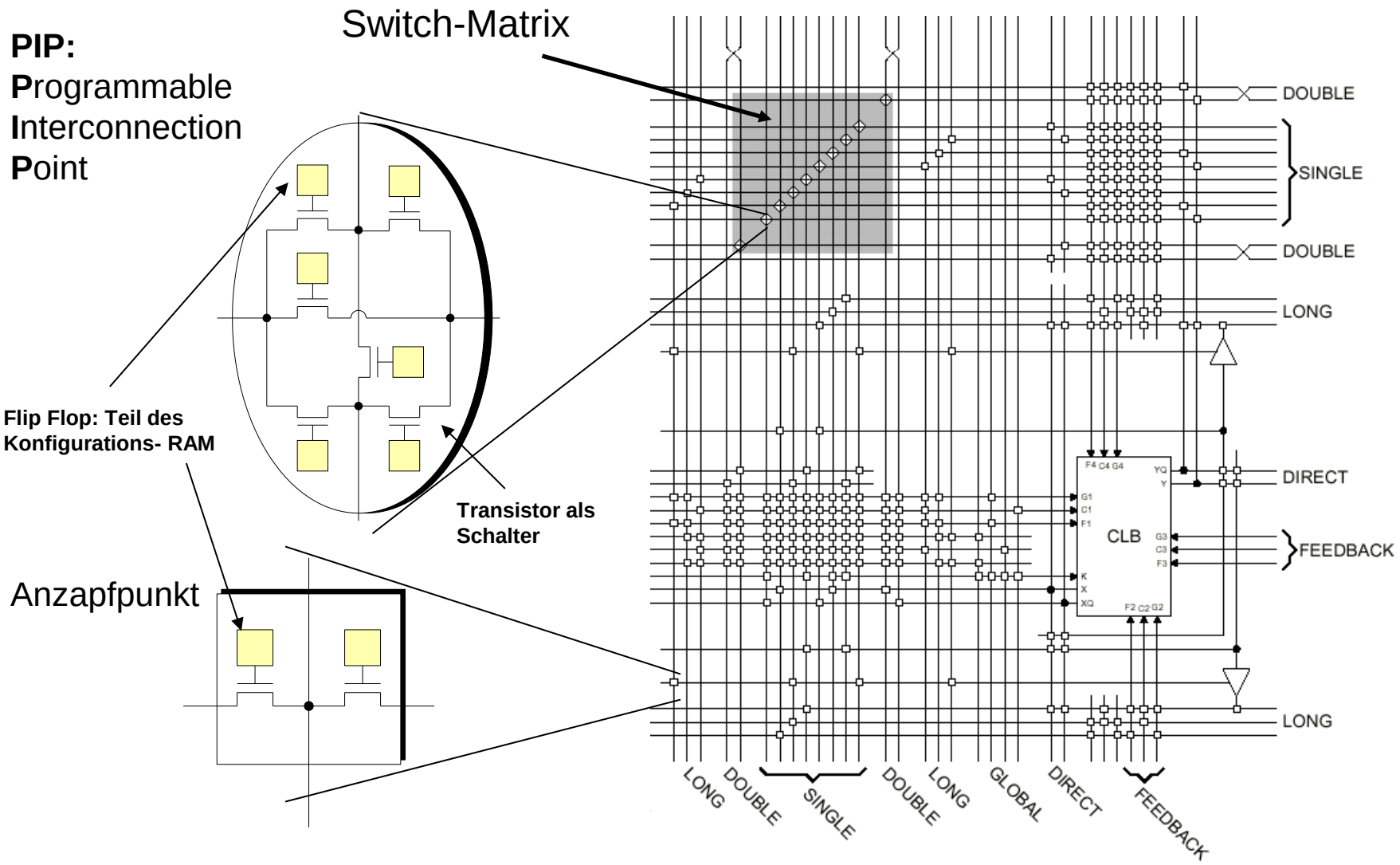


Programmierbares Gatter

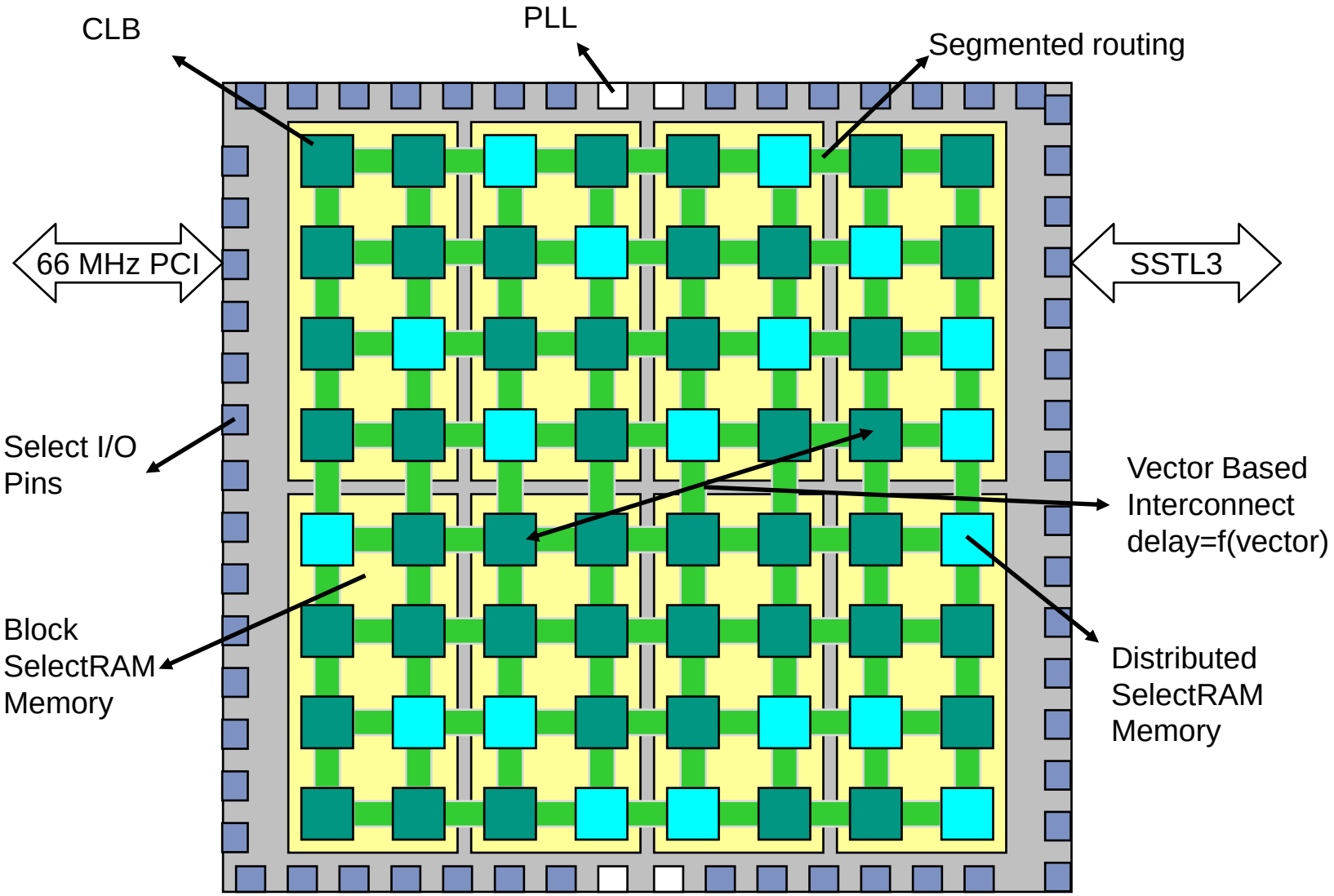
- Lösung:
  - Look-up-Table (LUT) mit 2 Eingängen



2.7.3 FPGA-Verbindungsstruktur mit PSM Switch-Matrix



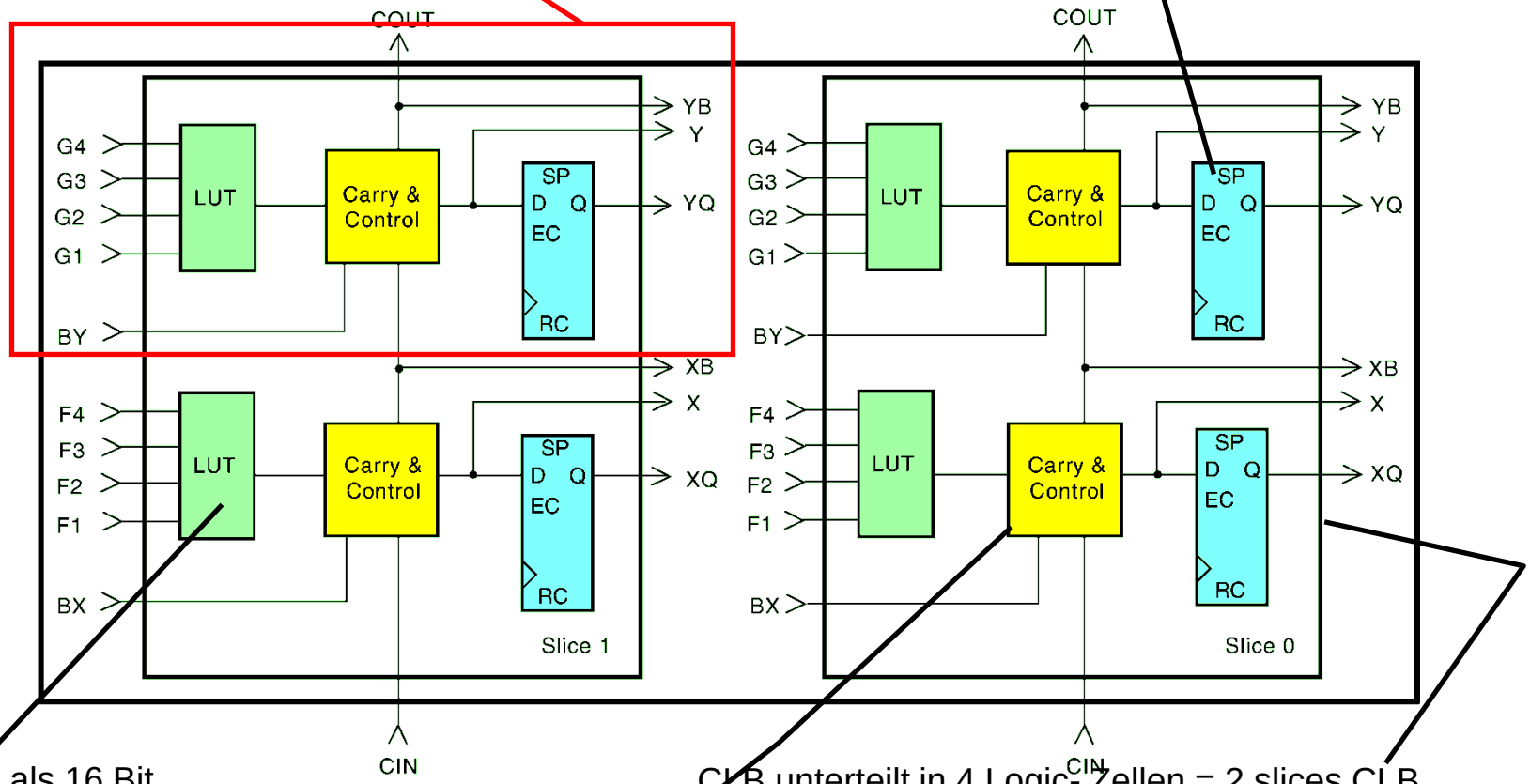
2.7.4 Xilinx Virtex: Funktionsblock-Architektur



2.7.4 Xilinx Virtex: CLB

Logic Cell
LUT mit 4 Eingängen
Carry-Logik und Speicher

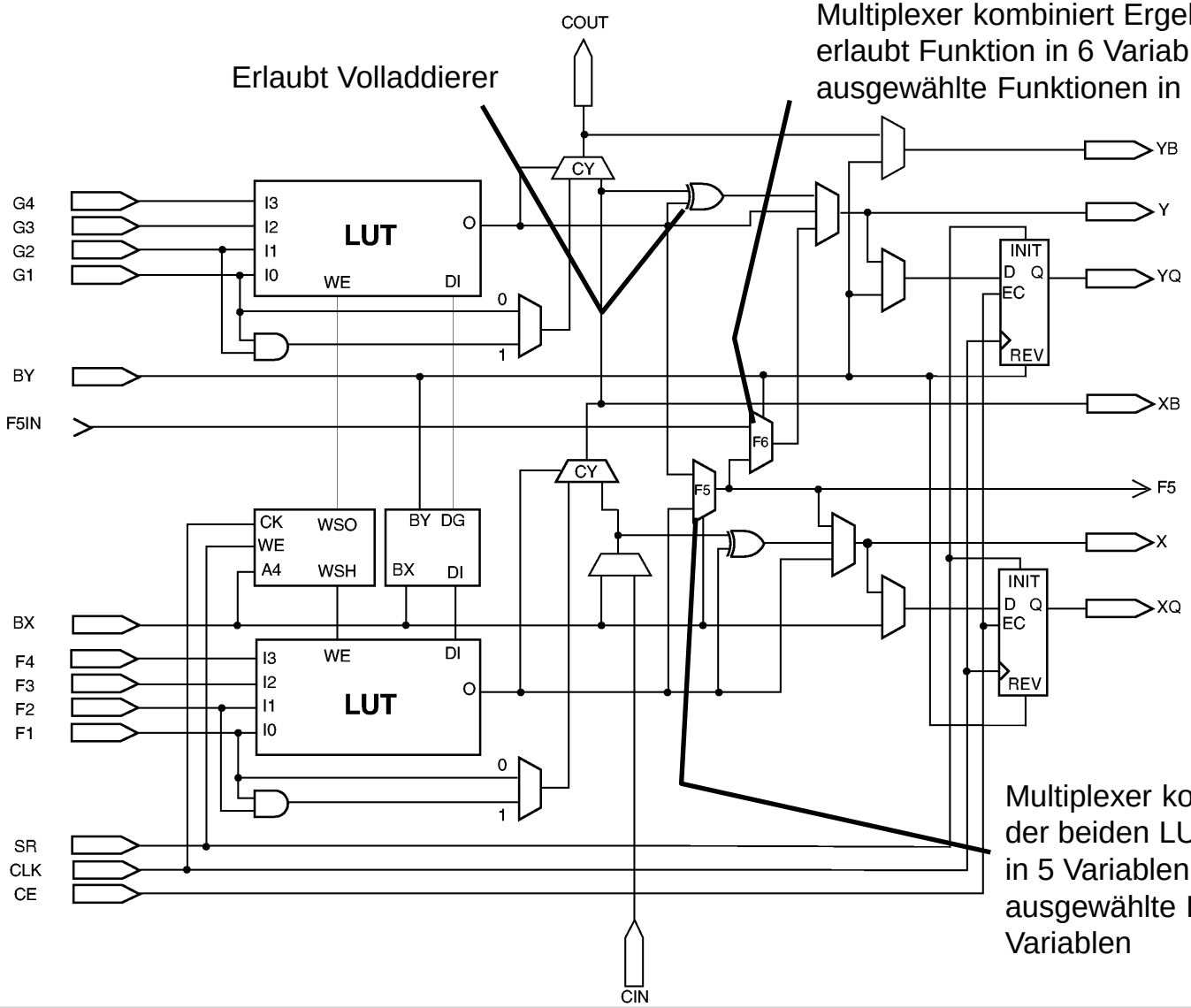
Speicher als D- FlipFlops
oder pegelgesteuerte Latches



LUT auch als 16 Bit
RAM verwendbar

CLB unterteilt in 4 Logic- Zellen = 2 slices CLB
Enthält zusätzliche Logik, um Funktionsgeneratoren
in 5 oder 6 Eingängen zu realisieren.

2.7.4 Detaillierte Ansicht eines Virtex-Slice



2.7.4 FPGA Beispiel: Xilinx 7-Series

- Artix-7 Family:
  - Optimized for lowest cost and power with small form-factor packaging for the highest volume applications.
- Kintex-7 Family:
  - Optimized for best price-performance with a 2X improvement compared to previous generation, enabling a new class of FPGAs.
- Virtex-7 Family:
  - Optimized for highest system performance and capacity with a 2X improvement in system performance. Highest capability devices enabled by stacked silicon interconnect (SSI) technology.

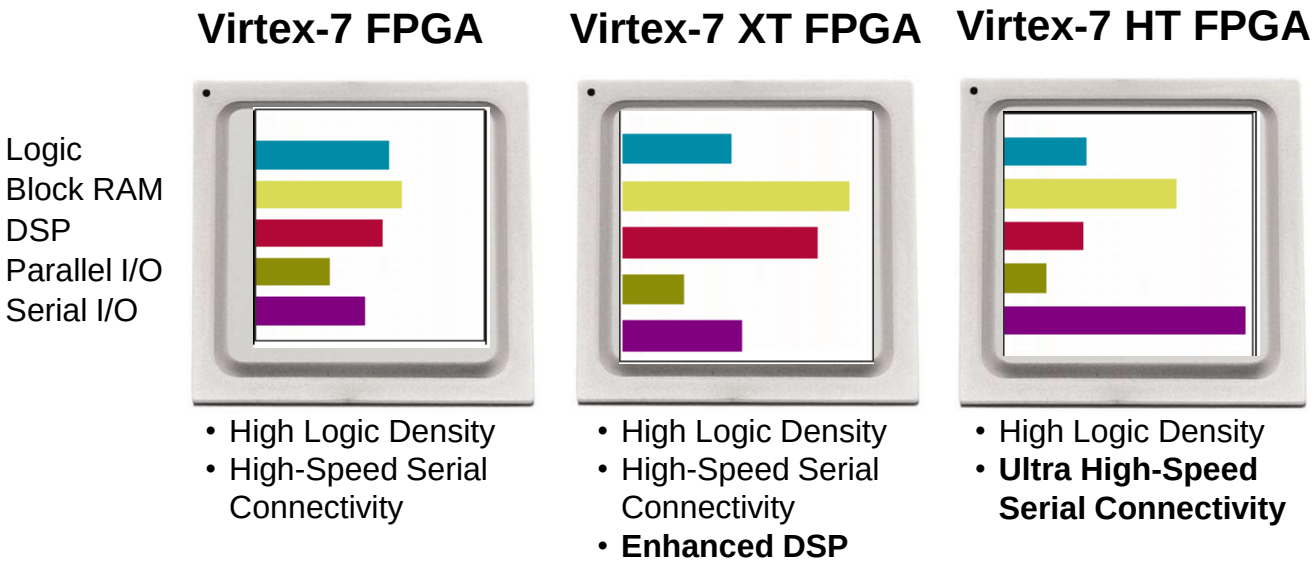
2.7.4 FPGA Beispiel: Xilinx 7-Serie

Maximum Capability	Artix-7 Family	Kintex-7 Family	Virtex-7 Family
Logic Cells	215K	478K	1,955K
Block RAM	13 Mb	34 Mb	68 Mb
DSP Slices	740	1,920	3,600
Peak DSP Performance	929 GMAC/s	2,845 GMAC/s	5,335 GMAC/s
Transceivers	16	32	96
Peak Transceiver Speed	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s
Peak Serial Bandwidth (Full Duplex)	211 Gb/s	800 Gb/s	2,784 Gb/s
PCIe Interface	X4 Gen2	X8 Gen2	X8 Gen2
Memory Interface	1,066 Mb/s	1,866 Mb/s	1,866 Mb/s
I/O Pins	500	500	1,200

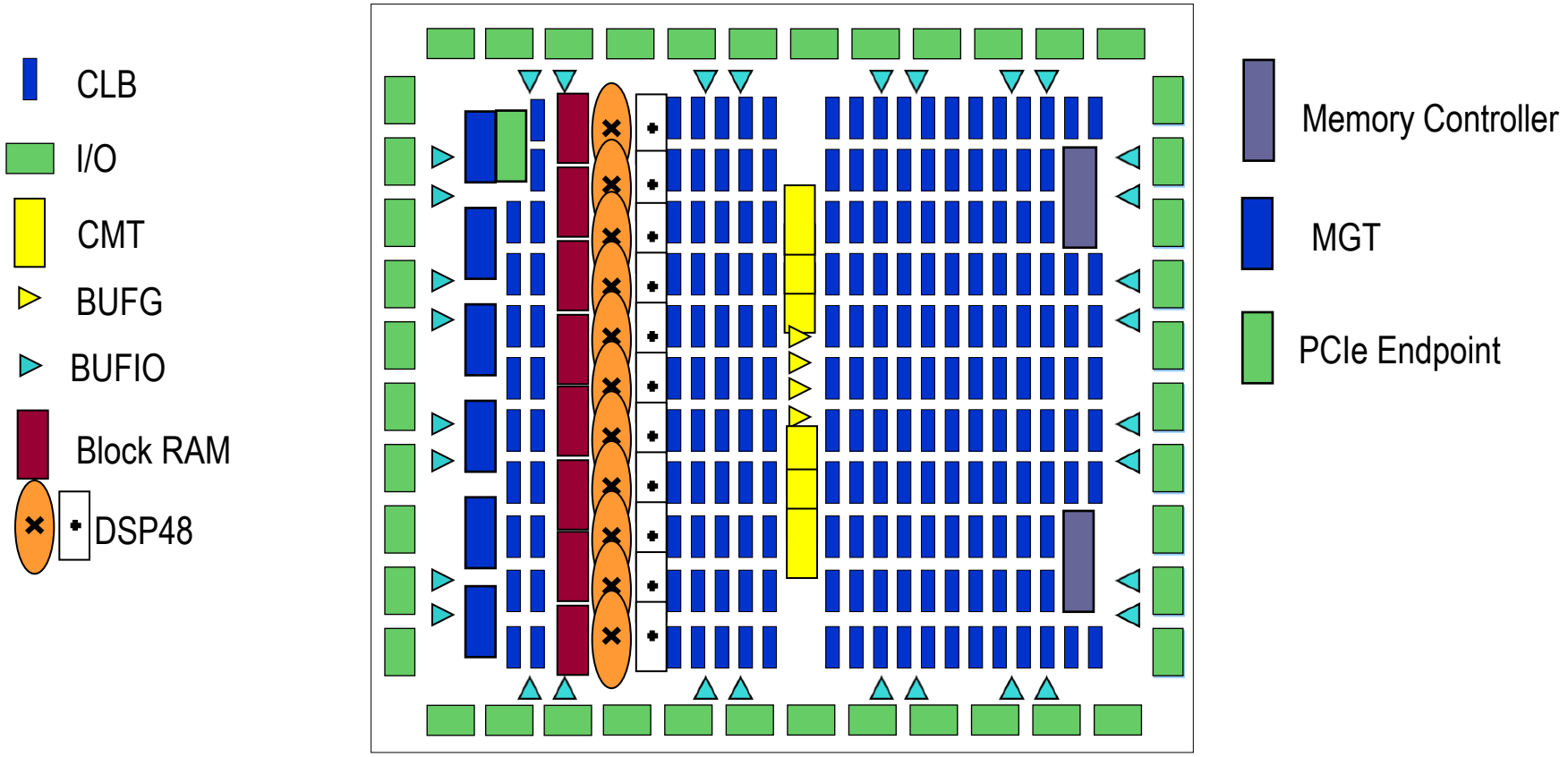
http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

2.7.4 FPGA Beispiel: Xilinx 7-Serie Sub-Familien

- The Virtex-7 family has several sub-families
  - Virtex-7:
    - General logic
  - Virtex-7XT:
    - Rich DSP and block RAM
  - Virtex-7HT:
    - Highest serial bandwidth

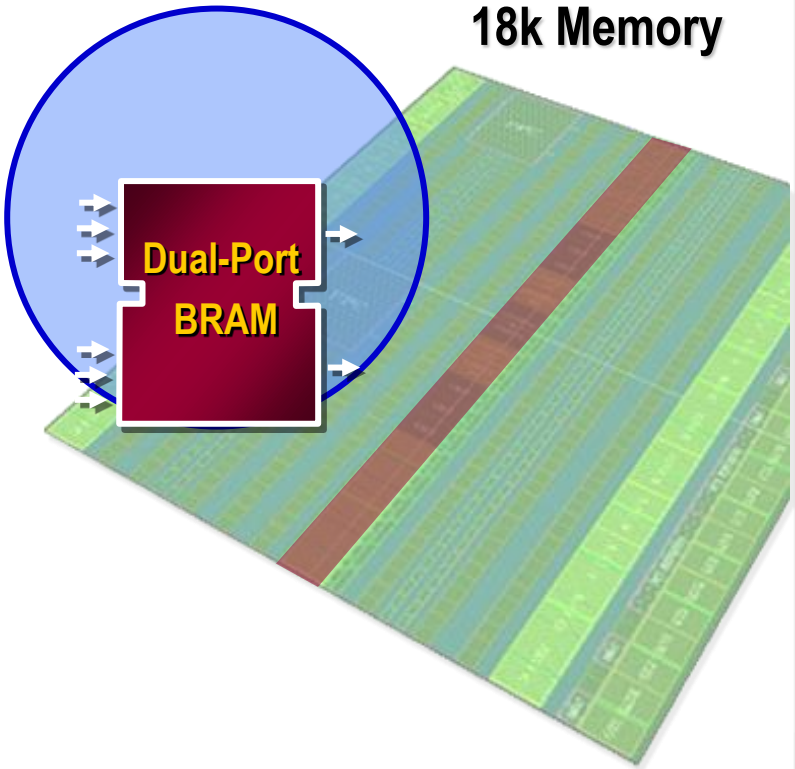


2.7.4 FPGA Beispiel: Spartan-6

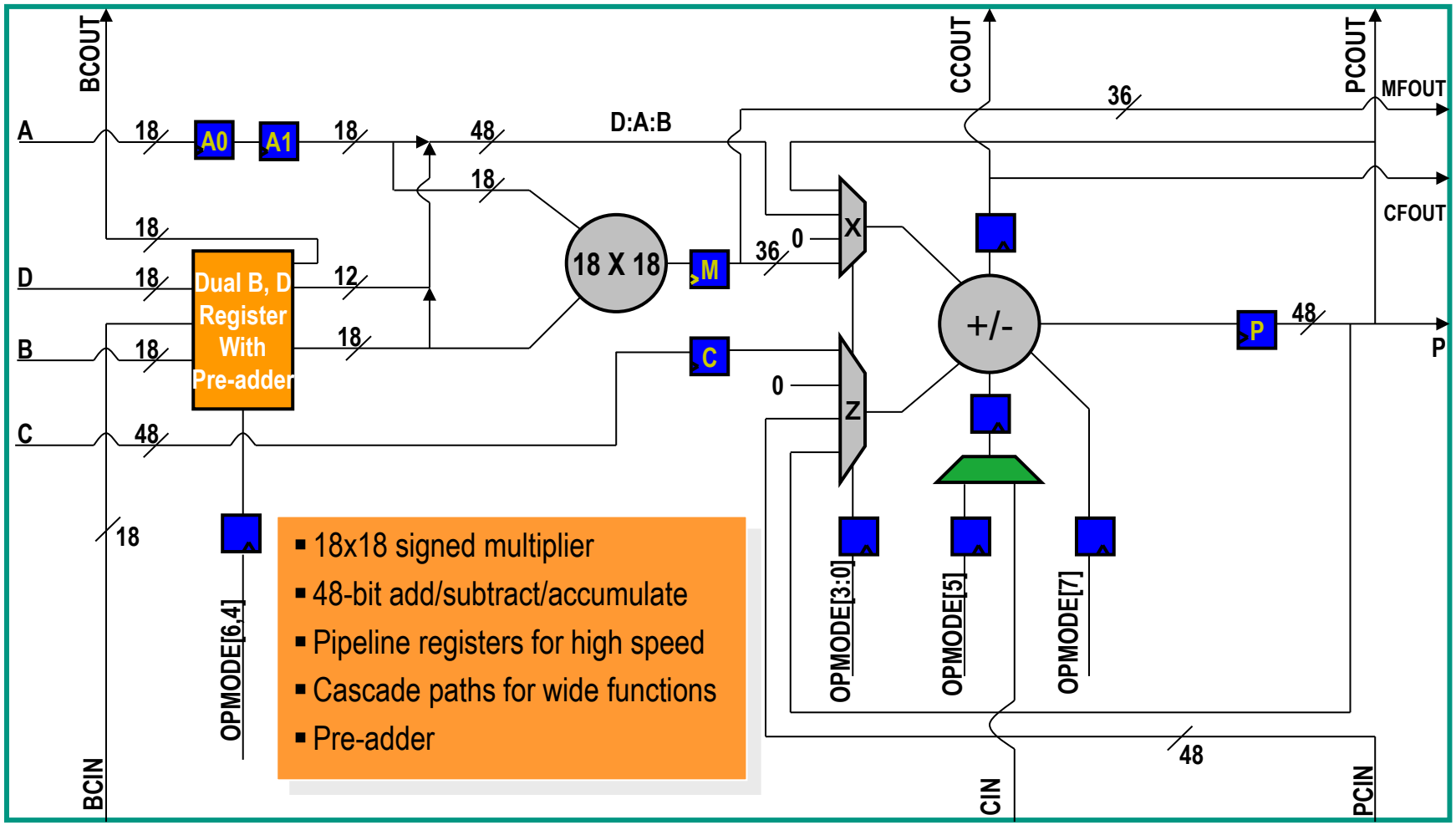


2.7.4 FPGA Beispiel: Spartan-6 Block RAM Features

- 18 kb size
  - Can be split into two independent 9-kb memories
- Performance up to 300 MHz
- Multiple configuration options
  - True dual-port, simple dual-port, single-port
- Two independent ports access common data
  - Individual address, clock, write enable, clock enable
  - Independent widths for each port
- Byte-write enable

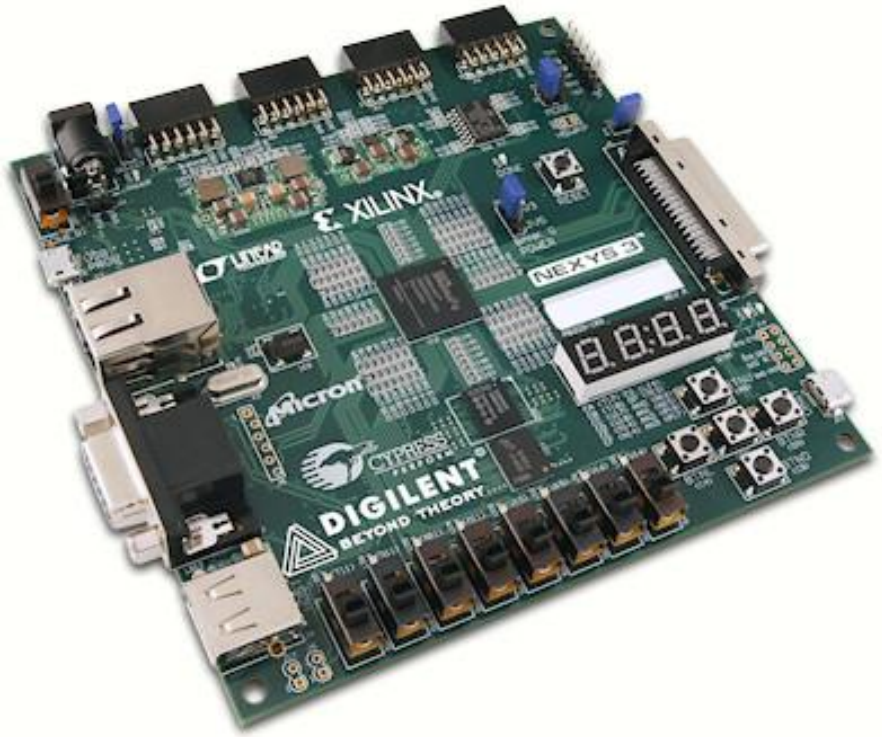


2.7.4 FPGA Beispiel: Spartan-6 DSP48A1 Slice



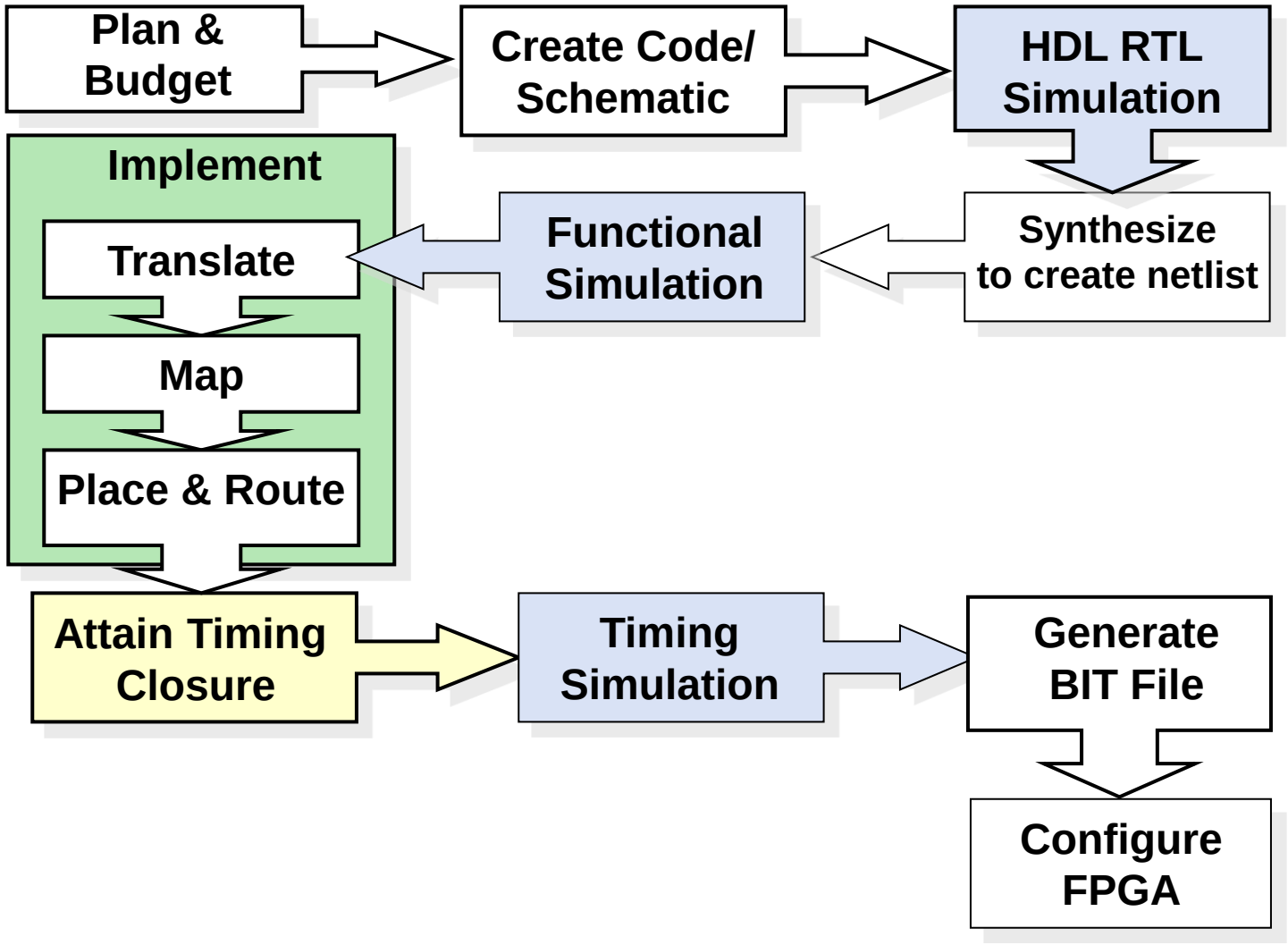
2.7.4 FPGA Beispiel: Nexys3 Spartan-6 FPGA Board

- Xilinx Spartan6 XC6LX16-CS324
- 16Mbyte Micron Cellular RAM
- 16Mbyte Micron Parallel PCM
- 16Mbyte Micron Quad-mode SPI PCM
- 10/100 SMSC LAN8710 PHY
- Digilent Adept USB port for power, programming & data transfers
- USB-UART
- Type-A USB host for mouse, keyboard or memory stick
- 8-bit VGA
- 100MHz fixed-frequency oscillator
- 8 slide switches, 5 push buttons, 4-digit 7seg display, 8 LEDs
- Four double-wide Pmod™ connectors, one VHDC connector
- Rugged plastic case, USB cable included



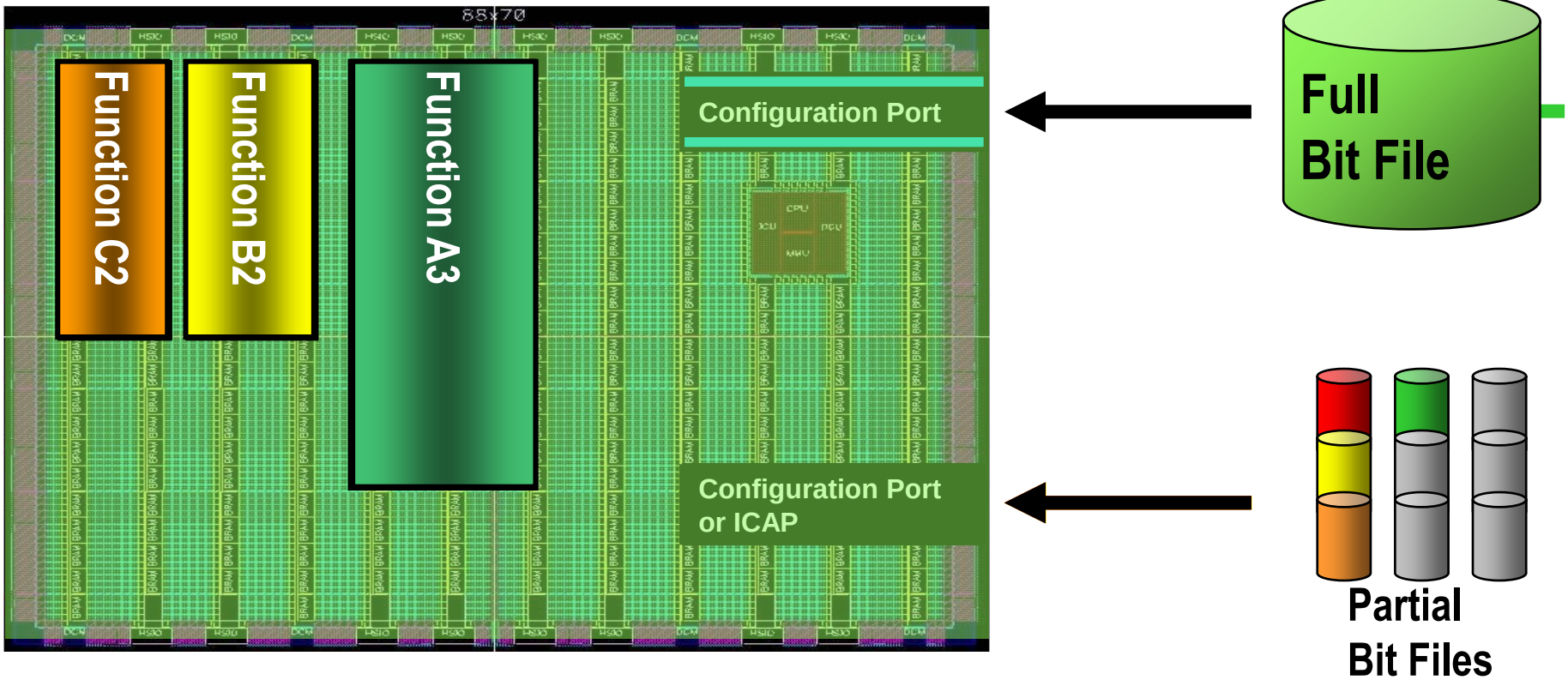
http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,897&Prod=NEXYS3

2.7.5 FPGA Design Flow



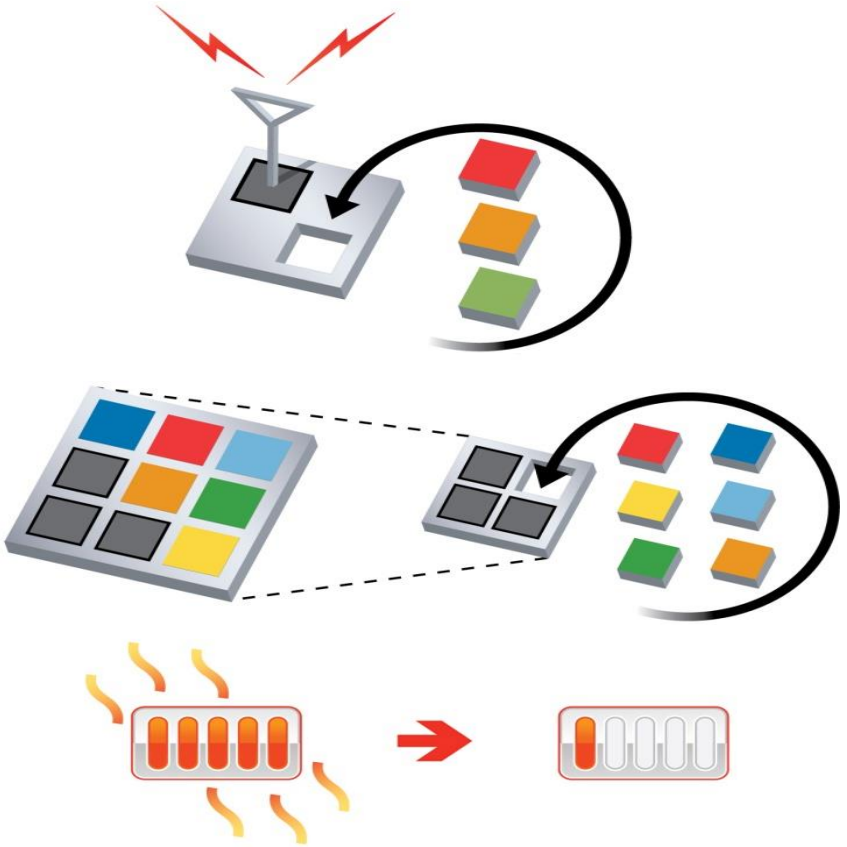
2.7.6 Was ist partielle Rekonfiguration?

- Partial Reconfiguration is the ability to dynamically modify blocks of logic by downloading partial bit files while the remaining logic continues to operate without interruption.



2.7.6 Partielle Rekonfiguration – Technologie & Nutzen

- Partial Reconfiguration enables:
  - System Flexibility
    - Perform more functions while maintaining communication links
  - Size and Cost Reduction
    - Time-multiplex the hardware to require a smaller FPGA
  - Power Reduction
    - Shut down power-hungry tasks when not needed



2.7.6 Einsatzgebiete von rekonfigurierbarer Hardware

- Glue Logic (Interfaces)

- Rapid Prototyping, Emulation
  - Ensemble von Gate-Arrays, die zur Emulation einer Schaltung vor der eigentlichen Herstellung erzeugt werden
  - Erlaubt schnelleres und besseres Debugging als mit reiner Simulation

- Eingebettete Systeme
  - schneller als Prozessor & flexibler als ASIC
  - ASIC Replacement
    - Stückzahl zu gering für ein ASIC
    - verkürzte Time-to-market
  - Prozessor-/Datenpfad-Replacement
    - Controller/Datenpfad als Teil eines Configurable Systems-on-Chip (CSoC)

- Custom Computing
  - Ziel: Verbinden der Flexibilität von Prozessoren mit der Performanz von ASICs
  - Hardwarebeschleunigung von Algorithmen

- Welche Entwurfsstilte gibts es? Was sind Vor- und Nachteile?
- Welche Gate-Array Technologien gibt es?
- Welche Technologien gibt es für programmierbare Logik? Welche Vor- bzw Nachteile haben diese?
- Wie sieht eine FPGA-Architektur aus?
- Wie wird die Logik bei SRAM-basierten FPGAs abgebildet?
- Was ist (partielle) dynamische Rekonfiguration?

Inhalt

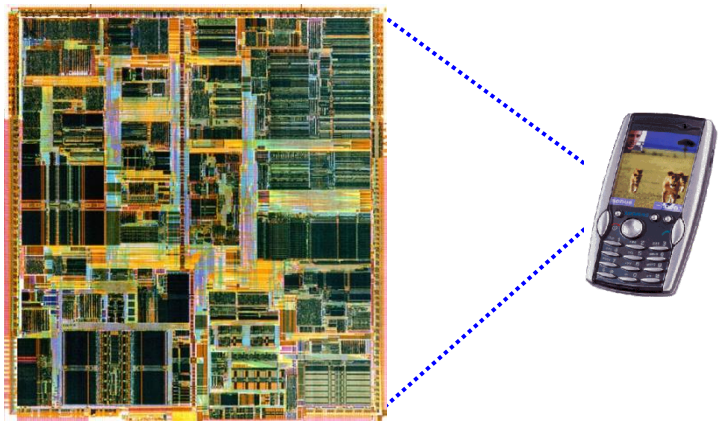
- 2.4 Spezialprozessoren
  - 2.4.1 Mikrocontroller (µC)
  - 2.4.2 Digitale Signalprozessoren (DSPs)
  - 2.4.3 Grafik Prozessoren (GPU)
- 2.5 Bus, Network-on-Chip (NoC) & Multicore
- 2.6 Anwendungs-spezifische Instruktionssatz Prozessoren (ASIP)
- 2.7 Field Programmable Gate Arrays (FPGA)
- **2.8 System-on-Chip (SoC)**

2.8.1 System-on-Chip (SoC): Kostenmodelle + Realisierungsbewertung

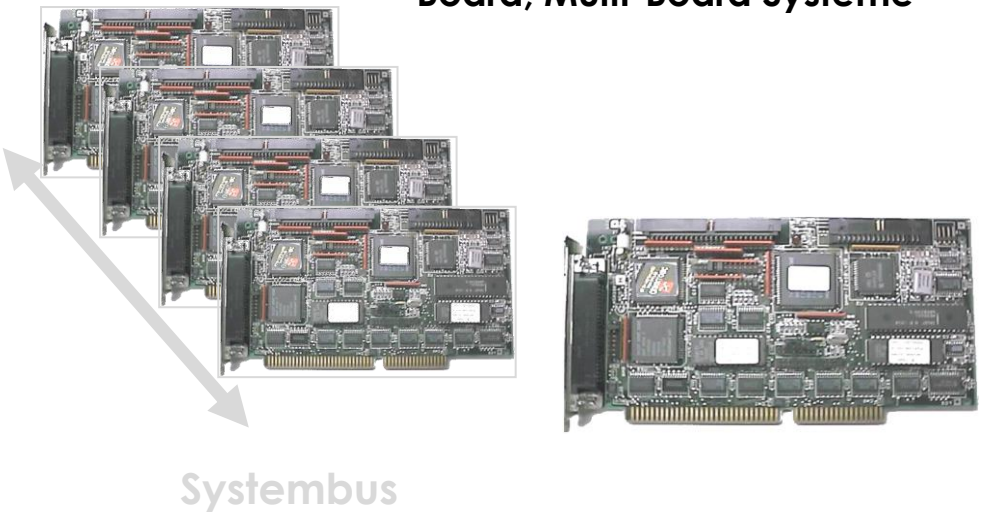
- Kriterien:
  - Kosten
    - Fläche + Entwicklung
  - Verlustleistung
  - Performanz
  - Flexibilität
    - Risikominimierung Time-to-Market

	SoC	Board	Multi-Board
Gewicht, Größe, Leistungsverbrauch	niedrig	hoch	Sehr hoch
Zuverlässigkeit	Sehr hoch	Niedrig/hoch	niedrig
Kosten bei hoher Stückzahl	niedrig	hoch	hoch

System-on-Chip (SoC)



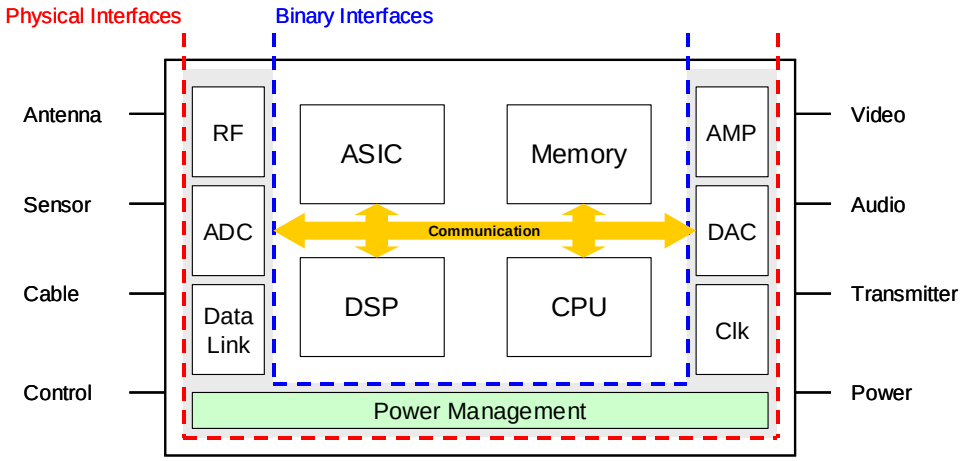
Board, Multi-Board Systeme



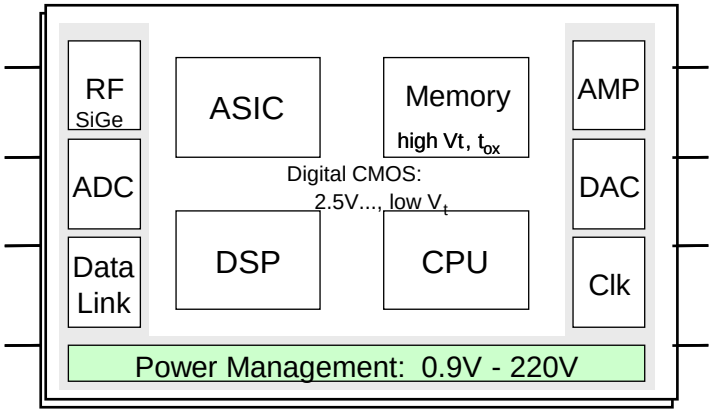
2.8.1 SoC: Funktionale und Technologische Perspektiven

- Aspekte für “Single Die” SoC:
  - unterschiedliche Prozess-Anforderungen
  - Unterschiedliche Spannungen erforderlich
  - Schaltungen: re-designed für Haupt-Technologien

<u>Funktional:</u> Komponenten + Interfaces



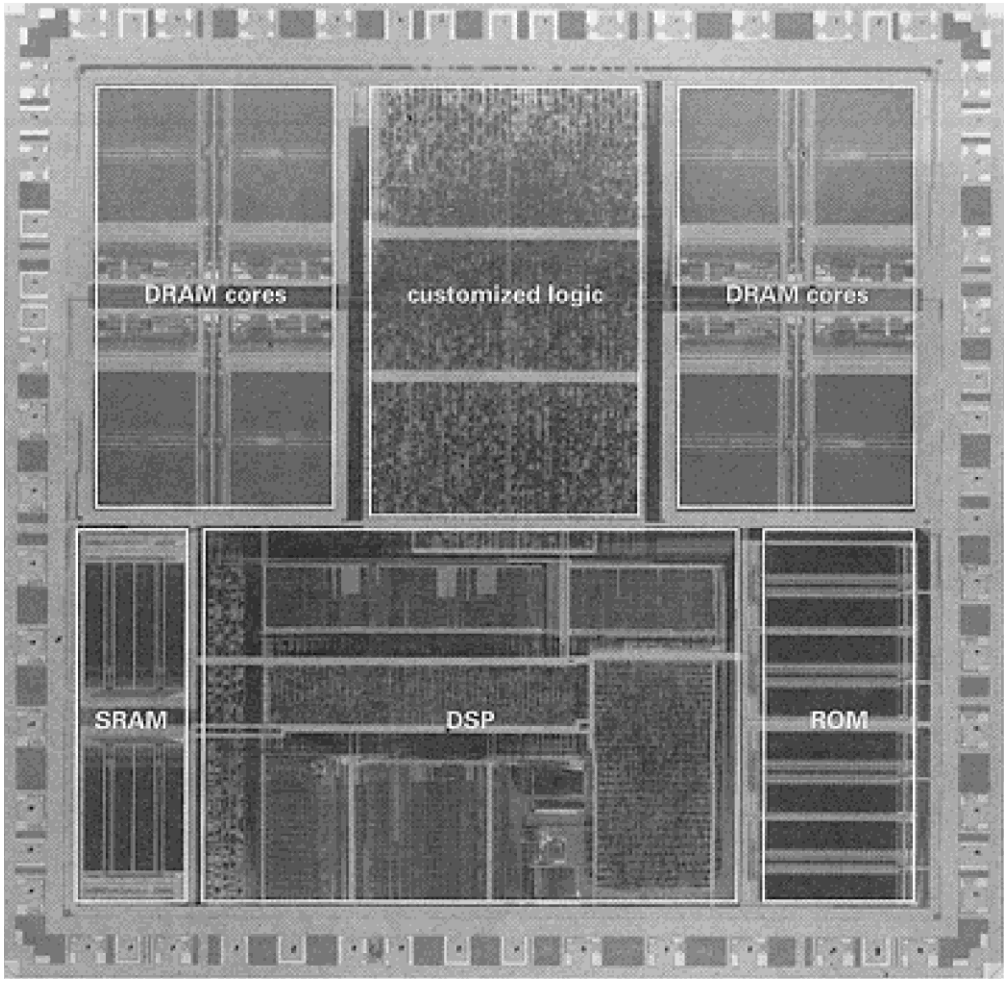
<u>Technologisch:</u> Komponenten + Realisierung



2.8.1 SoC Beispiel: Audio Processing SoC von Siemens

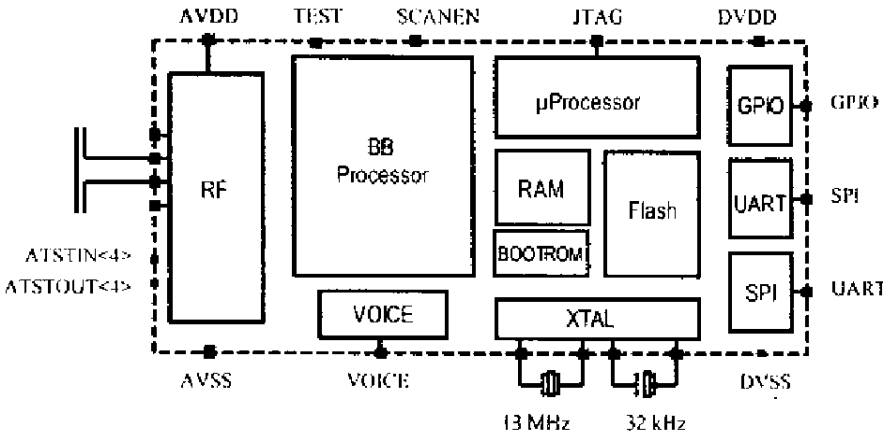
- 16 bit DSP
- 15 k gates customized logic
- SRAM, ROM
- 1 M Bit DRAM

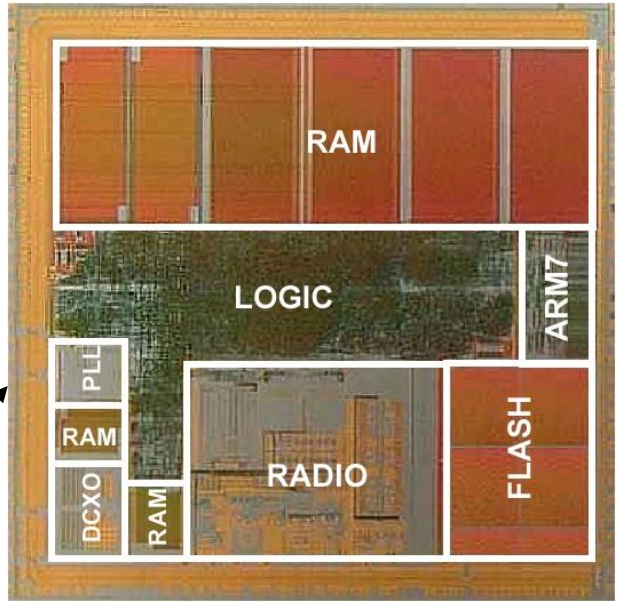
- Anwendungen:
  - Hörgeräte
  - Biometrie
  - Spracherkennung



2.8.1 SoC Beispiel: Bluetooth SoC von Alcatel

- Entwurf wegen Inkompatibilitäten zur Bluetooth Spezifikation eingemottet.
  - CMOS 0,25 □m, 2,5 V Core
  - 40,1 mm2 „die“, wovon 50 % vom RAM belegt werden
  - Empfindlichkeit –80dBm bei 0,1% BER
  - Erfüllt nur BT Spezifikation 1.0.b
  - 5 GHz VCO on Chip, mit automatischer Rekalibrierung
  - 48 KB RAM und 256 KB Flash ROM für Applikationssoftware
  - Entwicklungszeit :18 Monate
  - Rekonfigurierbare UART/Voice Interfaces
  - Frei programmierbares Interface (GPIO), erlaubt Applikationen für eingebettete Systeme, z.B. Bluetooth Headset
  - Demodulation auf „low IF“ mit 1MHz IF
  - Leistungsklasse 2 und 3, (125 mW bei Empfang
  - Antenne sowie andere externe Komponenten auch „on Chip“, d.h. im Chip- Gehäuse (in Ball Grid Array Package)
  - 0,18 □m Prozeß war geplant.





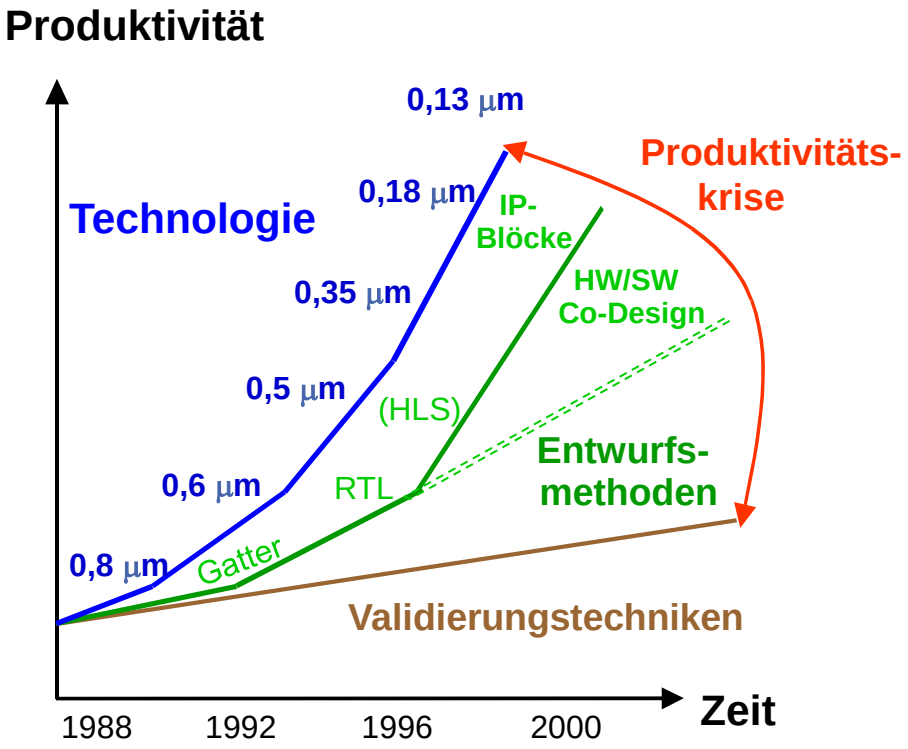
Single Chip Design

2.8.2 SoC - Produktivitätsgrenze

- Komplexe eingebettete Systeme benötigen neue Entwurfsmethoden
  - > 50% Entwicklungszeit für Simulation
  - > 25% Validierung/Verifikation

- Komplexität größer als Produktivitätssteigerung
  - Rapid Prototyping
  - HW/SW Co-Simulation
  - HW/SW Co-Verifikation
  - IP-basierter Entwurf

- Neue effiziente CAD-Methoden
  - Entwurfsraums Exploration



2.8.2 IP-basierte Entwurfsmethodik: IP-Blocktypen

- Soft blocks / IP
  - in einer HDL beschrieben
  - Synthetisierbar, ggf. technologieabhängig
- Firm blocks / IP
  - HDL und Netzlisten, evtl. auch Floorplanning
  - schneller synthetisierbar (technologie-spezifische Syntheseinformationen)
- Hard blocks / IP
  - komplettes Layout + Maskendaten für bestimmte Zieltechnologie
  - nicht mehr synthetisierbar (keine HDL Beschreibung)

Blocktyp	Flexibilität	Vorhersagbarkeit	Portabilität	IP-Schutz
Soft	Sehr flexibel	Unvorhersehbar	Unbegrenzt	Keiner
Firm	Flexibel	Unvorhersehbar	Abbildung auf Bibliothek	Keiner
Hard	Inflexibel	Sehr vorhersehbar	Prozess-Abbildung	gut

2.8.2 SoC IP Bibliothek
Beispiel: Aeroflex Gaisler

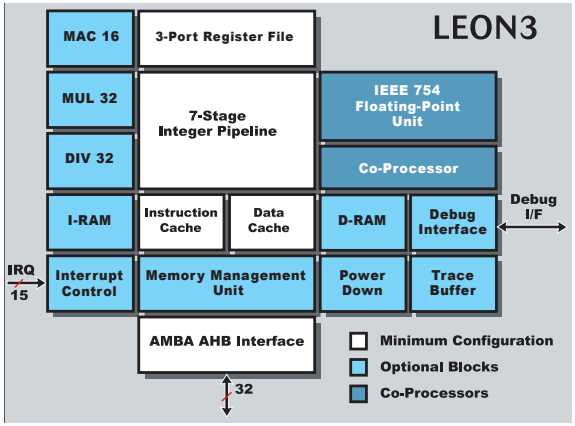
- Komplettes Framework für die Entwicklung von Prozessor-basierten SOC Designs:
  - LEON Prozessor Kern
  - Große IP Sammlung
    - Verhaltens-Simulatoren
    - Verwandte Software Entwicklungswerkzeuge
- Framework wurde für duale Nutzung entwickelt:
  - Entwicklung von kritischen Systemen z.B. Luft- und Raumfahrtsystems
  - Kosten-effizienten Verbraucher Produkte
- Weit verbreitet (insbesondere auch im akademischen Umfeld)

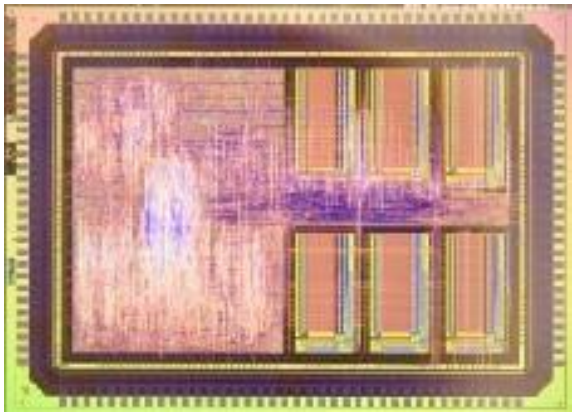
2.8.2 SoC IP Beispiel: Aeroflex Gaisler GrLib

- Integrierte SoC Entwicklungsplattform
- Basiert auf dem AMBA Bus Standard
  - LEON3 SPARC Prozessor
  - LEON4 SPARC Prozessor
  - Voll gepipelinete double-precision IEEE-754 floating point Einheit
  - 32-bit master/target PCI core mit DMA und FIFOs
  - Speicher Controller
  - SpaceWire codec mit RMAP support
  - 10/100/1000 Mbit Ethernet MAC
  - USB Host und Geräte Controller
  - CAN Controller
  - Timer, Interrupt Controller, UART, VGA Controller, PS/2 Interface, GPIO
  - AES Kryptographie

2.8.2 SoC IP Beispiel: Aeroflex Gaisler – Leon3

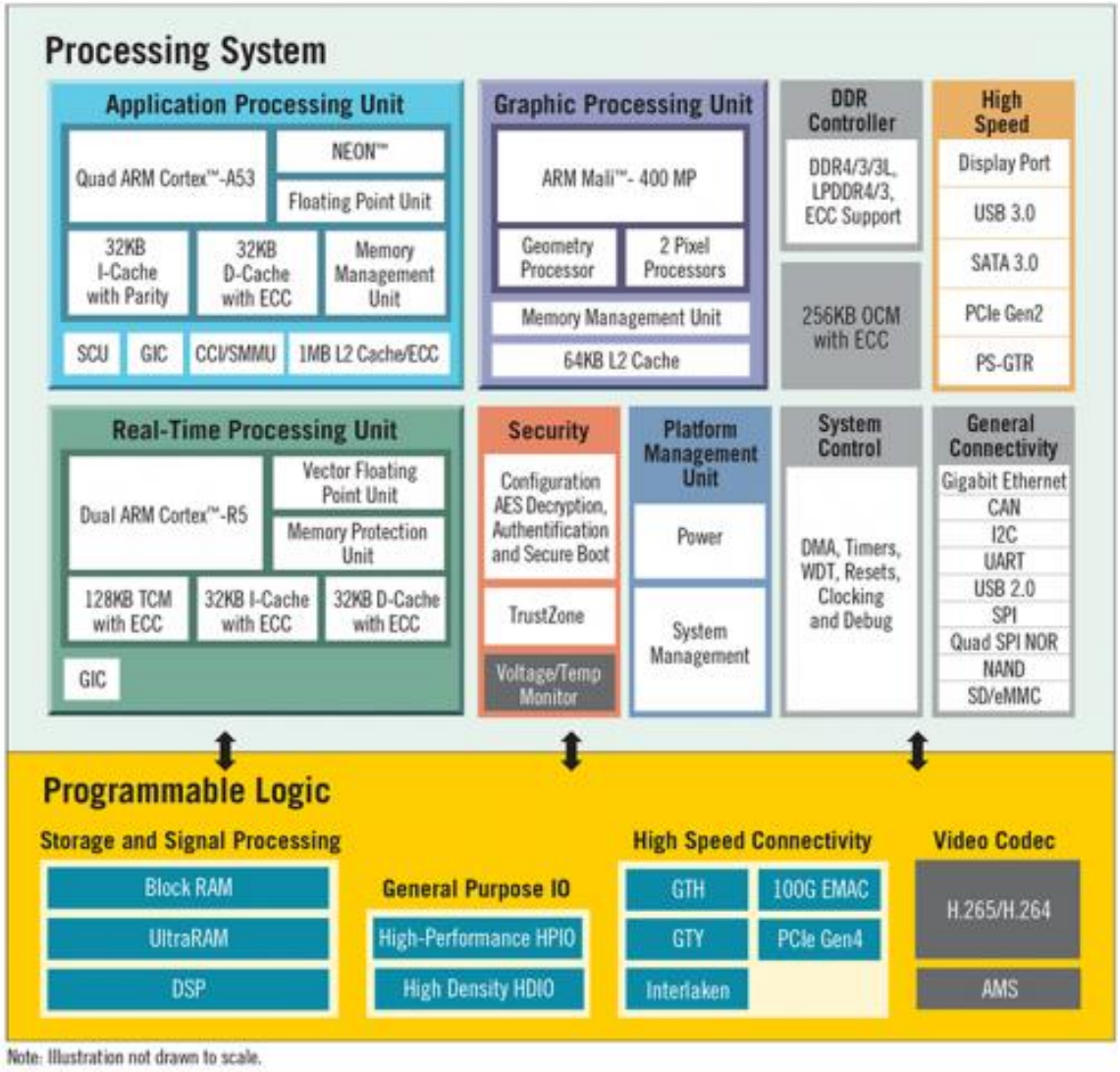
- Synthetisierbares VHDL Modell eines 32-bit Prozessors
- Hoch konfigurierbar, besonders passend für system-on-a-chip (SoC) Designs
  - SPARC V8 Instruktionssatz mit V8e Erweiterungen
  - 7-stufige Pipeline
  - Hardware Multiplizierer, Dividierer und MAC Einheit
  - High-performance, fully pipelined IEEE-754 FPU
  - Separater Instruktions- und Daten Cache (Harvard architecture)
  - AMBA-2.0 AHB bus interface
  - Symmetric Multi-processor support (SMP)
  - Bis zu 125 MHz in FPGA und 400 MHz auf 0.13 µm ASIC Technologien
  - High Performance: 1.4 DMIPS/MHz, 1.8 CoreMark/MHz (gcc -4.1.2)
- GNU GPL Lizenz: freie und uneingeschränkte Nutzung für Forschung and Lehre





http://www.gaisler.com/index.php/products/processors

2.8.2 SoC Beispiel: Zynq Ultrascale+



- Was ist ein System-on-Chip?
- Welche Kriterien zeigen die Relevanz von SoCs?
- Wo liegt das Problem im SoC-Entwurf? Welche Lösungen werden angetrebt?
- Was ist IP-basierter Entwurf? Wie funktioniert dieser?

Kapitel 2 Inhalt (I)

- 2.1 Allgemeiner Aufbau
- 2.2 Klassifikation
- 2.3 General-Purpose Prozessoren (GPP)
  - 2.3.1 Architekturen
    - 2.3.1.1 Akkumulatormaschine
    - 2.3.1.2 Stackmaschine
    - 2.3.1.3 Registersatzmaschine
  - 2.3.2 Performanzsteigerung
    - 2.3.2.1 Pipelining
    - 2.3.2.2 Superskalarität / Out-of-Order Execution
    - 2.3.2.3 Very Long Instruction Word (VLIW)
    - 2.3.2.4 Single Instruction Multiple Data (SIMD)
    - 2.3.2.5 Caches
    - 2.3.2.6 Multiple Instruction Multiple Data (MIMD)

Kapitel 2 Inhalt (II)

- 2.4 Spezialprozessoren
  - 2.4.1 Mikrocontroller (µC)
  - 2.4.2 Digitale Signalprozessoren (DSPs)
  - 2.4.3 Grafik Prozessoren (GPU)
- 2.5 Bus, Network-on-Chip (NoC) & Multicore
- 2.6 Anwendungs-spezifische Instruktionssatz Prozessoren (ASIP)
- 2.7 Field Programmable Gate Arrays (FPGA)
- 2.8 System-on-Chip (SoC)