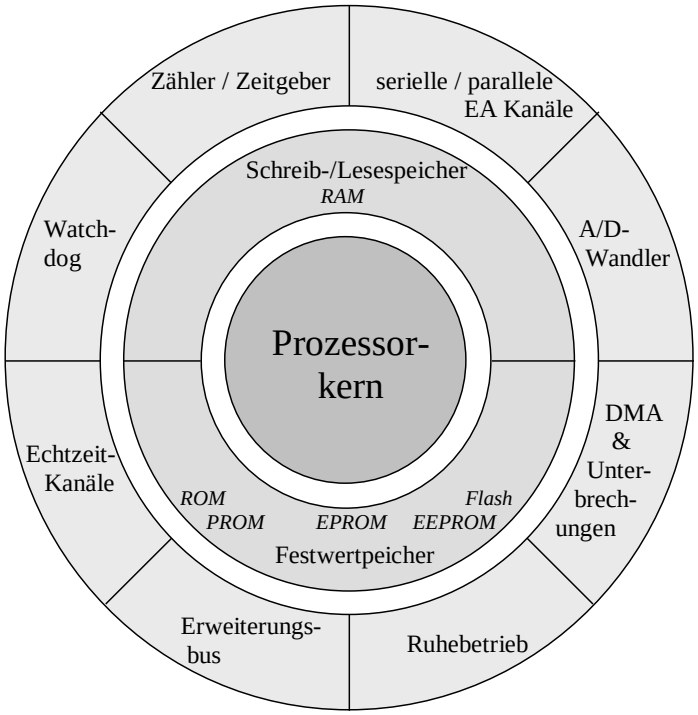
Inhalt

- 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

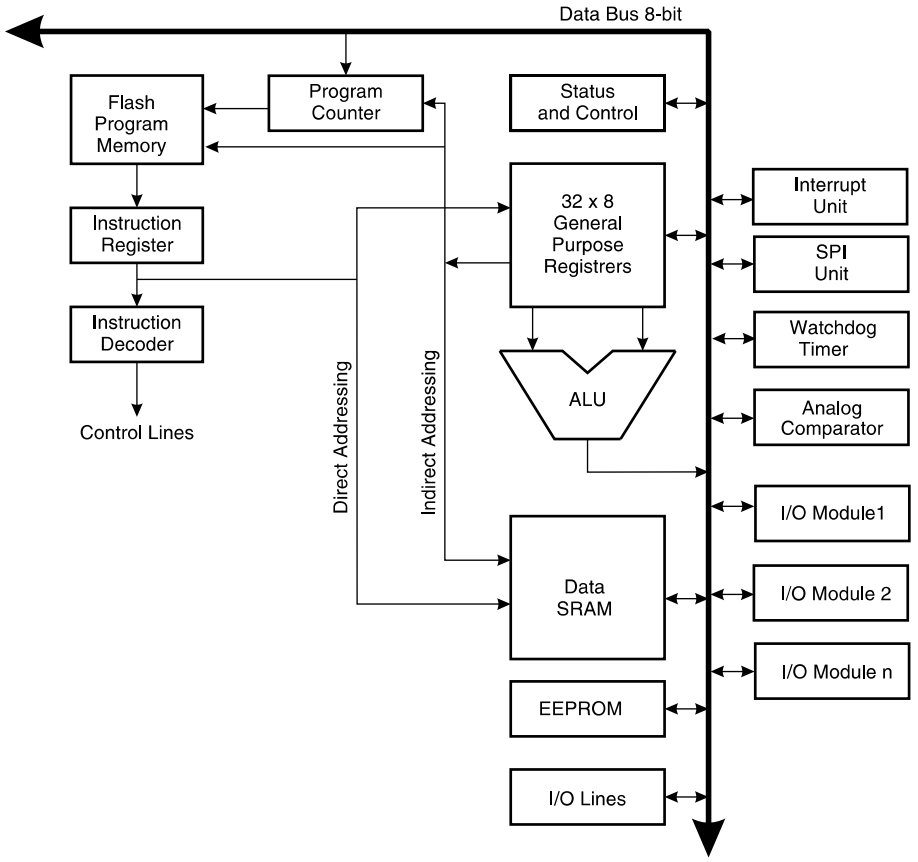
- 1. Eigens für den Mikrocontroller entwickelter einfacher Kern

- 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

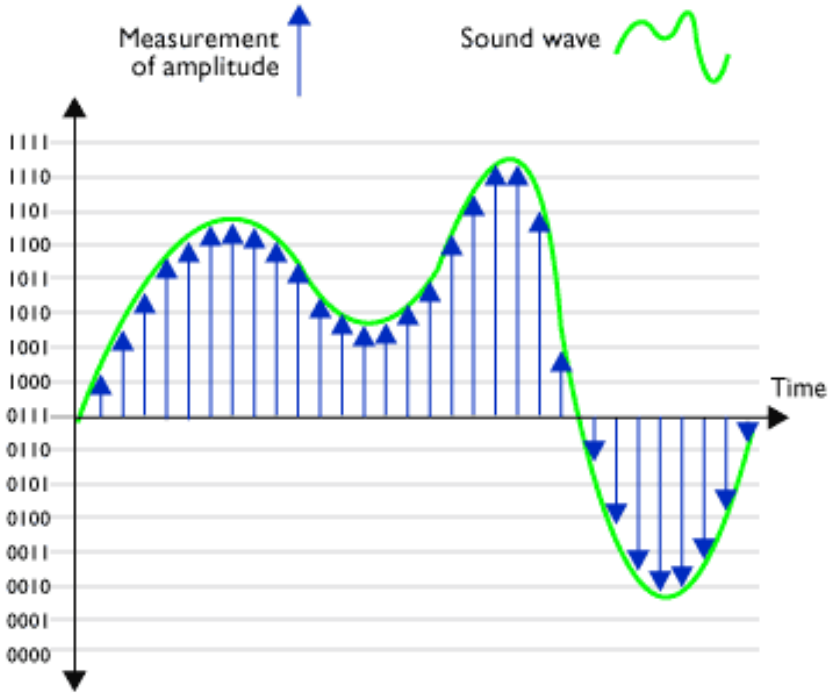
      -> Hat heute praktisch alle anderen ROM Technologien verdrängt

2.4.1 Microcontroller: I/O Module

- General Purpose Input/Output (GPIO)
  - Softwaregesteuertes Schalten von digitalem Input und Output
  - Typischerweise in Gruppen von 8 (1 Byte) oder größer
  - Beispiele: Steuerung von LEDs, Eingangstaster oder –schalter
- Analog/Digital Converter (ADC)
  - Umwandlung analoger Eingangsignale in einen digitalen Datenstrom
  - Beispiele: Temperatur, Geschwindigkeit, Beschleunigung, …
- Pulse Width Modulation (PWM)
  - Veränderung des duty cycle für einen gegebenen Zyklus
  - Beispiel: Geschwindigkeitsregelung eines E-Motors

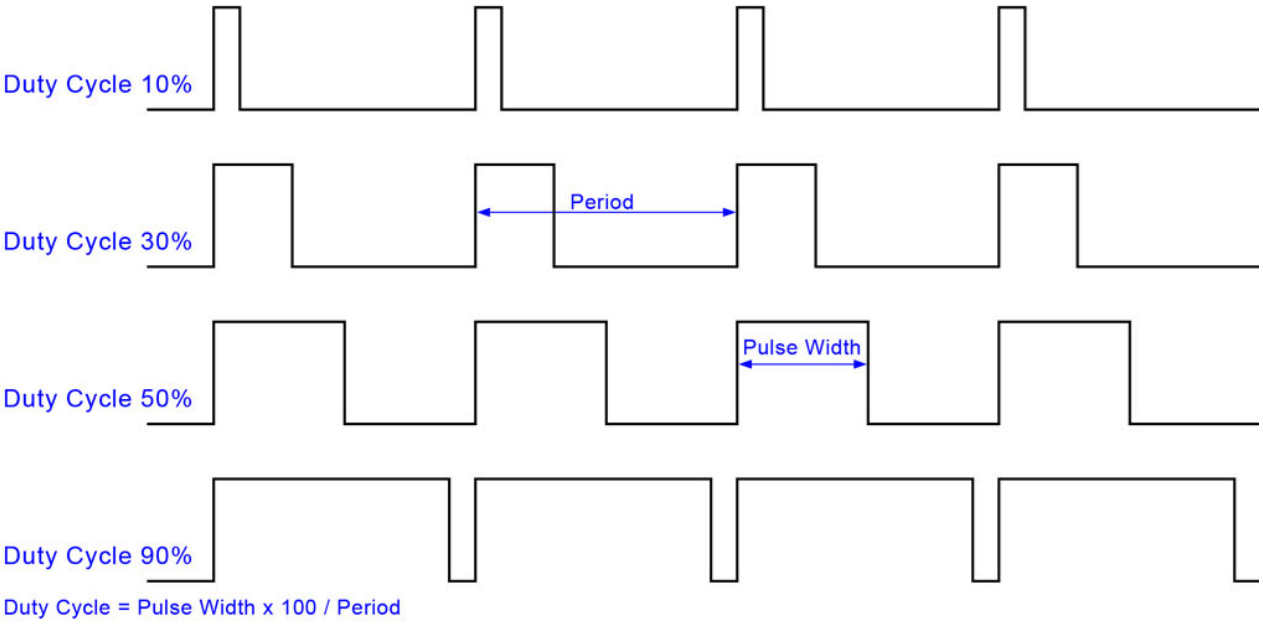
2.4.1 Analog/Digital Converter (ADC)

- Verwandelt analoge Signale in digitale Datenströme
- Amplitude eines kontinuierlichen analogen Signals wird zu bestimmten Zeitpunkten gesampelt (sampling rate) und das Ergebnis in einem Register abgelegt.
- Spannungsbereich und Auflösung hängen vom Controller ab.



2.4.1 Pulsweitenmodulation (PWM)

- Wird zur Intensitätssteuerung von Aktoren verwendet
- Änderung des On-/Off-Verhältnisses innerhalb eines festen Zyklus (duty cycle) steuert Intensität
- Beispiele:
  - Änderung des duty cycle ändert Geschwindigkeit von Motoren
  - Änderung des duty cycle ändert Helligkeit von LEDs



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

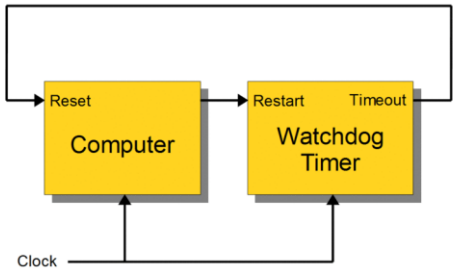
2.4.1 Mikrocontroller: Watchdog

- „Wachhund“ zur Überwachung der Programmaktivitäten eines Mikrocontrollers

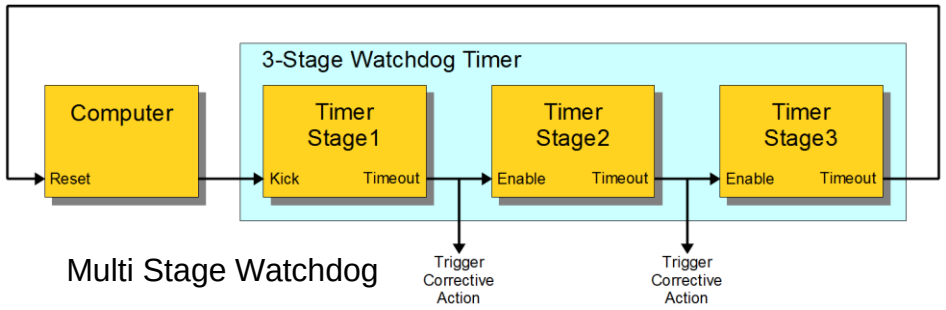
- Programm muss in regelmäßigen Abständen Lebenszeichen liefern
  - Bleiben diese aus, so nimmt der Wachhund einen Fehler im Programmablauf an => Reset

- 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

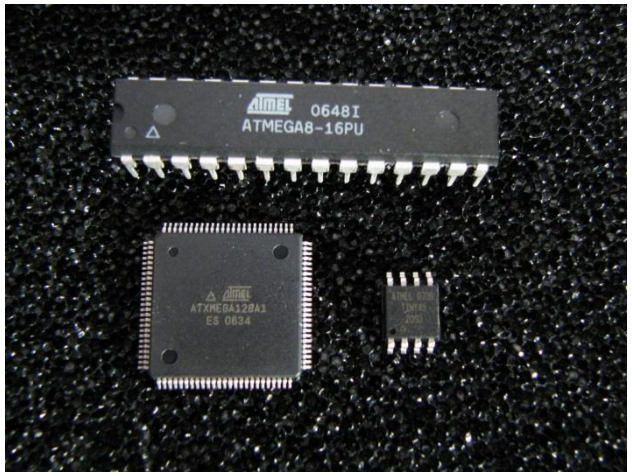
2.4.1 µC Beispiel: Intel MCS-51

- 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

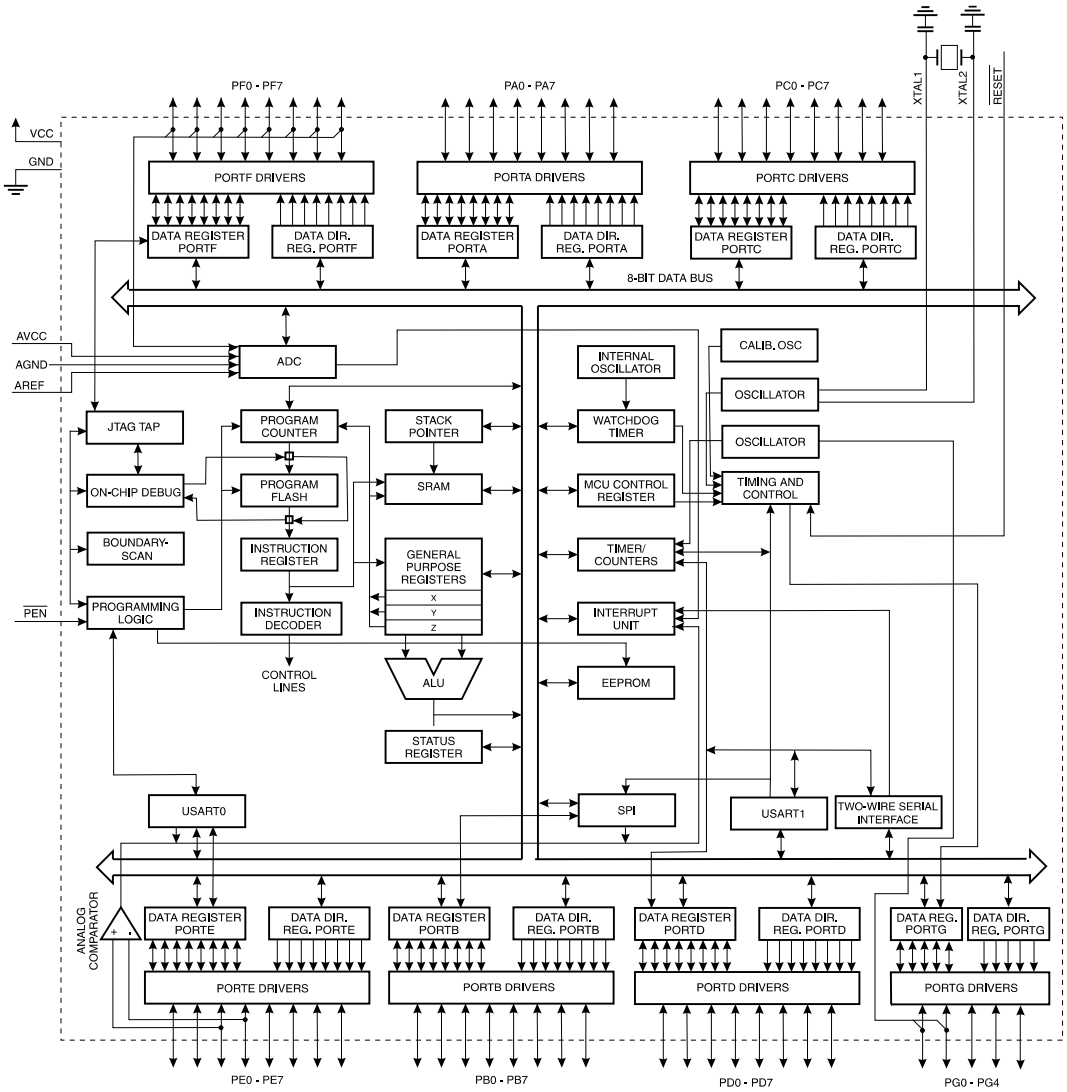
2.4.1 µC Beispiel: Atmel AVR

- 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

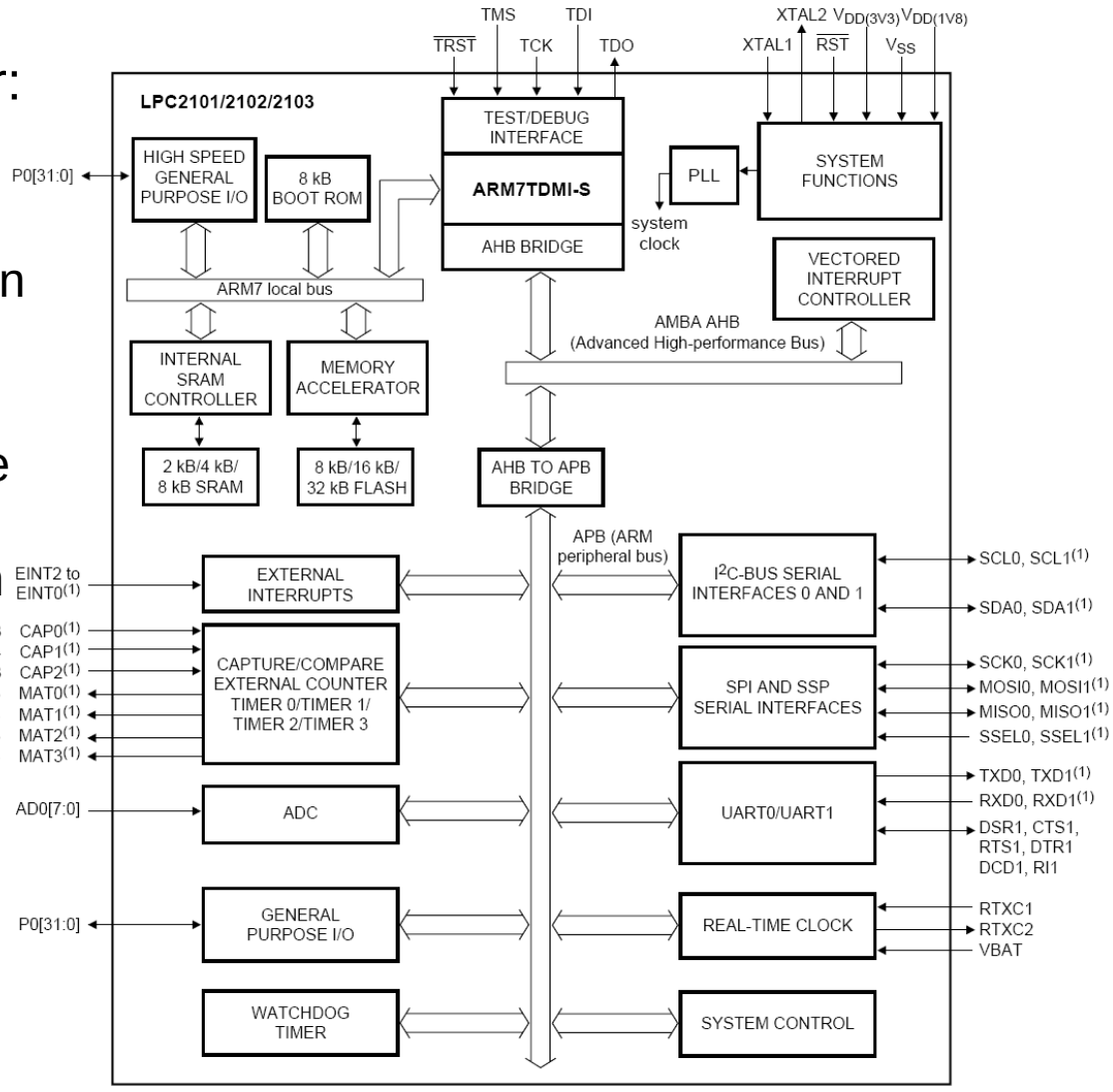
2.4.1 µC Beispiel: ARM7 TDMI

- 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

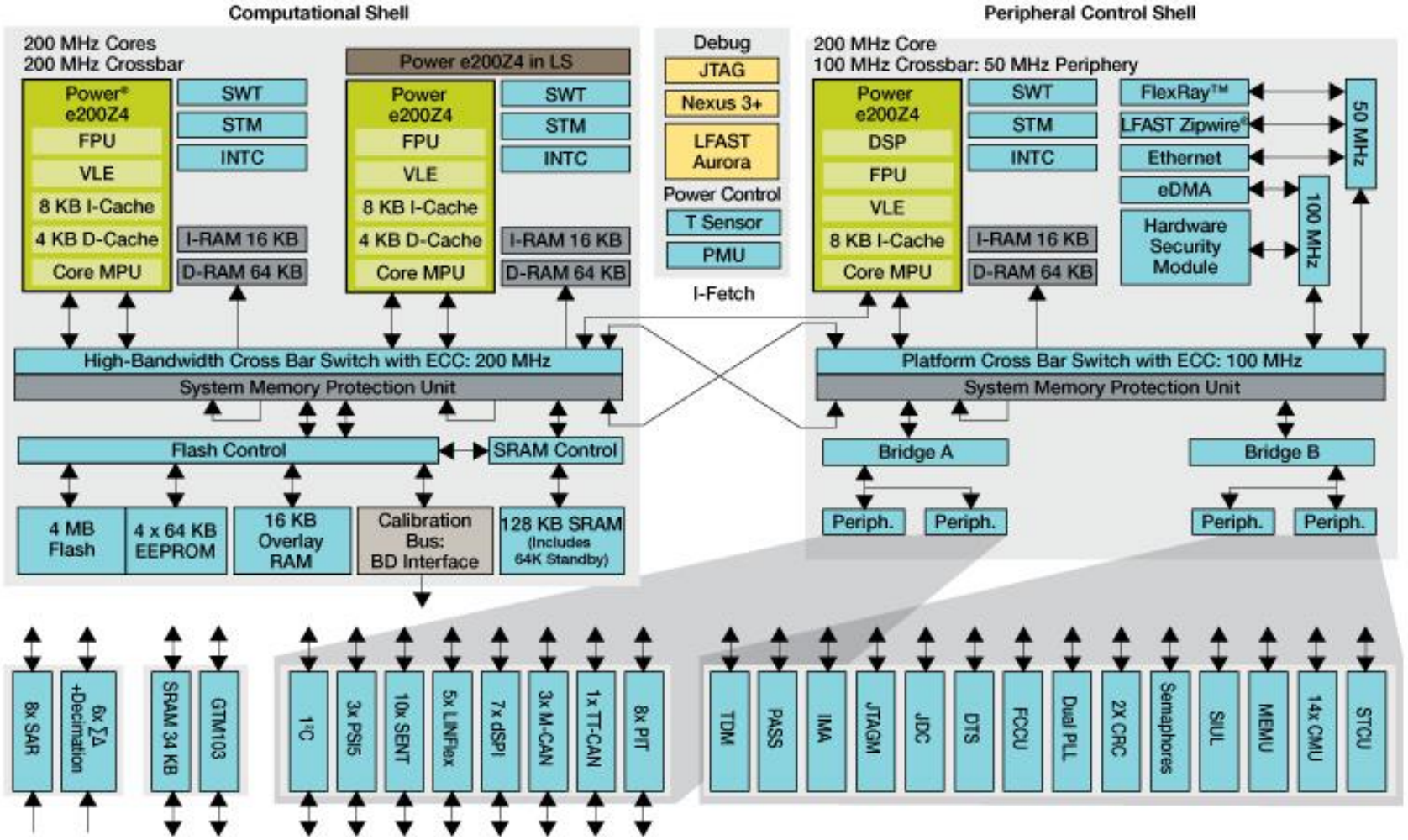
2.4.1 µC Beispiel: ARM7 TDMI

- 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

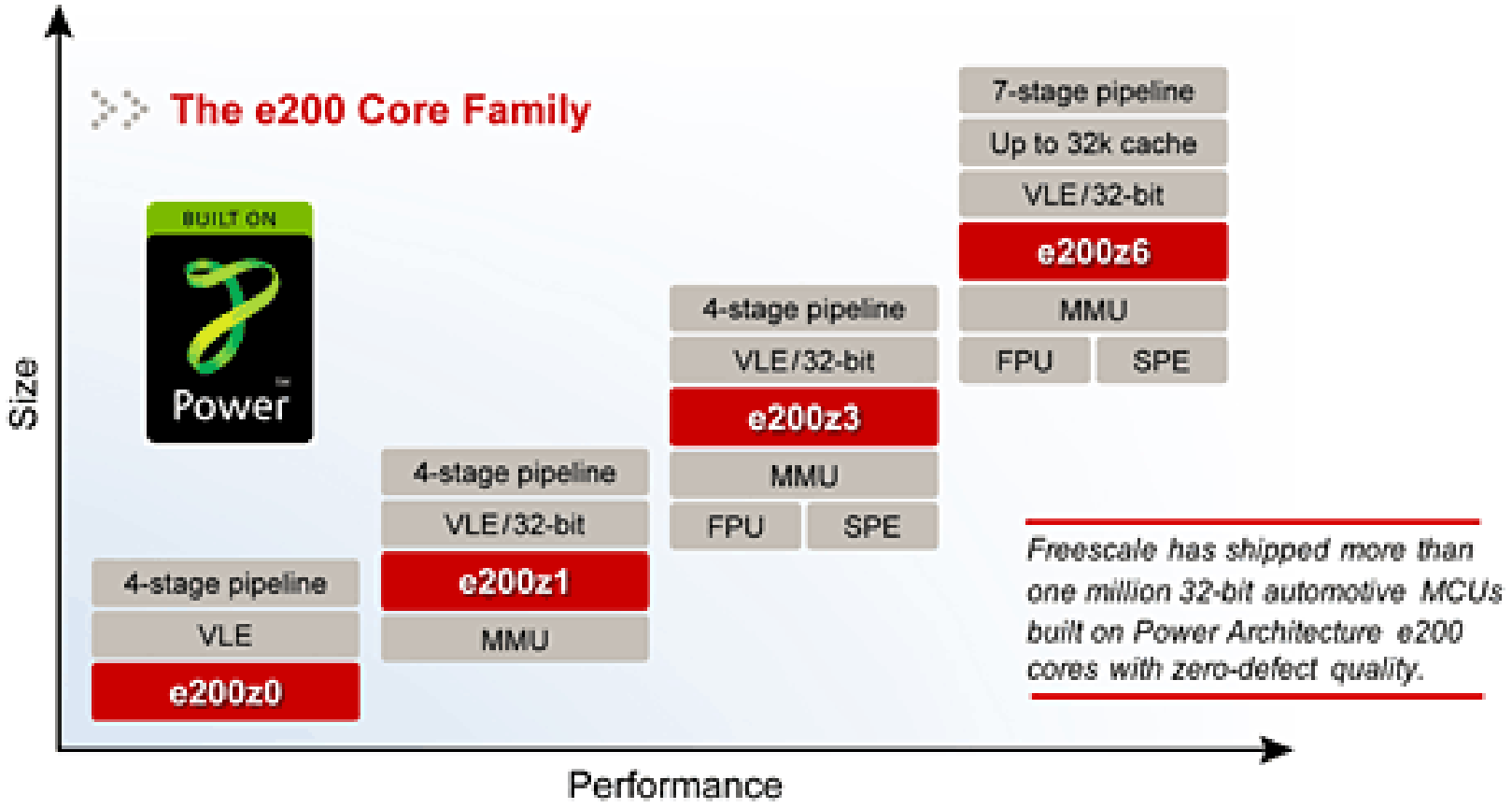


Beispiel: Controller für Motorsteuergeräte

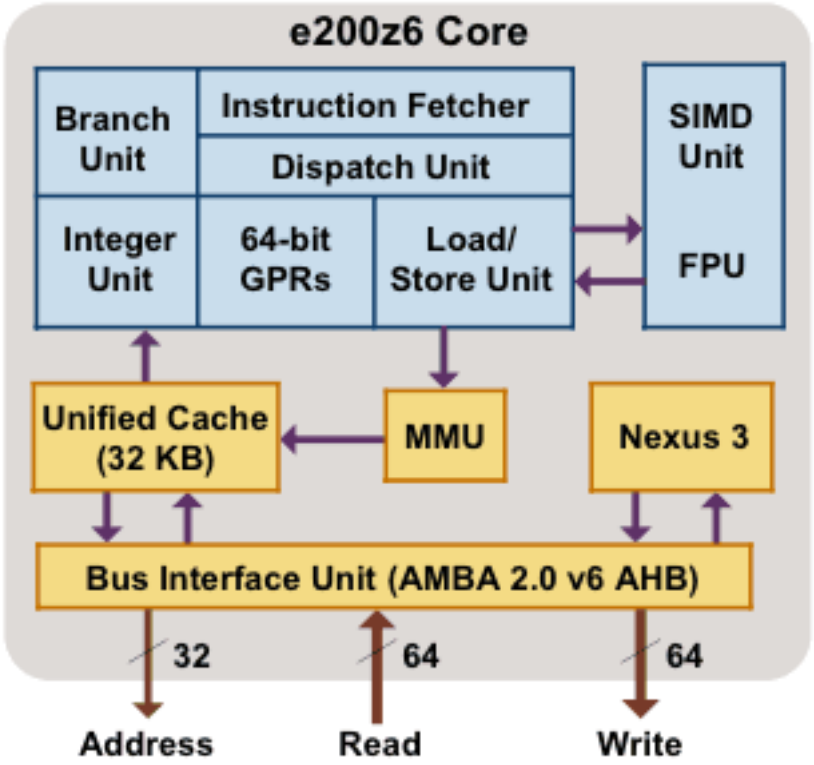
Qorivva MPC5746M Block Diagram



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

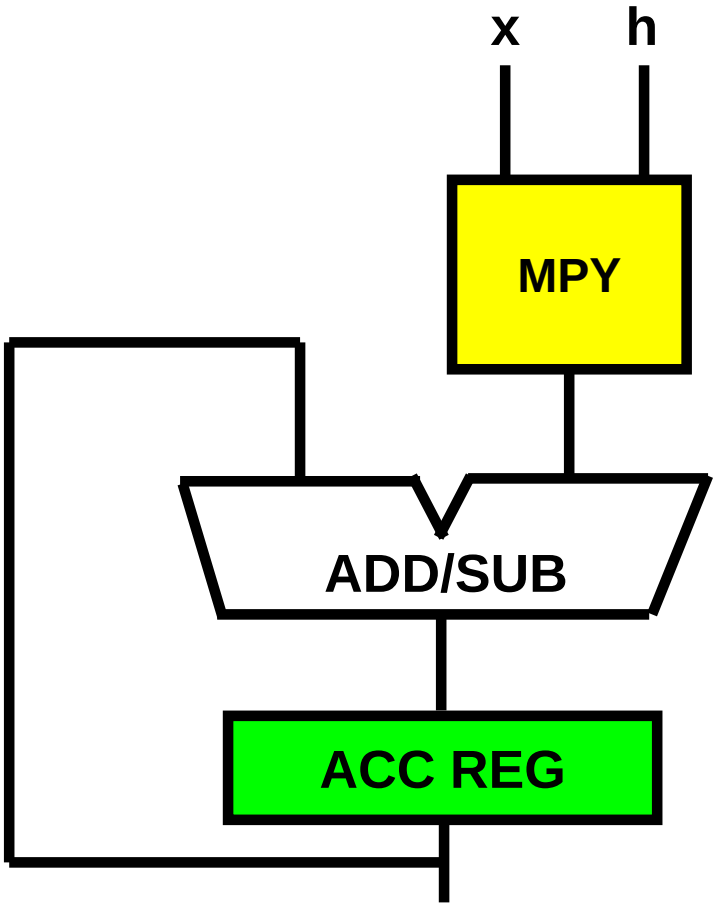
TMS320C3x Assembler

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

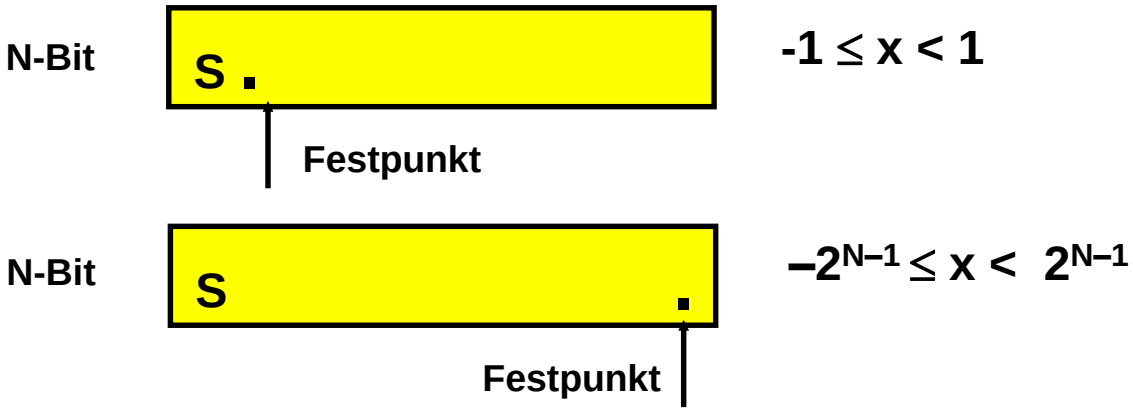
- Fixed Point (FX)
  - gleiche Mantissenbreite: kleiner (billiger) + schneller als Floating-Point
  - Entwurf von manchen Anwendungen durch Rundungs- und Skalierungsprobleme erschwert
  - Für viele DSP-Anwendungen ausreichend

- 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

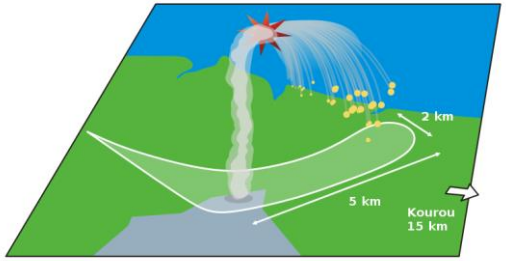


2.4.2 DSP: Arithmetik / Genauigkeit (II)

- 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

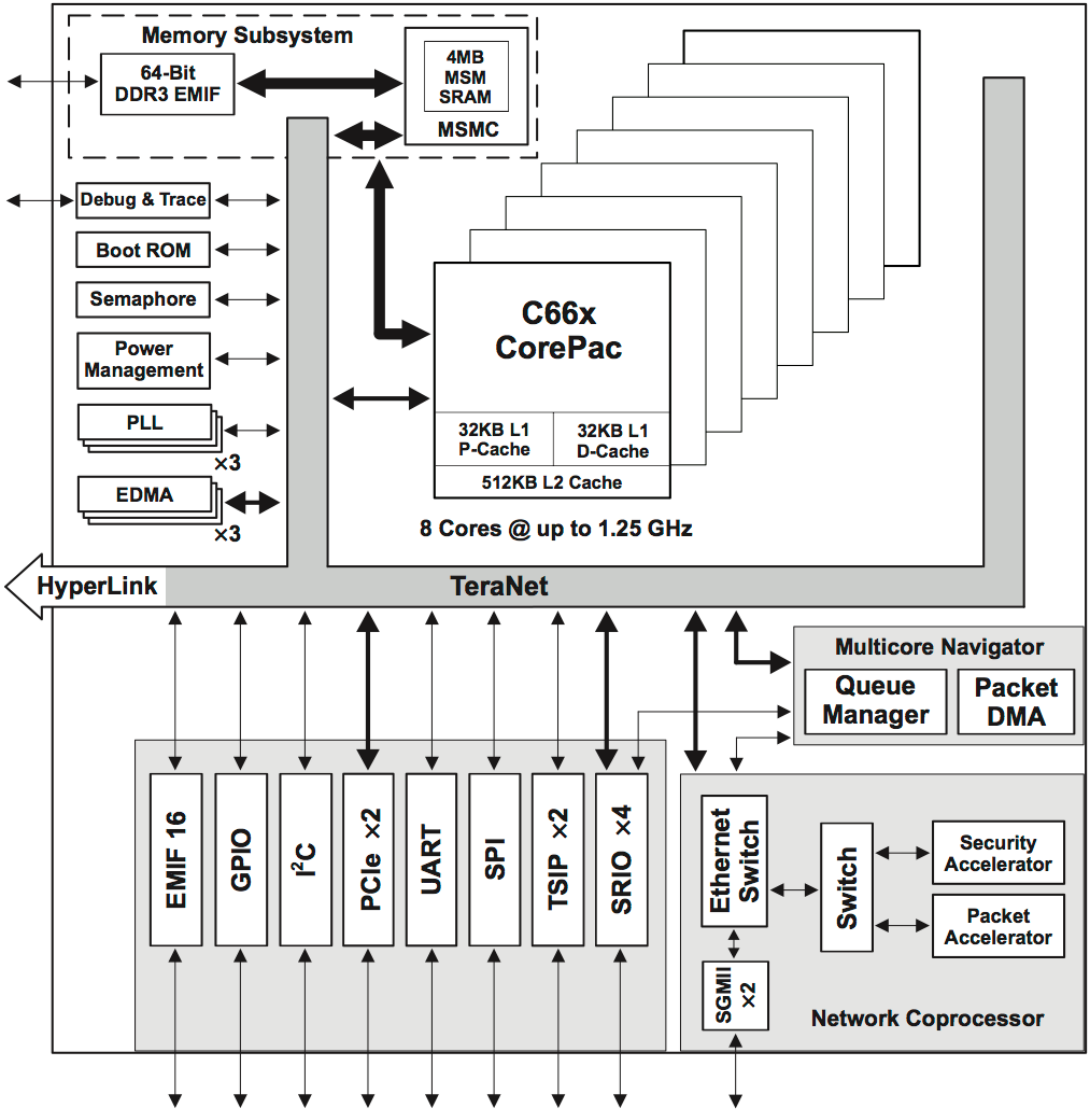
- DSPs stammen eigentlich von analogen Signalprozessoren (ASP)
  - Verwenden aber eine endliche Modulo-Arithmetik

- 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



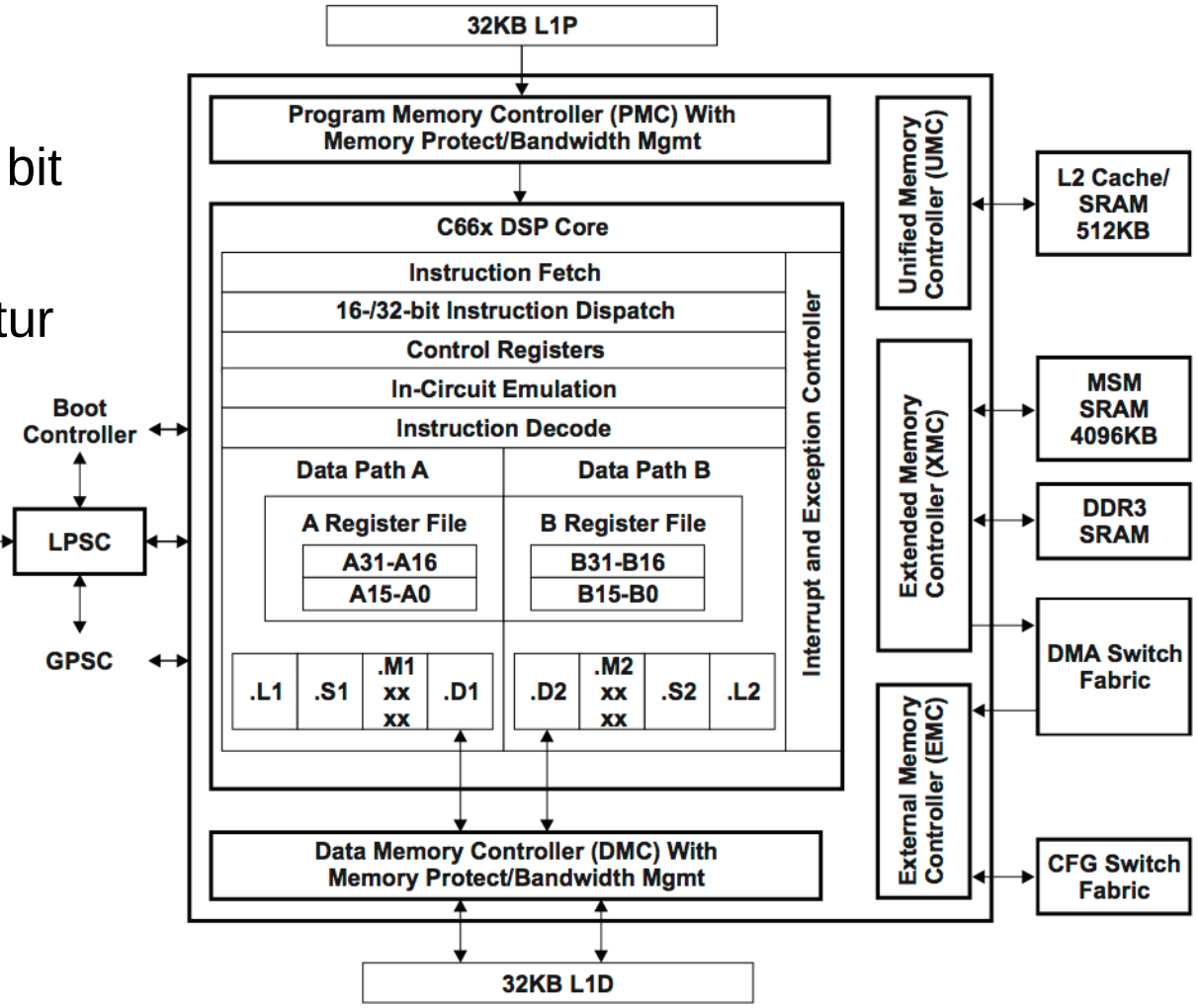
2.4.2 DSP Beispiel: TI TMS320C6678 (I)

- 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



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

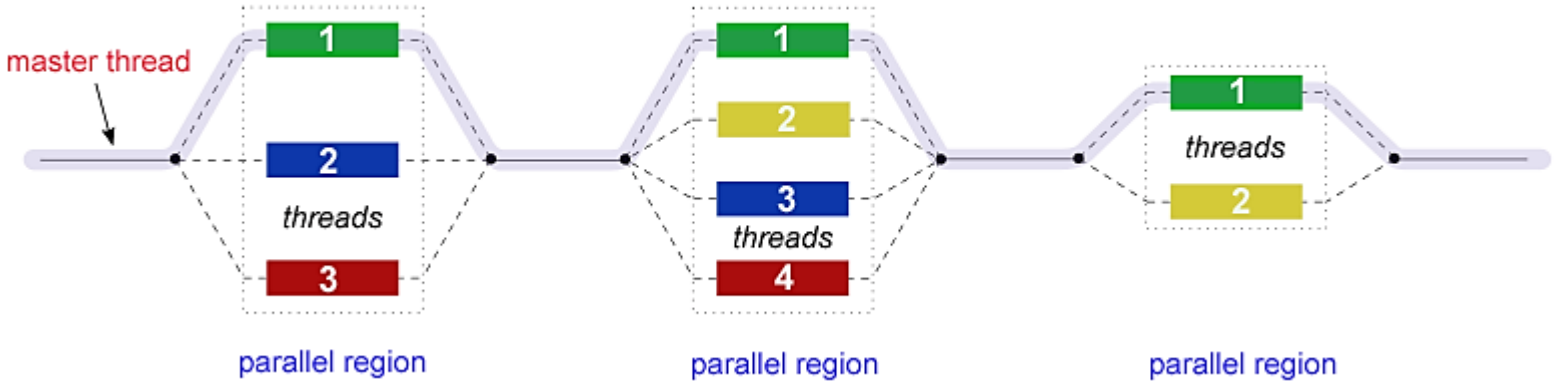
2.4.2 DSP Beispiel: TI Code Composer Studio (CCS)

- 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

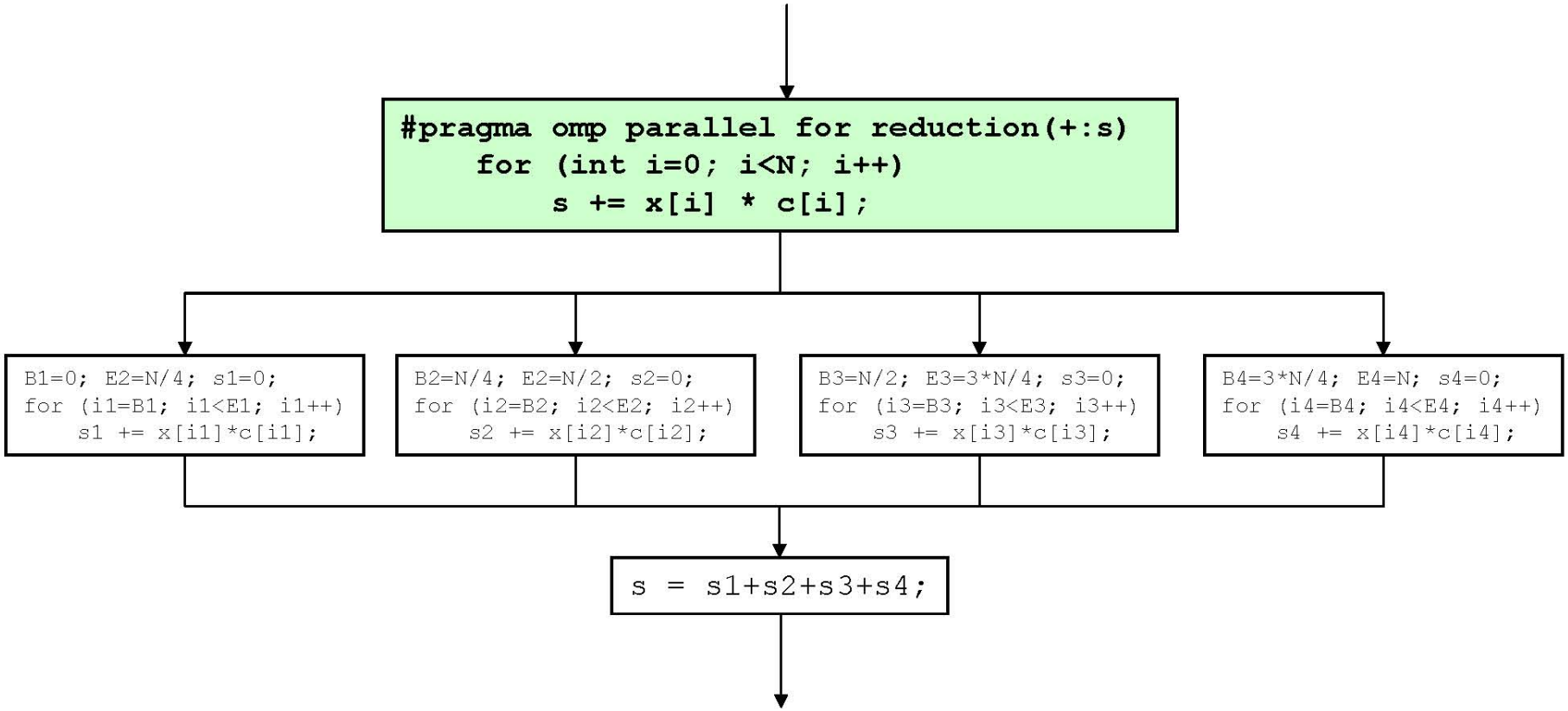
2.4.2.1 OpenMP Einführung

- 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*



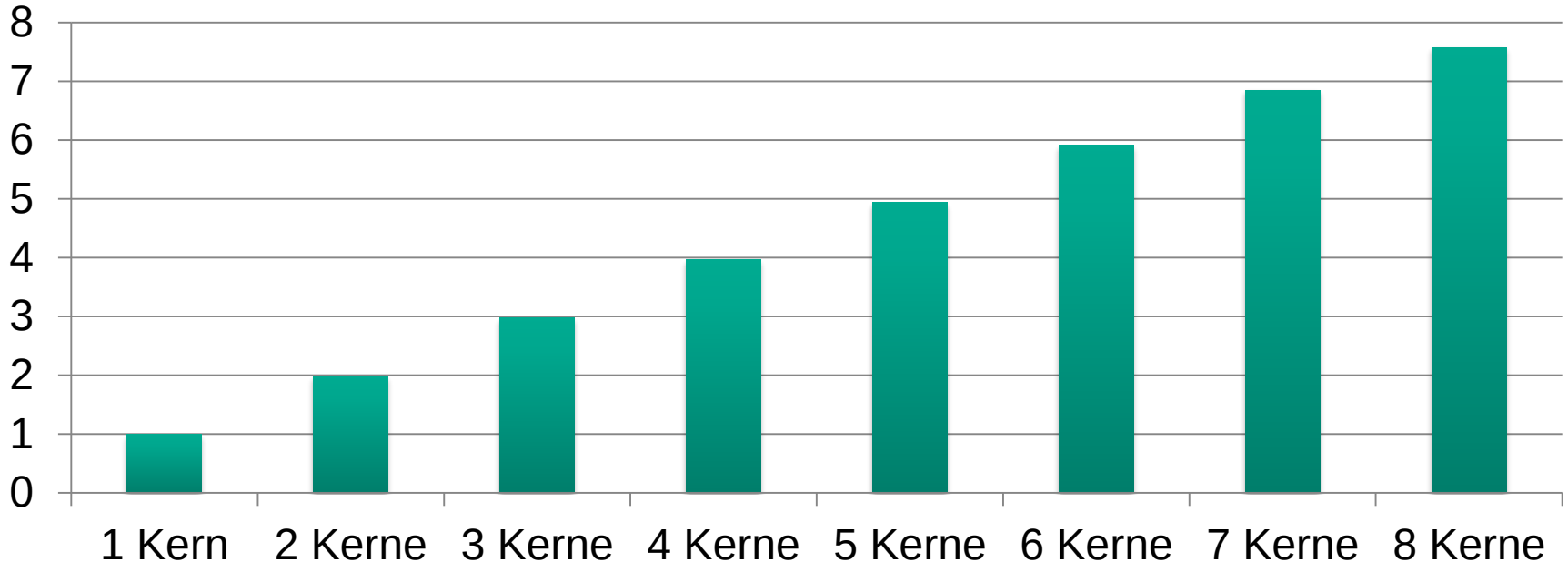
2.4.2.1 OpenMP

- 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.

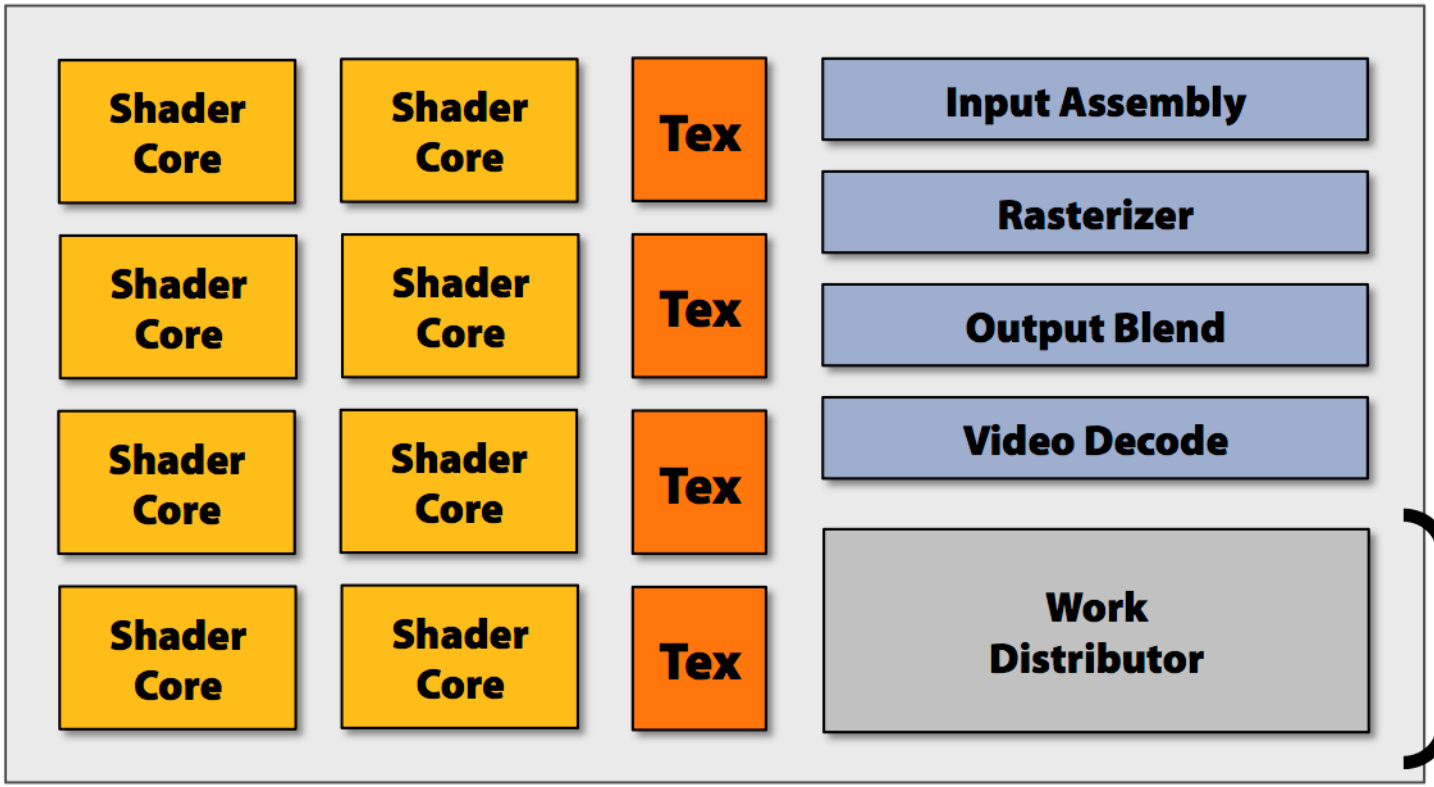
- Was ist ein digitaler Signalprozessor?
- Was ist der Unterschied zwischen einem Mikroprozessor und einem digitalen Signalprozessor?
- Wie werden DSPs heute programmiert?
- Wie kann Parallelität realisiert werden?

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)

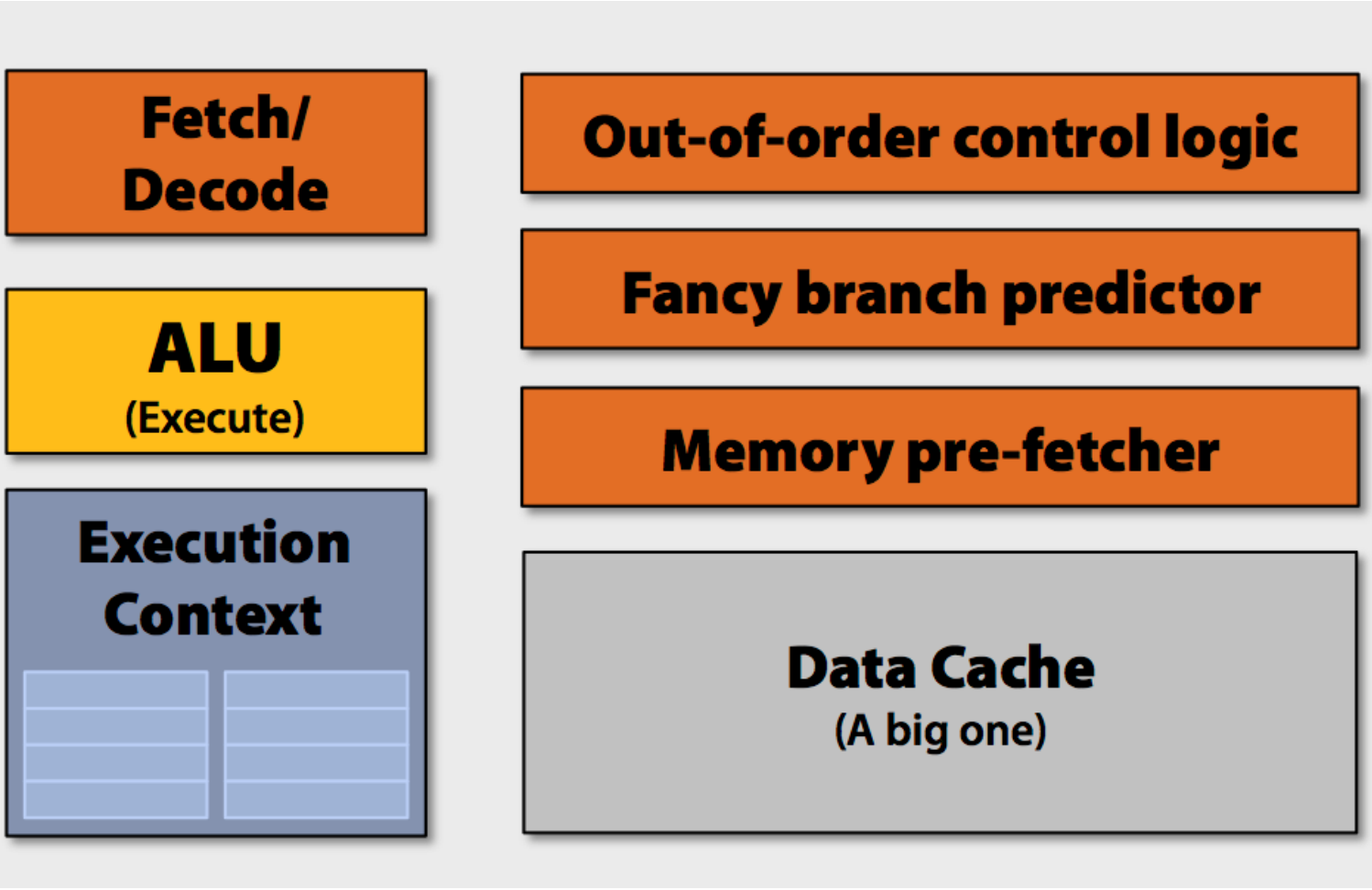
2.4.3 Was ist eine GPU?

- Heterogener Chip Multi-Prozessor
  - Speziell angepasst für Grafikanwendungen



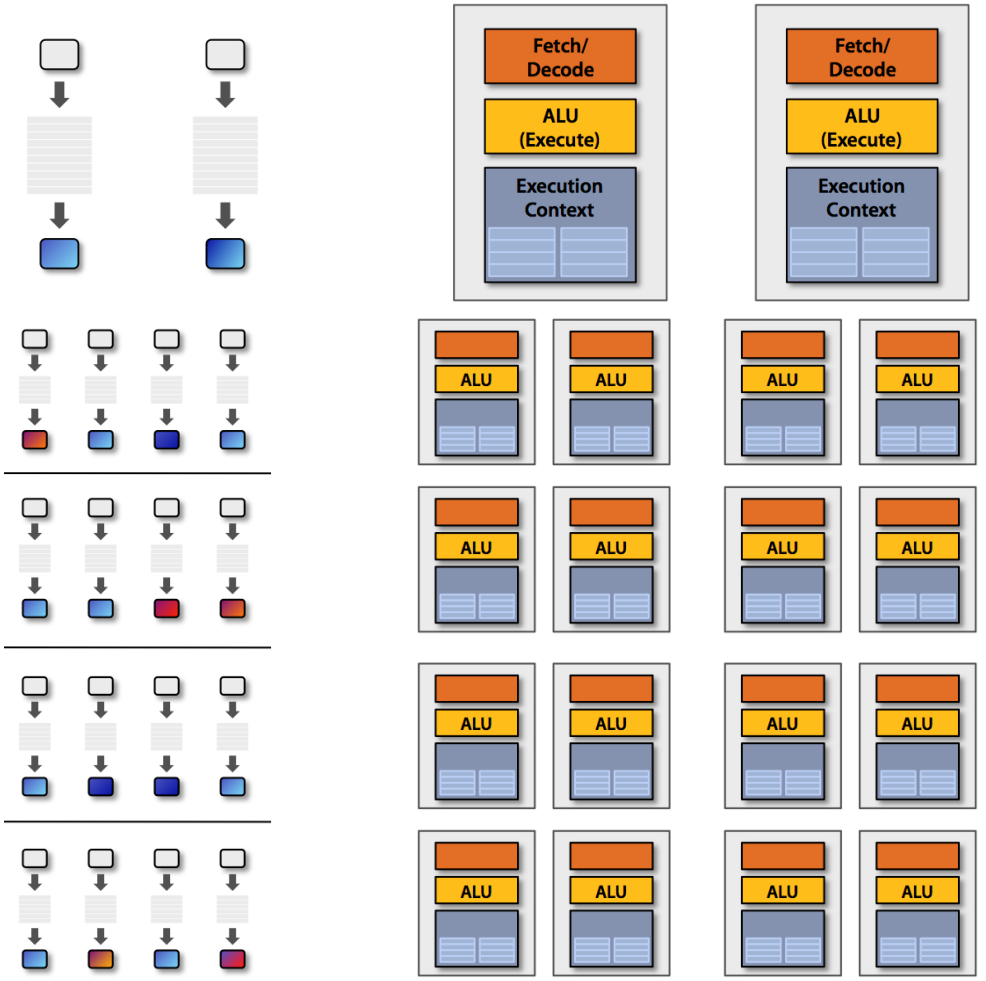
http://s08.idav.ucdavis.edu/fatahalian-gpu-architecture.pdf

2.4.3 CPU-style Kerne (I)

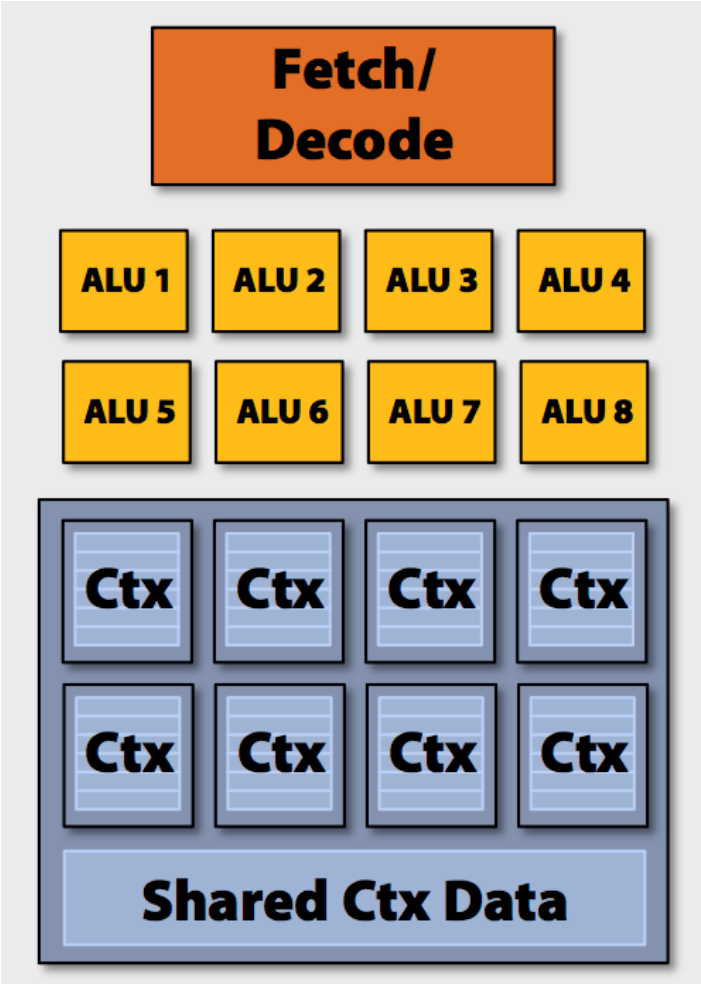


2.4.3 CPU-style Kerne (II)

- 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

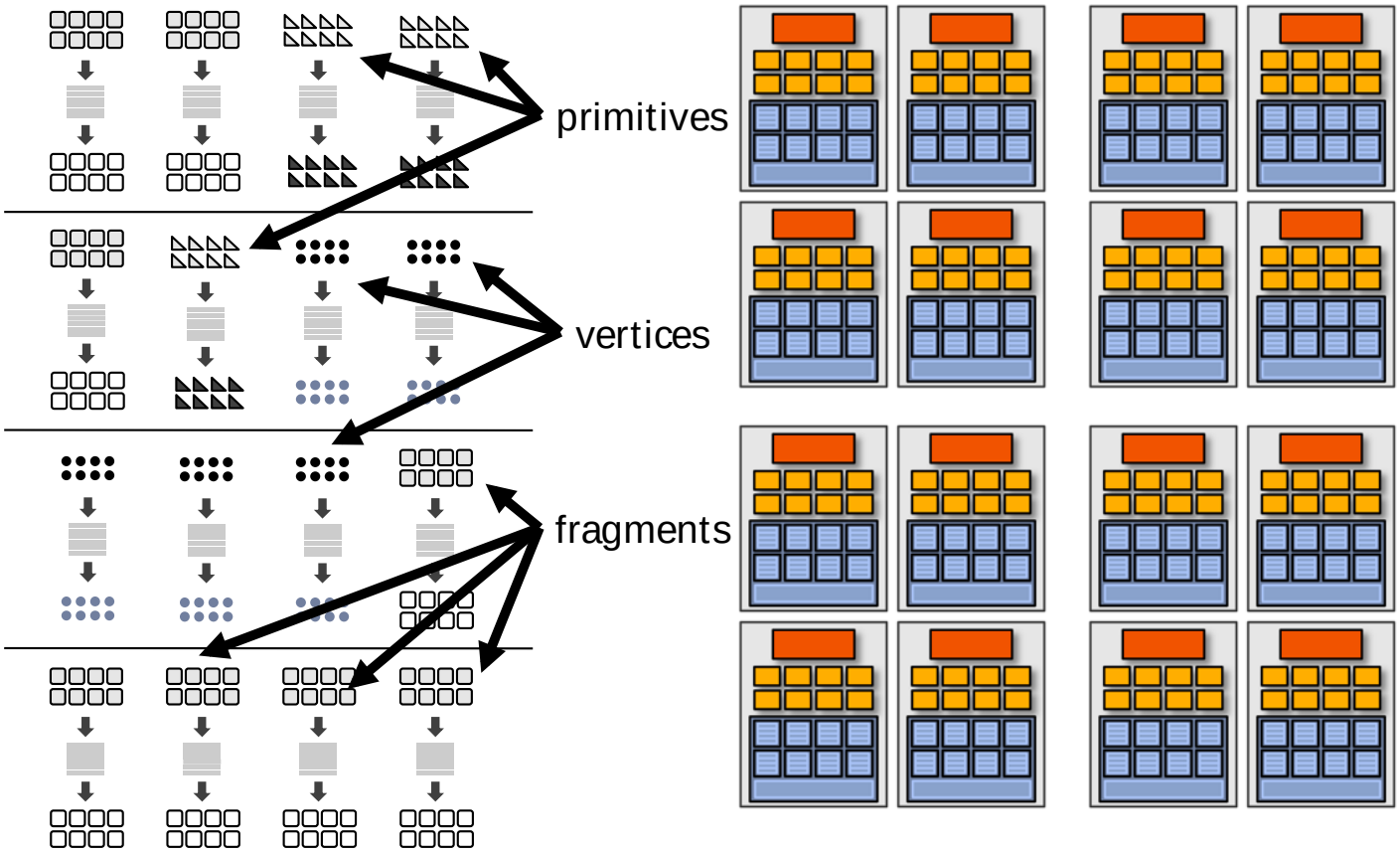


- Idee: Amortisieren von Kosten/Komplexität bei der Verwaltung von Instruktionsströmen zwischen vielen ALUs
- Hinzufügen von ALUs ➔ SIMD
  - Neu kompilierter Shader sollte auf Anzahl der im Shader Core vorhandenen ALUs angepasst sein
    - Hier: 8 parallele Fragmente für 8 ALUs

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

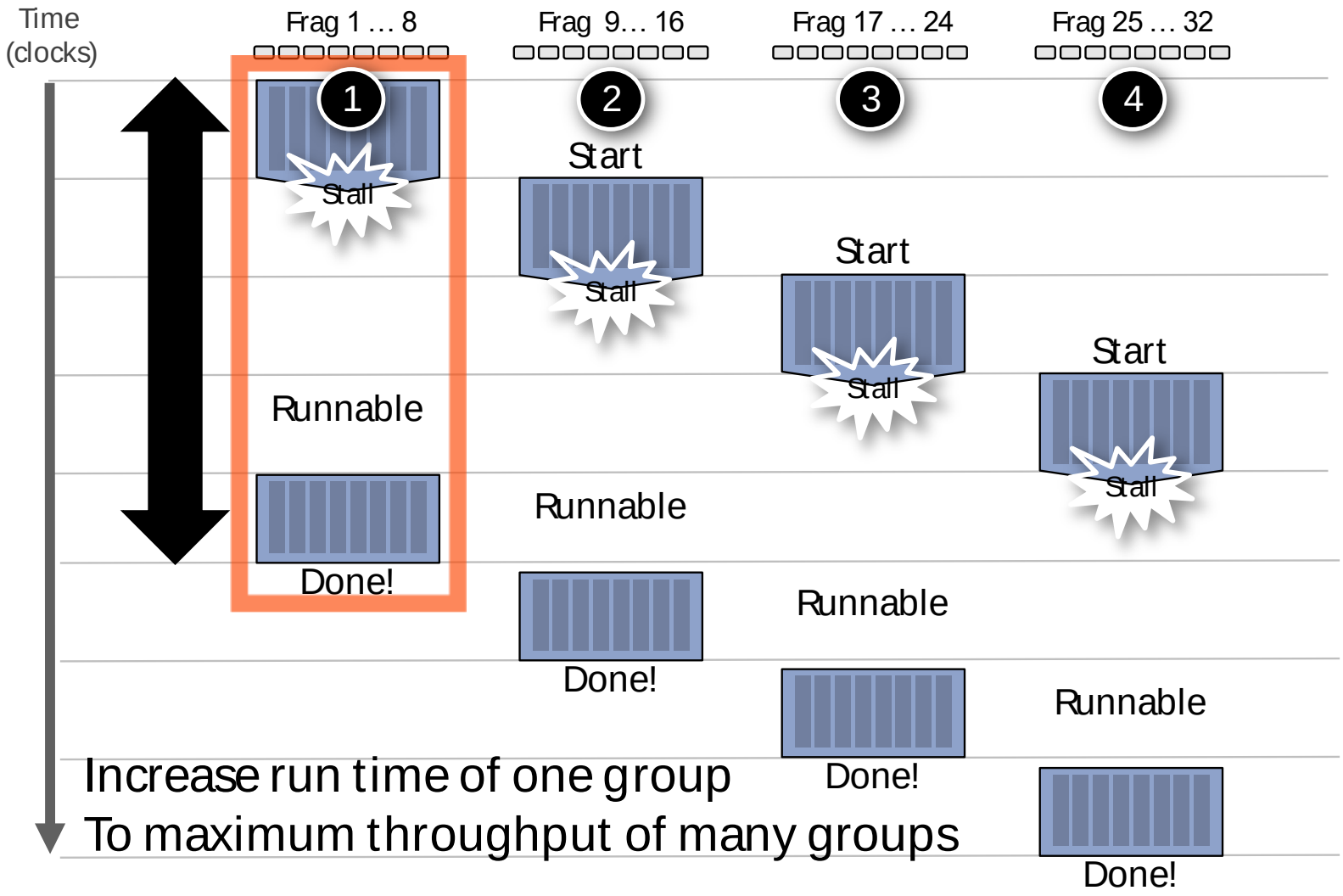
- SIMD Prozessierung impliziert nicht SIMD Instruktionen
- Option 1: Explizite Vektor Instruktionen
  - Intel/AMD x86 SSE, Intel Xeon Phi
- Option 2: Skalare Instruktionen, implizite Hardware Vektorisierung
  - Hardware entscheidet Verteilung der Instruktionsströme auf ALUs
  - NVIDIA GeForce (SIMT warps),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

- Realität: 16 bis 64 Fragmente teilen sich einen Instruktionsstrom

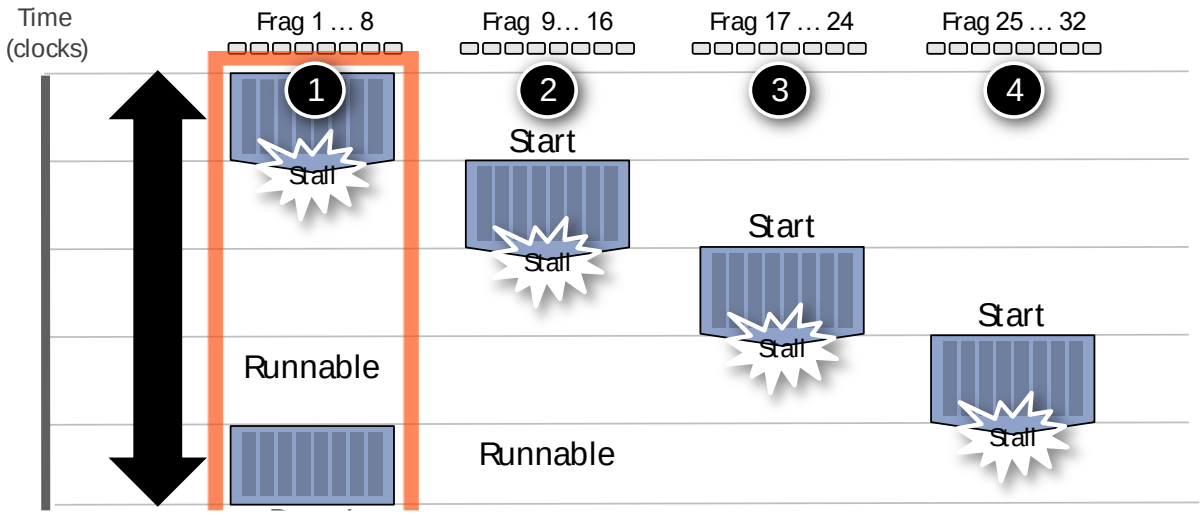
2.4.3 Stalls und Interleaved Processing

- 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.

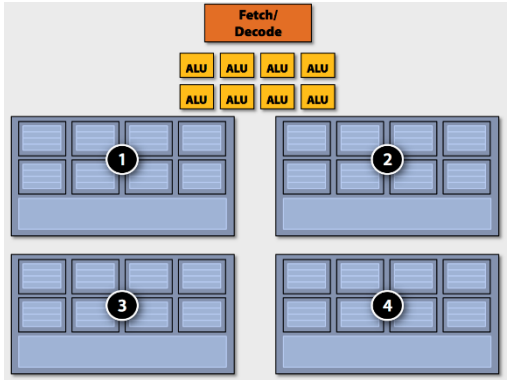
2.4.3 Durchsatz (I)



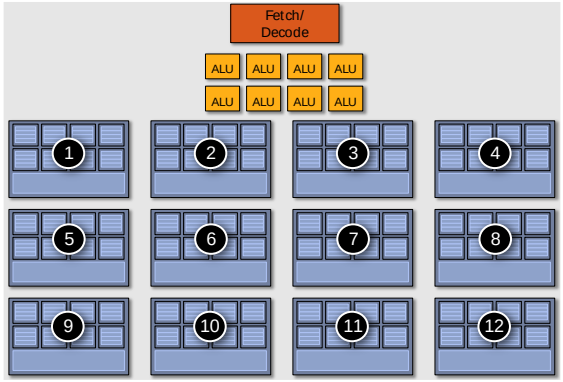
2.4.3 Durchsatz (II)



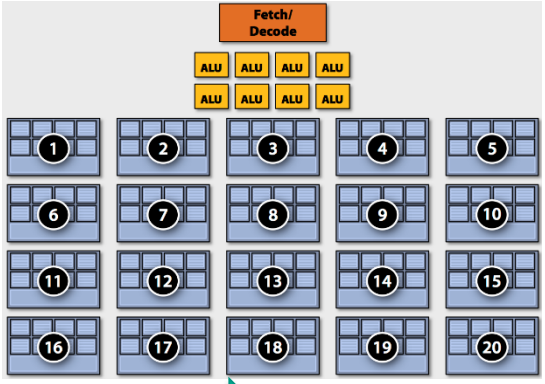
- 4 große Kontexte



12 mittlere Kontexte



20 kleine Kontexte



latency hiding ability

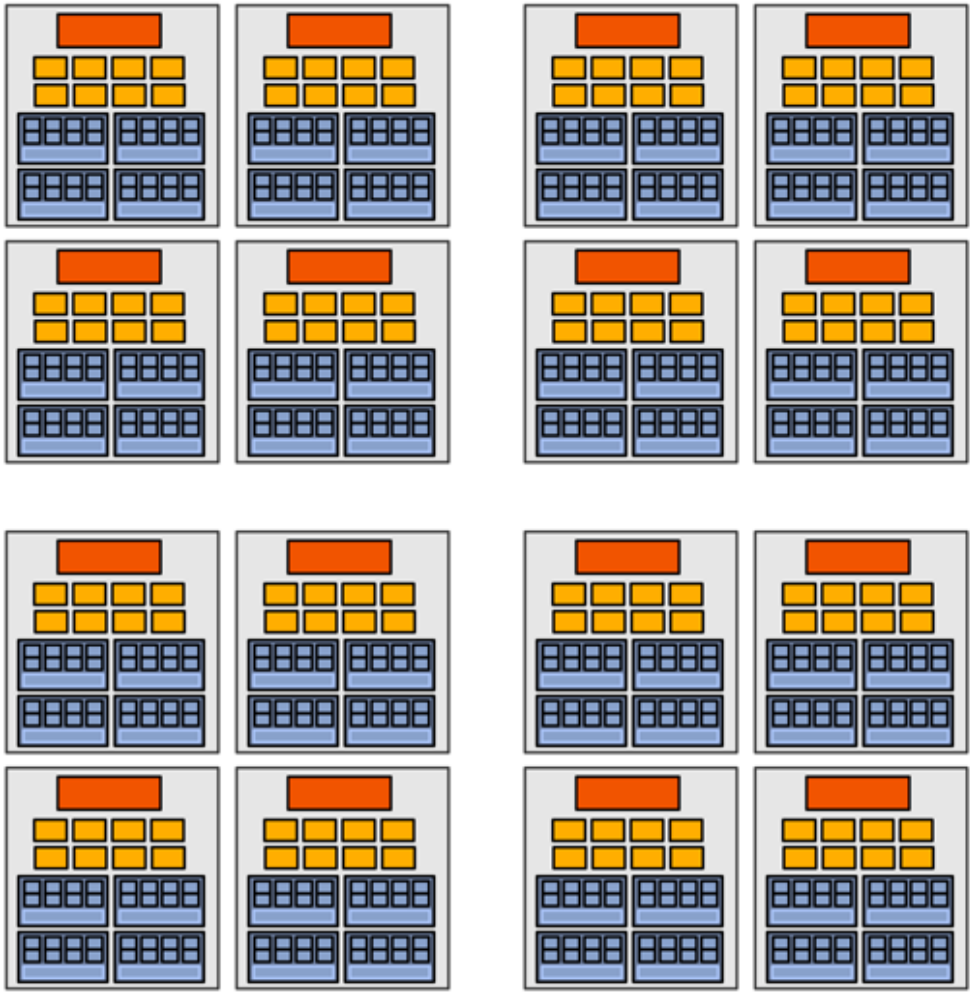
2.4.3 Beispiel

16 cores

8 mul-add ALUs per core (128 total)

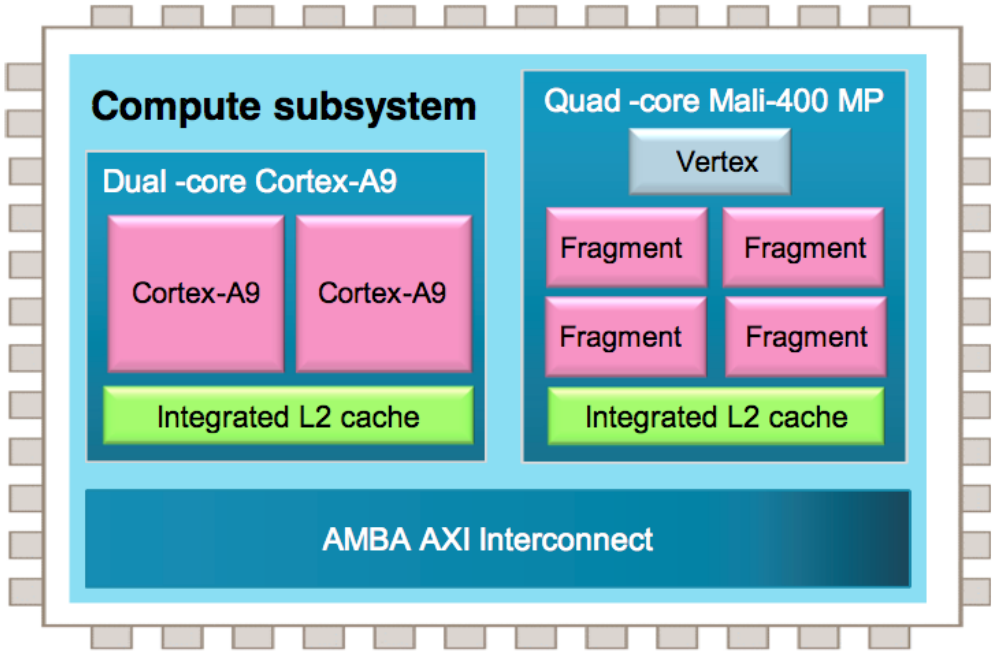
16 simultaneous instruction streams

64 concurrent (but interleaved) instruction streams



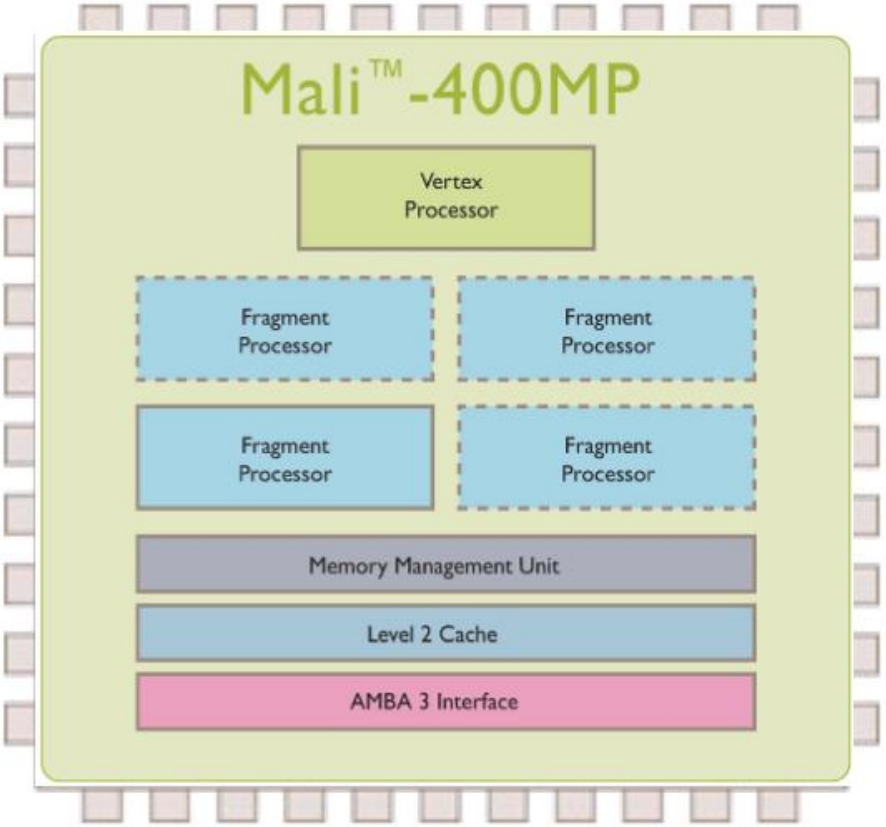
2.4.3.2 Mobile Geräte von heute

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



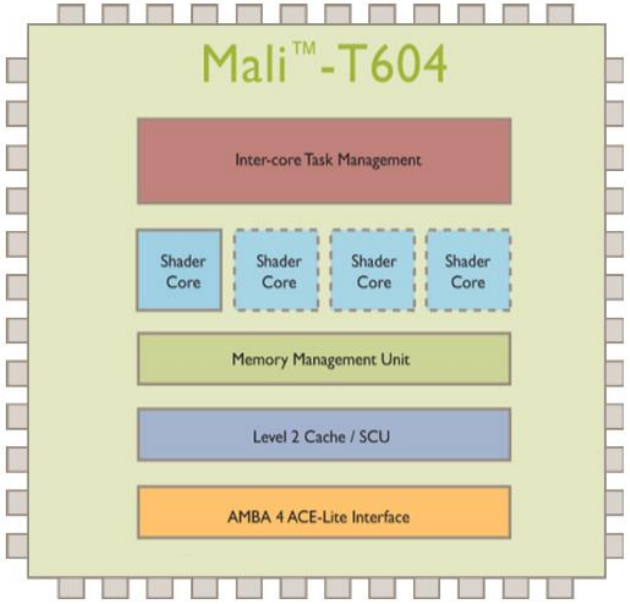
2.4.3.2 Beispiel: ARM Mali-400 MP

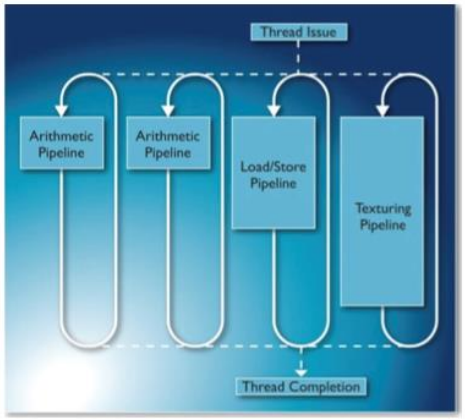
- Multi-Prozessor skalierbare Grafik Performanz
  - Bis zu 4 Fragment-Prozessoren
- 2D und 3D Grafikbeschleunigung
  - OpenGL ES 2.0 / 1.1 und OpenVG 1.1
- Hohe Performanz und Bildqualität
  - Skaliert bis 1080p Auflösungen mit Anti-Aliasing
- Effiziente Energie- und Bandbreiten-Ausnutzung
  - Integrierter 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 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





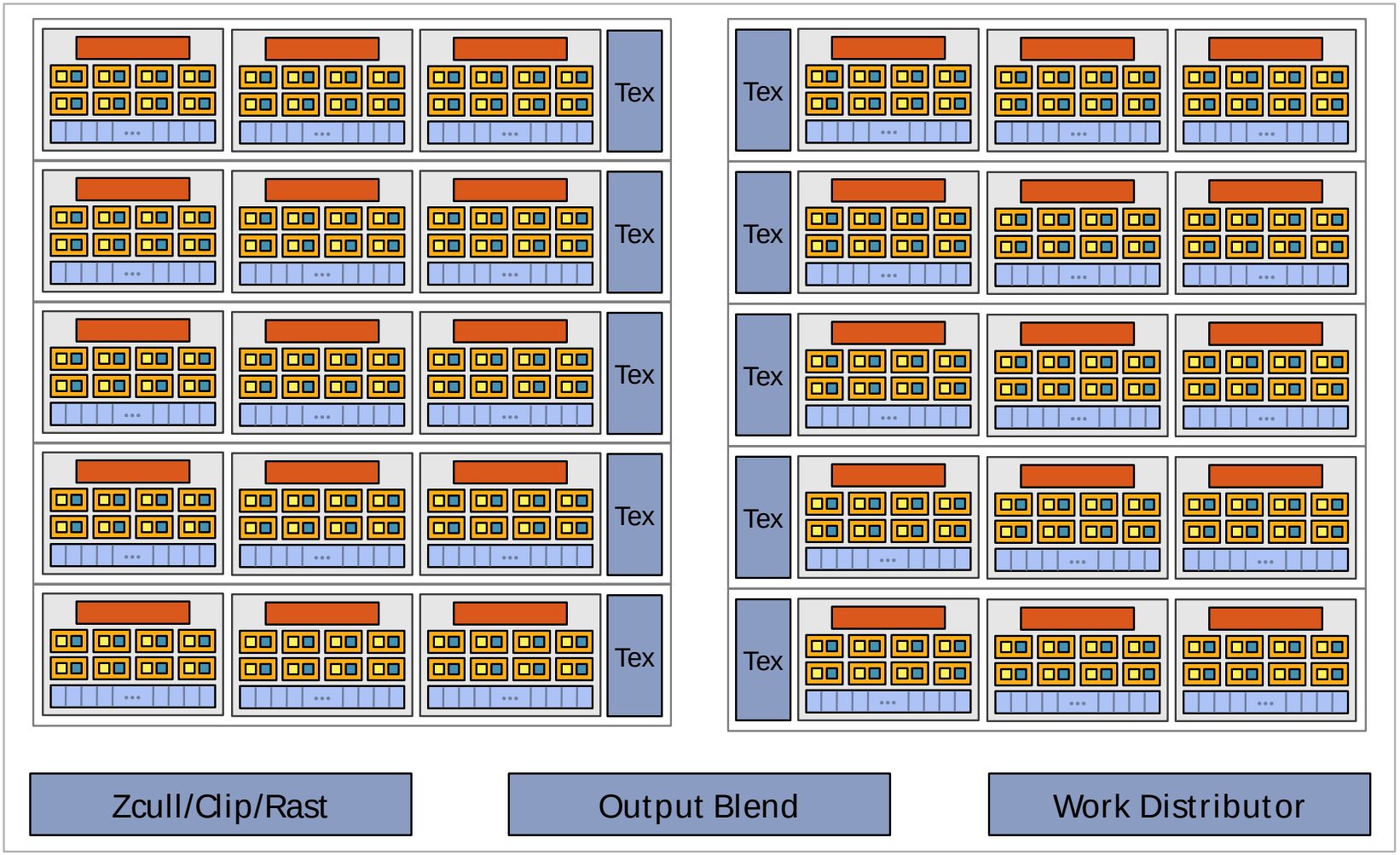
2.4.3.2 Beispiel: NVIDIA GTX 280

- NVIDIA-speak:
  - 240 stream processors
  - “SIMT execution” (automatic HW-managed sharing of instruction stream)

- Generic speak:
  - 30 processing cores
  - 8 SIMD functional units per core
  - 1 mul-add (2 flops) + 1 mul per functional units (3 flops/clock)
  - Best case: 240 mul-adds + 240 muls per clock
  - 1.3 GHz clock
  - 30 * 8 * (2 + 1) * 1.3 = 933 GFLOPS

- Mapping data-parallelism to chip:
  - Instruction stream shared across 32 fragments (16 for vertices)
  - 8 fragments run on 8 SIMD functional units in one clock
  - Instruction repeated for 4 clocks (2 clocks for vertices)

2.4.3.2 Beispiel: NVIDIA GeForce GTX 280



2.4.3.3 Beispiel: Nvidia GPU comparison

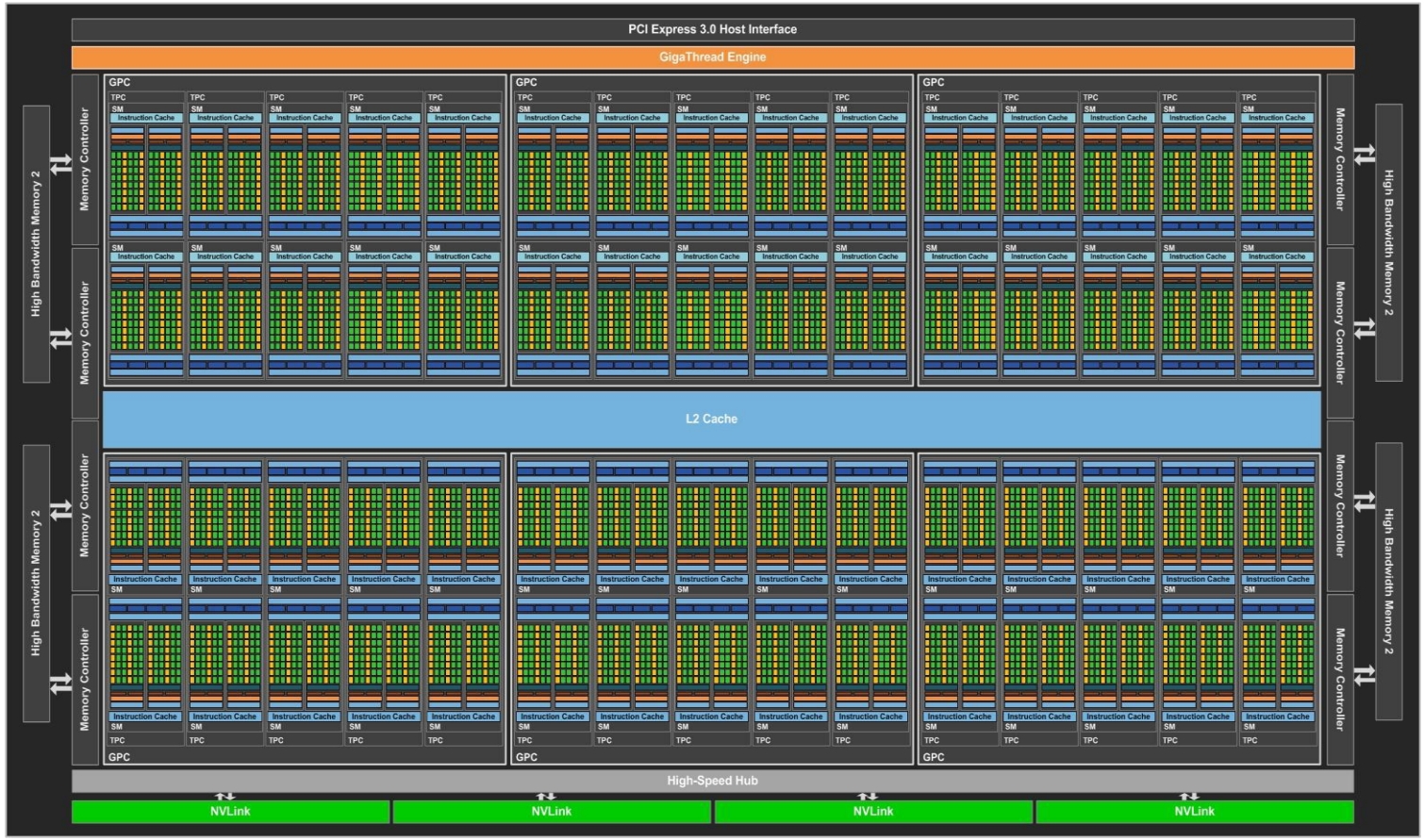
Tesla Products	Tesla K40	Tesla M40	Tesla P100
GPU	GK110 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)
SMs	15	24	56
TPCs	15	24	28
FP32 CUDA Cores / SM	192	128	64
FP32 CUDA Cores / GPU	2880	3072	3584
FP64 CUDA Cores / SM	64	4	32
FP64 CUDA Cores / GPU	960	96	1792
Base Clock	745 MHz	948 MHz	1328 MHz
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz
Peak FP32 GFLOPs[1]	5040	6840	10600
Peak FP64 GFLOPs[1]	1680	210	5300
Texture Units	240	192	224
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB
L2 Cache Size	1536 KB	3072 KB	4096 KB
Register File Size / SM	256 KB	256 KB	256 KB
Register File Size / GPU	3840 KB	6144 KB	14336 KB
TDP	235 Watts	250 Watts	300 Watts
Transistors	7.1 billion	8 billion	15.3 billion
GPU Die Size	551 mm²	601 mm²	610 mm²
Manufacturing Process	28-nm	28-nm	16-nm FinFET

[1] The GFLOPS in this chart are based on GPU Boost Clocks.

© NVIDIA, Whitepaper - NVIDIA Tesla P100

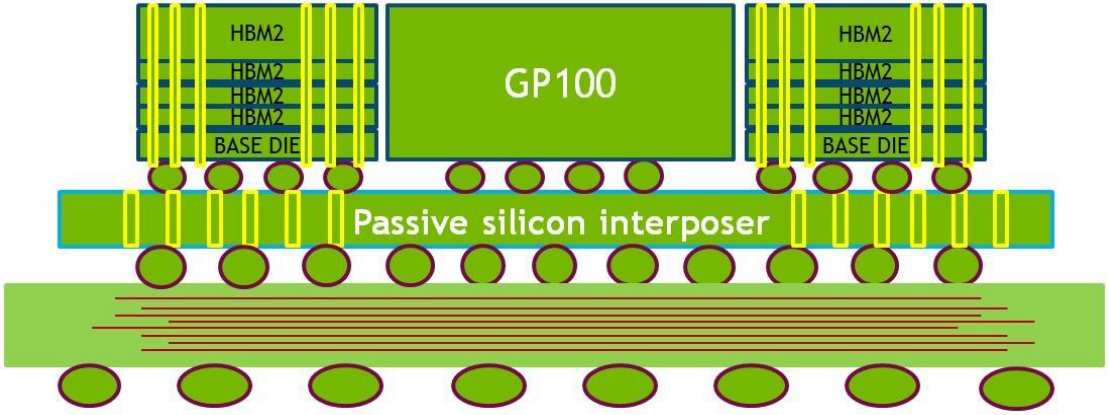
2.4.3.3 Beispiel: Nvidia GP100 SM Unit

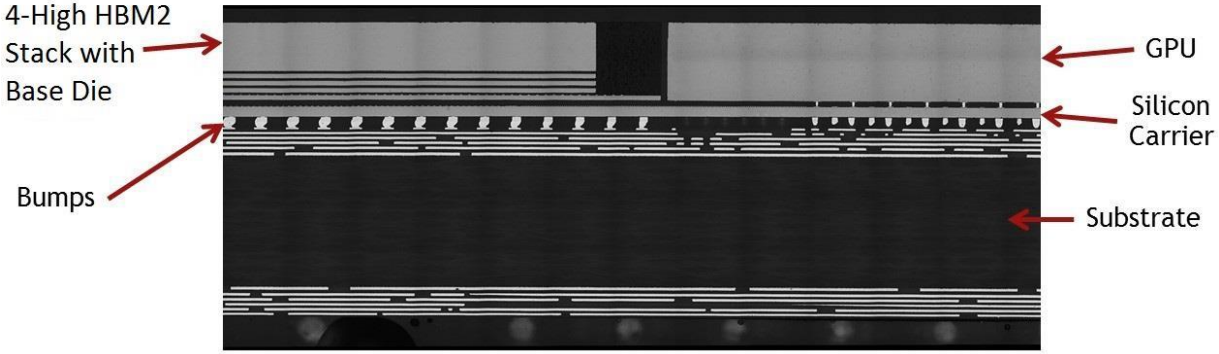
2.4.3.3 Beispiel: Nvidia GP100 Full GPU with 60 SM Units



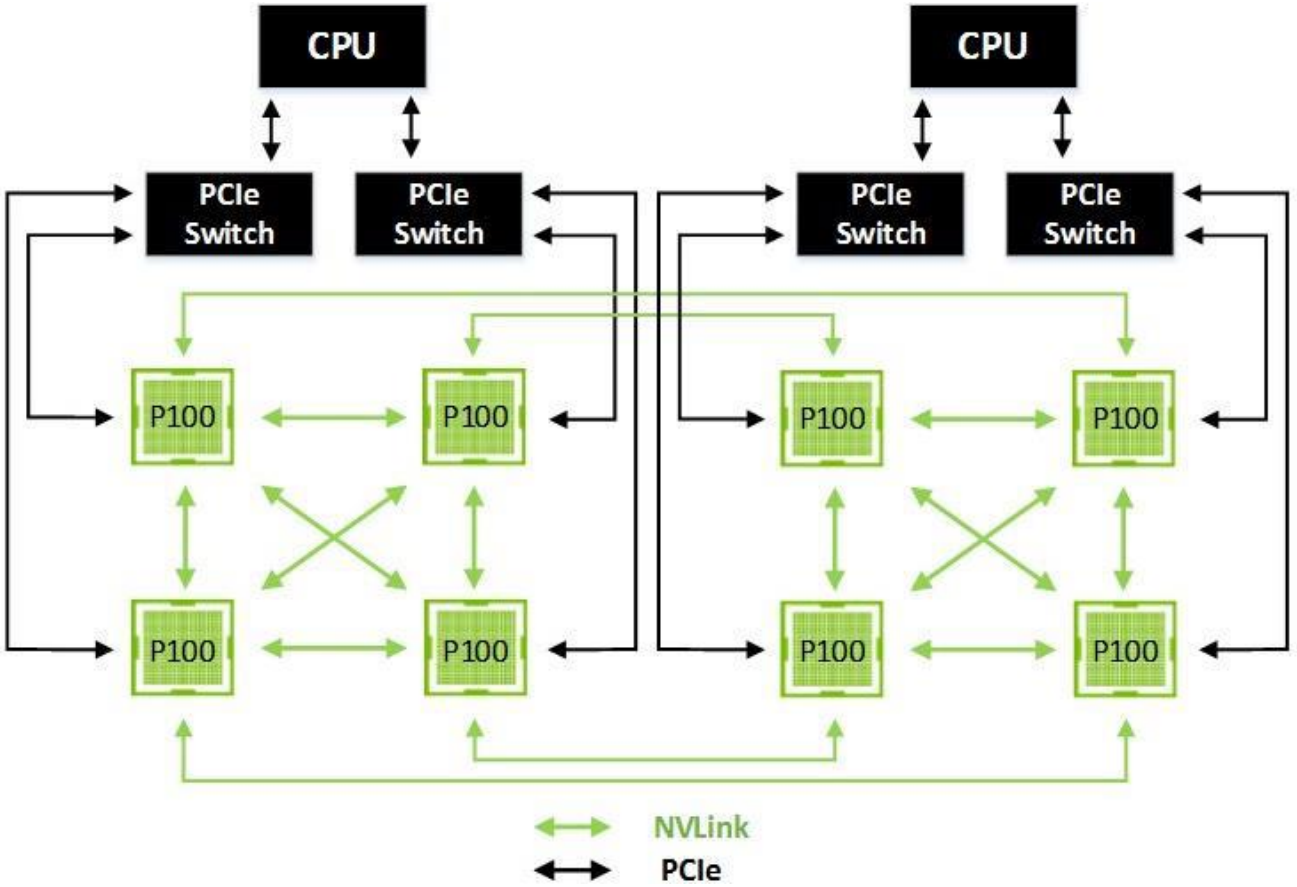
© NVIDIA, Whitepaper - NVIDIA Tesla P100

2.4.3.3 Beispiel: Nvidia GP100 Speicheranbindung





2.4.3.3 Beispiel: Nvidia GP100 Multi GPU Systeme und NVLink



© NVIDIA, Whitepaper - NVIDIA Tesla P100 https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

- 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?