# Adaptive Rechensysteme

## *Anwendungen, Architekturen und Werkzeuge*

**Andreas Koch**
**TU Darmstadt**
**FG Eingebettete Systeme und ihre Anwendungen**



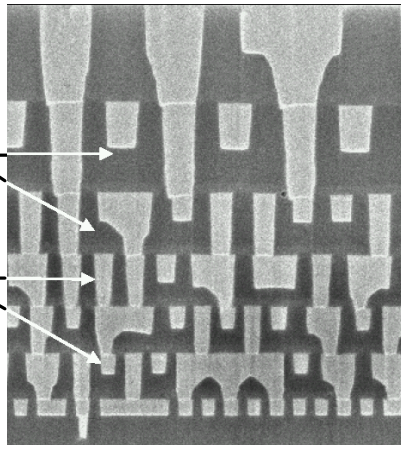
# Rekonfigurierbares Rechnen

- Niedergang konventioneller Prozessoren
- Praktische Anwendungen
- Grundlagen
- Hardware-Architekturen
- Software-Werkzeuge

# Chip-Fertigungstechnologie

- Immer kleinere Strukturbreiten
  - 250nm - 180nm - 130nm - 90nm - (65nm) - ...
- Neue Materialien
  - Verbesserte Isolation
    - * Low-K
    - >Niedrigere Kapazität
  - Verbesserte Leiter
    - * Kupfer statt Aluminium
    - > Niedrigerer Widerstand

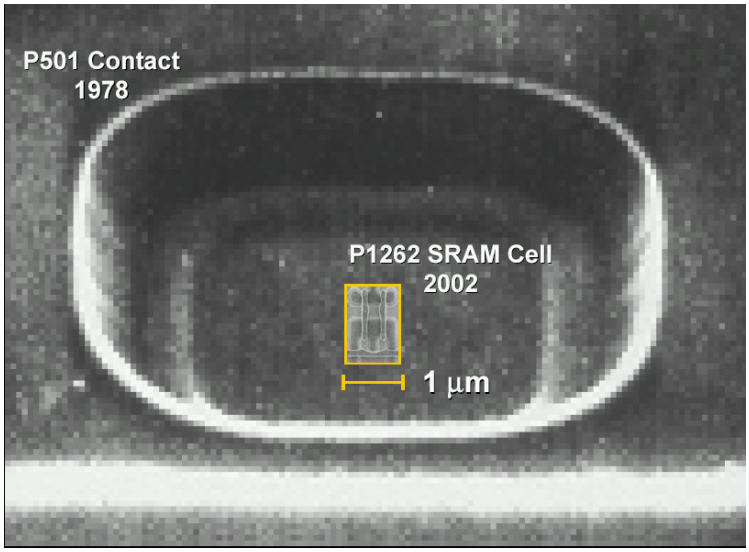
Vertikaler Schnitt



Quelle: Intel



# Höhere Packungsdichte




Quelle: Intel

# Moores Gesetz

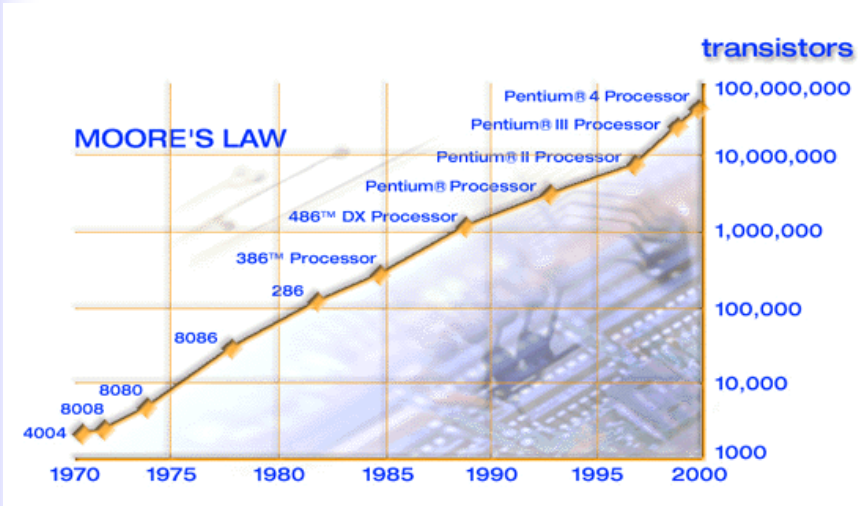
- **Gordon Moore**
  - **Mitgründer von Intel**
  - **Vorhersage von 1965, verfeinert 1975**

**2x Transistoren / 18 Monate**

- **Praktische Auswirkungen?**



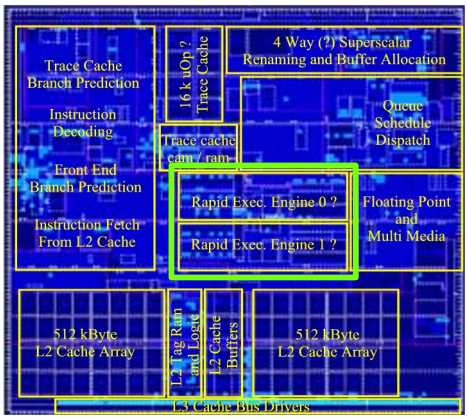
# Moores Gesetz - Beispiel




Quelle: Intel

# Praktische Einschränkung

- Nicht: 2x *Rechenleistung* / 18 Monate



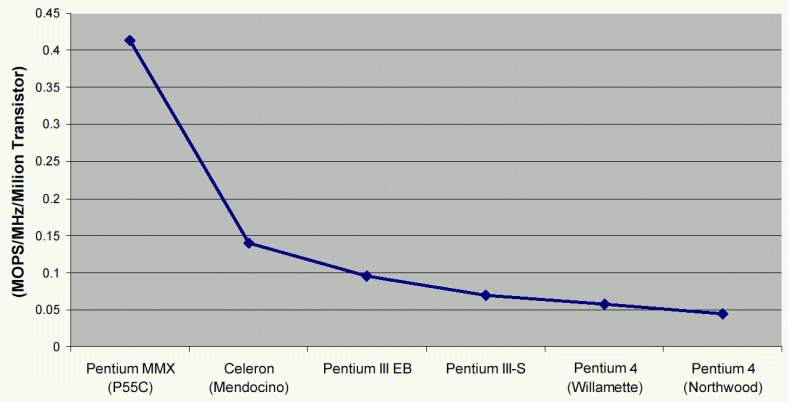

Quelle: H. de Vries



# Auswirkung auf Prozessoren

- Aktuelle Fertigungsprozesse: 300 Mio. Transistoren
  - ~1 Milliarde Transistoren in 2005 (Intel Montecito)
- Wofür diese Transistoren in einer CPU verwenden?
  - Größere Caches auf dem Chip
    - Intel Montecito: 18MB in L1 ... L3 Caches
  - Höhere Integration von Komponenten
    - Ansteuerung für externen Speicher integrieren
  - Mehrere Prozessoren auf einem Chip
    - HP PA-RISC 8800: 2x PA-RISC 8700
    - Intel Montecito: 2x Madison 9M (Itanium Architektur)
- ✗ ... nicht mehr effektiv in Relation zur Transistorzahl

# Sinkende Effizienz

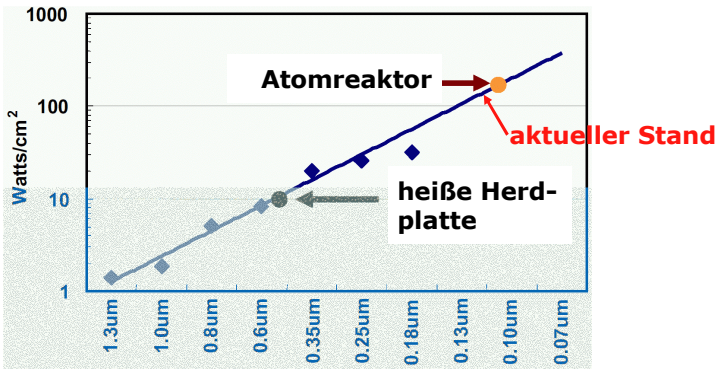



Quelle: J. Wawrzynek



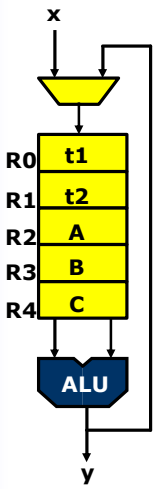
# Energiedichte

- **Energieverbrauch**
  - **Beim Umschalten von Transistoren**
    - Immer höhere Taktgeschwindigkeiten
  - **Leckströme („undichte" Transistoren)**
    - Bei kleineren Geometrien




Quelle: V. De, S. Borkar

# Konventioneller Prozessor



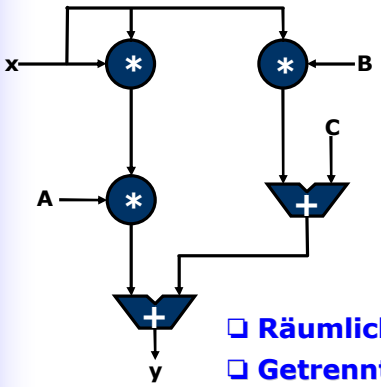

```
t1 := x
t2 := A * t1
t2 := t2 + B
t2 := t2 * t1
y  := t2 + C
```

$y = A\,x^2 + B\,x + C$

- Zeitliche Verteilung der Berechnung
- Wiederverwendung von Recheneinheiten
- Nur eine Operation pro Zeitschritt
- Gesteuert durch variable Software
- Berechnungsuniversell



# Konfigurierbarer Prozessor




$y = A\,x^2 + B\,x + C$

- Räumliche Verteilung der Berechnung
- Getrennte, dedizierte Recheneinheiten
- Mehrere Operationen pro Zeitschritt
- Gesteuert durch festes Steuerwerk
- Berechnungsuniversell durch Rekonfigurierbarkeit

# Ausnutzung des Ansatzes

- An Anwendungen angepasste Prozessor-Struktur
  - Wie bei anwendungsspezifischen Schaltungen (ASIC)
    - Dort aber nach Chip-Fertigung feste Struktur
- Idee: *Rekonfigurierbare* Recheneinheit (RCU)
  - Variable Struktur auch nach der Fertigung
  - Perfekt an individuelle Anwendungen anpassbar
- Sogar auf individuelle Eingabewerte *spezialisierbar*
  - Angepasste Struktur für
    - Aktuellen Schlüssel bei Ver-/Entschlüsselung
    - Aktuelle Suchsequenz beim Vergleich von Gensequenzen
    - Aktuelle Filter-Regeln bei Netzwerk-Firewalls



# Einschränkungen

- ✖ Aber nicht für alle Arten von Operationen sinnvoll
  - Ausnahmebehandlung
    - Fehler in Eingabedaten
  - Bibliotheksfunktionen wie printf()
    - Sehr kompliziert
    - In der Regel nicht zeitkritisch
- ➡ Dafür besser Standardprozessor (CPU)
  - Wiederverwendung von Recheneinheiten
    - Für unterschiedliche Aufgaben
    - Kein Verschwenden von „rechnender" Fläche

# Adaptive Rechensysteme

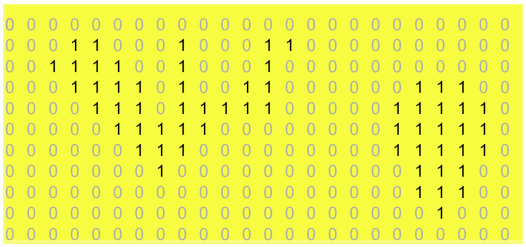
- Adaptive Rechensysteme (ACS) *kombinieren*
  - Programmierbarkeit
    - Prozessor mit fester Struktur (CPU)
  - Rekonfigurierbarkeit
    - Recheneinheiten mit variabler Struktur (RCU)
- Ziel: Hohe Rechenleistung bei hoher Effizienz



# Einige ACS Erfolge

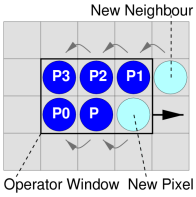
- Geschwindigkeit: *Gensequenzvergleich*
  - 1300x MasPar MP1
- Integrationsdichte: *LMS Korrelator*
  - 36x TMS320C54 DSPs → 1x ACS Chip
- Energieverbrauch: *VCELP Sprachkompression*
  - 25% Energie eines 1V DSP
- Vergleich mit aktuelleren Prozessoren

# Regionserkennung



Schwarz/Weiß-Bild

**Operatorfenster**



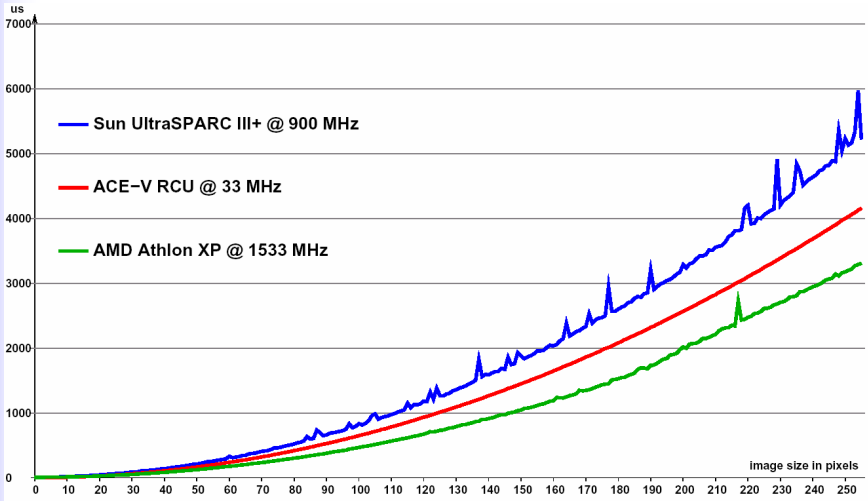

Matrix aus Region IDs

**Zusammenhangstabelle**

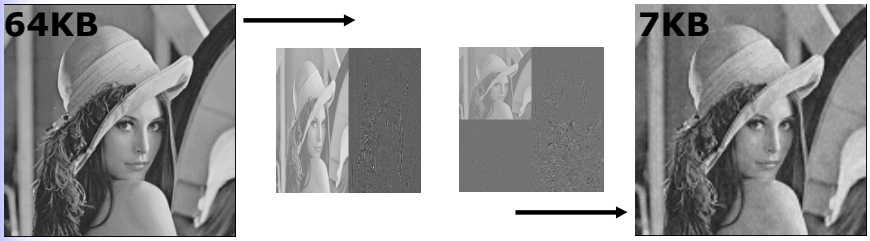
| Object ID | Adjacent to |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 4 |

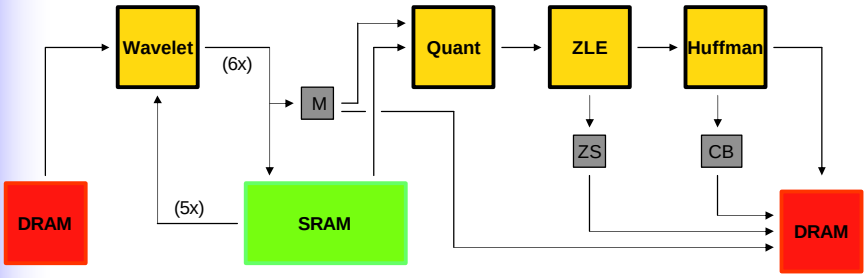


# Unoptimierte Rechenleistung



# Wavelet-Bildkompression

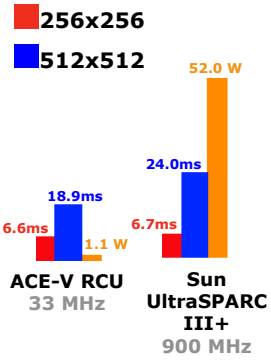


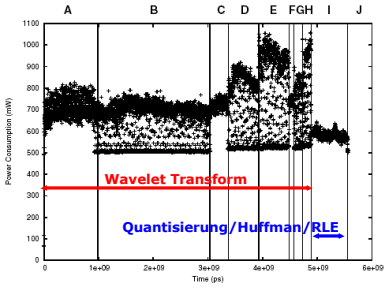



# Eckdaten

**Ausführungszeit**




**Leistungsaufnahme**




➡ **Adaptive Rechensysteme vielversprechend**

☆ **Insbesondere im Embedded-Bereich**

# Biologische Systeme

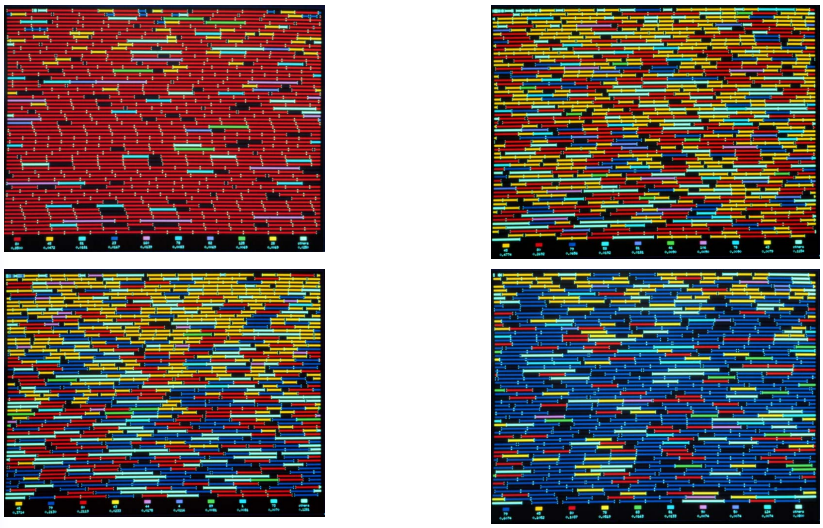
- Untersuchung von Populationen
  - Konkurrenz und Koexistenz
  - Populationsökologie
    - Räuber/Beuteverhältnis
  - Artenvielfalt
    - Interaktion zwischen Arten
  - Entwicklung von
    - Wirten
    - Parasiten
    - Hyper-Parasiten
- Idee: Nachbildung im Rechner
  - Beschleunigte Simulation
  - „Artificial Life"



# TIERRA

- Softwaresimulation mittels MIMD-Prozessor
  - Organismen sind parallele Programme
- Spezielle Prozessorfähigkeiten erlauben:
  - Mutation
    - Änderung der Instruktionssemantik
  - Rekombination
    - Austausch von Code zwischen Programmen
  - Tolerieren fehlerhafter Instruktionen
    - „Erkrankung"
  - Abbruch bei zu vielen Fehlern
    - „Tod"

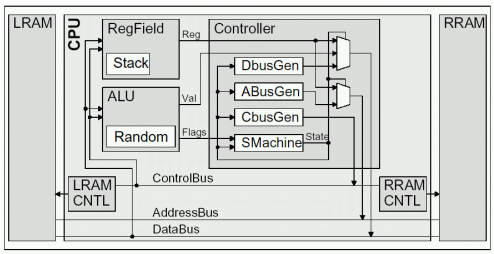
# Beispiel für Evolution



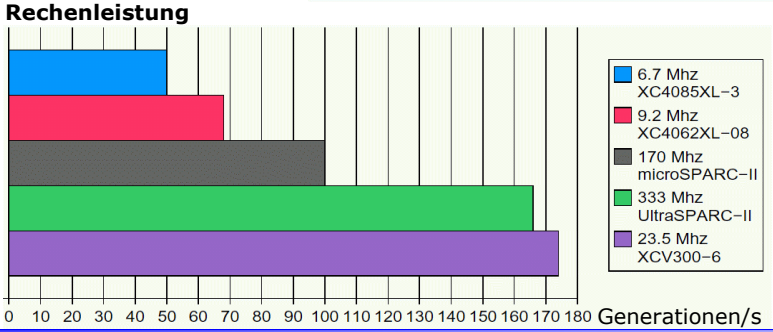


# TIERRA Spezialprozessor

- X86-ähnlich
- Spezialfunktionen, z.B.
  - Mutation
  - Marker als Sprungziel
- OS-Dienste in Hardware
  - Prozessverwaltung
  - Speicherverwaltung
- RISC-Ansatz zu groß
  - Komplexer Microcode




Rechenleistung



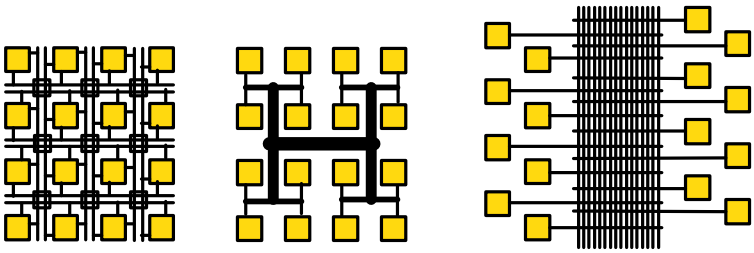
# ACS - Komponenten

- **Einfache CPU**
  - „Grundlast" der Rechenleistung
  - Systemsteuerung und I/O
  - Führt Abläufe ungeeignet für direkte HW-Umsetzung aus
- **Flexible RCU**
  - „Spitzenlast" der Rechenleistung
  - Führt HW-geeignete datenpfad-orientierte Abläufe aus
- **Speicher**
  - Gemeinsamer Hauptspeicher für CPU+RCU
    - RCU kann eigenständig auf Speicher zugreifen
  - Lokaler Speicher für RCU (optional)
- **Spezialschnittstellen an RCU (optional)**



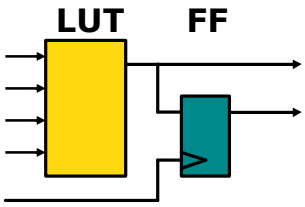
# Aufbau einer RCU

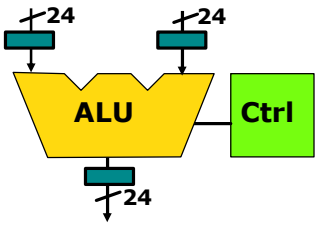
- **Fundamente**
  - Konfigurierbares Verbindungsnetzwerk
  - Konfigurierbare Funktionsblöcke
- **Viele Variationsmöglichkeiten!**
- **Beispiele für Verbindungsnetze**



Symmetrische Matrix | Hierarchische Matrix | Kreuzschienen-verteiler

# Beispiele für Funktionsblöcke

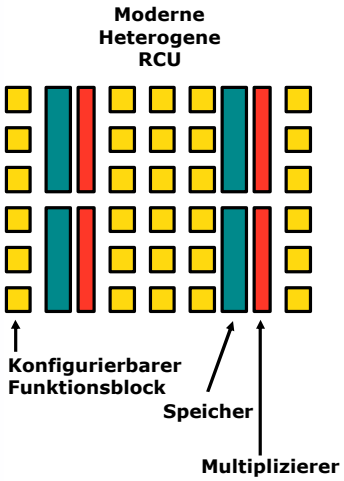



- **Wertetabelle (LUT)**
  - 4 Eingänge
  - 1 Ausgang
  - Bearbeitet Einzelbits
  - Speicherelement (FF)
  - Häufig in FPGAs
  - Universell einsetzbar
- **Arithmetisch-logische Einheit (ALU)**
  - 2 Eingänge
  - 1 Ausgang
  - Bearbeitet Datenworte
  - Registerbänke
  - Lokale Steuerung
  - Spezialisiert auf Rechenaufgaben

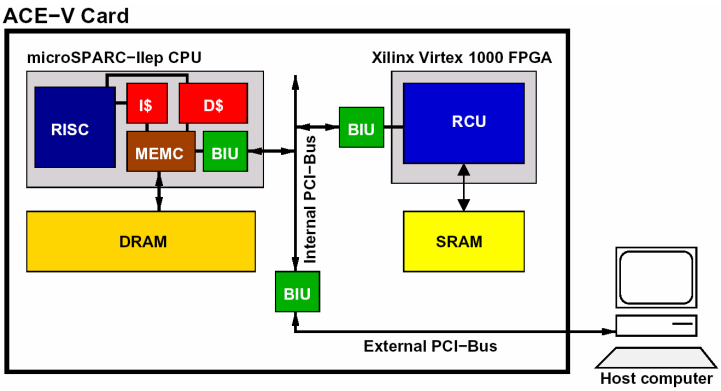


# Heterogene RCUs




- **Eingebettete feste Blöcke**
  - Schnelle Multiplizierer
  - Große Speicher (~Mb)
  - Sogar komplette CPU(s)
  - Taktmanipulation
  - Spezielle I/O Schnittstellen
- ✓ **Höhere Rechenleistung**
- ✓ **Bessere Flächeneffizienz**

# Diskrete ACS Architektur

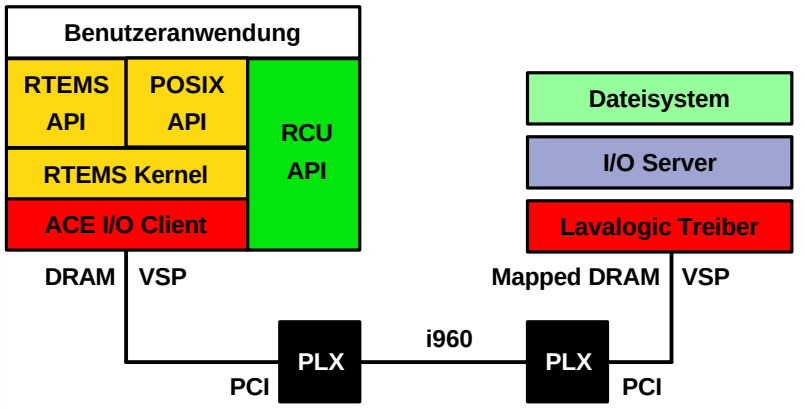



- Beispiel „ACE-V" Karte (TU BS, Abt. E.I.S.)
  - 100 MHz microSPARC-IIep RISC als CPU
  - Xilinx Virtex 1000 FPGA als RCU
  - RCU-lokaler SRAM Speicher, gemeinsames DRAM
  - Kommunikation über langsamen lokalen PCI-Bus



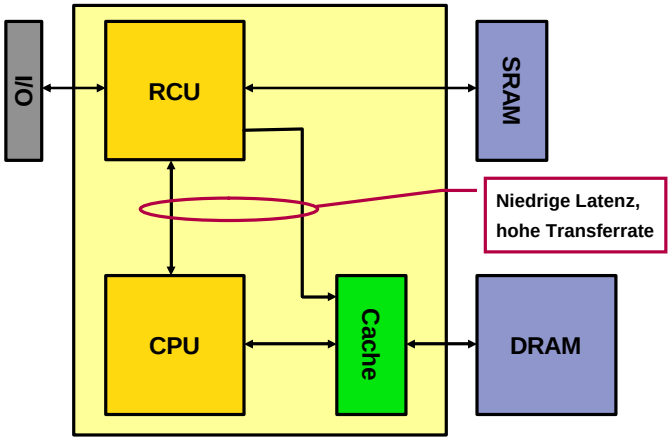
# Software-Architektur

- Embedded-System Ansatz
  - Eigener Port von Echtzeitbetriebssystem RTEMS
  - Eigene Treiber und APIs für Spezialhardware und RCU



# Ein-Chip ACS Architektur

- Viele Komponenten integriert auf einem Chip
  - Altera Excalibur, Xilinx Virtex-2 Pro und -4 FX, ...






# ACS-Programmierung

- Wandel im Laufe der Zeit
- ❶ Sehr hardware-nahe Formen
  - Schaltpläne
- ❷ Abstraktere Sprachen
  - Konventionelle Hardware-Beschreibungssprachen
    - Verilog, VHDL
  - Spezialsprachen
    - Lava, Pebble, Ruby, PLASMA, Lola, ...
- ❸ Höhere Programmiersprachen
  - C, C++, Java
  - MATLAB
- In Teilgebieten auch graphische Beschreibung
  - Signalverarbeitung
- ➡ Evolution der Werkzeugflüsse
  - Bis herunter zur geometrischen Ebene („Layout")

# Hochsprachen-Compiler

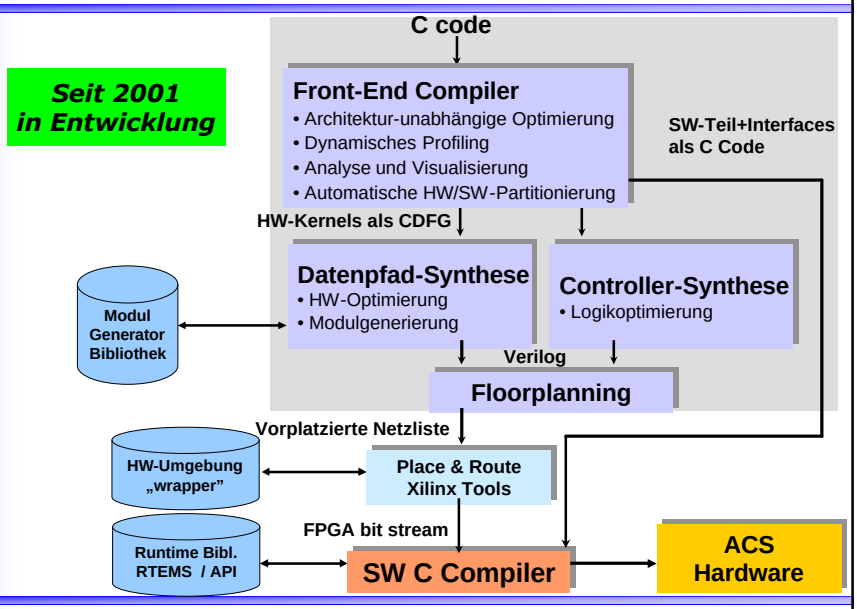
- Ursprung im Nimble Projekt zusammen mit
  - Synopsys, UC Berkeley, Lockheed Martin
- Eingabesprache ist ANSI/ISO C
  - Keine Einschränkungen („dusty deck")
  - Breite Benutzerbasis
- Grundlegende Ideen:
  - ILP in Schleifen ausnutzen (noch kein TLP!)
  - Gemeinsamer Speicher zwischen CPU und RCU
    - Zeiger austauschbar zwischen HW und SW Ausführung
  - Enge Kopplung zwischen CPU und RCU
    - Schneller Wechsel HW ←→ SW Ausführung möglich
  - Rekonfigurierbarkeit der RCU berücksichtigen



# Nimble Ergebnisse

- Wavelet-Bildkompression auf 133MHz ACS
  - 25 Schleifen insgesamt
  - davon 21 mit dem Nimble-Ansatz auf RCU realisierbar
  - dann 98.9% der gesamten Rechenzeit auf RCU
  - Beschleunigung gegenüber SW auf MIPS-II CPU: 3.25x
- DES-Keysearch auf 40MHz ACE-V
  - 12 Schleifen insgesamt
  - davon 4 via Nimble auf RCU realisierbar
  - dann 95.3% der gesamten Rechenzeit auf RCU
  - Beschleunigung gegenüber SW: 137x
    - inklusive Konfiguration!
  - Trick: Benutzerdefiniertes Makro für permute()

# Compiler COMRADE






# Front-End Compiler 1

- Konzentriert sich auf Schleifen
  - Benötigen Großteil der Rechenzeit
  - ➡ Als Hardware-Kernels ausführen
- Erlaubt auch HW-ungeeignete Anweisungen
  - Ein-/Ausgaben (z.B. printf)
  - Fließkomma-Berechnungen
- Programmanalyse durch Profiling erfaßt u.a.:
  - Häufigkeit von HW-ungeeigneten Anweisungen

# Front-End Compiler 2

- Bearbeitung HW-ungeeigneter Anweisungen
- Einfaches Beispiel:

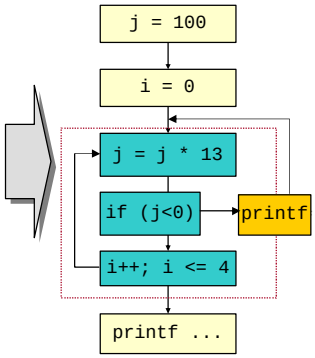
*Anwendungsprogramm*

```
#include <stdio.h>
main(int argc, char *argv[])
{
int i, j, k;
   k = j = atoi(argv[1]);
   for (i=0; i<4; i++)
   {
      j = j * 13;
      if (j<0) printf("Error\n");
   }
   printf( "result j = %d == %d",
           j, k*13*13*13*13);
}
```

Hardware Kernel

Nicht-HW Anweisung

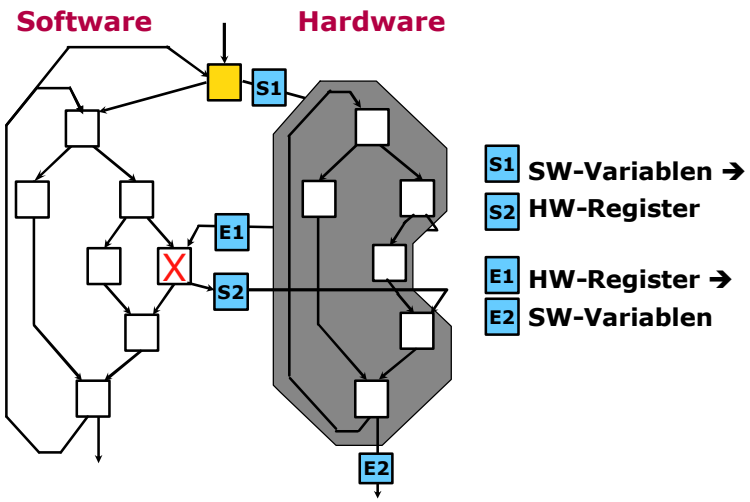
*HW/SW Partitionierung*



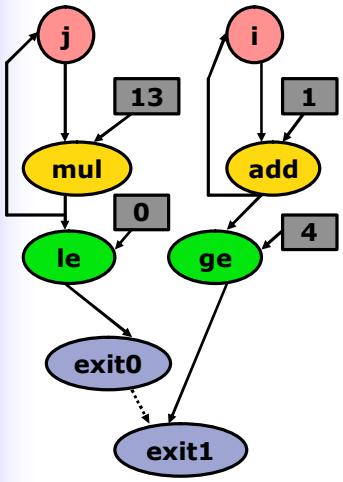



# Front-End Compiler 3

- Effizienter Wechsel zwischen HW/SW Ausführung



# Datenpfad-Synthese

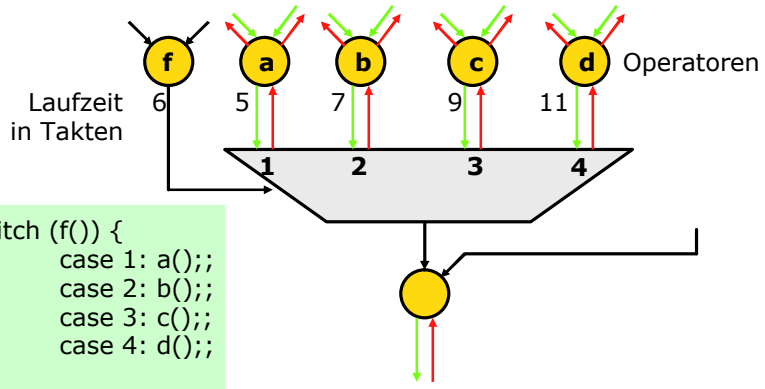



- Hardware-Kernel
  - Compiliert in CDFGs
- Operatoren
  - Abbildung auf HW-Instanzen
- Modulbibliothek
  - HW-Realisierungen für alle C Basisoperatoren
  - Grobe Abschätzungen über Zeit-/Flächenbedarf verfügbar
  - Liefert Instanzen als Layouts



# Steuerwerk-Synthese

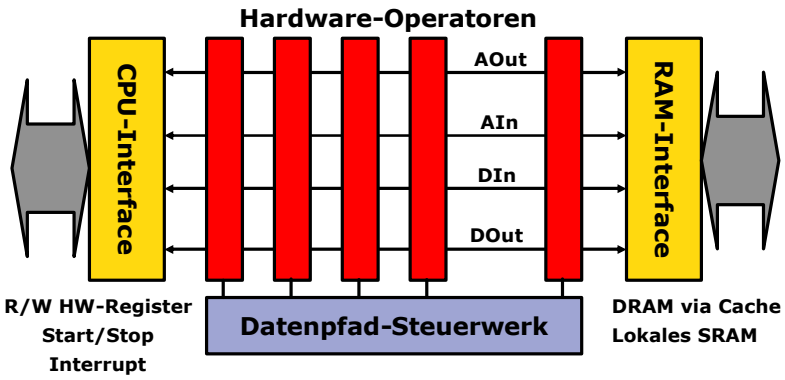
- Dynamisches Scheduling
- Spekulative Ausführung
- Abbruch von fehlspekulierten Berechnungen




```
switch (f()) {
        case 1: a();;
        case 2: b();;
        case 3: c();;
        case 4: d();;
}
```

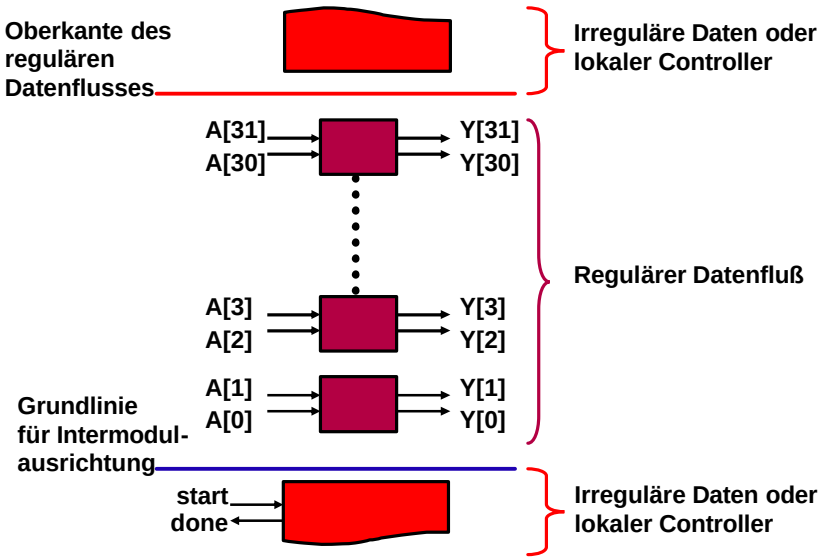
# Recheneinheiten auf RCU

- Datenpfad aus Multi-Bit HW-Operatoren auf FPGA
- Eingebettet in HW-Umgebung („wrapper")
  - CPU / RAM - Interfaces
  - Zentrale und Operator-lokale Steuerung

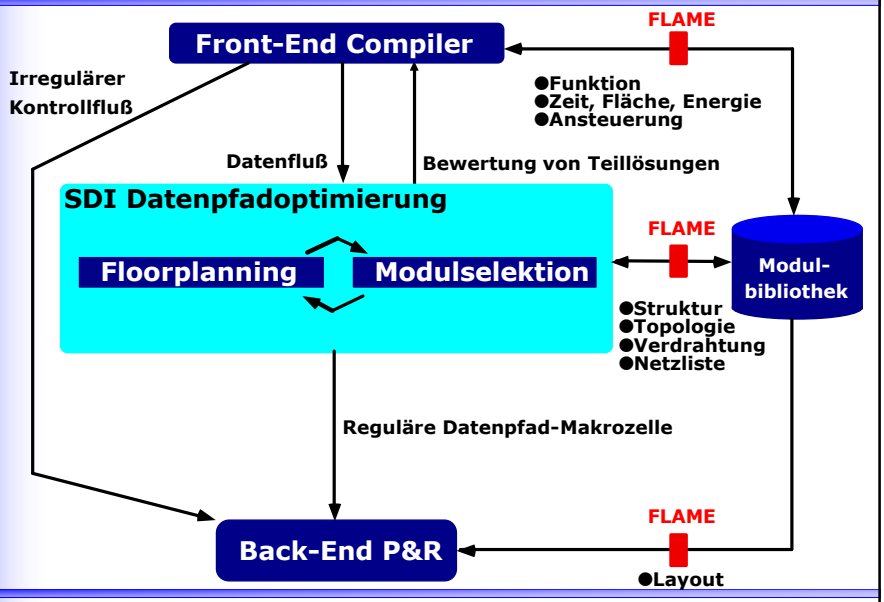





# Anreihbare Modul-Layouts



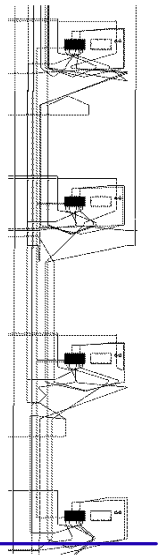
# Sicht auf die Hardware





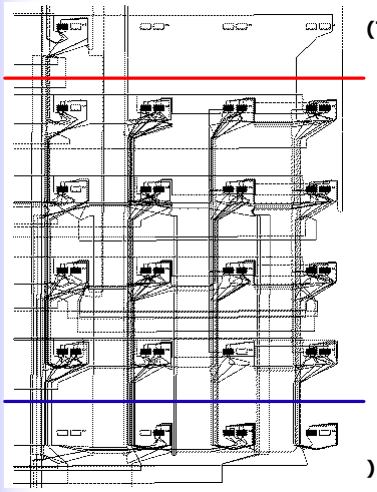

# Zeitverhalten+Flächenbedarf

```
(TECHNOLOGY "Xilinx" "Virtex" "XCV50PQ240I" "-4"
    (STATUS QUERYOK)
    (DPEXTENT
        ("CLB" (RECT 24 16 0 0)))
    (UNIT
        (TIMESCALE -10))
    (TIMING
        (("add")
            (FIXED
                (REQUIRED   (("a" 7 0) ("b" 7 0))    0 0 0)
                (ARRIVAL    (("sum" 7 0))            0 24)
                (CYCLETIME 24))))
    (AREA
        ("CLB" 4 4))
```



Grundlinie

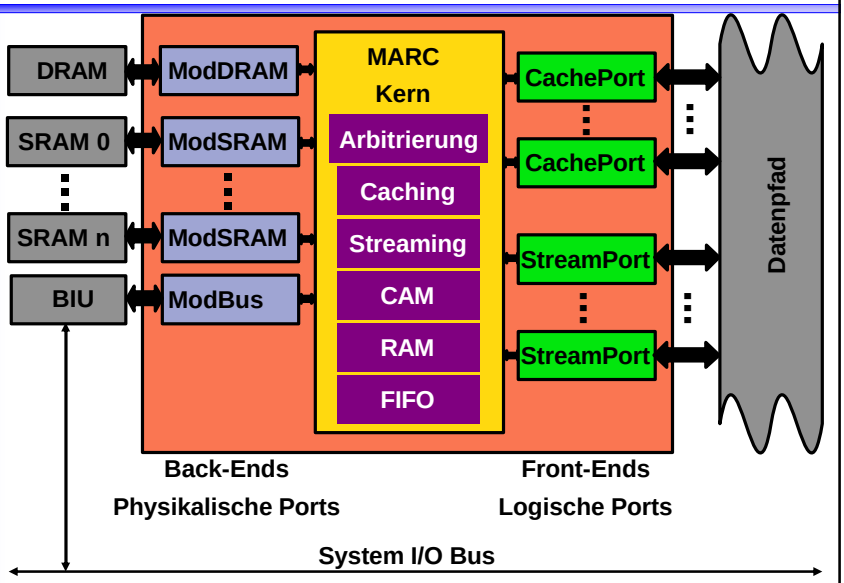
# Topologie



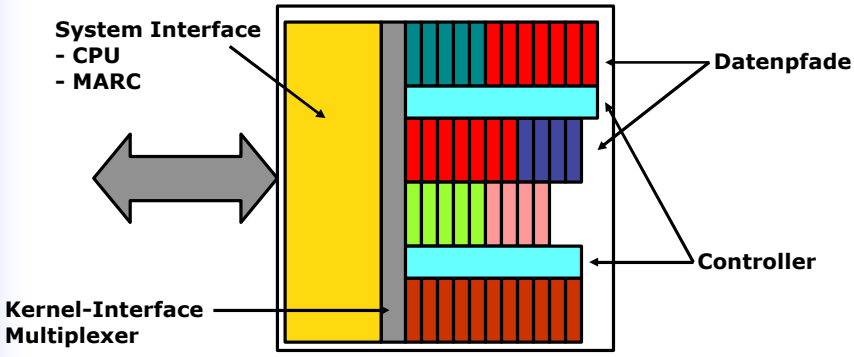
```
(TECHNOLOGY  "Xilinx" "Virtex"
             "XCV50PQ240I" "-4"
  (STATUS QUERYOK)
  (DPEXTENT
    ("CLB" (RECT 24 16 0 0)))
  (TOPOLOGY (MATRIX
    (SHAPE
      (("CLB" (RECT 4 6 0 0)))
      (ORIGIN "CLB" 0 1))
    (PORTLOC
      (PORTS (("a" 7 0) ("b" 7 0) ("out" 7 0))
        (PITCH 2 1)
        (COORD 0 0)
        (FOLDING HORIZONTAL LINEAR)))
))
```



# Speicheranbindung MARC



# Hardware-Integration




- Schnell umschaltbar über Multiplexer
  - Datenpfade
  - Steuerwerke
- Systemschnittstellen



# COMRADE Zwischenstand

- Auslagerung von Berechnungen auf RCU
  - Bis zu 80% der Instruktionen des Programmes
- Verschmelzen von CUs
  - Bis zu 99.9% weniger Rekonfigurationen
- Hardware-orientierte Optimierung
  - Nur 50-70% der Variablenbits relevant
- Hauptaufgaben derzeit
  - Hardware-Integrationsschritt
  - Bei Tests auftauchende Fehler beheben
- Dann endlich
  - Echte Laufzeitmessungen
- Viele Verbesserungsmöglichkeiten absehbar

# Zusammenfassung

- Motivation des rekonfigurierbaren Rechnens
  - Alternative zu Standardprozessoren
  - Anwendungsbeispiele
  - Hohes Potential (Rechenleistung, Energie, ...)
- Architekturen: Chip-/Systemebene
  - Adaptive Rechensysteme
- Programmierung: Compilierung von C
  - Ursprungssystem Nimble
  - Neuentwicklung COMRADE

➡ Hohes Potential:

Entwicklung steht noch am Anfang



# Ende