
Hardware-Entwurf von parallelen Systemen

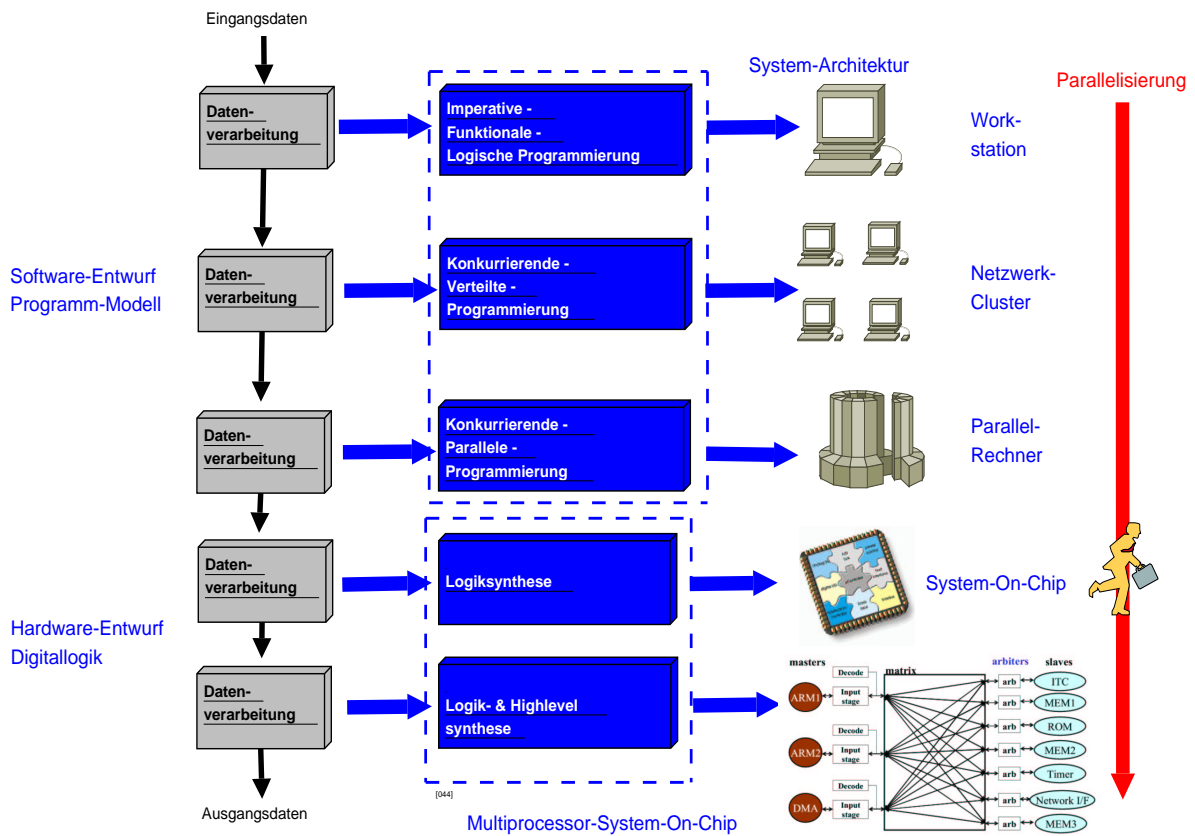
Logik- & High-Level-Synthese

Dr. Stefan Bosse

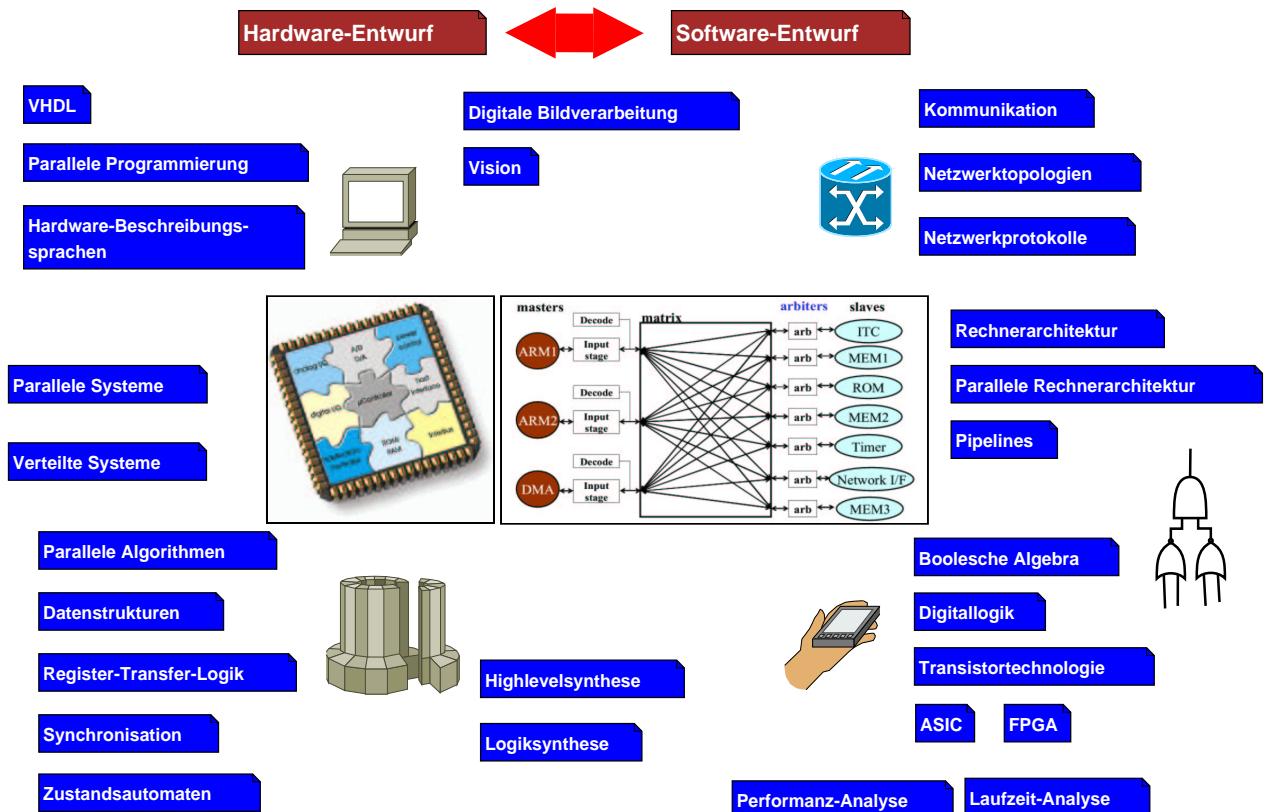
Universität Bremen

Vorlesung WS 2006 (V061023)

Überblick (I)



Überblick (II)



Parallele Systeme

Anwendungsgebiete für parallele Systeme

- ▶ Digitale Bildverarbeitung und automatische Bildinterpretation (Vision), wo sowohl hoher Datendurchsatz als auch niedrige Bearbeitungszeit gefordert werden,
- ▶ Komplexe Steuerungssysteme mit großer Anzahl von Freiheitsgraden, wie z.B. Positionssteuerung von Robotergelenken,
- ▶ Kommunikation
- ▶ Numerische Probleme wie Lösung von komplexen Differentialgleichungen für physikalische Probleme (Navier-Stokes-Gleichungen der Fluidodynamik), klimatische Berechnungen, Astronomie usw.

Motivation für parallele Systeme

- ▶ Steigerung der Energieeffizienz von mikroelektronischen Systemen:
 - ↳ komplexe generische Einprozessoranlagen besitzen ungünstige Energieeffizienz, aber:
 - ↳ Zerlegung eines sequenziellen Programmes in parallele Tasks
 - ↳ Ausführung parallel (konkurrierend) auf mehreren einfachen Prozessoren
 - ↳ Kann zu verbesserter Energieeffizienz des Gesamtsystems führen!
- ▶ Skalierung von parallelen Rechneranlagen auf Digitallogiksysteme für anwendungsspezifische Lösungen kann deutliche Reduktion der Hardware-Komplexität und der elektrischen Leistungsaufnahme im Vergleich zu generischen (Multi-) Prozessorsystemen bedeuten!

Rechenkomplexität bei Vision-Systemen (I)

- ▶ Digitale Bildverarbeitung und Bildinterpretation umfassen Algorithmen aus den Bereichen: **Signalverarbeitung** ▶ **Mathematische Verfahren** ▶ **Graphen** ▶ **KI**
- ▶ Diese unterschiedlichen Methoden und Algorithmen stellen unterschiedliche Anforderungen an die verwendete Verarbeitungsarchitektur! Generische Systeme können nicht alle Teilbereiche effizient (wenn überhaupt) bearbeiten.
- ▶ Aufgrund des dreidimensionalen Datenraums ($X \otimes Y \otimes T$) benötigen diese Algorithmen und Programme große Anzahl von Rechenoperationen OP und Rechenleistung $OPS = OP/sec$.

Beispiel: Bildmatrix $\Phi(x,y)$ mit 500×500 Pixel, je Pixel 24 Bit, Bildrate 10 Bilder/s

↳ Berechnung der mittleren Intensität eines Bildes:

$$I(\Phi) = \frac{1}{N} \sum_{i=1}^{500} \sum_{j=1}^{500} \Phi(i, j) \quad (1)$$

- ↳ Benötigt $500 \times 500 \times 10 \times 2$ (Speicherzugriff \oplus Addition) = 5 MIPS (Einheitsoperationen) bei 24 Bit Datenbreite und einer Datentransferrate von 7.5 MBytes/s.
- ▶ Komplexere Algorithmen wie
 - ◆ Bildtransformationen,
 - ◆ Objekterkennung und Bewegungserkennung usw.

erfordern Rechenoperationen im Bereich von 100-10000 GIPS!

Rechenkomplexität bei Vision-Systemen (II)

- ▶ Komplexität und Ressourcenaufwand werden bei Vision-Systemen zusätzlich erhöht durch:
 - ◆ Wechselwirkung zwischen verschiedenen (überlagerten) Algorithmen und Bearbeitungsverfahren,
 - ◆ Ein- und Ausgabe von Daten (hohe Datenmengen!),
 - ◆ Management von Systemressourcen, aber auch den Datenakquisitionsgeräten (Kamera),
 - ◆ Fehler- und Ausfalltoleranz, z.B. in sicherheitskritischen Bereichen.
- ▶ Diese zusätzlichen Anforderungen addieren sich zu der Rechenleistung, die für die eigentlichen Bildalgorithmen benötigt werden.
Z.B kann ein sehr schnelles Bildverarbeitungssystem nicht effizient genutzt werden, wenn die Ein- und Ausgabeeinheiten zu geringe Datentransferraten besitzen ▶ "Flaschenhals".
- ▶ Serielle Datenverarbeitung mit generischen Einprozessor-Systemen kann nicht die erforderliche Rechenleistung liefern, um eine Datenmenge D pro Zeiteinheit $[D/t]$ zu verarbeiten.

Grenzen :

- ◆ Zentrales Speicherkonzept und Limitierung durch Busverbindungen,
- ◆ Elektrische Verlustleistung \sim Rechenleistung,
- ◆ Hohe Komplexität von generischen Hochleistungsprozessoren (Design-Fehler!)

Rechenkomplexität bei Vision-Systemen (III)

- ▶ Parallele Datenverarbeitung bietet sich als Verarbeitungsarchitektur an.
- ▶ Viele Algorithmen der digitalen Bildverarbeitung besitzen inherente (implizite) Parallelität.
Beispiel: Mittelwertbildung
 - ↳ Partitionierung der Bildmatrix Φ in $N \times M$ Submatrizen φ_{nm}
 - ↳ Nebenläufige \equiv parallele Berechnung der einzelnen Mittelwerte $I_{nm}(\varphi_{nm})$ mit U unabhängigen aber gekoppelten Verarbeitungseinheiten (Prozessoren \oplus Speicher).
 - ↳ Im Idealfall reduziert sich die Bearbeitungszeit auf T/U .
- ▶ Die Möglichkeit und Granularität der Parallelisierung hängt von der Abstraktionsebene des Algorithmus ab:

Lowlevel-Algorithmen auf Bildebene (Digitale Bildverarbeitung)

Räumliche Zerlegung der Bilder und unabhängige Verarbeitung der Teilbilder, keine bis schwache Wechselwirkung zwischen den einzelnen Tasks (parallele Bearbeitungsabläufe)

Highlevel-Algorithmen auf Objektebene (Eigenschaftsauswertung)

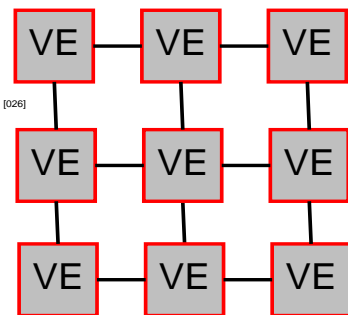
Partitionierung i.A. möglich, aber schwache bis starke Wechselwirkung zwischen den einzelnen Taks ▶ U.A. Datenbanksysteme und Suchalgorithmen!

- ▶ Eine Multiprozessor-Architektur muß in der Lage sein, alle Ebenen der Vision-Algorithmik **effizient** auszuführen.

Rechnerarchitektur für Vision-Systeme (I)

Über Maschennetz verbundene Prozessoren/Rechner

- ▶ Reguläre zweidimensionale Verbindungsstruktur mit großer Anzahl von Verarbeitungseinheiten (VE) quadratisch angeordnet.
 - ↳ Häufig ist jede VE mit seinen direkten Nachbarn (4) verknüpft.
 - ↳ Jede VE besitzt i.A. lokalen Speicher.
- ▶ Zweidimensionale Datenstrukturen wie Bilder lassen sich einfach auf diese Struktur abbilden.
- ▶ Maximale Parallelität nur erreichbar bei Algorithmen und Operationen auf Pixel-Ebene, d.h. bei kurzer "Berechnungslänge".
- ▶ Kommunikation in Netzen ist über weite Distanzen teuer (Zeit).
- ▶ Algorithmen wie Gruppierung oder Mustererkennung erfordern langreichweitige Berechnungen. Führt zu hohem Kommunikationsüberhang zwischen den VEs.

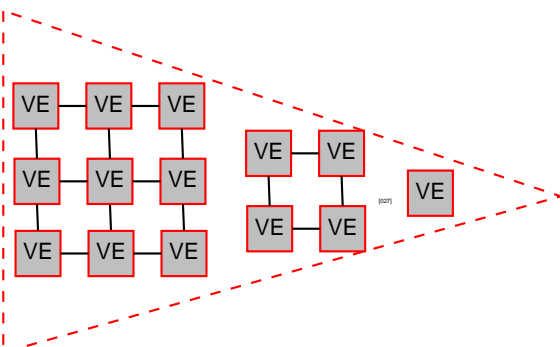


- Bei Maschennetz-Architekturen führen häufig alle VEs gleiche Instruktion durch - ungeeignetes Programmiermodell für Highlevel-Algorithmen (SIMD).
- Grenzwert: jede VE verarbeitet nur 1-Bit Daten, $N_{VE} \gg 1000$
- Lässt sich Datenmatrix nicht vollständig auf Netzstruktur partitionieren, ist Effizienz ebenfalls reduziert.

Rechnerarchitektur für Vision-Systeme (II)

Pyramidennetz

- ▶ Irreguläre dreidimensionale Verbindungsstruktur mit großer Anzahl von Verarbeitungseinheiten (VE), mehrstufig in (quadratischen) Ebenen angeordnet.
 - ↳ Jede VE ist mit seinen (4) direkten Nachbarn auf gleicher Ebene,
 - ↳ jeweils mit (4) VEs der unteren Ebene und (1) der oberen Ebene verknüpft ($\Sigma=9$).
 - ↳ Eine Pyramide besitzt $1/2 \log(N+1)$ Ebenen bei N VEs.
- ▶ Gut geeignet für Divide-and-Conquer-Verfahren, wo ein Problem immer weiter mit einem Teilungsfaktor 2 zerlegt wird.
- ▶ Nachteil: hoher Verbindungsaufwand (Netzwerk).
- ▶ Auf verschiedenen Ebenen können Bilder mit unterschiedlicher Auflösung/Matrixgröße parallel verarbeitet werden.



- Eine Pyramidenarchitektur ist geeignet, alle Ebenen der Algorithmik im Vision-Bereich effizient bearbeiten zu können.
- Sowohl SIMD- als auch MIMD- (verschiedene VEs bearbeiten unterschiedliche Instruktionen) Betrieb ist möglich.

Parallelität (in Vision-Systemen)

- ▶ Unterteilung in **räumliche und zeitliche Parallelität**
- ▶ Parallele Datenverarbeitung bedeutet Partitionierung eines seriellen Programms in eine Vielzahl von Subprogrammen oder **Tasks**.
- ▶ Weitere Unterteilung beider Dimensionen in Abhängigkeit von:
 - ◆ Art der Tasks/Algorithmen
 - ◆ Ausführungsmodell der Tasks und verwendete Rechnerarchitektur
 - ◆ Art und Umfang der Wechselwirkung zwischen Tasks
 - ◆ Kontroll- und Datenfluß eines Tasks

Räumliche Parallelität

Die Datenmenge D kann in Teilmengen $d_i \supset D$ zerlegt werden. Die minimal erreichbare Größe der Teilmenge gibt Granularität bei der Parallelisierung wieder.

Die Datenmenge D wird durch eine Verarbeitungsstufe in eine neue Datenmenge D' transformiert, die dann von nachfolgenden Verarbeitungsstufen weiter verarbeitet wird.

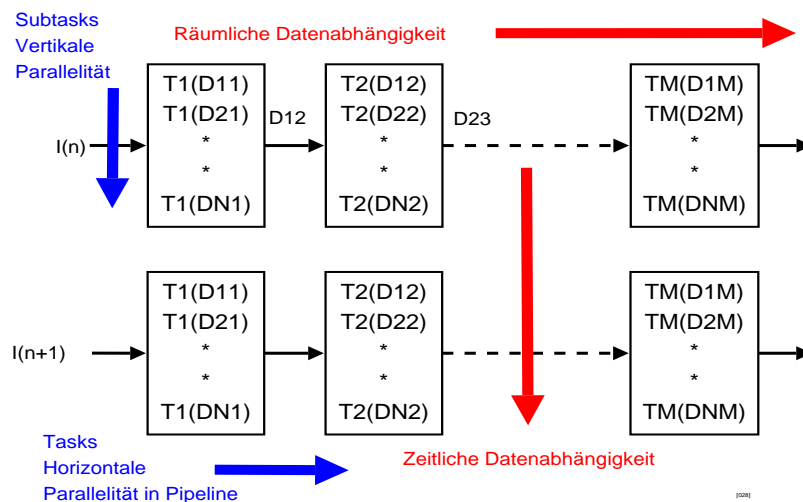
Beispiel : $D = \text{Bild} \rightarrow \text{Glättung} \rightarrow D' \rightarrow \text{Objekterkennung} \rightarrow D''$

Zeitliche Parallelität

Zeitliche Parallelität ist vorhanden, wenn eine Folge von gleichartigen Datenmengen $D(n)$ repetierend mit dem gleichen Algorithmus verarbeitet werden \rightarrow Pipeline-Verfahren.

Berechnungsmodell für Vision-Systeme :: Datenabhängigkeit

- ▶ Räumliche und zeitliche Parallelität führt zu räumlicher und zeitlicher Datenabhängigkeit.
- ▶ Räumliche Datenabhängigkeit findet auf Intra- und Intertaskenebene statt.
 - Intrataskebene** : Subtasks tauschen Daten aus \rightarrow Sequenzielle Ausführung.
 - Beispiel: $ST1: a=x+y; ST2: b=a+1; \rightarrow b(a) \rightarrow ST2(ST1)$
 - Intertaskenebene** : Übertragung von Daten zwischen Tasks in einer Pipeline.
- ▶ Zeitliche Datenabhängigkeit: Ergebnisse aus der Vergangenheit gehen in aktuelle Datenberechnung ein.
 - Beispiel: Bewegungserkennung aus einer Bild-Sequenz.



Berechnungsmodell für Vision-Systeme :: Rechenzeit

➤ Die Rechenzeit t_{tot} für die Ausführung einer Pipeline $T_1 \dots T_M$ enthält:

1. Zeit zum Einlesen der Daten $I(n)$,
2. Zeit für die Ausgabe der Daten und Ergebnisse,
3. Summe aller Ausführungszeiten der Tasks in der Pipeline, die sich aus Rechen- und Kommunikationszeiten zusammensetzen.

$$t_{tot} = \sum_{i=1}^m \tau_i + \sum_{i=1}^{m-1} t_d(D_{i,i+1}) + t_{in} + t_{out} \quad (2)$$

mit

$$\tau_i = \text{MAX} : t_{cp}(T_i(d_j)) \forall 1 \leq j \leq n_i + t_{comm}(T_i) \quad (3)$$

als die Zeit die benötigt wird, einen Task i unter Berücksichtigung von Datenabhängigkeiten der n_i Subtasks $T(d_j)$ zu bearbeiten: längste Bearbeitungszeit eines Subtasks bestimmt Bearbeitungszeit des Tasks!

T_i ist der i -te Task in der horizontalen Pipeline, d_i die Datenabhängigkeit, t_{cp} ist die Rechenzeit, t_{comm} die Intertaskkommunikationszeit, t_{in} und t_{out} die Zeit zum Datentransfer in und aus der Pipeline, und t_d die Datentransferzeit zwischen zwei Tasks.

➤ In Vision-Systemen sind die ersten Tasks i.A. low- und midlevel Algorithmen, und die letzten Tasks highlevel Algorithmen, die auf den Daten der unteren Ebenen aufbauen. Die einzelnen Datenströme D können daher von unterschiedlicher Größe und Struktur sein.

Klassifikation von Vision-Algorithmen :: Datenabhängigkeit

Lokal, statisch

Ausgangsdaten (\equiv Ergebnisse) hängen nur von eng begrenzter kurzreichweitiger Region der Eingangsdaten ab. Die Größe der Eingangsdaten-Region ist unveränderlich (statisch). ➤ Kommunikationsbedarf ist gering.

Lokal, dynamisch

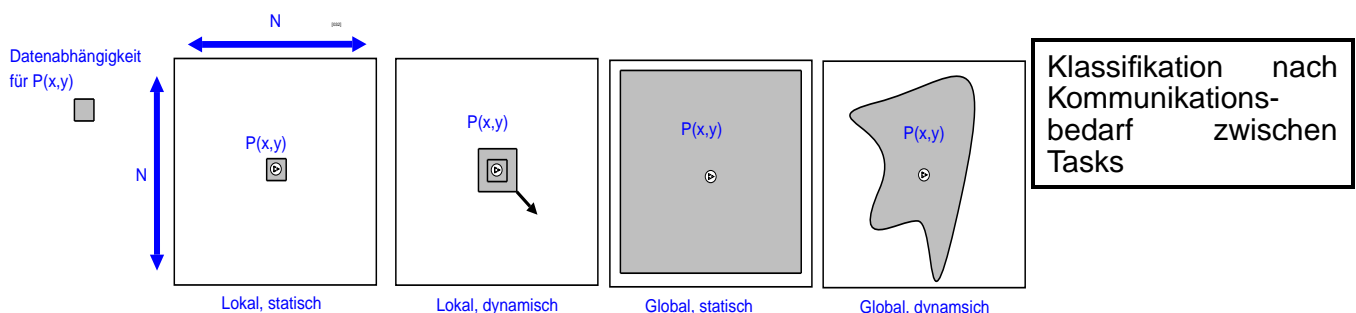
Die Größe der Eingangsdaten-Region ist parametrisiert und veränderlich (dynamisch). Z.B. bei mathematischer Faltung von Bildmatrizen ändert sich Größe der Region.

Global, statisch

Die Ausgangsdaten hängen gänzlich vom gesamter Eingangsdatenmenge ab. Abhängigkeit hängt nur von der Größe des Bildes, aber nicht von dessen Inhalt ab. Z.B. Fouriertransformation oder Histogramm-Funktionen. ➤ Kommunikationsbedarf ist groß.

Global, dynamisch

Ausgangsdaten hängen von variierenden Ausschnittsregion der Eingangsdaten ab. Die Berechnung ist vollständig datenabhängig. Z.B. Hough-Transformation.



Eigenschaften von Parallelarchitekturen für Vision-Systeme (I)

Rekonfigurierbarkeit

Unterschiedliche Ebenen der Algorithmik erfordern unterschiedliche Rechnerarchitekturen. Sowohl SI and MI-Ausführung muß dynamisch (re-)konfigurierbar möglich sein ➤ Dynamische Anpassung der Architektur an Algorithmen.

Flexible Kommunikation

Fein granulierte und Hochgeschwindigkeitskommunikation ist für effiziente Ausführung der Algorithmen und Tasks erforderlich. Kommunikation zwischen Tasks und den unterschiedlichen Verarbeitungsebenen.

Ressourcen Allokation und Partitionierbarkeit

Gesamtsystem muß aus unabhängigen Teilsystemen bestehen, um effiziente Implementierung der (unterschiedlichen) Tasks zu ermöglichen. Ressourcen (Prozessoren, Speicher) müssen dynamisch und fein granuliert an die Tasks/Prozesse gebunden werden können.

Lastbalanzierung und Task Scheduling

- Besonders für highlevel (=komplexe) Algorithmen mit starker Datenabhängigkeit sind Lastverteilung und zeitliches Taskscheduling von großer Bedeutung, aber nicht trivial.
- In lowlevel Algorithmen mit geringer Datenabhängigkeit erreicht man Lastbalanzierung meistens durch Datenpartitionierung.

Eigenschaften von Parallelarchitekturen für Vision-Systeme (II)

Unabhängigkeit von Topologie- und Datengröße

Durch die hohe Komplexität von Vision-Algorithmen muß die Architektur unabhängig von detaillierten Annahmen bezüglich Datenstruktur (z.B. Matrixgrößen oder Datenbreite) und bestimmter Algorithmik sein!

- Betrifft auch dynamisch konfigurierbare und skalierbare Kommunikationsstrukturen.

Fehlertoleranz

Bei der Bearbeitung von komplexen Aufgaben mit komplexen Systemen spielt Fehlertoleranz eine wichtige Rolle. Ein Ausfall einer einzelnen Komponente darf nicht zum Ausfall des gesamten Systems führen. Statische Redundanz ist aber teuer (Ressourcenbedarf)!

Ein- und Ausgabe (IO)

Neben geringer Bearbeitungs- und Rechenzeit ist der Datentransfer der Eingangs- und Ausgangsdaten (Ergebnisse) gleichbedeutend. Performanz und Architektur von IO ist ein Kernbestandteil paralleler Systeme!

NETRA: Parallelarchitektur für Visison (I)

Systemarchitektur

NETRA ist ein rekursiv definierter und hierarchischer Multiprozessor speziell für Vision-Systeme entwickelt.

DSP (Distributing-and-Scheduling-Processors)

Taskverteilung und Kontrolleinheiten. Erlauben dynamische Partitionierung (zeitlich- wie räumlich) von Programmcode auf Verarbeitungseinheiten.

PE (Processing Element)

Verrarbeitungseinheiten. Organisiert in Clustern der Größe 16-64 PEs. Bis 150 Cluster können gebildet werden.

Jeder PE ist ein generischer High-Performance-Prozessor mit schneller FPU (Floating-Point-Unit).

C (Cluster)

Bindung von PEs zu einer Einheit.

Jeder Cluster teilt sich einen gemeinsamen Datenspeicher. Dieser Speicher ist in einer Pipeline angeordnet.

Alle PEs können über einen Kreuzmatrixschalter C mit den Datenspeichern M verbunden werden.

Die DSPs sind ebenfalls über C mit den PEs verknüpfbar.

NETRA: Parallelarchitektur für Visison (II)

IC (Interconnect)

Globale Verbindungsmatrix die die Cluster mit Speichermodulen M verbindet.

➤ Aufgebaut als vollständiges unidirektionales Verbindungsnetz $M \times N$ mit einer Kreuzschaltermatrix.

➤ Vollständiges Verbindungsnetz benötigt keine Synchronisation (Mutex/Arbiter).

M

Speichermodul welches parallelen und geteilten Zugriff unterstützt.

Die Speichermodule M sind in einer Pipeline angeordnet.

Die Zugriffszeit eines Moduls sei T Taktzyklen. Jede Pipeline besteht aus T Speichermodulen.

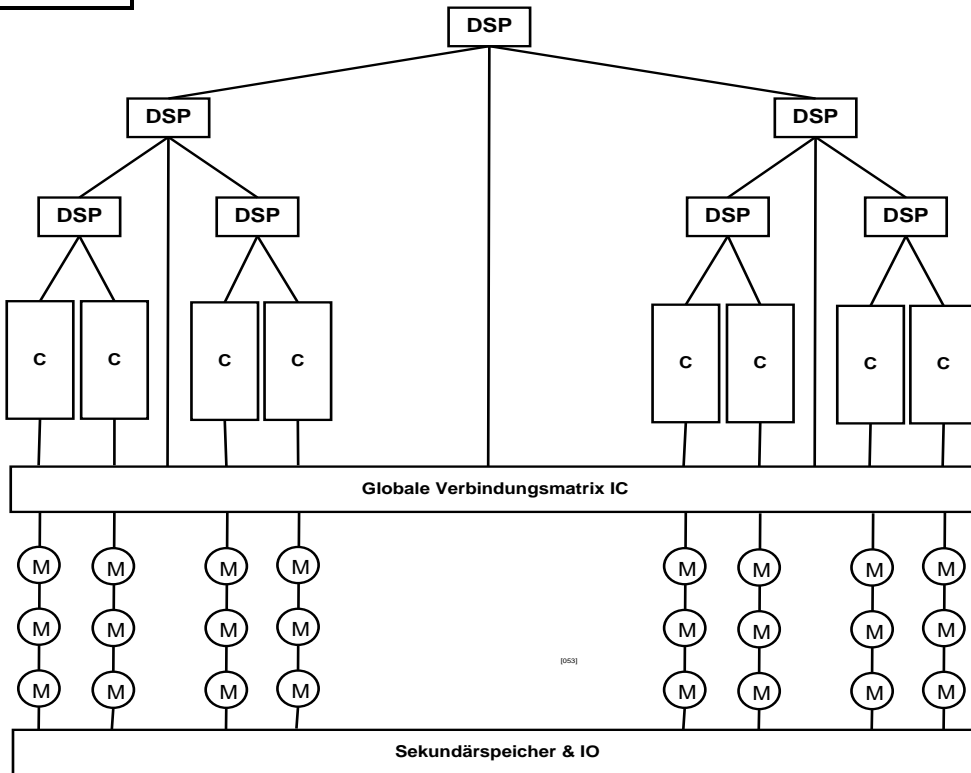
➤ Die Speicher-Pipeline kann daher einen Speicherzugriff pro Taktzyklus durchführen!

Weitere Eigenschaften

- Cluster können in SIMD, systolischer oder MIMD-Betriebsart konfiguriert werden.
In SIMD-Betriebsart führen alle PEs eines Clusters gleiche Operation durch.
In systolischer Betriebsart führen die PEs wiederholt Operationen auf einem Datenstrom durch.
- Der globale Speicher wird in Blöcke unterteilt: Daten- und Programmblöcke. Die Blockgröße ist variabel. Eine Anzahl von Blöcken bilden ein Programm oder eine Bildmatrix. Der globale Speicher besitzt Daten-Queues für die Zwischenpufferung.

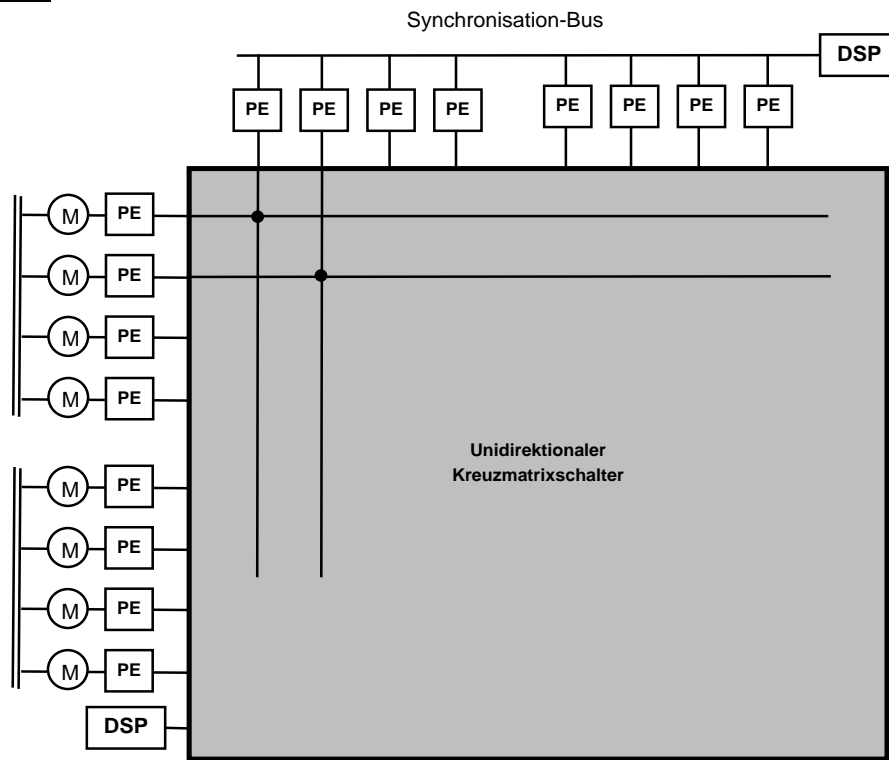
NETRA: Parallelarchitektur für Visison (III)

Systemarchitektur



NETRA: Parallelarchitektur für Visison (IV)

Prozessor-Cluster



System-On-Chip

SOC

System-On-Chip Hardware vereinigt auf einem einzigen Elektronikträger (Chip) ein (partiell) vollständiges System, welches aus einer Vielzahl von Subsystemen zusammengesetzt ist:

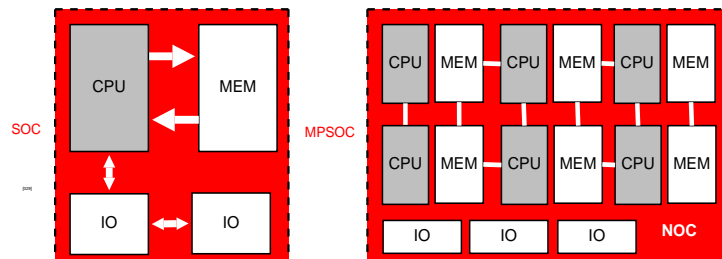
1. Verarbeitungseinheit: Mikroprozessorkern
2. Speicher: Programm-, Daten- Cache-Speicher
3. Peripherie: Schnittstellen
4. Spezielle Recheneinheiten: Floating-Point-Unit (FPU), Graphic-Processing-Unit (GPU)
5. Verbindungssysteme: Netzwerkstrukturen und Komponenten

MPSOC

Multiprozessor-Systeme auf einem einzigen Chip.

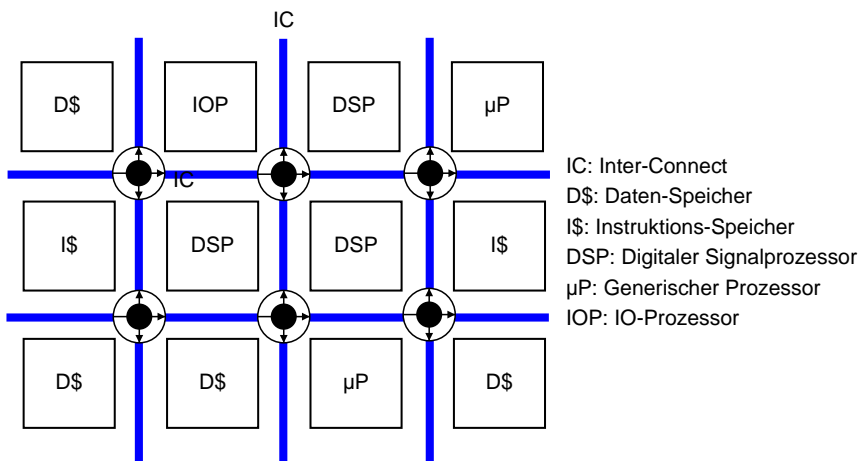
NOC

Netzwerk-Systeme als Verbindungsstruktur für die Kommunikation zwischen Subsystemen eines SOC.



System-On-Chip :: MPSOC

- ▶ MPSOC-Architekturen können homogen oder heterogen aufgebaut sein, d.h.
 - ◆ generisch aus Matrix mit gleichen Verarbeitungseinheiten oder
 - ◆ anwendungsspezifisch aus verschiedenen generischen und speziellen Systemkomponenten und Verarbeitungseinheiten zusammengesetzt.
- ▶ Bei MPSOC-Architekturen (≡ Paralleles System) nimmt die Verbindungsstruktur und Kommunikation eine zentrale Rolle ein:
 1. Performanz des Gesamtsystems
 2. Hardware-Ressourcen-Aufwand

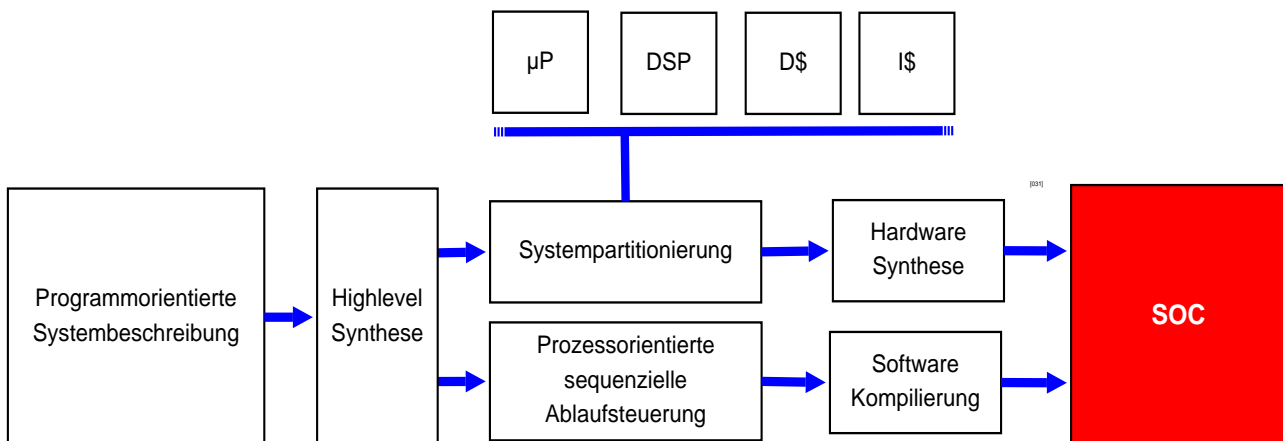


System-On-Chip :: Design

- ▶ Ein aktueller Low-Cost Fertigungsprozeß besitzt eine Transistorabmessung von ca. 150nm. Man erzielt Packungsdichten in der Größenordnung von 100k Logikgattern pro mm². Ein 50 mm² Chip kann ein System mit 5M Gattern implementieren.
- ▶ Synthese-Programme, die aus einer Highlevel-Systembeschreibung oder mittels Hardware-Beschreibungssprachen Logikgatter-Netzlisten ableiten, waren in der Vergangenheit auf 100k Gatter/Systemblock limitiert, aktuell sind zusammenhängende Blöcke bis zu einer Größe von einigen Mega Gattern beherrschbar.
 - ↳ Aber: die Logiksynthese ist mittlerweile limitierender Faktor im Hardware-Entwurf bei komplexen monolithischen Systemblöcken!
- ▶ Die Design-Komplexität eines Logikblockes wächst stärker als seine Anzahl von Gattern! Die System-Komplexität wächst schneller als die Anzahl der System-Blöcke!
 - ↳ Verifikation wird durch ebenfalls gestiegene Komplexität erschwert.
- ▶ Durch gestiegene System-Komplexität wird ein kombinierter **Hardware-Software-Ansatz** bei der Entwicklung notwendig:
 - ↳ Beschreibung des Systems durch abstrakte programmorientierte Ansätze
 - ↳ Synthese von Hardware-Modulen wie CPUs oder Kommunikationsstrukturen aus Software und Erzeugung von Programmen.
- ▶ Eingebettete Systeme (SOC+) besitzen großen Software-Anteil.
- ▶ Kommunikation und Protokolle werden immer komplexer. Höhere Rechenleistung ist die Folge mit klassischen μ P-Systemen.

System-On-Chip :: Hardware-Software-Co-Design

- Traditionell: Zuerst wird Hardware spezifiziert und synthetisiert mittels Hardware-Beschreibungssprachen (VHDL, Verilog-HDL), dann Software (C).
- HW-SW-Co-Design: Die Spezifikation der Hardware wird zeitgleich mit bzw. aus dem Software-Design abgeleitet.
- Ausgangspunkt: Highlevel Systembeschreibungssprachen \equiv Programmiersprache (SystemC, CONPRO)
 - ◆ Programmorientiert mit sequenzieller Ablaufsteuerung und Zustandsautomaten
 - ◆ Ableitung von Hardware-Beschreibung aus Daten- und Kontrollflußspezifikation
 - ◆ Parallelisierung durch Multi-Prozeß-Modell und Pipeline-Verfahren



Traditionelles SOC-Design

- ▶ Konventionelles Architekturmodell abgeleitet aus weit verbreiteter Rechnerarchitektur:
 - Prozessormodell → RISC basierend, konfigurierbarer oder erweiterbarer Kern, Datenbreite wählbar oder fest vorgegeben, Speicher, Peripherie, anwendungsspezifische Logikblöcke (RTL).
- ▶ Kommunikation zwischen einer Vielzahl von SOC-Chips müssen realisiert werden (PCB): aus ökonomischen Gründen Bussysteme mit kleinst möglicher Datenwortbreite!
- ▶ Vorteile von komplexen SOC-Lösungen:
 1. reduzierter Platzbedarf, reduzierte Kosten,
 2. erhöhte Performanz z.B. durch erhöhte Datenbusbreiten, bei erhöhter Zuverlässigkeit,
 3. und i.A. reduzierter elektrischer Leistungsbedarf.
- ▶ Nachteil: gestiegene Komplexität macht Verifikation schwierig!

Erweiterbare Prozessoren

Die Architektur des Prozessorkerns ist vorgegeben. Befehlssatz und Befehlsformate sind vorgegeben. Der Befehlssatz und Subkomponenten des Prozessors können erweitert werden, z.B. durch neue komplexere Spezialbefehle ($l\oplus: a+b/2$).

Vorteil: Compiler existieren, Nachteil: nur bedingt an Anwendung optimal anpassbar.

Konfigurierbare Prozessoren

Der Befehlssatz, die Architektur und Datenbreiten sind frei wählbar.

Vorteil: optimale Anpassung an Anwendung, Nachteil: keine Compiler verfügbar.

Digitallogik (I)

Logische Werte

Boolesche Logikwerte: $\{0,1\}$

Technologische Logikwerte: $\{0,1,L,H,Z,X\}$

Logische Funktionen

UND: $f(a,b)=a \wedge b = a \bullet b$

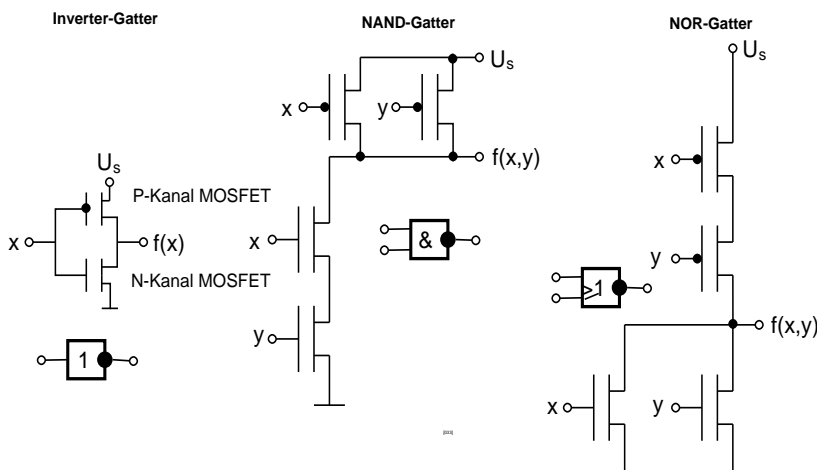
ODER: $f(a,b)=a \vee b = a + b$

Negation: $f(a)=\neg a$

Technologische Umsetzung

Gatter basierend auf CMOS-Transistortechnologie

0,1	Starke logische Werte.
L,H	Schwache logische Werte. Dürfen überlagert werden.
Z	Hochimpedanz-Wert (Bustreiber)
X	Don't Care.



Digitallogik (II)

Boolesche Variablen

V: Boolesche Logikwerte: {0,1}

Boolesche Funktionen

Abbildung von N booleschen Variablen (N-dimensionaler Vektor) auf M boolesche Ergebniswerte (M-dimensionaler Vektor) ➤ Skalare boolesche Funktion (M=1) ➤

f: $a_1 \times a_2 \times a_3 \times \dots \rightarrow y$

Normalformen

Ableitung aus Wahrheits-/Funktionstabellen mit 2^N Funktionswerten f und $2^N \times N$ Eingangswerten:

Disjunktive Normalform

Jeder Teilterm der DNF besteht aus einem Produkt der Eingangsvariablen. Die Teilterme werden summiert.

$$f(a_1, a_2, \dots, a_N) = (x_1^1 \cdot x_2^1 \cdot \dots \cdot x_N^1) + (x_1^2 \cdot x_2^2 \cdot \dots \cdot x_N^2) + \dots \quad (4)$$

Konjunktive Normalform

Jeder Teilterm der KNF besteht aus einer Summe der Eingangsvariablen. Die Teilterme bilden ein Produkt.

$$f(a_1, a_2, \dots, a_N) = (x_1^1 + x_2^1 + \dots + x_N^1) \cdot (x_1^2 + x_2^2 + \dots + x_N^2) \cdot \dots \quad (5)$$

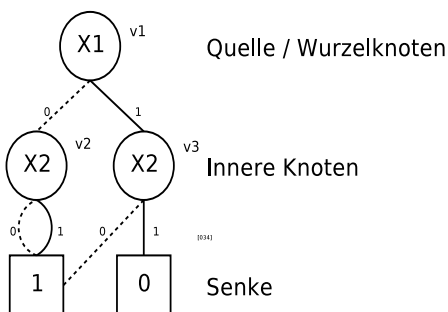
Digitallogik (III) :: Systematische Reduktion logischer Funktionen

Ziel der Minimierung: möglichst kleine Anzahl von Logikgatter-Komponenten bei der elektronischen Implementierung bei gleichzeitiger geringer Signallaufzeit, zwei gegenläufige Ziele!

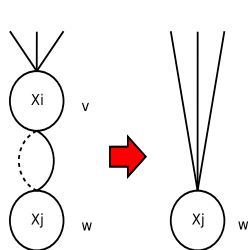
Minimierungsverfahren

1. Minimierung mittels Gesetzen der Booleschen Algebra ➤ nicht systematisch.
2. Karnaugh-Veitch-(KV) Diagramme ➤ Grafische Methode basierend auf zweidimensionalen Diagrammen ➤ beschränkt auf kleine Anzahl von Funktionsvariablen.
3. Quine-McCluskey-Verfahren ➤ systematisch ➤ Basiert auf Minimierung der Funktionsterme ➤ für mittlere Anzahl von Funktionsvariablen geeignet.
4. Binary-Decision-Diagrams (BDD) ➤ Basiert auf azyklischen und gerichteten Graphen ➤ systematisch ➤ häufig in Synthese-Programmen verwendet.

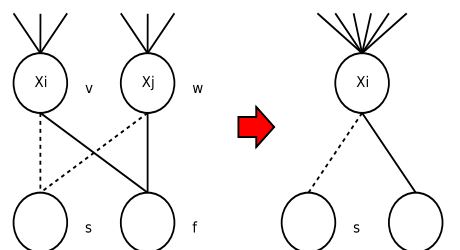
$$f(x_1, x_2) = \overline{x_1} + \overline{x_2}$$



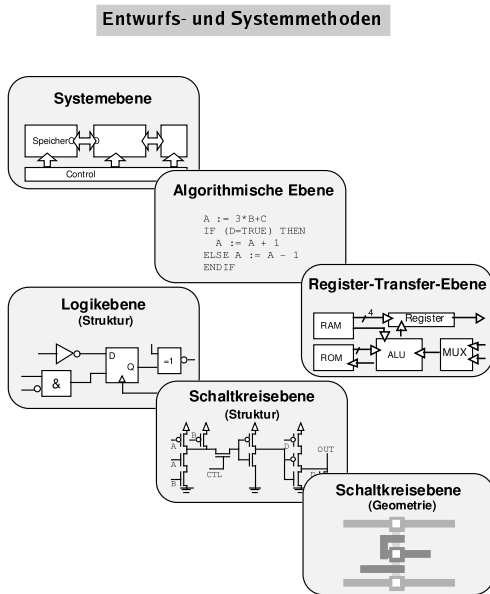
Deletion-Rule



Merging-Rule



Entwurfs- und Systemmethoden



© G. Lehmann/B. Wunder/M. Selz

- ▶ Unterteilung in verschiedene Abstraktions-ebenen.

Systemebene

Partitionierung in Subsysteme

Algorithmische Ebene

Verhaltensbeschreibung, HW-SW-Co Design

Register-Transfer-Ebene

Aufteilung des Entwurfs in Daten- & Kontrollpfade ▶ Zustandsautomaten

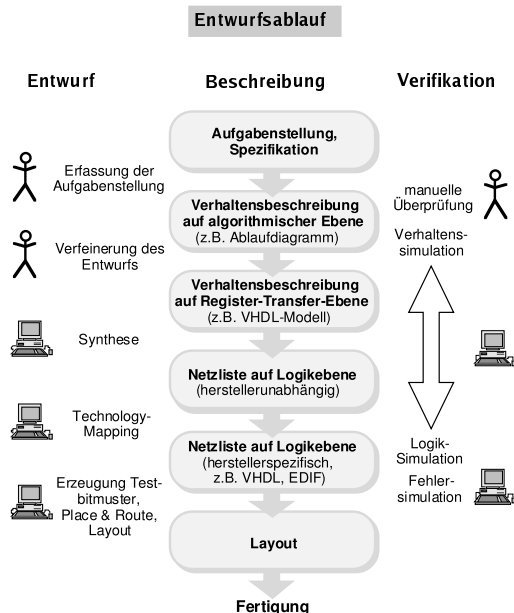
Logikebene

Synthese von booleschen Funktionen und Logikgatter-Netzlisten, Minimierung

Schaltkreisebene

Technologische Umsetzung der Netzlisten.

Entwurfsmethodik



© G. Lehmann/B. Wunder/M. Selz

- ▶ Der Systementwurf wird durch Simulation und Verifikation rückgekoppelt.
- ▶ Dem eigentlichen Syntheseprozess schließt sich das Technologie-Mapping an.
- ▶ Simulation kann auf Highlevel-Ebene mit Verhaltensmodell oder auf Lowlevel-Ebene mit Netzliste (Logik) stattfinden.

Verhaltensbeschreibung

Das Verhalten der Ausgangssignale einer Systemkomponente auf Eingangssignale wird mit einer Hardware-Beschreibungssprache spezifiziert.

Strukturbeschreibung

Schaltnetz bestehend aus Logikgattern und Verbindungen. Keine Spezifikation des Verhaltens.

Register-Transfer-Logik (RTL)

- Übergang von algorithmischer Ebene auf RT-Beschreibung durch Verhaltensbeschreibung VB mit einem Daten- und Kontrollfluß.
- Erzeugung einer RTL bedeutet das Abbilden von daten- und Kontrollfluß in zwei Dimensionen: Zeit und Fläche (Hardware).
- Der Datenteil besteht aus Verarbeitungseinheiten (VE): Addierer, Komperatoren, Multplizierer, Speicherelemente (Register), Datenselektoren (Multiplexer).
- Der Kontrollteil wird mit Zustandsdiagrammen \equiv Zustandsautomaten beschrieben.
- Die Steuer- oder Kontrollnetze aktivieren Speicher, schalten Datenmultiplexer ➤ Steuerung des Datenflusses.
- Die Bedingungsnetze liefern Ergebnisse von Test-Asudrücken für bedingte Verzweigungen im Zustandsdiagramm bzw. im Kontrollteil.

RTL-Implementierung aus VB bedeutet:

Scheduling

Zuordnung von Operationen zu Zeitschritten.

Ressourcen-Allokation

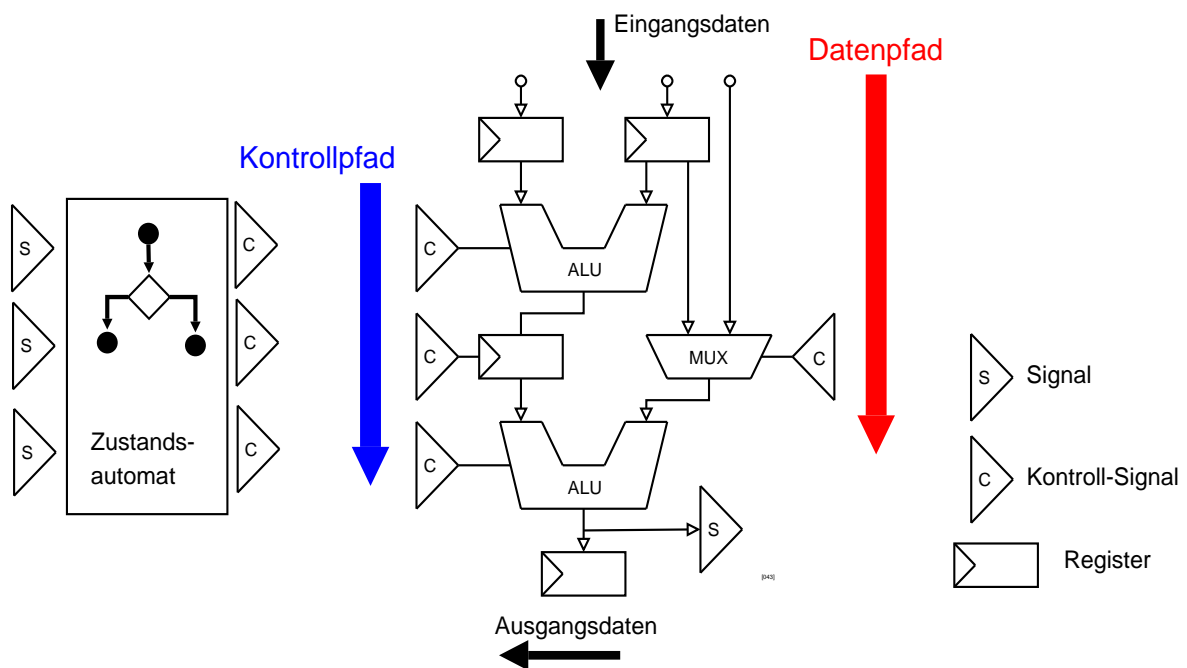
Bestimmung von Typ und Anzahl der Verarbeitungseinheiten, Speicher, Busse...

Ressourcen-Zuweisung

Zuordnung von VEs zu einzelnen Instanzen von Operationen.

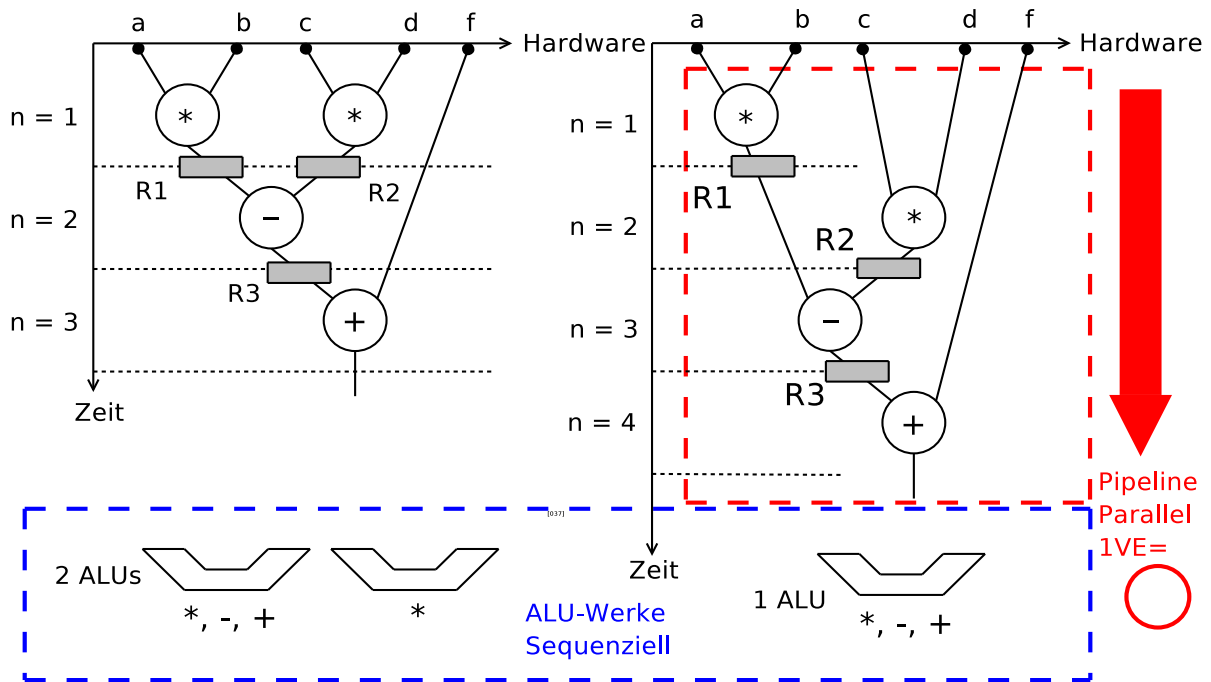
Register-Transfer-Logik (RTL) (II)

➤ Kontroll- und Datenpfade



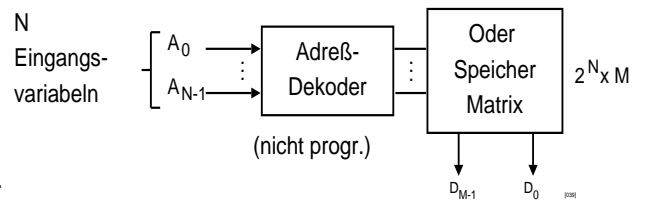
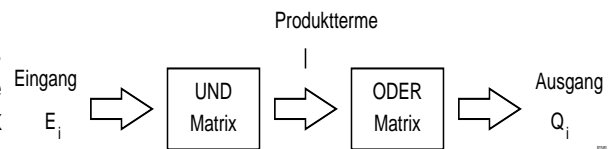
Register-Transfer-Logik (RTL) (III)

▶ Beispiel von RTL für Funktion $f(a,b,c,d,f)=a \cdot b - c \cdot d + f$



Programmierbare Digitallogik (PLD)

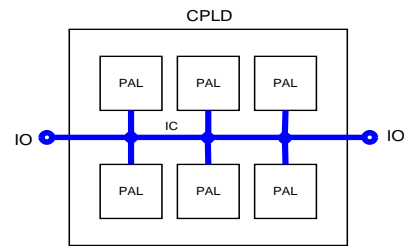
- ▶ Ausgangspunkt für programmierbare Logikbausteine (PLD) ist die disjunktive Normalform DNF zur Beschreibung von kombinatorischer Logik (ohne Register).
- ▶ Eine DNF lässt sich als UND-Verknüpfungsmatrix, die die Produktterme P_{ij} bildet, und einer ODER-Verknüpfungsmatrix darstellen, die die einzelnen Produktterme verbindet.
- ▶ Einfachste technologische Implementierung:
 - ➔ RAM/ROM-Bausteine
 - Adreßdeko­der ➔ UND-Matrix, nicht konfigurierbar, überbestimmt
 - Speicherzellen ➔ ODER-Matrix, konfigurierbar.
- ▶ Verbesserung:
 - ➔ Programmierbare Logikarrays
 - (PLA,PAL,GAL)
 - Konfigurierbare UND- & ODER-Matrizen, unterbestimmt dimensioniert.



Programmierbare Digitallogik :: CPLD & FPGA

CPLD

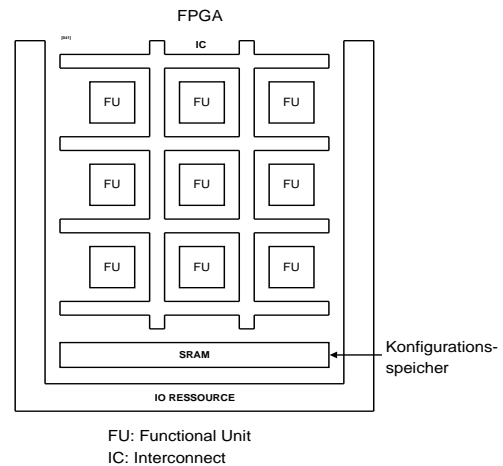
- Partitionierung eines gesamten PLD-Bausteins in Vielzahl kleinerer PAL-Blöcke (UND-ODER-Matrizen)
- Zur Implementierung sequenzieller Logik enthält jeder PAL-Block Registerelemente (FLIP-FLOPs).
 ↳ Registerdichte ist aber niedrig!
- Es muß konfigurierbare Verbindungsleitungen zwischen den PAL-Blöcken geben (Interconnect).



PAL: Und- Odermatrix, vollständig oder unvollständig bestimmt.
 IC: Zwischenverbindung= Interconnect
 IO: Input & Output Leitungen

FPGA

- FPGAs sind in Funktionseinheiten FU unterteilt.
- Die FUs sind mit einem flexiblen kanalbasierten Verbindungssystem verbunden.
- Logikfunktionen werden mit kleinen Lookup-Tables (RAM) realisiert.



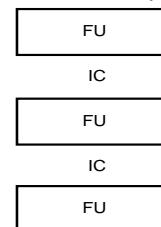
FU: Functional Unit
 IC: Interconnect

Programmierbare Digitallogik :: ASIC

Full-Custom

- Transistor- und Verbindungsebenen sind frei vom Anwender konfigurierbar.
- Setzt Kenntnisse aus der Transistor- und Herstellungstechnologie voraus.
- Sehr fehleranfällig.

Zeilen oder Spaltenstruktur

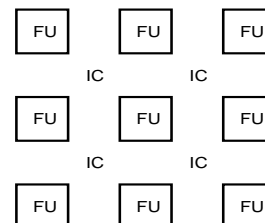


Standardzellen

- Für Grundelemente existieren bereits vorgefertigte Transistorzellen mit Maskenlayout.
- Grundelemente: aus Standardzellenbibliothek ▶
 1. Logikgatter
 2. Registerelemente (FLIP-FLOPs)
 3. RAM/ROM Zellen
 4. Addierer & Multiplizierer
 5. IO-Zellen usw.
- Jede Zelle kann in Äquivalentgatterzahl umgerechnet werden:

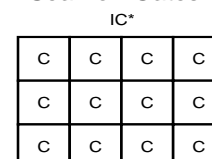
1 ÄQG = 4 Transistoren (NAND/NOR-Zelle)

Blockstruktur

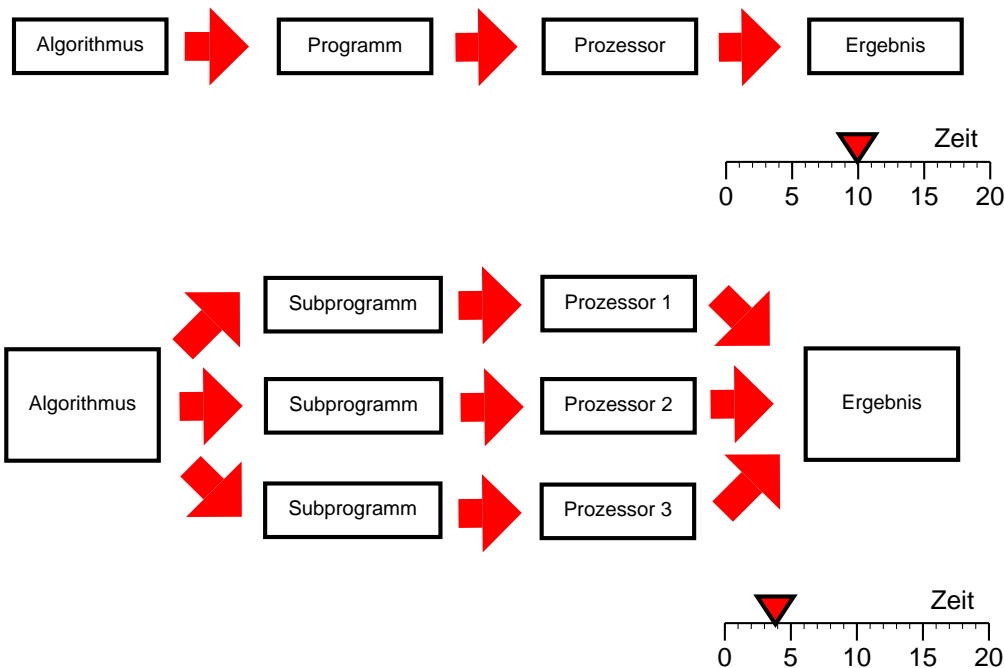


[642]

Sea - of - Gates



Sequenzielle und parallele Informationsverarbeitung



Sequenzielle und parallele Informationsverarbeitung

- ▶ Bei der Datenverarbeitung gibt es zwei Randbedingungen:
 1. Gesamte Rechenzeit ▶ Zeitdimension
 2. Gesamte Ressourcenbelegung ▶ Flächendimension
- ▶ Bei der Parallelverarbeitung wird eine weitere Dimension hinzugefügt:
Anzahl der Verarbeitungseinheiten N
- ▶ Die Nutzung von Parallelität führt zu einem Performanz-Gewinn:

$$\text{Speedup}(N \text{ Prozessoren}) \equiv \frac{\text{Performanz}(N \text{ Prozessoren})}{\text{Performanz}(1 \text{ Prozessor})} \quad (6)$$

- ▶ Der Performanz-Gewinn kann durch Bearbeitungszeit ausgedrückt werden:

$$\text{Speedup}(N \text{ Prozessoren}) = S(N) \equiv \frac{\text{Zeit}(1 \text{ Prozessor})}{\text{Zeit}(N \text{ Prozessoren})} \quad (7)$$

- ▶ Die sog. Skalierung bei der Parallelisierung ist i. A. nicht linear:

$$S(N) < N \quad (8)$$

- ▶ **Kommunikation** ist weitere Randbedingung bei der parallelen Datenverarbeitung.

Verteilung und Zusammenführung

Partitionierung

Zerlegung eines Algorithmus in konkurrierende Tasks

Verteilung

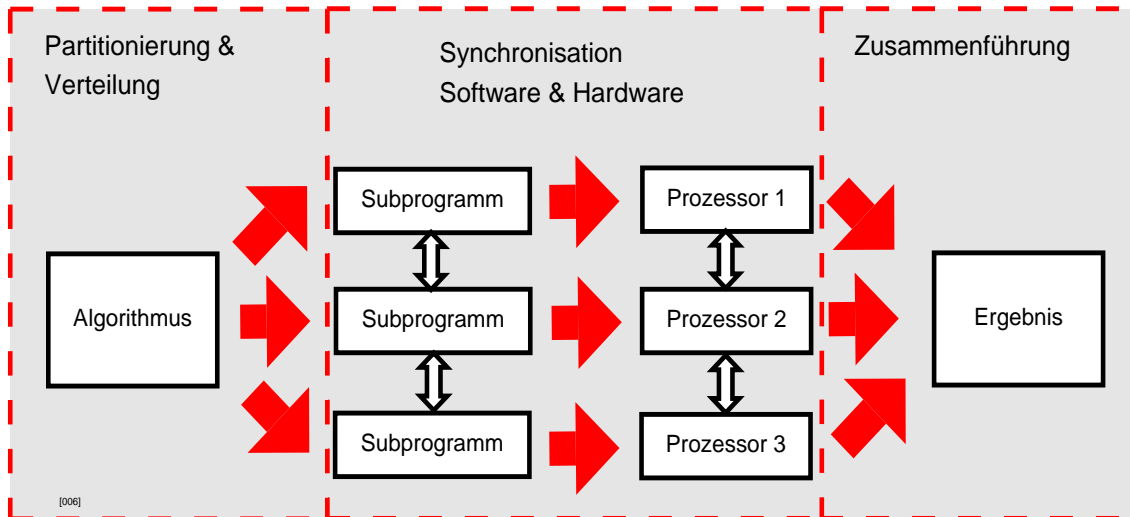
Zuweisung von Tasks an Prozessoren

Synchronisation

Ablaufsynchronisation

Zusammenführung

Teilergebnisse zu Gesamtergebnis zusammenfassen



Paralleler Algorithmus :: Matrixmultiplikation (I)

- ▶ Beispiel für Partitionierung, Verteilung und Zusammenführung.
- ▶ Multiplikation zweier Matrizen $A_{p,q}$ und $B_{q,r}$ $\rightarrow C_{p,r} = A_{p,q} \times B_{q,r}$

$$c_{i,j} = \sum_{k=1}^q a_{i,k} \cdot b_{k,j} \quad (9)$$

- ▶ **Sequenzieller Algorithmus :**

```

Input: Matrix A(p×q), B(q×r)
Output: Matrix C(p×r)
FOR i = 1 to p DO
  FOR j = 1 to r DO
    C[i,j] ← 0;
    FOR k = 1 to q DO
      C[i,j] ← C[i,j] + A[i,k] • B[k,j];
    END FOR k;
  END FOR j;
END FOR i;
  
```

- ▶ **Datenabhängigkeit :** Berechnung eines Wertes $C_{i,j}$ hängt außerhalb der FOR-k-Schleife von keinem anderen Wert $C_{n,m}$ mit $n \neq i$ und $m \neq j$ ab.

Paralleler Algorithmus :: Matrixmultiplikation (II)

- ▶ Partitionierung kann beliebig erfolgen, da die einzelnen Ergebniswerte c nicht voneinander abhängen.
- ▶ Mögliche Partitionierungen der drei For-Schleifen auf N parallel arbeitenden Verarbeitungseinheiten (VE):
 1. Jede VE berechnet einen C_{ij} -Wert. D.h. eine VE führt die FOR-k-Schleife für ein gegebenes i und j durch.
 - ▶ Jede VE benötigt dazu die i -te Zeile von A und die j -te Spalte von B .
 - ▶ Keine weitere Datenabhängigkeit!
 - ▶ Es werden $N=p \cdot r$ VEs benötigt.
 2. Eine VE berechnet eine Ergebnisspalte, d.h. führt die FOR-k und FOR-j-Schleifen durch.
 - ▶ Jede VE benötigt A und eine Spalte von B .
 - ▶ Es werden $N=p$ VEs benötigt.
 3. Eine VE führt eine Multiplikation und Addition innerhalb der FOR-K-Schleife durch.
 - ▶ Jede VE benötigt einen A und B -Wert.
 - ▶ Es werden $N=p \cdot r \cdot q$ VEs benötigt.
 - ▶ Jeweils eine VE für einen gegebenen C_{ij} -Wert führt die Zusammenführung der Zwischenergebnisse der FOR-k-VEs durch.
- ▶ Zusammenführung der Ergebnisdaten in den Fällen 1&2 trivial. Im Fall 3 besteht Zusammenführung im wesentlichen in der Summation der Zwischenergebnisse.

Paralleler Algorithmus :: Matrixmultiplikation (III)

- ▶ Beispiel: $p=q=r=2$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 1*5 + 2*7 & 1*6 + 2*8 \\ 3*5 + 4*7 & 3*6 + 4*8 \end{pmatrix} \quad (10)$$

Fall 1 ▶

VE1: $\{(1,2);(5,7)\}$ ▶ $C_{11}=1*5+2*7$
 VE2: $\{(3,4);(5,7)\}$ ▶ $C_{21}=3*5+4*7$
 VE3: $\{(1,2);(6,8)\}$ ▶ $C_{12}=1*6+2*8$
 VE4: $\{(3,4);(6,8)\}$ ▶ $C_{22}=3*6+4*8$

Fall 2 ▶

VE1: $\{(1,2);(3,4);(5,7)\}$ ▶
 I. $C_{11}=1*5+2*7$
 II. $C_{21}=3*5+4*7$
 VE2: $\{(1,2);(3,4);(6,8)\}$ ▶
 I. $C_{12}=1*6+2*8$
 II. $C_{22}=3*6+4*8$

Fall 3 ▶

VE1: $\{1;5\}$
 VE2: $\{2;7\}$ ▶ $C_{11}=VE1 \oplus VE2$
 VE3: $\{3;5\}$
 VE4: $\{4;7\}$ ▶ $C_{21}=VE3 \oplus VE4$

Fall 3 ▶

VE5: $\{1;6\}$
 VE6: $\{2;8\}$ ▶ $C_{12}=VE5 \oplus VE6$
 VE7: $\{3;6\}$
 VE8: $\{4;8\}$ ▶ $C_{22}=VE7 \oplus VE8$

Paralleler Algorithmus :: Matrixmultiplikation (IVa)

► Synchronisation und Programm:

Fall 1

Aufbau einer Barriere daß alle VEs ihre Berechnung für C_{ij} abgeschlossen haben, z.B. mit einer Semaphore barrier.

```
Process VE[i,j]:
  FOR k = 1 to q DO
    C[i,j] ← C[i,j] + A[i,k] • B[k,j];
  END FOR k;
  SEMA_UP(barrier);

Process MAIN:
  SEMA_INIT(barrier,0);
  FOR ALL (p,r) DO START Process VE[i,j];
  FOR N=1 to (p*r) DO SEMA_DOWN(barrier);
```

Paralleler Algorithmus :: Matrixmultiplikation (IVb)

► Synchronisation und Programm:

Fall 2

Aufbau einer Barriere daß alle VEs ihre Berechnung für C_j abgeschlossen haben.

```
Process VE[j]:
  FOR i = 1 to p DO
    FOR k = 1 to q DO
      C[i,j] ← C[i,j] + A[i,k] • B[k,j];
    END FOR k;
  END FOR i;
  SEMA_UP(barrier);

Process MAIN:
  SEMA_INIT(barrier,0);
  FOR ALL r DO START Process VE[j];
  FOR N=1 to r DO SEMA_DOWN(barrier);
```

Paralleler Algorithmus :: Matrixmultiplikation (IVc)

► Synchronisation und Programm:

Fall 3

Hier müssen allen VEs innerhalb der FOR-k Schleife synchronisiert werden.
Zusätzliche Barriere, daß alle VE-k-Cluster Berechnung abgeschlossen haben.

```
Process VE[i,j,k]:
  t[i,j,k] ← A[i,k] • B[k,j];
  SEMA_UP(barrier_t[i,j]);
  ► Ein VE[i,j,k] ist Master ⇒
Process VE[i,j] ∈ V[i,j,k]:
  SEMA_INIT(barrier[i,j],0);
  FOR N = 1 to q DO
    SEMA_DOWN(barrier_t[i,j]);
  END FOR N;
  FOR k = 1 to q DO
    C[i,j] ← C[i,j] + t[i,j,k];
  END FOR k;
  SEMA_UP(barrier);

Process MAIN:
  SEMA_INIT(barrier,0);
  FOR ALL (p,q,r) DO START Process VE[i,j,k];
  FOR N=1 to (p*r) DO SEMA_DOWN(barrier);
```

Paralleler Algorithmus :: Matrixmultiplikation (V)

► Erforderliches Speichermodell:

Für alle drei Fälle ist ein Speicher erforderlich, aus dem alle VEs gleichzeitig verschiedene Speicherzellen konkurrierend/parallel lesen können: **Concurrent-Read-RAM** (CR-RAM).
Ist dieses Modell nicht realisierbar (hoher Hardware-Aufwand), wird zusätzliche Synchronisation beim Lese-Speicherzugriff benötigt (i.A. implizit durch Hardware realisiert).

Paralleler Algorithmus :: Matrixmultiplikation (VI)

- **Laufzeit- und Kostenanalyse:**
Vereinfachung: $n=p=q=r$

Fall 0: Sequenzieller Algorithmus

Anzahl VE: 1
Laufzeit: $\Theta(n^3)$
Kosten: $\Theta(n^3)$

Fall 1

Anzahl VE: n^2
Laufzeit: $\Theta(n)$
Kosten: $\Theta(n^3)$

Fall 2

Anzahl VE: n
Laufzeit: $\Theta(n^2)$
Kosten: $\Theta(n^3)$

Fall 3

Anzahl VE: n^3
Laufzeit: $\Theta(\log n)$
Kosten: $\Theta(n^3 \log n)$

Fall 3b

Anzahl VE: $n^3/\log n$
Laufzeit: $\Theta(\log n)$
Kosten: $\Theta(n^3)$!!!

Berechnung der C-Summe im Fall 3

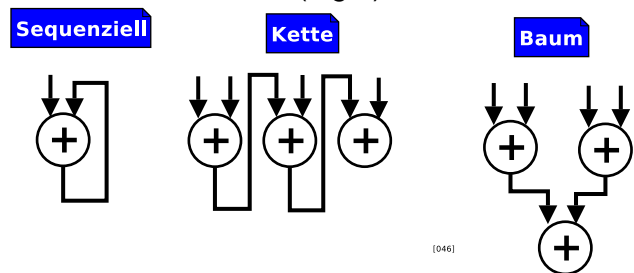
$$c_{i,j} = \sum_k t_{i,j,k} \quad (11)$$

wobei je eine VE einen t-Wert berechnet:

$$V_{i,j,k} \Rightarrow t_{i,j,k} \text{ mit Laufzeit } \Theta(1) \quad (12)$$

Implementierung der Summe:

1. Sequenziell mit einem Addierer
↳ Laufzeit: $\Theta(n)$
2. Kette aus $(n-1)$ Addierern
↳ Laufzeit $\Theta(n)$
3. Baum-Kaskade aus $(n-1)$ Addierern
↳ Laufzeit $\Theta(\log n)$



Paralleler Algorithmus :: Matrixmultiplikation (VII)

- Berechnung der Kommunikation mit Einheitswerten eines Nachrichtenaustauschs:
Message Passing (MP) ►
Aufwand eine Zeile einer Matrix zu versenden ist $MP=1$.

► Kommunikation Fall 1:

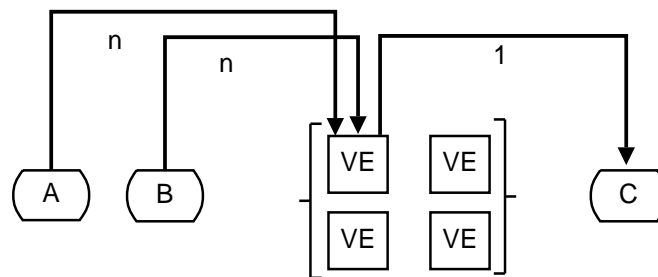
Vereinfachung: $n=p=q=r$

Berechnung eines C-Wertes mit einer VE erfordert:

$$VE_{i,j} \Rightarrow c_{i,j} \Rightarrow MP = MP(\rightarrow A_i \oplus \rightarrow B_j) + MP(C_{i,j} \rightarrow) = 2n + 1 \quad (13)$$

Summe aller Nachrichten für Berechnung von C mit n^2 VEs:

$$\sum_{i,j} VE_{i,j} \Rightarrow C \Rightarrow MP = n^2(2n + 1) = 2n^3 + n^2 \Rightarrow \Theta(n^3) \quad (14)$$



Paralleler Algorithmus :: Matrixmultiplikation (VIII)

► Kommunikation Fall 2:

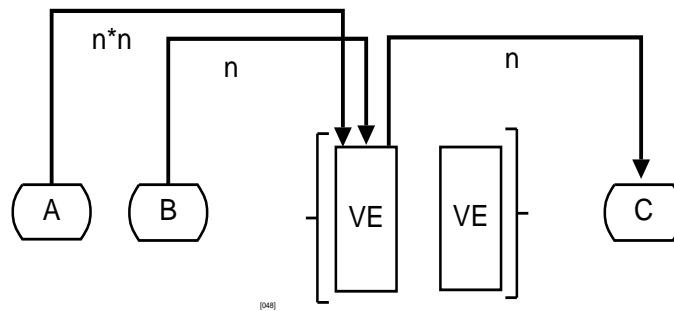
Vereinfachung: $n=p=q=r$

Berechnung einer C-Spalte mit einer VE erfordert:

$$VE_j \Rightarrow C_j \Rightarrow MP = MP(\rightarrow A_{i,j} \oplus \rightarrow B_j) + MP(C_{i,j} \rightarrow) = n^2 + n + n = n^2 + 2n \quad (15)$$

Summe aller Nachrichten für Berechnung von C mit n VEs:

$$\sum_j VE_j \Rightarrow C \Rightarrow MP = n(n^2 + 2n) = n^3 + 2n^2 \Rightarrow \Theta(n^3) \quad (16)$$



Paralleler Algorithmus :: Matrixmultiplikation (IX)

► Kommunikation Fall 3:

Vereinfachung: $n=p=q=r$

Berechnung eines t-Wertes mit einer VE erfordert:

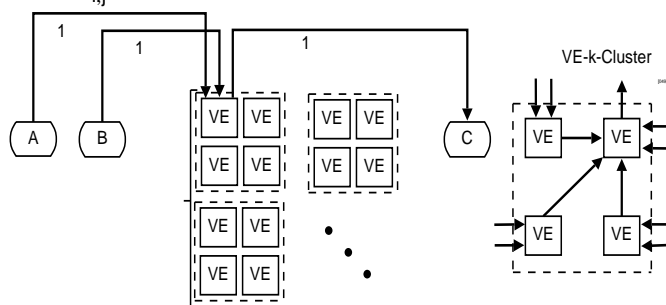
$$VE_{i,j,k} \Rightarrow t_{i,j,k} \Rightarrow MP = MP(\rightarrow A_{i,j} \oplus \rightarrow B_{i,j}) + MP(t_{i,j,k} \rightarrow) = 3 \quad (17)$$

Summe aller Nachrichten für Berechnung von C_{ij} mit n VEs:

$$\sum_k VE_{i,j,k} \Rightarrow C_{ij} \Rightarrow MP = 3n \quad (18)$$

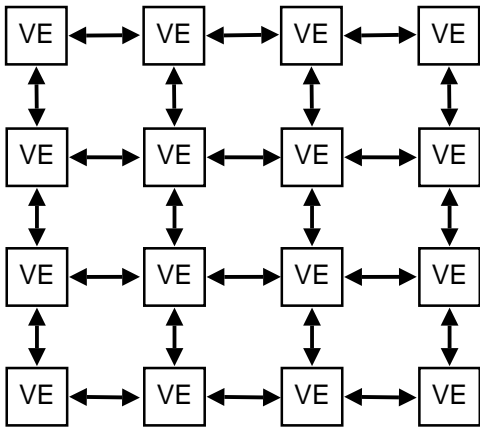
Summe aller Nachrichten für Berechnung von C mit n^3 VEs:

$$\sum_{i,j} VE_{i,j} \Rightarrow C \Rightarrow MP = n^2 3n = 3n^3 \Rightarrow \Theta(n^3) \quad (19)$$



Paralleler Algorithmus :: Matrixmultiplikation (X)

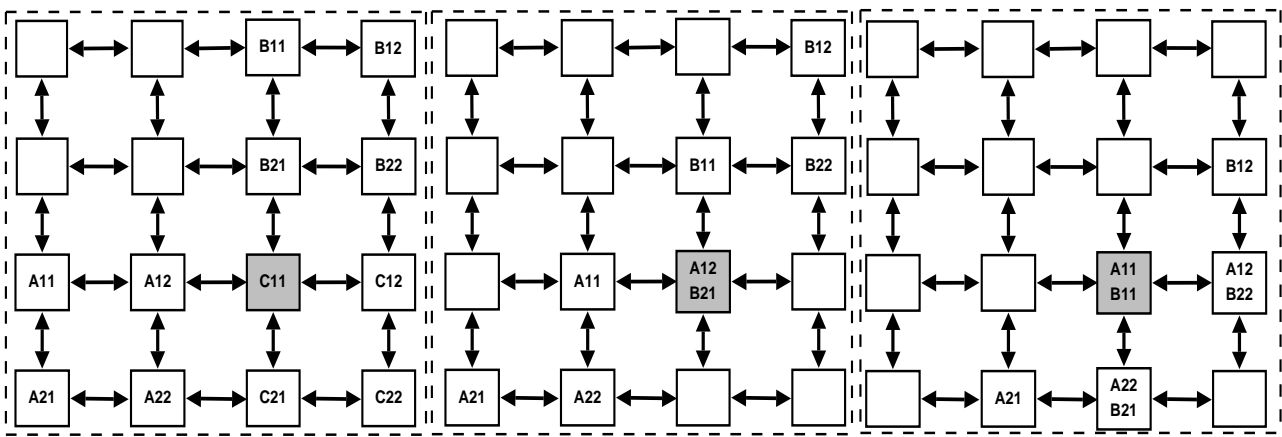
- ▶ Bisher konnten Matrizen ganzzahlig auf VEs verteilt werden. In der Realität ist aber $p=q=r \neq n!$
- ▶ Bisher wurde angenommen, daß alle VEs mit jeder anderen VE Daten mit einer Distanz/Ausdehnung $D=1$ austauschen kann. Nur möglich mit vollständig verbundenen Netzwerktopologien unter Verwendung von Kreuzschaltern.
- ▶ Gängige parallele und ökonomische Rechnertopologie: **Maschennetz**.
 - ↳ Besteht aus $n \times n$ VEs.
 - ↳ Jede VE kann mit seinem direkten Nachbarn kommunizieren.
 - ↳ Eine Nachricht hat eine maximale Reichweite von $(2n-1)$ VEs, $\Theta(n)$.



- Bisherige statische Partitionierung resultiert in zu hohem Kommunikationsaufwand in der Verteilung der Matrizen A und B sowie in der Zusammenführung der Matrix C.
- Dynamisch veränderliche Partitionierung paßt sich effizienter an Netzwerktopologie an.
- Bei Matrixoperationen mit zwei Matrizen ($n=p=q=r$) ist dafür ein $2n \times 2n$ Netz optimal geeignet.

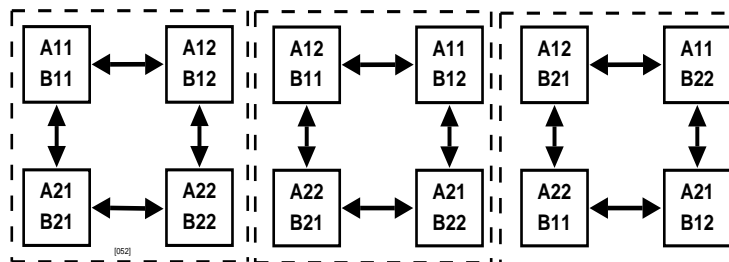
Paralleler Algorithmus :: Matrixmultiplikation (XI)

- ▶ Das $2n \times 2n$ Netz wird in vier Quadranten der Größe $n \times n$ unterteilt.
 - ↳ Die Matrizen A und B befinden sich initial im linken unteren und rechten oberen Quadranten.
 - ↳ Die Ergebnismatrix C wird schließlich im rechten unteren Quadranten zusammengeführt.
- ▶ Alle VEs die ein $A_{1,j}$ -Element besitzen werden dieses nach rechts verschoben. Alle VEs die ein $B_{i,1}$ -Element besitzen werden dieses nach unten verschoben.
 - ↳ Dieser Vorgang wird für weitere Zeilen (A) und Spalten (B) fortgesetzt.
 - ↳ Die einzelnen Elemente von A und B passieren die VEs im C-Quadranten. Die einzelnen C-Werte können mittels Summation berechnet werden.



Paralleler Algorithmus :: Matrixmultiplikation (XII)

- Die Laufzeit der Matrixmultiplikation auf dem Netz beträgt $\Theta(n)$, da n Verschiebungen durchgeführt werden.
- Die Kosten betragen $\Theta(n^3)$, vergleichbar mit sequenzieller Ausführung. D.h. diese Partitionierung und Architektur ist kostenoptimal.
- Das $2n \times 2n$ Netz wird nur in drei Quadranten genutzt. Benötigt werden daher nur $3n^2$ VEs. Aus Sicht der Rechnerarchitektur ungünstig zu implementieren und nicht generisch, d.h. abhängig vom verwendeten Algorithmus.
- Reduktion dieses Verfahrens auf $n \times n$ Netz möglich. Dazu werden die Matrizen A, B und C überlagert, d.h. $VE_{1,1} \rightarrow \{A_{1,1}, B_{1,1}, C_{11}\}$.
- Die Zeilenelemente von A werden dann nach vorherigen Ablaufschema gegen den Uhrzeigersinn rotiert (nur horizontal), und die Spaltenelemente von B im Uhrzeigersinn (nur vertikal).
- Man erhält gleiche asymptotische Grenzwerte für die Laufzeit $\Theta(n)$ und Kosten $\Theta(n^3)$!



Paralleler Algorithmus :: Matrixmultiplikation mit NETRA (I)

- Matrixmultiplikation $C=A \times B$ soll mit NETRA-Rechnerarchitektur in zwei verschiedenen Betriebsarten durchgeführt werden:

SIMD

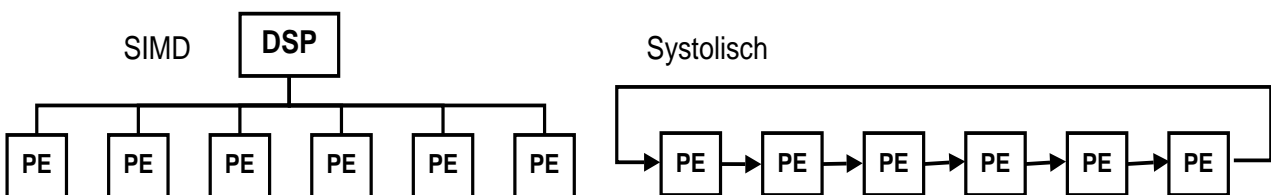
Single-Instruction-Multiple-Data-Betriebsart

Jede PE führt gleiche Instruktion durch.

Verbindung der PEs in Master-Slave-Architektur: DSP ist Master, PEs sind sternförmig um DSP angeordnet.

Systolisch

Die PEs sind in einer Pipeline angeordnet, und können Daten nur an ihren rechten Nachbarn senden.



Paralleler Algorithmus :: Matrixmultiplikation mit NETRA (II)

SIMD-Betriebsart

- Jedes Prozessorelement PE eines Clusters C erhält A und Spalte von B (Fall 2).
- Der DSP sendet Zeilen von A per broadcast an alle PEs im Cluster.
- Die PEs können die inneren Produkte von C berechnen.

DSP	P[k]
FOR i=0 TO P-1 DO CONNECT(DSP,P[i]); OUT(COL(B,i)); END FOR i; CONNECT(DSP,all) FOR i=0 TO P-1 DO FOR j = 0 to P-1 DO OUT(A[i,j]); END FOR j; END FOR i;	IN(COL(B,i)); C[i,k]=0; FOR j = 0 to P-1 DO IN(A[i,j]); C[i,k]=C[i,k]+A[i,j]*B[j,k]; END FOR j;

Paralleler Algorithmus :: Matrixmultiplikation mit NETRA (III)

Systolische-Betriebsart

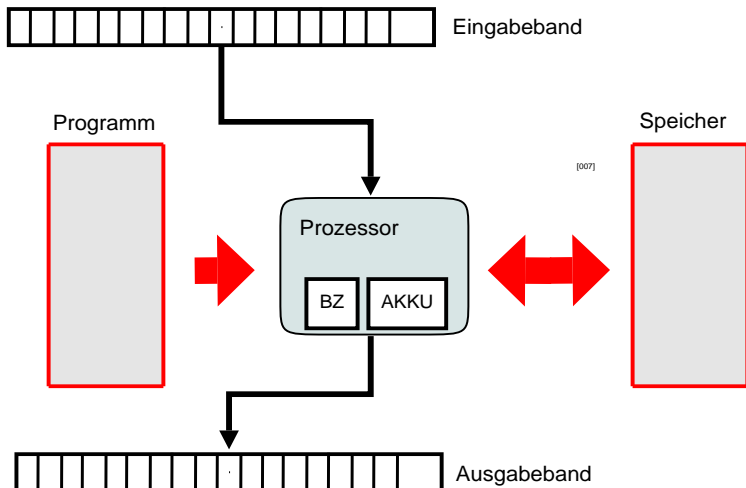
- Der DSP kann den Cluster als zirkuläres lineares Array rekonfigurieren. Die PEs sind in einer Pipeline angeordnet. Jedes Prozessorelement PE eines Clusters C erhält Zeile von A und Spalte von B.
- Der DSP sendet Zeile i von A an PE i im Cluster.
- Die PEs können die inneren Produkte von C berechnen. Jedes PE schiebt in der Pipeline Werte von A zu seinen nächsten Nachbarn.

DSP	P[i]
FOR i=0 TO P-1 DO CONNECT(DSP,P[i]); OUT(COL(B,i)); OUT(ROW(A,i)); END FOR i; FOR EACH P CONNECT(P[i] TO P[i+1] MOD P); C[i,i]=0; FOR j = 0 TO P-1 DO C[i,i]=C[i,i]+A[i,j]*B[j,i]; OUT(A[i,j]), IN(A[i-1,j]); END FOR j; REPEAT LOOP \forall ROW	IN(COL(B,i)); IN(ROW(A,i)); C[i,i]=0; FOR j = 0 TO P-1 DO C[i,i]=C[i,i]+A[i,j]*B[j,i]; OUT(A[i,j]), IN(A[i-1,j]); END FOR j; REPEAT LOOP \forall ROW

Rechnermodelle :: Verallgemeinerte Registermaschine (I)

► Die verallgemeinerte Registermaschine (Random Access Machine RAM) stellt ein Modell für konventionelle Einprozessorrechner dar, bestehend aus:

1. einer Rechen- oder Verarbeitungseinheit VE,
2. einem Programm,
3. einem Schreib- und Lesespeicher, bestehend aus abzählbar vielen Registern L_0, L_1, \dots ,
4. Eingabeband, von dem Daten nur einmalig sequenziell gelesen werden,
5. Ausgabeband, auf das die Ergebnisse sequenziell geschrieben werden.



In der Komplexitätstheorie sind RAMs ein wichtiges Hilfsmittel für Laufzeitberechnungen von Algorithmen und der Komplexität von Algorithmen.

Rechnermodelle :: Verallgemeinerte Registermaschine (II)

► Die verallgemeinerte Registermaschine unterstützt nur elementare Operationen

Assembler

1. Datentransfer
2. Speicher-/ Registerzugriff
3. Arithmetik
4. Sprünge

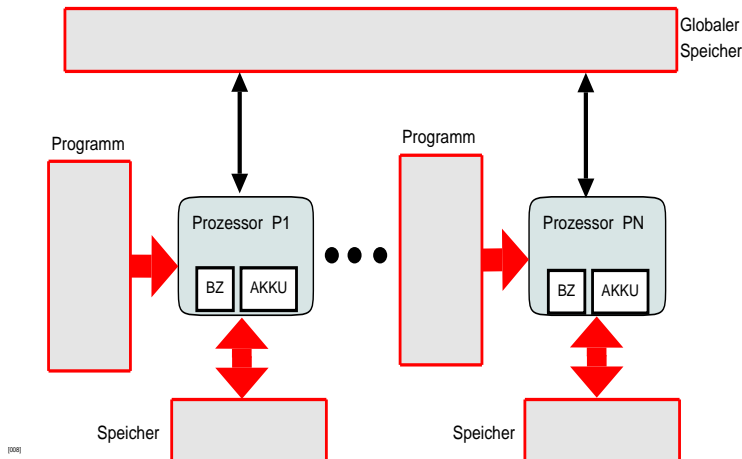
Befehl	Wirkung
LOAD op	$BZ \leftarrow BZ+1; Akku \leftarrow op;$
STORE op	$BZ \leftarrow BZ+1; op \leftarrow Akku;$
READ	$BZ \leftarrow BZ+1; Akku \leftarrow in();$
WRITE	$BZ \leftarrow BZ+1; out(Akku);$
ADD op	$BZ \leftarrow BZ+1; Akku \leftarrow Akku+op;$
SUB op	$BZ \leftarrow BZ+1; Akku \leftarrow Akku-op;$
JUMP i	$BZ \leftarrow i;$
JZERO i	IF $Akku=0$ THEN $BZ \leftarrow i$ ELSE $BZ \leftarrow BZ+1;$
HALT	Stop.

Adressierungsarten

Art	Wirkung
direkt	$L[\text{wert}]$
indireket	$L[L[\text{wert}]]$

Rechnermodelle :: Parallele Registermaschine (I)

- Erweiterung der allgemeinen Registermaschine (von Fortune und Wyllie, 1978) zu einer parallelen Registermaschine PRAM.
- Modellierung eines idealisierten speichergekoppelten Parallelrechners ohne Beachtung von Synchronisation und Speicherzugriffslatenz.
- Ein N-Prozessor PRAM besteht aus N identischen Prozessoren, die alle auf einen **gemeinsamen Speicher** mit den Speicherzellen G_0, G_1, \dots zugreifen.
- Alle Prozessoren arbeiten synchron (gemeinsamer Takt), jeder Prozessor gleicht einer verallgemeinerten RAM mit einem **lokalen Speicher**.
- Ein- und Ausgabeband werden im gemeinsamen Speicher implementiert.



Rechnermodelle :: Parallele Registermaschine (II)

➤ Die parallele verallgemeinerte Registermaschine unterstützt nur elementare Operationen

Assembler

Befehl	Wirkung
LOAD op	$BZ \leftarrow BZ+1; Akku \leftarrow op;$
STORE op	$BZ \leftarrow BZ+1; op \leftarrow Akku;$
READ	$BZ \leftarrow BZ+1; Akku \leftarrow in();$
WRITE	$BZ \leftarrow BZ+1; out(Akku);$
ADD op	$BZ \leftarrow BZ+1; Akku \leftarrow Akku+op;$
SUB op	$BZ \leftarrow BZ+1; Akku \leftarrow Akku-op;$
JUMP i	$BZ \leftarrow i;$
JZERO i	IF $Akku=0$ THEN $BZ \leftarrow i$ ELSE $BZ \leftarrow BZ+1;$
FORK z	$\Psi(P_i) \Rightarrow \Psi(P_j); BZ_j = z;$
HALT	Stop.

1. Datentransfer
2. Speicher-/ Registerzugriff
3. Arithmetik
4. Sprünge
5. Prozessor- und Prozeßkontrolle

Ψ : Lokaler Speicher \oplus Akku

➤ FORK: Wird diese Instruktion von Prozessor P_i ausgeführt, wird ein neuer Prozessor P_j ausgewählt:

- ◆ Prozessor j erbt Kopie des lokalen Speichers und des Akkus von Prozessor i
- ◆ Prozessor j wird gestartet mit $BZ=z$

Rechnermodelle :: Parallele Registermaschine (III)

Globaler Speicher

Eine PRAM besitzt (unlimitierten) globalen Speicher, der von allen Prozessoren geteilt wird:

Konkurrierendes Lesen - Exklusives Schreiben

Gleichzeitiges Lesen mehrerer Prozessoren einer Speicherzelle $\varphi(N)$ ist erlaubt, aber der Schreibzugriff auf einer Speicherzelle $\varphi(N)$ ist nur von einem Prozessor gleichzeitig erlaubt.

➔ Da keine Synchronisation modelliert wird, führt konkurrierender Schreibzugriff zum Halt der PRAM!

Lesen vor Schreiben

Mehrere Prozessoren dürfen eine Speicherzelle $\varphi(N)$ lesen, wenn gleichzeitig ein Prozessor einen Schreibzugriff durchführt. Der Inhalt von $\varphi(N)$ ändert sich erst, wenn ALLE lesenden Prozessoren ihren Zugriff beendet haben!

Gleichzeitiges Lesen und Schreiben

Mehrere Prozessoren dürfen gleichzeitig auf verschiedene Speicherzellen $\varphi(N_1), \varphi(N_2) \dots \varphi(N_i)$ zugreifen (lesend konkurrierend, schreibend exklusiv).

Ein Programm

Alle Prozessoren führen das gleiche Programm aus. Ein Programm ist eine endliche Sequenz von Instruktionen. Die Instruktionsströme müssen aber nicht identisch sein!

Start

Alle Speicher werden gelöscht - der erste Prozessor P_1 startet mit der Ausführung

Klassifikation Rechnerarchitektur (I)

Eine **Datenverarbeitungsanlage** enthält:

VE

Verarbeitungseinheit für Daten ➔ Generischer Prozessor, Applikationspezifischer Prozessor, Applikationspezifische Digitallogik, Teileinheit

SM

Speichermodul ➔ RAM, Registerbank, Cache

KE

Kontrolleinheit für die Ablaufsteuerung, kann in VE enthalten sein.

Klassifikation nach Flynn durch Beschreibung von Daten- und Kontrollfluß:

SISD

Single-Instruction \otimes Single-Data Stream

SIMD

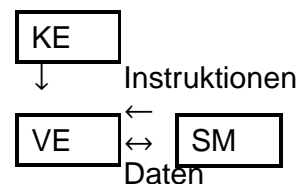
Single-Instruction \otimes Multiple-Data Stream

MISD

Multiple-Instruction \otimes Single-Data Stream

MIMD

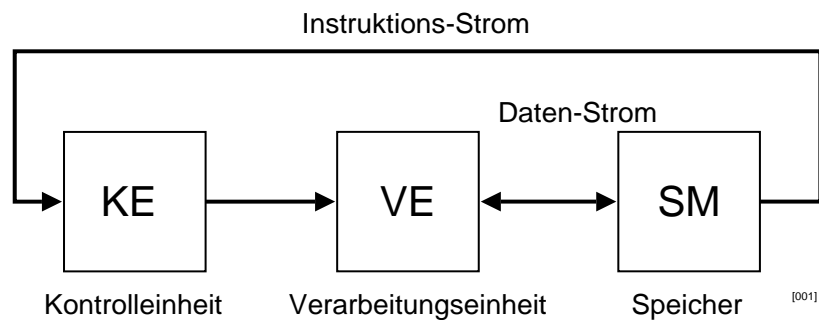
Multiple-Instruction \otimes Multiple-Data Stream



SISD-Architektur

Eigenschaften

- ▶ Von-Neuman-Architektur gehört zur SISD- Klasse
- ▶ SISD-Architekturen verarbeiten einen sequenziellen Daten- und Kontrollfluß.
- ▶ Nur eine VE und KE wird benötigt.
- ▶ Ein zentraler Speicher für Daten und Instruktionen
- ▶ Keine algorithmische Parallelisierung implementierbar - nur implizit mittels Pipeline-Verfahren.
- ▶ Programmiermodell: explizite Ablaufsteuerung; simulierte konkurrierende Programmierung mit Software-Threads.



SISD-Architektur :: Von-Neumann-Architektur

Phasen der Befehlsausführung

- I. Befehlsholphase:
 $\varnothing(BZ) \rightarrow BR$
- II. Dekodierungsphase:
 $BR \rightarrow \{S_1, S_2, \dots\}$
- III. Operandenholphase:
 $\varnothing \rightarrow R$
- IV. Befehlsausführung:
 χ
- V. Rückschreibephase:
 $R \rightarrow \varnothing$
- VI. Adreßrechnung:
 $\Psi(BZ, SR, BR) \rightarrow BZ$

BZ, BR, S

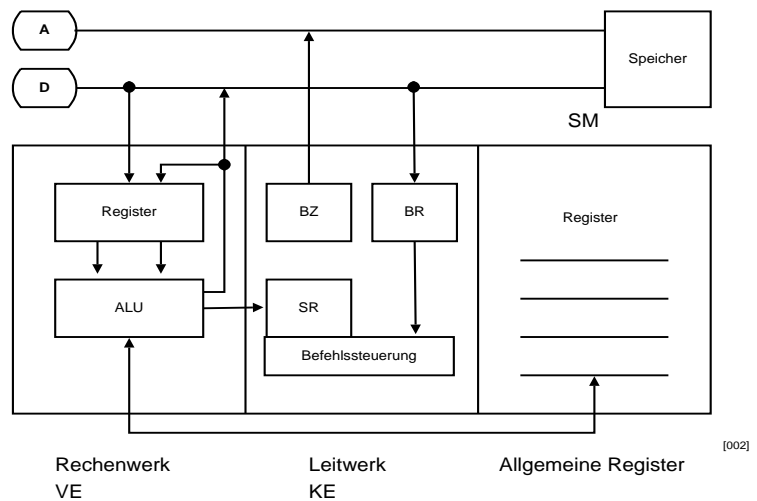
Befehlszähler, Befehlsregister, Steuersignale

SR, R, \varnothing

Statusregister, Register, Speicher

A, D

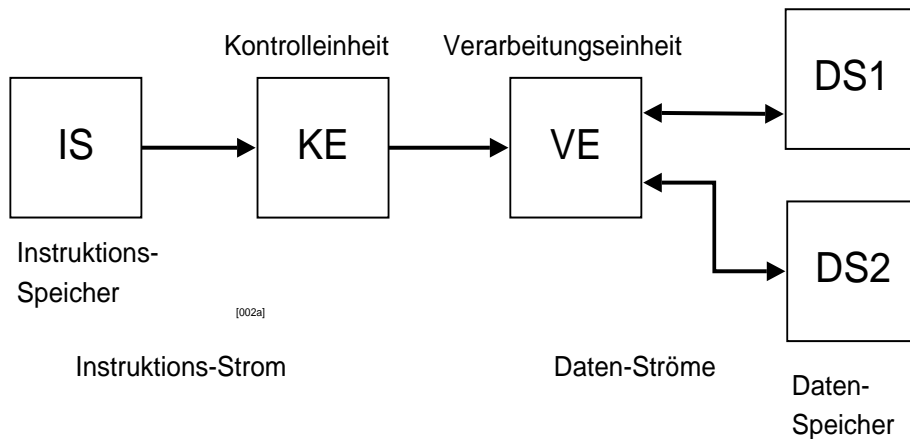
Adreß- und Datenbus



Rechenwerk, ALU \rightarrow VE
Leitwerk \rightarrow KE
Hauptspeicher \rightarrow SM

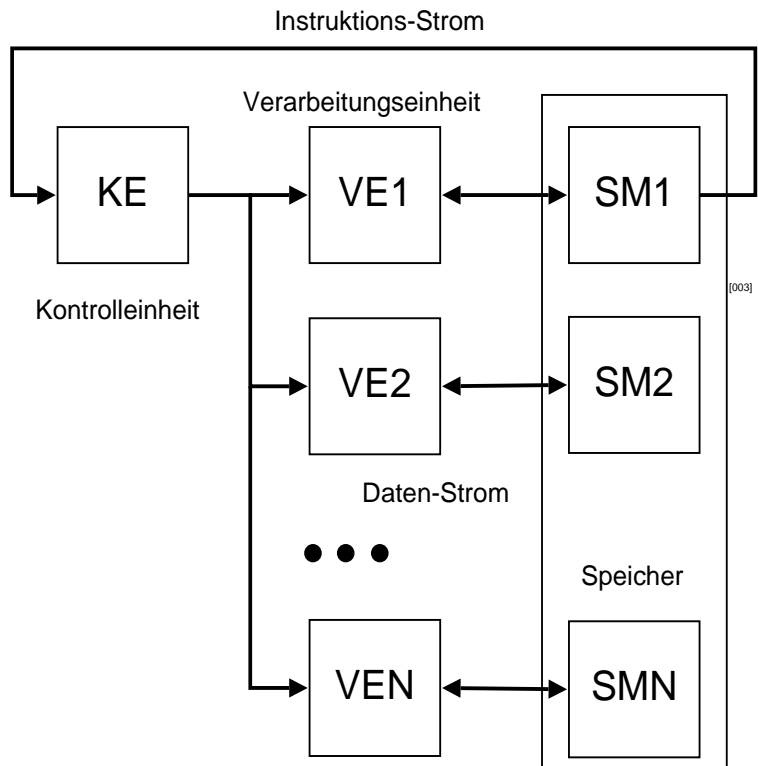
SISD-Architektur :: Harvard-Architektur

- Getrennte Daten- und Instruktionsspeicher
- Mehrere Datenbusse und Datenspeicher
- Spezialmaschine optimiert für datenintensive Algorithmen
- Parallelisierung auf Instruktionsebene: Teilphasen der Befehlsausführung können nebenläufig ausgeführt werden, z.B. Befehls- und Operandenholphase.



SIMD-Architektur

- **Parallelität auf Datenebene**
- Ablaufsteuerung mit einer gemeinsamen Kontrolleinheit KE
- **Zentrales Speichermodell**, aber jede Verarbeitungseinheit VE kann eigenen Speicher besitzen
- Jede VE bearbeitet gleiche Instruktion
- Datenoperation wird auf N VEs verteilt
- Gleiche Operation kann auf verschiedenen Operanden oder einem Operanden der Datenbreite $W=N \cdot W'$ ausgeführt werden
- Keine Programmsynchronisation erforderlich



SIMD-Architektur :: Vektor- & Array-Struktur (I)

→ Die Programmierung der SIMD-Architektur hängt von der verwendeten Subarchitektur ab.

Subarchitekturen

Vektorarchitektur

Die Verarbeitungseinheiten VE sind linear als Vektorstruktur angeordnet. Jede VE verfügt über eigenen Speicher. Die einzelnen VEs sind mit ihrem jeweiligen linken und rechten Nachbarn verknüpft, und können über diese Verbindungen Daten austauschen.

Arrayarchitektur

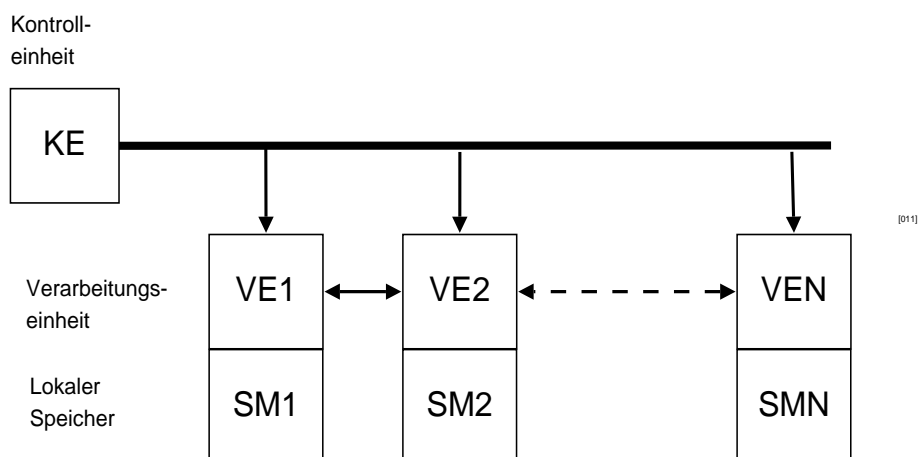
Die Verarbeitungseinheiten VE sind als zweidimensionale Matrixstruktur verknüpft. Jede VE besitzt Kommunikationsverknüpfungen mit bis zu maximal vier Nachbareinheiten, so daß Zeilen und Spaltenverbindungen bestehen.

→ Beide SIMD-Architekturen sind **Spezialmaschinen**. Sie sind gut geeignet für Algorithmen auf regulären Datenstrukturen, bei denen die gleiche Operation auf eine Vielzahl von Operanden angewendet werden soll.

- Numerische Matrixoperationen wie Matrix- und Vektormultiplikation,
- Digitale Bildverarbeitung mit zweidimensionalen Bildmatrizen

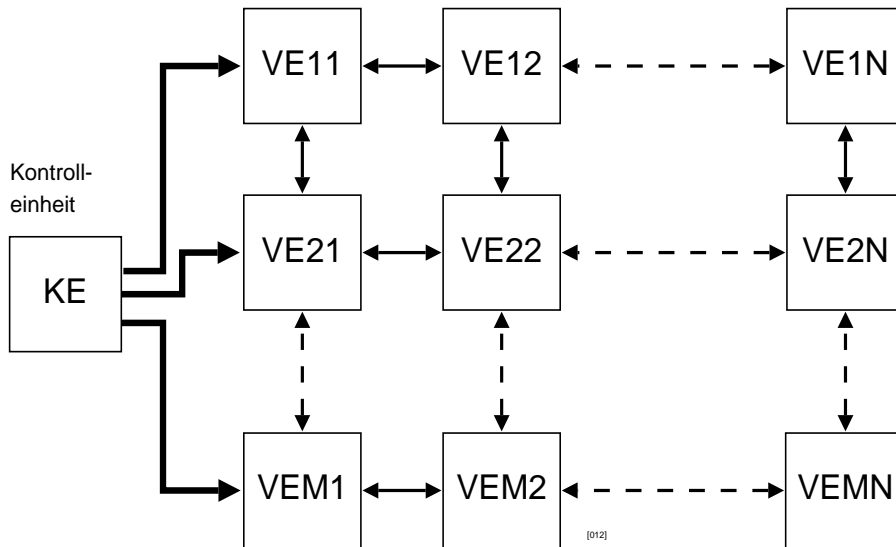
SIMD-Architektur - Vektor- & Arraystruktur (II)

Vektor-Architektur



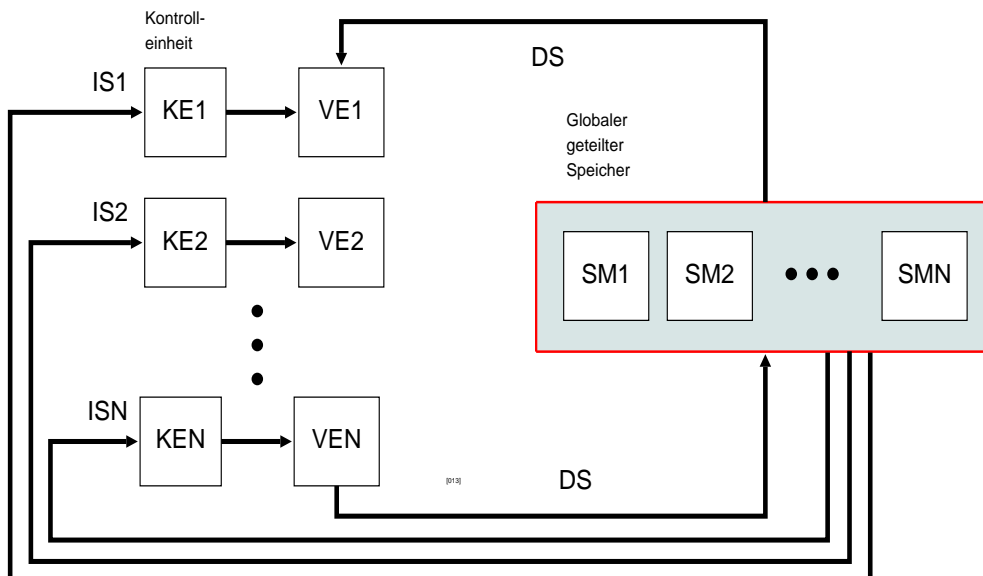
SIMD-Architektur - Vektor- & Arraystruktur (III)

Array-Architektur



MISD-Architektur

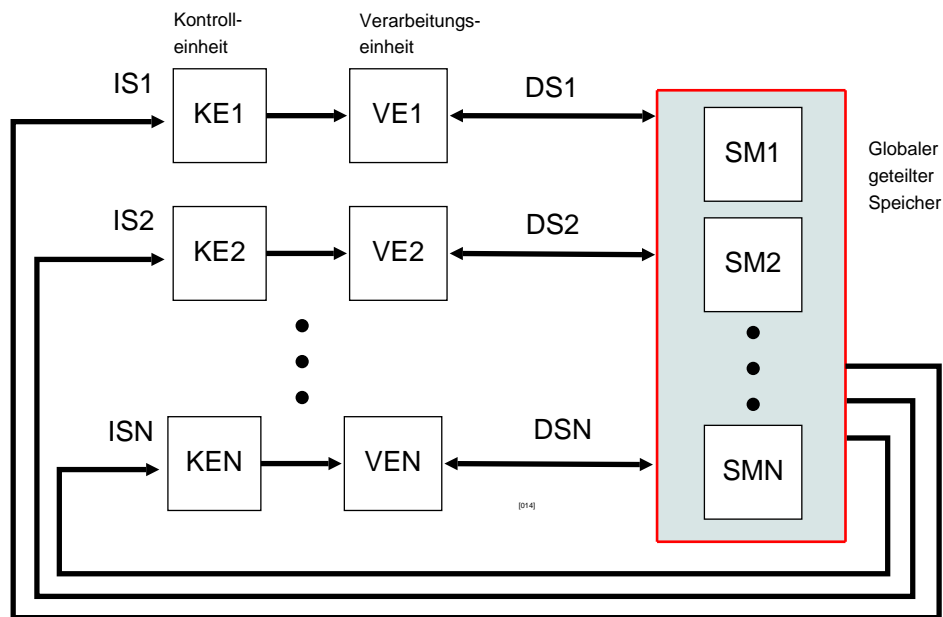
- Diese Architektur verarbeitet mehrere Instruktionen parallel. Jede VE hat eigenen IS.
- Alle Instruktionen operieren auf dem gleichen Datensatz.
- Pipeline-Computer (Instruktions-Pipeline in der CPU) gehören zur MISD-Klasse.
- Spezialmaschine - findet z.B. in der digitalen Bildverarbeitung und der Robotik Anwendung. Z.B. Objektklassifikator: das gleiche Video-Bild (oder Ausschnitt) wird auf verschiedene Algorithmen angewendet (Objektklassifikation).



MIMD-Architektur

- ▶ Jede VE arbeitet mit eigenen Instruktionsstrom.
- ▶ Jede VE arbeitet mit eigenen Datenstrom.
- ▶ Synchronisation zwischen den einzelnen VE ist erforderlich.
- ▶ Synchronisationsobjekte sind von allen VEs geteilte Ressourcen.

PRAM \equiv MIMD
Einschränkung:
gleiches Programm für
alle Prozessoren



Speichermodelle

Exklusives Lesen (ER - Exclusive Read)

Pro Zyklus kann höchstens ein Prozessor dieselbe Speicherzelle lesen.

Exklusives Schreiben (EW - Exclusive Write)

Pro Zyklus kann höchstens ein Prozessor dieselbe Speicherzelle beschreiben.

Konkurrierendes Lesen (CR - Concurrent Read)

Pro Zyklus können mehrere Prozessoren dieselbe Speicherzelle lesen.

Konkurrierendes Schreiben (CW - Concurrent Write)

Pro Zyklus können mehrere Prozessoren dieselbe Speicherzelle beschreiben. Es besteht Konfliktgefahr und Dateninkonsistenz, die mit verschiedenen Methoden gelöst werden können:

Common (C-CW)

Gleichzeitige Schreibzugriffe auf dieselbe Speicherzelle sind nur erlaubt, wenn alle Schreibzugriffe den gleichen Datenwert liefern.

Arbitrary (A-CW)

Einer der schreibenden Prozessoren gewinnt und belegt die Speicherzelle - Schutzmechanismus einer geteilten Ressource (Mutual Exclusion). Die anderen Schreibzugriffe werden ignoriert!

Priority (P-CW)

Auswahl des Prozessors nach Prioritätengewichtung (z.B. Prozessorindex).

Maßzahlen für parallele Systeme (I)

Berechnungszeit

Die Berechnungszeit T_{comp} (*computation time*) eines Algorithmus als Zeit, die für Rechenoperationen verwendet wird.

Kommunikationszeit

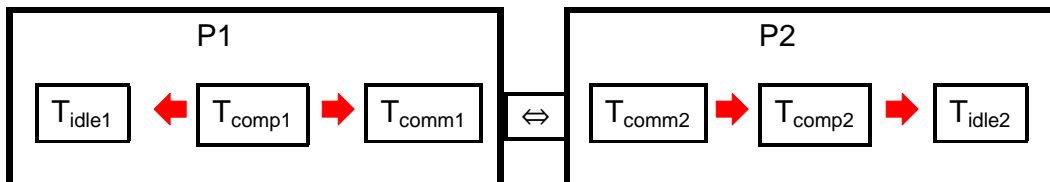
Die Kommunikationszeit T_{comm} (*communication time*) eines Algorithmus als Zeit, die für den Daten- bzw. Nachrichtenaustausch (Send- und Empfangsoperationen) zwischen Subprogrammen und Verarbeitungseinheiten verwendet wird.

Untätigkeitszeit

Die Untätigkeitszeit T_{idle} (*idle time*) eines Systems als Zeit, die mit Warten (auf zu empfangende oder sendende Nachrichten) verbracht wird.

↳ Es gilt:

$$T = T_{\text{comp}} + T_{\text{comm}} + T_{\text{idle}} = \frac{1}{N} \sum_n T_{\text{comp},n} + T_{\text{comm},n} + T_{\text{idle},n} \quad (20)$$



Maßzahlen für parallele Systeme (II)

Kommunikation

Übertragungszeit

Die Übertragungszeit T_{msg} (*message time*) ist die Zeit, die für das Übertragen einer Nachricht mit der Länge L Datenwörter zwischen zwei Prozessen oder Prozessoren benötigt wird. Sie setzt sich aus einer **Startzeit** T_s (*message startup time*) und einer **Transferzeit** T_w für ein Datenwort zusammen.

↳ Es gilt:

$$T_{\text{msg}} = T_s + L \cdot T_w \quad (21)$$

↳ Voraussetzung: Verbindungsnetz arbeitet konfliktfrei.

Startzeit

Die Startzeit wird durch die Kommunikationshard- und software bestimmt, die zur Initiierung eines Datentransfers benötigt wird, z.B. Overhead des Protokollstacks bei einer Software-Implementierung.

Transferzeit

Die Transferzeit wird durch die Bandbreite des Kommunikationsmediums und zusätzlich bei Software-Implementierung durch den Protokollstack (Datenkopie) bestimmt.

Maßzahlen für parallele Systeme (III)

Parallelisierungsgrad P

Die maximalze Anzahl von binären Stellen (bits) pro Zeiteinheit (Taktzyklus) die von einer Datenverarbeitungsanlage verarbeitet werden kann.

$$P = W \cdot B \quad (22)$$

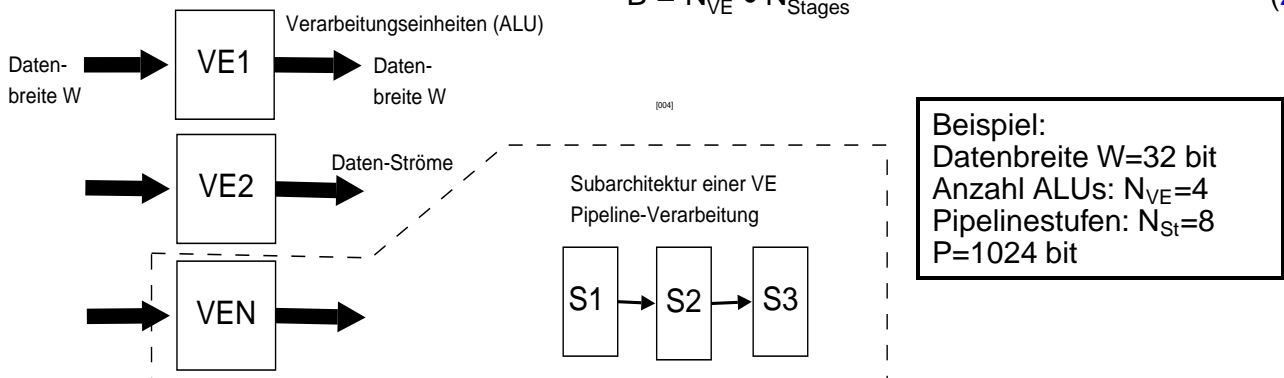
Wortlänge W

Die Wortlänge oder Wortbreite gibt die Anzahl der Bits eines Datenpfades an.

Bitslice-Länge B

Die Bitslice-Länge setzt sich zusammen aus der Anzahl von Verarbeitungseinheiten VE, die parallel ausgeführt werden können, und der Anzahl der Pipeline-Stufen einer VE:

$$B = N_{VE} \cdot N_{Stages} \quad (23)$$



Maßzahlen für parallele Systeme (IV)

Beschleunigung S und Kosten C

Die Beschleunigung gibt die Steigerung der Verarbeitungsgeschwindigkeit bzw. die Reduzierung der Verarbeitungszeit T an beim Übergang Anzahl Prozessoren $N=1 \rightarrow N>1$.

$$S(N) = \frac{T(1)}{T(N)}, \quad C(N) = T(N) \cdot N \quad (24)$$

Effizienz E

Die Effizienz gibt die relative Verbesserung in der Verarbeitungsgeschwindigkeit an, da die Leistungssteigerung S mit der Anzahl der Prozessoren normiert wird.

$$E(N) = \frac{S(N)}{N} \quad (25)$$

Es gilt in der Realität:

$$\frac{1}{N} \leq E(N) \leq 1 \quad (26)$$

Mehraufwand R

$$R(N) = \frac{X(N)}{X(1)} \quad (27)$$

mit X : Anzahl der auszuführenden (Einheits-)Operationen des Programms.

Maßzahlen für parallele Systeme (V)

Parallelindex I

Der Parallelindex gibt die Anzahl der parallelen Operationen pro Zeit-/Takteinheit an.

$$I(N) = \frac{X(N)}{T(N)} \quad (28)$$

Auslastung U

$$U(N) = \frac{I(N)}{N} \quad (29)$$

und entspricht dem normierten Parallelindex.

Beispiel

Ein Einprozessorsystem benötigt für die Ausführung von 1000 Operationen 1000 (Takt-)Schritte. Ein Multiprozessorsystem mit 4 Prozessoren benötigt dafür 1200 Operationen, die aber in 400 Schritten ausgeführt werden können:

$$\begin{aligned} X(1) &= 1000 \text{ und } T(1) = 1000, \quad X(4) = 1200 \text{ und } T(4) = 400 \\ \Rightarrow \\ S(4) &= 2.5 \text{ und } E(4) = 0.625, \quad I(4) = 3 \text{ und } U(4) = 0.75 \end{aligned}$$

Im Mittel sind 3 Prozessoren gleichzeitig aktiv, da jeder Prozessor nur zu 75% ausgelastet ist!

Amdahl's Gesetz (I)

Eine kleine Zahl von sequenziellen Operationen kann den Performanzgewinn durch Parallelisierung signifikant reduzieren.

- Sequenzieller Anteil der Berechnungszeit T eines Algorithmus in [%]: η
Paralleler Anteil: $(1-\eta)$
- Synchronisation zwischen konkurrierenden Tasks oder Verarbeitungseinheiten verursacht immer $\eta > 0$!

Beispiel: Schutz einer geteilten Ressource mit einer Mutex.

- Kommunikation verursacht immer $\eta > 0$!
- Es gilt dann für die gesamte Berechnungszeit eines parallelen Systems:

$$T(N, \eta) = \eta T(1) + \frac{(1-\eta)T(1)}{N} \quad (30)$$

Daraus läßt sich eine Obergrenze der Beschleunigung S mit zusätzlicher Abhängigkeit von η ableiten:

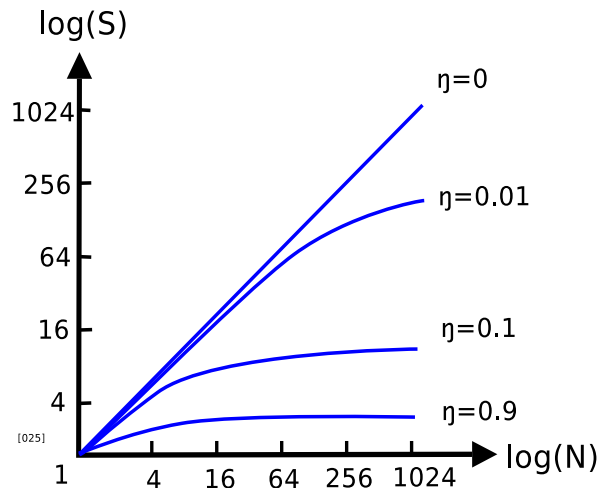
$$S(N, \eta) = \frac{T(1)}{T(N, \eta)} \leq \frac{N}{\eta(N-1) + 1} \quad (31)$$

Amdahl's Gesetz (II)

- ▶ Beispiel: Ein Algorithmus mit einem sequenziellen Anteil von $\eta=10\%$ und einem parallelen Teil von 90% wird A.) mit $N=10$ Prozessoren, und B.) mit $N=20$ Prozessoren ausgeführt. Die Obergrenze der Beschleunigung S ist dann:

$$\text{A.) } S(10)=5.26 \quad \text{B.) } S(20)=6.0$$

- ↳ Verdopplung der Prozessoren erhöht nicht mehr signifikant die Beschleunigung!
- ▶ Beschleunigung S in Abhängigkeit von η und Anzahl der Prozessoren N :



Parallele versus verteilte Systeme

Parallele Systeme

Shared Memory Multiprocessor

Eng gekoppelte Systeme mit einer Anzahl $N > 1$ von partiellen oder vollständigen Verarbeitungseinheiten und $M \geq 1$ Kontrolleinheiten, die wenigstens über eine gemeinsame Speicherressource verfügen.

↳ Die Kommunikation zwischen Verarbeitungs- und Kontrolleinheiten kann

- über den gemeinsamen globalen Speicher direkt,
- oder über direkte geschaltete Verbindungen,
- oder über Nachrichtenaustausch stattfinden.

Verteilte Systeme

Message Passing Multicomputer

Lose gekoppelte Systeme mit einer Anzahl $N > 1$ von vollständigen Verarbeitungs- und Kontrolleinheiten.

↳ Die Kommunikation findet nur nachrichtenbasiert über serielle oder parallele Kommunikationskanäle statt.

↳ Einzelne Knoten des verteilten Systems können autonom operieren.

Netzwerk und Kommunikation (I)

Kontrollarchitektur und Schichtenmodell für System-On und Off-Chip-Netzwerke

- ▶ Klassische Netzwerke verbinden vollständige Rechnerknoten miteinander, z.B. Ethernet. Nur kleiner Teil der Netzwerkkommunikation findet auf Hardware-Ebene statt. Protokollstack ist auf Programm-Ebene implementiert.
- ▶ Im Hardware-Entwurf gewinnen Netzwerke als Verbindungsstrukturen zwischen einzelnen Systemkomponenten an Bedeutung (System-Off-Chip-Netzwerk). Größerer Teil der Netzwerkkommunikation wird in Hardware implementiert.
- ▶ Aktuell durch zunehmende Erhöhung der funktionellen Leistungsdichte von Hardware-Systemen auf einem Chip gewinnen Netzwerkstrukturen innerhalb eines Chips zur Verbindung von Subsystemen an Bedeutung (System-On-Chip-Netzwerke).
- ▶ Bei System-On-Chip-Netzwerken sind die Energieeffizienz, die Leistungsfähigkeit und die benötigten Hardware-Ressourcen wesentliche Design-Parameter.
- ▶ Traditionell wurden Bussystemen als geteilte Kommunikations-Ressourcen eingesetzt. Die begrenzte Übertragungskapazität führt zu Übergang zu komplexeren Netzwerkstrukturen, aber verbunden mit erhöhten Ressourcenaufwand.
- ▶ Bussysteme (indirekte Netzwerke) besitzen eine geringere Energieeffizienz [J/bit] als direkte komplexe Netzwerke.

Netzwerk und Kommunikation (II)

Kontrollarchitektur und Schichtenmodell für System-On und Off-Chip-Netzwerke

Physikalischer-Layer

Elektrische oder optische Verbindungsleitungen zwischen einzelnen Netzwerkknoten. Übertragung von Daten ist als unzuverlässig klassifiziert. Datenverlust oder Manipulation kann auftreten.

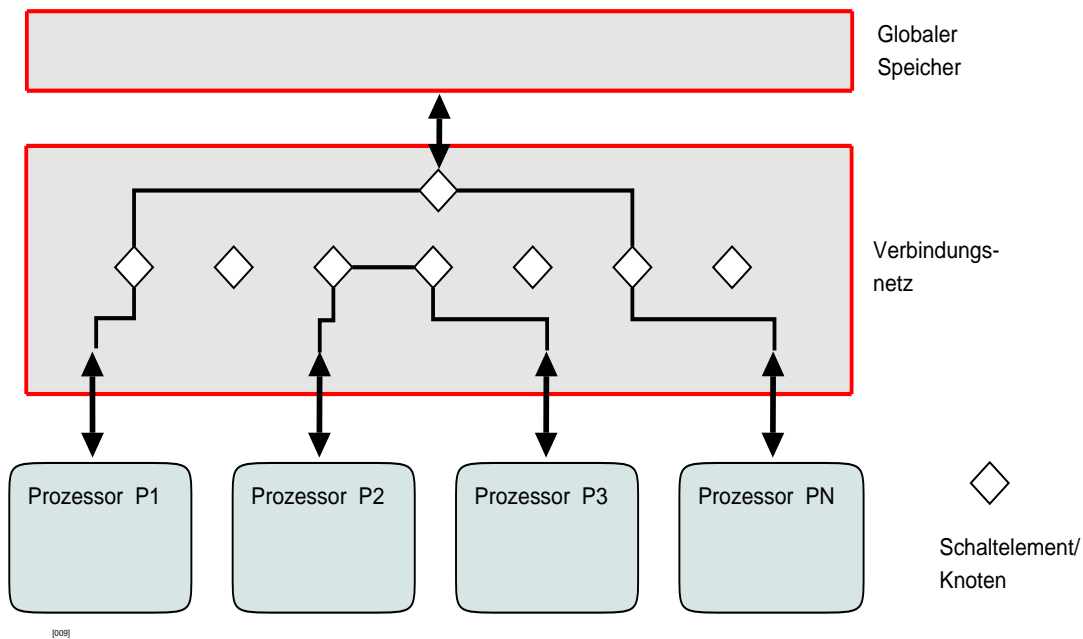
Datalink-Layer

Abstraktion des physikalischen Übertragungsmediums. Einführung von Zuverlässigkeit auf Protokoll-Ebene.

- Datenkodierung und -dekodierung
- Fehlererkennung- und -korrektur
- Aber: Energieeffizienz und Latenz kontra Zuverlässigkeit (Hardware-Overhead)
- Kontrolle von geteilten Kommunikationskanälen

Kommunikationsstrukturen (I)

- Das Verbindungsnetz ist das Medium, über das die Kommunikation der Verarbeitungs- und Kontrolleinheiten VE&KE (Prozessoren) untereinander und der Zugriff auf gemeinsame Ressourcen wie Hauptspeicher stattfindet.



Kommunikationsstrukturen (II)

- Beurteilungskriterien für die Bewertung:

Komplexität

Hardware-Aufwand für das Verbindungsnetz gemessen an der Anzahl und Art der Schaltelemente und Verbindungen.

Verbindungsgrad

Der Verbindungsgrad eines Kommunikationsknoten beschreibt die Anzahl der Verbindungen, die von dem Knoten zu anderen Knoten bestehen.

Geometrische Ausdehnung

Maximale Distanz für die Kommunikation (Anzahl der Knoten oder Schaltelemente) zwischen zwei Kommunikationsteilnehmern (VE/KE) [maximale Pfadlänge].

Regelmäßigkeit

Regelmäßige Strukturen in der Verbindungsmatrix lassen sich i.A. einfacher implementieren und modellieren (simulieren).

Erweiterbarkeit

Dynamische oder statische Anpassung an Veränderung der Parallelarchitektur.

Durchsatz/Übertragungsbandbreite

Maximale, meist theoretisch errechnete Übertragungsleistung des Verbindungsnetzes oder einzelner Verbindungen in [Mbits/s]

Kommunikationsstrukturen (III)

- Beurteilungskriterien für die Bewertung:

Skalierbarkeit

Fähigkeit eines Verbindungsnetzes, bei steigender Erhöhung der Knotenzahl die wesentlichen Eigenschaften beizubehalten, wie z.B. den Datendurchsatz oder die Kommunikationslatenz.

Blockierung

Ein Verbindungsnetz heißt blockierungsfrei, falls jede gewünschte Verbindung zwischen zwei Kommunikationsteilnehmern (inklusive Speicher) unabhängig von bestehenden Verbindungen hergestellt werden kann.

Ausfalltoleranz oder Redundanz

Möglichkeit, Verbindungen zwischen einzelnen Knoten selbst dann noch zu schalten, wenn einzelne Elemente des Netzes (Schaltelemente, Leitungen) ausfallen.

➔ Ein fehlertolerantes Netz muß also zwischen jedem Paar von Knoten mindestens einen zweiten, redundanten Weg bereitstellen.

Komplexität der Wegfindung

Art und Weise, wie zwischen einem Kommunikationsstart- und endpunkt eine Nachricht vom Sender zum Zielknoten bestimmt wird: **Routing**.

Die Wegfindung sollte einfach sein, und mittels eines schnellen Hardware-Algorithmus in jedem Verbindungselement implementierbar sein.

Kommunikationsstrukturen :: Netze (I)

Netzwerkverbindungen

Statische Netze

Ein statisches Netz wird durch fest installierte Verbindungen zwischen Netzknoten bestimmt.

Beispiel: Ethernet .

Dynamische Netze

Ein dynamisches Netz besitzt keine fest installierte, sondern schaltbare Verbindungen zwischen Netzknoten. Beispiel: Bussysteme .

Datentransfer

Durchschalte-oder Leitungsvermittlung

Direkte Verbindung zwischen Sender und Empfänger wird für die Zeit der Informationübertragung hergestellt.

➔ Durchschaltnetze erreichen i.A. höherer Durchsatzraten als Netze mit Paketvermittlung, aber Dauerverbindung reduziert u.U. Anzahl der möglichen Verbindungen.

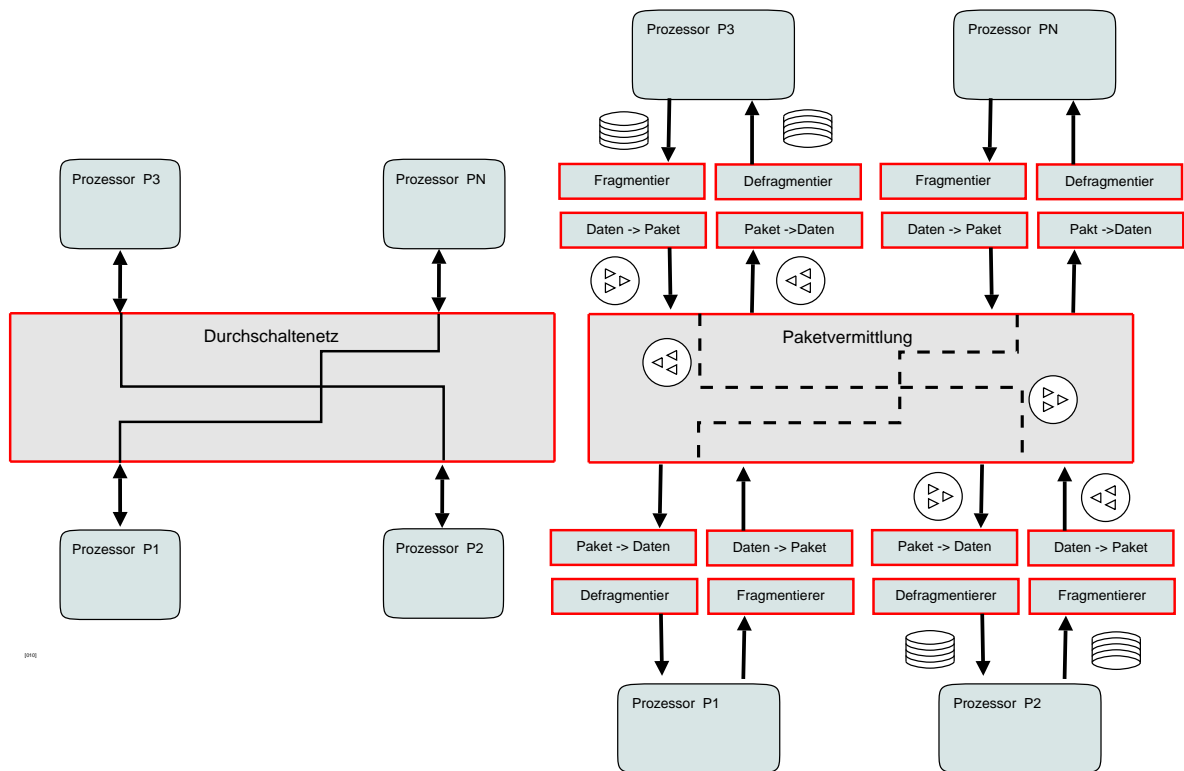
Paketvermittlung

Daten werden in Nachrichten (Paketen) gekapselt, i.A. mit fester Länge, und vom Sender an der Empfänger übertragen. Eine Datenkodierung und -dekodierung ist notwendig. Weiterhin ist i.A. eine Datenfragmentierung und -defragmentierung erforderlich.

Die Nachrichten werden vom Sender zum Empfänger entsprechend einem Wegfindalgorithmus zwischen einzelnen Knoten geroutet.

➔ Erhöhter Verwaltungsaufwand im Vergleich zu Durchschaltensetzen.

Kommunikationsstrukturen :: Netze (II)



Kommunikationsstrukturen :: Netze (III)

Vor- und Nachteile der Datentransfer-Verfahren

Schaltnetze

- ⊕ Keine Protokollimplementierung für den Nachrichtenaustausch erforderlich.
- ⊕ Die maximale Übertragungsbandbreite (Durchsatz) eines Kommunikationskanals kann erreicht werden.
- ⊖ Jeder Knoten muß mit N:1-Multiplexern ausgestattet werden.
- ⊖ Eine universelle $N \otimes N$ - Schaltmatrix (jeder Kommunikationsteilnehmer N_i kann mit jedem anderen Teilnehmer N_j mit $i \neq j$ verbunden werden) erfordert großen Hardware-Aufwand, und ist bei großen N nicht mehr wirtschaftlich.
- ⊖ Synchronisation muß explizit erfolgen.

Kommunikationsstrukturen :: Netze (IV)

Vor- und Nachteile der Datentransfer-Verfahren

Nachrichtaustausch

- ⊕ Über einen Kommunikationskanal können Verbindungen zwischen verschiedenen Kommunikationsteilnehmern aufgebaut werden.
- ⊕ Auch komplexe Netze können mit geringen Hardware-Aufwand (Routing) realisiert werden.
- ⊕ Implizite Synchronisation bereits vorhanden.

- ⊖ Implementierung eines Protokollstacks mit Datenfrag- und defragmentierung und Routing.
- ⊖ Geringerer Netto-Datendurchsatz durch Header- und Trailerdaten im Nachrichtenpaket.
- ⊖ Höhere Übertragungslatenz durch Paketadministration und Routing.

Beispiel:
Eine Implementierung eines IP-Protokollstacks mit Digitallogik benötigt ca. 1M Gates (ca. 4 Millionen Transistoren)!

Kommunikationsstrukturen :: Routing (I)

Flußsteuerung (Routing) bei nachrichtengekoppelten Netzen

Store-and-forward

- ➔ Nachricht wird von jedem Zwischenknoten in Empfang genommen
- ➔ Nachricht wird vollständig zwischengespeichert
- ➔ Sendung der Nachricht auf Pfad zum nächsten Knoten nach vollständigen Empfang

Virtual-cut-through

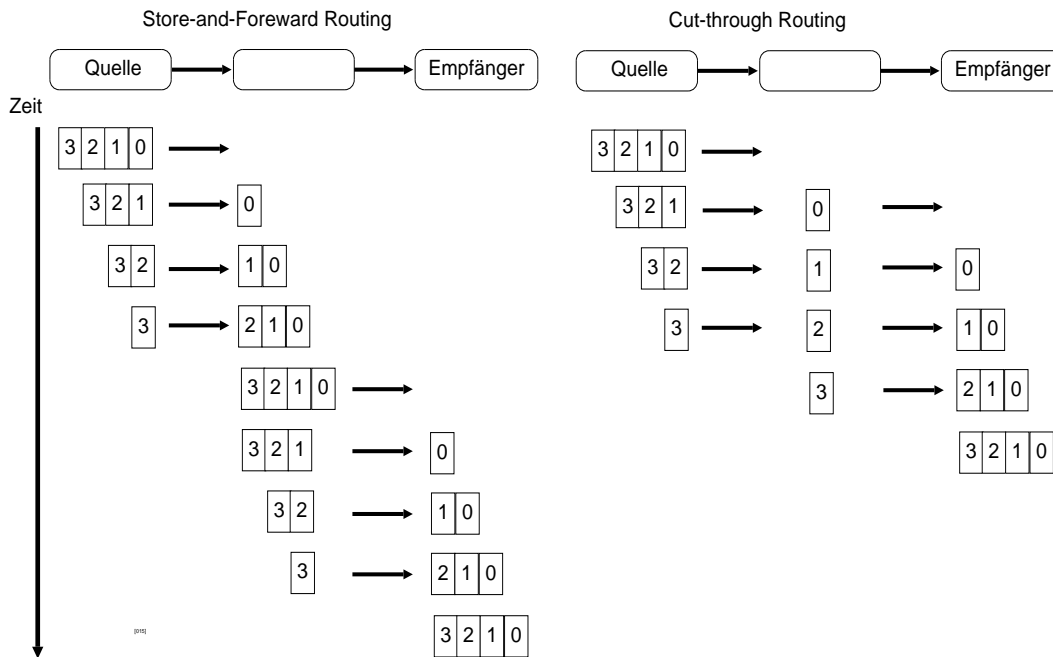
- ➔ Nachricht wird als Kette von Übertragungseinheiten transportiert
- ➔ Kopfteil der Nachricht enthält die Empfängeradresse und bestimmt den Weg
- ➔ Bei Ankunft der ersten Übertragungseinheit wird diese sofort an den nächsten Knoten weitergeleitet (Pipeline-Verfahren)
- ➔ Nachrichtenspeicherung nur im Konfliktfall (wenn Pfad belegt ist)

Wormhole-routing

- ➔ Ist der Pfad zum nächsten Knoten nicht belegt, Verhalten wie Cut-through
- ➔ Ist Kommunikationskanal belegt, müssen alle vorherigen Knoten ebenfalls Datentransfer einstellen, d.h. die Vorgänger-Knoten werden blockiert.
- ➔ D.h. keine Nachrichtenspeicherung in den Verbindungsknoten
- ➔ Es werden nur Puffer für eine Übertragungseinheit benötigt (Kopf).

Kommunikationsstrukturen :: Routing (II)

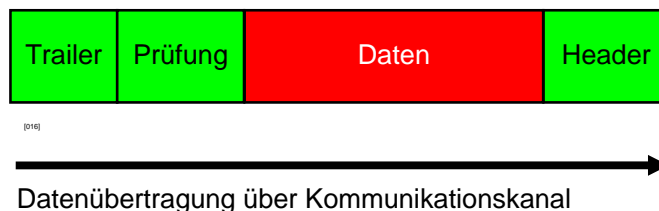
➔ Das Cut-through Routing benötigt deutlich weniger Kommunikationsschritte als das Store-and-Foreward Routing-Verfahren.



Kommunikation :: Performanz (I)

Nachrichtenaustausch

➔ Aufbau eines Nachrichtenpakets



Latenz

Die Zeit zum Übertragen einer Nachricht mit N Byte Daten vom Empfänger S zum Ziel D setzt sich zusammen aus:

$$T(N)_{S \rightarrow D} = T_{\text{Overhead}(O)} + T_{\text{RoutingDelay}(RD)} + T_{\text{Transfer}(T)} + T_{\text{Conflict}(C)} \quad (32)$$

$T_{\text{Overhead}} \gg \Theta(1)$

Zeit die benötigt wird, ein Paket vom Speicher in den Kommunikationskanal und aus dem Kanal in den Speicher zu übertragen ➔ abhängig von Hardware

$T_{\text{Transfer}} \gg \Theta(N)$

Zeit die zur eigentlichen Datenübertragung benötigt wird.

T_{Conflict}

Mittlere Zeit in der ein Kommunikationskanal blockiert ist. Abhängig von Auslastung des Kanals.

Kommunikation :: Performanz (II)

Nachrichtenaustausch

- ▶ Die Transferzeit von N Byte Nutzdaten erhöht sich durch die Paketadministration (Header & Trailer) N_{PA} und ergibt sich aus der Bandbreite B [Byte/s] des Kommunikationskanals:

$$T_{\text{Transfer}} = \frac{N + N_{PA}}{B} \quad (33)$$

- ▶ Die Routing-Zeit hängt vom verwendeten Routing-Verfahren ab:

$$T_{\text{Routing} \circ \text{SF}}(N, H) = H \left(\frac{N_{\text{Paket}}}{B} + \Delta \right) \quad (34)$$

$$T_{\text{Routing} \circ \text{CS}}(N, H) = \frac{N_{\text{Paket}}}{B} + H\Delta \quad (35)$$

- ↳ Dabei ist H die Anzahl von Knoten, die ein Paket auf seiner Route durchlaufen muß.
- ↳ Ein Hardware-Overhead beim Senden und Empfangen eines Knotens wird mit Δ gekennzeichnet.
- ↳ Es gilt:

$$T_{\text{Routing} \circ \text{CS}} < T_{\text{Routing} \circ \text{SF}} \quad (36)$$

Kommunikation :: Performanz (III)

Datenfragmentierung

- ▶ Ein Datenstrom oder ein großer Datensatz wird in kleinere Fragmente zerlegt, um einen Kommunikationskanal mit anderen Verbindungen bei niedriger Blockierungszeit T_{Conflict} nutzen zu können.
- ▶ Durch die Datenfragmentierung entstehen $M \geq 1$ aufeinanderfolgende Pakete mit einer einzelnen Datenlast von $N_F = N/M$. Die Routing-Zeit ändert sich dann zu:

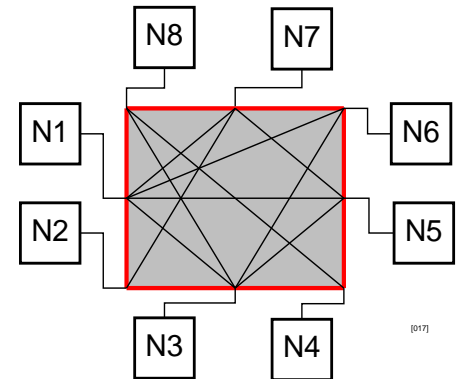
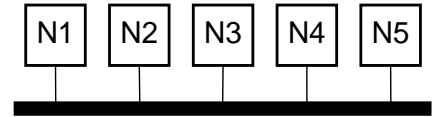
$$T_{\text{Routing} \circ \text{SF}}(N, H, N_F) = \frac{(N - N_F^{\text{Paket}})}{B} + H \left(\frac{N_F^{\text{Paket}}}{B} + \Delta \right) \quad (37)$$

$$T_{\text{Routing} \circ \text{CS}}(N, H) = \frac{N_{\text{Paket}}}{B} + H\Delta \quad (38)$$

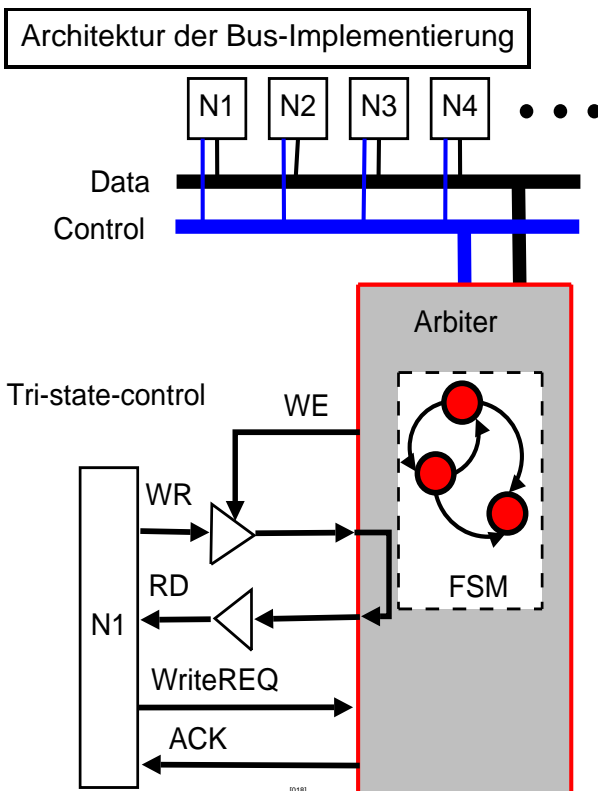
- ▶ Das Cut-through-Verfahren ergibt eine unveränderte Latenz bei Datenfragmentierung, das Store-Forward-Verfahren eine kleinere Latenz bei $H \gg 1$, da die Latenz jetzt proportional zur kleineren Paket- und nicht mehr Nachrichtengröße ist!

Netzwerktopologien :: Vollständig verbundenes Netz (I)

- Ein vollständig verbundenes Netz kann jeden Kommunikationsteilnehmer direkt mit jeden anderen verbinden.
- Es gibt nur ein Schaltnetz mit N Ein- und N Ausgängen.
- Die Ausdehnung ist für alle Pfade konstant 1.
- Der Grad des Netzes ist N.
- Einfachste Implementierung: Bus.
 - ◆ Vorteil: Hardware-Kosten $O(N)$
 - ◆ Nachteil: immer nur eine Verbindung zur gleichen Zeit; totale Bandbreite $O(1)$.
- Besser: Kreuzschienenverteiler.
 - ◆ Vorteil: Bandbreite $O(N)$
 - ◆ Nachteil: Hardware-Kosten $O(N^2)$
- **Fazit:**
 - ➔ Vollständig verbundenes Netz in der Praxis nicht praktikabel.



Netzwerktopologien :: Vollständig verbundenes Netz (II)



Busprotokoll ➔ Kontrollkommandos

LISTEN <dstaddr>

Aktueller Eigentümer des Busses (ACK=1) sendet eine Listen Nachricht an alle Teilnehmer. Nachfolgend wird nur der adressierte Teilnehmer Daten empfangen.

TALK <srcaddr>

Der aktuelle Eigentümer des Busses gibt seine Adresse bekannt.

UNLISTEN

Alle Teilnehmer warten auf Kontrollkommandos

UNTALK

Kein Teilnehmer ist Datensender.

Netzwerktopologien :: Vollständig verbundenes Netz (III)

```
entity bus01 is generic (N: integer := 8);
port (
  signal DATA: inout std_logic_vector(N-1 downto 0);
  signal WR0,WR1,WR2,WR3,WR4,WR5,WR6,WR7:in std_logic_vector(N-1 downto 0);
  signal RD0,RD1,RD2,RD3,RD4,RD5,RD6,RD7:out std_logic_vector(N-1 downto 0);
  signal REQ: in std_logic_vector(7 downto 0);
  signal ACK: out std_logic_vector(7 downto 0);
  signal CLK: in std_logic;
  signal RST: in std_logic);
end bus01;

architecture main of bus01 is
  signal temp_we,temp_ack: std_logic_vector(7 downto 0);
  signal temp_wr0,temp_wr1,temp_wr2,temp_wr3,
    temp_wr4,temp_wr5,temp_wr6,temp_wr7,
    temp_rd0,temp_rd1,temp_rd2,temp_rd3,
    temp_rd4,temp_rd5,temp_rd6,temp_rd7:
    std_logic_vector(N-1 downto 0);
begin
  DATA <= temp_wr0 when temp_we(0) = '1' else
    temp_wr1 when temp_we(1) = '1' else
    temp_wr2 when temp_we(2) = '1' else
    temp_wr3 when temp_we(3) = '1' else
    temp_wr4 when temp_we(4) = '1' else
    temp_wr5 when temp_we(5) = '1' else
    temp_wr6 when temp_we(6) = '1' else
    temp_wr7 when temp_we(7) = '1' else
    (others => 'Z');
  ...
end main;
```

VHDL
Verhaltensbeschreibung

Netzwerktopologien :: Vollständig verbundenes Netz (IV)

```
arbiter: process (CLK)
begin
  if CLK'event and CLK='1' then
    if RST = '1' then
      temp_ack <= "00000000";
    elsif temp_ack = "00000000" then
      if REQ(0) = '1' then
        temp_ack <= "00000001"; ➔ (1)
      elsif REQ(1) = '1' then
        temp_ack <= "00000010";
      elsif REQ(2) = '1' then
        temp_ack <= "00000100";
      elsif REQ(3) = '1' then
        temp_ack <= "00001000";
      elsif REQ(4) = '1' then
        temp_ack <= "00010000";
      elsif REQ(5) = '1' then
        temp_ack <= "00100000";
      elsif REQ(6) = '1' then
        temp_ack <= "01000000";
      elsif REQ(7) = '1' then
        temp_ack <= "10000000";
      end if;
    elsif (temp_ack and REQ) =
      "00000000" then
      temp_ack <= "00000000"; ➔ (2)
    end if;
  end if;
end process;
```

- Der Bus-Zugriff wird über einen "Schiedsrichter" geregelt ➤ Zustandsautomat .
- Immer nur ein Teilnehmer darf den Bus besitzen.
- Eine Bus-Anfrage wird mit der REQ-Leitung signalisiert. Jeder Teilnehmer hat eigenes Request-Signal.
- Ist die Bus-Anfrage erfolgreich, wird die ACK-Leitung aktiviert. Jeder Teilnehmer hat eigene Acknowledge-Leitung. Der Aktivierungsvektor temp_ack wird gespeichert (1).
- Der Arbitrer ist taktgesteuert.
- Erst wenn der Bus-Eigentümer sind REQ-Signal deaktiviert, wird temp_ack zurückgesetzt (2).
- Nur wenn temp_ack = 0, kann ein neuer Bus-Lock erfolgen.
- Implementierung einer Mutex.

Netzwerktopologien :: Vollständig verbundenes Netz (V)

```
temp_rd0 <= DATA; temp_rd1 <= DATA; temp_rd2 <= DATA; temp_rd3 <= DATA;
temp_rd4 <= DATA; temp_rd5 <= DATA; temp_rd6 <= DATA; temp_rd7 <= DATA;
RD0 <= temp_rd0; RD1 <= temp_rd1; RD2 <= temp_rd2; RD3 <= temp_rd3;
RD4 <= temp_rd4; RD5 <= temp_rd5; RD6 <= temp_rd6; RD7 <= temp_rd7;
temp_wr0 <= WR0; temp_wr1 <= WR1; temp_wr2 <= WR2; temp_wr3 <= WR3;
temp_wr4 <= WR4; temp_wr5 <= WR5; temp_wr6 <= WR6; temp_wr7 <= WR7;
temp_we <= REQ and temp_ack;
ACK <= temp_ack;
```

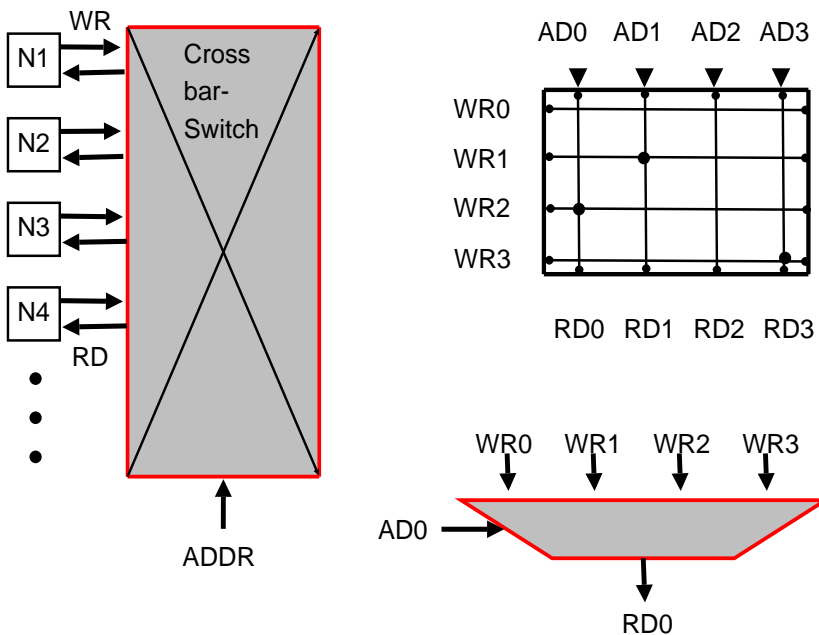
Logik-Synthese-Ergebnis

Äquivalentgatter
238
Längster Pfad
9.6 ns

Cell	Library	References	Total Area
a2_x2	sxlib	11 x 2	15 gates
a3_x2	sxlib	3 x 2	6 gates
a4_x2	sxlib	2 x 2	5 gates
inv_x1	sxlib	11 x 1	9 gates
inv_x2	sxlib	5 x 1	5 gates
inv_x4	sxlib	1 x 1	1 gates
na2_x1	sxlib	4 x 1	5 gates
na3_x1	sxlib	3 x 2	8 gates
na3_x4	sxlib	2 x 3	5 gates
na4_x1	sxlib	3 x 2	6 gates
no3_x1	sxlib	5 x 2	7 gates
no4_x1	sxlib	3 x 2	10 gates
noa2a2a2a24_x1	sxlib	16 x 5	75 gates
nts_x1	sxlib	8 x 2	16 gates
sff2_x4	sxlib	8 x 8	64 gates

Netzwerktopologien :: Vollständig verbundenes Netz (VI)

Architektur der Crossbar-Implementierung



- ▶ Jeder Adress-Selektor steuert einen Multiplexer
- ▶ Kombinatorische Logik.
- ▶ Jeder Kommunikationsteilnehmer kann mit jeden anderen direkt verbunden werden.
- ▶ Man erhält eine Multiplexer-Matrix.
- ▶ Multiplexer ⇔ UND-Verknüpfungsnetz

Netzwerktopologien :: Vollständig verbundenes Netz (VII)

```
entity crossbar01 is generic (N: integer := 8);
port (
  signal WR0,WR1,WR2,WR3,WR4,WR5,WR6,WR7:in std_logic_vector(N-1 downto 0);
  signal RD0,RD1,RD2,RD3,RD4,RD5,RD6,RD7:out std_logic_vector(N-1 downto 0);
  signal ADDR0,ADDR1,ADDR2,ADDR3 : in std_logic_vector(2 downto 0);
  signal ADDR4,ADDR5,ADDR6,ADDR7 : in std_logic_vector(2 downto 0)
);
end crossbar01;
architecture main of crossbar01 is
  signal temp_wr0,temp_wr1,temp_wr2,temp_wr3,
         temp_wr4,temp_wr5,temp_wr6,temp_wr7:
         std_logic_vector(N-1 downto 0);
begin
  RD0 <= temp_wr0 when ADDR0 = "000" else
        temp_wr1 when ADDR0 = "001" else
        temp_wr2 when ADDR0 = "010" else
        temp_wr3 when ADDR0 = "011" else
        temp_wr4 when ADDR0 = "100" else
        temp_wr5 when ADDR0 = "101" else
        temp_wr6 when ADDR0 = "110" else
        temp_wr7;
  temp_wr0 <= WR0 ;
  RD1 <=
  ...
end main;
```

VHDL
Verhaltensbeschreibung

Netzwerktopologien :: Vollständig verbundenes Netz (VIII)

Logik-Synthese-Ergebnis

Cell	Library	References	Total Area
a3_x2	sxlib	8 x 2	16 gates
inv_x1	sxlib	24 x 1	24 gates
na2_x1	sxlib	64 x 1	83 gates
no3_x4	sxlib	56 x 3	151 gates
noa2a2a2a24_x1	sxlib	128 x 5	602 gates

Äquivalentgatter
876
Längster Pfad
2.6 ns

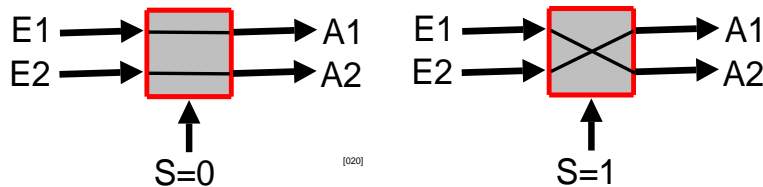
- Wesentlich höhere Anzahl an Hardware-Ressourcen im Vergleich zur Bus-Implementierung.
- Aber Latenzzeit (längste Signallaufzeit) niedriger als bei Bus-Implementierung (≈ 10 ns).
- Kein sequenzielles System - reine kombinatorische Logik.
- Reguläre Struktur durch Multiplexer.

Netzwerktopologien :: Permutationsnetze (I)

- ▶ Um den hohen Aufwand von Kreuzschienenverteiltern zu vermeiden, kann ein Verbindungsnetzwerk durch ein mehrstufiges Netzwerk mit **binären Schaltern** (Zweierschaltern) realisiert werden.
- ▶ Ein Zweierschalter besitzt zwei Eingänge $\{E_1, E_2\}$, zwei Ausgänge $\{A_1, A_2\}$ und ein Schalteingang S .
Es gibt zwei Schaltzustände:

$$A_1 = \left\{ \begin{array}{l} E_1 \text{ IF } S = 0 \\ E_2 \text{ IF } S = 1 \end{array} \right\}, \quad A_2 = \left\{ \begin{array}{l} E_2 \text{ IF } S = 0 \\ E_1 \text{ IF } S = 1 \end{array} \right\} \quad (39)$$

- ▶ Ein Permutationsnetz ist gekennzeichnet durch die und der Anzahl von Ein- und Ausgängen $p \times p$ und der Anzahl von Schaltstufen ▶ Ausdehnungsgrad $k = \log_2(p)$.
- ▶ Ein Permutationsnetz kann alle Eingänge i auf jeden Ausgang j mit $i \neq j$ schalten, aber nicht alle Pfade sind gleichzeitig schaltbar! ▶ Netz nicht konfliktfrei!
- ▶ **Reguläre Permutationsnetze** besitzen $p=2^k$ Eingänge und p Ausgänge, aufgebaut mit k Stufen und $p/2$ Zweierschaltern je Stufe.



Netzwerktopologien :: Permutationsnetze (II)

- ▶ Zwischen den Ausgängen der i -ten Stufe und den Eingängen der $(i+1)$ -ten Stufe existieren eindeutige Zuordnungen (Verbindungen) ▶ Permutationen.
- ▶ Ein- und Ausgänge werden mit einer Binärzahl als Adresse gekennzeichnet. Es gibt verschiedene Permutationstrategien, die eine Bitmanipulation dieser Adressen durchführen:

Mischpermutation M

- ▶ Kreisverschiebung der Adreßbits

$$M(a_n, a_{n-1}, \dots, a_1) = (a_{n-1}, \dots, a_1, a_n) \quad (40)$$

Kreuzpermutation K

- ▶ Vertauschen des höchst- und niederwertigsten Bits.

$$K(a_n, a_{n-1}, \dots, a_1) = (a_1, a_{n-1}, \dots, a_2, a_n) \quad (41)$$

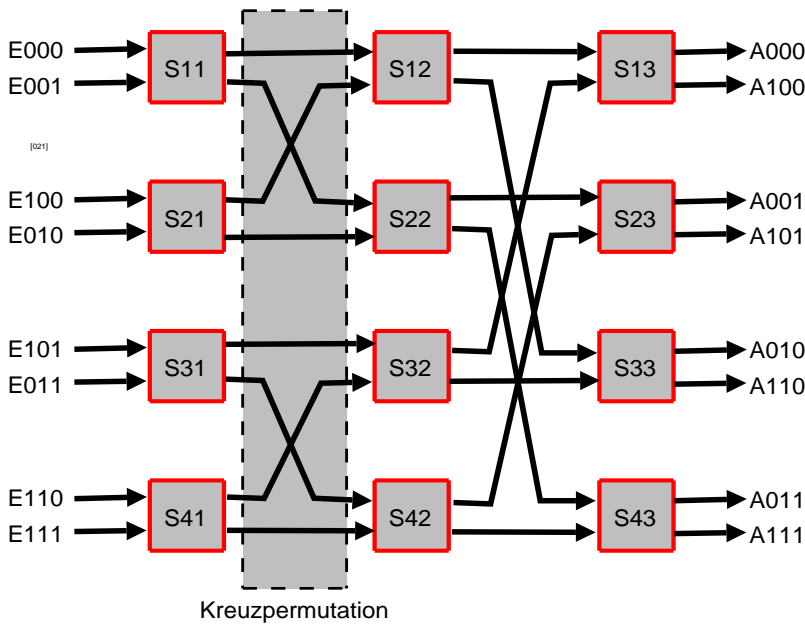
Tauschpermutation T

- ▶ Negation des niederwertigsten Bits.

$$T(a_n, a_{n-1}, \dots, a_1) = (a_n, a_{n-1}, \dots, a_2, \neg a_1) \quad (42)$$

Netzwerktopologien :: Permutationsnetze (III)

Dreistufiges-Schaltnetz (Banyan-Netzwerk)



- $k = \log_2(p) = 3$
- $p = 8$
- Verbindungsstruktur: Kreuzpermutation
- $x = k \cdot p / 2 = 12$ Zweierschalter.
- Nachteil: nicht alle Verbindungskombinationen sind konfliktfrei (gleichzeitig realisierbar)
- Adressierungs- und Konfliktmatrizen aufwendig zu implementieren.

Netzwerktopologien :: Permutationsnetze (IIIb)

➔ Schaltmatrix Σ

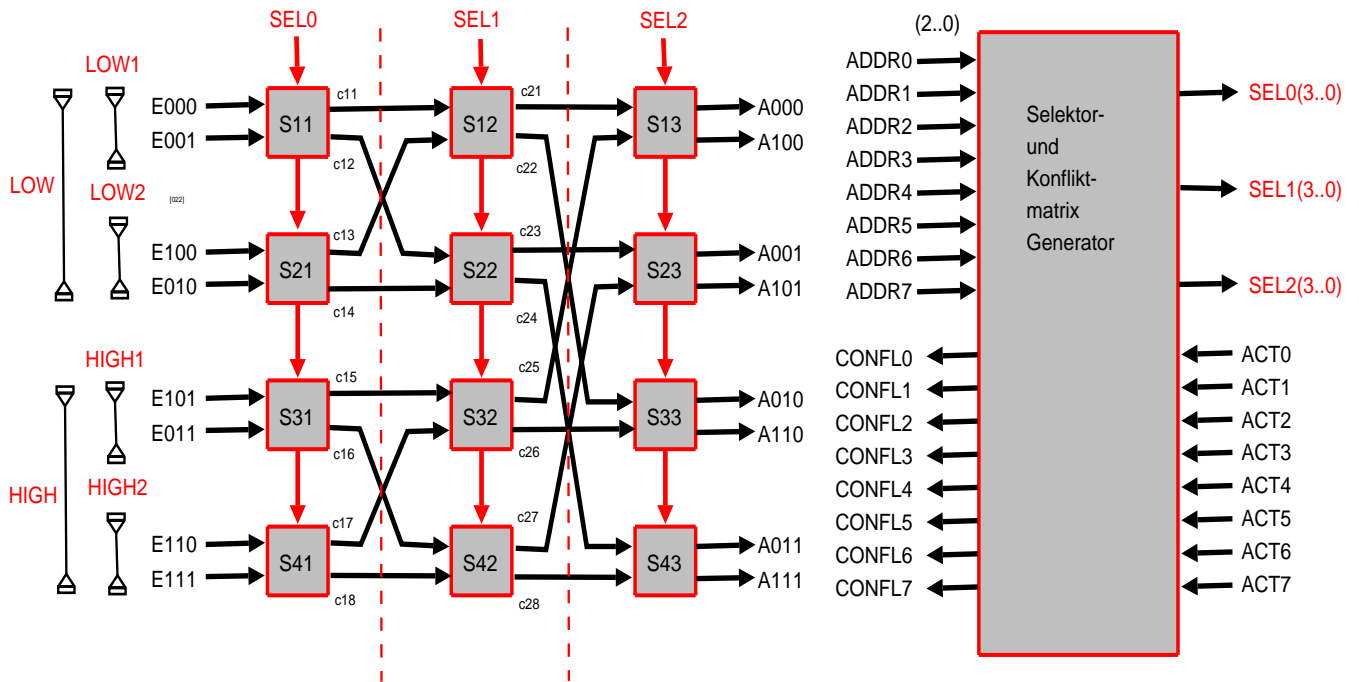
$$\Sigma(\text{ADDR}_i \forall i = 0..7) = \begin{pmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \\ S_{41} & S_{42} & S_{43} \end{pmatrix} \quad (43)$$

➔ Konfliktmatrix Γ

$$\Gamma(\text{ADDR}_i \forall i = 0..7, S_{ij} \forall i = 1..4, j = 1..3) = \begin{pmatrix} \gamma_0 & \gamma_4 \\ \gamma_1 & \gamma_5 \\ \gamma_2 & \gamma_6 \\ \gamma_3 & \gamma_7 \end{pmatrix} \quad (44)$$

Netzwerktopologien :: Permutationsnetze (IV)

Systemarchitektur: Zwischenverbindungen und Selektor-Generator

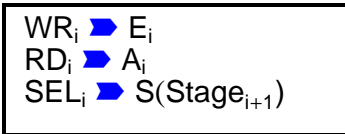


Netzwerktopologien :: Permutationsnetze (V)

```

entity switch3 is generic (N: integer := 8);
port (
  signal WR0,WR1,WR2,WR3,WR4,WR5,WR6,WR7:in std_logic_vector(N-1 downto 0);
  signal RD0,RD1,RD2,RD3,RD4,RD5,RD6,RD7:out std_logic_vector(N-1 downto 0);
  signal SEL0,SEL1,SEL2: in std_logic_vector(3 downto 0)
);
end switch3;
architecture main of switch3 is
  component switch2 is generic (N: integer := 8);
  port(
    signal E0,E1: in std_logic_vector(N-1 downto 0);
    signal A0,A1: out std_logic_vector(N-1 downto 0);
    signal S: in std_logic
  );
  end component;
  signal c11,c12,c13,c14,c15,c16,c17,c18: std_logic_vector(N-1 downto 0);
  signal c21,c22,c23,c24,c25,c26,c27,c28: std_logic_vector(N-1 downto 0);
begin
  -- stage 1
  S11: switch2 port map (
    E0 => WR0,
    E1 => WR1,
    A0 => c11,
    A1 => c12,
    S => SEL0(0)
  );
  ...

```



VHDL
Verhaltensbeschreibung

Netzwerktopologien :: Permutationsnetze (VI)

```
entity switch2 is
generic (N: integer := 8);
port (
  signal E0,E1: in
    std_logic_vector(N-1 downto 0);
  signal A0,A1: out
    std_logic_vector(N-1 downto 0);
  signal S: in std_logic
);
end switch2;
architecture main of switch2 is
begin
  A0 <= E0 when S = '0' else
    E1;
  A1 <= E1 when S = '0' else
    E0;
end main;
```

Logik-Synthese-Ergebnis

Cell	Library	References	Total Area
mx2_x2	sxlib	64 x	3 192 gates
nm2_x1	sxlib	128 x	2 294 gates

- Der eigentliche Zweierschalter switch2 wird als externe Komponente modelliert.
- Reine kombinatorische Logik mit Schaltkontrolle über Signal S.
- Adressierungs- und Konfliktlogik muß extern über Signalvektoren S eingekoppelt werden (fehlt noch).

Äquivalentgatter
486
Längster Pfad
1.3 ns

Netzwerktopologien :: Permutationsnetze (VII)

```
entity sw3_addr is
port (
  signal ADDR0,ADDR1,ADDR2,ADDR3 ...:
    in std_logic_vector(2 downto 0);
  signal ACT:
    in std_logic_vector(7 downto 0);
  signal SEL0,SEL1,SEL2:
    out std_logic_vector(3 downto 0);
  signal CONFL :
    out std_logic_vector(7 downto 0)
);
end sw3_addr;
architecture main of sw3_addr is
  signal sel_low,sel_low1,sel_low2,
    sel_high,sel_high1,sel_high2 :
    std_logic_vector(7 downto 0);
  signal sel_s11_low, sel_s11_high,
    sel_s21_low,...:
    std_logic_vector(7 downto 0);
  signal confl_s13,confl_s23,...
    confl_s21,confl_s31,confl_s41:
    std_logic;
begin
  ...
end;
```

Selektor- und Konfliktmatrix

- Jeder Ausgang A_i selektiert einen Eingang mit einer eigenen Adresse $ADDR(i)$.
- Die Ausgangssignale SEL bestimmen die Zustände der Zweierschalter S_{ij} , jeweils nach Stufen j gruppiert $\{0,1,2\}$.
- Da das Schaltnetz nicht konfliktfrei ist, gibt es für jeden Ausgang i ein Konfliktsignal $CONFL(i)$.
- Ein Ausgangspfad muß über $ACT(i)$ aktiviert werden.
- Die Hilfssignale sel_low und sel_high geben für jeden Pfad den Eingangsbereich an:
low= $E\{000,001,100,010\}$
high= $E\{101,011,110,111\}$
- Weitere Bereichseinteilung:
low1= $E\{000,001\}$
low2= $E\{100,010\}$ usw.

Netzwerktopologien :: Permutationsnetze (VIII)

Selektoren der letzten (3.) Stufe

```
sel_low(0) <= '1' when
  ACT(0)='1' and (
    ADDR0 = "000" or
    ADDR0 = "001" or
    ADDR0 = "100" or
    ADDR0 = "010") else '0';
sel_high(0) <= '1' when
  ACT(0)='1' and (
    ADDR0 = "000" or
    ADDR0 = "001" or
    ADDR0 = "100" or
    ADDR0 = "010") else '0';
...
confl_s13 <=
  (sel_low(0) and sel_low(4)) or
  (sel_high(0) and sel_high(4));
...
SEL2(0) <= sel_high(0) or sel_low(4);
...
```

- Die Zweierschalter S der letzten Stufe $\{S_{13}S_{23}S_{33}S_{43}\}$ selektieren den Eingangsbereich *low* oder *high*.
Im durchgeschalteten Zustand selektiert der obere Ausgang von S ➤ *low*, der untere ➤ *high*.
- Ein Konflikt tritt auf, wenn der obere und untere Ausgang von S den gleichen Eingangsbereich selektieren.

Netzwerktopologien :: Permutationsnetze (IX)

Selektoren der mittleren (2.) Stufe

```
sel_low1(0) <= '1' when
  ACT(0) = '1' and (
    ADDR0 = "000" or
    ADDR0 = "001") else '0';
sel_low2(0) <= '1' when
  ACT(0) = '1' and (
    ADDR0 = "100" or
    ADDR0 = "010") else '0';
sel_high1(0) <= '1' when
  ACT(0) = '1' and (
    ADDR0 = "101" or
    ADDR0 = "011") else '0';
sel_high2(0) <= '1' when
  ACT(0) = '1' and (
    ADDR0 = "110" or
    ADDR0 = "111") else '0';
...
SEL1 (0) <= sel_low2(0) or
  sel_low1(2) or
  sel_low2(4) or
  sel_low1(6);
...
```

- Die Zweierschalter S der mittleren Stufe $\{S_{12}S_{22}S_{32}S_{42}\}$ selektieren den Eingangsbereich *low1,low2* oder *high1,high2*.
- Ein Konflikt tritt auf, wenn der obere und untere Ausgang von S den gleichen Eingangsbereich selektieren.

```
confl_s12 <= '1' when (
  (sel_low1(0) and sel_low1(2)) or
  (sel_low2(0) and sel_low2(2)) or
  (sel_low1(4) and sel_low1(6)) or
  (sel_low2(4) and sel_low2(6))
) = '1' else '0';
...
```

Netzwerktopologien :: Permutationsnetze (X)

Selektoren der ersten (1.) Stufe

```

sel_s11_low(0) <= '1' when
  ACT(0)='1' and ADDR0 = "000"
  else '0';
sel_s11_low(1) <= '1' when
  ACT(0)='1' and ADDR1 = "000"
  else '0';
...
sel_s11_high(0) <= '1' when
  ACT(0)='1' and ADDR0 = "001"
  else '0';
...
SEL0(0) <=
  sel_s11_high(0) or sel_s11_low(1) or
  sel_s11_high(2) or sel_s11_low(3) or
  sel_s11_high(4) or sel_s11_low(5) or
  sel_s11_high(6) or sel_s11_low(7);
...
confl_s11 <= '1' when
  (sel_s11_low and sel_s11_high)
  /= "00000000"
  else '0';
confl_s21 <= '1' when
  (sel_s21_low and sel_s21_high)
  /= "00000000"
  else '0';
...

```

Seite 111

- ▶ Die Zweierschalter S der ersten Stufe $\{S_{11}S_{21}S_{31}S_{41}\}$ selektieren die einzelnen Eingänge.
- ▶ Ein Konflikt tritt auf, wenn der obere und untere Ausgang von S den gleichen Eingang selektieren.

```

CONFL(0) <= confl_s13 or
  ((sel_s11_low(0) or
  sel_s11_high(0)) and
  confl_s11) or
  ((sel_s21_low(0) or
  sel_s21_high(0)) and
  confl_s21) or
  ((sel_s31_low(0) or
  sel_s31_high(0)) and
  confl_s31) or
  ((sel_s41_low(0) or
  sel_s41_high(0)) and
  confl_s41) or
  (sel_s12 and confl_s12) or
  (sel_s32 and confl_s32);
...

```

Dr. Stefan Bosse • Hardware-Entwurf von parallelen Systemen

Netzwerktopologien :: Permutationsnetze (XI)

Logik-Synthese-Ergebnis

Cell	Library	References	Total Area
inv_x1	sxlib	54 x 1	54 gates
na2_x1	sxlib	14 x 1	18 gates
na3_x1	sxlib	16 x 2	27 gates
na4_x1	sxlib	28 x 2	56 gates
nao22_x1	sxlib	12 x 2	24 gates
nao2o22_x1	sxlib	6 x 2	14 gates
no2_x1	sxlib	2 x 1	3 gates
no3_x1	sxlib	20 x 2	34 gates
no4_x1	sxlib	16 x 2	32 gates
noa22_x1	sxlib	8 x 2	16 gates
noa2a22_x1	sxlib	2 x 2	5 gates
noa2ao222_x1	sxlib	4 x 2	9 gates
o2_x2	sxlib	2 x 2	3 gates
o3_x2	sxlib	4 x 2	8 gates
oa2a22_x2	sxlib	2 x 3	6 gates
on12_x1	sxlib	6 x 2	10 gates

- ▶ Im Vergleich zur eigentlichen Schaltmatrix benötigt der Selektor- und Konfliktgenerator gleich großen Ressourcenbedarf!
- ▶ Kontrollogik von konflikt-behafteten Netzen ist komplex!
- ▶ Aber: Mikrokod-Implementierung von Σ und Γ ohne zusätzlichen Hardware-Aufwand möglich ▶ FSM.

Äquivalentgatter
319
Längster Pfad
3.8 ns

Seite 112

Dr. Stefan Bosse • Hardware-Entwurf von parallelen Systemen

Netzwerktopologien :: Ringe und Arrays

Eindimensionale statische Netzstruktur

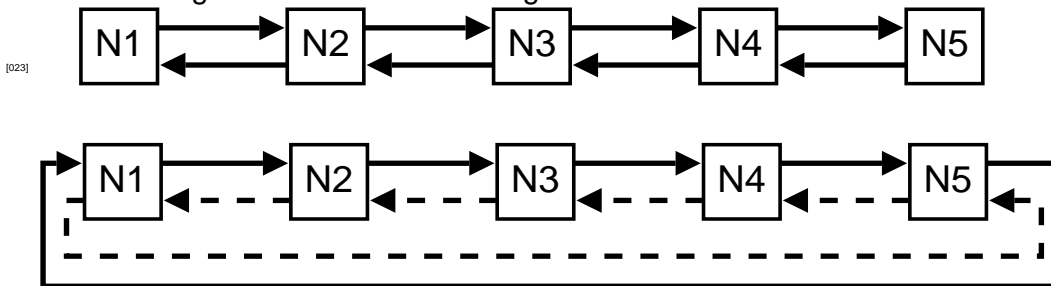
Lineares Array (Kette)

- N Kommunikationsknoten linear in einer Reihe angeordnet.
- Ausdehnung: $N-1$
- Grad (Anzahl der Nachbarn): $d=2$
- Routing: wähle Richtung und laufe "geradeaus".
- Skalierbar, alle Kommunikationskanäle können gleichzeitig arbeiten.
- Nur paket- und nachrichtenbasierte Kommunikation möglich.

Zweidimensionale statische Netzstruktur

Ring (geschlossene Kette)

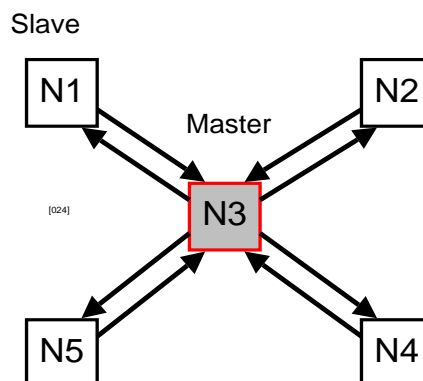
- wie lineares Array
- Ausdehnung: ➤ bidirektionaler Ring: $N/2$
- Ausdehnung: ➤ unidirektionaler Ring: $N-1$



Netzwerktopologien :: Stern

Master-Slave-Netzwerk

- Request-Reply Nachrichten: Ein Request wird an einen Slave-Knoten gesendet, der dann ein Reply an den Master zurücksendet. Die Slaves sind passiv, nur der Master ist aktiv.
- Ein ausgezeichneter Knoten arbeitet als Master. Bei der Stern-Struktur ist dies der zentrale Knoten.
- Alle umgebenden Knoten arbeiten als Slave. Ein Slave-Knoten darf direkt keine Nachrichten versenden.
- Das Routing findet über den zentralen Master-Knoten statt.
- Grad $d=2$ [Slave], $(N-1)*2$ [Master], Ausdehnung: 2



Synchronisation :: Lock

Algorithmus für ein Mutual-Exclusion-Lock

```
TYPE node = {
  ↑TYPE node next;
  BOOL locked;
};
TYPE node Lock, Prev;
◆
PRO lock (Lock, mynode) IS
  mynode.next ← NONE;
  ↪(1)
  Prev ← F&S(Lock, mynode);
  ↪(2)
  IF Prev ≠ NONE THEN
    mynode.locked ← TRUE;
    Prev.next ← mynode;
    WAITFOR mynode.locked=FALSE;
  END IF;
```

(1)

Lock zeigt aktuell auf vorheriges Endelement der Liste. Atomares Setzen von links nach rechts:

Prev ← Lock ← mynode

(2)

Andere Prozesse sind in der Liste und ein anderer Prozess hält den Lock. Einfügen von mynode am Anfang der Liste.

↪ F&S: fetch&store

Literatur

[UNG97]

T. Ungerer
Parallelrechner und parallele Programmierung
Spektrum, 1997

[TOK03]

M.O. Tokhi, M.A. Hossain, M. H. Shaheed
Parallel Computing for Real-time Signal Processing and Control
Springer, 2003

[CUL02]

D. E. Culler, J. P. Singh
Parallel Computer Architecture - A Hardware/Software Approach
Elsevier, 2002

[HWA86]

K. Hwang, F. A. Briggs
Computer Architecture and Parallel Processing
McGrawHill, 1986

Literatur

[HAC06]

G. D. Hachtel, F. Somenzi
Logic Synthesis and Verification Algorithms
Springer, 2006

[NAT96]

L. Natvig
Evaluating Parallel Algorithms: Theoretical and Practical Aspects
Thesis, University of Trondheim

[CHO90]

A. N. Choudhary, J. H. Patel
Parallel Architectures and Parallel Algorithms for Integrated Vision Systems
Kluwer Academic, 1990

[JER05]

A. A. Jerraya, W. Wolf (Ed.)
Multiprocessor Systems-on-Chips
Morgan-Kaufmann, 2005