KV Computer Microsystems

Teil III: Synthese

Andreas Koch

FG Eingebettete Systeme und ihre Anwendungen
Informatik, TU Darmstadt

Sommersemester 2005

Organisatorisches

Dozent
Andreas Koch koch@esa.informatik.tu-darmstadt.de
Sprechstunde Mi 14:00-15:00, E103

Wissenschaftlicher Mitarbeiter
Holger Lange lange@esa.informatik.tu-darmstadt.de
Sprechstunde Mi 14:00-15:00, E106

Web-Site
`http://www.esa.informatik.tu-darmstadt.de`

Material

Grundlage der Vorlesung:

Synthesis and Optimization of Digital Circuits
von Giovanni De Micheli, McGraw-Hill, 1994

Teil I

Einleitung

Gliederung

Einordnung

Modelle

- Abstrakt: Petri-Netze
- Konkreter: FSMD

Beschreibungen

- Sprache: VHDL

Synthese

Transformation von Modellen und Beschreibungen

Thematik

- Modellierung mit Hilfe von Abstraktionen und Sichten
- Syntheseverfahren
- Optimierungen
- Hardware-Strukturen

Hardware-Bezug

. . . aber keine Vorlesung im Schaltungsentwurf!

Abstraktion

Abstrahieren

Weglassen von *momentan* unnötigen Informationen

Abstraktion auf Architekturebene

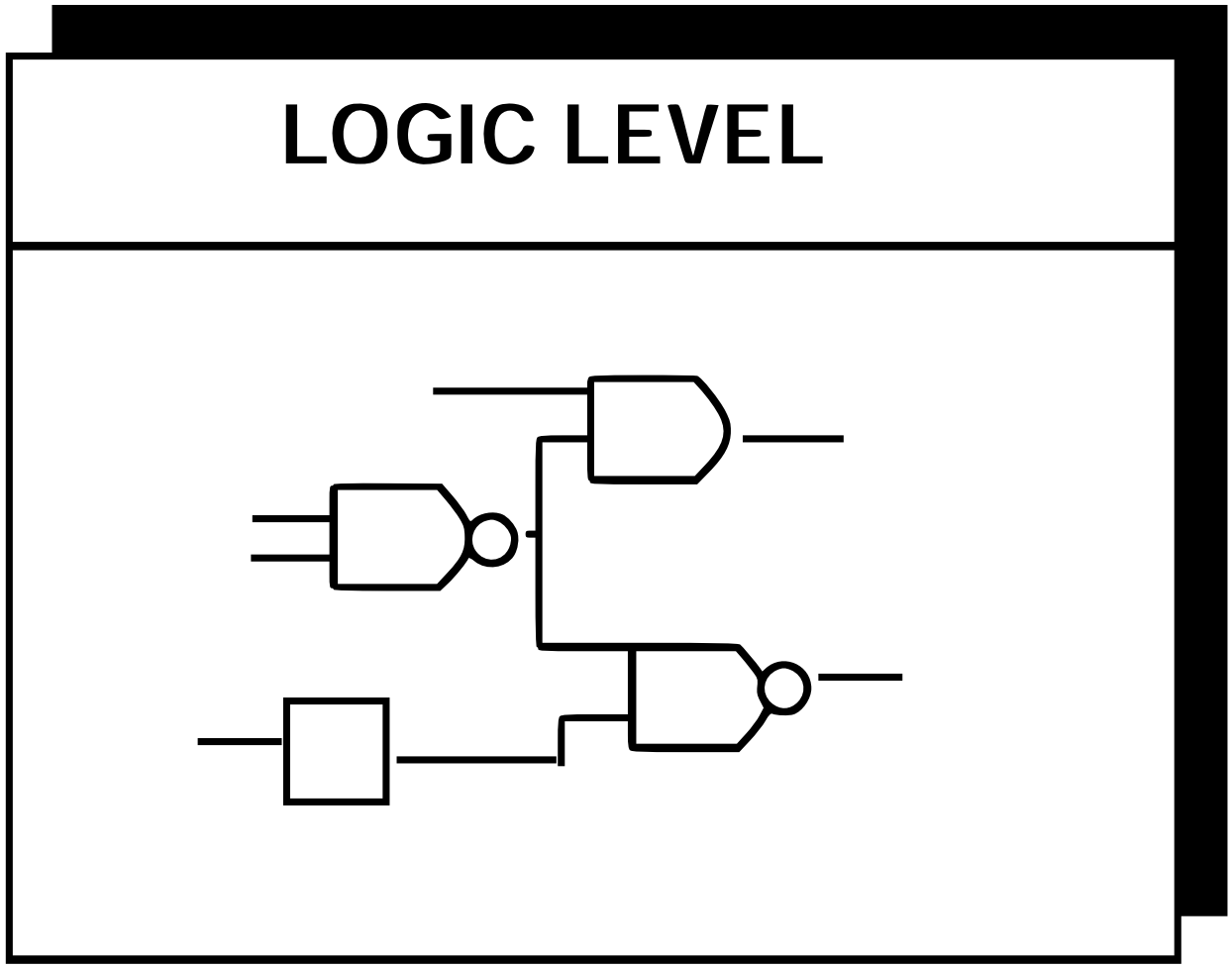
Abstrakte Operationen und Ressourcen

ARCHITECTURAL LEVEL

```
. . .
PC = PC + 1;
FETCH (PC);
DECODE (INST);
. . .
```

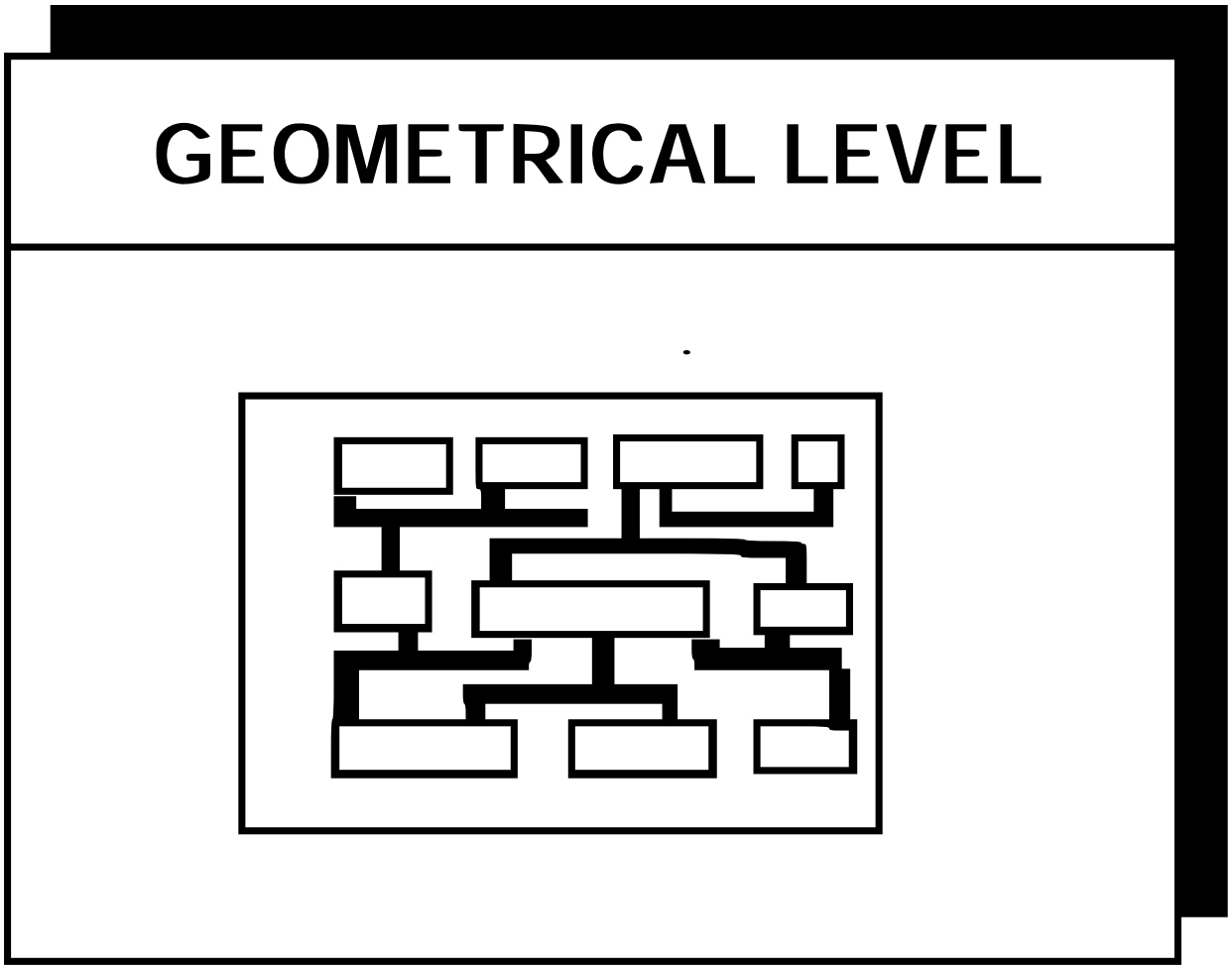
Abstraktion auf Logikebene

Logische Funktionen aus Gattern



Abstraktion auf geometrischer Ebene

Interagierende geometrische Objekte in 2-D oder 3-D



Sichten

Verhalten

- Funktion
- *Ohne* Beschreibung der Realisierung

Struktur

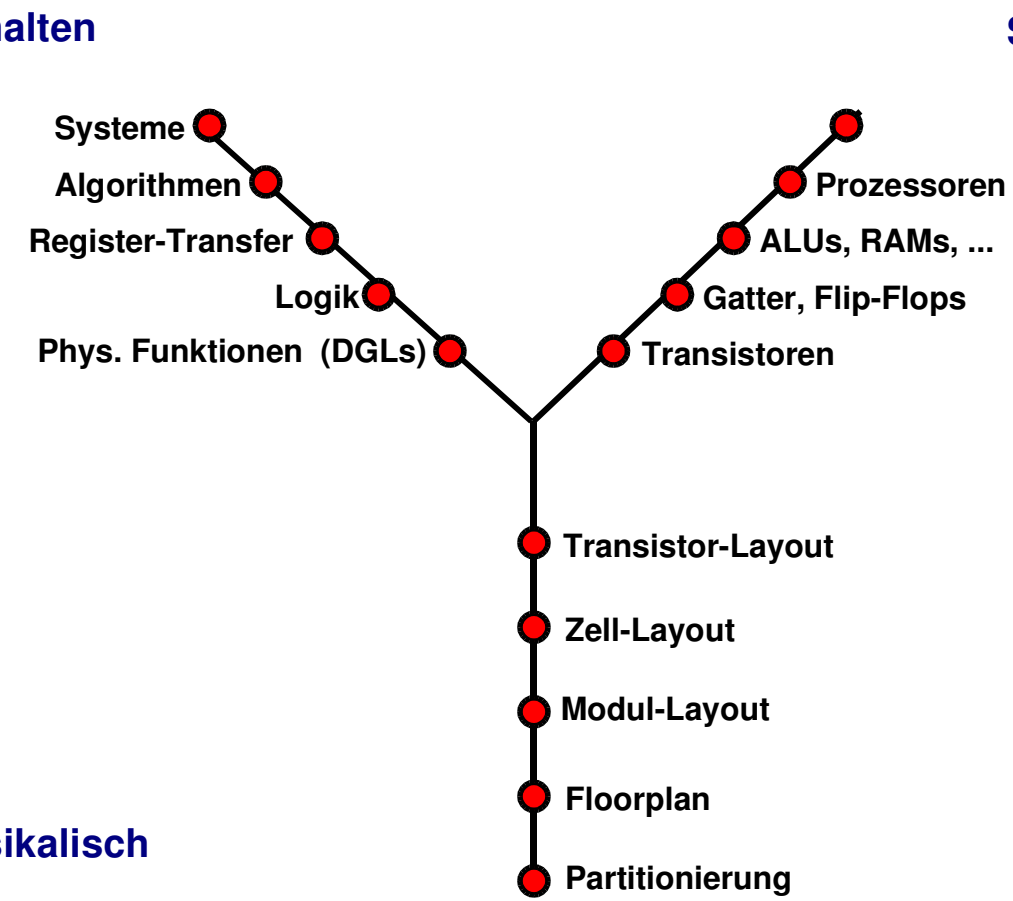
- Elemente
- Verbindungen

Physikalisch

- Reale Objekte
- Position und Abmessungen

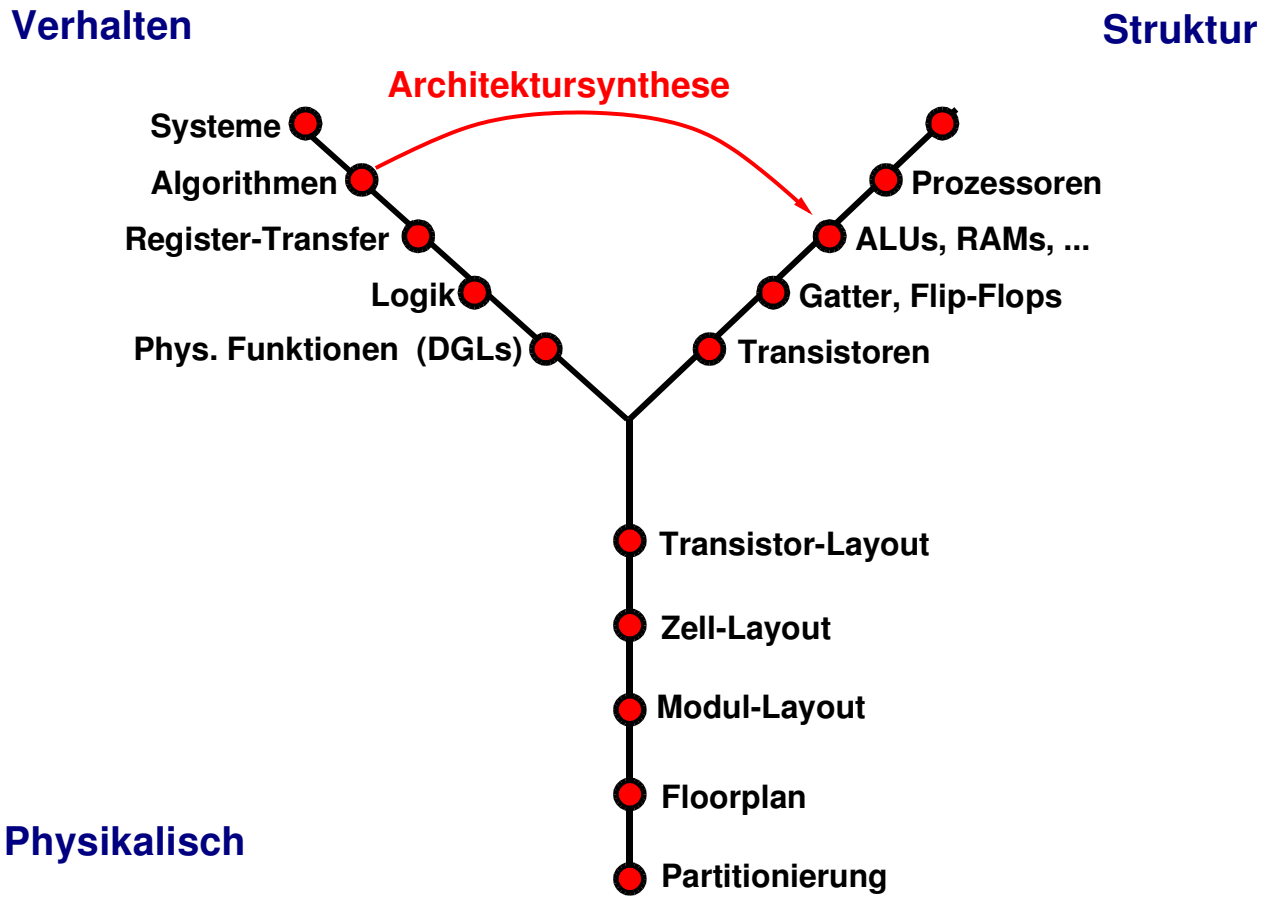
Vereinigung

Gajskis Y-Diagramm



Architektursynthese

Bildet Verhalten von Architekturebene auf Strukturen größerer Blöcke ab



Ablauf der Architektursynthese

Eingabe Verhaltensmodell

- Operationen
- Abhängigkeiten

Ausgabe Strukturelle Sicht

Datenpfad Verbundene Hardware-Ressourcen

Steuerwerk Auf Logikebene

= FSMD aus Teil II der VL

Beispiel: Lösung der DGL $y'' + 3xy' + 3y = 0$

Euler-Verfahren
Intervall $[0, a]$, Schrittweite dx, $x(0) = x$, $y(0) = y$, $y'(0) = u$

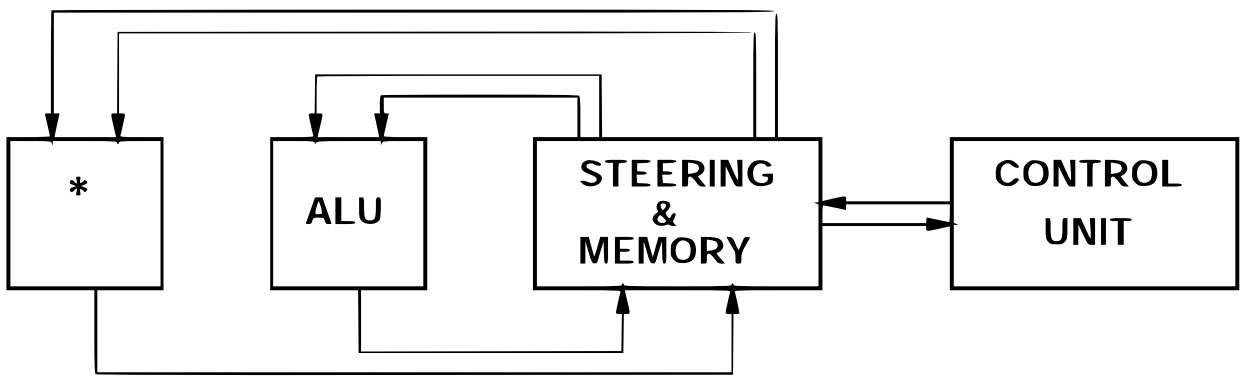
```
diffeq{
  read (x, y, u, dx, a);
  repeat {
    x1 = x + dx;
    u1 = u - (3 * x * u * dx) - (3 * y * dx);
    y1 = y + u * dx;
    c  = x1 < a;
    x  = x1; u = u1; y = y1;
  } until (c);
  write (y);
}
```

Synthetisierte Architektur: Strukturmodell

Eine Möglichkeit

Annahmen

- 1 Multiplizierer
- 1 ALU (+,-,<)
- 1 Speicher

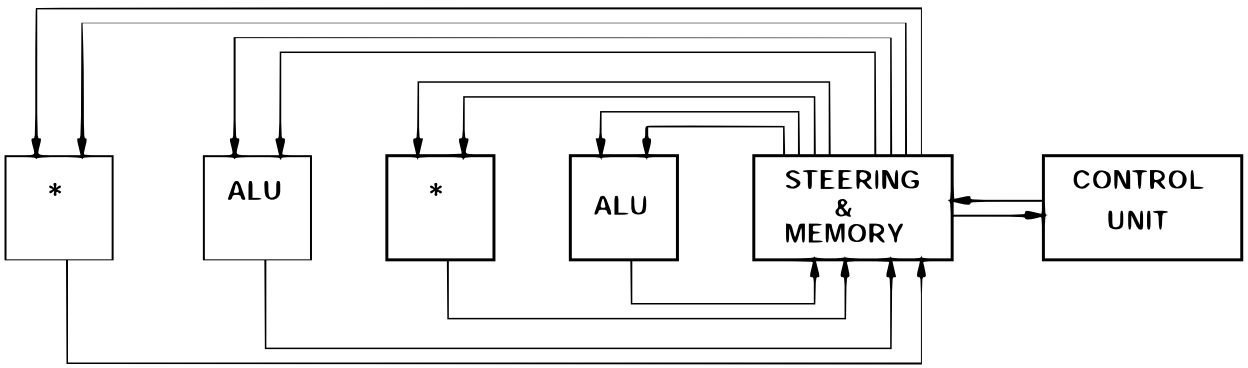


Synthetisierte Architektur: Strukturmodell

Eine *weitere* Möglichkeit

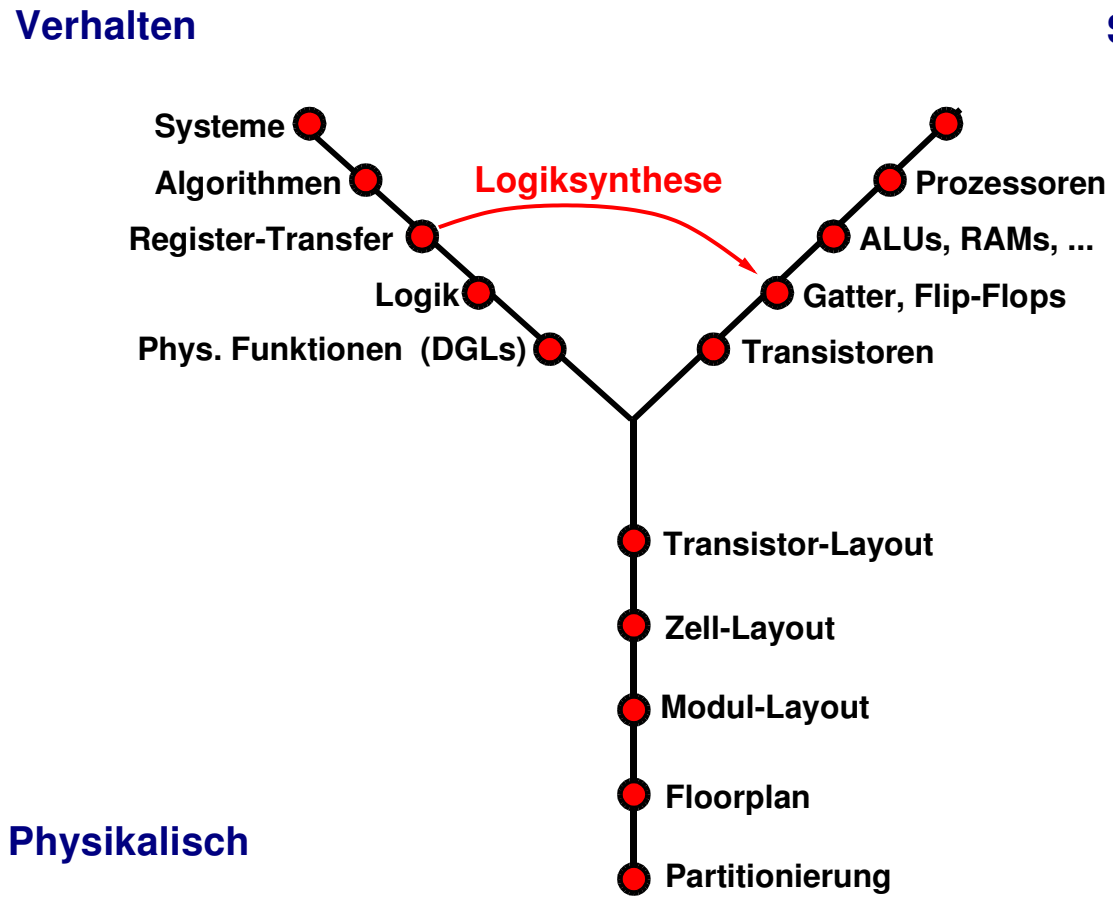
Annahmen

- 2 Multiplizierer
- 2 ALUs (+,-,<)
- 1 Speicher



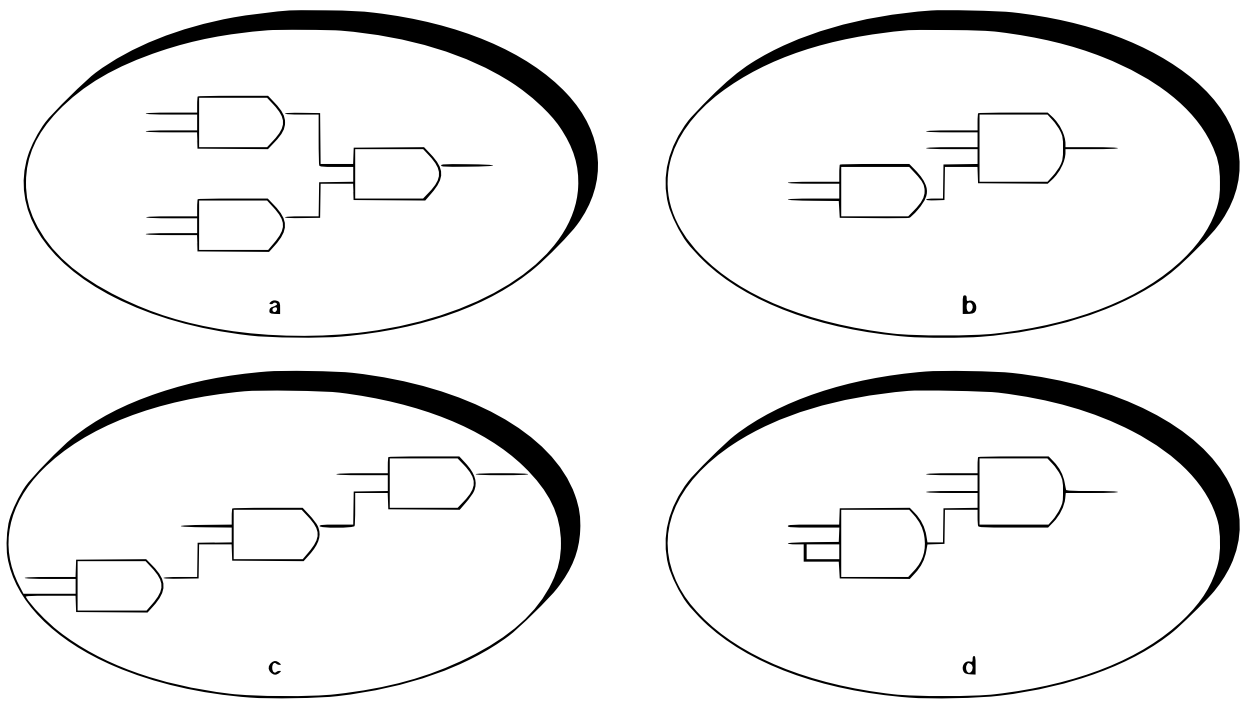
Logiksynthese

Bildet Verhalten von Logikebene auf Gatterstrukturen ab



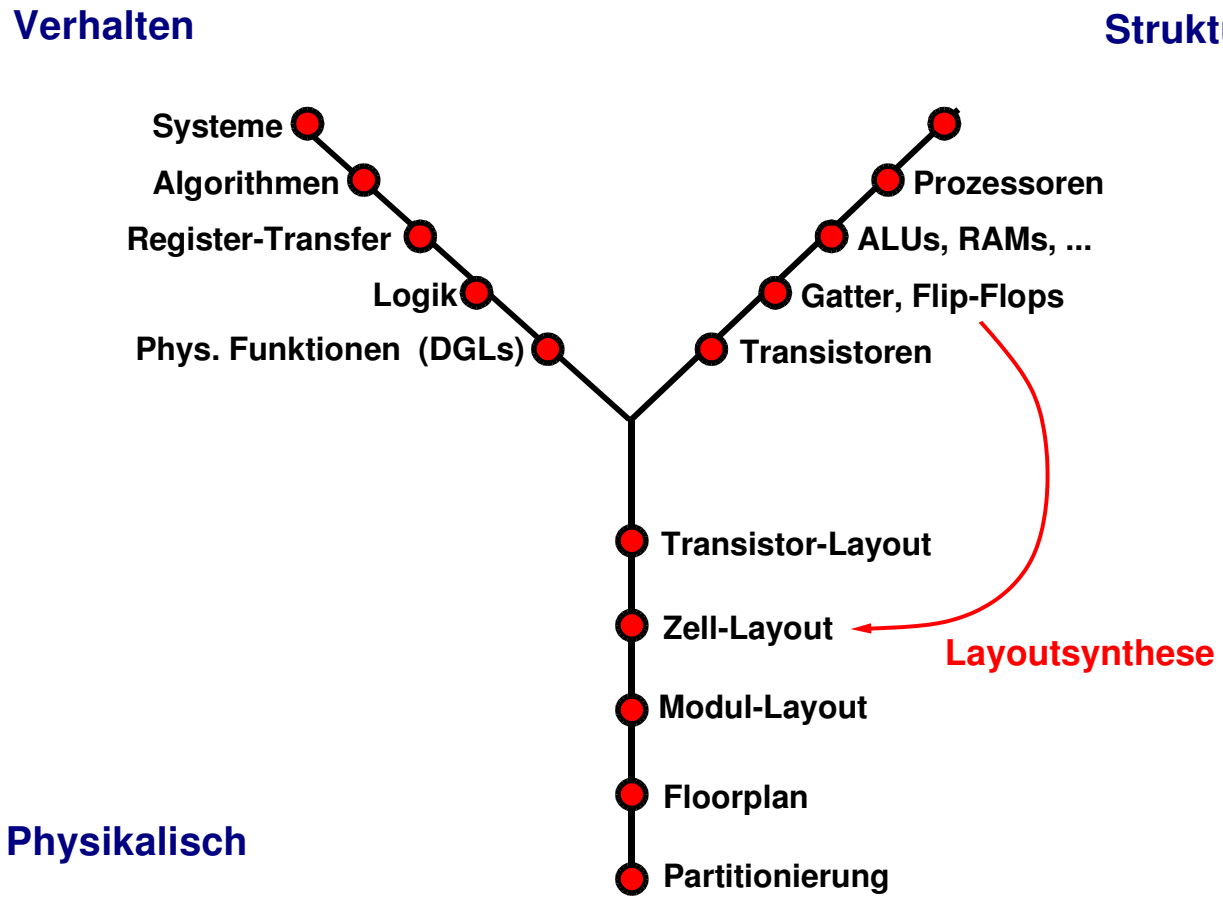
Beispiel: $f = pqrs$

Abbildung auf AND-Gatter mit 2 und 3 Eingängen

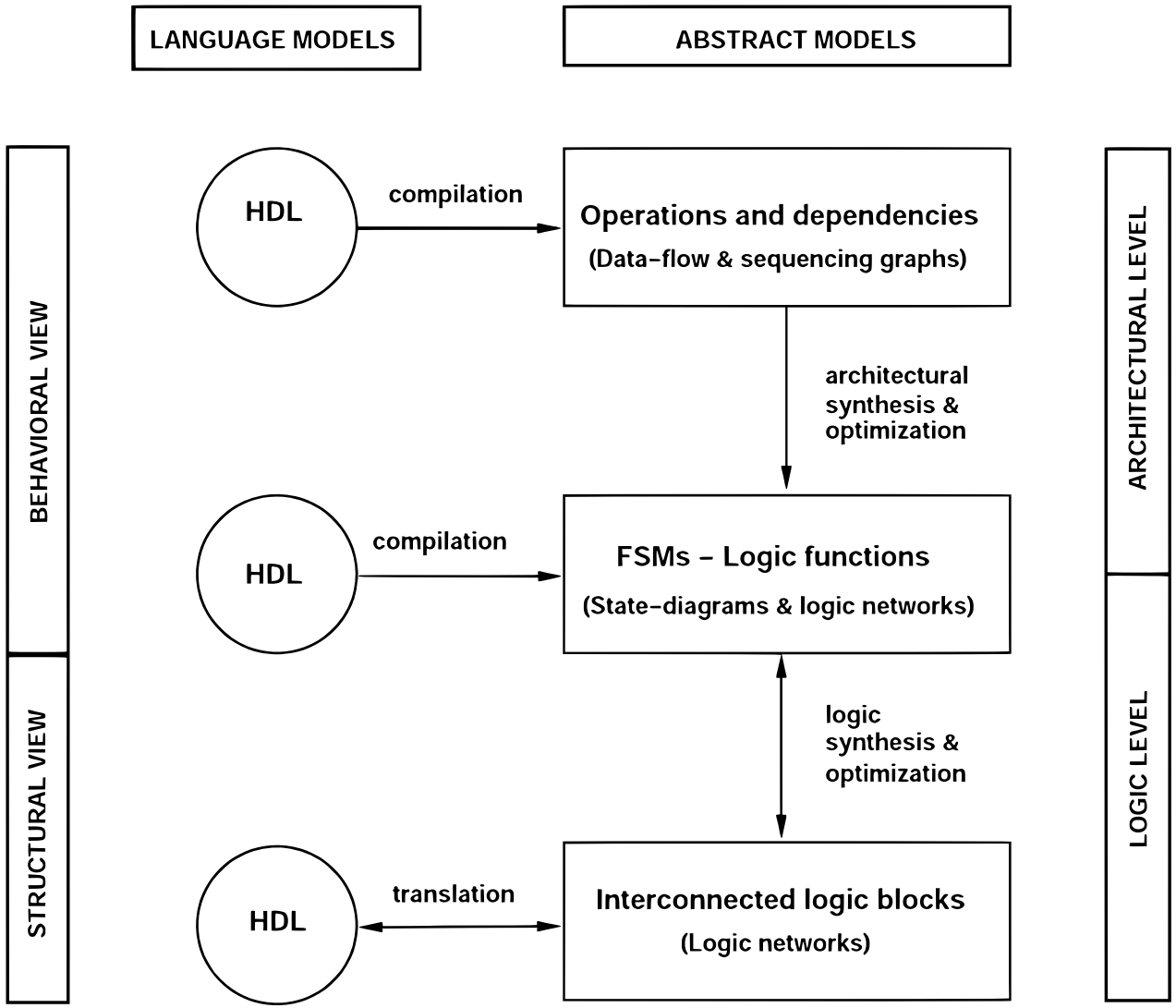


Layoutsynthese

Bildet Gatterstrukturen auf geometrische Objekte ab



Übersicht über Synthesefluss



Optimierungsziele

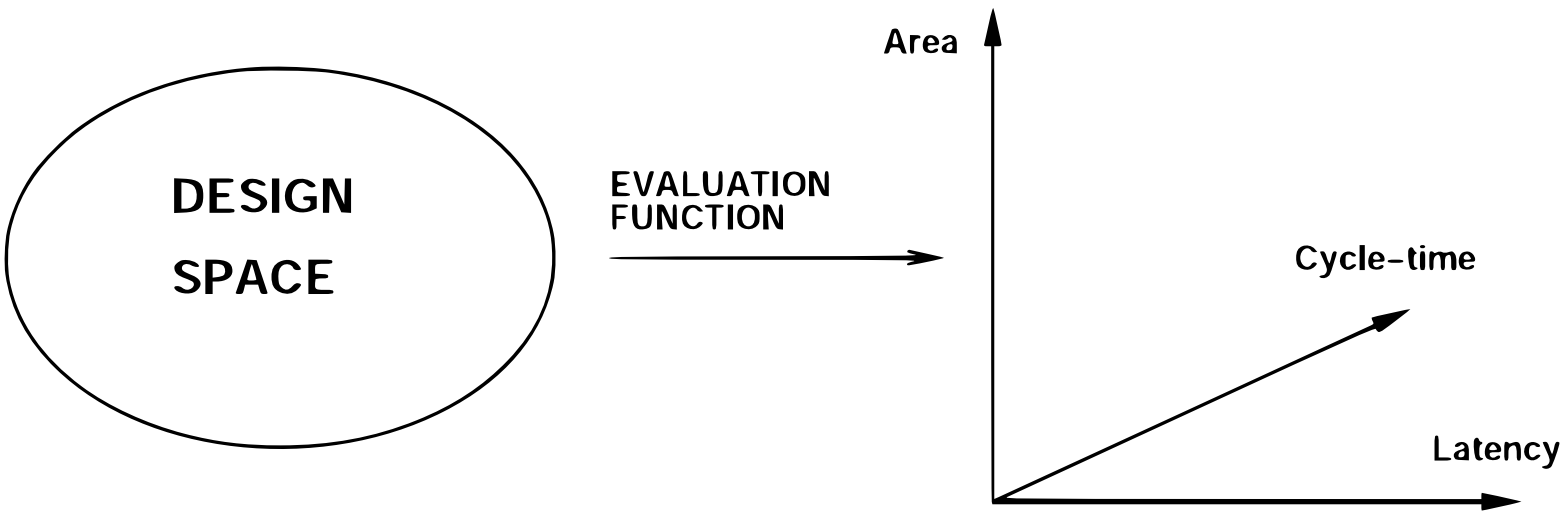
Optimierung nach mehreren Kriterien

- Rechenleistung
  - Verzögerungszeiten und Taktperiode
  - Latenz
  - Durchsatz (bei Pipelining)
- Energieverbrauch und max. Leistungsaufnahme
- Fläche und Herstellbarkeit (Gehäuse)
- Testbarkeit

Bewertung des Entwurfsraumes

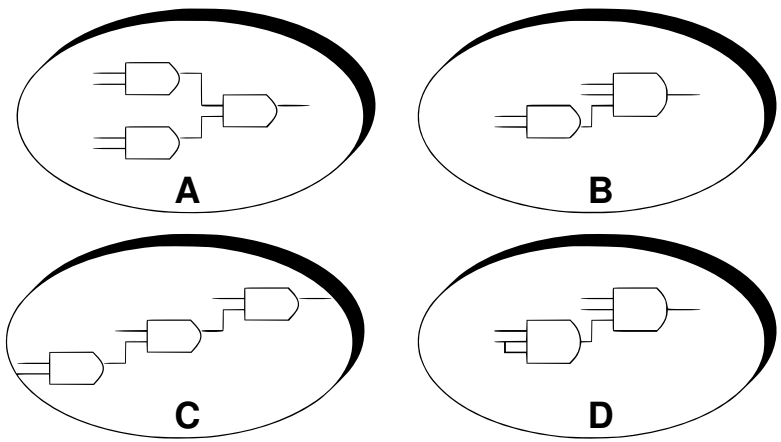
Entwurfsraum Raum *aller* Realisierungsmöglichkeiten

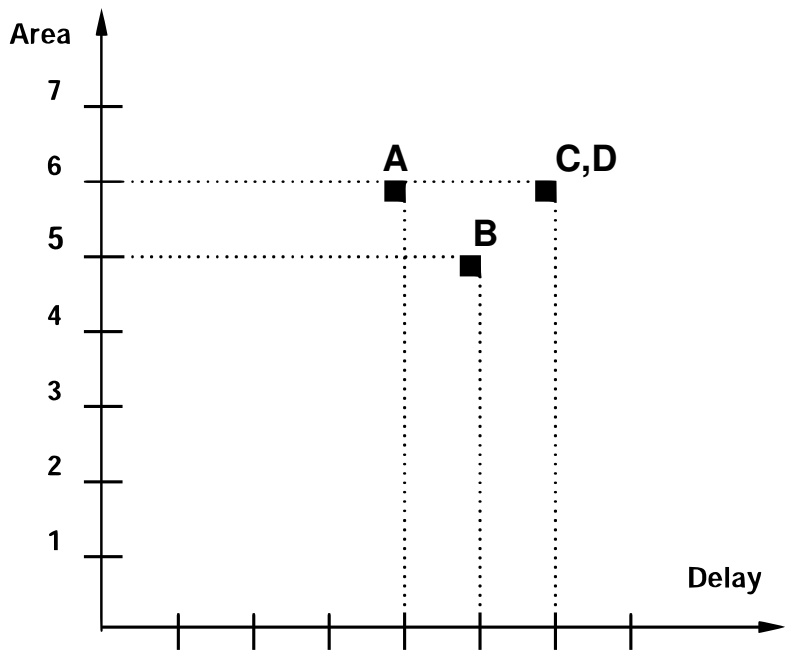
Bewertungsfunktion Liefert multi-dimensionale Charakterisierung einer konkreten Realisierung



Beispiel: $f = pqrs$

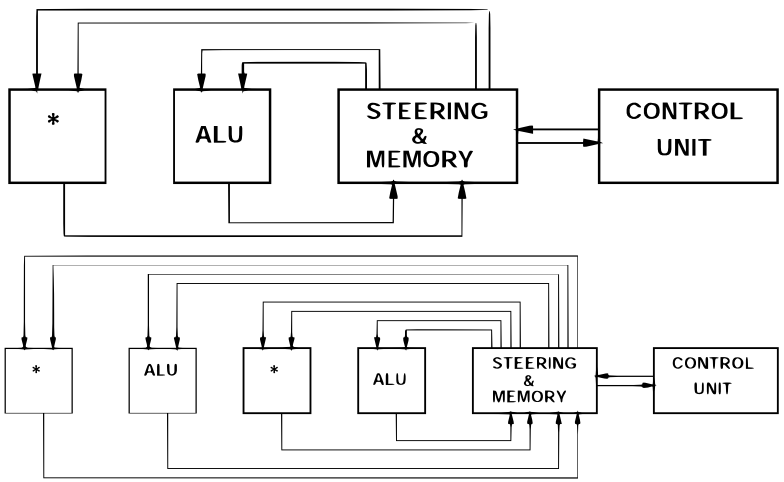
Name	Fläche	Verzögerung
AND2	2	2
AND3	3	3

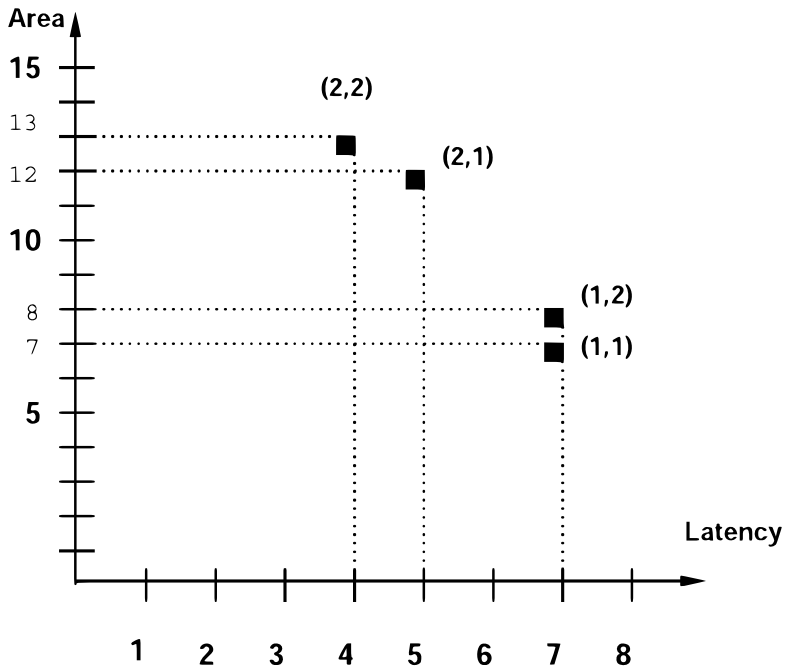




Beispiel: `diffeq()`

Name	Fläche	Verzögerung
Mult	5	1
ALU	1	1
Mem	1	0





(n,m) = (#Mult, #ALU)

Pareto-Punkte

Optimierung mehrerer Kriterien

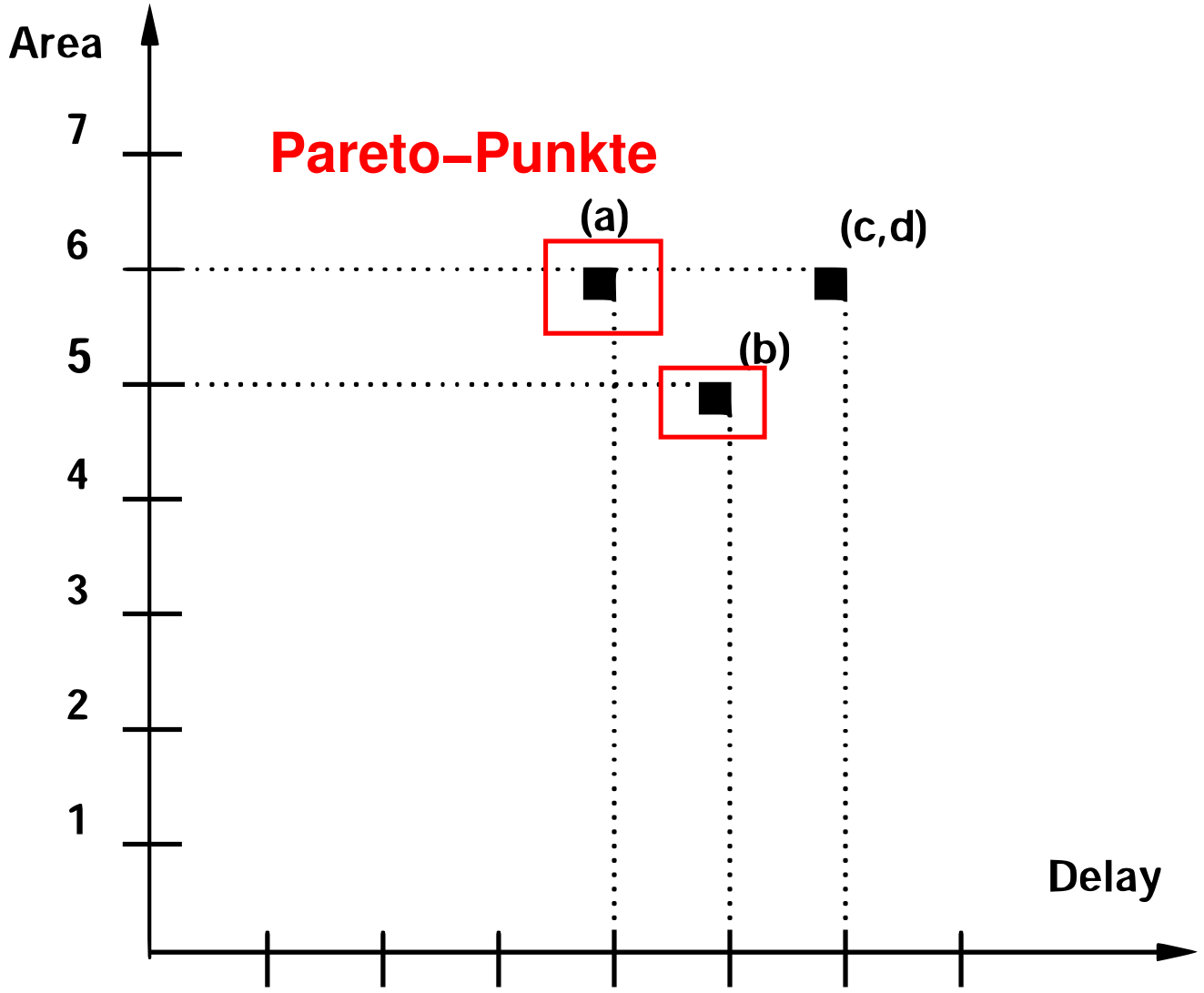
Definition

Ein Punkt $x \in \mathbf{R}^n$ ist ein *Pareto-Punkt* genau dann, wenn es keinen Punkt $y \in \mathbf{R}^n$ gibt, der $y_i \leq x_i, 1 \leq i \leq n$, mit mindestens einer echten Ungleichheit, hat.

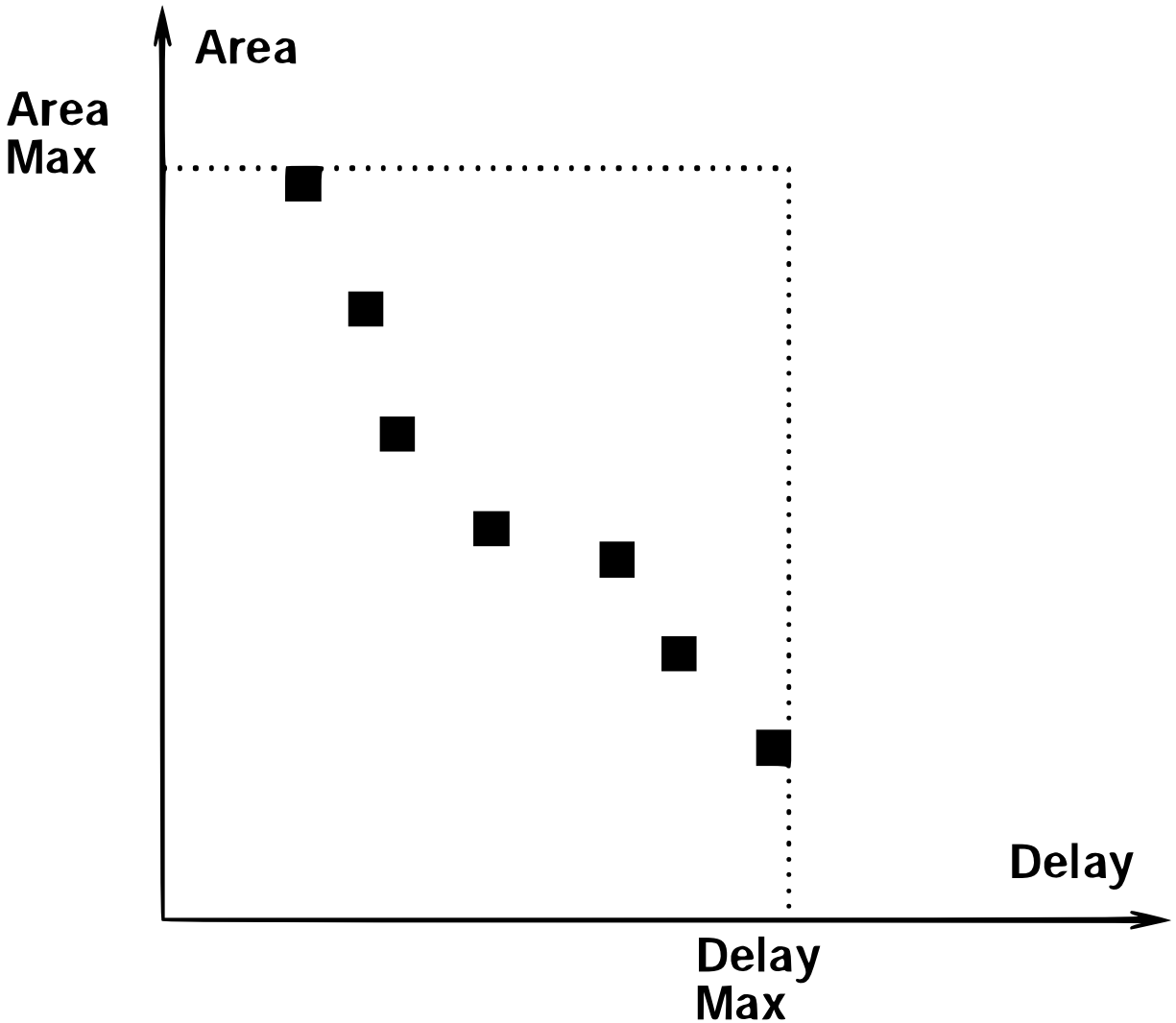
Pareto-Punkt

Entwurfspunkt, welcher durch einen anderen in keinem Kriterium verbessert werden kann, ohne dass dieser in mindestens einem anderen Kriterium eine Verschlechterung erleidet.

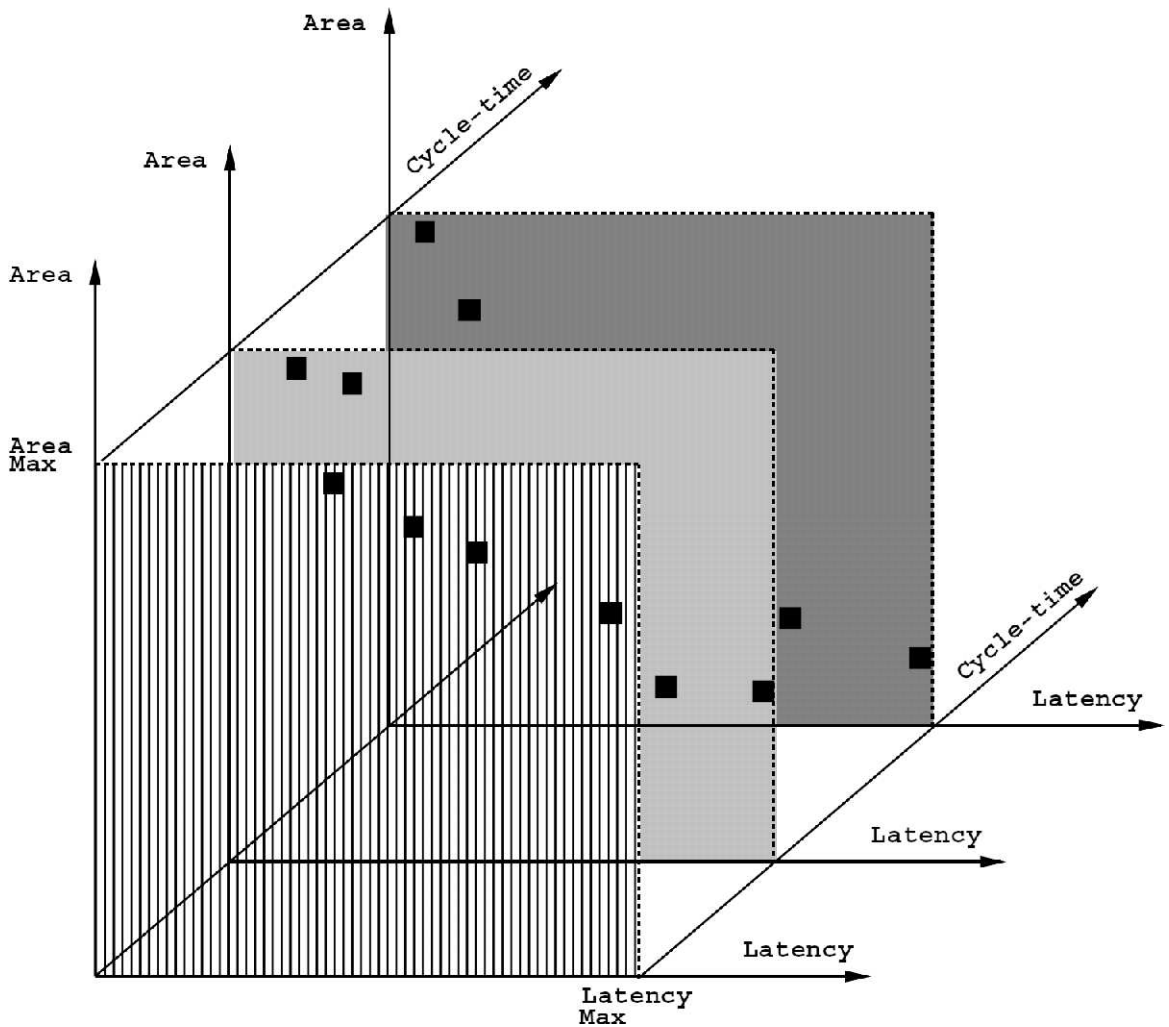
Beispiel: Pareto-Punkte



Bewertung kombinatorischer Schaltungen



Bewertung sequentieller Schaltungen



Höhere Entwurfsebenen

Höhere Produktivität im Hardware-Entwurf

Früher

- Polygone (Full-Custom Layout)
- Schaltpläne (Standardzellen)

Heute

- HDLs (Verilog, VHDL)

Zukunft (?)

- Hochsprachen (C, Java)
- Anwendungsgebiets-spezifisch (MATLAB)

Repräsentationen

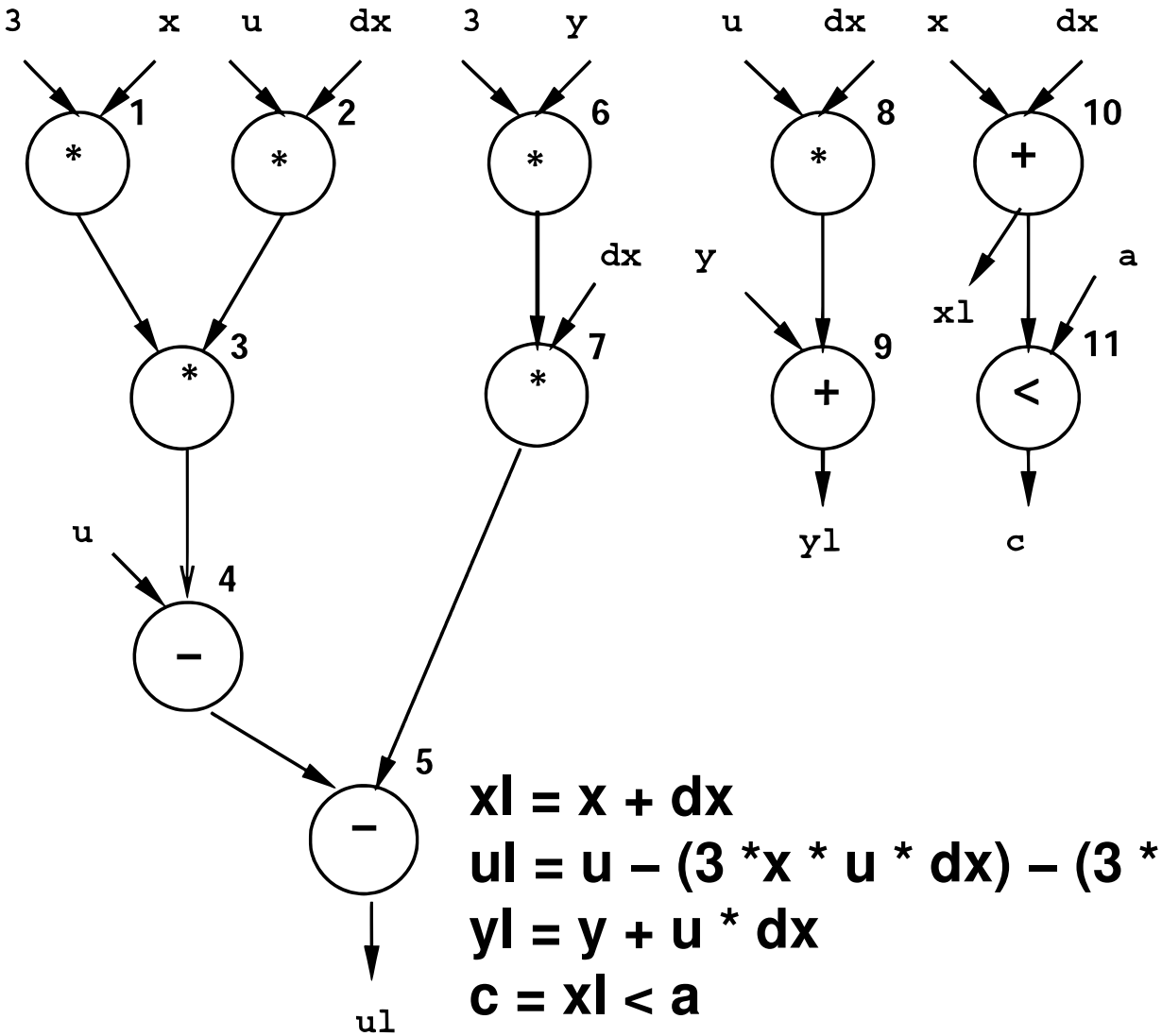
- Synthese findet *nicht* auf Syntax-Ebene statt
- Effiziente Zwischendarstellungen zur Compilierung
  - Endliche Automaten (FSMs, Teil II)
  - Datenflussgraphen (DFG)
  - Sequenzgraphen
  - Petri-Netz (Teil I)
- In der Regel graphenbasiert

Datenflussgraphen

- Verhaltenssicht auf Architekturebene
- Nützlich zur Darstellung von Datenpfaden (Teil II)

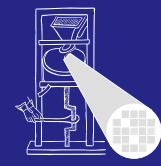
- Bestehen aus Knoten und gerichteten Kanten
  - Knoten Operationen
    - Brauchen $n \in \mathbf{N}_0$ Operanden
    - Liefern $m \in \mathbf{N}_0$ Ergebnisse
  - Kanten Datenabhängigkeiten
    - Zwischen Operationen
    - Vom Produzenten von Daten zu Konsumenten
    - Leere Kanten bei Ein- und Ausgabe

Beispiel: Datenflussgraph

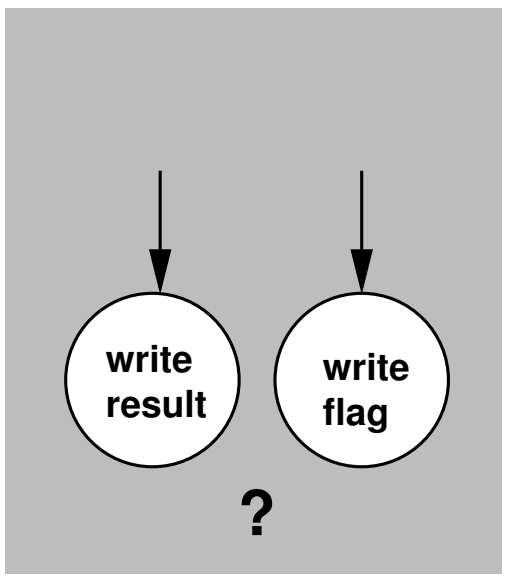


xl = x + dx
ul = u – (3 *x * u * dx) – (3 * y * dx)
yl = y + u * dx
c = xl < a

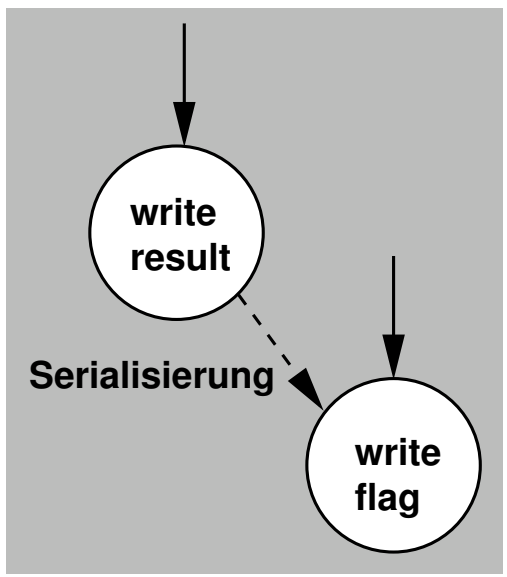
Kontrollflussdarstellung



- Im reinen DFG *keine*
  - Hierarchie (Unterfunktionsaufrufe)
  - Verzweigung
  - Wiederholung
- Auch keine *Serialisierung*
  - Zeitabhängigkeiten in der Ausführung



Reihenfolge undefiniert



Definierte Reihenfolge

Sequenzgraphen

Modellieren Daten *und* Kontrollfluss

- Modelliere durch Graphen
  - Datenfluss
  - Serialisierung
- Modelliere durch Hierarchie
  - Verzweigung
  - Wiederholung
  - Unterfunktionen

Aufbau von Sequenzgraphen

Graphenhierarchie aus Sequenzgraphentitäten $G_S = (V, E)$

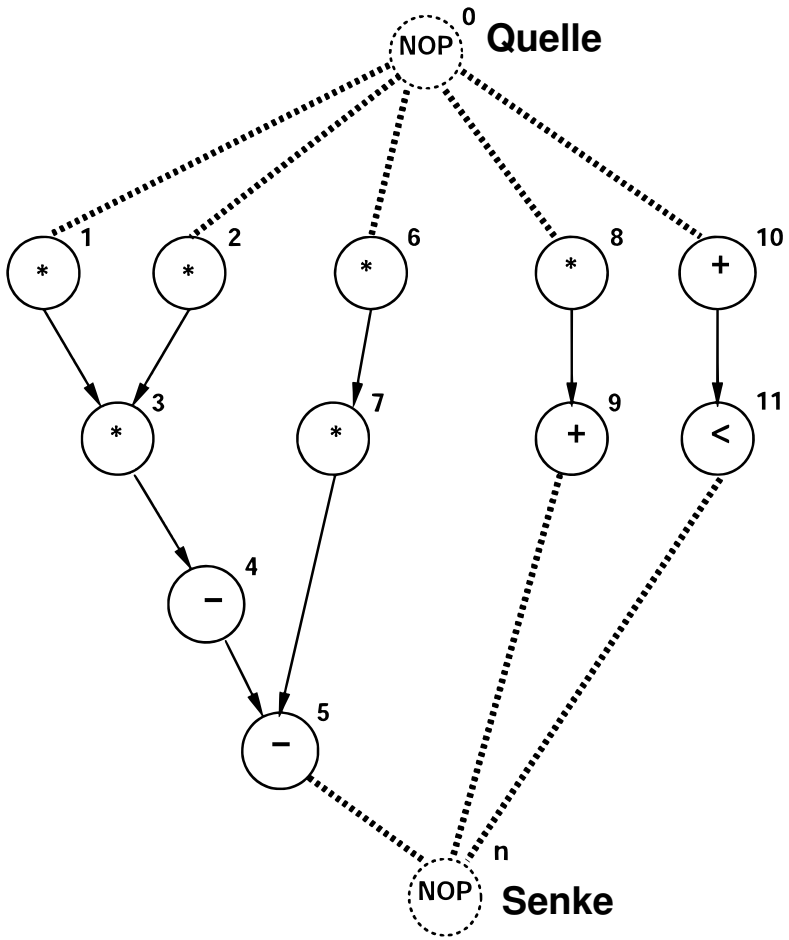
- V Knoten
  - Operationen
    - Wie DFG, aber jetzt auch Ein-/Ausgabe
  - Verkettungen (über Hierarchieebenen)
- E Gerichtete Kanten
  - Datenabhängigkeiten
  - Serialisierung

Eigenschaften

- Zyklenfrei (partiell geordnet)
- Polar: Quelle v_0 und Senke v_n
  - Modelliert als NoOp-Knoten

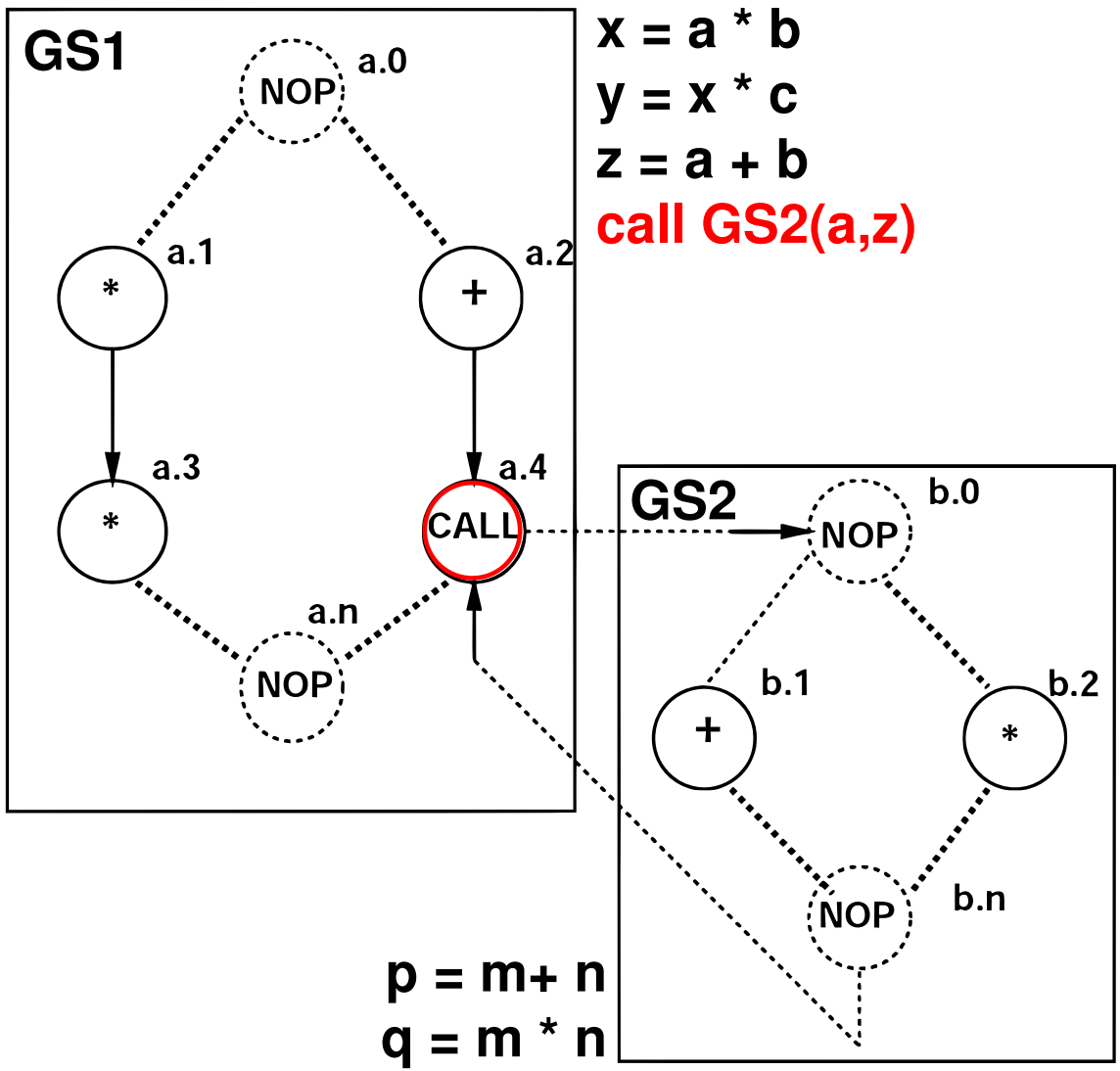
Vereinfachung Implizite Kanten für Ein-/Ausgabewerte

Beispiel: Flache Sequenzgraphentität

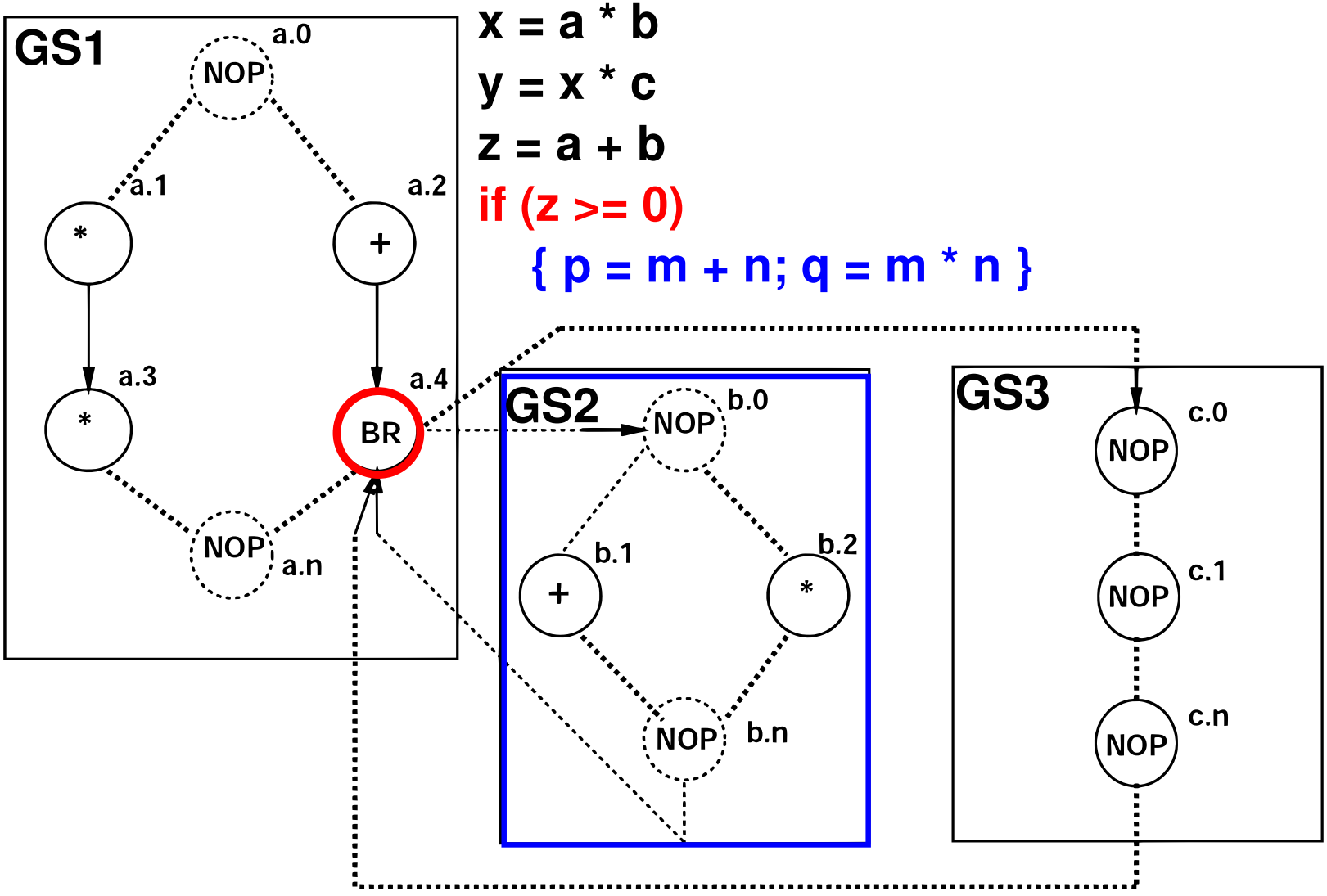


Parallele Ausführung von Pfaden

Beispiel: Unterfunktionsaufruf



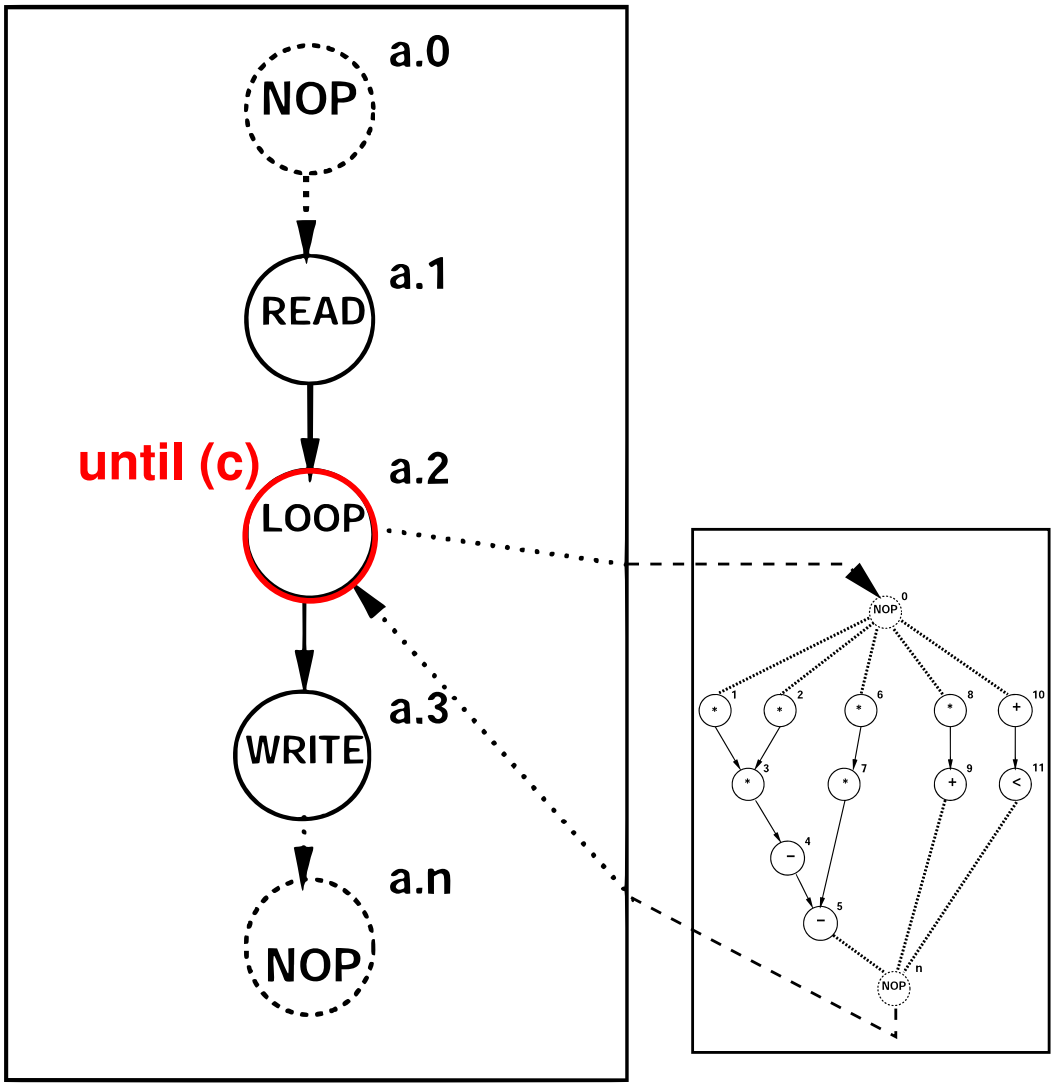
Beispiel: Verzweigung



Wiederholung in `diffeq()`


```
diffeq{
  read (x, y, u, dx, a);
  repeat {
    x1 = x + dx;
    u1 = u - (3 * x * u * dx) - (3 * y * dx);
    y1 = y + u * dx;
    c  = x1 < a;
    x  = x1; u = u1; y = y1;
  } until (c);
  write (y);
}
```

Beispiel: Wiederholung



Semantik von Sequenzgraphen

Markierung der Knoten

- Wartend auf Ausführung
- Ausführend
- Ausführung abgeschlossen

Feuern Mit der Ausführung beginnen

Semantik

Eine wartende Operation kann feuern, sobald alle ihre direkten Vorgänger ihre Ausführung abgeschlossen haben.

Reset Markiere alle Knoten als wartend

Starte Modell Durch Feuern der Quelle

Attributierte Sequenzgraphen

- Flächenbedarf
- Ausführungszeit
  - Datenunabhängig
  - Datenabhängig (z.B. Verzweigung, Wiederholung)
- Datenabhängige Ausführungszeit
  - Beschränkt (Verzweigungen)
  - Unbeschränkt (Wiederholung)
- Gesamtausführungszeit: Latenz
- Oft: Normierung mit Taktperiode, also Einheit „Takte“

Eigenschaften von Sequenzgraphen

Berechnet durch *bottom-up* Vorgehen

- Geschätzte Fläche
  - Aufaddieren des Flächenbedarfs aller Knoten
  - Annahme: Keine gemeinsame Nutzung (sharing)
- Geschätzte Ausführungszeit
  - Beschränkte Latenz
  - Länge der *längsten* Ausführungspfade

Teil II

Architektursynthese

Gliederung

Ablauf der Architektursynthese

1. Übersetze Eingabesprache in Sequenzgraphen
2. Optimiere Verhaltenssicht unabhängig von Implementierung
   - Ähnlich zu Software-Compiler
3. Synthese und Optimierung auf Architekturebene
   - Erzeuge grobe Struktur der Hardware-Recheneinheit
   - FSMD
   - Schätze Zeit, Fläche, etc. ab

Hardware-Modellierung

Schaltungsverhalten Sequenzgraphen

Bausteine Ressourcen

Randbedingungen Explizite Anforderungen

- Schränken Entwurfsraum ein
- Beispiele: Zeit- und Flächenbegrenzung

Ressourcen

Funktionale Ressourcen

- Führen Operationen auf Daten aus
- Beispiel: Arithmetische und logische Blöcke

Speicherressourcen

- Speichern Daten
- Beispiel: Speicherblöcke und Register

Schnittstellenressourcen

- Stellen Verbindungen zum Restsystem her
- Beispiel: Busse und Ports

. . . gut abschätzbar (Bibliothek, Modulgenerator)

Infrastruktur

Ressourcen alleine rechnen noch nicht!

- Verdrahtung
- Datenvermittlung (Multiplexer, NoC)
- Steuerwerke

. . . schlecht abschätzbar (Einzelsynthese erforderlich)

Charakterisierung von Schaltungen

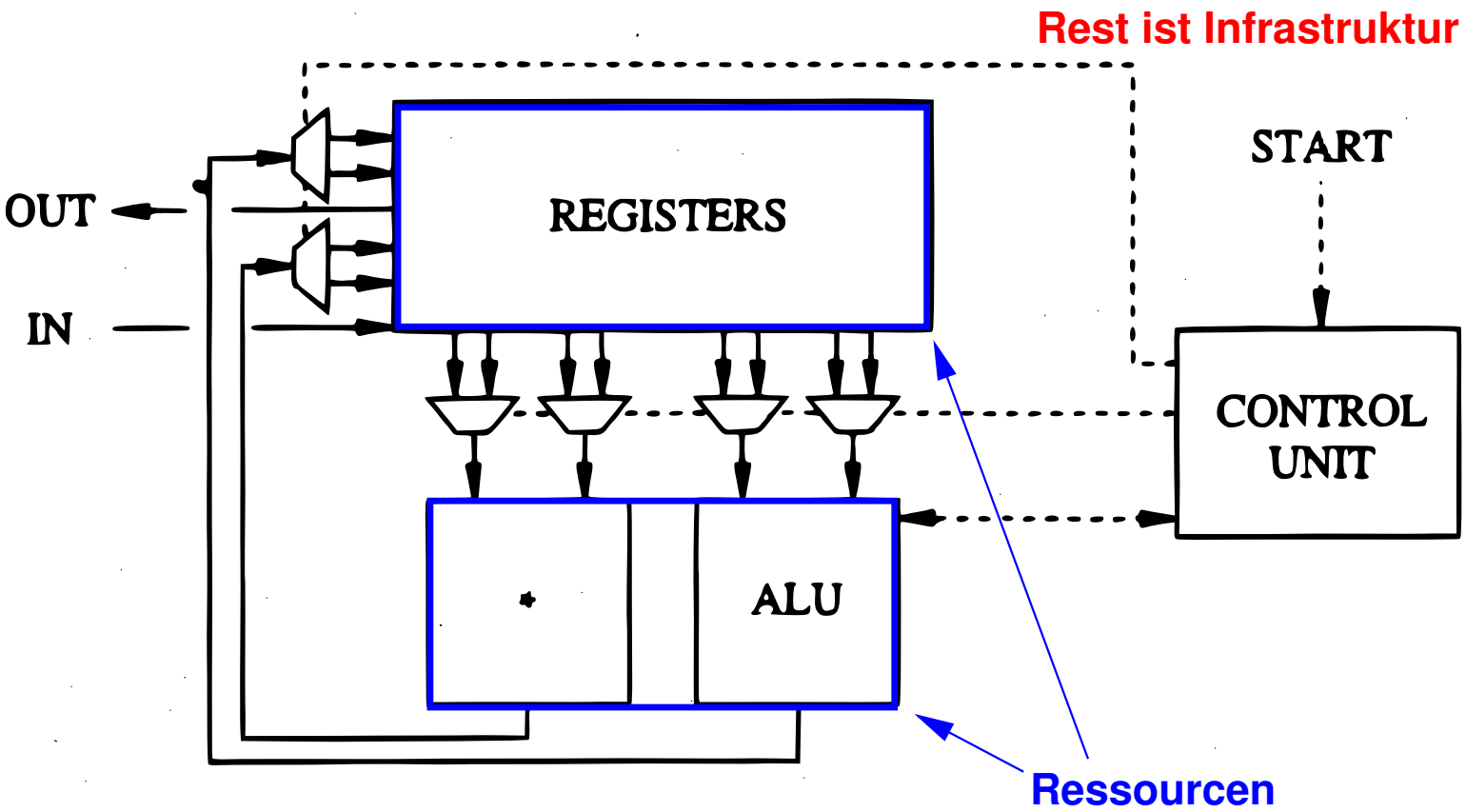
Ressource-dominiert

- Fläche und Zeitverhalten hängt nur von wenigen, gut charakterisierten Blöcken ab
- Beispiel: Anwendungen aus der Signalverarbeitung

Nicht-Ressource dominiert

- Fläche und Zeitverhalten hängen stark von vielen schlecht charakterisierten Infrastrukturelementen ab
- Beispiel: Moderner Mikroprozessor

Beispiel: Ressourcen



Randbedingungen

Zeitverhalten

- Taktperiode (Zykluszeit)
- Latenz
- Durchsatz (bei Pipelining)

Ressourcen

- Anzahlen von Instanzen der Arten
- Konkrete Vorgaben

Eingaben der Architektursynthese

Diesmal verfeinert

- Sequenzgraph $G_S = (V, E)$
  - Operationen $V = \{v_i : i = 0, 1, \cdots, n_{ops}\}$
- Funktionale Ressourcetypen $R = \{r_m : m = 0, 1, \cdots, n_{res}\}$
  - r_m sind charakterisiert in
    - Flächenbedarf $a(r_m) \in \mathbf{N}_0$
    - Zeitbedarf $d(r_m) \in \mathbf{N}_0$

    Definiert: $d(v_0) = d(v_n) = 0$
  - Nun benötigt: Abbildung von V zu R
  - Zunächst *Funktion* $\mathscr{T} : V \rightarrow R$
- Eine Menge C von Randbedingungen

Zeitliche Zuordnung der Operationen

Auch genannt *Ablaufplanung* oder *Scheduling*

- Annotiere jeden Operationsknoten v_i mit seiner *Startzeit* $t_i \in \mathbf{N}$

Ablaufplan

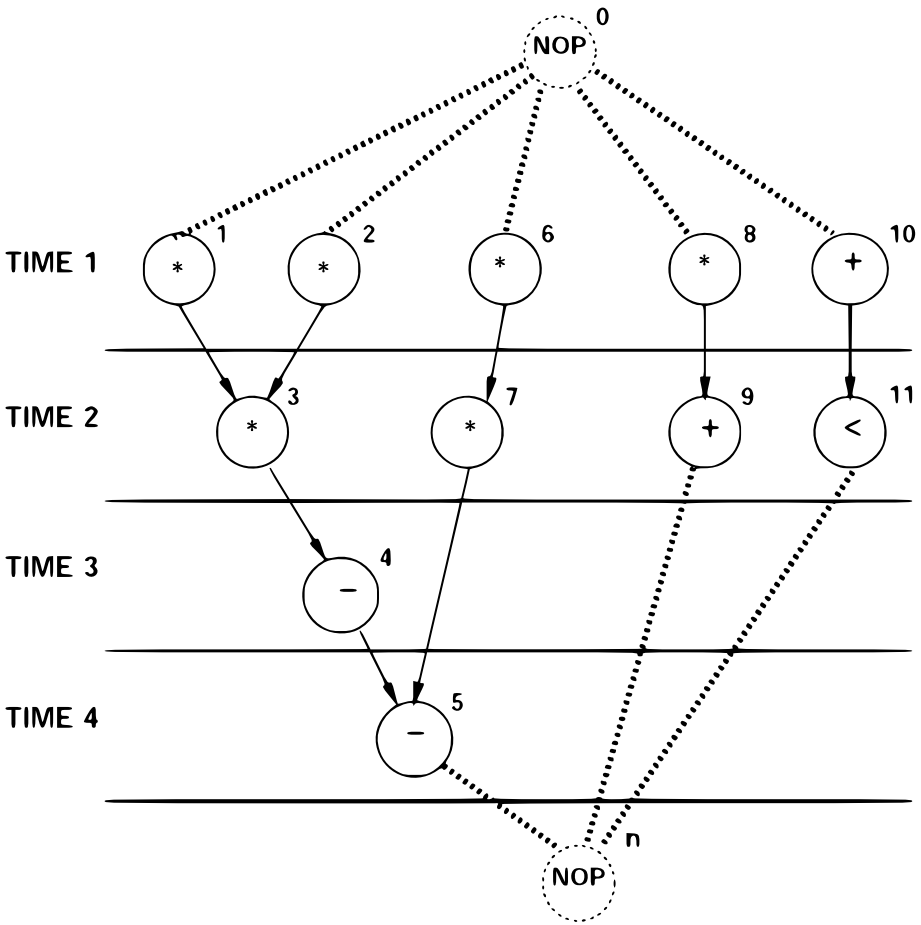
Funktion $\varphi : V \rightarrow \mathbf{N}$, mit $\varphi(v_i) = t_i$, so dass

$$\forall (v_i, v_j) \in E : t_j \geq t_i + d(\mathscr{T}(v_i)),$$

- Bestimmt Gesamtlatenz $\lambda = t_n - t_0$
- Legt die Berechnungsparallelität fest

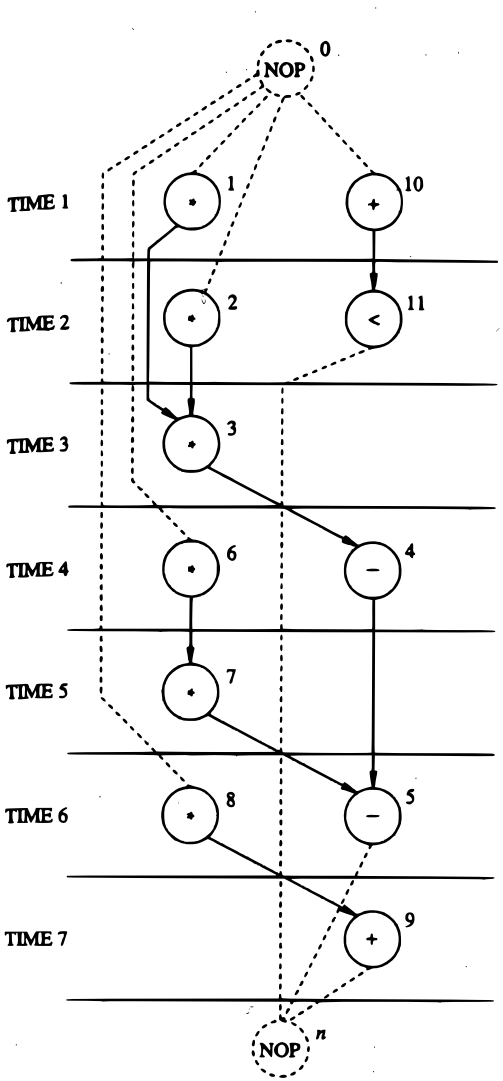
Ergebnis: Sequenzgraph mit geplantem Ablauf

Beispiel: Ablaufplan



Bei $\forall_{r \in R} d(r) = 1$ hier: $\lambda = t_n - t_0 = 5 - 1 = 4$

Beispiel: Ablaufplan mit Randbedingung



- Nur eine Instanz pro Ressourcetyp
- Jetzt $\lambda = 7$

Räumliche Zuordnung der Operationen

Bindung Zuordnung einer typkompatiblen Ressource an jede Operation

- Beispiel: Eine ALU ist typkompatibel zu den Operatoren +,-,>

Bindung

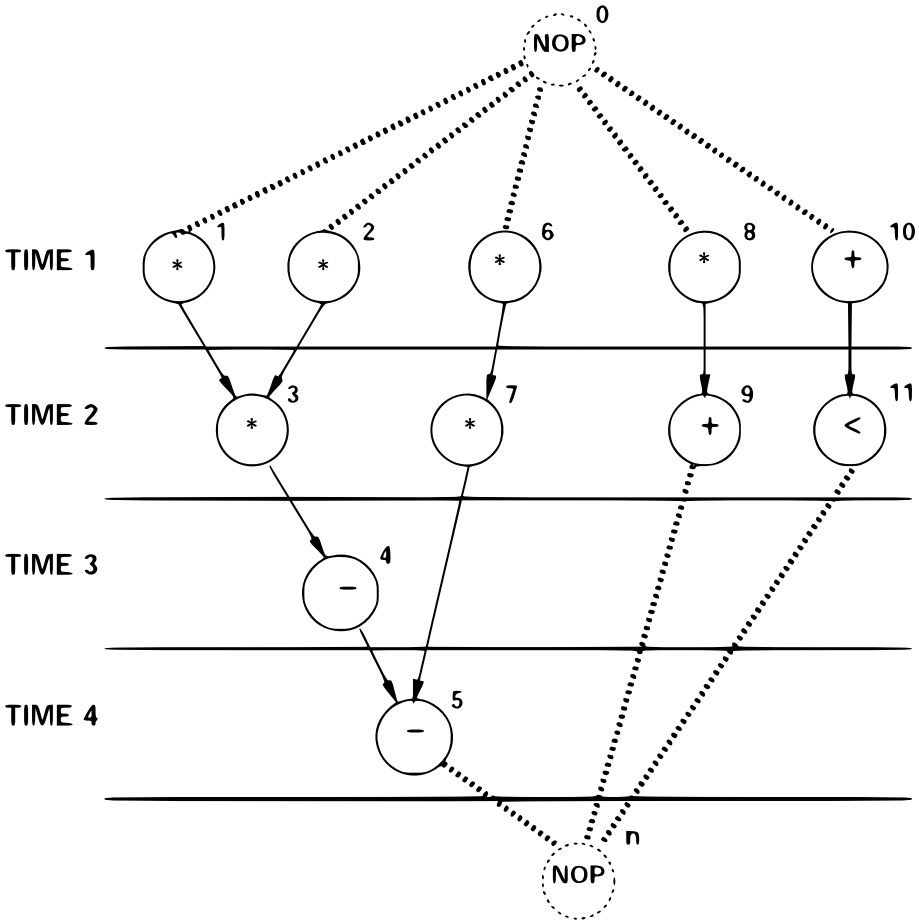
$\beta : V \rightarrow R \times \mathbf{N}$, so dass $\beta(v_i) = (r,k)$ bedeutet: Die Operation $v_i \in V$ mit dem Ressourcetyp $\mathscr{T}(v_i) = r$ wird an die k-te Instanz der Ressource r gebunden. Dies gilt für alle $0 \leq i \leq n_{ops}$.

Beispiel: Bindung

Dedizierte Ressource für jeden Operator

Mult. = Typ 1, ALU = Typ 2

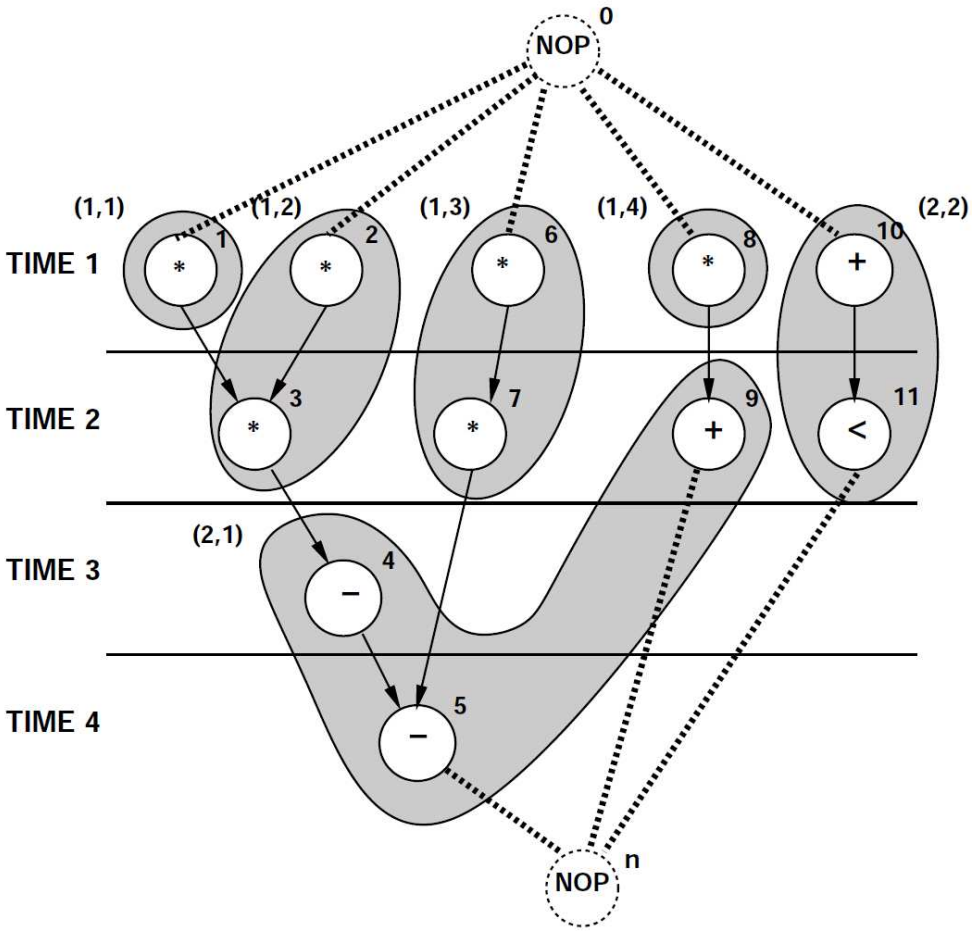
$\mathcal{T}(*) = 1$ $\mathcal{T}(+) = 2$ $\mathcal{T}(-) = 2$ $\mathcal{T}(>) = 2$



$\beta(v_1)$	(1,1)
$\beta(v_2)$	(1,2)
$\beta(v_3)$	(1,3)
$\beta(v_4)$	(2,1)
$\beta(v_5)$	(2,2)
$\beta(v_6)$	(1,4)
$\beta(v_7)$	(1,5)
$\beta(v_8)$	(1,6)
$\beta(v_9)$	(2,3)
$\beta(v_{10})$	(2,4)
$\beta(v_{11})$	(2,5)

Gemeinsame Nutzung von Ressourcen

Mehrere Ops. auf selbe Instanz zu unterschiedlichen Zeiten



$\beta(v_1)$	(1,1)
$\beta(v_2)$	(1,2)
$\beta(v_3)$	(1,2)
$\beta(v_4)$	(2,1)
$\beta(v_5)$	(2,1)
$\beta(v_6)$	(1,3)
$\beta(v_7)$	(1,3)
$\beta(v_8)$	(1,4)
$\beta(v_9)$	(2,1)
$\beta(v_{10})$	(2,2)
$\beta(v_{11})$	(2,2)

Ohne Verlust an Parallelität!

Ergebnis der Bindung

- Gebundener Sequenzgraph mit Ressource-Annotationen
- Häufige Randbedingung: Vorgabe von maximalen Anzahlen für jede Ressource

Beurteilung des Ergebnisses

Abschätzung

Ressource-dominierte Schaltung

Fläche Summe der a_k der gebundenen Ressourcen
Bestimmt durch *Bindung*

Latenz Differenz von Senken- und Quellenstartzeit
Bestimmt durch *Ablaufplanung*

Nicht-Ressource-dominierte Schaltung

Fläche Wird stark von Infrastruktur beeinflusst

Latenz Länge der Taktperiode auch von Infrastruktur beeinflusst

Beispiel für ressource-dominierte Abschätzung

`diffeq()` mit dedizierten Ressourcen

Fläche

- 6 Multiplizierer, 5 ALUs
- Annahmen: a(Multiplizierer)=5, a(ALU)=1
- Für Infrastruktur: 1 Flächeneinheit
- Fläche = $6 * 5 + 5 * 1 + 1$ = 36

Latenz

- Annahmen d(Multiplizierer)=35ns, d(ALU)=25ns
- Taktperiode von 40ns (jeder Op. braucht 1 Takt)
- Latenz = 4 Takte = 160ns

Architekturoptimierung

Berechne φ und β so, dass (Fläche, Latenz, Taktperiode) optimiert werden.
Vorgehen: Variiere eine Grösse als Parameter, bestimme je Wert die beiden anderen als Pareto-Punkte durch Ablaufplanung und Bindung.

Fläche/Latenz-Optimierung

- Für gegebene Werte der Taktperiode

Taktperiode/Latenz-Optimierung

- Für gegebene Fläche (via Bindungsvorgaben)

Fläche/Taktperiode-Optimierung

- Für gegebene Latenz (via Ablaufplanvorgaben)

Beispiel: Fläche/Latenz-Optimierung

Randbedingungen: Max. Fläche = 20, Max. Latenz = 8