

Rechnergestützer Entwurf digitaler Systeme

Sommersemester 2022

Sergei Sawitzki
saw@fh-wedel.de

FH Wedel (University of Applied Sciences)

Rechnergestützer Entwurf digitaler Systeme: Vorlesungs-Gesamtübersicht

1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme:

Gliederung der 1. Vorlesung

1. Einleitung und grundlegende Konzepte
 - 1.1. Organisatorisches
 - 1.2. Einordnung und historische Entwicklung
 - 1.3. Grundlegende Konzepte

1. Einleitung und grundlegende Konzepte

Lernziele

- ▶ Tiefgründiges Verstehen der Methoden und Algorithmen des modernen Schaltungs- und Systementwurfs sowie der Testung
- ▶ Kennenlernen der in den modernen Entwurfssystemen eingesetzten EDA-Algorithmen ($\text{EDA} \cong \text{electronic design automation}$, Entwurfsautomatisierung)
- ▶ Fähigkeit, den Schaltungsentwurfsprozess von der Spezifikation bis zur technischen Umsetzung zu begreifen und in diesen bei Bedarf gezielt eingreifen zu können
- ▶ Fähigkeit, neue EDA-Algorithmen zu entwerfen bzw. bestehende EDA-Algorithmen zu verändern
- ▶ Basis für weiterführende Eigenstudien

Kontaktdaten und Einordnung

Zeitplan: Sommersemester, 1 Vorlesung pro Woche,
montags 14:00–15:15 Uhr, SR6

Präsentationsform: Beamer, Handouts, Tafel

Kontakt: Sergei Sawitzki, Zimmer 208 (Altbau, im 2. OG),
Telefon: (04103)-8048-37, e-Mail: Sergei.Sawitzki@fh-wedel.de

Sprechstunde: Mittwochs 12:30–14:00 Uhr sowie nach
Vereinbarung

Prüfung: mündlich, Teil der Modulprüfung „Diskrete Systeme“

Kreditpunkte: 3,5 (1,5 Prüfung + 2 Praktikum von 5 ECTS für das
Gesamtmodul)

Fortsetzung: Praktikum „Rechnergestützer Entwurf digitaler
Systeme“ (in der 2. Hälfte des laufenden Semesters oder im
folgenden Semester), bei Interesse auch als Abschlussarbeit

Zeitraster

April 2022

			1	2	3		
4	5	6	7	8	9	10	
11	12	13	14	15	16	17	
18	19	20	21	22	23	24	
25	26	27	28	29	30		

Mai 2022

					1		
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	
23	24	25	26	27	28	29	
30	31						

Juni 2022

	1	2	3	4	5		
6	7	8	9	10	11	12	
13	14	15	16	17	18	19	
20	21	22	23	24	25	26	
27	28	29	30				

Juli 2022

					1	2	3
4	5	6	7	8	9	10	
11	12	13	14	15	16	17	
18	19	20	21	22	23	24	
25	26	27	28	29	30	31	

Achtung !

Am 19.04.2022 gilt der Montagsstundenplan! Keine Vorlesung am 06.06.2022 (Pfingstmontag)!

Feedback und Arbeitsweise

Feedback:

- ▶ jederzeit schriftlich (Mail), am besten zeitnah an der entsprechenden Vorlesung, damit es noch in der gleichen Vorlesungsperiode berücksichtigt werden kann
- ▶ Lehrevaluation am Ende der Vorlesungsperiode

Wie immer gilt: Es ist unser **gemeinsames Anliegen**, das Beste aus dieser Vorlesung zu machen!

Sonstiges:

- ▶ Termine und Aufgabenstellungen zum Praktikum werden rechtzeitig in der Vorlesung besprochen
- ▶ In den meisten Vorlesungen werden ergänzende Materialien (Datenblätter, Fachveröffentlichungen) eingesetzt.

Literatur

N. Sherwani: *Algorithms for VLSI Physical Design Automation*, 3rd edition, Kluwer Academic Publishers, 1999

V. Betz, J. Rose, A. Marquardt: *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, 1999

D. Jansen: *Handbuch der Electronic Design Automation*, Hanser Verlag 2001

J. M. Rabaey, A. Chandrakasan, B. Nikolić: *Digital Integrated Circuits: A Design Perspective*, 2nd edition, Prentice Hall 2003

G. Herrmann, D. Müller: *ASIC – Entwurf und Test*, Fachbuchverlag Leipzig, 2004

J. Lienig: *Layoutsynthese elektronischer Schaltungen – Grundlegende Algorithmen für die Entwurfsautomatisierung*, Springer Verlag, 2006

H. Veendrick: *Nanometer CMOS ICs*, Springer Verlag, 2008

Literatur

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

S. K. Lim: *Practical Problems in VLSI Physical Design Automation*, Springer Science+Media, 2008

C. L. Alpert, D. P. Mehta, S. S. Sapatnekar: *Handbook of Algorithms for Physical Design Automation*, CRC Press 2009

EDA-Arbeitsgruppe, Universität Hannover:

<http://edascript.ims.uni-hannover.de/>

Halbleitertechnologie von A bis Z: <http://www.halbleiter.org/>

And last but not least:

und



(mit kritischer Reflexion!)

Bezeichnungen und Konventionen

Die Vorlesungsunterlagen sind in Kapitel, Abschnitte und Unterabschnitte gegliedert (3 Gliederungsstufen, wobei nicht bei jedem Thema die maximale Gliederungstiefe genutzt wird). Zur Orientierung sind diese im Folienkopf angegeben. Bei einzelnen Präsentationen wird zusätzlich die Lage der aktuellen Seite innerhalb des Abschnitts sowie innerhalb der Gesamtpräsentation als Navigationsleiste angezeigt (diese Anzeige entfällt bei Druck der gesamten Vorlesungsunterlagen aus einer Datei).

Definitionen

sind eingerahmt und farblich hervorgehoben.

Fremdsprachige Begriffe erscheinen in Kursivschrift.

PERSONENNAMEN sind in KAPITÄLCHEN gesetzt (gilt nicht für das Titelblatt, das Literaturverzeichnis und die Maßeinheiten).

Gesetze, Sätze, Lemmata u. ä.

sind eingerahmt und farblich hervorgehoben.

Verfügbarkeit von Vorlesungsunterlagen

Die Präsentationen und sonstige begleitende Unterlagen erscheinen rechtzeitig vor Beginn der Vorlesung in Moodle. Zu jeder Vorlesung gibt es mindestens

- ▶ eine Datei mit Präsentation zum Ansehen
- ▶ eine Datei mit Handout zum Ausdrucken (4 Folien pro Seite)
- ▶ eine Datei ohne Overlays/Übergänge (für eigenhändige Gestaltung der Druckausgabe, z. B. 2 Folien pro Seite, skaliert, gespiegelt usw.)

Für konstruktive Verbesserungshinweise (auch wenn es dabei nur um einfache Tippfehler geht) bin ich immer dankbar!

Unterlagen aus dem vergangenen Jahr

Alle „Rechnergestützter Entwurf digitaler Systeme“-Vorlesungen des letzten Jahres sind in Moodle zu finden.

Zu beachten: Die Präsentationen werden von Semester zu Semester geringfügig angepasst.

- ➡ Vergessen Sie nicht, eventuelle Änderungen seitenweise zu ergänzen.

Die Unterlagen werden auch auf dem Handout-Server parallel weiter gepflegt, allerdings ist Moodle zunächst einmal die zentrale Anlaufstelle.

Was ist Rechnergestützer Entwurf digitaler Systeme?

Rechnergestützer Entwurf digitaler Systeme

(auch Entwurfsautomatisierung bzw. *computer-aided digital design, electronic design automation*) ist ein Teilgebiet der (technischen) Informatik, das den Aufbau und die Funktionsweise von Algorithmen und Entwurfswerkzeugen im digitalen Schaltung- und Systementwurf umfasst.

EDA-Werkzeuge decken den gesamten Entwurfsprozess von der Spezifikation bis zur Erzeugung der Fertigungsdaten ab:

- ▶ Anfänge gehen zurück auf die 70er Jahre des letzten Jahrhunderts
- ▶ Inzwischen Grundlage eines eigenständigen Industriezweigs mit mehreren Milliarden US Dollar Umsatz pro Jahr
- ▶ Nach wie vor ein Bereich der intensiven akademischen und industriellen Forschung

Interdisziplinarität

EDA-Werkzeuge bündeln das Know-How aus vielen technisch-naturwissenschaftlichen Disziplinen

- ▶ praktische Informatik (Programmierung, Datenstrukturen, Software-Engineering, Datenbanken)
- ▶ theoretische Informatik (Algorithmen, Berechenbarkeits- und Komplexitätstheorie)
- ▶ technische Informatik (Entwurfsmethoden, Synthese, Optimierung, Modellierung und Simulation)
- ▶ Physik (Elektro- und Thermodynamik, Eigenschaften von Festkörpern, Halbleiterphysik)
- ▶ Mathematik (BOOLEsche Algebra, Graphentheorie, Wahrscheinlichkeitsrechnung und Statistik, multidimensionale Optimierung, Visualisierung, Geometrie)
- ➡ Ein perfektes Einsatzgebiet für technische Informatiker

Ein einführendes Zitat als weitere Motivation

By reading this book, you take the first step toward joining the great journey of design automation. Any sufficiently advanced technology is indistinguishable from magic, and the magic of Design Automation is literally shaping our future (as well as making a lot of people fabulously wealthy).

Gary D. HACHTEL, Fabio SOPENZI, *Logic Synthesis and Verification Algorithms*, Springer 2006

Ein weiteres Zitat als Warnung:

Given that MLFM is only justified in practice for netlists with more than 100–200 nodes, any complete example illustrating it would look farfetched. The same applies to analytical placement algorithms, which only shine at the large scale. In fact, this is one of the reasons why competitive physical design algorithms are so hard to develop — small examples can be misleading.

Igor MARKOV, *A physical-design picture book (Review of S. K. Lim's: „Practical Problems in VLSI Physical Design Automation“)*

Einige Meilensteine

1950–1965	Manueller Entwurf
um 1970	Unterstützung durch Graphik-Programme bei der manuellen Erzeugung der Layout-Daten
1973	SPICE1 als erstes „richtiges“ EDA-Werkzeug
1975	Erste Platzierungs- und Verdrahtungs-Werkzeuge
1980–1988	Gründung der ersten industriellen Anbieter von EDA-Software, z. B. Mentor Graphics, Cadence Design Systems, Synopsys
1983	D. D. GAJSKI (geb. 1938): Y-Diagramm
1984	Erste Version von Verilog-HDL (Standardisierung durch IEEE in 1995)
1985	Vorstellung der TimberWolf-Software
1985	Erste Version von VHDL (Standardisierung durch IEEE in 1987)
seit 1990	Zunehmende Integration von Verifikation und Test-Komponenten in EDA-Software

Einige Meilensteine (fortgesetzt)

1999	<i>IEEE Standard for VHDL RTL Synthesis</i>
2000	Erste Version von SystemC (Standardisierung durch IEEE in 2005)
seit 2000	Verstärkter Einsatz von komplexeren IP-Modulen
2002	Erste Version von System Verilog (Standardisierung durch IEEE in 2005)
2020	EDA-Industrie-Einnahmen im 3. Quartal des Jahres belaufen sich auf fast 3 Milliarden US\$

Ein weiterer Meilenstein der EDA-Entwicklung

MEAD & CONVAY Revolution (ca. 1975–1980, auch Mikrochip-Entwurfs-Revolution)

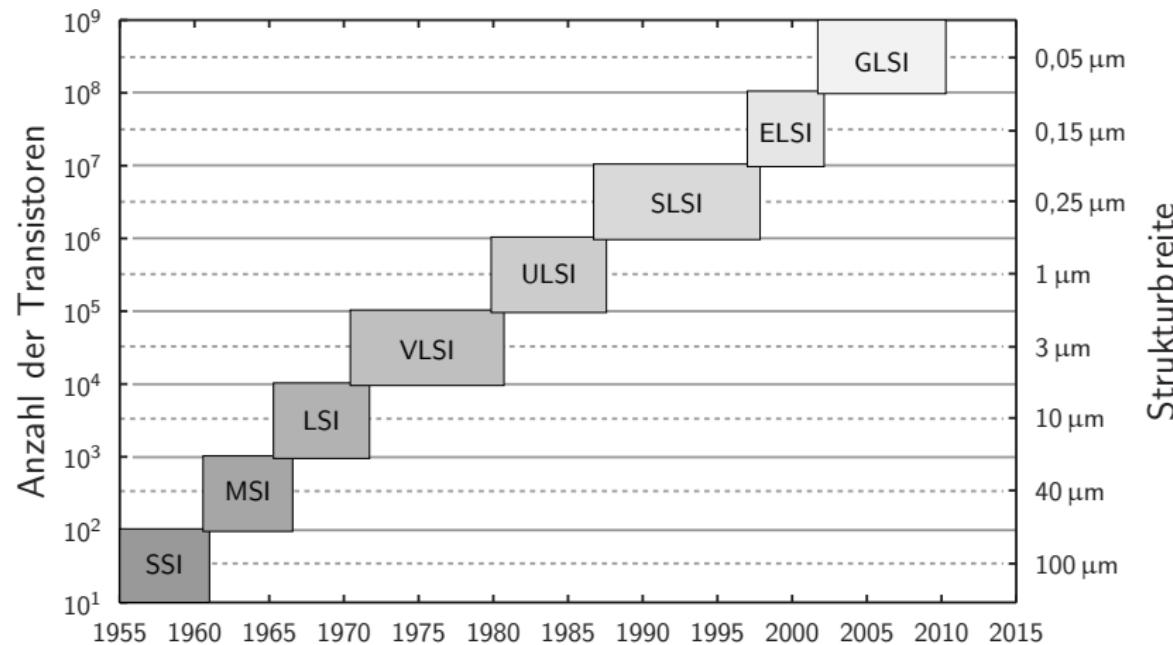
- ▶ Benannt nach Carver Andress MEAD (geb. 1934, amerikanischer Ingenieur und Informatiker) und Lynn Ann CONVAY (geb. 1938, amerikanische Ingenieurin und Informatikerin), die 1980 das bahnbrechende Buch „*Introduction to VLSI Systems*“ veröffentlicht haben
- ▶ Synonym für Entstehung und Etablierung des Entwurfs integrierter Schaltungen (sowie Entwurfsautomatisierung) als eigenständige wissenschaftliche Disziplin
- ▶ Trennung der Entwurfsmethodik von der Entwurfstechnologie
- ▶ In der Bundesrepublik Deutschland durch Reiner HARTENSTEIN (geb. 1934) als Mitinitiator des E. I. S. Projektes (Entwurf integrierter Schaltungen) ca. 1983 angestoßen

Generationen von EDA-Werkzeugen

Jahr	Gener- ation	Abstrakti- onsstufe	Merkmale
um 1970	1.	Transis- toren	Zeichenprogramme als Hilfe bei manueller Layout-Erzeugung, analoge Simulation
um 1980	2.	Logik- gatter	Automatische Platzierung und Verdrahtung, Logiksimulation
um 1990	3.	RTL	HDL-Unterstützung, Register-Transfer-Synthese und -Simulation, Timing-Analyse, formale Verifikation, automatische Testmustererzeugung
um 2000	4.	Algo- rithmus	IP-Wiederverwendung, „High-Level-HDL“, Hardware-Software-Codesign und Mixed-Mode-Entwurf

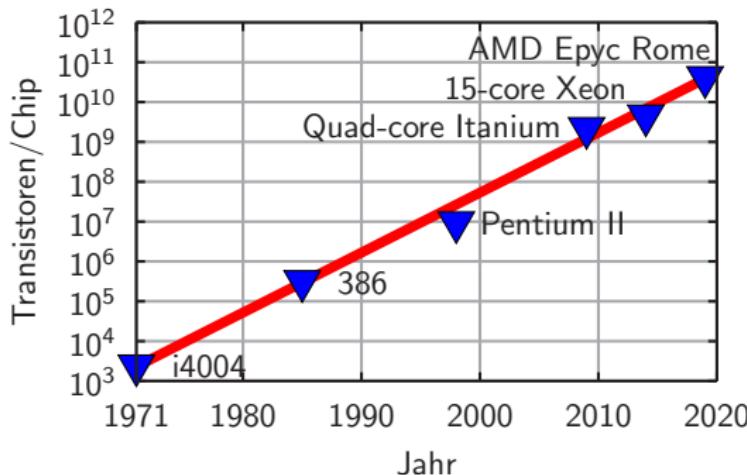
Mikroelektronik: Trend zur Verkleinerung

Überblick über die Steigerung der Integrationsdichte und Reduzierung der Strukturbreite in der Mikroelektronik:



Zum Vergleich: ITWissen, www.itwissen.info, SLSI-Technologie

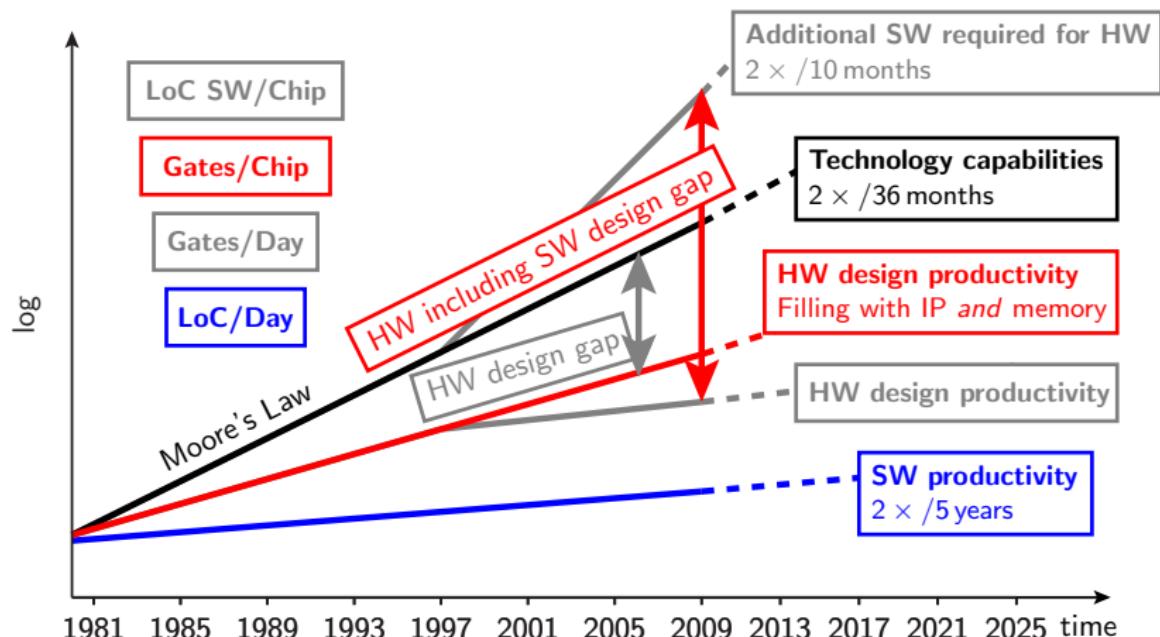
Das Gesetz von MOORE



Gordon MOORE (geb. 1929, US-amerikanischer Physiker und Ingenieur, Mitbegründer und ehemaliger CEO von Intel)

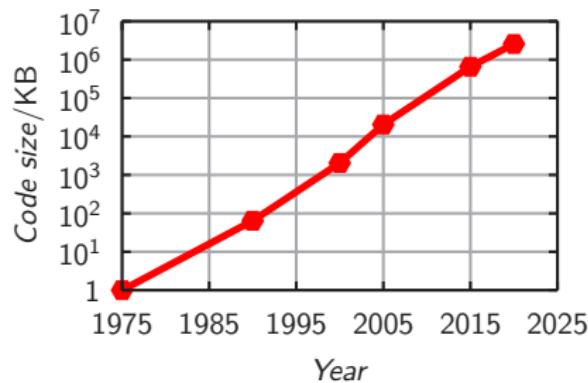
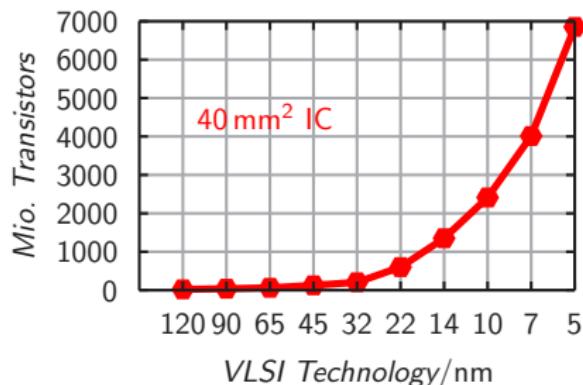
- ▶ Verdopplung der Anzahl von Transistoren pro Flächeneinheit alle 2 Jahre (quellenabhängig werden auch Zeitspannen von 18–24 Monaten genannt)
- ▶ seit etwa 2010 Verlangsamung auf ca. 2,5 Jahre
- ▶ prognostiziertes Ende nach 2025

Entwurfsproduktivität und technologische Möglichkeiten



Quelle: *International Technology Roadmap for Semiconductors, Chapter „Design“, Issue 2011*

Anstieg der Hardware- und Software-Komplexität

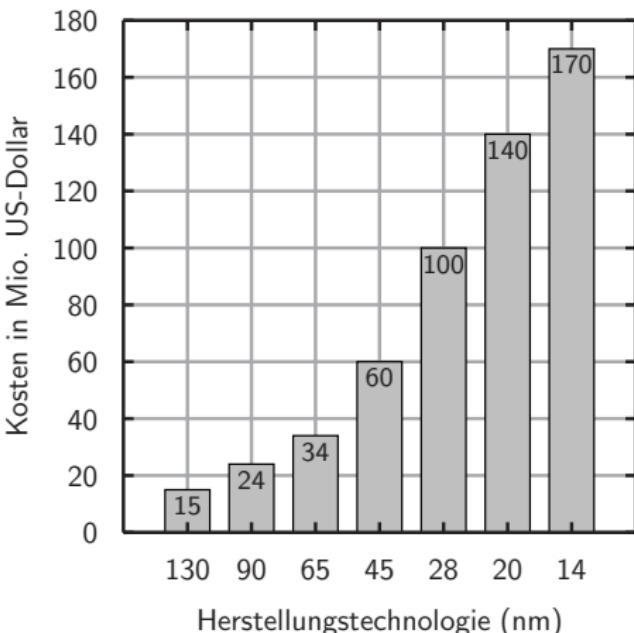


Gerät	Mio. Codezeilen (<i>lines of code</i> , ungefähr)
Rasierer	0,1–0,5
Fernseher, set-top-box	1–10
Handy, Smartphone	5–10
Oberklassen-Limousine	20–100

Zum Vergleich: 1 Mio. *lines of code* sind ca. 18 000 gedruckte Seiten

Anstieg der Entwicklungskosten

NRE (*non recurring engineering*) Kosten sind einmalig auftretenden Fixkosten. Im Bezug auf den ASIC-Entwurf sind es im Wesentlichen die Kosten für das Layout und die Maskenherstellung (in der nebenstehenden Abbildung enthalten).



Ergänzende Materialien

Entwicklungsprognosen für Halbleiter-Industrie:

<http://www.itrs2.net/2011-itrs.html>

<http://www.itrs2.net/2012-itrs.html>

<http://www.itrs2.net/>

Zusammenfassung

- ▶ Kurze Vorstellung der Vorlesungsinhalte, Aufbau, Kontaktdaten und organisatorischer Ablauf
- ▶ Vorstellung der Literaturquellen
- ▶ Wichtigste Grundbegriffe, Einordnung und Motivation
- ▶ Kurzer Ausflug in die Geschichte von Entwurfsautomatisierung

Rechnergestützer Entwurf digitaler Systeme: Vorlesungs-Gesamtübersicht

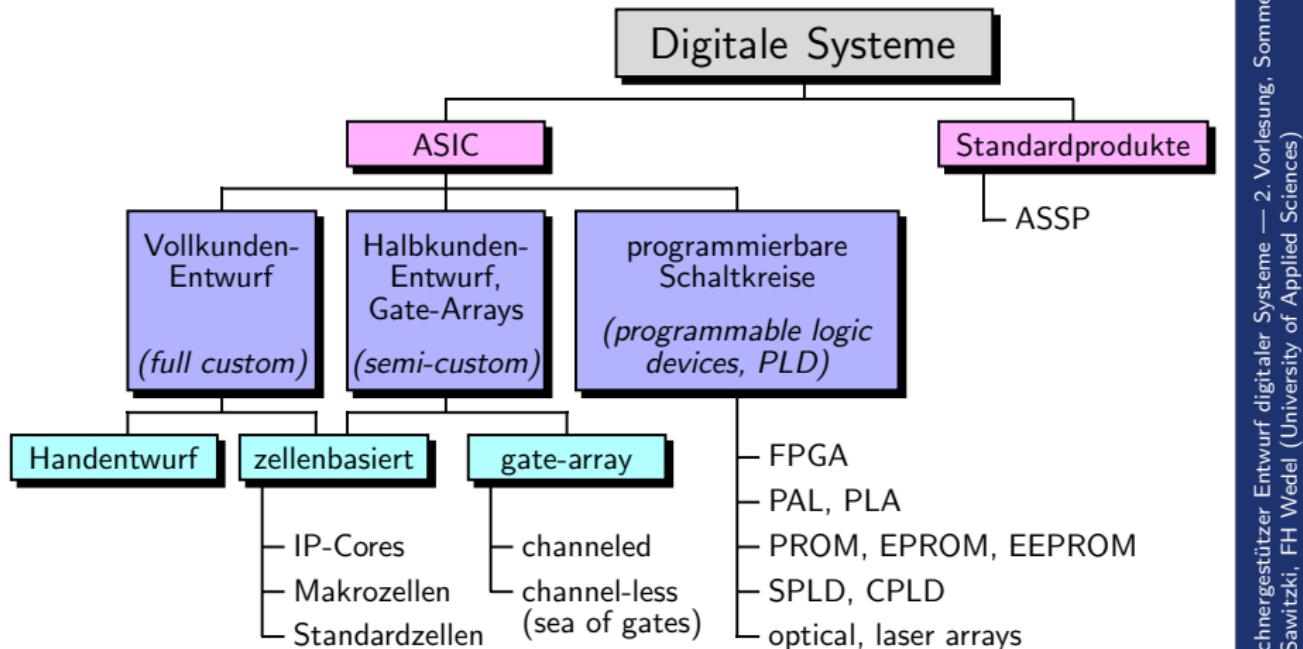
1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme:

Gliederung der 2. Vorlesung

1. Einleitung und grundlegende Konzepte
 - 1.1. Organisatorisches
 - 1.2. Einordnung und historische Entwicklung
 - 1.3. Grundlegende Konzepte
 - 1.3.1. Anwendungsspezifische Schaltkreise
 - 1.3.2. Allgemeiner Entwurfsablauf

Klassifikation digitaler Schaltungen und Systeme



Begriffserklärungen

ASIC, *application specific integrated circuit*: anwendungsspezifischer integrierter Schaltkreis

ASSP, *application specific standard product*: anwendungsspezifisch, jedoch nicht für einen bestimmten Kunden entworfen

FPGA, *field-programmable gate-arrays*: beim Endanwender programmierbare Schaltkreise

PAL, *programmable array logic*

PLA, *programmable logic array*

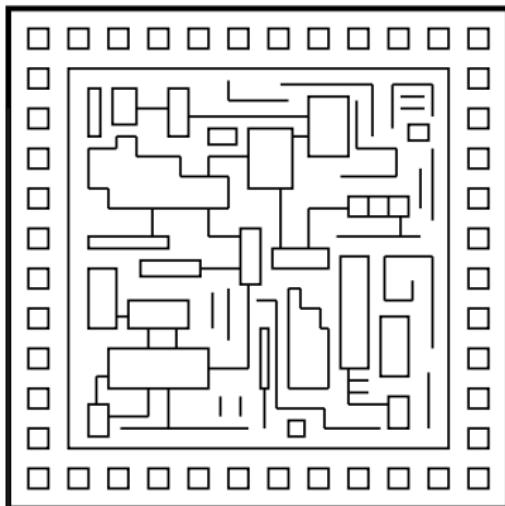
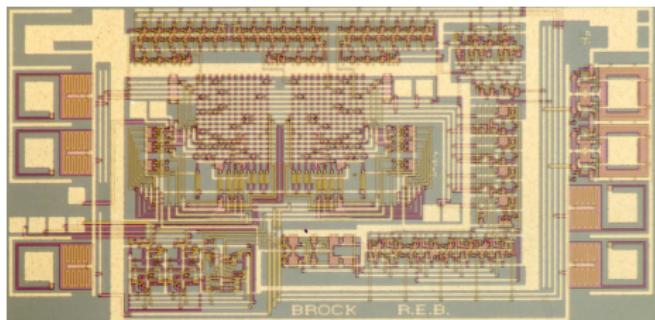
PLD, *programmable logic device*

SPLD, *simple programmable logic device*

CPLD, *complex programmable logic device*

((E)E)PROM, *((electrically) erasable) programmable read-only memory*

Typisches Layout eines ASIC in Handentwurf



Bildquelle: MIT (Ausschnitt aus dem 1978 Multi-Chip Set)

- ▶ keine reguläre Struktur erkennbar
- ▶ heutzutage nur bei Schaltkreisen geringerer Komplexität bzw. beim Entwurf von Standardzellen eingesetzt
- ▶ bevorzugter Stil beim Analog-Entwurf

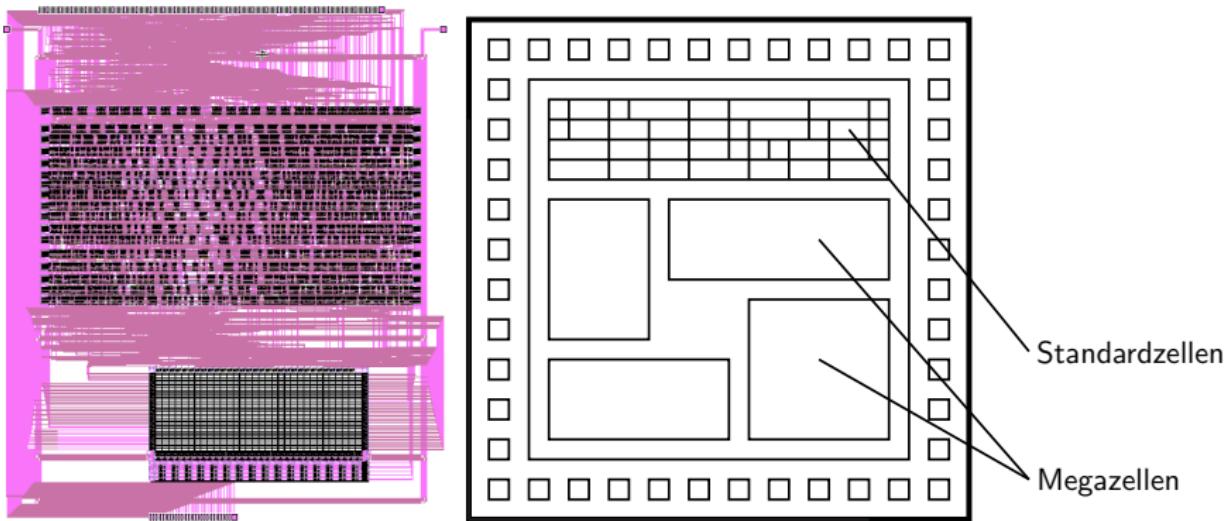
Merkmale eines Handentwurfs

- ▶ Ursprünglich die einzige existierende Entwurfstechnik für mikroelektronische Schaltungen
- ☺ Alle Entwurfs-Freiheitsgrade nutzbar, die einzigen Einschränkungen sind technologiebedingte *design rules*
- ➡ Sehr effiziente Flächennutzung, Layout nah am Optimum
- ☹ Hoher Zeitaufwand, hohe Komplexität
- ➡ Hohe Fehlerwahrscheinlichkeit
- ▶ Werkzeugunterstützung: Layout-Editor, meistens mit integrierter Entwurfsregelüberprüfung (*design rule check*)
- ▶ Praktisch nur bei Schaltungen geringer Komplexität oder bei Produkten mit sehr hohen Stückzahlen einsetzbar (meistens auch nur für Teile des Gesamtsystems)



Bildquelle: Valvo GmbH, 1966

Typisches Layout eines Standardzellen-ASIC



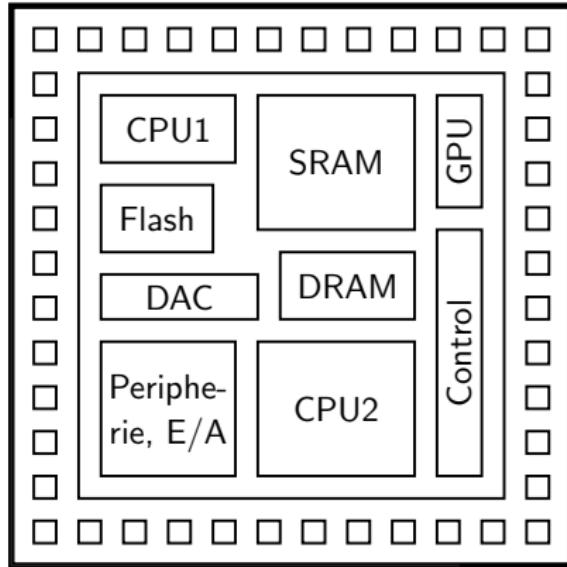
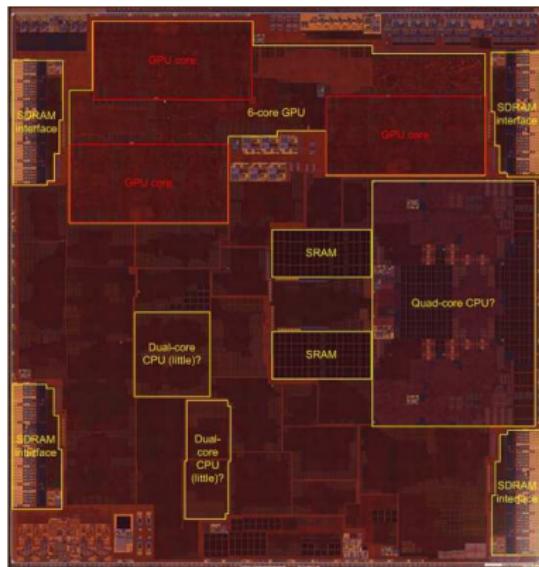
Bildquelle: Pldworld (32-Bit ALU mit Barrelshifter)

- ▶ reguläre Struktur
- ▶ Zellen sind in Standardzellen-Bibliotheken vordefiniert und aufeinander abgestimmt
- ▶ ein sehr weit verbreiteter Entwurfsstil

Merkmale eines Standardzellen-Entwurfs

- ▶ Alle Zellen haben eine feste Höhe (aber flexible Breite), Makrozellen können beliebige Abmessungen haben
- ☺ Reihenanordnungen sowie geometrisch sinnvolle Anbringung für Betriebsspannung und Verdrahtung möglich
- Erleichterung der Entwurfsaufgaben durch Bereitstellung von Entwurfsbibliotheken, Begünstigung von Wiederverwendung (*design reuse*) bei Makrozellen
- ▶ Bei mehreren Metallisierungsebenen entfällt die Notwendigkeit von Verdrahtungskanälen: OTC-Routing (*over the cell*)
- ▶ Werkzeugunterstützung: Durchgängige Unterstützung (z. B. von der Logik- bis zur Layout-Synthese)
- ▶ Dominierende Entwurfstechnik bei anwendungsspezifischen Schaltkreisen

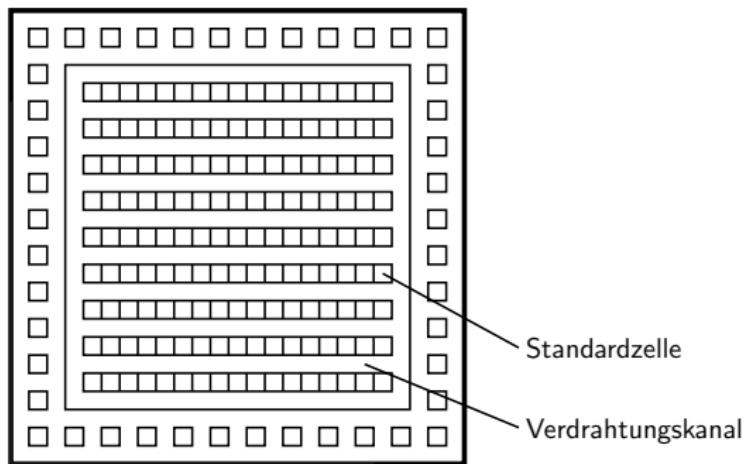
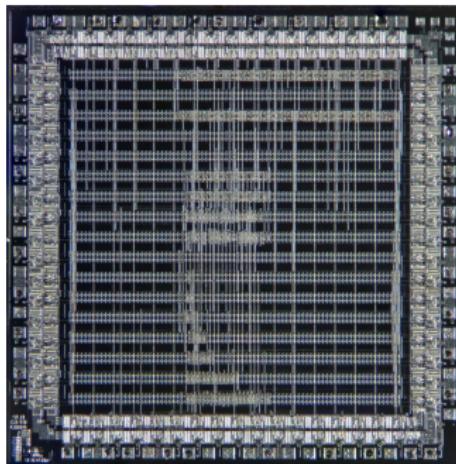
Typisches Layout eines System-on-Chip (SoC)



Bildquelle: Chipworks (Apple A10 Fusion SoC)

- ▶ einzelne Komponenten sind als IP-Blöcke verfügbar
- ▶ beim Entwurf werden die Komponenten zusammengefügt, bei Bedarf wird Verbindungslogik (*glue logic*) hinzugefügt

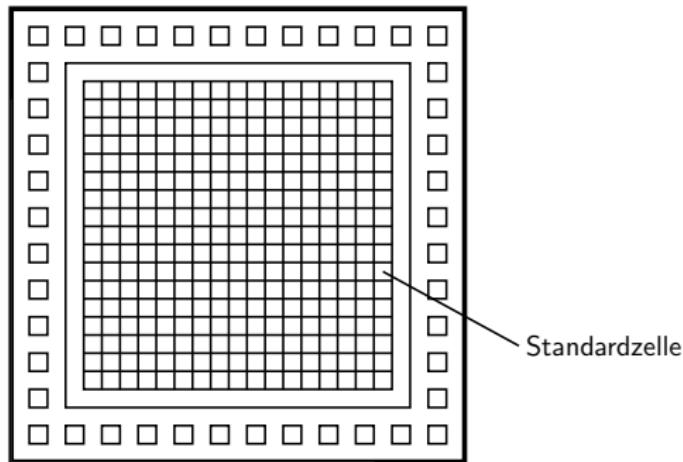
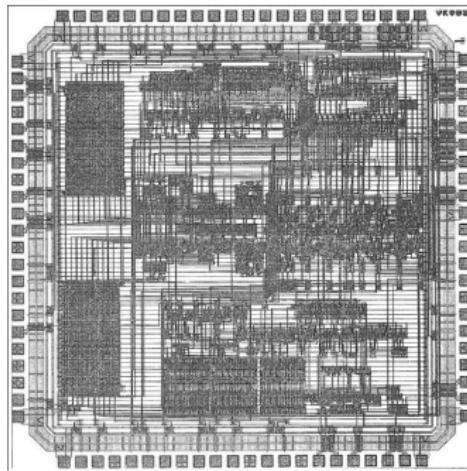
Typisches Layout eines *channeled gate-array*



Bildquelle: Antoine Bercovici, CC BY-SA 3.0 (S-MOS Systems SLA6140 Gate-Array)

- ▶ Für Logikzellen und Verdrahtungskanäle sind feste Bereiche definiert
- ▶ Logikzellen sind meistens einfache Gatter bzw. Transistoren
- ▶ Flächeneffizienz ist gering

Typisches Layout eines *channel-less gate-array, sea-of-gates*



Bildquelle: Zarlink Semiconductor Inc. (CLA60000 Gate-Array)

- ▶ Reguläre Anordnung von Logikzellen
- ▶ Logikzellen haben meistens eine Komplexität von 4–10 Transistoren
- ▶ Verdrahtung komplett flexibel
- ▶ Flächeneffizienz ist besser als bei *channelled gate array*

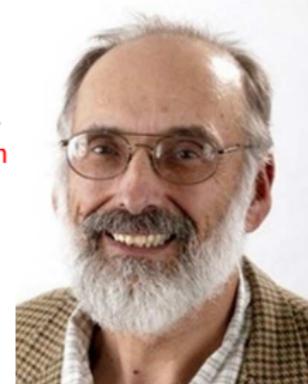
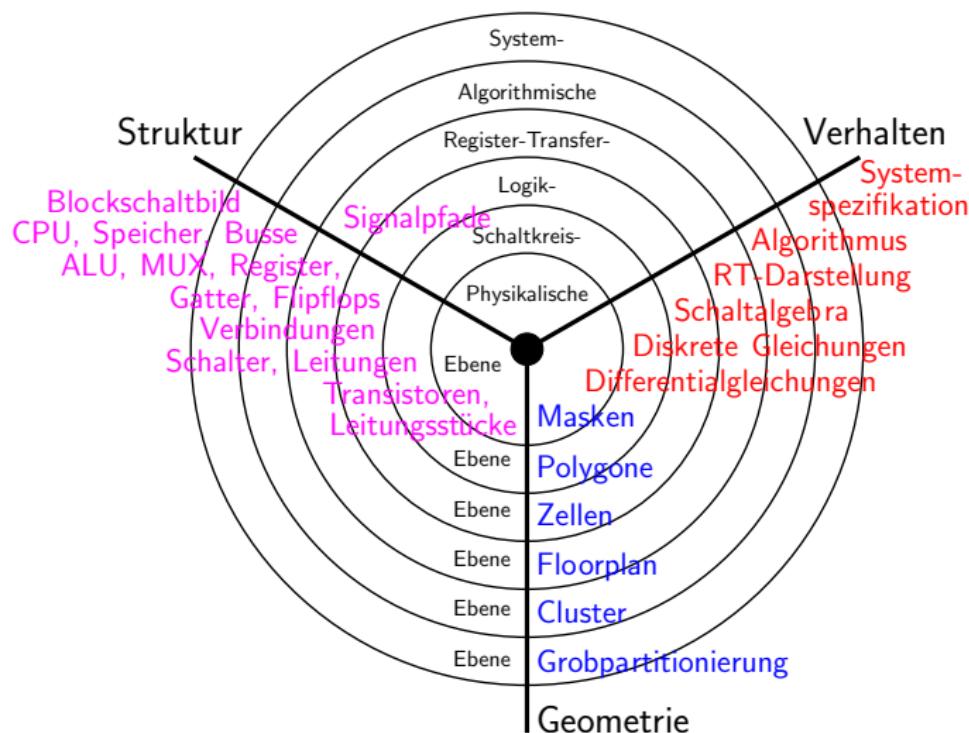
Merkmale eines Gate-Array-Entwurfs

- ☺ Wafer können bis auf die Verdrahtung vorgefertigt werden
- ➡ Nur die Maskensätze für die Verdrahtung sind kundenspezifisch, Kostenreduktion
- ☹ Nicht alle Gatter/Transistoren können in den Entwurf eingebunden werden
- ➡ Layouts sind nicht optimal
- ▶ Werkzeugunterstützung: Durchgängige Unterstützung (z. B. von der Logik- bis zur Layout-Synthese), Besonderheiten der Topologie müssen berücksichtigt werden (z. B. Breite der Verdrahtungskanäle)
- ▶ Früher oft die kostengünstigere Alternative zum Vollkunden- oder Standardzellenentwurf, heute zunehmend durch Field-Programmable Gate-Array abgelöst

Fertigungstechnologien

- ▶ Vollkundenentwurf und zellenbasierter Halbkundenentwurf:
 - ▶ ASIC ist komplett entworfen
 - ▶ Alle Herstellungsmasken gehen sofort in die Fertigung
 - ▶ Am Ende entsteht ein vollständiges Produkt
- ▶ Gate-Arrays:
 - ▶ ASIC ist vorgefertigt (bis auf die Verdrahtung/Metallisierungsebenen)
 - ▶ Vorgefertigte Gate-Arrays werden gelagert
 - ▶ Endprodukt entsteht durch nachträgliches Hinzufügen der Verdrahtung
 - ▶ Nur Masken der Metallisierungsebene gehen in die Endfertigung

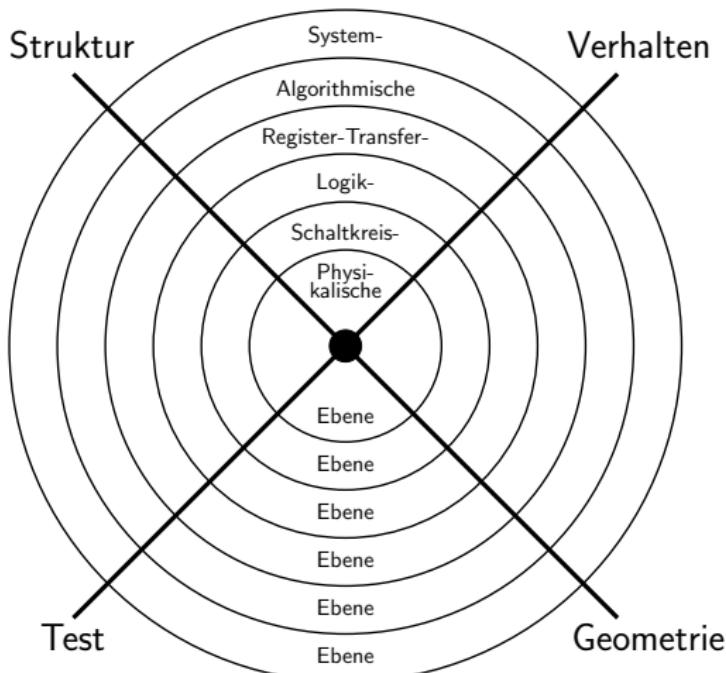
Y-Diagramm nach D. GAJSKI und R. H. KUHN (1983)



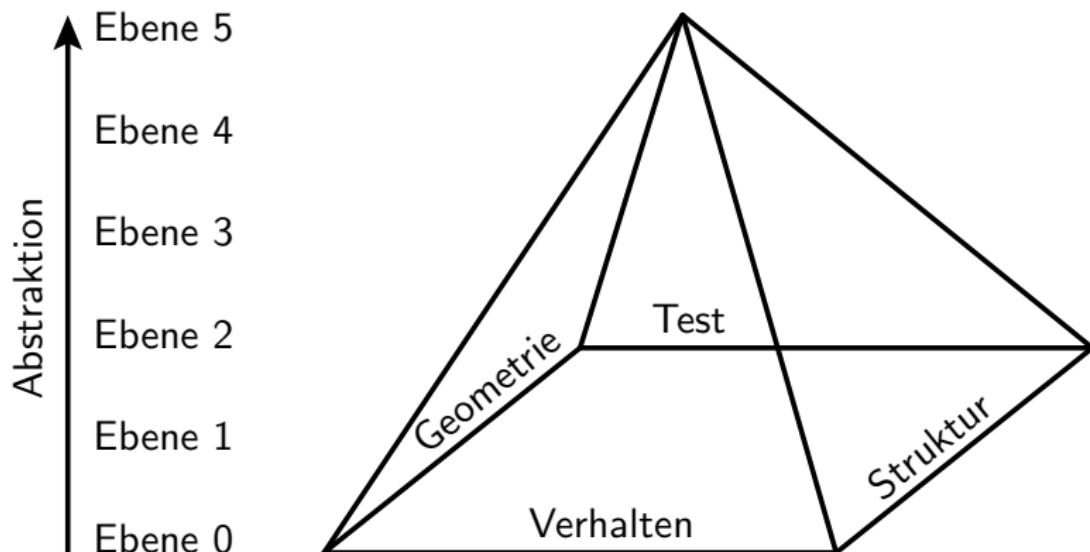
Daniel D. GAJSKI
(geb. 1938,
kroatisch-
amerikanischer
Ingenieur)

X-Diagramm

Erweiterung des Y-Diagramms um die Test-Sicht:

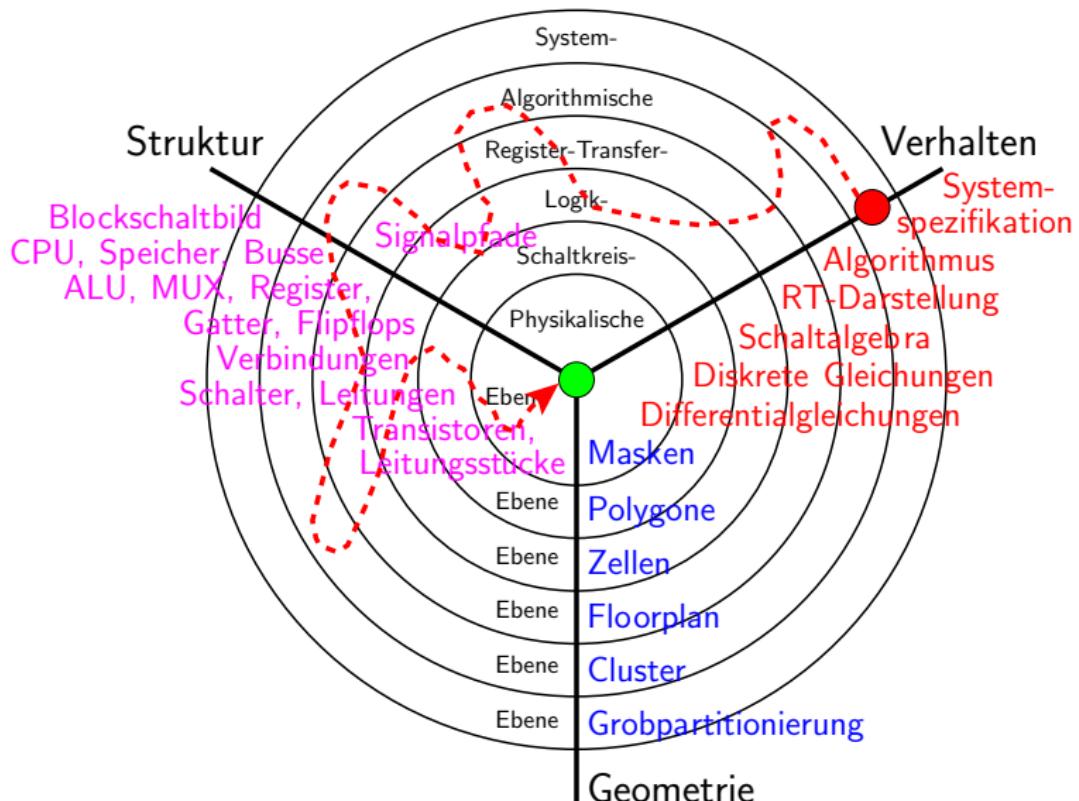


Modellierung des Entwurfsprozesses als Entwurfspyramide



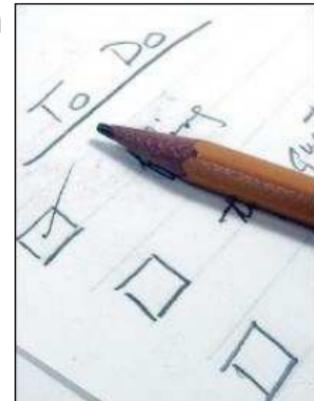
- ▶ gleiche Aussagekraft wie X-Diagramm
- ▶ zusätzlich eine qualitative Abbildung des Informationsgehaltes jeder Ebene

Beginn und Ende eines Entwurfsprozesses



Systemebene

- ▶ Beschreibung der globalen Systemeigenschaften
 - ▶ Technisch: Grenzwerte für Betriebsspannung, Taktfrequenz, Verlustleistung, Temperaturbereich u. ä.
 - ▶ Nichttechnisch: Preis u. ä.
- ▶ Anschlüsse, Schnittstellen
- ▶ Hardware-Software-Aufteilung, Grobe Aufteilung in Teilsysteme
- ▶ Verhaltensbeschreibung (meistens relativ abstrakt bzw. unvollständig)
- ➡ Kein Zeitmodell, keine Implementierungsdetails



Beschreibungsmittel: Verbale Beschreibungen, Blockschaltbilder, *high-level* Sprachen und Simulationssysteme

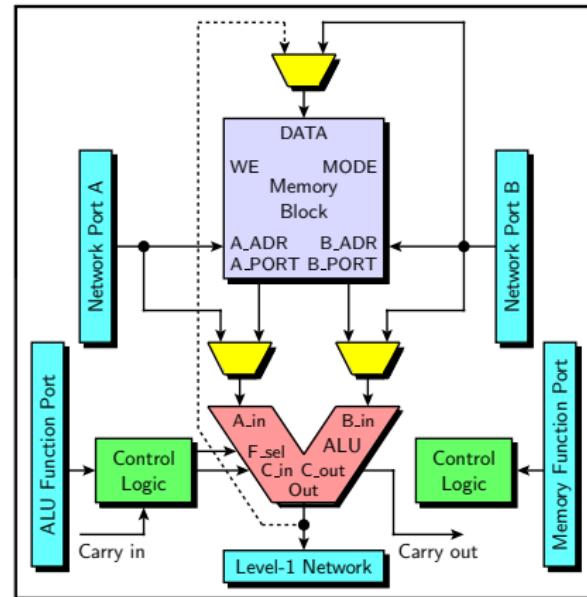
Algorithmische Ebene

- ▶ Verhaltensbeschreibung (Operatoren und Zuweisungen) ...
X = X*Y;
- ▶ Festlegung der globalen Module und Standards (CPU, Busse, evtl. Betriebssystem) IF (Z=1) THEN
X = X+2*Y
- ▶ Interpretationsalgorithmen für Teilsysteme ELSE
- ▶ Vorentscheidungen über die Architektur X = 2*X
- ▶ Weitestgehende Festlegung der Codierung, END IF
Formate und Schnittstellen ...
- ➡ Abstraktes Zeitmodell, erste Implementierungsdetails

Beschreibungsmittel: Programmiersprachen, Hardwarebeschreibungssprachen

Register-Transfer-Ebene

- ▶ RTL-Beschreibung (Daten- und Steuerfluss)
- ▶ Festlegung der Anzahl und Breite von Registern und Verarbeitungseinheiten
- ▶ Umsetzung der Teilsysteme in Form von Schaltnetzen und Schaltwerken
- ▶ Endgültige Entscheidungen über die Architektur
- ▶ Endgültige Festlegung der Codierung, Formate und Schnittstellen
- ➡ Taktzyklusgenaues Zeitmodell, Scheduling

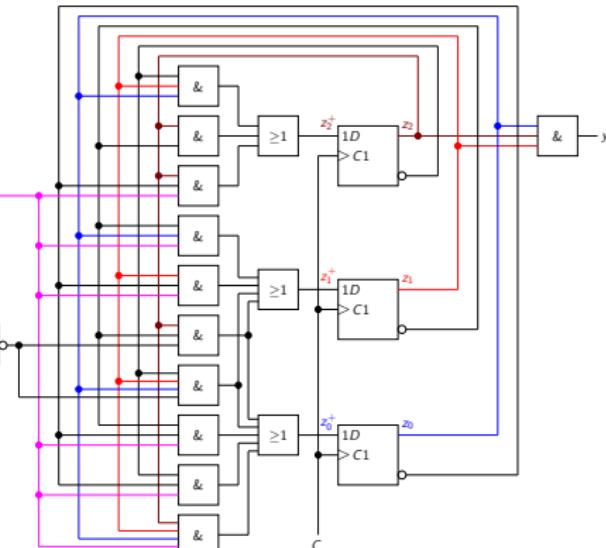


Beschreibungsmittel: Hardwarebeschreibungssprachen

Logik-Ebene

- ▶ Beschreibung mit Gattern und Speicherelementen
- ▶ Festlegung der logischen und zeitlichen Parameter (Anzahl und Art der Logik- und Speicherelemente)
- ▶ Implementierung von Schaltnetzen und Schaltwerken als Verbindungen von Gattern und Speicherelementen
- ▶ Festlegung der Basiszellen
- ➡ Kontinuierliches Zeitmodell, Verzögerungszeiten, diskrete Signalwerte

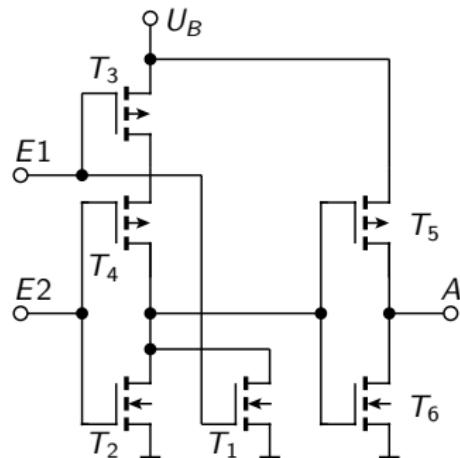
Beschreibungsmittel: Netzlisten von Basiszellen, schaltalgebraische Gleichungen



Schaltkreis-Ebene

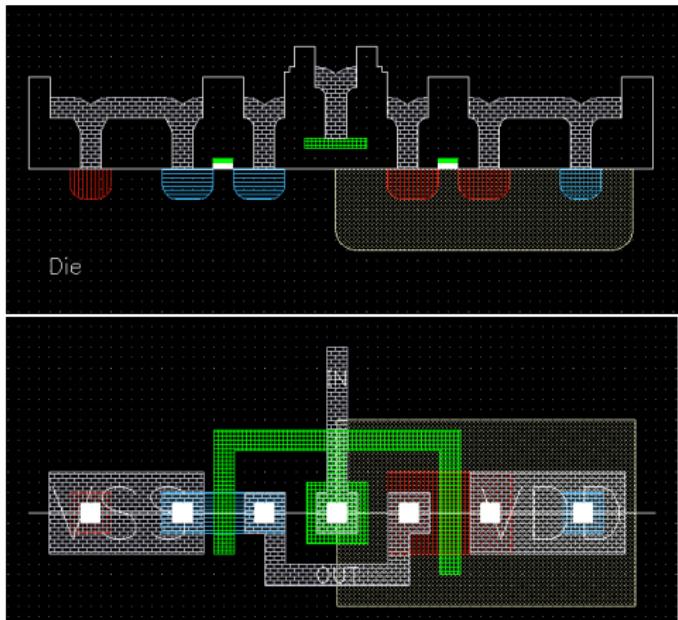
- ▶ Beschreibung mit Transistoren bzw. weiteren grundlegenden Schaltelementen
- ▶ Festlegung der Transistor-Art und -dimensionen
- ▶ Implementierung von Gattern mit Transistoren
- ▶ Implementierung von Verbindungen mit Leitungsabschnitten, symbolisches Layout
- ➡ Kontinuierliches Zeitmodell, kontinuierliche Signalwerte

Beschreibungsmittel: Netzlisten von Transistoren, vereinfachte Transistormodelle



Physikalische Ebene

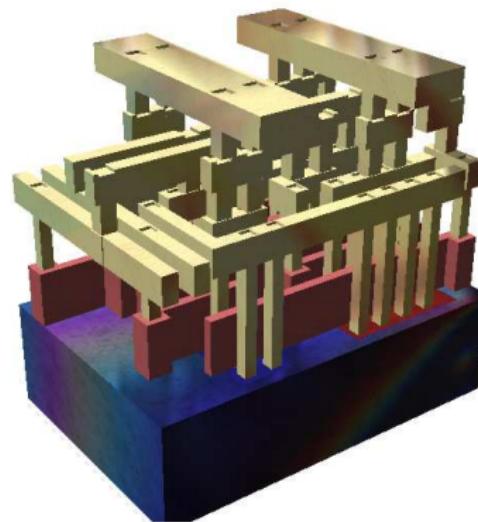
- ▶ Beschreibung der exakten geometrischen Anordnung der Elemente
- ▶ Festlegung der physikalischen Umsetzung des Systems auf einem Halbleiterkristall
- ▶ Implementierung von Transistoren in Form einer integrierten Halbleiter-Schaltung
- ▶ Implementierung von Verbindungen mit physikalischen Leitungen
- ➡ Exakte Zeitverhaltensinformation



Beschreibungsmittel: Polygone, Maskensätze

Vom Entwurf zur Fertigung

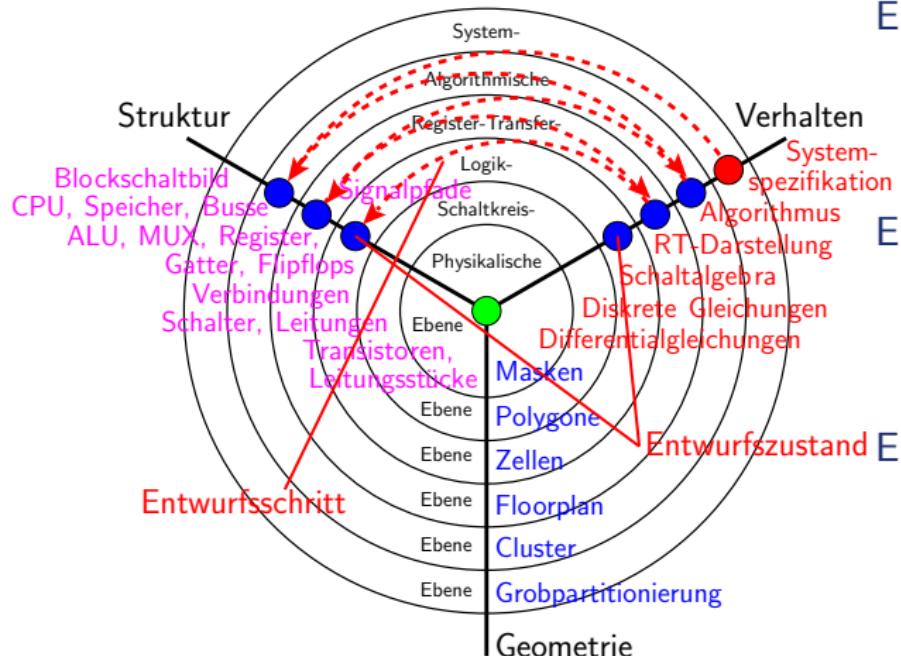
- ▶ *Tape-out*: Übergabe der Layout-Daten an Fertigung bzw. Erzeugung der Maskendaten
- ▶ Datenformate: GDSII (*Graphic Design System II*), OASIS (*Open Artwork System Interchange Standard*)
- ▶ Letzter Entwurfsschritt
- ▶ Moderne Fertigungsverfahren erfordern weitere Modifikationen der Layout-Daten (Auflösungsverbesserungsverfahren wie OPC = *optical proximity correction* usw.)



Standardzelle mit 3 Metallisierungsebenen (ohne Isolationsschichten) mit Verdrahtung in gelb (Metall), Gate-Strukturen in rot (Polysilizium) und Siliziumsubstrat in sonstigen Farben.

Bildquelle: David Carron, Wikipedia

Entwurfsfluss im Y-Diagramm



Entwurfszustand:

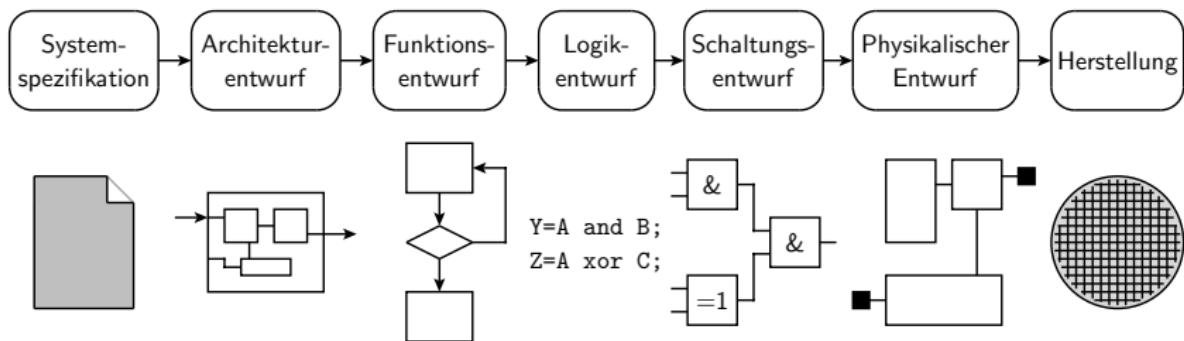
Schnittpunkt einer Sicht mit einer Beschreibungsebene

Entwurfsschritt:

Übergang zwischen zwei Entwurfszuständen

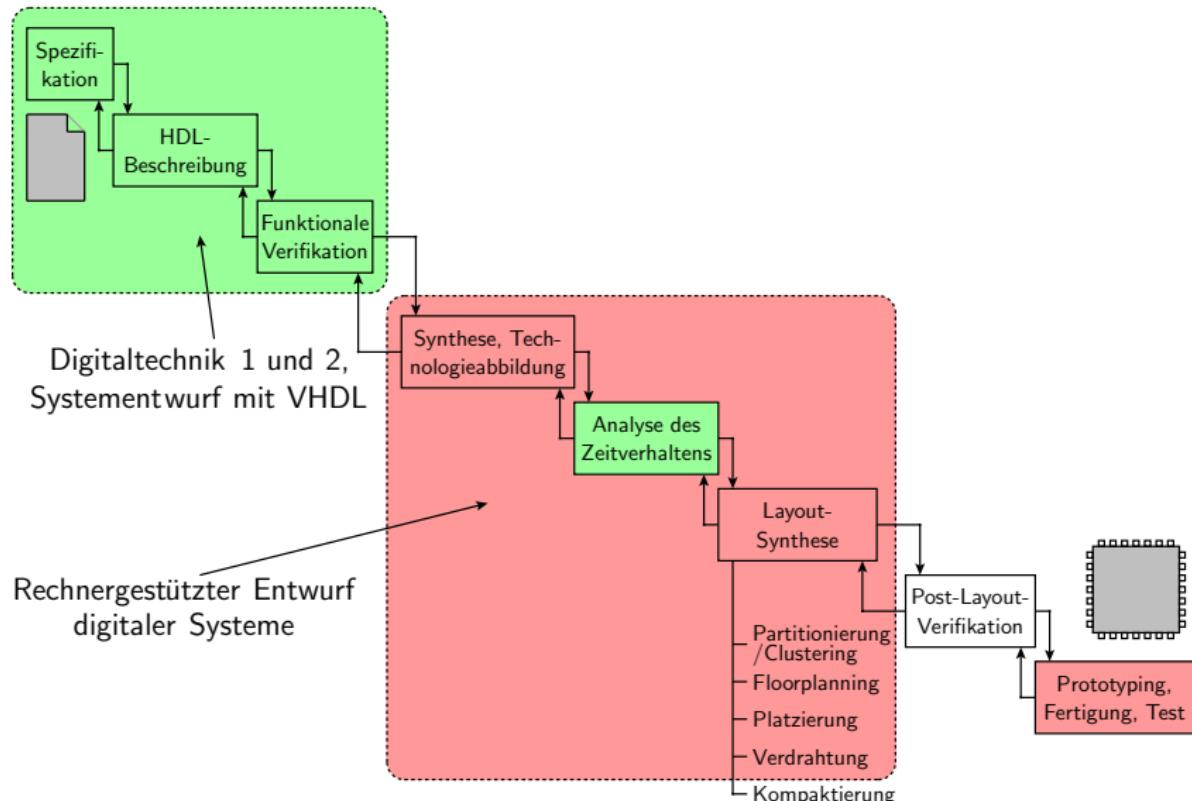
Entwurfsstil: Abfolge von Entwurfsschritten, die auf der physikalischen Ebene endet

Schematische Darstellung eines Top-Down-Entwurfs

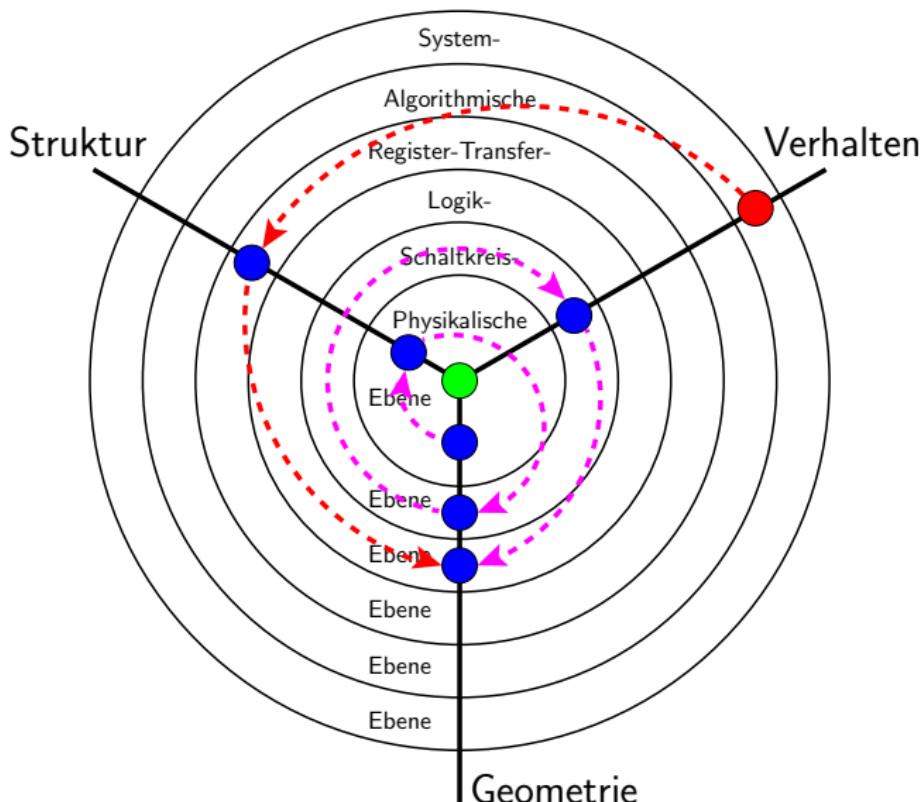


In der Praxis (zumindest für komplexe Systeme) kaum durchführbar, daher meistens *Meet-in-the-Middle-Ansatz* (auch als Jo-Jo-Ansatz bekannt).

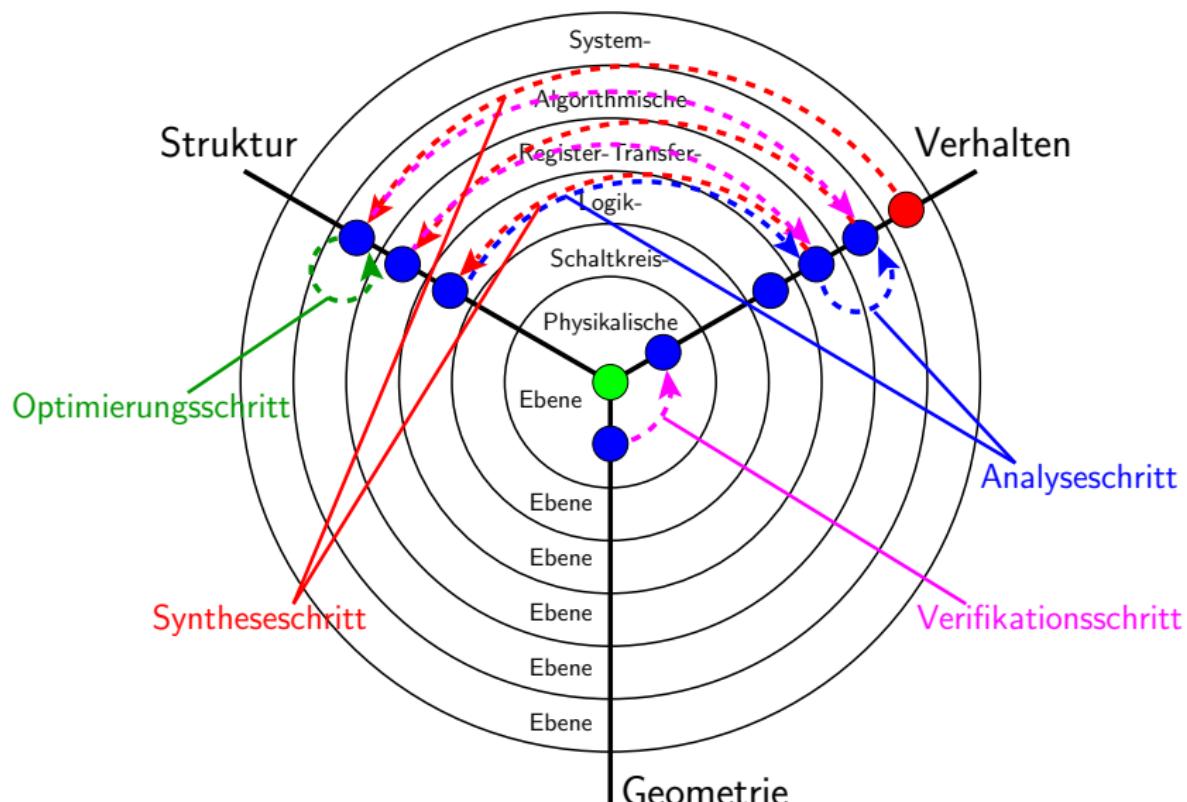
Technologische Schritte beim Top-Down-Entwurf



Beispiel für eine *Meet-in-the-Middle*-Entwurfsstrategie



Einteilung der Entwurfsschritte



Systematik der Entwurfsschritte

Syntheseschritt: Ein Schritt, der den Entwurfszustand dem Entwurfsziel (physikalisches Layout) näher bringt, d. h. ein Schritt, der eine Grenze zwischen den Ebenen in absteigender Richtung kreuzt bzw. ein Schritt, der auf einer Ebene in Richtung Geometriesicht verläuft. Dabei sinkt das Abstraktionsniveau und steigt das Detaillierungsniveau. Es entstehen neue Entwurfsinformationen.

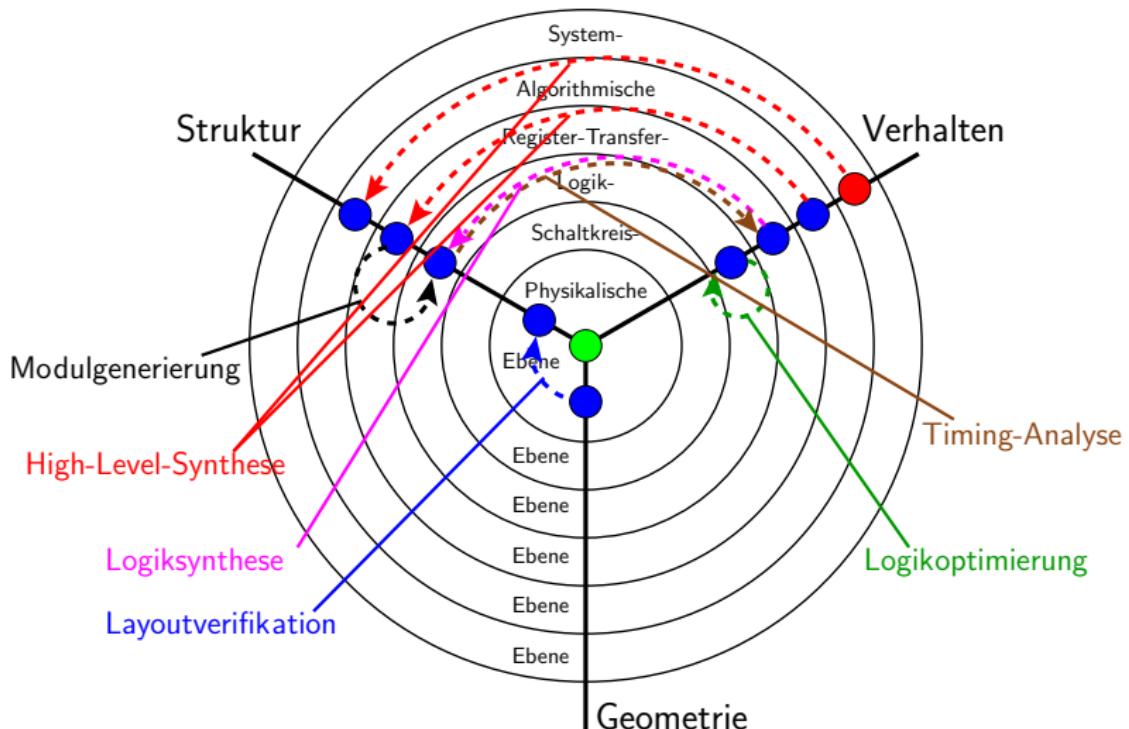
Analyseschritt: Ein Schritt, der den Entwurfszustand vom Entwurfsziel entfernt und dabei eine Grenze zwischen den Ebenen in aufsteigender Richtung kreuzt. Dabei steigt das Abstraktionsniveau und Detailinformationen werden zusammengefasst (generalisiert). Analyseschritte dienen meistens der Validierung.

Systematik der Entwurfsschritte (fortgesetzt)

Optimierungsschritt: Ein Schritt, der weder einen Ebenen- noch einen Sichtwechsel zur Folge hat, d. h. der Entwurfszustand bleibt gleich. Der Entwurf wird dabei nach vorgegebenen Kriterien optimiert (z. B. Zustandsminimierung bei einem Schaltwerk, Logikoptimierung usw.)

Verifikationsschritt: Ein Schritt, der auf einer Ebene weg vom Entwurfsziel verläuft (z. B. Geometrie \rightarrow Verhalten). Im Gegensatz zur Analyse erfolgt dabei kein Wechsel der Ebenen. Die Beschreibungen der beiden Sichten können (meistens vollautomatisch) verglichen werden, um die Einhaltung von Entwurfsregeln zu beweisen.

Viele Entwurfsschritte haben eine etablierte Bezeichnung



Zusammenfassung

- ▶ Vorstellung von Entwurfstechniken anwendungsspezifischer Schaltkreise
- ▶ Entwurfssystematik und Darstellung des Entwurfsablaufs

Rechnergestützer Entwurf digitaler Systeme: Vorlesungs-Gesamtübersicht

1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme:

Gliederung der 3. Vorlesung

1. Einleitung und grundlegende Konzepte

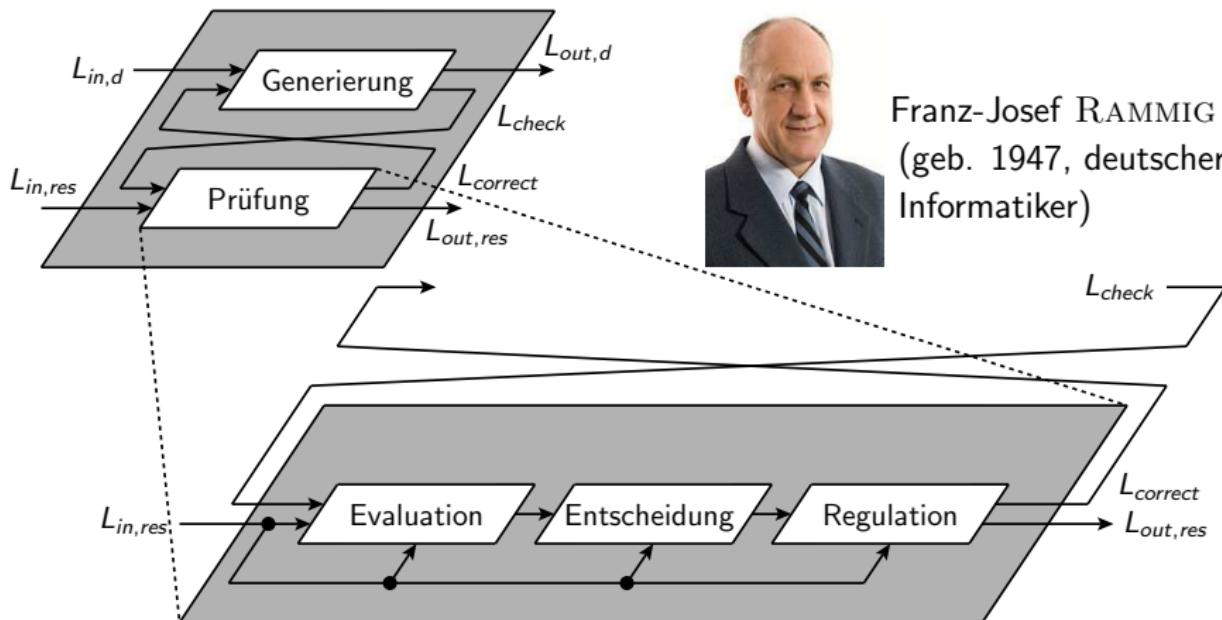
- 1.1. Organisatorisches
- 1.2. Einordnung und historische Entwicklung
- 1.3. Grundlegende Konzepte
 - 1.3.1. Anwendungsspezifische Schaltkreise
 - 1.3.2. Allgemeiner Entwurfsablauf

2. Synthese und Technologie-Abbildung

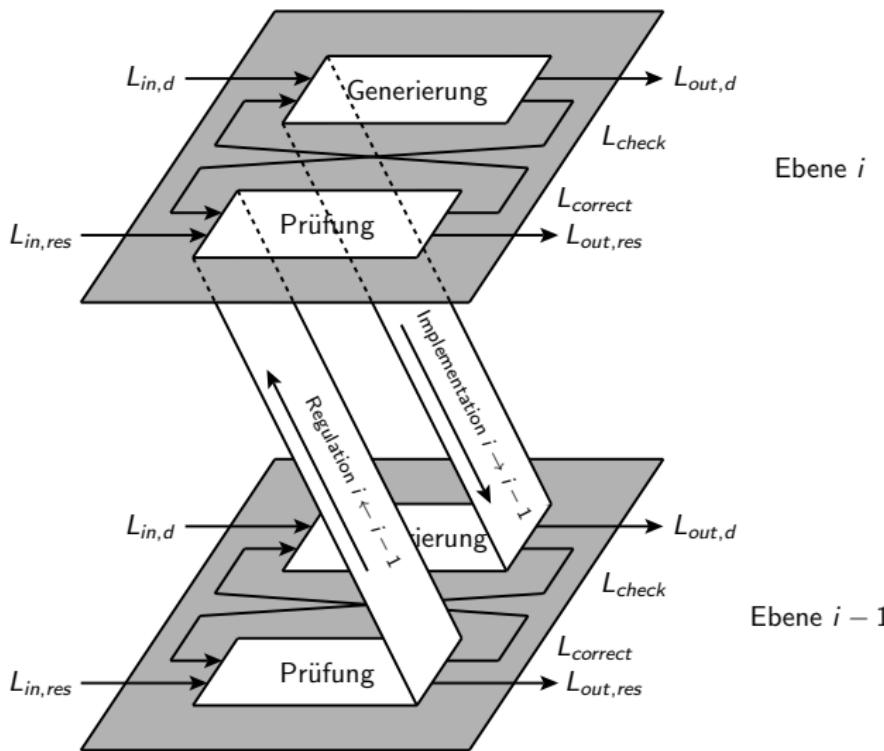
- 2.1. Einordnung
- 2.2. HDL-Synthese
- 2.3. Logiksynthese und Optimierung
- 2.4. Technologie-Abbildung

Makroskopisches Modell des Entwurfsablaufs innerhalb einer Entwurfsebene nach F.-J. RAMMIG

Ein- und Ausgabesprachen: $L_{in} = L_{in,d} \cup L_{in,res}$, $L_{out} = L_{out,d} \cup L_{out,res}$
 $d \cong design$, Entwurfsabsichten; $res \cong restrictions$, Einschränkungen



Mikroskopisches Modell des rückgekoppelten Entwurfsprozesses nach F.-J. RAMMIG



Erläuterungen zum mikroskopischen Modell

Generierungsaktivitäten innerhalb einer Ebene:

- ▶ Modifikation
- ▶ Optimierung

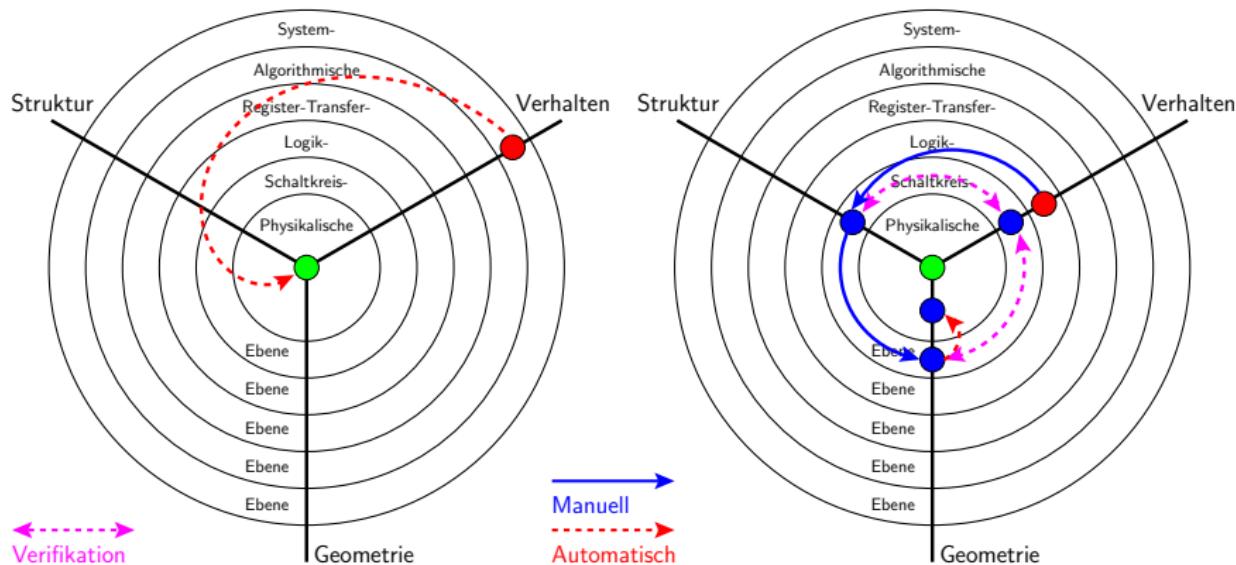
Abstieg zwischen zwei oder mehreren Ebenen:

- ▶ Implementation
- ▶ ersetzt eine Generierungsaktivität auf niedrigerer Ebene
- ▶ das Ergebnis wird nach makroskopischem Modell in der niedrigeren Ebene behandelt

Aufstieg zwischen zwei oder mehreren Ebenen:

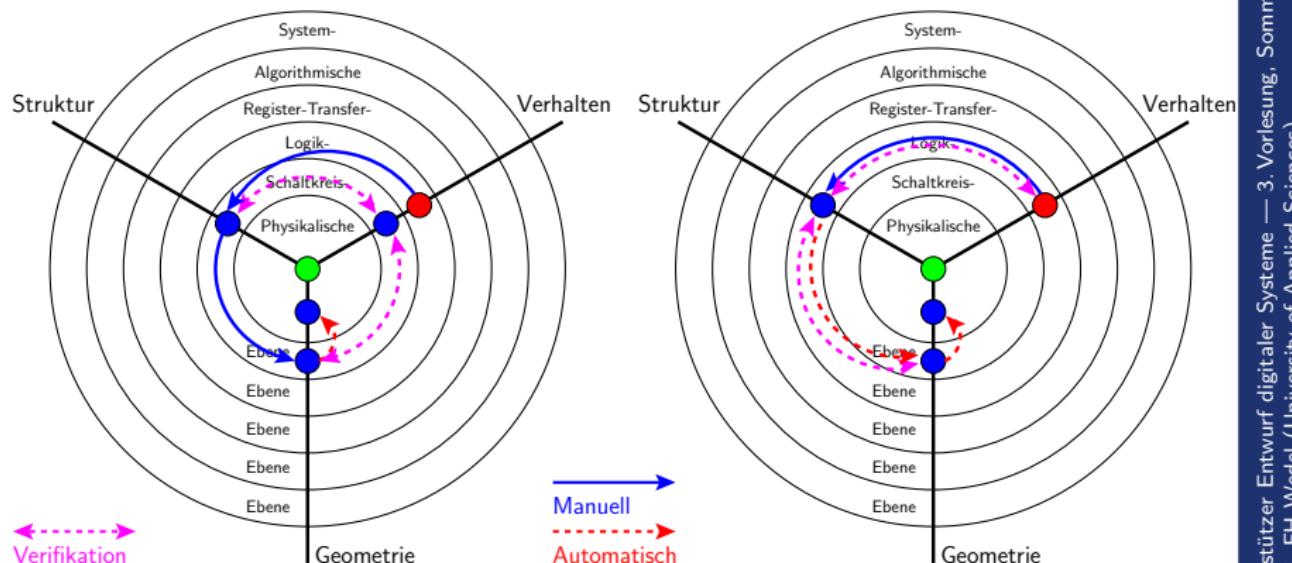
- ▶ Regulation
- ▶ propagiert Problembeschreibung zurück in die höhere Ebene, falls das Problem auf der niedrigeren Ebene nicht gelöst werden kann

Silicon Compiler und EDA der 1. Generation



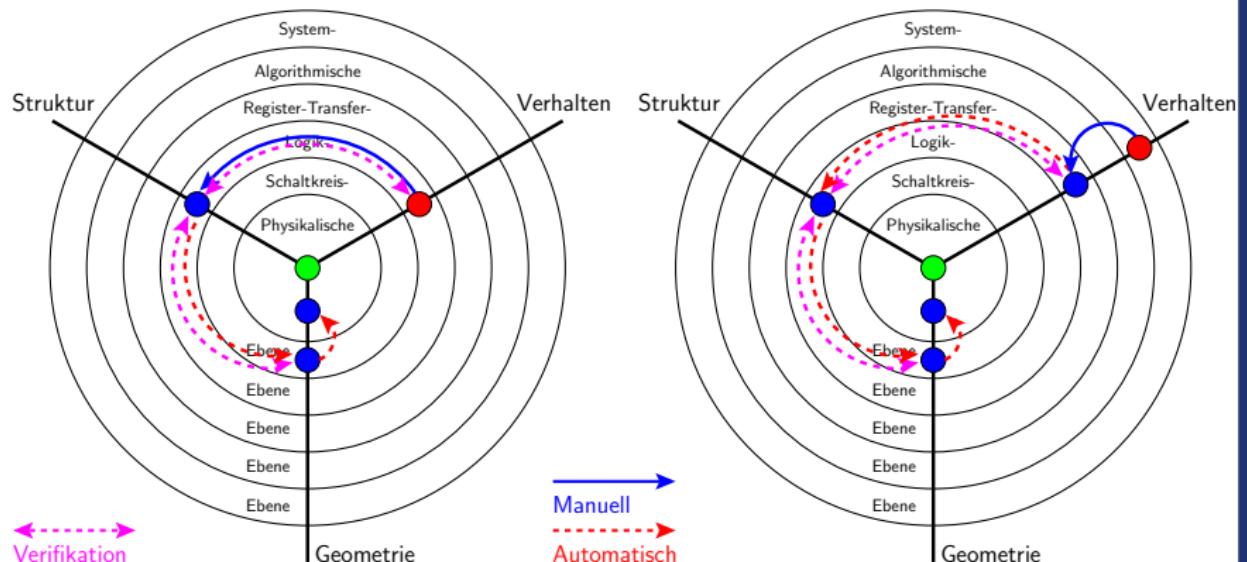
Traumvorstellung: Vollautomatische Layout-Synthese aus einer Verhaltensbeschreibung auf der Systemebene.

Übergang von der 1. zur 2. EDA-Generation



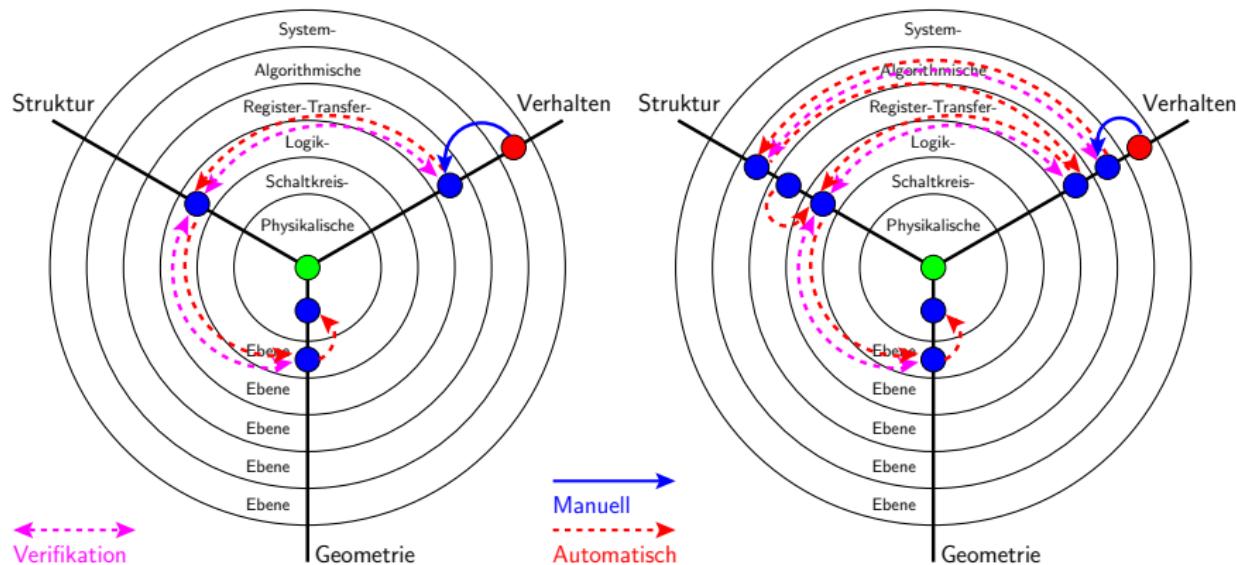
Erste Schritte der vollautomatischen Layout-Synthese

Übergang von der 2. zur 3. EDA-Generation



Vollautomatische Logik-Synthese

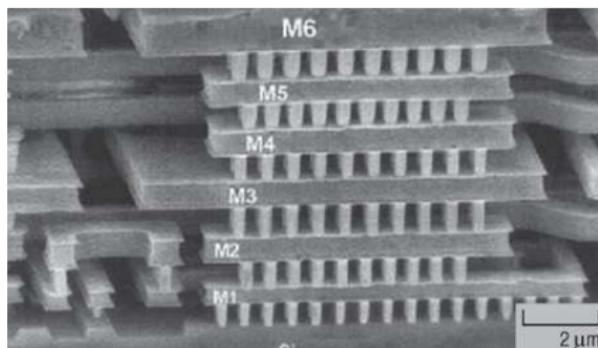
Übergang von der 3. zur 4. EDA-Generation



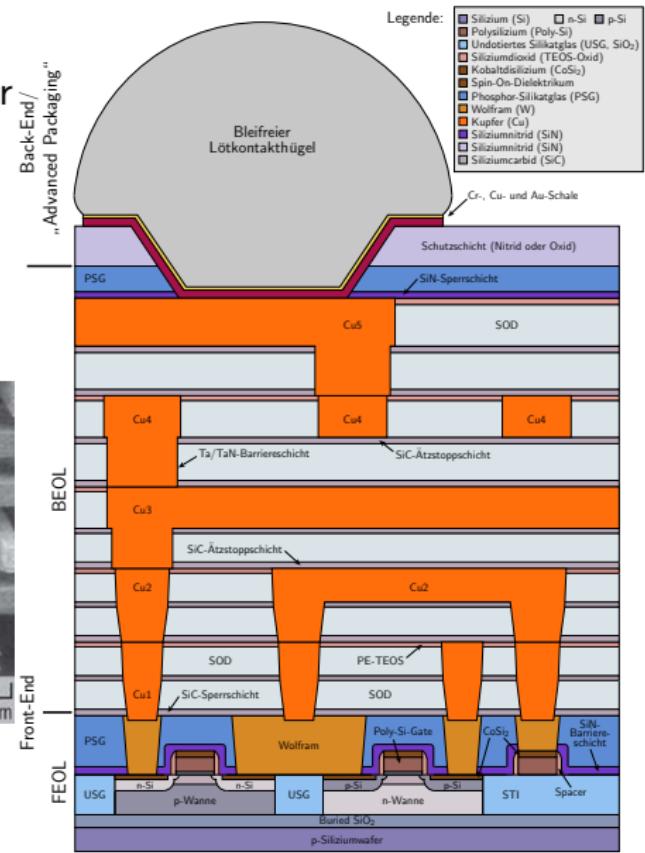
High-Level-Synthese und Verifikation, IP-Erzeugung und Wiederverwendung

Stand der Technik

Mikrofotografie und schematischer Querschnitt moderner integrierter Schaltkreise:



Quellen: UMC, Wikipedia



Hauptziel eines Entwurfs

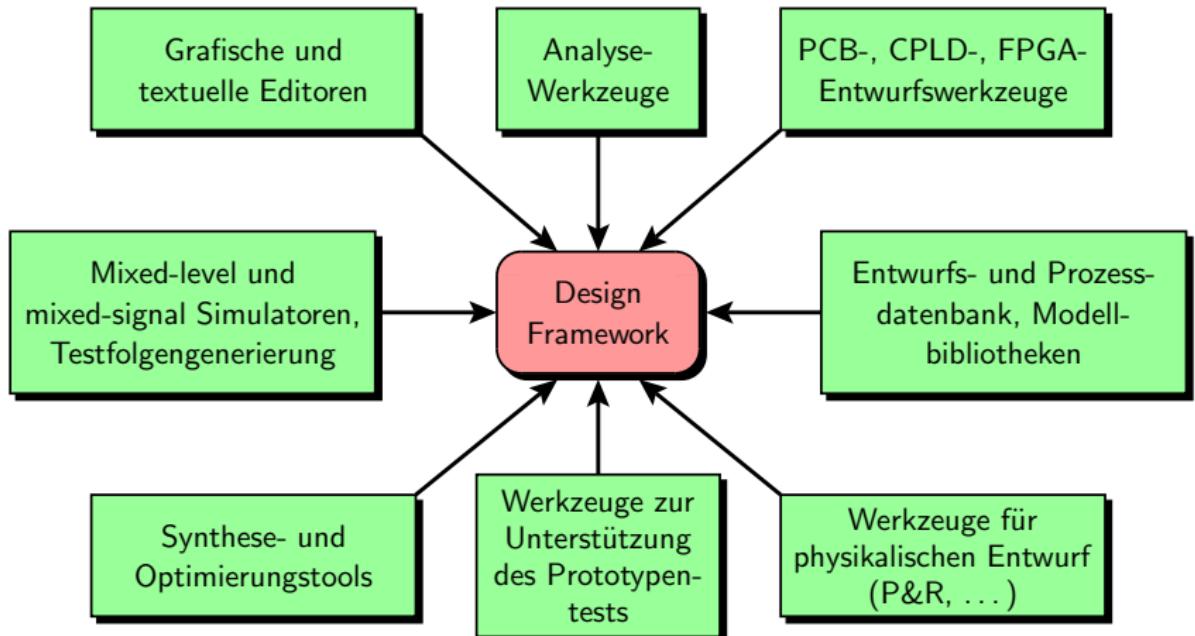
Schaffung eines fehlerfrei funktionierendes Produktes, das die Spezifikation erfüllt.

- ▶ Das Problem hat eine extrem hohe Komplexität, daher ist die Vision vom *Silicon Compiler* (Produkt aus der Spezifikation) nach wie vor nicht erfüllbar.
- ▶ Aufteilung in Teilprobleme, Schaffung von (gelegentlich „künstlichen“) Randbedingungen

Globale Optimierungsziele (Fläche, Leistungsbedarf, Taktfrequenz) resultieren in einer Reihe von Optimierungskriterien, die oft einander widersprechen: minimale Leitungslänge, minimale Anzahl von Vias usw.

- ▶ Heuristische Lösungen von Teilproblemen, die (hoffentlich) ein zufriedenstellendes Gesamtergebnis liefern
- ▶ Kein universelles CAD-Programm, sondern eine Vielzahl von Werkzeugen, die jeweils ein Teilproblem lösen

Wesentliche Bestandteile eines Entwurfssystems



Führende Anbieter von EDA-Software

Nicht herstellerspezifisch:

- ▶ Cadence Design Systems
- ▶ Synopsys
- ▶ Mentor Graphics (2017 von Siemens PLM übernommen)
- ▶ Magna Design Automation (2012 von Synopsys übernommen)

FPGA/CPLD Hersteller bieten ebenfalls eigene Lösungen an, die sich jedoch nur für den Entwurf mit den entsprechenden Schaltkreis-Familien eignen (herstellerspezifisch):

- ▶ Actel Corporation (2010 von Microsemi übernommen, Libero IDE)
- ▶ Altera Corporation (2015 von Intel übernommen, Quartus II bzw. Quartus Prime)
- ▶ Xilinx Inc. (2022 von AMD übernommen, Vivado Design Suite, ISE)

2. Synthese und Technologie-Abbildung

Begriffssystematik

Digitale Synthese, Logiksynthese (*logic synthesis*) ist die Umsetzung einer abstrakten Beschreibung (meistens RTL HDL) in eine (in der Regel nur aus Basiszellen bestehende) Netzliste. Abhängig vom Abstraktionsniveau und der Entwurfssicht (Struktur bzw. Verhalten) der Beschreibung wird gelegentlich auch von Algorithmen-, RT- bzw. High-Level-Synthese gesprochen. Entwurfszustand nach der Synthese ist eine Beschreibung auf Schaltkreisniveau (Struktur-Sicht).

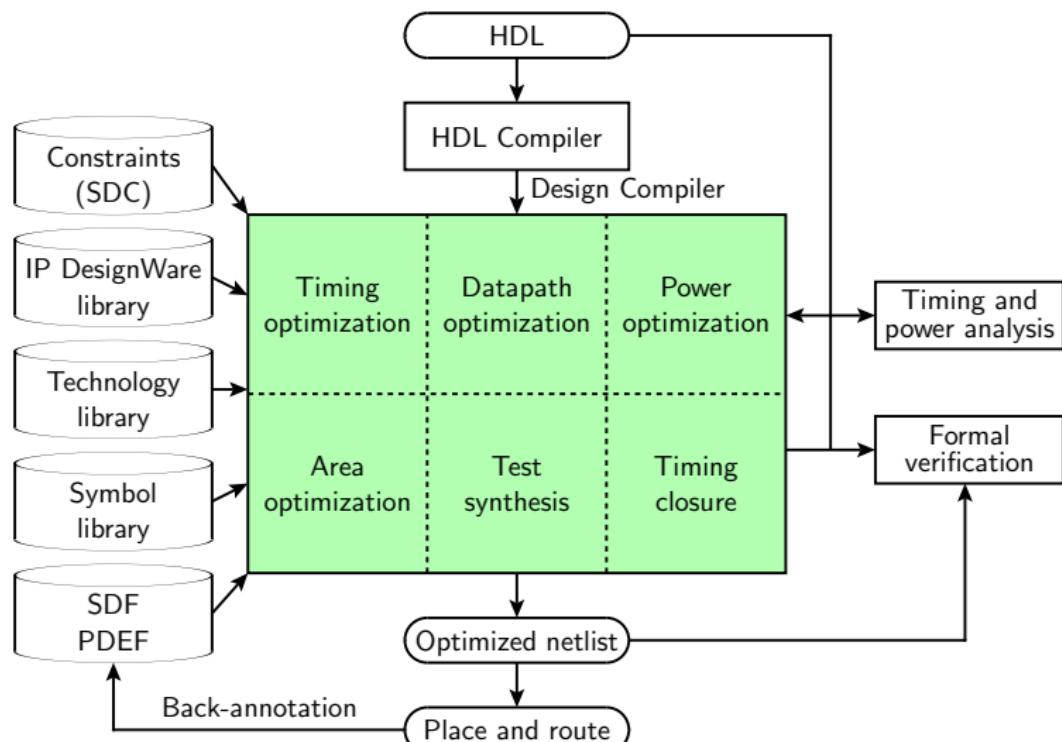
Technologie-Abbildung (*technology mapping*) ist Erzeugung einer Netzliste, die ausschließlich aus vorgegebenen Komponenten besteht (z. B. Basiszellen beim Standardzellen-Entwurf von ASIC). Technologie-Abbildung wird gelegentlich als Teil der Logik-Synthese betrachtet.

Als Teil der Logiksynthese wird auch Logikoptimierung betrachtet (wenngleich sie im Y-Diagramm einen separaten Entwurfsschritt darstellt).

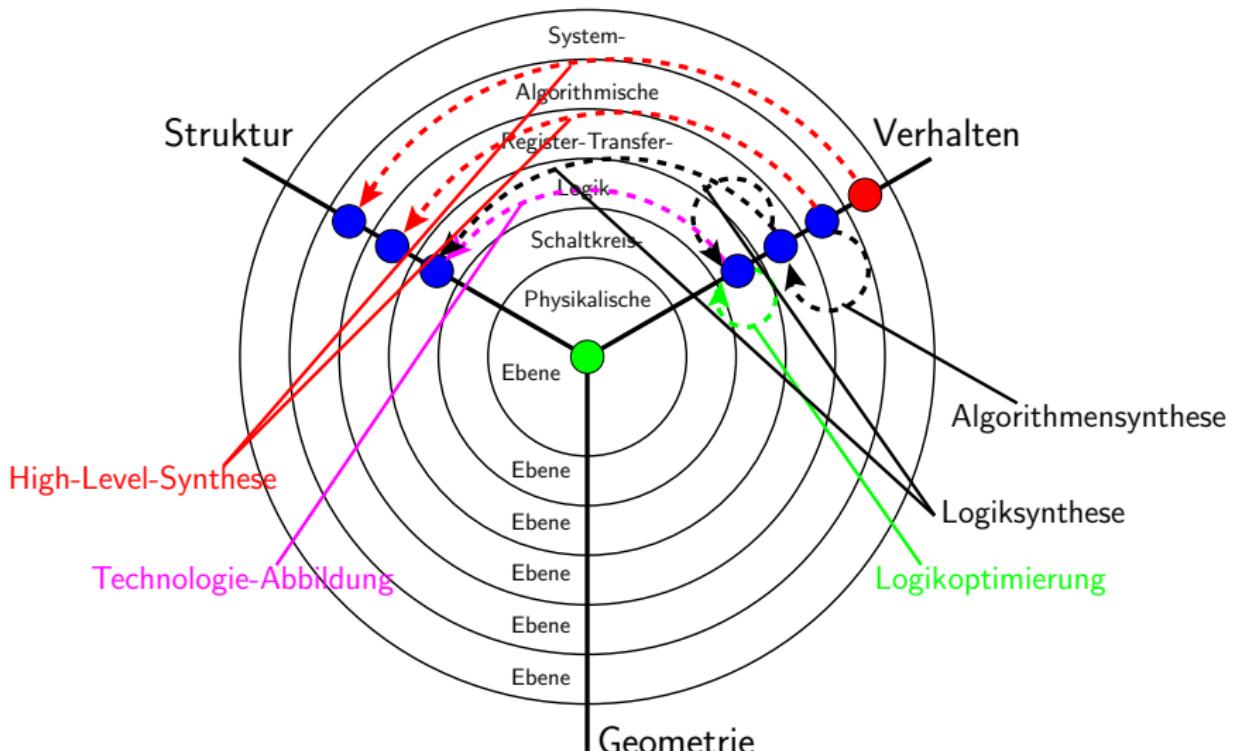
Logiksynthese vs. Layout-Synthese

- ▶ Beides Syntheseschritte
- ▶ Verbinden unterschiedliche Ebenen und Sichten: Logiksynthese ist ein Übergang von der Verhaltens- zur Struktursicht (bzw. ein Übergang zwischen verschiedenen Ebenen der Struktursicht) während Layout-Synthese immer in der Geometrie-Sicht endet und viele weitere Teilschritte beinhaltet:
 - ▶ Partitionierung
 - ▶ Floorplanning
 - ▶ Platzierung
 - ▶ Verdrahtung
 - ▶ Kompaktierung

Einbindung der Logiksynthese in der Gesamtentwurfsfluss am Beispiel von Synopsys Design Compiler



Synthese und Technologie-Abbildung im Y-Diagramm



Beispiel VHDL

Details in „Systementwurf mit VHDL“ betrachtet, hier nur eine Zusammenfassung wichtigster Konzepte:

- ▶ Funktionale Beschreibungen zur direkten Logik-Umsetzung, meistens als schaltalgebraische Gleichungen oder bedingte Signalzuweisungen (*conditional/selected assignment*)
 - ➡ Meistens zur Umsetzung von einfacheren Schaltnetzen
- ▶ Strukturelle Beschreibungen zur Abbildung von Hierarchien bzw. zur direkten Umsetzung von regulären Strukturen
- ▶ Verhaltensbeschreibungen zur Unterscheidung von sequentiellen und nebenläufigen Abläufen ➡ Beschreibung von Schaltwerken und anderen komplexeren Strukturen
- ▶ Aufteilung in Schnittstellen- und Funktionsbeschreibungen (ENTITY und ARCHITECTURE) sowie weitere Strukturierungsmittel (CONFIGURATION, PACKAGE, Funktionen, Prozeduren, Schleifen)

Vergleichbare Konzepte in Verilog-HDL

- ▶ Aufteilung in Schnittstellen- und Funktionsbeschreibung ist implizit: Nur eine Entwurfseinheit `module`
- ▶ Funktionale Beschreibungen durch Grundgatter (Bestandteil der Sprache) und nutzer-definierte Basiskomponenten in Tabellenform (*UDP, used defined primitives*)
- ▶ Strukturelle Beschreibungen durch Verwendung bereits definierter Module, Verknüpfung über (gleiche) Namen
- ▶ Verhaltensbeschreibungen durch Unterscheidung von blockierenden und nicht blockierenden Zuweisungen, sowie `initial` und `always` Prozessen (vergleichbar mit sequentiellen und nebenläufigen Anweisungen in VHDL)
- ▶ Weitere Strukturierungsmittel wie Tasks, Funktionen, Schleifen usw.

Beispiel: AND-Gatter in Verilog-HDL

```
module and_gate1 (c,a,b);      module and_gate3 (c,a,b);  
output c;                      output c;  
input a,b;                     input a,b;  
  
and and_instanz(c,a,b);        reg c;  
endmodule  
  
module and_gate2 (c,a,b);      always@(a or b)  
output c;                      begin  
input a,b;                     if (a) c = b;  
  
assign c = a & b;               else c = 1'b0;  
endmodule  
endmodule
```

(Drei äquivalente Beschreibungen)

Ein komplexeres Beispiel in Verilog-HDL

```

module example (out1,out2,in1,in2,in3);
  input  in1,in2,in3;
  output out1,out2;
  wire   w1,w2,w3;

  nor #3 gate1(w1,in1,in2);
  not #1 gate2(w2,in3);
  and #5 gate3(w3,w1,in3);
  nor #3 gate4(out2,w2,w3);
  sw  #5 sw1(out1,w1,w2);

endmodule

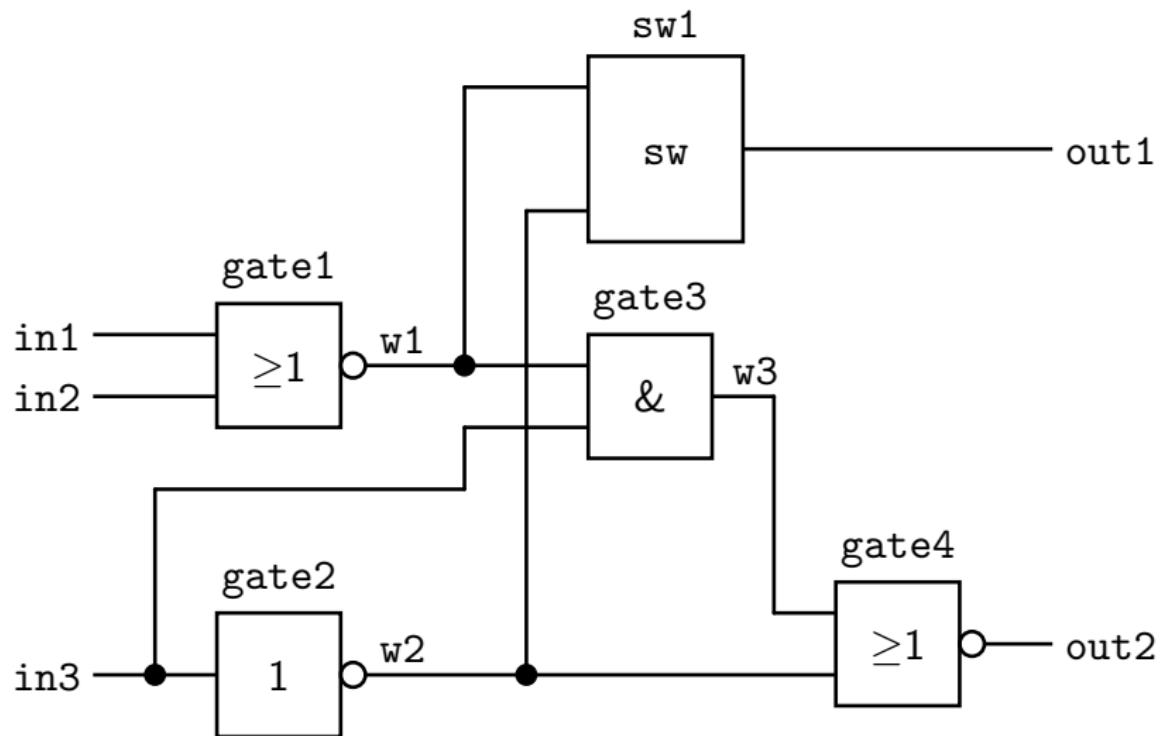
// r: steigende Flanke
// f: fallende Flanke
// ?: 0,1, oder x
// *: r oder f
// b: 0 oder 1, nicht x

  primitive sw (c,a,b);
    output c;
    input a,b;

    table
      // a  b  c Kommentar!
      0  0 : 0;
      0  1 : 1;
      1  ? : 1;
    end table
  endprimitive

```

Syntheseergebnis für das letzte Beispiel



Ein Schaltwerk am Beispiel eines Modulo-10-Zählers in Verilog-HDL

```
module mod10 (q,clk,res);
output [3:0]q;
input clk,res;
reg [3:0]q;

always @(posedge clk or posedge res)
begin
if (res)
#10 q=4'b0000;
else
if (q==4'b1001)#20 q=4'b0000;
else #20 q=q+1;
end
endmodule
```

Weitere Sprachelemente von Verilog-HDL

- ▶ Verkettung: `assign y={a,b};` (Bitbreite von y muss die Summe der Bitbreiten von a und b sein)
- ▶ Signalstärken: `supply0/1, strong0/1, pull0/1, weak0/1, medium0/1, small0/1, highz0/1` sowie Signalbasiswerte 0, 1, x und z
- ▶ Getrennte Angaben für HL- und LH-Verzögerungen, z. B.
`assign [3:0] (strong 1, pull 0) #(10,11) A = 0;`
- ▶ Blockierende und nichtblockierende Zuweisungen:

```

initial          initial
begin            begin
    a=#10 1; // 10      a<=#10 1; // 10
    b=#2  0; // 12      b<=#2  0; // 2
    c=#4  1; // 16      c<=#4  1; // 4
end              end

```

- ▶ Ereignissteuerung, Zu- und Abschaltung von Signaltreibern

Zusammenfassung

- ▶ Entwicklung von EDA-Werkzeugen
- ▶ Wiederholung der HDL-Grundkonzepte mit kurzen Beispielen in Verilog-HDL

Rechnergestützer Entwurf digitaler Systeme: Vorlesungs-Gesamtübersicht

1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme:

Gliederung der 4. Vorlesung

2. Synthese und Technologie-Abbildung

2.1. Einordnung

2.2. HDL-Synthese

2.3. Logiksynthese und Optimierung

2.3.1. Zweistufige Logik

2.3.2. Bündeloptimierung

2.3.3. ESPRESSO-Algorithmus

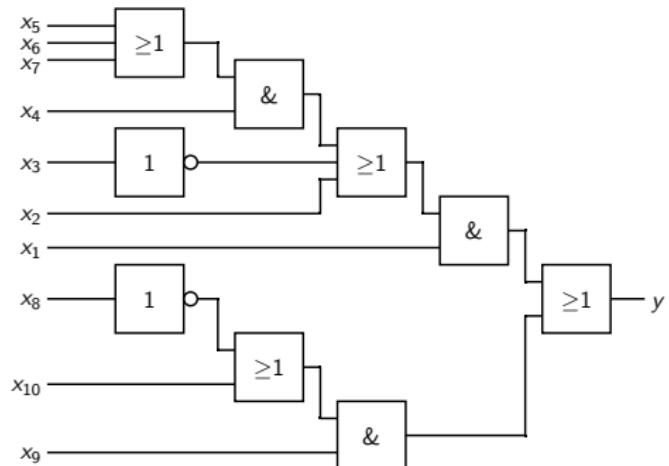
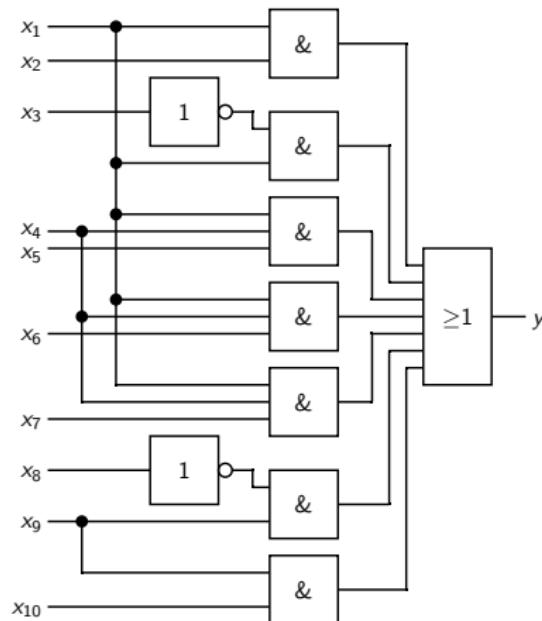
2.3.4. Mehrstufige Logik

2.3.5. XOR-Logik

2.4. Technologie-Abbildung

Zweistufige vs. mehrstufige Logik

Äquivalente Darstellung eines Schaltnetzes in zwei- und mehrstufiger Logik (Negation wird nicht als eine separate Logikstufe gezählt):



→ Ein Kompromiss zwischen Flächenbedarf und Schaltungstiefe.

Von zwei- zur mehrstufigen Logik

Naturgemäß ist die zweistufige Implementierung für die meisten (komplexen) Schaltungen nicht effizient. Bei einfacheren Schaltungen kann dagegen eine direkte Umsetzung der zweistufigen Darstellung mit Hilfe von PAL-, PLA oder (P)ROM-Bausteinen erfolgen. Nachfolgend werden die wichtigsten Grundlagen der Logikoptimierung wiederholt sowie einige neue Verfahren vorgestellt:

- ▶ Grundbegriffe und Verfahren der zweistufigen Logikoptimierung
- ▶ Gleichzeitige Optimierung von mehreren Schaltfunktionen
- ▶ Ansätze zur Optimierung mehrstufiger Logik
- ▶ ESPRESSO-Verfahren als Ursprung der meisten modernen Optimierungstechniken (sowohl für zwei- als auch für mehrwertige Logik)

Literal

Ein Literal \tilde{x} ist ein Teilausdruck einer Schaltfunktion, der aus einer einzigen Variablen x oder \bar{x} besteht: $\tilde{x} \in \{x, \bar{x}\}$.

Minimierungverfahren für zweistufige Logik

Zweistufige Darstellung kann unmittelbar aus der KDNF oder der KKNF erzeugt werden. Historisch wurden die ersten Optimierungsverfahren für zweistufige Logiksynthese entwickelt (*2-level logic synthesis, sum-of-products, SOP*):

	KV-Diagramme	QUINE-McCLUSKEY-Methode
Veröffentlichung	1952–1953	1955–1956
Anzahl der Eingänge	bis 6	beliebig
Ergebnis	Minimalform (Optimum)	
Laufzeit	—	exponentiell (<i>worst case</i>)
Wesentliche Merkmale	Graphisch orientiert	Sehr gut programmtechnisch umsetzbar

Zur Erinnerung: Einige Grundbegriffe (1)

Minterm

Sei $f(x_1, \dots, x_n)$ eine Schaltfunktion. Ein Minterm ist ein Ausdruck der Form $\bigwedge_{v=1}^n \tilde{x}_v$. Jeder Minterm hat nur bei einer einzigen Belegung seiner Schaltvariablen den Wert 1, sonst den Wert 0. Mit n Schaltvariablen können maximal 2^n verschiedene Minterme definiert werden.

Beispiel: Minterm $x_1 \wedge \bar{x}_2 \wedge x_3 \wedge \bar{x}_4 \wedge \bar{x}_5$ liefert bei der Belegung $(x_1, x_2, x_3, x_4, x_5) = (1, 0, 1, 0, 0)$ den Wert 1 und bei allen anderen Belegungen den Wert 0 (Minterm evaluiert bei dieser Belegung zu 1).

Maxterm

Sei $f(x_1, \dots, x_n)$ eine Schaltfunktion. Ein Maxterm ist ein Ausdruck der Form $\bigvee_{v=1}^n \tilde{x}_v$. Jeder Maxterm hat nur bei einer einzigen Belegung seiner Schaltvariablen den Wert 0, sonst den Wert 1. Mit n Schaltvariablen können maximal 2^n verschiedene Maxterme definiert werden.

Beispiel: Maxterm $x_1 \vee \bar{x}_2 \vee x_3 \vee \bar{x}_4 \vee \bar{x}_5$ liefert bei der Belegung $(x_1, x_2, x_3, x_4, x_5) = (0, 1, 0, 1, 1)$ den Wert 0 und bei allen anderen Belegungen den Wert 1 (Maxterm evaluiert bei dieser Belegung zu 0).

Einige Grundbegriffe (2)

Produktterm, Implikant, Primimplikant

Produktterm ist Konjunktion einiger Schaltvariablen (Literale). Ein Produktterm überdeckt (oder enthält) einen Minterm (anderen Produktterm), wenn er bei gleicher (gleichen) Variablenbelegung(en) wie dieser Minterm (der andere Produktterm) den Wert 1 liefert.

Implikant ist ein Produktterm, der nur Minterme überdeckt, die der Einsmenge entstammen.

Primimplikant ist ein Implikant, der von keinem anderen Implikanten überdeckt wird.

Beispiele: $a = \bar{x}_1 \bar{x}_2 \bar{x}_3$, $b = \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$,
 $c = \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4$, $d = x_1 x_2 x_3$, $e = x_1 x_2 \bar{x}_3$, $f = x_1 x_2$
sind Produktterme, a überdeckt b und c , f
überdeckt d und e . c , d , e und f sind Implikanten,
 f ist ein Primimplikant.

x_1	x_2	x_3	x_4	y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Einige Grundbegriffe (3)

Kernimplikant, Total redundanter Primimplikant

Kernimplikant (auch wesentlicher/essentieller Primimplikant) ist ein Primimplikant, der mindestens einen Minterm überdeckt, der von keinem anderen Primimplikanten überdeckt wird. Total redundanter Primimplikant ist ein Primimplikant der **nur** solche Minterme überdeckt, die auch von Kernimplikanten überdeckt werden.

Beispiele: $f = x_1x_2$ ist ein Kernimplikant, da Minterme $x_1x_2x_3x_4$ und $x_1x_2\bar{x}_3\bar{x}_4$ von keinem anderen Primimplikanten überdeckt werden.

$g = x_1\bar{x}_3x_4$ ist total redundant, weil $x_1x_2\bar{x}_3x_4$ vom Kernimplikanten f und $x_1\bar{x}_2\bar{x}_3x_4$ vom Kernimplikanten $h = \bar{x}_2\bar{x}_3x_4$ überdeckt sind.

x_1	x_2	x_3	x_4	y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Einige Grundbegriffe (4)

Partiell redundanter Primimplikant

ist ein Primimplikant der **nur** solche Minterme überdeckt, die auch von anderen Primimplikanten überdeckt werden (aber nicht ausschließlich von Kernimplikanten!)

Aufgabenstellung

Einen minimalen Satz von Primimplikanten finden, die alle Minterme der Einsmenge überdecken. Disjunktive Verknüpfung dieser Primimplikanten liefert eine Minimalform der Schaltfunktion.

- ▶ **Kernimplikanten** kommen **zwangsläufig** in der Minimalform vor
- ▶ **Total redundante Primimplikanten** kommen in der Minimalform **definitiv nicht** vor
- ▶ Minimalform ist eine disjunktive Verknüpfung von allen Kernimplikanten und n partiell redundanten Primimplikanten ($0 \leq n <$ Anzahl von partiell redundanten Primimplikanten)

Veranschaulichung von Grundbegriffen

mit Hilfe von KV-Diagrammen:

Index	x_4	x_3	x_2	x_1	y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

y	$\bar{x}_2\bar{x}_1$	\bar{x}_2x_1	x_2x_1	$x_2\bar{x}_1$	c
$\bar{x}_4\bar{x}_3$	a 1 ₀		0 ₁	0 ₃	1 ₂ a
\bar{x}_4x_3		1 ₄ b	1 ₅ b	1 ₇	1 ₆ a
x_4x_3		0 ₁₂	0 ₁₃	0 ₁₅	0 ₁₄
$x_4\bar{x}_3$		0 ₈	0 ₉	1 ₁₁ d	1 ₁₀ c

a ist essentiell (Kernimplikant) wegen 0

b ist essentiell wegen 5 und 7

c ist total redundant (2 ist von a und 10 ist von d überdeckt, a und d sind beide Kernimplikanten)

d ist essentiell wegen 11

$$\text{DMF: } y = \underbrace{\bar{x}_4\bar{x}_1}_{a} \vee \underbrace{\bar{x}_4x_3}_{b} \vee \underbrace{x_4\bar{x}_3x_2}_{d}$$

Veranschaulichung von Grundbegriffen

mit Hilfe von KV-Diagrammen (fortgesetzt):

Index	x_4	x_3	x_2	x_1	y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

y	$\bar{x}_2\bar{x}_1$	\bar{x}_2x_1	x_2x_1	$x_2\bar{x}_1$
$\bar{x}_4\bar{x}_3$	0_0	0_1	0_3	0_2
\bar{x}_4x_3	1_4 ^a	1_5 ^b	0_7	0_6
x_4x_3	0_{12}	1_{13} ^c	1_{15}	0_{14}
$x_4\bar{x}_3$	0_8	0_9	1_{11} ^d	0_{10}

^a ist essentiell wegen 4

^b und ^c sind partiell redundant

^d ist essentiell wegen 11

DMF: $y = \underbrace{\bar{x}_4x_3\bar{x}_2}_{a} \vee \underbrace{x_3\bar{x}_2x_1}_{b} \vee \underbrace{x_4x_2x_1}_{d}$

bzw. $y = \underbrace{\bar{x}_4x_3\bar{x}_2}_{a} \vee \underbrace{x_4x_3x_1}_{c} \vee \underbrace{x_4x_2x_1}_{d}$

Beispiel zum QUINE-McCLUSKEY-Verfahren

Ausgangspunkt ist die Schaltfunktion auf der Seite 96.

QUINESche Tabelle 0. Ordnung:

Dezimalindex	x_4	x_3	x_2	x_1	Primimplikant	Gruppe
0	0	0	0	0		0
2	0	0	1	0		1
4	0	1	0	0		
5	0	1	0	1		
6	0	1	1	0		2
10	1	0	1	0		
7	0	1	1	1		3
11	1	0	1	1		

Beispiel zum QUINE-McCLUSKEY-Verfahren (fortgesetzt)

QUINESche Tabelle 1. Ordnung:

Dezimalindex	x_4	x_3	x_2	x_1	Primimplikant	Gruppe
(0,2)	0	0	—	0		0
(0,4)	0	—	0	0		
(2,6)	0	—	1	0		
(2,10)	—	0	1	0	×	1
(4,5)	0	1	0	—		
(4,6)	0	1	—	0		
(5,7)	0	1	—	1		
(6,7)	0	1	1	—		2
(10,11)	1	0	1	—	×	

Beispiel zum QUINE-McCLUSKEY-Verfahren (fortgesetzt)

QUINESche Tabelle 2. Ordnung:

Dezimalindex	x_4	x_3	x_2	x_1	Primimplikant	Gruppe
$((0,2)(4,6))$	0	—	—	0	×	0
$((0,4)(2,6))$	0	—	—	0	×	→ redundant
$((4,5)(6,7))$	0	1	—	—	×	1
$((4,6)(5,7))$	0	1	—	—	×	→ redundant

Primimplikantentabelle:

Dezimalindex	Minterme								Kernimplikant
	0	2	4	5	6	7	10	11	
$(2,10)$		+					+		
$(10,11)$							+	⊕	×
$((0,2)(4,6))$	⊕	+	+		+				×
$((4,5)(6,7))$			+	⊕	+	⊕			×

Konstruktion der Minimalüberdeckung

Total redundanter Primimplikant (2,10) wird gestrichen, die Kernimplikanten müssen beibehalten werden.

- Die Minimalform der Schaltfunktion ist die Disjunktion der Kernimplikanten (10,11), ((0,2)(4,6)) und ((4,5)(6,7)):

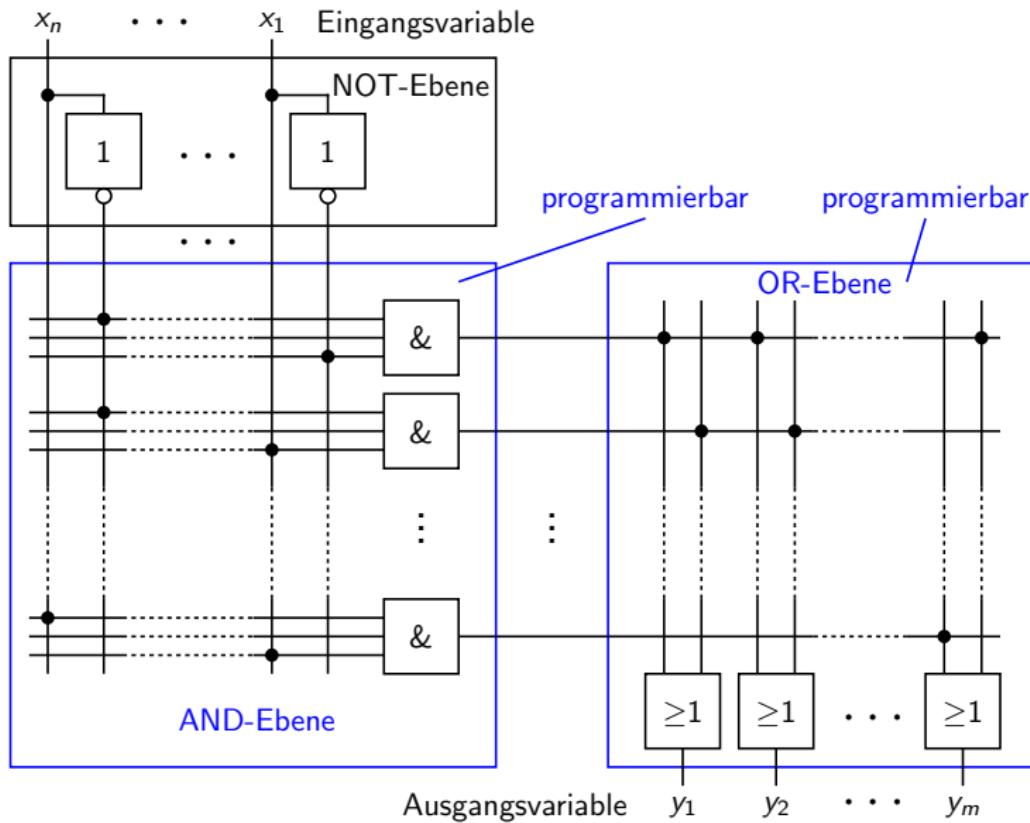
$$y = \underbrace{\bar{x}_4 \bar{x}_1}_{((0,2)(4,6))} \vee \underbrace{\bar{x}_4 x_3}_{((4,5)(6,7))} \vee \underbrace{x_4 \bar{x}_3 x_2}_{(10,11)}$$

Es gibt keine partiell redundanten Primimplikanten, die Minimalform ergibt sich allein aus der disjunktiven Verknüpfung der Kernimplikanten (das gilt nicht grundsätzlich!)

- Gleiches Ergebnis, wie bei einer Vereinfachung mit KV-Diagramm.

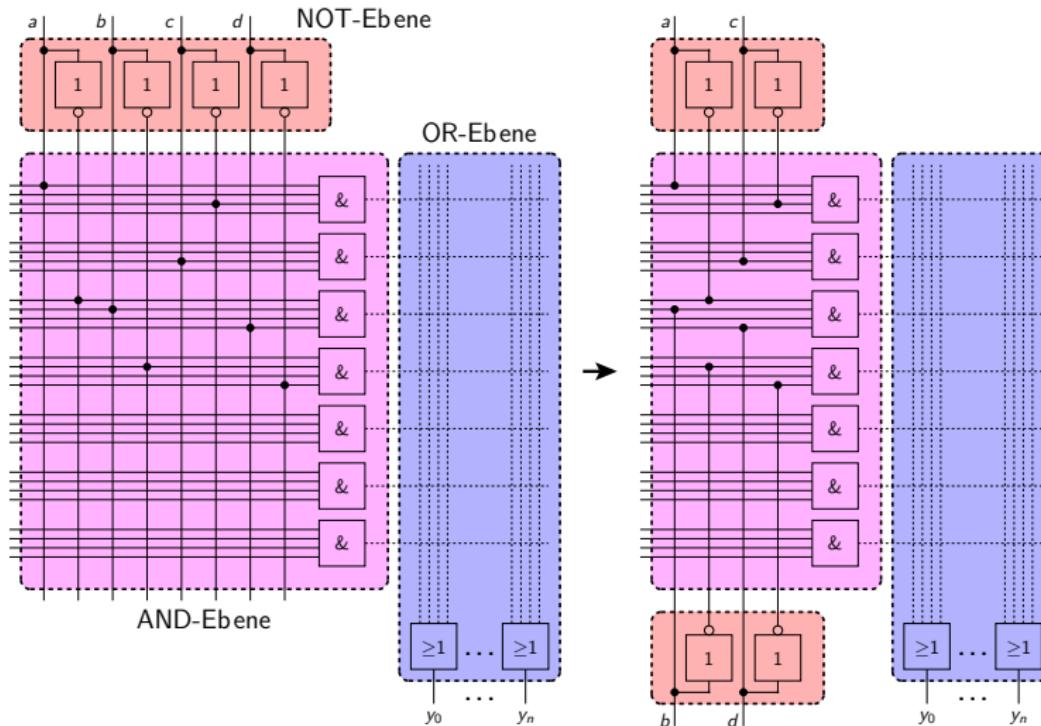
Im Allgemeinen ist das Problem der Findung einer minimalen Überdeckung nicht trivial und sehr rechenintensiv.

Zur Erinnerung: Grundstruktur eines PLA-Bausteins



Faltung

Weitere Optimierungsmöglichkeiten bei PLA-Implementierung:
 Faltung in AND- und OR-Ebene (*folding in AND- and OR-plane*)



Das Prinzip der Faltung in der AND-Ebene

- ▶ In der Regel wird nicht jede Eingangsvariable (Literal) in jedem Primimplikanten benötigt
- ➡ Findet man Paare von Literalen, die gemeinsam eine Spalte nutzen können, so kann die für die AND-Ebene benötigte Fläche zusätzlich reduziert werden.
- ▶ Annahme: negierte und nicht negierte Variablen kommen aus einer Richtung (Einschränkung der Lösungsmenge)
- ▶ Es existieren zwei Zuordnungsvarianten:
 - ▶ Gerade Zuordnung: ein Paar wird immer entweder aus 2 negierten oder aus 2 nichtnegierten Variablen gebildet (wie im letzten Beispiel)
 - ▶ Vertauschte (ungerade) Zuordnung: ein Paar wird immer aus je einer negierten und einer nichtnegierten Variablen gebildet

Faltung in AND-Ebene: Grundsätzliche Herangehensweise

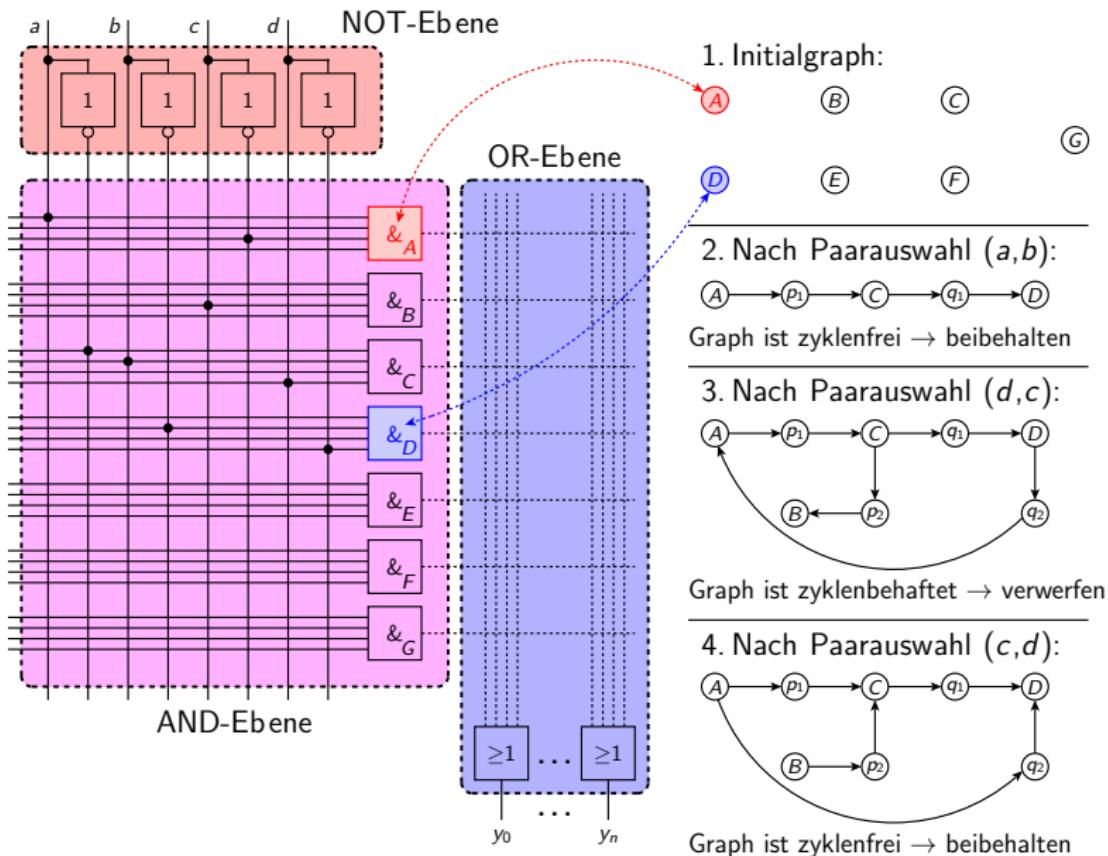
- ▶ Ein Paar von Literalen aussuchen, dadurch entstehen zwei Trennlinien, z. B. bei gerader Zuordnung p_i für nichtnegierte und q_i für negierte Literale.
- ➡ Die Anordnung der UND-Gatter wird entsprechend der Relation „oberhalb/unterhalb der Trennlinien p_i und q_i “ angepasst.
- ▶ Jedes weitere Paar von Literalen sorgt für zusätzliche Trennlinien und weitere Verfeinerung der Anordnung der UND-Gatter (bis alle Positionen festgelegt sind).
- ➡ Auswahl der Paare ist nicht zwangsläufig deterministisch, die endgültige Anordnung hängt von der Wahlreihenfolge ab
- ▶ Ein Paar $(\tilde{x}_i, \tilde{x}_j)$ ist bei gerader Zuordnung nicht zulässig, wenn ein Gatter sowohl x_i als auch x_j bzw. sowohl \bar{x}_i als auch \bar{x}_j verknüpft (entsprechend x_i und \bar{x}_j bzw. \bar{x}_i und x_j bei ungerader Zuordnung).

Mögliche Lösungsstrategie

Graphentheoretischer Ansatz (Knoten \cong UND-Gatter, Kanten \cong oberhalb/unterhalb Relation):

1. Initialgraph hat keine Kanten (nur N Knoten entsprechend der Anzahl der UND-Gatter)
2. Durch Auswahl eines Paares $(\tilde{x}_i, \tilde{x}_j)$ werden dem Graphen zwei temporäre Knoten p_i und q_j hinzugefügt (Schnitlinien) für nichtnegierte und negierte Literale:
 - ▶ Kante von Knoten r zu Knoten p_i , wenn Term r (UND-Verknüpfung r) x_i benötigt (Gatter oberhalb der Trennstelle)
 - ▶ Kante von Knoten p_i zu Knoten r , wenn Term r (UND-Verknüpfung r) x_j benötigt (Gatter unterhalb der Trennstelle)
 - ▶ Entsprechende Kantenbildung mit q_j und negierten Literalen
3. Ist der temporäre Graph zyklenfrei, so wird er beibehalten (das Paar ist zulässig), ansonsten wird er verworfen (das Paar ist nicht zulässig).

Beispiel zur Ermittlung der Lösung auf Seite 103



Beispiel zur Ermittlung der Lösung auf Seite 103 (fortgesetzt)

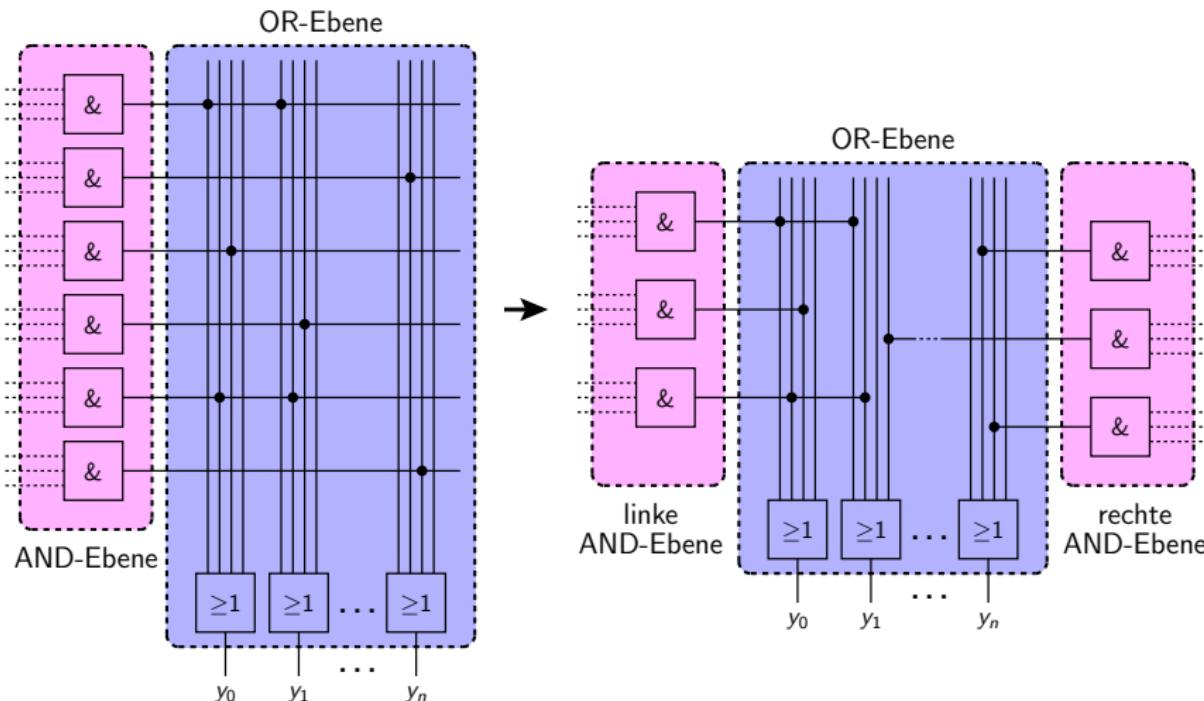
- ▶ Die ermittelten Paare sind (\tilde{a}, \tilde{b}) sowie (\tilde{c}, \tilde{d}) , womit alle Eingangsvariablen sortiert sind
- Endgültige Lösung, die einer Faltung „ \tilde{a} und \tilde{c} oben, \tilde{b} und \tilde{d} unten“ (oder spiegelverkehrt) entspricht.
- ▶ Paare (\tilde{b}, \tilde{d}) und (\tilde{d}, \tilde{b}) führen sofort zu den Zyklen in den Graphen, da UND-Gatter C b mit d sowie UND-Gatter D \bar{b} mit \bar{d} verknüpfen.
- Diese Paare können auch ohne Aufbau der Graphen sofort ausgeschlossen werden (siehe entsprechende Bedingung auf der Seite 105).

Die Reihenfolge der Paarauswahl bestimmt die Lösung, so könnten durch andere Paarbildung auch Lösungen $(\tilde{a}, \tilde{d}), (\tilde{c}, \tilde{b})$ oder $(\tilde{d}, \tilde{a}), (\tilde{b}, \tilde{c})$ ermittelt werden.

- Das Verfahren garantiert im Allgemeinen keine Optimallösung.

Faltung in OR-Ebene

Ähnliches Muster (jedoch ohne negierte Variablen), da in der Regel nicht jeder Produktterm von jedem ODER-Gatter benötigt wird:



Hintergrund

Bis jetzt wurden Verfahren zur Optimierung einzelner Schaltfunktionen diskutiert. In der Regel beinhaltet eine Schaltung eine Vielzahl von Schaltfunktionen:

- ▶ arithmetische und logische Schaltungen im Datenpfad
- ▶ Kontrollstrukturen im Steuerpfad
- ▶ Übergangs- und Ausgabefunktionen bei Schaltwerken
- ▶ ...

Prinzipiell kann für jede Schaltfunktion eine optimale oder nahezu optimale Lösung mit den vorgestellten Verfahren ermittelt werden. Die Summe von optimalen Teillösungen kann jedoch stark suboptimal sein.

- ➡ Es werden Verfahren benötigt, die einen Bündel von Schaltfunktionen gleichzeitig (d. h. unter Berücksichtigung von Synergieeffekten) optimieren können.

Ein triviales Beispiel

y_1	$\bar{x}_2\bar{x}_1$	\bar{x}_2x_1	x_2x_1	$x_2\bar{x}_1$
\bar{x}_3	0	0	0	1
x_3	0	1 <i>a</i>	1	1 <i>b</i>

y_2	$\bar{x}_2\bar{x}_1$	\bar{x}_2x_1	x_2x_1	$x_2\bar{x}_1$
\bar{x}_3	1	0	0	0
x_3	1 <i>c</i>	1 <i>a</i>	1	0

$$y_1 = \underbrace{x_3x_1}_{a} \vee \underbrace{x_2\bar{x}_1}_{b}$$

$$y_2 = \underbrace{x_3x_1}_{a} \vee \underbrace{\bar{x}_2\bar{x}_1}_{c}$$

a ist ein gemeinsamer Primimplikant (Kernimplikant), so dass bei schaltungstechnischer Implementierung ein UND-Gatter eingespart werden kann (Verzweigung des Ausgangs der $x_3 \wedge x_1$ Verknüpfung).

- Resultierende Schaltung besteht aus 3 UND- und 2 ODER-Verknüpfungen mit je zwei Eingängen.

Ein etwas weniger offensichtliches Beispiel

y_1	$\bar{x}_2\bar{x}_1$	\bar{x}_2x_1	x_2x_1	$x_2\bar{x}_1$
\bar{x}_3	0	0	0	1 <i>b</i>
x_3	0	0	1 <i>a</i>	1

y_2	$\bar{x}_2\bar{x}_1$	\bar{x}_2x_1	x_2x_1	$x_2\bar{x}_1$
\bar{x}_3	0	1 <i>c</i>	1	<i>d</i> 1
x_3	0	0	0	0

$$y_1 = \underbrace{x_3x_2}_{a} \vee \underbrace{x_2\bar{x}_1}_{b}$$

$$y_2 = \underbrace{\bar{x}_3x_1}_{c} \vee \underbrace{\bar{x}_3x_2}_{d}$$

y_1	$\bar{x}_2\bar{x}_1$	\bar{x}_2x_1	x_2x_1	$x_2\bar{x}_1$
\bar{x}_3	0	0	0	1 <i>b</i>
x_3	0	0	1 <i>a</i>	1

y_2	$\bar{x}_2\bar{x}_1$	\bar{x}_2x_1	x_2x_1	$x_2\bar{x}_1$
\bar{x}_3	0	1 <i>c</i>	1	1 <i>b</i>
x_3	0	0	0	0

$$y_1 = \underbrace{x_3x_2}_{a} \vee \underbrace{\bar{x}_3x_2\bar{x}_1}_{b}$$

$$y_2 = \underbrace{\bar{x}_3x_1}_{c} \vee \underbrace{\bar{x}_3x_2\bar{x}_1}_{b}$$

Einzel- vs. Bündelminimierung

$$y_1 = x_3x_2 \vee x_2\bar{x}_1 \quad y_2 = \bar{x}_3x_1 \vee \bar{x}_3x_2 \quad \text{bei Einzelminimierung}$$

$$y_1 = x_3x_2 \vee \bar{x}_3x_2\bar{x}_1 \quad y_2 = \bar{x}_3x_1 \vee \bar{x}_3x_2\bar{x}_1 \quad \text{bei Bündelminimierung}$$

Einzelminimierung: 4 UND-Gatter und 2 ODER-Gatter mit je 2 Eingängen

Bündelminimierung: 2 UND-Gatter mit je 2 Eingängen, 1 UND-Gatter mit 3 Eingängen und 2 ODER-Gatter mit je 2 Eingängen

Die Umformungen (bei Einzelminimierung)

$$y_1 = x_3x_2 \vee x_2\bar{x}_1 = x_2 \wedge (x_3 \vee \bar{x}_1) \quad \text{und} \quad y_2 = \bar{x}_3x_1 \vee \bar{x}_3x_2 = \bar{x}_3 \wedge (x_1 \vee x_2)$$

liefern eine noch bessere Lösung, sind jedoch nicht generell anwendbar sondern auf die Beschaffenheit der konkreten Schaltfunktionen zurück zu führen.

Ein komplexeres Beispiel („DT2“, Schaltwerksynthese)

z_0^+	$\bar{z}_1\bar{z}_0$	\bar{z}_1z_0	z_1z_0	$z_1\bar{z}_0$
$\bar{x}\bar{z}_2$	1 <i>a</i>	0	1 <i>b</i>	0
$\bar{x}z_2$	1	0	— <i>c</i>	1
xz_2	0	0	—	0
$x\bar{z}_2$	0	0	0	1 <i>d</i>

z_1^+	$\bar{z}_1\bar{z}_0$	\bar{z}_1z_0	z_1z_0	$z_1\bar{z}_0$
$\bar{x}\bar{z}_2$	0	1	0	1 <i>f</i>
$\bar{x}z_2$	0	1	—	0
xz_2	0	1 <i>e</i>	—	0
$x\bar{z}_2$	0	1	0	1 <i>f</i>

z_2^+	$\bar{z}_1\bar{z}_0$	\bar{z}_1z_0	z_1z_0	$z_1\bar{z}_0$
$\bar{x}\bar{z}_2$	0	0	1 <i>h</i>	0
$\bar{x}z_2$	0	0	— <i>i</i>	1
xz_2	0	1 <i>g</i>	—	0
$x\bar{z}_2$	0	1	1	0

$$\begin{aligned}
 z_2^+ &= \underbrace{xz_0}_{g} \vee \underbrace{\bar{x}z_1z_0}_{h} \vee \underbrace{\bar{x}z_2z_1}_{i} \\
 z_1^+ &= \underbrace{\bar{z}_1z_0}_{e} \vee \underbrace{\bar{z}_2z_1\bar{z}_0}_{f} \\
 z_0^+ &= \underbrace{x\bar{z}_1\bar{z}_0}_{a} \vee \underbrace{\bar{x}z_1z_0}_{b} \vee \underbrace{\bar{x}z_2z_1}_{c} \vee \underbrace{x\bar{z}_2z_1\bar{z}_0}_{d}
 \end{aligned}$$

Block *h* ist nicht optimal, dafür aber mit dem Block *b* identisch (erspart ein AND-Gatter durch Wiederverwendung).

Algorithmisches Optimierungsverfahren

Das Verfahren ist allgemein als „Identifikation gemeinsam genutzter Implikanten“ bekannt (*isolating shared implicants*) und wurde bisher mit Hilfe von KV-Diagrammen veranschaulicht. Für eine algorithmische Umsetzung ist eine formalisierte Vorgehensweise besser geeignet.

- Das Bündeloptimierungsverfahren nach Y.-H. SU und D. L. DIETMEYER (“*Computer Reduction of Two-Level, Multiple-Output Switching Circuits*”, IEEE Transactions on Computers, volume 18, number 1, pp. 58–63, January 1969)

Ein formalisiertes Basisverfahren, das vielfach nachgenutzt und verbessert wurde.

Funktionsprinzip: Darstellung des Funktionsbündels in Form einer Wahrheitwertetabelle oder KDNF und systematische Eliminierung der Redundanz, wobei automatisch ein Schaltnetz erzeugt wird.

Drei Minimierungsansätze

Das Verfahren nach Y.-H. SU und D. L. DIETMEYER basiert auf drei Minimierungsansätzen:

1. Minterme mit **komplett identischen** Ausgangsbelegungen können verschmolzen werden, wenn sie sich nur durch Belegung einer einzelnen Eingangsvariablen unterscheiden (wie QMCV, jedoch mit mehreren Ausgangswerten) ➔ trivial
2. Minterme, die bezüglich **eines oder mehrerer (aber nicht aller)** Ausgänge gleiche Belegungen aufweisen können partiell (d. h. nur bezüglich dieser bestimmten Ausgänge) verschmolzen werden, wenn sie sich nur durch Belegung einer einzelnen Eingangsvariablen unterscheiden.
3. Sind alle Verschmelzungen nach Punkt 1 und 2 durchgeführt, so können bestimmte Ausgangsbelegungen von total redundanten Primimplikanten stammen. Diese werden systematisch gesucht und eliminiert.

Vorbemerkungen zum Verfahren nach Y.-H. SU und D. L. DIETMEYER

- ▶ Ansatz 1 minimiert sowohl die Anzahl der UND-Gatter als auch die Anzahl der Eingänge von ODER-Gattern, Ansätze 2 und 3 dagegen nur die Anzahl der Eingänge von ODER-Gattern
- ▶ Nach Anwendung des Ansatzes 2 kann ein wiederholtes Anwenden des Ansatzes 1 möglich werden.
- ▶ Optimale Lösung wird nur dann erreicht, wenn alle Verschmelzungs- und Eliminierungsoptionen getestet werden, d. h. die Rechenzeit steigt schlimmstenfalls mit der Fakultätsfunktion!
- ➡ Verbesserung des Laufzeitverhaltens durch heuristische Verfahren.

Beispiel zur Veranschaulichung

Minterme bzw. Implikanten	Eingangsbelegung			Ausgangsbelegung	
	x_2	x_1	x_0	y_1	y_0
K_0	0	0	0	1	1
K_1	0	0	1	1	1
K_2	0	1	0	1	1
K_3	0	1	1	1	—
K_4	1	0	0	—	1
K_5	1	0	1	1	0
K_6	1	1	0	0	0
K_7	1	1	1	0	0

Implementierungsaufwand als Normalformschaltnetz: 6 UND-Gatter mit je 3 Eingängen, 1 ODER-Gatter mit 4 Eingängen und 1 ODER-Gatter mit 5 Eingängen

Anwendung des 1. Ansatzes

$$K_0 + K_1 : 000 \mid 11 + 001 \mid 11 \Rightarrow K_{0,1} : 0 \ 0 - \mid 11$$

$$K_0 + K_2 : 000 \mid 11 + 010 \mid 11 \Rightarrow K_{0,2} : 0 - 0 \mid 11$$

Folgen: 2 UND-Gatter mit 2 Eingängen statt 3 UND-Gatter mit 3 Eingängen, je ein Eingang weniger bei den ODER-Gattern.

Auch *don't care* Terme werden bei den Verschmelzungen berücksichtigt (hier aus Platzgründen weggelassen). Wichtig: Verschmolzene Zeilen werden gestrichen!

- Es spielen nur identische Ausgangsbelegungen eine Rolle, die mindestens eine „1“ oder mindesten ein „-“ enthalten.

Durch Verschmelzung entstandene Primimplikanten können an weiteren Verschmelzungen teilnehmen (wie bei QMCV, jedoch mit mehreren Ausgängen gleichzeitig).

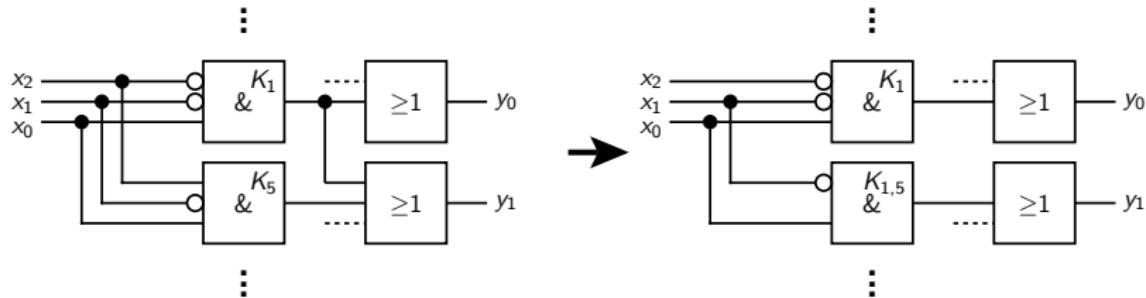
Anwendung des 2. Ansatzes

$$K_1 + K_5 : 001 \mid 11 + 101 \mid 10 \rightarrow K_{1,5} : -01 \mid 10$$

K_1 bleibt erhalten!

Folgen: Je ein Eingang weniger bei einem ODER- und UND-Gatter.

Auch *don't care* Terme werden bei den Verschmelzungen berücksichtigt (hier aus Platzgründen weggelassen). Wichtig: Die Zeile mit einem durch die Verschmelzung nicht abgedeckten Primimplikanten bleibt erhalten!



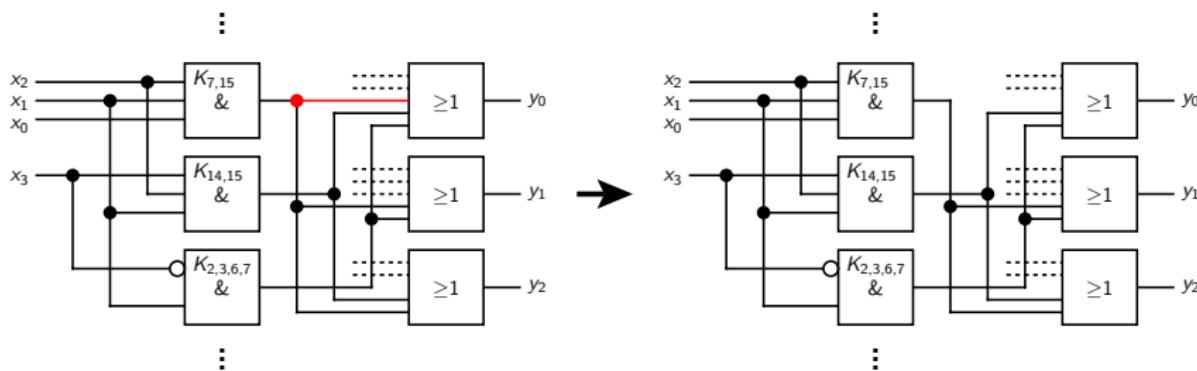
Anwendung des 3. Ansatzes

Minterme bzw. Implikanten	Eingangsbelegung				Ausgangsbelegung		
	x_3	x_2	x_1	x_0	y_2	y_1	y_0
...	
$K_{7,15}$	—	1	1	1	1	1	1
$K_{14,15}$	1	1	1	—	1	0	1
$K_{2,3,6,7}$	0	—	1	—	0	1	1
...	

Primimplikant $K_{7,15}$ ist bezüglich des Ausgangs y_0 total redundant (durch $K_{14,15}$ und $K_{2,3,6,7}$ überdeckt).

- In der Zeile $K_{7,15}$ kann in der Spalte y_0 ohne Veränderung der Schaltfunktion die „1“ invertiert werden (erspart einen Eingang eines ODER-Gatters).

Vereinfachung des Schaltnetzes im letzten Beispiel



Eingangsbelegung $(x_3, x_2, x_1, x_0) = (0, 1, 1, 1)$ ist durch die Primimplikanten $K_{7,15}$ und $K_{2,3,6,7}$ überdeckt, Eingangsbelegung $(x_3, x_2, x_1, x_0) = (1, 1, 1, 1)$ ist durch die Primimplikanten $K_{7,15}$ und $K_{14,15}$ überdeckt, Ausgang y_0 liefert „1“ in allen drei den Primimplikanten entsprechenden Zeilen.

→ Primimplikant $K_{7,15}$ ist bezüglich y_0 total redundant.

Anmerkungen zum Minimierungsverfahren nach Y.-H. SU und D. L. DIETMEYER

- ▶ Verfahren ist sehr gut formalisierbar (ähnlich dem QMCV-Verfahren), auf die Angabe des vollständigen Ablaufs wird hier aus Zeit- und Platzgründen verzichtet
- ➡ Details findet man in der Originalveröffentlichung
- ▶ Laufzeitverhalten ist (insbesondere für größere Probleme) sehr ungünstig, z. B. müssen alle Spaltenkombinationen auf Verschmelzungen untersucht werden, d. h. bei m Ausgängen und einer Auswahl von $N \leq m$ Spalten k Spaltenkombinationen mit $k = \binom{m}{N}$.
- ➡ Verzicht auf Optimumssuche, Einschränkung des Lösungsraums durch Heuristiken, z. B. ESPRESSO-Verfahren

Zusammenfassung

- ▶ Methoden zur Synthese und Optimierung zweistufiger Logik
- ▶ Faltung in UND- und ODER-Ebene bei zweistufiger Logik
- ▶ Gleichzeitige Optimierung mehrerer Schaltfunktionen

Rechnergestützer Entwurf digitaler Systeme: Vorlesungs-Gesamtübersicht

1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme:

Gliederung der 5. Vorlesung

2. Synthese und Technologie-Abbildung

2.1. Einordnung

2.2. HDL-Synthese

2.3. Logiksynthese und Optimierung

2.3.1. Zweistufige Logik

2.3.2. Bündeloptimierung

2.3.3. ESPRESSO-Algorithmus

2.3.4. Mehrstufige Logik

2.3.5. XOR-Logik

2.4. Technologie-Abbildung

Geschichte

- ▶ Ergebnis der Forschungsarbeiten von IBM und University of California at Berkeley (Anfang der 80er des letzten Jahrhunderts)
- ▶ Erste Version um 1982 (in Programmiersprache APL), erste Implementierung in Programmiersprache C in 1984
- ▶ 1986–1987 Ergänzungen und Erweiterungen (mehrwertige Logik, verbessertes Laufzeitverhalten, Hinzufügen von ESPRESSO-EXACT)
- ▶ Die aktuelle frei verfügbare Version ist um 1990 erschienen (auch als Bestandteil des SIS-Pakets verfügbar)
- ▶ Viele kommerziell verfügbare Werkzeuge verwenden Abwandlungen bzw. Weiterentwicklungen von ESPRESSO

Interne Darstellung von Schaltfunktionen

Positional Cube Notation (PCN)

- ▶ Negierte Variable ist mit „10“ kodiert, nichtnegierte Variable ist mit „01“ kodiert, *don't care* ist mit „11“ kodiert
- ▶ Nach einmaliger Festlegung der Reihenfolge von Variablen werden die Belegungen als Bitvektoren gespeichert

Literal	Kodierung
x	01
\bar{x}	10
—	11

x_1	x_2	x_3	Minterme	PCN
0	0	0	$\bar{x}_1 \wedge \bar{x}_2 \wedge \bar{x}_3$	10 10 10
0	0	1	$\bar{x}_1 \wedge \bar{x}_2 \wedge x_3$	10 10 01
0	1	0	$\bar{x}_1 \wedge x_2 \wedge \bar{x}_3$	10 01 10
0	1	1	$\bar{x}_1 \wedge x_2 \wedge x_3$	10 01 01
1	0	0	$x_1 \wedge \bar{x}_2 \wedge \bar{x}_3$	01 10 10
1	0	1	$x_1 \wedge \bar{x}_2 \wedge x_3$	01 10 01
1	1	0	$x_1 \wedge x_2 \wedge \bar{x}_3$	01 01 10
1	1	1	$x_1 \wedge x_2 \wedge x_3$	01 01 01

Interpretation von *don't care* Notation

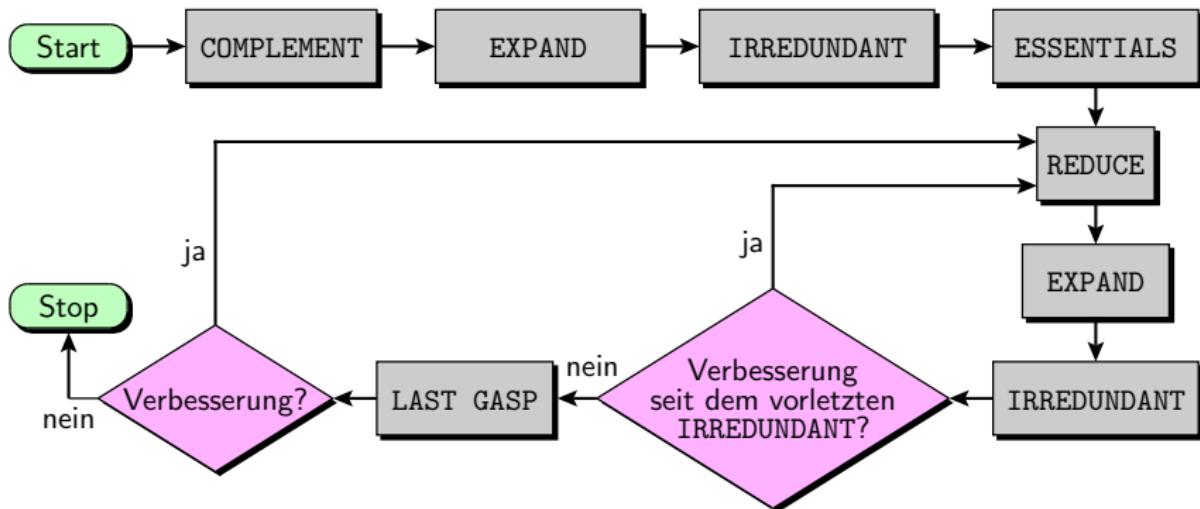
Don't care Kodierung wird auch dazu genutzt, die bei der Verschmelzung von Mintermen bzw. Implikanten wegfällenden Literale zu kennzeichnen, z. B.

$$x_1\bar{x}_2x_3 \vee x_1\bar{x}_2\bar{x}_3 = x_1\bar{x}_2 \wedge (x_3 \vee \bar{x}_3) = x_1\bar{x}_2 \cong 01\ 10\ 11$$

Minterme			\bar{x}_1x_1	\bar{x}_2x_2	\bar{x}_3x_3
\bar{x}_1	\bar{x}_2	\bar{x}_3	10	10	10
\bar{x}_1	\bar{x}_2	x_3	10	10	01
\bar{x}_1	x_2	\bar{x}_3	10	01	10
\bar{x}_1	x_2	x_3	10	01	01
x_1	\bar{x}_2	\bar{x}_3	01	10	10
x_1	\bar{x}_2	x_3	01	10	01
x_1	x_2	\bar{x}_3	01	01	10
x_1	x_2	x_3	01	01	01

Besonderheit: Markiert man die Stellen der PCN wie in der Tabelle links dargestellt ($\bar{x}_i x_i$ pro Variable), so kann die Mintermdarstellung unmittelbar aus PCM (ohne Kenntnis der Kodierung) abgelesen werden: Eine „1“ in einer Spalte zeigt an, dass die Variable im Minterm in der Form vorkommt, die dem Spaltenkopf entspricht.

Allgemeiner Ablauf



- ▶ Einzelne Schritte werden als Operatoren bezeichnet, es wird zwischen 6 verschiedenen Operatoren unterschieden
- ▶ EXPAND, IRREDUNDANT und REDUCE bilden den algorithmischen Kern des Verfahrens

COMPLEMENT-Schritt: Bildung der Nullmenge

- ▶ Im EXPAND-Schritt werden verschiedene Verschmelzungsmöglichkeiten einzelner Implikanten untersucht.
- ▶ In PCN lässt sich einfach feststellen, ob eine Verschmelzung möglich ist:
 1. Bitweise Summe (OR) der entsprechenden Zeilen bilden bzw. einfach die Kodierung der durch Verschmelzung wegfallenden Literale auf 11 setzen.
 2. Diese Summe mit jedem der Nullmenge zugehörigen Produktterm bitweise multiplizieren (AND). Sind alle Produkte ungültig (mindestens ein Literal ist mit 00 kodiert), so ist die Verschmelzung gültig. Stellt mindestens ein Produkt eine gültige PCN dar, so ist die Verschmelzung ungültig.
- Die zur Überprüfung der Legalität einer Verschmelzung benötigte Nullmenge in PCN wird im COMPLEMENT-Schritt gebildet.

EXPAND-Schritt: Konstruktion einer Überdeckung

- ▶ Im Gegensatz zu exakten Minimierungsverfahren werden nicht alle Primimplikanten gebildet
- ▶ Ausgehend von den Implikanten werden Primimplikanten durch „Expansion“ gebildet, dabei spielt die Reihenfolge eine Schlüsselrolle. Grundidee: Möglichst keine total redundanten Primimplikanten erzeugen.
 1. Implikanten nach vorgegebenen Kriterien sortieren.
 2. Primimplikanten zuerst mit den Implikanten bilden, die in der Sortierfolge am Anfang stehen.
 3. Für eine komplette Überdeckung sorgen.
- ➡ Eine Überdeckung entsteht durch direkte Konstruktion, nicht durch Suche eines globalen Minimums (wie es z. B. bei QMCV der Fall ist).

EXPAND-Schritt: Kriterium für das Sortieren der Implikanten

- ▶ Heuristik: Zuerst die Implikanten expandieren, die „am unwahrscheinlichsten“ von den anderen Implikanten überdeckt werden.
- ▶ Implikanten nach der Anzahl der gemeinsamen Werte (d. h. in gleicher Form vorkommender Variablen) sortieren.
- ▶ In PCN einfach implementierbar:
 1. Pro Spalte die Summe aller Einsen bilden
 2. Pro Zeile die Spaltensummen addieren, die einer „1“ in dieser Zeile entsprechen.
- ▶ Die entstehenden Zeilenbewertungen (Gewichte) dienen als Sortierkriterium, die Implikanten werden nach aufsteigenden Gewichten sortiert.

Ein Beispiel zur Bestimmung von Gewichten

Index	Minterme	\bar{x}_1x_1	\bar{x}_2x_2	\bar{x}_3x_3	\bar{x}_4x_4	Gewichte
14	$x_1x_2x_3\bar{x}_4$	0	1	0	1	10
15	$x_1x_2x_3x_4$	0	1	0	1	01
13	$x_1x_2\bar{x}_3x_4$	0	1	0	10	01
11	$x_1\bar{x}_2x_3x_4$	0	1	10	01	01
8	$x_1\bar{x}_2\bar{x}_3\bar{x}_4$	0	1	10	10	10
7	$\bar{x}_1x_2x_3x_4$	1	0	01	01	01
9	$x_1\bar{x}_2\bar{x}_3x_4$	0	1	10	10	01
Summe		1	6	3	4	25

→ Die Expansion soll mit Mintermen/Implikanten 7 und 8 zuerst durchgeführt werden.

Kriterien für die Implikanten-Auswahl

Bei der Expansion sind in der Regel mehrere Möglichkeiten vorhanden.

- Es wird diejenige Möglichkeit ausgewählt, bei der die meisten Implikanten zusammengefasst werden können.

Zielstellung: Nicht die größtmögliche Expansion, sondern die kleinste Anzahl von verbleibenden Implikanten (Reduzierung der Anzahl von UND-Gattern).

Anschließend wird durch weitere Expansion versucht, möglichst viele Überschneidungen mit den bereits gebildeten Implikanten zu erzeugen (Reduzierung der Anzahl von Literalen, d. h. von Eingängen der UND-Gatter). Mit Ausnutzung der *don't care* Werte werden die größtmöglichen Verschmelzungen gesucht.

- Ist auf das Problem der minimalen Überdeckung zurückführbar, jedoch wurde durch die Expansion der Suchraum signifikant reduziert.

Weitere Optimierung

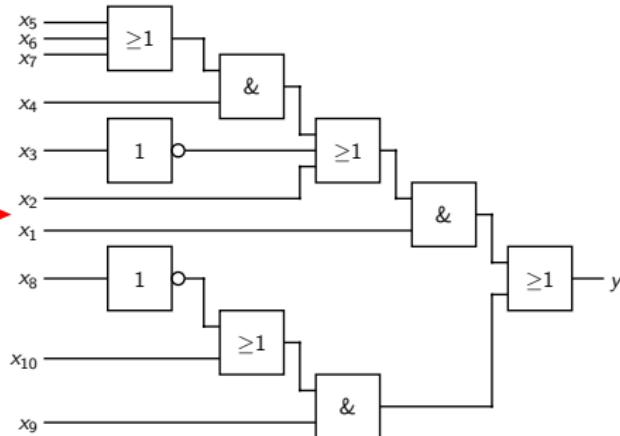
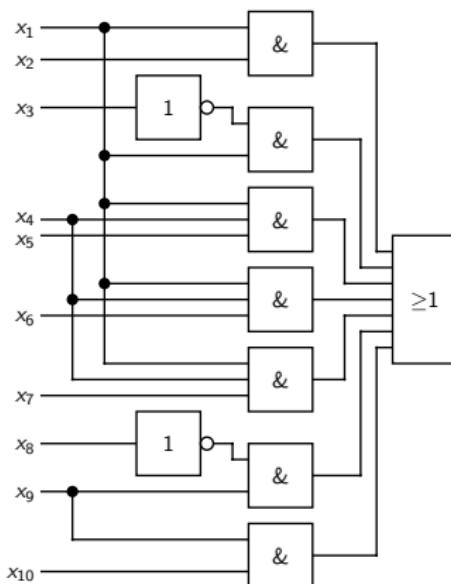
Prinzipiell liefert der **EXPAND**-Schritt bereits eine (oft gute) Lösung, diese wird jedoch noch mit weiteren Schritten optimiert:

- ▶ **IRREDUNDANT**-Operator entfernt alle total redundante Primimplikanten und einige partiell redundante Implikanten (heuristische Auswahl zu Gunsten eines partiell redundanten Implikanten, der die meisten von den anderen übernommenen Primimplikanten nicht überdeckten Minterme überdeckt). Durch Übernahmeentscheidungen entstehen evtl. neue total redundante Primimplikanten, die entfernt werden müssen.
- ▶ **REDUCE**-Operator revidiert einige durch **EXPAND** generierte Entscheidungen um evtl. weitere Verbesserungen zu erzielen. Dabei darf sich die Anzahl der Implikanten nicht vergrößern!
- ▶ **ESSENTIALS**-Operator markiert wesentliche Primimplikanten, um den Suchraum zu reduzieren.

Zusammenfassung und abschließende Betrachtungen

- ▶ Die Schritte **EXPAND**, **REDUCE** und **IRREDUNDANT** werden iterativ ausgeführt. Abbruchkriterium: Keine Reduzierung der Anzahl von Primimplikanten nach dem **IRREDUNDANT**-Schritt (heuristische Entscheidung).
- ▶ Nach dem Verlassen der Iterationsschleife wird mit dem **LAST_GASP**-Operator eine weiterer Optimierungsschritt durchgeführt, der einem Optimierungsdurchlauf entspricht, jedoch andere Heuristiken verwendet. Ist dabei eine Verbesserung möglich, so wird die „übliche“ Optimierungsschleife erneut durchlaufen.
- ➡ Minimierung der Wahrscheinlichkeit von Fehlentscheidungen durch Verwendung anderer Heuristiken.

Problemstellung



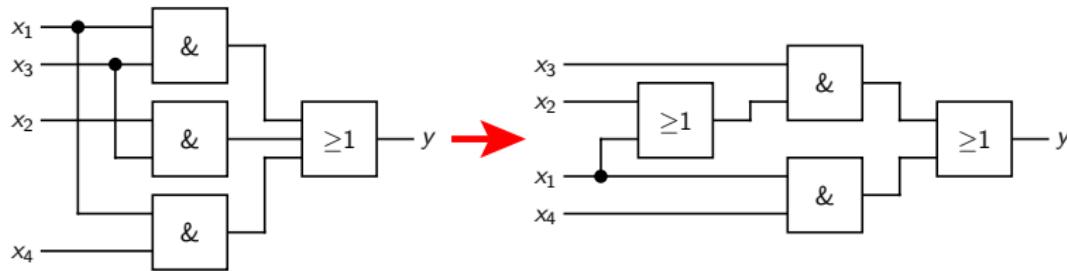
Zweistufige Darstellung ist eindeutig (z. B. kanonische Normalformen oder zweistufige Minimalformen), mehrstufige Darstellung ist vielfältig.

Faktorisierung

“Compared to two-level logic synthesis, the problem of optimum multilevel logic synthesis is an impossible dream.”

(Gary D. HACHTEL, Fabio SOPENZI, *Logic Synthesis and Verification Algorithms*, Springer 2006)

Überführung einer zweistufigen Darstellung in mehrstufige Form erfolgt mittels **Faktorisierung**:

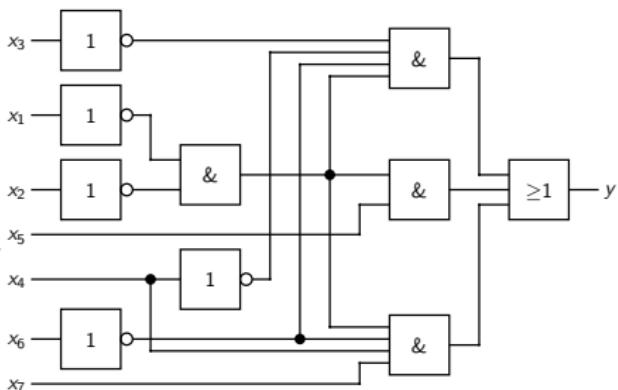
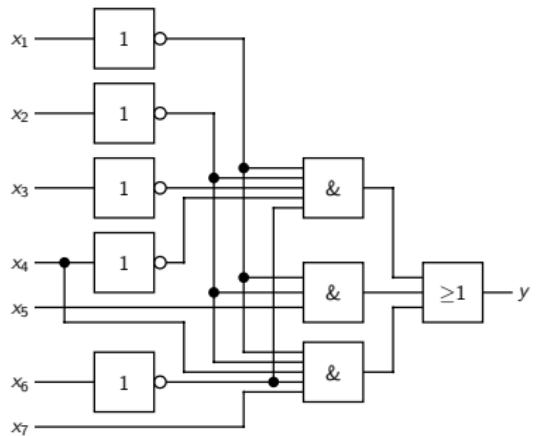


$$y = x_1x_3 \vee x_2x_3 \vee x_1x_4 \implies y = x_3 \wedge (x_1 \vee x_2) \vee x_1x_4$$

Weitere Möglichkeiten (nicht offensichtlich):

$$y = (x_1 \vee x_2x_3) \wedge (x_3 \vee x_1x_4), \quad y = (x_3 \vee x_1x_4) \wedge (x_1 \vee x_2)$$

Ein weiteres Beispiel zur Faktorisierung



$$\begin{aligned} y &= \overline{x}_1 \overline{x}_2 \overline{x}_3 \overline{x}_4 \overline{x}_6 \vee \overline{x}_1 \overline{x}_2 x_5 \vee \overline{x}_1 \overline{x}_2 x_4 \overline{x}_6 x_7 \\ &= ((\overline{x}_1 \wedge \overline{x}_2) \wedge \overline{x}_3 \overline{x}_4 \overline{x}_6) \vee ((\overline{x}_1 \wedge \overline{x}_2) \wedge x_5) \vee ((\overline{x}_1 \wedge \overline{x}_2) \wedge x_4 \overline{x}_6 x_7) \end{aligned}$$

Faktorisierung von Funktionsbündeln

Funktionsbündel $\{y_0, y_1, y_2\}$ in zweistufiger Darstellung:

$$y_0 = x_1 x_2 \vee x_1 \bar{x}_3 \vee x_2 x_4 \vee \bar{x}_3 x_4$$

$$y_1 = x_2 x_4 \vee \bar{x}_3 x_4$$

$$y_2 = \bar{x}_2 x_3 x_5$$

Faktorierte Darstellung:

$$y_0 = (x_1 \wedge (x_2 \vee \bar{x}_3)) \vee (x_4 \wedge (x_2 \vee \bar{x}_3)) = x_1 f_0 \vee x_4 f_0 = x_1 f_0 \vee y_1$$

$$y_1 = x_4 \wedge (x_2 \vee \bar{x}_3) = x_4 f_0$$

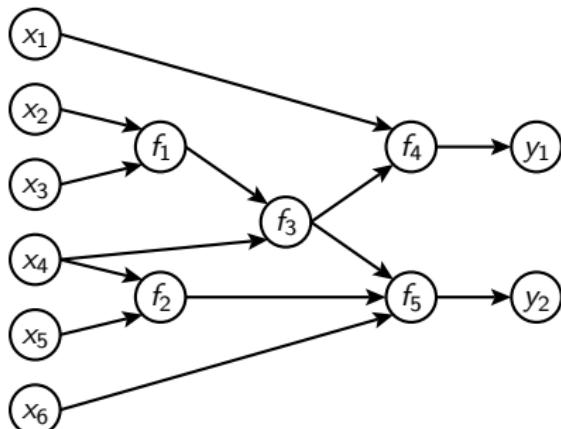
$$y_2 = x_5 \wedge (\bar{x}_2 \wedge x_3) = x_5 \wedge \overline{(x_2 \vee \bar{x}_3)} = x_5 \bar{f}_0$$

$$f_0 = x_2 \vee \bar{x}_3$$

mit f_0 als *intermediate factor variable*.

Darstellung von faktorisierten Schaltfunktionen

Azyklischer gerichteter Graph (BOOLEsches Netzwerk):



$$\begin{aligned}
 f_1 &= f_1(x_2, x_3) = \bar{x}_2 \vee \bar{x}_3 \\
 f_2 &= f_2(x_4, x_5) = \bar{x}_4 \vee \bar{x}_5 \\
 f_3 &= f_3(x_4, f_1) = \bar{x}_4 \wedge \bar{f}_1 \\
 f_4 &= f_4(x_1, f_3) = x_1 \vee \bar{f}_3 = y_1 \\
 f_5 &= f_5(x_6, f_2, f_3) \\
 &= (x_6 \wedge f_2) \vee (\bar{x}_6 \wedge \bar{f}_3) = y_2
 \end{aligned}$$

Struktur ist durch die Kantenmenge beschrieben, Funktion muss pro Knoten gespeichert werden (im Fall der Basisgatter können diese direkt eingezeichnet werden).

Vereinfachte Darstellung

Zur Vereinfachung der rechnerinternen Darstellung kann auf explizite Darstellung von Konjunktion und Disjunktion verzichtet werden:

- ▶ Ein Implikant ist eine Menge von Literalen
- ▶ Ein Teilfaktor oder eine Funktion ist eine Menge von Implikanten.

Beispielsweise

$$f = \underbrace{(x_1 \wedge x_3)}_{\text{Implikant } c_1} \vee \underbrace{(x_2 \wedge x_3)}_{\text{Implikant } c_2} \vee \underbrace{(x_1 \wedge x_4)}_{\text{Implikant } c_3}$$

$$f = \{c_1, c_2, c_3\}$$

$$c_1 = \{x_1, x_3\}, \quad c_2 = \{x_2, x_3\}, \quad c_3 = \{x_1, x_4\}$$

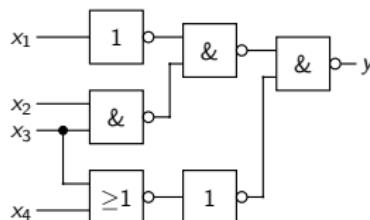
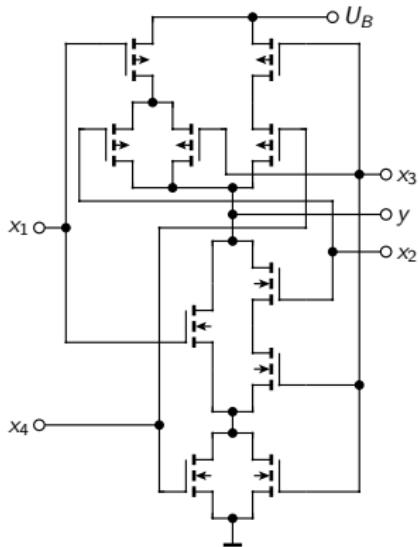
Weitere Darstellungskonventionen folgen.

Weitere Konventionen

Abhängig von den Anforderungen können zusätzliche Einschränkungen eingeführt werden:

- ▶ Jeder Knoten hat gleiche Funktion (z. B. NOR oder NAND)
 - ☺ Verringert Speicheranforderungen, da die Knotendarstellung einfacher wird
 - ☺ Viele Manipulationen können schneller ausgeführt werden, einfachere algorithmische Behandlung
 - ☺ Erhöht die Anzahl von Knoten (durch die Notwendigkeit der Überführung aller Gleichungen in das Grundoperatorenystem {NOR} bzw. {NAND})
 - ☹ Geschätzte Implementierungskosten können von den endgültigen Kosten stark abweichen
- ▶ Faktoren liegen immer in der SOP-Darstellung vor
 - ☺ Faktorweise Vereinfachung mit bekannten Optimierungstechniken möglich, Optimum ist in der Regel ermittelbar
 - ☹ Implementierungskostenschätzungen sind ungenau

Beispiel von Implementierungskosten einer Schaltfunktion



$$y = \overline{(x_1 \vee (x_2 \wedge x_3)) \wedge (x_3 \vee x_4)}$$

Bei CMOS-Handentwurf entspricht die Anzahl von Transistoren der doppelten Anzahl von Literalen. Bei einer Gate-Array-Implementierung ist dieses Verhältnis komplexer.

Faktorisierung vs. Schaltungsfläche

Verschiedene faktorierte Darstellungen der gleichen Funktion können große Unterschiede bei der Anzahl der Literale aufweisen:

$$\begin{aligned}
 y &= x_1x_2x_7 \vee x_1x_3x_7 \vee x_1x_4x_6 \vee x_1x_5x_6 \vee x_1x_6x_7 \vee x_2x_4 \\
 &\quad \vee x_3x_5 \vee x_2x_5 \vee x_3x_4 \\
 &= ((x_2 \vee x_3) \wedge (x_4 \vee x_5)) \\
 &\quad \vee (((x_4 \vee x_5 \vee x_7) \wedge x_6) \vee ((x_2 \vee x_3) \wedge x_7)) \wedge x_1) \text{ 12 Literale} \\
 &= ((x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee (x_1 \wedge x_7))) \\
 &\quad \vee ((x_4 \vee x_5 \vee x_7) \wedge x_1 \wedge x_6) \text{ 11 Literale} \\
 &= ((x_1 \wedge x_6) \vee x_2 \vee x_3) \wedge ((x_1 \wedge x_7) \vee x_4 \vee x_5) \text{ 8 Literale}
 \end{aligned}$$

Die ersten beiden Faktorisierungen heißen **algebraisch**, weil die SOP-Darstellung aus diesen unmittelbar durch das Auflösen von Klammern (entsprechend dem Distributivgesetz) gewonnen werden kann, die letzte Faktorisierung ist dagegen **boolesch**.

BOOLEsche Algebra und Divisionsoperation

In der BOOLEschen Algebra (und folglich in der Schaltalgebra) gibt es keine inverse Operation zu \bullet (bzw. \wedge), d. h. keine „Division“. Es gibt jedoch Quadrupel von schaltalgebraischen Ausdrücken (f, p, q, r) , für die

$$f = p \wedge q \vee r$$

gilt. In diesem Fall nennt man p den Divisor von f mit dem Divisionsergebnis (Quotienten) q und dem Divisionsrest r . Beispiel:

$$\begin{aligned} f &= (\bar{x} \vee z) \wedge (x \vee \bar{z}) \vee (\bar{x} \wedge y) \\ &= (\bar{x} \vee z) \wedge (y \vee \bar{z}) \vee (x \wedge z) \end{aligned}$$

→ Darstellung ist nicht eindeutig.

Division in der Schaltalgebra

Seien Q , R , F und P schaltalgebraische Ausdrücke in SOP-Darstellung. $(Q, R) := DIV(F, P)$ ist eine Division, wenn $F = (P \wedge Q) \vee R$ gilt.

Algebraische vs. BOOLEsche Division

Algebraische und BOOLEsche Division

Gilt $(Q, R) = DIV(F, P)$ und haben P und Q keine gemeinsamen Variablen, so heißt die Division algebraisch, ansonsten BOOLEsch.

Anmerkungen:

- ▶ Faktorisierungen, die nur auf algebraischer Division basieren, sind einfacher und schneller zu finden (da es im Allgemeinen weniger algebraische als BOOLEsche Faktorisierungen gibt)
- ▶ Einschränkung auf nur algebraische Faktorisierungen spart Rechenzeit, führt aber in der Regel zu einem schlechteren Ergebnis
- ➡ Folgende Betrachtungen gelten nur für algebraische Faktorisierung, da lediglich Grundprinzipien erläutert werden sollen.

Grundgedanke hinter den Faktorisierungsalgorithmen

- ▶ Möglichst viele gemeinsame Teiler finden
- ▶ Unter den gemeinsamen Teilern diejenigen aussuchen, die am meisten zur Vereinfachung der Funktionsdarstellung beitragen können.
- ▶ Der Rest sind Implementierungsfeinheiten:
 - ▶ algebraische oder BOOLEsche Faktorisierung verwenden
 - ▶ zusätzliche Kriterien zur Reduzierung des Suchraumes einführen
 - ▶ schnellere Division für triviale Operanden einführen (z. B. einzelne Literale oder Primimplikanten)
 - ▶ *don't cares* berücksichtigen
- ▶ Problematisch: Division ist nicht eindeutig!

Zusammenfassung

- ▶ Grundzüge heuristischer Logikoptimierung
- ▶ Übersicht zum ESPRESSO-Algorithmus zur Minimierung zweistufiger Logik
- ▶ Methoden zur Synthese und Optimierung mehrstufiger Logik

Rechnergestützer Entwurf digitaler Systeme: Vorlesungs-Gesamtübersicht

1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme:

Gliederung der 6. Vorlesung

2. Synthese und Technologie-Abbildung

2.1. Einordnung

2.2. HDL-Synthese

2.3. Logiksynthese und Optimierung

2.3.1. Zweistufige Logik

2.3.2. Bündeloptimierung

2.3.3. ESPRESSO-Algorithmus

2.3.4. Mehrstufige Logik

2.3.5. XOR-Logik

2.4. Technologie-Abbildung

Schwache Division

Schwache Division *WEAK_DIV* als Spezialfall der algebraischen Division von F durch P :

- ▶ $(Q, R) = DIV(F, P)$ ist eine algebraische Division
- ▶ R ist minimal (enthält kleinstmögliche Anzahl von Primimplikanten)
- ➔ Schwache Division ist (bei vorgegebenen F und P) eindeutig!

Beispiel:

$$F = x_1 x_4 \vee x_1 x_2 x_3 \vee x_2 x_3 x_4$$

$$P = x_1 \vee x_2 x_3$$

$$F = ((x_1 \vee (x_2 \wedge x_3)) \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_3)$$

$$\Rightarrow (Q, R) = WEAK_DIV(F, P) = (x_4, x_1 \wedge x_2 \wedge x_3)$$

Ein einfacher Algorithmus für schwache Division

```

WEAK_DIV( $F, P$ )
{
    foreach  $p_i \in P$ 
    {
         $V^{p_i} = \emptyset$ 
        foreach  $f_j \in F$ 
        {
            if ( $f_j$  contains all the literals of  $p_i$ ) {
                 $v_{ij} = f_j$  with the literals in  $p_i$  deleted
                 $V^{p_i} = V^{p_i} \cup v_{ij}$  }
        }
         $Q = \bigcap_i V^{p_i}$ 
         $R = F - PQ$ 
        return ( $Q, R$ )
    }
}

```

Anwendung des *WEAK_DIV* Algorithmus auf bereits diskutiertes Beispiel

$$\begin{aligned}
 F &= x_1x_4 \vee x_1x_2x_3 \vee x_2x_3x_4, P = x_1 \vee x_2x_3 \\
 p_1 &= \{x_1\}, p_2 = \{x_2, x_3\}, f_1 = \{x_1, x_4\}, f_2 = \{x_1, x_2, x_3\}, \\
 f_3 &= \{x_2, x_3, x_4\} \\
 v_{11} &= \{x_4\}, v_{12} = \{x_2, x_3\}, v_{22} = \{x_1\}, v_{23} = \{x_4\} \\
 V^{p_1} &= \{\{x_4\}, \{x_2, x_3\}\}, V^{p_2} = \{\{x_1\}, \{x_4\}\} \\
 Q &= \{\{x_4\}\} \\
 F &= \{\{x_1, x_4\}, \{x_1, x_2, x_3\}, \{x_2, x_3, x_4\}\}, P = \{\{x_1\}, \{x_2, x_3\}\} \\
 R &= F - PQ = \{\{x_1, x_2, x_3\}\} \\
 \Rightarrow & WEAK_DIV(F, P) = (x_4, x_1x_2x_3), \\
 \text{bzw. } & F = ((x_1 \vee (x_2 \wedge x_3)) \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_3) \\
 \Rightarrow & \text{wie bereits angegeben}
 \end{aligned}$$

Ein etwas komplexeres Beispiel

$$\begin{aligned}
 F &= x_1x_4 \vee x_1x_5x_6 \vee x_1x_2 \vee \bar{x}_2x_3x_4 \vee \bar{x}_2x_3x_5x_6, P = x_1 \vee \bar{x}_2x_3 \\
 p_1 &= \{x_1\}, p_2 = \{\bar{x}_2, x_3\}, f_1 = \{x_1, x_4\}, f_2 = \{x_1, x_5, x_6\}, \\
 f_3 &= \{x_1, x_2\}, f_4 = \{\bar{x}_2, x_3, x_4\}, f_5 = \{\bar{x}_2, x_3, x_5, x_6\}, \\
 v_{11} &= \{x_4\}, v_{12} = \{x_5, x_6\}, v_{13} = \{x_2\}, \\
 v_{24} &= \{x_4\}, v_{25} = \{x_5, x_6\} \\
 V^{p_1} &= \{\{x_4\}, \{x_5, x_6\}, \{x_2\}\}, V^{p_2} = \{\{x_4\}, \{x_5, x_6\}\} \\
 Q &= \{\{x_4\}, \{x_5, x_6\}\} \\
 F &= \{\{x_1, x_4\}, \{x_1, x_5, x_6\}, \{x_1, x_2\}, \{\bar{x}_2, x_3, x_4\}, \{\bar{x}_2, x_3, x_5, x_6\}\} \\
 P &= \{\{x_1\}, \{\bar{x}_2, x_3\}\} \\
 R &= F - PQ = \{\{x_1, x_2\}\} \\
 \Rightarrow & WEAK_DIV(F, P) = (x_4 \vee x_5x_6, x_1x_2), \\
 \text{bzw. } & F = ((x_1 \vee (\bar{x}_2 \wedge x_3)) \wedge (x_4 \vee (x_5 \wedge x_6))) \vee (x_1 \wedge x_2)
 \end{aligned}$$

Ein „naiver“ rekursiver Faktorisierungsalgorithmus

FACTOR(F,DIVISOR,DIVIDE)

{

 if (*F* has no factor) return (*F*)

D = *DIVISOR(F)*

 (*Q*,*R*) = *DIVIDE(F,D)*

 return (*FACTOR(Q)FACTOR(D) + FACTOR(R)*)

}

Für algebraische Faktorisierung kann *DIVIDE* durch *WEAK_DIV* ersetzt werden.

- Verfeinerungen sind in den Details der Implementierung von *DIVIDE* und *DIVISOR* Operationen verborgen, z. B. Erkennung von *R* und *Q* Paaren, die gemeinsame Literale haben und weiter faktorisiert werden können.

Beispiel einer algebraischen Faktorisierung

$$F = x_1 x_2 x_3 \vee x_1 x_2 x_4 \vee x_1 x_5 \vee x_1 x_6 \vee x_7$$

$$D = x_3 \vee x_4 \quad (\text{DIVISOR}(F))$$

$$Q = x_1 x_2$$

$$R = x_1 x_5 \vee x_1 x_6 \vee x_7$$

$$F = (x_1 \wedge x_2) \wedge (x_3 \vee x_4) \vee ((x_1 \wedge x_5) \vee (x_1 \wedge x_6)) \vee x_7$$

$$R = x_1 \wedge (x_5 \vee x_6) \vee x_7 \quad (\text{FACTOR}(R))$$

$$F = (x_1 \wedge x_2) \wedge (x_3 \vee x_4) \vee (x_1 \wedge (x_5 \vee x_6)) \vee x_7$$

Eine weitere Faktorisierung ist möglich (wird aber vom „naiven“ Algorithmus nicht erkannt):

$$F = x_1 \wedge (x_2 \wedge (x_3 \vee x_4) \vee (x_5 \vee x_6)) \vee x_7$$

Weitere Syntheseschritte

Eine endgültige Darstellung eines optimierten mehrstufigen Netzwerks (ggf. unter Berücksichtigung mehrerer Ausgangsfunktionen) entsteht durch mehrfaches Ausführen verschiedener Schritte:

Dekomposition (*Zerlegung, decomposition*): Darstellung der Schaltfunktion $f(X)$ in der Form $f(X) = f'(g(X), X)$ mit $X = (x_1, x_2, \dots, x_n)$ (meistens mit Hilfe von Faktorisierungsalgorithmen oder deren Abwandlungen)

Extraktion (*extraction*): Finden von gleichen Teilausdrücken in mehreren Funktionen

Substitution (*Ersetzung, substitution*): Umformung der Schaltfunktion unter Ausnutzung anderer Schaltfunktionen z. B. $f = (a \vee bc) \wedge (d \vee e) = g \wedge (d \vee e)$ mit $g = a \vee bc$.

Elimination (*collapsing*): Umkehrung der Substitution.

Beispiel zur mehrstufigen Logiksynthese

$$F_1 = abc \vee abd \vee ae \vee af \vee g$$

$$F_2 = ace \vee ade \vee bce \vee bde \vee cf \vee df$$

Dekomposition und Substitution:

$$F_1 = a \wedge (b \wedge (c \vee d) \vee e \vee f) \vee g = a \wedge Y_1 \vee g \text{ mit}$$

$$Y_1 = b \wedge X_1 \vee e \vee f$$

$$X_1 = c \vee d$$

$$F_2 = (c \vee d) \wedge (e \wedge (a \vee b) \vee f) = X_2 \wedge Z_2 \text{ mit}$$

$$Z_2 = e \wedge Y_2 \vee f$$

$$Y_2 = a \vee b$$

$$X_2 = c \vee d$$

Extraktion: $X_1 = X_2 = c \vee d$

Beispiel zur mehrstufigen Logiksynthese (fortgesetzt)

Komplette Faktorisierung:

$$F_1 = a \wedge (b \wedge (c \vee d) \vee e \vee f) \vee g = a \wedge Y_1 \vee g \text{ mit}$$

$$Y_1 = b \wedge X_1 \vee e \vee f$$

$$X_1 = c \vee d$$

$$F_2 = (c \vee d) \wedge (e \wedge (a \vee b) \vee f) = X_1 \wedge Z_2 \text{ mit}$$

$$Z_2 = e \wedge (a \vee b) \vee f \text{ (Elimination)}$$

Im Vergleich:

	zweistufig	mehrstufig
2-fach OR	—	4
3-fach OR	—	1
2-fach AND	4	4
3-fach AND	6	—
5-fach OR	1	—
6-fach OR	1	—
Schaltungstiefe	2	5

Ein weiteres Logiksynthese-Beispiel

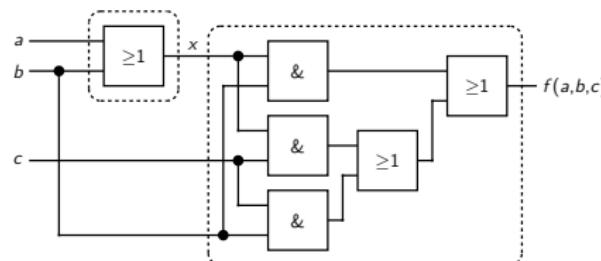
Faktorisierung und Substitution:

$$f(a,b,c) = (\underbrace{(a \vee b) \wedge b}_x) \vee (b \wedge c) \vee (\underbrace{(a \vee b) \wedge c}_x) = (x \wedge b) \vee (b \wedge c) \vee (x \wedge c)$$

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

f	$\bar{b}\bar{c}$	$\bar{b}c$	bc	$b\bar{c}$
x	0	1 <i>p</i>	1	<i>m1</i>
\bar{x}	0	0	1 <i>n</i>	0

$$f = \underbrace{(x \wedge b)}_m \vee \underbrace{(b \wedge c)}_n \vee \underbrace{(x \wedge c)}_p$$



Funktionale Abhangigkeiten durch Substitution

Kann der Ausdruck $f(a,b,c) = (x \wedge b) \vee (b \wedge c) \vee (x \wedge c)$ minimiert werden?

Nein, wenn x als unabhangige Variable behandelt wird, da fur sie jede mogliche Belegung angenommen werden muss

Ja, wenn die funktionalen Abhangigkeiten zwischen x , a und b berucksichtigt werden: Die Belegungen $(a,b,x) = (0,0,1)$, $(a,b,x) = (0,1,0)$, $(a,b,x) = (1,0,0)$ sowie $(a,b,x) = (1,1,0)$ konnen niemals auftreten, da $x = 0$ bei $b = 1$ ausgeschlossen ist.

- Die Belegungen $(x,b,c) = (0,1,0)$ und $(x,b,c) = (0,1,1)$ sind nicht moglich, der entsprechende Funktionswert kann als *don't care* angenommen werden.

Satisfability don't cares (SDC)

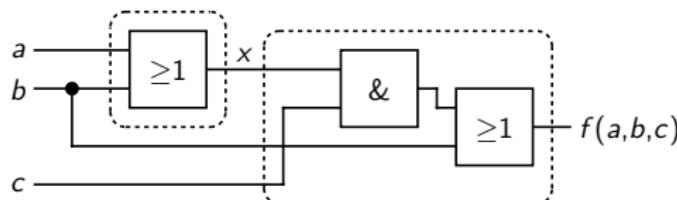
sind Eingangsbelegungen einer Schaltfunktion, die aufgrund von funktionalen Abhangigkeiten zwischen den Eingangsvariablen niemals auftreten konnen und fur die der Funktionswert somit frei wahlbar ist.

Logikminimierung mit SDC

Aus dem letzten Beispiel:

f	$\bar{b}\bar{c}$	$\bar{b}c$	$b\bar{c}$	$b\bar{c}$
x	0	1 $\textcolor{blue}{n}$	1	$\textcolor{red}{m}1$
\bar{x}	0	0	$d\bar{c}$	$d\bar{c}$

$$f = \underbrace{b}_{m} \vee \underbrace{(x \wedge c)}_{n}$$



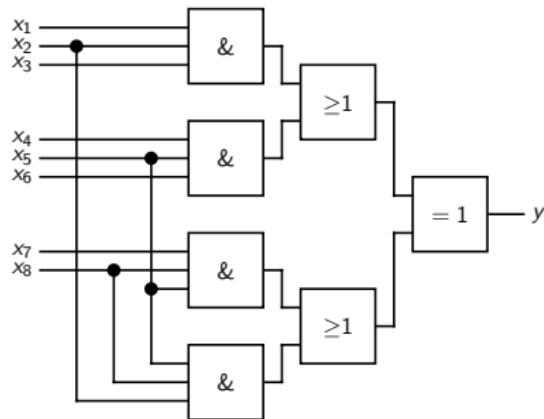
Formale Bestimmungsvorschrift für SDC einer internen Teilfunktion $y_i = f_i(x,y)$ mit x als Menge aller primären Eingänge und y als Menge aller internen Teilfunktionen lautet $SDC(i) = y_i \oplus f_i$, z. B.

$$\begin{aligned} SDC(x) &= \underbrace{x}_{y_i} \oplus \underbrace{(a \vee b)}_{f_i} = (\bar{x} \wedge (a \vee b)) \vee (x \wedge (\overline{a \vee b})) \\ &= (\bar{x} \wedge a) \vee (\bar{x} \wedge b) \vee (x \wedge \bar{a} \wedge \bar{b}). \end{aligned}$$

Alle Belegungen (a,b,x) , bei denen der Ausdruck in der letzten Zeile erfüllt ($=1$) ist, können niemals auftreten (siehe letzte Seite).

XOR-Darstellung von Schaltnetzen

Spezialfall der mehrstufigen Darstellung von Schaltnetzen: XOR-Logik



Darstellung basiert auf XOR-Verknüpfung von Implikanten, auch bekannt als REED-MULLER-Form, wobei keine Variable in negierter Form auftreten darf, z. B.

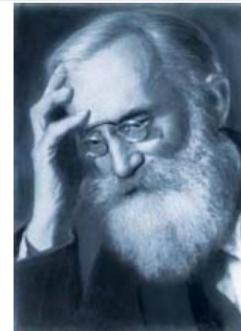
$$y = (x_1 x_2 \vee x_3 x_4) \oplus (x_4 \vee x_5 x_6)$$

In generalisierter REED-MULLER-Form (GRM) dürfen Implikanten sowohl invertierte als auch nicht invertierte Literale enthalten.

XOR-Zerlegung als Normalform

REED-MULLER-Normalform (auch Ringsummennormalform, RSNF, *algebraic normal form*, REED-MULLER-Entwicklung):

- ▶ basiert auf der Tatsache, dass $\{\oplus, \wedge, 1\}$ ein vollständiges Operatorenystem bilden
- ▶ beim Rechnen im \mathbb{Z}_2 entspricht \oplus Addition und \wedge Multiplikation, womit schaltalgebraische Operationen auf arithmetische Operationen zurückgeführt werden können
- ▶ zu jeder Schaltfunktion existiert eine eindeutige Darstellung im Form eines Polynoms mit Koeffizienten aus \mathbb{Z}_2 (gezeigt 1927 von Iwan SCHEGALKIN, daher auch als SCHEGALKIN-Polynom bekannt) ➔ Normalform
- ▶ die beiden anderen Namensgeber sind die amerikanischen Mathematiker/Informatiker Irving Stoy REED (1923–2012) und David Eugene MULLER (1924–2008)



Iwan Iwanowitsch
SCHEGALKIN
(1869–1947, russischer
Mathematiker)

Tabellarische Darstellung der REED-MULLER-Normalform

Zu jeder (homogenen zweiwertigen) Schaltfunktion existiert eine kanonische (positive) REED-MULLER-Normalform, die Basisterme der Form „1“ oder $B_k = \bigwedge_{i=1}^{n \geq 1} x_i$ mit XOR verknüpft, z. B.

x_3	x_2	x_1	y	B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7
0	0	0	1	x							
0	0	1	1	x	x						
0	1	0	0	x		x					
0	1	1	1	x	x	x	x				
1	0	0	0	x				x			
1	0	1	0	x	x			x	x		
1	1	0	0	x		x		x		x	
1	1	1	1	x	x	x	x	x	x	x	x

Aufstellen der REED-MULLER-Normalform

- ▶ Zeile 1: $y = 1$ ist nur in der Spalte B_0 erfüllt $\Rightarrow B_0$ hinzunehmen
- ▶ Zeile 2: $y = 1$ ist bereits durch B_0 gegeben, Hinzunahme von B_1 liefert den Funktionswert 0 ($1 \oplus 1$) $\Rightarrow B_1$ nicht hinzunehmen
- ▶ Zeile 3: $y = 0$, der Term B_0 liefert jedoch bisher den Funktionswert „1“, Hinzunahme von B_2 korrigiert das Problem ($1 \oplus 1 = 0$) $\Rightarrow B_2$ hinzunehmen
- ▶ Zeile 4: $y = 1$, $B_0 \oplus B_2 = 0$, somit muss $B_3 = x_1 \wedge x_2$ hinzugenommen werden $\Rightarrow B_3$ hinzunehmen
- ▶ ...

$$y = B_0 \oplus B_2 \oplus B_3 \oplus B_4 \oplus B_6 = 1 \oplus x_2 \oplus x_1 x_2 \oplus x_3 \oplus x_2 x_3$$



Irving Stoy REED
(1923–2012, amerikanischer Mathematiker)

Vor- und Nachteile der REED-MULLER-Form

Vorteile:

- ▶ Kompaktere Darstellung (im Vergleich zur SOP-Form) für viele Schaltfunktionen
- ▶ keine Inverter
- ▶ konstante Signallaufzeiten (gleiche Schaltungstiefe unabhängig von der Wahl des Eingangs)
- ▶ Invertierung einer Funktion ist sehr einfach
- ▶ Effiziente Faktorisierung möglich



David Eugene MULLER
(1924–2008, amerikanischer Mathematiker)

Nachteile:

- ▶ Benötigt XOR-Verknüpfungen
- ▶ Wird für große Anzahl von Eingangsvariablen ebenfalls komplex (wie SOP)

Beispiel zur Negation

Basierend auf den DEMORGANSchen Regeln:

$$y = (x_1 \vee x_2 x_3 \vee x_3 x_4) \oplus (x_5 \vee x_6)$$

$$\bar{y} = (x_1 \vee x_2 x_3 \vee x_3 x_4) \oplus (\bar{x}_5 \wedge \bar{x}_6)$$

$$\bar{y} = (\bar{x}_1 \bar{x}_2 \bar{x}_4 \vee \bar{x}_1 \bar{x}_3) \oplus (x_5 \vee x_6)$$

Beide invertierte Darstellungen haben gleiche Anzahl von Literalen, d. h. vergleichbar komplexe schaltungstechnische Umsetzung. Ein weiteres Umformungsbeispiel (doppelte Negation, z. B. UND vs. ODER ausbalancieren):

$$y = (x_1 \vee x_2 x_3 \vee x_3 x_4) \oplus (x_5 \vee x_6)$$

$$\bar{y} = (x_1 \vee x_2 x_3 \vee x_3 x_4) \oplus (\bar{x}_5 \wedge \bar{x}_6)$$

$$\bar{\bar{y}} = (\bar{x}_1 \bar{x}_2 \bar{x}_4 \vee \bar{x}_1 \bar{x}_3) \oplus (\bar{x}_5 \wedge \bar{x}_6) = y$$

XOR-Faktorisierung

XOR-Faktorisierung basiert auf der Tatsache, dass eine doppelte XOR-Verknüpfung mit einem und demselben Faktor sich aufhebt:

$$y = x_1 \wedge x_2$$

$$y = x_2 \oplus (x_2 \oplus (x_1 \wedge x_2)) = x_1 \wedge x_2$$

Faktorisierungsbeispiel:

$$\begin{aligned} y &= z \vee x_1 x_2 x_4 x_5 x_6 x_7 x_8 \vee x_3 x_4 x_5 x_6 x_7 x_8 \\ &\quad \vee \bar{x}_1 \bar{x}_8 \vee \bar{x}_2 \bar{x}_8 \vee \bar{x}_3 \bar{x}_8 \vee \bar{x}_4 \bar{x}_8 \vee \bar{x}_5 \bar{x}_8 \vee \bar{x}_6 \bar{x}_8 \vee \bar{x}_7 \bar{x}_8 \\ &= \bar{z} x_8 \oplus (\bar{z} x_8 \oplus (z \vee x_1 x_2 x_4 x_5 x_6 x_7 x_8 \vee x_3 x_4 x_5 x_6 x_7 x_8 \\ &\quad \vee \bar{x}_1 \bar{x}_8 \vee \bar{x}_2 \bar{x}_8 \vee \bar{x}_3 \bar{x}_8 \vee \bar{x}_4 \bar{x}_8 \vee \bar{x}_5 \bar{x}_8 \vee \bar{x}_6 \bar{x}_8 \vee \bar{x}_7 \bar{x}_8)) \\ &= \bar{z} x_8 \oplus (\bar{z} x_1 x_2 x_4 x_5 x_6 x_7 x_8 \vee \bar{z} x_3 x_4 x_5 x_6 x_7 x_8 \\ &\quad \vee \bar{z} x_1 x_2 x_3 x_4 x_5 x_6 x_7) \end{aligned}$$

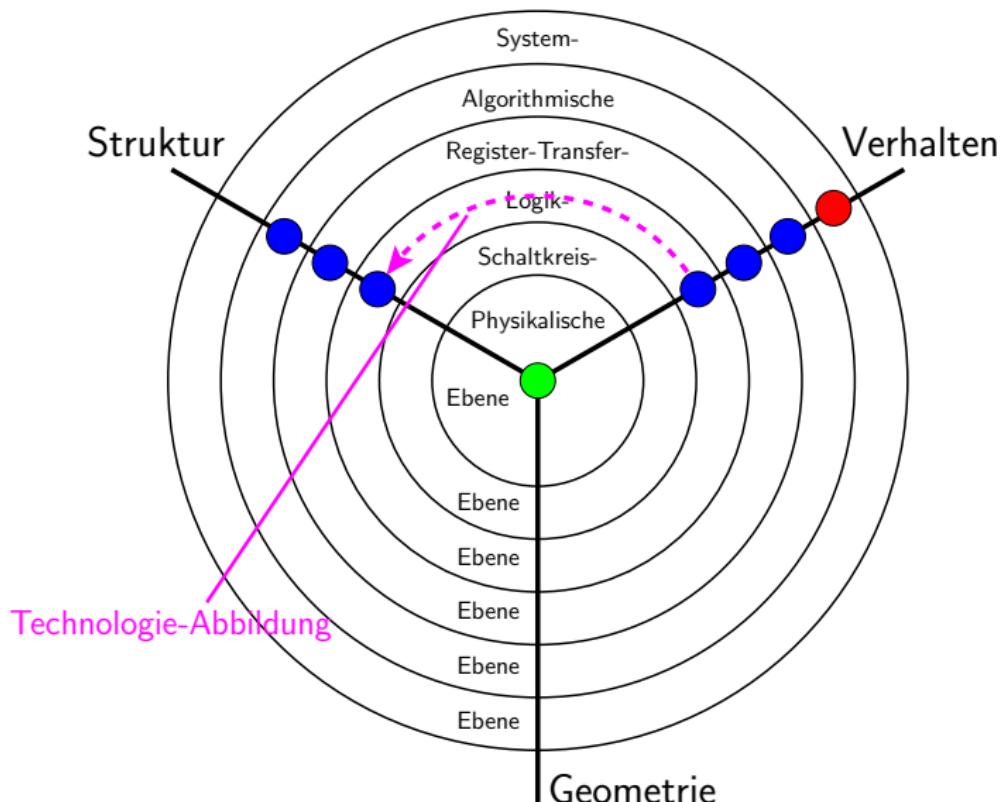
Ziel der Technologie-Abbildung

- ▶ Eine technologie-unabhängige Netzliste (Basisgatter, einfache Speicherelemente usw.) in eine funktional-äquivalente Darstellung überführen, die ausschließlich vorgegebene Komponenten beinhaltet
- ▶ Zusatzkriterien:
 - ▶ möglichst kostengünstig (weniger Komponenten, „preiswerte“ Komponenten)
 - ▶ möglichst kurze Verzögerungszeit

Keine Aufgaben der Technologie-Abbildung:

- ▶ Globale Logikoptimierung
- ▶ Globale Optimierung der Verzögerungszeit

Zur Erinnerung: Technologie-Abbildung im Y-Diagramm



Allgemeine vs. LUT-basierte Abbildung

Verfügbare Komponenten (Gatter) sind in den Technologie-Bibliotheken zusammengefasst. Unterschied zur LUT-basierten Technologie-Abbildung: Gatter sind nicht rekonfigurierbar.

- ➡ Es reicht nicht, eine Überdeckung der Netzliste mit k -beschränkten Kegeln zu finden, es wird eine exakte Übereinstimmung mit verfügbaren Elementen der Bibliothek gefordert

Eigenschaften eines guten Algorithmus:

- ▶ Bibliothek-unabhängig, d. h. Bibliotheken können als Parameter eingelesen werden
- ▶ Keine expliziten Annahmen über Verfügbarkeit von bestimmten Komponenten/Gattern
- ▶ Gutes Laufzeitverhalten

Verfeinerung der Modellierung

Technologie-Abbildung ist ein Synthese-Schritt, der Entwurf rückt näher an die physikalische Umsetzung heran

- Bisherige Modelle und Kostenkriterien reichen nicht mehr aus und müssen verfeinert werden

Bisher: Optimierungsziel ist Reduzierung der Anzahl von logischen Verknüpfungen und Literalen (die untereinander „gleichberechtigt“ sind)

- Weniger Gatter mit weniger Eingängen

Ab jetzt: Detaillierte Modelle, die Abhängigkeiten von der Ausfächerung (*fan-out*), Anzahl der Eingänge, physikalischen Anordnung usw. berücksichtigen

- Ein 3-fach UND-Gatter mit $fan-out=2$ hat eine andere Verzögerungszeit als ein 2-fach UND-Gatter mit $fan-out=1$

Zwei Klassen von Algorithmen

Regelbasierte: Bekannt aus dem Bereich der künstlichen Intelligenz,
hier nicht weiter betrachtet

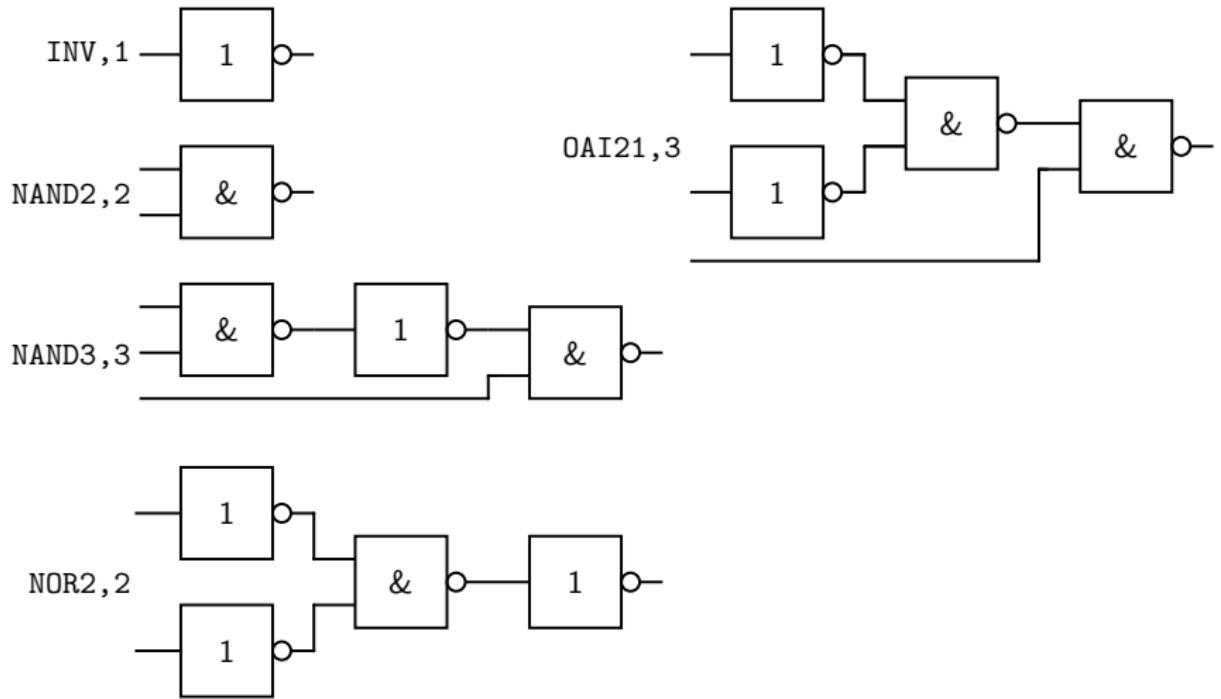
Graphenüberdeckung: Darstellung der Netzliste in Form eines
azyklischen gerichteten Graphen (*directed acyclic graph, DAG*)
und Suche nach einer (minimalen) Überdeckung des Graphen
mit Teilgraphen, die Elemente der Technologie-Bibliothek
repräsentieren. Ursprünglich wurden viele Techniken aus dem
Compilerbau übernommen, wo ähnliche Probleme bei der
Codegenerierung auftreten.

Wichtig

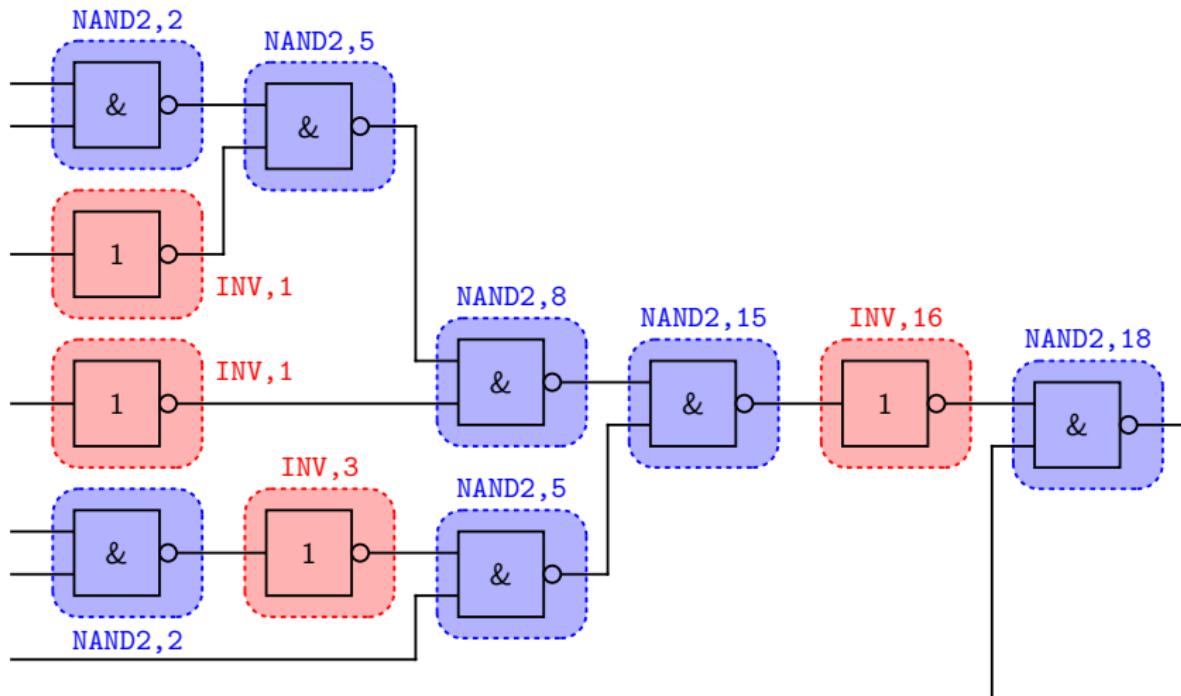
Elemente der Technologie-Bibliothek müssen ein vollständiges
Operatoren-System bilden!

Vereinbarungen: Netzliste liegt in kanonischer Form vor (alle Knoten
sind Elemente eines vollständigen Operatoren-Systems), der
entsprechende DAG wird als *subject DAG* bezeichnet.

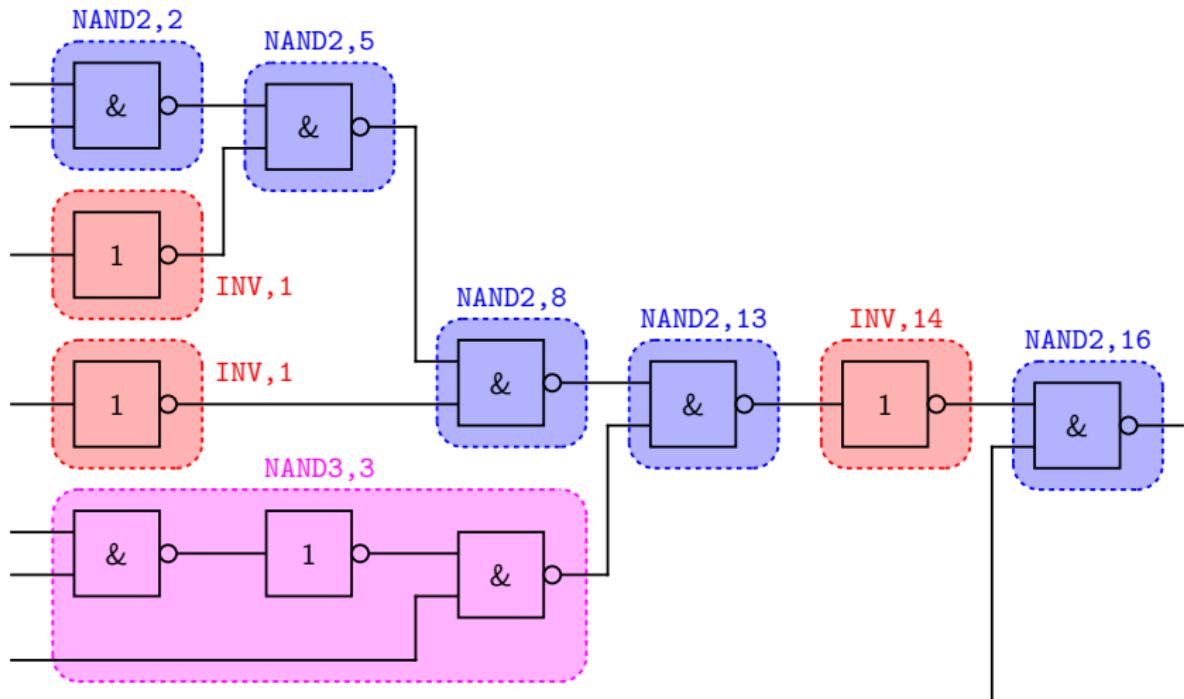
Beispiel einer Technologie-Bibliothek



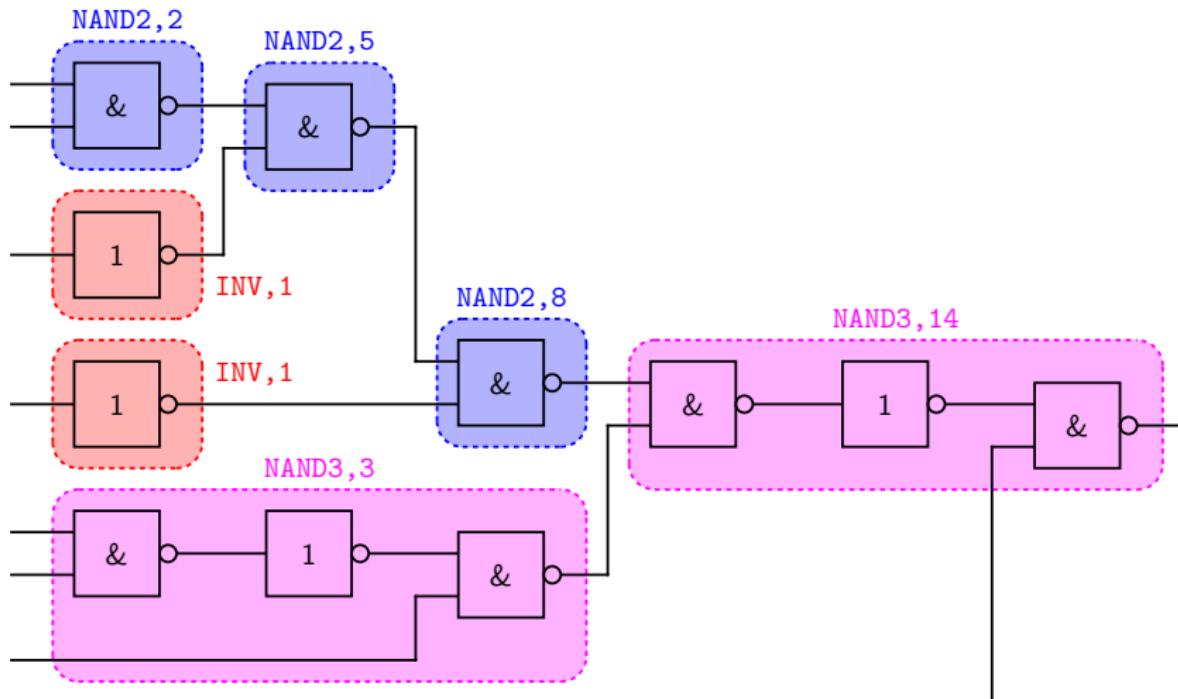
Beispiel einer Überdeckung einer Netzliste mit der gegebenen Technologie-Bibliothek



Eine bessere Überdeckung mit gleicher Bibliothek



Eine noch bessere Überdeckung mit gleicher Bibliothek



Zusammenfassung

- ▶ Synthese und Faktorisierung mit XOR-Logik
- ▶ Allgemeine Problemstellung der Technologie-Abbildung
- ▶ Technologie-Abbildung mittels Graph-Überdeckung

Rechnergestützer Entwurf digitaler Systeme: Vorlesungs-Gesamtübersicht

1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme: Gliederung der 7. Vorlesung

2. Synthese und Technologie-Abbildung

- 2.1. Einordnung
- 2.2. HDL-Synthese
- 2.3. Logiksynthese und Optimierung
- 2.4. Technologie-Abbildung

3. Partitionierung und Floorplanning

3.1. Partitionierung

- 3.1.1. Grundlagen
- 3.1.2. Clustering
- 3.1.3. RAJARAMAN-WONG-Algorithmus
- 3.1.4. KERNIGHAN-LIN-Algorithmus

3.2. Floorplanning

- 3.2.1. Definitionen und Datenstrukturen
- 3.2.2. Floorplan-Sizing-Algorithmus
- 3.2.3. Lineare Optimierung
- 3.2.4. Pinzuordnung

Ein altbekanntes Problem

- ▶ Lösungsraum ist sehr groß
- ▶ Problem des Findens einer optimalen Lösung ist NP-äquivalent (selbst bei einer Bibliothek mit nur 3 Komponenten und maximal 2 Ein- und Ausgängen pro Knoten)

Allerdings:

- ▶ Eingangs-DAG ist seinerseits ein Produkt von heuristischen Verfahren und daher meistens suboptimal
- ▶ Suche einer optimalen Überdeckung um jeden Preis ist nicht zwangsläufig sinnvoll

Hilfestellung: Zerlegung des DAG in Teilbäume und Suche einer Überdeckung in diesen mittels dynamischer Programmierung

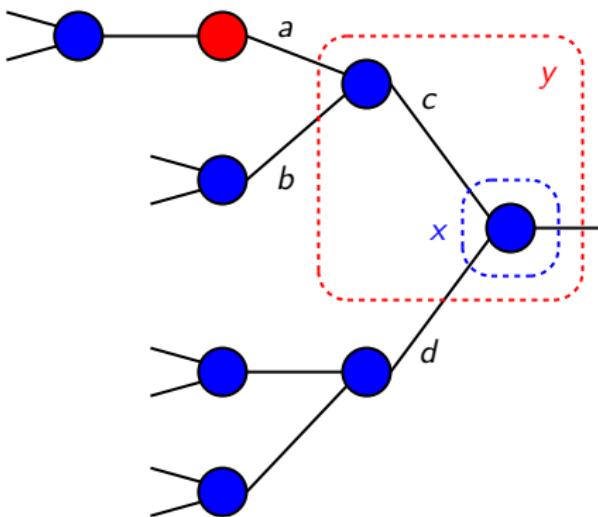
Baumüberdeckung

- ▶ Für Bäume existieren Algorithmen, die eine optimale Überdeckung in linearer Zeit finden
- ▶ Beweisbar: Optimale Überdeckung für einen Baum besteht aus einer Überdeckung seines Wurzelknotens kombiniert mit optimaler Überdeckung der durch Entfernen der Wurzel entstehenden Teilbäume
- ▶ Rekursives Verfahren, das von der Wurzel zu den Blättern eine Überdeckung generiert

Problem: Optimalität geht verloren, wenn Teilbäume wieder zu einem DAG zusammengefügt werden

- ▶ Wird zugunsten der Laufzeit in Kauf genommen, lokale Optimierungen sorgen für teilweise Verbesserung der Lösung

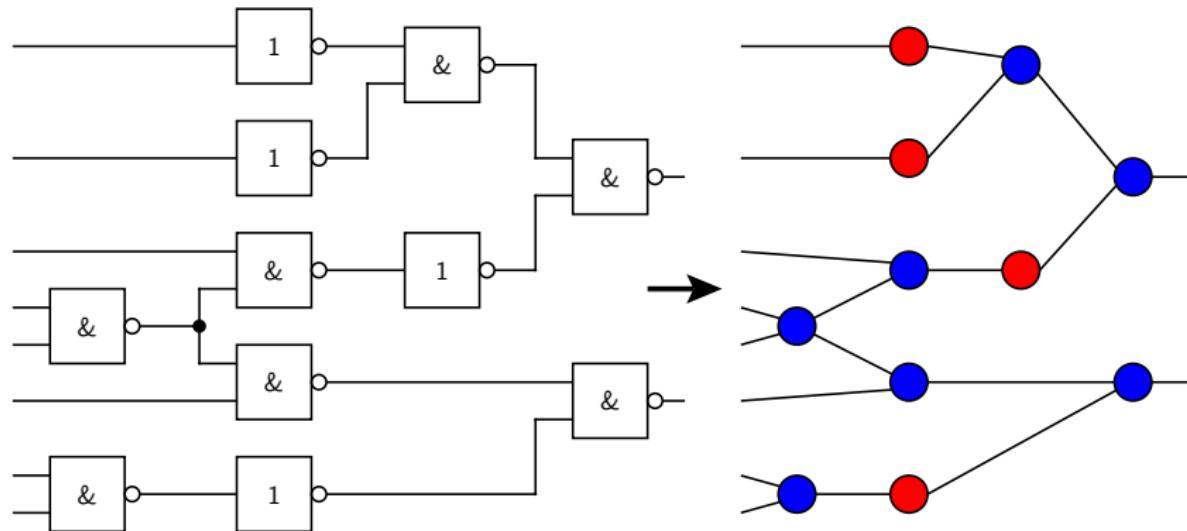
Ermitteln einer optimalen Überdeckung



Optimale Überdeckung für x nutzt die optimalen Überdeckungen für Teilbäume c und d , optimale Überdeckung für y nutzt die optimalen Überdeckungen für a , b und d .

1. Schritt

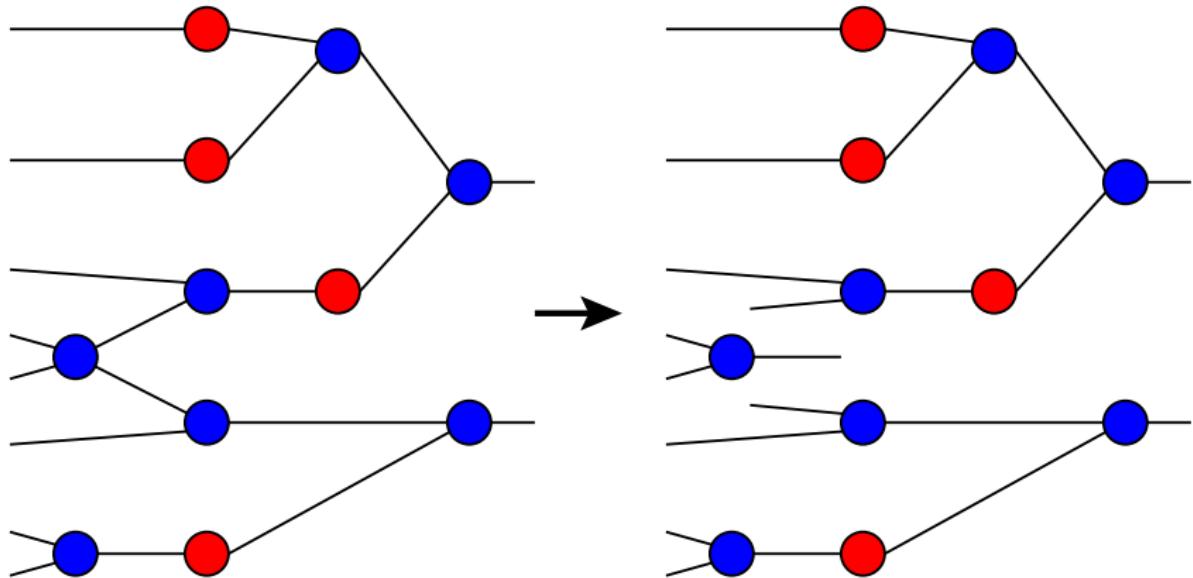
Von der Netzliste zum *subject DAG*:



Eventuell muss die Netzliste vorher in kanonische Form überführt werden (hier liegt diese bereits vor).

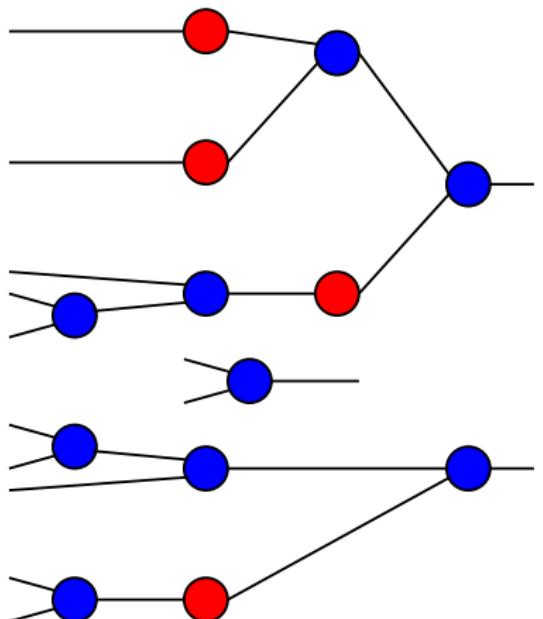
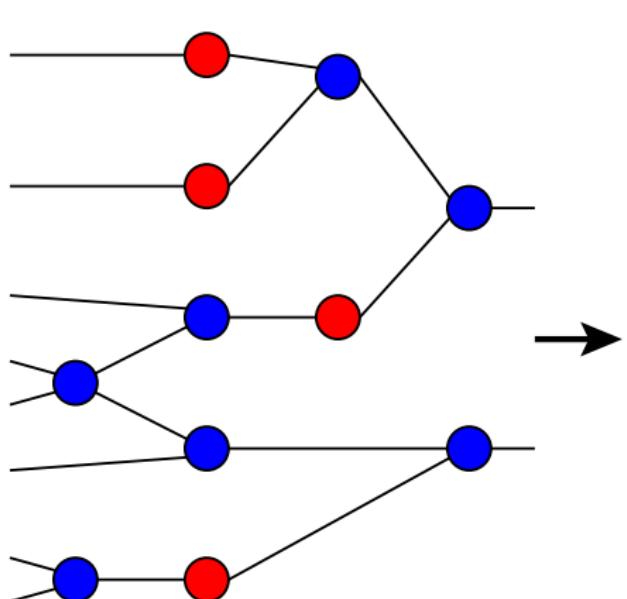
2. Schritt

Vom *subject DAG* zu Teilbäumen (hier durch einfaches Auftrennen der verzweigenden Kanten):



Zerlegung in Teilbäume

Mit Beibehaltung redundanter Knoten (erlaubt unter Umständen bessere Überdeckungen, die sonst nicht möglich wären):



Triviale Überdeckung

Durch Überführung der Netzliste in kanonische Form ist sichergestellt, dass auf jeden Fall eine triviale (meistens sehr schlechte) Überdeckung existiert.

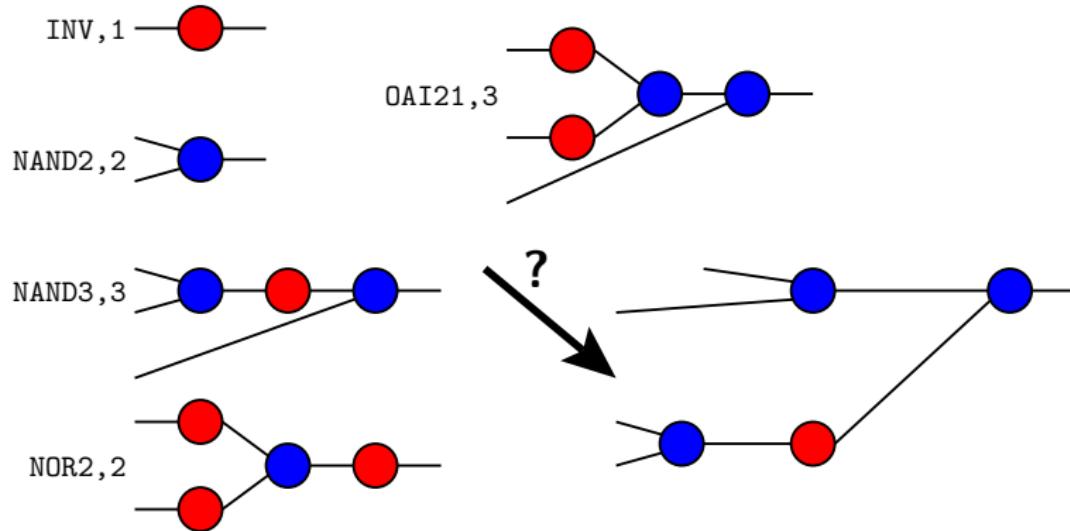
→ Initiale Technologie-Abbildung

Am Ende soll durch den Einsatz komplexer Komponenten aus der Technologie-Bibliothek eine bessere Lösung gefunden werden:

1. Matching: Suche nach gültigen Überdeckungen für alle Knoten aller Teilbäume eines Baums (Ersatz von Knoten der Teilbäume durch isomorphe Bäume, die Komponenten der Technologie-Bibliothek entsprechen)
2. Überdeckung: Konstruiere eine optimale Überdeckung des Baums durch Zusammenfügen der optimalen Überdeckungen von Teilbäumen

Überdeckungsbeispiel

Suche der optimalen Überdeckung für einen der Teilbäume aus dem letzten Beispiel:

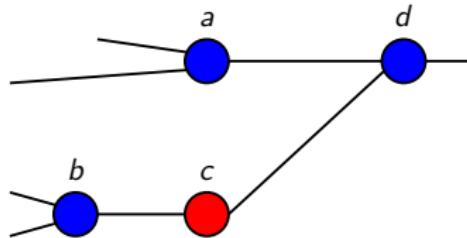


Eine triviale (nicht optimale) Überdeckung ist mit den Knoten INV und NAND2 möglich.

Topologische Ordnung

Die Suche wird durch topologische Ordnung (*topological order*) der Knoten gesteuert:

- ▶ Eine topologische Ordnung ist eine Listenanordnung, in der kein Knoten vor seinen Vorgängern steht (kann auch mathematisch als Relation formuliert werden)
- ▶ Es existieren in der Regel mehrere topologische Ordnungen einer Netzliste



Beispiele von topologischen Ordnungen für den bereits betrachteten Teilbaum:

$$\{a,b,c,d\}$$

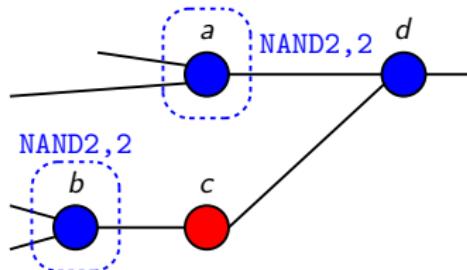
$$\{b,a,c,d\}$$

$$\{b,c,a,d\}$$

Für die nachfolgenden Betrachtungen wird die Ordnung $\{a,b,c,d\}$ gewählt.

Suche der optimalen Überdeckung

Beginnend bei den Blättern entsprechend der gewählten topologischen Ordnung $\{a, b, c, d\}$:

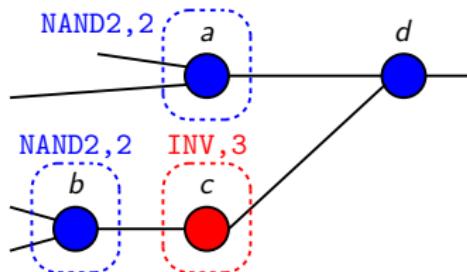


Die Überdeckungen für a und b sind trivial, da nur ein Match (NAND2) existiert

Kostenfunktion ist additiv (Fläche), d. h. Kosten einer gefundenen Überdeckung sind Kosten des Matches plus Kosten aller Überdeckungen der Eingänge dieses Matches. Kosten von primären (hier nicht explizit eingezeichneten) Eingangsknoten sind 0.

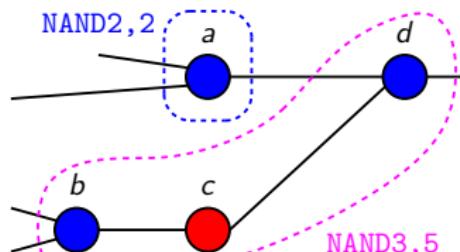
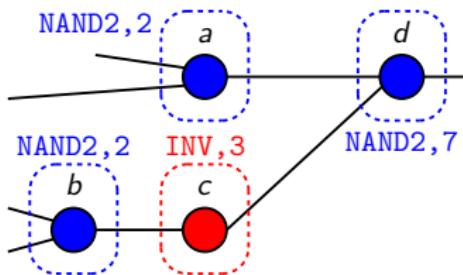
- Optimale Überdeckungen von a und b sind gefunden, Kosten sind jeweils 2, es kann die optimale Überdeckung für c gesucht werden.

Suche der optimalen Überdeckung des Knotens c



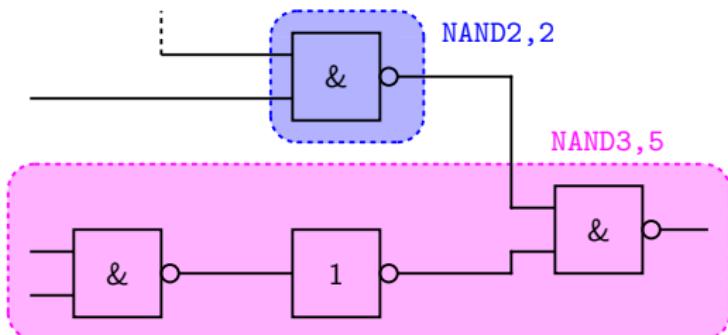
Ebenfalls trivial, da nur ein Match (INV) existiert. Kosten ergeben sich aus der Summe der Kosten eines INV-Knotens und des Eingangsknotens b .

Für den Knoten d existieren zwei Matches:



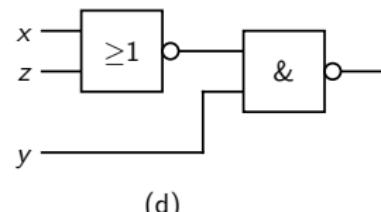
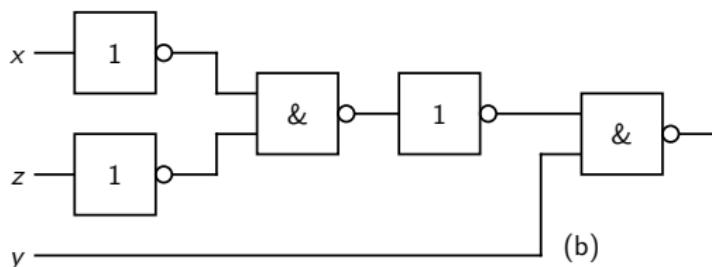
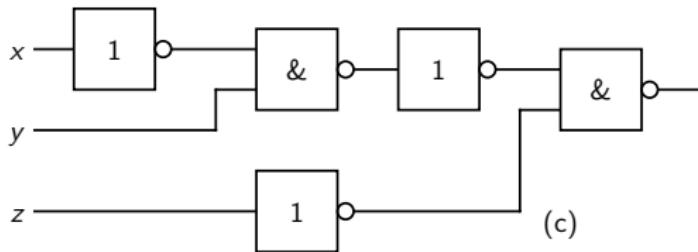
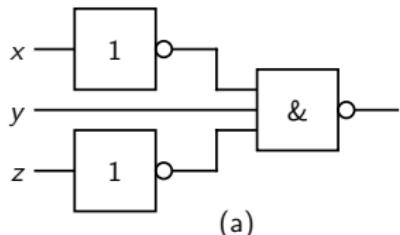
- Der Match mit NAND3 liefert eine kostengünstigere Überdeckung und ist daher optimal.

Resultierende Technologie-Abbildung für den betrachteten Teilbaum



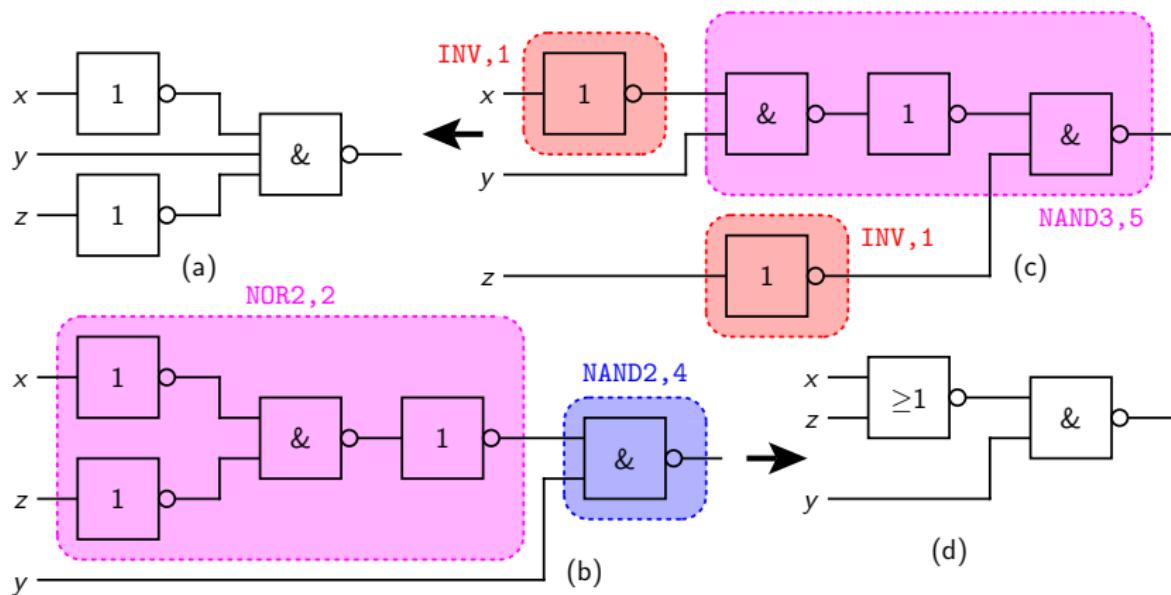
- ▶ Gleiche Prozedur für alle anderen Teilbäume des DAG
- ▶ Topologische Ordnung stellt sicher, dass bei der Ermittlung eines neuen Matches Kosten aller benötigten Eingangsknoten bekannt sind
- ▶ Gleiche Vorgehensweise für einen allgemeinen DAG ist nicht möglich, da für Knoten mit $\text{fanout} \geq 2$ in den jeweiligen Teilbäumen verschiedene Matches optimal sein können

Verschiedene Dekompositionen



Optimalität der Lösung hängt von der Wahl der Dekomposition ab.
Aus Laufzeitgründen wird jedoch nur eine Dekomposition betrachtet
(zumeist eine kanonische Form).

Einfluss der Dekomposition



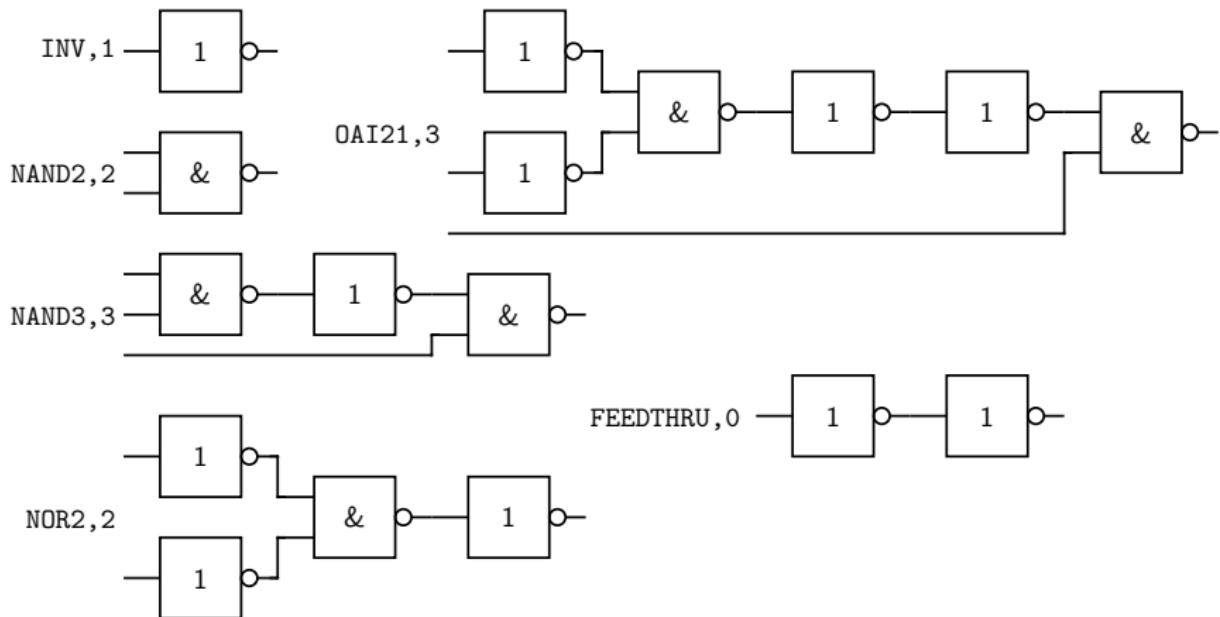
Dekomposition (b) führt zu einer kostengünstigeren Technologie-Abbildung als Dekomposition (c).

Einfluss der Dekomposition (fortgesetzt)

- ▶ Suboptimale Lösungen werden durch Festlegung auf eine Dekompositionsmöglichkeit zugunsten der Laufzeit in Kauf genommen.
- ▶ Bei unzufriedenstellender Qualität der Technologie-Abbildung kann der Dekompositionsschritt wiederholt werden (Rückkopplungsschleife).
- ▶ Einschränkung auf kanonische Darstellung und Gatter mit maximal 2 Eingängen bei *subject DAG* ist durchaus sinnvoll: Im letzten Beispiel wäre für die Variante (a) sonst nur ein Match möglich, kostengünstigere Alternative (d) somit prinzipiell ausgeschlossen.
- ▶ Innerhalb einer Technologie-Bibliothek sind durchaus mehrere Dekompositionen einer Komponente denkbar, da der Einfluss auf die Laufzeit vertretbar bleibt.

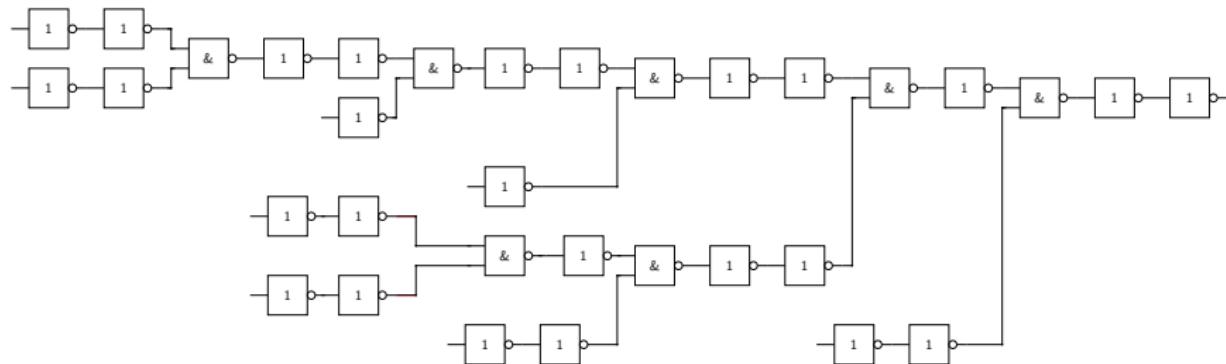
FEEDTHRU-Element

Zusätzliche lokale Optimierungen durch Ergänzung der Technologie-Bibliothek mit einem „kostenlosen“ FEEDTHRU-Element:



Einsatz von FEEDTHRU

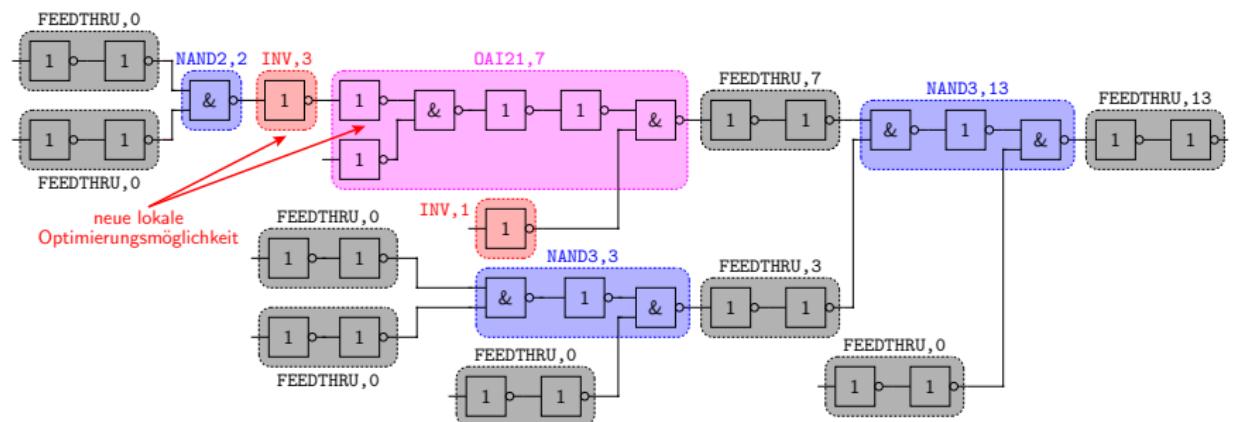
FEEDTHRU wird bei allen Verbindungen eingefügt, die nicht bereits einen Inverter enthalten:



FEEDTHRU entspricht keiner physikalischen Zelle und fügt weder zur Fläche noch zur Verzögerung zusätzliche Kosten hinzu (eine Überdeckung mit FEEDTHRU wird anschließend wieder entfernt)!

Kostengünstigere Lösung mit FEEDTHRU

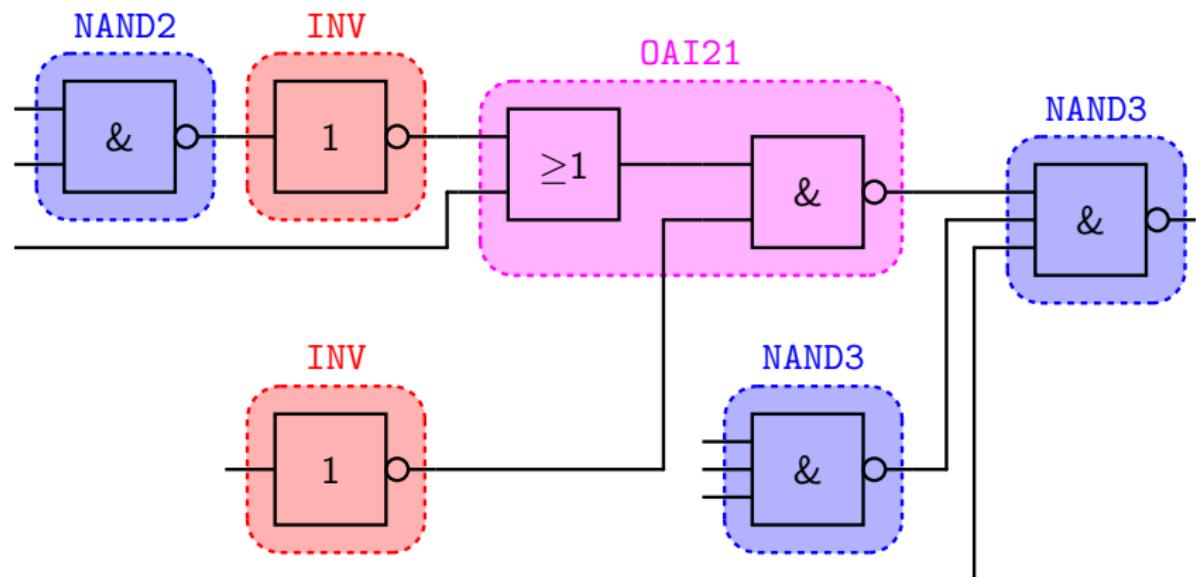
Vergleiche mit der Lösung aus der letzten Vorlesung:



Der eigentliche Gewinn entsteht durch die Möglichkeit der Trennung beider aufeinander folgenden Inverter, die aus einem FEEDTHRU-Element stammen (und sonst nicht vorhanden wären).

Ergebnis der Technologie-Abbildung

... das der auf der letzten Seite vorgestellten Überdeckung entspricht (und ausschließlich aus den in der Bibliothek enthaltenen Komponenten besteht):



Optimierung des Zeitverhaltens

Vorgestellter Algorithmus minimiert die Chipfläche. Was ist mit Timing?

Prinzipiell auch anwendbar, es gibt jedoch einige Probleme:

- ▶ Kostenfunktion ist zwar additiv, jedoch ist eine optimale Überdeckung eines Knotens auch von seinen Nachfolgern abhängig (Lastfaktor), für die es noch keine Überdeckung gibt.
- ▶ Alle Möglichkeiten vormerken, später (wenn die Überdeckung des Nachfolgers feststeht) die zutreffende auswählen.
- ▶ Wenn nur Verzögerungszeit als Kostenfunktion optimiert wird, verschwendet die resultierende Überdeckung Fläche.
- ▶ In einem zweiten Durchlauf Teilbäume mit positivem Schlupf flächenoptimal überdecken.

3. Partitionierung und Floorplanning

Einordnung

Prinzipiell kann eine auf die Zieltechnologie abgebildete Netzliste bereits platziert und verdrahtet werden. Bei den meisten komplexen Entwürfen ist die Anzahl der zu platzierenden Gatter/Basiszellen jedoch zu hoch um mit den bekannten Platzierungsverfahren in akzeptabler Zeit verarbeitet zu werden. Weiterhin kann es architekturspezifische Einschränkungen geben, z. B. maximale Anzahl der Anschlüsse oder hierarchische Verdrahtungsanordnung (siehe cluster-basierte FPGA). Diese Einschränkungen machen oft zusätzliche Optimierungsschritte notwendig, die vor der Platzierung durchgeführt werden:

- ▶ Partitionierung: Aufteilung der Gesamtschaltung in Teilschaltungen
- ▶ Floorplanning: Dimensionierung und Anordnung der Teilschaltungen innerhalb des Gesamtlayouts

Hintergründe

- ▶ Ursprünglich: Aufteilung einer komplexen Schaltung auf mehrere Träger (ICs, Boards)
 - ▶ Vorgegebene maximale Anzahl von Außenanschlüssen
 - ▶ Unterscheidung in *System-Level* und *Board-Level-Partitioning*
- ▶ aus heutiger Sicht meistens zur Vereinfachung der anschließenden Platzierung innerhalb eines IC („teile und herrsche“), *Chip-Level-Partitioning*

Gleiche Ein- und Ausgabe, unterschiedliche Vorgaben bzw. Zielstellungen, z. B. strikte Begrenzung der Anzahl der Anschlüsse oder der Gesamtfläche, starke oder kaum Unterschiede zwischen Partitionsgrößen usw.

→ Hier weitestgehend eingeschränkt auf *Chip-Level-Partitioning*

RENTsche Regel

RENTsche Regel (*Rent's rule*)

ist eine quantitative Beschreibung des Zusammenhangs zwischen Anzahl der Zellen (Gatter) einer digitalen Schaltung n_G und Anzahl ihrer Außenanschlüsse n_P :

$$n_P = t \cdot n_G^r$$

mit t als Anzahl von Pins pro Zelle und r als empirische (positive) Konstante < 1 (RENTscher Exponent, *Rent's exponent*).

Abhängig von der Schaltungsart liegen die typischen Werte für r zwischen 0,5 und 0,8.

- Eine Partitionierung in wenige große Teilschaltungen hat im statistischen Mittel mehr Anschlüsse pro Teilschaltung zur Folge als eine Partitionierung in viele kleine Teilschaltungen.



Edward Francis RENT
(1926–2008,
amerikanischer
Ingenieur)

Partitionierung vs. Clustering

- ▶ Gemeinsamkeiten: Beides Techniken zur Reduzierung der Anzahl der zu platzierenden Elemente, Trennung zwischen globaler und lokaler Verdrahtbarkeit
- ▶ Clustering: *Bottom-Up*, graduelles Wachstum der Cluster ausgehend von einer initialen Zuweisung, Anzahl der Cluster ist meistens nicht begrenzt (und sehr hoch), dafür ist meistens die Größe eines Clusters begrenzt (und verhältnismäßig klein)
- ▶ Partitionierung: *Top-Down*, Aufteilung der Netzliste in wenige Blöcke (Partitionen) unter Minimierung der Verbindungen zwischen diesen, Anzahl der Partitionen meistens begrenzt
- ▶ In der Literatur nicht immer klar getrennt, gelegentlich wird Clustering als Sonderfall der Partitionierung betrachtet

Grundbegriffe

Graph $G = (V, E)$ ist ein Tupel aus Menge der Knoten $V \neq \emptyset$ und Menge der Kanten $E \subseteq V \times V$. $H = (V', E')$ ist ein Teilgraph von G , wenn $V' \subseteq V$, $E' \subseteq V' \times V'$ sowie $E' \subseteq E$ gilt.

Hypergraph ist ein Graph, bei dem Kanten mehr als zwei Knoten verbinden können.

Multigraph ist ein Graph, bei dem gleiche Knotenpaare durch mehrere Kanten verbunden sein können.

Vollständiger Graph ist ein Graph, bei dem alle Knotenpaare durch Kanten verbunden sind.

Stern ist ein Graph bei dem ein Knoten ausgezeichnet ist. Dieser ist jeweils mit allen anderen Knoten verbunden, die wiederum mit keinen anderen außer diesem Knoten verbunden sind.

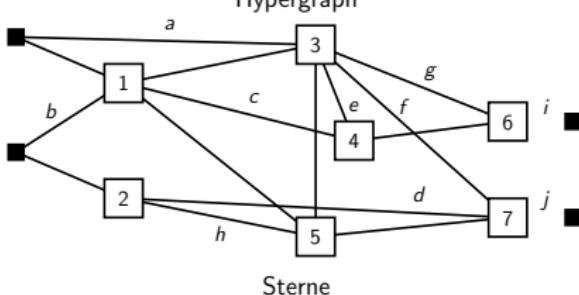
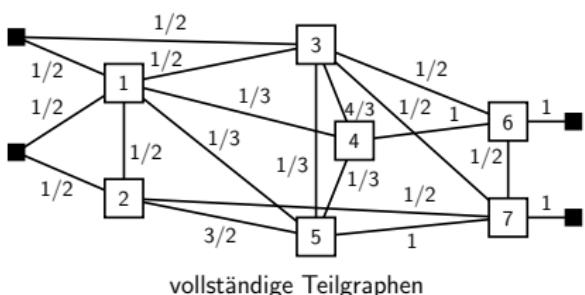
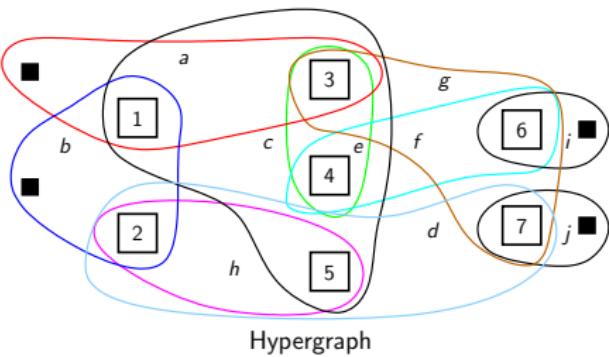
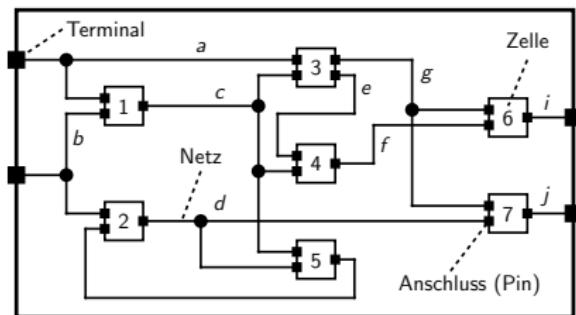
Weitere Verfeinerungen

- ▶ Bewertung der Kanten mit einer Gewichtsfunktion w_n
- ▶ Definition des Verbindungsgrades zweier Knoten c_{ij} :
 - ▶ Anzahl der Kanten, die zwei Knoten verbinden (bei ungewichteten Kanten)
 - ▶ Summe der Gewichte aller Kanten, die zwei Knoten verbinden (bei gewichteten Kanten) ➔ *Connectivity*:

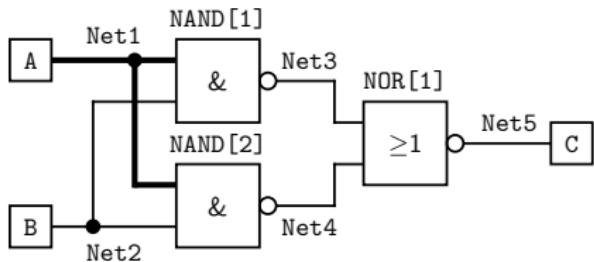
$$c_i = \sum_{j=1}^n c_{ij}$$

Viele Clustering und Partitionierungsalgorithmen basieren auf der Auswertung des Verbindungsgrades, z. B. Suche nach den Knoten mit hohem Verbindungsgrad als Kerne von Clustern.

Verschiedene Darstellungen einer Netzliste



Eine Beispielschaltung als Verbindungsgraph und Netzliste



(A: Net1)

(B: Net2)

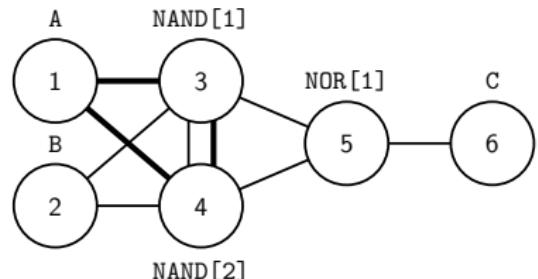
(C: Net5)

(NAND[1]: IN1 Net1, IN2 Net2, OUT Net3)

(NAND[2]: IN1 Net1, IN2 Net2, OUT Net4)

(NOR[1]: IN1 Net3, IN2 Net4, OUT Net5)

Pinorientiert



(Net1: A, NAND[1].IN1, NAND[2].IN1)

(Net2: B, NAND[1].IN2, NAND[2].IN2)

(Net3: NAND[1].OUT, NOR[1].IN1)

(Net4: NAND[2].OUT, NOR[1].IN2)

(Net5: NOR[1].OUT, C)

Netzorientiert

Auszug aus einer „echten“ Netzliste

```
...
INV_X1 _0850_ (
    .I(\stage_1.write_address [5]),
    .ZN(_0013_)
);
NAND2_X1 _0851_ (
    .A1(\stage_1.turn ),
    .A2(\stage_1.read_address [5]),
    .ZN(_0014_)
);
OAI21_X1 _0852_ (
    .A1(\stage_1.turn ),
    .A2(_0013_),
    .B(_0014_),
    .ZN(\stage_1.address1 [5])
);
...
```

Dichte eines Graphen

Dichte

eines Graphen $G = (V, E)$ ist wie folgt definiert:

$$\varepsilon = \frac{|E|}{\binom{|V|}{2}} \text{ mit } \binom{n}{2} = \frac{n!}{2 \cdot (n-2)!}$$

Für einen zusammenhängenden Graphen gilt

$$0 < \varepsilon \leq 1,$$

wobei der Maximalwert $\varepsilon = 1$ einem vollständigen Graphen entspricht.

Intuitiver Lösungsansatz: Suche nach disjunkten Teilgraphen mit hoher Dichte.

Ergänzende Materialien

Edward Francis RENT und RENTsche Regel:

- ▶ T. M. RENT, *Rent's Rule: A Family Memoir*, IEEE Solid State Circuits Magazine, Vol. 2, Issue 1, pp. 14–20, Winter 2010,
Dateiname auf dem Handout-Server:
[Zusatzmaterial/eda_v7_e_f_rent.pdf](#)

RENTsche Regel und frühe Abschätzung der Leitungsverzögerungen:

- ▶ D. STROOBANDT, *Recent Advances in System-Level Interconnect Prediction*, IEEE Circuits and Systems Society Newsletter, Volume 11, Number 4, pp. 1; 4–20; 48, December 2000,
Dateiname auf dem Handout-Server:
[Zusatzmaterial/eda_v7_rents_rule.pdf](#)

Zusammenfassung

- ▶ Allgemeine Problemstellung der Technologie-Abbildung
- ▶ Technologie-Abbildung mittels Graph-Überdeckung
- ▶ Einführung in Partitionierung

Rechnergestützer Entwurf digitaler Systeme: Vorlesungs-Gesamtübersicht

1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme:

Gliederung der 8. Vorlesung

3. Partitionierung und Floorplanning

3.1. Partitionierung

- 3.1.1. Grundlagen
- 3.1.2. Clustering
- 3.1.3. RAJARAMAN-WONG-Algorithmus
- 3.1.4. KERNIGHAN-LIN-Algorithmus

3.2. Floorplanning

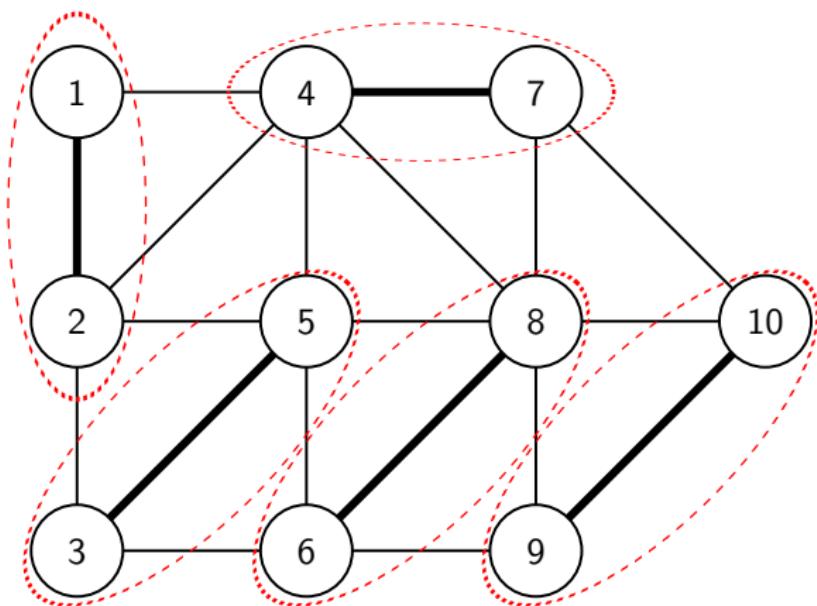
- 3.2.1. Definitionen und Datenstrukturen
- 3.2.2. Floorplan-Sizing-Algorithmus
- 3.2.3. Lineare Optimierung
- 3.2.4. Pinzuordnung

Formulierung des Clustering-Problems

- ▶ Es sei eine Netzliste in Form eines Graphen $G = (V, E)$ gegeben.
- ▶ Teile den Graphen in k Teilgraphen $\{G_1, G_2, \dots, G_k\}$ auf, so dass jeder Teilgraph höchstmögliche Dichte aufweist.
- ▶ Ein NP-äquivalentes Problem
- ➡ Ausschließlich heuristische Lösungsansätze

Clustering aus graphentheoretischer Sicht

Hier für Cluster-Größe 2 dargestellt:



„Dickere“ Kanten stellen Mehrfachverbindungen dar.

Einteilung der Heuristiken

- ▶ Nach der Anziehungsfunction (*Attract*)
 - ▶ Absorptions-basiert
 - ▶ Verbindungsgrad-basiert
 - ▶ Flächen-basiert (speziell für große Cluster gut geeignet)
- ▶ Hierarchisch, z. B. durch sukzessive Bipartition

Clustering kann die Leistung der anschließenden Partitionierung verbessern, da es in der Regel Knoten größeren Grades (Grad = Anzahl der ein- und ausgehenden Kanten) erzeugt.

Annahmen

- ▶ Jeder Knoten darf eine eigene Verzögerungszeit haben
- ▶ Verbindungen zwischen den Clustern haben eine Einheitsverzögerung
- ▶ Verbindungen innerhalb der Cluster haben keine Verzögerung
- ▶ Maximale Cluster-Größe ist durch eine Konstante vorgegeben
- ▶ Knotenvervielfachung ist erlaubt

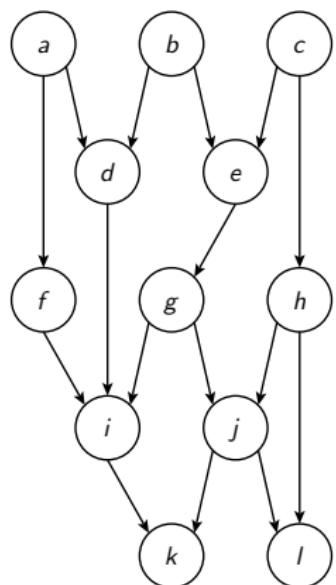
Optimierungsziel:

- ▶ Verzögerungszeit (maximale Verzögerung zwischen einem primären Eingang und einem primären Ausgang)

Vorgehensweise

1. Matrix Δ mit maximalen Verzögerungszeiten aufstellen.
2. Topologische Ordnung aller Knoten außer primärer Eingangsknoten aufstellen.
3. Labeling: Alle Knoten außer primärer Eingänge in topologischer Ordnung besuchen, Cluster vormerken.
 - ▶ Initiale Labels entsprechen der Verzögerungszeiten.
 - ▶ Durch Bildung der Teilbäume für jeden Knoten werden die Labels angepasst.
4. Clustering: Alle Knoten in umgekehrter topologischer Ordnung besuchen, feste Cluster-Zuweisung durchführen.
 - ▶ Zuerst die Cluster von primären Ausgängen bilden.
 - ▶ Anschließend iterativ die Knoten aus der Eingabemenge der bereits gebildeten Cluster bearbeiten.

Beispiel



Δ	a	b	c	d	e	f	g	h	i	j	k	l
a	0	0	0	1	0	1	0	0	2	0	3	0
b	0	0	0	1	1	0	2	0	3	3	4	4
c	0	0	0	0	1	0	2	1	3	3	4	4
d	0	0	0	0	0	0	0	0	1	0	2	0
e	0	0	0	0	0	0	1	0	2	2	3	3
f	0	0	0	0	0	0	0	0	1	0	2	0
g	0	0	0	0	0	0	0	0	1	1	2	2
h	0	0	0	0	0	0	0	0	0	1	2	2
i	0	0	0	0	0	0	0	0	0	0	1	0
j	0	0	0	0	0	0	0	0	0	0	1	1
k	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	0

Topologische Ordnung: $\{d, e, f, g, h, i, j, k, l\}$

Labeling

1. Primäre Eingänge erhalten als Label ihre Verzögerung zugewiesen, alle anderen Knoten werden mit 0 initialisiert.
2. Alle Knoten außer primärer Eingänge werden in topologischer Ordnung besucht, wobei entsprechende Teilbäume erzeugt werden. Für Knoten v und Teilbaum G_v :
 - 2.1. $l_v(x) = l(x) + \Delta(x, v)$, wobei $x \in G_v \setminus \{v\}$ gilt.
 - 2.2. Alle Knoten $x \in G_v \setminus \{v\}$ nach l_v absteigend in eine Menge S einsortieren.
 - 2.3. Knoten aus der Menge S sortiert entnehmen und dem Cluster C_v zuordnen, bis dieser voll ist.
 - 2.4. $l_1 = \max\{l_v(x) : x \in PI\}$, $l_2 = \max\{l_v(x) + D : x \in S\}$, wobei D für Inter-Cluster-Verzögerung steht
 - 2.5. $l(v) = \max\{l_1, l_2\}$

Labeling-Beispiel (1)

$D = 3$, Cluster-Größe = 4, Verzögerung = 1:

- ▶ Primäre Eingänge: $I(a) = I(b) = I(c) = 1$, $C_a = \{a\}$,
 $C_b = \{b\}$, $C_c = \{c\}$.
- ▶ Teilbaum $G_d = \{a, b, d\}$:

$$I_d(a) = I(a) + \Delta(a, d) = 1 + 1 = 2$$

$$I_d(b) = I(b) + \Delta(b, d) = 1 + 1 = 2$$

- ▶ $S = G_d \setminus \{d\} = \{a, b\}$
- ▶ $C_d = \{a, b, d\}$
- ▶ $I_1 = \max\{I_d(a), I_d(b)\} = 2$, $I_2 = 0$
- ▶ $I(d) = \max\{I_1, I_2\} = \max\{2, 0\} = 2$

Analoge Vorgehensweise für alle verbleibenden Knoten.

Labeling-Beispiel (2)

- Teilbaum $G_e = \{b, c, e\}$:

$$l_e(b) = l(b) + \Delta(b, e) = 1 + 1 = 2$$

$$l_e(c) = l(c) + \Delta(c, e) = 1 + 1 = 2$$

- $S = G_e \setminus \{e\} = \{b, c\}, C_e = \{e, b, c\}$
- $l_1 = \max\{l_e(b), l_e(c)\} = 2, l_2 = 0$
- $l(e) = \max\{l_1, l_2\} = \max\{2, 0\} = 2$
- Teilbaum $G_f = \{a, f\}$:

$$l_f(a) = l(a) + \Delta(a, f) = 1 + 1 = 2$$

- $S = G_f \setminus \{f\} = \{a\}, C_f = \{a, f\}$
- $l_1 = \max\{l_f(a)\} = 2, l_2 = 0$
- $l(f) = \max\{l_1, l_2\} = \max\{2, 0\} = 2$

Labeling-Beispiel (3)

- Teilbaum $G_g = \{b, c, e, g\}$:

$$l_g(b) = l(b) + \Delta(b,g) = 1 + 2 = 3$$

$$l_g(c) = l(c) + \Delta(c,g) = 1 + 2 = 3$$

$$l_g(e) = l(e) + \Delta(e,g) = 2 + 1 = 3$$

- $S = G_g \setminus \{g\} = \{b, c, e\}$, $C_g = \{b, c, e, g\}$

- $l_1 = \max\{l_g(b), l_g(c)\} = 3$, $l_2 = 0$

- $l(g) = \max\{l_1, l_2\} = \max\{3, 0\} = 3$

- Teilbaum $G_h = \{h, c\}$:

$$l_h(c) = l(c) + \Delta(c,h) = 1 + 1 = 2$$

- $S = G_h \setminus \{h\} = \{c\}$, $C_h = \{c, h\}$

- $l_1 = \max\{l_h(c)\} = 2$, $l_2 = 0$, $l(h) = \max\{l_1, l_2\} = \max\{2, 0\} = 2$

Labeling-Beispiel (4)

- Teilbaum $G_i = \{a, b, c, d, e, f, g, i\}$:

$$l_i(a) = l(a) + \Delta(a, i) = 1 + 2 = 3$$

$$l_i(b) = l(b) + \Delta(b, i) = 1 + 3 = 4$$

$$l_i(c) = l(c) + \Delta(c, i) = 1 + 3 = 4$$

$$l_i(d) = l(d) + \Delta(d, i) = 2 + 1 = 3$$

$$l_i(e) = l(e) + \Delta(e, i) = 2 + 2 = 4$$

$$l_i(f) = l(f) + \Delta(f, i) = 2 + 1 = 3$$

$$l_i(g) = l(g) + \Delta(g, i) = 3 + 1 = 4$$

- $S = G_i \setminus \{i\} = \{a, b, c, d, e, f, g\}$, $C_i = \{i, g, e, c\}$ bzw.

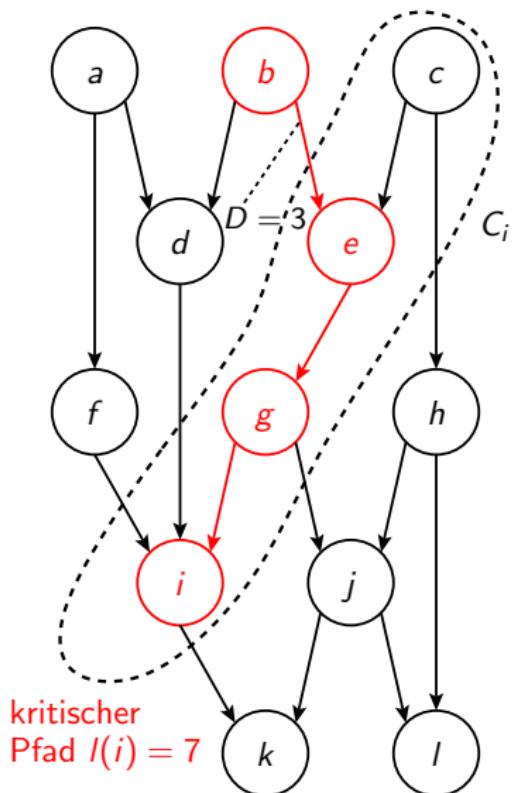
$$C_i = \{i, g, e, b\}$$

- $l_1 = \max\{l_i(c)\} = 4$, $S = \{b, a, d, f\} \neq \emptyset$

↳ $l_2 = \max\{l_v(x) + D : x \in S\} = \max\{4 + 3, 3 + 3, 3 + 3, 3 + 3\}$

- $l(i) = \max\{l_1, l_2\} = \max\{4, 7\} = 7$

Verzögerungszeit-Schätzung



- ▶ Die durch das Labeling des Knoten i ermittelte Verzögerung entspricht tatsächlich dem kritischen Pfad (unter Berücksichtigung der Inter-Cluster-Verzögerung)
- ▶ Eine ähnliche Lösung wäre auch bei einer Entscheidung für das Cluster $C_i = \{i, g, e, b\}$ entstanden.

Labeling: Das Endergebnis

Durch analoge Betrachtungen für die Knoten j , k und l wird folgendes ermittelt:

- ▶ $C_j = \{b, e, g, j\}$ bzw. $C_j = \{c, e, g, j\}$, $I(j) = 7$
- ▶ $C_k = \{g, i, j, k\}$, $I(k) = 8$
- ▶ $C_l = \{e, g, j, l\}$, $I(l) = 8$

Zusammenfassung der Labeling- und Clustering-Ergebnisse:

Knoten	Label	Cluster	Knoten	Label	Cluster
a	1	$\{a\}$	g	3	$\{b, c, e, g\}$
b	1	$\{b\}$	h	2	$\{c, h\}$
c	1	$\{c\}$	i	7	$\{c, e, g, i\}$
d	2	$\{a, b, d\}$	j	7	$\{b, e, g, j\}$
e	2	$\{b, c, e\}$	k	8	$\{g, i, j, k\}$
f	2	$\{a, f\}$	l	8	$\{e, g, j, l\}$

Clustering

1. Primäre Ausgänge der Menge L zuordnen. Menge der Cluster als leer initialisieren: $S_c = \emptyset$.
2. Einen Knoten v aus der Menge L entfernen und seinen Cluster C_v (wie beim Labeling ermittelt) fest zuweisen: $L = L \setminus \{v\}$, $S_c = S_c \cup \{C_v\}$
3. Eingänge von C_v berechnen: $\text{input}(C_v)$
4. Einen Knoten x aus der Menge $\text{input}(C_v)$ entfernen und zur Menge L hinzufügen
5. Stoppen, wenn $L = \emptyset$
6. Schritt 2 wiederholen.

Anmerkung: Knoten in der Menge L sollten in umgekehrter topologischer Ordnung hinzugefügt und entfernt werden.

Clustering-Beispiel (1)

- ▶ $L = \{k, l\}$, $S_c = \emptyset$
- ▶ $L = L \setminus \{k\} = \{l\}$, $S_c = C_k = \{g, i, j, k\}$,
 $\text{input}(C_k) = \{f, d, e, h\}$
- ➡ $L = \{l\} \cup \{f, d, e, h\} = \{l, f, d, e, h\}$
- ▶ $L = L \setminus \{l\} = \{f, d, e, h\}$,
 $S_c = S_c \cup C_l = S_c \cup \{e, g, j, l\} = \{C_k, C_l\}$,
 $\text{input}(C_l) = \{b, c, h\}$
- ➡ $L = \{f, d, e, h\} \cup \{b, c, h\} = \{f, d, e, h, b, c\}$
- ▶ $L = L \setminus \{f\} = \{d, e, h, b, c\}$,
 $S_c = S_c \cup C_f = S_c \cup \{a, f\} = \{C_k, C_l, C_f\}$, $\text{input}(C_f) = \emptyset$
- ➡ $L = \{d, e, h, b, c\} \cup \emptyset = \{d, e, h, b, c\}$

Clustering-Beispiel (2)

- ▶ $L = L \setminus \{d\} = \{e, h, b, c\}$,
 $S_c = S_c \cup C_d = S_c \cup \{a, b, d\} = \{C_k, C_l, C_f, C_d\}$,
 $\text{input}(C_d) = \emptyset$
- ↳ $L = \{e, h, b, c\} \cup \emptyset = \{e, h, b, c\}$
- ▶ $L = L \setminus \{e\} = \{h, b, c\}$,
 $S_c = S_c \cup C_e = S_c \cup \{b, c, e\} = \{C_k, C_l, C_f, C_d, C_e\}$,
 $\text{input}(C_e) = \emptyset$
- ↳ $L = \{h, b, c\} \cup \emptyset = \{h, b, c\}$
- ▶ $L = L \setminus \{h\} = \{b, c\}$,
 $S_c = S_c \cup C_h = S_c \cup \{c, h\} = \{C_k, C_l, C_f, C_d, C_e, C_h\}$,
 $\text{input}(C_h) = \emptyset$
- ↳ $L = \{b, c\} \cup \emptyset = \{b, c\}$

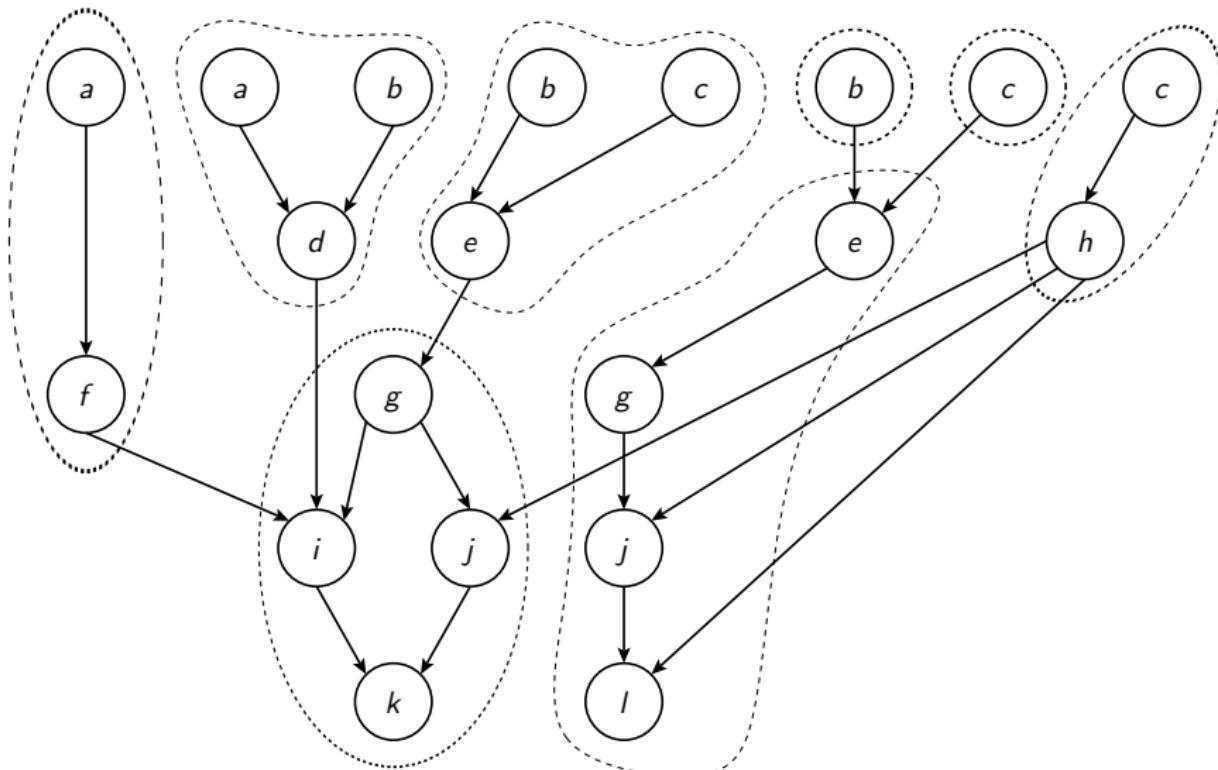
Clustering-Beispiel (3)

- ▶ $L = L \setminus \{b\} = \{c\}$,
 $S_c = S_c \cup C_b = S_c \cup \{b\} = \{C_k, C_l, C_f, C_d, C_e, C_h, C_b\}$,
 $input(C_b) = \emptyset$
- ➡ $L = \{c\} \cup \emptyset = \{c\}$
- ▶ $L = L \setminus \{c\} = \emptyset$,
 $S_c = S_c \cup C_c = S_c \cup \{c\} = \{C_k, C_l, C_f, C_d, C_e, C_h, C_b, C_c\}$,
 $input(C_c) = \emptyset$
- ➡ $L = \emptyset \cup \emptyset = \emptyset$
- ➡ Ende

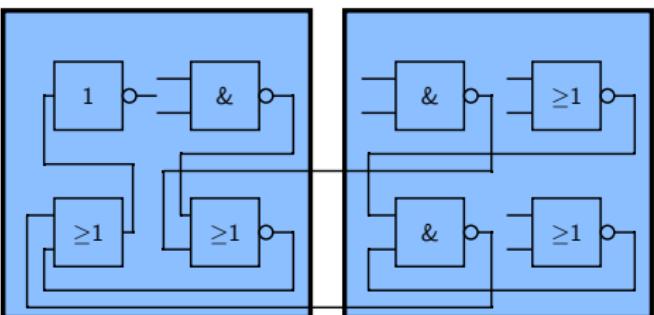
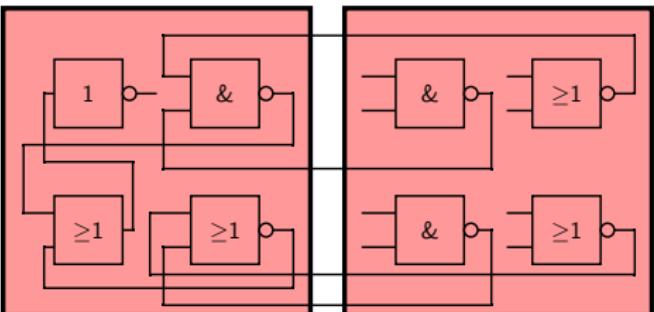
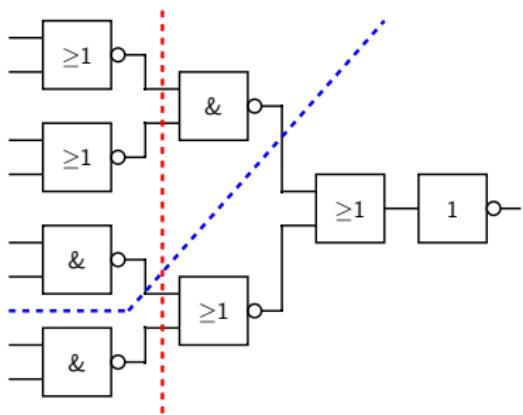
Cluster (einige Knoten müssen dupliziert werden):

$\{\{g, i, j, k\}, \{e, g, j, l\}, \{a, f\}, \{a, b, d\}, \{b, c, e\}, \{c, h\}, \{b\}, \{c\}\}$

Das Ergebnis



Zwei Partitionierungsvarianten einer Schaltung



Formalisierung des Partitionierungsproblems

Es sei eine Schaltung in Form eines Graphen $G = (V, E)$ gegeben. Jeder Knoten $v \in V$ besitze eine Fläche $s(v)$, jede Kante $e \in E$ besitze eine Wichtung $w(e)$. Gesucht ist eine Aufteilung des Graphen in k Teilgraphen $G_i = (V_i, E_i)$ mit $i = 1, 2, \dots, k$, so dass

$$Z = \sum_{e \in \Psi} w(e) \rightarrow \min$$

gilt, wobei Ψ der Menge der geschnittenen Kanten (Kanten, die externe Verbindungen zwischen den Teilgraphen repräsentieren) entspricht. Ψ wird auch als Schnittmenge (*cutset*) bezeichnet.

→ Minimierung der Anzahl von Signalen mit langen Laufzeiten.

Als Zusatzbedingung wird oft auch die Partitionsgröße (Summenbildung über $s(v)$ der beteiligten Knoten) eingeschränkt.

Globaler Ablauf

Für $k = 2$, kann verallgemeinert werden:

1. V zufällig in zwei gleich große Teilmengen aufteilen.
2. Ein Knotenpaar aus verschiedenen Teilmengen mit dem größten Einfluss auf die Schnittmenge vertauschen (und fixieren).
3. Schritt 2 solange wiederholen, bis alle Knoten fixiert sind (ein Pass ist damit beendet).
4. Innerhalb eines Passes nur die Tauschsequenz beibehalten, die zur größten Verringerung der Schnittmenge führt, alle anderen Vertauschungen aufheben.
5. Neue Pässe (ab Schritt 2) solange ausführen, bis keine Verbesserung mehr erreicht werden kann.

Schnittkosten sind entweder Anzahl der geschnittenen Kanten oder Summe der Wichtungen der geschnittenen Kanten.

Zusätzliche Vereinbarungen

- ▶ Kosten eines Knotens $D(v)$:

$$D(v) = \sum v_{\text{extern}} - \sum v_{\text{intern}},$$

mit v_{extern} als Anzahl der geschnittenen Kanten des Knotens v und v_{intern} als Anzahl der nicht geschnittenen Kanten des Knotens v .

- ▶ Gewinnwert Δg beim Vertauschen der Knoten a und b :

$$\Delta g = D(a) + D(b) - 2 \cdot c(a,b),$$

mit $c(a,b) = 1$ wenn $(a,b) \in E$ und $c(a,b) = 0$ wenn $(a,b) \notin E$.

- ➡ Großer Gewinnwert entspricht einem günstigen Tausch.

Wenn die vertauschten Knoten eine gemeinsame Kante besitzen, bleibt sie auch nach dem Tausch bestehen, daher ist die Korrektur mit dem Faktor $2 \cdot c(a,b)$ bei solchen Knotenpaaren notwendig.

Zusätzliche Vereinbarungen (fortgesetzt)

- ▶ Positiver Gewinn eines Passes G_m :

$$G_m = \sum_{i=1}^m \Delta g_i,$$

d. h. die Summe der Gewinnwerte aller Vertauschungen. Die Zielstellung eines Passes ist es

$$G_m \rightarrow \max$$

zu erreichen. Dabei werden die Vertauschungen vorerst nur „vorgemerkt“, um dann die Folge mit dem größten positiven Gewinn in dem jeweiligen Pass zu realisieren.

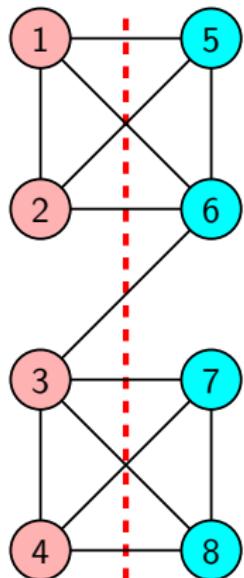
- ▶ Anmerkung: Auch negative Gewinnwerte sind möglich und zulässig, wenn der Gesamtgewinn am Ende stimmt!

Detaillierter Ablauf des Algorithmus

Nach zufälliger Aufteilung von V in zwei gleich große Teilmengen A und B :

1. $i = 1$, $D(v)$ für alle $v \in V$ berechnen.
2. a_i und b_i mit $\Delta g_i = D(a_i) + D(b_i) - 2 \cdot c(a_i, b_i) = \max$ vertauschen und fixieren.
3. Wenn alle Knoten fixiert sind, weiter mit Schritt 4 sonst $D(v)$ für alle nicht fixierten Knoten, die mit a_i und b_i verbunden sind neu berechnen, $i = i + 1$, weiter mit Schritt 2
4. Vertauschungssequenz 1 bis m ($1 \leq m \leq i$) bestimmen, bei der $G_m = \max$ gilt. Wenn $G_m > 0$, weiter mit Schritt 5, sonst ENDE.
5. m Vertauschungen durchführen, Knotenfixierungen aufheben, weiter mit Schritt 1.

Beispiel (1)



Schnittkosten: 9,
nicht fixiert: 1, 2, 3, 4, 5, 6, 7, 8

$$D(1) = 1 \quad D(5) = 1$$

$$D(2) = 1 \quad D(6) = 2$$

$$D(3) = 2 \quad D(7) = 1$$

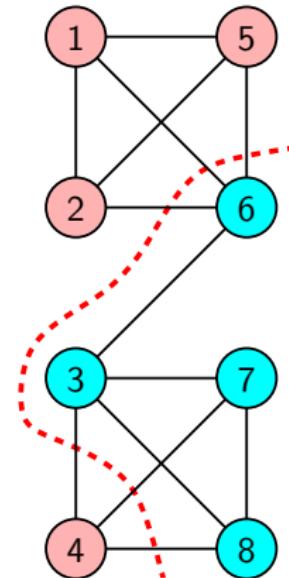
$$D(4) = 1 \quad D(8) = 1$$

$$\Delta g_1 = 2 + 1 - 0 = 3$$

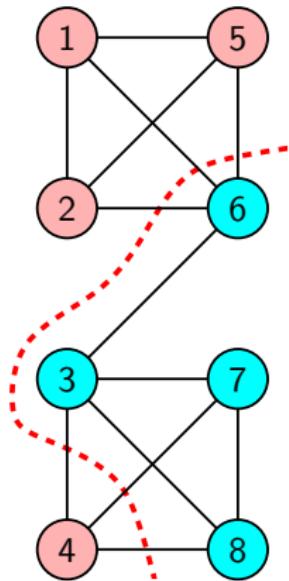
Austausch: (3,5)

$$G_1 = \Delta g_1 = 3$$

⇒ Schnittkosten: 6



Beispiel (2)



Schnittkosten: 6,
nicht fixiert: 1, 2, 4, 6, 7, 8

$$D(1) = -1$$

$$D(2) = -1$$

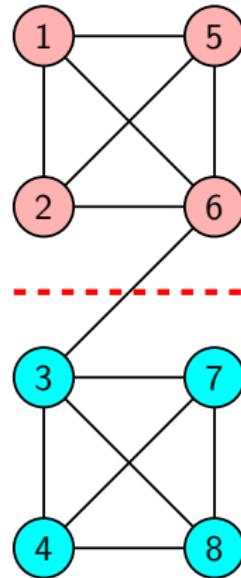
$$D(4) = 3$$

$$\Delta g_2 = 3 + 2 - 0 = 5$$

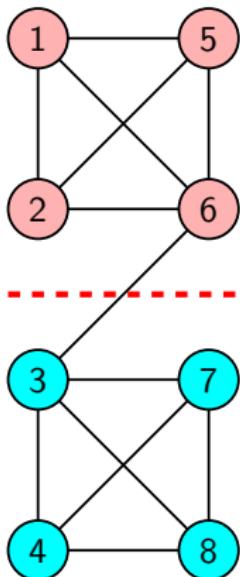
Austausch: (4,6)

$$G_2 = G_1 + \Delta g_2 = 8$$

⇒ Schnittkosten: 1



Beispiel (3)



Schnittkosten: 1,
nicht fixiert: 1, 2, 7, 8

$$D(1) = -3 \quad D(7) = -3$$

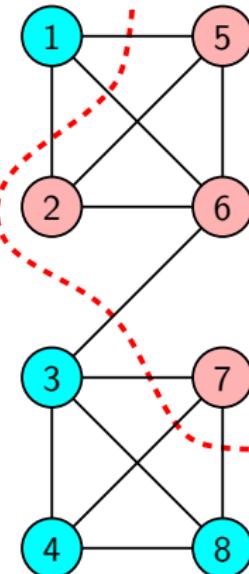
$$D(2) = -3 \quad D(8) = -3$$

$$\Delta g_3 = -3 - 3 - 0 = -6$$

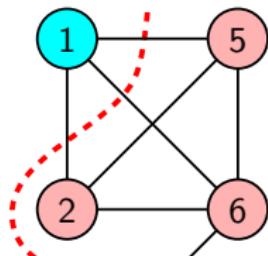
Austausch: (1,7)

$$G_3 = G_2 + \Delta g_3 = 2$$

■ Schnittkosten: 7



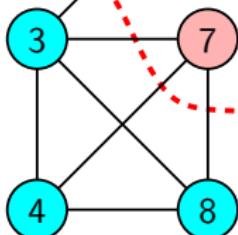
Beispiel (4)



Schnittkosten: 7,
nicht fixiert: 2, 8

$$D(2) = -1 \quad D(8) = -1$$

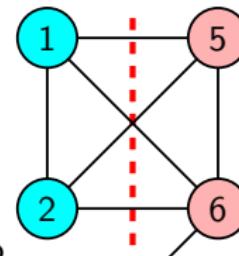
$$\Delta g_4 = -1 - 1 - 0 = -2$$



Austausch: (2,8)

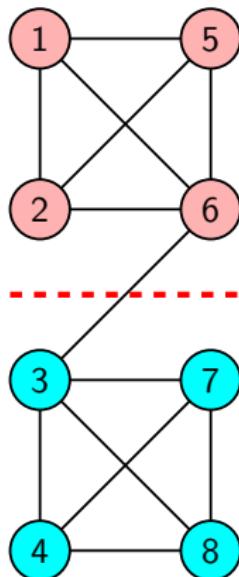
$$G_4 = G_3 + \Delta g_4 = 0$$

⇒ Schnittkosten: 9



Alle Knoten sind fixiert ⇒ Pass 1 ist zu Ende, die Vertauschungssequenz ((3,5),(4,6)) wird ausgeführt, $G_m = 8$. $G_m > 0$, d. h. es wird ein 2. Pass benötigt, dieser führt jedoch keine Verbesserung mehr herbei (und wird hier aus Übersichtlichkeitsgründen weggelassen).

Das Ergebnis



ist optimal, denn die Schnittmenge kann (in einem zusammenhängenden Graphen) nicht weniger als ein Element enthalten. Der Algorithmus liefert aber nicht grundsätzlich eine optimale Lösung.

- Die Rechenkomplexität ist $\mathcal{O}(n^3)$, kann durch Vorsortieren der Knoten auf $\mathcal{O}(n^2 \cdot \log n)$ gebracht werden.
- Oft steht die Lösung bereits nach nur 4 Pässen fest.

Erweiterungen:

- Ungleiche Partitionsgrößen
- Ungleiche Fläche einzelner Knoten
- Mehr als 2 Partitionen

Weitere Verfahren

FIDUCCIA-MATTHEYES-Algorithmus:

- ▶ Verschiebung einer Zelle statt Vertauschung von Paaren
- ▶ Einführung des Gleichgewichtsfaktors zur Vermeidung der Degradation
- ▶ Berücksichtigung von unterschiedlichen Zellengrößen
- ▶ Berücksichtigung von Netzen mit mehr als 2 Terminals
- ▶ Rechenzeitkomplexität von $\mathcal{O}(n)$

Simulierte Abkühlung (*simulated annealing*):

- ➡ gleicher Ablauf, lediglich die Kostenfunktion muss angepasst werden: Kein umschließendes Rechteck, sondern Anzahl der Schnittkanten (ggf. gewichtete Summe)

Ergänzende Materialien

Originalveröffentlichungen zu KERNIGHAN-LIN- und FIDUCCIA-MATTHEYES-Algorithmen:

- ▶ B. W. KERNIGHAN and S. LIN, *An Efficient Heuristic Procedure for Partitioning Graphs*, Bell Systems Technical Journal, Vol. 49, Issue 2, pp. 291–317, February 1970, Dateiname auf dem Handout-Server:
[Zusatzmaterial/eda_v8_kl_paper.pdf](#)
- ▶ C. M. FIDUCCIA and R. M. MATTHEYES, *A Linear-Time Heuristic for Improving Network Partitions*, in Proceeding of 19th Design Automation Conference, pp. 175–181, IEEE Press, June 14–16, Las Vegas, 1982, Dateiname auf dem Handout-Server: [Zusatzmaterial/eda_v8_fm_paper.pdf](#)

Zusammenfassung

- ▶ Formalisierung des Clustering-Problems
- ▶ Clustering nach RAJARAMAN und WONG
- ▶ Partitionierung nach KERNIGHAN und LIN

Rechnergestützer Entwurf digitaler Systeme: Vorlesungs-Gesamtübersicht

1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme:

Gliederung der 9. Vorlesung

3. Partitionierung und Floorplanning

3.1. Partitionierung

- 3.1.1. Grundlagen
- 3.1.2. Clustering
- 3.1.3. RAJARAMAN-WONG-Algorithmus
- 3.1.4. KERNIGHAN-LIN-Algorithmus

3.2. Floorplanning

- 3.2.1. Definitionen und Datenstrukturen
- 3.2.2. Floorplan-Sizing-Algorithmus
- 3.2.3. Lineare Optimierung
- 3.2.4. Pinzuordnung

Einleitung

Voraussetzungen:

- ▶ Partitionen mit ungefähren Flächenwerten (ausgehend von der Fläche einzelner Zellen)
- ➡ Ableitung der möglichen Konfigurationen (Seitenverhältnisse)

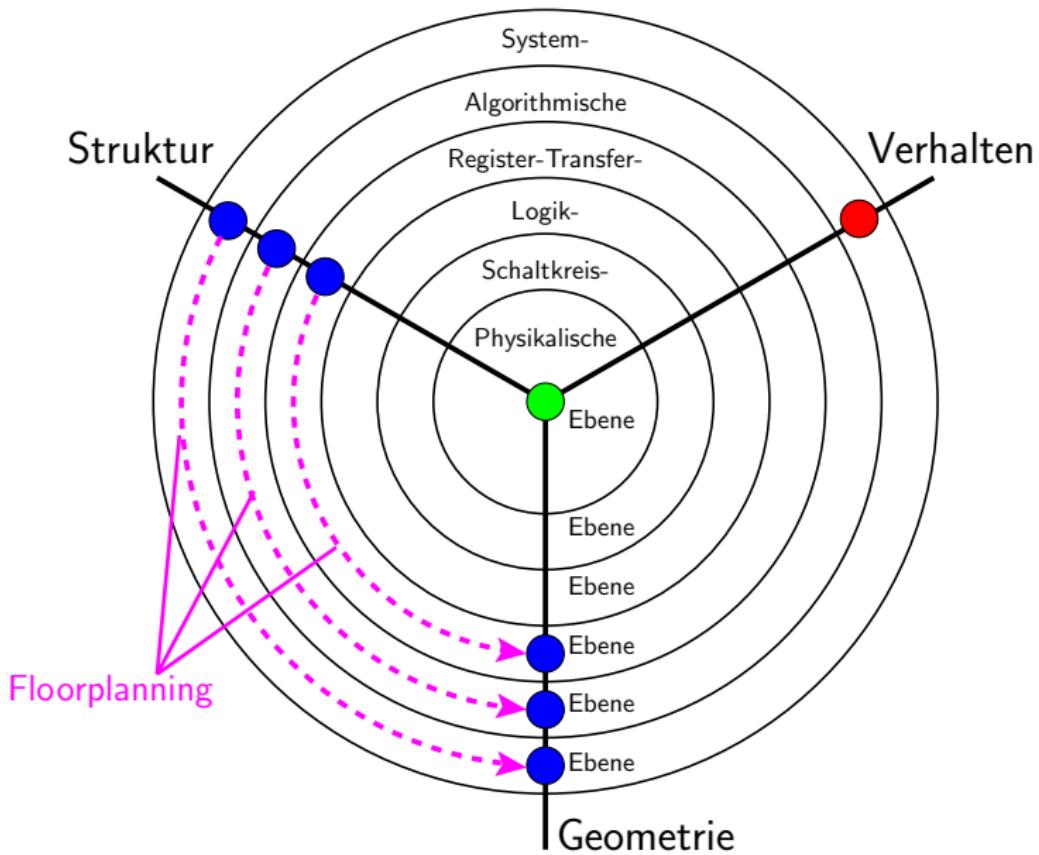
Aufgaben:

- ▶ Festlegung äußerer Anschlüsse von Partitionen
- ▶ Festlegung der Seitenverhältnisse von Partitionen
- ▶ Anordnung der Partitionen innerhalb der verfügbaren Chipfläche

Sinn:

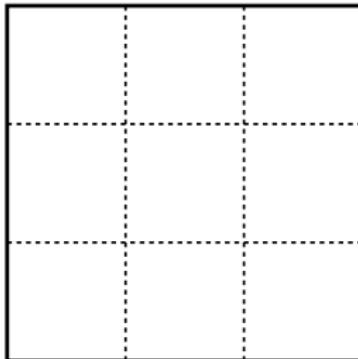
- ▶ Vereinfachung der Aufgaben von Platzierung und Verdrahtung (die dann lokal innerhalb der Partitionen durchgeführt werden können)

Einordnung in den Entwurfsprozess



Beispiel eines einfachen Floorplans

Topzelle:



Mögliche Konfigurationen:

Block A: $h = 4, w = 1$; $h = 1, w = 4$;
 $h = 2, w = 2$

Block B: $h = 2, w = 1$; $h = 1, w = 2$

Block C: $h = 3, w = 1$; $h = 2, w = 2$;
 $h = 1, w = 4$

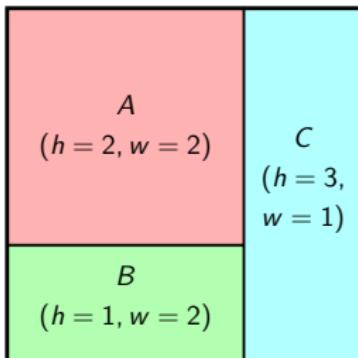
Auswahl einer Anordnung mit minimaler Gesamtfläche:

Block A: $h = 2, w = 2$

Block B: $h = 1, w = 2$

Block C: $h = 3, w = 1$

Gesamtfläche: 9



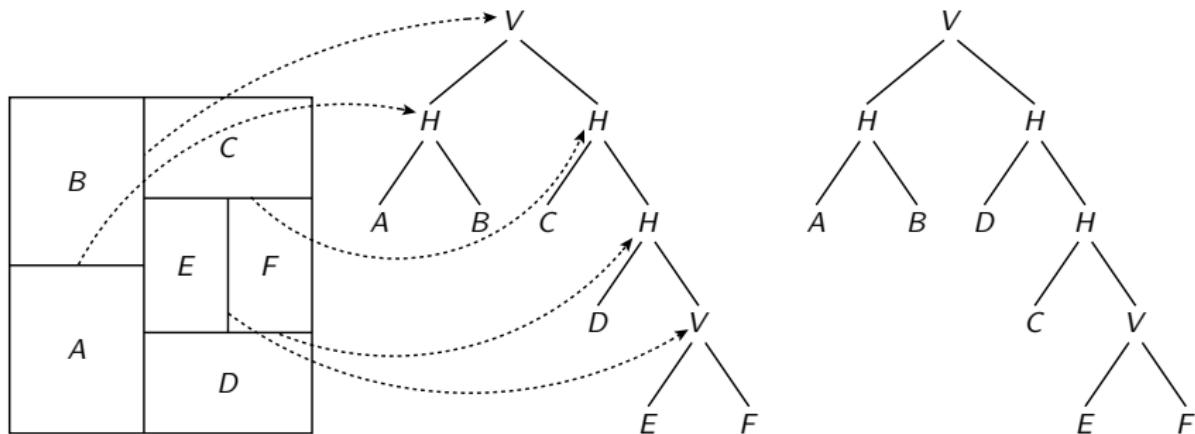
Optimierungsziele von Floorplanning-Algorithmen

- ▶ Gesamtfläche, repräsentiert durch
 - ▶ Fläche des umschließenden Rechtecks
 - ▶ eventuelle Einschränkungen bezüglich der Form und Abmessungen der Topzelle
- ▶ Gesamtverbindungslänge, repräsentiert durch
 - ▶ halben Umfang des umschließenden Rechtecks
 - ▶ Summe der Manhattan-Entfernungen d_{ij} gewichtet mit den Verbindungsgraden c_{ij}

$$L = \sum_{ij} c_{ij} \cdot d_{ij}$$

- ▶ Länge des minimalen Spannbaums (später mehr dazu)
- ▶ Gewichtete Kombination aus beiden Zielen

Darstellung eines Floorplans mit Hilfe eines Schnittbaums (*slicing floorplan tree*)



Alternativ können V und H vertauscht sein (wenn nicht die Richtung der Linie, sondern die Richtung der Aufteilung gemeint ist, z. B. Linie zwischen Block A und Block b als $V \cong$ vertikal aufteilende Linie).

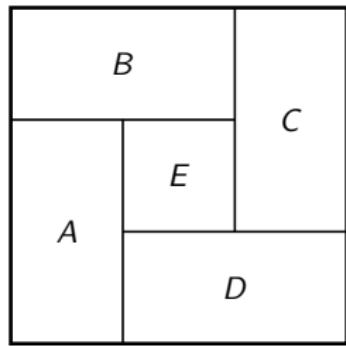
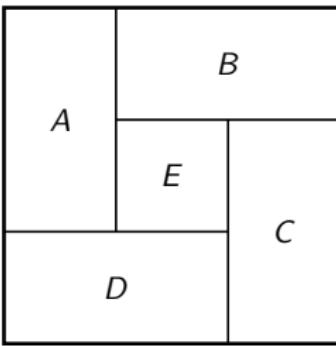
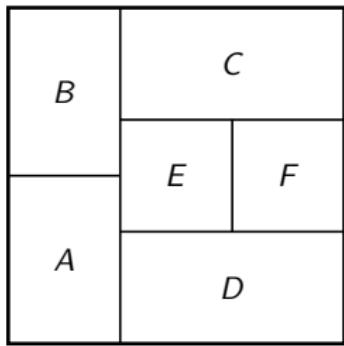
Verschiedene Floorplan-Varianten

Geschnittener Floorplan (*slicing floorplan*): Floorplan, bei dem alle Blöcke durch wiederholte vertikale und horizontale Teilung von übergeordneten Elternblöcken entstanden sind. Dieser ist immer als Schnittbaum darstellbar.

Ungeschnittener Floorplan (*non-slicing floorplan*): Floorplan, der nicht durch vertikale und horizontale Teilung von Elternblöcken gebildet werden kann. Dieser muss aus mindestens fünf Blöcken bestehen und ist nicht als Schnittbaum darstellbar, sondern als Floorplan-Baum.

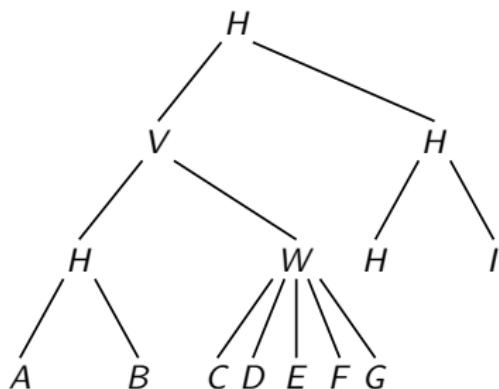
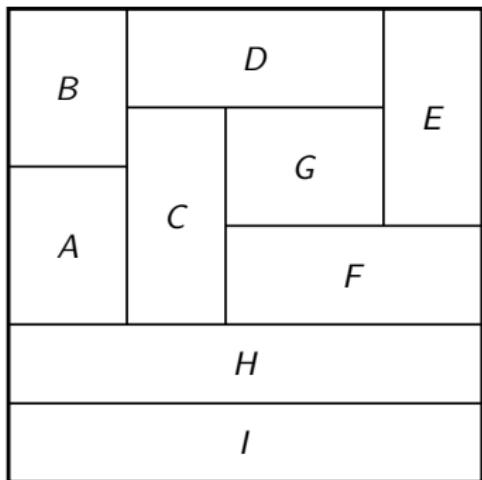
Ordnung l eines Floorplans gibt an, in wie viele Teile ein Rechteck maximal geteilt werden muss, um den Floorplan zu erhalten. Ein ungeschnittener Floorplan hat daher immer mindestens die Ordnung $l = 5$, ein geschnittener Floorplan immer die Ordnung $l = 2$.

Beispiel eines geschnittenen und zweier ungeschnittener Floorpläne



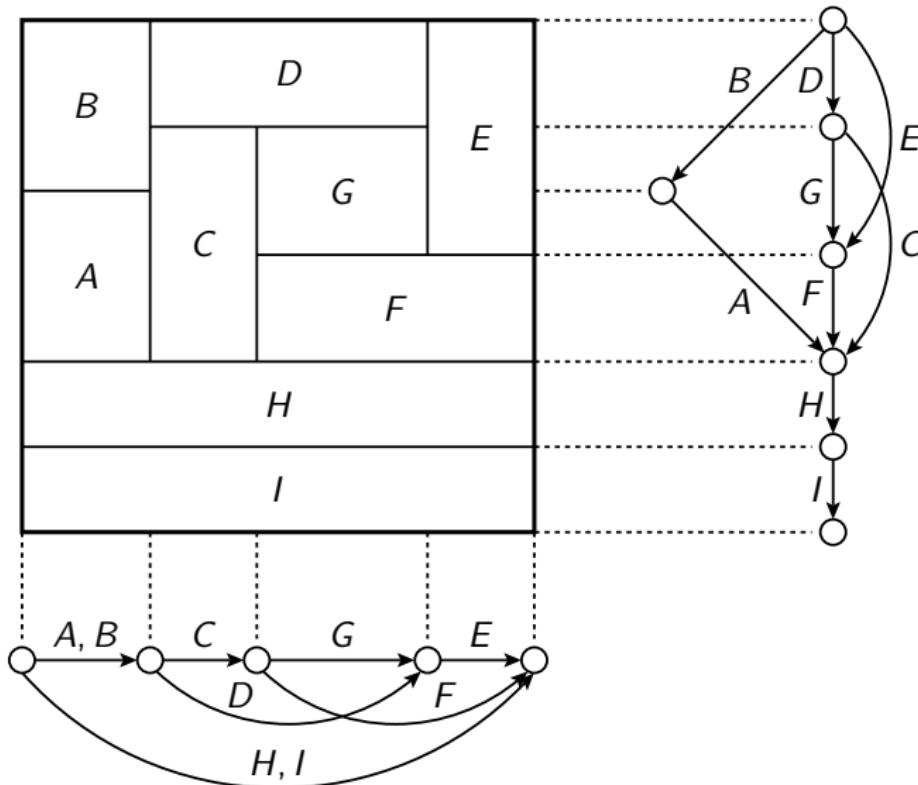
Ungeschnittene Floorpläne können als Floorplan-Bäume dargestellt werden. Allgemein können Floorpläne mit Hilfe von Polargraphen (*polar graph*) eindeutig abgebildet werden. Dabei werden pro Floorplan jeweils 2 Graphen benötigt: ein vertikaler und ein horizontaler.

Darstellung eines ungeschnittenen Floorplans mit einem Floorplan-Baum (*floorplan tree*)



W steht für „*Wheel*“ (Rad), wie der ungeschnittene Telfloorplan aus fünf Rechtecken **C**, **D**, **E**, **F** und **G** bezeichnet wird.

Darstellung eines Floorplans mit Hilfe von Polargraphen



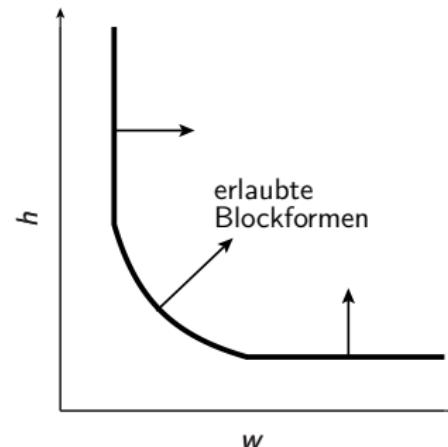
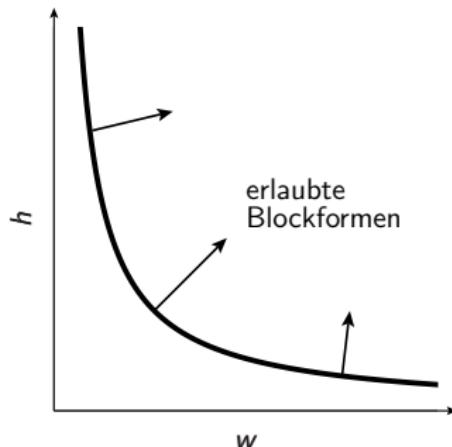
Beschreibung der Blockabmessungen durch Formfunktionen (*shape functions*)

Höhe h und Breite w ergeben die Fläche A einer Zelle: $h \cdot w = A$.

Damit ist die Formfunktion wie folgt definiert:

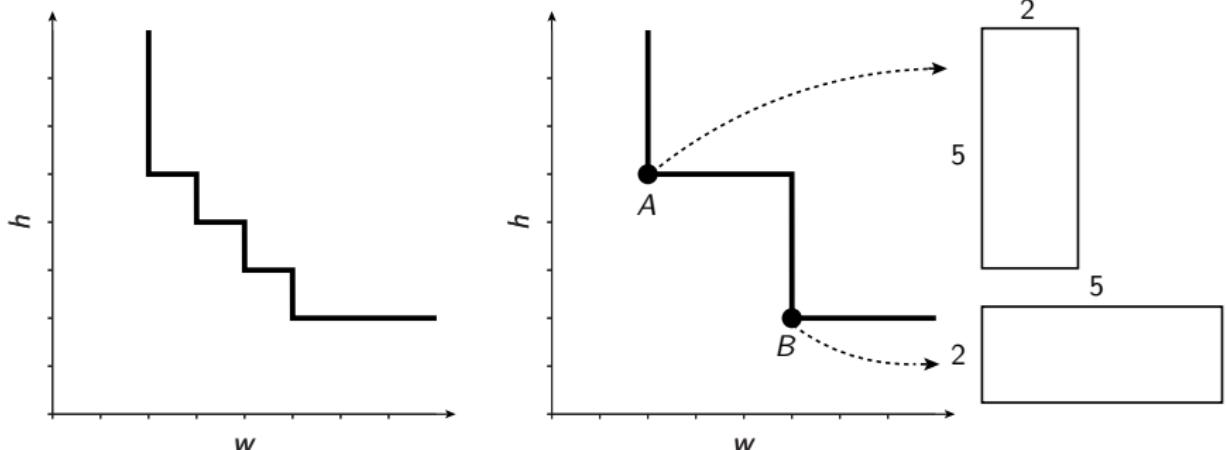
$$h(w) = \frac{A}{w} \text{ bzw. } w(h) = \frac{A}{h}$$

Die dadurch entstehende Hyperbel wird oft zusätzlich durch Minimalvorgaben für Höhe und Breite eingeschränkt:



Formfunktionen und Eckpunkte

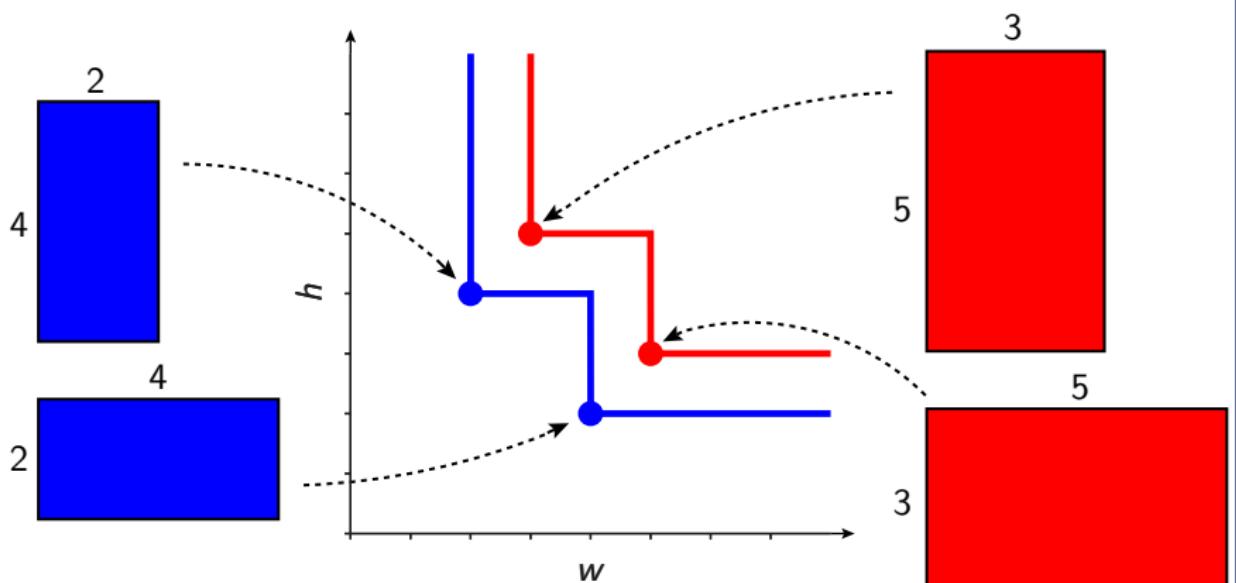
Meistens sind die Formfunktionen durch einige wenige diskrete Wertepaare definiert. Bei Bibliotheksblöcken sind sogar lediglich Rotationen um 90° zugelassen:



Die Punkte A und B repräsentieren dabei die beiden durch Drehung entstehenden Seitenverhältnisse und werden als **Eckpunkte** (*break points*) bezeichnet.

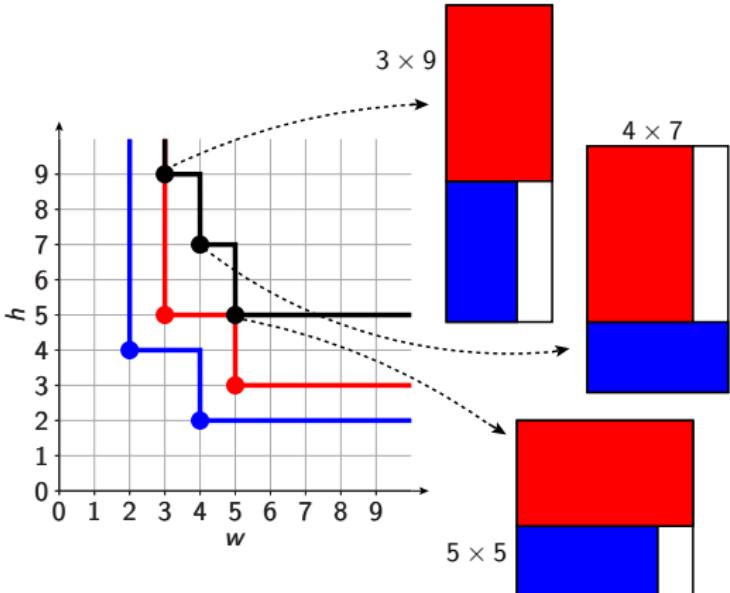
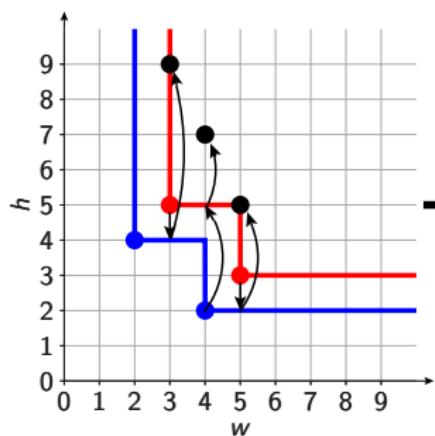
Rechnen mit Formfunktionen

Zur Ermittlung der Formfunktion der Topzelle müssen die Formfunktionen der einzelnen Blöcke ermittelt werden:



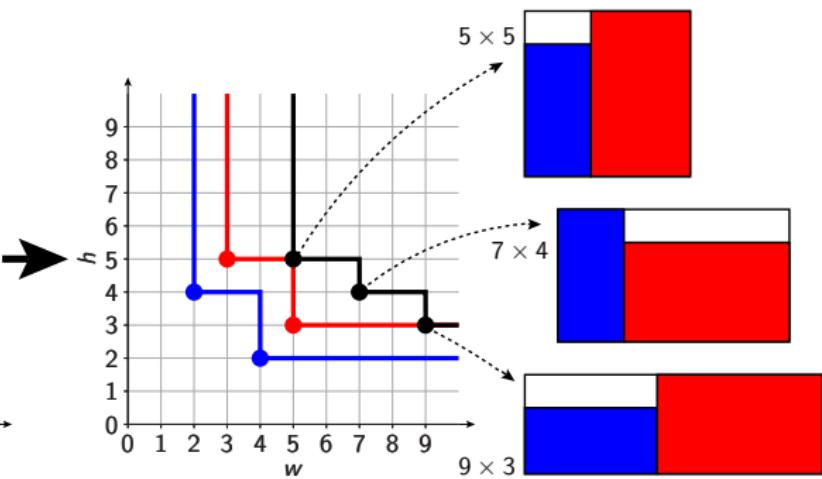
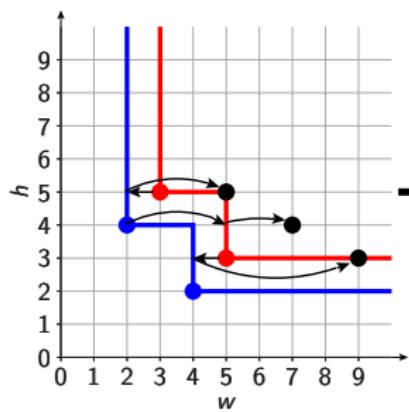
Vertikale Zusammensetzung

Die Formfunktion der Topzelle ergibt sich aus der Überlappung der Formfunktionen der Blöcke (hier am Beispiel der vertikalen Zusammensetzung):



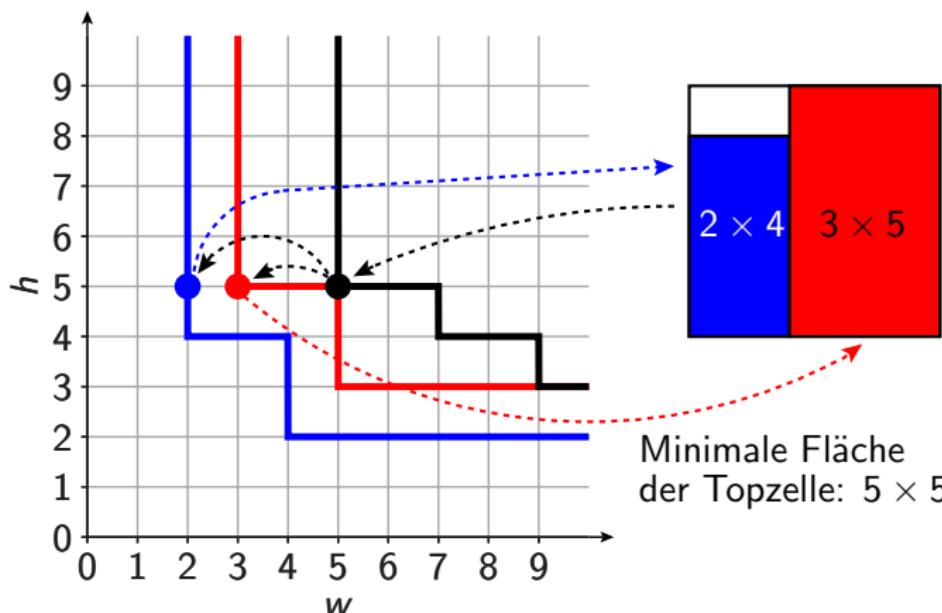
Horizontale Zusammensetzung

Die Formfunktion der Topzelle ergibt sich aus der Überlappung der Formfunktionen der Blöcke (hier am Beispiel der horizontalen Zusammensetzung):



Ermittlung minimaler Abmessungen der Topzelle

Aus der Formfunktion der Topzelle wird ihre minimale Fläche ermittelt (die immer einem der Eckpunkte entspricht). Anhand dieser werden dann die Abmessungen und Orientierung der Blöcke zurückverfolgt (hier am Beispiel der horizontalen Zusammensetzung):



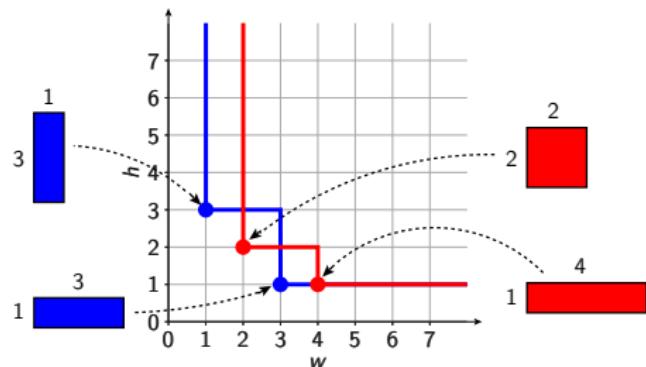
Allgemeiner Ablauf

1. Ermittlung der Formfunktionen aller der Topzelle zugeordneten Blöcke.
2. Ermittlung der Formfunktion der Topzelle
 - ▶ Kombination von Formfunktionen der Blöcke
 - ▶ Ermittlung der optimalen Formfunktion der Topzelle ausgehend von der Überlappung der Formfunktionen der Blöcke
3. Festlegung der Form der Topzelle und hierarchischer Aufbau des Schnittbaums bis alle Basisblöcke erreicht worden sind.

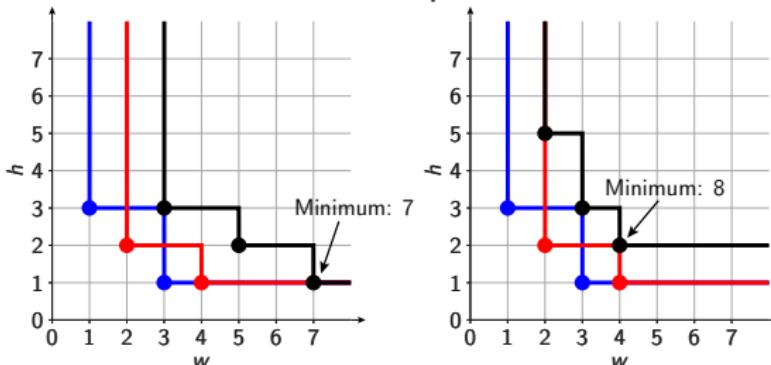
Polynomiale Laufzeitkomplexität bei geschnittenen Floorplänen (Ordnung 2), NP-Komplexität bei ungeschnittenen Floorplänen (Ordnung 5 und mehr).

Einfaches Beispiel mit 2 Blöcken

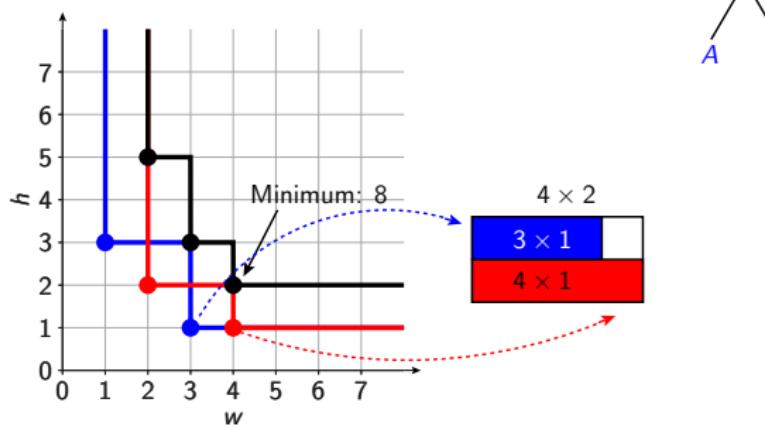
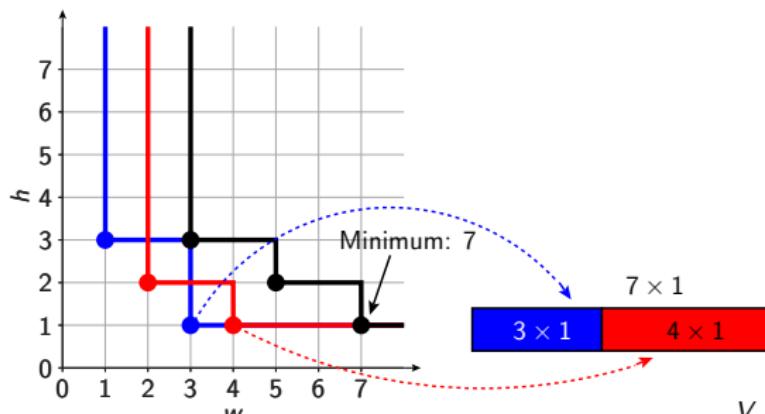
Formfunktionen der einzelnen Blöcke



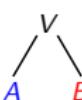
Ermittlung der Formfunktion der Topzelle:



Rückverfolgung der Formfunktion einzelner Blöcke



Beim Optimierungsziel
„Gesamtfläche der
Topzelle“ ist die
horizontale
Zusammensetzung
günstiger.



STOCKMEYER-Algorithmus

Formalisierte Umsetzung von Floorplan-Sizing im STOCKMEYER-Algorithmus (ausgehend von einem gegebenen suboptimalen Floorplan):

1. Der Schnittbaum wird von den Blättern zur Wurzel (*bottom-up*) durchlaufen, dabei werden für jeden Knoten die optimalen Zusammensetzungs-Kandidaten ermittelt. Auswahl erfolgt abhängig von den Abmessungen des rechten und linken Kinder-Knotens (w_l, h_l) und (w_r, h_r) .
2. Die Wurzel erhält eine Liste mit verschiedenen Zusammenfügungsmöglichkeiten.
3. Der Schnittbaum wird von der Wurzel zu den Blättern (*top-down*) durchlaufen, dabei werden die Entscheidungen zu Abmessungen und Orientierung der Blöcke (in den Blättern) festgelegt.

Auswahl-Kriterium bei STOCKMEYER-Algorithmus (1)

V-Knoten (vertikale Schnittlinie): Abmessungen der Kinderknoten nach steigender Breite und fallender Höhe vorsortieren. Die Abmessungen des V-Knotens errechnen sich dann als
 $d_x = (w_l + w_r, \max(h_l, h_r)).$

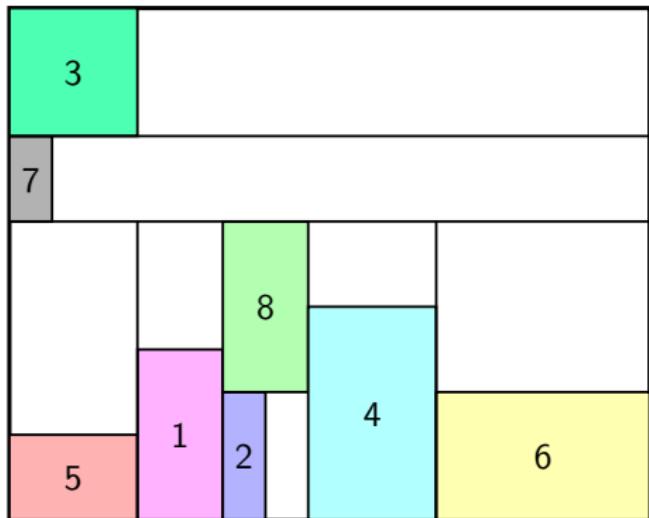
- ▶ $D = \emptyset$. Bei jeweils 1. Abmessung des linken und des rechten Knotens beginnen: $d_1 = (w_{l1} + w_{r1}, \max(h_{l1}, h_{r1})), D = D \cup d_1.$
- ▶ Wenn $h_{l1} < h_{r1}$, dann l_1 mit r_2 kombinieren (nächstes Abmessungspaar des rechten Knotens, gleiches Abmessungspaar des linken Knotens); wenn $h_{l1} > h_{r1}$, dann l_2 mit r_1 kombinieren (nächstes Abmessungspaar des linken Knotens, gleiches Abmessungspaar des rechten Knotens); wenn $h_{l1} = h_{r1}$, dann l_2 mit r_2 kombinieren (jeweils nächste Abmessungspaare). $D = D \cup d_2$ usw.
- ▶ Ende, wenn die Abmessungsliste(n) ausgeschöpft sind.

Auswahl-Kriterium bei STOCKMEYER-Algorithmus (2)

H-Knoten (horizontale Schnittlinie): Abmessungen der Kinderknoten nach fallender Breite und steigender Höhe vorsortieren. Die Abmessungen der *H*-Knotens errechnen sich dann als $(\max(w_l, w_r), h_l + h_r)$.

- ▶ $D = \emptyset$. Bei jeweils 1. Abmessung des linken und des rechten Knotens beginnen: $d_1 = (\max(w_{l1}, w_{r1}), h_{l1} + h_{r1})$, $D = D \cup d_1$.
- ▶ Wenn $w_{l1} < w_{r1}$, dann l_1 mit r_2 kombinieren (nächstes Abmessungspaar des rechten Knotens, gleiches Abmessungspaar des linken Knotens); wenn $w_{l1} > w_{r1}$, dann l_2 mit r_1 kombinieren (nächstes Abmessungspaar des linken Knotens, gleiches Abmessungspaar des rechten Knotens); wenn $w_{l1} = w_{r1}$, dann l_2 mit r_2 kombinieren (jeweils nächste Abmessungspaare). $D = D \cup d_2$ usw.
- ▶ Ende, wenn die Abmessungsliste(n) ausgeschöpft sind.

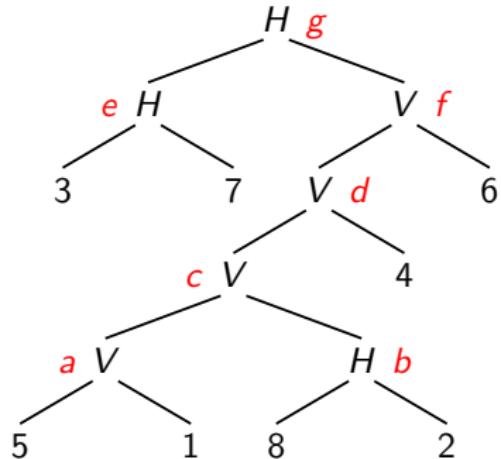
Beispiel zum STOCKMEYER-Algorithmus



Dimensionen der Blöcke 1 bis 8:

$$\{(2,4), (1,3), (3,3), (3,5), (3,2), (5,3), (1,2), (2,4)\}.$$

Fläche der Topzelle: $15 \times 12 = 180$.



Ablauf des STOCKMEYER-Algorithmus (1)

V-Knoten a : $L = \{(2,3), (3,2)\}$, $R = \{(2,4), (4,2)\}$:

- ▶ $l_1 = (2,3)$ und $r_1 = (2,4)$: $(2 + 2, \max\{3,4\}) = (4,4)$.
 $h_l < h_r \Rightarrow$ Als nächstes l_1 und r_2 zusammenfügen:
- ▶ $l_1 = (2,3)$ und $r_2 = (4,2)$: $(2 + 4, \max\{3,2\}) = (6,3)$.
 $h_l > h_r \Rightarrow$ Als nächstes l_2 und r_2 zusammenfügen:
- ▶ $l_2 = (3,2)$ und $r_2 = (4,2)$: $(3 + 4, \max\{2,2\}) = (7,2)$.

Abmessungsliste für den Knoten a : $\{(4,4), (6,3), (7,2)\}$.

Anmerkung: Die Zusammenfügung von $l_2 = (3,2)$ und $r_1 = (2,4)$ führt zu der ungünstigsten Kombination $(5,4)$. Diese wird durch die Wahl der Zusammenfügungskandidaten gar nicht erst betrachtet.

Die Größe der Abmessungsliste für interne Knoten ist somit $\mathcal{O}(|L| + |R|)$ und nicht $\mathcal{O}(|L| \cdot |R|)$

Ablauf des STOCKMEYER-Algorithmus (2)

H-Knoten b : $L = \{(4,2), (2,4)\}$, $R = \{(3,1), (1,3)\}$:

- ▶ $l_1 = (4,2)$ und $r_1 = (3,1)$: $(\max\{4,3\}, 2 + 1) = (4,3)$.
 $w_l > w_r \Rightarrow$ Als nächstes l_2 und r_1 zusammenfügen:
- ▶ $l_2 = (2,4)$ und $r_1 = (3,1)$: $(\max\{2,3\}, 4 + 1) = (3,5)$.
 $w_l < w_r \Rightarrow$ Als nächstes l_2 und r_2 zusammenfügen:
- ▶ $l_2 = (2,4)$ und $r_2 = (1,3)$: $(\max\{2,1\}, 4 + 3) = (2,7)$.

Abmessungsliste für den Knoten b : $\{(4,3), (3,5), (2,7)\}$.

Auch hier: Die ungünstigste Zusammenfügung von $l_1 = (4,2)$ und $r_2 = (1,3)$ mit den Abmessungen (4,5) wird vermieden.

Ablauf des STOCKMEYER-Algorithmus (3)

V-Knoten c : $L = \{(4,4), (6,3), (7,2)\}$, $R = \{(2,7), (3,5), (4,3)\}$:

- ▶ $l_1 = (4,4)$ und $r_1 = (2,7)$: $(4 + 2, \max\{4,7\}) = (6,7)$.
 $h_l < h_r \Rightarrow$ Als nächstes l_1 und r_2 zusammenfügen:
- ▶ $l_1 = (4,4)$ und $r_2 = (3,5)$: $(4 + 3, \max\{4,5\}) = (7,5)$.
 $h_l < h_r \Rightarrow$ Als nächstes l_1 und r_3 zusammenfügen:
- ▶ $l_1 = (4,4)$ und $r_3 = (4,3)$: $(4 + 4, \max\{4,3\}) = (8,4)$.
 $h_l > h_r \Rightarrow$ Als nächstes l_2 und r_3 zusammenfügen:
- ▶ $l_2 = (6,3)$ und $r_3 = (4,3)$: $(6 + 4, \max\{3,3\}) = (10,3)$.
 $h_l = h_r \Rightarrow l_3$ und r_4 , Ende der Liste R erreicht.

Abmessungsliste für den Knoten c : $\{(6,7), (7,5), (8,4), (10,3)\}$.

Entsprechende Fortführung liefert Listen für weitere Knoten:

d : $\{(9,7), (10,5), (13,4), (15,3)\}$, f : $\{(12,7), (13,5), (18,4), (20,3)\}$,

e : $\{(3,4)\}$, g : $\{(20,7), (18,8), (13,9), (12,11)\}$.

Zusammenfassung des *bottom-up* Laufs

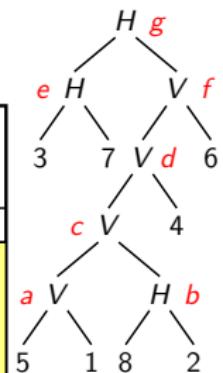
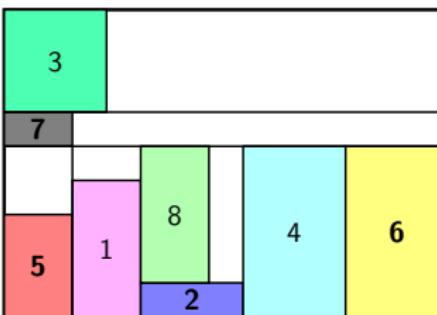
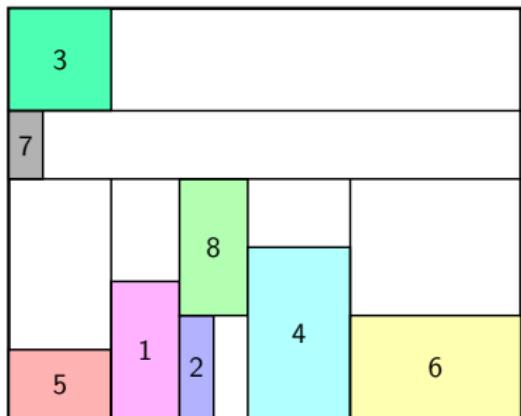
Knoten	Typ	Abmessungen
<i>a</i>	<i>V</i>	$L = \{(2,3), (3,2)\}, R = \{(2,4), (4,2)\},$ $D = \{(4,4), (6,3), (7,2)\}$
<i>b</i>	<i>H</i>	$L = \{(4,2), (2,4)\}, R = \{(3,1), (1,3)\},$ $D = \{(4,3), (3,5), (2,7)\}$
<i>c</i>	<i>V</i>	$L = \{(4,4), (6,3), (7,2)\}, R = \{(2,7), (3,5), (4,3)\},$ $D = \{(6,7), (7,5), (8,4), (10,3)\}$
<i>d</i>	<i>V</i>	$L = \{(6,7), (7,5), (8,4), (10,3)\}, R = \{(3,5), (5,3)\},$ $D = \{(9,7), (10,5), (13,4), (15,3)\}$
<i>f</i>	<i>V</i>	$L = \{(9,7), (10,5), (13,4), (15,3)\}, R = \{(3,5), (5,3)\},$ $D = \{(12,7), (13,5), (18,4), (20,3)\}$
<i>e</i>	<i>H</i>	$L = \{(3,3)\}, R = \{(2,1), (1,2)\}, D = \{(3,4)\}$
<i>g</i>	<i>H</i>	$L = \{(3,4)\}, R = \{(20,3), (18,4), (13,5), (12,7)\},$ $D = \{(20,7), (18,8), (13,9), (12,11)\}$

Bestimmung der Topzellen-Abmessungen

Minimale Fläche der Topzelle wird bei Abmessungen $13 \cdot 9 = 117$ erreicht. Erstellung des Floorplans durch einen *top-down* Lauf:

- ▶ Knoten g : $(13,9)$, zusammengesetzt aus $(3,4) \cong e$ und $(13,5) = f$
- ▶ Knoten e : $(3,4)$, zusammengesetzt aus $(3,3) \cong \text{Block } 3$ und $(2,1) \cong \text{Block } 7$
- ▶ Knoten f : $(13,5)$, zusammengesetzt aus $(10,5) \cong d$ und $(3,5) \cong \text{Block } 6$
- ▶ Knoten d : $(10,5)$, zusammengesetzt aus $(7,5) \cong c$ und $(3,5) \cong \text{Block } 4$
- ▶ Knoten c : $(7,5)$, zusammengesetzt aus $(4,4) \cong a$ und $(3,5) \cong b$
- ▶ Knoten a : $(4,4)$, zusammengesetzt aus $(2,3) \cong \text{Block } 5$ und $(2,4) \cong \text{Block } 1$
- ▶ Knoten b : $(3,5)$, zusammengesetzt aus $(2,4) \cong \text{Block } 8$ und $(3,1) \cong \text{Block } 2$

Floorplan vor und nach der Optimierung mit dem STOCKMEYER-Algorithmus



Fläche der Topzelle: vorher $15 \times 12 = 180$, nachher $13 \times 9 = 117$.

Offensichtlich kann der Floorplan weiter kompaktiert werden.

Ergänzende Materialien

Originalveröffentlichung zum STOCKMEYER-Algorithmus:

- ▶ L. STOCKMEYER, *Optimal Orientations of Cells in Slicing Floorplan Designs*, Information and Control, Vol. 57, Issues 2–3, pp. 91–101, May/June 1983, Dateiname auf dem Handout-Server: Zusatzmaterial/eda_v9_stockmeyer.pdf

Zusammenfassung

- ▶ Floorplan-Darstellung
- ▶ Floorplan-Sizing-Algorithmus nach STOCKMEYER

Rechnergestützer Entwurf digitaler Systeme: Vorlesungs-Gesamtübersicht

1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme: Gliederung der 10. Vorlesung

3. Partitionierung und Floorplanning

3.1. Partitionierung

- 3.1.1. Grundlagen
- 3.1.2. Clustering
- 3.1.3. RAJARAMAN-WONG-Algorithmus
- 3.1.4. KERNIGHAN-LIN-Algorithmus

3.2. Floorplanning

- 3.2.1. Definitionen und Datenstrukturen
- 3.2.2. Floorplan-Sizing-Algorithmus
- 3.2.3. Lineare Optimierung
- 3.2.4. Pinzuordnung

4. Platzierung und Verdrahtung

4.1. Platzierung

- 4.1.1. Min-Cut-Platzierung
- 4.1.2. Kräfteplatzierung
- 4.1.3. Weitere Verfahren

4.2. Verdrahtung

Mathematischer Hintergrund

Floorplanning durch ganzzahlige lineare Optimierung, auch als *Integer Linear Programming* (ILP) bzw. Floorplanning mit Gleichungssystemen bekannt. Grundprinzip:

- ▶ Floorplan-Kenngrößen (Abmessungen der Blöcke, Abmessungen der Topzelle) stellen Variable und Konstanten eines linearen (Un)Gleichungssystems dar.
- ▶ Die Lösung des Gleichungssystems entspricht einem Floorplan entsprechend einem vorgegebenem Optimierungsziel (meistens minimale Fläche der Topzelle)
- 😊 Eine exakte Lösung entspricht tatsächlich einem globalen Optimum, da das Problem analytisch behandelt wird
- 😢 Bei realistischen Problemgrößen ist die Anzahl der Variablen/Gleichungen enorm hoch, so dass das Finden einer exakten Lösung in akzeptabler Zeit nicht möglich ist
- ➡ Heuristische näherungsweise Lösungen

Randbedingungen

Optimierungsziel (*objective function*): $A = w \cdot h \rightarrow \min$, linearisiert durch Festhalten von w und Minimierung von h .

Überlappungsfreiheit (*non-overlap constraints*) für ein Blockpaar (i, j) mit Koordinaten der linken unteren Ecken (x_i, y_i) sowie (x_j, y_j) :

$$x_i + w_i \leq x_j \quad i \text{ liegt links von } j$$

$$x_j + w_j \leq x_i \quad i \text{ liegt rechts von } j$$

$$y_i + h_i \leq y_j \quad i \text{ liegt unterhalb von } j$$

$$y_j + h_j \leq y_i \quad i \text{ liegt oberhalb von } j$$

Zur Sicherstellung der Überlappungsfreiheit muss mindestens eine der Ungleichungen erfüllt sein.

Randbedingungen (fortgesetzt)

Verschiedene Zusatzeinschränkungen (*variable type constraints*):

Zum Beispiel $x_i \geq 0$, $x_j \geq 0$, $y_i \geq 0$, $y_j \geq 0$ usw.

Topzellenabmessungen: Vorgabe der maximalen Breite bzw. Höhe

$$x_i + w_i \leq W$$

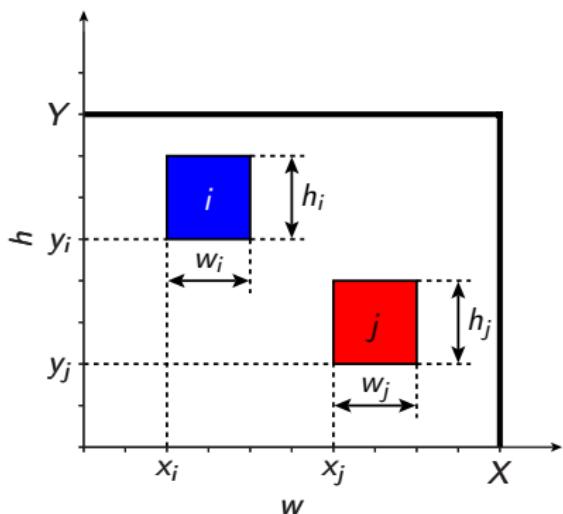
$$x_j + w_j \leq W$$

$$y_i + h_i \leq H$$

$$y_j + h_j \leq H$$

Vereinfachung der Darstellung: Einführung von Variablen $x_{ij} \in \{0,1\}$ und $y_{ij} \in \{0,1\}$, die für die Auswahl nur einer aus vier Bedingungen sorgen. Zusätzlich werden die oberen Schranken für Höhe und Breite der Topzelle definiert: $H = \sum h_i$ bzw. $W = \sum w_i$.

Formulierung des ILP-Problems am Beispiel von zwei Blöcken



$$\begin{aligned} x_i + w_i &\leq x_j + W \cdot (x_{ij} + y_{ij}) \\ x_j + w_j &\leq x_i + W \cdot (1 - x_{ij} + y_{ij}) \\ y_i + h_i &\leq y_j + H \cdot (1 + x_{ij} - y_{ij}) \\ y_j + h_j &\leq y_i + H \cdot (2 - x_{ij} - y_{ij}) \end{aligned}$$

Mit $x_{ij} = 0$ und $y_{ij} = 1$ wird die Ungleichung $y_i + h_i \leq y_j$ als einzige ausgewählt.

Obere Schranken werden als $H = h_i + h_j$ sowie $W = w_i + w_j$ berechnet. Bei Vorgabe der maximalen Höhe Y oder Breite X müssen zusätzliche Ungleichungen gelten: $x_i + w_i \leq X$ und $x_j + w_j \leq X$ sowie $y_i + h_i \leq Y$ und $y_j + h_j \leq Y$.

Beispiel zur Aufstellung eines Ungleichungssystems

4 Blöcke $\{(4,5), (3,7), (6,4), (7,7)\}$. Zur Vereinfachung seien Drehungen nicht zugelassen, für die Topzelle gelte $X = Y$.

Unbekannte Variable (gesuchte Größen): $x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4$

Hilfsvariable: $x_{12}, x_{13}, x_{14}, x_{23}, x_{24}, x_{34}, y_{12}, y_{13}, y_{14}, y_{23}, y_{24}, y_{34}$

Obere Schranken:

$$W = \sum w_i = 4 + 3 + 6 + 7 = 20$$

$$H = \sum h_i = 5 + 7 + 4 + 7 = 23$$

Unbekannte Größen sind so zu bestimmen, dass Y minimal ist:
 $(y^* \rightarrow \min)$.

Ungleichungen zur Überlappungsfreiheit

$$x_1 + w_1 \leq x_2 + 20 \cdot (x_{12} + y_{12})$$

$$x_1 - w_2 \geq x_2 - 20 \cdot (1 - x_{12} + y_{12})$$

$$y_1 + h_1 \leq y_2 + 23 \cdot (1 + x_{12} - y_{12})$$

$$y_1 - h_2 \geq y_2 - 23 \cdot (2 - x_{12} - y_{12})$$

$$x_1 + w_1 \leq x_4 + 20 \cdot (x_{14} + y_{14})$$

$$x_1 - w_4 \geq x_4 - 20 \cdot (1 - x_{14} + y_{14})$$

$$y_1 + h_1 \leq y_4 + 23 \cdot (1 + x_{14} - y_{14})$$

$$y_1 - h_4 \geq y_4 - 23 \cdot (2 - x_{14} - y_{14})$$

$$x_2 + w_2 \leq x_4 + 20 \cdot (x_{24} + y_{24})$$

$$x_2 - w_4 \geq x_4 - 20 \cdot (1 - x_{24} + y_{24})$$

$$y_2 + h_2 \leq y_4 + 23 \cdot (1 + x_{24} - y_{24})$$

$$y_2 - h_4 \geq y_4 - 23 \cdot (2 - x_{24} - y_{24})$$

$$x_1 + w_1 \leq x_3 + 20 \cdot (x_{13} + y_{13})$$

$$x_1 - w_3 \geq x_3 - 20 \cdot (1 - x_{13} + y_{13})$$

$$y_1 + h_1 \leq y_3 + 23 \cdot (1 + x_{13} - y_{13})$$

$$y_1 - h_3 \geq y_3 - 23 \cdot (2 - x_{13} - y_{13})$$

$$x_2 + w_2 \leq x_3 + 20 \cdot (x_{23} + y_{23})$$

$$x_2 - w_3 \geq x_3 - 20 \cdot (1 - x_{23} + y_{23})$$

$$y_2 + h_2 \leq y_3 + 23 \cdot (1 + x_{23} - y_{23})$$

$$y_2 - h_3 \geq y_3 - 23 \cdot (2 - x_{23} - y_{23})$$

$$x_3 + w_3 \leq x_4 + 20 \cdot (x_{34} + y_{34})$$

$$x_3 - w_4 \geq x_4 - 20 \cdot (1 - x_{34} + y_{34})$$

$$y_3 + h_3 \leq y_4 + 23 \cdot (1 + x_{34} - y_{34})$$

$$y_3 - h_4 \geq y_4 - 23 \cdot (2 - x_{34} - y_{34})$$

Ungleichungen für Zusatzeinschränkungen

Alle Koordinaten sind positiv, Hilfsvariablen sind binär:

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, \quad y_1 \geq 0, y_2 \geq 0, y_3 \geq 0, y_4 \geq 0,$$

$$x_{12}, x_{13}, x_{14}, x_{23}, x_{24}, x_{34} \in \{0,1\} \quad y_{12}, y_{13}, y_{14}, y_{23}, y_{24}, y_{34} \in \{0,1\}$$

Ungleichungen für Topzellenabmessungen:

$$x_1 + w_1 \leq y^*, x_2 + w_2 \leq y^*, \quad x_3 + w_3 \leq y^*, x_4 + w_4 \leq y^*,$$

$$y_1 + h_1 \leq y^*, y_2 + h_2 \leq y^*, \quad y_3 + h_3 \leq y^*, y_4 + h_4 \leq y^*$$

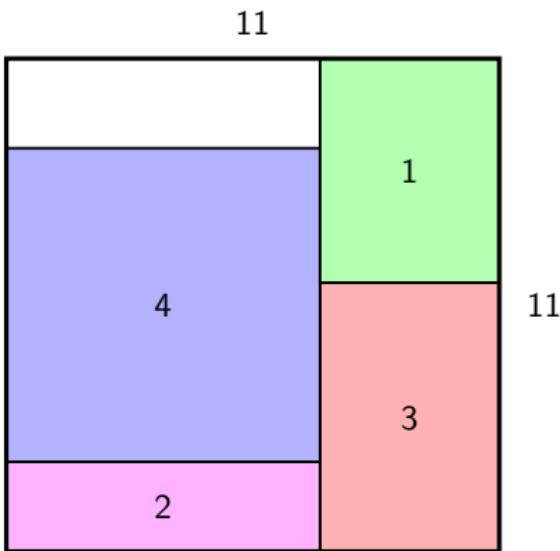
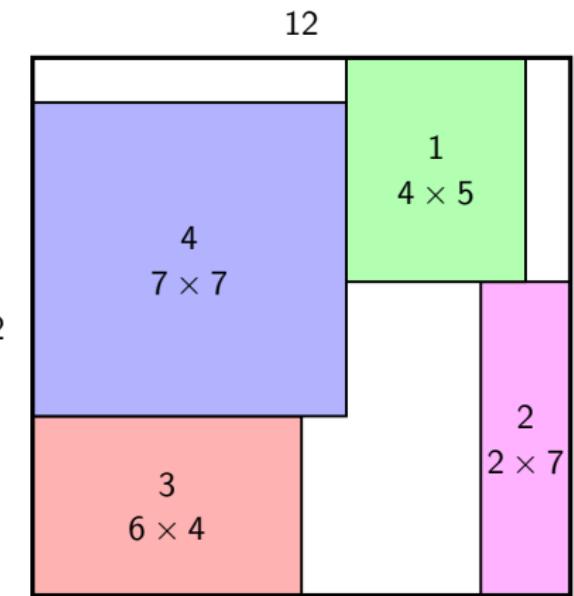
Analytische Lösung liefert $y^* = 12$ sowie

$$(x_1, y_1) = (7, 7), (x_2, y_2) = (9, 0), \quad (x_3, y_3) = (0, 0), (x_4, y_4) = (0, 4)$$

$$(x_{12}, y_{12}) = (1, 1), (x_{13}, y_{13}) = (1, 1) \quad (x_{14}, y_{14}) = (1, 0)$$

$$(x_{23}, y_{23}) = (1, 0), (x_{24}, y_{24}) = (1, 0) \quad (x_{34}, y_{34}) = (0, 1)$$

Graphische Darstellung der Lösung (links)



Dem rechten Floorplan entspricht die Lösung eines erweiterten Ungleichungssystems mit erlaubten Drehungen der Blöcke.

Drehung von Blöcken

- ▶ Einführung der Zusatzvariablen $z_x \in \{0,1\}$, wobei $z_x = 1$ einer Drehung entspricht (bei $z_x = 0$ sind die Ungleichungen wie im „drehungslosen“ Fall).
- ▶ W und H wird durch $M = \max\{W,H\}$ ersetzt, da bei Drehungen die „längsten Seiten“ die obere Schranke bestimmen.

Modifiziertes ILP-System aus dem letzten Beispiel (unvollständig, $z_x = 1$ „vertauscht“ Breite und Höhe):

$$x_1 + z_1 h_1 + (1 - z_1) \cdot w_1 \leq x_2 + 23 \cdot (x_{12} + y_{12})$$

$$x_1 - z_2 h_2 - (1 - z_2) \cdot w_2 \geq x_2 - 23 \cdot (1 - x_{12} + y_{12})$$

$$y_1 + z_1 w_1 + (1 - z_1) \cdot h_1 \leq y_2 + 23 \cdot (1 + x_{12} - y_{12})$$

$$y_1 - z_2 w_2 - (1 - z_2) \cdot h_2 \geq y_2 - 23 \cdot (2 - x_{12} - y_{12})$$

... ...

Flexible Breiten und Höhen

Einführung der Breiten von „flexiblen“ Blöcken als Zusatzvariable w_x (eine pro Block), lineare Approximierung der Höhenfunktion h_x für diese Blöcke ($A_x \cong$ Fläche des Blocks x):

$$h_x = \frac{A_x}{w_{x,\max}} + (w_{x,\max} - w_x) \cdot \frac{A_x}{w_{i,\max}^2} \quad (\text{TAYLOR-Reihe})$$

$$w_x \cdot h_x \geq A_x, \quad l_x \leq \frac{w_x}{h_x} \leq u_x$$

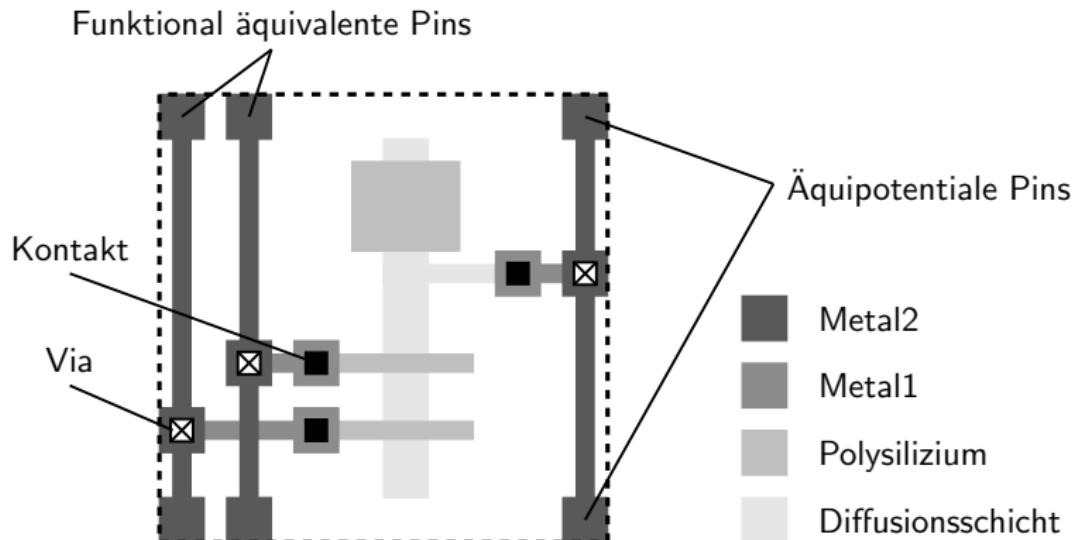
$$w_{x,\min} = \sqrt{A_x \cdot l_x}, \quad w_{x,\max} = \sqrt{A_x \cdot u_x}$$

Aus der Vorgabe der zulässigen Seitenverhältnisse (l_x als Minimalwert und u_x als Maximalwert) werden Ungleichungen für Breiten und Höhen der Blöcke aufgestellt: $w_{x,\min} \leq w_x \leq w_{x,\max}$ sowie $h_{x,\min} \leq h_x \leq h_{x,\max}$.

Anmerkung: Bei flexiblen Blöcken entfällt die Drehvariable z_x .

Problembeschreibung

Die durch die Partitionierung entstandenen Blöcke sind durch Netze verbunden. Netze sind an Außenanschlüsse der Blöcke (Pins) gekoppelt. Eine Zuordnung von Pins zu den Netzen, die anschließende Verdrahtung (innerhalb eines Blockes aber auch zwischen den Blöcken) vereinfacht, wird als **Pinzuordnung** (*pin assignment*) bezeichnet.



Einteilung der Anschlüsse in Gruppen

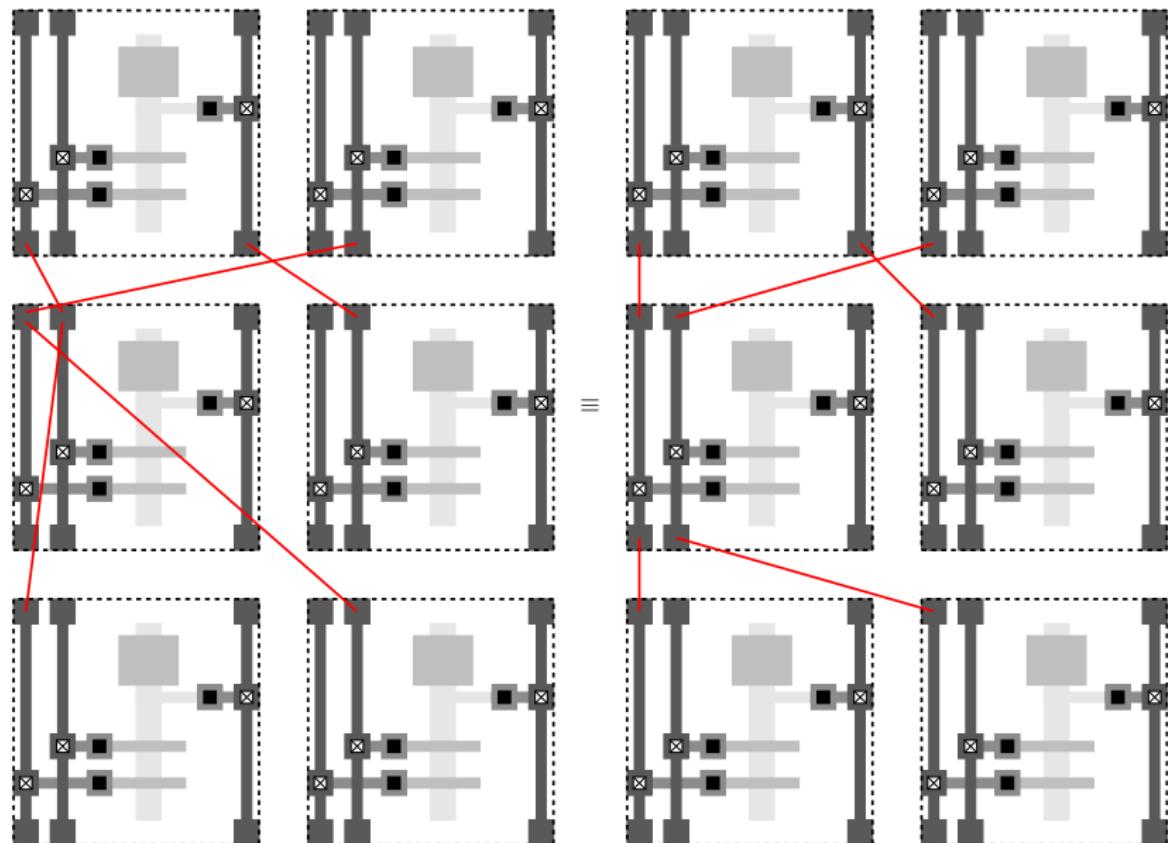
Am Beispiel eines NAND-Gatters sind verschiedene Gruppen von Pins erkennbar:

Funktional äquivalente Pins (*functionally equivalent pins*): Pins, die gegeneinander ausgetauscht werden können, ohne dass die logische Funktion der Schaltung davon beeinträchtigt wird.

Äquipotentielle Pins (*equipotential pins*): Pins, die zellen- bzw. blockintern miteinander verbunden und daher immer dem gleichen Netz zugeordnet sind.

Pinzuordnung ist eine Optimierungsaufgabe, bei der Netzanschlüsse (*net terminals*) auf äquipotentielle bzw. funktional äquivalente Pins verteilt werden. Dabei darf ein Netzanschluss einem beliebigen äquipotentialem Pin innerhalb der Menge funktional äquivalenter Pins zugeordnet werden. Optimierungsziel ist meistens die Verdrahtbarkeit (Verminderung der Überbelegung).

Einfluss der Pinzuordnung auf Verdrahtbarkeit



Formalisierte Problemstellung

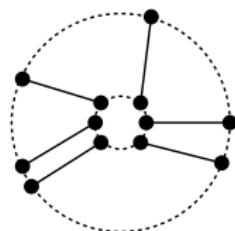
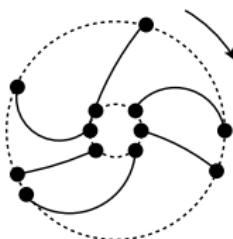
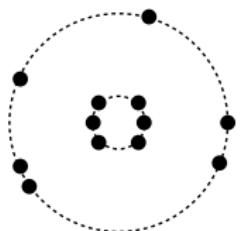
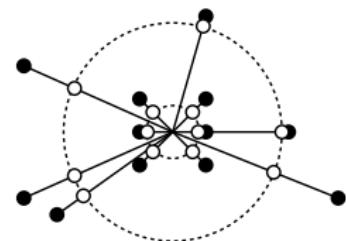
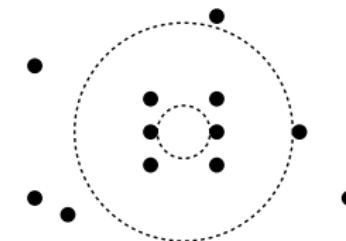
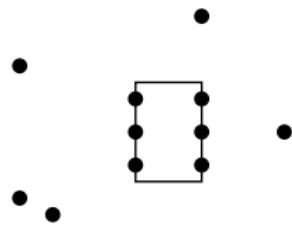
Sei $P = \{P_1, P_2, \dots, P_m\}$ eine Menge von Pins und $T = \{T_1, T_2, \dots, T_n\}$ eine Menge von Netzanschlüssen mit $m > n$. Die Zuordnung eines Netzanschlusses T_i zu einem Pin P_i ist durch die Netzliste für alle $i = 1, 2, \dots, n$ vorgegeben. Sei ε_{P_i} eine Menge von äquipotentialen Pins, die funktional äquivalent zu P_i sind.

Gesucht ist die Zuordnung $T_x \mapsto \varepsilon_{P_x}$ für alle $x = 1, 2, \dots, n$, die zu einer verbesserten (optimalen) Verdrahtbarkeit führt.

Optimale Verdrahtbarkeit:

- ▶ Keine Bereiche mit Überbelegung (*routing congestion*)
- ▶ Geringe Verdrahtungsdichte pro Verdrahtungskanal
(insbesondere bei Gate Arrays)

Pinzuordnung mit konzentrischen Kreisen



Pinzuordnung mit konzentrischen Kreisen: Ablauf

1. Gegeben: Festgelegte Anschlusspunkte außerhalb des Blocks und variable Pins des Blocks.
2. Zwei Kreise festlegen: Der innere innerhalb der (noch nicht zugeordneten) Pins des Blocks, der äußere gerade noch innerhalb der äußeren Netzanschlüsse.
3. Durch jeden Anschluss eine Gerade aus dem Kreiszentrum ziehen, Schnittlinien mit den Kreisen feststellen.
4. Pins und Netzanschlüsse auf die Kreise verlegen.
5. Eine willkürliche Verbindung zwischen zwei Punkten verschiedener Kreise herstellen, anschließend alle anderen Punkte sequentiell (kreuzungsfrei) verbinden.
6. Durch Rotation eine Zuordnung mit minimaler EUKLIDScher Gesamtverbindungsstrecke ermitteln.

4. Platzierung und Verdrahtung

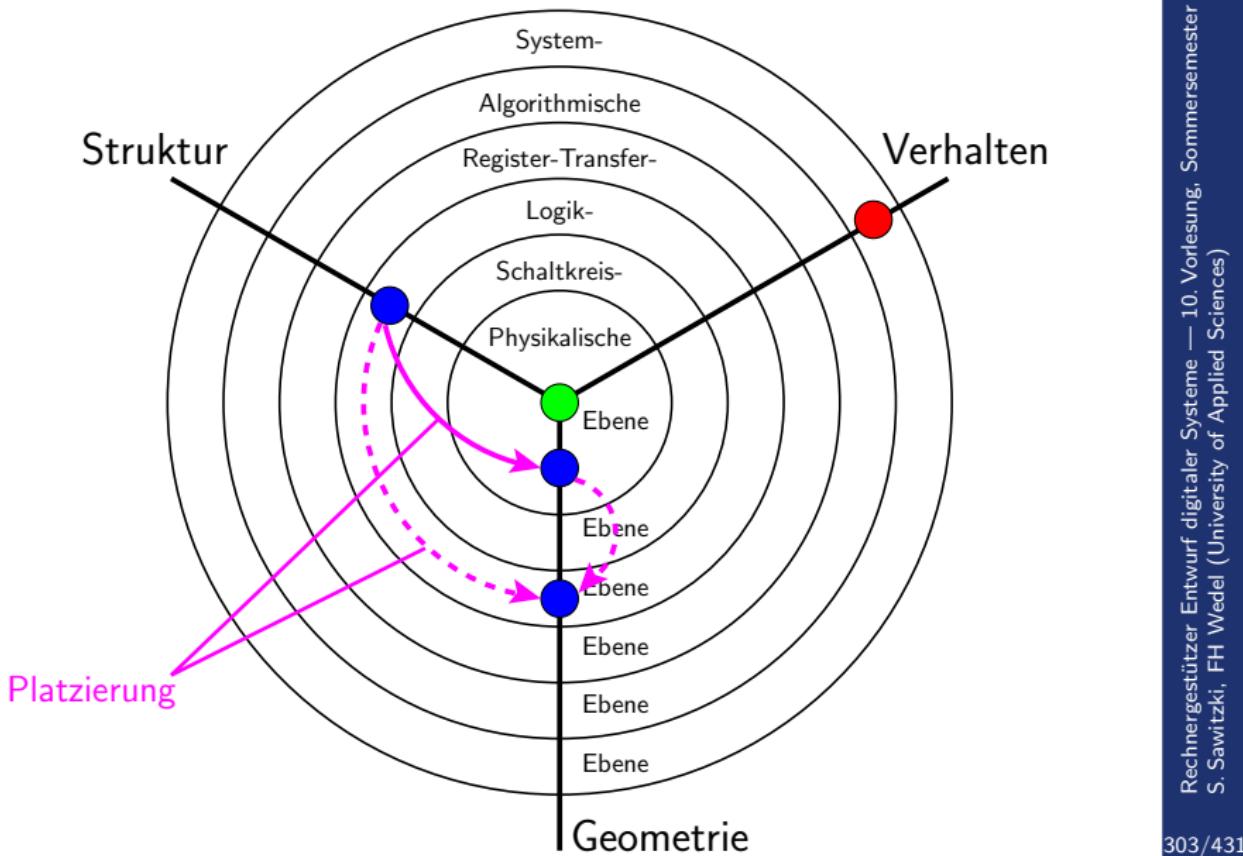
Aufgaben von Platzierung und Verdrahtung

Durch Partitionierung und Floorplanning ist die Gesamtschaltung in kleinere „handlichere“ Teilblöcke zerlegt, deren Position innerhalb der Topzelle bestimmt ist. Die Anordnung der Zellen innerhalb von Blöcken, aber auch der Verlauf von Verbindungsleitungen sind noch nicht festgelegt. Das erfolgt innerhalb der Platzierungs- und Verdrahtungsschritte:

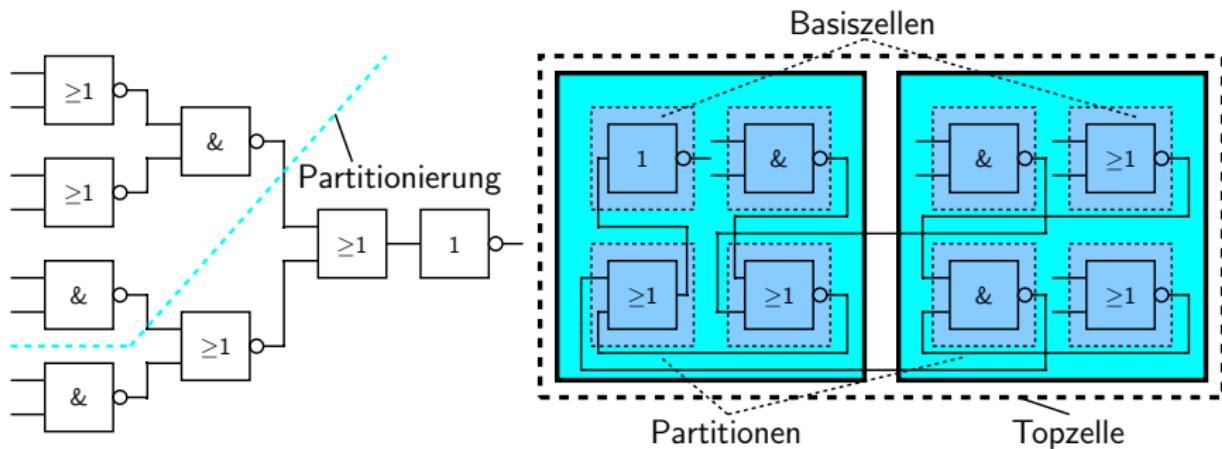
- ▶ Platzierung: Festlegung der Position einzelner Schaltungselemente auf der verfügbaren Layoutfläche
- ▶ Verdrahtung: Festlegung der Zuordnung der Verbindungen innerhalb der Netzliste zu den verfügbaren Leiterzugsegmenten und Verdrahtungsebenen

Beide Aufgaben müssen eine Reihe von Zusatzbedingungen (z. B. Überlappungsfreiheit, eingeschränkte Anzahl der Ressourcen) berücksichtigen. Bei der Durchführung sind vordefinierte Optimierungsziele zu berücksichtigen.

Einordnung in den Entwurfsprozess



Veranschaulichung des Platzierungsschrittes



- ▶ Bei einfacheren Netzlisten sind vorhergehende Partitionierung und Floorplanning nicht zwingend notwendig.
- ▶ Die Zellen sind symbolisch dargestellt (Rechtecke), die Funktion ist für die Platzierung ohne Belang ➡ Geometriesicht

Hauptoptimierungsziele von Platzierungsalgorithmen

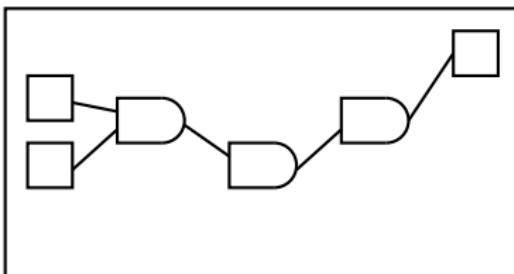
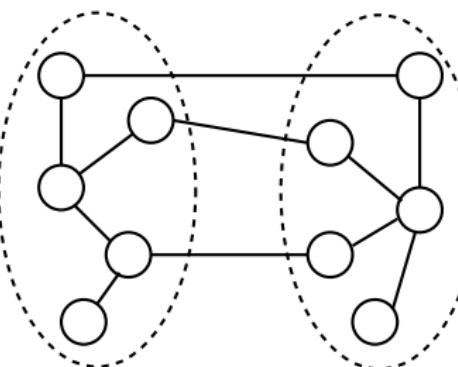
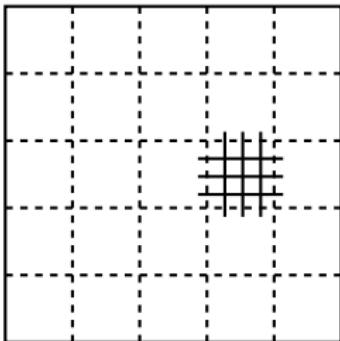
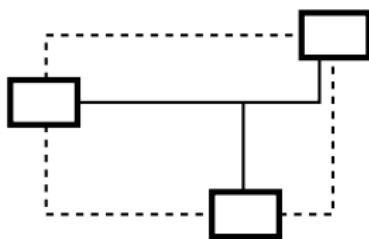
- ▶ Gesamtverbindungsstrecke
- ▶ Kritischer Pfad bzw. allgemeine Verzögerungszeitvorgaben für Netze
- ▶ Anzahl der von einer Schnittlinie geschnittenen Netze
- ▶ Erwartete lokale Verdrahtungsdichte

Weitere Optimierungsziele:

- ▶ Kurze Verbindungswege zu Außenanschlüssen, effiziente Anordnung der I/O-Zellen
- ▶ Reduzierung/Vermeidung von *Hot spots*, d. h. Gleichverteilung der Temperatur über die Layoutfläche

4.1. Platzierung

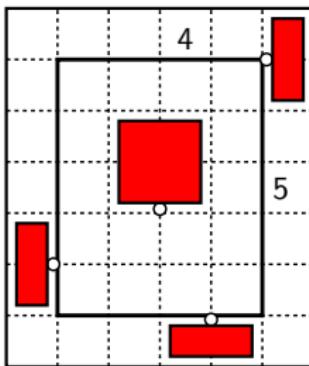
Veranschaulichung der Optimierungsziele bei Platzierung



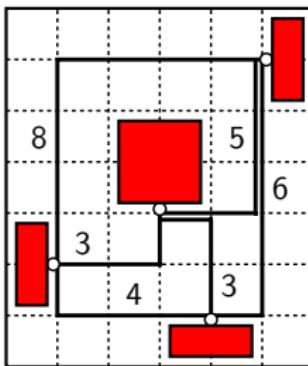
Abschätzung der Gesamtverbindungsstrecke

- ▶ Schlüsselschritt bei dem entsprechenden Optimierungsziel
- ▶ Anforderungen:
 - ▶ schnell ermittelbar (Rechenzeit)
 - ▶ gleicher Schätzfehler bei unterschiedlichen Netzen
(Netzunabhängigkeit)
- ▶ Metriken:
 - ▶ Bei Zweipunkt-Netzen: Manhattan-Abstand $x_{ij} + y_{ij}$, wobei x_{ij} bzw. y_{ij} horizontaler bzw. vertikaler Abstand zwischen den Zellen i und j ist
 - ▶ Bei Mehrpunkt-Netzen: viele verschiedene Möglichkeiten, häufig halber Netzmfang (halbe *bounding box*, auch halber Umfang des umschließenden Rechtecks genannt)

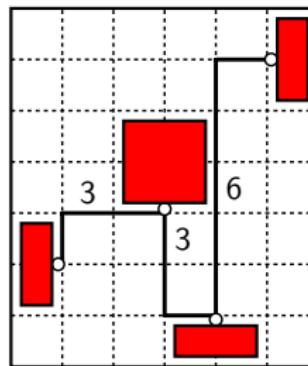
Verschiedene Metriken



Kosten: 9



Kosten: 14,5



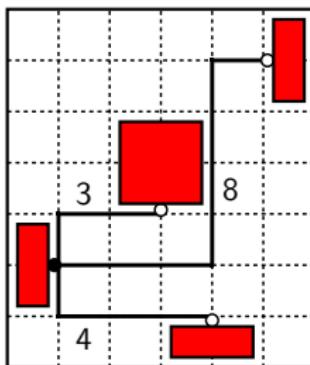
Kosten: 12

- ▶ Halber Netzumfang: $L = a + b = 4 + 5 = 9$
- ▶ Kompletter Graph (p ist Anzahl der Pins):

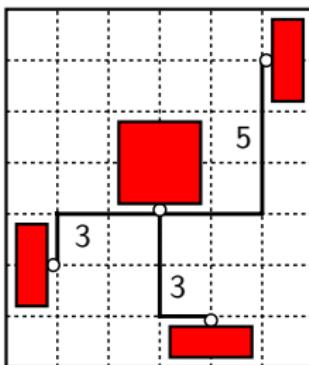
$$L = \frac{2}{p} \cdot \sum_{\forall Kanten \in \text{Netz}} \text{Kantenlängen} = 14,5$$

- ▶ Minimale Kette: $L = \sum a_n = 12$, wobei a_n immer zwei benachbarte Knoten auf dem kürzesten (rektilinearen) Weg verbindet

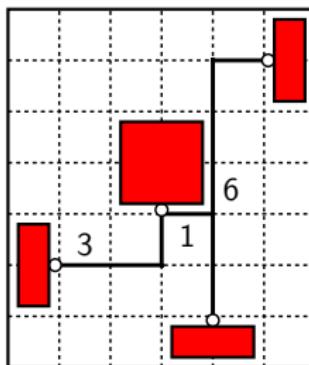
Verschiedene Metriken (fortgesetzt)



Kosten: 15



Kosten: 11

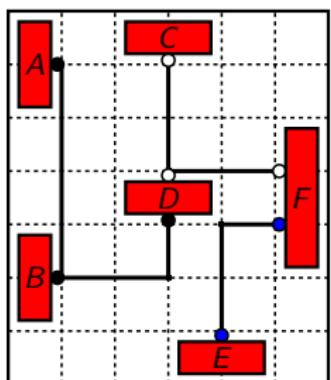


Kosten: 10

- ▶ Quelle-Senken-Verbindung: Annahme der Sterntopologie
- ▶ Minimaler rektilinearer Spannbaum: Summe der Zweipunkte-Verbindungen minimaler Länge
- ▶ STEINER-Baum: Spannbaum mit beliebiger Anzahl weiterer Punkte, Erzeugungsproblem ist NP-schwer.

Verfeinerung der Abschätzung durch Einführung der Netzgewichte

Beispielsweise kritische vs. unkritische Netze:



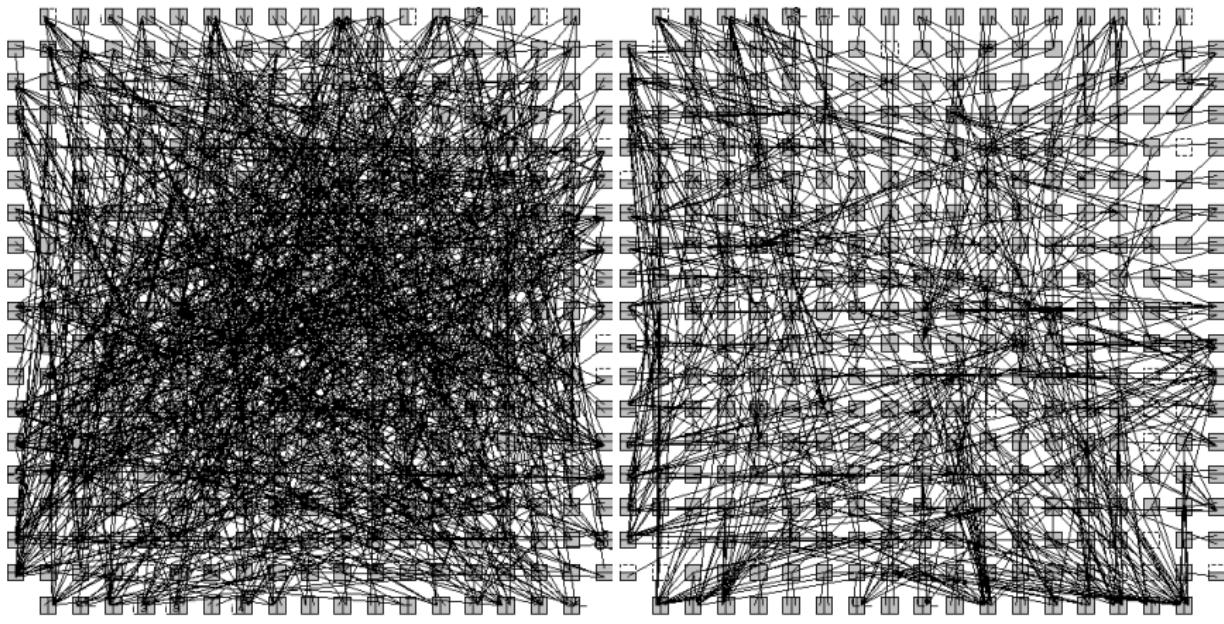
Netze	Gewichte
$N_1 = (A, B, D_1)$	$w_1 = 2$
$N_2 = (C, D_2, F_1)$	$w_2 = 4$
$N_3 = (F_2, E)$	$w_3 = 1$
$L(P) = 2 \cdot 7 + 4 \cdot 4 + 1 \cdot 3 = 33$	

$$L(P) = \sum_{n \in N} w_n \cdot d_n,$$

mit d_n als geschätzter Länge des Netzes n , w_n als Gewicht und N als Menge aller Netze.

4.1. Platzierung

Gleiche Netzliste mit zwei unterschiedlichen Platzierungen



Große Einteilung der Platzierungsalgorithmen

Konstruktiv: Ausgehend von einer leeren Layout-Fläche werden die Blöcke in einer vom Algorithmus vorgegebenen Reihenfolge platziert, bis jeder Block eine physikalische Position zugewiesen bekommen hat.

Iterativ: Ausgehend von einer Initialplatzierung versucht der Algorithmus schrittweise Verbesserungen herbeizuführen, bis die Qualität der Platzierung als akzeptabel angesehen wird oder die vorgegebene Laufzeit verstrichen ist.

Analytisch: Numerische Ermittlung eines Minimums der Kostenfunktion, das in der Regel einer „illegalen“ Platzierung entspricht, die mit Hilfe von Heuristiken „legalisiert“ wird.

Mischformen: Kombination mehrerer Vorgehensweisen, z. B. am Anfang konstruktiv und ab einer bestimmten Blockmenge iterativ.

Schnittbezogene Bewertungskriterien für eine Platzierung P

- ▶ Anzahl der Netze, die von einer vertikalen Schnittlinie x_i geschnitten werden $\Psi_P(x_i)$
- ▶ Anzahl der Netze, die von einer horizontalen Schnittlinie y_j geschnitten werden $\Psi_P(y_j)$
- ▶ Maximalwerte über alle Schnittlinien legen die Mindestanzahl der benötigten Leitungssegmente in einem Kanal an der Stelle des entsprechenden Schnittes fest:

$$x(P) = \max_i (\Psi_P(x_i)), \quad y(P) = \max_j (\Psi_P(y_j))$$

- ▶ Gesamtverbindungsänge

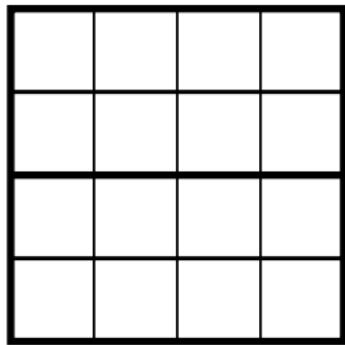
$$L(P) = \sum_i \Psi_P(x_i) + \sum_j \Psi_P(y_j)$$

Konstruktive Platzierung am Beispiel des Min-Cut-Verfahrens

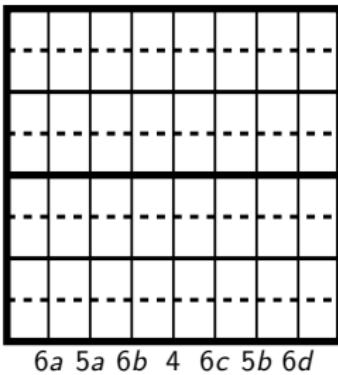
1. Aufteilung der Layout-Fläche in zwei Teilflächen (senkrecht oder waagerecht).
2. Partitionierung der Netzliste mit bekannten Algorithmen (z. B. KL oder FM) und Zuordnung der Partitionen zu den Teilflächen.
3. Rekursive Aufteilung der Teilflächen und weitere Partitionierung und Zuordnung. Schnittrichtung ist dabei alternierend (verschiedene Schnittwechselmuster möglich).
4. ENDE, wenn jede Teilfläche genau eine Zelle enthält, sonst zurück zum Schritt 3.

Suboptimal, da jeweils nur eine Schnittlinie betrachtet wird und frühere Schnitte nicht revidiert werden. Aus Laufzeitgründen ist keine globale Optimierung von $x(P)$, $y(P)$ und $L(P)$ möglich.

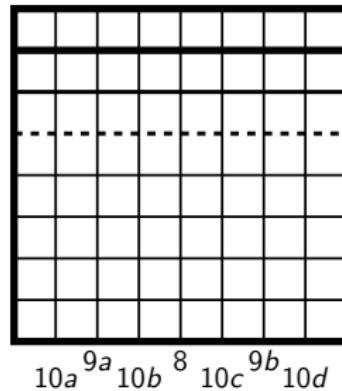
Verschiedene Schnittmuster



Quadratur



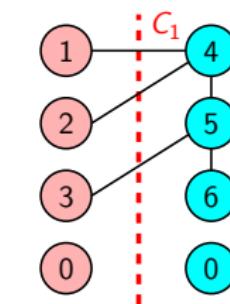
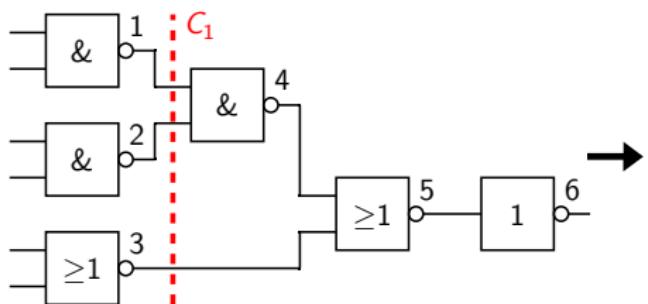
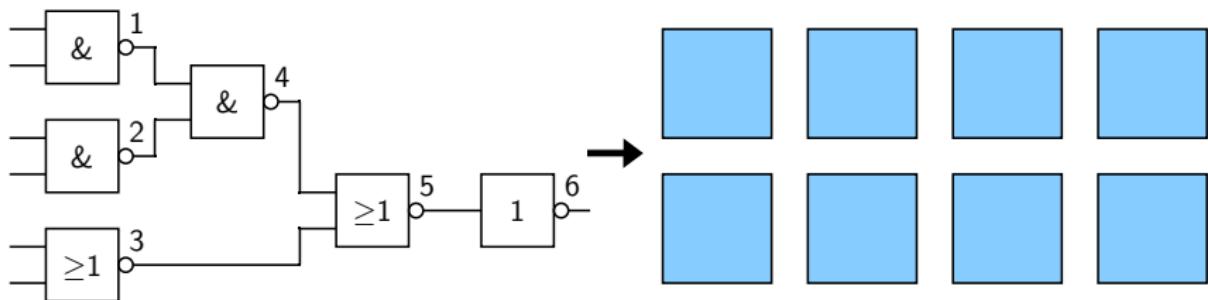
Halbierung



Reihen-Halbierung

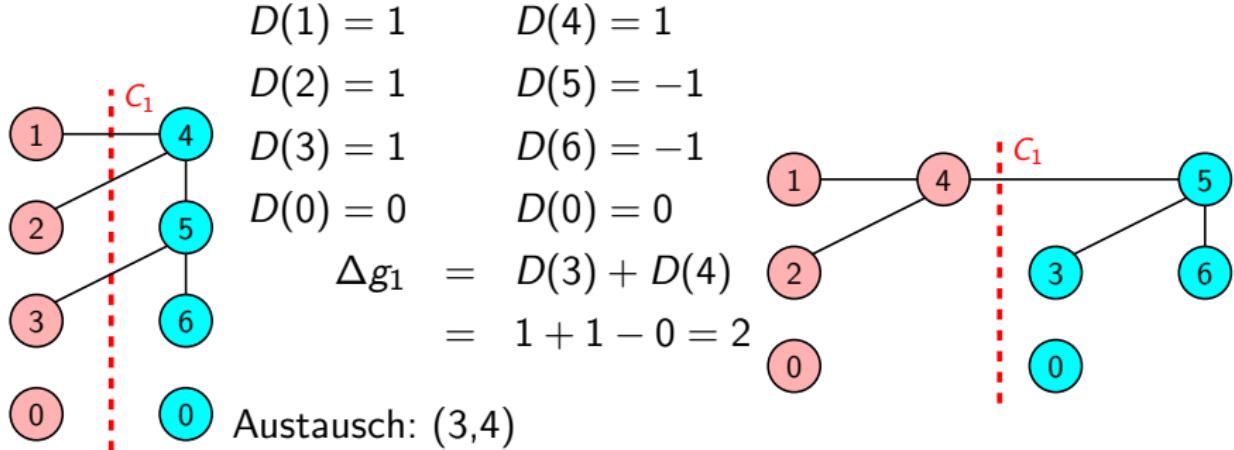
Sequentielle Zielfunktion, da optimaler Schnitt jeweils nur bezogen auf eine Schnittlinie ermittelt wird und die Gesamtlösung aus einer Folge von lokalen Optimierungsschritten entsteht.

Beispiel (1)



Beispiel (2)

Schnittkosten: 3,

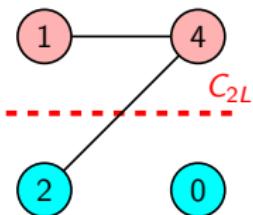


$$G_1 = \Delta g_1 = 2$$

➡ Schnittkosten: 1

Beispiel (3)

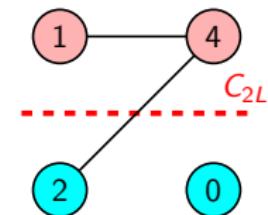
Schnittkosten: 1,



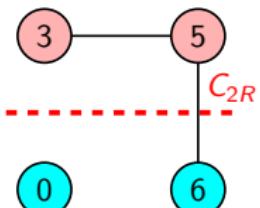
$$D(1) = -1 \quad D(2) = 1$$

$$D(4) = 0 \quad D(0) = 0$$

$$\text{Kein } \Delta g > 0$$

Kein Austausch \Rightarrow Schnittkosten: 1

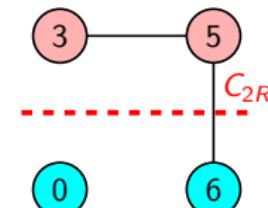
Schnittkosten: 1,



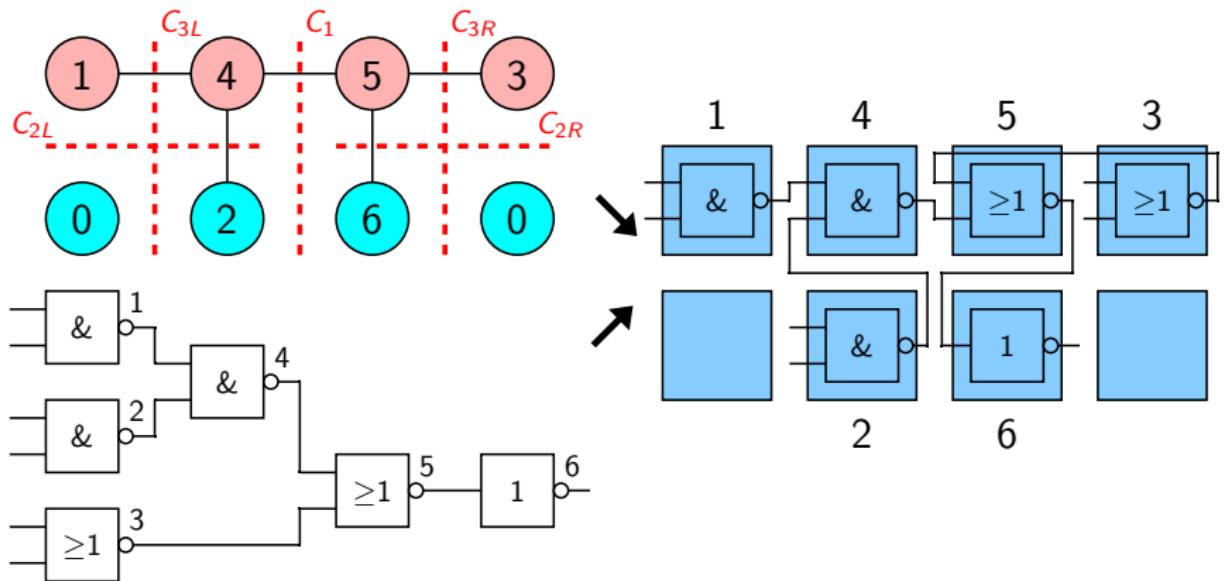
$$D(3) = -1 \quad D(6) = 1$$

$$D(5) = 0 \quad D(0) = 0$$

$$\text{Kein } \Delta g > 0$$

Kein Austausch \Rightarrow Schnittkosten: 1

Das Ergebnis



Anschlussfestlegung

Im vorhergehenden Beispiel wurde beim letzten Schnitt die Technik der Anschlussfestlegung (*terminal propagation*) angewandt:

- ▶ Berücksichtigt man die Lage von Anschlusspunkten bzw. von externen Anschlässen nicht, so können ungünstige Platzierungsentscheidungen getroffen werden. Daher:
- ▶ Beim Erzeugen eines neuen Schnittes wird ein eintretendes bzw. austretendes Netz mit einem „fiktiven Terminal“ fixiert.
- ▶ Liegt das fiktive Terminal in der Nähe der aktuellen Schnittlinie, so erfolgt die Partitionierung wie üblich
- ▶ Liegt das fiktive Terminal nicht in der Nähe der aktuellen Schnittlinie, so wird an seiner Stelle eine Dummy-Zelle erzeugt, die die Kostenfunktion so beeinflusst, dass die mit ihr verbundenen Zellen in die Nähe kommen. Begriff „Nähe“ kann quantifiziert werden.

Zusammenfassung

- ▶ Fortsetzung Floorplanning:
 - ▶ Analytisches Floorplanning mittels linearer Optimierung
 - ▶ Pinzuordnungsproblem
- ▶ Platzierung:
 - ▶ Bewertung und Kostenfunktionen
 - ▶ Grobe Einteilung der Algorithmen
 - ▶ Min-Cut

Rechnergestützer Entwurf digitaler Systeme:

Vorlesungs-Gesamtübersicht

1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme:

Gliederung der 11. Vorlesung

4. Platzierung und Verdrahtung

4.1. Platzierung

4.1.1. Min-Cut-Platzierung

4.1.2. Kräfteplatzierung

4.1.3. Weitere Verfahren

4.2. Verdrahtung

4.2.1. Globalverdrahtung

4.2.2. Feinverdrahtung

4.2.3. Flächenverdrahtung

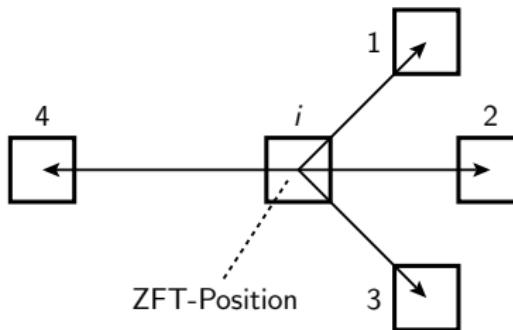
Physikalischer Hintergrund

- ▶ Modell eines mechanischen Systems: über Federn verbundene Körper, *force directed placement*
- ▶ Körper üben Kraft $F = c \cdot d$ aufeinander aus (c ist Federsteife, d ist Entfernung)
- ▶ Wenn alle Körper sich frei bewegen dürfen, strebt das Gesamtsystem einen Zustand des Kräftegleichgewichts an.
- ▶ Interpretiert man Körper als Zellen und Kraft als Darstellung der Verdrahtungskosten, so kann man das Prinzip auf Verdrahtungsprobleme übertragen, z. B. für zwei Zellen a und b : $\vec{F} = w_{ab} \cdot \vec{d}_{ab}$, bzw. $F = w_{ab} \cdot d_{ab}$ mit d_{ab} als Entfernung zwischen den Zellen a und b , $d_{ab} = \sqrt{(\Delta x_{ab})^2 + (\Delta y_{ab})^2}$. Für eine Zelle i , die mit mehreren Zellen $1, \dots, n$ verbunden ist, gilt

$$\vec{F} = \sum_{j=1}^n (w_{ij} \cdot \vec{d}_{ij}) \text{ mit } w_{ij} \text{ als Gewichtung der Verbindung}$$

Zero Force Target

Um das Kräftegleichgewicht zu erreichen, strebt jede Zelle eine ZFT-Position (*zero force target*) an:



$$\vec{F}_i = w_{i1} \cdot \vec{d}_{i1} + w_{i2} \cdot \vec{d}_{i2} + w_{i3} \cdot \vec{d}_{i3} + w_{i4} \cdot \vec{d}_{i4} \rightarrow \min$$

- ▶ Erweiterung um Abstoßungsgleichungen (Vermeidung von Überlappungen) sowie Gleichungen für unbewegliche Zellen (z. B. vordefinierte I/O-Pins)
- ▶ Lösung eines linearen Gleichungssystems mit aus der Mathematik bekannten Verfahren

Verschiedene Verfahren

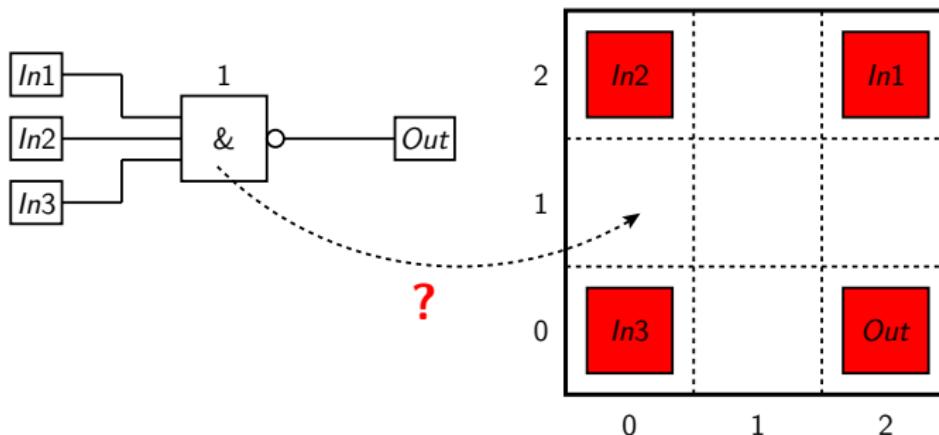
- ▶ Komplettes Gleichungssystem aufstellen, Lösung liefert die Zellenpositionen
- ▶ Für jede Zelle ZFT bestimmen, Überlappungen zulassen (illegal Initialplatzierung). Anschließend die Platzierung iterativ (z. B. durch Zellentausch) legalisieren.
- ▶ Abhängig vom Entwurfsstil können noch zusätzliche Schritte notwendig sein, z. B. Einordnung der Zellen in Zeilen bei einem channelled Standard-Zellen Gate-Array.

Ermittlung der ZFT-Position (x_i^0, y_i^0) einer Zelle i :

$$\sum_j w_{ij} \cdot (x_j^0 - x_i^0) = 0, \quad \sum_j w_{ij} \cdot (y_j^0 - y_i^0) = 0$$

$$x_i^0 = \frac{\sum_j w_{ij} \cdot x_j}{\sum_j w_{ij}}, \quad y_i^0 = \frac{\sum_j w_{ij} \cdot y_j}{\sum_j w_{ij}}.$$

Beispiel zur ZFT-Ermittlung



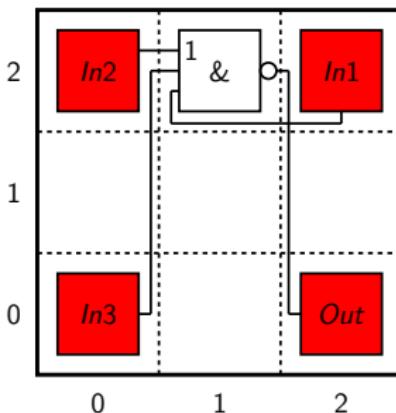
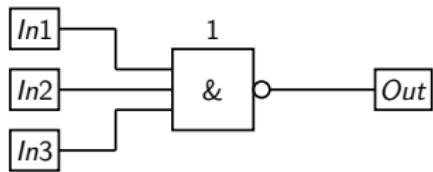
Fixierte Pad-Zellen: $In1(2,2)$, $In2(0,2)$, $In3(0,0)$, $Out(2,0)$,

Wichtungen: $w_{1In1} = 8$, $w_{1In2} = 10$, $w_{1In3} = 2$, $w_{1Out} = 2$.

$$x_1^0 = \frac{\sum_j w_{ij} \cdot x_j}{\sum_j w_{ij}} = \frac{w_{1In1} \cdot x_{In1} + w_{1In2} \cdot x_{In2} + w_{1In3} \cdot x_{In3} + w_{1Out} \cdot x_{Out}}{w_{1In1} + w_{1In2} + w_{1In3} + w_{1Out}} \approx 0,9$$

$$y_1^0 = \frac{\sum_j w_{ij} \cdot y_j}{\sum_j w_{ij}} = \frac{w_{1In1} \cdot y_{In1} + w_{1In2} \cdot y_{In2} + w_{1In3} \cdot y_{In3} + w_{1Out} \cdot y_{Out}}{w_{1In1} + w_{1In2} + w_{1In3} + w_{1Out}} \approx 1,6$$

Position des NAND-Gatters entsprechend der ZFT



Lösungsansätze bei bereits belegter ZFT-Position:

- ▶ Eine freie Position in der Nähe von ZFT suchen und belegen oder Kostenfunktion (z. B. $L(P)$) nach Vertauschen evaluieren und ggf. einen Tausch durchführen.
- ▶ Belegende Zelle auf nächste Position verdrängen (löst unter Umständen eine Kettenverschiebung aus).

Grober Ablauf des Algorithmus

1. Zufällige Anfangsplatzierung festlegen
2. Eine Zelle auswählen und ihre ZFT-Position berechnen
 - ▶ Falls ZFT-Position frei ist ➡ Zelle verschieben
 - ▶ Falls ZFT-Position belegt ist ➡ siehe letzte Seite
3. Schritt 2 wiederholen bis ein Abbruchkriterium erreicht ist.

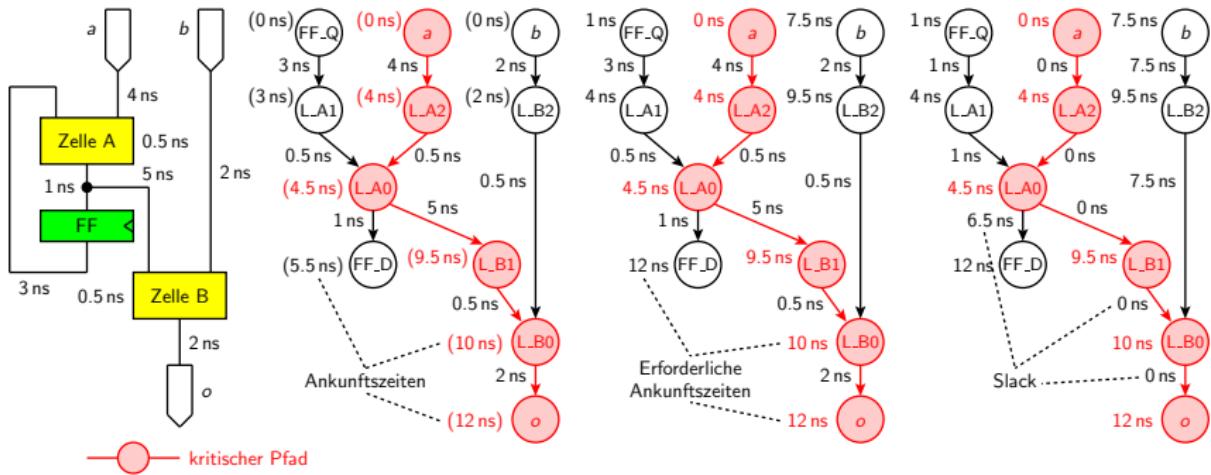
Anmerkungen:

- ▶ Es ist sinnvoll, die Zellen in vorbestimmter Reihenfolge auszuwählen (z. B. nach fallendem Verbindungsgrad)
- ▶ Um „endlose Tauschketten“ zu vermeiden, sollten einmal verschobene Zellen für eine weitere Verschiebung in der aktuellen Iteration gesperrt werden.
- ▶ Abbruchkriterium kann z. B. durch eine maximale Anzahl von Iterationen vorgegeben werden.

Übersicht

- ▶ *Simulated Annealing*: Vorlesung „Reconfigurable Computing“
- ▶ Quadratische Zuordnung (*quadratic assignment*): Numerische Lösung des Zuordnungsproblems bei quadratischer Kostenfunktion (darstellbar in Form einer quadratischen Matrix), zurückführbar auf das aus der Mathematik bekannte Eigenwert-Problem
- ▶ Neuronale Netze: Abbildung der Zellenkoordinaten auf Gewichte der Synapsen
- ▶ Evolutionäre Algorithmen: Iterative Verbesserung der Platzierung durch Selektion, Crossover und Mutation einer initialen Population (Menge von legalen Platzierungen).
- ▶ Timing-basierte Verfahren: Oft aus vorgestellten Verfahren durch Modifizieren der Kostenfunktion ableitbar

Grundlagen der Timing-Basierten-Verfahren



Ankunftszeit

$$T_{arrival}(i) = \max_{\forall j \in fanin(i)} (T_{arrival}(j) + delay(j,i))$$

Erforderlich

$$T_{required}(i) = \min_{\forall j \in fanout(i)} (T_{required}(j) - delay(i,j))$$

Slack

$$Slack(i,j) = T_{required}(j) - T_{arrival}(i) - delay(i,j)$$

Vorgehensweise bei der Timing-Analyse

- ▶ Ankunftszeiten werden iterativ berechnet (beginnend bei den Eingangsknoten), nachdem alle Eingangsknoten mit „0“ bewertet worden sind.
- ▶ Erforderliche Ankunftszeiten werden iterativ berechnet (beginnend bei den Ausgangsknoten), nachdem alle Ausgangsknoten mit der Verzögerung auf dem kritischen Pfad (D_{max}) bewertet worden sind.
- ▶ *Slack* wird berechnet, nachdem Ankunftszeiten und erforderliche Ankunftszeiten berechnet worden sind

Anmerkungen:

- ▶ Für alle Kanten auf dem kritischen Pfad gilt $Slack = 0$.
- ▶ *Slack* eines Pfades ist **nicht gleich** der Summe der *Slacks* aller Kanten dieses Pfades!

Aufgabe des Verdrahtungsschrittes

Verbindung aller Anschlüsse gleichen Potentials (\cong Netz) unter

- ▶ Zuordnung der durch die Netzliste vorgegebenen Verbindungen zu Leitungssegmenten und Verdrahtungsebenen
- ▶ Einhaltung von Randbedingungen (z. B. Kreuzungsfreiheit, Kapazität von Verdrahtungskanälen)
- ▶ Optimierung von Zielfunktionen (z. B. Verbindungslänge)

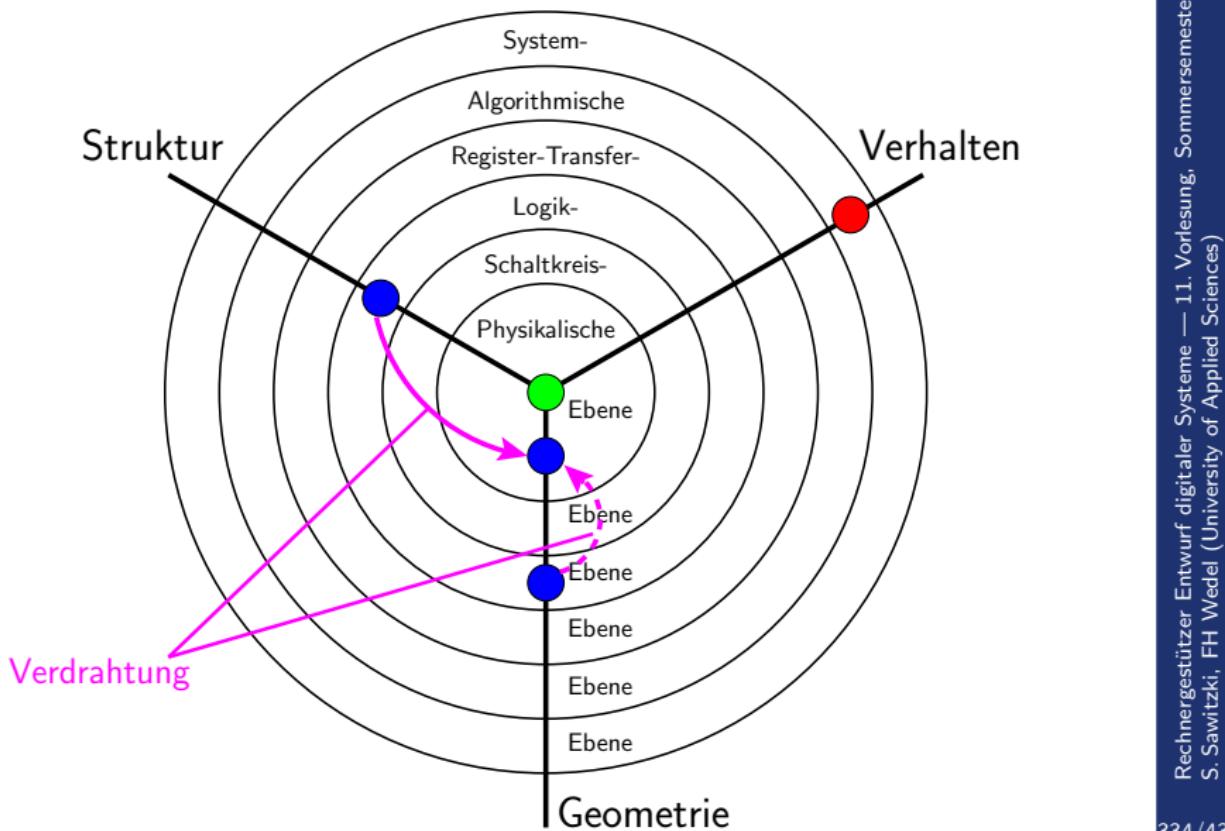
Eingabe:

- ▶ Platzierte Zellen
- ▶ Netzliste mit Angaben zu den zu verbindenden Anschlüssen

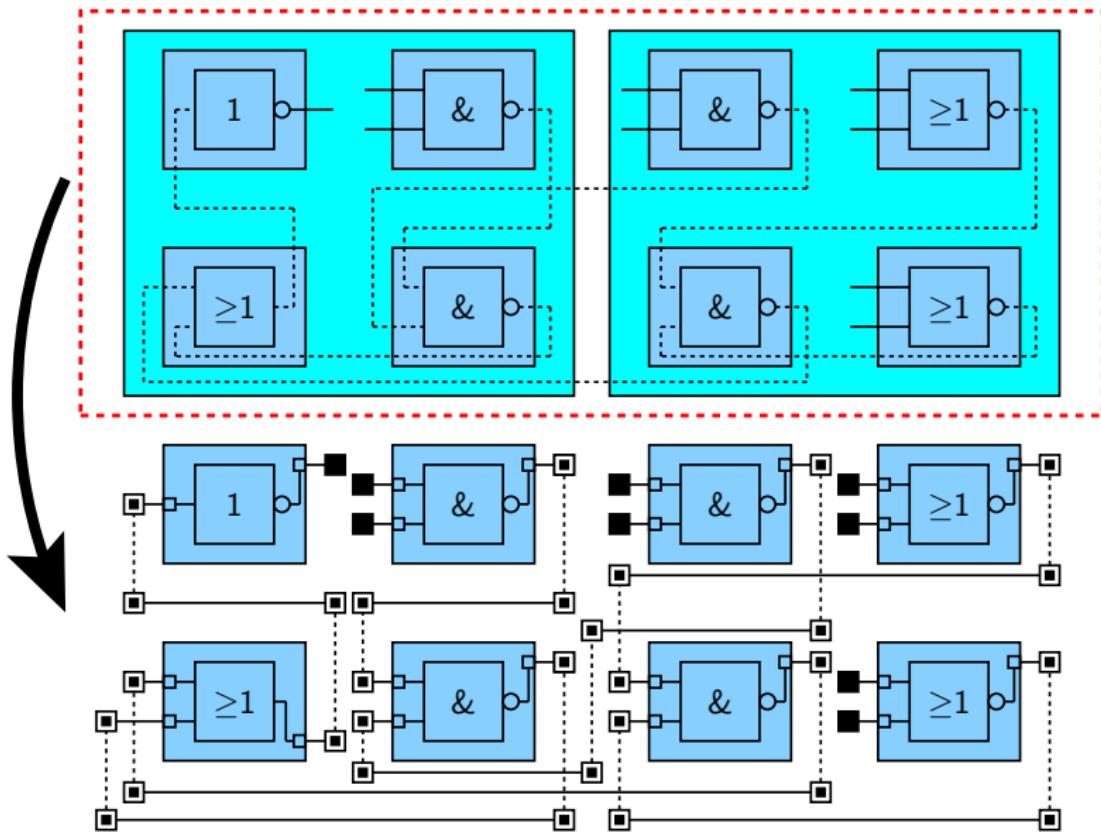
Ausgabe:

- ▶ Schaltungs-Layout

Einordnung in den Entwurfsprozess



Veranschaulichung des Verdrahtungsschrittes



Große Einteilung der Verdrahtungsalgorithmen

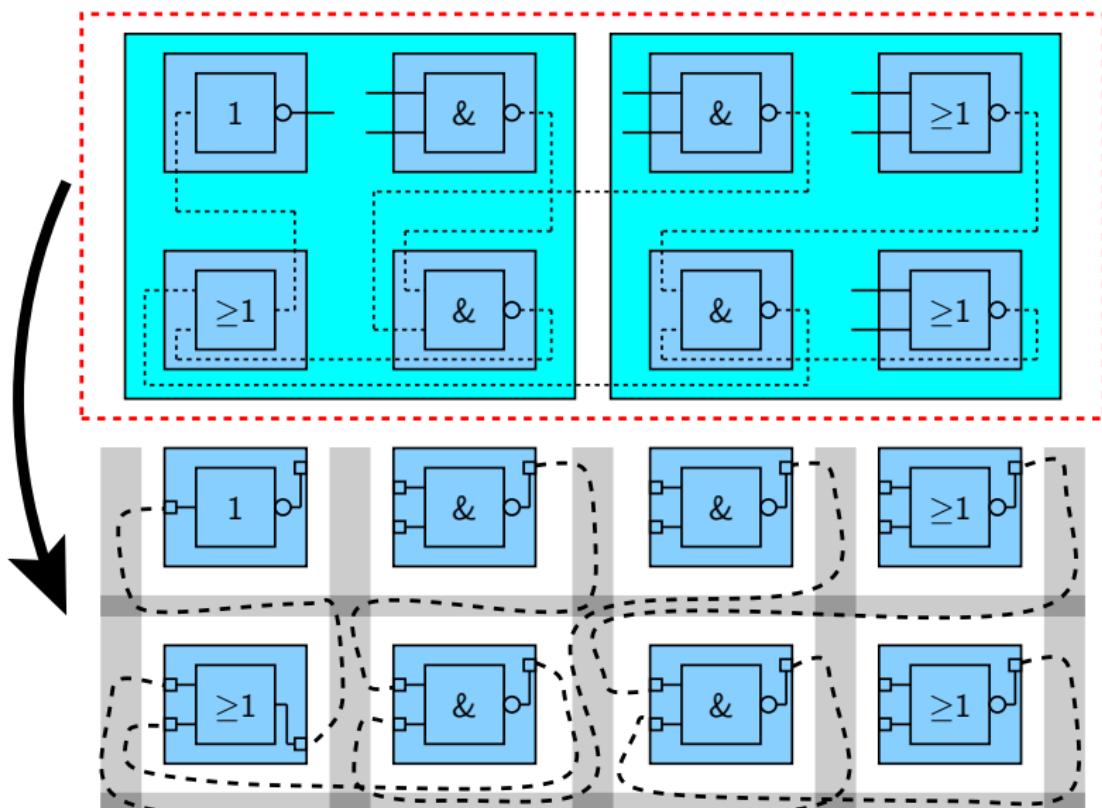
Spezialverdrahter (*special router*): Algorithmen zur Verdrahtung der Netze mit besonderen Aufgaben (Stromversorgung, Masse, Takt)

Globale Verdrahter (*global router*): Festlegung der Anschlüsse und Verdrahtungskanäle (auch Kanalsegmente) ohne Spezifikation einzelner Leitungen (Zuordnung zu Verdrahtungsregionen).

Detail-Verdrahter (*detailed router*): Festlegung der Leitungen innerhalb einer Verdrahtungsregion.

Flächenverdrahter: Kombinierte globale und Detail-Verdrahter (ein Algorithmus anstelle zwei aufeinanderfolgender Schritte). Verdrahtung auf der gesamten Layout-Fläche ohne vorherige Zuweisung der Regionen.

Aufgabe der Globalverdrahtung



Verdrahtungsregionen

Verdrahtungsregion (*routing region*)

ist ein Bereich des Layouts, der größer als das Layoutraster ist.
Kanäle und Switchboxen sind Beispiele von Verdrahtungsregionen.

Präzisierung der Aufgaben der Globalverdrahtung:

- ▶ Netzreihenfolge der Verdrahtung innerhalb der Netzliste festlegen
- ▶ Anschlussreihenfolge der Verdrahtung innerhalb eines Netzes festlegen

Die meisten Verdrahtungsalgorithmen sind sequentieller Natur, daher ist die Reihenfolge ausschlaggebend für die Qualität (oder auch Existenz) des Verdrahtungsergebnisses.

- ➡ Bewertung der Netze (Gewichtung, *Criticality*, Anzahl der Anschlüsse usw.)

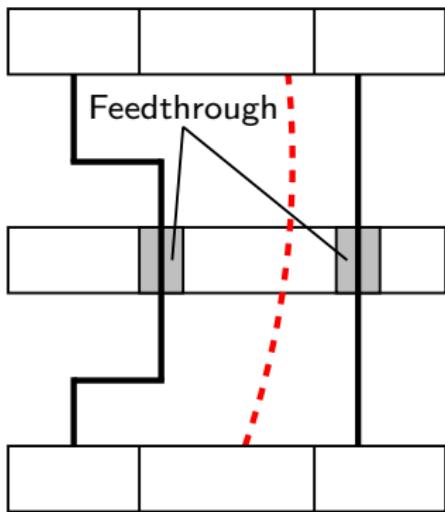
Hauptoptimierungsziele von Verdrahtungsalgorithmen (bei Globalverdrahtung)

- ▶ Ermittlung der Verdrahtbarkeit einer Platzierung
- ▶ Minimierung der benötigten Ressourcen pro Verdrahtungsregion bzw. Einhaltung der Kapazitätsvorgaben einer Region

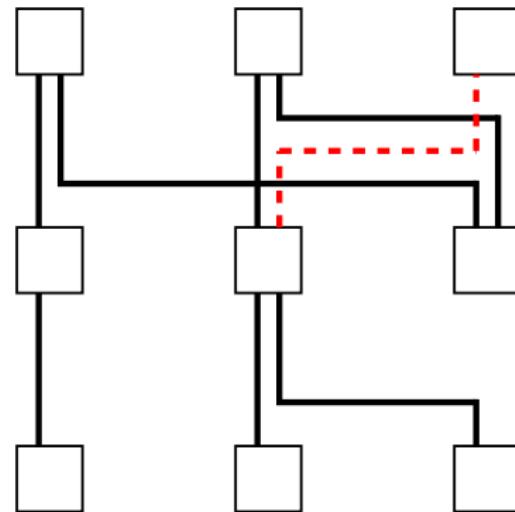
Ziele und Aufgaben der Globalverdrahtung sind von der Art des Entwurfs abhängig:

- ▶ Bei kundenspezifischen Entwürfen müssen z. B. die Kanäle (Lage und Kapazität) festgelegt werden, während bei Gate-Arrays die Kanäle bereits vorgegeben sind.
- ▶ Nach der Feststellung der Verdrahtbarkeit kann die Gesamtverbindungslänge oder die Länge des kritischen Pfades ein Optimierungskriterium sein.

Beispiele von nicht verdrahtbaren Platzierungen



Nicht genug Durchgangszellen

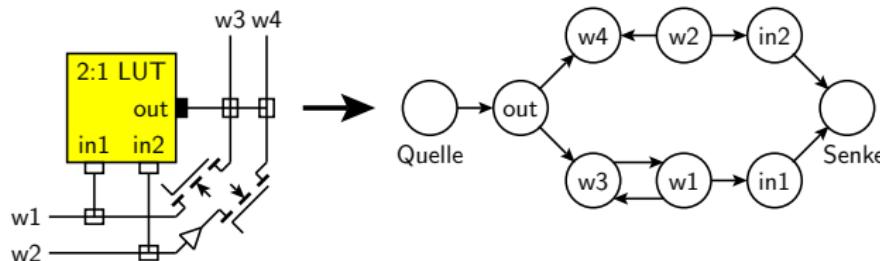


Überschreitung der maximalen Kanalbreite (hier =2)

- Eine schlechte Platzierung kann eine Verdrahtbarkeit ausschließen
(\cong selbst der beste Verdrahter bei uneingeschränkter Laufzeit findet keine Lösung!)

Modellierung der Verdrahtungsregionen mit Hilfe von *Routing-resource graph*

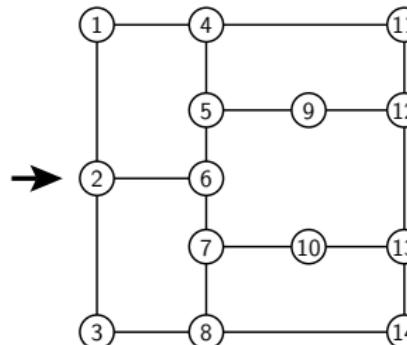
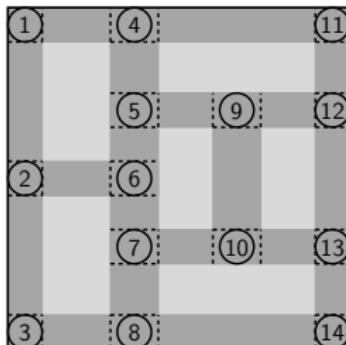
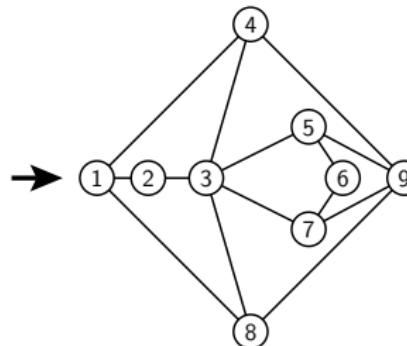
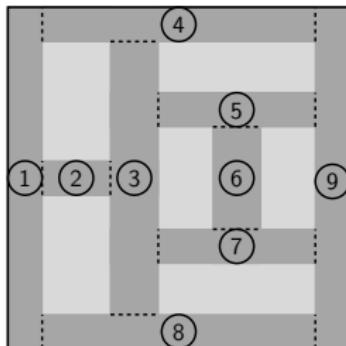
... am Beispiel eines FPGA:



- ▶ Jeder Anschlusspunkt und jede Leitung ist ein Knoten, die Verbindungen zwischen den Knoten repräsentieren Leitungen
- ▶ Gerichtet, da unidirektionale Verbindungen bestehen
- ▶ Quelle und Senke stellen die Gleichwertigkeit der LUT-Anschlüsse dar, da sie durch Umprogrammierung des Speicherinhaltes beliebig gegeneinander austauschbar sind (\cong funktional äquivalente Anschlüsse)

Verbindungsgraphen

Kanal-Verbindungsgraph (*channel connectivity graph*) und Switchbox-Verbindungsgraph (*channel intersection graph*):



Allgemeiner Ablauf der Globalverdrahtung

1. Verdrahtungsregionen festlegen (*region definition*): Kanäle, Switchboxen, Tiles. Aufstellung eines Kanalverbindungsgraphen bzw. Switchbox-Verbindungsgraphen.
 2. Zuordnung der Netze zu den Verdrahtungsregionen (*region assignment*): Verteilung von Netzen auf Regionen unter Berücksichtigung von Randbedingungen (Kanalkapazitäten, Laufzeiten, Gleichverteilung)
 3. Anschluss-Zuweisung (*pin assignment*): Festlegung von Anschlüssen von Netzen am Rande von Verdrahtungsregionen
- Festlegung aller Parameter, die für die Feinverdrahtung notwendig sind.

STEINERbäume

(Minimaler) rektilinearer STEINERbaum, STEINERknoten

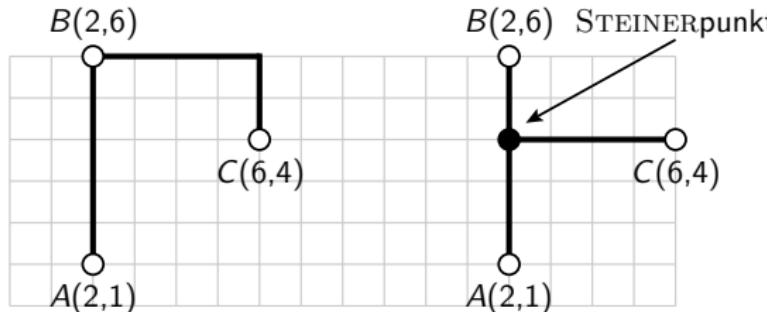
Gegeben sei ein Netz mit p Terminals, das auf einem Raster (horizontal und vertikal) angeordnet ist. Ein Graph heißt rektilinearer STEINERbaum (RST), wenn er alle p Pins und beliebig viele Rasterknoten als Punkte enthält. Dabei heißen alle Knoten, die keine Terminals sind, STEINERknoten. Ein RST mit minimaler Kantenlänge heißt minimaler rektilinearer STEINERbaum (MRST).

MRST hat folgende Eigenschaften:

- ▶ Für die Anzahl von STEINERknoten s gilt $0 \leq s \leq p - 2$.
- ▶ Knotengrad der Terminalknoten ist $1 \dots 4$, Knotengrad der STEINERknoten ist 3 oder 4.
- ▶ MRST liegt immer innerhalb des umschließenden Rechtecks (*minimal rectangle, MR*) des Netzes.
- ▶ Für Längen von MRST und MR gilt $L_{MRST} \geq L_{MR}/2$.

Erzeugung von MRST ist NP-äquivalent.

Spannbaum vs. STEINERbaum

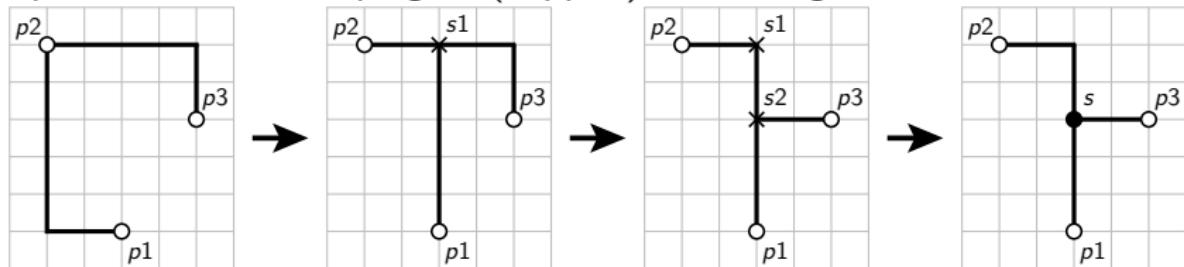


Jakob STEINER
(1796–1863, schweizer
Mathematiker)

- ▶ Ein minimaler rektilinearer Spannbaum (Graph, deren Kanten nur Terminals des Netzes auf dem kürzesten Weg verbinden, ohne Hinzunahme weiterer Knoten) kann in polynomieller Zeit erzeugt werden
- ▶ Minimaler rektilinearer Spannbaum löst das Problem der globalen Verdrahtung, ist jedoch stark suboptimal; STEINERbaum liefert bessere Verdrahtung sowie Anschlusspunkte für Mehrterminalnetze
- ▶ Heuristische Methoden zur Erzeugung eines MRST aus einem minimalen rektilinearen Spannbaum

Spannbaum \Rightarrow STEINERbaum

Beispiel zur Erzeugung eines Steinerbaums aus einem rektilinearen Spannbaum durch Spiegeln (Kippen) von L-Segmenten:



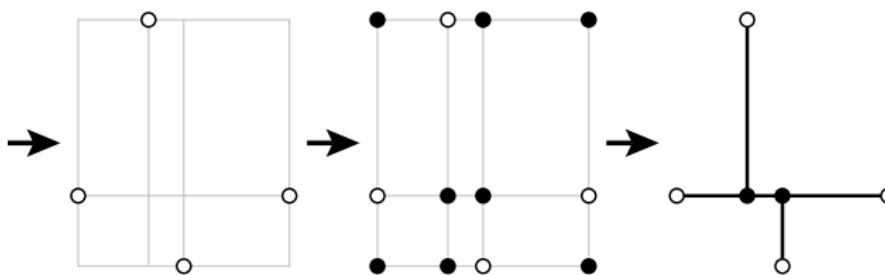
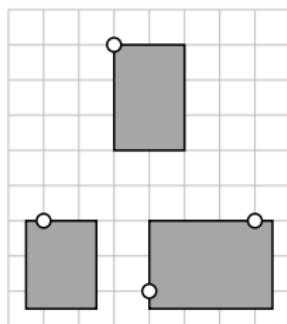
Verkürzung der Gesamtverbindungsstrecke:

- ▶ 13 im rektilinearen Spannbaum
- ▶ 9 im Steinerbaum

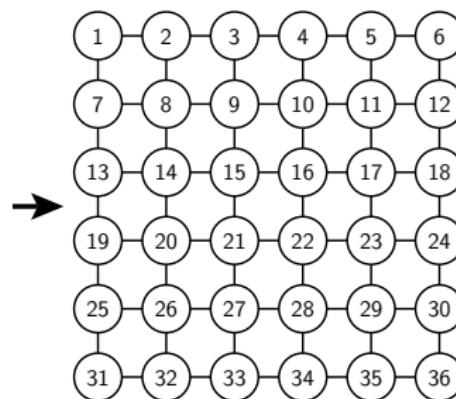
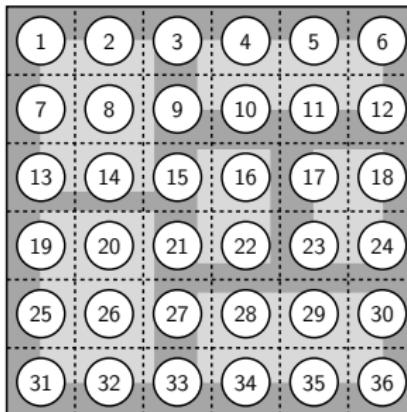
Im Allgemeinen können Steinerpunkte nur auf Kreuzungspunkten der Gitterlinien, die durch die Anschlusspunkte gehen, liegen (HANAN-Punkte).

M. HANAN, *On Steiner's Problem with Rectilinear Distance*, SIAM Journal on Applied Mathematics, Vol. 14, Issue 2, pp. 255–265, March 1966

Erzeugung von STEINER-Punkten mit Hilfe von HANAN-Punkten

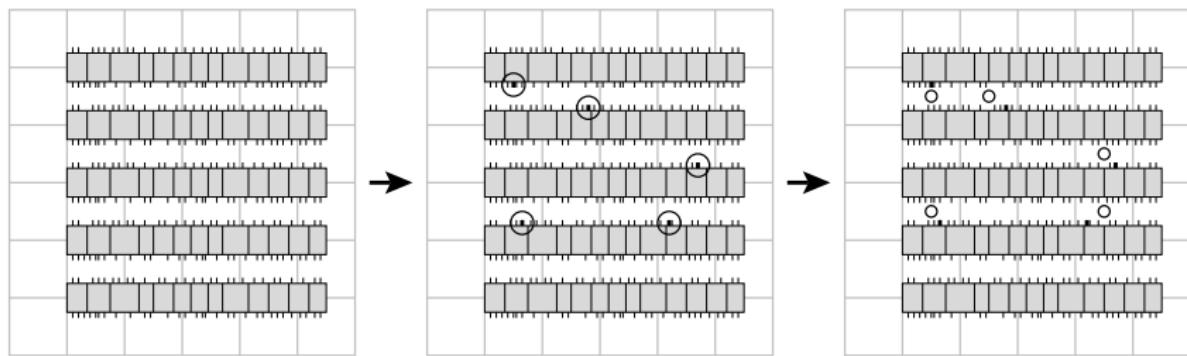


Gittergraph-
Modell:



Globalverdrahtung eines Gate-Arrays mit Hilfe von STEINER-Bäumen

Abbildung eines Standardzellen-Layouts auf einen Gittergraphen und Anordnung der Netz-Terminals zur STEINERbaum-Erzeugung:



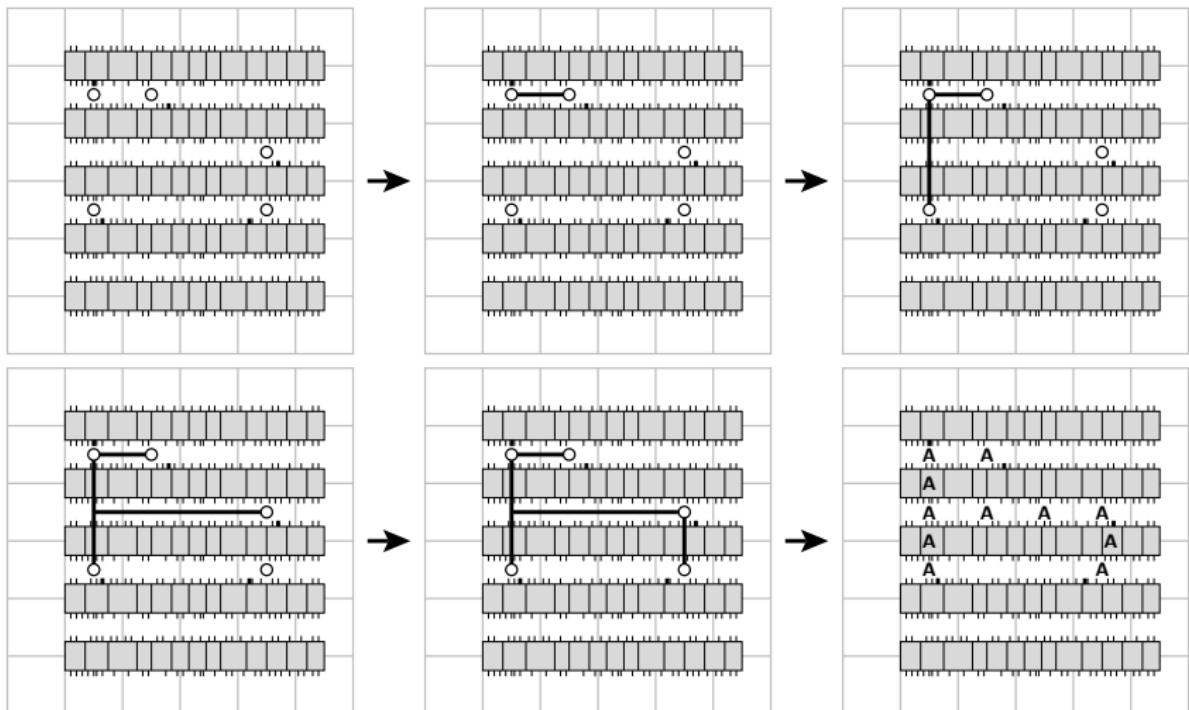
Die Anschlüsse werden in der Mitte der jeweiligen Globalzelle angenommen (genaue Position muss später bei der Detailverdrahtung berücksichtigt werden).

- Globalverdrahtung durch Aufspannen eines STEINERbaums.

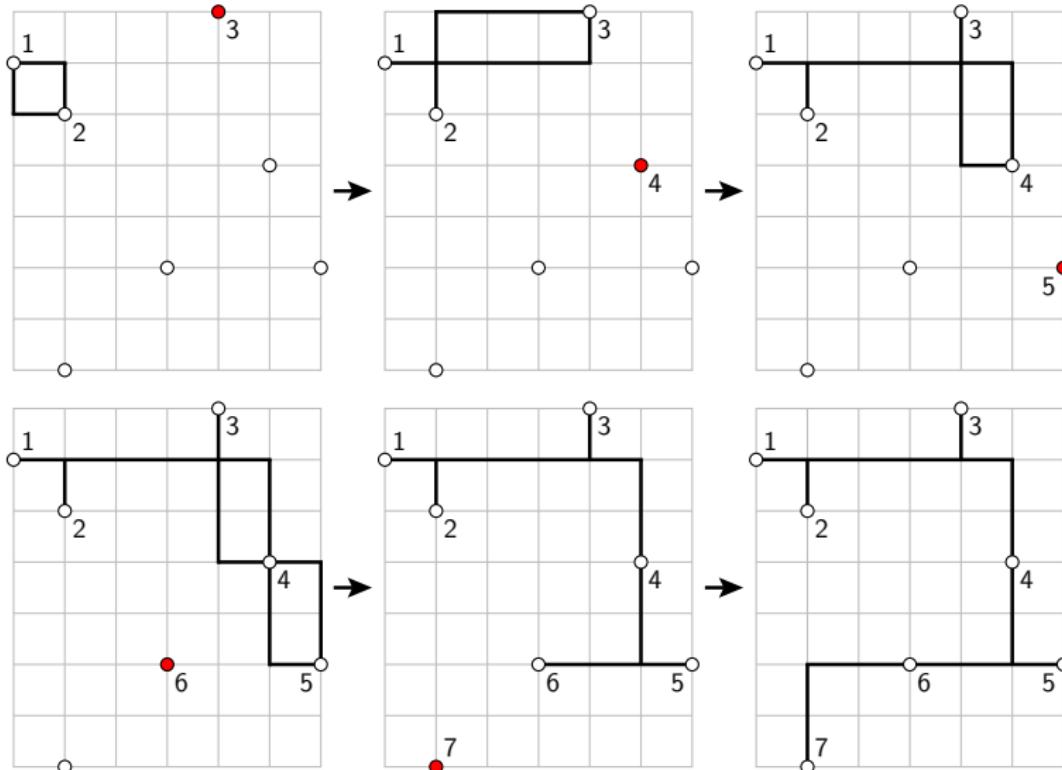
STEINERbaum-Algorithmus

1. Das Anschlusspaar mit minimalem Manhattan-Abstand ermitteln und das entsprechende minimale umschließende Rechteck erzeugen ($\cong 2 \text{ MRST}$).
2. Den Anschluss mit minimalem Manhattan-Abstand zur aktuellen MRST finden und mit dem umschließenden Rechteck verbinden.
3. Falls eine der erzeugten Verbindungen im Steinerknoten eines der beiden MRST endet, muss der andere MRST gelöscht werden.
4. Falls es weitere Anschlüsse gibt und mehr als zwei Maschen existieren, muss ein MRST willkürlich gelöscht und mit dem Schritt 2 weiter verfahren werden. Sonst (keine weiteren Anschlüsse mehr) alle Maschen eliminieren (entsprechende MRST löschen) und ENDE.

Veranschaulichung des STEINERbaum-Algorithmus



Ein etwas komplexeres Beispiel zum STEINERbaum-Algorithmus



Sortierte Wegsuche

Bei der Modellierung der Verdrahtungsregionen mit einem Verbindungsgraphen kann die Globalverdrahtung durch sortierte Wegsuche erfolgen:

1. Festlegen der Netzreihenfolge (z. B. durch Vorsortierung nach vorgegebenen Kriterien)
2. Anschlussreservierung für alle Netze
3. Globalverdrahtung eines ausgewählten Netzes:
 - 3.1. Aufheben der Anschlussreservierungen
 - 3.2. Auswahl von 2 zu verbindenden Terminals
 - 3.3. Suche des kürzesten Pfades (Abbruch, wenn kein Pfad existiert)
 - 3.4. Aktualisierung der Kanalkapazitäten
 - 3.5. Auswahl des nächsten Terminals und zurück zum Schritt 3.3 (falls kein Terminal vorhanden weiter mit Schritt 4)
4. Auswahl des nächsten Netzes und zurück zum Schritt 3 bzw. ENDE, wenn alle Netze verdrahtet

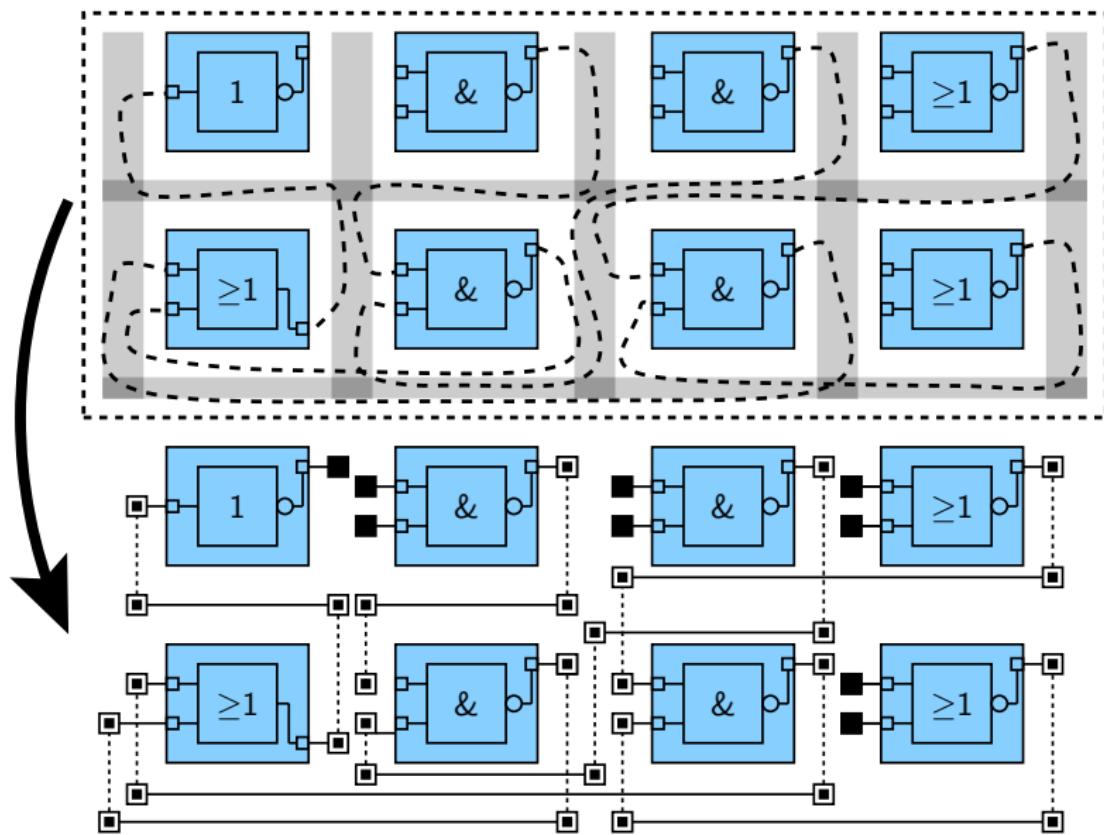
Weitere Algorithmenklassen für die Globalverdrahtung

Parallele Netz betrachtung durch sukzessive Aufteilung des Layouts und Zuweisung der Netze zu den so entstehenden Regionen (Vorsortieren und Aufteilen in 2-Terminal-Verbindungen entfällt).

Numerische Methoden, d. h. Darstellung des Verdrahtungsproblems in Form eines Gleichungssystems (und dessen Lösung). Es werden dabei alle Verdrahtungswege betrachtet, der Rechenaufwand ist jedoch (für Probleme nennenswerter Größe) enorm hoch.

Stochastische Algorithmen wie simulierte Abkühlung oder evolutionäre Algorithmen.

Aufgabe der Feinverdrahtung



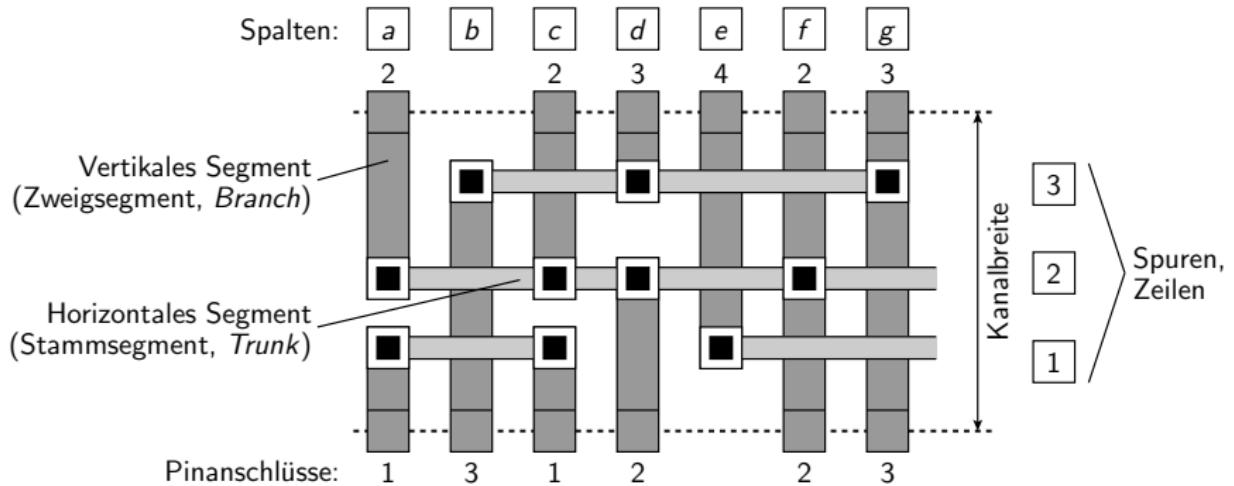
Kanalverdrahtung

Bei der Feinverdrahtung werden die bei der Globalverdrahtung auf die Verdrahtungsregionen aufgeteilten Netzsegmente innerhalb dieser Regionen einzelnen Verdrahtungswegen und -ebenen zugewiesen:

- ▶ Ursprünglich oft als Kanalverdrahtung auf zwei Verdrahtungsebenen bezogen
- ▶ Mittlerweile durch Steigerung der Anzahl von Verdrahtungsebenen als mehrdimensionales Verdrahtungsproblem formuliert (OTC-Verdrahtung, 3D-Switchbox-Verdrahtung)

Modell der Kanalverdrahtung: Freie rechteckige Fläche mit Anschlüssen auf **zwei gegenüberliegenden** Seiten. Zu verbindende Anschlüsse sind auf vertikalen Gitterlinien angeordnet. Horizontale bzw. vertikale Leitungssegmente bilden Zeilen bzw. Spalten.

Veranschaulichung des Kanalmodells bei Feinverdrahtung



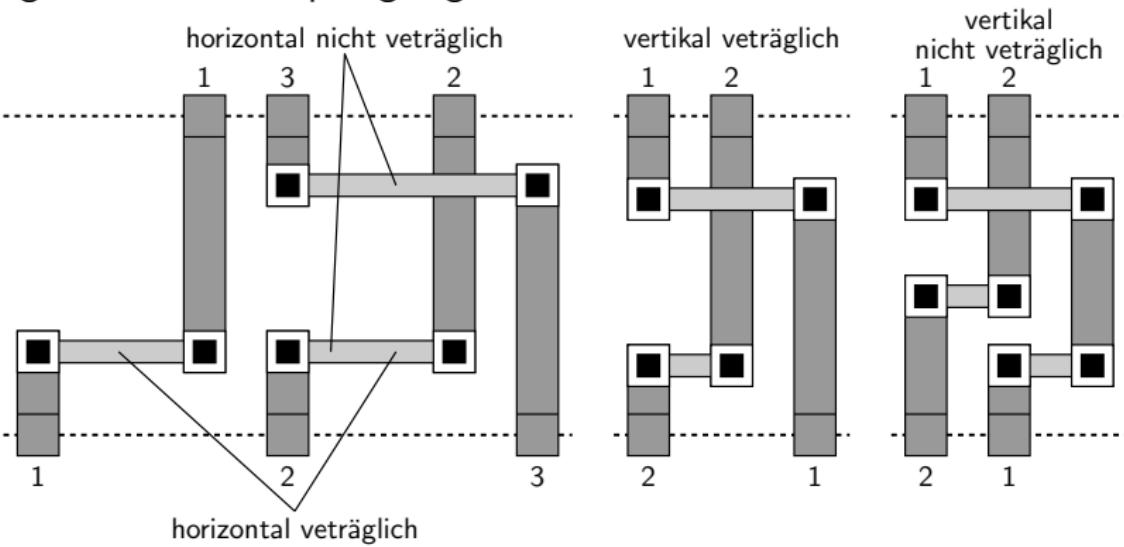
Anschlusskennzeichnung mit $BOT(k)$ und $TOP(k)$ Mengen durch explizite Auflistung der Netznummern ($0 \cong$ nicht angeschlossen):

$$BOT(k) = \{1, 3, 1, 2, 0, 2, 3\}$$

$$TOP(k) = \{2, 0, 2, 3, 4, 2, 3\}$$

Vertikale und horizontale Verträglichkeit (horizontal and vertical constraints)

Zwei Netze sind horizontal verträglich, wenn ihre horizontalen Segmente in eine Spur gelegt werden können.



Zwei Netze sind vertikal verträglich, wenn die Lage ihrer vertikalen Segmente eine Horizontalzuordnung mit minimaler Spuranzahl ermöglicht.

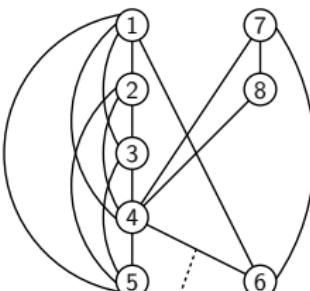
Zonen- und Graphendarstellung der horizontalen Verträglichkeit

Spalte:	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>j</i>	<i>l</i>
0	2	4	5	2	6	7	0	4	0	0	0

1	3	5	3	5	1	6	8	0	8	7
---	---	---	---	---	---	---	---	---	---	---

<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>j</i>	<i>l</i>
0	2	4	5	2	6	7	0	4	0	0
—	1	—	—	—	—	—	—	—	—	—
—	2	—	—	—	—	—	—	—	—	—
—	3	—	—	—	—	—	—	—	—	—
—	4	—	—	—	—	—	—	—	—	—
—	5	—	—	—	—	—	—	—	—	—
—	6	—	—	—	—	—	—	—	—	—
—	7	—	—	—	—	—	—	—	—	—
1	3	5	3	5	1	6	8	0	8	7

<i>S(c)</i>	<i>S(f)</i>	<i>S(g)</i>	<i>S(i)</i>
1	—	7	—
2	6	—	8
3	—	—	—
4	—	—	—
5	—	—	—



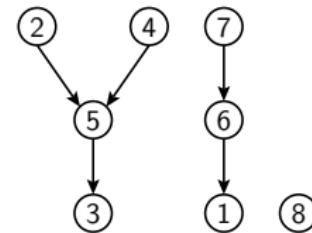
Netze benötigen unterschiedliche Spuren

Die minimale Kanalkapazität (Anzahl der Spuren) entspricht der maximalen Mächtigkeit der Mengen $S(k)$ bzw. dem längsten Pfad durch den Verträglichkeitsgraphen.

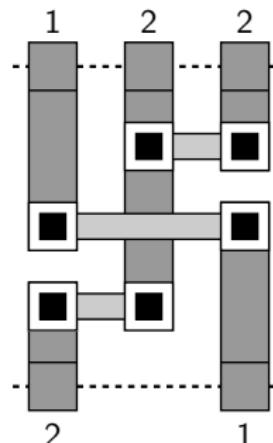
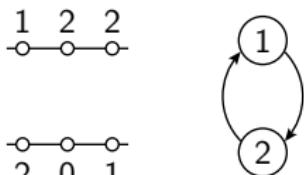
Graphendarstellung der vertikalen Verträglichkeit

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>j</i>	<i>l</i>
0	2	4	5	2	6	7	0	4	0	0	0

1	3	5	3	5	1	6	8	0	8	7
---	---	---	---	---	---	---	---	---	---	---



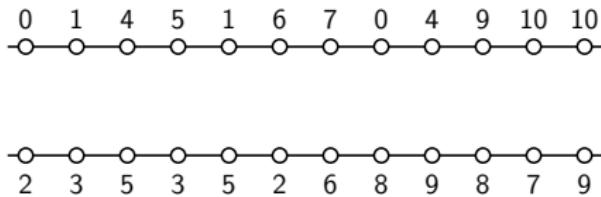
Eine gerichtete Kante oder ein Pfad zwischen den Knoten bedeuten, dass entsprechende Anschlüsse übereinander (auf gleicher Position) angeordnet sind.



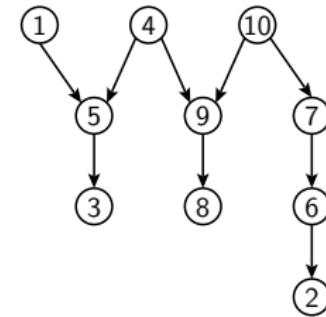
Left-Edge-Algorithmus

1. Aufstellung des vertikalen Verträglichkeitsgraphen (VVG) sowie der Zonendarstellung.
2. Aktuelle Spur $j = 1$ (obere Spur)
3. Für aktuelle Spur j werden alle Netze ohne Vorgänger im VVG verdrahtet: Das am weitesten links liegende Netz in der Zonendarstellung, anschließend weitere nicht überlappende (laut Zonendarstellung) und vorgängerlose (laut VVG) Netze. Nach dem Verdrahten werden entsprechende Netze im VVG und in der Zonendarstellung gelöscht.
4. Aktuelle Spur $j = j + 1$.
5. Wenn noch unverdrahtete Netze existieren, weiter mit dem Schritt 3, sonst ENDE.

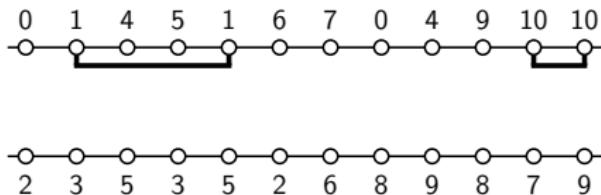
Ein Beispiel zum Left-Edge-Algorithmus



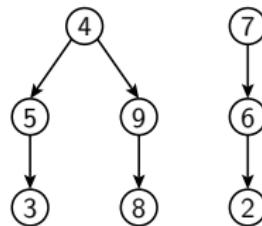
1			7
2			8
3			9
	4		10
5	6		



Die Netze ohne Vorgänger (1, 4 und 10) werden in der angegebenen Reihenfolge auf der Spur 1 verdrahtet, wobei nur die Netze 1 und 10 in einer Spur angeordnet werden können:

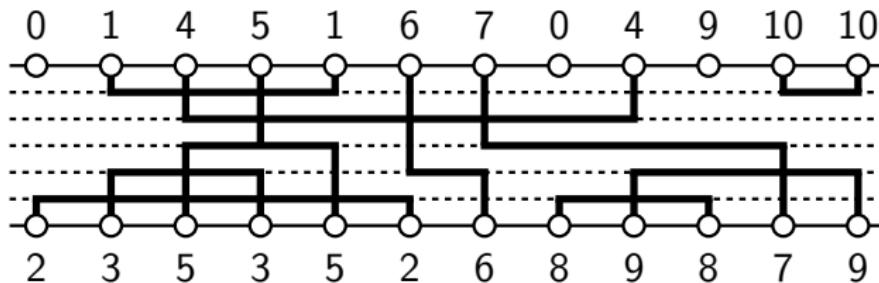


1			7
2			8
3			9
4			10
5	6		



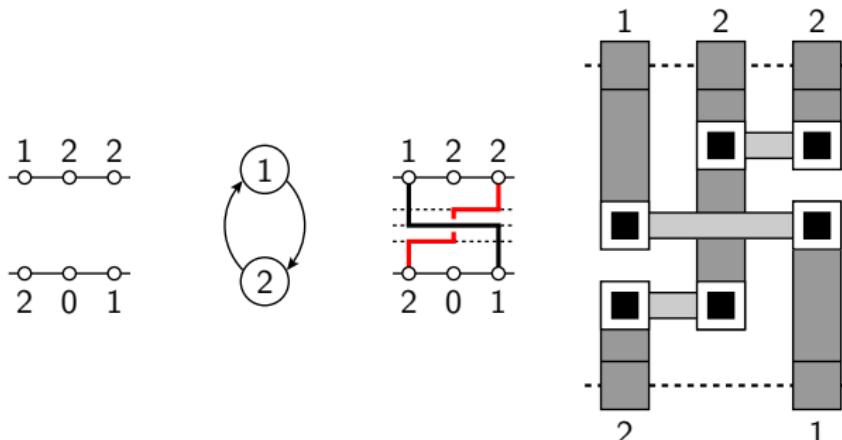
Ein Beispiel zum Left-Edge-Algorithmus (fortgesetzt)

Nach der Füllung der Spur 1 kommen Netze 4 und 7 für die 2. Spur in Frage. Das Netz 4 liegt weiter links und wird verdrahtet, das Netz 7 wird auf die nächste Spur verlegt. Endgültige Lösung:

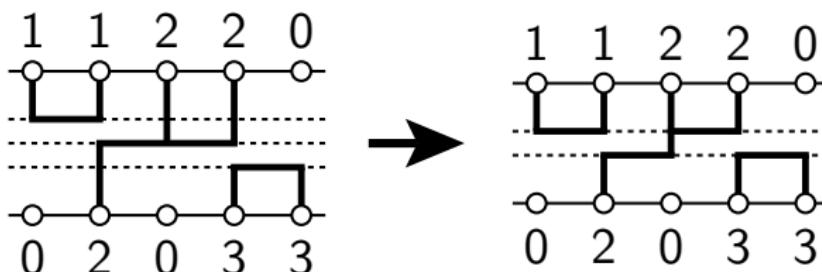


- ▶ Funktioniert nur bei zyklifenfreien VVG.
- ▶ Erweiterung auf Dogleg-Left-Edge-Algorithmus (Aufsplittung des horizontalen Netzsegmentes, *Dogleg* \cong „Hundebein“)

Ein Beispiel zur Dogleg-Einfügung

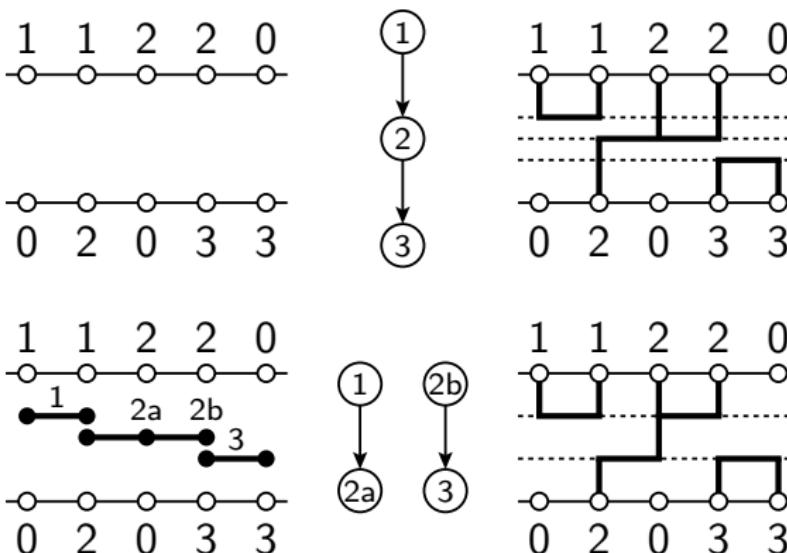


Zusätzlich auch zum Einsparen von Spuren:



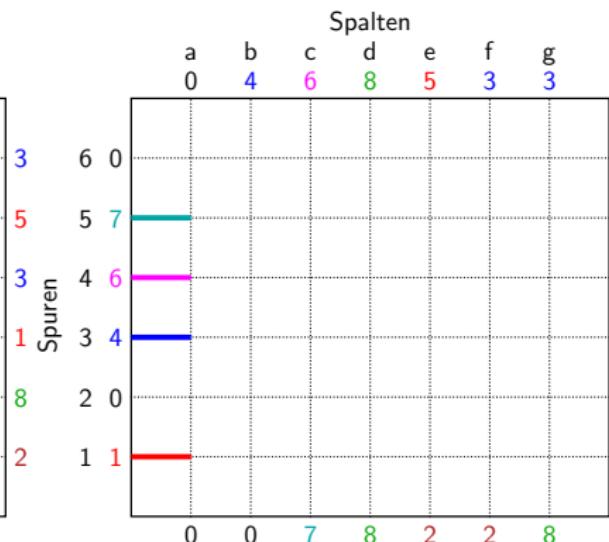
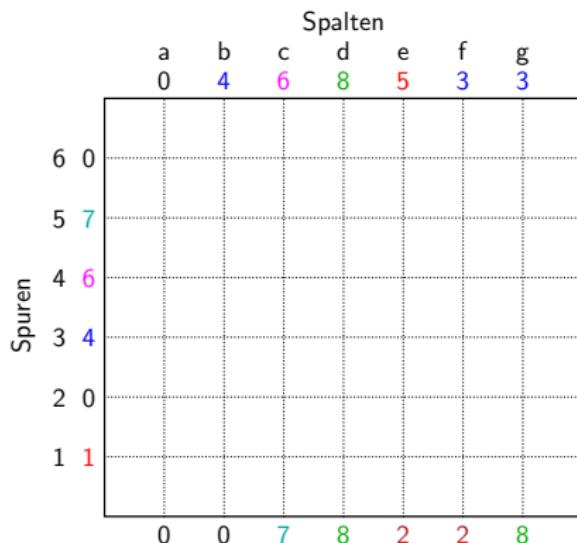
Grundlegende Vorgehensweise bei *Dogleg-Left-Edge*-Algorithmus

Alle Netze mit $p > 2$ Anschlüssen in $p - 1$ horizontale Segmente aufteilen (an den Spaltenpositionen, an denen das Netz einen Anschluss hat). Ansonsten ist das Verfahren dem klassischen Left-Edge-Algorithmus gleich.



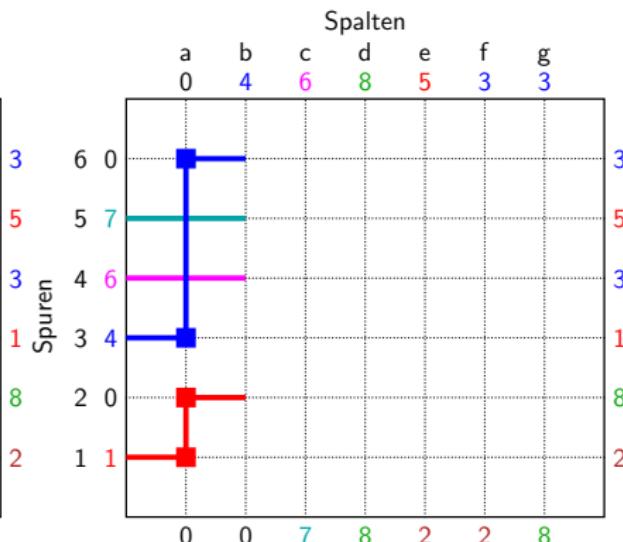
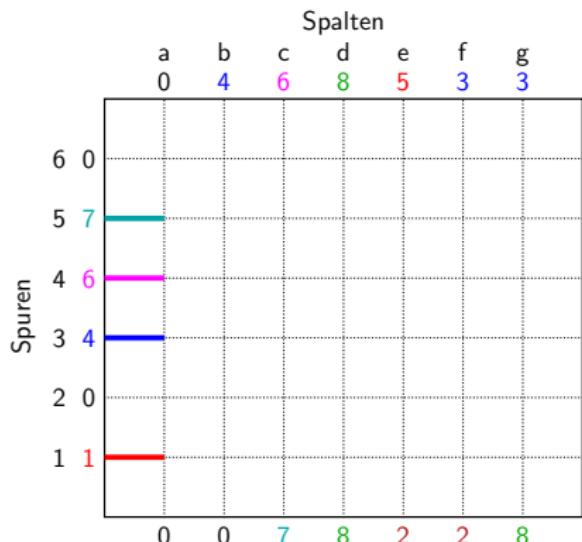
Beispiel zur Switchbox-Verdrahtung (1)

Greedy-Strategie, spaltenweise Verarbeitung von links nach rechts:



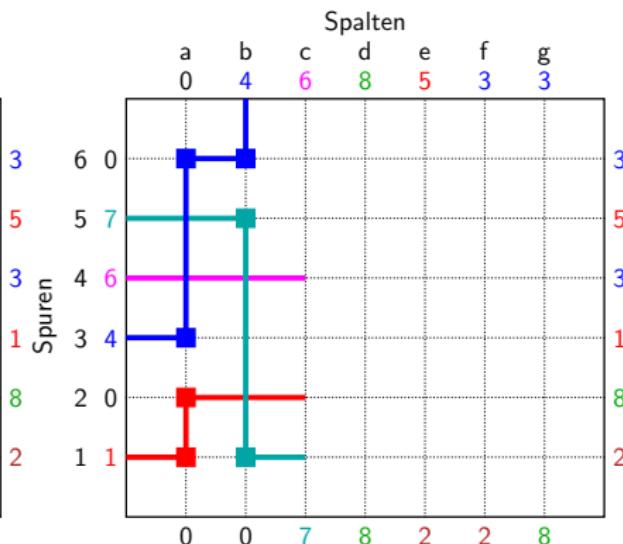
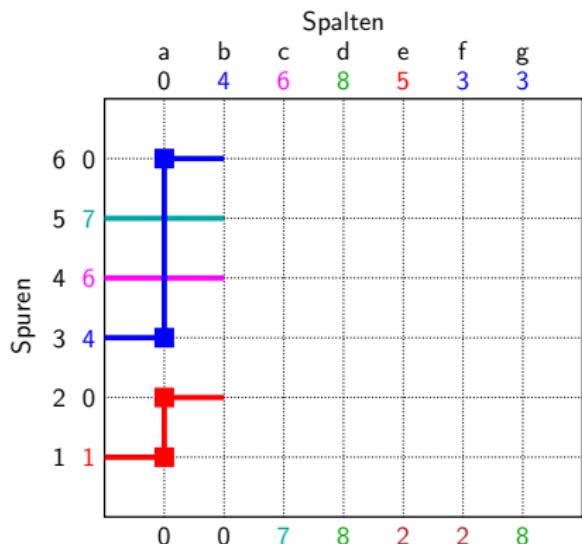
Am linken Rand starten: Erzeugung der Pinnetze 1, 4, 6 und 7,
Fortführung in den Spuren 1, 3, 4 und 5

Beispiel zur Switchbox-Verdrahtung (2)



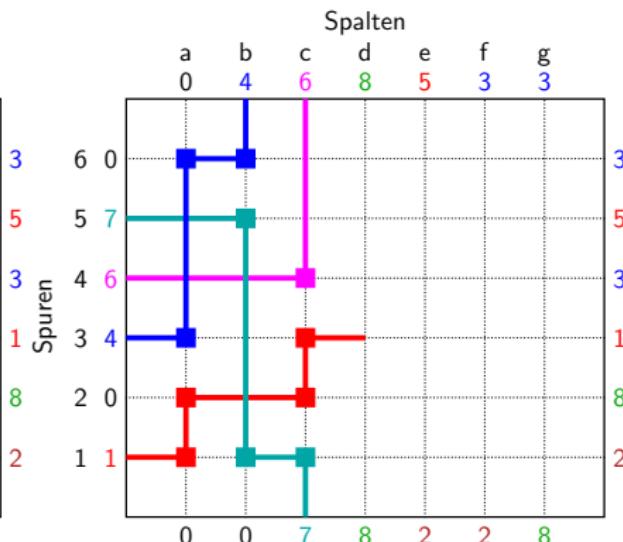
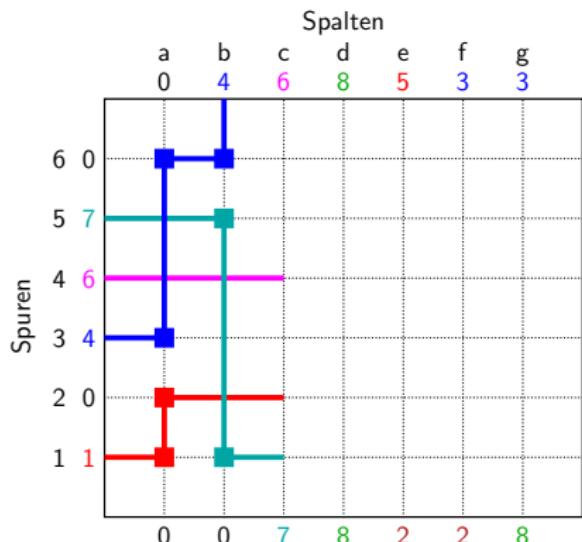
Spalte a: 4 ist das nächste Netz oben (Verlegung in die Spur 6),
Netz 1 muss in die Spur 3, daher Verlegung in die Spur 2 (nach
oben), Fortführung in den Spuren 2, 4, 5 und 6

Beispiel zur Switchbox-Verdrahtung (3)



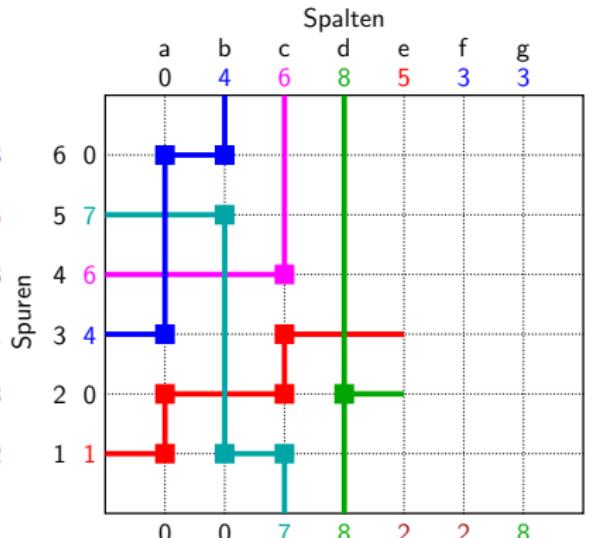
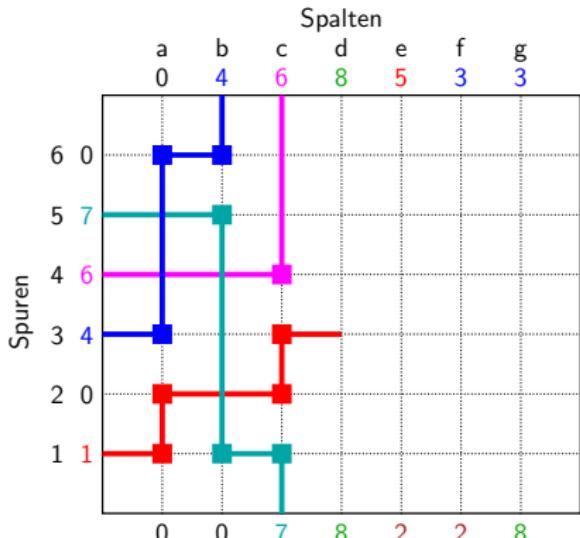
Spalte *b*: Netz 4 kann nach oben verdrahtet werden, 7 ist das nächste Netz unten (Verlegung in die Spur 1), Fortführung in den Spuren 1, 2 und 4

Beispiel zur Switchbox-Verdrahtung (4)



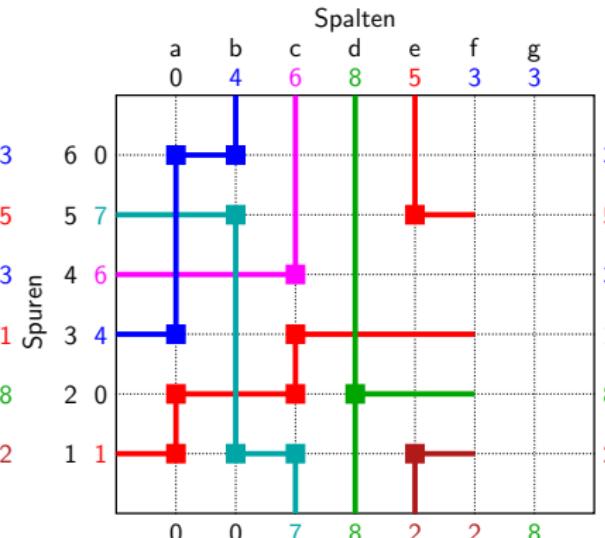
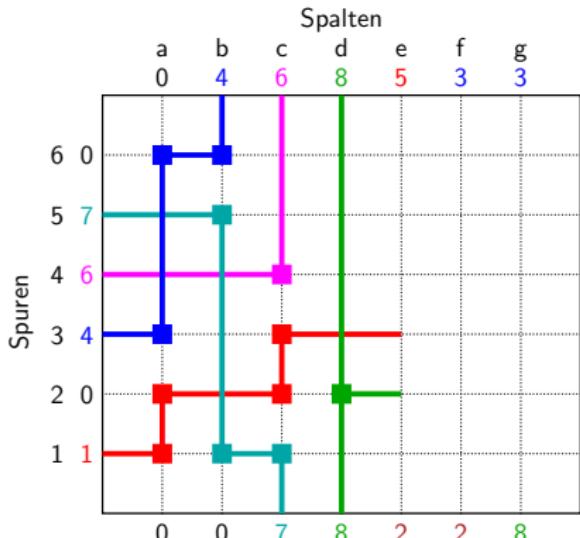
Spalte c: Netz 7 kann nach unten verdrahtet werden, Netz 6 kann nach oben verdrahtet werden, Netz 1 muss in die Spur 3, daher Verlegung in die Spur 3 (nach oben), Fortführung in der Spur 3

Beispiel zur Switchbox-Verdrahtung (5)



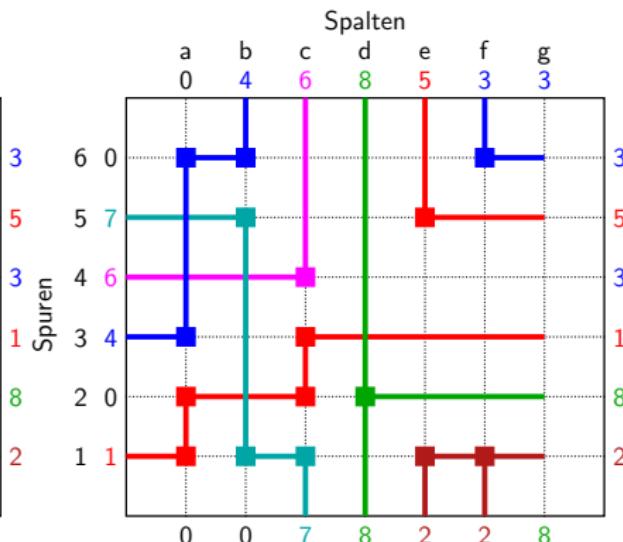
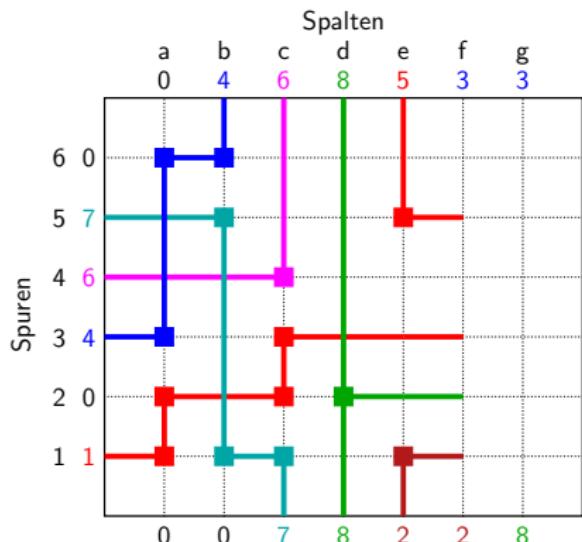
Spalte *d*: Netz 8 kann von unten nach oben verdrahtet (und in der Spur 2 fortgeführt) werden, Fortführung in den Spuren 2 und 3

Beispiel zur Switchbox-Verdrahtung (6)



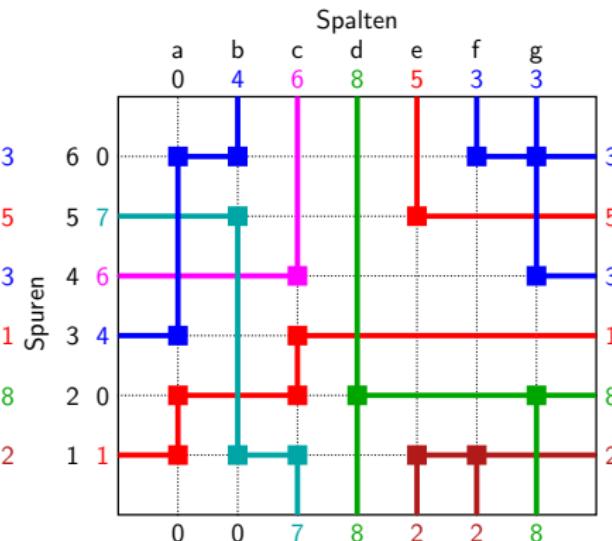
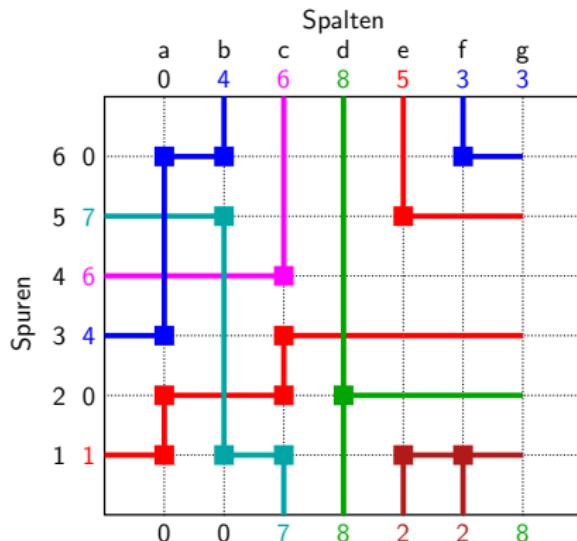
Spalte e: Erzeugung der Pinnetze 2 (Verlegung in die Spur 1) und 5 (Verlegung in die Spur 5), Fortführung in den Spuren 1, 2, 3 und 5

Beispiel zur Switchbox-Verdrahtung (7)



Spalte *f*: Erzeugung des Pinnetzes 3 (Verlegung in die Spur 6), Netz 2 kann nach unten verdrahtet werden, Fortführung in den Spuren 1, 2, 3, 5 und 6

Beispiel zur Switchbox-Verdrahtung (8)



Spalte g: Netz 8 kann nach unten verdrahtet werden, Netz 3 kann nach oben verdrahtet werden, Netz 3 muss in die Spur 4, daher Verlegung in die Spur 4 (nach unten), durch Fortführung in den Spuren 1, 2, 3, 4, 5 und 6 können alle Netze abschließend verdrahtet werden

Zusammenfassung

- ▶ Platzierung:
 - ▶ Kräfteplatzierung
 - ▶ Weitere Verfahren
- ▶ Verdrahtung:
 - ▶ Einteilung der Algorithmen
 - ▶ Globale, detaillierte und Flächenverdrahtung
 - ▶ Bewertung und Kostenfunktionen
 - ▶ Ausgewählte Algorithmen

Rechnergestützer Entwurf digitaler Systeme:

Vorlesungs-Gesamtübersicht

1. Einleitung und grundlegende Konzepte
2. Synthese und Technologie-Abbildung
3. Partitionierung und Floorplanning
4. Platzierung und Verdrahtung
5. Testung

Rechnergestützer Entwurf digitaler Systeme:

Gliederung der 12. Vorlesung

4. Platzierung und Verdrahtung

4.1. Platzierung

- 4.1.1. Min-Cut-Platzierung
- 4.1.2. Kräfteplatzierung
- 4.1.3. Weitere Verfahren

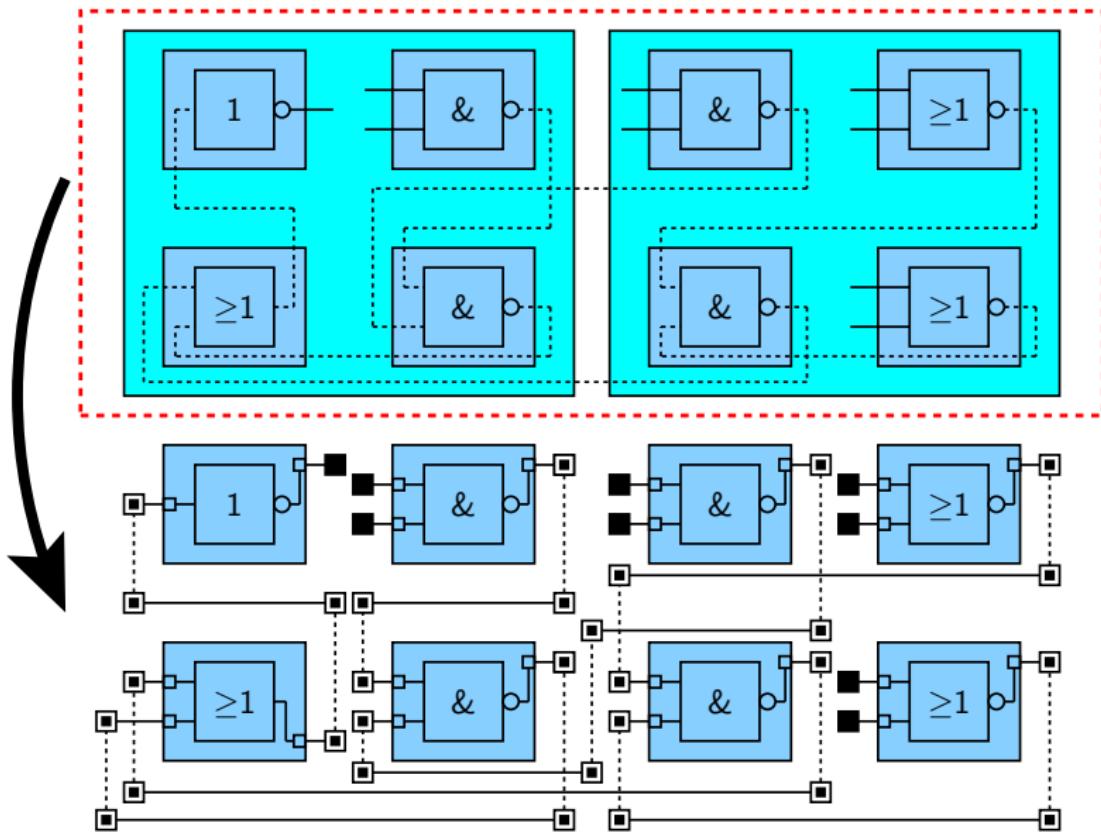
4.2. Verdrahtung

- 4.2.1. Globalverdrahtung
- 4.2.2. Feinverdrahtung
- 4.2.3. Flächenverdrahtung

5. Testung

- 5.1. Einleitung und Grundbegriffe
- 5.2. Fehlermodelle
- 5.3. Testmustergenerierung
 - 5.3.1. BOOLEsche Differenz
 - 5.3.2. D-Algorithmus

Verdrahtung in einem Schritt



Ziele der Flächenverdrahtung

Hauptziel:

- ▶ Herstellung von elektrisch und technologisch gültigen Verbindungen aller Netze entsprechend den Vorgaben der Netzliste

Weitere Optimierungsziele:

- ▶ Minimierung der Gesamtverbindungslänge
- ▶ Minimierung der Verbindungslänge des längsten Netzes
- ▶ Minimierung der Anzahl von Durchkontaktierungen (Vias)
- ▶ Minimierung der Verdrahtungsfläche und der Anzahl der benötigten Verdrahtungsebenen
- ▶ Minimierung der Anzahl von Leitungsknicken
- ▶ Gleichverteilung der Verdrahtungsdichte
- ▶ Vermeidung von Kopplungen und Übersprechen zwischen benachbarten Leitungen

Einteilung der Algorithmen

Raster- bzw. Labyrinthverdrahtung (*grid and maze routing*): Suche der kürzesten Verbindungen in einem Raster

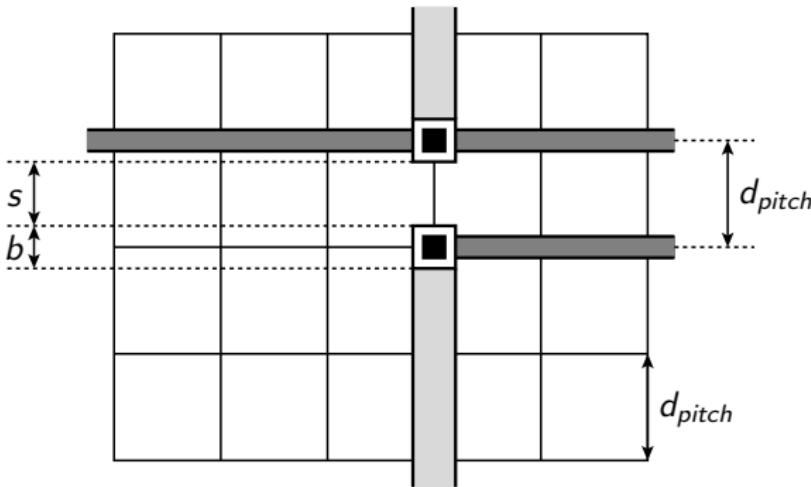
Linienverdrahtung (*line probe routing*): Suche von Schnittpunkten von Strahlen bzw. Geraden

Rasterfreie Verdrahtung (*shape based routing*): Berücksichtigung realer Formen und Abmessungen von Verdrahtungsbahnen

Wesentliche Probleme bzw. Unterscheidungskriterien von Algorithmen:

- ▶ Abarbeitungsreihenfolge von Netzen
- ▶ Abarbeitungsreihenfolge von Anschlüssen eines Netzes
- ➡ Beide Kriterien können einen entscheidenden Einfluss auf die Qualität des Endergebnisses haben.

Rastererzeugung



- b** ist die minimale Breite des Leiterzugs bzw. der Durchkontaktierung auf einer Ebene
- s** ist der minimale Abstand zwischen zwei Elementen auf einer Ebene

$$d_{pitch} = b + s \cong \text{minimaler Gitterabstand}$$

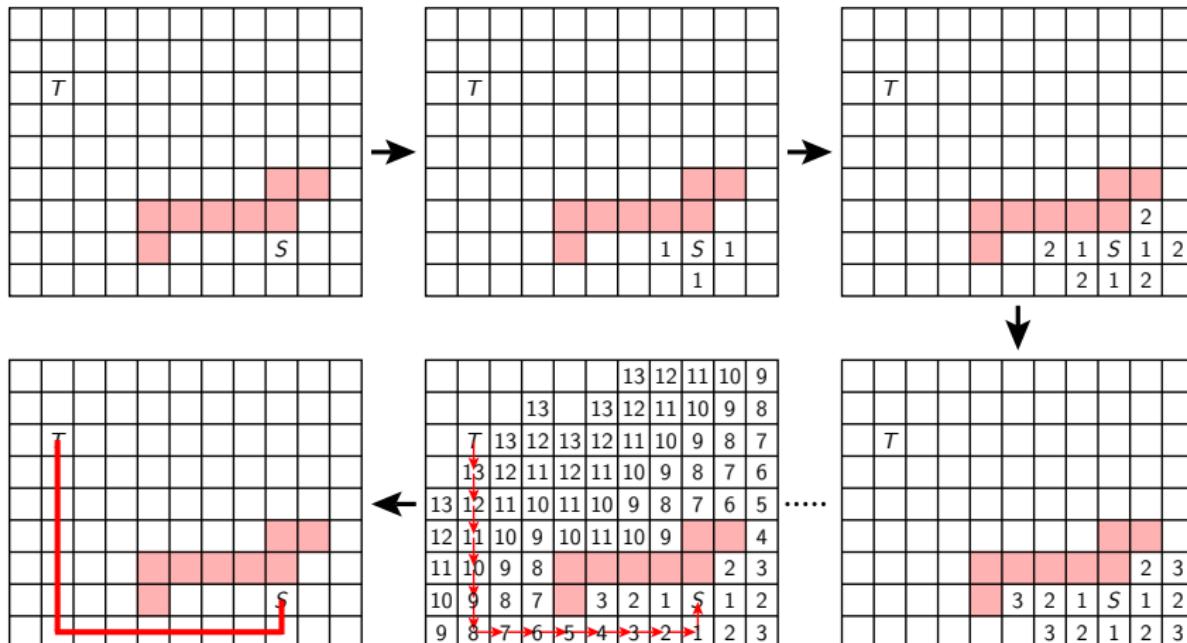
Ablauf des LEE-Algorithmus (*maze routing*)

Ausbreitungsphase (*wave propagation phase*): Sukzessive Indizierung aller benachbarten Punkte mit aufsteigenden Indizes (unmittelbare Nachbarn \cong Index 1, Nachbarn der unmittelbaren Nachbarn \cong Index 2 usw.) ausgehend von der Quelle bis die Senke erreicht ist.

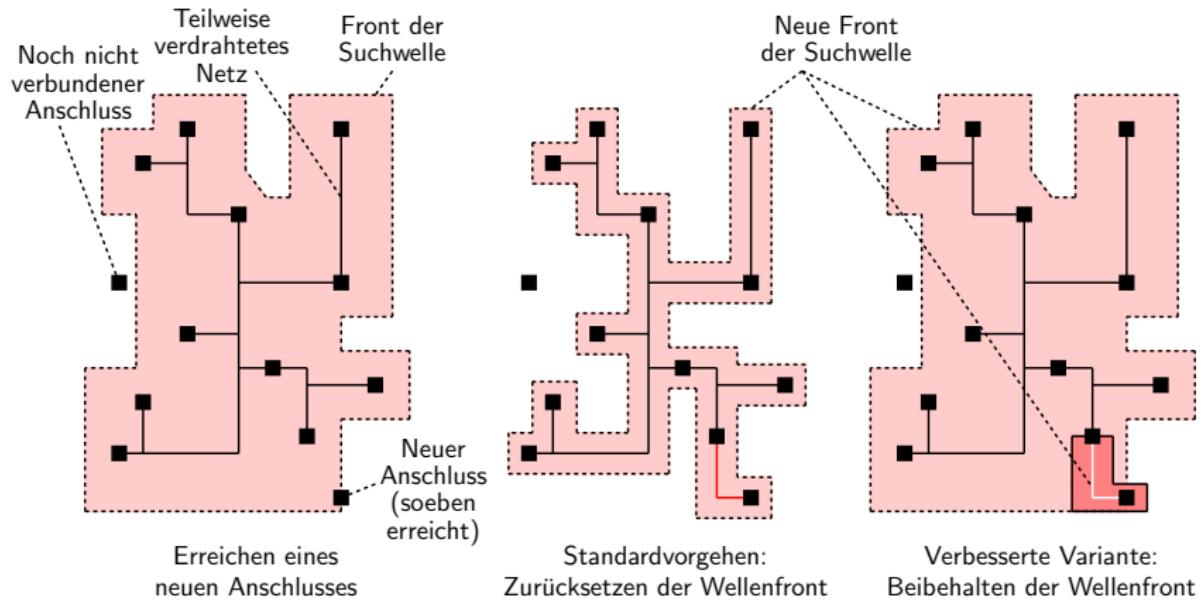
Rückverfolgungsphase (*retrace phase*): Rückverfolgung der Wellenfront von der Senke zur Quelle in Richtung absteigender Indizes unter Berücksichtigung der kürzesten Verbindung, wobei bei mehreren gleich langen Pfaden der mit den wenigsten Richtungswechseln bevorzugt wird.

Aufräumphase (*label clearance phase*): Rasterpunkte des gefundenen Weges als belegt markieren, alle Indizierungen löschen.

Rasterverdrahtung (am Beispiel von Maze-Algorithmus)

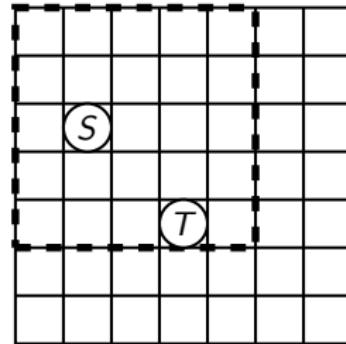
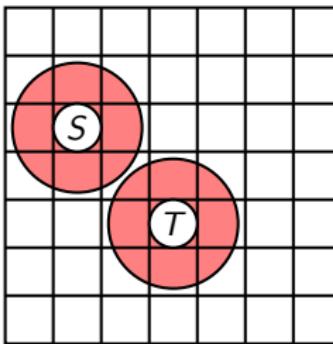
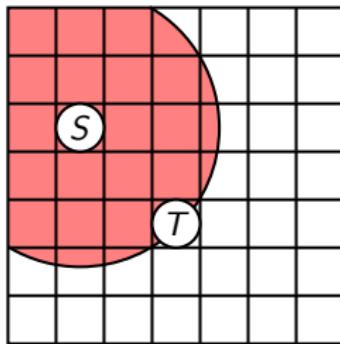


Erweiterung von Maze-Verdrahtung auf Netze mit mehr als 2 Anschlüssen



Das Beibehalten einer alten Wellenfront erlaubt unter Umständen ein schnelleres Finden neuer Anschlüsse.

Weitere Verbesserungen des Maze-Algorithmus

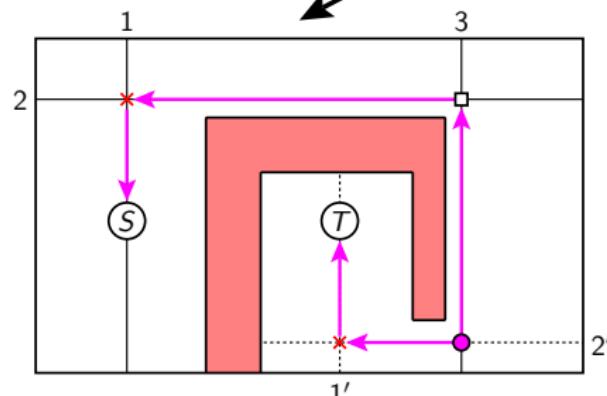
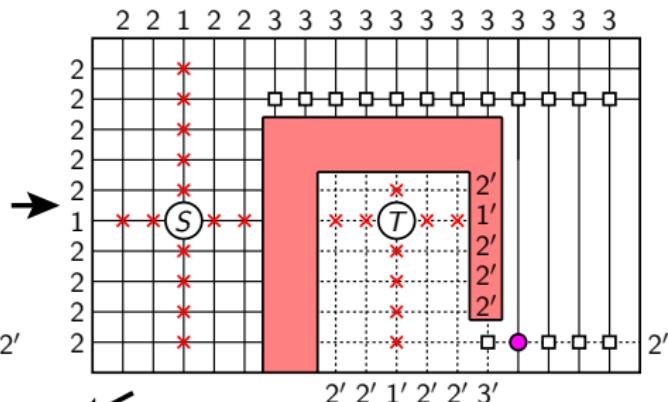
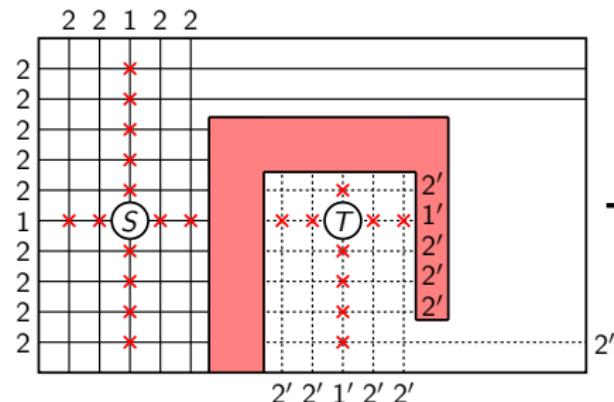


- ▶ Wahl eines Anschlusses in der Nähe des Randes
- ▶ Gleichzeitige Suche ausgehend von zwei zu verbindenden Anschlüssen
- ▶ Einschränkung des Suchraums (mit späterer Erweiterung oder Aufhebung, falls keine Lösung gefunden wird)

Ablauf des MIKAMI-TABUCHI-Algorithmus (Linienverdrahtung)

1. Erzeugen von Expansionslinien 1 bzw. 1', die jeweils an der Quelle und Senke starten und am nächsten Hindernis oder am Chiprand enden. Setzen $j = 1$ und $j' = 1'$.
2. Falls Geraden j und j' oder j und $(j - 1)'$ oder $(j - 1)$ und j' sich schneiden, Rückverfolgung auf den Geraden j und j' , $(j - 1)$ und $(j - 1)'$, $(j - 2)$ und $(j - 2)'$ usw. ausgehend vom Schnittpunkt zur Quelle und Senke (ENDE). Ansonsten weiter mit Schritt 3.
3. Erzeugen von Versuchslinien $(j + 1)$ bzw. $(j + 1)'$ senkrecht zu j bzw. j' in allen Ausgangspunkten auf j bzw. j' . Setzen $j = j + 1$, $j' = j' + 1$.
4. Weiter mit Schritt 2.

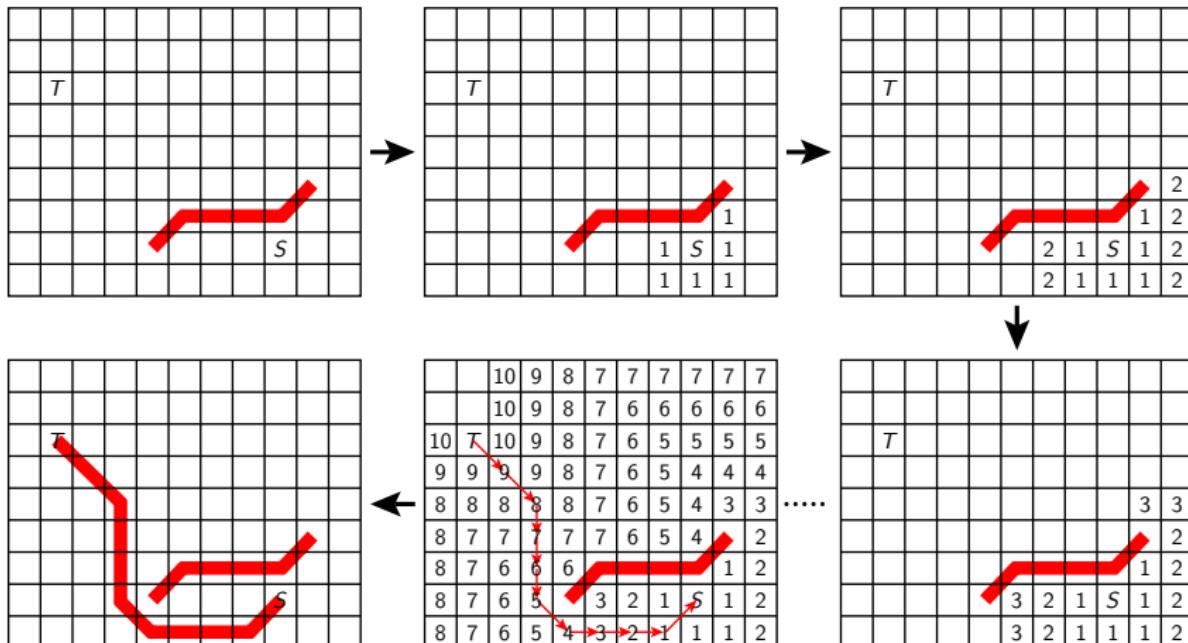
Linienverdrahtung (MIKAMI-TABUCHI-Algorithmus)



Weitere Aspekte (hier nicht betrachtet)

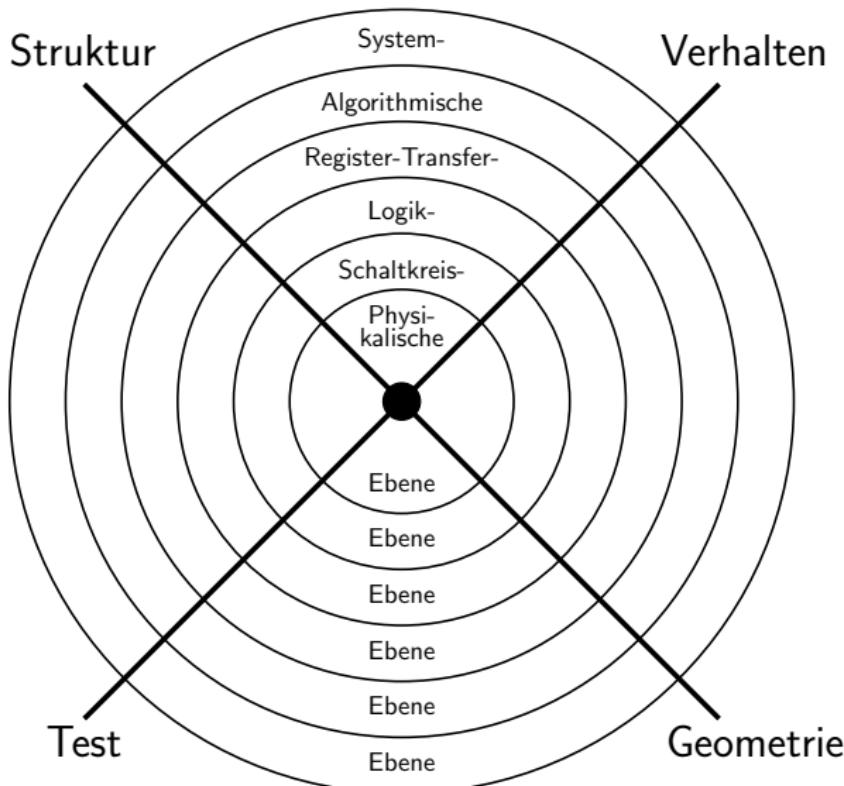
- ▶ Dreidimensionale Verdrahtung (insbesondere bei Multi-Chip-Modulen, Leiterplatten, zunehmend auch bei einzelnen Schaltkreisen)
 - ▶ Rasterverdrahtung
 - ▶ Mehrstufen-Verdrahtung
 - ▶ Planarverdrahtung
 - ▶ Turmverdrahtung
- ▶ Nicht-orthogonale Verdrahtungstopologien (Y- bzw. X-Verdrahtung)
 - ▶ Oktilineare STEINER-Bäume (globale Verdrahtung)
 - ▶ Oktilineare Wegsuche (Feinverdrahtung, z. B. modifizierter LEE-Algorithmus)

Oktilineare Wegsuche (erweiterter LEE-Algorithmus)



5. Testung

Zur Erinnerung: Testung im X-Diagramm



Zum Begriff der Testung

Allgemein: Alle Schritte, die zur Überprüfung der Übereinstimmung zwischen Ist- und Soll-Verhalten dienen, d. h.

Schaltungssimulation, Timing-Verifikation, Formale

Verifikation, *property-based design and test*,

Layout-Verifikation, *design rule check* (DRC), *electrical rule check* (ERC) usw.

- Solange die integrierte Schaltung noch nicht gefertigt ist, spricht man von **Verifikation**. Zielstellung: Entdeckung von Entwurfsfehlern

Speziell: Alle Schritte, die zur Trennung der fehlerfreien von fehlerhaften Chips dienen

- Liegt die integrierte Schaltung bereits als gefertigtes Produkt vor, spricht man vom **Test**. Zielstellung: Entdeckung von Fertigungsfehlern

Verifikation, Validierung und Testung

Verifikation: Ein Entwurfsschritt, der auf einer Entwurfsebene weg vom Entwurfsziel verläuft (und vor der Fertigung erfolgt).

Validierung: Ein Entwurfsschritt, der den Entwurfszustand vom Entwurfsziel entfernt und dabei eine Grenze zwischen den Ebenen in aufsteigender Richtung kreuzt (Validierung eines Syntheseschrittes, erfolgt vor der Fertigung). Ist nur dann zwingend notwendig, wenn der Syntheseschritt manuell oder durch ein nicht formal verifiziertes Verfahren durchgeführt wurde. Ansonsten ist die Gültigkeit des Schrittes durch „entwurfsbedingte Korrektheit“ (*correctness by design, correctness by construction*) sichergestellt.

Test: Ein Postproduktionsschritt zum Endecken der Produktionsfehler.

Einteilung der Fehler

Permanente Fehler: Zu jedem Zeitpunkt des Testes vorhanden, somit relativ leicht zu entdecken bzw. einzukreisen (z. B. Kurzschlüsse). Erkennung: „vollständiger“ Funktionstest.

Intermittierende Fehler (auch als transiente Fehler bezeichnet): Sporadisch auftretende Fehler, somit nicht ohne Weiteres reproduzierbar und schwer einzukreisen (z. B. Funktionsstörungen durch Strahlungseinfluss). Erkennung: exzessiver Test mit mehrfachen Wiederholungen unter verschiedenen Umgebungsbedingungen, *burn-in*.

Je genauer der Verlauf der Ausfallkurve bekannt ist (➡ statistische Untersuchungen), umso mehr lässt sich die Zuverlässigkeit des Endproduktes steigern („Eliminieren“ der Frühausfallphase).

Testung und Kosten

Testen ist teuer:

- ▶ Ausrüstung
- ▶ Arbeitszeit
- ▶ Qualitätssicherungskonzept

Nichttesten ist teuer:

- ▶ Wertminderung, Reparaturkosten
- ▶ Garantiefälle, Rückgaben und Vertragsstrafen
- ▶ Haftung für Folgeschäden

Sowohl wirtschaftliche als auch technische Erwägungen
(Sicherheitsklassen):

- ▶ höchste: Personenbeförderung, Energieversorgung
- ▶ hohe: „gewöhnliche“ Prozesssteuerung
- ▶ niedrige: Konsumgüter

Fehlerdiagnosekonzepte

Off-line (Verifikation und Testung): Beseitigung von Fehlerursachen vor der Fertigung bzw. von fehlerhaften Produkten vor dem Vertrieb

Fail safe (Fehlererkennung im laufenden Betrieb): Ein Fehlverhalten hat definierte (akzeptable) Folgen

Fehlertoleranz (Fehlererkennung und Beseitigung im laufenden Betrieb): Ein Fehler führt nicht zum Ausfall

Besonderheit bei digitalen Systemen

Die Anzahl der möglichen Eingangsbelegungen und Zustände ist enorm groß

- ▶ Testen durch „vollständiges Durchprobieren“ ist selbst bei kleineren Schaltungen nicht möglich (z. B. 32-Bit Addierer
➡ 2^{64} mögliche Eingangsbelegungen).
- ▶ Es werden intelligente, strukturierte Testverfahren benötigt.

Hintergrund

Testen durch vollständige Ausschöpfung aller Eingangsbelegungen (unter evtl. Berücksichtigung aller möglichen internen Zustände) ist nicht durchführbar.

- ➡ Systematische Fehlersuche durch gezieltes Anlegen ausgewählter Testsätze

Bevor eine strukturierte Erzeugung der Testsätze zur Erkennung von Fehlern erfolgen kann, muss festgelegt werden, welches Fehlverhalten detektiert werden soll

- ➡ Fehlermodelle, z. B.
 - ▶ Haftfehler
 - ▶ allgemeine Logikfehler
 - ▶ dynamische Fehler

Haftfehlermodell

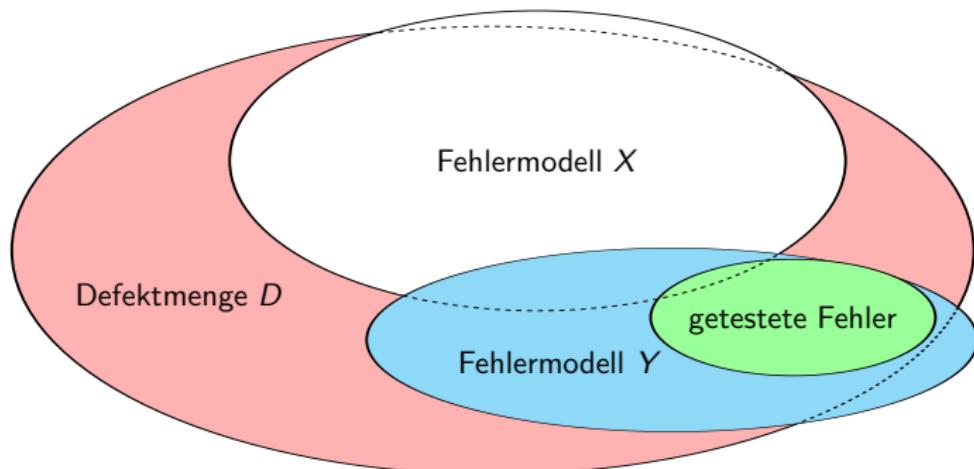
Haftfehlermodell (*stuck-at*-Fehlermodell)

ist die einfachste und die am weitesten verbreitete Annahme für die Testauswahl, die wie folgt definiert ist:

- ▶ ein Schaltungspunkt führt ständig Null (*stuck-at-0*, sa0) bzw.
- ▶ ein Schaltungspunkt führt ständig Eins (*stuck-at-1*, sa1)

Beim **Haft-Einzelfehlermodell** wird jeweils genau eine Leitung als fehlerhaft angenommen, d. h. es gibt $2 \cdot n$ Fehler bei n Leitungen. Ein vollständiger Test muss jeden dieser Fehler nachweisen können. Weiterhin sind **Haft-Mehrfachfehler** definiert, z. B. Zweifach- oder Dreifachfehler, bei denen jeweils mehrere Leitungen als festklemmend angenommen werden. Die Anzahl der möglichen Fehler steigt dabei signifikant auf $3^n - 1$ (somit auch der Aufwand für die Testsatzerzeugung). Die Vollständigkeit eines Testes wird mit Fehlerüberdeckung (Anzahl der getesteten Fehler im Verhältnis zu allen durch das Fehlermodell erfassten Fehler) gemessen.

Defektüberdeckung vs. Fehlerüberdeckung

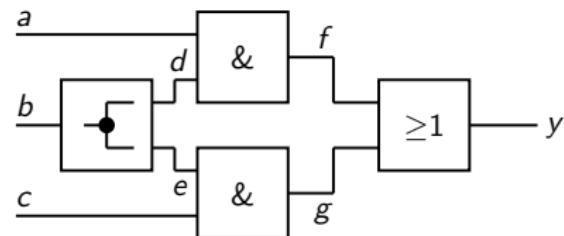
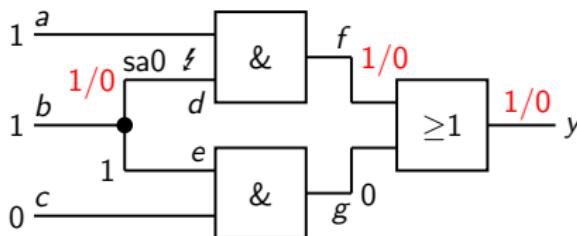


Defektüberdeckung ist das Verhältnis der Anzahl von getesteten Fehlern zur Anzahl aller möglichen Defekte des Chips

Fehlerüberdeckung ist das Verhältnis der Anzahl von getesteten Fehlern zur Anzahl aller durch ein Fehlermodell erfassten Fehler.

- Da ein Fehlermodell immer eine Vereinfachung darstellt, garantiert eine 100 % Fehlerüberdeckung keine Fehlerfreiheit.

Veranschaulichung des Haftfehlermodells



Den Fehler (die Fehlerwirkung, in dem Fall $sa0$ auf Leitung d) kann man beobachten, wenn man die betroffene Leitung auf 1 setzt. Dieser Vorgang heißt „Simulieren des Fehlers“ und lässt sich im skizzierten Fall durch die Belegung $b = 1$ bewerkstelligen. Einen Erfolg hat man dabei nur, wenn das Fehlverhalten auch am Ausgang sichtbar ist (\cong der Fehler wird zum Ausgang propagiert), d. h. es ist erforderlich, die Eingangsbelegungen $a = 1$ und $c = 0$ sicher zu stellen.

- $(a, b, c) = (1, 1, 0)$ ist ein geeignetes Testmuster zum Nachweis des Fehlers $sa0$ auf Leitung d .

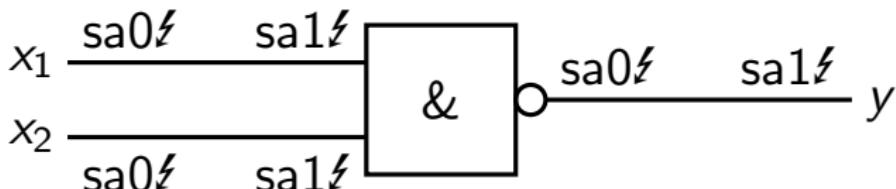
Fehleräquivalenz, Fehlerdominanz und Fehlerredundanz

Fehleräquivalenz: Zwei Fehler sind äquivalent, wenn sie dasselbe Fehlverhalten verursachen und mit der gleichen Menge von Testvektoren nachweisbar sind. Für die Fehlersimulation und Testsatzgenerierung ist es ausreichend, jeweils nur einen Repräsentanten einer Äquivalenzklasse zu berücksichtigen.

Fehlerdominanz: Ein Fehler wird von den anderen Fehlern dominiert, wenn er mit allen Tests dieser Fehler (sowie ggf. weiteren Tests) nachgewiesen werden kann. Für einen dominierenden Fehler kann die Testmustersuche entfallen, wenn sichergestellt ist, dass Testmuster von mindestens einem ihn dominierenden Fehler verwendet werden.

Fehlerredundanz: Einige angenommene Fehler können unter Umständen niemals sichtbar werden (keine Beeinträchtigung der Schaltungsfunktion bzw. Ausschluss der Fehlereintrittsbedingungen). Solche Fehler sind redundant und können aus der Fehlermenge entfernt werden.

Beispiele von äquivalenten und dominierenden Fehlern

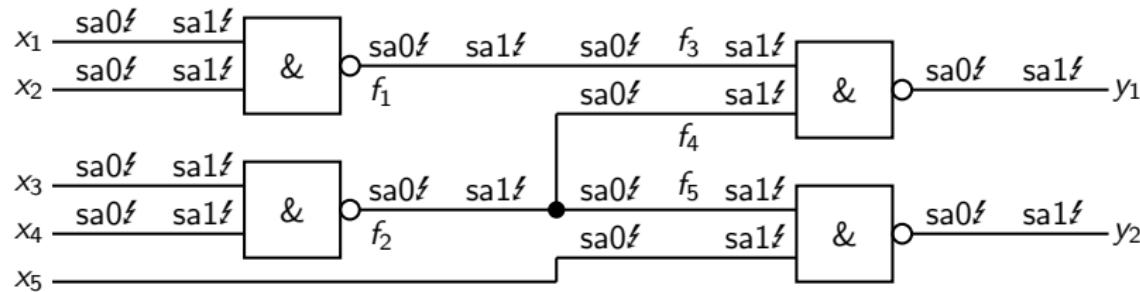


x_1	x_2	$\overline{x_1 \wedge x_2}$	x_1-sa0	x_1-sa1	x_2-sa0	x_2-sa1	$y-sa0$	$y-sa1$
0	0	1	1	1	1	1	0	1
0	1	1	1	0	1	1	0	1
1	0	1	1	1	1	0	0	1
1	1	0	1	0	1	0	0	1

- Fehler x_1-sa0 , x_2-sa0 und $y-sa1$ sind äquivalent.
- Fehler $y-sa0$ wird von den Fehlern x_1-sa1 und x_2-sa1 dominiert.

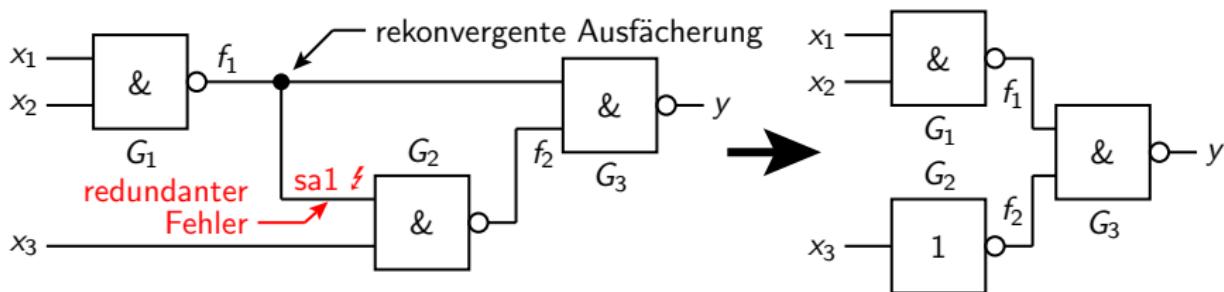
Reduzierung der Haftfehlermenge

Ausgangsposition: 24 Fehler. Durch Ausnutzen von Äquivalenzen wird die Fehleranzahl auf 14 und mit Dominanzen sogar auf 9 reduziert:



Fehleräquivalenzklassen	dominiert von	Fehleräquivalenzklassen	dominiert von
1 $x_1\text{-sa}0, x_2\text{-sa}0, f_1\text{-sa}1, f_3\text{-sa}1$		8 $f_2\text{-sa}0$	4, 6, 7, 11
2 $x_1\text{-sa}1$		9 $f_4\text{-sa}1$	
3 $x_2\text{-sa}1$		10 $y_1\text{-sa}0$	1, 4
4 $f_1\text{-sa}0, f_3\text{-sa}0, f_4\text{-sa}0, y_1\text{-sa}1$	2, 3	11 $f_5\text{-sa}0, x_5\text{-sa}0, y_2\text{-sa}1$	
5 $x_3\text{-sa}0, x_4\text{-sa}0, f_2\text{-sa}1$	9, 12	12 $f_5\text{-sa}1$	
6 $x_3\text{-sa}1$		13 $x_5\text{-sa}1$	
7 $x_4\text{-sa}1$		14 $y_2\text{-sa}0$	12, 13

Beispiel eines redundanten Fehlers



Durch Zusammenführen der Ausfächerung sind beide Eingänge des Gatters G_3 voneinander funktional abhängig, womit die Möglichkeiten der Anregung und Beobachtung interner Fehler (in diesem Fall f_1 -sa1) eingeschränkt werden. Zum Nachweis des angegebenen Fehlers muss Knoten f_1 Null führen (zur Beobachtung am Ausgang von G_2 muss weiterhin $x_3 = 1$ gelten). Zur Beobachtung des Fehlers am Ausgang y muss f_1 Eins führen

⇒ Widerspruch, d. h. der Fehler ist logisch nicht nachweisbar (Hinweis auf redundante Schaltungselemente, Optimierungspotential).

Zellenfehlermodell

Beim Zellenfehlermodell wird davon ausgegangen, dass Fehler sich immer in einem falschen kombinatorischen Verhalten einer Zelle (als Basisbildungsblock einer Schaltung, im einfachsten Fall ein Gatter) äußern und nicht auf Verbindungsstrukturen auswirken, z. B.

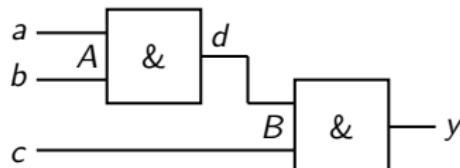
		korrekt	fehlerhaft
<i>a</i>	<i>b</i>	<i>y</i>	<i>y</i>
0	0	1	0
0	1	1	1
1	0	1	1
1	1	0	0

als Beispiel eines
allgemeinen Logikfehlers

Zur vollständigen Fehlerüberdeckung muss jede Zelle mit allen Eingangskombinationen getestet werden $\cong \sum_{i=1}^{n_G} 2^{e_i}$ mit n_G als Anzahl der Zellen und e_i als Anzahl der Eingänge der jeweiligen Zelle.

Beispiele von vollständigen Testmustern

Für Haft- und Zellenfehler einer einfachen Schaltung:

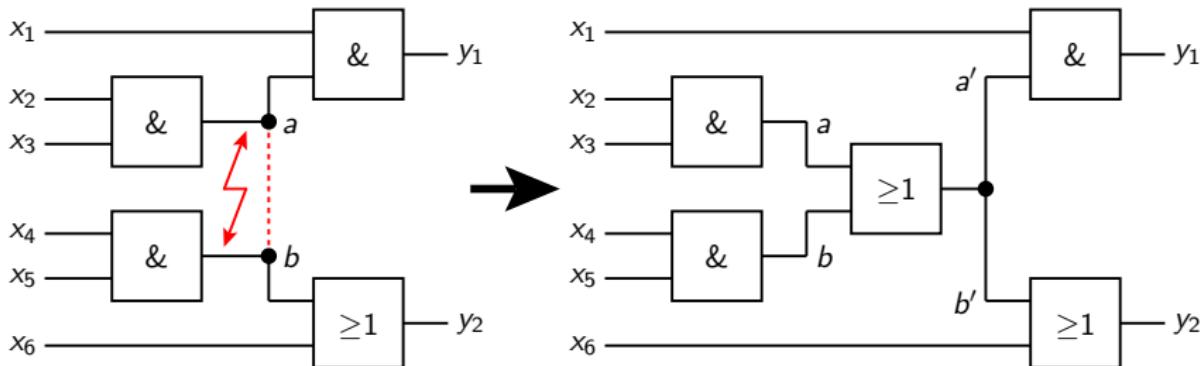


a	b	c	Nachweisbare Haftfehler
1	1	1	$a \text{ sa0}, b \text{ sa0}, c \text{ sa0}, d \text{ sa0}, y \text{ sa0}$
0	1	1	$a \text{ sa1}, d \text{ sa1}, y \text{ sa1}$
1	0	1	$b \text{ sa1}$
1	1	0	$c \text{ sa1}$

a	b	c	Nachweisbare Zellenfehler
0	0	1	(0,0) für Zelle A, (0,1) für Zelle B
0	1	1	(0,1) für Zelle A
1	0	1	(1,0) für Zelle A
1	1	1	(1,1) für Zelle A, (1,1) für Zelle B
0	0	0	(0,0) für Zelle B
1	1	0	(1,0) für Zelle B

Kurzschlussfehlermodell

Das Kurzschlussfehlermodell (*Bridging-Fehlermodell*) bildet Leitungskurzschlüsse nach (hier als OR-Fehler modelliert):



Abhängig von den Pegeln, die an den beiden Leitungen anliegen, führen die Kurzschlüsse zum komplexen Fehlerverhalten (bei gleichen Pegeln macht sich der Fehler nicht bemerkbar).

- ➡ Mit $n \cdot (n - 1)$ bei n Leitungen steigt die Anzahl der Fehler quadratisch. Unter Ausnutzung der Layout-Daten kann die Betrachtung auf benachbarte Leitungen eingeschränkt werden.

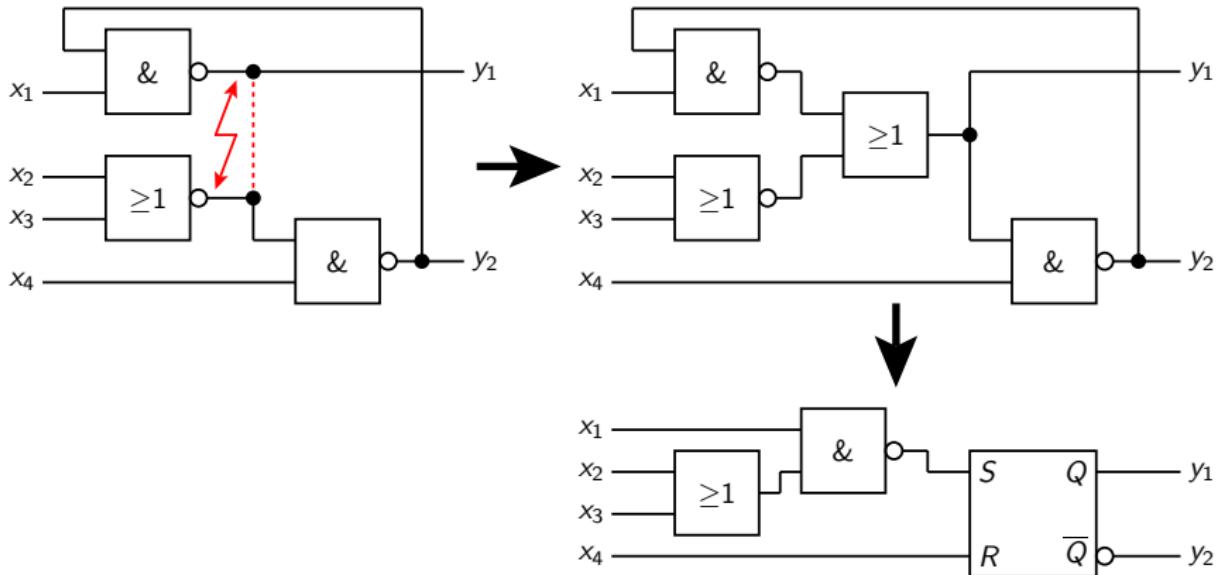
Verhalten der Schaltung bei Kurzschläüssen

Nachbildung von Kurzschlussfehlern ist abhängig von den Treibern der kurzgeschlossenen Leitungen:

- ▶ Sind die Impedanzen beider Treiber im Nullzustand niedriger als im Eins-Zustand, so kann der entsprechende Kurzschlussfehler mit UND-Verknüpfung modelliert werden.
- ▶ Sind die Impedanzen beider Treiber im Nullzustand höher als im Eins-Zustand, so kann der entsprechende Kurzschlussfehler mit OR-Verknüpfung modelliert werden.
- ▶ Besitzt eine der Leitungen einen stärkeren Treiber, so setzt sich ihr Pegel auf beiden Leitungen durch.
- ▶ Kurzschluss von Leitungen gleicher Impedanz erzeugt in der Regel eine Spannung im verbotenen Bereich.

Zusätzliches Problem: Unerwünschte Signalrückführungen durch Kurzschlüsse ➔ Kombinatorische Schleifen, Speicher- oder Oszillationsverhalten

Auswirkung eines Kurzschlusses in Form eines Speicherelements



Ein Kurzschluss ist sicher nachweisbar, wenn beide Leitungen unterschiedliche Pegel führen und gleichzeitig beobachtbar sind.

Dynamische Fehler

Als dynamische Fehler werden solche Fehler bezeichnet, die das Zeitverhalten einer Schaltung beeinflussen:

Gatterverzögerungsfehlermodell kann als dynamisches

Haftfehlermodell interpretiert werden: Für jeden Eingang eines logischen Gatters wird unterstellt, dass bei steigender oder fallender Flanke ein temporärer Haftfehler auftritt (was sich in verlängerter Verzögerungszeit eines Gatters äußert).

Pfadfehlermodell ist eine Erweiterung des

Gatterverzögerungsfehlermodells (das nur lokale Fehlerwirkungen abbildet) auf pfadbezogene Signallaufzeiten.

Datenhaltefehlermodell bildet eine unerwünschte (eigenständige)

Veränderung des Speicherzustandes ab, z. B. in DRAM-Zellen bei fehlender Auffrischung bzw. auch in SRAM durch Leitungsunterbrechungen.

Was bedeutet „Testmustergenerierung“?

Wenn ein Fehler an den Ausgängen der Schaltung beobachtbar ist, muss für die Eingänge der Schaltung eine entsprechende Belegung (Testvektor) generiert werden, mit der dieser Fehler nachgewiesen werden kann. Im folgenden werden Verfahren beschrieben, die eine automatische Testmustergenerierung (*automatic test pattern generation, ATPG*) ermöglichen.

Aufgabenstellung: Erzeugung von Testvektoren zum Nachweis aller (nachweisbaren) angenommenen Fehler (bzw. einer bestimmten Anzahl von angenommenen Fehlern entsprechend der gewünschten Fehlerüberdeckung). Einschränkung: Rein kombinatorische Schaltungen mit Haftfehlermodell. Als Verfahren zur Testmustergenerierung werden

- ▶ Boolesche Differenz und
- ▶ D-Algorithmus

betrachtet.

Schaltalgebraischer Hintergrund

Mit Hilfe der BOOLEschen Differenz $\frac{\partial f}{\partial x_i}$ von Funktion f nach Argument x_i kann ermittelt werden, ob eine Änderung der Belegung von x_i zu einer Änderung des Funktionswertes

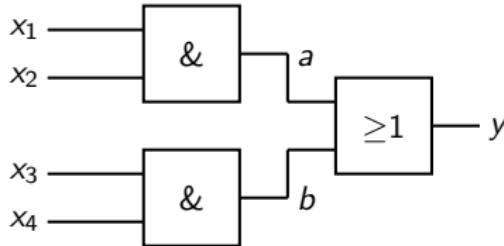
$y = f(x_1, x_2, \dots, x_i, \dots, x_n)$ führt. Diese ist wie folgt definiert:

$$\begin{aligned}\frac{\partial f}{\partial x_i} &= f(x_1, x_2, \dots, x_{i-1}, 0, x_{i+1}, x_{i+2}, \dots, x_n) \\ &\oplus f(x_1, x_2, \dots, x_{i-1}, 1, x_{i+1}, x_{i+2}, \dots, x_n)\end{aligned}$$

Gilt $\frac{\partial f}{\partial x_i} = 1$, so hängt der Funktionswert $f(x_1, x_2, \dots, x_i)$ von x_i ab, ansonsten hängt der Funktionswert $f(x_1, x_2, \dots, x_i)$ nicht von x_i ab.

→ Hilfestellung zur Bestimmung der Testvektoren bei angenommener Belegung mit einem Haftfehler.

Beispiel



$$y = f(x_1, x_2, x_3, x_4) = x_1 x_2 \vee x_3 x_4$$

$$\begin{aligned} \frac{\partial f}{\partial x_1} &= f(0, x_2, x_3, x_4) \oplus f(1, x_2, x_3, x_4) = ((0 \wedge x_2) \vee x_3 x_4) \oplus ((1 \\ &\wedge x_2) \vee x_3 x_4) = x_3 x_4 \oplus (x_2 \vee x_3 x_4) = x_2 \overline{x_3 x_4} = x_2 \wedge (\overline{x_3} \vee \overline{x_4}) \end{aligned}$$

Aus der Bedingung $\frac{\partial f}{\partial x_1} = 1$ folgt, dass f genau dann von x_1 abhängt, wenn $x_2 \wedge (\overline{x_3} \vee \overline{x_4}) = 1$ gilt.

- Bedingung für die Sensibilisierung des Ausgangs (Beobachtbarkeit).

Bedingungen für eine erfolgreiche Testmustererzeugung

Zusätzlich müssen die bereits eingeführten Bedingungen für die Fehlersimulation gelten:

- ▶ Bei Annahme von sa0 die betroffene Leitung auf Eins setzen
- ▶ bei Annahme von sa1 die betroffene Leitung auf Null setzen

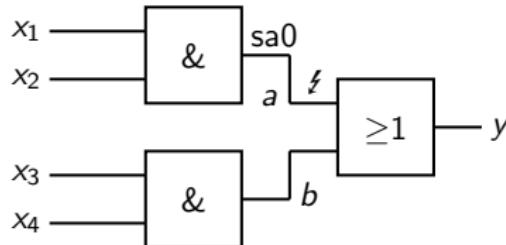
Somit können die Bestimmungsgleichungen der Testmuster für entsprechende Haftfehler abgeleitet werden:

$$\text{sa0 auf Leitung } x: \quad x \wedge \frac{\partial f}{\partial x} = 1$$

$$\text{sa1 auf Leitung } x: \quad \bar{x} \wedge \frac{\partial f}{\partial x} = 1$$

Lösungen der Gleichungen sind Testvektoren, die entsprechende Haftfehler nachweisen.

Beispiel (1)



Testmuster zum Nachweis des $a = sa0$ Fehlers.

$$y = a \vee x_3 x_4 \text{ mit } a = x_1 x_2$$

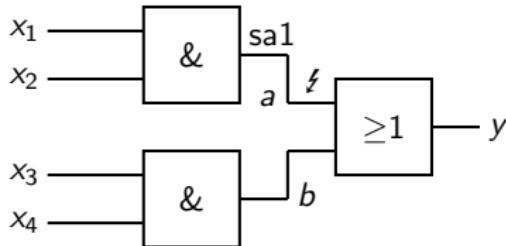
$$\begin{aligned} a \wedge \frac{\partial y}{\partial a} &= x_1 x_2 \wedge ((0 \vee x_3 x_4) \oplus (1 \vee x_3 x_4)) = x_1 x_2 \wedge (x_3 x_4 \oplus 1) \\ &= x_1 x_2 \wedge \overline{x_3 x_4} = x_1 x_2 \overline{x_3} \vee x_1 x_2 \overline{x_4}. \end{aligned}$$

$$a \wedge \frac{\partial y}{\partial a} = 1 \implies x_1 x_2 \overline{x_3} \vee x_1 x_2 \overline{x_4} = 1$$

→ alle zum Nachweis des Fehlers geeigneten Testmuster:

$$(x_1, x_2, x_3, x_4) = (1, 1, 0, 0), (x_1, x_2, x_3, x_4) = (1, 1, 0, 1) \text{ sowie} \\ (x_1, x_2, x_3, x_4) = (1, 1, 1, 0).$$

Beispiel (2)



Testmuster zum Nachweis des $a = sa1$ Fehlers.

$$y = a \vee x_3 x_4 \text{ mit } a = x_1 x_2$$

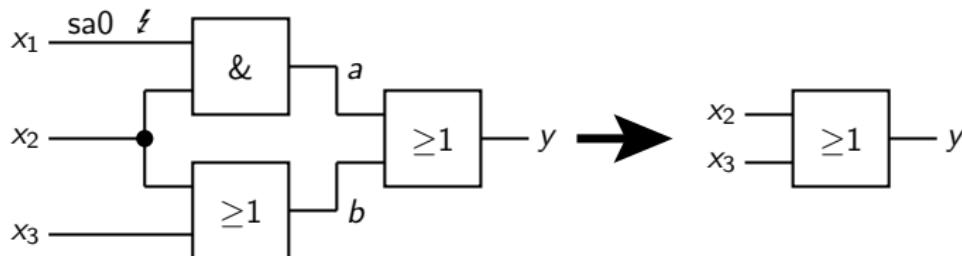
$$\begin{aligned} \bar{a} \wedge \frac{\partial y}{\partial a} &= \bar{x}_1 \bar{x}_2 \wedge (x_3 x_4 \oplus 1) = \bar{x}_1 \bar{x}_2 \wedge \bar{x}_3 \bar{x}_4 \\ &= (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_3 \vee \bar{x}_4) = \bar{x}_1 \bar{x}_3 \vee \bar{x}_1 \bar{x}_4 \vee \bar{x}_2 \bar{x}_3 \vee \bar{x}_2 \bar{x}_4. \end{aligned}$$

$$\bar{a} \wedge \frac{\partial y}{\partial a} = 1 \implies \bar{x}_1 \bar{x}_3 \vee \bar{x}_1 \bar{x}_4 \vee \bar{x}_2 \bar{x}_3 \vee \bar{x}_2 \bar{x}_4 = 1$$

→ alle zum Nachweis des Fehlers geeigneten Testmuster:

$$(x_1, x_2, x_3, x_4) = (0, -, 0, -), (0, -, -, 0), (-, 0, 0, -), (-, 0, -, 0).$$

Berechnung für einen redundanten Fehler



$$y = x_1 x_2 \vee (x_2 \vee x_3)$$

$$\begin{aligned} x_1 \wedge \frac{\partial y}{\partial x_1} &= x_1 \wedge (((0 \wedge x_2) \vee (x_2 \vee x_3)) \oplus ((1 \wedge x_2) \vee (x_2 \vee x_3))) = \\ &= x_1 \wedge ((x_2 \vee x_3) \oplus (x_2 \vee x_3)) = x_1 \wedge 0 = 0. \end{aligned}$$

$$x_1 \wedge \frac{\partial y}{\partial x_1} = 1 \text{ ist nicht erfüllbar.}$$

→ Der Fehler $x_1\text{-sa0}$ ist redundant.

Erweiterung auf Schaltungen mit mehreren Ausgängen

Bei einer Schaltung mit m Ausgängen y_1, \dots, y_m reicht es, wenn der Fehler an mindestens einem der Ausgänge beobachtbar ist.

Gleichung für einen sa0-Fehler an der Leitung a :

$$a \wedge \left(\frac{\partial y_1}{\partial a} \vee \frac{\partial y_2}{\partial a} \vee \dots \vee \frac{\partial y_m}{\partial a} \right) = 1$$

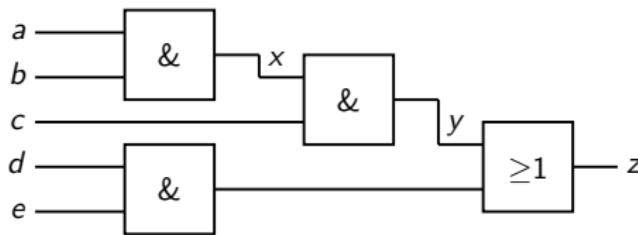
Gleichung für einen sa1-Fehler an der Leitung a :

$$\overline{a} \wedge \left(\frac{\partial y_1}{\partial a} \vee \frac{\partial y_2}{\partial a} \vee \dots \vee \frac{\partial y_m}{\partial a} \right) = 1$$

Außerdem gilt für BOOLEsche Differenzen die Kettenregel (falls es einen Pfad von x nach z gibt, auf dem y liegt):

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \wedge \frac{\partial y}{\partial x}$$

Veranschaulichung der Kettenregel



$$y = x \wedge c$$

$$\frac{\partial y}{\partial x} = (0 \wedge c) \oplus (1 \wedge c) = c$$

$$z = (d \wedge e) \vee y$$

$$\frac{\partial z}{\partial y} = ((d \wedge e) \vee 0) \oplus ((d \wedge e) \vee 1) = (d \wedge e) \oplus 1 = \overline{d \wedge e}$$

$$\begin{aligned} \frac{\partial z}{\partial x} &= \underbrace{c \wedge \overline{d \wedge e}}_{\text{Kettenregel}} = \underbrace{((0 \wedge c) \vee de) \oplus ((1 \wedge c) \vee de)}_{\text{direkte Berechnung}} = de \oplus (c \vee de) \\ &= (de \wedge \overline{(c \vee de)}) \vee (\overline{de} \wedge (c \vee de)) \end{aligned}$$

$$\begin{aligned} &= \underbrace{(de \wedge \overline{c} \wedge \overline{de})}_{=0} \vee \underbrace{(\overline{de} \wedge c) \vee (\overline{de} \wedge de)}_{=0} = \overline{d \wedge e} \wedge c \end{aligned}$$

Einige grundlegende BOOLEsche Differenzen als Rechenhilfe

AND: $\frac{\partial(a \wedge b)}{\partial a} = b$

OR: $\frac{\partial(a \vee b)}{\partial a} = \bar{b}$

XOR: $\frac{\partial(a \oplus b)}{\partial a} = 1$

NOT: $\frac{\partial \bar{a}}{\partial a} = 1$

Das Prinzip der Testmustergenerierung mit BOOLEschen Differenzen ist auch auf andere Fehlermodelle anwendbar, z. B. auf Kurzschlussfehlermodell. Durch komplexeres Fehlerverhalten sind die Gleichungen jedoch komplizierter.

Grundlage: D-Notation

D-Algorithmus ($D \hat{=} \text{defect, discrepancy}$) gehört zu Testmustererzeugungsverfahren, die auf dem Prinzip der Pfadsensibilisierung aufbauen:

- ▶ vom Grundgedanken her mit dem Verfahren der BOOLEschen Differenzen vergleichbar
- ▶ sucht nur einen Testsatz pro Fehler statt aller möglichen
- ▶ weniger rechenintensiv (dennoch exponentielle Laufzeit im ungünstigsten Fall).

Einführung der D-Notation:

- ▶ Die Situation „Sollwert 1, Fehlerwert 0“ wird als D dargestellt
- ▶ Die Situation „Sollwert 0, Fehlerwert 1“ wird als \overline{D} dargestellt
- ▶ Vier- bzw. fünfelementige Signalmenge $\{0, 1, D, \overline{D}\}$ bzw. $\{0, 1, D, \overline{D}, \times\}$

Funktionstabellen einiger Grundgatter im D-Kalkül

AND	0	1	D	\bar{D}	OR	0	1	D	\bar{D}	NOT	x	y
0	0	0	0	0	0	0	1	D	\bar{D}	0	1	
1	0	1	D	\bar{D}	1	1	1	1	1	1	0	
D	0	D	D	0	D	D	1	D	1	D	\bar{D}	
\bar{D}	0	\bar{D}	0	\bar{D}	\bar{D}	\bar{D}	1	1	\bar{D}	\bar{D}	\bar{D}	D

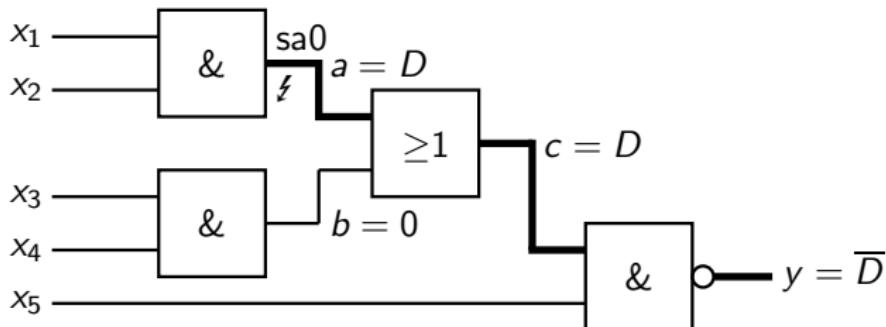
XOR	0	1	D	\bar{D}
0	0	1	D	\bar{D}
1	1	0	\bar{D}	D
D	D	\bar{D}	0	1
\bar{D}	\bar{D}	D	1	0

Die Funktionstabellen werden verwendet, um Signalwerte durch die zu testende Schaltung zu propagieren. Zielstellungen:

- ▶ Sensibilisierung des Ausgangs
- ▶ Ermittlung der sensibilisierenden Eingangsbelegung

- Ermittlung eines **D-Pfades** (auch D-Kette genannt) von der Fehlerstelle zum Ausgang.

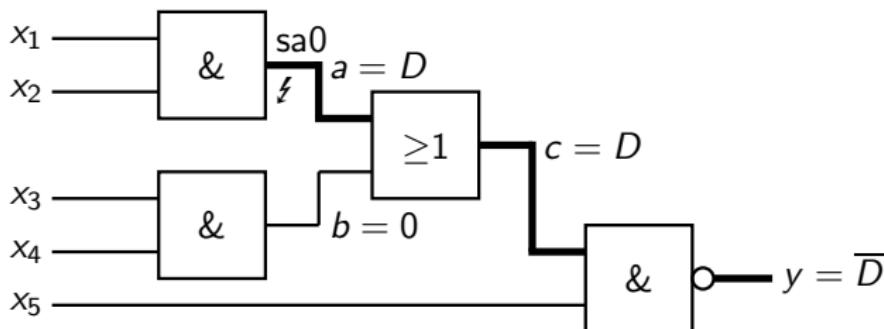
Beispiel einer D-Kette



Bei Fehlerannahme $a=sa0$ gilt $a = D$. Damit der Fehler am Ausgang beobachtbar ist, muss D oder \bar{D} zum Ausgang propagiert werden:

- ▶ $c = D$ ist bei $b = 0$ oder $b = D$ erreichbar (siehe Tabelle).
 $b = D$ erfordert eine Signalveränderung am Eingang b , die nicht vorausgesetzt werden kann, daher $b = 0$.
- ▶ $c = D$ kann nur dann als $y = \bar{D}$ zum Ausgang propagiert werden, wenn $x_5 = 1$ gilt.

Zusammenfassung der Entstehungsbedingungen für die gefundene D-Kette



$$\bar{b} \wedge x_5 = 1$$

$$b = x_3 \wedge x_4 \Rightarrow (\overline{x_3 \wedge x_4}) \wedge x_5 = 1$$

Aus der Bedingung der Fehlersimulation $a = 1$ folgt außerdem $x_1 = x_2 = 1$, womit der entsprechende Testsatz vollständig ist: $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 0, 0, 1)$. Anmerkung: Die Gleichung $(x_3 \wedge x_4) \wedge x_5 = 1$ entspricht der BOOLESchen Differenz von y nach a .

Funktionsgrundlage für den D-Algorithmus

Fehlersimulation und Fehlerpropagation

Ein Haftfehler $a\text{-sa}0$ (bzw. $a\text{-sa}1$) kann genau dann entdeckt werden, wenn

- ▶ dieser Fehler simulierbar ist, d. h. die Leitung a mit dem Wert D (bzw. \overline{D}) belegt werden kann sowie
- ▶ eine D-Kette von der Leitung a zu mindestens einem der primären Ausgänge erzeugt werden kann.

Eine Belegung der primären Eingänge, die diese beiden Bedingungen sicherstellt entspricht einem Testmuster für den genannten Fehler.

D-Algorithmus terminiert auch bei nicht beobachtbaren Fehlern genau dann, wenn keine D-Kette erzeugt werden kann oder der Fehler nicht simuliert werden kann:

- ▶ Testmuster nicht vorhanden.

Vorgehensweise

1. Entferne die Leitung a (verbindet einen Gatter-Ausgang A mit einem Gatter-Eingang E), für die ein Haftfehler angenommen worden ist.
2. Suche einen Zustand der Schaltung, in dem alle betroffenen Knoten einen der vier zulässigen Werte aufweisen: $\{0, 1, D, \overline{D}\}$, wobei insbesondere gelten muss:

$$A = \begin{cases} 0 & \text{für } a=\text{sa1} \\ 1 & \text{für } a=\text{sa0} \end{cases} \quad E = \begin{cases} \overline{D} & \text{für } a=\text{sa1} \\ D & \text{für } a=\text{sa0} \end{cases}$$

3. Für alle Schaltungseingänge muss $x_i \in \{0, 1\}$ gelten.
4. Für mindestens einen Schaltungsausgang muss $y_j \in \{D, \overline{D}\}$ gelten.
5. Existiert x_1, \dots, x_n , so ist es das gesuchte Testmuster, sonst ist der Fehler nicht beobachtbar.

Backtracking

Insbesondere Schritt 2 ist nicht trivial. In der Originalveröffentlichung wird das allgemeine Backtracking-Verfahren vorgeschlagen:

1. D bzw. \bar{D} in Richtung der Ausgänge propagieren (Bildung einer D-Kette)
2. Den für die Fehlersimulation erforderlichen Wert in Richtung der Eingänge rückpropagieren. Wenn ein Widerspruch entsteht (z. B. gleiche Pegel am Eingang und Ausgang eines Inverters), so muss der logische Wert des zuletzt zugewiesenen Knotens invertiert werden (Prinzip der binären Baumsuche). Besteht der Widerspruch fort, kann der gesamte Ast des Suchbaums entfernt und der vorletzte zugewiesene Knoten invertiert werden.
3. Besteht der Widerspruch nach dem vollständigen Backtracking immer noch fort, so kann kein Testmuster generiert werden
→ Der Fehler ist nicht beobachtbar.

Konflikttabelle als Hilfestellung bei der Berechnung

	0	1	\times	D	\overline{D}
0	0	K	0	K	K
1	K	1	1	K	K
\times	0	1	\times	D	\overline{D}
D	K	K	D	D	A
\overline{D}	K	K	\overline{D}	A	\overline{D}

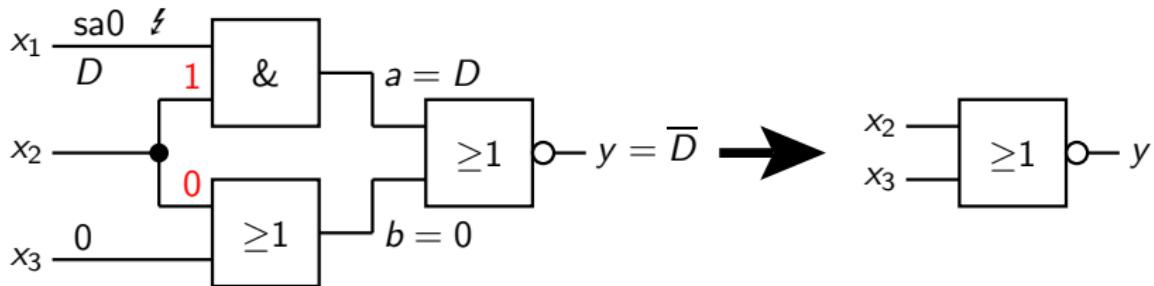
$\times \cong$ (noch) nicht initialisiert

$A \cong$ auflösbar, entweder D oder \overline{D} zuweisen

$K \cong$ nicht auflösbar, Backtracking

Beispiel eines redundanten Fehlers

Keine D-Kette ermittelbar:



Das Propagieren von D durch das AND-Gatter erfordert die Belegung $x_2 = 1$ (siehe Tabellen auf Seite 420). Das Propagieren von D durch das NOR-Gatter (von a zum Ausgang y) erfordert $b = 0$. Aus $b = 0$ folgt $x_2 = x_3 = 0$, was im Widerspruch zur bereits ermittelten Forderung $x_2 = 1$ steht (der nicht beseitigt werden kann).

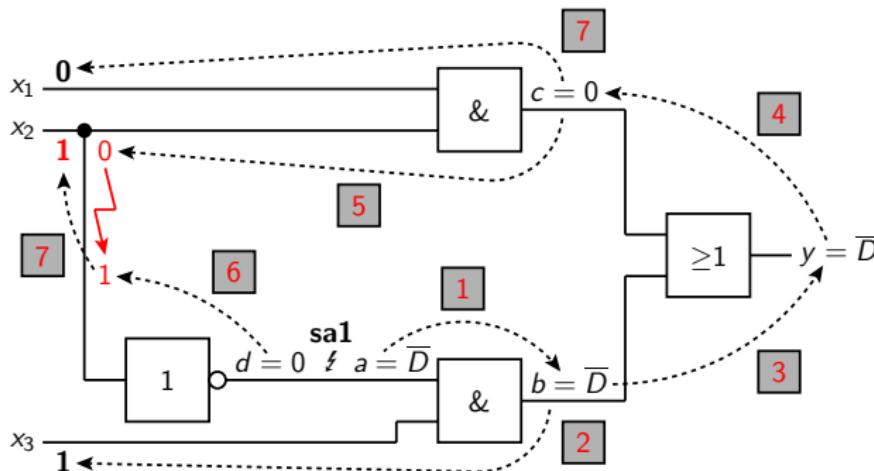
→ Ein Testvektor ist nicht generierbar, der Fehler ist redundant.

$$y = \overline{(x_1 \wedge x_2) \vee (x_2 \vee x_3)} = \overline{x_2 \wedge (x_1 \vee 1) \vee x_3} = \overline{x_2 \vee x_3}$$

Anmerkungen

- ▶ Kann keine D-Kette konstruiert werden, so deutet das auf eine Redundanz hin (falls diese nicht gewollt ist, kann die Schaltung optimiert werden).
- ▶ Exponentielle Laufzeit ist dem Backtracking zu verdanken: Schwer oder nicht nachweisbare Fehler erfordern eine (nahezu) komplette Durchsuchung des Baums.
- ➡ Ansatzpunkt für heuristische Suchverfahren:
 - ▶ Aussichtslose Suchwege möglichst schnell erkennen und verwerfen
 - ▶ Vielversprechende Suchpfade begünstigen
 - ➡ *Branch-and-Bound-Methode, PODEM-Algorithmus
(path-oriented decision making)*

Beispiel zum Ablauf mit Backtracking



1. Fehlerpropagation (D-Kette bilden): $a = \bar{D} \Rightarrow \underbrace{b = \bar{D}}_1 \Rightarrow \underbrace{x_3 = 1}_2$
 $\Rightarrow \underbrace{y = \bar{D}}_3 \Rightarrow \underbrace{c = 0}_4 \Rightarrow \underbrace{x_2 = 0, x_1 = -}_5$
 2. Fehlersimulation:
 $d = 0 \Rightarrow \underbrace{x_2 = 1}_6 \Rightarrow \text{Widerspruch!} \Rightarrow \underbrace{x_2 = 1}_7 \Rightarrow \underbrace{x_1 = 0}_7$
- Testmuster $(x_1, x_2, x_3) = (0, 1, 1)$

Zufallstest und sonstige Maßnahmen

Analytische Erzeugung von Testmustern ist rechenaufwändig. Eine einfachere Möglichkeit liefert der Zufallstest:

- ▶ Abschätzungen der Fehlerüberdeckung ausgehend von einer zufälligen Auswahl der Testfolgen
- ▶ Ausnutzung der statistischen Eigenschaften: Fehlerstichproben, *worst-case* Nachweiswahrscheinlichkeiten, Verteilungen
- ▶ Gewichteter Zufallstest

Weitere (schaltungstechnische) Testmaßnahmen:

- ▶ Prüfgerechter Entwurf
- ▶ Selbsttest
- ▶ Ein eigenständiges Forschungsgebiet

Zusammenfassung

Testung:

- ▶ Grundbegriffe des Testens
- ▶ Haftfehlermodell
- ▶ Automatische Testsatzerzeugung mit BOOLEscher Differenz
- ▶ Automatische Testsatzerzeugung mit D-Algorithmus