Electronic Design Automation (EDA)

Systementwurf

Systembegriff
Beispiel Antiblockiersystem
Signalverarbeitung
Hardware/Software- Partitionierung
Hardware oder Software?
Electronic Design Automation

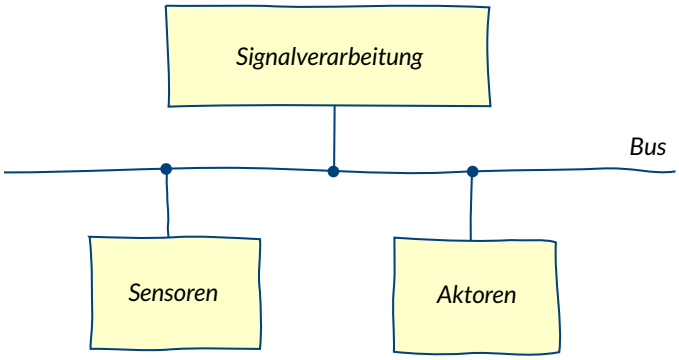
Date of generation: 8.7.2025

Systementwurf: Systembegriff

Systembegriff

Technische Systeme...

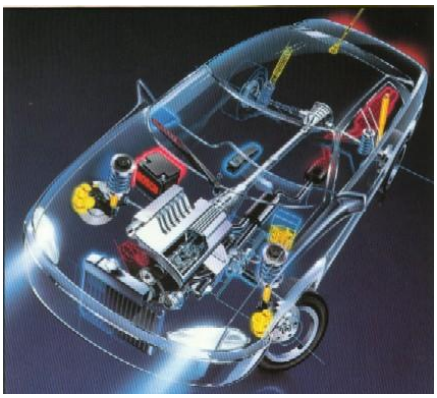
... umfassen im Wesentlichen Komponenten der Elektrotechnik, der Informatik sowie der mit ihnen wechselwirkenden Umgebungen. Sie bestehen in der Regel aus Sensor-, Aktor- und Signal-verarbeitungskomponenten bzw. -prozessen.



Grob vereinfacht besteht ein (technisches) System aus Sensoren und Aktoren, die den Kontakt mit der Umwelt herstellen, sowie einem Signalverarbeitungsteil. Sensoren nehmen physikalische Signale auf, die im Signalverarbeitungsteil Rechenprozessen unterworfen werden. Als Ergebnis der Verarbeitung werden Aktoren angesteuert, die entsprechende Aktionen an die Umwelt weitergeben. Die Komponenten kommunizieren häufig über einen gemeinsamen, so genannten (System-)Bus miteinander. Sensoren und Aktoren sind elektrischer, mechanischer, optischer, chemischer oder anderer Natur. Der Signalverarbeitungsteil ist heute in der Regel elektronischer Natur.

Systementwurf: Beispiel Antiblockiersystem

Beispiel: Antiblockiersystem



Quelle: Robert Bosch GmbH



Drehzahlsensor



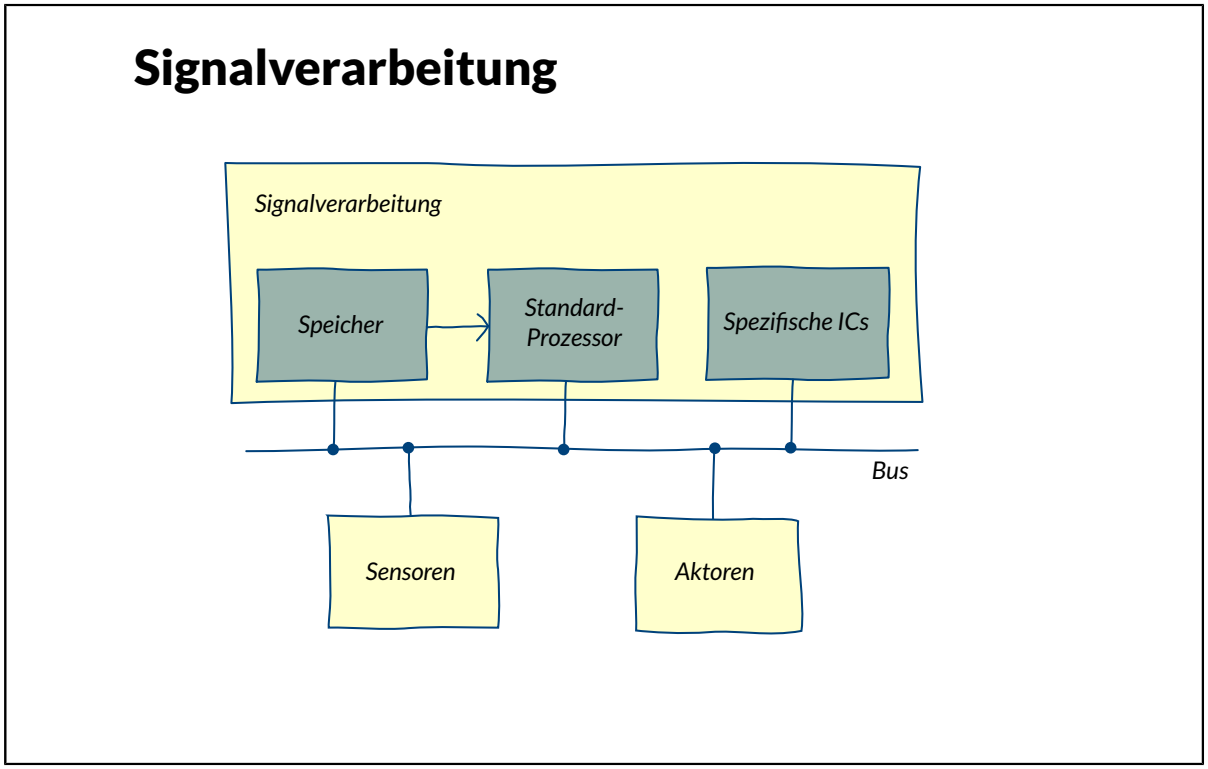
Hydraulik mit Steuergerät

Ein Beispiel für ein technisches System stellt das Antiblockiersystem (ABS) eines Autos dar.

Sensoren erfassen die Umdrehungsgeschwindigkeit der Räder des Fahrzeugs. Die entsprechenden Signale werden im Steuerungsbaustein verarbeitet, der schließlich entsprechende Signale an die hydraulischen Bremszylinder (Aktoren) abgibt.

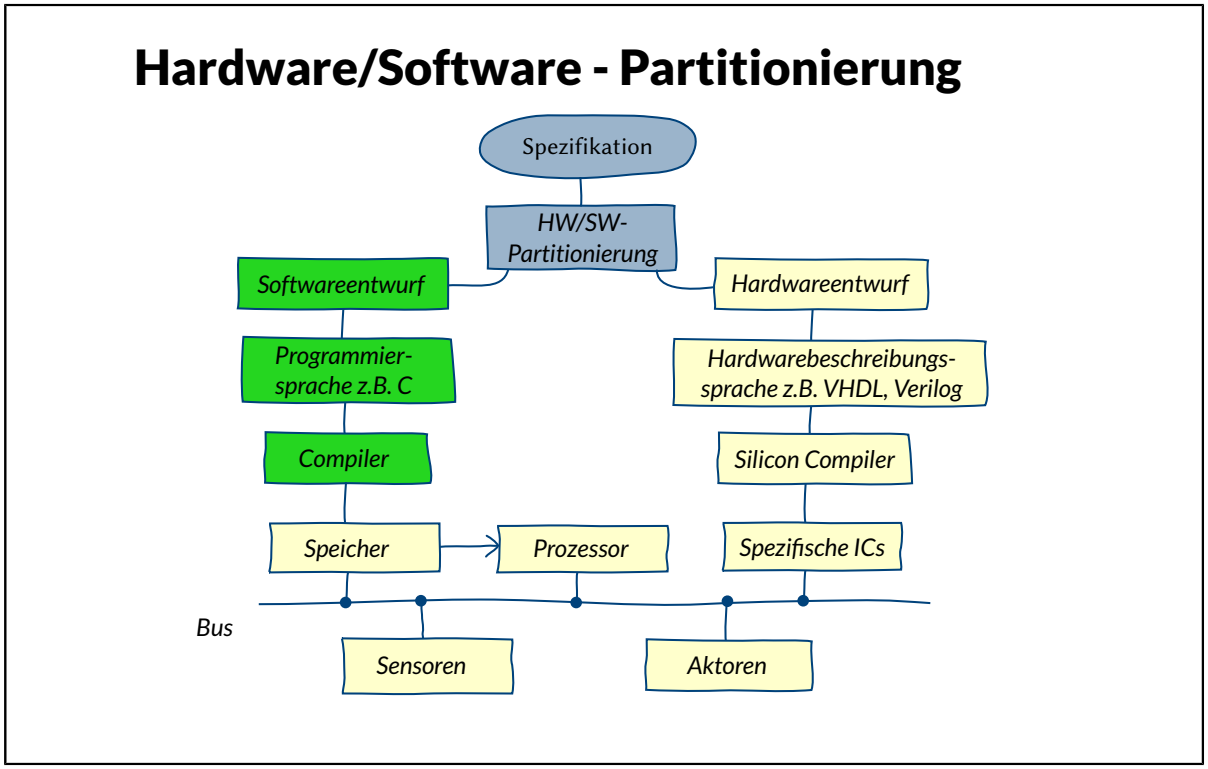
Im Rahmen dieser Vorlesung wollen wir uns auf den elektronischen Signalverarbeitungsteil konzentrieren.

Systementwurf: Signalverarbeitung



Wiederum grob vereinfacht besteht der Signalverarbeitungsteil aus einem universellen Rechenwerk (Standardprozessor), einem Speicher, der das auf dem Prozessor auszuführende Programm sowie die anfallenden Daten enthält, sowie weitere, für Spezialaufgaben zur Verfügung stehende, so genannte spezifische integrierte Schaltungen (Integrated Circuits, ICs). Während ein Speicher in der Regel unverzichtbar ist, können – je nach Aufgabenstellung – entweder der universelle Prozessor oder die spezifischen ICs entfallen.

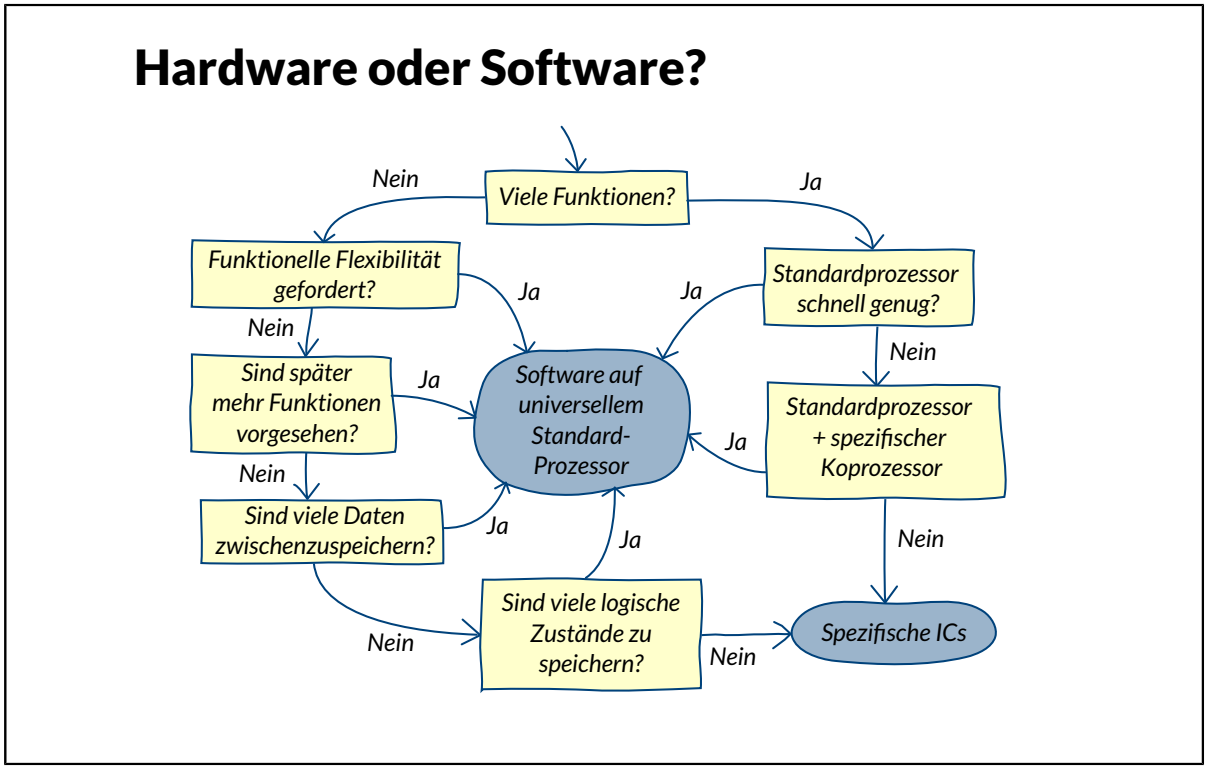
Systementwurf: Hardware/Software- Partitionierung



Ausgangspunkt für den Entwurf des Signalverarbeitungsteils eines Systems ist seine Spezifikation, in der die gewünschten Funktionen, die das System ausführen soll, beschrieben sind. Entsprechend der vorgenommenen Aufteilung in einen universellen Standardprozessor und spezifische ICs besteht der erste Teil der Entwurfsaufgabe darin zu entscheiden, welche der gewünschten Funktionen in – auf dem universellen Prozessor abzulaufende – Software und welche unmittelbar in – durch spezifische ICs zu realisierende – Hardware implementiert werden sollen.

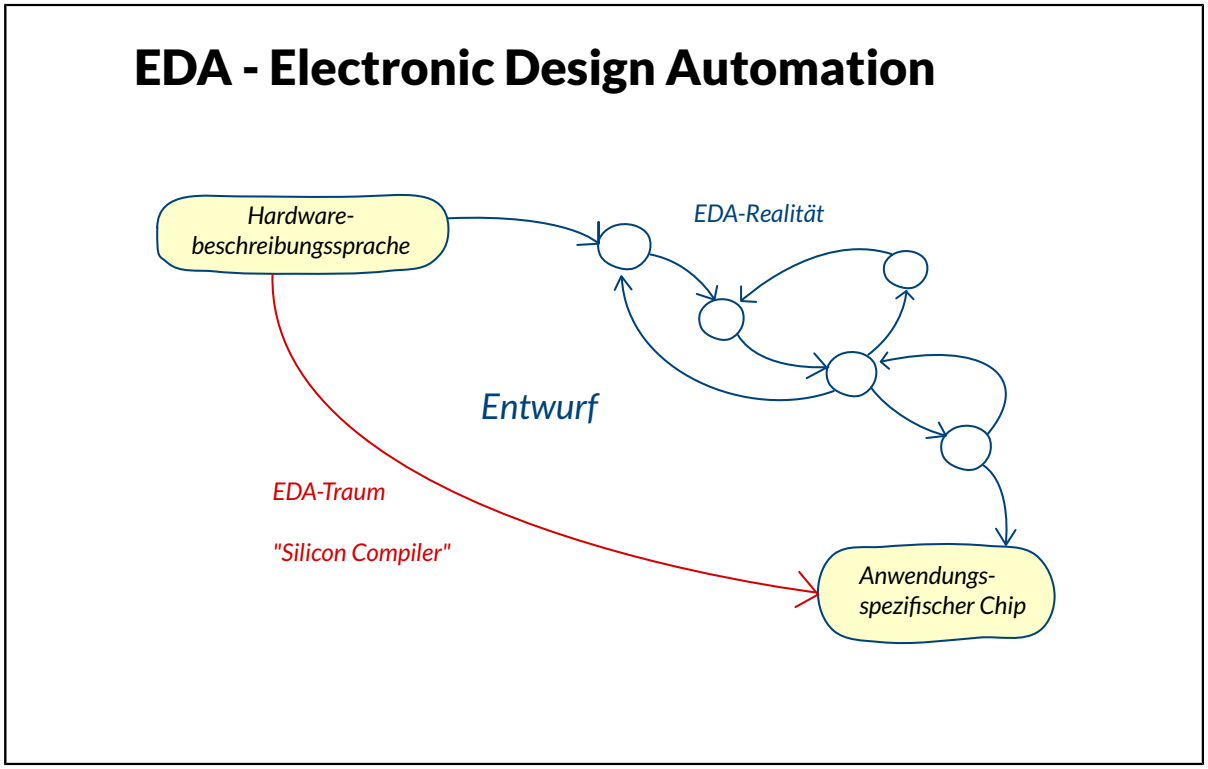
Der Entscheidungsprozess, welche Funktionen in Software bzw. in Hardware ausgeführt werden sollen, ist Gegenstand der sogenannten Hardware/Software-Partitionierung. Sie ist ein aktueller Forschungsgegenstand, jedoch nicht Gegenstand dieser Vorlesung, die sich auf Werkzeuge für den Hardwareentwurf beschränkt. Wie das Bild Hardware/Software - Partionierung zeigt, weist der Entwurf auf der gewählten, noch sehr abstrakten Betrachtungsebene große Ähnlichkeit mit dem Softwareentwurf auf. Die zu realisierende (Teil-)Funktion muss zunächst in einer geeigneten - maschinenlesbaren - Sprache beschrieben werden. Dies erfolgt beim Softwareentwurf in einer höheren Programmiersprache wie z.B. C, C++ oder Java. Auf der Hardwareseite wird das entsprechende Äquivalent als Hardware-Beschreibungssprache bezeichnet. Die am weitesten verbreiteten Hardwarebeschreibungssprachen sind Verilog und VHDL; dabei handelt es sich um formale, programmiersprachenähnliche Beschreibungen, die in der Lage sind, neben arithmetischen Ausdrücken, Kontrollanweisungen etc. auch Hardwarestrukturen und -eigenschaften zu beschreiben.

Systementwurf: Hardware oder Software?



Diese Frage ist nicht leicht zu beantworten. Einige Kriterien dazu sind beispielhaft dargestellt. Eine Prozessor-Lösung ist sicher stets dann vorzuziehen, wenn viele Daten sequentiell verarbeitet werden müssen und wenn die Lösung flexibel sein soll, da viele Funktionsänderungen durch Modifikation der Software leicht durchgeführt werden können. Die höchste Performance wird dagegen stets durch eine unmittelbare Hardware-Realisierung erzielt. Weitere nicht dargestellte Aspekte sind beispielsweise die Verlustleistung und vor allem die Kosten der gewählten Lösung, die stark von der zu realisierenden Stückzahl abhängig sind.

Systementwurf: Electronic Design Automation



Mit Hilfe eines Compilers wird die in einer höheren Programmiersprache beschriebene Software nun in der Regel vollautomatisch in die Maschinensprache des Prozessors übersetzt, so dass sie dort ausgeführt werden kann. Es liegt deshalb nahe, dieselbe Idee, auch für den Hardwareentwurf zu verwenden, und sich einen "Silicon-Compiler" vorzustellen, mit dessen Hilfe aus dem in der Hardware-Beschreibungssprache vorliegenden Entwurf ebenfalls vollautomatisch die für die Fertigung einer integrierten Schaltung erforderlichen Daten erzeugt werden. Tatsächlich ist dies – und wird es voraussichtlich auch bleiben – ein Traum. Der dazu erforderliche "Übersetzungsprozess" ist derartig komplex und aufwändig, dass er nur als Abfolge von zahlreichen unterschiedlichen Entwurfsschritten mit zunehmendem Detaillierungsgrad beherrscht werden kann. Viele dieser Schritte sind heute automatisiert bzw. teilautomatisiert. Wir sprechen dabei von Entwurfsautomatisierung (Electronic Design Automation, EDA) und sind damit beim eigentlichen Gegenstand dieser Vorlesung angelangt.

EDA als solches ist ein aktueller Forschungsgegenstand, dessen Status sich kontinuierlich verändert. In dieser Vorlesung wird der heutige Stand dargestellt, wobei der Schwerpunkt auf den grundsätzlichen Prinzipien und Algorithmen liegt. Der aktuelle Stand in der Praxis kann davon im Detail abweichen. Wichtig ist deshalb, die grundsätzliche Vorgehensweise und die zugrundeliegenden Zusammenhänge zu verstehen. Dies soll, anhand exemplarischer Beispiele vermittelt werden.

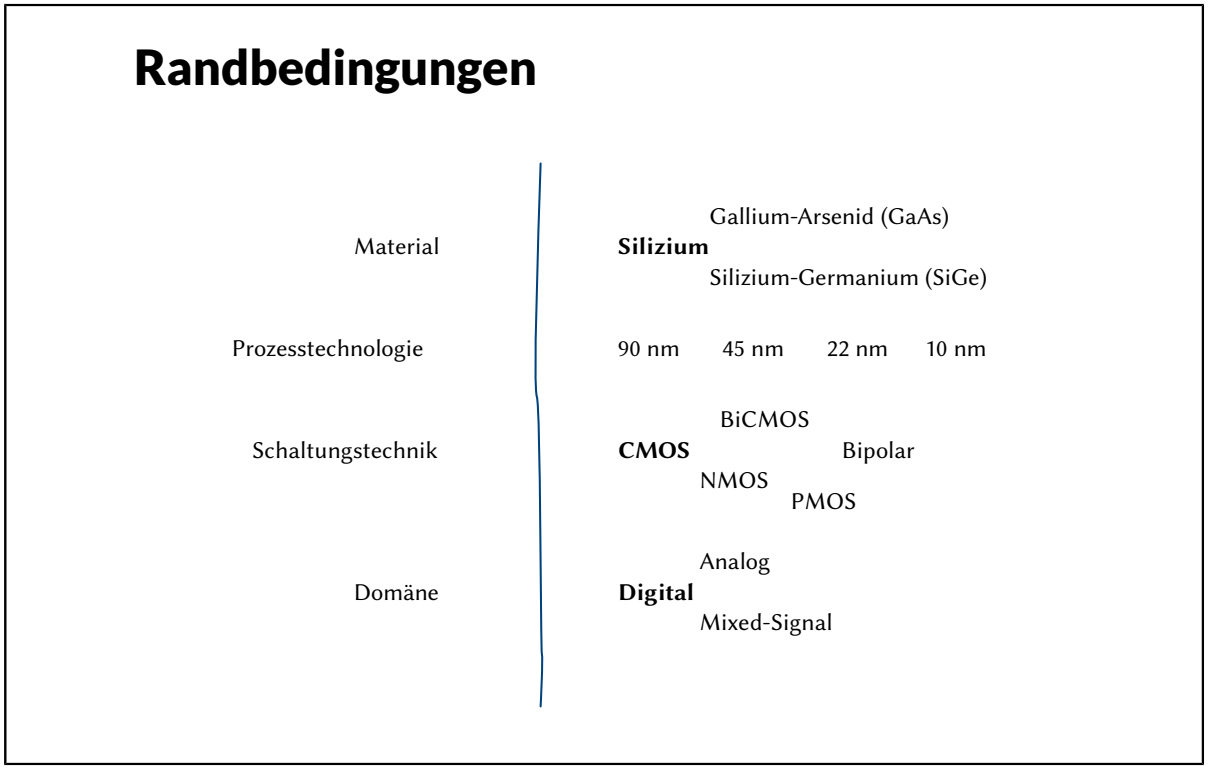
Electronic Design Automation (EDA)

Entwurf integrierter Schaltungen

Randbedingungen
Strukturorientierte Klassifizierung
Flexibilität hat ihren Preis
Optimierte Individualisten
Marktorientierte Klassifikation
Kosten und Stückzahlen
Kosten und Entwurfsmethoden
EDA-Ziele

Date of generation: 8.7.2025

Entwurf integrierter Schaltungen: Randbedingungen

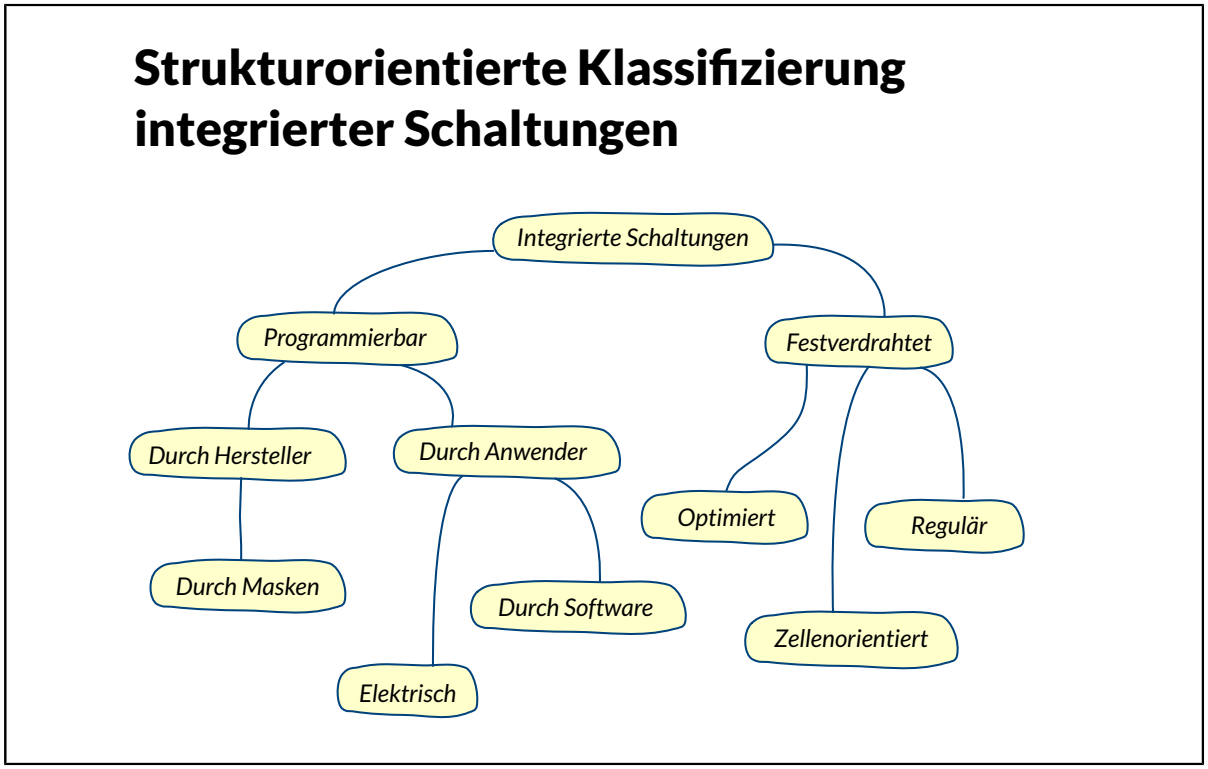


Die Frage, wie im Detail eine integrierte Schaltung entworfen wird, hängt von zahlreichen äußeren Randbedingungen ab. Wichtig sind z.B. die zur Herstellung verwendete Prozesstechnologie, die bei Standardtechnologien häufig durch die Angabe der so genannten minimalen Strukturbreite charakterisiert wird, und das Grundmaterial wie z.B. Silizium, Gallium-Arsenid (GaAs), Silizium-Germanium (SiGe) etc. Wegen seiner überragenden Bedeutung wird im Rahmen dieser Vorlesung ausschließlich davon ausgegangen, dass die zu entwerfenden Schaltungen in Silizium realisiert werden sollen. Als Schaltungstechnik wird - von einigen Ausnahmen abgesehen - CMOS vorausgesetzt.

Große Unterschiede bestehen auch zwischen dem Entwurf digitaler und analoger bzw. so genannter Mixed-Signal-Schaltungen. Hierauf wird bei der Behandlung der entsprechenden Werkzeuge eingegangen.

Ebenso von großer Bedeutung für die Entwurfstechnik ist eine Klassifizierung integrierter Schaltungen, die ihre Struktur berücksichtigt.

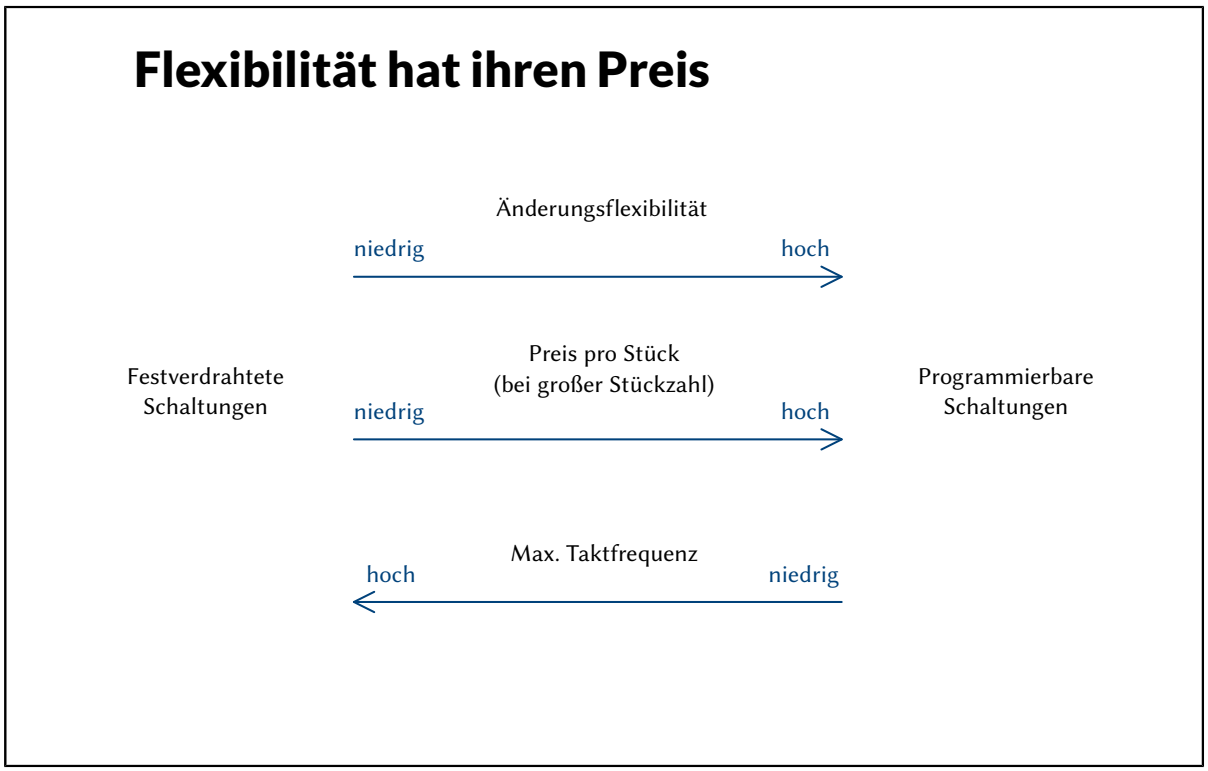
Entwurf integrierter Schaltungen: Strukturorientierte Klassifizierung



Programmierbare Schaltungen bieten die Möglichkeit, Hardware, deren Funktion noch nicht festgelegt ist, in großen Stückzahlen zu produzieren. Die individuelle Funktion wird durch einen - in sehr unterschiedlichen Formen realisierten - Programmiervorgang festgelegt. Er kann bereits beim Hersteller erfolgen, z.B. durch individuelle Masken, die die Verdrahtung einer Schaltung, die aus einer regulären Anordnung von Gattern oder Transistoren (Gate-Array) aufgebaut ist, festlegen. Als Sonderfall gehören auch Read-Only-Memories (ROMs) zu dieser Gruppe. Alternativ kann die Programmierung auch durch den Anwender erfolgen, entweder durch elektrische Vorgänge ("Durchbrennen" von Sicherungen, Einstellung der Leitfähigkeit von Transistoren) oder durch Programmierung, z.B. durch Ablegen einer Wahrheitstabelle in einem Speicher. Zu solchen anwenderprogrammierbaren Schaltungen zählen im Speicherbereich ROMs (PROMs), Erasable PROMs (EPROMs) und Electrically Erasable PROMs (EEPROMs) sowie im Logikbereich Programmable Logic Arrays (PLAs), Programmable Array Logic (PALs), Programmable Logic Devices (PLDs) sowie Field Programable Gate Arrays (FPGAs).

Im Fall von festverdrahteten Schaltingen wurde die Funktionalität der Schaltung schon vor der Produktion durch den Kunden festgelegt. Auf Beispiele der verschiedenen Ausprägungen festverdrahteter Schaltungen wird später auf der Folie "Optimierte Individualisten" eingegangen.

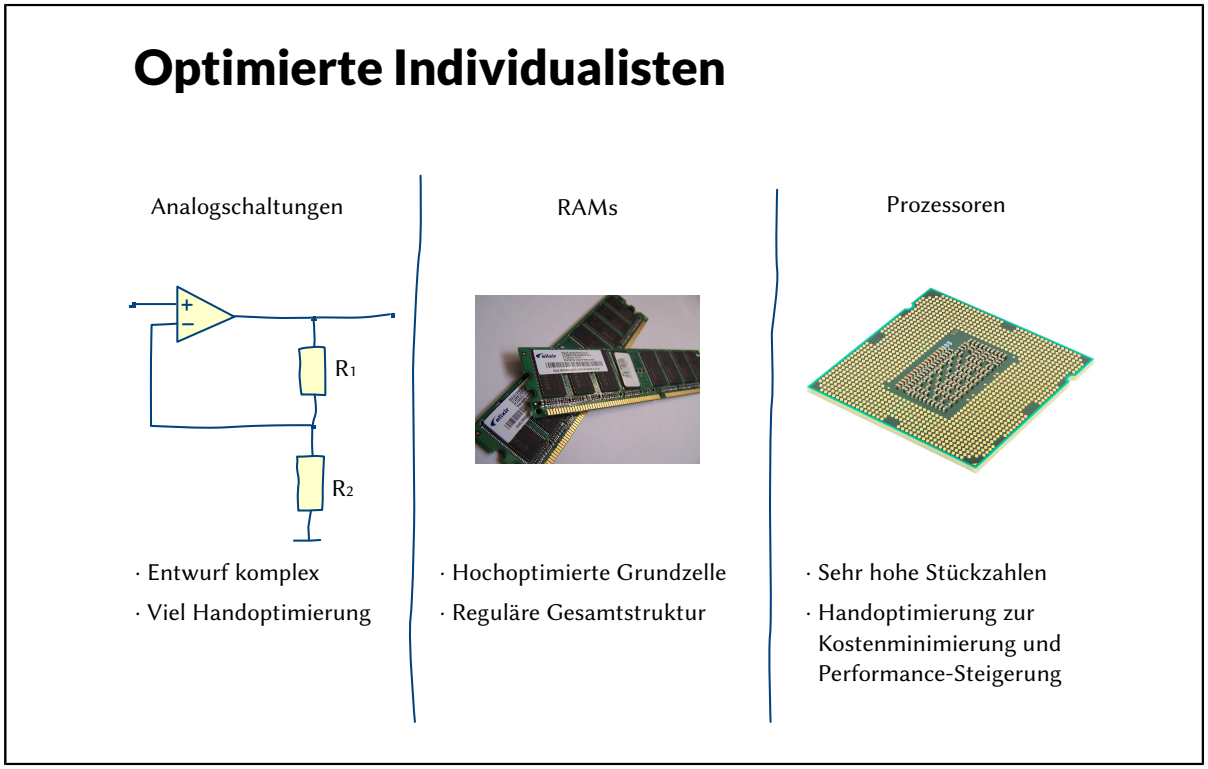
Entwurf integrierter Schaltungen: Flexibilität hat ihren Preis



Allen programmierbaren Logikschaltungen ist gemeinsam, dass sie aus Anwendersicht zwar sehr flexibel sind und eine schnelle Realisierung einer individuellen Schaltung ermöglichen, auf der anderen Seite aber auch große Chipflächen benötigen und deshalb vergleichsweise teuer sind, so dass ihr Einsatz i.a. nur bei kleinen Stückzahlen sinnvoll ist. Große Bedeutung haben heute FPGAs erlangt, insbesondere solche, die mehrfach programmiert werden können. Sie werden heute in hohen Komplexitäten (mehrere Millionen Gatter) zu relativ günstigen Preisen angeboten und stellen für Prototypen und Kleinstückzahlen ein attraktives Angebot dar. Auf die Besonderheiten beim Entwurf programmierbarer Schaltungen, die sich durch eine hohe Regularität auszeichnen, kann hier nicht detailliert eingegangen werden. Einige Hinweise werden bei den entsprechenden Werkzeugen, vor allem im Rahmen des Layoutentwurfs gegeben.

Sollen von einer Schaltung größere Stückzahlen (> 10.000) hergestellt werden oder werden besondere Anforderungen an die Performance (z.B. Taktrate) gestellt, ist eine festverdrahtete Lösung unumgänglich. Wenn nicht ausdrücklich etwas anderes gesagt wird, wollen wir im Folgenden stets solche Schaltungen betrachten (auch wenn viele der zu behandelnden Entwurfsschritte für programmierbare Schaltungen identisch sind).

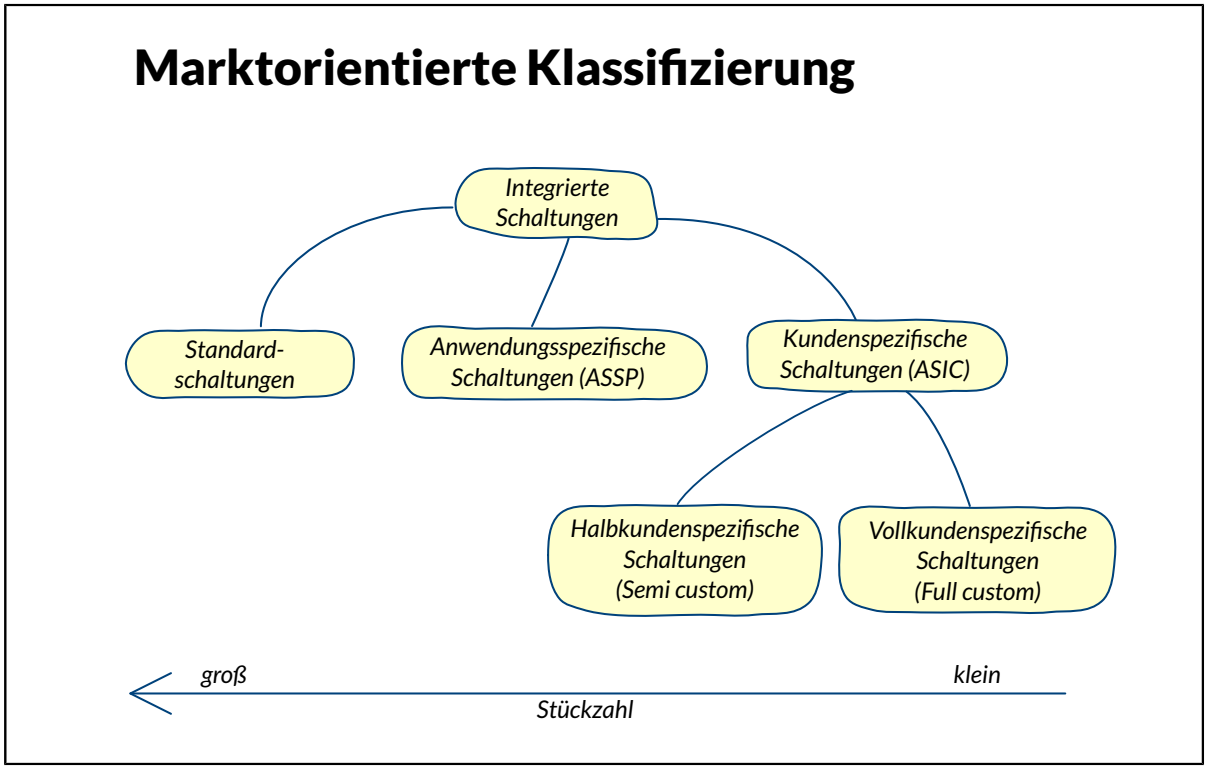
Entwurf integrierter Schaltungen: Optimierte Individualisten



Die individuellste Form einer integrierten Schaltung liegt dann vor, wenn auf jegliche Regularität verzichtet wird, d.h. dass jedes Gatter bzw. jeder Transistor individuell entworfen und optimiert werden. Dies ist erforderlich bei Analogschaltungen, aber auch bei solchen digitalen Schaltungen, bei denen höchste Performance bei kleinster Chipfläche angestrebt werden muss, wie z.B. bei in extrem hohen Stückzahlen hergestellten Prozessoren. Es liegt auf der Hand, dass in diesem Falle der Entwurf am aufwändigsten ist und viel Zeit und Aufwand, vor allem aber auch viel manuelle Optimierungsarbeit erfordert. Um eine Automatisierung zu ermöglichen, führt man so genannte Zellenkonzepte (Standardzellen, Makrozellen) ein, die einerseits die Wiederverwendung erprobter Schaltungsteile ("Reuse") aus sogenannten Bibliotheken ermöglichen, und andererseits durch eine gewisse Standardisierung verschiedene Entwurfsschritte vereinfachen. Hierauf wird im Rahmen des Layoutentwurfs eingegangen.

Eine Sonderstellung nehmen hochreguläre Schaltungen wie z.B. Random Access Memories (RAMs) ein. Hier ist zwar beim Entwurf der Grundzellen ein hoher Optimierungsaufwand erforderlich. Die Gesamtschaltung stellt dagegen wegen ihrer hohen Regularität keine besonderen Anforderungen an den Entwurf.

Entwurf integrierter Schaltungen: Marktorientierte Klassifikation

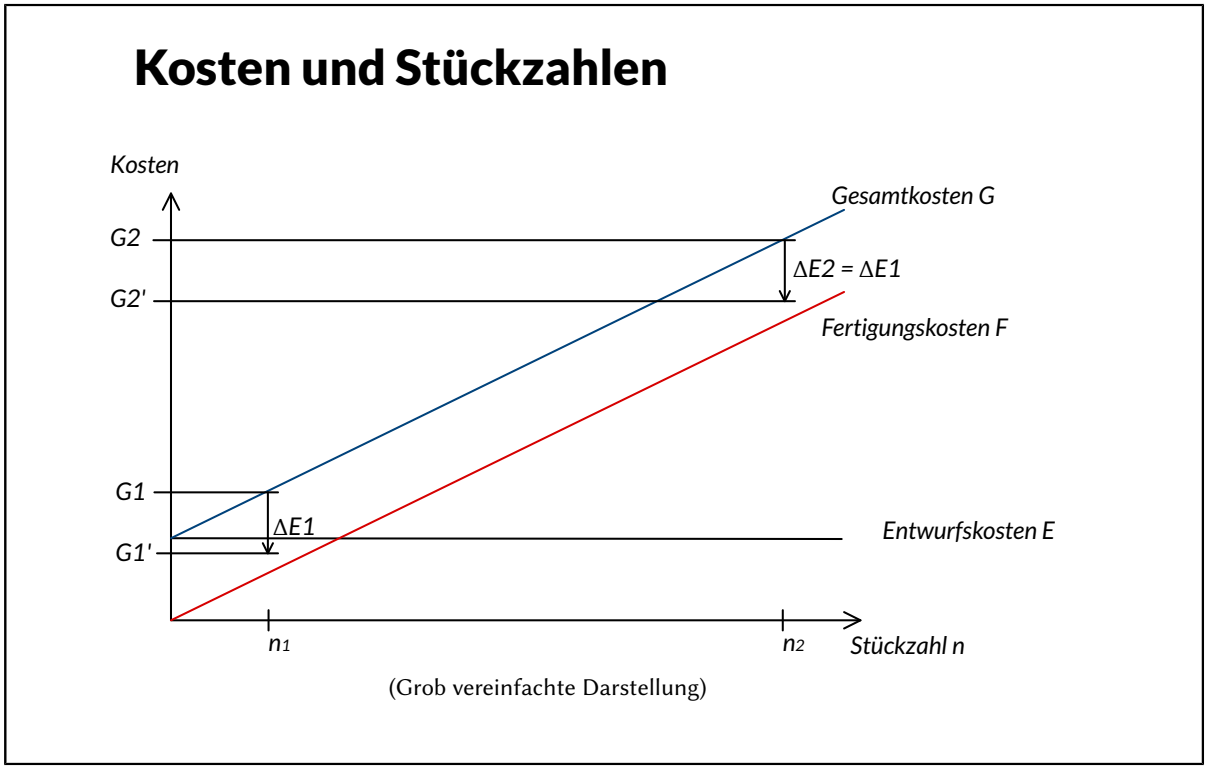


Häufig werden für integrierte Schaltungen Begriffe verwendet, die sich eher am Markt orientieren, die aber auch für den Entwurf von Bedeutung sind.

Für diese Begriffe gibt es keine eindeutigen Definitionen. Standardschaltungen sind Schaltungen, die von einem Halbleiterhersteller nach eigenen Vorstellungen entworfen und in sehr großer Stückzahl hergestellt werden, heute sind dies im Wesentlichen Speicher und Mikroprozessoren. Anwendungsspezifische Schaltkreise (ASSP = Application Specific Standard Product) sind solche Schaltungen, die für eine spezifische Anwendung entworfen wurden (z.B. ABS-Baustein, Basisbandbaustein für ein Mobiltelefon, etc.), jedoch nicht für einen speziellen Kunden. Dabei sind mittelgroße Stückzahlen erreichbar. Kundenspezifische Schaltungen (englisch meist ASIC = Application Specific Integrated Circuit: Hier wird der Raum für Missverständnisse deutlich!) werden für oder sogar durch einen bestimmten Kunden individuell entworfen. Hier sind die Stückzahlen in der Regel relativ gering.

Wesentlicher Hintergrund der Klassifizierung nach obigem Bild sind die Zusammenhänge zwischen Stückzahlen/Fertigungskosten und Entwurfsstilen/Entwurfszeit. Pauschal lässt sich feststellen, dass die gefertigten Stückzahlen pro Entwurf in der Abbildung von links nach rechts abnehmen. Insbesondere bei den halbkundenspezifischen Schaltungen sind sie häufig so klein, dass die Gesamtkosten von den Entwicklungskosten dominiert werden. Dazu ist eine andere Entwurfsmethodik notwendig als für die anderen Arten von Schaltungen.

Entwurf integrierter Schaltungen: Kosten und Stückzahlen



Letztlich ist das Ziel eines Halbleiterherstellers stets, seine Kosten zu minimieren, um seine Schaltungen am Markt möglichst preisgünstig anbieten zu können. Grob vereinfacht bestehen die Kosten aus Entwurfskosten und Fertigungskosten. Während die Entwurfskosten nur einmalig anfallen, sind die Fertigungskosten proportional - sehen wir hier der Einfachheit halber von den so genannten Fixkosten ab - zur gefertigten Stückzahl. Damit ist klar, dass der Einfluss der Entwurfskosten auf die Gesamtkosten stückzahlabhängig ist.

So führt bei einer kleinen Stückzahl n1 eine Reduktion der Entwurfskosten um 75% (Delta E1) zu einer Gesamtkostenreduktion ((G1-G1')/G1)) von ca. 60%. Bei einer großen Stückzahl n2 beträgt dagegen die Gesamtkostenreduktion nur noch ca. 20%.

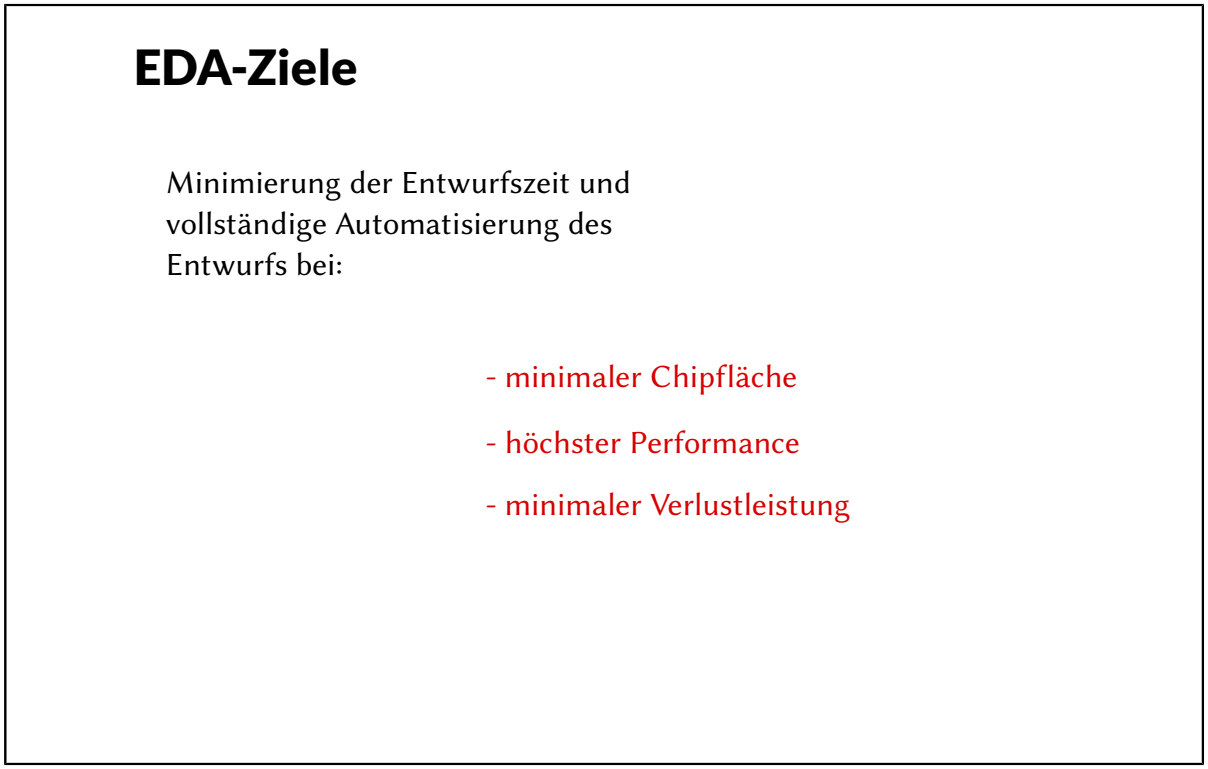
Entwurf integrierter Schaltungen: Kosten und Entwurfsmethoden



Wie beeinflussen nun Entwurfs- bzw. Fertigungskosten die Entwurfsmethodik? Betrachten wir dazu zunächst die Entwurfskosten. Diese sind in erster Linie proportional zur Entwurfszeit und zur Anzahl der eingesetzten Entwickler. Lassen wir letztere aus naheliegenden Gründen konstant, folgt, dass die Entwicklungszeit minimiert werden muss. Dies kann durch eine weitgehende Automatisierung des Entwurfs geschehen. Hierbei sinkt allerdings auf dem heutigen Stand der Technik die Qualität des Entwurfs, insbesondere steigt die zur Realisierung einer Schaltung benötigte Siliziumfläche an. Diese beeinflusst jedoch über die Materialkosten und über die mit wachsender Fläche stark sinkende Fertigungsausbeute die Fertigungskosten. Dominieren diese also die Gesamtkosten, gewinnt die Flächenoptimierung eine höhere Bedeutung als die Entwurfszeitminimierung. Im obigen Bild sind die sich daraus ergebenden Schlussfolgerungen zusammengefasst.

Standardschaltungen und anwendungsspezifische Schaltungen werden in der Regel große Stückzahlen aufweisen, werden also häufig "handoptimiert", bei kundenspezifischen Schaltungen dagegen dominieren wegen der geringeren Stückzahlen häufig die Entwurfskosten, hier ist also die Automatisierung besonders wichtig.

Entwurf integrierter Schaltungen: EDA-Ziele



Am Schluss dieser sehr vereinfachten Betrachtung muss jedoch betont werden, dass die vorgenommene strenge Trennung in einen entwurfskostendominierten und einen fertigungskostendominierten Ansatz in der Praxis so klar nicht auftritt. Die Minimierung von "Time-to-market" und damit auch der Entwurfszeit ist für alle Schaltungstypen ein wichtiges Ziel. Dies gilt natürlich auch für die Chipfläche, wobei häufig gar nicht die Chipfläche selbst, sondern die meist mit einer kleineren Fläche einhergehende Performancesteigerung einer Schaltung im Vordergrund des Interesses steht.

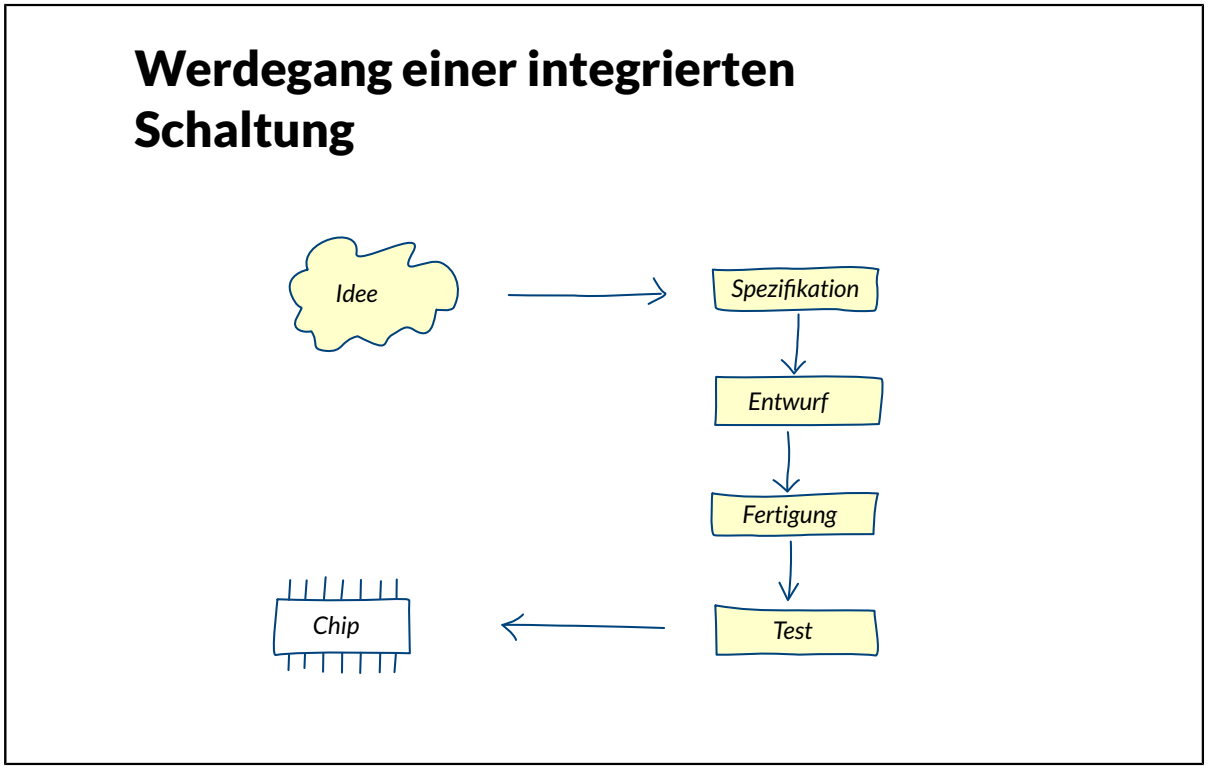
Hinzu kommt, dass alle vorgenommenen Klassifizierungen nur vorläufig und von begrenzter Gültigkeit sind, da sich die betrachtete Technik noch in rascher Entwicklung befindet. So dürfen auch die genannten Entwurfsansätze nicht scharf getrennt voneinander betrachtet werden. Ziel von EDA ist selbstverständlich eine vollständige Automatisierung des Entwurfs bei minimaler Chipfläche und höchster Performance. Da jedoch mit den Fortschritten der Entwurfstechnik auch die Komplexität der Entwurfsobjekte ständig zunimmt, kann dieses Ziel mittelfristig nicht erreicht werden. Wegen der unterschiedlichen Optimierungskriterien werden deshalb auch verschiedene Entwurfsstile erhalten bleiben. Diese werden sich jedoch wechselseitig durchdringen und befruchten.

Electronic Design Automation (EDA)

Entwurfsprozess

Werdegang einer integrierten Schaltung
Entwurf als Optimierungsproblem
Beherrschung der Komplexität
Funktioneller und physikalischer Entwurf
Produktivität
Entwurfsmodell Y-Diagramm
Ebenen und Sichten im Y-Diagramm
Systemebene
... Modellierungskonzept
... Beispiel
Algorithmische Ebene
... Beispiel
Register-Transfer-Ebene
... Beispiel
Gatterebene
... Beispiel
Elektrische Ebene
... Beispiel
Entwurfsstile
Synthese/Analyse
Silicon Compiler
Funktioneller/Physikalischer Entwurf
Top Down Entwurfsstil
Bottom Up Entwurfsstil
Meet in the Middle
EDA-Werkzeuge

Entwurfsprozess: Werdegang einer integrierten Schaltung

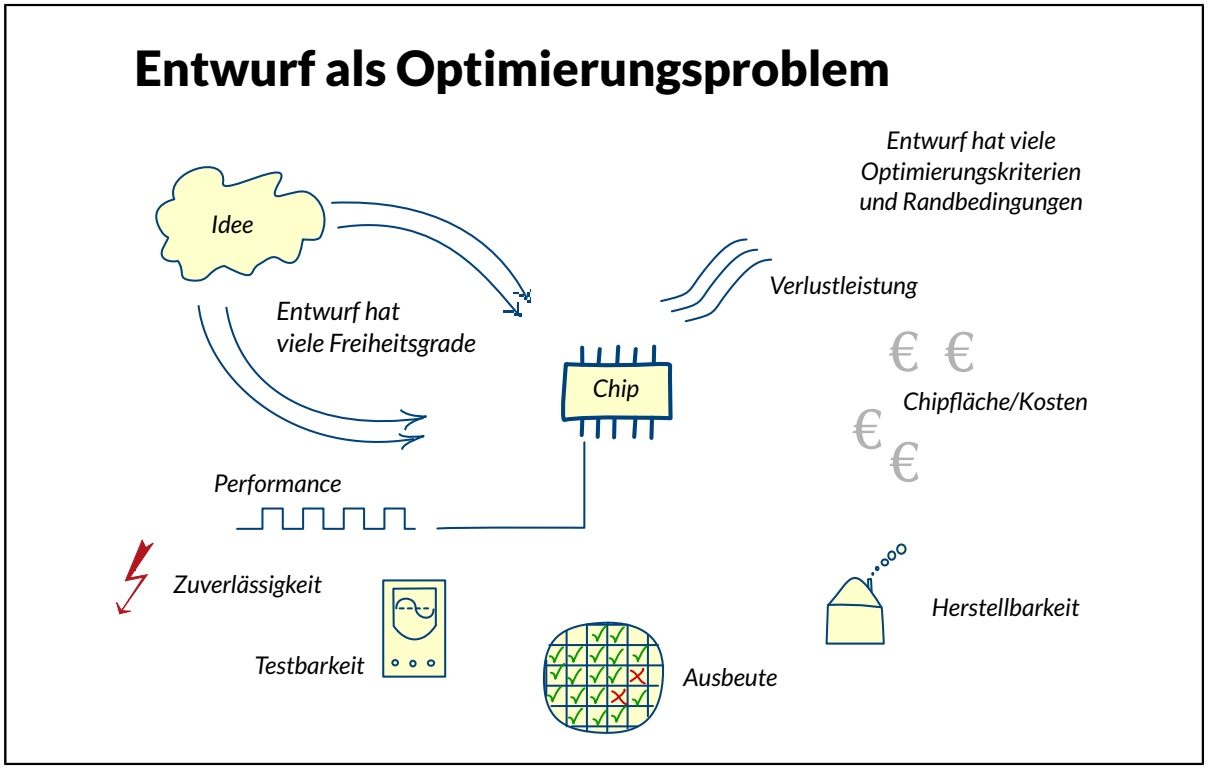


Im Kapitel Systementwurf haben wir bereits die Spezifikation als Ausgangspunkt des Entwurfs kennengelernt.

Wie bei jedem anderen Produkt folgt auf den Entwurfsprozess die Fertigung der integrierten Schaltung. Wie wir jedoch später sehen werden, beeinflussen die Eigenschaften des Fertigungsprozesses - anders als bei vielen anderen Produkten - den Entwurf in vielfältiger Weise. Insbesondere schränken sie die nahezu unendlich großen Entwurfsfreiheiten z.B. durch die Zahl der zur Verfügung stehenden Verdrahtungsebenen oder durch so genannte Entwurfsregeln wie Mindestbreiten oder Mindestabstände, mit denen im Entwurf die Toleranzen des Fertigungsprozesses berücksichtigt werden, wesentlich ein.

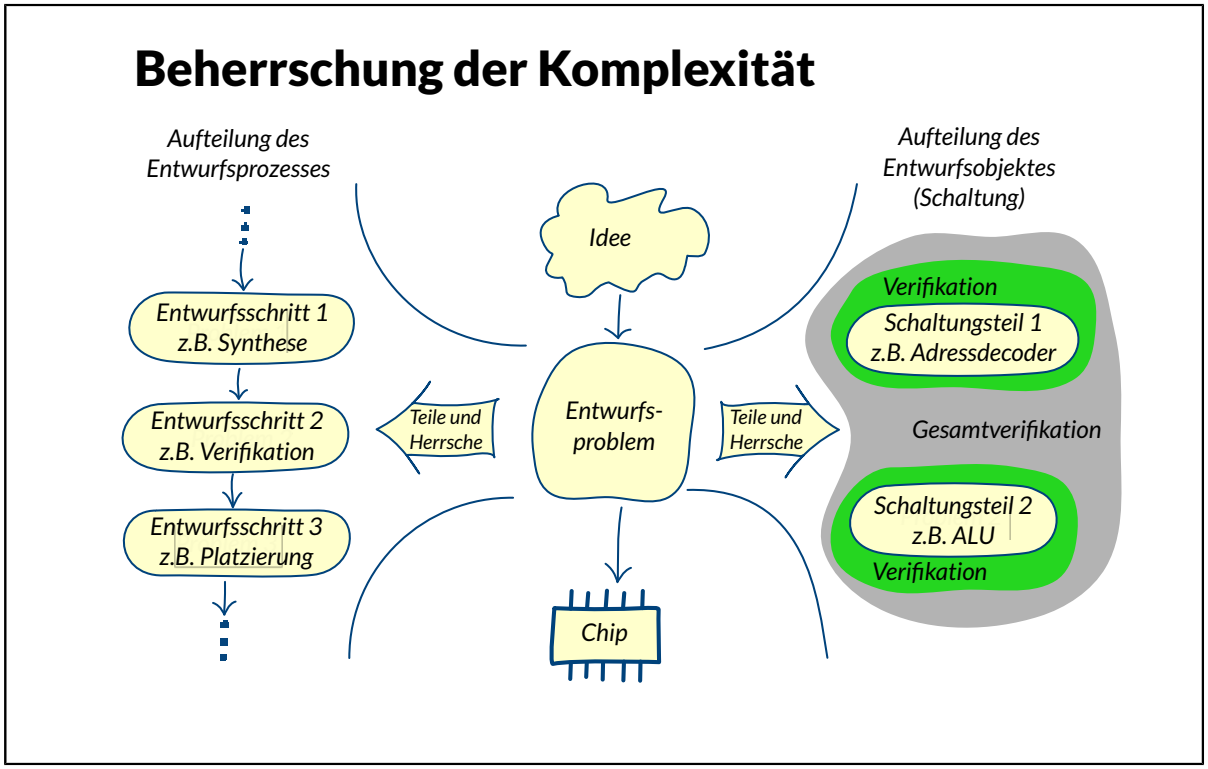
Auf die Fertigung folgt die so genannte Endprüfung, die bei integrierten Schaltungen auch als Test bezeichnet wird. Der Test hochkomplexer Schaltungen stellt eine besondere Herausforderung dar. Es ist deshalb erforderlich, bereits während des Entwurfs Maßnahmen vorzusehen, die einen späteren Test sinnvoll ermöglichen. Dazu gehört die Berechnung geeigneter Testmuster und Maßnahmen zur Erhöhung der Testbarkeit (Design for Testability, DFT).

Entwurfsprozess: Entwurf als Optimierungsproblem



Ziel des Entwurfsprozesses ist es zunächst, eine Schaltung zu entwerfen, die die Spezifikation erfüllt und diese Schaltung in einen Satz von so genannten Maskenbeschreibungen (Layout) umzusetzen, mit der die Schaltung gefertigt werden kann. Wegen der zahlreichen Freiheitsgrade im Entwurf wird es eine nahezu unendlich große Menge von Layouts geben, deren zugehörige Schaltungen die Spezifikation erfüllen. Es ist deshalb wichtig, den Entwurfsprozess als Optimierungsvorgang zu verstehen, der aus den zahlreichen Entwurfsalternativen die bestmögliche auswählt. Zu den Optimierungskriterien zählen dabei die Performance und die Chipfläche, die bereits bei der Erläuterung des Entwurfsprozesses erwähnt wurden, aber auch die Verlustleistung, deren Minimierung besondere Bedeutung zukommt, die Robustheit gegen Fertigungsschwankungen, die Testbarkeit und viele andere. Bei der Besprechung der einzelnen Entwurfswerkzeuge werden wir genauer erkennen, wie dieser Optimierungsvorgang verläuft. Sehr häufig werden wir auf gegenläufige Optimierungsziele stoßen, so dass ein geeigneter Kompromiss zu finden ist. Dies gilt in besonderem Maße für die grundlegenden Optimierungsziele Performance und Fläche.

Entwurfsprozess: Beherrschung der Komplexität

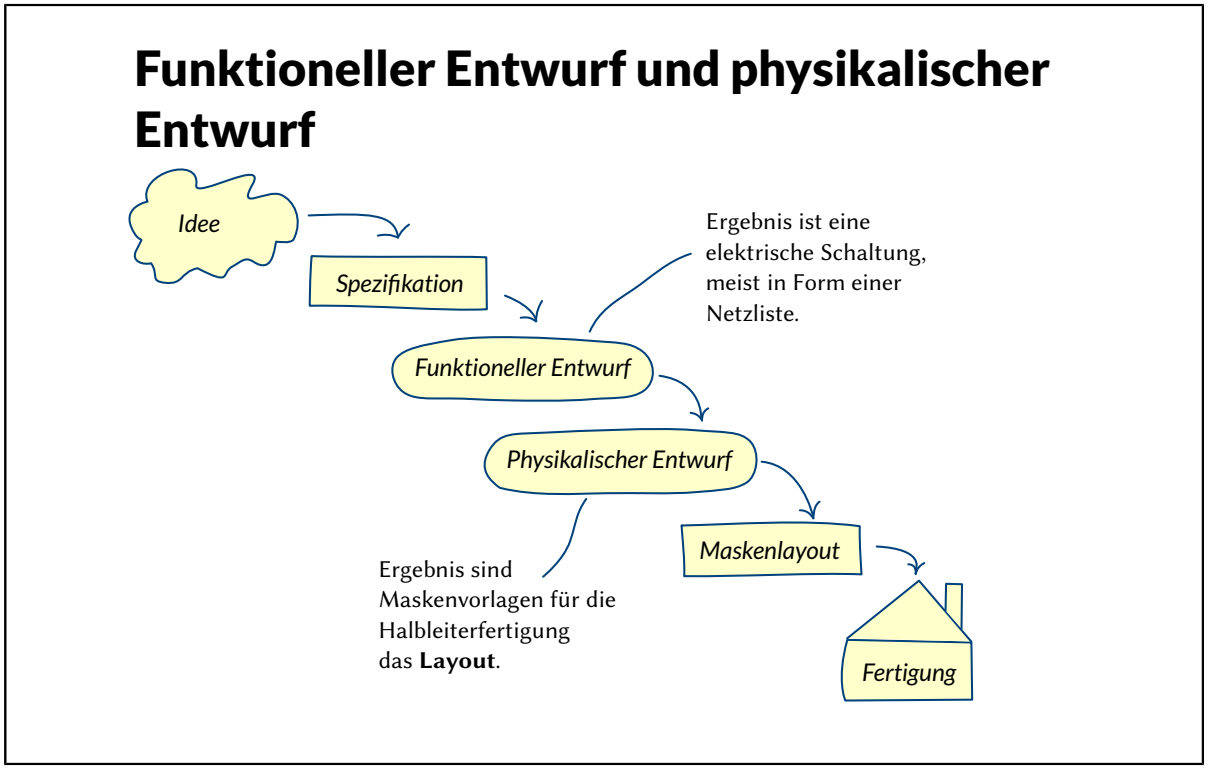


Beim Entwurf integrierter Schaltungen handelt es sich um einen Vorgang, der die Verwaltung und die systematische Behandlung von Millionen bzw. Milliarden einzelner Objekte betrifft. Es ist unschwer einzusehen, dass dieser Vorgang selbst ein äußerst komplexes Unternehmen darstellt, das nur durch eine systematische Durchführung beherrschbar ist. Daher ist es das Ziel von EDA, eine vollständige Entwurfsautomatisierung zu erreichen, d.h. eine weitestgehend ohne Benutzeraktion durchgeführte Transformation einer möglichst abstrakten Entwurfsspezifikation in eine Maskengeometrie.

Da ein vollständig automatischer Entwurf mit gleichzeitiger Garantie der Korrektheit des Entwurfsergebnisses zumindest heute utopisch erscheint, haben sich in der Praxis Entwurfsstrategien herausgebildet, die auf dem Stand der Technik das Komplexitätsproblem beherrschbar machen. Die beiden wichtigsten Strategien sind:

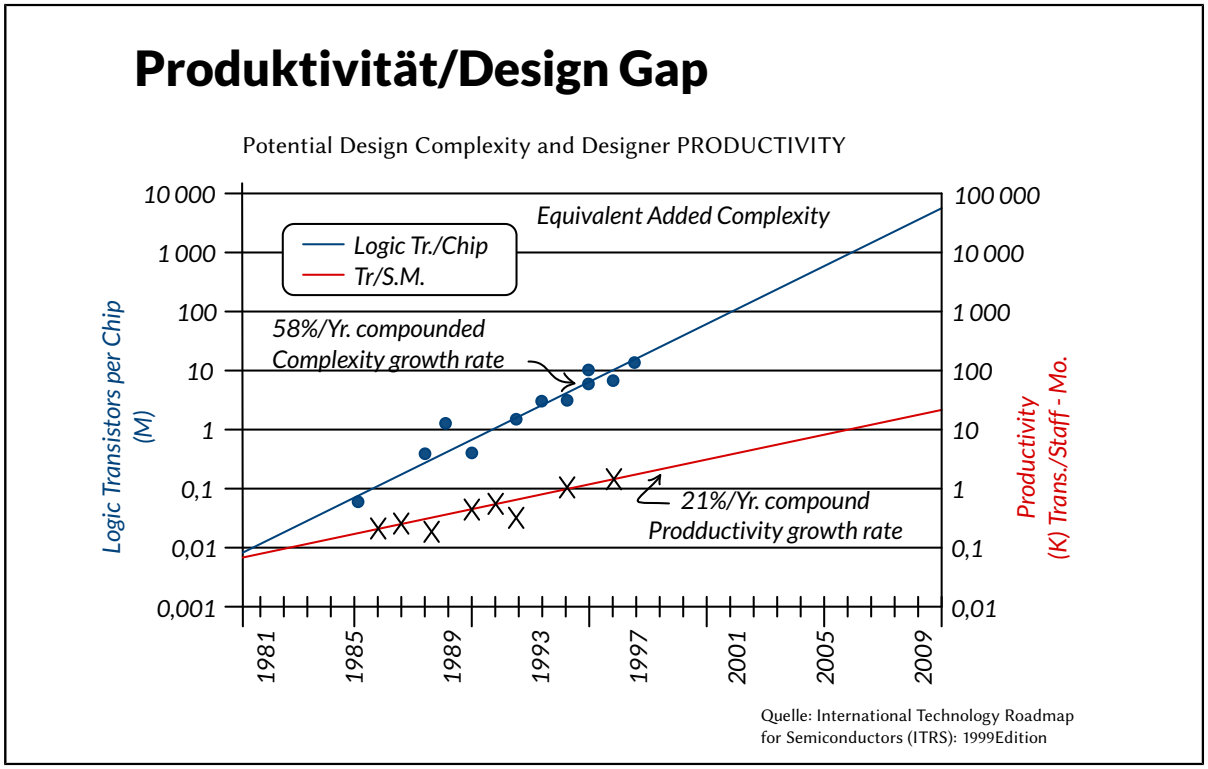
- Aufteilung des Entwurfsprozesses in einzelne Entwurfsschritte (Teile und Herrsche-Prinzip, divide and conquer). Dieses Prinzip wird einerseites bei der Aufteilung des gesamten Entwurfsvorgangs in einzelne Entwurfsschritte und andererseites bei der Aufteilung einer großen Schaltung in kleinere Teile mit sequentieller Abarbeitung verwendet.
- Einbau von Prüfschritten im Anschluss an jeden Entwurfsschritt (Verifikation). Der Entwurfsprozess kann als kontinuierliche Folge von synthetisierenden und analysierenden Schritten (Entwurfs- und Prüfschritte) angesehen werden.

Entwurfsprozess: Funktioneller und physikalischer Entwurf



In der Praxis hat sich eine Aufteilung des Entwurfsprozesses in zwei Phasen, den so genannten funktionellen und den physikalischen Entwurf bewährt. Der von der Spezifikation ausgehende funktionelle Entwurf endet mit dem Vorliegen einer elektrischen Schaltung, meist in Form einer so genannten Netzliste, im physikalischen Entwurf wird diese dann in ihre geometrische Repräsentation, die für die Fertigung erforderlichen Maskenvorlagen, übersetzt. Dabei sind – in wesentlich stärkerem Umfang als im funktionellen Entwurf – die Randbedingungen der zugrundegelegten Herstellungstechnologie zu beachten.

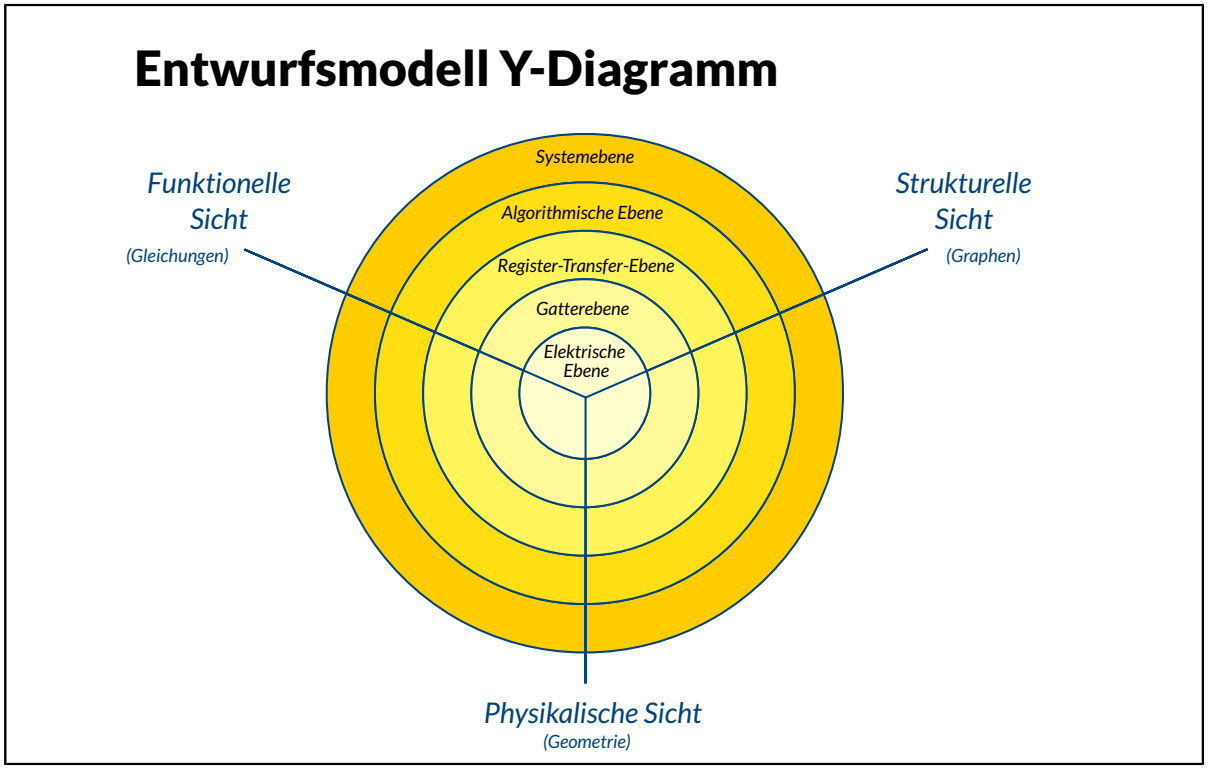
Entwurfsprozess: Produktivität



Kernaufgabe der EDA-Werkzeuge im Entwurfsprozess ist die Beherrschung der Komplexität des Problems (Machbarkeit). Daneben stellen sie ein wesentliches Hilfsmittel zur Steigerung der Entwurfsproduktivität (Effizienz) dar. Leider ist eine quantitative Ermittlung der Produktivität im Entwurf schwierig; alle Rechenmodelle dazu sind stark umstritten. Ein primitiver Ansatz wählt als Produktivitätsmaß die Zahl der von einem Entwickler pro Tag entworfenen Transistoren. Zwar ist es unstrittig, dass dieser Wert durch EDA Werkzeuge stark beeinflusst wird; gleichzeitig sind jedoch auch andere Effekte (Technologie, Designstil, Wiederverwendung von Schaltungsteilen, Fähigkeit der Entwickler u.a.) wirksam, die es nahezu unmöglich machen, den EDA-Einfluss zahlenmäßig nachzuweisen.

In industriellen Studien wurde gezeigt, dass die Entwurfsproduktivität in den letzten zehn Jahren deutlich langsamer gewachsen ist als die Problemgröße (gemessen in der Anzahl der Transistoren pro IC). Die sich damit öffnende Schere (Design Gap) wieder zu schließen, erfordert gewaltige Anstrengungen (und Aufwendungen) für die Entwurfsmethodik und die EDA-Werkzeuge. Diese Anstrengungen müssen vor allem den Entwurfsschritten gelten, bei denen der Automatisierungsgrad heute noch relativ gering ist.

Entwurfsprozess: Entwurfsmodell Y-Diagramm

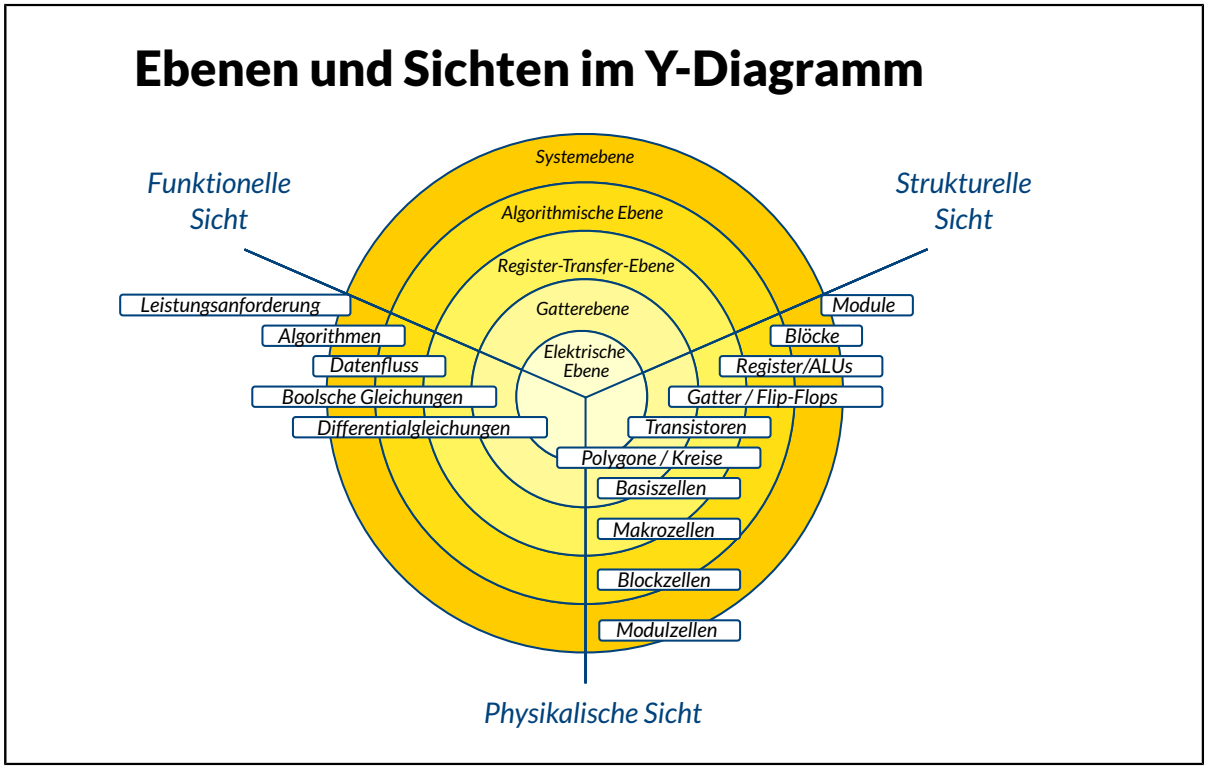


Es gibt viele Ansätze, die verschiedenen Dimensionen des Entwurfs in systematischer Weise zu einem Entwurfsmodell miteinander zu verknüpfen. Das am weitesten verbreitete und wohl auch – zumindest für das Grundverständnis – beste ist das 1983 von Gajski und Kuhn für digitale Schaltungen vorgestellte Y-Modell, das von Walker und Thomas 1985 weiter verfeinert wurde. Es bietet neben der Einfachheit und Eleganz der Darstellung den Vorteil, dass sich die einzelnen Entwurfsstile – wie wir später sehen werden – in übersichtlicher Weise graphisch darstellen lassen.

Im Y-Modell ist der Abstraktionsgrad eines Entwurfs in so genannten Entwurfsebenen dargestellt. Die verschiedenen Repräsentationen eines Entwurfs auf einer Ebene sind als drei prinzipiell unterschiedliche Sichten (funktionell, strukturell, physikalisch) dargestellt.

Es sei an dieser Stelle angemerkt, dass das hier verwendete Modell, in der konkret gewählten Terminologie und in der Bedeutung einzelner Begriffe nicht exakt mit dem Original übereinstimmt (was in unserem Falle schon wegen der Übersetzung nicht möglich ist), aber dem Grundgedanken in keiner Hinsicht widerspricht.

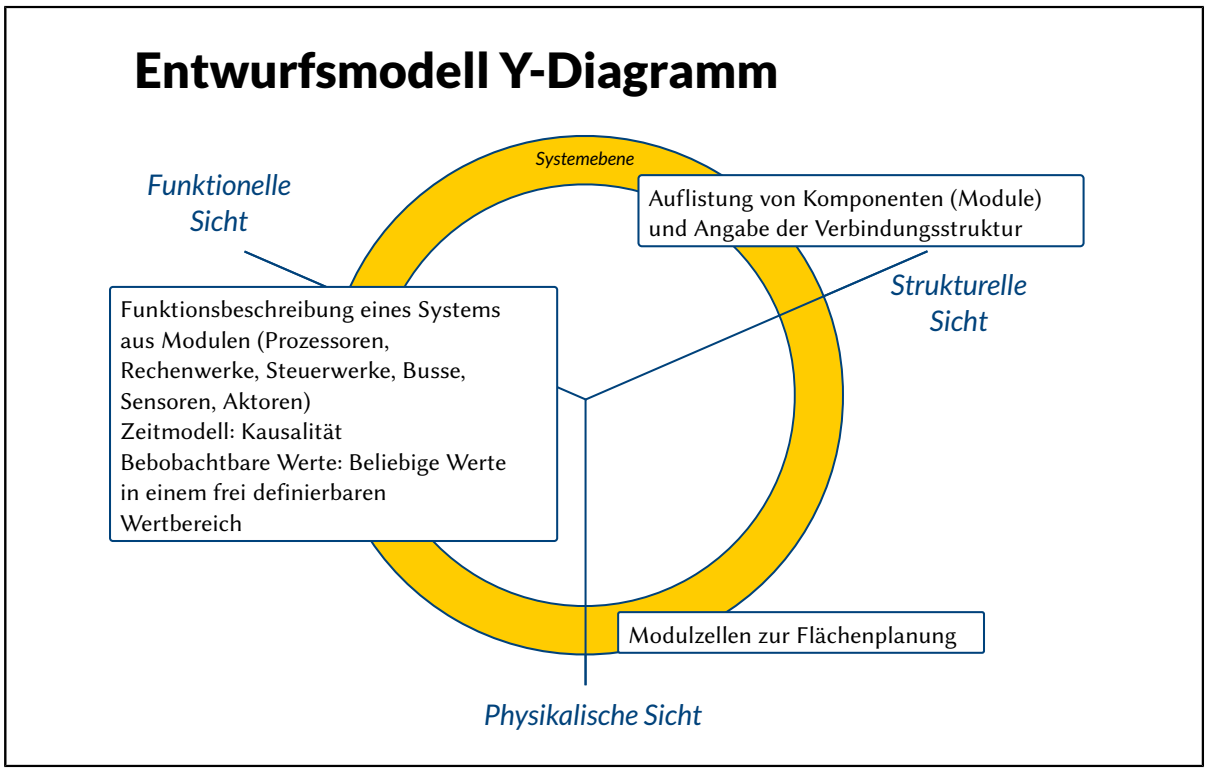
Entwurfsprozess: Ebenen und Sichten im Y-Diagramm



Das Modell zeigt drei Achsen, die in Form eines Y angeordnet sind. Sie gehen vom Zentrum einer Reihe konzentrischer Kreise aus. Jede dieser Achsen repräsentiert einen speziellen Aspekt des Entwurfsgegenstands, eine Sicht. Jede der Sichten beschreibt bestimmte Eigenschaften, die den Entwurfsgegenstand charakterisieren. Die Gesamtheit aller Sichten liefert eine vollständige Beschreibung aller relevanten Aspekte des Entwurfsgegenstands. Die funktionelle Sicht beschreibt alle Aspekte des Verhaltens des Entwurfsgegenstands. Dies beinhaltet Operationen, die vom Entwurfsgegenstand ausgeführt werden, sowie sein Zeitverhalten. Typisches Beschreibungsmittel sind mathematische Gleichungen. Die strukturelle Sicht beschreibt die logische Struktur bzw. die abstrakte Implementierung des Entwurfsgegenstands in Form der topologischen Anordnung von Komponenten und deren Verbindungen. Das dazu passende mathematische Modell sind Graphen, die häufig auf Netzlisten abgebildet werden. Die physikalische Sicht beschreibt die konkrete Implementation des Entwurfsgegenstands, das heißt die Realisierung der (strukturellen) Komponenten mit Hilfe realer physikalischer Objekte. Dies beinhaltet die exakte geometrische Ausdehnung und Anordnung aller Komponenten und Verbindungsstrukturen, mathematische Beschreibungsform ist also die Geometrie.

Es gibt Versionen dieses Entwurfsmodells, in denen weitere Aspekte des Entwurfs, wie Zeitverhalten oder Testbarkeit in weiteren Sichten repräsentiert werden. Hier werden solche Informationen jedoch als in den drei allgemeinen Sichten verteilt betrachtet. In jeder der drei Sichten besteht die Entwurfsbeschreibung aus mehreren Dokumenten, die den Entwurfsgegenstand jeweils unterschiedlich abstrakt und detailliert repräsentieren. Der jeweilige Abstraktions- bzw. Detailliertheitsgrad einer Beschreibung wird durch die Entwurfsebenen charakterisiert. In unserem Modell unterscheiden wir sechs Entwurfsebenen.

Entwurfsprozess: Systemebene



Funktionelle Sicht:

- Modellierungskonzept: Ein System, bestehend aus Modulen wie z.B. Prozessoren, Rechenwerke, Steuerwerke, Busse, Sensoren, Aktoren. Die jeweiligen Module sind charakterisiert durch ihre Funktionalität (z.B. dem Instruktionssatz), Leistungskriterien (z.B. Taktfrequenz), Kommunikationsprotokolle
- Zeitmodell: Kausalität
- Beobachtbare Werte: Beliebige Werte in einem frei definierbaren Wertebereich

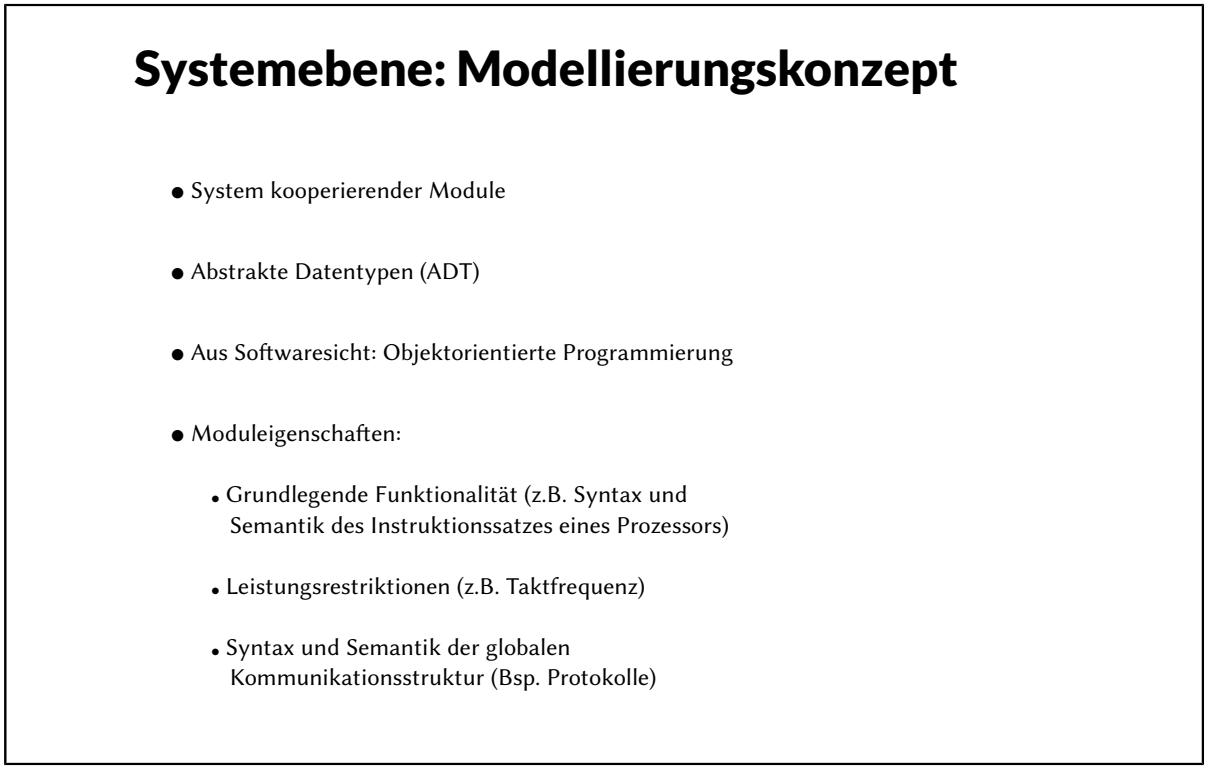
Strukturelle Sicht:

- Auflistung von Komponenten (Module) und Angabe der Verbindungsstruktur

Physikalische Sicht:

- Modulzellen, mit denen Flächenplanung im weiteren Sinn durchgeführt werden kann

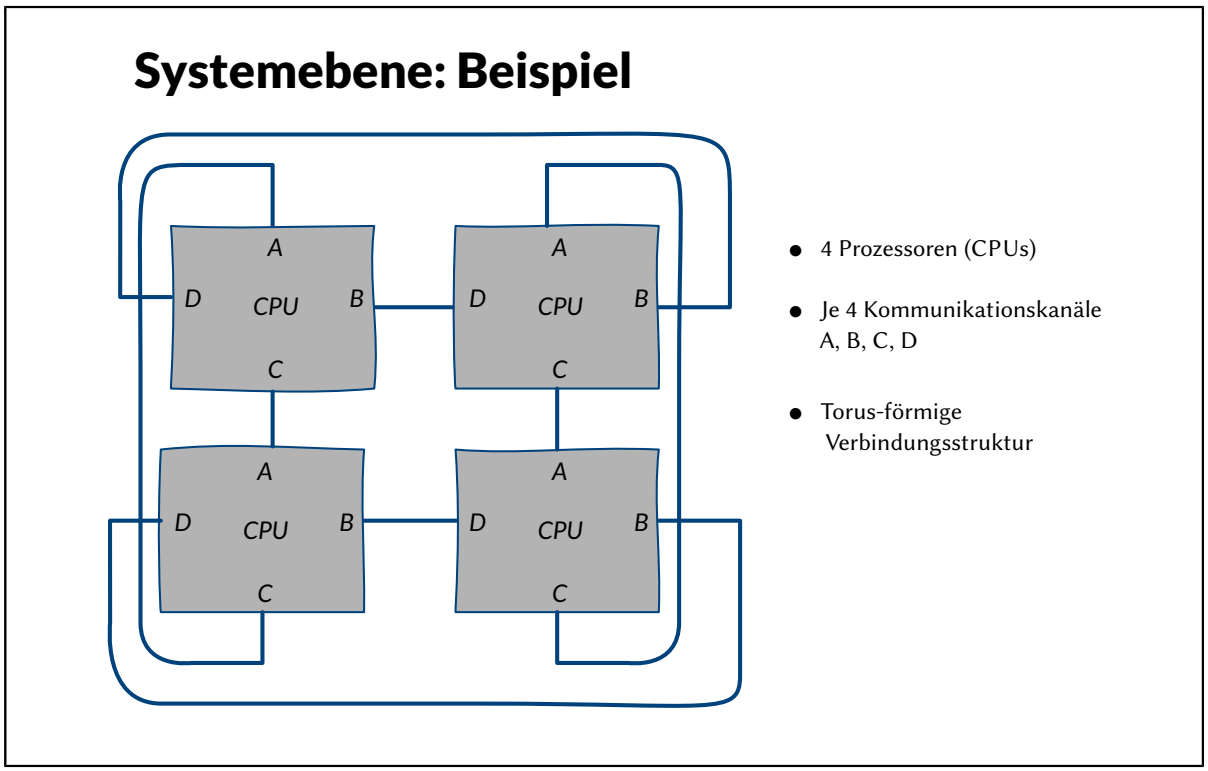
Entwurfsprozess: ... Modellierungskonzept



Das Modellierungskonzept auf dieser Ebene für die funktionelle Sicht (Verhalten) ist durch ein System kooperierender Module gegeben. Aus einer theoretischen Sichtweise sind abstrakte Datentypen (ADT) gut geeignet, als konzeptionelles Modell für diese Ebene zu dienen. Wenn man sich aus dem Bereich der Software nähert, so passt das Konzept der objektorientierten Programmierung sehr gut. Drei hauptsächliche Eigenschaften müssen pro beteiligtem Modul spezifiziert werden:

- Für jedes Modul muss die grundlegende Funktionalität angegeben werden. Dies bedeutet z.B. bei einem Prozessor nichts anderes, als Syntax und Semantik seines Instruktionssatzes zu definieren.
- Die grundlegenden Restriktionen auf dieser Ebene sind solche bezüglich der Leistung. Sie können global oder pro beteiligtem Modul spezifiziert sein, z.B. die Taktfrequenz.
- Schließlich müssen Syntax und Semantik der globalen Kommunikationsstruktur angegeben werden. Dies geschieht durch Definition von Protokollen für jede existierende Kommunikationsverbindung (Bus).

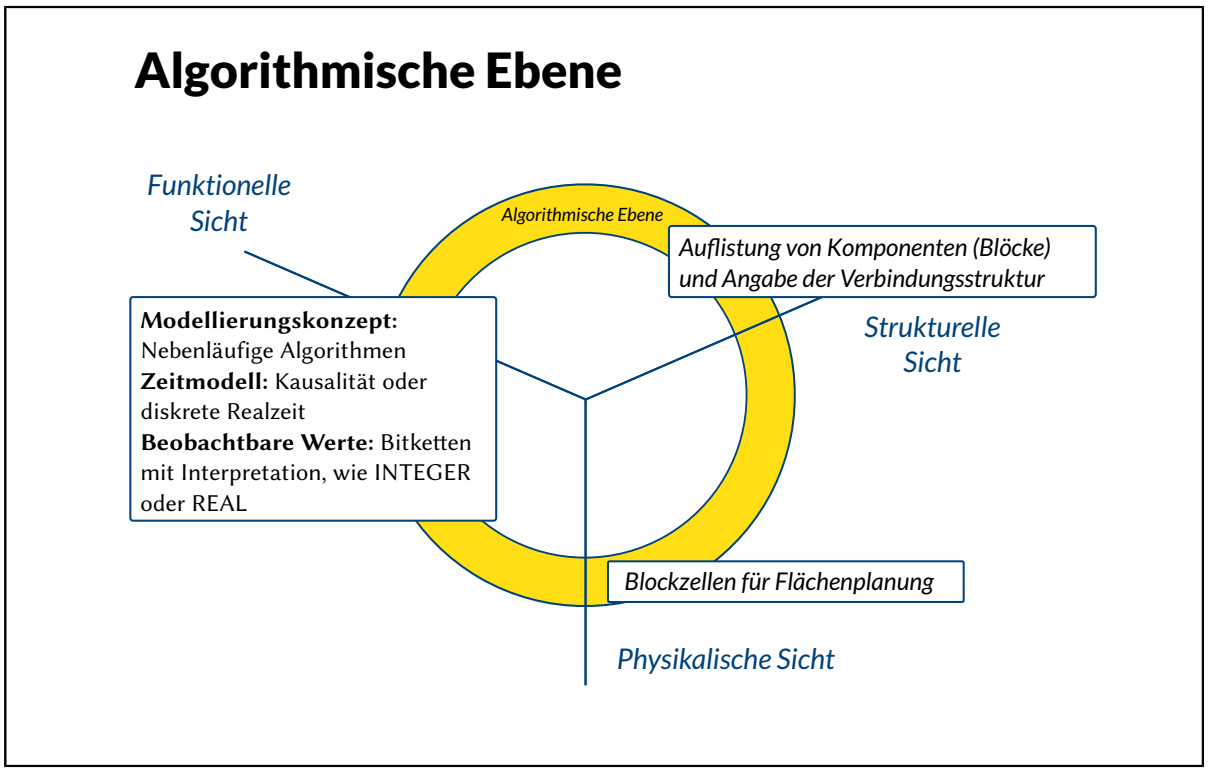
Entwurfsprozess: ... Beispiel



Die Struktur auf dieser Ebene ist durch einfaches Auflisten der beteiligten Komponenten und durch Angabe der Verbindungsstruktur gegeben. Diese Verbindungen (zumeist Busse) können auf dieser Ebene ebenfalls als Komponenten angesehen werden. Ihnen werden in der Verhaltenssicht die Kommunikationsprotokolle zugeordnet. Neben dieser statischen Struktur kann zusätzlich eine dynamische existieren. Sie gibt pro Modul an, von welchen anderen Modulen es Dienste anfordert und für welche es Dienste anbietet.

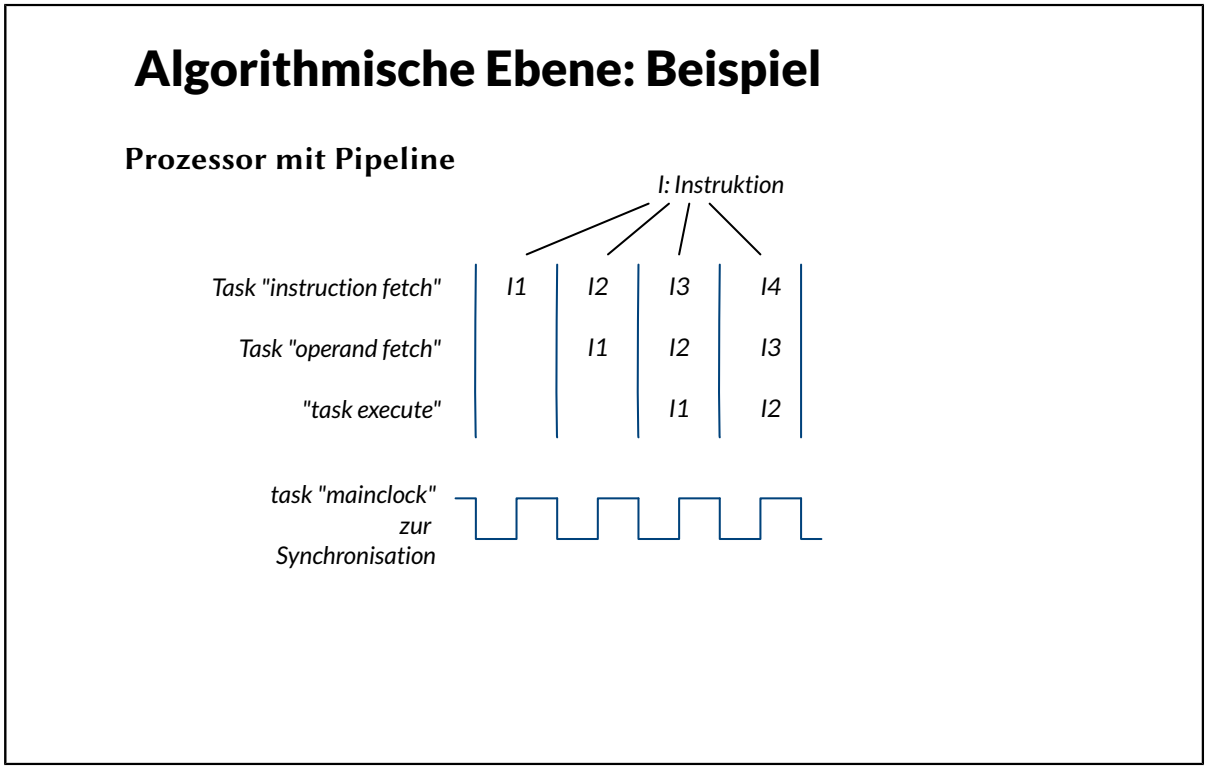
Auf dieser Ebene existiert sehr wenig geometrische Information. Andererseits finden grundlegende mechanische Entscheidungen auf dieser Ebene statt. So kann auf dieser Ebene beispielsweise die dreidimensionale Anordnung der Komponenten bestimmt werden.

Entwurfsprozess: Algorithmische Ebene



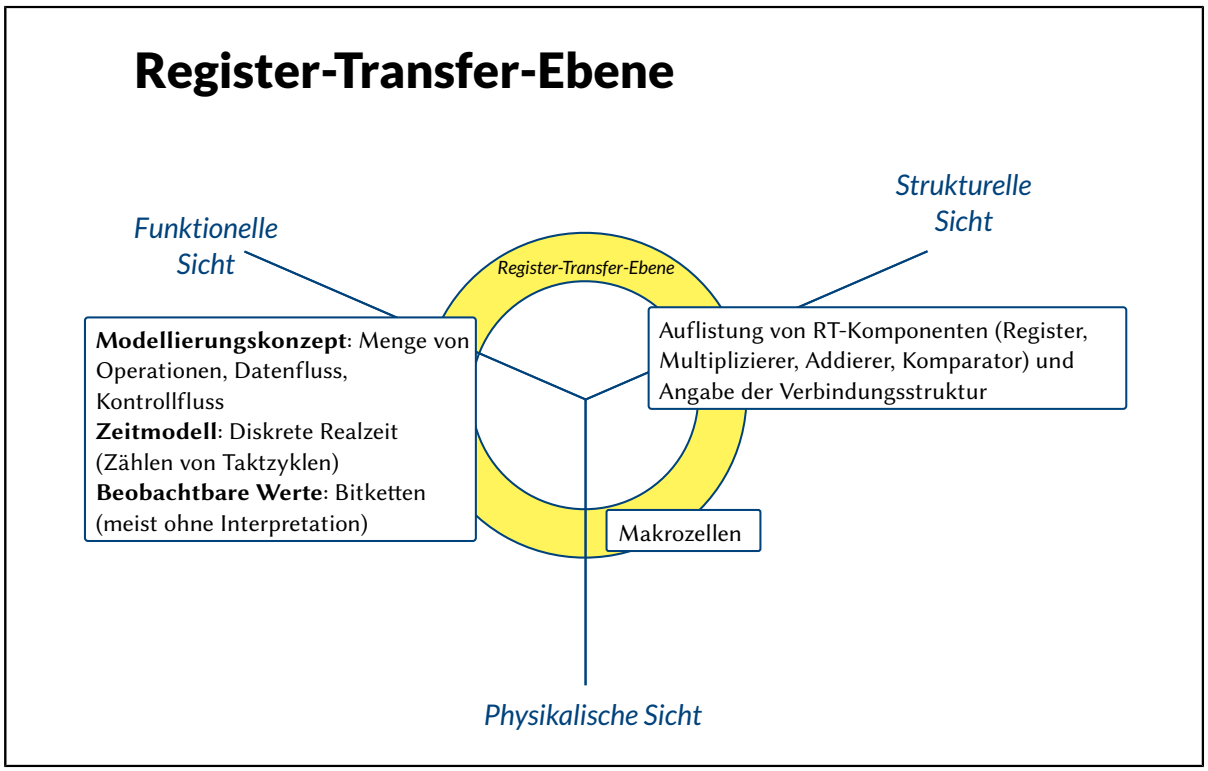
Algorithmen sind üblicherweise hochgradig nebenläufig. Daher erscheinen Modellierungskonzepte wie Petri-Netze oder Datenflussgraphen für diesen Zweck gut geeignet. Bezüglich des Zeitmodells interessiert man sich auf dieser Ebene in den meisten Fällen weiterhin nur für Kausalitäten. Allerdings wird in manchen Fällen ein diskretes Zeitmodell angenommen, wobei man ein bestimmtes Taktschema im Auge hat. Die beobachtbaren Werte sind konkreter als auf der Systemebene. Ihre Eigenschaft als Bitketten ist nun in den meisten Fällen sichtbar. Doch ist dies weiterhin von geringem Interesse, man konzentriert sich auf die typspezifische Interpretation (z.B. als Integer). Bezüglich der Struktur müssen wir zwischen der Komposition einer Kontrollstruktur aus Komponenten wie "While-Schleife" oder "Fork/Join" (Kontrollpfad) und der Struktur des Operationsteils des Algorithmus (Datenpfad) unterscheiden. Im letzteren Fall werden auf dieser Ebene bereits Hardware-Komponenten wie z.B. ALUs benutzt. Die geometrischen Informationen sind auf dieser Ebene immer noch recht abstrakt, jedoch findet hier bereits auf der Basis von Blockzellen eine konkrete Flächenplanung statt.

Entwurfsprozess: ... Beispiel



Angenommen wird ein gewöhnlicher Prozessor vom von-Neumann-Typ. Er habe einen Interpretationszyklus, bestehend aus "instruction fetch", "operand fetch" und "execute". Weiterhin wird angenommen, dass diese drei Aktivitäten im Pipelining ablaufen, also nebenläufig. Sie sollen mittels eines Taktes, genannt "mainclock", synchronisiert werden.

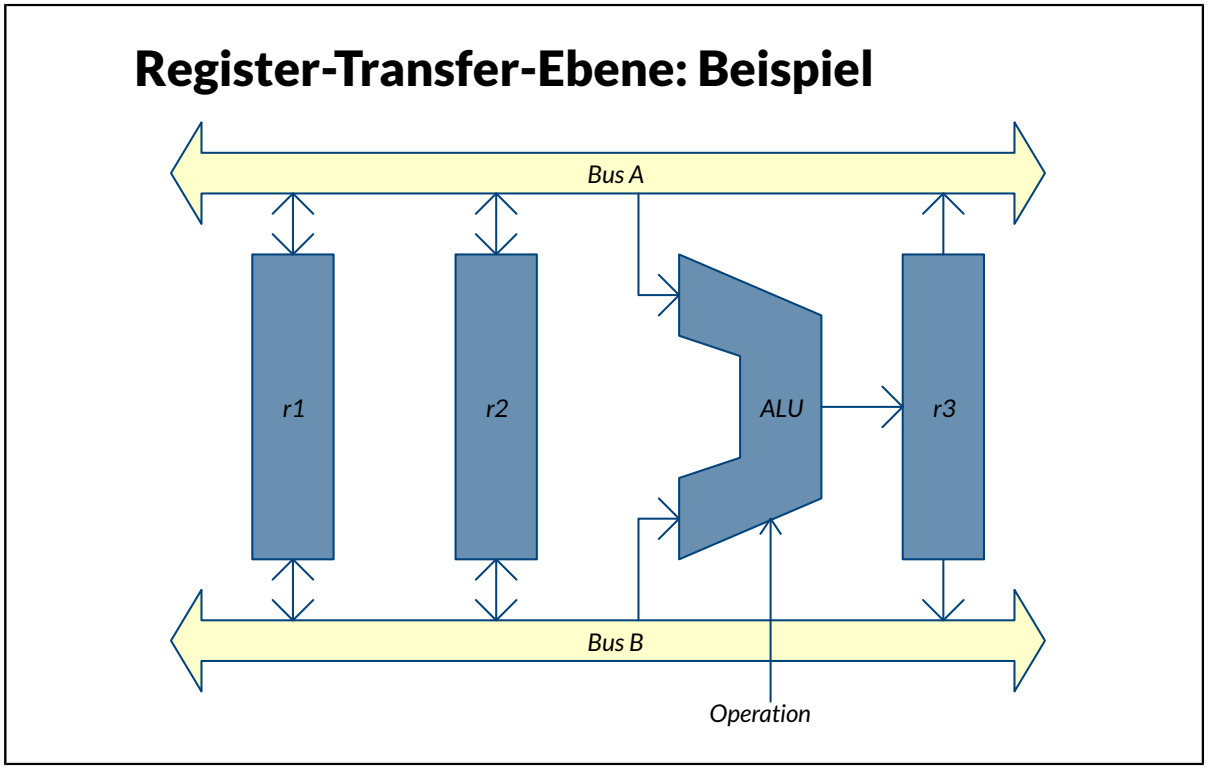
Entwurfsprozess: Register-Transfer-Ebene



Auf der algorithmischen Ebene wird das System meist in einer imperativen Weise betrachtet. Das heißt, dass die Sichtweise die des Steuerwerks ist (Kontrollfluss). Dieses entscheidet, wann nach welchen vorhergehenden Aktionen eine bestimmte Aktion durchgeführt werden darf. Die strikt sequentielle Anordnung in üblichen Programmiersprachen wird hier generalisiert, um auch Nebenläufigkeit zu erlauben. Auf der Registertransferebene (RT-Ebene; englisch Register-Transfer-Level, RTL) wird eine reaktive Sichtweise eingenommen. Das System wird nun aus Sicht der gesteuerten Objekte (Rechenwerke) betrachtet (Datenfluss). Jedes derartige Objekt beobachtet kontinuierlich eine objektspezifische Bedingung. Ist diese Bedingung wahr, führt das Objekt seine Aktion durch. Dabei modifiziert es üblicherweise die Bedingungen innerhalb des Bedingungsraumes, und kann die Ausführung anderer Objekte (einschließlich seiner selbst) ermöglichen.

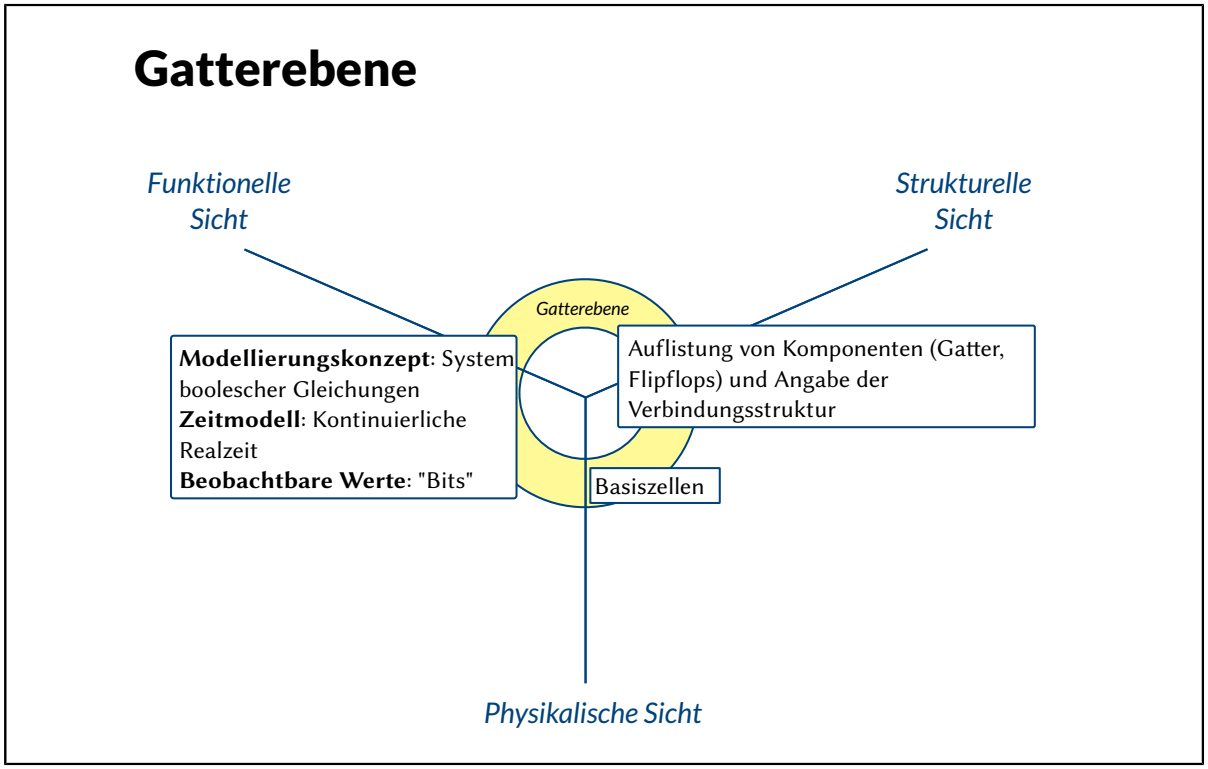
Auf dieser Ebene ist man üblicherweise an einem spezifischen synchronen Taktschema interessiert. Daher wird das Zeitschema auf dieser Ebene meist durch das Zählen von Taktzyklen gegeben. Es hängt von dem jeweiligen Implementierungskonzept für das Zeitschema ab, ob Taktpegel, steigende, fallende oder beide Flanken, oder gar eine Mischung dieser Techniken benutzt werden. Die beobachtbaren Werte sind nun Bitketten. In den meisten Fällen wird ihnen nicht mehr eine feste Interpretation (Typ) zugeordnet. Vielmehr werden sie von verschiedenen Objekten unterschiedlich interpretiert. Auf der RT-Ebene wird die endgültige Hardwarestruktur sichtbar. Das System wird daher als Verschaltung von Registertransfermodulen beschrieben. Typische derartige Module sind Register, ALU's, Multiplexer, Kodierer, Dekodierer, Schiebebausteine, etc. Aus physikalischer Sicht findet auf der RT-Ebene die endgültige Flächenplanung statt. Zunehmend häufig existieren für strukturelle Komponenten auch bereits physikalische Realisierungen (Makrozellen).

Entwurfsprozess: ... Beispiel



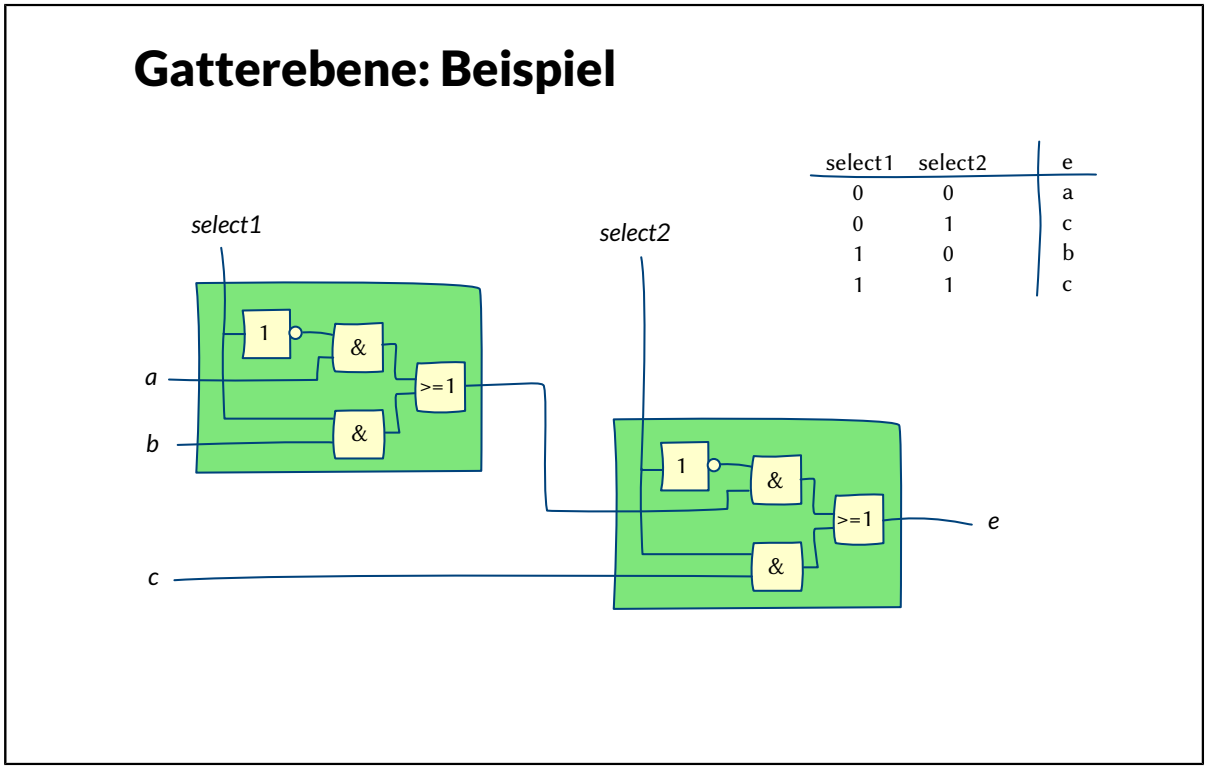
Dargestellt ist ein System bestehend aus zwei Bussen, zwei Registern, die je mit beiden Bussen bidirektional verbunden sind, und einer ALU, die von beiden Bussen gleichzeitig gelesen werden kann und ihr Ergebnis über eine dedizierte Verbindung in ein drittes Register schreibt. Dieses dritte Register kann auf beide Busse schreiben.

Entwurfsprozess: Gatterebene



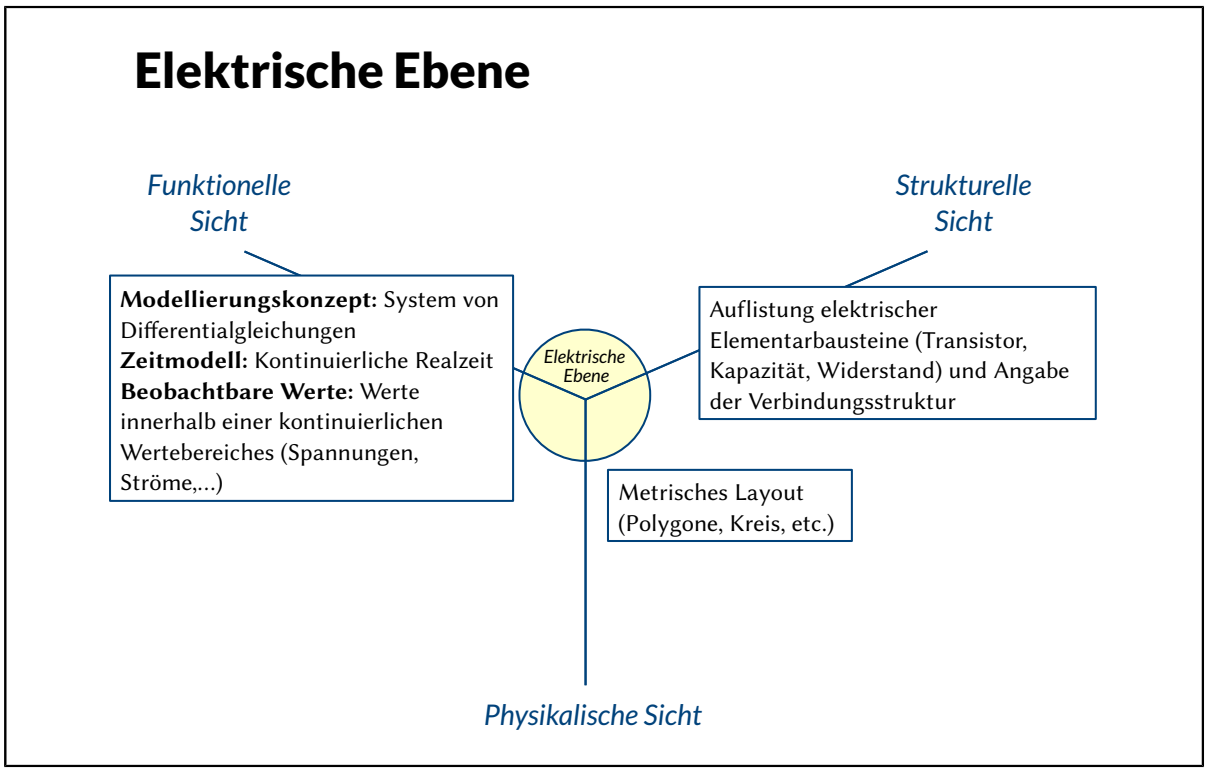
Beschreibungen auf der Gatterebene können als Erhöhung des Detailierungsgrads der Module auf der RT-Ebene angesehen werden. Allerdings geht die semantische Information über die Unterscheidung zwischen Daten- und Kontrollsignalen, wie sie auf der RT-Ebene noch vorhanden ist, vollständig verloren. Man hat nun lediglich ein Netz mit Gattern und Flipflops als Knoten und Einbit-Verbindungsleitungen als Kanten. Auf dieser Ebene ist man in vielen Fällen an präzisen Informationen über das Zeitverhalten interessiert. Daher ist das übliche Zeitmodell auf dieser Ebene das der kontinuierlichen Realzeit. Allerdings werden verschiedene approximative Verzögerungskonzepte benutzt, die von simplen Konzepten wie feste Nominalverzögerung bis hin zu Modellen reichen, die fast das analoge Verhalten der beteiligten Module widerspiegeln. Beobachtbare Werte sind "Bits" in einer mehrwertigen Logik. Die Knoten müssen nicht auf "Gatter" im engeren Sinn beschränkt sein, sondern sind Schaltungen, die durch einen booleschen Ausdruck bzw. durch ein Bündel boolescher Ausdrücke beschrieben werden können. Dadurch sind auch hierarchische Beschreibungen möglich. In der physikalischen Sicht stehen auf der Gatterebene häufig Bibliotheken mit so genannten Basiszellen zur Verfügung.

Entwurfsprozess: ... Beispiel



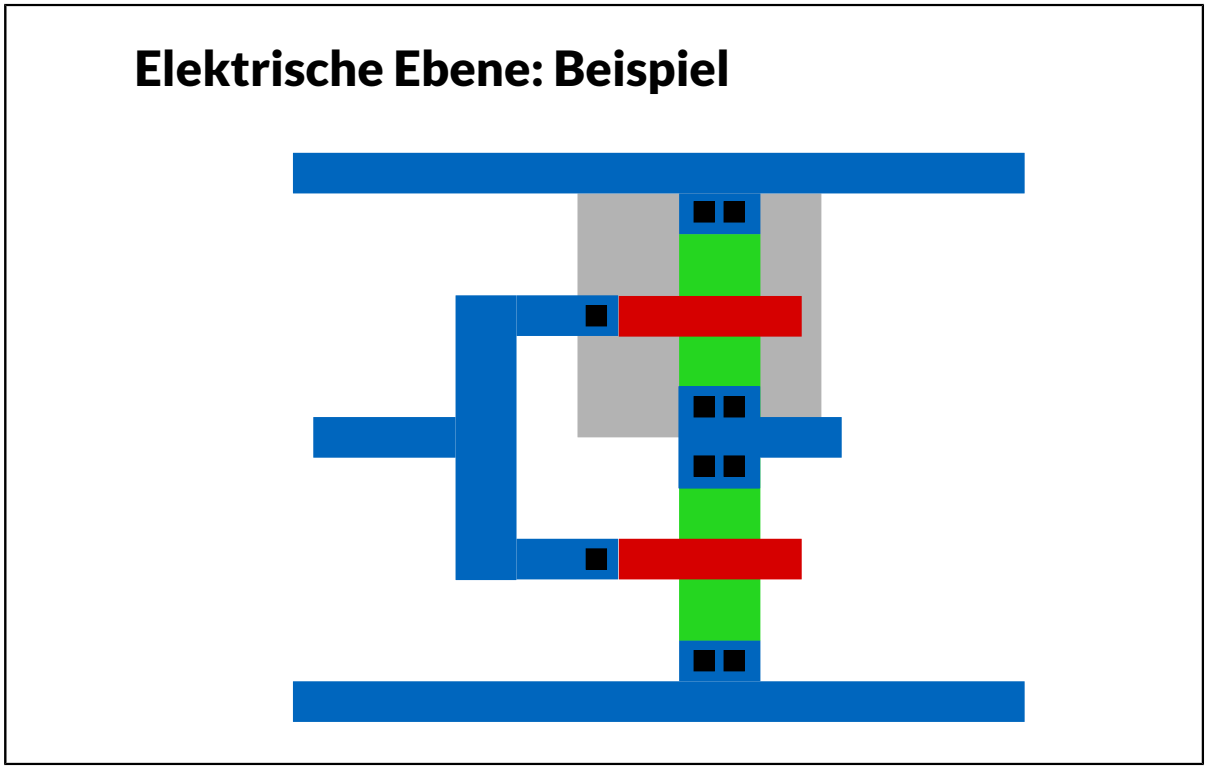
Die Folie zeigt einen 3-zu-1-Demultiplexer in einer möglichen Realisierung mit Standardgattern. Mit den beiden Eingangssignalen "select1" und "select2" kann aus den drei Eingängen der Schaltunge "a", "b" und "c" einer ausgewählt und auf den Ausgang "e" geschaltet werden. Gatterschaltungen können auch hierarchisch sein. Die beiden grün hinterlegten Gatteranordnungen könnten z.B. Instanzen eines 2-zu-1-Demultiplexers darstellen.

Entwurfsprozess: Elektrische Ebene



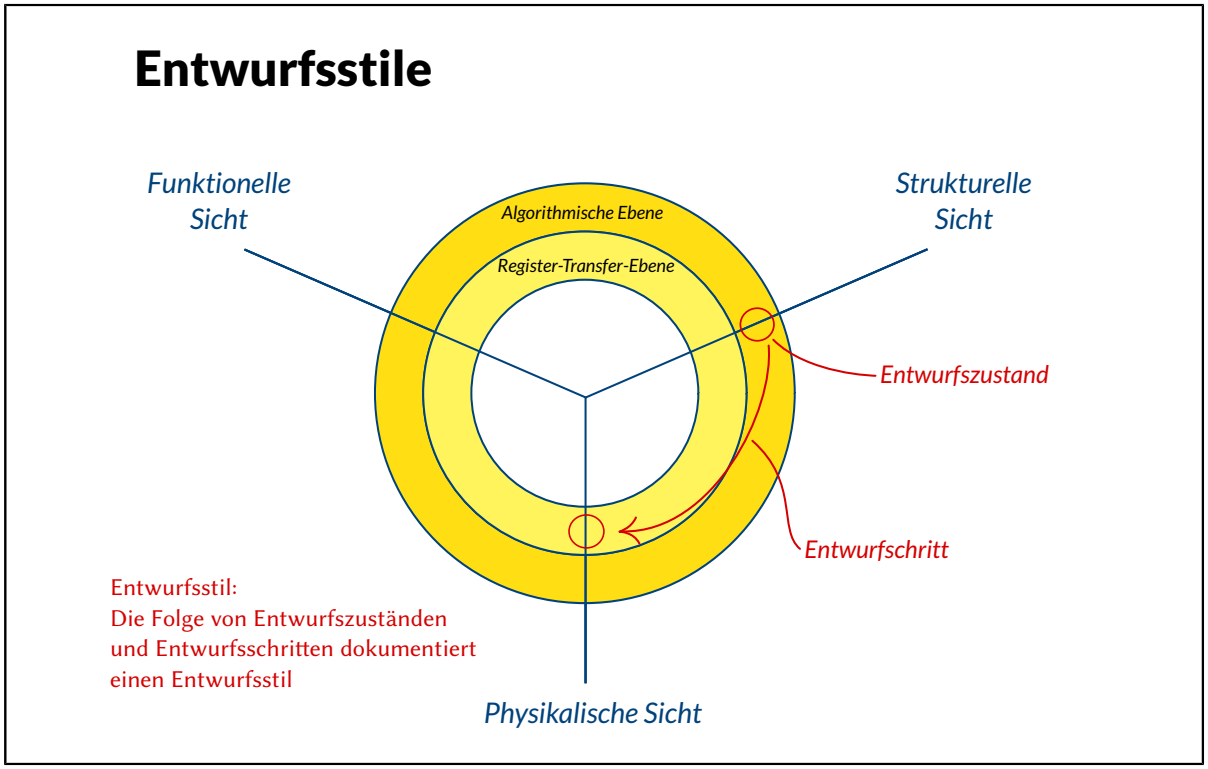
Auf dieser Ebene wird die digitale Interpretation der Schaltung aufgegeben und das analoge Verhalten betrachtet. Das Modellierungskonzept ist durch ein System von Differentialgleichungen über kontinuierlichen Wertebereichen und in kontinuierlichen Zeitbereichen gegeben. Die benutzten Elementarbausteine sind Widerstände, Kapazitäten, etc. Das metrische Layout unterscheidet sich vom symbolischen dadurch, dass jedes benutzte Objekt eine definierte Bemaßung hat.

Entwurfsprozess: ... Beispiel



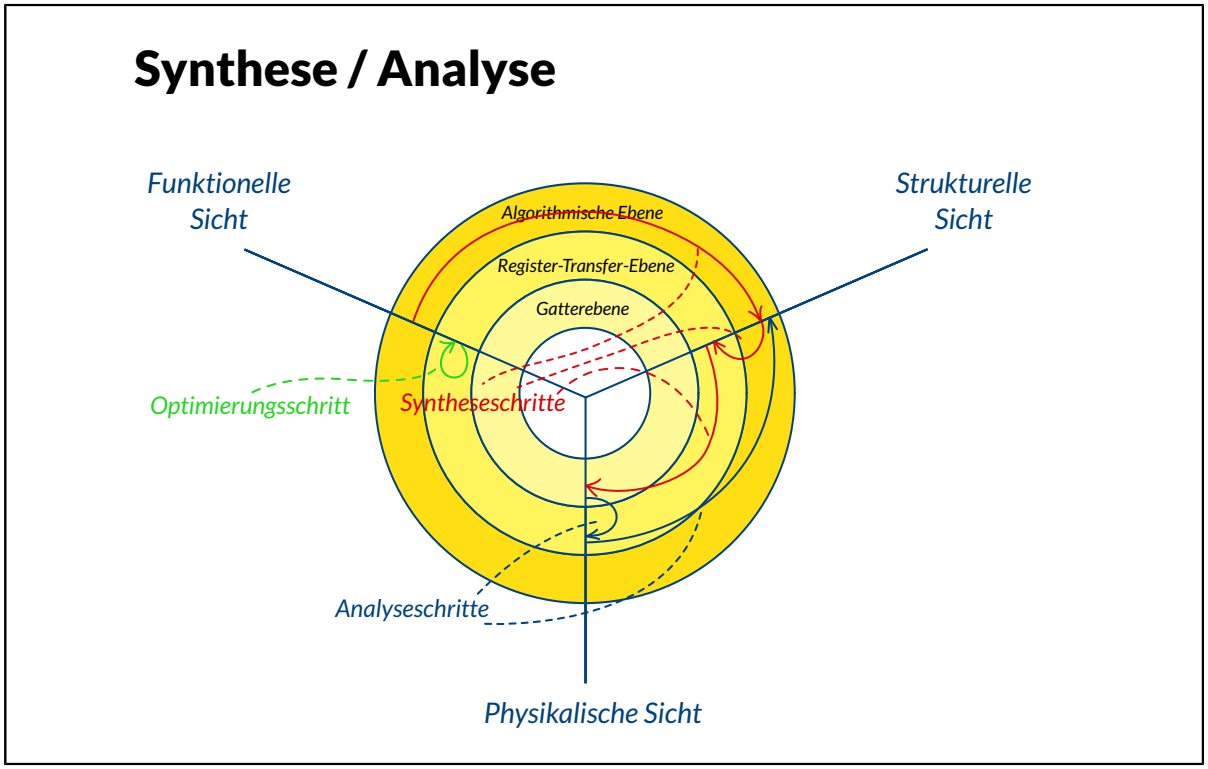
Das Beispiel zeigt die Layoutpolygone für einen CMOS-Inverter.

Entwurfsprozess: Entwurfsstile



Mit Hilfe des Y-Modells lassen sich beliebige Entwurfsstile darstellen. Jeder Entwurfszustand lässt sich durch einen Schnittpunkt zwischen einer der Achsen und einem der konzentrischen Kreise im Y-Modell repräsentieren. Ein Entwurfsschritt ist eine Abbildung zwischen zwei Entwurfszuständen. Ein Entwurfsstil ist durch eine Folge von Entwurfszuständen charakterisiert, die in der physikalischen Sicht auf der elektrischen Ebene endet. Der Ausgangspunkt des Entwurfsvorgangs, die Idee eines mikroelektronischen Systems, ist in diesem Modell gewöhnlich durch den Entwurfszustand "funktionelle Sicht - Systemebene" gekennzeichnet.

Entwurfsprozess: Synthese/Analyse



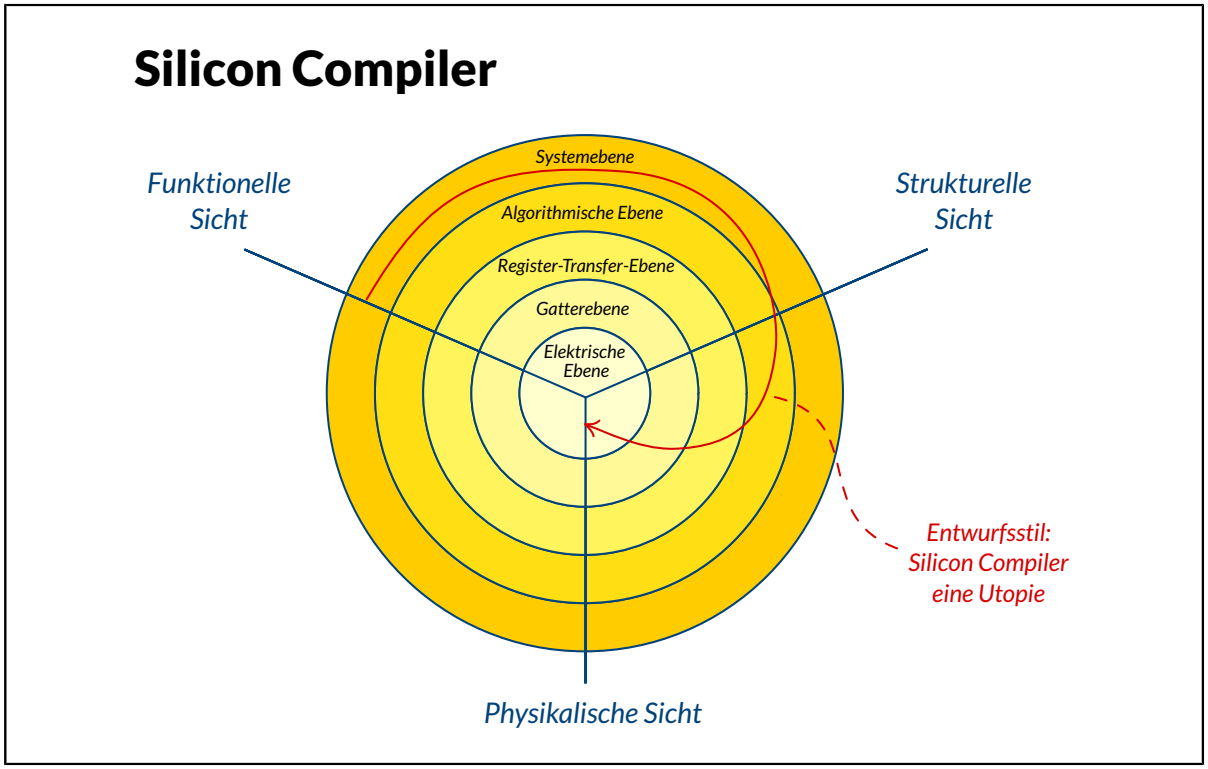
Betrachtet man einen Entwurfsschritt in Richtung auf das Entwurfsziel (ein metrisches Layout) hin, so wird die damit verbundene Abbildung als Syntheseschritt bezeichnet, betrachtet man die umgekehrte Richtung, so spricht man von einem Analyseschritt.

Ein Syntheseschritt führt das Entwurfsobjekt in einen Zustand, der der Realisierung näher liegt. Ein solcher Schritt ist gewöhnlich dadurch gekennzeichnet, dass der Abstraktionsgrad sinkt bzw. der Detailliertheitsgrad der Beschreibung steigt. Auf diese Weise wird in die Entwurfsbeschreibung neue Information eingebracht, die vorher nicht explizit vorhanden war. In diesem Sinne stellt ein Syntheseschritt einen kreativen Prozess dar. Die Disziplin der Entwurfsautomatisierung beschäftigt sich damit, Algorithmen zu entwickeln, die den kreativen Prozess der Synthese automatisieren.

Ein Analyseschritt vollzieht eine Abstraktion bzw. Extraktion. Aus einer gegebenen detaillierten Entwurfsbeschreibung werden abstrakte Informationen durch Zusammenfassen und Generalisieren von Details gewonnen. Solche Schritte dienen gewöhnlich zur Validierung eines Syntheseschritts. Dies ist insbesondere dann erforderlich, wenn ein Syntheseschritt manuell oder durch ein selbst nicht formal verifiziertes Verfahren durchgeführt wurde, so dass die "konstruktionsbedingte Korrektheit" (correctness by construction) nicht garantiert werden kann. Bei einem Analyseschritt wird immer ein Ebenenwechsel durchgeführt.

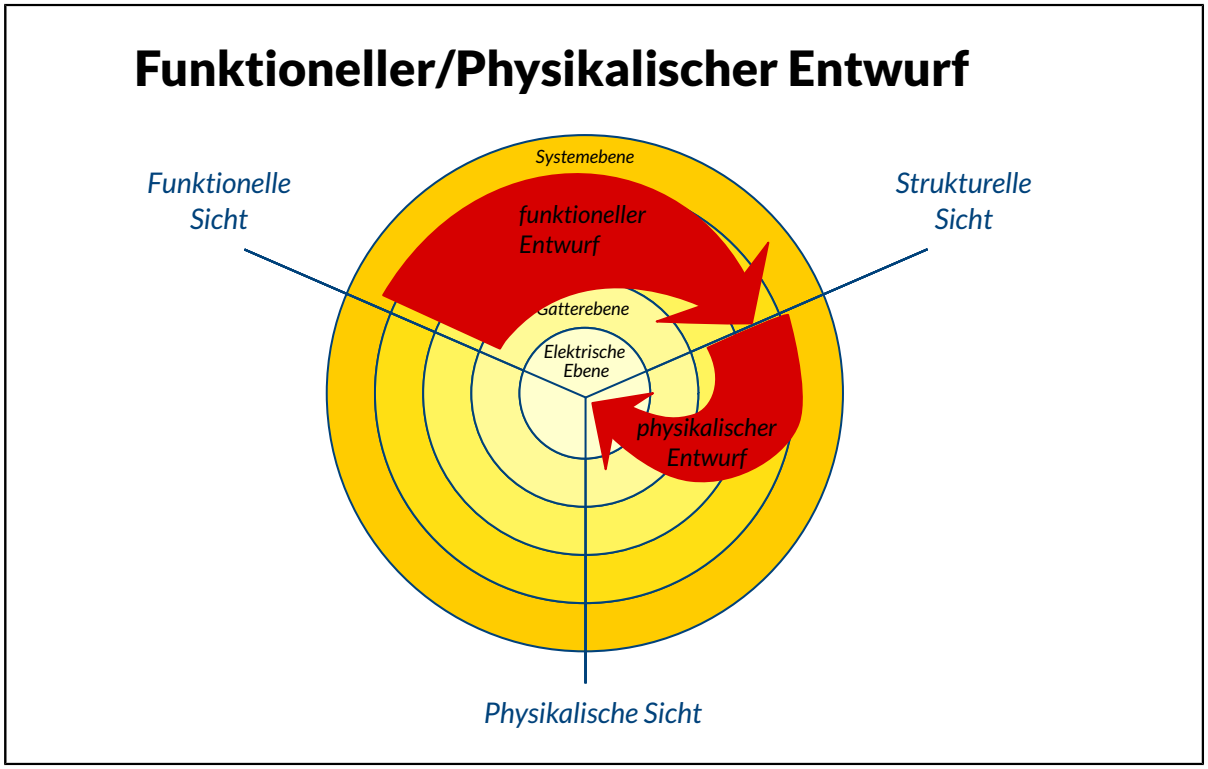
Ein Optimierungsschritt beeinhaltet keinen Ebenen- oder Sichtenwechsel. Der Entwurf wird in Hinblick auf ein (oder mehrere) Kriterien hin optimiert, um den in der Spezifikation angegebenen Anforderungen zu genügen.

Entwurfsprozess: Silicon Compiler



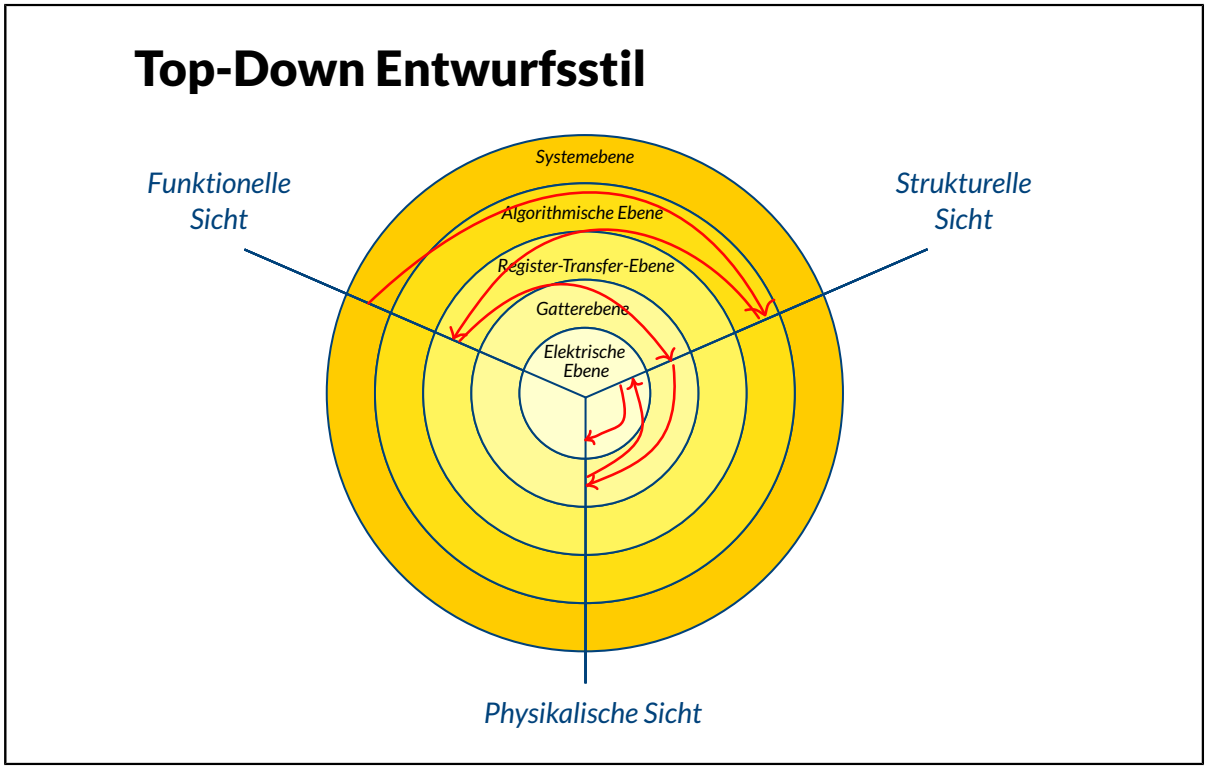
Die ideale Verwirklichung eines zeitoptimierten Entwurfsstils ist der vollautomatische Entwurf durch einen so genannten Silicon Compiler. Dieser erzeugt aus einer funktionellen Beschreibung (Spezifikation) in einer hohen Entwurfsebene automatisch ein metrisches Layout auf elektrischer Ebene. Obwohl der Begriff in der Literatur viel verwendet wurde und noch wird, erscheint seine Realisierung unter praktischen Randbedingungen nach wie vor utopisch.

Entwurfsprozess: Funktioneller/Physikalischer Entwurf



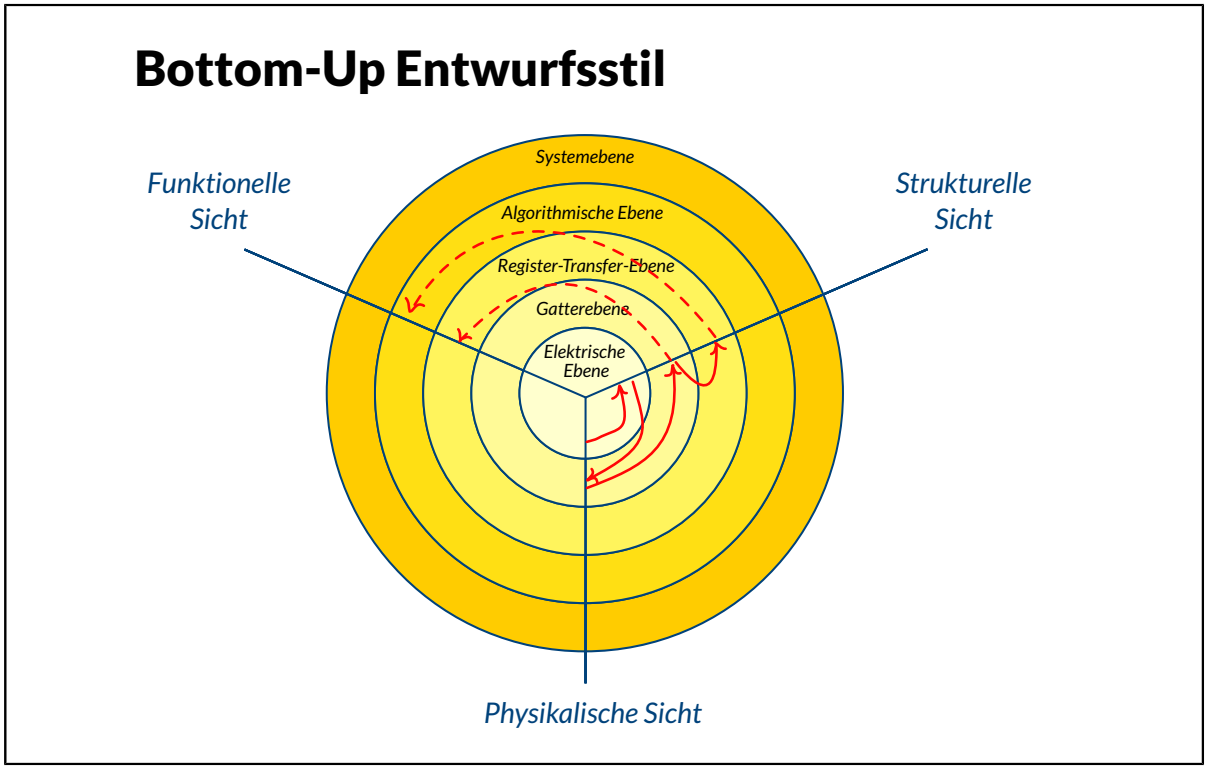
Stattdessen besteht heute ein Entwurf aus einer Vielzahl von Einzelschritten, mit denen der Weg von der Spezifikation bis zum metrischen Layout zurückgelegt wird. Wie bereits erwähnt ist dabei eine grobe Aufteilung zwischen funktionellen und physikalischen Entwurf sinnvoll. Dies kann ebenfalls in Y-Diagramm veranschaulicht werden.

Entwurfsprozess: Top Down Entwurfsstil



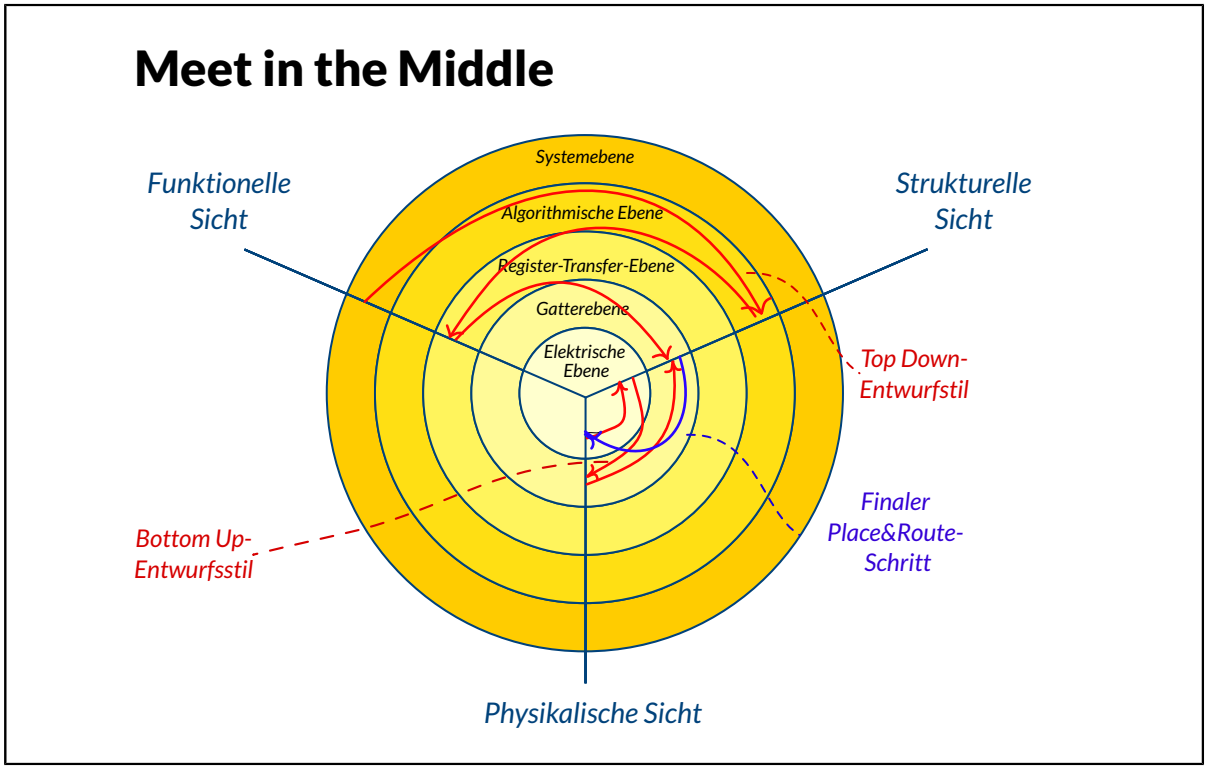
Wir haben bisher vorausgesetzt, dass der Entwurfsprozess auf hohen Entwurfsebenen beginnt und durch schrittweise Verfeinerung niedrigere Entwurfsebenen erreicht. Eine solche Vorgehensweise wird ganz allgemein als Top-Down-Entwurfsstil bezeichnet. Dieser Stil erscheint zur Beherrschung der Komplexität sinnvoll, begrenzt jedoch die Ausnutzung prinzipiell vorhandener Freiheitsgrade, da diese auf den mittleren Ebenen häufig nicht genutzt werden können, da Varianten nicht durchgepspielt werden können. Trotz allem wird der Top-Down Entwurfstil heute z.B. beim Analog-Entwurf sehr häufig noch eingesetzt.

Entwurfsprozess: Bottom Up Entwurfsstil



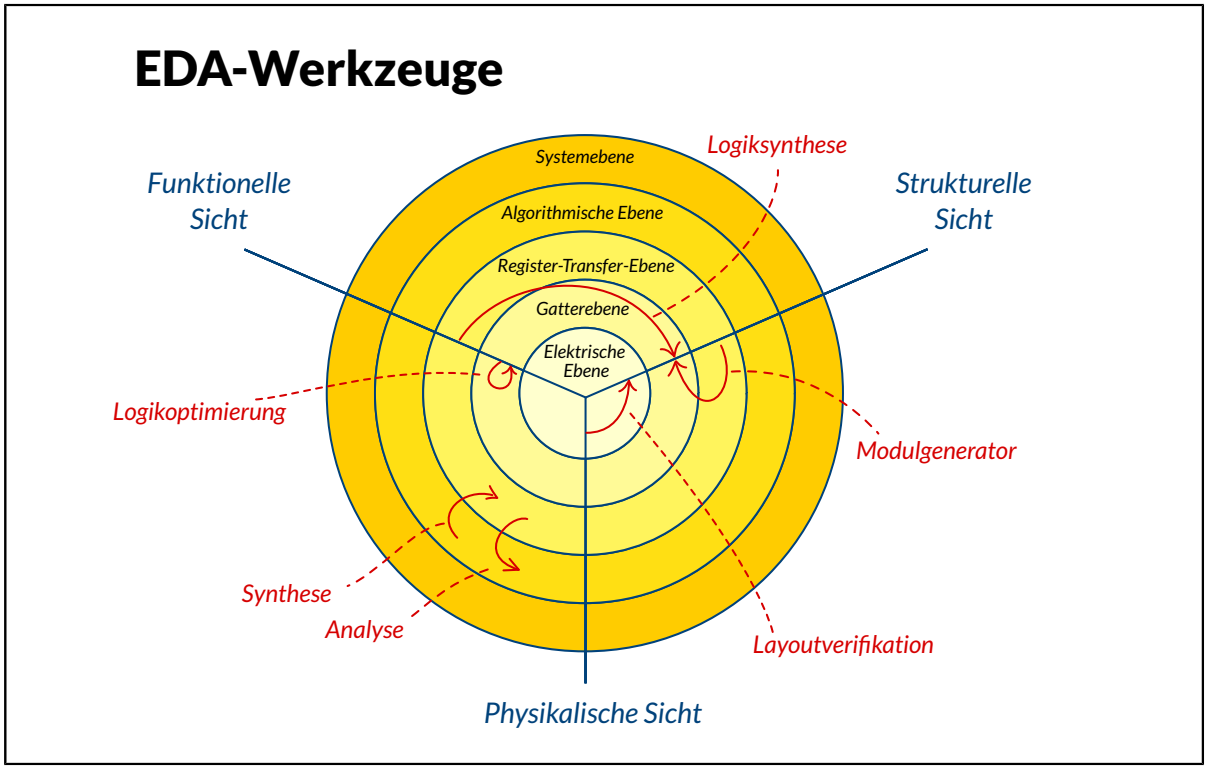
Historisch gesehen hat sich zunächst ein Bottom-Up-Entwurfsstil durchgesetzt, der, ausgehend von geometrischen Entwurfsregeln und elektrischen Transistoreigenschaften, aus Elementen niedriger Ebenen jeweils die Elemente der nächsthöheren Ebene zusammensetzt. Allerdings lässt sich dieses "aus kleineren Bausteinen größere zusammenbauen" nur bis zur Gatter-, maximal Register-Transfer-Ebene durchhalten. Danach ist die Zahl der möglichen Schaltungen einfach zu groß.

Entwurfsprozess: Meet in the Middle



Auf dem heutigen Stand der Technik ist eine "Meet-in-the-Middle"-Entwurfsstil üblich: Einerseits werden bei "Bottom-Up" die entsprechenden Gatter bis zu dieser Ebene entwickelt, andererseits wird der Weg "Top-Down" vom Funktionskonzept bis zur Gatter-Ebene eingeschlagen. Als Mitte lassen sich auch andere Ebenen heranziehen, beispielsweise die RT-Ebene. Nach dem beide Wege durchgeführt sind und eine Gatter(/RT-)netzliste vorliegt, muss jedoch immmer noch ein finaler Place and Route-Schritt durchgeführt werden.

Entwurfsprozess: EDA-Werkzeuge



Bevor wir uns den EDA-Werkzeugen im Einzelnen zuwenden, sollen hier anhand des Y-Diagramms noch einige allgemeine Definitionen vorgenommen werden. Dazu müssen wir uns die Tätigkeiten bei einem einzelnen Entwurfsschritt, also beim Übergang von einem Punkt des Diagramms zu nächsten, genauer ansehen.

Erzeugung (Synthese)

Aus den Dokumenten, die im Verlauf des Entwicklungsprozesses (z.B. Spezifikation, Netzliste, etc.) generiert wurden, muss das jeweils nächste erzeugt werden. Dies kann manuell, automatisch oder rechnergestützt geschehen. Ein Beispiel für automatisches Erzeugen sind Platzierungs- und Verdrahtungsprogramme, die aus Gatterschaltungen in Struktursicht Anordnungen von Zellen in physikalischer Sicht erzeugen.

Rechnergestützt wäre z.B. ein Platzierungs- und Verdrahtungsprogramm, das interaktive Eingriffe bzw. Vorplatzierung erlaubt, oder ein Programm, das nach Analyse einer algorithmischen Beschreibung gewichtete Vorschläge für eine Architektur macht, unter denen der Entwickler zu wählen hat.

Im Allgemeinen werden wir erzeugende EDA-Werkzeuge, bei denen sich gleichzeitig die Entwurfsebene und die Sicht (oder auch nur die Sicht) ändern als Synthesewerkzeuge und solche, bei denen sich nur die Entwurfsebene ändert, die Sicht jedoch gleich bleibt, als Generatoren bezeichnen. Ein Werkzeug, das aus booleschen Gleichungen eine Gatterschaltung erzeugt, ist demnach ein Logiksynthesewerkzeug, ein anderes Werkzeug, das aus einer komplexen Struktur, z.B. einem Multiplizierer eine Realisierung auf Gatterebene ableitet, ist ein (Modul-)Generator. Leider erfolgt die Benutzung dieser Begriffe nicht überall einheitlich. Insbesondere findet man bei Werkzeugen zur Optimierung von Gatterschaltungen oder bei der Generierung von Gatterschaltungen aus struktureller RTL(Register Transfer Level)-Sicht häufig - wenn auch fälschlicherweise - den Begriff "Logiksynthese".

Prüfen (Analyse)

Nur wenn der Entwurfsschritt automatisch gemacht wurde und das benutzte Hilfsmittel die Korrektheit des Ergebnisses garantiert, kann auf eine Überprüfung des Ergebnisses verzichtet werden. In allen anderen Fällen ist der Vorgang fehleranfällig, so dass eine Überprüfung zwingend notwendig ist.

Solche Prüfprogramme sind z.B. Design-Rule-Checker, die geometrische Regeln für Maskenlayouts überprüfen, Syntax-Checker für Funktionsbeschreibungen in höheren Beschreibungssprachen und vor allem Simulatoren für die verschiedenen Abstraktionsebenen. Andere Beispiele sind Netzlisten-Extraktoren zusammen mit Netzlisten-Vergleichern, die den Schritt von der Netzliste zum Maskenlayout (Layoutverifikation) überprüfen. Aber auch Programme, die Ergebnisse von Simulationen auf verschiedenen Abstraktionsebenen (intelligent) vergleichen, gehören dazu, ebenso wie die formale Verifikation. Häufig wird zwischen Validierung und Verifikation unterschieden. Unter Validierung versteht man das Einholen hinreichender Belege dafür, dass die Entwurfsziele erreicht wurden. Verifikation ist dagegen der Vergleich mit einem validierten (oder verifizierten) Standard, um die Einhaltung von Entwurfsregeln jeglicher Art zu beweisen.

Optimieren

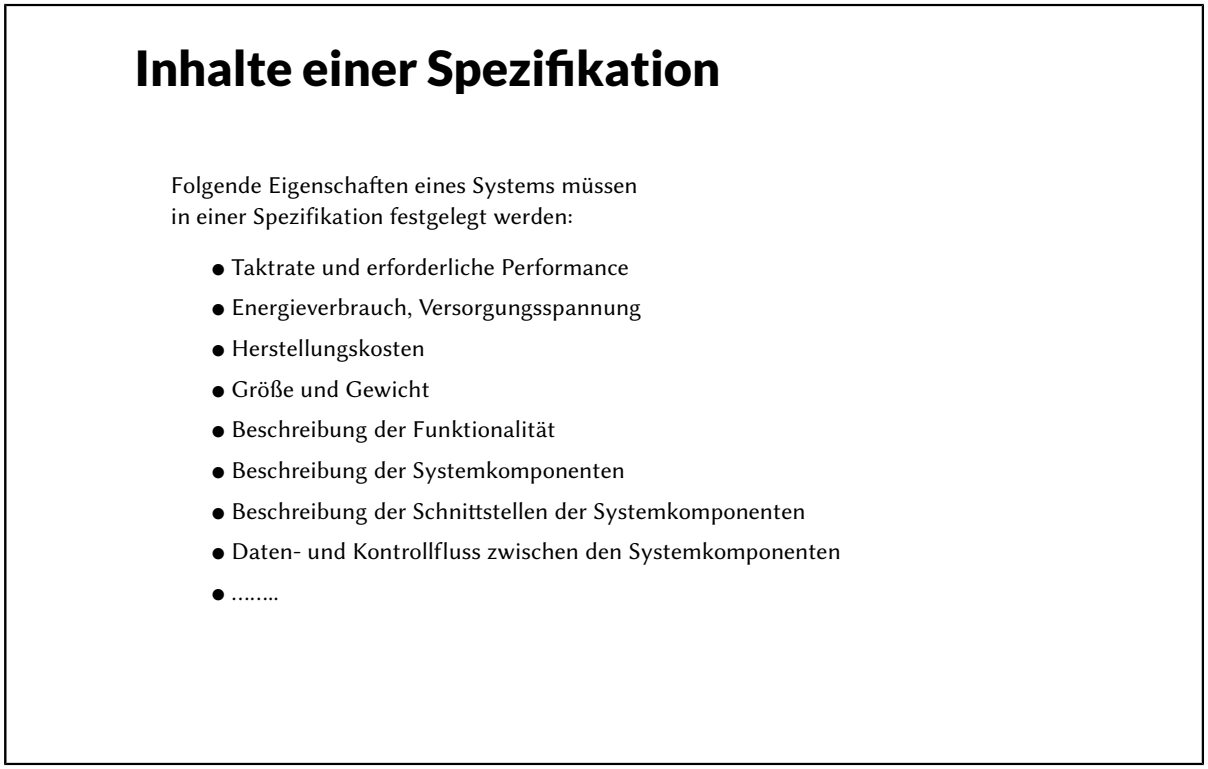
Genügt der erreichte Punkt im Y-Diagramm nicht allen Anforderungen, wird man durch Optimierung versuchen, diesen Mangel zu beseitigen. Beispiele sind Kompaktierer, die die benötigte Fläche im Maskenlayout verringern sollen, Logik-Optimierer, welche beispielsweise die Anzahl der benötigten Gatter verringern sollen, aber auch Architektur-Modifikationen, welche die Performance erhöhen sollen.

Electronic Design Automation (EDA)

Spezifikation

Inhalte einer Spezifikation
Beispielspezifikation Ampelsteuerung
Formale Beschreibung
Blockdiagramme
... für die Ampel
Zustandsübergangs-diagramme
... für die Ampel
Task-Flow-Graphen
... für die Ampel
System Level Design Language
... Vorteile
... Konstrukte

Spezifikation: Inhalte einer Spezifikation



Ausgangspunkt jedes Entwurfsprozesses ist die Spezifikation des zu entwerfenden Systems. Diese soll dazu dienen, alle zu Beginn des Entwurfsprozesses gewünschten Eigenschaften und Randbedingungen formalisiert zu dokumentieren. Insbesondere bei der Arbeit in größeren Teams ist eine solche formale Spezifikation wichtig, damit alle an einem Projekt Arbeitenden einen Überblick über das Gesamtsystem haben. Gleichzeitig dient die (formale) Spezifikation als Referenz bei der Verfeinerung des Systems im Laufe des Entwurfsprozesses. Idealerweise sollte eine Spezifikation vollständig und widerspruchsfrei sein.

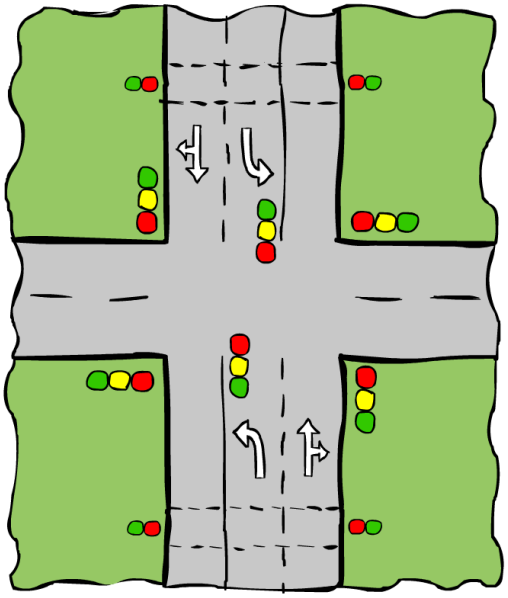
Eine Spezifikation wird auf einer hohen Abstraktionsebene erstellt. Sie kann eine Vielzahl von unterschiedlichen Informationen enthalten. Je nach System – elektrisch, elektro-mechanisch, digital, analog, mixed-signal – wird die Spezifikation nicht alle der oben genannten Daten bzw. zusätzliche Informationen beinhalten.

Spezifikation: Beispielspezifikation Ampelsteuerung

Beispielspezifikation Ampelsteuerung

Steuerung einer Hauptstraßen/Nebenstraßen-Ampel mit Fußgänger-Ampel für die Hauptstraße

- Akustischer Signalgeber für die Fußgängerampel
- Linksabbiegersensor
- Versorgungsspannung 220V
- Treiberleistung je Kanal 1.000 W
- Gesamtleistungsaufnahme < 10.000 W
- Wetterfestes Gehäuse
- Größe maximal 10x10x5 cm
- Taktrate 10 MHz

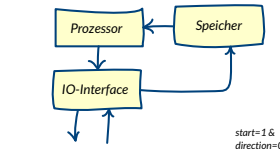


Als einfaches Beispiel sei hier eine Ampelsteuerung spezifiziert, also ein System, das im Wesentlichen lediglich ein Steuerwerk darstellt, das einen Kontrollfluss realisiert. Es beschreibt die Steuerung einer Hauptstraßen/Nebenstraßen-Ampel mit zusätzlicher Fußgänger-Ampel für die Hauptstraße und soll die in der Abbildung dargestellten Eigenschaften aufweisen.

Neben diesen globalen Eigenschaften werden der detaillierte Aufbau der Ampelsteuerung und die Funktionalität mit verschiedenen formalen Methoden beschrieben.

Spezifikation: Formale Beschreibung

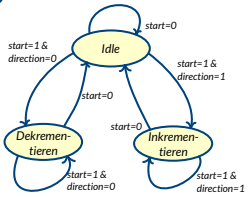
Formale Spezifikationsbeschreibung



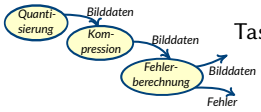
Blockdiagramme

Beschreiben den strukturellen Aufbau eines Systems

Zustandsüber-gangsdiagramme



Dienen zur Beschreibung der Funktionalität, insbesondere des Kontrollflusses.



Task-Flow-Graphen

Beschreiben den Datenfluss des Systems.

System-Level-Design-Language

```
#include "systemc.h"
SC_MODULE(nand2)
// declare nand2
sc_module
{
  sc_in<bool>
...
```

Sind spezielle Sprachen zur Systembeschreibung.

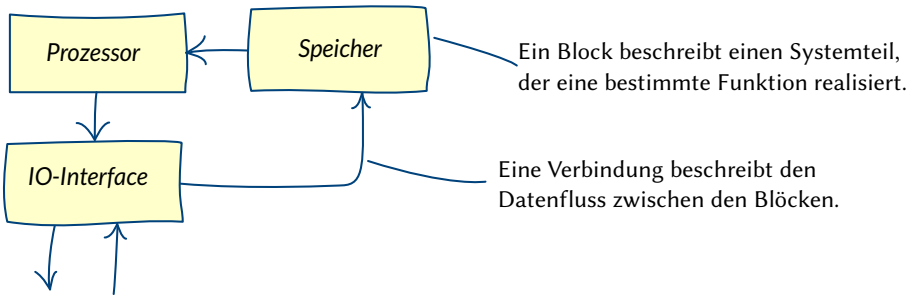
In den meisten Fällen wird die Spezifikation in natürlicher Sprache abgefasst, da bislang keine einheitliche Sprache zur Beschreibung von Spezifikationen existiert. Neben der natürlichen Sprache werden für Teile der Spezifikation auch Blockdiagramme, Zustandsübergangsdiagramme oder Task-Flow-Graphen verwendet. Während Blockdiagramme der Beschreibung der Struktur eines Systems dienen, beschreiben Zustandsübergangsdiagramme das Verhalten des Systems. Mit Task-Flow-Graphen wird der Datenfluss eines Systems spezifiziert.

Grundsätzlich kann die Spezifikation auch mit Hilfe einer formalen Sprache erfolgen. Dazu werden häufig Erweiterungen regulärer Programmierspachen (z. B. C) oder aber auch spezielle Systembeschreibungssprachen benutzt.

Spezifikation: Blockdiagramme

Blockdiagramme

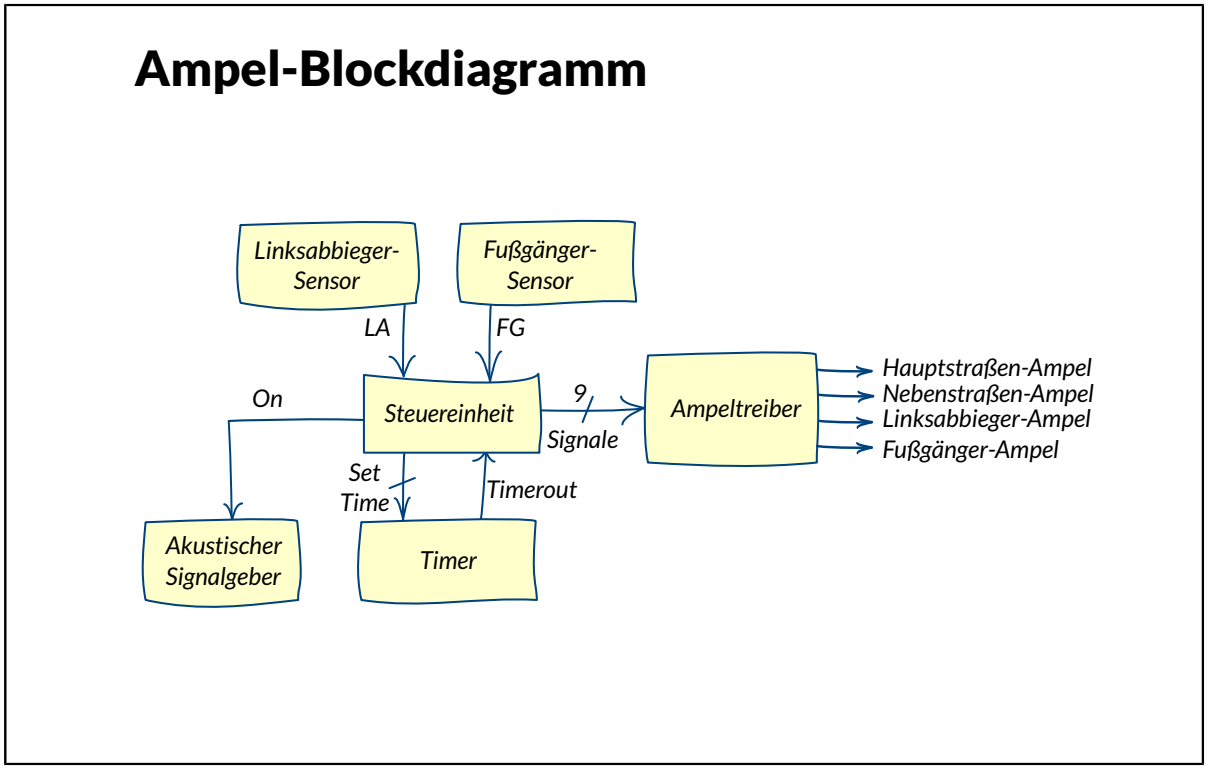
- Beschreiben die grundlegende Systemstruktur.
- Bestehen aus Blöcken und Verbindungen.



- Signalbreiten und Signalnamen können spezifiziert werden.
- Keine Beschreibung von Pegeln oder Protokollen.

Blockdiagramme werden in vielen Spezifikationen zur Beschreibung der grundlegenden Struktur eines Systems verwendet. Blockdiagramme bestehen aus Blöcken und Verbindungen zwischen den Blöcken, wobei jeder Block einer auszuführenden Aufgabe des Gesamtsystems entspricht. Die Verbindungen zwischen den Blöcken entsprechen dem Datenfluss zwischen den Blöcken und können bezüglich der Signalbreiten in einem Blockdiagramm spezifiziert werden. Ein Blockdiagramm basiert auf der Annahme, dass die Aufgaben des Gesamtsystems auf die Blöcke verteilt worden sind. Eine weitergehende Festlegung des Datenflusses zwischen den Modulen wie beispielsweise das Protokoll oder das Datenformat ist in einem Blockdiagramm nicht vorgesehen.

Spezifikation: ... für die Ampel



Im Bild ist das Blockschaltbild der Ampelsteuerung dargestellt.

Die Ampelsteuerung besteht aus zwei Sensoren, die registrieren, ob Linksabbieger an der Ampel stehen bzw. ob eine Querungsanforderung durch Fußgänger besteht.

Die Steuereinheit berechnet aus den Sensorsignalen sowie einem Timer den nächsten Ampelzustand.

Der Timer dient der Steuerung der Intervalllängen, in denen eine Ampel auf grün geschaltet ist. Dabei kann das Zeitintervall, nach dem das Timermodul den Timeout auslöst, von außen gesetzt werden. Dazu wird ein 4-bit breites Signal benutzt. Das Signal Timeout signalisiert den Ablauf des Zeitintervalls.

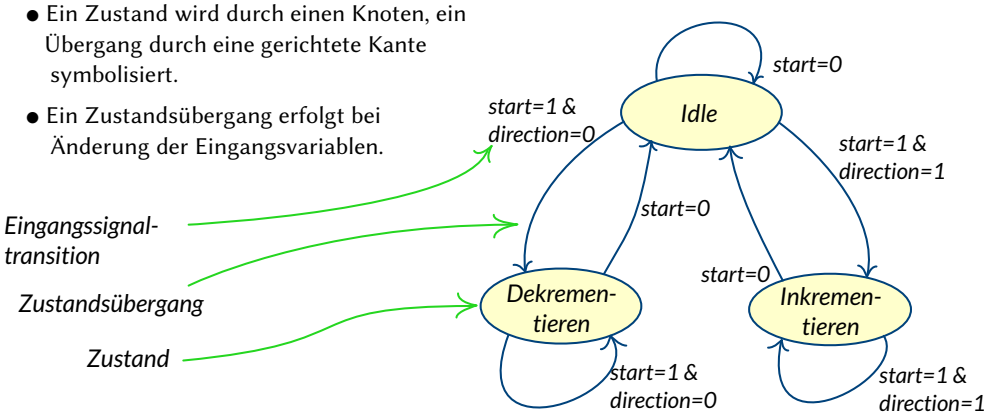
Die Ampeltreiber enthalten die Treiber für die Lampen der Ampel.

Der akustische Signalgeber signalisiert durch ein Tonsignal die Grünphase der Fußgängerampel.

Spezifikation: Zustandsübergangs-diagramme

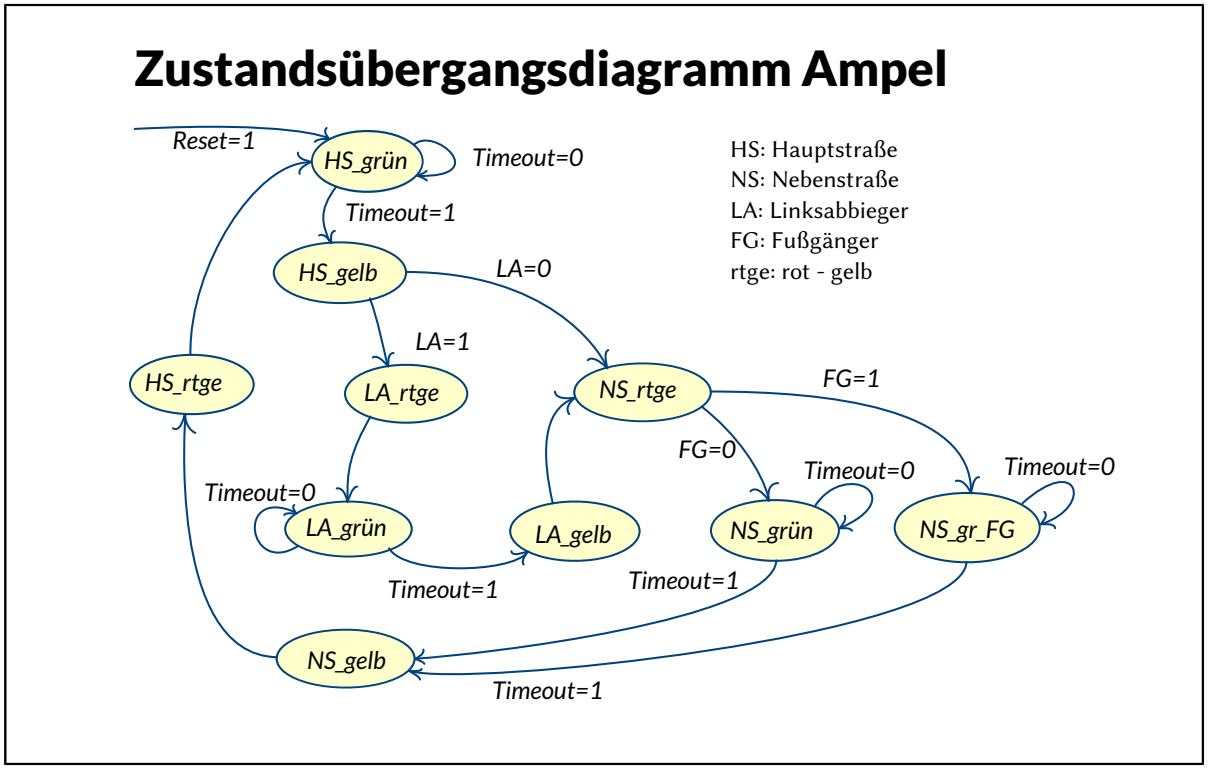
Zustandsübergangsdiagramme

- Beschreiben die Funktion eines Systems in Form eines gerichteten Graphen.
- Bestehen aus Zuständen und Übergängen.
- Ein Zustand wird durch einen Knoten, ein Übergang durch eine gerichtete Kante symbolisiert.
- Ein Zustandsübergang erfolgt bei Änderung der Eingangsvariablen.



Zustandsübergangsdiagramme beschreiben Zustände und Übergänge zwischen den Zuständen. Ein Systemzustand entspricht im Regelfall einem Satz von inneren Zuständen und/oder Ausgangswerten des Systems. Ein Zustandsübergang erfolgt jeweils bei der Änderung der Eingangsvariablen.
Die Zustandsübergänge werden in Zustandsübergangsdiagrammen durch Kanten dargestellt. Die Zustände werden durch Knoten beschrieben. An den Zustandsübergängen werden jeweils die Eingangssignaltransitionen notiert, unter denen der entsprechende Zustandswechsel erfolgt. Zustandsübergangsdiagramme eignen sich besonders gut, um endliche Automaten zu beschreiben. Die Beschreibung von Systemeigenschaften oder Strukturinformationen ist in Zustandsübergangsdiagrammen nicht möglich. Je nach Art des beschriebenen endlichen Automaten erfolgt die Darstellung der Ausgangssignale.

Spezifikation: ... für die Ampel

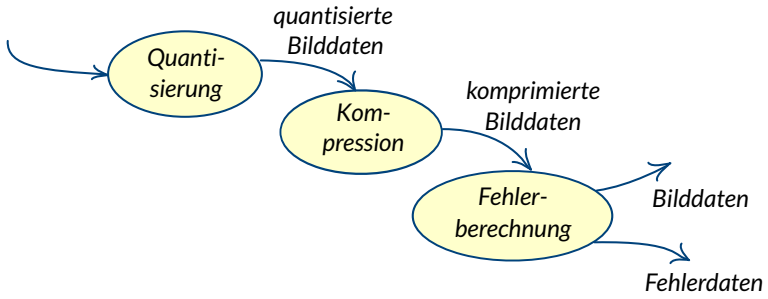


Die Abbildung zeigt das Zustandsübergangsdiagramm für das Modul "Ampelsteuerung" aus dem Blockschaltbild der Ampel. Nach einem Reset startet der endliche Automat in dem Zustand, in dem die Hauptstraße grün hat. Nach Ablauf der Grünzeit, was durch das Signal Timeout signalisiert wird, wechselt die Ampel der Hauptstraße auf gelb. In Abhängigkeit von den Sensoren der Linksabbieger (LA) bzw. Fußgänger (FG) erfolgt der nächste Zustandsübergang. Wurde der Linksabbieger-Sensor ausgelöst, erfolgt jetzt die Linksabbiegerphase. Anschließend folgt die Grünphase der Nebenstraße, wobei die Länge dieser Phase vom Auslösen des Fußgängersensors abhängt. Danach erfolgt wieder der Wechsel in den Initialzustand.

Spezifikation: Task-Flow-Graphen

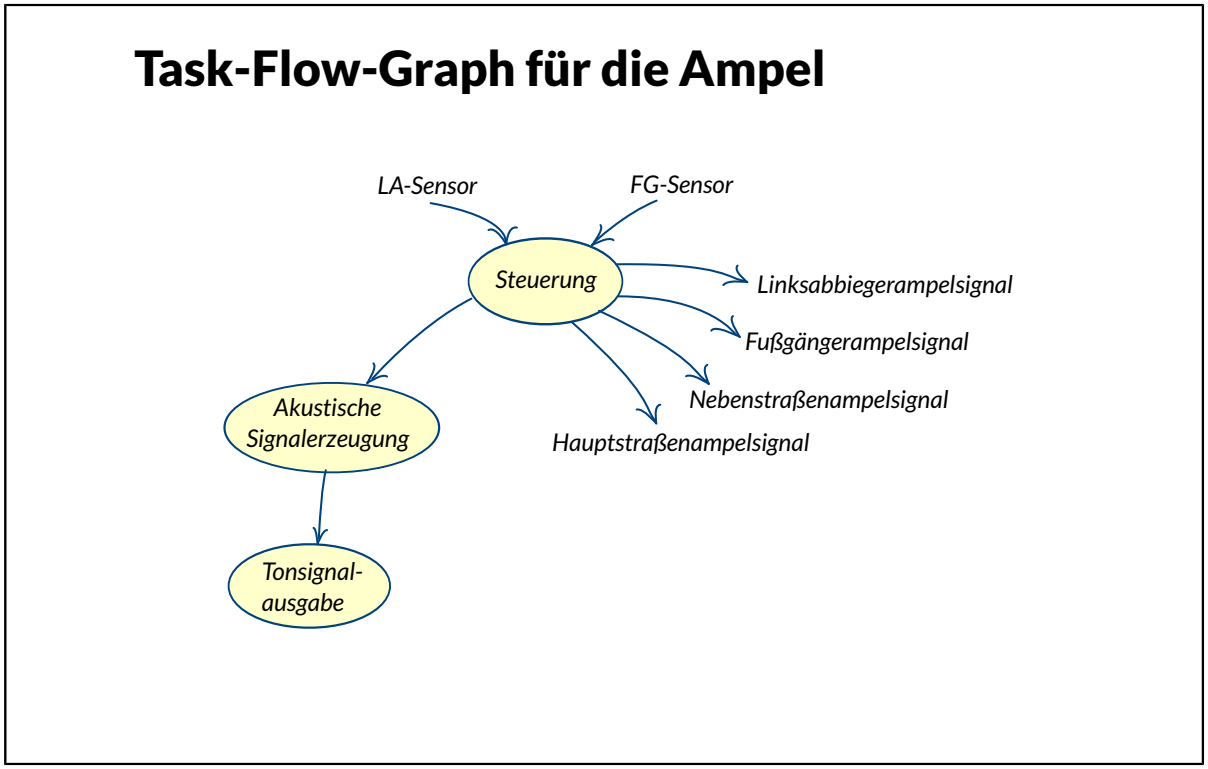
Task-Flow-Graphen

- Bestehen aus Tasks- und Taskgruppen.
- Ggf. erfolgt eine Annotierung des Kontroll- und Datenflusses.
- Ein hierarchischer Aufbau ist möglich.



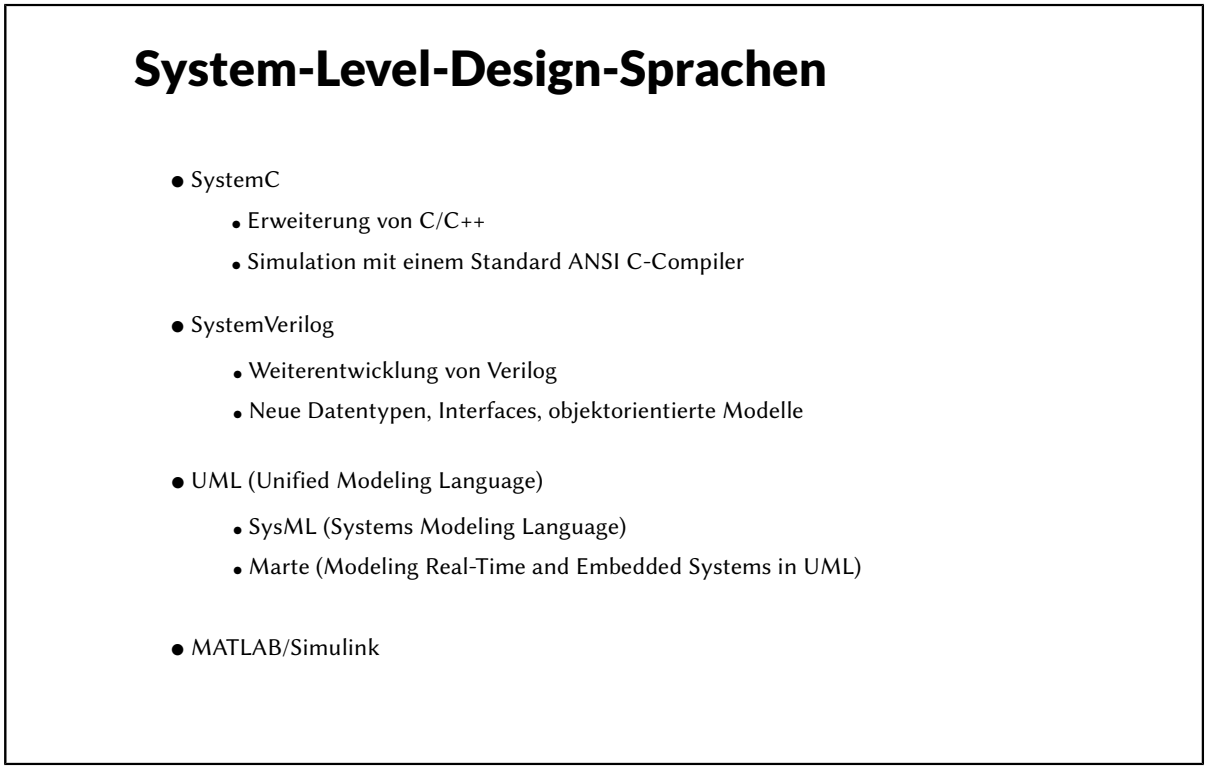
Task-Flow-Graphen werden häufig bei der System-Spezifikation genutzt. Sie stellen den Kontroll- und Datenfluss zwischen den einzelnen Aufgaben (Tasks) eines Systems dar. Jeder Knoten des Graphen stellt dabei eine Aufgabe des Systems dar, die aus vielen einzelnen Schritten bestehen kann. Jede Kante des Graphen repräsentiert einen Datenfluss. Die einzelnen Aufgaben des Systems können zu Aufgabengruppen zusammengefasst werden. Ferner können Task-Flow-Graphen hierarchisch aufgebaut sein, so dass ein Knoten des Graphen in einem weiteren Task-Flow-Graphen detailliert beschrieben werden kann.

Spezifikation: ... für die Ampel



Die Abbildung zeigt den Taskflow-Graphen für die Ampelsteuerung. Der Datenfluss besteht hier in der Signalisierung, dass etwas bestimmtes ausgeführt werden muss.

Spezifikation: System Level Design Language

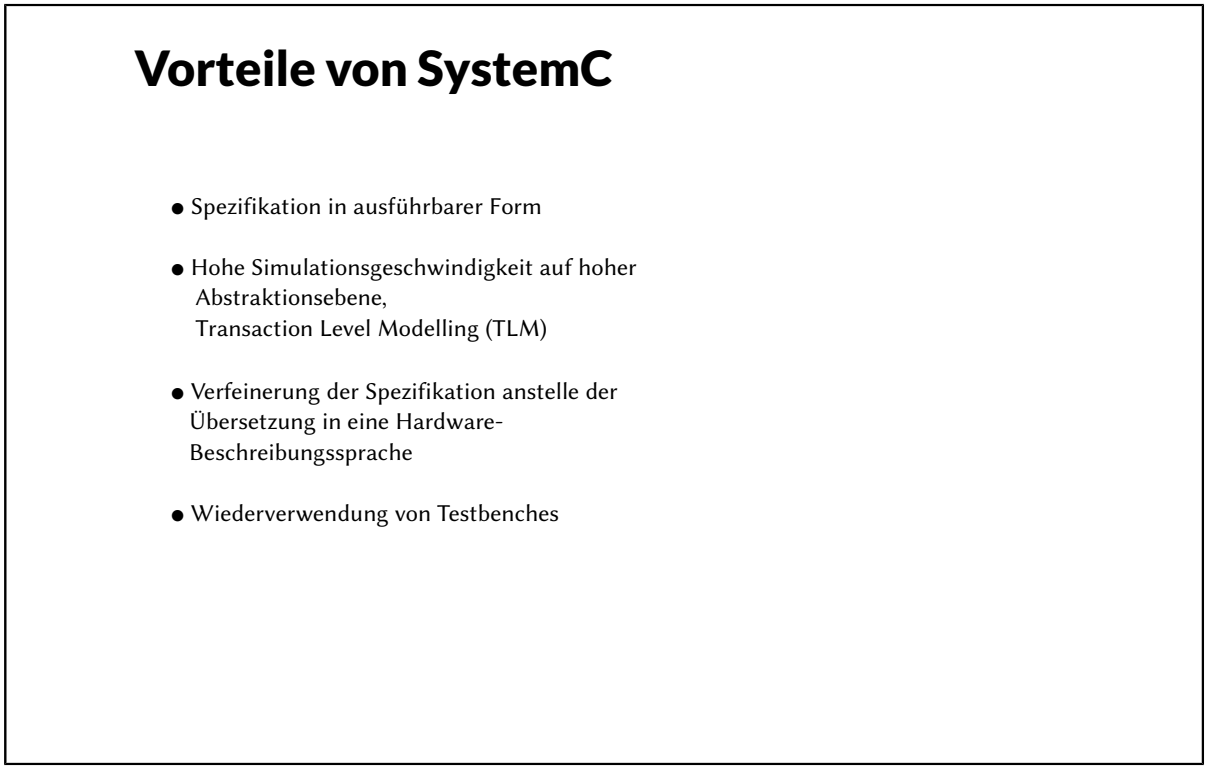


System-Level-Entwurfssprachen sollen eine einheitliche Darstellung für die Beschreibung eines vollständigen Systems – bestehend aus Hard- und Software - liefern. Bisher sind diese Sprachen wenig verbreitet. Zurzeit wird international versucht, einen Standard für eine solche Sprache zu schaffen. Als Quasi-Standard hat sich mittlerweile SystemC etabliert.

SystemC ist eine OpenSource-Initiative, die aus einem Zusammenschluss von Halbleiter-, IP- und EDA-Firmen besteht. SystemC ist frei verfügbar. Zur Simulation von SystemC-Code wird lediglich ein ANSI C++ Compiler benötigt.

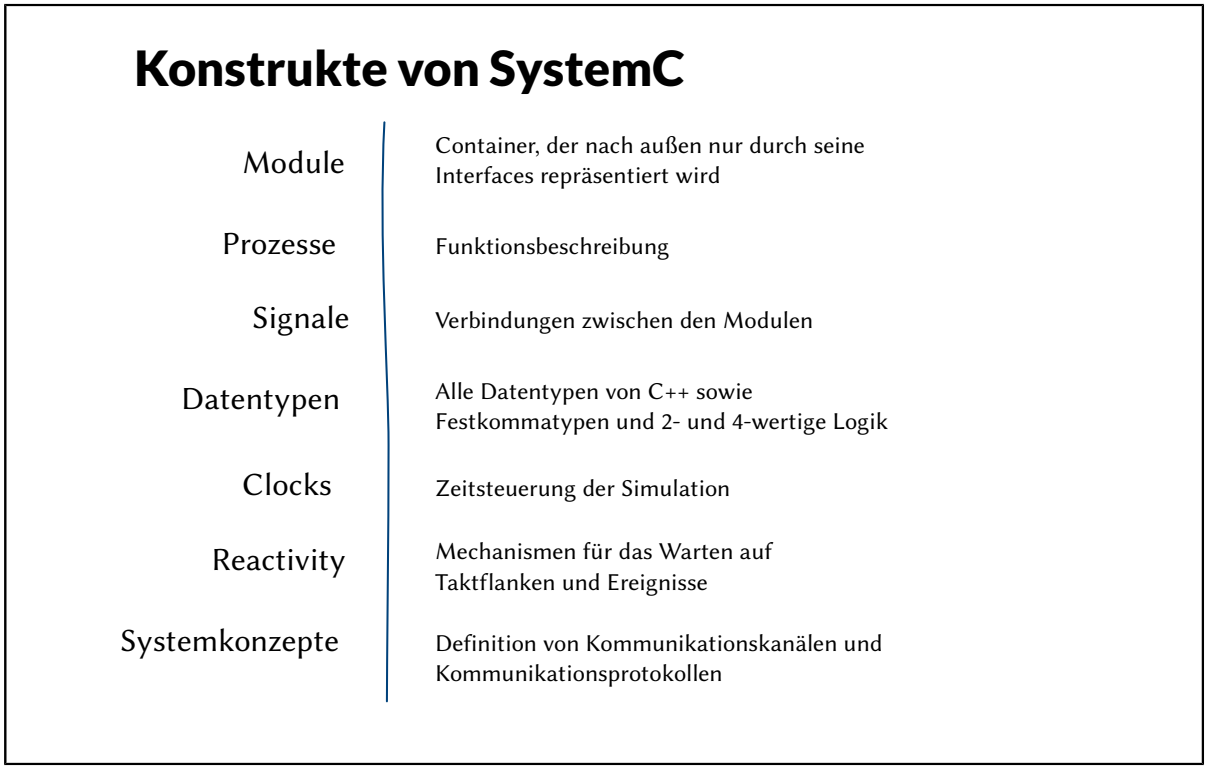
Neben der Entwicklung einer speziellen System-Level-Entwurfssprache wurde auch die Hardware-Beschreibungssprache Verilog um Konstrukte auf höheren Ebenen ergänzt.

Spezifikation: ... Vorteile



Der Einsatz von SystemC bietet verschiedene Vorteile gegenüber dem konventionellen HDL-Designflow. Wichtigster Vorteil ist die Ausführbarkeit der Spezifikation. Die auf hoher Abstraktionsebene beschriebenen Eigenschaften des Systems können bereits simuliert werden. Aufgrund des geringen Detailliertheitsgrads der Spezifikation erfolgt die Simulation auf dieser Ebene sehr schnell. Anschließend ist keine manuelle Umsetzung der Spezifikation in eine Hardware-Beschreibungssprache erforderlich, der C-Code wird lediglich verfeinert.

Spezifikation: ... Konstrukte



SystemC stellt verschiedene Konstrukte zur Systembeschreibung zur Verfügung. Die wichtigsten sind:

- Module: Diese definieren Container, die nach außen nur durch ihre Interfaces repräsentiert werden. Dadurch wird die Implementierung versteckt, was eine Möglichkeit zur Einbindung von bereits vorhandenen, sogenannten Intellectual-Property (IP)-Blöcken schafft.
- Prozesse: In Prozessen wird die eigentliche Funktionalität beschrieben. Dabei unterscheidet man verschiedene Typen von Prozessen. Method-Prozesse verhalten sich wie Funktionen. Thread-Prozesse werden dagegen nur einmal gestartet und werden immer wieder durchlaufen. CThread-Prozesse sind synchrone Thread-Prozesse.
- Signale: Diese definieren Verbindungen zwischen den Modulen und werden wie in anderen Hardware-Beschreibungssprachen verwendet, d.h. die Zuweisung von Signalwerten kann verzögert erfolgen.
- Datentypen: In SystemC stehen alle Datentypen von C++ sowie Festkommatypen und 2- und 4-wertige Logik zur Verfügung.
- Clocks dienen der Zeitsteuerung der Simulation. Dabei sind mehrere Clocks mit beliebigen Phasenlagen zueinander möglich.
- Reactivity: Diese definieren Mechanismen für das Warten auf Taktflanken und Ereignisse sowie das Überwachen von bestimmten Signalen.
- Systemkonzepte: Diese ermöglichen die Beschreibung von Kommunikationskanälen und abstrakten Kommunikationsprotokollen (Handshakes).

Electronic Design Automation (EDA)

Entwurfseingabe

Entwurfseingabe

Systemebene

...SystemC

...SystemVerilog

RT-Ebene

...Verilog

...VHDL

Gatterebene

...Gatternetzliste

...Schematic Entry

Elektrische Ebene/ Strukturelle Sicht

...Bauelementenetzliste

...Schematic Entry

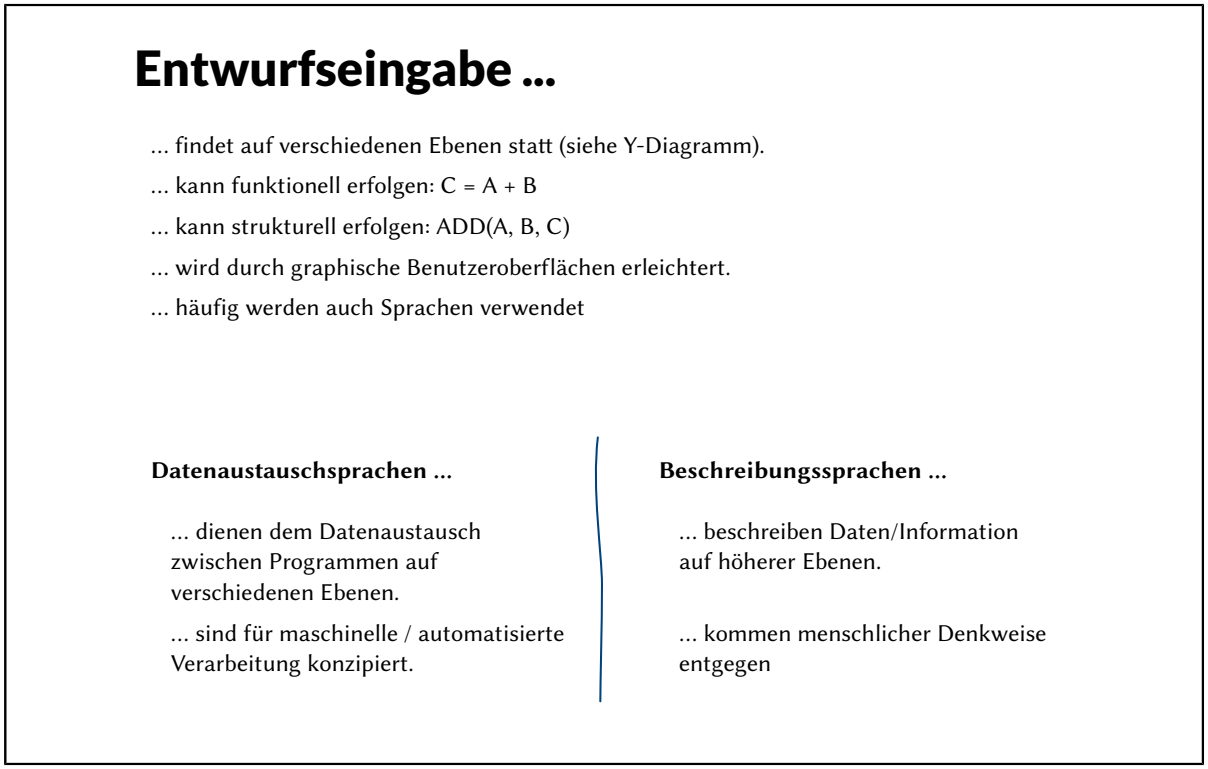
Elektrische Ebene/ Physikalische Sicht

...CIF

...GDSII

...Layout Entry

Entwurfseingabe: Entwurfseingabe



Der Entwickler kann den Entwurf auf verschiedenen Entwurfsebenen und in verschiedenen Sichten eingeben. Viele Entwurfsschritte können durch ein EDA-Tool vorgenommen werden, ein vollständig automatisierter Ablauf (Silicon Compiler) ist nicht realisierbar, deshalb muss der Entwickler in verschiedenen Stadien des Entwurfs Eingriffe von Hand vornehmen. Der Entwurf kann strukturell (ADD(A, B, C) oder funktionell (C=A+B) eingegeben werden. Die Entwurfseingabe basiert immer auf Verwendung einer Eingabesprache. Häufig wird dem Entwickler die Eingabe durch eine graphische Benutzeroberfläche vereinfacht. Man unterscheidet bei den Eingabesprachen:

1) Datenaustauschsprachen

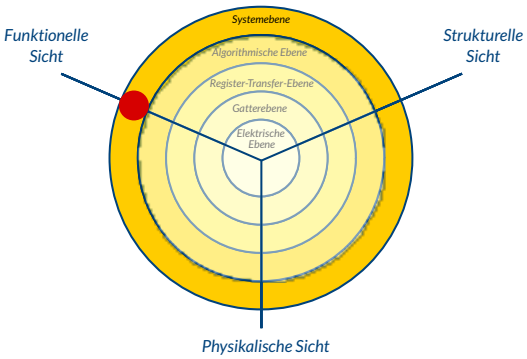
Diese Sprachen sind für den Austausch von Daten gedacht. Dabei handelt es sich hauptsächlich um einen maschinellen Austausch.

2) Beschreibungssprachen

Diese Sprachen dienen zur Beschreibung von Daten/Informationen auf einer Ebene mit einem höheren Abstraktionsgrad. Sie sind für die Benutzung durch den Menschen ausgelegt.

Entwurfseingabe: Systemebene

Systemebene / Funktionelle Sicht



- Oberste Eingabeebene
- Spezifikation als Eingabe
- Hardware- / Software-Funktionalität
- Eingabesprachen: SystemC, SystemVerilog, UML, SysML, Marte
- "Alternativen": C, C++, Mathematiksysteme wie Maple, Mathematica, MATLAB (inkl. SimuLink) u.a.

Die Systemebene ist die oberste Ebene für die Entwurfseingabe. Eine ideale Eingabe wäre die Systemspezifikation (z.B. 8 bit ALU bei 800 MHz lauffähig und mit folgendem Befehlssatz sowie weiteren spezifizierten Randbedingungen). Dies ist aber noch eine Vision. W ichtig ist, dass Sprachen auf dieser Ebene sowohl die Funktionalität von Hardware als auch von Software beschreiben können müssen, da Hardware/Software Co-Design immer wichtiger wird. Derzeitig unterstützte Eingabesprachen sind SystemC und System-Verilog für die Hardware. Alternative Eingaben auf Systemebene sind möglich mit C, C++, VML. Häufig wird für die Systembeschreibung auch mathematische Software wie Matlab, MAPLE oder Mathmatica genutzt.

Entwurfseingabe: ...SystemC

Eingabesprache SystemC

- C++ Klassen-Bibliothek
- Kompatibel und kompilierbar mit Standard-C-Compilern
- HW/SW-Co-Simulation / Co-Entwicklung möglich

```
#include"systemc.h"
SC_MODULE(counter)
{
```

```
// counter.cc
#include " counter.h"
void counter::onetwothree()
{
  if (clear) {
    countval = 0 ;
  } else if (load) {
    countval = din.read() ;
  } else {
    countval ++ ;
  }
  dot  = countval ;
}
```

SystemC ist eine C++ Klassen-Bibliothek. SystemC ist voll kompatibel und kompilierbar mit Standard-C++-Compilern und bietet eine einfache HW/SW-Co-Entwicklungsumgebung.

Entwurfseingabe: ...SystemVerilog

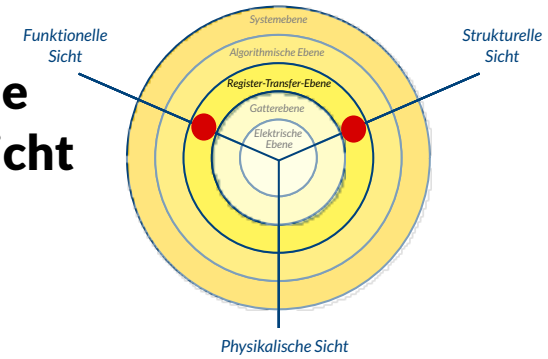
Eingabesprache SystemVerilog

- Erweiterung von Verilog:
  - Objektorientiert (Klassen)
  - Dynamische Fehler
  - Assoziative Fehler
  - String-Operationen
  - Interprozesssynchronisierung
  - Direct Programming Interface (DPI zum Aufruf von C-, C++-Funktionen)
- Kompatibel mit Verilog-Standard
- Annäherung an C++

SystemVerilog ist eine Erweiterung von Verilog-2001 und kompatibel zum Verilog-Standard.
SystemVerilog ist stark angelehnt an C++.

Entwurfseingabe: RT-Ebene

Register-Transfer-Ebene / Funktionelle bzw. Strukturelle Sicht



- Beschreibung durch Register und Operationen
- Eingabe durch Hardwarebeschreibungssprache (VHDL, Verilog)
- Spracherweiterung für analoge Blöcke

Auf der Register-Transfer-Ebene existieren Register und Operationen. Die Eingabe auf dieser Ebene erfolgt mit Hilfe einer Hardwarebeschreibungssprache (Hardware Description Language, HDL). Die am weitsten verbreiteten Sprachen sind Verilog und VHDL. Für die eigentlich digitalen HDLs existieren Erweiterungen, um analoge Blöcke zu beschreiben. HDLs sind in der Lage, sowohl Funktion als auch Struktur zu beschreiben.

Entwurfseingabe: ...Verilog

Eingabesprache Verilog

- 1984 / 1985 entwickelt von Moorby
- 1991 von Cadence übernommen
- 1995 IEEE Standard 1364 (Update 2001)
- Verilog-A zur Beschreibung analoger Schaltungen
- Code-Beispiel: Ampelsteuerung

```
. . .
always @(LA or FG or TimeOut
                    or current_state)
begin
  case (current_state)
  HS_gruen :
  begin
     hsa = gruen;
     lsa = rot;
     nsa = rot;
     fga_as = rt_fg
     if (TimeOut==1)
       next_state = HS_gelb;
     else
       next_state = HS_gruen;
  end
. . .
```

Verilog wurde 1984/1985 von Moorby und anderen entwickelt und 1991 von der EDA-Firma Cadence übernommen. Seit 1995 existiert der IEEE Standard 1364, der im Jahr 2001 aktualisiert wurde. Es existiert eine Spracherweiterung zu Verilog (Verilog-A) zur Beschreibung analoger Schaltungen.

Entwurfseingabe: ...VHDL

Eingabesprache VHDL

- VHDL = VHSIC-HDL =
  " Very High Speed Integrated Circuit - Hardware Descripiton Language "
- Entwicklung initiiert 1980 durch US Department of Defense
- Design-unabhängig und Simulator-unabhängig
- 1987 IEEE-Standard 1076 (Update 1993)
- VHDL-AMS-Erweiterung für Mixed-Signal-Schaltungen

```
entity NANDXOR is
  port (
     A, B  :  in    bit;
     C     :  in    bit;
     D     :  out   bit);
end NANDXOR;

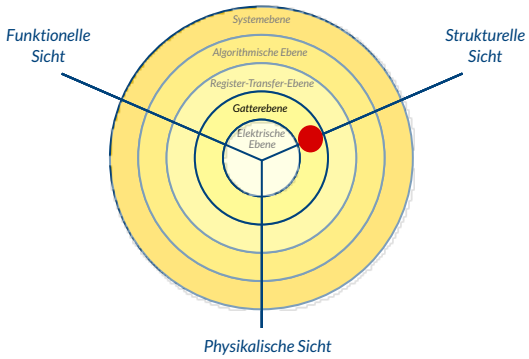
architecture RTL1 of NANDXOR is
begin
  D <= (A nand B) xor C;
end RTL1;

architecture RTL2 of NANDXOR is
begin
  process (A, B, C)
  begin
     if (C = '0') then
        D <= A nand B;
     else
        D <= A and B;
     end if;
  end process;
end RTL2;

achitecture RTL3 of NANDXOR is
signal T : bit;
begin
   T <= A nand B;
   p1 : process(T, C)
   begin
      D <= T xor C;
   end process p1;
end RTL3
```

Entwurfseingabe: Gatterebene

Gatterebene / Strukturelle Sicht



- Prozesstechnologie kann berücksichtigt werden
- Gatterbibliothek der Chip-Hersteller
- Eingangskapazität und Treiberstärke können beachtet werden
- Gatterverzögerungszeiten können als Schätzung zur Verfügung stehen

Die Eingabe auf Gatterebene muss die zu verwendende Prozesstechnologie berücksichtigen. Gatter sind in einer vom Chip-Hersteller zur Verfügung gestellten Bibliothek beschrieben. Nur diese Grundgatter können verwendet werden. Eingangskapazitäten und Treiberstärke müssen beachtet werden. Gatterverzögerungszeiten stehen als Schätzung zur Verfügung.

Entwurfseingabe: ...Gatternetzliste

Gatternetzliste

- Beispiel Verilog-Netzliste:
- Signalübergangszeitem können mit #(T_{fall}, T_{rise}) definiert werden.

```
module 4bit_compare (A, B, equal);

  input [3:0]  A;
  input [3:0]  B;
  output  equal;

  wire w1, w2, w3, w4, w5, w6;

  XNOR  # (0.2n, 0.3n)    G1 (A[0],B[0],w1);
  XNOR  # (0.2n, 0.3n)    G2 (A[1],B[1],w2);
  XNOR  # (0.2n, 0.3n)    G3 (A[2],B[2],w3);
  XNOR  # (0.2n, 0.3n)    G4 (A[3],B[3],w4);

  NAND  # (0.1n, 0.1n)    G5 (w1, w2, w5);
  NAND  # (0.1n, 0.1n)    G6 (w3, w4, w6);

  NOR     # (0.1n, 0.2n)    G7 (w5, w6, equal);

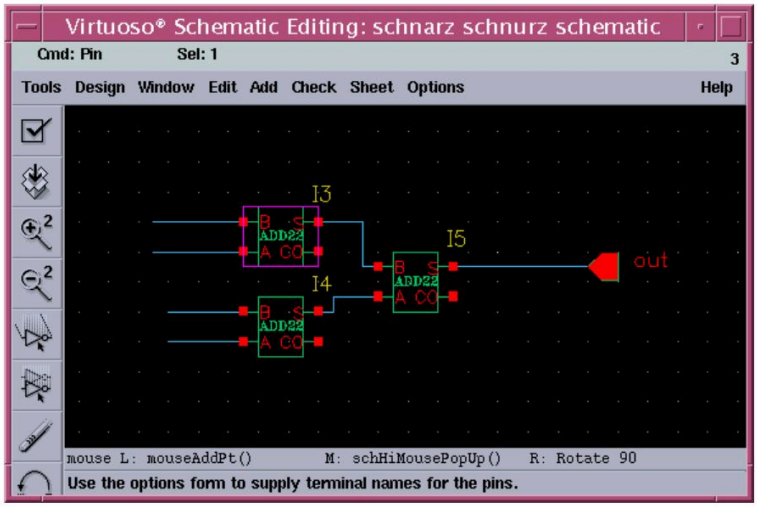
endmodule
```

Gatternetzlisten können beispielsweise mit Verilog beschrieben werden. Dort tretten die Gatter als Bibliotheksaufrufe in Erscheinung. Signalübergangszeiten können mit #(Tfall, Trise) definiert werden.

Entwurfseingabe: ...Schematic Entry

Schematic Entry auf Gatterebene

- Gatter in symbolischer Darstellung aus Gatterbibliothek
- Ideale Verbindungsleitungen

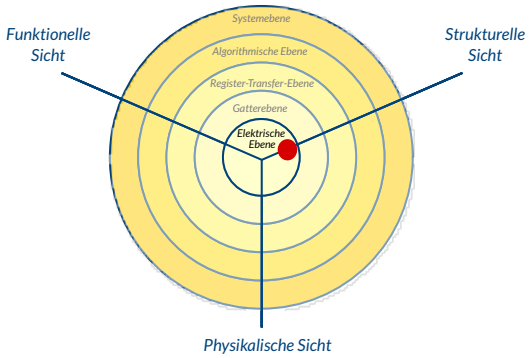


Eine Eingabe auf der Gatterebene kann auch mit Hilfe eines speziellen grafischen Editors (Schematic Entry) erfolgen. Die symbolische Darstellung der Gatter wird der Bibliothek entnommen. Die Verbindungsleitungen werden als ideal leitend angenommen.

Entwurfseingabe: Elektrische Ebene/ Strukturelle Sicht

Elektrische Ebene / Strukturelle Sicht

- Festlegung der Schaltungsstruktur (Topologie)
- Festlegung der Schaltungsdimensionierung
- Alles ist Analog!



Auf der elektrischen Ebene muss die Schaltungsstruktur (Topologie) mit den Parametern der Schaltungselemente (R, C, W, L, ...) definiert werden. Es wird nicht mehr zwischen digitalen und analogen Schaltungen unterschieden. Die eingegebenen Schaltungen bestehen aus Grundelemeneten (Widerstände, Kapazitäten, Induktivitäten, Transistoren, Dioden, Quellen). Die Eingabe kann in Form einer Bauelementenetzliste oder wiederum graphisch durch Schematic Entry erfolgen.

Entwurfseingabe: ...Bauelementenetzliste

Bauelementenetzliste

- Wichtigstes Format ist SPICE
  (Simulation Program with Integrated Circuit Emphasis).

```
V_Vcc       6    0   DC   10
Q_Q1        4    3        5     Q2N2222
R_R1        6    3        100k
R_RC        6    4        2.2k
C_Cout      4    2        1u
C_Cin       1    3        0.1u
R_R2        3    0        47k
R_RL        2    0        47k
R_RE        5    0        1k
V_VIN       1    0   DC   -10  AC  1  sin ( 0  0.01  1000hz )
```

Eine Bauelementenetzliste ist eine Schaltungsbeschreibung auf elektrischer Ebene, die als Text in einem vom Rechner lesbaren Format eingegeben oder rechnerintern erzeugt wird. Es gibt unterschiedliche Formate von Netzlisten, wie z.B. SPICE-Format, EDIF. Am ältesten und am meisten verbreitet ist das Format des Schaltungssimulators SPICE (Simulation Program with Integrated Circuit Emphasis), zum ersten Mal vorgestellt 1972. SPICE war nur auf Großrechnern lauffähig, konnte aber Entwicklungszeiten und -kosten stark reduzieren.

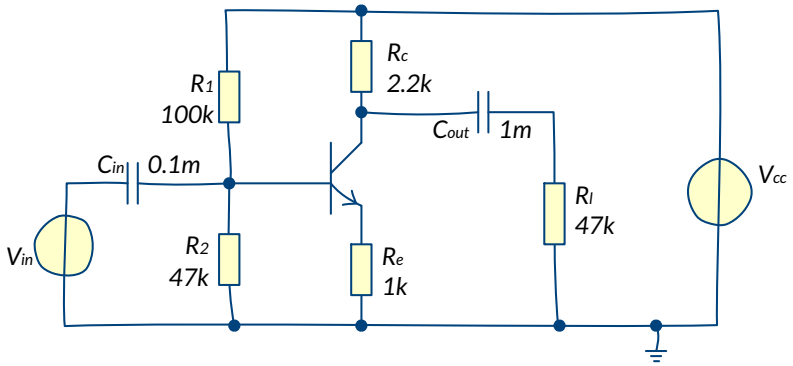
Später wurde das Simulationsprogramm SPICE von der Firma MicroSim in einer PC-Version mit dem Namen PSPICE veröffentlicht. Um PSPICE benutzerfreundlicher zu gestalten wurden ein Schaltplaneditor und ein "Software - Oszilloskop" (PROBE), was die Darstellung der Simulationsergebnisse am Monitor ermöglichte, hinzugefügt. Das Programm wurde weiterhin um Zusätze erweitert, die das Erstellen von Platinen-Layouts ermöglichen. PSPICE ist als Testversion (die Anzahl der Halbleitertypen sowie die Anzahl der Bauelemente in einer simulationsfähigen Schaltung sind eingeschränckt) kostenlos zugänglich.

Ein Nachteil des Programms ist, dass ausschließlich US-Schaltzeichen für analoge und digitale Bauelemente benutzt werden.

Entwurfseingabe: ...Schematic Entry

Schematic Entry auf elektrischer Ebene

- Graphisch unterstützte Eingabe von, aus Grundelementen aufgebauten, Schaltungen
- Bauelementebibliothek erforderlich
- Netzlister zur automtischen Erzeugung von Bauelementenetzlisten

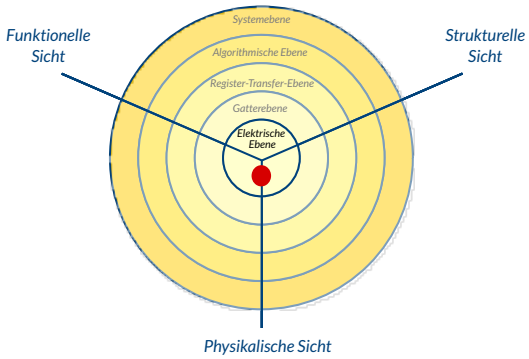


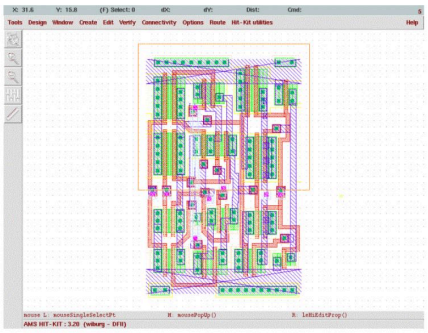
Mit Hilfe eines Schematic-Entry-Werkzeugs können Schaltungen auf elektrischer Ebene auch eingegeben werden. Für die Eingabe wird eine Bauelementebibliothek benötigt. Mit einem Netzlister können aus dem Schaltbild automatisch Bauelementenetzlisten erzeugt werden.

Entwurfseingabe: Elektrische Ebene/ Physikalische Sicht

Elektrische Ebene / Physikalische Sicht

- Layout des Chips besteht aus geometrischen Grundformen (Rechtecke, Linien, Polygone)
- Layout beschreibt Masken für die Chipfertigung
- Formate: CIF, GDSII





Auf der elektrischen Ebene und der physikalischen Sicht besteht die Schaltung aus geometrischen Grundmustern (Rechtecke, Linien, Polygone) mit denen Leiterbahnen und aktive Bauelemente beschrieben werden. Das Layout wird für die Herstellung der Fertigungsmasken benötigt. Auch in dieser Form kann eine Schaltungseingabe erfolgen. Dies ist mit einem graphischen Editor oder mit entsprechenden Beschreibungssprache (-format) wie z.B. CIF (Caltech Intermediate Format) oder GDSII (Graphical Design Station II) möglich.

Entwurfseingabe: ...CIF

Caltech Intermediate Format

- 1980 publiziert von Carver Mead und Lynn Conway in "Introduction to VLSI Systems" (Technical University of California)
- ASCII-Format (lesbarer Klartext)
- Standard für den Layout-Daten-austausch an Hochschulen

```
(nmos transistor)
DS 1 1 1;
L CONT;
B 90 90 0,0;
L NPLUS;
B 290 290 0,0;
L DIFF;
B 150 150 0,0;
L MET1;
B 190 190 0,0;
DF;
DS 2 1 1;
L MET1;
W 110 -1015,310 175,310;
W 110 -1025,-290 155,-290;
L POLY1;
W 80 -1040, -5 190,-5;
L DIFF;
B 680 1120 -480,0;
C1 T-480, 310;
C1 T-480,-290;
DF;
C 2;
```

Das Caltech Intermediate Format wurde 1980 von Carver Mead und Lynn Conway ("Introduction to VLSI Systems") vorgestellt. Es ist lesbarer Klartext im ASCII-Format. CIF ist eine Art Makrosprache, die sogar Kreisstrukturen erlaubt. Dieses Format wird hauptsächlich an Hochschulen verwendet.

Entwurfseingabe: ...GDSII

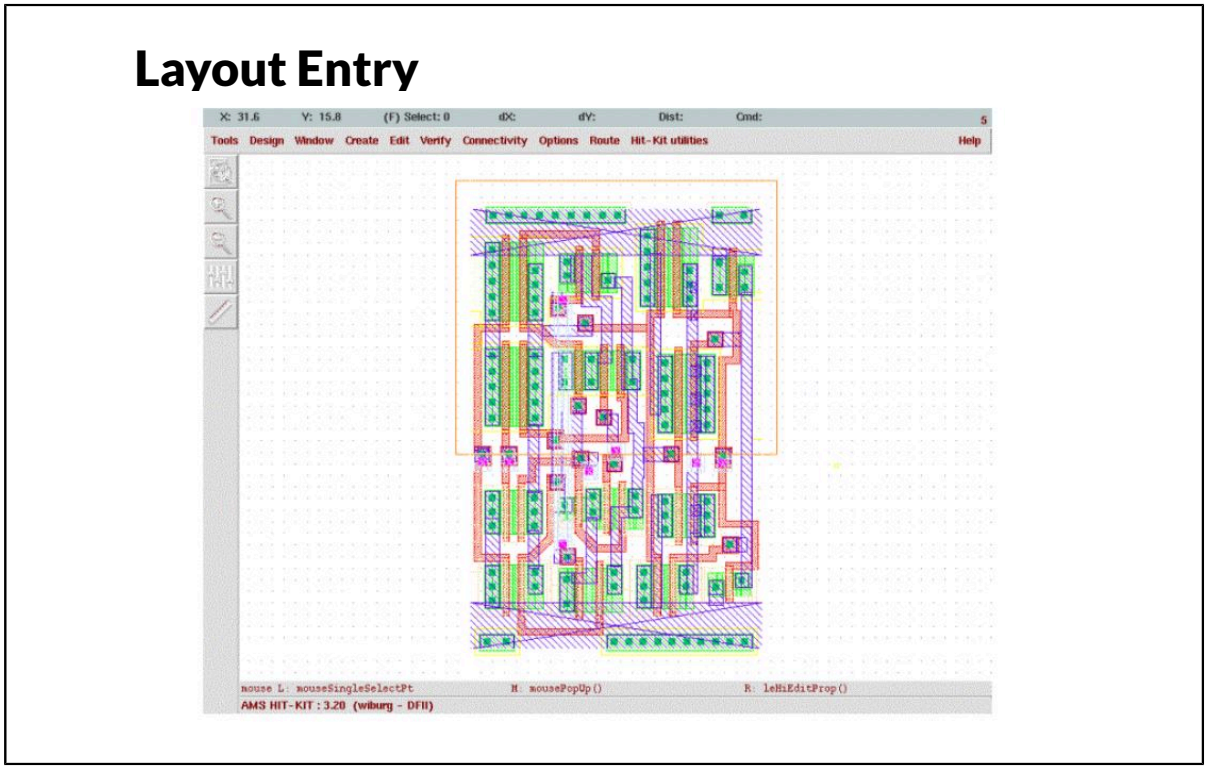
Graphic Design Station II

- In den 1970ern von der Firma Calma Co. eingeführt
- Eigentliches Format: STREAM (Binärformat)
- Standard für den Layout-Daten-austausch in der Industrie

0100011101
0011011
01111110
111100000
00110011111
11000000010
00111100
011101110
1111000
1111111110
0001
0111
11100

Was heute als GDSII-Format bezeichnet wird, hieß ursprünglich STREAM. Es war die Ausgabe der Calma Graphical Design Station II (entwickelt in den 1970ern), daher der Name GDSII. GDSII ist ein Binärformat und in der Industrie eine Art Standard für den Datenaustausch zwischen EDA-Werkzeugen oder auch zwischen Design-Teams.

Entwurfseingabe: ...Layout Entry



Eine grafische Layouteingabe ist mit Ebenen-orientierten Graphikprogrammen möglich. Auf den Ebenen werden Polygone angelegt. Häufige Operationen sind: copy, mirror, rotate, stretch.

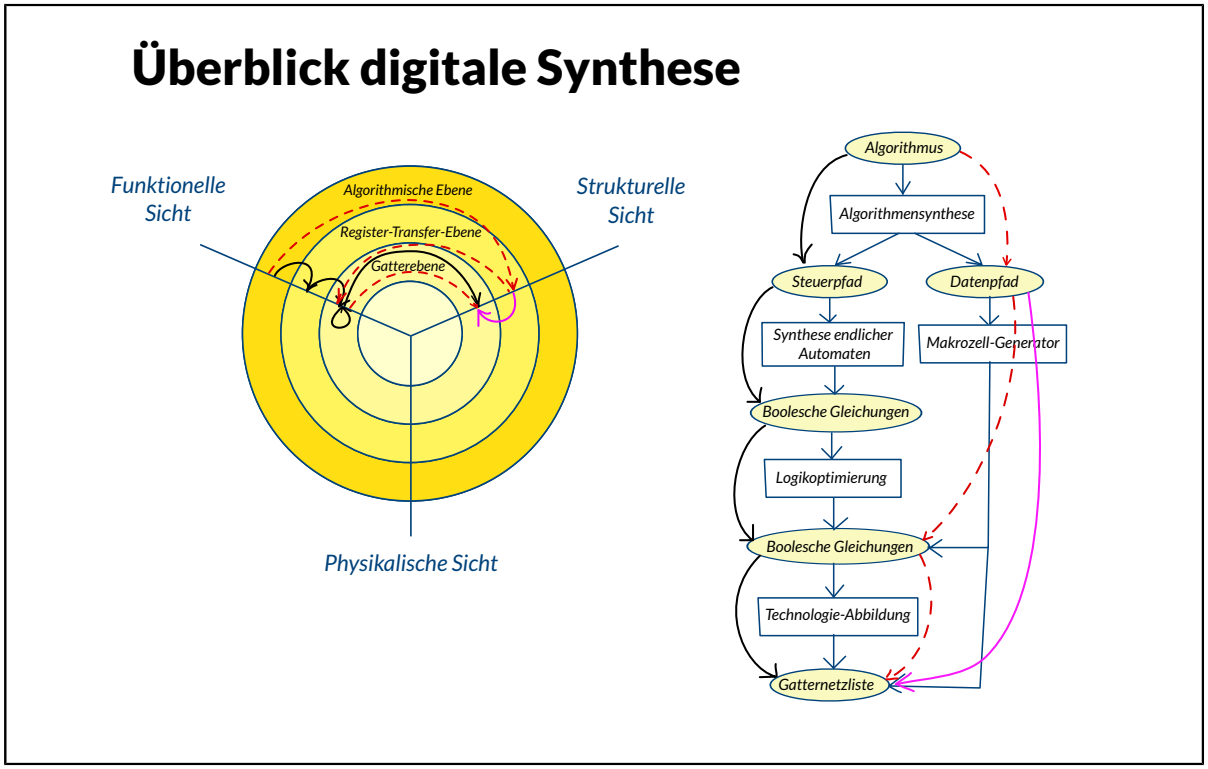
Electronic Design Automation (EDA)

Algorithmensynthese

Überblick digitale Synthese
Algorithmensynthese
Beispiel für einen Algorithmus
Abhängigkeitsgraph
Schleifen
Bedingte Verzweigungen
Schritte der Algorithmensynthese
Takt und kombinatorische Verzögerung
Scheduling – Erstellung eines Taktschemas
Scheduling Variante 1
Scheduling Variante 2
Scheduling Variante 3
Entwurfsraumexploration
Signalflussgraph
Varianten im Entwurfsraum
Pareto-Front
Ressourcenreservierung und Ressourcenzuordnung
Datenpfad mit Multiplexern
Datenpfad mit bidirektionalem Bus
Steuerpfad
Nachoptimierungsmöglichkeiten
Retiming

Date of generation: 8.7.2025

Algorithmensynthese: Überblick digitale Synthese



Aufgabe der digitalen Synthese ist die Übersetzung einer funktionellen Schaltungsbeschreibung in einer Hardware-Beschreibungssprache (HDL, z. B. Verilog, VHDL) in eine Gatternetzliste. Ausgangspunkt der digitalen Synthese im Y-Diagramm ist die algorithmische Ebene in der funktionellen Sicht, am Ende liegt die Schaltung auf der Gatterebene in struktureller Sicht vor.

Die digitale Synthese gliedert sich in vier Schritte, die in der Abbildung dargestellt sind:

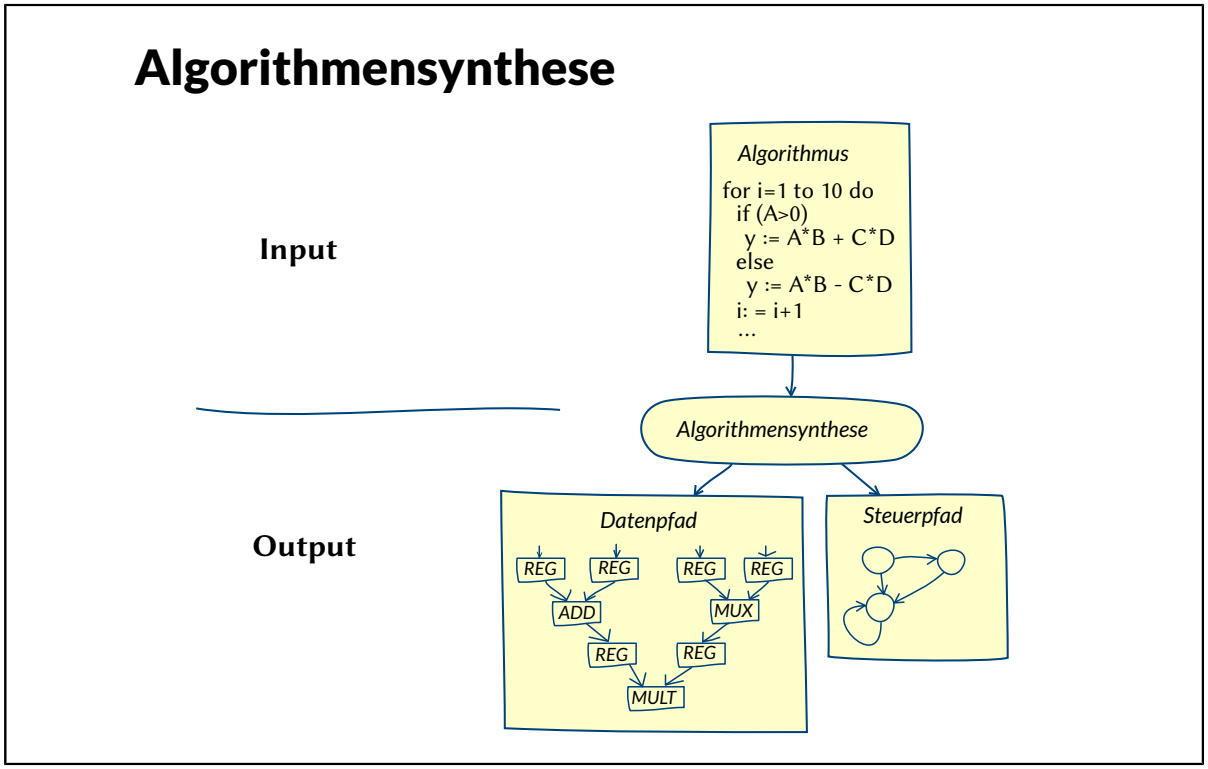
- **Algorithmensynthese**
  Hier wird aus der Verhaltensbeschreibung in einer HDL eine Register-Transfer-Struktur erzeugt. Dabei wird üblicherweise die Schaltung in einen Datenpfad und einen Steuerpfad zerlegt. Der Datenpfad beschreibt die vom Algorithmus auszuführenden arithmetischen Operationen, während der Steuerpfad den Ablauf des Algorithmus kontrolliert und als Zustandsübergangsdiagramm beschrieben werden kann. Für den Datenpfad findet bereits in diesem Schritt eine Optimierung von Kosten bzw. Performance statt.
- **Register-Transfer-Synthese**
  Hier werden der Daten- und der Steuerpfad getrennt voneinander synthetisiert. Der Steuerpfad, der als endlicher Automat (Finite State Machine, FSM) dargestellt werden kann, wird durch boolesche Gleichungen (Gatter) und Speicherelemente realisiert. Der Datenpfad wird ebenfalls mit optimierten booleschen Gleichungen (Gatter) und Speicherelementen (Register) realisiert. Für stark spezialisierte Strukturen wie Speicherfelder oder arithmetische Elemente wie Addierer oder Multiplizierer können durch Makrozellgeneratoren technologieabhängige Beschreibungen wie eine Gatternetzliste oder ein Layout erzeugt werden.
- **Logikoptimierung**
  Die im Rahmen der bisher ausgeführten Schritte entstandenen booleschen Gleichungen weisen noch ein erhebliches Optimierungspotential auf. Entsprechend der Kriterien Performance und Kosten können die logische Schaltungstiefe und die Anzahl der benötigten Gatter optimiert werden.
- **Technologie-Abbildung auf Logikebene**
  Das nach der Optimierung erhaltene System boolescher Gleichungen ist noch unabhängig von der Technologie, in der die Schaltung realisiert werden soll. In diesem Schritt wird es mit Hilfe einer

technologieabhängigen Bibliothek von Grundgattern in eine Gatternetzliste umgesetzt. Die Gatter sind hinsichtlich Fläche und Zeitverhalten charakterisiert.

Diese Schritte enthalten erneut Optimierungen, bei denen Kompromisse zwischen Performance und Kosten eingegangen werden. Ein weiteres Optimierungsziel kann der Leistungsverbrauch der Schaltung sein.

Aufgrund des hohen Abstraktionsgrads der Schaltungsbeschreibung auf algorithmischer und Register-Transfer-Ebene sind die Optimierungskriterien nur durch stark vereinfachende Modelle beschreibbar. Die Genauigkeit der Modelle nimmt auf den unteren Abstraktionsebenen zu. Daher kann oft erst nach der Technologie-Abbildung festgestellt werden, ob bestimmte Ziele von vorausgehenden Synthese-Schritten erreicht wurden. Dann ist ein erneutes Durchführen dieser Schritte erforderlich.

Algorithmensynthese: Algorithmensynthese



Die Algorithmensynthese verwendet als Eingabedaten eine generische Beschreibung des zu synthetisierenden Algorithmus. Für die Beschreibung wird eine sequenzielle Abfolge von Operationen in einer Hochsprache verwendet. Hardwarebeschreibungssprachen bieten die Möglichkeit, das sequenzielle Ablaufschema eines Algorithmus zu beschreiben.

Das Ziel der Algorithmensynthese ist die Umsetzung der gegebenen Verhaltensbeschreibung in eine äquivalente strukturelle Beschreibung auf RT-Ebene. Die zu erreichende Struktur besteht dabei aus einem so genannten Datenpfad und einem zugehörigen Steuerpfad. Der Datenpfad enthält alle vom Algorithmus benötigten arithmetischen Operatoren sowie Multiplexer und Register zur Datenzuführung und Zwischenspeicherung. Die Operatoren werden als arithmetische Makrozellen instanziiert. Mit Hilfe des Steuerpfads erfolgt die Ansteuerung des Datenpfads, so dass die Operationen in der durch die Verhaltensbeschreibung vorgegebenen Reihenfolge ausgeführt werden. Der Steuerpfad wird als endlicher Automat realisiert, dessen Zustandsübergangsdiagramm an den nächsten Syntheseschritt zur weiteren Verarbeitung übergeben wird.

Algorithmensynthese: Beispiel für einen Algorithmus

Beispiel für einen Algorithmus

Ziel: schrittweise Lösung der Differentialgleichung

$$\frac{d^2u}{dt^2} + 2\,\frac{du}{dt}\,t + 10u = 0$$

DGL

VHDL

```
entity DGL is
port( t_in, u_in, s_in, dt_in, T_in : IN REAL;
      u_out : OUT REAL; start : in std_logic);
end DGL;

architecture behv of DGL is
BEGIN
dgl_1: process( start)
VARIABLE t, u, s, dx, T, t1, s1, u1 : REAL;
  BEGIN
  t:= t_in; u:=u_in; s:=s_in; dt:=dt_in; T:=T_in;
  LOOP
    t1 := t+dt;
    u1 := u+(s*dt);
    s1 := s -(2*t*(s*dt))-(10*u*dt);
    t:=t1;  u:=u1;  s:= s1;
  EXIT WHEN t1>T;
 END LOOP;
 u_out <= u;
 end dlg_1;
end behv;
```

Beim Betrieb eines Vorgängermodells der Ampelsteuerung hat sich herausgestellt, dass aufgrund parasitäter Kapazitäten und Induktivitäten in den Signalleitungen Schwingungen auftreten, die durch eine nichtlineare Differentialgleichung zweiter Ordnung beschrieben werden können:

$$\frac{d^2u}{dt^2} + 2\,\frac{du}{dt}\,t + 10u = 0$$.

Die Spannung u soll deshalb während des Betriebes vorausgesagt werden können. Dazu soll ein entsprechender Algorithmus, der die Differentialgleichung in einem Zeitintervall T schrittweise numerisch löst, aus Performance-Gründen in einem spezifischen IC hardwaremäßig implementiert werden. Der dazu gehörige Algorithmus ist im Bild dargestellt. Nach Eingabe der Anfangswerte für t, u, s (=du/dt), der Schrittweite dt sowie des Intervalls T berechnet dieser den Spannungswert u am Ende des Intervalls gemäß folgender Rechnung:

t = t + dt

s = (du / dt)

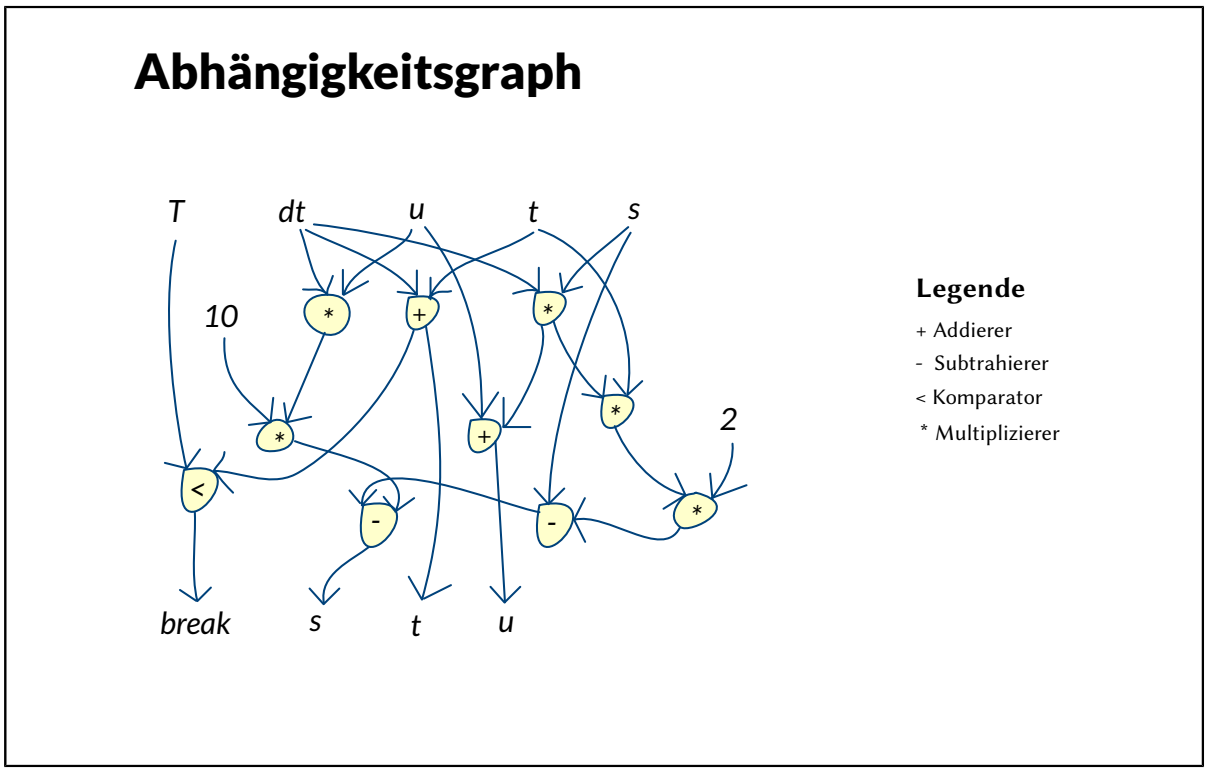
u = u + du = u + s*dt

s = s + ds = s + d(du / dt) = s + (d2u / dt)

mit "(d2u / dt) = -2t*du - 10u/dt" folgt

s = s - 2t*s*dt - 10*u*dt

Algorithmensynthese: Abhängigkeitsgraph



Die arithmetischen Operationen und ihre Abhängigkeiten voneinander stellen den Datenfluss eines Algorithmus dar. Datenabhängigkeiten entstehen dadurch, dass bestimmte Operationen zur Ausführung die Zwischenergebnisse vorangegangener Operationen benötigen. Aus den Datenabhängigkeiten ergibt sich somit eine Reihenfolge, in der die arithmetischen Operationen ausgeführt werden müssen.

Aus dem durch die Beschreibung in einer Hochsprache vorgegebenen Ablaufschema eines Algorithmus wird in der Algorithmensynthese zunächst ein so genannter Abhängigkeitsgraph extrahiert. In diesem Graphen sind alle Informationen über den Ablauf und die notwendigen arithmetischen Operationen enthalten. Der Graph enthält noch keine Informationen über den Zeitpunkt (Taktschritt), in dem eine Operation auszuführen ist. Es ist nur die extrahierte Reihenfolge der Operationen enthalten.

Während der Extraktion werden bereits Techniken zur Optimierung angewandt, um z. B. überflüssige Operationen im Algorithmus zu erkennen und zu entfernen. Hier kann auf bekannte Verfahren, die bei konventionellen Hochsprachenkompilern zur Anwendung kommen, zurückgegriffen werden.
Ein typisches Beispiel für eine solche Optimierung ist das Zusammenführen von identischen Berechnungen und das Verändern der Reihenfolge von Anweisungen, um zum Beispiel ein Zwischenergebnis einer Berechnung mehrfach für folgende Berechnungen verwenden zu können. Dabei ist darauf zu achten, dass durch die Optimierungen das Ergebnis des Algorithmus nicht verändert wird.

Je nach Vorgabe des Entwicklers kann bereits zu diesem Zeitpunkt eine Anpassung der Algorithmusstruktur an die gewünschten Kriterien der resultierenden Schaltung vorgenommen werden. So kann zum Beispiel das Assoziativgesetz auf die arithmetischen Operationen angewendet werden, oder es können vorhandene Schleifen aufgelöst werden (loop unrolling).

Algorithmensynthese: Schleifen

Schleifen

Ggf. Auflösung von Schleifen möglich.
- Anzahl der Schleifendurchläufe ist datenabhängig.
- Anzahl der Schleifendurchläufe ergibt sich meist erst zur Laufzeit.
- Hoher Ressourcen-Bedarf

```
for i=1...n
  x[i]=0;
next i
```

```
x[1]=0;
x[2]=0;
.....
x[n]=0;
```

Verschachtelte Schleifen
- Werden von innen nach außen abgebildet.
- Innere Schleife wird als unteilbare Operation betrachtet.

```
for i=1...n
  for j=1...m
    x[i,j]=0;
  next j
next i
```

```
for i=1,n
  vx[i]=0;
next i
```

Für Schleifen kann sich eine Auflösung der Schleife und damit eine Parallelisierung der einzelnen Durchläufe anbieten, um die Ausführungsgeschwindigkeit des Algorithmus zu erhöhen. Da dies jedoch nur möglich ist, wenn die einzelnen Durchläufe voneinander datenunabhängig sind und sich die Anzahl der benötigten Schleifendurchläufe nicht erst zur Laufzeit des Algorithmus ergibt, kann diese Methode nur selten angewendet werden. Eine Parallelisierung aller Schleifen hätte auch einen sehr hohen Ressourcenbedarf der Schaltung zur Folge. Daher werden die meisten Schleifen auch in der Hardware sequenziell abgearbeitet. Bei der Abbildung ineinander verschachtelter Schleifen wird zuerst für die innerste Schleife ein Taktschema erstellt und dieses dann als eine einzelne, unteilbare Operation mit einer Laufzeit von n Taktschritten behandelt. Die in der Hierarchie nächst äußere Schleife benutzt diesen neu entstandenen Operator genau wie alle anderen Operatoren. Dieses Verfahren setzt sich so lange fort, bis die äußere Schleife abgebildet ist.

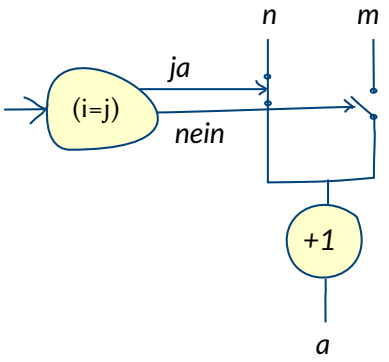
Algorithmensynthese: Bedingte Verzweigungen

Bedingte Verzweigungen

Abbildung bedingter Verzweigungen

→ "mutual exclusion" nutzen
Beispiel: Jeder Zweig eine Addition
→ Nur ein Addierer notwendig

```
if (i==j)
    a = n+1
else
    a = m+1
```



Bei der Abbildung einer bedingten Verzweigung kann zur Ressourcenminimierung ausgenutzt werden, dass die Zweige niemals gleichzeitig ausgeführt werden (mutual exclusion). So kann für den Fall, dass in jedem Zweig eine Addition ausgeführt werden muss, darauf verzichtet werden, zwei Addierer zu implementieren. Es reicht ein Addierer, der durch den Steuerpfad für die arithmetischen Operationen des jeweils aktiven Zweigs verwendet wird.

Algorithmensynthese: Schritte der Algorithmensynthese

Schritte der Algorithmensynthese

Abbildung des Abhängigkeitsgraphen
→ Zwei Dimensionen:
- Taktschema
- Hardwareressourcen

Abbildung auf Taktschema (Ablaufplanung, Scheduling)
→ Festlegung, welche Operation in welchem Taktschritt ausgeführt wird

Abbildung auf Hardwareressourcen
→ Festlegung, welche Operation von welcher Ressource durchgeführt wird

→ Zwei Schritte:
- Ressourcenreservierung (resource allocation)
- Ressourcenzuordnung (resource assignment)

Um den Algorithmus in eine Schaltung zu überführen, ist es nötig, den Abhängigkeitsgraphen in zwei Dimensionen abzubilden:

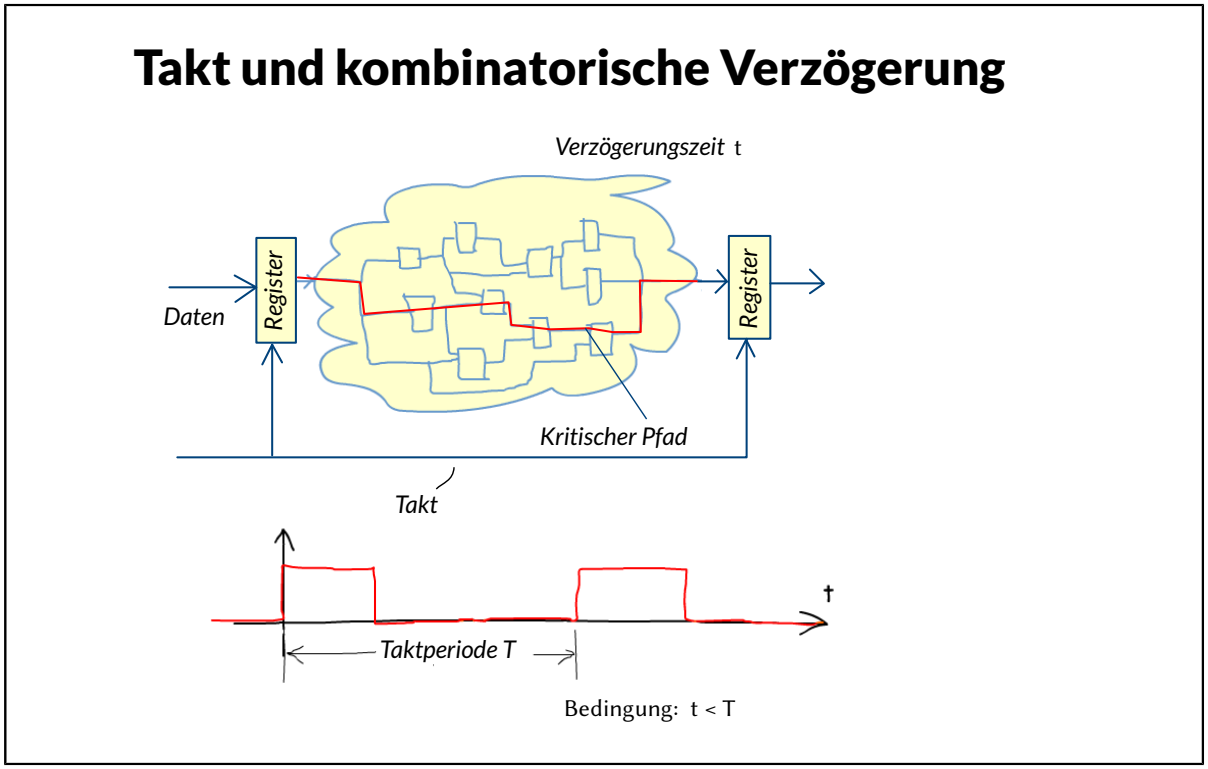
- Taktschema (Performanceoptimierung)
- Hardwareressourcen (Kostenoptimierung)

Die Abbildung auf ein Taktschema, das so genannte Scheduling, dient der Erstellung eines Zeitplans zur Ausführung des Algorithmus. Die Abbildung auf die vorhandenen Hardwareressourcen legt fest, welche Operation von welcher Ressource ausgeführt wird. Dieses geschieht in zwei Schritten.

- Ressourcenreservierung (resource allocation)
- Ressourcenzuordnung (resource assignment)

Die drei Schritte Scheduling, Resource Allocation und Resource Assignment können nicht als unabhängige Teilaufgaben durchgeführt werden, da sie sich gegenseitig stark beeinflussen. So ist die Erstellung eines Taktschemas nur möglich, wenn bereits Ressourcen reserviert sind. Wie viele Ressourcen zu reservieren sind, hängt jedoch vom erstellten Taktschema ab.

Algorithmensynthese: Takt und kombinatorische Verzögerung



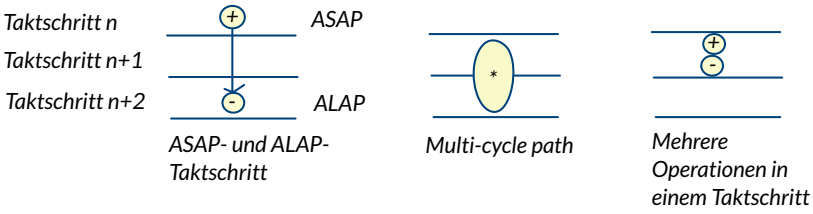
Bei synchronen Schaltungen, die hier ausschließlich behandelt werden, übernehmen speichernde Elemente wie Register oder Flip-Flops Eingangsdaten zu einem definierten Zeitpunkt, der durch einen Takt (z. B. seine positive Flanke) vorgegeben ist. Zwischen zwei Taktschritten werden die Daten in einer kombinatorischen Logik verarbeitet. Die dabei auftretende Gesamtverzögerungszeit t (kombinatorische Verzögerung) muss grundsätzlich kleiner sein als die Taktperiode T.

Der kritische Pfad ist der längste Pfad, den ein Signal durch einen kombinatorischen Schaltungsteil zwischen zwei speichernden Elementen innerhalb einer Taktperiode zurückzulegen hat. Die Länge des Pfads wird durch die Anzahl der zu durchlaufenden logischen Verknüpfungen eines Signals bestimmt, d.h.die Gesamtverzögerungszeit ergibt sich als Summe der Gatter- und Leitungsverzögerungszeiten auf dem Pfad. Der kritische Pfad bestimmt somit die maximal erreichbare Taktfrequenz einer Schaltung.

Algorithmensynthese: Scheduling – Erstellung eines Taktschemas

Scheduling - Erstellung eines Taktschemas

- Taktschema: Verteilung der Operationen auf die Taktschritte.
- Taktschema ist abhängig von Hardwareressourcen.
- Für Operationen jenseits des längsten Pfades:
  - ASAP-Taktschritt (As Soon As Possible)
  - ALAP-Taktschritt (As Late As Possible)
- "Multi-cycle path": Mehrere Taktschritte für eine Operation
- Mehrere Operationen in einem Taktschritt sind möglich



ASAP- und ALAP-Taktschritt

Multi-cycle path

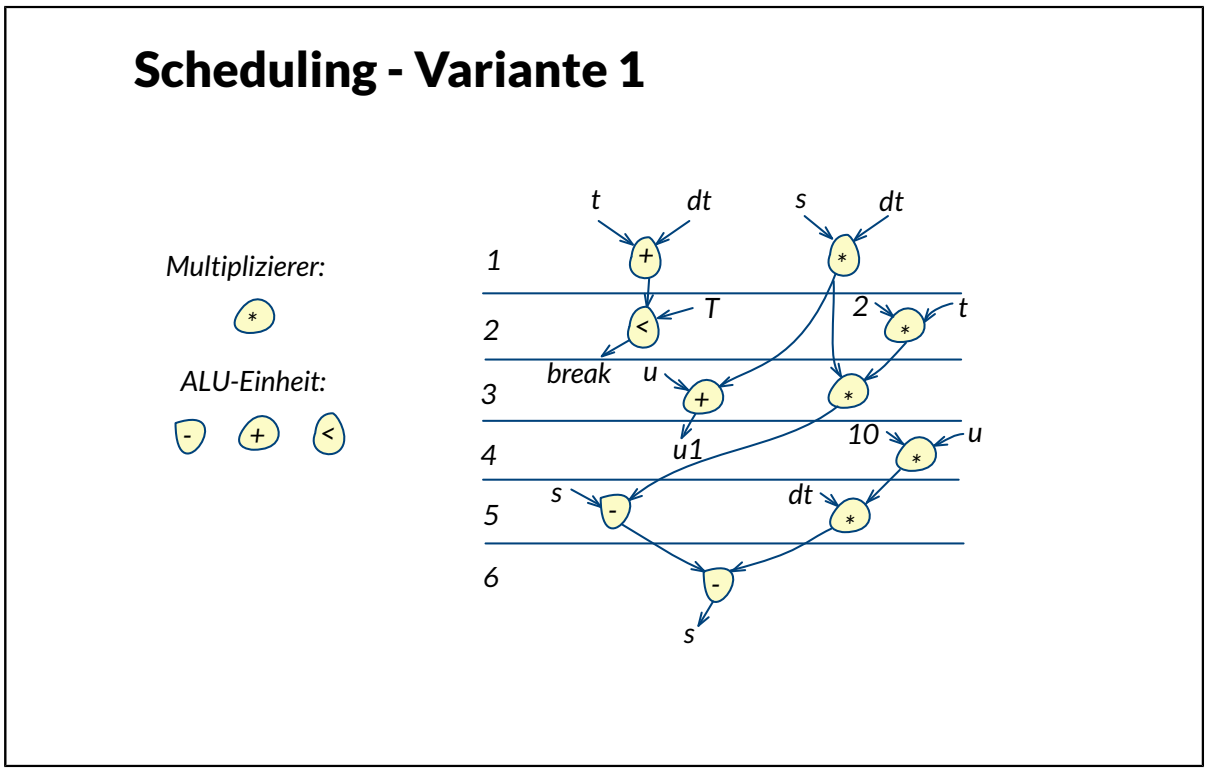
Mehrere Operationen in einem Taktschritt

Mit der Erstellung eines Taktschemas erfolgt die Festlegung, in welchem Taktschritt die einzelnen Operationen des Algorithmus ausgeführt werden. Das Taktschema ist dabei erheblich von der Anzahl der vorhandenen Hardwareressourcen abhängig. So müssen alle Operationen des Algorithmus zwangsläufig sequenziell abgearbeitet werden, wenn nur eine Ressource zur Ausführung von arithmetischen Operationen zur Verfügung steht. Sind jedoch Ressourcen in unbegrenzter Anzahl vorhanden, kann die Ausführung bis zum theoretisch möglichen Grad, der durch die Struktur des Algorithmus vorgegeben wird, parallelisiert werden.

Ist die Anzahl der zur Verfügung stehenden Ressourcen mindestens so groß wie die Anzahl der benötigten, entspricht die minimale Anzahl benötigter Taktschritte dem längsten Pfad des abzubildenden Algorithmus. Für die Operationen, die nicht auf diesem Pfad liegen, entsteht dabei ein Zeitfenster, welches den frühestmöglichen und spätestmöglichen Taktschritt angibt, in dem die Operation ausgeführt werden kann. Der früheste Taktschritt ist dabei der so genannte ASAP-Taktschritt (As Soon As Possible) und der späteste der so genannte ALAP-Taktschritt (As Late As Possible).

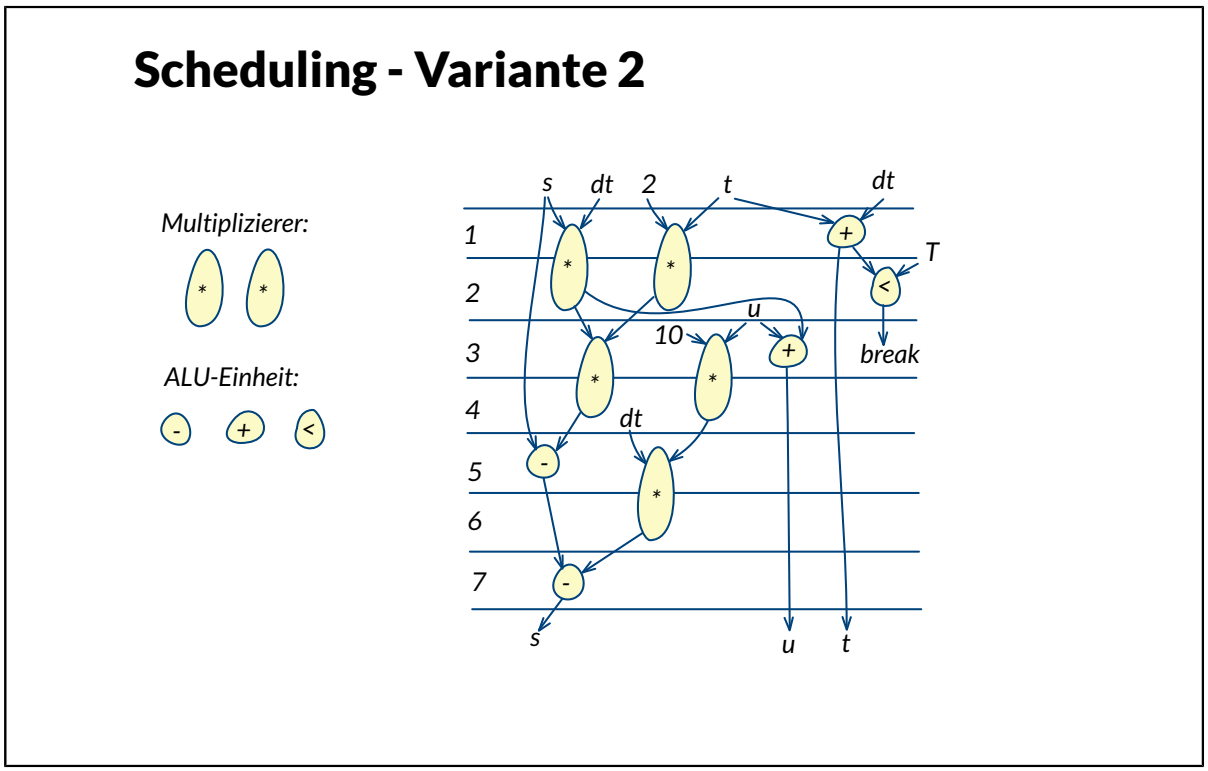
Wird für die Ausführung einer Operation mehr als ein Taktschritt benötigt, besteht die Möglichkeit, die Dauer dieser Operationen im Datenpfad auf mehrere Taktschritte des Steuerpfades auszudehnen. Der kombinatorische Pfad wird dabei nicht durch Register unterbrochen, sondern es wird bewusst akzeptiert, dass die kombinatorische Verzögerung in diesem Schaltungsteil größer ist als eine Taktperiode. Der so entstehende Pfad wird "multi-cycle path" genannt. Die Steuerung des Datenpfades wird in diesem Fall so angepasst, dass das Ergebnis des multi-cycle path erst nach der minimal notwendigen Anzahl von Taktschritten erwartet wird. Andererseits ist es möglich, mehrere Operationen sequenziell innerhalb eines Taktschrittes auszuführen, sofern die Summe aller Verzögerungen die Länge eines Taktschrittes nicht überschreitet.

Algorithmensynthese: Scheduling Variante 1



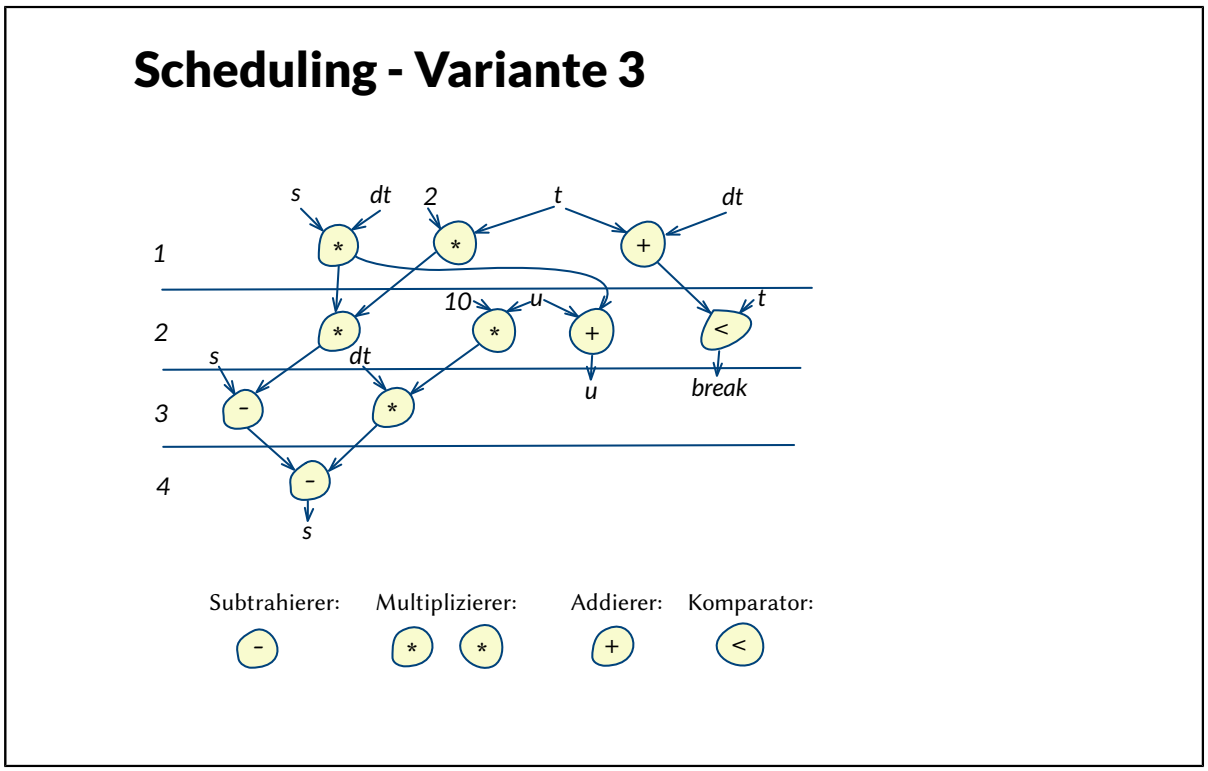
Als erste Scheduling-Variante ist hier eine Realisierung dargestellt mit einem schnellen, d.h. in einem Taktschritt arbeitenden Multiplizierer und einer ALU, die Additionen, Subtraktionen sowie Vergleiche ausführen kann. Mit diesen Ressourcen ist eine Abarbeitung des Algorithmus in 6 Taktschritten möglich.

Algorithmensynthese: Scheduling Variante 2



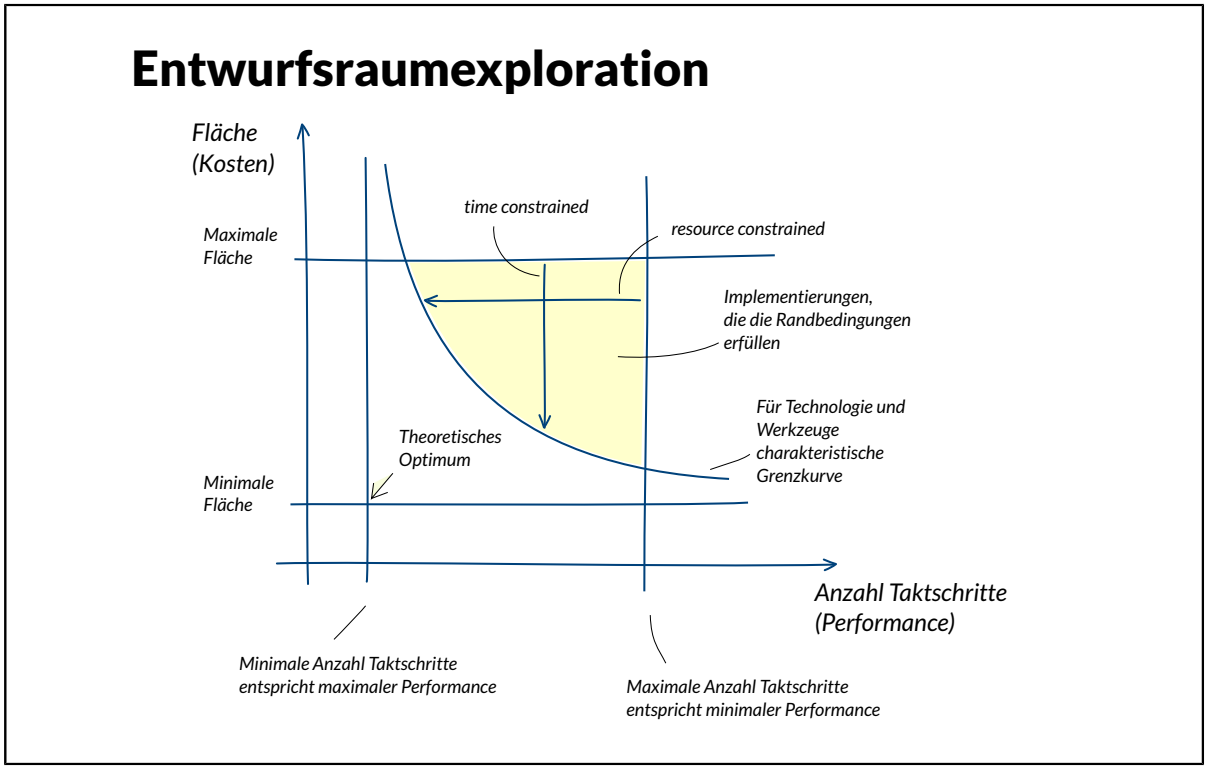
Als zweite Scheduling-Variante ist hier eine Realisierung mit zwei langsamen Multiplizierern dargestellt. Obwohl die beiden Multiplizierer parallel arbeiten können, sind für die Abarbeitung des Algorithmus 7 Taktschritte nötig.

Algorithmensynthese: Scheduling Variante 3



Hier ist eine Variante dargestellt, die zwei schnelle Multiplizierer, einen Addierer, einen Subtrahierer und einen Komparator verwendet. Damit ist eine Abarbeitung des Algorithmus in 4 Taktschritten möglich.

Algorithmensynthese: Entwurfsraumexploration



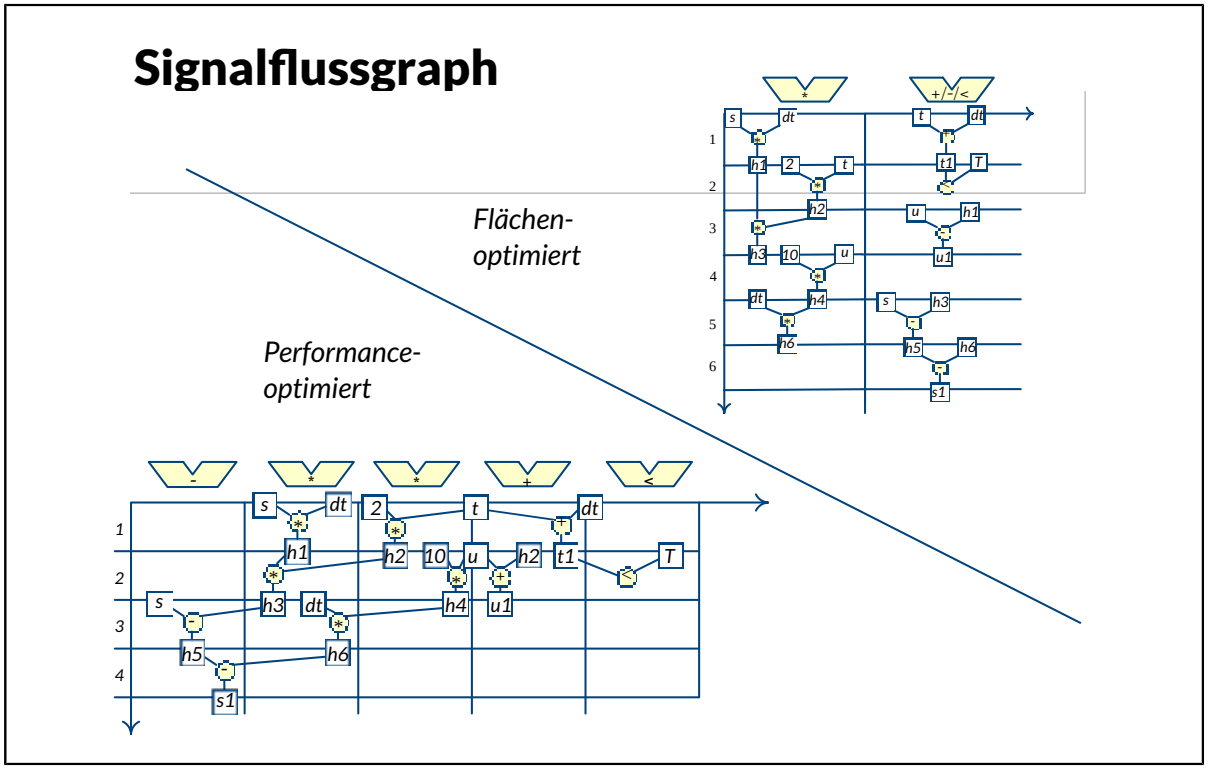
In der Abbildung wird der durch Fläche (Kosten) und Anzahl Taktschritte (Performance) beschriebene Entwurfsraum dargestellt. Die Grenzen, die durch die Spezifikation oder durch technologieabhäningige Parameter vorgegeben sind, sind in der Abbildung eingezeichnet.

Bei der Erstellung eines Taktschemas kann von zwei unterschiedlichen Randbedingungen ausgegangen werden:

- Anzahl der Ressourcen und damit die Fläche ist festgelegt (resource constrained)
- Anzahl der Taktschritte und damit die Performance ist festgelegt (time constrained)

Im Falle einer Festlegung der Anzahl der Ressourcen, ist es das Ziel des Scheduling, den Algorithmus mit den zur Verfügung stehenden Ressourcen in möglichst wenig Taktschritten abzuarbeiten.
Falls die Anzahl der Taktschritte festgelegt wird, soll der Algorithmus mit einer möglichst geringen Anzahl von Ressourcen innerhalb der vorgegebenen Anzahl von Taktschritten ausgeführt werden.
Diese beiden Fälle sind Teil des allgemeinen Optimierungsproblems, welches im Rahmen der Algorithmensynthese zu lösen ist. Das Optimierungsziel sind die beiden gegensätzlichen Kriterien minimaler Flächenbedarf und maximale Performance der Schaltung. Ein minimaler Flächenbedarf entspricht dabei einem minimalen Ressourcenbedarf, da jede zusätzlich verwendete Ressource auch zusätzliche Chipfläche beansprucht. Während der Optimierung ist darauf zu achten, dass die vorgegebenen Randbedingungen (constraints) eingehalten werden. Die Abbildung zeigt den durch Randbedingungen und Technologiegrenzen festgelegten gültigen Bereich im Entwurfsraum.

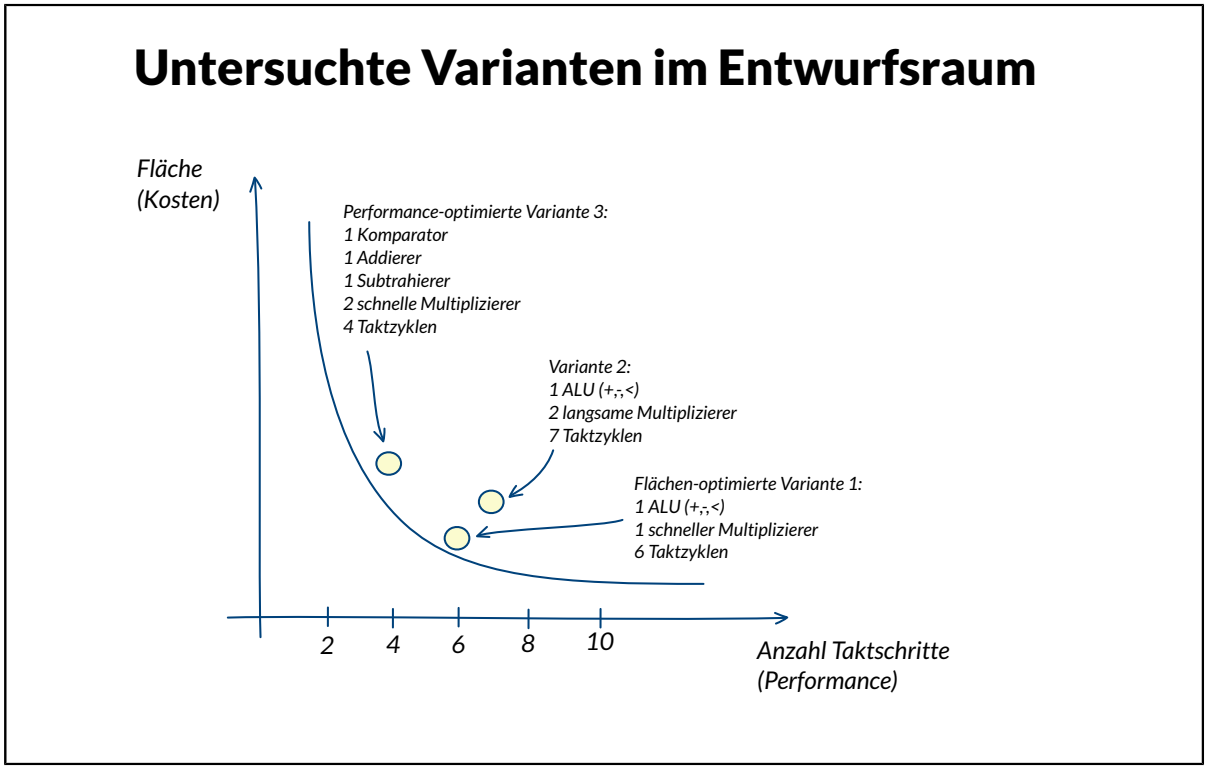
Algorithmensynthese: Signalflussgraph



Ein Signalflussgraph enthält alle notwendigen Informationen, um eine Schaltung auf RT-Ebene zu generieren. Er legt fest, welche Operation in welchem Taktschritt mit welcher Hardwareressource ausgeführt wird. Im Signalflussgraphen ist von oben nach unten die Abfolge der Taktschritte zu sehen. An den Übergängen von einem Takt zum nächsten sind die Register angeordnet, da sie die Speicherung von Daten über den aktuellen Taktzyklus hinaus übernehmen.

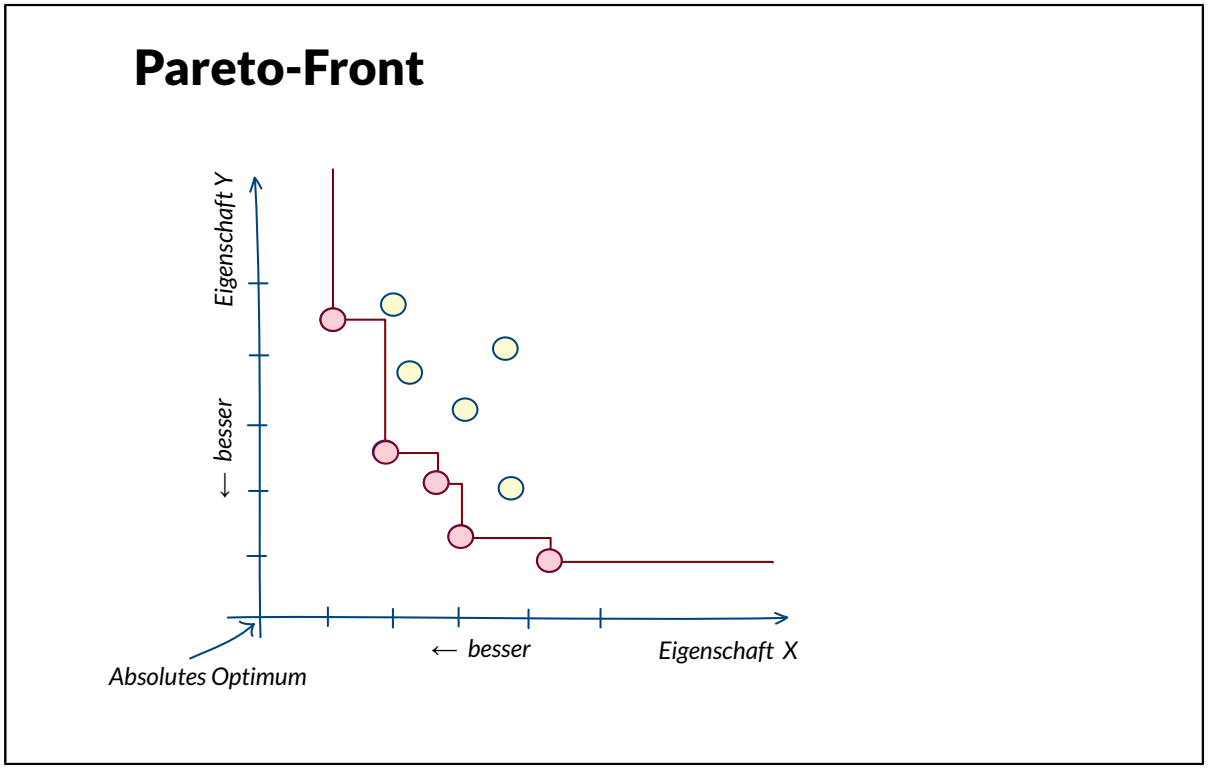
Die beiden Abbildungen zeigen noch einmal die Signalflussgraphen des Beispielalgorithmus mit unterschiedlichen Optimierungszielen. Beim Kosten-optimierten Signalflussgraphen erfolgte eine stärkere Reduktion des Ressourcenbedarfs, während beim Performance-optimierten Signalflussgraphen die Geschwindigkeit auf Kosten eines größeren Ressourcenbedarfs erhöht wurde.

Algorithmensynthese: Varianten im Entwurfsraum



Im Entwurfsraum ergibt sich für die untersuchten Varianten die gezeigte Anordnung. Unter der Bedingung , dass ein langsamer Multiplizierer zwar flächengünstiger ist als ein schneller, zwei langsame jedoch mehr Fläche benötigen als ein schneller, ergibt sich die Variante 1 als flächengünstigste.

Algorithmensynthese: Pareto-Front



Als gute Lösungen kommen nur Varianten in betracht, die auf einer sogenannten Pareto-Front liegen. Diese Front ist wie folgt definiert:

Alle Punkte auf der Pareto-Front sind solche, für die es nicht möglich ist, Punkte mit einer verbesserten Eigenschaft zu finden, ohne dass zugleich eine andere Eigenschaft verschlechtert werden müsste.

Algorithmensynthese: Ressourcenreservierung und Ressourcenzuordnung

Ressourcenreservierung und Ressourcenzuordnung

Ressourcenreservierung (resource allocation)

-> Die Anzahl ergibt sich aus Taktschema.
-> Es werden so viele Instanzen benötigt, wie parallel in einem Takt sein müssen.
-> Zur Speicherung der Zwischenergebnisse werden Register benötigt (Wiederverwendung möglich).

Ressourcenzuordnung (resource assignment)

-> Ressourcen sollen möglichst gleichmäßig ausgelastet werden.
-> Anzahl Multiplexer
- abhängig von Datenstrom
- in der Regel ein Multiplexer pro Makrozelle/Register.

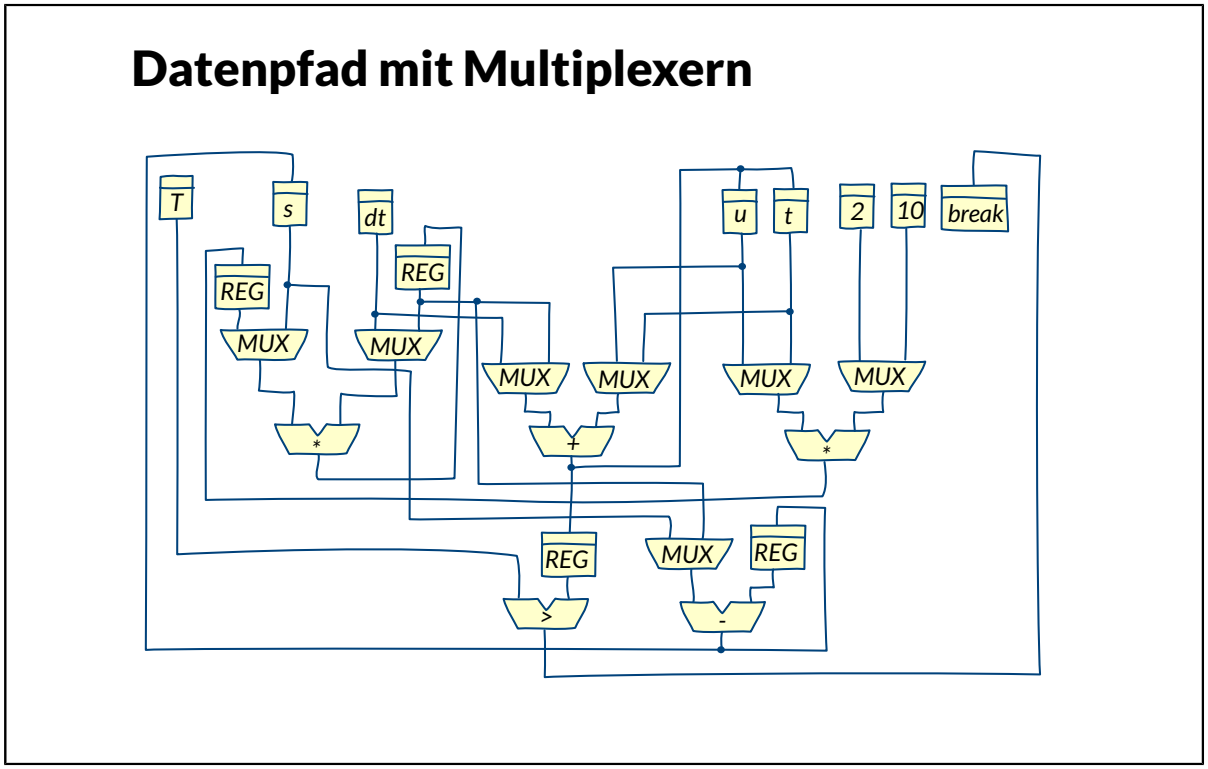
-> Anzahl Register
- ergibt sich aus maximaler Anzahl, der in einem Taktschritt zu speichernden Daten.

Bei der Ressourcenreservierung werden die Ressourcen instanziiert, die während der Erstellung des Taktschemas verplant wurden. Die Anzahl der einzelnen Ressourcen ergibt sich dabei direkt aus dem Taktschema. Von einem bestimmten Ressourcentyp werden genau so viele Einheiten benötigt, wie maximal in einem Taktschritt parallel aktiv sein müssen. Bei der Ressourcenzuordnung werden den einzelnen Operationen die instanziierten Ressourcen zugeordnet.

Für alle Daten, deren Generierung und Weiterverarbeitung in unterschiedlichen Taktschritten erfolgt, werden Register zur Speicherung benötigt. Die Anzahl der Register ergibt sich dabei aus der maximalen Anzahl gleichzeitig in einem Taktschritt zu speichernden Daten. Durch die Mehrfachverwendung von Registern für Daten, die ausschließlich in unterschiedlichen Taktschritten gültig sind, können Ressourcen eingespart werden. Das jeweils abzuspeichernde Datum wird in diesem Fall durch einen dem Registereingang vorangeschalteten Multiplexer selektiert. Die Dauer, die ein Datum in einem Register gespeichert werden muss, wird durch den ermittelten Ablaufplan des Algorithmus vorgegeben.

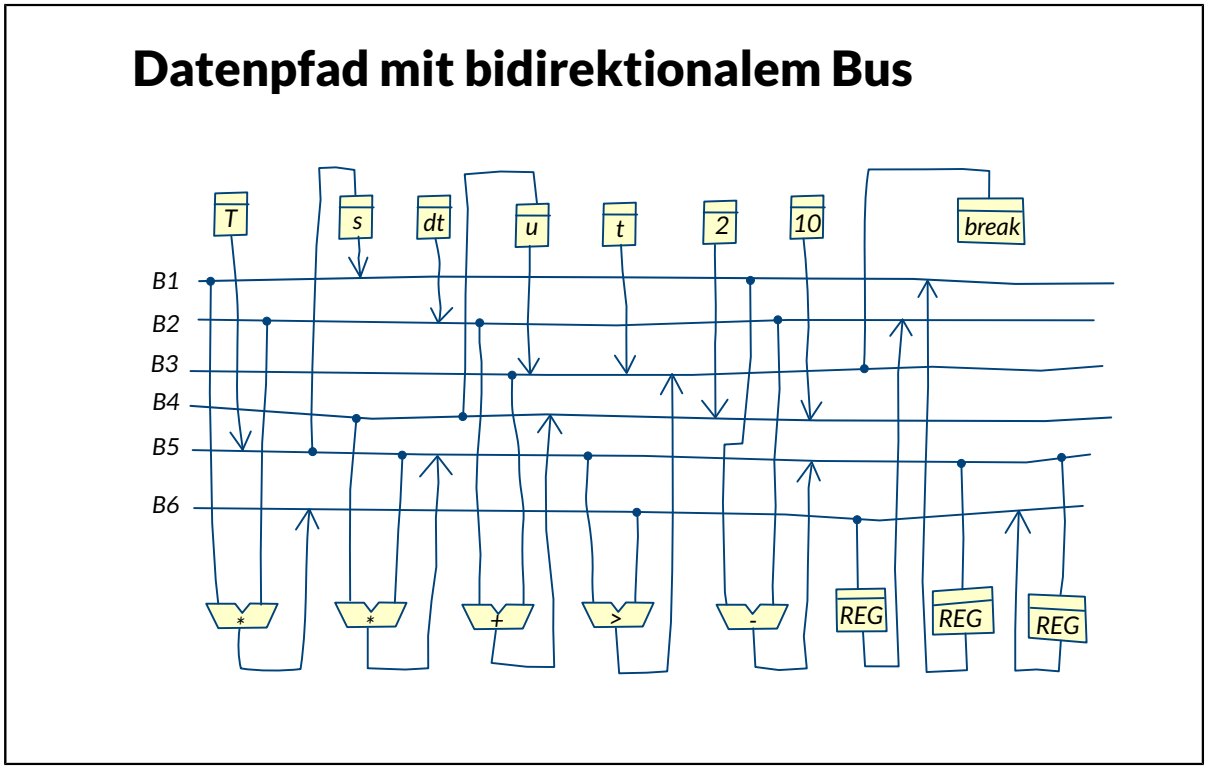
Wenn alle notwendigen Ressourcen instanziiert sind, müssen diese noch den einzelnen Operationen zugeordnet werden. Die Zuordnung sollte dabei so erfolgen, dass ein möglichst optimaler Datenfluss durch den Datenpfad gewährleistet ist und die Anzahl bzw. Breite der Multiplexer an den Blockeingängen minimal wird. Für die Minimierung der Multiplexerressourcen sollten die arithmetischen Makrozellen möglichst gleichmäßig ausgelastet werden, da eine Zuordnung vieler Operationen zu nur einer Ressource zu sehr komplexen und damit langsamen Multiplexerstrukturen führen würde.

Algorithmensynthese: Datenpfad mit Multiplexern



Aus dem Signalflussgraphen lassen sich die Struktur des Datenpfads und das Zustandsübergangsdiagramm des Steuerpfads generieren. Die Struktur des Datenpfads kann dabei aus der Struktur des Signalflussgraphen abgeleitet werden, indem für jeden Taktschritt bestimmt wird, welche Daten einem Block zugeführt werden müssen. Die Anzahl aller Möglichkeiten ergibt dabei die notwendige Anzahl der Eingänge des Eingangsmultiplexers. Alle in Frage kommenden Makrozellen, die eine Datenquelle für die auszuführenden Operationen sein können, müssen mit den Multiplexern verbunden werden. Datenquelle und Datensenke sind sowohl Register als auch Makrozellen, die arithmetische Operationen ausführen. Der sich daraus ergebende Datenpfad für das hier verwendete Beispiel in der Variante 3 ist in der Abbildung dargestellt.

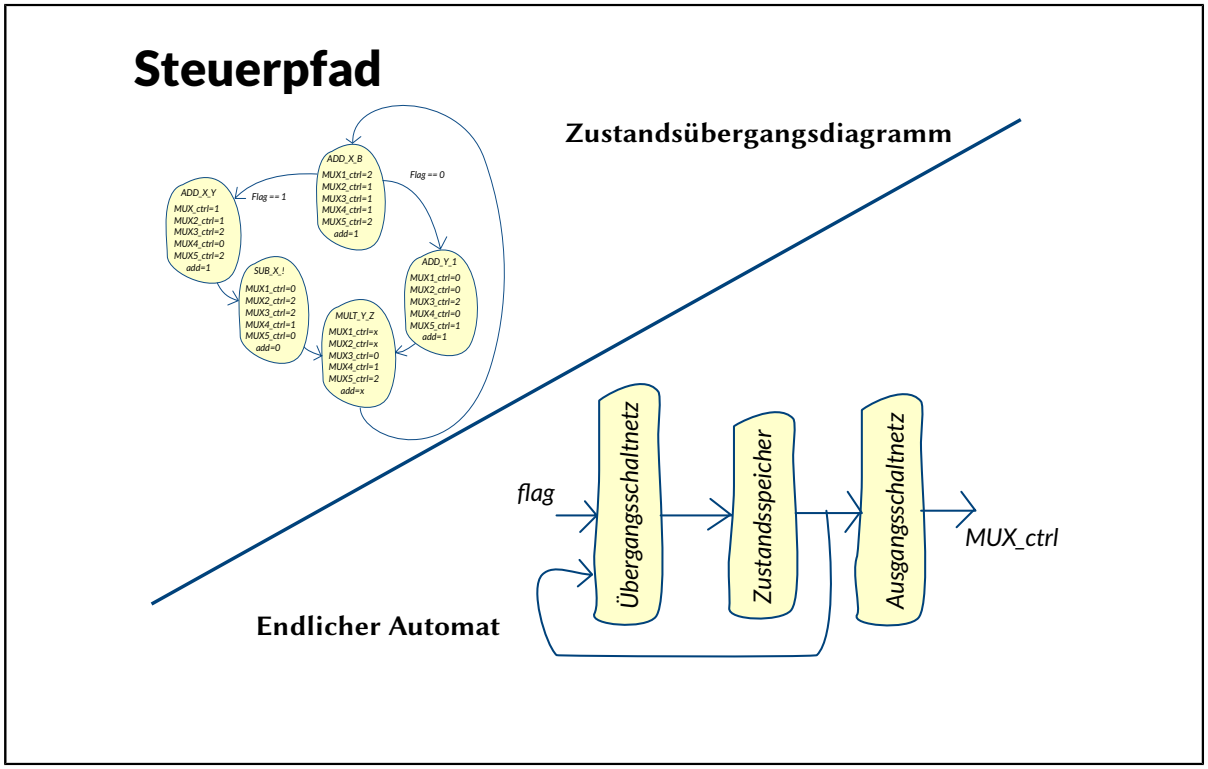
Algorithmensynthese: Datenpfad mit bidirektionalem Bus



Als Alternative zu einer Realisierung mit Multiplexern kann der Datenpfad auch mit einem bidirektionalen Bus zum Datentransfer zwischen den arithmetischen Makrozellen aufgebaut werden. Alle Makrozellen lesen und schreiben dabei auf einen gemeinsamen Datenbus. Über diesen werden die einzelnen Operatoren zu den Verarbeitungseinheiten transferiert. Das Bild zeigt exemplarisch den Aufbau eines Datenpfads der Variante 3 des Beispiels mit einem bidirektionalen Bus.

Busse sind zwar in der Regel kostengünstiger zu implementieren, da auf die (teuren) Multiplexer verzichtet werden kann, sie sind aber auch wesentlich langsamer als Multiplexerstrukturen.

Algorithmensynthese: Steuerpfad



Zur Beschreibung des Steuerpfads wird aus dem Signalflussgraphen ein so genanntes Zustandsübergangsdiagramm erstellt, welches einen endlichen Automaten beschreibt. Das Zustandsübergangsdiagramm lässt sich ebenfalls in Form eines Graphen darstellen. Die Knoten sind dabei die Zustände und die gerichteten Kanten die Zustandsübergänge. Zu jedem Zustandsübergang gehört ein Eingangsbitmuster, das den Zustandswechsel auslöst. Das resultierende Schaltnetz, welches in Abhängigkeit von den Eingangssignalen und dem aktuellen Zustand den Folgezustand bestimmt, wird Übergangsschaltnetz genannt. Zur Berechnung der Ausgänge des endlichen Automaten dient das so genannte Ausgangsschaltnetz. Wenn das Ausgangsschaltnetz die Ausgänge nur in Abhängigkeit vom aktuellen Zustand berechnet, wird der endliche Automat Moore-Automat genannt. Um einen so genannten Mealy-Automaten handelt es sich, wenn die Ausgänge eine Funktion des aktuellen Zustands und der Eingänge sind. Die Abbildung zeigt ein exemplarisches Zustandsübergangsdiagramm für eine Datenpfadrealisierung mit Multiplexern. Das Zustandsübergangsdiagramm enthält eine bedingte Verzweigung, deren Bedingung ein gesetztes bzw. nicht gesetztes Flag ist. Es handelt sich um einen Moore-Automaten.

Algorithmensynthese: Nachoptimierungsmöglichkeiten

Nachoptimierungsmöglichkeiten

1) Zustandsminimierung
-> Sequenzen übereinstimmender Zustände ermitteln und zusammenfassen.

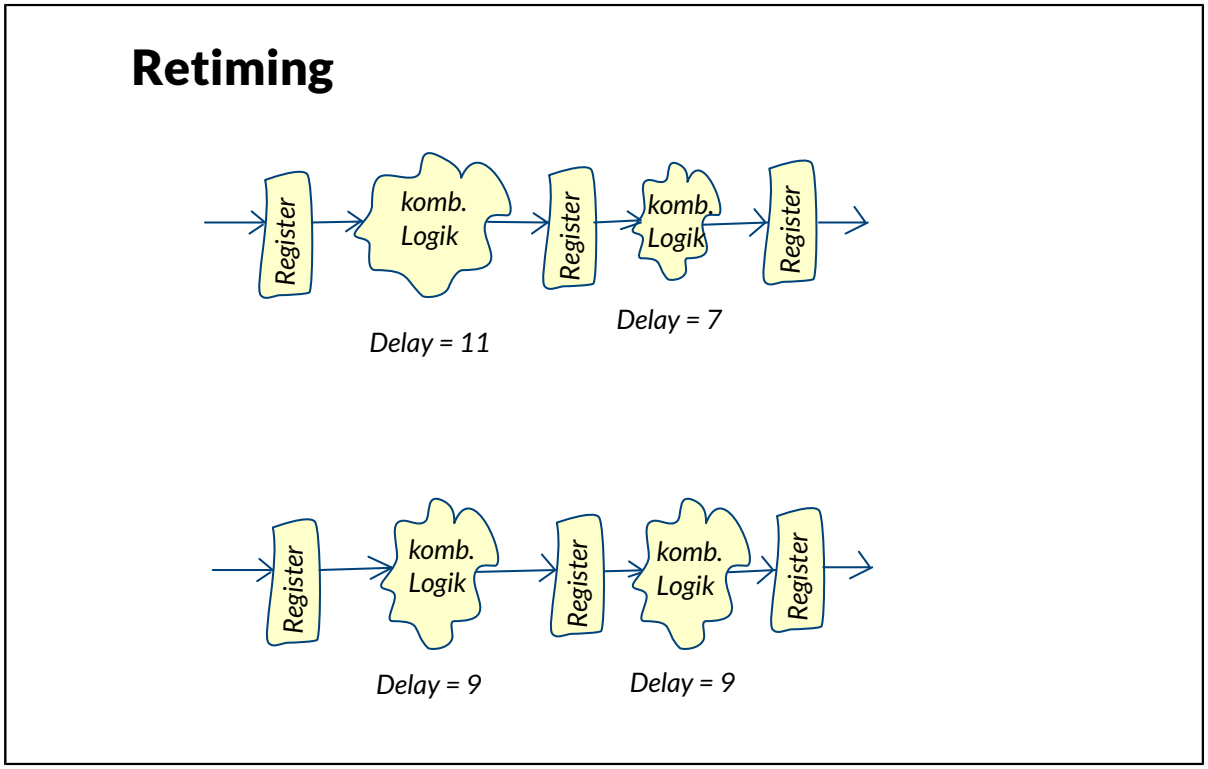
2) Retiming
-> Kombinatorische Schaltungsteile auf die zur Verfügung stehenden Taktzyklen möglichst gleichmäßig verteilen.

3) Pipelining
-> Komplexe Operationen in Teiloperationen (= Pipeline-Stufen) aufteilen.

Eine Reduzierung der notwendigen Ressourcen im Steuerpfad ist durch eine Zustandsminimierung des endlichen Automaten zu erreichen. Dabei wird das Zustandsübergangsdiagramm des endlichen Automaten auf Sequenzen von übereinstimmenden Zuständen untersucht. Zustände stimmen dann überein, wenn sie die gleichen Werte an den Ausgängen erzeugen und ihre Folgezustände identisch sind. In einem solchen Fall lassen sich die Zustände zusammenfassen.

Um die Schaltung auf eine möglichst hohe Taktfrequenz hin zu optimieren, kommt das so genannte Retiming zur Anwendung. Beim Retiming werden die kombinatorischen Schaltungsteile möglichst gleichmäßig auf die zur Verfügung stehenden Taktschritte verteilt. Dadurch wird die Länge des kritischen Pfads der Schaltung minimiert.

Algorithmensynthese: Retiming



Falls eine gleichmäßige Verteilung der kombinatorischen Blöcke einer Schaltung nicht ausreicht, um die gewünschte Taktfrequenz zu erreichen, müssen zusätzliche Register eingefügt werden. Dabei werden kombinatorische Blöcke der Schaltung, deren kritischer Pfad für die geforderte Taktfrequenz zu lang ist, durch Register in kleinere Blöcke unterteilt. Dadurch verkürzt sich der kritische Pfad und die maximal erreichbare Taktfrequenz steigt an. Auf Grund der eingefügten Register erhöht sich die Latenzzeit der Schaltung. Es werden also mehr Taktschritte für die gleiche Aufgabe benötigt. Durch die mögliche Erhöhung der Taktfrequenz kann dieser Geschwindigkeitsverlust aber wieder kompensiert werden.

Eine weitere geschwindigkeitssteigernde Maßnahme stellt das Pipelining dar. Hier wird in jedem Taktschritt immer nur jeweils ein Teil der gesamten Operation abgearbeitet. Das Ergebnis dieser Teilaufgabe wird in Registern gespeichert und im nächsten Taktschritt weiterverarbeitet. Die Teile einer Schaltung, die die einzelnen Teilaufgabe ausführen, werden Pipelinestufen genannt. Hat eine Teiloperation die Pipelinestufe verlassen, kann aus der vorhergehenden Stufe die nächste Operation übernommen und verarbeitet werden. Dies führt im Idealfall dazu, dass in jedem Taktschritt eine neue Operation in die aus den einzelnen Stufen zusammengesetzte Pipeline eingespeist werden kann. So kann trotz einer Verarbeitungszeit von mehreren Taktschritten für eine Operation, mit Pipelining die Gesamtbearbeitungszeit der Operation verringert werden. Ein Nachteil des Pipelinings ist jedoch der erhöhte Ressourcenbedarf der Schaltung, da jede Pipelinestufe ihre eigenen Ressourcen zur Verarbeitung der Operationen benötigt. Eine Mehrfachnutzung ist nur bedingt möglich.

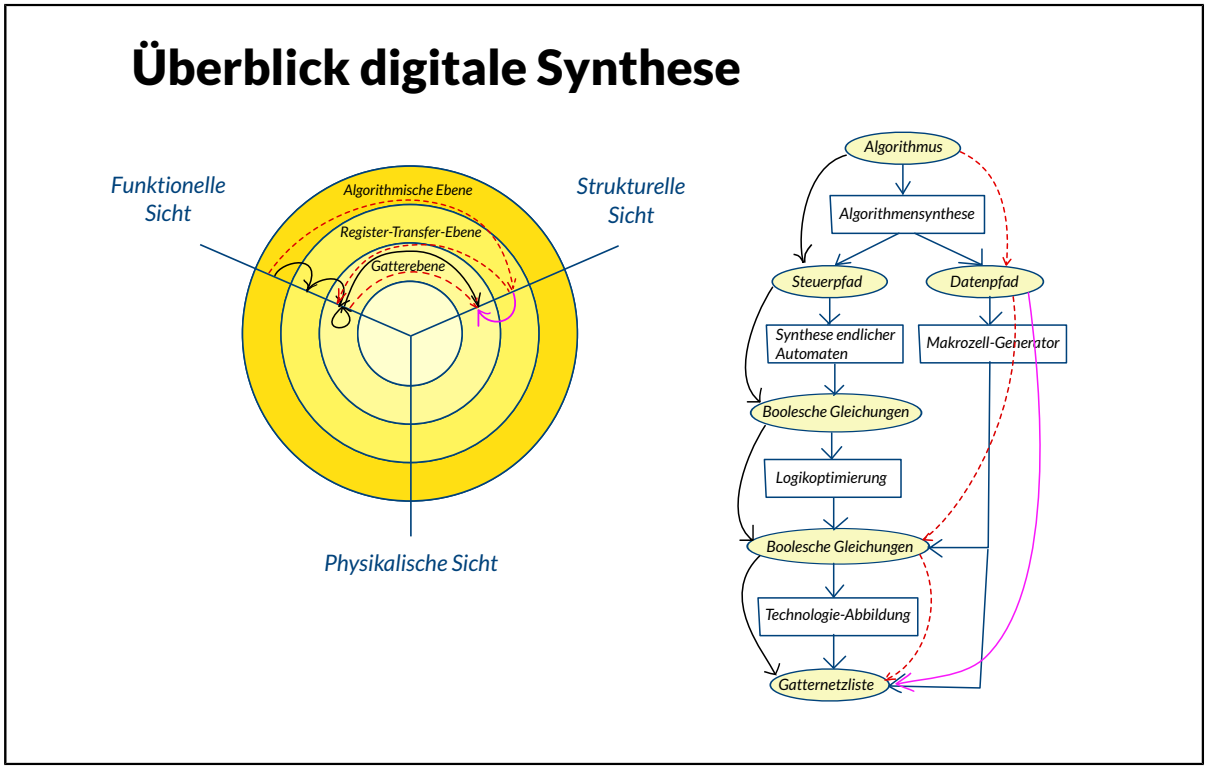
Electronic Design Automation (EDA)

Register-Transfer-Synthese

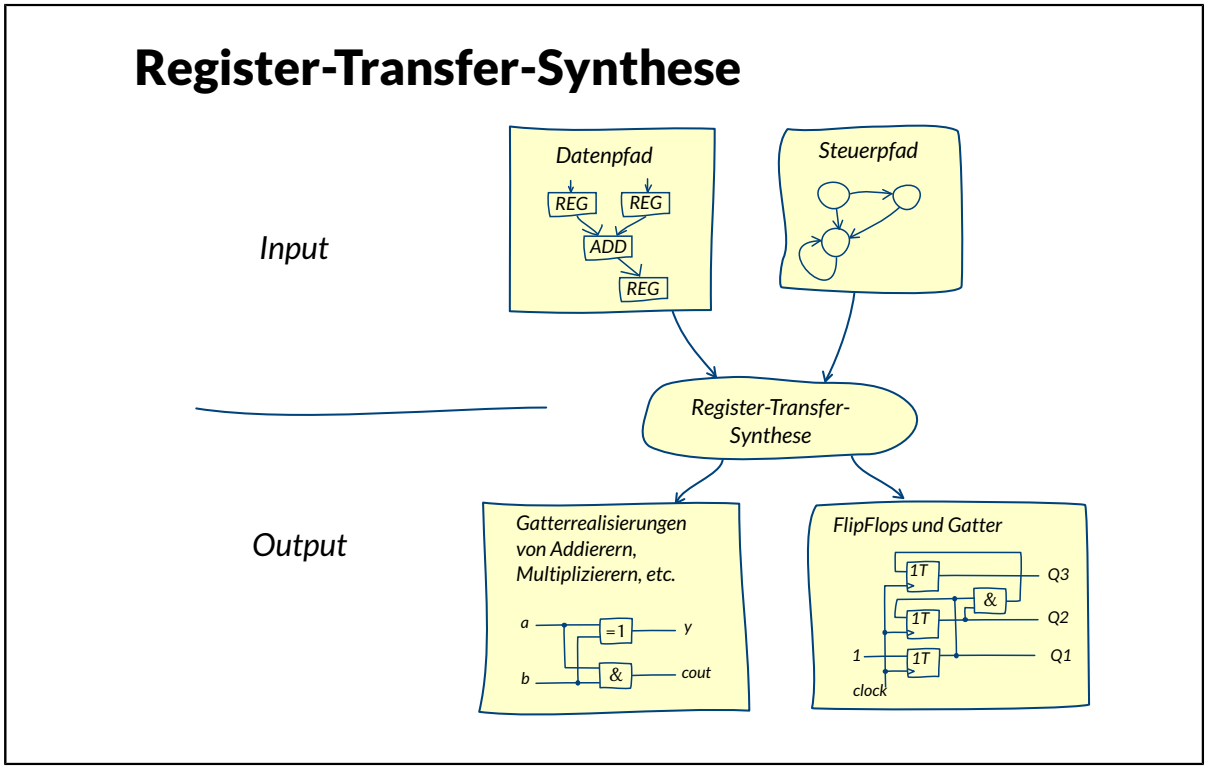
Überblick digitale Synthese
Register-Transfer-Synthese
Makrozellgenerator
Beispiel Addierer (1)
... (2)
... (3)
... (4)
Beispiel Speicher
Synthese endlicher Automaten
Zustandskodierung
FlipFlop-Auswahl und Schaltnetzsynthese

Date of generation: 8.7.2025

Register-Transfer-Synthese: Überblick digitale Synthese



Register-Transfer-Synthese: Register-Transfer-Synthese



Die Register-Transfer-Synthese setzt eine Schaltung auf Register-Transfer-Ebene in eine funktional äquivalente Schaltung auf Gatterebene um. Das Zeitverhalten der Register-Transfer-Ebene wird taktgenau beibehalten. Zur Synthese der Elemente des Datenpfades werden parametrisch steuerbare Generatoren (Makrozell-Generatoren*) eingesetzt, z.B. bei der Abbildung von Operationen wie Multiplikation oder Addition. Zur Synthese des Steuerpfades wird auf spezielle Synthese-Algorithmen für die Abbildung endlicher Automaten zurückgegriffen.

*: Bei der Generierung von Makrozellen handelt es sich eigentlich um einen Syntheseschritt. In der Literatur wird jedoch einheitlich der Begriff Makrozellen- oder Modul-Generator verwendet. Daher behalten wir diesen Begriff entgegen der sonst von uns verwendeten Nomenklatur bei.

Register-Transfer-Synthese: Makrozellgenerator

Makrozellgeneratoren

- Makrozellen:
  Addierer, Multiplizierer, Speicher, Schieberegister, Zähler, ...

- Höhere Komplexität als Standardzellen

- Höhere Regularität/wenige Parameter (z.B. Bitbreite, Wortlänge)

- Unterschiedliche Realisierungen mit unterschiedlichen Eigenschaften:

  Optimierung von:
  - Performance (Verzögerungszeit)
  - Kosten (Fläche),
  - Leistungsaufnahme,
  - ...

Zu den wesentlichen Elementen eines Datenpfades gehören Teilschaltungen wie Addierer, Multiplizierer, Speicher sowie weitere regulär aufgebaute Logikblöcke wie Schieberegister oder Zähler. Diese Teilschaltungen bezeichnen wir als Makrozellen. Makrozellen haben im Gegensatz zu Basis- bzw. Standardzellen eine erheblich höhere Komplexität.

Makrozellen zeichnen sich in der Regel durch eine hohe Regularität aus. Daher lassen sie sich durch wenige Parameter wie die Bitbreite bei einem Addierer oder die Wortlänge und die Anzahl der Worte bei einem Speicher beschreiben. Diese Parameter reichen aus, um mit Hilfe eines Makrozell-Generators aus einfachen Grundzellen die gewünschte Teilschaltung zu erzeugen. Beispielsweise können Speicher aus 1-Bit-Grundzellen oder Addierer aus 1-Bit-Voll-Addierern zusammengesetzt werden.

Register-Transfer-Synthese: Beispiel Addierer (1)

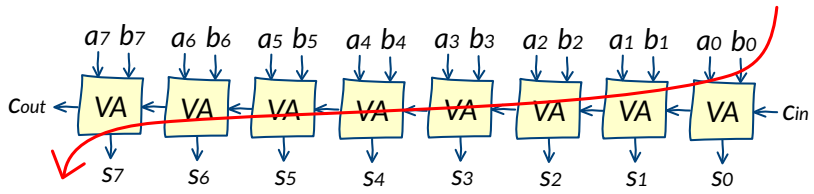
Beispiel Addierer: Variante 1

8-bit-Ripple-Carry-Addierer

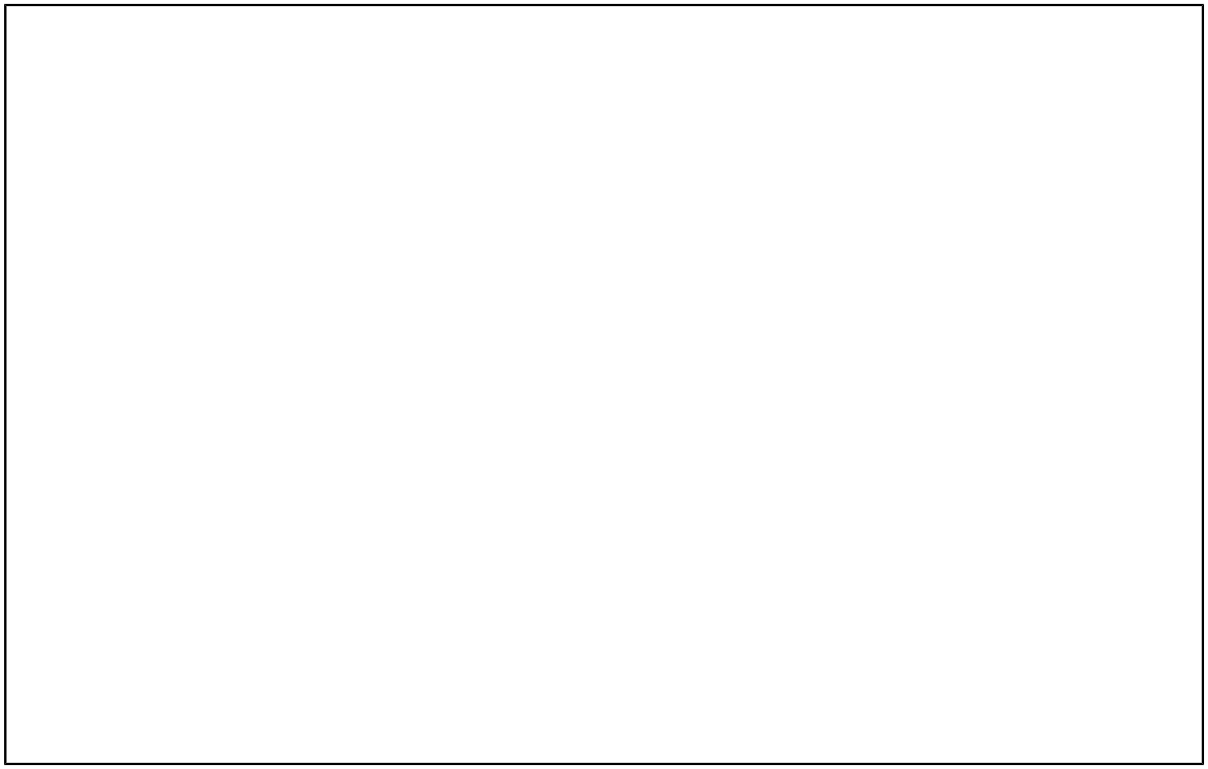
Kritischer Pfad durch:
- 8 Volladdierer

Flächenbedarf:
- 8 Volladdierer

	klein	schnell
8 bit	☒	☐
16 bit	☐	☐



Register-Transfer-Synthese: ... (2)



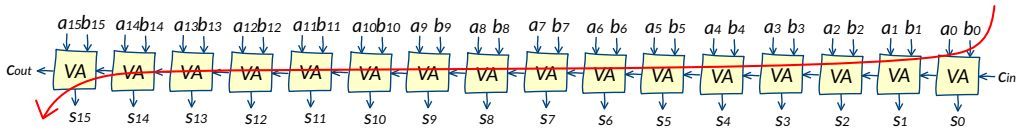
Beispiel Addierer: Variante 3

16-bit-Ripple-Carry-Addierer

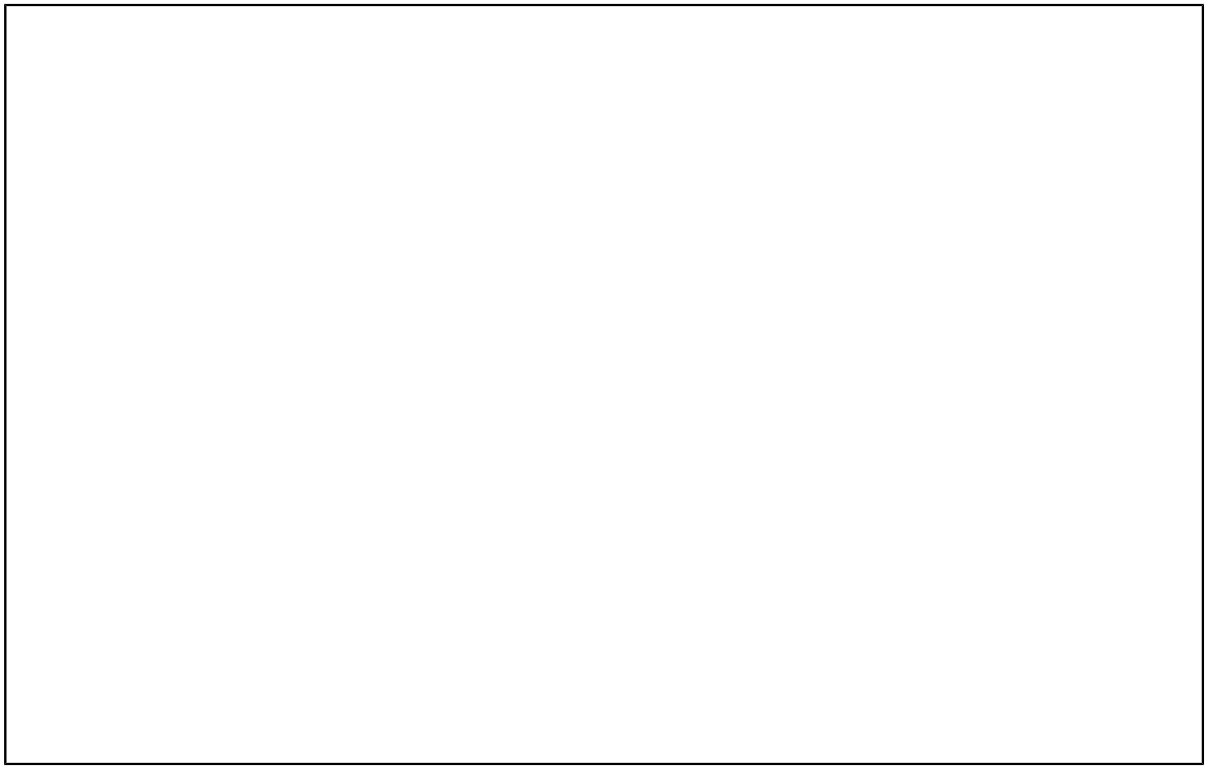
Kritischer Pfad durch:
- 16 Volladdierer

Flächenbedarf:
- 16 Volladdierer

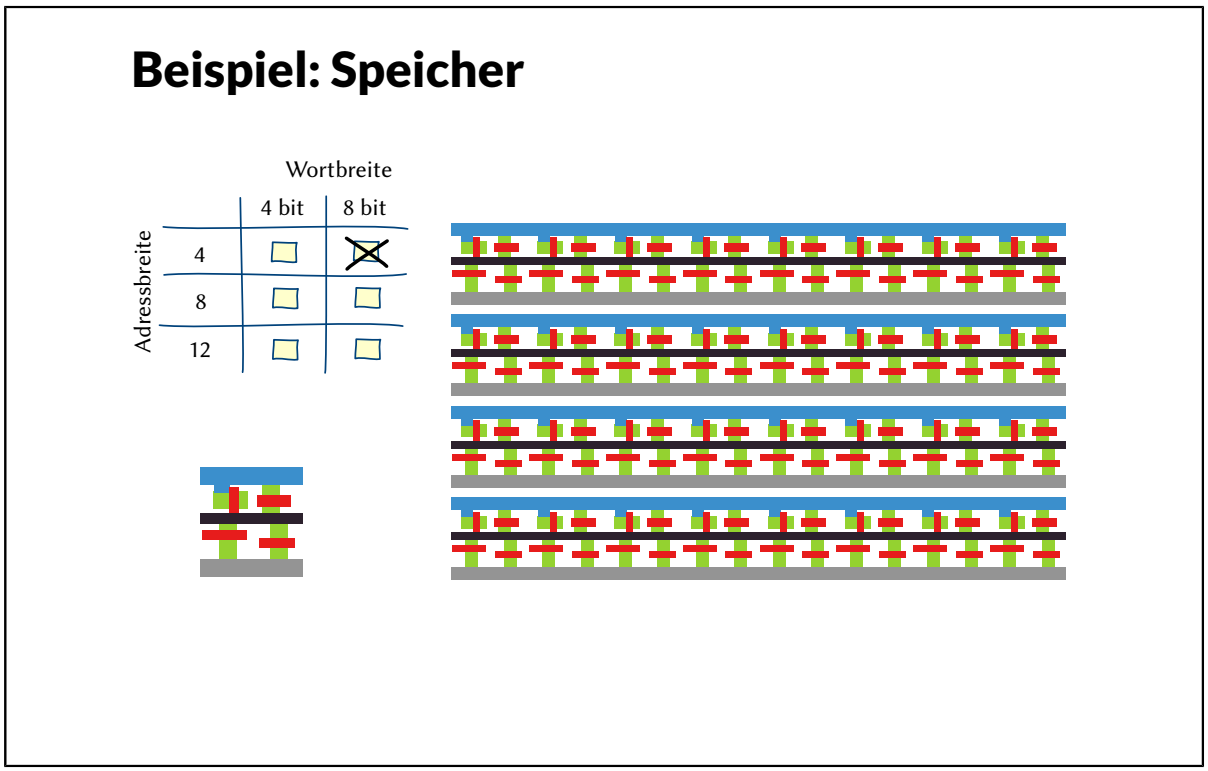
	klein	schnell
8 bit	□	□
16 bit	⊠	□



Register-Transfer-Synthese: ... (4)



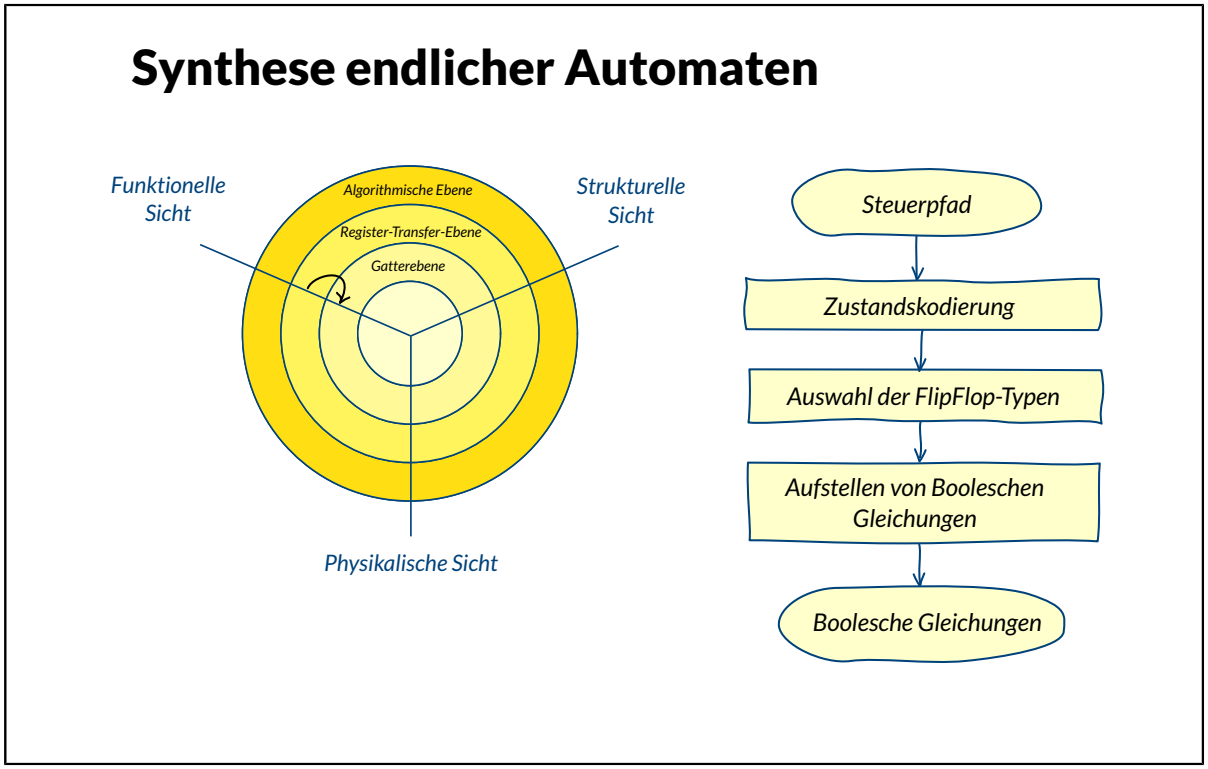
Register-Transfer-Synthese: Beispiel Speicher



Bei Strukturen wie Speichern (DRAM, SRAM, Flash) sind die Grundzellen stark spezialisiert. Das Layout wird manuell entworfen und ist noch optimiert. Ein Makrozell-Generator liefert dann nicht nur eine (zumeist triviale) Darstellung in funktionaler oder struktureller Sicht auf Gatterebene, sondern auch bereits ein Layout. Übliche Darstellungen sind: Ein zeitlich exaktes Modell auf Gatterebene für die Schaltungsverifikation, ein so genanntes Abstract, das im Wesentlichen die äußere Form und die Lage der Anschlussfelder beschreibt und das als Platzhalter für das Makrozell-Layout bei der Platzierung und Verdrahtung der Gesamtschaltung verwendet wird, und schließlich das Layout selbst, das nach Abschluss aller Entwurfsschritte den Platz des Abstracts einnimmt.

Mit einem entsprechenden Makrozell-Generator kann ein Halbleiterhersteller auch bei sehr zeit- und flächenkritischen Makrozellen wie bei eingebetteten dynamischen Speicherfeldern eine korrekte Funktion gewährleisten, weil er sein spezielles Know-How in den Makrozell-Generator integrieren kann. Im Allgemeinen wird man bei Speichern eine Kombination finden: Optimierte Zellen für das Speicherfeld und Standardzellen für die Adress-Dekoder-Logik.

Register-Transfer-Synthese: Synthese endlicher Automaten

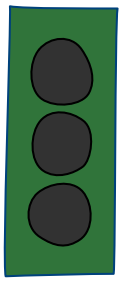


Aufgabe der Synthese endlicher Automaten ist es, Automaten in ein System von booleschen Gleichungen umzuwandeln. Die Anzahl der Zustände sei bereits minimiert und der Automat als Zustandsgraph beschrieben. In diesem Graphen sind die einzelnen Zustände als Knoten und die Zustandsübergänge, die auf Änderungen der Eingangsvariablen folgen, als gerichtete Kanten dargestellt.

Um zu booleschen Gleichungen zu gelangen, werden zunächst die Zustände kodiert, Flipfloptypen ausgewählt und das Übergangs- und das Ausgangsschaltnetz in booleschen Gleichungen beschrieben.

Register-Transfer-Synthese: Zustandskodierung

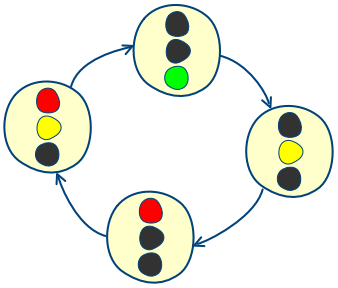
Zustandskodierung



Binär	Gray	One-Hot	"Ampel"
00	00	0001	001 (grün)
01	01	0010	010 (gelb)
10	11	0100	100 (rot)
11	10	1000	110 (rotgelb)
Minimale Anzahl von Zustandsbits	Nur ein Bitwechsel bei Zustandswechsel	Pro Zustand ein FlipFlop	Zustände = Ausgänge

Kleine Automaten:
- Finden der optimalen Kodierung möglich

Große Automaten:
- Heuristiken nötig



Mit der Zustandskodierung wird den Zuständen des Automaten eine binäre Repräsentation zugeordnet. Dadurch können die Zustände durch eine Menge von Flipflops realisiert werden. Ziel der Zustandskodierung ist es, minimale Kosten zu erzielen. Für PLA-Realisierungen soll die Anzahl der Produktterme nach der zweistufigen Logikminimierung minimal sein, für mehrstufige Realisierungen die Anzahl der Literale in der faktorisierten Form.

Für die Kodierung kleiner Automaten gibt es drei Standardverfahren:

- Binäre Kodierung
- Gray-Kodierung
- One-Hot-Kodierung

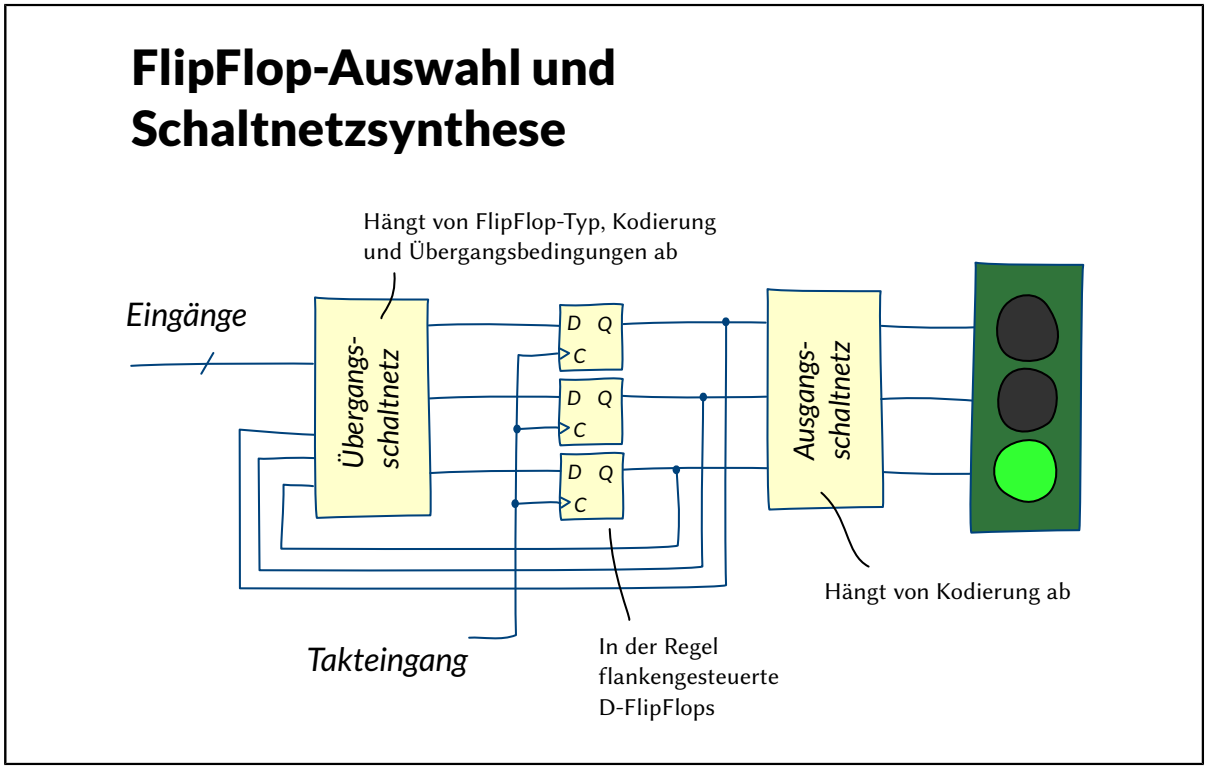
Bei der binären Kodierung werden die Zustände binär durchnummeriert. Mit n Flipflops lassen sich so 2^n Zustände kodieren. Dadurch wird eine minimale Anzahl von Flipflops verwendet, so dass wiederum geringe Kosten erreicht werden können.

Bei der Gray-Kodierung (auch einschrittige Kodierung genannt) werden den Zuständen Binärwerte derart zugeordnet, dass bei einem Zustandswechsel nur ein Flipflop seinen Wert ändert. Dadurch wird für die Setz- und Rücksetzvorgänge der Flipflops nur wenig Hardware benötigt.

Bei der One-Hot-Kodierung wird jeder Zustand durch ein Flipflop repräsentiert. Dadurch wird die Schaltung zunächst teuer, da viele Flipflops verwendet werden, aber man erwartet, dass für die Auswertung der Zustandsübergänge einfachere Schaltnetze ausreichen, da nur einzelne Zustandsbits ausgewertet werden müssen. Letzlich soll der Flächenaufwand stets minimal sein. Diese Kodierung bietet sich für FPGAs an, da viele Flipflops zur Verfügung stehen.

Für sehr wenige Zustände ist es möglich, sämtliche Kodierungen auszuprobieren, die Kosten der Realisierungen zu bestimmen und die günstigste auszuwählen. Für größere Automaten ist dies nicht mehr möglich. Dann werden heuristische Verfahren verwendet, um eine möglichst kostengünstige Kodierung zu finden.

Register-Transfer-Synthese: FlipFlop-Auswahl und Schaltnetzsynthese



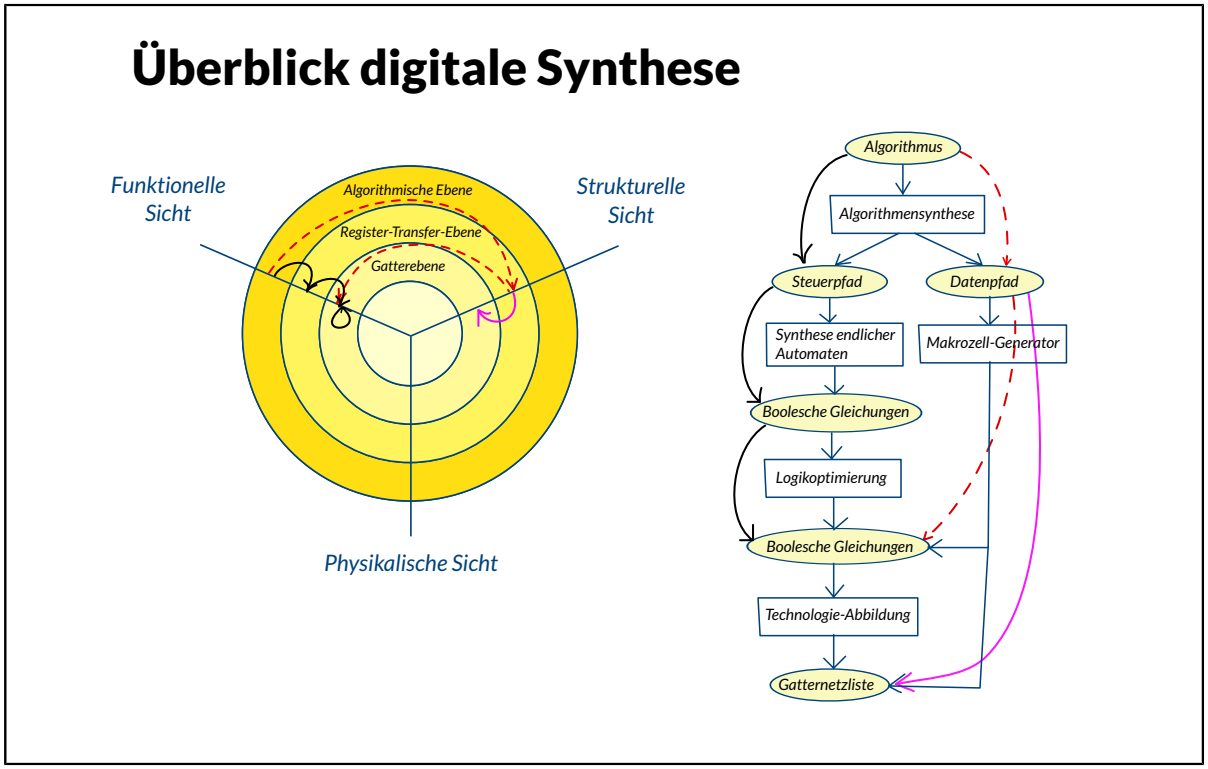
Nach der Zustandskodierung folgt die Auswahl der Flipflops (D-Flipflops sind hierfür am verbreitetsten). Die Steuereingänge der Flipflops bestimmen maßgeblich die aufzustellende Wahrheitstabelle für das Übergangsschaltnetz (Zustandsfolgetabelle). Die Wahrheitsstabelle des Ausgangsschaltnetzes (Ausgangstabelle) ist bereits durch die Zustandskodierung fest definiert. Die booleschen Gleichungen können unmittelbar aus den Tabellen abgelesen werden.

Electronic Design Automation (EDA)

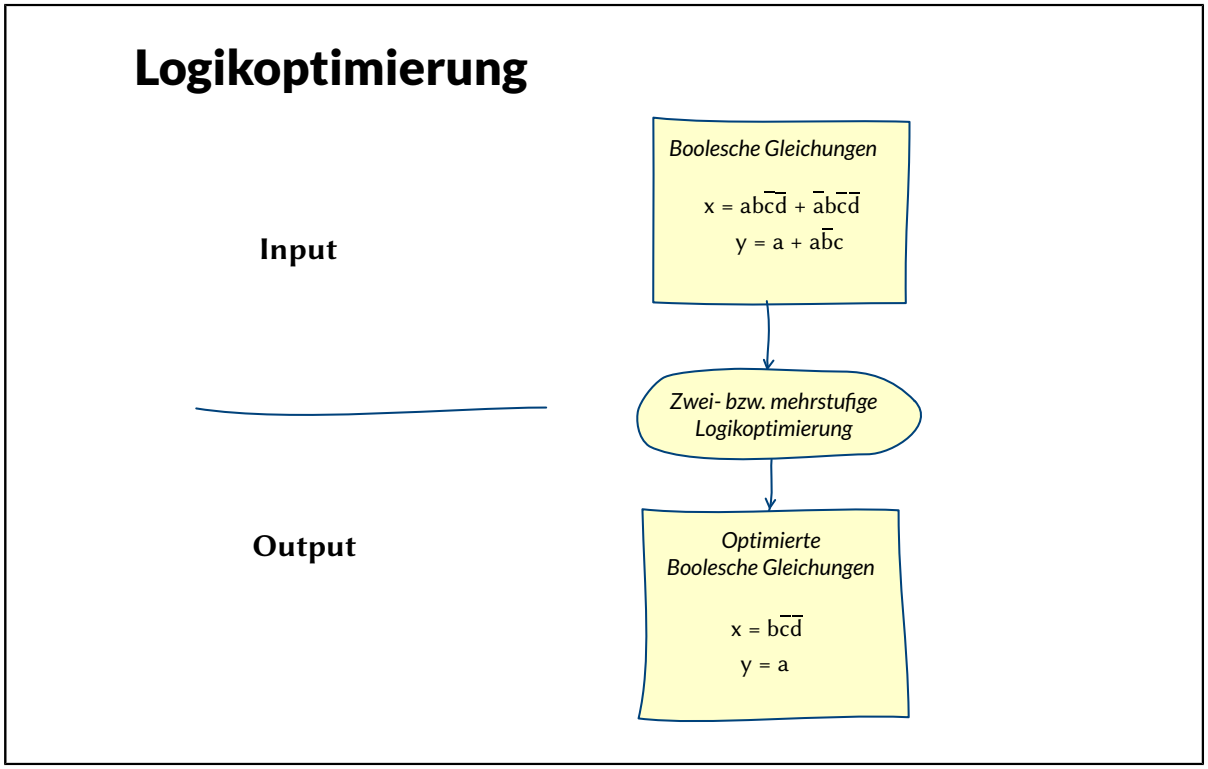
Logikoptimierung

Überblick digitale Synthese
Logikoptimierung
Begriffe
Mehrstufige Logik
Zweistufige Logik: Exakte Verfahren
... Heuristische Verfahren
... Expansion/Reduktion
... Streichen

Logikoptimierung: Überblick digitale Synthese



Logikoptimierung: Logikoptimierung

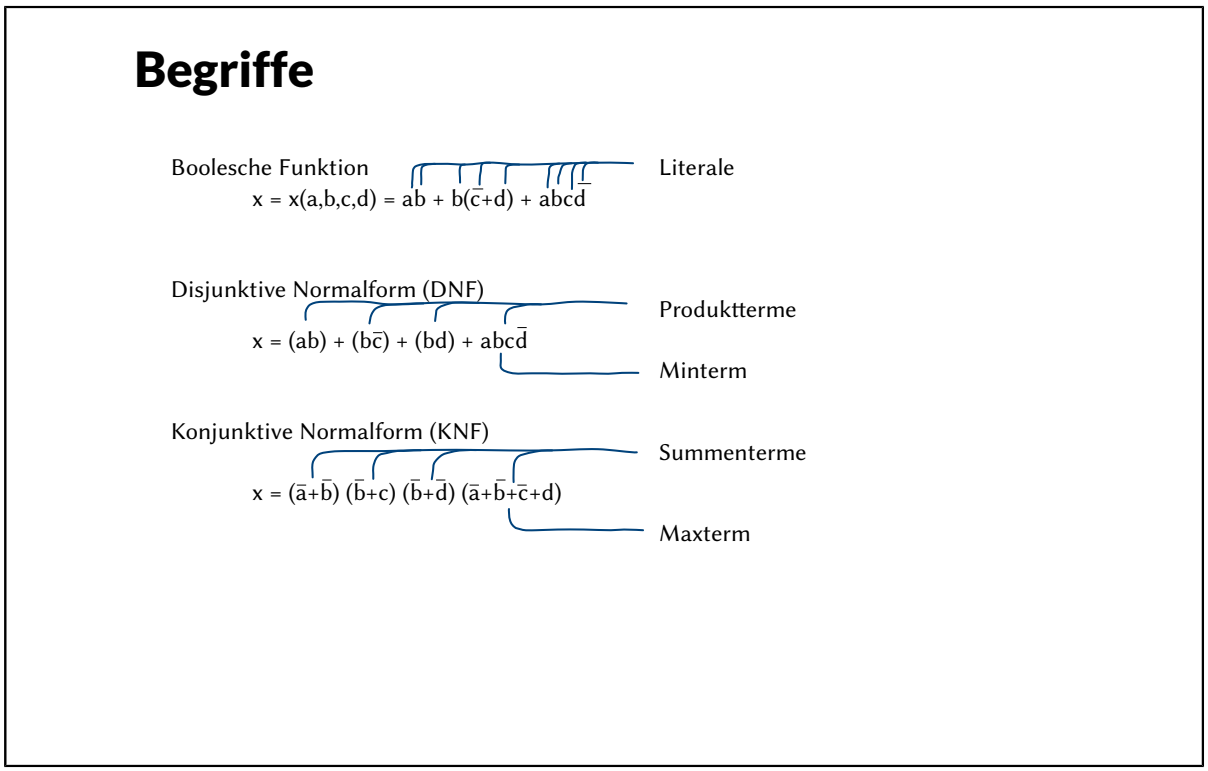


Bei der Logikoptimierung muss je nach Realisierungsform der Schaltung zwischen zwei Verfahren unterschieden werden:

- Mehrstufige Logikoptimierung für zellenbasierte Implementierungen wie Standardzellen, Gate Arrays oder FPGAs.
- Zweistufige Logikminimierung für zweistufige Implementierungen wie PLAs oder PLDs.

Wie stets heißen die Optimierungskriterien Kosten und Performance bzw. Fläche und Verzögerungszeit. Auf der Gatterebene wird die Fläche durch die Anzahl der auftretenden Literale (und damit später über die Anzahl und Größe der benötigten Gatter) und die Verzögerungszeit durch die Anzahl der logischen Stufen repräsentiert. Zweistufige Realisierungen sind deshalb auf dieser Abstraktionsebene die schnellst möglichen.

Logikoptimierung: Begriffe



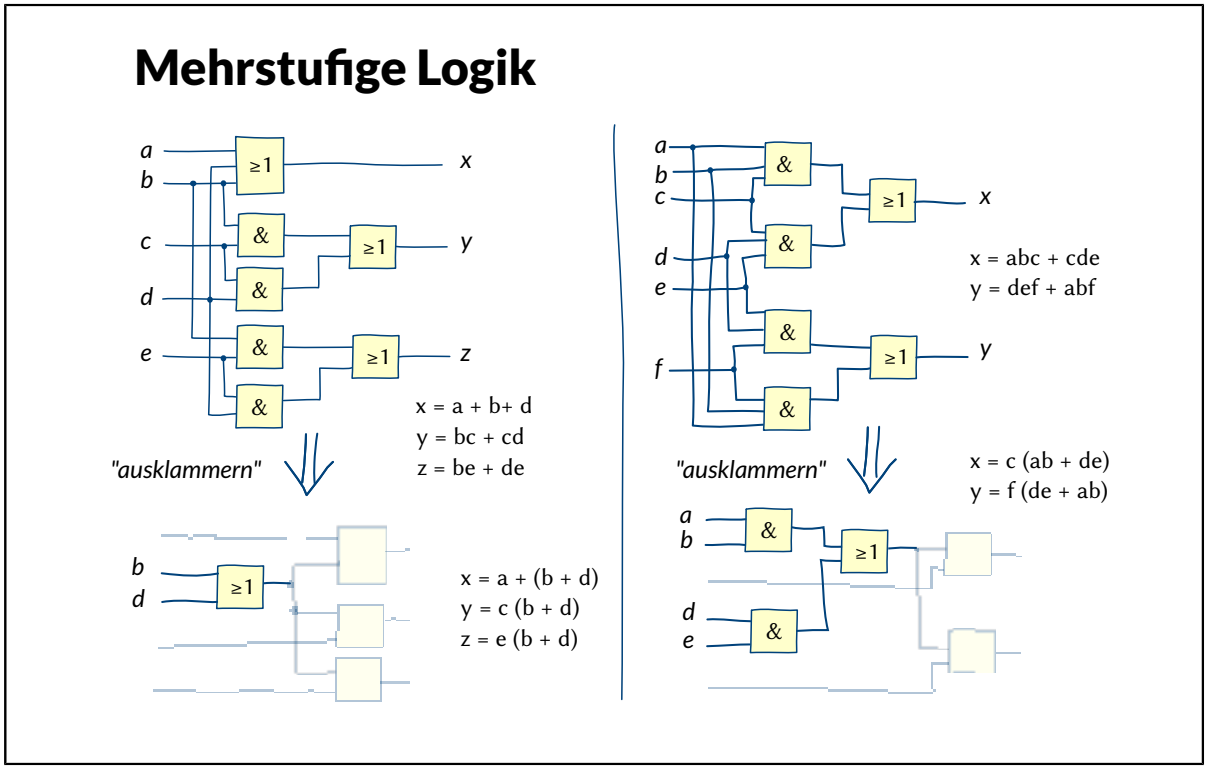
Produktterme bzw. Summenterme, die alle Variablen der booleschen Funktion invertiert oder nicht invertiert enthalten, heißen Minterme bzw. Maxterme. Enthält eine DNF bzw. KNF nur Minterme bzw. Maxterme, heißt sie kanonische DNF bzw. KNF.

Ein Produktterm p heißt Implikant für eine Funktion f, wenn gilt: p=1 => f=1. Ein Implikant heißt Primimplikant, wenn aus dem Produktterm kein Literal entfernt werden kann, ohne dass er seine Eigenschaft Implikant zu sein, verliert.

In der disjunktiven Normalform sind alle Produktterme Implikanten. In einer minimierten DNF sind alle Implikanten Primimplikanten.

Um die Schreibweise zu vereinfachen, wird im Folgenden ähnlich wie bei der Multiplikation das UND-Zeichen in den Gleichungen weggelassen.

Logikoptimierung: Mehrstufige Logik



Bei mehrstufiger Logik wird meist nach den Kosten (Anzahl der Gatter) optimiert, da vor der Technologie-Abbildung für die Performance noch keine genauen Informationen über das zeitliche Verhalten von Gattern vorliegen. Wie bereits erwähnt, wird stattdessen die Anzahl der Logikstufen zwischen Eingang und Ausgang des booleschen Netzwerkes als Maß für die Verzögerung verwendet.

Bei der Optimierung wird das System der booleschen Gleichungen umgeformt, um die Anzahl der Literale zu reduzieren und damit Fläche zu sparen. In erster Linie geht es um die Erkennung von gemeinsamen Unterfunktionen, die nur einmal implementiert werden müssen. Diese Unterfunktionen werden als Zwischenvariablen ("ausklammern") eingeführt. Die Abbildung auf der linken Seite zeigt die Gatterdarstellung folgenden Gleichungssystems vor und nach (unten) dem Einfügen einer Zwischenvariablen (b+d).

x = a+b+d= a+(b+d)

y = bc+cd = c(b+d)

z= be+de = e(b+d)

Dieser Schritt senkt die Kosten, da er Fläche spart (mehrfache Implementierungen werden vermieden), kann aber auch die Anzahl der Stufen erhöhen, wodurch die Schaltung langsamer und damit die Performance schlechter wird. Ein Beispiel dafür zeigt das folgende Gleichungssystem und die rechte Seite der Abbildung.

x = abc+cde = c(ab+de)

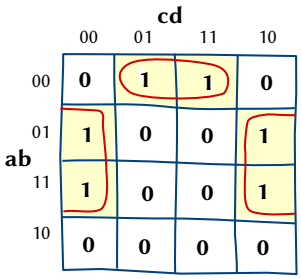
y = def+abf = f(ab+de)

Logikoptimierung: Zweistufige Logik: Exakte Verfahren

Zweistufige Logik: Exakte Verfahren

Exakte graphische Lösung mit Karnaugh Diagrammen

1) Disjunktive Normalform (DNF) graphisch darstellen.



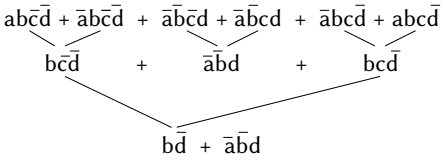
2) "Einsen zusammenfassen" (Absorptionsgesetz)

$\bar{a}\bar{b}d + b\bar{d}$

Exakte rechnerische Lösung mit Quine-McCluskey-Algorithmus

Wesentliche Schritte:
1) Berechnung aller Primimplikanten.
2) Extraktion der minimalen disjunktiven Form der Funktion

Zwei Implikanten lassen sich zusammenfassen, wenn sie sich an nur einer Position unterscheiden (wie beim Karnaugh-Diagramm).



Quine-McCluskey-Algorithmus hat exponentielle Komplexität $O(a_n)$!

Das Ziel ist, eine Summe von Produkttermen zu finden, die die gleiche Ausgangsfunktion beschreibt und möglichst wenig Implikanten und Literale enthält, um die Kosten zu minimieren.

Für die Minimierung zweistufiger Logik gibt es exakte Verfahren und Heuristiken.

Für wenige Eingänge lässt sich eine exakte Lösung mit dem Karnaugh-Diagramm finden. Dies ist eine 2-dimensionale Anordnung einer Funktionstabelle. Jedes Feld enthält genau einen Implikanten (Minterm). Bei benachbarten Feldern ändert sich die dazugehörige Eingangskombination nur bei einer Eingangsvariablen. Dadurch lassen sich benachbarte Felder zu einfacheren Implikanten zusammenfassen. Die Erklärung liefert das Absorptionsgesetz:

$$ab+a\bar{b}=a(b+\bar{b})=a$$

Das exakte Verfahren nach Quine und McCluskey lässt sich leichter in einem Programm implementieren als die Karnaugh-Diagramm-Minimierung. Es garantiert zwar ein Minimum, hat aber eine exponentielle Komplexität (O(n) = (3^n)/n) und ist daher nur für wenige Eingänge praktikabel. Im Wesentlichen basiert es auf zwei Schritten:

- Berechnung aller Primimplikanten: Zwei Implikanten lassen sich zusammenfassen, wenn sie sich an nur einer Position unterscheiden (wie beim Karnaugh-Diagramm).
- Extraktion der minimalen disjunktiven Form der Funktion.

Logikoptimierung: ... Heuristische Verfahren

Zweistufige Logik: Heuristische Verfahren

Bekannte exakte Verfahren sind zu langsam für komplexe Probleme, daher werden heuristische Optimierungsverfahren einsetzt (z.B. Espresso)

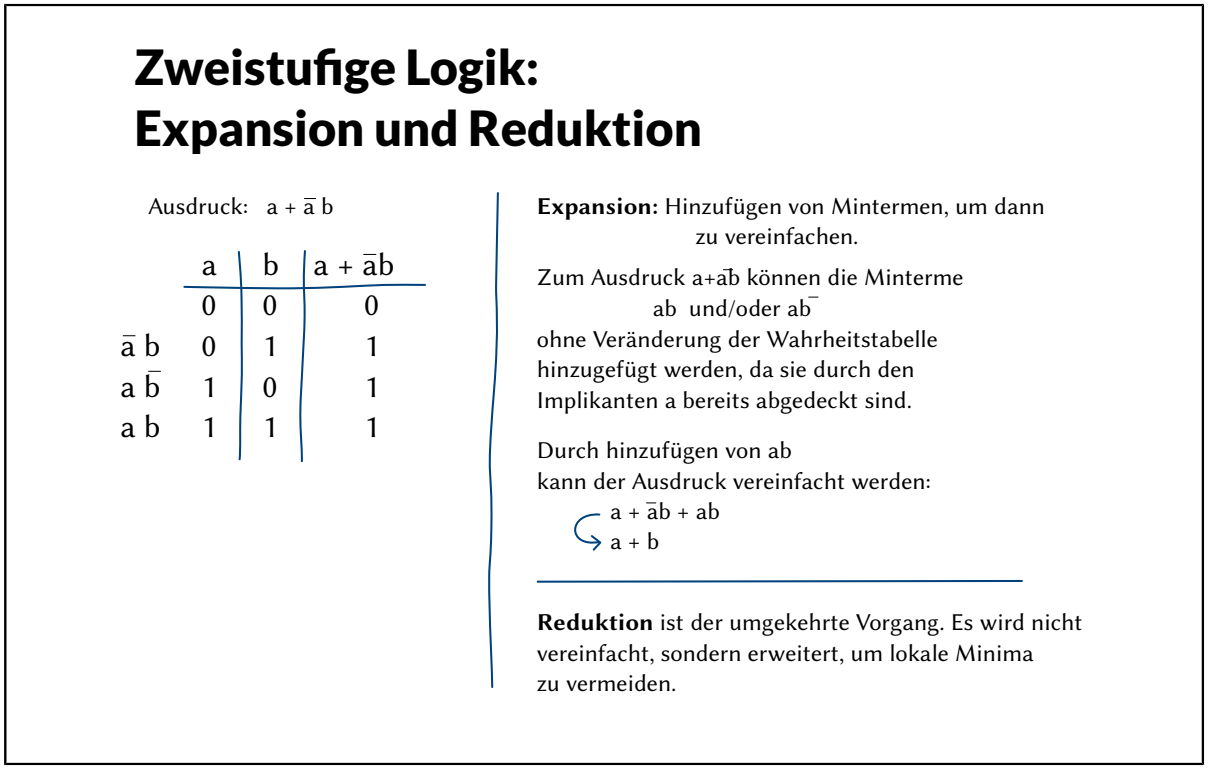
- Betrachtung von Untermengen von Implikanten
- Iteratives Vorgehen
- Rückschritte erlauben, um lokale Minima zu vermeiden

Das Espresso-Verfahren enthält drei elementare Schritte

- Implikanten expandieren
- Implikanten reduzieren
- Implikanten streichen

Aufgrund der Komplexität exakter Verfahren sind heuristische Verfahren gebräuchlich. Diese basieren zwar auf exakten Verfahren, betrachten jedoch nur eine Untermenge der Implikanten und optimieren iterativ. Das bekannteste Verfahren ist in dem Programm Espresso implementiert. Die Iterationen enthalten drei elementare Schritte: Implikanten expandieren, reduzieren oder streichen. Beim Expandieren und Streichen werden Literale entfernt und beim Reduzieren werden Literale hinzugefügt.

Logikoptimierung: ... Expansion/Reduktion



Expandieren

Durch das Expandieren einer Funktion durch weitere Implikanten ohne die Funktion und damit die Wahrheitstabelle zu verändern, kann sich die Möglichkeit ergeben Literale zu entfernen. Zum Beispiel kann durch Expansion folgender Ausdruck minimiert werden:

$a+\bar{a}b$

Die Wahrheitstabelle hierfür ist

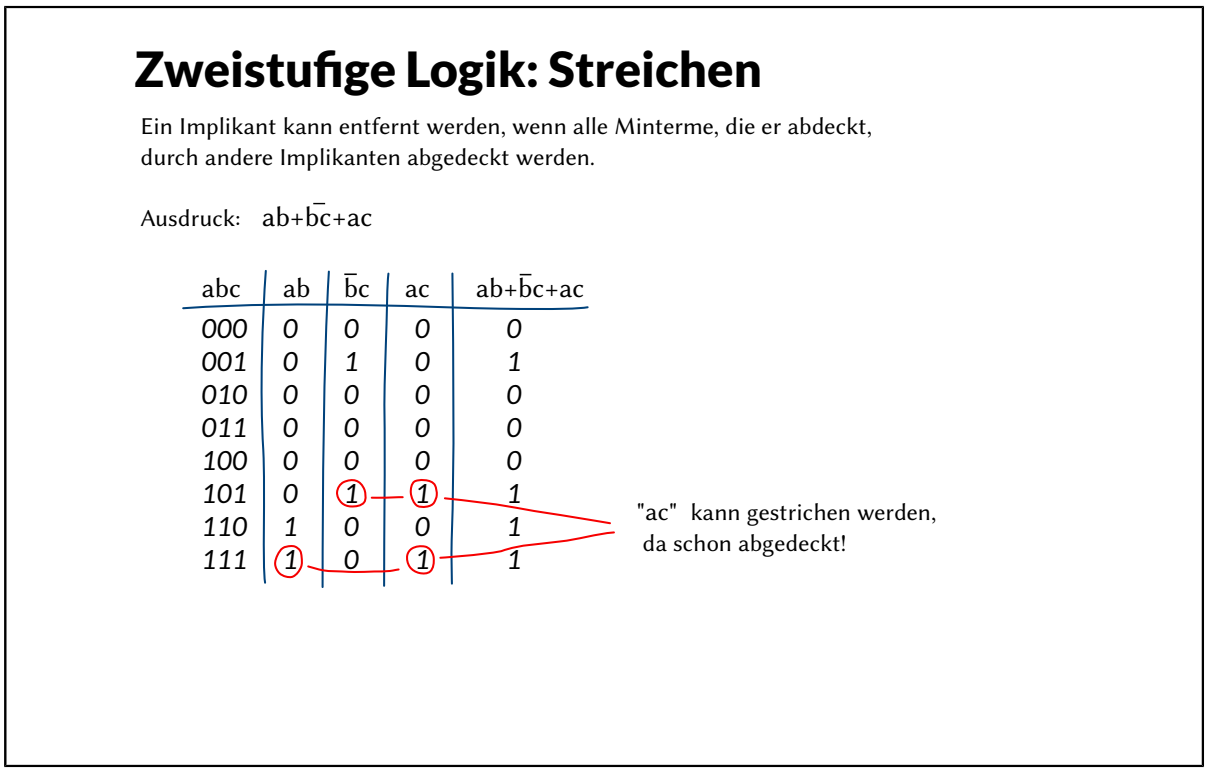
a	b	a+āb
0	0	0
0	1	1
1	0	1
1	1	1

Zu diesem Ausdruck gehören also die Minterme $\bar{a}b$, $a\bar{b}$ und ab. Es können Minterme hinzugefügt werden, die durch vorhandene Implikanten bereits abgedeckt werden, z.B. können in dem obigen Ausdruck Minterme hinzugefügt werden, die durch den Implikanten a abgedeckt werden, wie z.B. ab oder $a\bar{b}$. Wird der Ausdruck mit ab expandiert, vereinfacht sich der Ausdruck wegen $\bar{a}b+ab=b$ zu $a+b$. Beide Ausdrücke stellen die gleiche Funktion dar.

Reduktion

Reduktion dagegen liefert vom Ausdruck $a+b$ den Ausdruck $a+\bar{a}b$. Die Reduktion erhöht zwar die Kosten der Funktion, indem sie Literale hinzufügt, kann aber notwendig sein, um lokale Minima bei der Optimierung verlassen zu können.

Logikoptimierung: ... Streichen



Streichen

Ein Implikant kann entfernt werden, wenn alle Minterme, die er abdeckt, durch andere Implikanten abgedeckt werden. Z. B. kann in dem folgenden Ausdruck der Implikant ac gestrichen werden:

$$ab+\bar{b}c+ac$$

Die Funktionstabelle dieses Ausdrucks verdeutlicht, wieso dies möglich ist:

a b c	ab	$\bar{b}c$	ac	$ab+\bar{b}c+ac$
0 0 0	0	0	0	0
0 0 1	0	1	0	1
0 1 1	0	0	0	0
1 0 0	0	0	0	0
1 0 1	0	1	1	1
1 1 0	1	0	0	1
1 1 1	1	0	1	1

Die Minterme, die durch ac abgedeckt werden ($a\bar{b}c$ und abc), werden bereits durch die Implikanten ab und $\bar{b}c$ abgedeckt.

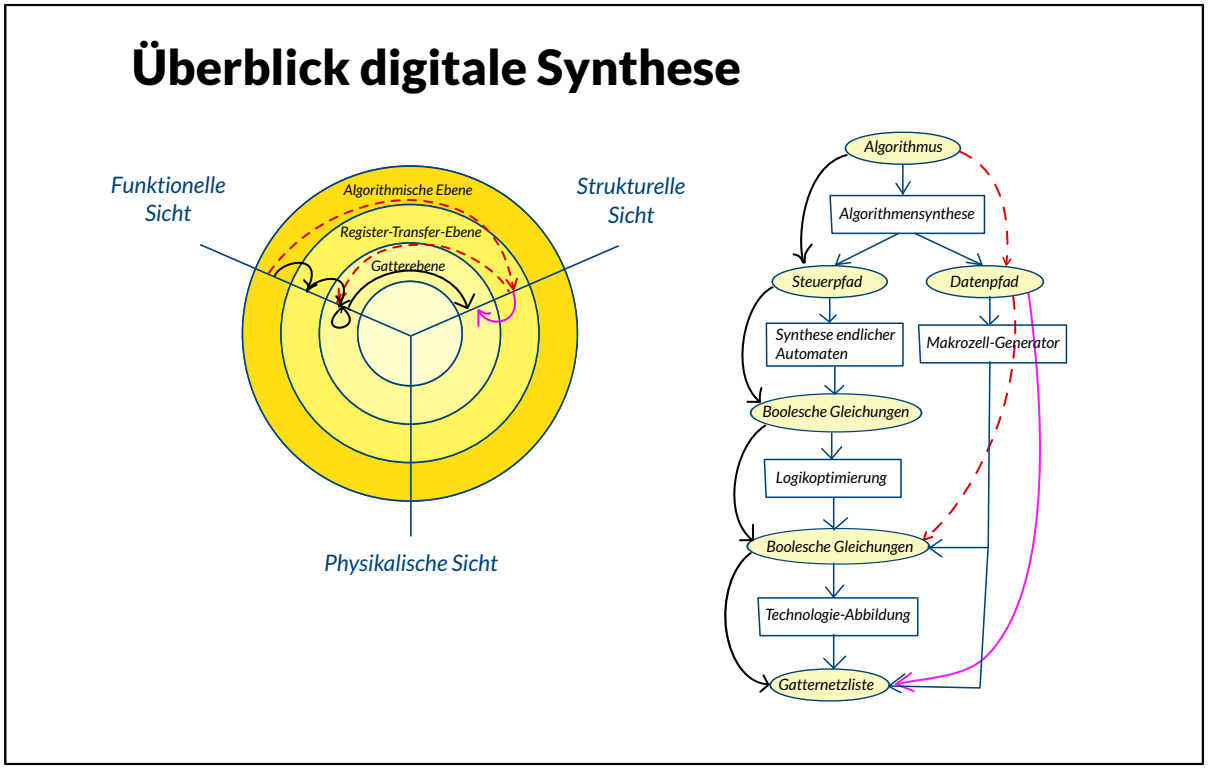
Electronic Design Automation (EDA)

Technology Mapping

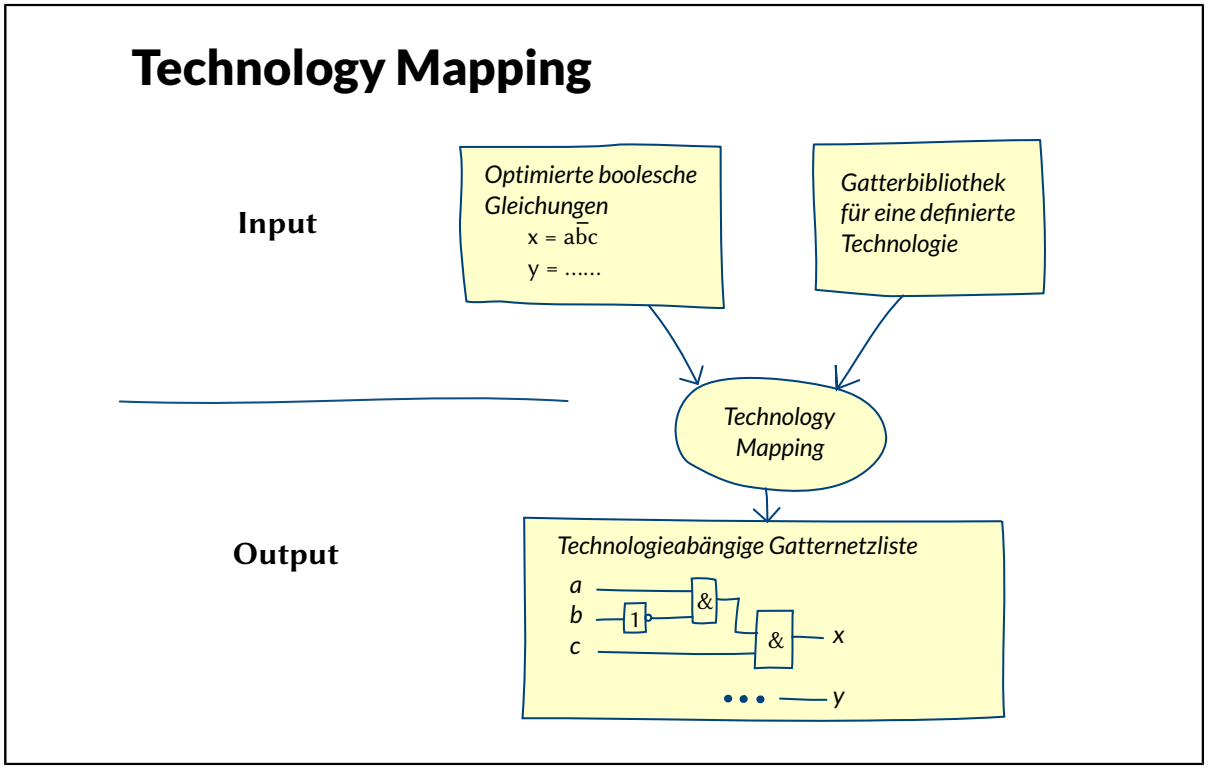
Überblick digitale Synthese
Technology Mapping
Abbildung durch die Abdeckung eines Baumes
Partitionierung des DAG
Dekomposition und Abdeckung
Beispiel

Date of generation: 8.7.2025

Technology Mapping: Überblick digitale Synthese



Technology Mapping: Technology Mapping



Die Technologie-Abbildung bildet den Abschluss der Synthese. Da Flipflops und Treiber in der Regel unmittelbar durch entsprechende Elemente einer Bibliothek ersetzt werden, ist die Hauptaufgabe der Technologie-Abbildung die Optimierung kombinatorischer Schaltungsteile. Ausgangspunkt hierfür sind zwei- oder mehrstufige logische Gleichungen. In diesem Kapitel werden nur Methoden für mehrstufige Darstellungen beschrieben. Zweistufige Darstellungen dienen im Allgemeinen als Eingabe für PLA-Generatoren und werden hier nicht behandelt.

Die booleschen Gleichungen werden mit verfügbaren Zellen einer Bibliothek implementiert, die zu einer bestimmten Technologie gehört. Die Bibliotheken enthalten Gatter (NOR, Inverter,...) und komplexere Elemente wie Komparatoren, Halbaddierer o.ä.. Die Gatter sind als boolesche Funktionen beschrieben und durch ihre Fläche und ihre Verzögerungszeiten charakterisiert. Die Verzögerungszeit setzt sich meist aus einem lastabhängigen und einem lastunabhängigen Anteil zusammen.

Technology Mapping: Abbildung durch die Abdeckung eines Baumes

Abbildung durch Abdeckung eines Baumes

- Primäres Ziel ist Kosten(Flächen)-Optimierung.
- Zur Schaltungsmodellierung wird in der Regel ein gerichteter azyklischer Graph benutzt (Directed Acyclic Graph = DAG).
- Ziel ist es, den DAG der Schaltung durch Subgraphen abzudecken, die die in der Bibliothek implementierten Basisfunktionen darstellen.
- Eine exakte Lösung des Problems ist zu aufwendig, daher Lösung durch regelbasierte oder andere heuristische Verfahren.
- Regelbasierte Verfahren finden optimale Lösungen in überschaubaren Teil-DAGs.
- Dies führt zu einem allgemeinen Baum-Abdeckungsproblem:
  1. Aufbau des DAG der Schaltung
  2. Partitionierung in Bäume
  3. Dekomposition der Bäume in Basisfunktionen
  4. Abdeckung aller Bäume der Schaltung mit Zellen aus der Bibliothek

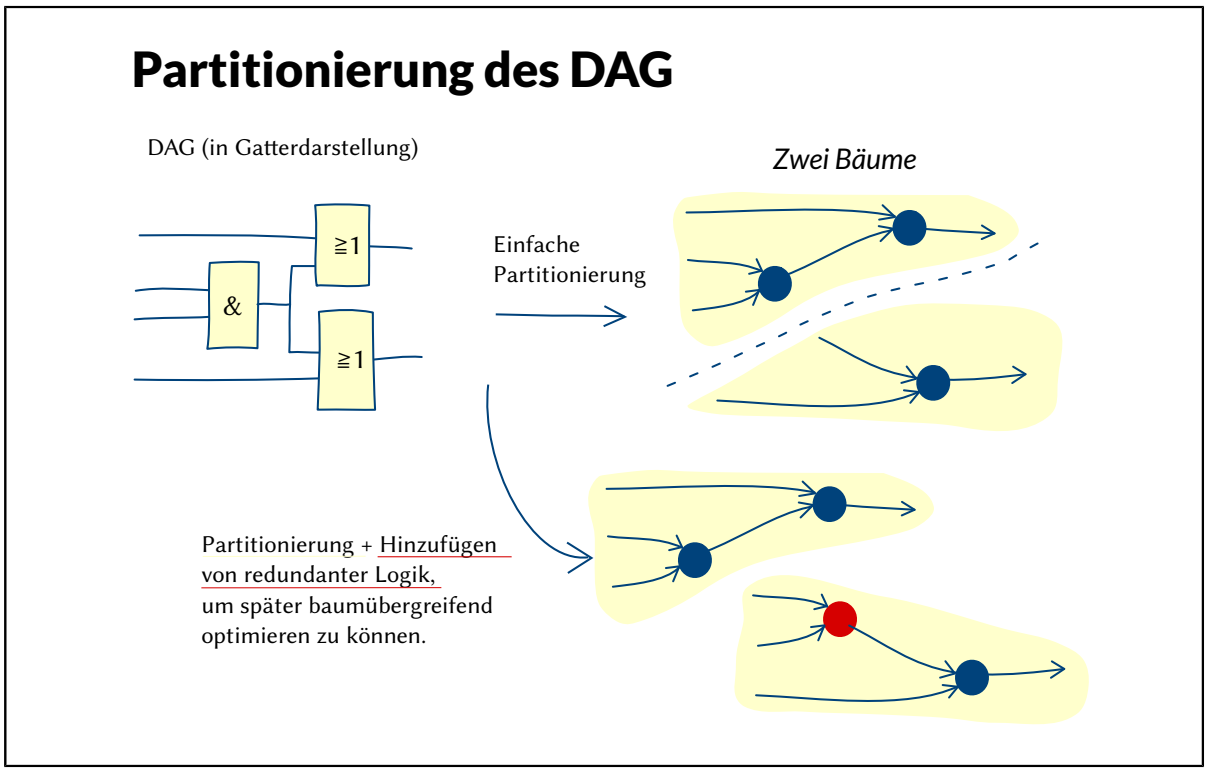
In erster Linie wird eine Implementierung gesucht, deren Kosten möglichst gering sind, indem eine flächenminimale Zellenkombination gesucht wird. Bei zeitkritischen Pfaden können stattdessen auch die schnellsten verfügbaren Zellen ausgewählt werden. Die Berücksichtigung der Performance in Form von Verzögerungszeiten während der Abbildung ist schwierig, da der lastabhängige Teil der Verzögerung erst am Ende der Abbildung bestimmt werden kann.

Häufig verwendete **heuristische** Methoden basieren auf einer Baum-Abdeckung. Diese besteht aus vier Schritten:

- Aufbau des DAG
- Partitionierung des Graphs (der Schaltung) in Bäume
- Dekomposition der Bäume (Schaltungsteile) in Basisfunktionen
- Abdeckung aller Bäume (der Schaltung) mit Zellen aus der Bibliothek

Die Abdeckung eines DAGs ist ein NP-hartes Problem, seine exakte Lösung ist daher zu aufwendig. Dagegen sind Abdeckungsalgorithmen für Bäume praktikabel. Daher wird der DAG in einen Wald von Bäumen zerlegt.

Technology Mapping: Partitionierung des DAG



Ein einfacher Weg, den DAG in Bäume zu zerlegen, besteht darin, alle Verzweigungen aufzutrennen und anschließend alle Bäume einzeln abzudecken. Der Nachteil davon ist, dass beim Abdecken nur lokal innerhalb eines Baums optimiert werden kann und nicht baumübergreifend.

Eine bessere Partitionierung kann erreicht werden, indem an den aufgetrennten Stellen zusätzliche Logik eingefügt wird. Der Pfad von den Eingangsvariablen bis zum aufgetrennten Knoten wird für jede Verzweigung hinzugefügt. Das erzeugt zwar Redundanz, ermöglicht aber eine baumübergreifende Optimierung.

Technology Mapping: Dekomposition und Abdeckung

Dekomposition und Abdeckung

Dekomposition des DAG der Schaltung in eine kanonische Darstellung mit Basisfunktion z.B. NAND, NOR mit zwei Eingängen (Subject Graph).

Bibliothek enthält alle Basisfunktionen als sogenannte Pattern Graphs
-> triviale Abdeckung existiert.

Ziel ist, durch Verwendung komplexer Zellen der Bibliothek (Komplexgatter) eine flächengünstige Lösung zu finden.

Die **Abdeckung** erfolgt in drei Schritten:

- **Matching:** Erstellung einer Liste aller möglichen Matchings für alle Teilbäume (Ersatz von Teilen des Subject Graphs durch isomorphe Pattern Graphs)
- **Optimierung:** Bestimmung des Matchings mit den niedrigsten Kosten für jeden Teilbaum
- **Akkumulation (Mapping):** Top-Down-Abdeckung des Gesamtbaumes ausgehend von der optimalen Zelle für die Wurzel des Baumes

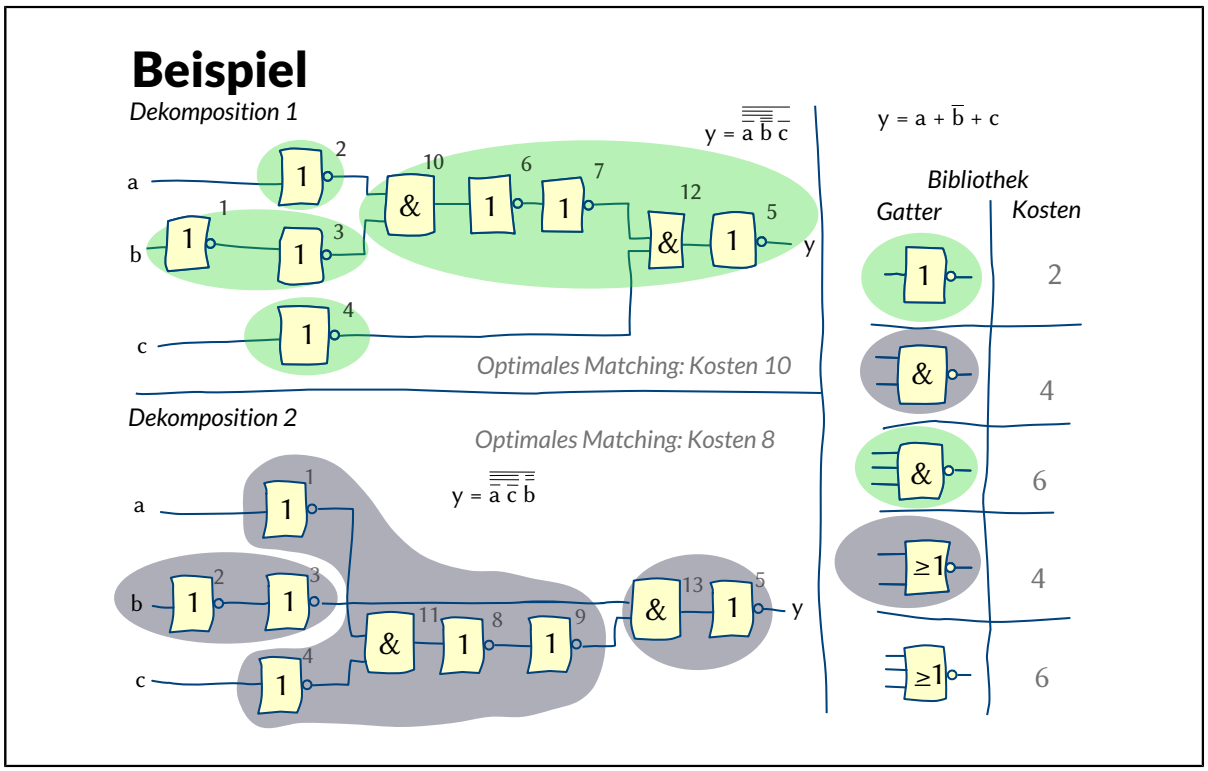
Für die eigentliche Abdeckung erfolgt eine Dekomposition der Schaltung in eine kanonische Darstellung mit Basisfunktionen (z.B. NAND-, NOR-Gatter mit 2 Eingängen und Invertern) für den DAG (Subject Graph genannt) und die Bibliothek (Pattern Graph genannt). Da die Bibliothek in der Regel auch diese Basisfunktionen enthält, gibt es mindestens eine triviale Abdeckung. Ziel ist es jedoch durch Verwendung komplexer Zellen der Bibliothek (Komplexgatter), z.B. Gatter mit mehr als zwei Eingängen, eine flächengünstigere Lösung zu finden.

Der Lösungsraum für die Abdeckung wird durch die Partitionierung und die Dekomposition eingeschränkt. Die Lösung erfolgt in zwei Schritten mit einem dynamischen Algorithmus:

Matching mit **Optimierung** Für alle Teilbäume eines Baums wird eine Liste mit allen möglichen Matchings erstellt. D.h. Teilbäume des Subject Graphs werden durch isomorphe Pattern Graphs ersetzt. Dies ist ein rekursives Bottom-Up-Verfahren, das das optimale Matching für jeden Teilbaum findet. Dabei wird von den Eingängen des Subject Graphs schrittweise zu den Ausgängen vorgegangen. Für jede zugeordnete Zelle eines Knotens des Baums werden die Kosten der Zelle und der optimalen Matchings der Eingänge dieser Zelle berechnet. Am Ende wird das optimale Matching an der Wurzel des Baums bestimmt (Matching mit den niedrigsten Kosten).

Akkumulation Hier werden in einem Top-Down-Verfahren die Ergebnisse der Optimierung genutzt, um eine Abdeckung des gesamten Baums zu erzielen. Ausgehend von der optimalen Zelle der Wurzel des Baumes, werden rekursiv die Teilbäume der Eingänge der zugeordneten Zelle zugeordnet.

Technology Mapping: Beispiel



Das Bild zeigt zwei Dekompositionen einer Schaltung (eines Baumes) in AND-Gatter mit zwei Eingängen und Inverter. Als einfaches Beispiel für eine Abdeckung wird die obere Dekomposition betrachtet. Die Bibliothek beinhaltet NAND- und NOR-Zellen mit bis zu drei Eingängen und Inverter. Die Kosten werden vereinfacht durch die Anzahl der Transistoren der Zelle bestimmt und setzen sich für die einzelnen Matchings zusammen aus den Kosten der zugeordneten Zelle und den Kosten für die optimalen Matchings der Pfade bis zu den Eingängen dieser Zelle.

1. Matching

Zuerst werden die Gatter an den Eingängen betrachtet. Für die Gatter 1, 2 und 4 gibt es jeweils nur ein Matching, nämlich einen Inverter, der hier die Kosten zwei hat.

Für Gatter 3 gibt es zwei Matchings: Zum einen Weglassen der beiden Inverter 1 und 3 mit den Kosten null, oder einen Inverter. Die Kosten für das zweite Matching setzen sich zusammen aus der zugeordneten Zelle, einem Inverter mit den Kosten zwei, und dem Pfad zum Eingang dieser Zelle. In dem Beispiel besteht dieser Pfad aus dem Gatter 1, für das gerade die Kosten zwei ermittelt wurden. Die Kosten insgesamt betragen also vier.

Für Gatter 10 gibt es nur ein Matching und zwar eine 2-fach NOR-Zelle, die Gatter 2, 3 und 10 abdeckt (not(x1+x2)=(not x1)(not x2)). Die Kosten davon sind vier für die NOR-Zelle und wiederum zwei für den Eingang von Gatter 3, d.h. Gatter 1. Insgesamt hat dieses Matching die Kosten sechs.

Für Gatter 6 gibt es zwei Matchings: Das eine ist ein Inverter mit den Kosten acht (zwei für den Inverter und sechs für dessen Eingang, das Matching für Gatter 10) und das andere ein 2-fach NAND, das die Gatter 10 und 6 abdeckt. Letzteres hat insgesamt die Kosten sechs (vier für die NAND-Gatter plus zwei für Gatter 2 plus Null für die sich aufhebenden Gatter 1 und 3). Da das erste teurer ist, wird es verworfen. So werden alle weiteren Gatter bis zum Ausgang bearbeitet.

2. Akkumulation

Danach werden ausgehend vom optimalen Matching des letzten Gatters die optimalen Matchings für dessen Eingänge bestimmt, und danach die Matchings für dessen Eingänge usw.. Für das Beispiel

wird das Ausgangsgatter am besten durch ein 3-fach NAND abgedeckt, das die Gatter 5, 6, 7, 10 und 12 beinhaltet. Diese Zelle wird als Teillösung gewählt. Danach müssen die optimalen Matchings für die Eingänge dieser Zelle der Lösung hinzugefügt werden. In diesem Falle sind dies die optimalen Matchings für die Gatter 2, 3 und 4 und das sind zwei Inverter für Gatter 2 und 4. Die Kosten der Lösung sind insgesamt zehn.

Dieser Algorithmus liefert für die zweite Dekomposition eine andere Lösung, nämlich ein 2-fach NOR, das die Gatter 1, 2, 3, 4, 8, 9 und 11 abdeckt, und ein 2-fach NAND für den restlichen Teil der Schaltung. Diese Lösung hat die Kosten acht und ist damit billiger als die erste Lösung.

Nachteile dieses Verfahrens sind, dass das Ergebnis von der Dekomposition abhängt und dass einige Gatter sich nicht als Baum darstellen lassen (wie z.B. XOR). Es gibt einige Erweiterungen dieses Verfahrens, die diese Nachteile vermeiden, dafür aber aufwendiger sind oder den Lösungsraum weiter einschränken.

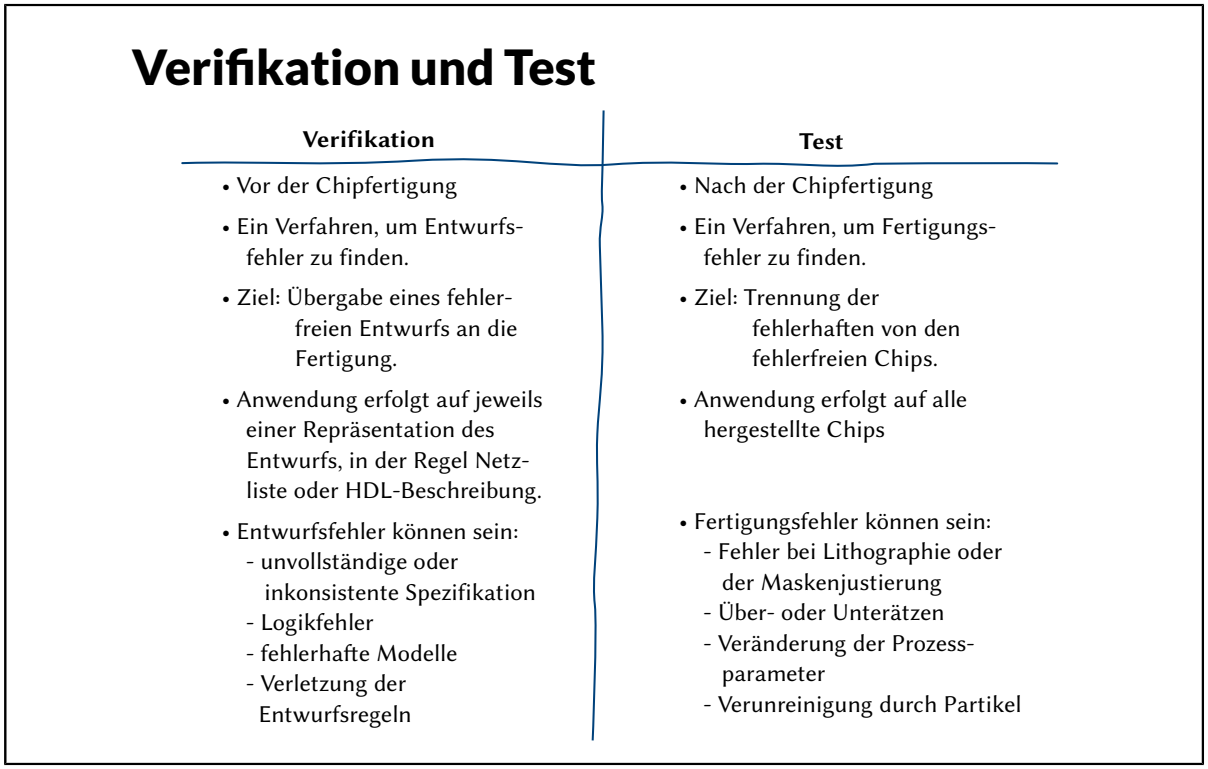
Electronic Design Automation (EDA)

Verification

Verifikation und Test
Der Sandy-Bridge-Bug
Das Verifikationsproblem
Verifikationswerkzeuge
Verifikationsstrategie: Beispiel

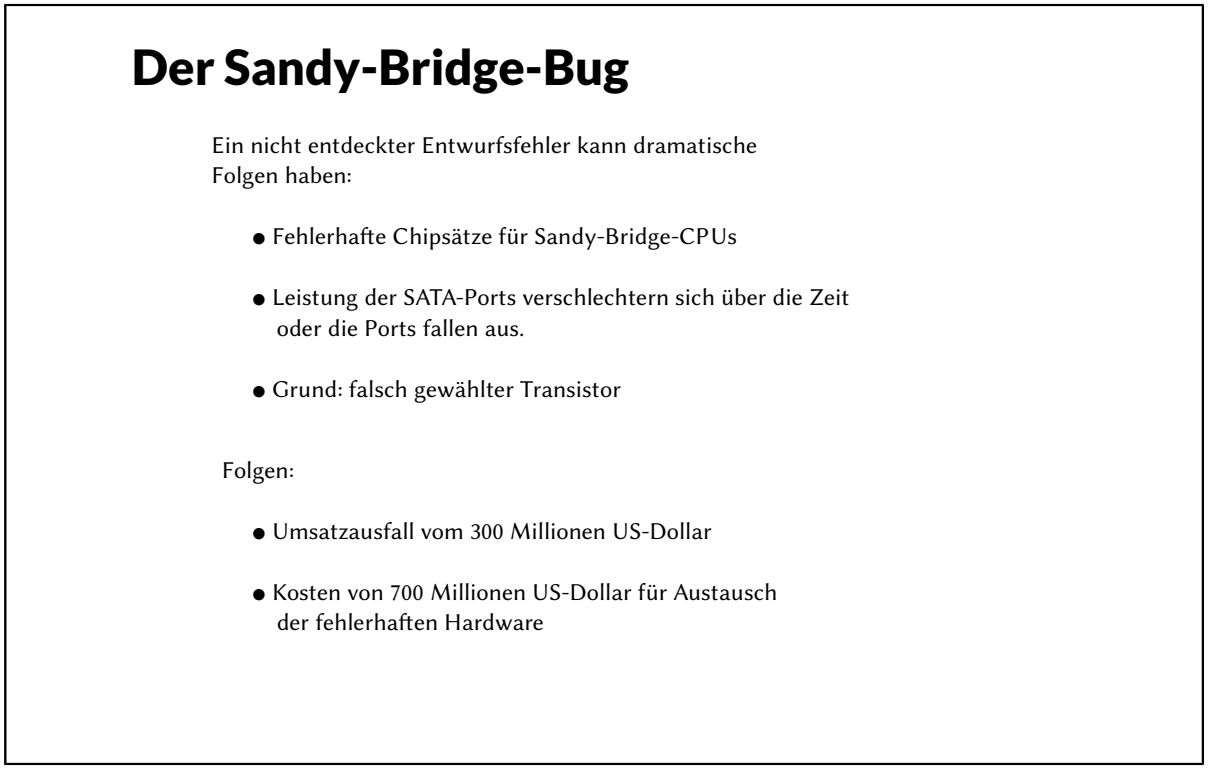
Date of generation: 8.7.2025

Verification: Verifikation und Test



Ein großer Teil des Aufwands beim Entwurf einer integrierten Schaltung beinhaltet die Analyse der Entwurfsergebnisse zur Überprüfung auf Einhaltung der Spezifikation. Je früher im Entwurfsprozess ein Entwurfsfehler entdeckt wird, desto leichter lässt er sich beheben. Die Überprüfung der Entwurfsschritte zu diesem Zweck wird als Verifikation bezeichnet. Nach der Fertigung des Chips muss die Spezifikation erneut überprüft werden, um festzustellen, ob möglicherweise während des Fertigungsprozesses defekte Chips entstanden sind. Verifiziert wird, um Entwurfsfehler zu beheben. Getestet wird, um defekte Chips auszusortieren.

Verification: Der Sandy-Bridge-Bug



Die Bedeutung von Verifikation und Test für eine erfolgreiche Produktentwicklung lässt sich an dem Beispiel des "Sandy-Bridge-Bugs" und seinen dramatischen Folgen belegen. Entwurfsfehler lassen sich nicht vollständig vermeiden. Ein Fehler im Design des Chipsatzes für die Sandy-Bridge-Prozessors ist übersehen worden.

Ein Fehler in einem der Metal-Layer führt zu einem folgenschweren Problem: Die SATA-Ports können mit der Zeit an Leistung verlieren oder ausfallen.

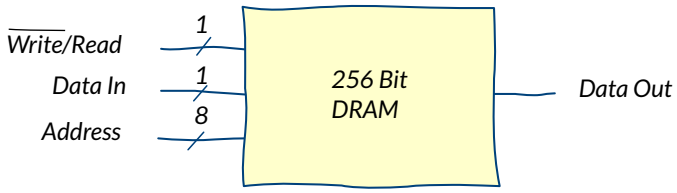
Der Fehler ist erst nach der Herstellung von mind. 100.000 Sandy-Bridge-Systemen bei externen PC-Herstellern erkannt worden.

Der Firma Intel blieb nichts anderes übrig als die fehlerhaften Chips umzutauschen. Die Markteinführung der Dual-Core-Prozessoren der Sandy-Bridge-Reihe wurde dadurch verzögert und die PC-Hersteller mussten in bereits produzierten Systemen die fehlerhaften Chipsätze ersetzen. Man geht von einem Umsatzausfall von ca. 300 Millonen US$ und weiteren 700 Millionen US$ Kosten für den Austausch der fehlerhaften Hardware aus.

Durch eine gründlichere Verifikation des Chips vor der Fertigung oder durch formale Verifikation hätte dieser Fehler vermieden werden können.

Verification: Das Verifikationsproblem

Das Verifikationsproblem



Angenommen,

- wir hätten alle Materie des Mondes zur Verfügung ($7*10^{22}$ kg), der Mond bestände nur aus Protonen und Elektronen
- jeder Computer bestünde aus einem Proton ($1,6*10^{-27}$ kg) und einem Elektronen,
- jeder Computer simulierte 10^{12} Eingangsbelegungen pro Sekunde,
- wir hätten die Simulation zum Zeitpunkt des Urknalls gestartet ($4,4*10^{17}$s),

dann
hätten wir bis heute etwa 17,5 % aller möglichen Eingangskombinationen und Zustände der Schaltung getestet (ca. 10^{80}).

Eine einfache Überlegung zeigt, dass eine vollständige Verifikation durch ein Ausprobieren aller möglichen Konstellationen - hier vollständige Simulation genannt - auch für kleine Schaltungen nicht möglich ist.

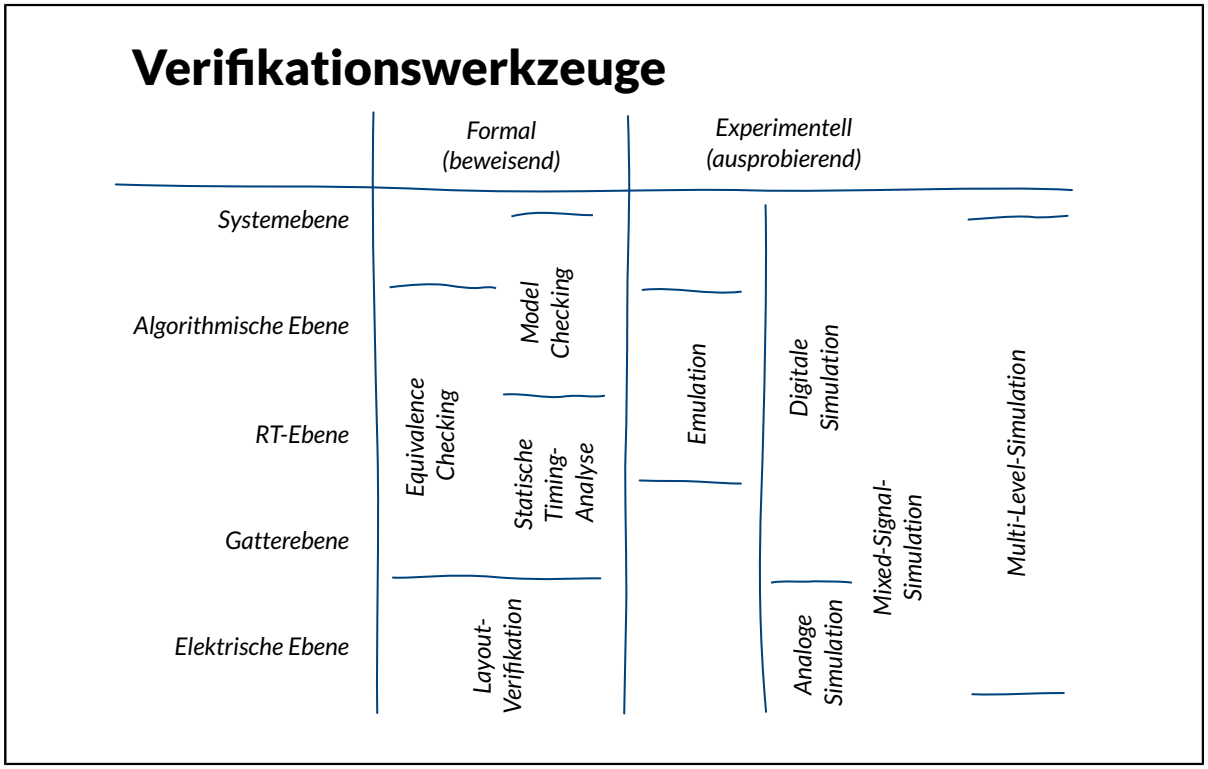
Hier sei ein einfaches 256 Bit-DRAM betrachtet. Ein solcher Speicher hat 2^256 verschiedene interne Zustände und zehn Eingangssignale (Schreib-/ Lesesteuerung, Dateneingang und eine acht Bit breite Adresse), d.h. es gibt 2^10 mögliche Eingangskombinationen. Insgesamt gibt es also 2^(256+10) = 2^266 ~ 10^80 zu untersuchende Konstellationen.

Angenommen

- wir hätten alle Materie des Mondes zur Verfügung, der der Einfachheit halber nur aus Protonen und Elektronen bestände,
- jeder Computer bestünde aus einem Protonen und im Mittel einem Elektron (sonst würde der Computer sich sehr negativ aufladen),
- jeder Computer simulierte 10^12 Eingangsbelegungen pro Sekunde,
- wir hätten die Simulation zum Zeitpunkt des Urknalls gestartet (14 Millarden Jahre),

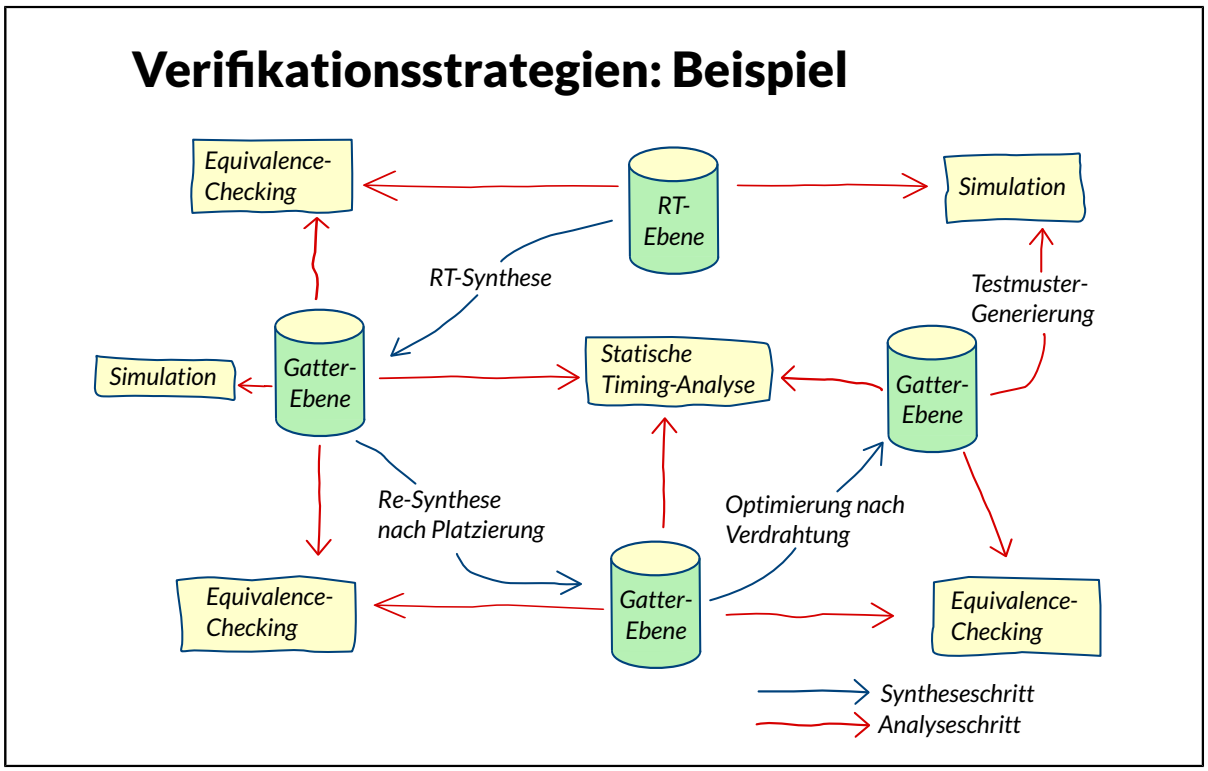
dann hätten wir bis heute etwa 17,5% aller möglichen Eingangskombinationen und Zustände der Schaltung getestet.

Verification: Verifikationswerkzeuge



Für die Entwurfsverifikation existiert auf den verschiedenen Entwurfsebenen eine Vielzahl unterschiedlicher Werkzeuge, die hier im Bild dargestellt sind und in den folgenden Abschnitten besprochen werden sollen. Einige davon haben einen beweisenden Charakter, d.h. die Korrektheit des Entwurfs wird zweifelsfrei festgestellt. Streng genommen dürfen nur diese Werkzeuge Verifikationswerkzeuge genannt werden. Andere dagegen, wie vor allem die Simulation, haben nur ausprobierenden Charakter. Da ein vollständiger Beweis mit diesen Werkzeugen nicht möglich ist, spricht man hier häufig auch von Validierungswerkzeugen.

Verification: Verifikationsstrategie: Beispiel



Verifikationswerkzeuge werden im Entwurfsablauf sehr häufig und auf allen Ebenen eingesetzt. Abschätzungen besagen, dass bei komplexen Schaltungen etwa 70 % der gesamten Entwurfszeit für die Verifikation aufgewendet werden müssen.

Das Bild zeigt beispielhaft wie bei der Bearbeitung einer Schaltung auf der RT- und der Gatterebene verschiedene Verifikationswerkzeuge zum Einsatz kommen.

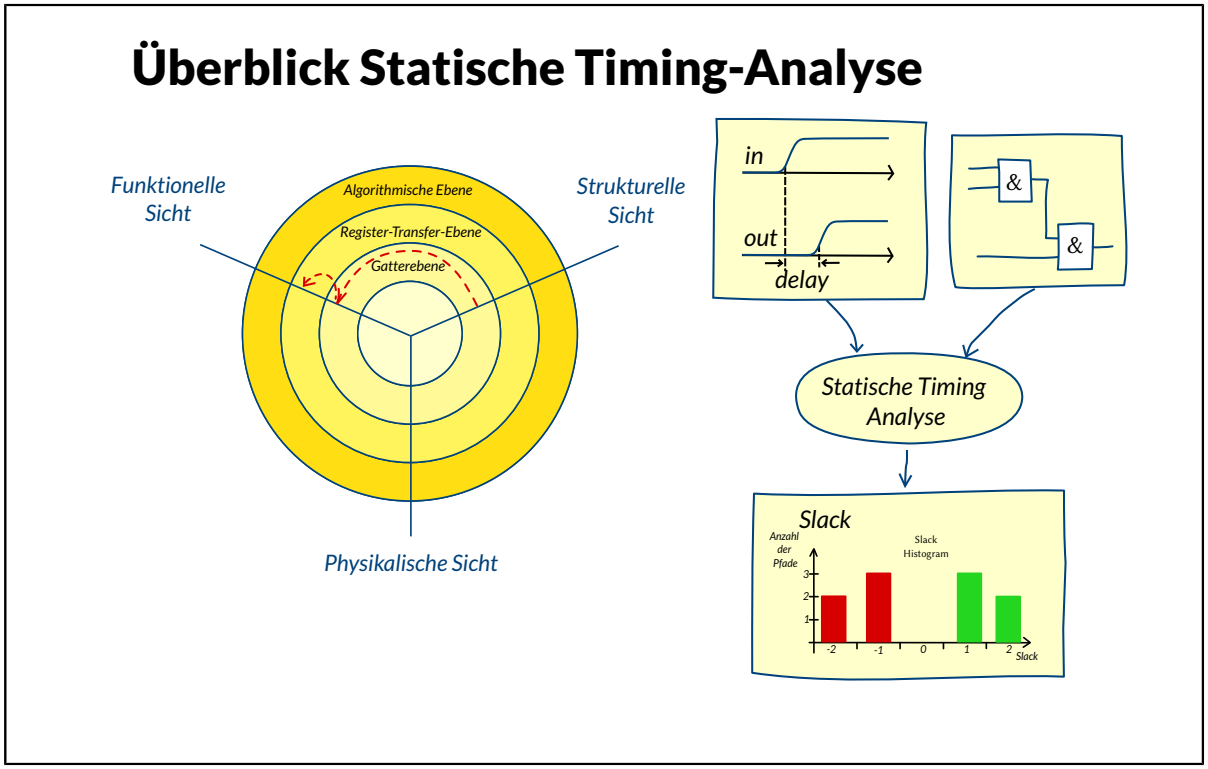
Electronic Design Automation (EDA)

Statische Timing-Analyse

Überblick
Delay
Elmore-Delay
Wire-Load-Modell
Pfad-Problem
Pfade/Cones
Kritischer Pfad
Setup- und Hold-Zeit
Ein- und Ausgänge
Falsche Pfade
Slack

Date of generation: 7.10.2026

Statische Timing-Analyse: Überblick



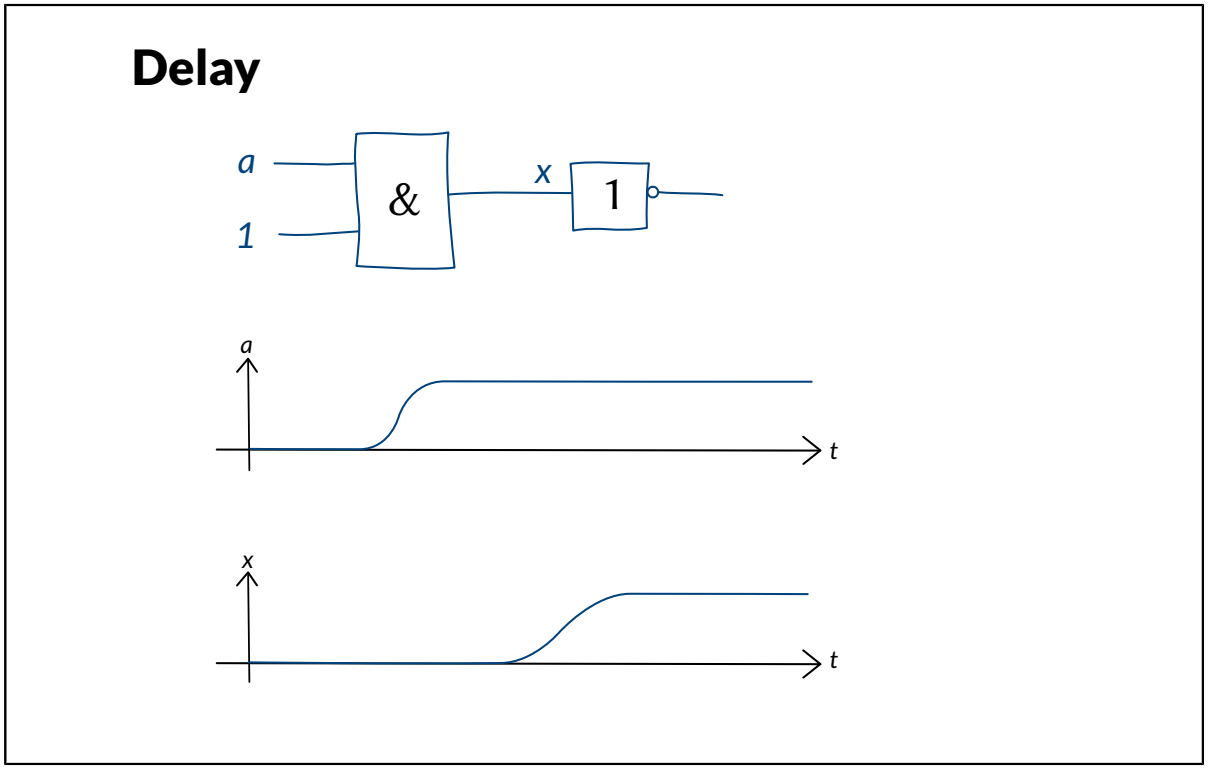
Input:

- Netzliste
- Delaymodell sowie -parameter aller Verbindungen
- Verzögerungszeiten der Elemente
- Taktschema, -perioden
- input arrival time, input slew rate, output capacitance load
- Einzuhaltende Bedingungen:
- Setup-, Hold-Time-Bedingungen der Speicherelemente
- output required arrival time

Output:

- Aussage, ob Bedingungen eingehalten werden
- Kritischer Pfad
- Path-Slack-Histogramm

Statische Timing-Analyse: Delay



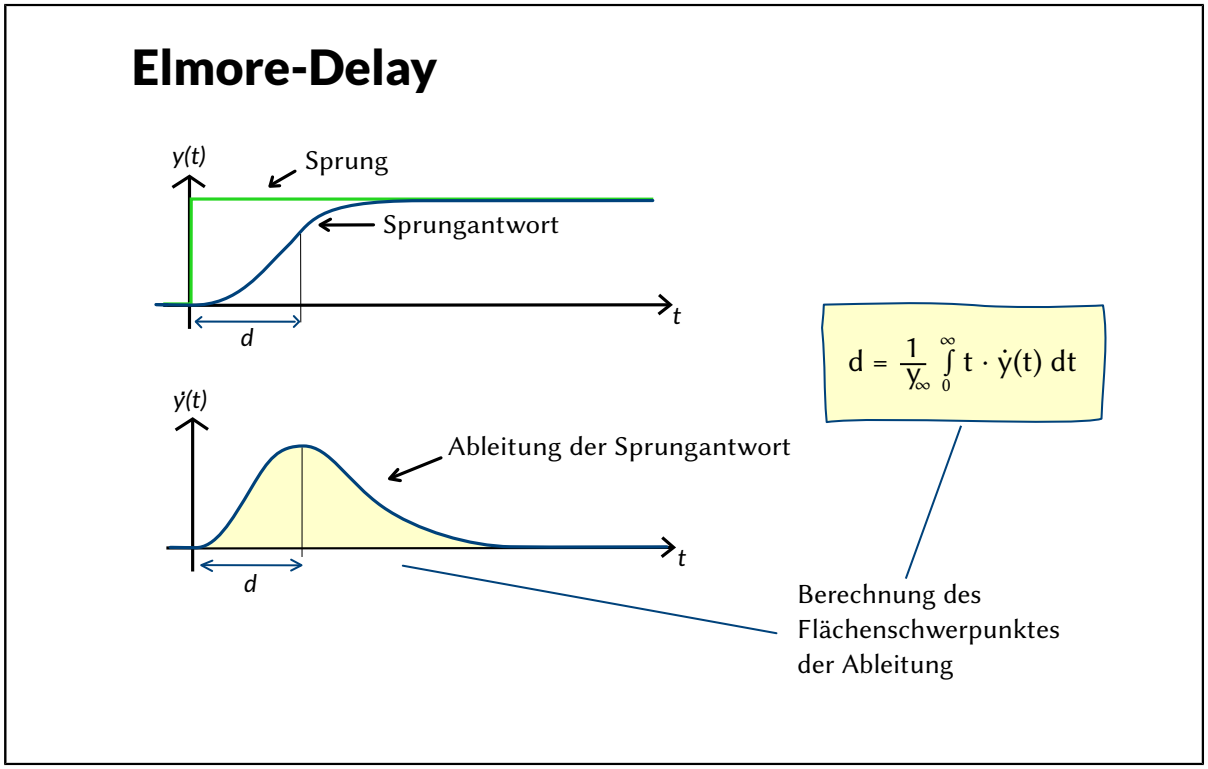
Schaltungen zur Realisierung von logischen Gatterfunktionen verhalten sich, wie bereits in vorigen Kapiteln erwähnt, nicht ideal. Die Realität ist durch folgende wesentliche Schaltungseigenschaften gekennzeichnet: Die Werte der Spannungen und Ströme sowie die Zeit sind kontinuierliche Größen. Außerdem sind die wirkenden elektrischen und magnetischen Felder über der Zeit stetig. Beides führt dazu, dass Ströme und Spannungen ebenfalls stetig sind, also jede Wertänderung Zeit benötigt.

Die Grafik zeigt eine Schaltung mit einem AND-Gatter. Die steigende Flanke am Eingang x bewirkt eine zeitlich verzögerte Flanke am Ausgang a. Allgemein bezeichnet das Delay die Verzögerung eines Signals zwischen einem Ein- und einem Ausgang eines Systems.

Ein Beispiel ist das so genannte 50%-Delay. Es ist definiert als die Zeitdifferenz zwischen den 50%-Schwellen der Signalübergänge beschreibt.

Betrachtet man das Ausgangssignal genauer, stellt man fest, dass es nicht einfach durch eine Verschiebung um den Wert des Delays aus dem Eingangssignal hervorgegangen ist. Zusätzlich hat sich auch die Kurvenform verändert. Es ist daher eine genauere Definition des Delays nötig.

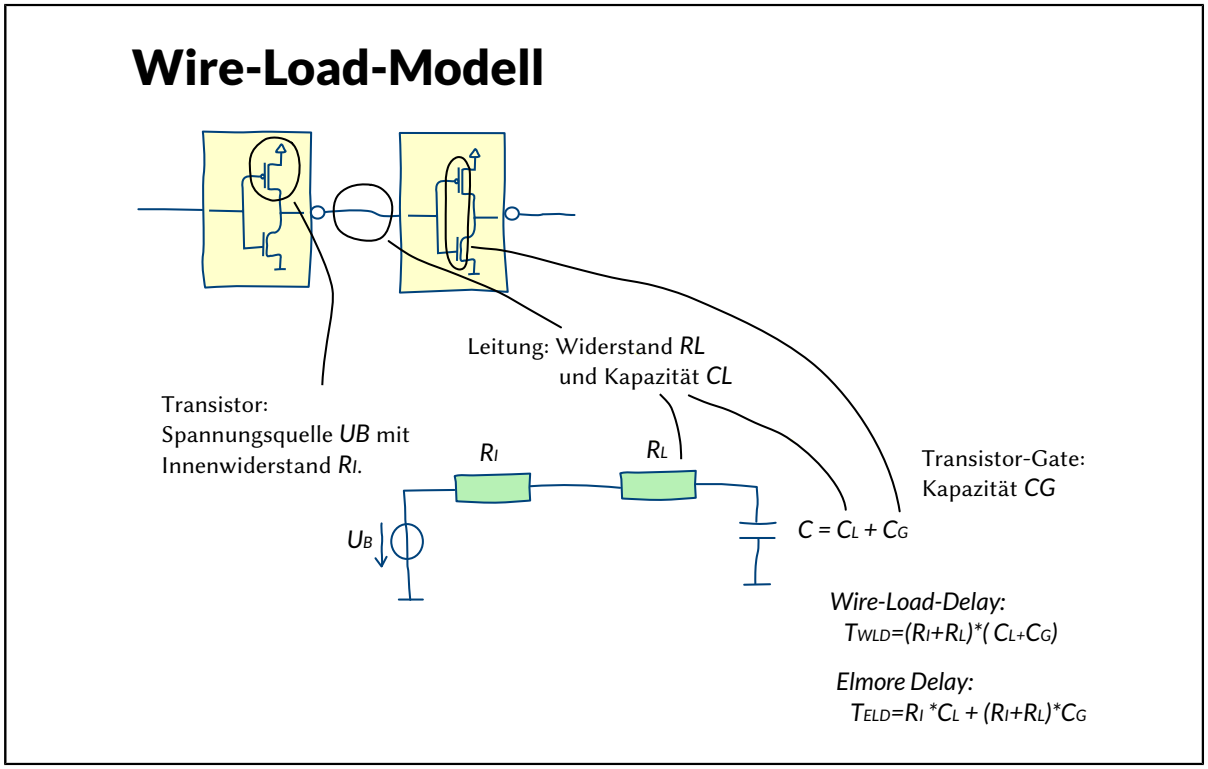
Statische Timing-Analyse: Elmore-Delay



Ein weiteres Beispiel für eine genauere Delay-Definition ist das so genannte Elmore-Delay. Es ist in folgender Weise definiert: Das System wird am Eingang mit einem Einheitssprung erregt. Die Ausgangskurve y(t) wird nach der Zeit abgeleitet. Der Flächenschwerpunkt dieser Ableitung ist das Elmore-Delay (Formel).

Wird das reale System durch ein lineares Modell beschrieben, so kann das Elmore-Delay in folgender Weise interpretiert werden: h(t)=dy(t)/dt sei die Impulsantwort des linearen Systems und H(s) seine Systemübertragungsfunktion. H(s) kann als Potenzreihenentwicklung von s dargestellt werden (Taylor-Reihe). Dabei werden die Koeffizienten als Momente i-ter Ordnung bezeichnet. Das Elmore-Delay ist identisch mit dem Moment erster Ordnung. Seine Berechnung ist für Netzwerke aus R und C direkt durch Summen- und Produktbildung möglich.

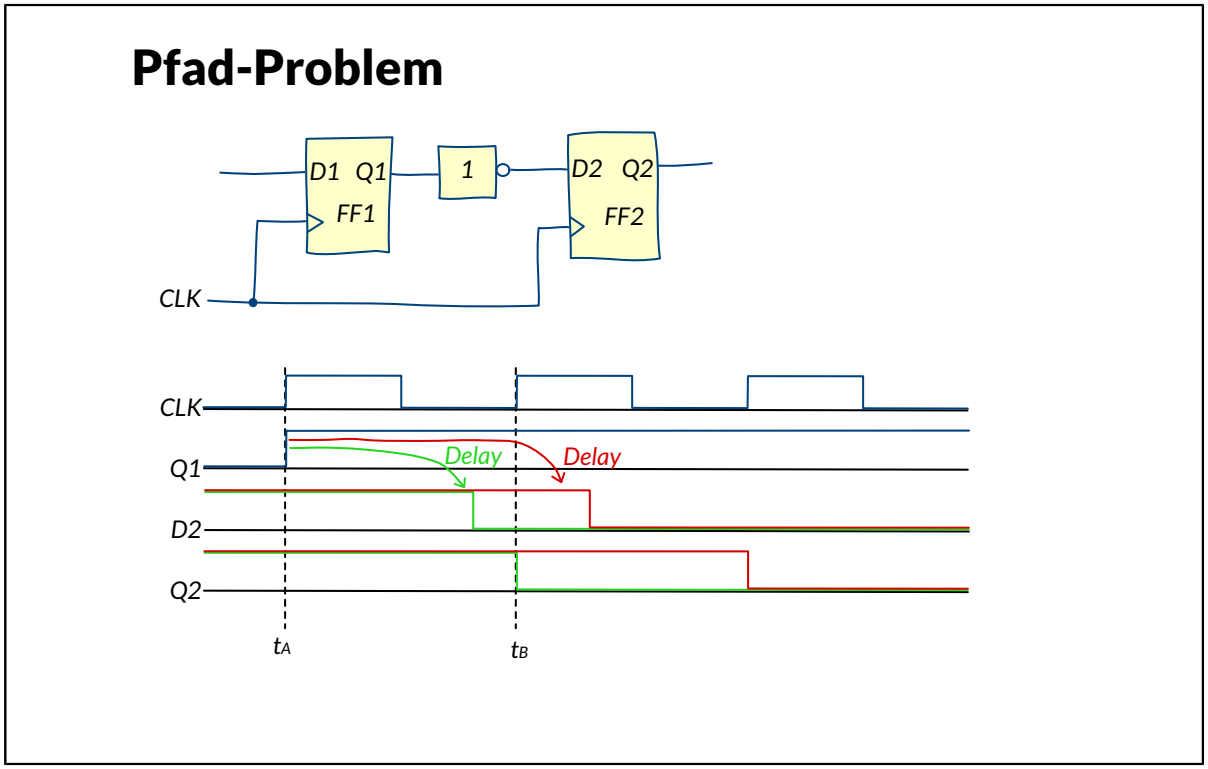
Statische Timing-Analyse: Wire-Load-Modell



Ein weiteres Problem entsteht, wenn das Delay einer mehrstufigen Logik aus den Delays der einzelnen Gatter berechnet werden soll. Die Delays von Teilsystemen können nur dann addiert werden, wenn es sich um rückwirkungsfreie Systeme handelt. Gatterschaltungen werden deshalb sinnvollerweise am Eingang eines Gatters aufgetrennt, da sich der Eingangstransistor im Wesentlichen rückwirkungsfrei verhält. Große Fehler entständen, wenn statt dessen an den Ausgängen der Gatter aufgetrennt würde, da sich die Last eines Gatters i.A. nicht rückwirkungsfrei verhält. Unter Last versteht man hier die Impedanz der Leitungen und die Eingangskapazitäten der angeschlossenen Gatter. Außerdem ist das Delay i.A. abhängig von der Flankenform. Diese wird bei der Delaybestimmung der einzelnen Gatter an Aus- und Eingang häufig als identisch angenommen.

Ein vereinfachtes Ersatzmodell für ein Gatter zeigt das Bild. Das Gatter selbst ist als Spannungsquelle mit Innenwiderstand modelliert. Der Innenwiderstand entsteht u.a. durch die Kanalwiderstände der Transistoren. Der Widerstand RL repräsentiert den Leitungswiderstand und die Kapazitäten CL und CG die Kapazität der Leitungen und der Gates der Transistoren. Das Delay hängt maßgeblich von der Last ab, d.h. von der Leitung und den angeschlossenen Gattern. Da Untersuchungen des Zeitverhaltens einer Schaltung bereits durchgeführt werden müssen, wenn noch kein Layout vorliegt, sind die Leitungseigenschaften noch nicht bekannt. Es werden daher zunächst übliche Werte aufgrund statistischer Untersuchungen verwendet. Dies wird als "Wire Load Model" bezeichnet. Sobald das Layout vorliegt, können aus den Geometrien exaktere Modelle für diese Lasten extrahiert und in das Schaltungsmodell annotiert werden. Programme, die aus elektrischen Ersatzmodellen ein Delay errechnen, nennt man Delay-Calculator.

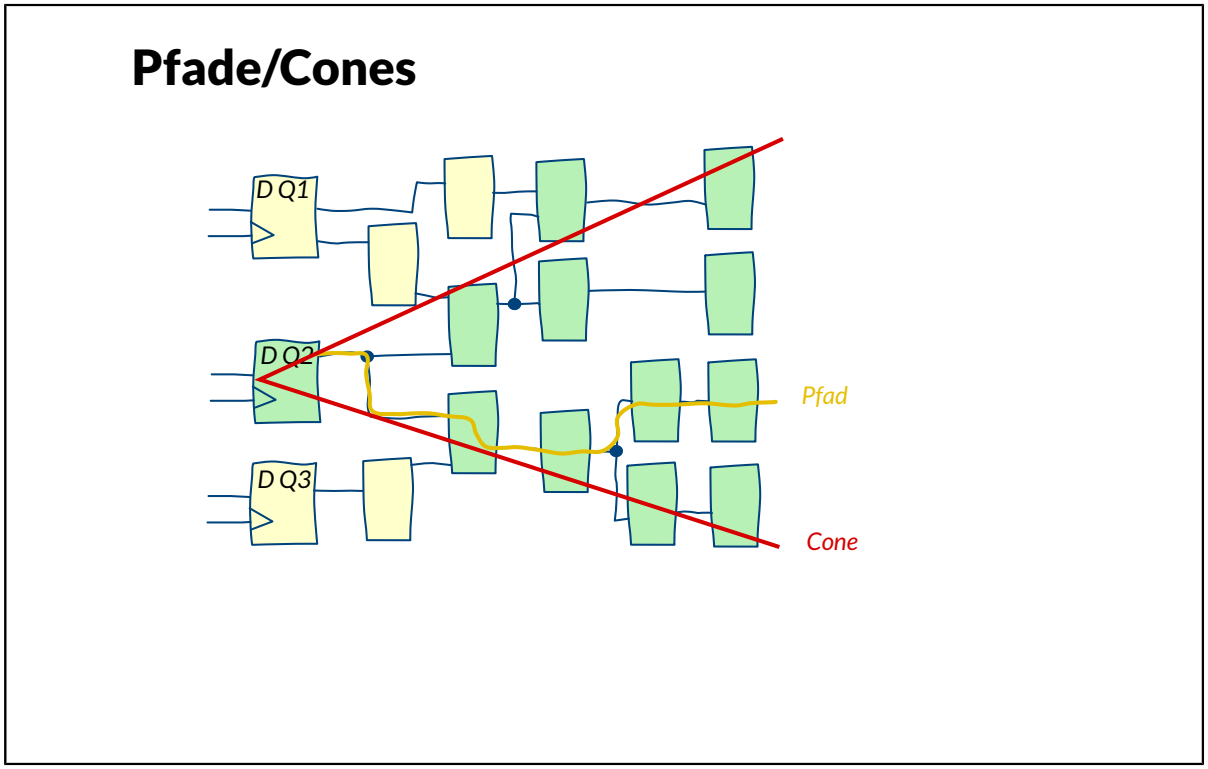
Statische Timing-Analyse: Pfad-Problem



Das Bild zeigt einen Ausschnitt aus dem Digitalteil der Ampelsteuerung. Aus dem zum Taktzeitpunkt tA in FF1 gespeicherten Zustandswert wird durch das Gatter ein neuer Wert erzeugt, der im folgenden Taktzeitpunkt tB von FF2 übernommen werden soll. Das Beispiel enthält einen kombinatorischen Pfad beginnend beim Ausgang Q1 über den Inverter bis zu Eingang D2.

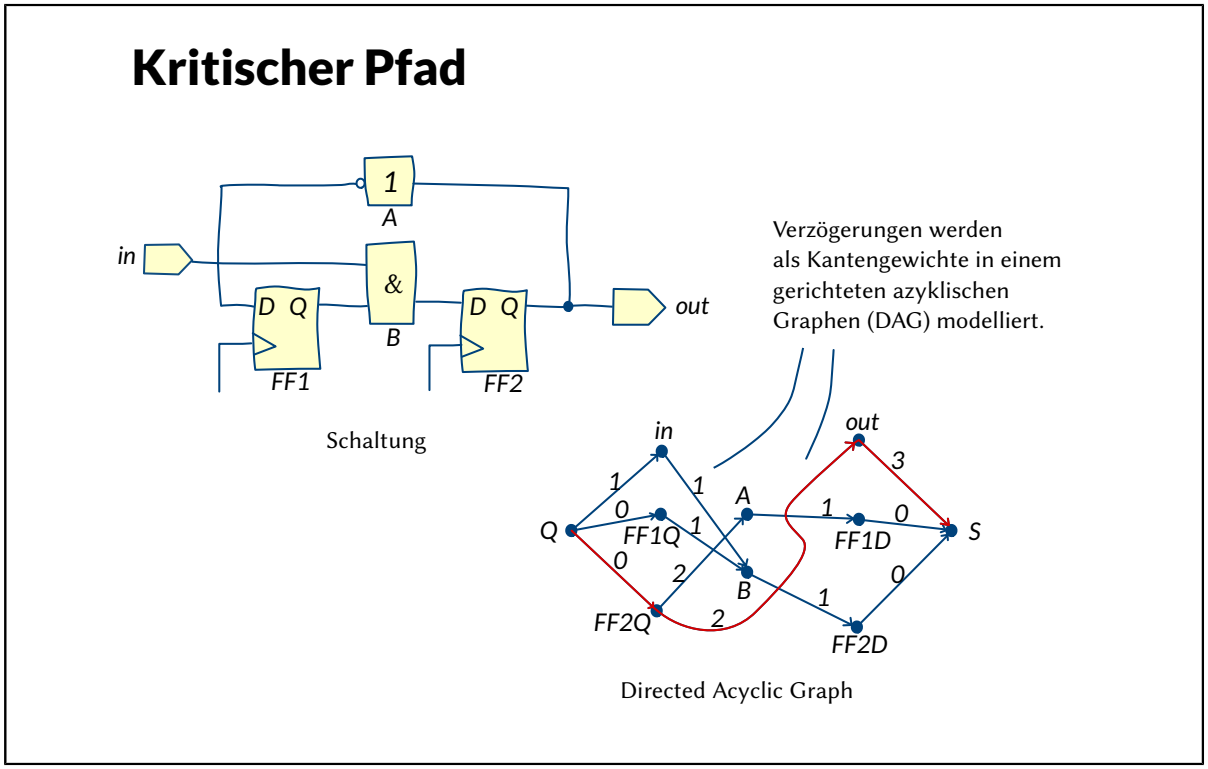
Berücksichtigt man eine Verzögerung, die auf diesem Pfad besteht, so ergeben sich die dargestellten Signale über der Zeit. Die gewünschte Funktion wird nur dann erfüllt, wenn das Pfad-Delay kleiner als die Taktperiode ist.

Statische Timing-Analyse: Pfade/Cones



Das Bild stellt einen anderen, größeren Ausschnitt der Schaltung dar. Ein Signalwechsel an Q2 kann sich auf die Ausgangssignale aller folgenden Gatter auswirken. Da der Ausgang eines Gatters wie im Beispiel mit Eingängen mehrerer anderer Gatter verbunden sein kann, gibt es mehrere Pfade, die von Q2 ausgehen. Es ergibt sich ein sich verbreiterndes Bündel von Pfaden. Dieses wird Cone (Kegel) genannt. In der Abbildung sieht man, dass ein Gatter oder Netz in mehreren Cones und damit auch in mehreren Pfaden enthalten sein kann. Allgemein ist ein Pfad eine Folge von miteinander in Signalflussrichtung verbundenen Gattern. Pfade beginnen an externen Eingängen oder Flip-Flop-Ausgängen und enden an externen Ausgängen oder Flip-Flop-Eingängen. Um sicher zu stellen, dass die Schaltung die gewünschte Funktion ausführt, muss für jeden Pfad die Verzögerung kleiner als die Taktperiode sein.

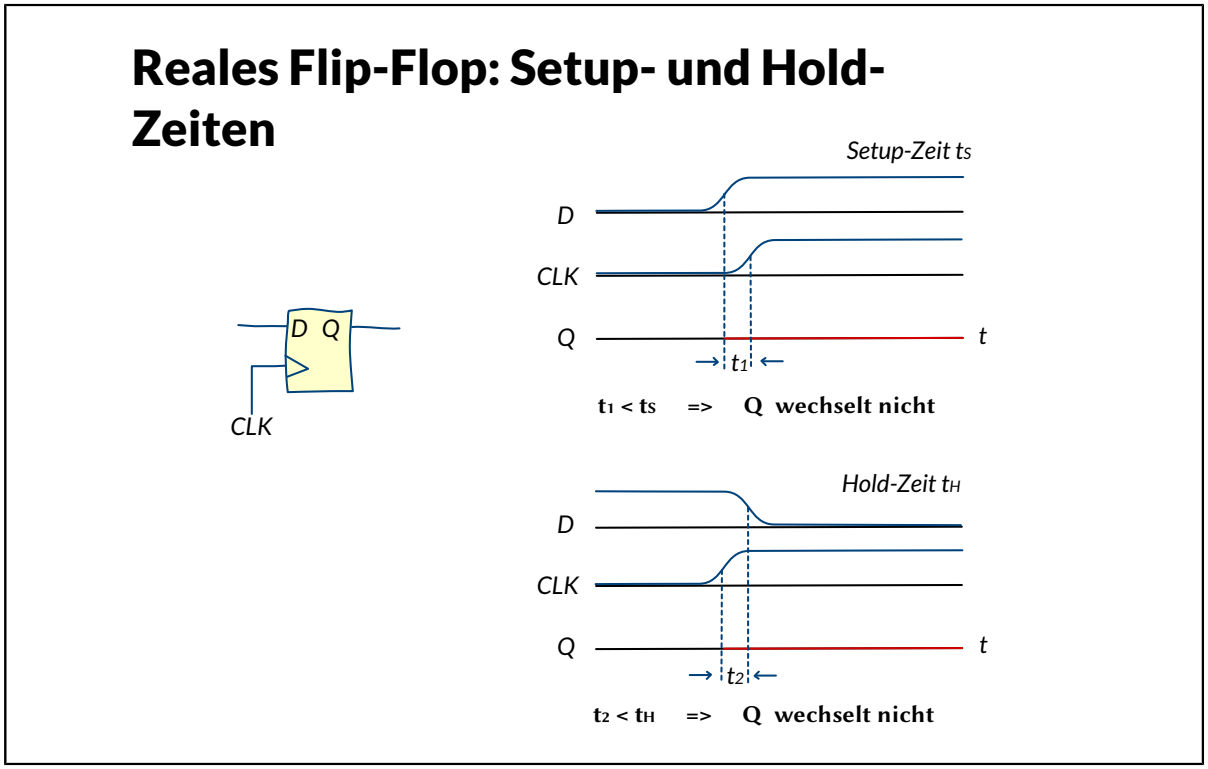
Statische Timing-Analyse: Kritischer Pfad



Zur Untersuchung des Zeitverhaltens wird eine synchrone digitale Schaltung in einen gerichteten azyklischen Graphen (DAG: Directed Acyclic Graph) umgewandelt. Die Ein- und Ausgänge der Schaltung und der Flip-Flops sowie die Logikgatter werden als Knoten modelliert. Für jede Verbindung zweier Schaltelemente wird eine gerichtete Kante eingeführt. Ein zusätzlicher Knoten Q (Quelle) wird mit allen Eingangsknoten und ein Knoten S (Senke) mit allen Ausgangsknoten über gerichtete Kanten verbunden. Die Kanten erhalten Gewichte, die die Zeitverzögerung zwischen den durch die adjazenten Knoten bezeichneten Signalen angeben.

Das Bild zeigt eine Schaltung und ihren zugehörigen DAG. Die Suche nach dem längsten Pfad von Q nach S liefert den kritischen Pfad Die Länge ist dabei die Summe der Kantengewichte entlang des Pfads. Der kritische Pfad bestimmt die maximale Taktfrequenz.

Statische Timing-Analyse: Setup- und Hold-Zeit



Untersucht man ein reales flankengesteuertes Flip-Flop, so fallen zwei Situationen auf, in denen es sich anders verhält, als man vom idealen Flip-Flop erwartet:

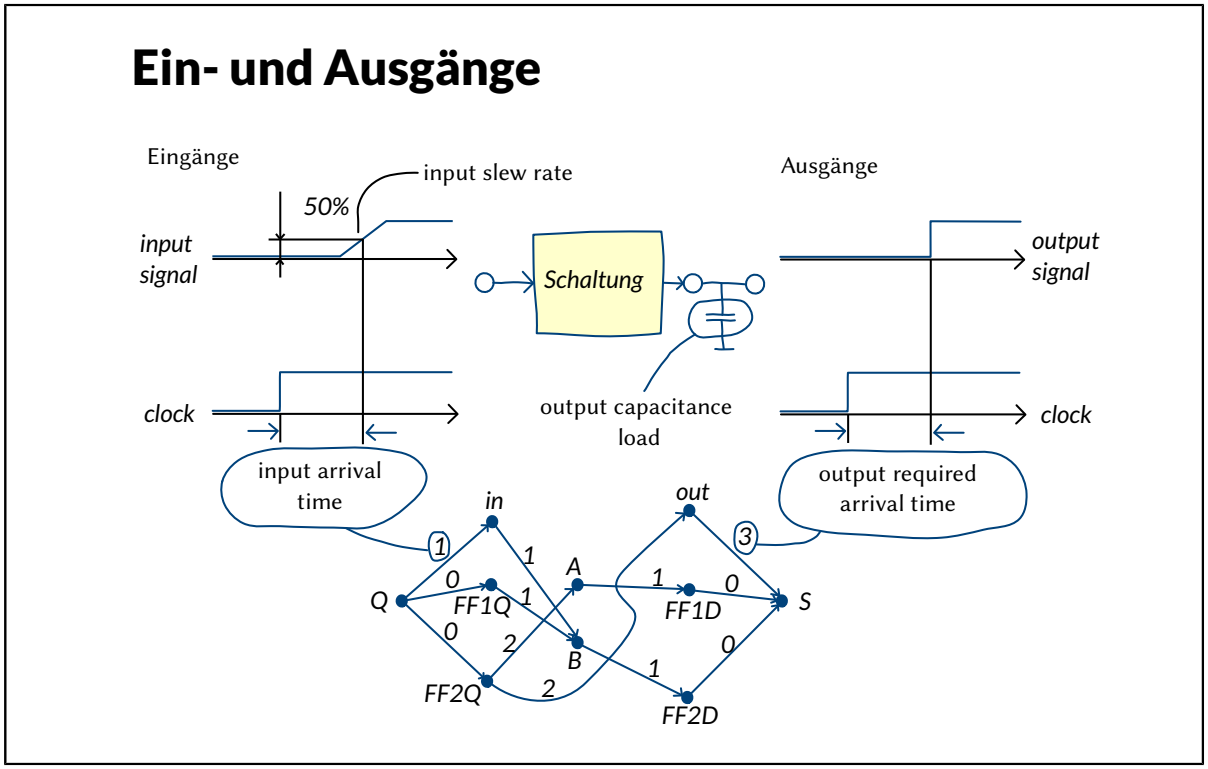
- Wechselt der Wert am Dateneingang erst sehr kurz vor der steigenden Taktflanke, dann wechselt der Ausgang Q nicht wie erwartet. Dies geschieht nur dann, wenn der Wechsel eine ausreichende Zeit vor der Taktflanke den neuen zu übernehmenden Wert besitzt.
- Wechselt der Wert am Dateneingang sehr kurz nach der steigenden Taktflanke, so wechselt der Ausgang Q nicht wie erwartet. Also muss der Wert auch eine bestimmte Zeit lang nach der Taktflanke noch den beabsichtigten Wert beibehalten.

Diesen Problemen wird durch zwei Bedingungen an das Timing begegnet:

Das Eingangssignal muss bereits um die Setup-Zeit ts vor und um die Hold-Zeit th nach der steigenden Taktflanke den gewünschten Wert haben.

Die Setup-Zeit ist eingehalten, wenn das Delay des kritischen Pfads kleiner oder gleich der Taktperiode minus der Setup-Zeit beträgt. Um zu prüfen, ob die Hold-Zeit eingehalten wird, muss das minimale Delay berechnet werden. Es muss also nicht nur der längste, sondern auch der kürzeste Pfad im DAG gefunden werden. Das minimale Delay (Länge des kürzesten Pfades) muss mindestens die Hold-Zeit betragen.

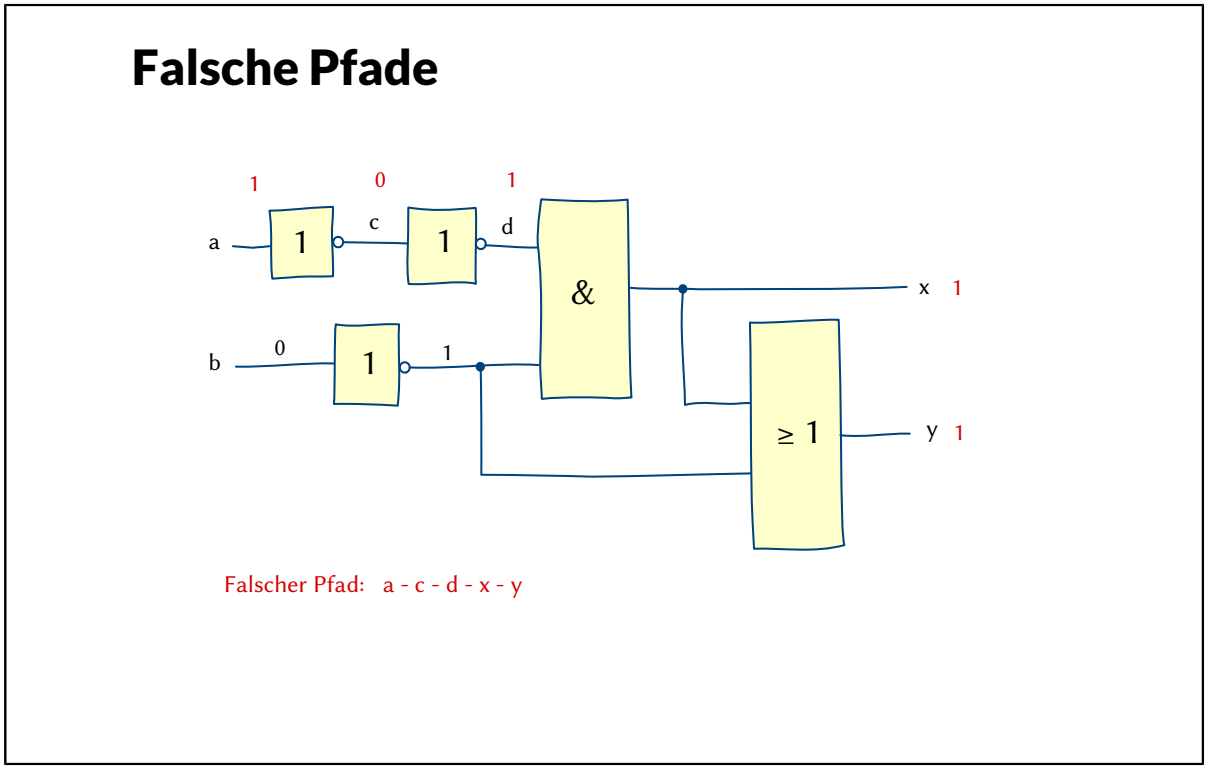
Statische Timing-Analyse: Ein- und Ausgänge



Weitere Timing-Bedingungen entstehen aus der Interaktion der zu untersuchenden Schaltung mit der Umgebung. Sind die Eingangssignale gegen den Takt versetzt, so wird dies als input arrival time bezeichnet. Sie kann im DAG durch Gewichtung der von Q ausgehenden Kanten berücksichtigt werden. In entsprechender Weise lässt sich eine output required arrival time in den Kanten berücksichtigen, die auf S weisen.

Die Kurvenform an den Eingängen wirkt sich auf das Delay aus. Dem wird in der Praxis durch Angabe einer input slew rate, also der Steigung von Signalwechseln an den Eingängen Rechnung getragen. An den Ausgängen bestimmt die angeschlossene Last das Delay, diese kann durch Angabe einer output capacitance load beschrieben werden.

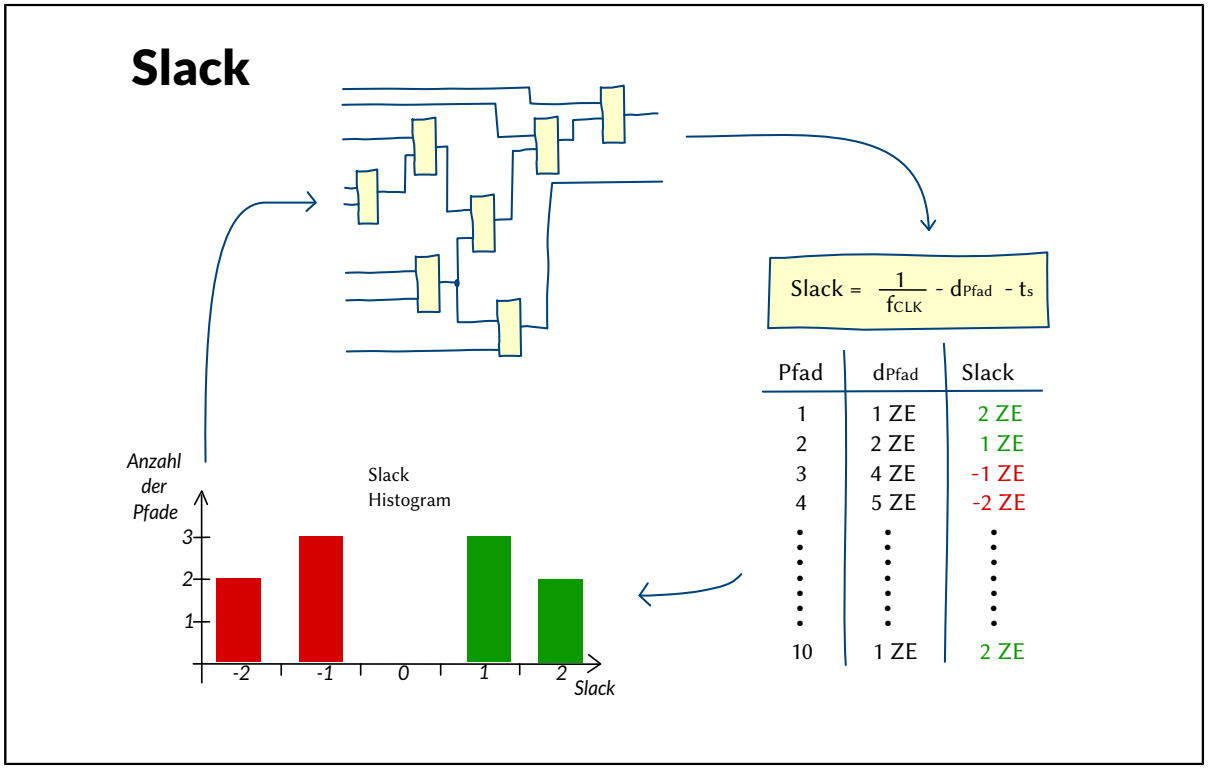
Statische Timing-Analyse: Falsche Pfade



Das Bild zeigt einen Ausschnitt aus einem Addierer. Unabhängig vom Wert des Eingangs x1 kann entlang des Pfads "a - c - d - x - y" kein Signalwechsel stattfinden. Diesen Pfad nennt man einen "falschen Pfad". Er ist nicht statisch sensibilisierbar. Ein Pfad ist dann statisch sensibilisierbar, wenn es eine Wertebelegung aller Signale gibt, die nicht zum Pfad gehören, so dass ein Wechsel am ersten Signal im Pfad eine Änderung der Signale entlang des Pfads verursacht. Um falsche Pfade erkennen zu können, ist die Kenntnis der Logikfunktionen der Elemente erforderlich.

Falsche Pfade können mit dem D-Algorithmus erkannt werden, der im Abschnitt Test beschrieben wird. Dort ist das Ziel, Signalwechsel über Pfade hinweg zu sensibilisieren und zu beobachten.
In Zusammenhang mit falschen Pfaden geht es darum, die Sensibilisierbarkeit von Pfaden zu überprüfen. Der D-Algorithmus leistet beides.

Statische Timing-Analyse: Slack



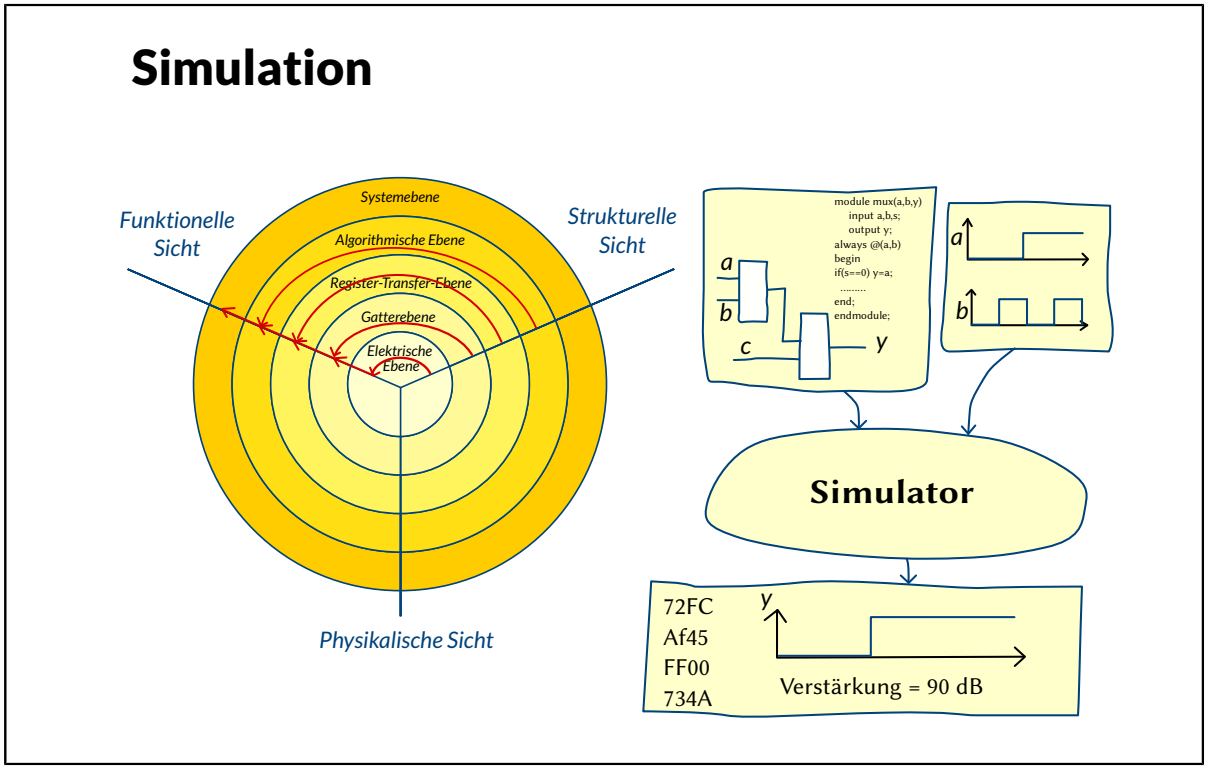
Als Slack wird die Zeit bezeichnet, die ein Pfad länger sein könnte, ohne die Timing-Bedingungen zu verletzen. Ein negativer Slack bedeutet, dass ein Pfad zu lang, die Setup-Zeit also unterschritten wird. Im Slack-Histogramm ist über dem Slack-Wert die Anzahl der diesen Slack-Wert besitzenden Pfade aufgetragen. Aus diesem Diagramm ist ersichtlich, ob nur wenige Pfade negativen Slack besitzen, und daher lokale Optimierungen sinnvoll sind, oder ob im anderen Fall grundlegende Architekturänderungen erforderlich sind, um ein Design zu erhalten, das die Timing-Bedingungen einhält.

Electronic Design Automation (EDA)

Digitale Simulation

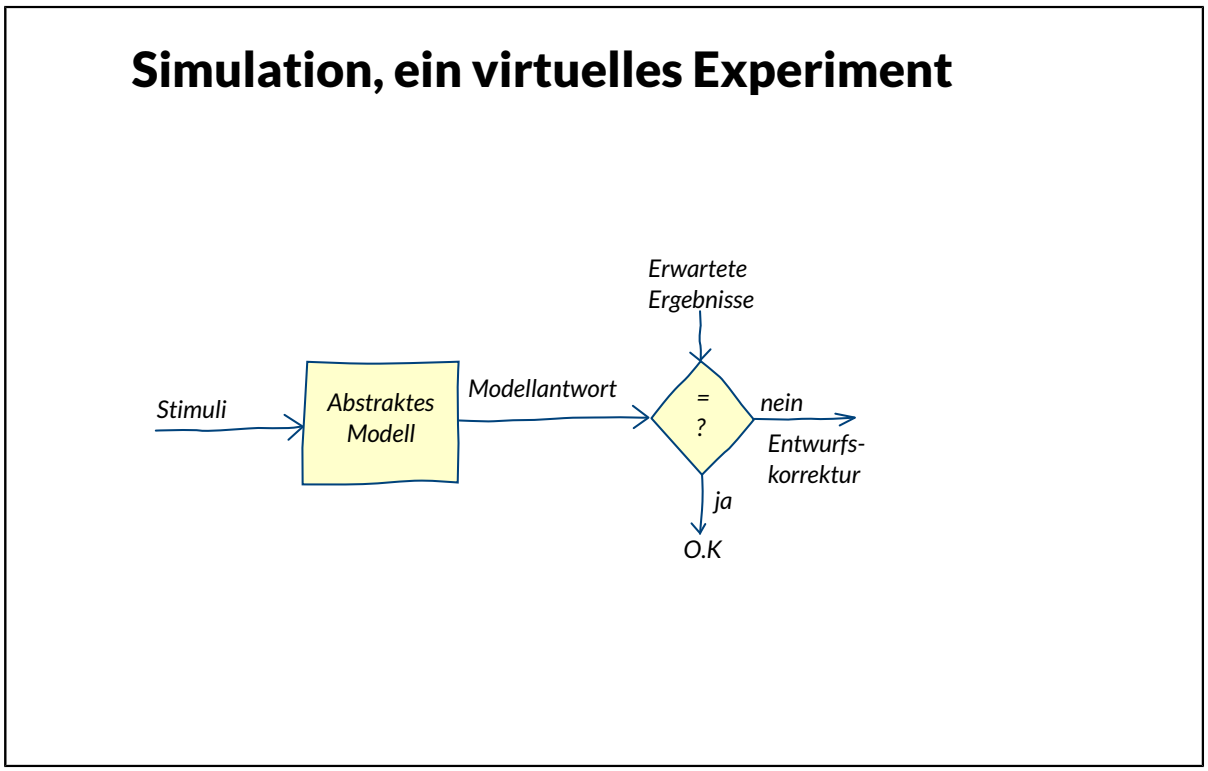
Simulation
Simulation, ein virtuelles Experiment
Simulationsebenen
Abstraktion
Wertdiskretisierung: Logikmodelle
Glitch
Delay-Modell I
Delay-Modell-II
Digitale Simulationsmethodik
Übersetzende Verfahren
Strukturierung
Compiled Code: Beispiel
Talbellengestützte Verfahren
Verkettete Datenstrukturen: Elemente
Verkettete Datenstrukturen: Beispiel
Äquitemporale Iteration
Ereignisse
Ablauf ereignisgesteuerter Simulation
Ereignisverwaltung
Ereignissteuerung: Beispiel
Multi-Level-Simulation

Digitale Simulation: Simulation



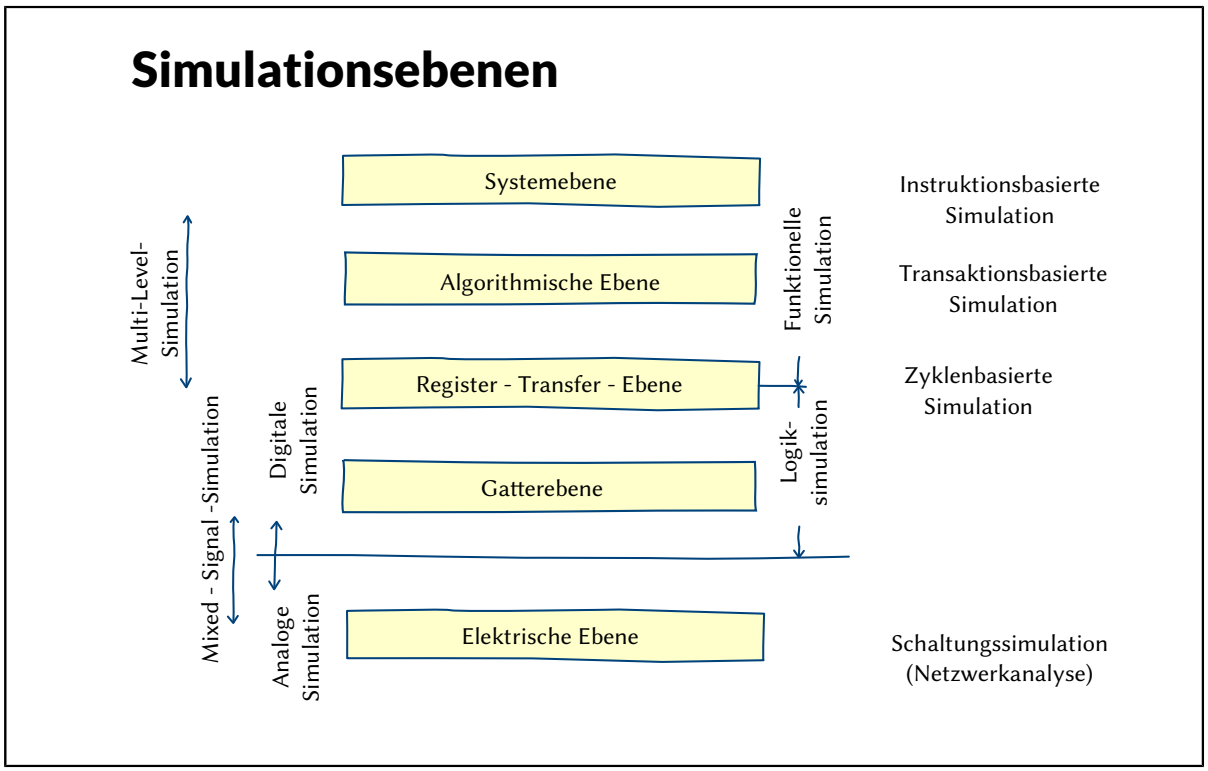
Simulation bedeutet, ein (Software-)Modell der Schaltung zu benutzen, um ihr Verhalten zu analysieren. Dazu werden die Eingänge des Schaltungsmodells mit Signalwerten (so genannten Stimuli) belegt und die berechneten Werte an den Ausgängen (oder auch an beliebigen Stellen innerhalb der Schaltung) beobachtet.

Digitale Simulation: Simulation, ein virtuelles Experiment



Simulation ist ein virtuelles Experiment. Zu seiner Durchführung versorgt der Entwickler das Modell der Schaltung mit sinnvollen Eingangswerten und überprüft durch die Simulation, ob am Ausgang der Schaltung die erwarteten Werte erscheinen. Es wird ein Testfall konstruiert und durch Simulation nachgewiesen, dass für diesen Testfall die Spezifikation erfüllt ist. Natürlich wird versucht, durch möglichst viele Testfälle alle Eventualitäten zu überprüfen. Die Menge aller möglichen Testfälle ist aber außer für sehr kleine Schaltungen viel zu groß, als dass sie jemals durch Simulation abgedeckt werden könnte. Ein 16-Bit-Addierer hat zum Beispiel 2^32 mögliche Eingangswerte. Ungefähr 4,3 Milliarden Testfälle müssen durchgerechnet werden, nur um einen einzigen 16-Bit-Addierer vollständig zu testen. Das im Begriff der Verifikation enthaltene Versprechen - der Beweis der Wahrheit - wird von Simulatoren nicht eingehalten. Simulation kann Fehler finden, aber nicht feststellen, ob eine Schaltung fehlerfrei ist.

Digitale Simulation: Simulationsebenen



Simulatoren sind auf allen Entwurfsebenen sehr wichtige EDA-Werkzeuge. Je nach Ebene sind die verwendeten Verfahren und Algorithmen jedoch sehr unterschiedlich. Auf der elektrischen Ebene löst ein **Schaltungssimulator** (historisch auch **Netzwerkanalyseprogramm** genannt) das eine elektrische Schaltung beschreibende System von nichtlinearen Differentialgleichungen. Dies ist für Analogschaltungen die sinnvolle Simulationsebene, auf der das Wert- und Zeitverhalten detailliert analysiert werden kann. Für größere - insbesondere digitale - Schaltungen sind Schaltungssimulatoren wegen ihrer hohen Rechenzeitanforderungen ungeeignet. Es muss daher unter Verzicht auf Genauigkeit eine höhere Abstraktionsebene gewählt werden.

In den 50er und 60er Jahren hat sich die **digitale Simulation** primär mit der Korrektheit der Funktion und kaum mit dem Zeitverhalten befasst. Die früheste Methode zur **Logiksimulation** ist die **"Compiled-Code"**-Methode, die später erläutert wird und heute vorwiegend für höhere Ebenen verwendet wird. Eine digitale Schaltung wird durch boolesche Gleichungen beschrieben und diese werden in jedem Taktschritt ausgewertet. Dieses Vorgehen ist sehr rechenzeitintensiv und nur für synchrone Schaltungen geeignet. Zudem wird das Zeitverhalten der Strukturelemente nicht berücksichtigt.

Mitte der 60er-Jahre wurde ein neues Verfahren, die **ereignisgesteuerte Logiksimulation**, vorgestellt, die sich schnell durchsetzte, da sie wesentlich schneller ist und die Simulation asynchroner Schaltungen sowie die Berücksichtigung von individuellen Verzögerungszeiten erlaubt. Sie vermeidet die Untersuchung solcher Schaltungsteile, die zu bestimmten Zeiten inaktiv sind. Digitale Schaltungen sind bis zu 90 % inaktiv.

Wegen der großen Komplexität der Schaltungen wird heute auf der Gatterebene wieder zwischen Funktion und Zeitverhalten getrennt. Es wird eine rein funktionelle Simulation durchgeführt und das Zeitverhalten mithilfe der statischen Timinganalyse analysiert. Diese Trennung von Funktion und Zeitverhalten hat sich auch auf den höheren Entwurfsebenen bewährt. Simulatoren auf diesen Ebenen sind weniger genau als Gatterebenensimulatoren, ihre Handhabung ist jedoch einfacher und sie sind erheblich schneller. Ihre Geschwindigkeit erreichen sie durch die Abstraktion des Zeitverhaltens, indem beispielsweise die Funktion nur zum Zeitpunkt einer Taktflanke analysiert wird (**zyklenbasierte Simulation**), sowie durch das Arbeiten mit Mehrbit- statt Einzelbitvariablen.

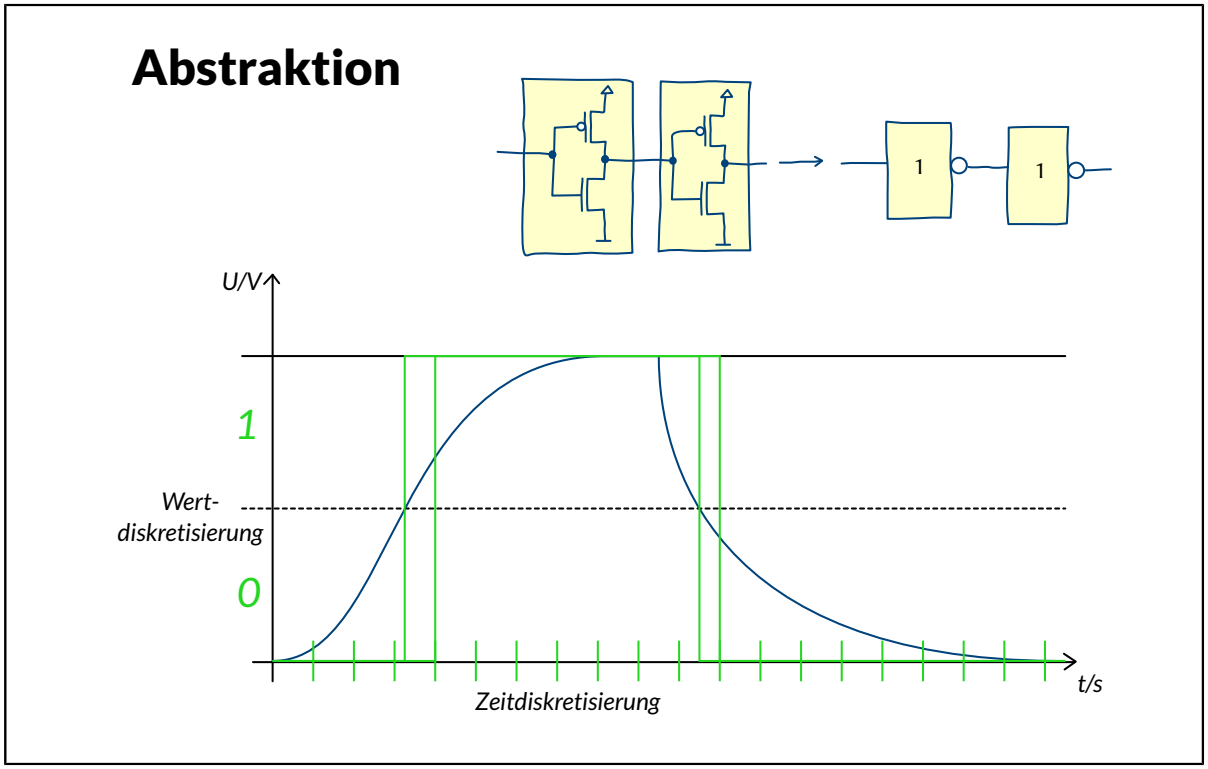
Zyklenbasierte Simulatoren sind der Register-Transfer-Ebene zuzuordnen. In der Ebene darüber nimmt die Abstraktion auch bei der Simulation weiter zu. So werden auf der algoritmischen Ebene nicht mehr die Werte einzelner Variablen betrachtet, sondern das Verhalten mehrerer Variablen in Form von sogenannten **Transaktionen**, wie z.B. das Einschreiben eines Datenwortes in einen Speicher, oder der Zugriff auf einen Bus mit dem zugehörigen Datentransfer.

Auf der Systemebene existieren schließlich noch sogenannte **Instruction Set**-Simulatoren, die den Instruktionssatz eines entworfenen Prozessors per Simulation nachbilden und damit in der Lage sind, grundsätzlich auf diesem "virtuellen" Prozessor auch Software ablaufen zu lassen. Dies ist aus Komplexitätsgründen aber nur für relativ kurze Programmteile möglich.

Da die Simulationszeit stets problematisch ist, entstanden für solche Fälle, in denen auf die Genauigkeit der Gatterebene nicht verzichtet werden konnte, so genannte **Hardwarebeschleuniger** und **Logikemulatoren**, bei denen der Simulationsalgorithmus in Hardware abgebildet wird. Es handelt sich also um spezielle Rechner, die mit großer Geschwindigkeit einen festverdrahteten bzw. anwenderprogrammierbaren Ablauf ausführen.

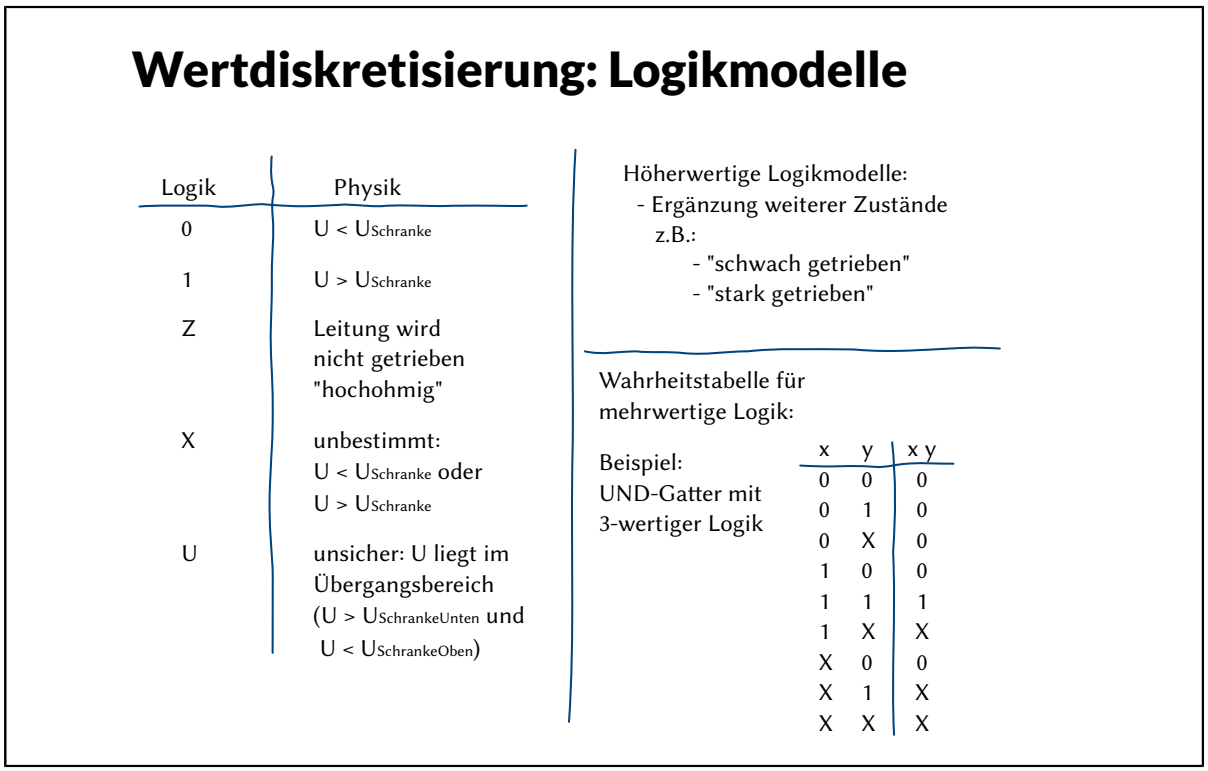
Simulation ist ein umfangreiches Gebiet und kann hier nicht vollständig behandelt werden. Die folgenden Betrachtungen werden sich deshalb auf die Gatterebene beschränken. Wegen ihrer grundlegenden Bedeutung soll dabei insbesondere die verzögerungszeitberücksichtigende Logiksimulation betrachtet werden. Alle weiter abstrahierende Simulatoren können ohne weiteres auf die hier beschriebenen zurückgeführt werden.

Digitale Simulation: Abstraktion



Für die Simulation auf Gatterebene muss die physikalische Realität mit kontinuierlichen Signalverläufen stark abstrahiert werden. Sowohl der Wertebereich als auch der Zeitliche Verlauf der Signale werden diskretisiert. In der einfachsten Form der digitalen Simulation kann ein Signal nur noch die Werte 1 und 0 annehmen. Der Simulator muss zu einem gegebenen diskreten Zeitpunkt alle Signalwechsel auswerten, sich unter Berücksichtigung der Gatter- und Leistungsverzögerungszeiten deren Auswirkung für zukünftige diskrrete Zeitpunkte merke und dann die Simulation am jeweils nächsten diskreten Zeitpunkt, der mit einer sogenannten Zeitschrittsteuerung gefunden wird, fortgesetzen.

Digitale Simulation: Wertdiskretisierung: Logikmodelle



Es liegt zunächst nahe, digitale Schaltungen mit zwei Werten (0 und 1) zu behandelen. Dies reicht jedoch für viele Anwendungen nicht aus, da beispielsweise auch hochohmige Zustände von Tri-State-Schaltungen behandelt werden müssen, oder Unbestimmtheit bzw. Unsicherheit modelliert werden soll.

Um elektrisch hochohmige Signale darstellen zu können, ergänzt man die Menge der Signalwerte um den Wert Z. Dieser Wert bedeutet ,dass das beterffende Signal nicht durch irgendwelche Schaltungsteile zur Annahme eines logischen Signalwechsel gezwungen wird.

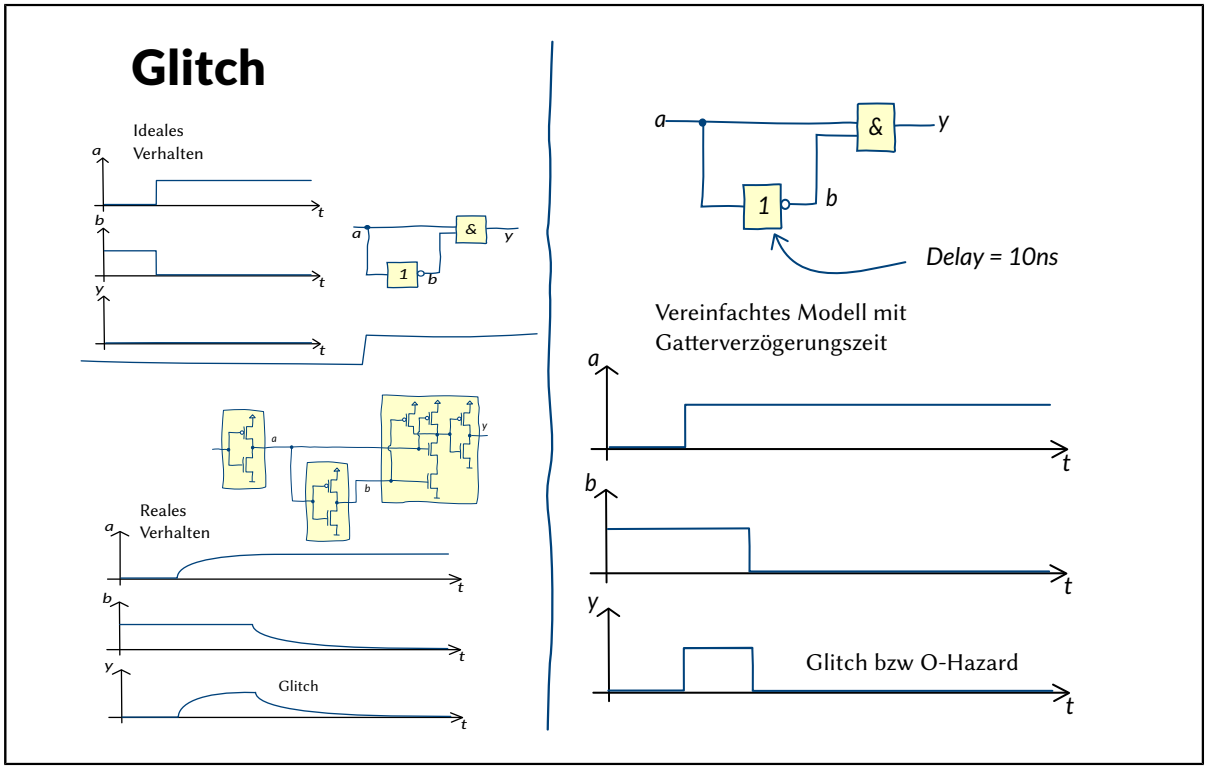
In den meisten Fällen wird die Menge {0,1,Z} um zwei weitere Werte erweitert:

- Ein Wert X mit der Interpretation "entweder 0 oder 1" d.h. es liegt Unbestimmtheit vor.
- Ein Wert U mit der Interpretation "elektrisch weder 0 noch 1", d.h. es liegt Unsicherheit in Form eines Zwischenwertes vor.

Auf diese Weise ist ein fünfwertiges Logikmodell mit der Wertemenge {0,1,Z,X,U} entstanden, für das die wntsprechendenVerknüpfungstabellen zu erstellen sind. Als vereinfachtes Beispiel zeigt das Bild dazu die Wahrheitstabelle fürein UND-Gatter in einer dreiwertigen Logik mit {0,1,X}.

Heutige Logiksimulatoren haben noch wesentlich größere Wertmengen. Damit können beispielsweise unterschiedliche Signalstärken (und damit in diskreter Form der Innenwiderstand des Signaltreibers) modelliert werden. Von Interesse ist auch die Erkennung steigender und fallender Flanken, kurzeitiger Pegeleinbrüche u.ä. über die DDarstellung von Signalwerten.

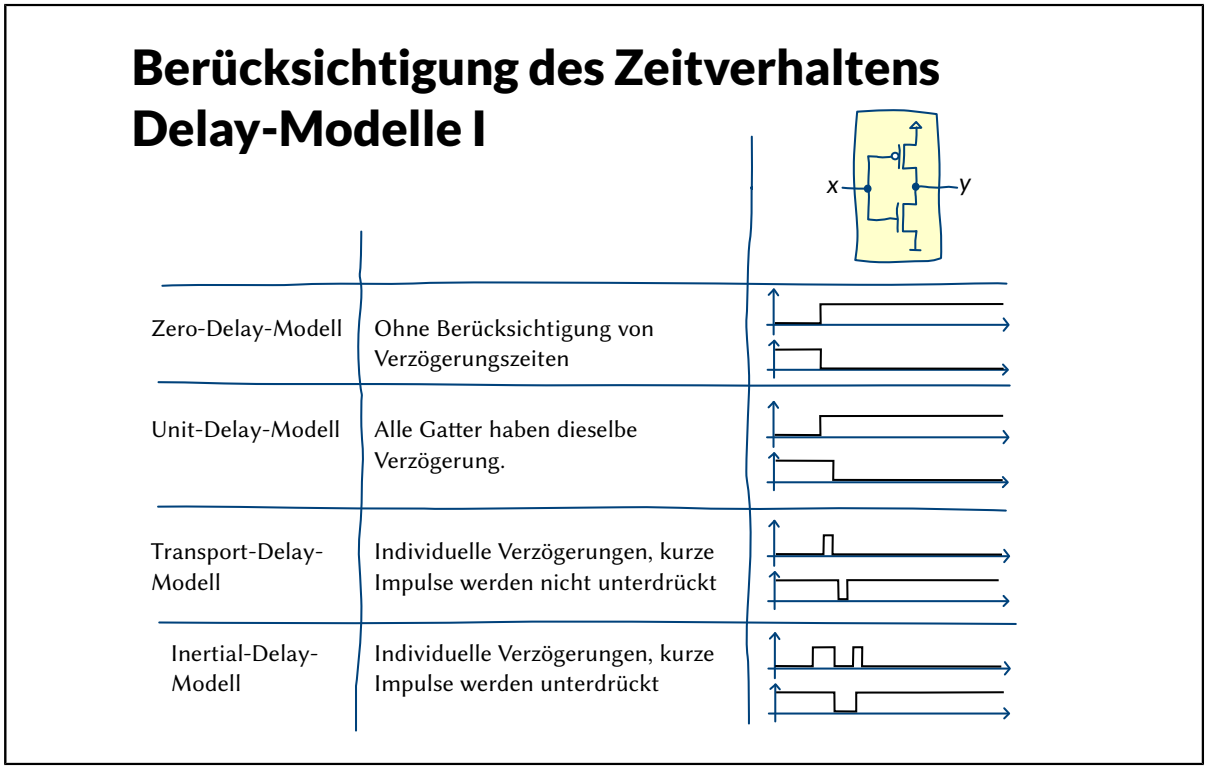
Digitale Simulation: Glitch



Ein Glitch oder ein so gennanter Hazard ist ein aufgrund unterschiedlicher Signallaufzeiten in den Pfaden einer Schaltung bedingte kurz Änderung des logischen Pegels. Ein 0-(1-)Hazard ist ein 1-(0-)Impuls auf einem Signal, welches aufgrund der logischen Funktion ständig 0(1) sein sollte.

Wir können Signale mit Mengen von Signalwertsequenztripeln der Form (a1,a2,a3) beschreiben. Die Signale a1 und a3 sollen stabile Signalpegel sein, d.h. aus {0,1}. Konstante Signale sind 0= {(0,0,0)} und 1={(1,1,1)}. Steigende (r) und fallende (f) Flanken können wir durch r={(0,0,1)(0,1,1)} bzw. f={(1,1,0)(1,0,0)} darstellen. Ein 0 Hazard liegt bei h={(0,1,0)} vor. Ein 1-Hazard ist entsprechend durch H={(1,0,1)} gegeben.

Digitale Simulation: Delay-Modell I



Zero-Delay

Bei synchronen Schaltungen ist man manchmal nicht an der Berechnung des genauen Zeitverlaufs von Signalwerten interessiert. Es reicht häufig zu wissen, welcher Wert sich an den Ausgängen der Schaltnetze nach "hinreichend" langem Warten einstellt. Dies entspricht der Tatsache, dass man die Taktrate entsprechend niedrig wählt.

Man braucht in diesem Fall nur noch die Funktion eines Gatters zu betrachten und kann daher die interessierenden Werte wesentlich schneller berechnen als in solchen Fällen, in denen auch das zeitliche Verhalten bestimmt werden muss. Die Verwendung eines Zero-Delay-Modells in einem Simulator bedeutet also, dass man das Verzögerungszeitverhalten ignoriert und eine rein funktionelle Simulation vornimmt.

Unit-Delay

Eine erste Berücksichtigung des Zeitverhaltens bietet die so genannte Unit-Delay-Simulation. Hierbei wird angenommen, dass alle Gatter ihre Ergebnisse gleich schnell berechnen. Man hat damit immer noch keine Information über den genauen zeitlichen Ablauf. Delays von komplexeren Gattern sind normalerweise größer als die von einfacheren Gattern. Diese Unterschiede können im Unit-Delay-Modell nicht berücksichtigt werden. Dadurch kann die Simulation einige der Pegeleinbrüche, die durch Delays erzeugt werden, nicht erkennen. Andere Pegeleinbrüche, die auftreten, wenn alle Gatter in etwa die gleiche Komplexität haben, können aber erkannt werden.

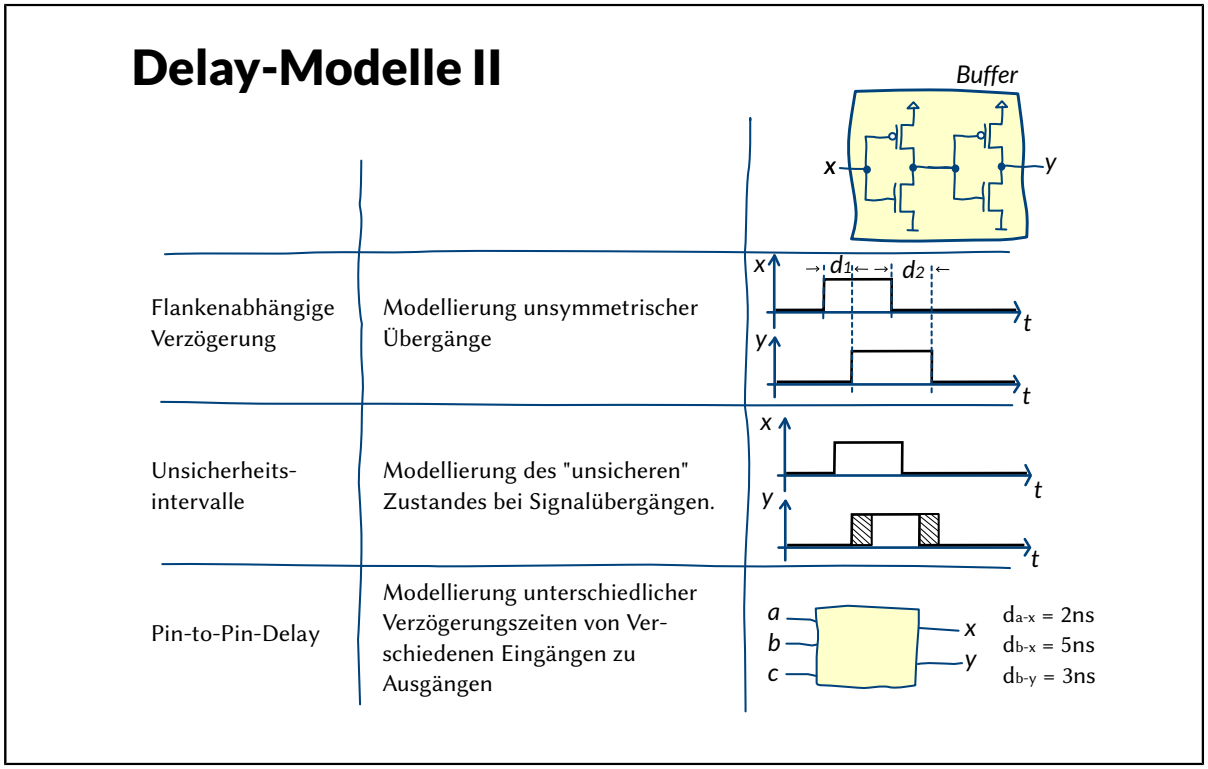
Transport-Delay

Die Transportverzögerung ermöglicht die Zuweisung individueller Verzögerungszeiten an die Gatter. Ändert sich also ein Eingangswert zum Zeitpunkt t, so ist die Wirkung am Ausgang zum Zeitpunkt t +d zu beobachten. Die Größe d wird Transport-Delay genannt. Dabei wird angenommen, dass auch beliebig kurze Impulse unverändert (lediglich verzögert) vom Eingang zum Ausgang übertragen werden.

Inertial-Delay

Aufgrund der vorhandenen Kapazitäten reagieren praktische Schaltungen nicht auf sehr kurze Eingangsimpulse, sie haben eine gewisse Trägheit (Inertia). Simulatoren bieten daher in der Regel die Möglichkeit, neben dem Transport-Delay eine untere Zeitschranke für relevante Eingangsimpulse anzugeben.

Digitale Simulation: Delay-Modell-II

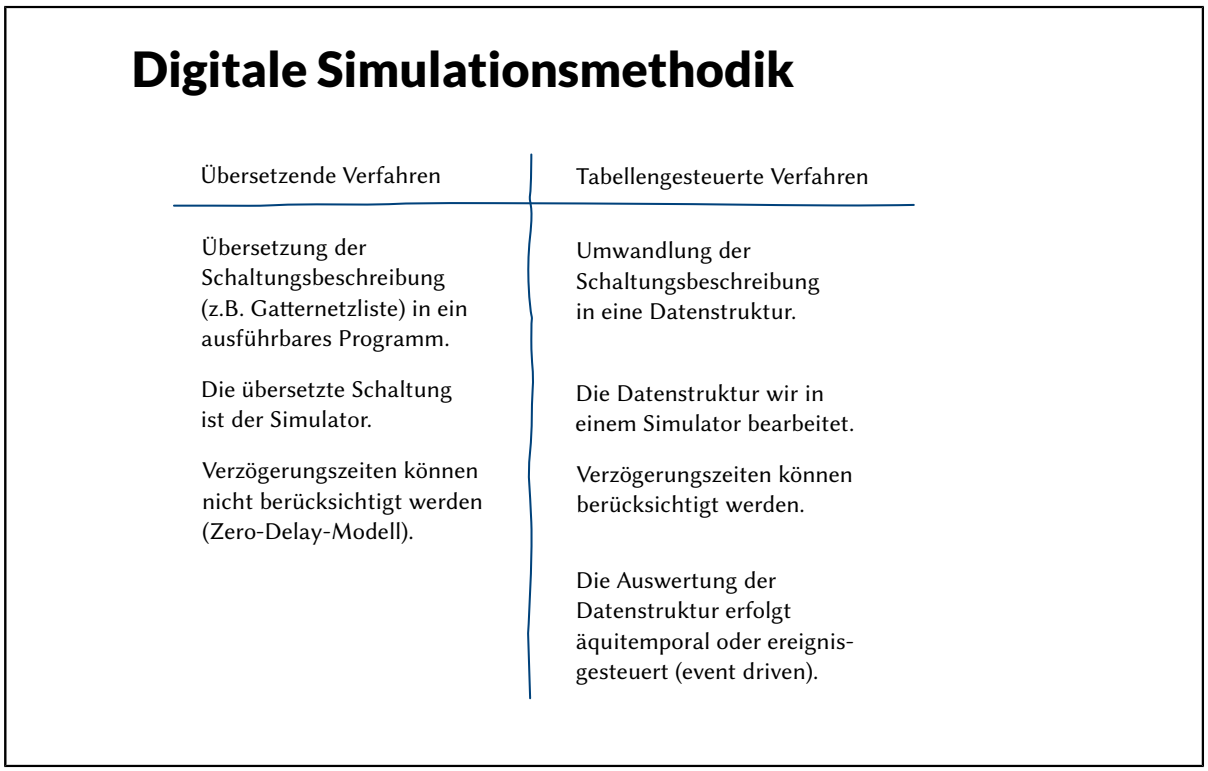


In einer weiteren Verfeinerung kann berücksichtigt werden, dass Signalwechsel auf ansteigende und abfallende Flanken häufig unterschiedlich viel Zeit benötigen.

Außerdem kann berücksichtigt werden, dass die genauen Verzögerungszeiten meist unbekannt sind oder beispielsweise Fertigungsschwankungen unterliegen und lediglich Bereiche (min,max) angegeben werden können.

Schließlich können die Verzögerungszeiten auf den Pfaden zwischen verschiedenen Ein- (und ggf. Aus-)gängen eines Gatters verschiedene Werte aufweisen. Dies wird durch so genannte Pin-to-Pin-Delays berücksichtigt.

Digitale Simulation: Digitale Simulationsmethodik

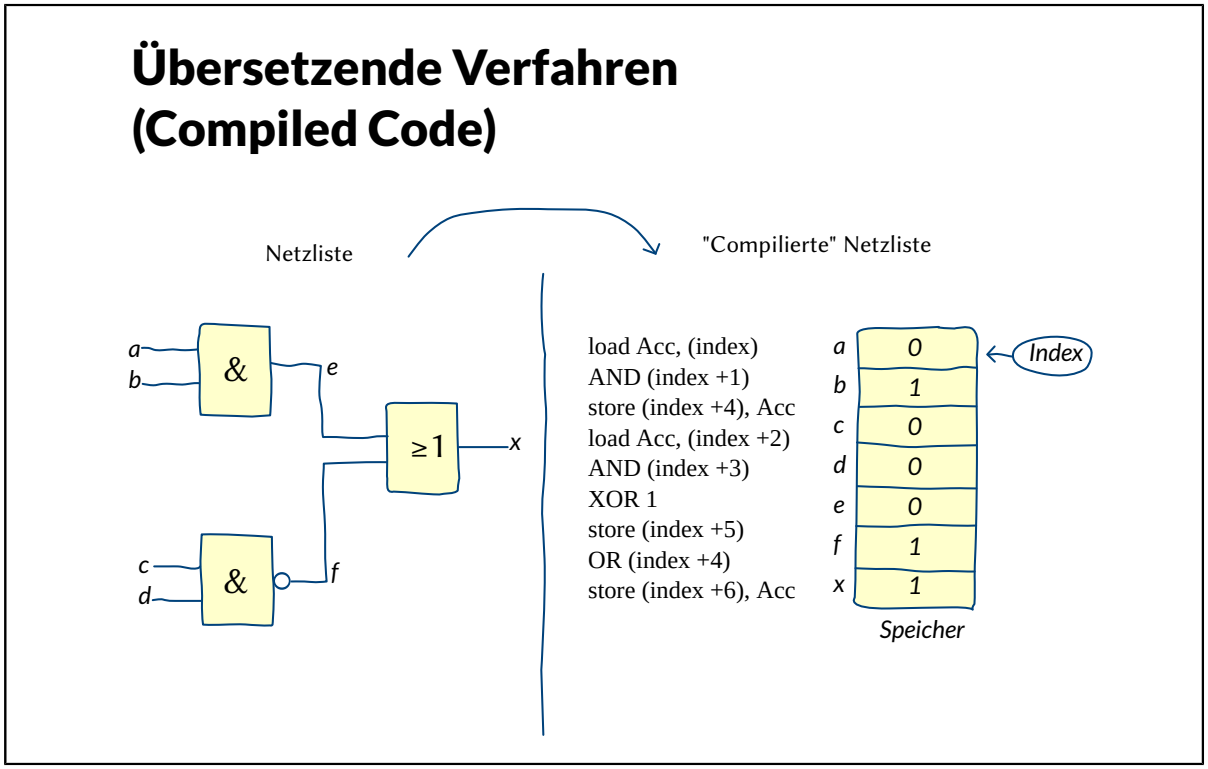


Die Aufgabe eines Simulationsalgorithmus besteht darin, ein Modellierungskonzept (Signalwerte, Zeit, logisches Verhalten) auf die Architektur eines Rechners zu übertragen. Die beste Lösung wird dabei erreicht, wenn die Architektur des Rechners genau mit dem Modellierungskonzept übereinstimmt (Algorithmus in Hardware) oder diesem zumindest ähnlich ist (Hardware-Beschleuniger). Nach diesem Grundsatz sind eine Reihe von speziellen Simulationsrechnern gebaut worden. Hardware-Beschleuniger, die durch besondere Rechnerarchitekturen, wie z.B. Pipelining, sequentielle Algorithmen um Faktoren von etwa 20-50 beschleunigen, haben auch kommerzielle Bedeutung erreicht.

Im Normalfall jedoch sind konventionelle von Neumann-Architekturen als Rechner zu benutzen. Hauptproblem dabei ist die Abbildung der im zu simulierenden Entwurf vorhandenen Parallelität auf die streng sequentielle Arbeitsweise des Rechners.

Wir unterscheiden dabei zwei grundsätzlich verschiedene Ansätze: übersetzende und tabellengesteuerte Verfahren. Bei den übersetzenden Verfahren wird die zu simulierende Schaltung durch ein (Maschinen- oder Hochsprachen-)Programm dargestellt. Hierfür sind naturgemäß Darstellungen in einer Hardware-Beschreibungssprache(HDL) besonders geeignet. Abgesehen von verschiedenen Steuerungsmechanismen stellt die Schaltungsbeschreibung selbst den Simulator dar. Dagegen wird bei tabellengesteuerten Verfahren die Schaltung in eine Datenstruktur umgesetzt. Diese Datenstruktur beschreibt alle logischen, topologischen und zeitlichen Eigenschaften der Schaltung und wird von einem schaltungsunabhängigen Programm, dem Simulator, bearbeitet.

Digitale Simulation: Übersetzende Verfahren



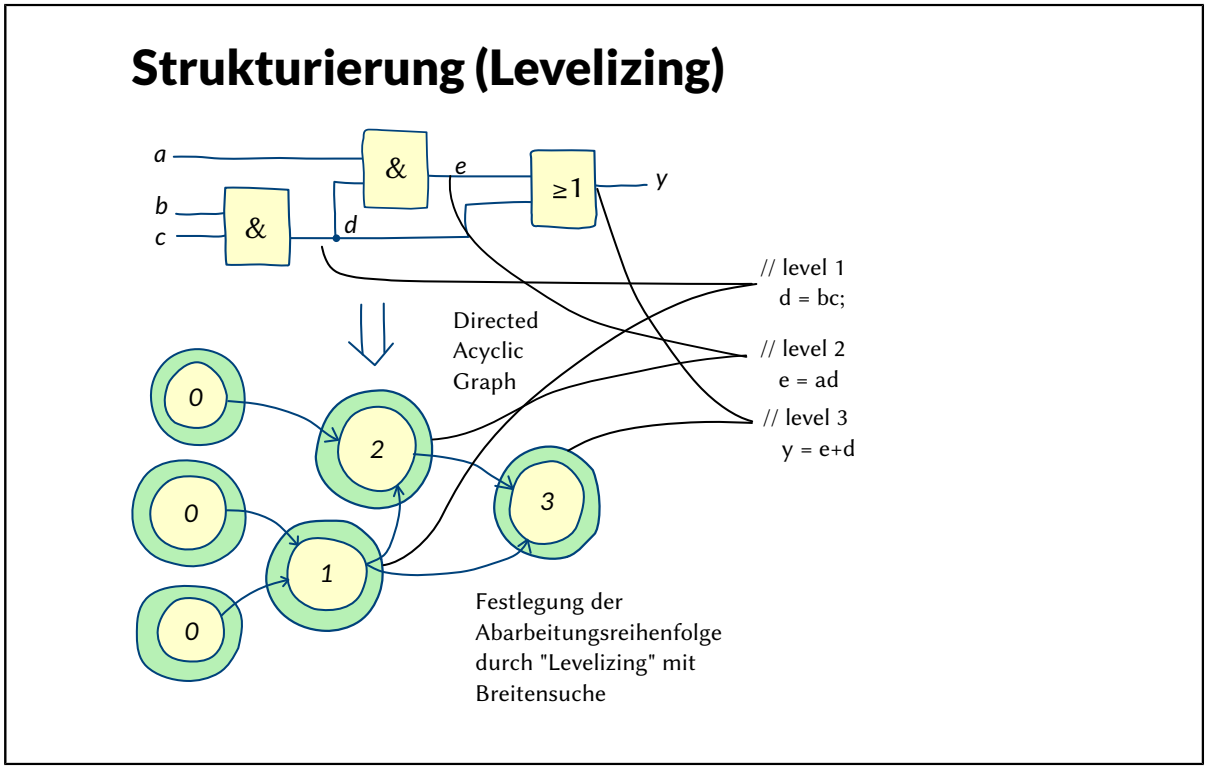
Zwar kann grundsätzlich jedes Modellierungskonzept, d.h. eine Schaltung auf beliebigen Ebenen des Y-Diagramms, in ausführbaren Programmcode umgesetzt werden, i.a. jedoch unterliegen übersetzende Verfahren (Compiled-Code-Simulation) insbesondere folgenden Einschränkungen:

- Es können nur kombinatorische oder strikt synchrone Schaltungen behandelt werden.
- Das genaue Zeitverhalten von Strukturelementen kann nicht direkt modelliert werden.

Ein Compiled-Code-Simulator übersetzt die Schaltungsbeschreibung in eine Folge von Maschinenbefehlen (oder Programmschnitten), die sowohl die Funktion als auch die Verbindungen der Schaltungselemente beschreibt. Für jedes dieser Elemente gibt es also eine oder mehrere Instruktionen sowie einen zugehörigen Eintrag im Speicher, der den momentanen Zustand des Elementes beschreibt.

Ein Beispiel für eine Compiled-Code-Simulation auf Gatterebene zeigt das Bild. Die Gatterfunktionen werden dabei direkt in Assembler-Befehle umgesetzt. Die zugrundeliegende Ein-Adress-Maschine führt die Operationen in einem Akkumulator aus. Eine direkte XOR-Operation mit "1" realisiert die Negation.

Digitale Simulation: Strukturierung



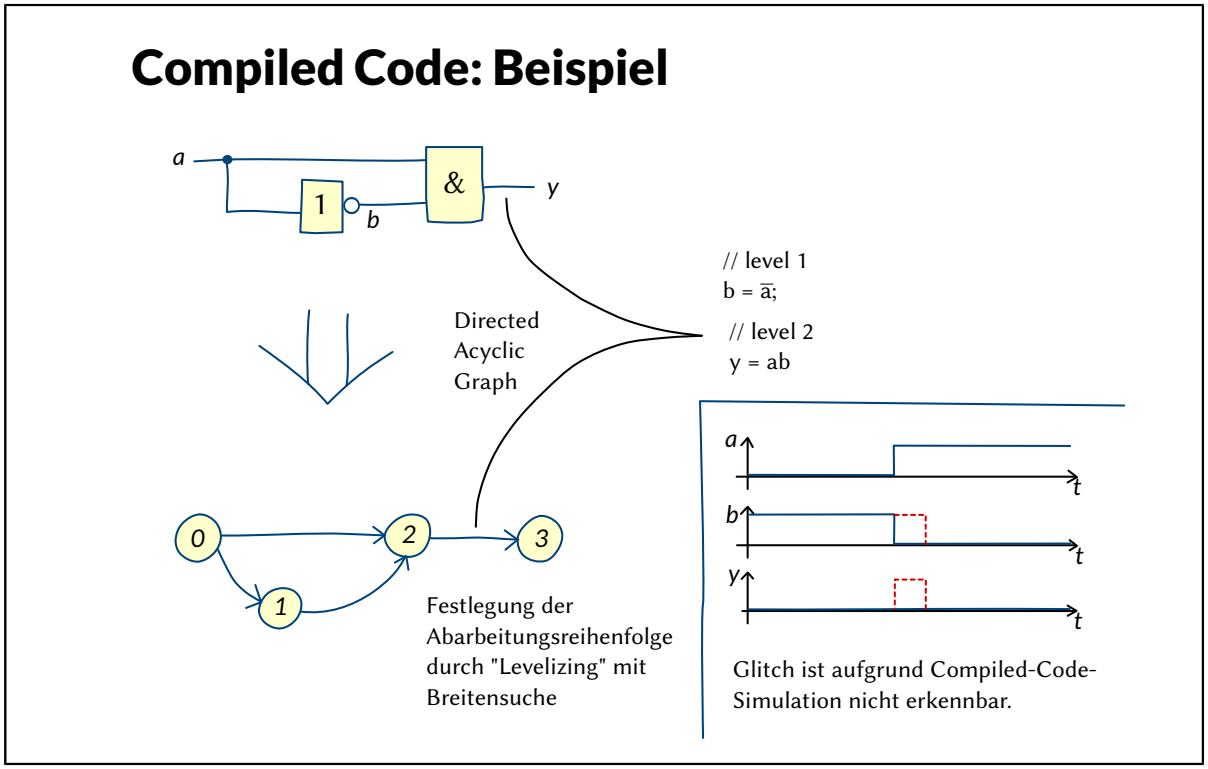
Um sicher zu stellen, dass kein logischer Wert eines Gatters verwendet wird, bevor er endgültig berechnet wurde, erfordert die Compiled-Code-Simulation eine Strukturierung der Schaltung in logische Ebenen (Levelizing). Auf diese Weise wird die (real parallel arbeitende) Schaltung in eine Sequenz von Rechnerschritten abgebildet.

Hierzu beschreibt man die Schaltung als gerichteten azyklischen Graphen (DAG), dessen Knoten die Gatter der Schaltung sowie die Ein- und Ausgänge repräsentieren. Jedem Knoten (K) wird eine nicht negative ganze Zahl level(K) zugewiesen, wobei gilt

- Für alle primären Eingänge Xi:level(Xi)=0
- Für alle anderen Knoten Ki:level(Ki)=1+max(level(Kj)), wobei Kj die Anfangsknoten aller bei Ki eingehenden Kanten bezeichnet.

Die sich nach dem Levelizing ergebende Sequenz, die in einer Programmiersprache beschrieben werden kann, lässt sich in einfacher Weise durch den Maschinencode eines beliebigen Prozessors ersetzen. Auf diese Weise erhält man einen schnellen Simulator, der jedoch auch rechnerabhängig ist. Im Falle der Hochsprache ist eine Übersetzung erforderlich. Dabei erzeugt ein Simulationsgenerator aus der Schaltungsbeschreibung den Simulator in einer Hochsprache. Ein gewöhnlicher Compiler übersetzt ihn in Maschinensprache. Erzeugt der Simulationsgenerator unmittelbar aus der Schaltungsbeschreibung Maschinencode, spricht man von einem "Native-Compiled"-Simulator.

Digitale Simulation: Compiled Code: Beispiel



Im ersten Beispiel wurde lediglich die Simulation eines Eingangsmusters beschrieben. Es ist jedoch leicht auf eine beliebige Folge von Eingangsmustern zu erweitern. Fügt man zusätzlich simulierte Register ein, können auch sequentielle Schaltungen simuliert werden.

Es soll nun gezeigt werden, dass ein solcher Simulator Timing-Probleme nicht erkennen kann. Man betrachtet dazu die im Bild dargestellte Schaltung. Aufgrund der logischen Gleichungen sollte ständig y=1 sein. Offenbar tritt aufgrund der Gatterverzögerung aber in der realen Schaltung ein statischer 0-Hazard nach Umschalten von x=0 auf x=1 auf, dessen Dauer der Verzögerungszeit des Inverters d entspricht.

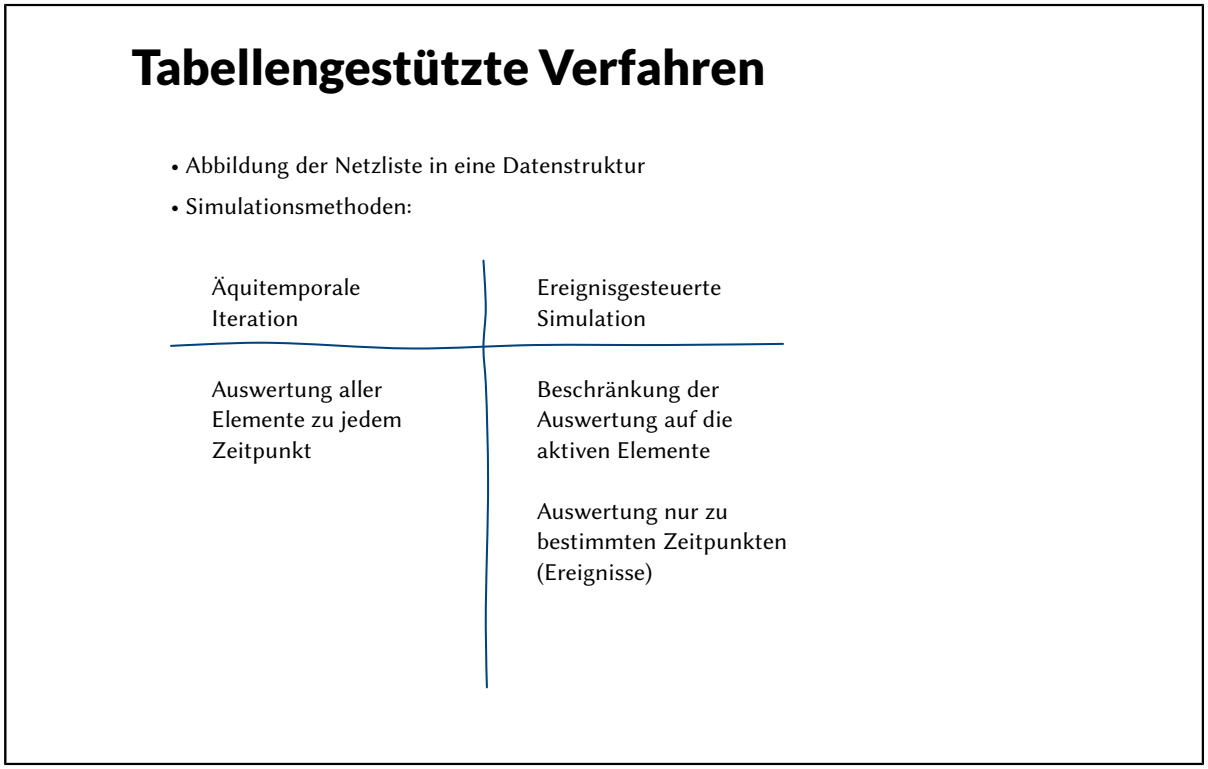
Simuliert man diese Schaltung mit dem obigen Verfahren, so wird als Reaktion auf einen neuen Eingabewert x zunächst a neu berechnet, bevor y aktualisiert wird. Damit wird der Pegeleinbruch vom Simulator aber nicht entdeckt. Der beschriebene Simulator simuliert lediglich das Verhalten einer verzögerungsfreien Schaltung. Er wird daher auch Zero-Delay-Simulator genannt. Für synchrone Schaltungen erzielt man mit einem solchen Simulator eine große Simulationsgeschwindigkeit. Für asynchrone Schaltungen kann man ihn nicht verwenden.

Compiled-Code-Simulation ist sehr effizient. Zwar wird ein zusätzlicher Aufwand für die Übersetzung benötigt, jedoch entsteht kein interpretativer Aufwand bei der eigentlichen Simulation. Compiled Code besitzt deshalb insbesondere für höhere Entwurfsebenen (z.B. RT-Level) große Bedeutung, da dort ein genaues Zeitmodell nicht erforderlich ist.

Die Simulationszeit von Compiled-Code-Simulatoren ist proportional zur Anzahl der Elemente. Sie kann durch verschiedene Maßnahmen verringert werden. Eine Methode besteht z.B. darin, unnötige Berechnungen zu vermeiden. So muss ein AND-Gatter nicht ausgewertet werden, wenn einer seiner Eingänge auf logisch 0 liegt. Die Länge des Codes steigt damit jedoch beträchtlich an.

Will man mehrwertige Logik benutzen oder stellen die Strukturelemente komplexere Funktionen dar, benutzt man i.a. Funktionstabellen oder Prozeduren für die einzelnen Elemente.

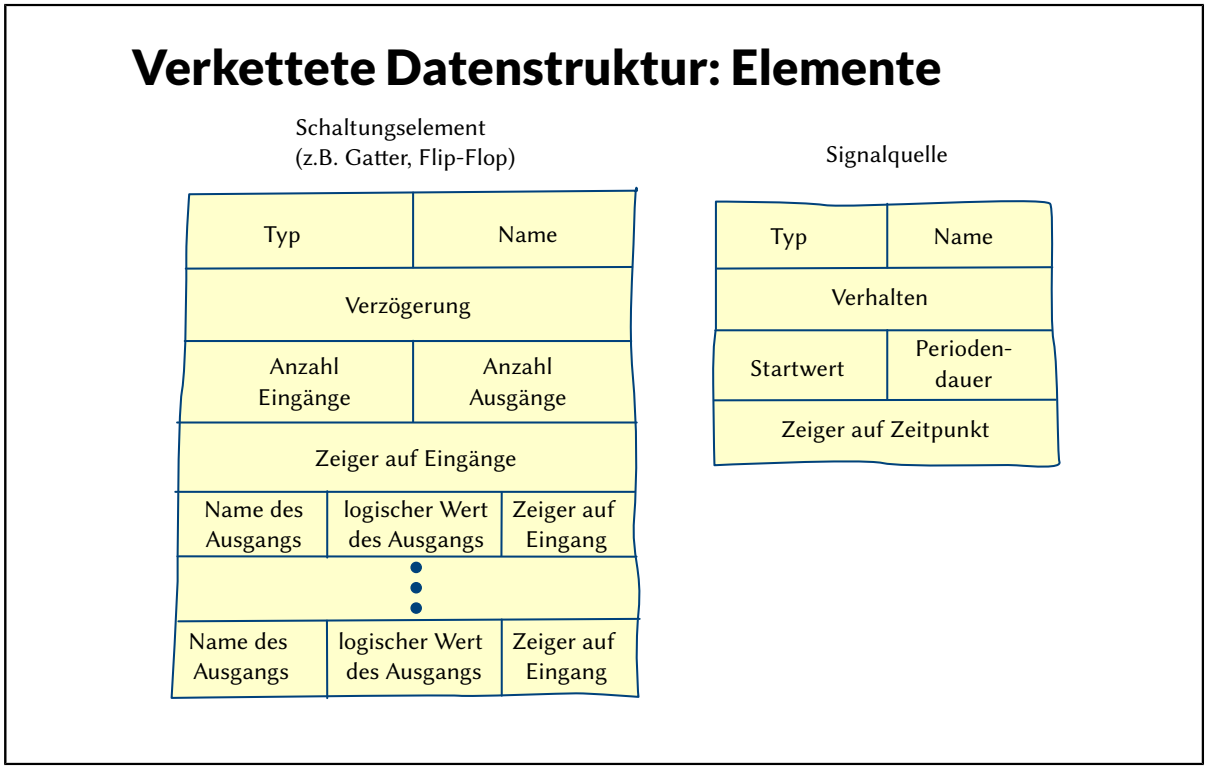
Digitale Simulation: Talbellengestützte Verfahren



Tabellengesteuerte Verfahren bilden die Schaltung, d.h. ihre Komponenten und Verbindungen in eine Datenstruktur ab. Dies können im Fall eines Unit-Delay-Simulators einfache Tabellen sein oder - für komplexe Delay-Modelle - spezielle Datenstrukturen, die vielfach mit Hilfe verketteter linearer Listen aufgebaut werden.

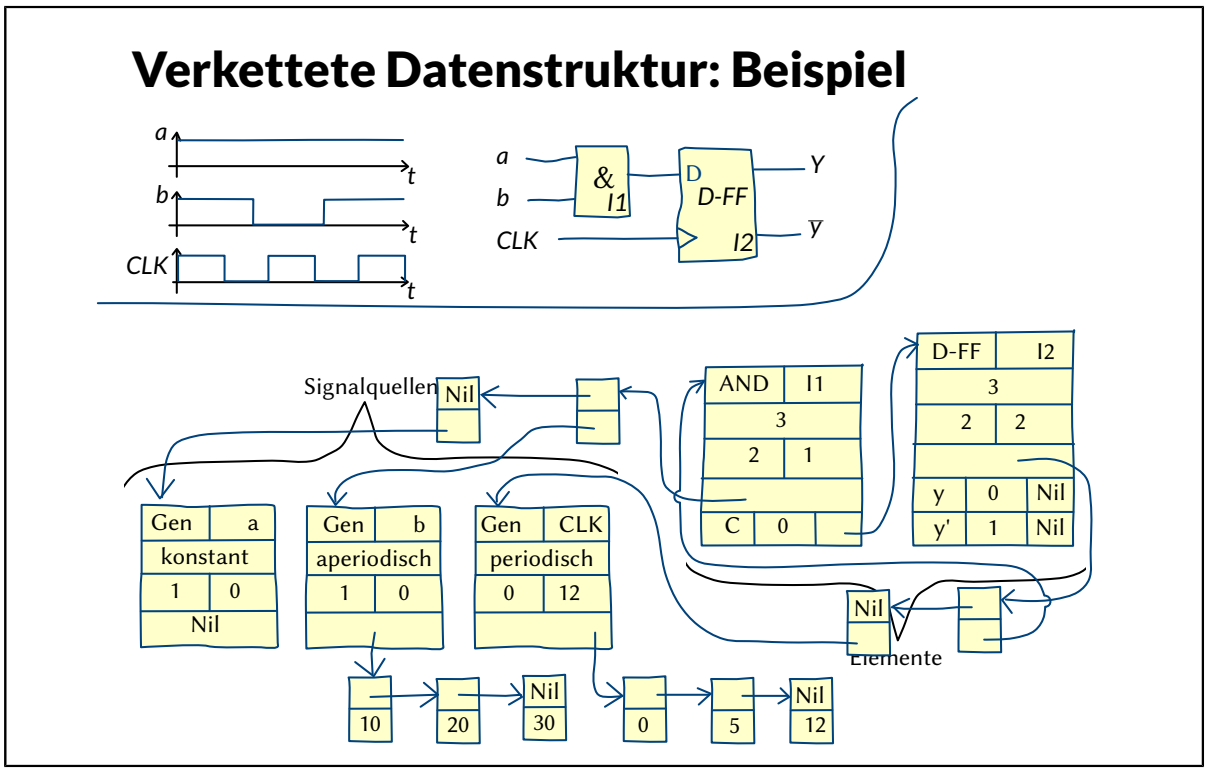
Bei der Bearbeitung dieser Datenstrukturen durch den Simulator wird zwischen der äquitemporalen Iteration und der so genannten ereignisgesteuerten Simulation unterschieden. Im ersten Fall macht der Simulator einen Zeitschritt und untersucht dann die logischen Zustände aller Schaltungselemente. Im zweiten Fall wird durch eine so genannte Ereignissteuerung dafür gesorgt, dass lediglich solche Elemente untersucht werden, an denen sich ein Eingangssignal verändert hat. Ereignissteuerung ist heute die Standardmethode bei der Simulation.

Digitale Simulation: Verkettete Datenstrukturen: Elemente



Elemente und Signale können durch geeignet aufgebaute Datensätze, wie hier beispielhaft gezeigt, beschrieben werden. Sie werden durch entsprechende Zeiger miteinander verkettet, so dass vollständige Schaltungen mitsamt der Stimuli beschrieben werden können.

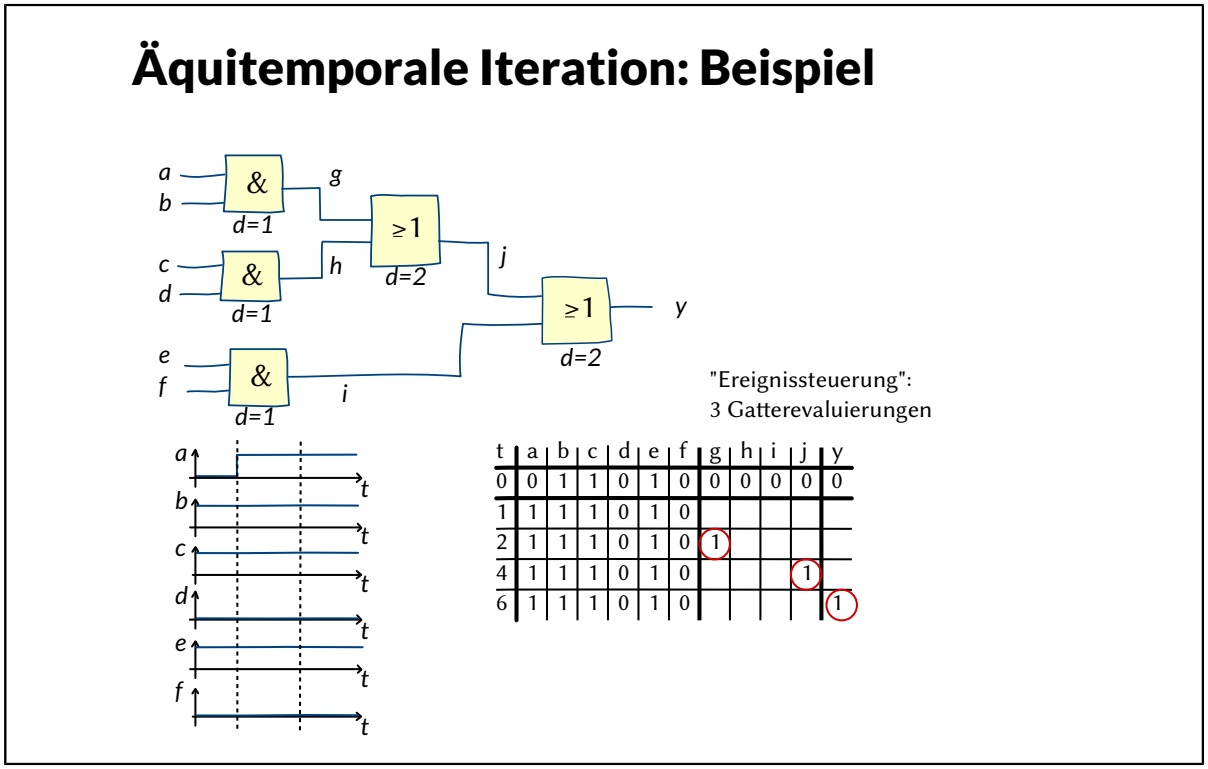
Digitale Simulation: Verkettete Datenstrukturen: Beispiel



Das Bild zeigt beispielhaft eine logische Schaltung, die aus einem D-Flip-Flop und einem AND-Gatter aufgebaut ist und mit einer aperiodischen Folge von Nullen und Einsen beaufschlagt wird. Das Flip-Flop wird mit einem unsymmetrischen Takt betrieben, der eine Taktperiode von 12 Zeiteinheiten aufweist. Um eine effiziente Bearbeitung zu ermöglichen, ist die Beschreibung mehrfach verkettet aufgebaut und enthält Redundanz. Alle Eingänge eines Elements sind jeweils in einer verketteten Liste zusammengefaßt. Die Ausgänge dagegen sind als Tabelle innerhalb der Elementbeschreibung aufgeführt.

Mit Hilfe eines geeigneten Verfahrens - das den eigentlichen Simulator darstellt - ist diese Datenstruktur entsprechend dem zeitlichen Ablauf zu bearbeiten. Dazu gibt es zwei unterschiedliche Vorgehensweisen.

Digitale Simulation: Äquitemporale Iteration

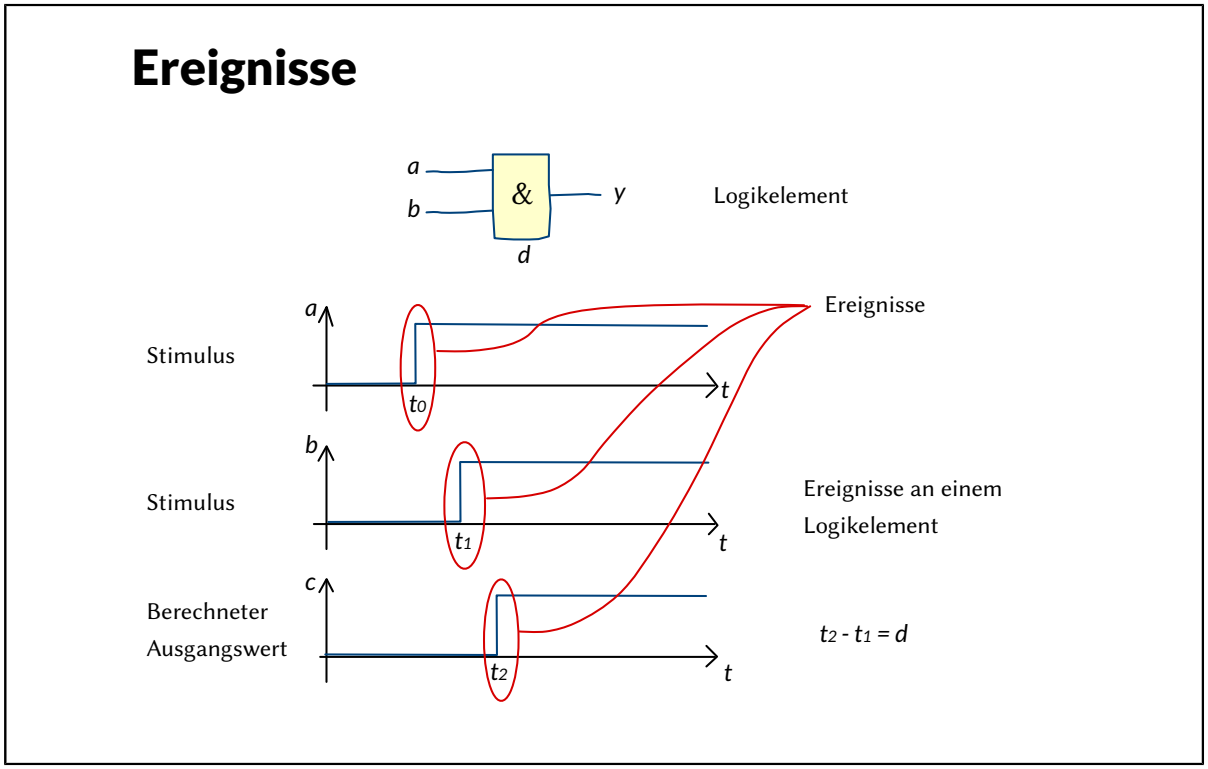


Bei der äquitemporalen Iteration wird für jeden Zeitpunkt die gesamte Schaltung untersucht, d.h. jedes Element wird ausgewertet. Anschließend wird die Zeit inkrementiert. Die Schrittweite kann sich ggf. von Iteration zu Iteration ändern. Sie ist jedoch stets für alle Schaltungselemente dieselbe.

Ein solches Verfahren ist leicht zu implementieren. Es ist jedoch sehr wenig effizient, da in einem Schritt normalerweise sehr viele Elemente ihren Zustand gar nicht ändern, sie also eigentlich gar nicht untersucht werden müssten. Auf Gatterebene liegt die Wahrscheinlichkeit für eine Änderung lediglich bei bis zu 10%. Die Effizienz steigt, wenn die Schaltungsaktivität zunimmt oder die Signalauflösung erhöht wird. Äquitemporale Iteration wird deshalb im Regelfall nur auf der RT-Ebene (grobes Zeitraster) oder auf der elektrischen Ebene (kontinuierliche Signalwerte) angewendet.

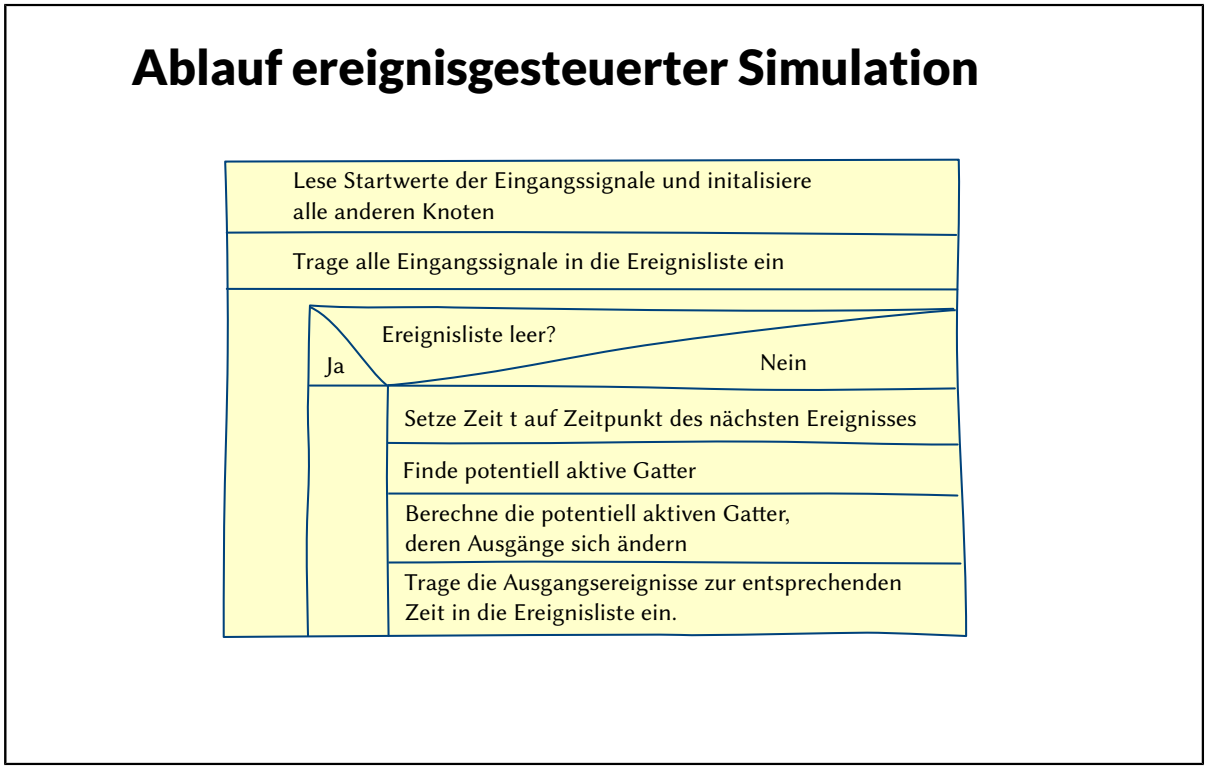
Das Beispiel zeigt eine Gatterschaltung, bei der sich zum Zeitpunkt t=1 ein Eingangssignal ändert. Bei den angenommenen Gatterverzögerungen sind dann 30 Gatterevaluierungen (5 Gatter in 6 Zeitschritten) erforderlich, um den Wert des Ausgangssignals zu bestimmen. Gelingt es, nur solche Gatter auszuwerten, bei denen sich das Eingangssignal tatsächlich geändert hat, kommt man mit 3 Gatterevaluierungen aus. Dies führt zum Konzept der Ereignissteuerung.

Digitale Simulation: Ereignisse



Ereignisgesteuerte Verfahren lösen das Effizienzproblem der äquitemporalen Iteration dadurch, dass sie die Berechnungen auf die notwendige Anzahl, d.h. die Berechnung der Gatter, bei denen sich mindenstens ein Eingangssignal ändert (Nichtidentität der Eingangssignale), beschränken. Immer, wenn sich Eingangssignale (Stimuli) oder der Ausgang eines Elements ändern, wird ein so genanntes Ereignis (Event) erzeugt. Nur Elemente, die mit diesem Netz verbunden sind, müssen bezüglich dieses Ereignisses untersucht werden. Diese Gatter heißen potentiell aktive Elemente. Auf diese Weise können - anders als beim Compiled-Code-Ansatz - kombinatorische, synchrone und asynchrone Schaltungen in gleicher Weise behandelt werden. Auch sind die verschiedensten Delay-Modelle implementierbar. Lediglich das Inertial-Delay-Modell führt zu Schwierigkeiten, da es erforderlich sein kann, ein aktiviertes Ereignis wieder zu deaktivieren.

Digitale Simulation: Ablauf ereignisgesteuerter Simulation



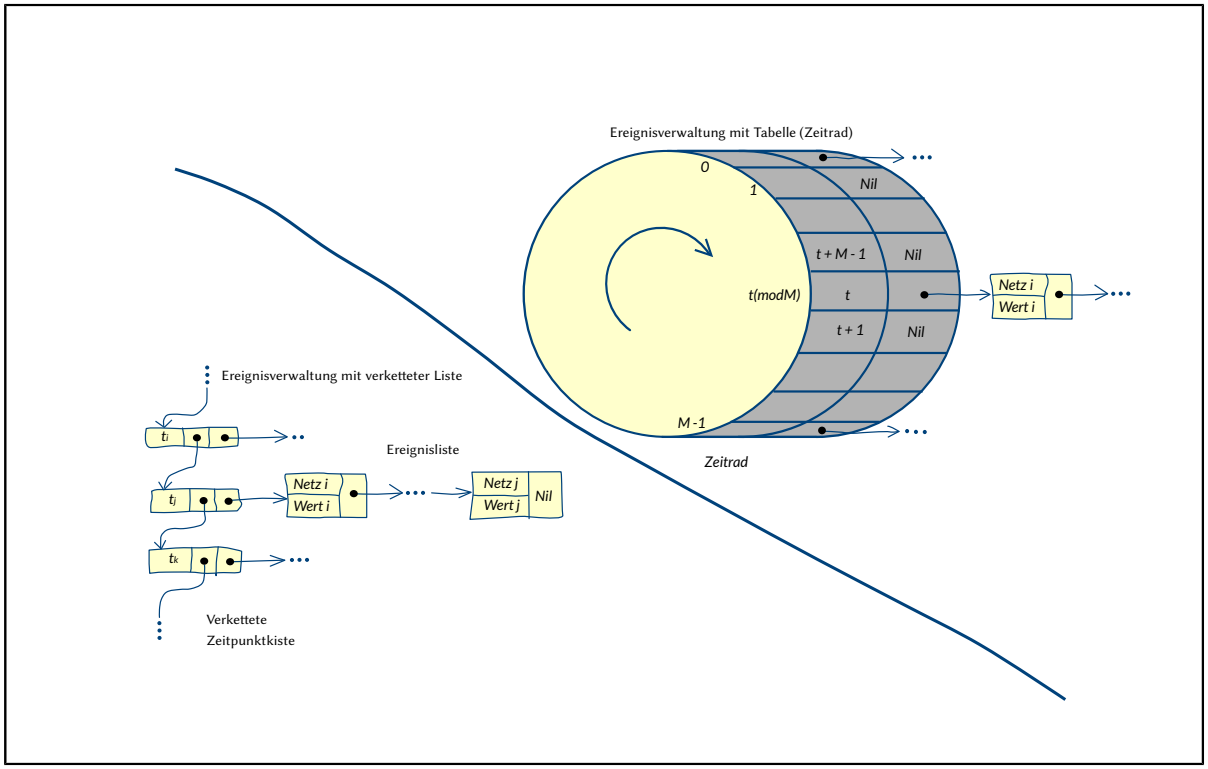
Mit den geschilderten Datenstrukturen ergibt sich der allgemeine Ablauf wie folgt: Zunächst werden alle Schaltungseingänge mit vorgegebenen Werten initialisiert. Alle anderen Netze werden auf X gesetzt. Das Zeitrad wird mit t0 initialisiert. Zu Beginn der Simulation sind alle mit den Schaltungseingängen verbundenen Elemente potentiell aktiv. Für jeden Zeitschritt des Zeitrads werden nun folgende Schritte ausgeführt:

Mit aktuellen Werten werden alle potentiell aktiven Elemente untersucht. Treten Änderungen an den Ausgängen auf, werden diese neuen Ereignisse in die Ereignisliste eingetragen. Die dort einzutragenden Zeitpunkte hängen von der aktuellen Zeit und den Verzögerungen der Elemente ab.

Diese Schleife wird bis zum Ende der Simulationszeit, das ist der Zeitpunkt, zu dem keine Ereignisse mehr vorhanden sind, durchlaufen.

Moderne Simulatoren nutzen die Geschwindigkeit der übersetzenden Verfahren und verbinden sie mit der Genauigkeit der Event-orientierten Verfahren. Deshalb werden zwar Event-Listen geführt, neue Zustandswerte aber durch übersetzten Code berechnet.

Digitale Simulation: Ereignisverwaltung



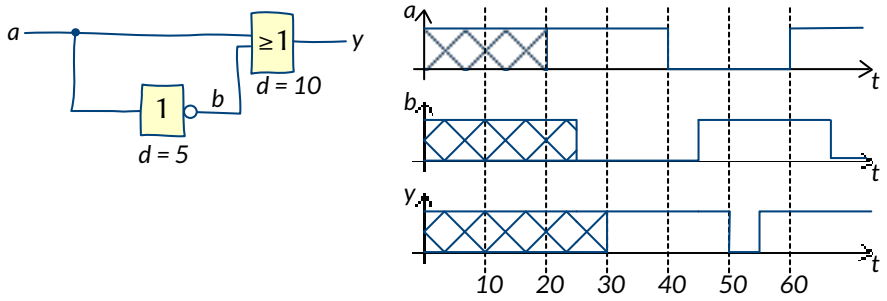
Mit der bereits beschriebenen Datenstruktur lassen sich in einfacher Weise die Eingangswerte für ein zu untersuchendes Element sowie die beim Auftreten eines Ereignisses potentiell aktiven Elemente finden. Eine weitere Datenstruktur wird zur Steuerung der Zeit und zur Verwaltung der Ereignisse benötigt. Wird ein Element K ausgewertet und festgestellt, dass sich sein Ausgangsnetz ändert, so muss die Änderung des Wertes dieses Netzes für einen Zeitpunkt t+d eingetragen werden, wobei t die aktuelle Simulationszeit und d die Verzögerung des betrachteten Elementes ist. Zur Verwaltung dieser Ereignisse lässt sich eine verkettete Liste verwenden.

In einer solchen Datenstruktur erfordert die Eintragung eines Ereignisses zum Zeitpunkt t das Durchsuchen der geordneten Zeitpunktliste, um die richtige Position für t zu finden. Die Suchzeit ist proportional zur Länge der Liste, die mit der Größe der Schaltung und ihrer Aktivität wächst. Häufig ist diese Liste sehr dicht belegt, d.h. dass die Differenzen zwischen zwei aufeinander folgenden Einträgen klein sind. Es liegt daher nahe, statt der Liste eine Tabelle (ein eindimensionales Feld) für die Zeitpunkte zu verwenden. Dann kann die Suche durch einen direkten Zugriff über den Index t ersetzt werden. Als Nachteil wirkt sich aus, dass Speicherplatz für Zeitpunkte ohne Ereignisse vergehalten werden muss. Der Speicherbedarf hierfür ist jedoch gering.

Dieses so genannte Zeitrad speichert Ereignisse zwischen der aktuellen Simulationszeit t und t+M-1, wobei M die Feldgröße darstellt. Alle Zeiten sind deshalb als t mod M zu deuten. Jedes Ereigniss für einen Zeitpunkt t+d mit t < M kann ohne Suche in das Zeitrad eingefügt werden. Ereignisse für Zeitpunkte, die über den im Zeitrad enthaltenen Bereich hinausgehen (t>(M-1)) werden in einer "Overflow"-Ereignisliste gespeichert. Einfügungen in diese Liste erfordern einen Suchvorgang, i.a. werden jedoch die meisten Ereignisse direkt im Zeitrad enthalten sein. Sobald der Zeitpunkt eines Ereignisses in den Bereich des Zeitrades fällt, sollte es deshalb dorthin übertragen werden.

Digitale Simulation: Ereignissteuerung: Beispiel

Ereignissteuerung: Beispiel



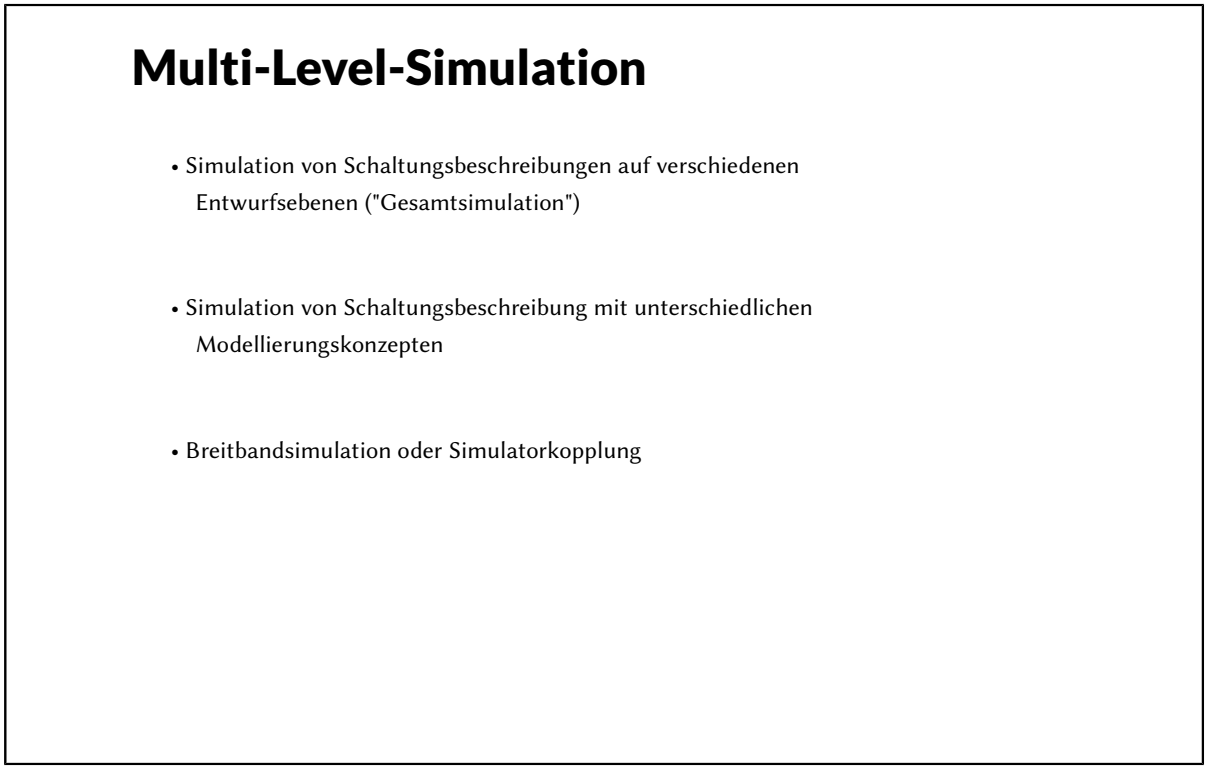
Zeit								
t	*0*	*20*	*25*	*30*	*40*	*45*	*50*	*55*
Signale								
a	*x*	*1*	*1*	*1*	*0*	*0*	*0*	*0*
b	*x*	*x*	*0*	*0*	*0*	*1*	*1*	*1*
y	*x*	*x*	*x*	*1*	*1*	*1*	*0*	*1*
Ereignisse								
1	{20, a=1}	{25, b=0}	{30, y=1}	{40, a=0}	{45, b=1}	{50, y=0}	{55, y=1}	.....
2	{40, a=0}	{30, y=1}	{40, a=0}	{60, a=1}	{50, y=0}	{55, y=1}	{60, a=1}	
3	{60, a=1}	{40, a=0}	{60, a=1}		{60, a=1}	{60, a=1}		
		...						

Das Bild zeigt den groben Ablauf einer ereignisgesteuerten Simulation für das bereits bei der Compiled-Code-Simulation betrachtete Schaltungsbeispiel. Im Zeitdiagramm ist ein Pegeleinbruch (Glitch) bei t=50 zu erkennen. Mit dem verwendeten Simulator können also - anders als beim einfachen Compiled-Code-Verfahren - mit Hilfe eines Transport-Delay-Zeitmodells Timing-Probleme solcher Art erkannt werden.

Ereignisgesteuerte Verfahren sind grundsätzlich für die meisten Verzögerungszeit- und Zustandsmodelle anwendbar. Für die Auswertung der Elemente sind Unterprogramm- oder Tabellentechniken (Wahrheitstabellen) gebräuchlich. Die Zeit für die Auswertung von Elementen bestimmt sehr wesentlich die Simulationsdauer, die Effizienz der Auswertung ist deshalb sehr wichtig.

Gegenüber der Compiled-Code-Simulation hat das ereignisgesteuerte Verfahren den Vorteil, dass in jedem Zeitschritt nur die potentiell aktiven Elemente ausgewertet werden. Der Nachteil liegt im entsprechenden Zeitbedarf für das Finden dieser Elemente und die Verwaltung der Ereignisse. Ereignisgesteuerte Simulatoren sind deshalb nur dann wesentlich schneller, wenn die Schaltungsaktivität gering ist.

Digitale Simulation: Multi-Level-Simulation



Wie bereits in der Einleitung zur Simulation besprochen, werden häufig Simulatoren benötigt, die in der Lage sind, mit Schaltungsbeschreibungen auf verschiedenen Entwurfsebenen umzugehen, insbesondere um eine Gesamtsimulation einer Schaltung durchführen zu können, wenn die Teile der Schaltung in unterschiedlicher Abstraktion bzw. mit unterschiedlichen Modellierungskonzepten beschrieben sind. Klassische Simulatoren dagegen erwarten eine Schaltungsbeschreibung auf genau einer Entwurfsebene.

Für eine Multi-Level-Simulation (wenn diese in einem Simulationslauf unterschiedlich beschriebene Teilschaltungen zulässt, spricht man auch von Mixed-Level-Simulation) können zwei unterschiedliche Ansätze verfolgt werden:

- Breitband-Simulator
- Simulatorkopplung

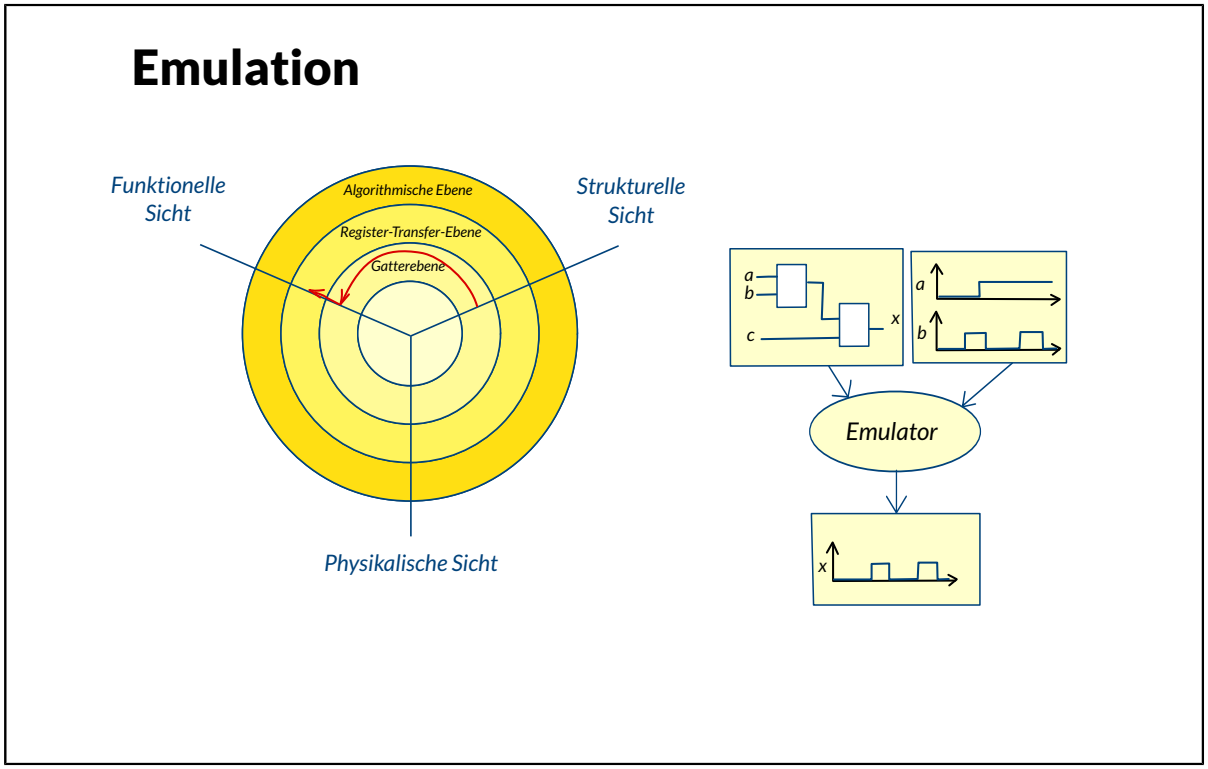
Für Schaltungen, die mit einer Hardwarebeschreibungssprache wie VHDL beschrieben sind, scheint ein Breitband-Simulator, d.h. ein Algorithmus, der alle Ebenen umspannt, der natürliche Ansatz zu sein. Solche Simulatoren werden auf der Basis ereignisgesteuerter Algorithmen realisiert. Da sehr unterschiedliche Modellierungskonzepte unterstützt werden müssen, sind Breitband-Simulatoren sehr komplex und deshalb für viele Anwendungen wenig effizient.

Electronic Design Automation (EDA)

Emulation

Emulation
Emulationskonzept
Schaltungsabbildung
Lookup-Tables (LUT)
CLB mit 2 Lookup-Tables (LUTs)
Rekonfigurierbare Netze
Partitionierung
Fiduccia-Mattheyses-Algorithmus
Anzahl der Pins
Eigenschaften der Emulation
Simulation/Emulation/ASIC
Emulationsbetriebsarten
Vector Debug Modus
In-Circuit-Emulation

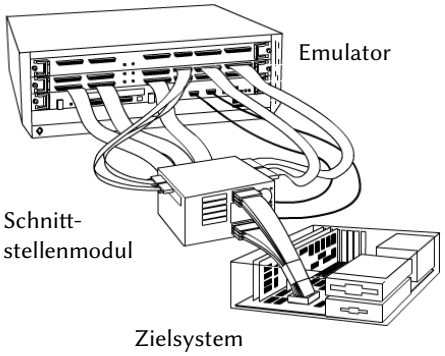
Emulation: Emulation



Moderne Schaltungen sind so komplex, dass man sie über sehr viele Taktschritte laufen lassen muss, um sie hinreichend zu simulieren. Bei einer Simulation auf Gatterebene könnte das Monate in Anspruch nehmen. Hinzu kommt die mangelnde Anbindung an die Peripherie. Jede Schaltung tauscht Daten mit ihrer Umgebung aus. Dazu muss nicht nur die Funktion der Schaltung für sich, sondern auch die Kommunikation mit ihrer Umgebung, der so genannten Peripherie, getestet werden. Fast immer ist die simulierte Schaltung aber zu langsam, um die Datenströme der Peripherie verarbeiten zu können und um schnell genug Steuersignale auszusenden. Um diese Nachteile zu umgehen, wendet man das Verfahren der Emulation an. Dabei wird das Verhalten einer Schaltung durch eine andere Schaltung nachgebildet. Das Verfahren ist also hardware-, nicht softwarebasiert. Dazu wird eine Emulator genannte Maschine verwendet, die statt der sonst üblichen festverdrahteten Schaltungen aus Einheiten rekonfigurierbarer Logik besteht. Der Emulator wird so konfiguriert, dass seine Funktion dem der zu emulierenden Schaltung entspricht; er verhält sich nach außen also wie die Schaltung selbst. Emulation findet grundsätzlich auf der Gatterebene statt, da die Grundelemente des Emulators einfache logische Funktionen realisieren.

Emulation: Emulationskonzept

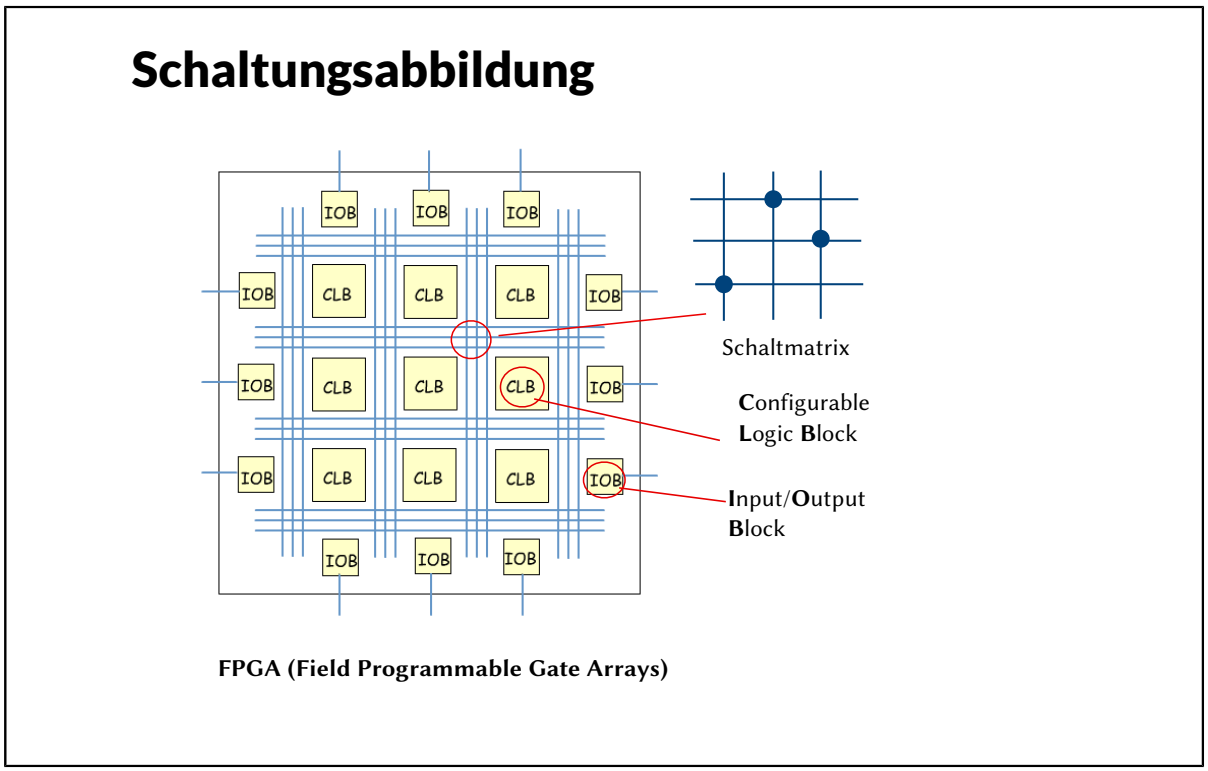
Emulationskonzept



- Virtueller Chip in programmierbarer Hardware
- In-System-Emulation möglich
- Ca. 1000fach schneller als Simulation
- Hoher Anschaffungspreis

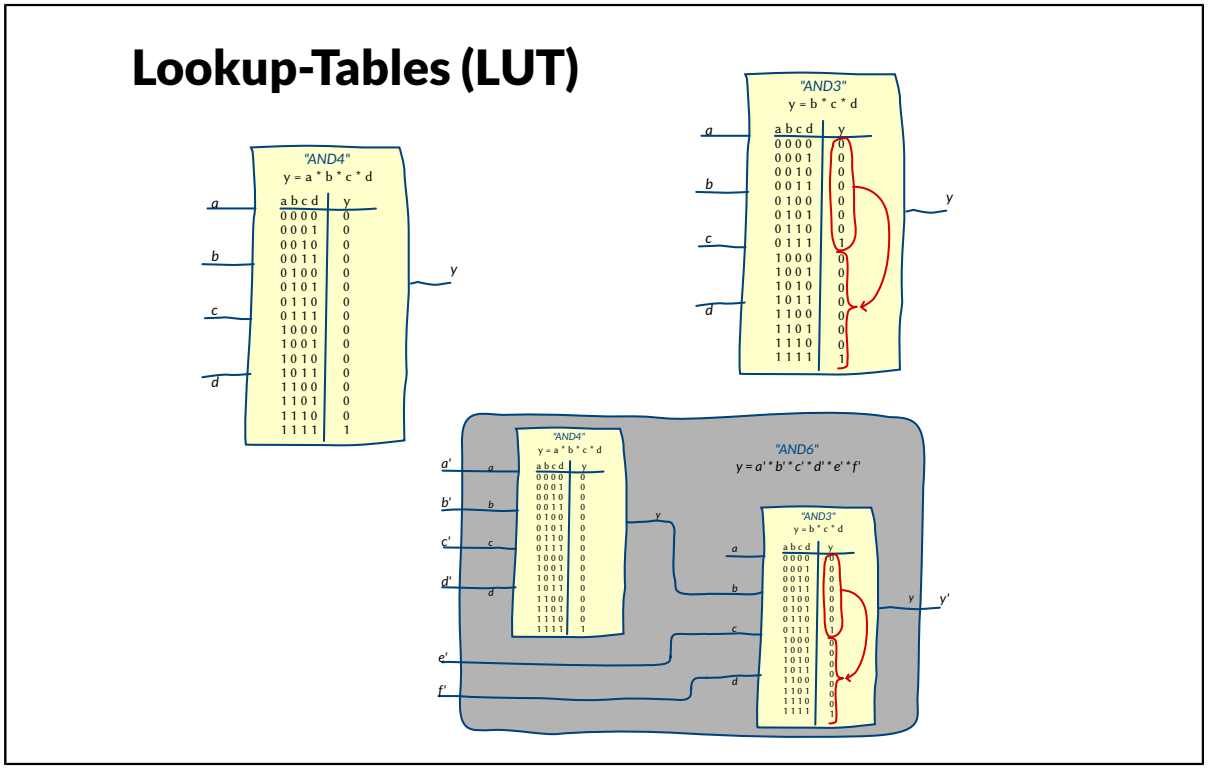
FPGAs (Field Programmable Gate Arrays) sind die Basisbauelemente für eine Emulation. Größere Emulationssysteme bestehen aus der Zusammenschaltung von Dutzenden von einzelnen FPGAs. Das funktionelle Modell des Chips wird durch Struktursynthese, Technologieabbildung und Partitionierung auf das Emulationssystem abgebildet. Es entsteht eine Art virtueller Chip in programmierter Hardware. Die Emulation einer Schaltung ist in der Regel langsamer als der fertige Chip, aber sie ist ca. eintausendmal schneller als eine Simulation. Die Geschwindigkeit reicht aus, um die meisten Chips schon vor ihrer Fertigung in ihrem späteren Zielsystem zu testen. Dies stellt einen enormen Vorteil gegenüber der Simulation dar, da das Zusammenwirken eines Chips mit dem Zielsystem in Software meist nur sehr schwer dargestellt werden kann. Ein Nachteil der Emulationssyteme ist der sehr hohe Anschaffungspreis von einigen 100.000 Euro bis hin zu mehreren Mio. Euro, da zudem wegen der steigenden Designkomplexität alle paar Jahre eine Neuanschaffung ansteht.

Emulation: Schaltungsabbildung



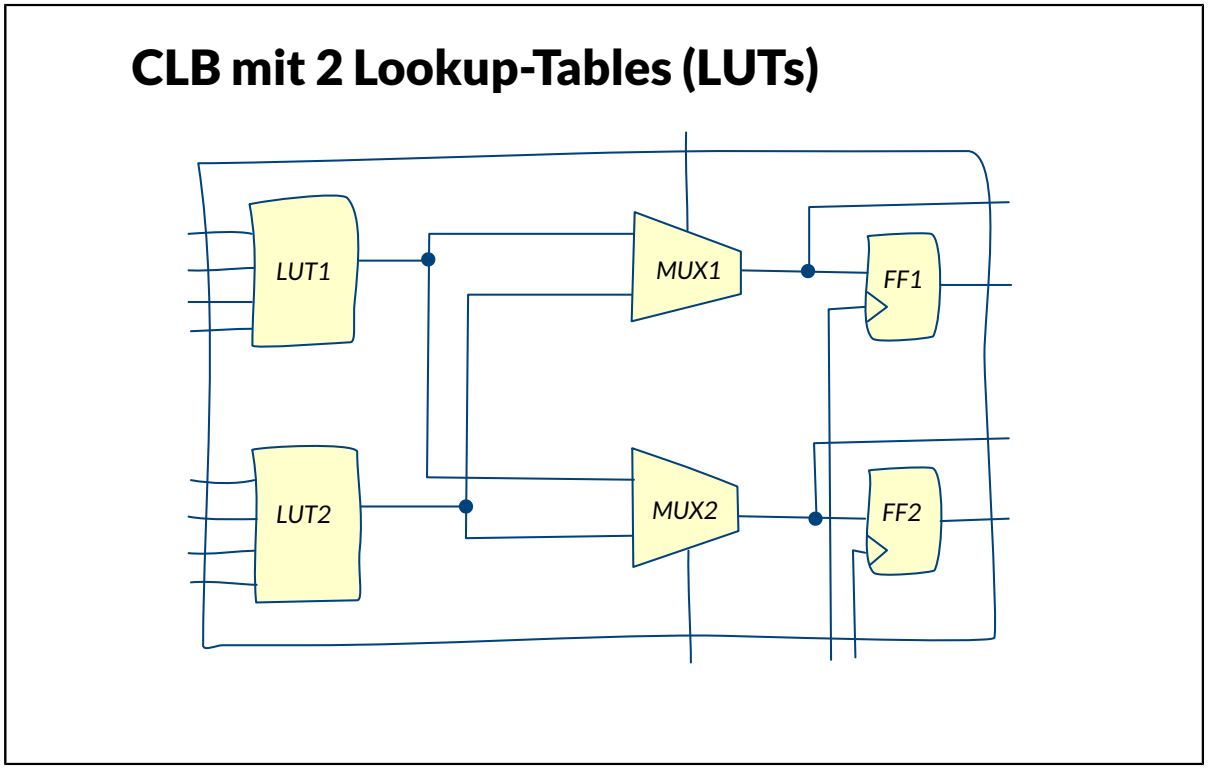
Ein Emulator besteht üblicherweise aus einer Matrix von so genannten FPGAs (Field Programmable Gate Arrays), die untereinander mit frei konfigurierbaren Leitungen über Schaltmatrizen verbunden werden können. Die FPGAs sind Bausteine, die rekonfigurierbare Logik enthalten. Eine Matrix so genannter CLBs (Configurable Logic Blocks) wird von einem Netz rekonfigurierbarer Leitungen durchzogen, mit denen die CLBs untereinander und mit den Ein- und Ausgangspins des FPGAs verbunden werden können. Die Abbildung einer Schaltung auf einen Emulator stellt eine Technologieabbildung dar. Ausgangspunkt ist die Gatterebene in funktionaler Sicht, in der noch keine Vorentscheidung über die zu verwendende Technologie getroffen ist. Der Mapping-Schritt bildet einzelne oder auch Gruppen von Logikgattern auf CLBs ab. CLBs werden in der Regel als sogenannte Lookup-Tables (LUT) realisiert.

Emulation: Lookup-Tables (LUT)



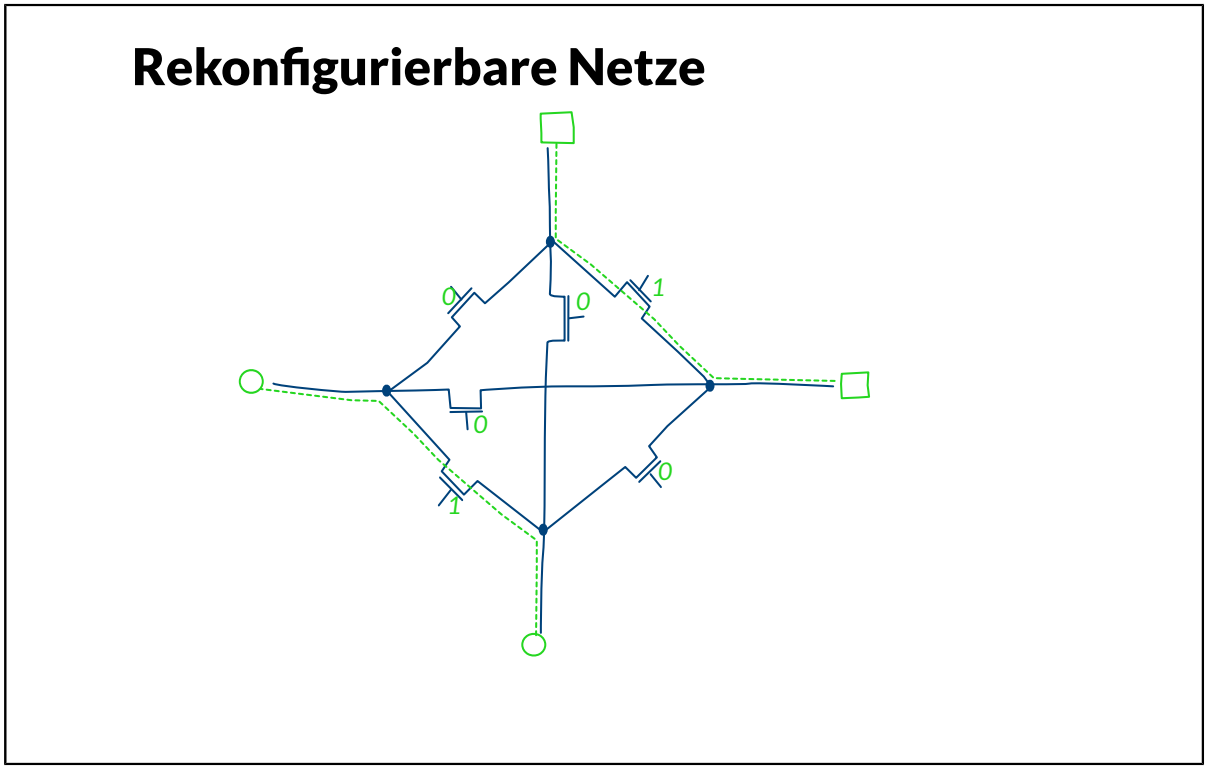
Eine LUT ist die Implementierung einer Wahrheitstabelle als Speicher. Betrachtet man die Eingänge der LUT als Adressleitungen des Speichers, kodiert jede Kombination von Eingangssignalen eine Speicheradresse. Am Ausgang der LUT liegt immer der Inhalt der ausgewählten Speicherzelle an. Dieser entspricht dem Wert der logischen Funktion für diese Eingangsbelegung. Durch Festlegen der Speicherinhalte lässt sich die logische Funktion, die die LUT modelliert, frei bestimmen. Um ein Logikgatter auf eine LUT abzubilden, wird die Wahrheitstabelle des Gatters auf die LUT übertragen. Hat das Logikgatter weniger oder gleich viele Eingänge wie die LUT, ist die Abbildung direkt möglich. Bleibt ein Eingang unbelegt, muss die Wahrheitstabelle doppelt in der LUT abgelegt werden, so dass das Ergebnis am Ausgang vom Wert des nicht belegten Signals unbeeinflusst bleibt. Sieht man die LUT als Speicher, halbiert sich der verwendete Adressraum. Hat ein Logikgatter mehr Eingänge als eine LUT, muss es auf mehrere LUTs aufgeteilt werden. Diese enthalten dann jeweils einen Teil der logischen Funktion des Gatters und bilden zusammengenommen die Funktion korrekt ab.

Emulation: CLB mit 2 Lookup-Tables (LUTs)



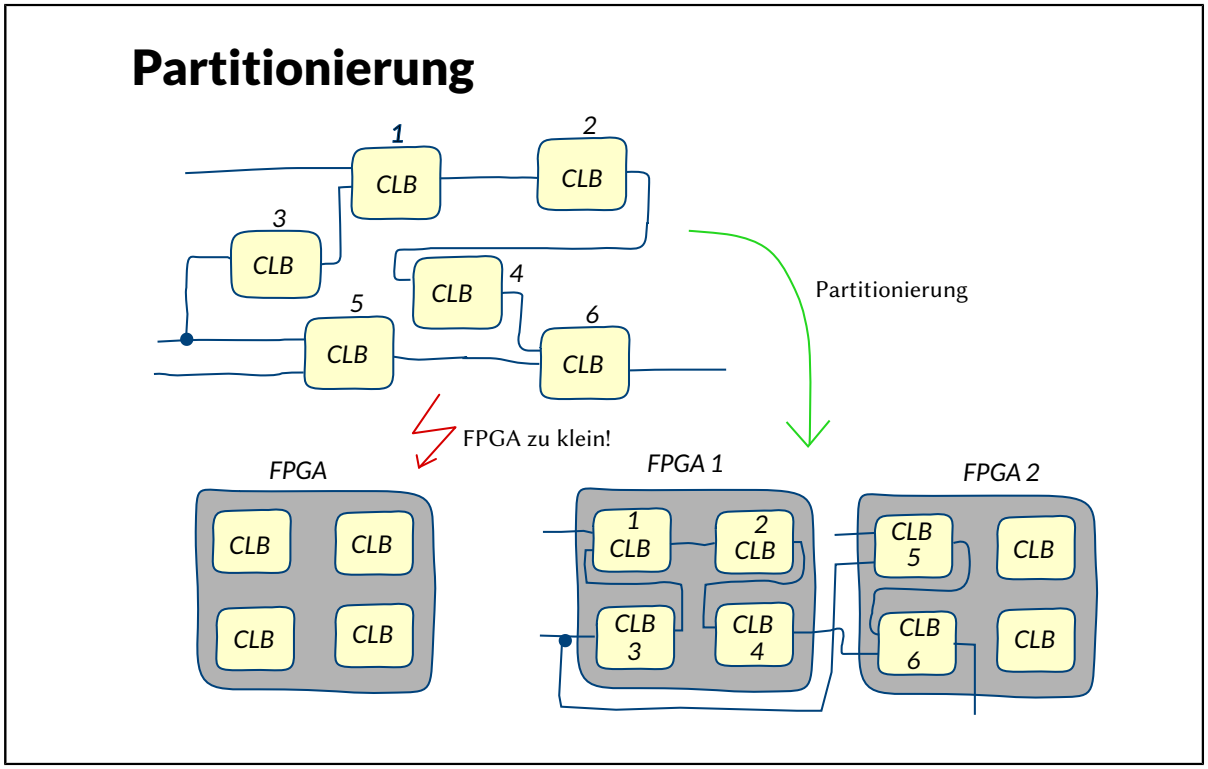
In der Praxis haben LUTs meist vier Eingänge und einen Ausgang, verfügen also über 16 Eingangssignalkombinationen. Neben einer oder oft auch zwei LUTs enthält ein CLB auch eine entsprechende Anzahl Flip-Flops. Diese sind notwendig, um sequentielle Logik modellieren zu können. Der Ausgang der LUTs kann dazu jeweils auf ein Flip-Flop geführt werden. Die Abbildung zeigt die Ausführung eines in der Praxis gebräuchlichen CLBs mit zwei LUTs, zwei Flip-Flops und der sie verbindenden Logik.

Emulation: Rekonfigurierbare Netze



Um die rekonfigurierbaren Logikblöcke untereinander verbinden zu können, sind rekonfigurierbare Netze notwendig. Dazu werden Verdrahtungskanäle zwischen den CLBs vorgesehen, in die Bündel von Signalleitungen gelegt werden. An den Stellen, wo Verbindungen zu den Ein- und Ausgängen der am Kanal gelegenen CLBs geschaffen werden müssen und vor allem an den Kreuzungspunkten der senkrecht zueinander verlaufenden Kanäle befinden sich Schaltmatrizen, die eine freie Konfiguration der Verbindungen ermöglichen. An jeder Stelle, wo zwei Leitungen sich kreuzen, werden so genannte Passtransistoren zwischen den Leitungen angebracht. Zudem werden die Leitungen selbst auch jeweils mit einem Passtransistor unterbrochen. Dadurch treffen sich vier getrennte Leitungen miteinander, von denen jede mit jeder über einen Passtransistor verbunden werden kann. Auf die Art können Überkreuzungen, Abzweigungen und Richtungswechsel realisiert werden. Die Konfiguration eines Knotenpunkts wird dadurch vorgenommen, dass man festlegt, welche Passtransistoren sperren und welche ein Signal durchschalten sollen. Jedes Gate eines Passtransistors ist mit einem Latch verbunden, das den Konfigurationswert speichert. Durch die Passtransistoren werden Leitungen von einem CLB zu einem anderen aus mehreren Leitungsteilen zusammengesetzt. Der Anzahl an Leitungen, die in einem Leitungskanal untergebracht werden können, sind jedoch Grenzen gesetzt. Es kommt daher in der Praxis vor, dass nicht alle CLBs eines FPGAs benutzt werden können, da bereits alle Leitungsresourcen für die Verbindungen anderer CLBs aufgebraucht wurden. Meist sind solche Engpässe nur auf einen Bereich eines FPGAs beschränkt. Analog zum Straßenverkehr spricht man auch von einer "Signal Congestion" (congestion (engl.) = Stau). Dann lässt sich Abhilfe meist dadurch schaffen, dass man die zu implementierenden Funktionen anders auf die CLBs im FPGA verteilt und damit die Leitungsdichte in dem Bereich verringert.

Emulation: Partitionierung



In vielen Fällen ist eine Schaltung zu groß, um sie vollständig auf ein einzelnes FPGA abbilden zu können. Dafür kann es mehrere Gründe geben: 1.Es gibt nicht genug CLBs im FPGA, um die Logik abzubilden. 2.Es sind mehr Leitungen zur Verbindung der CLBs nötig, als das FPGA insgesamt oder in einem bestimmten Bereich hat. 3.Die abzubildende Schaltung erfordert mehr Ein- und Ausgangspins, als das FPGA zur Verfügung stellen kann. In solchen Fällen muss die Schaltung auf mehrere FPGAs aufgeteilt werden. Dieser Vorgang heißt Partitionierung. Dabei sind vor allem zwei Randbedingungen zu beachten: die Länge des kritischen Pfads und die Anzahl der Ein- und Ausgänge. Jedes Schaltungselement und jede Leitung verzögern die Signale, die sie durchlaufen. Besonders groß ist die zusätzliche Verzögerung, wenn das Signal von einem FPGA zu einem anderen geführt wird. Die Leitungen außerhalb der FPGAs sind im Vergleich zu den internen Leitungen sehr lang und haben eine sehr große Kapazität. Beides erhöht die Verzögerungszeiten. Es muss deshalb darauf geachtet werden, dass der kritische Pfad und andere Pfade mit ähnlich großer Verzögerungszeit möglichst nicht über mehrere FPGAs führen, da sonst durch die zusätzliche Verzögerung die mögliche Taktfrequenz der Schaltung deutlich herabgesetzt wird.

Emulation: Fiduccia-Mattheyses-Algorithmus

Fiduccia-Mattheyses-Algorithmus

```
FM-Partitionierung(G(V,E), max_passes, tol):
   P ⟵ balancierte Zufallspartition von V
   wiederhole max_passes mal:
      gain(v) ⟵ ext(v) – int(v)   für alle v
      locked ⟵ ∅;  moves ⟵ [];  cum ⟵ 0;  best ⟵ 0;  best_i ⟵ –1

      solange es freie Knoten gibt:
         v ⟵ freier Knoten mit max gain(v), der die Balance einhält
         wenn keiner existiert: abbrechen

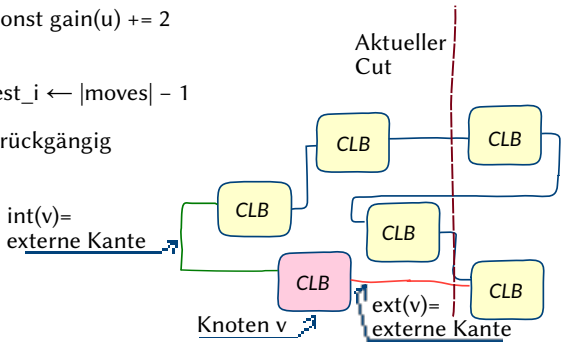
         verschiebe v in die andere Partition; locked ⟵ locked ∪ {v}
         für freie Nachbarn u von v:
            gain(u) –= 2 wenn P[u] = P[v], sonst gain(u) += 2

         cum += gain(v);  moves.add(v)
         wenn cum > best: best ⟵ cum;  best_i ⟵ |moves| – 1

      mache alle Moves nach Index best_i rückgängig
      wenn best = 0: abbrechen

   gib P zurück
```

Laufzeit linear in |E| Kanten und k passes :
O(k * |E|), aber heuristisches Ergebnis



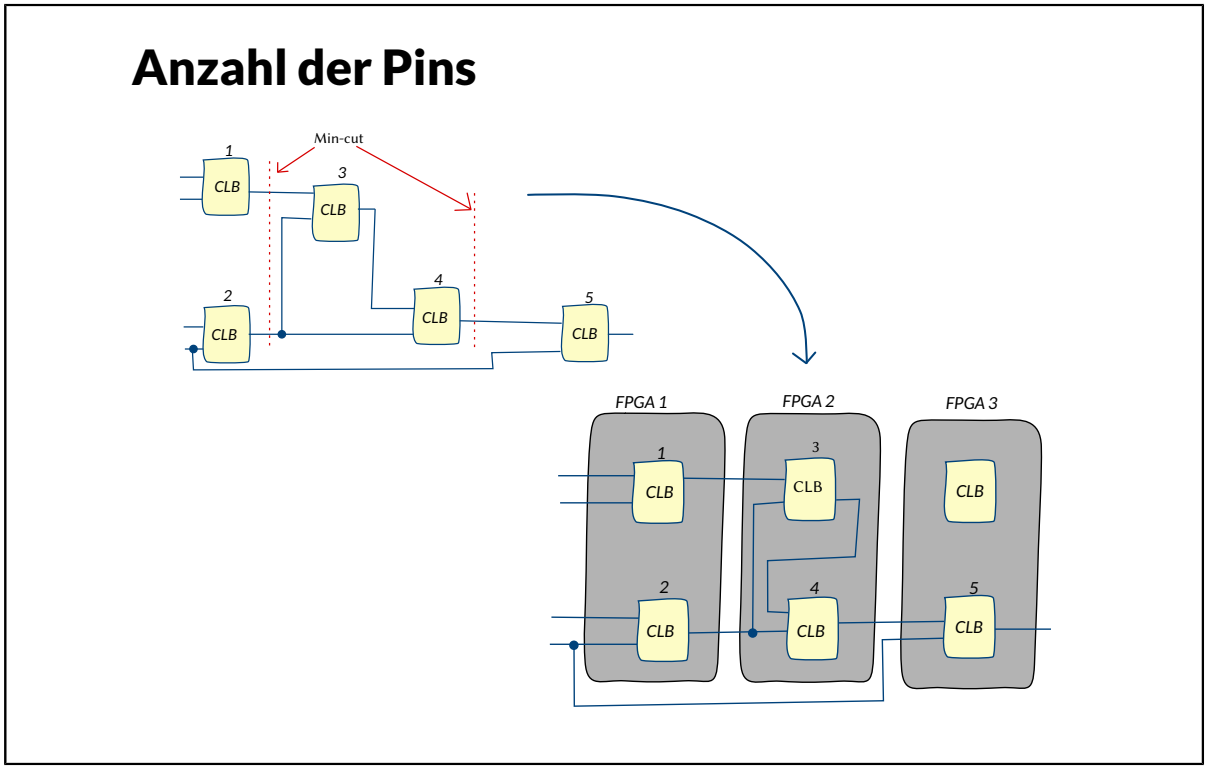
Das Partitionierungsproblem ist an sich sehr schwer zugänglich (NP-Hart), da es sehr viele Möglichkeiten der Zuordnung der Knoten zu den Partitionen gibt. Allerdings können wir es heuristisch effizient lösen. Dazu betrachten wir ein einfacheres Problem der balanzierten Bi-Partitionierung eines Graphen. Balanziert meint, das die Partitionen des Ergebnisses annähernd gleich groß sein müssen. Die Kanten seien einfach und haben alle das Gewicht 1. Die zusätzlichen Bedingungen wie kritischer Pfad in einer Partition und Sodnerbehandlung von Ein- und Ausgängen sowie Netze (Kanten) mit mehr als 2 angeschlossenen Knoten können dem Problem hinzugefügt werden, ohne das Problem stark zu beeinflussen. Des weiteren können wir auch mehr als 2 Partitionen für mehr als 2 FPGAs durch rekursive BiPartitionierung erreichen.

Für die balanzierte Bi-Partitionierung gibt es eine guten Algorithmus, den Fiduccia Matheyses Algorithmus, der von einer zufälligen Startpartitionierung ausgeht und dann in mehreren Durchläufen jeweils eine Liste von gains bestimmt, die aussagt, was es bringen würde, einen Knoten in die andere Partition zu verschieben. Diese Liste wird absteigend abgearbeitet und jeweils immer wieder upgedated, so dass jeder Knoten einmal verschoben wurde. Letztlich merkt sich der Algorithmus dann die beste Lösung jedes der ca. 10 Durchläufe und gibt diese zurück.

Es handelt sich um einen heuristischen Algorithmus mit keiner Garantie auf ein Optimum. Diese Eigenschaft ist für unser Problem akzeptabel denn letzlich müssen die Partitionen in die FPGAs passen und es müssen nicht zu viele Kanten für die IOs beschnitten werden. Wenn es ein paar mehr als das Optimum sind stört das nicht weiter.

Der Fiduccia-Mattheyses-Algorithmus hat eine nahezu lineare Laufzeit in der Kantenzahl des Graphen. Was deutlich besser ist als der Vorgänger, der Kerninghan-Lin-Algorithmus, der eine Laufzeit von O(n^2 * log(n)) für n Knoten aufweist. Moderne Partitionierungsalgorithmen arbeiten teilweise konstruktiv durch Lösen eines Eigenwert Problems auf eine modifizierten Adjazenzmatrix.

Emulation: Anzahl der Pins



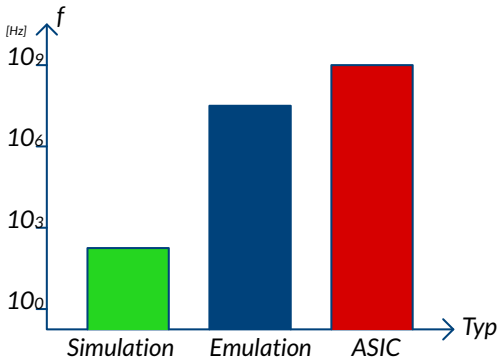
Ein FPGA hat nur eine begrenzte Anzahl an Pins, über die Eingangs- und Ausgangssignale geführt werden können. Wenn man eine Schaltung auftrennt, um sie auf mehrere FPGAs abzubilden, braucht man für jedes Signal, das von einem FPGA zu einem anderen führt, jeweils einen Pin an beiden FPGAs. Trennt man die Schaltung nun an einer ungünstigen Stelle auf, kann es passieren, dass man wesentlich mehr Signale aus einem FPGA zu- und abführen will, als dieses Pins hat. Dann kann die gewählte Partitionierung nicht durchgeführt werden. Aus diesem Grund sucht man bei der Aufteilung einer Schaltung nach dem so genannten Min-Cut. Das ist die Trennung einer Schaltung in zwei Teile, die die wenigsten Verbindungen zwischen den beiden Teilen aufweist.

Emulation: Eigenschaften der Emulation

Eigenschaften der Emulation

+ Emulation ist schneller als Simulation, aber langsamer als die gefertigte Schaltung (ASIC).

– Es gibt struktuelle Unterschiede zwischen der Abbildung auf FPGAs bzw. auf ASICs (z.B. bei der Behandlung von Tri-State Netzen).



– Das Timing der emulierten Schaltung ist anders als das der entworfenen Schaltung (z.B. können Glitches nicht detektiert werden).

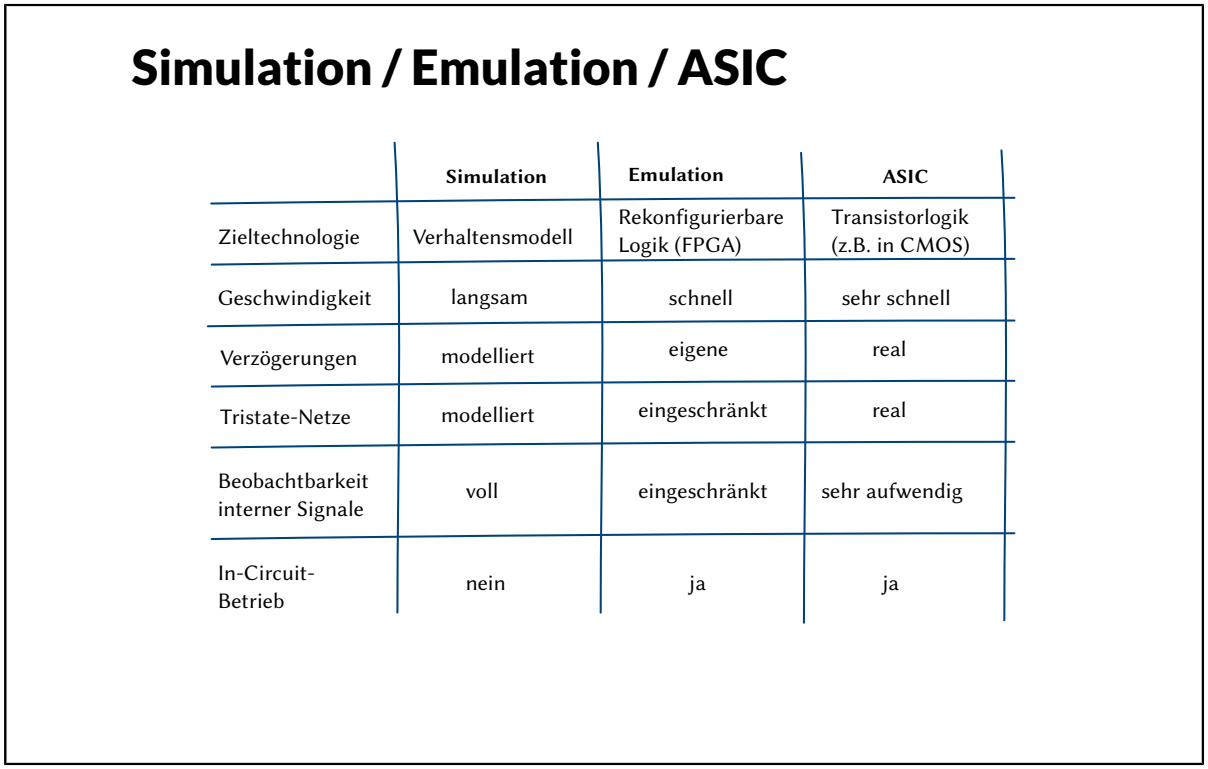
– Caches sind schlecht auf LUTs abzubilden. Extra Strukturen oder eigene Cache-Bausteine werden bebötigt.

● Emulation ist eine rein funktionelle Verifikationsmethode.

Die Emulation bietet offenkundig einige Vorteile gegenüber der Simulation. Trotzdem existieren beide nebeneinander, denn die Emulation hat auch einige Nachteile. - Emulation ist schneller als Simulation, aber langsamer als die gefertigte Schaltung. - Das Timing der emulierten Schaltung ist anders als das der entworfenen Schaltung (z.B.können Glitches nicht detektiert werden). - Es gibt strukturelle Unterschiede bei der Abbildung auf FPGAs bzw. auf ASICs (z.B. bei der Behandlung von Tri-State-Netzen) . - Emulation ist eine rein funktionelle Verifikationsmethode. Während Simulatoren heute mit Taktraten zwischen einigen Hertz und wenigen Kilohertz laufen, werden Emulatoren üblicherweise im Bereich von einem bis zu zweihundert Megahertz betrieben. Trotzdem sind sie immer noch langsamer als die gefertigte Schaltung. Das liegt an den zusätzlichen Verzögerungszeiten, die ein Emulator durch die rekonfigurierbare Logik hat. Dazu tragen sowohl die rekonfigurierbaren Netze mit ihren Passtransistoren als auch die CLBs mit ihren Konfigurationselementen bei. Besonders groß werden die Verzögerungszeiten, wenn die Schaltung zusätzlich auf mehrere FPGAs partitioniert werden muss. Um eine Schaltung schon vor der Fertigung verifizieren zu können, ist es wichtig, ihr späteres Zeitverhalten zu kennen. Dazu müssen die Verzögerungszeiten der einzelnen Gatter und nach Möglichkeit auch der Signalleitungen bekannt sein. Dann kann in der Schaltungssimulation die Verzögerung durch die einzelnen Schaltungskomponenten berücksichtigt und das zeitliche Verhalten der Schaltung realistisch nachgebildet werden. In der Emulation ist dies höchstens sehr eingeschränkt möglich. Da die Schaltung direkt auf die Hardware des Emulators abgebildet wird, arbeitet sie auch mit deren Verzögerungen. Ein CLB hat aber nur in Ausnahmefällen die gleiche Verzögerung wie das Gatter, was darauf abgebildet wurde, in einem ASIC. Genauso verhält es sich mit den Leitungen. Eine Emulation wird sich bei richtiger Wahl der Taktrate taktgenau verhalten. Effekte, die in ihrer Größenordnung darunter liegen, können nicht von einer Emulation abgeleitet werden. Dazu gehören zum Beispiel Glitches. FPGAs und damit die aus ihnen aufgebauten Emulatoren sind darauf ausgelegt, binäre Logik abzubilden. Das führt zu mehreren Einschränkungen: Heutige Schaltungen enthalten häufig so genannte dreiwertige Logik (engl. Tristate-Logic). Diese Art von Logik ist insbesondere für den Aufbau von Bussystemen geeignet. An einem solchen Bus sind die Ausgänge von mehr als einem Gatter angeschlossen. Normalerweise würden diese Ausgänge häufig verschiedene logische Pegel treiben und damit elektrische Kurzschlüsse verursachen. Die Tristate-Logik schafft hier Abhilfe. Nur ein Gatter treibt das Tristate-Netz, alle anderen sind im hochohmigen Zustand, wodurch der Signalpegel auf dem Netz eindeutig bleibt. Eine solche Anordnung stellt für einen Emulator allerdings ein Problem dar. Lookup-Tables müssen immer einen binären Wert

an ihrem Ausgang treiben. Multiplexer müssen immer eines ihrer Einganssignale zum Ausgang durchleiten. Eine direkte Abbildung von Tristate-Logik auf Emulatoren ist also nicht möglich. Um dieses Problem zu umgehen, verfügen einige FPGAs über zusätzliche Tristate-Logik-Resourcen. Damit ist eine direkte Abbildung möglich, jedoch wird der Abbildungsvorgang komplizierter, da das Abbildungswerkzeug die verschiedenen Logiktypen verwalten und verknüpfen muss. Eine andere Möglichkeit besteht darin, die Tristate-Logik in binäre Logik zu transformieren. Das ist dann möglich, wenn man einen Defaultwert vorgibt, den ein Tristate-Netz haben soll, wenn es von gar keinem Gatterausgang getrieben wird. Die resultierende binäre Logik ist umfangreicher als die ursprüngliche dreiwertige, lässt sich aber auch auf gewöhnliche FPGAs abbilden . Für diesen Fall wird besonders deutlich, dass eine Schaltung auf einem Emulator mit einer gleichwertigen als ASIC gefertigten Schaltung nichts weiter gemein hat als die gleiche Funktion. Durch Einführung zusätzlicher logischer Pegel zwischen "high" und "low" können in der Schaltungssimulation die Auswirkungen von nicht-digitalen Effekten wie unterschiedliche Treiberstärken modelliert werden. Bei Verwendung eines Emulators ist das nicht möglich.

Emulation: Simulation/Emulation/ASIC

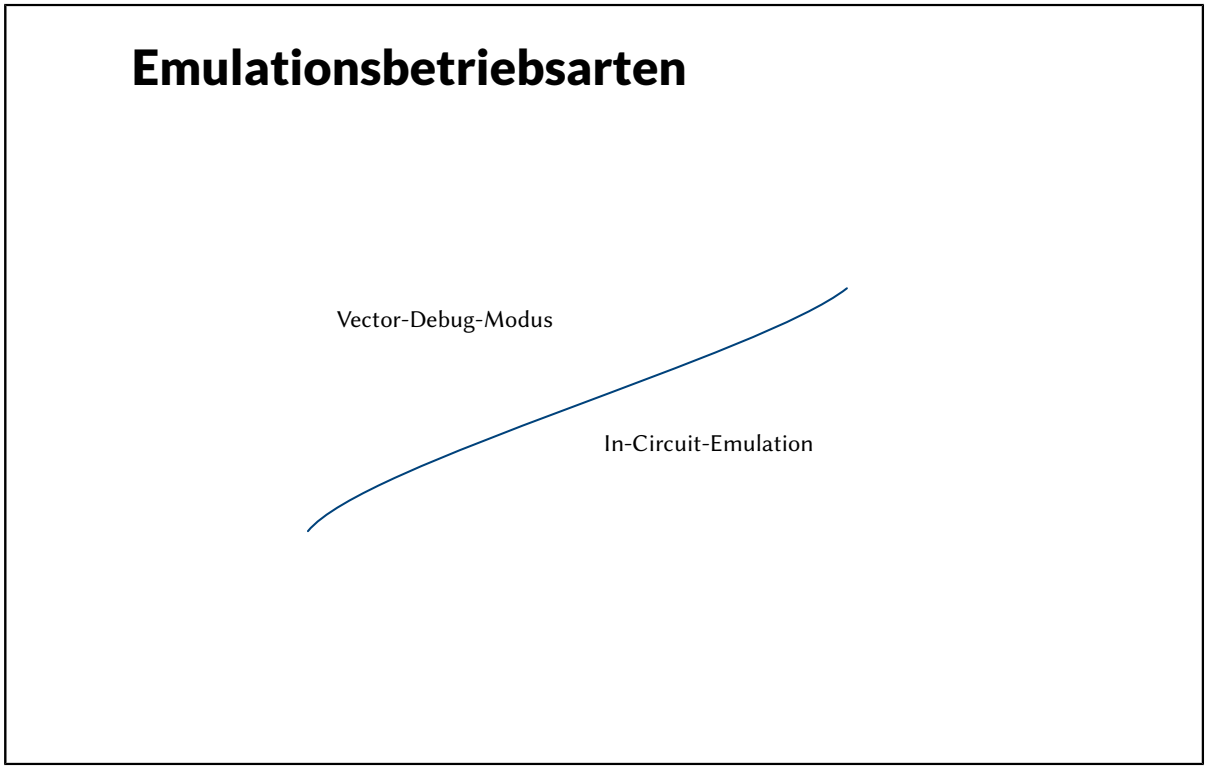


Simulation / Emulation / ASIC

	Simulation	Emulation	ASIC
Zieltechnologie	Verhaltensmodell	Rekonfigurierbare Logik (FPGA)	Transistorlogik (z.B. in CMOS)
Geschwindigkeit	langsam	schnell	sehr schnell
Verzögerungen	modelliert	eigene	real
Tristate-Netze	modelliert	eingeschränkt	real
Beobachtbarkeit interner Signale	voll	eingeschränkt	sehr aufwendig
In-Circuit-Betrieb	nein	ja	ja

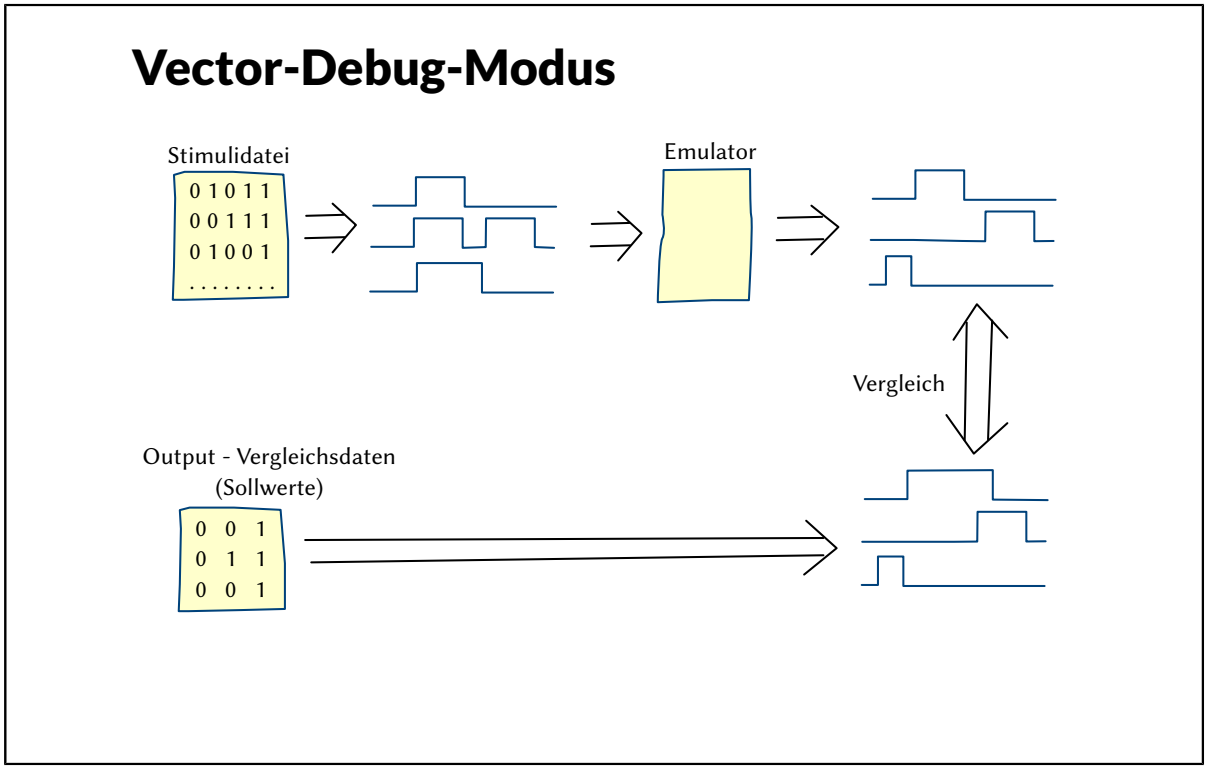
Die Tabelle zeigt eine Gegenüberstellung der Eigenschaften von Simulation, Emulation und gefertigtem ASIC einer Schaltung.

Emulation: Emulationsbetriebsarten



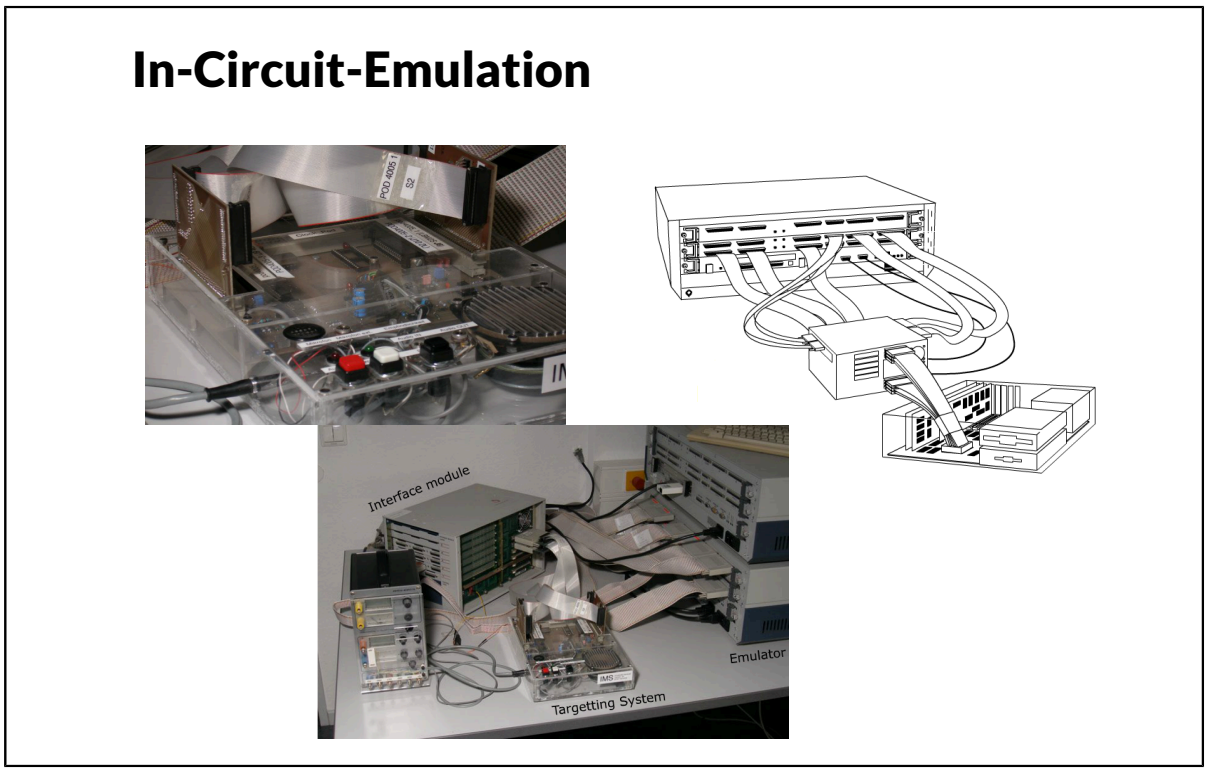
Ein Emulator kann in zwei Betriebsarten eingesetzt werden: - Vektor-Debug-Modus ("beschleunigte Simulation") - In-Circuit-Modus

Emulation: Vector Debug Modus



Im Vector Debug Mode wird die auf den Emulator abgebildete Schaltung genau wie bei einer Simulation mit festgelegten Stimuli angesteuert. Oftmals ist die Betrachtung der Schaltungsausgänge allein unzureichend, um das Verhalten der Schaltung nachvollziehen zu können. Darum bieten viele Emulationssysteme die Möglichkeit, in begrenztem Umfang auch Signale aus dem Inneren der emulierten Schaltung abzugreifen und zu speichern. In der Simulation sind dagegen alle Signale zu jedem Zeitpunkt zugänglich. Bei einem gefertigten ASIC ist dies nur mit einem sehr hohen Aufwnd möglich.. Dieser Emulationsmodus wird häufig als "beschleunigte Simulation" zur Verifikation der Schaltungsfunktion eingesetzt. Die Ausgangssignale der Emulation werden dazu mit vom Schaltungsentwickler festgelegten Sollsignalen verglichen, die seinen Erwartungen an die Funktion der Schaltung entsprechen. Stimmen die Emulationsergebnisse mit den Sollwerten überein, hat die Schaltung das vom Entwickler gewünschte Verhalten, ihre Funktion ist verifiziert.

Emulation: In-Circuit-Emulation



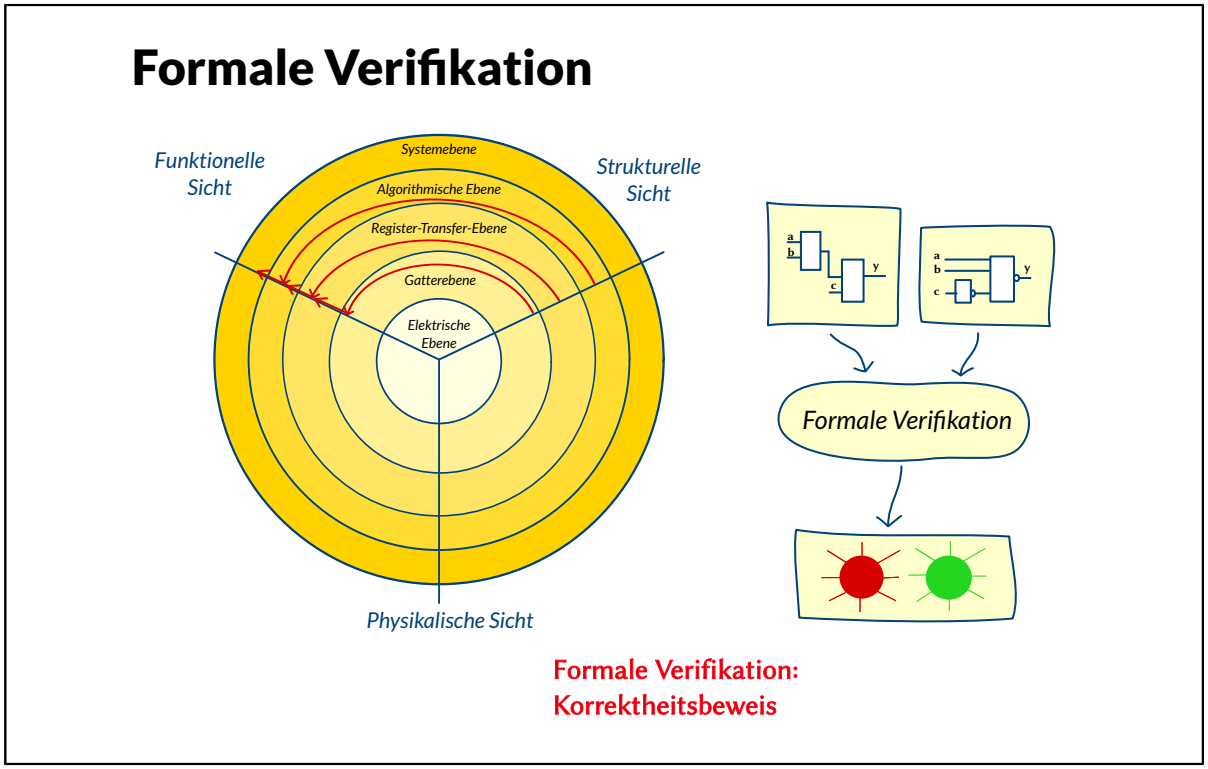
Bei der In-Circuit-Emulation wird der Emulator direkt an das Zielsystem angeschlossen, das für die zu emulierende Schaltung vorgesehen ist. Es ersetzt also den meist noch nicht gefertigten ASIC. Die Eingangsstimuli des Emulators sind die realen Signale des Systems. Es wird ebenso direkt von den Ausgangssignalen des Emulators angesteuert. Ein Problem stellt meist die Emulationsgeschwindigkeit dar. Obwohl sie der einer als ASIC gefertigten Schaltung nahekommt, liegt sie doch praktisch immer darunter. Die Peripherie ist aber meist für einen eng begrenzten Taktbereich entworfen. In dem Fall kommt es zu Fehlverhalten, wenn der Emulator die Peripherie mit zu geringer Geschwindigkeit ansteuert oder Daten von dort nicht schnell genug verarbeiten kann. Daher muss die Peripherie für eine In-Circuit-Emulation meist heruntergetaktet oder anderweitig neu mit der Schaltung synchronisiert werden. Trotzdem ist die In-Circuit-Emulation zur Verifikation der Kommunikation einer Schaltung mit ihrem Umfeld praktisch unerlässlich, gerade wenn weitere komplizierte Komponenten angesteuert werden, deren Verhalten ebenfalls nicht vollständig analytisch zu erfassen ist. Mit der In-Circuit Emulation wird also ebenso wie mit der Vector-Debug-Emulation ein funktioneller Test der Schaltung durchgeführt.

Electronic Design Automation (EDA)

Formale Verifikation

Formale Verifikation
Simulation und formale Verifikation
Werkzeuge der formalen Verifikation
Equivalence-Checking: Problemstellung
Erfüllbarkeitsproblem(Satisfiability,SAT)
SAT1-Problem
SAT-Equivalence-Checker
Graphenisomorphie
Binary Decision Diagrams (BDDs)
Binärer Entscheidungsbaum: Beispiel
Zusammenfassen identischer Teilbäume
Entfernen überflüssiger Knoten
Abhängigkeit der Knotenanzahl von der Entwicklungsreihenfolge
Vorteile der BDDs
Beispiel:ALU(SN74181)
Model-Checking
Symbolic Model Checking
Zustandsmengen
Temporale Logik
Elemente von CTL
CTL-Syntax
AX und EX
AF and EF
AG und EG
Beispiel: Modulo-3-Zähler

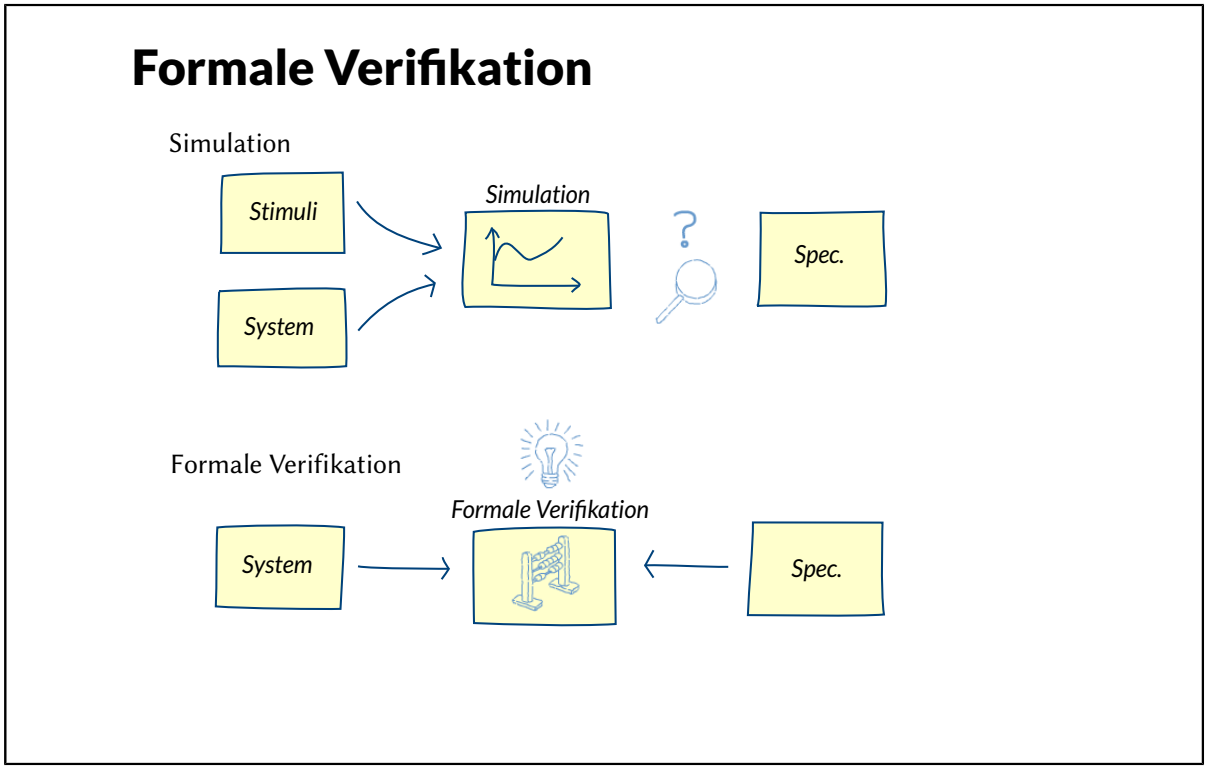
Formale Verifikation: Formale Verifikation



Unter formaler Verifikation versteht man Verfahren, die die korrekte Funktion eines Systems anhand von mathematischen Methoden nachweisen.

Voraussetzung für die erfolgreiche Anwendung von formalen Methoden zur Entwurfsverifikation ist eine formale Beschreibung des Testobjekts. Schaltungsdarstellungen in einer Hardware-Beschreibungssprache, aber auch strukturelle Darstellungen in Form von Netzlisten, genügen diesen Anforderungen und können somit mit formalen Methoden verifiziert werden.

Formale Verifikation: Simulation und formale Verifikation



Im Gegensatz zur Simulation, die aufgrund der endlichen Anzahl von Stimuli stets experimentellen Charakter hat, liefern die Methoden der formalen Verifikation einen mathematischen Beweis. Auf diese Weise können nicht nur Fehler im Entwurf gefunden, sondern es kann auch die Fehlerfreiheit eines Entwurfs bewiesen werden.

Formale Verifikation: Werkzeuge der formalen Verifikation

Werkzeuge der formalen Verifikation

Equivalence-Checker

- Überprüft die vollständige funktionelle Übereinstimmung von Implementierung und Spezifikation, oder verschiedene Implementierungen gegeneinander.
- Einfach und effizient einsetzbar.

Model-Checker

- Überprüft funktionelle Eigenschaften (properties) eines Systems.
- Zu überprüfende Eigenschaften müssen in einer formalen Sprache beschrieben werden.

Problem beim Model-Checking: Zustandsraumexplosion

- Formaler Beweis erfordert Analyse des gesamten Zustandsraums eines Systems.
- Exponentielle Laufzeit- und Speicherkomplexität.

Die wesentlichen heute gebräuchlichen formalen Verifikations-Werkzeuge können in zwei Gruppen eingeteilt werden:

Equivalence-Checker:

Diese Verfahren weisen die vollständige funktionale Übereinstimmung zwischen zwei unterschiedlichen Implementierungen nach. Eingesetzt wird Equivalence- Checking z.B. um das funktionelle Modell einer Schaltung mit der Gatternetzliste zu vergleichen und somit die Korrektheit des Synthese-Schritts im Entwurfsablauf einer integrierten Schaltung formal zu beweisen. Zeitraubende Simulationen der Gatternetzliste werden damit eingespart.

Die Verfahren des Equivalence-Checkings sind weit entwickelt. Für kombinatorische Systemteile lassen sich diese Verfahren sehr einfach und effizient einsetzen.

Model-Checker:

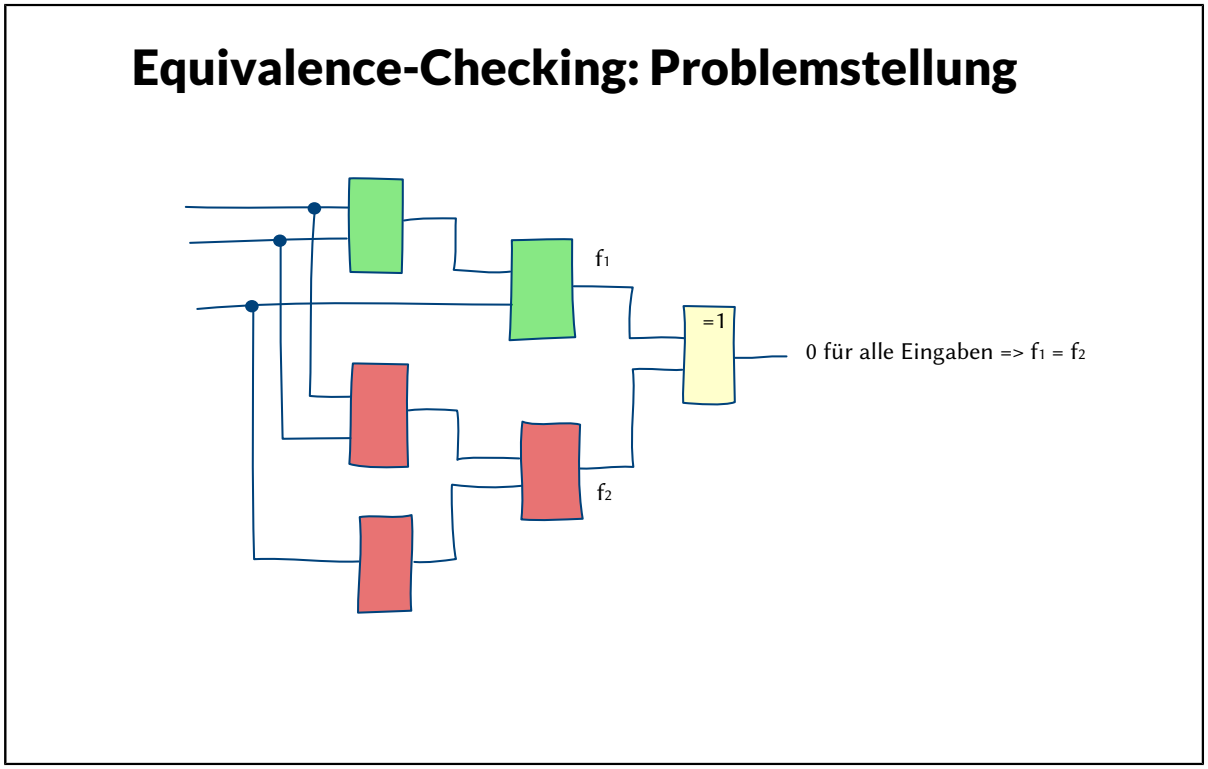
Model-Checker überprüfen einzelne Eigenschaften einer Schaltungsbeschreibung. Es kann beispielsweise zu überprüfen sein, ob eine Schaltung nach einer gewissen Zeit immer wieder in ihren Ruhezustand zurückkehrt. Die Schwierigkeit besteht darin, die zu überprüfenden Eigenschaften der Schaltung in der Eingabesprache für den Model-Checker zu beschreiben.

Generell gilt für Verfahren der formalen Verifikation, dass sie für den Anwender, den Schaltungsentwickler, schwerer zugänglich und ungewohnter sind als Simulatoren. Die notwendige Formalisierung der Eingaben stellt neue Anforderungen an den Design-Prozess und damit an den Entwickler. Auch die Ausgaben der formalen Verifikationswerkzeuge sind nicht immer leicht verständlich.

Die Grundidee des Modell-Checking ist eine Erreichbarkeitsanalyse im Zustandsraum. Damit liegt das Hauptproblem der formalen Verifikationsverfahren in der so genannten Zustandsexplosion (State Explosion). Um einen Beweis durchführen zu können, muss unter Umständen der gesamte Zustandsraum eines Systems untersucht werden. Die Zahl der zu untersuchenden Zustände steigt exponentiell mit der Anzahl der Zustandsraumdimensionen an. Die Speicher- und Laufzeitkomplexität

solcher Verfahren sind damit exponentiell in den Dimensionen des Zustandsraumes. Die exponentielle Komplexität stellt de facto eine obere Grenze für die behandelbaren Systemgrößen dar. Allerdings konnte durch verschiedene Optimierungen die Einsetzbarkeit der Verfahren erheblich gesteigert werden, so dass heute große digitale Blöcke handhabbar sind.

Formale Verifikation: Equivalence-Checking: Problemstellung



Die Grundaufgabe des Equivalence-Checking besteht darin, zwei boolesche Funktionen auf funktionelle Gleichheit zu überprüfen. Beim expliziten Ansatz betrachtet man die Verknüpfung beider Funktionen. Die beiden Ausgangssignale müssen für beliebige Eingangskombinationen stets gleich sein. Ist dies erfüllt, so sind die beiden Funktionen funktionell identisch. Dies ist im Bild durch die Verknüpfung von f_1 und f_2 über ein Exor-Gatter dargestellt.

Ein geeignetes Verifikationsverfahren ergibt sich aus der Idee des Widerspruchsbeweises. Nehmen wir an, die beiden Funktionen seien nicht identisch. Dann muss es eine Eingangskombination geben, für die am Ausgang des Exor-Gatters eine logische 1 auftritt. Umgekehrt gilt, dass, wenn es unter der Annahme des Ausgabewertes 1 gelingt, eine gültige Eingangskombination zu finden, die Schaltungen offenbar nicht identisch sind. Gelingt es nicht, eine solche Kombination zu finden, sind die beiden Funktionen identisch. Diese Fragestellung führt unmittelbar auf das sogenannte Erfüllbarkeitsproblem.

Formale Verifikation: Erfüllbarkeitsproblem(Satisfiability,SAT)

Erfüllbarkeitsproblem (Satisfiability, SAT)

- SAT1: Überprüfung der 1-Erfüllbarkeit von booleschen Funktionen, die in konjunktiver Normalform gegeben sind
- SAT0: Überprüfung der 0-Erfüllbarkeit von booleschen Funktionen, die in disjunktiver Normalform gegeben sind

Das Finden einer Eingangskombination für eine boolesche Gleichung bei gegebenem Ausgangswert ist als Erfüllbarkeitsproblem (Satisfiability-Problem) bekannt. Dieses Problem tritt in vielen Bereichen der Informatik auf.

Formale Verifikation: SAT1-Problem

SAT1- Problem

- Geg.: Boolesche Funktion f in n Variablen $x=(x_1, x_2, .., x_n)$
  in konjunktiver Normalform
- Beispiel: $f(x) = (x_1 + \overline{x_2}+...+ x_n) *...* (\overline{x_1} + x_2+...+ \overline{x_n})$
- Ges.: Belegung $x \in \{0,1\}_n$ mit $f(x)=1$, falls diese existiert
- Im schlechtesten Fall müssen alle möglichen Belegungen getestet werden.
- Anzahl der möglichen Belegungen: 2_n
  $n = 10$ ergibt $2_{10} = 1024 \approx 10_3$
  $n = 1000$ ergibt $2_{1000} = 2_{10^*100} \approx 10_{3^*100} = 10_{300}$
- Rechenzeitkomplexität von "Satisfiability": $O(2_n)$
  Satisfiability ist NP-vollständig.

SAT ist ein klassisches NP-vollständiges Problem und ist im Allgemeinen nicht effizient lösbar. Dennoch gibt es Verfahren, die in vielen praktischen Anwendungen gute Ergebnisse zeigen. Vor allem in den letzten 10 Jahren wurden neue Techniken entwickelt, die ein effizientes Lösen von Formeln mit einer großen Zahl von Variablen ermöglichen.

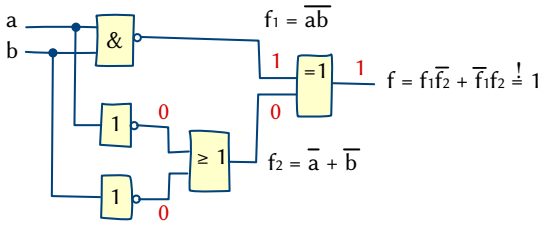
Formale Verifikation: SAT-Equivalence-Checker

SAT-Equivalence-Checker

Es wird ein EXOR-Gatter genommen und nachgewiesen,
dass f = 1 nicht erfüllbar ist, d.h. zu einem Widerspruch führt.

Beispiel :

strukturell :



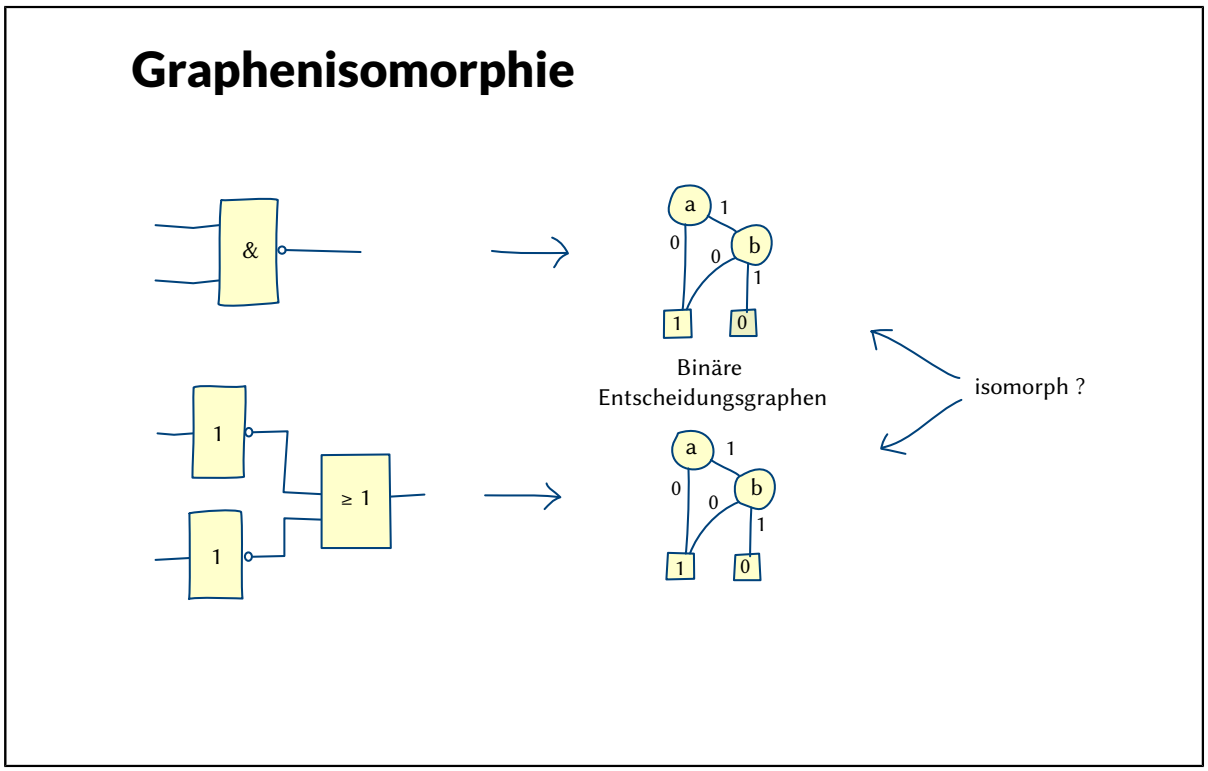
symbolisch :

$$\overline{a\,b}\,\overline{(\overline{a} + \overline{b})} + \overline{\overline{a\,b}}\,(\overline{a} + \overline{b}) \stackrel{!}{=} 1$$
$$(\overline{a} + \overline{b})\,(a\,b) + a\,b\,(\overline{a} + \overline{b}) \stackrel{!}{=} 1$$
$$\overline{a}ab + \overline{b}ab + ab\overline{a} + ab\overline{b} \stackrel{!}{=} 1$$
$$0 \neq 1$$

Im obigen Beispiel soll die Übereinstimmung der booleschen Funktionen f_1 und f_2 geprüft werden. Wie die Rechnung zeigt, kann es nicht gelingen, für a,b eine Wertebelegung zu finden, für die sich $f=1$ ergibt, da der Ausdruck $f_1\overline{f_2}+\overline{f_1}f_2$ stets 0 ist. Das kann mit einem normalen SAT-1 Löser gemacht werden.

In den entsprechenden Verifikationswerkzeugen findet der Widerspruchsbeweis auch auf andere Weise statt. Grundsätzlich geeignet sind SAT-Löser, BDD-basierte Löser und auch der D-Algorithmus, der im Abschnitt Test erläutert wird. Hier werden Signale vom Ausgang zu den Eingängen propagiert, bis sich entweder eine gültige Eingangsbelegung oder ein Widerspruch ergibt.

Formale Verifikation: Graphenisomorphie



Statt ein Erfüllbarkeitsproblem zu untersuchen, kann man beim Equivalence Checking versuchen, die beiden Systeme direkt miteinander zu vergleichen. Dazu bildet man die Systeme jeweils auf einen Graphen ab und versucht, die Isomorphie der beiden Graphen zu beweisen. Damit dies möglich ist (das generelle Graphenisomorphieproblem ist NP-vollständig), muss eine eindeutige (kanonische) Darstellung gefunden werden. Diese ist durch so genannte binäre Entscheidungsgraphen Binary Decision Diagrams (BDDs) gegeben.

Formale Verifikation: Binary Decision Diagrams (BDDs)

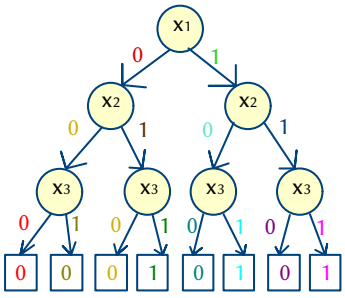
Binary Decision Diagrams (BDDs)

- Theorie der BDDs wurde bereits um 1960 entwickelt.
- BDDs sind eine kanonische Darstellung für boolesche Funktionen.
- Bryant zeigte 1986, dass fast alle logischen Operationen sehr effizient auf BDDs durchgeführt werden können, sofern eine feste Variablenordnung zugrundegelegt wird und gewisse Reduktionsregeln auf die BDDs angewandt werden.
- Durchbruch in der formalen Verifikation.

Binäre Entscheidungsgraphen wurden durch R. E. Bryant entscheidend weiterentwickelt. Mit dieser Art von Graphen lassen sich boolesche Funktionen in einer sowohl kanonischen als auch kompakten Darstellung speichern und verarbeiten.

Formale Verifikation: Binärer Entscheidungsbaum: Beispiel

Binärer Entscheidungsbaum: Beispiel

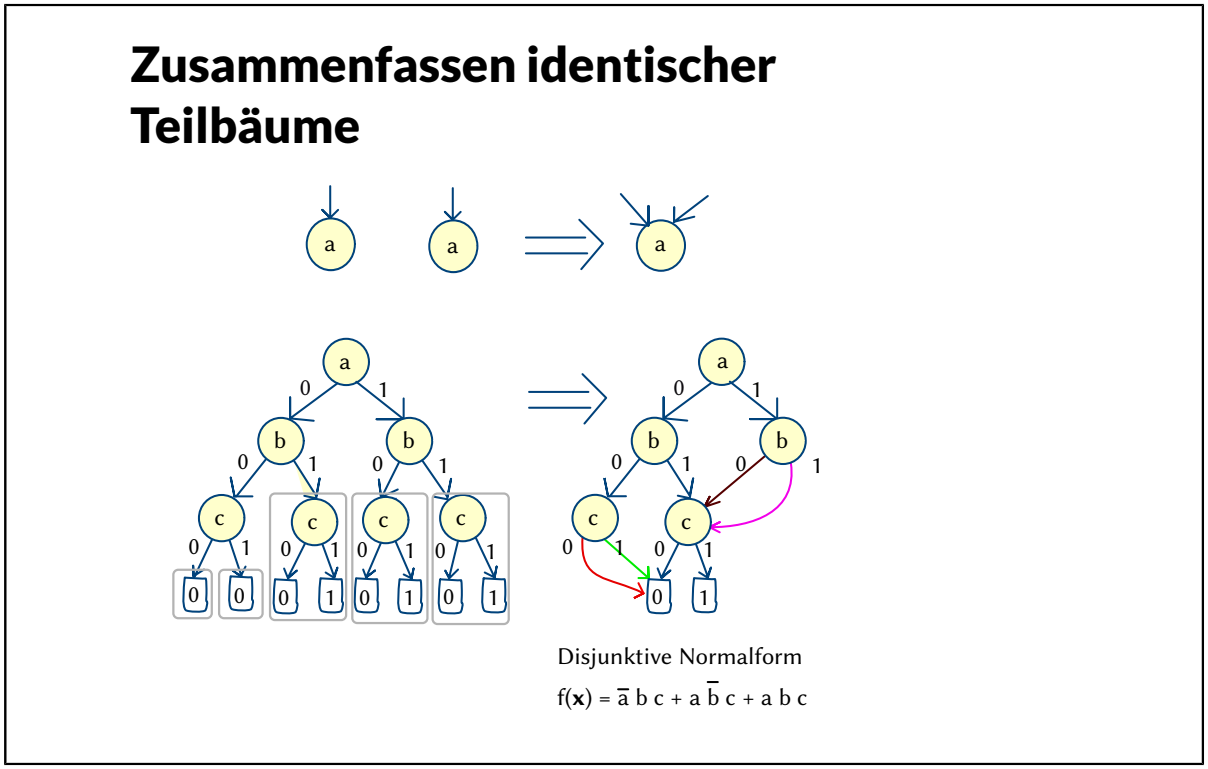


x_1	x_2	x_3	$f(\mathbf{x})$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Das Prinzip der Entscheidungsgraphen ist einfach: Von jedem Knoten des gerichteten Graphen gehen genau zwei gerichtete Kanten aus. Dem Knoten zugeordnet ist eine boolesche Variable, deren Wertebelegung mit 0 oder 1 durch die beiden Kanten repräsentiert wird. Nach dem booleschen Entwicklungssatz kann die Ausgangsfunktion durch diese Festlegung in zwei Teilfunktionen zerlegt werden. Wendet man dieses Prinzip für alle Eingangsvariablen an, so ergeben sich lediglich die beiden Senken 0 und 1 im Graphen. Der gerichtete Graph ist zyklenfrei. Durch einen Pfad wird eine Wertebelegung der Variablen eindeutig festgelegt. Die entsprechende Senke des Pfades weist dieser Wertebelegung einen Wahrheitswert zu.

Endet maximal eine Kante in jedem Knoten des Graphen, so entsteht ein binärer Entscheidungsbaum. Ordnet man alle Knoten, die derselben Variablen zugeordnet sind, auf gleicher Höhe an, so ergibt sich optisch eine übersichtliche Darstellung.

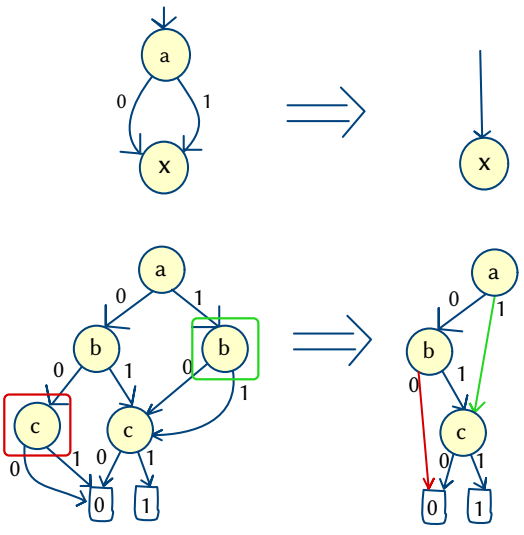
Formale Verifikation: Zusammenfassen identischer Teilbäume



Ein binärer Entscheidungsbaum kann durch zwei Vereinfachungsoperationen vereinfacht werden. Erstens können identische Teilbäume zusammengefasst werden und zweitens können Knoten, von denen beide Kanten auf den selben Folgeknoten zeigen, entfernt werden. Führt man darüber hinaus eine feste Reihenfolge für die booleschen Variablen im Entscheidungsgraphen ein, so entsteht ein geordneter Entscheidungsgraph (Ordered Binary Decision Diagram, OBDD). Der OBDD bildet eine kanonische Darstellung einer booleschen Funktion, die gegenüber den bekannten konjunktiven und disjunktiven kanonischen Normalformen oft kompakter und leichter zu manipulieren ist.

Formale Verifikation: Entfernen überflüssiger Knoten

Entfernen überflüssiger Knoten



OBDD

ROBDD

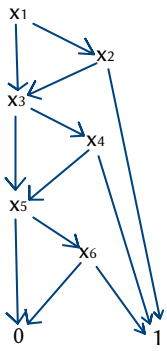
Äquivalente boolesche Funktion:

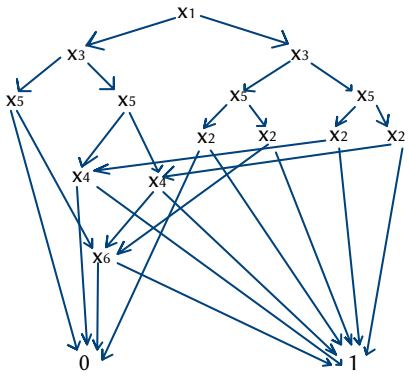
$f(\mathbf{x}) = c\,(\overline{a}\,b + a)$

Formale Verifikation: Abhängigkeit der Knotenanzahl von der Entwicklungsreihenfolge

Abhängigkeit der Knotenanzahl von der Entwicklungsreihenfolge

$$f(\mathbf{x}) = x_1x_2 + x_3x_4 + x_5x_6$$



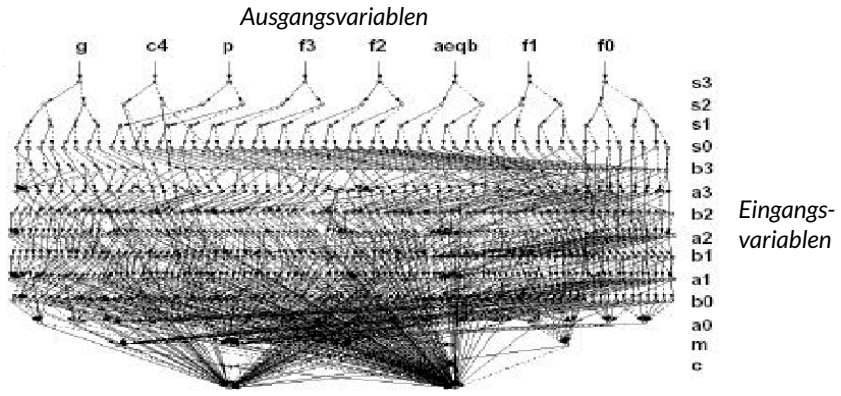


Die Wahl der Entwicklungsreihenfolge der einzelnen Variablen beeinflusst in kritischer Weise die Anzahl der Knoten des Graphen und damit den zur Äquivalenzprüfung benötigten Aufwand. Ihr ist also besondere Aufmerksamkeit zu widmen. Das Bild zeigt ein klassisches Beispiel, das 1986 von Bryant vergestellt wurde.

Formale Verifikation: Vorteile der BDDs

Vorteile der BDDs

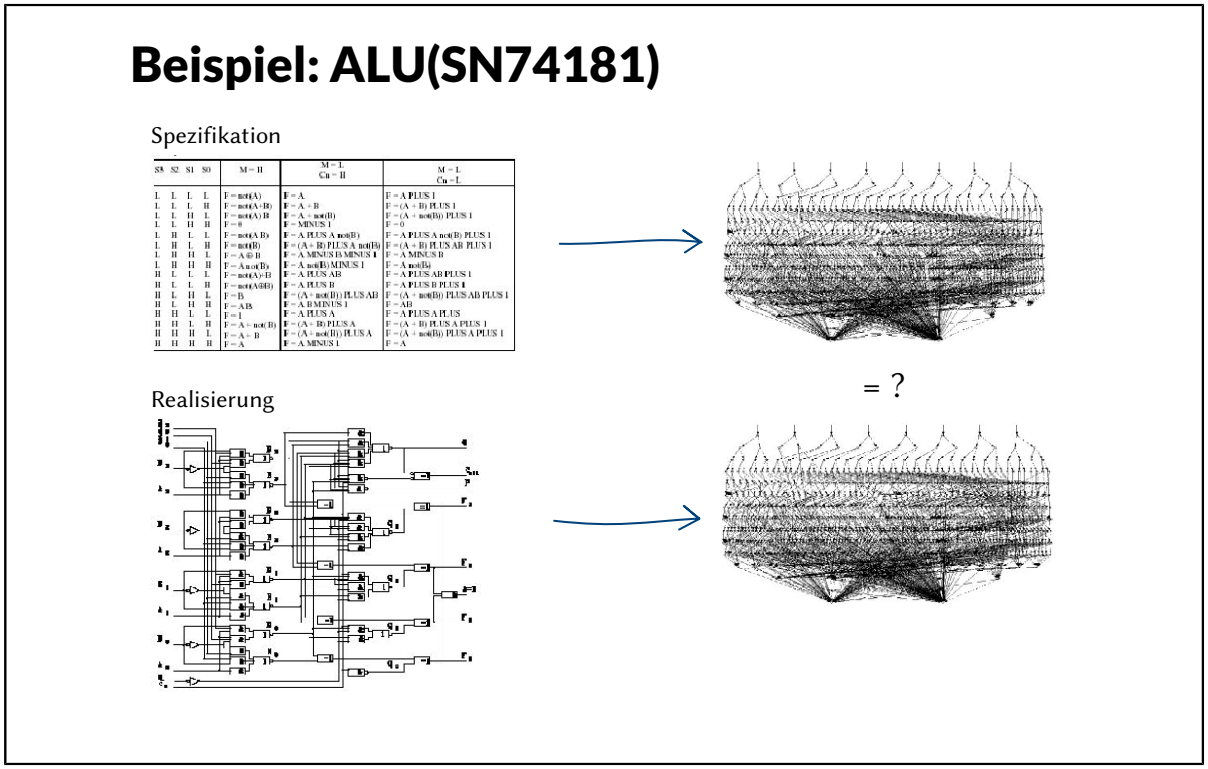
- Logische Funktionen können durch BDDs repräsentiert und in vielen Fällen in linearer Komplexität in Zeit und Speicher aufgebaut werden.
- Äquivalenztest kann in linearer Zeit durchgeführt werden.
- Beispiel : "Shared" OBDD einer 4-Bit-ALU(SN74181)



Digitale kombinatorische Systemteile lassen sich offensichtlich mit Hilfe von OBDDs beschreiben. Die dem System zu Grunde liegende booleschen Gleichungen werden in entsprechende BDDs abgebildet. Damit ist das digitale verzögerungsfreie Verhalten des Systems vollständig wiedergegeben. Für die Konstruktion der booleschen Gleichungen aus einer Netzliste heraus gibt es zwei Ansätze. Zum einen können die Funktionen ausgehend von allen Systemeingängen aufgestellt werden, zum anderen kann die Konstruktion aber auch von den Ausgängen her erfolgen. Das Ergebnis beider Verfahren ist das gleiche, lediglich die Laufzeit der beiden Verfahren kann sich für verschiedene Systeme unterscheiden.

Das Bild zeigt beispielhaft die "Shared" OBDD einer 4-Bit-ALU (SN74181). "Shared" bedeutet, dass alle (horizontal dargestellten) Ausgangsvariablen in einem gemeinsamen Baum dargestellt werden.

Formale Verifikation: Beispiel:ALU(SN74181)



Das gezeigte Bild gibt einen Eindruck, wie zwei Systeme, die in unterschiedlicher Darstellung vorliegen (Spezifikation in funktioneller Sicht, Realisierung in struktureller Sicht jeweils auf Gatterebene) über die Beschreibungsform BDD miteinander verglichen werden können.

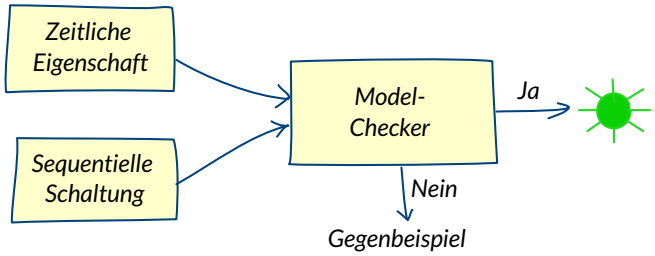
Wichtig ist, dass zum Beweis ihrer Übereinstimmung keinerlei Stimuli benötigt werden, man spricht deshalb beim Equivalence Checking auch von einer statischen Verifikationsmethode.

Für sequentielle Systemteile ist das Equivalence Checking deutlich schwieriger. In der Praxis kann es nur für Sonderfälle oder für kleine Beispiele verwendet werden. In diesem Script wird daher darauf auch nicht eingegangen.

Formale Verifikation: Model-Checking

Model-Checking

- Überprüfung zeitlicher Eigenschaften von sequentiellen Schaltungen.
- Bsp.: "Es kann nicht vorkommen, dass die Ampelsteuerung alle Signale gleichzeitig auf grün setzt."
- "Irgendwann wird jedes Ampellicht grün."



Kann die Eigenschaft nicht verifiziert werden, erzeugt der Model-Checker Informationen (Stimuli etc.) für ein entsprechendes Gegenbeispiel.

Model-Checking überprüft zeitliche Eigenschaften sequentieller Schaltungen. Das Anwendungsgebiet von Model-Checking-Verfahren ist vielfältig und reicht von der Software-Verifikation und der Verifikation von Übertragungs- Protokollen bis hin zur Überprüfung von digitalen Systemen. Entsprechend vielseitig sind auch die verwendeten Ansätze und Algorithmen. Die Grundlagen für das Model-Checking kommen aus der Informatik, wo die Verfahren zur Verifikation von Software genutzt werden. Die hier vorgestellten Algorithmen beziehen sich jedoch ausschließlich auf die Verifikation digitaler Systeme.

Beim Model-Checking werden einzelne Eigenschaften eines Systems untersucht. Dies setzt voraus, dass solche Eigenschaften in einer formalen Sprache beschrieben werden können. Je nach Art der Eigenschaft werden verschiedene Anforderungen an eine solche Sprache und damit auch an die Überprüfungsalgorithmen gestellt.

Formale Verifikation: Symbolic Model Checking

Symbolic Model Checking: Charakteristische Funktion

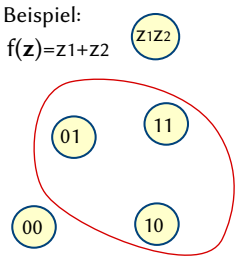
- Gegeben: Variablenvektor **z** = Zustandsvektor
- Eine Boolesche Funktion $f_A(\mathbf{z})$ kann eine Menge von Zuständen beschreiben:

$$f_A(\mathbf{z}) = \begin{cases} 1 & \text{falls } \mathbf{z} \in A \\ 0 & \text{falls } \mathbf{z} \notin A \end{cases}$$

- Sie heißt charakteristische Funktion
- Dann ist die beschriebene Zustandsmenge:

$$A = \{ \mathbf{z} \mid f_A(\mathbf{z}) = 1 \}.$$

Beispiel:
$f(\mathbf{z})=z1+z2$



Man erhält folgende Vorteile:
- Symbolische Beschreibung einer Zustandsmenge.
- Damit können u. U. sehr große Zustandsmengen mit einer kleinen Funktion beschrieben werden.
- Dies lässt sich auch durch BDDs realisieren.
- Übergangsfunktionen lassen sich ebenfalls so beschreiben.

Da bei sequentiellen Schaltungen (Automaten) sehr viele Zustände auftreten können, mussten Methoden gefunden werden, über Zustandsmengen mit einer großen Kardinalität zu argumentieren. Ein Durchbruch dazu war das Symbolic Model Checking. Im Wesentlichen werden dazu Beschreibungen für Zustandsmengen und Zustandsübergangsrelationen benötigt. Im Folgenden wird eine Speicherung dieser Daten mithilfe von charakterischen Funktionen erläutert.

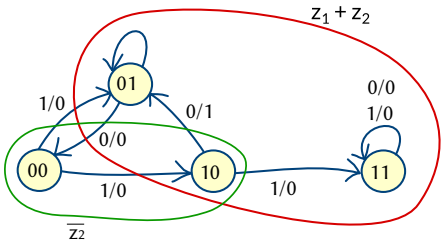
Betrachtet man alle Wertebelegungen des Variablenvektors z einer Booleschen Funktion fA, bei der das Ergebnis der Funktion 1 entspricht, so ergibt sich eine Menge von Werten A = {z | fA(z) = 1}. Identifiziert man einen Zustand eines Zustandsautomaten mit jeweils einem dieser Vektoren, können Zustandsmengen durch boolesche Funktionen und damit auch mit OBDDs beschrieben werden. Die boolesche Funktion fA(z) wird als charakteristische Funktion der Wertemenge A bezeichnet.

fA(z) = 1, falls z Element von A oder 0, falls nicht

Zur Beschreibung von Zustandsübergängen muss die Darstellung erweitert werden. Für jeden Zustand z1 bestimmt man einen oder mehrere Folgezustände z2. Das heißt, die Zustandsübergangs-Relation kann beispielsweise als delta(z1,z2) beschrieben werden. Damit können Algorithmen symbolisch über Millionen von Zuständen und deren sequentielle Abarbeitung argumentieren, ohne diese explizit aufzählen zu müssen.

Darstellung von Zustandsmengen

- Ausgangspunkt: Zustandsdiagramm einer sequentiellen Schaltung
- Boolesche Ausdrücke (charakteristische Funktionen) charakterisieren Mengen von Zuständen
- Beispiel: zwei Zustandsvariablen z_1, z_2
  - $z_1 + z_2$ charakterisiert die Menge {01, 10, 11}
  - $\overline{z_2}$ charakterisiert die Menge {00, 10}



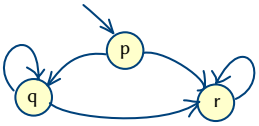


Formale Verifikation: Temporale Logik

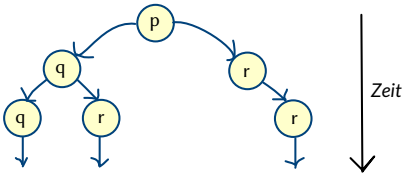
Temporale Logik

Zur Beschreibung von logischen Abläufen in der Zeit setzt man temporale Logik ein.

Zustandsübergänge werden in einen unendlichen Baum ("Berechnungsbaum") entflochten.



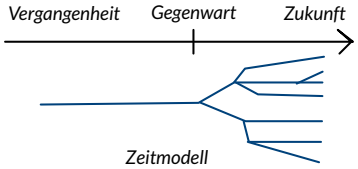
Zustandsübergangsdiagramm



Berechnungsbaum

Über die Pfade des Berechnungsbaums lassen sich Aussagen treffen - mit der Computation Tree Logic (CTL).

Das Zeitmodell lässt Verzweigungen in der Zukunft zu.



Zeitmodell

Für einfache Bedingungen in digitalen Systemen ist die boolesche Logik ausreichend. Beispielsweise könnte y = x1x2 eine zu überprüfende Spezifikationsbedingung sein. Bei genauerer Betrachtung fällt auf, dass für sequenzielle Systeme eine solche Bedingung eine unvollständige Beschreibung liefert. Soll diese Bedingung immer gelten oder ist es ausreichend, dass y = x1x2 irgendwann einmal gilt? Das heißt, für eine genauere Beschreibung sind zeitliche Zusammenhänge mit zu spezifizieren.

Dies kann durch temporale Logiken geschehen, in denen der Wahrheitsgehalt einer Funktion von der Zeit abhängt. Temporale Logiken werden in die beiden Klassen lineare temporale Logik (Linear Time Logic) und verzweigende temporale Logik (Branching Time Logic) eingeteilt. Bei den linearen temporalen Logiken geht man davon aus, dass das Systemverhalten in der Zukunft eindeutig bestimmt ist, dass es also keine verschiedenen Entwicklungen in der Zukunft geben kann. Dagegen lässt das verzweigende Zeitmodell in der Zukunft verschiedene Entwicklungen zu, die durch nichtdeterministisches Verhalten des Systems oder durch unterschiedliche Eingangsgrößen hervorgerufen werden können.

Bei dem verzweigenden Zeitmodell besteht die zeitliche Entwicklung nicht aus einer linearen Sequenz von Zuständen, sondern die Systemzustände fächern sich in der Zukunft auf. Es entsteht ein Baum, dessen Wurzel der momentane Systemzustand ist.

Demnach können zu einem Zeitpunkt in der Zukunft verschiedene Systemzustände existieren. Welchen Weg das System einschlagen wird, ist für die Betrachtung irrelevant, da alle Wege gleichberechtigt betrachtet werden, um eine allgemeingültige Aussage zu erhalten.

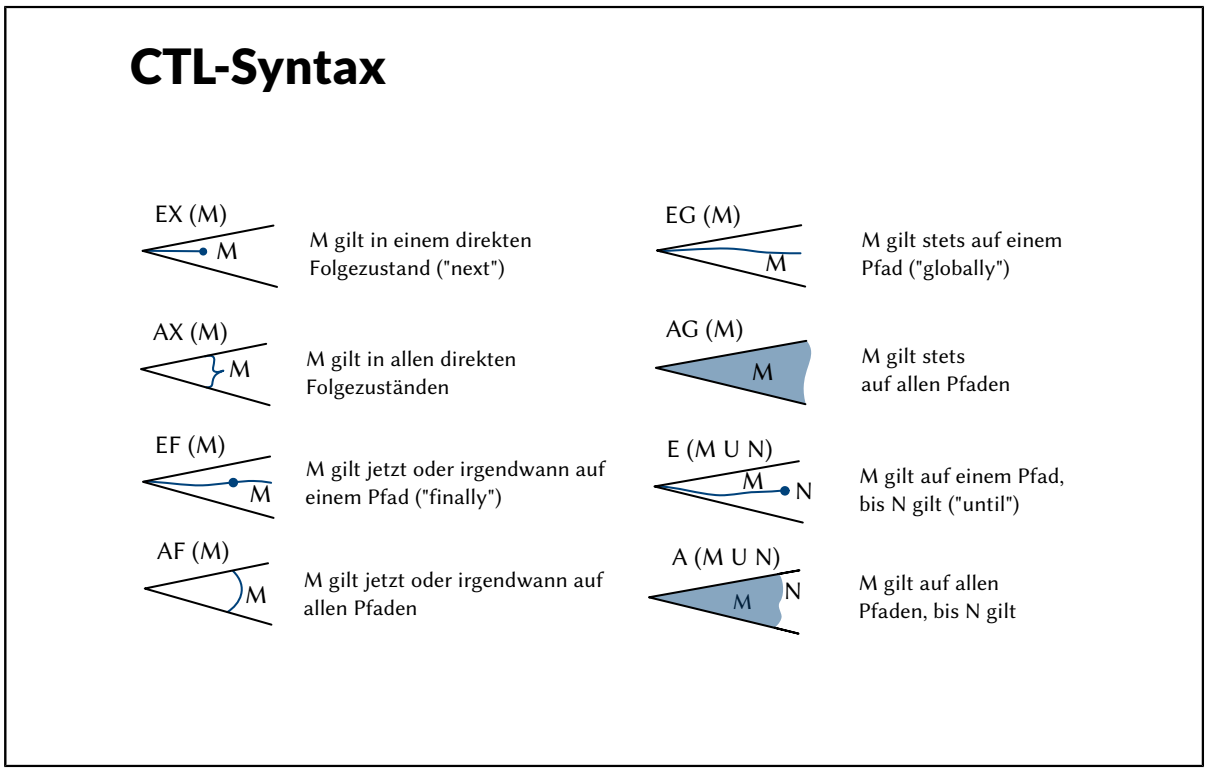
Formale Verifikation: Elemente von CTL

Elemente von CTL

Boolesche Operatoren	V = oder
	∧ = und
	¬ = nicht
Pfadquantoren	A = auf allen Pfaden
	E = auf mindestens einem Pfad
Temporale Operatoren	X = im nächsten Taktschritt (Ne**x**t)
	F = irgendwann (**F**inally)
	G = immer (**G**lobally)
	U = bis (**U**ntil)

Die in der Tabelle beschriebene temporale Logik wird CTL (Computeation Tree Logic) genannt. Sie geht auf einen Artikel von Clarke und Emerson aus dem Jahr 1981 zurück. Sie ergibt sich aus der booleschen Logik kombiniert mit vier temporalen Operatoren. Die vier Operatoren stehen für die Aussagen "im nächsten Taktschritt", "immer", "irgendwann" und "bis". Sie werden mit den Symbolen X, G, F und U abgekürzt. Die ersten drei temporalen Operatoren sind unär. Für die "bis"-Operation werden zwei Argumente benötigt. Der Einfachheit halber sind hier und im Folgenden bei den booleschen Funktionen nur "und", "oder" und "Negation" aufgeführt. Zusätzlich muss bestimmt werden, ob eine Bedingung auf einem Weg oder auf allen möglichen Wegen erfüllt sein muss. Dies wird durch zwei zusätzliche Pfadquantoren A und E geleistet. Eine vollständige CTL-Formel setzt sich aus jeweils einem Pfadquantor, einem temporalen Operator und einer bzw. zwei Argumentmengen zusammen.

Formale Verifikation: CTL-Syntax

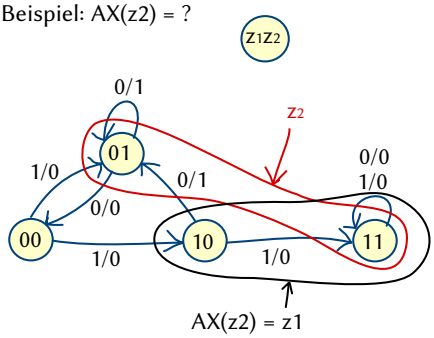


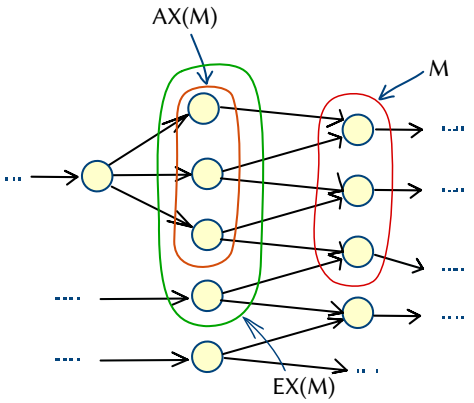
Aus der Kombination eines Pfadquantors mit einem temporalen Operator ergeben sich die acht dargestellten Ausdrücke, die jeweils auf Mengen oder auf durch boolesche Operatoren miteinander verknüpfte Mengen angewendet werden. Die Mengen beschreiben Mengen von Zuständen im Zustandsraum. Insofern befasst sich das Model-Checking mit der Erreichbarkeitsanalyse im Zustandsraum.

AX und EX

- AX(M) charakterisiert alle Zustände, die nur Nachfolgezustände haben, die durch M charakterisiert sind.
- EX(M) charakterisiert alle Zustände, von denen aus wenigstens ein Nachfolgezustand durch M charakterisiert ist.

Beispiel: AX(z2) = ?



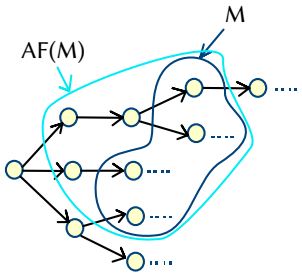


Ausschnitt aus dem Computation Tree

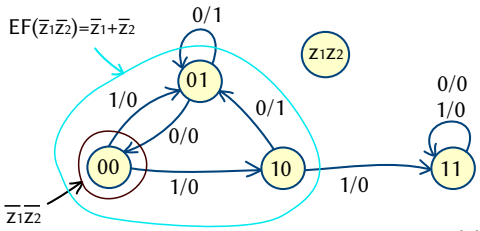
Formale Verifikation: AF and EF

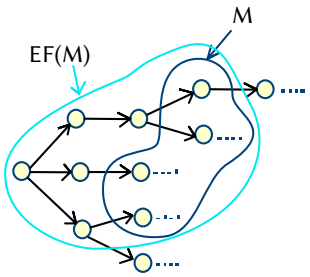
AF und EF

- AF(M): Zustände, von denen aus M unvermeidbar ist
  AF(M) charakterisiert alle Zustände, von denen aus M irgendwann sicher eintritt (Lebendigkeitseigenschaft), z.B. irgendwann zeigt die Ampel grün.
- EF(M): M ist erreichbar (jetzt oder später)



Beispiel: EF($\bar{z}_1\bar{z}_2$) = ?





"Von welchen Zuständen aus ist es möglich, wieder in den Anfangszustand zu kommen?"

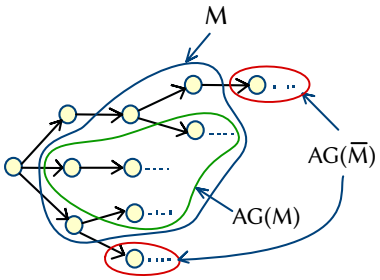
siehe AG und EG

Formale Verifikation: AG und EG

AG und EG

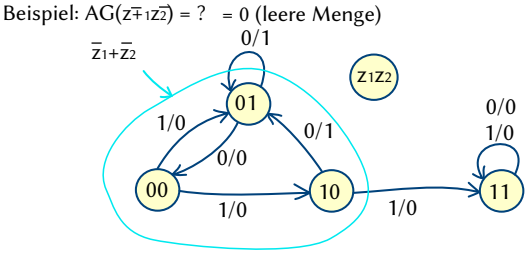
- AG(M): Alle aktuellen und alle Nachfolgezustände sind durch M charakterisiert.

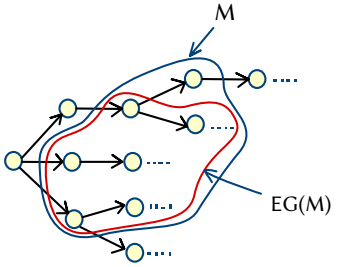
  AG($\overline{M}$) charakterisiert alle Zustände, von denen aus M nie eintreten kann (Sicherheitseigenschaft) z.B. es tritt nie ein, dass alle Ampeln grün sind.



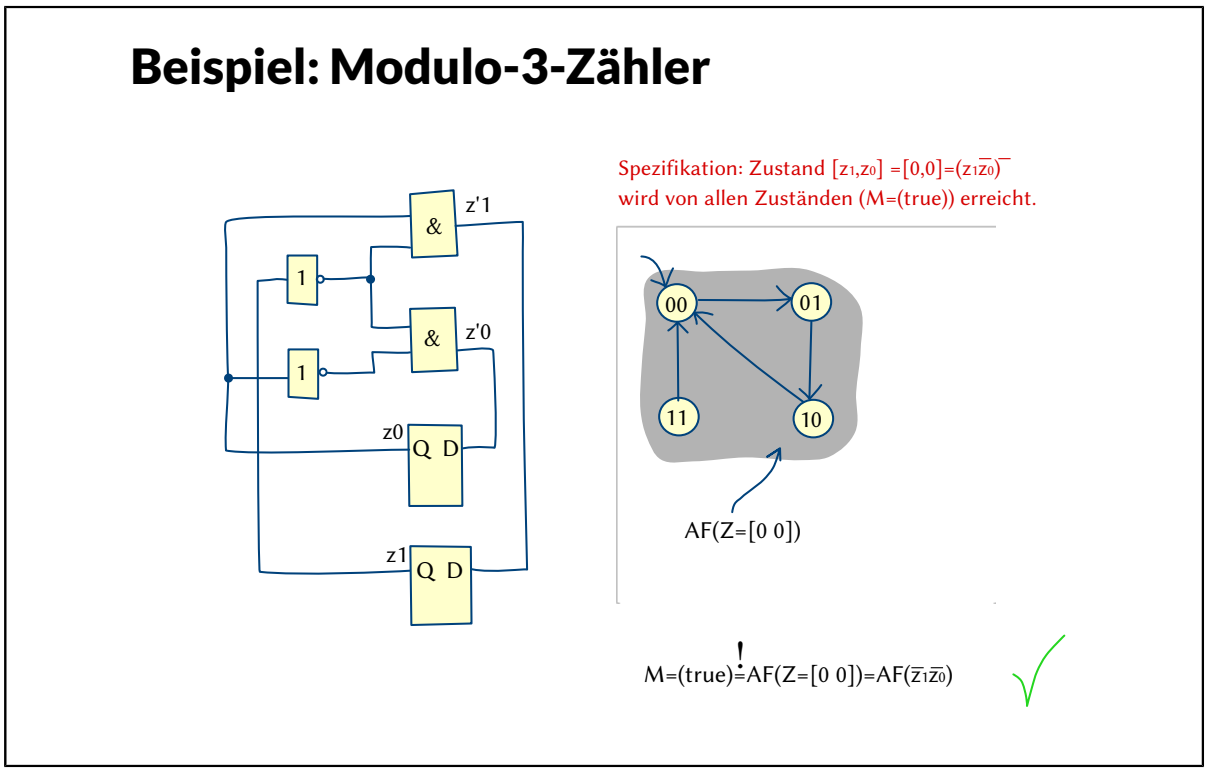
- EG(M): Es gibt mindestens einen Pfad, auf dem der aktuelle und alle Nachfolgezustände durch M charakterisiert sind.

Beispiel: AG($z\overline{+}_1z_2\overline{z}$) = ? = 0 (leere Menge)





Formale Verifikation: Beispiel: Modulo-3-Zähler



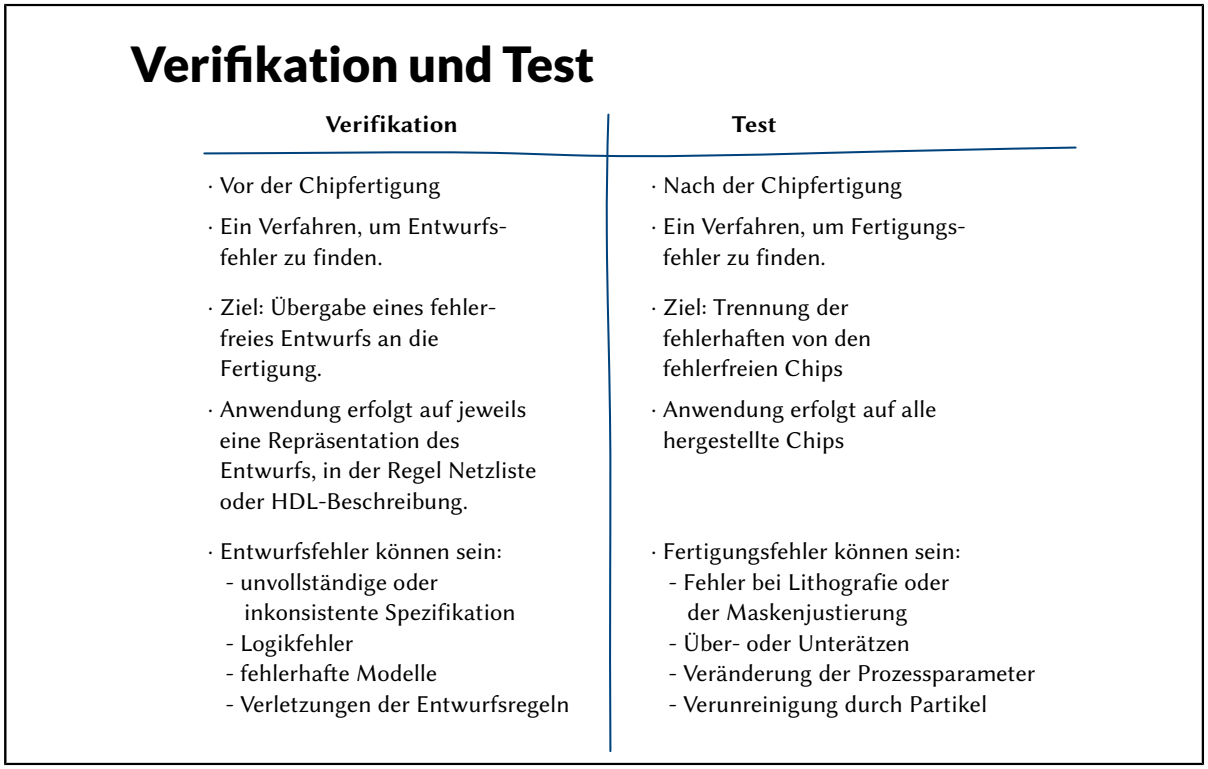
In diesem Beispiel wird beispielhaft eine Eigenschaft eines Modulo 3 Zählers überprüft. Zunächst muss dazu die Gatterschaltung in das Zustandsübergangsdiagramm umgewandelt werden. Dies geschieht im realen, OBDD-basierten Modelchecker jedoch nur implizit durch Aufstellen der OBDDs. Der Check-Prozess ist hier animiert dargestellt und liefert ein positives Ergebnis für die spezifizierte Eigenschaft, d.h. der Zustand [0,0] wird tatsächlich immer (d.h. von jedem anderen Zustand) erreicht.

Electronic Design Automation (EDA)

Test

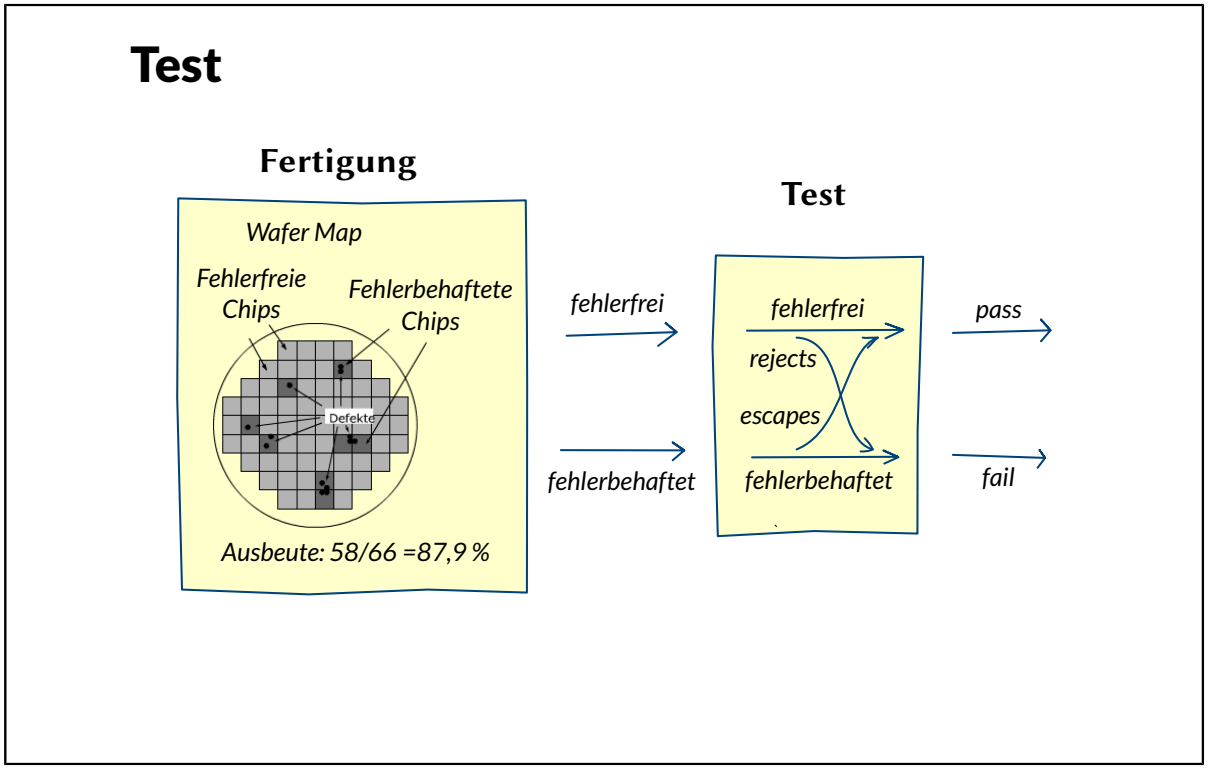
Verifikation und Test
Test
Chip-Ausbeute
Testbarkeitsindex
Testqualität
Vollständiger Test kombinatorischer Schaltungen
Vollständiger Test sequentieller Schaltungen
Testdurchführung
Testmuster
Funktionelle Testmuster
Strukturelle Testmuster
Fehlermodelle
Fehlererkennung und Testmuster
Einstellbarkeit (Steuerbarkeit)
Beobachtbarkeit
Beispiel: Fehlererkennung
D-Algorithmus
Implikation und Entscheidung
Testmustergenerierung mit D-Algorithmus
Beispiel zum D-Algorithmus
Scan Test
Fehlersimulation

Test: Verifikation und Test



Ein großer Teil des Aufwands beim Entwurf einer integrierten Schaltung beinhaltet die Analyse der Entwurfsergebnisse zur Überprüfung auf Einhatung der Spezifikation. Je früher im Entwurfsprozess ein Entwurfsfehler entdeckt wird, desto leichter lässt es sich beheben. Die Überprüfung der Entwurfsschritte zu diesem Zweck wird als Verifikation bezeichnet. Nach der Fertigung des Chips muss die Spezifikation erneut überprüft werden, um festzustellen, ob möglicherweise während des Fertigungsprozesses defekte Chips entstanden sind. Verifiziert wird, um Entwurfsfehler zu beheben. Getestet wird, um defekte Chips auszusortieren.

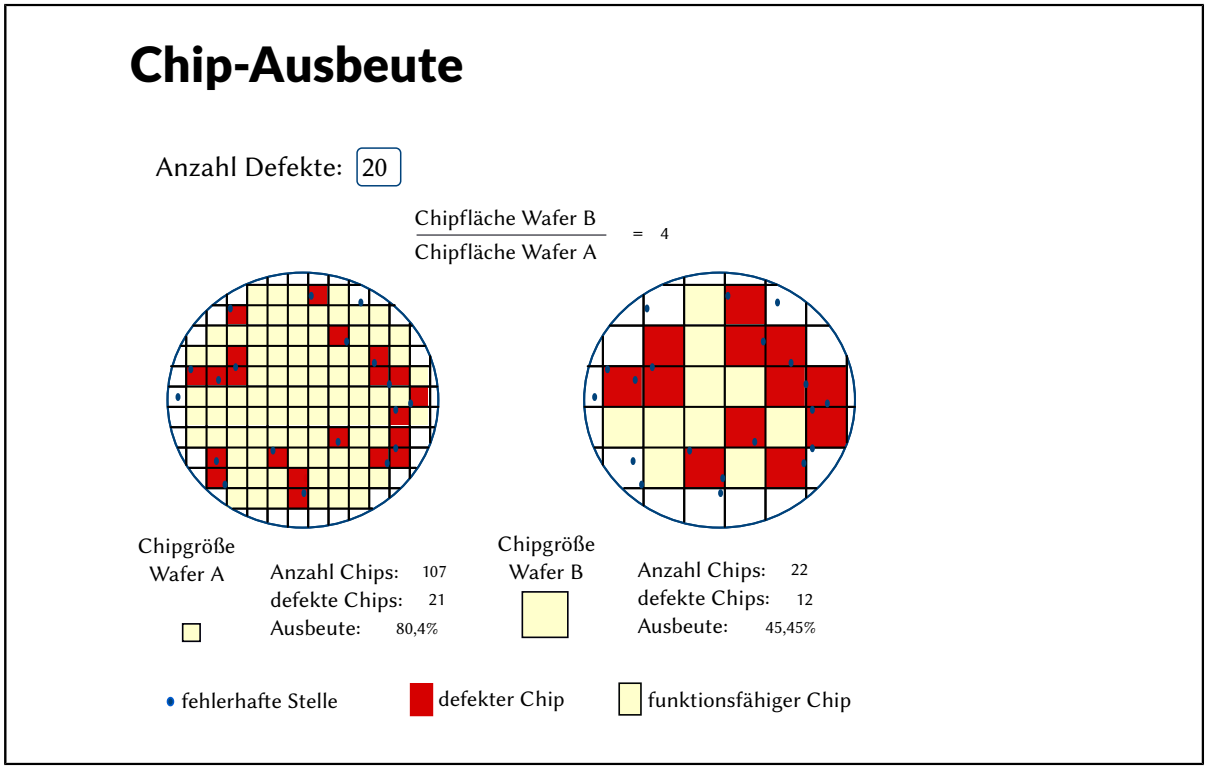
Test: Test



Wie bereits erwähnt soll der Test die fehlerhaften und fehlerfreien Chips voneinander trennen. Der Test soll dabei ausschließlich fehlerfreie Chips als solche markieren. Auf Grund der Komplexität der integrierter Schaltungen ist es nicht möglich die Schaltungen vollständig zu testen, so dass diese Anforderungen nicht vollständig erfüllt werden kann und fehlerbehaftete Chips als fehlerfrei "erkannt" werden. Es entstehen "escapes". Auf der anderen Seite kann auf Grund der Unvollkommenheit der Testumgebung (z.B. mangelnde Kontaktierung zwischen Testspitzen und Testpunkten) zum Aussortieren fehlerfreie Chips kommen, die als fehlerbehaftet erkannt werden. Es entstehen "rejects".

Die "escapes" führen zu den Ausfällen in den Zielsystemen und verursachen zusätzliche Kosten, die durch den Austausch der Chips bzw. der Systeme entstehen. Ein "reject" senkt künstlich die Ausbeute und trägt ebenfalls zur Steigung der Kosten bei.

Test: Chip-Ausbeute



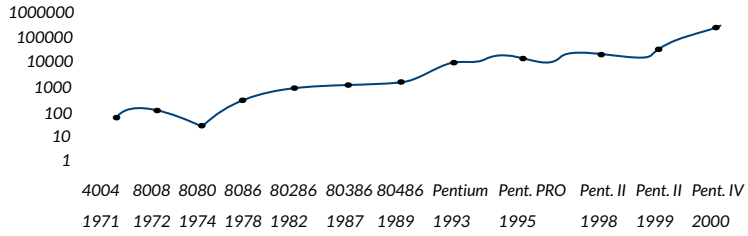
Test: Testbarkeitsindex

Testbarkeitsindex

Das Testbarkeitsindex ist ein Maß für die Erreichbarkeit schaltungsinterner Knoten

$$\text{Testbarkeitsindex} = \frac{\text{Anzahl Transistoren}}{\text{Anzahl externer Signalanschlüsse}}$$

Testbarkeitsindex von Intel-Mikroprozessoren:



4004	8008	8080	8086	80286	80386	80486	Pentium	Pent. PRO	Pent. II	Pent. II	Pent. IV
1971	1972	1974	1978	1982	1987	1989	1993	1995	1998	1999	2000

Die Testkosten sind bei der Herstellung integrierter Schaltungen ein wesentlicher Faktor. Sie steigen mit wachsender Komplexität. Ein wesentlicher Grund dafür ist, dass die Anzahl externer Anschlüsse nicht im gleichem Maße steigt, wie die Anzahl integrierter Transistoren. Hieraus folgt, dass die Zahl interner Zustände, die beim Test zu durchlaufen ist, zunimmt, wodurch die Testzeit und damit die Testkosten steigen.

Test: Testqualität

Testqualität

$$\text{Testqualität} = \frac{\text{Anzahl korrekter Bausteine}}{\text{Anzahl ausgelieferter Bausteine}}$$

Wenn 99,9 % gut genug sind, dann werden:

- heute 17 Kinder den falschen Eltern zugeordnet.
- jedes Jahr 268.000 defekte Autoreifen ausgeliefert.
- jedes Jahr 14.000 defekte PCs verkauft.
- jedes Jahr 2,5 Mio. Bücher mit einem falschen Schutzumschlag geliefert.
- täglich zwei unsichere Landungen auf dem Flughafen von Los Angeles stattfinden.
- in der nächsten Stunde 18.000 E-Mails ihren Adressaten verfehlen.

Quelle: K. - T. Cheng: "Lecture on VLSI Testing Techniques", University of California, Santa Barbara.

Die Qualität der ausgelieferten Chips ist ein wichtiges Verkaufsargument. Heute geforderte (und erreichte) Testqualitäten liegen im Bereich ppm (parts per million). 99,9 % fehlerfreie Chips sind also bei weitem zu wenig.

Test: Vollständiger Test kombinatorischer Schaltungen

Vollständiger Test kombinatorischer Schaltungen

Annahme: Ein Mikroprozessor mit 300 Signal-Eingängen sei rein kombinatorisch aufgebaut.

Dann ist die Anzahl möglicher Eingangskombinationen:

$$2_{300} = \text{ca. } 2 * 10_{90}$$

Nehmen wir weiter an, der Test wird mit einer Frequenz von 200 MHz ausgeführt.

Folgerung: Dann dauert der Test

$$\frac{2 * 10_{90}}{200 * 10_{6}s^{-1}} = \text{ca. } 3 * 10_{74} \text{ Jahre}$$

Ein vollständiger Test ist bereits bei kombinatorischen Schaltungen mit einer großen Anzahl von Eingängen unmöglich.

Test: Vollständiger Test sequentieller Schaltungen

Vollständiger Test sequentieller Schaltungen

Bei sequentiellen Schaltungen kommt zur Variation der Eingangsmuster die der internen Zustände hinzu.

Zahl der erforderlichen Tests:

2^{n+m}

n = Anzahl der Eingänge
m = Anzahl der Zustandbits

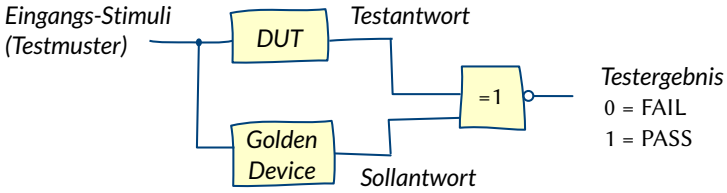
Beispiele:

D-Flipflop (data, reset, clock, enable) n = 4, m = 1	2^5 = 32 Tests
30-bit-Zähler (clock, reset) n = 2, m = 30	$2^{32} = 4 * 10^9$ Tests

Bei sequentiellen Schaltungen vervielfacht sich die Problemgröße durch die Anzahl der Zustände. Die Anzahl der notwendigen Test steigt auf 2n+m. n entspricht dabei der Anzahl der Eingänge und m der Breite der Zustandsvariable, die für die Kodierung der Zustände der sequentiellen Schaltung verwendet wird.

Test: Testdurchführung

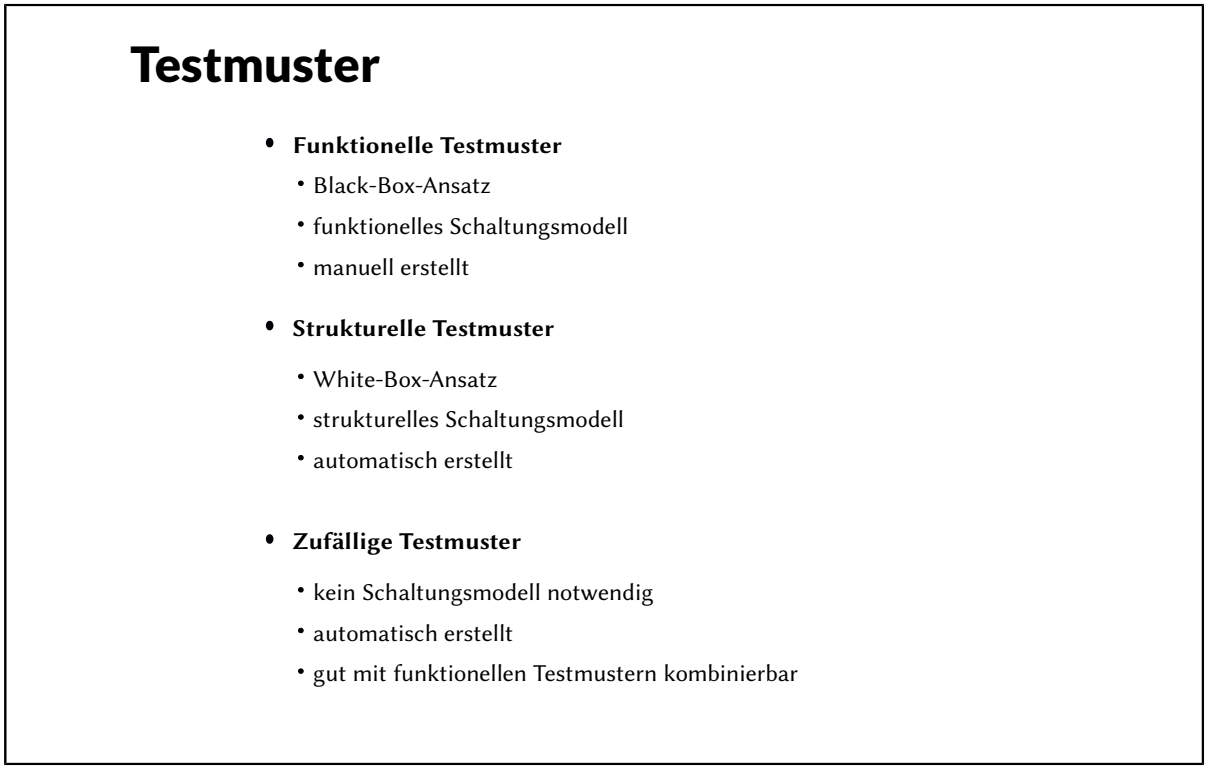
Testdurchführung



DUT: Device Under Test

Golden Device: Simulationsmodell oder bekannt fehlerfreie Schaltung

Test: Testmuster



Man unterscheidet drei Arten von Eingangsstimuli:

- **Funktionelle Testmuster**
  Der Entwickler erstellt, ausgehend von der Funktion des Prüflings, manuell Eingangsstimuli. Jede spezifizierte Funktion ist durch sie einmal anzusprechen. Je komplexer der Baustein ist, desto schwieriger gestaltet sich die manuelle Erstellung. Automatische Verfahren können nur in einfachen Fällen verwendet werden.
- **Strukturelle Testmuster**
  Die Eingangsstimuli werden entsprechend der logischen Schaltungsstruktur durch Stimuligeneratoren automatisch berechnet. Eine Reihe von Algorithmen (z.B. D-Algorithmus, PODEM-Algorithmus, FAN-Algorithmus) wurde dazu entwickelt. Die Schaltungskomplexität, besonders die sequentielle Tiefe, verschlechtert die Effizienz der Algorithmen. Durch geeignete Strukturmaßnahmen in der Schaltung oder durch Zusatzschaltungen kann die Effizienz jedoch wieder verbessert werden.
- **Zufällige Testmuster**
  Zufallstestmuster können auf einfache Weise generiert werden. Sie werden häufig als Ergänzung funktioneller Testmuster eingesetzt.

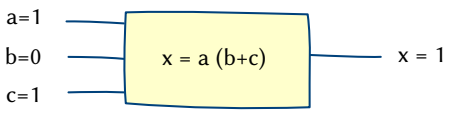
Wichtig ist es, eine Aussage zu treffen, welche Qualität die ausgewählten Testmuster besitzen, d.h. die Frage zu beantworten, welche Fehler erkannt werden können. Für ein internes Netz wird dabei berechnet, ob mit den vorhandenen Testmustern der Fehlerfall beobachtet werden kann. Zusätzlich wird ein Fehlerabdeckungsgrad als Maß für die Vollständigkeit der Fehlererkennung angegeben. Es errechnet sich aus der Anzahl der erkannten Fehler bezogen auf die Anzahl aller möglichen Fehler entsprechend dem zugrunde gelegten Modell. Zur Berechnung des Fehlerabdeckungsgrads werden so genannte Fehlersimulatoren eingesetzt.

Test: Funktionelle Testmuster

Funktionelle Testmuster

Funktionelle Testmuster basieren auf einer funktionellen Schaltungsbeschreibung.

Black-Box-Ansatz: Beschreibung des Ausgangsverhaltens in Abhängigkeit von der Eingangsbelegung.



Bei funktionellen Testmustern besteht die Gefahr, dass die Schaltung nur bezüglich einer beabsichtigten bzw. vorhersehbaren Funktion getestet wird. So bleiben leicht Fehler für "nicht vorhergesehene" Eingangsbelegungen unentdeckt.

Größter Nachteil funktioneller Testmuster ist, dass wegen eines fehlenden Fehlermodells keine Aussage über die Qualität einer Menge von Testmustern (Testmustersatz) gemacht werden kann.

Test: Strukturelle Testmuster

Strukturelle Testmuster

Strukturelle Testmuster basieren auf einer Strukturbeschreibung der Schaltung (White-Box-Ansatz).

Annahme von konkreten Fehlern in der Schaltung ⟶ Fehlermodelle erforderlich!

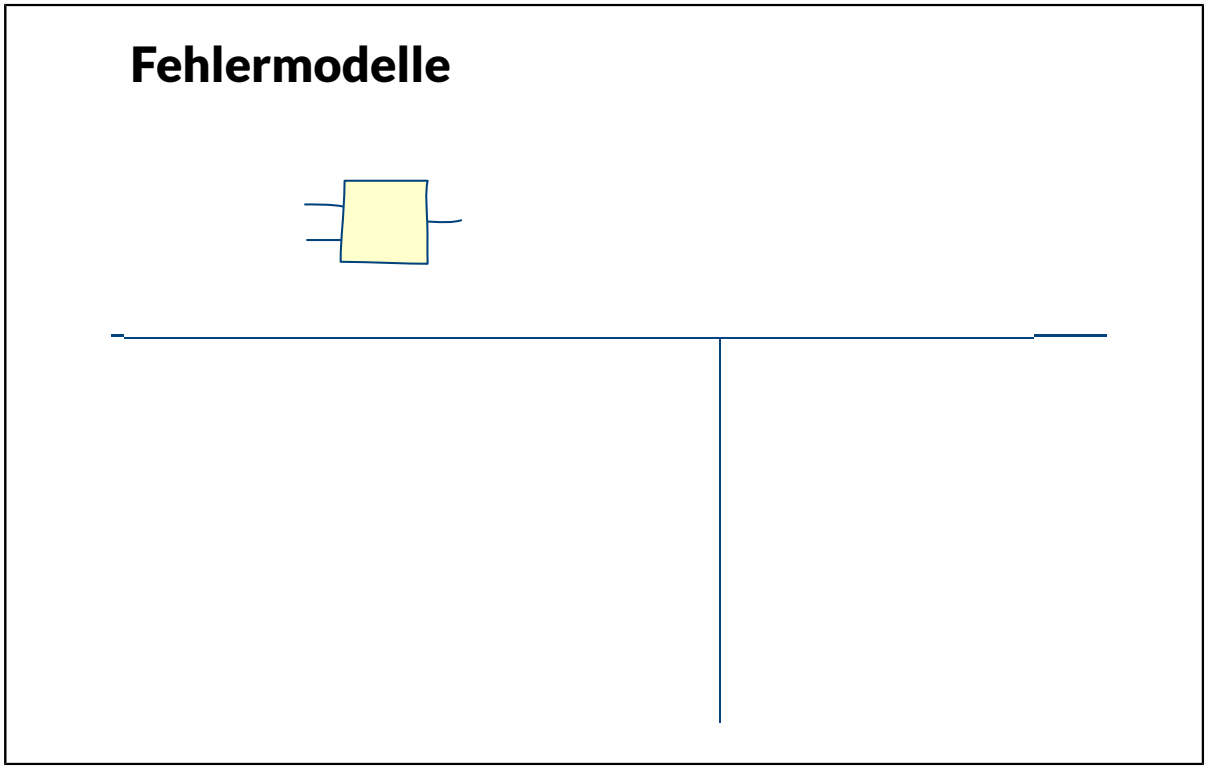
Qualität der strukturellen Testmuster ist messbar:

$$\text{Fehlerabdeckungsgrad} = \frac{\text{Anzahl der erkennbaren Fehler}}{\text{Anzahl aller möglichen Fehler gemäß des Fehlermodells}}$$

Strukturelle Testmuster werden erzeugt, indem entsprechend einem geeigneten Fehlermodell ein Fehler in der Schaltung an einem konkreten Ort angenommen wird und eine Eingangsbelegung berechnet wird, mit der dieser Fehler einstellbar und beobachtbar ist. Da hier tatsächlich einzeln Schaltungsfehler modelliert werden, spricht man auch von einem "defektorientierten" Test.

Die Qualität eines strukturellen Testmustersatzes ist über seinen Fehlerabdeckungsgrad messbar.

Test: Fehlermodelle



Ein Fehlermodell stellt eine Abstraktion physikalischer Fehlermechanismen dar. Ein spezifisches Modell deckt jeweils nur eine Untermenge von Fehlermechanismen ab, es gibt deshalb zahlreiche unterschiedliche Modelle, die sich teilweise auch überlappen.

Wegen der Komplexität des Testproblems benutzt man in der Regel sehr einfache Fehlermodelle, die sich jedoch in der Praxis bewährt haben. Das bekannteste ist das so genannte Haftfehlermodell (Stuck-at-0/1), das von folgenden Voraussetzungen ausgeht:

- Fehler treten nur in Verbindungsleitungen (Netzen) auf.
- Ein Fehler äußert sich stets in der Art, dass ein Netz ständig auf logisch 0 (stuck-at-0, sa 0) oder logisch 1 (stuck-at-1, sa 1) festgehalten wird.
- Die Schaltung enthält nur einen Fehler (Einzelhaftfehlermodell).

Das Haftfehlermodell ist das klassische Fehlermodell. Alle anderen Modelle beschreiben so genannte nicht-klassische Fehler, dazu gehören:

- Stuck-open-Fehler ("offener" Schaltungsknoten)
- Verzögerungsfehler (verzögerte Signalflanke)
- Übergangsfehler (ausbleibende Signalflanke)
- Stuck-on-Fehler (ständig eingeschalteter Transistor)
- Brückenfehler (Kurzschluss zwischen zwei Knoten)
- Crosstalk - Fehler (Fehler durch Übersprechen)

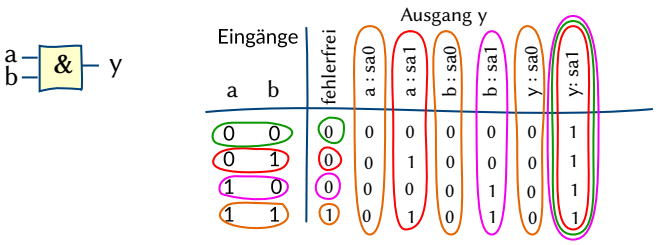
Haftfehler sind physikalisch gesehen Verbindungen zwischen einem Knoten und einer Versorgungsspannungsleitung (Masse, VDD). Das Haftfehlermodell hat folgende Vorteile:

- Wegen seiner Einfachheit lässt es sich leicht in Algorithmen zur automatischen Testmustergenerierung einbauen.

- Es modelliert eine relativ große Menge physikalischer Defekte.
- Es ist technologieunabhängig, da auf der Gatterebene angesiedelt.

Test: Fehlererkennung und Testmuster

Fehlererkennung und Testmuster



(a,b) = (0,1) erkennt die Fehler a:sa1 und y:sa1

(a,b) = (0,0) erkennt den Fehler y:sa1

(a,b) = (1,0) erkennt die Fehler b:sa1 und y:sa1

(a,b) = (1,1) erkennt die Fehler a:sa0, b:sa0 und y:sa0

a:sa0, b:sa0 und y:sa0 sind **äquivalente** Fehler, da jeder Test für einen von ihnen auch die anderen findet.

y:sa1 ist ein **dominanter** Fehler, da er auch von allen Testmustern für a:sa1 oder b:sa1 entdeckt wird.

Ein Fehler wird erkannt, wenn am Ausgang ein Wert festgestellt wird, der vom Wert der fehlerfreien Schaltung abweicht. Eine Eingangsbelegung, die die Erkennung eines Fehlers ermöglicht, heißt Testmuster. Ein Fehler kann durch mehrere Testmuster erkannt werden. Ein Testmuster kann mehrere Fehler erkennen. Dabei sind folgende Unterscheidungen wichtig:

"Fehler f1 dominiert Fehler f2, wenn jeder Test für Fehler f2 auch Fehler f1 entdeckt."

entsprechend:

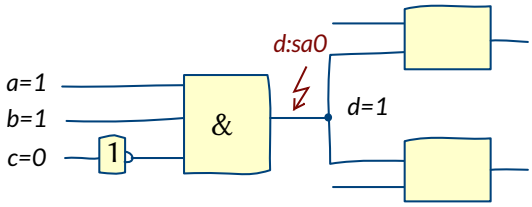
"Zwei Fehler f1 und f2 sind äquivalent, wenn jeder Test für Fehler f1 auch Fehler f2 findet, und umgekehrt."

Mit Hilfe dieser Fehlertypen kann die Liste aller zu testenden Fehler deutlich reduziert werden. Im Beispiel reicht es , für einen vollständigen Test, die Fehler a:sa0, a:sa1 und b:sa1 zu berücksichtigen. Ein Testmustersatz, der alle Fehler erkennt, wäre dann (a, b)={(0,1),(1,0),(1,1)}.

Test: Einstellbarkeit (Steuerbarkeit)

Einstellbarkeit (Steuerbarkeit)

Einstellbarkeit bezeichnet die Möglichkeit, ein Netz auf einen definierten Wert (0/1) setzen zu können:



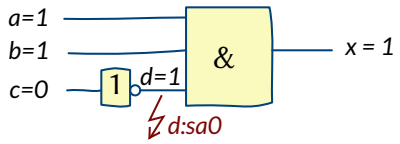
Der Testvektor $\mathbf{x}_T$=(a,b,c)=(1,1,0) zwingt d auf logisch 1.
d=1 ist also einstellbar.
d=1 braucht man z.B. für einen Fehler d:sa0.

Wenn ein Fehler testbar sein soll, muss er einstellbar und beobachtbar sein. Einstellbarkeit (Steuerbarkeit) heißt, es gelingt durch eine Eingangsbelegung ein Netz auf einen definierten Wert einzustellen. Insbesondere gelingt es damit, am Fehlerort einen vom Fehler abweichenden Wert einzustellen. Im Beispiel wäre dies für den Fehler sa0 am Ausgang d des UND-Gatters der Fall.

Test: Beobachtbarkeit

Beobachtbarkeit

Beobachtbarkeit bezeichnet die Möglichkeit, den Wert eines Netzes
am Ausgang einer Schaltung beobachten zu können.



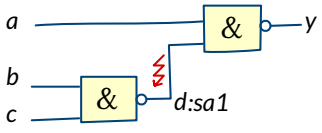
Der Eingang c=0 stellt hinter dem Inverter den Wert d=1 ein (Einstellbarkeit).
Dieser Wert ist durch den Testvektor $\mathbf{x}_T$=(a,b,c)=(1,1,0) am Ausgang y beobachtbar.
Die Werte für a=1 und b=1 sorgen also hier für die Beobachtbarkeit.

Beobachtbarkeit heißt, dass durch eine Eingangsbelegung ein Netz am Ausgang beobachtet werden kann. Insbesondere gelingt es damit, einen fehlerhaften Wert eines Netzes zum Ausgang zu propagieren.

Bei redundanzfreier kombinatorischer Logik ist grundsätzlich jeder Fehler steuer- und beobachtbar.

Test: Beispiel: Fehlererkennung

Beispiel: Fehlererkennung



Eingänge			d		Ausgang y	
a	b	c	fehlerfrei	d:sa1	fehlerfrei	d:sa1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	1	0	1	1	1	1
0	1	1	0	1	1	1
1	0	0	1	1	0	0
1	0	1	1	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	1	0

- 8 mögliche Eingangskombinationen
- Einstellbarkeit am Fehlerort erfordert b = c = 1
- Beobachtbarkeit am Ausgang ist nur für die Eingangsbelegung (a, b, c) = (1, 1, 1) gegeben.

Die Schaltung ermöglicht 8 verschiedene Eingangskombinationen.

Einstellbarkeit am Fehlerort erfordert b = c = 1.

Beobachtbarkeit am Ausgang ist nur für die Eingangsbelegung (a, b, c) = (1, 1, 1) gegeben (gültiges Testmuster).

Der Fehler sa0 am gleichen Ort wäre mit mehreren Testmustern testbar.

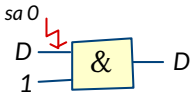
Test: D-Algorithmus

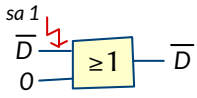
D-Algorithmus

D-Notation:

Logikwerte (fehlerfrei/fehlerhaft):

D = 1/0; $\overline{D}$ = 0/1; 0 = 0/0; 1 = 1/1; X = X/X;



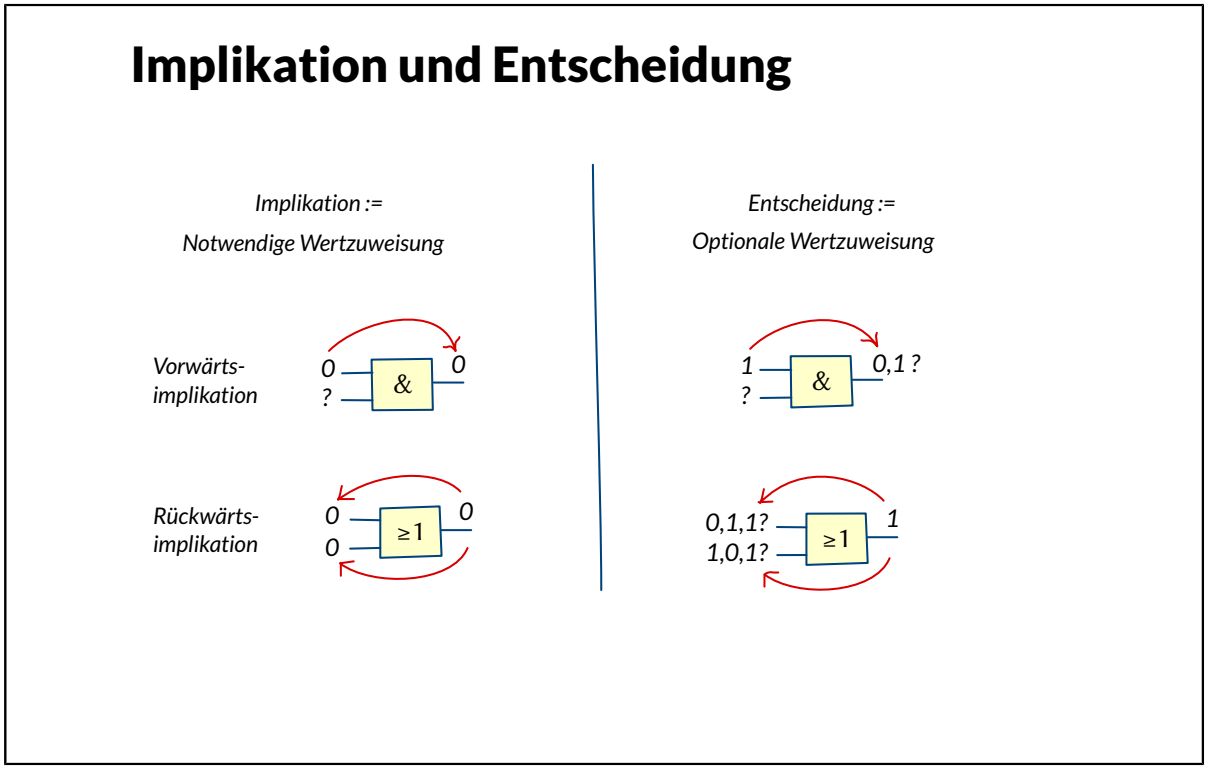


AND	0	1	X	D	$\overline{D}$
0	0	0	0	0	0
1	0	1	X	D	$\overline{D}$
X	0	X	X	X	X
D	0	D	X	D	0
$\overline{D}$	0	$\overline{D}$	X	0	$\overline{D}$

OR	0	1	X	D	$\overline{D}$
0	0	1	X	D	$\overline{D}$
1	1	1	1	1	1
X	X	1	X	X	X
D	D	1	X	D	1
$\overline{D}$	$\overline{D}$	1	X	1	$\overline{D}$

Für die automatische Generierung von Testmustern hat sich ein heute in zahlreichen Varianten verfügbarer Ansatz durchgesetzt, der auf das so genannte D (Discrepancy)-Kalkül von Roth zurückgeht. Dort wird für ein Signal, welches im fehlerfreien Fall den Wert '1' und im fehlerhaften Fall den Wert '0' besitzt (sa 0), der logische Wert 'D' notiert. Verhält es sich umgekehrt, d.h. ist der Wert im fehlerfreien Fall '0' und im fehlerhaften Fall '1', so wird dem Signal der logische Wert 'not D' (gesprochen "D quer") zugewiesen (sa 1). Ist der logische Wert für den fehlerfreien und fehlerhaften Fall gleich, nämlich '1' oder '0', so hat das Signal den Wert '1' bzw. '0'. Man erhält auf diese Weise eine fünfwertige Logik mit den Werten 0, 1, X, D und not D.

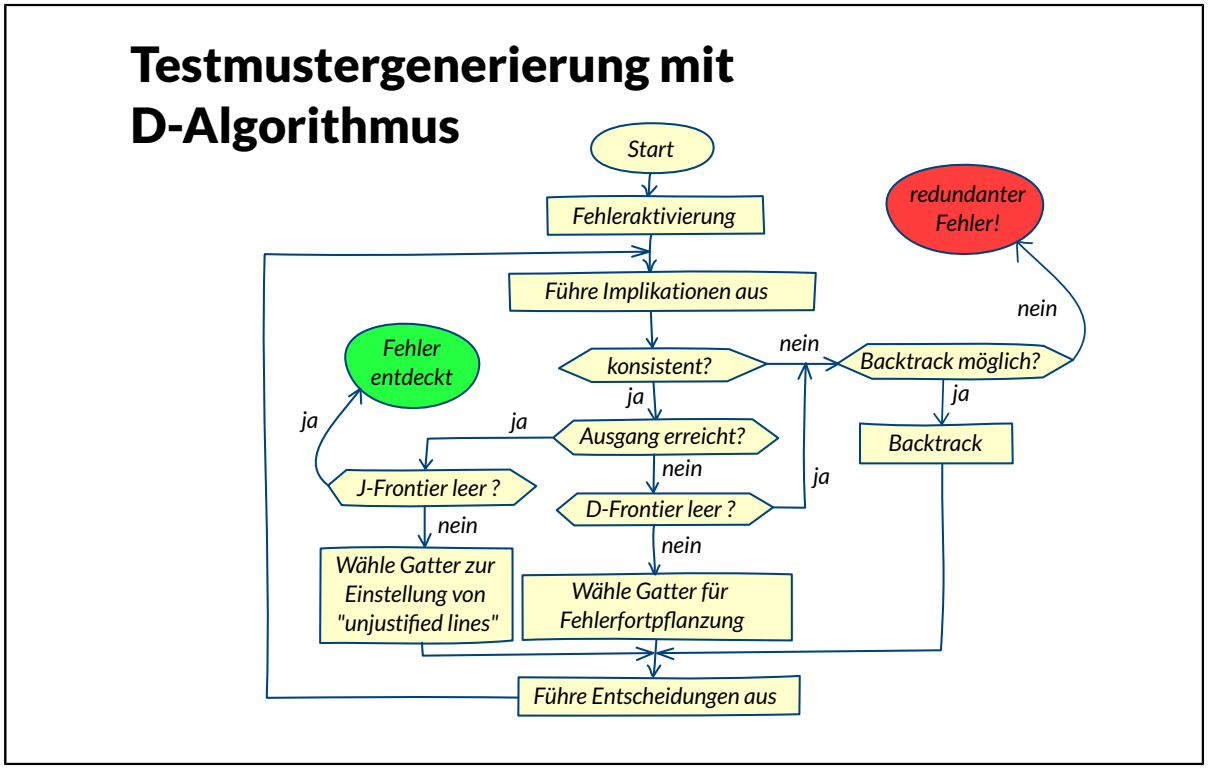
Test: Implikation und Entscheidung



Bei der Analyse logischer Schaltungen für die Testmustererzeugung erfolgen Wertzuweisungen. Es gibt zwei unterschiedliche Varianten:

- Implikationen sind notwendige Wertzuweisungen, die sich zwangsläufig und eindeutig ergeben.
- Entscheidungen sind optionale Wertzuweisungen, die Werte zuweisen, welche falsch sein können, d.h. die einen Konflikt herbeiführen können.

Test: Testmustergenerierung mit D-Algorithmus



Die Testmustergenerierung beginnt mit dem Rücksetzen aller Signale auf X (unbekannt) und dem Setzen des Signals am Fehlerort auf D (sa0) bzw. not D (sa1), der so genannten Fehleraktivierung. Nachdem das Fehlersignal gesetzt worden ist, werden alle möglichen logischen Implikationen ausgeführt, d.h. Wertzuweisungen, die sich aus dem Setzen des Fehlersignals eindeutig ergeben. Daraus ergibt sich eine Wertebelegung, die Ausgangspunkt für alle weiteren Schritte der Testmustergenerierung bildet. Bei den im Verlaufe der Testmustergenerierung entstehenden Wertebelegungen interessieren uns stets zwei Mengen von Netzen:

Die **D-frontier** (D-drive) besteht aus denjenigen Gattern, deren Wert am Ausgang (noch) unbestimmt ist, die aber ein oder mehrere Fehlersignale (D oder not D) an den Eingängen besitzen. Die D-frontier gibt an, wie weit sich fehlerhafte Signale in Richtung der primären Ausgänge ausgebreitet haben.

Die **J-frontier** besteht aus allen Gattern, deren Ausgänge einen festen logischen Wert besitzen, der sich jedoch (noch) nicht aus den Werten an den Gatter-Eingängen implizieren lässt. Die Ausgänge der J-frontier bezeichnet man oft auch als unjustified lines. Sie stellen Netze dar, an denen sich durch Wertzuweisungen im Inneren der Schaltung feste logische Werte ergeben haben, die aber noch nicht durch geeignete Wahl von Wertzuweisungen an den primären Eingängen eingestellt sind.

Die Aufgabe eines deterministischen Testmustergenerators lässt sich damit auch so beschreiben: Finde eine binäre Wertebelegung für die primären Eingänge der Schaltung, so dass

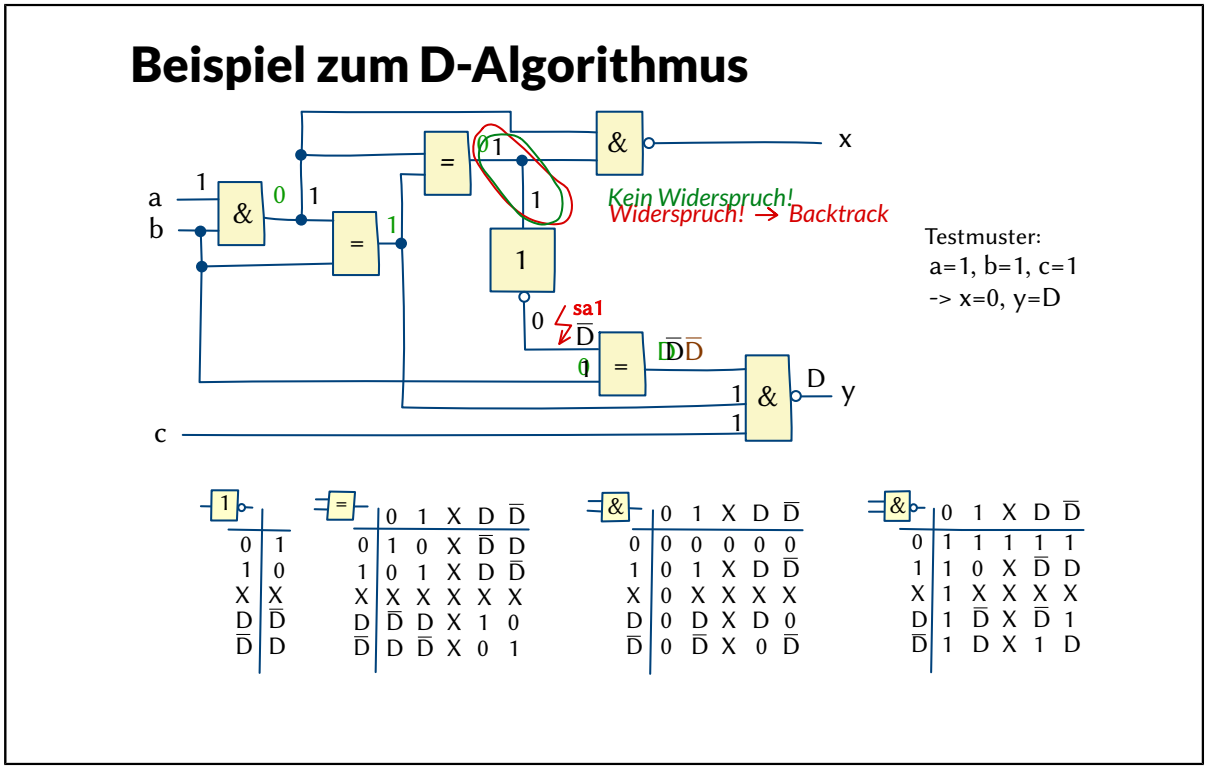
- die D-frontier einen primären Ausgang erreicht, d.h. wenigstens an einem primären Ausgang ein Fehlersignal anliegt (Beobachtbarkeit) und
- die J-frontier die primären Eingänge erreicht, d.h. im Inneren der Schaltung keine unjustified lines mehr existieren (Einstellbarkeit).

Der vorgestellte Algorithmus zur deterministischen Testmustergenerierung geht dabei so vor, dass er Schritt für Schritt durch geeignete Wertzuweisungen an geschickt ausgewählten Netzen die D-frontier in Richtung der primären Ausgänge und die J-frontier in Richtung der primären Eingänge verschiebt. Der Vorgang der Testmustererzeugung kann als eine Abfolge von optionalen und notwendigen Wertzuweisungen verstanden werden. Wie die notwendigen Wertzuweisungen bestimmt und wie die

optionalen Wertzuweisungen geschickt ausgewählt werden, variiert von Algorithmus zu Algorithmus und bestimmt die Leistungsfähigkeit der verschiedenen Tools. Dennoch ist der grundsätzlicher Ablauf bei allen herkömmlichen Verfahren weitgehend gleich.

Führt der Algorithmus zu einem inneren Widerspruch, der sich nicht mehr durch einen Backtrack (also eine optionale Wertzuweisung) auflösen lässt, hat man einen **redundanten Fehler** gefunden. Dies sollte gar nicht oder nur sehr selten vorkommen, da durch den Algorithmus bewiesen ist, dass es keine Belegung an den Eingängen geben kann, so dass der Fehler am Ausgang sichtbar ist. Solch redundante Fehler werden bei Auftreten oft durch geeignete Redesign-Maßnahmen entfernt.

Test: Beispiel zum D-Algorithmus



1	
0	1
1	0
X	X
D	$\bar{D}$
$\bar{D}$	D

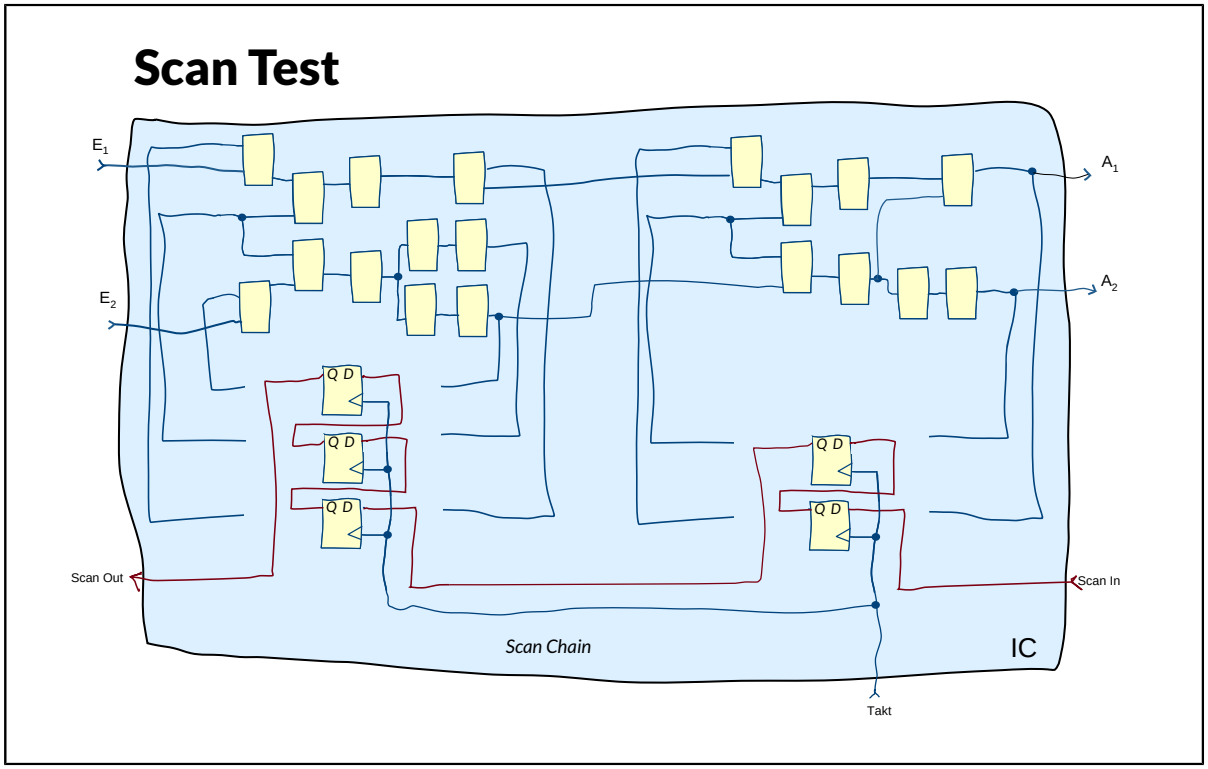
=	0	1	X	D	$\bar{D}$
0	1	0	X	$\bar{D}$	D
1	0	1	X	D	$\bar{D}$
X	X	X	X	X	X
D	$\bar{D}$	D	X	1	0
$\bar{D}$	D	$\bar{D}$	X	0	1

&	0	1	X	D	$\bar{D}$
0	0	0	0	0	0
1	0	1	X	D	$\bar{D}$
X	0	X	X	X	X
D	0	D	X	D	0
$\bar{D}$	0	$\bar{D}$	X	0	$\bar{D}$

&	0	1	X	D	$\bar{D}$
0	1	1	1	1	1
1	1	0	X	$\bar{D}$	D
X	1	X	X	X	X
D	1	$\bar{D}$	X	$\bar{D}$	1
$\bar{D}$	1	D	X	1	D

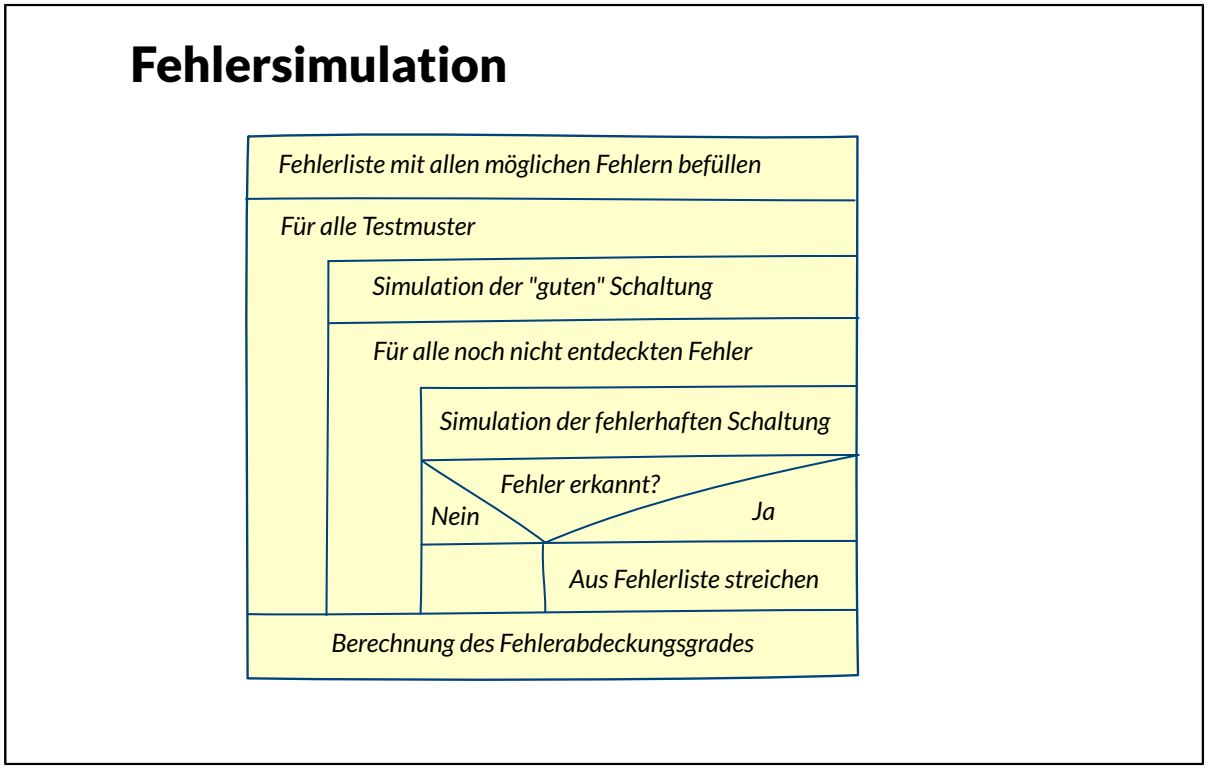
Im hier dargestellten Beispiel nehmen wir einen sa1-Fehler an und weisen dementsprechend das Fehlersignal not D zu. Die Animation zeigt, wie Wertzuweisungen innerhalb der Schaltung und an den Eingängen erfolgen und schließlich ein Testmuster erzeugt wird, mit dem der Fehler steuerbar (0 am Fehlerort) und beobachtbar (D an y) ist.

Test: Scan Test



Der Scan-Test dient dazu, während des Tests alle Werte von internen Flip-Flops nach außen zu führen. Die Flips-Flops sind dazu untereinander verkettet (Scan Chain) und das Ende Kette über einen Pin nach außen geführt. Dadurch können die Werte seriell nach außen zum Test geführt werden. Die Flip-Flops müssen dazu in einen sogenannten Shift-Betrieb geschaltet werden. Das Verketten der Scan Chain wird von speziellen EDA-Werkzeugen automatisiert durchgeführt.

Test: Fehlersimulation



Fehlersimulation dient dazu, die Qualität von Testmustersätzen zu bewerten. Dazu muss festgestellt werden, ob bei Annahme eines bestimmten Fehlers in der Schaltung ein Eingangsmuster geeignet ist, an einem Ausgang eine Veränderung gegenüber der fehlerfreien Schaltung hervorzurufen. Dies kann durch eine logische Simulation festgestellt werden.

Der Fehlerabdeckungsgrad ist der Quotient aus Anzahl gefundener Fehler und Anzahl aller Fehler gemäß Fehlermodells.

Ein Problem besteht in der Vielzahl möglicher Fehler (und Eingangsmuster), die zur Erreichung tragbarer Simulationszeiten parallele Methoden erforderlich macht. Parallele Methoden können mit einem Rennen verglichen werden, in dem eine beliebige Zahl von Teilnehmern (fehlerhafte Schaltungen) gegeneinander laufen. Mit einem normalen Logiksimulator lässt sich exakt dasselbe Ergebnis erzielen, aber jeder Teilnehmer muss einzeln laufen. Es müssen für N Fehler also N+1 Simulationsläufe durchgeführt werden, je einen für jeden Fehler und einen zusätzlichen für die fehlerfreie Schaltung. Neben der Zeiteinsparung haben parallele Methoden auch den Vorteil, dass sich der Vergleich zwischen fehlerhafter und korrekter Schaltung stark vereinfachen lässt.

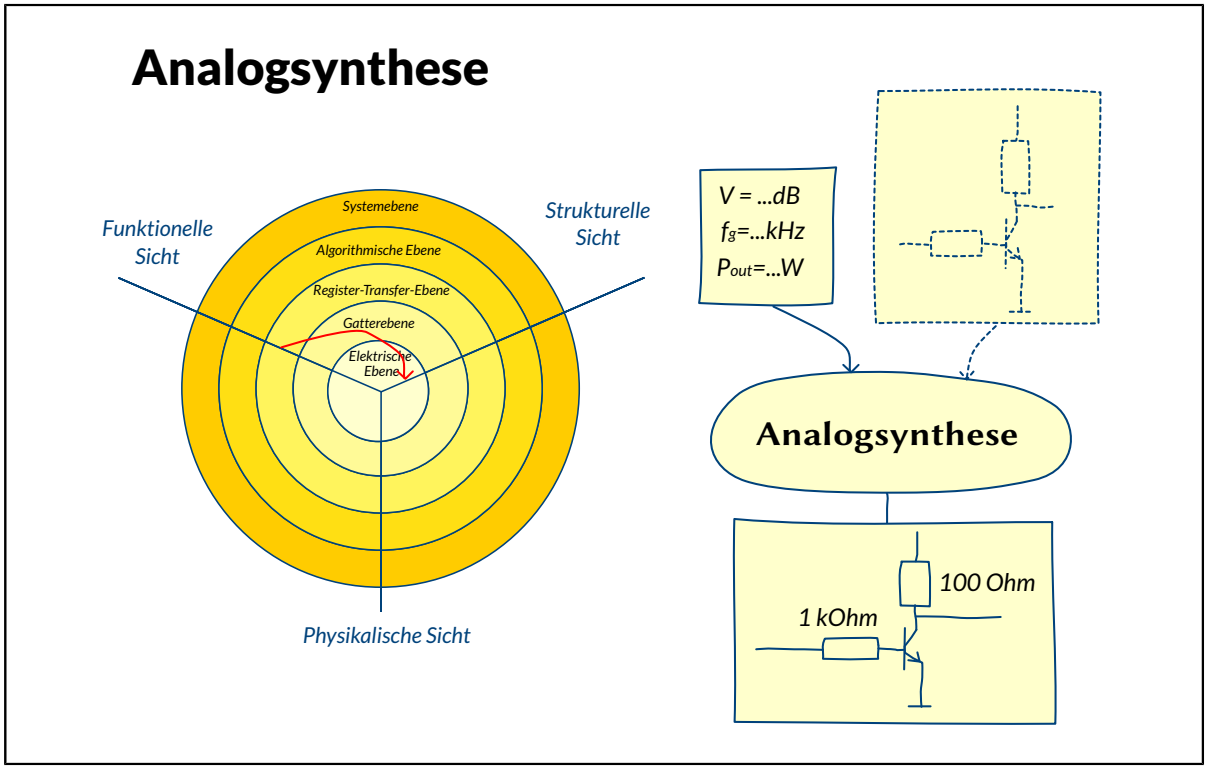
Die bekannten parallelen Algorithmen werden auch als simultane, deduktive und konkurrierende Verfahren bezeichnet. Dabei nutzen alle die Eigenschaft, dass im Normalfall die fehlerhafte Schaltung nur geringfügig von der korrekten abweicht.

Electronic Design Automation (EDA)

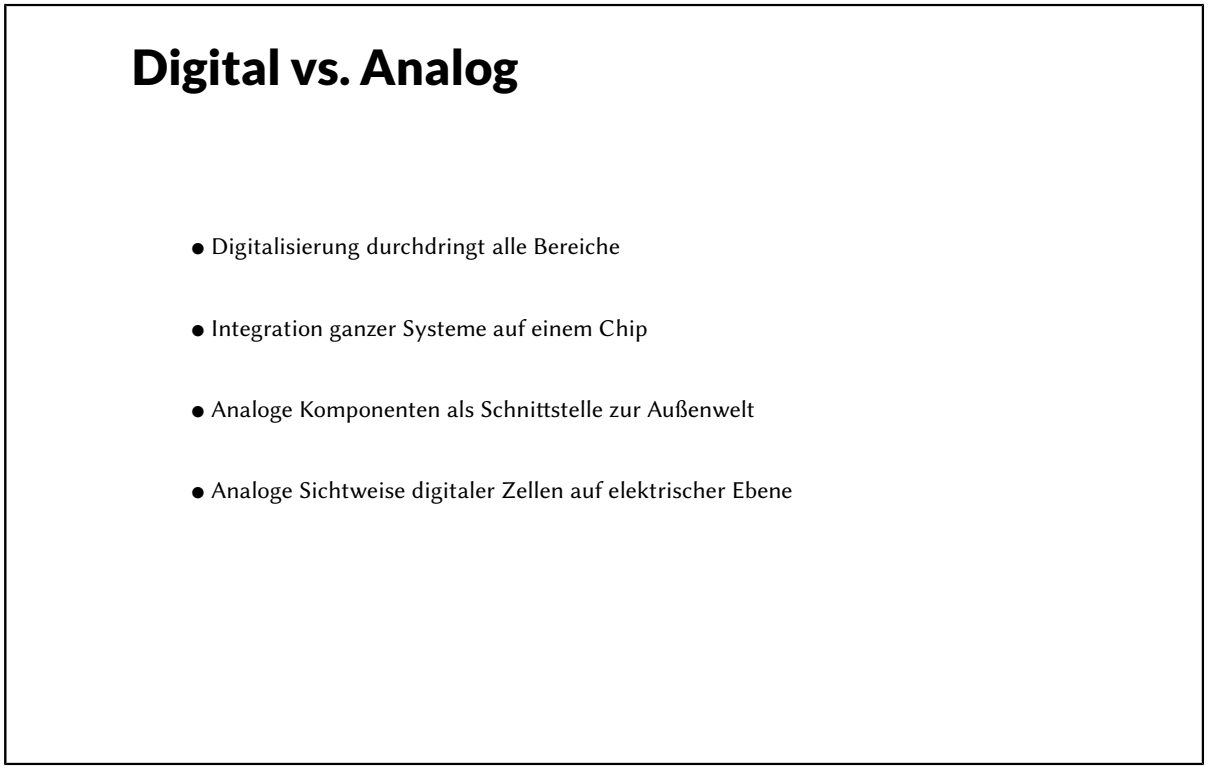
Analogsynthese

Analoge Synthese
Digital vs. Analog
Analoge Schnittstellen
Synthesewerkzeuge für den Analogschaltungsentwurf
Digital vs. Analogschaltungsentwurf
Topologiesynthese und Dimensionierung
Topologiesynthese
Schaltungsparameterraum
... Beispiel
Schaltungseigenschaftsraum
... Beispiel
Akzeptanzgebiet
... Beispiel
Analyseabbildung
Dimensionierungsabbildung
Abbildung des Akzeptanzgebietes
... Beispiel
Dimensionierungsproblem
Beispiel Dimensionierung durch Optimierung
Ausbeuteoptimierung
Entwurfszentrierung
Dimensionierungsverfahren
Wissensbasierte Verfahren
Beispiel eines Entwurfsplanes
Beispiel eines Entwurfsplanes II
Anwendung des Entwurfsplanes
Optimierungsbasierte Verfahren
Optimierer-Analysewerkzeug-Kopplung
Optimierer und Analysewerkzeuge

Analogsynthese: Analoge Synthese



Analogsynthese: Digital vs. Analog



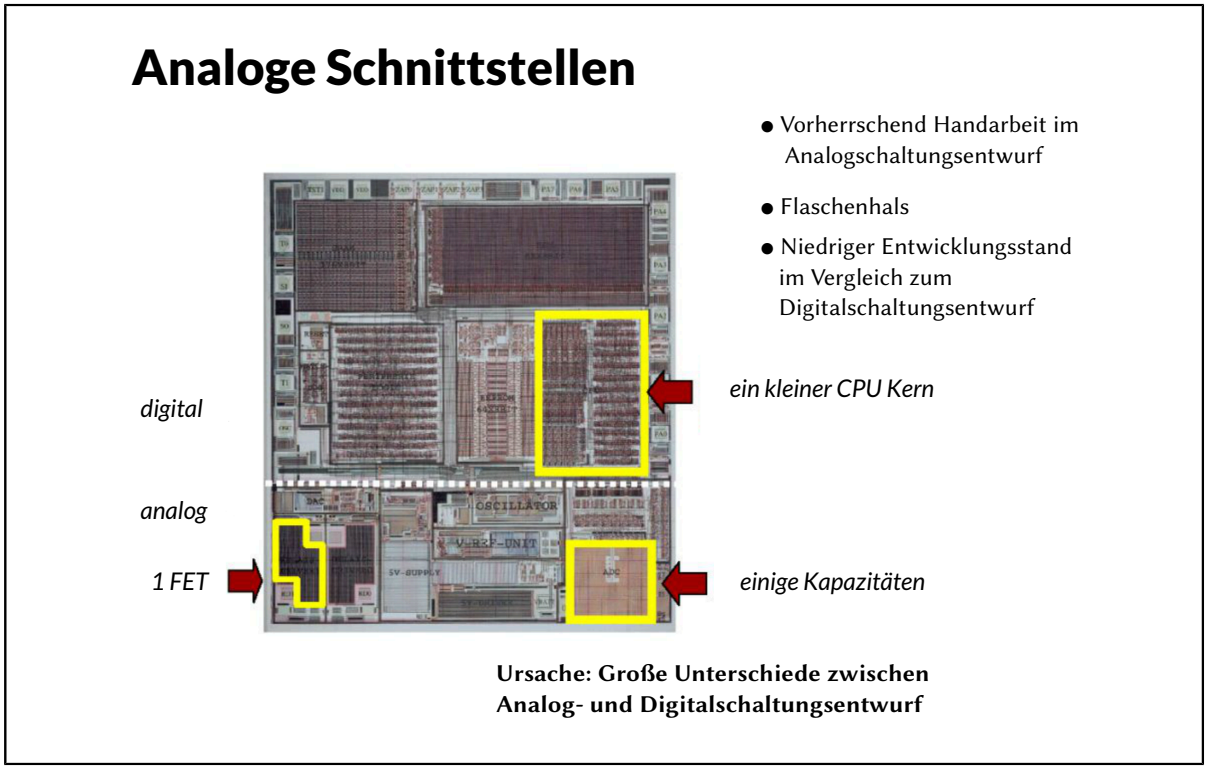
Trotz der voranschreitenden Digitalisierung haben analoge Komponenten große Bedeutung. Dieses trifft vor allem für die heute verbreitete Integration ganzer Systeme auf einem Chip, so genannter Systems-on-Chip (SOC) zu. Analoge Schaltungsteile werden vor allem als Schnittstellen zur Außenwelt benötigt. Auch digitale (Standard-)Zellen werden als analoge Schaltungen betrachtet und ihr Entwurf bzw. ihre Optimierung wird auf elektrischer Ebene durchgeführt.

Analogsynthese: Analoge Schnittstellen



Beispiele für analoge Schnittstellen finden sich z. B. in der Kommunikations- und Automobilelektronik sowie in der Medizintechnik: RF-Frontends in Mobiltelefonen, Beschleunigungssensoren mit A/D-Wandlern in Airbag-Auslösesteuerungen oder ESP-Systemen, DC/DC-Wandler zum Anschluss elektronischer Komponenten an Bordspannungsversorgungen, Messverstärker zur EKG-Bestimmung in Herzschrittmachern. Da die Außenwelt grundsätzlich analog ist, wird sich an der Notwendigkeit analoger Blöcke an den Schnittstellen auch in Zukunft nichts ändern.

Analogsynthese: Synthesewerkzeuge für den Analogschaltungsentwurf



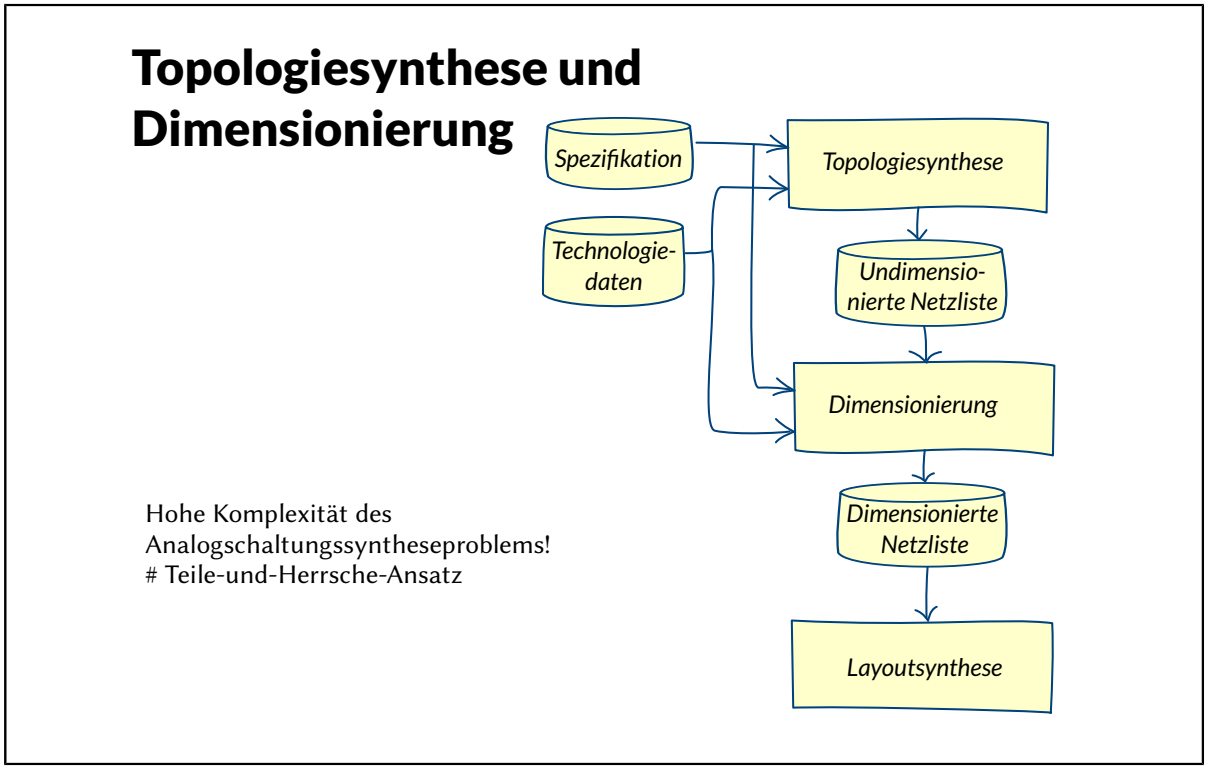
Während zur Erstellung von Digitalschaltungen leistungsfähige Synthesewerkzeuge zur Verfügung stehen, sind die Fähigkeiten von Werkzeugen zum automatischen Entwurf von Analogschaltungen bis heute vergleichsweise bescheiden. In der Praxis wird der Entwurf fast ausschließlich von erfahrenen Schaltungsentwicklern per Hand durchgeführt. Dadurch kommt es bei Systemen mit analogen und digitalen Blöcken vielfach zu einem Flaschenhals beim Analogentwurf: Wie auf dem Bild gezeigt, nimmt der Analogteil meist nur einen geringen Teil der Gesamtchipfläche ein und enthält nur vergleichsweise wenige Bauelemente. Trotzdem benötigt er ein Vielfaches der Entwurfszeit des Digitalteils. Der Bedarf an EDA-Werkzeugen für den automatischen Analogentwurf ist daher hoch. Der Grund für den geringen Erfolg bei der Automatisierung des Analogschaltungsentwurfs durch Synthesewerkzeuge im Vergleich zum Digitalschaltungsentwurf liegt in den grundsätzlichen Unterschieden zwischen Digital- und Analogschaltungsentwurf.

Analogsynthese: Digital vs. Analogschaltungsentwurf

Digital- vs. Analogschaltungsentwurf

	Digitalschaltungsentwurf	Analogschaltungsentwurf
Schaltungsgröße in Transistoren	Millionen	Dutzende
Typischer Entwurf eines Transistor	Minimalabmessungen	Optimierung unter Verwendung aller Freiheitsgrade
Vorherrschender Entwurfsstil	automatisiert	handoptimiert
Wieder-verwendbarkeit	Standardzellen	Wiederverwendbare Topologien, die in der Regel individuell angepasst werden müssen
Hierarchie	Weitgehende Abstraktion	Wenige Abstraktionsmöglichkeiten
Spezifikation	Synthesefähige Sprache	Schaltungsklasse + Schaltungseigenschaften

Analogsynthese: Topologiesynthese und Dimensionierung



Bei der analogen Schaltungssynthese sind eine Netzliste und die Schaltungsparameter bei gegebener Spezifikation und Technologiedaten gesucht. Dies stellt ein gemischt-kombinatorisch-nichtlineares Optimierungsproblem (MINLP, mixed-integer-nonlinear programming problem) dar. Zur Reduktion der hohen Komplexität wird es nach dem Teile und Herrsche-Prinzip in zwei Teilprobleme zerlegt: Topologiesynthese (kombinatorisches Optimierungsproblem, IP) und Dimensionierung (nichtlineares Optimierungsproblem, NLP). Dadurch muss auf Optimalität bei der Lösung des Schaltungssyntheseproblems verzichtet werden.

Analogsynthese: Topologiesynthese

Topologiesynthese

Topologieselektion

- Bibliotheksansatz
- Unflexibel

Topologiegenerierung

- Konstruktiv, "kreativ"
- Genetische Optimierung o. ä.
- Ausschließlich Forschung

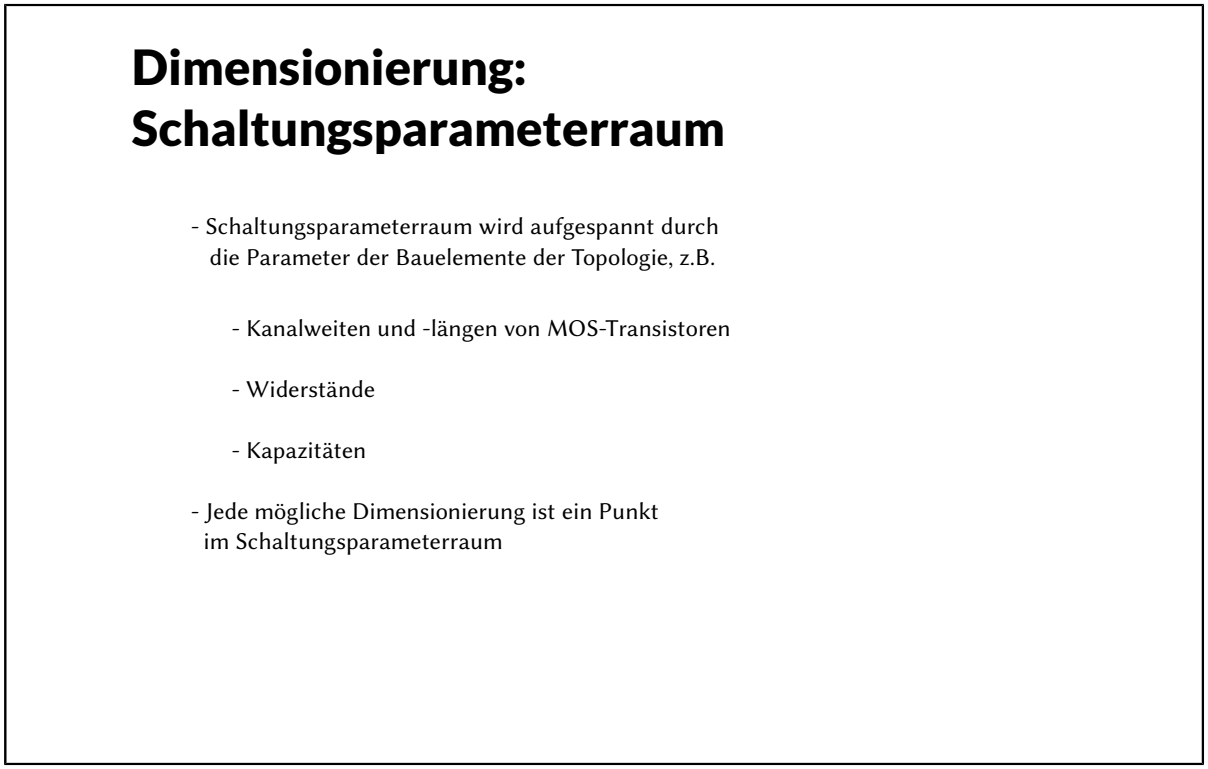
Problem: Topologiebewertung OHNE Dimensionierung

Die Topologieselektion ist ein Bibliotheksansatz zur Lösung des Topologiesyntheseproblems, der auf der Wiederverwendung (Reuse) von einmal erstellten Topologien, teilweise unter Verwendung von Hierarchie aufbaut. Der Nachteil eines solchen Ansatzes besteht darin, dass die Bibliothek starr ist. Sie ist infolge des rasanten technologischen Wandels sehr schnell veraltet und unbrauchbar. Zusätzlich fällt der hohe Erstellungsaufwand einer umfassenden Bibliothek an.

Ein automatisches Topologiesyntheseverfahren muss in der Lage sein, neuartige Topologien zu generieren, wenn der technologische Fortschritt dies erfordert. Dies kann eine Topologieselektion mit Bibliotheksansatz nicht leisten. Es ist stattdessen ein konstruktiver, d. h. kreativer, Syntheseprozess erforderlich. Algorithmen zur konstruktiven Topologiesynthese sind bisher jedoch ausschließlich Gegenstand der Forschung. Veröffentlichte Ansätze stützen sich häufig auf genetische Optimierungsverfahren.

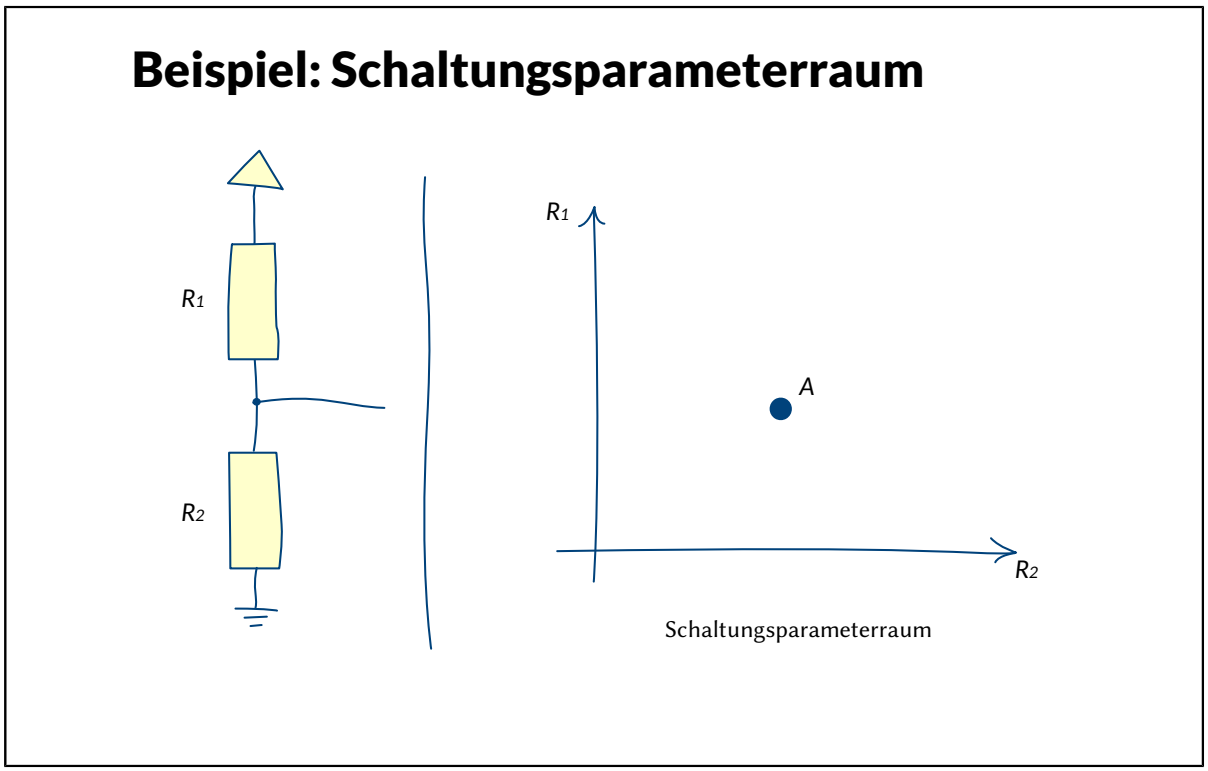
Sowohl bei der Topologieselektion als auch bei der konstruktiven Topologiegenerierung besteht die Schwierigkeit in der Bewertung der Topologien hinsichtlich der in der Spezifikation geforderten Eigenschaften und der Auswahl der besten. Dabei darf nach dem Teile und Herrsche-Prinzip der Dimensionierungsschritt nicht ausgeführt werden.

Analogsynthese: Schaltungsparameterraum



Der Schaltungsparameterraum wird durch alle im Entwurf zu bestimmenden Bauelementeparameter gebildet. Dies sind z. B. die Kanalweiten und -längen von MOS-Transistoren und die Werte von Widerständen und Kapazitäten. Damit entspricht jeder möglichen Dimensionierung genau ein Punkt im Schaltungsparameterraum.

Analogsynthese: ... Beispiel



Das Bild zeigt einen aus zwei Widerständen R1 und R2 bestehenden Spannungsteiler. Für diesen sind R1 und R2 die Bauelementeparameter, die bestimmt werden sollen. Hieraus ergibt sich der dargestellte, zweidimensionale Schaltungsparameterraum. Punkt A repräsentiert eine mögliche Dimensionierung, durch die für R1 und R2 Zahlenwerte festgelegt sind.

Für komplexere Schaltungen ergeben sich eine höhere Anzahl von Dimensionen für den Schaltungsparameterraum.

Analogsynthese: Schaltungseigenschaftsraum

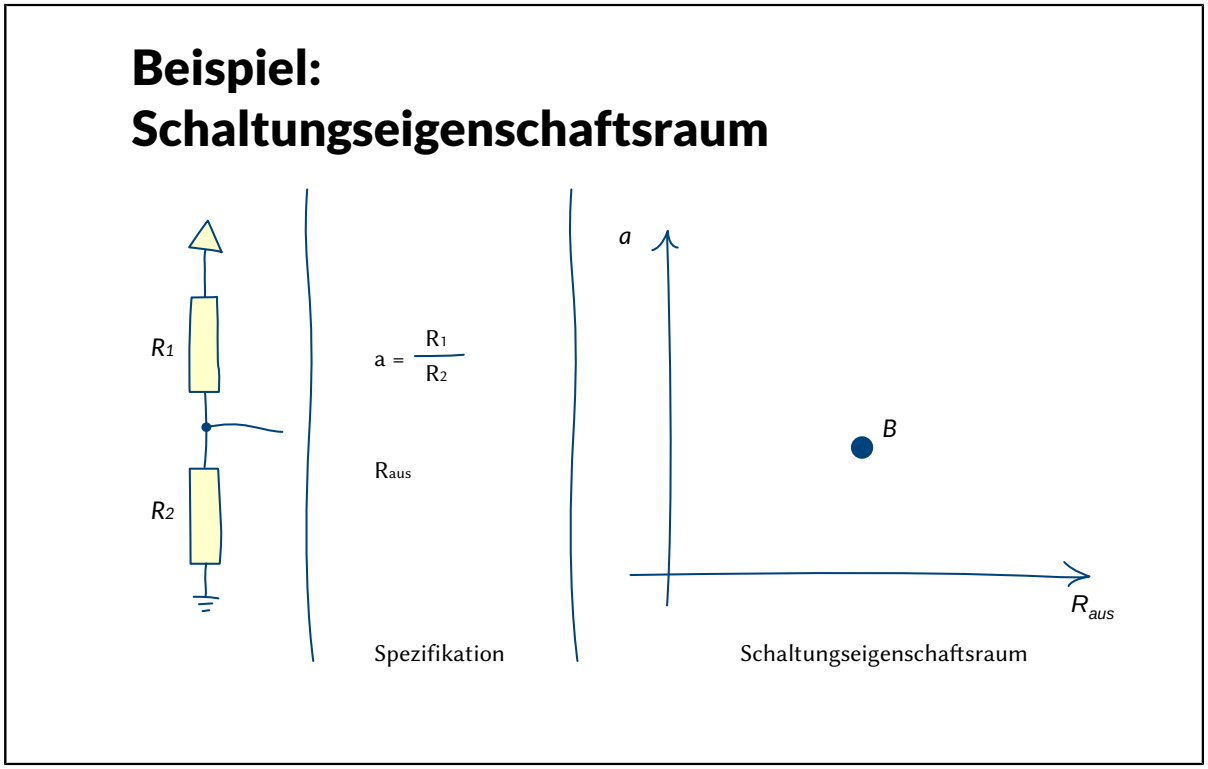
Dimensionierung: Schaltungseigenschaftsraum

- Schaltungseigenschaftsraum wird aufgespannt durch spezifizierte Eigenschaftsgrößen der Schaltung, z.B.
  - Verstärkung
  - Bandbreite
  - Rauschen
  - Stromverbrauch
  - Fläche etc.

- Zu jeder möglichen Dimensionierung gehört ein Punkt im Schaltungseigenschaftsraum.

Der Schaltungseigenschaftsraum wird von den in der Spezifikation verwendeten Eigenschaftsgrößen aufgespannt. Typische Eigenschaftsgrößen bei Analogschaltungen sind Verstärkung, Bandbreite, Rauschen, Leistungsverbrauch und Chipflächenbedarf. Für jede mögliche Dimensionierung einer Schaltung ergibt sich genau ein Punkt im Schaltungseigenschaftsraum.

Analogsynthese: ... Beispiel



Das Bild zeigt erneut den aus zwei Widerständen R1 und R2 bestehenden Spannungsteiler. Die spezifizierten Eigenschaftsgrößen seien das Teilungsverhältnis a und der Ausgangswiderstand Raus. Sie bilden den dargestellten Schaltungseigenschaftsraum. Der Punkt B zeigt die Eigenschaftsgrößen, die sich durch eine bestimmte Dimensionierung ergeben.

Analogsynthese: Akzeptanzgebiet

Akzeptanzgebiet

- Die Spezifikation gibt Grenzen für Eigenschaften vor.
  Sie beschreiben das erlaubte Gebiet im Schaltungseigenschaftsraum,
  das sogenannte Akzeptanzgebiet.

- Jede mögliche Dimensierung entspricht einem Punkt
  im Schaltungseigenschaftsraum. Liegt der Punkt

  - innerhalb des Akzeptanzgebietes

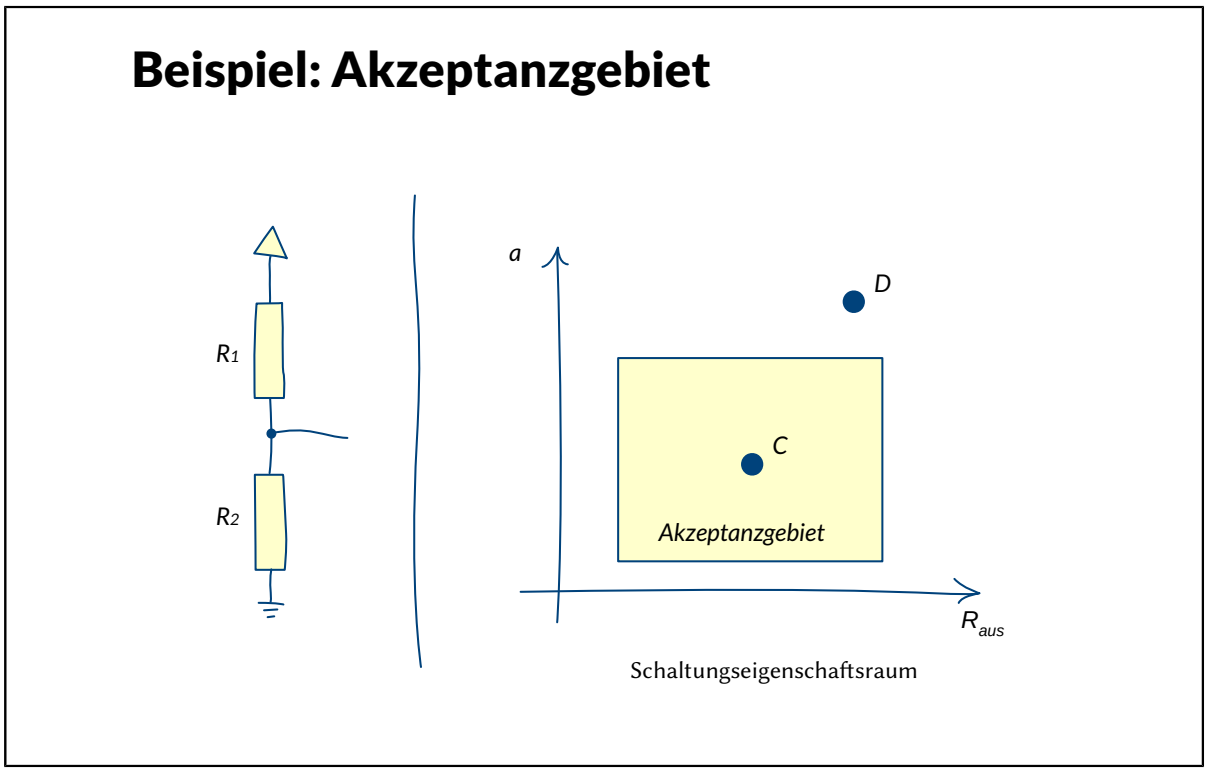
    => Spezifikation erfüllt

  - außerhalb des Akzeptanzgebietes

    => Spezifikation verletzt

Die Spezifikation gibt Unter- und Obergrenzen für die Schaltungseigenschaftsgrößen an. Dadurch wird im Schaltungseigenschaftsraum ein erlaubtes Gebiet für die Werte der Eigenschaftsgrößen definiert, das so genannte Akzeptanzgebiet. Liegt der einer bestimmten Dimensionierung entsprechende Eigenschaftspunkt im Schaltungseigenschaftsraum innerhalb des Akzeptanzgebietes, ist die Spezifikation für diese Dimensionierung erfüllt. Liegt er außerhalb, ist sie verletzt.

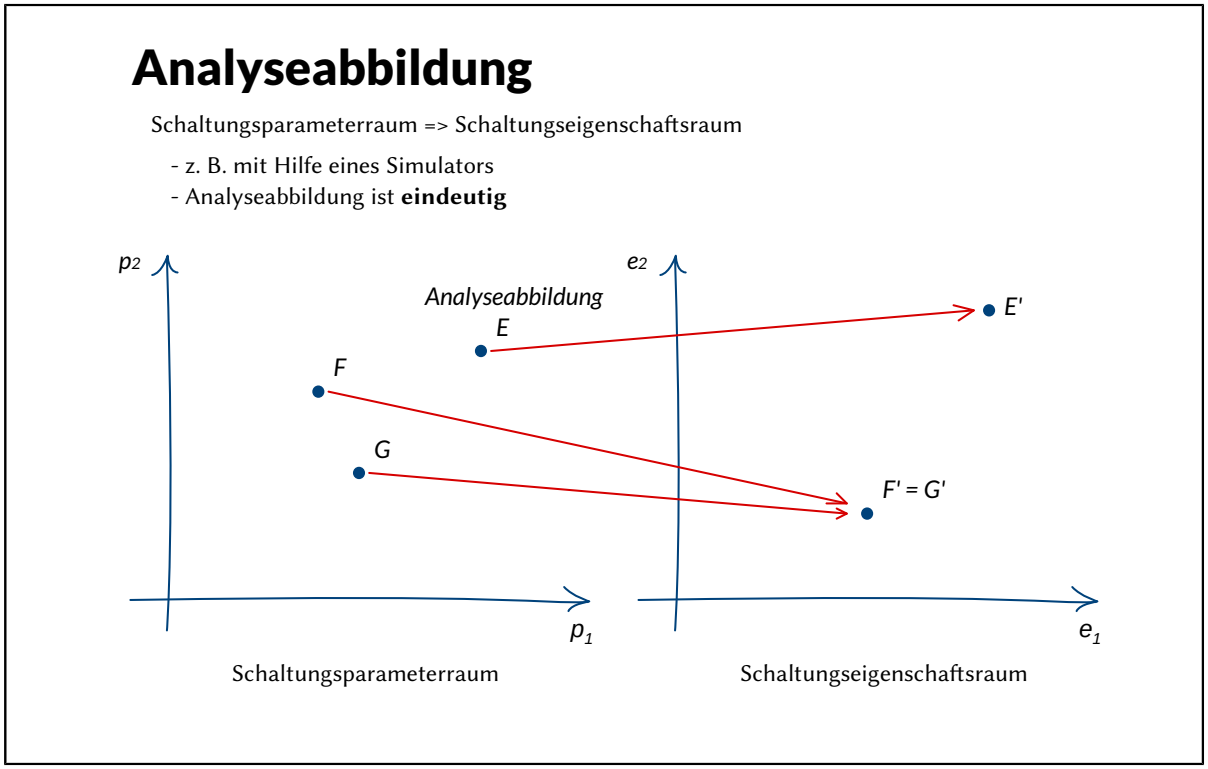
Analogsynthese: ... Beispiel



Das Bild zeigt erneut den aus zwei Widerständen R1 und R2 bestehenden Spannungsteiler, sowie den Schaltungseigenschaftsraum für die Eigenschaften Teilungsverhältnis a und Ausgangswiderstand Raus. Zusätzlich ist das durch die Spezifikation definierte Akzeptanzgebiet dargestellt. Die Dimensionierung, die Punkt C ergibt, erfüllt die Spezifikation, diejenige, die Punkt D ergibt, verletzt sie.

Die dargestellte achsenparallele rechteckige Form ist typisch für Akzeptanzgebiete im Schaltungseigenschaftsraum, da in der Spezifikation feste Ober- und Untergrenzen für Schaltungseigenschaftsgrößen gegeben sind. Falls für eine Größe nur eine Ober- oder eine Untergrenze angegeben ist, ist das Akzeptanzgebiet nach unten bzw. oben offen.

Analogsynthese: Analyseabbildung

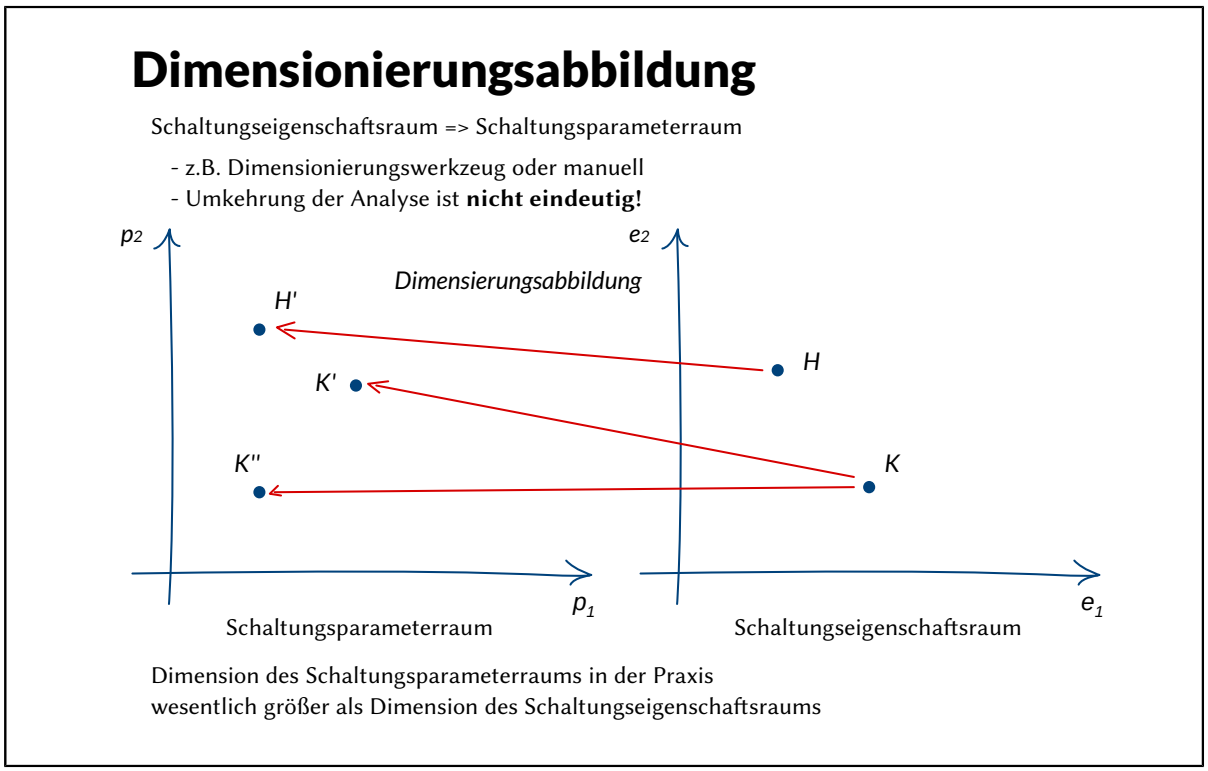


Die Zuordnung der Punkte in Schaltungsparameter- und Schaltungseigenschaftsraum zueinander kann in zwei Richtungen geschehen: Durch Abbildung vom Schaltungsparameter- in den Schaltungseigenschaftsraum oder umgekehrt.

Bei der Analyseabbildung handelt es sich um einen Analysevorgang Sie ist in der Regel eindeutig. Sie wird in der Praxis meist durch einen Schaltungssimulator durchgeführt.

Das Bild zeigt den Schaltungsparameter und -eigenschaftsraum für das Spannungsteilerbeispiel sowie exemplarisch die Abbildung von drei Punkten E, F und G aus dem Schaltungsparameter- in den Schaltungseigenschaftsraum mit Hilfe einer Analyseabbildung. Die Bildpunkte F' und G' fallen im Schaltungseigenschaftsraum zusammen, d. h. die Dimensionierungen der Punkte F und G weisen die gleichen Eigenschaften auf.

Analogsynthese: Dimensionierungsabbildung



Im Fall der Dimensionierungsabbildung handelt es sich um einen Synthesevorgang. Sie kann als Umkehrung der Analyseabbildung definiert werden, ist dadurch aber nicht eindeutig definiert, da mehrere Dimensionierungen die gleichen Eigenschaften haben können.

Die Dimension des Schaltungsparameterraums ist bei praktischen Problemen wesentlich größer als die des Schaltungseigenschaftsraums.

Im Bild ist ein Beispiel für eine Dimensionierungsabbildung dargestellt: Die Punkte H und K werden aus dem Schaltungseigenschafts- in den Schaltungsparameterraum abgebildet. Für den Punkt K ergeben sich aufgrund der Mehrdeutigkeit der Dimensionierungsabbildung zwei Bildpunkte K' und K''.

Analogsynthese: Abbildung des Akzeptanzgebietes

Abbildung des Akzeptanzgebietes

Bei Dimensionierung gesucht:

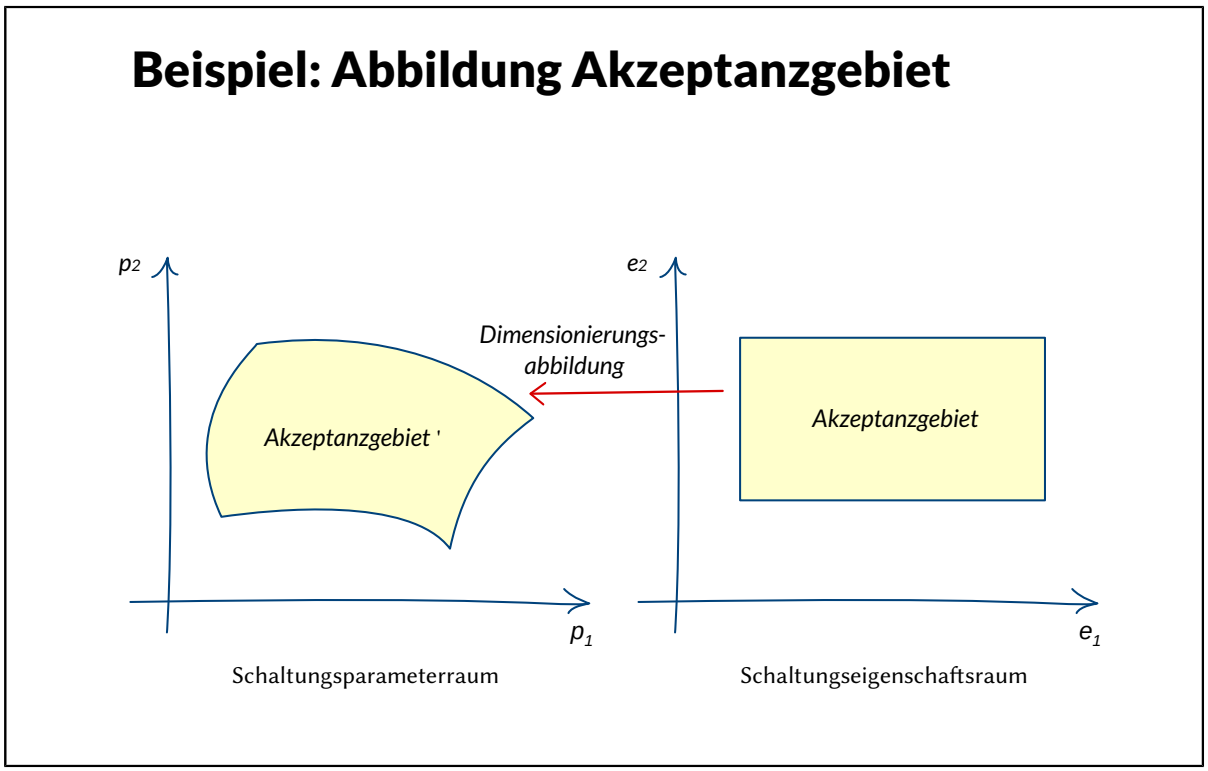
Punkte im Schaltungsparameterraum mit Bildpunkten im Schaltungseigenschaftsraum innerhalb Akzeptanzgebiet.

Ansatz:

Abbildung des gesamten Akzeptanzgebietes mit Dimensionierungsabbildung

Aufgabe der Dimensionierung ist es, Punkte im Schaltungsparameterraum zu bestimmen, die die Spezifikation erfüllen. Ihre Bildpunkte im Schaltungseigenschaftsraum müssen daher innerhalb des Akzeptanzgebietes liegen. Zur Bestimmung aller Punkte mit dieser Eigenschaft ist das Akzeptanzgebiet aus dem Schaltungseigenschaftsraum mit Hilfe der Dimensionierungsabbildung in den Schaltungsparameterraum abzubilden.

Analogsynthese: ... Beispiel



Das Bild zeigt Schaltungsparameter und -eigenschaftsraum für das Spannungsteilerbeispiel. Das im Schaltungseigenschaftsraum gegebene Akzeptanzgebiet wird mit der Dimensionierungsabbildung in den Schaltungsparameterraum abgebildet. Die Form des Bildes "Akzeptanzgebiet'" ist i. A. nicht mehr rechteckig.

Analogsynthese: Dimensionierungsproblem

Dimensionierungsproblem

Akzeptanzgebiet im Schaltungsparameterraum enthält mehr als einen Punkt.

Verwendung eines Optimierungszieles zur Auswahl des besten Punktes

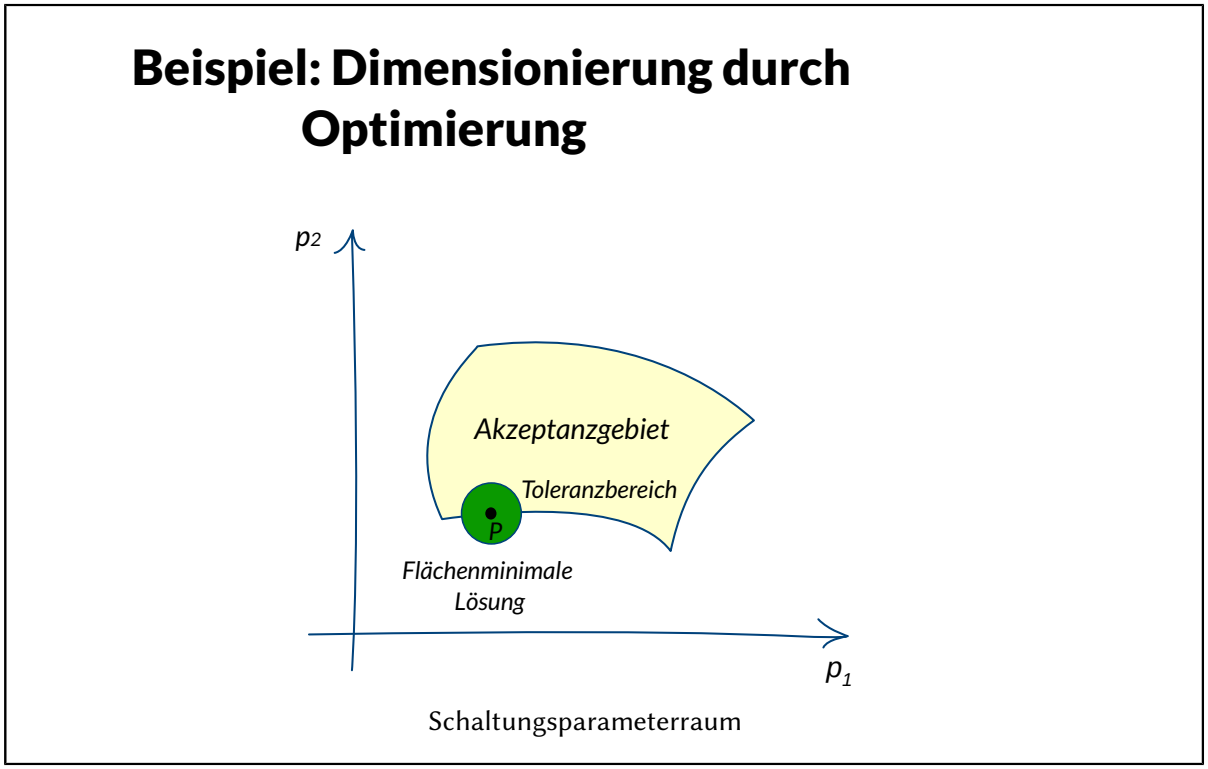
- Minimierung von Chipfläche, Stromverbrauch
- Maximierung von Bandbreite, Verstärkung o.ä.

→ **Dimensionierungsproblem ist ein nichtlineares Optimierungsproblem (NLP) mit der Randbedingung:**

-Die Lösung muss innerhalb des Akzeptanzgebietes liegen

Das Akzeptanzgebiet und sein Bild im Schaltungsparameterraum enthalten eine unendliche Anzahl von Punkten, die die Spezifikation erfüllen. Für den Entwurf wird jedoch nur genau ein Satz von Schaltungsparametern für die Topologie, also ein Punkt im Schaltungsparameterraum, benötigt. Es muss daher ein Punkt im Akzeptanzgebiet ausgewählt werden. Dieses kann nach verschiedenen Kriterien geschehen, die formal als Optimierungsziele ausgedrückt werden. Beispielsweise kann unter allen Punkten derjenige gewählt werden, der eine minimale Chipfläche oder einen minimalen Stromverbrauch hat. Andere gebräuchliche Ziele sind Maximierung von Bandbreite oder Herstellungsausbeute. Ein solches Optimierungsziel wird zusätzlich in die Spezifikation aufgenommen. Damit ist das Dimensionierungsproblem ein nichtlineares Optimierungsproblem (NLP) mit der Randbedingung, dass die Lösung innerhalb des Akzeptanzgebietes liegen muss.

Analogsynthese: Beispiel Dimensionierung durch Optimierung



Bei der Abbildung des Akzeptanzgebietes im Schaltungsparameterraum wurde durch Minimierung der Chipfläche der Punkt P bestimmt. Dieser liegt auf dem Rand des Akzeptanzgebietes. Durch Schwankungen der Widerstandswerte infolge von Toleranzen kann es leicht passieren, dass der Punkt das Akzeptanzgebiet verlässt. In diesem Fall ist die Spezifikation verletzt, und eine derartige Schaltung würde beim Test im Anschluss an die Herstellung aussortiert. Dadurch kommt es zu Ausschuss und entsprechend höheren Kosten.

Analogsynthese: Ausbeuteoptimierung

Ausbeuteoptimierung

Prozess unterliegt Toleranzen:

z.B. Widerstand +/- 20%

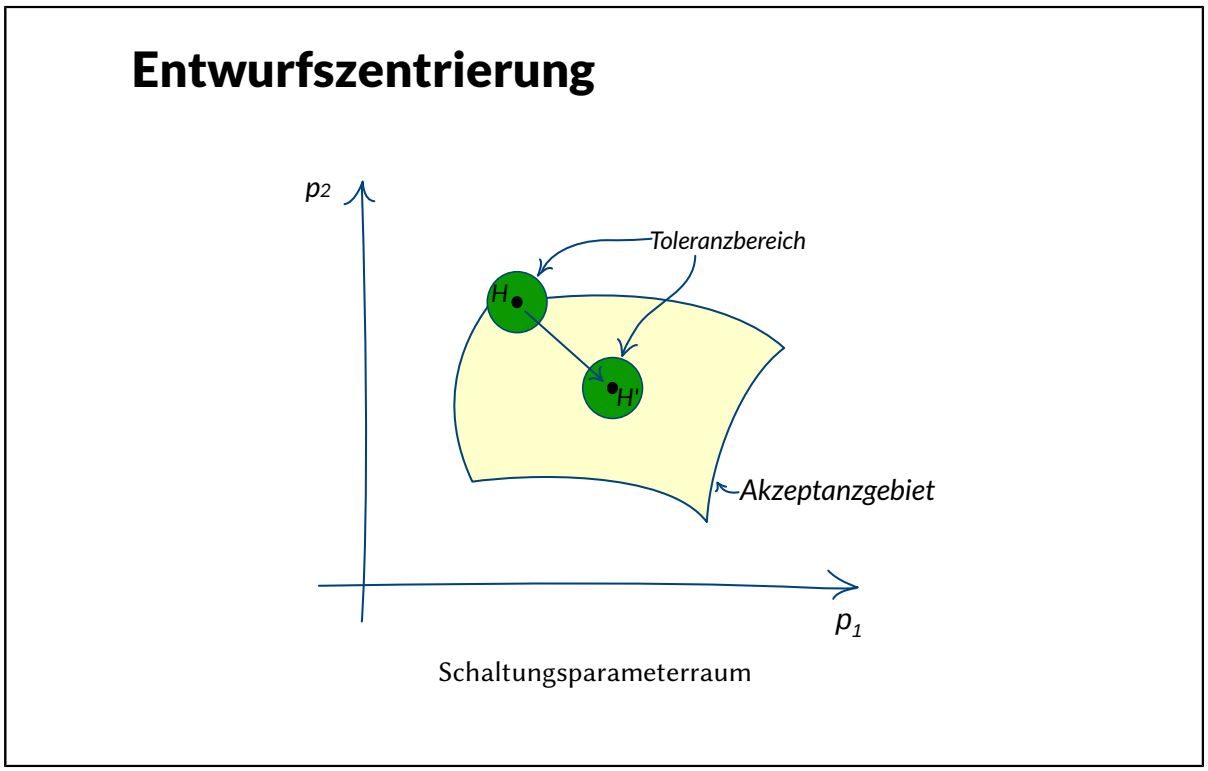
Eigenschaftsschwankungen können Spezifikation verletzen

Ausbeute als Zielfunktion

- Möglichst viele hergestellte Schaltungen sollen Spezifikation erfüllen
- Ausbeute = (Anzahl fehlerfreier Chips) / (Gesamtzahl hergestellter Chips)
- Produktionsprozess muss analysiert (=simuliert) werden (sehr aufwändig)

Bei der Herstellung integrierter Schaltungen kommt es infolge von Toleranzen zu statistisch verteilten Abweichungen von den vorgegeben Schaltungsparametern. Beispielsweise unterliegen Widerstände in integrierten Schaltungen Toleranzen von ca. 20%. Durch diese Schwankungen kommt es zu Spezifikationsverletzungen bei einem Teil der hergestellten integrierten Schaltungen, d. h. die Ausbeute liegt unter 100%. Sie ist allgemein als Quotient aus der Anzahl der Schaltungen, die die Spezifikation einhalten, zu der Gesamtzahl der Schaltungen definiert. Um eine maximale Ausbeute zu erzielen, muss diese als Zielfunktion der Optimierung gewählt werden. Dadurch wird sichergestellt, dass so viele herstellte Schaltungen wie möglich die Spezifikation einhalten. Im Rahmen der Optimierung muss die Ausbeute durch Analyse bestimmt werden, indem die statistischen Vorgänge der Fertigung simuliert werden. Derartige Simulationen sind jedoch sehr aufwändig.

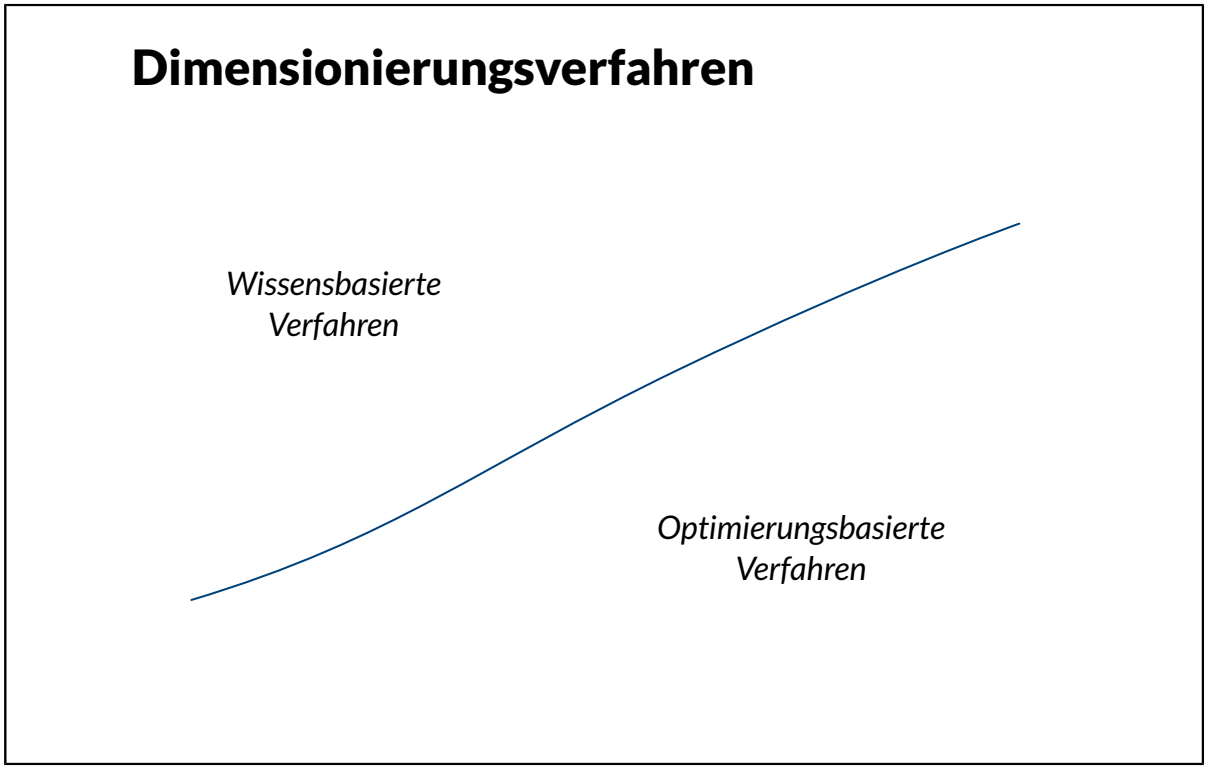
Analogsynthese: Entwurfszentrierung



Ein Ansatz, die Ausbeute ohne Simulation der statistischen Vorgänge zu erhöhen, besteht in der so genannten Entwurfszentrierung. Dabei wird ein Punkt im Zentrum des Bildes des Akzeptanzgebietes im Schaltungsparameterraum gewählt, der möglichst weit von der Begrenzung des Akzeptanzgebietes entfernt ist. Durch diesen "Sicherheitsabstand" wird erreicht, dass kleine Schwankungen der Schaltungsparameter nicht zum Verlassen des Akzeptanzgebietes führen, so dass die Ausbeute verbessert wird.

Obwohl dieser Ansatz ohne eine aufwändige Ausbeuteanalyse auskommt, verbleibt immer noch die Notwendigkeit, den Rand des Akzeptanzgebietes zu bestimmen oder abzuschätzen. Dieses erfordert immer noch einen hohen Rechenaufwand.

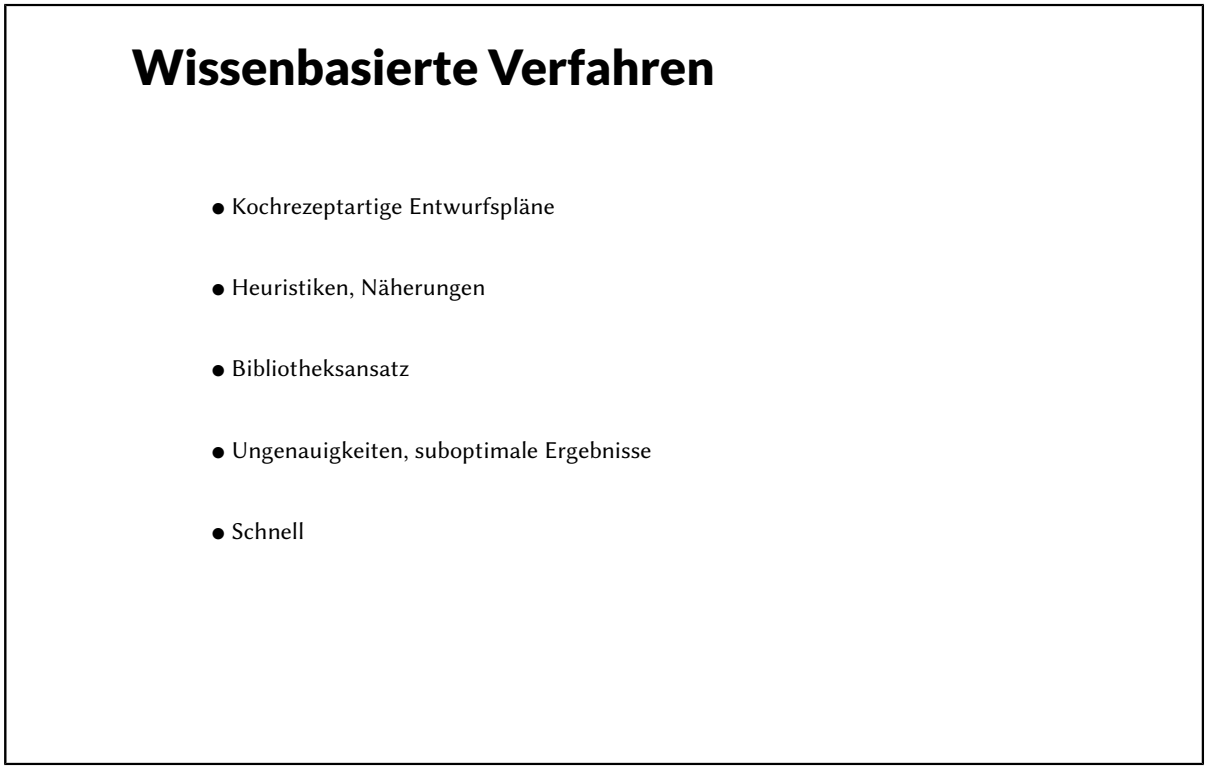
Analogsynthese: Dimensionierungsverfahren



Praktische Dimensionierungsverfahren lassen sich in zwei Klassen einteilen:

- wissensbasierte und
- optimierungsbasierte Verfahren.

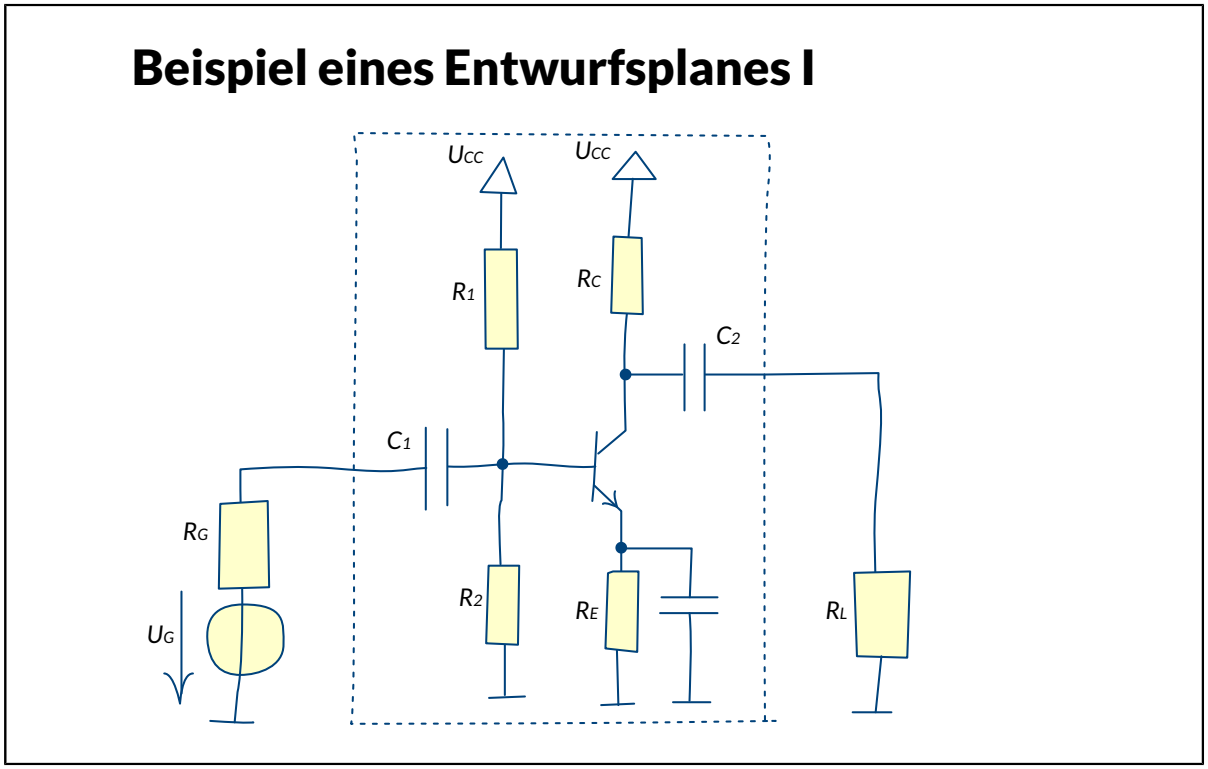
Analogsynthese: Wissensbasierte Verfahren



Bei der Klasse der wissensbasierten Verfahren werden kochrezeptartige Entwurfspläne verwendet. Diese sind dem Handentwurf nachempfunden und basieren auf dem Wissen von Schaltungsentwicklern. Dieses Wissen muss für jede Topologie erfasst und in eine maschinenlesbare Sprache umgewandelt werden. Es finden in der Regel Heuristiken und Näherungen Anwendung. Dieser Ansatz führt zu einem Bibliothekskonzept, wobei die Bibliothek aus Topologien und den zugeordneten Entwurfsplänen besteht. Eine solche Bibliothek ist aufwändig zu erstellen und hat infolge des rasanten technischen Fortschritts nur eine kurze Lebensdauer.

Durch die Verwendung von Näherungen und Heuristiken kommt es bei Verwendung von Entwurfsplänen zu suboptimalen Ergebnissen und Abweichungen vom gewünschten Verhalten, die u. U. bis zur Spezifikationsverletzung reichen können. Vorteilhaft ist der sehr geringe Rechenaufwand. Von Spezialgebieten abgesehen, haben wissensbasierte Ansätze keine praktische Bedeutung mehr.

Analogsynthese: Beispiel eines Entwurfsplanes

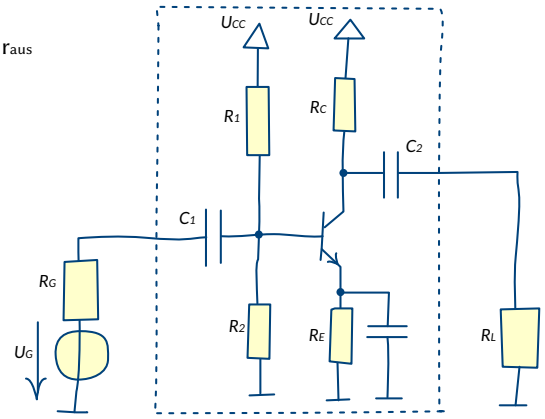


Das Bild zeigt einen einstufigen Verstärker mit Bipolartransistor (Emitterverstärker). Mit Hilfe eines Entwurfsplanes sollen die Widerstände R1, R2, RC und RE für vorgegebene AC-Verstärkung v und Ausgangswiderstand Raus dimensioniert werden.

Analogsynthese: Beispiel eines Entwurfsplanes II

Beispiel eines Entwurfsplanes II

- C_1, C_2, C_3 sind AC-Kurschlüsse
- Näherung: $R_C = r_{aus}$
- Heuristik: $R_E = R_C / 10$
- Näherungen: $r_{ein} >> R_G$, $r_{aus} << R_L$
  mit $g_m = di_C/du_{BE} = I_C/U_T$ und $V = g_m * r_{aus}$
  $\Rightarrow I_C = (V * U_T) / r_{aus}$
- Näherung: $U_{BE} = 0{,}7$ V
- Heuristik: $U_{CC} / (R_1 + R_2) = I_C / 10$
- Näherung: $I(R_2) = I(R_1)$
  $\Rightarrow R_2 = 10 (R_E + U_{BE} / I_C)$,
  $R_1 = 10 U_{CC} / I_C - R_2$



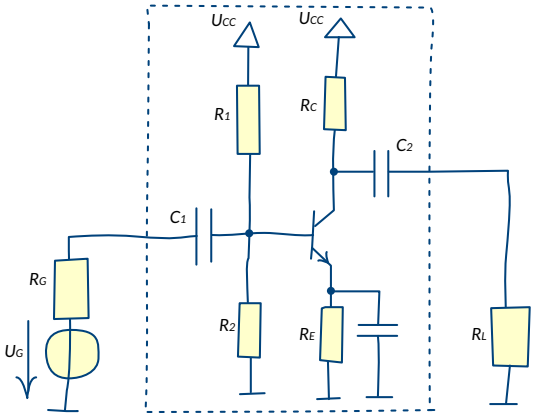
Bei der Aufstellung des Entwurfsplans wird angenommen, dass die Kapazitäten C1, C2 und C3 sich im Wechselstrombetrieb wie Kurzschlüsse verhalten. Für kleine RC kann der Ausgangswiderstand des Transistors gegenüber RC vernachlässigt werden, so dass RC = Raus. Zur Stabilisierung des Arbeitspunktes soll die Driftverstärkung nicht größer als 10 sein. Gleichzeitig soll der Spannungsabfall an RE nicht größer sein als nötig, um einen maximalen Ausgangsspannungshub zu ermöglichen. Daher wird RE = RC / 10 gewählt. Mit den Näherungen Rein # RG, Raus # RL ist V = (IC / UT) * Raus, so dass IC = (v*UT) / Raus bestimmt werden kann. Damit der Basisspannungsteiler möglichst unabhängig von IB ist, wird der Querstrom zu 10*IB gewählt. Bei einem angenommenen BETA = 100 ergibt dies UCC / (R1 + R2) = IC / 10. Dann kann I(R2) = I(R1) angenommen werden. Ferner wird UBE = 0.7 V angenähert. Eine Masche über R2, die Basis-Emitter-Strecker und RE ergibt dann: R2 = 10 (RE + UBE / IC) und aus UCC / (R1 + R2) = IC / 10 folgt: R1 = 10 UCC / IC - R2.

Analogsynthese: Anwendung des Entwurfsplanes

Anwendung des Entwurfsplanes

- V = 40 dB
- r_{aus} = 1 kΩ
- U_{CC} = 10 V

- R_C = 1 kΩ
- R_E = 100 Ω
- R_2 = 3,8 kΩ
- R_1 = 36,2 kΩ



- Spice-Simulation mit Q2N2222, R_G = 0 Ω, R_L = unendlich:

  V = 38,8 dB, r_{aus} = 970 Ω

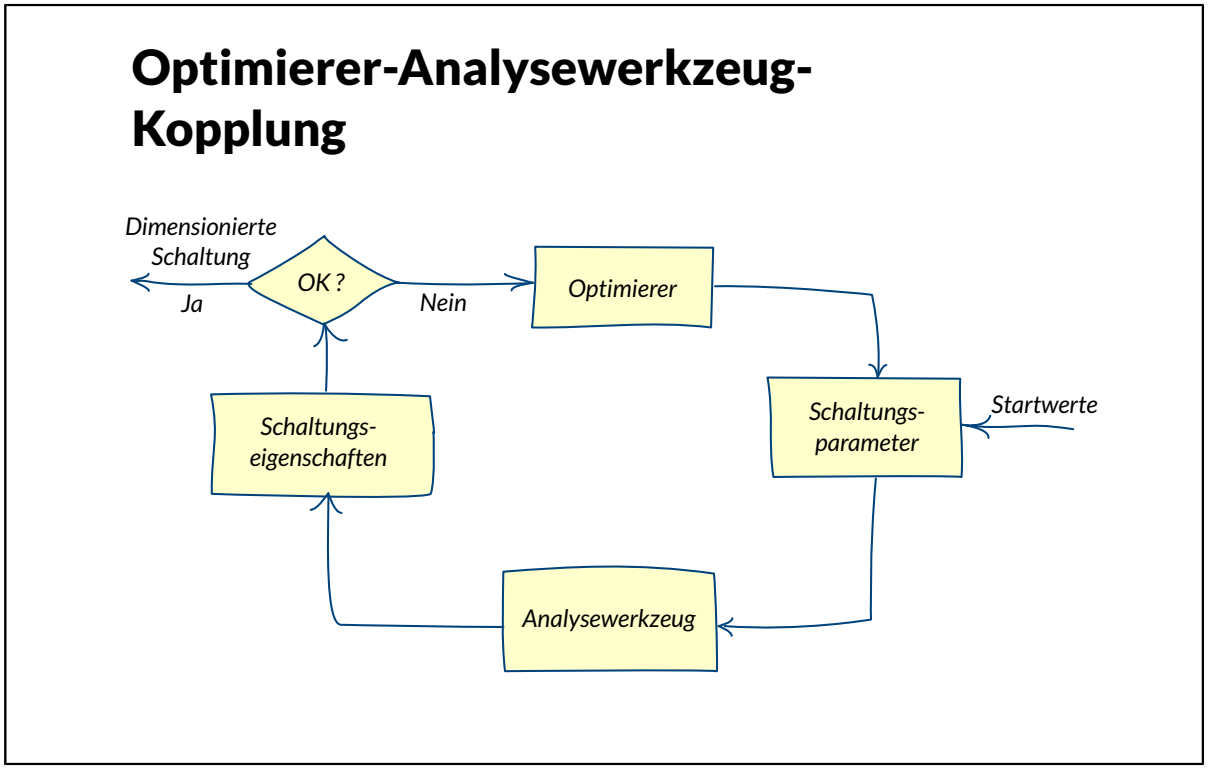
Die Anwendung des Entwurfsplanes auf eine Dimensionierungsaufgabe mit V = 40 dB, Raus = 1 kOhm und UCC = 10 V ergibt: RC = 1 kOhm, RE = 100 Ohm, R2 = 3,8 kOhm und R1 = 36,2 kOhm. Zur Kontrolle des Ergebnisses wird eine Eigenschaftsanalyse mit dem Simulator Spice durchgeführt. Dieser liefert bei einem Transistor des Typs 2N2222, RG = 0 und Leerlauf am Ausgang des Verstärkers V = 38,8 dB und Raus = 970 Ohm. Diese Werte zeigen, dass die mit dem Entwurfsplan erzielten Eigenschaften von den vorgegebenen Sollwerten leicht abweichen. Dies ist auf die Näherungen zurückzuführen.

Analogsynthese: Optimierungsbasierte Verfahren



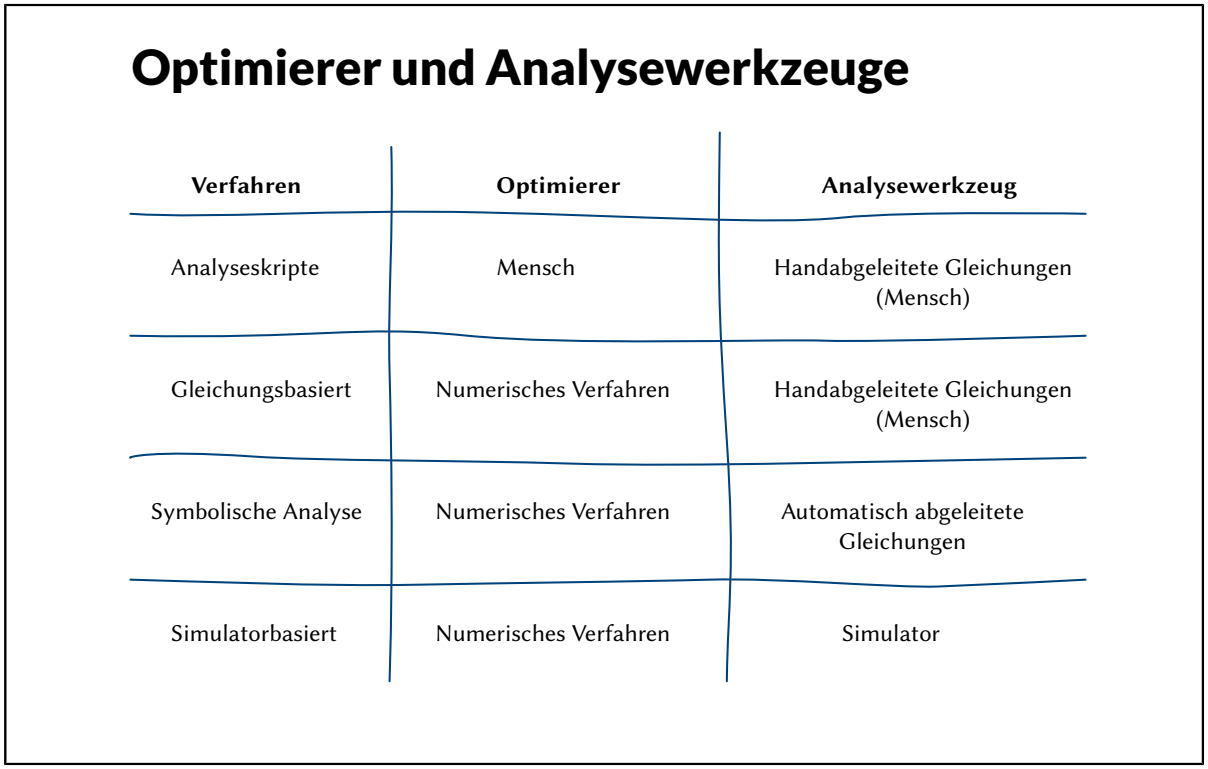
Optimierungsbasierte Verfahren lösen das nichtlineare Optimierungsproblem mit Randbedingungen, das sich aus der Dimensionierungsaufgabe ergibt, mit einem iterativen Optimierungsverfahren. Dabei sind grundsätzlich keine Näherungen notwendig: Durch die Allgemeinheit der Optimierer-Analysewerkzeug-Kopplung hängt die Genauigkeit nur vom Analysewerkzeug ab. Gegenüber wissensbasierten haben optimierungsbasierte Ansätze einen erheblich höheren Rechenaufwand. Das iterative Vorgehen kann zu Konvergenzproblemen führen.

Analogsynthese: Optimierer-Analysewerkzeug-Kopplung



Wie im Bild gezeigt, wird bei einem optimierungsbasierten Ansatz ein Optimierer mit einem beliebigen Analysewerkzeug gekoppelt: Der Optimierer übergibt dem Analysewerkzeug einen Satz Schaltungsparameter, das Analysewerkzeug bestimmt die dafür gültigen Schaltungseigenschaften und gibt diese an den Optimierer zurück. Auf der Grundlage der Eigenschaften verändert der Optimierer die Schaltungsparameter und übergibt sie erneut dem Analysewerkzeug. Diese Schleife wird so lange durchlaufen, bis die Spezifikation unter Berücksichtigung aller Toleranz grenzen und -bedingungen erfüllt ist.

Analogsynthese: Optimierer und Analysewerkzeuge



Der optimierungsbasierte Ansatz erlaubt es, verschiedene Komponenten als Optimierer und Analysewerkzeug zu koppeln. Eine chronologische Aufstellung der verschiedenen Verfahren in der Entwicklung des optimierungsbasierten Ansatzes ist in der Tabelle dargestellt. Heutiger Stand ist die Kopplung eines numerischen Optimierungswerkzeuges mit einem Schaltungssimulator. Dieses ermöglicht hohe Genauigkeit und die Ausnutzung aller Analysearten, die der Simulator bietet. Abhängig von dem verwendeten Optimierungsverfahren fällt jedoch eine große Zahl an Simulationen und damit ein hoher Rechenaufwand an.

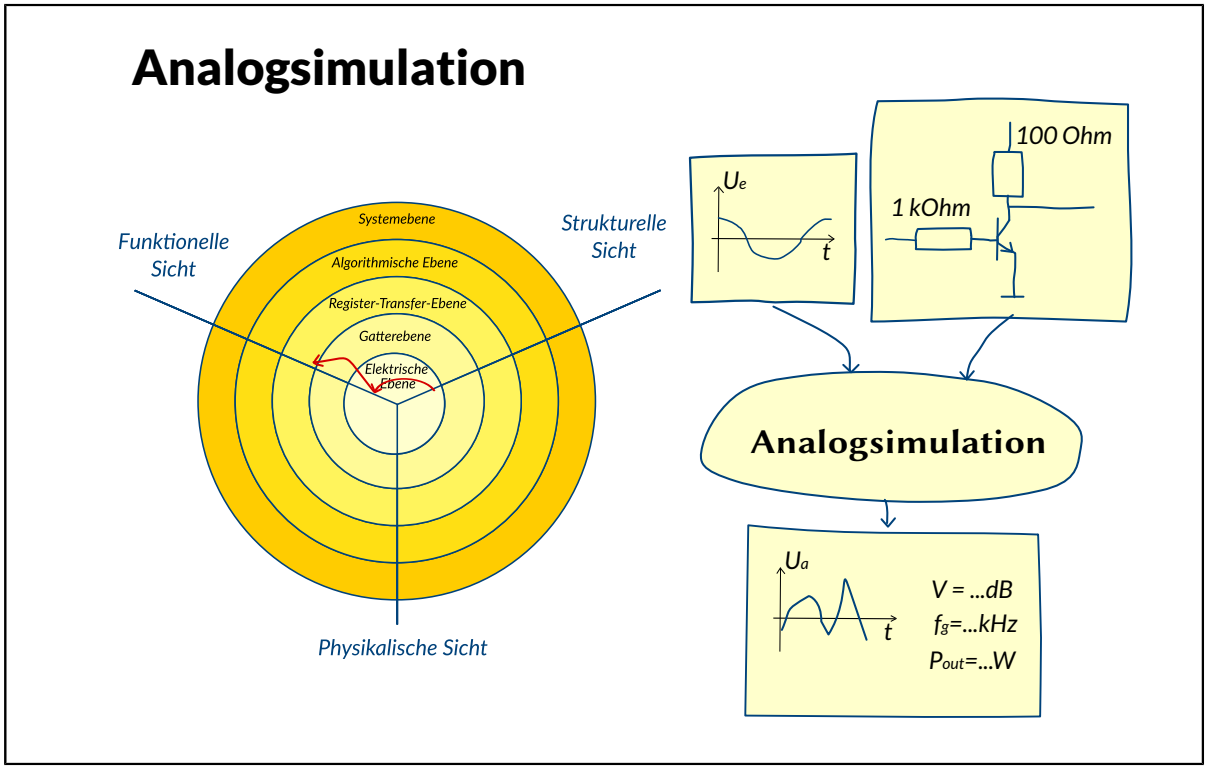
Electronic Design Automation (EDA)

Analoge Simulation

Analoge Simulation
Schaltungsbeschreibung
Analysearten
Transiente Analyse
Aufstellen der Schaltungsgleichungen
Lösen der Schaltungsgleichungen
Berechnung der Zeitfunktionen
Implizites Eulerverfahren
Trapezverfahren
Linearisierung
Newton-Raphson-Methode
...Eigenschaften
Lösen linearer Gleichungssysteme
DC-Analyse
AC-Analyse
Bode-Diagramm
Weitere Analyse-Arten

Date of generation: 8.7.2025

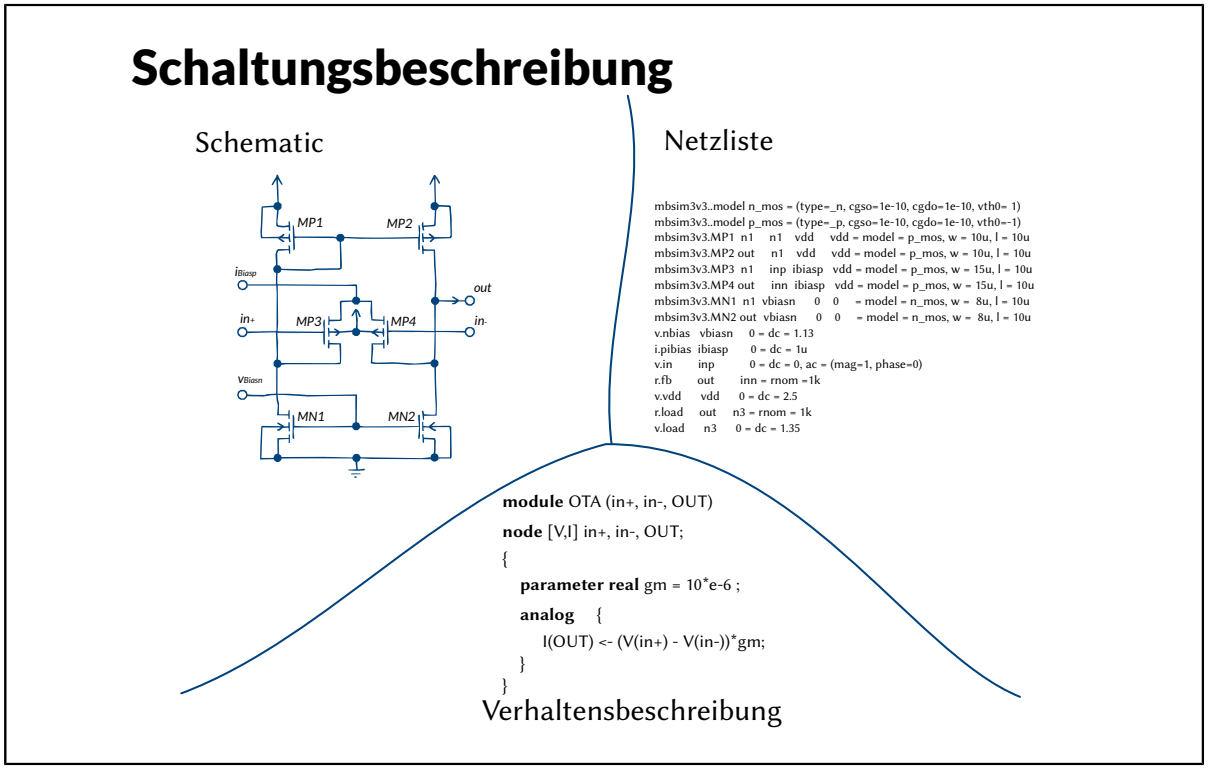
Analoge Simulation: Analoge Simulation



Beim Entwurf integrierter analoger Schaltungen hat sich die Simulation als Hauptwerkzeug für die Analyse etabliert. Sie basiert auf der numerischen Lösung von Gleichungssystemen.

Die analoge Simulation ist ein universales Werkzeug, das als Hilfsmittel auch in vielen anderen Bereichen eingesetzt wird wie z.B. der Synthese, oder bei Ausbeuteuntersuchungen.

Analoge Simulation: Schaltungsbeschreibung

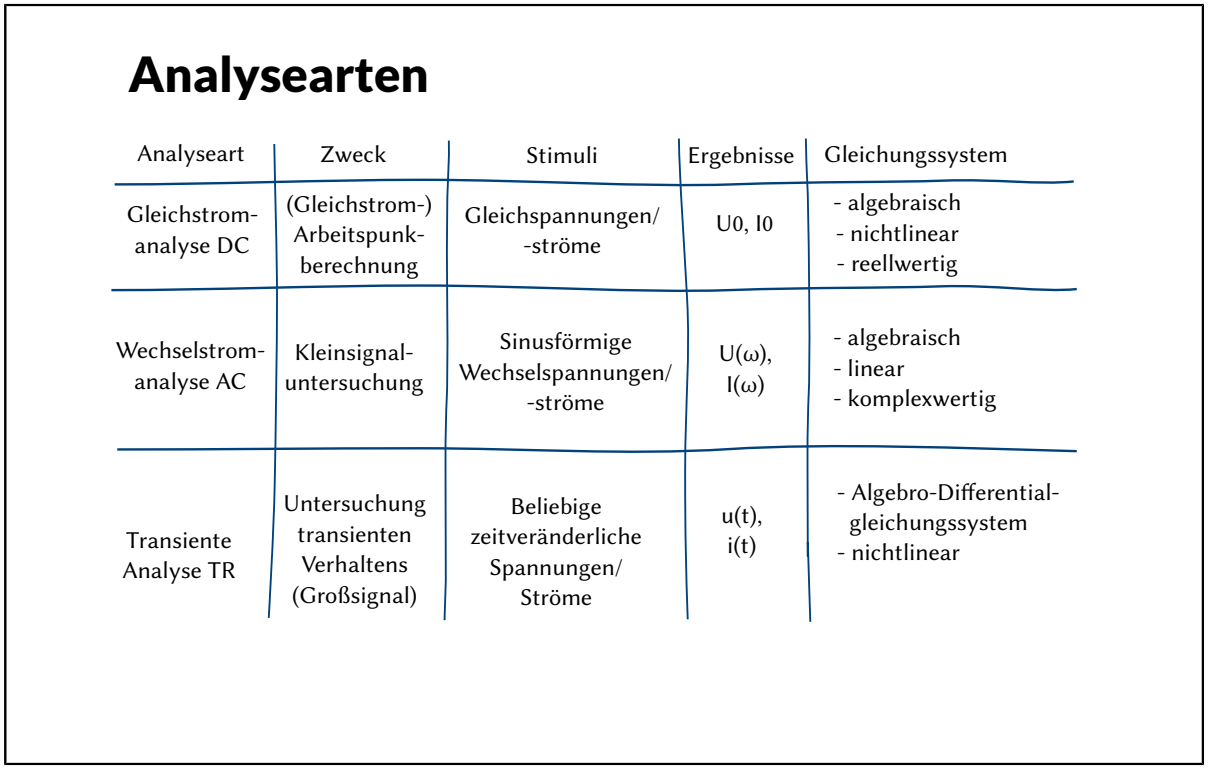


Bei Analogschaltungen erfolgt die Eingabe häufig als Schaltbild (Schematic) oder durch das Schreiben einer Netzliste von Hand. Im Bild ist das Schematic eines OTAs (Operational Transconductance Amplifier) dargestellt. Er besteht aus 6 Transistoren: MP3 und MP4 für die Eingangsdifferenzstufe, MP1 und MP2 fuer den Stromspiegel und MN1 und MN2 fuer das Biasnetzwerk.

Die Netzliste der Schaltung kann entweder durch die automatische Umwandlung aus einem Schematic durch einen so genannten Netzlister erzeugt werden, oder aber sie ist handgeschrieben. Für einen Simulator stellt in der Regel die strukturelle Beschreibung als Netzliste die Eingabe dar. Netzlisten können neben der eigentlichen Beschreibung der Schaltung durch Bauelemente und deren Verbindungen außerdem Anweisungen zur Simulation enthalten. Außerdem erfordert die Simulation die Definition von Versorgungsspannungen und Stimuli an den Eingängen.

Eine weitere Form der Beschreibung analoger Schaltungen ist durch die Verwendung von Verhaltensmodellierungssprachen wie VHDL-AMS oder VERILOG-A gegeben. Anstatt die Schaltung durch ihre Struktur darzustellen, wird die Funktion beschrieben. Das Beispiel zeigt eine Beschreibung in VERILOG-A. Der Ausgangsstrom berechnet sich aus der Differenz der Eingangsspannungen multipliziert mit dem idealen Leitwert des OTA.

Analoge Simulation: Analysearten



Analysearten

Analyseart	Zweck	Stimuli	Ergebnisse	Gleichungssystem
Gleichstrom-analyse DC	(Gleichstrom-) Arbeitspunk-berechnung	Gleichspannungen/ -ströme	U0, I0	- algebraisch - nichtlinear - reellwertig
Wechselstrom-analyse AC	Kleinsignal-untersuchung	Sinusförmige Wechselspannungen/ -ströme	U(ω), I(ω)	- algebraisch - linear - komplexwertig
Transiente Analyse TR	Untersuchung transienten Verhaltens (Großsignal)	Beliebige zeitveränderliche Spannungen/ Ströme	u(t), i(t)	- Algebro-Differential-gleichungssystem - nichtlinear

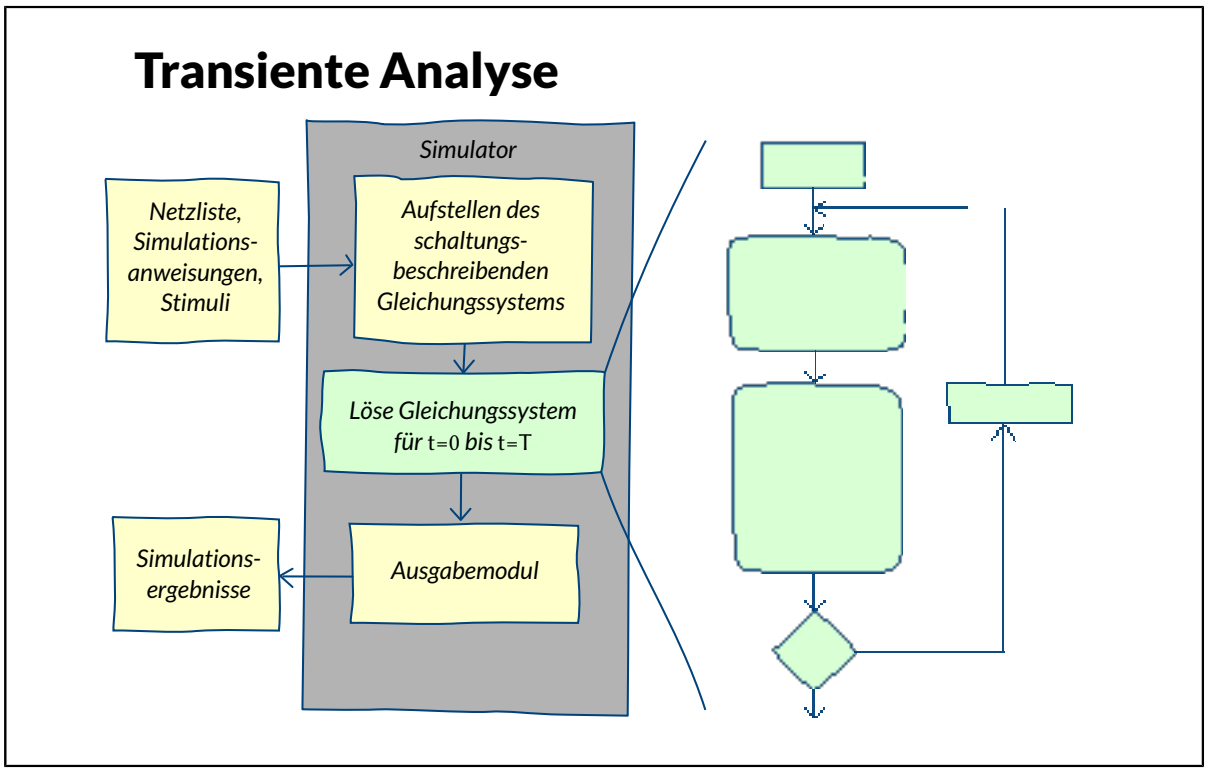
Ein Analogsimulator kann in verschiedenen Betriebsarten genutzt werden. Die grundlegenden Betriebsarten sind die Gleichstromanalyse (DC), die Wechselstromanalyse (AC) und die transiente Analyse (TR).

Die Gleichstromanalyse dient der Ermittlung eines Gleichstromarbeitspunktes. Die Schaltung wird dabei mit allen Nichtlinearitäten, aber ohne Zeitabhängigkeiten betrachtet, d.h. z.B. dass Induktivitäten durch einn Kurzschluss und Kapazitäten durch einen unendlich großen Widerstand (Unterbrechung) modelliert werden. Es ist ein algebraisches, nichtlineares, reellwertiges Gleichungssystem zu lösen.

Die Wechselstromanalyse dient zur Berechnung des Kleinsignalverhaltens der Schaltung. Untersucht wird das Verhalten bei kleinen Auslenkungen um den Arbeitspunkt herum mit sinusförmigen Wechselgrößen als Stimuli. Alle Nichtlinearitäten werden durch lineare Modelle ("Kleinsignalmodelle") dargestellt. Es muss ein algebraisches, lineares, komplexwertiges Gleichungssystem gelöst werden.

Die transiente Analyse stellt den allgemeinsten Fall dar. Als Stimuli können beliebige zeitabhängige Größen auftreten (Großsignalbetrachtung), alle Nichtlinearitäten bleiben erhalten. Es muss deshalb ein nichtlineares, reellwertiges Algebro-Differentialgleichungssystem gelöst werden. Die (gewöhnlichen) Differentialgleichungen sind von 1. Ordnung und haben konstante Koeffizienten.

Analoge Simulation: Transiente Analyse



Die transiente Analyse bildet den zeitlichen Verlauf der Variablen der vorliegenden Schaltung nach. Sie ist damit die anschaulichste Analyseart, da der Mensch das Denken in zeitlichen Abläufen gewohnt ist und man sich somit die Vorgänge der Schaltung am besten vor Augen führen kann. Viele Kenngrößen lassen sich aus der transienten Simulation ableiten, wie z.B. die Slewrate (Steigung der Ausgangskennlinie bei einem Sprung als Eingangserregung), die Settlingtime (Zeit, die das Ausgangssignal bei einem Sprung als Eingangserregung benötigt, bis es im eingeschwungenen Zustand ist), etc. Gleichzeitig ist es (neben einigen Analysen, die mehrfache Analyseaufrufe beinhalten) die mathematisch komplexeste Analyseart.

Neben der Netzliste benötigt der Simulator Angaben zur Simulation. Bei der transienten Analyse bestehen diese mindestens aus den Stimuli, der Simulationszeit T und der Schrittweite delta t. Moderne Simulatoren haben eine integrierte Schrittweitensteuerung, so dass delta t lediglich als Richtwert dient. Weitere Simulationsangaben können u.a. den Simulationsalgorithmus oder die nach der Simulation darzustellenden Signalverläufe betreffen.

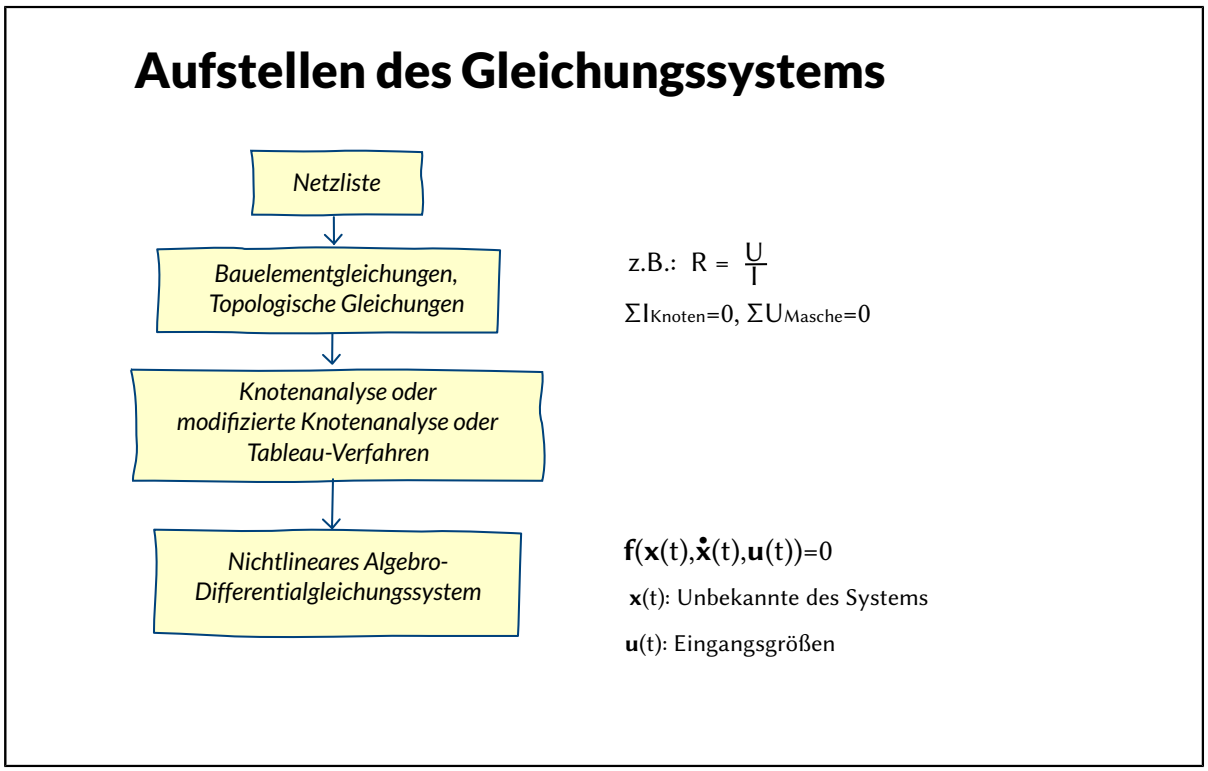
Die Netzliste wird vom Simulator eingelesen und in ein Gleichungssystem umgewandelt. Hierzu benötigt jeder Simulator eine Bauelementebibliothek, in der Modelle der Grundelemente der Schaltung abgelegt sind. Grundelemente sind Transistoren (Bipolar und MOS), Dioden, Widerstände, Kondensatoren, Spulen und elektrische Quellen. Die Elemente der Netzliste werden durch die repräsentierenden Gleichungen der Bauelementebibliothek (Bauelementmodelle) ersetzt und zu einem Gleichungssystem zusammengestellt.

Nun muss das Gleichungssystem in jedem Zeitschritt delta t, beginnend bei t=0, gelöst werden. H at die Simulationszeit den Punkt T erreicht, ist die Simulation beendet und die darzustellenden Signalverläufe werden in einer Datei abgelegt. Alternativ können sie auch sofort in einem geeigneten Tool (Waveform-Viewer) dargestellt werden.

Die eigentliche Lösung des nichtlinearen Algebro-Differentialgleichungssystems besteht aus mehreren Schritten. Zunächst müssen die zeitabhängigen Funktionen berechnet werden. Die sich ergebenden Differenzengleichungssysteme müssen mit iterativen Methoden wie der Newton-Raphson-Methode gelöst werden. Dies verdeutlicht, dass zur Berechnung des zeitabhängigen Großsignalverhaltens zwei

geschachtelte Schleifen notwendig sind: die äußere über die Zeit, und die innere zur Bestimmung des nichtlinearen Verhaltens in jedem Zeitschritt.

Analoge Simulation: Aufstellen der Schaltungsgleichungen

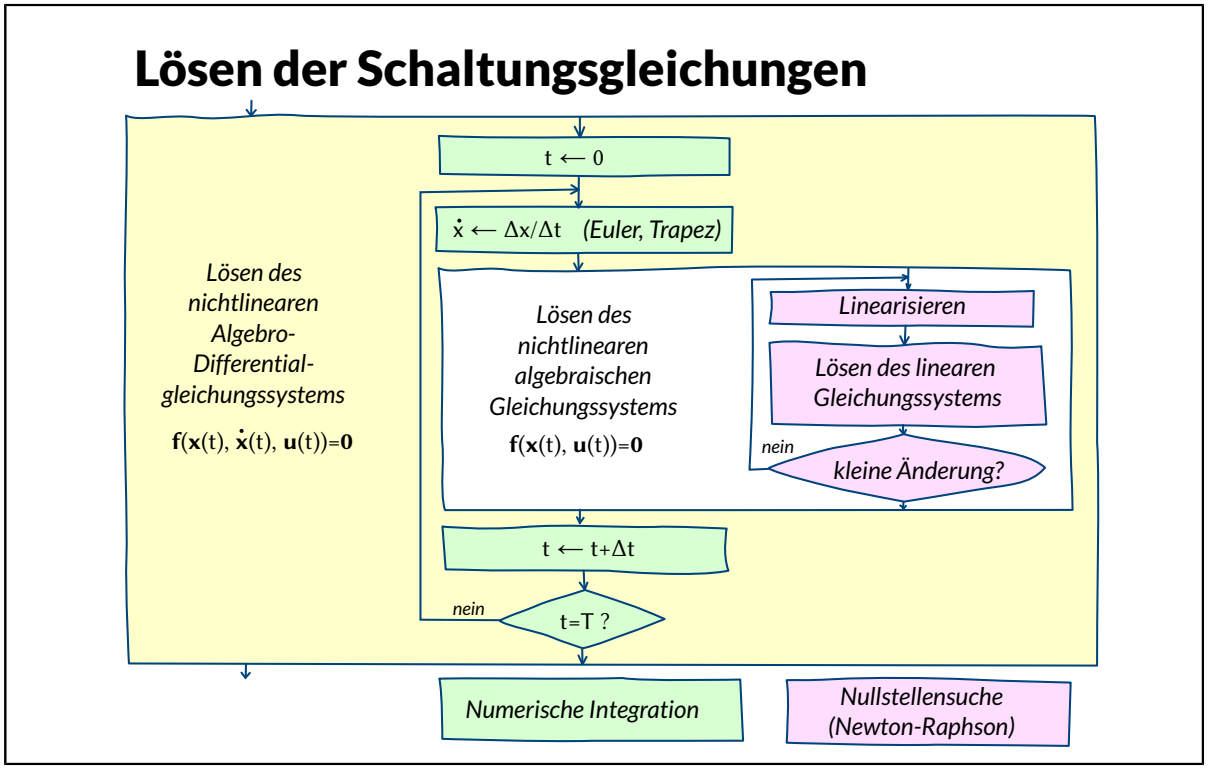


Die Überführung der Netzliste in ein Gleichungssystem ist notwendig, da die Netzliste nur die Platzhalter für die mathematische Beschreibung der Bauelemente durch Gleichungen enthält. Zum Aufstellen des Gleichungssystems existieren verschiedene Verfahren. Die bekanntesten sind die Knotenanalyse, die modifizierte Knotenanalyse und das Tableauverfahren.

Das Tableauverfahren verwendet die Kirchhoffschen Knoten- und Maschenregeln. Hierdurch entsteht ein relativ großes, schwach besetztes Gleichungssystem. Der Lösungsvektor enthält alle Zweigströme und Knotenpotentiale.

Die Knotenanalyse verwendet lediglich die Kirchhoffsche Knotenregel. Hierbei gelten die Einschränkungen, dass die Bauelemente auf Leitwerte, der Vektor der Unbekannten auf die Knotenspannungen und die Eingangsgrößen auf Stromquellen beschränkt sind. Diesen Nachteil beseitigt die modifizierte Knotenanalyse, die die Verwendung beliebiger Bauelemente erlaubt.
Die zu lösende Systemmatrix wird dadurch zwar unsymmetrisch, jedoch bedeutet dies für heutige Lösungsalgorithmen kein Problem, so dass der Vorteil, beliebige Bauelemente verwenden zu können, deutlich überwiegt.

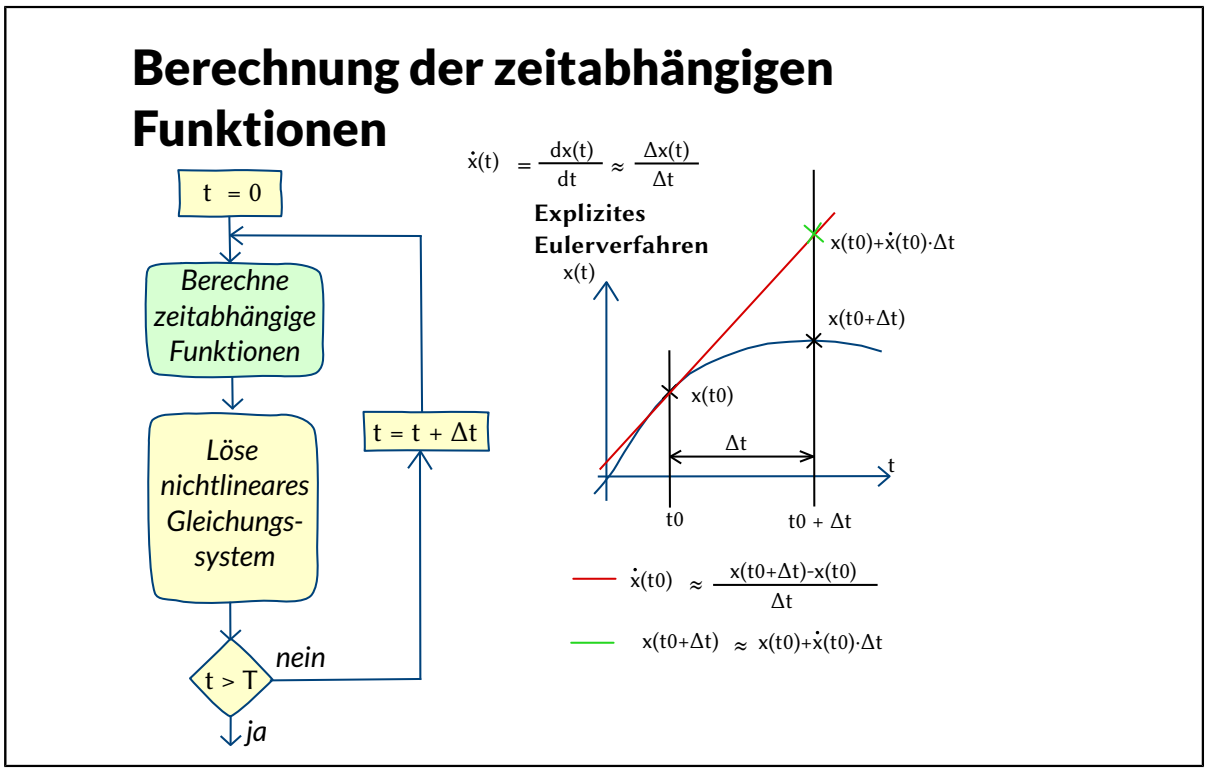
Analoge Simulation: Lösen der Schaltungsgleichungen



Die Schaltungsgleichungen bilden ein System von nichtlinearen impliziten Gleichungen, die teilweise Ableitungen nach der Zeit enthalten. Das Lösen dieses nichtlinearen Algebro-Differentialgleichungssystems geschieht durch numerische Integration, wobei Ableitungen durch Differenzenquotienten ersetzt werden. Die entstehenden nichtlinearen algebraischen Gleichungssysteme können näherungsweise mit dem Newton-Raphson-Verfahren gelöst werden. Dabei wird das Gleichungssystem wiederholt linearisiert und in jedem Schritt eine bisherige Näherungslösung verbessert. Bei kleiner Schrittweite der numerischen Integration genügt hier auch ein einzelner Linearisierungsschritt.

Allgemein wird bei diesem Vorgehen die Vereinfachung einer Problemstellung jeweils durch eine Iteration erkauft.

Analoge Simulation: Berechnung der Zeitfunktionen

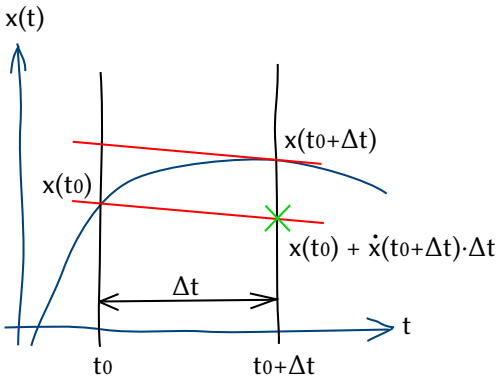


Der wesentliche Schritt ist die Integration im Zeitbereich. Sie dient der Berechnung der unbekannten Größen für den nächsten Zeitschritt. Die Integration kann entweder analytisch oder numerisch erfolgen. In der Praxis haben sich numerische Integrationsverfahren durchgesetzt, in denen die Ableitungen durch einen Differenzenquotienten nachgebildet werden. Die Verfahren werden in implizite und explizite Verfahren 1. oder höherer Ordnung eingeteilt. Sie unterscheiden sich in ihrer Genauigkeit, in der Stabilität und dem Rechenaufwand.

Das bekannteste explizite Verfahren ist das explizite Eulerverfahren. Es überführt den Differentialquotienten in einen Differenzenquotienten mit bereits bekannten Größen aus dem vorhergegangenen Zeitschritt. Diese Methode hat für die bei der Schaltungssimulation auftretenden Formen von Gleichungen eine ungenügende Stabilität und eine Neigung zur Fehlerfortpflanzung, weswegen sie nicht eingesetzt wird.

Analoge Simulation: Implizites Eulerverfahren

Implizites Eulerverfahren



$$\dot{x}(t_0+\Delta t) \approx \frac{x(t_0+\Delta t) - x(t_0)}{\Delta t}$$

$$x(t_0+\Delta t) \approx x(t_0) + \dot{x}(t_0+\Delta t)\cdot\Delta t$$

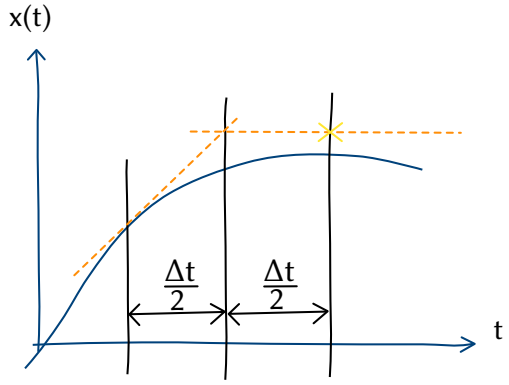
Im Gegensatz zum expliziten Eulerverfahren ist das implizite Eulerverfahren stabiler bezüglich des Integrationsfehlers!

Für das implizite Eulerverfahren lassen sich Stabilität und die Kompensation des Integrationsfehlers nachweisen. Es ersetzt ebenfalls den Differentialquotienten durch den Differenzenquotienten, benutzt jedoch zur Berechnung die Ableitung im nächsten Zeitpunkt $t_0+\Delta t$.

Analoge Simulation: Trapezverfahren

Trapezverfahren

$$x(t_0+\Delta t) \approx x(t_0) + \frac{\dot{x}(t_0+\Delta t)+\dot{x}(t_0)}{2}\Delta t$$



Das Trapezverfahren...

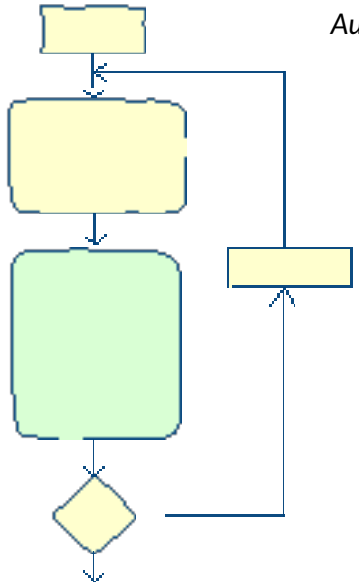
... ensteht aus Taylor-Reihenentwicklung mit Abbruch nach dem zweiten Glied.

... ist ein implizites Verfahren zweiter Ordnung

Eine implizite Methode 2. Ordnung ist das Trapezverfahren. Diese Berechnungsvorschrift lässt sich durch Taylorreihenentwicklung mit Abbruch nach dem 2. Glied herleiten. Es existieren weitere Methoden, wobei das bekannteste das Verfahren von Gear ist. Generell gilt, dass explizite Verfahren, auch höherer Ordnung, wegen der bereits für das explizite Eulerverfahren angeführten Gründe nicht für die Schaltungssimulation geeignet sind, so dass im allgemeinen implizite Methoden verwendet werden. Das implizite Eulerverfahren ist bereits eine geeignete und vielfach in Simulatoren eingesetzte Methode, ebenso das Trapezverfahren. Allgemein hat sich herausgestellt, dass Verfahren der Ordnung 1 bis 2 für die Schaltungssimulation gut geeignet sind. Verfahren höherer Ordnung weisen eine eingeschränkte Stabilität bezüglich der Schrittweite delta t auf, was für die praktische Schaltungsanalyse nur dann sinnvoll ist, wenn sehr wenig bis keine Schaltvorgänge in dem Simulationsintervall vorhanden sind. Da in integrierten Schaltungen Schaltvorgänge jedoch wichtig sind, werden diese Verfahren in der Praxis selten eingesetzt.

Analoge Simulation: Linearisierung

Linearisierung



Aus Taylor-Reihenentwicklung:

$$\mathbf{f}(\mathbf{x},\mathbf{u}) \approx \mathbf{f}(\mathbf{x}_0,\mathbf{u}) + \left.\frac{d\mathbf{f}(\mathbf{x},\mathbf{u})}{d\,\mathbf{x}}\right|_{\mathbf{x}_0} (\mathbf{x}-\mathbf{x}_0)$$

Jakobimatrix: $\mathbf{J}_{x0} := \left.\frac{d\mathbf{f}(\mathbf{x},\mathbf{u})}{d\,\mathbf{x}}\right|_{\mathbf{x}_0}$

Gesucht ist $\mathbf{x}$ *für:*

$$\mathbf{f}(\mathbf{x},\mathbf{u}) = \mathbf{0} \Longrightarrow \mathbf{f}(\mathbf{x}_0,\mathbf{u}) + \mathbf{J}_{x0} \cdot (\mathbf{x}-\mathbf{x}_0) = \mathbf{0}$$

$$\mathbf{x}1 = \mathbf{x}0 - (\mathbf{J}x0)\text{-}1 \cdot \mathbf{f}(\mathbf{x}0,\mathbf{u})$$

$$\mathbf{x}i+1 = \mathbf{x}i - (\mathbf{J}xi)\text{-}1 \cdot \mathbf{f}(\mathbf{x}i,\mathbf{u})$$

Lineares GS:

$$-\mathbf{J}xi \cdot \Delta\mathbf{x} = \mathbf{f}(\mathbf{x}i,\mathbf{u})$$

mit $\Delta\mathbf{x} := \mathbf{x}_{i+1} - \mathbf{x}_i$

Als gängiges Verfahren bei nichtlinearen Gleichungssystemen hat sich die Newton-Raphson Methode etabliert. Die Berechnungsvorschrift leitet sich aus der Bildung der Taylorreihe für das nichtlineare Gleichungssystem mit Abbruch nach dem ersten Glied ab. J bezeichnet die Jacobimatrix, die die partiellen Ableitungen von F(X,U) im aktuellen Iterationspunkt enthält. Die aus dem "Nullsetzen" der Taylor-Reihen-Näherung für F(X,U) entstehende iterative Rechenvorschrift führt zur "Newton-Raphson-Methode".

Analoge Simulation: Newton-Raphson-Methode

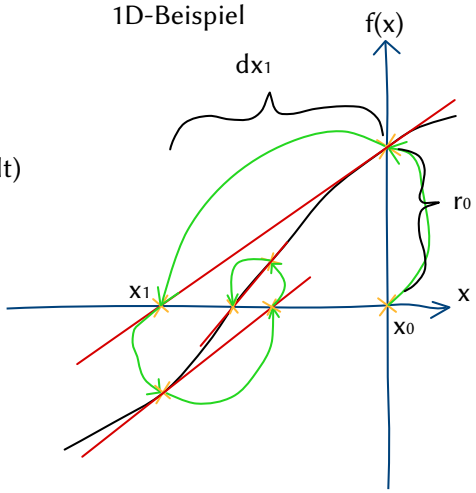
Newton-Raphson-Methode

BEGIN

Setze $i = 0$,
Schätze Initial-Lösung x_0
WHILE(Abbruchkriterium nicht erfüllt)
Berechne Residuum $r_i = f(x_i)$
Berechne Jakobimatrix J_i
Berechne $dx_{i+1} = -(J_i)^{-1} * r_i$
Berechne $x_{i+1} = x_i + dx_{i+1}$
Setze $i = i + 1$

END

1D-Beispiel



Abbruchkriterium:
$dxi < e_{absolut} + e_{relativ} * \min(x_i, x_{i-1})$
oder $ri < e_{relativ}$

Im Bild ist die Newton-Raphson-Methode beispielhaft für eine Funktion mit einer Variable f(x) gezeigt.

Zu Beginn müssen die Startwerte für die unbekannten Spannungen und Ströme (x0) geschätzt werden. Häufig werden alle Werte auf null gesetzt, außer denjenigen, die direkt mit einer Quelle verbunden sind und dem entsprechenden Erregungswert erhalten. Zudem gibt es bei allen modernen Simulatoren die Möglichkeit, Bauelementen Initialwerte zuzuweisen, wie z.B. eine Spannung an einem Kondensator. Ist dies der Fall, werden diese Initialwerte für die Schätzung der Startwerte berücksichtigt.

Dann wird als erster Schritt innerhalb der Iteration das Residuum r berechnet. Das Residuum bestimmt sich durch Einsetzen des geschätzten Wertes für x in das nichtlineare Gleichungssystem. Da es sich bei den Lösungswerten x während der Iteration immer um eine genäherte Lösung der wirklichen Lösung handelt, ergibt sich f(x,u) nie ganz zu Null. Diese Abweichung bezeichnet man als Residuum r.

Daraufhin wird die Jacobimatrix J bestimmt. Durch Einsetzen von J und r kann nun delta x berechnet werden. Im letzten Rechenschritt wird die neue Näherung aus den Werten der vorherigen Näherung und delta X berechnet. Nun ist nur noch das Update für die Iterationsvariable durchzuführen.

Für das Abbruchkriterium wird in der Regel die Kombination aus einem absoluten Fehlerkriterium mit einem relativen Fehlerkriterium verwendet. Da es in einigen Fällen jedoch trotz sehr geringer Variation in x noch zu starken Änderungen der Funktionswerte kommen kann, wird häufig ebenfalls das Residuum als Maß für die Änderung der Funktionswerte auf einen genügend kleinen Wert überprüft. Ist eine dieser Bedingungen nicht erfüllt, muss die Iteration fortgesetzt werden.

Analoge Simulation: ...Eigenschaften

Eigenschaften der Newton-Raphson-Methode

Quadratische Konvergenz, falls:
- Startlösung genügend genau
- Jakobimatrix lipschitzstetig
- Jakobimatrix für die exakte Lösung von X nichtsingulär

Die allgemeine Konvergenz ist jedoch nicht sichergestellt!

Bestimmung der Jakobimatrix:
- durch in Bauelementenmodellen implementierten Funktionen
- durch vom Benutzer anzugebende Funktionen
- durch Differenzquotienten
- durch symbolische Bildung bzw. automatische Ableitung der Funktionen.

Komplexität des Linearisierungsschrittes:
- Berechnung der nichtlinearen Bauelemente: O(N)
- Berechnung der Jakobimatrix aus N Variablen mit Sparse-Matrix-Techniken: O(N1.7)
- Berechnung des linearisierten Gleichungssystems: O(N1.5)

Das Newton-Raphsonverfahrens konvergiert bei genügend genauer Startlösung dann quadratisch, wenn die Jakobimatrix Lipschitz - stetig (die erste Ableitung der Funktion ist durch die so genennante Lipschitzkonstante L beschränkt; es gilt also |f(x)-f(y)|<=L|x-y| für alle x,y) und die Jakobimatrix für die exakte Lösung von X nichtsingulär ist. Die allgemeine Konvergenz ist jedoch nicht sichergestellt.

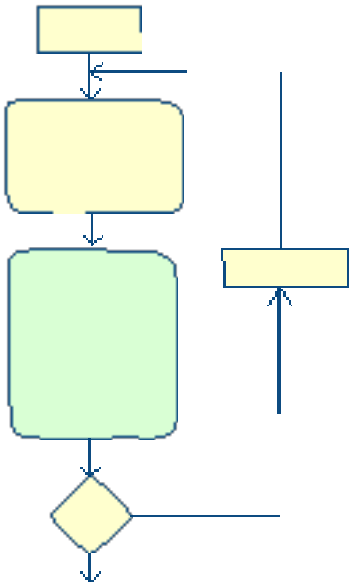
Die partiellen Ableitungen der Jacobimatrix können auf verschiedene Arten bestimmt werden: durch in Bauelementmodellen implementierte Funktionen, durch vom Benutzer anzugebende Funktionen, durch Differenzenquotienten oder durch symbolische Bildung bzw. automatische Ableitung aus den Funktionen. Alle Verfahren haben verschiedene Vor- und Nachteile. Gängig und mit guten Konvergenzeigenschaften behaftet sind bereits in den Bauelementmodellen implementierte Ableitungen, die jedoch die Verwendung anderer Modelle ausschließen. Die Verwendung von nutzereigenen Ableitungsfunktionen birgt das Risiko von Implementierungsfehlern und daraus resultierenden Konvergenzproblemen; zudem bedeutet es einen zusätzlichen Implementierungsaufwand für den Benutzer. Differenzenquotienten sind sehr einfach zu berechnen, führen jedoch zum Teil zu numerischen Instabilitäten. Die automatische Ableitung bzw. symbolische Bildung der Ableitungsfunktionen bedeutet einen erhöhten Rechenaufwand, ist dafür aber genau und wenig fehleranfällig.

Die Komplexität der DC-Analyse wird durch drei Faktoren bestimmt:
- Berechnung der nichtlinearen Bauelemente: O(N)
- Berechnung der Jacobimatrix aus N Variablen mit Sparse-Matrix-Techniken: $O(N_{1.7})$
- Berechnung des linearisierten Gleichungssystems: $O(N_{1.5})$

Insgesamt ergibt sich unter Verwendung von Sparse-Matrix-Techniken eine Komplexität von $O(N_{1.5...1.7})$.

Analoge Simulation: Lösen linearer Gleichungssysteme

Lösen linearer Gleichungssysteme



Verfahren zur Lösung linearer Gleichungssysteme:

- Gauss-Elimination
- LU-Zerlegung

Die Jakobimatrix ist bei realen Schaltungen in der Regel schwach besetzt.
→ Sparse-Matrix-Techniken

Zur Lösung eines linearen oder linearisierten Gleichungssystems existieren verschiedene Verfahren wie die Gauss-Elimination oder die LU-Zerlegung. Da die Matrix J bei der Simulation realer Schaltungen sehr schwach besetzt ist, kommen Sparse-Matrix-Techniken zum Einsatz, die bei der Lösung Rechenzeit und Speicherplatz sparen.

Analoge Simulation: DC-Analyse

DC-Analyse

Arbeitspunktbestimmung = Gleichstrom- = DC-Analyse:

- Statische Lösung des nichtlinearen Gleichungssystems
- im eingeschwungenen Zustand
- ohne Beachtung der Zeitabhängigkeiten
  - "Kurzschluss" der Induktivitäten
  - "Auftrennen" der Kapazitäten
- Lösung mit Newton-Raphson-Methode.

Berechnung von DC-Übertragungsfunktionen = DC-Transfer-Analyse

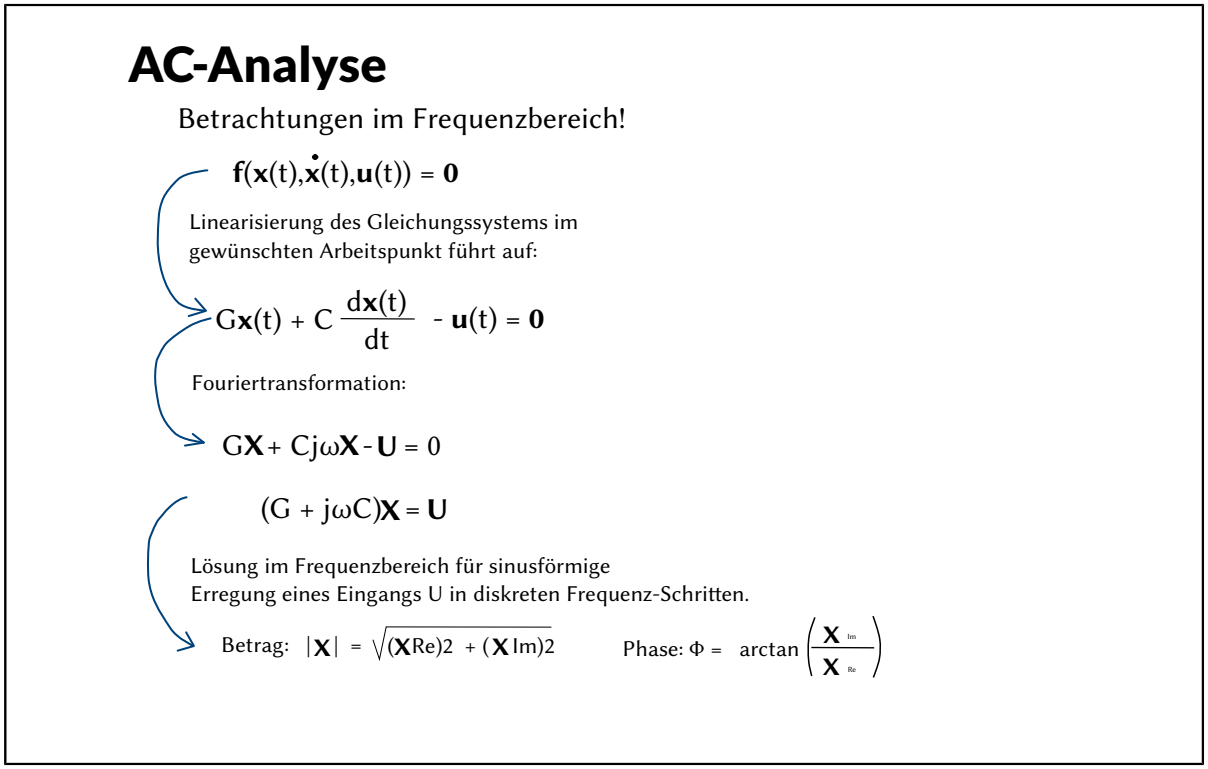
- Variationen einer Komponente des Eingangsvektors U
- Diskrete Schritte in festgelegten Werteintervall
- Für jeden Wert: DC-Analyse
- Ergebnis: Kurvenverlauf der Ausgangsgrößen über der variierten Eingangsgröße

Die Arbeitspunktbestimmung, auch Gleichstrom- oder DC-Analyse genannt, bestimmt die statische Lösung des nichtlinearen Gleichungssystems im eingeschwungenen Zustand ohne Beachtung der Zeitabhängigkeiten. Sie wird im Allgemeinen als Startlösung für alle anderen Analysen verwendet.

Es muss, wie bei der transienten Analyse, ein nichtlineares Gleichungssystem mit der Newton-Raphson-Methode gelöst werden. Da Zeitabhängigkeiten nicht betrachtet werden, können Spulen kurzgeschlossen und Kapazitäten weggelassen werden.

Neben der Arbeitspunktbestimmung ist die Berechnung von Übertragungsfunktionen, die so genannte DC-Transfer (DT) Analyse, eine wichtige Anwendung der Gleichstromanalyse. Hierbei wird eine Komponente des Eingangsvektors U in diskreten Schritten in einem festgelegten Werteintervall verändert. Für jeden neuen Wert dieses Intervalls wird die DC-Analyse durchgeführt, so dass nach Beendigung aller DC-Analysen der Kurvenverlauf einer beliebigen Ausgangsgröße in Abhängigkeit von der variierten Eingangsgröße als nichtlineare Übertragungsfunktion zur Verfügung steht.

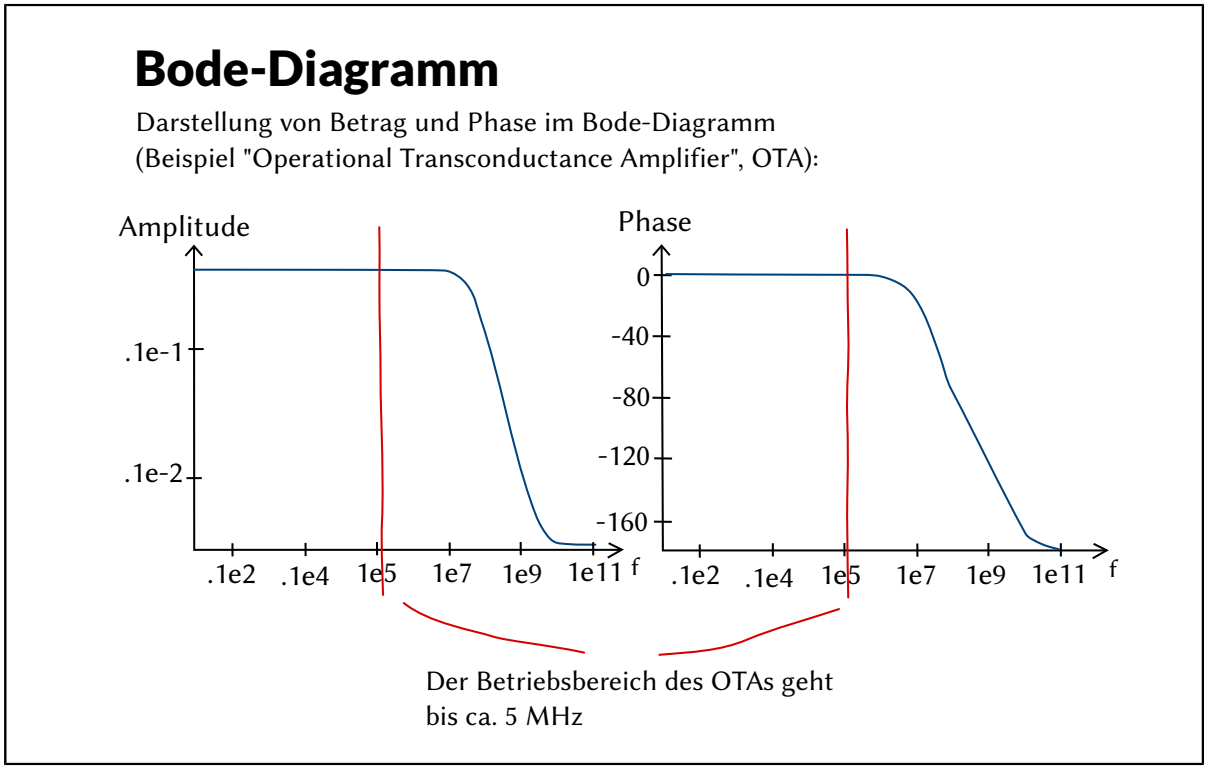
Analoge Simulation: AC-Analyse



Bei der AC-Analyse werden die Systemgleichungen im Frequenzbereich berechnet. Dazu wird zunächst das nichtlineare Gleichungssystem im gewünschten Arbeitspunkt linearisiert. Mit Hilfe der Fouriertransformation wird es in den Frequenzbereich überführt.

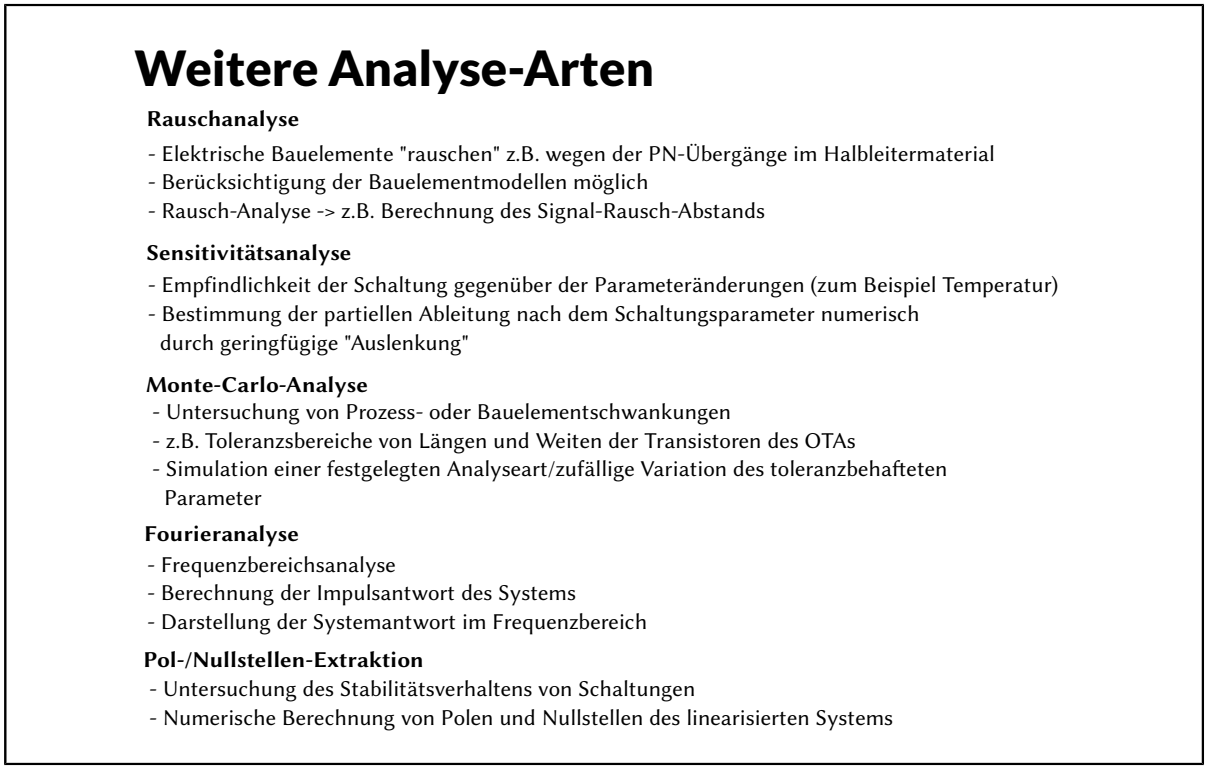
Am Eingang der Schaltung werden nun sukzessive sinusförmige Erregungen verschiedener Frequenzen in diskreten Schritten angenommen. Das Gleichungssystem wird in jedem Frequenzpunkt berechnet, so dass als Ergebnis in jedem Frequenzpunkt die Phase und der Betrag der Schaltungsvariablen zur Verfügung stehen, die als Bodediagramm dargestellt werden können.

Analoge Simulation: Bode-Diagramm



In Bodediagramm des OTA-Beispiels sind Verlauf von Betrag und Phase beispielhaft zu sehen. Der OTA hat einen Übertragungsbereich bis zu einer Frequenz von ca. 5 MHz.

Analoge Simulation: Weitere Analyse-Arten



Rauschanalyse

Jedes elektrische Bauelement weist einen bestimmten Anteil an Rauschen auf. Eine Rauschart wird beispielsweise durch PN-Übergänge im Halbleitermaterial hervorgerufen. Rauschen kann in den Bauelementmodellen bereits mit modelliert werden. Soll dieses bei der Analyse berücksichtigt werden, muß die Rauschart im Parametersatz mit angegeben werden. Es können dann spezielle Kennwerte wie Signal-Rausch-Abstand berechnet werden.

Sensitivitätsanalyse

Bei der Sensitivitätsanalyse wird die Empfindlichkeit der Schaltung gegenüber Parameteränderungen berechnet. Der zu untersuchende Parameter wird geringfügig ausgelenkt und numerisch die partielle Ableitung nach dem zu untersuchenden Parameter gebildet.

Monte-Carlo-Analyse

Die Monte-Carlo Analyse dient der Untersuchung von Prozess- oder Bauelementeschwankungen. Hierzu werden beispielsweise die Längen und Weiten der Transistoren des OTAs mit Toleranzen angegeben, sowie eine Wahrscheinlichkeitsverteilung für die Abweichungen vom Nominalwert der Längen und Weiten innerhalb der angegebenen Toleranz und die Anzahl der durchzuführenden Experimente. Der Simulator variiert dann zufällig vor jeder Simulation mit der vorgegebenen Wahrscheinlichkeitsverteilung die Bauelementewerte neu. Sind alle Simulationen abgeschlossen, kann man z.B. den Ausgangsstrom des OTAs in Abhängigkeit von den Variationen der Bauelementwerte betrachten und daraus Rückschlüsse für Matching, Größe der Transistorlängen und -weiten u.ä. ziehen.

Fourieranalyse

Die Fourieranalyse ist ebenfalls eine Frequenzbereichsanalyse. Allerdings wird, anders als bei der AC-Analyse, die Antwort auf einen Dirac-Stoß in Form eines Spektrums der Frequenzanteile des Signals dargestellt.

Pol-/Nullstellen Extraktion

Hinweise über das Stabilitätsverhalten von Schaltungen lassen sich den Polen und Nullstellen erhalten, die sich aus der Übertragungsfunktion errechnen lassen. Liegen alle Pole in der linken Halbebene, so arbeitet die Schaltung stabil, sonst kann es zu ungewollten Oszillationen kommen. Einige moderne Simulatoren haben die Möglichkeit, die Pole und Nullstellen numerisch zu bestimmen.

Electronic Design Automation (EDA)

Mixed Signal Simulation

Mixed-Signal-Simulation

Erweiterte Simulationskerne

Kopplung von Simulatoren

Analog nach Digital

Digital nach Analog

Zeitschrittsynchronisation

Lockstep/Backtracking

Mixed Signal Simulation: Mixed-Signal-Simulation

Mixed-Signal-Simulation

- Die Bedeutung von gemischt analog/digitalen Schaltungen hat stark zugenommen.

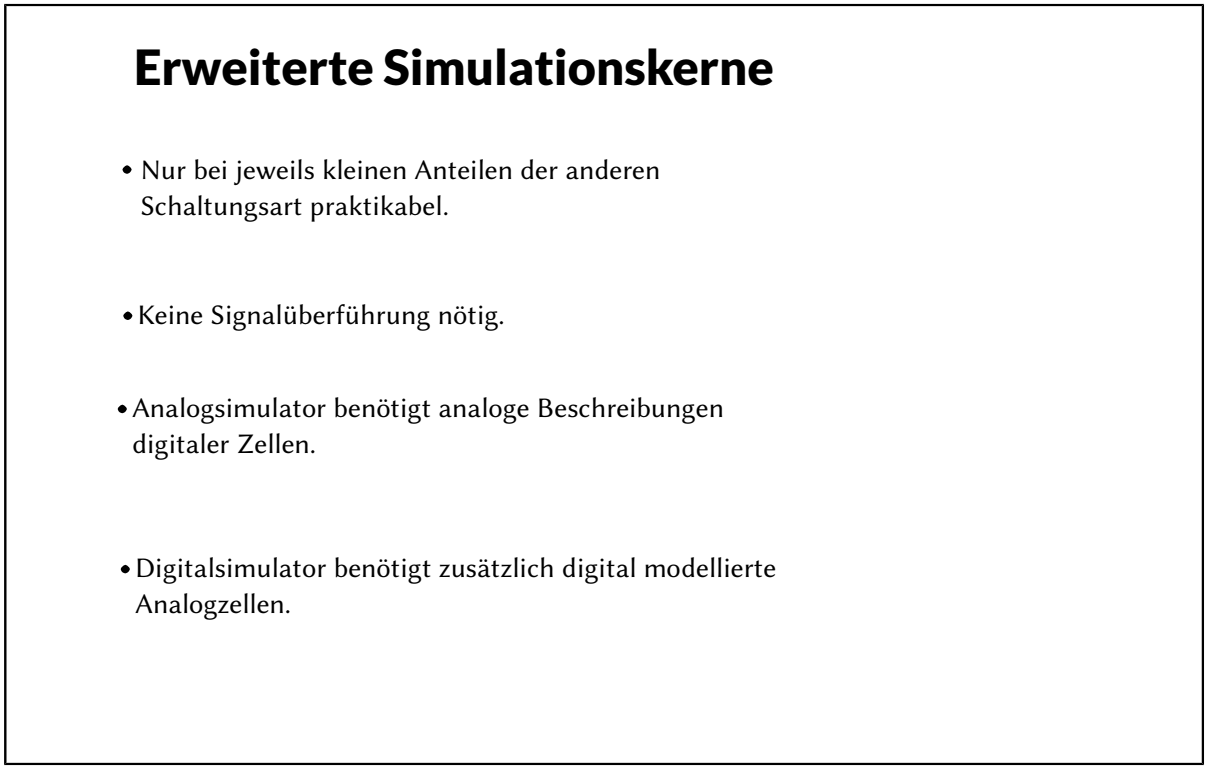
- Analog-/ und Digitalsimulation beruhen auf unterschiedlichen Verfahren:
  - analog: Lösen umfangreicher Differentialgleichungssysteme mit numerischen Vefahren.
  - digital: zeit- und wertdiskret, Auswertung einfacher Datenstrukturen zu diskreten Zeitpunkten.

- Die aufwendigen Berechnungen der Analogsimulation arbeiten langsamer als die Digitalsimulation.

In den letzten Jahren hat die Bedeutung von gemischt analog/digitalen Schaltungen stark zugenommen. Die Simulation solcher Schaltungen ist jedoch problematisch, da die Verfahren für Analog- und Digitalschaltungen sehr unterschiedlich sind. Während bei einem Analogsimulator komplexe Differentialgleichungsysteme mit numerischen Verfahren gelöst werden, können sich digitale Simulatoren aufgrund verschiedener Eigenheiten digitaler Schaltungen (zeit- und wertdiskrete Signale) auf die Auswertung einfacher Datenstrukturen zu diskreten Zeitpunkten beschränken (Ereignissteuerung).

Die Schwierigkeit einer gemischten Simulation (Mixed-Signal-Simulation) liegt insbesondere in der unterschiedlichen Geschwindigkeit, mit der die beiden Schaltungsarten ausgewertet werden können. Im Idealfall soll es möglich sein, den gesamten Leistungsumfang des Digitalsimulators zu nutzen, ohne die Präzision bei den analogen Schaltungsberechnungen einzuschränken.

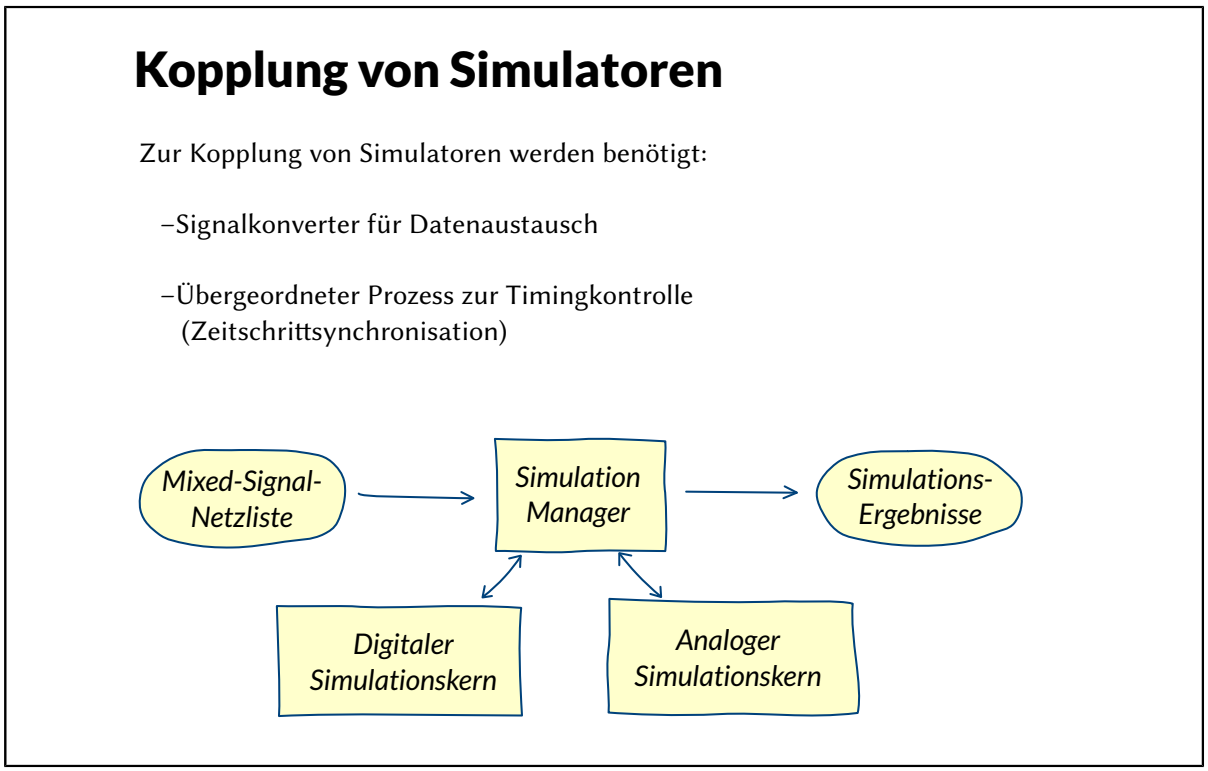
Mixed Signal Simulation: Erweiterte Simulationskerne



Besteht eine Schaltung überwiegend aus Bauelementen einer der beiden Welten, ist es praktikabel, die Fähigkeiten bestehender analoger bzw. digitaler Simulatorkerne um die Komponenten der jeweils anderen Welt zu erweitern. Eine Umrechung in die jeweils andere Signaldarstellung ist dadurch nicht notwendig.

So erweiterte Analogsimulatoren beinhalten analoge Beschreibungen digitaler Elemente und liefern Simulationsergebnisse ohne Präzisions- und Laufzeitverluste. Nicht ohne weiteres universell einsetzbar sind dagegen erweiterte Digitalsimulatoren. Für sie müssen die analogen Bauteile je nach Anwendung mit den Beschreibungsmöglichkeiten des Simulators digital modelliert werden.

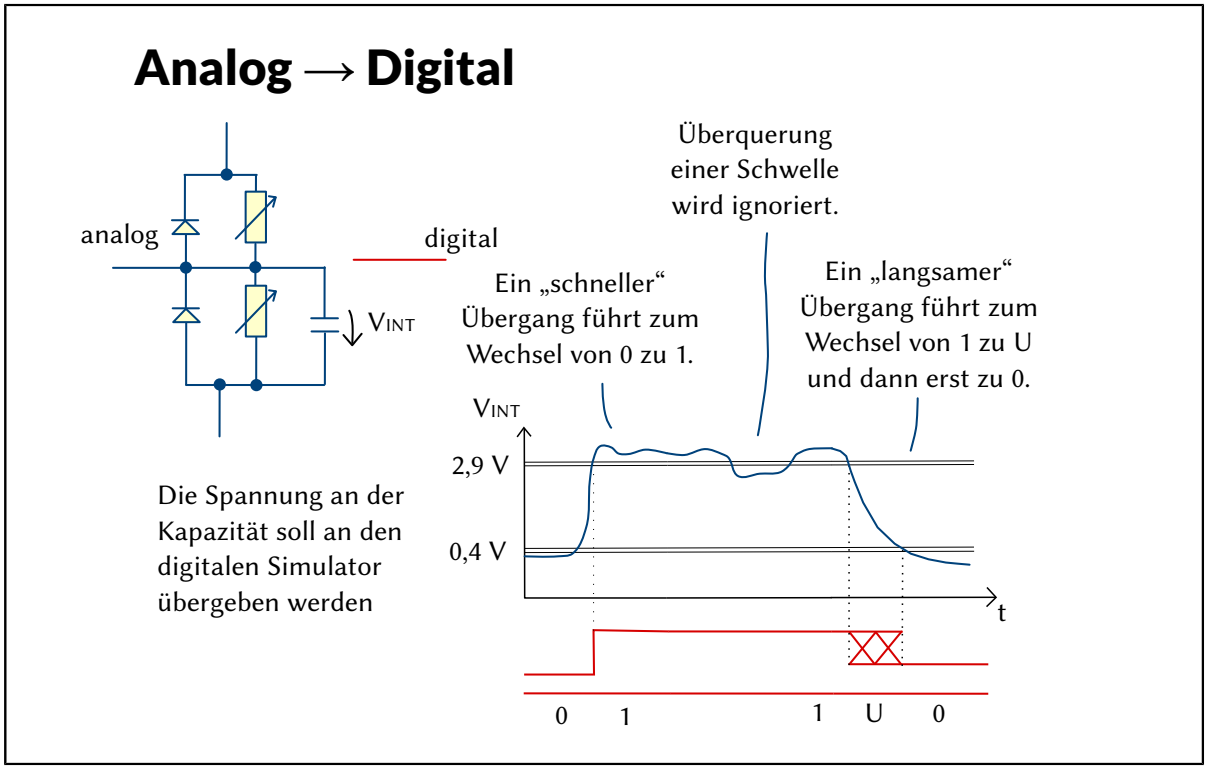
Mixed Signal Simulation: Kopplung von Simulatoren



Ein weiterer Ansatz ist die lose Kopplung eines digitalen mit einem analogen Simulator. Allgemein ist es hier notwendig, jeden Verbindungspunkt zwischen Analog- und Digitalsimulator über eine Konverterstruktur zu führen. Durch die dort erfolgende Wertkonvertierung werden die digitalen auf analoge Signale und die analogen auf digitale Signale abgebildet.

Die Kopplung wird durch einen übergeordneten Prozess vorgenommen, der den Datenaustausch kontrolliert. Seine Aufgabe besteht in der Synchronisation beider Simulatoren, die durch eine Abstimmung der Zeitschritte in den Berechnungen erfolgt (Zeitschrittsynchronisation). Zusätzlich kann durch diesen Prozess auch eine einheitliche Netzlistenverwaltung implementiert werden.

Mixed Signal Simulation: Analog nach Digital



Die Schnittstelle von der analogen zur digitalen Welt gestaltet sich vergleichsweise einfach. Im Allgemeinen verbindet man logische Zustände mit definierten Spannungen (beispielsweise 0 V für logisch Null und 3.3 V für logisch Eins). In der Praxis wird ein bestimmter Grenzwert (Threshold-Spannung) für die interne Spannung an der Schnittstelle definiert, ab der sich ein Zustandswechsel vollzieht. Tritt während der Simulation eine Grenzwertüberschreitung auf, wird für den Digitalsimulator ein entsprechendes Ereignis (Event) erzeugt.

Durch Definition von zwei Grenzwerten ist es zusätzlich möglich, den im Digitalen verwendeten U-Zustand (unsicherer Zustand) einzuführen. Betrachtet man die oben dargestellte beispielhafte Konvertierung, so erfolgt bei einem hinreichend schnellen Anstieg der Spannung über die Grenzwerte von 0.4 V nach 2.9 V ein Zustandswechsel von logisch 0 nach logisch 1. Bei langsamen Transitionen wird jedoch der U-Zustand generiert.

Modifiziert man den vorgestellten Ansatz, ist es möglich, Gatter mit Hysterese-Eingängen nachzubilden. Ein Signalwechsel von Null nach Eins erfolgt hier bei einem Grenzwert von 2.9 V. Befindet sich das Gatter im Logisch-1-Zustand, erfolgt der Wechsel zu Null jedoch erst wieder bei einem Spannungswert von 0.4 V. Als Gatter mit einer derartigen Hysterese sei beispielspielsweise ein einfacher Schmitttrigger genannt.

Mixed Signal Simulation: Digital nach Analog

Digital → Analog

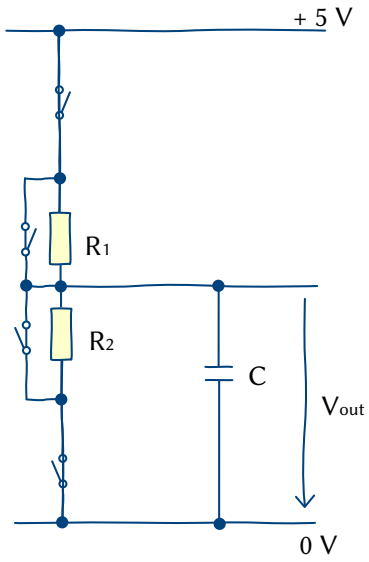
Probleme:

„Digitale Sprünge“ führen zu numerischen Problemen im Analogsimulator.
→ Modellierung der Übergänge durch Flanken mit endlicher Steigung.

Behandlung des X-Zustands.
→ X-Zustand muss in realen Wert umgewandelt werden.

Behandlung des U- und Z-Zustandes:

Digital	R1	R2	V_{out}
0	hoch	niedrig	nahe 0V
1	niedrig	hoch	nahe 5V
U	niedrig	niedrig	nahe 2,5V
Z	hoch	hoch	hochohmig



Die Konvertierung digitaler in analoge Signale gestaltet sich schwieriger, da am Analogteil nicht einfach die digitalen Signalsprünge umgesetzt werden können. Die resultierenden Unstetigkeiten im Signalverlauf könnten durch die numerischen Verfahren des Analogsimulators nicht behandelt werden. Stattdessen müssen beim Signalwechsel fallende bzw. steigende Flanken modelliert werden, die in ihrem Verlauf der verwendeten Technologie angepasst werden können. Weiterhin ist es wünschenswert, die digitalen Signalwerte U, X und Z (hochohmiger Ausgang) in die Betrachtung mit einzubeziehen.

Der nicht bekannte Zustand X (das Signal sei 1 oder 0) ist in den analogen Modellen nicht vorhanden. Zu seiner Abbildung in ein analoges Signal existieren einfache Wege, die je nach Einsatz und Anwendung gewählt werden. So wird X beispielsweise als mittlerer Pegel oder als 0- oder 1-Pegel behandelt, der letzte bekannte Wert des Knotens wird beibehalten oder 0- oder 1-Pegel werden willkürlich zugewiesen. Allerdings sind die so gewonnenen Simulationsergebnisse für die Gesamtschaltung zweifelhaft. Zumindest sollte der Entwickler beim Eintreffen eines nicht bekannten Zustands an der Schnittstelle eine Warnung erhalten, wodurch ihm Gelegenheit gegeben wird, den digitalen Teil der Schaltung so zu dimensionieren, dass dieser Zustand die Schnittstelle nicht erreicht.

Der unsichere Zustand "U" kann besser abgebildet werden, z.B. durch einen mittleren Spannungspegel.

Ein hochohmiger Zustand impliziert, dass zwischen dem treibenden Gatter und der Schnittstelle ein offener Schalter existiert. Eine praktische Umsetzung einer Schnittstelle, die den Z-Zustand mit einbezieht, ist oben dargestellt. Die Widerstände R1 und R2 werden abhängig vom gewünschten Zustand geschaltet. Wählt man für R1 einen großen und für R2 einen kleinen Wert entsprechend des Ausgangswiderstandes des Logik-Gatters, so stellt sich eine Ausgangsspannung nahe 0 V ein. Dies entspricht einem Logisch-0-Zustand. Für eine logische 1 werden R1 und R2 umgekehrt gewählt. Sind beide Widerstandswerte groß, ist der Ausgang praktisch vom Digitalteil getrennt und somit hochohmig. Weiterhin kann die Ausgangskapazität des Logikgatters durch die Kapazität C modelliert werden.

Mixed Signal Simulation: Zeitschrittsynchronisation

Zeitschrittsynchronisation

- Kopplung analoger und digitaler Simulatoren erfordert Synchronisation der ereignis- und zeitschrittorientierten Abläufe.
- Einfache aber langsame Lösung:
  - Analoge Zeitschritte → Digitale Ereignisse
- Schnelle aber schwierige Lösung:
  - Ereignisgesteuerter Analogsimulator
- Schnellere Lösungen:
  - Lockstep-Algorithmus
  - Backtracking-Algorithmus

Ein weiteres Problem bei der Kopplung analoger und digitaler Simulatoren stellt die Synchronisation der ereignis- und zeitschrittorientierten Simulationsabläufe dar. Sie muss dafür Sorge tragen, dass die Geschwindigkeit eines digitalen Simulators erhalten bleibt, ohne dass für das Schaltverhalten wesentliche Signale im analogen Bereich ignoriert werden. Die einfachste Form der Synchronisation der Zeitverwaltung besteht darin, die Zeitschrittsteuerung des Analogsimulators auch auf den digitalen Bereich zu übertragen. Diese Art der Verbindung ist leicht durchzuführen und liefert darüber hinaus die aus der Analogsimulation gewohnt präzisen Ergebnisse. Sie hat aber den Nachteil, dass der Simulator extrem langsam wird.

Die Angleichung der analogen Simulation an die Ereignissteuerung verspricht eine Simulationsgeschwindigkeit, die der Leistungsfähigkeit digitaler Simulatoren entspricht. Allerdings ist sie nur sehr schwer zu verwirklichen, denn die Ereignissteuerung basiert auf der Tatsache, dass eintreffende Ereignisse - im Gegensatz zu analogen Signalen - vorausberechenbar sind. Im Folgenden werden zwei Ansätze vorgestellt, die eine solche Synchronisation nährungsweise verwirklichen.

Mixed Signal Simulation: Lockstep/Backtracking

Lockstep / Backtracking

Lockstep-Algorithmus:

- Ein Zeitraster wird für beide Simulatoren vorgegeben.
- Der Analogsimulator führt bei Konvergenzproblemen zusätzliche Zwischenschritte durch.
- Bei digitalen Ereignissen zwischen vorgegebenen Zeitschritten: → Anpassung der Zeitschritte für beide Simulatoren.

Backtracking-Algorithmus:

- Die Simulatoren arbeiten zunächst unabhängig voneinander.
- Digitale Events, die den Analogteil beeinflussen, lösen ein Backtracking aus: Analoge Simulationsergebnisse werden zum Teil verworfen und die Simulation wird zu einem Zeitpunkt vor dem digitalen Ereignis neu gestartet.

Der Lockstep-Algorithmus gibt beiden Simulatoren ein festes Zeitraster vor. Sowohl Analog-, als auch Digitalsimulator werten ihren Schaltungsteil im Regelfall zu gleichen Zeitpunkten aus. Werden jedoch bei der Analogsimulation zusätzliche Zwischenschritte benötigt, um Konvergenzprobleme oder Rundungsfehler handhaben zu können, werden diese durchgeführt. Treten umgekehrt zwischen den Zeitschritten digitale Ereignisse auf, werden die Intervalle des Lockstep-Algorithmus für beide Simulatoren entsprechend angepasst.

Beim Backtracking-Ansatz werden beiden Simulatoren keine festen Zeitschritte vorgegeben. Der Analogsimulator ist zunächst unabhängig vom Auftreten digitaler Events, wodurch gegenüber dem Lockstep-Algorithmus ein Zeitgewinn zu verzeichnen ist. Treten jedoch digitale Events auf, die direkt die analoge Schaltung beeinflussen, muss ein Teil der bereits ermittelten Simulationsergebnisse verworfen werden und zu einem Zeitpunkt vor dem digitalen Event zurückgesprungen werden (Backtracking).

Das Backtracking bietet insbesondere auf Mehrprozessorsystemen Vorteile, da beide Simulatoren weitgehend unabhängig arbeiten und dadurch jeweils eine CPU für sich optimal ausnutzen können. Bei Schaltungen mit besonders großer Interaktion zwischen dem Digital- und dem Analogteil wird die Performance jedoch wiederum reduziert, da durch das Backtracking Simulationen wiederholt werden müssen. Ein weiterer Nachteil sind die relativ großen Datenmengen, die pro Simulationsschritt vorgehalten werden müssen. Um eine Berechnung zu einem früheren Zeitpunkt durchführen zu können, muss jeweils die vollständige Schaltung gespeichert werden.

Electronic Design Automation (EDA)

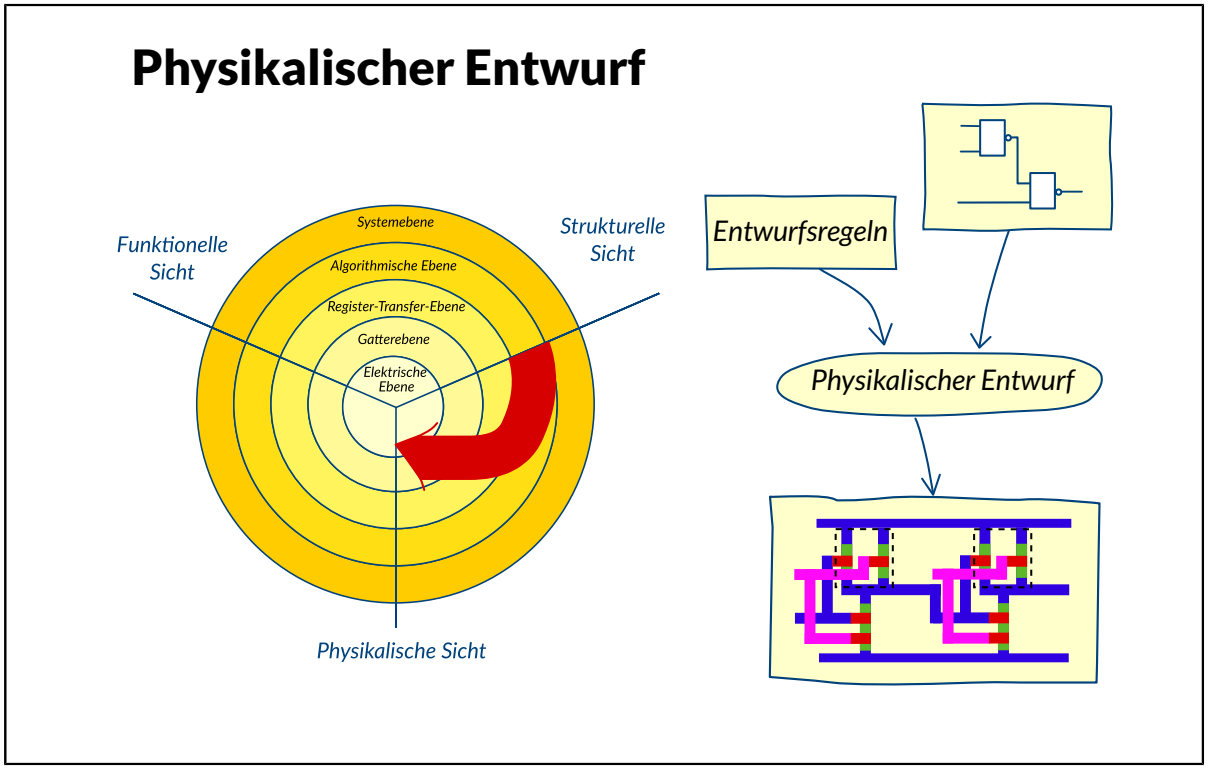
Das Layoutproblem

Physikalischer Entwurf

Das Layoutproblem

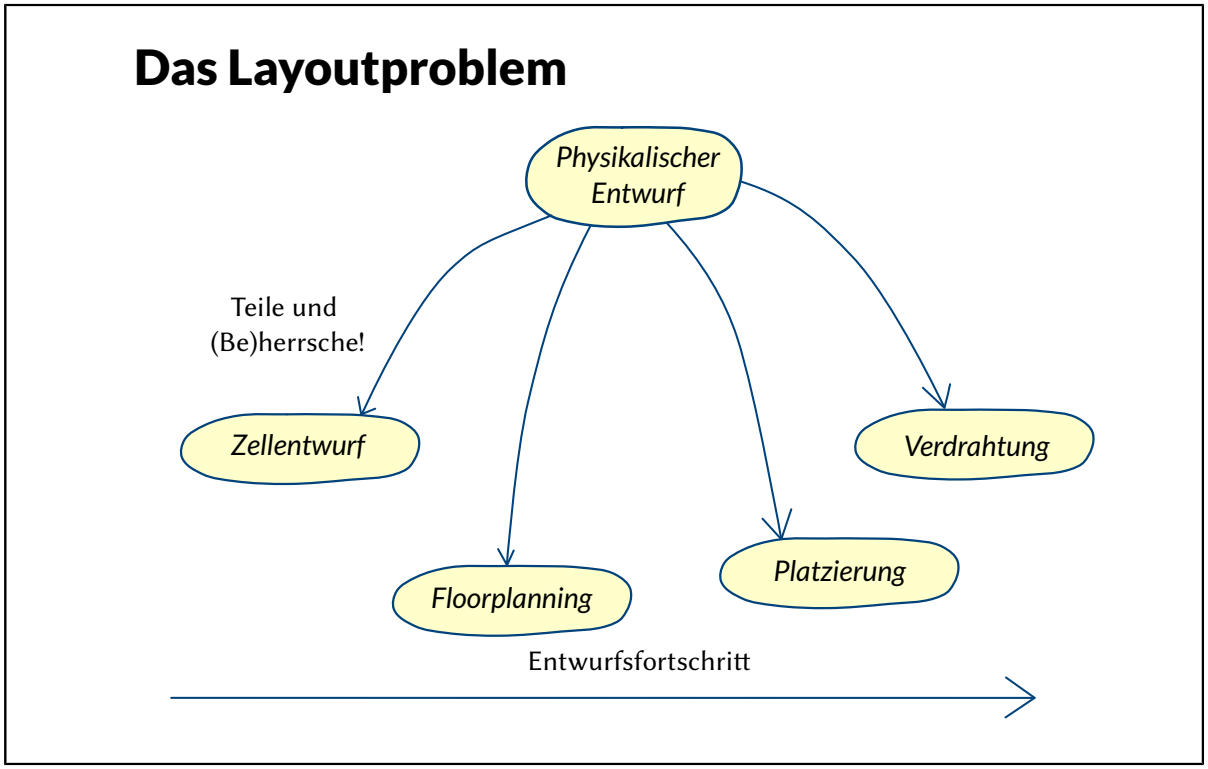
Date of generation: 8.7.2025

Das Layoutproblem: Physikalischer Entwurf



Das Ziel des Entwurfs von integrierten Schaltungen ist die Erzeugung der geometrischen Strukturen, die für die anschließende Halbleiterfertigung benötigt werden. Die Schaltung liegt nach dem funktionellen Entwurf in der Regel als strukturelle Beschreibung auf Gatterebene oder elektrischer Ebene vor. Benötigt wird eine physikalische Beschreibung auf elektrischer Ebene - das so genannte Schaltungslayout. Das Layout besteht aus geometrischen Beschreibungen der Komponenten der Schaltung und ihrer Verbindungen. Das Layout muss Entwurfsregeln (Design Rules) genügen, die vom Halbleiterherstellungsprozess abgeleitet werden.

Das Layoutproblem: Das Layoutproblem



Der physikalische Entwurf ist ein sehr komplexer Prozess, der in der Regel in eine Abfolge handhabbarer kleiner Entwurfsschritte unterteilt werden muss, damit er überhaupt bewältigt werden kann.

Der Zellentwurf ist die Voraussetzung für die Abstraktion, die zur Beherrschung des Layoutproblems z.B. auf Gatterebene erforderlich ist. Durch den Entwurf des Layouts der einzelnen Gatter entsteht eine Zell-Bibliothek, die die Objekte für die Algorithmen zum Erzeugen des geometrischen Layouts der Gesamtschaltung bereitstellt.

Die Anordnung dieser Zellen auf einer Layoutfläche ist das Platzierungsproblem. Wenn die Menge der zu platzierenden Zellen zu groß wird, um mit derzeit existierenden Algorithmen noch ein befriedigend gutes Ergebnis zu erzielen, ist vor der eigentlichen Platzierung ein Floorplanning erforderlich. Dazu wird die Gesamtschaltung in größere Blöcke partitioniert, eine möglichst günstige Anordnung der Blöcke gefunden und anschließend die einzelnen Blöcke nacheinander an den Platzierungsalgorithmus übergeben. Nach der Platzierung der Zellen der Makros und dem Speicher folgt der Verdrahtungsschritt, bei dem die erforderlichen Verbindungen zwischen den auf der Layout-Fläche platzierten Zellen erzeugt werden müssen.

Durch das Teile-und-Hersche-Prinzip kann für das sehr komplexe Layoutproblem auch für sehr große Schaltungen in ausreichend kurzer Zeit eine befriedigende Lösung gefunden werden.

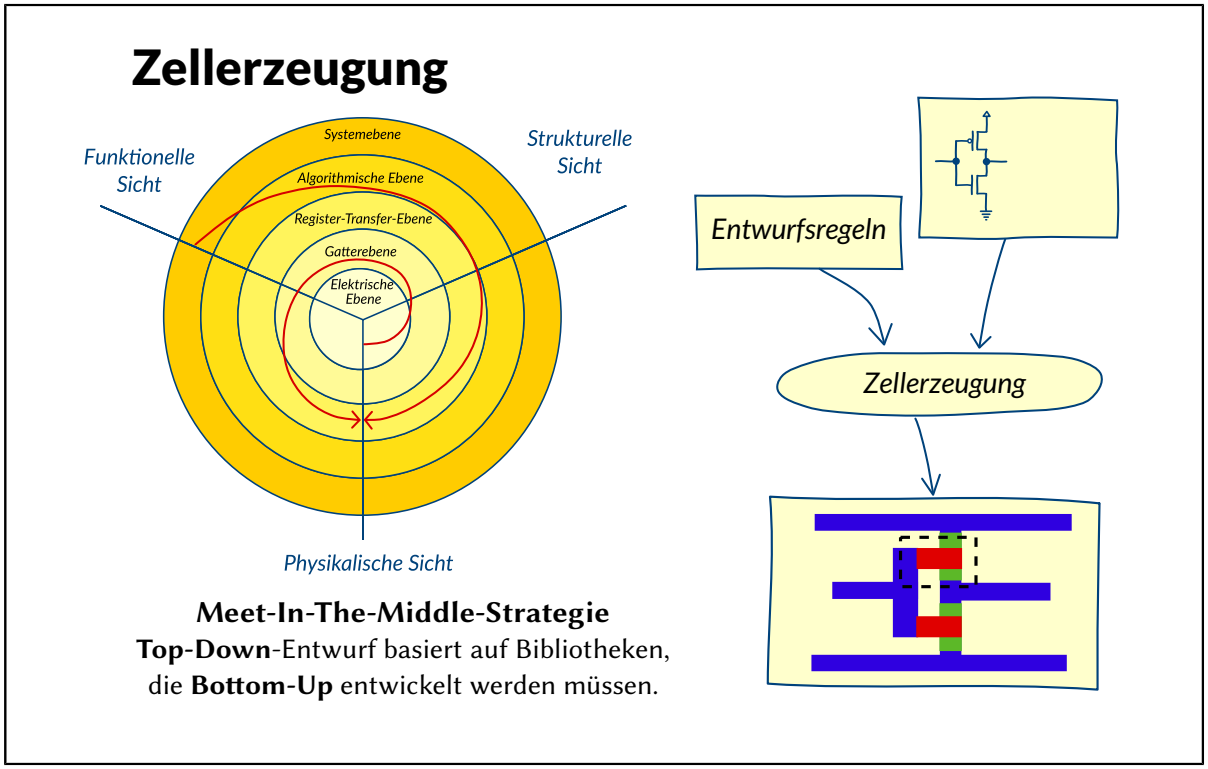
Electronic Design Automation (EDA)

Zellerzeugung

Zellerzeugung
Standardzellen
Standardzellen Beispiele
Manueller Entwurf
Kopplung Schematic/Layout
Zellgeneratoren
Physikalische Parametrierung
Funktionale Parametrierung

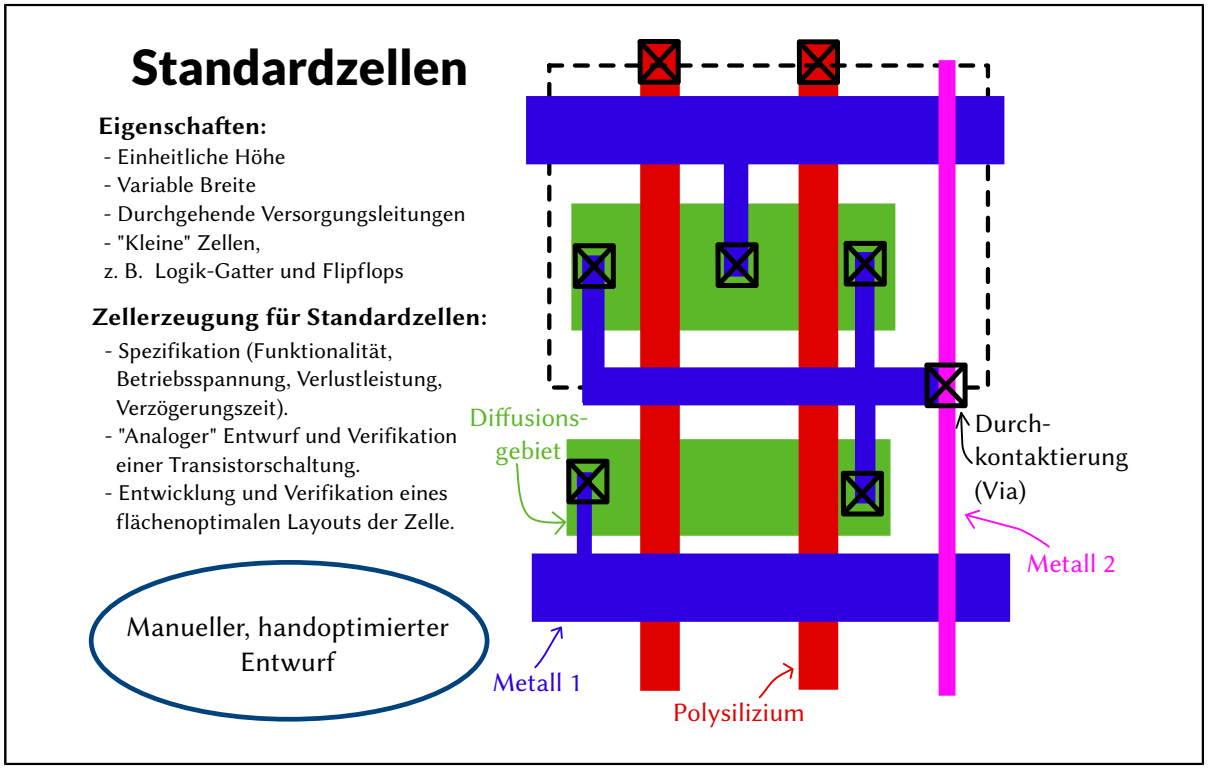
Date of generation: 8.7.2025

Zellerzeugung: Zellerzeugung



Um einen Meet-In-The-Middle Entwurfsablauf durchführen zu können, sind für alle Komponenten der "Mittelebene" (in der Regel die Gatterebene) Bibliothekselemente zur Verfügung zu stellen, die alle für den Entwurf erforderlichen Informationen in allen Sichten enthalten (Bibliotheksentwurf). Insbesondere muss für jede Komponente ein Layout vorhanden sein. Auf Gatterebene verwendet man in der Regel so genannte Standardzellen.

Zellerzeugung: Standardzellen



Standardzellen sind durch eine einheitliche Bauhöhe gekennzeichnet, so dass sie sich in Reihen aneinander ordnen lassen und zu einem einfachen Verdrahtungsschema führen. Die Breite der Zellen variiert entsprechend ihrer Komplexität. Die Verdrahtung der Versorgungsspannungen vereinfacht sich stark; durch Aneinanderfügen sind die Versorgungsspannungen verschiedener Zellen verbunden. Die logischen Anschlüsse der Zellen werden an einer oder beiden Seiten herausgeführt. Auch diese können später leicht angeschlossen werden. Diese Struktur eignet sich jedoch nur für kleine Zellen, wie Gatter, Register, Zähler u.ä. Größere Blöcke, wie Speicher, passen nicht in das Strukturschema einer einheitlichen Bauhöhe und können daher nur separat realisiert werden. Dieses Problem wird bereits im Floorplanning betrachtet.

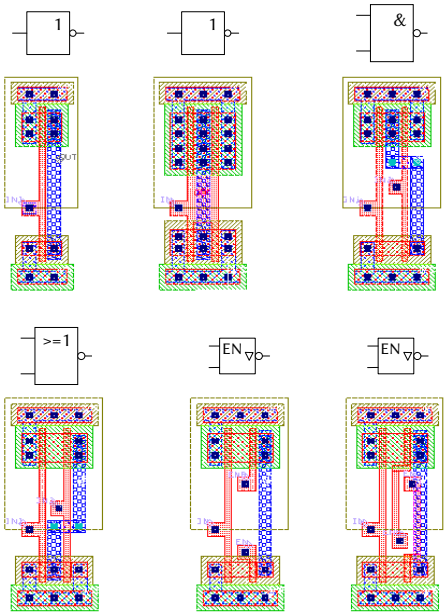
Zunächst ist für jede Zelle eine Spezifikation zu erstellen. Neben der Grundfunktionalität des Gatters z.B. NAND oder NOR-Gatter sind Betriebsspannung, Verlustleistung und z.B. Verzögerungszeit zu spezifizieren. Nun wird für jede Zelle eine Transistorschaltung entwickelt und verifiziert, die die geforderten Eigenschaften erfüllt. Schließlich sind die Layouts für diese Zellen zu entwickeln. Diese werden dann in der Bibliothek zur Verwendung von Platzierungs- und Verdrahtungswerkzeugen abgelegt.

Das Layout einer Zelle selbst sollte auf Grund seiner häufigen Verwendung flächenoptimiert sein. Aus diesem Grund werden diese Layouts in der Regel manuell erstellt.

Zellerzeugung: Standardzellen Beispiele

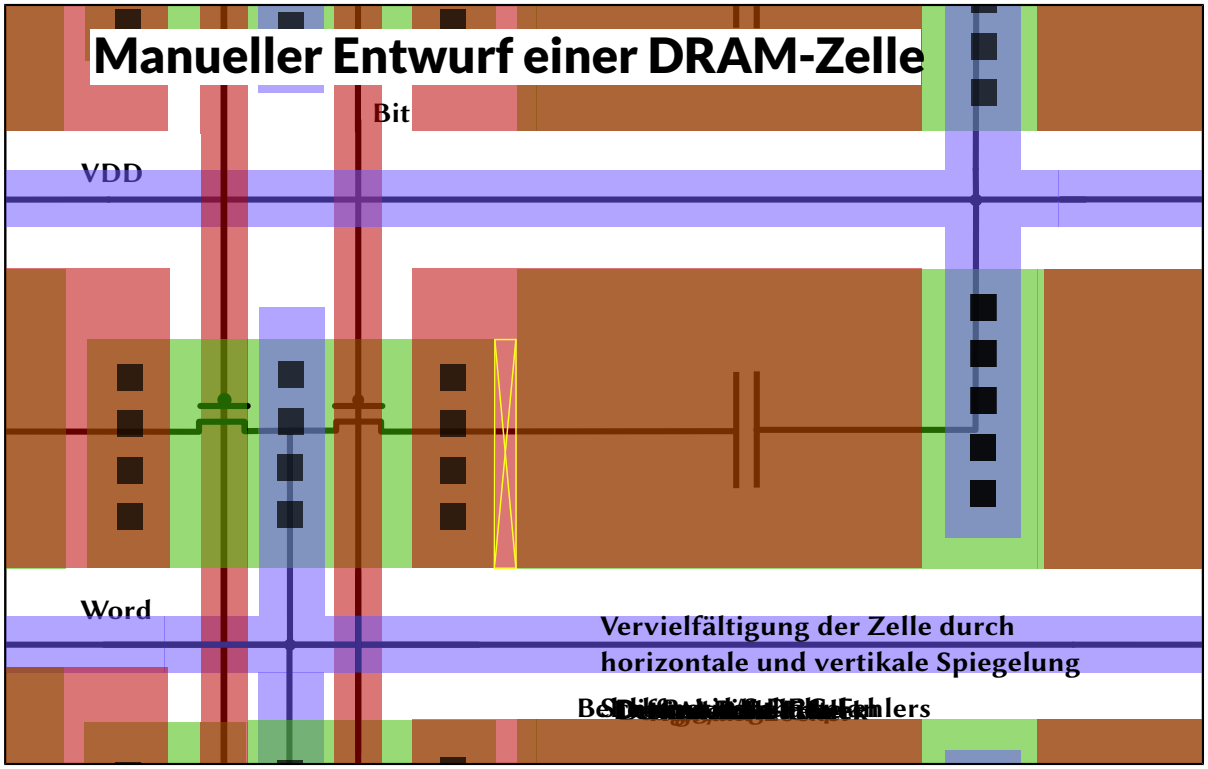
Standardzellen: Beispiele

Beispiele für die einfachsten Logik-Gatter



Hier sind einige einfache Standardzellen von einfachen Logikgattern aufgeführt. In typischen Bibliotheken sind ca 50-200 solcher (meist etwas komplexerer) Standardzellen enthalten. In unserem Beispiel sind zwei Inverter mit unterschiedlicher Stärke (Fan Out), je ein NAND und NOR Gatter sowie zwei Tristate-Treiber, die die gleiche Funktion, aber unterschiedliche Anschlusslagen zur Verdrahtung aufweisen.

Zellerzeugung: Manueller Entwurf



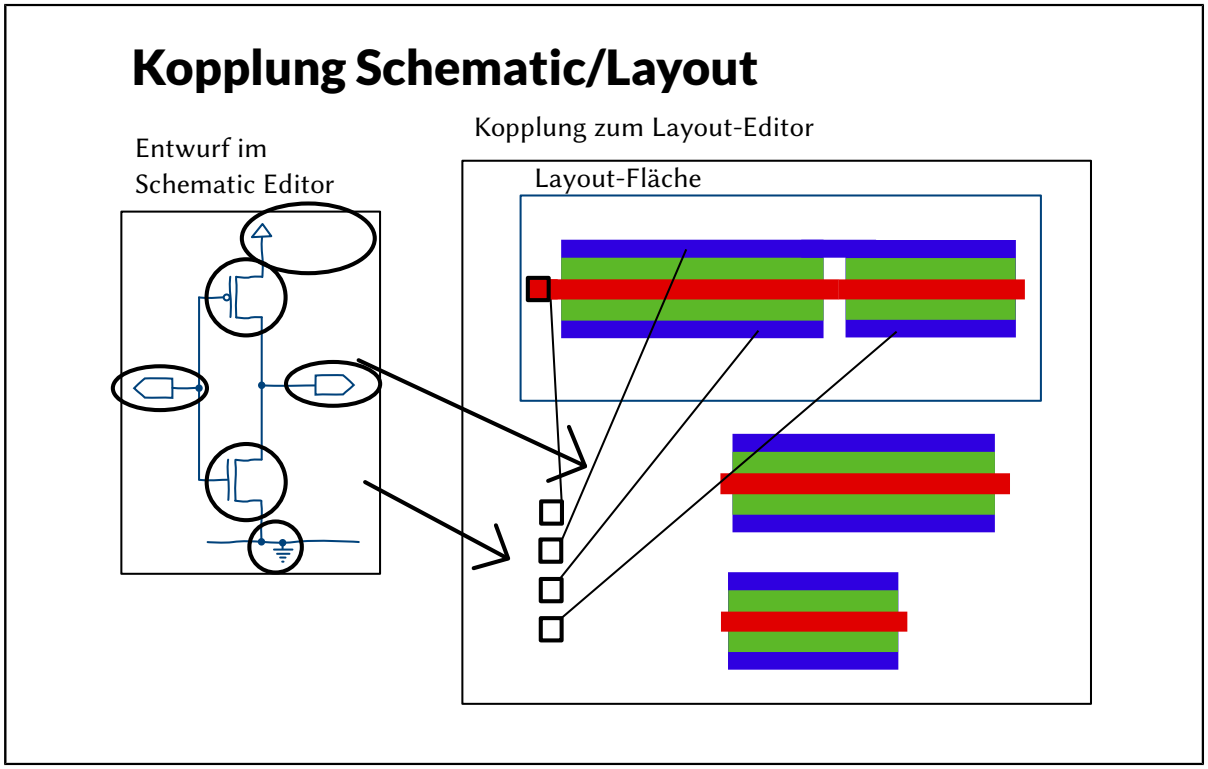
Ist es bei endlicher Komplexität erforderlich, die Chipfläche zu optimieren, ist nach wie vor der Mensch allen automatischen Verfahren überlegen. Für viele Anwendungen (Standard , Analogschaltungen, Zellentwurf) sind deshalb nach wie vor manuelle Verfahren dominierend, d.h. die detaillierte Geometrie aller Masken wird vom Menschen entworfen. Die Arbeitstechnik hat sich dabei seit den 60er Jahren über die Verwendung von Bleistift und Radiergummi, über die Digitalisierung von Handentwürfen sowie den ersten an Großrechner angeschlossenen Grafiksystemen (Calma, Computervision) zu interaktiven intelligenten Grafikeditoren auf Workstations entwickelt. Dem Layouter stehen dort geometrische Grundstrukturen zur Verfügung, die er abrufen und durch verschiedene Operationen manipulieren kann (verschieben, drehen, skalieren, löschen, usw.) Der Entwurf arbeitet polygonbasiert.

Im Beispiel der DRAM-Zelle werden zuerst die Drain-, Gate- und Source-Gebiete des Transistors durch Wahl der Diffusiongebiete bestimmt. Zusätzlich wird eine Diffusionsfläche für den Speicherkondensator der DRAM-Zelle benötigt. Das Gate wird in Polysilizium realisiert (Bit-Leitung). Mit Metall-Flächen und Kontakten zwischen den Layern wird die Zelle verdrahtet. Es muss beim Erstellen auf Symmetrie geachtet und dafür gesorgt werden, dass mehrere Zellen leicht zu matrixförmig aufgebauten Speichern zusammengeschaltet werden können. Dazu sind die Wort- und Bitleitungen an den Kanten herauszuführen.

Nachdem die Zelle gelayoutet wurde, können im Layout-Editor noch Prüfungen stattfinden. Zum einen muss die Schaltung gegen die Netzliste geprüft werden. Zum anderen müssen die Entwurfsregeln überprüft, d.h. der Design-Rule-Check(DRC) durchgeführt werden. Dies geschieht in der Regel interaktiv, indem nach Aufruf des DRCs die Fehler als blinkende Elemente im Layout grafisch dargestellt werden und so leicht beseitigt werden können.

Durch vertikale und horizontale Spiegelung entsteht aus einer DRAM-Zelle ein Feld von Speicherzellen.

Zellerzeugung: Kopplung Schematic/Layout

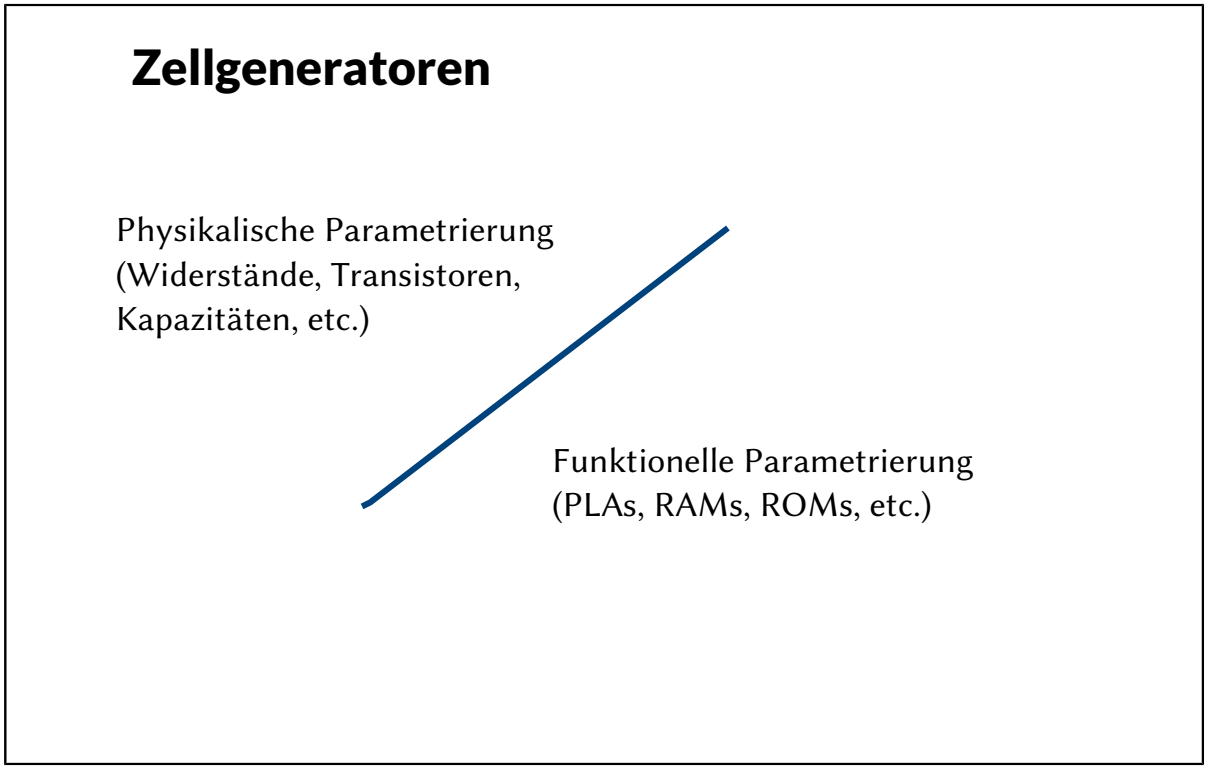


Eine bessere Unterstützung des Designers kann durch eine Kopplung des Schematics mit dem Layout erreicht werden. Für den Standardzellenentwurf, aber auch für den Entwurf von analogen Zellen wird in der Regel eine solche Kopplung verwendet. Im Bild ist zunächst nur der Schaltplan vollständig vorhanden. Es wird eine neue Layoutfläche erstellt (rechts im Bild). Auf diese werden die Bauelemente des Schaltplans automatisch als Layout platziert. Die Elemente können aus einer Bibliothek stammen oder als parametrierte Zellen direkt aus den Spezifikationsgrößen des Bauelements erstellt worden sein (siehe physikalische Parametrierung). Die Bauelemente sind durch so genannte Flylines (Gummibänder) verbunden, die die noch nicht vorhandenen elektrischen Verbindungen darstellen. Der Layouter kann nun den Platzierungsvorschlag interaktiv bearbeiten und insbesondere die elektrischen Verbindungen entwurfsregelkorrekt verlegen.

Auch für diese Aufgabe wird Unterstützung vom Layoutwerkzeug zur Verfügung gestellt. Es können Leitungen vorgegebener Breite automatisch als rechtwinklige Pfade und zwischen den Ebenen durch automatisches Einfügen von Vias gezogen werden. Weiterhin können automatische Router (so genannte Point-to-Point-Router) für die Verlegung einzelner oder auch mehrerer Leitbahnen gleichzeitig eingesetzt werden.

Durch die Kopplung von Schematic und Layout wird neben der Vereinfachung des Entwurfsprozesses eine Korrektheit des Layouts sichergestellt, da weder Bauelemente noch Leitungen übersehen werden können.

Zellerzeugung: Zellgeneratoren



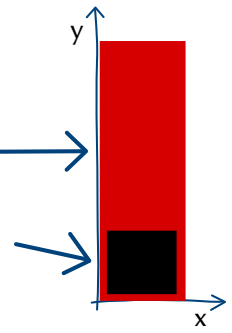
Die manuelle Erzeugung, auch von kleinen Layouts oder nur von Bauelementen aus einzelnen Polygonen in den unterschiedlichen Layern, ist fehleranfällig, mühsam und muss ständig wiederholt werden. Aus diesem Grunde sind Zellgeneratoren entwickelt worden, die für ein gegebenes Bauelement oder für kleine (meist digitale) Zellen das Layout automatisch erstellen. Zwei solcher Generatoren sind in den folgenden beiden Abschnitten beschrieben.

Zellerzeugung: Physikalische Parametrierung

Physikalische Parametrierung

```
procedure (resistor(Rnom)
  Lnom = Rnom/10E9
  LLx = 0
  LLy = 0
  URx = 10
  URy = Lnom

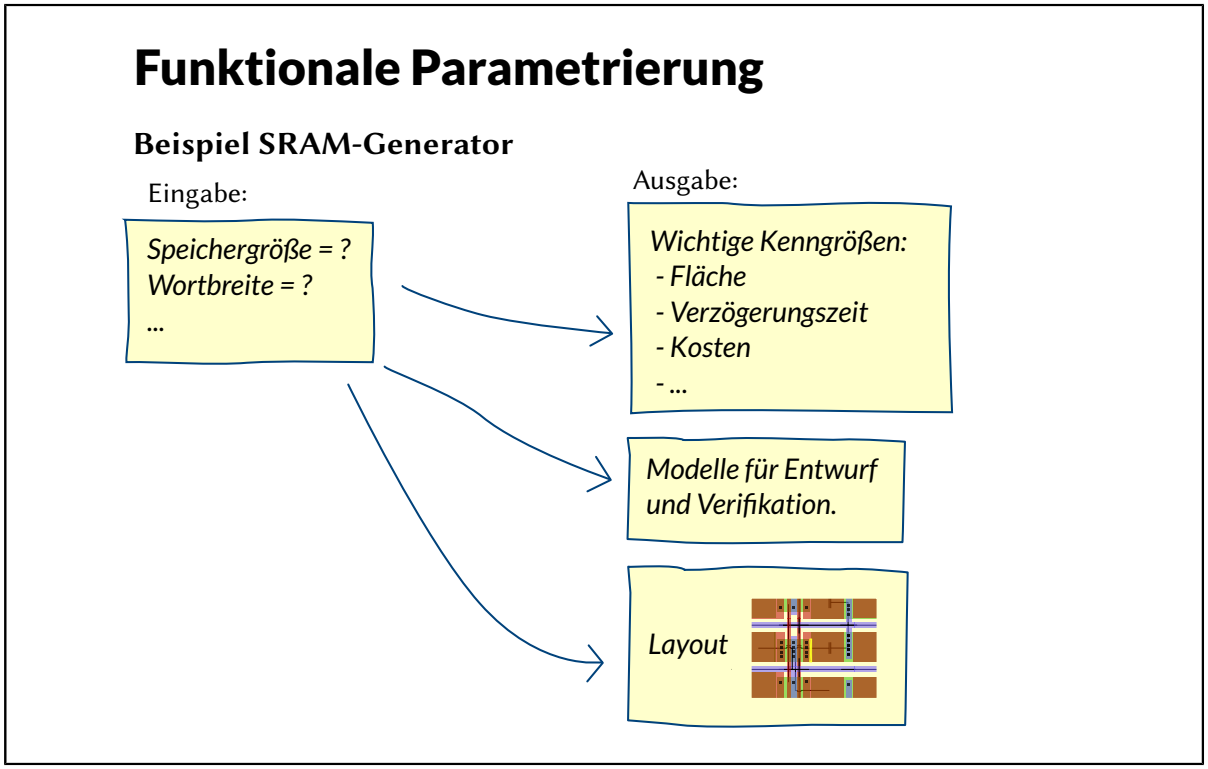
  dBCreateRect(cuID List("Poly" "drawing")
                        List(LLx:LLy URx:URy)
  dBCreateRect(cuID List("VIA" "drawing")
                        List(LLx+1:LLy+1 LLx+9:LLy+9)
...
)
```



Die einfachste Form der Parametrierung von Zellen erzeugt Layouts für die Basiselemente einer Schaltung also für Transistoren, Widerstände und Kondensatoren. Diese Zellen sind häufig in geometrischen Größen parametriert, wie in der Breite und der Länge des Kanals des MOS-Transistors. Elektrische Parameter können unter Kenntnis der Technologie verhältnismäßigleicht in geometrische umgerechnet werden. Diese geometrischen Größen sind direkt per Programm in Polygone umzusetzen. Ein Beispiel für den Code zur Erzeugung eines Widerstands ist im Bild gezeigt.

Das entstehende Layout ist im Bild rechts dargestellt. Neben den Layoutstrukturen für die elektrische Funktion können auch Kontakte, Schutzstrukturen wie Guardringe und Wannen generiert werden. Außerdem sind auch gefaltete und damit platzsparende Transistor-Layouts möglich, die bei großem W/L-Verhältnis (Kanalweite zu -länge) durch mäanderförmiges Verlegen des Gates erzeugt werden.

Zellerzeugung: Funktionale Parametrierung



Ein Höchstmaß an Flexibilität wird durch eine vollautomatische Generierung der Zellen erreicht. Der Benutzer spezifiziert dabei das Verhalten der Zelle mit Hilfe funktioneller Beschreibungen; hier genügt also nicht mehr eine einzelne Zahl. Ein Beispiel ist die Angabe einer booleschen Gleichung zur Definition eines PLAs. Typische Vertreter funktionell parametrierbarer Zellen sind PLA, RAM, ROM, Finite State Machine (FSM) bis hin zum mikroprogrammierbaren Mikroprozessor. Solche Zellgeneratoren verwenden vorwiegend, aber nicht ausschließlich, die Methode des Aneinandersetzens (butting) von Zellelementen. Oft sind dabei die Zellelemente selbst nicht mehr starr, sondern prozedural beschrieben. Ein wichtiger Aspekt des Generatorkonzepts besteht darin, dass nicht nur das Zelllayout, sondern gleichzeitig ein entsprechendes Simulationsmodell sowie die erforderlichen Testmuster generiert werden können. Ein grundsätzliches und noch nicht endgültig gelöstes Problem bei der Zellgenerierung ist die Sicherung der Gewährleistung von Zelleigenschaften: Feste Zellen, aber auch verschiedene Varianten von einfach parametrierbaren Zellen können probeweise gefertigt und vermessen werden. Dadurch wird sichergestellt, dass die durch elektrische Simulation ermittelten Zelleigenschaften auch auf einem realen Chip eingehalten werden. Bei der Vielfalt der Varianten frei generierbarer Zellen ist dies jedoch nicht erschöpfend möglich. Statistische Aussagen werden herangezogen.

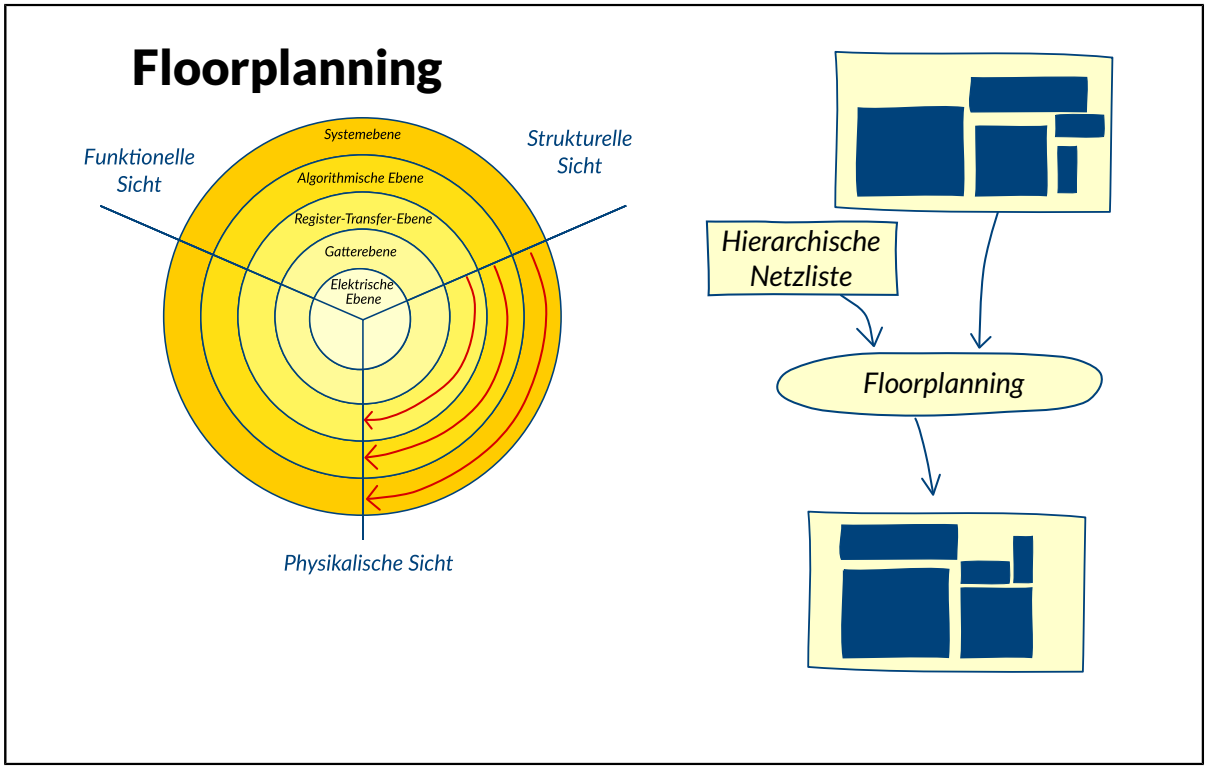
Electronic Design Automation (EDA)

Floorplanning

Floorplanning
Floorplanning als Optimierungsproblem
Abschätzung der Optimierungsziele
Slicing Floorplan
Schnittbaum, umgekehrte polnische Notation(UPN)
Algorithmen (1): Simulated Annealing
...Pseudo-Code
...Akzeptanz
...Überwinden lokaler Minima
...Anwendung auf das Floorplanning-Problem
Algorithmen (2): Graphen-Dualisierung
Vom Graph zum Floorplan
Algorithmen (3): Lineare Optimierung
Lineare Optimierung (2)
Lineare Optimierung (3)

Date of generation: 8.7.2025

Floorplanning: Floorplanning



Floorplanning ist ein verallgemeinertes Platzierungsproblem. Ausgehend von einer hierarchischen Netzliste und einer Bibliothek von möglichen geometrischen Ausprägungen der einzelnen Zellen wird eine Grobplanung des Chip-Layouts angefertigt. Dabei werden sowohl die Fläche des Chips festgelegt als auch die Positionen der Makro-, Block- oder Modulzellen auf dem Chip bestimmt.

Die Schwierigkeit beim Floorplanning resultiert aus dem zusätzlichen Freiheitsgrad, der auf der flexiblen Form der Zellen beruht. Daher benötigt das Floorplanning zusätzlich Bibliotheksinformationen über die möglichen geometrischen Realisierungen der einzelnen Zellen. Diese können sehr vage sein und lediglich Angaben über die Gesamtfläche einer Zelle enthalten, Grenzen für das Seitenverhältnis beinhalten oder aber eine oder mehrere exakt vorgegebene Alternativen für die Form und Größe der Zelle vorgeben.

Grundsätzlich kann die Form einer Zelle ganz beliebig sein. Auf Grund der Komplexität des Problems werden jedoch von den meisten Algorithmen nur rechteckige Zellen zugelassen, teilweise gibt es Erweiterungen für Zellen mit L- oder T-Form oder sogar beliebigen rechtwinkligen Formen. Im folgenden wird jedoch, wie allgemein üblich, von rechteckigen Formen der Zellen ausgegangen.

Floorplanning: Floorplanning als Optimierungsproblem

Floorplanning als Optimierungsproblem

Optimierungskriterien:

- Fläche
- Gesamtverdrahtungslänge
- Länge des kritischen Pfads
- Temperaturverteilung
- Power

Randbedingungen:

- Seitenverhältnis des Chips
- ...

Floorplanning ist ein Optimierungsproblem. Ziel ist es, die Positionen und Formen der Zellen derart zu bestimmen, dass die Chipfläche und die Verdrahtungslänge der globalen Netze, welche die Zellen miteinander verbinden, minimiert werden, um geringe Fertigungskosten sowie hohe Performance zu bewirken. Dazu sollten die Zellen so angeordnet werden, dass möglichst wenig nutzbare Fläche verloren geht und stark verbundene Zellen möglichst dicht bei einander liegen.

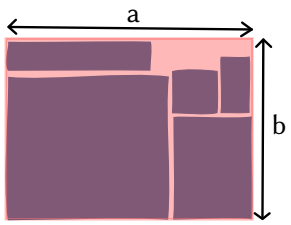
Für einen gegebenen Floorplan ist der Verschnitt und somit die Chipfläche eindeutig bestimmbar, während die Verdrahtungslänge nur abgeschätzt werden kann, da die echte Verdrahtung erst später vorgenommen wird. Dabei kann entweder die Abschätzung der Gesamtlänge aller Netze oder die Länge des längsten Netzes berücksichtigt werden. Daneben können weitere Optimierungskriterien und Randbedingungen auftreten, wie z.B. die Temperaturverteilung des Chips oder ein gewünschtes Seitenverhältnis.

Floorplanning: Abschätzung der Optimierungsziele

Abschätzung der Optimierungsziele

Fläche:

Kleinstes umschließendes Rechteck: $F=a*b$



Gesamtverdrahtungslänge:

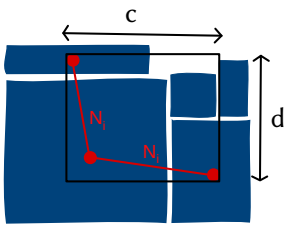
Verdrahtungslänge eines Netzes N_i:

Halber Umfang des umschließendes Rechtecks:

$L_i=c+d$

Gesamtverdrahtungslänge:

$L_{ges} = \Sigma_i L_i$



Zur Bewertung eines Floorplans werden insbesondere die Chipfläche und die Gesamtverdrahtungslänge betrachtet. Da im Allgemeinen nur rechteckige Chips gefertigt werden können, wird die Chipfläche definiert über ein Rechteck, das alle Zellen beinhaltet, das heißt über die Fläche des umschließenden Rechtecks.

Für die Verdrahtungslänge werden beim Floorplanning die globalen Netze betrachtet, welche die Zellen miteinander verbinden. Da es sich beim Floorplanning um ein sehr frühes Entwurfsstadium handelt, in dem nur wenig Informationen über das endgültige Layout vorhanden sind, muss eine grobe Abschätzung der Verdrahtungslängen genügen. Dafür eignet sich der halbe Umfang der Bounding-Box. Die Bounding-Box stellt das umschließende Rechteck aller Anschlüsse eines Netzes dar. Die Netzansclusspunkte sind noch nicht bekannt. Deshalb werden die Mittelpuntke der Zellen als Netzanschlusspunkte gewählt.

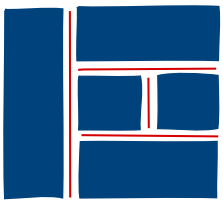
Die Gesamtverdrahtungslänge als zu minimierendes Optimierungsziel ergibt sich aus der Summe der halben Bounding-Boxen aller Netze. Das sind in der Regel sehr viele Netze.

Floorplanning: Slicing Floorplan

Slicing Floorplan

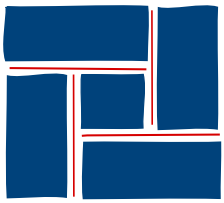
Slicing

Rekursive Zweiteilung der Fläche ist möglich.



Non-Slicing

Rekursive Zweiteilung der Fläche ist nicht möglich.

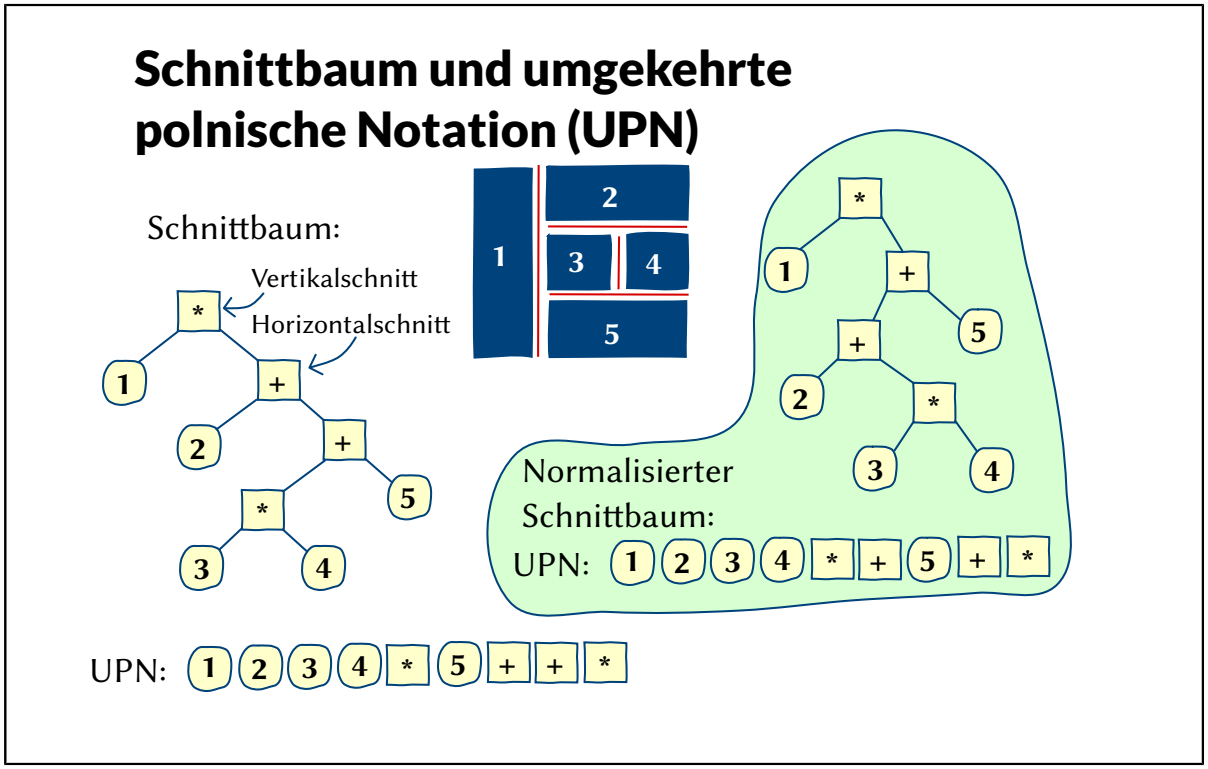


Aufgrund der verschiedenen Datenstrukturen, mit denen Floorplans repräsentiert und gespeichert werden können, ergibt sich eine Unterteilung in zwei verschiedene Klassen von Floorplans: Slicing Floorplans und Non-Slicing Floorplans.

Floorplans, die sich durch eine rekursive Zweiteilung der Layoutfläche erhalten lassen, werden Slicing Floorplans genannt. Alle anderen, allgemeinen Floorplans ohne Einschränkung gehören zu den Non-Slicing Floorplans. Ein Beispiel für einen Non-Slicing Floorplan ist das so genannte Rad (s. Abb.).

Beschränkt man sich durch die Wahl der Datenstruktur auf Slicing Floorplans, so wird auch der Raum der möglichen Lösungen eingeschränkt und damit kann das globale Optimum eventuell nicht erreicht werden. Andererseits können durch diese Einschränkung die Daten sehr effizient gespeichert und der beschränkte Lösungsraum wesentlich effizienter durchsucht werden. Daher finden beide Arten von Floorplans Verwendung.

Floorplanning: Schnittbaum, umgekehrte polnische Notation(UPN)



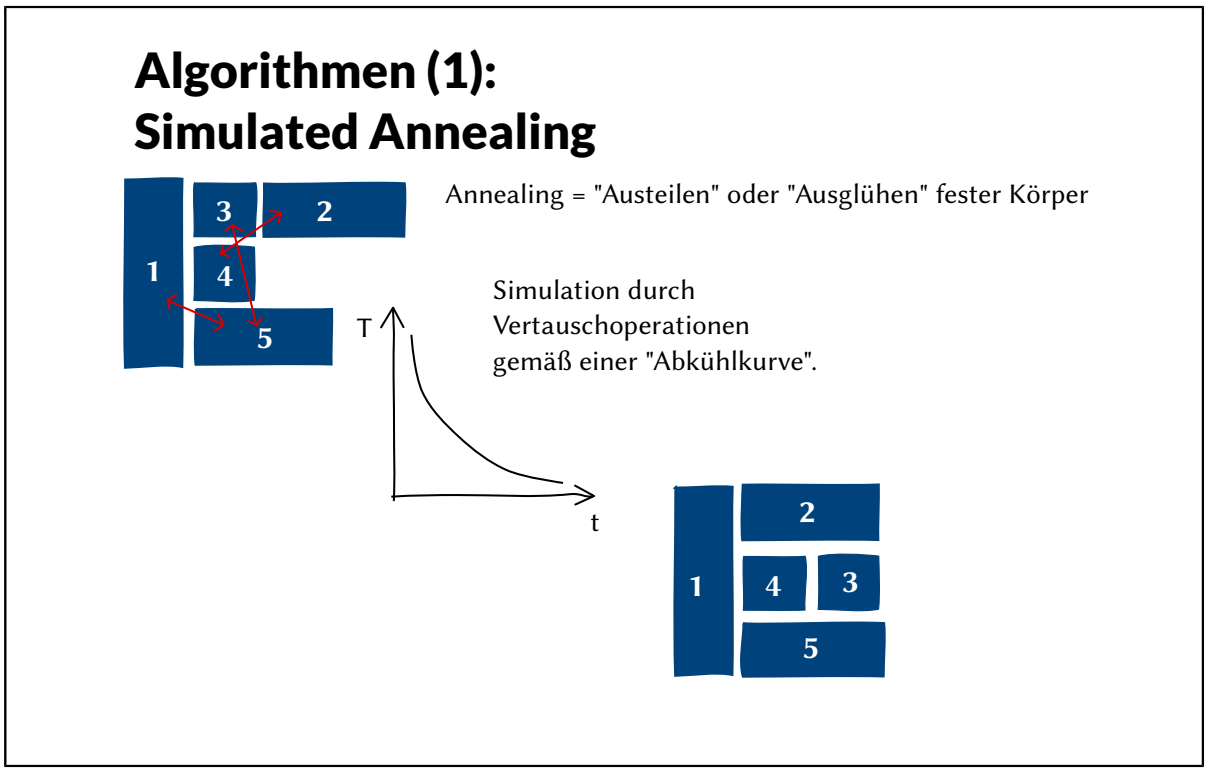
Die am häufigsten verwendete Datenstruktur für das Floorplanning sind so genannte Schnittbäume, welche durch die umgekehrte polnische Notation (UPN) beschrieben werden können. Schnittbäume sind geordnete, vollständige, binäre Bäume. In einem Schnittbaum werden die einzelnen Module als Blätter des Baums dargestellt, während die Verzweigungsknoten Schnittlinien repräsentieren. Dabei enthält jeder Knoten ein Label für die Schnittrichtung. Diese gibt an, ob die beiden Teilbäume dieses Knotens durch einen horizontalen oder einen vertikalen Schnitt miteinander verbunden sind, d.h. ob deren Teilflächen nebeneinander oder übereinander liegen. Durch Schnittbäume können lediglich Slicing Floorplans dargestellt werden.

Durchläuft man einen Schnittbaum mit einem Post-Order-Durchlauf, so erhält man den zu dem Schnittbaum äquivalenten UPN-Ausdruck. Die Zellen sind die Operanden, die durch ihre Nummer gekennzeichnet sind. "+" bzw. "*" sind Operatoren für einen horizontalen bzw. vertikalen Schnitt.

Es existieren i.A. mehrere verschiedene Schnittbäume bzw. UPN-Ausdrücke, welche den gleichen Floorplan beschreiben. Um zu einer normalisierten Form zu kommen, wird eine zusätzliche Bedingung an die Datenstruktur geknüpft:

In einem normalisierten Schnittbaum muss die Schnittrichtung des rechten Teilbaums immer verschieden sein von der seines Vorgängerknotens. Für die normalisierte UPN folgt, dass keine zwei aufeinander folgenden Operatoren gleich sein dürfen.

Floorplanning: Algorithmen (1): Simulated Annealing



Simulated Annealing ist ein Verfahren, das das Ausglühen (Annealing) von festen Körpern nachbildet. Dabei werden Spannungen in festen Körpern entfernt. Das Ergebnis ist ein Zustand minimaler Energie.

Beim Ausglühen, wie auch beim Algorithmus; sind verschiedene Aspekte zu berücksichtigen, um ein möglichst gutes Ergebnis zu erzielen. Zum einen muss die Starttemperatur, also die Temperatur, bei der das Ausglühen beginnen soll, hoch sein, damit die Moleküle in starke Schwingungen versetzt werden. Entsprechend einer zu definierenden Abkühlkurve (Annealing Schedule) wird die Temperatur gesenkt und die Moleküle ordnen sich in einem spannungsfreien Verbund an.

Floorplanning: ...Pseudo-Code

Simulated Annealing: Pseudo-Code

```
T = Anfangstemperatur;
Initiallösung;
BEWERTE (Initiallösung);
while(! Gefrierpunkt erreicht)
{
   while(! Gleichgewicht für T erreicht)
   {
      VERÄNDERE (aktuelle Lösung);
      BEWERTE (neue Lösung);
      if(AKZEPTANZ)
      {
          Übernehme neue Lösung;
      }
   }
   Senke Temperatur T;
}
```

Simulated Annealing wird für verschiedene Optimierungsprobleme eingesetzt. Dem Körper entspricht im Algorithmus eine mögliche Lösung, die Schwingung der Moleküle wird durch Störungen bzw. VERÄNDERUNGEN der aktuellen Lösung zu einer neuen realisiert.

Zu jeder Temperatur werden so viele VERÄNDERUNGEN durchgeführt, bis ein Gleichgewichtszustand für diese Temperatur erreicht ist. Das kann z.B. nach einer festen Anzahl von Veränderungsschritten der Fall sein, oder wenn sich die Qualität der Lösung über eine bestimmte Anzahl von Schritten nicht mehr verändert hat. Als Kostenfunktion zur BEWERTUNG wird im Allgemeinen die gewichtete Summe von Fläche und Verdrahtungslänge betrachtet.

Floorplanning: ...Akzeptanz

Simulated Annealing: Akzeptanz

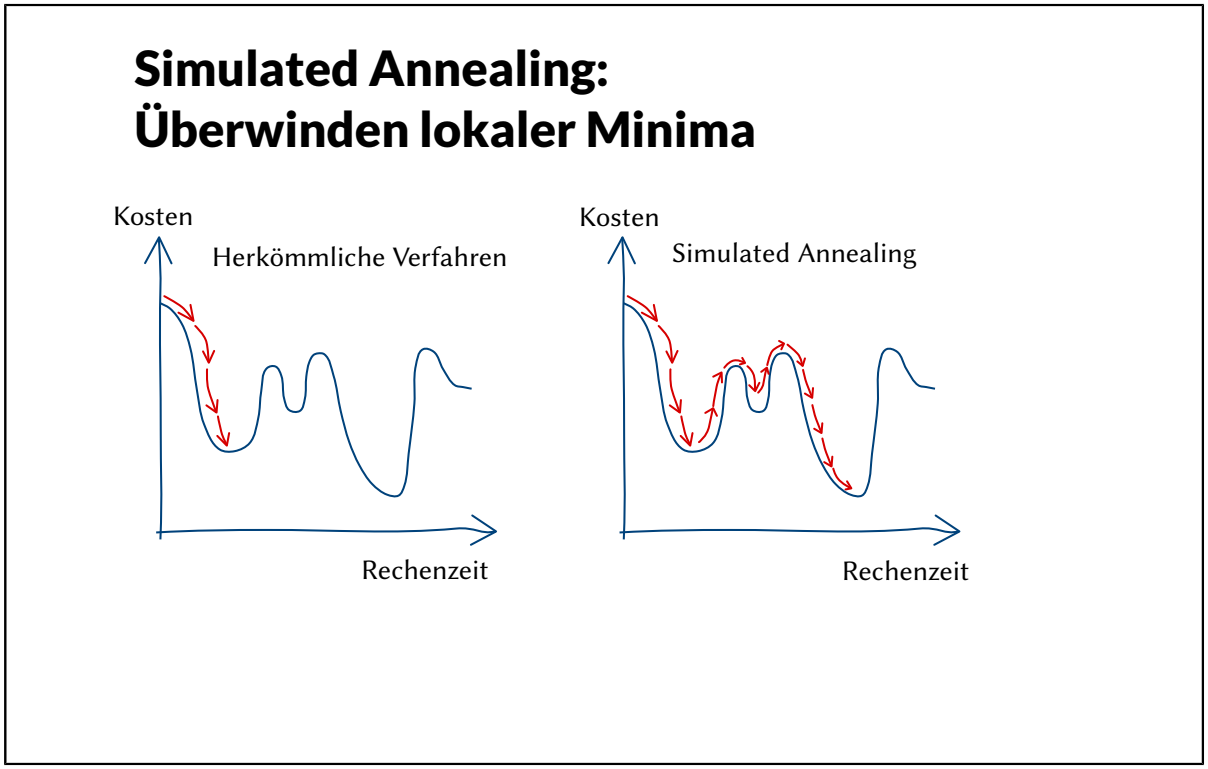
```
ΔK = NeueKosten - AlteKosten;

if ((ΔK < 0)  or  (RANDOM (0,1) < EXP(-ΔK/kT)))
  {
    Akzeptanz der neuen Lösung
  }
```

Entsprechend der Temperatur schwingen die Moleküle zunächst mehr, später weniger stark. Dementsprechend werden je nach Temperatur mehr oder weniger starke Verschlechterungen der aktuellen Lösung akzeptiert. Veränderungen, die zu einer Verbesserung führen, werden immer akzeptiert. Neben der Temperatur und der Differenz der Kosten der Lösungen hängt die Akzeptanz einer Veränderung auch noch vom Zufall ab. Dieser sorgt dafür, dass es sich beim Simulated Annealing um einen stochastischen, nicht-deterministischen Algorithmus handelt. Je geringer die Temperatur ist, desto geringer ist auch die Wahrscheinlichkeit, dass eine Verschlechterung akzeptiert wird. Je stärker sich die Kosten erhöhen, desto unwahrscheinlicher wird die Akzeptanz dieser Veränderung. Meistens wird dieser Zusammenhang durch die Boltzmann-Wahrscheinlichkeitsfunktion exp((NeueKosten-AlteKosten) / (k*Temperatur)) modelliert, wobei k die Boltzmann-Konstante ist.

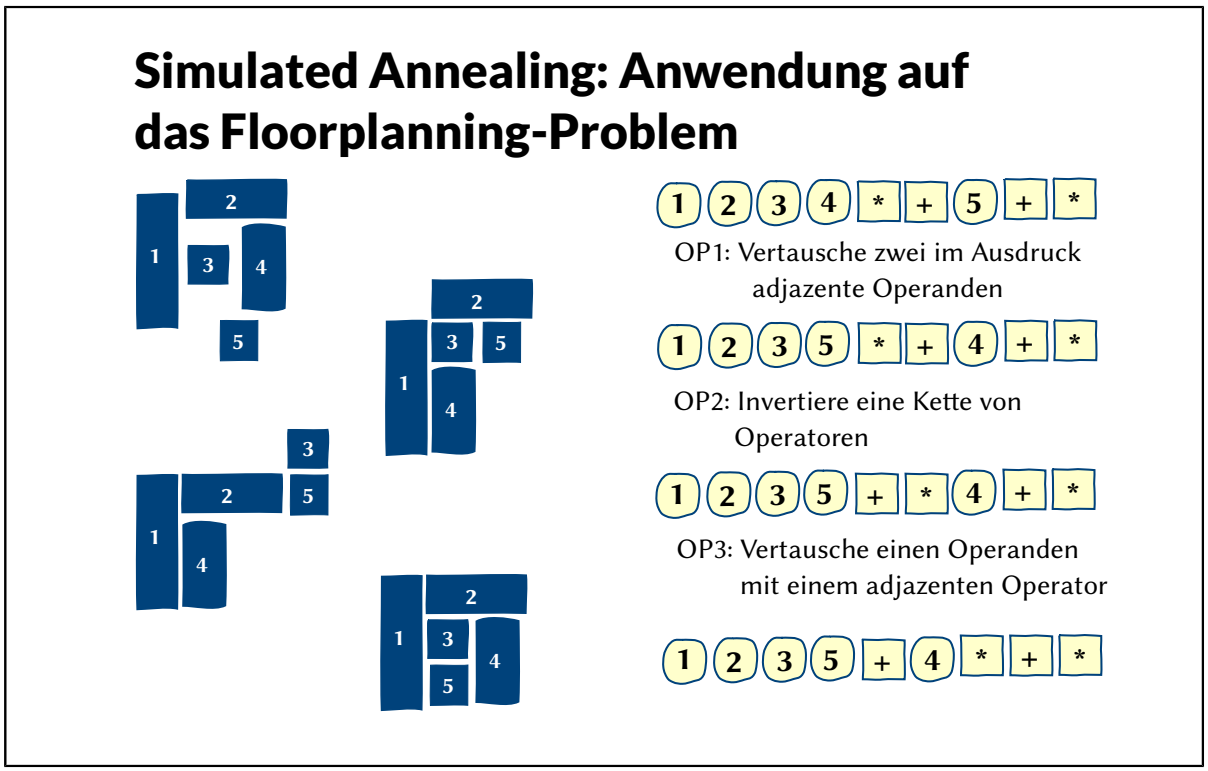
Im Allgemeninen wird die letzte bekannte beste Lösung noch einmal seperat abgespeichert.

Floorplanning: ...Überwinden lokaler Minima



Das Verfahren wird in verschiedenen EDA-Bereichen angewandt. Es ist einfach zu implementieren, liefert gute Ergebnisse, ist aber sehr rechenzeitintensiv. Das Verfahren hat die Eigenschaft, dass lokale Minima überwunden werden können. Der Grund hierfür liegt in der Tatsache, dass auch Platzierungsergebnisse, die höhere Kosten als die vorhergehende Platzierung erzeugen, akzeptiert werden können.

Floorplanning: ...Anwendung auf das Floorplanning-Problem



Beim Floorplanning ist Simulated Annealing das derzeit am meisten verwendete Verfahren. Ein bekannter Floorplanning-Algorithmus basierend auf Simulated Annealing arbeitet dabei mit der umgekehrten polnischen Notation (UPN). Die Startlösung wird im Allgemeinen zufällig generiert. Entscheidend für den Algorithmus sind der Satz der erlaubten Veränderungen und die Kostenfunktion.

Veränderungen:

Ausgehend von einem normalisierten UPN-Ausdruck wird eine neue Lösung, die wiederum eine normalisierte UPN ist, durch eine der folgenden drei Operationen bestimmt:

OP1: Vertausche zwei im Ausdruck adjazente Operanden

OP2: Invertiere eine Kette von Operatoren (* wird zu + und umgekehrt)

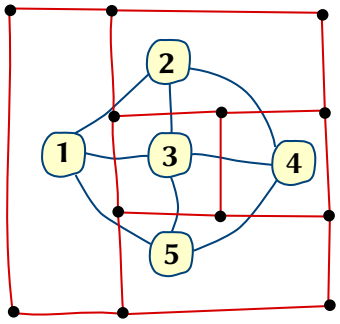
OP3: Vertausche einen Operanden mit einem adjazenten Operator, falls das Ergebnis wieder eine normalisierter UPN-Ausdruck ist.

Dieser Satz von Veränderungen ist vollständig, d.h. ausgehend von jedem beliebigen normalisierten UPN-Ausdruck kann jeder beliebige andere über eine Folge dieser Operationen erreicht werden.

Floorplanning: Algorithmen (2): Graphen-Dualisierung

Algorithmen (2): Graph-Dualisierung

Graph der Schaltung



Dualer Rechteckgraph

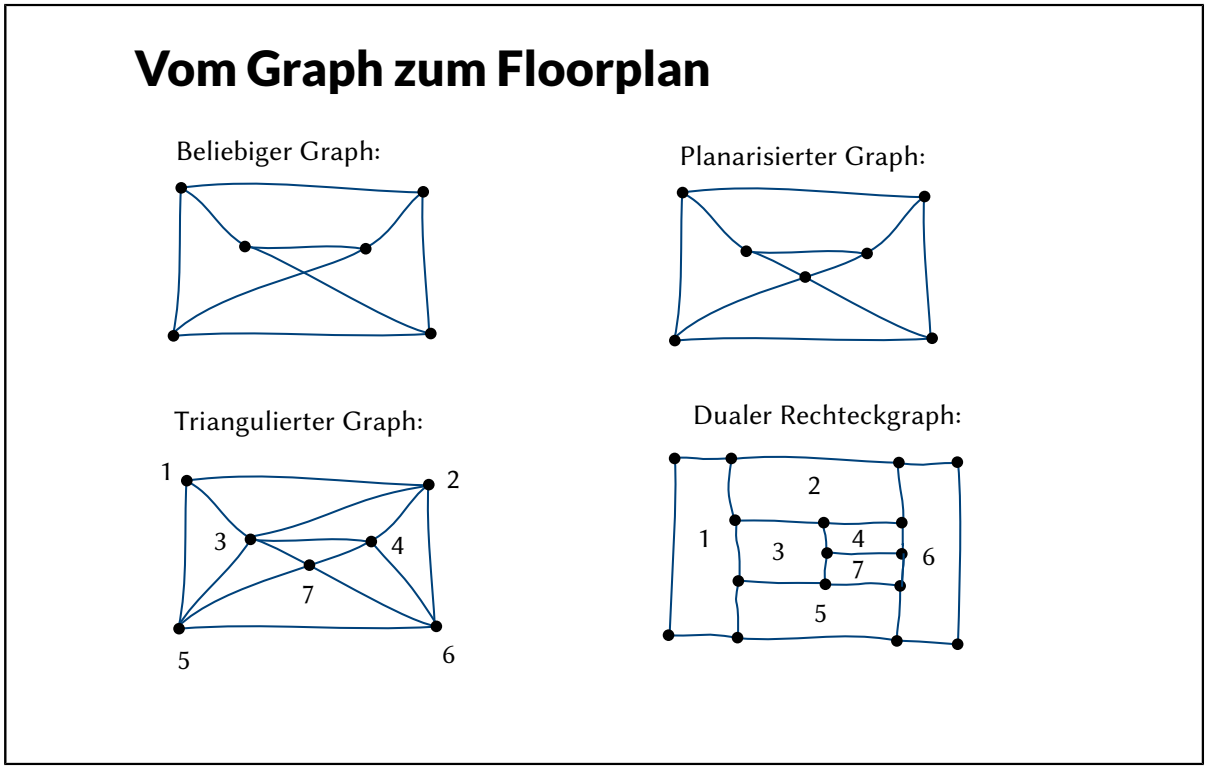
Bei der Graphen-Dualisierung wird die Schaltung zunächst durch einen Graphen G dargestellt, dessen Knoten die Zellen der Schaltung und die Kanten deren Verbindungen untereinander darstellen. Ein Floorplan kann daraus ermittelt werden, indem der zu G duale Rechteckgraph (DRG) aufgestellt wird. Dieser besteht aus nicht-überlappenden Rechtecken, welche die folgenden Bedingungen erfüllen:

1. Jeder Knoten v_i entspricht einem Rechteck R_i.

2. Für jede Kante (v_i , v_j) sind die Rechtecke R_i und R_j adjazent.

Die Graph-Dualisierung führt zur Maximierung von Adjazenzen von Blöcken, die stark (z.B. durch mehtfache Verbindungen) miteinander verbunden sind.

Floorplanning: Vom Graph zum Floorplan



Nicht jeder Graph besitzt einen dualen Rechteckgraphen. Allerdings besitzen planare, triangulierte Graphen (PTG) einen dualen Rechteckgraphen. Um aus einem beliebigen Graphen einen PTG zu machen, sind die folgenden Schritte notwendig:

1. Planarisierung: An Kreuzungspunkten werden Knoten hinzugefügt oder es werden überlappende Kanten entfernt.

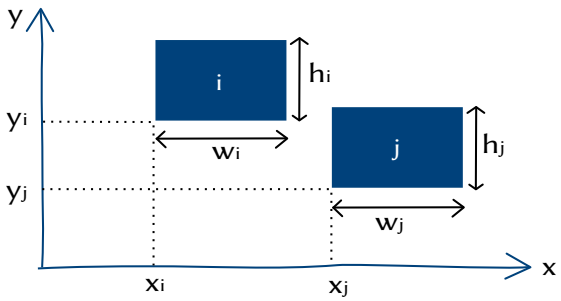
2. Triangulierung: In Gebiete, die keine Dreiecke sind, werden zusätzliche Kanten eingefügt.

Zusätzliche Knoten entsprechen im Floorplan Platzhalter-Rechtecken, zusätzliche Kanten bedingen Adjazenzen von Zellen, die keine Verbindungen haben, durch das Entfernen von Kanten werden Verbindungen von Zellen ignoriert. Daher sollen bei diesen Schritten so wenig Veränderungen wie nötig durchgeführt werden, um eine gute Ergebnisqualität zu erreichen.

Für PTGs existieren Algorithmen, um die Topologie, die in dem planaren Graphen enthalten ist, in einen dualen Rechteckgraphen umzusetzen, der allerdings nicht eindeutig sein muss.

Floorplanning: Algorithmen (3): Lineare Optimierung

Algorithmen (3): Lineare Optimierung (1)



Formulierung des Problems durch ein System von Ungleichungen:

Horizontale Beziehungen:

$x_i + w_i \leq x_j$ (i ist links von j)

$x_j + w_j \leq x_i$ (i ist rechts von j)

Vertikale Beziehungen:

$y_i + h_i \leq y_j$ (i ist unterhalb von j)

$y_j + h_j \leq y_i$ (i ist oberhalb von j)

Lineare Optimierung ist ein analytisches Verfahren, mit dem sich das Floorplanning-Problem zurückführen lässt auf das Lösen eines linearen Ungleichungssystems. Für diese Aufgabe existieren aus der Mathematik zahlreiche Lösungsverfahren. Hauptaufgabe ist also die Formulierung des Floorplanning-Problems durch einen Satz von linearen Ungleichungen für die Randbedingungen und eine lineare Kostenfunktion.

Die Kostenfunktion (beim Floorplanning insbesondere die Größe der Chipfläche) ist eine nichtlineare Funktion (A=X*Y), kann aber leicht linearisiert werden, indem eine Variable (z.B. die Chipbreite) als fest angenommen wird und die andere minimiert wird. Im folgenden werden die wichtigsten Randbedingungen durch lineare Ungleichungen beschrieben.

Randbedingung Überlappungsfreiheit:

In einem gültigen Floorplan dürfen sich keine zwei Zellen überlappen. Für zwei Zellen i und j, deren linke untere Ecke jeweils an der Koordinate (xi, yi) bzw. (xj, yj) liegt und die die Höhe hi (bzw. hj) und Breite wi (bzw. wj) aufweisen, ist diese Bedingung erfüllt, wenn mindestens eine der abgebildeten Gleichungen gültig ist.

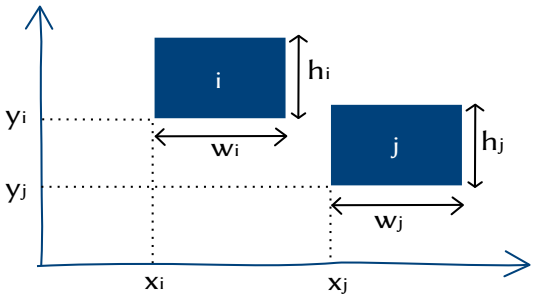
Floorplanning: Lineare Optimierung (2)

Lineare Optimierung (2)

Es muss immer nur eine Gleichung erfüllt sein. Daher: Einführung von Hilfsgrößen:

W: Obergrenze für Gesamtbreite
H: Obergrenze für Gesamthöhe

p_{ij}, q_{ij}: aus der Menge {0,1} steuern die Auswahl der gültigen Bedingungen



$$x_i + w_i \le x_j + W * (p_{ij} + q_{ij})$$
$$x_j + w_j \le x_i + W * (1 - p_{ij} + q_{ij})$$

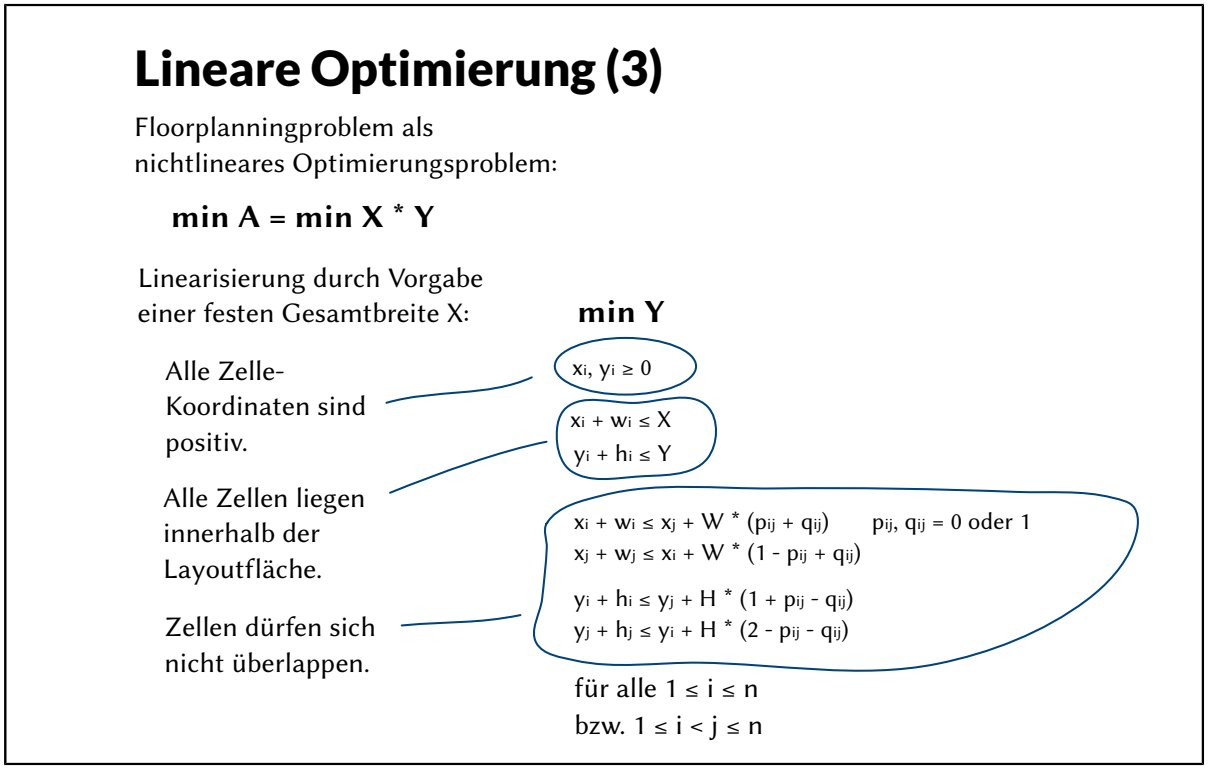
$$y_i + h_i \le y_j + H * (1 + p_{ij} - q_{ij})$$
$$y_j + h_j \le y_i + H * (2 - p_{ij} - q_{ij})$$

Zum Beispiel:
$p_{ij} = 0$ und $q_{ij} = 1$
$y_i + h_i \le y_j$

Da es ausreicht, dass eine dieser Gleichungen erfüllt ist, werden für jedes Paar von Zellen i und j zwei zusätzliche Variablen pij und qij eingeführt, die nur die Werte 0 und 1 annehmen können und die Auswahl der jeweiligen Bedingung darstellen. Außerdem werden zwei zusätzliche Größen W und H eingeführt, welche die oberen Grenzen der Breite und der Höhe der gültigen Gesamtlösung angeben.

Dadurch ergibt sich das dargestellte Ungleichungssystem. Zum Beispiel muss für pij=0 und qij=1 die Ungleichung yi + hi <= yj erfüllt sein, während alle anderen Gleichungen auf Grund des zusätzlichen Terms W bzw. H keine Rolle spielen. Mindestens eine der ursprünglichen Gleichungen ist also bei jeder Wahl von p und q erfüllt.

Floorplanning: Lineare Optimierung (3)



Für einen gültigen Floorplan müssen die folgenden Bedingungen zutreffen:

1. Alle Zelle-Koordinaten sind positiv, d.h. 0 <= x_i und 0 <= y_i

2. Jede Zelle befindet sich vollständig innerhalb des den Floorplan umgebenden Rechtecks mit Breite X und Höhe Y, d.h. $x_i + w_i <= X$ und $y_i + h_i <= Y$.

3. Keine zwei Zellen überlappen sich.

Ziel ist es, die Fläche A=X*Y des Floorplans zu minimieren. Dies ist jedoch eine nichtlineare Bedingung. Eine Möglichkeit ist es, die Breite X festzuhalten und die Höhe Y zu minimieren. Insgesamt folgt dann für den linearen Optimierungsalgorithmus der angegebene Satz von Bedingungen.

Erlaubt man den Zellen eine flexible Form, so sind h_i und w_i ebenfalls unbestimmte Variablen, für die weitere Gleichungen eingeführt werden können. Auch die Abschätzung der Verdrahtungslänge kann durch einen Satz von geeigneten Gleichungen berücksichtigt werden.

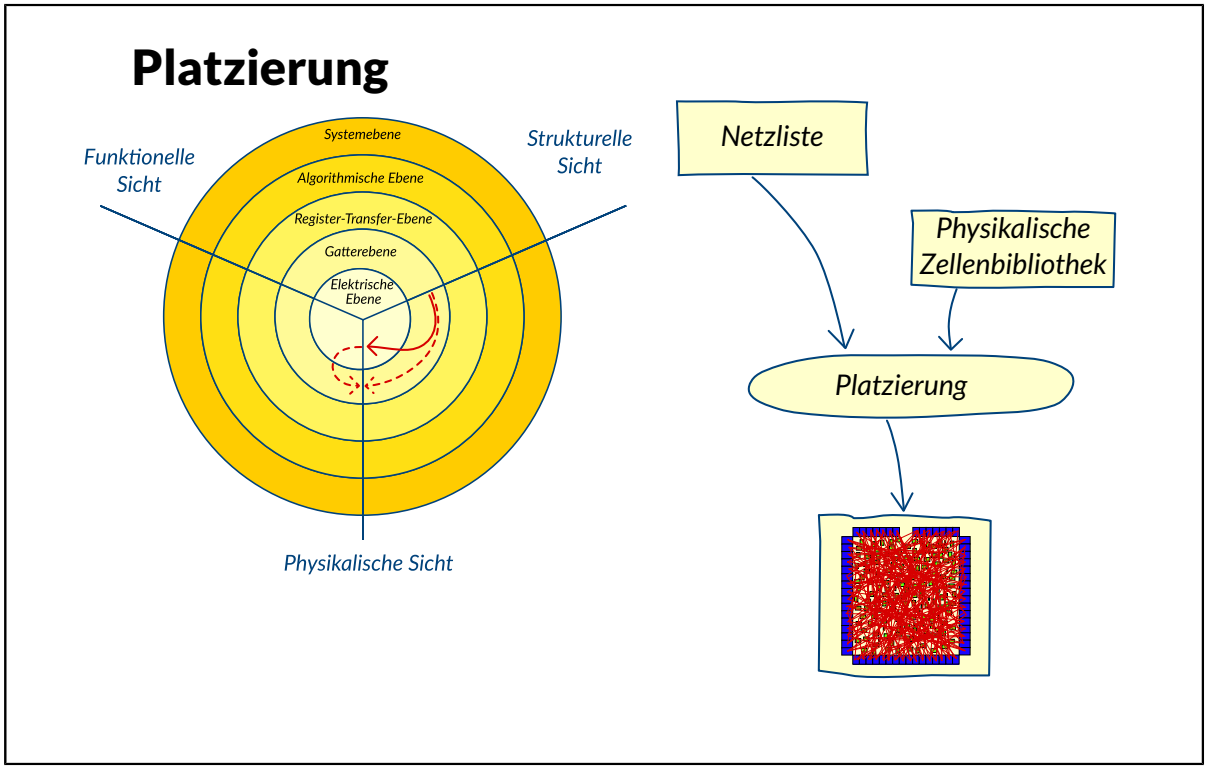
Neben den vorgestellten Verfahren werden für das Floorplanning auch noch genetische Algorithmen, Min-Cut- basierte Algorithmen, Cluster-Verfahren und andere verwendet. Diese werden auch bei der Platzierung eingesetzt.

Electronic Design Automation (EDA)

Platzierung

Platzierung
Platzierung als ..Optimierungsproblem
..Zuordnungsproblem I
..Zuordnungsproblem II
Optimierungskriterien
..Fläche I
..Fläche II
..Timing
..Congestion
..Weitere
Verdrahtungslängenabschätzung
..Bounding Box
..Kette
..Spannbaum
..Steinerbaum
Klassen von Platzierungsverfahren
Platzierungsverfahren
..Clusterverfahren
..Clusterverfahren: Beispiel
..Min-Cut
..Min-Cut: Beispiel
..Quadratische Optimierung
....Feder-Masse-Modell
....Energie-Berechnung
....Kräftegleichgewicht
....Initialplatzierung
....Abstoßende Kräfte
....Dichtefunktion
....Beispiel
....Legalisierung
..Simulated Annealing
..Genetische Algorithmen
....Beispiel
....Operatoren
......Crossover
......Mutation
......Selektion
Bewertung der Verfahren

Platzierung: Platzierung



Während des Syntheseschrittes Platzierung erfolgt der Übergang von der strukturellen Beschreibung in Form einer Netzliste in eine geometrische Anordnung der Bauelemente. Als Eingabedaten dienen die Netzliste und die physikalische (Zell-) Bibliothek. Die Netzliste stellt die Verbindungen (logische Anordnung) der Bauelemente dar und repräsentiert damit den Schaltungsgraphen. Gesucht wird die geometrische Anordnung der Bauelemente und ihrer Verbindungen auf der Layoutfläche. Die Unterschiede zwischen Floorplanning und Platzierung bestehen in der festen geometrischen Form der Zellen und in der erheblich größeren Anzahl der Zellen, also in der Problemgröße. Das Ziel der Platzierung ist eine überlappungsfreie Anordnung der Zellen, die eine automatische Verdrahtung ermöglicht.

Platzierung: Platzierung als ..Optimierungsproblem

Platzierung als Optimierungsproblem

Gegeben: Zellen mit Verbindungen untereinander

Gesucht: Überlappungsfreie Anordnung der Zellen,
so dass automatische Verdrahtung gewährleistet ist

Optimierungsziel: Gesamtverdrahtungslänge

$$L_{Gesamt} = \sum_{i} l_i$$

Feste Plätze auf der Layoutfläche:

-> Kombinatorisches quadratisches Zuordnungsproblem

Die Platzierung stellt ein Optimierungsproblem in der Regel mit Nebenbedingungen dar. Gegeben sei eine Anzahl von Zellen, die auf einer vorgegebenen Fläche platziert werden sollen. Die gefundene Anordnung soll eine automatische Verdrahtung des Systems ermöglichen. Optimierungsziel ist die Gesamtverdrahtungslänge. Damit kann einerseits die Fläche minimiert werden, andererseits die Performance, da kürzere Leitungen kürzere Verzögerungszeiten bedingen. Zusätzlich können Verzögerungszeitvorgaben für einzelne Leitungen in Form von Nebenbedingungen angegeben werden.

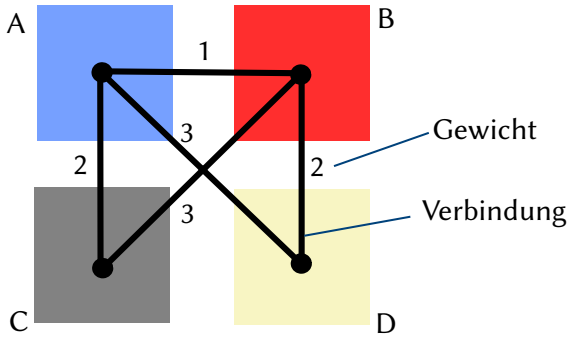
Setzte man vereinfachend voraus, dass für die Platzierung lediglich feste Plätze auf der Layoutfläche zur Verfügung stehen, kann Platzierung als ein kombinatorisches quadratisches Zuordnungsproblem gesehen werden. Darunter versteht man die Anordnung von n Zellen auf n verschiedene Positionen.

Platzierung: ..Zuordnungsproblem I

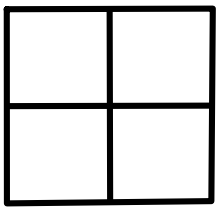
Platzierung als Zuordnungsproblem I

Gegeben: Schaltung aus vier Elementen und Verbindungen zwischen den Elementen

Anordnung:



Layoutfläche:



$$\text{Verbindungskosten } K = \sum_{i=1}^{n} \text{Gewicht } c_i * \text{Distanz } d_i$$

n: Anzahl der Verbindungen

Als Kosten K_{ij} einer Verbindung wird der Kostenbeitrag angesehen, der bei der Zuordnung der Zelle i an die Position j entsteht. Die Summe der Kosten soll minimiert werden. Dies stellt sicher, dass die Verbindungen zwischen den Zellen möglichst kurz sind. Durch Gewichtsfaktoren c_{ij} können die Anzahl der Verbindungen zwischen zwei Zellen sowie ihre Dicke oder Leitfähigkeit eingehen. Aufgrund der Anordnung auf der Layoutfläche entsteht eine räumliche Entfernung d_{ij} von zwei miteinander verbundenen Zellen. Die Kosten einer Verbindung berechnen sich dann aus dem Abstand der beiden Zellen multipliziert mit dem Gewicht, also $k_{ij} = c_{ij} * d_{ij}$.

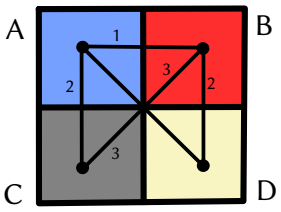
Die Gesamtkosten berechnen sich aus der Summe Kosten der Einzelverbindungen, die sich aus dem Gewicht der Verbindung multipliziert mit dem Abstand zusammensetzen. Es wird über alle Verbindungen summiert.

Insgesamt gibt es n! (hier: 24) Möglichkeiten, die Zellen auf der Layoutfläche anzuordnen.

Um das Minimum zu bestimmen, könnte man alle n! Möglichkeiten untersuchen und für jede Möglichkeit die Kosten berechnen. Bei derzeitigen Problemgrößen ist dies jedoch nicht realistisch. Da jedoch effiziente exakte Algorithmen für dieses Problem nicht verfügbar sind, müssen Heuristiken angewendet werden. Unter einer Heuristik versteht man ein Verfahren, das in einer bestimmten Zeit nicht die optimale, jedoch eine annehmbar gute Lösung liefert.

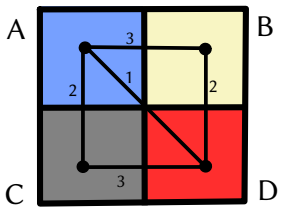
Platzierung: ..Zuordnungsproblem II

Platzierung als Zuordnungsproblem II



$$K = \sum_{i=1}^{n} c_i \cdot d_i$$

$$K = 1\cdot1 + 2\cdot1 + 2\cdot1 + 3\cdot(1+1)+3\cdot(1+1)$$
$$= 17$$

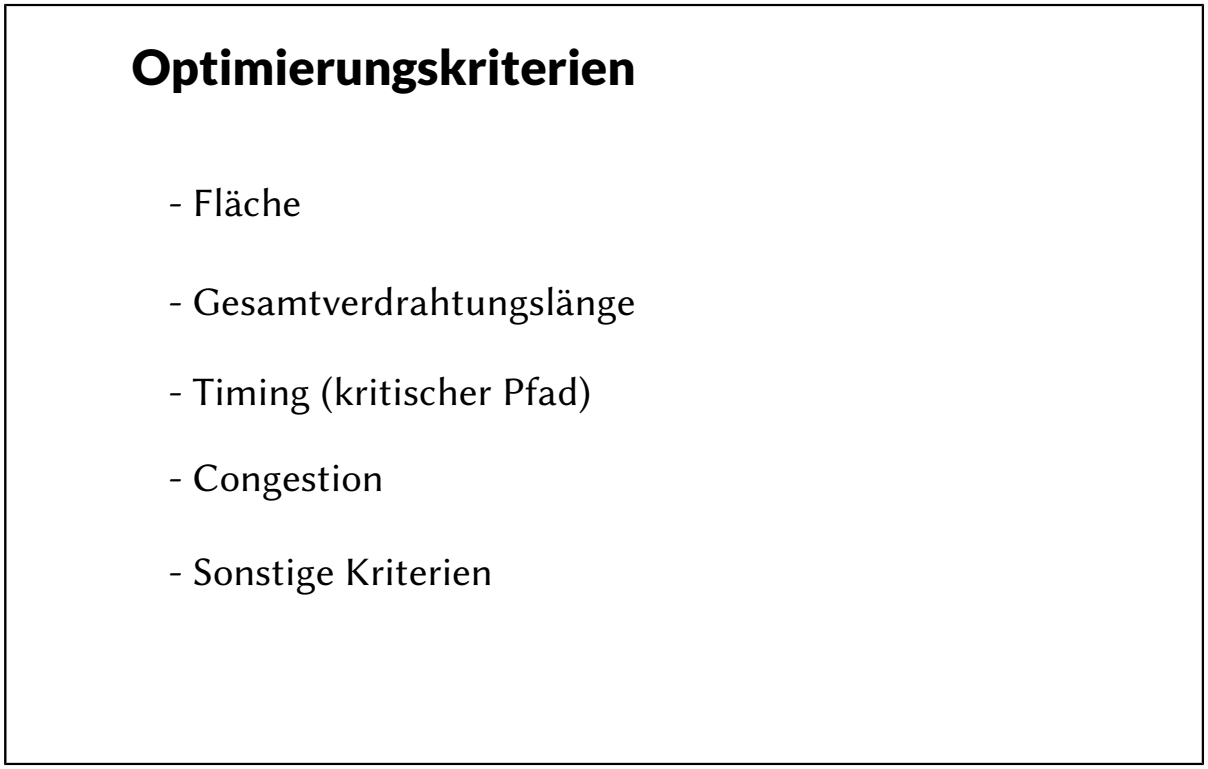


$$K = 3\cdot1 + 2\cdot1 + 2\cdot1 + 3\cdot1 + 1\cdot(1+1)$$
$$= 12$$

Beispielhaft werden hier zwei Anordnungen herausgesucht und auf ihre Kosten hin untersucht. Dabei wird für den Abstand der Manhattan-Abstand verwendet.

Die Kosten werden für die erste Anordnung folgendermaßen berechnet: Die Verbindung zwischen den Zellen A und B weist ein Gewicht von 1 auf. Der Abstand ist ebenfalls 1, so dass die Kosten für diese Verbindung 1 betragen. Anschließend werden die Verbindungen zwischen den Zellen A und C bzw. B und D betrachtet. Beide Verbindungen haben ein Gewicht von 2. Die Abstände sind jeweils 1, so dass die Einzelkosten sich zu 2 ergeben. Die Verbindungen zwischen den Zellen A und D bzw. B und C weisen ein Gewicht von 3 auf, wobei der Manhattan-Abstand 2 beträgt. Die Einzelkosten ergeben sich dann zu jeweils 6. Die Gesamtsumme errechnet sich aus der Summe der Einzelkosten, also zu 17.
Die gleiche Vorgehensweise liefert für die zweite Anordnung einen Wert von 12, also ein wesentlich besseres Ergebnis.

Platzierung: Optimierungskriterien



Im allgemeinen Fall können sehr unterschiedliche Kriterien in die Kostenfunktion einbezogen werden. Jedes Design weist in seiner Spezifikation Eigenschaften auf, die Einflüsse auf die Gewichtung der Optimierungskriterien haben. Soll beispielsweise ein Schaltungsentwurf einen hohen Systemtakt ermöglichen, so ist das Timing von erhöhtem Interesse, während es bei einem nicht so performanten System weniger stark Berücksichtigung finden wird.

Im Folgenden sollen vier Aspekte angesprochen werden:

- Fläche
- Timing
- Congestion
- Sonstige Kriterien

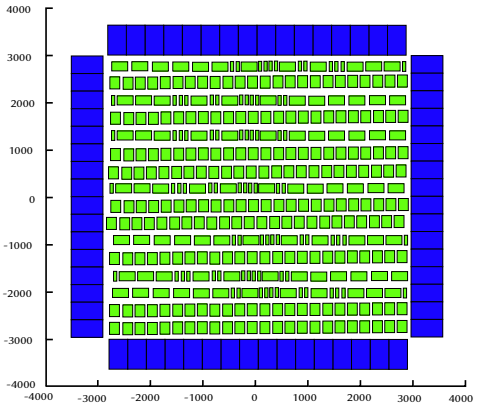
Platzierung: ..Fläche I

Optimierungskriterium: Fläche I

- Layoutfläche = Zellfläche + Verdrahtungsfläche

- Minimierung der Gesamtverdrahtungslänge entspricht Flächenminimierung

Platzierung mit Verdrahtungskanälen:



Ziel ist hierbei die Minimierung der Layoutfläche, die sich aus der Fläche für die Zellen und der Fläche für die Verdrahtung zusammensetzt. Flächenminimierung bedeutet, dass alle Zellen so nah wie möglich beieinander zu platzieren sind und gleichzeitig die Gesamtverdrahtungslänge minimiert wird. Steht die von Zellen belegte Fläche nicht für die Verdrahtung zur Verfügung, ist die Minimierung der Gesamtfläche gleichbedeutend mit der Minimierung der Verdrahtungsfläche, die bei konstanter Leitbahnbreite proportional zur Summe aller Leitbahnlängen ist. Grundsätzlich ist die Länge der Leitbahnen während der Platzierung noch unbekannt, da eine tatsächliche Ausführung der Verdrahtung viel zu zeitaufwändig wäre. Die Verdrahtungslänge muss deshalb abgeschätzt werden. Dies ist auf verschiedene Arten möglich. Zum einen kann der lineare Abstand als Verdrahtungslänge definiert werden, zum anderen der quadratische Abstand. Bei der Wahl der linearen Verdrahtungslänge erhält man im Vergleich zur quadratischen Verdrahtungslänge sehr viele kurze Verbindungsleitungen zwischen den Zellen, aber auch einige sehr lange Leitungen.

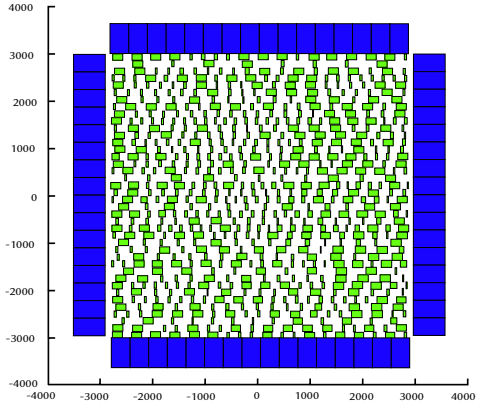
Als Abstand kann sowohl der euklidische als auch der Manhattan-Abstand verwendet werden. Wegen seiner einfachen Berechnung wird letzterer in der Regel vorgezogen.

Platzierung: ..Fläche II

Optimierungskriterium: Fläche II

- In moderner Halbleitertechnologie:
  Verdrahtung über den Zellen möglich
- Flächenoptimierung nicht sinnvoll, da Fläche von der Zellfläche bestimmt wird
- Festlegung der Layoutfläche schon vor der Platzierung

Platzierung ohne Verdrahtungskanäle:



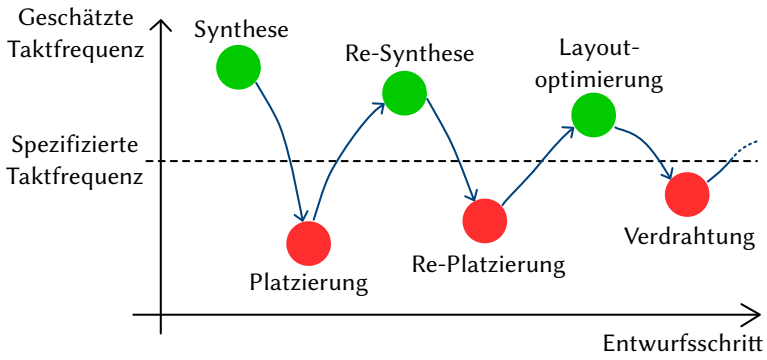
Stehen - wie in modernen Halbleitertechnologien üblich - genügend Verdrahtungsebenen zur Verfügung, kann "über" den Zellen verdrahtet werden. Damit ist keine zusätzliche Verdrahtungsfläche mehr erforderlich. Eine Flächenoptimierung im eigentlichen Sinne ist nicht mehr sinnvoll.
Häufig wird die Layoutfläche schon vor der Platzierung festgelegt und ändert sich während des Platzierungsschrittes auch nicht mehr. In besonderen Anwendungen mit hohen Stückzahlen oder harten Flächenrandbedingungen ist allerdings heute noch die Flächenminimierung von Bedeutung.

Platzierung: ..Timing

Optimierungskriterium: Timing

- Taktraten bei integrierten Schaltungen steigen stetig
- Timing-Closure-Problem
- Diskrepanz zwischen Abschätzung der Delays und tatsächlichen Delays
- Gesamtverdrahtungslänge berücksichtigt Delays unzureichend

⟶ Neue Ansätze notwendig



Bei Schaltungen, die hohe Taktraten aufweisen, spielt das Timing eine große Rolle. Da die Leitungen neben einem Widerstand auch Kapazitäten und Induktivitäten aufweisen, ist eine Ausbreitung der elektrischen Signale mit Lichtgeschwindigkeit nicht möglich. Es vergeht eine gewisse Zeit, bis ein Signal die Strecke von dem Ausgang einer Zelle bis zum Eingang der nächsten Zelle zurückgelegt hat. Der Begriff "Timing Closure" spielt in den heutigen Schaltungen eine sehr wichtige Rolle. Man versteht darunter den Unterschied zwischen den Verzögerungszeiten (Delay) nach der Platzierung bzw. nach der Verdrahtung und den in früheren Entwurfsschritten modellierten Verzögerungszeiten. Sind die Unterschiede nicht vernachlässigbar, ist die Funktionsfähigkeit des gesamten Designs nicht mehr garantiert. Die Abbildung zeigt dazu beispielhaft die notwendigen Iterationen. Für die Berechnung der Verdrahtungslängen bietet sich die quadratische Verdrahtungslänge an, da bei dieser Art der Kostenfunktion die Standardabweichung der Verdrahtungslängen kleiner ist als bei linearer Verdrahtungslängenberechnung.

Platzierung: ..Congestion

Optimierungskriterium: Congestion

- Ziel: Gewährleistung einer automatischen Verdrahtung

- Strategie: Gleichmäßige Verteilung der Verdrahtung auf der Layoutfläche

- Zuweisung von Leitungen zu Teilflächen eines (groben) Rasters

- Congestion: Überschreitung der Kapazität einzelner oder mehrerer Teilflächen

Ziel der Platzierung ist es, eine Anordnung der Zellen zu finden, die eine automatische Verdrahtung der Zellen gewährleistet. Während der Platzierung ist die Verdrahtung abzuschätzen, da die Verdrahtung im Design-Flow erst nach der Platzierung durchgeführt wird. Dabei spielt der Begriff Congestion eine wichtige Rolle, da dieser angibt, wie die Verteilung der Verdrahtungsleitungen auf der Layoutfläche gestaltet ist. Durch Gebiete mit hoher Congestion verlaufen nach der Abschätzung sehr viele Verbindungsleitungen, durch Gebiete mit niedriger Congestion verlaufen wenige Verbindungsleitungen. Das Gebot dieses Optimierungskriteriums ist die gleichmäßige Verteilung der Verdrahtung auf der Layoutfläche. Die Congestion wird generell für eine gerasterte Layoutfläche bestimmt. Jeder Rasterfläche wird eine bestimmte Kapazität zugewiesen. Diese Anzahl von Leitungen darf durch diese Rasterfläche verlaufen, ohne dass es zu Congestion-Problemen kommt. Bei der Platzierung werden jeder Rasterfläche Leitungen zugewiesen. Liegt die Gesamtzahl über der Kapazität, so bedeutet das, dass durch diese Rasterfläche mehr Leitungen verlaufen würden als sie aufnehmen kann. In modernen Halbleitertechnologie stehen inzwischen sehr viele Verdrahtungsebenen zur Verfügung. Aus diesem Grunde sind Congestion-Probleme bei angemessener Platzierungsqualität kaum noch zu erwarten.

Platzierung: ..Weitere

Weitere Optimierungskriterien

- Elektromagnetische Kopplungen
- Thermische Effekte
- Technologische Randbedingungen
- Geometrische Entwurfsregeln

Als weitere Kriterien seien elektromagnetische Kopplungen der durch die Platzierung bedingten Leitungsverläufen, thermische Effekte und technologische Randbedingungen genannt. Elektromagnetische Kopplungen sind parasitäre Effekte, die ebenfalls zu einer Fehlfunktion des Designs führen können. Auf Grund einer schlechten Platzierung kann es passieren, dass Leitungen so verdrahtet werden, dass sich Signale z.B. durch kapazitive Kopplung gegenseitig beeinflussen. Im schlimmsten Falle können Signalpegel so stark verfälscht werden, dass die Funktionsfähigkeit der gesamten Schaltung nicht mehr garantiert werden kann.

Unter thermischen Effekten versteht man Probleme bei der Wärmeabführung in den Zellen. Vor allem schaltaktive Elemente sind besonders zu betrachten, da in CMOS-Schaltungen besonders bei Schaltaktivitäten Ströme fließen. Auf Grund ungünstiger Anordnung der Zellen kann es passieren, dass sich Zellen mit großer Verlustleistung gehäuft in einem Gebiet befinden. Folge davon ist eine erhebliche Erwärmung der Schaltung in dieser Region, was im schlimmsten Fall zu einem Ausfall führen kann. Unter technologischen Randbedingungen sind die Design Rules der verwendeten Technologie zu verstehen, beispielsweise die Anzahl der für die Verdrahtung zur Verfügung stehenden Metallebenen. Darüber hinaus können die in der Bibliothek vorhandenen Zellen sowie weitere geometrische Entwurfsregeln die Platzierung maßgeblich beeinflussen.

Platzierung: Verdrahtungslängenabschätzung

Verdrahtungslängenabschätzung

- Verdrahtungsinformation geht in die Platzierung ein
- Verdrahtung folgt auf die Platzierung

→ Verdrahtungslängenabschätzung notwendig

- Bounding Box
- Kette
- Minimaler Spannbaum
- Steinerbaum

Zur Erzeugung einer Platzierung wird Verdrahtungsinformation benötigt, da diese fester Bestandteil vieler Kostenfunktionen ist. Die korrekte Information steht aber erst im Anschluss an die durchgeführte Verdrahtung, die nach der Platzierung berechnet wird, zur Verfügung. Daher ist eine Abschätzung der Verdrahtungslänge während der Platzierung vorzunehmen. Nachfolgend sind die gängigen Möglichkeiten aufgelistet:

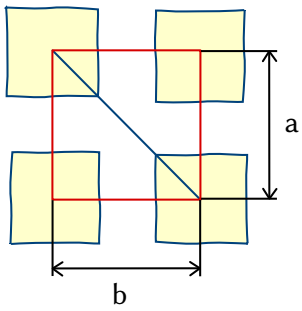
- Bounding-Box
- Kette
- Minimaler Spannbaum
- Steinerbaum

Platzierung: ..Bounding Box

Abschätzung: Halbe Bounding-Box

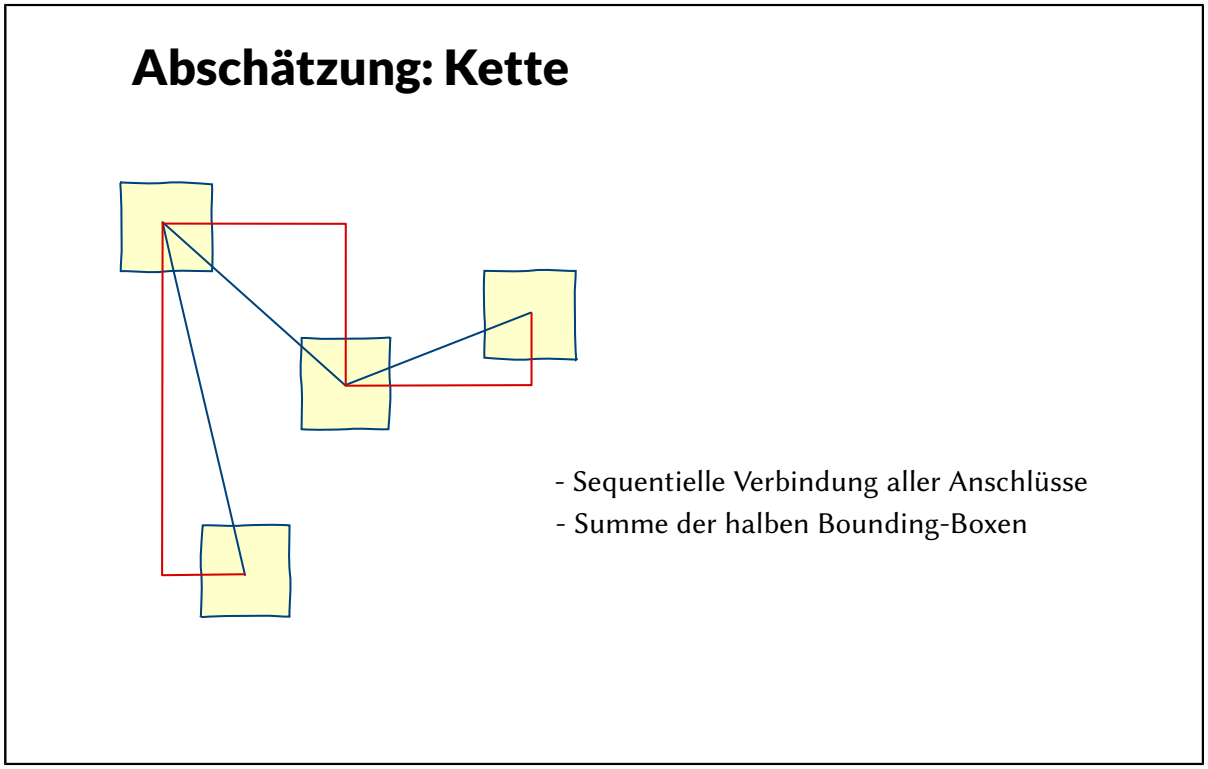
- Halber Umfang des umschließenden Rechtecks

- Schnell zu berechnen, sehr häufig verwendet



Die Bounding-Box ist die einfachste der Abschätzungsmöglichkeiten. Da sie sehr schnell zu berechnen ist, ist sie auch die am meisten verwendete. Alle anderen Abschätzungen benötigen zum Teil erheblich mehr Rechenzeit. Die Bounding-Box stellt den halben Umfang des umschließenden Rechtecks der zu verbindenden Anschlüsse dar. Es sollte beachtet werden, dass bei Zweipunkt- und Dreipunktnetzen die Abschätzung mit der halben Bounding-Box dieselben Ergebnisse liefert wie die Steinerbaum-Abschätzung, die später behandelt wird.

Platzierung: ..Kette

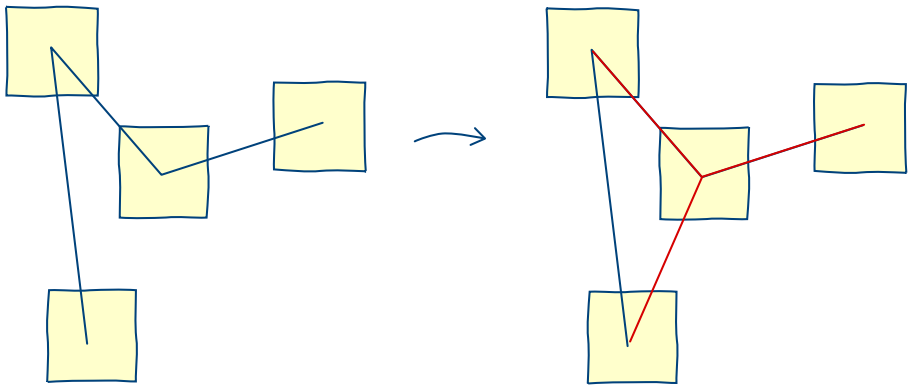


Bei der Kette werden die zu verbindenden Anschlüsse sequentiell miteinander verbunden, wobei die Reihenfolge der Verbindungen frei wählbar ist. Die Gesamtlänge kann als Summe der halben Bounding-Box-Umfänge von jeweils zwei aufeinander folgenden Anschlüssen berechnet werden.

Platzierung: ..Spannbaum

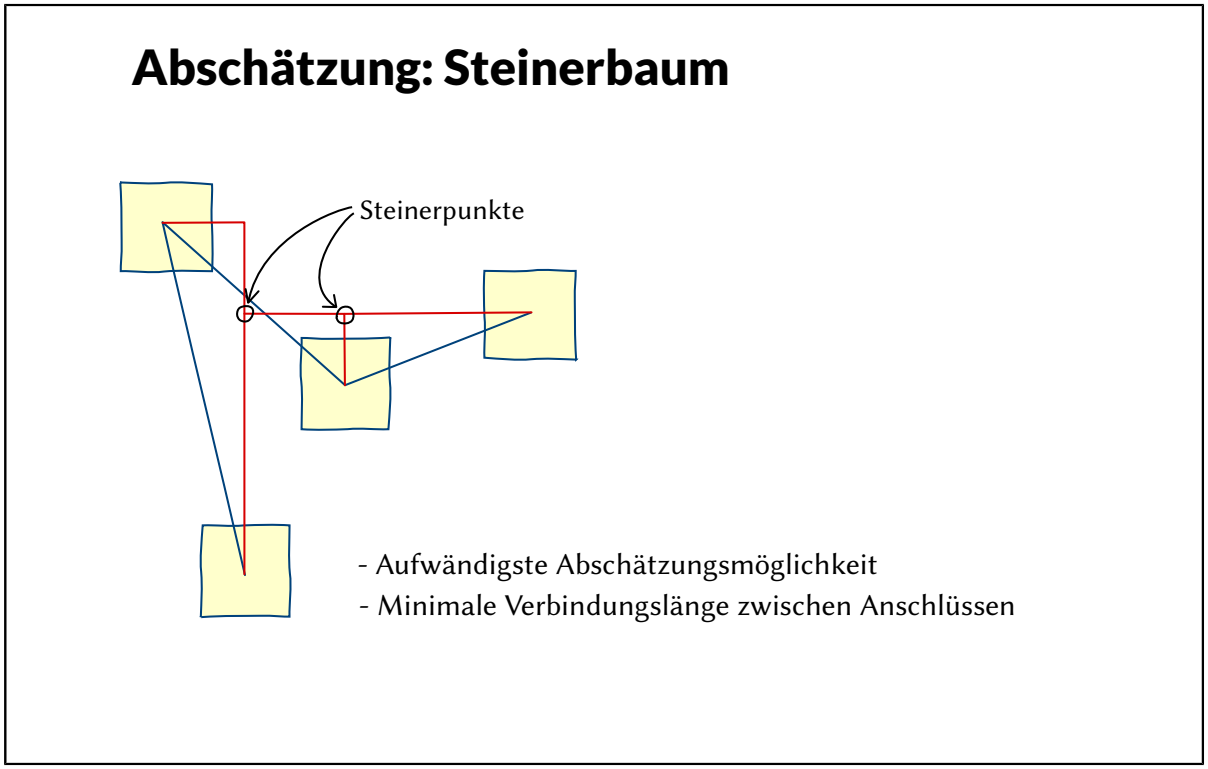
Abschätzung: Minimaler Spannbaum

- Minimaler Spannbaum: kürzester verbundener Teilgraph mit allen Knoten



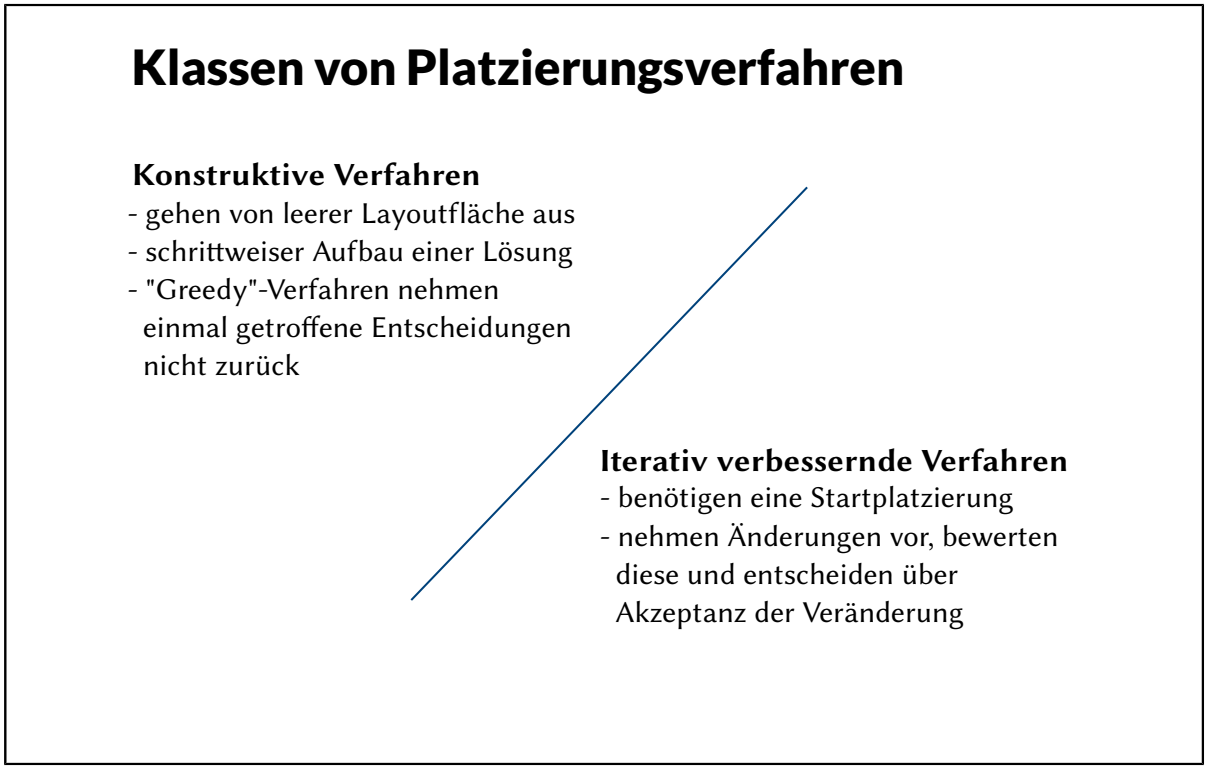
Bei der Berechnung des minimalen Spannbaumes ist es zunächst notwendig, alle Anschlüsse miteinander zu verbinden. Diese Anordnung wird als vollständiger Graph bezeichnet. Der minimale Spannbaum ist der kürzeste verbundene Teilgraph des vollständigen Graphen, der alle Knoten enthält.

Platzierung: ..Steinerbaum



Der Steinerbaum stellt die aufwändigste Abschätzung der Verdrahtungslänge dar, einhergehend mit einem erheblich höheren Rechenaufwand. Der charakteristische Unterschied des Steinerbaums zum Spannbaum ist die Tatsache, dass zusätzlich zu den schon vorhandenen Punkten (das sind die Verbindungspunkte an den Zellen) spezielle Verzweigungspunkte (so genannte Steinerpunkte) eingefügt werden können. Der Steinerbaum beschreibt die minimale Verbindungslänge zwischen den Anschlüssen.

Platzierung: Klassen von Platzierungsverfahren

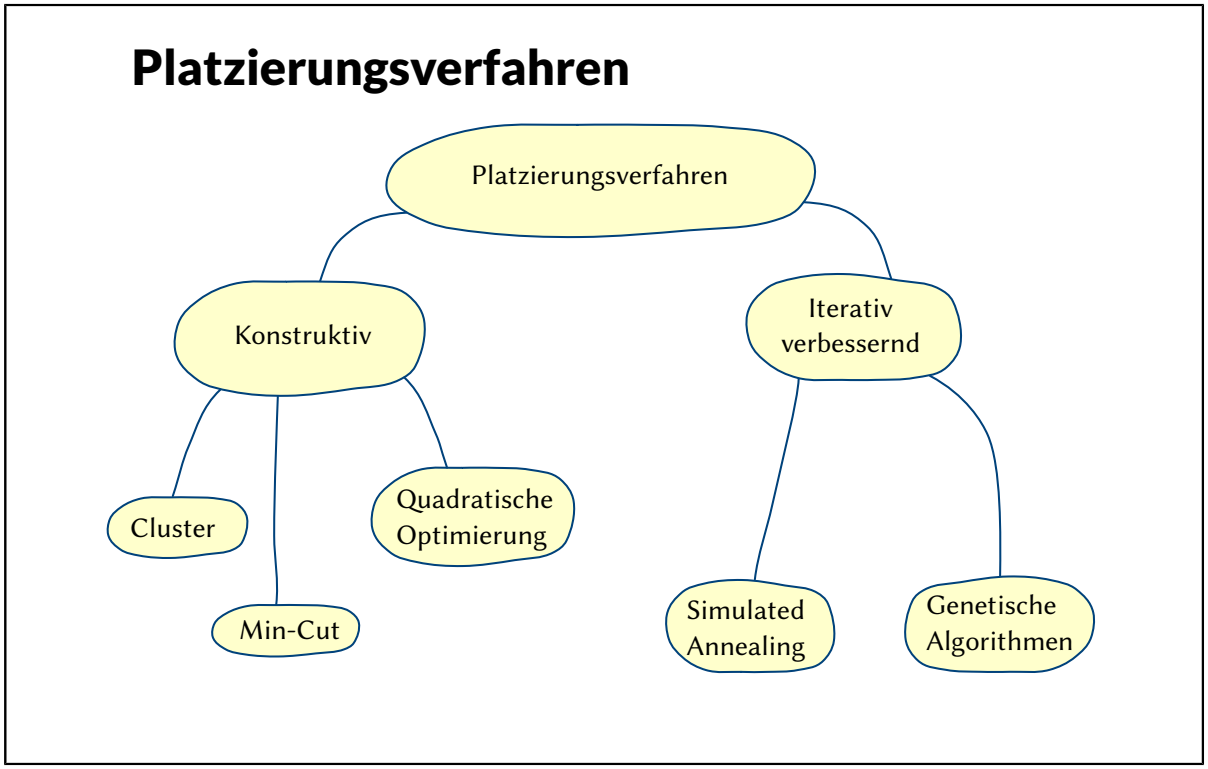


Optimierungsverfahren zur Platzierung werden in zwei Gruppen unterteilt, in konstruktive und iterativ verbessernde Optimierungsverfahren.

Konstruktive Optimierungsverfahren gehen von einer leeren Layoutfläche aus und bauen die Platzierung nach und nach auf. Schrittweise werden die Zellen auf der Layoutfläche angeordnet. Werden platzierte Zellen nicht mehr verschoben, d.h. werden einmal getroffene Entscheidungen nicht mehr revidiert, wird diese Vorgehensweise als "greedy" (gierig) bezeichnet. Zu der Klasse der konstruktiven Optimierungsverfahren gehören partitionsbasierte Verfahren. Partitionsbasierte Verfahren entsprechen der Layouterzeugungsstrategie "divide-and-conquer". Da die Problemgröße zu groß ist, wird durch die Partitionierung die Problemgröße so weit verkleinert, dass man diese beherrschen kann. Bei der Platzierung wird die Partitionierung weitergeführt, bis in einem Teil der Partition nur noch eine Zelle vorhanden ist.

Die iterativen Optimierungsverfahren benötigen eine Startplatzierung. Diese wird in der Praxis häufig mit konstruktiven Platzierungsverfahren gewonnen, kann aber auch zufällig erzeugt worden sein. Diese Optimierungsverfahren führen Veränderungen an der Startplatzierung durch und bewerten anhand der Kostenfunktion, ob die Veränderung zu einer Verbesserung der Platzierung geführt hat oder nicht. Anschließend wird entschieden, ob die Veränderung durchgeführt oder verworfen wird. Das Verfahren wird abgebrochen, wenn sich keine Verbesserungen durch die Veränderungen einstellen oder eine bestimmte Schranke für die Kostenfunktion unterschritten wird.

Platzierung: Platzierungsverfahren



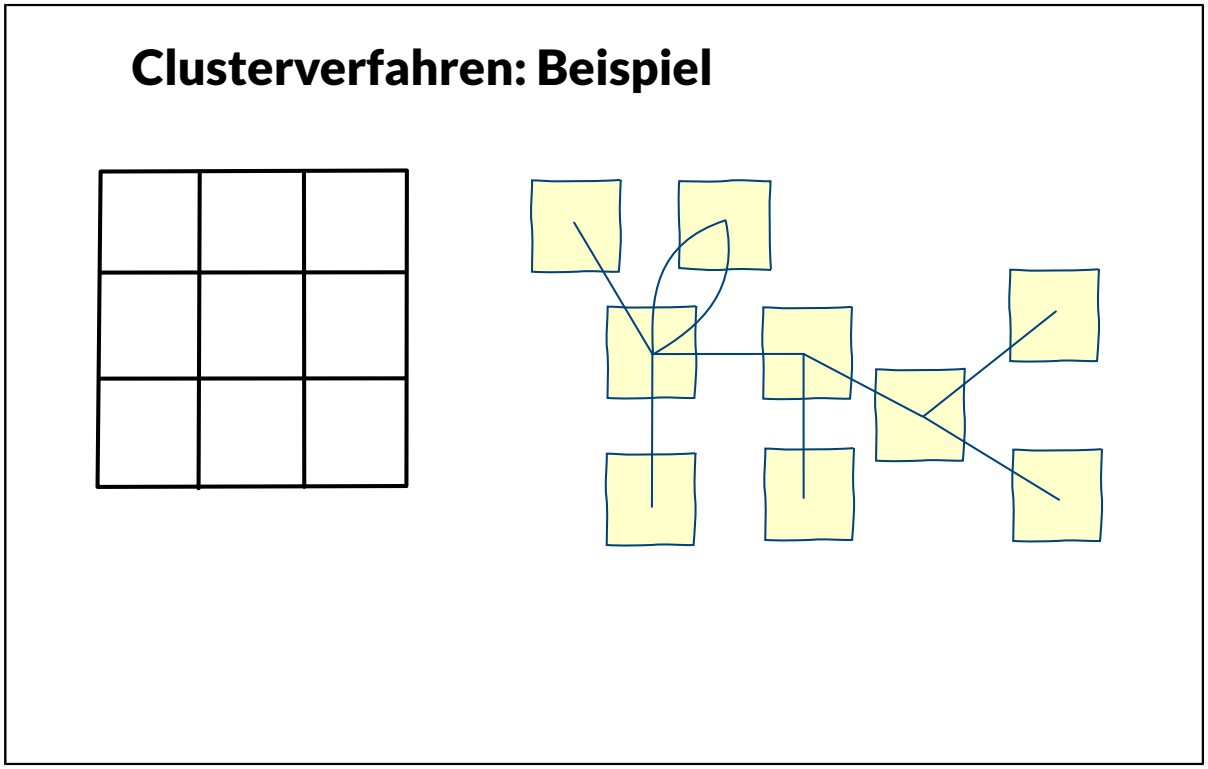
Platzierung: ..Clusterverfahren

Clusterverfahren

- Intuitiver Platzierungsansatz
- Bottom-up-Ansatz
- Greedy-Vorgehen

Das Cluster-Verfahren stellt einen intuitiven Platzierungsansatz dar. Bei diesem Verfahren wird zunächst eine Zelle auf der Layoutfläche platziert. Sequentiell werden die nächsten Zellen ausgewählt und ebenfalls auf der Layoutfläche platziert. Die Auswahl der zweiten und der folgenden Zellen ist nicht einheitlich geregelt. Meistens berücksichtigt man, wie stark die Verbindungen zu den schon platzierten Zellen sind. Generell wird eine Liste mit der Platzierungsreihenfolge unter Berücksichtigung der eben genannten Eigenschaft erstellt, die dann sequentiell abgearbeitet wird. Dieses Verfahren hat die Bezeichnung Cluster-Verfahren erhalten, da mit jeder Zelle, die zusätzlich auf der Layoutfläche platziert wird, eine weitere Zelle der Gruppe hinzugefügt wird. Die stark miteinander vernetzten Zellen werden in räumlicher Nähe angeordnet. Die explizite Platzierung in räumlicher Nähe wird nach Berechnung der Kostenfunktion für die jeweilige Anordnung durchgeführt. Dabei ist nachteilig, dass für die Berechnung der Kosten nur die Zellen eingehen können, die sich schon auf der Layoutfläche befinden. Alle noch nicht platzierten Zellen gehen nicht in die Berechnung der Kostenfunktion ein. Neben der Auswahl nach den meisten Verbindungen zu den schon platzierten Zellen gibt es Varianten, bei denen zufällig die nächsten Zellen ausgesucht und auf der Layoutfläche platziert werden. Dieser Ansatz beschreibt einen Bottom-Up-Ansatz. Getroffene Entscheidungen werden nicht revidiert, d.h. das Cluster-Verfahren ist "greedy".

Platzierung: ..Clusterverfahren: Beispiel



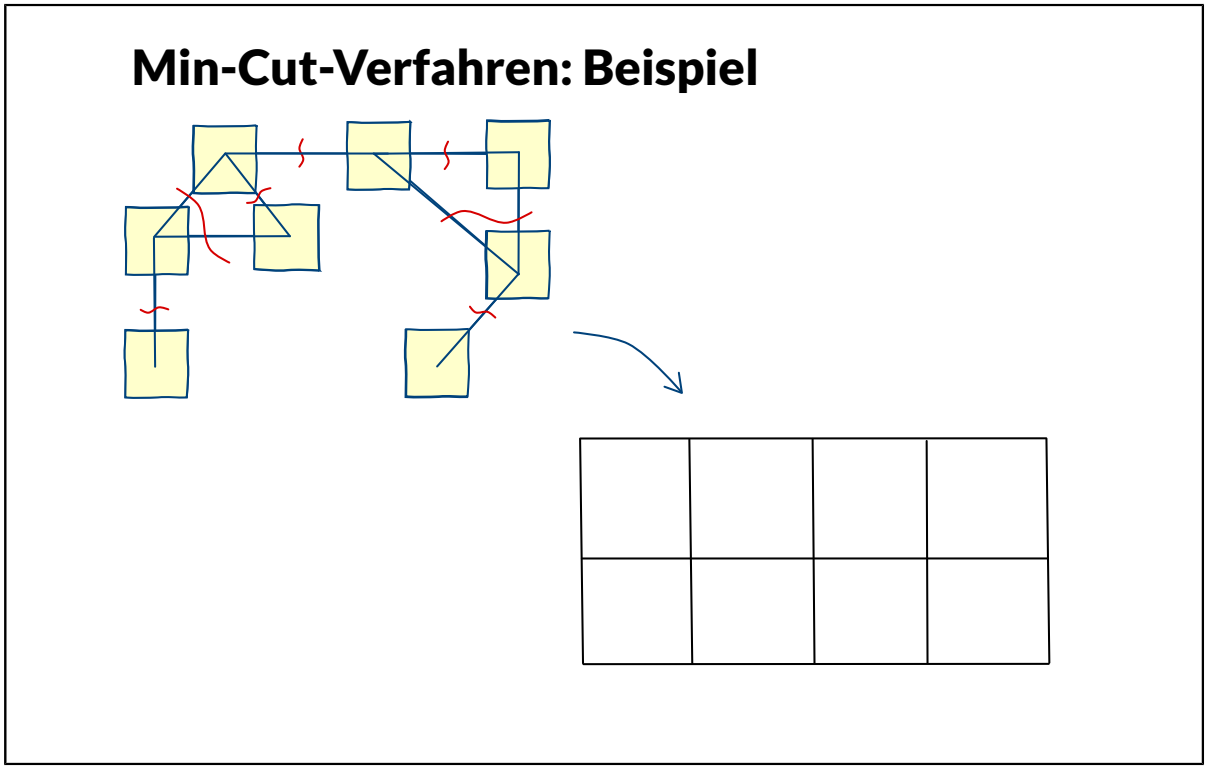
Platzierung: ..Min-Cut

Min-Cut-Verfahren

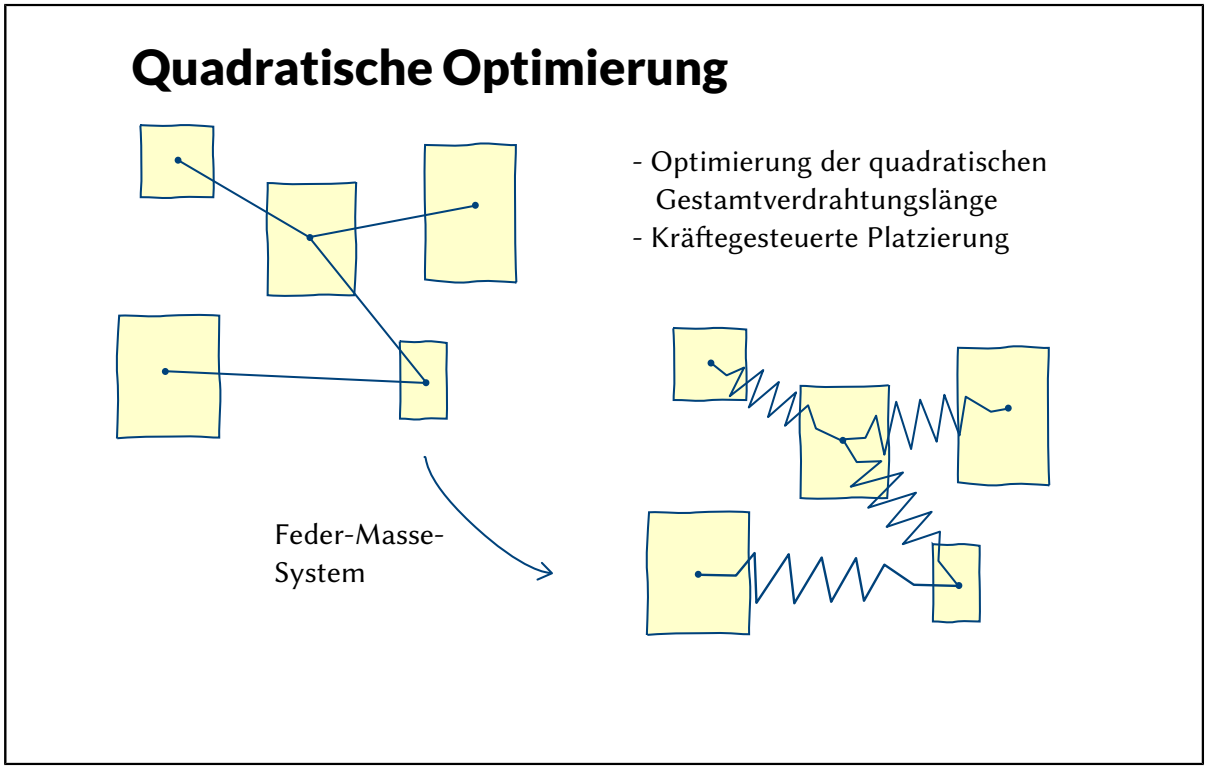
- Top-Down-Ansatz
- Rekursive Aufteilung (Partionierung) der Gesamtschaltung und der Layoutfläche
- Häufig: Aufteilung in zwei Teile (Bipartionierung)
- Bedingung: Möglichst kleine Anzahl von Verbindungen zwischen Teilpartionen

Das Min-Cut-Verfahren ist ein Partitionierungsalgorithmus. Es stellt im Gegensatz zum Cluster-Verfahren einen Top-Down-Ansatz dar. Das Grundprinzip des Min-Cut-Verfahrens beruht auf der rekursiven Aufteilung der Gesamtschaltung in jeweils zwei oder mehrere Teile. Die Aufteilung in zwei Teile wird Bipartitionierung, die Aufteilung in mehrere (>2) Teile wird Multiway-Partitionierung genannt.Gleichzeitig wird die Layoutfläche entsprechend den prozentualen Flächenanteilen der Teilpartitionen aufgeteilt. Untersuchungen haben gezeigt, dass eine Bipartitionierung (also die Aufteilung in zwei möglichst gleich große Teilpartitionen) die besten Ergebnisse liefert. Dabei ist die Aufteilung der Fläche abwechselnd horizontal und vertikal durchzuführen. Die Partitionierung wird so lange durchgeführt, bis jede Teilpartition nur noch eine einzige Zelle aufweist. Rekursiv wird dann die Platzierung aufgebaut. Bei der Aufteilung einer Partition in Teilpartitionen wird darauf geachtet, dass möglichst wenige Verbindungen zwischen den beiden Teilpartitionen existieren. Das bedeutet, dass möglichst wenige Schnitte bei den Verbindungen gesetzt werden müssen. Diese Tatsache hat dem Verfahren den Namen "Min-Cut-Verfahren" gegeben.

Platzierung: ..Min-Cut: Beispiel



Platzierung: ..Quadratische Optimierung



Der Platzierungsansatz Quadratische Optimierung ist auch unter dem Begriff "Kräftegesteuerte Platzierung" bekannt. Unter der quadratischen Optimierung versteht man die Optimierung der quadratischen Abstände zwischen den Zellen. Das Verfahren modelliert das aus der Mechanik bekannte Feder-Masse-System, wobei die Zellen die Massen und die Verbindungen zwischen diesen die Federn darstellen. Die Massen werden als Punkte modelliert, d.h. sie haben keine physikalische Ausdehnung.

Platzierung: ....Feder-Masse-Modell

Feder-Masse-Modell

- Hookesches Gesetz

- Anziehende Kräfte zwischen Zellen aufgrund der Federkraft

- Gewichtung durch unterschiedliche Federkonstanten

- Generell: Unterscheidung zwischen festen und beweglichen Zellen

Aufgrund des Hookeschen Gesetzes verursachen die Federn bei räumlicher Entfernung der Zellen anziehende Kräfte. Die Federn modellieren die elektrischen Verbindungen zwischen den Zellen. Dabei können verschiedenen Verbindungen unterschiedliche Federkonstanten zugewiesen werden, wodurch eine größere Flexibilität und Einsetzbarkeit des Verfahrens gewährleistet wird. Sollen beispielsweise zwei Zellen aufgrund von Timing-Anforderungen sehr nah beieinander liegen, so kann die Federkonstante der Verbindung zwischen diesen Zellen höher eingestellt werden, so dass die anziehende Kraft aufgrund dieser Feder ebenfalls größer wird. Häufig wird die Federkonstante entsprechend der Zahl der Verbindungen zwischen zwei Zellen eingestellt. Generell wird zwischen beweglichen und festen (z.B. Ports) Zellen unterschieden.

Platzierung: ....Energie-Berechnung

Energie-Berechnung

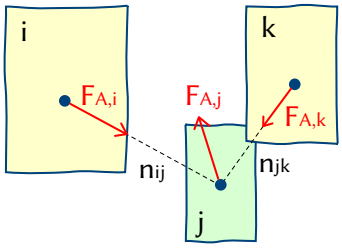
- Physikalische Systeme: Anordnung in einem energiearmen Zustand
- Potentielle Energie einer Feder

$$E = \int \vec{F}\vec{ds} = \int c \cdot x \, dx = c \cdot \frac{x^2}{2} + \text{const.}$$

- Summe über alle Federn liefert potentielle Energie des Systems

$$E_{ges} = \sum_i E_i$$

- Entspricht Summe der quadrierten Längen



Physikalische Systeme sind bestrebt, einen möglichst energiearmen Zustand einzunehmen.

Die potentielle Energie einer Feder wird durch eine Integration über die entsprechende Auslenkung der Feder berechnet. Die Federkraft berechnet sich zu Federkonstante mal Auslenkung. Die Integration über die Auslenkung liefert die potentielle Energie. Werden die potentiellen Energien aller Federn addiert, so erhält man die potentielle Energie des Federsystems. Diese Energie ist proportional zu der Summe der quadrierten Längen.

Platzierung: ....Kräftegleichgewicht

Kräftegleichgewicht

- Kräfte zwischen beweglichen sowie zwischen festen und beweglichen Zellen liefern Energiebetrag

  $E_1 = 0{,}5\ \mathbf{x}^T\ \mathbf{C}\ \mathbf{x}$

- Kräfte zwischen festen und beweglichen Zellen liefern zusätzlichen Energiebetrag

  $E_2 = \mathbf{x}^T\ \mathbf{d}$

- Kräfte zwischen festen Zellen liefern zusätzlich konstanten Energiebetrag
- Gestamtenergie des Federsystems ergibt sich als

  $E_{total} = 0{,}5\ \mathbf{x}^T\mathbf{C}\ \mathbf{x} + \mathbf{x}^T\ \mathbf{d} + \text{const.}$

- Minimieren durch Ableitung und Lösung der Gleichung

  $\mathbf{C}\ \mathbf{x} + \mathbf{d} = \mathbf{0}$

- Entspricht Kräftegleichgewicht des Federsystems

Zu den bisher beschriebenen Kräften zwischen den beweglichen Zellen treten Kräfte zwischen den festen und beweglichen Zellen sowie zwischen festen Zellen untereinander auf. Diese liefern zusätzliche Energiebeiträge.

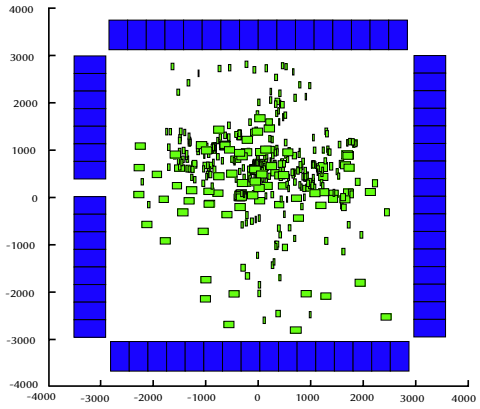
Da physikalische Systeme bestrebt sind, den Zustand minimaler Energie anzunehmen, wird das Minimum der potentiellen Energie durch die Ableitung der Matrixgleichung berechnet. Die Ableitung ist C*x+d. Extremwerte lassen sich durch die Nullstellen der Ableitung bestimmen. Physikalisch entspricht die Berechnung der neuen Auslenkungen X einem Kräftegleichgewicht. Man bestimmt also die Position der Masseteilchen, bei der die auf die Masseteilchen wirkenden Kräfte sich aufheben.

Mathematisch betrachtet lässt sich die Kostenfunktion für die quadratische Optimierung mittels einer Matrixgleichung beschreiben. Die Matrix C beinhaltet die Verbindungen zwischen den beweglichen Zellen sowie zwischen festen und beweglichen Zellen, während Verbindungen zwischen beweglichen und festen Zellen zusätzlich Beiträge zu der Matrix d liefern. Das Minimum der Kostenfunktion kann mittels der Gleichung Cx + d = 0 gefunden werden. Diese Gleichung stellt die Ableitung der Kostenfunktion dar. An dieser Gleichung sieht man die Notwendigkeit der festen Zellen für diesen Ansatz. Sind keine festen Zellen vorhanden, so ist der Vektor d ein Nullvektor. Das führt dazu, dass die Platzierung x ebenfalls ein Nullvektor wird. Alle Zellen liegen dann im Ursprung übereinander.

Platzierung: ....Initialplatzierung

Initialplatzierung

Lösung der Matrixgleichung ergibt Initialplatzierung:



Die Lösung der Gleichung Cx + d = 0 stellt lediglich eine so genannte Initialplatzierung dar. Es sind noch viele Überlappungen vorhanden.

Platzierung: ....Abstoßende Kräfte

Abstoßende Kräfte

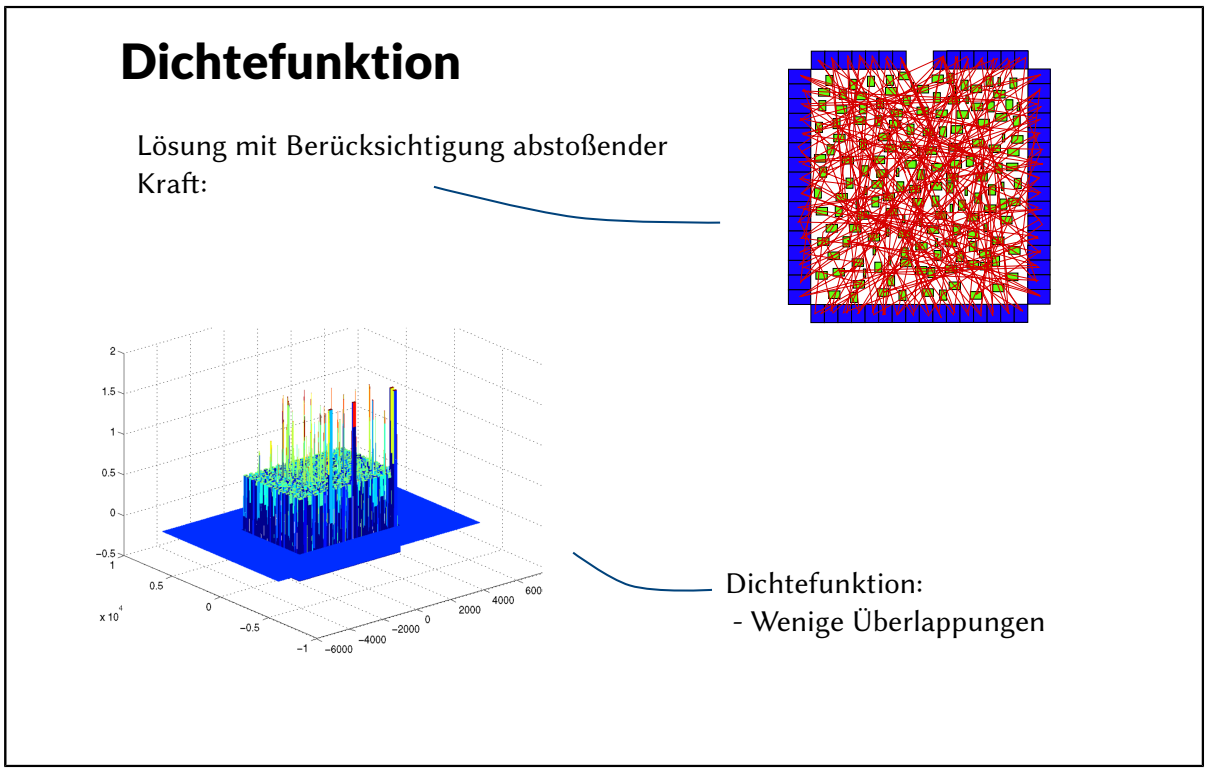
- Extreme Ungleichverteilung auf der Layoutfläche
- Viele Überlappungen
- Einführung von abstroßenden Kräften **e** für eine gleichmäßige Verteilung

$$\mathbf{C\,x + d + e = 0}$$

- Iterative Lösung durch schrittweise Aktualisierung von **e**

Aufgrund der Überlappungen ist eine extreme Ungleichverteilung auf der Layoutfläche vorhanden. Um diese Ungleichverteilung zu beseitigen, werden abstoßende Kräfte eingeführt. Die zu lösende Gleichung lautet dann C*X + d + e = 0. Die abstoßenden Kräfte werden iterativ angepasst. Mit jeder neuen Iteration werden zunächst die abstoßenden Kräfte aktualisiert, und anschließend wird die neue Platzierung berechnet.

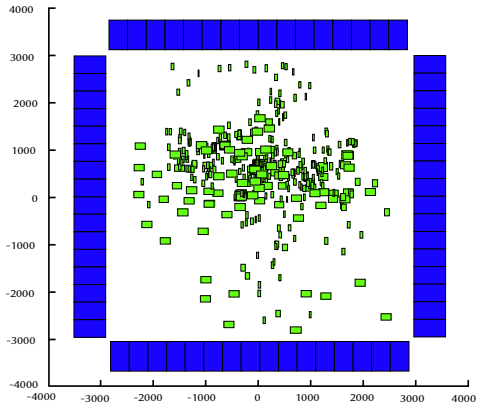
Platzierung: ....Dichtefunktion



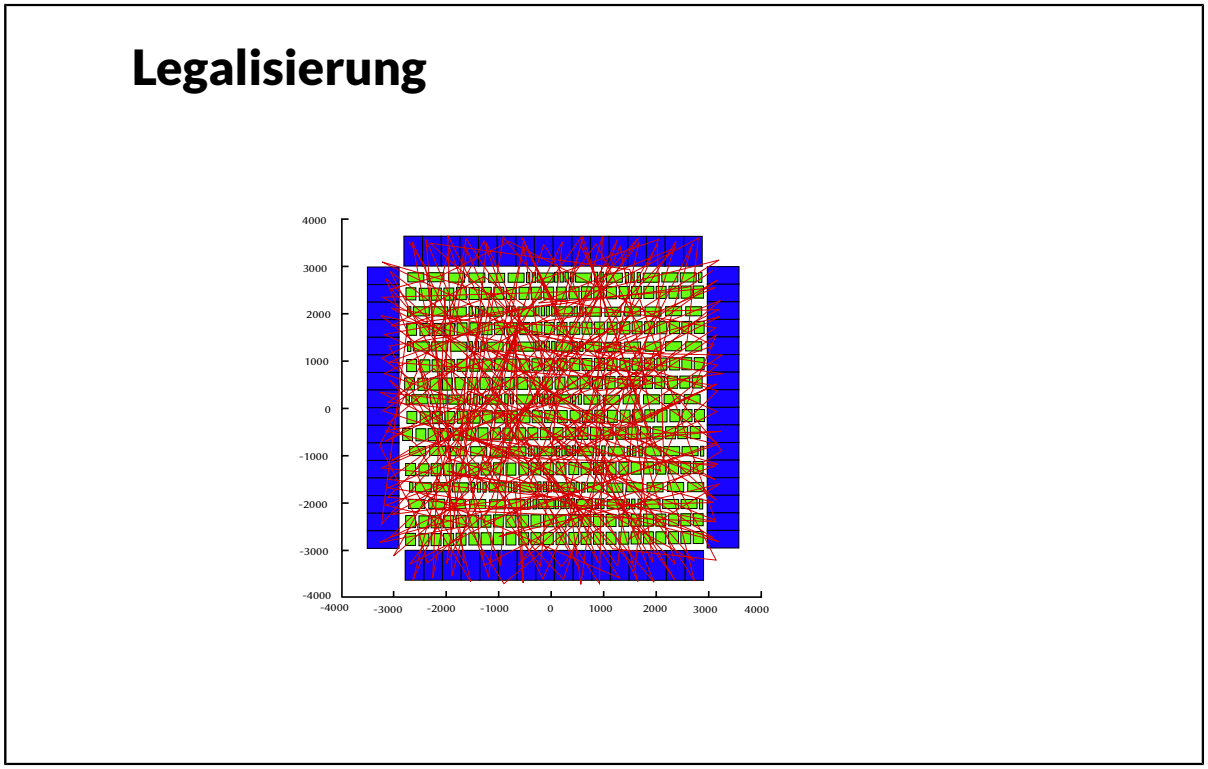
Die Größe der abstoßenden Kräfte wird häufig proportional zur Anzahl der sich in einem Flächenbereich überlappenden Zellen gewählt. Hiermit kann eine so genannte Dichtefunktion bestimmt werden. Das Bild zeigt eine mit Berücksichtigung abstoßender Kräfte berechnete Platzierung. Die Zellen sind relativ gleichmäßig auf der Layoutfläche verteilt. Da nur noch einige wenige Überlappungen existieren, sind nur wenige Peaks in der Dichtefunktion vorhanden.

Platzierung: ....Beispiel

Beispiel



Platzierung: ....Legalisierung



Zum Abschluss der Platzierung findet häufig (z.B bei der Platzierung von Standardzellen) ein so genannter Legalisierungsschritt statt. Dabei werden die berechneten Platzierungspositionen so angepasst, dass sich eine "regelmäßige" Platzierung, z.B. in Reihen, ergibt.

Platzierung: ..Simulated Annealing

Simulated Annealing

- Häufig eingesetztes Optimierungsverfahren
- Liefert sehr gute Ergebnisse
- Rechenintensiv

Simulated Annealing ist ein Optimierungsverfahren, das schon beim Floorplanning ausführlich vorgestellt worden ist. Aufgrund der vielen Einsatzmöglichkeiten und der einfachen Implementierung wird Simulated Annealing auch für die Platzierung eingesetzt. Das Verfahren liefert sehr gute Ergebnisse, ist aber auch sehr rechenintensiv.

Platzierung: ..Genetische Algorithmen

Genetische Algorithmen

- Individuen: Mögliche (gültige!) Lösungen des Platzierungsproblems
- Gene: Symbole für die Darstellung der Lösung
- Chromosomen: Kette von Symbolen
- Generation: Iterationsschritt
- Population: Menge von Individuen
- Fitness: Bewertungsfunktion für die Güte

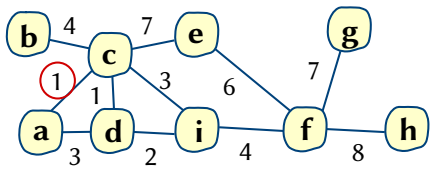
Genetische Algorithmen sind der Evolution in der Natur nachempfunden. In der Natur passen sich die Lebewesen der Umgebung, in der sie leben an. Diejenigen, die an die Umgebung am besten angepasst sind, haben eine erheblich größere Überlebenswahrscheinlichkeit. Das Phänomen wird als "survival of the fittest" bezeichnet. Genetische Algorithmen werden auch für Platzierung integrierter Schaltungen verwendet. Mögliche Lösungen des Platzierungsproblems werden als Individuen bezeichnet. Sie werden häufig durch eine Kette von Symbolen repräsentiert. Die Symbole, die für die Darstellung der Lösung benötigt werden, werden als Gene bezeichnet. Eine Kette von Symbolen bestimmter Länge wird als Chromosom bezeichnet. Da genetische Algorithmen iterativ sind, wird jede Iteration mit dem Begriff der Generation gekennzeichnet. Eine Menge gültiger Platzierungen (Individuen) wird mit Population bezeichnet. Nach jeder Iteration werden die Individuen der Populationen anhand von Fitness-Tests bewertet. Dieses stellt die Kostenfunktion bei den genetischen Algorithmen dar. Die Fitness bewertet somit die Qualität der Platzierung.

Platzierung: ....Beispiel

Genetische Algorithmen: Beispiel

Graph der Schaltung:

Knoten: Zellen
Kanten: Verbindungsgewichte



Positiondefinition:

6	7	8
3	4	5
0	1	2

Mögliche Lösung: [aghcbidef]

$Kosten_{fi} = 1 * 1 = 1$

d	e	f
c	b	i
a	g	h

Fitness: $\frac{1}{1}$

Als Beispiel sei hier ein Zuordnungsproblem betrachtet. Die neun Knoten des Graphen repräsentieren neun Zellen, die auf die neun Plätze 0 bis 8 verteilt werden sollen. An den Kanten des Graphen sind die Gewichte für die einzelnen Verbindungen angetragen. Die Fitness wird hier als der Kehrwert der gewichteten Manhattan-Abstande berechnet. Eine mögliche Lösung des Problems ist in Teil (c) der Abbildung dargestellt. Die Darstellung der Lösung erfolgt über eine Zeichenkette der Länge 9, wobei das erste Zeichen der Zeichenkette der Position 0 zugeordnet wird und das letzte Zeichen der Position 8. Demnach kann man als Lösung [aghcbidef] als Lösungs-Zeichenkette angeben. Die Fitness berechnet sich dann zu 1/85.

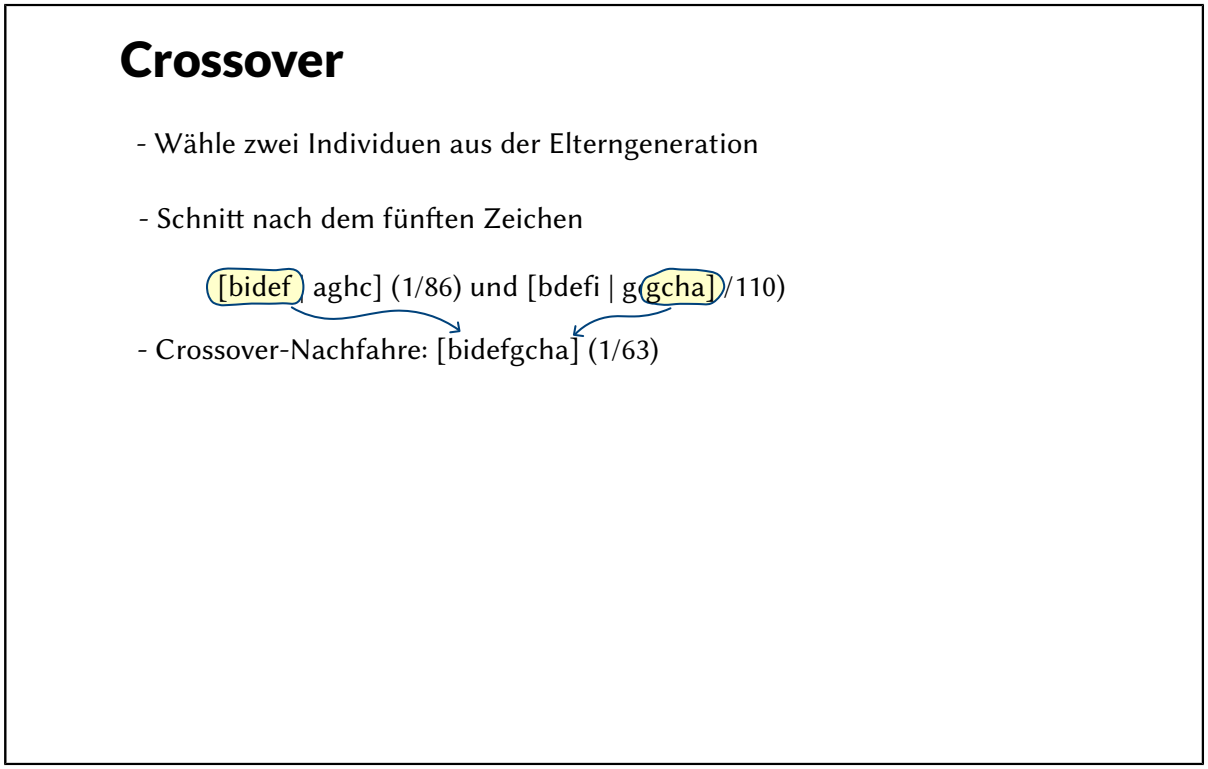
Platzierung: ....Operatoren

Genetische Operatoren

- Erzeugung neuer Generation durch Anwendung genetischer Operatoren
  - Crossover
  - Mutation
  - Selektion
- Anzahl der Individuen in einer Generation beschränkt

Aus den Eltern werden die Nachfahren (neue Generation) durch die Anwendung dreier genetischer Operatoren gewonnen. Die drei genetischen Operatoren sind Selektion, Crossover und Mutation. Den durch die drei Operatoren gewonnenen Individuen, den Nachfahren, werden Fitnesswerte zugewiesen. Die neue Generation ergibt sich aus einer bestimmten Anzahl von Individuen aus der Elterngeneration und einer bestimmten Anzahl von Individuen aus der Nachfolgegeneration. Die Anzahl der Individuen in der Population ist beschränkt. Da für die Nachfolgegeneration bevorzugt Individuen höherer Fitness ausgewählt werden, werden Individuen mit schlechten Genen und schließlich schlechter Fitness aus der Population gelöscht. Das Ergebnis ist, dass die Population im Verlauf der Generationen zunehmend bessere Platzierungsergebnisse aufweist.

Platzierung: .....Crossover

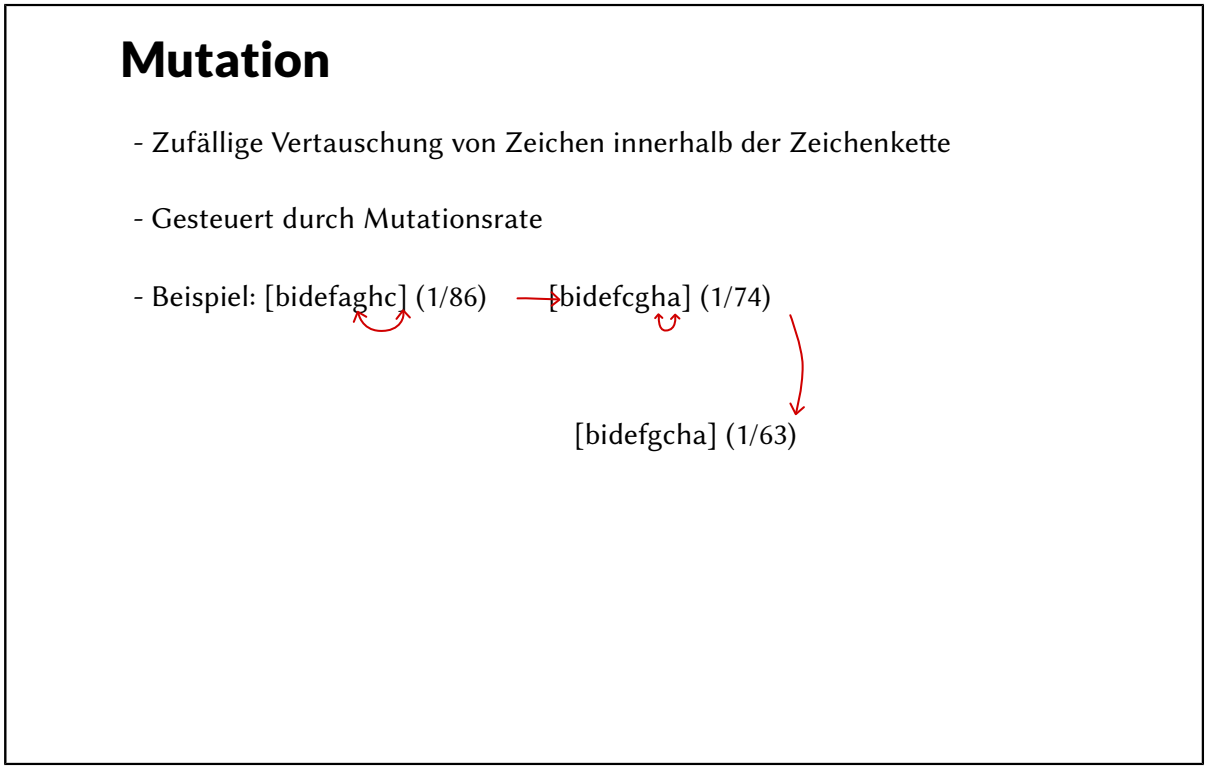


Crossover ist eine sehr wichtige genetische Operation. Es benötigt mehr als ein Individuum aus der Elterngeneration, um ein Individuum der Nachfolgegeneration zu erzeugen. im gezeigten Beispiel besteht das Crossover aus einem zufällig ausgewählten Schnitt und der Kombination aus der linken Hälfte des einen Elternteils mit dem anderen Teil des anderen Elternteils.

Wird zufällig der Schnitt nach dem fünften Zeichen durchgeführt, so erhält man mit den dargestellten Elternindividuen, von dem ersten Elternteil wird der linke Teil der Zeichenkette, von dem zweiten Elternteil der rechte Teil der Zeichenkette übernommen, das neue Individuum mit der Fitness (1/63).

Das hier vorgestellte Beispiel ist ein Sonderfall, da Zeichen aus dem linken Teil des ersten Elternteils in dem rechten Teil des zweiten Elternteils nicht vorkommen. Wäre dieses der Fall, so wäre der Nachfahre keine gültige Lösung des Platzierungsproblems. Aus diesem Grunde sind verschiedene Modifikationen an dem Crossover-Operator vorzunehmen, so dass die Nachfahren gültige Lösungen darstellen.

Platzierung: .....Mutation



Die Mutation ist eine weitere genetische Operation. Nachfahren werden hier dadurch erzeugt, dass zufällig Vertauschungen an den Genen durchgeführt werden. Die Mutation wird durch die Mutationsrate gesteuert. Sie gibt das Verhältnis der Vertauschungen zur Anzahl der Gene an.

Das Beispiel zeigt das ursprüngliche Individuum. Durch zweifache Mutation (Vertauschung von a mit c, anschließend Vertauschung von c und g) entsteht das Nachfolgeindividuum, das zufälligerweise dieselbe Fitness aufweist.

Platzierung: .....Selektion

Selektion

- Auswahl der Individuen für Nachfolgegeneration

- Populationsgröße bleibt konstant

- Auswahlmöglichkeiten:
  - Zufällig
  - Auswahl der besten Individuen
  - Kombination der beiden Ansätze

Nebenbedingung bei der Auswahl der Individuen für die Nachfolgegeneration ist, dass die Populationsgröße konstant bleibt.

Der Ansatz aus der Vereinigungsmenge der aktuellen Generation und den erzeugten Nachfahren zufällig Individuen auszuwählen, kann dazu führen, dass die Fitness der Population fällt. Auf der anderen Seite bleibt eine globale Sicht erhalten, da die Möglichkeit besteht, auf Grund der Akzeptanz der schlechteren Individuen, lokale Minima zu überwinden. Bei der Auswahl der besten Individuen steigt die Fitness mit jeder neuen Generation, da nur die Individuen mit den besten Fitnesswerten in die Nachfolgegeneration übernommen werden.

Eine Kombination der beiden Ansätze scheint vielversprechend, da die Vorteile beider Ansätze kombiniert werden können.

Platzierung: Bewertung der Verfahren

Bewertung der Verfahren

Verfahren	Qualität	Rechenzeit
Cluster	-	+
Min-Cut	0	+
Quadratische Optimierung	+	+
Simulated Annealing	+	0
Genetische Algorithmen	+	-

Während die konstruktiven Platzierungsalgorithmen (Cluster-Verfahren oder Min-Cut-Verfahren) sehr schnell sind, erreichen die Platzierungsergebnisse nicht die Qualität der iterativ verbessernden Algorithmen. In der Praxis werden aus diesem Grunde zumeist konstruktive Platzierungsalgorithmen mit nachgeschaltetem iterativ verbesserndem Teil verwendet. Damit sind derzeit gängige Problemgrößen handhabbar. Die Ergebnisqualität der kräftegesteuerten Platzierungsverfahren ist mit den Platzierungsergebnissen des derzeit gängigsten Verfahrens, des Simulated Annealing, vergleichbar. Wird dem Simulated Annealing jedoch hinreichend viel Rechenzeit zur Verfügung gestellt, so findet der Algorithmus ein sehr gutes Ergebnis. Das ist der Grund dafür, dass dem Simulated Annealing bei der Platzierungsqualität ein besserer Wert zugewiesen wird als dem Min-Cut-Verfahren.

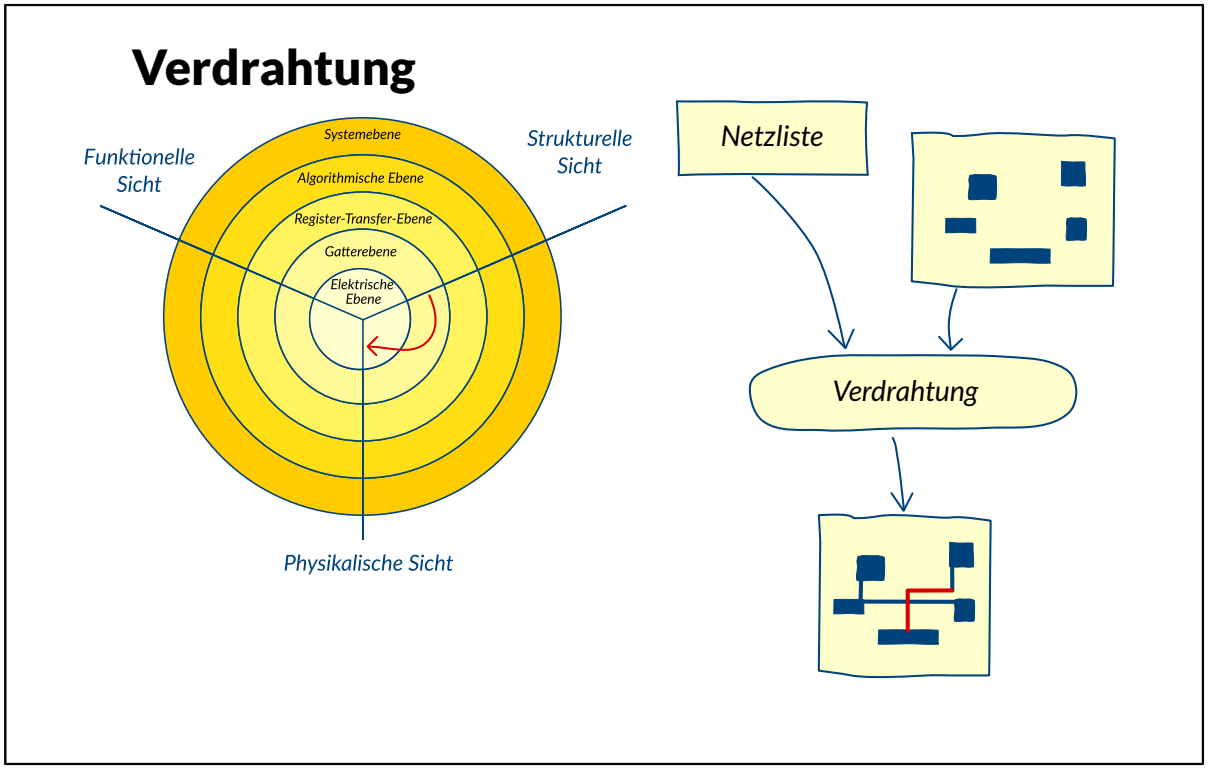
Electronic Design Automation (EDA)

Verdrahtung

Verdrahtung
Randbedingungen
Verwandte Probleme
Verdrahtungsverfahren
Verdrahtungsreihenfolge
Globalverdrahter
Modellierung
Datenstrukturen zur Modellierung(1)
Datenstrukturen zur Modellierung(2)
Regionszuweisung
Detailverdrahtung
Kanalverdrahter: Kanalmodell
Left-Edge-Algorithmus
"Greedy" Kanalverdrahter
Beispiel Left-Edge-Algorihmus
Switchboxverdrahtung: Beispiel
Wellenfrontverdrahter
Ausbreitung einer Welle
Verbesserung des Speicher-/Laufzeitbedarfs
Bewertung: Wellenfrontverfahren
Liniensuchverfahren
Hightower- und Mikami-Verfahren
Bewertung: Liniensuchverfahren
Linienexpansionsverfahren
Bewertung: Linienexpansionsverfahren

Date of generation: 8.7.2025

Verdrahtung: Verdrahtung



Nach der Platzierung erfolgt die Verdrahtung. Für die Verdrahtung ist es wichtig zu wissen, an welchen Stellen die genauen Orte der zu verbindenden Terminals liegen. Sie liegen nach der Platzierungsphase fest und sind nicht mehr verschiebbar.

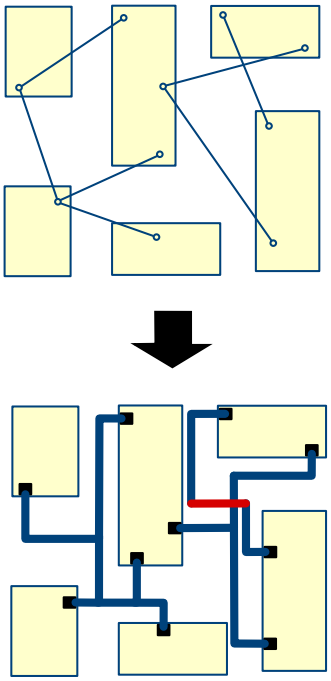
Ziel der Verdrahtung ist ein vollständiges Layout, das neben den in der Platzierungsphase festgelegten Positionen der Zellen die Geometrie aller elektrischen Verbindungen enthält. Auf der Basis dieses Layouts werden später die Masken für die Chipherstellung erstellt.

Verdrahtung: Randbedingungen

Randbedingungen

Die Verdrahtung erzeugt aus der platzierten Zellen und der Verbindungsinformationen der Netzliste ein vollständiges Layout, indem alle entsprechenden Anschlüsse gemäß den Randbedingungen miteinander verbunden werden.

- **Randbedingungen**
  - Einhaltung aller Entwurfsregeln
    - Leitungsbreiten und -abstände
    - Kontaktüberlappungen
    - Kontaktabstände
- **Ziele**
  - Minimierung der Verdrahtungslängen
  - Minimierung der Delays kritischer Netze
  - Minimierung der Verdrahtungsfläche
  - Minimierung von Durchkontaktierungen
  - Minimierung von Leitungsknicken



Das Verdrahtungsprogramm hat durch die Technologie vorgegebene Regeln, so genannte Entwurfsregeln, zu beachten. Diese Entwurfsregeln (Design Rules) definieren u.a., minimale Leitungsbreiten, minimale Abstände von Leitungen, Knicken, Vias, usw. Die Funktion der Schaltung ist nur gewährleistet, wenn die Entwurfsregeln eingehalten werden.

Die Länge aller Leitungen (Gesamtverdrahtungslänge) ist das Hauptmaß für die Bewertung der Verdrahtung. Je größer die Verdrahtungslängen sind, desto größer sind parasitäre Effekte wie Leitungslaufzeiten und desto größer ist der Platzbedarf für die Verdrahtung. Außerdem wird der Einfluss der Leitungslängen auf das Delay, das die maximale Taktfrequenz der Schaltung bestimmt, immer bedeutender.

Um den Einfluss auf das Leitungsdelay modellieren zu können, gibt es verschiedene Leitungsmodelle (C, RC, mehrfach RC, RCL, Leitung), die entsprechend der gewünschten Genauigkeit des Ergebnisses und des zulässigen Berechnungsaufwands unterschieden werden.

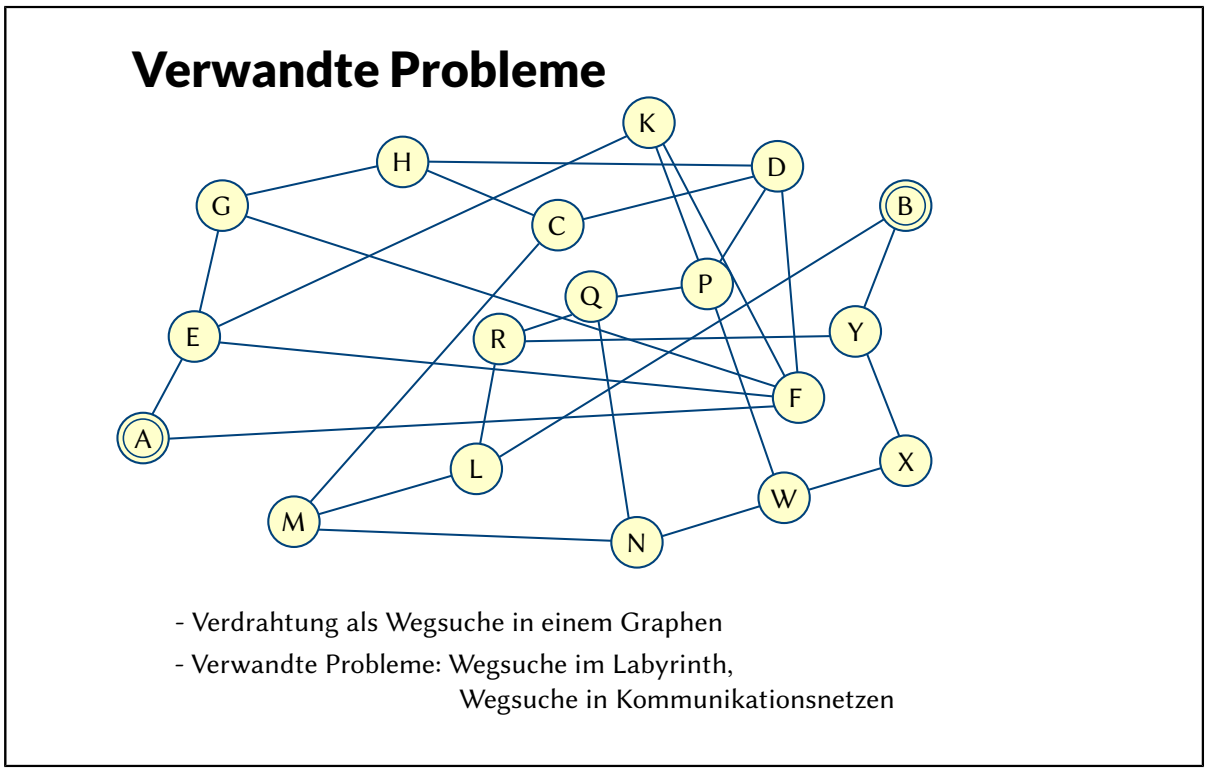
Das einfachste aber ungenaueste Leitungsmodell ist eine Kapazität zwischen der Leitung und dem Substrat (Massepotential), welche multipliziert mit dem Ausgangswiderstand des Treibers (Zeitkonstante) das Delay für eine Leitung repräsentiert.

Eine Verbesserung bringt ein RC-Modell, welches auch die Leitung mit einem Widerstand versieht. Dieses Modell ist genauer als das C-Modell und wird oft in Abschätzungen der Kosten für eine Verdrahtung verwendet.

Zusätzlich zur Substratkapazität werden bei modernen Verdrahtern auch noch Kopplungskapazitäten zu anderen Leitungen betrachtet, da auch diese einen erheblichen Einfluss auf das Delay haben können.

Die Abschätzung der Leitungseigenschaften bei der Verdrahtung muss sehr einfach sein, da aufwendige Verfahren sie stark verlangsamen würden. Genauere Verfahren und Leitungsmodelle werden in der Extraktion eingesetzt und im Kapitel "Extraktion" beschrieben.

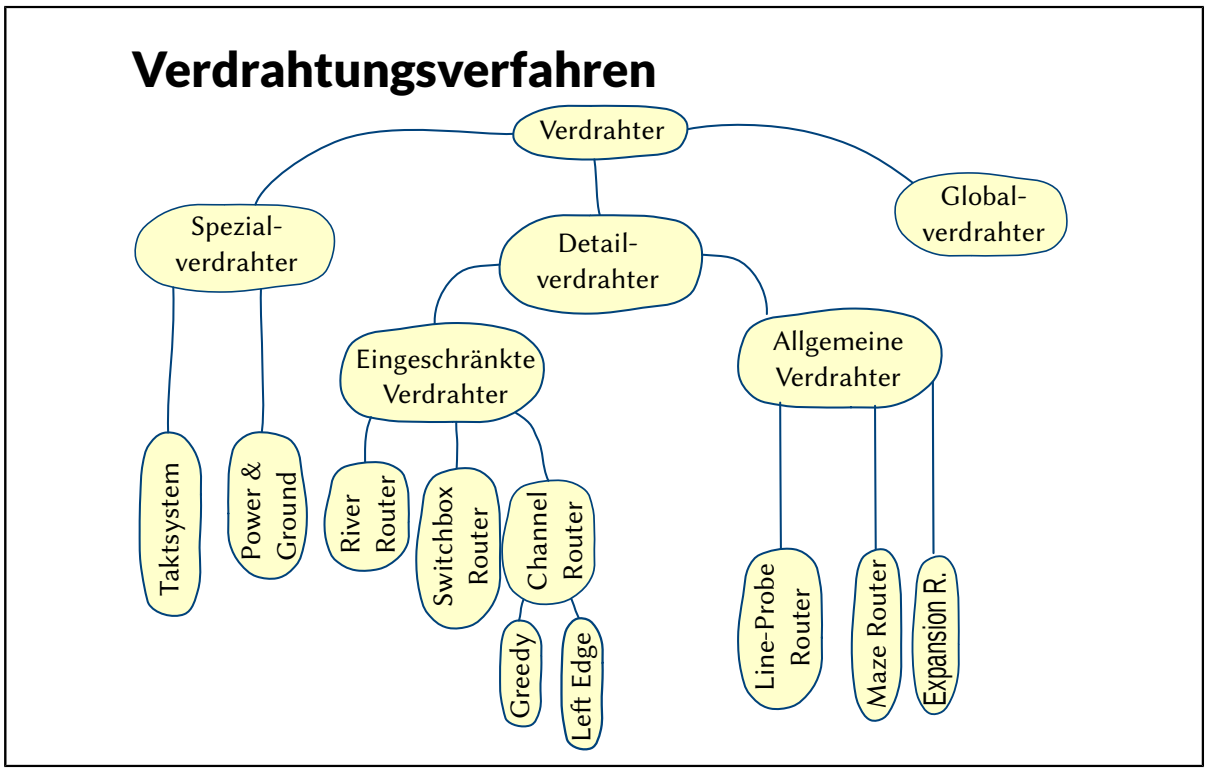
Verdrahtung: Verwandte Probleme



Der Ursprung der Wegesuche auf dem Computer geht auf das Problem zurück, einen Weg durch einen Graphen zu finden. Dieser gewichtete oder ungewichtete Graph kann z.B. ein Labyrinth oder Kommunikationsnetz darstellen.

In einer Veröffentlichung von Moore von 1959 wurde zum ersten Mal wissenschaftlich untersucht, wie algorithmisch ein Weg durch einen Graphen mit einem Rechner gefunden werden kann.

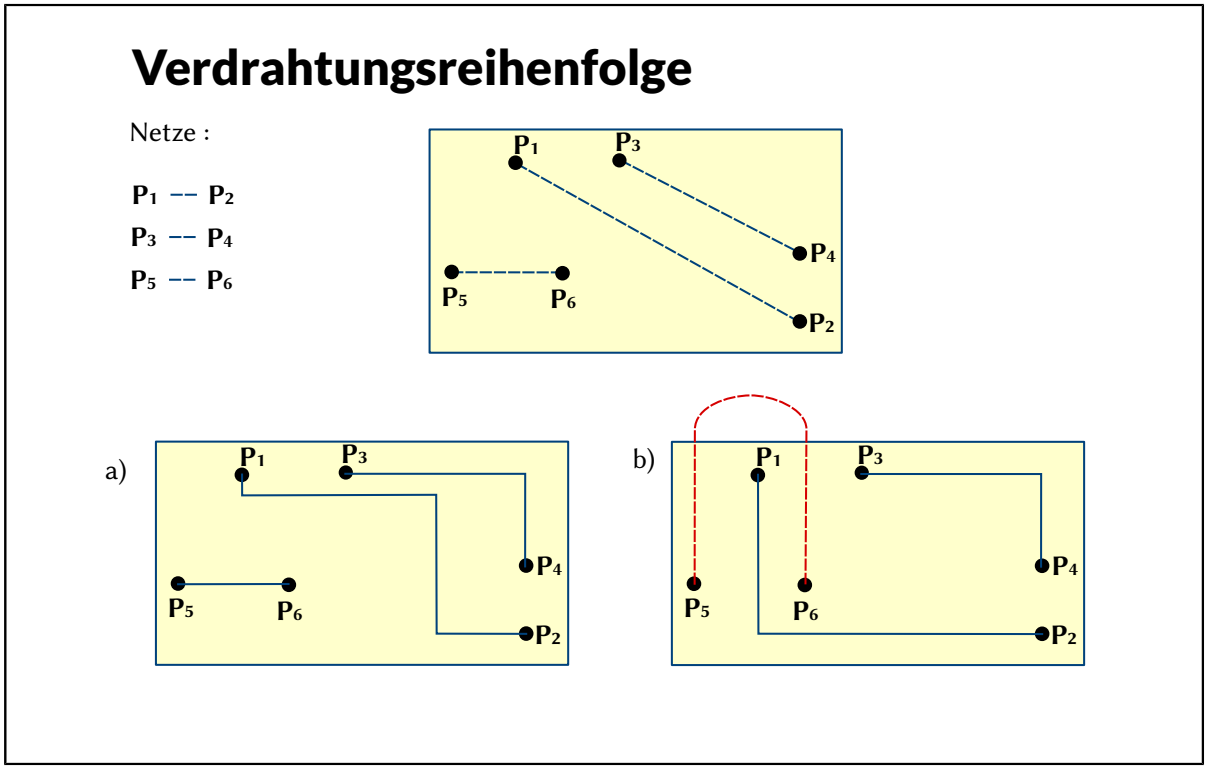
Verdrahtung: Verdrahtungsverfahren



Eine Einteilung der Verdrahtungsverfahren kann auf verschiedene Weise geschehen. Häufig werden die Verfahren in Global-, Detail- und Spezialverdrahter eingeteilt. Hier soll hauptsächlich auf die Detailverdrahter eingegangen werden.

Die detaillierten Verdrahter werden wiederum unterteilt in allgemeine Verdrahter und eingeschränkte Verdrahter. Die allgemeinen Verdrahter können beliebige Verbindungen erstellen, während die eingeschränkten Verdrahter nur spezielle Anordnungen behandeln, z.B. so genannte Kanäle oder Switchboxen. Hierauf wird in den Kapiteln Global- und Detailverdrahter eingegangen.

Verdrahtung: Verdrahtungsreihenfolge



Die Verdrahtungsreihenfolge hat einen großen Einfluss auf die Verdrahtbarkeit einer Schaltung.

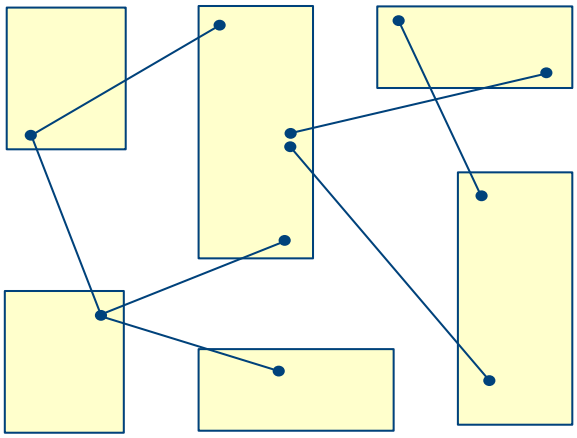
Mit einer schlecht gewählten Reihenfolge der zu verdrahtenden Netze kann eine Schaltung nicht mehr verdrahtbar sein. Aus diesem Grund gibt es Ordnungsstrategien für die Reihenfolge der Netze. So werden die Netze häufig nach ihren Funktionen sortiert, z.B. werden erst die Power- und Ground-Netze, dann die Clock-Netze und anschließend der Rest verdrahtet. Der Rest wiederum kann auch sortiert werden, z.B. nach der Länge der Leitungen.

Es ist jedoch nicht möglich, eine allgemeine Ordnungsstrategie zu formulieren, die die Verdrahtbarkeit und eine minimale Verdrahtungslänge sichern kann.

Verdrahtung: Globalverdrahter

Globalverdrahter

- Aufteilung des Verdrahtungsproblems (Divide & Conquer)
  - Globalverdrahtung (Zuweisung von Verbindungen an Regionen)
  - Detailverdrahtung (Geometrische Ausführung der Verbindungen)



Um das Verdrahtungsproblem zu vereinfachen, wird es in Global- und Detailverdrahtung aufgeteilt. Die Globalverdrahtung ist der erste Schritt bei der Verdrahtung. Sie wird für die grobe Steuerung der späteren detaillierten Verdrahtung genutzt.

Die Globalverdrahtung selbst kann in zwei Abschnitte unterteilt werden:

- Modellierung
- Regionszuweisung

Begonnen wird mit der Modellierung der Fläche für die Globalverdrahtung. Die gesamte freie Fläche wird in kleine Bereiche, so genannte Regionen, geteilt und in einem Graphen abgebildet.

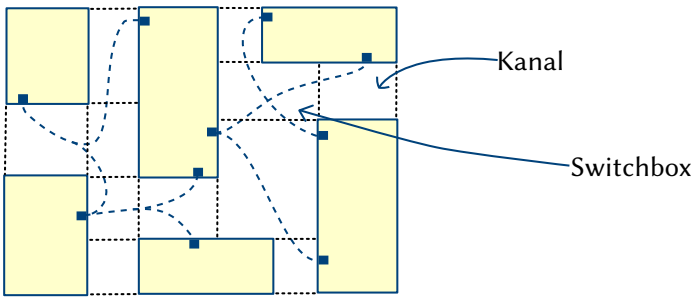
Im zweiten Schritt werden jedem Netz Regionen zugewiesen. Ausschließlich in diesen Regionen wird das Netz vom Detailverdrahter verdrahtet. Gleichzeitig mit der Zuweisung wird die Congestion (Anhäufung) der Regionen überprüft.

Das Problem der Globalverdrahtung ist mit dem Problem der Verbindungssuche in Kommunikationsnetzen verwandt.

Verdrahtung: Modellierung

Modellierung

- Aufteilung der gegebenen Verdrahtungsfläche in Teilflächen, die den Restriktionen des verwendeten Verdrahtungsalgorithmus genügen.
- Zuweisung von Verdrahtungsflächen zu jedem zu verdrahtenden Netz.
- Eine solche Teilmenge beinhaltet die Information über die grobe Netztopologie, nicht jedoch über den genauen Verlauf.



Vor Beginn der Globalverdrahtung sind die Positionen der Terminals bekannt, die verbunden werden sollen. Des weiteren ist für die Verdrahtung zwischen den Zellen Freiraum reserviert. Damit in dem anschließenden Detailverdrahtungsschritt die spezialisierten Verdrahter die Verdrahtung durchführen können, müssen die Freiflächen in kleine Rechtecke aufgeteilt werden, die später bei der Detailverdrahtung einzeln bearbeitet werden. Durch die Zerlegung entstehen Rechtecke zwischen jeweils zwei Zellen, die so genannten Kanäle. Dort liegen die Anschlussterminals der Zellen immer nur an gegenüberliegenden Seiten. Die Kanäle werden später von einem Kanalverdrahter bearbeitet.

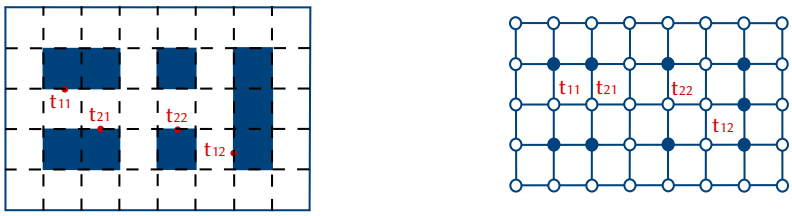
Die Rechtecke, die mehrere Känale miteinander verbinden, werden als Switchboxen bezeichnet. Switchboxen können maximal vier Kanäle miteinander verbinden.

Der Vorteil einer Aufteilung in Kanäle und Switchboxen ist, dass die Kanäle in beliebiger Reihenfolge ohne späteres "rip-up-and-reroute" (Aufreißen und Wiederverdrahten) abgearbeitet werden können, um anschleißend die Switchboxen zu verdrahten. Würden keine Switchboxen sondern ausschließlich Kanäle verwendet, müssten diese Kanäle aufgrund ihrer Abhängigkeiten in bestimmten Reihenfolgen abgearbeitet werden, da sonst Widersprüche auftreten können. Diese Reihenfolge ist nur bei "Slicing Floorplans" einfach zu bestimmen und wird umgekehrt zur Schnittreihenfolge im Floorplan festgelegt.

Verdrahtung: Datenstrukturen zur Modellierung(1)

Datenstrukturen zur Modellierung (1)

- Grid Graph



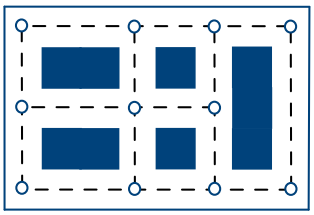
Die Datenstrukturen, in die Kanäle und Switchboxen mit deren Kanalkapazitäten eingetragen werden, sind häufig Graphen. Die Kanalkapazitäten ergeben sich aus der Anzahl der Layer für Spuren und der Anzahl der Spuren selbst, die für die Verdrahtung zur Verfügung stehen.

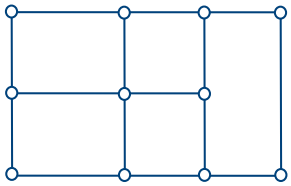
Das einfachste Modell ist der Grid Graph. Er unterteilt die Layoutfläche in gleich große Quadrate, welche durch Knoten darstellt werden. Die Nachbarschaft der Quadrate wird durch Kanten zwischen den Knoten beschrieben. Die Größe der Quadrate wird so gewählt, dass die Kante immer eine Kanalkapazität von eins, also einer Leitung, besitzen.

Verdrahtung: Datenstrukturen zur Modellierung(2)

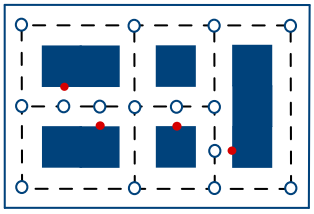
Datenstrukturen zur Modellierung (2)

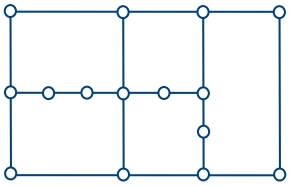
- Channel Intersection Graph





- Extended Channel Intersection Graph

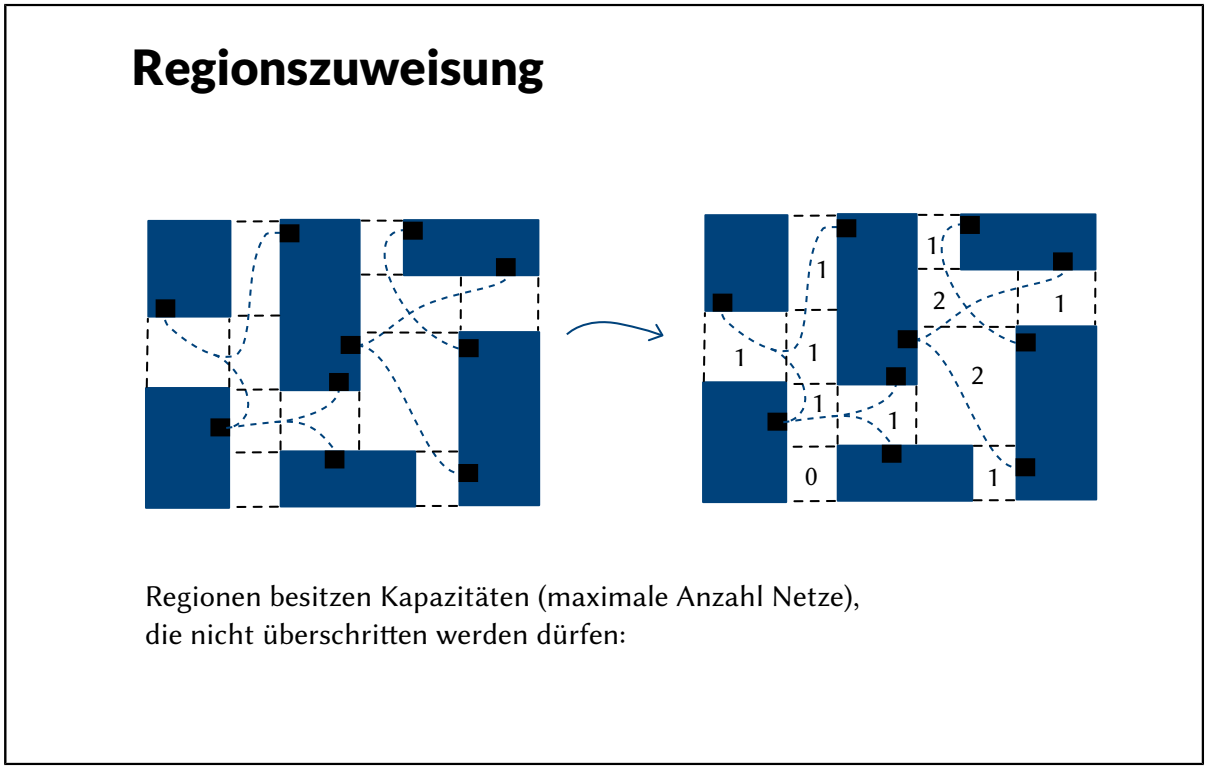




Das genaueste und allgemeinste Modell für die Globalverdrahtung ist der Channel Intersection Graph. Die Knoten in diesem Modell stellen die Kreuzung zwischen mehreren Kanälen dar. Kanäle wiederum werden durch Kanten repräsentiert, die Attribute für die Kanallänge und -kapazität tragen.

Erweitert werden kann dieses Modell um die Terminalpositionen, die durch zusätzlich eingefügte Knoten repräsentiert werden.(Extended Channel Intersection Graph)

Verdrahtung: Regionszuweisung



Die Verfahren, die bei der Regionszuweisung benutzt werden, unterscheiden sich nicht grundlegend von denen der Detailverdrahtung, sie werden nur in einer höheren Abstraktion verwendet. Somit wird während der Globalverdrahtung nur der grobe Verlauf der Verdrahtung bestimmt und dieser nicht bis ins Detail festgelegt.

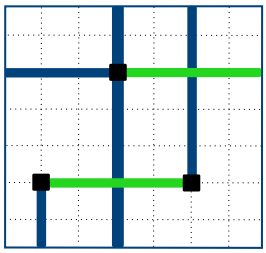
Bei der Regionszuweisung werden die Leitungen den Kanälen und Switchboxen zugeordnet. Ist die Kapazität eines Kanals erschöpft, kann keine weitere Leitung in diesen Kanal gelegt werden. Somit muss die Verdrahtung über einen anderen Weg durchgeführt werden.

Im Bild ist eine beispielhafte Zerlegung und Regionszuweisung dargestellt. Dabei wurde jeder Region eine Zahl zugewiesen, die die Zahl der im dieser Region zu verdrahtenden Netze angibt.

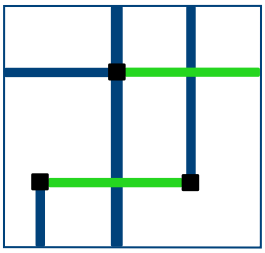
Verdrahtung: Detailverdrahtung

Detailverdrahtung

- Geometrische Ausführung der Verbindungen
- Problem ist Wegesuche
- Hier behandelt:
  - Kanalverdrahter
  - rasterfreie und rasterbasierte Punkt-zu-Punkt-Verdrahter



rasterbasiert



rasterfrei

Im Rahmen der Detailverdrahtung wird die geometrische Ausführung der Verbindungen festgelegt. Das zu lösende Problem ist das der Wegesuche, wie es in ähnlicher Form bei der Suche eines Weges im Labyrinth auftritt. Von der Vielzahl der möglichen Vorgehensweisen werden hier nur die Grundformen behandelt, die Kanalverdrahtung sowie die Punkt-zu-Punkt-Verdrahtung, die den allgemeinen Fall darstellt.

Bei der Punkt-zu-Punkt-Verdrahtung wird häufig zwischen rasterfreien und rasterbasierten Verfahren unterschieden.

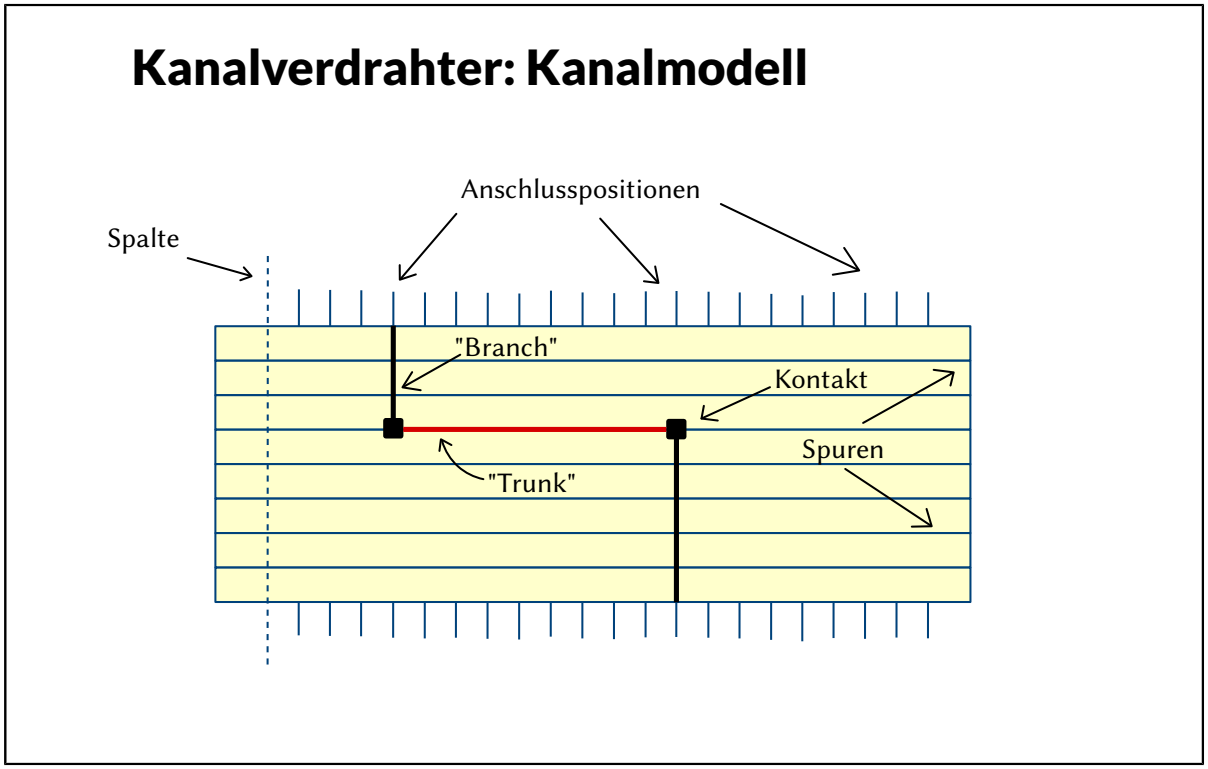
Bei den rasterbasierten Verfahren, wird ein Raster über die gesamte Layoutfläche gelegt. Die Rastergröße wird so gewählt, dass die von den Entwurfsregeln vorgegebenen Abstände eingehalten werden.

Die Rastergröße ergibt sich aus dem Abstand und der Breite der Leitungen. Bei einem Mehrlagenprozess werden die minimalen Abstände des ungünstigsten Layers gewählt, da die Raster für Durchkontaktierungen auf allen Ebenen gleich sein müssen.

Durch die Rasterung ergeben sich Einschränkungen. Der Abstand und die Breite der Leitungen werden immer vom ungünstigsten Layer bestimmt, somit wird Fläche in den anderen Layern nicht vollständig genutzt. Die Rasterung der Layoutfläche ist zudem nur für eine Leitungsbreite optimal. Wird mit einer anderen Leitungsbreite verdrahtet, müsste die Layoutfläche erneut gerastert werden.

Rasterfreie Modelle haben diese Einschränkungen nicht. Sie können die gesamte Layoutfläche nutzen und sind nicht an Wege gebunden, die durch ein Raster vorbestimmt werden.

Verdrahtung: Kanalverdrahter: Kanalmodell



Ein Kanal ist ein Rechteck, das an zwei gegenüberliegende Seiten Terminals aufweist. Diese Terminals gehören zu Netzen, die zu verdrahten sind.

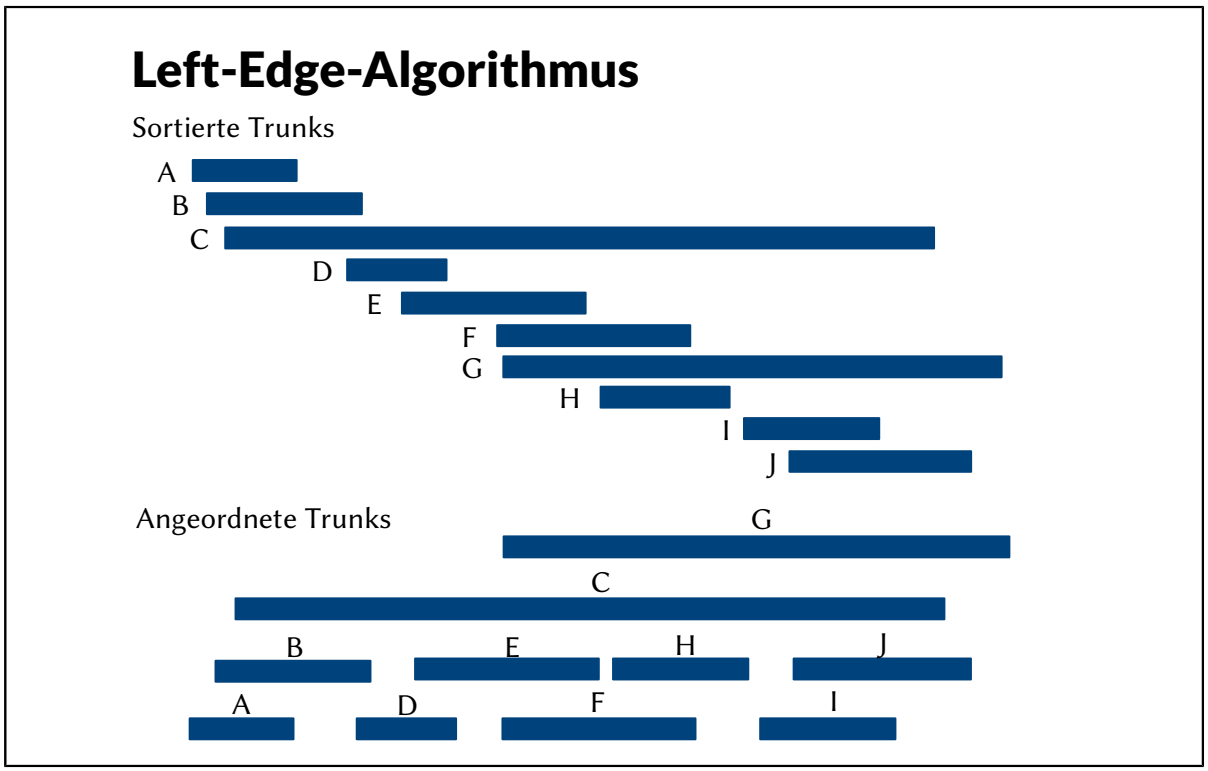
Es wird angenommen, dass die Seiten mit den Terminals oben und unten in dem Rechteck liegen. An den Seiten können Netze in den Kanal eintreten oder den Kanal verlassen. Sie werden dort dem Switchboxverdrahter übergeben.

Üblicherweise stehen in einem Kanal für die Verdrahtung vertikale Verbindungen (Branch) und horizontale Verbindungen (Trunk) zur Verfügung, die jeweils auf eigenen Metalllagen verdrahtet werden .

Die Kanalkapazität berechnet sich aus der Anzahl der Spuren, in die Trunks gelegt werden und die Anzahl an Metalllagen, die Trunks aufnehmen können.

Die Aufgabe des Kanalverdrahters ist es, mit möglichst wenig Spuren die Verdrahtung innerhalb des Kanals vollständig durchzuführen. Kanäle haben also grundsätzlich keine vorgegebene Höhe, sondern diese soll durch die benötigte Spurzahl minimiert werden.

Verdrahtung: Left-Edge-Algorithmus



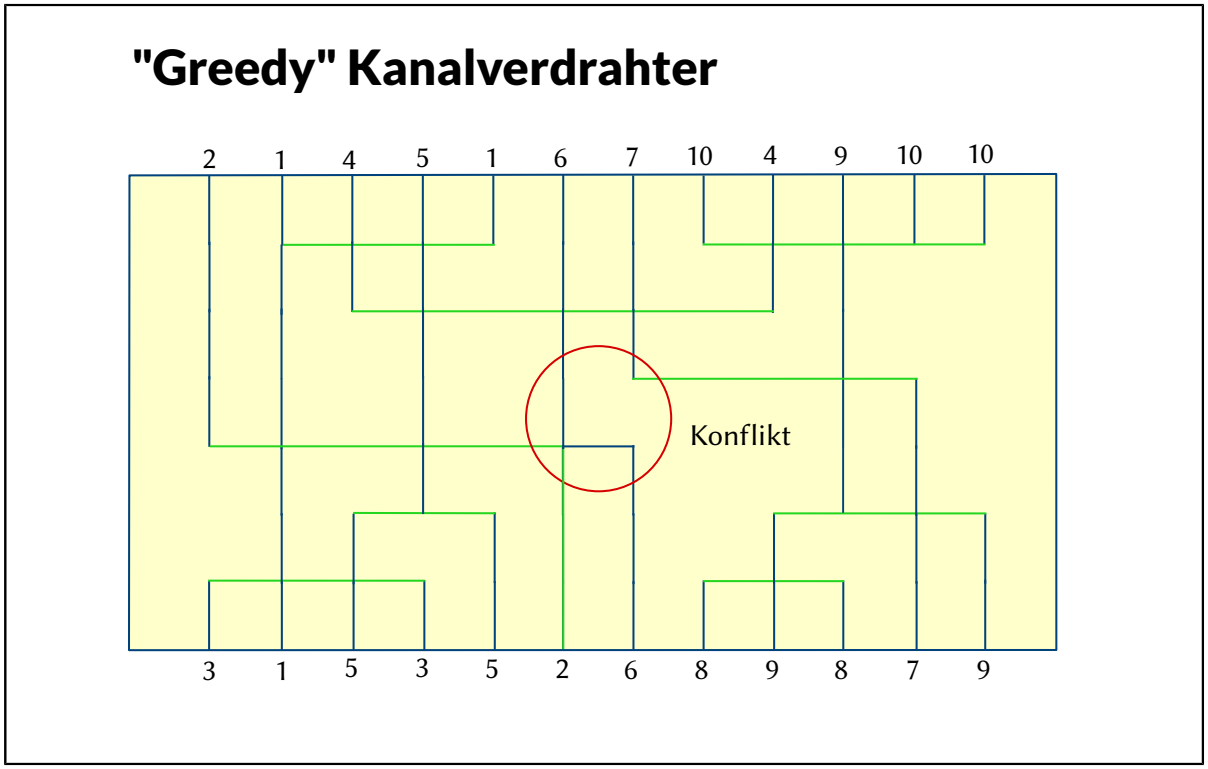
Da nicht für jeden Trunk eine neue Spur verwendet werden muss, können mehrere Trunks auf eine Spur gelegt werden. Hierzu dient der Left-Edge-Algorithmus.

Trunks beginnen und enden an den Positionen der entsprechenden Terminals. Sie werden nach ihren linken Anfangspositionen (left edge) sortiert. Anschließend wird der erste Trunk in der sortierten Liste entfernt und der ersten Spur zugeordnet. Um die Spur zu füllen, werden die Anfangspunkte der übrigen Trunks der Liste mit dem Endpunkt des Trunks aus der Spur verglichen. Liegt der Anfangspunkt rechts vom Endpunkt, so wird dieser Trunk hinter den vorherigen in die Spur eingefügt und aus der Liste entfernt. Dieses wird solange durchgeführt, bis die Spur gefüllt ist oder kein Trunk mehr in der Liste enthalten ist, der in die aktuelle Spur eingefügt werden kann.

Nun wird mit dem ersten der verbliebenen Trunks aus der Liste eine neue Spur angelegt und das Verfahren beginnt von vorn.

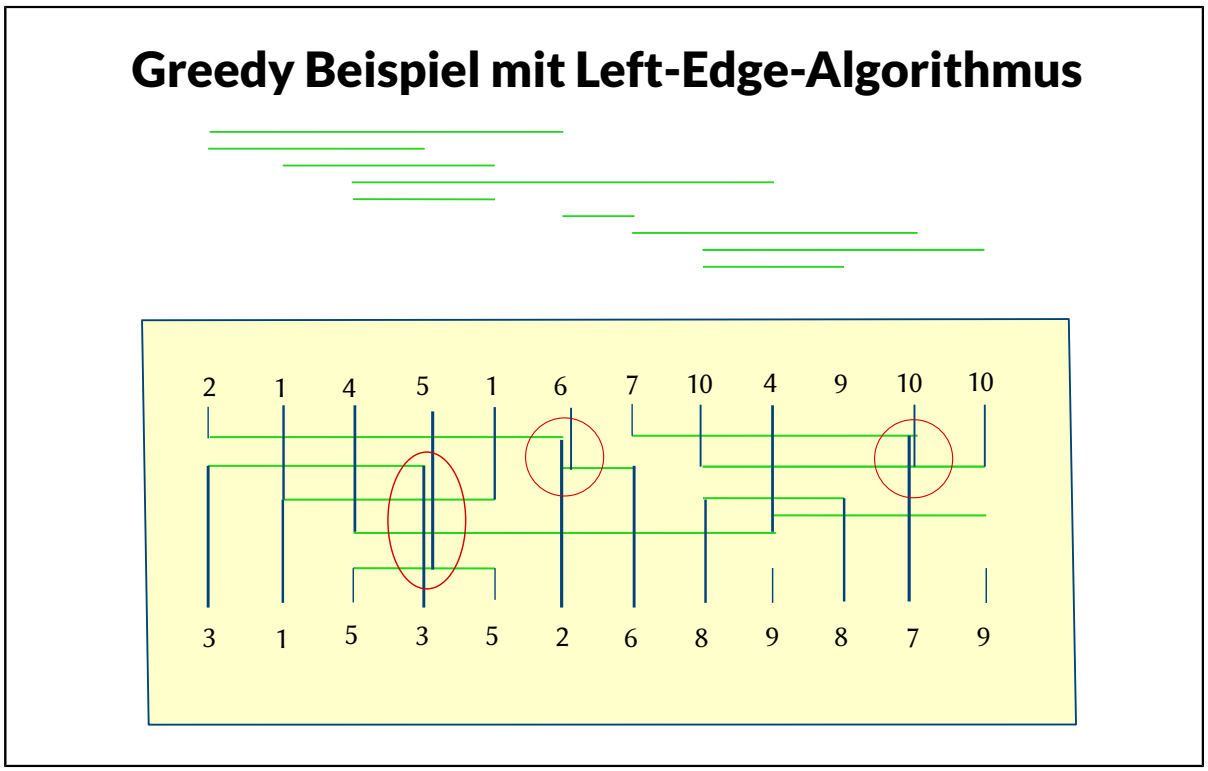
Im dargestellten Beispiel werden für 10 Trunks 4 Spuren benötigt. Der Left-Edge-Algorithmus liefert trotz seiner Einfachheit für die Minimierung der Spurzahl eine optimale Lösung, sofern es nicht zu Konflikten kommt, wenn Terminals an gleichen Positionen angeschlossen werden müssen.

Verdrahtung: "Greedy" Kanalverdrahter



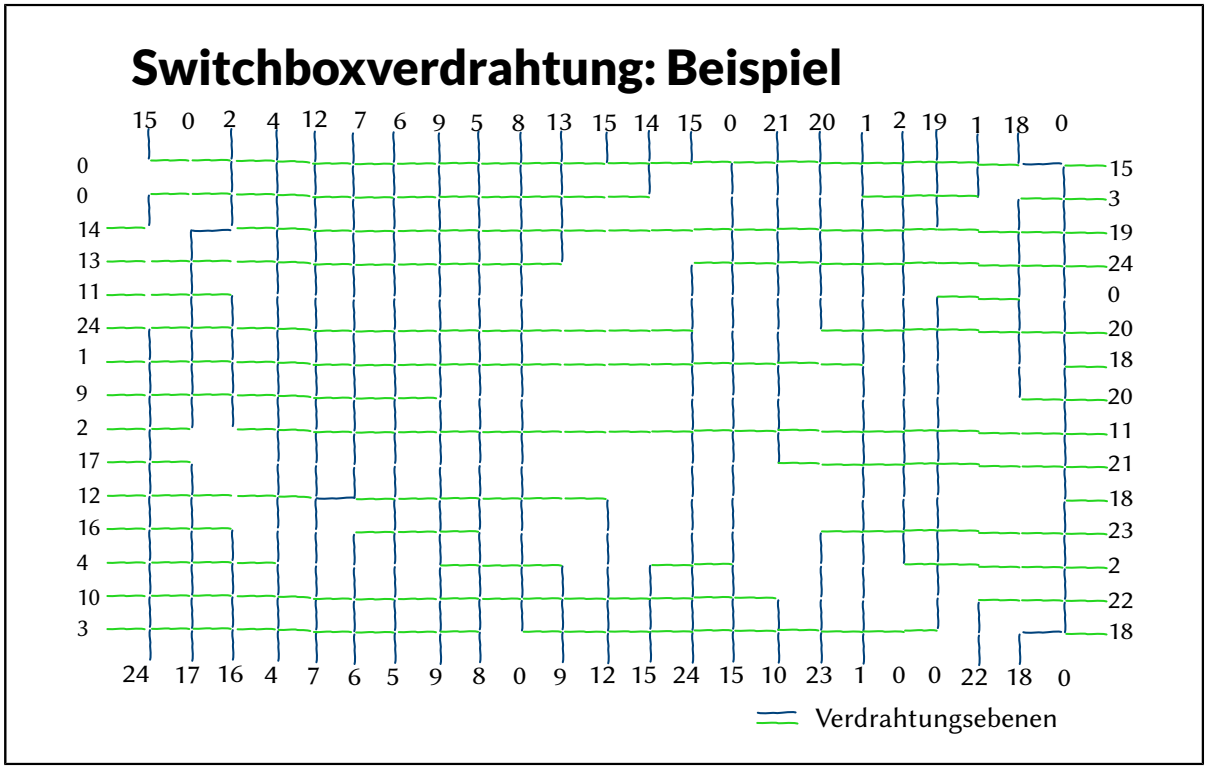
Die Abbildung zeigt beispielhaft das Vorgehen eines zum Left-Edge-Algorithmus alternativen Kanalverdrahters, der sich schrittweise von unten und oben in den Kanal vorarbeitet und die jeweils möglichen Verbindungen legt bzw. vorbereitet. Er arbeitet grundsätzlich “greedy”, d.h. er nimmt einmal getroffene Entscheidungen nicht zurück. Wie in der Mitte erkennbar, kann dies jedoch zu Konflikten führen, die durch Rücknahme eines Schritts gelöst werden müssen.

Verdrahtung: Beispiel Left-Edge-Algorihmus



Es wird eine beispielhafte Verdrahtung mit Hilfe des Left-Edge-Algorithmus berechnet. Allerdings gibt es einige Konflikte, die nachträglich aufgelöst werden müssen. Das könnte wieder weitere Spuren erfordern.

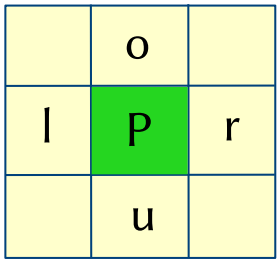
Verdrahtung: Switchboxverdrahtung: Beispiel



Die Abbildung zeigt beispielhaft das Ergebniss einer Switchboxverdrahtung mit zwei Verdrahtungsebenen. Dabei sind die zu verbindenden Anschlüsse an den Rändern mit ihren Netznummern gekennzeichnet.

Verdrahtung: Wellenfrontverdrahter

Wellenfrontverdrahter



o, r, l und u sind direkte Nachbarn des Punktes P

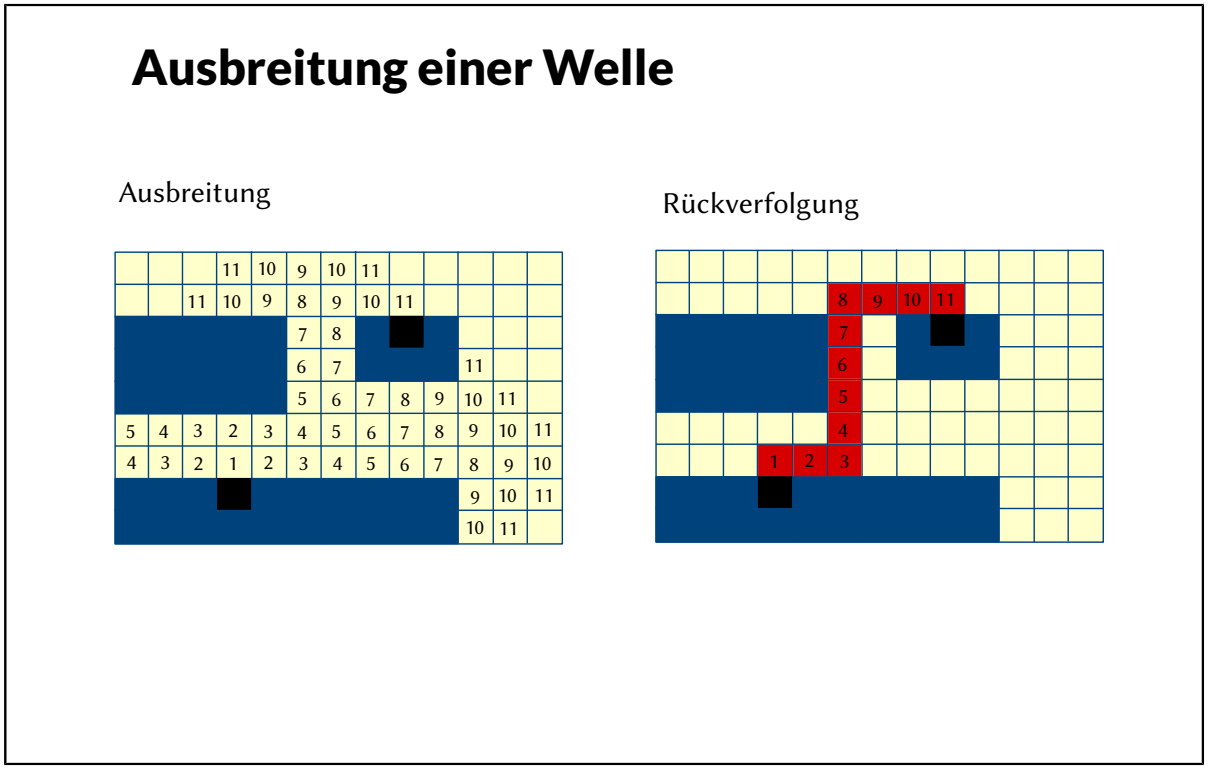
				4						
			4	3	4					
		4	3	2	3	4				
	4	3	2	1	2	3	4			
4	3	2	1	0	1	2	3	4		
	4	3	2	1	2	3	4			
		4	3	2	3	4				
			4	3	4					
				4						

Der Wellenfront-Verdrahter ist ein rasterbasierter Punkt-zu-Punkt-Verdrahter. Eine anschauliche Vorstellung des Verfahrens liefert eine Welle, die sich vom Startpunkt der Verdrahtung aus in alle Richtungen ausbreitet, wie durch einen Stein, der ins Wasser geworfen wird. Die Welle läuft an Hindernissen im Layout vorbei und erreicht, wenn es einen Weg gibt, in jedem Falle das Ziel. Es handelt sich um eine Breitensuche.

Das Verfahren ist auch unter der Bezeichnung“Maze-Router” oder Lee-Algorithmus bekannt.

Der Startwert wird mit dem Wert 1 initialisiert. Die direkten Nacxhbarn (rechts, links, untzer- und oberhalb) erhalten den um 1 erhöhten Wert. Dieser Vorgang wird solange fortgesetzt, bis der Zielpunkt erreicht wurde.

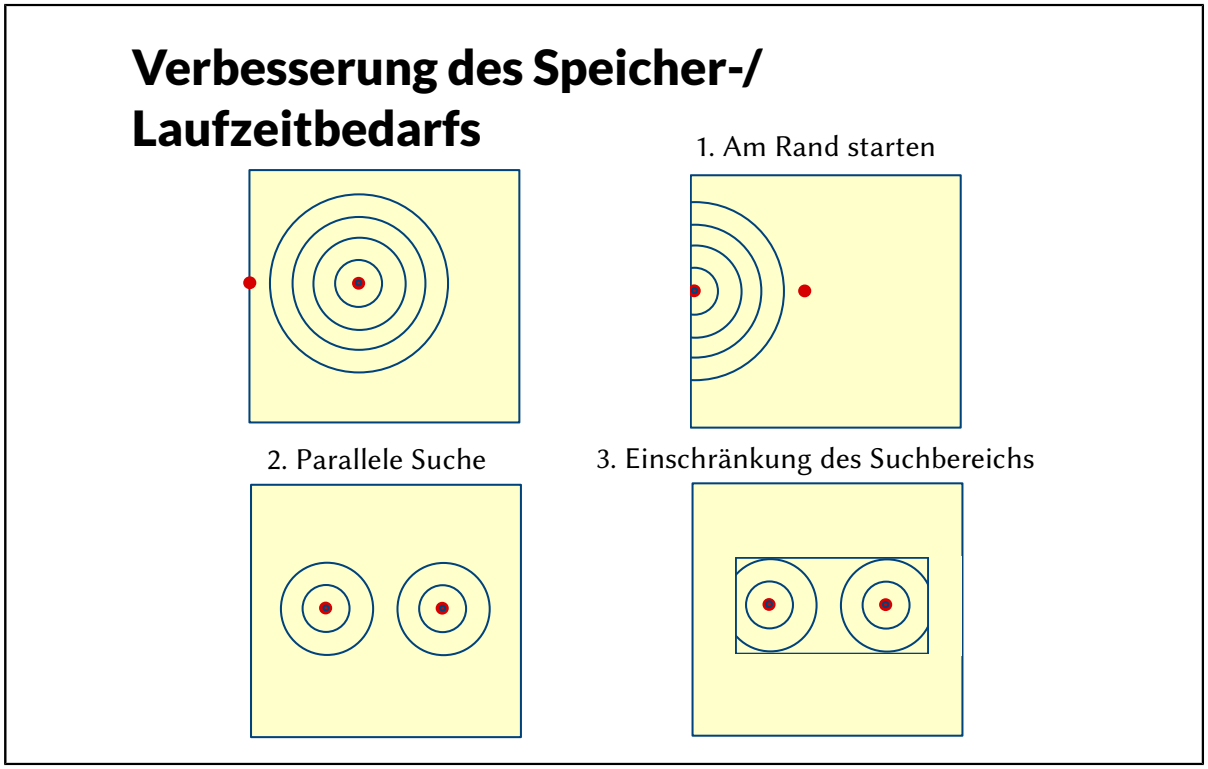
Verdrahtung: Ausbreitung einer Welle



Wenn das Ziel erreicht wurde, hat sich die Welle zwischen Start und Ziel ausgebreitet. Damit ist bekannt, dass es eine Lösung gibt und wie weit Start und Ziel auf kürzestem Wege auseinander liegen.

Unbekannt ist jedoch der Weg. Er kann durch Rückverfolgung gefunden werden. Die Rückverfolgung startet am Ziel und geht den Weg zurück, indem das jeweils folgende Feld mit dem nächst kleineren Wert ausgewählt wird. Dieser Weg muss nicht eindeutig sein, da es mehrere kürzeste Wege geben kann.

Verdrahtung: Verbesserung des Speicher-/Laufzeitbedarfs



Das beschriebene Verfahren hat einen hohen Speicher- und Laufzeitbedarf, da jedes Rasterfeld einen ganzzahligen Wert trägt und im Worst Case alle Felder mit einem Wert beschrieben werden müssen.

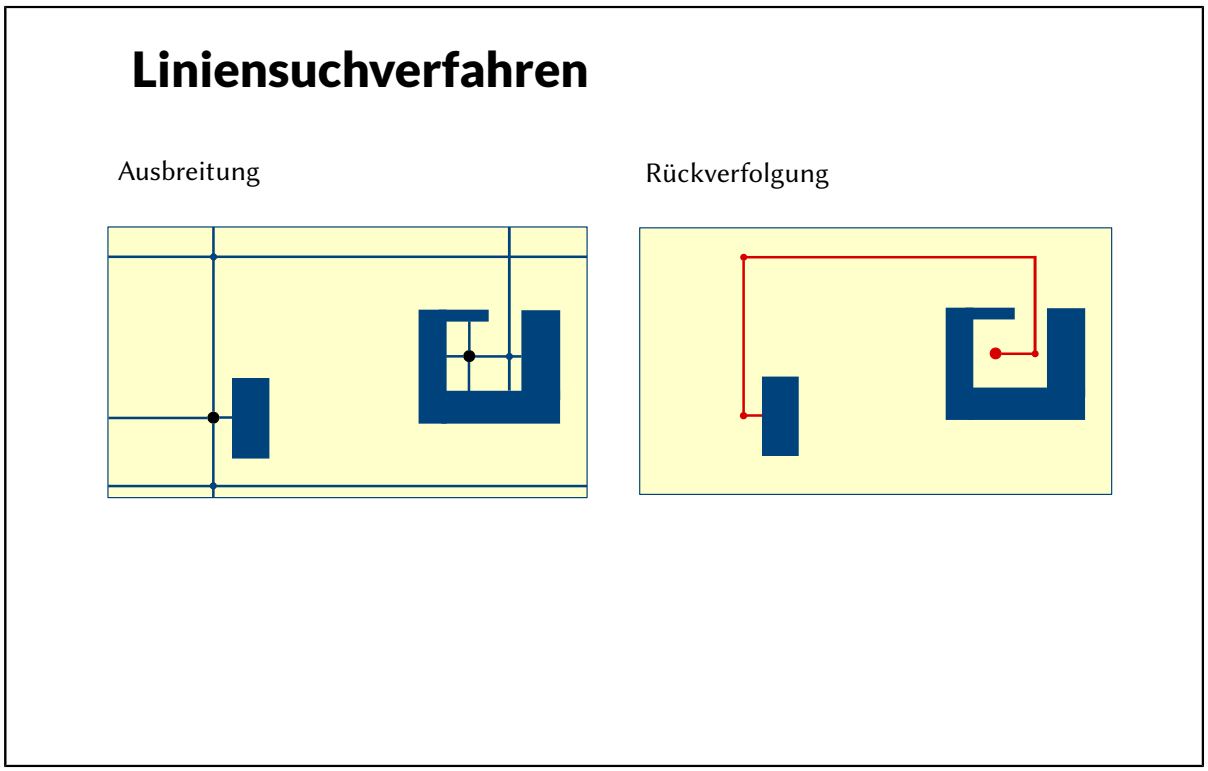
Kommen mehrere Punkte als Startpunkte in Frage, kann die Laufzeit des Verfahrens verbessert werden, indem der dem Rand am nähesten liegende Punkt als Startpunkt ausgewählt wird. Ein simultanes Aufbauen der Welle von Start und Ziel aus verbessert ebenfalls die Laufzeit durch die Verringerung des Flächenbedarfes des Suchbereiches. Treffen sich beide Wellen, so ist ein Weg gefunden.

Eine andere Möglichkeit zur Rechenzeit- und Speicherersparnis besteht darin, die Fläche, in der nach einem Weg gesucht werden darf, auf ein heuristisch bestimmtes Maß einzuschränken.

Bewertung: Wellenfrontverfahren

- **Vorteile**
  - Falls eine Verbindung zwischen zwei Punkten möglich ist, wird sie gefunden.
  - Die gefundene Verbindung ist in jedem Falle die kürzeste mögliche Verbindung zwischen den zwei Punkten.
  - Der Algorithmus ist einfach zu implementieren.
- **Nachteile**
  - Es wird jeweils nur eine Verbindung auf einmal gezogen.
  - Der Algorithmus neigt dazu, mit den früh angelegten Verbindungen Wege für spätere Verbindungen zu versperren, und erreicht damit nicht unbedingt 100% Verdrahtungsrate.
  - Das wiederholt auszuführende Markierungsverfahren resultiert in sehr hohen Laufzeiten.
  - Die Layoutpräsentation führt zu hohem Speicherplatzbedarf.

Verdrahtung: Liniensuchverfahren



Liniensuchverfahren suchen ähnlich den Wellenfrontverdrahtern einen Pfad vom Start zum Ziel. Jedoch unterscheiden sie sich in der Erstellung der Teillösungen. Während die Wellenfrontverdrahter für neue Teillösungen einen Rasterpunkt hinzufügen, expandieren die Liniensucherverfahren rasterlos mit ganzen Linien.

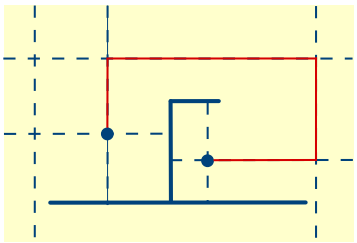
Das Hightower- und das Mikami-Verfahren starten mit der Erzeugung der Versuchslinien im Start- und Zielpunkt, indem bei beiden Verfahren im Start- und Zielpunkt jeweils eine horizontale und eine vertikale Linie erzeugt wird. Diese Versuchslinien werden solange ausgedehnt, bis sie auf ein Hindernis oder die Grenze der Verdrahtungsfläche stoßen. Schneiden sich die Start- und Zielversuchslinien, so ist ein Weg gefunden und die Suche wird abgebrochen. Schneiden sich die Start- und Zielversuchslinien nicht, werden iterativ senkrecht zu den letzten Versuchlinien neue erzeugt.

Verdrahtung: Hightower- und Mikami-Verfahren

Hightower- und Mikami-Verfahren

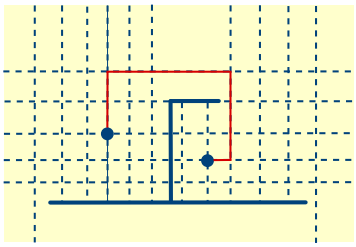
Hightower-Verfahren:

Es wird genau eine Versuchslinie erzeugt.



Mikami-Verfahren:

Es werden mehrere Versuchslinien (in einem festen Raster) erzeugt.



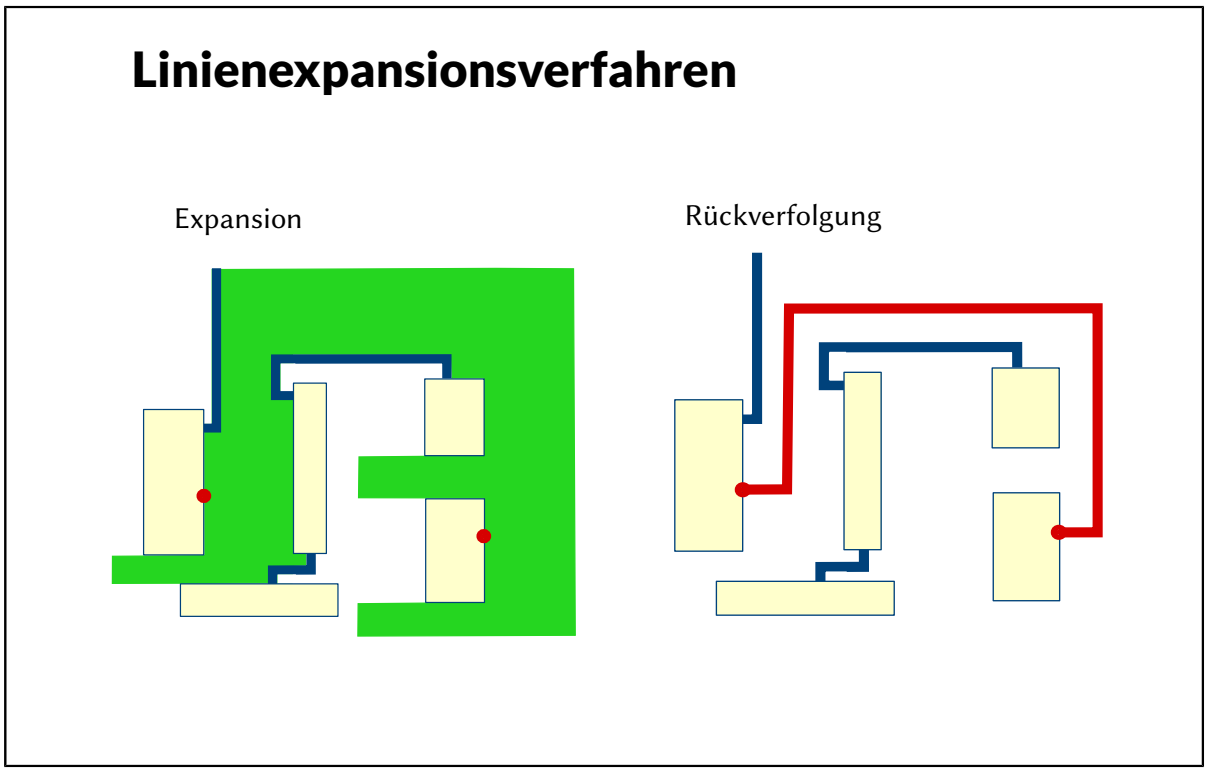
Das Hightower-Verfahren erzeugt die Versuchslinien anhand von charakteristischen Punkten von Hindernissen, um diese zu umgehen. Dieses Vorgehen ist stark heuristisch und findet somit nicht immer eine Lösung, auch wenn es eine gibt.

Das Mikami-Verfahren erzeugt neue Versuchslinien nach einem festen Raster, das sich nicht nach der Umgebung richtet. Ist die Größe des Rasters fein genug gewählt, wird die Lösung in jedem Fall gefunden, wenn es eine Lösung gibt.

Bewertung: Liniensuchverfahren

- **Vorteile**
  - Laufzeit und Speicherbedarf sind geringer als beim Wellenfrontverfahren.
  - Die Verfahren sind einfach zu implementieren.
  - Das Mikami-Verfahren findet eine Lösung, falls eine existiert.
  - Das Verfahren ist besonders für Verdrahtungen mit relativ wenigen Hindernissen auf der Verdrahtungsfläche gut geeignet.
- **Nachteile**
  - Der Algorithmus neigt dazu, mit den früh angelegten Verbindungen Wege für spätere Verbindungen zu versperren, und erreicht damit nicht unbedingt 100% Verdrahtungsrate.
  - Das Mikami-Verfahren findet zwar recht kurze Verbindungen, jedoch auf Kosten vieler Fluchtpunkte.
  - Das Hightower-Verfahren kann nicht garantieren, eine Lösung zu finden, selbst wenn eine existiert.

Verdrahtung: Linienexpansionsverfahren



Das Linienexpansionsverfahren ist ein rasterfreies Verfahren, das als Kombination aus Wellenfrontverdrahter und Liniensuchverfahren verstanden werden kann. Im Unterschied zu den Wellenfrontverdrahtern und Liniensuchverfahren werden keine konkreten Wege gefunden, sondern nur Bereiche, in denen die Leitungen verlegt werden können.

Das Verfahren expandiert senkrecht zu Expansionslinien Expansionsflächen. Die Expansionsflächen werden durch die umgebenden Hindernisse begrenzt. Kanten, die keine Hindernisse darstellen, sind aktive Linien, welche in einem folgenden Schritt wieder expandiert werden können. Dieses Vorgehen wird iterativ weitergeführt, bis sich die Expansionsgebiete von Start- und Zielpfad überschneiden, womit eine Lösung gefunden wurde. Ein konkreter Weg für die Leitung wird in einem anschließenden Rückverfolgungsschritt gefunden.

Bewertung: Linienexpansionsverfahren

- Vorteile

- Die Geschwindigkeit des Algorithmus liegt zwischen der des Wellenfront- und der des Liniensuch-Verfahrens.
- Der beanspruchte Speicherbedarf ist sehr gering.
- Falls eine Verbindung existiert, wird sie auch gefunden.

- Nachteile

- Das Verfahren ermittelt keine konkrete Leitungsgeometrie, sondern eine Folge von Verdrahtungsgebieten, in denen in einem zweiten Schritt die eigentliche Leitung verlegt werden muss.
- Es ist eher für kleine Verdrahtungsprobleme geeignet, z.B. der interaktiven Analogverdrahtung.

Electronic Design Automation (EDA)

Design Rule Check

Design Rule Check

Entwurfsregeln

Geometrierestriktionen

Prozessbedingte Abweichungen

Justiergenauigkeit

Regelprüfung

DRC-Beispiel 1

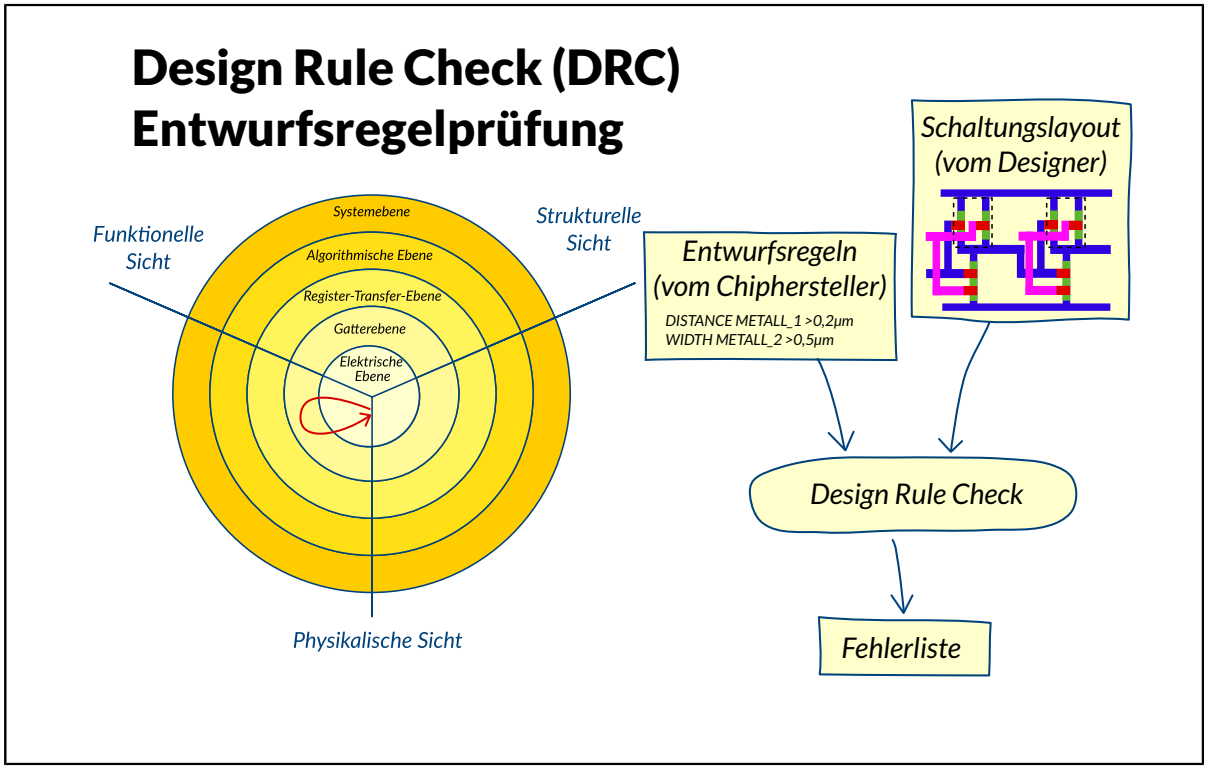
DRC-Beispiel 2

Scheinfehler

Electrical Rule Check

Elektrische Entwurfsregeln

Design Rule Check: Design Rule Check



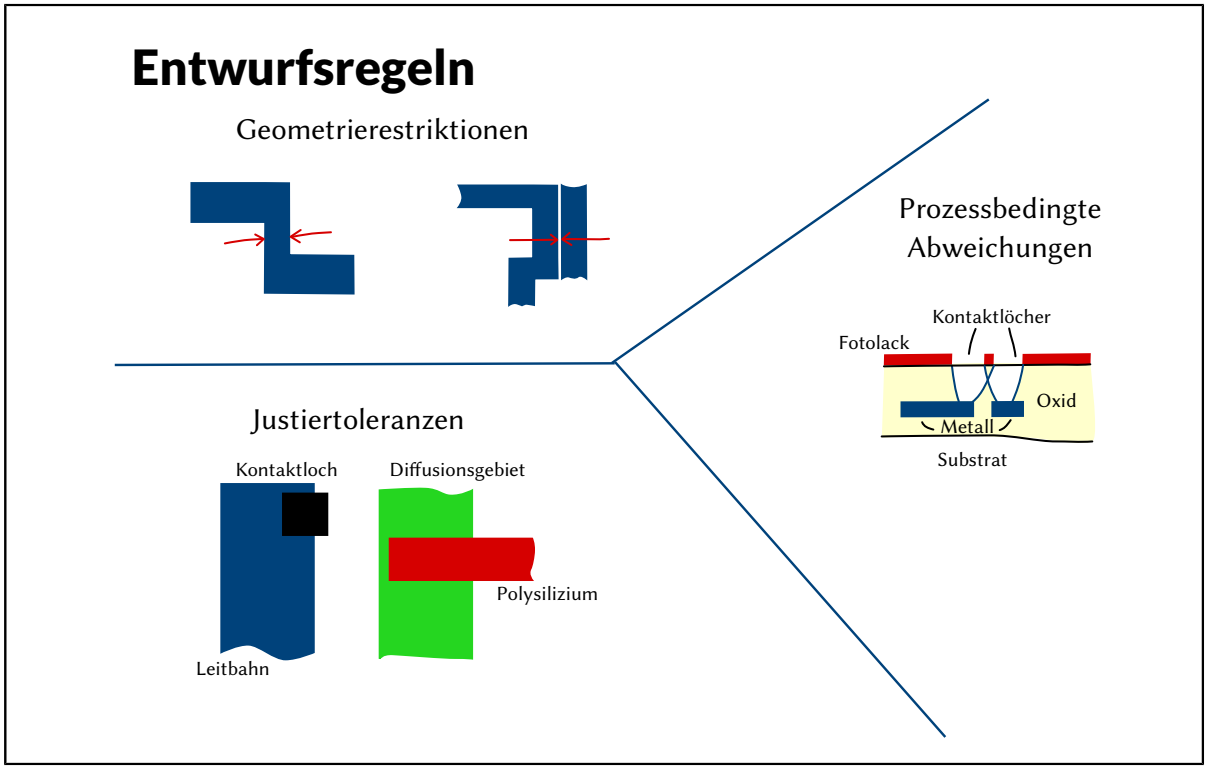
Auch bei den erfahrensten Schaltungsentwicklern können bei der Layouterzeugung Fehler auftreten, die eine volle Funktionsfähigkeit der Schaltung beeinträchtigen. Zudem besitzt jeder Chiphersteller unterschiedliche Restriktionen, die für den jeweiligen Technologieprozess gelten. Diese Einschränkungen gewährleisten die zuverlässige Funktionalität der gefertigten Schaltung. Daher müssen vom Chiphersteller Bedingungen, die im Layout gelten sollen, dem Schaltungsentwickler vorgegeben werden. Bei Fehlern muss ein Redesign vorgenommen werden, das im hierarchischen Entwurf einen Rücksprung auf eine höhere Ebene entspricht. Obwohl das einen erheblichen Zeitaufwand bedeutet, ist diese Vorgehensweise weitaus kostengünstiger, als die Fertigung einer nicht funktionsfähigen Schaltung.

Die Verifikation des Schaltungslayouts wird durch geeignete Prüfprogramme automatisch durchgeführt, wobei die Regeln für die Prüfung vom Chiphersteller vorgegeben sind, aber vom Schaltungsentwickler erweitert werden können. Vor der Einführung von Software-Werkzeugen etwa Mitte der siebziger Jahre wurden ausschließlich visuelle Kontrollen durchgeführt. Diese waren jedoch sehr langsam und fehlerträchtig und mit wachsender Komplexität auch für den Menschen kaum zumutbar. Bei der heutigen Anzahl von Regeln in Verbindung mit den vielen beteiligten Layern ist eine automatisierte Prüfung unumgänglich.

Die Layoutprüfung ist ein technologieabhängiger Vorgang. Während die technologiespezifische Information anfangs unmittelbar in die Programme hinein codiert wurde, gelang es später, die Prüffunktionen so zu verallgemeinern, dass die Technologieabhängigkeit in den Bereich der Programmparameter und Eingabedaten verlagert werden konnte. Dazu dienen heute so genannte Technologiebeschreibungssprachen. Mit ihrer Hilfe lassen sich die Grundelemente eines Prozesses (Ebenen, Prozessparameter), seine geometrischen Beschreibungselemente (geometrische Elemente, Standardbreiten, etc.) und vollständige Prüfsequenzen beschreiben.

Jeder Chiphersteller gibt für den Herstellungsprozess eine Reihe von geometrischen Restriktionen an. In den so genannten Entwurfsregeln (Design Rules) werden z.B. Mindestabstände , Mindestbreiten , Überlappungen und Innenlagen der einzelnen Layer und Layerkombinationen festgelegt. Entwurfsregeln sind neben dem Layout die Eingangsparameter der Entwurfsregelprüfung. Regelverletzungen und eine Fehlerliste stellen die Ausgabe dar.

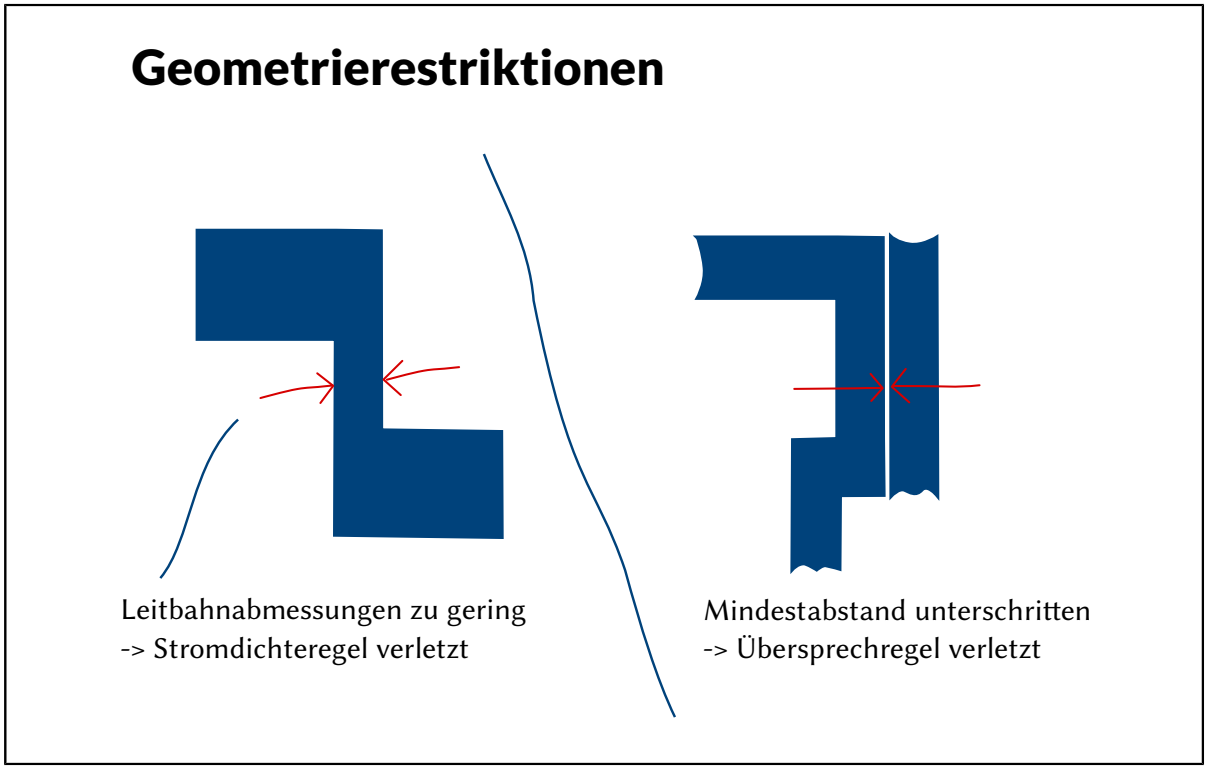
Design Rule Check: Entwurfsregeln



Entwurfsregeln können in drei Kategorien eingeordnet werden, die im Folgenden erläutert werden.

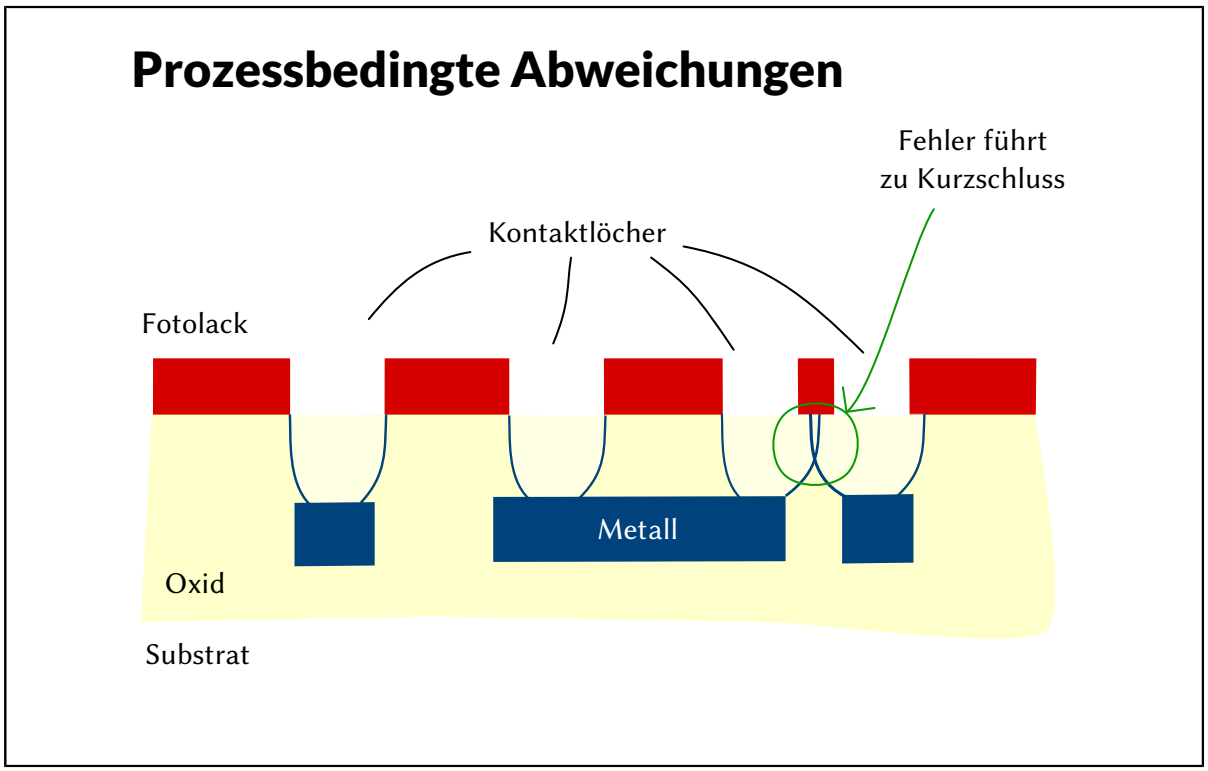
- Elektrisch bedingte Geometrierestriktionen
- Prozessbedingte Abweichungen
- Justiertoleranzen von Masken

Design Rule Check: Geometrierestriktionen



Im Layout werden geometrische Merkmale überprüft, die aufgrund elektrischer Randbedingungen eingehalten werden müssen. Das steht im Gegensatz zur elektrischen Regelprüfung, die eine Schaltung topologisch prüft. Zu der geometrischen Prüfung gehört beispielsweise die Überprüfung einer erforderlichen Mindestleitbahnbreite für eine Strombelastung, die aus einer höchstzulässigen Stromdichte resultiert, oder Abstandsbedingungen zwischen Bauelementen und/oder Leiterbahnen, um Feldstärken und Kapazitäten in Grenzen zu halten.

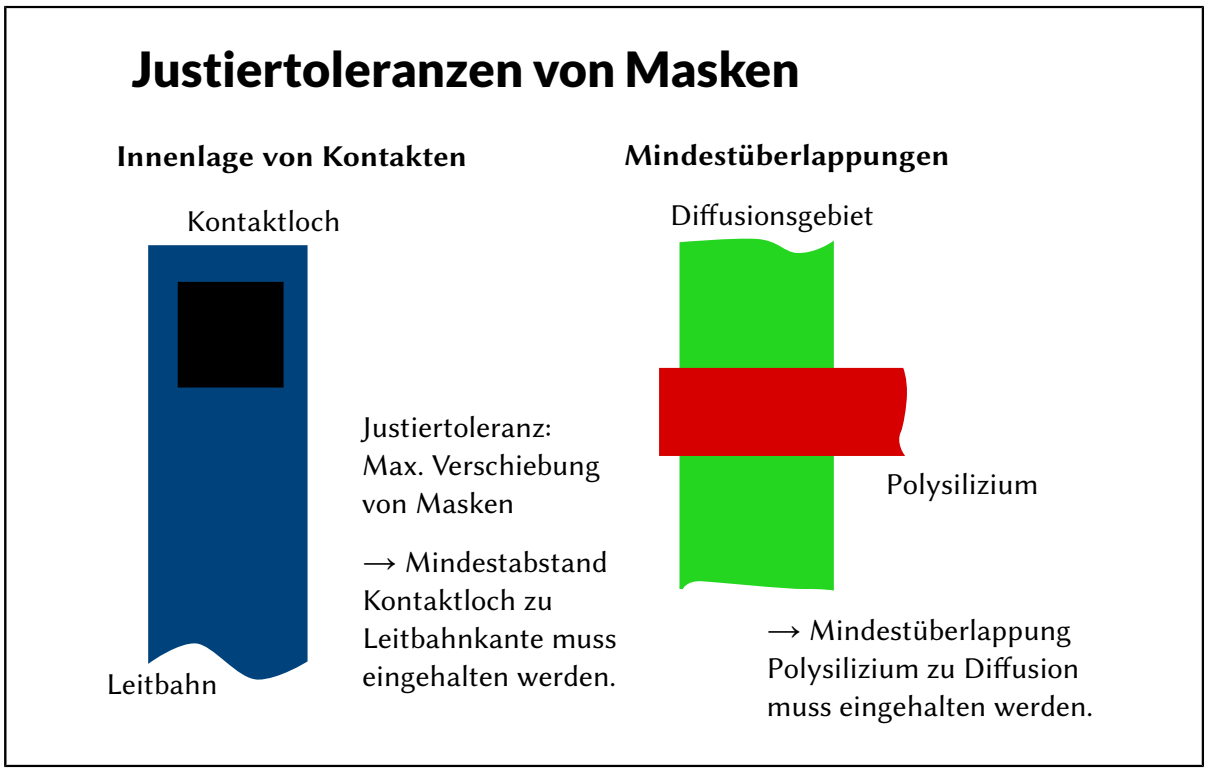
Design Rule Check: Prozessbedingte Abweichungen



Ein Beispiel für prozessbedingte Abweichungen ist die endliche Auflösung von Ätzprozessen, die je nach Herstellungsprozess variieren. Aber auch die laterale Ausbreitung einer Diffusion wird den prozessbedingten Abweichungen zugeordnet. Die Abbildung zeigt links Kontaktlöcher für eine Durchkontaktierung zu einer zweiten Metallebene mit dem erforderlichen Mindestabstand. Die Kontaktlöcher sind unter den Öffnungen im Fotolack aufgeweitet, weil der Ätzprozess nicht ideal vertikal abläuft.

Rechts ist der Mindestabstand nicht eingehalten, so dass es zu einer Überschneidung zweier Kontaktlöcher und damit zu einem Kurzschluss kommt.

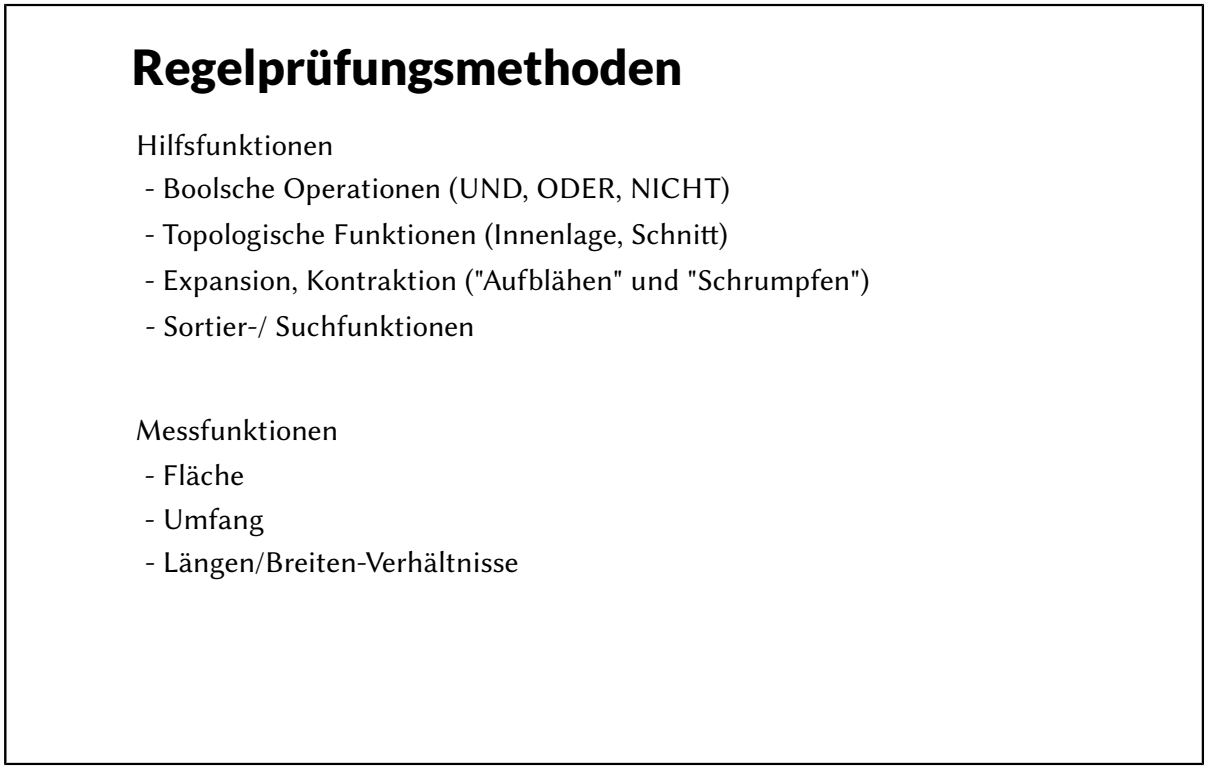
Design Rule Check: Justiergenauigkeit



Die Justiergenauigkeit der Masken bei der Fotolithographie liegt in der Größenordnung der optischen Auflösung der Prozesse. Daher muss zum Beispiel die Innenlage von Kontakten in Leitbahnen eingehalten werden, um diese Toleranzen zulassen zu können.

Ein anderes Beispiel ist eine Mindestüberlappung beim Aufbau von Bauelementen in mehreren Schichten, die zur Sicherheit vorhanden sein muss.

Design Rule Check: Regelprüfung



Das Layout besteht aus einer Vielzahl von Polygonen. Mit Hilfe geometrischer Operationen und Verknüpfungen von Layern können die Polygone bearbeitet werden, um sie anschließend mit Hilfe von Messfunktionen zu überprüfen. Eine Regel kann aus mehreren Operationen und Messfunktionen bestehen. Dabei gibt es oft unterschiedliche Möglichkeiten, eine Regel zu definieren.

Beispielsweise ist es möglich, den Mindestabstand zwischen allen Polygonen zu überprüfen, indem alle Polygonabstände gemessen werden. Die zweite Möglichkeit besteht darin, alle Polygone um den halben Mindestabstand aufzublähen und dann die Schnittmenge aller Polygone zu berechnen. Ist die resultierende Fläche nicht leer, so wurde der Mindestabstand nicht eingehalten.

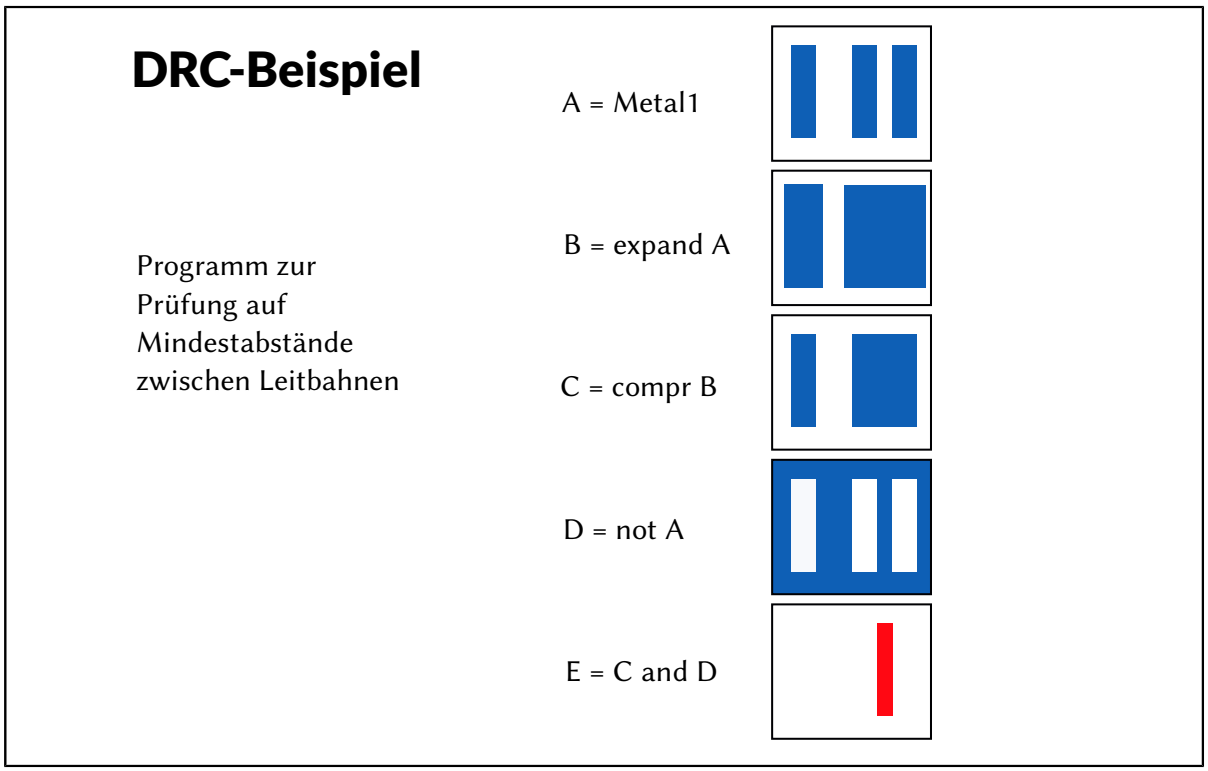
Hilfsfunktionen

Eingesetzt werden
- Sortier-/Suchfunktionen
- topologische Funktionen (Innenlage, Schnitt)
- boolsche Operationen (UND, ODER, NICHT)
- Expansion, Kontraktion("Aufblähen" und "Schrumpfen")

Messfunktionen

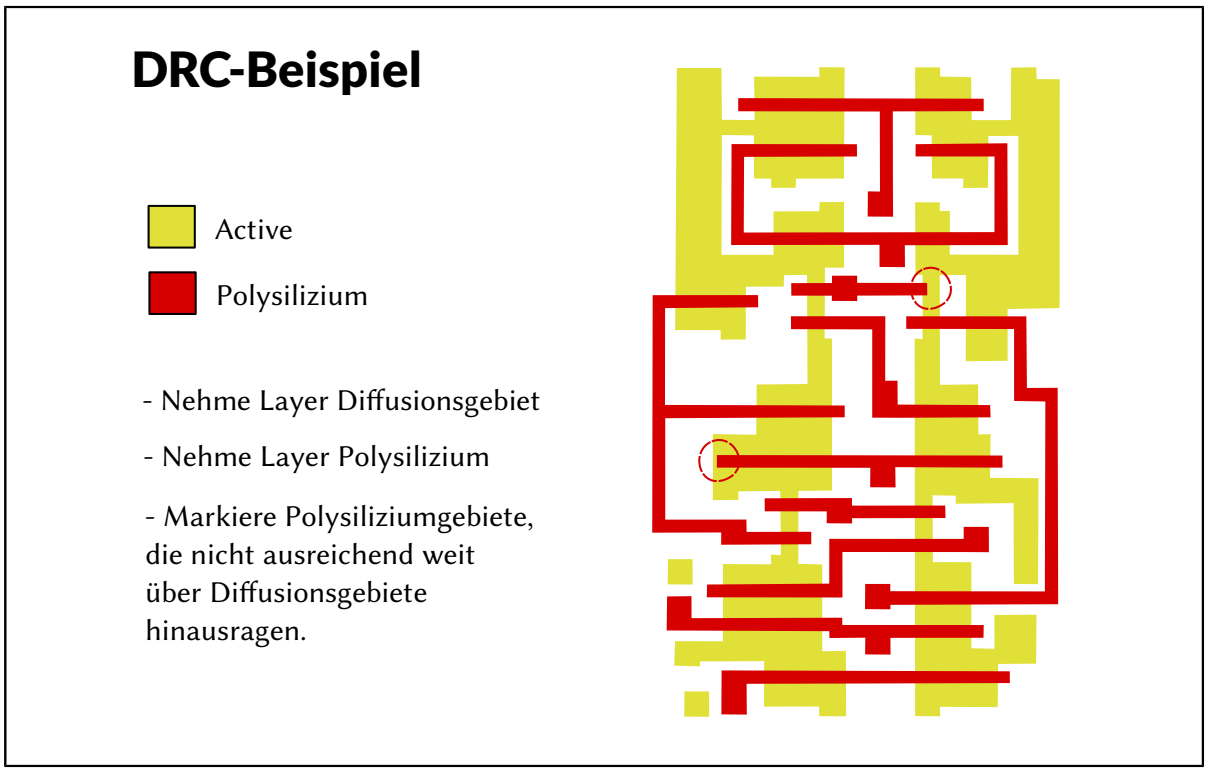
Als Messfunktionen stehen zur Verfügung:
- Fläche
- Umfang
- Längen/Breiten-Verhältnisse

Design Rule Check: DRC-Beispiel 1



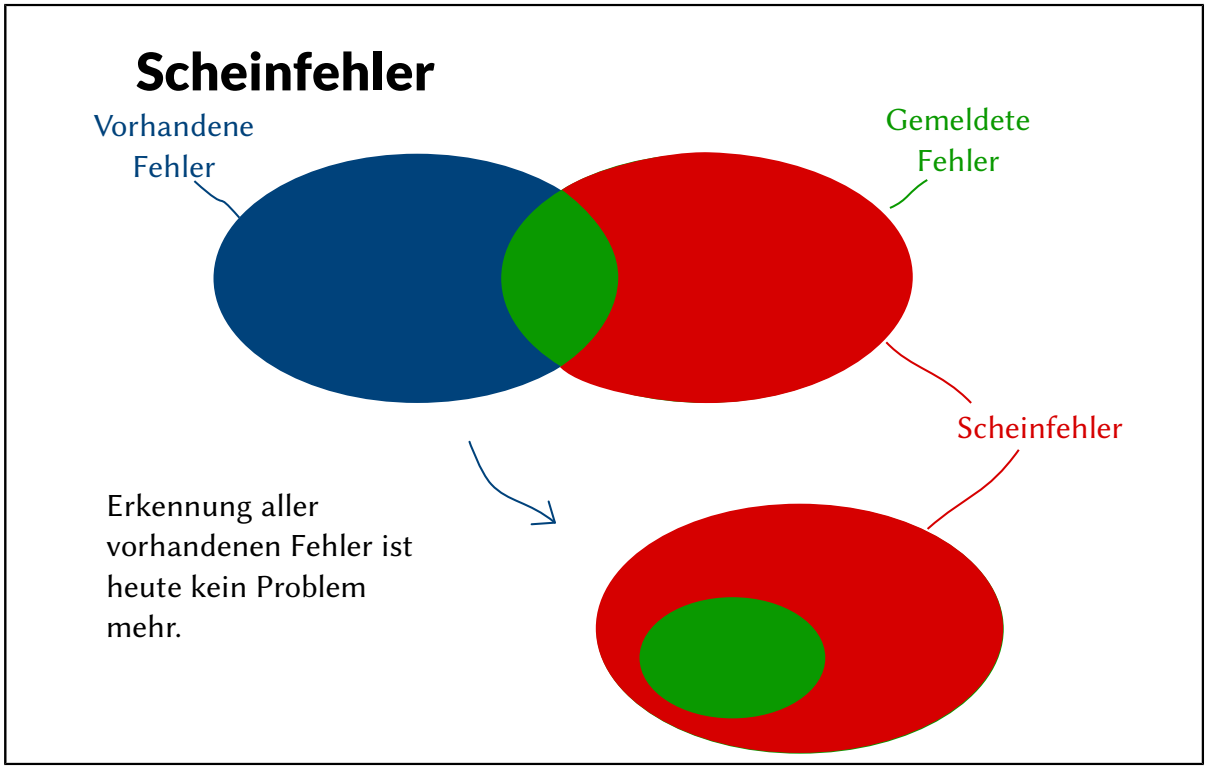
Das Beispiel zeigt eine Regelfolge zur Prüfung auf Mindestabstände zwischen Metallleitungen auf einer Ebene. Die Prüfanweisungen ist in Pseudocode angegeben. Der Kern besteht darin, die Polygone um d/2 zu expandieren und anschließend wieder zu kontrahieren. Dabei ist d der Mindestabstand zwischen zwei Metallleitungen.

Design Rule Check: DRC-Beispiel 2



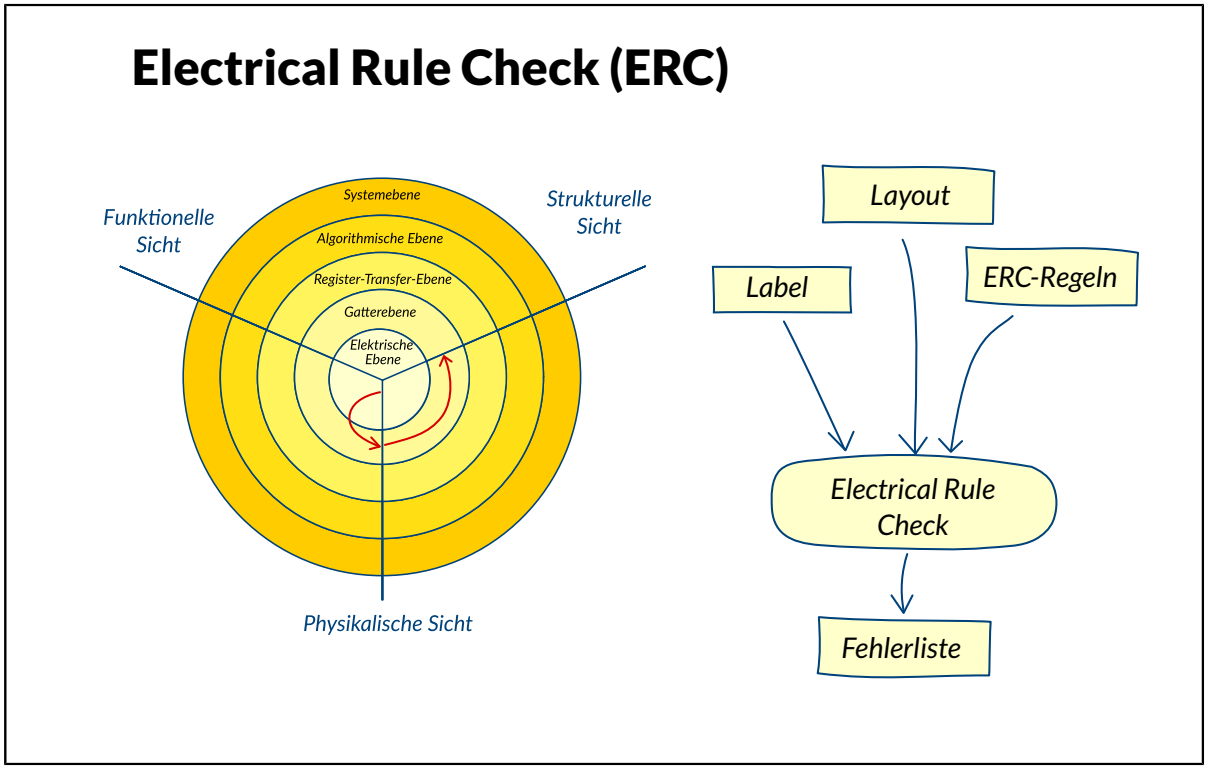
Die Prüfoperationen werden auf das Layout einer Flip-Flop-Standardzelle angewandt.

Design Rule Check: Scheinfehler



Fehler können bei einer Prüfung in zwei Formen auftreten. Wenn ein Fehler im Layout existiert, dieser jedoch nicht erkannt wird oder wenn kein Fehler vorliegt, vom Prüfprogramm jedoch ein solcher gemeldet wird. Der erste Fall tritt in heutigen Prüfprogrammen kaum noch auf. Der zweite Fall, die so genannten Scheinfehler können auch von heutigen Prüfprogrammen nicht verhindert werden.

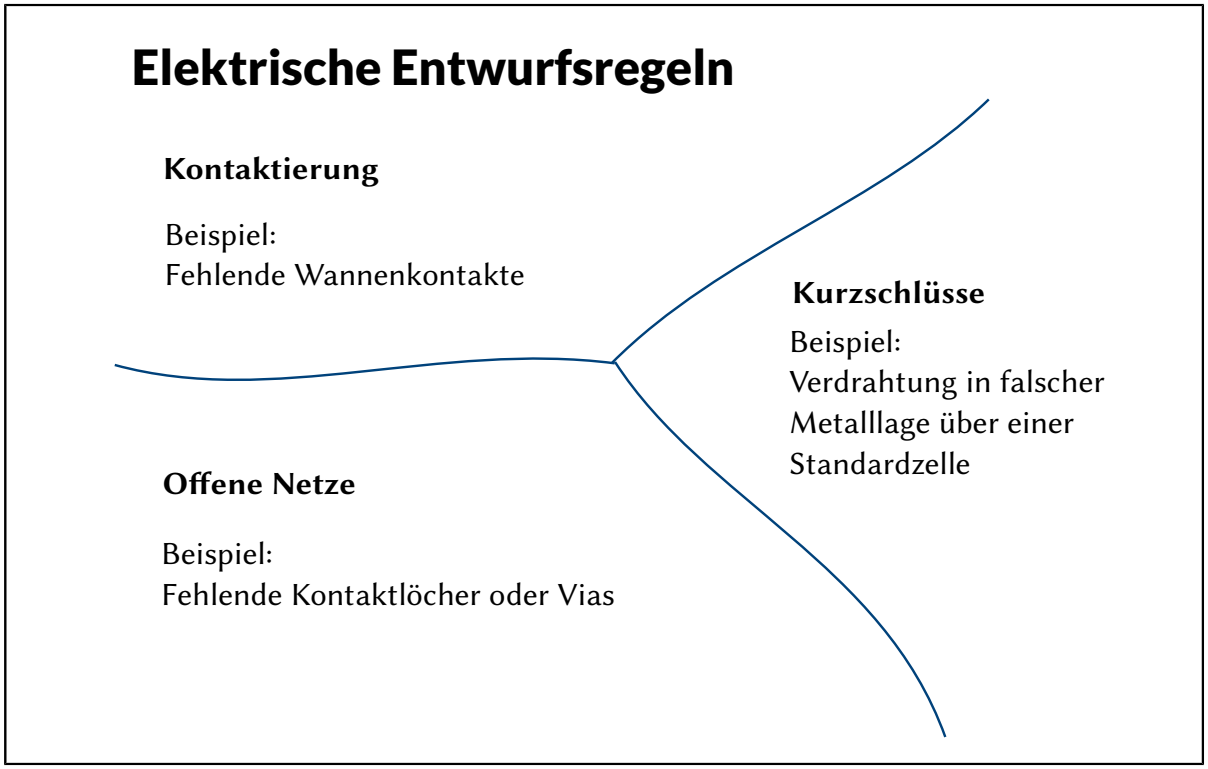
Design Rule Check: Electrical Rule Check



Im Gegensatz zur Entwurfsregelprüfung werden bei der Prüfung elektrischer Regeln die Struktur und Integrität von Netzen im Layout untersucht, damit Schaltungsfehler wie z.B. Kurzschlüsse und offene Netze vermieden werden können.

Bei der Prüfung elektrischer Regeln werden zusätzlich zum Layout und den Regeln so genannte Label als Eingangsparameter verwendet. Ein Label ist eine eindeutige Bezeichnung für ein Netz und wird benötigt, um dasselbe Netz in unterschiedlichen Layern identifizieren zu können. Das Ergebnis der elektrischen Prüfung sind Regelverletzungen und eine Fehlerliste.

Design Rule Check: Elektrische Entwurfsregeln



Die Entwurfsregeln werden vom Chiphersteller vorgegeben, können aber durch die Formulierung in einer Technologiebeschreibungssprache vom Schaltungsdesigner angepasst werden. Anhand des Layouts mit eindeutig gekennzeichneten Netzen können einfache topologische Merkmale der Schaltung geprüft werden. Diese werden im folgenden kurz erläutert.

In diese Kategorie gehört z.B. die Bedingung, dass jede Wanne ein bestimmtes Potential haben muss, damit die darin realisierten Bauelemente funktionsfähig sind. Die parasitären pn-Übergänge müssen in Sperrrichtung gepolt sein, d.h. eine n-Wanne muss mit der Betriebsspannung und eine p-Wanne muss mit dem Nullpotential verbunden sein.

Eine nicht gewünschte Verbindung zwischen zwei Netzen, also ein Kurzschluss, kann insbesondere entstehen, wenn ein Layout hierarchisch erstellt wird. Dabei werden die Grundelemente, meist Standardzellen, durch das so genannte Abstract repräsentiert, das nur die Anschlusspunkte des Grundelements, nicht aber sein Layout beschreibt. Wird vom Schaltungsentwickler (oder durch das Verdrahtungswerkzeug) eine Leitung über ein solches Abstract geführt, kann ein Kurzschluss entstehen, wenn in der Standardzelle der gleiche Layer verwendet wird. Dieser Kurzschluss ist für den Schaltungsentwickler nicht zu erkennen, weil das Layout der Standardzelle für ihn nicht sichtbar ist.

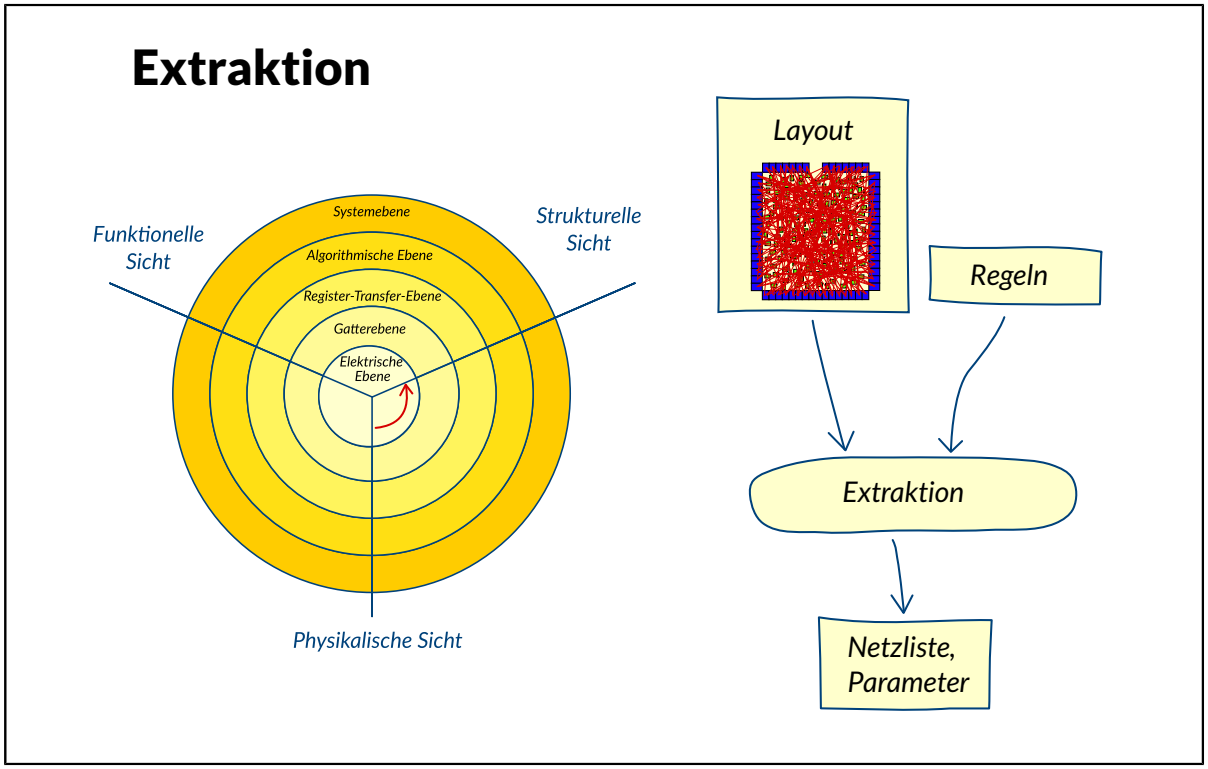
Offene Netze sind vom Schaltungsentwickler schwer zu identifizieren, da sie typischerweise durch kleine Details im Layout verursacht werden, wie z.B. fehlende Kontaktlöcher oder Vias. Durch offene Verbindungen können ganze Schaltungsteile ausfallen, wodurch eine Überprüfung aller Leitungen unumgänglich ist.

Electronic Design Automation (EDA)

Extraktion

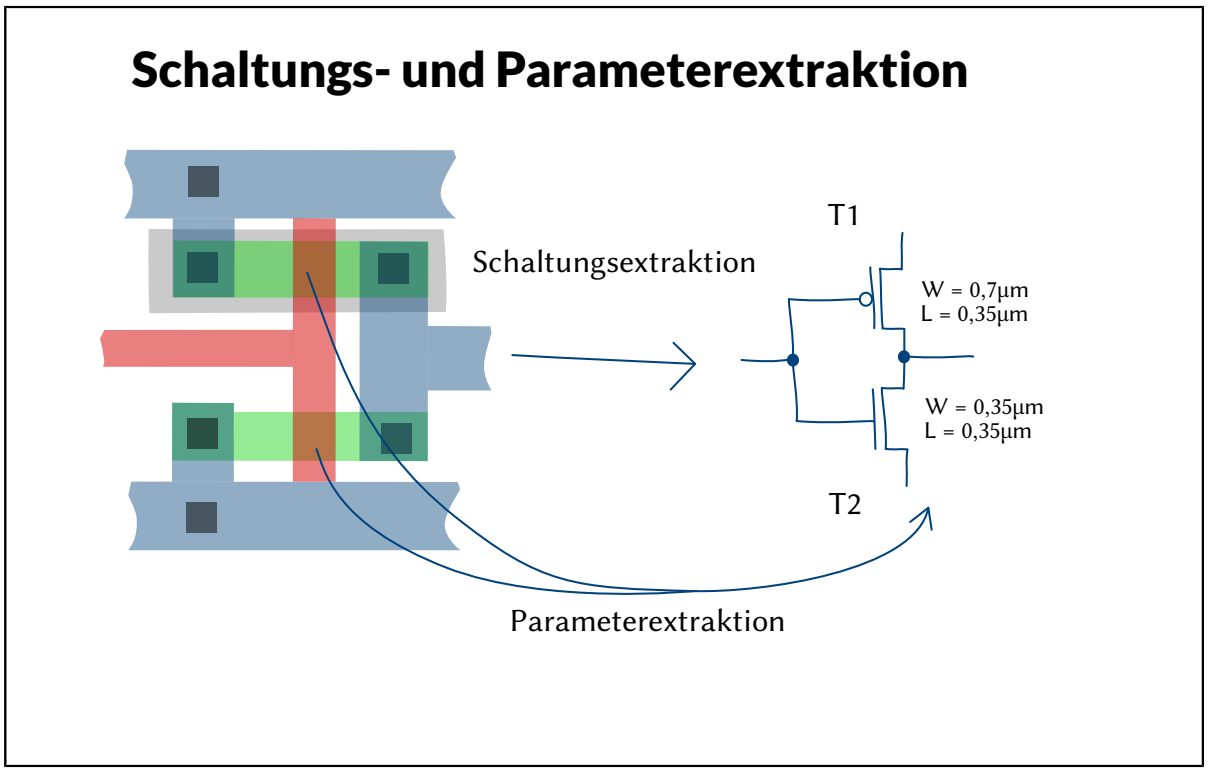
Extraktion
Schaltungs- und Parameterextraktion
Schaltungsextraktion
Bauelementerkennung
Verdrahtungsanalyse
Netzlistenerstellung
Parameterextraktion
Parasitäre Elemente
Leitbahnparasiten
Aktive Siliziumparasiten
Passive Siliziumparasiten

Extraktion: Extraktion



Der Begriff Extraktion beim Entwurf integrierter Schaltungen bezeichnet die Extraktion der elektrischen Eigenschaften aus dem Layout. Anhand eines Regelsatzes wird aus den Layoutdaten unter Zuhilfenahme von Technologieinformationen eine Netzliste auf elektrischer Ebene extrahiert. Eine Extraktion ist notwendig, da die beim DRC durchgeführte Überprüfung geometrischer Regeln zur Gewährleistung der korrekten Funktion des Layouts nicht ausreicht. Über dies gibt weitere Fehlerquellen, die durch einen DRC nicht erfasst werden. Beim Layoutentwurf können Fehler in der Struktur der Schaltung, z.B. durch falsche Verbindungen oder Kurzschlüsse, auftreten. Die elektrischen Eigenschaften der geometrischen Strukturen können die spezifizierten Grenzwerte verfehlen. Wird z.B. eine Leitbahn zu lang oder zu dünn ausgeführt, so kann ihr elektrischer Widerstand und damit ihre Verzögerungszeit zu groß werden. Um das Layout auf diese Fehlerquellen hin überprüfen zu können, wird eine Extraktion durchgeführt.

Extraktion: Schaltungs- und Parameterextraktion



Die Extraktion wird in zwei Schritten durchgeführt:

- Schaltungsextraktion
- Parameterextraktion

Als Ergebnis einer Schaltungsextraktion mit anschließender Parameterextraktion liegt eine Netzliste auf elektrischer Ebene des Layouts vor, die als Eingabe für einen Schaltkreissimulator verwendet werden kann. Sie kann daher zur genaueren Simulation der Schaltung verwendet werden. Die extrahierte Netzliste bildet die Grundlage für den ERC (Electrical Rule Check), eine topologische Prüfung, bei der überprüft wird, ob ein gegebener Satz elektrischer Regeln eingehalten wird (z.B. Kurzschlüsse oder unverbundene Netze). Eine weitere topologische Prüfung ist der Vergleich von extrahierter Netzliste und Netzliste der ursprünglichen Schaltung. Dieser Vorgang wird als Layout versus Schematic (LVS) bezeichnet.

Extraktion: Schaltungsextraktion

Schaltungsextraktion

1. Schritt:
Bauelementerkennung

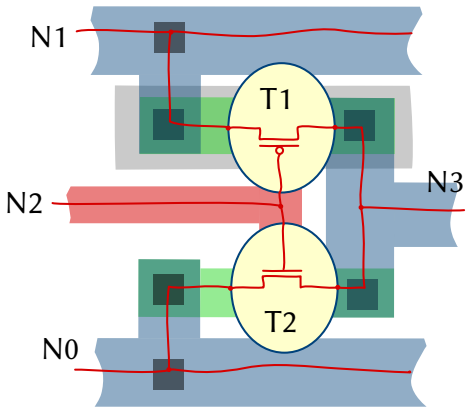
T1 NMOS
T2 PMOS

2. Schritt:
Verdrahtungsanalyse

N0 - N3

3. Schritt:
Netzlistenerstellung

T1 NMOS N2 N3 N1
T2 PMOS N2 N3 N0



Bei der Schaltungsextraktion wird die Netzliste extrahiert. Bei CMOS-Schaltungen sind dazu drei Schritte notwendig.

- Bauelementerkennung
- Verdrahtungsanalyse
- Netzlistenerstellung

Extraktion: Bauelementerkennung

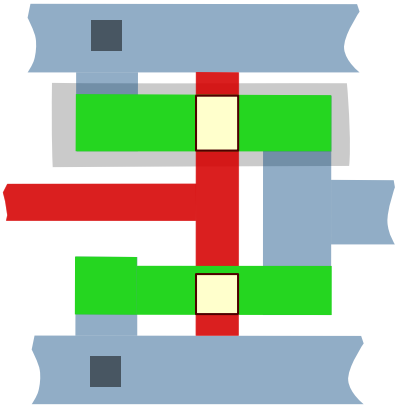
Bauelementerkennung

Beispiel: Erkennung der Transistoren

1) Nehme Layer Polysilizium

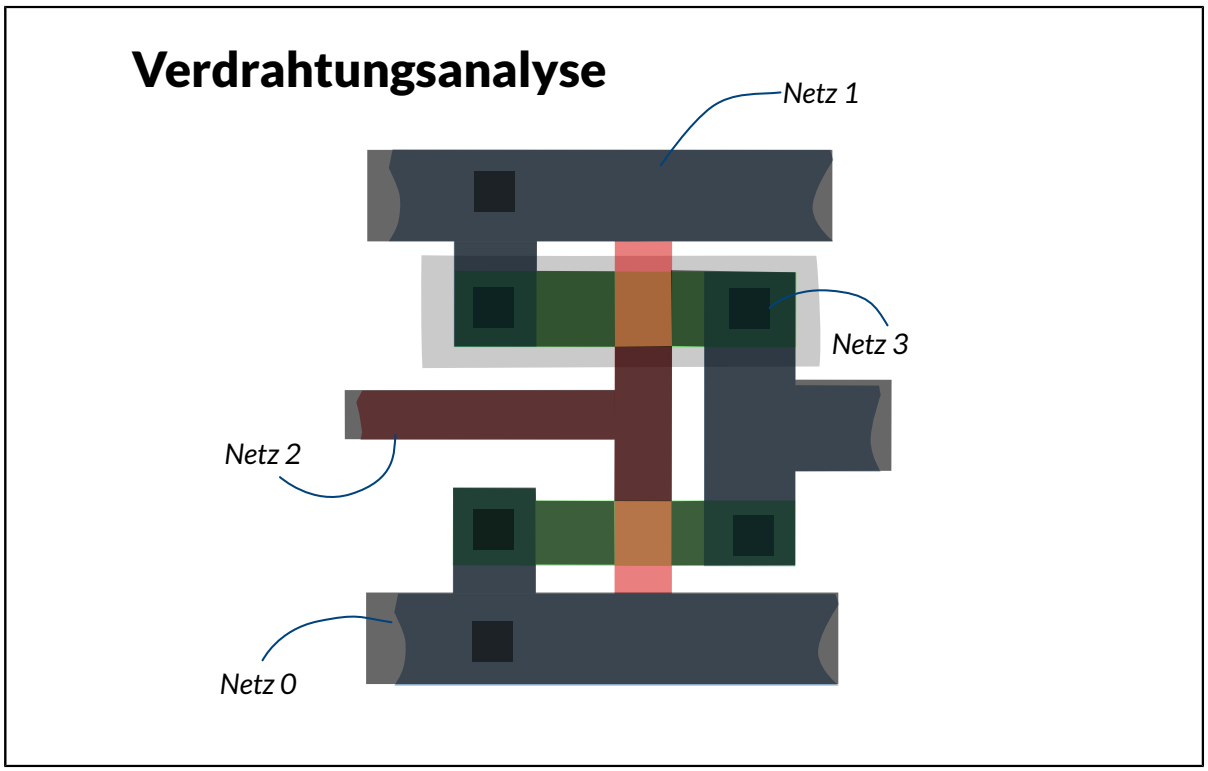
2) Nehme Layer Diffusion

3) Bilde Layer Polysilizium und Diffusion



Die Bauelementerkennung erfolgt aus der geometrischen Struktur des Layouts. Sie beginnt bei den Transistoren. Diese können leicht an der Topologie ihrer Gates lokalisiert werden. Dazu werden boolesche Operationen an den Layoutdaten durchgeführt (z.B. NGATE=(POLY and NDIFF)). Aufgrund ähnlicher und zum Teil komplexerer Operationen können sämtliche Bauelemente erkannt werden.

Extraktion: Verdrahtungsanalyse



Nach der Bauteilerkennung folgt die Verdrahtungsanalyse. Sie umfasst sämtliche Strukturen, die nicht den Bauelementen zugeordnet werden konnten. Es werden alle Netze bestimmt, indem alle Strukturen zusammen gefasst werden, die aufgrund von Kontaktierungen und Überlappungen auf gleichem elektrischen Potential liegen.

Extraktion: Netzlistenerstellung

Netzlistenerstellung

Bauelement T1

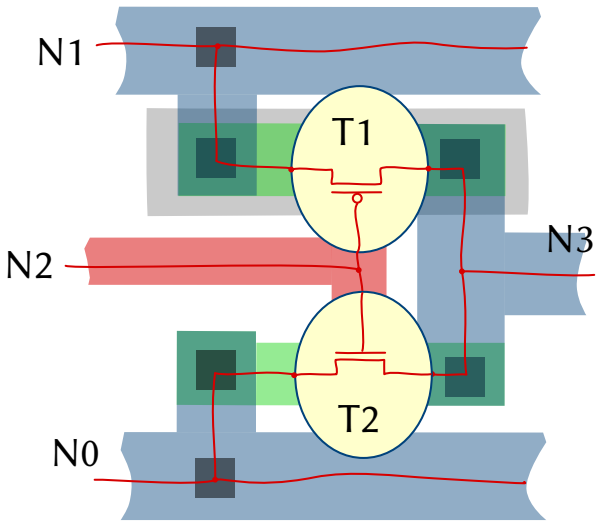
Drainanschluss: N3
Gateanschluss: N2
Sourceanschluss: N1

Bauelement T2

...

Netzliste:

T1 N3 N2 N1
T2 N3 N2 N0



Im letzten Schritt wird die Netzliste erstellt, indem die Anschlussknoten der Bauelemente den Netzen zugeordnet werden, bei Transistoren sind dieses die Source-, Drain und Gateanschlüsse, bei Dioden die Anode und Kathode. In den praktischen Ausführungen von Extraktionsprogrammen sind diese Schritte jedoch nicht eindeutig voneinander trennbar.

Extraktion: Parameterextraktion

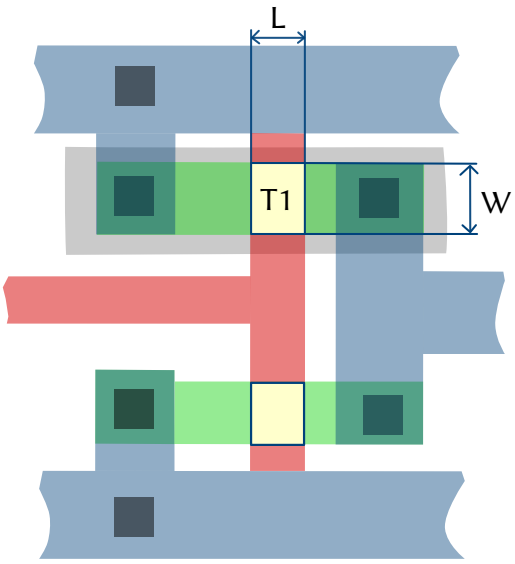
Parameterextraktion

Bestimmung der Parameter aus der Geometrie der Schaltung:

Beispiel Transistor T1:

Kanallänge: L

Kanalbreite: W



Die bisher bestimmte Netzliste enthält nur Strukturinformationen. Netzlisten zur analogen Schaltungssimulation und zum LVS enthalten zusätzlich die elektrischen Kenngrößen der Bauelemente. Diese werden bei der Parameterextraktion aus den geometrischen Daten des Layouts und Technologieinformationen gewonnen. In der Netzliste werden die Bauelemente durch ihre Bauelementmodelle repräsentiert. Es existieren unterschiedliche Modelle für die gleichen Bauelemente. Je nach verwendetem Bauelementmodell werden unterschiedliche Parameter extrahiert. Für MOS-Transistoren stellen die Weite W und die Länge L die wichtigsten Parameter dar. Die Bestimmung der Parameter W und L für Transistor T1 ergibt sich aus den Abmaessungen des Gatebereichs. Je nach Komplexität und Anwendungsfall werden aber auch weitere Größen extrahiert, wie z.B. Kapazitäten zwischen Gate bzw. Substrat und Source bzw. Drain. Diese Parameter beeinflussen ebenfalls das Verhalten des Transistors, sind jedoch nicht Resultat des geplanten Schaltungsentwurfs, sondern Folgen parasitärer Effekte.

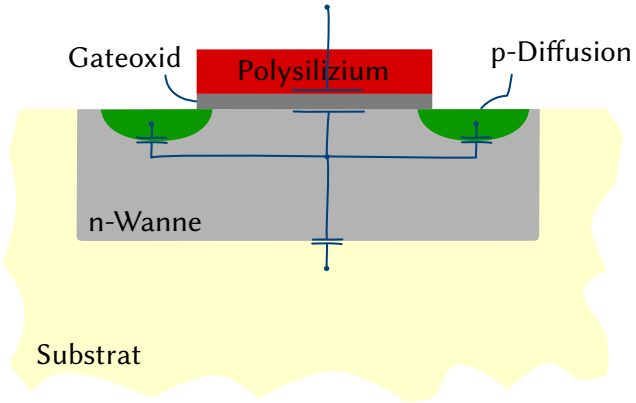
Extraktion: Parasitäre Elemente

Parasitäre Elemente

Beispiel: Kapazitäten im P-Kanal-Transistor

- Grenzschichtkapazitäten
- Gatekapazität

Generelle Einteilung:
- Leitbahnparasiten
- Aktive Parasiten im Halbleiter
- Passive Parasiten im Halbleiter

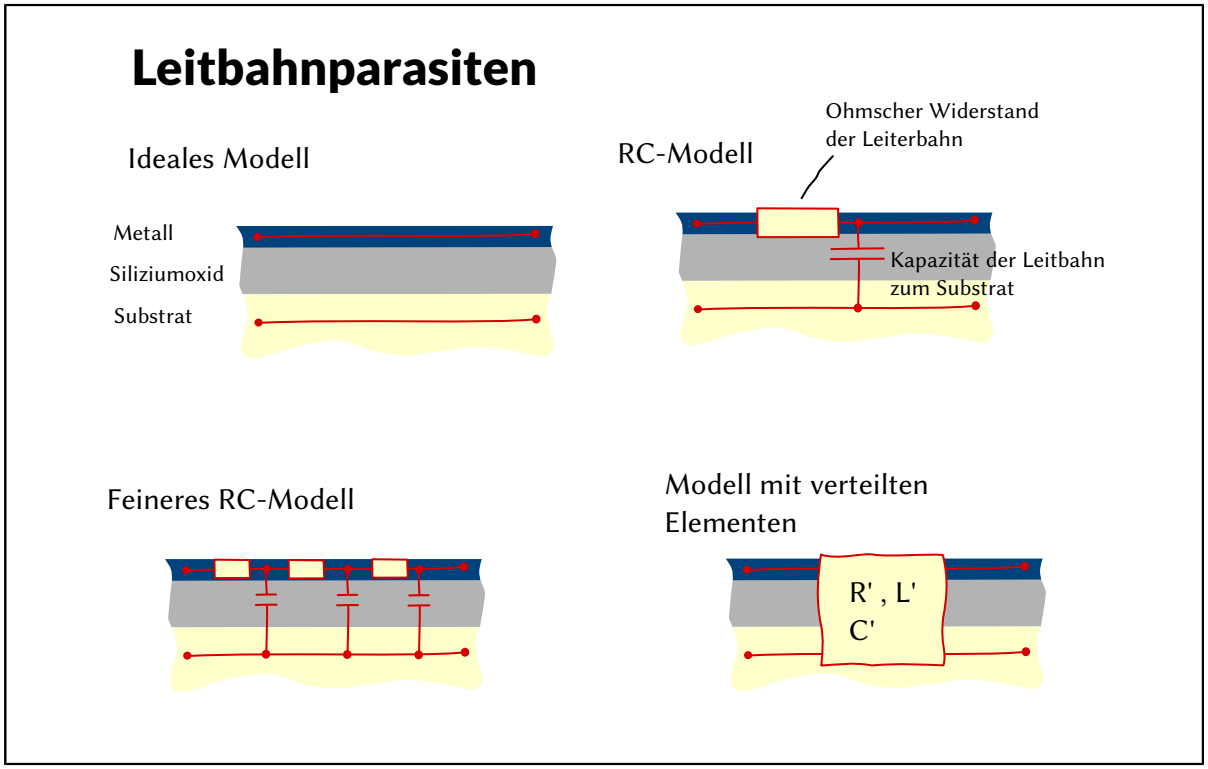


Parasitäre Effekte entstehen aufgrund von physikalischen Effekten, die nicht in direktem Zusammenhang mit der gewünschten Funktion der Schaltung stehen. Sie ergeben sich aus der Realisierung der integrierten Schaltung und können erst nach Entwurf des Layouts berücksichtigt werden. Parasitäre Effekte treten nicht nur innerhalb der Bauelemente auf. Um parasitäre Effekte auch außerhalb von Bauelementen zu berücksichtigen, werden zusätzliche parasitäre Elemente extrahiert. Diese parasitären Elemente beschreiben die Eigenschaften der parasitären Effekte bei einer analogen Schaltungssimulation. Das Ergebnis ist eine erweiterte elektrische Netzliste, die neben den Nutzelementen der Schaltung zusätzlich parasitäre Elemente enthält.

Innerhalb der Halbleiterstrukturen können parasitäre Effekte unterschieden werden, die auf aktive oder passive Elemente führen. Da ein großer Teil der Schaltung aus Verbindungen der Bauelemente untereinander besteht, nehmen die parasitären Effekte der Verbindungsstrukturen (Leitbahnen) eine besondere Rolle ein. Es ergibt sich eine Unterteilung in drei Gruppen von parasitären Elementen:

- Leitbahnparasiten
- Aktive Parasiten im Halbleiter
- Passive Parasiten im Halbleiter

Extraktion: Leitbahnparasiten



Leitbahnparasiten entstehen durch die nichtidealen Eigenschaften der Verbindungsstrukturen. Beim Entwurf integrierter Schaltungen war es lange Zeit ausreichend, Leitbahnen als ideale Verbindung zwischen zwei Knoten A und B zu betrachten. Durch den technologischen Fortschritt werden die physikalischen Strukturen immer kleiner und die Frequenzen der Spannungen und Ströme immer höher. Dies führt dazu, dass immer genauere Modelle für Leitbahnen berücksichtigt werden müssen.

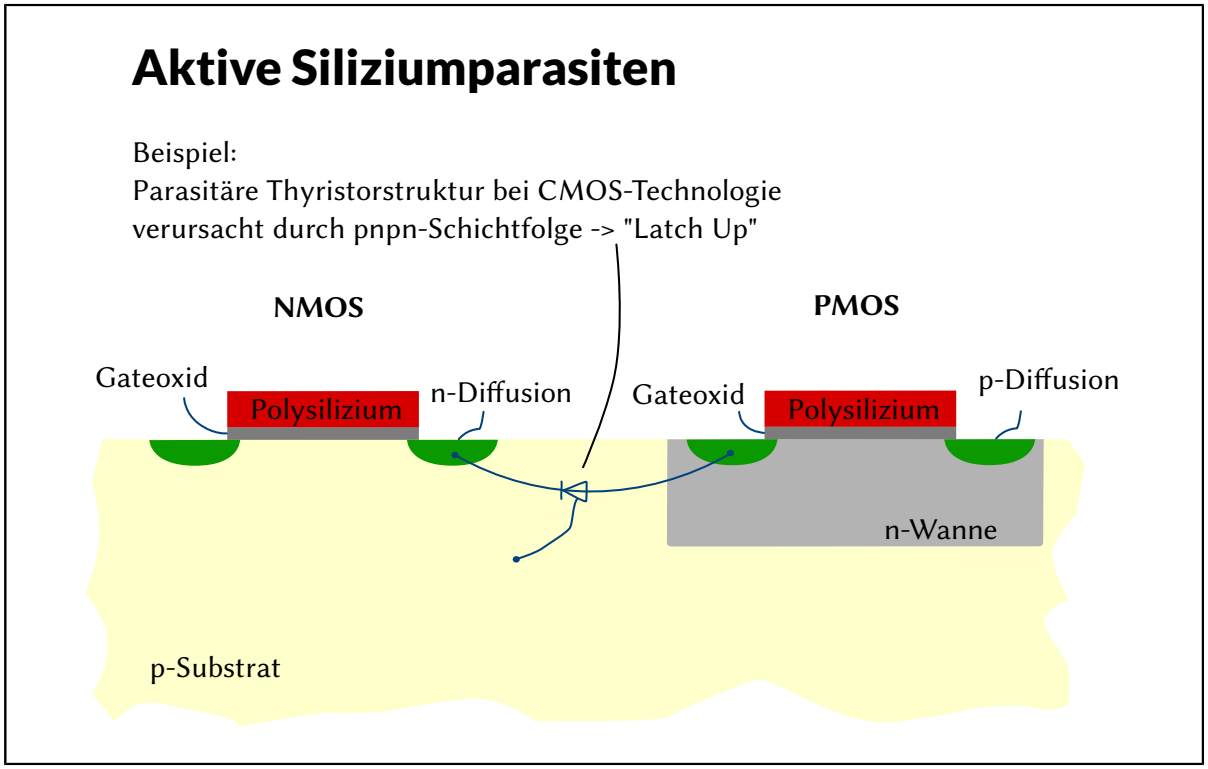
Zunächst wurde das elektrische Feld zwischen Leitbahn und dem darunter liegenden Halbleitersubstrat berücksichtigt. Dazu wurde eine Kapazität zwischen Leitbahn und Masseknoten eingefügt.

Zur Abschätzung der unmittelbarer Leitungsverzögerungen in digitalen Schaltungen wurde ein Modell benötigt, in dem die Widerstände mit den Kapazitäten der Leitbahn ein einfaches Verzögerungsglied darstellen. Dieses Modell kann durch eine abschnittsweise Betrachtung weiter verfeinert werden. Darüber hinaus ist es heute häufig notwendig auch das Übersprechen zu lateral oder vertikal benachbarten Leitbahnen durch Koppelkapazitäten zu berücksichtigen. Dies führt zu kombinierten R/C-Modellen mit steigender Komplexität.

Mit weiterem technologischen Fortschritt wurde es notwendig, Leitungsreflexionen zu berücksichtigen. Dazu wurden Modelle der Leitungstheorie mit verteilten Elementen (R', C' und L') verwendet, die den Zusammenhang zwischen Ausgangs- und Eingangsgröße anhand der allgemeinen Leitungsgleichungen darstellen. Eine noch genauere dreidimensionale Betrachtung ergibt sich aus der Anwendung der Feldtheorie. Durch Anwendung der Maxwellschen Gleichungen werden die elektromagnetischen Felder der Leitbahnen berechnet. Die Integration solcher Modelle in die Netzliste stellt besondere Anforderungen an den Analogsimulator, da die Eingangsgrößen nicht mehr Spannungen und Ströme, sondern elektrische und magnetische Feldgrößen darstellen.

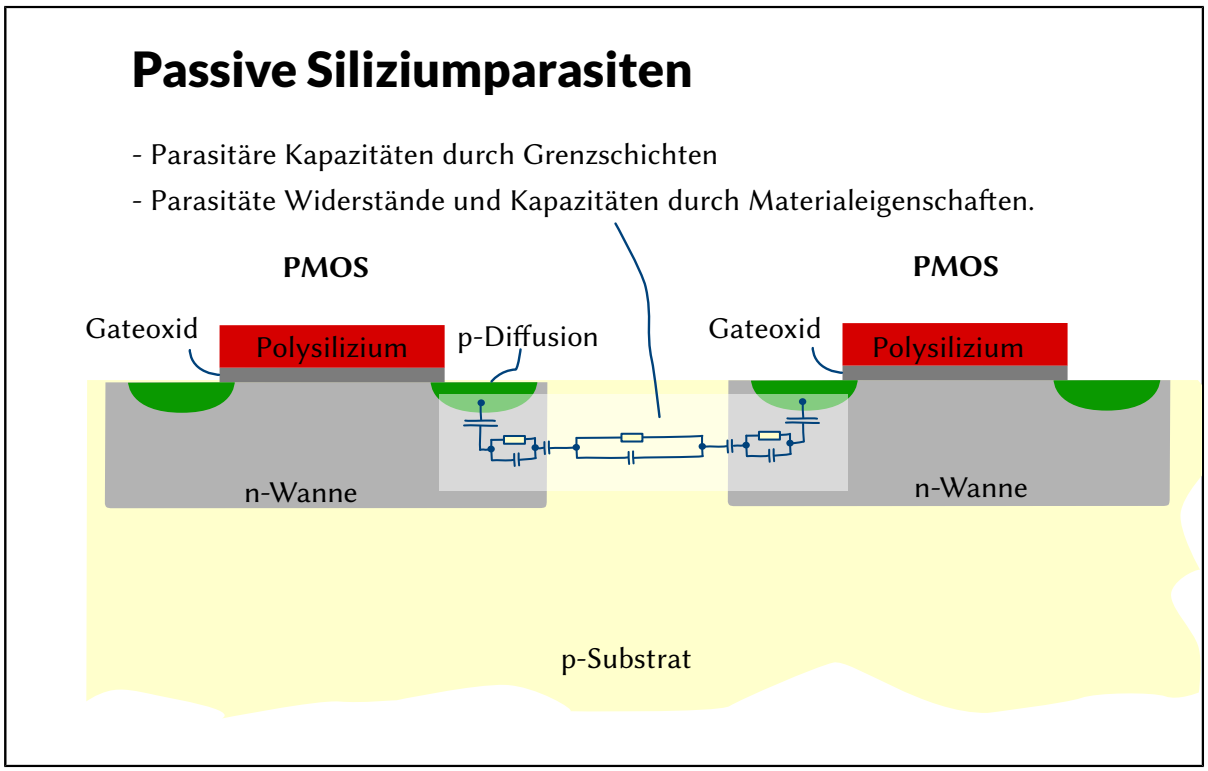
Die höhere Genauigkeit, die man mit komplexeren Modellen erreicht, erfordert einen größeren Aufwand in der Simulation. Der Entwickler muss abwägen, welche Genauigkeit notwendig ist und wie viel Zeit die entsprechende Simulation benötigen darf.

Extraktion: Aktive Siliziumparasiten



Aktive Parasiten innerhalb des Halbleiters entstehen durch Kombination von Grenzschichten unterschiedlich dotierter Gebiete wie npn, pnp oder pnpn. Solche Kombinationen bilden zum Beispiel bipolare Nutztransistoren. Es kommt aber auch zu parasitären Kombinationen solcher Gebiete, aus denen parasitäre bipolare Transistoren entstehen. Besonders gefährlich können dabei pnpn-Kombinationen sein. Diese bilden einen Thyristor, der durch ungünstige Potentialverteilungen zum Zünden gebracht werden kann. Dadurch fließen sehr große parasitäre Ströme, die den Halbleiter zerstören können. Dieses Zünden wird als Latch-Up-Effekt bezeichnet.

Extraktion: Passive Siliziumparasiten



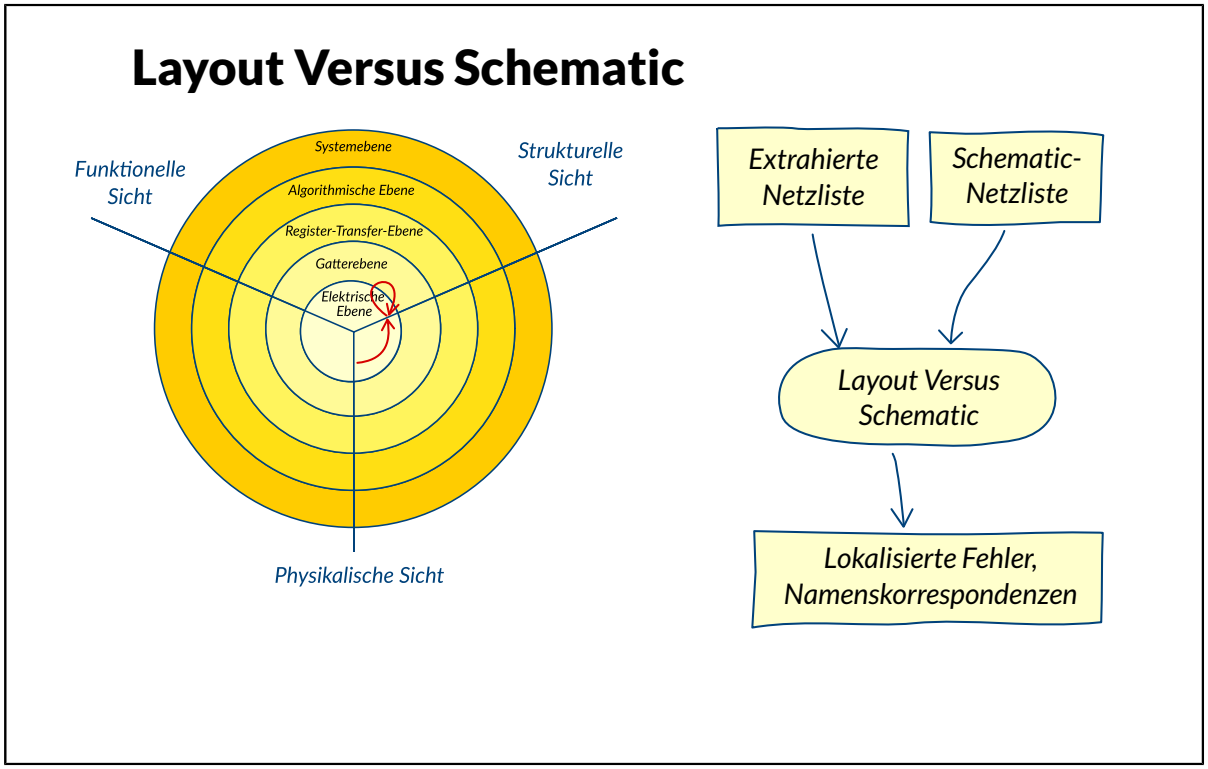
Passive Parasiten im Halbleiter entstehen sowohl an den Grenzflächen unterschiedlich dotierter Gebiete, als auch innerhalb homogen dotierter Gebiete. An einem pn-Übergang entsteht eine Kapazität, deren Kapazitätswert von der Spannung über der Grenzschicht abhängt. Innerhalb homogen dotierter Halbleiter ergeben sich parasitäre Widerstände und Kapazitäten aus den Materialeigenschaften. Das Bild zeigt beispielhaft passive parasitäre Elemente zwischen zwei Wannen.

Electronic Design Automation (EDA)

Layout Versus Schematic

Layout Versus Schematic
Darstellung der Schaltung als Graph
Isomorphie
Beispiel

Layout Versus Schematic: Layout Versus Schematic

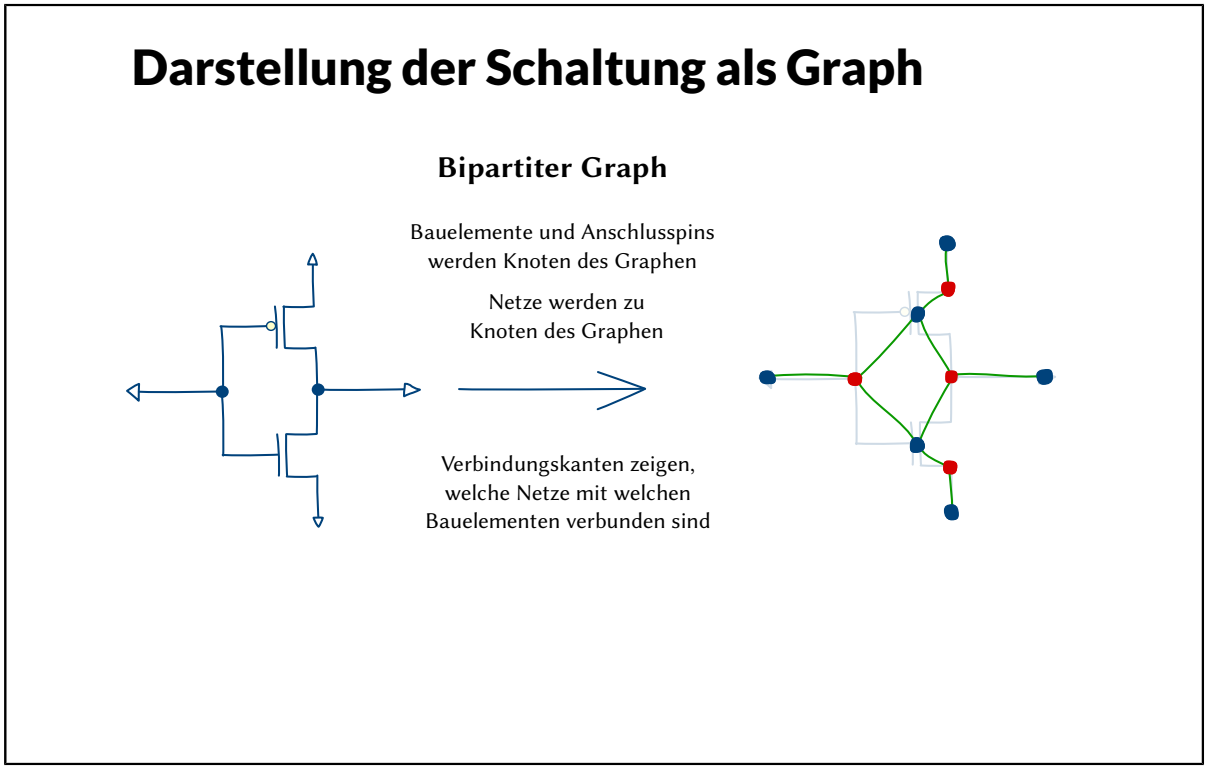


Der Begriff LVS steht für Layout Versus Schematic und bezeichnet den automatischen Vergleich zweier Netzlisten: Einerseits die aus dem Layout extrahierte Netzliste und andererseits die Netzliste, die dem Layout zu Grunde liegt, also das Ergebnis des Schaltungsentwurfs (Schematic).

Bei diesem Vergleich wird durch topologische Layoutprüfung die Übereinstimmung der extrahierten Istschaltung mit der dem Layoutentwurf zugrunde liegenden Sollschaltung überprüft. Die dabei angewandten Algorithmen und Verfahren können zum Vergleich beliebiger Netzlisten auf elektrischer Ebene verwendet werden.

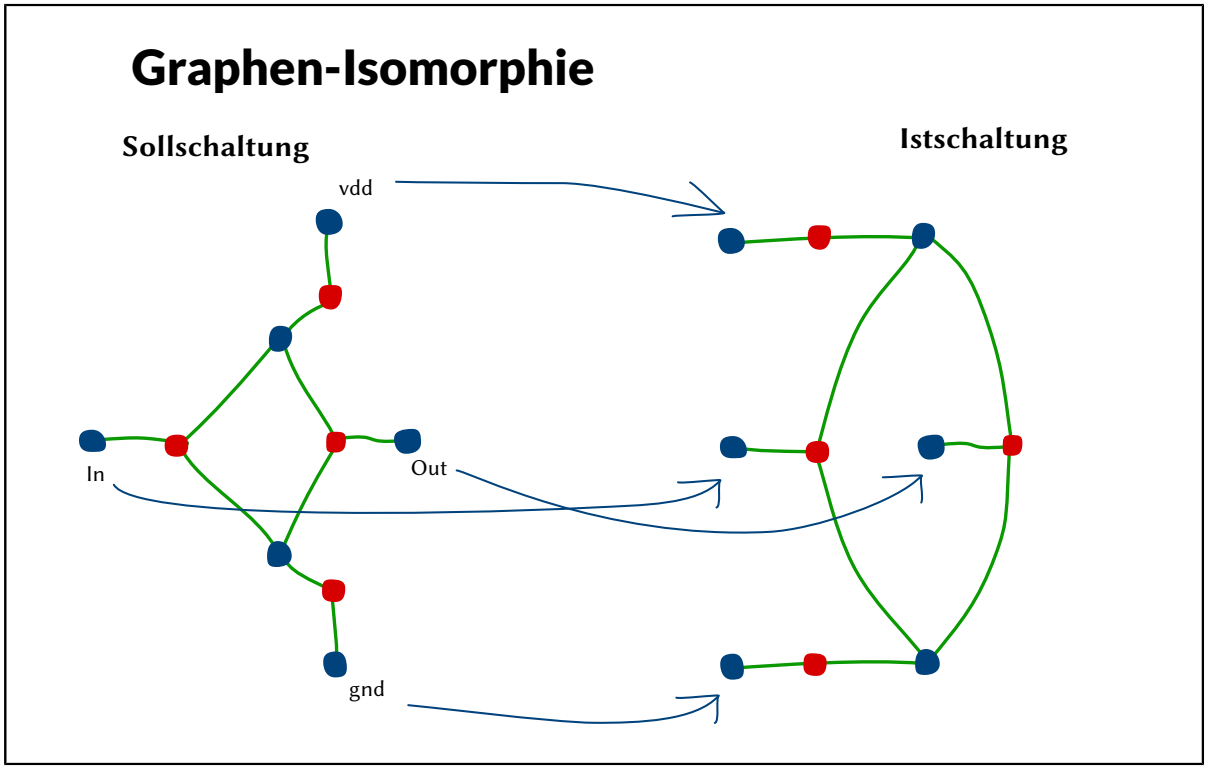
Als Eingabe werden die beiden zu vergleichenden Netzlisten benötigt. Dabei enthält die aus dem Layout extrahierte Netzliste keine parasitären Elemente, sondern nur die Nutzelemente. Die Ausgabe des LVS gibt lokalisierte Fehler aus und erzeugt eine Liste der Namenskorrespondenzen (Zuordnung der Namen von Schaltungselementen aus der entworfenen Netzliste zu Elementen der extrahierten Netzliste). Fehler können einerseits in der Struktur der Netzlisten, andererseits in unterschiedlichen Parametern der Bauelemente auftreten.

Layout Versus Schematic: Darstellung der Schaltung als Graph



Zunächst werden die zu vergleichenden Schaltungen als Graphen dargestellt. Beispielsweise können die Bauelemente und Netze als Knoten und die Verbindungen durch Kanten dargestellt werden. Alle Verfahren zum LVS lassen sich dann auf die Isomorphieuntersuchung zweier Graphen zurückführen. Dabei handelt es sich um ein bekanntes Problem, das im allgemeinen Fall, in dem keine Namenskorrespondenzen zwischen den Graphen bekannt sind, zur Klasse der NP-vollständigen (NP: Nicht-deterministisch Polynomial) Probleme gehört. Bei einem NP-vollständiges Peoblem nimmt die Rechenzeitkomplexität mit wachsender Problemgröße drastisch zu; diese Klasse von Problemen liese sich nur mit Hilfe einer nicht-deterministischen Turing-Maschine effizient lösen. Da keine solche Maschine existiert, ist notwendig das Problem zu vereinfachen und geeignete Algorithmen zur Lösung zu finden.

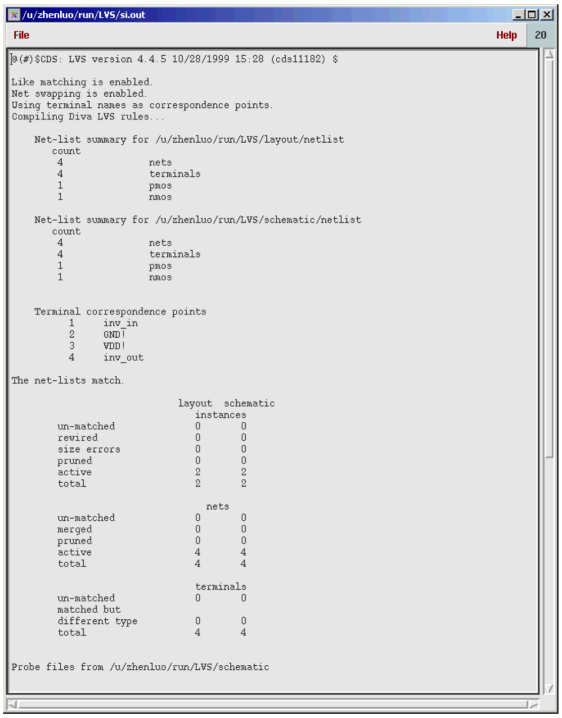
Layout Versus Schematic: Isomorphie



Die Isomorphieuntersuchung wird wesentlich vereinfacht, wenn Namenskorrespondenzen bekannt sind. Oft sind die Terminalnamen in den zu vergleichenden Netzlisten gleich und können daher als Start-Korrespondenzen verwendet werden. Außerdem werden mögliche Korrespondenzen bei übereinstimmenden Bauteiltypen sowie gleicher Anzahl der Anschlüsse verwendet, um die Prüfung auf Isomorphie zu beschleunigen.

Layout Versus Schematic: Beispiel

Beispiel



Das Bild zeigt ein Vergleichsergebnis zweier Schaltungen. In der extrahierten Schaltung haben die Netze automatisch Namen zugewiesen bekommen, die sich von denen der Nominalschaltung unterscheiden.