

VLSI SYSTEMENTWURF

Begleitende Literatur

Das vorliegende Vorlesungsskript ist so konzipiert, daß es den zu erarbeitenden Stoff vollständig abdeckt, mit Ergänzungen und zusätzlichen Beispielen an einigen Stellen. Das Studium des Skriptes kann jedoch nicht den Besuch der Vorlesung ersetzen. Für interessierte Studenten, die den Stoff oder Teile davon aus einer etwas anderen Perspektive betrachten möchten, ist die unten aufgeführte Liste begleitender Literatur gedacht. Die Liste beansprucht in keiner Weise Vollständigkeit und gibt nur die uns bekannten Bücher wieder, die wir als zum Thema und zur vertiefenden Einführung für sinnvoll betrachten.

VLSI - Design Techniques for Analog and Digital Circuits

Randall L. Geiger, Phillip E. Allen und Noel R. Strader, McGraw-Hill Series in Science and Engineering, 1990

CAD für die Mikroelektronik

Hans Spiro, Oldenbourg Verlag

VHDL: Hardware Description and Design

Roger Lipsett, Carl Schaefer and Cary Ussery, Kluwer Academic Press, 1989

Inhaltsverzeichnis

1	Einleitung	1
2	Entwurf integrierter Schaltungen	5
2.1	Hierarchischer und strukturierter Schaltungsentwurf	5
2.2	Repräsentationen einer Zelle	8
2.3	Aufgaben der Entwurfswerkzeuge	10
2.4	Designablauf	11
2.5	Entwurfstile	14
2.5.1	Anwender-programmierbare Logik	15
2.5.2	Gate-Arrays	18
2.5.3	Standardzellen	19
2.5.4	Makrozellen	20
2.5.5	Handlayout	21
2.5.6	Auswahlkriterien	22
2.6	Was ein Chip kostet	23
2.6.1	Kosten der Entwicklung	23
2.6.2	Kosten der Fertigung	24
3	Schaltungs-Simulation	29
3.1	Simulation auf Bauelementeniveau	29
3.2	Kirchhoffsche Gleichungen	33
3.3	Bauelementgleichungen	35
3.4	Modifizierte Knotenspannungsanalyse	37
3.5	Wechselstromanalyse	38
3.6	Dynamische Analyse	38
3.7	Linearisierung von Bauelementekennlinien	40
3.8	Ablauf einer Analog-Simulation	42
4	Logik-Simulation	45
4.1	Compilierte Simulation	46

4.2	Ereignisgesteuerte Logiksimulation	48
4.3	Behandlung von Races	49
4.4	Halbschrittverfahren	50
4.5	Modellierung des Laufzeitverhaltens	51
4.6	Scheduling	53
5	RC-Simulation	55
6	Hardware-Beschreibungssprachen	59
6.1	Motivation und Hintergrund	59
6.2	Anwendungsgebiete von HDLs	60
6.2.1	Spezifikation / Dokumentation	60
6.2.2	Simulation	60
6.2.3	Synthese	61
6.2.4	Test	61
6.2.5	Formale Verifikation	61
6.3	Charakterisierung einer HDL	61
6.4	Sprachen	64
7	Logik-Synthese	65
7.1	Logikminimierung	65
7.1.1	Abbildung von Logikfunktionen	72
8	Testfreundlicher Schaltungsentwurf	75
8.1	Grundbegriffe	75
8.2	Partitionierung und Scan-Techniken	79
8.3	Externer Test	82
8.4	Testmustergenerierung	84
8.4.1	D-Algorithmus	86
8.4.2	Testmustergenerierung für sequentielle Schaltungen	87
8.5	Selbsttest	89
8.5.1	Testmustergenerierung	89
8.5.2	Bewertung der Schaltungsantworten	90
8.6	Fehlersimulation	90
9	Layout-Ebene	93
9.1	Hardware eines Entwurfssystems	94
9.2	Der Layouteditor	95
9.2.1	Datenhaltung	95
9.2.2	Graphische Datenelemente	95

9.2.3	Editiermöglichkeiten	95
9.3	Layoutrepräsentation und Grundalgorithmen	99
9.3.1	Polygone	99
9.3.2	Bitraster	101
9.3.3	Kanten	102
9.3.4	Plane-Sweep-Algorithmus	102
9.4	Prüfung geometrischer Entwurfsregeln	104
9.4.1	Geometrische Entwurfsregeln	104
9.4.2	Übersicht über die Grundoperationen	105
9.4.3	Boole'sche Flächenoperationen	107
9.4.4	Metrische Operationen	108
9.4.5	Selektionsoperationen	109
9.4.6	Meßoperationen	110
9.4.7	Formulierung eines Regelsatzes	115
9.4.8	Schaltungsextraktion	119
9.4.9	Artwork	120
9.5	Schaltungsvergleich	123
9.5.1	Anforderungen	124
9.5.2	Algorithmen zum Schaltungsvergleich	125
9.6	Prüfung elektrischer Entwurfsregeln	126
10	Layout-Synthese	129
10.1	Symbolisches Layout	130
10.2	Prozedurales Layout	134
10.3	Hierarchisches Layout	137
10.3.1	Plazierung und Verdrahtung	138
10.3.2	Plazierung	141
10.3.3	Verdrahtung	143
10.3.4	Channel Routing	144
10.3.5	Maze-Routing	146
10.4	Layoutkompaktierung	148
A	Schaltungsentwurf mit VHDL	151
A.1	Einführung	151
A.2	Übersetzung und Simulation von VHDL	153
A.3	Organisation eines VHDL-Modells	154
A.4	Sprachelemente	154
A.5	STANDARD-Package	156
A.6	Parallele Anweisungen	157

A.7	Sequentielle Anweisungen	159
A.8	Beispiele	160
A.8.1	Volladdierer	160
A.8.2	Multiplexer	162
A.8.3	Komparator	163
A.8.4	Addierer	164
A.8.5	Multiplizierer	165

Kapitel 1

Einleitung

Viele Aufgabenstellungen erfordern heutzutage den Einsatz von Computern. Vielfach prägt der Computer dabei die Methodik in entscheidendem Maße. Dies drückt sich auch in einer Reihe von Schlagworten aus, die die Rechner-Unterstützung für bestimmte Aufgaben bezeichnen. So spricht man beispielsweise von CAD (computer-aided design), CAT (computer-aided test) oder CAQ (computer-aided quality management). Diese Techniken werden durch das bereits genannte Schlagwort CAE (computer-aided engineering) zusammengefasst.

An einem einfachen Beispiel - der Entwicklung und Produktion eines Rades - soll die Rolle des Entwurfs und der Entwurfswerkzeuge näher beleuchtet werden. Am Anfang steht zunächst eine Idee, die mit Hilfe von Entwurfswerkzeugen festgehalten und verfeinert wird. So könnte, wie in Abbildung 1.1 dargestellt, der Entwurf eines Rades in eine Steinplatte gemeißelt werden. Die Werkzeuge, die dazu benutzt werden, sind Entwurfswerkzeuge. Nach und nach werden die Fertigungsvorschriften festgelegt. So soll das Rad rund sein und ein Loch in der Mitte haben. Wenn dieser Prozess abgeschlossen ist, liegt der Entwurf fest, und man kann zur Produktion übergehen. Um nun aus einem Felsbrocken ein Rad zu meißeln, benötigt man wiederum Werkzeuge, die sogenannten Produktionswerkzeuge.

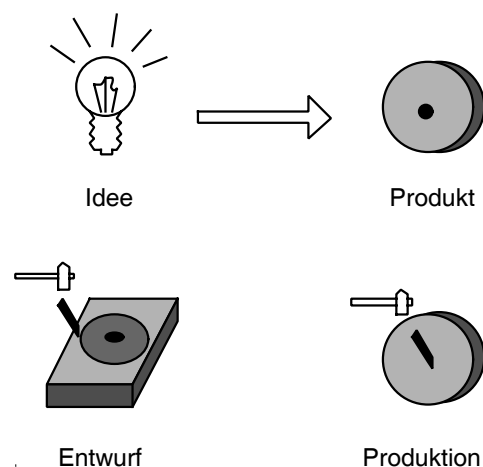


Abbildung 1.1: Unterscheidung Entwurf – Produktion.

Die Produkte, d.h. hier die mikroelektronischen Schaltungen, werden von den verschiedensten Faktoren beeinflusst. Die wichtigsten von ihnen sind Technologie und Entwurfsmethodik:

- Fertigungstechnologie

Die zur Verfügung stehende Technologie bestimmt die entstehenden Fertigungskosten. Im Laufe der Zeit wurden immer komplexere Schaltungen möglich. Dies folgt aus den ständig verkleinerten Abmessungen von Bauelementen und Leitungen auf den Chips. Ein weiterer wichtiger Kostenfaktor ist die Ausbeute an funktionierenden Schaltungen, die ebenfalls von der „Güte“ der Technologie abhängt. Auch die Leistungsfähigkeit (performance) der Schaltungen wird von der Technologie beeinflusst. Mit der Weiterentwicklung der Technologie werden höhere Schaltgeschwindigkeiten und niedrigere Verlustleistungen möglich. Neue Bauelemente bis hin zu Sensoren und Aktoren erschließen weitere Anwendungsgebiete.

- Entwurfsmethodik

Die zur Verfügung stehenden CAD-Werkzeuge beeinflussen das Produkt über die entstehenden Entwicklungskosten wesentlich. Ausgereifte effiziente Werkzeuge, die leicht zu bedienen sind, senken die Entwicklungszeit und damit auch die Zeit, die benötigt wird, um ein Produkt auf den Markt zu bringen.

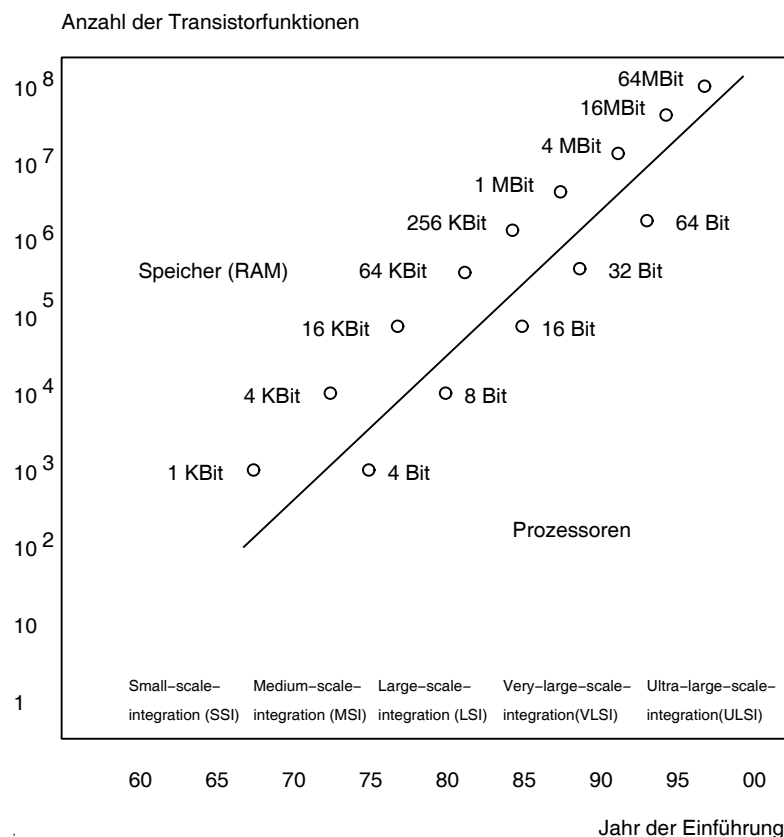


Abbildung 1.2: Entwicklung der Mikroelektronik.

Die Anfänge der Elektronik in miniaturisierter Form - der Mikroelektronik - reichen zurück bis zum Ende der Sechziger Jahre. Seitdem hat die Mikroelektronik eine stürmische Entwicklung erlebt. Über Jahrzehnte hinweg konnte die Leistungsfähigkeit der Schaltungen etwa alle drei Jahre verdoppelt werden, wobei die Preise im Wesentlichen konstant blieben. Kaum eine technische Disziplin hat eine solche Dynamik über einen längeren Zeitraum erreicht. Wenn die Automobilindustrie in den letzten 30 Jahren eine ähnliche Entwicklung durchgemacht hätte, würde ein Mercedes der S-Klasse nur einige Mark kosten. Aufgrund der ebenfalls erreichten Miniaturisierung wäre er allerdings auch nur einige Zentimeter groß. Die Abbildung 1.2 zeigt die Entwicklung für dynamische RAM-Bausteine (random access memory) und Mikroprozessoren. Die Prozessoren laufen dabei den Speicherbausteinen um einige Jahre nach. Dies liegt daran, daß die Speicher sehr regelmäßig aufgebaut sind, so daß hier der Entwurfsaufwand deutlich geringer ausfällt als bei den Prozessoren.

Mit der gestiegenen Komplexität der Schaltungen ist auch die Designaufgabe komplizierter geworden, zumal einige weitere Randbedingungen den Entwurf erschweren. So muß in der Mikroelektronik eigentlich immer das erste Design funktionieren, da man keinerlei Reparaturen an gefertigten Schaltungen ausführen kann. Ein einziger Schaltungsfehler führt zur Wiederholung des teuren Fertigungsdurchlaufs mit entsprechend geänderten Designdaten. Eine iterative Systementwicklung, wie etwa beim Aufbau einer Platine, ist nicht möglich. Dort kann man die Entwicklung quasi am lebenden Objekt durchführen und solange löten, bis die Schaltung funktioniert. In der Mikroelektronik hingegen muß bereits der Entwurf - hier beispielsweise der Schaltplan - durch Simulation und andere Methoden vor der Fertigung eingehend verifiziert werden.

Neben der Forderung, beim ersten Mal alles richtig machen zu müssen, kommt außerdem ein ganz erheblicher Zeitdruck hinzu. Dieser resultiert aus der Tatsache, daß in der Mikroelektronik für jede Produktklasse jeweils nur ein sehr kurzes Zeitfenster existiert, innerhalb dessen man mit einer betreffenden Schaltung Geld verdienen kann. Dies läßt sich besonders gut mit dem Konzept der Lebenszyklen von Produkten - hier von Schaltungen - illustrieren. Zu diesem Zweck werden die Erträge, die mit einem Produkt erwirtschaftet werden, gegen die Zeit aufgetragen, siehe Abbildung 1.3. Je kürzer ein solcher Lebenszyklus ist, desto wichtiger ist es, als erster mit dem betreffenden Produkt auf dem Markt zu sein (time to market). Mittlerweile werden bei einigen Produktklassen „Modellwechsel“ schon im Jahresrhythmus vorgenommen. Damit können heute schon einige Monate Verzögerung bei der Markteinführung den wirtschaftlichen Erfolg vereiteln.

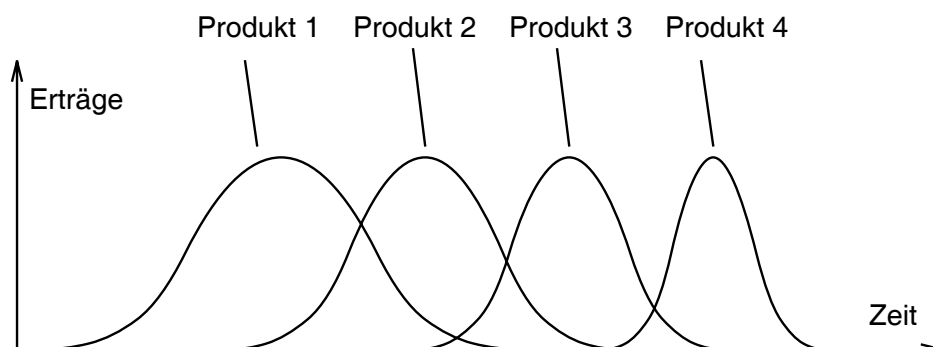


Abbildung 1.3: Kürzer werdende Lebenszyklen von Produkten.

Wesentlich für die Umsetzung der Methodik ist die eingesetzte CAE-Software, wobei man die beiden Konzepte kaum voneinander trennen kann. Vielfach kauft man mit dem Werkzeug auch gleich die Methodik, ohne sich dessen bewußt zu sein. CAE-Werkzeuge helfen beim Entwurf. Sie analysieren Schaltungen, suchen Fehler und automatisieren den Entwurfsablauf zumindest teilweise. Ohne Rechnerunterstützung ist die Mikroelektronik nicht denkbar.

Kapitel 2

Entwurf integrierter Schaltungen

2.1 Hierarchischer und strukturierter Schaltungsentwurf

Es hat sich im Laufe der Entwicklung der CAE-Methoden gezeigt, daß eine Strukturierung des Entwurfsprozesses nach Abstraktionsebenen besonders geeignet ist, um die Schaltungskomplexität in den Griff zu bekommen. Eine Schaltung wird danach in verschiedenen Abstraktionen betrachtet. Typische Abstraktionsebenen sind beispielsweise das Bauelementeniveau mit Transistoren, Widerständen etc., das Gatter- oder Logikniveau mit UND-, ODER- und NICHT-Gattern etc. und die Registertransferebene mit Registern, Rechenwerken (algorithmic and logic unit, ALU) und vielen anderen Blöcken. Abbildung 2.1 stellt die typischen Abstraktionsebenen dar.

Um eine Schaltung in verschiedenen Abstraktionen darzustellen, müssen Teilschaltungen definiert werden, die meist ein einheitliches Abstraktionsniveau aufweisen. Diese Teilschaltungen bestehen in der Regel aus funktional oder strukturell zusammengehörigen Komponenten und werden Zellen genannt. Eine Zelle läßt sich auch induktiv anhand ihrer Bestandteile definieren. Die Komponenten einer Zelle sind ihrerseits Zellen oder Primitive wie Transistoren (Schaltplan) oder Polygone (Layout). Vier Grundsätze kennzeichnen mit Hilfe des Zellkonzeptes einen effizienten Schaltungsentwurf.

- Entwurfshierarchie

Die Entwurfshierarchie folgt direkt aus dem Zellkonzept. Eine Zelle kann nur Primitive oder Aufrufe anderer Zellen enthalten. So besteht eine NAND-Zelle aus den zugehörigen Transistoren. Eine Flipflop-Zelle hingegen könnte NAND-Zellen enthalten; eine Register-Zelle wiederum Flipflop-Zellen. Diese Hierarchie setzt sich fort bis hin zu einer „obersten“ Zelle, die die Gesamtschaltung bezeichnet. Würde man alle Aufrufe von Zellen in dieser „obersten“ Zelle auflösen, indem man sie durch die aufgerufene Zelle selbst ersetzt, und dies sukzessive fortführen bis kein Zellaufruf mehr vorliegt, dann enthielte die „oberste“ Zelle alle Transistoren der Schaltung. Dies sind die Elemente, die sich nicht weiter auflösen lassen. Dieses Auflösen der Hierarchie wird auch Expansion oder Flachmachen (expansion, explosion, flattening) der Zelle genannt.

Eine Schaltungshierarchie entspricht einem Baum, siehe Abbildung 2.2. Die Wurzel des Baums ist die oberste Zelle, die Blätter des Baums bestehen aus Primitiven. Eine Instanz

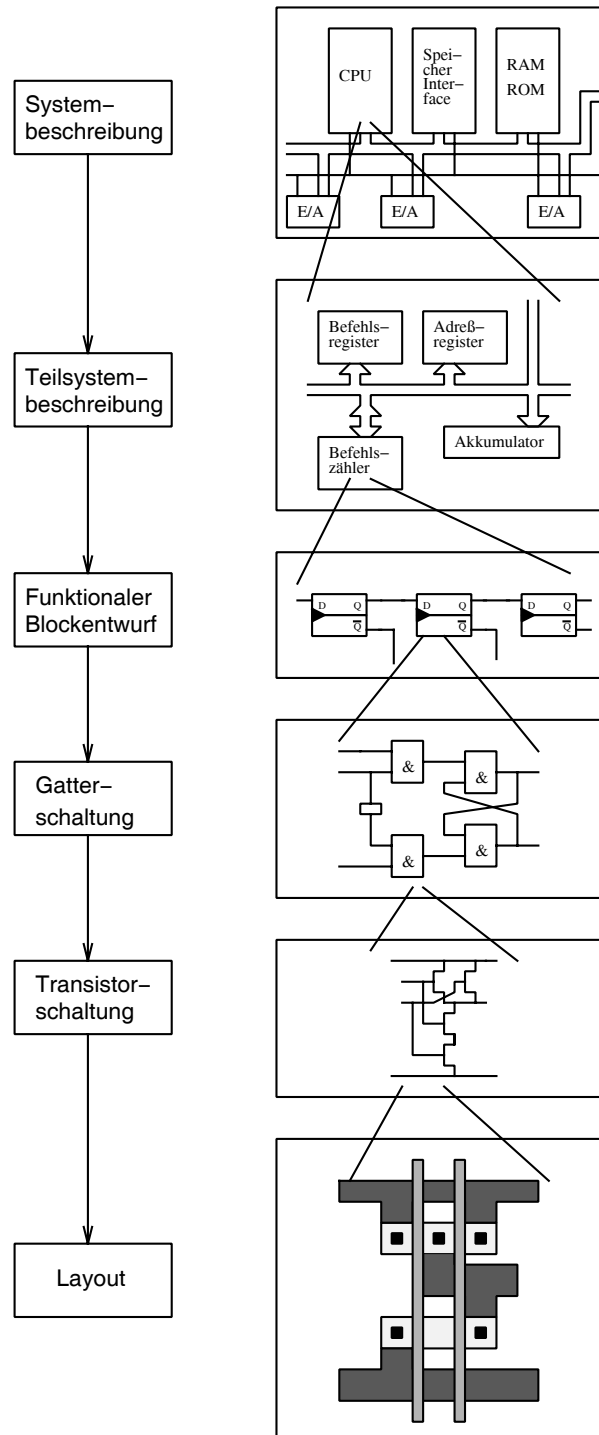


Abbildung 2.1: Abstraktionsebenen im Schaltungsentwurf.

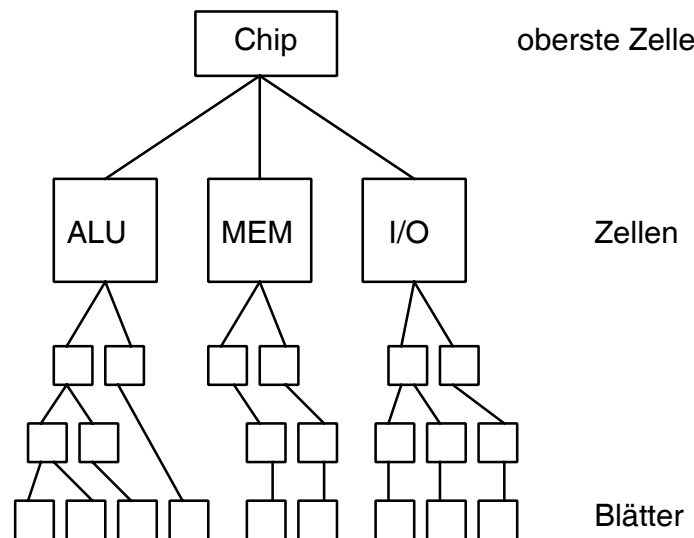


Abbildung 2.2: Beispiel einer Schaltungshierarchie.

einer bestimmten Zelle hat grundsätzlich genau einen Vater in der Hierarchie. Das Hierarchiekonzept vereinfacht den Entwurf wesentlich, da die Schaltungen durch die Hierarchie weitaus übersichtlicher und platzsparender repräsentiert werden können. So wird in einer hierarchischen Schaltung, die 50 Flipflops enthält, ein Flipflop nur einmal entworfen, aber 50 mal aufgerufen. Das Flipflop mag unter anderem aus vier NAND-Gattern bestehen. Dieses wurde seinerseits nur einmal entworfen, aber jetzt schon 200 mal eingesetzt.

- Entwurfsmodularität

Ein Entwurf sollte entsprechend der Struktur der Schaltung sukzessive in kleinere Schaltungsbereiche aufgespalten werden, die entsprechend leichter zu entwerfen sind. Entscheidend dabei ist, daß die Interaktion zwischen den Schaltungsmodulen möglichst gering bleibt, so daß nur wenig Aufwand für die Kopplung der Module eingesetzt werden muß. Denkbar sind hier die verschiedensten Strategien, wie beispielsweise eine Modularisierung nach funktionalen Gesichtspunkten, d.h. alles was zu einer bestimmten (Teil-)Funktion einer Schaltung gehört, wird einem bestimmten Modul zugeschlagen.

- Entwurfsregularität

Schaltungen sollten möglichst so entworfen werden, daß ihre Zellen zwar nur einmal entworfen, aber möglichst oft aufgerufen werden. Mit dieser Strategie kann ein erheblicher Teil des Entwurfsaufwands eingespart werden. Außerdem steigt die Zuverlässigkeit der Schaltung, da die Anzahl der Gelegenheiten zum Einbau von Fehlern sinkt. Die Regularität eines Entwurfs läßt sich anhand eines sogenannten Regularitätsfaktors messen, der die Anzahl der explizit gezeichneten Transistoren (Schaltplan oder Layout) in Relation zur Gesamtanzahl der Transistoren setzt.

- Entwurfslokalität

Die Bedeutung der Entwurfslokalität ist bereits bei den vorausgehenden Strategien angekommen. Entscheidend bei aller Modularisierung – wie ganz allgemein bei einem reduktionistischen Ansatz – ist es, daß die Wechselwirkung zwischen zwei Modulen möglichst gering ist. Für eine hierarchische Schaltung ergibt sich die Wechselwirkung beispielsweise durch die Anschlüsse mit deren Hilfe eine aufgerufene Zelle mit ihrer Umgebung verbunden wird. Insbesondere sollte der Entwurf von Zellen möglichst ohne die Betrachtung der Umgebung der Zelle durchführbar sein.

Anhand eines Beispiels soll diese Begriffsbildung gefestigt werden. Für den Entwurf einer Kutsche würde die Hierarchie dazu eingesetzt, die Entwurfsaufgabe bis auf beherrschbare Einheiten – z.B. ein Rad – herunterzubrechen. Sinnvolle Module entsprechend der Modularität wären hier die Achse mit zwei Rädern, oder der Aufbau mit Sitz. Die Regularität kommt dadurch zum Ausdruck, daß man ein Rad nur einmal entwirft, aber mehrfach einsetzt. Die Lokalität schließlich zeigt sich für das Rad darin, daß seine wichtigsten Eigenschaften (rund, Nabe im Zentrum des Rades etc.) vom übrigen Aufbau der Kutsche völlig unabhängig sind.

2.2 Repräsentationen einer Zelle

Es hat sich im Laufe der Zeit als notwendig erwiesen, verschiedene Sichten oder Repräsentationen von Zellen für verschiedene Aufgaben vorzusehen. Je nachdem welche Operationen durchgeführt werden sollen, werden unterschiedliche Repräsentationen der Zelle verwendet. Man unterscheidet drei Gruppen von Sichten: Verhaltensbeschreibungen, Strukturbeschreibungen und Geometriebeschreibungen. Diese lassen sich jeweils auf verschiedenen Hierarchieebenen beobachten. Die Tabelle 2.3 zeigt die wichtigsten Sichten auf verschiedenem Hierarchieniveau auf.

	Verhalten, Funktion	Struktur	Geometrie, Layout
System	Spezifikation,	Prozessor, Memory	Partitionierung
Algorithmus	I/O-Algorithmen	Module	Floorplan
Register-Transfer Ebene	RT-Beschreibung, (Datenfluß)	Funktionseinheiten	Blöcke Makrozellen
Logik	Boole'sche Gleichungen	Gatter, Flipflops	Layoutzellen
Bauelemente	Kirchhoff'sche Gleichungen Bauelementegleichungen	Transistoren, Widerstände etc.	Polygone

Abbildung 2.3: Sichten einer mikroelektronischen Schaltung.

Die abstrakteste Sicht für das Verhalten einer Schaltung ist die Spezifikation, die ja gerade Informationen darüber enthält, was die Schaltung unter den verschiedensten Randbedingungen lei-

sten soll. Etwas detaillierter sind die I/O-Algorithmen, die zumindest das Verhalten an den Klemmen einer Schaltung wiedergeben. Register-Transfer- oder auch RT-Beschreibungen repräsentieren eine Schaltung durch seine speichernden Elemente und die Logikfunktionen zwischen ihnen. All dies läßt sich auf Boole'sche Gleichungen abbilden, die wiederum für die Implementierung durch elektronische Bauelemente auf Kirchhoff'sche Gleichungen und Bauelementgleichungen zurückgeführt werden können. Im Strukturbereich geben die Repräsentationen auf den verschiedenen Abstraktionsebenen einfach die Hierarchie der Schaltung wieder. Für die Geometrie des Layouts schließlich wird auf oberster Ebene dessen Aufteilung oder Partitionierung auf verschiedene Teillayouts oder Chips betrachtet. Der Floorplan liefert bereits eine grobe Anordnung der Layoutblöcke in der Ebene. Die Layouts der Blöcke und (Makro-)Zellen entsprechen der geometrischen Repräsentation der Funktionseinheiten, Gatter und Flipflops im Strukturbereich. Die Polygone schließlich umschreiben die Flächen des Layouts.

Die oben dargestellte Einordnung der Sichten kann auch mit Hilfe eines Y- oder Gajski-Diagramms dargestellt werden, Abbildung 2.4. Anhand eines solchen Diagramms kann auch der Verlauf einer Schaltungssynthese dargestellt werden. So startet die Synthese einer Schaltung typischerweise mit einer Register-Transfer (RT) Beschreibung und generiert daraus letztendlich ein Layout.

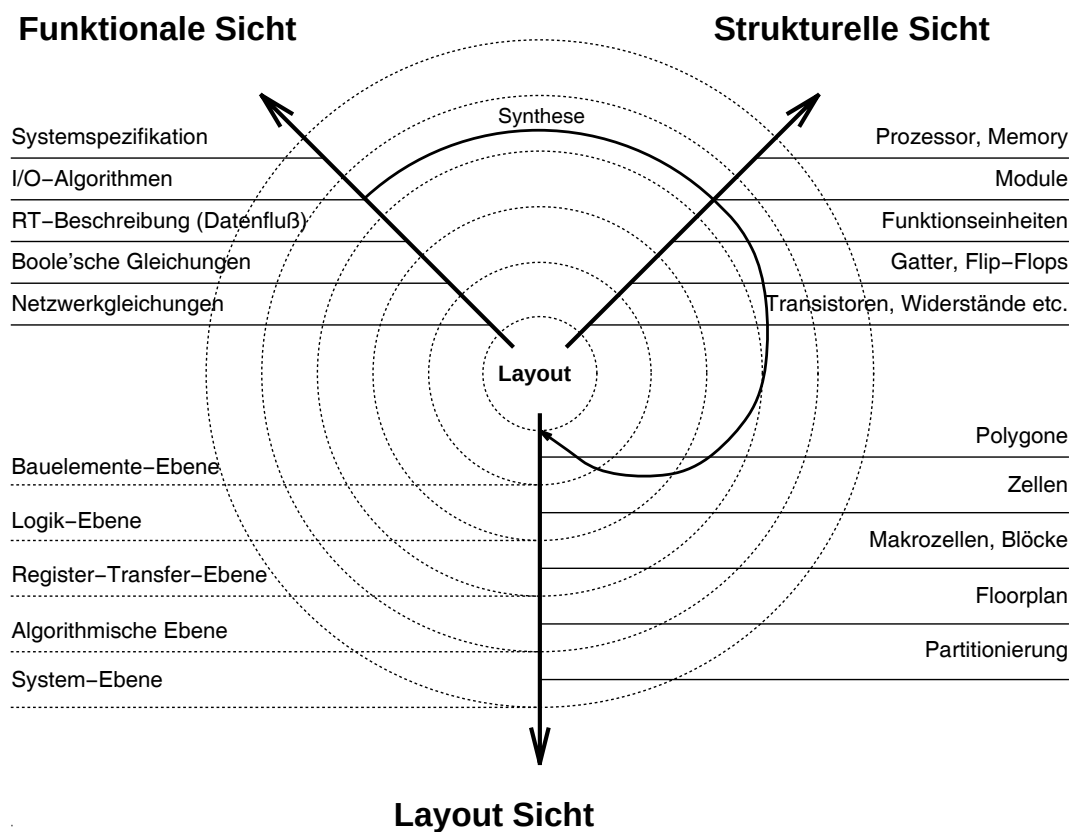


Abbildung 2.4: Gajski-Diagramm.

2.3 Aufgaben der Entwurfswerkzeuge

Der Aufgabenbereich von CAE-Werkzeugen zum Entwurf integrierter Schaltungen läßt sich in die folgenden sieben Bereiche einteilen:

- **Schaltungseingabe**

Diverse Werkzeuge stehen zur Eingabe von Schaltungen zur Verfügung. Falls Schaltpläne erstellt werden, spricht man von einem Schaltplan- oder Schematic-Editor. Für die Eingabe von Geometrie benötigt man einen Layout-Editor.

- **statische Verifikation**

Die Hauptaufgabe solcher Werkzeuge besteht darin, eine Schaltungsrepräsentation bezüglich einer Menge von Regeln zu überprüfen. So sollte für ein Layout gelten, daß die Aluminium-Leiterbahnen eine gewisse Breite aufweisen, da ansonsten eine leitende Verbindung nicht gewährleistet werden kann. Eine Verifikation kann aber auch in einer Konsistenzprüfung gegenüber einer Referenzschaltung bestehen. Hier würde überprüft, ob zwei aus verschiedenen Quellen stammende Schaltungen exakt gleich sind.

- **dynamische Verifikation**

Hier liegt die Intention darin, das Verhalten einer Schaltung zum Beispiel durch eine Schaltungs- oder Logik-Simulation zu überprüfen. Eine andere Anwendung wäre die Überprüfung des Zeitverhaltens durch eine Timingsimulation.

- **Optimierung**

Optimierungswerkzeuge verändern die Schaltung, so daß sie den gewünschten Eigenschaften näher kommt. Hierunter wäre beispielsweise die Minimierung der Logikfunktionen einer Schaltung zu fassen.

- **Synthese**

Nachdem die vorangegangenen Werkzeuge die Schaltungen eines Entwicklers überprüfen oder verbessern, sollen ausgehend von abstrakteren Schaltungsbeschreibungen detailliertere generiert werden. Dies ist für verschiedene Repräsentationen möglich. Zwei Formen der Synthese sind dabei vorherrschend: die Transformation von einer Register-Transfer-Beschreibung in eine Gatter-Beschreibung und von einer Gatter-Beschreibung in das betreffende Layout.

- **Design Assistants**

Hierbei handelt es sich in der Regel um Expertensysteme, die dazu dienen, einen Entwurfsablauf oder eine Lösung auszuwählen. Es werden also nicht Schaltungsbeschreibungen verarbeitet, sondern Informationen über die Handhabung von Schaltungsbeschreibungen. Solche Werkzeuge stammen vielfach aus dem Forschungsbereich, haben sich aber in der kommerziellen Schaltungsentwicklung bislang nicht durchgesetzt.

- Frameworks

Frameworks sind nicht im engeren Sinne Entwurfswerkzeuge, sondern Hilfsmittel mit denen die Entwurfswerkzeuge kombiniert und zu integrierten Systemen zusammengebunden werden. Ziel ist unter anderem der freie Datenaustausch zwischen den verschiedensten Werkzeugen. In Abbildung 2.5 wird die graphische Oberfläche eines Frameworks (Design Framework II, Cadence Design Systems) und hier insbesondere die Kopplung von Schaltplaneingabe und Gattersimulation gezeigt.

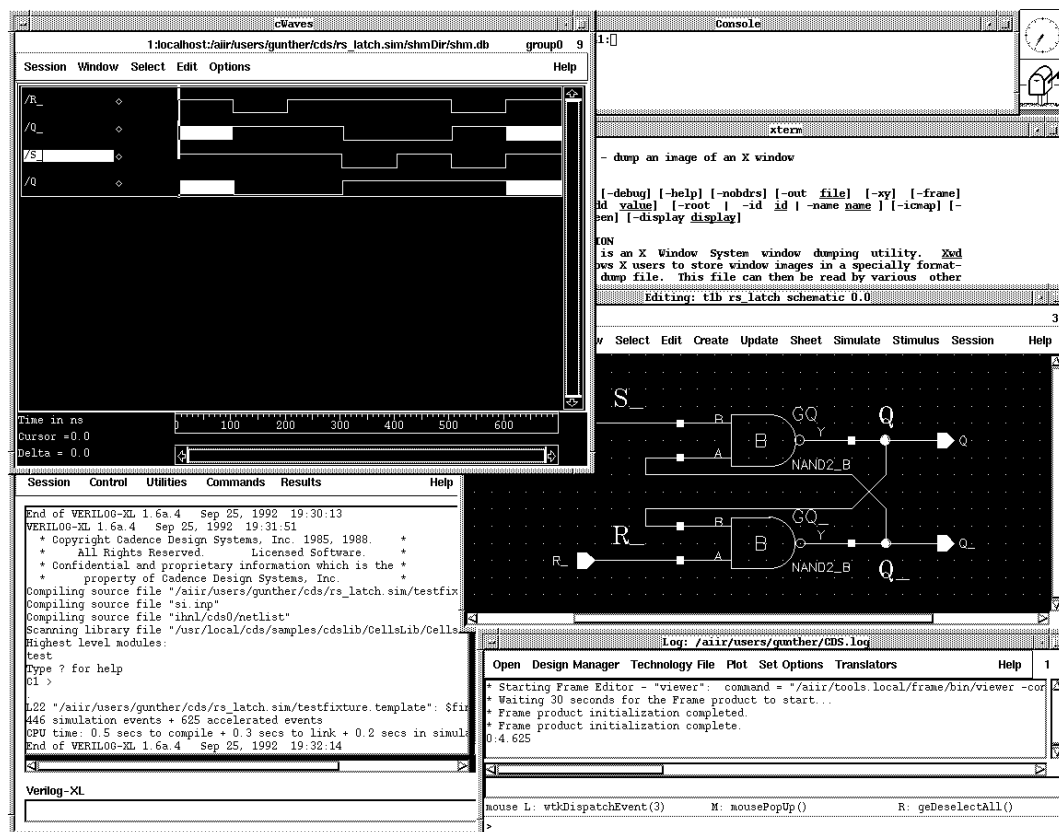


Abbildung 2.5: Kopplung von Schaltplaneingabe und Gattersimulation in einem Framework.

2.4 Designablauf

Die oben beschriebenen Entwurfswerkzeuge gliedern sich in den Designablauf, d.h. in die Sequenz der Aktionen zum Chipentwurf ein. Nimmt man die Abstraktion der jeweils betrachteten Schaltungsteile zum Maßstab, so kann man zwei Entwufsrichtungen von oben nach unten (Top-Down) und von unten nach oben (Bottom-Up) in der Abstraktion definieren. Von oben nach unten bedeutet hier von abstrakteren Beschreibungen zu weniger abstrakten. Die Gegenrichtung definiert sich analog.

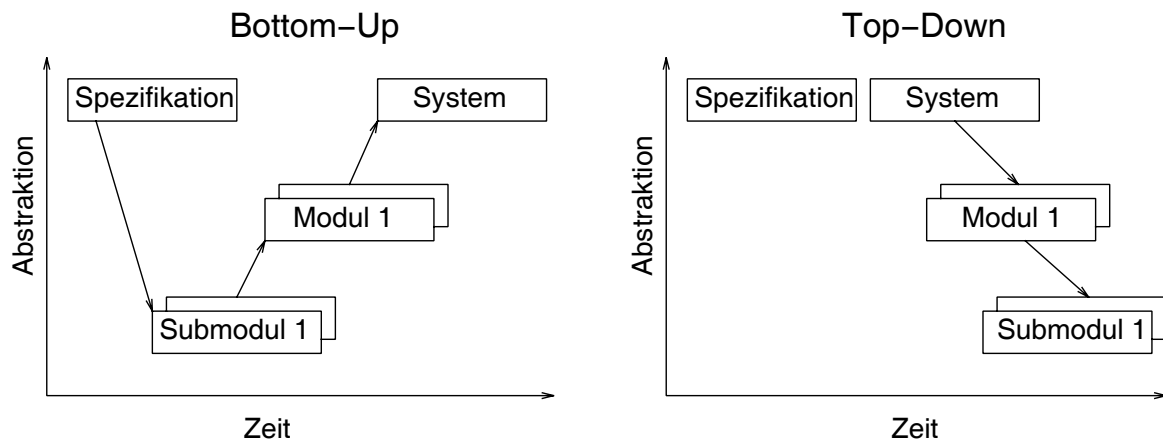


Abbildung 2.6: Designablauf: Top-Down versus Bottom-Up.

Der „Bottom-Up“-Ansatz wurde bis etwa 1985 recht intensiv verfolgt. Dies vor allem deshalb, weil ein Schaltungsentwurf bis dahin typischerweise in Schaltplänen formuliert wurde. Simulieren kann man solche Beschreibungen erst dann, wenn alle Subkomponenten bis zur Ebene der Gatter oder Transistoren entworfen sind. Da jeder Entwurfsschritt in der Regel durch eine Simulation verifiziert wird, ist es hier unabdingbar, mit kleineren Blöcken anzufangen und diese sukzessive zu größeren Schaltungen zusammenzubauen. Dies führt allerdings dazu, daß Entwurfsfehler und -probleme auf der Systemebene erst sehr spät auffallen. Damit gestaltet sich deren Beseitigung unter Umständen recht schwierig. Im schlimmsten Fall erweist sich die Architektur der Schaltung als untauglich, so daß alle vorher geleisteten Arbeiten mit einem Schlag überflüssig geworden sind. Außerdem ist es gerade bei komplexeren Schaltungen oftmals nicht ganz einfach, mit dem sukzessive aufgebauten System die ursprünglich gewünschte Spezifikation zu treffen. Dies wird durch die Abbildung 2.6 illustriert, die für beide Designrichtungen die Abstraktion der entworfenen Komponenten gegen die Zeit aufträgt. In beiden Fällen startet man mit einer Spezifikation, die jeweils auf sehr abstraktem Niveau angesiedelt ist. Dies folgt schon aus der Aufgabe einer Spezifikation, die im Wesentlichen beschreibt, was die Schaltung unter verschiedenen Randbedingungen tun soll, aber nicht, wie dies erreicht wird. Bei der „Bottom-Up“-Strategie springt man dann auf das unterste Abstraktionsniveau, z.B. Transistoren oder Gatter, und arbeitet sich in der Abstraktion hoch, bis ein Systementwurf vorliegt.

Im Gegensatz dazu wird beim „Top-Down“-Ansatz aus der Spezifikation direkt eine Systembeschreibung entwickelt, die im Wesentlichen das Verhalten des Systems beschreibt. Dies wurde erst durch die Einführung der Hardware-Beschreibungssprachen möglich. Davon ausgehend wird der Systementwurf immer weiter verfeinert, bis man schließlich bei den Primitiven, d.h. bei den Transistoren und Gattern, ankommt. Insbesondere gibt es seit einigen Jahren die Möglichkeit diesen Prozess teilweise zu automatisieren. So kann man aus Register-Transfer-Beschreibungen Gatternetzlisten synthetisieren, wodurch sich der Entwurfsprozess beschleunigt. Allerdings ist auch der „Top-Down“-Ablauf nicht ohne Probleme. Bei der physikalischen Umsetzung des Systementwurfs ergeben sich oftmals Effekte, wie etwa Verzögerungszeiten, die Modifikationen am Systementwurf erfordern. Damit bewegt man sich zeitweise aufwärts in der Abstraktion. Trotzdem bleibt die Hauptrichtung Top-Down insgesamt erhalten.

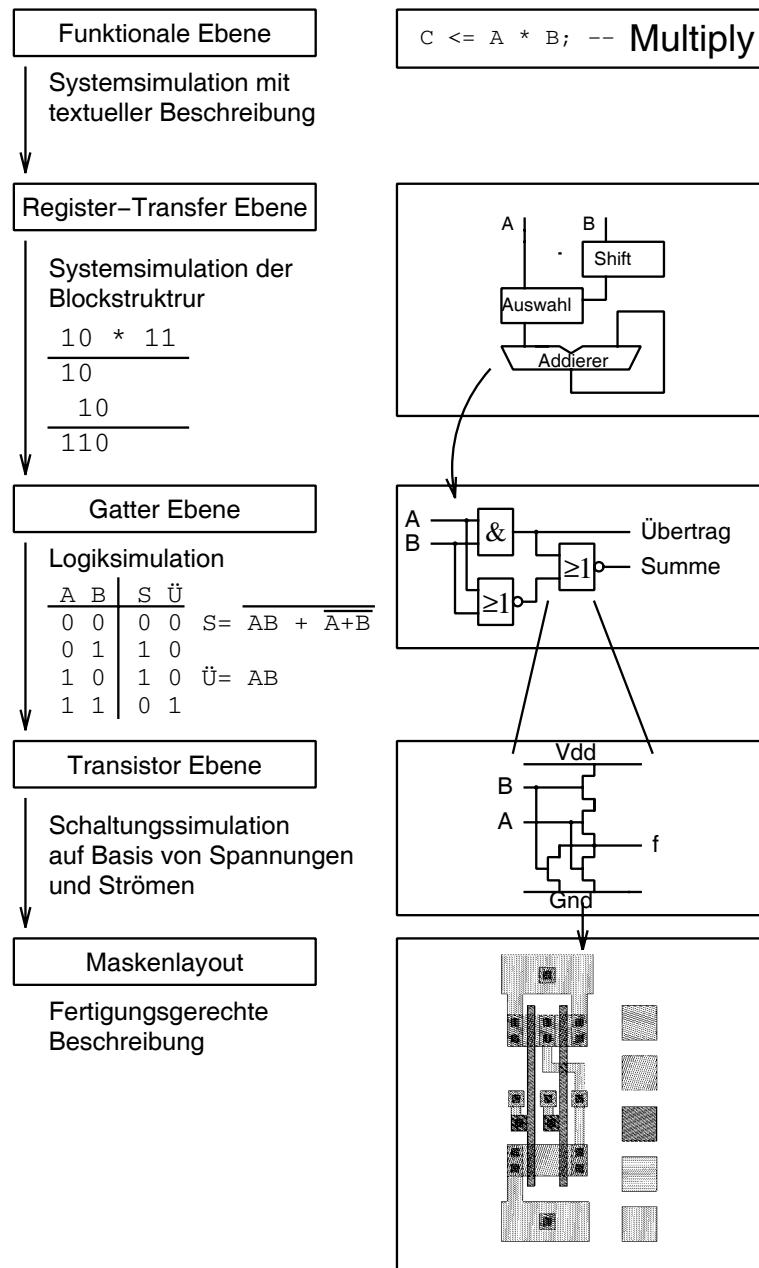


Abbildung 2.7: Designablauf.

Um die „Top-Down“-Methodik an einem Beispiel zu illustrieren, zeigt die Abbildung 2.7 den fiktiven Ablauf der Entwicklung einer Multiplizierer-Schaltung. Zunächst wird auf der Verhaltensebene die Funktion des Multiplizierers beschrieben. Die Multiplikation wird im Rahmen des Designablaufs auf die Addition und einige andere Operationen zurückgeführt, so daß eine Beschreibung auf der Register-Transfer-Ebene entsteht, die von Registern und Boole'schen Operationen zwischen ihnen bestimmt ist. Diese Boole'schen Funktionen können auf Logikgatter zurückgeführt werden, denen schließlich ein Layout zugeordnet wird. Von diesem Ablauf können die Übergänge von der Register-Transfer-Ebene zur Gatter-Ebene und von der Gatter- zur Layout-Ebene durch

Logik- und Layoutsynthese automatisiert werden. Damit ist für Standard-Digitalschaltungen der Entwurf bereits mit der Erstellung einer synthesesfähigen Register-Transfer-Beschreibung abgeschlossen.

2.5 Entwurstile

Der Entwickler kann für die Realisierung seiner Schaltung unter einer Vielzahl von Alternativen wählen, die sich in ihren Parametern wie Stückkosten, Entwicklungskosten, Fertigungskosten etc. unterscheiden. Für Schaltungen, die in hohen Stückzahlen benötigt werden, kann entsprechend mehr Aufwand getrieben werden. Bei kleinen Stückzahlen rechnet es sich häufig besser, einen entsprechend adaptierbaren Standard-Baustein zu wählen.

Diese Entwurstile gliedern sich in halb-kundenspezifische (semi-custom) und voll-kundenspezifische Schaltungen (full-custom). Zur ersten Gruppe gehören die anwender-programmierbare Logik und die Gate-Arrays. Zur zweiten die Standardzellschaltungen, Makrozellentwürfe und Schaltungen, die im Handlayout entworfen wurden.

Eine weitere wichtige Unterscheidung liefert der Begriff des ASIC (application specific integrated circuit), der eine Schaltung beschreibt, die speziell für einen Kunden entwickelt wurde. Im Gegensatz dazu spricht man von Standardbausteinen, wenn es sich um Schaltungen handelt, die für viele Kunden interessant sind, wie etwa Mikroprozessoren oder Speicher.

Zunächst soll im Rahmen einer ersten Einordnung die typische Komplexität solcher Schaltungen dargestellt werden, Abbildung 2.8.

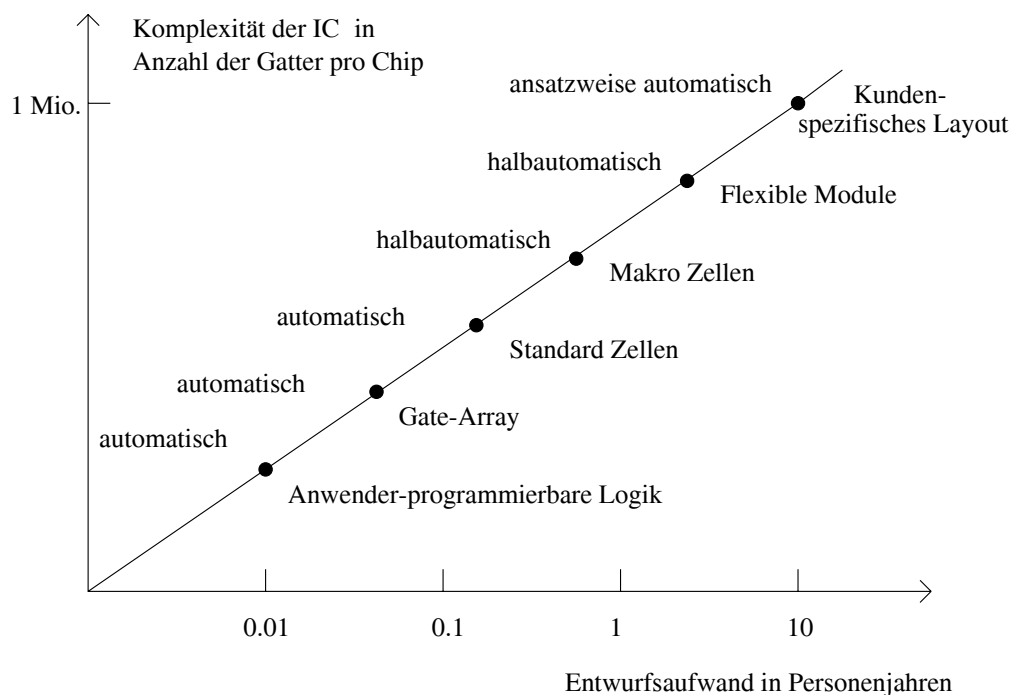


Abbildung 2.8: Schaltungskomplexitäten verschiedener Entwurstile.

2.5.1 Anwender-programmierbare Logik

Diese Art der Schaltungsrealisierung basiert darauf, einen Fundus von Logikfunktionen auf einem Chip zur Verfügung zu stellen und das gewünschte Verhalten der Schaltung durch eine geeignete „Programmierung“ einzustellen. Damit ist nicht die Programmierung im Sinne eines C- oder Assembler-Programms gemeint. Vielmehr geht es hier um zweierlei:

- Universelle Logikblöcke sind so einzustellen, daß die gewünschten Logikfunktionen realisiert werden.
- Verbindungsleitungen zwischen Logikblöcken sind zu unterbrechen oder durchzuschalten.

Die Realisierung dieser Bausteine folgt der Idee, daß man sich ein Logikgatter mit zwei Eingängen (z.B. UND, ODER) auch als Minispeicher mit zwei Adressleitungen und einer Datenleitung vorstellen kann. Die Logikwerte an den Adressleitungen können vier Kombinationen annehmen für jeweils ein Ergebnisbit zur Verfügung steht. Durch geeignetes Setzen dieses Speichers lassen sich damit alle Logikgattertypen mit zwei Eingängen darstellen. Dasselbe gilt auch für größere Logikblöcke, für die entsprechend mehr Speicher zur Verfügung stehen muß. Die Programmierung entspricht damit im wesentlichen der Speicherung der entsprechenden Einträge. Dasselbe gilt für die Information, welche Schalter in der Leitungsmatrix geöffnet und welche geschlossen sind. Diese Information wird mit den typischen Methoden zur reinen Datenspeicherung auf einem Chip festgehalten:

- Durchbrennen von Schmelzsicherungen, wie beim PROM (programmable read only memory)
- Ladungsspeicherung, wie beim EEPROM (electrically erasable programmable read only memory)
- D-Latches, wie beim statischen RAM (random access memory)

Anwender-programmierbare Logikbausteine oder auch PLDs (programmable logic device) lassen sich grob in folgende Klassen einteilen:

1. PLA (programmable logic array)

Ein PLA realisiert zweistufige Boole'sche Funktionen, die in der Regel durch Summen von Produkten beschrieben werden. Es besteht aus einer UND- und einer ODER-Matrix, die jeweils programmierbar sind. Die UND-Matrix verknüpft die invertiert und nicht-invertiert zur Verfügung stehenden Eingangsvariablen zu beliebigen Produkttermen. Dies wird dadurch erreicht, daß die Kreuzungspunkte der Matrix programmierbar sind, d.h. man kann bei Bedarf eine Verbindung zwischen den sich kreuzenden Leitungen herstellen, siehe Abbildung 2.9. Darüberhinaus ist die ODER-Matrix ebenfalls frei programmierbar, so daß man die Produktterme beliebig zu Logikfunktionen zusammenstellen kann. Insbesondere ist es möglich, einmal generierte Produktterme mehrfach zu nutzen, was insgesamt den Hardwareaufwand senkt.

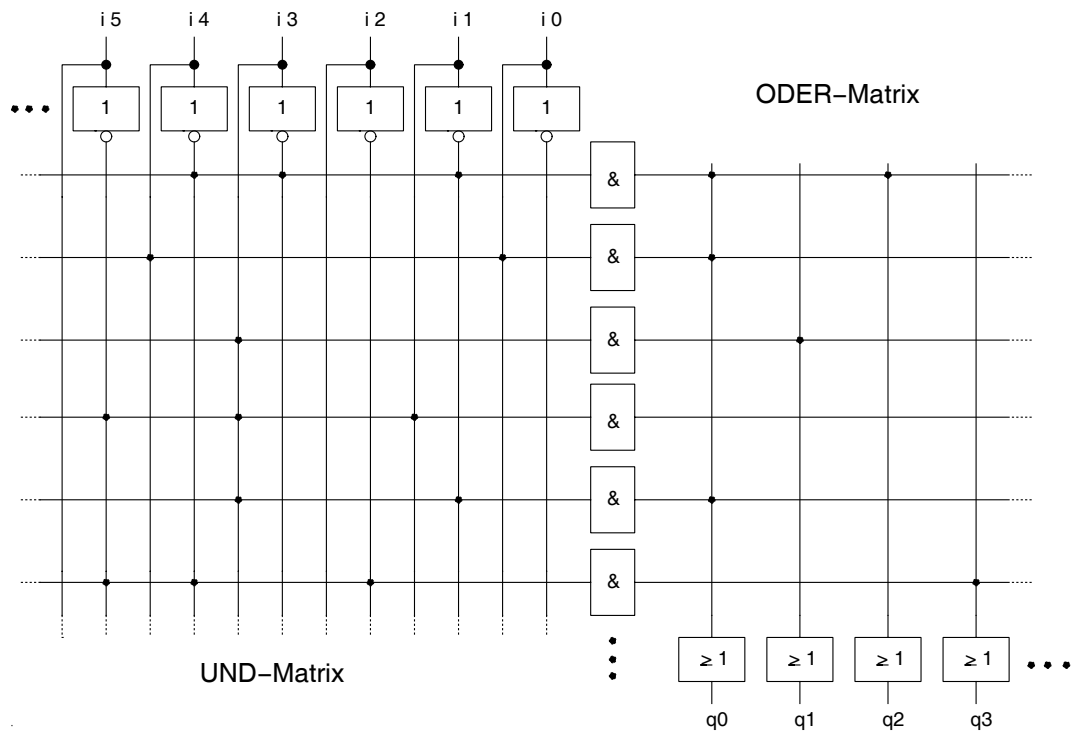


Abbildung 2.9: Struktur eines PLA.

Neben der Implementierung der zweistufigen Logikfunktion sind mit den Ausgängen der ODER-Matrix oftmals zusätzliche Blöcke verbunden, die eine bitweise Speicherung der Logikwerte und deren Zurückführung auf die Eingänge der UND-Matrix gestatten. Damit lassen sich mit diesen Bausteinen recht einfach Zustandsautomaten realisieren.

2. PAL/GAL (programmable array logic / generic array logic)

Ein PAL oder GAL entspricht einem PLA, bei dem man auf die frei programmierbare ODER-Ebene verzichtet und stattdessen einige fest verdrahtete ODER-Gatter einsetzt. Dies wirft zunächst einige Beschränkungen auf. So ist die Anzahl der Eingänge der ODER-Gatter in der Regel auf 4 bis 8 begrenzt. Außerdem lassen sich die Produktterme anders als beim PLA nicht mehrfach einsetzen. Dafür sinkt der Schaltungsaufwand im Vergleich zum PLA beträchtlich.

3. CPLD (complex PLD)

Bei CPLDs liegt die Idee darin, einige komplexere Logikblöcke zur Verfügung zu stellen, und diese mit einer Schaltermatrix zu verbinden. Die Logikblöcke können ihrerseits wiederum aus einem PAL bestehen. Der effektive Einsatz dieser PLDs hängt sehr von der Anwendung ab. In einigen Fällen kann die benötigte Funktion nicht optimal auf die Funktionsblöcke abgebildet werden, so daß hier ein Großteil der auf dem Baustein vorhandenen Logik ungenutzt bleibt. Andererseits lassen sich, wie auch bei PLAs und PALs, die ergebenden Laufzeiten recht gut abschätzen, da die Verbindungsstruktur einfach und regelmäßig aufgebaut ist.

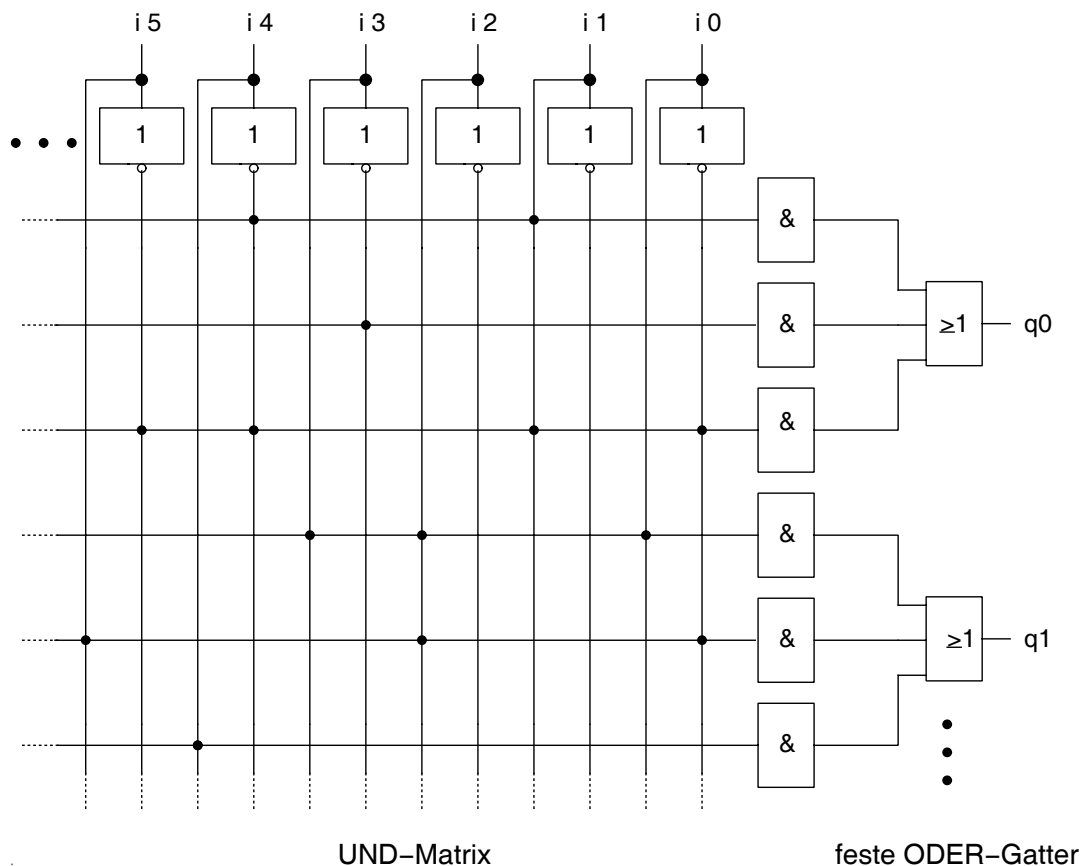
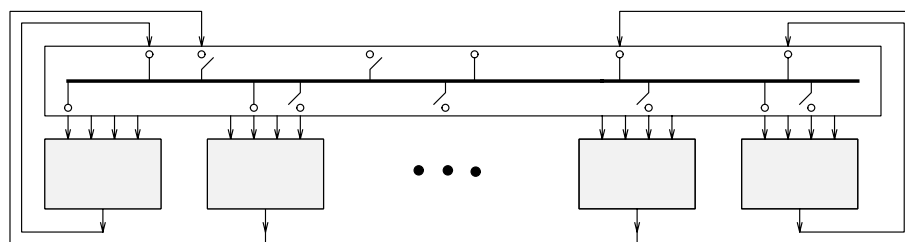


Abbildung 2.10: Struktur eines PAL/GAL.

4. FPGA (field programmable gate array)

Bei einem FPGA ist ein Array von Logikblöcken mit kleiner Eingangszahl (z.B. 4-10) in der Ebene angeordnet. Diese Blöcke emulieren die Funktion einiger weniger Gatter. Die Verbindungen zwischen ihnen werden durch ein hierarchisches Leitungsnetz bewerkstelligt, das wie die Funktion der Blöcke vom Anwender programmiert wird. Diese Programmierung besteht in der Zuordnung der Funktionen zu den Blöcken, also einer Art Platzierung (placement), und der Bereitstellung der Verbindungen durch eine geeignete Zusammenschaltung diverser Leitungssegmente, also einer Art Verdrahtung (routing). Durch die hier typische



Mehrstufigkeit der Logik ist das Laufzeitverhalten aus der Logikbeschreibung und dem Datenblatt allein nicht vorhersagbar, da es entscheidend vom jeweiligen Routing abhängt. Es sollte daher durch Simulationen überprüft werden. Abbildung 2.12 zeigt die Struktur eines solchen FPGA-Bausteins.

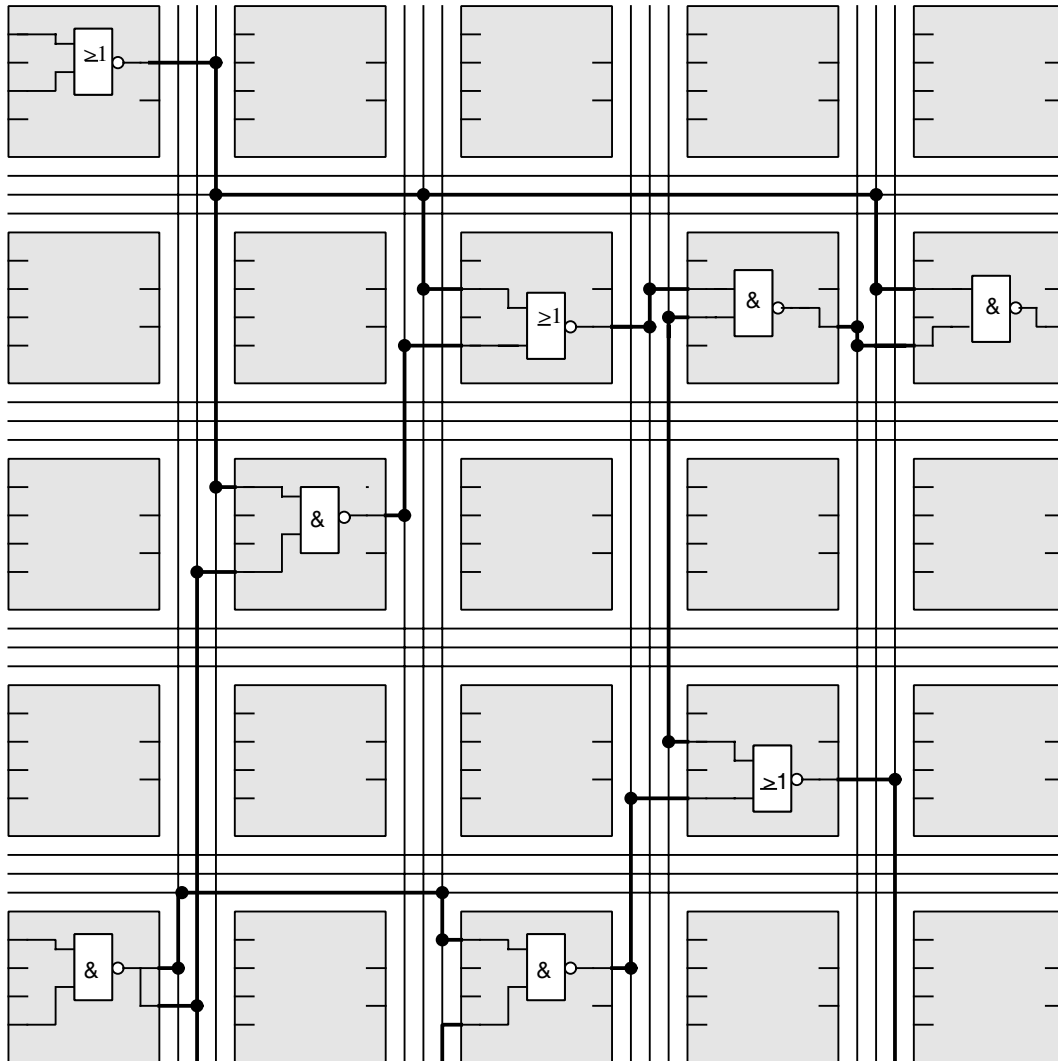


Abbildung 2.12: Struktur eines FPGA.

2.5.2 Gate-Arrays

Hier geht man von vorfabrizierten Chips - den Mastern - aus, die bereits alle Prozessschritte außer der Strukturierung der Metallschichten durchlaufen haben. Pro Kundensaltung werden daher nur 2 bis 4 Fabrikationsmasken benötigt, gegenüber 12 bis 16 wie etwa beim Standardzellentwurf. Auf fest vorgegebenen Plätzen enthalten die Master in einer streifen- oder matrixartigen Anord-

nung die Grundzellen, welche aus wenigen (2-7) nicht oder nur teilweise miteinander verbundenen Transistoren bestehen.

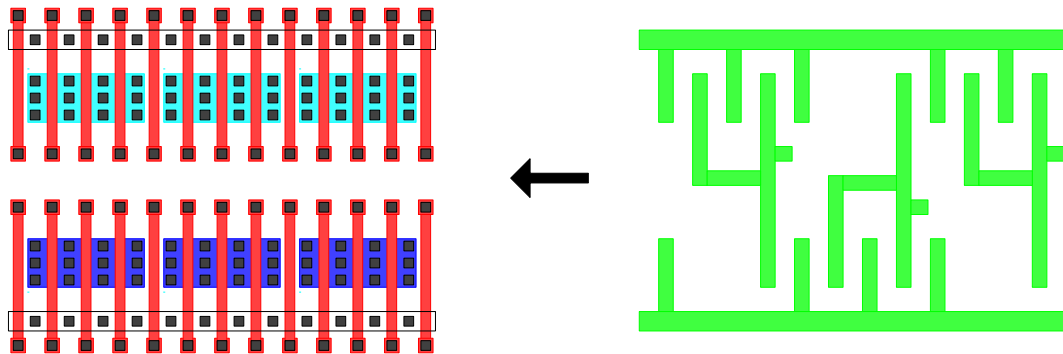


Abbildung 2.13: Anordnung von Gate-Array Grundzellen.

Funktionsfähige Zellen entstehen erst durch entsprechende Metall-Verbindungen zwischen den Transistoren einer Grundzelle. Zur Realisierung elementarer Gatter-Funktionen werden dafür vorbereitete Verdrahtungsmakros in Bibliotheken bereitgehalten. Neben den Verbindungen innerhalb der Zellen ist auch für die Leitungen zwischen den Zellen zu sorgen, die ebenfalls in Metall geführt werden. Dazu stehen zwischen den Grundzellen feste Verdrahtungsgebiete (channel) zur Verfügung. Deren Aufnahmekapazität für Leitungen ist vorgegeben, was dazu führt, daß man in manchen Fällen einen größeren und damit teureren Master einsetzen muß, nur weil ein Kanal die ihm zugewiesenen Leitungen nicht aufnehmen kann. Um diese Beschränkungen aufzuheben, werden in letzter Zeit zunehmend sogenannte „channelless“ Gate-Arrays oder Sea-of-Gate-Schaltungen angeboten, die mit einer weiteren Metallisierungsebene die Verdrahtung frei über die Zellen erlauben.

Die Verbindungen innerhalb und zwischen den Zellen bestimmen also gemeinsam die Metallisierungsmaske, mit der aus universell einsetzbaren Mastern ASICs werden. Diesen Vorgang bezeichnet man auch als Personalisierung. Heute verfügbare Master enthalten über 100000 Transistoren und sind außer in CMOS auch in ECL erhältlich.

2.5.3 Standardzellen

Die Zellen liegen hier nicht in vorgefertigter Form, sondern in vorentwickelter Form vor. Sie wurden sehr sorgfältig von Hand entworfen und sind nun mit Layout, Schaltplan (schematic) oder Ersatzschaltung sowie mit ihren elektrischen Eigenschaften beschrieben. Zellbibliotheken enthalten erprobte Zellen für viele elementare Funktionen wie logische Gatter, Flipflops, Multiplexer und viele mehr. Damit diese Zellen möglichst einfach zu größeren Blöcken kombinierbar sind, erfüllen sie bestimmte geometrische Anforderungen: Standardzellen sind rechteckig und werden auf der Metallebene in voller Breite von den Speisespannungsleitungen V_{dd} und V_{ss} durchquert; die Signaleingänge liegen auf der Polysilizium- oder der zweiten Metallebene.

Damit die Versorgungsleitungen nahtlos aneinander passen, müssen alle Standardzellen dieselbe Höhe aufweisen. Die Zellbreite hingegen kann mit der unterschiedlichen Komplexität der Zel-

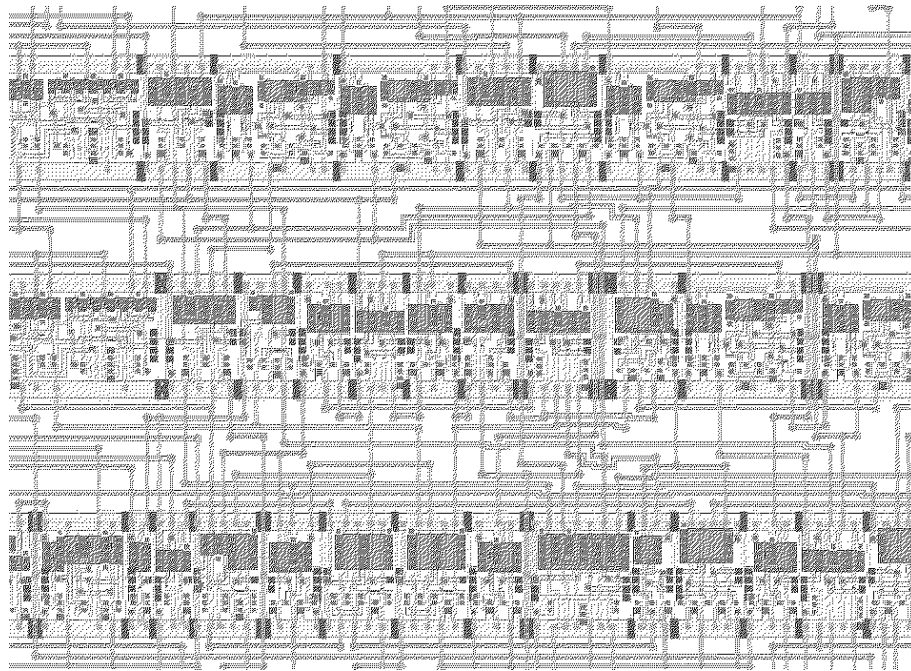


Abbildung 2.14: Ausschnitt aus einem Standardzell-Layout.

le variieren. Standardzellen werden in einfachen oder doppelten Reihen angeordnet. Horizontale und vertikale Zwischenräume dienen als Verdrahtungskanäle, deren variable Abmessungen an die Anzahl der durchgeführten Leitungen angepaßt werden können.

Der erste Schritt beim Schaltungsentwurf besteht darin, die einzelnen Schaltungsblöcke aus vorhandenen Bibliothekszellen zusammenzustellen. Anschließend müssen diese zweckmäßig platziert und verdrahtet werden. Der Aufbau - oder Ankauf - einer Zellbibliothek stellt eine große Anfangsinvestition dar, die sich in einer erheblich verbesserten Produktivität bei der Implementierung der einzelnen Schaltungen auszahlt. Der Entwurf umfaßt alle Fertigungsebenen.

2.5.4 Makrozellen

Makrozellen realisieren komplexere Funktionen. Typische Beispiele sind RAM (random access memory), ROM (read only memory), PLAs (programmable logic arrays), Zähler, Rechenwerke oder endliche Automaten. Makrozellen sind oftmals parametrisierbar, d.h. der Schaltungsentwickler kann für bestimmte Schaltungsparameter numerische Werte vorschreiben. Beispielsweise kann er die Stärke der Ausgangstreiber in Abhängigkeit der Lastkapazität spezifizieren und eine entsprechend ausgelegte Makrozelle erhalten. Weitergehende Parametrisierung betrifft die Funktionalität der Zelle, etwa ein n -Bit Addierwerk oder ein $n \times m$ Bit RAM. Für den Entwickler ist es dabei unwichtig, ob das CAD-System die Zellen fertig ausgelegt abrufen kann oder sie gemäß den Spezifikationen jeweils neu generiert. Wegen ihrer Parametrisierbarkeit weisen Makrozellen sehr uneinheitliche Abmessungen auf, sind aber in der Regel ebenfalls rechteckig. Häufig müssen in einer Schaltung neben den Makrozellen auch noch einige Standardzellblöcke untergebracht werden. Das Ergebnis des Entwurfs umfaßt auch hier alle Fertigungsebenen.

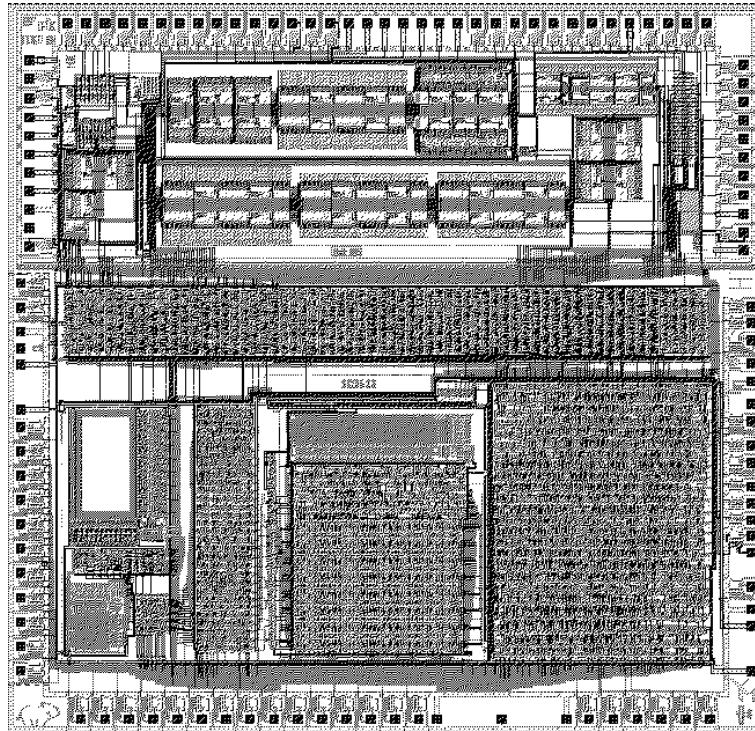


Abbildung 2.15: Anordnung von Makrozellen auf dem Chip.

2.5.5 Handlayout

Beim Handentwurf arbeitet der Entwicklungsingenieur auf dem Niveau der Entwurfsebenen. Einerseits bedeutet dies die völlige Freiheit, beliebige Strukturen zu entwerfen, soweit sie vom Fertigungsprozess her möglich sind. Dieser Aspekt ist vor allem in der Analog- und Mikrosystemtechnik wichtig. Andererseits ist es sehr aufwendig, selbst einfache Schaltungselemente wie Kontakte oder Transistoren durch eine Anzahl von Flächenstücken auf mehreren Ebenen zu konstruieren. Alle für den Entwickler wesentlichen Angaben zum Herstellungsprozess sind in den geometrischen Designregeln zusammengefaßt, die die für das sichere Funktionieren der Schaltung notwendigen Abstandsbeziehungen zwischen den einzelnen Strukturen auf dem Maskenniveau beschreiben. Das Einhalten dieser Anforderungen beim Entwickeln der Schaltung fällt beim traditionellen Handlayout voll in die Verantwortung des Entwicklers. Erst ein symbolisches Layout mit anschließender automatischer Kompaktierung befreit den Designer von dieser mühseligen Arbeit und produziert Schaltungen, die aufgrund ihrer Konstruktion korrekt sind (correctness by construction). Die Produktivität beim Schaltungslayout ist aufgrund des aufwendigen Entwurfsvorgangs gering. Hier muß also die höhere Flexibilität beim Entwurf durch längere Entwurfszeiten bezahlt werden. Inklusive Prüfung, Korrektur und Dokumentation rechnet man im Schnitt mit ca. 10 bis 30 entworfenen Bauelementen pro Tag. Im einzelnen hängt dieser Wert natürlich von der Regularität der Schaltung ab. Bei Speicherbausteinen ist die Regularität sehr hoch, bei „wilder“ Logik wesentlich niedriger. Das Ergebnis des Entwurfs umfaßt ebenfalls alle Entwurfsebenen. Abbildung 2.16 zeigt ein von Hand entworfenes Layout. Es handelt sich dabei um ein integriertes Drucksensorsystem

in Oberflächenmikromechanik. Neben den kreisförmigen Druckelementen, die zur Erhöhung des Signals 24-fach parallel geschaltet wurden, enthält das Layout eine analoge Ausleseschaltung.

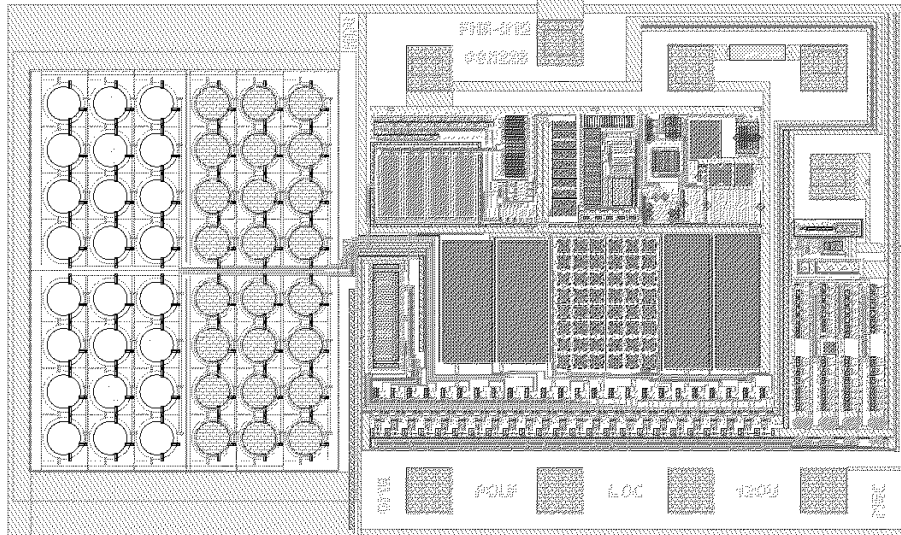


Abbildung 2.16: Schaltungsbeispiel Handlayout.

2.5.6 Auswahlkriterien

Diverse sich teilweise widersprechende Ziele werden während des Entwurfsvorgangs verfolgt. Für welche der oben besprochenen IC-Techniken man sich nun entscheidet, hängt im wesentlichen von den folgenden Kriterien ab.

- Chipfläche (primär kostenbestimmend bei sehr großen Stückzahlen)
- Entwicklungsaufwand (primär kostenbestimmend bei kleinen Stückzahlen)
- Leistungsverbrauch
- Arbeitsgeschwindigkeit
- Zeitbedarf von der letzten Änderung der Schaltung bis zur Chipauslieferung
- Einsatzbedingungen und Zuverlässigkeit
- Durchführbarkeit von Entwurfsänderungen
- Schutz gegen Nachahmung
- Montagekosten der Gesamtschaltung
- Anpassungsmöglichkeit an neue Herstellungsprozesse

- Einbeziehung von analogen Schaltungsteilen

Das Hauptkriterium für die Entscheidung zwischen mehreren konkurrierenden Techniken, welche die grundsätzlichen Anforderungen erfüllen, ist sicher die Wirtschaftlichkeit. Welches die kostengünstigste Lösung ist, hängt wiederum in hohem Maße von der Stückzahl ab, über die zu Beginn der Entwicklung in der Regel nur Schätzungen vorliegen.

2.6 Was ein Chip kostet

Dieses Skript widmet sich vor allem der technischen Dimension bei der Entwicklung einer integrierten Schaltung. Dem gegenüber steht die wirtschaftliche, die sich in Mark und Pfennig ausdrücken läßt. Nur in sehr wenigen Fällen kann man bei einer Entwicklung das technisch Machbare verwirklichen, ohne die Kosten zu berücksichtigen. In der Regel geht es schlicht darum, die Spezifikation mit möglichst geringen Kosten zu erfüllen. Ziel der Entwicklung ist es also häufig nicht die Maximierung der Performance einer Schaltung, sondern des Quotienten aus Performance und Kosten. Wenn man hier nicht mit der Stange im Nebel stochern will, muß man eine Vorstellung davon haben, wie sich die Kosten eines Chips ausrechnen lassen. Dies ist Gegenstand des vorliegenden Kapitels.

Dabei kann man zunächst die Kosten für Entwicklung und Fertigung unterscheiden. Die Entwicklungskosten orientieren sich an der eingesetzten Arbeitszeit der beteiligten Ingenieure, die Fertigungskosten am Aufwand für die Produktion der integrierten Schaltungen. Die in den folgenden Abschnitten gegebenen typischen Werte beruhen auf aktuellen Angaben (1997) eines großen Halbleiterherstellers.

2.6.1 Kosten der Entwicklung

Wie bereits oben angedeutet werden die Entwicklungskosten auf Basis der für die Entwicklung eingesetzten Personenjahre¹ kalkuliert. Je nach Gehaltsstruktur und Aufgabengebiet wird das Personenjahr eines Ingenieurs mit DM 200.000 bis DM 300.000 angesetzt. Dabei entfällt etwa die Hälfte dieses Betrages auf Lohn- und Lohnnebenkosten. Die andere wird als Gemeinkostenanteil bezeichnet, der mit den Aufwänden für Büroräume, Workstations, Software, Verwaltung etc. korrespondiert. Typischerweise arbeiten mehrere Ingenieure an einem Entwicklungsprojekt, so daß hier der zeitliche Aufwand der Beteiligten akkumuliert wird. Die im Rahmen der Entwicklung aufgelaufenen Kosten werden schließlich auf die verkauften Chips umgelegt, so daß die zu erwartenden Stückzahlen den vertretbaren Entwicklungsaufwand wesentlich bestimmen.

Neben dem Entwicklungsaufwand ist auch die Entwicklungszeit zu berücksichtigen. Insgesamt ließe sich vermuten, daß diese proportional zur Anzahl der am Projekt arbeitenden Ingenieure sinkt. Hier ist Vorsicht geboten, da die Effizienz des Einzelnen mit der Anzahl der Teammitglieder abnimmt. Dies folgt schon daraus, daß der Aufwand für die Abstimmung zwischen den Teammitgliedern quadratisch mit deren Anzahl steigt, da im schlimmsten Fall jeder mit jedem kommunizieren muß. Wenn man außerdem die im Normalfall recht kurze Projektlaufzeit von drei Monaten bis zu zwei Jahren berücksichtigt, so ist ein Projektvolumen von mehr als zwanzig Personenjahren

¹ Politisch korrekte Umschreibung des verpönten aber dennoch gebräuchlichen Mannjahrs.

in aller Regel nicht sinnvoll. Komplexere Aufgaben lassen sich nur dann beherrschen, wenn es eine eindeutige, natürliche Partitionierung des Problems gibt.

2.6.2 Kosten der Fertigung

Allgemeines

Integrierte Schaltungen werden auf Silizium-Scheiben gefertigt, die typischerweise einen Durchmesser von 4, 6 oder 8 Zoll aufweisen, und Wafer genannt werden. Die einzelne Schaltung auf dem Wafer wird Die² genannt, siehe Abbildung 2.17. Chargen von jeweils 20 oder 50 Wafern werden durch die Fertigungslinie geschleust.

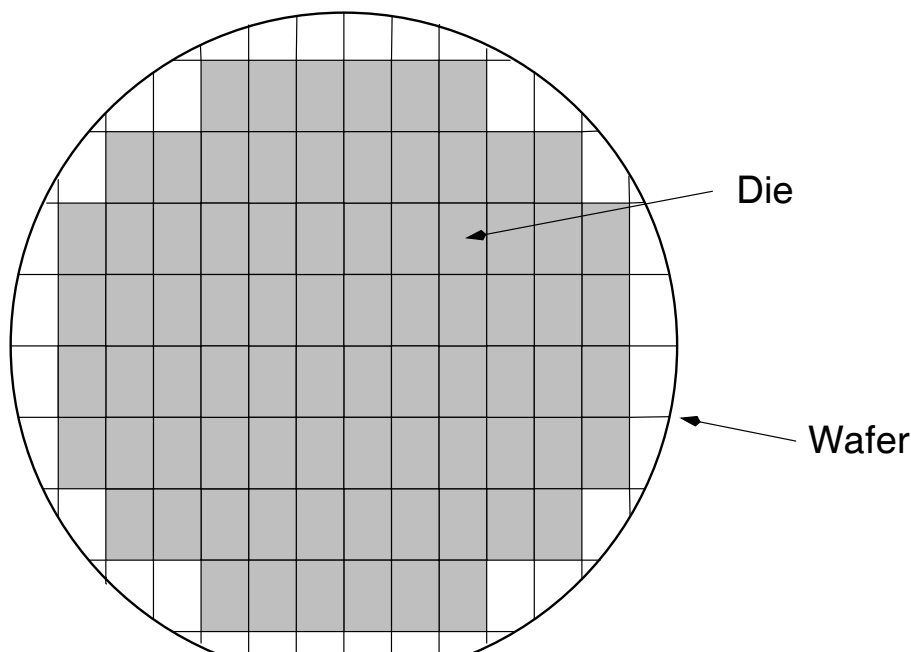


Abbildung 2.17: Wafer und Die.

Die Fertigungskosten eines gehäusten, funktionierenden Chips beinhaltet die Kosten zur Produktion, zum Test und zur Verpackung des Die in ein Gehäuse. Diese Kalkulation legt die Kosten zunächst auf die gefertigten Dies um. Verkaufen lassen sich allerdings nur die funktionierenden Dies. Damit muß man die Kosten pro Die durch die Ausbeute dividieren. Bei genauerer Betrachtung handelt es sich allerdings hier um verschiedene Ausbeuten, nämlich die der Fertigung und die der Verpackung in ein Gehäuse. Dies, die den Test in unverpacktem Zustand nicht bestanden haben, werden erst gar nicht verpackt. Im folgenden werden die Kosten für einen verpackten, funktionierenden Chip detailliert berechnet.

²Englisch für Gesenk, Gußform.

Kosten eines Die

Zur Berechnung der Kosten eines einzelnen Die, K_{Die} ist es notwendig die Kosten pro Wafer durch die Anzahl der funktionierenden Dies auf einem Wafer zu teilen. Für diese spielen hier zwei Faktoren eine Rolle: Die Fläche des Die und die Ausbeute, d.h. der Quotient aus der Anzahl der funktionierenden Dies und der Gesamtzahl der Dies auf dem Wafer.

$$K_{Die} = \frac{K_{Wafer}}{Z_{Die} \cdot A_{Die}} \quad (2.1)$$

wobei K_{Wafer} die Kosten für einen Wafer, Z_{Die} die Anzahl der Dies pro Wafer und A_{Die} die Ausbeute der Dies bezeichnen. Die Anzahl der Dies pro Wafer hängt sehr von Fläche und Umriß der Dies sowie von der Größe des Wafers ab. Die folgende Abschätzung liefert einen guten Anhalt:

$$Z_{Die} = \frac{\pi \cdot R_{Wafer}^2}{F_{Die}} - \frac{2 \cdot \pi \cdot R_{Wafer}}{\sqrt{L_{Die}^2 + B_{Die}^2}} - Z_{Testdie} \quad (2.2)$$

wobei R_{Wafer} den Radius des Wafers, L_{Die} die Länge des Die, B_{Die} die Breite des Die und $Z_{Testdie}$ die Anzahl der Testdies beschreibt. Testdies werden an einigen Stellen statt der regulären Dies eingefügt, um den Fertigungsprozeß und -resultat besser kontrollieren zu können. Die Abschätzung läuft darauf hinaus, daß man zunächst einfach die Flächen von Wafer und Die durcheinander dividiert. Am Rand kann ein Teil der Waferfläche nicht ausgenützt werden, da nur vollständig abgebildete Dies zu gebrauchen sind. Dies wird durch einen Korrekturterm berücksichtigt, der die verschenkte Fläche entlang des Umfangs des Wafers betrachtet. Man betrachtet hier den 'worst case' indem man den Umfang des Wafers in Abschnitte aufteilt, die jeweils der Diagonalen des Die entsprechen. Die Anzahl der Abschnitte entspricht der Anzahl der Dies, die man durch den Verschnitt am Rand abrechnen muß.

Wenn man von einem 6-Zoll Wafer und einem Die mit den Abmessungen von 10×12 mm ausgeht, so kann man bei 5 Testdies mit einer Anzahl von $152 - 31 - 5 = 117$ Dies mit einer Fläche von 120 mm^2 rechnen. Betrachtet man ein Die mit den Abmessungen 15×16 mm, so verdoppelt sich die Fläche auf 240 mm^2 . In diesem Fall verbleiben $76 - 22 - 5 = 49$ Dies.

Aus den obigen Berechnungen geht die der Geometrie entsprechende, maximale Anzahl von Dies hervor. Damit ist aber noch nicht gesagt, daß alle gefertigten Dies auch funktionieren. Wenn die Kosten pro funktionsfähigen Die berechnet werden sollen, ist die Ausbeute zu berücksichtigen. Die im folgenden gegebene Formel beschreibt diese trotz einiger Vereinfachungen in guter Näherung:

$$A_{Die} = A_{Wafer} \cdot \left(1 + \frac{D \cdot F_{Die}}{\alpha}\right)^{-\alpha} \quad (2.3)$$

Hier bezeichnen A_{Wafer} die Wafer-Ausbeute, die zur Berücksichtigung der komplett verdorbenen Wafer herangezogen wird. Für diese Wafer und die zugehörigen Dies fallen keine weitere Testkosten an. D ist die Anzahl der Defekte pro Einheitsfläche und steht damit für alle Fehlermechanismen, die den Ausfall einzelner Dies verursachen. Die Konstante α schließlich berücksichtigt den Einfluß des Fertigungsprozesses. Einfachere Prozesse wie Standard-CMOS weisen höhere Ausbeuten auf ($\alpha \approx 2.0$), bei komplexeren Prozessen, wie zum Beispiel BiCMOS, kann mehr schiefe gehen ($\alpha > 2.0$). Wenn man nun die Wafer-Ausbeute A_{Wafer} mit 0.95, die Anzahl der Defekte pro

Einheitsfläche (z.B. 100 mm^2) D mit 1.0 und einen Standard-CMOS Prozeß annimmt, so kann man bei einer Fläche von 120 mm^2 (240 mm^2) mit einer Ausbeute A_{Die} von 37% (19%) rechnen. Damit kann die Anzahl der funktionierenden Dies pro Wafer mit 43 (9) angesetzt werden.

Insgesamt berechnen sich die Kosten eines Die, nach Gleichung 2.1 wie folgt: Mit einem Preis von DM 1000,00 pro prozessierten Wafer ergibt sich bei einer Fläche von 120 mm^2 :

$$K_{Die} = \frac{DM1000,00}{117 \cdot 0.37} = DM23,10$$

Für eine Fläche von 240 mm^2 setzt man an:

$$K_{Die} = \frac{DM1000,00}{49 \cdot 0.19} = DM107,41$$

Die wesentliche Quintessenz dieses Abschnitts liegt darin, daß die Kosten eines Die mindestens mit der dritten Potenz der Fläche ansteigen, also:

$$K_{Die} = f(F_{Die}^3)$$

Testkosten

Die Ausfallquoten in der Fertigung haben noch eine weitere negative Konsequenz: Es muß in einem Test untersucht werden, welche Dies funktionieren und welche nicht. Dabei bietet es sich an, diesen Test bereits vor der Verpackung der Dies in Gehäuse vorzunehmen, da dann die nicht unbeträchtlichen Verpackungskosten für die ohnehin nicht funktionierenden Dies entfallen. Die Testkosten werden auf Basis der Dauer der Benutzung eines Testautomaten kalkuliert. Typische Werte liegen zwischen DM 80,00 und DM 150,00 für eine Stunde Testzeit. Diese Werte ergeben sich aus den Investitions- und Betriebskosten des Testautomaten sowie dessen Auslastung. Spezielle Testautomaten - etwa solche zum Test von gemischt digital/analoge Schaltungen - generieren einen entsprechend höheren Stundensatz.

Für unser Beispiel gehen wir von einem Stundensatz von DM 100,00 aus. Schwer testbare Schaltungen erfordern längere Testzeiten. Die Testbarkeit einer Schaltung ist damit ebenso wie die Schaltungsfläche ein Optimierungsziel, dessen Auswirkung sich finanziell quantifizieren läßt. Wir gehen von zwei durchschnittlichen Testzeiten aus: 10 Sekunden und 100 Sekunden. Damit kostet der Test eines Die DM 0,28 für 10 Sekunden und DM 2,80 für 100 Sekunden Testzeit. Problematisch hierbei ist allerdings, daß zwar alle Dies getestet werden; die Kosten dafür werden aber auf die funktionierenden Dies geschlagen. Damit lassen sich die Testkosten $K_{T/Die}$ durch die folgende Formel beschreiben:

$$K_{T/Die} = \frac{K_{Tester} \cdot T_{Test}}{A_{Die}} \quad (2.4)$$

wobei K_{Tester} die Kosten des Testautomaten pro Zeiteinheit und T_{Test} die durchschnittliche Testzeit beschreiben. Damit werden die funktionierenden Dies einer Fläche von 120 mm^2 bei einer Testzeit von 10 Sekunden (100 Sekunden) mit Testkosten von DM 0,76 (DM 7,57) belastet. Für eine Fläche von 240 mm^2 ergeben sich Testkosten von DM 1,47 (DM 14,74).

Kosten für die Verpackung

Eine mikroelektronische Schaltung kann erst dann eingesetzt werden, wenn sie in ein geeignetes Gehäuse einpaßt und mit dessen Anschlüssen verbunden (gebondet) wurde. Auch hier fallen Kosten an. Zunächst sind dies die Gehäusekosten, die sich je nach Zahl der Anschlüsse, abführbarer Verlustleistung und möglicher Die-Größe zwischen DM 2,00 und DM 200,00 bewegen. Hinzu kommen Kosten für Einbau bzw. Bonden von ca. DM 2,00 und schließlich die Kosten für das sogenannte Burn-In von z.B. DM 0,50. Beim Burn-In werden die Chips für kurze Zeit bei erhöhter Temperatur und/oder Versorgungsspannung betrieben. Damit wird eine Alterung der Schaltungen erreicht. Geht man nun davon aus, daß die Ausfallwahrscheinlichkeit einer Schaltung mit ihrem Alter kontinuierlich abnimmt, so fallen durch das Burn-In Chips beim End-Test der eingepackten Schaltungen durch, die sonst erst in der Anwendung versagt hätten. Das Ziel muß es jedoch immer sein, fehlerhafte Chips so früh wie möglich zu erkennen.

Die Verpackungskosten $K_{P/Die}$ berechnen sich nach folgender Formel:

$$K_{P/Die} = \frac{K_{Package} + K_{Bonden} + K_{Burn-In}}{A_{Packaging}} \quad (2.5)$$

wobei $K_{Package}$ für die Gehäusekosten, K_{Bonden} für die Bondkosten, $K_{Burn-In}$ für die Kosten des Burn-In und $A_{Packaging}$ für die Ausbeute der Verpackung inklusive Burn-In steht. Wir nehmen nun für den Chip mit 120 mm^2 (240 mm^2) ein Gehäuse für DM 10,00 (DM 60,00) an. Hinzu kommt die Ausbeute $A_{Packaging}$, die wir mit 90 % ansetzen. Für die Schaltung mit einer Fläche von 120 mm^2 (240 mm^2) ergeben sich Verpackungskosten von DM 13,89 (DM 69,44).

Summe der Fertigungskosten

In Tabelle 2.1 werden die Gesamtkosten eines Chip tabellarisch erfaßt. Es fällt auf, daß die Verdopplung der Chipfläche die Kosten pro Chip um den Faktor 4 bis 5 steigen läßt. Die Verzehnfachung der Testzeit läßt die Chipkosten immerhin noch um 8 % bis 20 % steigen. Mit anderen Annahmen kommt man bei der Berechnung der Kosten natürlich auch zu anderen Ergebnissen. Die Arithmetik der Kostenberechnung bleibt jedoch im Wesentlichen gleich.

Kostenart	Fläche 120 mm^2	Fläche 240 mm^2
Fertigungskosten pro Die	DM 23,10	DM 107,41
Testkosten pro Die, 10 Sek. (100 Sek.)	DM 0,76 (DM 7,57)	DM 1,47 (DM 14,74)
Verpackungskosten pro Die	DM 13,89	DM 69,44
Gesamtkosten	DM 37,75 (DM 44,56)	DM 178,32 (DM 191,59)

Tabelle 2.1: Akkumulierte Fertigungskosten für eine integrierte Schaltung.

Kapitel 3

Schaltungs-Simulation

3.1 Simulation auf Bauelementeniveau

Die Prototypenentwicklung integrierter Schaltungen ist mit hohen Kosten verbunden. Daher muß während des Entwurfs das Schaltungsverhalten fortlaufend auf Einhaltung der gewünschten Funktion überprüft werden. Eine integrierte Schaltung kann jedoch nicht – wie im Breadboard-Aufbau – sukzessive zusammengesetzt werden, da man in diesem Fall unter Umständen einige Millionen diskrete Bauelemente auf Platinen unterbringen müsste. Außerdem könnten technologiebedingte Einflüsse nicht ausreichend erfasst werden. Als Ausweg bietet sich die rechnergestützte Simulation der Schaltung an. Mit geeigneten Simulationswerkzeugen läßt sich das Schaltungsverhalten parallel zum Entwurf auf verschiedenen Abstraktionsniveaus überprüfen. So können zeit- und kostenintensive Wiederholungen des Fertigungsdurchlaufs mit geänderten Designdaten (redesign) durch eine geeignete Simulation vermieden werden.

Der folgende Abschnitt über die Schaltungs-Simulation, oder auch Analog-Simulation genannt, befasst sich mit den elektrischen Repräsentationen einer Schaltung. Diese werden durch einen Schaltplan (schematic) oder eine Netzliste (netlist) gegeben. Schaltpläne werden mit einem Schaltplan-Editor erstellt und modifiziert. Abbildung 3.1 zeigt den Schaltplan eines NAND-Gatters.

Eine Netzliste unterscheidet sich vom Schaltplan dadurch, daß sie nur noch die topologische Information über die Schaltung beinhaltet. Im folgenden wird eine einfache Transistornetzliste gezeigt, die die in Abbildung 3.1 gezeigte Schaltung repräsentiert. Der erste Eintrag in der Bauelementeile bezeichnet den Instanzname des Transistors, M1 ... M4; das M steht für MOS-Transistor. Die folgenden Einträge stehen für die Signale, die an Drain, Gate, Source und Bulk des betreffenden Transistors angeschlossen sind. Das P bzw. N bezeichnet schließlich den zugehörigen Parametersatz für den Transistor. Die Parameter W und L schließlich stehen für die Weite und Länge der betreffenden Transistoren.

```
.SUBCKT NAND2 IN1 IN2 OUT
M1 OUT IN1 VDD VDD P W=4u L=2u
M2 OUT IN2 VDD VDD P W=4u L=2u
M3 OUT IN1 ND1 VSS N W=2u L=2u
M4 ND1 IN2 VSS VSS N W=2u L=2u
.ENDS NAND2
```

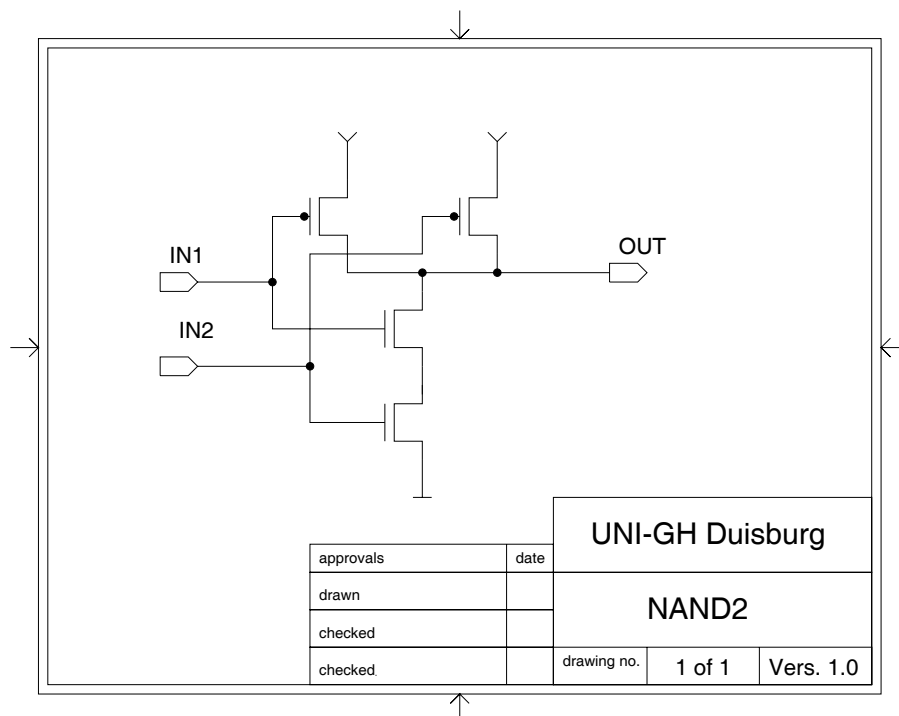


Abbildung 3.1: Schaltplan eines NAND-Gatters.

Die Bauelemente-Ebene ist die unterste elektrische Hierarchieebene, die für den Schaltungsentwurf von Bedeutung ist, und weist mithin die größte Detailtreue, aber auch die größten Datenmengen auf. Diese Genauigkeit muß jedoch durch erhebliche und oftmals untragbare Laufzeiten bezahlt werden. Damit ist die Simulation von Bauelementeschaltungen auf die Betrachtung von Teilschaltungen beschränkt, die nach einer Daumenregel weniger als Tausend Bauelemente enthalten sollen. Falls größere Schaltungen simuliert werden müssen, so ist man gezwungen, von der analogen Betrachtung der Bauelemente auf die digitale Betrachtung von Gattern überzugehen.

Im einzelnen gibt es vier Einsatzmöglichkeiten für die Schaltungsanalyse per Simulation:

- Spezifikationsprüfung

Hierbei soll die Einhaltung der Forderungen, die sich aus der Spezifikation der Schaltung ergeben, überprüft werden. Typischerweise betrachtet man anschließend die Ausgaben der letzten Simulation (golden log) als zur Dokumentation gehörig.

- Überprüfung einer Verfeinerung von einer übergeordneten Hierarchie auf die Bauelemente-Ebene

Die Transformation einer Schaltungsbeschreibung von einer Ebene auf eine niedrigere kann durch einen Vergleich von Simulationsergebnissen verifiziert werden. Dabei legt man in beiden Simulationen dieselben Eingangsbelegungen an, so daß sich theoretisch dieselben Antworten der Schaltungen einstellen müssten. Die Konsistenz zwischen den beiden Beschreibungen ist dann nachgewiesen, wenn die Simulationsergebnisse nur unwesentliche Abweichungen aufweisen. Auf diese Weise kann man beispielsweise einen Operationsverstärker

auf abstraktem Niveau betrachten und dann zu einer Transistorschaltung übergehen. Dieser Schritt läßt sich durch entsprechende Simulationen verifizieren.

- Überprüfung einer Transformation innerhalb der Bauelemente-Ebene

Falls eine Transformation einer Schaltungsbeschreibung auf derselben Ebene in eine andere Sicht durchgeführt wird, ist ebenfalls eine Verifikation anhand von Simulationsvergleichen möglich. Hier kann man beispielsweise an alternative Realisierungen eines Operationsverstärkers denken.

- Überprüfung einer Bauelemente-Schaltung unter Einbeziehung von Informationen aus dem Layout

Falls der Einfluß der physikalischen Realisierung einer Schaltung berücksichtigt werden soll, kann man eine Schaltungs-Extraktion durchführen, die als Ergebnis eine Netzliste mit gewünschten und parasitären Bauelementen liefert und diese simulieren.

Je nach Anwendungsfall bieten sich drei klassische Verfahren zur Simulation von Bauelementeschaltungen an, die im folgenden anhand einer kleinen Beispielschaltung, Abbildung 3.2, untersucht werden sollen.

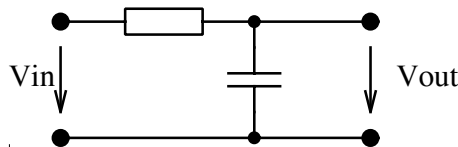


Abbildung 3.2: Beispielschaltung.

1. Gleichstromanalyse oder *DC-Analyse*

Hier werden die resultierenden Gleichspannungen und -ströme berechnet, die durch externe Gleichspannungs- bzw. Stromquellen angeregt werden. Das Verfahren dient zur Ermittlung der Gleichstromübertragungs-Charakteristik und des Arbeitspunktes. Dabei werden Kapazitäten zu Leitungsunterbrechungen und Induktivitäten zu Kurzschlüssen. Die Ausgangsspannung V_{out} ist für die Beispielschaltung gleich der Eingangs-Gleichspannung V_{in} .

2. Wechselstromanalyse oder *AC-Analyse*

Hier wird das Kleinsignalverhalten der Schaltung über alle Frequenzen ermittelt. Deshalb wird für diese Simulationsart auch der Begriff *Kleinsignalanalyse* verwendet. Die typische Anwendung ist die Berechnung des Frequenzganges der Schaltung. Die Beispielschaltung wird mit $V_{in} = \sin(\omega t)$ zum Tiefpass. Für die Beziehung zwischen V_{in} und V_{out} ergibt sich:

$$V_{out} = \frac{\frac{1}{j\omega C}}{R + \frac{1}{j\omega C}} V_{in} = \frac{1}{1 + j\omega RC} V_{in} \quad (3.1)$$

Die Verstärkung ergibt sich zu

$$\frac{V_{out}}{V_{in}} = \frac{1}{1 + j\omega RC} \quad \omega = 0 \Rightarrow \frac{V_{out}}{V_{in}} = 1, \quad \omega = \infty \Rightarrow \frac{V_{out}}{V_{in}} = 0 \quad (3.2)$$

Wenn man die Verstärkung gegen die Frequenz aufträgt, erhält man ein Bode-Diagramm, Abbildung 3.3.

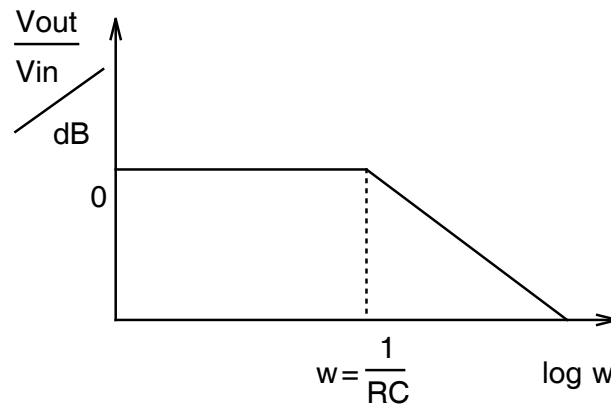


Abbildung 3.3: Bode-Diagramm für die Kleinsignalanalyse.

3. Dynamische Simulation oder *Transienten-Analyse*

Für externe zeitabhängige Erregungen wird das nichtlineare Schaltungsverhalten im Zeitbereich simuliert. Ein praktischer Anwendungsfall ist die Berechnung der Sprungantwort, Abbildung 3.4.

$$V_{out} = V_{in}(1 - e^{-\frac{t}{RC}}) \quad (3.3)$$

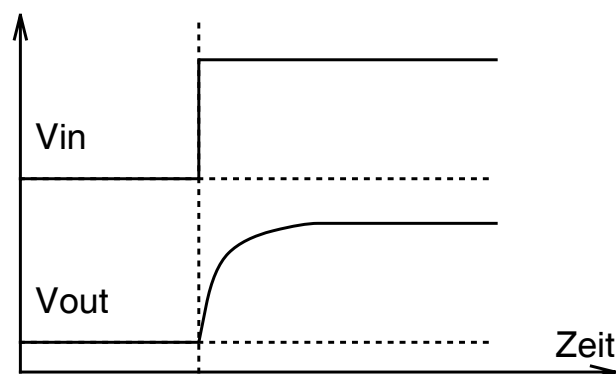


Abbildung 3.4: Sprungantwort bei der dynamischen Analyse.

Ein universeller Analog-Schaltungssimulator sollte alle aufgezählten Analysearten unterstützen. Kennzeichnendes Merkmal eines solchen Werkzeugs ist – unabhängig von der Analyseart – die Aufstellung und Lösung eines Gleichungssystems für die Schaltung. Folgende Grundlagen gehen in die Analog-Simulation ein:

- Knoten- und Maschengleichungen (Kirchhoff),
- Bauelementgleichungen (Ohm),
- Matrizenrechnung, Lösung von Gleichungssystemen (Gauß),
- Numerische Integration (Euler),
- Linearisierung von Bauelementekennlinien (Newton).

Im folgenden soll der Sachverhalt zunächst bezüglich der Gleichstromanalyse untersucht werden.

3.2 Kirchhoffsche Gleichungen

Die Anwendung der Knotengleichungen soll anhand der folgenden Abbildung 3.5 verdeutlicht werden. Folgende Knotengleichungen lassen sich für diese Schaltung aufstellen:

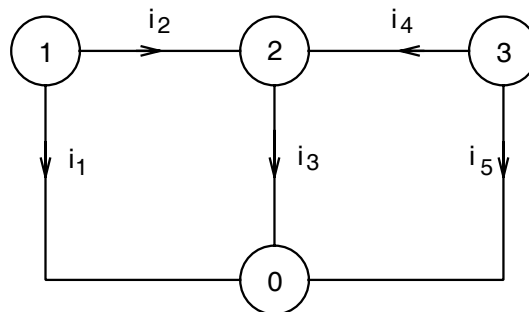


Abbildung 3.5: Anwendung der Kirchhoffschen Knotengleichungen.

$$\begin{aligned} i_1 + i_2 &= 0 \\ -i_2 + i_3 - i_4 &= 0 \\ i_4 + i_5 &= 0 \end{aligned}$$

Dabei wird der Knoten 0 als Referenzknoten nicht berücksichtigt. Dies ist zulässig, da sich für ein Netzwerk mit k Knoten ohnehin nur $k - 1$ linear unabhängige Knotengleichungen gewinnen lassen. Aus diesen Gleichungen ergibt sich die folgenden Matrizenform:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \end{bmatrix} = \underline{0} \quad (3.4)$$

$$\underline{A} \cdot \underline{i} = \underline{0} \quad (3.5)$$

Die Matrix $\underline{A}$ wird auch Knoten-Inzidenzmatrix genannt. Sie bezeichnet die Verbindungen zwischen den einzelnen Knoten - die Zweige - und die Richtung der in den Zweigen fließenden Ströme.

Die Kirchhoffschen Maschengleichungen werden ebenfalls benötigt, siehe Abbildung 3.6. Sie setzen die Knotenspannungen mit den Zweigspannungen in Relation. Die Bestimmung der Knotenspannungen ist ja gerade das Ziel der Analog-Simulation, während die Zweigspannungen zur Anwendung der Bauelementgleichungen benötigt werden. Die Knotenspannungen seien im folgenden alle auf den Knoten 0 bezogen.

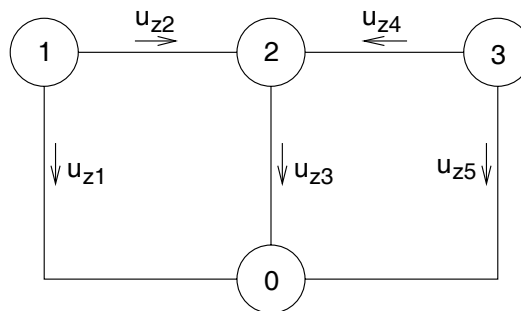


Abbildung 3.6: Anwendung der Kirchhoffschen Maschengleichungen.

$$\begin{aligned} u_{z1} &= u_{k1} \\ u_{z2} &= u_{k1} - u_{k2} \\ u_{z3} &= u_{k2} \\ u_{z4} &= -u_{k2} + u_{k3} \\ u_{z5} &= u_{k3} \end{aligned}$$

Auch diese Gleichungen lassen sich in Matrizenform darstellen:

$$\begin{bmatrix} u_{z1} \\ u_{z2} \\ u_{z3} \\ u_{z4} \\ u_{z5} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_{k1} \\ u_{k2} \\ u_{k3} \end{bmatrix} \quad (3.6)$$

Die Matrizenform lässt sich wiederum recht kompakt durch die folgende Formel darstellen. Dabei ist wesentlich, daß wiederum die Knoten-Inzidenzmatrix auftaucht; diesmal allerdings in transponierter Form.

$$\underline{u_z} = \underline{A}^T \cdot \underline{u_k} \quad (3.7)$$

3.3 Bauelementgleichungen

Neben der Topologie der Schaltung sollen nun auch die Bauelemente betrachtet werden. Für die in Abbildung 3.7 dargestellte Schaltung ergeben sich die folgenden Zweiggleichungen:

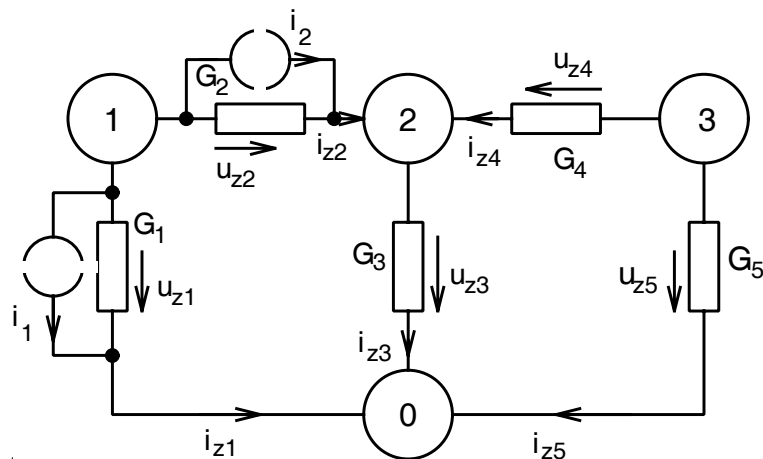


Abbildung 3.7: Anwendung der Bauelementgleichungen.

$$\begin{aligned} i_{z1} &= G_1 \cdot u_{z1} + i_1 \\ i_{z2} &= G_2 \cdot u_{z2} + i_2 \\ i_{z3} &= G_3 \cdot u_{z3} \\ i_{z4} &= G_4 \cdot u_{z4} \\ i_{z5} &= G_5 \cdot u_{z5} \end{aligned}$$

$$\begin{bmatrix} G_1 & 0 & 0 & 0 & 0 \\ 0 & G_2 & 0 & 0 & 0 \\ 0 & 0 & G_3 & 0 & 0 \\ 0 & 0 & 0 & G_4 & 0 \\ 0 & 0 & 0 & 0 & G_5 \end{bmatrix} \begin{bmatrix} u_{z1} \\ u_{z2} \\ u_{z3} \\ u_{z4} \\ u_{z5} \end{bmatrix} + \begin{bmatrix} i_1 \\ i_2 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} i_{z1} \\ i_{z2} \\ i_{z3} \\ i_{z4} \\ i_{z5} \end{bmatrix} \quad (3.8)$$

Die oben aufgestellten Matrizen werden in kompakter Schreibweise zu:

$$\underline{G} \cdot \underline{u}_z + \underline{i} = \underline{i}_z \quad (3.9)$$

Mit den Maschengleichungen $\underline{u}_z = \underline{A}^T \cdot \underline{u}_k$ ergibt sich für die obige Gleichung durch Einsetzen von $\underline{u}_z$:

$$\underline{G} \cdot \underline{A}^T \cdot \underline{u}_k + \underline{i} = \underline{i}_z \quad (3.10)$$

Anschließend wird die Matrix $\underline{A}$ von links hermultipliziert:

$$\underline{A} \underline{G} \underline{A}^T \cdot \underline{u}_k + \underline{A} \cdot \underline{i} = \underline{A} \cdot \underline{i}_z \quad (3.11)$$

Nach Kirchhoff ist die Summe der Ströme an einem Knoten gleich 0, also: $\underline{A} \cdot \underline{i}_z = \underline{0}$. Insgesamt gilt:

$$\underline{A} \underline{G} \underline{A}^T \cdot \underline{u}_k = -\underline{A} \cdot \underline{i} \quad (3.12)$$

Schließlich kann man noch den Term $\underline{A} \underline{G} \underline{A}^T$ durch die sogenannte Knotenadmittanzmatrix $\underline{Y}$ zusammenfassen, die nur von der Schaltungstopologie abhängt. Außerdem kann der Term $-\underline{A} \cdot \underline{i}$ als Anregungsvektor $\underline{i}_q$ aufgefasst werden, der für jeden Knoten die Summe aller eingepprägten Ströme angibt. Damit wird die obige Gleichung zu:

$$\underline{Y} \cdot \underline{u}_k = \underline{i}_q \quad (3.13)$$

Für das obige Beispiel ergibt sich das folgendes Gleichungssystem:

$$\begin{bmatrix} G_1 + G_2 & -G_2 & 0 \\ -G_2 & G_2 + G_3 + G_4 & -G_4 \\ 0 & -G_4 & G_4 + G_5 \end{bmatrix} \begin{bmatrix} u_{k1} \\ u_{k2} \\ u_{k3} \end{bmatrix} = \begin{bmatrix} -i_1 - i_2 \\ i_2 \\ 0 \end{bmatrix} \quad (3.14)$$

Die Knotenadmittanzmatrix kann auch in einem recht einfachen Verfahren direkt aus der Schaltung abgelesen werden. Zunächst trägt man in der Diagonalen für jeden Knoten gerade die Summe der an diesen Knoten angeschlossenen Leitwerte ein. Für den Knoten 2 wäre die Summe an der Position (2,2) einzusetzen. Für die übrigen Positionen (n,m) $n \neq m$ wird der negative Wert des Leitwertes zwischen den Knoten n und m eingetragen. Damit wird die Matrix symmetrisch zur Diagonalen, da an den Positionen (n,m) und (m,n) dieselben Eintragungen vorgenommen werden. Dies gilt unter der Voraussetzung, daß die Schaltung nur passive Bauelemente enthält.

Die Gleichungsformulierung in der Knotenspannungsanalyse ist immer eindeutig und recht einfach durchzuführen. Außerdem läßt sie sich auf alle drei Analysearten - Gleichstromanalyse, Wechselstromanalyse und dynamische Analyse - anwenden. Zur Lösung eines solchen Gleichungssystems stehen verschiedene Verfahren zur Verfügung. Das bekannteste ist die Gauß'sche Elimination. Hierbei wird die das Gleichungssystem beschreibende Matrix in Dreiecksform umgewandelt. Dies bedeutet, daß alle Elemente der Matrix, die oberhalb (unterhalb) der Hauptdiagonalen liegen, zu Null geworden sind. Die Umformung wird durch eine Linearkombination der Matrixzeilen bewerkstelligt. Aus der Dreiecksform ergibt sich, daß man den Wert der ersten Variablen direkt aus dem System ablesen kann, da die zugehörige Gleichung nur eine Variable beinhaltet. Diesen Wert setzt man in die zweite Gleichung ein und bekommt dadurch den Wert der zweiten Variablen. Sukzessive erhält man schließlich die komplette Lösung.

Die Anzahl der Multiplikationen für die Lösung des aufgestellten Gleichungssystems wächst in etwa mit dem Faktor n^3 (n = Rang der Knotenadmittanzmatrix). Damit erhält der zugehörige Algorithmus eine Laufzeitkomplexität von mindestens $O(n^3)$.

3.4 Modifizierte Knotenspannungsanalyse

Bei der Knotenspannungsanalyse wurden bislang die Gleichungen so formuliert, daß die Anregungen als Ströme vorgegeben und die Knotenspannungen damit als Unbekannte über die Bauelementgleichungen berechnet wurden. Für eine Spannungsquelle funktioniert dies nicht ohne weiteres, da hier ja gerade die Spannung vorgegeben und der Strom ausgerechnet werden soll. Nicht-ideale Spannungsquellen könnte man zwar noch in Stromquellen umrechnen; für ideale Spannungsquellen ist dies jedoch nicht möglich. Aus diesem Grund wurde die modifizierte Knotenspannungsanalyse entwickelt. Die Idee bei der modifizierten Knotenspannungsanalyse besteht darin, zusätzliche Gleichungen in das Gleichungssystem aufzunehmen, um die notwendigen Informationen berücksichtigen zu können.

Das Gleichungssystem $\underline{Y} \cdot \underline{u}_k = \underline{i}_q$ soll nun wie folgt erweitert werden:

$$\begin{matrix} i \\ j \end{matrix} \begin{bmatrix} & i & j & & & \\ Y & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & -1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cdot \\ u_k \\ \cdot \\ \cdot \\ \hline i_e \end{bmatrix} = \begin{bmatrix} \cdot \\ i_q \\ \cdot \\ \cdot \\ \hline u_e \end{bmatrix} \quad (3.15)$$

Für eine Spannungsquelle zwischen den Knoten i und j , siehe Abbildung 3.8, ergeben sich in den zusätzlichen Zeilen und Spalten folgende Werte: 1 in Zeile und Spalte i und -1 in Zeile und Spalte j . Zur Knotenadmittanzmatrix wird für jede neue Bestimmungsgleichung eine Zeile und eine Spalte hinzugefügt. Im Spannungs- und Stromvektor wurden außerdem die zugehörigen Ströme und Spannungen i_e und u_e hinzugefügt.

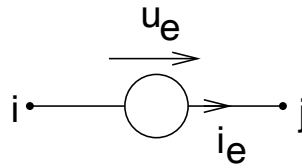


Abbildung 3.8: Spannungsquelle zwischen den Knoten i und j in der modifizierten Knotenspannungsanalyse.

3.5 Wechselstromanalyse

Der Übergang von der DC-Analyse zur AC-Analyse kann recht einfach durch Einsetzen geeigneter Impedanzen bewerkstelligt werden. Für Kapazitäten wird die Impedanz angegeben mit:

$$Z_C = \frac{1}{j\omega C} \quad Y_C = \frac{1}{Z_C} \quad (3.16)$$

Die Impedanz von Induktivitäten berechnet sich wie folgt.

$$Z_L = j\omega L \quad Y_L = \frac{1}{Z_L} \quad (3.17)$$

Unter Verwendung komplexer Arithmetik läßt sich die modifizierte Kontenspannungsanalyse ohne weiteres anwenden.

3.6 Dynamische Analyse

Bei der Transienten-Analyse werden Spannungen und Ströme als Funktionen der Zeit betrachtet.

$$i = C \frac{du}{dt} \quad u = L \frac{di}{dt} \quad (3.18)$$

Damit erhält man ein Differentialgleichungssystem der Form:

$$F(u, \dot{u}, t) = 0 \quad (3.19)$$

Für gegebene Anfangsbedingungen läßt sich dann eine Lösung berechnen. Die erste Ableitung der Spannung nach der Zeit läßt sich näherungsweise durch einen Differenzenquotienten darstellen.

$$\dot{u}(t) = \frac{du}{dt} \approx \frac{\Delta u}{\Delta t} = \frac{u(t) - u(t-h)}{t - (t-h)} = \frac{u(t) - u(t-h)}{h} \quad (3.20)$$

wobei h den Zeitschritt des Differenzenquotienten bezeichnet. Durch Umstellung der Gleichung 3.20 ergibt sich:

$$u(t) = u(t-h) + h\dot{u}(t) \quad (3.21)$$

so daß sich $u(t)$ aus dem alten Wert $u(t-h)$ und der sich aus der zeitlichen Ableitung $\dot{u}(t)$ ergebenden Veränderung von u ergibt. Falls man nun die Beziehung zwischen Spannung und Strom an einer Kapazität betrachtet, ergibt sich folgende Gleichung:

$$i = C\dot{u} = C \frac{du}{dt} = C \left[\frac{u(t) - u(t-h)}{h} \right] \quad (3.22)$$

Dabei stellt sich die Frage zu welchem Zeitpunkt der Strom in der obigen Gleichung betrachtet wird. Insbesondere kommen die Zeitpunkte t oder $t-h$ in Frage.

Die numerische Integration, bei der der Wert $i \triangleq i(t-h)$ verwendet wird, ist unter dem Begriff „Forward Euler“ bekannt. Die Gleichung für die Spannung wird dann nach Umformung von Gleichung 3.22 zu:

$$u(t) = u(t-h) + \frac{h}{C} i(t-h) \quad (3.23)$$

Falls man $i \triangleq i(t)$ setzt heißt die Methode „Backward Euler“. Aus der Gleichung 3.22 ergibt sich:

$$u(t) = u(t-h) + \frac{h}{C} i(t) \quad (3.24)$$

Schließlich kann man auch einen Mittelweg beschreiten. Bei der „Trapezintegration“ setzt man:

$$i \triangleq \frac{i(t) + i(t-h)}{2} \quad (3.25)$$

und erhält mit Gleichung 3.22:

$$u(t) = u(t-h) + \frac{h}{2C} [i(t) + i(t-h)] \quad (3.26)$$

Ein wichtiges Beurteilungskriterium der Methoden zur numerischen Integration ist die Stabilität. Das Verfahren ist genau dann stabil, wenn die Summe aller Amplituden von u endlich ist. Hier zeigen die Trapezintegration und das Backward-Euler Verfahren Vorteile gegenüber der Forward-Euler Methode, die aus diesem Grund nur recht selten eingesetzt wird.

Im folgenden soll die Umsetzung des Backward-Euler Ansatzes etwas genauer betrachtet werden. Zu diesem Zweck lösen wir die Gleichung 3.24 nach $i(t)$ auf:

$$i(t) = \frac{C}{h} u(t) - \frac{C}{h} u(t-h) \quad (3.27)$$

Wenn man nun C/h als Leitwert G eines Widerstandes betrachtet und $(C/h)u(t-h)$ als Strom einer parallelgeschalteten Stromquelle, so kann man für die Betrachtung des Stroms an einer Kapazität ein Ersatzschaltbild aus einem Widerstand und einer Stromquelle angeben:

$$i = Gu - I \quad (3.28)$$

Für den Strom ergeben sich also zwei Anteile von denen einer von der Spannung abhängt und der andere nicht. Mit den beiden letzten Gleichungen kann man nun ein Ersatzschaltbild angeben, Abbildung 3.9.

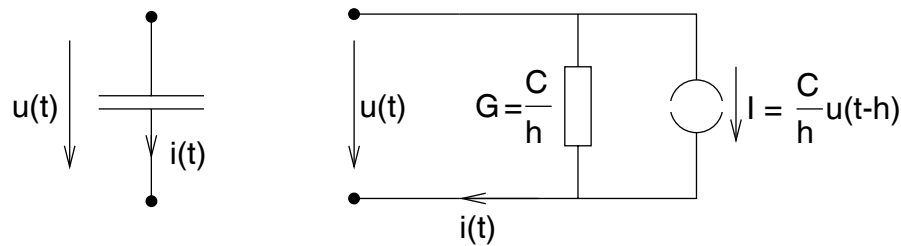


Abbildung 3.9: Ersatzschaltbild für einen Kondensator.

3.7 Linearisierung von Bauelementekennlinien

In aller Regel weisen aktive Bauelemente wie Dioden und Transistoren einen nicht-linearen Zusammenhang zwischen ihren Spannungen und Strömen auf. Dies führt zu einem nicht-linearen Gleichungssystem, das nicht ohne weiteres lösbar ist. Durch die Linearisierung der Kennlinien im Arbeitspunkt transformiert man das nicht-lineare Gleichungssystem in ein lineares, das mit Standardverfahren, wie der Gauß'schen Elimination gelöst werden kann. Zu diesem Zweck geht man von einem gedachten Arbeitspunkt aus und legt dort eine Tangente an die Kennlinie an. Danach löst man das sich ergebende lineare Gleichungssystem, woraus sich ein neuer Arbeitspunkt ergibt. In dieser Weise tastet man sich iterativ an den realen Arbeitspunkt heran. Diese Tangente gilt dabei jeweils als lineare Kennlinie, siehe Abbildung 3.10. Dieses Verfahren wurde erstmals von Newton eingesetzt und diente dazu, den Nulldurchgang von komplizierteren Funktionen zu bestimmen. Wie sich gleich zeigen wird, entspricht dies einer Ersatzschaltung für das betreffende Bauelement, die durch eine Parallelschaltung von einem Widerstand und einer Stromquelle beschrieben wird. In der Nähe des Arbeitspunktes verhalten sich das nicht-lineare Bauelement und die Ersatzschaltung annähernd gleich. Dieser Ansatz ist unter dem Namen „Newton-Raphson-Verfahren“ in der Literatur bekannt.

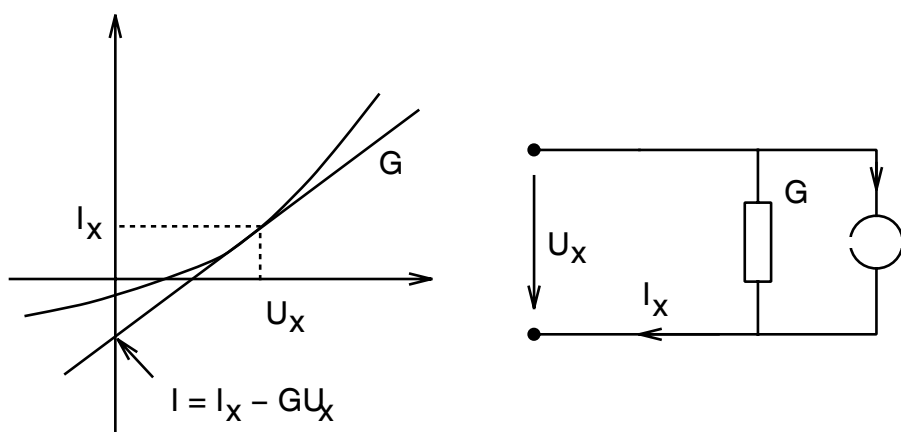


Abbildung 3.10: Linearisierung einer nicht-linearen Bauelementekennlinie.

Newton gibt folgende Formel für den neuen x -Wert, der näher am Nulldurchgang der Funktion liegen soll, an:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (3.29)$$

Die Aufgabe bei der Linearisierung von Bauelementekennlinien ist allerdings etwas anders gelagert. Wenn man zwei temporäre Arbeitspunkte nacheinander betrachtet, kann man die Steigung der Sekante durch die beiden Punkte mit folgendem Differenzenquotienten beschreiben, wobei die Steigung der Sekante ein gutes Maß für die Steigung der Tangente an Punkt x_n , d.h. für $f'(x_n)$ ist.

$$f'(x_n) \approx \frac{f(x_{n+1}) - f(x_n)}{x_{n+1} - x_n} \quad (3.30)$$

Wir lösen nun nach $f(x_{n+1})$ auf und betrachten die resultierende Gleichung als Grundlage für Linearisierung einer Bauelementekennlinie:

$$f(x_{n+1}) = f(x_n) + f'(x_n)(x_{n+1} - x_n) \quad (3.31)$$

Diese Gleichung entspricht der Entwicklung einer Taylor-Reihe um den temporären Arbeitspunkt, wobei man nach dem linearen Glied abbricht. Betrachten wir nun ein beliebiges, nicht-lineares Bauelement, das wie üblich durch den Zusammenhang zwischen Spannung und Strom charakterisiert wird:

$$i = f(u) \quad (3.32)$$

Die Gleichung 3.31 wird damit zu:

$$i_{n+1} = i_n + f'(u_n)(u_{n+1} - u_n) \quad (3.33)$$

Die Steigung $f'(u)$ definiert sich dabei wie folgt:

$$f'(u_n) = \left. \frac{di}{du} \right|_{u_n} = g_n \quad (3.34)$$

wobei g_n einen Leitwert repräsentiert. Dieser läßt sich wiederum in Gleichung 3.33 einsetzen:

$$i_{n+1} = g_n u_{n+1} + i_n - g_n u_n \quad (3.35)$$

Nachdem g_n und $g_n u_n$ bei der Berechnung von i_{n+1} festliegen, kann man sie auch durch die Konstanten G und I ersetzen, die wiederum einem festen Leitwert und einem festen Strom entsprechen.

$$i_{n+1} = G u_{n+1} + I \quad (3.36)$$

Mit diesen Betrachtungen ergibt sich ein Ersatzschaltbild für beliebige Bauelemente, das aus einem Leitwert und einer Stromquelle besteht, siehe Abbildung 3.10. Für den aktuellen Arbeitspunkt U_x, I_x wird eine Tangente an die Kennlinie angelegt, deren Steigung dem Wert von G entspricht, während ihr Schnittpunkt mit der Ordinate dem $I = I_x - G U_x$ entspricht. Für die nächste Iteration

werden G und I dann neu berechnet. Nach und nach konvergiert die Tangente des Ersatzschaltbildes an die Bauelementekennlinie hin zum realen Arbeitspunkt. Durch Verwendung dieser Ersatzelemente für nicht-lineare Zweige läßt sich die Analyse nicht-linearer Schaltungen auf die Analyse linearer (Ersatz-)Schaltungen zurückführen.

Die Berechnung von G und I sollen abschließend am Beispiel einer Diode illustriert werden. Die Bauelementekennlinie einer Diode läßt sich wie folgt beschreiben:

$$I_D = I_S \left(e^{\frac{U_D}{V_T}} - 1 \right) \quad (3.37)$$

Das I_D entspricht dabei dem I_x in Abbildung 3.10, das U_D dem U_x . G und I berechnen sich wie folgt:

$$G = \frac{dI_D}{dU_D} = \frac{I_S}{V_T} e^{\frac{U_D}{V_T}} \quad (3.38)$$

$$I = I_D - G \cdot U_D = I_S \left[\left(1 - \frac{U_D}{V_T} \right) e^{\frac{U_D}{V_T}} - 1 \right] \quad (3.39)$$

3.8 Ablauf einer Analog-Simulation

Das in Abbildung 3.11 gezeigte Flußdiagramm stellt den Ablauf einer Analog-Simulation dar. Dabei wurden aus Gründen der Übersichtlichkeit einige Teilbereiche weggelassen, die zwar nicht direkt mit der eigentlichen Simulation zusammenhängen, aber doch für einen reibungslosen Ablauf unverzichtbar sind. So enthält ein Simulator in aller Regel einige Routinen, die dazu dienen, eine Schaltung zu lesen und zu interpretieren. Außerdem sollten bei Syntaxfehlern in der Schaltungsbeschreibung möglichst aussagekräftige Fehlermeldungen gegeben werden. Nach Ablauf einer Simulation werden die Ergebnisse aufbereitet und gegebenenfalls auch graphisch dargestellt. Zu diesem Zweck sollte ein Werkzeug zur Verfügung stehen, das in der Lage ist, die resultierenden Signale genau darzustellen, den zu betrachtenden Zeitbereich zu wählen oder falls nötig, bestimmte Teilbereiche der Darstellung zu vergrößern. Weiterhin ist im Rahmen eines Frameworks eine Anbindung des Simulators an das Schaltplaneingabeprogramm wünschenswert, damit die darzustellenden Signale direkt in der Schaltung selektiert werden können.

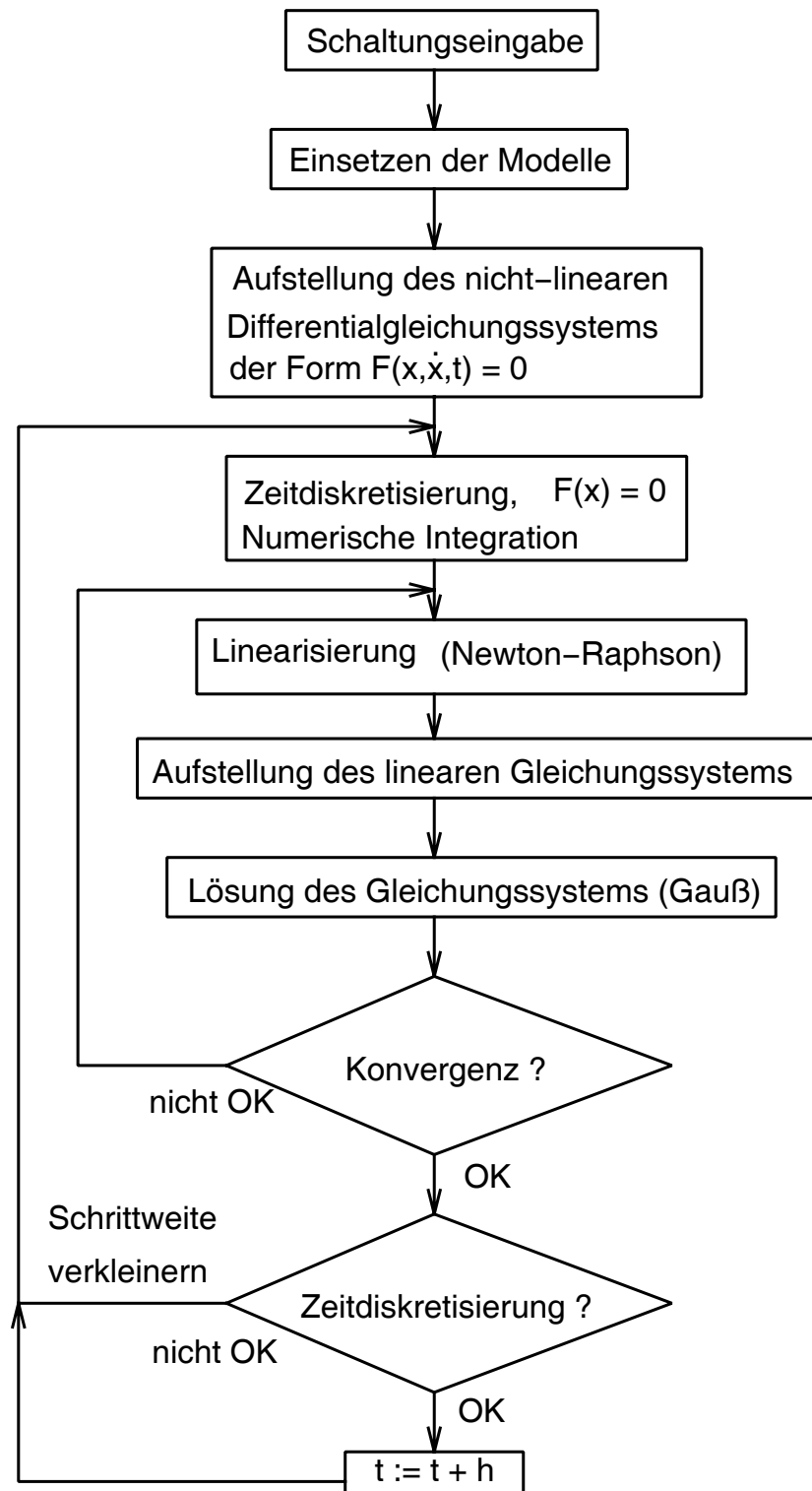


Abbildung 3.11: Ablauf einer Analog-Simulation.

Kapitel 4

Logik-Simulation

Die Simulation auf Bauelementeebene kann vielfach wegen des mit ihr verbundenen Aufwands nicht für größere Schaltungen durchgeführt werden. Wenn man diese Schaltungen trotzdem per Simulation analysieren möchte, muß man Abstriche bei der Genauigkeit machen. Für Digital-schaltungen ist es in aller Regel möglich, zwei oder drei Logikwerte, z.B. $\{0,1,X\}$, statt der bisher verwendeten kontinuierlichen Potentiale einzusetzen. Weiterhin ist es in aller Regel nicht notwendig, jeden einzelnen Transistor zu berechnen. Statt dessen werden Gatter betrachtet, die jeweils den Logikwert am Ausgang aus den Logikwerten an den Eingängen berechnen. Typischerweise enthalten CMOS Gatter zwei, vier oder mehr Transistoren, die nun nicht mehr explizit betrachtet werden müssen. Damit ergibt sich eine erhebliche Datenreduktion durch die Betrachtung der Schaltungen auf Logikebene. Außerdem wird auch das Zeitraster an die in der Schaltung vorhandene Aktivität angepaßt. Solange sich kein Logikwert in der Schaltung ändert, muß auch keine neue Berechnung durchgeführt werden. Damit geht man bei der Logiksimulation zu einem ereignisgesteuerten Ablauf über. Dies bedeutet, daß grundsätzlich nur die Gatter ausgewertet werden, deren Eingänge sich geändert haben. Dies ist sinnvoll, da sich in Gatterschaltungen im Vergleich zu Transistor-schaltungen nur vergleichsweise wenig Rückkopplungen befinden, an denen die Resultate durch mehrfache Iteration gewonnen werden müssen. Außerdem bleibt die Anzahl der Iterationen durch die begrenzte Wertemenge gering. Diese Methodik bildet eine völlige Abkehr von der im vorigen Kapitel betrachteten Simulation durch die Lösung von Gleichungssystemen.

Wie auch für die Abarbeitung von Programmen bieten sich für die Logiksimulation zwei grundlegende Methoden an: die compilierte Simulation und die interpretierte Simulation. Für die compilierte Variante wird zunächst die Schaltung analysiert und durch ein äquivalentes Programm ersetzt, das das Verhalten der Schaltung genau nachbilden soll. Dieses Programm wird anschließend übersetzt und gebunden, so daß es im Rahmen einer Simulation ausgeführt werden kann. Bei der interpretierten Logiksimulation liegt der Fall anders. Hier existiert ein Simulationsprogramm, das zunächst die zu simulierende Schaltung in ihren Speicher lädt. Jedem Netz ist eine Variable zugeordnet, die den Logikwert zum aktuellen Zeitpunkt aufnimmt. Für jeden geänderten Logikwert in der Schaltung oder an den Eingängen werden die an die entsprechenden Leitungen angeschlossenen Gatter im nächsten Zeitschritt ausgewertet. Gegebenenfalls werden die Logikwerte der Netze an den Ausgängen der Gatter auf einen anderen Wert eingestellt. Dieser Zyklus wiederholt sich solange, bis sich in der Schaltung keine Änderungen mehr ergeben. Diese beiden grundlegenden Methoden der Logiksimulation sollen anhand eines Beispiels näher erläutert werden.

4.1 Compilierte Simulation

Aus der Schaltung 4.1 lassen sich recht einfach einige Gleichungen ableiten, die das logische Verhalten der Schaltung beschreiben. Diese Gleichungen könnte man auch als Programm betrachten, das im Rahmen einer compilierten Simulation aufgestellt wird.

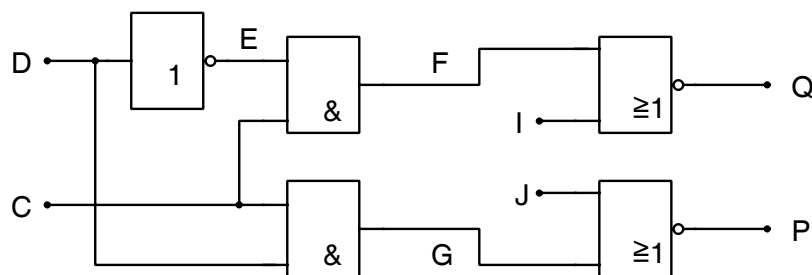


Abbildung 4.1: Beispielschaltung für die Logiksimulation.

Ein solches Programm für die Schaltung von Abbildung 4.1 könnte die folgende Form haben:

INPUTS: D, C, I, J;

E = $\neg D$;

F = E $\cdot$ C;

G = D $\cdot$ C;

P = $\neg (G + J)$;

Q = $\neg (F + I)$;

Grundsätzlich gilt, daß das Programm in jedem Simulationsschritt komplett abzuarbeiten ist, obwohl sich vielleicht nur ein kleiner Teil der Logikwerte ändert. Die Tabelle 4.1 stellt die sich ergebenden Logikwerte dar.

C	D	I	J	E	F	G	P	Q
1	1	0	1	0	0	1	0	1
1	0	0	1	1	1	0	0	0

Tabelle 4.1: Compilierte Simulation für das Schaltungsbeispiel ohne Rückkopplung.

Die oben dargestellte Schaltung soll nun etwas modifiziert werden, um die Probleme bei der Logiksimulation aufzuzeigen (Abbildung 4.2). Insbesondere werden zwei Rückkopplungen eingeführt, so daß *ein* Durchlauf durch die Schaltung nicht mehr ausreichend ist, um die Logikwerte komplett zu berechnen. Insbesondere sind auch Schaltungen vorstellbar, die ohnehin an einigen Knoten keine stabilen Logikwerte annehmen. Ein Ring von drei Invertern gehört zu dieser Art von Schaltungen. Die Schwierigkeit für die Logiksimulation besteht darin, solche Situationen zu erkennen und adäquate Maßnahmen einzuleiten, um die Simulation auch an diesen Stellen möglichst aussagekräftig zu machen.

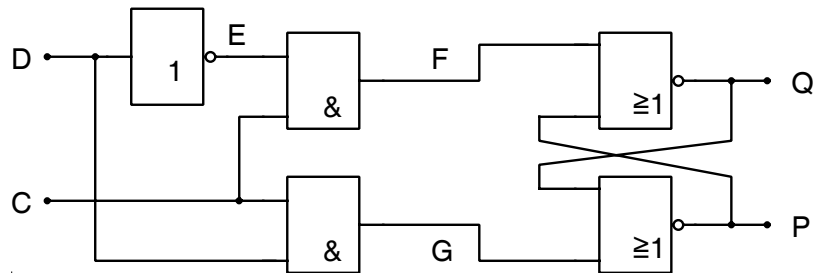


Abbildung 4.2: Modifizierte Beispielschaltung für die Logiksimulation.

Das zugehörige Programm sieht dann wie folgt aus:

INPUTS: D, C;

E = $\neg D$;

F = $E \cdot C$;

G = $D \cdot C$;

P = $\neg (G + Q)$;

Q = $\neg (F + P)$;

Damit ist die Simulation abhängig von den Anfangswerten von P und Q, da diese in die Bestimmungsgleichungen eingehen. Wesentlich bei diesem Programm ist, daß zum Lesen und Schreiben der Werte zwei unterschiedliche Sätze von Variablen verwendet werden, so daß die Reihenfolge der Bearbeitung innerhalb einer Iteration für das Ergebnis keine Rolle spielt.

C	D	E	F	G	P	Q	
1	0	1	1	0	1	0	Anfangswerte für P und Q lassen sich berechnen
1	1	1	1	0	1	0	D geht auf 1
1	1	0	1	1	1	0	1. Iteration
1	1	0	0	1	0	0	2. Iteration
1	1	0	0	1	0	1	Endzustand
1	0	0	0	1	0	1	D geht auf 0
1	0	1	0	0	0	1	1. Iteration
1	0	1	1	0	0	1	2. Iteration
1	0	1	1	0	0	0	3. Iteration
1	0	1	1	0	1	0	Endzustand

Tabelle 4.2: Compilierte Simulation für die Beispielschaltung mit Rückkopplung.

4.2 Ereignisgesteuerte Logiksimulation

Entscheidend für den effizienten Ablauf ist, daß während einer Logiksimulation nur ein Bruchteil der Schaltungen aktiv ist und mithin Änderungen in den Logikwerten aufweist. Für Gatter, deren Eingänge sich nicht ändern, sind auch keine weiteren Berechnungen durchzuführen, da der alte Wert übernommen werden kann. Diese Betrachtungsweise nennt man ereignisgesteuert (event-driven). Der Einsparungen durch die interpretierte oder ereignis-gesteuerte Logiksimulation werden bezüglich des Beispiels aus Abbildung 4.2 durch die folgenden Beobachtung motiviert. Falls beispielsweise C gleich 0 ist, können sich nur D und E ändern; alle übrigen Netze liegen auf festem Potential, da F und G in jedem Fall auf logisch 0 liegen. Auch hier soll zur Verdeutlichung eine Logiksimulation durchgeführt werden (Tabelle 4.3).

C	D	E	F	G	P	Q	
0	0	1	0	0	0	1	Anfangswerte
1			1	0			„externes“ Ereignis Berechne alle an C angeschlossene Gatter neu Berechne alle an F angeschlossene Gatter neu Berechne alle an Q angeschlossene Gatter neu Berechne alle an P angeschlossene Gatter neu
1	0	1	1	0	1	0	Stabiler Zustand
0	1		0	0			„externes“ Ereignis Berechne alle an C und D angeschlossene Gatter neu Berechne alle an E und F angeschlossene Gatter neu
0	1	0	0	0	1	0	Endzustand

Tabelle 4.3: Ereignisgesteuerte Simulation für die Beispielschaltung mit Rückkopplung.

Der Algorithmus der ereignis-gesteuerten Simulation läuft wie folgt ab.

Algorithmus 4.1 (Ereignis-gesteuerte Simulation)

```

i = 0;
Ai = Inputs;
do
    Bi = Menge aller an Ai angeschlossenen Gatter;
    Berechne die neuen Ausgänge von Bi;
    Ai+1 = geänderte Inputs + geänderte Ausgänge von Bi;
    i = i + 1;
while Ai+1 ≠ 0;

```

4.3 Behandlung von Races

Races in einer Schaltung führen dazu, daß der sich einstellende Zustand von den Gatter- und Leitungsverzögerungen abhängt. Im Rahmen der einfachen Logiksimulation führen Races oftmals dazu, daß die Reihenfolge der Betrachtung der Gatter innerhalb eines Zeitschritts die Simulationsergebnisse bestimmt. Dies ist ein ausgesprochen unerwünschter Zustand, da das Problem nicht deutlich sichtbar zu Tage tritt. Dies soll an einem RS-Latch, Abbildung 4.3, illustriert werden.

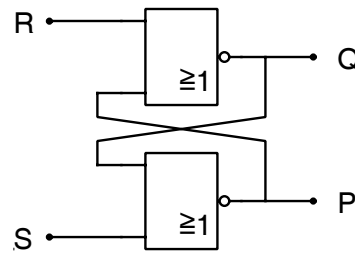


Abbildung 4.3: RS-Latch.

Für das RS-Latch ergibt sich die in Tabelle 4.4 dargestellte Wahrheitstabelle. Man erkennt, daß es sich hier um ein speicherndes Element handelt, da bei der Eingabebelegung $R=0$ und $S=0$ die Werte zum Zeitpunkt n für P und Q den zugehörigen Werten zum Zeitpunkt $n-1$ entsprechen.

R	S	P(n)	Q(n)
0	1	0	1
1	0	1	0
0	0	P(n-1)	Q(n-1)
1	1	0	0

Tabelle 4.4: Wahrheitstabelle für das RS-Latch.

Falls im Rahmen einer Simulation die Verzögerungszeiten vernachlässigt - d.h. zu Null gesetzt - werden, ergibt sich ein Problem der Kausalität, da dann Ursache und Wirkung zur gleichen Zeit eintreten.

Selbst wenn für die Gatter Verzögerungszeiten größer Null angesetzt werden, ergeben sich Schwierigkeiten. Dies wird an der in Tabelle 4.5 gezeigten Simulation deutlich. Man erkennt, daß das Ergebnis der Simulation von der Reihenfolge der Betrachtung der einzelnen Gatter abhängig ist. Dem Benutzer des Simulators sollte zumindest gemeldet werden, daß das Ergebnis der Simulation am Ausgang dieser Gatter fragwürdig ist. Um dies zu erreichen, wird das sogenannte Halbschrittverfahren angewendet.

	R	S	P	Q	
1	0	0	0	1	Anfangszustand
2	1	0			externes Ereignis
3				0	
4			1		
5	1	0	1	0	stabiler Zustand
6		1			externes Ereignis
7			0		stabiler Zustand
8	1	1	0	0	
9a	0	0			externes Ereignis, R wird zuerst betrachtet
10a				1	
11a	0	0	0	1	
9b	0	0			externes Ereignis, S wird zuerst betrachtet
10b			1		
11b	0	0	1	0	

Tabelle 4.5: Simulation eines RS-Latch.

4.4 Halbschrittverfahren

Beim Halbschrittverfahren wird zusätzlich zu den bekannten Logikwerten ein Wert U ($\hat{=}$ unbekannt) eingeführt. Beim Übergang von logisch 0 zu logisch 1 oder umgekehrt wird jeweils zunächst der Zwischenwert U für das Netz angesetzt. Damit teilt sich ein Simulationsschritt in zwei Halbschritte auf. Im ersten Simulationshalbschritt wird für die sich ändernden Logikwerte ein U eingesetzt und simuliert. Im zweiten Halbschritt betrachtet man den endgültigen Logikwert und führt den zweiten Simulationshalbschritt aus. Die Logikmodelle der Gatter ändern sich entsprechend. Beispielhaft zeigt die Tabelle 4.6 das neue Logikmodell eines NOR-Gatters mit zwei Eingängen.

	0	1	U
0	1	0	U
1	0	0	0
U	U	0	U

Tabelle 4.6: Erweiterte Wahrheitstabelle für ein NOR-Gatter.

Die Simulation des RS-Latch gestaltet sich dann wie in Tabelle 4.7 gezeigt. Insbesondere liefert nun der Übergang von $R=1$ und $S=1$ zu $R=0$ und $S=0$ die Werte U für die Ausgänge, d.h. der Anwender des Simulationsprogramms wird darauf hingewiesen, daß sinnvolle Ergebnisse nicht zu erzielen waren.

	R	S	P	Q	
1	1	1	0	0	Anfangszustand
2	U	U			externes Ereignis
3				U	
4			U		
5	0	0			externes Ereignis
6				U	
7			U		
8	0	0	U	U	stabiler Zustand

Tabelle 4.7: Simulation eines RS-Latch.

4.5 Modellierung des Laufzeitverhaltens

Um das Zeitverhalten einer Schaltung abschätzen zu können, werden die Logikwerte beim Durchgang durch ein Gatter mit einer spezifischen Verzögerung behaftet. Dies ist in Abbildung 4.4 für einen Inverter dargestellt.

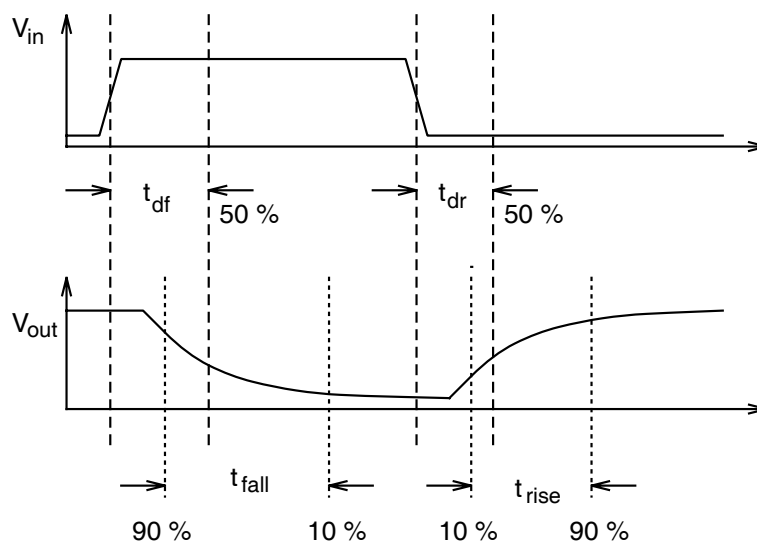


Abbildung 4.4: Verzögerungen an einem Inverter.

Zum einen ergeben sich aus den kontinuierlichen Signalverläufen die Zeiten, die zum Übergang von High auf Low (t_{fall}) und Low auf High (t_{rise}) zu berücksichtigen sind. Zu diesem Zweck werden jeweils die Zeitpunkte betrachtet, an denen das kontinuierliche Signal 10 % und 90 % des High-Pegels ausmacht. Außerdem kann man für den Inverter die Verzögerungszeit der fallenden Flanke (t_{df}) und der steigenden Flanke (t_{dr}) ablesen. Für diese Werte ist jeweils der Zeitpunkt wichtig, an dem das Eingangs- und Ausgangssignal genau 50 % des High-Pegels aufweist.

Für die Logiksimulation sind die Gatter normalerweise mit einer Verzögerungszeit für die steigende und die fallende Flanke behaftet. Falls keine Verzögerungszeiten vorliegen oder diese nicht berücksichtigt werden, spricht man von einer „zero-delay“ Simulation. In den Frühzeiten der CAD für Mikroelektronik ging man in manchen Fällen zur Vereinfachung des Simulationsablaufs von einer einheitlichen Verzögerung für alle Gatter aus. Diese Methode ist unter dem Schlagwort „unit delay“ bekannt und wird heute kaum noch eingesetzt.

Bei der Logiksimulation geht man von den in Abbildung 4.4 gezeigten kontinuierlichen Potentialen zu diskreten Logikwerten über. Um den Effekt der Verzögerung wenigstens teilweise berücksichtigen zu können, wird der reale Inverter mit einer Verzögerungszeit für die steigende Flanke (t_{dr}) und einer Verzögerungszeit für die fallende Flanke (t_{df}) versehen (Abbildung 4.5). Diese Zeiten müssen nicht notwendigerweise symmetrisch sein.

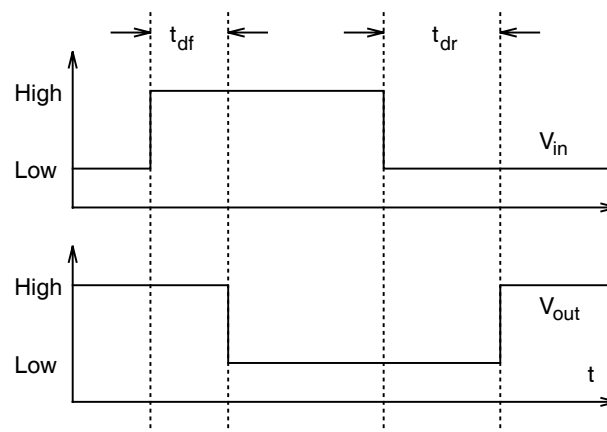


Abbildung 4.5: Idealisierte Verzögerungen an einem Inverter.

Damit kann ein realer Inverter durch einen idealen Inverter, der nur die Logikfunktion beinhaltet, und ein Verzögerungselement τ modelliert werden (Abbildung 4.6). Diese Verzögerungselemente sind zustandsbehaftet, d.h. ihre Verzögerung ist abhängig vom anliegenden Logikwert.

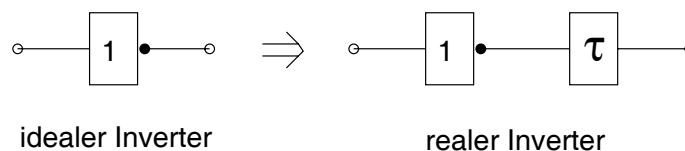


Abbildung 4.6: Modellierung eines realen Inverters.

4.6 Scheduling

Das Zusammenspiel von ereignisgesteuerter Simulation und der Betrachtung von Verzögerungen der einzelnen Gatter erfordert es, die Ereignisse in ihrer zeitlichen Abfolge zu betrachten. Zu diesem Zweck legt man eine Ereignistabelle (schedule) an, die die Ereignisse zusammen mit ihrem Zeitpunkt aufnimmt. Dies wird anhand des in Abbildung 4.7 dargestellten Beispiels verdeutlicht.

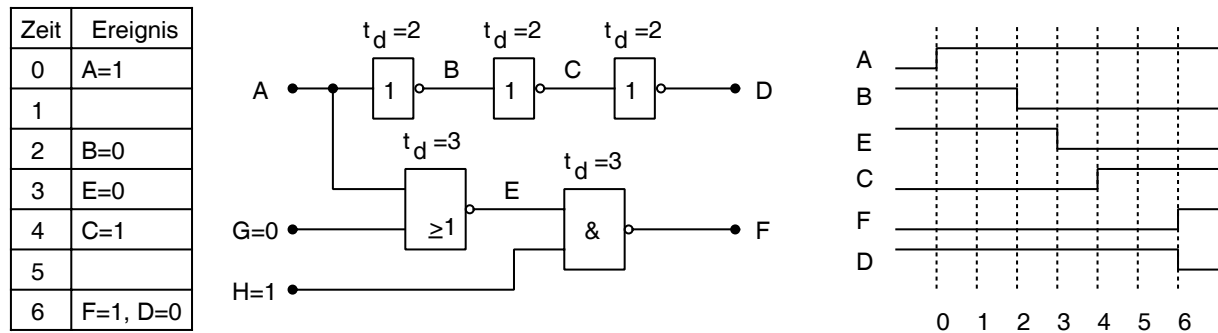


Abbildung 4.7: Aufstellung einer Ereignistabelle zur Logiksimulation.

Der Bereich, über den sich die noch abzuarbeitenden Ereignisse erstrecken, ist in aller Regel sehr klein. Er ist durch die längste Verzögerungszeit innerhalb der Schaltung begrenzt. Ereignisse, die in der Vergangenheit liegen, können aus dem Speicher gelöscht werden. Damit kann die Ereignisliste zyklisch angelegt werden, wobei die Länge des Zyklus durch die längste Verzögerungszeit bestimmt wird. Sie wird zu $t_{d_{max}} + 1$. Immer wenn das Ende der Liste erreicht ist, werden die nächsten Ereignisse wieder am Anfang der Liste eingetragen. Dies hält den Speicherplatzbedarf für die Ereignisliste gering. Eine zyklische Ereignisliste wird auch als Time-Wheel bezeichnet.

Kapitel 5

RC-Simulation

Die Analog-Simulation liefert zwar sehr genaue Werte; oftmals ist jedoch die benötigte Laufzeit viel zu hoch, um sinnvolle Simulationen durchführen zu können. Dies liegt zum einen daran, daß die Anzahl der Rechenoperationen, die für die Matrizenrechnungen benötigt werden, mit $O(n^3)$, $n \triangleq$ Anzahl der Bauelemente, anwächst. Außerdem muß das gesamte Gleichungssystem gelöst werden, auch wenn nur ein kleiner Teil der Schaltung neu berechnet werden müsste. Aus diesem Gründen zieht man es oftmals vor, weniger genaue Simulationsverfahren wie zum Beispiel die RC-Simulation anzuwenden. Hier steht die Betrachtung des Zeitverhaltens der Schaltung im Vordergrund. Die Abstraktion gegenüber der Analog-Simulation besteht darin, daß die Transistoren als ideale Schalter betrachtet werden, die mit einem Widerstand in Reihe geschaltet sind, siehe Abbildung 5.1. Für Gatespannungen oberhalb eines bestimmten Schwellwertes ist der Schalter geschlossen (NMOS-Transistor), ansonsten ist er offen. Für MOS Transistoren in einer Digitalschaltung ist diese Betrachtungsweise in der Regel ausreichend.

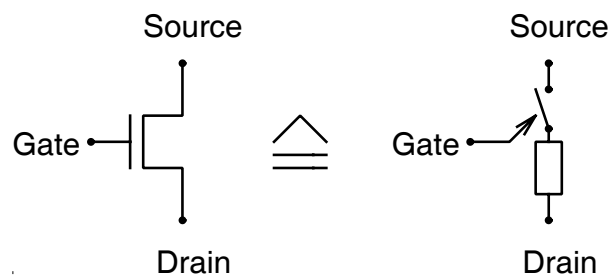


Abbildung 5.1: Schaltermodell eines MOS-Transistors.

Das Zeitverhalten läßt sich anhand dieses einfachen Modells durch die Betrachtung der Zeitkonstanten der Transistorwiderstände und der Lastkapazitäten berechnen. Die Lastkapazitäten ergeben sich aus der jeweiligen Summe der Kapazitäten der Gates, die von einem Knoten aus angesteuert werden. Anhand des folgenden Beispiels eines OR-Gatters (NOR + Inverter) läßt sich dies recht gut veranschaulichen.

Der Schaltungsknoten N_1 wird über die parallel geschalteten Widerstände R_3 und R_4 , sowie über die Kapazität C_{Last} entladen. Die Zeitkonstante ergibt sich damit zu:

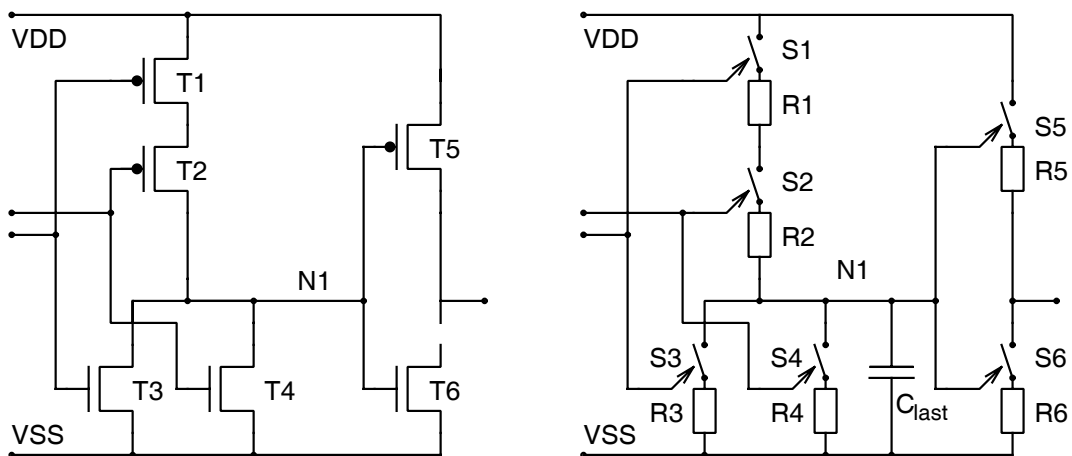


Abbildung 5.2: OR-Gatter auf Schalterniveau.

$$t = R_3 \parallel R_4 \cdot C_{Last} \quad (5.1)$$

Daraus ergibt sich eine entsprechende Verzögerung, die in Abbildung 5.3 aufgetragen wird. Diese Verzögerung ist ein gutes Modell für die tatsächlich auftretenden Laufzeiteffekte.

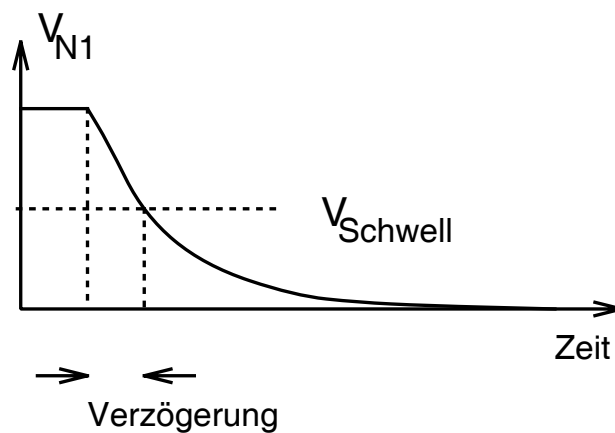


Abbildung 5.3: Verzögerung im Schaltermodell.

Viele Schaltungssituationen, die auf Gatterniveau nicht befriedigend gelöst werden können, sind von der RC-Simulation bei moderaten Laufzeitanforderungen bearbeitbar. Stellvertretend sollen hier zwei Beispiele gegeben werden: die Simulation eines NMOS Pass-Transistors und einer Konkurrenzsituation beim Buszugang.

Die Situation am Pass-Transistor ist in Abbildung 5.4 dargestellt. Je nachdem ob der High-Übergang oder der Low-Übergang des vorgeschalteten Inverters betrachtet wird, ergibt sich eine etwas andere Schaltung.

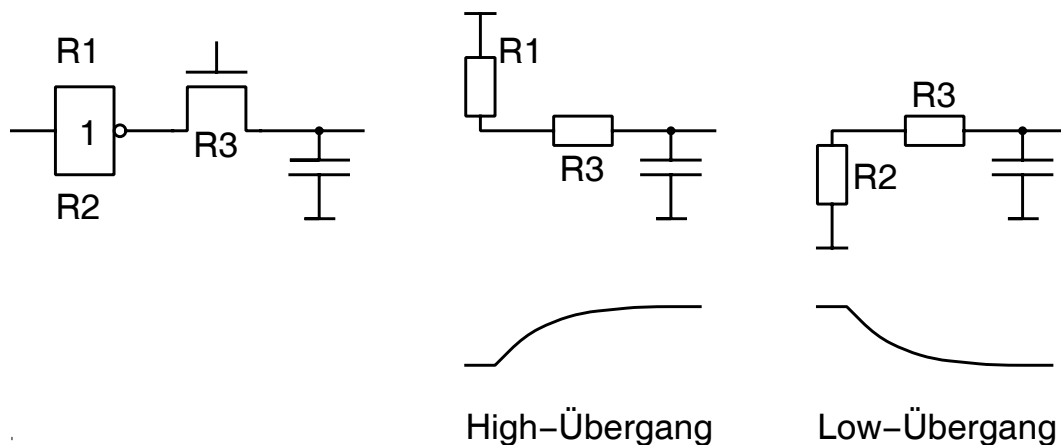


Abbildung 5.4: RC-Simulation eines Transferrgatters.

Die Konfliktsituation, falls zwei Treiber gleichzeitig Nachrichten auf den Bus senden wollen, ist in Abbildung 5.5 dargestellt. Auch hier ergibt sich mit der Modellierung eine sinnvolle Lösung. Die Ausgangsspannung V_{N_1} ergibt sich durch den Spannungsteiler zu:

$$V_{N_1} = \frac{R_4 + R_6}{R_1 + R_3 + R_4 + R_6} V_{dd} \quad (5.2)$$

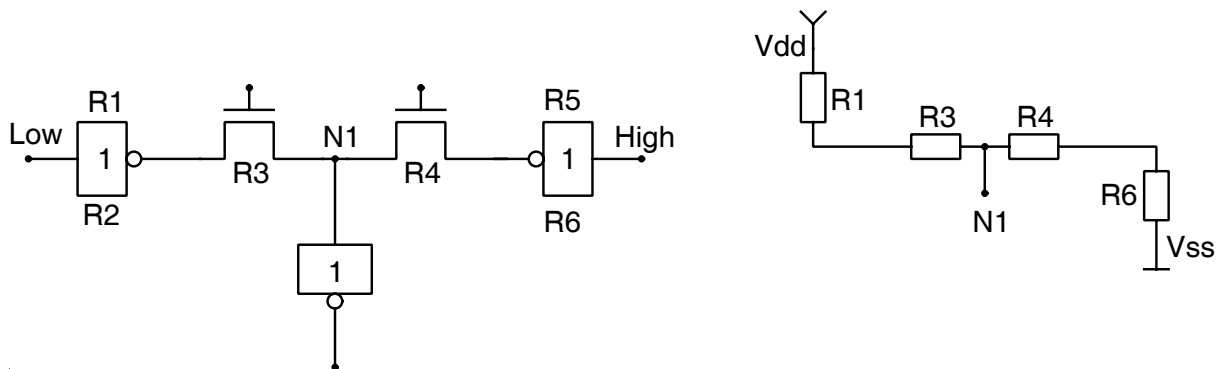


Abbildung 5.5: RC-Simulation für den konkurrierenden Buszugriff.

Zu ähnlichen Ergebnissen kommt man, wenn man die Widerstände nicht explizit betrachtet, sondern ihren Effekt durch eine „Signalstärke“ der Logikwerte modelliert. Jedem Schaltungsknoten ist neben einem der drei üblichen Logikwerte $\{0, 1, X\}$ auch eine Signalstärke zugeordnet. Folgende Signalstärken werden häufig verwendet: $\{\text{driven, weak, charged}\}$. Die Signalstärke „driven“ ergibt sich dann, wenn das Signal durch einen Treiber oder einen Gatterausgang eingestellt wird. Falls es über ein oder mehrere Transferrgatter oder Pass-Transistoren mit entsprechenden Source-Drain Widerständen geleitet worden ist, entspricht seine Stärke dem Wert „weak“. Für hochohmige Knoten schließlich bleibt der alte Signalwert durch die unvermeidlichen, parasitären Kapazitäten gespeichert; die Signalstärke wird mit „charged“ bezeichnet. Je stärker das Signal ist, umso durch-

setzungsfähiger ist es, falls die Situation eintritt, daß ein Knoten von zwei Signalen angesteuert wird. Driven-high verknüpft mit weak-low ergibt driven-high; weak-high und charged-low ergibt weak-high. Bei gleicher Signalstärke und konkurrierenden Signalwerten ergibt sich jeweils ein X; so wird driven-high verknüpft mit driven-low zu driven-X. Die Simulation mit verschiedenen Signalstärken wird auch Schaltersimulation (switch-level simulation) genannt.

Kapitel 6

Hardware-Beschreibungssprachen

6.1 Motivation und Hintergrund

Mit der Möglichkeit Schaltungen mit mehreren Millionen Transistoren zu fertigen, stellt sich natürlich die Frage, wie die Designaufgabe in den Griff zu bekommen ist. Vielleicht die wichtigste Strategie in diesem Zusammenhang ist die von oben nach unten gerichtete „Top-Down“-Entwurfsmethodik. Bis zum Ende der 80’er Jahre war es üblich, Schaltpläne zu zeichnen und ausgehend von Logikgattern immer komplexere Gebilde zu entwickeln. Dabei bewegt man sich in der Abstraktion von unten nach oben, also „Bottom-Up“. Eine Komponente kann erst dann eingesetzt und simuliert werden, wenn all ihre Teilkomponenten und deren Teilkomponenten bereits vollständig definiert sind. Nur dann ist es möglich, die entstandene Hierarchie für einen Logiksimulator auf Gatterniveau zu expandieren. Letztlich bestimmte hier das (Simulations-)Werkzeug die Entwurfsmethodik. Problematisch ist die „Bottom-Up“-Methode vor allem deshalb, weil die zu lösende Aufgabe in der Regel auf sehr abstraktem Niveau spezifiziert wird. Für einen Mikroprozessor gehört der Befehlssatz sicherlich zur Spezifikation. Von dieser Spezifikation ausgehend möchte man direkt in den Entwurfsprozess einsteigen und den Entwurf schrittweise verfeinern. Die Einzelheiten auf Gatterniveau sollten - wenn überhaupt - erst sehr spät im Entwurfsablauf zu berücksichtigen sein.

Es stellt sich nun die Frage, wie man eine Schaltung auf abstraktem Niveau beschreiben soll, wenn die Implementierung ihrer Komponenten noch nicht feststeht. Mit einem Schaltplan ist dies nicht zu leisten, da hier ein abstrakter Block ja gerade durch seine Komponenten beschrieben wird. Dieses Problem wurde durch die sogenannten Hardwarebeschreibungssprachen (engl.: hardware description language, HDL¹) adressiert. Diese erlauben die Formulierung von Komponenten-Modellen auf Basis ihres Verhaltens und nicht nur ihrer Struktur. Eine solche Verhaltensbeschreibung läßt die innere Struktur einer Komponente außer acht und beschränkt sich auf die Berücksichtigung des Verhaltes in Funktion und Timing an seinen Klemmen. Eine solche Beschreibung ist genauso simulierbar wie eine Gatternetzliste und bietet die Grundlage für eine weitere Verfeinerung.

¹Der Begriff 'HDL' wird im folgenden durchgängig anstatt des Wortungetüms Hardwarebeschreibungssprache verwendet. Beide sind zueinander synonym.

Der endgültige Durchbruch der „Top-Down“-Entwurfsmethodik wurde schließlich durch die Entwicklung von Verfahren zur Schaltungssynthese erreicht. Diese bilden HDL-Beschreibungen einer Schaltung in detailliertere HDL-Beschreibungen ab, wobei die unterste Ebene durch eine - ebenfalls in einer HDL formulierbare - Gatternetzliste gebildet wird. Im industriellen Einsatz ist bereits die Logiksynthese, die ausgehend vom Register-Transferriveau Gatterschaltungen generiert. Diese lassen sich anschließend automatisch auf ein Layout oder ein FPGA abbilden. Die Synthese auf Basis reiner Verhaltensmodelle (high level synthesis) ist zur Zeit noch im Forschungsbereich anzusiedeln. Immerhin sind die unteren Ebenen des Digitalschaltungsentwurfs durch Synthese im wesentlichen automatisierbar, was vielleicht nicht in jedem Fall zu optimalen Schaltungen führt, aber massiv Entwurfszeit spart und die Fehlermöglichkeiten eines Schaltungsentwicklers drastisch einschränkt.

6.2 Anwendungsgebiete von HDLs

Wie bereits in der Einleitung angedeutet, gibt es eine ganze Reihe von Anwendungsgebieten für eine HDL. Die wichtigsten sind Spezifikation, Simulation, Synthese, Test und formale Verifikation.

6.2.1 Spezifikation / Dokumentation

Eine formalisierte Schaltungsbeschreibung auf Verhaltensebene, wie sie durch eine HDL gegeben ist, bildet eine präzise Spezifikation und Dokumentation für eine Schaltung. In vielen Fällen haben sich bei informellen Spezifikationen im nachhinein Probleme ergeben, da beispielsweise bestimmte Betriebszustände nicht vorausgesehen und damit auch nicht spezifiziert waren. Mit einer formalen, programmähnlichen Spezifikation werden diese Schwierigkeiten entschärft. Hier wird in der Regel sofort deutlich, wenn ein System nur unvollständig spezifiziert ist. Außerdem steht bei allen Streitfällen die HDL-Beschreibung als Referenz zur Verfügung. Im Zweifel wird simuliert und das Ergebnis gilt. Außerdem ergibt sich auf diesem Wege automatisch der Einstieg in einen durchgängigen Entwurfsablauf. Auf Basis abstrakterer Beschreibungen werden immer detailliertere Darstellungen entworfen oder generiert, wobei diese Beschreibungen gegeneinander verifiziert werden können. Auf diese Weise wird das Problem der Konsistenz zwischen der textuellen Spezifikation eines Pflichten- oder Lastenheftes und dem entwickelten System umschifft. Letztlich löst dieser Ansatz nicht nur die Frage „Entwickle ich das System richtig?“, sondern vor allem die Frage „Entwickle ich das richtige System?“.

6.2.2 Simulation

Der Entwurf heutiger Schaltungen ist ohne Simulation nicht mehr denkbar. Für praktisch alle HDLs gibt es einen oder sogar mehrere Simulatoren. Damit eröffnen die HDLs die Möglichkeiten der Top-Down Entwurfsmethodik. Die Simulation von HDL-Beschreibungen ähnelt stark der Simulation von Gatterschaltungen. Auch hier verläuft die Simulation ereignisgesteuert, d.h. es werden jeweils nur die Komponenten neu berechnet, deren Eingänge sich geändert haben. Die Verwandtschaft zwischen HDL-Simulation und Gattersimulation ist auch nicht weiter verwunderlich, da eine HDL-Beschreibung ja durchaus auch aus einer Zusammenschaltung von Logikgattern

bestehen kann. Ein HDL-Simulator muß in seiner Funktion also eine Obermenge eines Gattersimulators simulieren können.

6.2.3 Synthese

Wie oben angedeutet, besteht der Entwurf einer Schaltung oftmals in einer schrittweisen Verfeinerung einer Hardware-Beschreibung. Gerade für Übergänge auf den unteren Ebenen, wie etwa zwischen Registertransfer- und Gatterniveau, gibt es eine ganze Reihe von Methoden und Werkzeugen, die diesen Schritt automatisieren. Typischerweise endet der Entwurfsvorgang mit einer Gatternetzliste. Hier schließen sich direkt weitere Synthesewerkzeuge an, die die Gatterbeschreibung auf deren Realisierung, wie etwa ein Gate-Array oder ein CPLD abbilden.

6.2.4 Test

Einige Werkzeuge zur automatischen Generierung von Testmustern und zur Analyse der Testbarkeit einer Schaltung bauen auf textuellen Hardwarebeschreibungen auf. HDLs bieten mit ihrem Modellierungsinstrumentarium in aller Regel auch gleich die Möglichkeit, Testmuster zu formulieren. Damit ist eine HDL auch für diesen 'Betriebsfall' optimal ausgerüstet; es ist kein zusätzliches Format für die Testmuster notwendig.

6.2.5 Formale Verifikation

Die Verifikation der Funktion einer Schaltung wird in der Regel durch Simulation erreicht. Es gibt allerdings auch einige andere Ansätze. So versucht man auf mathematischer Ebene die korrekte Funktion einer Schaltung zu 'beweisen'. Gerade bei sehr großen Schaltungen ist eine Simulation auf Gatterebene zur Verifikation des Schaltungsentwurfs kaum noch möglich. Hier sind systematischere Methoden gefragt, um zu überprüfen, ob die Gatterschaltung die Funktion der Schaltung hinreichend genau abbildet. Die Beschreibung der Schaltung basiert auch hier auf HDLs.

6.3 Charakterisierung einer HDL

Eine HDL ähnelt auf den ersten Blick einer Programmiersprache, wie C oder Pascal. Es gibt eine Reihe von Schlüsselworten, denen eine besondere Bedeutung zukommt. Eine spezielle Syntax muß eingehalten werden. Schaltungsmodelle werden in ein Zwischenformat übersetzt, bevor eine Simulation ablaufen kann. Es gibt allerdings auch eine Reihe gravierender Unterschiede zu Programmiersprachen. So läuft ein Programm im Normalfall sequentiell ab, d.h. zu einem bestimmten Zeitpunkt wird immer nur eine Anweisung abgearbeitet. Dies ist für die Beschreibung von Hardware nicht akzeptabel. So arbeiten alle Gatter einer Logikschaltung grundsätzlich parallel.

Welche Abstraktionsebenen einer Schaltung können nun mit einer HDL beschrieben werden? Wie bereits oben angedeutet, hat es sich bei der Entwicklung mikroelektronischer Schaltungen als ausgesprochen hilfreich erwiesen, den Entwurf zunächst auf sehr abstrakter Ebene durchzuführen und ihn dann schrittweise immer weiter zu verfeinern. Eine HDL trägt diesem Ablauf dadurch Rechnung, daß sie Modellierungsmöglichkeiten auf verschiedenen Abstraktionsebenen anbietet. Ein

Entwurfsschritt dient dazu, eine HDL-Beschreibung in eine andere HDL-Beschreibung auf niedrigerem Abstraktionsniveau abzubilden. Somit verläuft der Entwurfsprozess weitgehend innerhalb der HDL. Typischerweise deckt eine HDL Schaltungsmodelle bis herunter zum Logikniveau im Verhaltens- und Strukturbereich ab, siehe Abbildung 6.1. Die Bauelemente-Ebene wird nur rudimentär bzw. im Rahmen von Erweiterungen unterstützt. So wird im Moment ein zusätzlicher Standard (VHDL-AMS) entwickelt, der dazu dienen soll, neben den digitalen Komponenten auch analoge zu modellieren. Das Layout hingegen wird in einer HDL nicht betrachtet.

	Verhalten, Funktion	Struktur
System	Spezifikation, Prozesse	Prozessor, Memory
Algorithmus	I/O-Algorithmen	Module
Register-Transfer Ebene	RT-Beschreibung, Datenfluß	Funktionseinheiten
Logik	Boole'sche Gleichung	Gatter, Flipflop

Abbildung 6.1: Sichten, die mit einer HDL modelliert werden können.

Wie bereits oben angedeutet, kann mit einer HDL sowohl das Verhalten als auch die Struktur einer Schaltung beschrieben werden. Der hier wesentliche Unterschied zwischen Verhalten und Struktur soll im folgenden kurz verdeutlicht werden. Eine Addition von vier Zahlen kann, was ihre Funktion betrifft, eindeutig beschrieben werden:

$$y = a + b + c + d$$

Die Reihenfolge der Auswertung des Ausdrucks ist dabei unerheblich, da das Kommutativgesetz für die Addition gilt. Falls die Addition auf der strukturellen Ebene betrachtet wird, kann die Reihenfolge nicht mehr vernachlässigt werden. Alternativ ergeben sich zum Beispiel die folgenden zwei Möglichkeiten:

$$y = (a + b) + (c + d)$$

$$y = ((a + b) + c) + d$$

Eine entsprechende Realisierung durch Addierer ist in Abbildung 6.2 angegeben. Dabei stellt sich heraus, daß die linke Realisierung des Ausdrucks schneller arbeitet als die rechte.

Ein wesentlicher Vorteil einer HDL besteht darin, daß die Komplexität der Schaltungsbeschreibung erheblich reduziert wird. So stehen höhere Operationen wie die Addition, Subtraktion, Multiplikation oder Division für jeweils einige hundert oder sogar einige tausend Gatter. Dies bedingt, daß die Lesbarkeit der Beschreibung in der Regel wesentlich über der anderer Schaltungsbeschreibungen liegt. Damit erleichtert sich auch die Wiederverwendung von Teilbeschreibungen, die ursprünglich für andere Schaltungen entwickelt worden sind. Schließlich eröffnen Hardwarebeschreibungssprachen in der Regel die Möglichkeit, die einzelnen Teilschaltungen in unterschiedlichen Abstraktionen zu betrachten, siehe Abbildung 6.3. Damit können Schaltungen oder Systeme bereits komplett

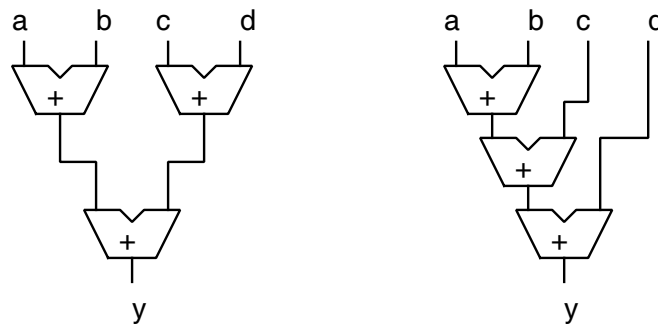


Abbildung 6.2: Alternative Strukturrealisierungen eines Additionsausdrucks.

simuliert werden, wenn jedes ihrer Module eine abstrakte Verhaltensbeschreibung besitzt. Dies bietet zunächst einen Geschwindigkeitsvorteil gegenüber einer Komplettsimulation des fertigen Entwurfs. Außerdem können die einzelnen Blöcke nach und nach im Rahmen des Entwurfsablaufs verfeinert werden, bis der Entwurf die benötigte Abstraktion für die einzelnen Teile erreicht hat. Insbesondere können die Verfeinerungen von mehreren Schaltungsentwicklern *unabhängig* voneinander durchgeführt werden. Durch diese „Verschränkung“ der Ingenieursarbeit (concurrent engineering) kann die Entwurfszeit selbst für sehr komplexe Systeme im Rahmen gehalten werden. In Zukunft werden Methoden zur Partitionierung der Ingenieursarbeiten weiter an Bedeutung gewinnen, da die organisatorische Beherrschung komplexer, stark verkoppelter Systeme in Zukunft immer mehr zur Einschränkung des Machbaren beitragen wird.

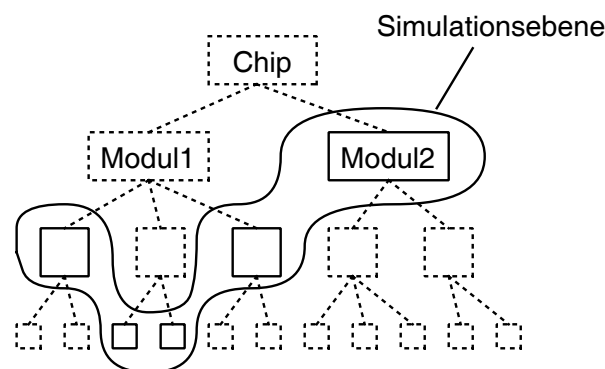


Abbildung 6.3: Simulation auf heterogenem Abstraktionsniveau.

Abschließend ist noch zu betonen, daß die Benutzung von HDLs und insbesondere die Festlegung auf die synthetisierbaren Konstrukte einer HDL wiederum zu einer erheblichen Einschränkung der Designfreiheit führen. Verallgemeinert kann man sagen, daß eine geeignete Beschränkung der Freiheitsgrade beim Entwurf oftmals erst die Möglichkeit zur Automatisierung von Synthesefunktionen eröffnet, die für den allgemeinen Fall nicht besteht.

6.4 Sprachen

Viele HDLs wurden in der letzten Jahren definiert. Einige der verbreiteteren HDLs wurden von Softwareanbietern eingeführt und dienen als Grundlage für den Entwurf und die Synthese digitaler Systeme. Typische Vertreter aus dieser Gruppe sind die M-HDL der Mentor Graphics Corporation oder die Verilog-HDL von Cadence Design Systems. Man spricht in diesem Zusammenhang auch von proprietären HDLs. Eine weitere Gruppe von HDLs stammt aus dem universitären Bereich, wie z.B. BDS von der University of California, Berkeley oder daCapo von der Universität Dortmund. Diese Sprachen haben allerdings nur im akademischen Bereich Verbreitung gefunden. Sie bilden jedoch mit ihren innovativen Ideen oftmals die Grundlage für kommerzielle Beschreibungssprachen.

Die zur Zeit wichtigste Sprache VHDL (VHSIC² hardware description language) stammt aus keinem der beiden vorgenannten Bereiche. Sie ist das Produkt der Normungsbestrebungen eines Komitees, in dem die meisten größeren CAD-Anbieter und CAD-Nutzer, aber auch Vereinigungen wie die IEEE³ vertreten sind. Der große Vorteil solcher Standards liegt darin, daß dieselbe Schaltungsbeschreibung mit den Programmen der verschiedensten CAD-Anbieter verarbeitet werden kann. Nachteilig ist allerdings, daß jeder Anbieter möglichst viel von seinen bislang verwendeten Konstrukten in den Standard retten will. Damit tendieren solche Standards dazu, einen gewaltigen Befehlsumfang anzunehmen, was wiederum ihre Verbreitung hemmt. Im Falle von VHDL hat der im nordamerikanischen Bereich bei weitem größte Anwender, das amerikanische Verteidigungsministerium, dem Standard dadurch zum Durchbruch verholfen, daß es dessen Einhaltung zur notwendigen Voraussetzung für die Erteilung von Aufträgen gemacht hat. Damit waren alle CAD-Anbieter gezwungen, VHDL zu unterstützen. Seit dem Jahr 1987 ist VHDL ein Standard, der laufend verbessert und erweitert wird. So macht man sich Gedanken um eine Erweiterung auf den analogen Bereich und spricht in diesem Zusammenhang von VHDL-AMS (VHDL analog / mixed signal).

²VHSIC = very high speed integrated circuits, amerikanisches Förderprogramm zur Entwicklung besonders leistungsfähiger integrierter Schaltungen.

³IEEE = institute of electrical and electronics engineers (sprich: Ei-Trippel-Ih), kann als internationaler VDE betrachtet werden.

Kapitel 7

Logik-Synthese

Nachdem in den vorigen Kapiteln die Simulation also Analyse von Schaltungen betrachtet wurde, soll es jetzt um die Synthese solcher Schaltungen gehen. Basis einer solchen Synthese ist eine Beschreibung auf Register-Transfer-Niveau die typischerweise in einer Hardware-Beschreibungssprache wie etwa VHDL formuliert wird. In einem ersten Schritt werden die HDL-Konstrukte in ihre direkte Entsprechung in der Logik übersetzt. So wird etwa aus einer bedingten Zuweisung ein Multiplexer dessen Steuerung der Bedingung der Zuweisung folgt. Einige Beispiele dafür finden sich im Anhang über die Hardware-Beschreibungssprache VHDL.

Nun gibt es für jede Logikfunktion unendlich viele Darstellungen, die sich in ihrem Aufwand erheblich unterscheiden. Oftmals sind solche Darstellungen eher ungünstig, wenn sie im Rahmen eines Syntheselaufs aus einer höheren Beschreibung generiert worden sind. Um nun zu einer akzeptablen Schaltung zu kommen, sind zwei Operationen notwendig: die Minimierung von Logik und die Abbildung auf bestimmte Sätze von grundlegenden Logikfunktionen (z.B. NAND, NOR etc.). Der erste Punkt dient einer möglichst effizienten Umsetzung der Logikfunktionen, während der zweite einer günstigen Abbildung der Logikfunktionen auf einen gegebenen Satz von Standardzellen oder Gate-Array-Zellen gewidmet ist.

7.1 Logikminimierung

Die Logikminimierung basiert im wesentlichen auf den grundlegenden Arbeiten von Boole und Shannon:

- Laws of Thought (Boole) 1850
- Mathematical Switching Theory (Shannon) 1938

Das Gerüst bildet dabei die boole'sche Algebra, die im folgenden definiert werden soll.

Definition 7.1 (boole'sche Algebra)

Eine boole'sche Algebra ist ein 6-tupel $(B, +, \cdot, \neg, 0, 1)$

$$(1) \quad B \text{ (Traegermenge, z.B. } B = \{0, 1\})$$

$$(2) \quad +, \cdot : B^2 \rightarrow B \quad (\text{'Addition' } \hat{=} \text{OR, 'Multiplikation' } \hat{=} \text{AND})$$

$$(3) \quad - : B \rightarrow B \quad (\text{'Komplement' } \hat{=} \text{NOT})$$

$$(4) \quad 0, 1 \in B \quad (\text{'Neutrale Elemente'})$$

Folgende Gesetze gelten innerhalb einer boole'schen Algebra:

(B1, Kommutativität)

$$\forall a, b \in B : a + b = b + a, a \cdot b = b \cdot a$$

(B2, Distributivität)

$$\forall a, b, c \in B : (a + b) \cdot c = (a \cdot c) + (b \cdot c), (a \cdot b) + c = (a + c) \cdot (b + c)$$

(B3, Nullelement, Einselement)

$$\forall a \in B : a + 0 = a, a \cdot 1 = a$$

(B4, Komplement)

$$\forall a \in B : a + \bar{a} = 1, a \cdot \bar{a} = 0$$

(B5)

$$\exists a, b \in B : a \neq b$$

Definition 7.2 (boole'sche Funktion)

Innerhalb einer boole'schen Algebra sind bereits einfache Funktionen wie die Addition oder die Multiplikation definiert. Eine Abbildung

$$f : B^n \rightarrow B$$

heißt boole'sche Funktion mit n Eingangsvariablen.

Aus den obigen Definitionen folgt noch eine weitere wichtige Gesetzmäßigkeit, der Satz von de Morgan.

Satz 7.1 (de Morgan)

$$\forall a, b \in B : \overline{a \cdot b} = \bar{a} + \bar{b}$$

Das Karnaugh-Veitch-Diagramm¹ ist die übersichtlichste Form zur Darstellung und Minimierung einer logischen Funktion. Ein Karnaugh-Veitch-Diagramm besteht aus einer Matrix, bei der jedes Matrixelement eine Belegung der Eingangsvariablen mit Nullen und Einsen repräsentiert. Für

¹Synthesis of Logic Circuits (Karnaugh) 1953

n Variablen liegt die Anzahl der Matrixelemente bei 2^n . Die Elemente der Matrix sind so angeordnet, daß sich senkrecht oder waagerecht benachbarte Elemente nur bezüglich des Wertes einer einzigen Logikvariablen unterscheiden. Die Logikfunktion ist dadurch bestimmt, daß einer bestimmten Menge von Matrixelementen eine 1 zugeordnet wird, den anderen eine 0. Dieser Wert entspricht dem Ergebnis der Logikfunktion für die entsprechende Belegung der Eingangsvariablen. Aus dem Karnaugh-Veitch-Diagramm läßt sich direkt ein geschlossener Ausdruck für die Logikfunktion ableiten. Dieser Ausdruck setzt sich aus einer Summe von Produkten zusammen, wobei für jedes Matrixelement, dessen Wert eins war, ein Produktterm aufgestellt wird. Jeder dieser Terme enthält alle Eingangsvariablen in invertierter oder nicht-invertierter Form. Falls die betreffende Eingangsvariable des aktuellen Matrixelements 0 war, so enthält der Produktterm die Variable in invertierter Form; im anderen Fall wird die Variable nicht invertiert. Diese Produktterme werden auch Minterme genannt. Der Zusammenhang zwischen den Matrixelementen und den Mintermen wird durch die Abbildung 7.1 verdeutlicht.

		$\bar{b}$	b	b	$\bar{b}$	
		$\bar{a}$	$\bar{a}$	a	a	
$\bar{d}$	$\bar{c}$	0	4	12	8	0 $\bar{d}\bar{c}\bar{d}$
d	$\bar{c}$	1	5	13	9	1 $\bar{d}\bar{c}d$
d	c	3	7	15	11	2 $\bar{d}cd$
$\bar{d}$	c	2	6	14	10	3 $\bar{d}c\bar{d}$

0	$\bar{d}\bar{c}\bar{d}$	8	$ab\bar{c}\bar{d}$
1	$\bar{d}\bar{c}d$	9	$\bar{d}cd$
2	$\bar{d}cd$	10	$abcd$
3	$\bar{d}cd$	11	$abcd$
4	$ab\bar{d}$	12	$ab\bar{c}\bar{d}$
5	$ab\bar{c}d$	13	$ab\bar{c}d$
6	$abcd$	14	$abcd$
7	$abcd$	15	$abcd$

Abbildung 7.1: Karnaugh-Veitch-Diagramm und Minterme.

Die Anwendung von Karnaugh-Veitch-Diagrammen zur Logikminimierung soll nun anhand eines Beispiels erläutert werden. Gegeben ist die in Abbildung 7.2 dargestellte Belegung der Matrixelemente mit Nullen und Einsen. Diese Belegung entspricht einer Logikfunktion in den Variablen a, b, c und d .

		$\bar{b}$	b	b	$\bar{b}$	
		$\bar{a}$	$\bar{a}$	a	a	
$\bar{d}$	$\bar{c}$	1	0	0	1	
d	$\bar{c}$	0	1	1	0	
d	c	1	1	1	1	
$\bar{d}$	c	1	0	0	1	

		$\bar{b}$	b	b	$\bar{b}$	
		$\bar{a}$	$\bar{a}$	a	a	
$\bar{d}$	$\bar{c}$	1	0	0	1	
d	$\bar{c}$	0	1	1	0	
d	c	1	1	1	1	
$\bar{d}$	c	1	0	0	1	

Abbildung 7.2: Beispiel für die Logikminimierung mit Karnaugh-Veitch-Diagrammen.

Im Karnaugh-Veitch-Diagramm werden nun Gruppen von Einsen zusammengefasst. Da sich benachbarte Felder nur in einer einzigen Logikvariablen unterscheiden, kann man aufgrund des folgenden Zusammenhangs einfachere Formeln für die Gruppen angeben.

$$axyz + \bar{a}xyz = (a + \bar{a})xyz = xyz$$

Die in Abbildung 7.2 dargestellte Logikfunktion f lässt sich anhand der eingezeichneten Gruppen wie folgt aufschreiben:

$$f = bd + cd + \bar{b}\bar{d} + \bar{b}c$$

Bevor die Methoden zur Logikminimierung im einzelnen besprochen werden, sollen zunächst noch einige grundlegende Definitionen gegeben werden.

Definition 7.3 (Implikant)

Ein Implikant k einer Logikfunktion f ist ebenfalls eine Logikfunktion mit der folgenden Eigenschaft. Für alle Belegungen der Eingangsvariablen gilt:

$$k(a, b, c, \dots) = 1 \Rightarrow f(a, b, c, \dots) = 1$$

Immer wenn der Implikant einer Funktion für eine Belegung der Logikvariablen zu 1 wird, so wird auch die Funktion mit derselben Belegung zu 1. Man sagt auch: der k impliziert f .

Definition 7.4 (Primimplikant)

Ein Implikant k einer Funktion f ist ein Primimplikant genau dann, wenn es keinen weiteren Implikanten k' von f gibt, der eine Verkürzung von k darstellt.

Eine Verkürzung eines Implikanten bedeutet immer, daß er umso mehr Minterme überdeckt, d.h. die Anzahl der Minterme, deren Summe genau die Funktion des Implikanten ergibt, ist entsprechend größer. Je kürzer die Implikanten werden, umso kompakter kann eine Funktion repräsentiert werden. Im Beispiel von Abbildung 7.2 lauten die Primimplikanten wie folgt: bd , $\bar{b}\bar{d}$, cd und $\bar{b}c$.

Definition 7.5 (Kernprimimplikant)

Ein Kernprimimplikant ist ein Primimplikant, der Minterme überdeckt, die von keinem anderen Primimplikanten überdeckt werden.

Die Kernprimimplikanten der Beispielfunktion lauten bd und $\bar{b}\bar{d}$.

Definition 7.6 (minimale Überdeckung)

Eine minimale Überdeckung der Minterme einer Logikfunktion ergibt sich aus den Kernprimimplikanten und einer minimalen Menge weiterer Primimplikanten, die die restlichen Minterme überdecken.

Damit kann ein exaktes Verfahren für die Logikminimierung angegeben werden:

Algorithmus 7.1 (Logikminimierung)

*Aufstellung der Minterme der Funktion;
Bestimmung aller Primimplikanten;
Bestimmung einer minimalen Überdeckung;
Ergebnis: minimale zweistufige Logik;*

Die Karnaugh-Veitch-Diagramme bieten eine einfache und anschauliche Möglichkeit zur Logikminimierung. Es ergeben sich jedoch zwei Probleme: für mehr als vier Variablen wird die graphische Darstellung unhandlich; bei n Eingangsvariablen liegt die Zahl der Matrixelemente bei 2^n . Außerdem ist die Zusammenfassung von Gruppen für den Menschen ein einfacher Prozeß der Mustererkennung. Für einen Rechner ist das Problem jedoch ungleich schwieriger, da Mustererkennung bis heute zu den Aufgaben gehört, die nur unvollkommen durch einen Rechner erledigt werden können. Aus diesem Grund wurden einige andere Methoden zur Aufstellung von Primimplikanten und zur Berechnung von Überdeckungen entwickelt, die sich leichter auf einem Rechner realisieren lassen.

Die Bestimmung der Primimplikanten ist ein Problem, für das eine ganze Reihe von Algorithmen entwickelt worden sind. An dieser Stelle soll der wohl einfachste und auch älteste Ansatz beschrieben werden, die Methode von Quine² und McCluskey³. Der Algorithmus besteht darin, sämtliche Minterme der Funktion aufzustellen und dann auf je zwei Terme eine Vereinfachungsoperation anzuwenden. Diese Operation wird üblicherweise als „simpler Konsensus“ bezeichnet.

Definition 7.7 (simpler Konsensus)

Seien ξ ein Produktterm und $x \cdot \xi$ und $\bar{x} \cdot \xi$ Produktterme, die die gewünschte Funktion implizieren. Die beiden Produktterme dürfen durch ξ ersetzt werden, weil:

$$x \cdot \xi + \bar{x} \cdot \xi = (x + \bar{x}) \cdot \xi = \xi$$

Der simple Konsensus wird solange wie möglich auf die Minterme oder die schon verkürzten Produktterme angewendet. Wenn er nicht mehr anwendbar ist, ist aus der Menge der Minterme die Menge der Primimplikanten geworden.

Aus der Abbildung 7.1 ergibt sich ein Index für die Minterme der in Abbildung 7.2 dargestellten Funktion. Folgende Minterme müssen für die Funktion überdeckt werden: $m_0, m_2, m_3, m_5, m_7, m_8, m_{10}, m_{11}, m_{13}$ und m_{15} . Diese Minterme werden nach der Anzahl der in ihnen vorkommenden nicht-invertierten Variablen - mit einer '1' gekennzeichnet - sortiert und in die Tabelle 7.1 eingetragen. Grundsätzlich können nur Minterme aus benachbarten Gruppen einen Konsensus aufweisen. Anschließend werden alle in Frage kommenden Mintermpaare auf einen simplen Konsensus untersucht und die Ergebnisse in Tabelle 7.2 eingetragen. Die neu entstandenen Terme werden dann ihrerseits auf einen simplen Konsensus untersucht, siehe Tabelle 7.2, untere Hälfte. Aus der Tabelle 7.2 lassen sich die Primimplikanten direkt aus dem unteren Abschnitt ablesen: $\bar{b} \cdot \bar{d}$, $\bar{b} \cdot c$, $c \cdot d$ und $b \cdot d$.

²Simplifications of truth functions (1952).

³Minimization of boolean functions (1956).

Anzahl der nicht-inv. Variablen	a	b	c	d	Minterm
0	0	0	0	0	m_0
1	1	0	0	0	m_8
1	0	0	1	0	m_2
2	0	0	1	1	m_3
2	0	1	0	1	m_5
2	1	0	1	0	m_{10}
3	0	1	1	1	m_7
3	1	0	1	1	m_{11}
3	1	1	0	1	m_{13}
4	1	1	1	1	m_{15}

Tabelle 7.1: Minterme nach der Anzahl der nicht-invertierten Variablen sortiert.

Minterme	a	b	c	d
m_0, m_2	0	0	X	0
m_0, m_8	X	0	0	0
m_2, m_3	0	0	1	X
m_2, m_{10}	X	0	1	0
m_8, m_{10}	1	0	X	0
m_3, m_7	0	X	1	1
m_3, m_{11}	X	0	1	1
m_5, m_7	0	1	X	1
m_5, m_{13}	X	1	0	1
m_{10}, m_{11}	1	0	1	X
m_7, m_{15}	X	1	1	1
m_{11}, m_{15}	1	X	1	1
m_{13}, m_{15}	1	1	X	1
$m_{0,2}, m_{8,10}$	X	0	X	0
$m_{2,3}, m_{10,11}$	X	0	1	X
$m_{3,7}, m_{11,15}$	X	X	1	1
$m_{5,7}, m_{13,15}$	X	1	X	1

Tabelle 7.2: Zusammenfassung der Minterme durch die Anwendung des simplen Konsensus.

Eine minimale, die Minterme überdeckende Menge von Primimplikanten ergibt sich anhand der Tabelle in Abbildung 7.3. Für das angegebene Beispiel sind zwei minimale Überdeckungen denkbar:

$$f(a,b,c,d) = \begin{cases} \bar{b}\bar{d} + bd + \bar{b}c \\ \bar{b}d + bd + cd \end{cases}.$$

Prim-impl.	Minterm-index	0	2	3	5	7	8	10	11	13	15	
$\bar{b}\bar{d}$	0, 2, 8, 10											Kernprimimplikant (KPI) (Minterme 0,8)
$\bar{b}c$	2, 3, 10, 11											
cd	3, 7, 11, 15											
bd	5, 7, 13, 15											Kernprimimplikant (Minterme 5, 13)
	KPI											von KPI überdeckte Minterme

von einem einzigen Primimplikanten überdeckte Minterme
 nicht von Kernprimimplikanten überdeckte Minterme

Abbildung 7.3: Finden einer minimalen Überdeckung.

Mit dem oben angegebenen Algorithmus steht eine vollständige Lösung zur Generierung einer minimalen, zweistufigen⁴ Darstellung einer Logikfunktion zur Verfügung. Damit läßt sich ein PLA auf minimale Art und Weise realisieren. Es stellt sich allerdings die Frage, wie man zu einer mehrstufigen Lösung kommt, die gegebenenfalls weniger Gatter oder zumindest weniger Transistoren aufweist. Die Antwort liegt in einer Operation, die Faktorisierung genannt wird. Diese Operation entspricht einer mehrfachen Anwendung des Distributivgesetzes. Dies soll an einem kleinen Beispiel verdeutlicht werden: Gegeben ist die Funktion $f = ad + abc + abe$.

$$f = ad + abc + abe = a(d + bc + be) = a(d + b(c + e))$$

Die sich daraus ergebenden Schaltungen werden im Bild 7.4 gezeigt. Folgende Unterschiede ergeben sich zwischen den beiden Realisierungen: zum einen ist der Aufwand bei der mehrstufigen Realisierung deutlich geringer. Die Anzahl der Gatter ist zwar gleich; auf Transistorniveau ergibt sich jedoch ein Verhältnis von 30 zu 24 Transistoren, da bei der zweistufigen Realisierung drei Gatter mit drei Eingängen benötigt werden, während die mehrstufige Lösung mit Gattern auskommt, die nur zwei Eingänge aufweisen. Bei der Verzögerungszeit wiederum ist die zweistufige Lösung

⁴1. Stufe AND, 2. Stufe OR.

überlegen, da hier nur zwei Gatterlaufzeiten anzusetzen sind; im mehrstufigen Fall aber vier. Letztendlich muß bei diesen sich widersprechenden Zielfunktionen ein Kompromiß gefunden werden, der beide Kriterien ausreichend berücksichtigt.

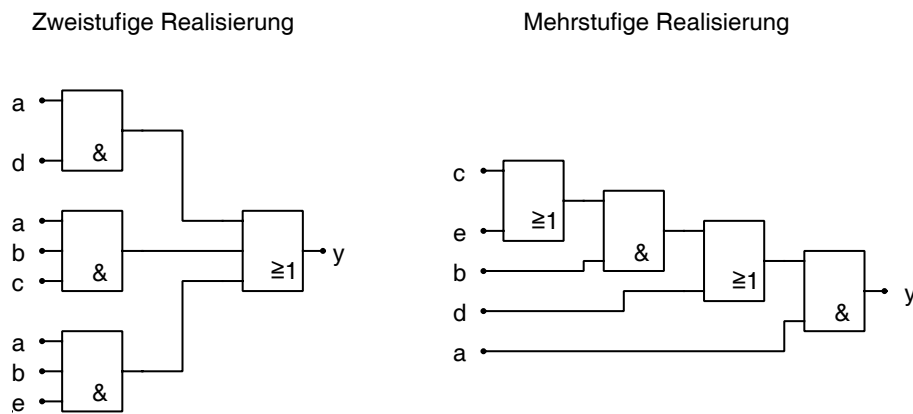


Abbildung 7.4: Zweistufige und mehrstufige Gatterrealisierung einer Funktion.

Die Faktorisierung kann man auch über die sogenannte Shannon-Zerlegung ableiten.

Definition 7.8 (Shannon-Zerlegung)

Sei f eine boole'sche Funktion und x eine beliebige Eingangsvariable von f .

$$(1) \quad f = (\bar{x} \cdot f_{\bar{x}=0}) + (x \cdot f_{x=1})$$

$$(2) \quad f = (\bar{x} + f_{\bar{x}=1}) \cdot (x + f_{x=0})$$

Mit der Shannon-Zerlegung nach der Variablen a wird die der oben gegebene Funktion $f = ad + abc + abe$ umgeformt. Zunächst ergeben sich folgende Ausdrücke für $f_{a=1}$ und $f_{a=0}$:

$$f_{a=1} = d + bc + be$$

$$f_{a=0} = 0$$

Damit ergibt sich für die gesamte Funktion:

$$f = \bar{a} \cdot f_{a=0} + a \cdot f_{a=1} = a(d + bc + be) + 0 = a(d + bc + be)$$

7.1.1 Abbildung von Logikfunktionen

Vielfach werden boole'sche Funktionen zur Realisierung mit Standardzellen oder Gate-Arrays auf die zugehörigen Zellbibliotheken abgebildet. In diesen Bibliotheken steht ein bestimmter Satz von Gattern zur Verfügung, um die betreffenden Logikfunktionen auszuführen. Bereits ein NAND-Gatter oder ein NOR-Gatter ist ausreichend, um alle denkbaren Funktionen abzudecken. In aller

Regel stehen jedoch auch sogenannte Komplexgatter zu Verfügung (z.B. ein AND-OR-INVERT- oder AOI-Gatter). Die Benutzung dieser Gatter führt zu einer besonders günstigen Umsetzung der Logikfunktion, da mit ihnen mehrere Grundfunktionen gleichzeitig abgearbeitet werden können. Von diesen Komplexgattern ist aber im Rahmen der Logikminimierung nicht die Rede. Dort werden in aller Regel nur die Grundfunktionen wie AND, OR oder NOT betrachtet. Das Ziel bei der Abbildung von Logikfunktionen liegt darin, die Stellen in einer Schaltung zu identifizieren, an denen einige Grundgatter durch ein günstigeres Komplexgatter ersetzt werden können. Das Auffinden solcher Bereiche wird beispielsweise durch die sogenannte Peephole-Technik bewerkstelligt. Mit der Peephole-Technik werden Teilschaltungen herausgeschnitten und bezüglich ihrer Wahrheitstabelle mit den in der Bibliothek vorhandenen Zellen verglichen. Die Einsetzung eines Komplexgatters kann zur Folge haben, daß eine andere Einsetzung scheitert, siehe Abbildung 7.5. Insgesamt sollte ein solcher Algorithmus auf Angaben über Laufzeit- und Flächenkosten für jeden Gattertyp basieren. Die gewünschte Lösung ist die, bei der die Gesamtkosten minimal werden.

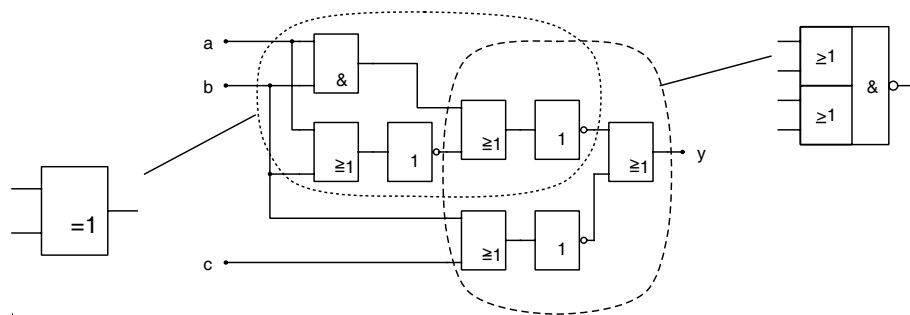


Abbildung 7.5: Zwei Peephole-Ersetzungen schließen sich aus.

Kapitel 8

Testfreundlicher Schaltungsentwurf

8.1 Grundbegriffe

Beim Test integrierter Schaltungen geht es in erster Linie darum, herauszufinden, ob die gefertigten Schaltungen auch wirklich funktionieren. Da man typischerweise mit einem Anteil fehlerhafter Chips zwischen 5 % und 50 % rechnen muß, ist die Prüfung der Funktionsfähigkeit von entscheidender Bedeutung. Wenn man erst auf Platinen- oder Systemebene anfängt, Fehler einzugrenzen und aufzudecken, dann steigt der benötigte Aufwand unverhältnismäßig stark an. Erschwerend kommt hinzu, daß ein vollständiger Test der Schaltung alle denkbaren Betriebszustände und -abläufe durchspielen muß. Dies verbietet sich aus Laufzeitgründen für größere Schaltungen. Selbst die unvollständigen Tests, die heutzutage angewendet werden, tragen bis zu 50 % zu den Gesamtkosten der Schaltung bei. Die einzige Chance zur Beherrschung der oben genannten Probleme besteht darin, die Testbarkeitsaspekte bereits während des Entwurfsablaufs zu berücksichtigen. Dies schränkt natürlich die Entwurfsfreiheit ein, ist aber unabdingbar wenn man überhaupt einen sinnvollen Test durchführen will.

Welche Ursachen haben die Fehlfunktionen einer gefertigten integrierten Schaltung? Typische Probleme ergeben sich aus unbeabsichtigten Kurzschlüssen und Unterbrechungen zwischen Leitungen, Abbildung 8.1. Ein anderes Problem liegt vor, wenn die Kontaktlöcher zum Aktivgebiet zu tief geätzt werden, so daß sich ein Kurzschluß zur Wanne oder zum Substrat, also mithin zu einer der Versorgungsspannungen ergibt.

An einem kleinen Beispiel, einem Uhren-IC, soll nun aufgezeigt werden, warum der Test integrierter Schaltungen so schwierig ist, Abbildung 8.2.

Der angeschlossene Quarz schwingt zum Beispiel mit 32 kHz. Die Anzahl der Taktzyklen, die pro Zeiteinheit verstreichen müssen werden in der Tabelle 8.3 dargestellt.

Das Uhren-IC entspricht also mit $4.0 \cdot 10^{12} \approx 2^{42}$ einem Zähler mit 42 Bit Länge. Wenn man einen Echtzeittest durchführen möchte, so muß man aus naheliegenden Gründen mindestens vier Jahre Testzeit investieren. Dies ist mit Sicherheit untragbar.

Im Laufe der letzten Jahre hat sich eine Vielzahl von Testmethoden herausgebildet, die anhand von Abbildung 8.4 eingeordnet werden. Diese Testmethoden lassen sich zunächst in zwei Kategorien einteilen: eingebaute Tests und externe Tests. Die eingebauten Tests werden mit der Hardware des zu testenden Systems durchgeführt, während die externen Tests zusätzliche Geräte benötigen.

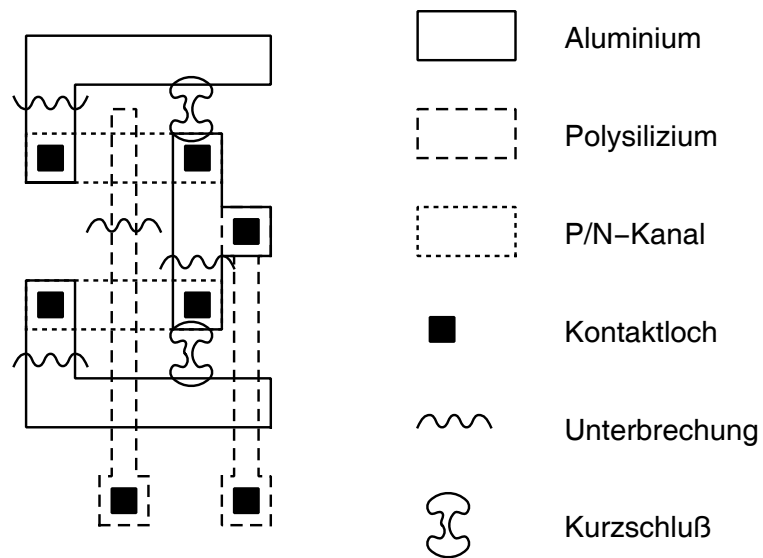


Abbildung 8.1: Potentielle Fehlermöglichkeiten an einem Inverter (Auswahl).

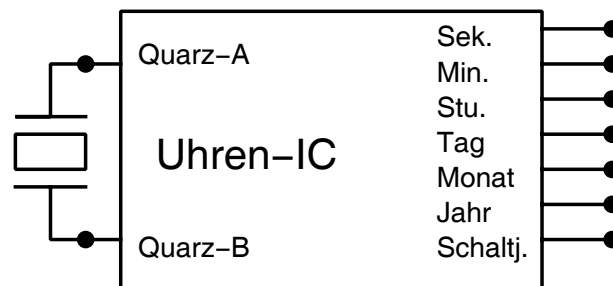


Abbildung 8.2: Testprobleme am Beispiel eines Uhren-IC.

Zeiteinheit	Anzahl Taktzyklen
1 Sekunde	32000
1 Minute	1920000
1 Stunde	$1.15 \cdot 10^8$
1 Tag	$2.76 \cdot 10^9$
1 Monat	$8.3 \cdot 10^{10}$
1 Jahr	$1.0 \cdot 10^{12}$
1 Schaltjahr	$4.0 \cdot 10^{12}$

Abbildung 8.3: Anzahl der Taktzyklen pro Zeiteinheit.

Dies sind zum Beispiel Automaten, die Testmuster an die Schaltung anlegen und die Resultate bewerten. Die eingebauten Tests lassen sich wiederum in zwei Gruppen einteilen: die Gruppe der beim normalen Betrieb mitlaufenden Tests und die Gruppe der in einem Testmodus ablaufenden Tests. Die erste Gruppe stützt sich auf Redundanzen in Daten oder Hardware. Mit Hilfe zusätzlicher Datenbits (z.B. Parity-Bit) läßt sich feststellen, ob die Daten konsistent sind. Bei redundanter Hardware, z.B. zwei Rechnern statt einem, werden Fehler dadurch erkannt, daß sich die Ergebnisse der Hardwareblöcke unterscheiden. Falls man besonderes Gewicht auf die Sicherheit der Ergebnisse legt, kann man die Redundanz auf die Spitze treiben, indem man drei oder fünf Blöcke dieselbe Arbeit tun läßt. Das Resultat ergibt sich jeweils aufgrund einer Mehrheitsentscheidung. Beispielsweise wurde der Bordcomputer des Space Shuttle dreifach ausgeführt, so daß die Ergebnisse anschließend verglichen und bewertet werden können. Falls ein eingebauter Test einen gesonderten Testmodus benötigt, so kann es sich dabei um einen Softwaretest oder einen Hardwaretest handeln. Softwaregesteuerte Selbsttests sind bei Mikroprozessoren gang und gäbe. Sie arbeiten eine vorher festgelegte Sequenz von Anweisungen ab. Wenn während dieses Ablaufs kein Fehler auftritt, ist die Wahrscheinlichkeit recht hoch, daß der Mikroprozessor in Ordnung ist. Diese Idee läßt sich auch in Hardware realisieren. Zu diesem Zweck muß dafür gesorgt werden, daß die Testhardware Prüfmuster generiert, diese an die zu testende Hardware anlegt und die Resultate mit Sollwerten vergleicht. Einer der bekanntesten Ansätze zu diesem Thema ist unter dem Namen BILBO (built-in logic block observation) bekannt. Bei den externen Test-Methoden sind der funktionale und der strukturierte Test zu unterscheiden. In beiden Fällen sollen Stimuli für die Schaltung erzeugt werden, die möglichst viele Fehler aufdecken. Die funktionalen Testmuster basieren auf der gewünschten Funktion der Schaltung. So kann man einen Mikroprozessor recht gut dadurch testen, daß man die Testmuster entsprechend der von ihm realisierten Operationen auswählt. Der strukturierte Test befaßt sich dagegen mit der Frage, welche Fehler überhaupt in der Schaltung vorkommen können und wie man die Teststimuli wählen muß, damit solche Fehler an den Ausgängen der Schaltung beobachtet werden können. Hier werden vor allem die für die Mikroelektronik wichtigen Bereiche des Tests dargestellt. Diese sind in der Abbildung 8.4 durch einen Stern gekennzeichnet.

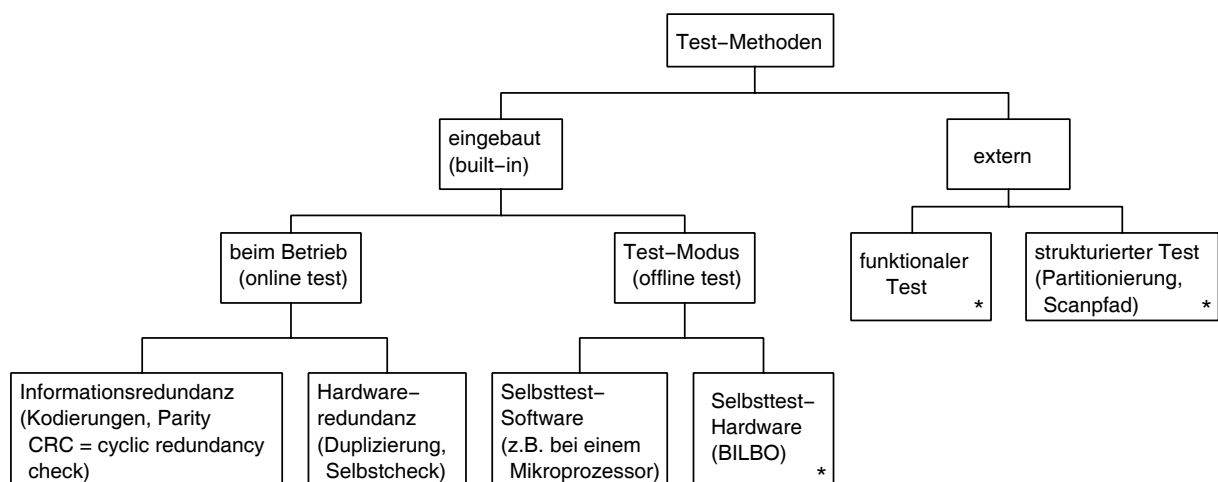


Abbildung 8.4: Testmethoden.

Die Fehlerursachen und -mechanismen sind ausgesprochen vielfältig. Um die Fehlerdetektion in den Griff zu bekommen, hat es sich als notwendig erwiesen, ein Modell für den Einfluß des Fehlers zu entwickeln. Insbesondere sollte dieses Modell für möglichst viele Fehlerursachen gültig sein, so daß man letztendlich die Technologie für die Generierung von Testmustern vernachlässigen und sich auf die Betrachtung des Modells beschränken kann.

Das mit Abstand wichtigste Fehlermodell ist das Haftfehlermodell (stuck at). Ein Fehler liegt in diesem Zusammenhang genau dann vor, wenn ein Netz der Schaltung permanent auf logisch 0 oder 1 liegt. Damit ist das Modell für den Fall gültig, daß ein Kurzschluß zu einer der Versorgungsleitungen vorliegt. Die meisten anderen Probleme führen ebenfalls dazu, daß ein Netz den Logikwert nicht mehr wechseln kann.

Um herauszufinden, ob ein Fehler gemäß des Modells an einem bestimmten Schaltungsknoten vorliegt, ist es notwendig, den Knoten auf den entgegengesetzten Wert einzustellen, und das im Fehlerfall differierende Ergebnis an den Schaltungsausgängen zu beobachten.

Definition 8.1 (Einstellbarkeit)

Ein Knoten einer Schaltung ist genau dann einstellbar, wenn es Testmuster gibt, die an die Eingänge der Schaltung angelegt bewirken, daß dieser Knoten auf 0 (für einen stuck-at-1-Fehler, s-a-1) und auf 1 (für einen stuck-at-0-Fehler, s-a-0) eingestellt wird.

Definition 8.2 (Beobachtbarkeit)

Die Beobachtbarkeit eines Knotens ist genau dann gegeben, wenn es Testmuster gibt, die an die Eingänge der Schaltung angelegt bewirken, daß das Vorhandensein eines Fehlers an diesem Knoten zu einem differierenden Logikwert an einem Ausgang führt.

Diese Definitionen werden anhand eines Beispiels verdeutlicht (Abbildung 8.5). Die primären Eingänge der Schaltung sind : a, b, c, d, e, f und g; die primären Ausgänge p und q. Die primären Eingänge sind definitionsgemäß ohne Probleme einstellbar; die primären Ausgänge beobachtbar. Wenn nun der Knoten x auf s-a-0 und s-a-1 Fehler untersucht werden soll, so muß man zunächst Werte an die Eingänge anlegen, die am Knoten x einen dem Fehler entgegengesetzten Wert einstellen. Dies wird anhand der Tabelle 8.6 dargestellt.

Falls man nun für eine Schaltung einen Satz von Testmustern zusammengestellt hat, stellt sich die Frage, wieviele Fehler tatsächlich erkannt werden. Diese Frage ist nicht einfach zu beantworten, da fast alle Testmuster nicht nur den Fehler aufdecken, für den sie entwickelt worden sind, sondern auch noch einige weitere Fehler. Warum das so ist, wird an dem folgenden Beispiel deutlich. Für die in der Abbildung 8.7 gezeigte Schaltung sei ein Test für den Knoten x s-a-0 entwickelt worden. Dieser Test kann auch dazu verwendet werden, den Knoten y auf einen s-a-0-Fehler zu prüfen. Dies liegt an folgender Tatsache: wenn der Knoten y konstant auf 0 liegt, dann auch aufgrund des AND-Gatters der Knoten x.

Die Fehlerüberdeckung ist ein Maß dafür, wieviele Fehler in Relation zur maximal möglichen Anzahl von Fehlern erkannt werden. Sie berechnet sich wie folgt:

$$\text{Fehlerueberdeckung (in Prozent)} = \left(\frac{\# \text{ beobachtbareFehler}}{\# \text{ denkbareFehler}} \cdot 100 \right) \quad (8.1)$$

Die Anzahl der denkbaren Fehler entspricht der Anzahl der Knoten mal zwei für s-a-0- und s-a-1 Fehler an jedem Knoten. Gewünscht werden Fehlerüberdeckungen zwischen 95 und 100 %. Dies

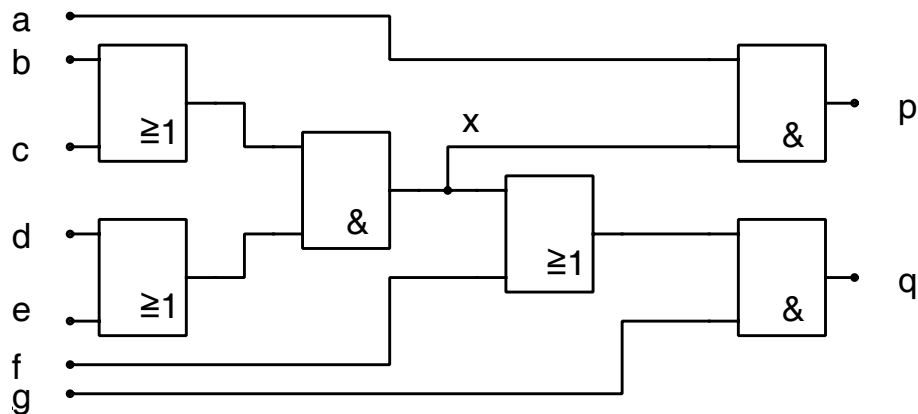


Abbildung 8.5: Beispielschaltung zur Erläuterung von Kontrollierbarkeit und Beobachtbarkeit.

ist jedoch nicht in allen Fällen zu bewerkstelligen. Ein Beispiel für einen nicht erkennbaren Fehler wird in Abbildung 8.8 gezeigt. Um den Knoten x auf 1 einzustellen muß der Knoten y den Wert 0 erhalten. Dies verhindert die Beobachtung des Fehlers x s-a-0, da der Pfad durch das zweite UND-Gatter nicht sensibilisiert werden kann. Die tiefere Ursache für dieses Problem liegt in einer Schaltungs-Redundanz. Durch geeignete Maßnahmen zur Optimierung der Schaltung, könnten diese Probleme automatisch entfernt werden.

8.2 Partitionierung und Scan-Techniken

Wie in vielen anderen Bereichen der CAD für Mikroelektronik kann man auch im Bereich des Tests oftmals nur dann zu akzeptablen Lösungen kommen, wenn man die Schaltung so unterteilt, daß sich die Einzelprobleme möglichst unabhängig voneinander lösen lassen. Dieser Vorgang wird auch die Partitionierung einer Schaltung genannt. Eine Möglichkeit der Partitionierung für das Beispiel des Uhren-IC besteht darin, den 42-Bit Zähler in drei 14-Bit Zähler zu zerlegen. Die Teilzähler sollten also in einem besonderen Testmodus unabhängig voneinander zu betreiben sein. Damit muß man für einen Test nicht mehr bis $2^{42} \approx 4 \cdot 10^{12}$ sondern nur noch dreimal bis $2^{14} \approx 16000$ zählen. Dies benötigt bei der im Beispiel vorgesehenen Taktrate eine Testzeit von dreimal einer halben Sekunde. Gegebenenfalls können die drei Zähldurchläufe auch gleichzeitig ablaufen. Dies ist ein gewaltiger Vorteil gegenüber der Testzeit von vier Jahren. Die Partitionierung erfordert hier allerdings etwas mehr Hardwareaufwand in Form zusätzlicher Pads und Gatter.

a	b	c	d	e	f	g	x (ok)	x (s-a-0)	x (s-a-1)	p (ok)	p (s-a-0)	p (s-a-1)	Test für
1	1		1				1	0	1	1	0	1	x (s-a-0)
1	0	0					0	0	1	0	0	1	x (s-a-1)

Abbildung 8.6: Einstellung und Beobachtung von s-a-0- und s-a-1-Fehlern am Knoten x.

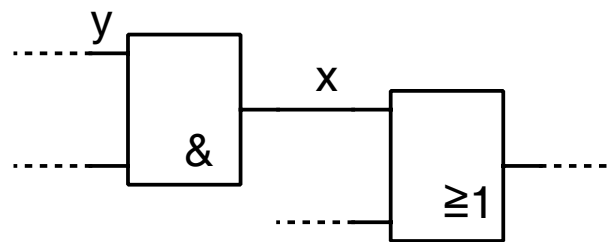


Abbildung 8.7: Beispielschaltung zur Erläuterung der Fehlerüberdeckung.

Die Partitionierung kann man allerdings durchaus noch weiter treiben. Eigentlich ist man ja nur daran interessiert, sicherzustellen, daß jedes Bit des Zählers einmal von 0 nach 1 und einmal von 1 nach 0 geschaltet wurde. Das LSB (least significant bit) des 14-Bit-Zählers wird jedoch viele Tausend mal auf 0 und 1 eingestellt, ohne daß man dadurch neue Informationen gewinnt. Dies wird an der Tabelle 8.9 deutlich. Die mit der Bemerkung 'bereits getestet' versehenen Zeilen können im Test übersprungen werden, falls man in der Lage ist, den Zähler auf beliebige Werte einzustellen. Dann benötigt man im Höchstfall nur noch zwei Taktzyklen pro Bit, oder 84 Taktzyklen ($\approx 2.5\text{ms}$) für den 42-Bit-Zähler.

Abschließend stellt sich die Frage, wie der Zähler geladen wird. Wenn dies parallel vor sich gehen soll, sind 42+1 Pins notwendig, Abbildung 8.10 a). Diesen Aufwand möchte man in der Regel nicht treiben, da auch die Gehäusegröße einen Kostenfaktor darstellt. Abhilfe kann dadurch geschaffen werden, daß man anstatt der Pads eine Kette von Flip-Flops einfügt, die als Schieberegister geschaltet sind. In dieses Schieberegister wird zunächst der gewünschte Zählerstand eingeschoben. Anschließend wird er parallel in die eigentlichen Zähler-Flip-Flops übernommen, Abbildung 8.10 b). Noch effizienter ist es allerdings, den Zähler in einem gesonderten Modus als Schieberegister zu betreiben (Scan-Pfad). Eine denkbare Schaltung ist in Abbildung 8.10 c) dargestellt. Im Falle des 42-Bit-Zählers werden jeweils 42 Taktzyklen zum Einschieben der Daten und ein Taktzyklus für den eigentlich zu prüfenden Übergang benötigt. Anschließend sind jeweils nochmal 42 Taktzyklen notwendig, um die Ergebnisse wieder herauszuschieben. An Hardware ist ein zusätzliches Pad notwendig, das die Umschaltung vom Schiebemodus in den Normalmodus steuert. Den Eingang zum Einschieben der Daten und den Ausgang zum Ausschieben der Daten lassen sich per Multiplexer ($\hat{=}$ And-Or-Invert-Gatter) auf bereits vorhandene Pads legen. Schließlich benötigt der Scan-Pfad einen Multiplexer für jedes Flip-Flop mit dem dessen Eingang zwischen Schieberegister-

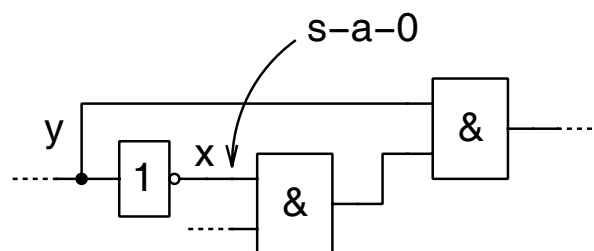


Abbildung 8.8: Nicht erkennbarer Fehler stuck-at-0 an Knoten x.

LSBs eines Zählers	Bemerkungen
...00000	
...00001	Bit 0: 0 → 1
...00010	Bit 1: 0 → 1, Bit 0: 1 → 0
...00011	bereits getestet
...00100	Bit 2: 0 → 1, Bit 1: 1 → 0
...00101	bereits getestet
...00110	bereits getestet
...00111	bereits getestet
...01000	Bit 3: 0 → 1, Bit 2: 1 → 0
...01001	bereits getestet
...01010	bereits getestet
...01011	bereits getestet
...01100	bereits getestet
...01101	bereits getestet
...01110	bereits getestet
...10000	Bit 4: 0 → 1, Bit 3: 1 → 0
...	

Abbildung 8.9: 1-0- und 0-1-Übergänge eines Zählers.

ster und Normalmodus umgeschaltet wird. Außerdem ist noch etwas zusätzliche Verdrahtung für das Schieberegister notwendig.

Mit Hilfe eines Scan-Pfades läßt sich also eine Schaltung auf einen beliebigen Zustand einstellen. Außerdem können per Scan-Pfad beliebige Zustände beobachtet werden, da man die Werte der Flip-Flops auch wieder herauschieben kann. Der große Vorteil dieser Methode liegt darin, daß sich durch den Scan-Pfad eine sehr gut geeignete Partitionierung ergibt. Im Prinzip wirkt der Scan-Pfad so, als würde man die Schaltung an den Flip-Flops aufschneiden. Jeder Flip-Flop-Eingang entspricht einem neuen Schaltungsausgang; jeder Flip-Flop-Ausgang einem neuen Schaltungseingang. Diese neuen Ein- und Ausgänge sind nun wie die primären Ein- und Ausgänge ebenfalls kontrollierbar bzw. beobachtbar. Damit sind die Schaltungsteile kleiner geworden, für die jeweils Testmuster generiert werden sollen. Außerdem sind diese Schaltungsteile grundsätzlich kombinatorisch, da ja alle speichernden Elemente in den Scan-Pfad aufgenommen wurden. Dies vereinfacht die algorithmische Seite der Testmustergenerierung wesentlich.

Das Scan-Pfad-Konzept läßt sich aber nicht nur auf den Test einer einzelnen integrierten Schaltung anwenden. Auch bei Test und Fehlersuche auf Platinen- oder Systemniveau ist der Scanpfad ausgesprochen nützlich. Unter dem Namen „Boundary-Scan“ ist eine Methodik bekannt geworden, die sich innerhalb kürzester Zeit durchgesetzt hat und mittlerweile auch genormt ist. Die Hauptidee besteht darin, die Pads einer integrierten Schaltung so zu modifizieren, daß man den Logikwert an diesen Pads nach außen und innen über ein Schieberegister einstellen kann. Insbesondere werden die Schieberegister der einzelnen Chips auf einer Platine geeignet miteinander verkoppelt, so daß man für die komplette Platine mit einem einzigen Scanpfad auskommt. Mit dem Boundary-Scan

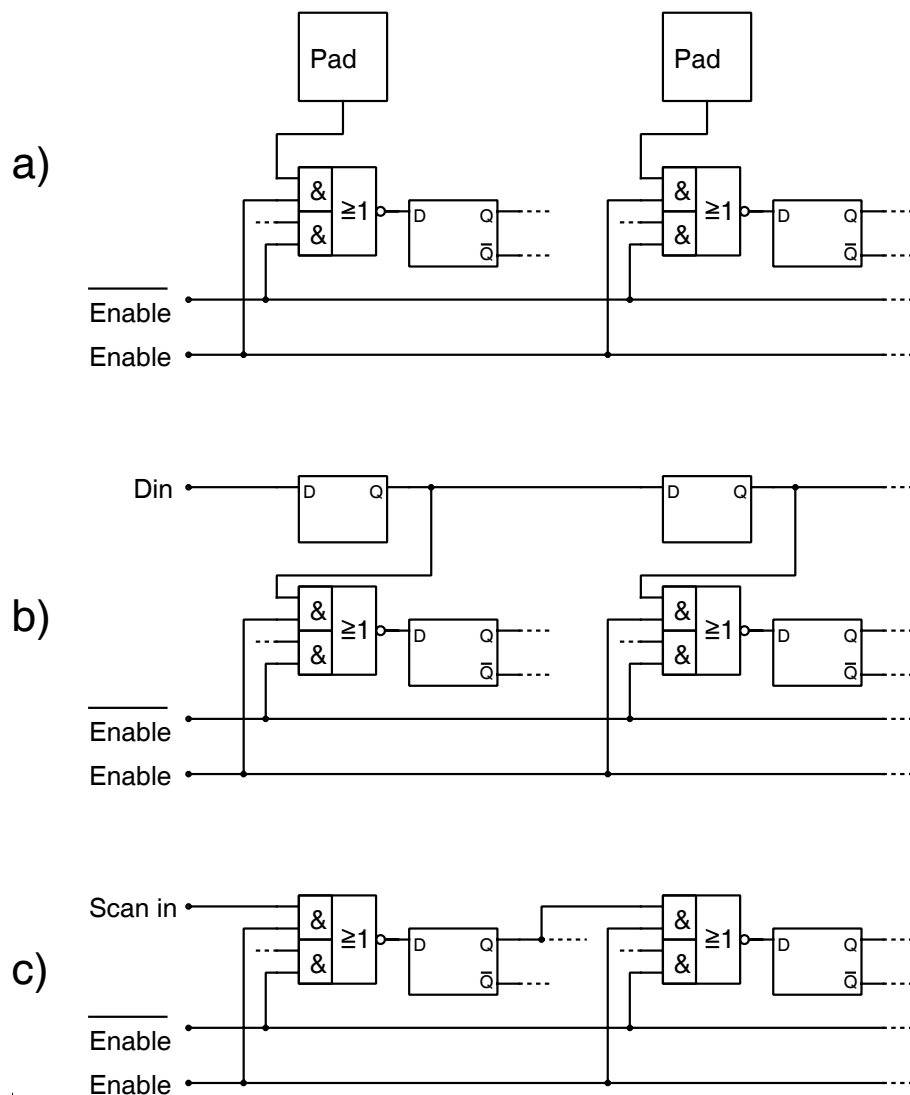


Abbildung 8.10: Paralleles Laden der Flipflops a), seriell Laden der Flipflops b), Einbau eines Scanpfades c).

können nun in eine Platine eingelötete Chips unabhängig voneinander untersucht werden. Auf dieselbe Art und Weise ist es auch möglich, die Verbindungen zwischen den Chips zu überprüfen.

8.3 Externer Test

Ein Test integrierter Schaltungen basiert in aller Regel darauf, daß man geeignete Logikwerte - die Stimuli - an die Eingänge der Schaltung anlegt und die Ausgaben der Schaltung mit Sollwerten vergleicht. Falls es hier Diskrepanzen gibt, wurde ein Fehlverhalten der gerade getesteten Schaltung aufgedeckt. In der Regel wird dieses Verfahren durch externe Testautomaten durchgeführt. Andere Verfahren beruhen darauf, daß die Testhardware in die Schaltung integriert wird. Diese Methoden werden seltener angewendet, da die Chipfläche einer der wesentlichen Kostenfaktoren

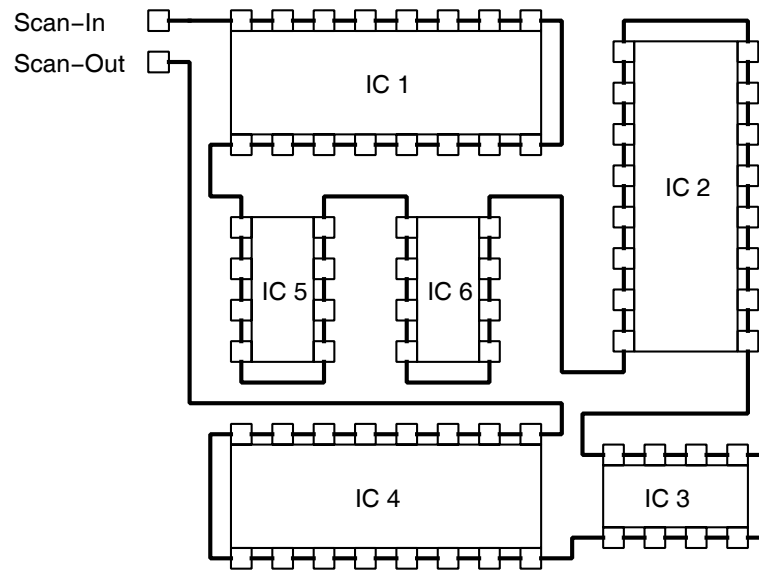


Abbildung 8.11: Boundary-Scan.

ist und die Testhardware auf jedem Chip unterzubringen ist. Die Abbildung 8.12 zeigt das Schema eines Testautomaten.

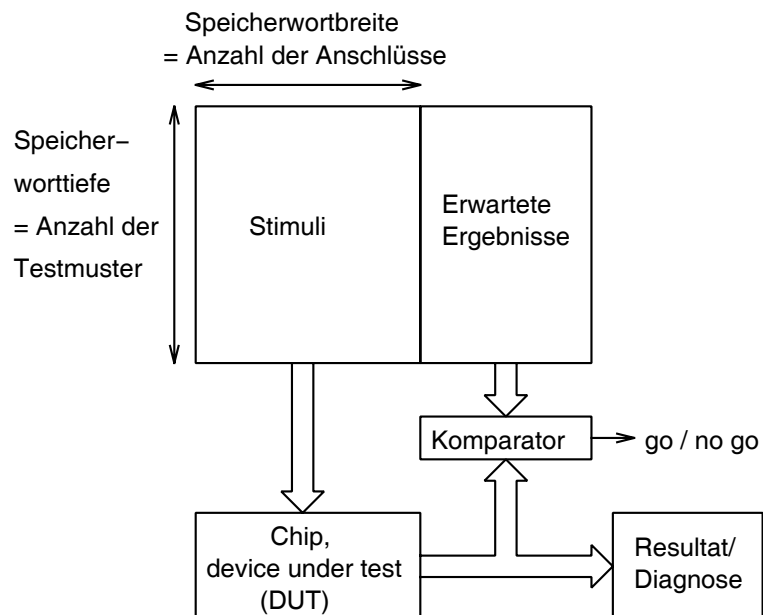


Abbildung 8.12: Schema eines Testautomaten.

8.4 Testmustergenerierung

Grundsätzlich lassen sich zwei Arten der Testmustergenerierung unterscheiden. Zum einen kann die Erzeugung von Testmustern an der Funktion der Schaltung orientiert sein. Die Stimuli sollen den Funktionsumfang der Schaltung möglichst vollständig durchspielen, um sicherzustellen, daß der getestete Chip funktioniert. Ein Vorteil dieser funktionalen Testmustergenerierung liegt darin, daß bei der Simulation der Schaltung bereits ähnliche Stimuli angefallen sind. Diese können nun als Grundlage für die Testmuster dienen. Problematisch ist allerdings, daß sich der vollständige funktionale Test genauso wie die vollständige Simulation aus Laufzeitgründen verbietet. Außerdem erfordert die Aufstellung der funktionalen Testmuster in der Regel noch manuelle Arbeit.

Neben der funktionalen Testmustergenerierung hat in den letzten Jahren die systematische Testmustergenerierung immer mehr an Gewicht gewonnen. Hier ist die Funktion der Schaltung eher nebensächlich. Im Vordergrund steht die Frage, welche Fehler überhaupt auftreten können. Für jeden dieser denkbaren Fehler wird ein spezieller Test entwickelt. In vielen Fällen deckt ein solcher Test noch eine Reihe weiterer Fehler auf, so daß es sich lohnt, aus der Gesamtmenge der Tests eine minimale Menge auszuwählen, die alle Fehler überdeckt. Diese Betrachtung des Testproblems erfordert allerdings ein geeignetes Fehlermodell, da nun nicht mehr geprüft wird, ob eine Schaltung mit einigen funktionalen Stimuli korrekt läuft, sondern ob bestimmte Fehler gemäß eines Fehlermodells in der Schaltung vorliegen. Das bereits in Abschnitt 8.1 auf Seite 78 erwähnte Haftfehlermodell soll hier verwendet werden. Um herauszufinden, ob ein Knoten innerhalb einer Schaltung auf einem festen Logikwert liegt, muß man zunächst dafür sorgen, daß dieser auf den entgegengesetzten Wert eingestellt wird. Anschließend sollten die Eingänge der Schaltung so gesetzt werden, daß der Unterschied zwischen der fehlerhaften und der korrekten Schaltung an einem Ausgang sichtbar wird. Letzteres bezeichnet man auch als die Sensibilisierung eines Pfades von dem betrachteten Knoten zu einem Ausgang. Eine solche Pfadsensibilisierung wird in Abbildung 8.13 dargestellt. Nur wenn die anderen Eingänge der Gatter wie in der Abbildung beschaltet werden, ist der Pfad von Netz p zu Netz q sensibilisiert.

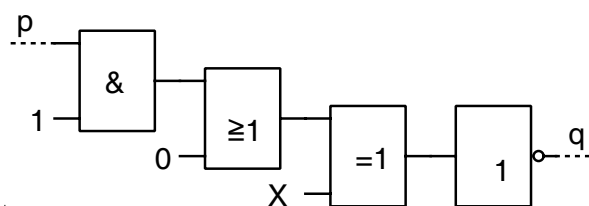


Abbildung 8.13: Pfadsensibilisierung.

Ein Algorithmus zur Generierung systematischer Testmuster benötigt in der Regel die folgenden Datenstrukturen: eine Entscheidungstabelle, eine Fehlerliste und eine Liste, die Implikationen aufnimmt. Die Entscheidungstabelle hält fest, welche Netze im Rahmen der Einstellung von Knoten und Sensibilisierung von Pfaden auf welche Werte eingestellt werden sollen. Insbesondere ist es wichtig, Widersprüche mit vorangegangenen Entscheidungen möglichst frühzeitig zu entdecken. Außerdem werden alternative Möglichkeiten gespeichert. Die Fehlerliste nimmt die bereits durch vorangegangene Tests abgedeckten Fehler auf. Die Liste der Implikationen schließlich ist dazu ge-

dacht, möglichst viele Informationen aus jeder Festlegung zu gewinnen. Falls beispielsweise gefordert wird, daß der Eingang eines AND-Gatters auf 0 liegt, dann ergibt sich aus der Implikation, daß der Ausgang des AND-Gatters ebenfalls auf 0 liegt. Die Implikationen dienen dazu, Widersprüche bei der Zuweisung von Logikwerten an die Netze möglichst früh aufzudecken. Der Ablauf der Pfadsensibilisierung und die Datenstrukturen sollen nun anhand eines Beispiels verdeutlicht werden. Ziel ist es, Testmuster für die in Abbildung 8.14 dargestellte Schaltung¹ zu generieren. Insbesondere soll ein Test für den Fehler a (s-a-1) gefunden werden.

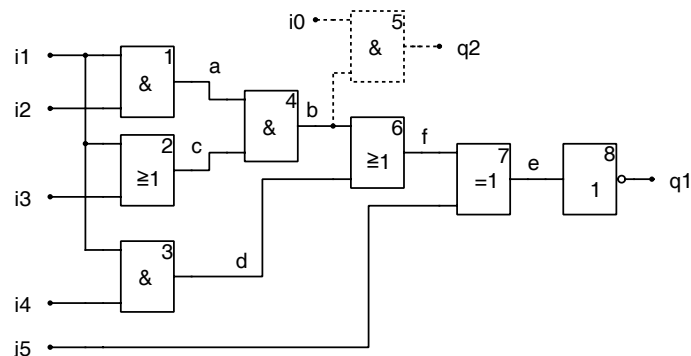


Abbildung 8.14: Beispiel für die Testmuster-generierung durch Pfadsensibilisierung.

Die Entscheidungstabelle für diesen Test wird in Tabelle 8.1 gezeigt. Mit den in der Entscheidungstabelle vorgenommenen Festlegungen ergibt sich folgendes Testmuster:

i1	i2	i3	i4	i5
0	X	1	X	0

Festlegung	Alternativen	Implikationen	Bemerkungen
i1 = 0	i2 = 0	d = 0	Einstellen von a auf 0
c = 1			Sensibilisierung des AND-Gatters 4
i1 = 1	i3 = 1		Erfüllung von c = 1, Widerspruch !
i3 = 1			Erfüllung von c = 1
d = 0			Sensibilisierung des OR-Gatters 6,
			Folgt aus der obigen Implikation
i5 = 0	i5 = 1		EXOR-Gatter 7 bei beiden Werten sensibilisiert

Tabelle 8.1: Entscheidungstabelle mit Implikationen.

Wie sich aus der in Abbildung 8.2 dargestellten Berechnung ergibt, findet das oben genannte Testmuster tatsächlich den gewünschten Fehler.

¹Der gestrichelte Teil soll zunächst nicht betrachtet werden.

Knoten	kein Fehler	Fehler
a	0	1
b	0	1
f	0	1
e	0	1
o1	1	0

Tabelle 8.2: Unterschied zwischen fehlerhafter und fehlerfreier Schaltung.

Die oben beschriebene Methode der Pfadsensibilisierung weist einen wesentlichen Nachteil auf: die Suche nach einem Ausgang, zu dem ein Pfad sensibilisiert werden soll, verläuft ungerichtet. So kann es passieren, daß ein günstigerer, kürzerer Pfad zu einem Ausgang übersehen wird. In Abbildung 8.14 wäre dies der gestrichelt gezeichnete Pfad zum Ausgang q2. Besser wäre eine Untersuchung aller Pfade zur selben Zeit. Im folgenden Abschnitt wird ein solcher Algorithmus, der D-Algorithmus, beschrieben.

8.4.1 D-Algorithmus

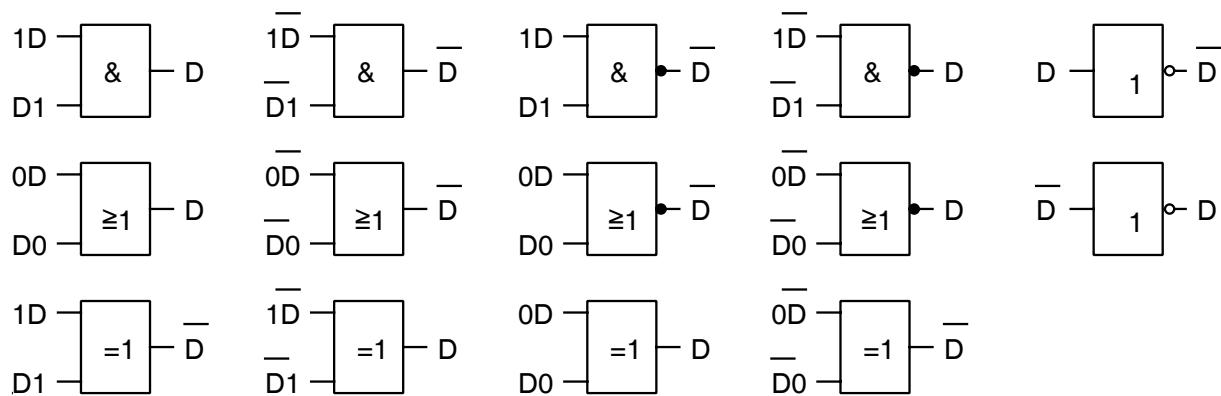
Der D-Algorithmus basiert auf der D-Notation², die die fehlerfreie und die fehlerbehaftete Schaltung gleichzeitig beschreibt. Das D entspricht hier einem besonderen Logikwert und steht für eine Diskrepanz zwischen der fehlerhaften und der fehlerfreien Schaltung. Damit können nun beide Schaltungen anhand einer Überlagerung gleichzeitig beschrieben werden.

fehlerfreie Maschine	0	0	1	1
fehlerbehaftete Maschine	0	1	0	1
D-Notation	0	$\bar{D}$	D	1

Die Aufgabe des D-Algorithmus besteht nun darin, den Logikwert D oder $\bar{D}$, der durch einen stuck-at-Fehler an dem fehlerhaften Knoten induziert wurde, zu einem beliebigen Ausgang zu propagieren. Diese Propagierung verläuft nach den in Abbildung 8.15 gezeigten Regeln.

Die Propagierung der $D/\bar{D}$ -Werte soll gleichzeitig über alle denkbaren Pfade geschehen. Zu diesem Zweck wird ein Vektor von Logikwerten betrachtet, der für jeden Knoten der Schaltung einen Eintrag aufweist, Tabelle 8.3. Zu Beginn stehen alle Logikwerte auf X. Anschließend werden zunächst die Festlegungen eingetragen, die notwendig sind, damit der betrachtete Knoten auf einen geeigneten Logikwert eingestellt werden kann. Von dem Fehlerknoten ausgehend werden dann „Wellenfronten“ betrachtet. In der ersten Wellenfront liegen die Knoten, die vom Fehlerknoten aus in einem Gatterdurchgang erreicht werden können. In der zweiten Wellenfront liegen die Knoten, die mit zwei Gatterdurchgängen erreicht werden, und so fort. Der Gatterdurchgang kann nur in Signalrichtung, d.h. von den Eingängen des Gatters zu den Ausgängen erfolgen. Sobald man einen

²J.P. Roth (1966).

Abbildung 8.15: Propagierung von $D/\bar{D}$ -Werten.

Knoten erreicht, der einen primären Ausgang bildet, ist die Suche beendet. Bei jedem Gatterdurchlauf werden die für die Sensibilisierung des Gatters notwendigen Einstellungen vorgenommen und ebenfalls in den Vektor eingetragen. Der Ablauf für die Beispielschaltung von Abbildung 8.14 wird in Tabelle 8.3 dargestellt.

i0	i1	i2	i3	i4	i5	a	b	c	d	e	f	q1	q2	Bemerkungen
X	X	X	X	X	X	$\bar{X}$	X	X	X	X	X	X	X	Anfangszustand
X	0	X	X	X	X	$\bar{D}$	X	X	X	X	X	X	X	Einstellung der Fehlerknotens
X	0	X	X	X	X	$\bar{D}$	$\bar{X}$	X	0	X	X	X	X	Implikation
X	0	X	X	X	X	$\bar{D}$	$\bar{D}$	1	0	X	X	X	X	Propagierung durch Gatter 4
X	0	X	1	X	X	$\bar{D}$	$\bar{D}$	1	0	X	$\bar{X}$	X	X	Einstellung $c=1$
1	0	X	1	X	X	$\bar{D}$	$\bar{D}$	1	0	X	$\bar{D}$	X	$\bar{D}$	Propagierung durch Gatter 5, 6, $\bar{D}$ am Ausgang q2

Tabelle 8.3: Anwendung des D-Algorithmus auf die Beispielschaltung.

8.4.2 Testmustergenerierung für sequentielle Schaltungen

Bislang wurden Methoden und Algorithmen zur Generierung von Testmustern für kombinatorische Schaltungen betrachtet. Bei sequentiellen Schaltungen ist das Problem ungleich schwerer. Damit ein Schaltungsknoten eingestellt werden kann, ist es oftmals notwendig, nicht nur die Eingänge, sondern auch die speichernden Elemente der Schaltung auf geeignete Werte zu setzen. Nun hängt aber der Wert eines Flip-Flop in der Regel von den Werten anderer Flip-Flops ab, die nun ihrerseits eingestellt werden müssen. Ähnliche Ketten von Abhängigkeiten ergeben sich auch bei der Beobachtung von Knoten. Hier muß man ebenfalls manchmal über eine ganze Anzahl weiterer Flip-Flops gehen, bis man den gewünschten Logikwert zu einem Ausgang transportiert hat.

Angenommen, man möchte den Ausgang des obersten Bits eines Zählers auf 1 einstellen. In der Regel kann man die Flip-Flops zurücksetzen, aber nicht setzen. Dann bleibt nichts anderes übrig, als den Zähler komplett durchzuzählen, bis man die gewünschte Einstellung vornehmen kann. Dies

kann unter Umständen sehr lange dauern. Ein anderes Problem tritt zutage, wenn sich umfangreiche Zyklen in der Schaltung befinden. Eine Gatterschaltung kann man aufgrund der Signalflußrichtung der Gatter als partiell gerichteten Graphen betrachten, d.h. den Kanten wird gegebenenfalls eine Richtung zugeordnet. Zyklen sind hier Sequenzen von Kanten, die der Kantenrichtung nicht entgegenlaufen und auf denselben Knoten zurückgeführt werden. Ein Zyklus sorgt dafür, daß Einstellungen für Flip-Flops oftmals mit den über den Zyklus zurücklaufenden Festlegungen kollidieren. In aller Regel liegen auch mehrere Zyklen ineinander, so daß sich ein komplexes Beziehungsgeflecht ergibt, das in manchen Fällen gar keine Lösung für die Einstellung oder Beobachtung von Knoten erlaubt. In anderen Fällen ist die Laufzeit zur Berechnung einer solchen Lösung untragbar.

Die sequentielle Testmustergenerierung ist zur Zeit ein Schwerpunkt der CAD-Forschung. Hier soll ein einfacher, auf der kombinatorischen Testmustergenerierung aufbauender Algorithmus dargestellt werden, der allerdings in seiner Effizienz zu wünschen übrig läßt. Der Algorithmus basiert auf einer iterativen, kombinatorischen Testmustergenerierung und ist nur für synchrone Schaltwerke einsetzbar. Zunächst wird die sequentielle Schaltung an den Flip-Flops aufgeschnitten. Flip-Flop-Eingänge werden zu Pseudo-Ausgängen, Flip-Flop-Ausgänge zu Pseudo-Eingängen (Abbildung 8.16). Damit liegt eine kombinatorische Schaltung vor. Wenn nun ein Test für einen bestimmten Knoten generiert werden soll, dann kann man dies zunächst auf Basis dieser Schaltung tun. Insbesondere wird darauf geachtet, möglichst wenig Pseudoeingänge und Pseudoausgänge zu benutzen. Für diese ergibt sich das Problem, daß sie in früheren Taktzyklen eingestellt (Pseudoeingänge) oder in späteren Taktzyklen beobachtet (Pseudoausgänge) werden müssen.

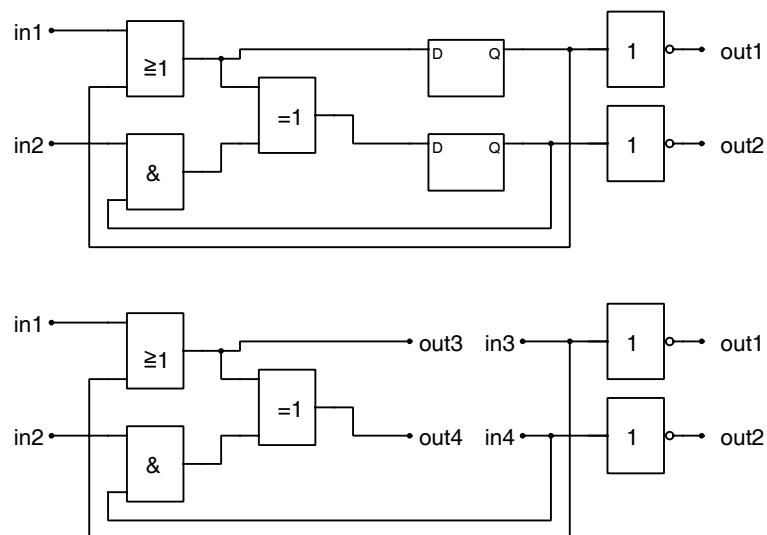


Abbildung 8.16: Aufschneiden einer Gatterschaltung an den Flip-Flops.

Zur Einstellung der Pseudoeingänge und Beobachtung der Pseudoausgänge steht die vorherige und die nachfolgende Taktphase zur Verfügung. Zum Einsatz kommt auch hier wieder der Algorithmus für die kombinatorische Testmustergenerierung. Bei diesen Einstellungen und Beobachtungen versucht man wiederum möglichst wenig Pseudoeingänge und -ausgänge zu benutzen, da ansonsten

eine weitere Taktphase vor oder nach den bisher benötigten Taktphasen erforderlich ist. Wenn keine weiteren Pseudoeingänge und -ausgänge benutzt werden, ist der Test für den aktuellen Fehler vollständig. Falls in der Schaltung umfangreichere Zyklen vorhanden sind, kann die Anzahl der Takte zum Test eines einzigen Fehlers erhebliche Werte annehmen. Trotz aller Bemühungen steckt die sequentielle Testmustergenerierung noch in den Kinderschuhen. In aller Regel behilft man sich damit, daß man entweder die sequentielle Schaltung durch einen Scanpfad in eine kombinatorische umwandelt oder ein spezielles Testkonzept entwickelt, das ganz auf die aktuelle Schaltung zugeschnitten ist.

8.5 Selbsttest

Wenn eine integrierte Schaltung sich selbst testen soll, muß man wie beim externen Test auch dafür sorgen, daß geeignete Testmuster zur Verfügung stehen und der Vergleich mit den Sollwerten gewährleistet ist.

8.5.1 Testmustergenerierung

Die einfachste, aber auch aufwendigste Methode die Testmuster bereitzustellen, ist ein ROM auf dem Chip. Im ROM werden Testmuster abgelegt und mit einem Zähler nach und nach adressiert. Mit dem ROM läßt sich eine beliebige Auswahl von Testmustern festhalten. In aller Regel wird jedoch wesentlich weniger Chipfläche benötigt, wenn man die Testmuster erst zur Testzeit generiert. Dies könnte beispielsweise durch einen Zähler bewerkstelligt werden. Ein Zähler liefert einen kompletten Satz von 2^n Testmustern bei n Variablen. Die Reihenfolge, in der die Zahlen gezählt werden, ist dabei allerdings unerheblich. Wichtig ist, daß alle Zahlen einmal auftreten, bevor es zur ersten Wiederholung kommt. Eine solche Schaltung kann mit Hilfe eines sogenannten „linear feedback shift register“ (LFSR³) realisiert werden. Ein LFSR ist gegenüber einem klassischen Zähler deutlich weniger aufwendig. Abbildung 8.17 zeigt ein Beispiel eines solchen LFSR. Das gezeigte LFSR erzeugt alle Testmuster außer (0 0 0).

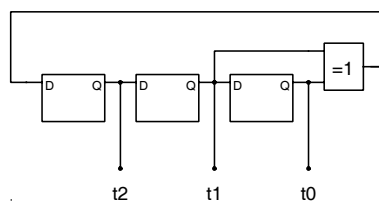


Abbildung 8.17: Testmustergenerierung durch ein LFSR.

³McCluskey (1981).

8.5.2 Bewertung der Schaltungsantworten

Bei der Bewertung der Schaltungsantworten steht man vor demselben Problem wie bei der Testmustersgenerierung. Auch hier müßte man eigentlich die erwarteten Antworten der Schaltung in einem ROM ablegen, um die Testantworten der Schaltung bewerten zu können. Dies verbietet sich jedoch wegen des damit verbundenen Aufwands. Eine fast gleichhohe Sicherheit erreicht man, indem man die Schaltungsantworten in geeigneter Weise komprimiert. Eine einfache Möglichkeit besteht darin, die Antworten der Schaltung zu addieren und gegebenenfalls die Überläufe zu ignorieren. Das Ergebnis reduziert sich zu einer einzigen Zahl, der Summe. Wenn die errechnete Summe gleich der erwarteten Summe ist, dann ist die Wahrscheinlichkeit groß, daß die Schaltung richtig funktioniert hat. Es kann jedoch vorkommen, daß sich Mehrfachfehler wieder aufheben. Auch in diesem Fall konnte der Hardwareaufwand durch den Übergang von einem klassischen Addierer zu einem etwas anders geschalteten LFSR verringert werden (Abbildung 8.18). In der Abbildung werden die zu komprimierenden Werte an den Eingängen i_0 bis i_2 angeliefert. Das Kompressionsergebnis liegt an den Ausgängen s_0 bis s_2 an. Ein solches LFSR wird auch Signaturregister genannt.

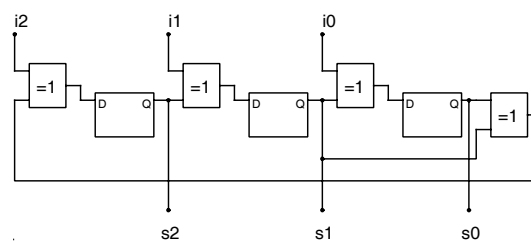


Abbildung 8.18: Signaturanalyse durch ein LFSR.

8.6 Fehlersimulation

Oftmals stellt sich für einen gegebenen Satz von Testmustern die Frage, wieviele Fehler tatsächlich mit ihnen erkannt werden können. Die Antwort darauf liefert eine Fehlersimulation. Zu diesem Zweck werden zunächst sämtliche Testmuster mit der Schaltung simuliert (Gutsimulation). Dies liefert Erwartungswerte für die an den Ausgängen auftretenden Ergebnisse. Anschließend fügt man einen Fehler (z.B. Knoten x, s-a-1) in die Schaltung ein und simuliert erneut alle Testmuster. Falls sich eine Diskrepanz zur Gutsimulation ergibt, kann der Fehler erkannt werden, und man geht zum nächsten Fehler über. Dieser wird in die Schaltung eingefügt, und die Simulation startet erneut. Wenn dies für alle denkbaren Fehler durchgeführt worden ist, kennt man den Prozentsatz der Fehler, die erkannt werden. Im Normalfall sollte er nicht unter 95 % liegen. Die Fehlersimulation gehört zu den aufwendigsten Aufgaben der CAD für Mikroelektronik. Um doch noch erträgliche Laufzeiten auch für VLSI-Schaltungen zu erreichen sind heutzutage Hardwarebeschleuniger auf dem Markt. Hierbei handelt es sich um Spezialcomputer, die dazu ausgelegt sind, Fehlersimulationen besonders schnell durchzuführen.

Auch algorithmisch sind einige Verbesserungen gegenüber dem Grundalgorithmus möglich. Die wichtigste ist die Parallelisierung des Ablaufs. Die Simulationen für die einzelnen Fehler können

unabhängig voneinander durchgeführt werden. Außerdem werden die typischen Operationen, die bei einer Logiksimulation Verwendung finden (AND, OR, NOT), im Rechner 32- oder 64-fach parallel ausgeführt. Solche Logikoperationen werden immer je nach Wortlänge bitweise für 4 oder 8 Bytes durchgeführt. Dies ermöglicht es, bis zu 64 Simulationen gleichzeitig durchzuführen. Dabei werden die Logikwerte an einem Schaltungsknoten für 3-wertige Logik - z.B. '0', '1', 'X' - durch 2 Worte dargestellt. Mit zwei Bits lassen sich vier verschiedene Logikwerte kodieren. Die beiden untersten Bits in den beiden Worten stehen beispielsweise für die Logikwerte der Gutsimulation. Die Bits Nummer 2 bis 32 stehen für die Logikwerte von 31 mit verschiedenen Fehlern behafteten Simulationen (Abbildung 8.19).

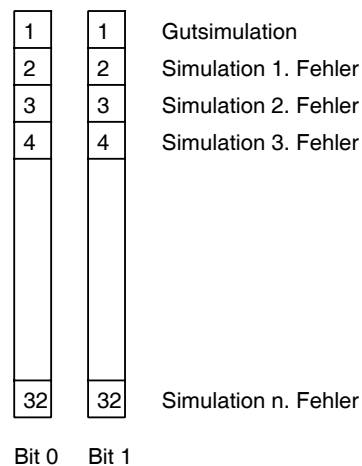


Abbildung 8.19: Kodierung der Logikwerte bei der parallelen Fehlersimulation.

Kapitel 9

Layout-Ebene

In der Frühzeit der Mikroelektronik - den sechziger Jahren - waren die Schaltungen noch ohne Schwierigkeiten zu überschauen. Layouts mit einigen zehn Transistoren konnten noch auf Papier oder Folie gezeichnet werden, obwohl dies schon für kleine Schaltungen recht mühsam war. Die rasante Entwicklung der Mikroelektronik wäre nicht möglich gewesen, wenn der Entwurfsablauf nicht durch die Rechnerunterstützung wesentlich vereinfacht worden wäre. Auch heute sind, wenn es nicht um sehr reguläre Schaltungen geht, die CAD-Werkzeuge der begrenzende Faktor für die Komplexität der Chips. Von der Technologie her sind heutzutage einige Milliarden Transistoren in einer Schaltung möglich. Aktuell (Herbst 1996) sind Labormuster eines 4-GBit DRAM Bausteins verfügbar, der mindestens soviele Transistoren wie Bits aufweist. Zur Zeit weisen die nicht so regulären Prozessorchips gerade erst einige Millionen Transistoren auf. Auf der untersten Abstraktionsebene, die mithin die größten Datenmengen enthält, kommen die Probleme mit den riesigen Datenmengen am stärksten zum Tragen. Dies soll durch eine Abschätzung für das Layout eines Inverters verdeutlicht werden, Abbildung 9.1. Man erkennt, daß für den Inverter mit zwei Transistoren im Layout ca. 20 Rechtecke benötigt werden. Es ergibt sich im Überschlag ein Verhältnis von etwa 10 Rechtecken auf einen Transistor.

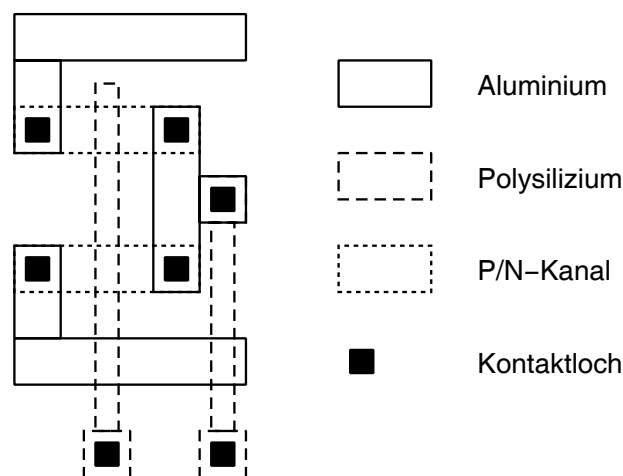


Abbildung 9.1: Layout eines Inverters.

Um ein Rechteck in der Datenbasis zu repräsentieren, benötigt man mindestens 20 Bytes. Diese verteilen sich wie folgt:

Entwurfsebene	2 Bytes
x_1, y_1	8 Bytes
x_2, y_2	8 Bytes
Attribute	2 Bytes
Summe	20 Bytes

Die Repräsentation des Layouts einer Schaltung mit einer Million Transistoren benötigt nach dieser Rechnung 200 Megabytes, falls die Hierarchie der Schaltung nicht ausgenutzt wird. Solche Datenmengen sind insgesamt nur schwer zu verarbeiten und insbesondere kaum noch im Hauptspeicher einer Workstation unterzubringen. Wesentlich sind also hier intelligente Algorithmen, um der Datenflut Herr zu werden.

9.1 Hardware eines Entwurfssystems

Für den Entwurf mikroelektronischer Schaltungen werden leistungsfähige Rechner - in aller Regel Workstations - benötigt, damit man einen Schaltungsentwurf darstellen, verändern, überprüfen und gegebenenfalls auch automatisch generieren kann. Typische CAD-Workstations weisen heute einen Durchsatz von 20-100 Mips, d.h. Millionen Instruktionen pro Sekunde, und einen Hauptspeicherausbau von 32 bis 256 MByte auf. Einige weitere Hardwarekomponenten sind spezifisch für CAD-Anwendungen:

- Ein Graphikbildschirm, d.h. üblicherweise ein 19"-Farbbildschirm mit einer Auflösung von mindestens 1280×1024 Bildpunkten.
- Ein Graphikkoprozessor bildet die Schnittstelle zwischen Rechner und Bildschirm. Er hat die Aufgabe, den Rechner weitgehend von der Graphikdarstellung zu entlasten.
- Die Eingabe graphischer Grundelemente geschieht üblicherweise mit zwei verbreiteten Gerätetypen. Zum einen gibt es die Maus mit einer Auflösung, die vergleichbar mit der des Bildschirms ist. Mittlerweile etwas aus der Mode gekommen sind Graphik-Tablets mit ca. 200×200 auflösbaren Punkten, auf denen man mit einer Art Stift arbeitet.
- Zur graphischen Ausgabe werden in der Regel Plotter verwendet, die in der Lage sind, ganze Chips in einem Format DIN A0 oder DIN A1 darzustellen. Falls man sich nur einen groben Überblick von einer Schaltung verschaffen möchte oder nur ein kleiner Teil einer Schaltung dargestellt werden soll, ist es möglich, den aktuellen Inhalt des Graphikbildschirms als „Hardcopy“ auszudrucken. Die Auflösung kann dabei naturgemäß nicht besser sein als die des Bildschirms.

9.2 Der Layouteditor

9.2.1 Datenhaltung

Von zentraler Bedeutung für ein graphisches CAD-Entwurfssystem ist die Datenbasis. Der Aufbau dieser Datenbasis ist wesentlich für die Leistungsfähigkeit des gesamten Systems und wird daher von keinem Systemanbieter offengelegt. Ein wichtiger Aspekt bei der Datenbasisdefinition ist die Implementierung einer schnellen Suche nach einzelnen Graphikelementen mittels eines komplexen Zeigersystems. So soll für lokale Änderungen an einer Schaltung nicht jeweils die Gesamtschaltung betrachtet werden müssen. Bei den Elementen der Datenbasis unterscheidet man zwischen den graphischen Grundelementen, die direkt in Maskendaten umgesetzt werden können, und den hierarchischen Elementen, die zum Aufruf von Unterzellen innerhalb der aktuell bearbeiteten Zelle dienen. Die Unterzellen bestehen ihrerseits wiederum aus graphischen und weiteren hierarchischen Elementen. Dabei ist in manchen Systemen die Verschachtelungstiefe begrenzt. Außerdem sind rekursive Aufrufe in aller Regel verboten. Auf diese Weise läßt sich sehr einfach eine hierarchische Struktur aufbauen, wie sie im zweiten Kapitel eingeführt worden ist.

Die einzelnen Layoutzellen - auch Strukturen genannt - werden in einer sogenannten Bibliothek abgelegt, die zur Verwaltung, Versionierung und Zugriffskontrolle bei mehreren Schaltungsentwicklern dient. So sollte beispielsweise eine Situation vermieden werden, in der zwei Schaltungsentwickler gleichzeitig Änderungen an einer Zelle anbringen. Immer wieder benutzte Zellen werden in sogenannten Referenzbibliotheken abgelegt, die für die Benutzer üblicherweise nur zum lesenden Zugriff freigegeben sind. So könnte man beispielsweise einen Katalog von Standardzellen in einer Referenzbibliothek ablegen.

9.2.2 Graphische Datenelemente

Die Abbildungen 9.2 und 9.3 enthalten die wichtigsten Graphik-Elemente, die in modernen Entwurfssystemen zur Verfügung stehen. Die Bezeichnungen und Parameter orientieren sich an den Namen, die innerhalb des Calma-GDSII Systems verwendet werden. Das Calma-GDSII System wird zwar nicht mehr häufig eingesetzt, ist aber einer der Urväter aller CAD-Systeme. Sie sind jedoch in ähnlicher Form in allen graphischen Entwurfssystemen enthalten. Der Parameter LAYER bezeichnet die Namen der Entwurfsebenen der Elemente, während der Parameter DATATYPE eine weitere Unterteilung innerhalb einer Entwurfsebene ermöglicht. Die Koordinaten werden bei allen Systemen im 32-Bit-Integer Format abgespeichert. Wenn man für die kleinste auflösbare Einheit einen Nanometer wählt, dann liegt die maximal darstellbare Chipgröße immer noch bei ca. vier Metern und damit weit oberhalb der heutigen technologischen Möglichkeiten.

9.2.3 Editiermöglichkeiten

Ein Layouteditor dient zum manuellen Erstellen und Korrigieren von Layouts. Außerdem werden die Ausgaben der Layout-Prüfprogramme typischerweise in die graphische Datenbasis eines Layouteditors eingetragen, so daß sie zur Korrektur etwaiger Fehler mit dem graphischen Editor zur Verfügung stehen.

Graphische Grundelemente			
Bezeichnung	Funktion	Parameter	Daten
BOUNDARY	geschlossener Polygonzug zur Begrenzung einer Fläche	LAYER DATATYPE	Koordinaten der Eckpunkte
PATH	Definition einer Leitung mit konstanter Breite	LAYER DATATYPE WIDTH PATHTYPE	Koordinaten der Eckpunkte der Mittellinie
TEXT	Erzeugung von Text innerhalb einer Maske (Kommentar, Maskenbeschriftung)	LAYER DATATYPE WIDTH ANGLE MAGNIFCATION JUSTIFICATION STRING	Koordinate des Referenzpunktes

Abbildung 9.2: Layout-Grundelemente

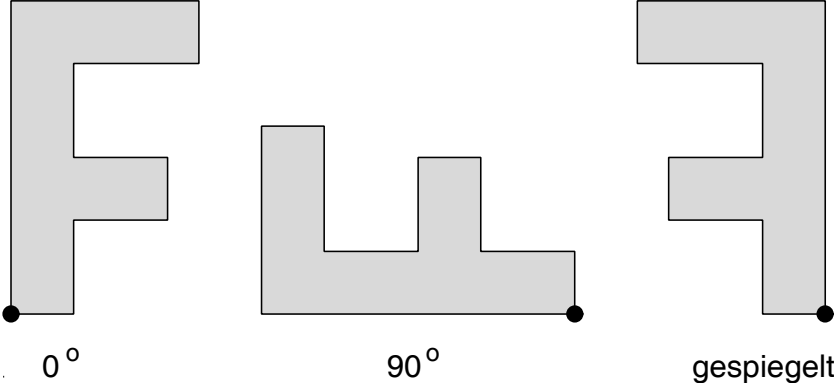
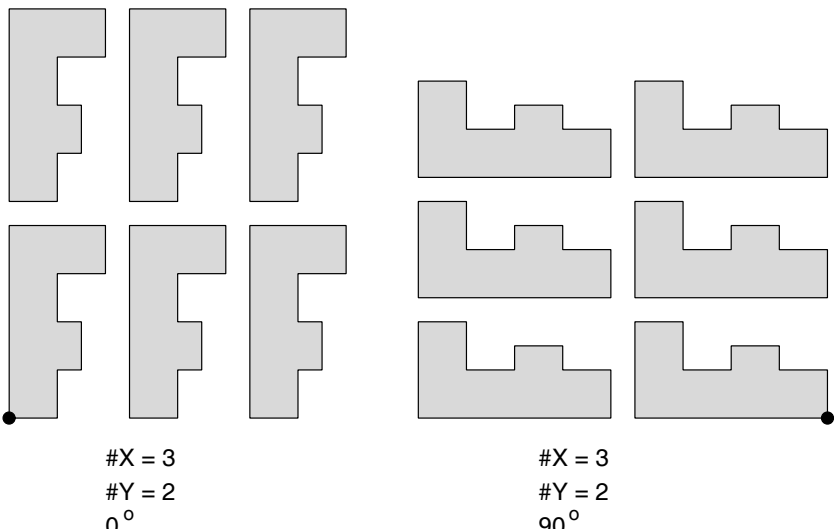
Hierarchische Datenelemente			
Bezeichnung	Funktion	Parameter	Daten
SREF	Aufruf einer Struktur (Structure-Reference)	STRUCTURE (Name) ANGLE REFLECTION MAGNIFICATION	Koordinate des Aufrufpunktes
	 0° 90° gespiegelt		
AREF	Aufruf einer Struktur mit matrizenartiger Wiederholung (Array- Reference)	STRUCTURE ANGLE MAGNIFICATION REFLECTION X/YREPEAT X/YSPACING	Koordinate des Aufrufpunktes
	 #X = 3 #X = 3 #Y = 2 #Y = 2 0° 90°		

Abbildung 9.3: Hierarchische Layout-Elemente

Der Editor arbeitet grundsätzlich innerhalb einer vorgewählten Struktur. Dabei sind dann alle Elemente innerhalb dieser Struktur erreichbar, während die Elemente der Unterstrukturen normalerweise nicht ohne weiteres manipuliert werden können. Moderne Systeme bieten darüber hinaus auch diese Möglichkeit, wobei man jedoch berücksichtigen muß, daß die Änderungen an allen Aufrufen der Struktur zum Tragen kommen.

Die eigentlichen Editieroperationen können auf einzelne Elemente oder auf ganze Elementgruppen angewandt werden. Bevor jedoch eine Manipulation möglich ist, muß das Element oder die Elemente aus der Gesamtheit aller Elemente ausgewählt werden. Diese Selektion geschieht über die Eingabe von Koordinaten in der Nähe des gewünschten Elements oder durch die Definition eines Fensters, wobei alle Objekte innerhalb des Fensters ausgewählt werden. Da in der Ebenendarstellung einer integrierten Schaltung möglicherweise mehrere Elemente am selben Punkt existieren, gibt es viele Methoden zur Beschränkung der Auswahlmöglichkeiten. Neben den naheliegenden Möglichkeiten, wie der Selektierbarkeit verschiedener Entwurfsebenen oder Elementtypen, können in manchen Systemen auch Elemente, die zu einer bestimmten Signalleitung gehören, selektiert werden. Dies erfordert allerdings, daß zusätzlich zur geometrischen Information auch die elektrische Zugehörigkeit gespeichert ist. Diese Art der Layoutdarstellung wird auch symbolisches Layout genannt und ist Gegenstand eines späteren Abschnitts. Neben der Selektierbarkeit von Objekten ist auch ihre Sichtbarkeit nach ähnlichen Kriterien einzustellen. Zu den wichtigsten Manipulationsmöglichkeiten zählen Operationen wie:

- Kopieren, Verschieben und Löschen von Elementen
- Drehen und Spiegeln von Elementen
- Vergrößern oder Verkleinern von Elementen
- Einfügen, Verschieben und Löschen von Punkten oder Segmenten graphischer Grundelemente

Neben diesen Operationen, die auf die Koordinaten der Elemente wirken, lassen sich natürlich auch sämtliche Parameter der selektierten Elemente verändern. Außer den reinen Editierfunktionen gibt es noch eine Vielzahl von Betriebsparametern des graphischen Editors. Folgende Eigenschaften lassen sich über Parameter einstellen:

- Darstellung der einzelnen Masken (Farbe, Schraffierung)
- Auflösungstiefe der Darstellung der Unterstrukturen
- Betriebsarten zur Koordinateneingabe (z.B. nur orthogonal)
- Koordinatenraster (sichtbares und erreichbares)
- Sichtbarkeit der Entwurfsebenen und Elementtypen
- Selektierbarkeit der Entwurfsebenen und Elementtypen

Bei einer geschickten Einstellung dieser Parameter läßt sich die Effektivität der Arbeit mit dem Editor erheblich steigern. Da offensichtlich sehr viele Parameter eingestellt werden können, und außerdem bestimmte Befehlssequenzen häufig wiederholt werden müssen, bieten fast alle graphischen Entwurfssysteme die Möglichkeit, komplexe Editorfunktionen als Makroaufrufe zu definieren.

9.3 Layoutrepräsentation und Grundalgorithmen

In diesem Abschnitt werden die grundlegenden Datenstrukturen eingeführt, die für Algorithmen zur automatischen Manipulation und Kontrolle von Layoutdaten Verwendung finden. Diese unterscheiden sich naturgemäß erheblich von den im Abschnitt über den Layouteditor vorgestellten Eingabemöglichkeiten. Sie richten sich vielmehr nach den spezifischen Anforderungen der zugehörigen Algorithmen. Layouts lassen sich durch Polygone, Bitraster, Kanten und Trapeze repräsentieren. Die ersten drei Möglichkeiten werden im folgenden beschrieben. Die Trapezdarstellung wird im wesentlichen im Rahmen kommerzieller Pakete eingesetzt, so daß hier zu wenig Informationen vorliegen, um eine sinnvolle Beschreibung zu geben.

9.3.1 Polygone

Ein Polygon ist eine Sequenz von Koordinatenpunkten in der Ebene, die in sich selbst zurückgeführt wird, d.h. der erste und der letzte Punkt fallen aufeinander. Einander folgende Polygonpunkte werden durch gerade Streckenabschnitte verbunden. Sie umschließen Fläche mit einer festgelegten Orientierung, z.B. gegen den Uhrzeigersinn. Das Beispieldiagramm von Abbildung 9.4 enthält 13 Punkte, wobei der erste und der letzte (1, 13) aufeinanderfallen. Das Loch in der Fläche wird durch eine entsprechende Einstülpung (Punkte 8 - 11) dargestellt. Hierbei ergibt sich eine Konstruktionslinie¹ auf der die Strecken (7 → 8) und (11 → 12) in entgegengesetzter Richtung aufeinander verlaufen. Damit fallen auch die Punkte 7 und 12 aufeinander.

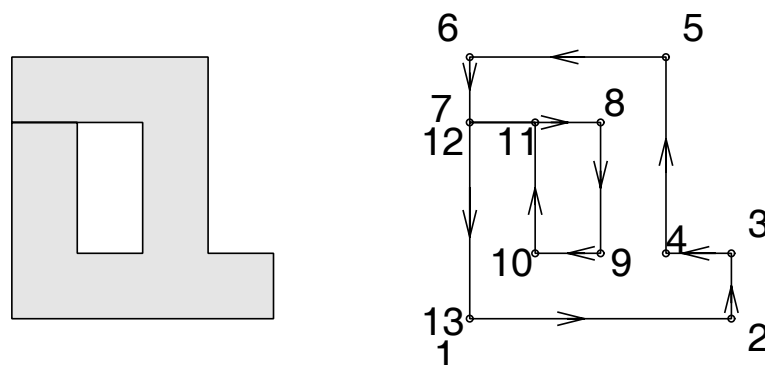


Abbildung 9.4: Flächendarstellung durch Polygone.

¹Eine Konstruktionslinie ergibt sich auch im linken Bild, da das Graphiksystem, mit dem die Zeichnungen erstellt wurden, polygonorientiert ist.

Polygone können sich selbst oder andere Polygone überlappen. Die Abbildung 9.5 zeigt links eine Selbstüberlappung und rechts zwei Polygone, die sich gegenseitig überlappen.

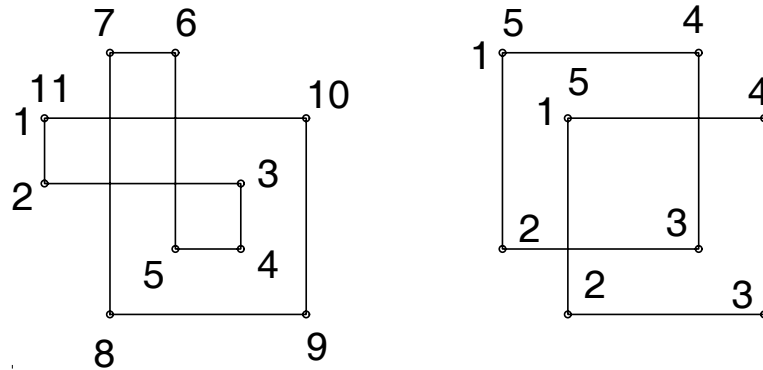


Abbildung 9.5: Überlappende Polygondarstellung.

Vielfach wird bei den Graphikoperationen eine überlappungsfreie Flächendarstellung gefordert, die zunächst durch Verschmelzung überlappender Polygone hergestellt werden muß. Die resultierenden Polygone bilden dann jeweils eine maximale Umschreibung der spezifizierten Fläche. Abbildung 9.6 zeigt die Polygone von Abbildung 9.5 in verschmolzener Form.

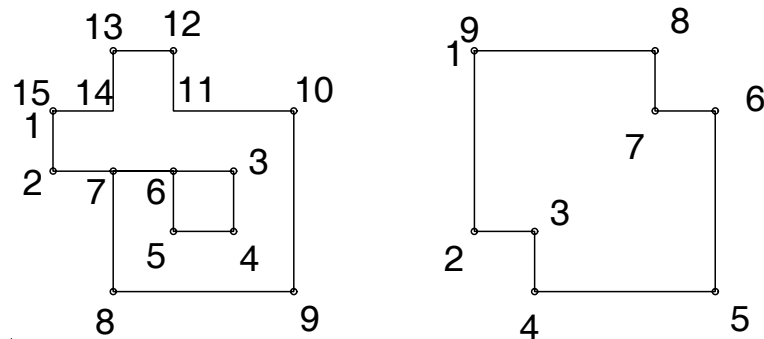


Abbildung 9.6: Verschmolzene Polygondarstellung.

Falls die Polygone nicht in verschmolzenem Zustand betrachtet werden, ergeben sich massive Probleme beispielsweise bei der Kontrolle von geometrischen Entwurfsregeln z.B. für die Prüfung der Mindestbreite. Durch das „Aneinandersetzen“ von Polygonen können Verletzungen der Mindestbreite aufgehoben oder induziert werden (Abbildung 9.7).

Die Verschmelzung führt in manchen Fällen zu extrem großen und unübersichtlichen Polygonen. So entstehen bei großen Schaltungen für Versorgungs- oder Taktleitungen Polygone mit einigen zehntausend Punkten. Diese sind kaum noch sinnvoll als Ganzes zu verarbeiten. Aus diesem Grund wird die Flächenbeschreibung durch Polygone in modernen Werkzeugen praktisch nicht mehr eingesetzt.

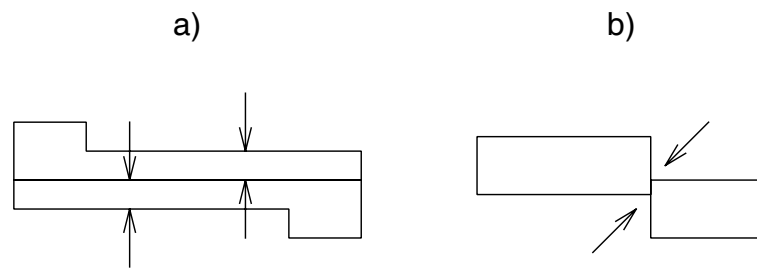


Abbildung 9.7: Probleme bei der Mindestbreitenprüfung für nicht verschmolzene Polygone, a) Vortäuschung einer Verletzung, b) Vortäuschung der Verletzungsfreiheit.

9.3.2 Bitraster

Eine weitere Flächenbeschreibung ist die Bitrasterdarstellung. Hier werden quadratische „Kacheln“ (tile) einer festen Kantenlänge mit jeweils einem Bit versehen, das anzeigt, ob diese Kachel zur beschriebenen Fläche gehört oder nicht. Ein wichtiger Vorteil liegt hier darin, daß eine überlappende Darstellung erst gar nicht möglich ist, so daß damit hier auch keine Probleme auftreten. Die Bitrasterbeschreibung kann allerdings nur für sehr grobe Auflösungen verwendet werden, da ansonsten der Speicherbedarf überproportional ansteigt. So vervierfacht sich die Anzahl der Kacheln bei halbiertem Mindestschrittlänge. Schrägen können nur durch eine Treppe approximiert werden, so daß sich an dieser Stelle in jedem Falle Ungenauigkeiten ergeben. Schließlich kann eine schräge Kante kaum noch als Einheit betrachtet und verarbeitet werden, so daß ein Teil der denkbaren Algorithmen von vornherein ausscheidet. Abbildung 9.8 zeigt die Rasterdarstellung einer einfachen Fläche mit einem Loch. Aufgrund der oben geschilderten Probleme wird die Rasterdarstellung nur noch für Aufgaben eingesetzt, wo eine Pixeldarstellung in der Ausgabe gefordert ist, wie etwa bei der Datenaufbereitung für den Plotter.

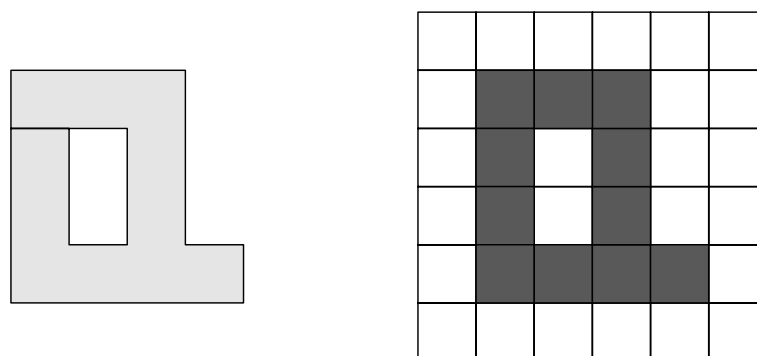


Abbildung 9.8: Flächendarstellung durch Bitraster.

9.3.3 Kanten

Die Kantenbeschreibung ist weit verbreitet und dient vielfach als grundlegende Datenstruktur für Algorithmen zur Flächenbearbeitung. Eine Kante ist hierbei ein gerichteter Streckenabschnitt. Sie enthält folgende Komponenten:

- zwei Koordinatenpaare, die Endpunkte des Streckenabschnitts
- ein Feld mit Flags (u.a. das Orientierungsflag)
- eine Polygonnummer, die die Zugehörigkeit einer Kante zu einem Polygon anzeigt

Anhand der Orientierung² einer Kante läßt sich entscheiden, auf welcher Seite der Kante sich die Fläche befindet. In Richtung der Kante gesehen, befindet sich die Fläche hier immer auf ihrer linken Seite. Die Kantendarstellung ist zwar weniger kompakt, da jede Koordinate zweimal abgelegt werden muß; sie ermöglicht jedoch ein lokales Arbeiten ohne daß der gesamte Polygonkontext mitgeführt werden muß. Außerdem besteht die Möglichkeit, alle waagerechten (oder senkrechten) Kanten wegzulassen, da diese lediglich redundante Information enthalten, die sich im Rahmen der Flächenverarbeitung leicht rekonstruieren läßt. Abbildung 9.9 illustriert die Kantendarstellung. Die Kantendarstellung ist zur Zeit die gängigste im Rahmen der Layoutdatenverarbeitung.

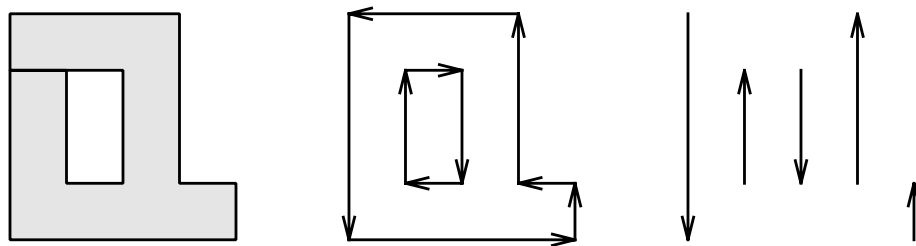


Abbildung 9.9: Flächendarstellung durch Kanten (mit und ohne senkrechte Kanten).

9.3.4 Plane-Sweep-Algorithmus

Der wohl am häufigsten für Geometrieoperationen eingesetzte Algorithmus ist der Plane-Sweep-Algorithmus. Im folgenden wird dieser - auch Scanline-Algorithmus genannte - Ansatz für die Kantendarstellung beschrieben; grundsätzlich ist er jedoch für praktisch alle denkbaren Flächenbeschreibungen formulierbar. Seine wesentliche Idee besteht darin, aus dem zweidimensionalen Problem der Bearbeitung von Fläche ein eindimensionales, nämlich die Bearbeitung auf einer Schnittlinie - der Scanline - zu machen. Eine waagerechte Scanline wird dabei ereignisgetrieben von unten nach oben über die Fläche geführt, wobei sich jeweils immer nur die Kanten im Hauptspeicher befinden, die von der Scanline geschnitten oder zumindest berührt werden. Dies sorgt zunächst dafür, daß immer nur ein kleiner Teil der Kanten im Hauptspeicher gehalten werden muß.

²Die Orientierung einer Kante wird durch die Pfeilspitze angedeutet.

Bei den heutigen Komplexitäten der einzelnen Ebenen einer VLSI-Schaltung ist dies ein wichtiger Punkt, der im wesentlichen durch die lokale Natur der meisten Flächenoperationen motiviert wird. Der Scanline ist eine Datenstruktur zugeordnet, die die aktuell geschnittenen Kanten enthält. Auf der Scanline selbst ist die Abarbeitungsrichtung ebenfalls festgelegt: von links nach rechts. Somit ist die Reihenfolge der Ereignisse (Ein- und Ausfügen einer Kante bezüglich der Scanline-Datenstruktur) festgelegt. Kanten, die schon von der Scanline geschnitten wurden, aber bereits wieder aus ihr gelöscht wurden, liegen in der „Vergangenheit“. Die aktuell geschnittenen Segmente liegen in der „Gegenwart“ und die später geschnittenen in der „Zukunft“. Anhand der vorgenannten Ereignisse wird dann jeweils die eigentliche Flächenoperation beschrieben. Abbildung 9.10 illustriert den Plane-Sweep-Ablauf.

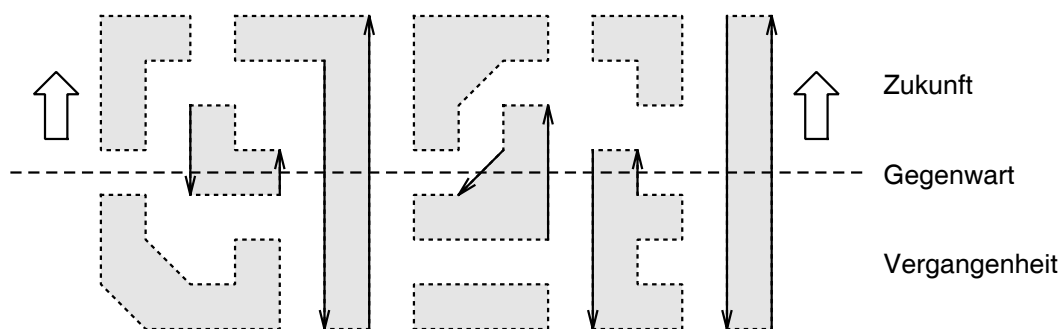


Abbildung 9.10: Plane-Sweep-Ablauf.

Ein wesentlicher Punkt bei der Implementierung des Plane-Sweep-Algorithmus ist das Schieben der Scanline über die Fläche. Dies wird durch die Abarbeitung einer Sequenz von Ereignissen bewerkstelligt. Dabei wird jedes Einfügen und Löschen einer Kante in die Scanline-Datenstruktur als Ereignis betrachtet. Jedem Ereignis ist ein ganz bestimmter Zeitpunkt relativ zu den anderen Ereignissen zugeordnet. Dieser ergibt sich für das Einfügen der Kante aus den Koordinaten ihres unteren Punktes und für das Löschen aus der Lage ihres oberen Punktes. Die zeitliche Sequenz kann also auf eine räumliche Sequenz abgebildet werden, die wiederum mit einer Sortierung der Kanten nach ihrem unteren Punkt bewerkstelligt wird. Das Sortierkriterium für das Einfügen von Kanten im Plane-Sweep-Durchlauf besteht aus drei Teilen, und ist in unserem Fall wie folgt definiert:

1. aufsteigende y-Koordinate des unteren Punktes der Kante
2. aufsteigende x-Koordinate desselben Punkts
3. Winkel des Segments

Damit erfolgt das Einfügen in die Scanline in der Reihenfolge der sortierten Kanten. Die Scanline selbst befindet sich immer an der y-Koordinate, die der gerade eingefügten Kante entspricht. Mit jeder neuen y-Koordinate der Scanline muß geprüft werden, ob Kanten aus der Scanline-Datenstruktur gelöscht werden müssen, weil sie von der Scanline nicht mehr berührt oder geschnitten werden.

Die Scanline wird oftmals als höhenbalancierter (Binär-)Baum implementiert, woraus sich selbst im ungünstigsten Fall ein recht guter Wert für die Abschätzung der Laufzeit ergibt. Diese Abschätzung wird mit den Mitteln der Komplexitätstheorie durchgeführt. Insgesamt ergibt sich eine Komplexität von $O(n \log n)$ (n = Anzahl der Kanten) für einen Durchlauf, da jede Kante genau einmal in den Baum eingefügt und aus dem Baum gelöscht werden muß. Beide Operationen erfordern eine Laufzeit proportional zu $O(\log n)$, so daß sich insgesamt $O(n \log n)$ ergibt. Ansonsten schlägt noch die Sortierung der Kanten zu Buche, die aber ebenfalls mit $O(n \log n)$ abgeschätzt werden kann. Die Laufzeitkomplexität liegt damit insgesamt bei $O(n \log n)$, d.h. nur unwesentlich über einer linearen Laufzeit. Dies ist ein wesentlicher Grund dafür, daß sich der Plane-Sweep-Algorithmus durchgesetzt hat, da selbst große Datenmengen effizient abgearbeitet werden können. Außerdem ergibt sich eine Hauptspeicherkomplexität von $O(\sqrt{n})$, d.h. zu jedem Zeitpunkt muß nur ein Anteil von Kanten im Hauptspeicher gehalten werden, der in der Größenordnung mit der Wurzel der Gesamtanzahl der Kanten wächst. Dies bedeutet, daß der Hauptspeicherausbau typischer Workstations praktisch immer ausreicht, um einen Scanline-Durchlauf zu ermöglichen.

9.4 Prüfung geometrischer Entwurfsregeln

Die Kontrolle geometrischer Entwurfsregeln (design rule check, DRC) ist seit langem ein Muß beim Handlayout integrierter Schaltungen. Sie kann als die Überprüfung einer definierten Syntax der Schaltung betrachtet werden. Bei der Prüfung geometrischer Entwurfsregeln kamen in den letzten Jahren vor allem zwei Faktoren zum Tragen: zum einen ist die Komplexität der Layouts stark angestiegen. Zum anderen sind die Entwurfsregeln und Randbedingungen immer komplizierter geworden. Zu den einfachen Standards wie Leitungsbreite und -abstand sind im Laufe der Zeit viele spezielle Regeln hinzugekommen, die auch Ausbeute und Zuverlässigkeitsaspekte berücksichtigen. Außerdem kommen im Rahmen der Strukturverkleinerung technologische Effekte zur Geltung, die zusätzliche Regeln induzieren.

9.4.1 Geometrische Entwurfsregeln

An der Schnittstelle zwischen CAD und der Fertigung hat es sich als notwendig erwiesen, die geometrische Struktur der Bauelemente und Leitungen einigen grundlegenden Regeln zu unterwerfen. Damit soll sichergestellt werden, daß die Entwürfe mit befriedigender Ausbeute in der betreffenden Technologie gefertigt werden können. Die grundlegenden Regeln sollen für die Entwurfsebenen Mindestabstände, Mindestbreiten oder Mindestüberhänge sicherstellen. Insgesamt sind Regelsätze von 100 bis 200 Regeln dabei keine Seltenheit.

Man unterscheidet folgende Ursachen für graphische Entwurfsregeln:

- physikalische Eigenschaften von Bauelementen
- parasitäre Einflüsse
- Materialeigenschaften, z.B. spezifischer Widerstand
- apparative Ungenauigkeiten bei der Photolithographie und Maskenherstellung

Die einzelnen Effekte ziehen Entwurfsregeln nach sich, die sicherstellen sollen, daß der betreffende Effekt die Schaltung nicht negativ beeinflusst.

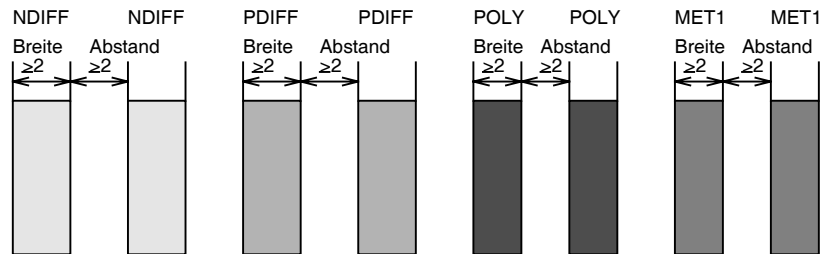
- Punch-through, Lawinenmultiplikation, Schwellspannungsreduktion
⇒ Minimale Kanallänge der MOS-Transistoren
- Latch-up
⇒ Abstand von P-/N-Gebieten zueinander
⇒ Substrat- und Wannenkontaktierung
- Abbildungsgenauigkeit (abhängig von der Belichtungstechnik und vom Photolack) sowie Beugung und Brechung des Lichts an der Maske oder der nicht planaren Chipoberfläche
⇒ Mindestbreiten und Mindestabstände innerhalb einer Entwurfsebene
- Justiergenauigkeit der Masken auf den Wafer
⇒ Mindestabstände und -überhänge zwischen verschiedenen Maskenebenen
- Übertragungsgenauigkeit von den Strukturen auf der Maske zu den Strukturen auf dem Chip. Diese wird beeinträchtigt durch die Photolackentwicklung und das Ätzverfahren.
⇒ Mindestbreiten und -abstände innerhalb einer Ebene
⇒ Dimensionierung von Kontaktlöchern.

Der folgende, vereinfachte Satz von Entwurfsregeln, siehe Abbildung 9.11, ist charakteristisch für solche Regelsätze. Die Werte der Entwurfsregeln werden in λ angegeben. Ein λ entspricht der kleinsten Längeneinheit und könnte beispielsweise einen Mikrometer betragen. Eine Skalierung der Technologie ist ohne weiteres durch eine Skalierung des λ -Wertes erreichbar.

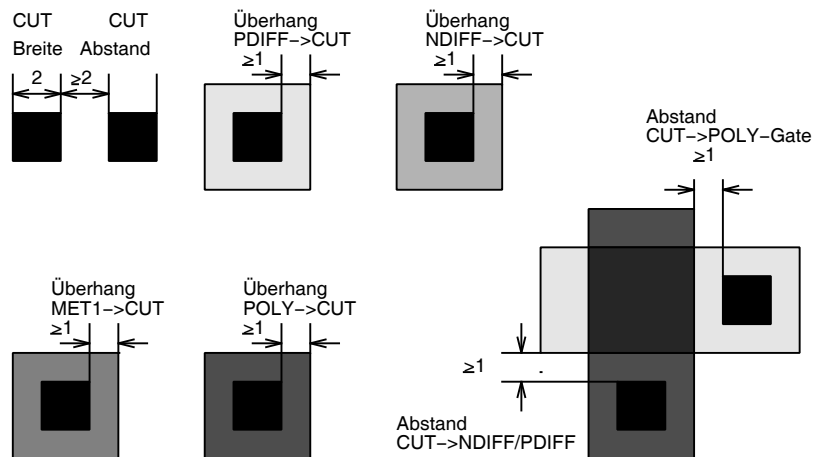
9.4.2 Übersicht über die Grundoperationen

Üblicherweise setzt sich ein System zur Entwurfsregelkontrolle aus diversen Operatoren zusammen, die jeweils eine Grundfunktion repräsentieren. Zur Aufbereitung der Daten stehen Boole'sche Verknüpfungen zwischen Maskenebenen, Sizingoperationen, topologische Operationen und diverse Selektionsmechanismen zur Verfügung. Mit diesen Modulen werden aus den Layoutebenen zusätzliche Flächeninformationen gewonnen. Dazu gehören zum Beispiel die vom Schaltungsentwickler nicht explizit definierten Gates, die sich aus dem Flächenschnitt von Aktivgebiet und Polysilizium ergeben. Für die Entwurfsregelkontrolle werden die aufbereiteten Flächen mit Hilfe eines Meßmoduls auf Designregelverletzungen getestet.

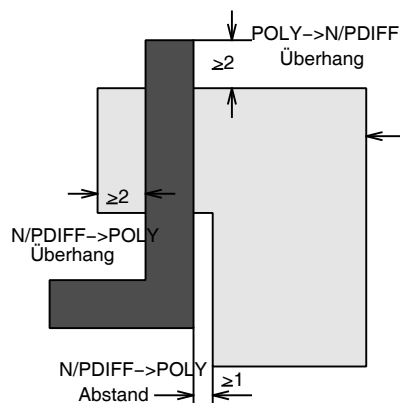
Bei der Beschreibung der im folgenden aufgeführten Flächenoperationen werden im wesentlichen die Begriffe „Fläche“ und „zusammenhängende Teilfläche“ verwendet. Diese Bezeichnungen wurden gewählt, um die Beschreibungen der Operationen unabhängig von der rechnerinternen Repräsentation durch Polygone, Kanten usw. zu machen. Im einzelnen bezeichnet der Begriff „Fläche“ die einer Entwurfsebene zugeordnete Überdeckung in der Ebene, also die Gesamtüberdeckung aller Polygone dieser Entwurfsebene. Der Begriff „zusammenhängende Teilfläche“ steht für ein einzelnes Polygon, wobei man grundsätzlich von „verschmolzenen“ Polygonen ausgeht, die somit eine maximale Umschreibung der Teilfläche bilden.

Mindestbreiten und Mindestabstände in λ 

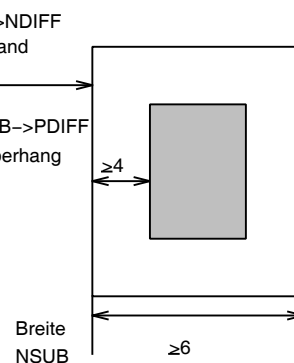
Kontaktlochregeln (CUT)



Transistor Regeln



Wanne (NSUB)

Abbildung 9.11: Typische Entwurfsregeln; die einzelnen Werte sind in λ gegeben.

9.4.3 Boole'sche Flächenoperationen

Die wichtigsten und am häufigsten eingesetzten Flächenmanipulationen sind die Boole'schen Flächenoperationen. Aus zwei Flächen A, B ergibt sich eine dritte C nach den folgenden Definitionen. Anders als beim rechnerinternen Koordinatengitter wird die Ebene zur Formalisierung der einzelnen Operationen als Kollektion von beliebig fein verteilten Rasterpunkten betrachtet. Sei f_A die charakteristische Funktion der Fläche A, wobei für alle kontinuierlichen Punkte (x,y) in der Ebene gilt:

Definition 9.1

$$f_A(x,y) := \begin{cases} 1 & \text{falls } (x,y) \in A \\ 0 & \text{sonst} \end{cases}$$

Die Operationen AND, OR, MINUS und EXOR definieren sich folgendermaßen.

Definition 9.2

$$\begin{array}{lll} C = AND(A,B) & : \iff & f_C = f_A \wedge f_B \\ C = OR(A,B) & : \iff & f_C = f_A \vee f_B \\ C = MINUS(A,B) & : \iff & f_C = f_A \wedge \neg f_B \\ C = EXOR(A,B) & : \iff & f_C = (f_A \wedge \neg f_B) \vee (\neg f_A \wedge f_B) \end{array}$$

Zusätzlich zur formalen Umschreibung sollen die Boole'sche Flächenoperationen durch eine graphische Darstellung (Abbildung 9.12) erläutert werden.

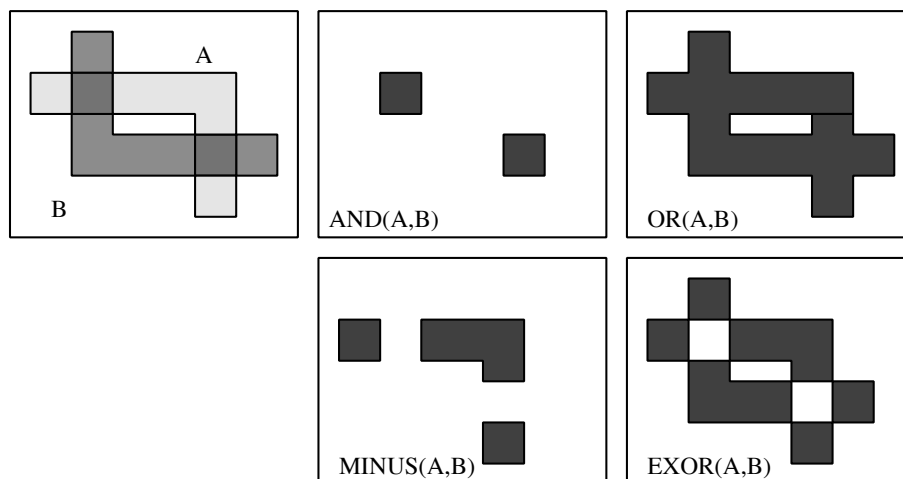


Abbildung 9.12: Ergebnis der Boole'schen Operationen.

Mit der oben eingeführten formalisierten Darstellung läßt sich auch die Inversion einer Fläche beschreiben. Hierbei sollte allerdings zur praktische Handhabung der betrachtete Flächenbereich - also die Grundmenge - beschränkt sein. Dies ergibt sich bei der Anwendung am Rechner automatisch durch den begrenzten Zahlenbereich.

Definition 9.3 (Flächeninversion)

$$C = \neg A \quad : \Longleftrightarrow \quad f_C = \neg f_A$$

Genau wie alle Boole'schen Funktionen durch die mehrfache Anwendung der NAND-Funktion darstellbar sind, könnte man alle Boole'schen Flächenoperationen durch ein Flächen-AND und eine Flächeninversion erreichen. Dies würde jedoch die benötigten Laufzeiten erheblich vergrößern. Zu den Boole'schen Operationen kann auch die CLEAN-Operation gerechnet werden, die einer Veroderung einer Fläche mit sich selbst entspricht und im wesentlichen dazu eingesetzt wird, überlappende Polygone miteinander zu verschmelzen, um so eine maximale Flächenumschreibung zu erhalten.

Die definierten Boole'schen Flächenoperationen sind für die Datenaufbereitung unentbehrlich und werden vielfach eingesetzt. Beispielsweise wird die Überlappungsfläche zwischen Diffusion und Polysilizium - also die Transistorgates - durch eine Boole'sche AND-Operation erzeugt. Source- und Draingebiete werden gebildet, indem die Gatefläche von der Diffusionsfläche abgezogen wird. Um bei der Überprüfung der Gates die erlaubten Gateüberstände mit zu berücksichtigen, wird eine weitere Klasse von Operationen benötigt, die metrischen Operationen.

9.4.4 Metrische Operationen

Die metrischen Operationen³ dienen dazu, die Fläche einer Ebene je nach Vorzeichen des zugehörigen Werts zu expandieren oder kontrahieren. Es handelt sich hierbei jedoch *nicht* um eine einfache Skalierung, d.h. um die Multiplikation aller Koordinatenwerte mit einem bestimmten Faktor. Vielmehr wird bezüglich der bestehenden Fläche eine Schicht auf- oder abgetragen. Zunächst soll eine Variante der charakteristischen Funktion definiert werden. Der euklidische Abstand zweier Punkte in der Ebene sei gegeben durch die Funktion d :

Definition 9.4

$$f_{ds(A)}(x,y) \quad := \quad \begin{cases} 1 & \text{falls } \exists (x',y') \in A, d((x,y),(x',y')) \leq s \\ 0 & \text{sonst} \end{cases}$$

Somit definiert sich das Sizing um den Wert s wie folgt:

Definition 9.5

$$B = SIZE_s(A) \quad : \Longleftrightarrow \quad f_B = \begin{cases} f_{ds(A)} & \text{falls } s \geq 0 \\ \neg f_{ds(\neg A)} & \text{sonst} \end{cases}$$

Abbildung 9.13 zeigt das Ergebnis eines Oversize und eines Undersize auf eine Fläche mit zwei Teilflächen, wobei eine dieser Teilflächen ein Loch aufweist. Mit der Sizingoperation kann sich die Topologie der Ebene ändern. Beim Oversize, d.h. der Sizingwert ist größer als 0, können Löcher „zuwachsen“, oder vorher getrennte Flächen können „zusammenwachsen“. Beim Undersize

³Synonym dazu: Sizing.

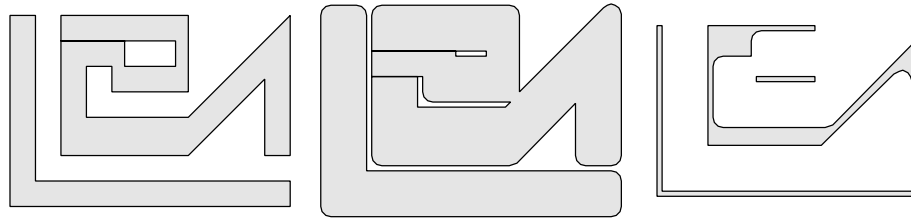


Abbildung 9.13: Ergebnis einer Sizingoperation, Sizingwert größer 0 (Oversize) und Sizingwert kleiner 0 (Undersize).

ergeben sich analoge Effekte, wenn man diese Operation - entsprechend der Formalisierung - als Oversize der inversen Fläche betrachtet. Dem Zuwachsen von Löchern (Oversize) entspricht das Verschwinden von Fläche (Undersize). Dem Zusammenwachsen von Fläche (Oversize) entspricht das Zusammenwachsen von Löchern (Undersize).

Wenn man beim Sizing eine strenge Definition zugrunde legt, entstehen beispielsweise beim Oversize eines Quadrates vier Kreisabschnitte, die die Layoutbeschreibung wesentlich verkomplizieren. In der Regel ist beim Sizing um den Wert d ein Fehler von $\sqrt{2} \cdot d$ akzeptabel, so daß dann aus einem Quadrat beim Oversize wieder ein entsprechend größeres Quadrat entsteht.

Bei der Überprüfung der Polysiliziumgates einer Schaltung müssen, zusätzlich zur reinen Überlappungsfläche von Diffusion und Polysilizium, die erlaubten Gateüberstände mitbetrachtet werden (Abbildung 9.14). Zu diesem Zweck wird zunächst die im ersten Schritt extrahierte Überlappungsfläche durch einen Oversize um den Wert der Überlappung expandiert. Anschließend erfolgt eine AND-Operation mit dem Gesamt-Polysilizium, um den Gateumriß zu rekonstruieren.

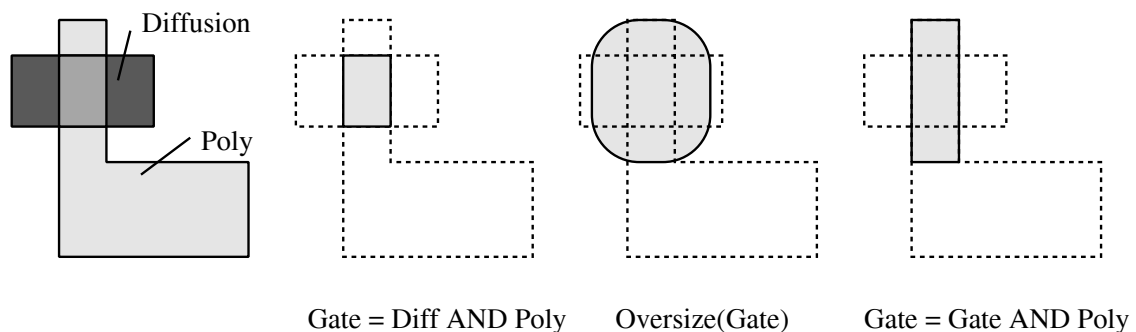


Abbildung 9.14: Extraktion der Gateflächen mit Überstand.

9.4.5 Selektionsoperationen

In diesem Abschnitt sollen einige Operationen betrachtet werden, die es erlauben, bestimmte Teilflächen einer Ebene zu selektieren. Diese Auswahl kann zunächst nach topologischen Gesichtspunkten getroffen werden. So kann man Teilflächen selektieren, die eine Teilfläche einer anderen Ebene überlappen oder berühren. An welcher Stelle die Überlappung auftritt ist dabei völlig uner-

heblich. Nachdem die Kriterien der Flächenoperationen bisher lokal entscheidbar waren, handelt es sich nun um eine Operation, die von Eigenschaften abhängt, die jeweils zusammenhängenden Teilflächen zugeordnet werden.

Solche topologische Operationen werden beispielsweise dazu benötigt, um Guardringe anhand ihrer Überlappung oder Berührung mit der Wanne zu identifizieren. Für diese Guardringe gelten oftmals spezielle Designregeln, die nur dann überprüfbar sind, wenn sich die entsprechenden Teilflächen sicher identifizieren lassen. Eine andere Anwendung liegt in dem Test, ob alle Wannen mit Wannenkontakten versehen sind. Durch die Topologieoperationen lassen sich die Wannenflächen selektieren, die eine Überlappung mit Wannenkontakten haben. Alle übrigen Wannenflächen werden nicht kontaktiert und sind somit offensichtlich fehlerhaft.

Neben der topologischen Selektion gibt es eine ganze Reihe weiterer Kriterien der Teilflächen, die zur Selektion verwendet werden können. Die wichtigsten Eigenschaften sind hierbei:

- die Anzahl der Punkte der zugehörigen polygonalen Beschreibung,
- die Fläche (z.B. in nm^2),
- der Umfang,
- die Identifikation, die jeder Teilfläche zugeordnet ist.

Für die ersten drei Kriterien sollten jeweils Bereiche angebbar sein. Außerdem ist neben der eigentlichen Selektion auch die invertierende Selektion wünschenswert, bei der gerade die Teilflächen selektiert werden, die die spezifizierten Eigenschaften nicht erfüllen. Schließlich ist es auch wichtig, daß eine Kombination von Eigenschaften als Selektor verwendet werden kann. Eine typische Anwendung ist die Selektion fehlerhafter Kontaktlöcher, die nicht viereckig sind, einen bestimmten Umfang und eine bestimmte Fläche aufweisen.

9.4.6 Meßoperationen

Die Meßoperationen dienen dazu, in den aufbereiteten Flächendarstellungen Entwurfsregelverletzungen zu markieren. Sie bilden die Essenz der geometrischen Regelkontrolle. Die üblicherweise implementierten Standardoperationen prüfen auf:

1. Mindestabstand,
2. Mindestbreite,
3. Mindestüberlappung,
4. Mindestüberhang.

Innerhalb einer Ebene werden die Operationen 1 und 2 durchgeführt. Die Überprüfungen 1, 3 und 4 beziehen sich auf die Teilflächen zweier Ebenen. Abbildung 9.15 illustriert die vier grundlegenden Regelkontrollen.

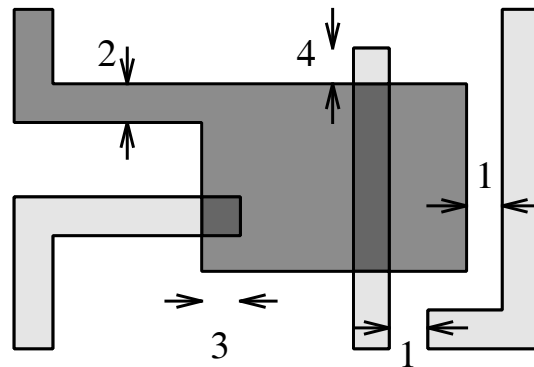


Abbildung 9.15: Kontrolloperationen: (1) Mindestabstand, (2) Mindestbreite, (3) Mindestüberlappung, (4) Mindestüberhang.

Im folgenden wird ein beispielhafter Algorithmus für die Meßoperationen vorgestellt, der die Anwendung des Plane-Sweep-Algorithmus illustrieren soll. Bei der Implementierung kantenbasierter Checks kann man davon ausgehen, daß eine Entwurfsregelverletzung grundsätzlich an einzelnen Paaren von Kanten⁴ sichtbar wird. Diese sind beispielsweise die beteiligten, gegenüberliegenden Kanten an der Stelle, wo sich zwei Flächen zu nahe kommen. Abhängig von der betrachteten Regelkontrolle sind oftmals weitere Kanten aus dem lokalen Bereich zu betrachten, um zu entscheiden, ob sich zwei Kandidaten für eine Designregelverletzung „sehen“.

Die kantenbasierte Entwurfsregelüberprüfung läßt sich im wesentlichen auf wenige Grundoperationen reduzieren. Zunächst müssen diejenigen Kantenpaare extrahiert werden, die Kandidaten für eine Entwurfsregelverletzung sein können. Da praktisch alle Entwurfsregeln lokalen Charakter besitzen, ist hier der Abstand das Hauptkriterium. Die Berechnung des Abstandes zwischen zwei beliebig schrägen Kanten ist jedoch recht aufwendig, so daß aus Effizienzgründen zunächst nur die Überlappung von Rechtecken überprüft wird, die die Kanten umschließen. Die Rechtecke werden vorher um den halben Abstandswert der zu überprüfenden Regel erweitert, siehe Bild 9.16. Somit ist nun das Problem zu lösen, für eine Menge von Rechtecken in der Ebene sämtliche Überschneidungen zu erkennen. Die Seiten dieser Rechtecke liegen jeweils parallel zur x- und y-Achse.

Um nun alle Paare sich überlappender Rechtecke zu finden, wird in einem „Plane-Sweep“ eine waagerechte Scanline von unten nach oben über die Maskenebene geschoben. Die Überprüfung der y-Überlappung ergibt sich im wesentlichen implizit, da in der aktuellen Datenstruktur nur diejenigen Rechtecke abgelegt sind, die von der Scanline geschnitten werden. Die x-Überlappung wird jeweils explizit überprüft.

Für jedes neue y-Niveau werden nun folgende Aktionen durchgeführt: in einem ersten Durchlauf durch die als Liste organisierte Scanline S werden alle „neuen“ Rechtecke nach x- Koordinaten einsortiert. Weiterhin wird eine ebenfalls nach x sortierte Hilfsliste H aufgebaut. Diese enthält einen Platzhalter für jedes Rechteck, das keine y-Überlappung mehr mit den „neuen“ Rechtecken, d.h. der Scanline, hat und mithin zum Löschen ansteht. Im zweiten Durchlauf werden nun S und H „gegeneinander“ durchgesehen, um für jeden Platzhalter aus H alle überlappenden Rechtecke in S zu finden. Durch die x-Sortierung der Liste S ist gewährleistet, daß alle Rechtecke, die ein Element

⁴Dabei wird eine Flächendarstellung mit den zugehörigen waagerechten Kanten vorausgesetzt.

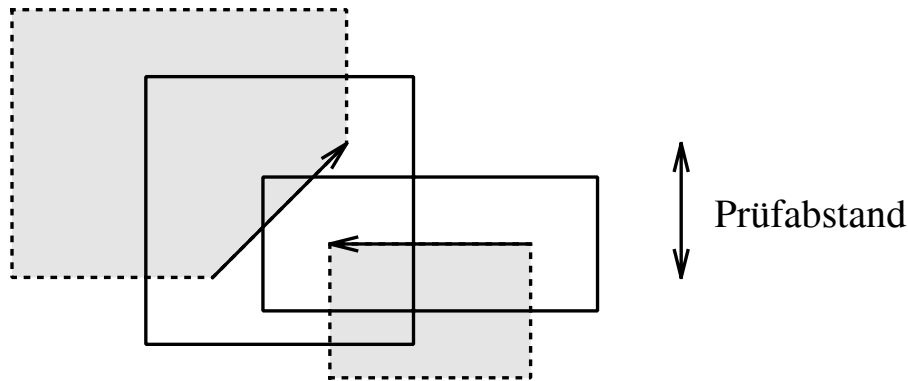


Abbildung 9.16: Detektion von Kandidatenkanten anhand einer Überlappungsprüfung der umschließenden Rechtecke.

in H überlappen, hintereinander liegen. Gleichzeitig werden alle Rechtecke in S gelöscht, für die ein Platzhalter existiert.

Die Überlappungsprüfung für eine Kante findet statt, unmittelbar bevor sie aus der Scanline gelöscht wird. Deshalb ist das doppelte Auftreten von Kantenpaaren, d.h. A interagiert mit B und B mit A, ausgeschlossen. Bei Kantenpaaren, die im selben Durchlauf gelöscht werden, kann eine feste Reihenfolge zwischen diesen vorgegeben werden. Das Ergebnis der oben beschriebenen Operation ist die sukzessive Generierung von jeweils einer Kante plus einer Liste von Kanten, die möglicherweise mit dieser interagieren. Die Einzelkante bildet in Verbindung mit einer Kante aus der Liste das zu untersuchende Kantenpaar, während die übrigen Kanten aus der Liste zur Überprüfung der „Sichtbarkeit“ dienen. Zu diesem Zweck wird untersucht, ob die Sicht zwischen den beiden Kanten nicht durch andere Kanten derselben Entwurfsebenen versperrt ist. In aller Regel werden Fehler nur gewertet, wenn die Sicht zwischen den verursachenden Kanten nicht versperrt ist. Mit diesem Algorithmus lassen sich auch mehrere verschiedene Maskenebenen bearbeiten, indem ihre Kanten zusammensortiert werden. Damit sind auch Regelkontrollen zwischen Maskenebenen möglich.

Im folgenden soll gezeigt werden, wie Entwurfsregelverletzungen bezüglich eines Kantenpaars mit einigen einfachen Überprüfungen erkannt werden können. Möglich sind in diesem Zusammenhang sowohl Regelkontrollen innerhalb einer Maskenebene als auch zwischen mehreren Maskenebenen. Es werden folgende Designregeltypen überprüft:

1. Mindestbreite,
2. Mindestabstand innerhalb einer Maskenebene,
3. Mindestabstand bezüglich desselben Polygons (Notches),
4. Mindestabstand zwischen zwei Maskenebenen,
5. Mindestüberhänge,
6. Mindestüberlappungen.

Es können auch in einem Lauf beliebige Kombinationen von Regeln überprüft werden, wobei die Laufzeit nur unwesentlich ansteigt, da das Sammeln der Kantenpaare die Hauptaufgabe darstellt. Diese ist bei einem solchen Lauf nur einmal zu bewältigen. Tabelle 9.1 zeigt ein Beispiel für einen Mehrfachcheck.

Mindestbreite	(POLYSILIZIUM)
Mindestabstand	(POLYSILIZIUM)
Mindestbreite	(DIFFUSION)
Mindestabstand	(DIFFUSION)
Mindestüberhang	(POLYSILIZIUM über DIFFUSION)
Mindestüberhang	(DIFFUSION über POLYSILIZIUM)

Tabelle 9.1: Beispiel für einen Mehrfachcheck.

Folgende Kriterien entscheiden darüber, ob zwischen den Kanten A und B eines Kantenpaares eine Entwurfsregelverletzung vorliegt:

- a) Zugehörigkeit zu Polygonen,
- b) Zugehörigkeit zu Maskenebenen,
- c) euklidischer Abstand,
- d) Winkel zwischen den (gerichteten) Kanten nach Translation der Fußpunkte der Kanten aufeinander,
- e) Aufspannen von „Fläche“ oder „Loch“ zwischen den beiden Kanten,
- f) Kante liegt innerhalb der „Flächenhalbebene“, die von der anderen Kante definiert wird,
- g) besondere Kantenmerkmale: Länge, senkrecht/waagerecht, parallel etc.,
- h) elektrische Informationen: z.B. Schaltungsknoten oder Signalnamen,
- i) Sichtbarkeit (möglicherweise versperrt durch Kanten aus derselben Maskenebene).

Diese Kriterien sind alle in konstanter Zeit zu prüfen. Beispielhaft sollen nun einige Kontrollen anhand der oben genannten Kriterien gebildet werden, wobei nicht relevante Kriterien weggelassen wurden.

Mindestbreite, Bild 9.17:

- a - beide Kanten stammen aus demselben Polygon
- b - beide Kanten stammen aus derselben Maskenebene
- c - der Abstand zwischen den Kanten ist kleiner als spezifiziert

d - der Winkel zwischen den Kanten liegt zwischen 90 und 270 Grad

e - die Kanten spannen „Fläche“ auf

i - die Kanten „sehen“ sich zumindest teilweise

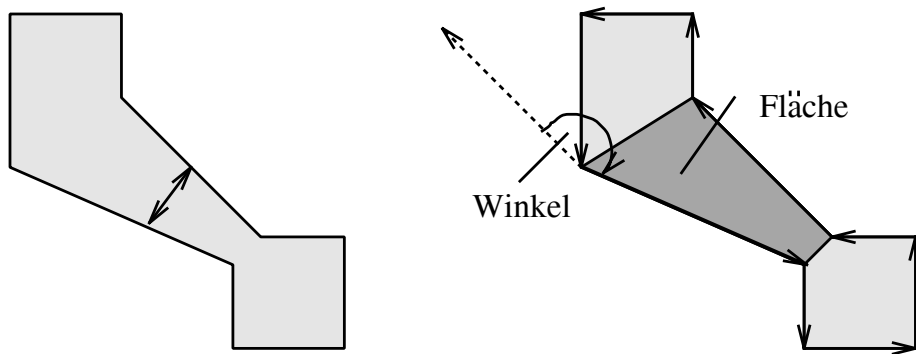


Abbildung 9.17: Designregelverletzung: Mindestbreite.

Mindestabstand über zwei Maskenebenen, Bild 9.18:

b - die Kanten stammen aus verschiedenen Maskenebenen

c - der Abstand zwischen den Kanten ist kleiner als spezifiziert

d - der Winkel zwischen den Kanten liegt zwischen 90 und 270 Grad

e - die Kanten spannen ein „Loch“ auf

i - die Kanten „sehen“ sich zumindest teilweise

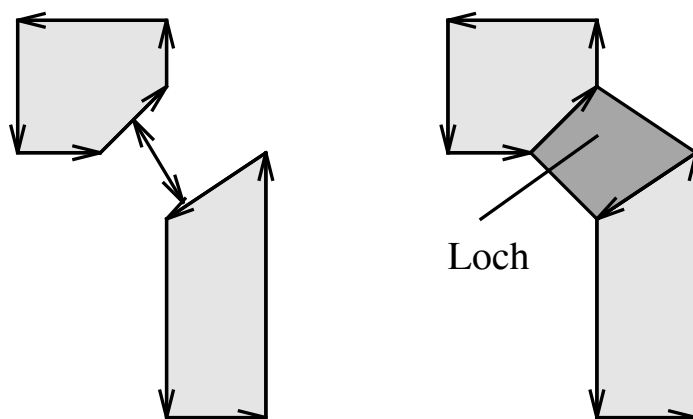


Abbildung 9.18: Designregelverletzung: Mindestabstand.

Mindestüberhang von Maskenebene A über Maskenebene B, Bild 9.19:

- b - die Kanten stammen aus verschiedenen Maskenebenen
- c - der Abstand zwischen den Kanten ist kleiner als spezifiziert
- d - der Winkel zwischen den Kanten liegt nicht zwischen 90 und 270 Grad
- f - die Kante aus B liegt in der „Flächenhalbebene“ der Kante aus A
- i - die Kanten „sehen“ sich zumindest teilweise

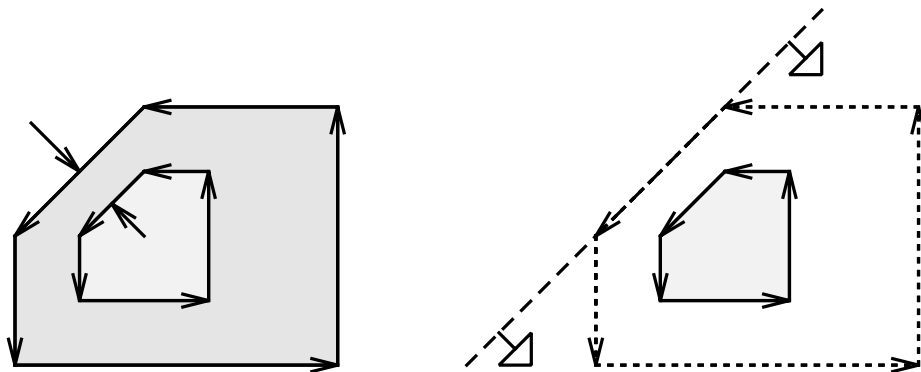


Abbildung 9.19: Designregelverletzung: Überhang Maskenebene A über B.

Wesentlich bei dem vorliegenden Algorithmus zur Prüfung geometrischer Entwurfsregeln ist, daß keine korrekten Flächenbeschreibungen vorausgesetzt werden. Beliebige Kantenmengen können einer Regelprüfung unterworfen werden. Damit lassen sich sukzessive Selektionen durchführen, was die Realisierung besonders komplizierter Prüfungen, z.B. konditionaler Regeln, ermöglicht.

9.4.7 Formulierung eines Regelsatzes

Typischerweise wird zur Prüfung der Designregeln eine Sprache benötigt, in der die gewünschten Designregeln formuliert werden können. Eine solche Sprache enthält in der Regel Konstrukte für die folgenden Operationen:

- Einlesen und Schreiben von Layoutdateien,
- Messung von Abständen, Breiten, Überhängen etc.,
- Boole'sche Operationen,
- metrische Operationen,
- Selektionsmechanismen,
- Berührungsoperationen.

Im folgenden wird eine Realisierung der in Abbildung 9.11 auf Seite 106 aufgestellten Regeln angegeben. Sie wurde in der Sprache HISCRIPIT des Verifikationssystems HIPARE-SYSTEM formuliert.

```
//=====
//
// Prozedur      : Lambda-Regeln
//
// Pruefung von geometrischen Entwurfsregeln mit Hipare-system
//
// Technologie : Lambda
// Bauelemente : PMOS, NMOS
// Ebenen      : Standard
// Autor       : Georg Pelz
// Datum      : 3.12.96
// Version     : 1.1
//
//=====

// Einlesen einer Layoutdatei
// $1 entspricht dem ersten Parameter der Prozedur, $2 dem zweiten

indata( filename = '$1', structurename = '$2');

// Definition der einzelnen Entwurfsebenen (= layer), die in der Layoutdatei
// mit Nummern bezeichnet werden und hier mit Namen.

input ( layername = NDIFF      , layernumber = 20); // N-Aktivgebiet
input ( layername = PDIFF      , layernumber = 21); // P-Aktivgebiet
input ( layername = POLY       , layernumber = 25); // Polysilizium
input ( layername = MET1       , layernumber = 32); // Metallisierungsebene
input ( layername = CUT        , layernumber = 29); // Kontaktloch
input ( layername = NSUB       , layernumber = 35); // N-Wanne

// Leitungsbreiten von NDIFF, PDIFF, MET1, POLY, CUT, CUTMM

width   ( layername = NDIFF, value = 2000,
          report = "Breite NDIFF (2 lambda)");

width   ( layername = PDIFF, value = 2000,
          report = "Breite PDIFF (2 lambda)");

width   ( layername = MET1, value = 2000,
          report = "Breite MET1 (2 lambda)");

width   ( layername = POLY, value = 2000,
          report = "Breite POLY (2 lambda)");

width   ( layername = CUT, value = 2000,
          report = "Breite CUT (2 lambda)");

// Abstände zwischen Flächenebenen desselben Materials
// NDIFF, PDIFF, MET1, POLY, CUT, CUTMM
```

```

spacing1 ( layername = NDIFF, value = 2000,
           report = "Abstand NDIFF (2 lambda)");

spacing1 ( layername = PDIFF, value = 2000,
           report = "Abstand PDIFF (2 lambda)");

spacing1 ( layername = MET1, value = 2000,
           report = "Abstand MET1 (2 lambda)");

spacing1 ( layername = POLY, value = 2000,
           report = "Abstand POLY (2 lambda)");

spacing1 ( layername = CUT, value = 2000,
           report = "Abstand CUT (2 lambda)");

// Ueberhang MET1 ueber CUT
//           POLY ueber CUT
//           NDIFF ueber CUT
//           PDIFF ueber CUT

overhang (layername1 = MET1, layername2 = CUT, value = 1000,
          report = "MET1 "uber CUT (1 lambda)");

overhang (layername1 = POLY, layername2 = CUT, value = 1000,
          report = "POLY "uber CUT (1 lambda)");

overhang (layername1 = NDIFF, layername2 = CUT, value = 1000,
          report = "NDIFF "uber CUT (1 lambda)");

overhang (layername1 = PDIFF, layername2 = CUT, value = 1000,
          report = "PDIFF "uber CUT (1 lambda)");

// Erzeugung der N- und P-Gates

NG = merge (operation = PDIFF and POLY);
PG = merge (operation = NDIFF and POLY);

NGATE_SIZED = size (layername = NG, value = 2000);
PGATE_SIZED = size (layername = PG, value = 2000);

NGATE = merge (operation = NGATE_SIZED and POLY);
PGATE = merge (operation = PGATE_SIZED and POLY);

// Gatel"angen

width ( layername = NGATE, value = 2000,
        report = "L"ange NGATE (2 lambda)");

width ( layername = PGATE, value = 2000,
        report = "L"ange PGATE (2 lambda)");

// "Uberhaenge
// PGATE "uber PDIFF

```

```

//      PDIFF "uber PGATE
//      NGATE "uber NDIFF
//      NDIFF "uber NGATE

overhang (layername1 = PGATE, layername2 = PDIFF, value = 2000,
         report = "PGATE "uber PDIFF (2 lambda)");

overhang (layername1 = PDIFF, layername2 = PGATE, value = 2000,
         report = "PDIFF "uber PGATE (2 lambda)");

overhang (layername1 = NGATE, layername2 = NDIFF, value = 2000,
         report = "NGATE "uber NDIFF (2 lambda)");

overhang (layername1 = NDIFF, layername2 = NGATE, value = 2000,
         report = "NDIFF "uber NGATE (2 lambda)");

// Abstand PGATE/NGATE zu CUT

spacing2 ( layername1 = PGATE, layername2 = CUT, value = 1000,
         report = Abstand PGATE <-> CUT (1 lambda)");

spacing2 ( layername1 = NGATE, layername2 = CUT, value = 1000,
         report = Abstand NGATE <-> CUT (1 lambda)");

// Abstand NDIFF/PDIFF zu POLY (Leitung)

TMP          = merge ( operation = POLY minus PGATE);
POLY_LEITUNG = merge ( operation = TMP  minus NGATE);

spacing2 ( layername1 = POLY_LEITUNG, layername1 = NDIFF, value = 1000,
         report = "Abstand POLY_LEITUNG <-> NDIFF (1 lambda)");

spacing2 ( layername1 = POLY_LEITUNG, layername1 = PDIFF, value = 1000,
         report = "Abstand POLY_LEITUNG <-> PDIFF (1 lambda)");

// Abstand NDIFF zu NSUB (Wanne)

spacing2 ( layername1 = NDIFF, layername1 = NSUB, value = 8000,
         report = "Abstand NDIFF <-> NSUB (8 lambda)");

// "Uberhang NSUB "uber PDIFF

overhang (layername1 = NSUB, layername2 = PDIFF, value = 4000,
         report = "NSUB ueber PDIFF (4 lambda)");

// Breite NSUB

width      ( layername = NSUB, value = 6000,
         report = "Breite NSUB (6 lambda)");

// Abbildung der Fehler in ein Layoutfile

outdata (filename = "error");

```

9.4.8 Schaltungsextraktion

Nach der Entwicklung eines den Designregeln entsprechenden Layouts stellt sich die Frage, ob die richtige Schaltung entworfen worden ist. Eine solche Prüfung könnte entweder durch Simulation oder durch Vergleich mit dem Schaltplan durchgeführt werden. Sowohl Simulation als auch Vergleich benötigen eine Netzliste der als Layout vorliegenden Schaltung, d.h. eine Liste aller Bauelemente und deren Verbindungsstruktur. Gegenstand dieses Abschnitts ist die Schaltungsextraktion, die gerade die Transformation vom Layout zur Netzliste bewerkstelligt. Dies wird dadurch erreicht, daß die Bauelemente aus der Geometrie rückerkannt werden. Dabei werden die Transistoren an ihrer geometrischen Struktur erkannt. Bei Kapazitäten und Widerständen wird jeweils eine besondere Entwurfsebene definiert, die manuell über die betreffenden Bauelemente gelegt wird und somit das Erkennen der gewünschten Bauelemente wesentlich erleichtert.

Widerstände können beispielsweise durch Polybahnen realisiert werden. Welche Polybahn nun einen gewünschten Widerstand und welche einen parasitären Widerstand repräsentiert, wird durch diese zusätzliche Ebene festgelegt. Für MOS-Transistoren ist eine zusätzliche Entwurfsebene nicht notwendig. Diese können einfach anhand der Überkreuzung von Diffusion und Polysilizium erkannt werden, was durch ein Boole'sche AND über die betreffenden Ebenen erreicht wird. Diese Gates werden anschließend von der Diffusionsfläche abgezogen. Die Source- und Drainflächen ergeben sich durch Berührungsoperationen zwischen Gate und der übrigen Diffusion. Aus dem Umriß der Gateflächen lassen sich außerdem Parameter wie das W und L des Transistors berechnen. Schließlich werden die Gateflächen auch vom Polysilizium abgezogen, um Leitungspoly von Gatepoly zu unterscheiden.

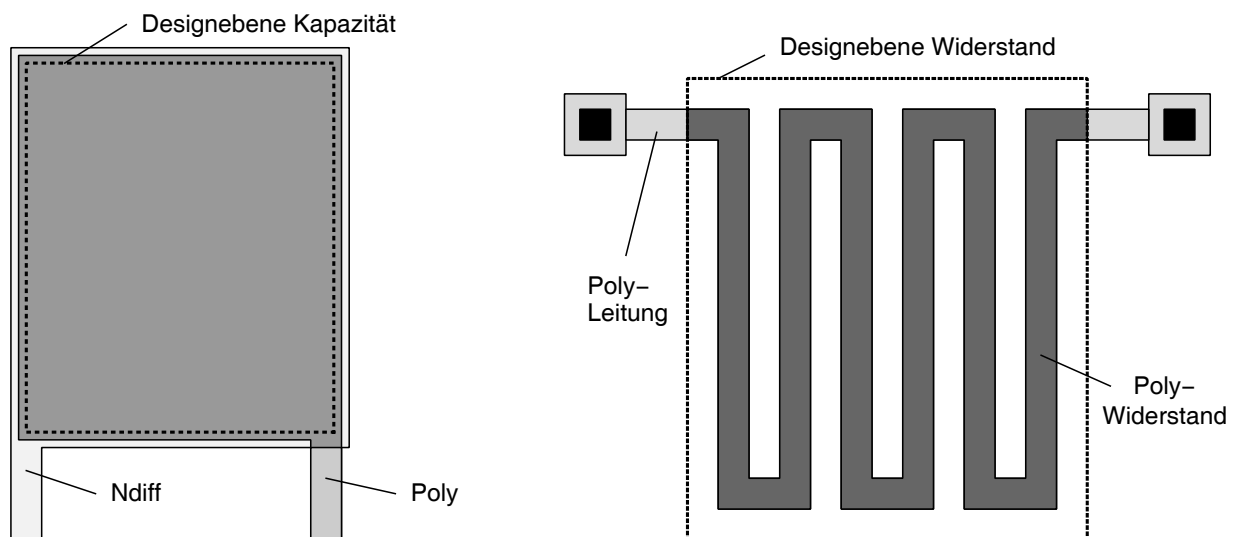


Abbildung 9.20: Extraktion von Kapazitäten (links) und Widerständen (rechts) durch zusätzliche Entwurfsebene.

Eine Möglichkeit zur Realisierung von Widerständen besteht beim MOS-Entwurf darin, daß recht lange Polyleitungen in Mäanderform gefaltet werden. Je nach der Technologie sind die spezifischen Widerstände unterschiedlich. Ein typischer Wert liegt bei $25 \Omega/\square$. Damit wird der Wider-

stand der Leitung mit einer Länge $l = n \times \text{Breite der Leitung}$ zu $n \times 25\Omega$. Dieser und andere Parameter lassen sich ebenfalls extrahieren. Typischerweise wird eine Sonderebene über die Polysiliziumflächen gelegt, die als Widerstände gedacht sind. Mit Hilfe einer Boole'schen Flächenoperation werden diese Flächenstücke anschließend aus dem Polysilizium herausgeschnitten und können zur Extraktion gesondert behandelt werden.

Eine Kapazität ergibt sich in einem MOS-Prozeß dadurch, daß man zwei Flächen - z.B. Metall und Polysilizium - übereinander legt. Jede dieser Flächen bildet eine Kondensatorplatte. Beeinflußt von der Technologie ergeben sich die Werte für die Kapazität in Abhängigkeit von der Plattenfläche. Für die Extraktion wird auch hier eine Sonderebene über jede Kapazität gelegt.

Auf diese Art kann man für alle Bauelementetypen Ebenen definieren, die genau die betreffenden Bauelemente bezeichnen. Die Verbindungen zwischen den Bauelementen ergeben sich durch Metall- und Polysiliziumleitungen. Zunächst bildet jedes Polygon einer solchen Entwurfsebene einen eigenständigen elektrischen Knoten. Unter Berücksichtigung von Kontaktlöchern, die ein Metall- und ein Polysiliziumpolygon berühren, werden solche Knoten zu größeren Knoten zusammengefaßt. Wenn dies erreicht ist, werden die Kontaktlöcher zu den Source- und Drainflächen untersucht. Dadurch ergeben sich die Anschlüsse zu den betreffenden MOS-Transistoren. Dort wo Polysilizium an Gateflächen oder Widerstandsflächen angrenzt, ergeben sich ebenfalls Anschlüsse. Diese werden durch eine Berührungsoperation detektiert.

Neben der Extraktion von Bauelementen und ihren Parametern lassen sich auch parasitäre Bauelemente berechnen. Diese sind zwar für einen Schaltungsvergleich hinderlich, ermöglichen aber eine Simulation auf Bauelementeebene unter Berücksichtigung von parasitären Effekten. Auf diese Art und Weise können Leitungswiderstände, Koppelkapazitäten oder Kapazitäten zum Substrat berücksichtigt werden. Die Schaltungssimulation - insbesondere mit parasitären Bauelementen - ist aber für nennenswerte Schaltungen in aller Regel zu aufwendig. Um trotzdem parasitäre Effekte bewerten zu können, verwendet man Programme (delay calculator), die aus den parasitären Widerstands- und Kapazitätswerten reale Verzögerungen berechnen. Diese Information wird anschließend in die ursprüngliche Gatterschaltung zurückpropagiert, so daß eine Logiksimulation unter Berücksichtigung der parasitären Effekte ablaufen kann.

9.4.9 Artwork

Üblicherweise wird der Entwurf nicht in den für die Fertigung verwendeten Maskenebenen sondern in Entwurfsebenen, in denen eine Schaltungsentwicklung leichter durchgeführt werden kann. Die automatische Generierung der Fertigungsebenen aus den Entwurfsebenen stellt die sichere Einhaltung bestimmter Entwurfsregeln sicher. Außerdem wird die Überprüfung der Entwurfsregeln vereinfacht. Ein Vorteil der Entwurfsebenen besteht zum Beispiel darin, daß die N- und P-Aktivgebiete hier jeweils durch eine eigene Ebene dargestellt werden. Bei den Maskenebenen wird die gesamte Aktivebene erst durch N- und P-Dotierungen personalisiert. Ein solcher Entwurf mit Maskenebenen ist deutlich fehlerträchtiger. Die erste Aufgabe des Artwork besteht darin, die Entwurfsebenen in Maskenebenen umzuwandeln. Dies geschieht im wesentlichen auf Basis der oben vorgestellten Flächenoperationen.

Im zweiten Schritt der Datenaufbereitung werden Effekte im Fertigungsprozeß, die die Dimensionierung der einzelnen Flächen betreffen, durch eine Kontraktion oder Expansion (Sizing) einzelner Flächen kompensiert. Beispielsweise wird die Ausdiffusion der Wannenimplantation durch eine

Kontraktion der Wannenebene im Entwurf berücksichtigt. Außerdem werden in diesem Schritt - ebenfalls durch Sizing - die Verzerrungen, die sich durch die Geräte zur Reticle/Maskenherstellung ergeben, berücksichtigt.

In der dritten Stufe werden aus den abstrakten Entwurfsdaten physikalische Strukturen erzeugt. Wie bei der Herstellung von Schwarzweißfotos wird zunächst eine Art „Negativ“ benötigt, das Reticle. Die Reticles bestehen aus einer Glasplatte auf die lichtundurchlässige Chromstrukturen aufgebracht werden. Um nun die Siliziumoberfläche strukturieren zu können, wird auf den Wafer zunächst gleichmäßig ein Fotolack aufgebracht. Mit Hilfe des Reticles wird dieser dann entsprechend der Chromschicht belichtet. Der belichtete Fotolack wird entfernt; an unbelichteten Stellen bleibt er stehen. Diese Stellen sind im folgenden Prozeßschritt für die Bearbeitung nicht zugänglich. Somit wird die Strukturierung der Chipoberfläche zum Beispiel durch Ätzen an den freien Stellen ermöglicht. Bei anderen Verfahren werden aus den Reticles Masken erzeugt, die ihrerseits zur selektiven Belichtung der Chipoberfläche verwendet werden. Wesentlich für diesen Vorgang ist das Reticle, dessen Chromschicht seinerseits durch einen Fotoprozeß strukturiert werden muß. Zunächst wird die Glasplatte komplett mit Chrom und dann mit Fotolack überzogen. Abhängig vom Design der entsprechenden Entwurfsebene müssen nun bestimmte Stellen des Fotolacks belichtet werden. Der belichtete Fotolack wird entfernt, und das darunterliegende Chrom kann weggeätzt werden, so daß das Reticle an diesen Stellen lichtdurchlässig wird. Entscheidend bei der Herstellung der Reticles ist die Belichtung. Hier kann man sich nicht mit einem „Negativ“ behelfen; dieses soll ja gerade erst entstehen.

Es gibt zwei Verfahren, um diese Belichtung durchzuführen:

- Beschreiben mit Elektronenstrahlen (e-beam),
- Belichten rechteckiger oder trapezförmiger Bereiche durch einen Automaten (optical pattern generator).

Bei der Elektronenstrahl-Methode wird das gesamte Reticle durch ein feines Bildraster dargestellt. Für jeden Rasterpunkt ist damit entschieden, ob an dieser Stelle ein Elektronenstrahlpunkt gesetzt werden soll. Anschließend werden die Punkte zeilenweise belichtet. Das Verfahren ähnelt dem Aufbau eines Fernsehbildes.

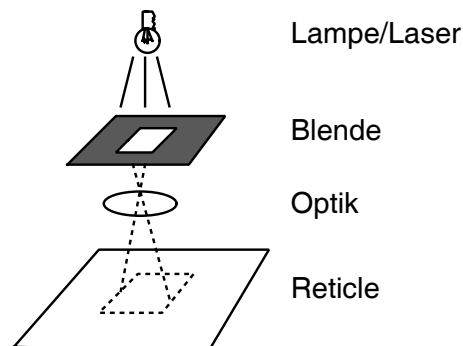


Abbildung 9.21: Schema eines Optical Pattern Generator.

Die zweite Methode - auf die hier näher eingegangen werden soll - bedient sich eines Automaten, der aus einer Lichtquelle und einer Mechanik besteht, die den Lichteinfall auf bestimmte rechteckige Gebiete auf dem Reticle beschränkt (Abbildung 9.21). Zu dieser Mechanik gehören folgende Bestandteile:

- zwei Blenden, die die Größe des Rechtecks (Breite und Höhe) einstellen,
- eine Drehmechanik, mit der auch gedrehte Rechtecke (bis 90 Grad) belichtet werden können,
- ein Verfahrtsch, mit dessen Hilfe Rechtecke an beliebigen Positionen des Reticles platziert werden können.

Durch Überlappung und Aneinandersetzen von Rechtecken lassen sich fast beliebige Flächen überdecken (coverage) (Abbildung 9.22). In zwei Fällen kann die genaue Überdeckung der gewünschten Fläche nicht garantiert werden. Spitze Winkel, die nach außen weisen, lassen sich allenfalls approximativ handhaben.

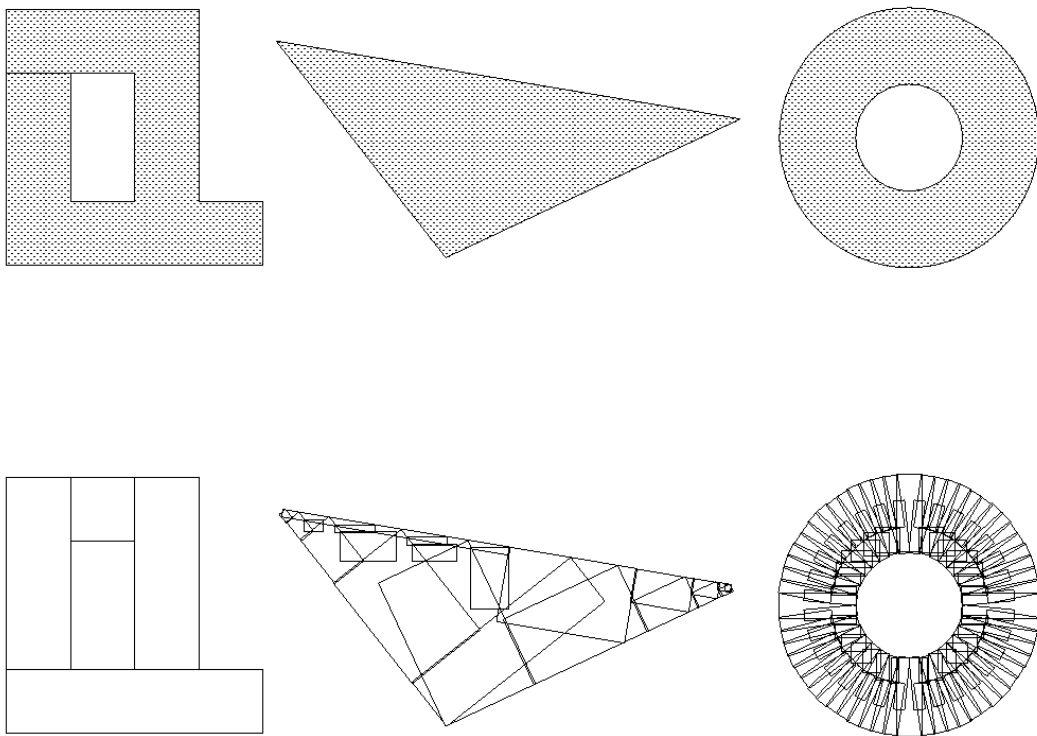


Abbildung 9.22: Rechtecküberdeckung von Flächen.

Die zweite Schwierigkeit ergibt sich dann, wenn die zu überdeckende Fläche eine Breite hat, die unter der minimal möglichen Blendenweite liegt. Hier ragt die belichtete Fläche zwangsläufig über die gewünschte Fläche hinaus.

Die wesentliche Aufgabe bei der Erzeugung von Recticles besteht nun darin, eine geeignete Rechtecküberdeckung für die gewünschte, durch umschließende Kanten beschriebene Fläche zu finden. Dieses Problem wird auch Fracturing genannt.

9.5 Schaltungsvergleich

Zusätzlich zu den bisher behandelten rein geometrischen Methoden zur Verifikation eines Layouts, welche alle auf die Einhaltung der physikalischen Entwurfsregeln abzielen, fällt ein wesentlicher Teil der Arbeit auf Konsistenzprüfungen zwischen dem Layout und der ursprünglich entworfenen Netzliste. Speziell beim Handentwurf von Layout erfordert der Designablauf eine solche Prüfung der Konsistenz. Wie sich im vorangegangenen Abschnitt gezeigt hat, kann man aus dem Layout eine Bauelementenetzliste extrahieren. Diese kann mit der ursprünglich zur Simulation verwendeten Netzliste verglichen werden. Wie dies geschieht, soll anhand eines vereinfachten Ablaufes verdeutlicht werden, Abbildung 9.23.

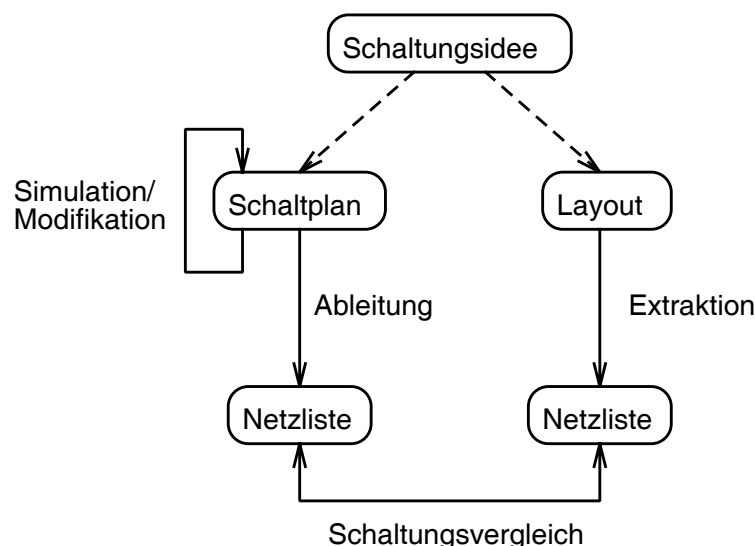


Abbildung 9.23: Vereinfachter Designablauf für den Handentwurf.

Normalerweise entsteht beim Handentwurf aus der Schaltungsidee zunächst ein Schaltplan (schematic). Diese erste Schaltungsbeschreibung wird zunächst intensiv simuliert und entsprechend modifiziert, bis die gewünschte Funktion erreicht ist. Anschließend wird die geometrische Implementierung der Schaltung - das Layout - von Hand erstellt. Hierbei wird der Entwerfer oftmals nur rudimentär mit Informationen aus dem Schaltplan unterstützt. Dieser manuelle Prozeß ist sehr fehleranfällig, da solche Schaltungen oft nur schlecht zu überschauen sind. Bereits kleinste Fehler können jedoch fatale Auswirkungen haben. Um nun einen Konsistenzvergleich zwischen Schaltplan und Layout zu bewerkstelligen, müssen beide zunächst in eine vergleichbare Darstellung überführt werden. Hierzu wird eine Netzliste aus dem Schaltplan abgeleitet und eine weitere aus dem Layout extrahiert. Diese Netzlisten bestehen aus Schaltungsprimitiven, wie Transistoren, Widerständen, Kapazitäten, Dioden (etc.). Durch einen Netzlistenvergleich können nun die Lay-

outfehler erkannt werden, die eine Änderung der Netzliste zur Folge haben, z.B. unbeabsichtigte Unterbrechungen und Kurzschlüsse.

9.5.1 Anforderungen

Die universelle Anwendbarkeit eines Werkzeugs zum Schaltungsvergleich soll durch folgende Anforderungen gesichert werden:

- **Topologisch verschiedene, aber funktional identische Schaltungen müssen richtig behandelt werden.**

Jeder Schaltungsvergleich, der den Praxisanforderungen gerecht werden soll, ist in erster Linie auch ein Topologievergleich. Die Topologie wird dabei in der Regel in Form eines Graphen (s.u.) abgelegt. Vergleiche auf Verhaltensbasis sind grundsätzlich auch möglich. Beim Funktionalvergleich durch Simulation ist man allerdings auf die Auswahl der „richtigen“ Stimuli angewiesen, da eine erschöpfende Simulation wegen des immensen Aufwands praktisch immer unmöglich ist. Der Funktionsvergleich durch einen formalen, mathematischen Beweis befindet sich zur Zeit noch im Forschungsstadium.

Leider kann die durchgängige, strukturelle Isomorphie der zu vergleichenden Netzlisten nicht vorausgesetzt werden. Beispielsweise sind die Eingangssignale eines NAND-Gatters vertauschbar, ohne daß sich die Funktion der Schaltung ändert. Die Transistortopologie erfährt dadurch jedoch durchaus eine Änderung. Diese Vertauschbarkeit wird oft ausgenutzt, um ein kompakteres Design zu erreichen. Ein geeigneter Schaltungsvergleich sollte nun in der Lage sein, dies bei einer topologischen Isomorphieprüfung zu berücksichtigen.

- **Die oft verschiedenen Hierarchien zwischen Layout und Schaltplan müssen berücksichtigt werden.**

Gerade in einem Layout, das von Hand erzeugt wurde, folgt die Hierarchie vielfach der geometrischen Zweckmäßigkeit, und nicht der elektrischen Zugehörigkeit. Daraus resultieren oft völlig verschiedene Hierarchien in Schaltplan und Layout, die im allgemeinen Fall nicht zur Reduzierung des Aufwands ausgenutzt werden können. Falls allerdings die Hierarchien isomorph sind, so soll der Schaltungsvergleich auf hierarchische Art und Weise durchgeführt werden.

- **Die Fehleranzeige muß eindeutig, unmißverständlich und an der richtigen Stelle erfolgen.**

Dies ist ein großes Problem, da die Netzlistendaten im Gegensatz zu den Geometriedaten alles andere als lokal sind. Beispielsweise ist bei einer Digitalschaltung ein Großteil der NMOS-Transistoren an Masse angeschlossen. Daraus folgt, daß Fehler oft erstaunliche „Fernwirkungen“ entfalten. Wenn, wie bei praktisch allen kommerziellen Werkzeugen, zwei flache Netzlisten mit Bauelementen verglichen werden, so führt beispielsweise die Vertauschung des Taktsignals und des invertierten Taktsignals auf oberster Ebene im Schaltplan zu zahllosen Fehlermeldungen, die kaum noch auf ihre Ursache hindeuten.

9.5.2 Algorithmen zum Schaltungsvergleich

Die Ansätze zum Schaltungsvergleich können in vier Gruppen unterteilt werden: manuelle Markierung, Simulation, Refinement mit Erweiterungen und der hierarchische Vergleich.

Der Schaltungsvergleich wird zur einfachen Aufgabe, wenn die meisten Schaltungselemente in der extrahierten Netzliste eine eindeutige Markierung tragen, die mit den Markierungen im Schaltplan korrespondiert. Einige der frühen Vergleichssysteme wenden diese Methode erfolgreich an. Voraussetzung ist allerdings, daß die Markierungen vor dem Vergleich manuell im Layout vergeben werden. Dies ist eine schwierige und ihrerseits fehleranfällige Arbeit.

In manchen Systemen wird der Vergleich nicht auf den Schaltungen, sondern auf den Simulationsergebnissen durchgeführt. Auf diese Art und Weise werden nur funktionale Aspekte berücksichtigt. Die funktionale Isomorphie kann allerdings nur durch eine erschöpfende Simulation garantiert werden, die sich in aller Regel aus Laufzeitgründen verbietet. Außerdem zeigen Unterschiede in den Simulationsergebnissen nur die Existenz von Inkonsistenzen zwischen den beiden Schaltungen auf. Der fehlerhafte Teil der Netzliste wird jedoch nicht lokalisiert.

Viele Werkzeuge aus der Literatur bilden nicht-hierarchische Schaltungen auf Graphen ab und setzen Graphisomorphie-Techniken ein, um die Topologie der beiden Schaltungen zu vergleichen. Die meisten verwenden Refinement (s.o.) als Grundprinzip. In beiden Schaltungen wird die Menge aller Netze oder aller Komponenten in Klassen von Elementen mit homogenen Eigenschaften partitioniert. Typische Eigenschaften sind zum Beispiel die Anzahl der zu einem Netz benachbarten Komponenten oder der Typ einer Komponente. Die Initialpartition wird sukzessive verfeinert, indem jeweils die Eigenschaften der Nachbarn mit berücksichtigt werden. Damit wird für jedes Element der Schaltung nach und nach der Bereich vergrößert, der betrachtet wird, um es von allen anderen Elementen zu unterscheiden. Elemente in einelementigen Klassen einer Schaltung können direkt mit Elementen der entsprechenden Klassen der anderen Schaltung identifiziert werden, da die Eigenschaften und somit auch die Klassen invariant unter Isomorphie sind. Falls jedoch Symmetrien in den Graphen auftreten, stellen sich Probleme ein, da symmetrische Zweige nicht topologisch unterschiedlich sind. Damit können die durch die Symmetrie entstandenen Äquivalenzklassen V_1 , V_2 und V_3 nicht in einelementige Klassen partitioniert werden. Üblicherweise wird ein Backtracking-Ansatz gewählt, um zu einer vollständigen Abbildung zu kommen.

In den letzten Jahren wurden zu dem Grundalgorithmus einige Verbesserungen erreicht. Die wichtigste war die Einführung der Gattererkennung, um das Problem für digitale (Teil-)Schaltungen vom Bauelementeniveau auf das Gatterniveau zu transformieren. Zusätzlich zur Reduktion der Komplexität wurde die Handhabung einiger wichtiger Probleme wesentlich vereinfacht, wie z.B. die Anschlußvertauschbarkeit. Beispielsweise können die Eingänge eines NAND vertauscht werden, ohne daß sich die Funktion der Schaltung ändert. Auf Bauelementeebene führt dies zu einer Vertauschung der NMOS-Transistoren, während auf Gatterebene lediglich die Eingänge als vertauschbar markiert werden müssen. Für Komplexgatter (z.B. And-Or-Invert-Gatter), die Vertauschbarkeiten von Anschlüssen und Anschlußgruppen aufweisen, ist das Vergleichsproblem auf der Ebene der Bauelemente kaum noch effizient zu lösen, da alle Permutationen von Anschlüssen betrachtet werden müssen. Viele Algorithmen wurden vorgeschlagen, um eine sinnvolle Gatterformierung zu erreichen. Die meisten verwenden die sogenannte „path-tracing“ Methode, um die Bauelemente in Gruppen zu partitionieren, aus denen Logikgatter erkannt werden können. Diese Algorithmen sind typischerweise auf die Bedürfnisse digitaler NMOS und CMOS Schaltungen

zugeschnitten. Dabei wird funktionale Information in die Gatterformierung einprogrammiert. Die Methode basiert darauf, sogenannte Signalflüsse zu berechnen. Quellen des Signalflusses sind die Schaltungseingänge und die Versorgungsleitungen. Der Signalfluß wird nur über Transistorkanäle, aber nicht über Gates verfolgt. Daraus ergibt sich eine Aufteilung der Transistoren einer Schaltung in Gruppen, die über den Signalfluß zusammenhängen. Diese Aufteilung kann man auch als Partition betrachten, da ein solcher Transistor grundsätzlich nur Mitglied einer Klasse sein kann. MOS-Gatter bilden in aller Regel solche Gruppen. Für Analogschaltungen oder nicht-MOS Technologien erhält man allerdings oftmals unsinnige Partitionen, die mit den Intentionen des Schaltungsentwicklers nichts mehr zu tun haben.

Einige hierarchische Ansätze wurden zur Lösung des Isomorphieproblems vorgeschlagen. Die meisten Methoden setzen eine exakt gleiche Hierarchie in Schaltplan und extrahierter Netzliste voraus. Dies ist oftmals nicht der Fall, da die Schaltplanhierarchie in der Regel die funktionale Organisation der Schaltung reflektiert, während die Layouthierarchie der geometrischen Zweckmäßigkeit folgt. In anderen Ansätzen werden die Hierarchien modifiziert, um sie isomorph zu machen. Für den allgemeinen Fall ist der hierarchische Schaltungsvergleich jedoch nicht einsetzbar.

9.6 Prüfung elektrischer Entwurfsregeln

Die Prüfung elektrischer Entwurfsregeln dient normalerweise dazu, eine aus dem Layout extrahierte Netzliste einer ersten Plausibilitätskontrolle zu unterziehen. So überprüft man beispielsweise, ob alle Knoten in der Schaltung einen Pfad über Transistorkanäle nach V_{ss} und V_{dd} aufweisen. Falls dies für einen Knoten nicht der Fall ist, dann kann dieser auch nicht auf logisch „1“ oder „0“ eingestellt werden, da keine leitende Verbindung mit der entsprechenden Versorgungsleitung hergestellt werden kann. Andere Regeln befassen sich mit der Polung von Dioden oder dem Bulk-Anschluss von MOS-Transistoren.

Die wesentlichen Eigenschaften eines Werkzeugs, das eine solche Netzlistenkontrolle durchführt, bestehen in der:

- **Flexibilität der Überprüfungen**

Da die Entwurfsregeln über Netzlisten stark von Faktoren, wie der gewählten Schaltungstechnik, der Teststrategie oder vom tatsächlichen Entwurf abhängen, variieren sie teilweise sogar von Schaltung zu Schaltung. Aus diesem Grund verursacht ein Mangel an Flexibilität einen nicht akzeptablen Aufwand für die Formulierung und Modifikation solcher Regelprüfungen.

- **Effizienz der Ausführung**

Da die hierarchische Bearbeitung von Schaltungen nicht immer möglich ist, muß die allgemeine Anwendbarkeit des Werkzeugs durch effiziente Ausführung der Checks gesichert werden. Beispielsweise sollen extrahierte, flache Schaltungen überprüft werden. Diese enthalten möglicherweise parasitäre Bauelemente, wie z.B. Koppelkapazitäten. Damit ist eine hierarchische Bearbeitung mit Sicherheit ausgeschlossen.

Die Überprüfung von elektrischen Regeln und Testbarkeitsregeln wird seit Jahren intensiv erforscht, und viele Algorithmen wurden vorgeschlagen. Die Arbeiten lassen sich in zwei Gruppen

einteilen: Programme und Expertensysteme. Programme bieten eine gute Abarbeitungseffizienz aber leiden an einem Mangel an Flexibilität. Für neue Regelprüfungen müssen neue Programme geschrieben werden. Variationen von Regeln erfordern Modifikationen von Programmen. In fast allen Fällen ist diese Programmieraufgabe schwierig und fehleranfällig. Der Arbeitsaufwand wird untragbar, wenn die angewendeten Regeln nicht wie in praktisch allen kommerziellen Systemen auf eine kleine Teilmenge generell gültiger Regeln beschränkt wird. Expertensysteme und ähnliche Ansätze bieten gute Flexibilität, aber haben Probleme mit der Laufzeit und den Speicheranforderungen. Die bislang untersuchten Schaltungen umfassen in der Regel höchstens einige Tausend Elemente.

Kapitel 10

Layout-Synthese

In diesem Abschnitt sollen Lösungen für die automatische Generierung von Layout dargestellt werden. Verschiedene Gütekriterien sind bei der Layoutsynthese zu betrachten:

- Einhaltung der geometrischen Entwurfsregeln

Die Einhaltung der geometrischen Entwurfsregeln (z.B. Leitungsbreite, Leitungsabstand und vieles mehr) ist eine notwendige, aber nicht hinreichende Bedingung für die Funktionsfähigkeit einer Schaltung.

- Leistungsfähigkeit der aus dem Layout resultierenden Schaltung

Die Spezifikation für eine Schaltung legt Werte für die Leistungsfähigkeit einer Schaltung fest, die etwa in Kategorien wie Taktrate oder Leistungsaufnahme gemessen wird. Das Erreichen der Spezifikation ist wesentliche Voraussetzung für den Erfolg der Schaltung. Die Leistungsfähigkeit einer Schaltung wird zu einem guten Teil durch ihre physikalische Realisierung, also das Layout bestimmt.

- Flächenbedarf der Schaltung

Der Flächenbedarf der Schaltung bestimmt die Anzahl der Chips pro Wafer und mithin die Stückkosten. Hinzu kommt außerdem, daß die Ausbeute sinkt, wenn die Chips größer werden. Beide Effekte bewirken zusammen, daß die Fläche einer der wesentlichen Faktoren ist, der die Wirtschaftlichkeit einer Schaltung bestimmt.

Die Layoutsynthese, ebenso wie das Handlayout, ist den oben genannten Zielen verpflichtet, wobei man üblicherweise davon ausgeht, daß die Einhaltung der Entwurfsregeln automatisch sichergestellt wird. Dies setzt allerdings eine fehlerfreie Synthesesoftware und deren korrekte Bedienung voraus. Man spricht in diesem Zusammenhang von der Korrektheit durch die Konstruktion (correctness by construction). Ein weiterer Vorteil der automatischen Synthese liegt in einer dramatischen Verkürzung der Entwurfszeit. Schließlich ermöglicht die Automatisierung der Layouterzeugung das schnelle Anpassen von Layouts an neue Technologien und damit neue Designregelsätze, während Handlayouts oftmals von Grund auf neu entwickelt werden müssten.

Die automatische Layoutsynthese hat jedoch auch Nachteile. Vielfach müssen gegenüber handentworfenen Schaltungen Nachteile in Fläche und Leistungsfähigkeit in Kauf genommen werden.

Außerdem ist eine automatische Layoutgenerierung üblicherweise recht unflexibel, so daß Sonderbauelemente, spezielle Entwurfsregeln und viele andere Randbedingungen nur unvollständig berücksichtigt werden können. Dies kommt insbesondere beim Analogentwurf zum Tragen, für den die Layoutsynthese trotz einiger Erfolge immer noch in den Kinderschuhen steckt.

Nachdem in einem symbolischen Schaltungsentwurf die Topologie einer Schaltung festgelegt wurde, kann die Erstellung des physikalischen Layouts automatisch durchgeführt werden. Dieser Schritt wird auch Kompaktierung genannt, da es hier in erster Linie darum geht, ein möglichst kompaktes (d.h. platzsparendes) Layout zu erhalten.

10.1 Symbolisches Layout

Das symbolische Layout hat seinen Ursprung in den frühen Siebziger Jahren. Damals erkannte man, daß sich komplexe Schaltungen leichter beherrschen lassen, wenn man Layoutelemente (z.B. Transistoren) durch Symbole darstellt. Diese Symbole lassen sich auf der nächsthöheren Abstraktionsebene zu weiteren Symbolen kombinieren. Somit ergibt sich ein Hierarchiebaum, dessen Wurzel durch ein Symbol für die gesamte Schaltung gebildet wird.

Auf diese Art und Weise läßt sich eine Schaltung sehr rationell darstellen. Außerdem besteht die Möglichkeit, den Symbolen weitere Informationen anzuhängen. Diese können beispielsweise die Testbarkeit oder Simulation der Schaltung betreffen. Weiterhin sind Informationen über den elektrischen Kontext und die Funktion eines Elements in dessen Symbol darstellbar. Wenn nur die Geometrie des Layouts - also das physikalische Layout - zur Verfügung steht, sind solche Daten nur sehr mühselig durch eine Extraktion zurückzugewinnen.

Von diesem Ansatz ausgehend, kann eine Layoutstrategie entwickelt werden, bei der der Schaltungsentwickler lediglich die Topologie einer Schaltung vorgibt. Dies geschieht unter Verwendung von Symbolen und Verbindungen zwischen Symbolen. Die Umwandlung der symbolischen Schaltung in ein physikalisches Layout kann automatisiert und einem Computer überlassen werden. Bei diesem Schritt wird die Einhaltung der geometrischen Entwurfsregeln automatisch erzwungen (correctness by construction). Die schwierige und fehlerträchtige Aufgabe der Erstellung eines physikalischen Layouts wird für den Schaltungsentwickler reduziert auf das Finden einer guten Schaltungstopologie.

Es gibt viele Variationen, wie symbolisches Layout aussehen kann. Beispielsweise gibt es Systeme, die das Layout ohne Änderungen zum physikalischen Layout darstellen, aber zusätzlich zur Geometrie festhalten, zu welchem elektrischen Knoten ein Layoutelement gehört. Andere Systeme verwenden möglichst einfache Symbole, um den manuellen Schaltungsentwurf zu vereinfachen. Dies ist in Abbildung 10.1 auf Seite 131 dargestellt.

Leitungen werden im symbolischen Layout durch Linien markiert, wobei die Leitungsbreite vernachlässigt wird. Die Überkreuzung einer Polylinie mit einer Diffusionslinie ergibt einen Transistor. Durchgezogene Linien bezeichnen Metallbahnen, wobei eine dünne Linie Metall1 und eine dicke Linie Metall2 beschreibt. Dies hat seine Entsprechung in der Realität, da die Schichtdicke von Metall2 deutlich größer als die von Metall1 ist. Der Grund dafür liegt darin, daß Metall2 bei einer Zweilagigen-Metallisierung die oberste Designebene ist, wodurch die Anforderungen bezüglich der Planarität wesentlich weniger rigide ausfallen. Die Diffusionsstreifen werden gepunktet dargestellt, wobei die P-Diffusionen dicker gezeichnet werden als die N-Diffusionen. Die

Motivation hierfür liegt darin, daß die Weite der PMOS-Transistoren oftmals größer als die der NMOS-Transistoren gewählt wird. Dies liegt an der geringeren Ladungsträgerbeweglichkeit von Löchern gegenüber von Elektronen. Das Polysilizium wird gestrichelt dargestellt. Die Kontaktlöcher schließlich werden durch ein Kreuz markiert, die Vias, d.h. die Kontakte zwischen Metall1 und Metall2, durch einen Kreis. Diese Festlegungen werden in Abbildung 10.1 illustriert.

————	Metall1 (grün)
-----	p-Diffusion (hellblau)
.....	n-Diffusion (dunkelblau)
-----	Poly (rot)
×	Kontaktloch (schwarz)
————	Metall2 (gelb)
○	Via (schwarz)

Abbildung 10.1: Legende für das symbolische Layout.

Zunächst sollen einige Inverter-Realisierungen in symbolischem Layout betrachtet werden. Abbildung 10.2 (a) zeigt die Transistorschaltung eines Inverters. Ein erstes symbolisches Layout, das dem Transistorschaltbild des Inverters nachempfunden wurde, ist in Abbildung 10.2 (b) dargestellt. Eingänge und Ausgänge werden hier zu den Seiten herausgeführt. Wenn man diese nun, wie bei Standardzellen üblich, von oben oder unten anschließt und außerdem die Transistoren um 90 Grad dreht, dann ergibt sich die Darstellung in 10.2 (c). Bei dieser werden die Anschlüsse nach außen in Polysilizium geführt, um die Versorgungsschienen zu überwinden. Dies wäre eine typische Lösung für einen Fertigungsprozeß mit nur einer Metallisierungslage. Der Klarheit halber wurden hier die Polyanschlüsse bis an die Zellgrenze geführt. Diese gehören aber in der Regel nicht zur Zelle, sondern werden im Rahmen der Verdrahtung zwischen den Zellen ausgeführt. Dies vor allem weil der Anschluß in der Regel nur entweder von oben oder unten erfolgt. Damit ist der andere Streifen Polysilizium nach außen an die Zellgrenze überflüssig und trägt lediglich zur parasitären Kapazität der Polysiliziumbahn bei.

Wenn der Inverter nun größere Lasten treiben soll, so müssen die Transistorweiten erhöht werden. Alternativ dazu kann man den PMOS- und NMOS-Transistor verdoppeln, wie es in 10.2 (d) gezeigt wird. Für dieses Beispiel sollen außerdem zwei Metallebenen zur Verfügung stehen, so daß für die Außenanschlüsse die zweite Metallebene verwendet werden kann. Dies wird hier durch die Vias, d.h. die Metall1-Metall2-Verbindungen angedeutet. Die senkrechten Metall2-Leitungen gehören ebenso wie die senkrechten Polyleitungen nicht zur Zelle, sondern zur Verdrahtung zwischen den Zellen.

Die Abbildung 10.3 zeigt drei Realisierungen eines NAND2-Gatters. In 10.3 (a) werden die Anschlüsse des NAND2 von links und rechts herangeführt. Die Darstellung in 10.3 (b) zielt auf ein Gatter, das ohne Verwendung einer zweiten Metallebene entwickelt wurde. Das symbolische Layout in 10.3 (c) schließlich sieht die zweite Metallisierungslage für die senkrechten Stichleitungen der Anschlüsse zu den Vias vor.

Die Transistorschaltungen einfacher Gatter lassen sich recht leicht in ein symbolisches Layout umsetzen. Bei Komplexgattern erfordert dies deutlich mehr Aufwand. In diesem Zusammenhang

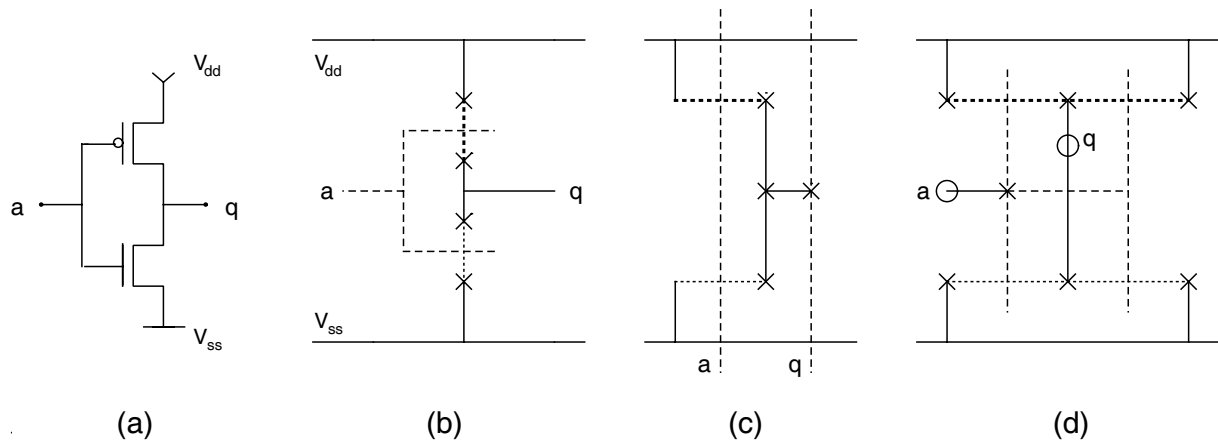


Abbildung 10.2: Verschiedene Realisierungen eines Inverters in symbolischem Layout.

lassen sich diverse Layout-Strategien definieren, von denen die am weitesten verbreitete hier vorgestellt werden soll. Unter der Maßgabe, daß ein möglichst einfaches und dichtes Layout entstehen soll, nimmt man folgende Festlegungen vor:

- N- und P-Diffusion werden als zwei einfache, waagerechte Streifen zwischen den beiden Versorgungsschienen geführt. Unterbrechungen dieser Streifen sollten möglichst vermieden werden. Damit hat man sich außerdem für Einheitstransistoren entschieden, da die Breite der Streifen konstant sein soll.
- Die Polysilizium-Gates werden als senkrechte Streifen konstanter Breite über die Zelle geführt. Überkreuzungen zwischen den Polysiliziumstreifen sollten möglichst vermieden werden, da zu diesem Zweck eine Metallbrücke für eines der Gates eingeführt werden muß.

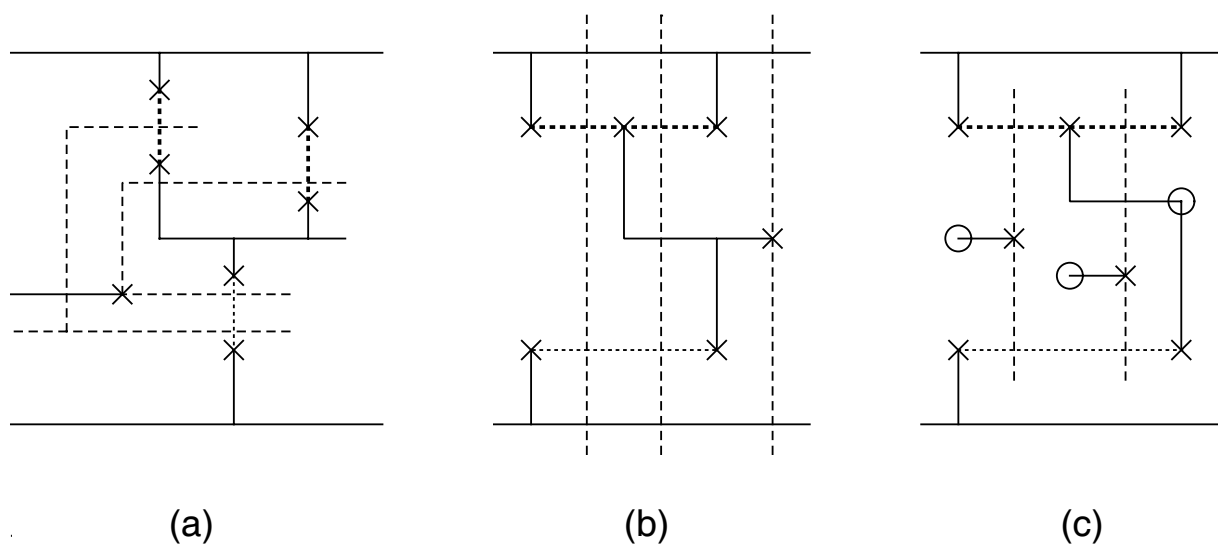


Abbildung 10.3: Verschiedenen Realisierungen von NAND-Gattern in symbolischem Layout.

Entscheidend für das Erreichen der obigen Ziele ist die Wahl der Reihenfolge der Gates von links nach rechts und damit der betreffenden Transistoren. Um das Problem hier einer automatisierten Lösung zugänglich zu machen, bildet man den N- und den P-Zweig eines komplementären CMOS-Gatters auf zwei zueinander duale Graphen ab. Die Knoten der Graphen werden durch die Verbindungen zwischen den Transistoren gebildet, die Kanten durch die Transistoren selbst. P- und N-Graph werden dann zunächst getrennt betrachtet. Gesucht ist nun eine zusammenhängende Sequenz von Kanten, also mithin von Transistoren. Eine solche Folge wird hier betrachtet, weil man im Layout nur dann Transistoren direkt aneinander plazieren darf, wenn sie an dieselbe Leitung angeschlossen sind. Da nun alle Transistoren im Gatter berücksichtigt werden, muß die Sequenz alle Kanten beinhalten. Falls eine solche Folge im Graphen gefunden werden kann, so kann man die Transistoren in jedem Fall in einem einzigen Streifen Diffusion anordnen. Ein solcher Pfad, der unterbrechungsfrei alle Kanten/Transistoren besucht, heißt auch Euler-Pfad. Damit verbleibt noch das Problem, die Polystreifen überkreuzungsfrei anzuordnen. Dies ist gerade dann möglich, falls ein gleicher Euler-Pfad für den N- und den P-Graphen existiert. Der obige Ablauf soll an einem Beispiel illustriert werden. Zu diesem Zweck wird ein AND-OR-INVERT- oder AOI-Gatter gewählt, siehe Abbildung 10.4, das auch schon in der CMOS-Einführung betrachtet wurde.

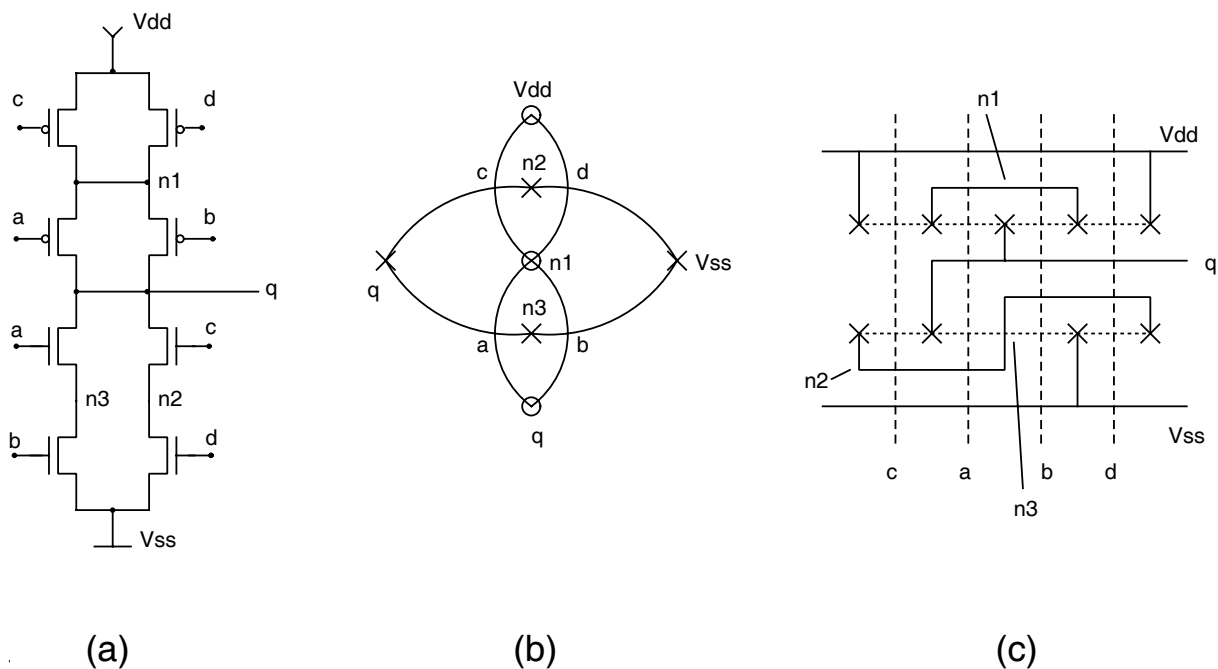


Abbildung 10.4: Entwicklung eines symbolischen Layouts für ein Komplexgatter am Beispiel des AND-OR-INVERT-Gatters.

In 10.4 (a) wird zur Erinnerung das Transistorschaltbild dargestellt. Der zugehörige N- und P-Graph werden in 10.4 (b) abgebildet. In diesen Graphen lassen sich nun diverse Euler-Pfade finden. Beispiele für Euler-Pfade im P-Graphen sind etwa: a-b-c-d, c-b-a-d oder c-a-b-d. In diesem Fall soll der Pfad c-a-b-d für das symbolische Layout zugrunde gelegt werden, da er auch im N-Graphen einen Euler-Pfad darstellt. Damit werden die Polysilizium-Gates und mithin die Transistoren im

symbolischen Layout in der Sequenz c-a-b-d von links nach rechts angeordnet. Das Resultat findet sich in Abbildung 10.4 (c). Dabei ist zu beachten, daß die Leitung n3 durch die N-Diffusion zwischen dem a- und dem b-Transistor gebildet wird.

10.2 Prozedurales Layout

Das Handlayout hat einen wesentlichen Nachteil: oftmals ist für eine Zelle im voraus bekannt, daß sie in verschiedenen Ausprägungen benötigt wird. Dies könnte zum einen die Technologie der Zelle betreffen. Andere Einstellungsmöglichkeiten, wie das W/L Verhältnis der Transistoren oder die Treiberfähigkeit einer Zelle, spielen ebenfalls eine Rolle. Die geschickte Handhabung von variierenden Zellen ist auch auf höheren Ebenen sinnvoll. So möchte man die Struktur eines ROM nur einmal entwerfen, aber den Speicherinhalt des ROM ohne großen Aufwand verändern können. Schließlich ist es ebenfalls wünschenswert, auch die Größe des ROM durch Parameter einstellen zu können. Diese Argumentation gilt auch für viele andere Blöcke, wie RAMs, PLAs, Datenpfade, Multiplizierer und viele mehr.

Um die verschiedenartigen Parameter berücksichtigen zu können, ist eine programmartige Struktur der Layoutbeschreibung erforderlich. Zusätzlich zu den typischen Konstrukten einer Programmiersprache werden bei den prozeduralen Layoutbeschreibungssprachen Befehle eingeführt, die die üblichen Layoutkonstrukte, wie Polygone, Texte oder Strukturen erzeugen. Im folgenden wird eine vereinfachte Beschreibung eines Inverters dargestellt. Sie stammt aus dem GDT-System¹ und berücksichtigt neben der Geometrie auch die elektrische Verbindungsstruktur. Mithin entsteht beim Synthesevorgang symbolisches Layout.

```
#####
#
# Inverter Zelle
#
# Parameter:
#   pdw - Breite des Pulldown-Transistors (Voreinstellung 6)
#   pdl - L"ange des Pulldown-Transistors (Voreinstellung 3)
#   puw - Breite des Pullup-Transistors (Voreinstellung 18)
#   pul - L"ange des Pullup-Transistors (Voreinstellung 3)
#
# Technologie: SCMOS
#
#####

L:: TECH scmos
CELL inv_G (pdw = 6, pdl = 3, puw = 18, pul = 3) {

# Platzierung der Transistoren

TN pulldown R90 W = pdw L = pdl AT (0, 0);
TP pullup R90 W = puw L = pul AT (0, 40);
```

¹Mentor Graphics Inc.

Platzierung und Verdrahtung der Kontaktflächen zum N-Aktivgebiet

```
MNDIFF pdcon_l R90 W=pdw AT (pulldown.X - MNDIFF_TNg - pdl/2, pulldown.Y);
MNDIFF pdcon_r R90 W=pdw AT (pulldown.X + MNDIFF_TNg + pdl/2, pulldown.Y);
WIRE NDIFF pulldown.d VER HOR pdcon_l;
WIRE NDIFF pulldown.s VER HOR pdcon_r;
```

Platzierung und Verdrahtung der Kontaktflächen zum P-Aktivgebiet

```
MPDIFF pucon_l R90 W=puw AT (pullup.X - MPDIFF_TPg - pul/2, pullup.Y);
MPDIFF pucon_r R90 W=puw AT (pullup.X + MPDIFF_TPg + pul/2, pullup.Y);
WIRE PDIFF pullup.d VER HOR pucon_l;
WIRE PDIFF pullup.s VER HOR pucon_r;
```

Platzierung und Verdrahtung des Wannenkontakts

```
NUM cwell_x = pullup.X - MNSUB_TPg - pul/2;
NUM cwell_y = pullup.Y + puw/2 + NDIFF1_PDIFF + MNSUB_NDIFF1_R;
MNSUB cwell AT (cwell_x, cwell_y);
WIRE MET1 pucon_l VER HOR cwell;
```

Platzierung und Verdrahtung des Substratkontakts

```
NUM csub_x = pulldown.X - MPSUB_TNg - pdl/2;
NUM csub_y = pulldown.Y - pdw/2 - NDIFF_PDIFF1 - MPSUB_PDIFF1_R;
MPSUB csub AT (csub_x, csub_y);
WIRE MET1 pdcon_l VER HOR csub;
```

Platzierung der N-Wanne

```
RECT NWELL (cwell.X - MNSUB_NSUB_R, cwell.Y + MNSUB_NSUB_R) TO
    (pucon_r.X + MPDIFF_NSUB_R, pullup.Y - pullup.NSUB.L/2);
```

Verdrahtung des Eingangsnetzes und Platzierung eines Anschlusses

```
WIRE POLY pulldown.gr VER HOR pullup.gl;
NUM offset = pdw/2 - MNDIFF_MET1_R;
NUM input_x = pdcon_r.X - MPOLY_MET1W;
NUM input_y = pulldown.Y + offset + MNDIFF_MPOLY;
MPOLY cin (input_x, input_y);
POLY cin HOR VER pulldown.gr;
IN MET1 input AT cin;
```

Verdrahtung des Ausgangsnetzes und Platzierung eines Anschlusses

```
OUT MET1 output (MAX(pucon_r.X, pdcon_r.X), input_y);
MET1 output VER HOR pdcon_r;
MET1 output VER HOR pucon_r;
```

VDD und GND

```
VDD MET1 vdd_l AT cwell;
VDD MET1 vdd_r AT (output.X, cwell.Y);
WIRE MET1 vdd_l HOR VER vdd_r;
GND MET1 gnd_l AT csub;
```

```
GND MET1 gnd_r AT (output.X, csub.Y);
WIRE MET1 gnd_l HOR VER gnd_r;
}
```

Ein Inverter ist noch eine recht einfache Schaltung. Im folgenden soll die Strategie zum Aufbau eines komplexeren Blocks - eines PLA - dargestellt werden. Damit ist die Realisierung der Grundstruktur mit UND- bzw. ODER-Ebene durch ein Layout gemeint, aber nicht der anwenderprogrammierbare PLA-Baustein aus dem zweiten Kapitel. In der UND-Stufe werden die Minterme aufbereitet und in der ODER-Stufe zusammengefaßt. Abbildung 10.5 zeigt die Funktionsweise eines PLA, wobei die Realisierung der AND- bzw. OR-Funktion durch zwei NOR-Stufen realisiert wird. Dabei geht man von invertierten Ein- und Ausgängen aus. Aus dem NOR mit invertierten Eingängen wird logisch ein AND. Ein NOR mit invertiertem Ausgang wird wieder zum OR. Die NOR-Funktion selbst wird wie folgt erzeugt: falls mindestens ein an eine Leitung angeschlossener NMOS-Transistor durchschaltet, wird die Leitung auf logisch 0 gezogen; in allen anderen Fällen wird sie durch die Pull-Up-Widerstände auf logisch 1 gebracht. Dies entspricht einer NOR-Funktion mit vielen Eingängen.

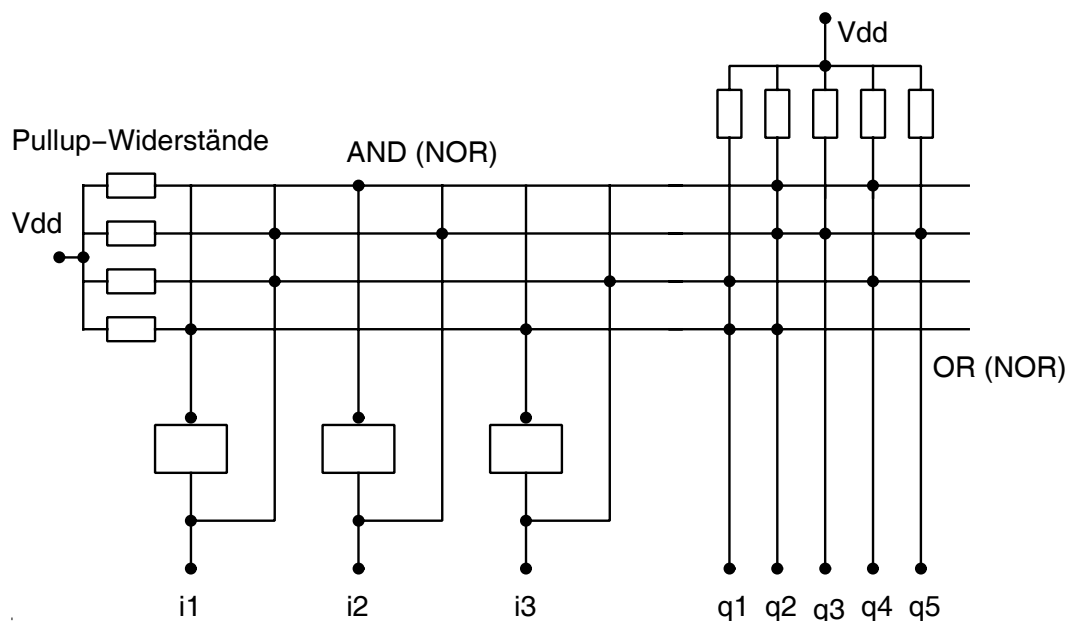


Abbildung 10.5: Grundschemata eines PLA.

Dieses recht einfache Schema läßt sich aus einigen, wenigen Komponenten zusammensetzen, die immer wieder auftauchen. Dies wird in Abbildung 10.6 dargestellt. Bei einer Parametrisierung der Anzahl der Eingangsvariablen, Minterme oder Funktionen dieses PLA muß man nur die jeweilige Anzahl der Eingangsspalten, Zeilen oder Ausgangsspalten variieren.

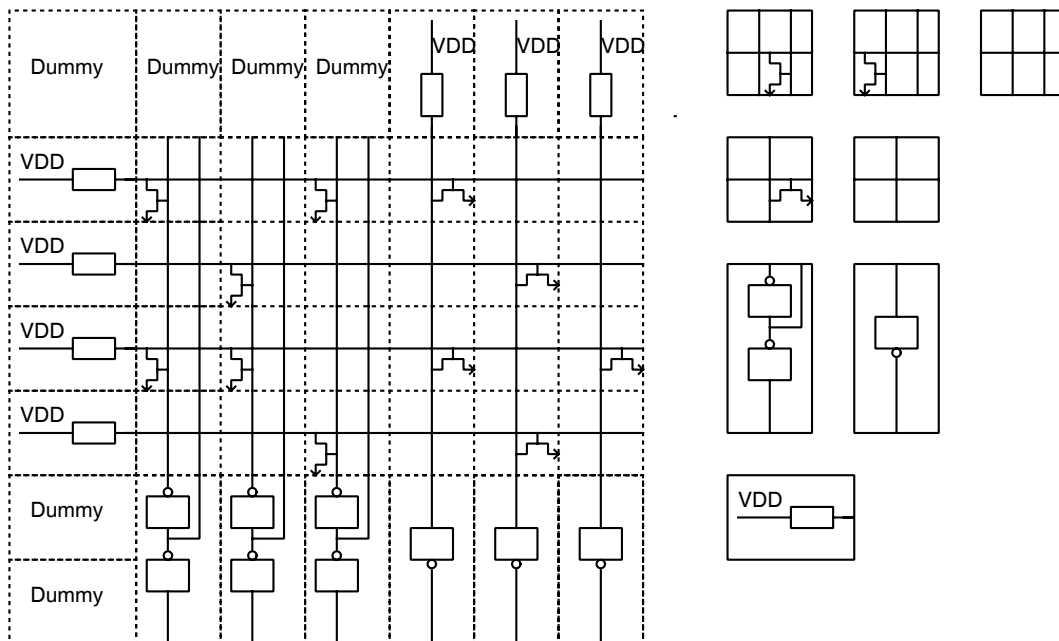


Abbildung 10.6: Strategie zum Aufbau eines parametrisierbaren PLA.

10.3 Hierarchisches Layout

Die wesentliche Frage bei der Synthese größerer Layouts ist die, wie man die Komplexität von Schaltung und Aufgabe in den Griff bekommt. Die Antwort lautet, wie bei vielen anderen komplexen Aufgaben: „Teile und Herrsche“. Wenn die Gesamtaufgabe in viele, mehr oder weniger unabhängig voneinander lösbare Teilaufgaben unterteilt werden kann, so vereinfacht sich das Problem wesentlich. Mehrere Aspekte spielen dabei eine Rolle: Zum einen kann eine Aufgabe funktional unterteilt werden. Beispielsweise wird die Layoutsynthese in die Platzierung (placement) von im vorhinein entworfenen Schaltungsteilen und deren Verdrahtung (routing) unterteilt. Damit lassen sich die zugehörigen Algorithmen wesentlich besser beherrschen. Neben der funktionalen Unterteilung erfolgt meist auch eine regionale Aufteilung. So wird bei der Layoutsynthese die zur Verdrahtung zur Verfügung stehende Fläche oftmals in Teilflächen unterteilt, die unabhängig voneinander verarbeitet werden. Zum einen reduziert dies die jeweils zu einem Zeitpunkt zu bearbeitenden Datenmengen. Da außerdem die meisten beteiligten Algorithmen eine Komplexität größer als linear aufweisen, ist die Bearbeitung vieler kleiner Probleme wesentlich effizienter durchzuführen, als die Bearbeitung der Summe der Teilprobleme.

Die Algorithmen zur Layoutsynthese werden vom Entwurstil wesentlich beeinflusst. Üblicherweise stehen hier folgende Realisierungsformen zur Verfügung: Standardzell-, Makrozell- und Gate-Array-Entwurf. Die Bausteine der anwender-programmierbaren Logik fallen nicht in diese Kategorie, da es sich hier um reine Standardschaltungen handelt, deren Layout festliegt und die erst durch eine geeignete Einstellung ihrer speichernden Elemente an die Aufgabe angepasst werden. Für FPGAs muß lediglich entschieden werden, welche Gatter an welchen Einbauplätzen realisiert werden und welche Signale welchen vorgefertigten Leitungen zugeordnet werden. Dies entspricht einem Spezialfall der Platzierung und Verdrahtung.

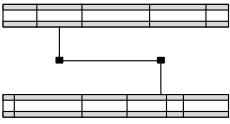
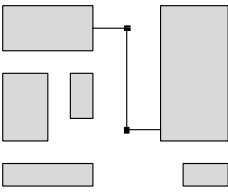
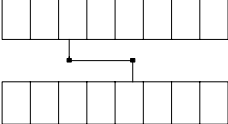
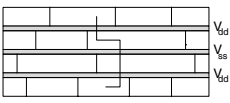
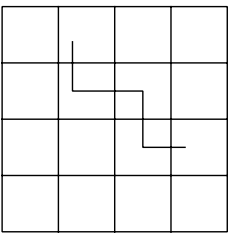
	Standardzellen	Makrozellen	Gate-Arrays
gesonderte Verdrahtungsgebiete			
Verdrahtung über die Zellen		—	

Tabelle 10.1: Anwendungen von Platzierung und Verdrahtung.

10.3.1 Platzierung und Verdrahtung

Beim Standardzellentwurf wird eine Schaltung aus diversen Teilschaltungen, den Zellen, zusammengesetzt, die jeweils eine primitive Logikfunktion darstellen. Alle Zellen weisen die gleiche Höhe auf; außerdem enthält jede Zelle Stücke von Versorgungsleitungen, die jeweils aneinandergesetzt die Versorgungsschienen bilden. Mehrere solcher Streifen werden zu einem Standardzellblock zusammengefasst, wobei sich zwischen den Streifen freie Fläche zur Führung der Verdrahtung befindet. Das Resultat wird in Abbildung 10.7 für eine Anordnung mit zwei Streifen dargestellt.

Die erste Phase besteht aus einer Platzierung, d.h. die Zellen werden in Streifen angeordnet. In der zweiten Phase wird die Verdrahtung zwischen den Zellen ausgeführt. Dazu stehen jeweils rechteckige Verdrahtungsflächen, die Kanäle, zur Verfügung. Nachdem zunächst entschieden wurde, welche Leitungen welche Kanäle benutzen (globale Verdrahtung, global routing), wird das Verdrahtungsproblem für jeden Kanal einzeln gelöst (lokale Verdrahtung, local routing), siehe Abbildung 10.8.

Beim Makrozellentwurf werden Zellen betrachtet, die beliebige Größe und einen beliebigen Umriß aufweisen. Diese Zellen können aus verschiedenen Quellen stammen. Denkbar sind hier: Standardzellblöcke, mit speziellen Generatoren erzeugte Blöcke, z.B. RAM, ROM, PLA, Datenpfad u.v.m., oder aus Makrozellen zusammengesetzte Makrozellen. Hier gliedert sich der Ablauf ebenfalls in Platzierung, globale Verdrahtung und lokale Verdrahtung, siehe Abbildung 10.9.

Beim Entwurf von Gate-Arrays liegt das Problem etwas anders. Drehte es sich bei den beiden vorangegangenen Entwurfsstilen um die Verwendung von im vorhinein entworfenen Schaltungsteilen, geht es nun um vorgefertigte Schaltungsteile. Dabei greift man auf sogenannte Master zurück, deren Funktion durch die noch fehlenden Metallisierungsebenen personalisiert wird. Dabei enthält ein solcher Master Tausende von vorgefertigten Gattern, die durch eine entsprechende Metallisierung

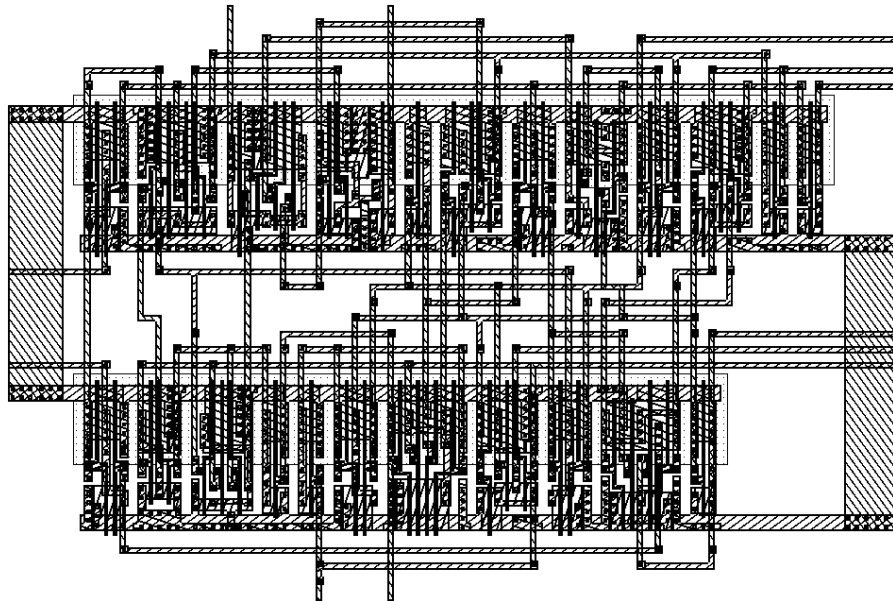


Abbildung 10.7: Streifenweise Anordnung von Standardzellen.

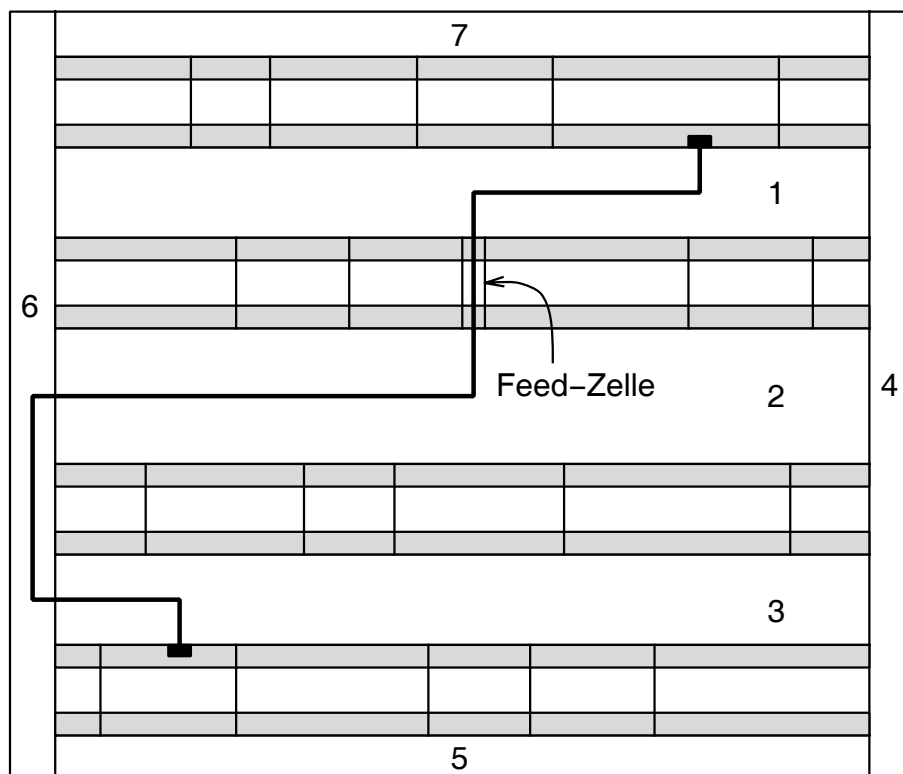


Abbildung 10.8: Platzierung und Verdrahtung von Standardzellen.

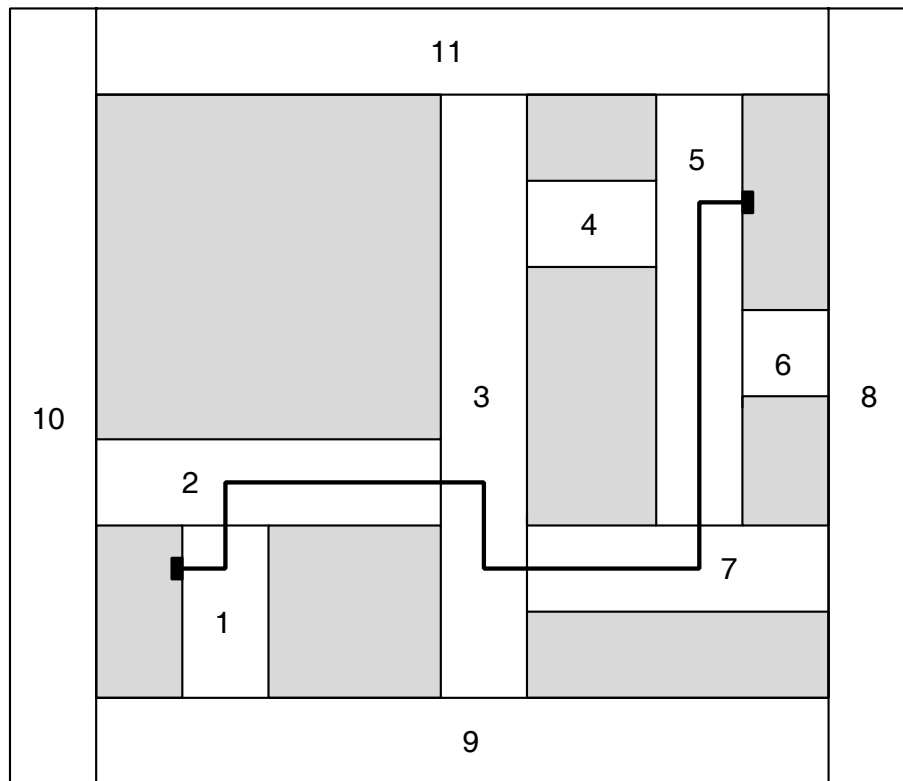


Abbildung 10.9: Platzierung und Verdrahtung von Makrozellen.

rung in die verschiedenen Grundgatter umgewandelt werden. Der Problematik bei dieser Entwurfsmethode liegt darin, daß diese Zellen nicht verschoben werden können. Deshalb muß bereits bei der Platzierung, d.h. bei der Zuweisung von Gattern in der Netzliste zu den Einbauplätzen auf dem Master, besonders auf die Verdrahtbarkeit geachtet werden. Aus diesem Grund wird oftmals ein guter Teil der Chipfläche verschenkt, nur weil die Verdrahtungsfläche bei einem kleineren Master an wenigen Stellen nicht ausreicht.

Nach diesen einleitenden Bemerkungen soll der Einsatz der Werkzeuge zur Layoutsynthese anhand eines typischer Entwurfsablaufs skizziert werden. Wenn die Schaltung vollständig als Schaltplan oder HDL-Beschreibung entworfen und simuliert wurde, macht man sich Gedanken darüber, wie die Schaltung sinnvoll aufgeteilt werden kann (Partitionierung), da in der Regel die verschiedensten Realisierungsmöglichkeiten zur Verfügung stehen. Diese sind aber jeweils nur für eine begrenzte Gruppe von Schaltungen sinnvoll anwendbar. So kann man zwar ein RAM auch aus Gattern bauen; dies ist jedoch in aller Regel nicht sinnvoll, da die reguläre RAM Struktur sehr günstig durch einen speziellen Generator aufzubauen ist. Das weitere Vorgehen hängt davon ab, ob der Entwurfsvorgang Top-down oder Bottom-up verläuft.

Beim Top-down-Entwurf werden die im Rahmen der Partitionierung ausgesuchten Blöcke als schwarze Kisten betrachtet, von denen lediglich Flächen- und Umrißschätzungen und die Anschlüsse bekannt sind. Daraus ergeben sich Abweichungen, die später korrigiert werden müssen. Insbesondere steht hier aber auch eine große Zahl von Alternativen zur Verfügung, die dazu eingesetzt werden können, insgesamt ein besseres Ergebnis zu erzielen. Wenn eine Teilschaltung bei-

spielsweise durch einen Standardzellenblock realisiert werden soll, so ergeben sich Wahlmöglichkeiten bezüglich der Anzahl der Zellreihen. Wenn nun ein solcher Block zur Erzielung eines besonders kompakten Gesamtlayouts eine längliche Form aufweisen müßte, so verringert man einfach solange die Anzahl der Streifen, bis ihre wachsende Breite den gewünschten Umriß erzeugt. Alternativ kann man auch die Anzahl der Streifen erhöhen, so daß viele kurze Streifen übereinander angeordnet werden. Ein RAM oder ROM kann ebenfalls in seinem Umriß verändert werden, z.B. durch eine Anordnung in zwei flachen oder einer hohen Bank. Auf ähnliche Art lassen sich auch die Umrisse von Blöcken aus anderen Quellen variieren. Somit muß ein Optimierungsproblem mit einer Unzahl von Freiheitsgraden gelöst werden. Eine optimale Lösung ist in aller Regel nicht zu erzielen; annehmbare Lösungen sind jedoch schon erreichbar. Diese Art der Anordnung von Zellen in der Ebene wird Floorplanning genannt. Nach dem Floorplanning, werden die Verbindungen zwischen den Blöcken geführt. Dies wird Verdrahtung oder auch Routing genannt. Anschließend nimmt man die Layoutgenerierung für die einzelnen Blöcke in Angriff. Dabei sollen die Vorgaben aus dem Floorplan möglichst genau berücksichtigt werden. Falls es sich bei dem Block um einen Standardzellenblock handelt, wird durch Platzierung und Verdrahtung ein Layout generiert, falls es sich um einen Block handelt, der mit prozeduralem Layout zu generieren ist, wird der entsprechende Generator aufgerufen. Nachdem die Layoutsynthese für alle Blöcke durchgeführt wurde, wird der Floorplan selbst auf die nun genau bekannten Umrisse angepaßt. Die sich dabei ergebenden Änderungen sollten allenfalls marginale Auswirkungen haben.

Falls der Entwurf jedoch Bottom-up verläuft, liegt das Problem etwas anders. Hier stehen die Zellen bereits im vorhinein fest, so daß keinerlei Variationen für ein kompakteres Layout ausgenutzt werden können. Die Anordnung von Standardzellen ist ein typisches Problem aus dieser Kategorie. Standardzellen sind festgelegt und können lediglich in ihrer Position verändert werden. Die Lösung dieses Problems wird auch Platzierung genannt. Die Platzierung ist somit in gewissem Sinn ein Spezialfall des Floorplanning. Nach der Platzierung wird auch hier eine Verdrahtung durchgeführt. Damit ist die Layoutgenerierung für diesen Block abgeschlossen. Nun kann dieser Block seinerseits in anderen Blöcken platziert und an andere Blöcke angeschlossen werden.

Der letzte Punkt in beiden Entwurfsabläufen besteht darin, herauszufinden, ob die Layoutgenerierung fehlerfrei ablief und die parasitären Effekte die Funktion der Schaltung nicht negativ beeinflussen. Zu diesem Zweck werden die generierten Schaltungen auf die Einhaltung der Entwurfsregeln überprüft. Zur Kontrolle des Timing der Schaltung unter Berücksichtigung der Leitungsverzögerungen kann man sich die Leitungsverzögerungen aus der Layoutgeometrie bestimmen lassen und diese in einer Logiksimulation berücksichtigen. Solche Werkzeuge heißen auch Delay Calculator. Falls sich keine nennenswerten Unterschiede zu den Simulationsergebnissen ohne Betrachtung der parasitären Verzögerungen ergeben haben, kann die Schaltung zur Fertigung freigegeben werden.

10.3.2 Platzierung

Wie im vorigen Abschnitt angedeutet, gliedert sich die hier beschriebene automatische Erzeugung von Layout in zwei Phasen: Platzierung respektive Floorplanning und Verdrahtung. Platzierung und Floorplanning basieren auf denselben Grundlagen. Die Darstellung der Algorithmen soll hier auf den Bereich der Platzierung beschränkt bleiben.

Die Platzierung wird durch den Stil der Layoutsynthese wesentlich beeinflusst. Falls es sich um einen Standardzellentwurf handelt, werden die Zellen gleicher Höhe streifenweise angeordnet, wobei sich die Verdrahtungsfläche zwischen den Streifen befindet. Bei einer Makrozellplatzierung werden beliebige Zellen in der Ebene angeordnet. Für die Platzierung von Gate-Arrays schließlich werden die Gatter den festgelegten Einbauplätzen zugeordnet. Im Rahmen der Platzierung soll vor allem darauf geachtet werden, daß die Verdrahtung möglichst einfach und damit flächengünstig ausgelegt werden kann. Für Gate-Arrays kommt hinzu, daß die Verdrahtbarkeit garantiert werden muß. Hier können die Zellen nicht mehr verschoben werden, um einer Überfüllung bestimmter Verdrahtungsbereiche zu begegnen. Das Platzierungsproblem ist ein Optimierungsproblem, bei dem die Berechnung einer optimalen Lösung einen exponentiellen Aufwand erfordert. Damit wird eine gute, wenn auch nicht optimale Lösung gesucht.

Im folgenden soll vor allem die Makrozellplatzierung betrachtet werden. Dieses Problem hat im wesentlichen zwei Facetten: zum einen soll eine Menge von Rechtecken oder anderen geometrischen Formen überlappungsfrei in der Ebene platziert werden, zum anderen soll der Aufwand für die Verbindungen minimiert werden. Problematisch dabei ist, daß der Flächenbedarf für die Verdrahtung nur ungenau abgeschätzt werden kann, so daß zunächst ausreichend Platz einkalkuliert wird. Nach der eigentlichen Platzierung und Verdrahtung wird dann eine Kompaktierung durchgeführt. Im folgenden wird der Min-Cut Algorithmus beschrieben, der eine Standardlösung des Platzierungsproblems darstellt, Abbildung 10.10.

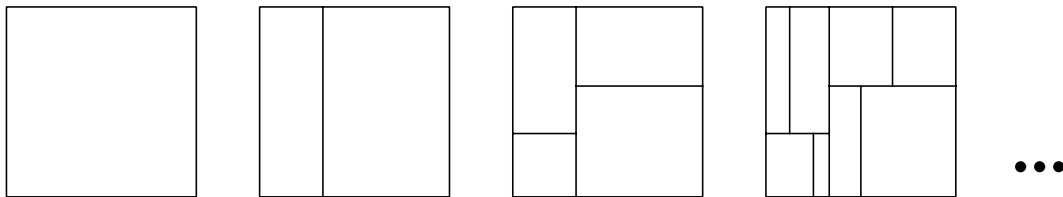


Abbildung 10.10: Makrozellplatzierung im Min-Cut-Verfahren.

Der Min-Cut Algorithmus ist ein iteratives Verfahren, bei dem eine Menge von Zellen in jeweils zwei Hälften geteilt wird. Die eine Hälfte der Zellen wird links (oberhalb) einer gedachten Schnittlinie platziert, die andere Hälfte rechts (unterhalb). Anschließend wird jede Hälfte ihrerseits unterteilt bis sich in jeder Menge nur noch eine Zelle befindet. Die Minimierung des Verdrahtungsaufwands wird dadurch erreicht, daß die Anzahl der durch einen Schnitt betroffenen Verbindungen minimiert wird. Eng verkoppelte Schaltungsteile tendieren dann dazu, beieinander zu bleiben. Dies ist gerade der Effekt, den man erreichen möchte. Ein Beispiel soll den Ablauf des Min-Cut Algorithmus verdeutlichen. In der folgenden Sequenz können zusätzlich zur Platzierung einige Optimierungen angewendet werden. Blöcke können gedreht oder gespiegelt eingebaut werden.

In einigen Fällen weist der Algorithmus aber auch Nachteile auf. Manche gute Platzierungslösungen lassen sich nicht durch den Min-Cut Algorithmus erzeugen, Abbildung 10.11. Dies ergibt sich daraus, daß ein Schnitt beim Min-Cut immer eine Gerade ist.

Die Abbildung 10.12 zeigt eine Min-Cut- und eine Zufalls-Platzierung im Vergleich, wobei die eingezeichneten Linien die „Luftlinien“ der zu verdrahtenden Verbindungen beschreiben. Es fällt

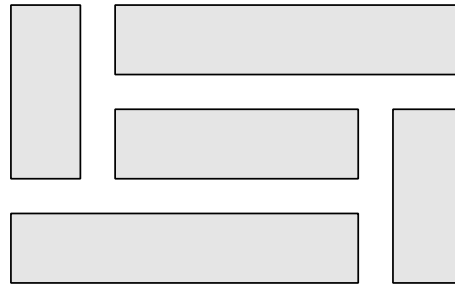


Abbildung 10.11: Platzierung, die nicht durch den Min-Cut Algorithmus erreichbar ist.

auf, daß die Verdrahtung bei der Min-Cut-Platzierung deutlich lokaler verläuft, was große Vorteile bei der Verdrahtung mit sich bringt.

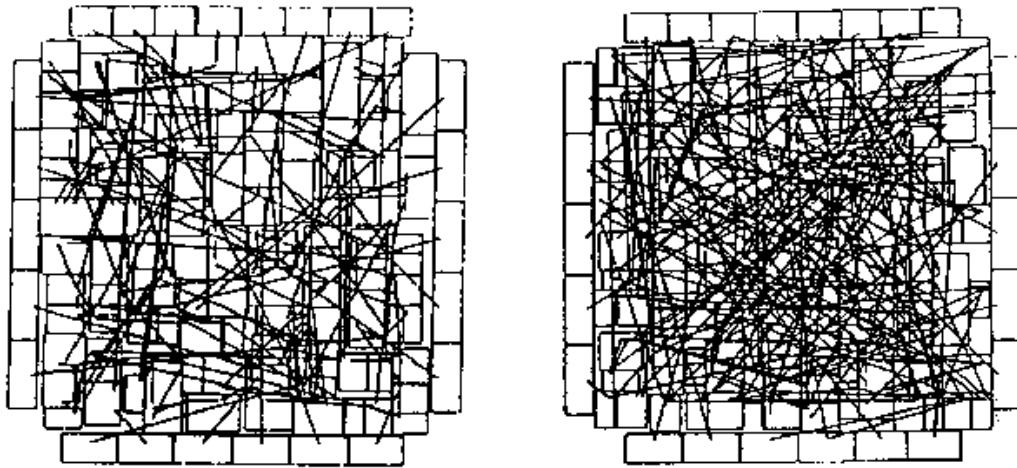


Abbildung 10.12: Vergleich: Min-Cut Platzierung (links), Zufallsplatzierung (rechts).

10.3.3 Verdrahtung

Bevor die eigentliche Verdrahtung stattfindet, werden zunächst die Bereiche bezeichnet, die zur Verdrahtung verwendet werden können. Da das Verdrahtungsproblem sehr komplex ist und moderne Schaltungen hunderttausende von Signalen aufweisen, kann die Verdrahtung nicht in einem Zug erledigt werden. Vielmehr wird die Verdrahtungsfläche in kleinere Regionen aufgeteilt, die jeweils unabhängig zu bearbeiten sind. Die gewählte Verdrahtungsmethode hängt dabei sehr von der Gestaltung dieser Teilflächen ab.

Die eigentliche Verdrahtung einer Schaltung unterteilt sich in die globale und lokale Verdrahtung. Bei der globalen Verdrahtung wird entschieden, durch welche Teilflächen die einzelnen Signale zu ziehen sind. Dabei sollte vor allem darauf geachtet werden, daß alle Teilflächen gleichmäßig ausgenutzt werden und sich keine Überfüllung ergibt. Insbesondere in der Chipmitte entstehen

oftmals überfüllte Teilflächen, die das Synthesegesamtergebnis negativ beeinflussen. Die lokale Verdrahtung wird in den meisten Fällen durch Channel-Routing oder Maze-Routing erzeugt.

10.3.4 Channel Routing

Channel Routing wird für Standardzell-, Makrozell- und Gate-Array-Entwürfe verwendet, bei denen gesonderte Verdrahtungsgebiete zur Verfügung stehen. Diese werden Kanäle genannt und sind bestimmten Beschränkungen unterworfen, um die beteiligten Algorithmen möglichst einfach zu halten. Ein Kanal repräsentiert eine längliche, oftmals rechteckige, freie Fläche, die oben und unten Anschlüsse aufweist. Diesen Anschlüssen sind jeweils Signale zugeordnet. Damit wird definiert, welche Anschlüsse miteinander zu verbinden sind. Im Gegensatz zu den festen Anschlüssen oben und unten werden an den Kanalgrenzen links und rechts verschiebbare Anschlüsse angebracht, die für Signale gedacht sind, die sich über mehrere Kanäle erstrecken.

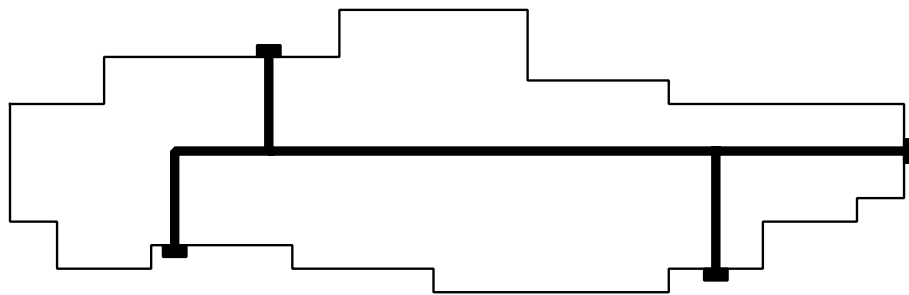


Abbildung 10.13: Kanalmodell.

Die wichtigste Eigenschaft eines Kanals ist, daß er nach links und rechts konvex ist und mithin keine Überhänge aufweist, siehe Abbildung 10.14. Dieses Merkmal vereinfacht das Problem wesentlich, so daß die zum Channel Routing eingesetzten Algorithmen in der Regel wesentlich effizienter arbeiten als die Maze-Algorithmen, die beliebige Verdrahtungsflächen verdrahten können.

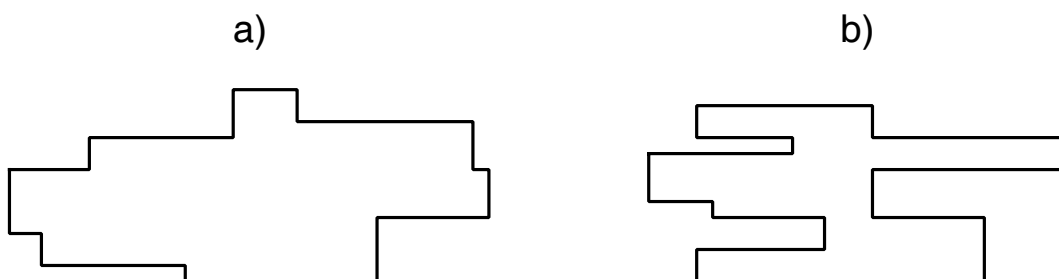


Abbildung 10.14: a) korrekter Kanal, b) inkorrekter Kanal.

Üblicherweise stehen zur Verdrahtung mindestens zwei leitende Ebenen zur Verfügung, z.B. Polysilizium, Metall, so daß in jedem Fall Überkreuzungen verschiedener Signale möglich sind, ohne

daß ein Kurzschluß entsteht. Manchmal kann für ein Verdrahtungsproblem eine planare, also überkreuzungsfreie Lösung garantiert werden, so daß eine einzige Verdrahtungsebene ausreichend ist. Dieser Spezialfall wird River Routing genannt.

Üblicherweise wird das Kanalverdrahtungsproblem dadurch vereinfacht, daß Segmente von Leitungen nur senkrecht oder waagerecht verlaufen dürfen (manhattan routing). Falls beliebige Schrägen erlaubt sind, lassen sich zwar etwas kompaktere Lösungen erreichen; der Aufwand steigt jedoch immens. Weiterhin wird oftmals die Vereinbarung getroffen, daß horizontale Segmente einer Leitung grundsätzlich in einer Ebene, z.B. Metall, geführt werden und senkrechte Segmente in einer anderen, z.B. Polysilizium. Die Verbindungen zwischen beiden Ebenen werden - wo benötigt - durch die Kontaktlöcher hergestellt. Um nun eine Lösung für das Verdrahtungsproblem zu erreichen, beschränkt man sich vielfach darauf, zunächst nur die waagerechten Leitungssegmente zu betrachten, da sich die senkrechten danach automatisch ergeben. Man formuliert zunächst eine Reihe von Bedingungen (constraints), deren Einhaltung unbeabsichtigte Kurzschlüsse sicher vermeidet. Zum einen sollen horizontale Segmente, die eine Überlappung in x-Richtung aufweisen, einen Mindestabstand in y-Richtung erhalten. Zum anderen müssen die senkrechten Segmente sicher nach oben und unten geführt werden können, woraus sich Bedingungen ergeben, nach denen ein Segment immer oberhalb (in y-Richtung) eines anderen zu liegen kommen muß. Dies wird in der folgenden Abbildung 10.15 erläutert.

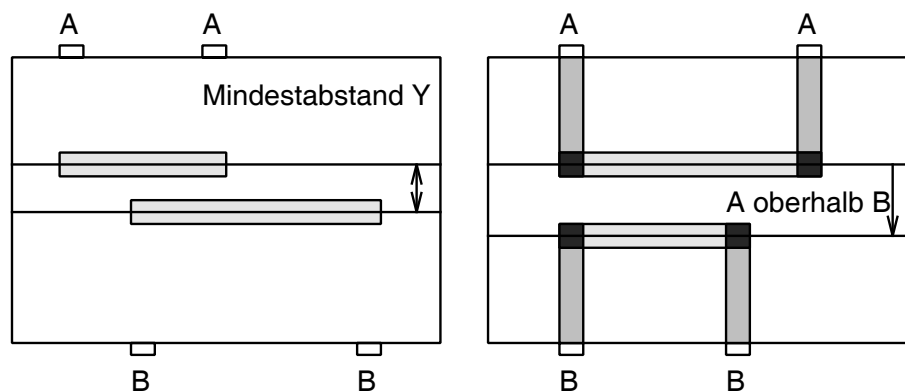


Abbildung 10.15: Bedingungen für die Platzierung waagerechter Leitungssegmente.

Im folgenden wird der Left-Edge-Algorithmus zur Kanalverdrahtung beschrieben. Die Vorbereitungen zur Anwendung des Left-Edge-Algorithmus bestehen darin, sämtliche Bedingungen zu ermitteln und in eine Datenstruktur (constraint graph) einzutragen. Danach soll zunächst nur eine Rangfolge festgelegt werden, welche Segmente oberhalb und unterhalb eines jeden Segments liegen. Später werden den waagerechten Segmenten die genauen y-Koordinaten zugeordnet und die Stichleitungen zu den Kanalgrenzen gezogen. Bei dieser Methode werden die waagerechten Segmente zeilenweise von oben nach unten angeordnet. Dabei wird jeweils das Segment gewählt, das keine Bedingungen verletzt und dabei in der aktuellen Zeile möglichst wenig Platz zu seinem Vorgänger läßt. Falls es hierbei noch Wahlmöglichkeiten gibt, wird das Segment mit der größten Ausdehnung in x-Richtung zuerst ausgewählt. Das zeilenweise Vorgehen dient dazu, die überlappungsfreie Platzierung der waagerechten Segmente automatisch zu gewährleisten. Damit bleibt

jeweils nur noch zu beachten, daß die Bedingungen vom zweiten Typ eingehalten werden. In der folgenden Sequenz wird der Algorithmus anhand eines Beispielkanals, Abbildung 10.16 a) erläutert. Die Abbildung 10.16 b) zeigt die horizontalen Segmente und die zwischen ihnen geltenden Bedingungen. In der Abbildung 10.16 c) erfolgt die zeilenweise Aufteilung der Segmente unter Beachtung der Bedingungen. Die Abbildung 10.16 d) schließlich zeigt das Ergebnis der Kanalverdrahtung.

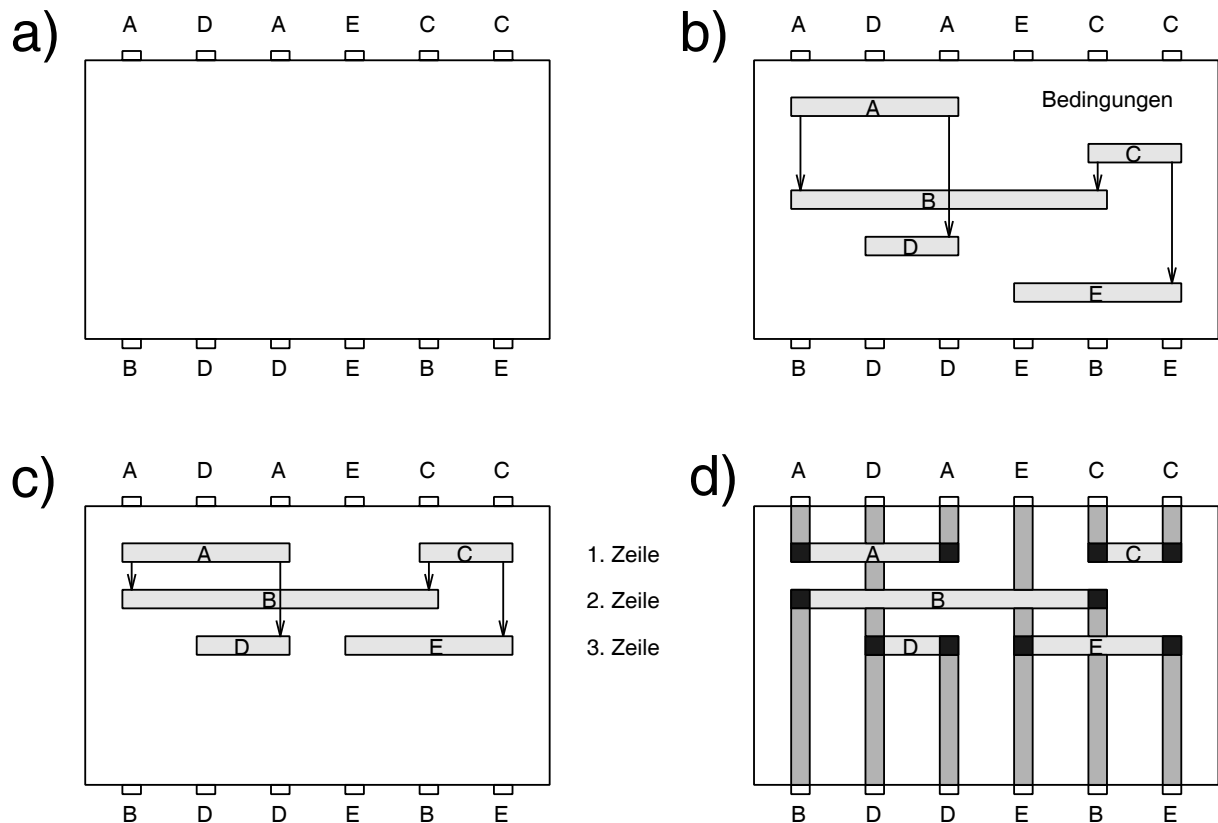


Abbildung 10.16: a) Beispielkanal, b) Waagerechte Segmente, c) Aufteilung der waagerechten Elemente, d) Ergebnis der Kanalverdrahtung.

10.3.5 Maze-Routing

Beim Maze Routing weist die Verdrahtungsfläche eine beliebige Form auf (Abbildung 10.18), so daß auch Entwürfe verdrahtet werden können, bei denen die Verbindungen über die Zellen hinweg gezogen werden. Beliebte Beispiele für die Leistungsfähigkeit von Maze-Routern sind die Grundrisse begehrter Labyrinth. Die gewünschte Verbindung wird durch die Namensattribute der Anschlüsse festgelegt.

Zumeist wird die Verdrahtungsfläche als Ebene von Rasterpunkten betrachtet. Im einfachsten Fall wird, ausgehend von einem Anschluß, eine „Wellenfront“ ausgeschickt. Dies bedeutet, daß im

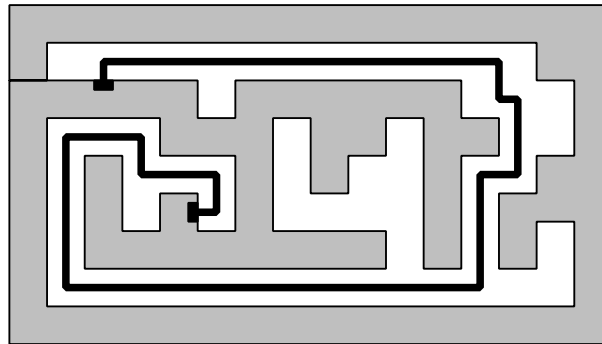


Abbildung 10.17: Maze Routing.

jeweils nächsten Schritt diejenigen Rasterpunkte hinzukommen, die an bereits betrachtete Punkte angrenzen. Abbildung 10.18 zeigt die entstehenden Wellenfronten.

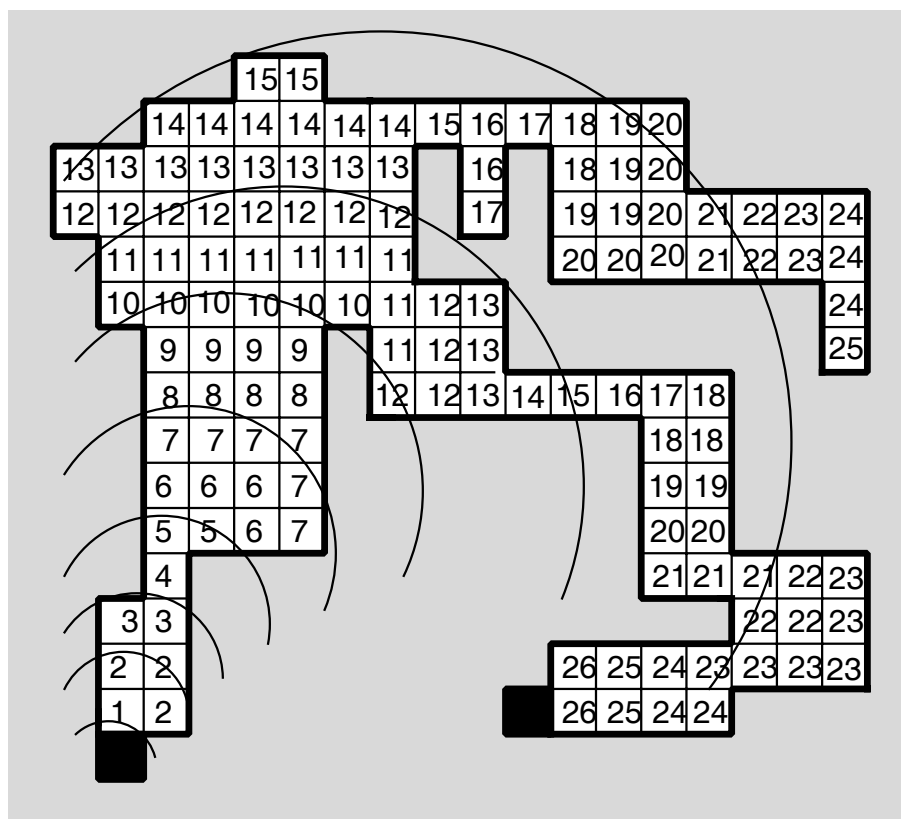


Abbildung 10.18: Wellenfronten beim Maze Routing.

Wenn man mit einem solchen Punkt auf einen Anschluß trifft, der mit dem ersten Anschluß verbunden werden soll - mithin also gesuchte Namensattribut aufweist - kann ohne große Schwierigkeiten ein Weg zwischen den beiden Anschlüssen gefunden und eine Leitung gezogen werden. Zu diesem

Zweck geht man von dem Anschluß aus, der mit der n -ten Wellenfront aufgefunden worden war. Benachbart zu diesem Anschluß ist mindestens ein Rasterpunkt, der der Wellenfront $n-1$ zuzurechnen ist. Für diesen muß wiederum ein Rasterpunkt existieren, der aus der Wellenfront $n-2$ stammt. Damit läßt sich ein in der Länge minimaler Weg zum ersten Anschluß finden. Der dabei beanspruchte Platz steht allerdings den folgenden Leitungen nicht mehr zur Verfügung. Daraus folgt, daß das Verdrahtungsergebnis von der Reihenfolge abhängt, nach der die Anschlüsse betrachtet werden. Demzufolge müssen manchmal Leitungen wieder entfernt und anderweitig verlegt werden, um ein gutes Verdrahtungsergebnis zu erzielen. Diese Technik ist unter dem Namen „rip up and reroute“ bekannt.

Dieser Algorithmus zum Maze-Routing wird auch nach seinem Erfinder Lee-Algorithmus genannt. Es hat den Vorteil, daß er leicht zu implementieren ist. Nachteilig an diesem Algorithmus - wie überhaupt am Maze-Routing - ist allerdings der hohe Laufzeit- und Speicherbedarf, da die inhärent notwendige Rasterrepräsentierung sehr aufwendig ist. Erschwerend kommt hinzu, daß mehrere Ebenen, z.B. Polysilizium, Metall1, Metall2, gleichzeitig repräsentiert werden müssen. Außerdem ergeben sich weitere Probleme, wenn das Verdrahtungsgebiet nicht ausschließlich durch senkrechte und waagerechte Segmente begrenzt ist.

In manchen Ansätzen wird eine Wellenfront von beiden Anschlüssen ausgesendet. Wenn es überhaupt eine Verdrahtungsmöglichkeit gibt, dann treffen sich die beiden Wellenfronten irgendwann. Dieser Ansatz hat gegenüber der ursprünglichen Methode den Vorteil, daß hier nur zweimal die halbe Anzahl der Wellenfronten und damit wesentlich weniger Pixel betrachtet werden müssen.

10.4 Layoutkompaktierung

Wie bereits in der Einleitung zu diesem Kapitel erwähnt, ist ein wesentliches Gütekriterium für ein Layout, ob per Hand oder automatisch erzeugt, die von ihm eingenommene Fläche. Um diese Fläche zu minimieren, wird üblicherweise ein Verfahren eingesetzt, das sich *Kompaktierung* nennt. Dieses Verfahren beruht auf geometrischen Operationen, berücksichtigt aber die elektrischen und physikalischen Vorgaben des Entwurfs. Die Voraussetzung zur Kompaktierung ist ein symbolisches Layout, das heißt eine Repräsentation und Einbeziehung der Netzliste. Diese wird von allen beschriebenen Layoutsynthese-Verfahren erzeugt, könnte aber im Prinzip auch durch eine Extraktion des reinen Layouts gewonnen werden, ist also immer verfügbar.

Der Ablauf einer Kompaktierung läßt sich wie folgt beschreiben. Symbole der untersten Ebene, d.h. die Blätter im Hierarchiebaum, haben gemäß ihrer Parameter eine Darstellung in physikalischem Layout. Für einen Transistor könnte ein solcher Parameter das W/L-Verhältnis sein. Diese Layoutstückchen werden für die Kompaktierung als invariant angesehen. Anders die Verbindungen zwischen Symbolen, die im wesentlichen aus Metall- und Polybahnen, sowie den zugehörigen Kontakten bestehen. Diese Leitungen können mehr oder weniger beliebig „gebogen“ werden, um die gewünschten Verbindungen überlappungsfrei und entsprechend der Entwurfsregeln herzustellen. Aufgabe der Kompaktierung ist es nun, die genauen Koordinaten der Schaltungselemente und Leitungen unter Einhaltung der Designregeln so festzulegen, daß die Fläche des Entwurfs minimiert wird, siehe Abbildung 10.19.

Die Chipflächenminimierung stellt aber nicht die einzige Einsatzmöglichkeit eines Kompaktors dar. Ein Kompaktor kann auch zur automatischen Korrektur von Entwurfsregelverletzungen ver-

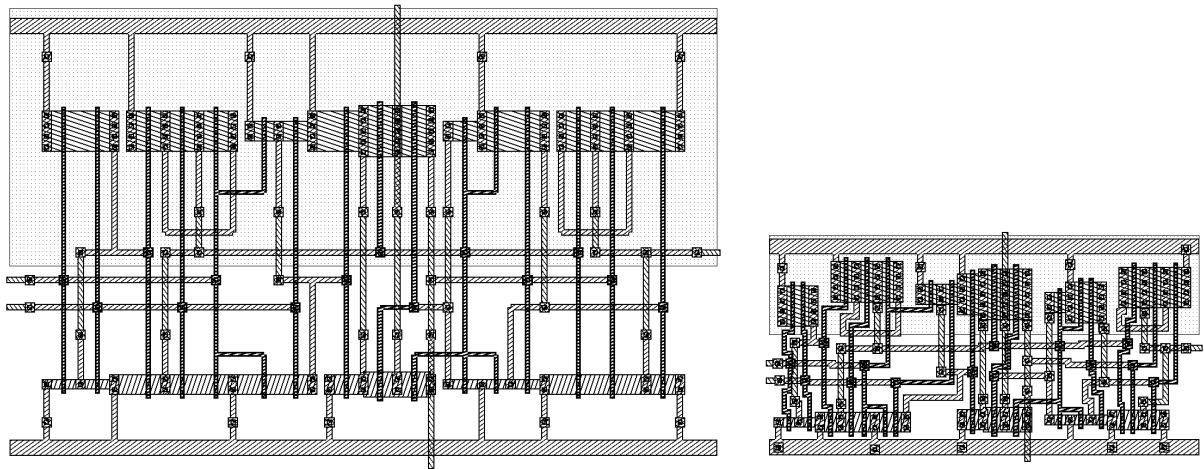


Abbildung 10.19: Layoutkompaktierung.

wendet werden. Bei der Neuberechnung der Element- und Leitungskoordinaten einer fehlerhaften Schaltung wird die Einhaltung der Designregeln automatisch gewährleistet. Schließlich kommt es vor, daß bestehende Schaltungen mit einer neuen Technologie gefertigt werden sollen. Hier reicht eine einfache Skalierung in der Regel nicht aus, um die Einhaltung der neuen Designregeln sicherzustellen. Auch hier kann ein Kompaktor eingesetzt werden, um eine solche Schaltung automatisch anzupassen.

Layoutkompaktierung ist ein zweidimensionales Problem, d.h. sowohl die x- als auch die y-Koordinaten der Elemente und Leitungsstücke sollen verändert werden. Eine wesentliche Vereinfachung ergibt sich, wenn man die Aufgabe in zwei eindimensionale Einzelprobleme zerlegt. Im ersten Schritt erfolgt die Kompaktierung in y-Richtung, d.h. die y-Werte variieren, während die x-Werte konstant bleiben. Im zweiten Schritt folgt die Kompaktierung in x-Richtung. Man kann natürlich auch zuerst in x-Richtung und dann in y-Richtung kompaktieren. Die Zerlegung der Aufgabe erleichtert die Implementierung wesentlich und führt zu um Größenordnungen geringeren Programmlaufzeiten. Der Nachteil dieser Methode liegt jedoch darin, daß das kompaktierte Layout nicht die minimal mögliche Ausdehnung aufweist.

Bei der Kompaktierung einer Schaltung, wie bei jeder Optimierungsaufgabe, sind einige Randbedingungen einzuhalten, die den Erfolg der Operation sicherstellen. Dies sind zunächst die bereits oben erwähnten Designregeln. Weiterhin muß gewährleistet sein, daß die Verbindungsstruktur der Schaltung erhalten bleibt. Schließlich sind auch Benutzerwünsche zu berücksichtigen. Diese bestehen beispielsweise in gewünschten maximalen Leitungslängen bestimmter Verbindungen, um die Verzögerung zeitkritischer Signale zu begrenzen.

Es ist nun wünschenswert, diese Randbedingungen einheitlich zu modellieren. Hierzu wird oft ein System von Gleichungen und Ungleichungen gewählt. Jede einzelne (Un)gleichung wird als Bedingung (constraint), also als Einschränkung für die Schaltung bezeichnet. An dieser Stelle soll zunächst die Kompaktierung in einer Richtung (z.B. in y-Richtung) beschrieben werden. Hierbei werden alle x-Koordinaten festgehalten, die y-Koordinaten können sich ändern. Im folgenden soll nun beispielhaft gezeigt werden, wie die erwähnten drei Typen von Randbedingungen dargestellt werden können:

1. Designregel für den Mindestabstand zweier (gerader) Metallbahnstücke M1 und M2, falls sich deren x-Bereiche überschneiden (Abbildung 10.20).

$$y(M1) - y(M2) \geq d$$

2. Anschluß von Leitung L an Kontakt K (Abbildung 10.20)

$$|y(L) - y(K)| \leq d \text{ oder alternativ } y(L) = y(K)$$

3. maximale Leitungslänge zwischen Anschluß A1 und A2 (Abbildung 10.20)

$$y(B1) - y(B2) \leq d$$

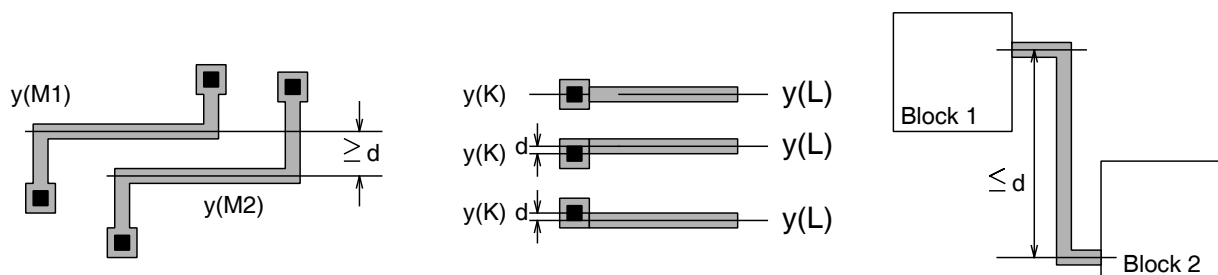


Abbildung 10.20: Bedingungen für die Kompaktierung.

Mit den für eine Schaltung generierten (Un)gleichungen ergibt sich ein (Un)gleichungssystem. Das gewünschte Kompaktierungsergebnis entspricht immer einer Lösung des (Un)gleichungssystems. Die Existenz einer Lösung ist jedoch nicht garantiert, da beispielsweise strenge Benutzerbedingungen des dritten Typs unter Einhaltung der Topologie nicht immer mit den Designregeln in Einklang zu bringen sind. Layoutelemente mit unerfüllbaren Bedingungen müssen erkannt und markiert werden.

Das Geflecht der Bedingungen für ein komplettes Layout kann mathematisch durch einen Bedingungsgraphen angegeben werden. Die einfachste Variante dieses Graphen gibt nur die Designregelbedingungen wieder. In diesem gerichteten Graphen repräsentieren die Knoten die Layoutelemente und Leitungsstücke. Eine Kante führt von Knoten A zu Knoten B, falls sich die zugehörigen x-Bereiche überschneiden und A topologisch über B liegt ($y(A) > y(B)$). Dieser Graph muß grundsätzlich zyklensfrei sein. Ein Weg maximaler Länge in diesem Graphen wird „kritischer Pfad“ genannt. Die „Länge“ des kritischen Pfades entspricht der durch Kompaktierung erreichbaren minimalen Höhe des Layouts. Die y-Koordinaten der Elemente im kritischen Pfad liegen fest, während alle anderen Elemente einen Spielraum besitzen. Innerhalb dieses Spielraums können sie verschoben werden, ohne daß sich die Höhe des Layout vergrößert. Dieser Freiheitsgrad sollte genutzt werden, um die Leitungslängen und die Anzahl der Knicke (jogs, doglegs) in Leitungen zu minimieren. Die Ausnutzung dieses Freiheitsgrades ist gerade auch bei den sogenannten Spacerprogrammen von entscheidender Bedeutung. Bei diesen wird nur bis zu einer bestimmten gewünschten Größe kompaktiert, die in der Regel über der minimal möglichen liegt. Die Spielräume der einzelnen Elemente sind entsprechend größer.

Anhang A

Schaltungsentwurf mit VHDL

A.1 Einführung

Die hier gegebene Darstellung von VHDL kann nur eine grobe Übersicht bieten und ersetzt damit **nicht** die Lektüre eines VHDL-Buchs.

VHDL ist eine Hardware-Beschreibungssprache im klassischen Sinn. Sie ist der ADA-Programmiersprache nachempfunden und enthält viele Konstrukte einer prozeduralen Programmiersprache. Daneben gibt es eine ganze Reihe von Kommandos zur Formulierung von nebenläufigen Ereignissen und deren Timing. Gerade die Möglichkeit auch parallel ablaufende Vorgänge zu modellieren, ist für Beschreibung von Schaltungen unabdingbar. Schließlich gilt für jede Komponente einer Schaltung, daß sie unabhängig von den anderen Komponenten ihre Eingänge auswertet und ihre Ausgänge stellt.

Die Simulation von VHDL-Beschreibungen ermöglicht deren schnelle Verifizierung. Simulationen sind auf allen von VHDL unterstützten Abstraktionsebenen möglich.

Weiterhin kann VHDL auch im Rahmen der Spezifikation eingesetzt werden. Hier steht die Modellierung des gewünschten Verhaltens im Vordergrund. Eine in VHDL formulierte Spezifikation ist formal beschrieben und außerdem simulierbar, was deren Interpretation wesentlich erleichtert. Damit ergibt sich ein Einstieg in einen integrierten Entwurfsablauf, da die Systementwicklung bis auf die Gatterebene ebenfalls mit VHDL stattfinden kann. Entwurfsschritte bilden damit VHDL-Beschreibungen auf detailliertere VHDL-Beschreibungen ab.

Mithin ist VHDL eine Programmiersprache, eine Simulationssprache und eine Dokumentations- bzw. Spezifikationssprache. VHDL ist als IEEE-Standard weithin akzeptiert. Simulations- und Synthese-Werkzeuge der verschiedensten Anbieter setzen auf VHDL auf.

VHDL kann, wie bereits oben angedeutet, zur Modellbildung auf verschiedenen Abstraktionsebenen verwendet werden. Typischerweise können folgende Ebenen unterschieden werden, wobei Überlappungen durchaus auftreten:

- Systemebene

Dieser Begriff ist nur sehr schwammig zu definieren, da er stark von der Perspektive des jeweiligen Betrachters abhängt. So wird ein Chipdesigner seine Schaltung als System bezeichnen, während der Entwickler auf Platineebene eine andere Vorstellung hat. Der Inge-

nieur, der aus solchen Platinen einen Fernseher aufbaut, schließlich betrachtet die Platinen als Komponenten und sein Gerät als System. Dieser Sicht der Dinge wollen wir uns hier anschließen.

- Platinenebene

Hier geht es vor allem um das Zusammenspiel zwischen den Komponenten auf einer Platine. Die Vernetzung zwischen den Komponenten wird auf struktureller Ebene beschrieben, die Komponenten selbst auf möglichst abstrakter und damit verhaltensorientierter Ebene.

- Instruktionssatzebene

Hier geht es vor allem um Schaltungen die einem gegebenen Befehlssatz gehorchen, also etwa Mikroprozessoren oder DSP-Schaltungen¹. Typischerweise wird hier die Sequenz der Abarbeitung von Befehlen modelliert, was gerade für Mikroprozessoren leicht erreichbar ist, da die sich die Grundoperationen einer CPU recht einfach auf den Teil von VHDL abbilden lassen, der einer Programmiersprache ähnelt.

- Registertransferebene

Diese Ebene beschreibt die Register einer Schaltung und die Boole'schen Operationen zwischen ihnen. Auf dieser Ebene wird oftmals die Eingabe für ein Synthesewerkzeug gegeben, das anschließend daraus eine Gatterschaltung generiert. Auf Registertransferebene wird außerdem oftmals die Modellbildung für Zustandsautomaten angesiedelt.

- Strukturebene

Die rein strukturelle Ebene ist die VHDL-Entsprechung eines Schaltplans. Hier wird ausschließlich die Zusammenschaltung einiger Grundelemente und der sukzessive Aufbau von Hierarchie beschrieben. Funktionale Aspekte werden nicht betrachtet.

- Gatterebene

Beschreibungen auf dieser Ebene enthalten lediglich die Verbindungsstruktur eines „Gatterhaufens“. Diese Ebene ist typischerweise die Zielebene eines Syntheselaufs, woraus folgt, daß Syntheseprogramme in der Regel Beschreibungen in einer bestimmten HDL auf detailliertere Beschreibungen in derselben HDL abbilden.

Man kann nun eine ganze Reihe von Regeln angeben, wie „guter“ VHDL-Code aussehen sollte. Dabei mag sich mancher an die Empfehlungen für einen „guten“ Programmierstil erinnert fühlen.

1. VHDL-Modelle müssen der VHDL-Syntax folgen.
2. VHDL-Modelle sollten ein einheitliches Aussehen (look and feel) haben.
3. Sowohl Lesbarkeit als auch Wartbarkeit eines VHDL-Modells sollte gewährleistet sein.
4. Überflüssiger Code sollte vermieden werden.

¹DSP = digital signal processor.

5. VHDL-Modelle, die nicht der Synthese dienen, sollten auf Simulationsgeschwindigkeit optimiert sein.
6. VHDL-Modelle, die synthetisiert werden sollen, müssen den zusätzlichen Regeln des Anbieters der Synthese-Software gehorchen.
7. VHDL-Code sollte kohäsiv sein, d.h. zusammengehörige Teile sollten in nur einer Entwerfseinheit untergebracht werden.

A.2 Übersetzung und Simulation von VHDL

Im Rahmen dieser Beschreibung wenden wir uns vor allem der „compilierten Simulation“ zu. Einzelheiten in der Unterscheidung zwischen „compilierter Simulation“ und „interpretierter Simulation“ finden sich im Abschnitt über die Logiksimulation in Teil III. Ausgehend von einer textuellen VHDL-Beschreibung kommt man in mehreren Schritten zu einer simulierbaren Beschreibung. In einem ersten Schritt wird der VHDL-Text analysiert und auf Syntaxfehler geprüft. Falls keine Probleme auftreten, wird die VHDL-Beschreibung in ein Zwischenformat übersetzt, das typischerweise auf einer Programmiersprache, z.B. C, beruht. Diesen Vorgang nennt man auch *Compilation*. Damit wird die Simulation durch das Ausführung des betreffenden Programms bewerkstelligt. Zuvor findet jedoch noch die sogenannte *Elaboration* statt, die das übersetzte Modell in einen simulationsfähigen Zustand bringt. Im Rahmen der *Elaboration* werden folgende Aufgaben wahrgenommen:

- **Hierarchieexpansion**
Schaltungen werden mit Hierarchie entwickelt, damit Komponenten nur einmal entworfen, aber mehrfach eingesetzt werden können. Für die Simulation muß für jeden Aufruf einer Zelle eine eigene Instanz generiert werden, da die einzelnen Komponenten eines Typs unterschiedlichen Eingaben ausgesetzt sind.
- **Einsetzen von Defaultwerten**
Variablen und Konstanten können Standardwerte besitzen, die Verwendung finden, falls keine anderen Zuweisungen zum Tragen kommen. Für Konstanten mutet dies etwas verwunderlich an. Für bestimmte Typen von Konstanten wird deren Wert erst von einer übergeordneten Zelle gesetzt. Dies geschieht, wie die Zuweisung von Variablenwerten, im Rahmen der *Elaboration*.
- **Anbindung von Architekturen an die Komponenten**
Hier wird für jeden Aufruf einer Komponente die gewünschte Architektur bereitgestellt. Dies folgt aus der Tatsache, daß für eine Komponente durchaus mehrere Beschreibungen, d.h. Architekturen, entwickelt werden können, die verschiedenen Zwecken dienen.
- **Anforderung von Speicher für diverse Objekte**
In diesem Schritt wird für alle Objekte, z.B. Variablen, ein entsprechender Raum im Speicher reserviert.

Nach der Elaboration erfolgt die Initialisierung in der alle Signale einmal berechnet werden und alle Prozesse bis zum ersten Warte-Kommando durchlaufen. Dies entspricht dem „Arbeitspunkt“ der Schaltung. Dann wird die Simulationszeit auf 0 gesetzt und die eigentliche Simulation kann beginnen.

A.3 Organisation eines VHDL-Modells

Alle übersetzten VHDL-Modelle, die zu einem Entwurf gehören, werden in einer Bibliothek (library) zusammengefaßt. So könnten beispielsweise in einer Bibliothek für eine CPU Controller- oder ALU-Modelle liegen. Diese Modelle werden als Zellen (entity) bezeichnet, wobei das Interface der Zelle mit dem Entity-Kommando (s.u.) beschrieben wird. Die Sichten einer Zelle werden in VHDL als Architekturen (architecture) bezeichnet, siehe Abbildung A.3. Typischerweise werden je nach Bedarf mehrere Architekturen einer Zelle entwickelt. In einem Schaltplan entspricht die Entity einem Symbol und die Architektur der Schaltung, die das Symbol repräsentiert.

```
entity NOR2 is
    port (A, B: in std_logic;
          Q: out std_logic) ;
end NOR2 ;
architecture A1 of NOR2 is
begin
    . . .
end A1 ;
architecture A2 of NOR2 is
begin
    . . .
end A2 ;
```

Abbildung A.1: Deklaration einer Entity / Architektur.

Welche der Architekturen - zum Beispiel im Rahmen einer Simulation - zum Einsatz kommt wird durch eine Konfiguration (configuration) festgelegt. Deklarationen, die von mehreren Modellen benutzt werden, können in einem Package abgelegt werden, wobei es hier eine Zweiteilung zwischen der reinen Deklaration (package) und ihrer Realisierung (package body) gibt, Abbildung A.3. Ein Funktionskopf gehört beispielsweise in die erste Kategorie, ihre Realisierung in die zweite. Die vorgenannten fünf Konstrukte (entity, architecture, configuration, package, package body) werden auch als Entwurfseinheiten (design units) bezeichnet.

A.4 Sprachelemente

VHDL enthält eine Vielzahl von Konstrukten. Die wichtigsten Sprachelemente sollen im folgenden vorgestellt werden. Die Schreibweise von Namen (Groß- bzw. Kleinschreibung) wird von VHDL

```

package MYPACKAGE is
    ...
    Deklarationen
    ...
end MYPACKAGE;
package body MYPACKAGE is
    ...
    Implementierungen
    ...
end MYPACKAGE;

```

Abbildung A.2: Deklaration eines Package.

ignoriert. Kommentare beginnen in VHDL mit zwei aufeinanderfolgenden Minuszeichen und reichen bis zum Ende einer Zeile. VHDL ist eine mit Schlüsselworten gesteuerte Sprache; somit gibt es einige reservierte Namen:

abs	access	after	alias	all	and	architecture
array	assert	attribute	begin	block	body	buffer
bus	case	component	configuration	constant	disconnect	downto
else	elsif	end	entity	exit	file	for
function	generate	generic	guarded	if	in	inout
is	label	library	linkage	loop	map	mod
nand	new	next	nor	not	null	of
on	open	or	others	out	package	port
procedure	process	range	record	register	rem	report
return	select	severity	signal	subtype	then	to
transport	type	units	until	use	variable	wait
when	while	with	xor			

Folgende grundlegende Sprachelemente stehen zur Verfügung:

VHDL-Datenobjekte lassen sich folgenden vier Klassen zuordnen:

1. **constant**: hat einen unveränderlichen Wert.
2. **variable**: hat einen veränderlichen Wert.
3. **signal**: enthält alte, einen gegenwärtigen und zukünftige Werte. Nur zukünftige Werte können mittels Zuweisung ($\leq$) verändert werden.
4. **file**: entspricht einer Folge les- und schreibbarer Werte.

All diese Datenobjekte besitzen einen Typ. Einem Typ wiederum sind bestimmte Werte zugeordnet, z.B. ganze Zahlen größer 0. Der Typ eines Datenobjekts ist unveränderlich. Die auf ein Datenobjekt ausführbaren Operationen werden durch den Typ bestimmt. Einige Eigenschaften eines Typs lassen sich als Attribute der Datenobjekte bzw. des Typs selbst abfragen. VHDL kennt vier grundlegende Typen:

```

- Bezeichner beginnen mit a-z;
- ab dem zweiten Buchstaben sind auch Ziffern erlaubt.
COUNT aBc X f123

- Zahlen; das '_'-Zeichen ändert die Zahl nicht
12 0 1E5 123_456 3.141_593
- Zahlen mit Basis 2 .. 16, Format: Basis#Zahl#Exponent
X#FOF# B#1111_0000_1111# O#4711#

- Zeichenkonstanten
'A' 'B' 'C' " ' '@' ' '

- Zeichenketten
"Hi there!" "Hello world!"

- physikalische Größen
10 ns 2.2 V 50 pF

```

Abbildung A.3: VHDL-Literale.

1. „universal integer“: Ganze Zahlen mit wenigstens 32Bit.
2. „physical types“: Ganze Zahlen mit Skalierung.
3. „floating point“: Gleitkommazahlen.
4. „enumeration types“: Daten mit endlichem Wertevorrat, der aus Bezeichnern und / oder Zeichenkonstanten besteht und durch Aufzählung definiert wird.

Ausdrücke werden in VHDL durch die in Abbildung A.4 dargestellten Operatoren aufgebaut.

A.5 STANDARD-Package

Einige Informationen werden grundsätzlich vor einer VHDL-Simulation zur aktuellen Beschreibung dazugeladen. Dies geschieht in Form eines Package. In folgenden wird ein Teil des STANDARD-Package dargestellt, das unter anderem einige wichtige Datentypen vordefiniert, Abbildung A.5.

Die ASCII-Zeichen FS und US sind wegen möglicher Namenskonflikte mit den Zeiteinheiten fs und us vorsichtshalber FSP und USP getauft worden. Ohne die im STANDARD-Package definierten Datentypen kommt ein VHDL-Programm nicht aus. So ist das Ergebnis einer Vergleichsoperation von dem hier definierten Typ `Boolean`; eine Bitstring-Konstante besitzt den Typ `Bit_Vector`. Das Package-Konzept hat sich als geeignetes Mittel erwiesen, die eigentliche VHDL-Sprache vergleichsweise klein zu halten, aber dafür Erweiterungen des Sprachumfangs zu unterstützen.

```

- Vergleichsoperatoren
- kleiner, kleiner gleich, gleich, größer gleich, ungleich
<, <=, =, >=, >, /=

- Logische Operatoren, definiert auf den Typen Boolean, bit etc.
and or nand nor xor not

- Arithmetische Operatoren
- ** (= Potenz), mod (= modulo), rem (= remainder bzw. Rest),
- abs (= absolut)
+ - * / ** mod rem abs
- Konkatenation
"Hallo" & " " & "Leute!"

```

Abbildung A.4: VHDL-Ausdrücke.

A.6 Parallele Anweisungen

Die Abarbeitung von Kommandos ist, wie oben schon angedeutet, zunächst parallel, d.h. alle Kommandos werden gleichzeitig ausgeführt. Dasselbe Verhalten wäre auch bei einer Schaltung zu erwarten, bei der ja beispielsweise auch alle Gatter gleichzeitig ausgewertet werden. Die wichtigste parallele Anweisung ist die Signalzuweisung, die mit '`<=`' bezeichnet wird. Von einer später vorgestellten Ausnahme abgesehen, werden alle Signalzuweisungen grundsätzlich parallel ausgeführt. Da ein Signal sowohl eine Historie als auch eine Liste mit bereits bekannten zukünftigen Ereignissen besitzt, kann man einer solchen Zuweisung auch eine zeitliche Verzögerung zuordnen. Beispielsweise würde ein AND-Gatter mit zwei Nanosekunden Verzögerungszeit folgendermaßen bezeichnet:

```
q <= a and b after 2 ns;
```

Diese Zuweisung läßt sich auch durch Bedingungen steuern, so daß man einen Multiplexer recht einfach formulieren kann:

```
q <= a when ( select = '0' ) else b;
```

Es hat sich allerdings herausgestellt, daß sich manche Modelle besser in sequentiellem Code formulieren lassen. Um diesem Umstand Rechnung zu tragen, wurde die Prozeß-Anweisung (process) eingeführt. Die Prozeß-Anweisung selbst ist parallel, da die Prozesse parallel mit allen anderen parallelen Kommandos ausgeführt werden. Ein Prozeß bildet jedoch eine Endlosschleife über die in ihm definierten, sequentiellen Anweisungen.

```

package STANDARD is
  - Aufzählungstyp mit Namen
  type Boolean is (FALSE , TRUE);
  - Aufzählungstyp mit Characters
  type Bit is ('0' , '1');
  type Character is (
    NUL , SOH , STX , ETX , EOT , ENQ , ACK , BEL ,
    BS , HT , LF , VT , FF , CR , SO , SI ,
    DLE , DC1 , DC2 , DC3 , DC4 , NAK , SYN , ETB ,
    CAN , EM , SUB , ESC , FSP , GSP , RSP , USP ,
    ' ' , '!' , ' ' , '#' , '$' , '%' , '&' , '"' , - etc.
    DEL );
  - für Nachrichten an den Benutzer
  type Severity_Level is (
    NOTE , WARNING , ERROR , FAILURE );
  - Bereich der numerischen Werte ist implementierungsabhängig
  type Integer is range -2_147_483_647 to 2_147_483_647;
  type Real is range -1E38 to 1E38;
  - physikalischer Typ
  type Time is range -1E18 to 1E18 units
    fs;
    ps = 1000 fs;
    ns = 1000 ps; us = 1000 ns; ms = 1000 us;
    sec = 1000 ms; min = 60 sec; hr = 60 min;
  end units;
  - Funktion 'NOW' liefert die aktuelle Simulationszeit
  function Now return Time;
  - Typen mit beschränktem Bereich
  subtype Natural is Integer range 0 to Integer'high;
  subtype Positive is Integer range 1 to Integer'high;
  - Definition von ARRAY-Typen
  type String is array (Positive range <>) of Character;
  type Bit_Vector is array (Natural range <>) of Bit;
end STANDARD;

```

Abbildung A.5: VHDL Standard Package.

A.7 Sequentielle Anweisungen

Innerhalb eines Prozesses kann man Modelle formulieren, wie man es von Programmiersprachen gewohnt ist. Es gibt eine Reihe von Konstrukten, die aus prozeduralen Programmiersprachen entlehnt worden sind, Abbildung A.7.

```

test : process
  variable fak : Integer; - vordefinierter Typ
  variable mant : Bit_Vector ( 31 downto 0 ); - vordefinierter Typ
  variable weekday : day; - muß als Aufzählungstyp deklariert werden
  variable anwesende : Natural; - Untertyp von integer
  variable a , b , delta : Real; - in package Standard definiert
  variable enable : Boolean; - in package Standard definiert
  variable sig : Bit;
begin
  - Endlosschleife ...
  - Beispiel for-Anweisung
  - N! berechnen:
  fak := 1;
  for j in 2 to N loop fak := fak * j; end loop;
  - Beispiel while-Anweisung
  - Normalisieren einer Mantisse
  while mant(31) = '0' loop
    mant := left_shift(mant);
  end loop;
  - case-Anweisung
  case weekday is
    when sat | sun => anwesende := 1;
    when mon | fri => anwesende := 3;
    when others => anwesende := 5;
  end case;
  - if-then-else-Anweisung
  if abs(a - b) < delta then
    delta := delta / 2.0;
  elsif a - b > 0.0 then
    b := b + delta;
  else
    b := b - delta;
  end if;
  - Sequentielle Signal Zuweisung
  if enable then
    sig <= '1' after 1 ns;
  end if;
end process test;

```

Abbildung A.6: Kombinatorische VHDL-Anweisungen.

A.8 Beispiele

Im Rahmen dieses Abschnitts soll der Schaltungsentwurf mit VHDL anhand einiger Beispiele illustriert werden. Aus diesem Grund wurde darauf geachtet, daß alle Beispiele synthetisierbar sind und damit automatisch auf eine Gatternetzliste abgebildet werden können.

A.8.1 Volladdierer

Das erste Beispiel ist ein Volladdierer, an dem die grundlegenden Modellierungsstrategien aufgezeigt werden sollen. Zu diesem Zweck wird zunächst die Schnittstelle der Zelle zur Außenwelt mit dem Entity-Kommando definiert, Abbildung A.8. Im Normalfall kann ein Modell auf einer der drei folgenden Ebenen modelliert werden: Verhalten (behavior), Datenfluß (dataflow) und Struktur (structure), siehe Abbildung 6.1 auf Seite 62. Alle drei Sichten sind in Abbildung A.8 dargestellt. Die Verhaltenssicht beschreibt die Abbildung von logischen Werten an den Eingängen auf logische Werte an den Ausgängen. Bei der am Datenfluß orientierten Methode wird ein Satz von arithmetischen und logischen Gleichungen aufgestellt. Diese Gleichungen beschreiben den Informationsfluß durch die Komponenten der Schaltung. Die Strukturbeschreibung schließlich stellt lediglich eine Liste der Komponenten und Verbindungen eines Systems - mithin eine Netzliste - auf. Den einzelnen Komponenten ist in dieser Beschreibung keinerlei Funktion zugeordnet. Der Voll-Addierer besteht dabei aus zwei Halbaddierern und einem OR-Gatter, siehe Abbildung A.7.

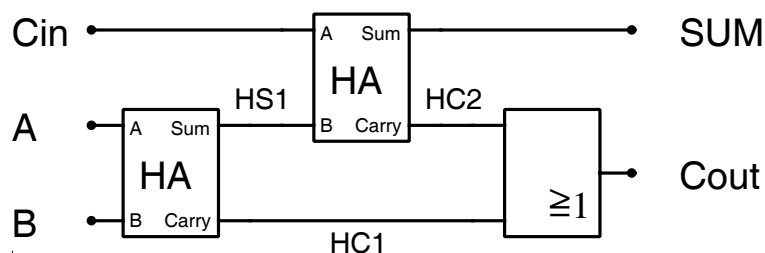


Abbildung A.7: Graphische Darstellung der Strukturbeschreibung eines Volladdierers.

```

library IEEE;
use IEEE.std_logic_1164.all;

entity full_adder is
port (A ,B ,CIN: in std_logic;
      SUM ,COUT: out std_logic);
end full_adder;

architecture behavior of Full_Adder is
begin
  process (A ,B ,CIN)
    variable tmp : std_logic_vector(2 downto 0);
  begin
    tmp (2) := A; tmp (1) := B; tmp (0) := CIN;
    case tmp is
      when "000" =>
        SUM <= '0'; COUT <= '0';
      when "001" =>
        SUM <= '1'; COUT <= '0';
      when ...
    end case;
  end process;
end behavior;

architecture dataflow of full_adder is
begin
  SUM <= A xor B xor CIN;
  COUT <= (A and B) or (A and CIN) or (B and CIN);
end dataflow;

architecture structure of full_adder is
  ...
begin
  ...
  inst1 : half_adder(A ,B ,HC1 ,HS1);
  inst2 : half_adder(CIN ,HS1 ,HC2 ,SUM);
  inst3 : or_gate(HC1 ,HC2 ,COUT);
end structure;

```

Abbildung A.8: Beschreibung des Verhaltens, des Datenflusses und der Struktur eines Voll-Addierers.

A.8.2 Multiplexer

Die Aufgabe eines Multiplexers besteht darin, eine von zwei möglichen Eingangsleitungen A oder B auf einen Ausgang Q durchzuschalten. Welcher der Eingänge dabei berücksichtigt wird, entscheidet sich anhand einer Steuerleitung SEL. Die in Abbildung A.9 gezeigten VHDL-Beschreibungen stellen ebenfalls einen Multiplexer dar. Dieser lässt sich sowohl durch eine parallele Anweisung (Architektur `syn_par`) als auch durch einen Prozess mit sequentiellen Anweisungen (Architektur `syn_seq`) beschreiben. Das Syntheseresult sollte dabei genau die Multiplexerzelle der unterliegenden Bibliothek sein.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux is
  port (A, B, SEL: in std_logic;
        Q : out std_logic);
end mux;

architecture syn_par of mux is
begin
  Q <= A when SEL = '1' else B;
end syn_par;

architecture syn_seq of mux is
begin
  assign : process (SEL ,A ,B)
  begin
    if SEL = '1' then
      Q <= A;
    else
      Q <= B;
    end if;
  end process;
end syn_seq;
```

Abbildung A.9: Parallele und sequentielle VHDL-Beschreibung eines Multiplexers.

A.8.3 Komparator

Ein Komparator dient dazu, zwei Bitvektoren A und B auf Gleichheit zu prüfen. Abbildung A.10 zeigt ein solches VHDL-Modell, wobei sich das Ergebnis nur zu bestimmten Zeitpunkten, und damit synchron zu einem Takt, verändern soll. Schließlich wird außerdem ein Reset-Eingang vorgesehen, so daß man den Ausgang Q auch explizit auf 0 setzen kann. Die Synthese dieser Beschreibung sollte eine Gatterschaltung ergeben, bei der die beiden Eingangs-Vektoren bitweise durch EXORs verglichen werden. Die Ausgänge der EXOR-Gatter wiederum werden mit einem NOR verknüpft. Das resultierende Signal liegt bei Gleichheit auf 1 und Ungleichheit auf 0. Es wird in einem D-Flipflop zwischengespeichert, das seine Werte nur - wie im Modell vorgesehen - mit der steigenden Taktflanke ändert. Das resultierende D-Flipflop verfügt schließlich noch über einen Reseteingang, so daß auch diese Funktion einfach realisiert werden kann.

```

library IEEE ;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity comp is
    port (CLK: in std_logic;
          RESET: in std_logic;
          A: in std_logic_vector(3 downto 0);
          B: in std_logic_vector(3 downto 0);
          Q: out std_logic);
end comp;

architecture syn of comp is
begin
    comp : process(CLK , RESET)
    begin
        if RESET = '0' then
            Q <= '0';
        elsif rising_edge(CLK) then
            if A = B then
                Q <= '1';
            else
                Q <= '0';
            end if;
        end if;
    end process;
end syn;

```

Abbildung A.10: VHDL-Beschreibung eines Komparators.

A.8.4 Addierer

Das in Abbildung A.11 dargestellte Modell eines 4-Bit-Addierers verwendet den auch für Bit-Vektoren (`std_logic_vector`) verwendbaren '+'-Operator. Üblicherweise ist im Rahmen des Syntheseprogramms einstellbar, welcher Typ von Addierer eingesetzt werden soll. Explizit ist allerdings noch die Behandlung von Überlaufen zu formulieren. Zu diesem Zweck wird die Addition auf zwei 5-Bit-Vektoren durchgeführt, TMP1 und TMP2, deren oberstes Bit zunächst auf 0 steht. In den unteren vier Bit entsprechen sie den Summanden A und B. Ein eventuell auftretender Überlauf entsteht im obersten Bit und kann dort separat behandelt werden. Auch der Addierer arbeitet synchron zum Takt CLK. Außerdem läßt sich der 5-Bit Ergebnisvektor mit dem RESET-Eingang explizit auf 0 setzen.

```

library IEEE ;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity adder is
  port (CLK : in std_logic;
        RESET : in std_logic;
        A : in std_logic_vector(3 downto 0);
        B : in std_logic_vector(3 downto 0);
        Q : out std_logic_vector(4 downto 0));
end adder;

architecture syn of adder is
  signal TMP1 , TMP2: std_logic_vector(4 downto 0);
begin
  TMP1 <= '0' & A;
  TMP2 <= '0' & B;
  adder : process(CLK ,RESET)
  begin
    if RESET = '0' then
      Q <= "00000";
    elsif rising_edge(CLK) then
      Q <= TMP1 + TMP2;
    end if;
  end process;
end syn;

```

Abbildung A.11: VHDL-Beschreibung eines 4-Bit Addierers.

A.8.5 Multiplizierer

Abbildung A.12 zeigt die VHDL-Beschreibung eines 4x4-Bit Multiplizierers, der nach dem „shift-and-add“-Verfahren arbeitet. Man beachte, daß hier die Multiplikation nicht durch den `**`-Operator dargestellt, sondern auf Addition und bitweises Schieben zurückgeführt wird. Die `wait`-Anweisungen sorgen dabei dafür, daß die nötigen Additionen in aufeinanderfolgenden Takt-Zyklen stattfinden. Das Schieben der Operanden wird durch eine geeignete Wahl der betreffenden Indizes bewerkstelligt.

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity mult is
  port(CLK: in std_logic;
        A : in std_logic_vector(3 downto 0);
        B : in std_logic_vector(3 downto 0);
        Q : out std_logic_vector(7 downto 0));
end mult;

architecture syn of mult is
begin
  mult : process
    variable PP, RES : std_logic_vector(7 downto 0);
  begin
    wait until rising_edge(CLK);
    RES := To_StdLogicVector(0,8);
    for I in 0 to 3 loop
      wait until rising_edge(CLK);
      PP := To_StdLogicVector(0,8);
      if B(I) = '1' then
        PP((I + 3) downto I) := A;
      end if;
      RES := RES + PP;
    end loop;
    wait until rising_edge(CLK);
    Q <= RES;
  end process;
end syn;

```

Abbildung A.12: VHDL-Beschreibung eines 4x4-Bit Multiplizierers.

Glossar

cell	Zelle, Schaltung bzw. Schaltungsteil
CIW	Cadence Kontrollfenster
cursor	graphischer Positionsanzeiger
dialogbox	Fenster für Ein- und Ausgabe
design	Entwurf
Design Rule Check, DRC	geometrische Entwurfsregelkontrolle
Electrical Rule Check, ERC	elektrische Entwurfsregelkontrolle
extraction	Schaltungs-Extraktion
form	Dialogfenster
grid	Raster, einstellbares Raster im graphischen Editor
instance	Instanz, Referenz einer Zelle
layout	Layout
layer	graphische Entwurfsebene, z.B. Poly, Metall
library	Bibliothek, nimmt die Zellen eines Entwurfs auf
library-browser	Fenster zur Auswahl von Zellen
netlist	Netzliste, textuelle Darstellung der Verbindungen zwischen Bauelementen
manhattan	rechtwinklig, d.h. im Layout kommen nur rechte Winkel vor
octagonal	oktagonal, d.h. nur 45 und 90 Grad Winkel im Layout
path	Linie mit einstellbarer Breite, z.B. für das Layout von Leitungen
pin	Anschluß, z.B. an ein Bauelement
polygon	flächenbeschreibendes Vieleck im Layout
properties	Attribute, Eigenschaften von Objekten, z.B. R für Widerstände
rectangle	Rechteck im Layout
reference-library	Referenzbibliothek, enthält Zellen von allgemeinem Interesse
schematic	Schaltplan
setup	Einstellung
shell	Programm für die Kommandoeingabe im Betriebssystem
simulation	Simulation
terminal	Anschluß, oft synonym zu pin gebraucht
via	Kontaktloch zwischen zwei Metallebenen
view	Sicht einer Zelle, typische Sichten: Layout, Schaltplan
window	Fenster, graphischer Arbeitsbereich
wire	Leitung

Index

- Anwender-programmierbare Logik, 15
- application specific integrated circuit, 14
- ASIC, 14

- BDS, 64
- Bedingungsgraph, 150
- Bitraster, 101
- Bottom-Up, 11

- CAD, 1
- CAE, 1
- CAQ, 1
- CAT, 1
- complex programmable logic device, 16
- computer-aided design, 1
- computer-aided engineering, 1
- computer-aided quality management, 1
- computer-aided test, 1
- concurrent engineering, 63
- constraint, 149
- constraint graph, 145, 150
- constraints, 145
- correctness by construction, 129
- CPLD, 16

- daCapo, 64
- delay calculator, 141
- design assistant, 10
- design rule check, 104
- Designflus
 - Bottom-Up, 11
 - Top-Down, 11
- Designregelprüfung, 104
 - Abstand, 110
 - Breite, 110
 - Überhang, 110
 - Überlappung, 110
- doglegs, 150
- DRC, 104

- E-Beam, 121
- Elektronenstrahl-Belichtung, 121
- Entwurfshierarchie, 5
- Entwurfslokalität, 8
- Entwurfsmethodik, 2
- Entwurfsmodularität, 7

- Entwurfsregularität, 7
- Entwurfswerkzeuge, 1
- Euler-Pfad, 133
- expansion, 5
- explosion, 5

- Fertigungstechnologie, 2
- field programmable gate array, 17
- Flächenoperationen
 - Dimensionskontrolle, 110
 - Selektion, 109
- flattening, 5
- Floorplanning, 141
- FPGA, 17
- Framework, 11
- full-custom, 14

- Gajski-Diagramm, 9
- GAL, 16
- Gate-Array, 18
- generic array logic, 16
- Geometriebeschreibung, 8
- global routing, 138

- Handlayout, 21
- hardware description language, 59
- Hardwarebeschreibungssprache, 59
- HDL, 59
- Hierarchie, 5
 - Auflösung der, 5
- Hierarchieebenen, 8
- high level synthesis, 60

- IEEE, 64

- jogs, 150

- Kante, 102
- Kompaktierung, 148

- Layout
 - Kompaktierung, 148
 - Synthese, 137
- local routing, 138
- Lokalität, 8

- M-HDL, 64

- Makrozellen, 20
- manhattan routing, 145
- Modularität, 7
- PAL, 16
- Pattern Generator, 121
- PLA, 15
- placement, 17, 137, 141
- Plane-Sweep, 102
- Plazierung, 17, 137, 141
- PLD, 15
- Polygon, 99
- Produktionswerkzeuge, 1
- programmable array logic, 16
- programmable logic array, 15
- programmable logic device, 15
- redesign, 29
- Regularität, 7
- Reticle, 121
- routing, 17, 137
- Scanline, 102
- Schaltplan, 10
- Schaltplan-Editor, 10
- Schaltung
 - dynamische Verifikation einer, 10
 - Eingabe einer, 10
 - Optimierung einer, 10
 - Simulation einer, 10
 - statische Verifikation einer, 10
 - Synthese einer, 10
- Schaltungsextraktion, 119
- schematic, 10
- schematic editor, 10
- semi-custom, 14
- Standardbaustein, 14
- Standardzellen, 19
- Strukturbeschreibung, 8
- Top-Down, 11
- Verdrahtung, 17, 137
 - globale, 138
 - lokale, 138
- Verhaltensbeschreibung, 8
- Verilog, 64
- VHDL, 64
 - Architektur, 154
 - Bibliothek, 154
 - Compilation, 153
 - Design-Einheiten, 154
 - Elaboration, 153
 - Entity, 154
 - Konfiguration, 154
 - Package, 154
 - VHDL-AMS, 64
 - Y-Diagramm, 9
 - Zelle, 5