

# HANSER



## Leseprobe

zu

# Grundlagen der Technischen Informatik

von Dirk W. Hoffmann

Print-ISBN: 978-3-446-47779-7

E-Book-ISBN: 978-3-446-47813-8

Weitere Informationen und Bestellungen unter

<https://www.hanser-kundencenter.de/fachbuch/artikel/9783446477797>

sowie im Buchhandel

© Carl Hanser Verlag, München



# Vorwort

---

Die Computertechnik hat in wenigen Jahrzehnten eine Entwicklung vollzogen, die in ihrer Geschwindigkeit und Intensität einzigartig ist. Setzten sich die ersten Computer noch aus vergleichsweise wenigen Schaltkreisen zusammen, so verrichten in jedem modernen Arbeitsplatzrechner, Tablet-PC oder Smartphone Abermillionen von Transistoren ihren Dienst und führen in jeder Sekunde Milliarden von Berechnungen aus. Doch so rasant die Entwicklung der letzten Jahrzehnte auch war: Vergleichen wir die Maschinen der Pionierzeit mit unseren modernen Rechenboliden, so lassen sich eine Reihe von Grundprinzipien identifizieren, die sich im Laufe der Zeit zwar weiterentwickelt, aber im Kern nicht verändert haben. Diese Grundprinzipien, zusammen mit ihren modernen Ausprägungen, formen das Gebiet der technischen Informatik und sind Gegenstand des vorliegenden Buchs.

Geschrieben habe ich das Buch für Bachelor-Studenten der Fachrichtungen Informatik, Elektrotechnik, Informationstechnik und verwandter Studiengänge. Inhaltlich habe ich mich dabei an den typischen Lehrinhalten orientiert, die im Grundstudium an Hochschulen und Universitäten vermittelt werden. Neben dem Grundlagenwissen aus den Gebieten der Halbleitertechnik, der Zahlendarstellung und der booleschen Algebra werden die Entwurfsprinzipien kombinatorischer und sequenzieller Hardware-Komponenten bis hin zur Beschreibung moderner Prozessor- und Speicherarchitekturen vermittelt. Damit spannt das Buch den Bogen von den mathematischen Grundlagen digitaler Schaltelemente bis hin zu den ausgefeilten Hardware-Optimierungen moderner Hochleistungscomputer.

Es ist mir ein besonderes Anliegen, den Stoff anwendungsorientiert und didaktisch ansprechend zu vermitteln. Damit das Buch sowohl vorlesungsbegleitend als auch zum Selbststudium eingesetzt werden kann, werden die Lehrinhalte aller Kapitel durch zahlreiche Übungsaufgaben komplementiert. Des Weiteren habe ich etliche Anwendungsbezüge mit aufgenommen, um eine enge Verzahnung zwischen Theorie und Praxis zu erreichen.

Seit dem Erscheinen der letzten Auflage habe ich wieder zahlreiche Zuschriften erhalten, über die ich mich sehr gefreut habe. Namentlich bedanken möchte ich mich bei Herrn Prof. Dr. Michael Wiehl für wertvolle Hinweise im Bereich der MOS-Schaltungstechnik. Inzwischen erscheinen die *Grundlagen der technischen Informatik* in der siebten Auflage, und ich bin weiterhin jedem aufmerksamen Leser für Hinweise zu Verbesserungsmöglichkeiten oder Fehlern dankbar.

---

## Symbolwegweiser



Definition



Satz, Lemma, Korollar



Leichte Übungsaufgabe



Mittelschwere Übungsaufgabe



Schwere Übungsaufgabe

---

## Lösungen zu den Übungsaufgaben

In wenigen Schritten erhalten Sie die Lösungen zu den Übungsaufgaben:

1. Gehen Sie auf die Seite [www.dirkwhoffmann.de/TI](http://www.dirkwhoffmann.de/TI)
2. Geben Sie den neben der Aufgabe abgedruckten Webcode ein
3. Die Musterlösung wird als PDF-Dokument angezeigt



# Inhaltsverzeichnis

---

<b>1</b>	<b>Einführung</b>	<b>11</b>
1.1	Was ist technische Informatik? . . . . .	11
1.2	Vom Abakus zum Supercomputer . . . . .	13
1.3	Wohin geht die Reise? . . . . .	30
<b>2</b>	<b>Halbleitertechnik</b>	<b>33</b>
2.1	Halbleiter . . . . .	34
2.1.1	Atommodell von Bohr . . . . .	34
2.1.2	Reine Halbleiter . . . . .	37
2.1.3	Dotierte Halbleiter . . . . .	39
2.2	Integrierte Schaltelemente . . . . .	41
2.2.1	Halbleiterdioden . . . . .	41
2.2.2	Bipolartransistoren . . . . .	42
2.2.3	Feldeffekttransistoren . . . . .	46
2.3	Chip-Fertigung . . . . .	51
2.3.1	Produktion integrierter Schaltkreise . . . . .	51
2.3.2	Integrationsdichte . . . . .	57
2.4	Übungsaufgaben . . . . .	58
<b>3</b>	<b>Zahlendarstellung und Codes</b>	<b>59</b>
3.1	Zahlensysteme . . . . .	60
3.2	Rechnerinterne Zahlenformate . . . . .	67
3.2.1	Darstellung natürlicher Zahlen . . . . .	67
3.2.2	Darstellung rationaler Zahlen . . . . .	73
3.3	Zahlencodes . . . . .	80
3.3.1	Tetraden-Codes . . . . .	80
3.3.2	Fehlererkennende Codes . . . . .	84
3.4	Übungsaufgaben . . . . .	86
<b>4</b>	<b>Boolesche Algebra</b>	<b>89</b>
4.1	Axiomatisierung nach Huntington . . . . .	90
4.1.1	Mengenalgebra . . . . .	91
4.1.2	Schaltalgebra . . . . .	93
4.2	Boolesche Ausdrücke und Aussagen . . . . .	95
4.2.1	Abgeleitete Operatoren . . . . .	97
4.2.2	Erfüllbarkeit und Äquivalenz . . . . .	100
4.2.3	Strukturelle Induktion . . . . .	102

4.2.4	Dualitätsprinzip . . . . .	105
4.3	Rechnen in booleschen Algebren . . . . .	109
4.3.1	Abgeleitete Umformungsregeln . . . . .	109
4.3.2	Vereinfachung boolescher Ausdrücke . . . . .	111
4.3.3	Vollständige Operatorensysteme . . . . .	117
4.4	Normalformdarstellungen . . . . .	119
4.4.1	Konjunktive und disjunktive Normalform . . . . .	119
4.4.2	Reed-Muller-Normalform . . . . .	122
4.4.3	Binäre Entscheidungsdiagramme . . . . .	125
4.5	Übungsaufgaben . . . . .	133
<b>5</b>	<b>Schaltnetze</b>	<b>139</b>
5.1	Grundlagen der Digitaltechnik . . . . .	140
5.1.1	Schaltkreisfamilien . . . . .	140
5.1.2	MOS-Schaltungstechnik . . . . .	145
5.1.3	Lastfaktoren . . . . .	155
5.2	Schaltungssynthese . . . . .	156
5.2.1	Zweistufige Schaltungssynthese . . . . .	157
5.2.2	BDD-basierte Schaltungssynthese . . . . .	158
5.2.3	FDD-basierte Schaltungssynthese . . . . .	159
5.3	Formelsynthese . . . . .	161
5.3.1	Funktionale Formelsynthese . . . . .	161
5.3.2	Relationale Formelsynthese . . . . .	163
5.3.3	Definitorische Formelsynthese . . . . .	164
5.4	Komplexitätsanalyse . . . . .	167
5.5	Zeitverhalten digitaler Schaltungen . . . . .	169
5.5.1	Signalausbreitung und -verzögerung . . . . .	169
5.5.2	Störimpulse . . . . .	171
5.6	Übungsaufgaben . . . . .	175
<b>6</b>	<b>Minimierung</b>	<b>181</b>
6.1	Minimierungsziele . . . . .	182
6.2	Karnaugh-Veitch-Diagramme . . . . .	186
6.2.1	Minimierung partiell definierter Funktionen . . . . .	190
6.2.2	Konstruktion Hazard-freier Schaltungen . . . . .	194
6.2.3	Minimierung mehrstelliger Funktionen . . . . .	196
6.3	Quine-McCluskey-Verfahren . . . . .	197
6.4	Übungsaufgaben . . . . .	201
<b>7</b>	<b>Standardschaltnetze</b>	<b>205</b>
7.1	Motivation . . . . .	206
7.2	Multiplexer und Demultiplexer . . . . .	206
7.3	Komparatoren . . . . .	213
7.4	Präfix-Logik . . . . .	215

7.5	Addierer . . . . .	218
7.5.1	Halb- und Volladdierer . . . . .	218
7.5.2	Carry-ripple-Addierer . . . . .	220
7.5.3	Carry-look-ahead-Addierer . . . . .	221
7.5.4	Conditional-Sum-Addierer . . . . .	224
7.5.5	Präfix-Addierer . . . . .	227
7.5.6	Carry-save-Addierer . . . . .	229
7.6	Inkrementierer . . . . .	232
7.7	Subtrahierer . . . . .	233
7.8	Multiplizierer . . . . .	234
7.8.1	Matrixmultiplizierer . . . . .	235
7.8.2	Carry-save-Multiplizierer . . . . .	238
7.8.3	Wallace-Tree-Multiplizierer . . . . .	241
7.8.4	Dadda-Tree-Multiplizierer . . . . .	246
7.9	Barrel-Shifter . . . . .	249
7.10	Arithmetisch-logische Einheit . . . . .	251
7.11	Programmierbare Logikbausteine . . . . .	253
7.12	Übungsaufgaben . . . . .	256
<b>8</b>	<b>Schaltwerke</b>	<b>265</b>
8.1	Digitale Speicherelemente . . . . .	266
8.1.1	Asynchrone Speicherelemente . . . . .	267
8.1.2	Taktzustandgesteuerte Speicherelemente . . . . .	271
8.1.3	Taktflankengesteuerte Speicherelemente . . . . .	274
8.1.4	Bevorrechtigte Eingänge . . . . .	281
8.1.5	CMOS-Implementierung . . . . .	282
8.2	Vom Flipflop zum Schaltwerk . . . . .	285
8.2.1	Endliche Automaten . . . . .	286
8.2.2	Schaltwerksynthese . . . . .	289
8.3	Übungsaufgaben . . . . .	293
<b>9</b>	<b>Standardschaltwerke</b>	<b>299</b>
9.1	Register . . . . .	300
9.1.1	Auffangregister . . . . .	300
9.1.2	Schieberegister . . . . .	302
9.1.3	Universalregister . . . . .	304
9.1.4	Akkumulatoren . . . . .	305
9.2	Zähler . . . . .	308
9.2.1	Synchrone Binärzähler . . . . .	309
9.2.2	Asynchrone Binärzähler . . . . .	313
9.2.3	Mischzähler . . . . .	314
9.2.4	Instruktionszähler . . . . .	316
9.3	Hauptspeicher . . . . .	318
9.3.1	SRAM-Speicher . . . . .	318

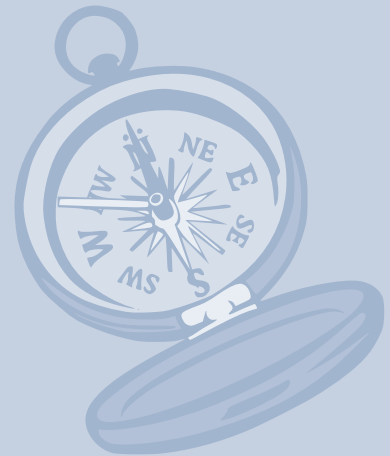
9.3.2	DRAM-Speicher . . . . .	320
9.3.3	Fehlererkennung und -korrektur . . . . .	327
9.4	Übungsaufgaben . . . . .	330
<b>10</b>	<b>Register-Transfer-Entwurf</b>	<b>335</b>
10.1	Entwurf komplexer Systeme . . . . .	336
10.1.1	Operationswerksynthese . . . . .	338
10.1.2	Steuerwerksynthese . . . . .	340
10.2	Mikroprogrammierung . . . . .	343
10.3	Übungsaufgaben . . . . .	349
<b>11</b>	<b>Mikroprozessortechnik</b>	<b>351</b>
11.1	Elemente eines Mikrorechners . . . . .	352
11.1.1	Von-Neumann-Architektur . . . . .	352
11.1.2	Aufbau der CPU . . . . .	356
11.2	Ein einfacher Modellprozessor . . . . .	360
11.3	Übungsaufgaben . . . . .	374
<b>12</b>	<b>Rechnerstrukturen</b>	<b>377</b>
12.1	Rechnerklassifikation nach Flynn . . . . .	378
12.2	Instruktionsarchitekturen . . . . .	379
12.2.1	CISC-Prozessoren . . . . .	380
12.2.2	RISC-Prozessoren . . . . .	385
12.3	Methoden zur Leistungssteigerung . . . . .	389
12.3.1	Pipelining . . . . .	389
12.3.2	Cache-Speicher . . . . .	394
12.4	Leistungsbewertung . . . . .	400
12.4.1	Maßzahlen zur Leistungsbewertung . . . . .	400
12.4.2	Benchmarks . . . . .	403
12.5	Übungsaufgaben . . . . .	406
<b>A</b>	<b>Notationsverzeichnis</b>	<b>411</b>
<b>B</b>	<b>Abkürzungsverzeichnis</b>	<b>413</b>
<b>C</b>	<b>Glossar</b>	<b>415</b>
	<b>Literaturverzeichnis</b>	<b>433</b>
	<b>Namensverzeichnis</b>	<b>437</b>
	<b>Sachwortverzeichnis</b>	<b>439</b>

## 6 Minimierung

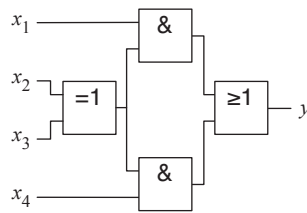
---

### In diesem Kapitel werden Sie . . .

- die typischen Minimierungsziele des Schaltungsentwurfs kennen lernen,
- ein Grundverständnis für Kostenfunktionen entwickeln,
- das Konstruktionsprinzip von Karnaugh-Veitch-Diagrammen verstehen,
- erlernen, wie sich boolesche Funktionen mit Hilfe von Karnaugh-Veitch-Diagrammen grafisch minimieren lassen,
- mit dem Quine-McCluskey-Verfahren einen alternativen Weg zur Schaltungsminimierung beschreiten.







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

**Abbildung 6.1:** Strukturbild und Wahrheitstabelle unseres Schaltnetzbeispiels

## 6.1 Minimierungsziele

In den beiden vorangegangenen Kapiteln haben wir mit der booleschen Algebra das mathematische Fundament zur formalen Beschreibung beliebiger Schaltfunktionen geschaffen und gezeigt, wie sich mit Hilfe der elementaren Logikgatter selbst komplexe Funktionen auf direktem Weg in Hardware-Schaltungen übersetzen lassen. Eine wichtige Erkenntnis dieser Überlegungen wollen wir an dieser Stelle nochmals aufgreifen: Auf der Logikebene besitzt jede boolesche Funktion mehr als eine Darstellung, die jede für sich zu einer ganz unterschiedlichen Implementierung auf der Hardware-Ebene führt. Zwei der möglichen Darstellungen haben wir in Form der disjunktiven Normalform (DNF) und der konjunktiven Normalform (KNF) bereits ausführlich untersucht. Für die direkte Umsetzung in eine Digitalschaltung sind beide Normalformdarstellungen allerdings nur bedingt geeignet, da deren Größe, wie in Abschnitt 4.4.1 gezeigt, für die meisten Schaltfunktionen exponentiell mit der Anzahl der Eingangsvariablen wächst.

In der Praxis stellt uns die große Auswahl an Implementierungsmöglichkeiten vor ein ernst zu nehmendes Problem und die Suche nach der optimalen Darstellung ist ein wichtiger Teil der täglichen Arbeit eines Hardware-Entwicklers. In diesem Kapitel wollen wir uns deshalb mit der Frage auseinandersetzen, wie wir unter den vielen verschiedenen Möglichkeiten der Schaltungsdarstellung die optimale Variante auswählen oder von Grund auf konstruieren können.

Zur Lösung dieses Problems bleibt der Hardware-Entwickler glücklicherweise nicht auf sich alleine gestellt. Heute stehen uns zur Minimierung boolescher Ausdrücke leistungsfähige Verfahren zur Seite, mit deren Hilfe die optimale Hardware-Implementierung einer booleschen Funktion nahezu automatisiert erzeugt werden kann. Bevor wir jedoch in die Tiefen der verschiedenen Minimierungsverfahren eintauchen, wollen wir uns nochmals genauer mit der Frage beschäftigen, was wir unter dem Begriff der *optimalen Schaltung* in diesem Zusammenhang eigentlich verstehen müssen.

Hierzu betrachten wir das Schaltnetz in Abbildung 6.1, das uns bereits im vorherigen Kapitel als fruchtbares Beispiel diente. Durch die sukzessive Rückverfolgung der Ausgangsleitung  $y$  bis hin zu den Eingängen konnten wir die funktionale Formeldarstellung direkt aus dem Strukturbild ableiten:

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

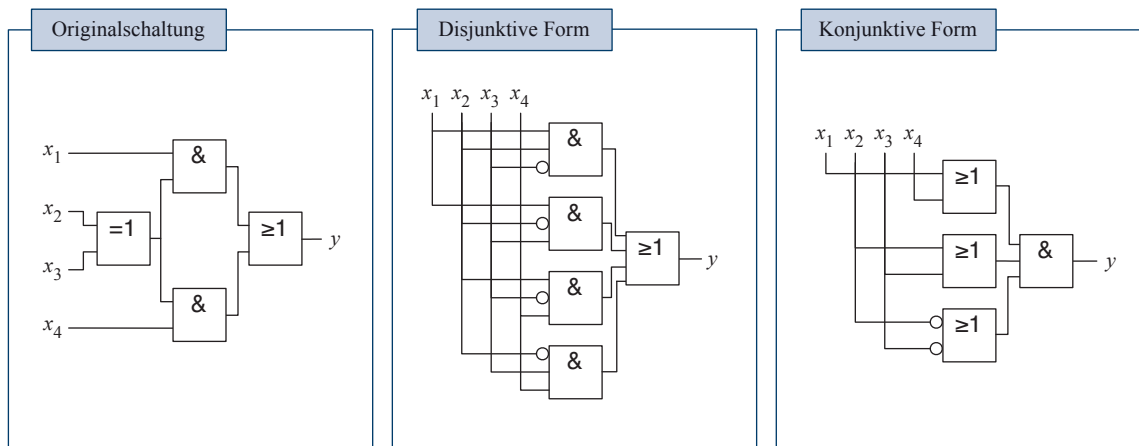


Abbildung 6.2: Die drei Beispielschaltungen im Vergleich

Indem wir den XOR-Operator durch seine disjunktive Darstellung  $x_2\bar{x}_3 \vee \bar{x}_2x_3$  ersetzen, können wir den Ausdruck wie folgt umformen:

$$\begin{aligned} & (x_1 \wedge (x_2 \oplus x_3)) \vee (x_4 \wedge (x_2 \oplus x_3)) \\ &= (x_1 \wedge (x_2\bar{x}_3 \vee \bar{x}_2x_3)) \vee (x_4 \wedge (x_2\bar{x}_3 \vee \bar{x}_2x_3)) \\ &= x_1x_2\bar{x}_3 \vee x_1\bar{x}_2x_3 \vee x_2\bar{x}_3x_4 \vee \bar{x}_2x_3x_4 \end{aligned}$$

In der transformierten Gleichung wird die boolesche Funktion durch vier konjunktive Terme aufgebaut, die auf der obersten Ebene disjunktiv verknüpft sind. In anderen Worten: Die Formel liegt in *disjunktiver Form* vor.

Die vorgenommene Schaltungstransformation ist natürlich bei weitem nicht die einzig mögliche. Genauso gut können wir die Teilformel  $x_2 \oplus x_3$  zunächst mit Hilfe des Distributivgesetzes separieren und den XOR-Operator anschließend durch den äquivalenten Ausdruck  $(x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3)$  ersetzen. Auf diese Weise erhalten wir die folgende alternative Schaltungsdarstellung:

$$\begin{aligned} & (x_1 \wedge (x_2 \oplus x_3)) \vee (x_4 \wedge (x_2 \oplus x_3)) \\ &= (x_1 \vee x_4) \wedge (x_2 \oplus x_3) \\ &= (x_1 \vee x_4) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3) \end{aligned}$$

Die drei disjunktiven Unterterme sind auf der obersten Ebene UND-verknüpft, d. h., der erzeugte boolesche Ausdruck liegt jetzt in *konjunktiver Form* vor. Zusammen mit der Originalschaltung haben wir für

Neben den klassischen Minimierungszielen *Fläche* und *Laufzeit* kommen in der Praxis weitere Minimierungsziele hinzu. Insbesondere im Bereich der mobilen Endgeräte spielt die Leistungsaufnahme einer Hardware-Schaltung heute eine zentrale Rolle, da sich der Stromverbrauch unmittelbar auf die Batterielaufzeit auswirkt.

Auch im Bereich des *Home Entertainments* (Wohnzimmer-PC) sind der Leistungsaufnahme enge Grenzen gesetzt. Die hier eingesetzten lüfterlosen Geräte können die mit zunehmendem Stromverbrauch entstehende Wärme weit weniger effizient abführen als z. B. ein aktiv gekühlter Desktop-Computer.

Originalschaltung:

$$C_A = 4$$

$$C_S = 3$$

Disjunktive Form:

$$C_A = 5$$

$$C_S = 2$$

Konjunktive Form:

$$C_A = 4$$

$$C_S = 2$$

**Tabelle 6.1:** Im Hinblick auf den Flächenverbrauch der verschiedenen Implementierungsvarianten bescheinigt die Metrik  $C_A$  sowohl der Originalschaltung als auch der konjunktiven Form die geringsten Kosten. In Bezug auf die Schaltgeschwindigkeit sind, wie die Auswertung der Metrik  $C_S$  nahelegt, sowohl die disjunktive als auch die konjunktive Variante der Originalschaltung vorzuziehen.

dieselbe boolesche Funktion damit insgesamt drei verschiedene Darstellungen erzeugt, deren direkte Umsetzungen in Hardware in Abbildung 6.2 dargestellt sind.

Bevor wir unter den drei Schaltungen die optimale Implementierungsvariante bestimmen können, müssen wir uns zunächst auf die *Kriterien* einigen, die wir der Bewertung einer Schaltung zu Grunde legen wollen. Zwei der klassischen Bewertungskriterien im industriellen Hardware-Entwurf haben wir bereits an mehreren Stellen kennen gelernt: den *Flächenbedarf* und die *Laufzeit* einer Hardware-Schaltung.

## Kostenfunktionen

In der Praxis wird das Minimierungsziel mit Hilfe einer *Kostenfunktion* modelliert, die jede Schaltung entsprechend der definierten Gütekriterien analysiert und in Form von *Kosten* quantitativ bewertet. Mit der so entstehenden Metrik können verschiedene Hardware-Implementierungen untereinander verglichen werden – je niedriger die Kosten einer Schaltung sind, desto stärker erfüllt die Schaltung die gesetzten Gütekriterien. Die weiter oben formulierten Minimierungsziele *Flächenbedarf* und *Laufzeit* lassen sich beispielsweise wie folgt quantitativ approximieren:

### ■ $C_S$ = Schaltungstiefe

Die Tiefe einer Schaltung ist definiert als die Anzahl der Logikgatter, die ein Signal von den Eingängen zu den Ausgängen *maximal* durchlaufen muss. Die Schaltungstiefe wird im Hardware-Entwurf als grobes Maß für die Geschwindigkeit einer Hardware-Implementierung eingesetzt.

### ■ $C_A$ = Gatteranzahl

Die Anzahl der Logikgatter ist ein guter Anhaltspunkt für den Flächenbedarf einer Schaltung. Inverter (NOT-Gatter) zählen wir in diesem Fall nicht zu den Gattern hinzu, da diese auf Transistorebene in Abhängigkeit der gewählten Basistechnologie ohnehin vorliegen oder sich oft ohne zusätzlichen Platzbedarf in die benachbarten Logikgatter integrieren lassen und damit keinen weiteren physikalischen Platz benötigen.

Tabelle 6.1 fasst die Auswertungsergebnisse der beiden Kostenfunktionen  $C_A$  und  $C_S$  für alle drei Implementierungsvarianten zusammen. Für

beide der verfolgten Minimierungsziele gibt es jedoch keinen eindeutigen Sieger. Offensichtlich sind die Metriken  $C_A$  und  $C_S$  noch zu grob gewählt und die Frage ist berechtigt, ob wir die Definition der beiden Kostenfunktionen nicht in die eine oder die andere Richtung präzisieren können.

Untersuchten wir die physikalische Implementierung einer Hardware-Schaltung auf Transistorebene, so würden wir feststellen, dass nicht alle Logikgatter die gleiche physikalische Größe besitzen. Gatter mit vielen Eingängen benötigen deutlich mehr Platz als Gatter mit wenigen Eingängen – die Anzahl der Eingänge eines Gatters haben wir in der Definition von  $C_A$  bisher jedoch vollständig ignoriert. Die Metrik  $C_A$  können wir verbessern, indem wir jedes Gatter mit der Anzahl seiner Eingänge gewichten. Dies führt uns direkt zur Definition der Metrik  $C'_A$ :

$$C'_A = \sum_{\text{Gatter } g} \text{Anzahl der Eingänge von } g$$

Die Laufzeitmetrik  $C_S$  können wir ebenfalls weiter präzisieren, indem wir sie mit der Flächenmetrik  $C'_A$  verschmelzen. Dabei gewichten wir die Laufzeitkosten so hoch, dass sie weiterhin dominieren und die Flächenkosten nur bei gleich schnellen Schaltungen eine Rolle spielen:

$$C'_S = (100 \times C_S) + C'_A$$

Die Auswertung der optimierten Metriken  $C'_A$  und  $C'_S$  ist in Tabelle 6.2 zusammengefasst. Jetzt sprechen die berechneten Werte eine eindeutige Sprache: Wollen wir eine kompakte und damit kostenökonomische Schaltung herstellen, so sind wir mit der Originalschaltung am besten beraten. Ist uns stattdessen die Schaltgeschwindigkeit wichtig, eignet sich die konjunktive Variante am besten. Die disjunktive Form schaltet zwar gleich schnell, verbraucht jedoch mehr Fläche und ist damit nicht mehr die erste Wahl.

Unsere Betrachtungen zeigen eindringlich, dass die Güte einer Schaltung keine allgemeingültige Eigenschaft ist, sondern in Abhängigkeit des Minimierungsziels stark variiert. Kurzum: Die Güte einer Schaltung ist relativ. Durch die Aufstellung einer geeigneten Kostenfunktion haben wir jedoch die Möglichkeit, den Erfüllungsgrad der Minimierungsziele quantitativ zu messen und damit ein Mittel an der Hand, aus verschiedenen Implementierungen die für uns optimale Schaltung auszuwählen. Ein konkretes Verfahren, mit dessen Hilfe sich eine minimierte Darstellung systematisch erstellen lässt, sind wir bisher allerdings schuldig geblieben. Genau hiermit werden wir uns im nächsten Abschnitt beschäftigen und sehen, wie sich beliebige boolesche Funktionen auf überraschend einfache Weise grafisch minimieren lassen.

In der Praxis stehen dem Hardware-Entwickler weitere Möglichkeiten zur Verfügung, um die Kostenfunktionen zu optimieren. Genau wie im Software-Entwurf auf bestehende Programmibliotheken zurückgegriffen wird, werden im Hardware-Entwurf *Zellbibliotheken* eingesetzt, die genaue Daten über die zur Verfügung stehenden Logikgatter enthalten. Neben der funktionalen Beschreibung jeder Zelle sind dort auch deren Flächenverbrauch und Schaltgeschwindigkeit exakt spezifiziert.

Wie wir bereits in Abschnitt 5.5 herausgearbeitet haben, muss zur Abschätzung der Geschwindigkeit in zunehmendem Maße auch die Leitungsverzögerung in die Kostenfunktion integriert werden. Der Grund hierfür liegt in den heute üblichen, extrem geringen Strukturweiten im zweistelligen Nanometerbereich. Mit der beständigen Zunahme der Integrationsdichte wird die Gesamtgeschwindigkeit immer mehr durch die Leitungsverzögerung (*net delay*) und immer weniger durch die Gatterverzögerung (*gate delay*) bestimmt.

Originalschaltung:

$$C'_A = 8$$

$$C'_S = 308$$

Disjunktive Form:

$$C'_A = 16$$

$$C'_S = 216$$

Konjunktive Form:

$$C'_A = 9$$

$$C'_S = 209$$

**Tabelle 6.2:** Auswertung der erweiterten Kostenfunktionen für unsere Beispielschaltungen. Die verfeinerte Metrik zeigt, dass die Originalschaltung am kompaktesten und für die Realisierung einer schnellen Schaltung die konjunktive Variante am besten geeignet ist.

## 6.2 Karnaugh-Veitch-Diagramme

	$x_4$	$x_3$	$x_2$	$x_1$	$y$
10	1	0	1	0	1
11	1	0	1	1	1



$$y = x_4 \wedge \bar{x}_3 \wedge x_2$$

	$x_4$	$x_3$	$x_2$	$x_1$	$y$
3	0	0	1	1	1
11	1	0	1	1	1



$$y = \bar{x}_3 \wedge x_2 \wedge x_1$$

	$x_4$	$x_3$	$x_2$	$x_1$	$y$
12	1	1	0	0	1
13	1	1	0	1	1



$$y = x_4 \wedge x_3 \wedge \bar{x}_2$$

	$x_4$	$x_3$	$x_2$	$x_1$	$y$
5	0	1	0	1	1
13	1	1	0	1	1



$$y = x_3 \wedge \bar{x}_2 \wedge x_1$$

	$x_4$	$x_3$	$x_2$	$x_1$	$y$
11	1	0	1	1	1
12	1	1	0	0	1



Nicht zusammenfassbar

**Tabelle 6.3:** Einige benachbarte Zeilen lassen sich durch einen einzigen booleschen Ausdruck gemeinsam repräsentieren, bei anderen ist eine solche Zusammenfassung nicht möglich.

Karnaugh-Veitch-Diagramme, kurz KV-Diagramme, sind eine spezielle grafische Darstellung für boolesche Funktionen, aus der sich eine minimierte zweistufige Schaltungsdarstellung auf einfache Weise ableiten lässt [49, 90]. Um die Idee, die hinter der grafischen Minimierung steckt, genauer zu verstehen, erinnern wir uns zunächst noch einmal an die Ableitungsregel zur Extraktion der disjunktiven Normalform einer booleschen Funktion. Ausgehend von der Wahrheitstafeldarstellung erzeugen wir für jede Variablenbelegung der *Einsmenge* einen *Minterm*, der genau für diese Variablenbelegung zu 1 und für alle anderen Variablenbelegungen zu 0 wird. Die disjunktive Normalform ergibt sich auf einen Schlag durch einfache ODER-Verknüpfung aller Minterme. Das Konstruktionsschema der disjunktiven Normalform wirft die Frage auf, ob wir wirklich für jedes Element der Einsmenge einen eigenen Minterm erzeugen müssen oder vielleicht mehrere Variablenbelegungen durch einen gemeinsamen Teilausdruck repräsentieren können. Die fünf Beispiele in Tabelle 6.3 zeigen, dass sich einige Zeilen der Wahrheitstafel durchaus durch einen einzigen Term charakterisieren lassen, andere hingegen nicht. Damit haben wir die Grundidee der Minimierung nach Karnaugh und Veitch bereits umrissen. Anders als im Fall der Normalform versuchen wir, die Elemente der Einsmenge mit *möglichst wenigen* konjunktiven Termen *vollständig* zu beschreiben. Je mehr Elemente der Einsmenge wir mit einem einzigen konjunktiven Term repräsentieren können, desto kürzer wird die erzeugte Formeldarstellung.

Das unterste Beispiel in Tabelle 6.3 zeigt, dass nicht alle Paare von Variablenbelegungen durch einen einzigen Term charakterisiert werden. Die abgebildeten Beispiele lassen vermuten, dass die Zusammenfassung genau dann möglich ist, wenn sich die Variablenbelegungen in genau einer Variablen – der sogenannten *freien Variablen* – unterscheiden:



### Definition 6.1

Gegeben seien zwei Belegungen der Variablen  $x_1, \dots, x_n$ . Die Variable  $x_i$  heißt *gebunden*, falls sie in beiden Belegungen den gleichen Wert besitzt. Ist die Variable  $x_i$  unterschiedlich belegt, so wird sie als *frei* bezeichnet. Zwei Variablenbelegungen heißen *benachbart*, wenn sie sich in genau einer freien Variablen unterscheiden.

Damit lässt sich unsere Vermutung wie folgt formulieren:


**Satz 6.1**

Zwei Variablenbelegungen lassen sich genau dann durch einen einzigen konjunktiv verknüpften Term repräsentieren, wenn sie im Sinne von Definition 6.1 benachbart sind.

Die Gültigkeit des Satzes ergibt sich ohne Umwege durch Anwendung der Rechenregeln der booleschen Algebra, wie die folgende Umformung exemplarisch an den Variablenbelegungen der zehnten und elften Zeile der Wahrheitstafel zeigt.

$$\begin{aligned}
 & (x_4 \wedge \bar{x}_3 \wedge x_2 \wedge \bar{x}_1) \vee (x_4 \wedge \bar{x}_3 \wedge x_2 \wedge x_1) \\
 &= ((x_4 \wedge \bar{x}_3 \wedge x_2) \wedge \bar{x}_1) \vee ((x_4 \wedge \bar{x}_3 \wedge x_2) \wedge x_1) \\
 &= (x_4 \wedge \bar{x}_3 \wedge x_2) \wedge (\bar{x}_1 \vee x_1) \\
 &= (x_4 \wedge \bar{x}_3 \wedge x_2) \wedge 1 \\
 &= (x_4 \wedge \bar{x}_3 \wedge x_2)
 \end{aligned}$$

In der obigen Umformung werden zunächst die gebundenen Variablen ausgeklammert. Die anschließende Anwendung der Regeln der inversen und neutralen Elemente löscht den verbleibenden Ausdruck aus und die freie Variable verschwindet. Finden wir zwei oder mehr freie Variablen vor, so lassen sich die gebundenen Variablen zwar ebenfalls ausklammern, die Variablen des Restausdrucks löschen sich jedoch nicht mehr aus. Die Zusammenfassung scheidet.

Damit haben wir eines der Kernelemente der von Karnaugh und Veitch verfolgten Minimierungsstrategie freigelegt: Durch die Zusammenfassung benachbarter Variablenbelegungen reduzieren wir sukzessive die Anzahl der zur Darstellung benötigten Teilausdrücke und erhalten auf diese Weise eine minimierte disjunktive Darstellung der Schaltfunktion. Ein Blick auf die Beispiele in Tabelle 6.3 zeigt jedoch, dass es recht schwierig ist, benachbarte Variablenbelegungen im Sinne von Definition 6.1 aus der Wahrheitstafel abzulesen. Weder sind alle benachbarten Belegungen untereinander angeordnet noch sind untereinander angeordnete Variablenbelegungen stets benachbart. Kurzum: Die Wahrheitstafeldarstellung ist zur Schaltungsminimierung denkbar ungeeignet.

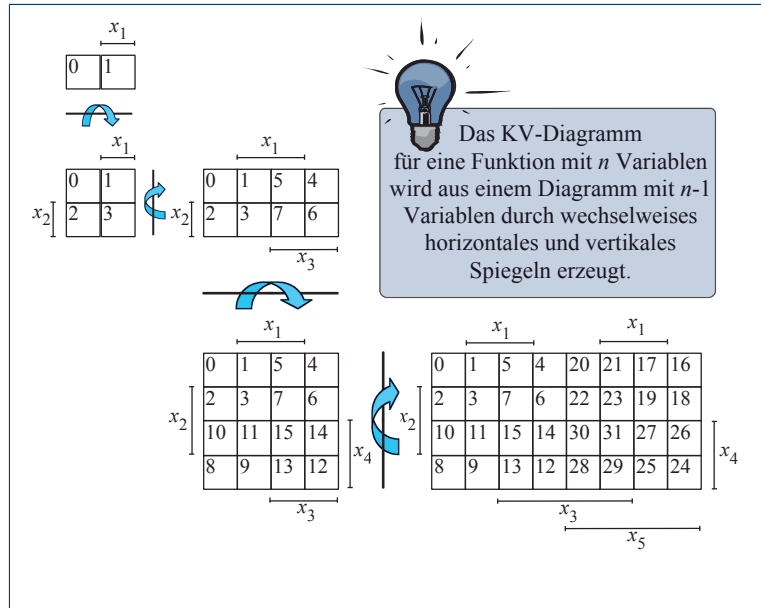
Karnaugh und Veitch erkannten als Erste, dass sich die Wahrheitswerte einer booleschen Funktion intelligenter anordnen lassen, indem wir uns von der tabellarischen Darstellung lösen und die Wahrheitswerte stattdessen in ein zweidimensionales Diagramm eintragen, das nach den Konstruktionsregeln in Abbildung 6.3 erzeugt wird. Das einfachste KV-Diagramm besteht aus genau zwei Feldern und dient zur Darstellung

Die grafische Minimierung mit Hilfe von KV-Diagrammen, wie wir sie heute kennen, geht auf die Arbeiten der beiden Amerikaner Maurice Karnaugh und Edward W. Veitch zurück, die ihre Ergebnisse bereits Anfang der Fünfzigerjahre publizierten. Das folgende Diagramm stammt aus der Originalarbeit von E. Veitch aus dem Jahre 1952 [90]:

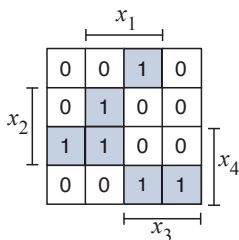
		0	0	1	1	w
		0	1	0	1	x
y	z					
0	0	x	x	x		
0	1					
1	0					x
1	1	x	x	x	x	

Veitch hatte als Erster die Idee, die Wahrheitswerte einer booleschen Funktion in einer Matrix anzuordnen. Genau wie im Falle der heute verwendeten Karnaugh-Veitch-Diagramme wird jede konkrete Variablenbelegung durch ein einzelnes Feld repräsentiert. Statt den Funktionswert 0 oder 1 in die Diagrammfelder einzutragen, markierte Veitch alle Einsfelder mit einem Kreuz. Von den Notationsfeinheiten abgesehen, unterscheiden sich die *Veitch-Diagramme* in einem wesentlichen Punkt: Wie die Abbildung zeigt, ordnete Veitch die Variablenbelegungen an den Diagrammrändern entsprechend der normalen binären Zählweise an, so dass sich die Variablenbelegungen benachbarter Felder nicht jeweils in einer einzigen Variablen unterscheiden. Erst Karnaugh entwickelte die Darstellung zur heutigen Form weiter, indem er für die Randmarkierungen nicht mehr das Binärsystem, sondern den Gray-Code als Ordnungsrelation wählte. Erst so ist sichergestellt, dass sich die Variablenbelegungen benachbarter Felder in genau einer Variablen unterscheiden. In dieser Form wurden KV-Diagramme zu dem, was sie heute sind: einem der wichtigsten Hilfsmittel der manuellen zweistufigen Schaltungsminimierung.

**Abbildung 6.3:** Das Konstruktionsschema von Karnaugh-Veitch-Diagrammen. Jedes Feld des KV-Diagramms repräsentiert eindeutig eine bestimmte Belegung der Eingangsvariablen, die sich anhand der Randmarkierung rekonstruieren lässt. Die Variable  $x_i$  wird genau dann mit 1 belegt, wenn sich das entsprechende Feld im Bereich der Randmarkierung  $x_i$  befindet. Jedes Feld eines KV-Diagramms entspricht damit genau einer Zeile in der Wahrheitstafel und kann durch Übertragung der Funktionswerte ausgefüllt werden. Die optional angegebenen Zahlen innerhalb der Felder entsprechen den Zeilennummern in der Wahrheitstafeldarstellung.



einstelliger Funktionen  $y = f(x_1)$ . Das linke Feld beschreibt den Funktionswert  $f(0)$  und das rechte Feld den Funktionswert  $f(1)$ . Größere Diagramme werden durch wechselseitiges horizontales und vertikales Spiegeln erzeugt, so dass sich in jedem Schritt die Anzahl der Variablen um eins erhöht und die Anzahl der Felder verdoppelt. Folgerichtig werden zur Konstruktion des KV-Diagramms für die Darstellung einer  $n$ -stelligen Funktion  $y = f(x_n, \dots, x_1)$  exakt  $(n - 1)$  Spiegelschritte benötigt. Abbildung 6.4 zeigt das ausgefüllte KV-Diagramm für unser Eingangsbeispiel.



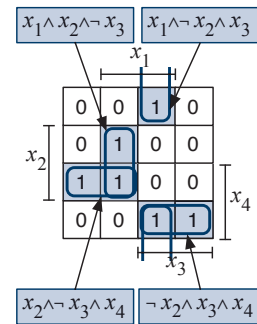
**Abbildung 6.4:** Das fertig ausgefüllte KV-Diagramm für unser Eingangsbeispiel

Durch das spiegelbasierte Konstruktionsprinzip von KV-Diagrammen ist gewährleistet, dass nebeneinanderliegende Felder im Sinne von Definition 6.1 benachbart sind – eine Eigenschaft, die sich im Übrigen auch über die Ränder des KV-Diagramms hinaus erstreckt. Hierdurch sind wir in der Lage, benachbarte Variablenbelegungen durch die Bildung von Blöcken grafisch zusammenzufassen. Angewendet auf das KV-Diagramm unseres Eingangsbeispiels erhalten wir die in Abbildung 6.5 dargestellte Blocküberdeckung. Sind alle Einsen des KV-Diagramms überdeckt, können wir daraus sofort eine disjunktive Darstellung der repräsentierten Schaltfunktion erzeugen. Dazu berechnen wir die konjunktiven Teilterme für jeden Block und verknüpfen diese anschließend mit Hilfe der ODER-Operation. Die konjunktiven Terme werden als *Implikanten* bezeichnet. Der entstandene Gesamtausdruck ist genau für

diejenigen Variablenbelegungen gleich 1, die in der Überdeckungsmenge liegen, und repräsentiert damit die gesuchte Funktion. Für unser Beispiel ergibt sich die folgende Darstellung, die exakt der weiter oben auf algebraischem Wege gewonnenen disjunktiven Form entspricht:

$$y = (x_1x_2\bar{x}_3) \vee (x_1\bar{x}_2x_3) \vee (x_2\bar{x}_3x_4) \vee (\bar{x}_2x_3x_4)$$

Das skizzierte Vorgehen zur Schaltungsminimierung wollen wir anhand der in Tabelle 6.4 spezifizierten Funktion weiter vertiefen. Genau wie in unserem ersten Beispiel konstruieren wir zunächst das KV-Diagramm durch wechselweise Spiegelung und übertragen die Werte der Wahrheitstafel. Anschließend werden benachbarte Felder der Einsmenge durch die Bildung von Zweierblöcken zusammengefasst. Wie in Abbildung 6.6 gezeigt, gibt es hier im Gegensatz zum ersten Beispiel mehrere Möglichkeiten, die Einsmenge zu überdecken. Damit haben wir eine wichtige Eigenschaft der Blockbildung herausgearbeitet:



**Abbildung 6.5:** Benachbarte Felder können zu Blöcken zusammengefasst werden. Jeder Block lässt sich durch einen einzigen konjunktiven Term beschreiben.



Die Überdeckung der Einsmenge eines KV-Diagramms ist im Allgemeinen nicht eindeutig.

Eine zweite wichtige Eigenschaft betrifft benachbarte Blöcke, die eine zentrale Eigenschaft der Einzelfelder erben:



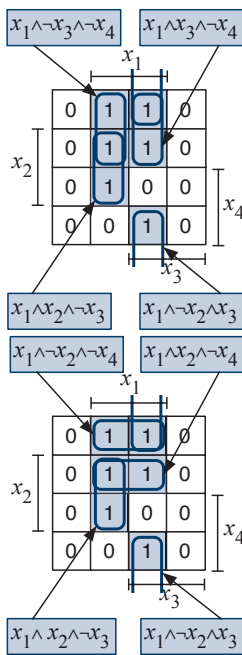
Die Implikanten gleich großer benachbarter Blöcke unterscheiden sich in genau einer Variablen.

Als Beispiel betrachten wir die benachbarten Zweierblöcke der KV-Diagramme in Abbildung 6.6. Im ersten Diagramm unterscheiden sich beide Implikanten in der Belegung der Variablen  $x_3$ , im zweiten Diagramm in der Belegung der Variablen  $x_2$ . Damit können wir auch hier Satz 6.1 anwenden und, wie in Abbildung 6.7 gezeigt, beide Blöcke zu einem Viererblock zusammenfassen. Mit jeder Verschmelzung verschwindet die freie Variable im Implikanten des neuen Blocks, so dass der neue Viererblock durch den Implikanten  $x_1 \wedge \bar{x}_4$  vollständig und eindeutig beschrieben ist.

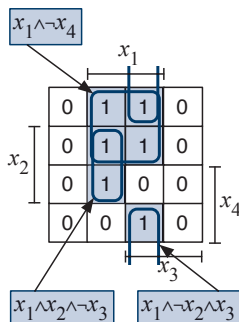
	$x_4$	$x_3$	$x_2$	$x_1$	$y$
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

**Tabelle 6.4:** Die spezifizierte Funktion ist genau dann gleich 1, wenn der aus den Eingangsvariablen gebildete Binärwert ungerade und nur durch eins und sich selbst teilbar ist.





**Abbildung 6.6:** Die Einsmenge der dargestellten booleschen Funktion kann mit Zweierblöcken auf verschiedene Weise überdeckt werden.



**Abbildung 6.7:** Benachbarte Zweierblöcke lassen sich zu Viererblöcken zusammenfassen.



### Definition 6.2

Kann ein Block in einem KV-Diagramm nicht weiter vergrößert werden, so sprechen wir von einem *Primblock*. In entsprechender Weise bezeichnen wir den zu einem Primblock gehörigen konjunktiven Term als *Primimplikant*.

Die Bildung von Primblöcken bringt uns die folgenden Vorteile:

- Mit zunehmender Blockgröße sinkt die *Anzahl der Variablen*, die zu dessen Beschreibung benötigt werden, und damit die *Größe eines einzelnen Implikanten* in der Formeldarstellung.
- Mit zunehmender Blockgröße sinkt die *Anzahl der Blöcke*, die zur Überdeckung der Einsmenge benötigt werden, und damit die *Gesamtzahl der Implikanten* in der Formeldarstellung.

Damit ist der Weg zur Schaltungsminimierung mit Hilfe von KV-Diagrammen klar umrissen. Nachdem das KV-Diagramm ausgefüllt ist, versuchen wir die Einsmenge mit einer *minimalen Anzahl* von *Primblöcken* zu überdecken. Ist die Überdeckung konstruiert, so erzeugen wir die minimierte Darstellung durch die disjunktive Verknüpfung aller zugehörigen Primimplikanten.

Damit können wir die minimierte Schaltungsdarstellung für unsere Beispielfunktion sofort aus Abbildung 6.7 ablesen:

$$y = (x_1 \wedge \bar{x}_4) \vee (x_1 \wedge x_2 \wedge \bar{x}_3) \vee (x_1 \wedge \bar{x}_2 \wedge x_3)$$

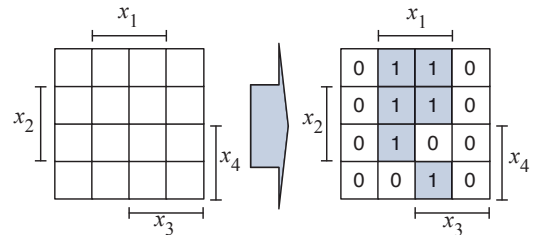
Die korrespondierende Hardware-Schaltung ist in Abbildung 6.9 zu sehen. Abschließend sind in Abbildung 6.8 die einzelnen Schritte des Minimierungsverfahrens in Form einer Übersicht zusammengefasst.

## 6.2.1 Minimierung partiell definierter Funktionen

Viele der in der Praxis auftretenden Schaltfunktionen sind nur unvollständig definiert. In anderen Worten: Für gewisse Eingabekombinationen spielt der Ausgabewert keine Rolle. Häufig treten partiell definierte Funktionen dann auf, wenn die modellierte Hardware-Komponente als Teilkomponente in eine größere Schaltung eingebettet ist und aufgrund der Struktur der Umgebungslogik nur ganz bestimmte Bitmuster an den

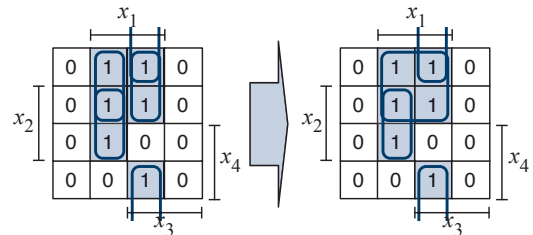
### Schritt 1: Erstellung des KV-Diagramms

Ausgehend von dem einfachsten KV-Diagramm mit zwei Feldern wird das KV-Diagramm für  $n$ -stellige Funktionen durch abwechselndes horizontales und vertikales Spiegeln erzeugt. Anschließend werden die Funktionswerte aus der Wahrheitstafel in das Diagramm eingezeichnet. Die Variablenbelegung für jedes Feld kann aus der Randmarkierung abgelesen werden.



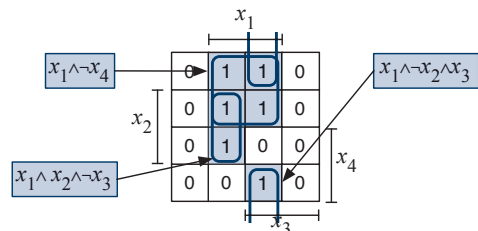
### Schritt 2: Bestimmung der Primblöcke

Jedes Feld der Einsmenge des KV-Diagramms wird mit einem Einerblock belegt. Gleich große Nachbarblöcke werden sukzessive zu immer größeren Blöcken zusammengefasst. Die Zusammenfassung wird fortgesetzt, bis keine größeren Blöcke mehr gebildet werden können und damit alle Primblöcke gefunden sind. Bei jeder Zusammenfassung verdoppelt sich die Größe des neu entstehenden Blocks.



### Schritt 3: Bestimmung einer vollständigen Überdeckung

Gesucht ist die *kleinste* Menge von Primblöcken, die zusammen die Einsmenge *vollständig* überdecken. Dazu wählen wir zunächst alle Primblöcke aus, die ein einzelnes Feld der Einsmenge *alleine* überdecken. Reichen diese noch nicht zur Überdeckung aller Einsfelder aus, nehmen wir weitere Primblöcke hinzu, bis eine vollständige Überdeckung erreicht ist.

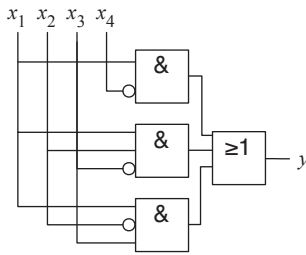


### Schritt 4: Extraktion der disjunktiven Minimalform

Jeder Primblock mit  $2^k$  Feldern wird durch einen Primimplikanten mit  $n - k$  Variablen charakterisiert. Die *disjunktiven Minimalform* erhalten wir durch einfache ODER-Verknüpfung aller Primimplikanten. Im Allgemeinen gibt es mehrere minimale Überdeckungen für die Einsmenge einer Funktion, so dass auch die disjunktive Minimalform nicht immer eindeutig ist.

$$y = (x_1 \wedge \bar{x}_4) \vee (x_1 \wedge x_2 \wedge \bar{x}_3) \vee (x_1 \wedge \bar{x}_2 \wedge x_3)$$

Abbildung 6.8: Die Minimierung nach Karnaugh und Veitch im Überblick



**Abbildung 6.9:** Hardware-Umsetzung der minimierten disjunktiven Form unserer Beispielfunktion

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

**Tabelle 6.5:** Beschränken wir die Menge der Eingabekombinationen auf die Menge der gültigen BCD-Ziffern, so erhalten wir durch die entstehenden Don't-Care-Belegungen zusätzliche Freiheitsgrade für die Minimierung.

Eingängen generiert werden. In anderen Fällen beschränkt die Spezifikation einer Hardware-Schaltung die auftretenden Eingabekombinationen auf bestimmte Bitmuster.


So könnte beispielsweise die Spezifikation unserer Beispielfunktion aus Abbildung 6.4 regeln, dass die Schaltung ausschließlich für die Klassifikation von BCD-Ziffern verwendet werden darf. Da die gültigen Eingabekombinationen damit nur noch die Zahlen im Bereich von 0 bis 9 umfassen, muss die konstruierte Schaltung nur noch für die Bitmuster 0000 (0) bis 1001 (9) eine definierte Ausgabe erzeugen. Für alle anderen Bitmuster, die sogenannten *Don't-Care-Belegungen*, können wir die Ausgabe stattdessen nach Belieben auf 0 oder 1 setzen. Obwohl der gewählte Wahrheitswert der Don't-Care-Kombinationen für das korrekte Funktionieren keine Rolle spielt, ist eine vorzeitige Festlegung auf einen speziellen Wert nicht ratsam – wir würden uns auf einen Schlag eines erheblichen Minimierungspotenzials entledigen. Aus diesem Grund markieren wir die Funktionswerte aller Don't-Care-Belegungen zunächst mit einem Bindestrich („-“), wie in Tabelle 6.5 dargestellt.

Genau wie oben tragen wir jetzt die Funktionswerte in das durch wechselseitige Spiegelung erzeugte KV-Diagramm ein und fassen benachbarte Felder zu immer größeren Blöcken zusammen. Für alle Felder, die mit „-“ markiert sind, haben wir die freie Wahl, ob wir sie mit einem Block überdecken oder unberücksichtigt lassen. Damit wird das Optimierungspotenzial deutlich, das durch die unvollständige Definition einer booleschen Funktion entsteht. Eine Don't-Care-Kombination nehmen wir genau dann zur Einsmenge hinzu, wenn wir dadurch größere Blöcke und damit eine kürzere Formelardarstellung erhalten.

Abbildung 6.10 zeigt, dass wir für unsere Beispielfunktion eine minimale Überdeckung erhalten, wenn wir *keine* der Don't-Care-Belegungen in die Blockbildung einbeziehen. Damit ist die Schaltfunktion für alle Belegungen der Don't-Care-Menge gleich 0. Wie im abgebildeten KV-Diagramm zu erkennen ist, können wir die Einsmenge dann mit einem einzigen Primblock überdecken und erhalten die folgende reduzierte Schaltungsdarstellung:

$$y = x_1 \wedge \overline{x_4}$$

Das Beispiel zeigt, dass durch die geschickte Ausnutzung der Don't-Care-Belegungen die Größe der Hardware-Implementierung deutlich reduziert werden kann. Die modifizierte Schaltung besteht, wie in Abbildung 6.11 dargestellt, nur noch aus einem einzigen UND-Gatter mit zwei Eingängen und benötigt damit weniger als ein Viertel des Platzbedarfs der Originalschaltung. Wir halten die Strategie für die Minimierung partiell definierter Funktionen wie folgt fest:

 Don't-Care-Belegungen werden zunächst in das KV-Diagramm übernommen. Die Funktionswerte werden während der Blockbildung so gewählt, dass maximal große Primblöcke entstehen.

**Inverse Blockbildung**

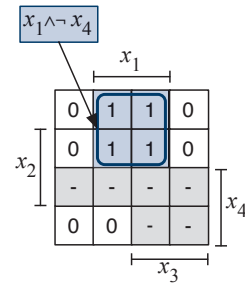
Die Minimierung einer Schaltfunktion mit Hilfe von KV-Diagrammen produziert eine minimierte Schaltungsdarstellung in disjunktiver Form. Die Auswahl der kleinsten Menge von Primblöcken zur Überdeckung aller Einsfelder stellt sicher, dass es keine andere disjunktive Form gibt, mit der die gleiche Schaltfunktion kürzer dargestellt werden kann. In anderen Worten: Das Verfahren von Karnaugh und Veitch liefert uns eine disjunktive Minimalform. Mit einem kleinen Trick können wir mit Hilfe des gleichen Verfahrens neben einer disjunktiven Minimalform auch eine konjunktive Minimalform aus dem KV-Diagramm ableiten. Dazu konstruieren wir das KV-Diagramm wie gewohnt, überdecken im Zuge der Blockbildung jedoch nicht die Einsmenge, sondern die Nullmenge der zu minimierenden Funktion.

Zur vollständigen Überdeckung der Nullmenge unserer Primzahlfunktion sind drei Blöcke notwendig, wie das entsprechende KV-Diagramm in Abbildung 6.12 zeigt. Haben wir eine vollständige Überdeckung gefunden, können wir für jeden Block, wie im Falle der disjunktiven Form, einen Primimplikanten ableiten und auf oberster Ebene mit Hilfe der ODER-Operation verbinden. Da wir aber im Gegensatz zu oben die Nullmenge und nicht die Einsmenge überdeckt haben, bildet die resultierende disjunktive Form nicht die Funktion  $y$ , sondern deren Negation  $\bar{y}$  ab. Fassen wir alle Primimplikanten aus Abbildung 6.12 wie gewohnt zusammen, so erhalten wir das folgende Zwischenergebnis:

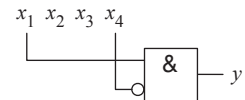
$$\bar{y} = (\bar{x}_1) \vee (\bar{x}_2 \wedge \bar{x}_3 \wedge x_4) \vee (x_2 \wedge x_3 \wedge x_4)$$

Obwohl die Funktion in disjunktiver Form vorliegt, sind wir fast am Ziel. Mit Hilfe der Doppelnegation und der Regel von De Morgan können wir die disjunktive Minimalform der Funktion  $\bar{y}$  auf einen Schlag in die gesuchte konjunktive Minimalform der Originalfunktion  $y$  überführen:

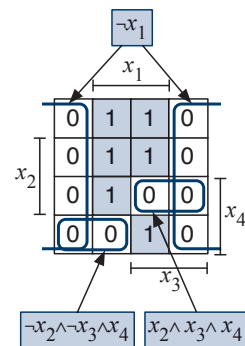
$$\begin{aligned} y &= \overline{\bar{y}} = \overline{(\bar{x}_1) \vee (\bar{x}_2 \wedge \bar{x}_3 \wedge x_4) \vee (x_2 \wedge x_3 \wedge x_4)} \\ &= \overline{(\bar{x}_1)} \wedge \overline{(\bar{x}_2 \wedge \bar{x}_3 \wedge x_4)} \wedge \overline{(x_2 \wedge x_3 \wedge x_4)} \\ &= (x_1) \wedge (x_2 \vee x_3 \vee \bar{x}_4) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) \end{aligned}$$



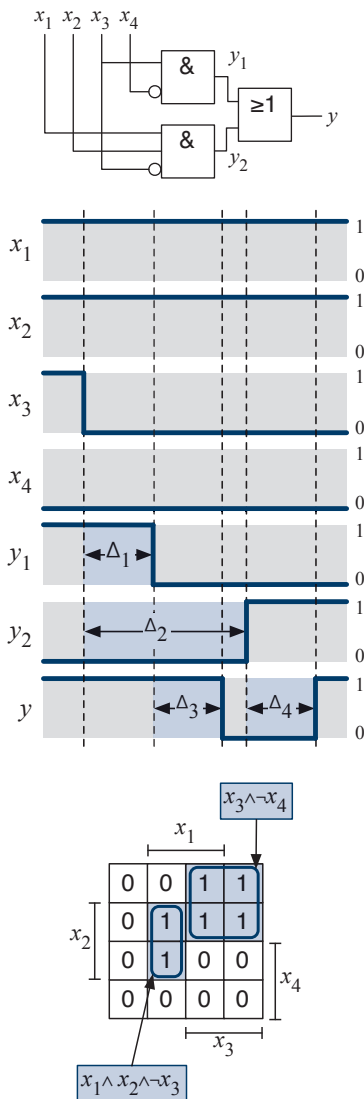
**Abbildung 6.10:** KV-Diagramm unserer modifizierten Beispielfunktion



**Abbildung 6.11:** Hardware-Implementierung unserer Beispielfunktion unter geschickter Ausnutzung der Don't-Care-Belegungen



**Abbildung 6.12:** Erster Schritt zur Erzeugung einer konjunktiven Minimalform: Überdeckung der Nullmenge



**Abbildung 6.13:** Das abgebildete Schaltnetz produziert für den dargestellten Signalwechsel einen Logik-Hazard, wenn die Schaltzeit  $\Delta_2$  größer ist als die Summe der Schaltzeiten  $\Delta_1$  und  $\Delta_3$ . Im KV-Diagramm lassen sich potenzielle Hazards auf einfache Weise erkennen.

Es gilt an dieser Stelle unbedingt zu beachten, dass sich die Polarität aller Variablen durch die Anwendung der Regel von De Morgan umkehrt. Da sich jedoch weder die Anzahl der Primimplikanten ändert noch Variablen hinzukommen oder entfallen, können wir die konjunktive Normalform auch direkt aus dem Diagramm ablesen. Bei der Bestimmung der Primimplikanten müssen wir im Vergleich mit dem Originalverfahren daher die Variablen einfach in umgekehrter Polarität aufnehmen und mit Hilfe der ODER- anstelle der UND-Verknüpfung verbinden. Anschließend werden alle Primimplikanten auf oberster Ebene mit Hilfe der UND-Verknüpfung zu einer konjunktiven Minimalform verbunden.

### 6.2.2 Konstruktion Hazard-freier Schaltungen

Neben der Erzeugung einer minimierten Darstellung in disjunkti- ver oder konjunktiver Form können KV-Diagramme dazu verwendet werden, bestimmte Aspekte des Schaltverhaltens einer Hardware-Implementierung zu untersuchen. Insbesondere eröffnet uns die spezielle visuelle Anordnung der Funktionswerte die Möglichkeit, mit wenigen Blicken zu erkennen, ob ein Schaltnetz anfällig gegenüber Logik-Hazards ist. Wie wir in Abschnitt 5.5.2 bereits gelernt haben, sprechen wir von einem Logik-Hazard immer dann, wenn der Wechsel eines einzigen Eingangssignals kurzzeitig zu einem Wechsel des Ausgangssignals führen kann, obwohl der Wert des Schaltnetzausgangs nach den Regeln der booleschen Algebra konstant bleiben müsste. Als Grund für die auftretenden *Störimpulse (Hazards)* haben wir die Laufzeitdifferenzen der verschiedenen Signalwege ausgemacht.

Wir wollen uns der Problematik der Logik-Hazards nun von der Seite der KV-Diagramme aus nähern und betrachten hierzu erneut die Hardware-Schaltung, die wir bereits in Abschnitt 5.5.2 zur Demonstration eines Logik-Hazards herangezogen haben. Das Schaltnetz und der verursachende Signalverlauf sind zusammen mit dem entsprechenden KV-Diagramm in Abbildung 6.13 dargestellt.

Der Störimpuls entstand, da wir in unserem Beispiel die Verzögerungszeit  $\Delta_2$  größer als die Summe der Verzögerungszeiten  $\Delta_1$  plus  $\Delta_3$  wählten. Sind die Signale  $x_1, x_2, x_3$  auf 1 und  $x_4$  auf 0, so verursacht der Wechsel von  $x_3$  auf 0 einen Logik-Hazard am Schaltnetzausgang  $y$ . Ein genauerer Blick auf den Signalverlauf offenbart, dass der Störimpuls genau deshalb entsteht, da sich aufgrund der Signallaufzeiten *alle* Ausgänge der UND-Stufe kurzzeitig im Zustand 0 befinden.

Hier kommt uns die Anordnung der Funktionswerte innerhalb des KV-

Diagramms zugute und ermöglicht, das potenzielle Auftreten eines Logik-Hazards einfach zu erkennen. Den im Zeitdiagramm festgehaltenen Störimpuls haben wir erzeugt, indem wir die Belegung der Eingangsvariablen  $(x_1, x_2, x_3, x_4)$  von  $(1, 1, 1, 0)$  auf  $(1, 1, 0, 0)$  wechseln ließen. Die Belegung der Eingangssignale mit  $(1, 1, 1, 0)$  wird im KV-Diagramm durch das Feld in der zweiten Zeile und dritten Spalte repräsentiert. Wie in Abbildung 6.14 dargestellt, entspricht der Wechsel der Variablen  $x_3$  von 1 auf 0 der Bewegung um ein Feld nach links.

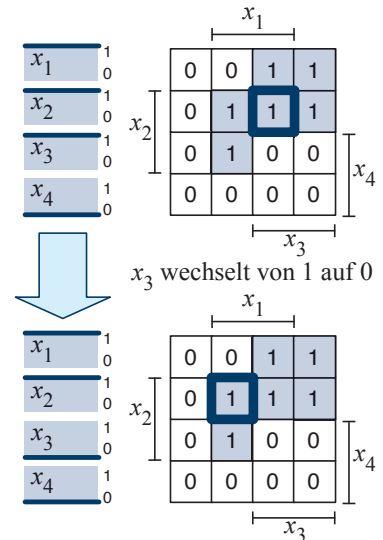
Werfen wir jetzt einen Blick auf die von uns gebildeten Primblöcke, so wird klar, warum die Ausgänge beider UND-Gatter gleichzeitig den Wert 0 einnehmen können: Beide Blöcke liegen *überlappungsfrei* nebeneinander. Durch das Verlassen des rechten Blocks  $x_3 \wedge \bar{x}_4$  fällt das Signal  $y$  auf 0 und wird mit dem gegebenen zeitlichen Verhalten erst später durch das Betreten des linken Blocks  $x_1 x_2 \bar{x}_3$  wieder zu 1. Wir halten unsere Beobachtung wie folgt fest:



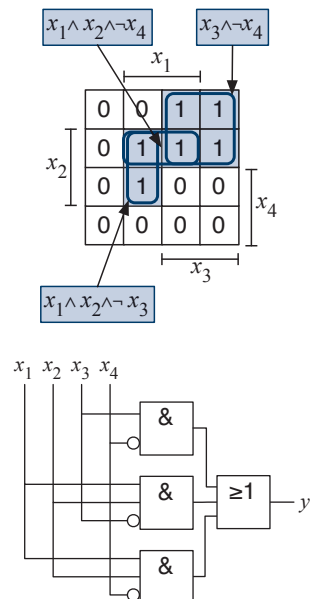
Ein Logik-Hazard kann immer dann entstehen, wenn zwei Primblöcke im KV-Diagramm überlappungsfrei aneinandergrenzen.

Anders formuliert bedeutet die angestellte Überlegung nichts anderes, als dass Logik-Hazards effektiv verhindert werden, wenn für jeden Übergang zwischen zwei Eingangsbelegungen der Einsmenge stets ein Gatter existiert, dessen Ausgangssignal für beide Belegungen gleich 1 ist. In anderen Worten: Eine Schaltfunktion ist gegen Logik-Hazards abgesichert, falls alle Paare benachbarter Einsfelder im KV-Diagramm mindestens einem gemeinsamen Primblock angehören.

Wir bekommen hiermit unmittelbar ein Verfahren zur Seite gestellt, mit dem wir eine Schaltfunktion nachträglich gegen Logik-Hazards absichern können. Anhand des KV-Diagramms überprüfen wir zunächst, ob die Primblöcke so gewählt sind, dass überlappungsfreie Einsübergänge existieren. Ist dies der Fall, überdecken wir sukzessive jede dieser *Nahtstellen* mit einem zusätzlichen Block. Für unsere Beispielschaltung können wir den einzigen überlappungsfreien Übergang beseitigen, indem wir, wie in Abbildung 6.15 gezeigt, den Zweierblock  $x_1 \wedge x_2 \wedge \bar{x}_4$  als weiteren Primblock hinzunehmen. Die entstehende Hardware-Implementierung ist ebenfalls dargestellt und entspricht exakt der bereits in Abschnitt 5.5.2 ohne Begründung eingeführten Implementierung.

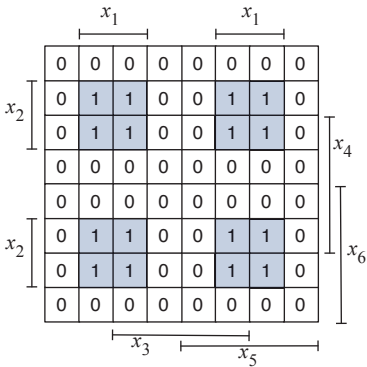
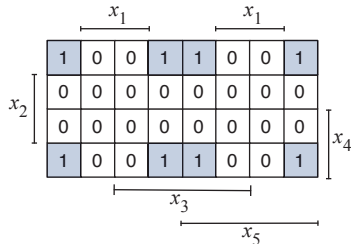


**Abbildung 6.14:** Der Wechsel des Signals  $x_3$  von 1 auf 0 entspricht im KV-Diagramm der Bewegung um ein Feld nach links.

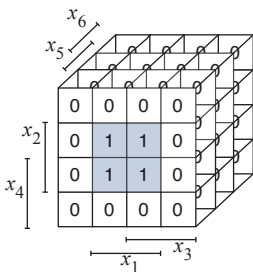
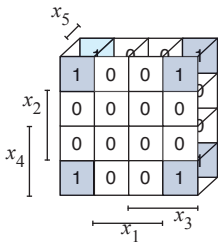


**Abbildung 6.15:** Beseitigung der Hazard-Problematik durch Hinzufügen eines weiteren Blocks

■ Zweidimensionale Darstellung:



■ Dreidimensionale Darstellung:



**Abbildung 6.16:** In KV-Diagrammen mit fünf oder mehr Variablen wird die Blockbildung schwierig.

### 6.2.3 Minimierung mehrstelliger Funktionen

Alle der bisher betrachteten Beispielschaltungen konnten wir mit Hilfe von KV-Diagrammen auf recht einfache und vor allem effiziente Weise in eine disjunktive oder konjunktive Minimalform überführen. Die Schaltnetze, die wir zur Minimierung herangezogen haben, waren jedoch allesamt vergleichsweise klein, so dass wir uns in diesem Abschnitt mit der Frage beschäftigen wollen, ob sich Schaltfunktionen mit fünf oder mehr Eingangsvariablen genauso einfach minimieren lassen.

Dazu betrachten wir die beiden KV-Diagramme in Abbildung 6.16. Wie im Falle kleinerer Diagramme sind die Variablenbelegungen zweier nebeneinander angeordneter Felder im Sinne von Definition 6.1 benachbart. In anderen Worten: Sie unterscheiden sich in der Belegung von genau einer Variablen. In Diagrammen mit bis zu vier Variablen galt bisher auch die Umkehrung, d. h., die Felder benachbarter Variablenbelegungen grenzen im KV-Diagramm unmittelbar aneinander. Genau diese Eigenschaft geht in Diagrammen mit fünf oder mehr Variablen verloren, mit einschneidenden Konsequenzen für die Blockbildung: Da die Felder benachbarter Variablenbelegungen jetzt weit voneinander entfernt angeordnet sein können, setzen sich viele Blöcke aus mehreren Fragmenten zusammen. Konnten wir die Primblöcke in kleineren Diagrammen mit nahezu einem einzigen Blick ausmachen, so müssen wir jetzt schon genauer hinschauen, um alle zusammenfassbaren Felder zu erkennen. So lassen sich die Einsmengen beider Beispielfunktionen aus Abbildung 6.16 (oben) mit einem einzigen Block überdecken und wir erhalten die folgende Formeldarstellung:

Erstes Diagramm:  $= \bar{x}_1 \wedge \bar{x}_2$

Zweites Diagramm:  $= x_1 \wedge x_2$

Eine Möglichkeit, das Nachbarschaftsproblem für Diagramme mit fünf oder mehr Variablen zu lösen, ist die dreidimensionale Anordnung der einzelnen Felder, wie in Abbildung 6.16 (unten) gezeigt. Diese Art der Darstellung besitzt jedoch zwei wesentliche Nachteile. Zum einen lässt sich die räumliche Verteilung der Diagrammfelder nur schwierig auf Papier bringen. Zum anderen ist die Erkennung von Blöcken über drei Dimensionen hinweg ein recht schwieriges Unterfangen, so dass diese Diagrammart für die praktische Arbeit insgesamt keine ernsthafte Alternative bietet. Ohnehin wird das Problem durch das Ausweichen in die dritte Dimension lediglich verlagert, da sich auf diese Weise nur Funktionen mit bis zu sechs Variablen darstellen lassen.

## 6.3 Quine-McCluskey-Verfahren

Mit den KV-Diagrammen haben wir ein effizientes Hilfsmittel zur Minimierung boolescher Funktionen kennen gelernt. Obwohl die Anwendung des Verfahrens auf Funktionen mit fünf oder sechs Variablen immer noch möglich ist, wird die Minimierung durch die kompliziertere Blockbildung erheblich erschwert. Betrachten wir noch größere Funktionen mit sieben oder mehr Variablen, so stößt die grafische Minimierung vollends an ihre Grenzen.

Genau hier setzen die tabellarischen Minimierungsverfahren an, mit deren Hilfe auch vielstellige Funktionen minimiert werden können. Das Quine-McCluskey-Verfahren (QMCV) ist deren bekanntester Vertreter und geht auf die bereits Mitte der Fünfzigerjahre veröffentlichten Arbeiten der beiden Amerikaner Willard Van Orman Quine und Edward J. McCluskey zurück [60, 75]. Die Minimierung nach Quine und McCluskey läuft in drei Schritten ab.

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

### Schritt 1: Konstruktion der ersten Quine'schen Tabelle

Ähnlich der Minimierung mit Hilfe von KV-Diagrammen werden im Verfahren von Quine und McCluskey benachbarte Variablenbelegungen zu immer größeren Blöcken zusammengefasst. Die Blöcke werden dabei jedoch nicht grafisch markiert, sondern in Form von Tabelleneinträgen untereinander aufgelistet. Die erste Quine'sche Tabelle wird konstruiert, indem zunächst alle Implikanten nullter Ordnung bestimmt werden. Ein solcher Implikant repräsentiert exakt eine Variablenbelegung der Einsmenge und entspricht damit genau einer einzigen Zeile der Wahrheitstafel. Folgerichtig können wir die Implikanten, wie in Tabelle 6.6 gezeigt, ohne weiteres Zutun aus der Wahrheitstafel der zu minimierenden Funktion ablesen.

Ausgehend von den Implikanten nullter Ordnung konstruieren wir, wie in Tabelle 6.7 dargestellt, die Implikanten der Ordnung eins, zwei usw., indem wir alle Variablenbelegungen, die sich in genau einer Variablen unterscheiden, zusammenfassen und in die Tabelle aufnehmen. Zur einfacheren Orientierung tragen wir in die erste Spalte jeweils die Indizes der zusammengefassten Zeilen ein und markieren die zusammengefassten Variablenbelegungen zusätzlich mit einem Haken. Damit das Verfahren eine optimale Lösung produziert, müssen wir darauf achten, *alle* möglichen Implikanten zu berechnen. Das bedeutet, dass wir zur Konstruktion der Implikanten der Ordnung  $n + 1$  *alle Paare* von Implikanten



	$x_4$	$x_3$	$x_2$	$x_1$
1	0	0	0	1
3	0	0	1	1
5	0	1	0	1
7	0	1	1	1
11	1	0	1	1
13	1	1	0	1

**Tabelle 6.6:** Als Vorbereitung zur Konstruktion der ersten Quine'schen Tabelle werden alle Variablenbelegungen der Einsmenge aus der Wahrheitstafel extrahiert.



Quine'sche Tabelle  
nullter Ordnung:

	$x_4$	$x_3$	$x_2$	$x_1$	
1	0	0	0	1	✓
3	0	0	1	1	✓
5	0	1	0	1	✓
7	0	1	1	1	✓
11	1	0	1	1	✓
13	1	1	0	1	✓

Quine'sche Tabelle  
erster Ordnung:

	$x_4$	$x_3$	$x_2$	$x_1$	
1,3	0	0	-	1	✓
1,5	0	-	0	1	✓
3,7	0	-	1	1	✓
3,11	-	0	1	1	
5,7	0	1	-	1	✓
5,13	-	1	0	1	

Quine'sche Tabelle  
zweiter Ordnung:

	$x_4$	$x_3$	$x_2$	$x_1$
1,3,5,7	0	-	-	1

**Tabelle 6.7:** Die vollständig entwickelte Quine'sche Tabelle für unsere Beispielfunktion

der Ordnung  $n$  untersuchen müssen. Des Weiteren gibt es zu beachten, dass Implikanten der Ordnung zwei oder höher während des Aufbaus der ersten Quine'schen Tabelle stets mehrfach erzeugt werden. Um Duplikate zu vermeiden, wird die Quine'sche Tabelle daher nur mit neuen Implikanten erweitert.

Zwischen den Einträgen der Quine'schen Tabelle und den Blöcken in KV-Diagrammen besteht eine direkte Beziehung. Jeder Implikant der ersten Quine'schen Tabelle der Ordnung  $n$  entspricht einem Block im KV-Diagramm, der exakt  $2^n$  Felder überdeckt. Auch die Primimplikanten besitzen einen direkten Partner in der ersten Quine'schen Tabelle – sie entsprechen genau denjenigen Einträgen, die nicht mehr weiter zusammengefasst werden können und daher nicht mit einem Haken markiert sind.

### Schritt 2: Konstruktion der Primimplikantentafel

Ist die erste Quine'sche Tabelle vollständig aufgebaut, werden alle Primimplikanten in die zweite Quine'sche Tabelle – die *Primimplikantentafel* – übertragen. Die zweite Quine'sche Tabelle enthält für jede Variablenbelegung der Einsmenge eine separate Spalte sowie für jeden Primimplikanten eine eigene Zeile. Die von einem Primimplikanten abgedeckten Variablenbelegungen werden in der entsprechenden Spalte mit einem Kreuz markiert. Für unsere Beispielfunktion ist die zweite Quine'sche Tabelle in Tabelle 6.8 dargestellt.

### Schritt 3: Konstruktion einer minimalen Überdeckung

Ist die zweite Quine'sche Tabelle ausgefüllt, so versuchen wir, analog zur grafischen Minimierung mit KV-Diagrammen, die Einsmenge mit einer minimalen Anzahl von Primimplikanten zu erfassen. Folgerichtig gehen wir auch hier in zwei Schritten vor:

- Zunächst bestimmen wir diejenigen Primimplikanten, die eine Variablenbelegung *alleine* überdecken, da wir diese zur Funktionsdarstellung auf jeden Fall benötigen. In der Primimplikantentafel sind diese Belegungen mit dem puren Auge zu erkennen – wir müssen lediglich nach Spalten suchen, die mit einem einzigen Kreuz markiert sind. In unserem Beispiel decken die Primimplikanten  $(-,0,1,1)$ ,  $(-,1,0,1)$ ,  $(0,-,-,1)$  jeweils eine bestimmte Variablenbelegung alleine ab und werden daher vorab ausgewählt.

	$x_4$	$x_3$	$x_2$	$x_1$	1	3	5	7	11	13
3,11	-	0	1	1		×			×	
5,13	-	1	0	1			×			×
1,3,5,7	0	-	-	1	×	×	×	×		

**Tabelle 6.8:** In der zweiten Quine'schen Tabelle werden alle Primimplikanten zeilenweise angeordnet. Die Kreuze markieren alle von einem Primimplikanten abgedeckten Elemente der Einsmenge.

- Jetzt fügen wir so lange weitere hinzu, bis eine vollständige Überdeckung erreicht wird. In unserem Beispiel decken bereits die im ersten Schritt ausgewählten Primimplikanten alle Elemente der Einsmenge ab, so dass wir keine weiteren Implikanten benötigen.

Anschließend werden die ausgewählten Primimplikanten in ihre Formeldarstellung übersetzt und disjunktiv verknüpft. So erhalten wir auf einen Schlag eine disjunktive Minimalform. Angewendet auf unsere Beispielfunktion, erhalten wir exakt die gleiche Lösung wie im Fall der Minimierung mit Hilfe von KV-Diagrammen:

$$y = (x_1 \wedge \bar{x}_4) \vee (x_1 \wedge x_2 \wedge \bar{x}_3) \vee (x_1 \wedge \bar{x}_2 \wedge x_3)$$

Im direkten Vergleich mit der grafischen Minimierung nach Karnaugh und Veitch besticht das Verfahren von Quine und McCluskey vor allem durch seine gute Automatisierbarkeit und findet sich in abgewandelter Form in vielen Algorithmen der computergestützten Schaltungssynthese wieder. Bedingt durch seine mechanische Natur kann das Verfahren auf boolesche Ausdrücke mit einer beliebigen Anzahl Variablen angewendet werden.

### Minimierung partiell definierter Funktionen

Partiell definierte Funktionen können ebenfalls mit Hilfe des Quine-McCluskey-Verfahrens minimiert werden. Um das Potenzial der Don't-Care-Belegungen optimal zu nutzen, dürfen wir uns entsprechend der Minimierung mit KV-Diagrammen nicht frühzeitig auf einen festen Funktionswert festlegen. Deshalb nehmen wir alle Don't-Care-Belegungen, wie in Tabelle 6.9 demonstriert, zunächst in die erste Quine'sche Tabelle mit auf. Anschließend vervollständigen wir die Tabelle wie bisher durch die sukzessive Zusammenfassung passender Implikanten. Das Beispiel lässt deutlich werden, dass durch die Hinzunahme aller Don't-Care-Belegungen innerhalb der ersten Quine'schen Tabelle erheblich mehr Kombinationsmöglichkeiten entstehen. Je mehr zusammenfassbare Implikanten vorhanden sind, desto höher wird die Wahrscheinlichkeit, auch größere Implikanten bilden zu können.

Die Minimierung boolescher Funktionen fällt in die Klasse der *NP-harten* Probleme. Dies bedeutet für die Praxis, dass die Laufzeit eines Algorithmus, der das Minimierungsproblem exakt löst, exponentiell mit der Anzahl der Variablen der zu minimierenden Schaltfunktion wächst. Genau wie die Methode von Karnaugh und Veitch gehört auch die Vorgehensweise von Quine und McCluskey zu den exakten Verfahren, so dass die Komplexität des Minimierungsproblems die praktische Anwendbarkeit der Algorithmen stark limitiert. In der Praxis wird das Quine-McCluskey-Verfahren deshalb nicht in seiner Reinform eingesetzt. Stattdessen wird der Algorithmus mit zahlreichen Heuristiken kombiniert, die das exponentielle Anwachsen von Speicherplatz und Rechenzeit verhindern. Die Exaktheit des Verfahrens wird aus Komplexitätsgründen bewusst aufgegeben, mit der Konsequenz, dass nicht mehr in jedem Fall die optimale Lösung gefunden wird. Zu den bekanntesten heuristischen Verfahren zur Schaltungsminimierung gehört das an der UC Berkeley entwickelte Espresso-System [63], das sowohl im akademischen als auch im industriellen Umfeld gleichermaßen Verwendung findet.

Implikanten nullter Ordnung:					Implikanten erster Ordnung:					Implikanten zweiter Ordnung:						
	$x_4$	$x_3$	$x_2$	$x_1$		$x_4$	$x_3$	$x_2$	$x_1$		$x_4$	$x_3$	$x_2$	$x_1$		
1	0	0	0	1	✓	1,3	0	0	-	1	✓	1,3,5,7	0	-	-	1
3	0	0	1	1	✓	1,5	0	-	0	1	✓	3,7,11,15	-	-	1	1
5	0	1	0	1	✓	3,7	0	-	1	1	✓	12,13,14,15	1	1	-	-
7	0	1	1	1	✓	3,11	-	0	1	1	✓	5,7,13,15	-	1	-	1
10	1	0	1	0	✓	5,7	0	1	-	1	✓	10,11,14,15	1	-	1	-
11	1	0	1	1	✓	5,13	-	1	0	1	✓					
12	1	1	0	0	✓	7,15	-	1	1	1	✓					
13	1	1	0	1	✓	10,11	1	0	1	-	✓					
14	1	1	1	0	✓	10,14	1	-	1	0	✓					
15	1	1	1	1	✓	11,15	1	-	1	1	✓					
						12,13	1	1	0	-	✓					
						12,14	1	1	-	0	✓					
						13,15	1	1	-	1	✓					
						14,15	1	1	1	-	✓					

	$x_4$	$x_3$	$x_2$	$x_1$	1	3	5	7	10	11	12	13	14	15
1,3,5,7	0	-	-	1	×	×	×	×						
3,7,11,15	-	-	1	1		×		×		×				×
12,13,14,15	1	1	-	-							×	×	×	×
5,7,13,15	-	1	-	1			×	×				×		×
10,11,14,15	1	-	1	-					×	×			×	×

**Tabelle 6.9:** Boolesche Minimierung von Don't-Care-Funktionen mit dem Quine-McCluskey-Verfahren. Die oberen drei Tabellen bilden zusammen die erste Quine'sche Tabelle. Unten ist die Primimplikantentafel abgebildet.

Ist die erste Quine'sche Tabelle vollständig konstruiert, übertragen wir alle berechneten Primimplikanten in die zweite Quine'sche Tabelle und bestimmen wie gewohnt eine minimale Überdeckung der *Einsmenge*. An dieser Stelle gilt es zu beachten, dass die Variablenbelegungen der Don't-Care-Menge nicht überdeckt werden *müssen*. Zur besseren Unterscheidung sind die entsprechenden Belegungen in Tabelle 6.9 grau unterlegt. Ein Blick auf die zweite Quine'sche Tabelle zeigt, dass die Elemente der Einsmenge bereits durch einen einzigen Primimplikanten abgedeckt werden, und wir erhalten mit

$$y = x_1 \wedge \overline{x_4}$$

die gleiche Minimalform, die wir bereits weiter oben mit der Hilfe von KV-Diagrammen ermittelt haben.

## 6.4 Übungsaufgaben

### Aufgabe 6.1



**Webcode**  
**5865**

Ein KV-Diagramm für  $n$ -stellige Funktionen wird durch (1)\_\_\_\_\_faches wechselseitiges (2)\_\_\_\_\_ und (3)\_\_\_\_\_ spiegeln konstruiert. Ein KV-Diagramm für  $n$ -stellige Funktionen enthält genau (4)\_\_\_\_\_ Felder. Zwei Blöcke lassen sich genau dann zu einem größeren Block zusammenfassen, falls sie (5)\_\_\_\_\_ und (6)\_\_\_\_\_ sind. Ein Block, der nicht mehr vergrößert werden kann, heißt (7)\_\_\_\_\_. Jeder Block wird durch einen booleschen Ausdruck beschrieben, den sogenannten (8)\_\_\_\_\_. Zur Erzeugung einer (9)\_\_\_\_\_ Minimalform wird die Einsmenge mit Primblöcken überdeckt. Durch die Überdeckung der Nullmenge lässt sich in analoger Weise eine (10)\_\_\_\_\_ Minimalform entwickeln.

Variablenbelegungen, die an den Eingängen einer Schaltung nicht anliegen können oder dürfen, werden als (11)\_\_\_\_\_ -Belegung bezeichnet und im KV-Diagramm mit (12)\_\_\_\_\_ markiert. Die entsprechenden Felder werden in der Blockbildungsphase genau dann überdeckt, wenn hierdurch (13)\_\_\_\_\_ Blöcke entstehen.

In KV-Diagrammen ab (14)\_\_\_\_\_ Variablen stehen benachbarte Variablenbelegungen nicht mehr in jedem Fall unter- oder nebeneinander. Mit Hilfe dreidimensionaler Diagramme können Funktionen mit maximal (15)\_\_\_\_\_ Variablen minimiert werden.

Störimpulse, die dann entstehen, wenn sich genau eine Eingangsvariable ändert, werden als (16)\_\_\_\_\_ bezeichnet. Eine mit Hilfe von KV-Diagrammen minimierte Funktion ist gegen solche Störimpulse abgesichert, wenn es keine (17)\_\_\_\_\_ Primblöcke gibt. Selbst in abgesicherten Schaltungen können Störimpulse entstehen, die als (18)\_\_\_\_\_ bezeichnet werden.

**Aufgabe 6.2****Webcode****5235**

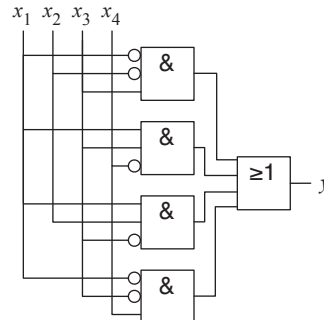
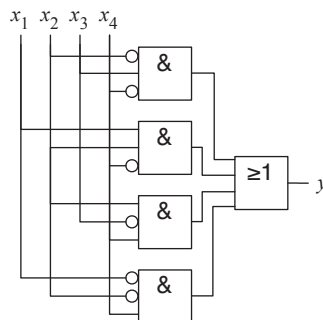
Erzeugen Sie für die beiden unten abgebildeten Funktionen  $y_1$  und  $y_2$  ein KV-Diagramm und berechnen Sie eine disjunktive Minimalform.

	$x_3$	$x_2$	$x_1$	$y_1$
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

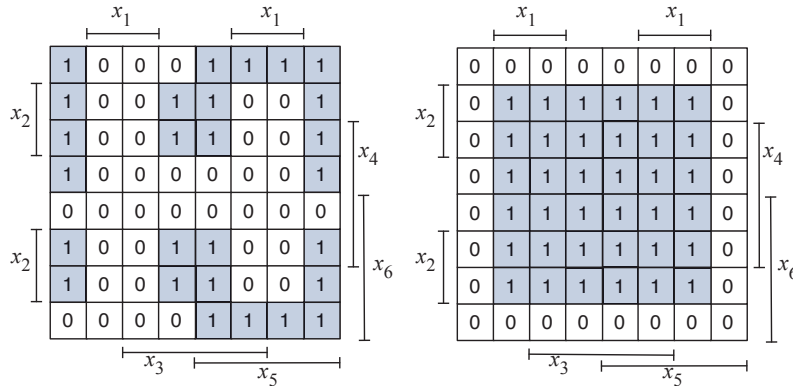
	$x_4$	$x_3$	$x_2$	$x_1$	$y_2$
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

**Aufgabe 6.3****Webcode****5421**

Sind die folgenden beiden Schaltnetze äquivalent? Stellen Sie zur Beantwortung der Frage für beide Schaltungen ein KV-Diagramm auf und tragen Sie die Funktionswerte sowie die durch die UND-Glieder repräsentierten Blöcke ein. Was stellen Sie fest?

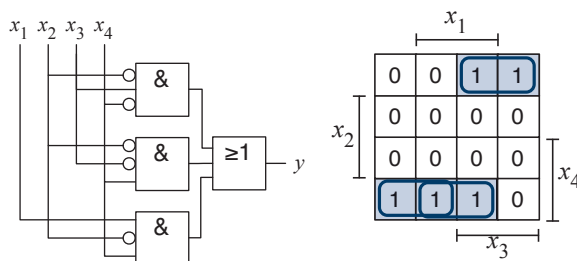


Minimieren Sie die booleschen Funktionen, die durch die folgenden KV-Diagramme gegeben sind:

**Aufgabe 6.4****Webcode****5129**

Beachten Sie, dass in KV-Diagrammen mit fünf oder mehr Variablen benachbarte Variablenbelegungen nicht mehr in jedem Fall nebeneinander angeordnet sind und Blöcke dadurch aus verschiedenen Fragmenten zusammengesetzt sein können.

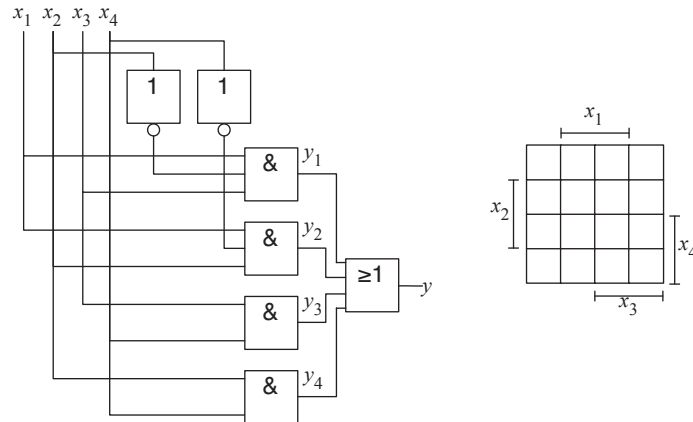
Betrachten Sie das folgende Schaltnetz sowie das zugehörige KV-Diagramm. Ist die Schaltung gegen Logik-Hazards abgesichert? Falls ja, warum? Falls nein, sichern Sie das Schaltnetz gegen Logik-Hazards ab.

**Aufgabe 6.5****Webcode****5954**

**Aufgabe 6.6**

**Webcode**  
**5392**

Bei dem abgebildeten Schaltnetz handelt es sich um dasjenige, das Sie im Übungsteil des letzten Kapitels bereits auf Hazards untersucht haben. Tragen Sie die implementierte Funktion in das abgebildete KV-Diagramm ein und sichern Sie das Schaltnetz gegen Logik-Hazards ab.



Überprüfen Sie anhand Ihrer Lösung, ob Sie bei Ihrer Analyse im letzten Kapitel alle Hazards gefunden haben.

**Aufgabe 6.7**

**Webcode**  
**5112**

In Kapitel 4 haben Sie die Reed-Muller-Normalform kennen gelernt, die eine boolesche Funktion durch die XOR-Verknüpfung mehrerer Basisterme darstellt, die ausschließlich nicht negierte, konjunktiv verknüpfte Variablen enthalten dürfen. Können Sie sich eine Blockbildungsvorschrift vorstellen, mit deren Hilfe sich die Reed-Muller-Normalform aus einem KV-Diagramm extrahieren lässt?

**Aufgabe 6.8**

**Webcode**  
**5343**

Geben Sie alle vierstelligen Funktionen an, für die die disjunktive Minimalform gleich der disjunktiven Normalform und gleichzeitig die konjunktive Minimalform gleich der konjunktiven Normalform ist.

Tipp: Überlegen Sie sich hierzu zunächst, wie das KV-Diagramm dieser Funktionen aussehen müsste.

# Sachwortverzeichnis

## Symbole

2-Bit-Prädiktion, 394  
2-aus-5-Code, 84  
2421-Code, 83  
7-Segment-Anzeige, 80, 135  
74210-Code, 84, 85  
8421-Code, 81

## A

Abakus, 13, 415  
Abgeleiteter Operator, 97  
Abhängige Variable, 94  
Absolute Adressierung, 381  
Absoluter Sprung, 317  
Absorptionsgesetz, 109  
Addierer, 218, 415  
    Carry-look-ahead-, **221**, 263  
    Carry-ripple-, **220**, 263  
    Carry-save-, 229  
    Conditional-Sum-, 224  
    Präfix-, 227  
    serieller, 333  
Additionssystem, 60, 415  
Adressbus, 356, 360, 415  
Adressdecoder, 320  
Adresse, 254  
Adressierung  
    absolute, 381  
    indirekte, 381  
    postinkrementierende, 386  
    relative, 381  
    speicherindirekte, 382  
Adressierungsart, 381, 415  
Adressierungsgranularität, 318  
Adressmodifikator, 347  
Adressmultiplexing, 323  
Adresspin, 323  
Äquivalenz, 100

Äquivalenz-Operation, 99  
Aiken-Code, 81, 83  
Akkumulator, 305, 337, 415  
Akzeptor, 40, 286, 415  
Algorithmus  
    Euklidischer, 375  
Allgemeingültigkeit, 100  
Allzweckregister, 415  
Alphabet  
    Ausgabe-, 286  
    Eingabe-, 286  
Alpha-Teilchen, 328, 334  
Analytische Maschine, 16  
Anstiegszeit, 170  
Antivalenz-Operation, 99  
Antivalenzfunktion, **125**, 157, 215  
Arabisches System, 61  
Architektur  
    ARM-, 387  
    Big-Endian-, 68  
    Harvard-, 20, **353**, 361  
    Little-Endian-, 68  
    Load-Store-, 385  
    PowerPC-, 386  
    Von-Neumann-, 352  
    x86-, 379  
Arithmetisch-logische Einheit, 251, 415  
ARM-Architektur, 387  
ASCC, 19  
Assembler, 353, 415  
Assembler-Sprache, 416  
Assoziativer Cache, 397  
Assoziativgesetz, 110  
Asynchroner Zähler, 313, 330  
Asynchrones RS-Latch, 267  
Atom, 34  
Auffangregister, 300, 416  
Auflösungsgenauigkeit, 75, 77  
Aufzugssteuerung, 343, 350  
Ausdruck

äquivalenter, 100  
allgemeingültiger, 100  
boolescher, 96  
erfüllbarer, 100  
tautologischer, 100

Ausführungsphase, 359  
Ausgabealphabet, 286  
Ausgabefunktion, 286  
Ausgaberegister, 337  
Ausgabeschaltnetz, 290  
Ausgangslastfaktor, 155  
Ausgangssignal, 140  
Automat  
    Akzeptor, 286  
    endlicher, 286  
    Mealy-, 288  
    Moore-, 288  
    Transduktor, 286  
Axiome  
    von Huntington, **90**  
    von Robbins, 91

## B

Back-end of line, 54  
Backus-Naur-Form, 356  
Bändermodell, 36  
Bank, 324  
Bank select, 326  
Barrel-Shifter, **249**, 259, 416  
Basis, 43, 61, 416  
Basistechnologie, 140  
    ECL, 141  
    MOS, 141  
    TTL, 140  
Baumstruktur, 96  
BCD-Code, **81**, 192, 259, 416  
BDD, **125**, 158  
Bedingter Sprung, 343, 363  
Befehlsausführung



- spekulative, 393
  - Befehlsdecoder, 251
  - Befehlsholphase, 359
  - Befehlssatz, 352, 416
  - Benchmark, 403, 416
    - Dhrystone-, 404
    - Eispack-, 405
    - Lapack-, 405
    - Linpack-, 402, 405
    - natürlicher, 404
    - SPEC-, 405
    - synthetischer, 404
    - Whetstone-, 404
  - Benchmark-Kollektion, 403
  - Berechnungsmodell
    - universelles, 352, 363
  - Bevorrechtigte Eingänge, 416
  - Bevorrechtigter Eingang, 281
  - Bidirektionaler Zähler, 308
  - Big-Endian, 416
  - Big-Endian-Architektur, 68
  - Bindungsenergie, 36, 416
  - Bindungslücke, 38
  - Bindungspriorität, 99
  - Binärcode, 416
  - Binärcodiertes Dezimalsystem, 80
  - Binäres Entscheidungsdiagramm, **125**, 158, 416
    - geordnetes, 126
    - reduziertes, 127
  - Binärsystem, 62, 416
  - Binärzähler, 308
  - Bipolartransistor, 416
  - Biquinär-Code, 84, 85
  - Bit, 67, 416
  - Blockbildung, 189
    - inverse, 193
  - Blockmultiplikation, 338, 349, 417
  - Blue tape, 55
  - Bohr'sches Atommodell, 34
  - Boolesche Algebra, 90, 417
    - Mengenalgebra, 91
    - Schaltalgebra, 90, 93
  - Boolesche Differenz, 131
  - Boolesche Funktion, 94, 417
  - Boolesche Konstanten, 417
  - Boolescher Ausdruck, 96, 417
  - Branch prediction table, 394
  - Brown'sche Molekularbewegung, 38
  - Buffer, 170
  - Bug, 20
  - Bus, 417
  - Bus-Knoten, 355
  - Bus-Topologie, 355
  - Byte, 67, 417
- ## C
- Cache, 394
    - assoziativer, 397
    - Block, 395
    - Controller, 394
    - direkt abgebildeter, 395
    - Hit, 394, 396
    - Level-*n*-, 395
    - Miss, 394, 396
    - Speicher, 417
    - vollassoziativer, 410
  - Carry-Bit, 218, 358, 364
  - Carry-look-ahead-Addierer, **221**, 263, 417
  - Carry-ripple-Addierer, **220**, 263, 417
  - Carry-save-Addierer, 229, 417
  - Carry-save-Format, **229**, 238, 417
  - Carry-save-Multiplizierer, 238, 417
  - Cell delay, 171
  - Cell-Prozessor, 30
  - Central processing unit, 27, 352, **356**
  - Charakteristik, 75, 417
  - Charakteristische Funktion, 163, 418
  - Chipausbeute, 53
  - Chip select, 326
  - Church'sche These, 363
  - CISC, 406, 418
  - CISC-Prozessor, 380
  - Clock enable, 281
  - CMOS-Schaltung, 151, 282
  - CMOS-Technik, 418
  - Code, 418
    - 2-aus-5-, 84
    - 2421-, 83
    - 74210-, 84, 85
    - 8421-, 81
    - Aiken-, 81, 83
    - BCD-, **81**, 192, 259
  - Biquinär-, 84, 85
  - einschrittiger, 83
  - Excess-3-, 81
  - fehlererkennender, 84
  - Glixon-, 88
  - Gray-, 81, **83**, 187, 288, 291
  - Hamming-, 329
  - m-aus-n-, 85
  - mehrschrittiger, 83
  - One-Hot-, 84, **85**, 368
  - progressiver, 83
  - reflektierter Biquinär-, 84, 85
  - Stibitz-, 81
  - Walking-, 85
  - Code-Distanz, 85
  - Column address strobe, 323
  - Compiler, 353
  - Conditional-Sum-Addierer, 224, 418
  - Core microarchitecture, 30
  - CPI-Wert, 402
  - CPU, 27, 352, **356**
  - Current window pointer, 407
- ## D
- D-Flipflop, 276
  - D-Latch, 272
  - Dadda-Tree-Multiplizierer, 246, 418
  - Data line, 320
  - Datenbus, 355, 418
  - Datenfluss, 356
    - rückgekoppelter, 337
  - Datenregister, 365
  - Datenspeicher, 361
  - Datenwort, 336
  - Davio-Entwicklung
    - negative, 131
    - positive, 131
  - DCTL, 142
  - DDR-RAM, 326
  - De Morgan'sche Regel, 107, 111
    - erweiterte, 134
  - Decode, 359, 389
  - Decoder, 212, 346
  - Decodierphase, 359
  - Defektdichte, 53
  - Defektelektron, 38

Definitorische Form, 418

Delay

Cell, 171

Net, 171

Demultiplexer, **211**, 257, 418

Dezimalsystem, 61, 418

binär codiertes, 80

Dhrystone-Benchmark, 404

Dicing, 55

Die, 55

Differenzenmaschine, 15

Differenzenmethode, 15

Digitaler Signalprozessor, 74, 251

Digitaltechnik, 140

DIMM, 321

DIN

40900, 157

66000, 97

Diode, 41, 418

Direkte Kommunikation, 354

Direkter Sprung, 363

Disjunktion, 93

Disjunktions-Matrix, 253

Disjunktive Minimalform, 418

Disjunktive Normalform, 119, 212, 419

Displacement, 382, 419

Distanz

Code-, 85

Hamming-, 85, 329

Distributivgesetz, 91

Division, 259

Don't-Care-Belegung, 192, 199, 419

Donator, 39

Doppelnegationsgesetz, 110

Dot diagram, 244

Dotierung, 39, 419

Double-precision-Format, 77

Drain, 47

DRAM, 27, 320, 419

DTL, 142

Duale Gleichung, 107

Dualer Operator, 108

Dualitätsprinzip, 105, 107, 419

Dynamische Sprungvorhersage, 394

Dynamischer Hazard, 419

Dynamischer Speicher, 320

## E

E-Flipflop, 297

ECL-Technik, 141

Eigenleitung, 39

Ein-/Ausgabebaustein, 354

Einerkomplement, 70, 419

Eingabealphabet, 286

Eingaberegister, 337

Eingang

bevorrechtigter, 281

-ssignal, 140

-slastfaktor, 155

Einkristall, 38

Einschrittiger Code, 83

Einsmenge, 120, 419

Einzelkernprozessor, 352

Eispack-Benchmark, 405

Electronic Design Automation, 171

Elektron

Paarbildung, 38

Rekombination, 38

Elektronenloch, 38

Elektronenmangelleiter, 40

Elektronenstrom, 39

Elektronenüberschussleiter, 39

Elementaroperatoren, 419

Eliminationsgesetz, 109

ELSI, 25

EM64T, 419

Emitter, 43

Emitter-Basis-Strecke, 43

Emitter-Kollektor-Strecke, 43

Enable-Eingang, 213

Encoder, 346

Endlicher Automat, 286, 419

Endlosschleife, 368

ENIAC, 20

Entscheidungsdiagramm, 419

Binäres, **125**, 158

funktionales, **130**, 159

Entwicklungssatz

von Shannon, 128

Erfüllbarkeit, 100

Erholzeit, 323

ESI, 387

Espresso, 199

Euklidischer Algorithmus, 375

Excess-3-Code, 81

Execute, 359, 389

Exponent, 74, 420

Extended-precision-Format, 79

Extra-large-scale integration, 25

## F

Fallzeit, 170

Fan-In, 155

Fan-Out, 155

FDD, **130**, 159

FDIV-Bug, 74

Feedback loop, 282

Fehlererkennender Code, 84, 420

Fehlererkennung, 327

Fehlerkorrektur, 327

Fehlerkorrigierender Code, 420

Feldeffekttransistor, 47, 420

Ferritkernspeicher, 24

Festkommaformat, 18, 73, 420

Festwertspeicher, 254

FET, 47

Fetch, 359, 389

Finalzustand, 286

Flächenbedarf, 184

Flanke, 170

Flankensteuerung, 274

negative, 275

positive, 275

Flaschenhals

Von-Neumann-, 356

Fließbandprinzip, 389

Fließkommazahl, 74

Flip-Chip-Verfahren, 56, 425

Flipflop, 274, 420

D-, 276

E-, 297

JK-, 279

Master-, 276

RS-, 277

Slave-, 276

T-, 278

Floating state, 145

Flynn-Taxonomie, 378, 420

Folgeadresse, 347

- Formelsynthese, 161, 420  
 definitorische, 164  
 funktionale, 161  
 relationale, 163
- Freie Variable, 94
- Frequenzteiler, 310
- Front-end of line, 53
- Funktion  
 charakteristische, 163  
 partielle, 190
- Funktionales Entscheidungsdiagramm,  
**130**, 159, 420
- Funktionseinheit, 337
- Funktions-Hazard, 174, 421
- Funktionstabelle, 94
- ## G
- Gate, 47
- Gatter, 156, 421
- Gatternetzliste, 157
- Gauß'sche Summenformel, 104, 223
- Gesetz von Moore, 32
- ggT, 375
- Gibi, 64
- Giga-scale integration, 25
- Gleichheitstest, 69
- Gleichung  
 duale, 107
- Gleitkomma-division, 74
- Gleitkommaformat, 18, 74, 421
- Glixon-Code, 88
- GPU, 74, 408
- Grafikprozessor, 74, 408
- Grammatik, 96
- Gray-Code, 81, **83**, 187, 288, 291
- GSI, 25
- ## H
- Halbaddierer, 218, 421
- Halbleiter, 34, 421  
 dotierter, 39  
 reiner, 37
- Hamming-Code, 329
- Hamming-Distanz, 85, 329, 421
- Hamming-Würfel, 85
- Handshaking, 325
- Handshaking-Protokoll, 349
- Hardware-Schaltung, 140  
 Hazard-freie, 194  
 Stromverbrauch, 174, 184  
 Zeitverhalten, 169
- Hardware-Schleife, 382
- Harvard-Architektur, 20, **353**, 361, 421
- Harvard Mark I, 19
- Hazard, 171, 194, 421  
 -frei, 194  
 dynamischer, 172  
 funktionaler, 174  
 Logik-, 194  
 logischer, 172  
 Pipeline-, 392  
 statischer, 171
- Hexadezimalsystem, 63, 421
- High-Pegel, 142
- High-Pegelbereich, 142
- Hilfsregister, 357
- Hitzesensor, 384
- Hochintegration, 140
- Huffman-Normalform, 290, 421
- Huntington'sche Axiome, **90**, 421
- ## I
- I/O, 354
- IA-32-Architektur, 422
- IA-64-Architektur, 422
- Idempotenzgesetz, 109
- IEEE 754, 422
- IEEE-754, 77
- Implikationsoperator, 99, 118
- Indirekte Adressierung, 381
- Indirekte Kommunikation, 355
- Individualisierung, 253
- Induktion  
 strukturelle, 102  
 vollständige, 102
- Ingot, 51, 422
- Initialzustand, 286
- Inkrementierer, 232, 422
- Input/Output, 354
- Instabiler Zustand, 269
- Instruktionsarchitektur, 379, 422
- CISC-, 380
- RISC-, 385
- Instruktionsdecoder, 356, **368**, 376, 422
- Instruktionsregister, 365
- Instruktionszähler, 316, **358**, 365, 422
- Integrationsdichte, **57**, 58, 422
- Integrierter Schaltkreis, 26
- Intel, 27
- Interleaving, 324
- Interrupt, 328
- Intervallgrenze, 75
- Inverses Element, 91
- Inversionszone, 49
- Ion, 35
- IPC-Wert, 402
- ## J
- JFET, 47
- JK-Flipflop, 279
- ## K
- Kanallänge, **57**, 58, 422
- Karnaugh-Veitch-Diagramm, 186, 422
- Kerbensystem, 60, 422
- kgV, 375
- Kibi, 64
- Koeffizienten-Matrix, 235
- Kofaktor, 422  
 negativer, 128  
 positiver, 128
- Kollektor, 43
- Kombinatorische Schaltung, 157
- Kommunikation  
 direkte, 354  
 indirekte, 355
- Kommutativgesetz, 91
- Komparator, 213, 259, 422
- Komplexgatter, 175
- Komplexitätsanalyse, 167
- Kondensator, 320, 327
- Konjunktion, 93
- Konjunktions-Matrix, 253
- Konjunktive Minimalform, 423
- Konjunktive Normalform, 119, 423
- Konklusion, 99

Konsistenzfunktion, 164  
 Kontrollbus, 356  
 Kontrollfluss, 356  
 Korrektur-Tetrade, 82  
 Kostenfunktion, 184, 423  
 Kreuzprodukt, 408  
 Kristallgitter, 37  
 KV-Diagramm, 186  
     dreidimensionales, 196

## L

Label, 348  
 Ladung, 320, 327  
 Lapack-Benchmark, 405  
 Large-scale integration, 25  
 Lastfaktor, 155  
 Latch, 274, 423  
     D-, 272  
     RS-, asynchrones, 267  
     RS-, synchrones, 272  
 Laufzeit, 184  
 Layer, 54  
 Leckstrom, 151, 321  
 Leistungsbewertung, 400  
 Leitungsband, 36  
 Leitungselektron, 36  
 Level-*n*-Cache, 395  
 Linpack-Benchmark, 402, 405  
 Literal, 119, 423  
 Little-Endian, 423  
 Little-Endian-Architektur, 68  
 Load-Store-Architektur, 385, 423  
 Löcherstrom, 39  
 Lochkarte, 16  
 Lochstreifen, 18  
 Logik  
     -ebene, 156  
     -gatter, 156  
     -polarität, 144  
     -zelle, 156  
     negative, 143  
     positive, 143  
 Logik-Hazard, 172, 194, 423  
 Logikpolarität, 423  
 Low-Pegel, 142  
 Low-Pegelbereich, 142

LSI, 25

## M

m-aus-n-Code, 85  
 Mainframe, 26  
 Majoritätsträger, 40  
 Makro, 368  
 Manchester Mark I, 23  
 Mantisse, 73, 423  
 Maschinenbefehl, 423  
 Master-Flipflop, 276  
 Master-Slave-Flipflop, 276, 423  
 Matrix  
     Disjunktions-, 253  
     Koeffizienten-, 235  
     Konjunktions-, 253  
 Matrixmultiplizierer, 235, 261, 423  
 Maxterm, **119**, 424  
 Mealy-Automat, 288, 424  
 Mebi, 64  
 Medium-scale integration, 25  
 Mehrkernprozessorsystem, 352  
 Mehrphasentaktgeber, 365, 376  
 Mehrschrittiger Code, 83  
 Mengenalgebra, 91  
 MFLOPS, 402  
 Micromosaic, 25  
 Mikroprogramm, 347, 384  
 Mikroprogrammierung, 18, 343, 424  
 Mikroprozessor, 27, 352, **356**, 424  
 Mikrorechner, 352  
 MIMD, 378  
 Minicomputer, 26  
 Minimierung, 424  
     grafische, 186  
     mehrstelliger Funktionen, 196  
     partieller Funktionen, 190, 199  
     tabellarische, 197  
 Minimierungsziel, 182  
 Minoritätsträger, 40  
 Minterm, **119**, 212, 253, 424  
 MIPS, 402  
 Mischzähler, 314  
 MISD, 378  
 Mnemonic, 424  
 Mode

Nibble, 324

Page, 324

Modell

    funktionales, 156

Modellprozessor, 360

Modulo-Operation, 375

Moore's law, 32

Moore'sches Gesetz, 32, 424

Moore-Automat, 288, 424

MOS-Schaltung, 145

MOS-Technik, 141, 424

MOSFET, 424

    n-Kanal-, 145

    p-Kanal-, 145

MSI, 25

Multiple Instruction

    Multiple Data, 378

    Single Data, 378

Multiplexer, **207**, 256, 337, 424

Multiplexing, 28

Multiplikation, 259, 367

Multiplizierer, 234, 424

    Carry-save-, 238

    Dadda-Tree-, 246

    Matrix-, 235, 261

    Wallace-Tree-, 241

Multiprozessorsystem, 352, 396, 399

MWIPS, 404

## N

n-Kanal, 49

n-Kanal-JFET, 47

n-Leiter, 40

Nachkommanormalisierung, 76

NaN, 79

NAND-Funktion, 99

Napierstäbchen, 14

Negation, 93, 252

Negationskreis, 157

Negationstheorem, 106

Negative Logik, 143, 424

Negative-Bit, 364

Negative-Flag, 358

Net delay, 171

Netzliste, 157

Neunerkomplement, 82, 85

Neutrales Element, 91  
 Nibble mode, 324  
 NMOS-Schaltung, 148  
 NMOS-Technik, 425  
 NOR-Funktion, 99  
 NOR-Gatter, 268  
 Normalform, 102, 425  
   disjunktive, 119, 212  
   konjunktive, 119  
   Reed-Muller-, 122  
 Normalisierung, 425  
   Nachkomma-, 76  
   Vorkomma-, 76  
 Normalisierungsregel, 76  
 Not a number, 79  
 NP-hart, 199  
 Nullmenge, 120, 425

**O**

ODER-Matrix, 376  
 ODER-Verknüpfung, 93  
 OFDD, **130**, 159  
 Offsetzähler, 308  
 O-Kalkül, 167, 425  
 Oktalsystem, 62, 425  
 One-Hot-Code, 84, **85**, 368, 425  
 Opcode, 19  
 Opcode-Feld, 387  
 Operationswerk, 338, 425  
 Operationswerksynthese, 338  
 Operator  
   abgeleiteter, 97  
   dualer, 108  
 Operatorensystem  
   vollständiges, 97, 117

**P**

p-Kanal-JFET, 47  
 p-Leiter, 40  
 p-Wanne, 53  
 Packaging, 55, 425  
 Page, 322  
 Page miss, 398  
 Page mode, 324  
 Parallele Präfix-Funktion, 216

Parallelmultiplizierer, 425  
 Paritätsbit, 125, 328  
 Paritätscode, 125  
 Paritätsfunktion, **125**, 157, 215  
 Partialprodukt, 234, 239  
 Partielle Funktion, 425  
 Patriot-Abwehrsystem, 66  
 Peirce-Funktion, 99  
 Performance rating, 401  
 Pipeline, 18, **389**, 426  
   -Hazard, 392  
   flush, 393  
   Superpipelining, **391**, 409  
 Pipeline-Hazard, 425  
 Pixel-Shader, 408  
 PLA, 253  
 Planartechnik, 50, 52, 426  
 PLD, 253  
 PMOS-Schaltung, 145  
 PMOS-Technik, 426  
 pn-Übergang, 41, 426  
 Polaritätsindikator, 144  
 Positive Logik, 143, 426  
 Postinkrement, 381  
 Postinkrement-Adressierung, 386  
 PowerPC-Architektur, 386  
 Prädikatenlogik, 165  
 Prädiktion  
   2-Bit-, 394  
 Präfix-Addierer, 227, 426  
 Präfix-Funktion  
   parallele, 216  
 Präfix-Logik, 215, 426  
 Präfix-Schreibweise, 63  
 Prämisse, 99  
 Primblock, 190  
 Primimplikant, 190  
 Primimplikantentafel, 198  
 Program counter, 358  
 Programmable logic array, 253  
 Programmable logic device, 253  
 Programmierbare Logik, 253, 258, 426  
 Programmspeicher, 361  
 Programmsteuerung, 352  
 Programmzähler, 316  
 Progressiver Code, 83  
 Protected mode, 29  
 Prüfbit, 329

Pseudo-Tetrade, 81  
 Puffer, 170  
 Punktdiagramm, 243

## Q

Quantor, 426  
   boolescher, 165  
 Quine-McCluskey-Verfahren, 197, 426  
 Quine'sche Tabelle, 197

## R

Radioaktive Strahlung, 328  
 Radix-4-SRT-Division, 74  
 RAM, 318, 426  
 Random access memory, 318  
 Range gate, 66  
 Rank, 326  
 Rationale Zahl, 63  
 Raumladungszone, 41  
 Read-only memory, 254  
 Read-Signal, 319  
 Rechenregel, 102  
   abgeleitete, 109  
 Rechenwerk, 356, 426  
 Rechner, 352  
 Rechnerklassifikation  
   nach Flynn, 378  
   nach Instruktionsarchitektur, 379  
 Rechteckschwingung, 271  
 Reduktionszelle  
   Carry-save-, 230, 239  
 Reed-Muller-Normalform, 122, 426  
 Reflektierter Biquinär-Code, 84, 85  
 Refresh-Logik, 327  
   erweiterte, 329  
 Regel  
   von De Morgan, 107, 111  
 Register, 300, 337, 427  
   Akkumulator-, 305  
   Auffang-, 300  
   Ausgabe-, 337  
   Daten-, 365  
   Eingabe-, 337  
   Hilfs-, 357  
   Instruktions-, 365

- Schiebe-, 302
  - Stapel-, 358
  - Status-, 358, 363
  - Umlauf-, 303
  - Universal-, **304**, 357
  - Register-Transfer-Ebene, 336, 427
  - Register-Transfer-Entwurf, 427
  - Registerbreite, 300, 360
  - Registerfenster, 407
  - Registersatz, 356
  - Rekonfigurierbarer Speicher, 384
  - Rekonvergenz, 161, 172, 427
  - Relais, 17
  - Relative Adressierung, 381
  - Relativer Sprung, 317, 363
  - Reset, 281, 331, 350, 427
  - Resistor, 43
  - Resistor-Transistor-Logik, 142
  - Ringzähler, 303
  - RISC, 406, 427
  - RISC-Prozessor, 385
  - Robbins-Algebra, 91
  - ROBDD, **125**, 158
  - Röhre, 22
  - ROM, 254, 427
  - Römische Zahlen, 60, 427
  - Row address strobe, 323
  - RS-Flipflop, 277
  - RS-Latch
    - asynchrones, 267
    - synchrones, 272
  - RTL, 142
  - Rückkopplungsschleife, 282
  - Rücksprungadresse, 358
  - Rückwärtszähler, 308
  - Rundungsfehler, 67
- S**
- Schalenmodell, 34
  - Schaltalgebra, 90, 93, 427
  - Schaltfunktion, 94
  - Schaltkreis, 427
    - integrierter, 26
  - Schaltkreisfamilie, 140, 427
  - Schaltnetz, 157, 427
    - Ausgabe-, 290
    - Hazard-freies, 194
    - Übergangs-, 290
    - zweistufiges, 158
  - Schaltung
    - kombinatorische, 157
    - sequenzielle, 266
  - Schaltungssynthese, 156, 428
    - BDD-basierte, 158
    - FDD-basierte, 159
    - zweistufige, 157
  - Schaltwerk, 266, 428
  - Schaltwerksynthese, 285, **289**
  - Schickard'sche Rechenuhr, 14
  - Schiebeoperation, 234
  - Schieberegister, 302, 428
    - rotierendes, 250
  - Schrankensteuerung, 350
  - Schrittlänge, 308
  - Schrittzählung, 317
  - Schwebezustand, 145
  - Scrubbing, 329
  - SDR-RAM, 326
  - SDRAM, 325
  - Seite, 322
  - Sequencer, 365, 428
  - Sequenzielle Schaltung, 266
  - Sequenzielles Element, 266
  - Shader, 408
  - Shannon'scher Entwicklungssatz, 128, 428
  - Sheffer-Funktion, 99
  - Sheffer-Stroke, 117
  - Signal
    - Ausgangs-, 140
    - Eingangs-, 140
  - Signalausbreitung, 169
  - Signalflanke, 170
  - Signalpfad, 336
  - Signalverzögerung, 169
  - SIMD, 378, 408
  - SIMM, 321
  - Single cristal, 38
  - Single Instruction
    - Multiple Data, 378
    - Single Data, 378
  - Single-precision-Format, 77
  - SISD, 378
  - Slave-Flipflop, 276
  - SLSI, 25
  - Small-scale integration, 25
  - Soft error, 328
  - Soroban, 13
  - Source, 47
  - SPARC-Prozessor, 407
  - SPEC-Benchmark, 405
  - SpeedStep-Technologie, 384
  - Speicher, **318**, 428
    - Cache-, 394
    - Daten-, 361
    - DDR-, 326
    - dynamischer, 320
    - Programm-, 361
    - rekonfigurierbarer, 384
    - SD-, 325
    - SDR-, 326
    - statischer, 318
    - virtueller, 398
  - Speicherbank, 324
  - Speicherchip, 321
  - Speicherelement, **266**, 428
    - asynchrones, 267
    - D-Flipflop, 276
    - D-Latch, 272
    - einflankengesteuertes, 275
    - flankengesteuertes, 274
    - JK-Flipflop, 279
    - RS-Flipflop, 277
    - RS-Latch
      - asynchrones, 267
      - synchrones, 272
    - T-Flipflop, 278
    - taktzustandsgesteuertes, 271
    - zweiflankengesteuertes, 275
  - Speicherhierarchie, 395, 428
  - Speicherindirekte Adressierung, 382
  - Speichermatrix, 322
  - Speicherordnung, 68, 428
  - Speicherphase, 359
  - Speicherschleife, 282
  - Speicherseite, 322
  - Speicherzeit, 169
  - Spekulative Befehlsausführung, 393
  - Spin-Coating-Verfahren, 52
  - Sprung
    - absoluter, 317
    - bedingter, 343, 363

direkter, 363  
 relativer, 317, 363  
 unbedingter, 363  
 Sprungbedingung, 347  
 Sprungbefehl, 358, 367  
 Sprungvorhersage, 394, 428  
   dynamische, 394  
 SRAM, 318, 428  
 SRT-Division, 74  
 SSI, 25  
 Stabiler Zustand, 269  
 Stack, 358  
 Stapel, 358  
 Stapelregister, 428  
 Startzustand, 286  
 Statischer Hazard, 428  
 Statischer Speicher, 318  
 Statusbit, 429  
 Statusregister, 358, 363, 429  
 Statusvariable, 338  
 Stellensystem, 61  
 Stellenwertsystem, 61, 429  
 Stelligkeit, 94  
 Stellvariable, 338  
 Steuerbus, 429  
 Steuerflussabhängigkeit, 393, 409, 429  
 Steuerlogik, 337  
 Steuersignal, 354  
 Steuervariable, 338, 347  
 Steuerwerk, 338, 356, 429  
   -synthese, 340  
   adressmodifizierendes, 346  
   fest verdrahtetes, 343, 349  
   mikroprogrammiertes, 343, 349  
 Stibitz-Code, 81  
 Strahlung  
   radioaktive, 328  
 Strichsystem, 60  
 Stromverbrauch, 174, 184  
 Strukturbreite, 57, 429  
 Strukturelle Induktion, 102, 429  
 Stschoty, 13  
 Störimpuls, 171, 194  
 Suan pan, 13  
 Subtrahierer, 233, 429  
 Subtraktionsregel, 60  
 Suffix-Notation, 63  
 Summenbit, 219

Super-large-scale integration, 25  
 Supercomputer, 405  
 Superpipelining, **391**, 409, 429  
 Superskalartechnik, **392**, 409, 429  
 Swizzle-Maske, 408  
 Synchroner Zähler, 309, 331  
 Synchrones RS-Latch, 272  
 Synergistic Processing Unit, 30  
 Synthese  
   Operationswerk-, 338  
   Schaltwerk-, 285  
   Steuerwerk-, 340  
 System/360, 25

## T

T-Flipflop, 278  
 T-Glied, 154  
 Takt, 372, 429  
   differenzieller, 326  
 Taktflankensteuerung, 274  
 Taktfrequenz, 271, 400  
 Taktgeber  
   Mehrphasen-, 376  
 Taktsignal, 271  
 Taktzustandssteuerung, 271  
 Tautologie, 100, 429  
 Taxonomie  
   nach Flynn, 378  
   nach Instruktionsarchitektur, 379  
 Teile-und-herrsche, 224  
 Terminal, 27  
 Tetrade, 81  
 Tetraden-Code, 80  
 Timing closure, 171  
 Top-500-Liste, 405  
 Transduktor, 286, 430  
 Transistor, 23, 42, 156, 320, 430  
 Transistorebene, 156  
 Transition, 269  
 Transmissionsglied, 154  
 Triode, 22  
 Trägermenge, 91  
 TTL-Technik, 140  
 Turing-Maschine, 363

## U

Überdeckung  
   minimale, 198  
 Übergangsschaltnetz, 290  
 Übergangstabelle, 273, 430  
 Überhitzungsschutz, 384  
 Übertragsadditionsregel, 70  
 Übertragsbit, 218  
   Rekursionsschema, 221  
 Übertragslogik, 314  
 ULSI, 25  
 Ultra-large-scale integration, 25  
 Umladestrom, 151, 174  
 Umlaufregister, 303, 330  
 Unbedingter Sprung, 363  
 UND-Verknüpfung, 93  
 Universalregister, **304**, 357, 430  
 Universelles Berechnungsmodell, 352, 363  
 Unterprogrammaufruf, 358, 407

## V

Vakuumröhre, 22  
 Valenzband, 37  
 Valenzelektron, 36  
 Variable  
   abhängige, 94  
   freie, 94  
   Status-, 338  
   Stell-, 338  
   Steuer-, 338, 347  
 Variablenbelegung  
   benachbarte, 186  
   inkonsistente, 164  
   konsistente, 164  
 Vektorrechner, 378  
 Venn-Diagramm, 92  
 Verdrängungsstrategie, 398  
 Verlustleistung, 151  
 Vertex-Shader, 408  
 Very-large-scale integration, 25  
 Verzögerung  
   RAS/CAS, 323  
 Verzögerungsglied, 271  
 Verzögerungslogik, 393

Verzögerungszeit, 169  
 Virtueller Speicher, 398, 430  
 VLIW-Prozessor, 392  
 VLSI, 25  
 Volladdierer, 218, 430  
 Vollasoziativer Cache, 410  
 Vollständige Induktion, 102, 430  
 Vollständiges Operatorensystem, 97, 117, 430  
 Von-Neumann-Architektur, 23, 352, 430  
 Von-Neumann-Flaschenhals, 356, 430  
 Von-Neumann-Rechner, 430  
 Vorkommanormalisierung, 76, 78  
 Vorwärtszähler, 308  
 Vorzeichenbitdarstellung, 69, 234, 431

## W

Wafer, 51, 53, 431  
 Wahrheitstabelle, 94, 431  
 Wahrheitstafel, 94  
 Walking-Code, 85  
 Wallace-Tree-Multiplizierer, 241, 431  
 Wärmeentwicklung, 151  
 Wartezyklus, 393, 409, 431  
 Whetstone-Benchmark, 404  
 Widerstand  
   spezifischer, 37  
 Wire-Bond-Verfahren, 56  
 Wiring layer, 54  
 Word line, 320  
 Write, 389  
 Write back, 359  
 Write enable, 371

Write-In-Strategie, 399  
 Write-Through-Strategie, 398

## X

x86-Architektur, 379, 431  
 XLSI, 25  
 XOR-Verknüpfung, 99

## Z

Z3, 17  
 Zahlencode, 80  
 Zahlendarstellung  
   explizite, 77  
   gepackte, 77  
   implizite, 77  
   ungepackte, 77  
 Zahlenformat, 431  
   äquidistantes, 74  
   Auflösungsgenauigkeit, 75  
   Festkommadarstellung, 73  
   Gleitkommadarstellung, 74  
   rechnerinternes, 67  
 Zahlensystem, 60, 431  
   *b*-adisches, 62  
   binäres, 62  
   eindeutiges, 69  
   hexadezimales, 63  
   oktales, 62  
   redundantes, 69  
   symmetrisches, 69  
   unäres, 60  
 Zähler, 308, 431

asynchroner, 313, 330  
 bidirektionaler, 308  
 Binär-, 308  
 gemischter, 314  
 Instruktions-, 316, **358**, 365  
 Offset-, 308  
 synchroner, 309, 331  
 Zeitdiagramm, 268, 431  
 Zeitverhalten, 169  
 Zellbibliotheken, 185  
 Zelle, 156  
 Zentraleinheit, 352  
 Zero-Bit, 358, 364  
 Ziffer, 60  
 Zugriffszeit  
   CAS-, 323  
   RAS-, 323  
 Zustand  
   -smenge, 286  
   -ssteuerung  
   negative, 274  
   positive, 274  
   -svariable, 267  
   -svektor, 285  
   -sübergang, 269  
   -sübergangsfunktion, 286  
   -sübergangsgraph, 269  
   instabiler, 269, 295  
   stabiler, 269, 295  
 Zustandsvariable, 431  
 Zweierkomplement, **71**, 233, 252, 431  
 Zweistufiges Schaltnetz, 431  
 Zwischenglied, 140  
 Zykluszeit, 323  
 Zählrichtung, 308