H. Liebig · T. Flik · P. Rechenberg
A. Reinefeld · H. Mössenböck

Das Ingenieurwissen
Technische Informatik

Ingenieurwissen

Springer Vieweg

Das Ingenieurwissen: Technische Informatik

Hans Liebig • Thomas Flik • Peter Rechenberg
Alexander Reinefeld • Hanspeter Mössenböck

Das Ingenieurwissen: Technische Informatik

Springer Vieweg

Hans Liebig
TU Berlin
Berlin, Deutschland

Thomas Flik
TU Berlin
Berlin, Deutschland

Peter Rechenberg
Universität Linz
Linz, Österreich

Alexander Reinefeld
Zuse-Institut
Berlin, Deutschland

Hanspeter Mössenböck
Universität Linz
Linz, Österreich

ISBN 978-3-662-44390-3
DOI 10.1007/978-3-662-44391-0

ISBN 978-3-662-44391-0 (eBook)

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Springer Vieweg
Das vorliegende Buch ist Teil des ursprünglich erschienenen Werks „HÜTTE – Das Ingenieurwissen", 34. Auflage, Heidelberg, 2012.

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Vieweg ist eine Marke von Springer DE. Springer DE ist Teil der Fachverlagsgruppe Springer Science+Business Media.
www.springer-vieweg.de

Vorwort

Die HÜTTE Das Ingenieurwissen ist ein Kompendium und Nachschlagewerk für unterschiedliche Aufgabenstellungen und Verwendungen. Sie enthält in einem Band mit 17 Kapiteln alle Grundlagen des Ingenieurwissens:

- Mathematisch-naturwissenschaftliche Grundlagen
- Technologische Grundlagen
- Grundlagen für Produkte und Dienstleistungen
- Ökonomisch-rechtliche Grundlagen

Je nach ihrer Spezialisierung benötigen Ingenieure im Studium und für ihre beruflichen Aufgaben nicht alle Fachgebiete zur gleichen Zeit und in gleicher Tiefe. Beispielsweise werden Studierende der Eingangssemester, Wirtschaftsingenieure oder Mechatroniker in einer jeweils eigenen Auswahl von Kapiteln nachschlagen. Die elektronische Version der Hütte lässt das Herunterladen einzelner Kapitel bereits seit einiger Zeit zu und es wird davon in beträchtlichem Umfang Gebrauch gemacht.

Als Herausgeber begrüßen wir die Initiative des Verlages, nunmehr Einzelkapitel in Buchform anzubieten und so auf den Bedarf einzugehen. Das klassische Angebot der Gesamt-Hütte wird davon nicht betroffen sein und weiterhin bestehen bleiben. Wir wünschen uns, dass die Einzelbände als individuell wählbare Bestandteile des Ingenieurwissens ein eigenständiges, nützliches Angebot werden.

Unser herzlicher Dank gilt allen Kolleginnen und Kollegen für ihre Beiträge und den Mitarbeiterinnen und Mitarbeitern des Springer-Verlages für die sachkundige redaktionelle Betreuung sowie dem Verlag für die vorzügliche Ausstattung der Bände.

Berlin, August 2013

H. Czichos, M. Hennecke

Das vorliegende Buch ist dem Standardwerk *HÜTTE Das Ingenieurwissen 34. Auflage* entnommen. Es will einen erweiterten Leserkreis von Ingenieuren und Naturwissenschaftlern ansprechen, der nur einen Teil des gesamten Werkes für seine tägliche Arbeit braucht. Das Gesamtwerk ist im sog. Wissenskreis dargestellt.

Technische Informatik

H. Liebig, Th. Flik, P. Rechenberg, A. Reinefeld, H. Mössenböck

Mathematische Modelle

H. Liebig, P. Rechenberg

Digitale Systeme

H. Liebig

Programmierung
P. Rechenberg, H. Mössenböck

Technische Informatik

H. Liebig
Th. Flik
P. Rechenberg
A. Reinefeld
H. Mössenböck

In der Informatik verbindet sich das axiomatische, logisch-strukturtheoretische Denken der Mathematik mit dem konstruktiven und ökonomischen, d. h. praktisch-ingenieurmäßigen Handeln der Technik. Die Informatik ist daher sowohl eine Strukturwissenschaft, die abstrakt (und immateriell) betrieben wird, als auch eine Ingenieurwissenschaft, die sich konkret (und materiell) mit der Entwicklung, dem Bau und dem Betrieb technischer Produkte befasst.

Von anderen Gebieten der Wissenschaften und der Technik unterscheidet sich die Informatik durch die Art dieser Produkte. Sie sind zum einen *Hardware*, die aufgrund ihrer materiellen Beschaffenheit nur schwer verändert werden kann, und zum andern *Software*, die – mechanistisch betrachtet – leicht zu verändern ist. Trotz ihrer gegensätzlichen Erscheinung sind Hardware und Software jedoch im Prinzip bis auf ein unvermeidliches Minimum an Rechenmaschine gegeneinander austauschbar.

In der Praxis existieren Hardware und Software nicht getrennt für sich, sondern bilden eine Einheit, wobei die Grenzziehung zwischen beiden von konstruktiven und ökonomischen Bedingungen abhängt. Diese Hardware-Software-Systeme erstrecken sich von kleinsten bis hin zu größten Anwendungen, die in ihrer Komplexität an die Grenzen der physikalischen Realisierbarkeit wie der intellektuellen Beherrschbarkeit stoßen.

Wie in anderen Wissenschaften, so versucht man auch in der Informatik, das millionenfache Zusammenwirken ihrer „Atome“ (bei der Hardware die elektronischen Schalter, bei der Software die Befehle der Programme) durch modulare und hierarchische Gliederung, d. h. durch wiederholte Abstraktion zu beherrschen.

Dieser mit *Technische Informatik* überschriebene Teil J orientiert sich am *Computer* als *dem* technischen Repräsentanten der Informatik. Er beginnt mit einer kurzen Einführung in wichtige theoretische

Tabelle 0–18. Historische Entwicklung von Hardware und Software

Generation	Entwicklung der Hardware	Entwicklung der Software
1	*Bis Ende der fünfziger Jahre* Elektronenröhren als Schaltelemente; zentrale Speicher von wenigen hundert Maschinenwörtern.	*Bis Ende der fünfziger Jahre* Assemblercode; einfachste Betriebssysteme; lochkartenorientierter Einzelbetrieb.
2	*Seit Anfang der sechziger Jahre* Transistorschaltkreise; Ferritkern-, Band-, Trommel-, Plattenspeicher.	*Seit Anfang der sechziger Jahre* FORTRAN, ALGOL 60, COBOL; umfangreichere Betriebssysteme; lochkartenorientierter Stapelbetrieb.
3	*Seit Mitte der sechziger Jahre* Teilweise integrierte Schaltkreise.	*Seit Mitte der sechziger Jahre* PL/I, Pascal, APL, C; Hochkomplexe Betriebssysteme; dialogorientierter Mehrbenutzerbetrieb.

H. Liebig et al., *Das Ingenieurwissen: Technische Informatik,*
DOI 10.1007/978-3-662-44391-0_1, © Springer-Verlag Berlin Heidelberg 2014

Tabelle 0–18. (Fortsetzung)

Generation	Entwicklung der Hardware	Entwicklung der Software
4	*Seit Anfang der siebziger Jahre* Überwiegend hochintegrierte Schaltkreise; Halbleiterspeicher; 8-Bit-Prozessoren auf einem Chip, 1 000 bis 10 000 Transistorfunktionen, Taktfrequenzen um 1 MHz.	*Seit Anfang der achtziger Jahre* Programmiersprachen: Modula-2, Ada, Lisp, Prolog; Programmierumgebungen mit grafischer Benutzeroberfläche; hochentwickelte Textverarbeitungs- und Grafikprogramme.
5	*Seit Anfang der achtziger Jahre* Hochintegrierte Schaltkreise; 16- und 32-Bit-Prozessoren auf einem Chip, bis zu 200 000 Transistorfunktionen, Taktfrequenzen um 8 MHz.	*Seit Anfang der neunziger Jahre* Vordringen der objektorientierten Programmierung; Programmiersprachen: C++ und Java; Vereinigung der Verarbeitung von Text-, Audio- und Videodaten (Multimedia); Ausnutzung des Internets und seiner Kommunikationsmöglichkeiten (z. B. elektronische Post).
	Seit den neunziger Jahren 32- und 64-Bit-Prozessoren mit integrierten Caches und interner Parallelverarbeitung, bis zu 500 Millionen Transistorfunktionen, Taktfrequenzen von mehreren GHz.	
	Im neuen Jahrhundert Stromsparende Prozessoren; viele Prozessorkerne auf einem Chip, Kerne mit Multithreading, Grafikbeschleuniger-Chips mit vielen hundert Kernen, viele Milliarden Transistorfunktionen, keine weitere Steigerung der Taktfrequenz.	*Im neuen Jahrhundert* C#; Integrierte Softwareentwicklungsumgebungen; Programmkommunikation über das Internet.

Konzepte, den *Mathematischen Modellen*, fährt fort mit der Beschreibung des Zusammenwirkens der elektronischen Schaltungen, den *Digitalen Systemen*, behandelt dann Struktur- und Betriebsaspekte in Rechnern, d. h. die *Rechnerorganisation*, und schließt mit der Nutzbarmachung von Rechenanlagen für die verschiedensten Anwendungen, d. h. mit ihrer *Programmierung*. Dabei bleiben entsprechend dem Grundlagencharakter des Werkes besondere, nur für bestimmte Anwendungsgebiete relevante Computersysteme und -programme ausgespart.

Zur Kennzeichnung der historischen Entwicklung von Hardware und Software hat sich eine Einteilung in Generationen eingebürgert, die in der voranstehenden Tabelle skizziert ist.

Mathematische Modelle
H. Liebig, P. Rechenberg

1 Boole'sche Algebra

Die Boole'sche Algebra wurde 1854 von G. Boole zur Formalisierung der Aussagenlogik formuliert und 1938 von C. Shannon auf die Beschreibung von Funktion und Struktur sog. kombinatorischer Relais-Schaltungen angewendet. Seither wird sie zum Entwurf digitaler Rechensysteme eingesetzt. – Die Entsprechung zwischen falschen und wahren Aussagen in der Logik (Aussage x = falsch/wahr) und offenen und geschlossenen Schaltern in der Technik (Schalter x = offen/geschlossen) bildet die Grundlage des Rechnerbaus. Mit der Boole'schen Algebra können nämlich sowohl Aussagenverknüpfungen als auch Schalterverknüpfungen beschrieben werden (Boole'sche Variable $x = 0/1$).

1.1 Logische Verknüpfungen und Rechenregeln

1.1.1 Grundverknüpfungen

Die wichtigsten Grundoperationen sind in Tabelle 1–1 dargestellt. Zu ihnen zählen die Negation (NICHT, NOT; Operationszeichen: Überstreichungen oder vorangestelltes ¬), die Konjunktion (UND, AND; Operationszeichen: · oder ∧) und die Disjunktion (ODER, OR; Operationszeichen + oder ∨). Diese sog. Boole'schen Grundverknüpfungen stehen einerseits für Verbindungen von „Ja-/Nein-Aussagen" (z. B. umgangssprachlichen Sätzen, die nur wahr oder falsch sein können), können aber andererseits auch als Verknüpfungen binärer Systemzustände (z. B. von elektrischen Signalen) angesehen werden (siehe 3.1).

Negation $y = \bar{x}$: $y = 1$, wenn *nicht* $x = 1$ (d. h., $y = 1$, wenn $x = 0$).

Konjunktion $y = x_1 \cdot x_2$: $y = 1$, wenn $x_1 = 1$ *und* $x_2 = 1$.

Disjunktion $y = x_1 + x_2$: $y = 1$, wenn $x_1 = 1$ *oder* $x_2 = 1$.

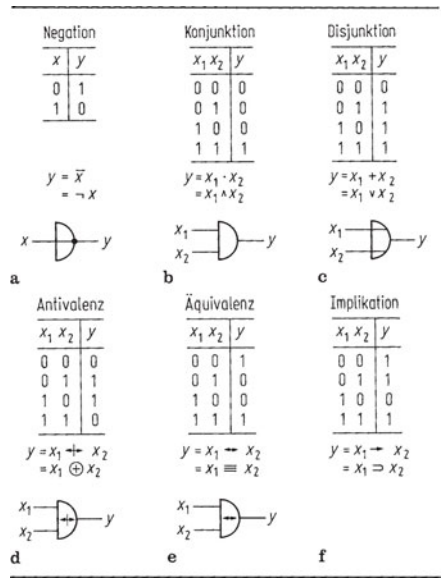
Weitere Grundverknüpfungen sind die Antivalenz (ENTWEDER ODER, XOR; Operationszeichen: ↮ oder ⊕), die Äquivalenz (ÄQUIVALENT; Operationszeichen: ↔ oder ≡) sowie die Implikation (IMPLIZIERT, Operationszeichen: → oder ⊃).

Antivalenz $y = x_1 \not\leftrightarrow x_2$: $y = 1$, wenn *entweder* $x_1 = 1$ *oder* $x_2 = 1$ (d. h., x_1 ist ungleich x_2).

Äquivalenz $y = x_1 \leftrightarrow x_2$: $y = 1$, wenn x_1 *äquivalent* x_2 (d. h., x_1 ist gleich x_2).

Implikation $y = x_1 \rightarrow x_2$: $y = 1$, wenn x_1 *impliziert* x_2 (d. h., x_2 bezieht x_1 ein bzw. x_2 ist größer/gleich x_1).

Tabelle 1–1. Logische Operationen; Wahrheitstabellen, Formeln, Symbole



1.1.2 Ausdrücke

Logische Konstanten, Aussagenvariablen, Grundverknüpfungen und aus ihnen zusammengesetzte komplexere Verknüpfungen werden zusammenfassend als Ausdrücke bezeichnet. In Analogie zu arithmetischen Ausdrücken ist festgelegt, dass · Vorrang vor + hat.

Ferner ist es weithin üblich, den Bereich einer Negation durch Überstreichung anzugeben und, wenn es nicht zu Verwechslungen kommen kann, Malpunkte wegzulassen. Klammern dürfen jedoch nur dann weggelassen werden, wenn für die eingeklammerte Verknüpfung das Assoziativgesetz gilt (das ist für $\cdot$, $+$, $\leftrightarrow$ und $\not\leftrightarrow$ der Fall, nicht jedoch für $\rightarrow$). – Werden Ausdrücke mit den Symbolen der Tabelle 1–1 dargestellt, so wird ihre „Klammerstruktur" durch die Symbolstufung besonders anschaulich (vgl. z. B. die Gleichung und das Blockbild für y_1 in Bild 1–1).
Die Vielfalt der Darstellungsmöglichkeiten erlaubt es, einzelne Operationen durch andere zu beschreiben. Eine gewisse Standardisierung ergibt sich, wenn nur die Operationen $^-$, $\cdot$ und $+$ benutzt werden; $\not\leftrightarrow$, $\leftrightarrow$ und $\rightarrow$ lassen sich damit folgendermaßen ausdrücken (vgl. die Tabellen 1–1d bis f für die drei Grundverknüpfungen Antivalenz, Äquivalenz und Implikation)

$$x_1 \not\leftrightarrow x_2 = \bar{x}_1 \cdot x_2 + x_1 \cdot \bar{x}_2 \,,$$
$$x_1 \leftrightarrow x_2 = \bar{x}_1 \cdot \bar{x}_2 + x_1 \cdot x_2 \,,$$
$$x_1 \rightarrow x_2 = \bar{x}_1 + x_2 \,.$$

Auswertung. Zur Auswertung von Ausdrücken legt man Tabellen an: Links werden im Spaltenkopf die im Ausdruck vorkommenden Variablen und zeilenweise sämtliche Kombinationen von 0 und 1 eingetragen. Rechts werden für alle diese Kombinationen die Werte der Teilausdrücke so lange ausgewertet und niedergeschrieben, bis der Wert des Ausdrucks feststeht. Tabelle 1–2 gibt ein Beispiel.

Tabelle 1–2. Auswertung der Ausdrücke $a \cdot b + c$ und $(a + c) \cdot (b + c)$; beide haben für alle 0/1-Kombinationen von a, b und c die gleichen Werte, d. h., es gilt $a \cdot b + c = (a + c) \cdot (b + c)$

			$a \cdot b$	$a \cdot b + c$	$a + c$	$b + c$	$(a + c) \cdot (b + c)$
a	b	c	↓	↓	↓	↓	↓
0	0	0	0	0	0	0	0
0	0	1	0	1	1	1	1
0	1	0	0	0	0	1	0
0	1	1	0	1	1	1	1
1	0	0	0	0	1	0	0
1	0	1	0	1	1	1	1
1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1

1.1.3 Axiome

Die folgenden Axiome (1-1a) bis (1-5b) definieren mit den Variablen *a, b, c*, den Konstanten 0, 1 und den Operationen $-$, $\cdot$, $+$ die *Boole'sche Algebra*, die sich durch abweichende Rechenregeln und Operationen sowie durch das Fehlen von Umkehroperationen von der gewöhnlichen Algebra unterscheidet.

$$a \cdot b = b \cdot a \,, \tag{1-1a}$$
$$a + b = b + a \,, \tag{1-1b}$$
$$(a \cdot b) \cdot c = a \cdot (b \cdot c) \,, \tag{1-2a}$$
$$(a + b) + c = a + (b + c) \,, \tag{1-2b}$$
$$(a + b) \cdot c = a \cdot c + b \cdot c \,, \tag{1-3a}$$
$$a \cdot b + c = (a + c) \cdot (a + c) \,, \tag{1-3b}$$
$$a \cdot 1 = a \,, \tag{1-4a}$$
$$a + 0 = a \,, \tag{1-4b}$$
$$a \cdot \bar{a} = 0 \,, \tag{1-5a}$$
$$a + \bar{a} = 1 \,. \tag{1-5b}$$

Axiome (1-1a) und (1-1b) erlauben das Vertauschen von Operanden (Kommutativgesetze); Axiome (1-2a) und (1-2b) das Weglassen von Klammern (Assoziativgesetze), solange nicht $+$ und $\cdot$, wie in den Axiomen (1-3a) und (1-3b), gemischt auftreten (Distributivgesetze). Der durch (1-3a) beschriebene Vorgang wird auch als „Ausmultiplizieren", in Analogie dazu der durch (1-3b) beschriebene als „Ausaddieren" bezeichnet. Axiome (1-4a) und (1-4b) definieren das 1-Element und das 0-Element (Existenz der neutralen Elemente); Axiom (1-5a) mit (1-5b) definiert die „Überstreichung" (Existenz des Komplements). – Dass die Axiome für die in Tabelle 1–1 definierten logischen Operationen gültig sind, lässt sich durch Auswertung der Ausdrücke auf beiden Seiten des Gleichheitszeichens zeigen (siehe z. B. Tabelle 1–2 für (1-3b)).

Dualität. Den Axiomen ist eine Symmetrie zu eigen, die durch ihre paarweise Nummerierung betont ist. Sie ist gekennzeichnet durch Vertauschen von $\cdot$ und $+$ sowie 0 und 1 und wird als Dualität bezeichnet. Wenn, wie in (1-1a) bis (1-5b), zwei Ausdrücke äquivalent sind, so sind es auch die jeweiligen dualen

Ausdrücke. Dieses *Dualitätsprinzip* gilt nicht nur für die Axiome, sondern auch für alle Sätze.

1.1.4 Sätze

Aus den Axiomen der Boole'schen Algebra lässt sich eine Reihe von Sätzen ableiten, die zusammen mit den Axiomen als Rechenregeln zur Umformung von Ausdrücken dienen. (Einfacher als aus den Axiomen sind die Sätze durch Auswertung beider Gleichungsseiten zu beweisen.)

$$a \cdot a = a\,, \quad a + a = a\,, \tag{1-6a,b}$$

$$0 \cdot a = 0\,, \quad 1 + a = 1\,, \tag{1-7a,b}$$

$$a + a \cdot b = a\,, \quad a \cdot (a + b) = a\,, \tag{1-8a,b}$$

$$a + \bar{a} \cdot b = a + b\,, \quad a \cdot (\bar{a} + b) = a \cdot b\,, \tag{1-9a,b}$$

$$\overline{a \cdot b} = \bar{a} + \bar{b}\,, \quad \overline{a + b} = \bar{a} \cdot \bar{b}\,, \tag{1-10a,b}$$

$$\bar{\bar{a}} = a\,. \tag{1-11}$$

Sätze (1-6) bis (1-9) erlauben es, Boole'sche Ausdrücke zu vereinfachen bzw. Schaltungen hinsichtlich ihres Aufwands zu minimieren; (1-10a) und (1-10b), die De-Morgan'schen Regeln, erlauben es zusammen mit (1-11), die Operationen NICHT, UND und ODER durch NAND (negated AND) oder NOR (negated OR) auszudrücken, d. h. Schaltungen nur aus NAND-Schaltkreisen (siehe Bild 1–2c) oder nur aus NOR-Schaltkreisen (siehe Bild 1–2d) aufzubauen.

1.2 Boole'sche Funktionen

1.2.1 Von der Mengen- zur Vektordarstellung

Eine Funktion f bildet eine Menge E von Eingangselementen (Eingabemenge, Urmenge) in eine Menge A von Ausgangselementen (Ausgabemenge, Bildmenge) ab, formal beschrieben durch

$$f\colon E \to A\,,$$

wobei es sich hier stets um Mengen mit diskreten Elementen handelt. Zur Beschreibung von Funktionen gibt es eine Vielzahl an Darstellungsmitteln. Wenn die Anzahl der Eingangselemente nicht zu groß ist, bedient man sich gerne der Tabellendarstellung. Bei der in Tabelle 1–3a definierten Funktion sind zwar alle Eingangs- und Ausgangselemente aufgeführt, aber

Tabelle 1–3. Tabellendarstellungen einer Funktion. **a** Darstellung von $f\colon E \to A$ mit Elementen von Mengen; **b** Darstellung von f, aufgefasst als eine Funktion $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ mit den Werten Boole'scher *Vektoren* bzw. als *zwei* Funktionen $y_1 = f_1(x_1, x_2, x_3)$ und $y_2 = f_2(x_1, x_2, x_3)$ mit den Werten Boole'scher *Variablen*

a

$E \to A$	
e_0	a_0
e_1	a_1
e_2	a_1
e_3	a_2
e_4	a_1
e_5	a_2
e_6	a_2
e_7	a_3

b

$x_1\,x_2\,x_3$	$y_1\,y_2$
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

über ihre Art ist nichts ausgesagt; sie ergibt sich aus der jeweiligen Anwendung. In der elektronischen Datenverarbeitung sind die Elemente wegen der heute verwendeten Schaltkreise *binär* codiert, d. h., jedes Element von E und von A ist umkehrbar eindeutig durch 0/1-Kombinationen verschlüsselt; auf diese Weise entstehen aus den Eingangselementen Boole'sche Eingangs*variablen.* Tabelle 1–3b zeigt eine Codierung für die in Tabelle 1–3a definierte Funktion.

Beschreibung mit Boole'schen Vektoren. Funktionen mit binär codierten Elementen lassen sich durch Boole'sche Ausdrücke beschreiben, sie heißen dann Boole'sche Funktionen. Ihre Realisierung mit Schaltern (i. Allg. Transistoren) bezeichnet man als Schaltnetze (siehe 3).

Im einfachsten Fall ist eine Boole'sche Funktion von n Veränderlichen eine Abbildung der 0/1-Kombinationen der n unabhängigen Variablen x_1, $x_2, \ldots, x_n$ (Eingangsvariablen, zuweilen auch kurz: Eingänge) in die Werte 0 und 1 einer abhängigen Variablen y (Ausgangsvariable, zuweilen kurz: Ausgang). Fasst man die Eingangsvariablen zu einem Boole'schen Vektor zusammen (Eingangsvektor $\boldsymbol{x}$), so lässt sich dies kompakt durch $y = f(\boldsymbol{x})$ beschreiben. Liegen m Funktionen $y_1 = f_1(\boldsymbol{x}), \ldots, y_m = f_m(\boldsymbol{x})$ mit m Ausgangsvariablen vor (Ausgangsvektor $\boldsymbol{y}$), so lassen sich diese ebenso zusammenfassen und durch $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ beschreiben. Es entsprechen sich also

$$f\colon E \to A \quad \text{und} \quad \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})\,.$$

Darin sind die binär codierten Elemente von E die Werte von $\boldsymbol{x}$ und die binär codierten Elemente von A die Werte von $\boldsymbol{y}$.
Eine Funktion, bei der für sämtliche 0/1-Kombinationen ihrer Eingangsvariablen die Funktionswerte ihrer Ausgangsvariable(n) definiert sind heißt vollständig definiert (totale Funktion), andernfalls unvollständig definiert (partielle Funktion).

1.2.2 Darstellungsmittel

Für Boole'sche Funktionen sind verschiedene Darstellungsformen möglich, die meist ohne Informationsverlust ineinander transformierbar sind. Deswegen bedeutet der Entwurf eines Schaltnetzes i. Allg. die *Transformation* von verbalen *Angaben* über die Funktion in eine wirtschaftlich akzeptable *Schaltung* für die Funktion; d. h. die Transformation der Beschreibung ihrer Funktionsweise in die Beschreibung ihrer Schaltungsstruktur.

Tabellen (Wertetabellen, Wahrheitstabellen; Tabelle 1–3b). In der Tabellendarstellung steht in jeder Zeile links eine 0/1-Kombination der Eingangsvariablen (ein Wert des Eingangsvektors, Eingangswert), der rechts die zugehörigen Werte der Ausgangsvariablen (der Wert des Ausgangsvektors, Ausgangswert) zugeordnet sind. Tabellenzeilen mit demselben Ausgangswert werden manchmal zu einer Zeile zusammengefasst, wobei eine Eingangsvariable, die den Ausgangswert nicht beeinflusst, durch einen Strich in der Tabelle gekennzeichnet wird. – Bei unvollständig definierten Funktionen sind die nicht definierten Wertezuordnungen nicht in der Tabelle enthalten.

Tafeln (Karnaugh-,Veitch-, auch kurz KV-Diagramme; Bild 1–1a). Bei der Tafeldarstellung sind die Eingangsvariablen entsprechend der matrixartigen Struktur der Tafeln in zwei Gruppen aufgeteilt. Die 0/1-Kombinationen der einen Gruppe werden nebeneinander den Spalten, die der anderen Gruppe zeilenweise untereinander den Zeilen der Tafel zugeordnet. Jede Tafel hat so viele Felder, wie es mögliche Kombinationen der Eingangswerte gibt, d. h. bei n Variablen 2^n Felder. In die Felder werden die Ausgangswerte eingetragen: entweder zusammengefasst als Vektor in eine einzige Tafel oder als dessen einzelne Komponenten in so viele Tafeln, wie der Vektor Komponenten hat. – Bei unvollständig definierten Funktionen entsprechen den nicht definierten Wertezuordnungen leere Felder (Leerstellen), sie werden auch als „don't cares" bezeichnet und spielen bei der Minimierung von Funktionsgleichungen eine wichtige Rolle.

Gleichungen (Bild 1–1b). In der Gleichungsdarstellung stehen die Ausgangsvariablen links des Gleichheitszeichens, und die Eingangsvariablen erscheinen innerhalb von Ausdrücken rechts des Gleichheitszeichens. Bei mehreren Ausgangsvariablen hat das Gleichungssystem so viele Gleichungen, wie der Ausgangsvektor Komponenten hat. – Bei unvollständig definierten Funktionen kann der eingeschränkte Gültigkeitsbereich einer Gleichung als Bedingung ausgedrückt werden.

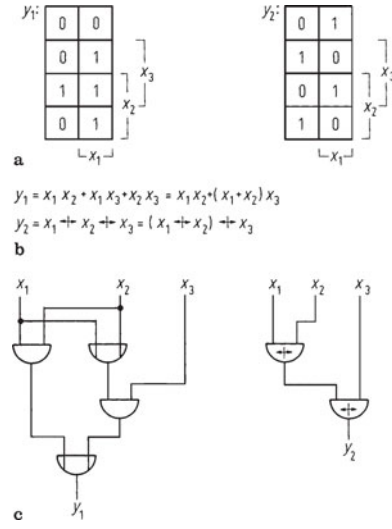


Bild 1–1. Darstellungsmittel für Boole'sche Funktionen am Beispiel der Funktion in Tabelle 1–3b. **a** Tafeln; **b** Gleichungen; **c** Blockbilder. Die abgebildeten Funktionen sind in mehrfacher Weise interpretierbar: 1. $y_1 = 1$, wenn 2 oder mehr der 3 Kandidaten x_i zustimmen (2-aus-3-Voter), 2. $y_2 = 1$, wenn die Quersumme der Dualzahl $x_3x_2x_1$ ungerade ist (Paritätsprüfung), 3. y_1 als Übertrag und y_2 als Summe bei der Addition der drei Dualziffern x_1, x_2, x_3 (Volladdierer, siehe 3.2.2)

Blockbilder (Strukturbilder, Schaltbilder; Bild 1–1c). Blockbilder beschreiben sowohl die formelmäßige Gliederung wie die schaltungsmäßige Struktur einer Boole'schen Funktion und haben somit eine Brückenfunktion beim Schaltnetzentwurf. Sie werden z. B. mit den in Tabelle 1–1 dargestellten Symbolen gezeichnet, die entsprechend der „Klammerstruktur" miteinander zu verbinden sind. Die Eingangsvariablen sind die Eingänge des Schaltnetzes. Die Negation einer Variablen wird entweder durch einen Punkt am Symbol oder durch Überstreichung der Variablen dargestellt. Die Ausgangsvariablen sind die Ausgänge des Schaltnetzes. – Im Blockbild kann die Eigenschaft einer Funktion, unvollständig definiert zu sein, nicht zum Ausdruck gebracht werden.

Verallgemeinerung der Darstellungsmittel. Nicht alle der aufgeführten Darstellungsmittel sind unbeschränkt anwendbar. Tabellen sind durch ihre Zeilenzahl begrenzt. Zeilen mit gleichen Ausgangswerten werden deshalb gerne zu einer Zeile zusammengefasst. Tafeln sind auf etwa sechs bis acht Eingangsvariablen begrenzt. Gleichungen bedürfen einer gewissen Übersichtlichkeit; sie lassen vektorielle Beschreibungen nur sehr eingeschränkt zu. – Bei umfangreichen Aufgabenstellungen abstrahiert man deshalb von der Boole'schen Algebra und wählt anwendungsspezifische Beschreibungsformen, z. B. die in höheren Programmiersprachen oder auf der sog. Registertransfer-Ebene üblichen Ausdrucksmittel (siehe 4), wie die Gleichung $Z = X + Y$ bzw. ein „+"-Kästchen oder das „+"-Zeichen für die Addition von zwei Dualzahlen (siehe z. B. in Bild 4–10). Sie sind Ausgangspunkt für den Entwurf von operativen Schaltnetzen, wie die folgende Kette von Darstellungstransformationen zeigt: „+"-Zeichen in Bild 4–10 → Bild 3–1a → Bild 3–1b → Bild 1–1a → Bild 1–1b → Bild 1–1c in der 3. Interpretation.

1.3 Normal- und Minimalformen

1.3.1 Kanonische Formen Boole'scher Funktionen

Unter den zahlreichen Möglichkeiten, eine Boole'sche Funktion durch einen Boole'schen Ausdruck zu beschreiben, gibt es bestimmte, die sich durch Übersichtlichkeit und Einfachheit besonders auszeichnen.

Normalformen. Jede Boole'sche Funktion kann in zwei charakteristischen Formen geschrieben werden: 1. als disjunktive Normalform, das ist eine i. Allg. mehrstellige Disjunktion (ODER-Verknüpfung) von i. Allg. mehrstelligen Konjunktionstermen (UND-Verknüpfungen): Bild 1–2a; 2. als konjunktive Normalform, das ist eine i. Allg. mehrstellige Konjunktion (UND-Verknüpfung) von i. Allg. mehrstelligen Disjunktionstermen (ODER-Verknüpfungen): Bild 1–2b.
Die disjunktive Normalform kann mit NAND-Gliedern (Bild 1–2c) und die konjunktive Normalform mit NOR-Gliedern (Bild 1–2d) dargestellt werden. Das ist deshalb wichtig, weil in der Technik vielfach nur NOR- oder NAND-Schaltkreise zur Verfügung stehen. Die Eingangsvariablen sind unmittelbar in normaler oder negierter Form an die Verknüpfungsglieder angeschlossen (man beachte den Wechsel der Überstreichung bei x_4).

Ausgezeichnete Normalformen. Enthalten alle Terme einer Normalform sämtliche Variablen der

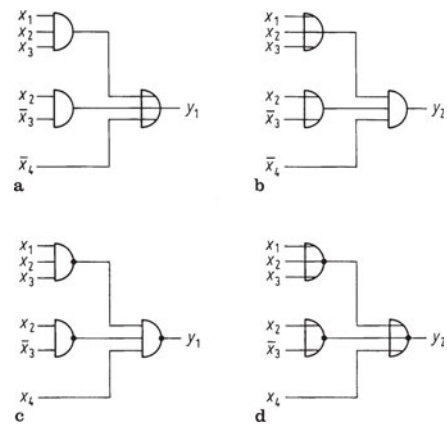


Bild 1–2. Blockbilder von Normalformen sowie davon abgeleiteter Formen. **a** Disjunktive Normalform (hier $y_1 = x_1x_2x_3 + x_2\bar{x}_3 + \bar{x}_4$); **b** konjunktive Normalform (hier $y_2 = (x_1 + x_2 + x_3) \cdot (x_2 + \bar{x}_3) \cdot \bar{x}_4$); **c** NAND/NAND-Form; **d** NOR/NOR-Form. Die in a und b bzw. c und d abgebildeten Formen sind jeweils dual, nicht äquivalent. Die in a und c bzw. b und d abgebildeten Formen sind hingegen äquivalent

Funktion genau einmal (normal oder negiert) und sind gleiche Terme nicht vorhanden, so liegt eine eindeutige Struktur vor, die als ausgezeichnete disjunktive bzw. ausgezeichnete konjunktive Normalform bezeichnet wird. – Jede Boole'sche Funktion lässt sich von einer in die andere Normalform umformen, mit den angegebenen Rechenregeln allerdings z. T. nur unter erheblichem Rechenaufwand; besser geht es unter Zuhilfenahme von Tafeln.

1.3.2 Minimierung von Funktionsgleichungen

Die Minimierung von Funktionsgleichungen dient zur Vereinfachung von Schaltnetzen. Sie hat heute wegen der geringeren Kosten der Transistoren innerhalb hochintegrierter Schaltungen nicht mehr die Bedeutung wie früher. Trotzdem wird sie beim Schaltungsentwurf, insbesondere beim automatisierten, computergestützten Entwurf, zur optimalen Nutzung der Chipfläche eingesetzt; und auch in der Programmierung, z. B. zur übersichtlicheren Formulierung bedingter Anweisungen, kann sie nützlich sein.
Die Minimierung besteht aus zwei Teilen: 1. dem Aufsuchen sämtlicher Primimplikanten, das ergibt UND-Verknüpfungen mit wenigen Eingängen, und 2. der Ermittlung der minimalen Überdeckung der Funktion, das ergibt wenige UND-Verknüpfungen und damit auch eine ODER-Verknüpfung mit wenigen Eingängen.

Primimplikant. Für jeden Konjunktionsterm einer Boole'schen Funktion in disjunktiver Normalform gilt, dass, wenn er den Wert 1 hat, auch die Funktion selbst den Wert 1 hat. Mit anderen Worten, jeder Konjunktionsterm impliziert die Funktion, man sagt, er ist Implikant der Funktion. Lässt sich aus einem solchen Implikanten keine Variable herausstreichen, ohne den Funktionswert zu ändern, so heißt er Primimplikant oder Primterm. In der Tafel sind Primterme anschaulich „rechteckige" Felder (ggf. unzusammenhängend) mit „maximal vielen" Einsen unter Einbeziehung von Leerstellen, die sich durch einen einzigen Konjunktionsterm darstellen lassen (z. B. sind $\bar{a}\bar{b}\bar{c}\bar{d}$, $a\bar{b}c$ und ad (auch $a\bar{b}c$ und bd), nicht aber z. B. $a\bar{b}cd$ Primterme der Funktion f entsprechend den drei (fünf) umrandeten Feldern in Bild 1–3).

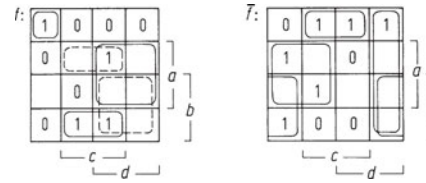


Bild 1–3. Tafeln unvollständig definierter Funktionen f und $\bar{f}$ mit vier Variablen. Die gestrichelt eingerahmten Primtermfelder sind zur Gleichungsdarstellung der Funktion unnötig

Minimale Überdeckung. Alle Konjunktionsterme einer Funktion, disjunktiv zusammengefasst, stellen die Funktion in ihrer Gesamtheit dar, man sagt, sie bilden eine Überdeckung der Funktion. Lässt sich aus einer solchen Überdeckung kein Term streichen, ohne die Funktion zu ändern, so heißt sie minimale Überdeckung. In der Tafel gibt es dann kein umrandetes Feld, das durch zwei oder mehrere andere „erzeugt" wird (z. B. ist $f = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}bc + ad$, nicht aber $f = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}bc + ad + bd$, eine minimale Überdeckung der Funktion f aus Bild 1–3).

Minimale Normalform. Die Minimierung führt gewöhnlich auf minimale disjunktive Normalformen. Programmierbare Verfahren zur exakten Minimierung folgen strikt der oben beschriebenen Zweiteilung (siehe z. B. [1]). Bei programmierten heuristischen Verfahren (siehe z. B. [2]) sowie bei manuellen grafischen Verfahren werden hingegen meist beide Teile zusammengefasst, wobei in Kauf genommen wird, gelegentlich nicht ganz das absolute Minimum an Verknüpfungen zu erhalten. Zur grafischen Minimierung wird die Funktion als Tafel dargestellt, und es werden alle jene Konjunktionsterme disjunktiv verknüpft herausgeschrieben, die jeweils maximal viele Einsen/Leerstellen umfassen, und zwar so lange, bis alle Einsen der Tafel berücksichtigt sind (z. B. entsteht gemäß Bild 1–3 $f = ad + \bar{a}bc + \bar{a}\bar{d}\bar{c}\bar{d}$). – Der hier skizzierte Minimierungsprozess bezieht sich auf Funktionen mit einem Ausgang. Funktionen mit mehreren Ausgängen werden grafisch komponentenweise, algorithmisch hingegen als Ganzes minimiert. Solche Funktionen spielen bei der Chipflächenreduzierung von Steuerwerken (siehe 4.4.1) eine gewisse Rolle.

„Rechnen" mit Tafeln. Die Minimierung wird auch beim Rechnen mit Boole'schen Funktionen angewendet, da es anderenfalls äußerst mühsam wäre, „don't cares" in den Rechenprozess einzubeziehen.
Beispiel: Es soll die zu $f = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}bc + ad$ negierte Funktion $\bar{f}$ gebildet werden, und zwar unter Berücksichtigung der „don't cares" aus Bild 1–3. Nach Ablesen der „günstigsten" Primterme ergibt sich $\bar{f} = \bar{a}\bar{b}c + a\bar{d} + b\bar{c} + \bar{c}d$.

1.4 Boole'sche Algebra und Logik

In der mathematischen Logik wird die Boole'sche Algebra zur Beschreibung der logischen Struktur von Aussagen benutzt. Statt der Symbole 0 und 1 benutzt man deshalb meist *falsch* und *wahr*, F und W bzw. in den Programmiersprachen *false* und *true* oder F und T.
In der mathematischen Logik sind die folgenden Symbole üblich: Negation ¬, Konjunktion ∧, Disjunktion ∨, Implikation →, Äquivalenz ↔. Aus Gründen der Einheitlichkeit wird im Folgenden hier auch die Symbolik der Boole'schen Algebra, d. h. auch die der Schaltalgebra benutzt.

Beispiel. Der Satz: „Wenn die Sonne scheint und es warm ist oder wenn ich müde bin und nicht schlafen kann, gehe ich spazieren." ist eine logische Verknüpfung der elementaren Aussagen „Die Sonne scheint" (A), „Es ist warm" (B), „Ich bin müde" (C), „Ich kann schlafen" (D), „Ich gehe spazieren" (E). Er wird als Formel der Aussagenlogik so geschrieben:

$$(A \wedge B) \vee (C \wedge \neg D) \rightarrow E \quad \text{bzw.} \quad (A \cdot B + C \cdot \overline{D}) \rightarrow E \ .$$

1.4.1 Begriffe

In der formalen Logik wird allein die *Struktur* von Aussagen betrachtet, nicht die Frage, ob sie inhaltlich wahr oder falsch sind. Die klassische formale Logik lässt für jede Aussage nur die beiden Möglichkeiten *wahr* und *falsch* zu („tertium non datur"); andere Logiksysteme (intuitionistische Logik, mehrwertige Logik, modale Logik) lockern diese Einschränkung. In der klassischen Logik unterscheidet man *Aussagenlogik* und *Prädikatenlogik*. Die Aussagenlogik betrachtet nur die Verknüpfungen elementarer Aussagen, d. h. ganzer Sätze. In der Prädikatenlogik kommt die Aufteilung der Sätze in Subjekte (Individuen) und Prädikate (Boole'sche Funktionen von Subjekten) und die Einführung von Quantoren hinzu. Da sich logische Formeln nach den Regeln der Boole'schen Algebra kalkülmäßig transformieren, z. B. vereinfachen oder in Normalformen überführen lassen, spricht man auch von Aussagen- und Prädikaten*kalkül*. Die folgenden Ausführungen beziehen sich hauptsächlich auf die einfachere, für viele Anwendungen aber ausreichende Aussagenlogik.
Die Wahrheit einer zusammengesetzten Formel der Aussagenlogik hängt nur von der Wahrheit ihrer Elementaraussagen ab. Dabei sind drei Fälle zu unterscheiden:

- *Allgemeingültigkeit* (*Tautologie*): Die Formel ist, unabhängig von der Wahrheit ihrer Elementaraussagen, immer wahr. Beispiel: „Wenn der Hahn kräht auf dem Mist, ändert sich das Wetter, oder es bleibt, wie es ist." ($H \rightarrow \overline{W} + W$).
- *Unerfüllbarkeit* (*Kontradikation*): Die Formel ist, unabhängig von der Wahrheit oder Falschheit ihrer Elementaraussagen, immer falsch. Beispiel: „Der Hahn kräht auf dem Mist und kräht nicht auf dem Mist." ($H \cdot \overline{H}$).
- *Erfüllbarkeit*: Es gibt mindestens eine Belegung der Elementaraussagen mit *wahr* oder *falsch*, die die Formel wahr macht. Beispiel: „Wenn der Hahn kräht auf dem Mist, ändert sich das Wetter." ($H \rightarrow \overline{W}$).

Wenn X eine Formel der Aussagen- oder Prädikatenlogik ist, gelten folgende wichtige Beziehungen:

X ist allgemeingültig $\leftrightarrow \overline{X}$ ist unerfüllbar

X ist unerfüllbar $\leftrightarrow \overline{X}$ ist allgemeingültig

X ist erfüllbar $\leftrightarrow \overline{X}$ ist nicht allgemeingültig

Die große Bedeutung der formalen Logik für Mathematik und Informatik beruht vor allem darauf, dass sich mit ihr die Begriffe des logischen Schließens und des mathematischen Beweisens formal fassen lassen. Darauf bauen Verfahren zum automatischen Beweisen und die sog. logischen Programmiersprachen, wie Prolog, auf.