Rechneraufbau und Rechnerstrukturen

Von
Walter Oberschelp,
Gottfried Vossen

10., überarbeitete und erweiterte Auflage

Oldenbourg Verlag München Wien

Prof. Dr. Gottfried Vossen lehrt seit 1993 Informatik am Institut für Wirtschaftsinformatik der Universität Münster. Er studierte, promovierte und habilitierte sich an der RWTH Aachen und war bzw. ist Gastprofessor u.a. an der University of California in San Diego, USA, an der Karlstad Universitet in Schweden, an der University of Waikato in Hamilton, Neuseeland sowie am Hasso-Plattner-Institut für Softwaresystemtechnik in Potsdam. Er ist europäischer Herausgeber der bei Elsevier erscheinenden Fachzeitschrift Information Systems.

Prof. Dr. Walter Oberschelp studierte Mathematik, Physik, Astronomie, Philosophie und Mathematische Logik. Nach seiner Habilitation in Hannover lehrte er als Visiting Associate Professor an der University of Illinois (USA). Nach seiner Rückkehr aus den USA übernahm er den Lehrstuhl für Angewandte Mathematik an der RWTH Aachen, den er bis zu seiner Emeritierung im Jahr 1998 inne hatte.

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0
www.oldenbourg-wissenschaftsverlag.de

Lektorat: Margit Roth
Herstellung: Anna Grosser
Umschlagkonzeption: Kraxenberger Kommunikationshaus, München
Gedruckt auf säure- und chlorfreiem Papier
Druck: Oldenbourg Druckerei Vertriebs GmbH & Co. KG
Bindung: R. Oldenbourg Graphische Betriebe Binderei GmbH

ISBN 3-486-57849-9
ISBN 978-3-486-57849-2

Kapitel 3

Optimierung und Test von Schaltnetzen

Schaltnetze lassen sich nach den Ausführungen des letzten Kapitels beschleunigen, d. h. in ihrem zeitlichen Verhalten verbessern, indem man sie mit zusätzlicher Hardware ausstattet. Wir wollen als nächstes ein gewissermaßen dazu „komplementäres" Problem behandeln, nämlich das der *Vereinfachung* von Schaltnetzen, d. h. das Einsparen von Hardware, wann immer es möglich ist, aber ohne Veränderung des Verhaltens eines Schaltnetzes. Ferner behandeln wir Techniken zur Fehlerdiagnose und das Phänomen der Hasards.

3.1 Vereinfachung von Schaltnetzen

3.1.1 Das Verfahren von Karnaugh

Wir haben bereits in Kapitel 1 etwa in Beispiel 1.14 gesehen, wie man Schaltfunktionen, die z. B. in DNF gegeben sind, durch Anwendung der Rechenregeln der Booleschen Algebra vereinfachen kann. Wir betrachten zunächst zwei Beispiele, welche eine Anwendung von Satz 1.4 darstellen:

Beispiel 3.1 (a)

$$\begin{aligned} f(x_1, x_2, x_3) &= \overline{x}_1 x_2 x_3 + x_1 x_2 x_3 \\ &= \underbrace{(\overline{x}_1 + x_1)}_{=1} x_2 x_3 \\ &= x_2 x_3 \end{aligned}$$

Hier wurde offensichtlich folgende Vereinfachungsregel, welche auch unter dem Namen *Resolutionsregel* bekannt ist, angewendet:

> *Kommen in einer SOP (Sum of Products, vgl. Kapitel 1, also einer disjunktiven Form) zwei Summanden vor, welche sich in* genau einer *komplementären Variablen unterscheiden, so können diese beiden Summanden durch den ihnen gemeinsamen Teil ersetzt werden.*

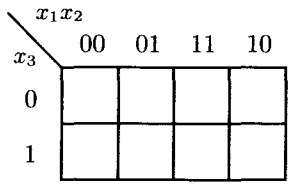

Abbildung 3.1: Karnaugh-Diagramm-Schema für $n = 3$.

(b) Die Resolutionsregel darf auch mehrfach angewendet werden, z. B.

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= x_1\overline{x}_2x_3x_4 + x_1\overline{x}_2\overline{x}_3x_4 + x_1x_2x_3x_4 + \overline{x}_1\overline{x}_2\overline{x}_3x_4 + \overline{x}_1\overline{x}_2x_3x_4 \\ &= x_1\overline{x}_2x_4 + x_1x_3x_4 + \overline{x}_2\overline{x}_3x_4 + \overline{x}_1\overline{x}_2x_4 \\ &= \overline{x}_2x_4 + x_1x_3x_4 + \overline{x}_2\overline{x}_3x_4 \end{aligned}$$

□

Eine mehrfache Anwendung der Vereinfachungsregel beruht dabei auf dem Gesetz $x + x = x$, welches das Verdoppeln von Summanden erlaubt. Wir werden nun ein graphisches Verfahren nach Karnaugh vorstellen, mit dessen Hilfe man leicht eine Übersicht über alle möglichen Resolutionen zu einer gegebenen Booleschen Funktion der Stellenzahl ≤ 4 erhält:

Definition 3.1 Ein *Karnaugh-Diagramm* (engl. *Karnaugh map* bzw. kurz *K-map*) einer Booleschen Funktion $f : B^n \to B$ mit $n \in \{3, 4\}$ ist eine graphische Darstellung der Funktionstafel von f durch eine 0-1-Matrix der Größe 2×4 für $n = 3$ bzw. 4×4 für $n = 4$, deren Spalten mit den möglichen Belegungen der Variablen x_1 und x_2 und deren Zeilen mit den möglichen Belegungen der Variablen x_3 bzw. x_3 und x_4 beschriftet sind (vgl. Abbildung 3.1 bzw. 3.2). Die Reihenfolge der Beschriftung erfolgt dabei so, dass sich zwei zyklisch benachbarte Spalten oder Zeilen nur in genau einer Komponente unterscheiden. (Zyklisch benachbart heißt, dass auch obere und untere Zeile bzw. linke und rechte Spalte als benachbart angesehen werden.)

In die entsprechenden Felder der Matrix werden nun die Funktionswerte von f eingetragen, wobei es ausreicht, nur die Einsen tatsächlich zu markieren. Jedem Minterm von f mit einschlägigem Index entspricht dann genau eine 1 im Karnaugh-Diagramm von f und umgekehrt. Folglich entsprechen zwei zyklisch benachbarte Einsen zwei Mintermen, welche sich in genau einer komplementären Variablen unterscheiden und auf die somit die Resolutionsregel angewendet werden kann. Zwei solche Einsen bilden einen so genannten „Zweierblock", und der durch Resolution entstehende Term hat gerade eine Variable weniger als die ihm zugrunde liegenden Minterme. Diese Beobachtung lässt sich verallgemeinern auf Einer-, Vierer-, Achter-, und Sechzehner-Blöcke wie folgt: Rechteckige $2^r \times 2^s$-Blöcke ($r, s \in \{0, 1, 2\}$) von zyklisch benachbarten Einsen entsprechen $2^r \cdot 2^s$ Mintermen, welche sich paarweise höchstens in $r + s$ Variablen

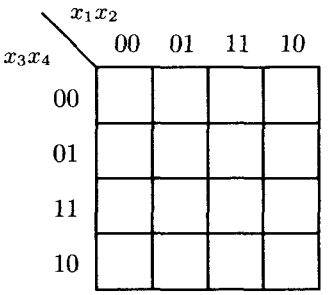

Abbildung 3.2: Karnaugh-Diagramm-Schema für $n = 4$.

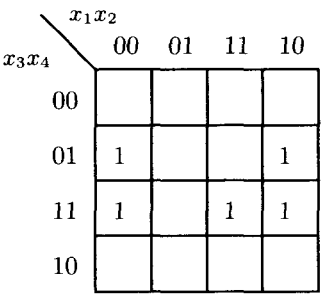

Abbildung 3.3: Karnaugh-Diagramm zu Beispiel 3.1 (b).

unterscheiden, wobei alle Möglichkeiten des negierten bzw. nicht negierten Auftretens dieser Variablen vorkommen. Daher lässt sich die Summe dieser Minterme durch wiederholte Resolution zu dem Term vereinfachen, welcher gemeinsamer Bestandteil aller dieser Minterme ist. Die Gestalt dieses Terms ist dabei aus der Rand-Beschriftung des Karnaugh-Diagramms abzulesen.

Beispiel 3.1 (b) (Fortsetzung):

$$f = x_1\overline{x}_2x_3x_4 + x_1\overline{x}_2\overline{x}_3x_4 + x_1x_2x_3x_4 + \overline{x}_1\overline{x}_2\overline{x}_3x_4 + \overline{x}_1\overline{x}_2x_3x_4$$

f nimmt also für folgende Argumente den Wert 1 an: 1011, 1001, 1111, 0001, 0011. Daher erhält man das in Abbildung 3.3 gezeigte Karnaugh-Diagramm. □

Das oben Gesagte bedeutet für das Auffinden einer vereinfachten Darstellung von f aus dem Karnaugh-Diagramm folgendes: Man überdecke alle im Diagramm auftretenden Einsen durch möglichst große „Resolutions-Blöcke" der Form $2^r \times 2^s$, d. h.

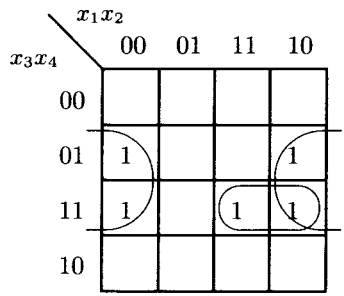

Abbildung 3.4: Überdeckung der Einsen in Beispiel 3.1 (b).

man markiere maximal große Rechtecke von Einsen, wähle von diesen so viele aus, dass jede Eins mindestens in einem Block vorkommt, und bilde die Summe der diesen Blöcken entsprechenden Terme. Für das Diagramm aus Abbildung 3.3 erhalten wir die in Abbildung 3.4 gezeigte Situation. In diesem Diagramm hängt der Vierer-Block nicht von x_1 und nicht von x_3 ab, da seine Einsen sowohl dort stehen, wo x_1 bzw. x_3 Null ist, als auch dort, wo diese Variablen den Wert Eins annehmen. Der ihm entsprechende Term enthält also nur x_2 und x_4 und hat, da die Einsen gerade in den Spalten (Zeilen) für $x_2 = 0$ ($x_4 = 1$) stehen, die Form $\overline{x}_2x_4$. Eine analoge Überlegung liefert für den Zweierblock den Term $x_1x_3x_4$, so dass wir erhalten:

$$f = \overline{x}_2x_4 + x_1x_3x_4.$$

Allgemein liefert ein Block mit 2^k Einsen ($k \in \{0, \cdots, 4\}$) einen Term mit $n - k$ Variablen. Das Auffinden des einem Block entsprechenden Terms wird etwas erleichtert, wenn man als Zeilen- bzw. Spaltenbeschriftung statt der Variablenwerte die Variablen selbst wie folgt verwendet: Man bezeichne die Spalten bzw. Zeilen mit x, für welche die Variable x den Wert 1 annimmt (und die anderen mit $\overline{x}$).

Beispiel 3.2

$$\begin{aligned} f(x_1, x_2, x_3, x_4) \;=\; & \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3x_4 + x_1x_2\overline{x}_3x_4 \\ & + \overline{x}_1x_2x_3x_4 + x_1x_2x_3x_4 + \overline{x}_1\overline{x}_2x_3\overline{x}_4 + x_1\overline{x}_2x_3\overline{x}_4 \end{aligned}$$

Ein Karnaugh-Diagramm zu f (mit alternativer Beschriftung) ist in Abbildung 3.5 gezeigt. In diesem Beispiel sind alle Einsen durch zwei Viererblöcke zu überdecken, und wir erhalten als vereinfachte Form

$$f = x_2x_4 + \overline{x}_2\overline{x}_4 \; .$$

(Diese Funktion wird in Kapitel 4 eingehend weiter diskutiert.) □

Bei der Auswahl der Blöcke, welche alle Einsen in einem Diagramm überdecken, ist es unter Umständen nicht sinnvoll, unbedingt die größten Blöcke (bzgl. der Anzahl

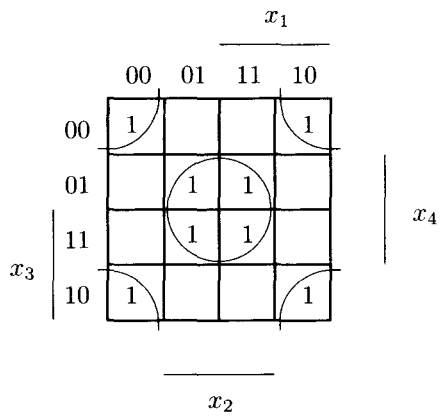

Abbildung 3.5: Karnaugh-Diagramm zu Beispiel 3.2.

der Einsen) zu berücksichtigen, z. B. in der in Abbildung 3.6 gezeigten Situation: Die isoliert stehenden Einsen in den Rand-Zeilen bzw. -Spalten sind durch Zweier-Blöcke überdeckbar, welche dann bereits die Einsen des mittleren Vierer-Blocks erfassen.

Bisher haben wir bei der Behandlung von Schaltnetzen immer vorausgesetzt, dass die zu realisierende Boolesche Funktion *total* war, d. h. dass für $f : B^n \to B$ der Definitionsbereich von f ganz B^n umfasste oder — anders ausgedrückt — dass alle 2^n Elemente von B^n als Argumente für f möglich waren. Häufig tritt jedoch der Fall ein, dass nur gewisse der 2^n Inputs, etwa k, möglich sind, und somit die Funktionswerte von f für $2^n - k$ Argumente durch die betreffende Aufgabenstellung nicht festgelegt werden (wie z. B. beim früher besprochenen Problem „Euler-Kreis in Graphen mit 5 Punkten"). Diese restlichen Argumente-Tupel bezeichnet man als *„Don't-Care"*-Fälle. Ist eine gegebene Boolesche Funktion f *partiell*, so kann man beim Entwurf eines Schaltnetzes für f offensichtlich für die Don't-Care-Argumente willkürlich Funktionswerte festsetzen d. h. f wird zu einer totalen Funktion vervollständigt („fortgesetzt"). Ist f drei- oder vierstellig und will man ein möglichst einfaches Schaltnetz für f mit dem Karnaugh-Verfahren entwerfen, so bietet sich an, für Don't-Care-Argumente den Funktionswert 1 zu unterstellen, wenn dadurch bereits vorhandene Blöcke vergrößert werden können. Selbstverständlich brauchen Don't-Care-Stellen nicht durch Blöcke überdeckt zu werden.

Beispiel 3.3 Sei f für $x \in \{0, \ldots, 9\}$ definiert durch:

$$f(x) := \begin{cases} 1 & \text{falls } x \in \{1, 5, 8, 9\} \\ 0 & \text{sonst} \end{cases}$$

Zur Binärcodierung der 10 Argumente werden hier vierstellige Dualzahlen benötigt, mit denen man jedoch $2^4 = 16$ Argumente codieren könnte, so dass wir sechs Don't-Care-Fälle erhalten. In Tabelle 3.1 bzw. im Karnaugh-Diagramm von Abbildung 3.7

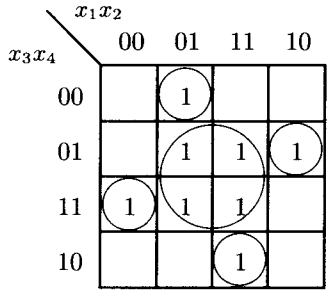

Abbildung 3.6: Karnaugh-Diagramm mit „isolierten" Einsen.

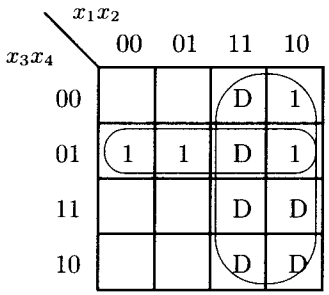

Abbildung 3.7: Karnaugh-Diagramm zu Beispiel 3.3.

sind diese durch „D" gekennzeichnet, wobei D hier dem Funktionswert 1 entsprechen soll. Unter Ausnutzung der Don't-Cares erhält man:

$$f(x_1, x_2, x_3, x_4) = x_1 + \overline{x}_3 x_4.$$

Zum Vergleich geben wir die Darstellung von f an, welche man ohne Ausnutzung der Don't-Cares aus dem Karnaugh-Diagramm erhält:

$$f(x_1, x_2, x_3, x_4) = x_1\overline{x}_2\overline{x}_3 + \overline{x}_1\overline{x}_3 x_4.$$

□

Es sei an dieser Stelle bemerkt, dass man aus dem Karnaugh-Diagramm einer drei- oder vierstelligen Funktion auch auf einfache Weise (nicht notwendig komple-

Tabelle 3.1: Boolesche Funktion aus Beispiel 3.3.

x	x_1	x_2	x_3	x_4	f
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
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	1
9	1	0	0	1	1
A	1	0	1	0	D
B	1	0	1	1	D
C	1	1	0	0	D
D	1	1	0	1	D
E	1	1	1	0	D
F	1	1	1	1	D

mentfreie) Ringsummen-Darstellungen gewinnen kann. Dazu wählt man zunächst zyklisch benachbarte Blöcke aus, die nunmehr inhomogen sein dürfen (d. h. außer Einsen dürfen sie auch Nullen enthalten). Nun hat man darauf zu achten, dass jede 1 durch eine ungerade und jede 0 durch eine gerade Anzahl der ausgewählten Blöcke erfaßt wird. Für die Boolesche Funktion aus Beispiel 3.1 (b) erhält man etwa (vgl. Abbildung 3.8):

$$f = x_4 \dotplus \overline{x}_1 x_2 x_4 \dotplus x_1 x_2 \overline{x}_3 x_4.$$

Auf eine Begründung dieses Vorgehens sei hier verzichtet (vgl. Aufgabe 3.8).

Außerdem sei darauf hingewiesen, dass die Beschriftung der Ränder von Karnaugh-Diagrammen gemäß Definition 3.1 so zu erfolgen hat, dass sich zwei (zyklisch) benachbarte Spalten oder Zeilen nur in genau einer Komponente unterscheiden. Beim Übergang von einer beliebigen Stelle im Karnaugh-Diagramm zu einer benachbarten in vertikaler oder horizontaler Richtung ändert sich also immer nur genau ein Bit. Dieses „Bauprinzip" ist dem *Gray-Code* entliehen, der wie die (dem Leser inzwischen vertraute) natürliche Binärcodierung zur Verschlüsselung z. B. der Dezimalziffern benutzt werden kann und auch heute noch insbesondere bei der Analog/Digital-Umwandlung (A/D-Umwandlung) eine Rolle spielt. Für die Dezimalziffern kann er z. B. wie in Tabelle 3.2 gewählt werden. Wesentliches Merkmal des Gray-Codes ist, dass sich die Codierungen zweier aufeinanderfolgender Ziffern und von 9 und 0 nur an genau einer der vier Stellen unterscheiden. Man beachte, dass dieser Code im Allgemeinen *nicht* eindeutig bestimmt ist. (Jedoch gibt es Verfahren, mit denen man Gray-Codes für Dualzahlen systematisch erzeugen kann, ohne die Wortlänge vergrößern zu müssen.) Bei der A/D-Umwandlung z. B. von Meßwerten in Bitfolgen bedeutet diese Eigenschaft, dass nur *eine* Bit-Stelle mit Unsicherheit behaftet ist, verursacht etwa durch nicht ganz exakte Positionierung eines Meßwertgebers.

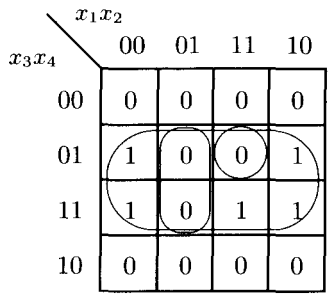

Abbildung 3.8: Karnaugh-Diagramm zu Beispiel 3.1 (b).

Tabelle 3.2: Mögliche Gray-Codierung der Dezimalziffern.

x	Gray-Codewort zu x	Alternative z. B.
0	0000	0000
1	0001	0001
2	0011	0011
3	0010	0010
4	0110	0110
5	0111	1110
6	0101	1010
7	0100	1011
8	1100	1001
9	1000	1000

3.1.2 Das Verfahren von Quine und McCluskey

Das im letzten Abschnitt vorgestellte Karnaugh-Verfahren ermöglicht das Auffinden einer einfacheren Darstellung als z. B. die DNF für Boolesche Funktionen der Stellenzahl ≤ 4. Dieses Verfahren lässt sich auch auf Boolesche Funktionen höherer Stellenzahl erweitern: so betrachtet man für $n = 5$ etwa zwei Oberflächen eines dreidimensionalen Würfels. Jedoch wird der geometrische Aufbau des Diagramms bereits für $n \geq 6$ zu kompliziert, um auf einfache Art überdeckende Blöcke von Einsen auffinden zu können. In diesem Abschnitt werden wir ein anderes Vereinfachungsverfahren vorstellen, welches für Boolesche Funktionen beliebiger Stellenzahl besser geeignet ist. Dazu stellen wir zunächst einige Begriffe bereit:

Definition 3.2 Eine Boolesche Funktion $f : B^n \to B$ liegt in *disjunktiver Form* vor, wenn f als Summe von Termen

$$\sum_{i=1}^{k} M_i \ , \ k \geq 1,$$

dargestellt ist. Dabei verstehen wir unter einem Term M_i künftig ein Produkt der Form

$$\prod_{j=1}^{l} x_{i_j}^{\alpha_j} \ , \ l \geq 1,$$

wobei x^α für $\alpha \in B$ wie in Definition 1.4 erklärt ist.

Wir haben bereits in Kapitel 1 darauf hingewiesen, dass man in der englischsprachigen Literatur in diesem Zusammenhang auch von der *Sum of Products*-Darstellung, kurz SOP, spricht, da es sich hierbei um eine Summe von Produkten handelt. Die disjunktive Normalform einer Booleschen Funktion $f : B^n \to B$ ist damit eine disjunktive (bzw. SOP-) Form, bei welcher jeder Term ein Minterm ist, d. h. jeder Term enthält alle n Variablen. Die Terme einer beliebigen disjunktiven Form von f enthalten im Allgemeinen weniger als n Variablen. Disjunktive Formen geben Anlaß zu so genannten *zweistufigen* Schaltungen: Auf der ersten Stufe werden die Werte der einzelnen (Produkt-) Terme mit Hilfe von (großen) Und-Gattern berechnet, auf der zweiten werden diese Ergebnisse durch ein (großes) Oder-Gatter verknüpft. Derartige Schaltungen sind für praktische Realisierungen besonders wünschenswert, da sie geringe Signal-Laufzeiten aufweisen (vgl. die Bypass-Schaltung in Abschnitt 2.5).

Definition 3.3 Sei $f : B^n \to B$ eine Boolesche Funktion, und sei d eine Darstellung von f in disjunktiver Form aus der Menge $D(f)$ aller Darstellungen von f. Für d erklären wir die *Kosten* $K(d)$ wie folgt:

(i) Für $d \equiv x_{i_1}^{\alpha_1} \cdot x_{i_2}^{\alpha_2} \cdot \ldots \cdot x_{i_t}^{\alpha_t} : K(d) := t - 1$
(ii) Für $d \equiv M_1 + \ldots + M_k : K(d) := (k-1) + \sum_{i=1}^{k} K(M_i)$

Bei dieser Festlegung der Kosten einer disjunktiven Form d stellen wir uns anschaulich ein Schaltnetz für d vor, welches aus Und- und Oder-Gattern und Invertern besteht. Die Kosten von d sind dann gerade die Anzahl der Und- und Oder-Gatter dieses Netzes, wobei wir annehmen, dass Inverter keine Kosten verursachen.

Beispiel 3.4 (a) Sei $f : B^n \to B$ in DNF dargestellt durch

$$d \equiv M_1 + \ldots + M_k.$$

Da jeder Term nun Minterm ist, hat d die Kosten

$$\begin{aligned} K(d) &= (n-1)k + k - 1 \\ &= n \cdot k - 1 \end{aligned}$$

(b) (Fortsetzung von Beispiel 3.2) Wegen (a) gilt für die DNF von $f : B^4 \to B$:

$$K(d) = 4 \cdot 8 - 1 = 31.$$

Für die mit Hilfe des Karnaugh-Verfahrens gewonnene disjunktive Form

$$d \equiv x_2x_4 + \overline{x}_2\overline{x}_4$$

erhält man

$$\begin{aligned} K(d) &= (2-1) + K(x_2x_4) + K(\overline{x}_2\overline{x}_4) \\ &= 1 + 1 + 1 \\ &= 3 \end{aligned}$$

□

Teil (b) von Beispiel 3.4 zeigt exemplarisch, wie das, was das Karnaugh-Verfahren leistet, allgemein formuliert werden kann als

Vereinfachungsproblem Boolescher Funktionen:

> Man bestimme zu einer gegebenen Booleschen Funktion $f : B^n \to B$, welche z. B. als Tabelle oder in DNF vorliege, eine sie darstellende disjunktive Form d mit minimalen Kosten $K(d)$.

Nach dem bisher Gesagten ist nun klar, dass es sich beim Karnaugh-Verfahren um ein Verfahren zur Lösung dieses Problems handelt für $n = 3$ oder $n = 4$. Für das angekündigte weitere Lösungsverfahren benötigen wir noch:

Definition 3.4 Sei $f : B^n \to B$ eine Boolesche Funktion. Ein Term M heißt *Implikant* von f, kurz $M \leq f$, falls $M(x) \leq f(x)$ für alle $x \in B^n$ gilt, d. h.

$$M(x) = 1 \;\Rightarrow\; f(x) = 1 \text{ für alle } x \in B^n.$$

Ein Implikant M von f heißt *Primimplikant* (von f), falls keine echte Verkürzung von M noch Implikant von f ist.

Beispiel 3.1 (b) (Fortsetzung) Implikanten von f sind: $x_1\overline{x}_2x_3x_4$, $x_1\overline{x}_2\overline{x}_3x_4$, $x_1x_2x_3x_4$, $\overline{x}_1\overline{x}_2\overline{x}_3x_4$, $\overline{x}_1\overline{x}_2x_3x_4$, $x_1\overline{x}_2x_4$, $x_1x_3x_4$, $\overline{x}_1\overline{x}_2x_4$, $\overline{x}_2\overline{x}_3x_4$, $\overline{x}_2x_4$. Primimplikanten sind $\overline{x}_2x_4$ und $x_1x_3x_4$.

Zur letzten Definition sind einige Anmerkungen zu machen:

1. Minterme zu einschlägigen Indizes einer Booleschen Funktion f sind Implikanten von f.
2. Ist M Implikant von f, und ist m ein Minterm von f derart, dass M eine Verkürzung von m ist, so gilt $m \leq M$, d. h. m ist Implikant von M.
3. Im Karnaugh-Diagramm entsprechen rechteckige Blöcke von Einsen den Implikanten und maximale derartige Blöcke den Primimplikanten.

Der folgende Satz zeigt nun, dass die im Karnaugh-Verfahren für $n = 3, 4$ verwirklichte Idee, das Vereinfachungsproblem auf die Bestimmung von Primimplikanten zurückzuführen, für disjunktive Formen tatsächlich Kostenminimalität garantiert:

Satz 3.1 Sei $f : B^n \to B$ eine Boolesche Funktion, $f \not\equiv 0$. Ist $d \equiv M_1 + \ldots + M_k$ eine Darstellung von f als disjunktive Form mit minimalen Kosten, so sind die $M_i, i = 1, \ldots, k$, Primimplikanten von f.

Beweis: Wir bemerken zunächst, dass alle $M_i, i = 1, \ldots, k$, Implikanten von f sind: Sei $x \in B^n$ beliebig und $M_i(x) = 1$. Dann folgt $f(x) = 1$, da f durch d als Disjunktion der M_i dargestellt ist. Angenommen, einer der Terme, etwa M_j, ist kein Primimplikant von f. Dann existiert nach Definition 3.4 eine echte Verkürzung V von M_j, welche ebenfalls Implikant von f ist, und es gilt $M_j \leq V$. Wir erhalten dann eine weitere Darstellung von f als disjunktive Form d', indem wir in d den Term M_j durch V ersetzen:

$$d' \equiv M_1 + \ldots + M_{j-1} + V + M_{j+1} + \ldots + M_k.$$

Die Kosten dieser Darstellung ergeben sich zu:

$$K(d') = (k-1) + \sum_{i=1}^{j-1} K(M_i) + K(V) + \sum_{i=j+1}^{k} K(M_i).$$

Da V *echte* Verkürzung von M_i ist, folgt $K(d') < K(d)$. Dies ist aber ein Widerspruch zu der Voraussetzung, dass die Kosten von d bereits minimal waren. Daraus folgt die Behauptung. $\triangledown$

Nach diesen Vorbereitungen können wir nun ein Vereinfachungsverfahren für Boolesche Funktionen skizzieren, welches zuerst 1952 von W. Quine angegeben und 1956 von E. McCluskey verbessert wurde. Es besteht aus den folgenden Schritten:

1. Bestimmung aller Primimplikanten;
2. Kosten-minimale Auswahl von Primimplikanten.

Wir beschreiben dieses Verfahren an folgendem Beispiel:

Beispiel 3.5 Sei $f : B^4 \to B$ gegeben durch die DNF- Darstellung

$$\begin{aligned} f \quad = \quad & \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2x_3\overline{x}_4 + x_1\overline{x}_2x_3x_4 \\ & + x_1x_2\overline{x}_3\overline{x}_4 + x_1x_2\overline{x}_3x_4 + x_1x_2x_3\overline{x}_4. \end{aligned}$$

□

Wir erläutern zunächst Schritt (1) des Verfahrens: Alle Primimplikanten lassen sich offensichtlich durch wiederholte Anwendung der aus Abschnitt 3.1 bekannten Resolutionsregel (bzw. für $n = 3, 4$ mit dem Karnaugh-Verfahren) ausgehend von Mintermen ermitteln. Dabei unterscheiden sich zwei Terme, auf welche die Regel anwendbar ist, in genau einer Variablen, die in dem einen negiert, in dem anderen nicht negiert vorkommt. Daher teile man die zu betrachtenden Minterme anhand der Anzahl der vorkommenden Negationszeichen in Gruppen ein wie in Tabelle 3.3

Tabelle 3.3: Minterme zu Beispiel 3.5 (gemäß Anzahl der Negationen).

Gruppe	Minterm	einschlägiger Index	Dezimaldarstellung des Index
1	$x_1\overline{x}_2x_3x_4$	1011	11
	$x_1x_2\overline{x}_3x_4$	1101	13
	$x_1x_2x_3\overline{x}_4$	1110	14
2	$\overline{x}_1x_2x_3\overline{x}_4$	0110	6
	$x_1x_2\overline{x}_3\overline{x}_4$	1100	12
3	$\overline{x}_1x_2\overline{x}_3\overline{x}_4$	0100	4
4	$\overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4$	0000	0

Tabelle 3.4: Tabelle 3.3 nach (erster) Anwendung der Resolutionsregel.

Gruppe	Implikant	Index	Minterm-Nummern
1	$x_1\overline{x}_2x_3x_4$	1011	11
	$x_2x_3\overline{x}_4$	*110	6, 14
	$x_1x_2\overline{x}_3$	110*	12, 13
	$x_1x_2\overline{x}_4$	11*0	12, 14
2	$\overline{x}_1x_2\overline{x}_4$	01*0	4, 6
	$x_2\overline{x}_3\overline{x}_4$	*100	4, 12
3	$\overline{x}_1\overline{x}_3\overline{x}_4$	0*00	0, 4

gezeigt. Alle Paare von Mintermen, auf welche die Resolutionsregel anwendbar ist, findet man damit durch die Betrachtung aller Mintermpaare aus benachbarten Gruppen. Alle dabei gewonnenen verkürzten Implikanten tragen wir in eine neue Tabelle ein, welche ebenfalls vier Spalten enthält, jedoch in der Indexspalte „heraus gefallene" Variablen durch * kennzeichnet und in der Nummernspalte die Nummern aller beteiligten einschlägigen Indizes enthält; ferner übernehmen wir in die neue Tabelle alle Implikanten, auf welche die Regel bereits nicht mehr anwendbar ist (vgl. Tabelle 3.4). Das sind genau diejenigen Implikanten, die nicht als Resolutionspartner benutzt worden sind; sie können auch später nicht mehr benutzt werden, da die noch aktiven Implikanten immer kürzer werden. Mit dieser neuen Tabelle wird das Verfahren iteriert, und zwar solange, bis eine gerade erzeugte Tabelle mit der zuletzt erzeugten übereinstimmt. Diese Tabelle enthält dann nur noch Primimplikanten, und zwar alle von ihnen (vgl. Tabelle 3.5).

Als nächstes erläutern wir Schritt (2) des Verfahrens: Den in Schritt (1) festgestellten Zusammenhang zwischen Primimplikanten und Mintermen halten wir nun in einer Matrix $\mathcal{A} = (a_{ij})$ fest, deren Zeilen Primimplikanten und deren Spalten Minterme repräsentieren wie folgt:

$$a_{ij} := \begin{cases} 1 & \text{falls Minterm } j \leq \text{Primimplikant } i \text{ (im Sinne von Definition 2.4)} \\ 0 & \text{sonst} \end{cases}$$

Tabelle 3.5: Alle Primimplikanten zu Beispiel 3.5.

Gruppe	Implikant	Index	Minterm-Nummern
1	$x_1\overline{x}_2x_3x_4$	1011	11
	$x_1x_2\overline{x}_3$	110*	12, 13
	$x_2\overline{x}_4$	*1*0	4, 6, 12, 14
3	$\overline{x}_1\overline{x}_3\overline{x}_4$	0*00	0, 4

Tabelle 3.6: Implikationsmatrix zu Beispiel 3.5.

Minterm / Primimplikant	0	4	6	11	12	13	14
$x_1\overline{x}_2x_3x_4$	0	0	0	1	0	0	0
$x_1x_2\overline{x}_3$	0	0	0	0	1	1	0
$x_2\overline{x}_4$	0	1	1	0	1	0	1
$\overline{x}_1\overline{x}_3\overline{x}_4$	1	1	0	0	0	0	0

a_{ij} hat also den Wert 1, falls der j-te Minterm an der Bildung des i-ten Primimplikanten beteiligt war (vgl. Tabelle 3.6). In dieser Matrix hat man nun noch eine Auswahl von Zeilen, d. h. Primimplikanten, so zu treffen, dass einerseits die von diesen Zeilen gebildete Teilmatrix in jeder Spalte mindestens eine Eins enthält, andererseits die Gesamtkosten dieser Primimplikanten minimal sind unter allen möglichen Auswahlen mit der ersten Eigenschaft.

Im laufenden Beispiel benötigt man alle vier Primimplikanten, um alle Minterme zu „überdecken": Die Unentbehrlichkeit des ersten Primimplikanten folgt z. B. daraus, dass in der Spalte des Minterms 11 keine andere Eins als die in der ersten Zeile vorhanden ist. Damit erhält man als kostengünstigste Darstellung d von f:

$$d \equiv x_1\overline{x}_2x_3x_4 + x_1x_2\overline{x}_3 + x_2\overline{x}_4 + \overline{x}_1\overline{x}_3\overline{x}_4$$

mit $K(d) = 11$.

Es sei darauf hingewiesen, dass Schritt (2) unter Umständen sehr hohen Aufwand erfordert, wenn man viele mögliche Auswahlen bzgl. ihrer Kosten miteinander vergleichen muss. Von einer optimalen algorithmischen Präzisierung des Schrittes (2) kann also im Quine-McCluskey-Verfahren noch nicht die Rede sein. Wir werden darauf und auf die allgemeine Komplexitätsproblematik des Überdeckungsphänomens in Kapitel 4 genauer eingehen.

3.2 Fehlerdiagnose von Schaltnetzen

Wir wollen nun einen zweiten Problemkreis behandeln, welcher auch eine Art Optimierung darstellt, nämlich die *Fehlerdiagnose von Schaltnetzen*. Dies ist für die Produktion von Schaltelementen von großer Bedeutung. Die heutige VLSI-Technologie stellt sehr große Schaltnetze, ja ganze CPUs und deren unmittelbare „Umgebung"

(vgl. Kapitel 8) auf einem Chip zur Verfügung. Zur Zeit lassen sich viele Millionen Bauteile auf einem typischerweise quadratischen Chip unterbringen; die sich in voller Entwicklung befindliche Nanotechnik wird noch wesentlich kompaktere Chips liefern. Bei dieser Größe kann man natürlich im Allgemeinen nicht mehr nach defekten Bauteilen oder gerissenen Verbindungsdrähten auf einem Chip suchen, sondern man muss sich oft darauf beschränken, Chips als Ganzes auf Defekte zu testen und gegebenenfalls auszusondern.

Wie kann man nun eine solche behavioristische Qualitätsprüfung bzw. Endkontrolle vornehmen? Betrachten wir z. B. — wie in diesem Kapitel schon mehrfach geschehen — ein Addiernetz für 16-stellige Dualzahlen: Eine Möglichkeit besteht sicher darin, alle möglichen Inputs anzulegen und die entsprechenden Ergebnisse zu verifizieren. In diesem Fall wären somit $2^{32} \approx 4 \cdot 10^9$ Argumente-Tupel („Tests") durch die Schaltung zu schicken, was offensichtlich ein zu hoher Aufwand ist. Wir werden nun eine klassische Methode vorstellen, nach welcher man aus der Menge aller möglichen Tests für ein gegebenes Schaltnetz diejenigen auswählen kann, mit der sich bereits alle Fehler einer bestimmten Art aufdecken lassen. Die Fehlerart wird dabei bestimmt durch eine vorher zu treffende *Fehlerannahme*, die z. B. wie folgt lauten kann:

(a) Es tritt im gegebenen Schaltnetz höchstens ein Fehler auf;

(b) der Defekt, welcher einen Fehler verursacht, ist ein gerissener Verbindungsdraht.

Teil (b) beschreibt einen sehr häufig auftretenden Defekt; neben diesem sind noch Kurzschlüsse, defekte Gatter durch fehlerhafte Halbleiter usw. denkbar. Dieser Fehler (b) wird auch als *0-Verklemmung* (engl. „stuck-at 0") bezeichnet (unter der Vorstellung, dass ein defekter Draht keinen Impuls übermitteln kann).

Unter dieser Ein-Fehler-Annahme (a) + (b) wollen wir nun für das in Abbildung 1.4 angegebene Schaltnetz der Booleschen Funktion aus Beispiel 1.13 einen so genannten *minimalen Testvektor* bestimmen. Dazu geben wir das Schaltnetz wieder als DAG an (vgl. Abbildung 3.9). In diesem Beispiel können 18 Drähte reißen, und wir verschaffen uns zunächst in Tabelle 3.7 eine Übersicht über die Auswirkungen eines defekten Drahtes auf nachfolgende Gatter bzw. Leitungen und insbesondere auf den Output; diesen bezeichnen wir mit f_i, falls Draht i gerissen ist. Aus $f(x_1, x_2, x_3) = \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 x_3 + x_1 x_2 x_3$ erhalten wir für die f_i folgende Darstellungen:

$$
\begin{aligned}
f_1 &= \overline{0} \cdot x_2 x_3 + x_1 \overline{x}_2 x_3 + x_1 x_2 x_3 = x_2 x_3 + x_1 x_3 \\
f_2 &= 0 \cdot x_2 x_3 + x_1 \overline{x}_2 x_3 + x_1 x_2 x_3 = x_1 \overline{x}_2 x_3 + x_1 x_2 x_3 = x_1 x_3 \\
f_3 &= \overline{x}_1 x_2 x_3 + x_1 x_2 x_3 = x_2 x_3 \\
f_4 &= \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 x_3 \\
f_5 &= x_1 \overline{x}_2 x_3 + x_1 x_2 x_3 = x_1 x_3 \\
f_6 &= \overline{x}_1 x_2 x_3 + x_1 x_3 \\
f_7 &= x_2 x_3 \\
f_8 &= \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 x_3 \\
f_9 &= x_1 x_3
\end{aligned}
$$

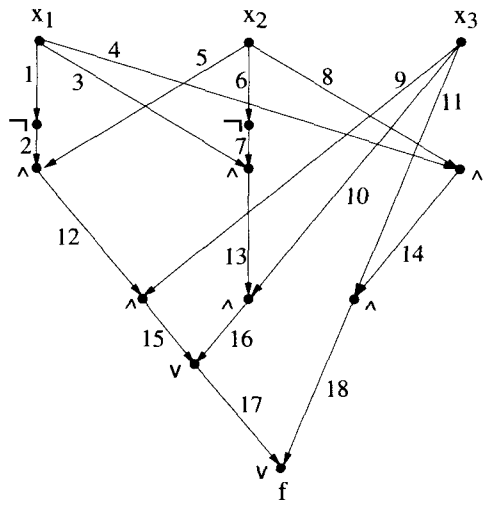

Abbildung 3.9: DAG zu Beispiel 1.13 mit „Draht-Nummern".

Tabelle 3.7: Fehlermöglichkeiten in Abbildung 3.9 (Ausfalltafel).

x_1	x_2	x_3	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	1	1	0	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1	1	0	1	1
1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	1	1	1	0	1

x_1	x_2	x_3	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	1	1	0	1	0	1
1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	1	0	1	1	0	0	1
1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	1	1	0	1	1	1	0

Tabelle 3.8: Ausfallmatrix zu Abbildung 3.9.

x_1	x_2	x_3	f_1	f_2	f_3	f_4	f_{17}
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	1	1	0	1	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	0
1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	1

$$
\begin{aligned}
f_{10} &= x_2x_3 \\
f_{11} &= \overline{x}_1x_2x_3 + x_1\overline{x}_2x_3 \\
f_{12} &= 0 \cdot x_3 + x_1\overline{x}_2x_3 + x_1x_2x_3 = x_1x_3 \\
f_{13} &= x_2x_3 \\
f_{14} &= \overline{x}_1x_2x_3 + x_1\overline{x}_2x_3 \\
f_{15} &= x_1x_3 \\
f_{16} &= x_2x_3 \\
f_{17} &= x_1x_2x_3 \\
f_{18} &= \overline{x}_1x_2x_3 + x_1\overline{x}_2x_3
\end{aligned}
$$

Tabelle 3.7 zeigt die so genannte *Ausfalltafel*, welche sich verkürzen lässt zu einer *Ausfallmatrix* durch Weglassen doppelter Spalten, denn offensichtlich gilt:

$$
\begin{array}{lllllllll}
f_1 & = & f_6 & & & & & & \\
f_2 & = & f_5 & = & f_9 & = & f_{12} & = & f_{15} \\
f_3 & = & f_7 & = & f_{10} & = & f_{13} & = & f_{16} \\
f_4 & = & f_8 & = & f_{11} & = & f_{14} & = & f_{18}
\end{array}
$$

Als Ausfallmatrix erhalten wir die in Tabelle 3.8 gezeigte Matrix. Diese Matrix zeigt für alle möglichen Eingaben die beobachtbaren globalen Abweichungen vom gewünschten Ausgabeverhalten, welche durch 0-Verklemmung entsprechend unserer Fehlerannahme erzeugbar sind. Da das Ziel ist, aus der Menge aller möglichen Inputs (hier $2^3 = 8$ Stück) diejenigen herauszufinden, mit deren Hilfe bereits alle Abweichungen vom geforderten Verhalten feststellbar sind, bietet sich zunächst der Übergang von der Ausfallmatrix zur so genannten *Fehlermatrix* an, die man erhält, indem man jede f_i-Spalte durch $f \leftrightarrow f_i$ ersetzt. Diese Matrix ist in Tabelle 3.9 gezeigt; jede Spalte zeigt gerade die Stellen (durch eine 1) an, an der das beobachtete Verhalten von f abweicht. Unsere Aufgabe besteht nun noch darin, möglichst wenig Tests (d. h. Input-Zeilen) so auszuwählen, dass jeder Fehler aufgedeckt wird. In diesem Beispiel gibt offensichtlich { 3, 5, 7 } einen minimalen Testvektor, d. h. allein durch probeweises Anlegen von (0, 1, 1), (1, 0, 1) und (1, 1, 1) an die Schaltung sind *alle* aus unserer Fehlerannahme resultierenden Defekte erkennbar. Die Anzahl der Tests ist somit von

Tabelle 3.9: Fehlermatrix zu Abbildung 3.9.

Zeilen-Nr.	x_1	x_2	x_3	$f \not\leftrightarrow f_1$	$f \not\leftrightarrow f_2$	$f \not\leftrightarrow f_3$	$f \not\leftrightarrow f_4$	$f \not\leftrightarrow f_{17}$
0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	0	1	1	0	1	0	0	1
4	1	0	0	0	0	0	0	0
5	1	0	1	0	0	1	0	1
6	1	1	0	0	0	0	0	0
7	1	1	1	0	0	0	1	0

8 auf 3 gefallen. (Man beachte, dass in diesem Beispiel der Defekt Nr. 1, welcher zur Funktion f_1 führt, nach außen hin nicht als Defekt erkennbar ist wegen $f_1 \equiv f$; liegt also dieser Defekt vor, leistet das Schaltnetz dennoch das Gewünschte.)

Diese exemplarisch vorgeführte Methode bezeichnet man auch als *schaltungsabhängige Fehlerdiagnose*; damit kann man feststellen, ob ein gegebenes Schaltnetz, dessen logischer Aufbau bekannt ist, mit einem bestimmten Fehler, welcher in einer Fehlerannahme spezifiziert wird, behaftet ist oder nicht. Die *Lokalisierung* eines Fehlers ist mit dieser Methode offensichtlich im Allgemeinen nicht möglich.

Natürlich gibt es bei großen Schaltungen ernst zu nehmende Fehlerquellen, die von ganz anderer Natur sind als die hier diskutierten Stuck-At-Annahmen bei höchstens einem Fehler. Hierzu gehören Defekte durch das Auftreten von Hasards, die im folgenden Abschnitt behandelt werden und die durch Analyse der zu realisierenden Funktionen im Prinzip vermeidbar wären. Auch andere physikalisch bedingte Fehler wie z. B. Probleme mit der Spannungsversorgung oder kapazitativ bedingte Überlagerungs-Störungen sind mit der Testmengen-Methodik nicht zu bewältigen.

Diese Probleme müssen deshalb bereits in der Phase des Chip-Entwurfs (Design) angegangen werden. Die physikalische Realität lässt sich aber in der Praxis nicht sauber vom reinen Logik-Entwurf trennen. Deshalb gehört die Entwicklung praktisch anwendbarer automatischer Testmethoden für den Entwurf und für die Fertigungs-Endkontrolle von Chips zu den wichtigsten Aufgaben der praktischen Informatik, die nur in interdisziplinärer Zusammenarbeit gelöst werden können.

3.3 Hasards in Schaltnetzen

Zum Abschluss dieses Kapitels wollen wir noch einen dritten Aspekt der Verbesserung von Schaltnetzen behandeln, welcher mit der technischen Funktionssicherheit derartiger Schaltungen zusammenhängt. Im letzten Abschnitt haben wir physikalische Probleme, die bei der Realisierung von Schaltnetzen auftreten können, mit unserer „Black-Box-Philosophie" in Verbindung gebracht. Für die Praxis des Aufbaus und der Verwendung von Schaltnetzen sind viele physikalische Tatsachen wesentlich. Wir wollen einige dieser Tatsachen formulieren und ihre Konsequenzen studieren:

1. Jedes Signal, welches ein Gatter durchläuft, hat eine zwar kurze, aber nicht vernachlässigbare Laufzeit;
2. Änderung von Input-Signalen, welche logisch gleichzeitig erfolgen, können im Allgemeinen physikalisch nicht gleichzeitig stattfinden.
3. Signal-Laufzeiten können für die einzelnen Gatter eines Schaltnetzes unterschiedlich groß sein (Spezifizierung von Annahme 1).

Eine Folge aus diesen Annahmen, welche für die Praxis tatsächlich realistisch sind, ist, dass das Umschalten von gewissen Input-Signalen für ein Schaltnetz auf andere Signalwerte einer Verzögerung unterliegt. Diese Verzögerung kann dazu führen, dass sich der Wert am Ausgang des Schaltnetzes kurzzeitig ändert, was aber unter Umständen unerwünscht ist.

Beispiel 3.6 Sei $f(x_1, x_2, x_3) = x_1x_3 + x_2\overline{x}_3$, so gilt:

$$\begin{aligned} f(1,1,0) &= 1 \\ f(1,1,1) &= 1 \\ f(1,0,0) &= 0 \\ f(1,0,1) &= 1 \end{aligned}$$

□

Sei nun S irgendein Schaltnetz für f, an dessen Eingängen das Input-Tupel (1, 1, 0) anliege. Soll dann auf den Input (1, 0, 1) umgeschaltet werden, so könnte dies gemäß Annahme (2) in zwei unterschiedlichen Abfolgen erfolgen: Entweder wird zunächst der Wert von x_2 von 1 in 0 und dann der Wert von x_3 von 0 in 1 geändert oder diese Umschaltung verläuft in umgekehrter Reihenfolge. Wird der zweite Weg beschritten, ändert sich das Ausgangssignal von S nicht, da auch für das „Zwischen-Tupel" der Funktionswert gleich 1 ist. Wird jedoch der erste Weg gewählt, „kippt " das Ausgangssignal von S kurzzeitig auf 0 (wegen $f(1,0,0) = 0$), bevor es dann wieder den „richtigen" Wert 1 annimmt. Dieses Verhalten ist unerwünscht z. B. dann, wenn das Ausgangssignal ein sehr empfindliches Gerät steuert, welches im schlimmsten Fall defekt wird, wenn das Signal auch nur kurze Zeit auf 0 kippt.

Phänomene dieser Art nennt man *Hasards* (engl.: hazard; Gefahr, Risiko). Wie bei der Fehlerdiagnose unterscheidet man zwei Arten von Hasards: (Schaltungsunabhängige) *Funktionshasards*, welche nur durch das „Übergangsverhalten" der gegebenen Booleschen Funktion (entsprechend Annahme (2) oben) bedingt sind, und (schaltungsabhängige) *Schaltungshasards*, welche sich als Folgerung aus Annahme (3) ergeben. (Darüber hinaus unterscheidet man bei beiden Arten noch *statische* und *dynamische* Hasards: Ein statischer Hasard bewirkt die unerwünschte Veränderung eines Output-Signals während des „Umkippens" von Inputsignalen; ein dynamischer Hasard liegt vor, wenn sich das Ausgangssignal für neue Inputs tatsächlich ändern soll, die Veränderung sich aber erst nach einem gewissen „Flimmern" endgültig einstellt. Wir befassen uns hier nicht weiter mit dynamischen Hasards.)

Die Boolesche Funktion aus Beispiel 3.6 besitzt also einen *Funktions*hasard. Wir wollen diesen Hasard-Begriff präzisieren:

Definition 3.5 Sei $f : B^n \to B$ eine Boolesche Funktion und seien $a_0 \in B^k$ $(1 \leq k < n)$, $a_1 \in B^{n-k}$, $a = (\{a_0, a_1\})$, $b = (\{a_0, \overline{a}_1\})$. (Die [unterbestimmte] Schreibweise $a = (\{a_0, a_1\})$ soll dabei andeuten, dass eine gewisse Auswahl der Komponenten von a den Vektor a_0 bildet und die dabei nicht berücksichtigten Komponenten den Vektor a_1 bilden. In Spezialfällen[1] kann also a_0 den Anfang und a_1 das Ende des Vektors a bedeuten.) f besitzt einen (statischen) *Funktionshasard* (für den Input-Wechsel von a nach b), falls gilt:

(i) $f(a) = f(b)$;

(ii) es gibt ein $a'_1 \in B^{n-k}$ so, dass für $c = (\{a_0, a'_1\})$ gilt: $f(a) \neq f(c)$.

Beispiel 3.6 (Fortsetzung): Setze für $k = 1$

$$\begin{aligned} a_0 &= (x_1) = (1) \\ a_1 &= (x_2, x_3) = (1, 0) \\ a'_1 &= (x'_2, x'_3) = (0, 0) \end{aligned}$$

so gilt:

$$\begin{aligned} f(a) &= f(a_0, a_1) = f(1, 1, 0) = 1, \\ f(c) &= f(a_0, a'_1) = f(1, 0, 0) = 0, \\ f(b) &= f(a_0, \overline{a}_1) = f(1, 0, 1) = 1. \end{aligned}$$

Also besitzt f einen Funktionshasard für den Inputwechsel von (1, 1, 0) nach (1, 0, 1). □

Beispiel 3.7 (vgl. Abschnitt 3.1) Sei $f : B^4 \to B$ durch das in Abbildung 3.10 gezeigte Karnaugh-Diagramm gegeben. Zur Bestimmung der Funktionshasards von f (gemäß Definition 3.5) stellen wir eine Tabelle auf, welche für $k = 1, 2, 3$ in Abhängigkeit von a_0 die a_1-Tupel angibt, für die ein Hasard vorliegt. Die Tabelleneinträge werden dabei durch Betrachtung des Karnaugh-Diagramms wie folgt ermittelt:

$k = 1$: Wählt man zunächst $a_0 = (0)$, so ist nur die „linke" Hälfte des Karnaugh-Diagramms zu betrachten. Die Stellen im Diagramm, an denen komplementäre a_1-Tupel liegen, sind in Abbildung 3.11 durch Linien verbunden. Wegen Bedingung (i) von Definition 3.5 reicht es nun, solche durch eine Linie verbundenen Paare von Punkten zu betrachten, an denen jeweils der gleiche Funktionswert steht. Annahme (2) oben besagt dann, dass das Umschalten von einem Input-Tupel auf ein anderes nicht entlang dieser Linie, sondern nur über horizontale oder vertikale Nachbar-Felder (eventuell zyklische Nachbarn!) erfolgen kann, also z. B. für den Wechsel von 0001 auf 0110 wie in Abbildung 3.12 gezeigt. Hat eines dieser „Zwischenfelder" einen anderen Funktionswert als Start und Ziel, so liegt (für diesen Input-Wechsel) ein Funktionshasard vor.

Für die Wahl $a_0 = (1)$ betrachtet man analog die rechte Hälfte des Karnaugh-Diagramms. Der erste Teil der Tabelle lautet damit wie in Tabelle 3.10 angegeben.

[1] Im Sinne einer Vereinfachung der Schreibweise werden wir künftig nur diesen Spezialfall betrachten. Unter dieser Annahme entfallen dann die geschweiften Klammern.

x_3x_4 \ x_1x_2	00	01	11	10
00	0	1	0	0
01	0	1	1	1
11	1	1	1	0
10	0	0	1	0

Abbildung 3.10: Karnaugh-Diagramm zu Beispiel 3.7.

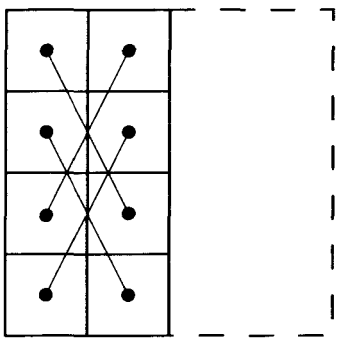

Abbildung 3.11: Bestimmung der Funktionshasards in Beispiel 3.7 (1).

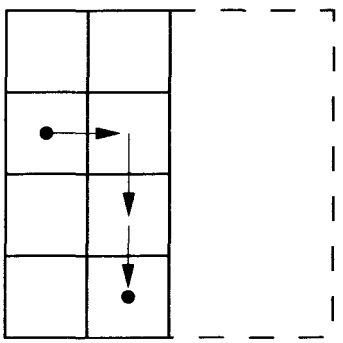

Abbildung 3.12: Bestimmung der Funktionshasards in Beispiel 3.7 (2).

Tabelle 3.10: Zu Beispiel 3.7: $k = 1$.

k	a_0	a_1	$f(a_0, a_1) = f(a_0, \overline{a}_1)$	mögl. a_1'	$f(a_0, a_1')$
1	(0)	(0, 0, 1)	0	(1, 0, 1)	1
		(1, 0, 0)	1	(0, 0, 1)	0
1	(1)	(1, 0, 0)	0	(1, 0, 1)	1
		(0, 0, 1)	1	(0, 1, 1)	0

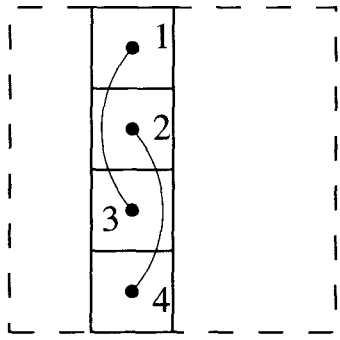

Abbildung 3.13: Bestimmung der Funktionshasards in Beispiel 3.7 (3).

$k = 2$: Für a_0 hat man nun vier Wahlmöglichkeiten, welche jeweils einer Spalte des Karnaugh-Diagramms entsprechen. Komplementäre a_1-Tupel liegen dann an den in Abbildung 3.13 exemplarisch für Spalte 2 gezeigten Stellen. Da ein Input-Wechsel z. B. von 1 nach 3 nur über 2 erfolgen kann, ist das Auffinden der Hasards gegenüber Fall $k = 1$ bereits wesentlich leichter: Ist der Funktionswert an den Stellen 1 und 3 identisch, und weicht er an der Stelle 2 von diesem Wert ab, so liegt offensichtlich ein Funktionshasard vor. Die Fortsetzung von Tabelle 3.10 lautet damit wie in Tabelle 3.11 angegeben.

$k = 3$: Nun ergeben sich acht Wahlmöglichkeiten für a_0, welche jeweils der ersten oder zweiten Hälfte einer Spalte des Karnaugh-Diagramms entsprechen. Da man also nur noch benachbarte Zeilen zu betrachten hat, so dass Wahlmöglichkeiten für a_1' entfallen, ist unmittelbar einzusehen, dass bereits alle Funktionshasards von

Tabelle 3.11: Zu Beispiel 3.7: $k = 2$.

k	a_0	a_1	$f(a_0, a_1) = f(a_0, \overline{a}_1)$	mögl. a_1'	$f(a_0, a_1')$
2	(0 , 0)	(0 , 1)	0	(1 , 1)	1
2	(0 , 1)	(0 , 0)	1	(1 , 0)	0
2	(1 , 1)	(0 , 1)	1	(0 , 0)	0
2	(1 , 0)	(0 , 0)	0	(0 , 1)	1

f gefunden sind.

Zusammenfassend haben wir für f bei folgenden Input-Wechseln einen Funktionshasard gefunden (selbstverständlich liegen auch jeweils in der umgekehrten Richtung Funktionshasards vor):

$$\begin{array}{lll} (1)\ 0001 & \rightarrow & 0110 \\ (2)\ 0100 & \rightarrow & 0011 \\ (3)\ 1100 & \rightarrow & 1011 \\ (4)\ 1001 & \rightarrow & 1110 \\ (5)\ 0001 & \rightarrow & 0010 \\ (6)\ 0100 & \rightarrow & 0111 \\ (7)\ 1101 & \rightarrow & 1110 \\ (8)\ 1000 & \rightarrow & 1011 \end{array}$$

Hierzu kommen unter Umständen weitere, welche sich ergeben, wenn man sich nicht nur auf den hier beschriebenen Spezialfall (vgl. letzte Fußnote) beschränkt. □

Es ist klar, dass man sich bei Funktionshasards im Allgemeinen auf deren *Erkennung* beschränken muss, da sie von der gegebenen Booleschen Funktion abhängen, nicht aber von einem speziellen Schaltnetz. Systematischere Erkennungsalgorithmen als das in Beispiel 3.7 exerzierte Nachrechnen der Definition werden im Rahmen der *Schaltkreistheorie* behandelt. Vermeiden lassen sich Funktionshasards nur durch eine Abänderung der gegebenen Booleschen Funktion, jedoch ist es möglich, durch künstliche *Synchronisation* die in dieser Art Hasard liegende Gefahr zu überspielen; man hat dann dafür zu sorgen, dass das Umkippen einzelner Input-Bits beim Übergang von einem auf einen anderen Input in einer Reihenfolge geschieht, welche das Ausgangs-Signal unverändert lässt (falls eine solche existiert). In Beispiel 3.6 würde dies z. B. durch die Reihenfolge „zuerst x_3, dann x_2" beim Wechsel von (1, 1, 0) auf (1, 0, 1) gewährleistet.

Die nun zu betrachtenden *Schaltungs*hasards sind — wie gesagt — eine Folge aus obiger Annahme (3): Verschiedene Signalwege können unterschiedliche Signallaufzeiten bewirken.

Beispiel 3.6 (Fortsetzung) Wir betrachten die in Abbildung 3.14 gezeigte Schaltung für $f = x_1x_3 + x_2\overline{x}_3$ und einen Input-Wechsel von (1, 1, 0) nach (1, 1, 1). Dann gilt: $f(1,1,0) = f(1,1,1) = 1$, und es liegt *kein* Funktionshasard vor, da es bei diesem Übergang kein „Zwischentupel" gibt. Dennoch kann hier Unerwünschtes passieren: Beim Umschalten von x_3 von 0 auf 1 sind die beiden Signalwege ACE und BDE zu durchlaufen. Unterstellen wir nun (gemäß Annahme 3), dass z. B. der Weg ACE „langsamer" ist als BDE, so kann folgende Situation eintreten: Der linke Eingang von Gatter E steht *noch* auf 0, während der rechte *schon* auf 0 steht (als Ausgang von Gatter D, das den neuen Input $x_3 = 1$ bereits verarbeitet hat). Am Ausgang von E erscheint dann der Wert 0, d. h. wieder liegt eine kurzzeitige Fehlfunktion vor, welche aber nicht durch einen Funktionshasard verursacht wird. Wir fassen dies wie folgt zusammen:

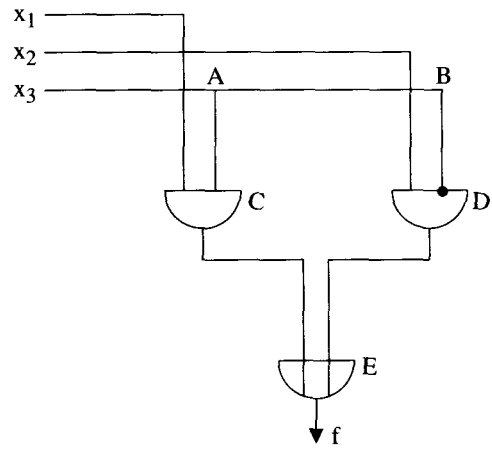

Abbildung 3.14: Schaltung zu Beispiel 3.6.

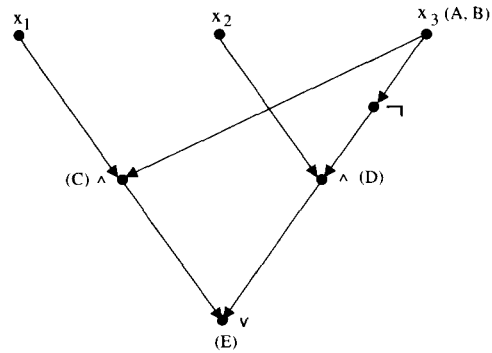

Abbildung 3.15: DAG zu Beispiel 3.6.

Definition 3.6 Sei $f : B^n \to B$ eine Boolesche Funktion, S ein Schaltnetz, welches f realisiert, und $a, b \in B^n$. S besitzt einen (statischen) *Schaltungshasard* (logischen Hasard) für den Input-Wechsel von a nach b, falls gilt:

(i) $f(a) = f(b)$:

(ii) f besitzt keinen Funktionshasard für den Wechsel von a nach b:

(iii) während des Wechsels von a nach b ist am Ausgang von S eine vorübergehende Fehlfunktion beobachtbar.

Im in Abbildung 3.14 angegebenen Schaltnetz liegt der Grund für den auftretenden Schaltungshasard offenbar in dem Fan-Out von x_3 (an den Stellen A und B) und der anschließenden Re-Konvergenz des *kritischen* x_3-Signals an der Stelle E: aus der Darstellung dieses Netzes als DAG (gemäß Definition 1.7) ist sofort ersichtlich, dass es *zwei* Wege vom Input x_3 zum Output E gibt (vgl. Abbildung 3.15).

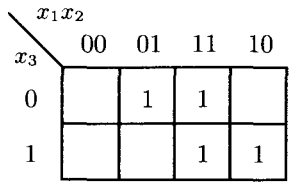

Abbildung 3.16: Karnaugh-Diagramm zu Beispiel 3.6.

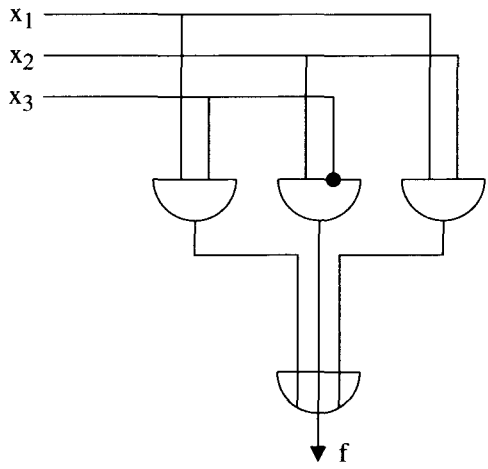

Abbildung 3.17: Erweiterung der Schaltung aus Abbildung 3.14.

Methoden zur *Erkennung* von Schaltungshasards werden ebenfalls im Rahmen der Schaltkreistheorie behandelt. Wir beschränken uns hier darauf, eine Methode zur *Beseitigung* von Schaltungshasards anzugeben. Der in obigem Schaltnetz auftretende logische Hasard ist offensichtlich durch zusätzliche Hardware („Gegenschaltung") eliminierbar wie folgt:

Betrachtet man das in Abbildung 3.16 gezeigte Karnaugh-Diagramm von f, so erkennt man, dass f die drei Primimplikanten x_1x_3, $x_2\bar{x}_3$ und x_1x_2 besitzt. Von diesen reichen zur Darstellung von f die ersten beiden aus, was zu der in Abbildung 3.14 gezeigten Schaltung führte. Schaltet man nun den dritten Primimplikanten hinzu, so erkennt man unmittelbar, dass der Schaltungshasard beim Input-Wechsel von (1, 1, 0) nach (1, 1, 1) eliminiert ist, da der Summand x_1x_2 jetzt ein konstantes Output-Signal garantiert. Dies ist eine zweistufige Schaltung, welche *alle* Primimplikanten von f enthält. Diese Beobachtung lässt sich verallgemeinern zu einem *hinreichenden* Kriterium für die Vermeidung von Schaltshasards:

Satz 3.2 *(Eichelberger 1965)* Ein zweistufiges Schaltnetz S für eine Boolesche Funktion f in disjunktiver Form ist frei von (statischen) Schaltungshasards, wenn die

Und-Gatter von S in einer 1-1-Korrespondenz zu den Primimplikanten von f stehen, d. h. jedes Und-Gatter von S realisiert einen Primimplikanten von f, und jedem Primimplikanten von f entspricht ein Und-Gatter in S.

Beweis: O. B. d. A. beweisen wir den Satz nur für den Fall, dass *eine* Variable, etwa x_n, ihren Wert ändert, und das Tupel $(x_1, \ldots, x_{n-1})$ unverändert seinen Wert $(a_1, \ldots, a_{n-1})$ behält. Nehmen wir ferner an, dass f keinen Funktionshasard für den Input-Wechsel von $(a_1, \ldots, a_{n-1}, 0)$ nach $(a_1, \ldots, a_{n-1}, 1)$ (oder umgekehrt) besitzt und dass $f(a_1, \ldots, a_{n-1}, 0) = f(a_1, \ldots, a_{n-1}, 1)$ gilt, so bleibt gemäß Definition 3.6 zu zeigen: Während des Wechsels von $(a_1, \ldots, a_{n-1}, 0)$ nach $(a_1, \ldots, a_{n-1}, 1)$ ist (unter der oben angegebenen Voraussetzung) am Ausgang von S *keine* vorübergehende Fehlfunktion zu beobachten. Wir unterscheiden zwei Fälle:

(a) $f(a_1, \ldots, a_{n-1}, 0) = f(a_1, \ldots, a_{n-1}, 1) = 1$: Dann sind offensichtlich

$$x_1^{a_1} \cdot \ldots \cdot x_{n-1}^{\alpha_{n-1}} x_n$$

und

$$x_1^{\alpha_1} \cdot \ldots \cdot x_{n-1}^{\alpha_{n-1}} \overline{x}_n$$

einschlägige Minterme von f. Aus diesen erhält man durch Anwendung der Resolutionsregel den Implikanten $x_1^{\alpha_1} \cdot \ldots \cdot x_{n-1}^{\alpha_{n-1}}$. Ist dieser bereits Primimplikant, ist nichts mehr zu zeigen. Anderenfalls existiert ein Primimplikant von f, welcher echte Verkürzung dieses Implikanten ist. In beiden Fällen gibt es also einen Primimplikanten von f, der x_n *nicht* enthält, und dem nach Voraussetzung ein Und-Gatter in S entspricht. Dieses Gatter stellt dann sicher, dass der Wechsel des Input-Signals x_n am Ausgang von S *nicht* zu beobachten ist.

(b) $f(a_1, \ldots, a_{n-1}, 0) = f(a_1, \ldots, a_{n-1}, 1) = 0$: Sei dann M ein beliebiger Primimplikant von f, so gilt wegen $M \leq f$ (vgl. Definition 2.4):

$$M(a_1, \ldots, a_{n-1}, 0) = M(a_1, \ldots, a_{n-1}, 1) = 0$$

Das bedeutet aber, dass dasjenige Und-Gatter des Schaltnetzes, welches M entspricht, seinen Output nicht ändert, wenn x_n seinen Wert wechselt. Da M beliebig gewählt war, gilt dies *für alle* Primimplikanten von f. Da also alle Und-Gatter nicht auf den x_n-Wechsel reagieren, ist diese Input-Änderung auch nicht am Ausgang von S zu beobachten.

Für den Fall, dass mehrere Variablen (gleichzeitig) ihren Wert ändern, verläuft die Argumentation völlig analog (vgl. Aufgabe 3.9). $\triangledown$

Der Satz von Eichelberger liefert also eine Möglichkeit, Schaltungen zu entwerfen (durch erhöhten Hardwareaufwand), welche frei sind von Schaltungshasards. Er zeigt einmal mehr die Bedeutung der Primimplikanten einer Booleschen Funktion und von Algorithmen, diese zu bestimmen. Allerdings sei angemerkt, dass die in diesem Satz angegebene Bedingung *nicht notwendig* für die Vermeidung von Schaltungshasards ist.

Es sei in diesem Zusammenhang auf eine für die Praxis ebenfalls denkbare Methode zur de facto-Vermeidung von Schaltungshasards hingewiesen: Durch Verlängerung von Verbindungsdrähten können gegebenenfalls unterschiedliche Signallaufzeiten

ausgeglichen werden, und zwar mit einer solchen Genauigkeit, dass der Hasard am Ausgang nicht mehr erkennbar ist.

3.4 Übungen

Hinweis: Zu den mit * gekennzeichneten Übungen sind im Internet Lösungen erhältlich.

3.1 Die Funktionen $f_i : B^4 \to B, i = 1, 2$ seien wie folgt definiert:

$$f_1(x_1, x_2, x_3, x_4) := \begin{cases} 1 & \text{falls } x_1x_2x_3x_4 \text{ ein Codewort des Gray-Codes} \\ 0 & \text{sonst} \end{cases}$$

$$f_2(x_1, x_2, x_3, x_4) := \begin{cases} 1 & \text{falls } x_1x_2x_3x_4 \text{ echter Teiler von 1101001 ist} \\ 0 & \text{sonst} \end{cases}$$

(a) Man entwerfe unter Verwendung des Karnaugh-Verfahrens möglichst einfache Schaltnetze zur Berechnung der f_i.

(b) Man vereinfache die unter (a) gewonnene Schaltung für f_2 durch Ausnutzung der Don't-Care-Fälle unter der Voraussetzung, dass f_2 nur für $0000 \leq (x_1x_2x_3x_4)_2 \leq 1001$ definiert sei.

3.2 Im Karnaugh-Diagramm für vierstellige Boolesche Funktionen werden jeweils 2^k ($k \in \{0, \ldots, 4\}$) Einsen — falls möglich — zu einem Block zusammengefasst.

(a) Wie viele verschiedene solcher Blöcke gibt es in Abhängigkeit von k?

(b) Sei $f : B^4 \to B$ und K_f ein zugehöriges Karnaugh-Diagramm. Zwei Blöcke in K_f heißen *unabhängig*, wenn sie nicht beide zusammen durch einen Block überdeckt werden können. Man zeige, dass jede Eins in höchstens sechs paarweise unabhängigen Blöcken liegen kann.

(c) Man gebe eine Funktion f an, für welche es tatsächlich eine von sechs paarweise unabhängigen Blöcken überdeckte Eins gibt.

*3.3 Die Funktion $f : B^5 \to B$ habe genau die folgenden einschlägigen Indizes: 1, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 16, 18, 19, 20, 23, 26, 27, 31. Mit Hilfe des Verfahrens von Quine und McCluskey bestimme man eine Kosten-minimale disjunktive Darstellung für f.

*3.4 Herr Maier liebt dass Wetten. Er hat von einem undurchsichtigen Anbieter zwei Angebote bekommen, leider aber keine Ahnung von den Inhalten dieser Wetten: Formel 1 und Rechnen. Die Angebote lassen sich wie folgt beschreiben:

$$f_1(x_1, x_2, x_3, x_4) := 1$$

falls *genau eine* der folgenden Bedingungen eintritt:

1. Michael S. (x_1) kommt ins Ziel, Ralf S. (x_2) aber nicht.