

1. Boolesche Räume

Jede Belegung von k Variablen kann als Binärvektor aus k Binärwerten x_1, \dots, x_k mit $x_i \in \{0,1\}$; $i = 1 \dots k$ dargestellt werden. Der Boolesche Raum B^k umfaßt alle k -stelligen Binärvektoren (Abb. 1.1). Jeder Binärvektor entspricht einem Punkt des Raumes. Es gibt 2^k k -stellige Binärvektoren und somit 2^k Raumpunkte.

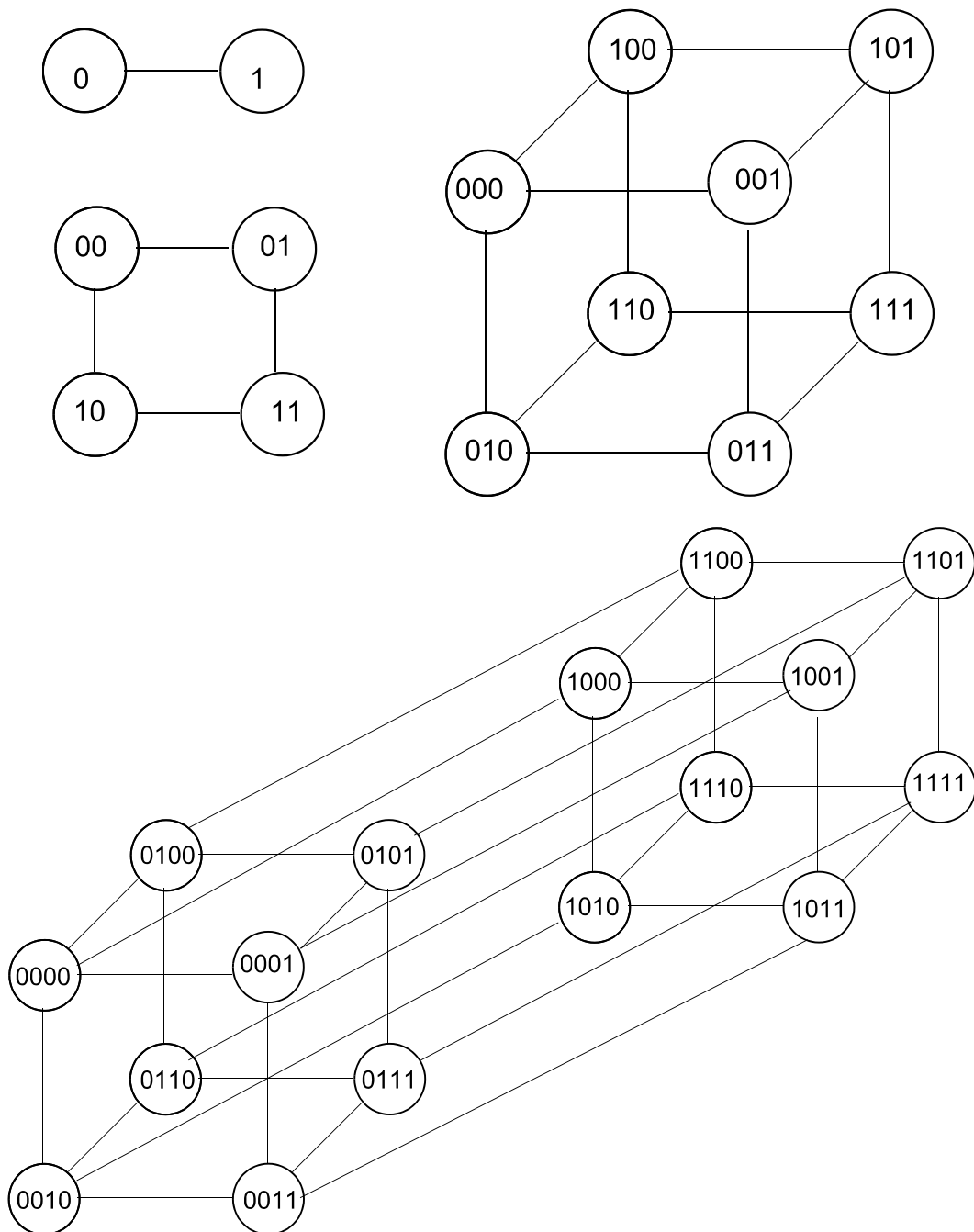


Abb. 1.1 Boolesche Räume mit 1, 2 und 3 Dimensionen. Die Kanten verbinden Raumpunkte, die voneinander den Abstand 1 haben

Der Boolescher Raum B^k kann als metrischer Raum aufgefaßt werden (Abb. 1.2). Zwei Punkte des Raums haben voneinander einen Abstand n , wenn sich die zugehörigen Binärvektoren in n Variablenpositionen voneinander unterscheiden (Hamming-Metrik). Abstand $d =$ Quersumme der bitweisen Antivalenzverknüpfung beider Binärvektoren x, y :

$$d = \sum (x_i \oplus y_i)$$

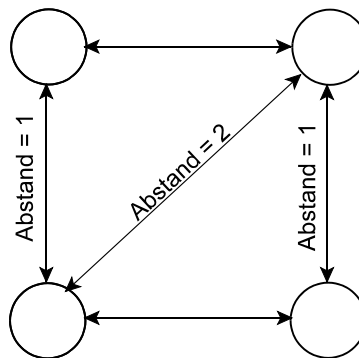


Abb. 1.2 Abstände im Booleschen Raum

2. Die Lösungsmenge

Eine Boolesche Funktion von k Variablen weist jedem der Punkte des Booleschen Raums B^k einen der beiden Binärwerte zu. Die Menge aller Punkte, denen der Funktionswert 1 zugeordnet wird, heißt On-Set, die Menge aller Punkte, denen der Funktionswert 0 zugeordnet wird, heißt sinngemäß Off-Set. Manchmal läßt sich zudem eine Menge von Punkten angeben, die für das betreffende Problem keine Bedeutung hat und somit außer Betracht bleiben kann (Don't-care-Set).

Boolesche Funktionen kommen typischerweise in Booleschen Gleichungen vor: $y = f(x_1 \dots x_k)$.

Die Menge aller Punkte des Booleschen Raums B^k , die Lösungen der Booleschen Gleichung sind, heißt Lösungsmenge.

Alle Booleschen Gleichungen können in zwei einfache Standardformen überführt werden:

$$f(x_1 \dots x_k) = 0 \text{ bzw. } f(x_1 \dots x_k) = 1.$$

Die Lösungsmenge einer Booleschen Gleichung $f(x_1 \dots x_k) = 0$ entspricht dem Off-Set.

Die Lösungsmenge einer Booleschen Gleichung $f(x_1 \dots x_k) = 1$ entspricht dem On-Set (Abb. 2.1).

Viele Boolesche Gleichungen werden von vornherein so aufgestellt, daß sich der Funktionswert 1 ergibt: $f(x_1 \dots x_k) = 1$.

Gleichungen der Form $f(x_1 \dots x_k) = 0$ können auf einfache Weise umgestellt werden:

$$f(x_1 \dots x_k) \oplus 1 = 1$$

Deshalb werden typischerweise On-Set und Lösungsmenge als Synonyme gebraucht.

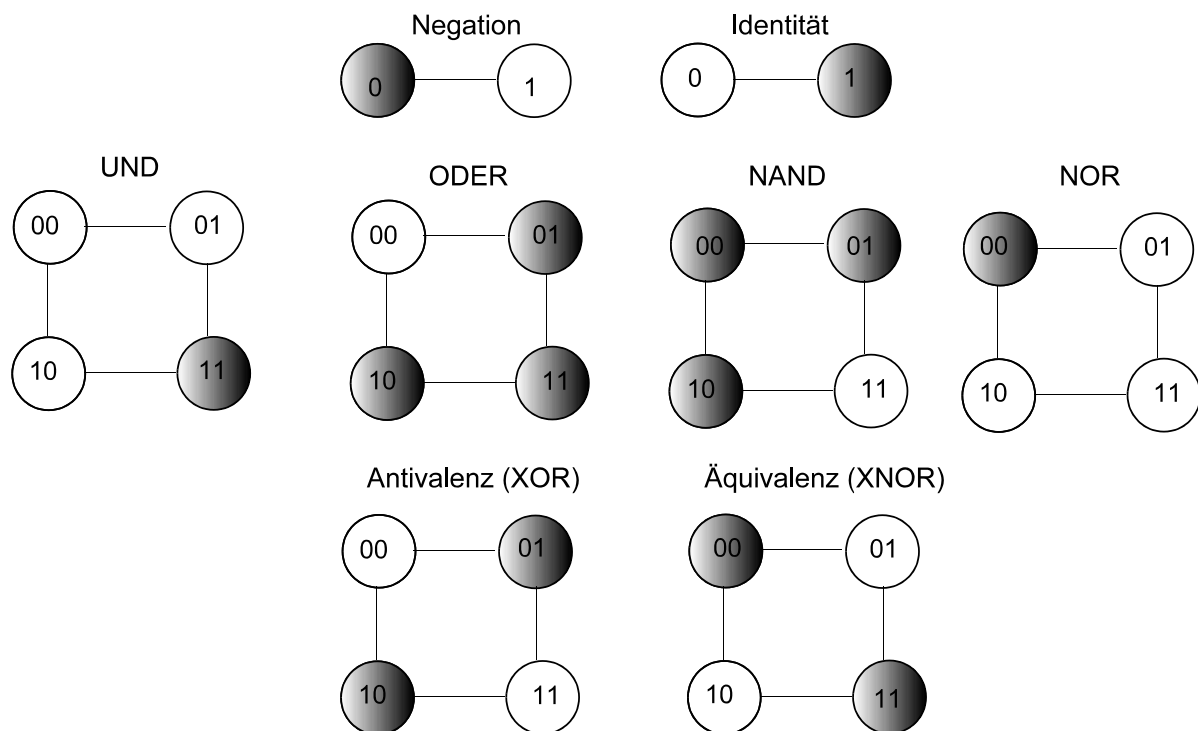


Abb. 2.1 Lösungsmengen (On-Sets) einiger elementarer Boolescher Funktionen

Elementare Boolesche Funktionen und Lösungsmengen

Den elementaren Booleschen Funktionen entsprechen isomorphe Operationen über Lösungsmengen:

- UND $\hat{=}$ Durchschnitt ($\wedge \hat{=} \cap$),
- ODER $\hat{=}$ Vereinigung ($\vee \hat{=} \cup$),
- Antivalenz $\hat{=}$ symmetrische Differenz ($\oplus \hat{=} \Delta$),
- Negation $\hat{=}$ Komplement.

Die Mächtigkeit der Lösungsmenge

Die Lösungsmenge einer Booleschen Funktion mit n Variablen kann praktisch immer auf höchstens 2^{n-1} Belegungen gebracht werden. Ist die Anzahl der Belegungen, die zur Lösungsmenge gehören, $> 2^{n-1}$, so liegt es nahe, die *negierte* Funktion zu untersuchen.

3. Rechenregeln der Schaltalgebra

Verwendung von Klammern

1. Assoziativität: bei gleichartigen Verknüpfungen ist die Klammeranordnung bzw. die Reihenfolge der Ausführung gleichgültig:

$$a \text{ @ } (b \text{ @ } c) = a \text{ @ } b \text{ @ } c = (a \text{ @ } b) \text{ @ } c. \text{ In Kurzform: } a(bc) = abc = (ab)c$$

$$a \vee (b \vee c) = a \vee b \vee c = (a \vee b) \vee c$$

2. Kommutativität: bei gleichartigen Verknüpfungen ist die Reihenfolge der Variablen gleichgültig:

$$a \text{ @ } b = b \text{ @ } a$$

$$a \vee b = b \vee a$$

3. Distributivität: eine Funktion vor einer Klammer ist auf alle Variablen in der Klammer anzuwenden; die jeweiligen Ergebnisse sind gemäß der eingeklammerten Funktion zu verknüpfen (vgl. das schulmäßige Ausmultiplizieren von Klammerausdrücken, beispielsweise $a(b + c) = ab + ac$):

$$a \cdot (b \vee c) = (a \cdot b) \vee (a \cdot c). \text{ In Kurzform: } a(b \vee c) = ab \vee ac$$

$$a \vee (b \cdot c) = (a \vee b) \cdot (a \vee c). \text{ In Kurzform: } a \vee (bc) = (a \vee b)(a \vee c)$$

Allgemein (wobei die Symbole $*$, $\#$ wahlweise für UND, ODER, Antivalenz bzw. Äquivalenz ($\cdot, \vee, \oplus, \equiv$) stehen):

- für gleichartige Verknüpfungen:
 - $a * (b * c) = a * b * c = (a * b) * c$ (Assoziativität),
 - $a * b = b * a$ (Kommutativität).
- für verschiedenartige Verknüpfungen: $a * (b \# c) = (a * b) \# (a * c)$ (Distributivität).

Hinweis: Antivalenz und Äquivalenz sind *nicht* distributiv!

Vorrang der Konjunktion

Bei Schreibweise ohne Konjunktionssymbol hat die Konjunktion Vorrang, sofern nicht durch Klammern eine andere Reihenfolge zum Ausdruck gebracht wird:

$$abc \vee def = (a \cdot b \cdot c) \vee (d \cdot e \cdot f)$$

Negation durch Überstreichen

Eine lückenlose Überstreichung mehrerer Symbole entspricht einer Einklammerung:

$$\overline{a \vee b \vee c} = \neg(a \vee b \vee c)$$

Doppelte Negation

$\overline{\overline{a}} = a$ (Negation der Negation = Bejahung (Identität))

Verknüpfungen mit sich selbst (Idempotenz)

$$a \cdot a = a; a \vee a = a$$

Verknüpfungen mit Festwerten

$$\overline{0} = 1; \overline{1} = 0$$

$$0 \cdot 0 = 0; 0 \cdot 1 = 0; 1 \cdot 0 = 0; 1 \cdot 1 = 1; 0 \cdot a = 0; 1 \cdot a = a$$

$$0 \vee 0 = 0; 0 \vee 1 = 1; 1 \vee 0 = 1; 1 \vee 1 = 1; 0 \vee a = a; 1 \vee a = 1$$

$$0 \oplus 0 = 0; 0 \oplus 1 = 1; 1 \oplus 0 = 1; 1 \oplus 1 = 0; 0 \oplus a = a; 1 \oplus a = \overline{a}$$

$$0 \equiv 0 = 1; 0 \equiv 1 = 0; 1 \equiv 0 = 0; 1 \equiv 1 = 1; \mathbf{0} \equiv a = \overline{a}; 1 \equiv a = a$$

Reduktion auf Festwerte

$$a \cdot \overline{a} = 0; a \vee \overline{a} = 1; a \oplus \overline{a} = 1; a \equiv \overline{a} = 0; a \oplus a = 0; a \equiv a = 1$$

$$a \oplus a \oplus a = 1; a \oplus a \oplus a \oplus a = 0 \text{ (Variablenanzahl ungerade: 1, gerade: 0)}$$

$$a \equiv a \equiv a = 0; a \equiv a \equiv a \equiv a = 1 \text{ (Variablenanzahl ungerade: 0, gerade: 1)}$$

Antivalenz-Rechenregeln

$$\overline{a \oplus b} = a \oplus \overline{b} = \overline{a \oplus \overline{b}} = a \equiv b$$

$$\overline{a \oplus \overline{b}} = a \oplus b$$

UND mit Äquivalenz:

$$a \cdot b = a \equiv b \equiv (a \vee b)$$

ODER mit Antivalenz:

$$a \vee b = a \oplus b \oplus ab$$

Einsetzungsregel

Anstelle einer Variablen in einer Schaltfunktion kann eine beliebige Schaltfunktion eingesetzt (substituiert) werden:

$$f_1(a, b, c, \dots) = f_1(a, f_2, c, \dots)$$

Hier wurde anstelle der Variablen b die Schaltfunktion f_2 eingesetzt. f_2 kann sowohl von Variablen abhängen, die auch in f_1 vorkommen, als auch von weiteren Variablen.

Eine Schaltfunktion in einer Schaltfunktion heißt auch Teilschaltfunktion. Man kann Teilschaltfunktionen und einzelne Variable wechselseitig einsetzen.

Ersetzungsregel

Jede Teilschaltfunktion f_i in einer Schaltfunktion f kann durch eine äquivalente Teilschaltfunktion f_i^* ersetzt werden. Zwei Schaltfunktionen f_i und f_i^* sind äquivalent, wenn sie die gleiche Wahrheitstabelle haben.

$$f(a, b, f_1, f_2, f_3) = f(a, b, f_1^*, f_2^*, f_3^*); \text{ wenn gilt } f_i = f_i^*$$

DeMorgansche Regeln

Diese beschreiben die Dualität zwischen UND und ODER. Sie gelten für einzelne Variable sowie für Schaltfunktionen gleichermaßen (Einsetzungs- und Ersetzungsregel):

1. *DeMorgansche Regel*: negiertes UND = ODER der einzeln negierten Variablen.

$$\overline{f_1 f_2} = \overline{f_1} \vee \overline{f_2}; \quad \overline{\overline{f_1} \vee \overline{f_2}} = f_1 f_2$$

2. *DeMorgansche Regel*: negiertes ODER = UND der einzeln negierten Variablen.

$$\overline{f_1 \vee f_2} = \overline{f_1} \overline{f_2}; \quad \overline{\overline{f_1} \overline{f_2}} = f_1 \vee f_2$$

Kürzungsregeln

1. Kürzungsregel: $f_1 \vee f_1 f_2 = f_1$

$$f_1 \cdot (f_1 \vee f_2) = f_1$$

2. Kürzungsregel: $f_1 f_2 \vee f_1 \overline{f_2} = f_1$

$$(f_1 \vee f_2) \cdot (f_1 \vee \overline{f_2}) = f_1$$

3. Kürzungsregel: $f_1 \vee \overline{f_1} f_2 = f_1 \vee f_2$

$$f_1 \cdot (\overline{f_1} \vee f_2) = f_1 f_2$$

(f_1, f_2 können Variable oder Teilschaltfunktionen sein (Einsetzungs- und Ersetzungsregel).)

Auf Variable angewandte Kürzungsregeln (Auswahl):

$$a \cdot (a \vee b) = a; \quad a \cdot (\overline{a} \vee b) = a \cdot b$$

$$\overline{a} \cdot (a \vee b) = \overline{a} \cdot b; \quad \overline{a} \cdot (\overline{a} \vee b) = \overline{a} \vee b$$

$$\mathbf{a \vee (a \cdot b) = a; \quad a \vee (\bar{a} \cdot b) = a \vee b;}$$

$$\bar{\mathbf{a}} \vee (\mathbf{a} \cdot \mathbf{b}) = \bar{\mathbf{a}} \vee \mathbf{b}; \quad \bar{\mathbf{a}} \vee (\bar{\mathbf{a}} \cdot \mathbf{b}) = \bar{\mathbf{a}} \vee \mathbf{b}$$

Elementarfunktionen mit mehr als 2 Variablen

UND

$\mathbf{a \cdot b \cdot c \dots = a \cdot (b \cdot (c \dots))}$. Wahrheitswert 1: wenn alle Variable a, b, c... den Wahrheitswert 1 haben.
Wahrheitswert 0: hierzu genügt es, daß eine der Variablen den Wahrheitswert 0 hat.

ODER

$\mathbf{a \vee b \vee c \vee \dots = a \vee (b \vee (c \vee \dots))}$. Wahrheitswert 0: wenn alle Variable a, b, c... den Wahrheitswert 0 haben.
Wahrheitswert 1: hierzu genügt es, daß eine der Variablen den Wahrheitswert 1 hat.

NAND

$\mathbf{a \cdot b \cdot c \dots}$. Wahrheitswert 0: wenn alle Variable a, b, c... den Wahrheitswert 1 haben.
Wahrheitswert 1: hierzu genügt es, daß eine der Variablen den Wahrheitswert 0 hat.

NOR

$\mathbf{a \vee b \vee c \vee \dots}$. Wahrheitswert 1: wenn alle Variable a, b, c... den Wahrheitswert 0 haben.
Wahrheitswert 0: hierzu genügt es, daß eine der Variablen den Wahrheitswert 1 hat.

Antivalenz

$\mathbf{a \oplus b \oplus c \oplus \dots = a \oplus (b \oplus (c \oplus \dots))}$. Wahrheitswert 0: wenn die Anzahl der Einsen gerade ist.
Wahrheitswert 1: wenn die Anzahl der Einsen ungerade ist.

Äquivalenz

$\mathbf{a \equiv b \equiv c \equiv \dots = a \equiv (b \equiv (c \equiv \dots))}$. Wahrheitswert 0: wenn die Anzahl der Einsen ungerade ist.
Wahrheitswert 1: wenn die Anzahl der Einsen gerade ist.

4. Gleichungssysteme

Mit Booleschen Gleichungen kann man ebenso umgehen wie mit den üblichen algebraischen Gleichungen: die Gleichung wird nicht verändert, wenn man auf beide Seiten die gleiche Operation anwendet.

Die allgemeine Form einer Booleschen Gleichung

Zwei Boolesche Funktionen $f(\underline{x})$, $g(\underline{y})$ werden einander gleichgesetzt:

$$f(\underline{x}) = g(\underline{y}).$$

Hierbei bezeichnen x , y beliebige Tupel Boolescher Variabler:

$$\underline{x} = \{x_1, x_2, \dots, x_n\}, \underline{y} = \{y_1, y_2, \dots, y_m\}.$$

Homogenisierung

Jede beliebige Boolesche Gleichung läßt sich durch Antivalenzverknüpfung in eine homogene Gleichung (deren rechte Seite Null ist) umwandeln:

$f(\underline{x}) \oplus g(\underline{y}) = g(\underline{y}) \oplus g(\underline{y})$; es gilt aber: $g(\underline{y}) \oplus g(\underline{y}) = 0$. Also folgt:

$$f(\underline{x}) \oplus g(\underline{y}) = 0.$$

Gleichungssysteme

Gelegentlich ergeben sich mehrere zusammengehörende Boolesche Gleichungen, mit anderen Worten: ein Gleichungssystem:

$$f_1(\underline{x}_1) = g_1(\underline{y}_1),$$

$$f_2(\underline{x}_2) = g_2(\underline{y}_2),$$

$$f_3(\underline{x}_3) = g_3(\underline{y}_3) \text{ usw.}$$

Die einzelnen Gleichungen kann man zunächst homogenisieren:

$$f_1(\underline{x}_1) \oplus g_1(\underline{y}_1) = 0,$$

$$f_2(\underline{x}_2) \oplus g_2(\underline{y}_2) = 0,$$

$$f_3(\underline{x}_3) \oplus g_3(\underline{y}_3) = 0 \text{ usw.}$$

Zusammenfassung zu einer einzigen homogenen Gleichung

Mehrere homogene Boolesche Gleichungen lassen sich zu einer einzigen homogenen Booleschen Gleichung zusammenfassen, indem man sie miteinander disjunktiv verknüpft (die resultierende Gleichung ist nur dann = 0, wenn jede der disjunktiv verknüpften Gleichungen = 0 ist):

$$(f_1(\underline{x}_1) \oplus g_1(\underline{y}_1)) \vee (f_2(\underline{x}_2) \oplus g_2(\underline{y}_2)) \vee (f_3(\underline{x}_3) \oplus g_3(\underline{y}_3)) \vee \dots = 0$$

Zur praktischen Bedeutung

Durch Homogenisieren und Zusammenfassen lassen sich viele Problemstellungen, die auf Booleschen Gleichungen beruhen, dadurch lösen, daß man die Nullstellen einer einzigen Booleschen Gleichung untersucht. Die Nullstellen einer Booleschen Gleichung ergeben sich unmittelbar aus der Wahrheitstabelle (indem man nur jene Variablenbelegungen betrachtet, die dem Funktionswert 0 entsprechen) oder aus einem binären Entscheidungsdiagramm (indem man die mit 0 belegten Endknoten aufsucht und sich von dort aus die jeweiligen Variablenbelegungen erschließt).

Auflösen von Booleschen Gleichungen

Viele Boolesche Gleichungen (aber nicht alle) lassen sich nach ihren einzelnen Variablen auflösen. Auflösen nach einer Variablen x_i heißt, eine gegebene Gleichung so umzustellen, daß auf der einen Seite nur die Variable x_i und auf der anderen eine Boolesche Funktion steht, die x_i nicht enthält.

Gegeben: $f(x_1, x_2, \dots, x_i, \dots, x_n) = 0$; *gesucht:* $x_i = h(x_1, x_2, \dots, x_n)$

Die Funktion $h(x_1, x_2, \dots, x_n)$ muß so bestimmt werden, daß, wenn diese Funktion anstelle von x_i in die ursprüngliche Gleichung eingesetzt wird, sich wiederum 0 ergibt:

$$f(x_1, x_2, \dots, h(x_1, x_2, \dots, x_n), \dots, x_n) = 0.$$

Beispiel:

Die Gleichung $a \bar{b} \vee \bar{a} b = 0$ ist nach b aufzulösen. Hier sieht man die Lösung fast auf den ersten Blick: $\underline{b \equiv a}$. Beweis durch Einsetzen: $a \bar{a} \vee \bar{a} a = 0$.

5. Normalformen

Eine Normalform ist die Darstellung einer Schaltfunktion als Schaltgleichung, die nach bestimmten Regeln aufgebaut ist. In der Schaltalgebra kennt man verschiedene Normalformen. Jede beliebige Schaltgleichung ist in jeder Normalform darstellbar.

Aufbau von Normalformen

Wir betrachten eine Boolesche Funktion, die von den Variablen x_1, x_2, \dots, x_n abhängt. Normalformen bestehen aus "Bausteinen" (Termen), die als konjunktive oder disjunktive Verknüpfung aller Variablen aufgebaut sind, wobei die Variablen entweder nicht negiert oder *einzel*n negiert auftreten:

Konjunktionsterme (Minterme):

$$K_i = x_1 \wedge x_2 \wedge \dots \wedge x_n = x_1 x_2 \dots x_n \quad (x_1, x_2, \dots, x_n \text{ einzeln negiert oder nicht negiert}).$$

Beispiel: $K_i = \bar{x}_1 \bar{x}_2 x_3$. *Unzulässig:* $K_i = \bar{x}_1 \bar{x}_2 x_3$.

Disjunktionsterme (Maxterme):

$$D_i = x_1 \vee x_2 \vee \dots \vee x_n \quad (x_1, x_2, \dots, x_n \text{ einzeln negiert oder nicht negiert}).$$

Beispiel: $D_i = x_1 \vee \bar{x}_2 \vee \bar{x}_3$. *Unzulässig:* $D_i = x_1 \vee \bar{x}_2 \vee x_3$

Mit diesen Termen lassen sich verschiedene Normalformen bilden:

a) *disjunktive Normalform:*

$$f = K_1 \vee K_2 \vee \dots \vee K_n$$

b) *konjunktive Normalform:*

$$f = D_1 \wedge D_2 \wedge \dots \wedge D_n$$

c) *Antivalenz-Normalform:*

$$f = K_1 \oplus K_2 \oplus \dots \oplus K_n$$

d) *Äquivalenz-Normalform:*

$$f = D_1 \equiv D_2 \equiv \dots \equiv D_n$$

In der Praxis ist vor allem die disjunktive Normalform (DNF) von Bedeutung, gelegentlich auch die konjunktive Normalform (KNF).

Begriffsbildungen

Summen und Produkte

Gelegentlich bezeichnet man – mit Bezug auf naheliegende Analogien – die Disjunktion als Summe und die Konjunktion als Produkt (Disjunktion: $a + b$; Konjunktion: $a \cdot b$ oder ab).

Sum of Products (SOP oder S.O.P.) ist eine übliche Bezeichnung für disjunktive Normalformen (= disjunktive Verknüpfung von Konjunktionen).

Product of Sums (POS oder P.O.S.) ist eine übliche Bezeichnung für konjunktive Normalformen (= konjunktive Verknüpfung von Disjunktionen).

Minterme

Diese Bezeichnung für Konjunktionsterme erklärt sich daher, weil es in einer disjunktiven Normalform (DNF, SOP) genügt, daß ein einziger Konjunktionsterm erfüllt ist, damit sich ein Funktionswert = 1 ergibt.

Maxterme

Diese Bezeichnung für Disjunktionsterme erklärt sich daher, weil es in einer konjunktiven Normalform (KNF, POS) erforderlich ist, daß alle Disjunktionsterme erfüllt sind, damit sich ein Funktionswert = 1 ergibt.

Implikanden

Dies ist eine andere Bezeichnung für die Konjunktions- bzw. Disjunktionsterme.

Elementarkonjunktionen, Elementardisjunktionen

Dies sind Implikanden (Konjunktions- bzw. Disjunktionsterme), die *alle* Variablen (negiert oder nicht negiert) enthalten.

Kanonische Normalformen

Als “kanonisch” bezeichnet man Normalformen “im eigentlichen Sinne”, nämlich solche, in denen jeder der Konjunktionsterme (DNF) oder der Disjunktionsterme (KNF) tatsächlich alle Variablen (negiert oder nicht negiert) enthält – mit anderen Worten: Normalformen, die ausschließlich aus Elementarkonjunktionen oder aus Elementardisjunktionen aufgebaut sind.

Reduzierte Normalformen

Eine Normalform heißt “reduziert”, wenn nicht jeder der Konjunktionsterme (DNF) oder der Disjunktionsterme (KNF) *alle* Variablen (negiert oder nicht negiert) enthält (wenn also in wenigstens einem der Terme wenigstens eine Variable fehlt - mit anderen Worten: wenn wenigstens ein Implikand keine Elementarkonjunktion oder Elementardisjunktion ist).

Hinweise:

1. Alle Schaltfunktionen lassen sich als kanonische Normalformen darstellen, aber nicht alle Schaltfunktionen als reduzierte.
2. Die Bezeichnung “reduziert” wird meistens weggelassen; man unterscheidet dann zwischen kanonischen und “gewöhnlichen” (nicht näher bezeichneten) Normalformen.

Erfüllung von Normalformen

Problem: Es ist eine bestimmte Variablenbelegung gegeben. Erfüllt diese Belegung die jeweiligen Gleichung oder nicht?

Die disjunktive Normalform

- ist bereits dann erfüllt, wenn die Variablenbelegung nur einem einzigen Implikanden in allen Variablenpositionen entspricht,
- ist nicht erfüllt, wenn die Variablenbelegung allen Implikanden in wenigstens jeweils einer Variablenposition nicht entspricht.

Die konjunktive Normalform

- ist dann erfüllt, wenn die Variablenbelegung allen Implikanden in wenigstens jeweils einer Variablenposition entspricht,
- ist bereits dann nicht erfüllt, wenn die Variablenbelegung nur einem einzigen Implikanden in keiner Variablenposition entspricht.

Normalformen und Schaltungstechnik:

- die disjunktive Normalform entspricht den Schaltungsstrukturen UND-ODER bzw. NAND-NAND,
- die konjunktive Normalform entspricht den Schaltungsstrukturen ODER-UND bzw. NOR-NOR.

Von der Wahrheitstabelle zur kanonischen disjunktiven Normalform:

- es werden alle Belegungen ausgewertet, die den Funktionswert Eins ergeben (On-Set).
- für jede dieser Belegungen wird ein Konjunktionsterm gebildet, der aus allen Variablen besteht (Elementarkonjunktion).
- die mit Null belegten Variablen werden im Konjunktionsterm negiert.

Von der Wahrheitstabelle zur kanonischen konjunktiven Normalform:

- es werden alle Belegungen ausgewertet, die den Funktionswert Null ergeben (Off-Set).
- für jede dieser Belegungen wird ein Disjunktionsterm gebildet, der aus allen Variablen besteht (Elementardisjunktion).
- die mit Eins belegten Variablen werden im Disjunktionsterm negiert.

Prinzip:

Der Funktionswert 1 ergibt sich dann, wenn keine der Variablenbelegungen vorliegt, die den Funktionswert Null ergibt. Eine Variablenbelegung $x_1x_2\dots x_n = 0$ liegt dann nicht vor, wenn gilt $\overline{x_1} \vee \overline{x_2} \vee \dots \vee \overline{x_n} = 1$ (Negation nach DeMorgan).

Welche Normalform? – Naheliegende Ansätze zur Auswahl:

1. **Aufwandsminimierung:** Die Funktionswerte ansehen. Überwiegt die Anzahl der Nullen, die disjunktive Normalform wählen, überwiegt die Anzahl der Einsen, die konjunktive Normalform.
2. **Schaltungstechnik:** Welche Gatterfunktion ist die elementare (die den weiteren Schaltungen zugrunde liegt)? Wenn NAND (Beispiel: TTL), die disjunktive Normalform wählen, wenn NOR (Beispiel: ECL), die konjunktive.

Von der disjunktiven zur konjunktiven Normalform:

1. Funktion nach DeMorgan negieren,
2. Klammern auflösen,
3. Funktion erneut nach DeMorgan negieren.

Von der konjunktiven zur disjunktiven Normalform:

1. Klammern auflösen,
2. redundante Konjunktionsterme streichen.

Disjunktive Normalformen und Belegungslisten

Die praktische Bedeutung disjunktiver Normalformen ergibt sich aus zwei Tatsachen:

1. sie hängt auf einfache Weise mit anderen Darstellungsformen zusammen (Abb. 5.1),
2. sie entspricht direkt einer zweistufigen UND-ODER- bzw. NAND-NAND-Schaltungsstruktur.

Wahrheitstabelle:

a	b	c	R
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Kanonische disjunktive Normalform (KDNF):

$$R = \bar{a}b\bar{c} \vee \bar{a}bc \vee a\bar{b}c$$

Belegungsliste (binär):

a	b	c
0	1	0
0	1	1
1	0	1

Disjunktive Normalform (DNF, S.O.P.-Darstellung):

$$R = \bar{a}b \vee a\bar{b}c$$

Belegungsliste (ternär):

a	b	c
0	1	-
1	0	1

Abb. 5.1 Darstellungsweisen einer Schaltfunktion

Implikanten und Belegungen

Jeder Implikant einer disjunktiven Normalform entspricht direkt einer Variablenbelegung, die die Boolesche Funktion erfüllt:

- jede nichtnegiert vorkommende Variable entspricht einer Eins,
- jede negiert vorkommende Variable entspricht einer Null,
- jede fehlende (nicht vorkommende) Variable ist bedeutungslos (Don't Care) und entspricht einem Strichelement (-).

Beispiele: Wir betrachten eine Funktion von 3 Variablen a, b, c .

1. *Beispiel:* $\bar{a}\bar{b}c \hat{=} a = 0, b = 1, c = 0$

2. *Beispiel:* $\bar{a}b \hat{=} a = 0, b = 1, c = -$

Die kanonische disjunktive Normalform (KDNF) entspricht einer binären, die "gewöhnliche" (reduzierte) DNF einer ternären Belegungsliste.

Von der reduzierten zur kanonischen disjunktiven Normalform (KDNF)

Der Weg entspricht jenem von der ternären zur binären Belegungsliste:

- Implikanden, die bereits Elementarkonjunktionen sind, werden direkt in die KDNF übernommen,
- jeder weitere Implikand wird zunächst in die entsprechend ternäre Variablenbelegung gewandelt; dann werden die Strichelemente aufgelöst. Die resultierenden binären Variablenbelegungen werden dann wieder in Elementarkonjunktionen gewandelt. So werden aus einem Implikanden, dem die einzige Variable x_i fehlt, zwei Implikanden (der erste mit x_i und der zweite mit \bar{x}_i). Aus einem Implikanden, in dem zwei Variable x_i, x_j fehlen, werden vier Implikanden (so daß alle Kombinationen der Variablenbelegung vorkommen: $(\bar{x}_i, \bar{x}_j; \bar{x}_i, x_j; x_i, \bar{x}_j; x_i, x_j)$ usw.

Beispiel: aus $a b$ wird $a b \bar{c} \vee a b c$.

Von der kanonischen zur reduzierten Normalform

Der einfachste Fall: zwei Elementarkonjunktionen oder Elementardisjunktionen unterscheiden sich nur in einer Variablen voneinander (die in einem Implikanden negiert und im anderen nichtnegiert vorkommt): dann können diese beiden Implikanden durch einen einzigen Implikanden ersetzt werden, in dem diese Variable fehlt.

Beispiel: $a b c \vee a b \bar{c} = a b$.

Dies entspricht dem Zusammenfassen von zwei binären Variablenbelegungen, die sich nur in einer Variablenposition voneinander unterscheiden, zu einer einzigen ternären Belegung, die in dieser Position einen Strich enthält. Das entspricht sinngemäß der Anwendung der 2. Kürzungsregel.

Beispiel: $a b = f_1; c = f_2; (a b) c \vee (a b) \bar{c} = a b$.

6. DNF-Minimierung nach Quine-McCluskey

Das Minimierungsverfahren hat das Ziel, aus einer gegebenen kanonischen disjunktiven Normalform eine äquivalente disjunktive Normalform zu bestimmen, die (1) eine minimale Anzahl an Implikanden aufweist und deren Implikanden (2) so kurz wie möglich sind (d. h. so wenige Variable wie möglich enthalten).

Hinweis:

Das entspricht technisch einer zweistufigen UND-ODER- bzw. NAND-NAND-Schaltung mit minimalem Aufwand.

Das Verfahren besteht aus 2 Schritten:

1. Schritt - Bestimmung der Primimplikanden:

1. die kanonische disjunktive Normalform (KDNF) wird in eine binäre Belegungsliste überführt,
2. es wird versucht, deren Einträge zu ternären Belegungen (Primimplikanden) zusammenzufassen¹. Abb. 6.1 veranschaulicht die Zusammenfassungsregeln.

1): Ein Primimplikand ist ein Implikand, der sich nicht mehr gemäß Abb. 6.1 (gleichbedeutend: gemäß der 2. Kürzungsregel) mit anderen Implikanden zusammenfassen läßt.

2. Schritt - Auswahl eines minimalen Satzes an Primimplikanden:

Aus den im ersten Schritt bestimmten Primimplikanden werden jene herausgesucht, die notwendig und hinreichend sind, um die Schaltfunktion zu erfüllen.

Primimplikand

$$\text{a) } \begin{array}{cccc} 1 & \boxed{0} & 0 & 1 \\ 1 & \boxed{1} & 0 & 1 \end{array} \Rightarrow 1 - 0 1 \quad \text{b) } \begin{array}{ccc} 1 & \boxed{-} & 0 \quad \boxed{-} \\ 1 & \boxed{-} & 0 \quad \boxed{1} \end{array} \Rightarrow 1 - 0 -$$

Abb. 6.1 Zusammenfassungenregeln beim Verfahren nach Quine-McCluskey

- erste Stufe der Zusammenfassung (binär \Rightarrow ternär): zwei binäre Belegungen werden zu einer einzigen ternären zusammengefaßt, wenn sie sich nur in einer einzigen Variablenposition unterscheiden. Diese wird dann mit einem Strich belegt.
- Zusammenfassung ternärer Belegungen: zwei ternäre Belegungen lassen sich zusammenfassen, wenn sie (1) Striche nur in gleichen Variablenpositionen haben¹⁾ und sich (2) ansonsten nur in einer Variablenposition unterscheiden (diese wird - als Ergebnis der Zusammenfassung - mit einem weiteren Strich belegt).

Hinweis:

Hier wird deutlich, daß “-” nicht ohne weiteres “Don’t Care” bedeutet (mit der gedanklichen Assoziation: “kann vernachlässigt werden”). Vielmehr beschreiben Strich-Belegungen jeweils einen Unterraum des durch alle Variablen gegebenen Booleschen Raums B^k (es sind alle Kombinationen der mit Strichen belegten Variablen zu berücksichtigen. Nur Variablenbelegungen, die den gleichen Unterraum betreffen, dürfen miteinander verglichen werden.

Der 1. Schritt des Minimierungsablaufs beruht auf einer Durchmusterung der Belegungsliste, wobei versucht wird, Belegungen zusammenzufassen, und zwar zunächst gemäß Abbildung 6.1a und nachfolgend gemäß Abbildung 6.1b. Dabei entsteht jeweils eine neue Liste. Belegungen, die sich nicht zusammenfassen lassen, verbleiben als Primimplikanden und werden in die jeweils nächste Liste übernommen. Der Ablauf endet, wenn eine Liste entstanden ist, deren Belegungen sich nicht mehr zusammenfassen lassen

Beispiel:

Folgende, durch ihre KDNF gegebene Schaltfunktion ist zu minimieren:

$$a \bar{b} \bar{c} \bar{d} \vee a b \bar{c} \bar{d} \vee a \bar{b} \bar{c} d \vee a b \bar{c} d \vee \bar{a} b c \bar{d} \vee \bar{a} \bar{b} \bar{c} d \vee \bar{a} \bar{b} c \bar{d}$$

Der 1. Schritt

Tabelle 6.1 veranschaulicht die einzelnen Teil-Schritte des Schrittes 1 nach Quine-McCluskey.

1): Mit anderen Worten: alle Strich-Belegungen müssen übereinstimmen.

Erklärung der einzelnen Spalten:

- 1) die einzelnen Elementarkonjunktionen der KDNF,
- 2) diese Spalte enthält die ursprüngliche (binäre) Belegungsliste. Im Interesse der Übersichtlichkeit sind die Einträge gemäß der Anzahl der Einsen sortiert¹⁾. Hiermit beginnt die Durchmusterung der Liste. Sie wird von unten abgebaut. Die unterste Belegung (1101) wird aus der Liste gestrichen, und alle anderen Belegungen werden mit der gestrichenen daraufhin verglichen, ob ein Zusammenfassen gemäß Abbildung 1a möglich ist²⁾.
- 3) die Belegungen, die mit 1101 zusammengefaßt werden konnten, bilden die ersten Belegungen der 2. (unteren) Liste. Oben: die gemäß 2) verkürzte Liste, wovon wiederum die unterste Belegung (1001) gestrichen wird.
- 4) die 2. (untere Liste) wurde um die Belegungen erweitert, die mit 1001 zusammengefaßt werden konnten. Oben: die gemäß 3) verkürzte Liste, wovon wiederum die unterste Belegung (0110) gestrichen wird.
- 5) die Belegung 0110 läßt sich mit keiner anderen zusammenfassen. Sie bleibt somit als Primimplikand erhalten und wird direkt in die 2. (untere) Liste aufgenommen. Oben: die gemäß 4) verkürzte Liste, wovon wiederum die unterste Belegung (1100) gestrichen wird.
- 6) die Belegung, die mit 1100 zusammengefaßt werden konnte, wird in die 2. (untere) Liste aufgenommen. Oben: die gemäß 5) verkürzte Liste, wovon wiederum die unterste Belegung (1000) gestrichen wird.
- 7) die Belegung, die mit 1000 zusammengefaßt werden konnte, wird in die 2. (untere) Liste aufgenommen. Oben: die gemäß 6) verkürzte Liste, wovon wiederum die unterste Belegung (0001) gestrichen wird.
- 8) die beiden letzten Belegungen der 1. (oberen) Liste (vgl. Spalte 7) lassen sich offensichtlich zusammenfassen. Das Ergebnis wird in die 2. (untere) Liste übernommen, die somit komplett ist. Nun wird die 2. (untere) Liste daraufhin untersucht, ob sich Belegungen gemäß Abbildung 6.1b zusammenfassen lassen. Es sind nur Belegungen miteinander zu vergleichen, die Striche ausschließlich an gleichen Positionen haben. Das wird in Tabelle 6.2 im einzelnen dargestellt.
- 9) was sich offensichtlich nicht zusammenfassen läßt, ist der verbliebene Primimplikand (a). Mit b...e kennzeichnen wir Belegungen, die sich zusammenfassen lassen.
- 10) die zusammengefaßten Belegungen bilden das Ergebnis. Zusammenfassungen, die jeweils das gleiche Ergebnis liefern (wie b und c bzw. d und e) werden jeweils nur einmal aufgenommen³⁾. In der so gebildeten Liste sind keine weiteren Möglichkeiten des Zusammenfassens zu erkennen, so daß die Minimierung beendet ist.
- 11) so sieht die minimierte (ternäre) Belegungsliste aus. Die entsprechende Schaltfunktion steht unter der Tabelle.

1): hat eine Belegung eine bestimmte Anzahl von Einsen, so muß man diese lediglich mit allen anderen Belegungen vergleichen, die die gleiche Anzahl an Einsen oder eine Eins weniger enthalten (Ablaufbeschleunigung). Deshalb auch das "Abbauen von unten" (= mit den Belegungen, die die meisten Einsen enthalten).

2): die Belegungen, die sich zusammenfassen lassen, sind in der Tabelle mit einem **x** gekennzeichnet.

3): grundsätzlich heißt das: nach jeder Zusammenfassung nachsehen, ob die gleiche Belegung schon in der Ergebnis-Liste (hier: in Spalte 10) steht.

1	2	3	4	5	6	7	8	9	10	11
$\bar{a} \bar{b} \bar{c} \bar{d}$	0000	0000	0000	0000	0000 X	0000 X				
$\bar{a} \bar{b} \bar{c} d$	0001	0001 X	0001	0001	0001	0001				
$a \bar{b} \bar{c} \bar{d}$	1000	1000 X	1000	1000 X	1000					
$a \bar{b} \bar{c} d$	1100 X	1100	1100	1100						
$\bar{a} b c \bar{d}$	0110	0110	0110							
$a \bar{b} c d$	1001 X	1001								
$a b \bar{c} d$	1101									
		110- 1-01	-001 100- 110- 1-01	0110 -001 100- 110- 1-01	1-00 0110 -001 100- 110- 1-01	-000 1-00 0110 -001 100- 110- 1-01	000- -000 1-00 0110 -001 100- 110- 1-01	b c d a c b,e e d	a: b,c: d,e:	0110 -00- 1-0-

Ursprüngliche KDNF: $a \bar{b} \bar{c} \bar{d} \vee a \bar{b} \bar{c} d \vee a \bar{b} c \bar{d} \vee a \bar{b} c d \vee a b \bar{c} d \vee a b c \bar{d} \vee a b c d \vee \bar{a} \bar{b} \bar{c} d \vee \bar{a} \bar{b} c \bar{d} \vee \bar{a} b \bar{c} d$

Minimiert: $\bar{a} b c \bar{d} \vee \bar{b} \bar{c} \vee a \bar{c}$

Tabelle 6.1 Minimierung nach Quine-McCluskey: Schritt 1 (Ablaufbeispiel)

1	2	3	4	5	6
000- -000 1-00 \times 0110 -001 100- 110- 1-01	000- -000 1-00 0110 -001 100- \times 110-	000- \times -000 1-00 0110 -001 100-	000- -000 \times 1-00 0110 -001	000- \times -000 \times 1-00 \times 0110	
	1-0-	1-0-	-00- 1-0-	-00- 1-0-	0110 -00- 1-0-

Tabelle 6.2 Minimierung nach Quine-McCluskey: Zusammenfassung der zweiten Liste (Tabelle 6.1, Spalte 8)

Das Vorgehen entspricht dem von Tabelle 6.1, nur werden hier die Strichelemente mit berücksichtigt. Aus Spalte 5 ist ersichtlich, daß ein weiteres Zusammenfassen nicht mehr gelingt. Die Belegung 0110 läßt sich mit keiner anderen zusammenfassen und bleibt als Primimplikand erhalten. Die anderen Belegungen in Spalte 5 entfallen, da sie bereits bei Zusammenfassungen berücksichtigt wurden. Spalte 6 enthält alle übriggebliebenen Belegungen (vgl. Spalte 11 in Tabelle 6.1).

Hinweise:

1. Die Tabellen sollen den Optimierungsablauf Schritt für Schritt veranschaulichen. Beim praktischen Optimieren von Hand wird man üblicherweise jede Liste nur einmal aufstellen. Wichtig ist hierbei, (1) die bereits "erledigten" Variablenbelegungen deutlich zu kennzeichnen und (2) keine Belegung zu übersehen.
2. Was auch vorkommen kann¹⁾: daß ganze Spalten ausschließlich mit Strichen belegt sind. Solche Variablen sind wirklich "Don't Care" (= redundant) und können weggelassen werden.

Der 2. Schritt

Es wird nachgesehen, welche Primimplikanden welche Variablenbelegungen erfüllen, um einen minimalen Satz an Primimplikanden zu bestimmen. Grundlage hierfür ist eine Primimplikandentabelle (Tabelle 6.3). Diese besteht aus einer ersten Spalte mit allen in Schritt 1 (Tabellen 6.1 und 6.2) ermittelten Primimplikanden (entsprechend Spalte 11 von Tabelle 6.1.1) sowie aus jeweils einer weiteren Spalte für jede der Variablenbelegungen, die die Schaltfunktion erfüllen soll. Für jeden Primimplikanden wird jede Variablenbelegung, die er erfüllt, im entsprechenden Kreuzungspunkt von Zeile und Spalte mit einem X gekennzeichnet. (Hierzu ist es notwendig, die Strichelemente aufzulösen und die entsprechenden Variablenbelegungen mit jenen der obersten Zeile zu vergleichen. Bei Übereinstimmung ist ein X einzutragen.)

1): im Entwurfsprozeß (bei der Überführung der Aufgabenstellung in eine Schaltungslösung) wird man zunächst intuitiv Boolesche Variable einführen und damit Schaltfunktionen aufstellen. Es kann sich dann – als Ergebnis der Minimierung – durchaus ergeben, daß einige Variable gar nicht erforderlich sind.

abcd	0000	0001	1000	1100	0110	1001	1101
0110					X		
-00-	X	X	X			X	
1-0-				X		X	X

Tabelle 6.3 Minimierung nach Quine-McCluskey. Schritt 2: Primimplikantentabelle

Der Minimierungsablauf:

1. Wir suchen nach Spalten, die *nur ein einziges X* enthalten. Die Primimplikanten den jeweiligen Zeilen sind unbedingt erforderlich (*essentielle Primimplikanten*).
2. Es ist zu prüfen, welche anderen Variablenbelegungen von den essentiellen Primimplikanten gleichsam mit erledigt (abgedeckt) werden.
3. Bleiben Variablenbelegungen übrig, die nicht von essentiellen Primimplikanten abgedeckt werden, so ist nach einer minimalen Zahl von Primimplikanten zu suchen, die diese Variablenbelegungen erfüllen.
4. Das Endziel: alle Variablenbelegungen müssen erfüllt (abgedeckt (covered)) werden, und zwar mit einer minimalen Anzahl an Primimplikanten.

Im bisher behandelten Beispiel (Tabelle 6.3) sind alle Primimplikanten essentiell, denn ersichtlicherweise gibt es für jeden der Primimplikanten wenigstens eine Spalte, in der ein einziges X steht. Wir betrachten deshalb ein weiteres Beispiel (Tabelle 6.4).

abcd	0100	1000	0101	0110	1001	1010	1101
0-00 (1)	X						
-000 (2)		X					
100- (3)		X			X		
10-0 (4)		X				X	
1-01 (5)					X		X
01- (6)	X		X	X			
-1-1 (7)			X				X

Tabelle 6.4 Eine weitere Primimplikantentabelle. (1)...(7): laufende Nummern der Primimplikanten

Die 7 Primimplikanten sollen die 7 Variablenbelegungen erfüllen. Ersichtlicherweise sind die Primimplikanten 4 und 6 essentiell, denn sie haben in jeweils einer Spalte nur ein einziges X. Wir markieren die betreffenden Zeilen (in der Tabelle durch entsprechende Hinterlegung gekennzeichnet) und suchen in diesen Zeilen die weiteren X-Einträge auf. Gibt es weitere Primimplikanten, die in den betreffenden Spalten X-Einträge haben, so sind diese überflüssig, da die betreffenden Variablenbelegungen schon vom jeweiligen essentiellen Primimplikanten abgedeckt werden.

Im einzelnen:

- Primimplikand 4 (10-0) deckt auch die Variablenbelegung 1000 ab, so daß Primimplikand 2 (-000) gar nicht mehr benötigt wird und Primimplikand 3 (100-) entfallen könnte, falls die weitere von ihm abgedeckte Belegung 1001 anderweitig abgedeckt wird (was tatsächlich der Fall ist)
- Primimplikand 6 (01-) deckt auch die Variablenbelegungen 0100 und 0101 ab. Damit könnten die Primimplikanden 1 (0-00) und 7 (-1-1) entfallen.
- Was verbleibt, sind die Variablenbelegungen 1001 und 1101. Diese werden ersichtlicherweise durch den Primimplikanden 5 (1-01) abgedeckt. Somit sind auch sämtliche von den Primimplikanden 3 und 7 abgedeckten Belegungen "erledigt", also können diese Primimplikanden tatsächlich entfallen.
- Es verbleibt ein minimaler Satz aus den Primimplikanden 4, 5 und 6. (10-0, 1-01, 01-). Als Boolesche Gleichung: $a \bar{b} \bar{d} \vee a \bar{c} d \vee \bar{a} b$.

7. Minimierung mittels Karnaugh-Plan (KV-Diagramm)

Die Karnaugh-Plan (auch Karnaugh-Veitch- (KV-) Diagramm; Abb. 7.1) ist eine zweidimensional angeordnete Wahrheitstabelle, aus der Zusammenfassungsmöglichkeiten unmittelbar zu erkennen sind. Hierzu sind die Binärvektoren, die den einzelnen Variablenbelegungen entsprechen, nicht gemäß dem üblichen Binärkode, sondern gemäß dem sog. Gray-Code (Tabelle 7.1) geordnet.

Zahlenwert	Binärkode				Gray-Code				Binär-Äquivalent
	B ₃	B ₂	B ₁	B ₀	G ₃	G ₂	G ₁	G ₀	
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	1
2	0	0	1	0	0	0	1	1	3
3	0	0	1	1	0	0	1	0	2
4	0	1	0	0	0	1	1	0	6
5	0	1	0	1	0	1	1	1	7
6	0	1	1	0	0	1	0	1	5
7	0	1	1	1	0	1	0	0	4
8	1	0	0	0	1	1	0	0	12
9	1	0	0	1	1	1	0	1	13
10	1	0	1	0	1	1	1	1	15
11	1	0	1	1	1	1	1	0	14
12	1	1	0	0	1	0	1	0	10
13	1	1	0	1	1	0	1	1	11
14	1	1	1	0	1	0	0	1	9
15	1	1	1	1	1	0	0	0	8

Tabelle 7.1 Binärkode und Gray-Code

Wichtig ist, daß sich beim Gray-Code von Zählwert zu Zählwert nur jeweils n einziges Bit ändert.

Umrechnung der Bitpositionen eines Zählwertes:

- Gray-Code aus Binärcode: $G_n = B_n \oplus B_{n+1}$
- Binärcode aus Gray-Code: $B_n = G_n \oplus B_{n+1}$

Hierbei ist n eine beliebige Bitposition. n+1 bezeichnet die jeweils links daneben angeordnete (= höherwertige) Bitposition. Ist n die höchstwertige Bitposition, so wird $B_{n+1} = 0$ angesetzt.

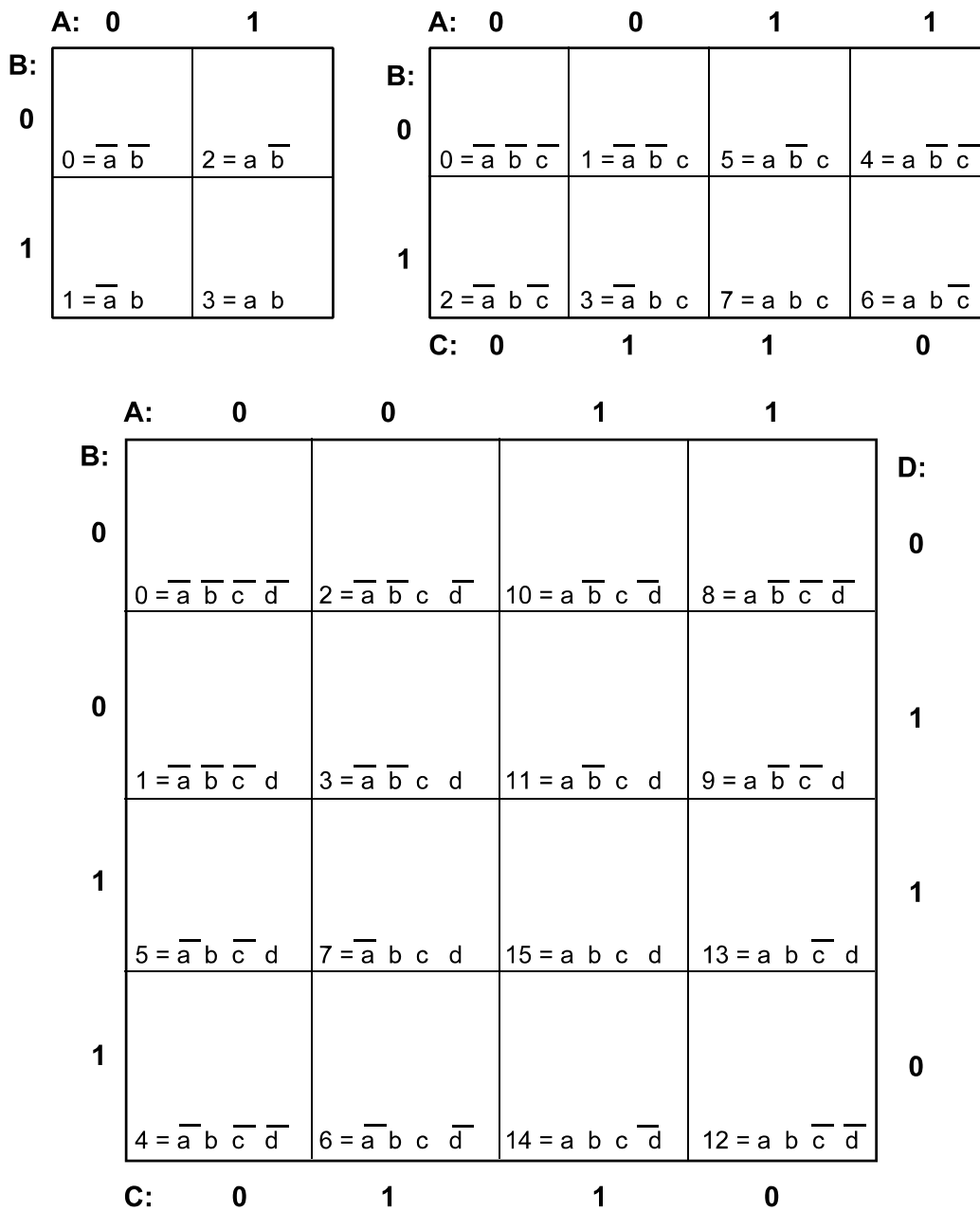


Abb. 7.1 Karnaugh-Pläne für 2, 3 und 4 Variable

Vorgehensweise:

1. Wahrheitswerte eintragen (Einsen (On-Set) + Don't Cares). Keine Eins vergessen! Ggf. Strichelemente auflösen. Beispiel: eine Schaltfunktion von 3 Variablen a, b, c enthalte einen Produktterm a b. Dann müssen in die Felder a b c und $a b \bar{c}$ Einsen eingetragen werden.
2. Nachsehen, was sich zusammenfassen läßt. Benachbarte Einsen oder Don't Cares zu möglichst großen Blöcken zusammenfassen.
3. Ergebnis ablesen. Es entfallen alle Variable, die in einem zusammengefaßten Block sowohl wahr als auch negiert vorkommen. Alle gleichbleibenden Variablen (nur wahr oder nur negiert) bilden einen Implikanden (UND-Term).

Zusammenfassungsregeln:

1. Benachbarte Einsen (bzw. Don't Cares) lassen sich zusammenfassen.
2. Die Anzahl der zusammengefaßten Einsen (bzw. Don't Cares) muß eine Zweierpotenz sein (2, 4, 8 usw.).
3. An entgegengesetzten Rändern angeordnete Einsen (bzw. Don't Cares) lassen sich zusammenfassen (Wrap Around).
4. Es können nur Belegungen zusammengefaßt werden, die nebeneinander oder übereinander stehen (nur waagrecht und senkrecht vorgehen; nicht diagonal).
5. Blöcke, die nur Don't Cares enthalten, werden nicht berücksichtigt.
6. Einsen (bzw. Don't Cares) dürfen in mehreren Blöcken enthalten sein.
7. Sind alle Einsen eines Blockes bereits in anderen Blöcken enthalten, so wird der erstgenannte Block nicht berücksichtigt.

Die Abb. 7.2 bis 7.4 veranschaulichen typische Zusammenfassungen. Abb. 7.5 zeigt die Minimierung des Beispiels von Kapitel 6 (vgl. Tabelle 6.1). Anschließend werden die Zusammenfassungsregeln anhand eines Praxisbeispiels (Siebensegmentdecoder) veranschaulicht.

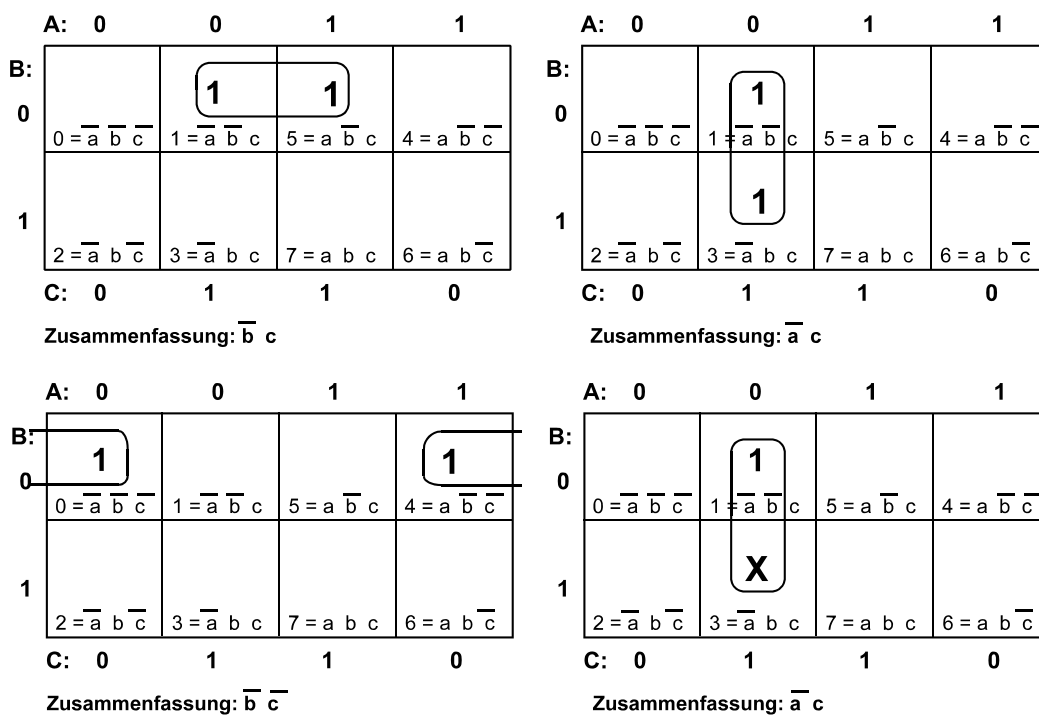


Abb. 7.2 Zusammenfassung von Zweierblöcken (Beispiele)

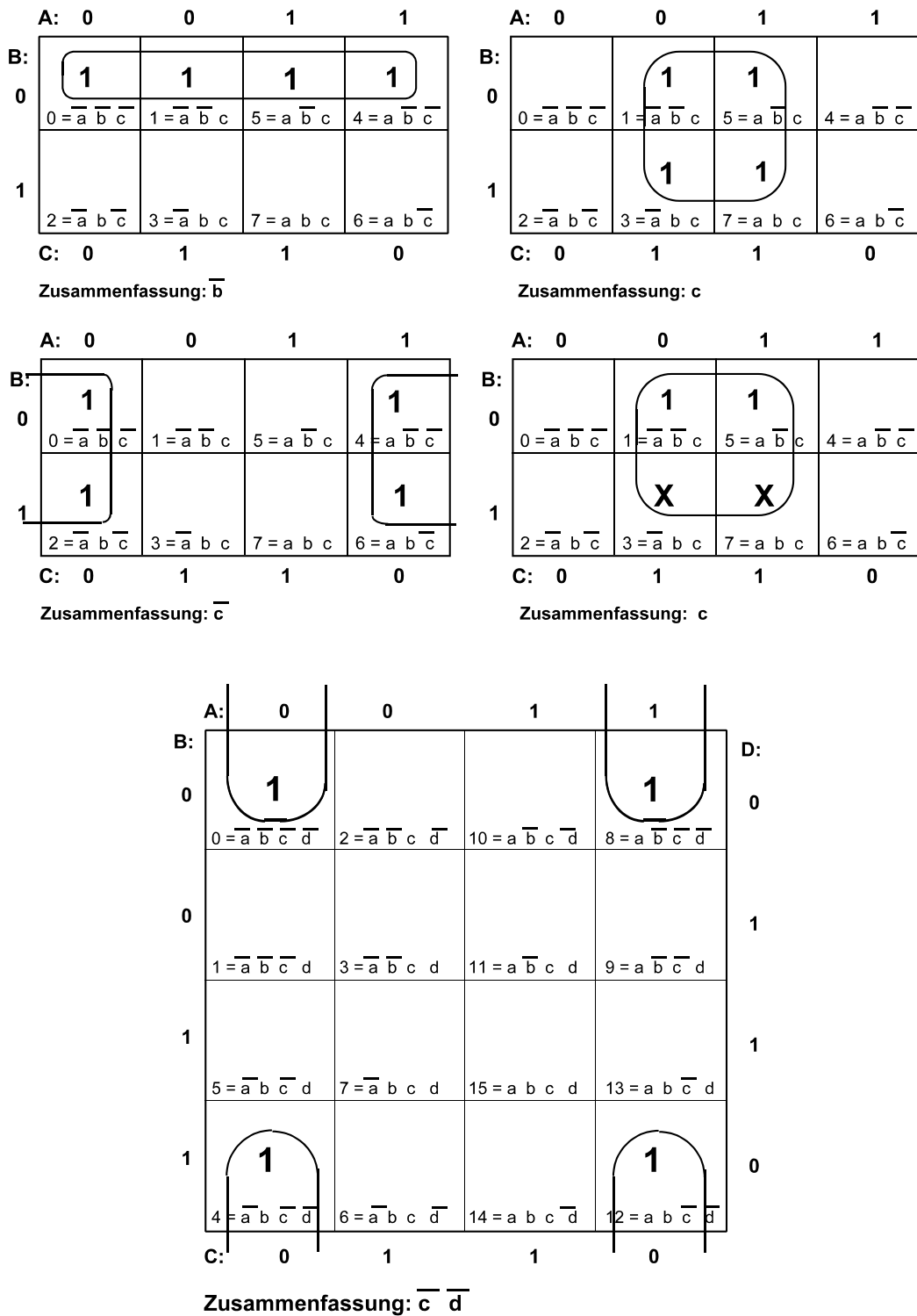


Abb. 7.3 Zusammenfassung von Viererblöcken (Beispiele)

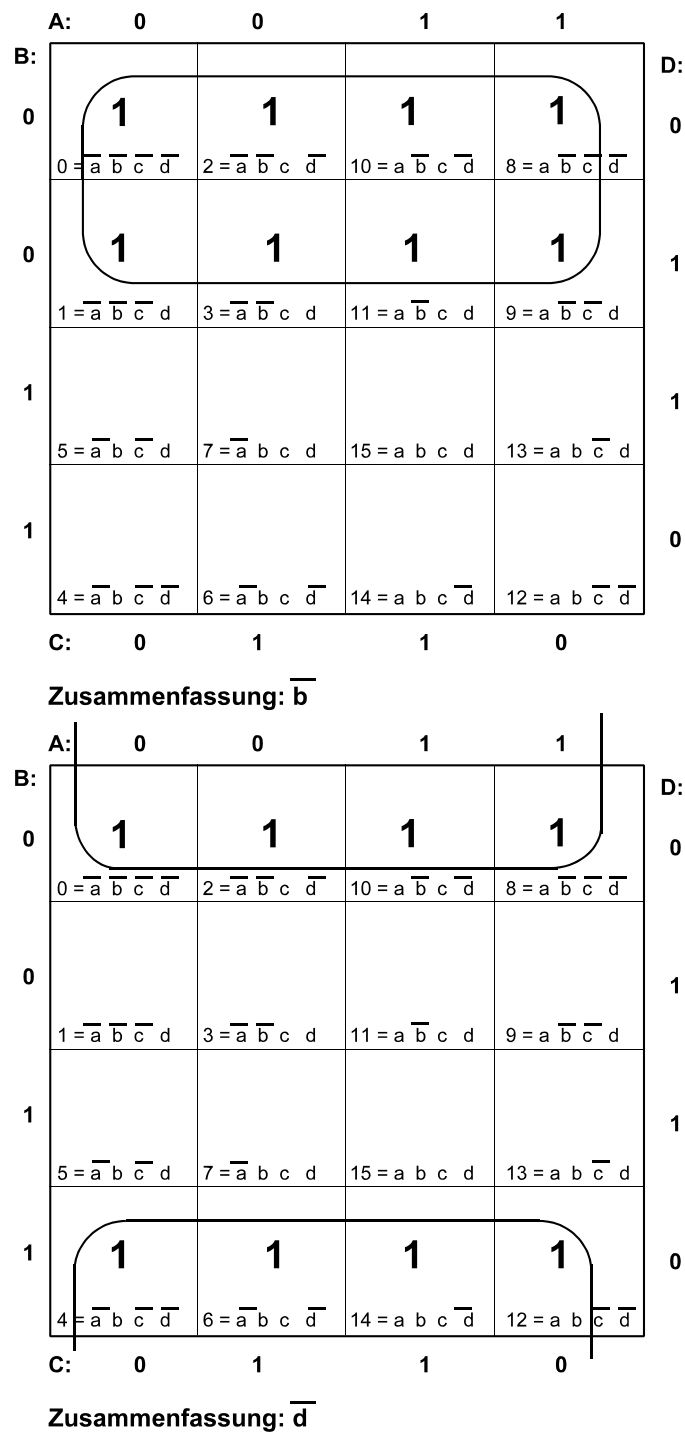
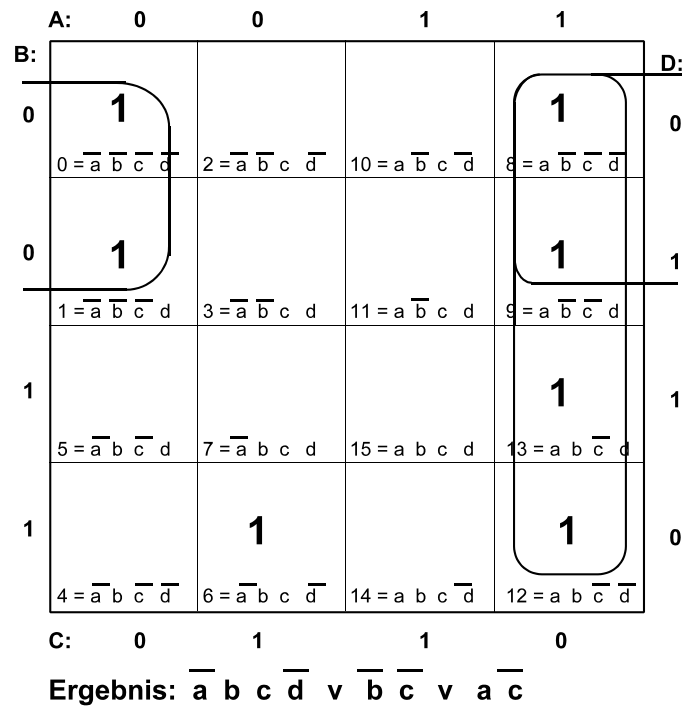


Abb. 7.4 Zusammenfassung von Achterblöcken (Beispiele)



Die ursprüngliche Schaltfunktion:

$$\bar{a} \bar{b} \bar{c} \bar{d} \vee \bar{a} \bar{b} c \bar{d} \vee \bar{a} \bar{b} c d \vee \bar{a} b \bar{c} \bar{d} \vee \bar{a} b c \bar{d} \vee \bar{a} b c d \vee a \bar{b} \bar{c} \bar{d} \vee a \bar{b} c \bar{d} \vee a \bar{b} c d \vee a b \bar{c} \bar{d} \vee a b c \bar{d} \vee a b c d$$

Abb. 7.5 Das Beispiel von Kapitel 6

Beispiel: der Siebensegmentdecoder

Eine im Binärcode (vier Bits mit Werten zwischen 0 (0000) und 9 (1001)) gegebene Dezimalzahl ist auf einer Siebensegmentanzeige darzustellen (Abb. 7.6, Tabelle 7.2). Hierzu sind die eingangsseitigen Binärwerte in entsprechende Erregungen der sieben Segmentleitungen (A...G) umzuschlüsseln. Binärwerte > 9 treten nicht auf. Diese Belegungen (1010...1111 bzw. AH...FH) können also ggf. als Don't Cares zur Schaltungsoptimierung herangezogen werden.

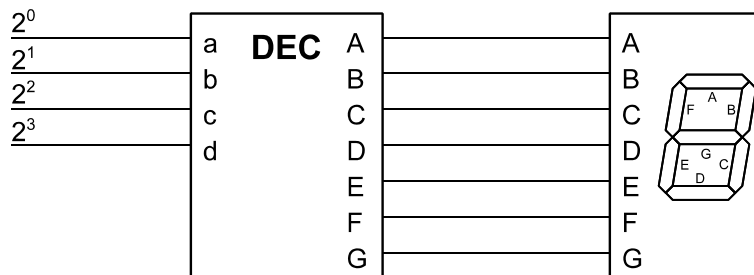


Abb. 7.6 Siebensegmentdecoder

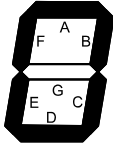
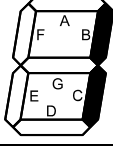
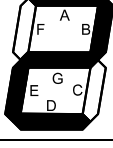
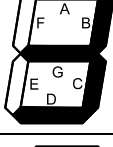
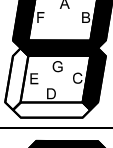
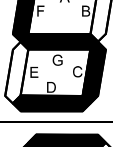
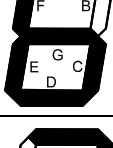
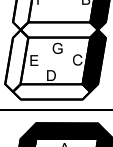
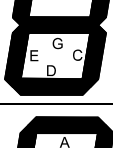
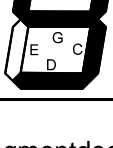
d	c	b	a	G	F	E	D	C	B	A	
0	0	0	0	0	1	1	1	1	1	1	
0	0	0	1	0	0	0	0	1	1	0	
0	0	1	0	1	0	1	1	0	1	1	
0	0	1	1	1	0	0	1	1	1	1	
0	1	0	0	1	1	0	0	1	1	0	
0	1	0	1	1	1	0	1	1	0	1	
0	1	1	0	1	1	1	1	1	0	1	
0	1	1	1	0	0	0	0	1	1	1	
1	0	0	0	1	1	1	1	1	1	1	
1	0	0	1	1	1	0	1	1	1	1	

Tabelle 7.2 Übersicht über die Informationswandlungen des Siebensegmentdecoders

Intuitiv entwerfen

Wir entschlüsseln zunächst die eingangsseitige Binärzahl. Hierzu dient ein BCD-zu-Dezimal-Decoder (4-zu-10 Decoder), der gemäß der anliegenden Binärzahl jeweils eines der Ausgangssignale 0...9 erregt (1-aus-n-Code). Dann müssen wir nur nachsehen, bei welchen Dezimalzahlen die einzelnen Segmente zum Leuchten zu bringen sind (Tabelle 7.3). Beispiel: Segment A muß leuchten bei den Zahlen 0, 2, 3, 5, 6, 7, 8, 9. Es liegt nahe, für jedes Segment ein entsprechendes ODER-Gatter vorzusehen (Abb. 7.7).

Segment	9	8	7	6	5	4	3	2	1	0	On-Set
A	1	1	1	1	1	0	1	1	0	1	8
B	1	1	1	0	0	1	1	1	1	1	8
C	1	1	1	1	1	1	1	0	1	1	9
D	1	1	0	1	1	0	1	1	0	1	7
E	0	1	0	1	0	0	0	1	0	1	4
F	1	1	0	1	1	1	0	0	0	1	6
G	1	1	0	1	1	1	1	1	0	0	7

Tabelle 7.3 Die Erregung der sieben Segente in Abhängigkeit von der jeweiligen Binärzahl. 0 = leuchtet nicht, 1 = leuchtet. Die Spalte "On-Set" gibt die Anzahl der Einsen an

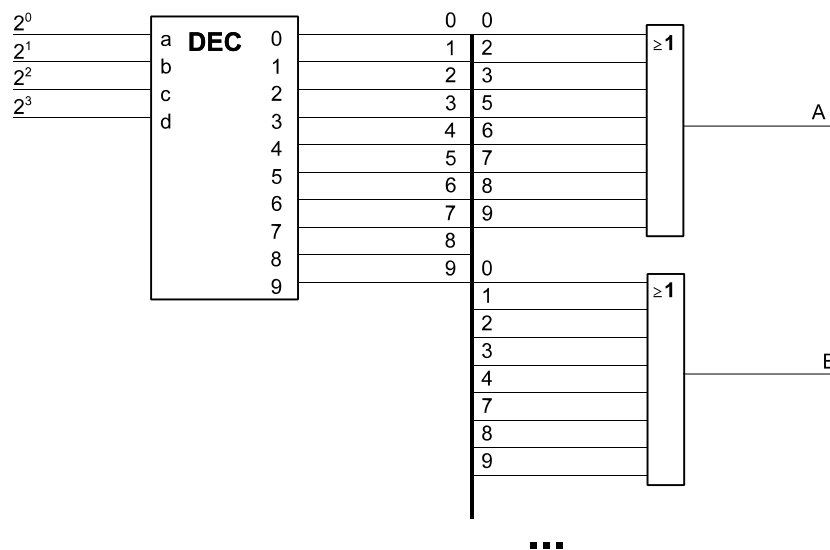


Abb. 7.7 Ganz nach Gefühl entwerfen... Siebensegmentdecoder auf Grundlage eines BCD-zu-Dezimal-Decoders (Schaltungsausschnitt). Es ist nur die Ansteuerung der Segmente A und B dargestellt. Es sind die Signale angeschlossen, die in Tabelle 7.3 mit einer Eins gekennzeichnet sind

Diese Lösung ist aber – wegen der hohen Eingangszahlen der ODER-Gatter – offensichtlich unwirtschaftlich. Der Ausweg: wir sehen nach, ob es einfacher wird, wenn wir die Segmente erfassen, die jeweils nicht leuchten sollen, und die Schaltfunktionen invertieren. Beispiel: Segment A leuchtet nicht bei den Zahlen 0 und 4. Somit liegt es nahe, ein entsprechende NOR-Verknüpfung vorzusehen (Abb. 7.8). Bei welchen Segmenten sich diese Verfahrensweise lohnt, ergibt sich aus der Mächtigkeit des On-Set (vgl. Tabelle 7.3). Da es sich um 10 Zahlenwerte handelt, werden im Booleschen Raum insgesamt 10 Punkte belegt. Es liegt nahe, die invertierte Schaltung dann zu wählen, wenn der On-Set mehr als 5 Punkte enthält. Das ist offensichtlich bei allen Segmente außer Segment E der Fall.

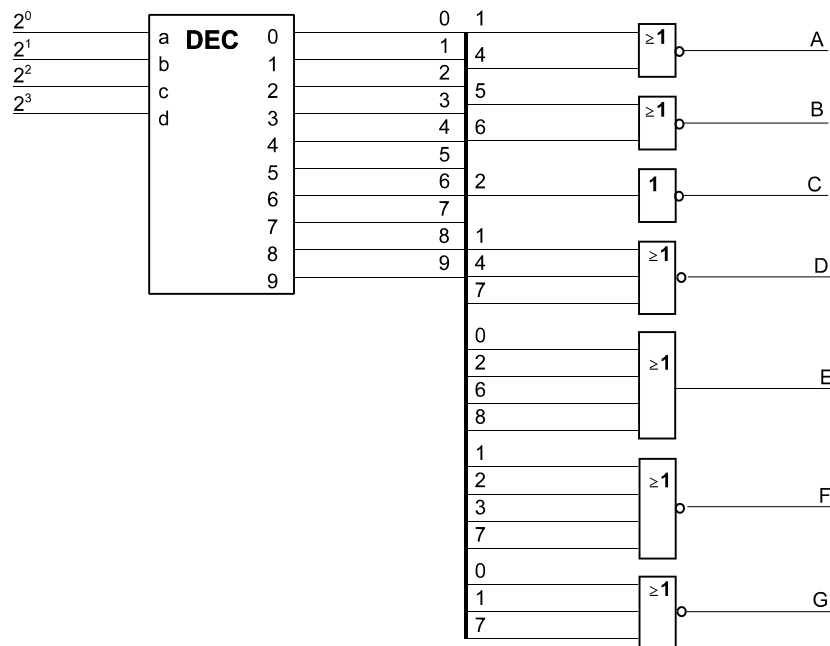


Abb. 7.8 Ein intuitiv entworfener Siebensegmentdecoder

Der BCD-zu-Dezimal-Decoder ist gesondert zu entwerfen (wobei die Don't-Care-Belegungen 1010 bis 1111 ggf. zu Optimierungszwecken genutzt werden können). Im Beispiel ergibt sich noch eine weitere Vereinfachungsmöglichkeit ... (Aufgabe: welche?)

Der Decoderausgang „9“ wird gar nicht genutzt und kann somit weggelassen werden.

Mit Karnaugh-Plan entwerfen

Wir entwerfen eine Schaltung, die – ohne zwischengeordnete Decoderschaltung – die gesamte Informationswandlung BCD zu Siebensegment erledigt. Hierzu ist je Segment ein Karnaugh-Plan anzulegen und gemäß Tabelle 7.2 auszufüllen (Abb. 7.9 bis 7.15). Die Variablenbelegungen 10 bis 15 werden als Don't Cares eingetragen.

Geht es noch einfacher?

Aus Tabelle 7.3 geht hervor, daß die On-Sets der Segmente A, B, C, D, F, G jeweils mehr als 5 Belegungen umfassen. Deshalb soll auch ausprobiert werden, inwiefern sich Schaltfunktionen optimieren lassen, die inaktive (= nicht leuchtende) Segmente betreffen (diese Funktionen sind dann lediglich noch zu negieren). Die Abb. 7.16 bis 7.21 zeigen die entsprechenden Karnaugh-Pläne.

Variantenvergleich

In Abb. 7.22 sind die Schaltfunktionen der optimierten Varianten zusammengefaßt. Die Abb. 7.23 und 7.24 zeigen die entsprechenden Schaltpläne. Tabelle 7.4 gibt einen vergleichenden Überblick über die verschiedenen Lösungen.

Hinweis:

Implikanden (UND-Terme), die in mehreren Schaltfunktionen vorkommen, müssen nur einmal vorgesehen werden (ob dies tatsächlich eine weitere Aufwandsverringerung bedeutet, hängt von der jeweiligen Technologie ab).

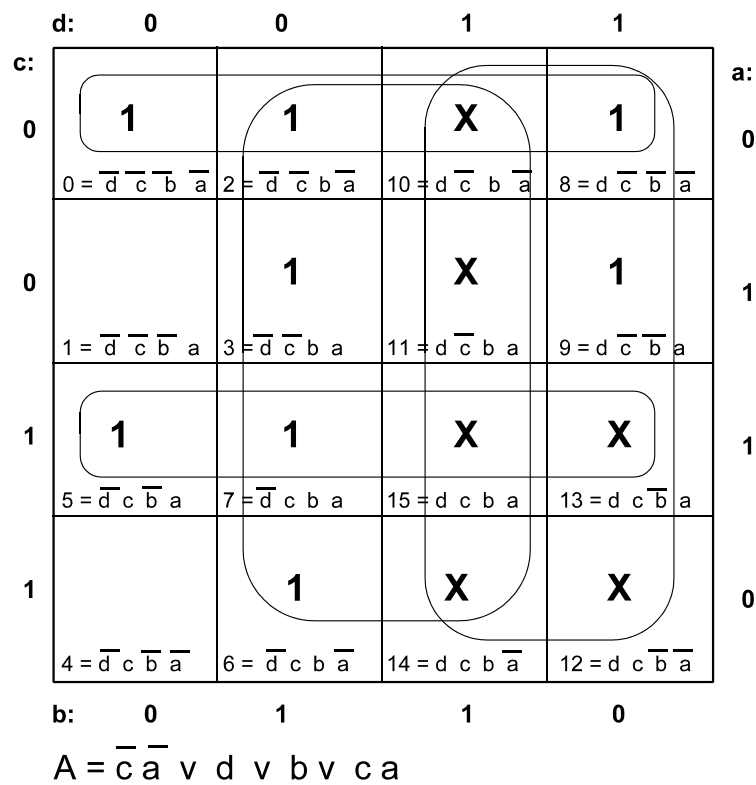


Abb. 7.9 Karnaugh-Plan für Segment A

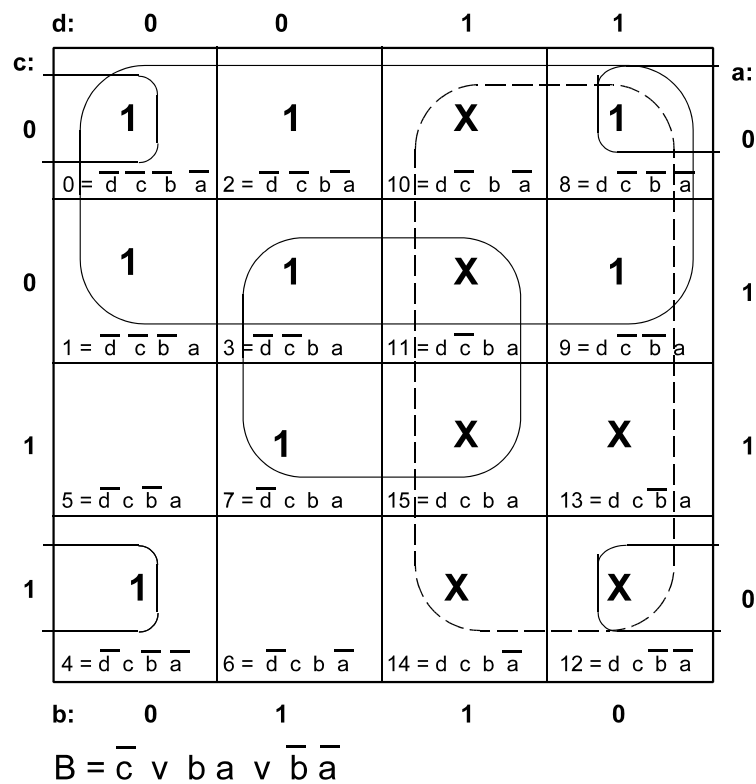


Abb. 7.10 Karnaugh-Plan für Segment B. Der gestrichelt dargestellte Block umfaßt nur Einsen, die bereits in anderen Blöcken enthalten sind. Er wird deshalb nicht berücksichtigt

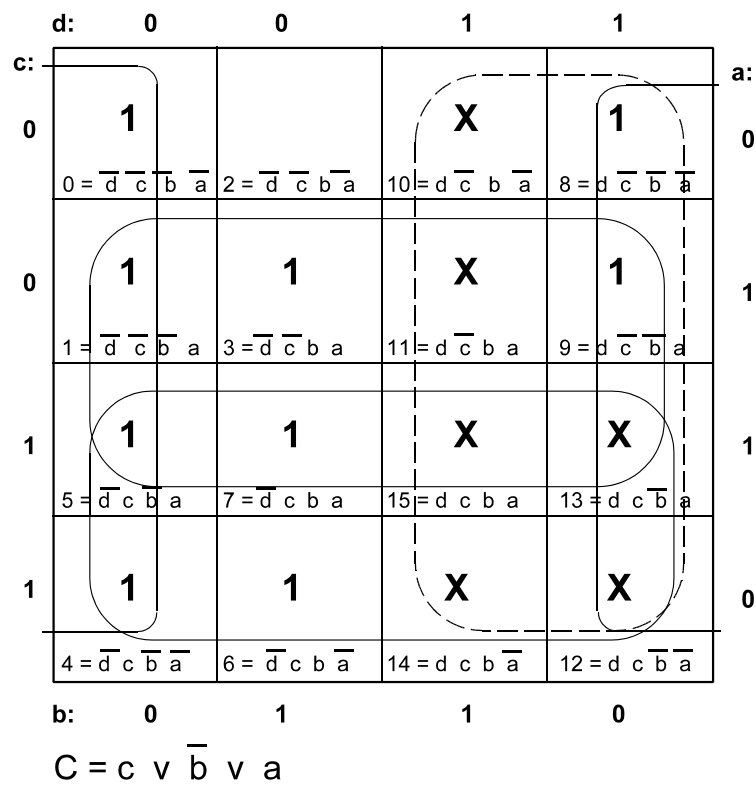


Abb. 7.11 Karnaugh-Plan für Segment C. Der gestrichelt dargestellte Block umfaßt nur Einsen, die bereits in anderen Blöcken enthalten sind. Er wird deshalb nicht berücksichtigt

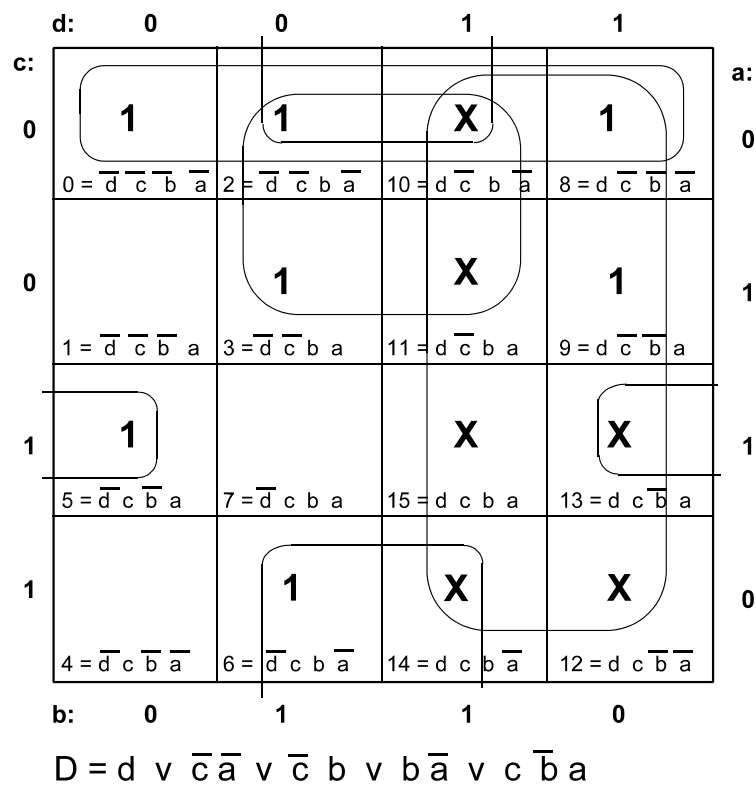


Abb. 7.12 Karnaugh-Plan für Segment D

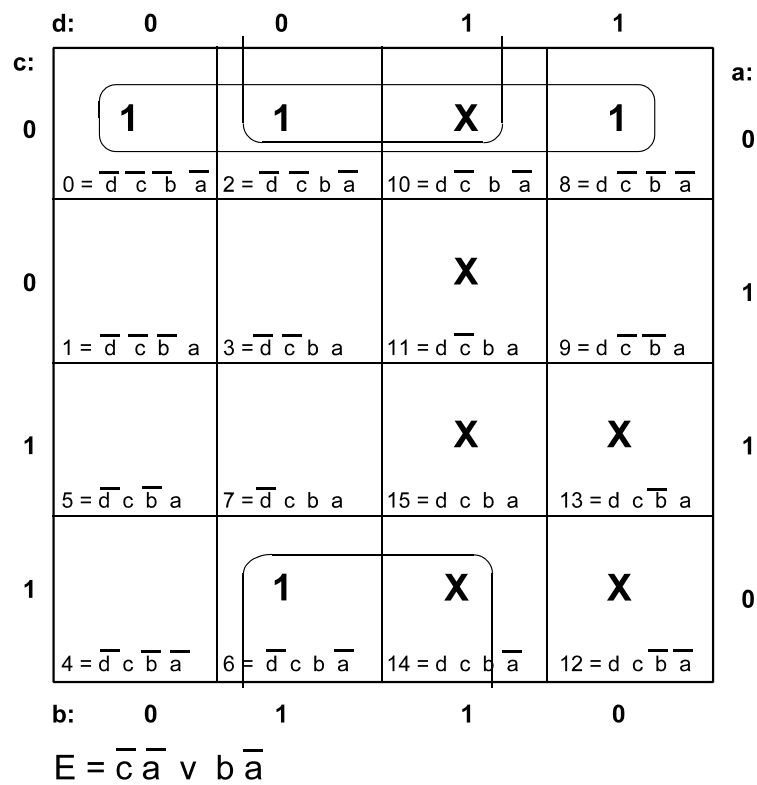


Abb. 7.13 Karnaugh-Plan für Segment E

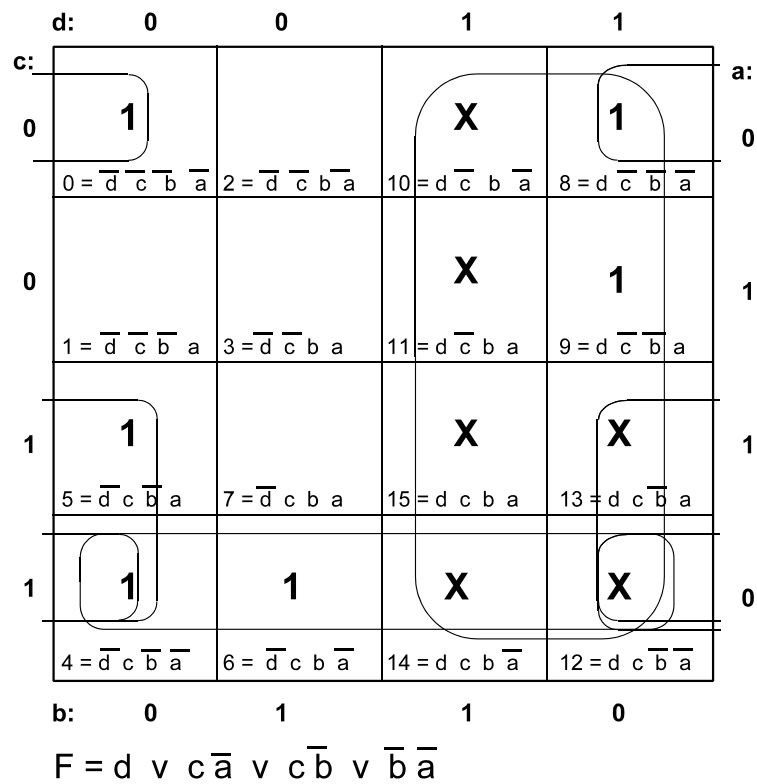


Abb. 7.14 Karnaugh-Plan für Segment F

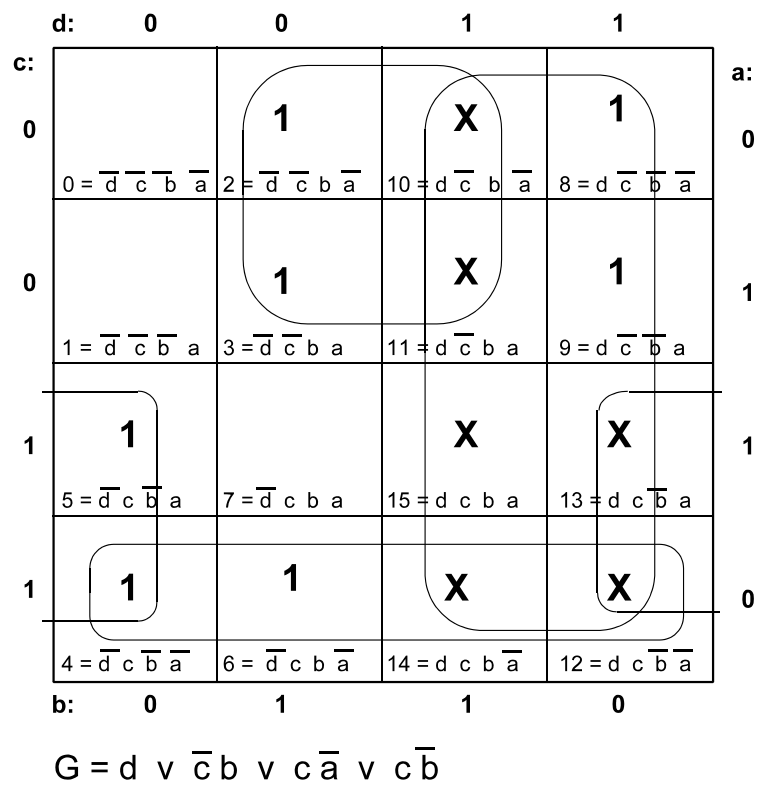


Abb. 7.15 Karnaugh-Plan für Segment G

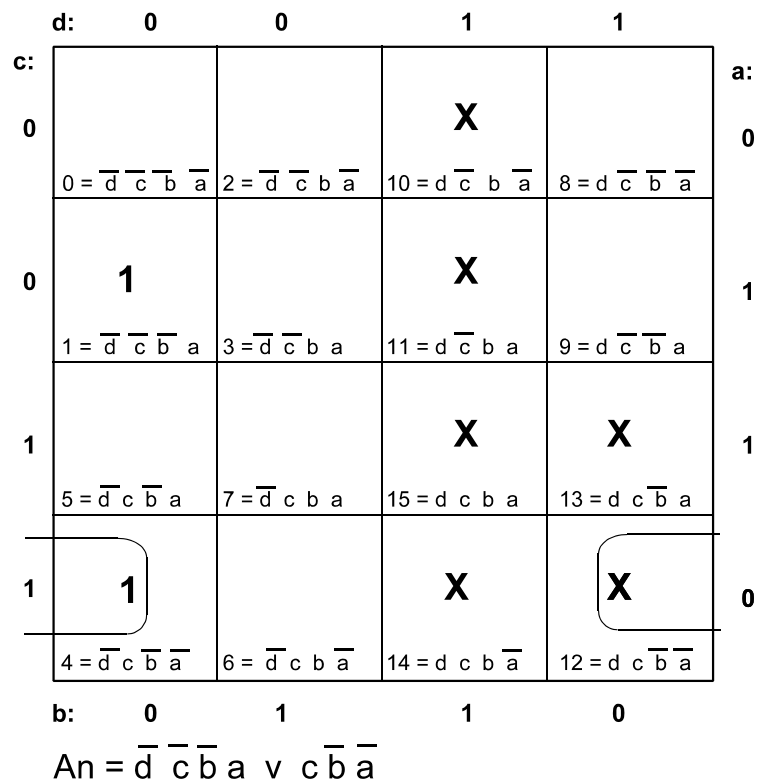


Abb. 7.16 Karnaugh-Plan für Segment A. Invertierte Ansteuerung

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

$B_n = c b \bar{a} \vee c \bar{b} a$

Abb. 7.17 Karnaugh-Plan für Segment B. Invertierte Ansteuerung

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

$C_n = \bar{c} b \bar{a}$

Abb. 7.18 Karnaugh-Plan für Segment C. Invertierte Ansteuerung

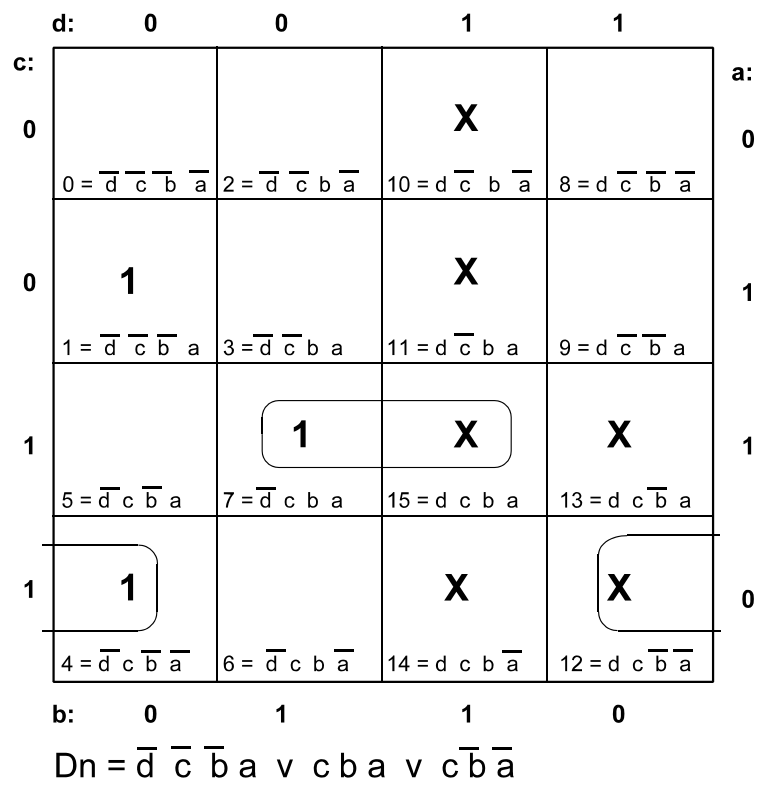


Abb. 7.19 Karnaugh-Plan für Segment D. Invertierte Ansteuerung

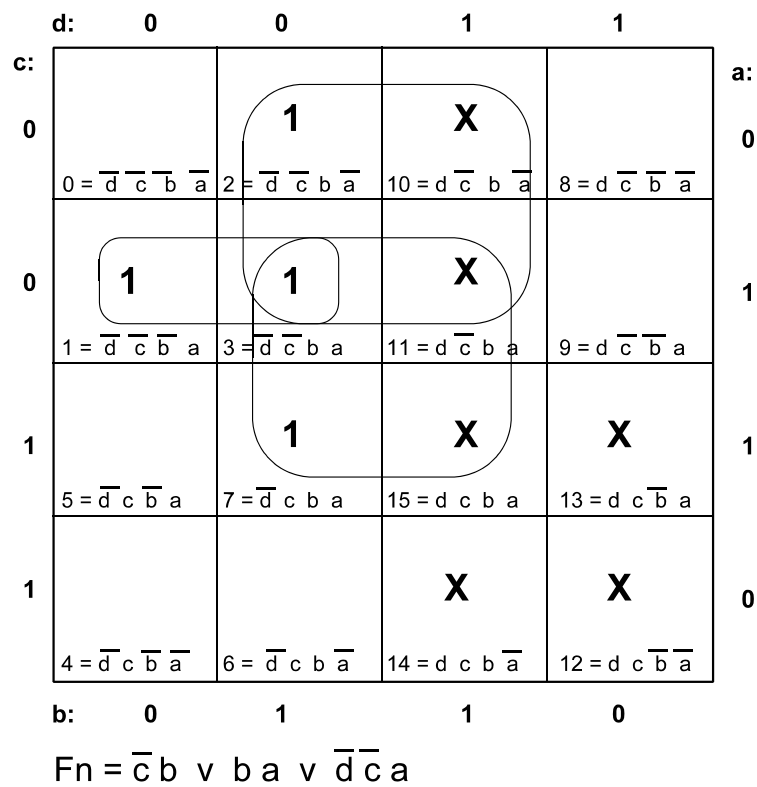


Abb. 7.20 Karnaugh-Plan für Segment F. Invertierte Ansteuerung

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

$G_n = \bar{d} \bar{c} \bar{b} \vee c b a$

Abb. 7.21 Karnaugh-Plan für Segment G. Invertierte Ansteuerung

Schaltfunktionen des Siebensegmentdecoders

a) direkt

$$\begin{aligned}
 A &= \bar{c} \bar{a} \vee d \vee b \vee c a \\
 B &= \bar{c} \vee b a \vee \bar{b} \bar{a} \\
 C &= c \vee \bar{b} \vee a \\
 D &= d \vee \bar{c} \bar{a} \vee \bar{c} b \vee b \bar{a} \vee c \bar{b} a \\
 E &= \bar{c} \bar{a} \vee b \bar{a} \\
 F &= d \vee c \bar{a} \vee c \bar{b} \vee \bar{b} \bar{a} \\
 G &= d \vee \bar{c} b \vee c \bar{a} \vee c \bar{b}
 \end{aligned}$$

b) invertiert

$$\begin{aligned}
 A_n &= \bar{d} \bar{c} \bar{b} a \vee c \bar{b} \bar{a} \\
 B_n &= c b \bar{a} \vee c \bar{b} a \\
 C_n &= \bar{c} b \bar{a} \\
 D_n &= \bar{d} \bar{c} \bar{b} a \vee c b a \vee c \bar{b} \bar{a} \\
 E &: \text{wie direkte Implementierung} \\
 F_n &= \bar{c} b \vee b a \vee \bar{d} \bar{c} a \\
 G_n &= \bar{d} \bar{c} \bar{b} \vee c b a
 \end{aligned}$$

Abb. 7.22 Die Schaltfunktionen im Überblick

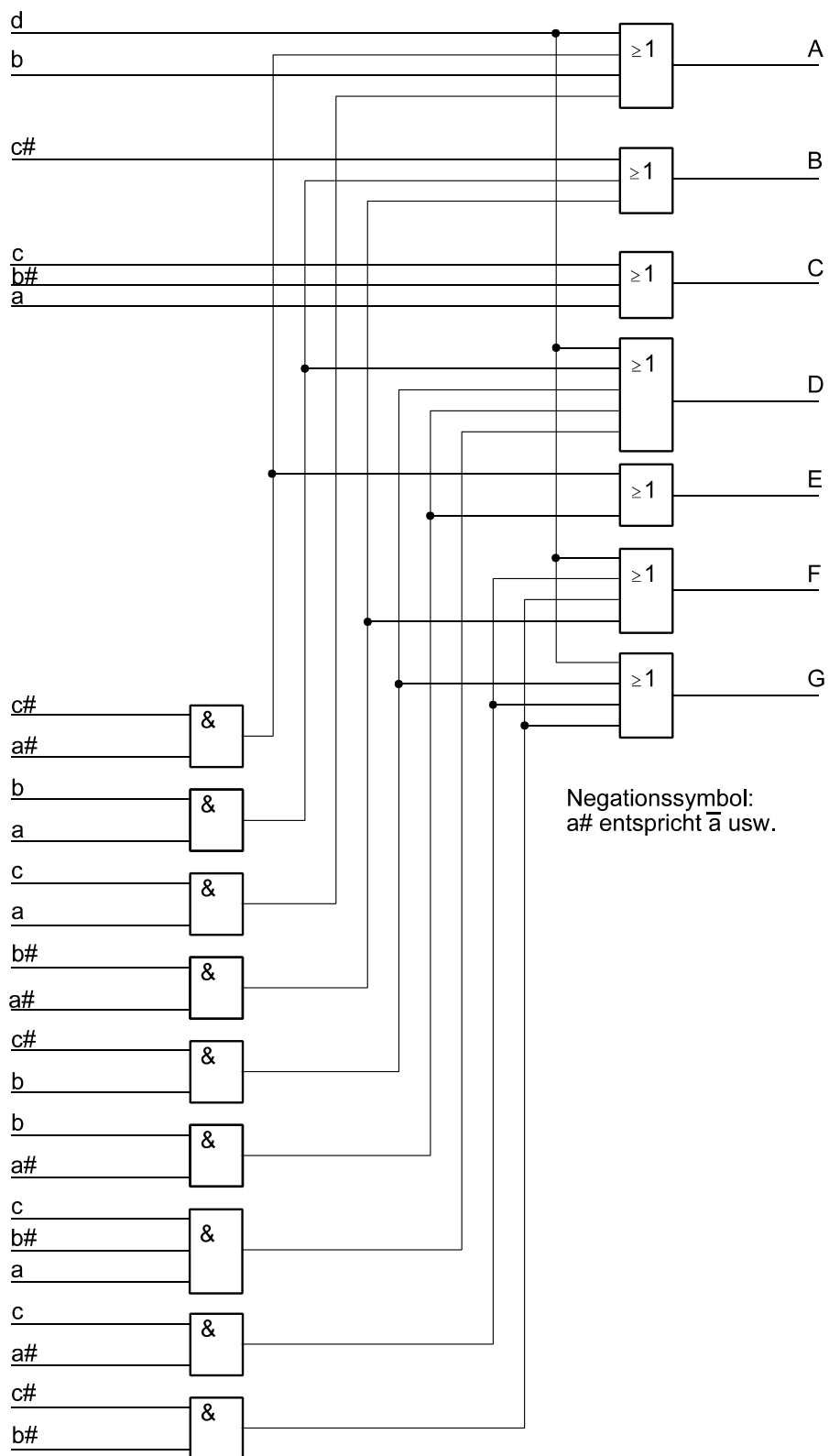


Abb. 7.23 Siebensegmentdecoder (1). Direkte Implementierung

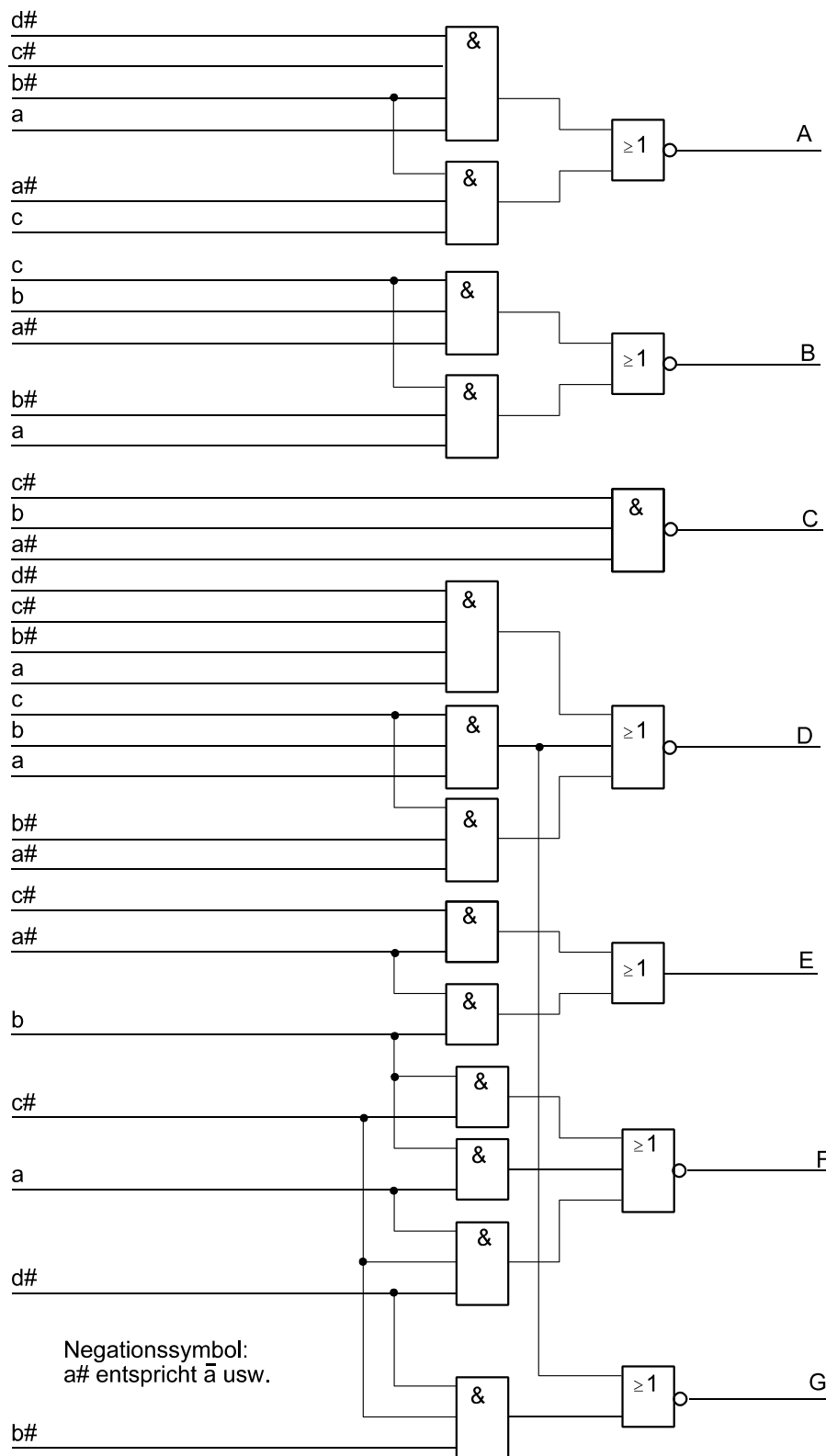


Abb. 7.24 Siebensegmentdecoder (2). Invertierte Ansteuerung (nur Segment E direkt)

Entwurf	ganz naiv; Tabelle 7.3, Abb. 7.7	invertiert (intuitiv); Abb. 7.8	direkt (K-Plan); Abb. 7.23	invertiert (K-Plan); Abb. 7.24
ODER-Verknüpfungen	7	7 ^{*)}	7	6
ODER-Eingänge, insgesamt	49	19	25	14
UND-Verknüpfungen	10	9	9	14
UND-Eingänge, insgesamt	40 ^{**)}	36 ^{**)}	19	40
Negationen	4	4	3	4
Gatteranzahl, insgesamt ^{***)}	17	16	16	20
Gatter-Eingänge, insgesamt	89	55	54	54

*) : einschließlich Inverter für Segment C; **) : BCD-zu-Dezimal-Decoder nicht optimiert (10 bzw. 9 UND-Gatter mit vier Eingängen); ***) : ohne Negatoren für die Eingangssignale.

Tabelle 7.4 War es die Mühe wert? - Optimierungsergebnisse im Vergleich

Was die Gatteranzahl betrifft, so nehmen sich die verschiedenen Entwürfe nicht viel – da kann sogar die volkstümliche Lösung gemäß Abb. 7.7 noch gut mithalten. Beträchtliche Unterschiede gibt es hingegen bei der Gesamtzahl der Gatter-Eingänge. In dieser Hinsicht ist die intuitiv gefundene Lösung gemäß Abb. 7.8 kaum schlechter als die mühsam mittels Karnaugh-Plan optimierte Schaltung von Abb. 7.23. Was anfänglich nicht zu erwarten war: die mit KV-Diagramm optimierte invertierte Ansteuerung ist sogar ungünstiger als die direkte (4 Gatter mehr bei insgesamt gleicher Anzahl an Eingängen). (Die ursprünglichen Schaltfunktionen sind zwar einfacher, es ergeben sich jedoch weniger Gelegenheiten zum Zusammenfassen.)

Praxistip:

Mitdenken lohnt sich – soll heißen: vor dem Rechnen erst einmal aus dem Problemverständnis heraus nach naheliegenden Vereinfachungen suchen. Typische Prüfpunkte:

- ist die invertierte Funktion womöglich günstiger? Das lohnt sich gelegentlich auch dann, wenn mit Entwurfssoftware gearbeitet wird – nicht alle Programme prüfen solche Alternativen durch, und manchmal kann die Software gar nicht wissen, daß es eine derartige Möglichkeit gibt.
- welche Belegungen kommen nie vor, können also als Don't Cares angesetzt werden?
- kann man die Funktion in (einfachere) Teilfunktionen zerlegen, die nur noch die auf einfache Weise miteinander verknüpft werden müssen?
- werden Teilfunktionen bereits anderweitig verwendet?
- gibt es andere Funktionen, die nur noch ergänzt werden müssen (z. B. durch Hinzufügen weiterer Produktterme)?