# Minimierung mittels KV-Diagramm (Karnaugh-Plan)

**Grundlagen der Schaltungsminimierung**

Die Schaltfunktion ist als disjunktive oder konjunktive Normalform gegeben. Unterscheiden sich zwei Terme nur ein einer Variablenposition, so können beide Terme durch einen einzigen Term ersetzt werden, in dem die betreffende Variable fehlt: Wenn $abc \vee ab\bar{c} = 1$, dann muß $ab = 1$ sein, und es ist offensichtlich gleichgültig, welchen Wert c hat. Also kann c weggelassen werden:

$$abc \vee ab\bar{c} = ab$$

Das gilt sinngemäß dann, wenn in vier ansonsten gleichen Termen (Implikanden) zwei Variable in allen Kombinationen vorkommen, in acht ansonsten gleichen Termen drei Variable usw. (allgemein k Variabe in $2^k$ ansonsten gleichen Termen). Beispiel:

$$ab\bar{c}\bar{d} \vee ab\bar{c}d \vee abc\bar{d} \vee abcd = ab$$

Diese Verkürzung entspricht der Wandlung von der binären zur ternären Belegungsliste, das Weglassen einer Variablen entspricht dem Einfügen eines Strichelements. Es lassen sich offensichtlich nur Belegungen zusammenfassen, deren zugeordnete Punkte im Booleschen Raum benachbart sind, also einen Abstand von Eins haben. Zwei benachbarte Punkte bilden einen eindimensionalen Unterraum im Raum $B^n$. Eine Elementarkonjunktion oder der zugehörige binäre Eintrag der Belegungsliste wählen einen einzelnen Punkt im Raum $B^n$ aus. Ein um eine Variable verkürzter Minterm oder ein ternärer Eintrag der Belegungsliste (mit einem Strichelement in der jeweiligen Variablenposition) sind im Grunde Kurzangaben für zwei benachbarte Raumpunkte, mit anderen Worten, sie beschreiben einen eindimensionalen Unterraum.

Beispiele (Abb. 1):

- Der Term $a\bar{b}\bar{c}$ oder der Belegungslisteneintrag 010 beschreiben den Raumpunkt 010 des Booleschen Raums $B^3$.
- Der Term $a\bar{b}$ oder der Belegungslisteneintrag 01– beschreiben die beiden benachbarten Raumpunkte 010 und 011, die einen eindimensionalen Unterraum bilden.

Sinngemäß verhält es sich, wenn aus vier Termen zwei Variable entfernt werden können. Eine solche Vereinfachung ist dann möglich, wenn vier Punkte des Booleschen Raums $B^n$, die zur Lösungsmenge gehören, untereinander einen Abstand von Eins haben (Abb. 2). Diese Raumpunkte bilden einen Unterraum der Dimension zwei. Beispielsweise beschreibt ein Term $\bar{a}$ oder ein Belegungslisteneintrag 0 – – die vier Raumpunkte 000, 001, 010 und 011.

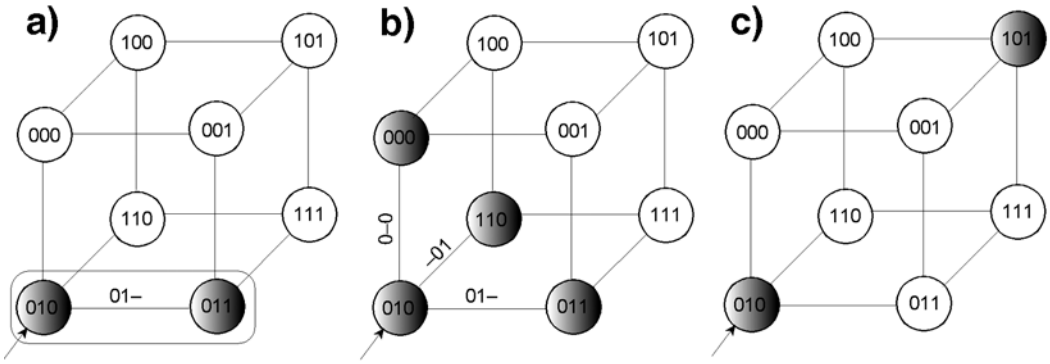



***Abb. 1*** Der Boolesche Raum und die Vereinfachung von Mintermen. Die Darstellung bezieht sich auf den Raumpunkt 010 (Pfeil). Wenn benachbarte Raumpunkte zur Lösungsmenge gehören, ist ein Zusammenfassen möglich. a) gehört beispielsweise der Raumpunkt 011 zur Lösungsmenge, so ergibt sich eine Zusammenfassung 01–. Eine solche ternäre Angabe beschreibt im Grunde einen aus zwei Punkten bestehenden eindimensionalen Unterraum. b) alle Raumpunkte, die vom Raumpunkt 010 den Abstand 1 haben. c) der Raumpunkt 101 hat einen zu großen Abstand und kann somit nicht in die Zusammenfassung einbezogen werden.

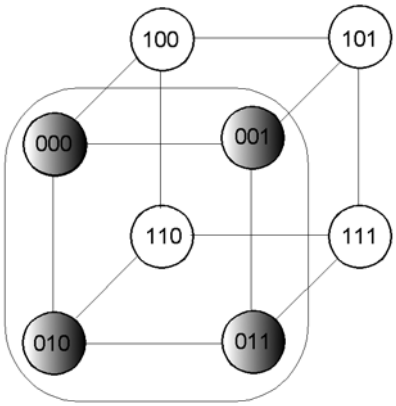



***Abb. 2*** Der Boolesche Raum und weitere Zusammenfassungen. Gehören die Raumpunkte 010, 000, 001 und 011 zur Lösungsmenge, so bilden sie einen zweidimensionalen Unterraum, der sich durch eine Angabe der Form 0 – – beschreiben lässt.

Mit anderen Worten, das Kürzen läuft darauf hinaus, im Booleschen Raum $B^n$ Unterräume von möglichst hoher Dimension aufzufinden, deren Punkte zur Lösungsmenge gehören.

Das KV-Diagramm (Karnaugh-Veitch-Diagramm, Karnaugh-Plan) ist eine zweidimensionale Wahrhertstabelle, die so formatiert ist, daß diese Zusammenfassungsmöglichkeiten visuell erkannt werden können[1]. Hierzu sind die Binärvektoren, die den einzelnen Variablenbelegungen entsprechen, nicht gemäß dem üblichen Binärcode, sondern gemäß dem sog. Gray-Code (Tabelle 1) geordnet. Wichtig ist, daß sich beim Gray-Code von Zählwert zu Zählwert nur jeweils n einziges Bit ändert.

1: Also ohne rechnen zu müssen (“man sieht es sofort...”).

| Zahlen-wert | Binärcode | | | | Gray-Code | | | | Binär-Äqui-valent |
|---|---|---|---|---|---|---|---|---|---|
| | $B_3$ | $B_2$ | $B_1$ | $B_0$ | $G_3$ | $G_2$ | $G_1$ | $G_0$ | |
| 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 1*** Binärcode und Gray-Code.

*Umrechung 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 bezeichnte die jeweils links daneben angeordnete (= höherwertige) Bitposition. Ist n die höchstwertige Bitposition, so wird $B_{n+1} = 0$ angesetzt.

**KV-Diagramme**
Abb. 3 zeigt KV-Diagramme für 2, 3 und 4 Variable.

*Vorgehensweise:*

1. Wahrheitswerte eintragen. Üblicherweise trägt man nur den Wahrheitswert ein, der der jeweiligen Lösungsmenge entspricht. Es sind typischerweise die Einsen. Kommt einem Feld der jeweils andere Wahrheitswert zu, läßt man es frei (Nullen werden nicht eingetragen). Kein Feld vergessen! Fehlt in einem Term eine Variable, so entspricht dies einem Strichelement in der zugehörigen Belegungsliste. Solche Strichelemente sind aufzulösen. 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 $\overline{c}$ Einsen eingetragen werden.

2. Falls möglich, Don't-Care-Bedingungen eintragen (ergeben sich aus dem Anwendungsfall). Sie werden beispielsweise mit einem X bezeichnet, um sie von der eigentlichen Lösungsmenge unterscheiden zu können.

3. Nachsehen, was sich zusammenfassen lässt. Benachbarte Einträge und Don‘t Cares zu möglichst großen Blöcken zusammenfassen. Manchmal sind verschieden Zusammenfassungen möglich. Dann gibt es kein eindeutiges Ergebnis der Minimierung.

4. Ergebnis ablesen. Es entfallen alle Variable, die in einem zusammengefaßten Block sowohl wahr als auch negiert vorkommen. Alle in der jeweiligen Zusammenfassung gleichbleibenden Variablen (nur wahr oder nur negiert) bilden einen Implikanden der Schaltfunktion (DNF: UND-Term, KNF: ODER-Term).

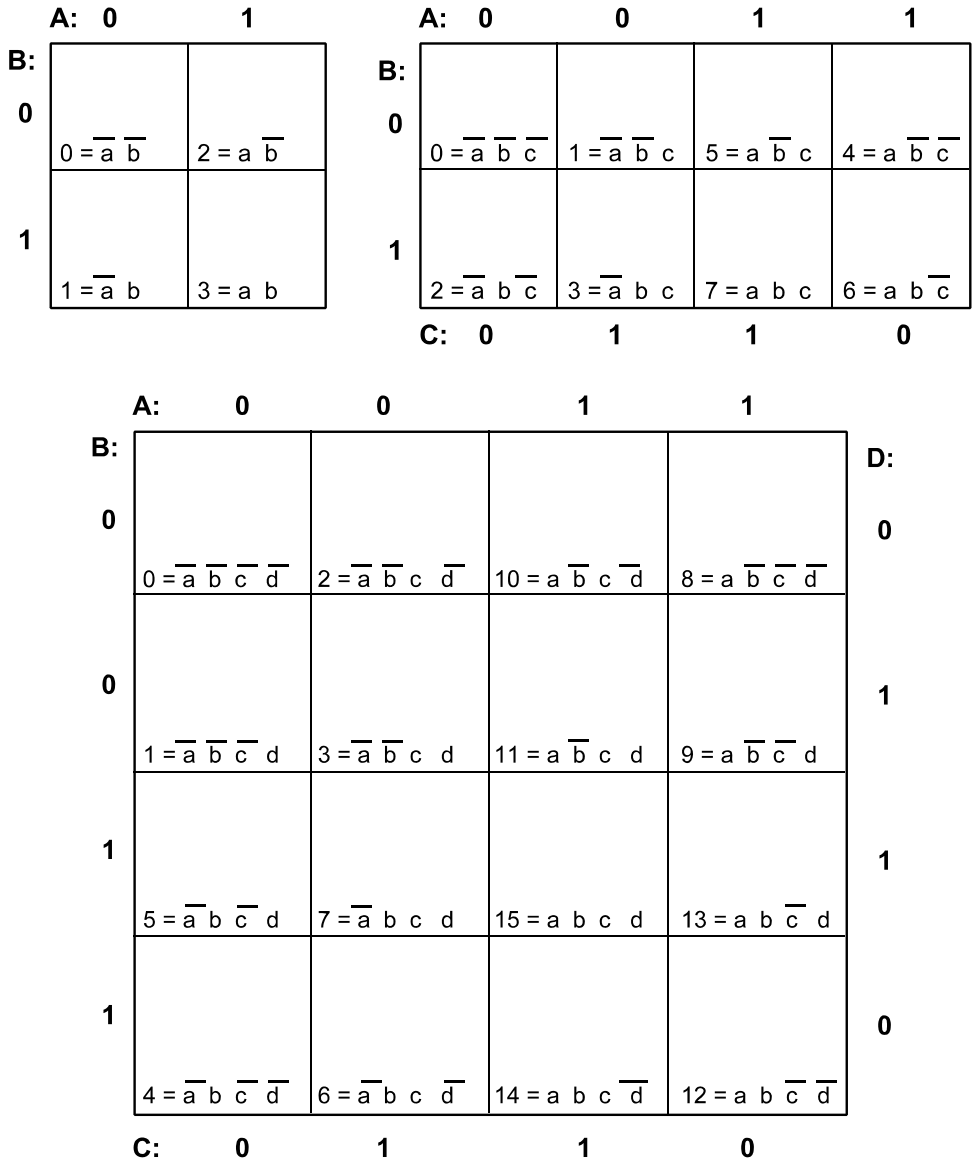



***Abb. 3*** KV-Diagramme für 2, 3 und 4 Variable.

*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 entgegesetzten Rändern angeordnete Einsen (bzw. Don´t Cares) lassen sich zusammenfassen (Wrap Around).

4. Es können nur Belegungen zuammengefaßt werden, die nebeneinander oder übereinander stehen (nur waagerecht und senkrecht vorgehen; nicht diagonal).

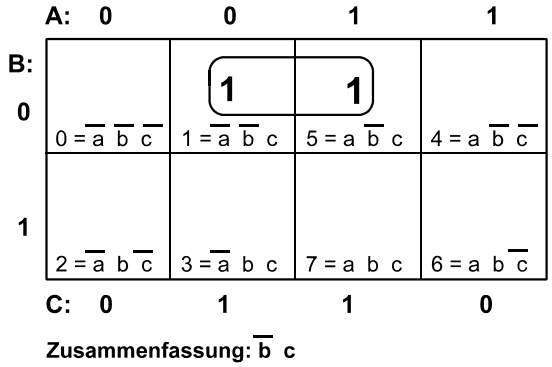
5. Blöcke, die nur Don´t Cares enthalten, werden nicht berücksichtigt.

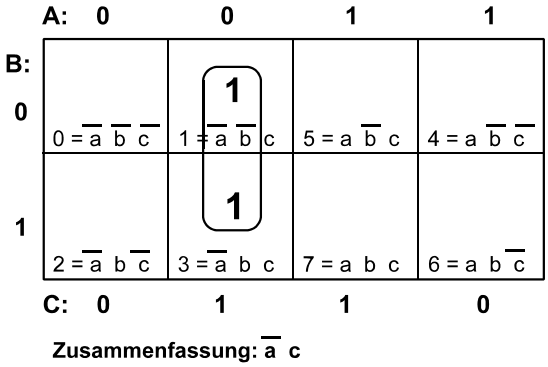
6. Einsen (bzw. Don´t Cares) dürfen in mehreren Blöcken enthalten sein.

7. Das Zusammenfassen wird beendet, wenn alle Einsen berücksichtigt worden sind (auch dann, wenn es weitere Blöcke gibt, die sich zusammenfassen lassen).

*Keine überflüssigen Zusammenfassungen*
Die Minimierung ist beendet, wenn alle irgendwie benachbarten Einsen in möglichst großen Blöcken zusammengefasst wurden. Es ist falsch, darüber hinaus noch alle weiteren möglichen Zusammenfassungen zu berücksichtigen. Prinzip: sind alle Einsen eines derartigen Blockes bereits in anderen Blöcken enthalten, so wird der betreffende Block nicht berücksichtigt. Ansonsten würde sich ein sog. nichtessentieller Primimplikand ergeben. Ein solcher Term ist aus Sicht der Funktionsweise nicht falsch, aber an sich unnötig, weil es bereits andere Terme gibt, die die gleichen Variablenbelegungen erfüllen. Diese Terme sind mit Kürzungsregeln nicht mehr wegzuschaffen.

Die Abb. 4 bis 7 zeigen typische Zusammenfassungen. Anschließend werden die Zusammenfassungsregeln anhand eines Praxisbeispiels (Siebensegmentdecoder) veranschaulicht.



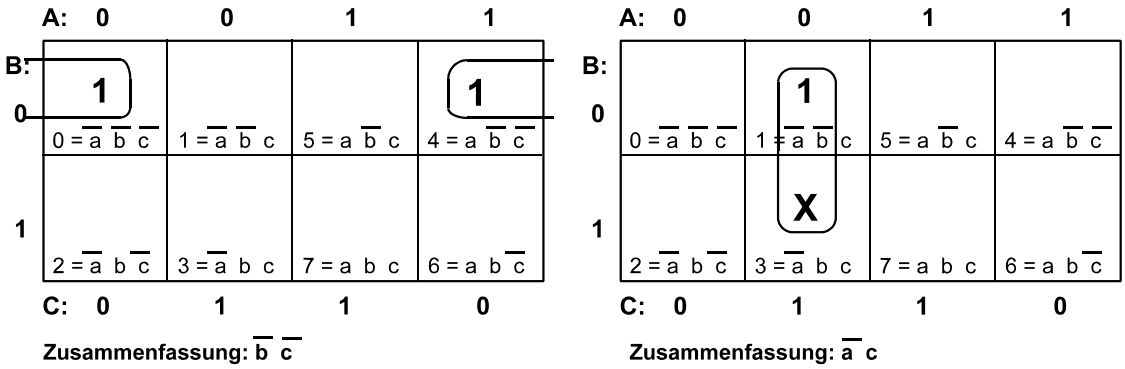
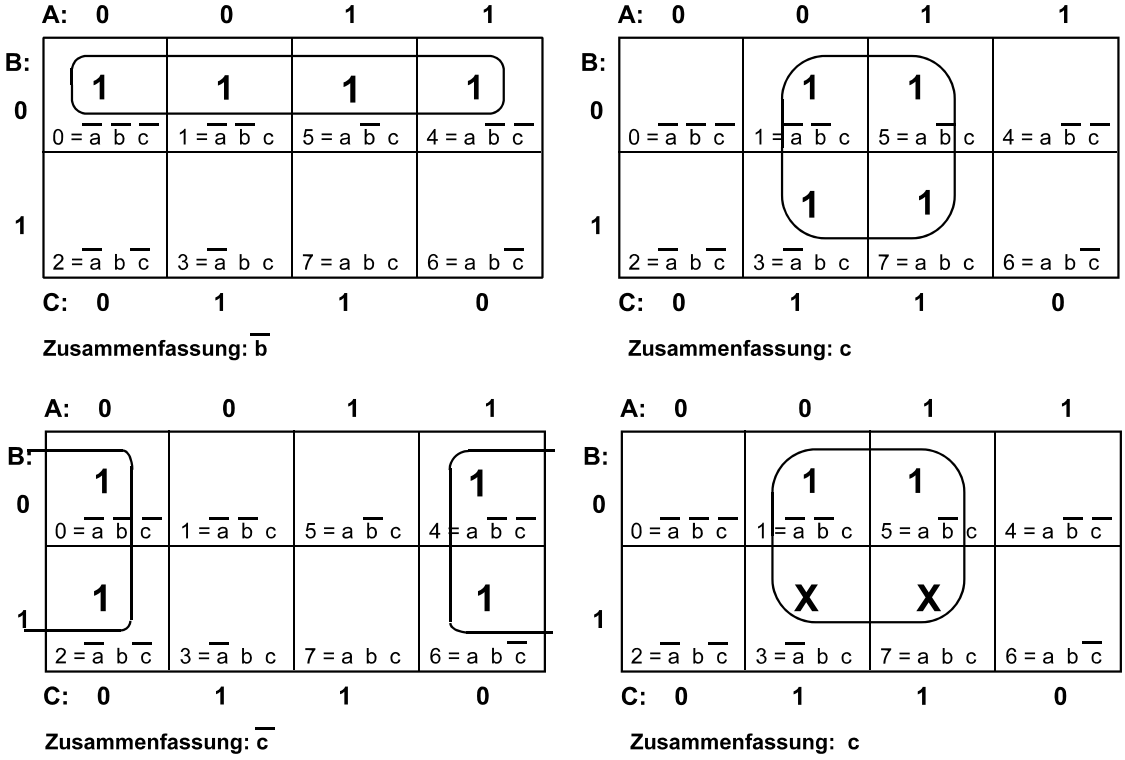



***Abb. 4*** Zusammenfassung von Zweierblöcken (Beispiele).



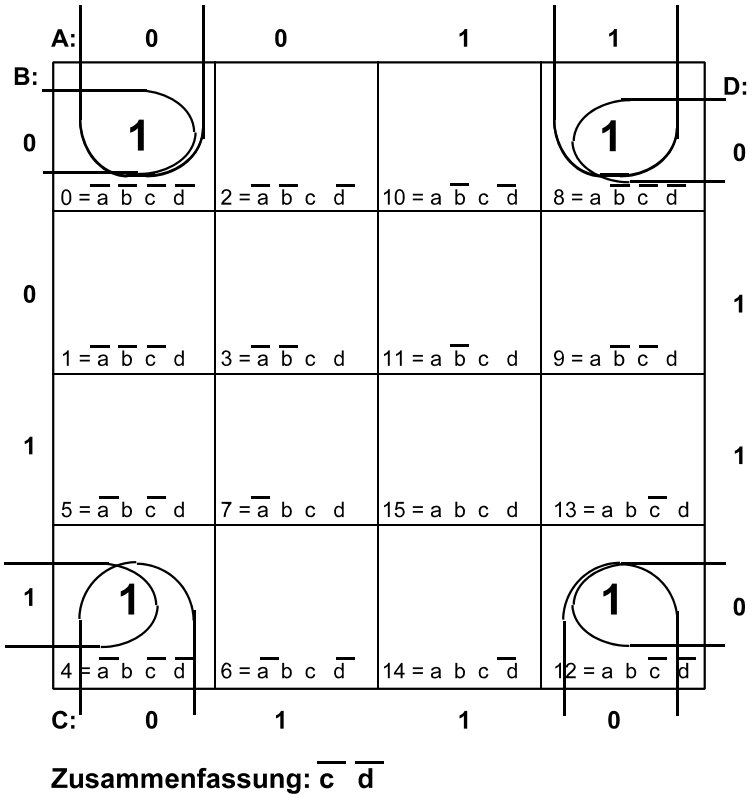
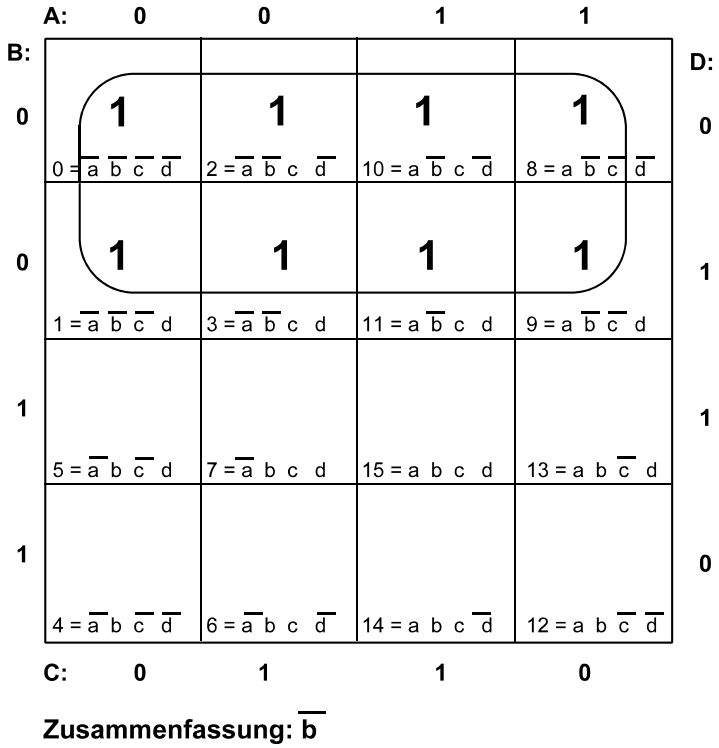



***Abb. 5*** Zusammenfassung von Viererblöcken (Beispiele).



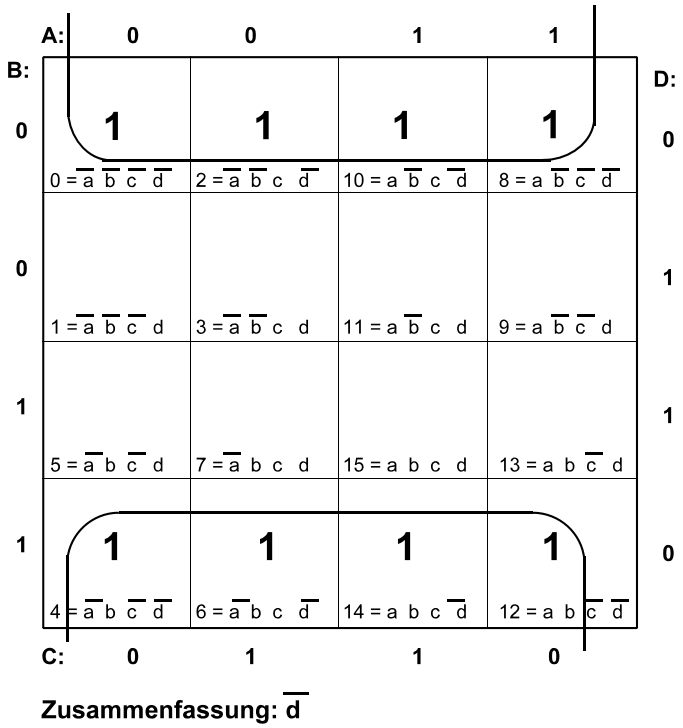
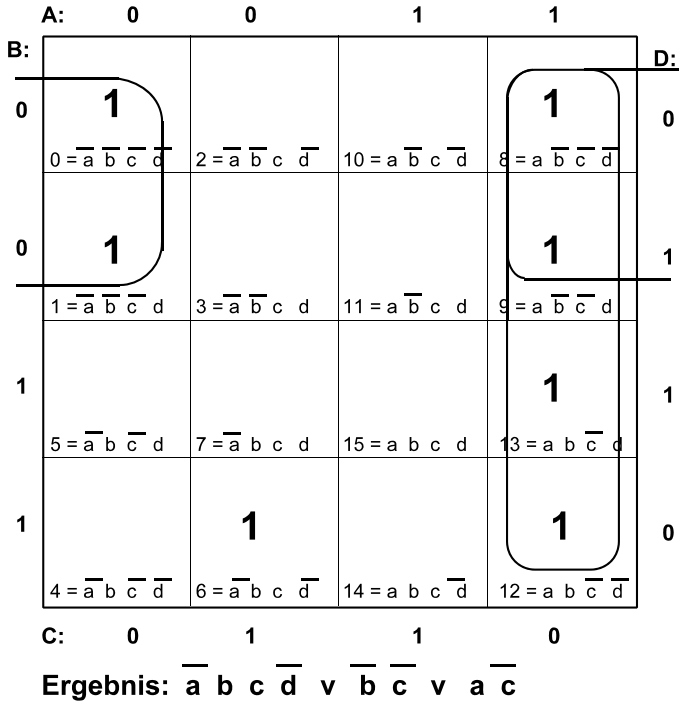



***Abb. 6*** Zusammenfassung von Achterblöcken (Beispiele).




Die ursprüngliche Schaltfunktion:

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

***Abb. 7*** Ein Beispiel.

**Praxisbeispiel: der Siebensegmentdecoder**

Eine im Binärcode (vier Bits mit Werten zwischen 0 (0000) und 9 (1001)) gegebene Dezimalzahl ist auf einer Siebensegmentanzeige darzustellen (Abb. 8, Tabelle 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.

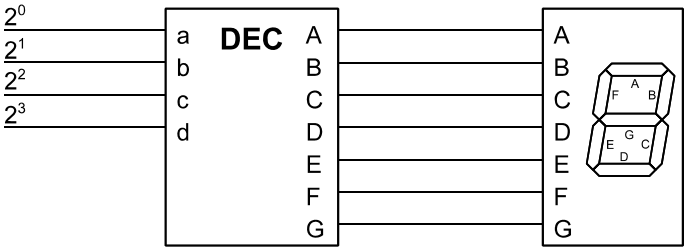



***Abb. 8*** Siebensegmentdecoder.

| 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 2*** Zur Funktionsweise 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 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. 9).

| 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 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.

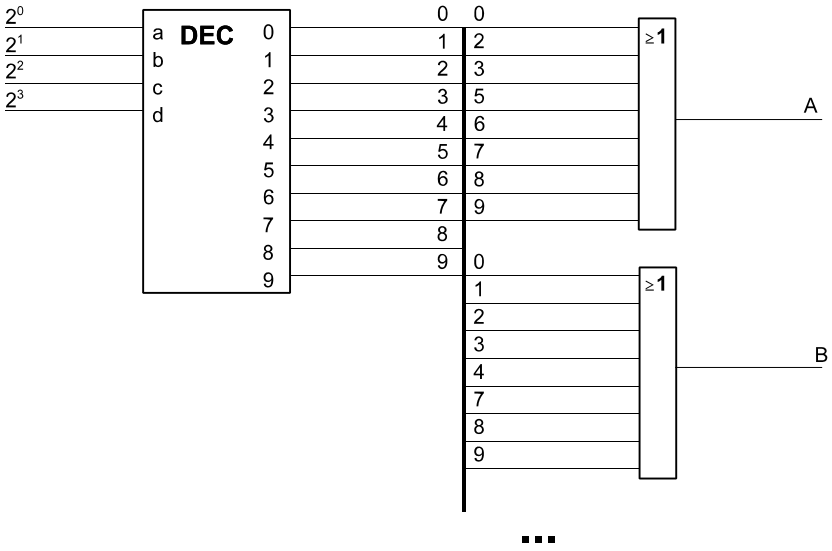



***Abb. 9*** Ganz nach Gefühl entwerfen... Siebensegmentdecoder auf Grundlage eines BCD-zu-Dezimal-Decoders (Schaltungsausschnitt). Es ist nur die Ansteuerung der Segmente A und B dargestellt. Es sind die Signale angeschlossen, die in Tabelle 3 mit einer Eins gekennzeichnet sind.

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. 10). Bei welchen Segmenten sich diese Verfahrensweise lohnt, ergibt sich aus der Mächtigkeit des On-Set (vgl. Tabelle 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.

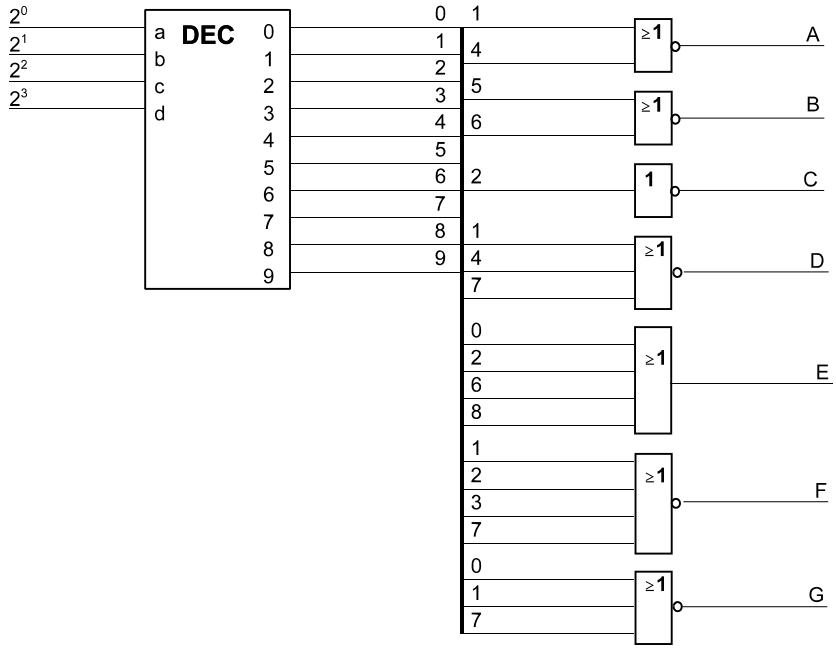



***Abb. 10*** Ein intuitiv entworfener Siebensegmentdecoder.

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 KV-Diagramm entwerfen**

Wir entwerfen eine Schaltung, die – ohne zwischengeordnete Decoderschaltung – die gesamte Informationswandlung BCD zu Siebensegment erledigt. Hierzu ist je Segment ein KV-Diagramm anzulegen und gemäß Tabelle 2 auszufüllen (Abb. 11 bis 17). Die Variablenbelegungen 10 bis 15 werden als Don‘t Cares eingetragen.

*Geht es noch einfacher?*

Aus Tabelle 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. 18 bis 23 zeigen die entsprechenden KV-Diagramme.

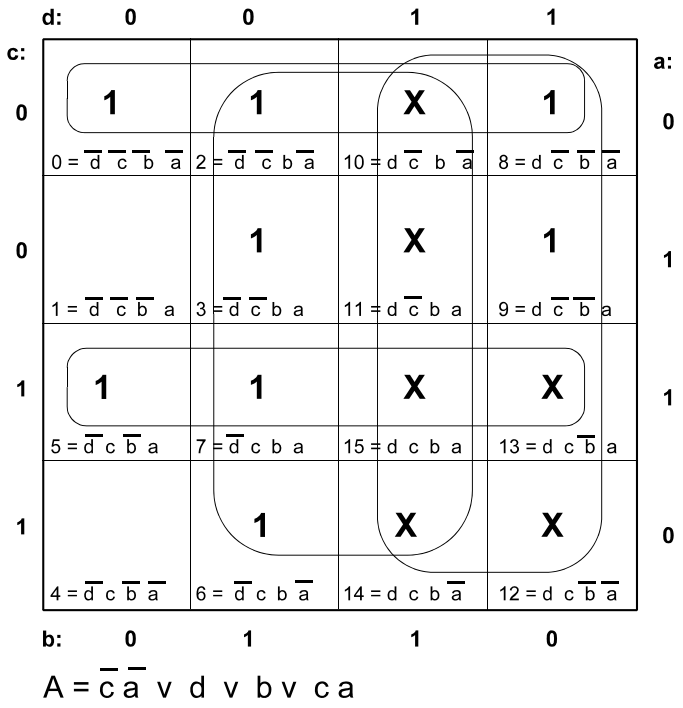



***Abb. 11*** KV-Diagramm für Segment A.

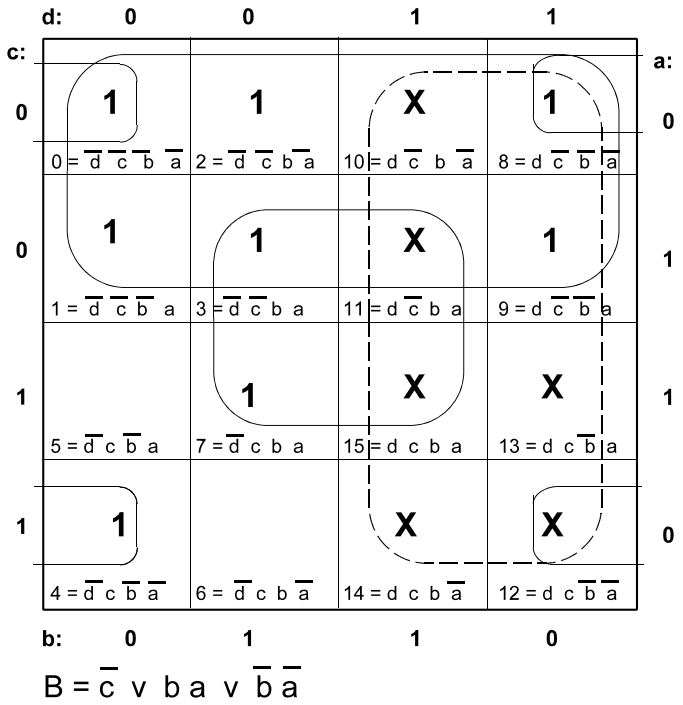



***Abb. 12*** KV-Diagramm für Segment B. Der gestrichelt dargestellte Block umfaßt nur Einsen, die bereits in anderen Blöcken enthalten sind. Er wird deshalb nicht berücksichtigt.

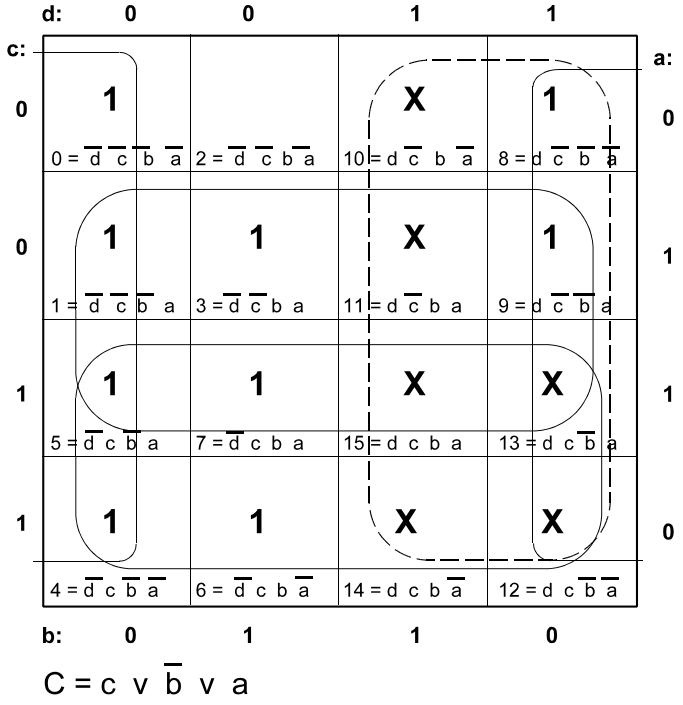



***Abb. 13*** KV-Diagramm für Segment C. Der gestrichelt dargestellte Block umfaßt nur Einsen, die bereits in anderen Blöcken enthalten sind. Er wird deshalb nicht berücksichtigt.

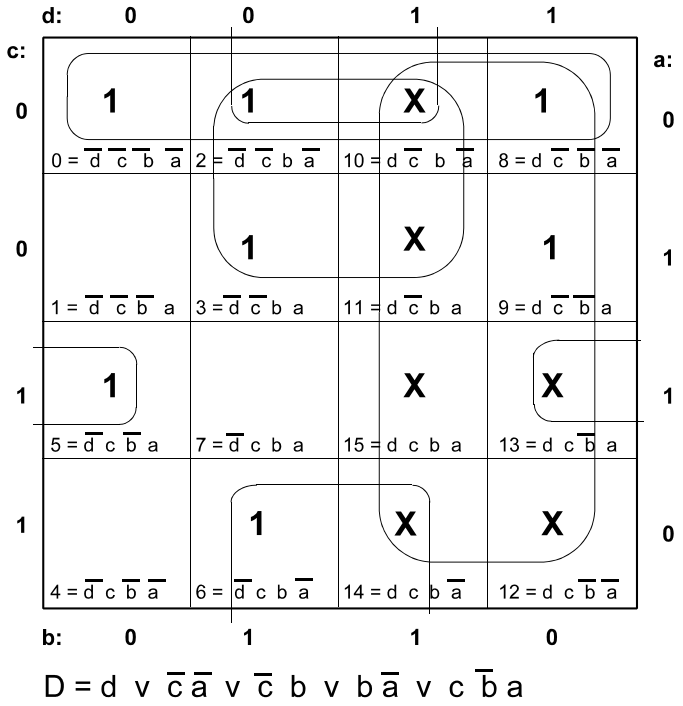



***Abb. 14*** KV-Diagramm für Segment D.

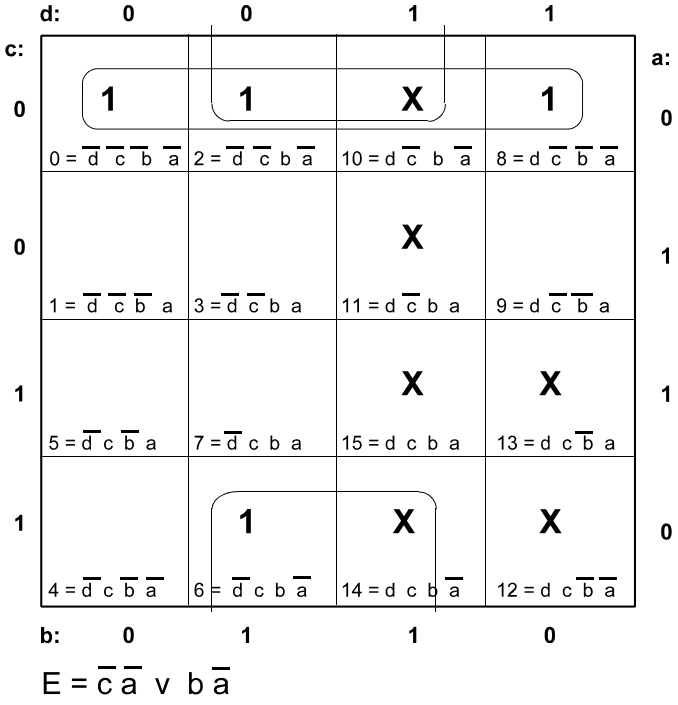



***Abb. 15*** KV-Diagramm für Segment E.

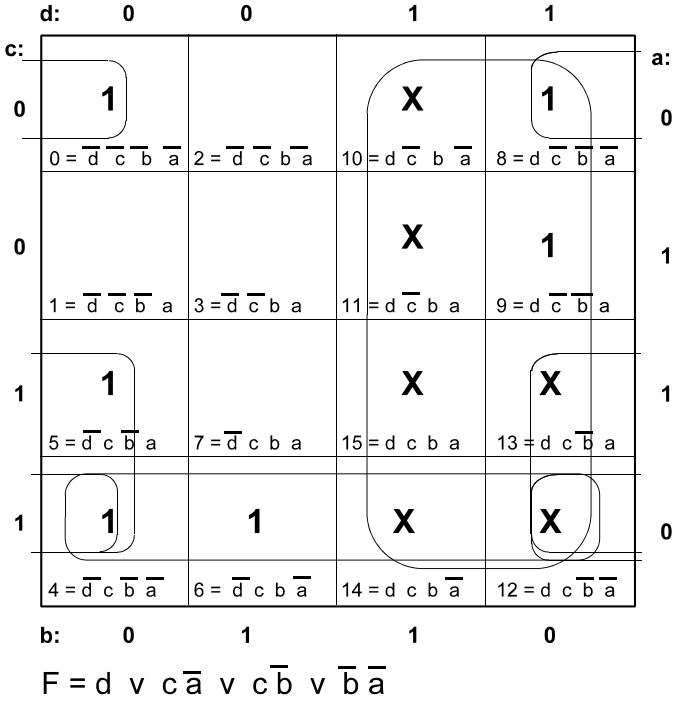



***Abb. 16*** KV-Diagramm für Segment F.

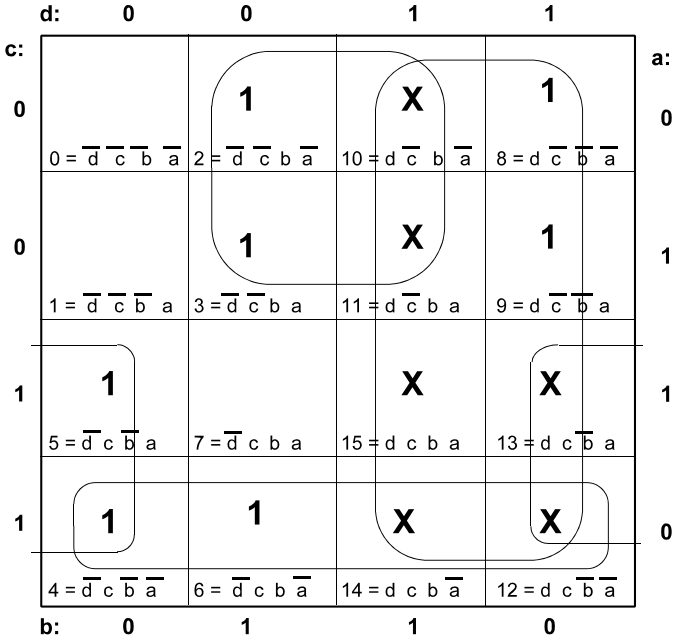



***Abb. 17*** KV-Diagramm für Segment G.

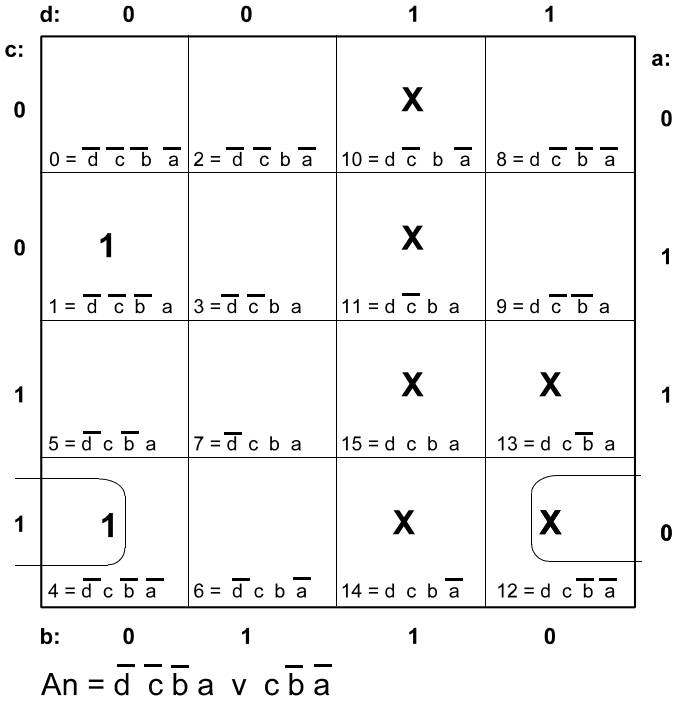



***Abb. 18*** KV-Diagramm für Segment A. Invertierte Ansteuerung.

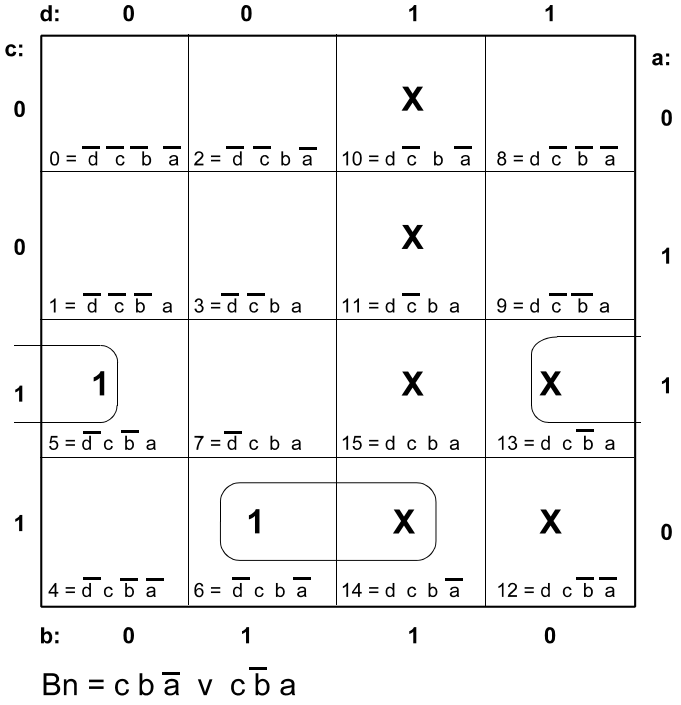



***Abb. 19*** KV-Diagramm für Segment B. Invertierte Ansteuerung.

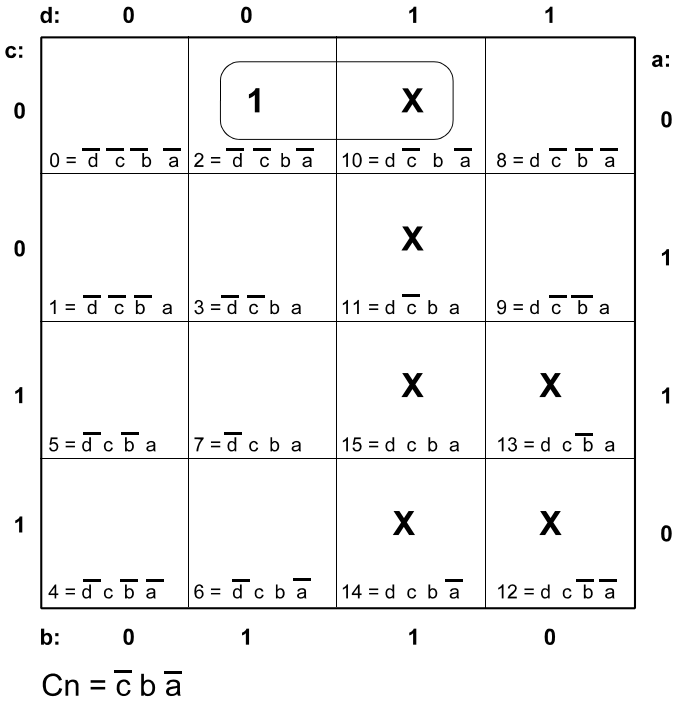



***Abb. 20*** KV-Diagramm für Segment C. Invertierte Ansteuerung.

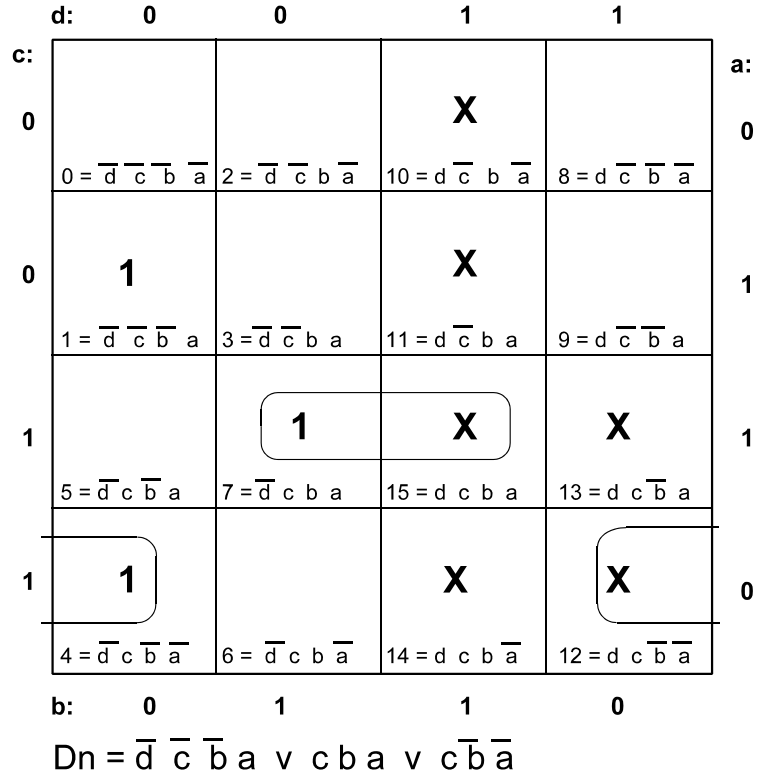



***Abb. 21*** KV-Diagramm für Segment D. Invertierte Ansteuerung.

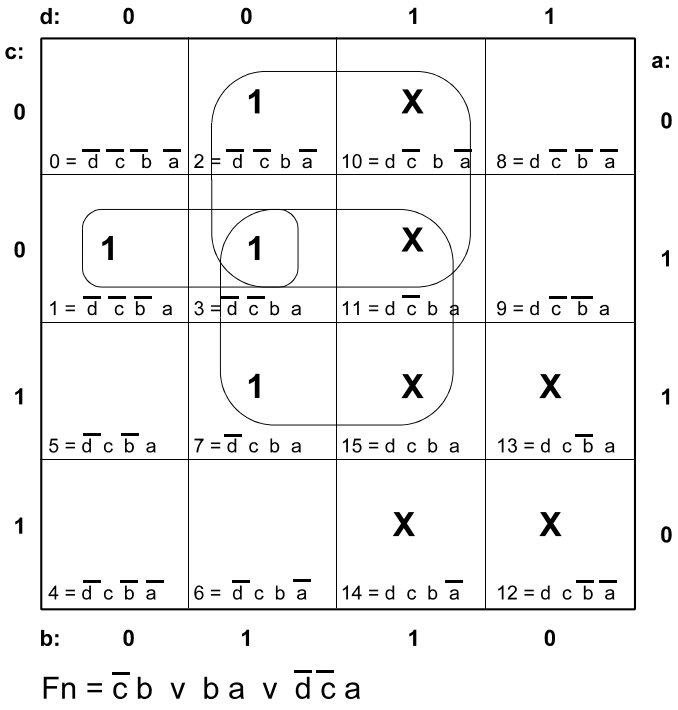



***Abb. 22*** KV-Diagramm für Segment F. Invertierte Ansteuerung.

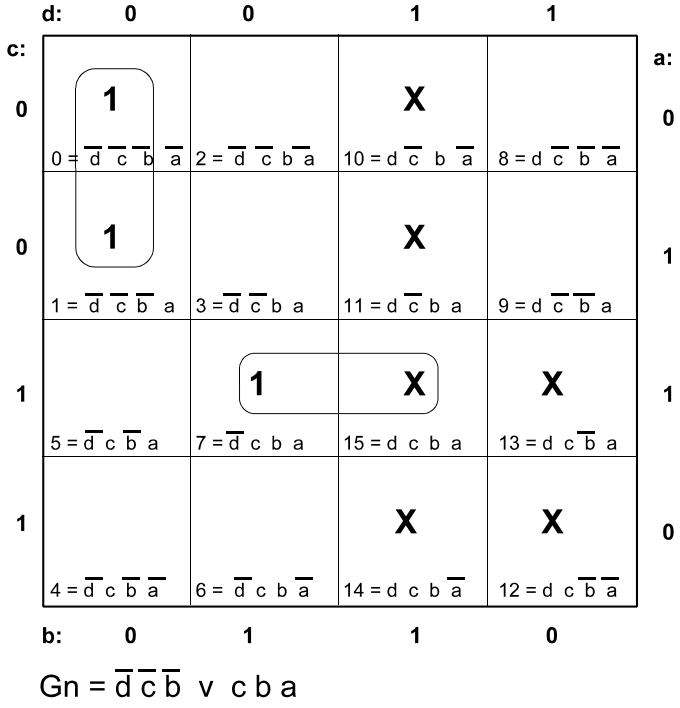



***Abb. 23*** KV-Diagramm für Segment G. Invertierte Ansteuerung.

*Variantenvergleich*

In Abb. 24 sind die Schaltfunktionen der optimierten Varianten zusammengefaßt. Die Abb. 25 und 26 zeigen die entsprechenden Schaltpläne. Tabelle 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).

Schaltfunktionen des Siebensegmentdecoders

| a) direkt | b) invertiert |
|---|---|
| A = c̄ ā v d v b v c a | An = d̄ c̄ b̄ a v c b̄ ā |
| B = c̄ v b a v b̄ ā | Bn = c b ā v c b̄ a |
| C = c v b̄ v a | Cn = c̄ b ā |
| D = d v c̄ ā v c̄ b v b ā v c b̄ a | Dn = d̄ c̄ b̄ a v c b a v c b̄ ā |
| E = c̄ ā v b ā | E: wie direkte Implementierung |
| F = d v c ā v c b̄ v b̄ ā | Fn = c̄ b v b a v d̄ c̄ a |
| G = d v c̄ b v c ā v c b̄ | Gn = d̄ c̄ b̄ v c b a |

***Abb. 24*** Die Schaltfunktionen im Überblick.

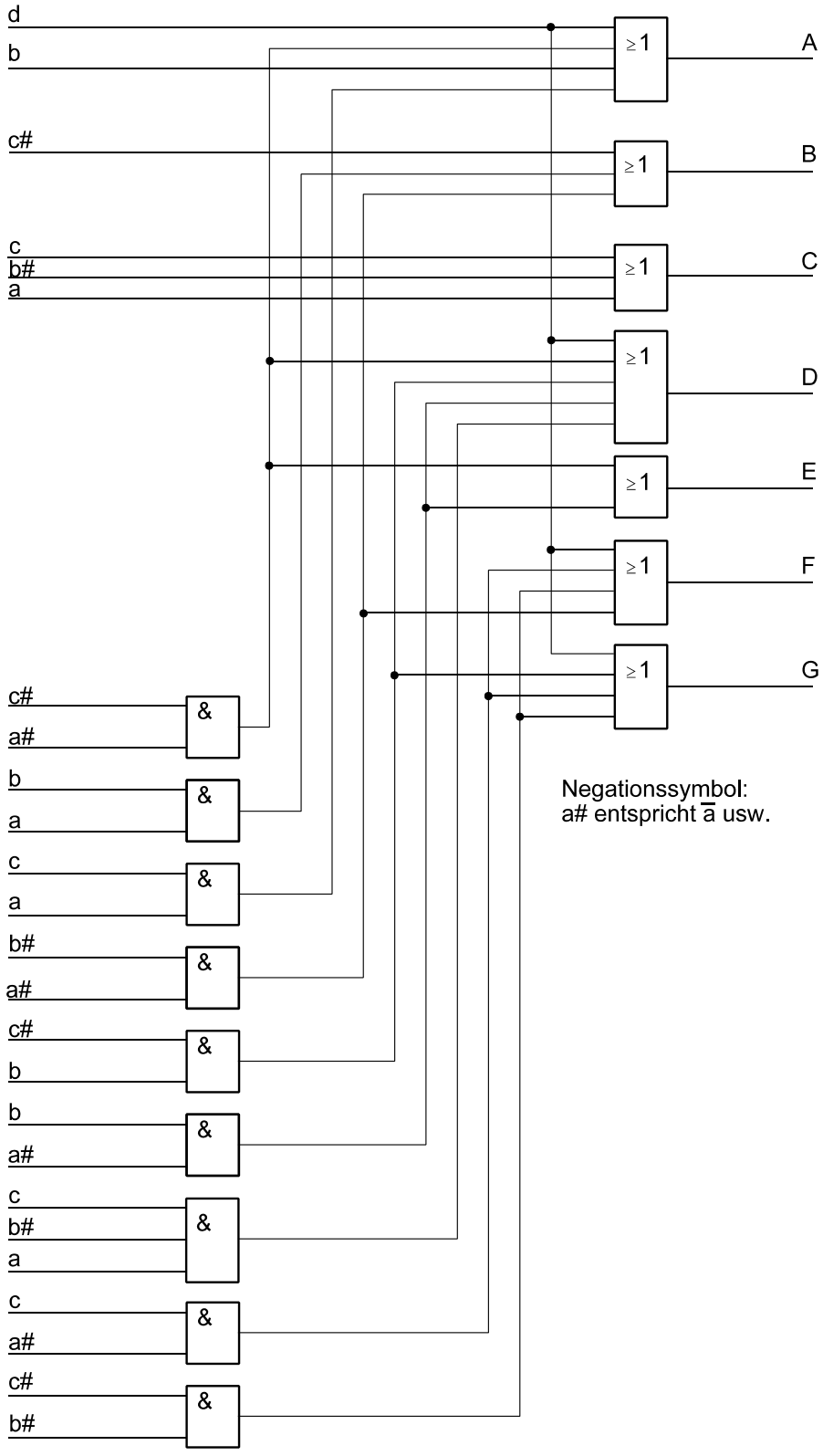



***Abb. 25*** Siebensegmentdecoder (1). Direkte Implementierung.

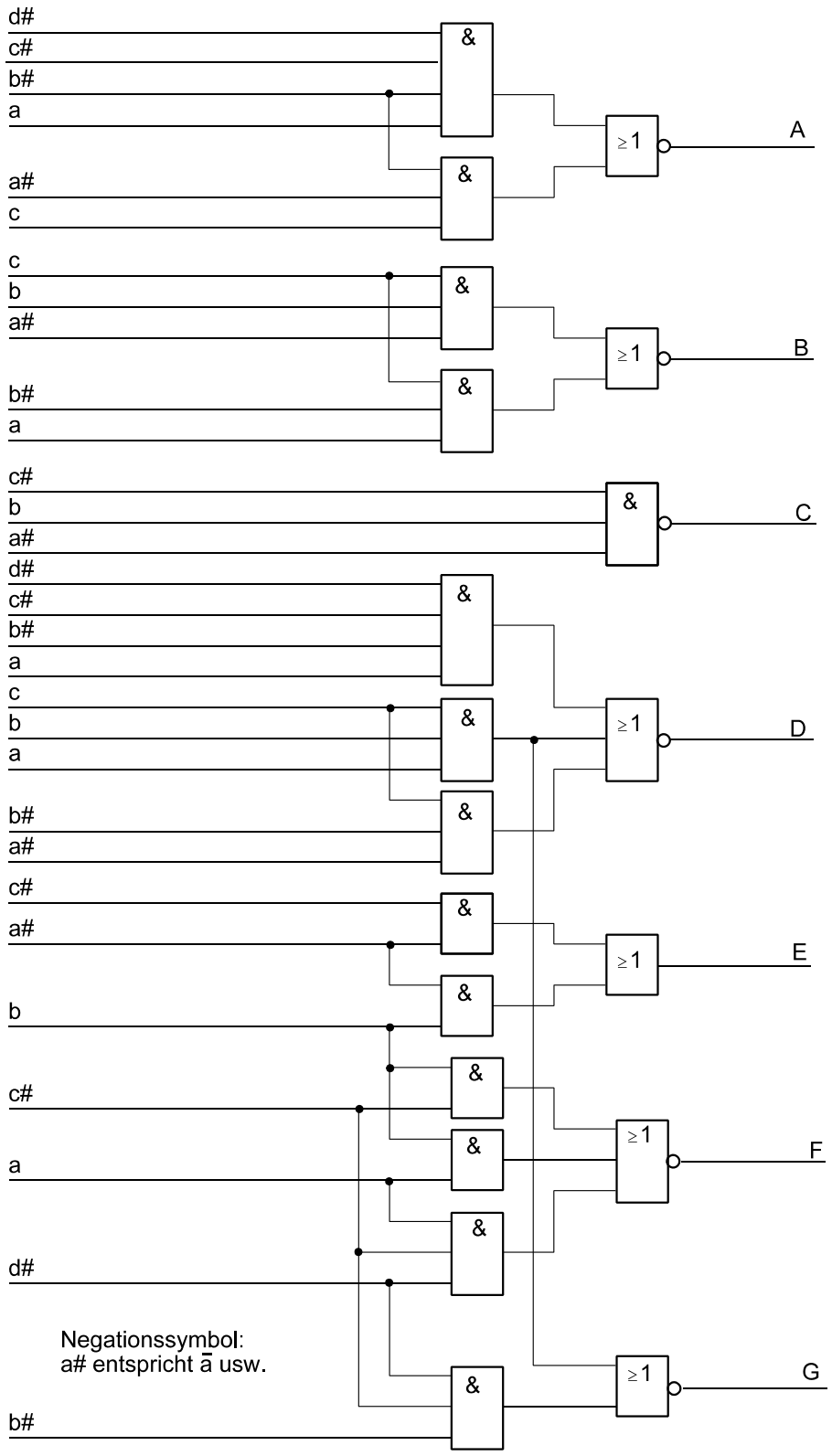



***Abb. 26*** Siebensegmentdecoder (2). Invertierte Ansteuerung (nur Segment E direkt).

| Entwurf | ganz naiv; Tabelle 3, Abb. 9 | invertiert (intuitiv); Abb. 10 | direkt (K-Plan); Abb. 25 | invertiert (K-Plan); Abb. 26 |
|---|---|---|---|---|
| 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 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. 9 noch gut mithalten. Beträchtliche Unterschiede gibt es hingegen bei der Gesamtzahl der Gatter-Eingänge. In dieser Hinsicht ist die intuitiv gefundene Lösung gemäß Abb. 10 kaum schlechter als die mühsam mittels KV-Diagramm optimierte Schaltung von Abb. 25. Was anfänglich nicht zu erwarten war: die mit KV-Dagramm 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)?