4. SCHALTNETZE

4.1 Definition und Entwurf von Schaltnetzen

Nachdem wir gesehen haben, dass es Sinn macht mit digitalen, genauer mit binären Signalen zu arbeiten und dass die Schaltalgebra ein geeignetes formales Mittel zur Beschreibung digitaler Systeme ist, sollen in diesem Kapitel die formalen Methoden zum Entwurf digitaler Systeme beschrieben werden.

Schwerpunkte liegen hierbei auf der Analyse und Synthese digitaler Schaltungen mit Methoden zur Minimierung des Realisierungsaufwands.

Ausgangspunkt dafür ist der Hauptsatz der Schaltalgebra:

> Jede boolesche Funktion $f:\{0,1\}^N \rightarrow \{0,1\}$ lässt sich als Disjunktion von Produkttermen (DNF) und als Konjunktion von Summentermen (KNF) darstellen.

Außerdem werden typische digitale Schaltungen exemplarisch vorgestellt, um die Anwendung der Methoden in der Praxis zu zeigen.

Es geht also einen Schritt weiter in Richtung konkreter Entwurf und Realisierung technischer Systeme.

Ein Schaltnetz (kombinatorische Schaltung, „Combinational logic“, Funktionsbündel) ist eine Anordnung, die digitale Signale derart verarbeitet, dass die Signale an den Ausgängen zu jedem beliebigen Zeitpunkt (unter Vernachlässigung von Übergangs- und Verzögerungszeiten) allein von den Zuständen an den Eingängen abhängen.

D. h., ein Schaltnetz hat kein Gedächtnis.

$$f : \{0,1\}^N \rightarrow \{0,1\}^M$$

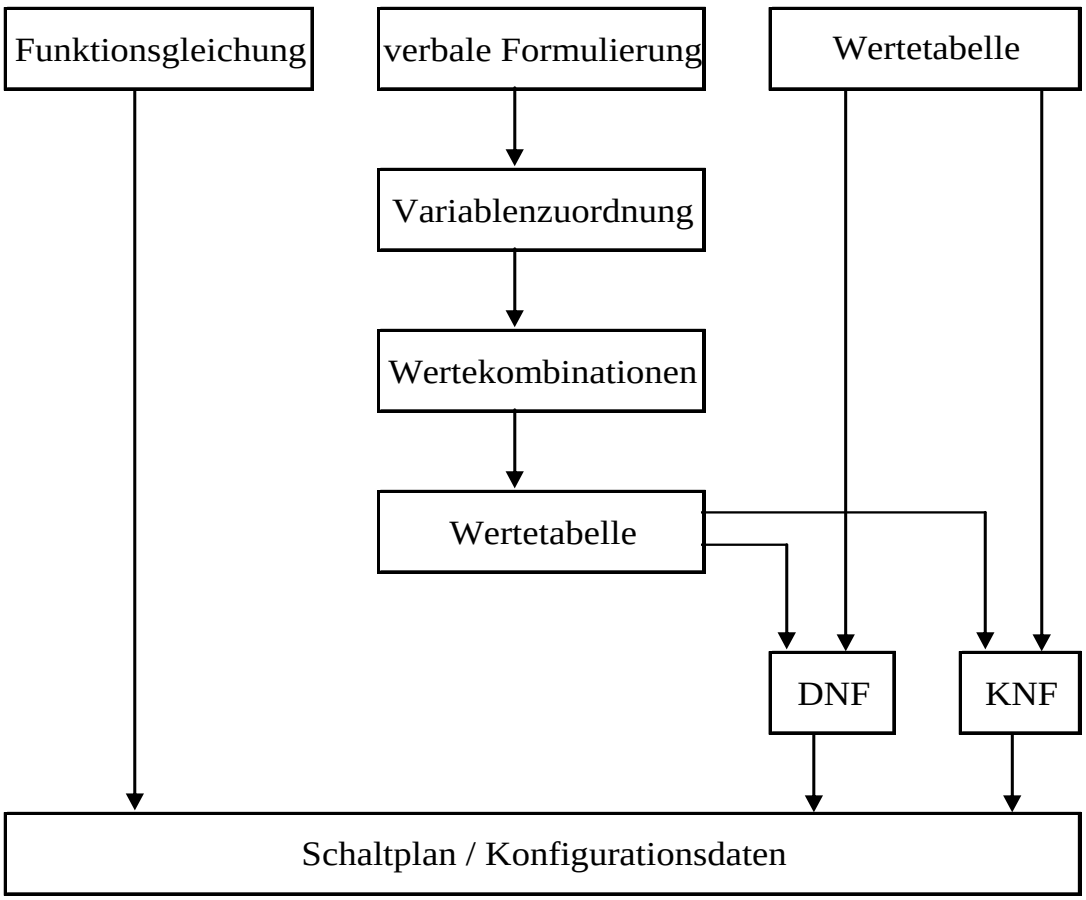
$$z_1 = f_1(x_1, x_2, ..., x_N)$$
$$z_2 = f_2(x_1, x_2, ..., x_N)$$
$$\vdots$$
$$z_M = f_M(x_1, x_2, ..., x_N)$$

f_i: Schaltfunktionen

Vektorschreibweise: $\underline{\mathbf{z}} = \mathbf{f}(\underline{\mathbf{x}})$

Die Realisierung von Schaltnetzen kann z. B. mittels Gattern (z. B. UND, ODER, NICHT) oder programmierbarer Logikbausteine (s. u.) erfolgen.

Der Entwurf von Schaltnetzen erfolgt ausgehend von einer gegebenen Aufgabenstellung, die in verschiedener Form beschrieben sein kann.



Das Ergebnis ist je nach verwendeter Implementierungsform z. B. ein Schaltplan für Gatter oder eine Spezifikation für die Konfiguration programmierbarer Logikbausteine.

Vielfach werden die Schaltfunktionen noch für die jeweilige Implementierungsform optimiert, z. B.:

- minimale Anzahl logischer Verknüpfungen
- minimale Anzahl von Stufen
- minimale Anzahl Verbindungen
- Ausnützung vorhandener Bausteine
- minimale Signallaufzeiten durch die Schaltung
- Herstellungskosten
- ...

4.2 Minimieren von Schaltfunktionen

Minimierungsverfahren dienen dazu, den Realisierungsaufwand (Kosten) und/oder die Laufzeit von digitalen Schaltungen zu optimieren. Weil komplexere Schaltungen für die algebraische Umformung per Hand zu unübersichtlich sind, werden systematisch arbeitende Verfahren eingesetzt.

4.2.1 Ausgangsbasis

Der Ausgangspunkt für eine Minimierung ist oft die Darstellung in *disjunktiver Normalform* (DNF), d. h. die Disjunktion von Konjunktionen (nicht unbedingt Minterme!).

z. B. $\bar{a}bc + a\bar{b} + c\bar{d} + abc$

Angelpunkt zur Reduktion komplexer Funktionen ist:

$$\boxed{p \cdot x + p \cdot \bar{x} = p \cdot (x + \bar{x}) = p \cdot (1) = p}$$

wobei p ein beliebiger Term sein kann.

Beispiel: Minimierung von $a\bar{b} + \bar{a}b + \bar{a}\bar{b}$ durch ***algebraische Umformung***.

$$\bar{a}b + \bar{a}\bar{b} = \bar{a}(b + \bar{b}) = \bar{a}$$

$$a\bar{b} + \bar{a} = (a + \bar{a})(\bar{b} + \bar{a}) = \bar{a} + \bar{b} = \overline{ab}$$

Die Minimierung kann die Anzahl der Stufen optimieren oder andere (Längen-/Kosten-)Maße.

Längendefinitionen (Beispiele)

Länge L(A) eines Ausdrucks A in DNF:

L_v (A) = Anzahl der Variablenzeichen in A

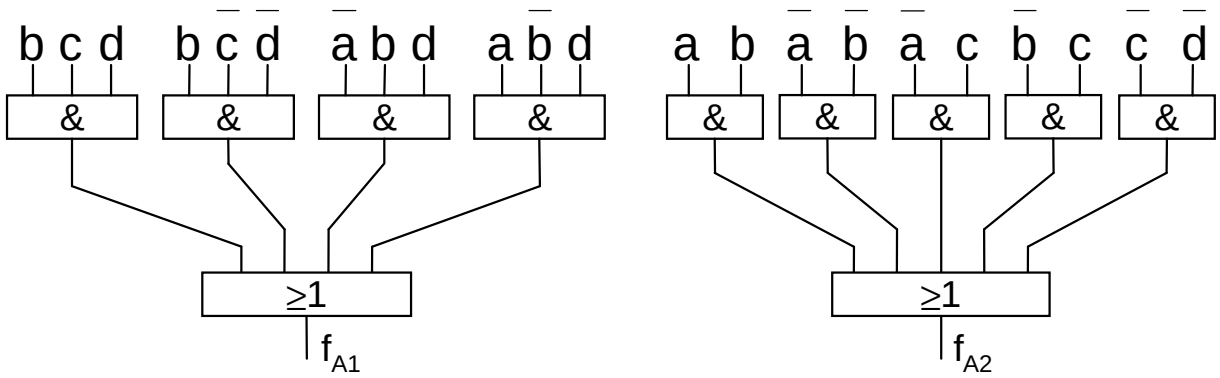
L_d (A) = L_v (A) + Zahl disjunktiv verknüpfter Terme mit mehr als einer Variablen.

L_b (A) = Anzahl der zur Realisierung von A benötigten (integrierten) Bausteine

Beispiel: $A_1 = bcd + b\bar{c}\bar{d} + \bar{a}bd + a\bar{b}c$

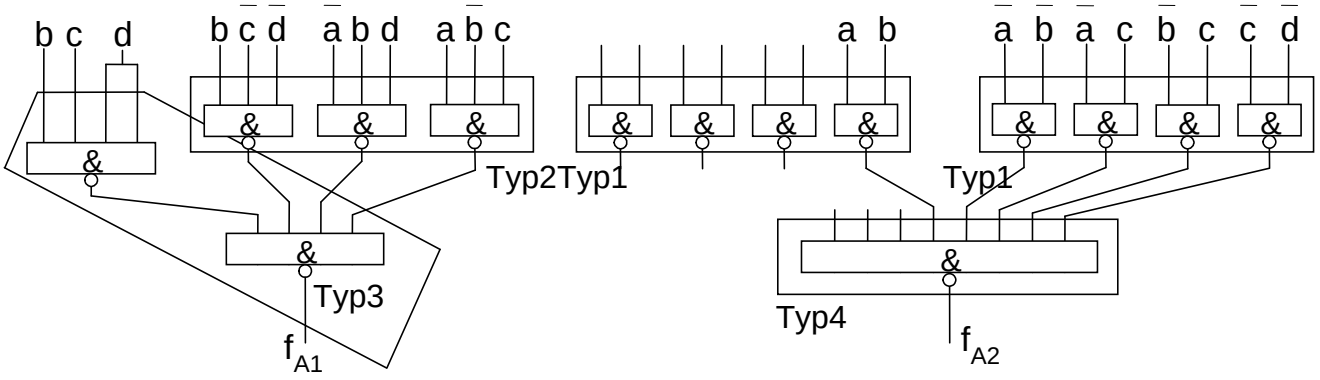
$A_2 = ab + \bar{a}\bar{b} + \bar{a}c + \bar{b}c + \bar{c}\bar{d}$

Realisierung mit diskreten Schaltkreisen



$L_v(A_1) = 12$	$L_v(A_2) = 10$	d. h. $L_v(A_1) > L_v(A_2)$
$L_d(A_1) = 16$	$L_d(A_2) = 15$	d. h. $L_d(A_1) > L_d(A_2)$
$L_b(A_1) = 5$	$L_b(A_2) = 6$	d. h. $L_b(A_1) < L_b(A_2)$

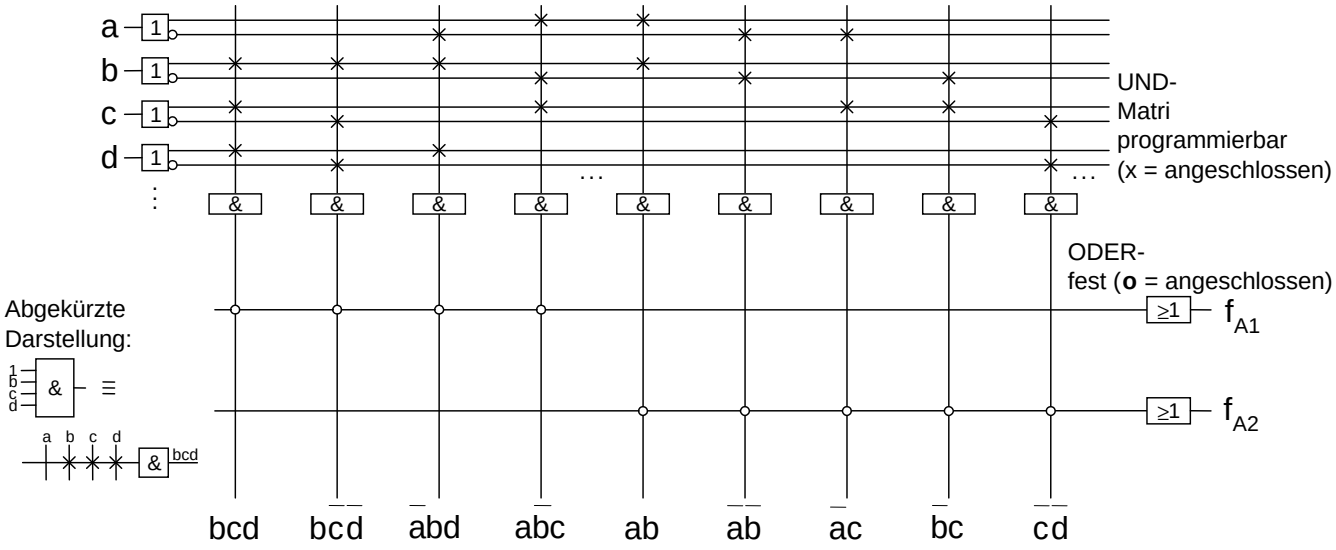
Realisierung mit (einfachen, vorgegebenen) integrierten (Standard-)Schaltkreisen



$L_b(A_1) = 2$ $L_b(A_2) = 3$ d. h. $L_b(A_1) < L_b(A_2)$

(1 Baustein Typ 2, (2 Bausteine Typ 1,
1 Baustein Typ 3) 1 Baustein Typ 4)

Realisierung mit programmierbarer Array-Logik (PALs)



Hier ist: $L_b(A_1) = L_b(A_2) = 1$

→ Längenmaße: Anzahl Produktterme (primäre Kosten)
Anzahl benötigter Verknüpfungspunkte

A_1 und A_2 sind gleichzeitig realisierbar und trotzdem das PAL nur teilweise genutzt. Die Reduktion der Anzahl der Terme ist i. Allg. aber wichtig, da die Anzahl der Eingänge der ODER-Matrix (Produktterme) begrenzt ist.

Definitionen

Ein Term $T = x_1^{d1} x_2^{d2} \ldots x_n^{dn}$ mit $d_i \in \{0,1,2\}$ einer Funktion in disjunktiver Normalform (DNF) heißt Primimplikant oder Primterm, wenn in der DNF kein Term T* existiert, der sich gemäß

$$xy + x\bar{y} = x$$

mit T zu einem einfacheren Term zusammenfassen lässt.

Dabei bedeutet:
$x^0 = \bar{x}$
$x^1 = x$
x^2 : x kommt in T nicht vor.

Eine DNF A heißt disjunktive Minimalform (DMF) bzgl. einer Längendefinition L, wenn es für die durch A dargestellte Funktion keine DNF A* gibt, die bzgl. L kürzer ist, d. h. wenn kein A* existiert mit $L(A^*) < L(A)$.

Satz: Gegeben sei eine Längendefinition, bei der die Länge eines Ausdrucks nicht steigt, wenn eine Variable gestrichen wird. Dann existiert zu jeder Schaltfunktion wenigstens eine DMF, die Disjunktion von Primimplikanten ist.

Bemerkungen: Für L_v, L_d und L_b erfüllt.

Es kann mehrere alternative DMF gleicher Länge geben.

Jede Minimalfunktion bzgl. $L_v(A)$ ist eine Disjunktion von Primtermen bzw. eine Konjunktion von Maxtermen.

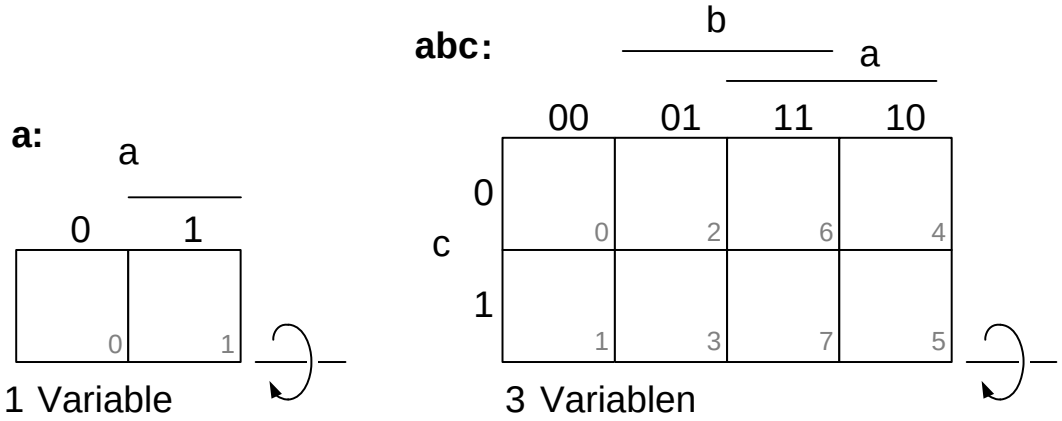
Neben der algebraischen Minimierung gibt es grafische Minimierungsverfahren (für bis zu 6 Variablen) und systematisch arbeitende Minimierungsverfahren. → s. unten

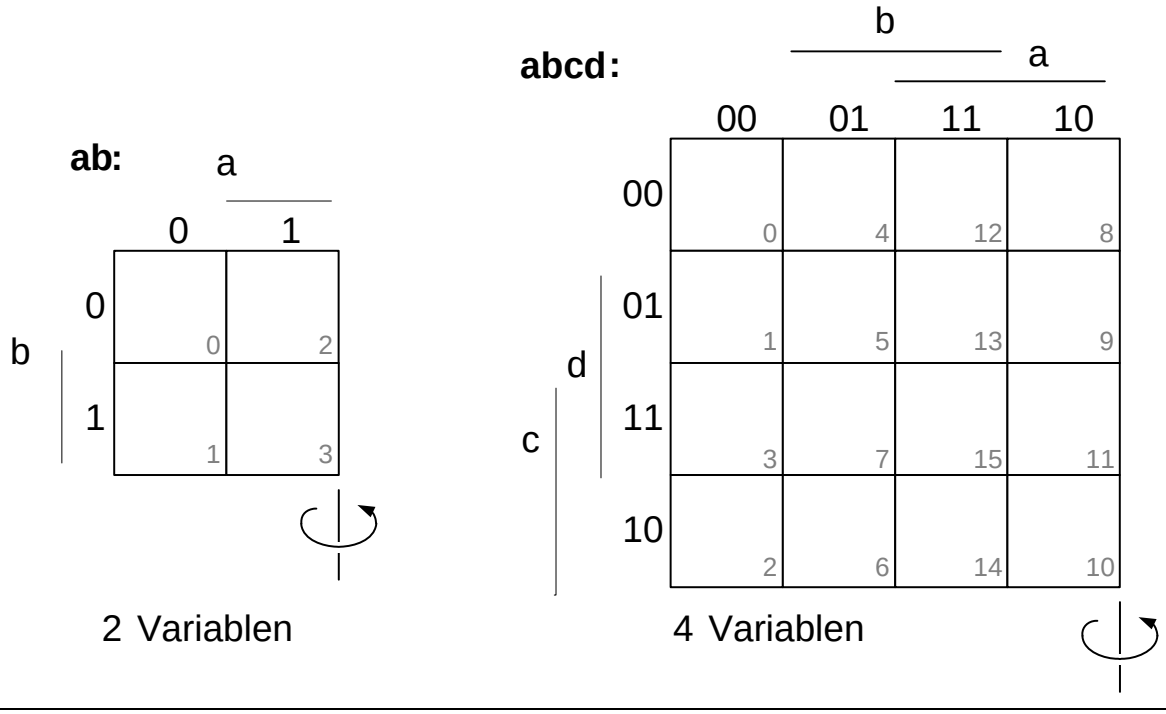
4.2.2 Graphisches Minimierungsverfahren nach Karnaugh-Veitch

I. Darstellung einer Schaltfunktion als KV-Diagramm

Ein KV-Diagramm ist eine graphische Repräsentation einer Wertetabelle mit einer Einteilung eines Rechtecks in 2^N Felder, die den Mintermen der Schaltfunktion entsprechen.

Benachbarte (oder gegenüber liegende) Felder unterscheiden sich *genau um eine Variable* (möglich bis zu 4 Variablen, ab 5 bereits recht unübersichtlich, da 'Nachbarn' nicht mehr unbedingt in direkt benachbarten Feldern liegen).





II. Eintragung der zur DKN gehörigen Minterme

Beispiel 1:

$$f(a,b,c) = ab + a\bar{b} + \bar{a}\bar{b}c$$
$$= abc + ab\bar{c} + a\bar{b}c + a\bar{b}\bar{c} + \bar{a}\bar{b}c$$

c \ a, b	00	01	11	10
0	f(0, 0, 0)	f(0, 1, 0)	f(1,1, 0)	f(1, 0, 0)
1	f(0, 0, 1)	f(0, 1, 1)	f(1, 1, 1)	f(1, 0, 1)

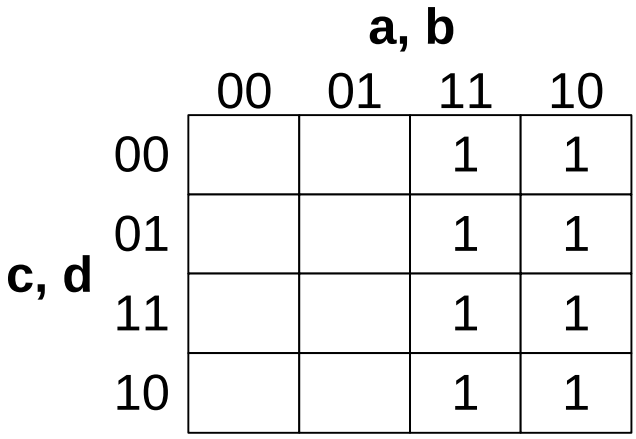
c \ a, b	00	01	11	10
0	0	0	1	1
1	1	0	1	1

Hinweise: Es muss nicht unbedingt die DKN (d. h. alle Minterme) aufgestellt werden, um das KV-Diagramm auszufüllen. Es geht auch von der DNF aus.

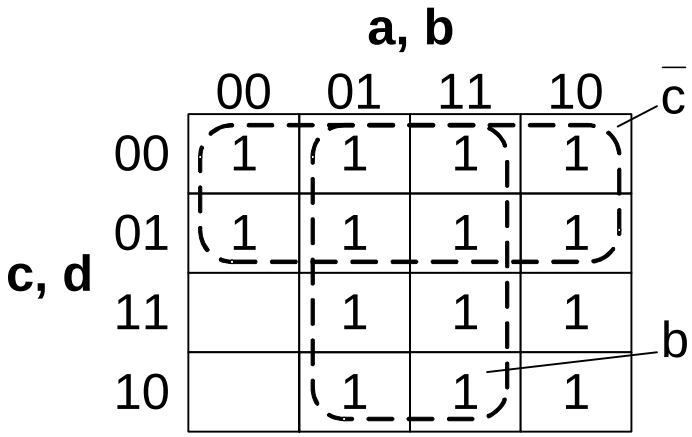
Die Nullen können im KV-Diagramm auch weggelassen werden.

Beispiel 2:

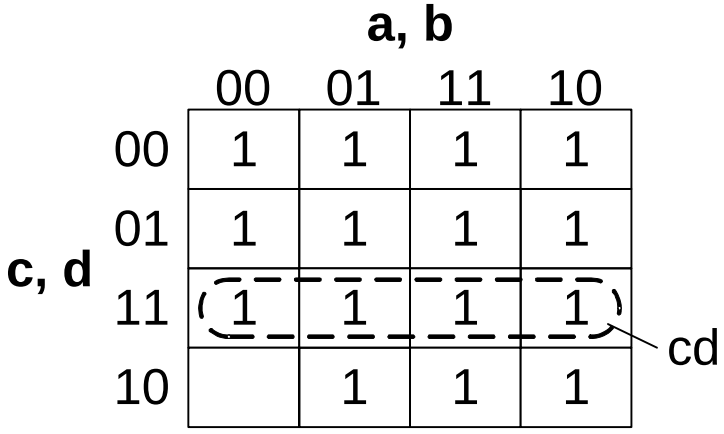
$$f(a,b,c,d) = a + b + \bar{c} + cd$$



a = 1
entspricht
ganzem Feld
(8 Minterme)



b = 1 und $\bar{c}$ = 1
(c = 0) entsprechend



cd = 1 entspricht
einer Zeile
(4 Minterme)

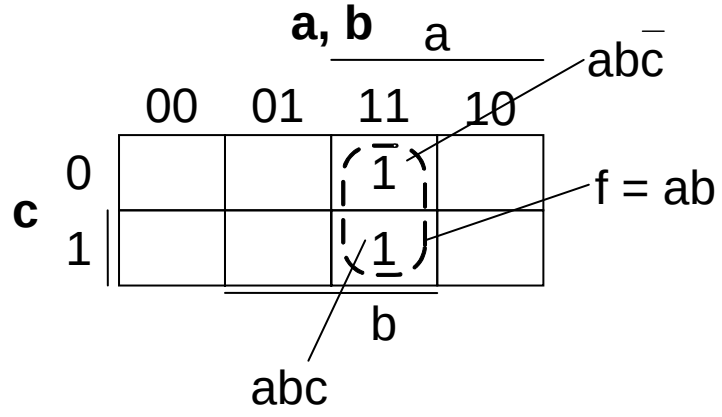
III. Verschmelzung benachbarter Minterme

(1) Die Variable, um die sich die Felder unterscheiden, fällt weg.

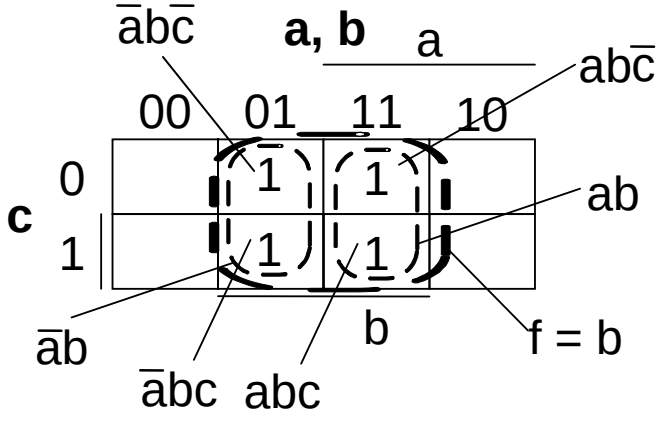
(2) Die benachbarten Minterme reduzieren sich auf einen einzigen Term mit den verbleibenden gemeinsamen Variablen.

(3) Verallgemeinerung auf benachbarte Felder mit 2^i, i = 1, 2, 3 ... Variablen. Es fallen dann jeweils i Variablen aus dem Term weg.

Beispiele: $f(a,b,c) = abc + ab\bar{c} = ab$



$f(a,b,c) = \bar{a}\,b\,\bar{c} + \bar{a}\,bc + abc + ab\,\bar{c} = b$



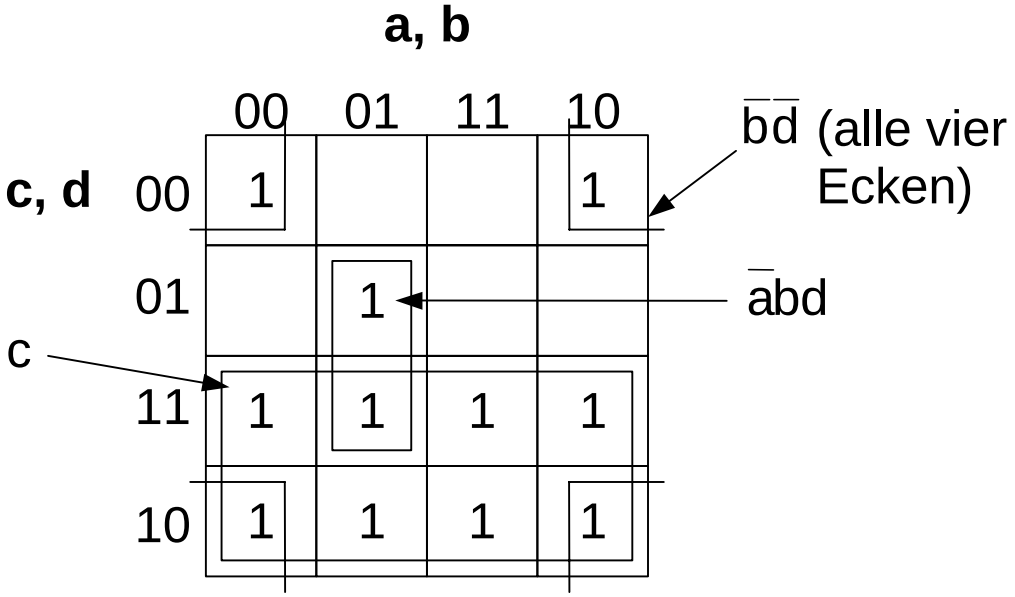
Kochrezept:

* Mit möglichst wenig Schleifen alle ´1´ erfassen:
  - nur benachbarte ´1´, die sich in einer Variablen unterscheiden, d.h. achsparallel wg. Faltungssymmetrie
  - Schleifengröße entspricht Zweierpotenz
  - Einzelne ´1´ können in mehreren Schleifen auftreten. D.h., mehrere zusammengefasste Felder dürfen sich überlappen.

* Die Nachbarschaft erstreckt sich auch über die Ränder hinaus (torusförmiger Abschluss)!

* Prüfen: Terme der Ordnung p entsprechen Feldern der Größe 2^p. D.h., werden 2^p Felder zusammengefasst, reduziert sich die Anzahl Variablen in einem Term um p.

Beispiel:



$$\rightarrow \quad f(a,b,c,d) = c + \bar{b}\bar{d} + \bar{a}bd$$

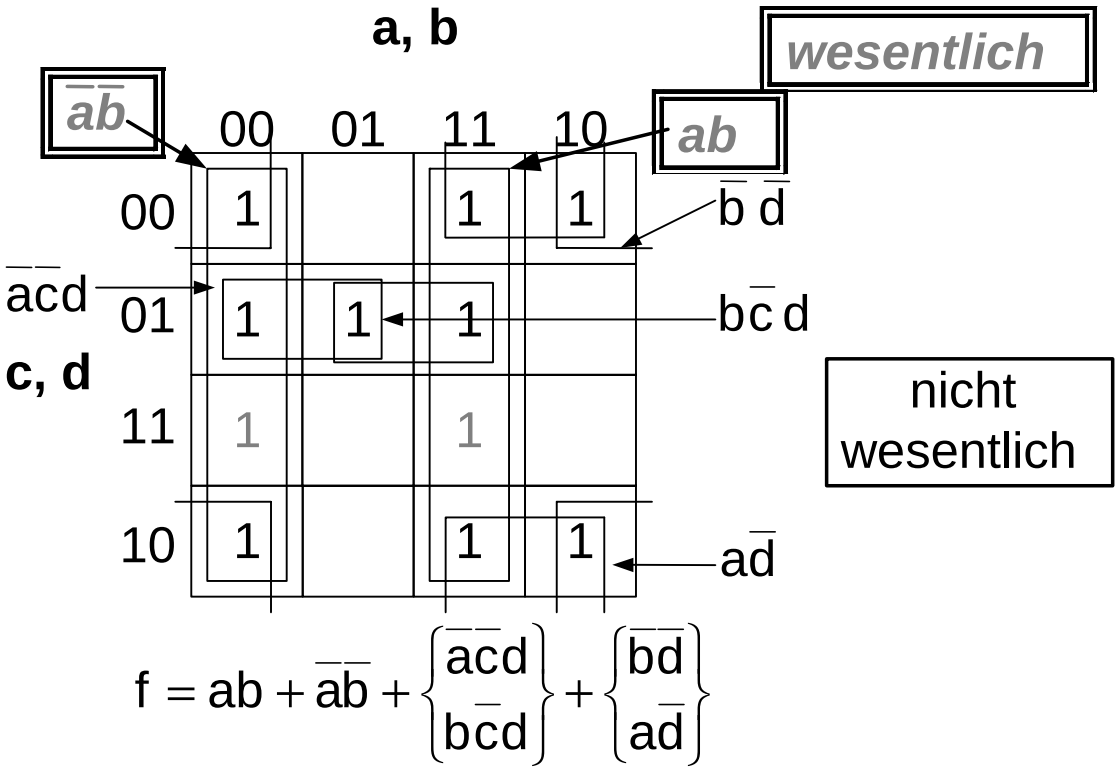
Minimierung mit KV-Diagrammen

- Primimplikanten:
  Durch Zusammenfassen von Mintermen zu *möglichst großen* Feldern der Größe 2^i erhält man die Primimplikanten der Funktion.

- Wesentliche Primimplikanten:
  Diejenigen Felder, die als *einzige* einen Minterm überdecken, entsprechen den *wesentlichen* Primimplikanten und müssen in die DMF aufgenommen werden.

- Minimale Restüberdeckung:
  Die noch nicht abgedeckten Minterme müssen mit einer *minimalen* Anzahl der verbleibenden (nicht wesentlichen) Primimplikanten überdeckt werden (i. Allg. nicht eindeutig!).

Beispiel: (vgl. auch Quine-McCluskey-Beispiel, s. unten)



KV-Diagramme mit 5 und 6 Variablen

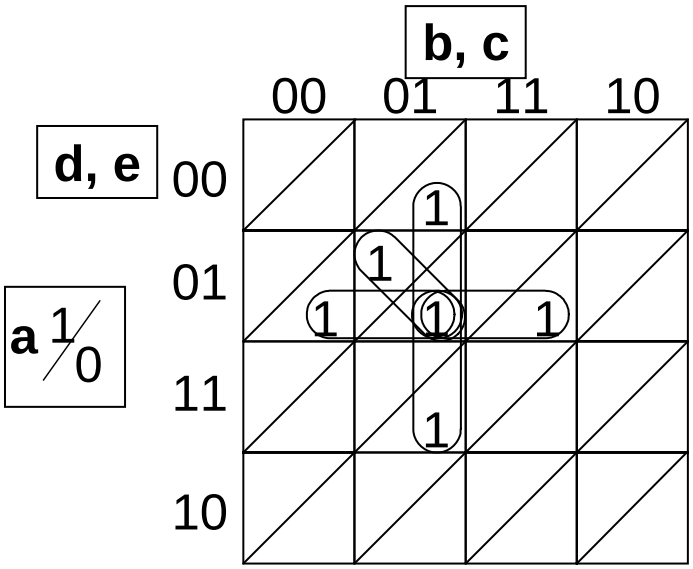
5 Variablen:

Ein KV-Diagramm mit 5 Variablen kann aus 2 KV-Diagrammen mit 4 Variablen abgeleitet werden, wobei Minterme in den jeweils sich entsprechenden Feldern in beiden Teildiagrammen ebenfalls benachbart sind.

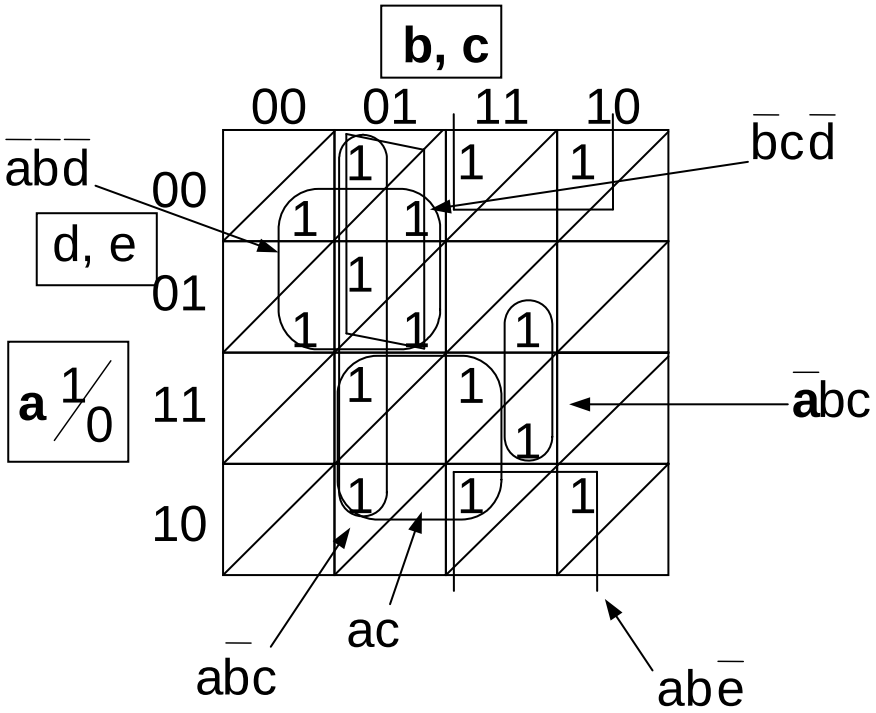
Eine übliche Darstellung besteht aus zwei versetzten 4-Variablen-Diagrammen mit zusätzlichen Diagonalen, wobei Einträge oberhalb der Diagonale zu dem einen 4-Variablen-Teildiagramm gehören, die unterhalb zu dem anderen.

D. h., Minterme ober- und unterhalb der Diagonalen im gleichen Kästchen sind benachbart.

Beispiel: Nachbarterme von Minterm $\bar{a}\bar{b}c\bar{d}e$



Beispiel:

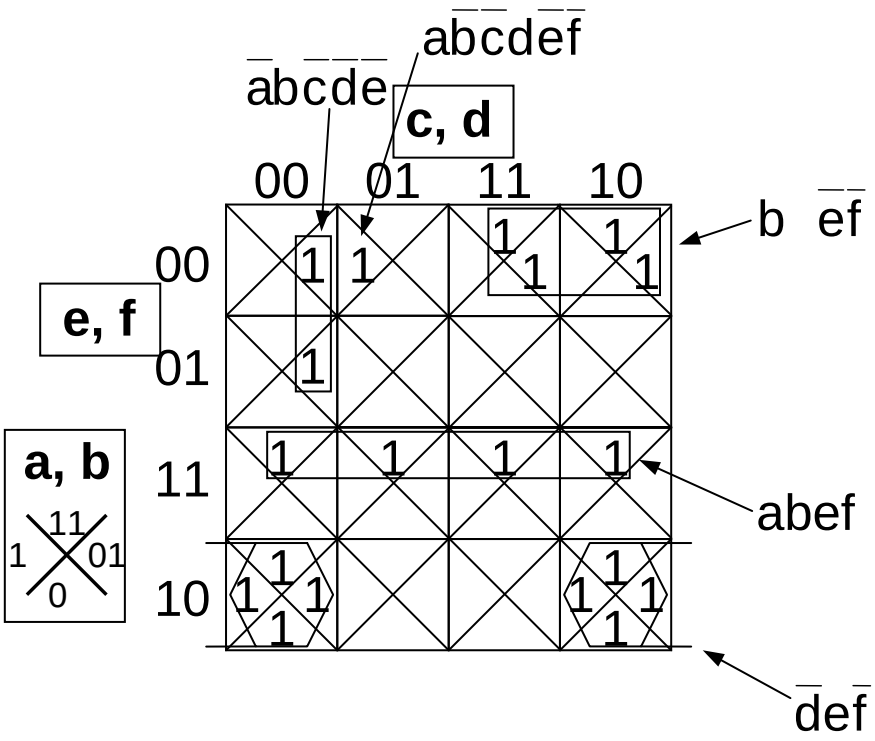


$$f = \bar{a}\bar{b}\bar{d} + ab\bar{e} + ac + \bar{a}bc + \left\{ \begin{matrix} a\bar{b}c \\ \bar{b}c\bar{d} \end{matrix} \right\}$$

Achtung: Felder in benachbarten Kästchen dürfen nur zusammengefasst werden, wenn sie im gleichen oberen oder unteren Teilfeld (d. h. oberhalb bzw. unterhalb der Diagonalen) stehen!

6-Variablen Diagramme

KV-Diagramme mit 6 Eingangsvariablen können aus 4 Teildiagrammen mit je 4 Variablen aufgebaut werden, die durch 4 Teilfelder im 4-Variablen-Diagramm darstellbar sind.



$$f = a\bar{b}cd\bar{e}\bar{f} + \bar{a}bcde + b\ \bar{e}\bar{f} + abef + \bar{d}\bar{e}\bar{f}$$

KV-Diagramm für mehr als 6 Variablen sind nicht mehr überschaubar!

Systematische Vorgehensweise bei der KV-Minimierung:

Ziel: Mit möglichst wenig Feldern alle Einsen der Funktion erfassen

→ möglichst große Felder finden

Strategie: Erst wesentliche Primimplikanten ermitteln, dann minimale Restüberdeckung. D. h., mit möglichst wenig Schritten alle Einsen erfassen.

Schritte:

(1) Wähle einen Minterm (eine 1), der noch nicht abgedeckt ist.

(2) Betrachte die benachbarten Einsen dieses Minterms. Falls ein einziger Term nach dem maximalen Verschmelzen den Minterm abdeckt, ist dieser Term ein *wesentlicher Primimplikant* und muss in die DMF aufgenommen werden.

(3) Wiederhole (1) und (2), bis alle wesentlichen Primimplikanten gefunden worden sind.

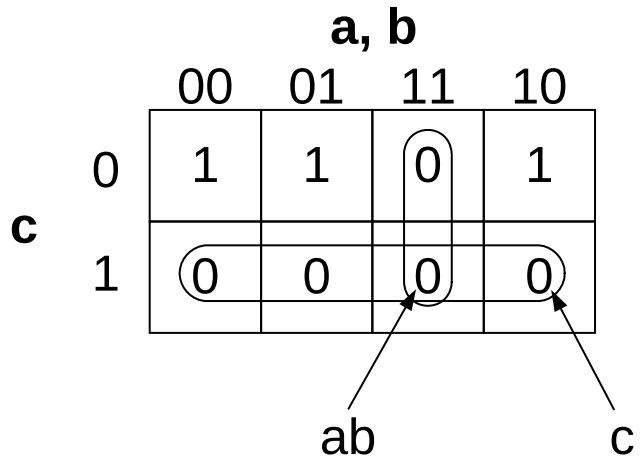
(4) Suche eine minimale Menge von Primimplikanten, die alle übrigen, noch nicht abgedeckten Minterme (Einsen) überdecken. Gibt es mehrere solcher Mengen, wähle die mit den wenigsten Variablen (minimale Restüberdeckung).

Konjunktive Minimalform (KMF)

Analog zur DMF enthält die konjunktive Minimalform KMF die minimale Anzahl von Variablen und Termen in *konjunktiver* Normalform (KNF).
Sie bietet sich für Schaltfunktionen mit wenigen ´0´ bzw. einer „günstigen“ Anordnung der ´0´ an.

KV-Diagramm mit Mintermen:
Zusammenfassen der Felder mit “0” minimiert $\bar{f}$ statt f, d. h. $\bar{f}$ liegt als DMF vor. Anwendung des De Morganschen Gesetzes liefert dann f in KMF.

Beispiel:



DMF: $f = \bar{a}\bar{c} + \bar{b}\bar{c}$

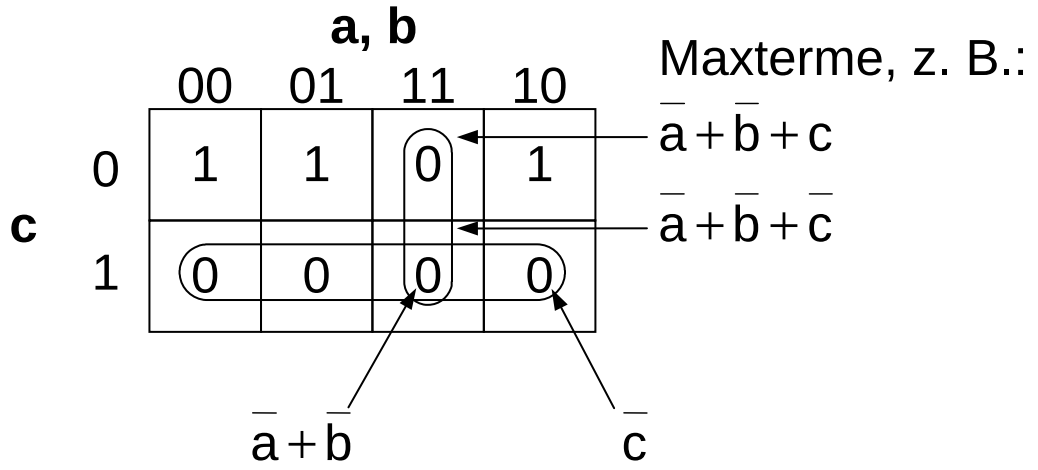
DMF von $\bar{f}$: $\bar{f} = c + ab$

De Morgan: $f = \overline{c + ab} = \bar{c} * \overline{ab}$

KMF von f $= \mathbf{\bar{c}(\bar{a} + \bar{b})}$

KV-Diagramm mit Maxtermen

Analog zum KV-Diagramm für Minterme lassen sich auch KV-Diagramme für *Maxterme* konstruieren.



$$KM : f = \bar{c}(\bar{a}+\bar{b})$$

Die Maxterme können dann analog zu Mintermen mittels der Beziehung

$$(x+y)(x+\bar{y}) = x$$

zusammengefasst und damit die KMF direkt bestimmt werden.

Weiteres graphisches Minimierungsverfahren:

Händlerscher Kreisgraph (bis 5 Variable)

4.2.3 Algorithmisches Verfahren nach Quine-McCluskey

Ausgangsbasis: Satz von Quine

Sei $f \neq 0$. Dann besteht eine Minimalform einer Schaltfunktion f ausschließlich aus Primimplikanten von f.

Schritte:

I. Erstellen der disjunktiven kanonischen Normalform (DKN)

II. Ermitteln der Primimplikanten

- Einteilen der Minterme in Klassen K_i, wobei K_i alle Minterme mit i nichtnegierten Variablen enthält.
- Zusammenfassen von Mintermen benachbarter Klassen gemäß

$$xy + x\bar{y} = x.$$

- Verschmolzene Minterme werden abgehakt.
- Die neu entstandenen Terme höherer Ordnung werden ggf. weiter verschmolzen und abgehakt, bis keine weitere Zusammenfassung möglich ist.
- Nicht mehr verschmelzbare Terme bzw. Minterme sind die gesuchten Primimplikanten (auch Primterme genannt).

III. Bestimmen der wesentlichen Primimplikanten

(wesentlich: enthält als einziger einen der Minterme und muss daher **unbedingt** verwendet werden)

IV. Auswahl der minimalen Restüberdeckung

(d.h. der durch die wesentlichen Primimplikanten nicht abgedeckten Minterme mittels unwesentlicher Primimplikanten)

Beispiel:

$$f = ab + \bar{a}\bar{b}c + b\bar{c}d + a\bar{b}\bar{d} + \bar{a}\bar{b}\bar{c}$$

Schritt I: DKN bilden

Erweitern analog zu: $ab = abc + ab\bar{c}$

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

→ **DKN** $f = abcd + ab\bar{c}d + abc\bar{d} + ab\bar{c}\bar{d} +$
$a\bar{b}c\bar{d} + a\bar{b}\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bcd +$
$\bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c}\bar{d}$

Schritt II: Ermitteln der Primimplikanten

Teilschritt a: Bilde eine Tabelle und trage in die erste Spalte die binär dargestellten Indizes aller Minterme von f sortiert nach der Anzahl der Einsen.

Teilschritt b: Fasse Terme aus benachbarten Gruppen, die sich nur in einer Stelle unterscheiden, zusammen und markiere sie.

Teilschritt c: Wiederhole bis keine Terme mehr zusammengefasst werden können.

Zum Verschmelzen verwendete Terme können abgehakt werden.

Die am Ende verbleibenden Terme sind die Primimplikanten.

Schritt II: Primimplikanten

Klasse	#	Minterme	verschmolzene Minterme	neue Terme	verschmolzene Terme	neue Terme
K_0	0	$\bar{a}\bar{b}\bar{c}\bar{d}$	0,1	$\bar{a}\bar{b}\bar{c}$	0,1 - 2,3	$\bar{a}\bar{b}$
			0,2	$\bar{a}\bar{b}\bar{d}$	0,2 - 1,3	$\bar{a}\bar{b}$
					0,2 - 8,10	$\bar{b}\bar{d}$
			0,8	$\bar{b}\bar{c}\bar{d}$	0,8 - 2,10	$\bar{b}\bar{d}$
K_1	1	$\bar{a}\bar{b}\bar{c}d$	1,3	$\bar{a}\bar{b}d$		
	2	$\bar{a}\bar{b}c\bar{d}$	2,3	$\bar{a}\bar{b}c$		
			1,5	$\bar{a}c\bar{d}$		
	8	$\bar{a}bc\bar{d}$	2,10	$\bar{b}c\bar{d}$		
			8,10	$\bar{a}\bar{b}d$	8,10 - 12,14	$\bar{a}d$
			8,12	$\bar{a}c\bar{d}$	8,12 - 10,14	$\bar{a}d$
K_2	3	$\bar{a}\bar{b}cd$	5,13	$\bar{b}cd$		
			12,13	$\bar{a}bc$	12,13 - 14,15	ab
	5	$\bar{a}b\bar{c}d$	10,14	$\bar{a}c\bar{d}$		
	10	$\bar{a}bc\bar{d}$	12,14	$\bar{a}b\bar{d}$	12,14 - 13,15	ab
	12	$\bar{a}\bar{b}c\bar{d}$				
K_3	13	$\bar{a}bcd$	13,15	abd		
			14,15	abc		
	14	$\bar{a}bc\bar{d}$				
K_4	15	abcd				

Schritt III: Wesentliche Primimplikanten identifizieren

Die Primimplikanten werden in einer Matrix, der Primimplikantentabelle, den Mintermen gegenüber gestellt, d.h. alle Minterme angekreuzt, die in einem Primimplikanten enthalten sind.

Primimplikantentabelle: (Überdeckungsmatrix)

	Minterme											
Prim-impl.	0	1	2	3	5	8	10	12	13	14	15	WP
$\bar{a}\bar{c}d$		x			x							
$b\bar{c}d$					x				x			
$\bar{a}\bar{b}$	x	x	x	Ⓧ								←
$\bar{b}\bar{d}$	x		x			x	x					
$a\bar{d}$						x	x	x		x		
ab								x	x	x	Ⓧ	←

Anmerkung: Ein x kennzeichnet hier die Überdeckung eines Minterms mit einem Primimplikanten.

Wenn z. B. Primimplikant $\bar{a}\bar{b}$ in die DMF aufgenommen wird, dann werden die Minterme

$$m_0,\ m_1,\ m_2 \text{ und } m_3$$

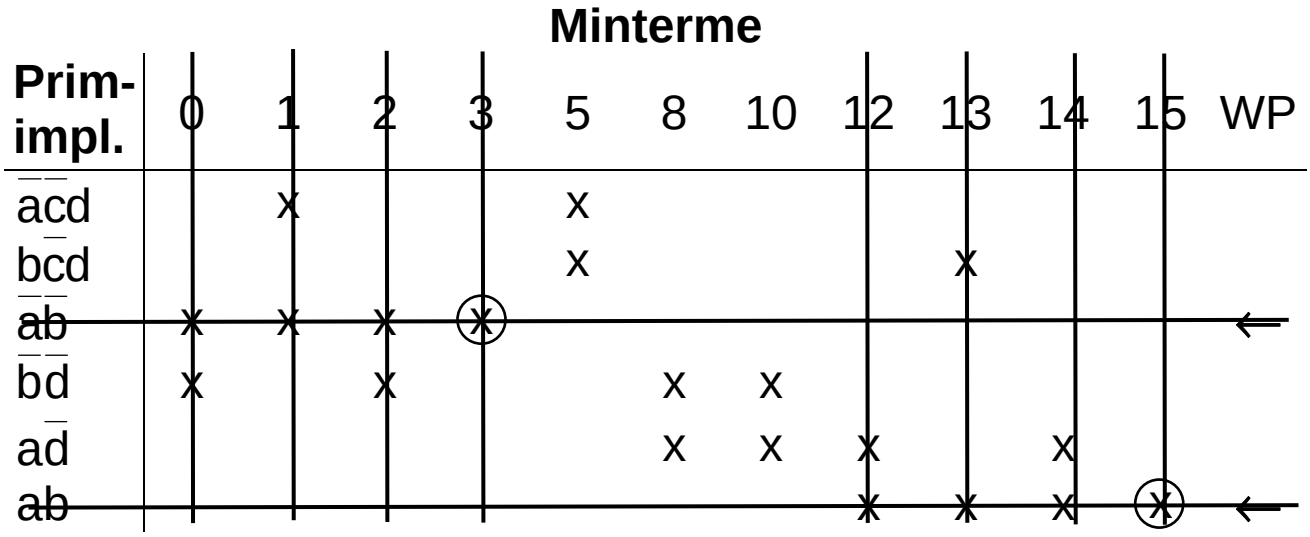
genau dann 1, wenn $\bar{a}\bar{b} = 1$ ist.

Wesentliche Primimplikanten (Kernimplikanten, Hauptterme) decken als einzige einen Minterm ab und **müssen** daher verwendet werden.

Das sind hier $\bar{a}\,\bar{b}, ab$, da sie als einzige die Minterme 3 bzw. 15 abdecken.

Die durch die Primimplikanten abgedeckten Minterme können gestrichen werden.

Primimplikantentabelle:



Schritt IV: Minimale Restüberdeckung bestimmen

Die wesentlichen Primimplikanten und die durch sie abgedeckten Minterme werden aus der Primimplikantentabelle entfernt.

Die noch nicht durch die wesentlichen Primimplikanten abgedeckten Minterme müssen durch eine *minimale* Menge der unwesentlichen Primimplikanten überdeckt werden.

	5	8	10
$\bar{a}\bar{c}d$	x		
$b\bar{c}d$	x		
$\bar{b}\bar{d}$		x	x
$a\bar{d}$		x	x

Bei einfachen Schaltfunktionen kann die minimale Restüberdeckung unmittelbar aus der Tabelle bestimmt werden.

Minimierte Schaltfunktion (DMF)

Die DMF enthält die Disjunktion aus allen wesentlichen Primimplikanten und die unwesentlichen Primimplikanten aus einer minimalen Restüberdeckung. Für letzteres gibt es oft mehrere (hier 2 + 2) gleichwertige Lösungen.

Hier: $$f = ab + \bar{a}\bar{b} + \left\{\begin{matrix}\bar{a}\bar{c}d\\ b\bar{c}d\end{matrix}\right\} + \left\{\begin{matrix}\bar{b}\bar{d}\\ a\bar{d}\end{matrix}\right\}$$

Wenn die Restüberdeckung nicht so offensichtlich zu finden ist wie hier, kann nach folgenden Regeln systematischer vorgegangen werden:

Spaltendominanz:

Eine Spalte S_i einer Primimplikantentabelle dominiert eine Spalte S_j, wenn sie mindestens in allen Zeilen einen Eintrag hat, für die auch die Spalte S_j einen Eintrag besitzt. D.h., jeder Primplikant (Zeile), der Spalte S_j überdeckt, überdeckt auch Spalte S_i.

→ Die dominierende Spalte S_i kann entfernt werden, weil die Spalte S_j auf jeden Fall überdeckt werden muss und damit dann auch die Spalte S_i überdeckt wird.

Zeilendominanz:

Eine Zeile P_i dominiert eine Zeile P_j, wenn sie mindestens alle Spalten der Zeile P_j überdeckt.

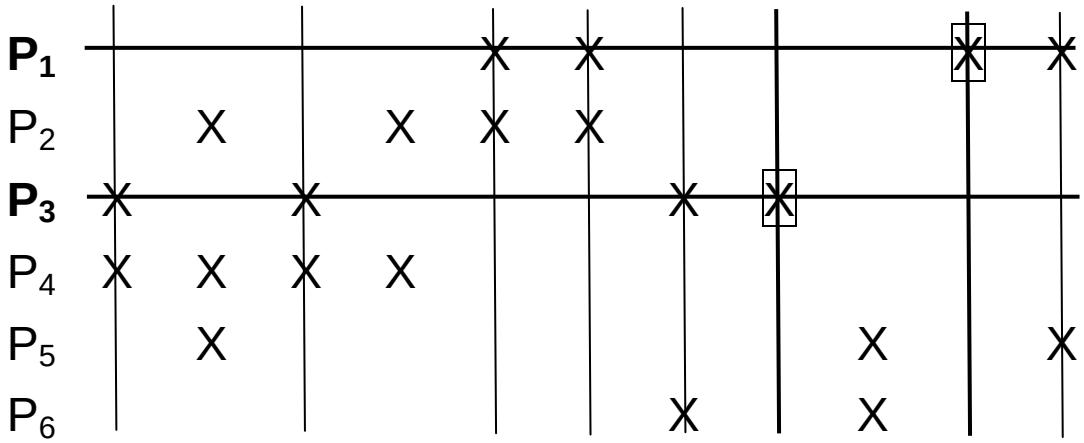
→ Dominierte Zeile P_j löschen.

Bei gleicher Anzahl abgedeckter Spalten die Zeile mit den höheren Kosten löschen.

1. Schritt: Bestimmung der wesentlichen Primimplikanten

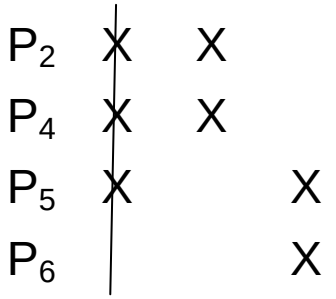
P_1						X	X				X	X
P_2		X		X		X	X					
P_3	X		X					X	X			
P_4	X	X	X	X								
P_5		X								X		X
P_6								X		X		

und Eliminieren der Primimplikanten sowie der dadurch abgedeckten Minterme



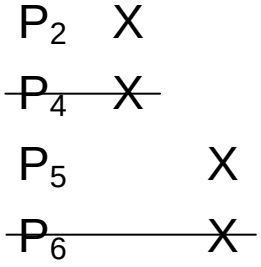
2. Schritt: Spaltendominanz

Entfernen aller Minterme im Rest, die einen anderen Minterm dominieren



3. Schritt: Zeilendominanz

Entferne alle Primterme, die durch einen anderen, nicht längeren dominiert werden



➔ Minimal ist die Lösung: **P_1**, **P_3**, P_2, P_5.

Systematische Bestimmung der minimalen Restüberdeckung

Es gibt weitere systematische Verfahren zum Auffinden der minimalen Restüberdeckung (programmierbar), die vor allem bei komplexeren Schaltfunktionen angewandt werden.

Beispiel: Verfahren von Petrick

(1) Ordne den *Zeilen* der Tabelle mit den unwesentlichen Primimplikanten die Schaltvariablen P_1, P_2, ... zu.

Beispiel:

		0	1	2	5	6	7
P_1	$\bar{a}\bar{b}$	x	x				
P_2	$\bar{a}\bar{c}$	x		x			
P_3	$\bar{b}c$		x		x		
P_4	$b\bar{c}$			x		x	
P_5	ac				x		x
P_6	ab					x	x

(2) Bilde eine Schaltfunktion P, die 1 liefert, wenn alle *Spalten* (Minterme) abgedeckt sind.

P ist eine Konjunktion von ODER-verknüpften Termen ($P_{i0} + P_{i1} + ...$), wobei die P_{i0}, P_{i1}, ... denjenigen Zeilen entsprechen, die die Spalte i überdecken (KNF). Hier:

$$P = (P_1+P_2)(P_1+P_3)(P_2+P_4)(P_3+P_5)(P_4+P_6)(P_5+P_6)$$

(3) Minimiere diese Schaltfunktion P durch Ausmultiplizieren und Reduktion mittels X+XY=X (relativ einfach, da keine negierten Terme):

$$P = P_1P_4P_5 + P_1P_2P_5P_6 + P_2P_3P_4P_5 + P_1P_3P_4P_6 + P_2P_3P_6$$

(4) Jeder Term steht für eine Menge von unwesentlichen Primimplikanten, die jeweils alle Minterme überdecken.

→ Wähle diejenigen Terme mit der geringsten Anzahl von Variablen aus:

$$P_1P_4P_5 \ , \ P_2P_3P_6$$

D. h., die Primimplikanten P1, P4 und P5 bzw. alternativ P2, P3 und P6 decken jeweils alle Minterme ab.

(5) Bestimme für jeden dieser Terme die Anzahl von Schaltvariablen in der zugehörigen DNF aus unwesentlichen Primimplikanten. Das Minimum entspricht der gesuchten minimalen Restüberdeckung, in diesem Beispiel also:

$$F = \bar{a}\bar{b} + b\bar{c} + ac \quad \text{oder}$$

$$F = \bar{a}\bar{b} + \bar{b}c + ab$$

mit je 6 Schaltvariablen.

(6) Die gesuchte disjunktive Minimalform DMF kann nun bestimmt werden, indem die wesentlichen Primimplikanten noch zu einer der Funktionen F mit der minimalen Restüberdeckung hinzugenommen werden.

Für Handauswertung bei größeren Schaltfunktionen recht mühsam, aber leicht programmierbar. Das Verfahren von Petrick hat allerdings exponentielle Laufzeit. Für die Logiksynthese komplexer Schaltnetze werden deshalb in der Praxis oft Verfahren angewendet, die nur eine "gute" Lösung liefern.

4.2.4 Minimierung partiell definierter Schaltfunktionen

Für einige Variablenbelegungen ist bei partiell definierten Schaltfunktionen der Funktionswert nicht definiert (bzw. beliebig 0 oder 1, d.h. “don’t care“), z. B. weil eine Kombination von Eingangssignalen nicht auftreten kann. Folglich kann das Ausgangssignal für don´t cares geeignet gewählt werden, um eine Schaltfunktion zu minimieren.

$$f: S \rightarrow B \text{ mit } S \subset B^n \qquad B = \{0,1\}$$

Beispiel:

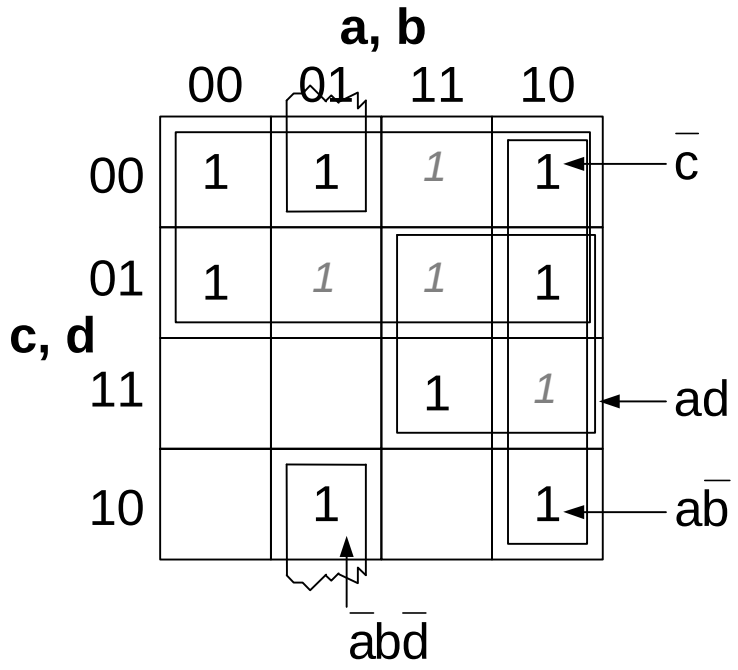
a	b	c	d	f(a, b, c, d)
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	x
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	0
1	1	1	1	1

c, d \ a, b	00	01	11	10
00	1	1	x	1
01	1	x	x	1
11			1	x
10		1		1

x = don’t care

Minimierung im KV-Diagramm:

Don’t care-Terme können leicht zur Minimierung verwendet werden, indem sie selektiv (virtuell) so auf 1 gesetzt werden, dass möglichst große ´1´-Felder entstehen. Sie können, müssen aber nicht für die Bildung von möglichst großen ´1´-Feldern berücksichtigt werden.



$$\mathbf{f = \bar{c} + ad + a\bar{b} + \bar{a}b\bar{d}} \qquad \mathbf{(L_V(f) = 8)}$$

statt

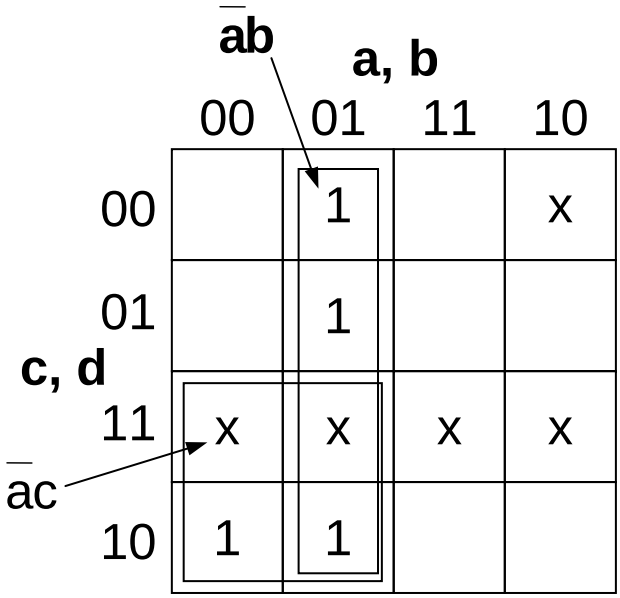
$$f = \bar{b}\bar{c} + \bar{a}b\bar{d} + a\bar{b}\bar{d} + abcd \qquad (L_V(f) = 12)$$

ohne Ausnutzung von don´t cares bzw.
bei don’t care = 0

Minimierung mittels Quine-McCluskey:

Veranschaulichung im KV-Diagramm:

Beispiel:



Zunächst alle Don't care-Terme „virtuell" auf 1 setzen und die nichtredundanten Minterme (also nicht die don´t cares) markieren. (Hier sind sie fett gedruckt.)

$f(a,b,c,d) = \mathbf{m_2} + m_3 + \mathbf{m_4} + \mathbf{m_5} + \mathbf{m_6} + m_7 + m_8 + m_{11} + m_{15}$

(m_i: Minterm i, z. B. $m_2 = \bar{a}\bar{b}c\bar{d}$)

Ermittlung der Primimplikanten (Schritt II)

	#	Minterme	Verschmelzung		Verschmelzung		Primimp.
K_1	**2**	0010	**2**,3	001-	**2**,3 - **6**,7	0-1-	$\bar{a}c$
	4	0100	**2**,6	0-10	**2**,**6** - 3,7	0-1-	
	8	1000	**4**,5	010-	**4**,**5** - **6**,7	01--	$a\bar{b}\bar{c}\bar{d}$
			4,6	01-0	**4**,**6** - **5**,7	01--	$\bar{a}b$
K_2	3	0011	3,7	0-11	3,7 - 11,15	--11	cd
	5	0101	3,11	-011	3,11 - 7,15	--11	
	6	0110	**5**,7	01-1			
			6,7	011-			
K_3	7	0111	7,15	-111			
	11	1011	11,15	1-11			
K_4	15	1111					

Bei Schaltfunktionen mit Redundanzen werden redundante Minterme zwar in die Tabelle aufgenommen und zum Verschmelzen verwendet, aber nicht bei der Ermittlung der Primimplikanten berücksichtigt.

Deshalb ergibt sich hier:

$$f = \bar{a}c + \bar{a}b$$

Bemerkung: Hier alternative Form der Tabelle mit Darstellung der Minterme als Bitvektoren
Bei Verschmelzung steht '-' für weggefallene Variablen.

Wesentliche Primimplikanten (Schritt III)

Nur *'echte'* (notwendige) Minterme übernehmen, denn die don´t care-Terme dienen nur der Vereinfachung (Verschmelzung) und sind keine Primimplikanten.

	Minterme				
Prim-implik.	2	4	5	6	
$\bar{a}c$	(x)			x	← $\bar{a}c$ wesentlich
$\bar{a}b$		(x)	(x)	x	← $\bar{a}b$ wesentlich
cd					
$a\bar{b}\bar{c}\bar{d}$					

Restüberdeckung (Schritt IV)

entfällt hier

und damit ist

$$\mathbf{f(a,b,c,d) = \bar{a}c + \bar{a}b}$$

(vgl. KV-Diagramm)

4.3 Minimierung von Schaltnetzen mit mehreren Ausgängen

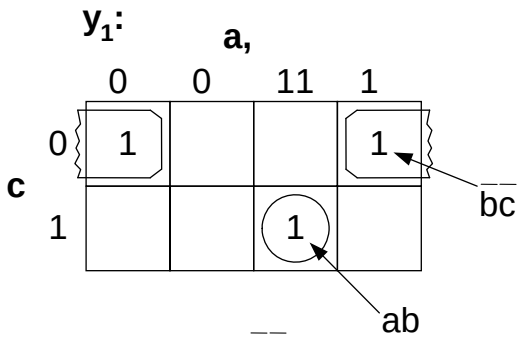
Die Minimierung der einzelnen Schaltfunktionen liefert bei Schaltnetzen mit mehreren Ausgängen meist nicht die minimale Lösung für das gesamte Schaltnetz (Anzahl Gatter, Anzahl Leitungen), da häufig Teile der Schaltung mehrfach genutzt werden können.
Meist ist es das primäre Ziel, die Anzahl von Gattern zu reduzieren, und dann erst die Anzahl der Leitungen.

Beispiel:

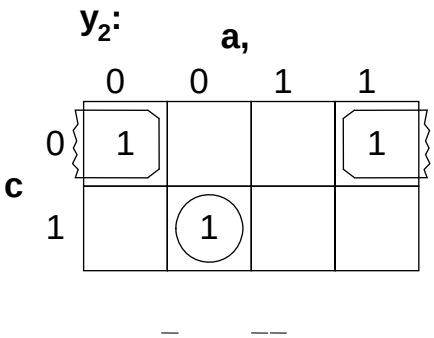
f(a, b, c) = (y_1, y_2, y_3)

$$y_1 = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + abc$$
$$y_2 = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + \bar{a}\bar{b}c$$
$$y_3 = \bar{a}\bar{b}c + a\bar{b}\bar{c} + a\bar{b}c + ab\bar{c} + abc$$

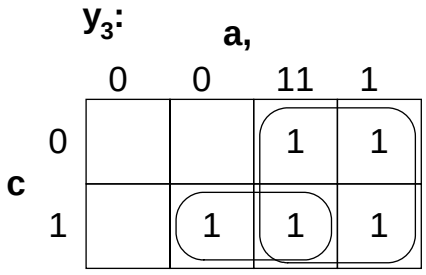
Einzelminimierung mit KV-Diagramm



$y_1 = ab + \bar{b}\bar{c}$

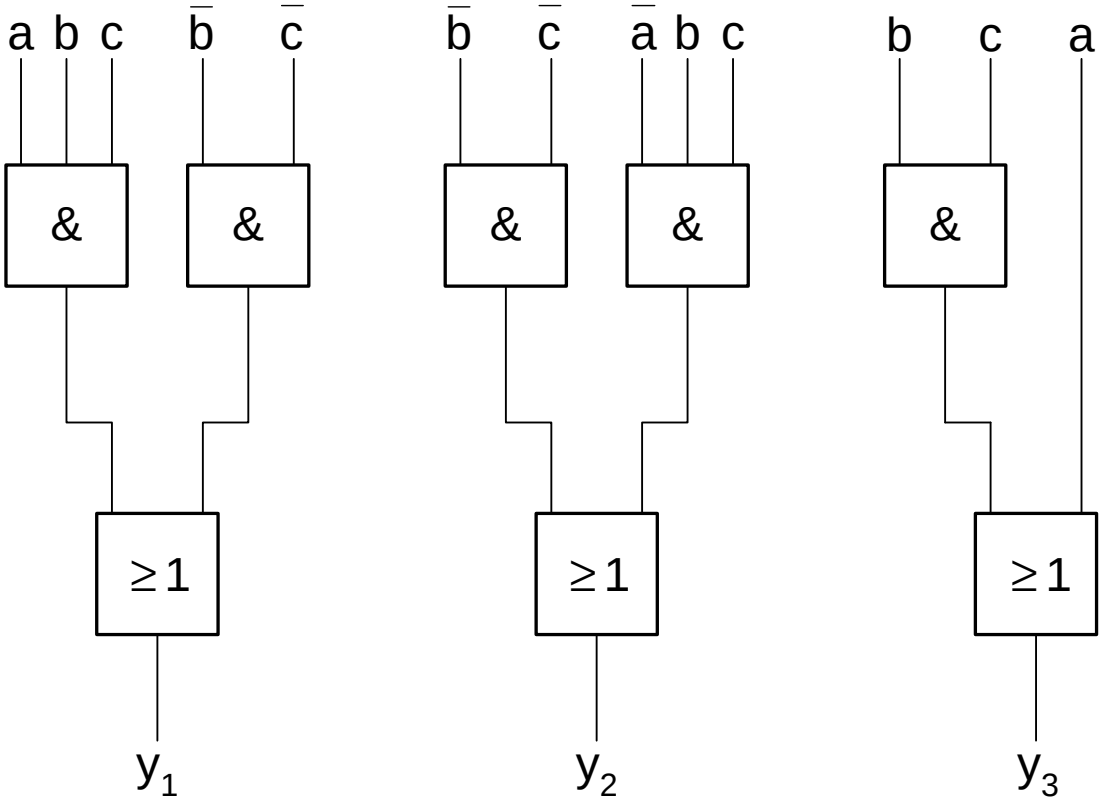


$y_2 = \bar{a}b + \bar{b}\bar{c}$

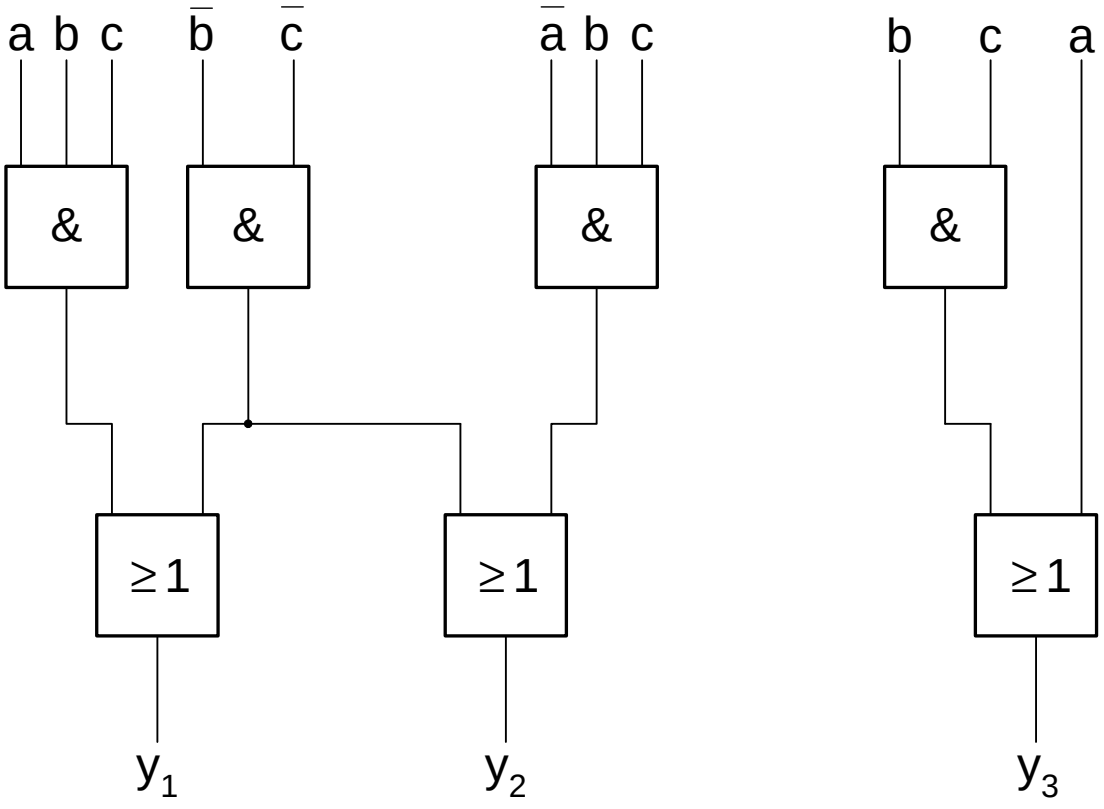


$y_3 = a + b$

Realisierung mit UND/ODER-Gattern:



Vereinfachung durch Mehrfachbenutzung von Termen, hier: $(\bar{b}\bar{c})$ (siehe auch KV-Diagramm)

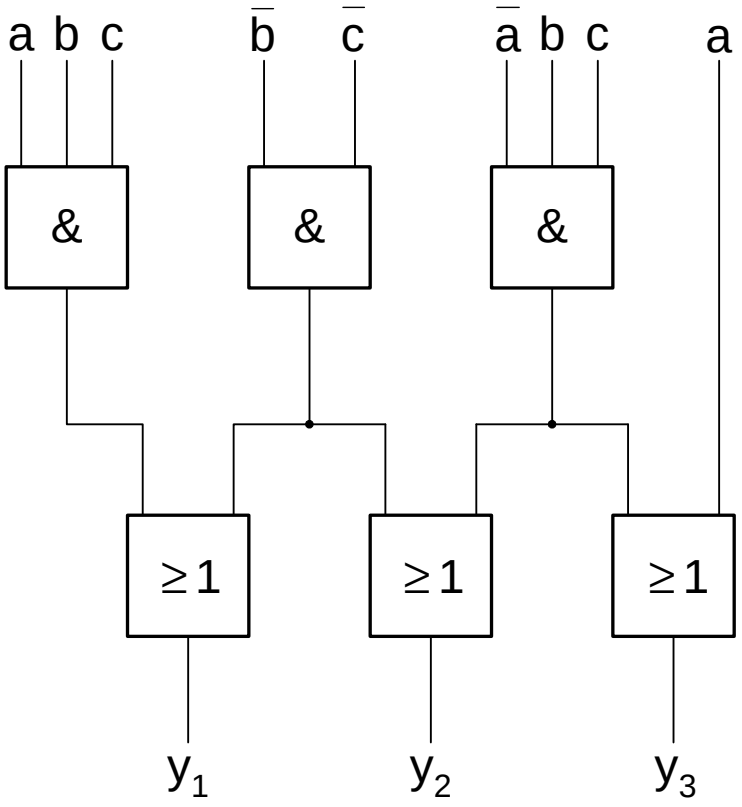


Weitere Vereinfachung durch Abkehr von DMF für y_3:

$$y_3 = a + bc(a + \bar{a}) = a + abc + \bar{a}bc = a + \bar{a}bc$$

(siehe auch KV-Diagramm)

Minimale Schaltung:



Schaltnetze mit Verzweigungen heißen auch *vermaschte Schaltnetze.*

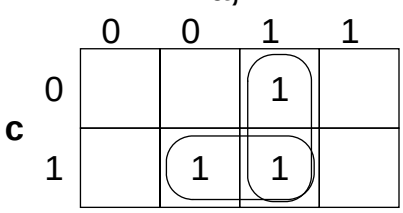
Terme, die in mehreren Schaltfunktionen des Schaltnetzes auftreten heißen *Koppelterme.*
(hier z. B.: $\bar{b}\bar{c}$, $\bar{a}bc$)

Es gibt systematische Verfahren, die die Minimierung von Schaltnetzen unter Berücksichtigung von Koppeltermen erlauben.

Mehrfach-Ausgangs-Minimierung mit KV-Diagrammen

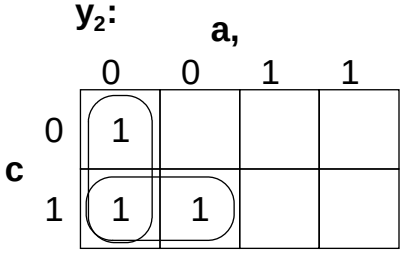
Beispiel **$f(a, b, c) = (y_1, y_2)$**

y_1:

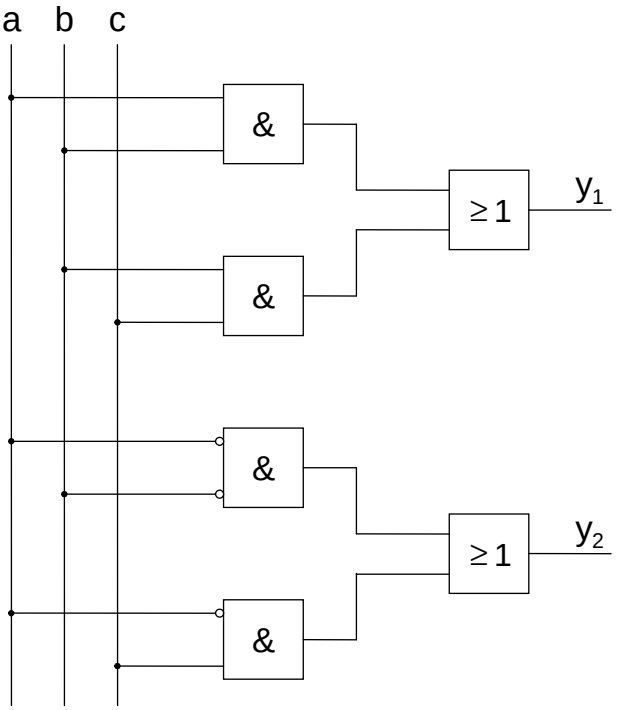


$y_1 = a + b$

y_2:

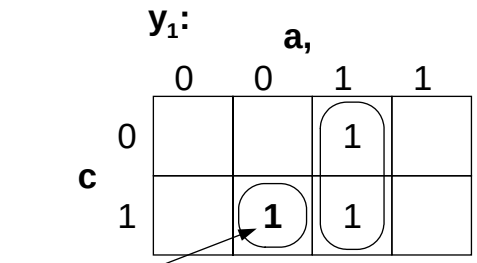


$y_2 = \bar{a}\bar{b} + \bar{a}c$



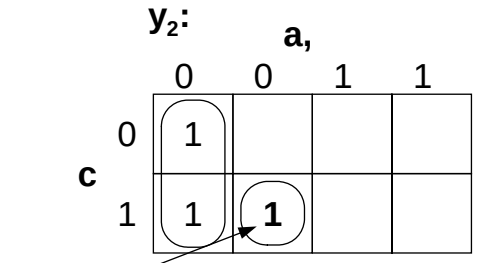
Unabhängige Realisierung ohne Koppelterme

y_1:

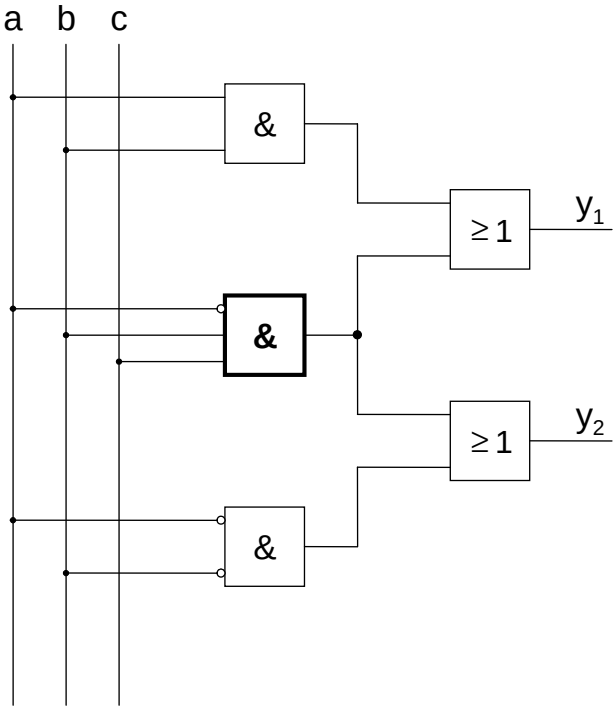


$y_1 = a + \bar{a}\mathbf{b}$

y_2:



$y_2 = \bar{a}\bar{b} + \bar{a}\mathbf{b}$



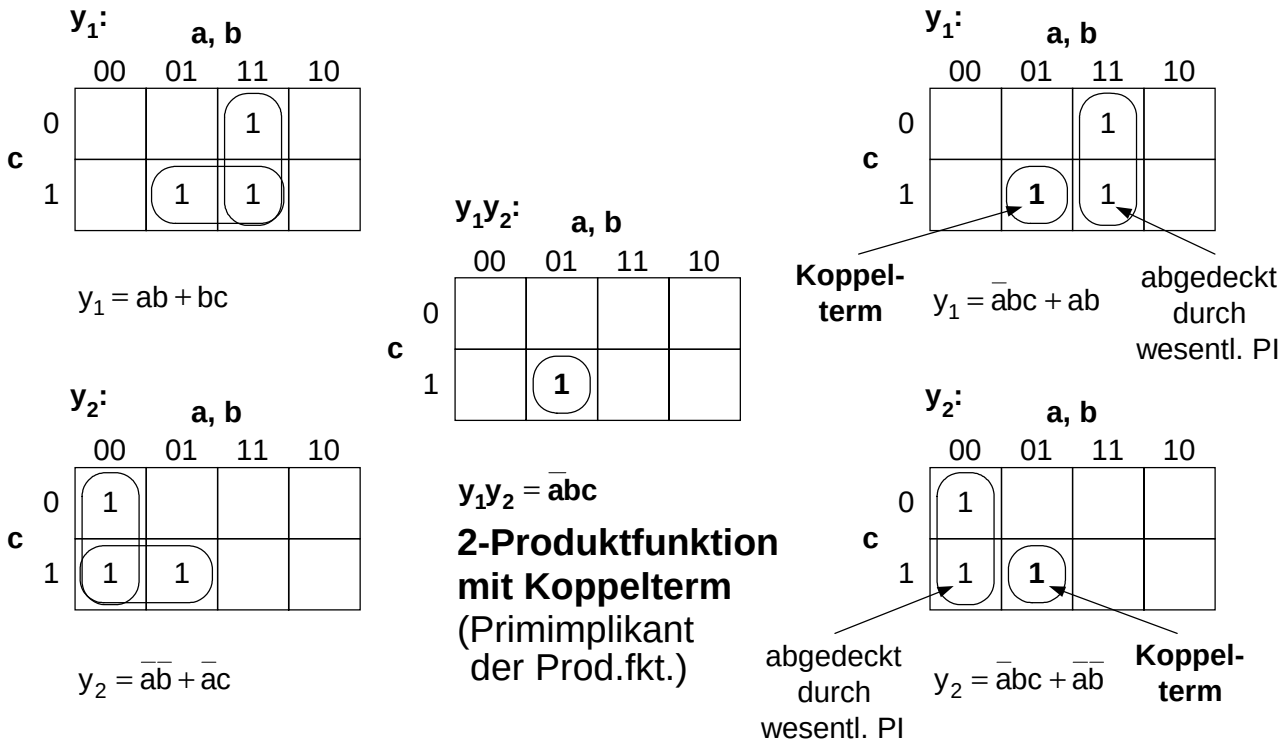
Minimierte Schaltung mit Koppelterm

Systematische Ermittlung der Koppelterme auch als Primimplikanten der m-Produktfunktionen

Vorgehensweise (KV-Diagramme oder Quine-McCluskey)

(1) Bilde alle Produktfunktionen aus m = 2, 3, ... n Ausgangsfunktionen (m-Produktfunktionen mit $2 \leq m \leq n$).
(2) Ermittle die Primimplikanten der Produktfunktionen (Kandidaten für Koppelterm).
(3) Ermittle die bereits durch die wesentlichen Primimplikanten außerhalb der Produktfunktionen abgedeckten Minterme der Einzelfunktionen.
(4) Decke die restlichen Minterme der Einzelfunktionen durch eine minimale Menge der Primimplikanten der Produktfunktionen ab.

<u>Beispiel:</u> m = 2



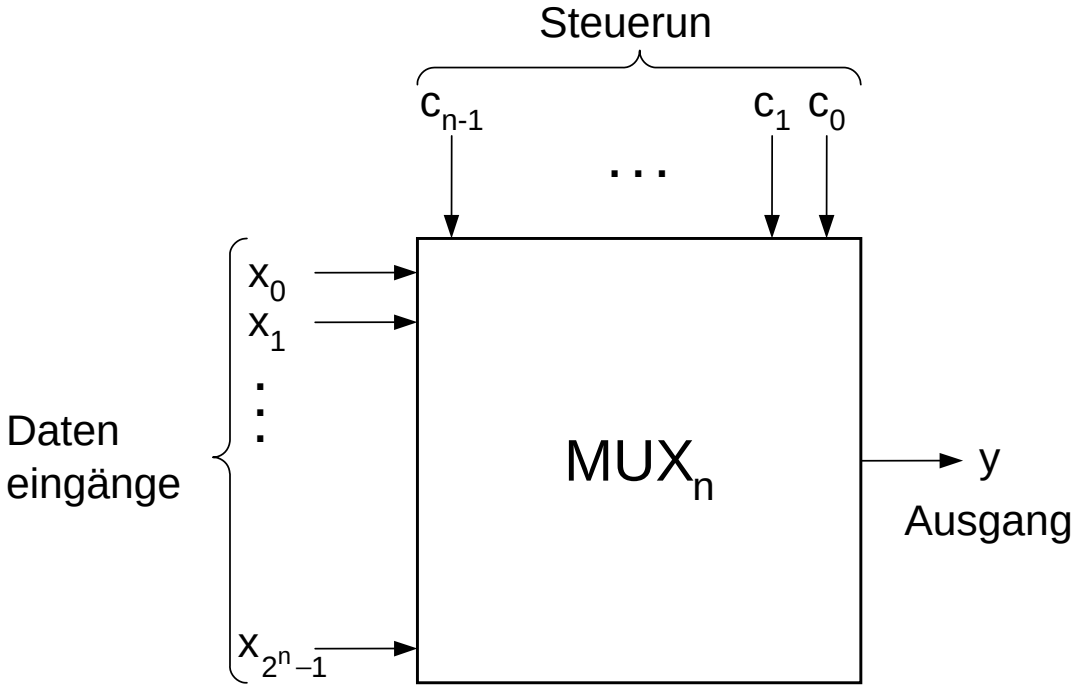
Sehr hohe Komplexität: 2^n-n-1 Produktfunktionen! Daher meist heuristische Verfahren (z. B. Espresso II), die nicht unbedingt das absolute Minimum, aber eine „gute“ Lösung finden.

4.4 Wichtige Schaltnetze

Multiplexer (MUX)

Multiplexer werden zur Auswahl von Datenquellen, also als Datenselektor, eingesetzt.

Sie beinhalten einen Decoder für die Steuersignale, so dass mit n Steuereingängen bis zu 2^n Quellen ausgewählt werden können.



$$y = \mathbf{mux}(c_{n-1},\ldots,c_0,x_0,\ldots x_{2^n-1}) = x_{(c_{n-1},\ldots,c_0)_2}$$

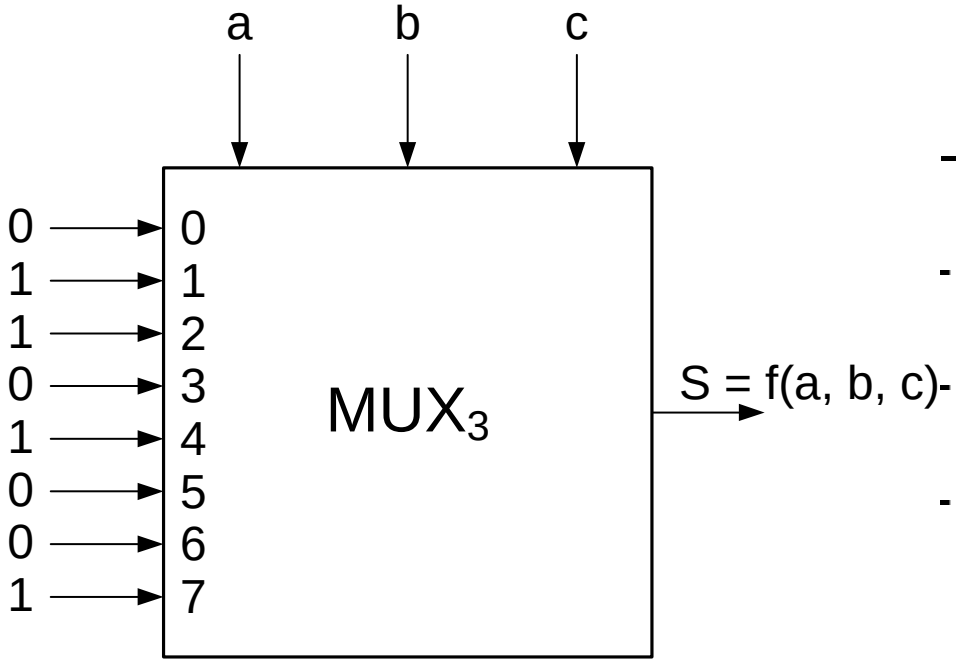
Derjenige Eingang x_i wird auf den Ausgang y durchgeschaltet, dessen Index i gleich den Steuereingängen interpretiert als Dualzahl ist.

Realisierung beliebiger Schaltfunktionen mit Multiplexern

$$f(c_{n-1},\ldots,c_1,c_0) = \sum_{i=0}^{2^n-1} m_i \cdot f_i$$

Prinzip: Jeder Eingang x_i entspricht einem Minterm m_i und wird genau dann auf 1 gesetzt, wenn er in der DKNF enthalten ist, sonst 0. Somit kann mit einem (2^n-zu-1)-Multiplexer jede beliebige Schaltfunktion mit n Variablen realisiert werden, indem die Eingänge entsprechend der zu realisierenden Schaltfunktion mit 0 bzw. 1 belegt werden.

Beispiel: Summe beim Volladdierer



#	a	b	c	S
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

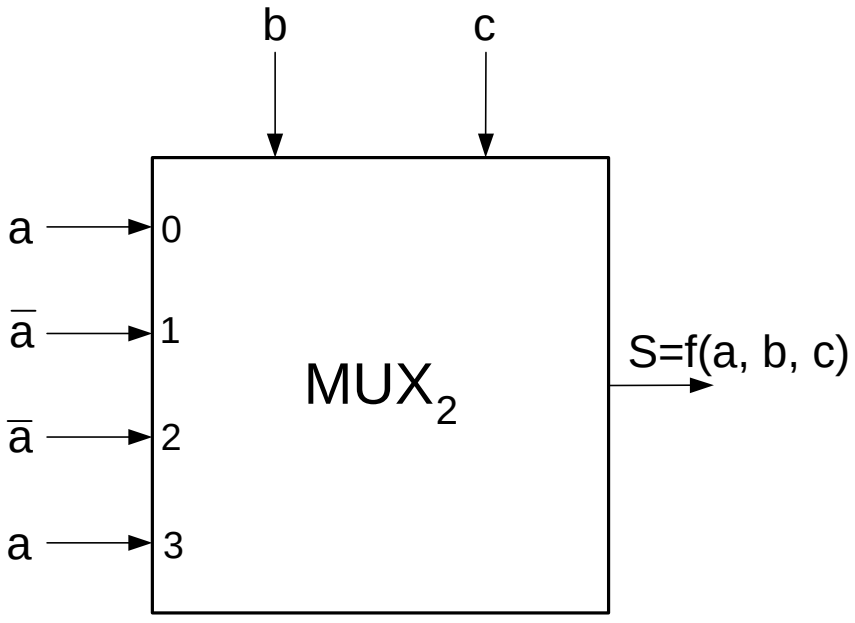
$$S = f(a,b,c) = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc$$
$$= m_1 + m_2 + m_4 + m_7$$

Hinweis: Es reicht auch ein (2^{n-1}-zu-1)-Multiplexer für alle n-stelligen Schaltfunktionen, denn die Adresse und der zugehörige Dateneingang sind UND-verknüpft.

Wird ein Steuersignal (hier a) geeignet auf die Multiplexereingänge geschaltet, muss der Multiplexer dann nur noch halb so groß sein. Denn jeder Eingang des Multiplexers kann für 2 Minterme $p\bar{a}$ und *pa*, d. h. zwei Zeilen der Wahrheitstafel, genutzt werden.

Er kann mit $\bar{a}$, *a*, 0 oder 1 stets so beschaltet werden, dass sich gerade der gewünschte Funktionswert f ergibt.

Beispiel: Summe beim Volladdierer (vgl. oben)



Multiplexer als Integrierte Schaltung

Der TTL-Baustein SN74251und seine Varianten sind z.B. 8-zu-1-Multiplexer mit einem Tristate-Ausgang (s. unten).

MOTOROLA

SN54/74LS251

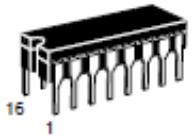
8-INPUT MULTIPLEXER WITH 3-STATE OUTPUTS

The TTL/MSI SN74LS251 is a high speed 8-Input Digital Multiplexer. It provides, in one package, the ability to select one bit of data from up to eight sources. The LS251 can be used as a universal function generator to generate any logic function of four variables. Both assertion and negation outputs are provided.

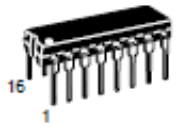
- Schottky Process for High Speed
- Multifunction Capability
- On-Chip Select Logic Decoding
- Inverting and Non-Inverting 3-State Outputs
- Input Clamp Diodes Limit High Speed Termination Effects

8-INPUT MULTIPLEXER
WITH 3-STATE OUTPUTS

LOW POWER SCHOTTKY



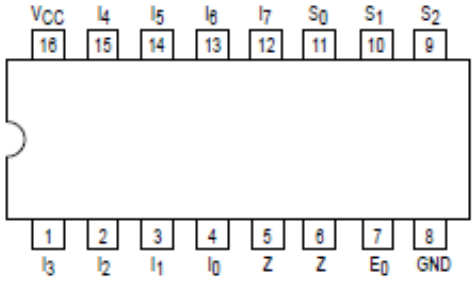
J SUFFIX
CERAMIC
CASE 620-09



N SUFFIX
PLASTIC
CASE 648-08

D SUFFIX
SOIC
CASE 751B-03

CONNECTION DIAGRAM DIP (TOP VIEW)



PIN NAMES

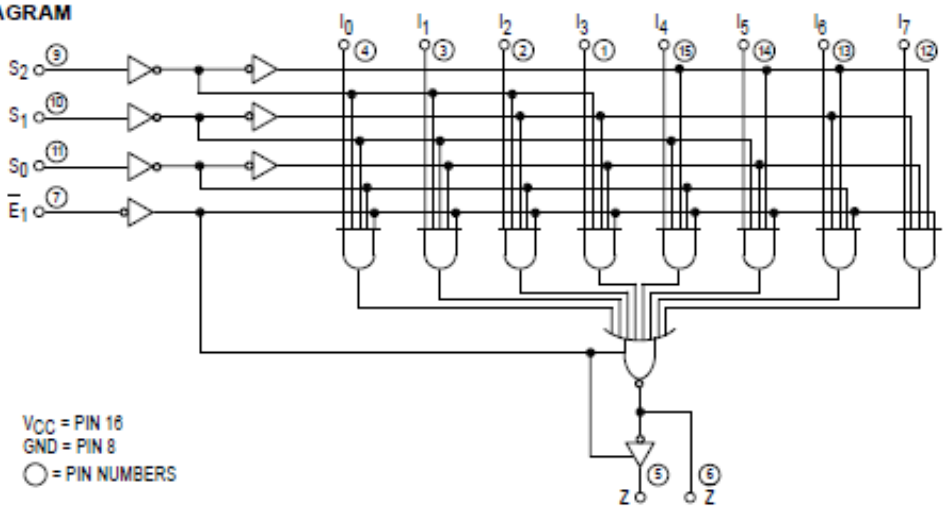
		LOADING (Note a) HIGH	LOW
$\overline{S_0}$–$\overline{S_2}$	Select Inputs	0.5 U.L.	0.25 U.L.
$\overline{E_0}$	Output Enable (Active LOW) Inputs	0.5 U.L.	0.25 U.L.
I_0–I_7	Multiplexer Inputs	0.5 U.L.	0.25 U.L.
$\overline{Z}$	Multiplexer Output	65 U.L.	15 U.L.
Z	Complementary Multiplexer Output	65 U.L.	15 U.L.

NOTES:
a. 1 TTL Unit Load (U.L.) = 40 µA HIGH/1.6 mA LOW.

ORDERING INFORMATION

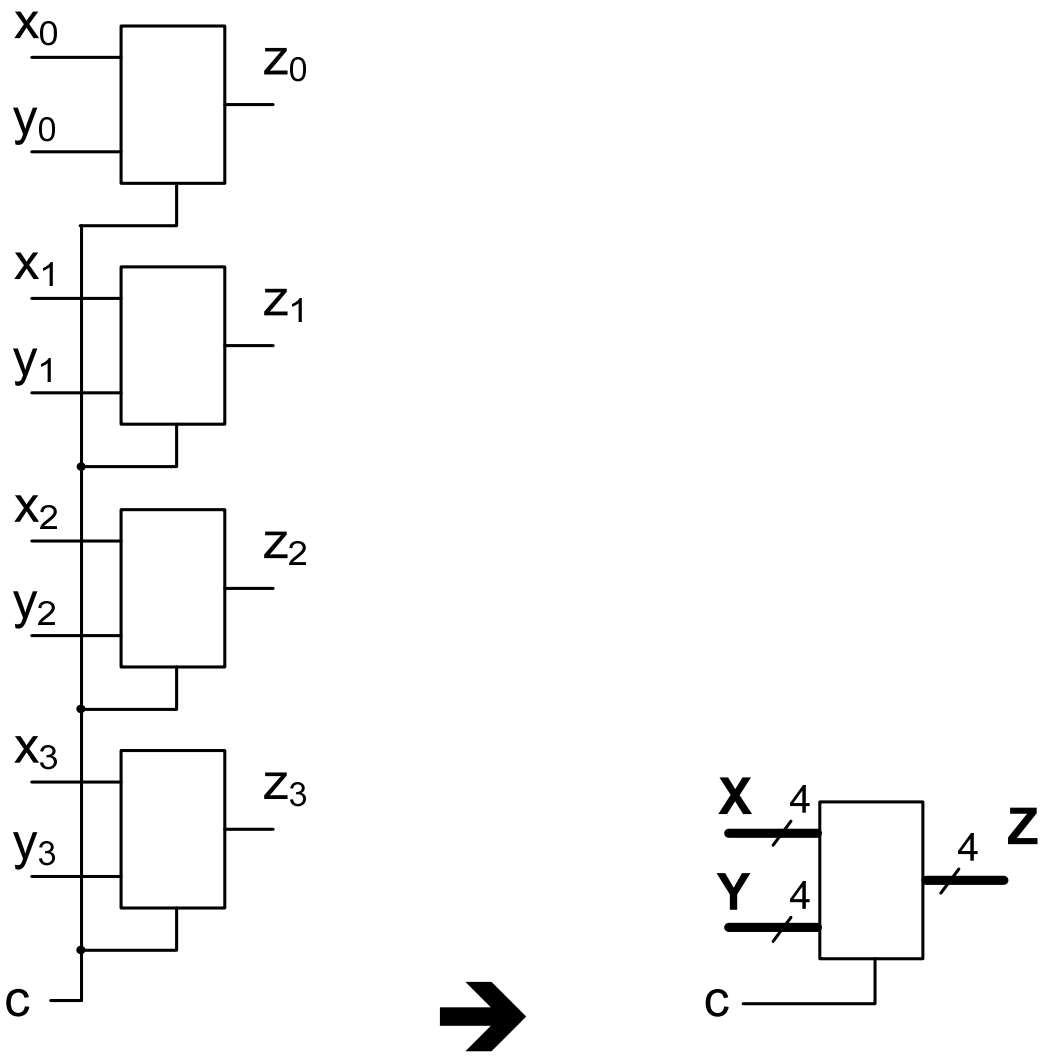
SN54LSXXXJ	Ceramic
SN74LSXXXN	Plastic
SN74LSXXXDW	SOIC

LOGIC DIAGRAM



Mehrfach-Multiplexer

Multiplexer werden auch zur Auswahl von ganzen Datenvektoren eingesetzt und dazu mehrere 1-Bit-Mulitplexer parallel geschaltet, die dann gemeinsam von einem Steuersignal, hier c, gesteuert werden.



Für parallele Multiplexer (z.B. bei Bitvektoren) reicht ein 1-aus-n-Decoder für die Steuersignale.

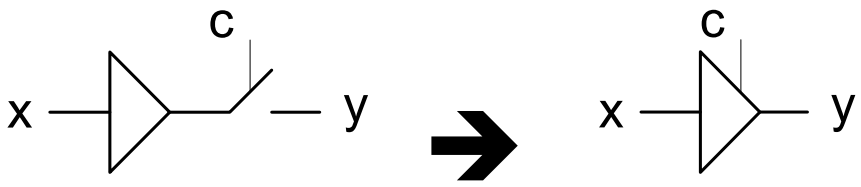
Anstatt für jedes Bit einen Multiplexer zu malen, wird ein Multiplexer auch für die ganzen Bitvektoren gezeichnet.

Multiplexer und Busse

Für größere Multiplexer werden auch größere Gatter und damit viele Leitungen und Chipfläche benötigt. Gerade in Mikroprozessoren und anderen Rechenschaltungen werden deshalb so genannte **Busse** eingesetzt.

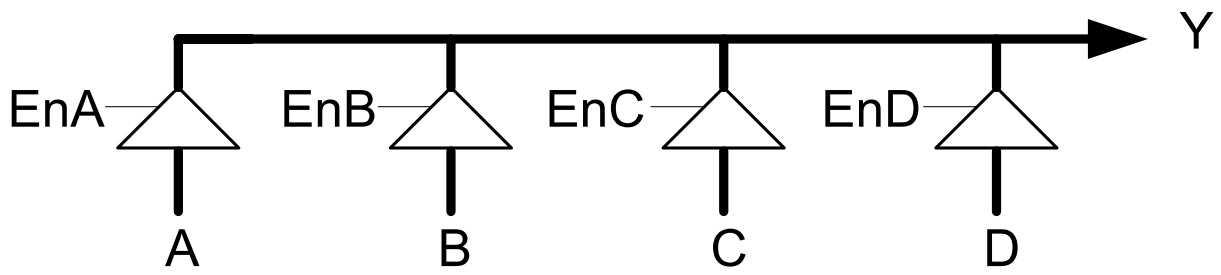
Bei Bussen werden Leitungen mehrfach für die Übertragung von alternativen Datenquellen genutzt.

Dafür werden an den Ausgängen der Datenquellen **Tristate-Treiber** (Three-State-) eingesetzt. D.h., mit einem Steuersignal wird der Ausgang aktiv bzw. inaktiv (passiv, hochohmig) geschaltet. Nur im aktiven Zustand hat der Treiber die Möglichkeit, den Leitungszustand auf ´0´ oder ´1´ zu setzen.



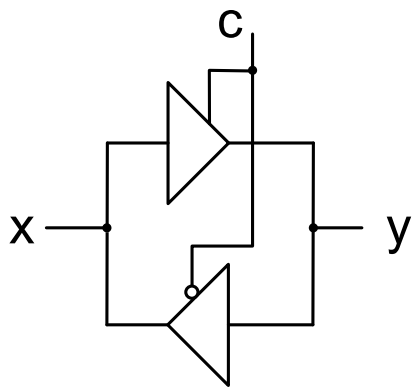
Der inaktive, hochohmige Zustand wird in den logischen Gleichungen bzw. Funktionstabellen meist mit ´Z´ gekennzeichnet.

Mit solchen Tristate-Treibern lassen sich dann ganz elegant Multiplexer und Busse realisieren. Voraussetzung ist, dass immer genau ein Treiber pro Leitung bzw. Satz von Treibern pro Bus aktiviert wird.



Bidirektionale Busse

Gerade an den Schnittstellen von Mikroprozessoren und ihrer Peripherie sowie innerhalb der Schaltungen müssen Signale häufig in beide Richtungen ausgetauscht werden. Dazu werden bidirektionale Treiber, auch Transceiver genannt, bzw. bidirektionale Busse verwendet.



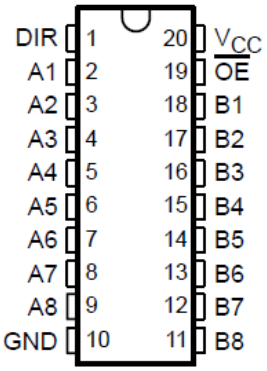
SN54LS245, SN74LS245
OCTAL BUS TRANSCEIVERS
WITH 3-STATE OUTPUTS

SDLS146A – OCTOBER 1976 – REVISED FEBRUARY 2002

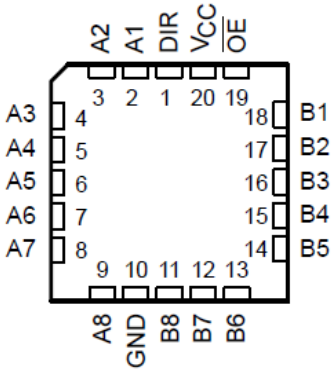
- **3-State Outputs Drive Bus Lines Directly**
- **PNP Inputs Reduce dc Loading on Bus Lines**
- **Hysteresis at Bus Inputs Improves Noise Margins**
- **Typical Propagation Delay Times Port to Port, 8 ns**

TYPE	I_{OL} (SINK CURRENT)	I_{OH} (SOURCE CURRENT)
SN54LS245	12 mA	–12 mA
SN74LS245	24 mA	–15 mA

SN54LS245 . . . J OR W PACKAGE
SN74LS245 . . . DB, DW, N, OR NS PACKAGE
(TOP VIEW)



SN54LS245 . . . FK PACKAGE
(TOP VIEW)



description

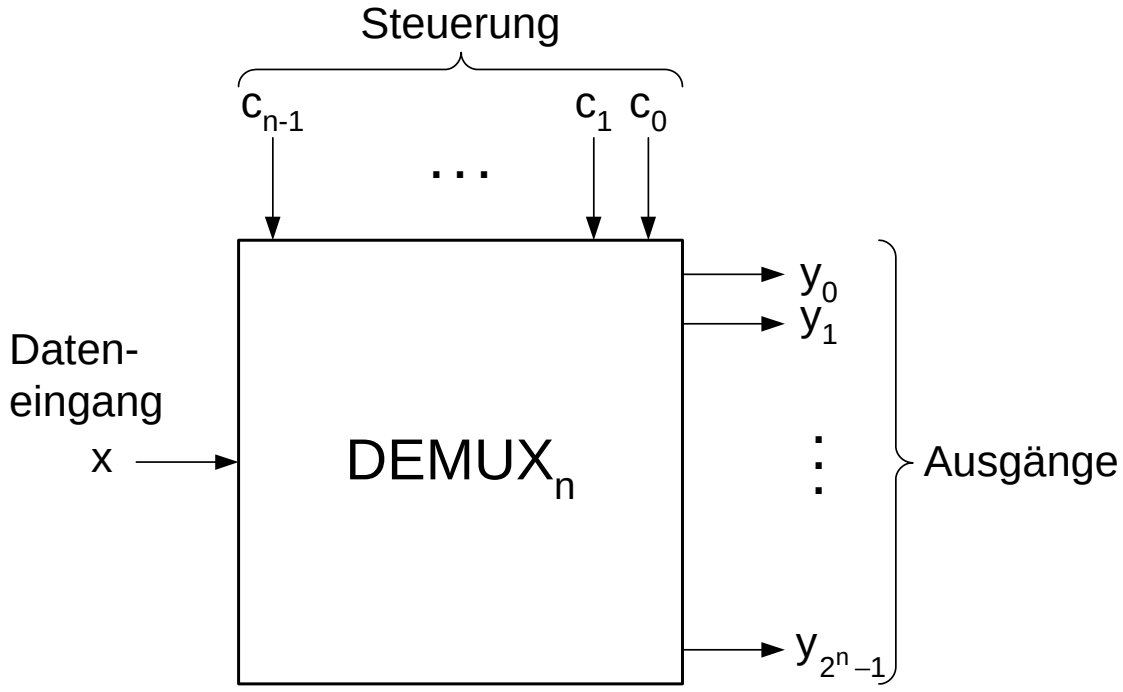
These octal bus transceivers are designed for asynchronous two-way communication between data buses. The control-function implementation minimizes external timing requirements.

The devices allow data transmission from the A bus to the B bus or from the B bus to the A bus, depending on the logic level at the direction-control (DIR) input. The output-enable ($\overline{OE}$) input can disable the device so that the buses are effectively isolated.

Demultiplexer (DEMUX)

Demultiplexer dienen der selektiven Verteilung von Datenströmen an alternative Datensenken.

Mit n Steuereingängen kann eine Datenquelle (hier x) auf bis zu 2^n Ausgänge (y_0, …, y_{2^n-1}) geschaltet werden.



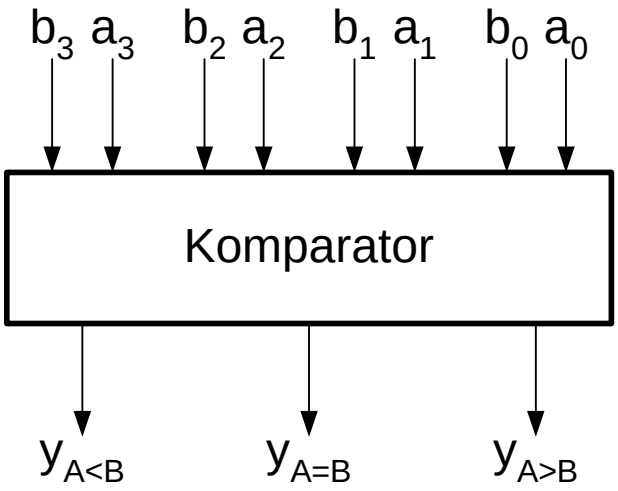
$$\mathbf{demux}(x, c_{n-1}, \ldots, c_0) = (y_0, \ldots, y_{2^n-1})$$

$$\text{mit: } y_i = \begin{cases} x, & \text{falls } i = (c_{n-1}, \ldots, c_0)_2 \\ 0, & \text{sonst (neutrales Element)} \end{cases}$$

Eingang x wird also auf denjenigen Ausgang y_i durchgeschaltet, dessen Index i gleich den Steuereingängen interpretiert als Dualzahl ist. Die übrigen Ausgänge y_i sind Null.

Komparatoren für (pos.) Dualzahlen

4-Bit Komparator (Vergleicher)



$$\mathbf{komp}(a_{N-1},\ldots,a_0,b_{N-1},\ldots,b_0) = (y_{A<B}, y_{A=B}, y_{A>B})$$

mit: $y_{A<B} = 1$ falls $(a_{N-1}, \ldots, a_0)_2 < (b_{N-1}, \ldots, b_0)_2$, sonst 0
$y_{A=B} = 1$ falls $(a_{N-1}, \ldots, a_0)_2 = (b_{N-1}, \ldots, b_0)_2$, sonst 0
$y_{A>B} = 1$ falls $(a_{N-1}, \ldots, a_0)_2 > (b_{N-1}, \ldots, b_0)_2$, sonst 0

Realisierung durch Subtrahierer oder direkt als Schaltnetz möglich

Bei einer Realisierung als zweistufiges Schaltnetz wird bei größeren Wortbreiten der Realisierungsaufwand recht groß. Deswegen wird der Vergleicher kaskadiert, also der Vergleich stufenweise durchgeführt und die Ergebnisse zusammengefasst.

$\rightarrow$ mehrstufige Logik

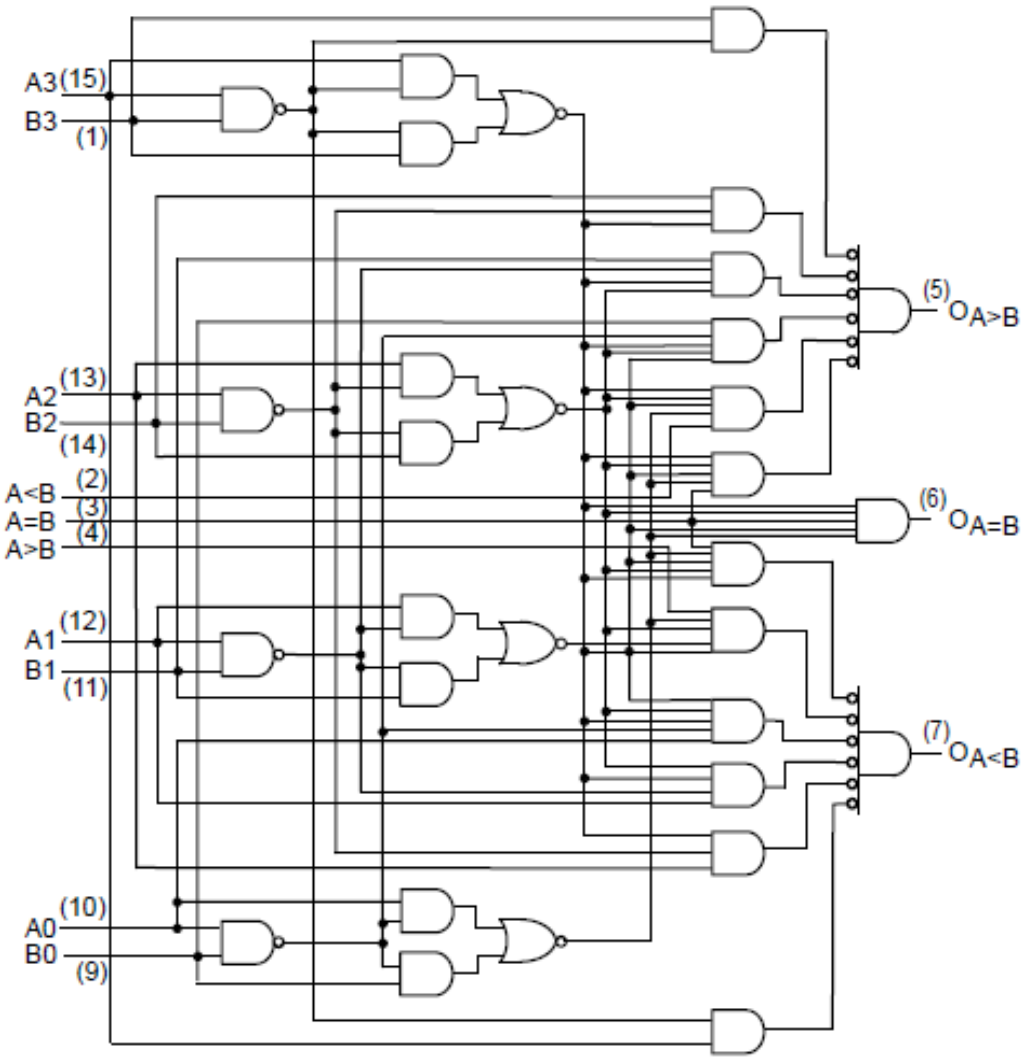
Realisierung als TTL-IC

Ein 4-Bit-Komparator in TTL-Technik ist der SN7485.

TRUTH TABLE

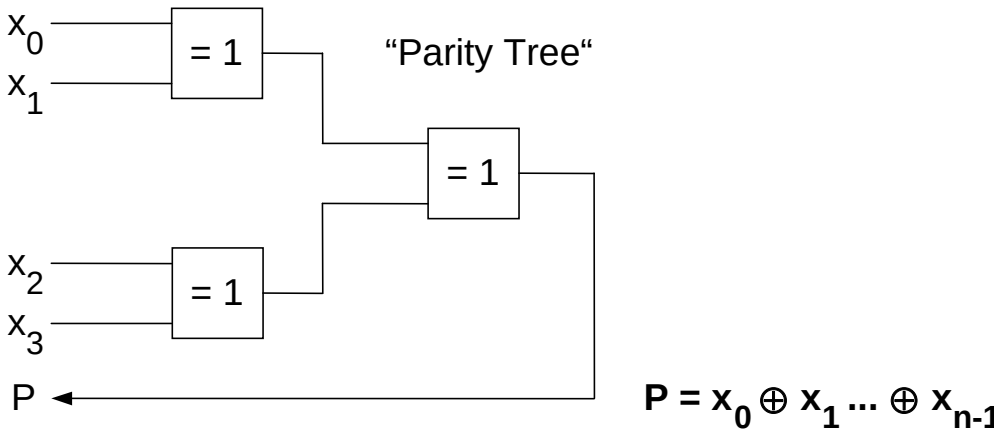
COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
A_3,B_3	A_2,B_2	A_1,B_1	A_0,B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$O_{A>B}$	$O_{A<B}$	$O_{A=B}$
$A_3>B_3$	X	X	X	X	X	X	H	L	L
$A_3<B_3$	X	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2>B_2$	X	X	X	X	X	H	L	L
$A_3=B_3$	$A_2<B_2$	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1>B_1$	X	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1<B_1$	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B1$	$A_0>B_0$	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0<B_0$	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	L	L	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	H	L	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	X	X	H	L	L	H
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	H	L	L	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	L	L	H	H	L

H = HIGH Level
L = LOW Level
X = IMMATERIAL



Paritätsgenerator/-prüfer zur Fehlererkennung

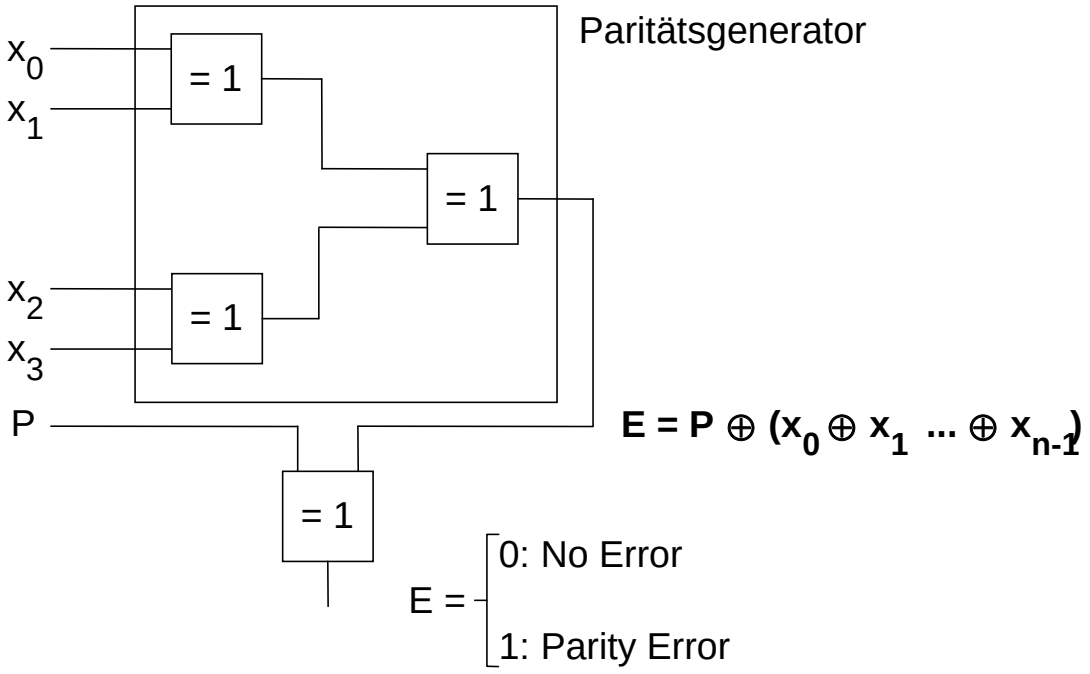
Paritätsgenerator für gerade Parität (even parity):



Paritätsbit P ergänzt n Datenbits x_0, x_1 ... x_{n-1} so um ein weiteres Bit, dass immer eine *gerade* Anzahl von Einsen entsteht.

$\Rightarrow$ Einfach-Fehler im gesamten Bitvektor x_0, x_1, ... x_{n-1}, P (inkl. Paritätsbit) durch Paritätsprüfung erkennbar.

Paritätsprüfer für gerade Parität



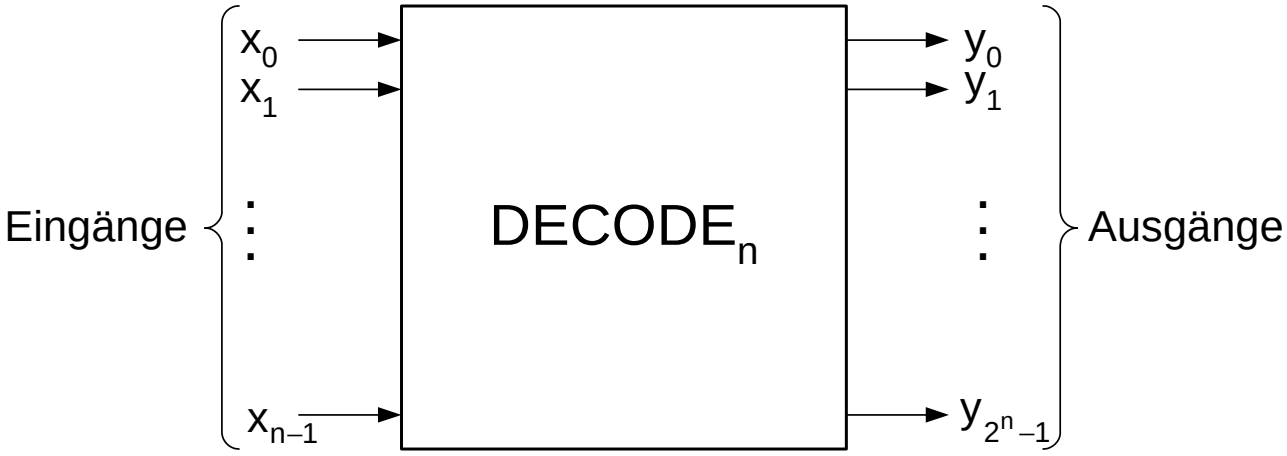
Ungerade Parität: analog mit Ergänzung auf *ungerade* Anzahl von Einsen (odd parity).

Code-Umsetzer

Schaltnetze können allgemein zur Codeumsetzung (Codierung/Decodierung) zwischen Codes eingesetzt werden. Solche Codeumsetzer überführen allgemein eine Menge von Eingangswerten auf eine Menge von Ausgangswerten. Sie realisieren in der Regel eine Vektorfunktion, z. B.:

- Dezimal-zu-BCD-Decoder
- BCD-zu-Dezimal-Decoder
- BCD-zu-7-Segment-Decoder
- BCD-Code in Gray-Code und umgekehrt

Decodierer (1-aus-n-Decoder)



$$\mathbf{decode(x_0,\ldots,x_{n-1}) = (y_0,y_1,\ldots,y_{2^n-1})}$$

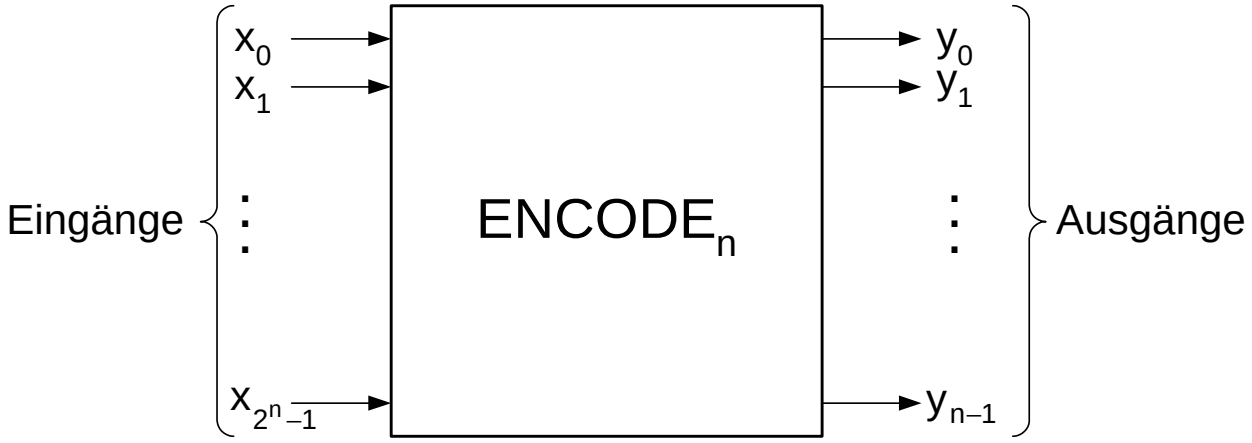
$$\text{mit: } y_i = \begin{cases} 1, & \text{falls } (x_{n-1},\ldots,X_0)_2 = i \\ 0, & \text{sonst} \end{cases}$$

Diejenige Ausgangsleitung y_i wird 1, deren Index i als Dualzahl am Eingang anliegt (1-aus-n-Code). Alle anderen sind 0.

Ein 1-aus-n-Decoder wählt z.B. in Multiplexern aus oder aktiviert Treiber von Bussen oder Speicherstellen.

Beispiel: **Codierer (Encoder)**

bestimmt die Bitstellennummer eines auf ´1´ gesetzten Bits.



$$\mathbf{encode(x_0, x_1, \ldots, x_{2^n-1}) = (y_0, y_1, \ldots, y_{n-1})}$$

mit: $(y_{n-1}, \ldots, y_0)_2 = i$, wenn $x_i = 1$

Voraussetzung: Nur *genau ein* x_i ist 1, alle anderen 0 (1-aus-n Code)

Prioritäts-Codierer: Sind mehrere Eingänge gleichzeitig 1, bestimmt der mit dem höchsten Index i den Ausgang.

Beispiel: 8-zu-3-Prioritäts-Encoder

(Ausgang d unterscheidet zwischen $x_0 = 1$ und alle Eingänge 0)

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	a	b	c	d
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	0	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

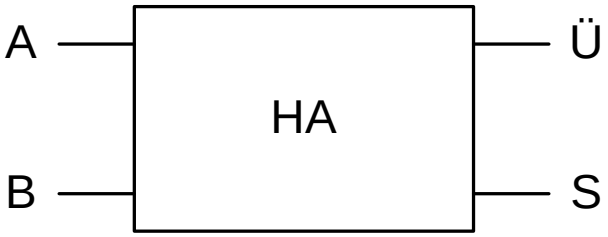
4.5 Rechenschaltungen

4.5.1 Addierer

Halbaddierer

Ein Halbaddierer addiert zwei einstellige Binärzahlen zu einem Summenbit S und einem Übertrag Ü.

A	B	S	Ü
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

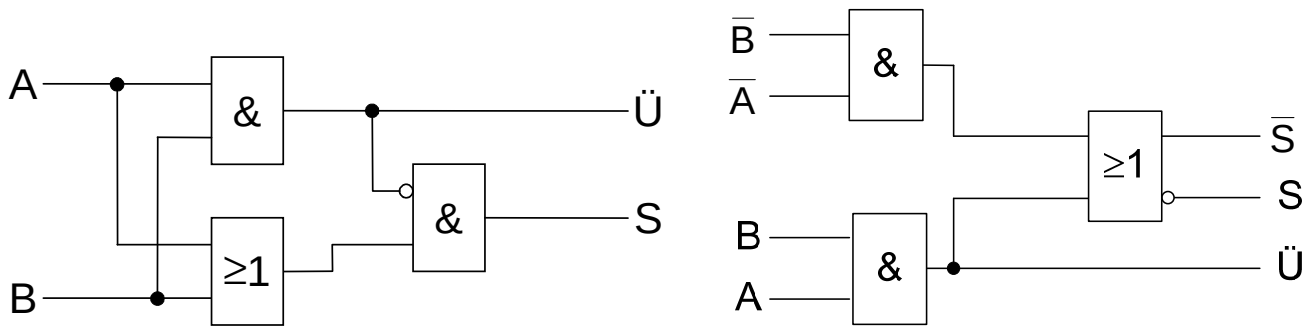


Schaltfunktionen dazu:

$$S = A \oplus B = \overline{A}\,B + A\,\overline{B} \quad = (A+B)\cdot(\overline{AB})$$

$$Ü = AB$$

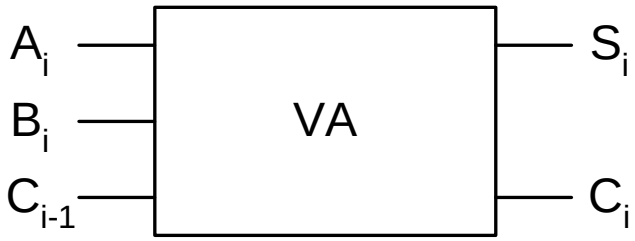
Realisierung mit Grundgattern unter Nutzung von Koppelterm AB und alternative Realisierung mit $\mathbf{S} = \overline{\mathbf{AB} + \overline{\mathbf{AB}}}$:



Volladdierer

Ein Volladdierer berücksichtigt auch einen Übertrag C_{i-1} (Carry) von der niederwertigeren Stelle i-1.

Durch Kaskadierung können so beliebig breite Addierer realisiert werden.



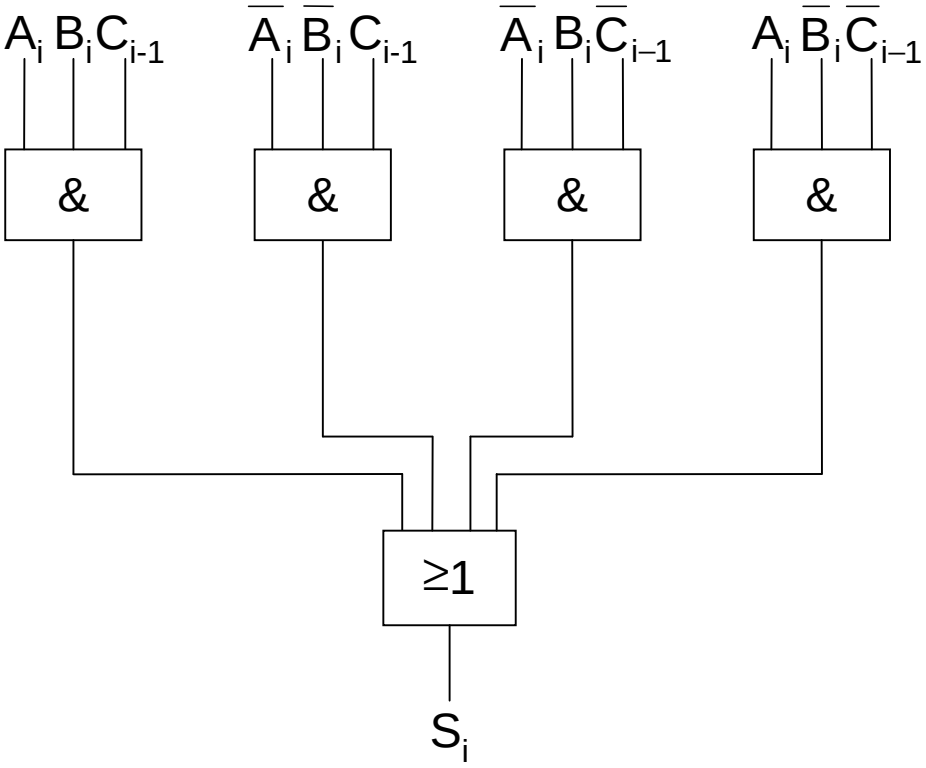
A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

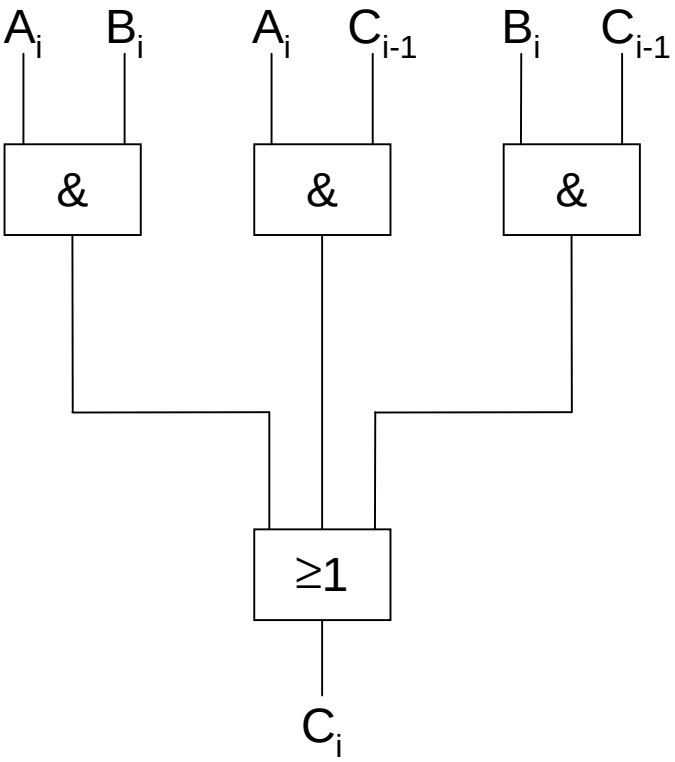
Schaltfunktion:

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$
$$= A_i B_i C_{i-1} + \overline{A}_i \overline{B}_i C_{i-1} + \overline{A}_i B_i \overline{C}_{i-1} + A_i \overline{B}_i \overline{C}_{i-1}$$

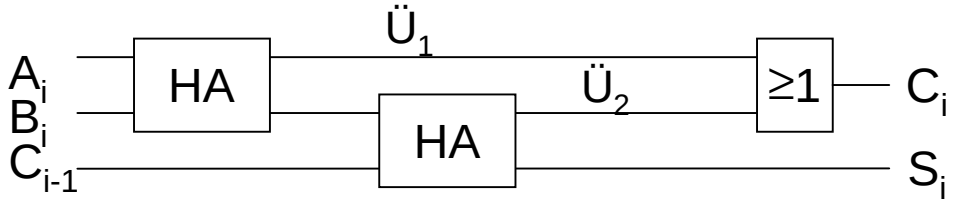
$$C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1}$$

Schaltung gemäß DNF

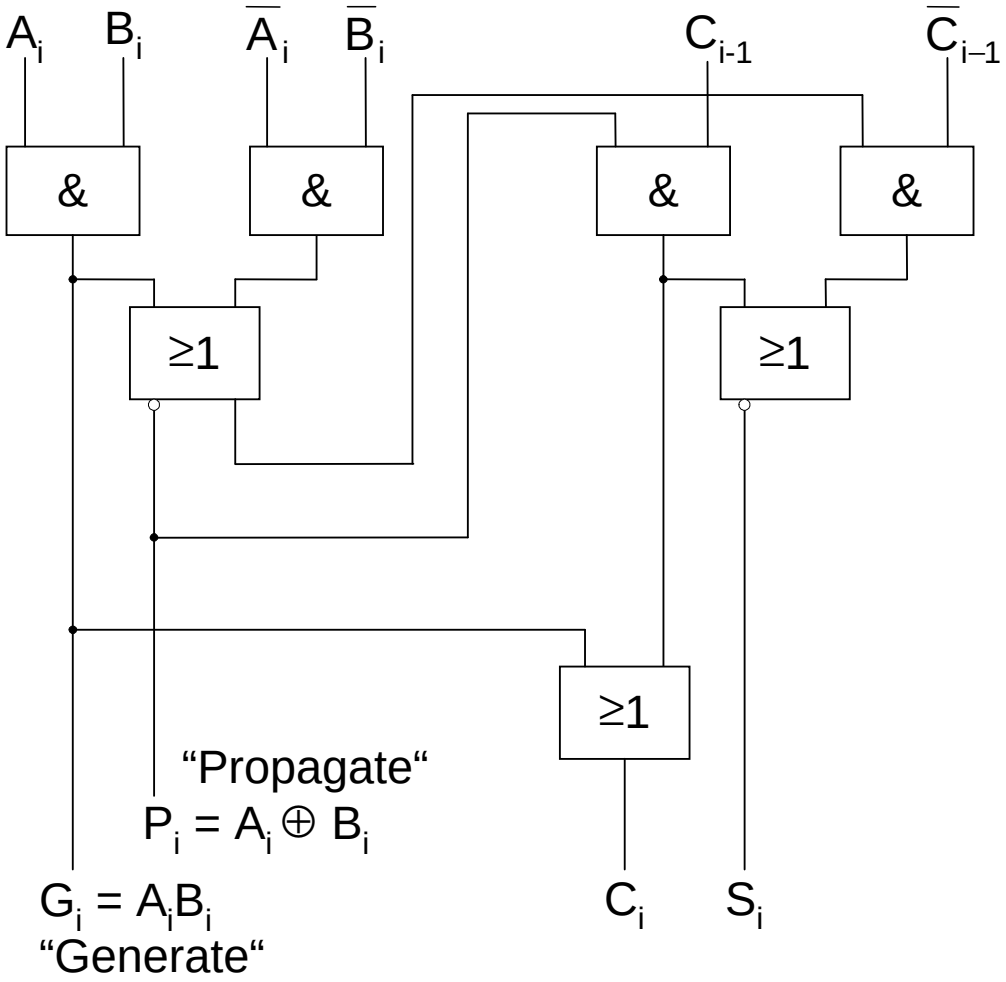




Volladdierer aus 2 Halbaddierern



Beispiel:



Vergleich:

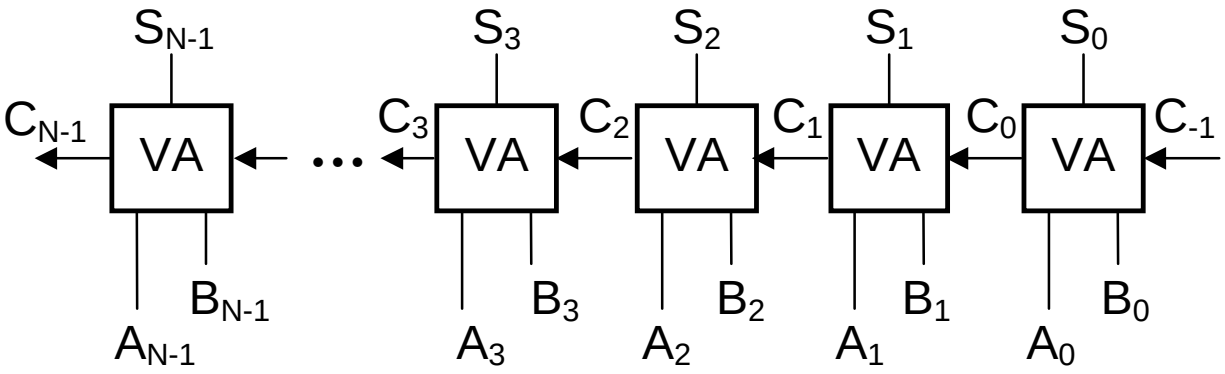
Volladdierer: 9 Gatter; 2 Gatterlaufzeiten: 2 τ

2 Halbaddierer: 7 Gatter; 4 Gatterlaufzeiten: 4 τ

Anmerkung: Hier vereinfachte Laufzeitbetrachtung durch EInheitsverzögerung τ (Unit Delay) pro Gatter.

4.5.2 Paralleladdierer

Paralleladdierer bilden die Summen der jeweiligen Binärstellen parallel und lassen den Übertrag durch die Stufen von der niederwertigsten bis zur höchstwertigen Stelle fortpflanzen (Ripple-Carry- oder Carry-Chain-Addierer).



Nachteil: Es muss gewartet werden, bis der Übertrag bis zur (N-1)-ten Stelle, also durch N Stufen, durchgelaufen ist.

Additionszeit: $2N\ \tau$ (Volladdierer-Realisierung)

bzw. $(2N + 2)\ \tau$ (Halbaddierer-Realisierung)

Voraus-Ermittlung von Überträgen
(Parallele Überlauflogik,"carry look-ahead")

Ziel: Schnellere Generierung der Carry-Signale

Ansatz: Hilfsschaltung zur parallelen Generierung der Carry-Signale unter Ausnutzung der Hilfsvariablen G_i und P_i der Zwischenergebnisse

Übertrag der Stufe i:

$$C_i = A_i B_i + (A_i \oplus B_i) C_{i-1}$$
$$= G_i + P_i C_{i-1}$$

$G_i = A_i B_i$ gibt an, ob Carry erzeugt wird *("generate")*

$P_i = A_i \oplus B_i$ gibt an, ob Carry weitergegeben (= 1) oder absorbiert (= 0) wird *("propagate")*

Leicht aus Halbaddierer-Realisierung zu gewinnen (s. o.).

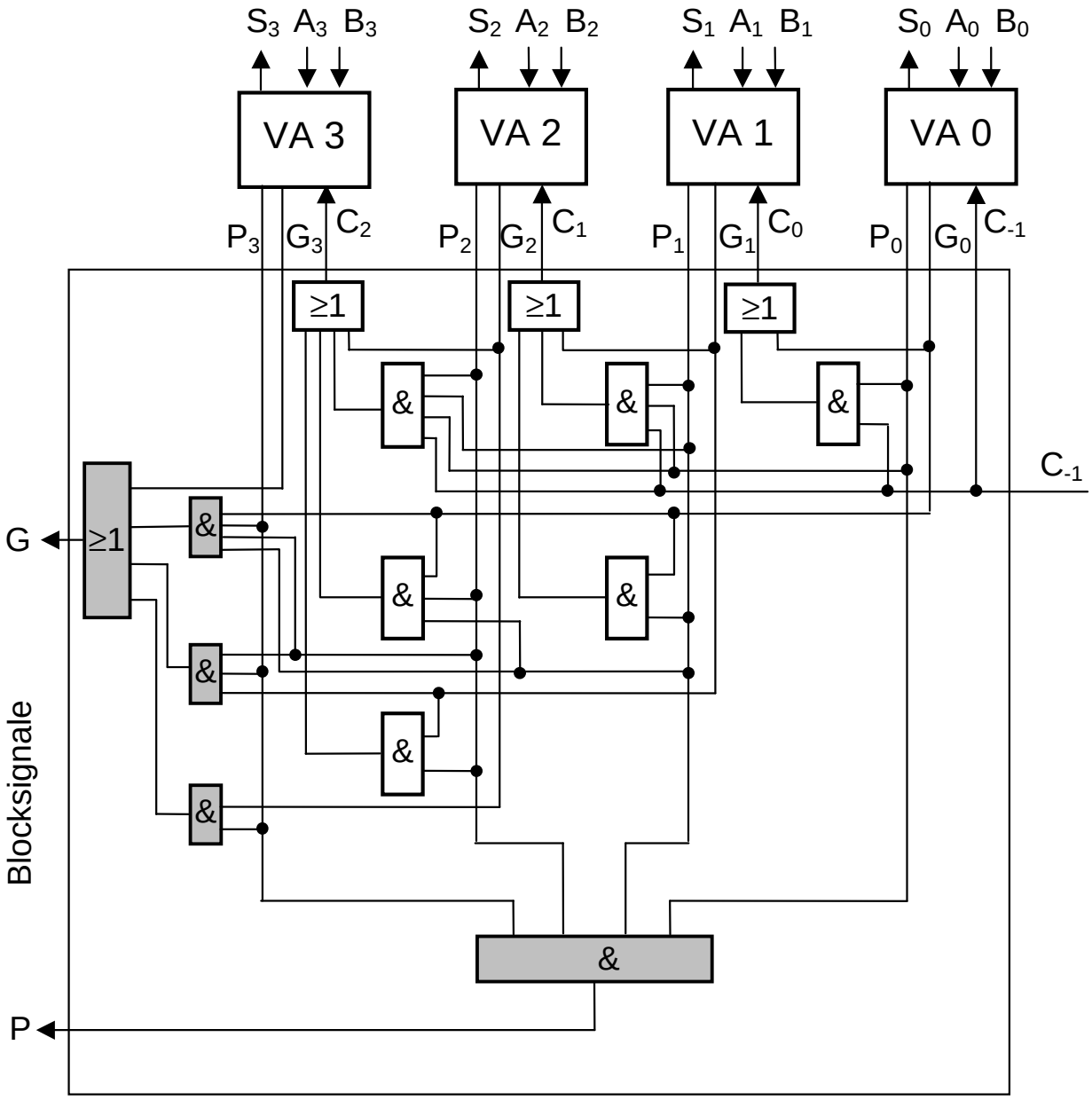
Für einzelne Überträge gilt damit:

$C_0 = G_0 + P_0C_{-1}$
$C_1 = G_1 + P_1C_0 = G_1 + P_1G_0 + P_1P_0C_{-1}$
$C_2 = G_2 + P_2C_1 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{-1}$
$C_3 = G_3 + P_3C_2 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{-1}$

-
-
-

Alle Überträge lassen sich damit in nur 2 Gatterlaufzeiten vorab ermitteln.

Beispiel: ÜES (Übertragungs-Ermittlungs-Schaltung, Carry-Look-Ahead-Generator) für 4-Bit Addierer mit Propagate/Generate-Ausgängen



Ripple Carry: 8 τ, 36 Gatter (Volladdierer)
10 τ, 28 Gatter (Halbaddierer)

Carry look-ahead: 6 τ, 28 + 14 = 42 Gatter (Halbaddierer)

→ Der Aufwand wächst von Stelle zu Stelle überproportional.

Größere Wortlängen

Die vollständige Vorabermittlung des Übertrags ist theoretisch möglich, aber zu aufwendig bei großen Wortbreiten.

Deshalb Kaskadierung von ÜES (meist Blöcke mit 4 Bit)

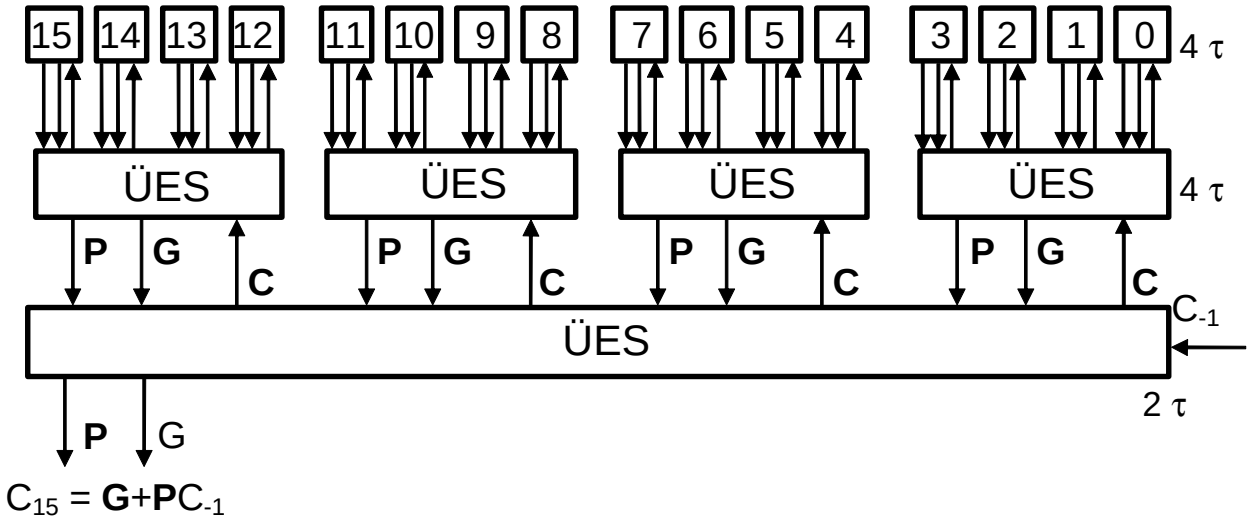
$$C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{-1}$$
$$= \mathbf{G} + \mathbf{P}C_{-1}$$

G = $G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$ **Block-**Generate

P = $P_3P_2P_1P_0$ **Block-**Propagate

Zusammenfassung von Blöcken mit *gleicher* ÜES für breite Addierer möglich.

Beispiel: 16-Bit Addierer mit 4-Bit ÜES (mit weiterem 4-Bit ÜES zu 64 Bit kaskadierbar)



Addierzeit: 10 τ Aufwand: 182 Gatter

Ripple-Carry: 32 τ Aufwand: 144 Gatter (Volladdierer)

34 τ Aufwand: 112 Gatter (Halbaddierer)

Realisierung als Integrierte Schaltung

Die Realisierung als TTL-IC ist der SN7483-Baustein und seine Varianten.

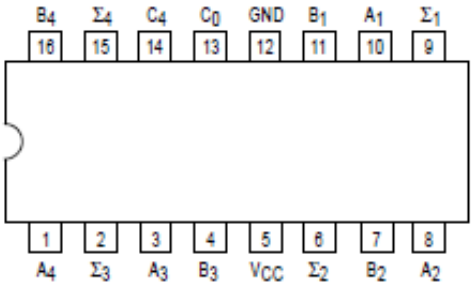
MOTOROLA

SN54/74LS83A

4-BIT BINARY FULL ADDER WITH FAST CARRY

The SN54/74LS83A is a high-speed 4-Bit binary Full Adder with internal carry lookahead. It accepts two 4-bit binary words (A_1-A_4, B_1-B_4) and a Carry Input (C_0). It generates the binary Sum outputs ($\Sigma_1-\Sigma_4$) and the Carry Output (C_4) from the most significant bit. The LS83A operates with either active HIGH or active LOW operands (positive or negative logic). The SN54/74LS283 is recommended for new designs since it is identical in function with this device and features standard corner power pins.

CONNECTION DIAGRAM DIP (TOP VIEW)



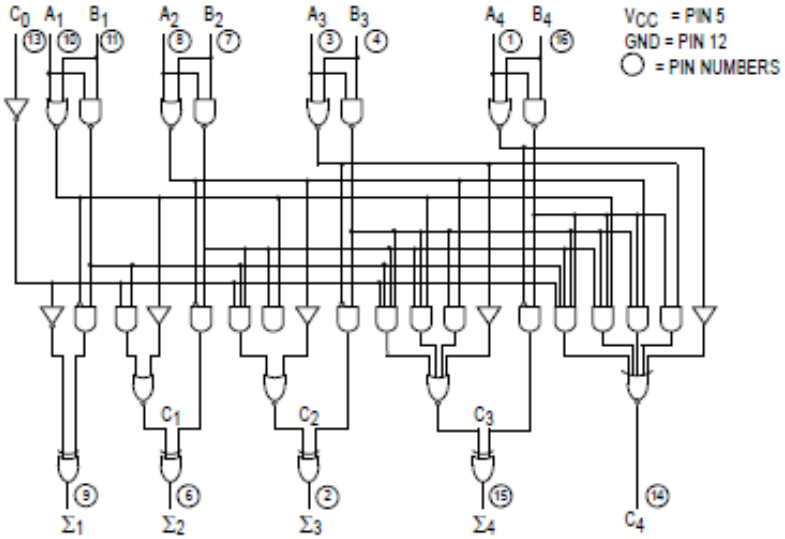
NOTE:
The Flatpak version has the same pinouts (Connection Diagram) as the Dual In-Line Package.

PIN NAMES

		LOADING (Note a) HIGH	LOW
A_1-A_4	Operand A Inputs	1.0 U.L.	0.5 U.L.
B_1-B_4	Operand B Inputs	1.0 U.L.	0.5 U.L.
C_0	Carry Input	0.5 U.L.	0.25 U.L.
$\Sigma_1-\Sigma_4$	Sum Outputs (Note b)	10 U.L.	5 (2.5) U.L.
C_4	Carry Output (Note b)	10 U.L.	5 (2.5) U.L.

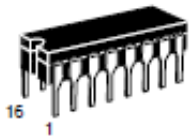
NOTES:
a) 1 TTL Unit Load (U.L.) = 40 µA HIGH/1.6 mA LOW.
b) The Output LOW drive factor is 2.5 U.L. for Military (54) and 5 U.L. for Commercial (74) Temperature Ranges.

LOGIC DIAGRAM

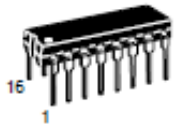


4-BIT BINARY FULL ADDER WITH FAST CARRY

LOW POWER SCHOTTKY



J SUFFIX
CERAMIC
CASE 620-09



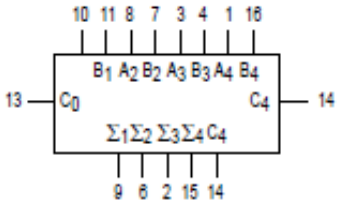
N SUFFIX
PLASTIC
CASE 648-08

D SUFFIX
SOIC
CASE 751B-03

ORDERING INFORMATION

SN54LSXXJ	Ceramic
SN74LSXXN	Plastic
SN74LSXXD	SOIC

LOGIC SYMBOL



4.5.3 Subtraktion

Prinzipien: - Subtraktion analog zu Addierer in speziellem Schaltnetz

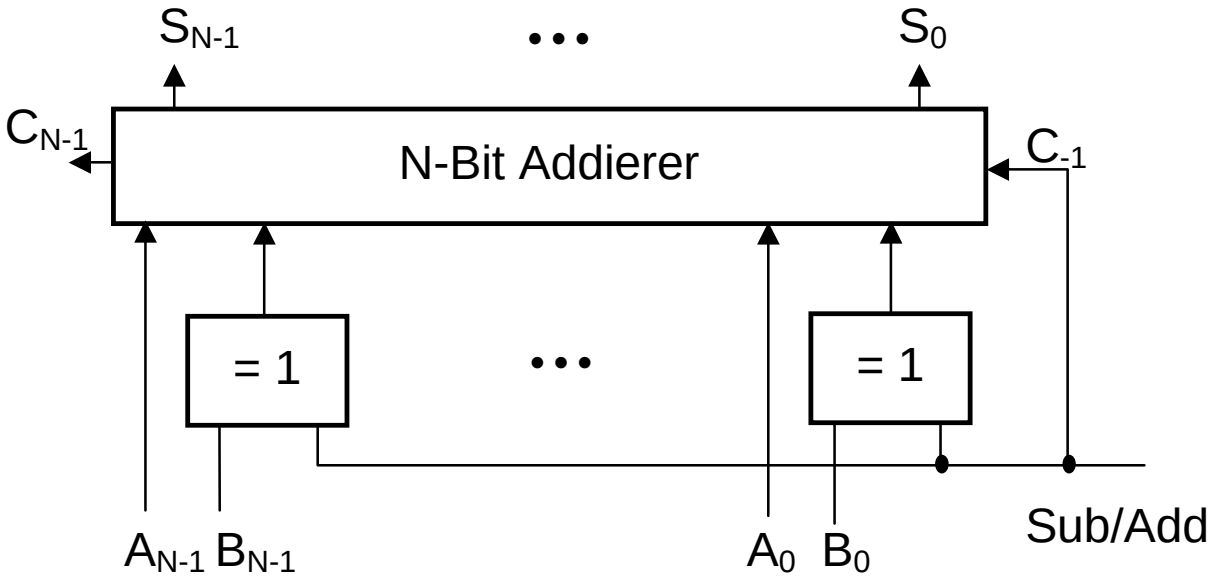
- Subtraktion durch Addition des Komplements (benötigt mehr Operationen, aber am häufigsten weil meist in Kombination mit Addierer)

Negative Zahlen in Komplementdarstellung:

Einerkomplement: $C_{-1} = C_{N-1}$ (Einserrücklauf)

Zweierkomplement: $C_{-1} = 1$, C_{N-1} ignorieren.

Addierer / Subtrahierer (Zweierkomplement)
(Addition bei Sub/Add = 0, Subtraktion bei Sub/Add = 1)



Durch Sub/Add = 1, also $C_{-1} = 1$, wird aus dem Einerkomplement von B ein Zweierkomplement.

Bit C_{N-1} ist das Vorzeichen des Ergebnisses.

Negative Differenz S = A - B richtig im Zweierkomplement.

4.5.4 Multiplikation

Für kleinere Wortbreiten kann die Multiplikation direkt in einem Schaltnetz vorgenommen werden.

Prinzip: Stellengewichtete Addition abhängig von Multiplikator-Bits

Hier: Nur positive Zahlen; Verallgemeinerung auf negative Zahlen möglich.

$M = (m_{N-1}, m_{N-2}, \ldots m_0)_2$; $Q = (q_{N-1}, q_{N-2}, \ldots, q_0)_2$

$$M \cdot Q = \sum_{i=0}^{N-1} q_i \cdot M \cdot 2^i$$

$M \cdot 2^i$ Schieben um i Stellen nach links

$q_i \cdot M$ entspricht $q_i \wedge (m_{N-1}, m_{N-2}, \ldots m_0)_2$

Vergleiche Multiplikation per Hand:

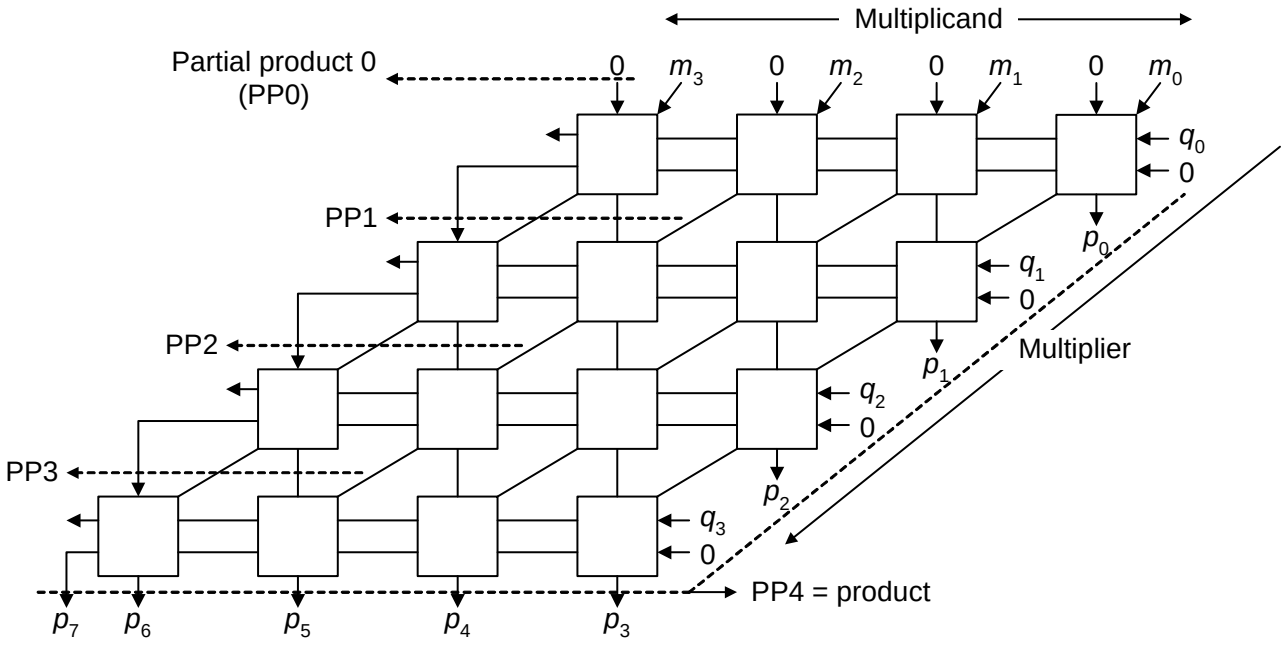
Beispiel:

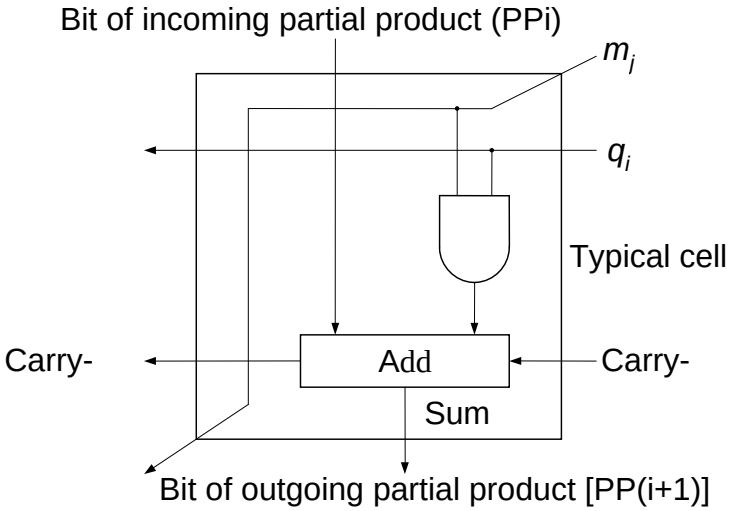
```
 0 1 1 1  x  1 1 0 1
-------------------
           0 1 1 1
         0 0 0 0
       0 1 1 1
     0 1 1 1
-------------------
 1 0 1 1 0 1 1
```

Realisierung der N Additionen mit N Addierern aus je N Volladdierern (Ripple-Carry Adder), die verschoben hintereinander geschaltet sind. Die jeweiligen Partial-Produkte PP_i werden stufenweise geeignet gesteuert durch die Multiplikator-Bits q_i über UND-Gatter aufaddiert.

Aufbau: Strukturell einfache, VLSI-freundliche Realisierung mittels einheitlicher, einfacher Zellen aus Volladdierer und UND-Gatter.

Beispiel: Asynchroner vierstelliger Parallel-Multiplizierer





Aufwand:
N^2 Zellen aus Volladdierer plus UND-Gatter (d.h. $O(N^2)$)

Vergleiche Shift/Add-Mul.:
O(N log N) mit Carry-Lookahead, O(N) mit Ripple-Carry

Zeitbedarf:
τ für UND-Gatter sowie 2τ für Volladdierer, d. h.

$T_{MUL} = (2N - 2)\ 2\tau + 3\tau$
$= (4N - 1)\ \tau = O(N)$

Vergleiche Shift/Add-Mul.:
O(N log N) mit Carry- Lookahead, $O(N^2)$ mit Ripple-Carry

Die Schaltnetz-Realisierung ergibt schnelle asynchrone Multiplizierer (Combinational Array Multiplier).

Es ist keine Ablaufsteuerung/Kontrolleinheit erforderlich!

Der Hardwareaufwand ist bei heutiger Integrationsdichte von integrierten Schaltungen für kleine bis mittlere Wortbreiten akzeptabel.

Weitere schnelle Algorithmen für Multiplizierer bekannt: z. B. Carry-Save Multiplikation, Algorithmen mit Zusammenfassung von Bitgruppen wie der Booth-Algorithmus, der gleich mehrere benachbarte Bits des Multiplikators gleichzeitig betrachtet.

Weil der Hardwareaufwand aber *quadratisch* mit der Wortbreite wächst, wird die Multiplikation für größere Wortbreiten nicht mehr in einem Schritt in einem Schaltnetz berechnet, sondern auf aufeinander folgende stellengewichtete Additionen in der selben Hardwareeinheit mit Zwischenspeichern der Zwischenergebnisse abgebildet.

Für diese mehrschrittige („sequentielle“) Verarbeitung reichen in Hardware nur ein Addierer und ein Zwischenspeicher sowie ein **Steuerwerk** (s. unten), das die Abfolge der Verarbeitungsschritte und das Speichern von Zwischenergebnissen kontrolliert.

Bei einer solchen mehrschrittigen Realisierung („*Rechenwerke*“) sind auch Multiplikationen von vorzeichenbehafteten und von Fest- sowie Gleitpunktzahlen leicht bei vertretbarem Hardwareaufwand zu realisieren (s. unten).