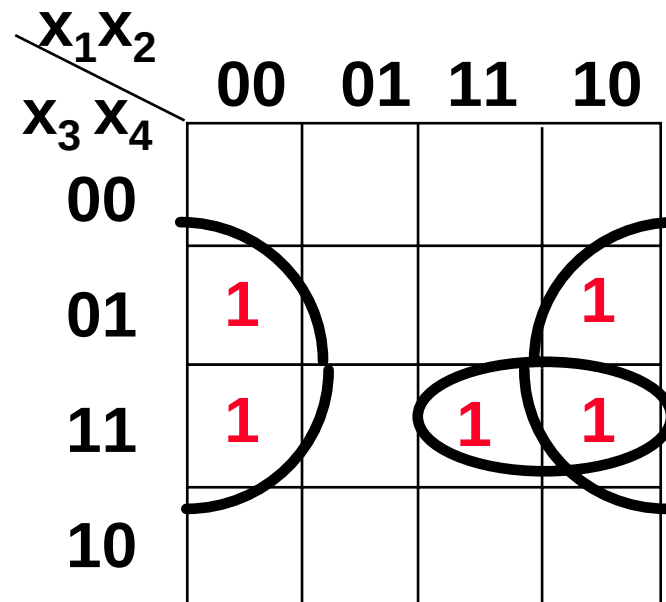


4. Schaltnetze und ihre Optimierung



Synthese von Schaltkreisen
Minimierung
Verfahren von Karnaugh
Unvollständig definierte Schaltfunktionen
Quine-McCluskey Algorithmus
Das Überdeckungsproblem
Fehlerdiagnose
PLAs -Programmable Logic Arrays

Synthese von Schaltkreisen

Zu jeder Schaltfunktion gibt es unendlich viele Boolesche Terme und damit **unendlich viele Schaltkreise**, die sie realisieren.

Normalformen führen oft zu komplexen Schaltkreisen (man denke z.B. an die Addition von zwei 16Bit Zahlen).

=> **“special purpose” Verfahren, Modulkonzept**

=> **Minimierung von Schaltkreisen:**

Bestimmung des einfachsten und billigsten Booleschen Terms bzgl. eines Kostenmaßes

Bewertung von Schaltnetzen

Definition: Sei $B(X)$ die Menge der Booleschen Terme über einer Variablenmenge $X = \{x_1, x_2, \dots, x_n\}$.

Eine **Kostenfunktion** $K : B(X) \rightarrow \mathbb{N}$ sei definiert durch:

- $K(x_i) = K(0) = K(1) = 0$
(Leitungen kostenlos)
- $K(t_1 + t_2) = K(t_1 * t_2) = K(t_1) + K(t_2) + 1$
- $K(t') = K(t)$
(Negation kostenlos)

Definition eines einfachen Kostenmaßes:

“Kosten Boolescher Terme =
Zahl der AND- bzw.
OR-Gatter des
entsprechenden
Schaltkreises”

Negationen, Leitungslängen,
Ein- und Ausgänge usw.
bleiben unberücksichtigt.

Beispiel:

$$\begin{aligned} & K(x_1(x_2+x_3)x_4 + x_3) \\ &= K(x_1(x_2+x_3)x_4) + K(x_3) + 1 \\ &= \dots = 4 \end{aligned}$$

Eingeschränktes Minimierungsproblem

Bestimmung der billigsten **zweistufigen Realisierung** einer Schaltfunktion, d.h. eine Realisierung, bei der zunächst nur AND-Gatter und anschließend nur OR-Gatter oder umgekehrt durchlaufen werden. Die **disjunktive und konjunktive Normalform** gehören zu den zweistufigen Realisierungen. Im folgenden betrachten wir nur Vereinfachungen der disjunktiven Normalform.

Definition: $P(X)$ ist die Menge der **polynomiellen** Booleschen Terme über der Variablenmenge $X = \{x_1, x_2, \dots, x_n\}$:

$$P(X) = \{ p \mid p = 0 \vee p=1 \vee$$

$$p = a_1 + \dots + a_m \text{ mit } a_i = x_{i_1}^{\varepsilon_{i1}} * \dots * x_{i_k}^{\varepsilon_{ik}} \}$$

a_i heißt **Monom**.

Ein billigstes Polynom zur Realisierung von f heißt **Minimalpolynom von f** .

Beispielpolynome

- Die disjunktive Normalform einer Schaltfunktion ist ein Polynom, bei dem jedes Monom ein Minterm ist.
- Das Polynom $p = x_1'x_3 + x_2x_3'$ definiert die Schaltfunktion

x1	x2	x3	$x_1'x_3 + x_2x_3'$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

disjunktive Normalform:

$$x_1'x_2'x_3 + x_1'x_2x_3' + x_1x_2'x_3 + x_1x_2x_3'$$

Vereinfachung:

$$\begin{aligned} &= x_1'(x_2' + x_2)x_3 + (x_1' + x_1)x_2x_3' \\ &= x_1'1x_3 + 1x_2x_3' \\ &= x_1'x_3 + x_2x_3' \end{aligned}$$

Resolutionsregel

Kommen in einem Polynom zwei Monome vor, die sich in **genau einer** komplementären Variablen unterscheiden, so können diese beiden Monome durch den ihnen gemeinsamen Teil ersetzt werden.

$$\begin{array}{ccc} x_1^{\epsilon_1} * \dots * \boxed{x_i} * \dots * x_n^{\epsilon_n} & & x_1^{\epsilon_1} * \dots * \boxed{x_i'} * \dots * x_n^{\epsilon_n} \\ & \searrow \quad \swarrow & \\ & x_i + x_i' = 1 & \\ & \swarrow \quad \searrow & \\ x_1^{\epsilon_1} * \dots * x_{i-1}^{\epsilon_{i-1}} * x_{i+1}^{\epsilon_{i+1}} * \dots * x_n^{\epsilon_n} \end{array}$$

Beispielresolutionen

$$f(x_1, x_2, x_3, x_4)$$

$$= x_1 \overset{\text{green}}{x_2'} \overset{\text{red}}{x_3} x_4 + \overset{\text{brown}}{x_1} x_2' \overset{\text{red}}{x_3'} x_4 + x_1 \overset{\text{green}}{x_2} \overset{\text{red}}{x_3} x_4 + \overset{\text{brown}}{x_1'} x_2' \overset{\text{blue}}{x_3'} x_4 + x_1' x_2' \overset{\text{blue}}{x_3} x_4$$

$$= \overset{\text{red}}{x_1} x_2' x_4 + x_1 x_3 x_4 + x_2' x_3' x_4 + \overset{\text{red}}{x_1'} x_2' x_4$$

$$= x_2' x_4 + x_1 x_3 x_4 + x_2' x_3' x_4$$

Eine mehrfache Verwendung der Resolutionsregel beruht auf dem Gesetz $x + x = x$.

Das Verfahren von Karnaugh

Mit Hilfe einer graphischen Darstellung der Funktionstafel von Schaltfunktionen mit 3 oder 4 Argumenten erhält man eine Übersicht über mögliche Resolutionen.

Ein **Karnaugh-Diagramm** einer Schaltfunktion $f : \underline{2}^n \rightarrow \underline{2}$ mit $3 \leq n \leq 4$ ist eine graphische Darstellung der Funktionstafel durch eine 0-1-Matrix der Größe 2×4 für $n=3$ und 4×4 für $n=4$ mit folgender Beschriftung:

$x_1 x_2$		00	01	11	10
x_3	0				
	1				

$x_1 x_2$		00	01	11	10
$x_3 x_4$	00				
	01				
	11				
	10				

Das Verfahren von Karnaugh 2

Beispiel:

$$f(x_1, x_2, x_3, x_4)$$

$$= x_1 x_2' x_3 x_4 + x_1 x_2' x_3' x_4 + x_1 x_2 x_3 x_4 + x_1' x_2' x_3' x_4 + x_1' x_2' x_3 x_4$$

Karnaugh-Diagramm zu f:

$x_1 x_2$ $x_3 x_4$	00	01	11	10
00				
01	1			1
11	1		1	1
10				

Bei der Zeilen- und Spaltenbeschriftung wird der zwei-stellige **Gray-Code** verwendet, d.h. zwei zyklisch benachbarte Zeilen oder Spalten unterscheiden sich in genau einer Komponente.

Je zwei benachbarte Einsen liefern eine Resolution.

Das Verfahren von Karnaugh 3

Man überdeckt alle im Karnaugh-Diagramm auftretenden Einsen durch möglichst große **Blöcke der Form $2^r \times 2^s$** und bildet die diesen Blöcken entsprechenden Monome. Dies liefert eine vereinfachte Darstellung der Funktion, denn die Blöcke entsprechen jeweils $2^r \times 2^s$ Mintermen, die durch sukzessive Resolutions vereinfacht werden können.

Karnaugh-Diagramm zu f:

$x_1 x_2$ $x_3 x_4$		00	01	11	10
00					
01	1				1
11	1			1	1
10					

Viererblock: $x_2' x_4$

Zweierblock: $x_1 x_3 x_4$

=> vereinfachte
Darstellung:

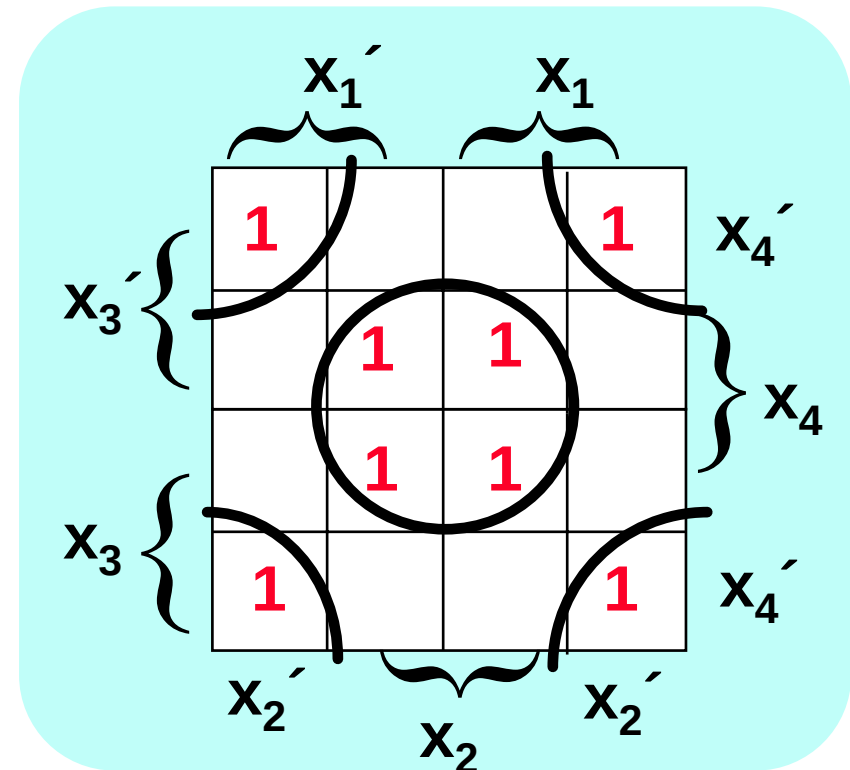
$$f(x_1, x_2, x_3, x_4) = x_2' x_4 + x_1 x_3 x_4$$

Alternative Beschriftung

Oft verwendet man zur Zeilen- bzw. Spaltenbeschriftung statt der Variablenwerte die Variablen selbst und zwar bezeichnet man die Spalten bzw. Zeilen mit x , für die die Variable x den Wert 1 annimmt (und die anderen mit x').

Beispiel:

$$\begin{aligned}
 & f(x_1, x_2, x_3, x_4) \\
 = & \quad x_1' x_2' x_3' x_4' + x_1 x_2' x_3' x_4' \\
 & + x_1' x_2 x_3' x_4' + x_1 x_2 x_3' x_4' \\
 & + x_1' x_2 x_3 x_4' + x_1 x_2 x_3 x_4' \\
 & + x_1' x_2' x_3 x_4' + x_1 x_2' x_3 x_4' \\
 = & \quad x_2 x_4 + x_2' x_4'
 \end{aligned}$$



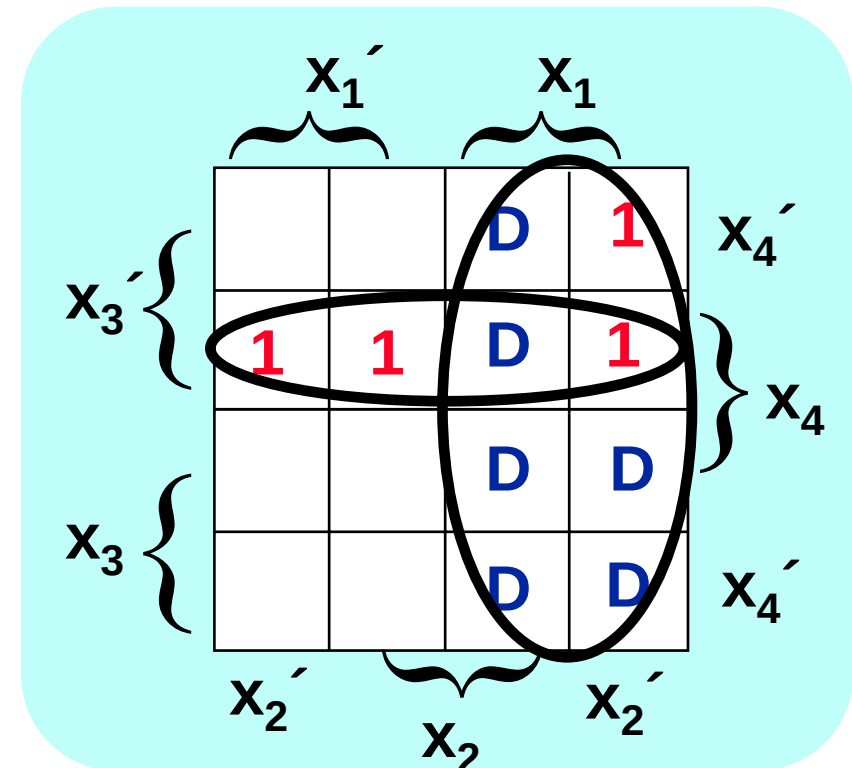
Don't Care-Einträge

Schaltfunktionen sind nicht immer für alle möglichen Eingabetupel definiert. Die Argumenttupel, für die durch die Problemstellung kein Funktionswert festgelegt wird, bezeichnet man als **“Don't Care”**-Fälle, da man beim Entwurf eines Schaltkreises für diese Argumente willkürlich Funktionswerte festsetzen kann.

Beispiel: f sei für $0 \leq x \leq 9$ definiert durch

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

$$\Rightarrow f(x_1, x_2, x_3, x_4) = x_1 + x_3' x_4$$



Implikanten und Primimplikanten

Ein Monom a heit **Implikant von f** ,

falls $a(x_1, \dots, x_n) \leq f(x_1, \dots, x_n)$ fr alle $x_1, \dots, x_n \in \underline{2}$

Ein Implikant a von f heit **Primimplikant (von f)**,

falls kein Teilmonom von a Implikant von f ist.

Beispiel: $f(x_1, x_2, x_3, x_4)$

$$= x_1 x_2' x_3 x_4 + x_1 x_2' x_3' x_4 + x_1 x_2 x_3 x_4 + x_1' x_2' x_3' x_4 + x_1' x_2' x_3 x_4$$

Implikanten von f :

$x_1 x_2' x_3 x_4,$
 $x_1' x_2' x_3' x_4,$
 $x_1 x_2 x_4,$
 $x_2' x_3' x_4,$

$x_1 x_2' x_3' x_4,$
 $x_1' x_2 x_3 x_4,$
 $x_1' x_2' x_4,$
 $x_2' x_3 x_4,$

$x_1 x_2 x_3 x_4,$

**$x_1 x_3 x_4,$ Primimpli-
 $x_2' x_4$ kanten**

Bestimmung von Minimalpolynomen

Karnaugh:

- rechteckige Blöcke von Einsen $\implies$ Implikanten
- maximale derartige Blöcke $\implies$ Primimplikanten.

Das eingeschränkte Minimierungsproblem wird somit für $n = 3$ und 4 auf die Bestimmung von Primimplikanten zurückgeführt. Dass dies Kostenminimalität garantiert, zeigt der folgende Satz:

Satz: Sei $f : \underline{2}^n \rightarrow \underline{2}$ eine Schaltfunktion, nicht konstant 0.

Ist $M_1 + M_2 + \dots + M_k$ Minimalpolynom von f ,
so sind die M_i ($1 \leq i \leq k$) Primimplikanten von f .

Quine-McCluskey-Algorithmus

Eingabe: Schaltfunktion $f : \underline{2}^n \rightarrow \underline{2}$ mit n beliebig

Ausgabe: Minimalpolynom für f

Methode:

I.) Bestimmung aller Primimplikanten von f

» wiederholte Anwendung der Resolutionsregel

II.) kostenminimale Auswahl von Primimplikanten

» Überdeckung der Minterme durch Primimplikanten



schwieriges
Problem

Quine-McCluskey - Phase I

I.1) Teile die Minterme der disjunktiven Normalform von f anhand der Anzahl der vorkommenden Negationszeichen in Gruppen ein.

Beispiel:

$$\begin{aligned}
 f(x_1, x_2, x_3, x_4) &= x_1 x_2 x_3 x_4' \\
 &+ x_1 x_2 x_3' x_4' \\
 &+ x_1 x_2 x_3' x_4 \\
 &+ x_1 x_2' x_3' x_4' \\
 &+ x_1 x_2' x_3' x_4 \\
 &+ x_1 x_2' x_3 x_4' \\
 &+ x_1 x_2' x_3 x_4
 \end{aligned}$$

Gruppeneinteilung:

Gruppe	Minterme	einschlägiger Index	
		dual	dezimal
1	$x_1 x_2' x_3 x_4$	1011	11
	$x_1 x_2 x_3' x_4'$	1101	13
	$x_1 x_2 x_3 x_4'$	1110	14
2	$x_1' x_2 x_3 x_4'$	0110	6
	$x_1 x_2 x_3' x_4'$	1100	12
3	$x_1' x_2 x_3' x_4'$	0100	4
4	$x_1' x_2' x_3' x_4'$	0000	0

Quine-McCluskey - Phase I (Forts.)

I.2) Betrachte alle Mintermpaare aus benachbarten Gruppen und versuche die Resolutionsregel anzuwenden.

Bilde eine neue Tabelle, in der alle verkürzten Implikanten eingetragen werden. Ferner werden Monome, auf die die Resolutionsregel nicht mehr anwendbar ist, übernommen.

Beispiel (Forts.):

Gruppe	Implikanten	einschlägige Indices dual	dezimal
1	$x_1 x_2' x_3 x_4$	1011	11
	$x_1 x_2 x_3'$	110*	13, 12
	$x_2 x_3 x_4'$	*110	14, 6
	$x_1 x_2 x_4'$	11*0	14, 12
2	$x_1' x_2 x_4'$	01*0	4, 6
	$x_2 x_3' x_4'$	*100	4, 12
3	$x_1' x_3' x_4'$	0*00	0, 4

Quine-McCluskey - Phase I (Forts. 2)

I.3) Iteriere Schritt I.2, bis sich die Tabelle nicht mehr ändert.

=> Die Tabelle enthält dann genau die Primimplikanten.

Beispiel (Forts.):

Gruppe	Implikanten	einschlägige Indices	
		dual	dezimal
1	$x_1 x_2' x_3 x_4$	1011	11
	$x_1 x_2 x_3'$	110*	13, 12
	$x_2 x_4'$	*1*0	4,6,12,14
3	$x_1' x_3' x_4'$	0*00	0, 4

Quine-McCluskey - Phase II

II.1) Man erstelle eine Matrix (a_{ij}) mit

» Primimplikanten p_i als Zeilen und

» Mintermen m_j der DNF als Spalten

wobei $a_{ij} := \begin{cases} 1 & \text{falls } m_j \leq p_i \\ 0 & \text{sonst} \end{cases}$

Beispiel (Forts.):

Primimplikant \ Minterm							
	0	4	6	11	12	13	14
$x_1 x_2' x_3 x_4$	0	0	0	1	0	0	0
$x_1 x_2 x_3'$	0	0	0	0	1	1	0
$x_2 x_4'$	0	1	1	0	1	0	1
$x_1' x_3' x_4'$	1	1	0	0	0	0	0

Quine-McCluskey - Phase II (Forts.)

II.2) Finde in dieser Matrix eine Auswahl von Zeilen, d.h. Primimplikanten, so dass

- (i) die von diesen Zeilen gebildete Teilmatrix **in jeder Spalte mindestens eine Eins** enthält und
- (ii) die **Gesamtkosten** dieser Primimplikanten **minimal** sind unter allen möglichen Auswahlen mit (i)

Beispiel (Forts.):

Minterm							
	0	4	6	11	12	13	14
Primimplikant							
$x_1 x_2' x_3 x_4$	0	0	0	1	0	0	0
$x_1 x_2 x_3'$	0	0	0	0	1	1	0
$x_2 x_4'$	0	1	1	0	1	0	1
$x_1' x_3' x_4'$	1	1	0	0	0	0	0

Vereinfachungsregeln

1 Ein Primimplikant heißt **wesentlich**, falls es einen Minterm gibt, der **nur** von diesem Primimplikanten überdeckt wird. In der entsprechenden Matrixspalte findet sich nur eine 1.

=> Ein wesentlicher Primimplikant muß zum Minimalpolynom gehören.

2 Gilt für Mintermspalten s_i und s_j **komponentenweise $s_i \leq s_j$** , so **streiche s_j** , denn jede Überdeckung von s_i überdeckt auch s_j .

	...	m_i	...	m_j
p_1		1		1
p_2		0		0
p_3		1		1
p_4		0		1

	m1	m2	m3	...
...				
pr	0	1	1	0
...				
ps	0	1	1	1
...				

3 Ist für Primimplikantenzeilen $z_r \leq z_s$ und $K(z_r) \geq K(z_s)$, so **streiche z_r** , denn z_s ist billiger und leistungsfähiger.

Überdeckungsmatrix

Eine 0-1-Matrix $A = (a_{ij})_{1 \leq i \leq n, 1 \leq j \leq m}$ heißt eine **Überdeckungsmatrix**, falls

- (a) alle Spalten voneinander verschieden sind
- (b) die Nullspalte fehlt.

Die Spalten einer solchen Matrix heißen **Objekte**, die Zeilen (Überdeckungs-) **Mengen**. Eine Menge i **überdeckt** das Objekt j : $\Leftrightarrow a_{ij} = 1$.

Bsp (Quine/McCluskey): Objekte = einschlägige Minterme
Mengen = Primimplikanten

Minterm \ Primimplikant	0	4	6	11	12	13	14
$x_1 x_2' x_3 x_4$	0	0	0	1	0	0	0
$x_1 x_2 x_3'$	0	0	0	0	1	1	0
$x_2 x_4'$	0	1	1	0	1	0	1
$x_1' x_3' x_4'$	1	1	0	0	0	0	0

Das Überdeckungsproblem

spezielles Überdeckungsproblem:

Suche möglichst wenige Mengen, die alle Objekte überdecken.

allgemeiner: Die Mengen i haben Kosten $K(i)$ (auch Gewichte genannt).

allgemeines Überdeckungsproblem:

Suche überdeckendes Mengensystem M mit möglichst geringen Gesamtkosten $\sum_{\{i \in M\}} K(i)$

Bsp.

	1	2	3	4
1	0	1	0	0
2	1	1	0	1
3	0	1	1	1
4	0	1	0	1
5	1	0	0	1

$\forall i . K(i) = 1$

Beobachtung:

$z_i \leq z_j$ (elementweises $\leq$) bedeutet:
 z_j überdeckt alles, was z_i überdeckt

$s_i \leq s_j$ (elementweises $\leq$) bedeutet:
 s_j wird immer überdeckt,
wenn s_i überdeckt wird

Im Beispiel gilt: $z_1 \leq z_2$, $s_1 \leq s_4$

Vereinfachungsregeln

Zeilenregel: $Z_i \leq Z_j \wedge K(i) \geq K(j) \Rightarrow$ streiche Z_i

Die Menge Z_j überdeckt alles,
was Z_i überdeckt und ist billiger.
 Z_i ist also entbehrlich.

Spaltenregel: $S_i < S_j \Rightarrow$ streiche S_j

Das Objekt S_j wird immer überdeckt,
wenn S_i überdeckt wird.
Deshalb braucht S_j nicht betrachtet zu werden.

Die sukzessive Anwendung von Zeilen- und Spaltenregel liefert ein **nichtdeterministisches** Verfahren zur Vereinfachung der Überdeckungsmatrix.

Das Verfahren terminiert auf jeden Fall mit einer nicht weiter zu vereinfachenden Matrix ($\Rightarrow$ **irreduzible** Matrix).

Komplexität des Überdeckungsproblems

Das Überdeckungsproblem ist **NP-vollständig**, d.h., es ist nicht bekannt, ob es einen polynomiellen (d.h. effizienten) Algorithmus zur Lösung dieses Problems gibt. *Wahrscheinlich nicht!*

P: polynomiell lösbare Probleme

NP: nicht-deterministisch polynomiell lösbare Probleme

Es lässt sich für *irgendeine* optimale Lösung des Problems in polynomieller Zeit (nichtdet., d.h. in Entscheidungssituationen von "außen" richtig gesteuert) verifizieren, dass es sich tatsächlich um eine Optimallösung handelt.

NP-vollständige Probleme: Es ist bewiesen, dass es *entweder für alle* Probleme dieser Klasse *oder für keines* von ihnen ein polynomielles (deterministisches) Lösungsverfahren gibt.

?

ungelöstes Problem der Informatik: **P = NP**

klar $P \subseteq NP$

Vermutung $P \neq NP$, aber noch nicht bewiesen.

Fehlerdiagnose in Schaltkreisen

VLSI-Design, Endkontrolle von Chips, Funktionstest nach Fertigung

Beispiel: 16-Bit Addierer $\Rightarrow$ 2^{32} mögl. Eingaben, aber
4 Mrd. Test nicht durchführbar

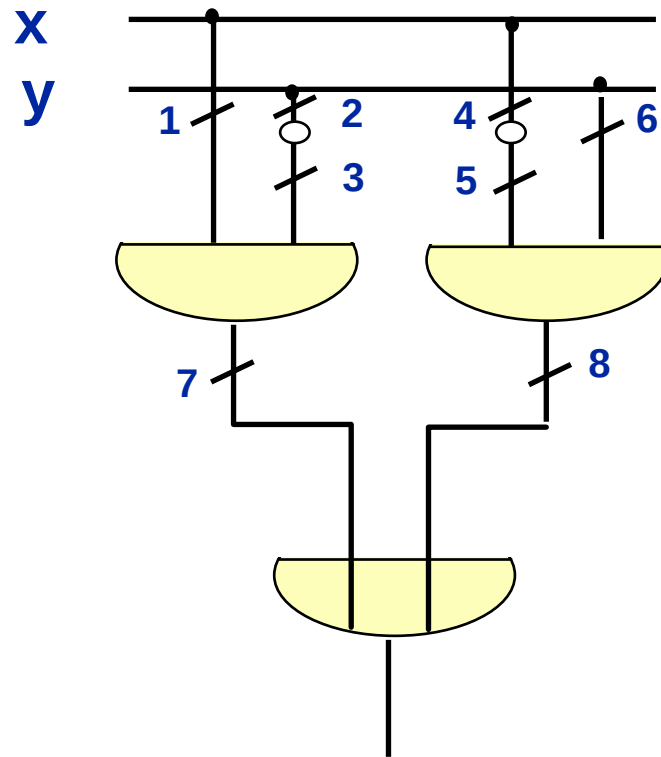
$\Rightarrow$ Wähle Teilmenge von Tests aus,
die alle Fehler einer bestimmten Art aufdeckt

Dazu macht man meist eine bestimmte **Fehlerannahme**, etwa

- (a) es tritt höchstens ein Fehler auf (**"1-Fehler Annahme"**)
- (b) der Defekt ist ein gerissener Verbindungsdraht
(oder z.B. defektes Gatter, Kurzschluß ...)

Beispiel

Halbaddierer (S-Ausgang) $S = xy' + x'y$



8 mögl. Leitungsrisse
(0-Verklemmungen)
3 mögl. Gatterfehler

Fehlerterme:

S_i = Ergebnis bei diesem Fehler
Ein-Fehler-Annahme!

$$S_1 = 0 \cdot y' + x'y = x'y$$

$$S_2 = x \cdot 1 + x'y = x + x'y = x + y$$

$$S_3 = x \cdot 0 + x'y = x'y$$

$$S_4 = xy' + 1 \cdot y = xy' + y = x + y$$

$$S_5 = xy' + 0 \cdot y = xy'$$

$$S_6 = xy' + x' \cdot 0 = xy'$$

$$S_7 = 0 + x'y = x'y$$

$$S_8 = xy' + 0 = xy'$$

Überprüfung dieser Fehlerfälle in der Ausfallmatrix

x	y	S ₁	S ₂	S ₃	S ₄	S ₅	S ₆	S ₇	S ₈
0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	1	0
1	0	0	1	0	1	1	1	0	1
1	1	0	1	0	1	0	0	0	0

Fehlermatrix: Ersetze S_i durch S_i ⊕ S

x	y	S ₁ ⊕ S	S ₂ ⊕ S	S ₅ ⊕ S
0	0	0	0	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Testvektoren: minimale Zeilenauswahl, die alle Einsen überdeckt.

=> 3 Testvektoren 01, 10, 11 sind notwendig, um alle Fehler zu finden.

Optimierung von Schaltkreisen

bisher:

- Minimierung durch Bestimmung von Minimalpolynomen
- “Fehlerdiagnose”

jetzt: **technische Funktionssicherheit**

Annahmen:

- » Jedes Signal, das ein Gatter durchläuft, hat eine Laufzeit.
- » Änderung von Eingabewerten, die logisch gleichzeitig erfolgen, können physikalisch nacheinander stattfinden.
- » Signallaufzeiten können für unterschiedliche Gatter unterschiedlich groß sein.

=> Hazards (Hasards, engl. Gefahr, Risiko):

unerwünschte Änderungen von Ausgangswerten beim Umschalten von Eingangsignalen.

Hazards

- Funktionshazards:
schaltungsunabhängig, Folgerung
aus Annahme 2

Beispiel: $f(x_1, x_2, x_3) = x_1 x_3 + x_2 x_3'$

$x_1 x_2 \backslash x_3$	00	01	11	10
0	0	1	1	0
1	0	0	1	1

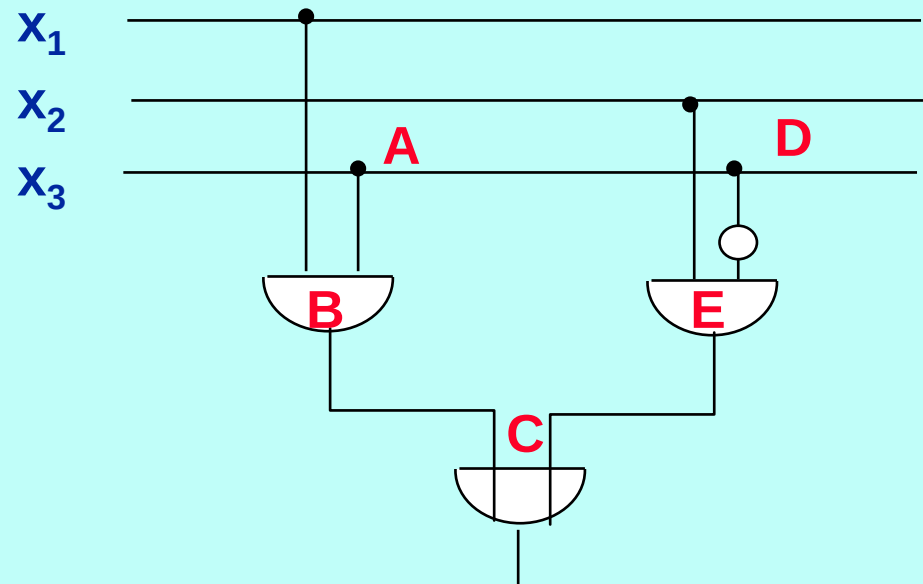
Primimplikanten: $x_1 x_3$, $x_2 x_3'$, $x_1 x_2$

Es gilt auch:

$$f(x_1, x_2, x_3) = x_1 x_3 + x_2 x_3' + x_1 x_2$$

- Schaltungshazards:
schaltungsabhängig,
Folgerung aus Annahme 3

Beispiel: $f(x_1, x_2, x_3) = x_1 x_3 + x_2 x_3'$

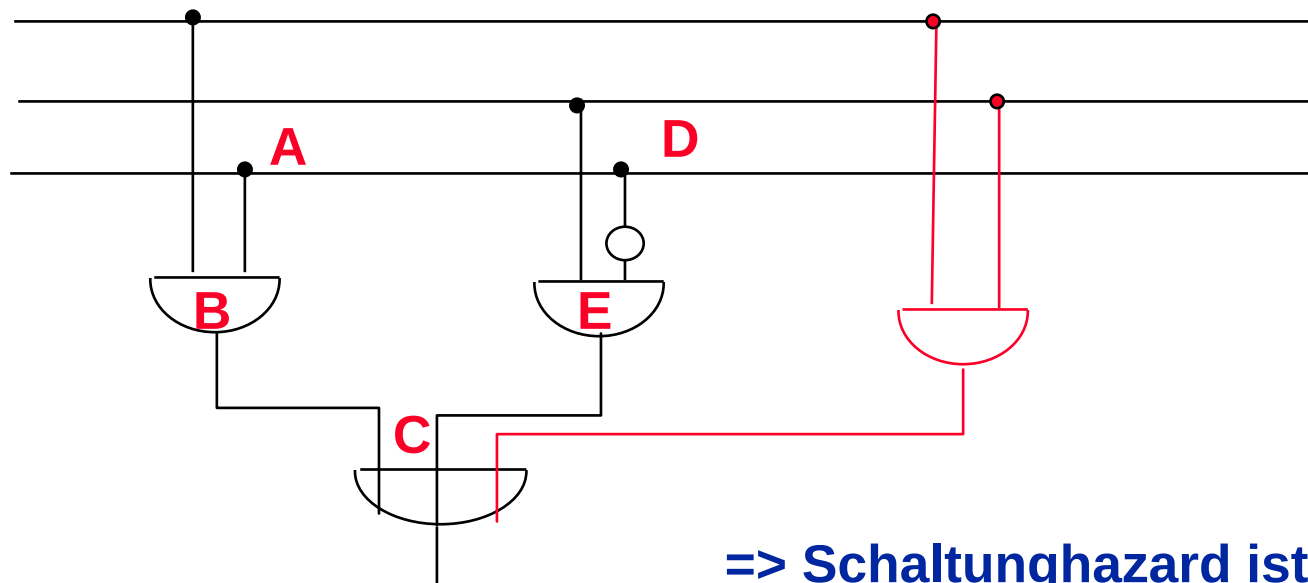


$f(1,1,0) = f(1,1,1) = 1$, aber Ausgang
kurzzeitig auf 0, falls Signalweg
ABC langsamer als DEC

Eliminierung von Schaltungshazards

Schaltungshazards können durch Hinzunahme von zusätzlichen Primimplikanten als “**Gegenschaltung**” eliminiert werden,
im Beispiel:

$$f(x_1, x_2, x_3) = x_1 x_3 + x_2 x_3' = x_1 x_3 + x_2 x_3' + \textcolor{red}{x_1 x_2}$$



=> Schaltungshazard ist beseitigt

Satz (Eichelberger 1969):

Ein zweistufiges Schaltnetz S für eine Schaltfunktion f in Polynomform ist frei von Schaltungshazards, falls die UND-Gatter (Monome) in einer 1-1-Korrespondenz zu den Primimplikanten von f stehen.

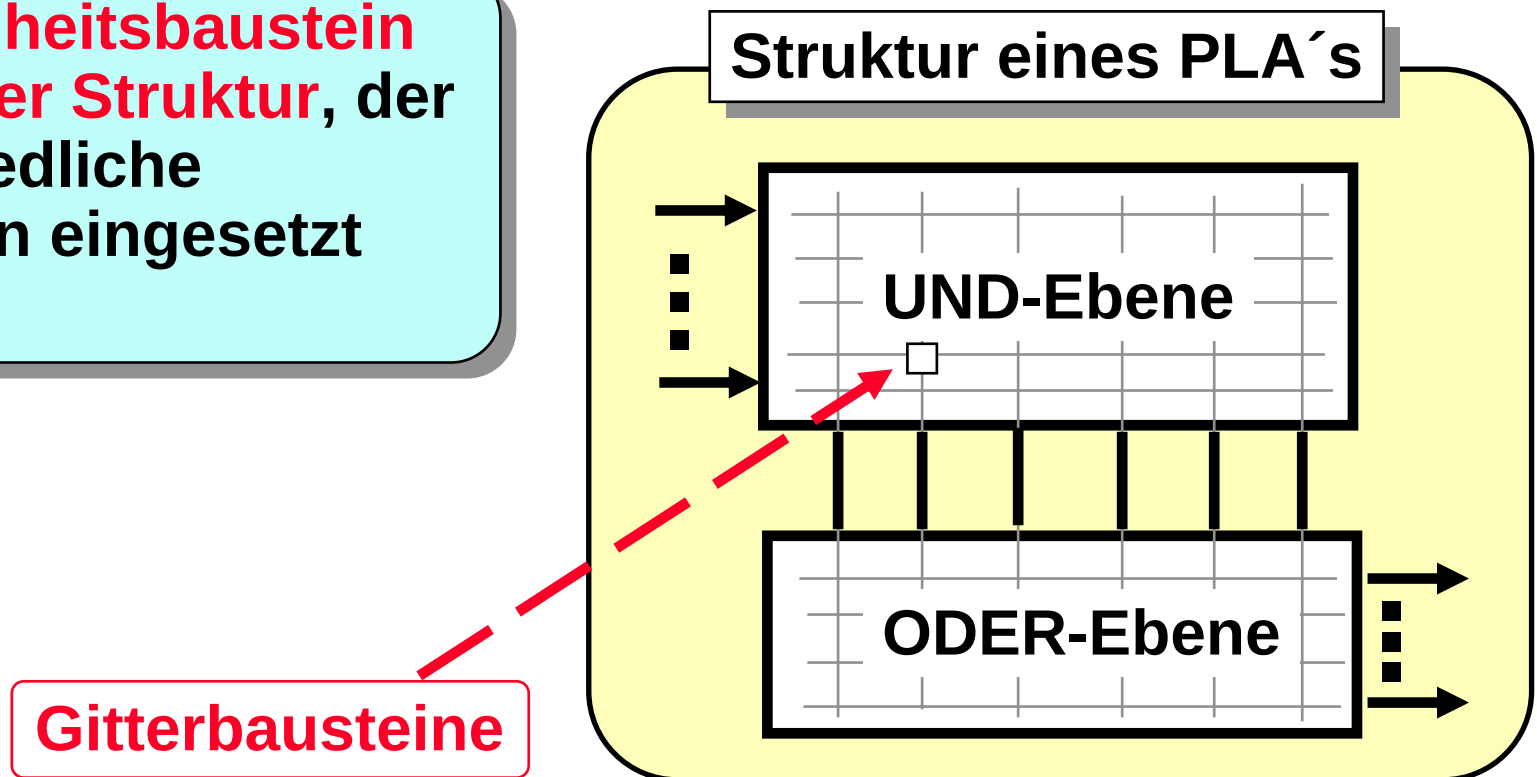
Programmable Logic Arrays (PLA's)

bisher:

Schaltfunktion	-> DNF	-> Minimalpolynom
		-> zweistufige Realisierung

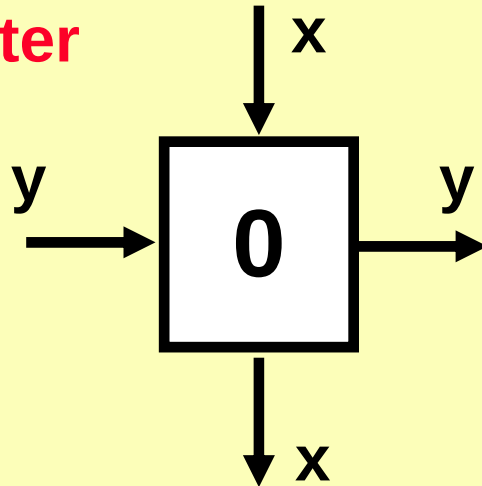
Problem: unterschiedlich formatierte Gatter erschweren automatisierte Herstellung mit Halbleitertechnik

Entwickle Einheitsbaustein mit homogener Struktur, der für unterschiedliche Anwendungen eingesetzt werden kann.

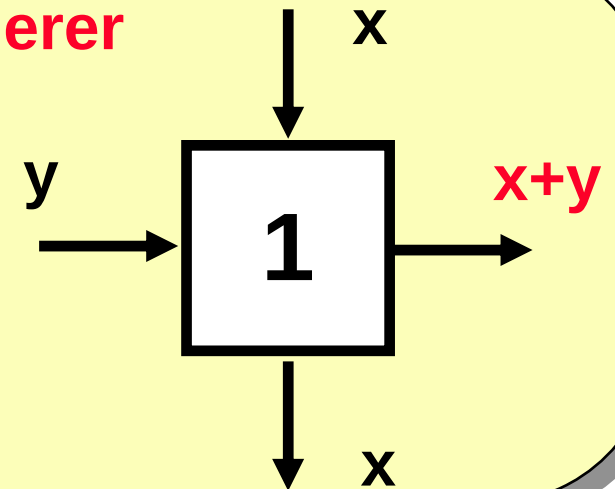


Gitterbausteine

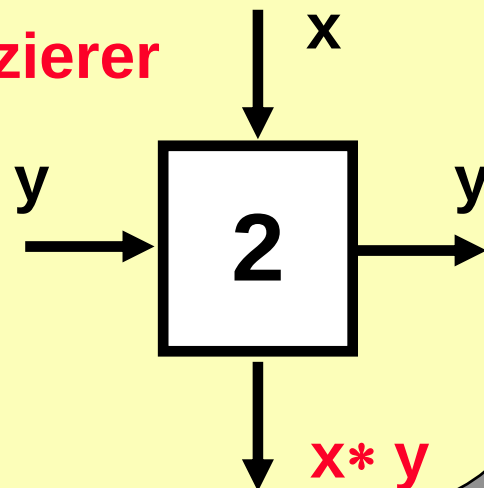
Identer



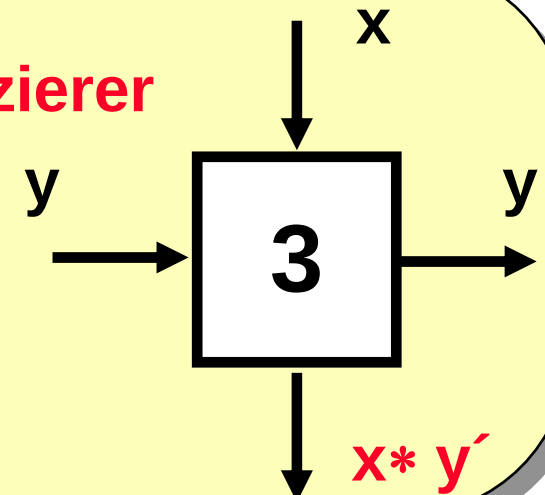
Addierer



Multiplizierer

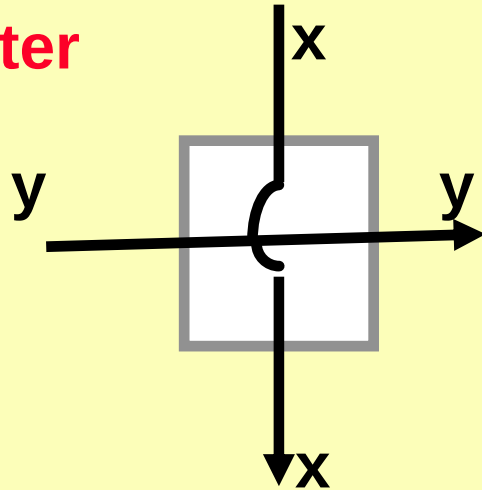


Negat-Multiplizierer

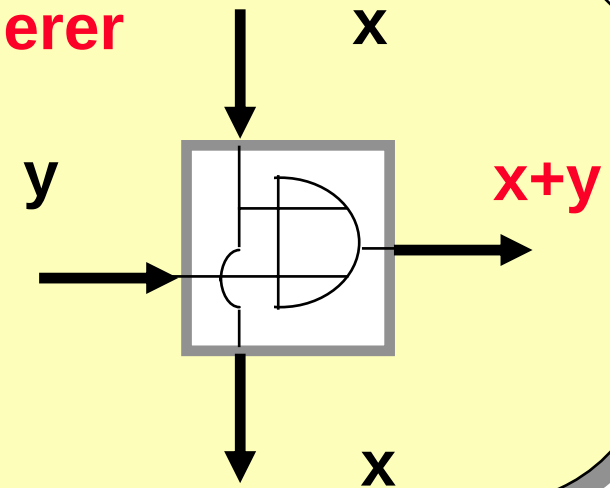


Realisierung der Gitterbausteine

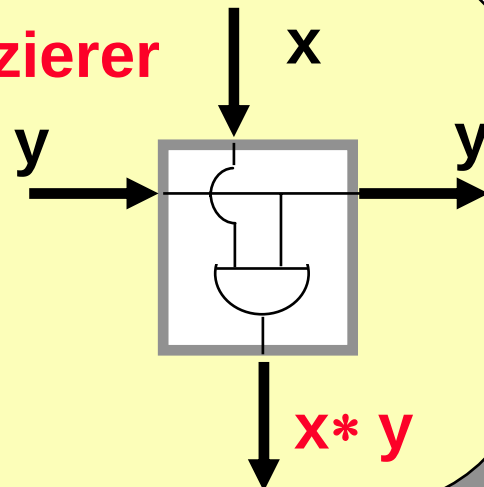
Identer



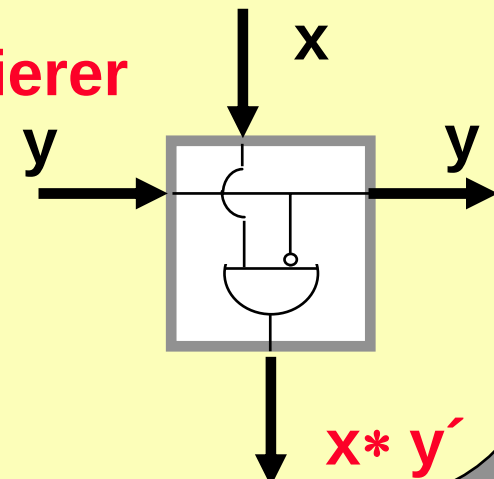
Addierer



Multiplizierer



Negat-Multiplizierer



Realisierung von Schaltfunktionen mit PLA's

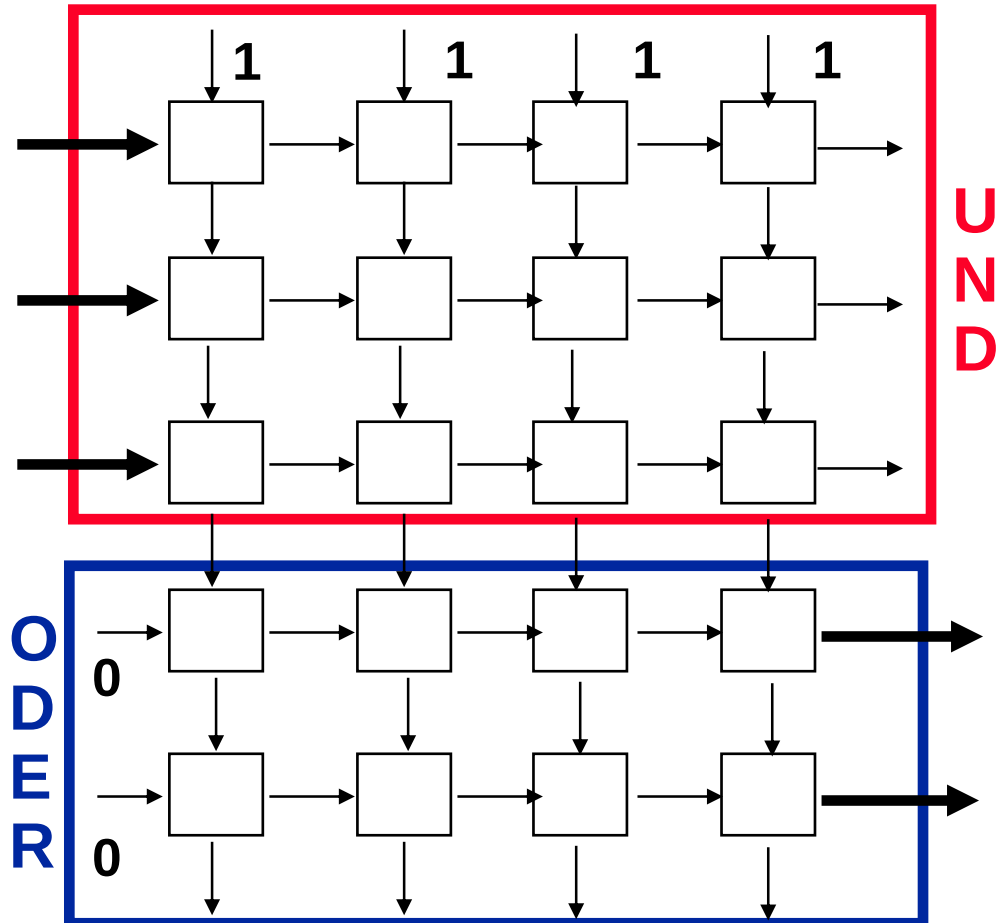
Zur Realisierung einer Schaltfunktion

$$f: \underline{2}^n \rightarrow \underline{2}^m$$

benötigt man ein PLA mit

- n Inputs,
- m Outputs und
- $1 \leq k \leq 2^n$ Spalten.

Beispiel: $n=3, m=2, k=4$



In der **UND-Ebene** werden nur **Identier** und **Multiplizierer** verwendet, in der **ODER-Ebene** nur **Identier** und **Addierer**.

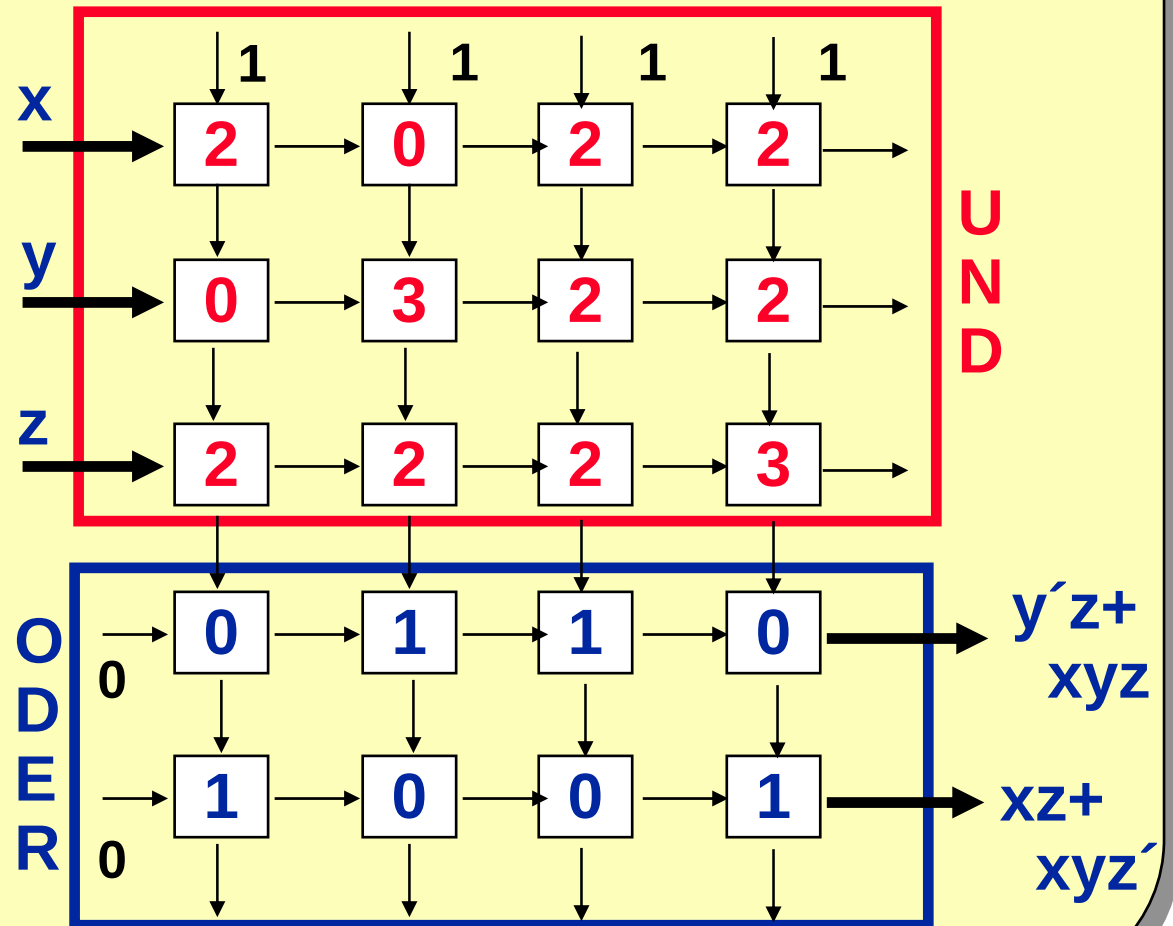
Beispiel

Sei $f: \underline{2}^3 \rightarrow \underline{2}^2$
definiert durch

$$f(x,y,z) = (y'z + xyz, xz + xyz').$$

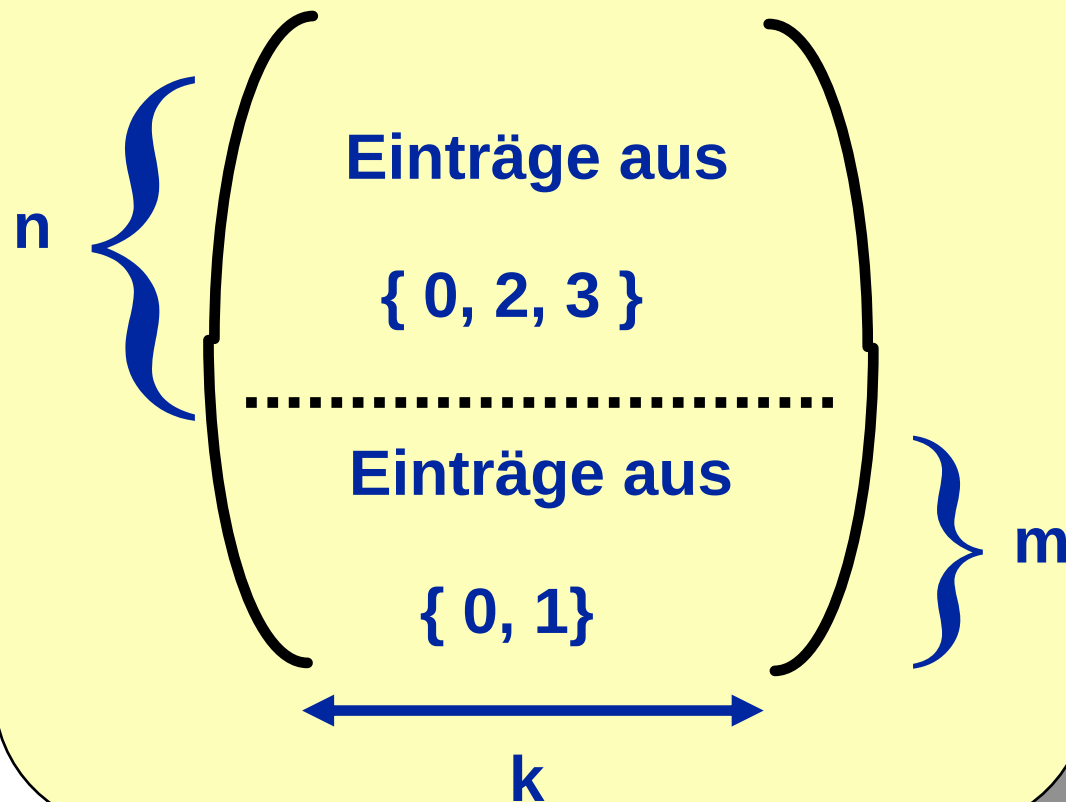
In den Spalten der Und-Ebene werden die Produktterme erzeugt, in den Zeilen der Oder-Ebene die Summen. Eine Funktion in DNF kann somit direkt in einem PLA realisiert werden.

PLA zur Realisierung von f:



Logische Beschreibung von PLA's

Die Beschreibung von PLA's erfolgt meist durch eine $(n+m) \times k$ Matrix, in der nur die Typen der Gitterbausteine vermerkt werden:

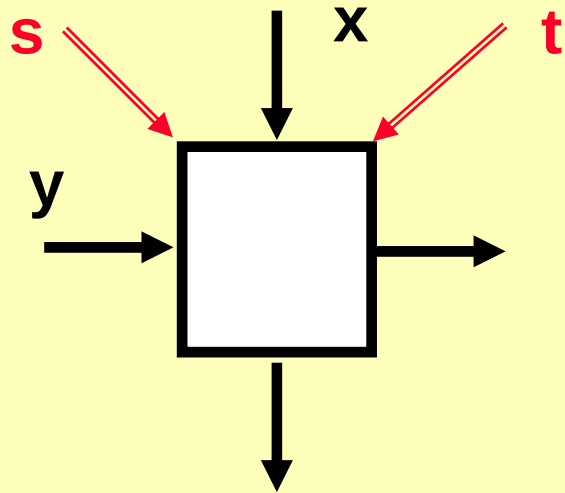


Beispiel (Forts.): Die Matrix

$$\begin{pmatrix} 2 & 0 & 2 & 2 \\ 0 & 3 & 2 & 2 \\ 2 & 2 & 2 & 3 \\ \dots & \dots & \dots & \dots \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

realisiert die obige Beispielfunktion f .

Programmierbare Bausteine



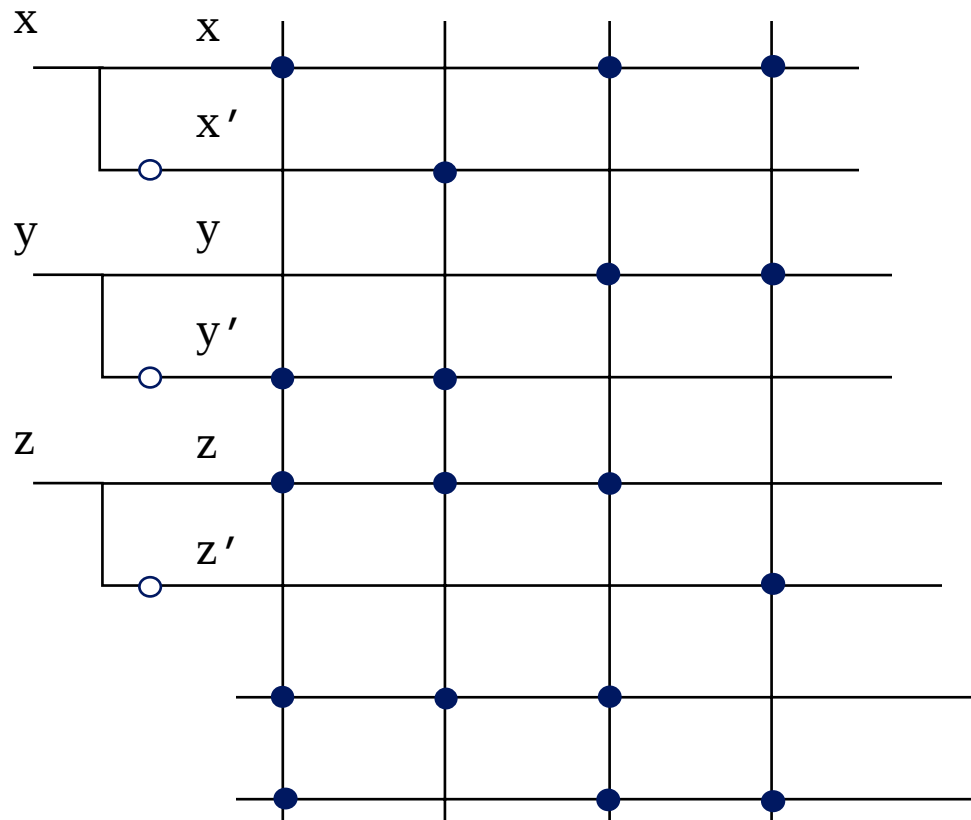
Die **Steuereingänge s und t** bestimmen das Verhalten des Bausteins:

s	t	Typ
0	0	0
0	1	1
1	0	2
1	1	3

PLA-Satz

Durch geeignete Eintragungen in die PLA-Matrix kann jede Schaltfunktion (der gewünschten Dimensionierung) realisiert werden.

PLAs - Variante der Darstellung



- ist **Multiplizierer** in UND-Ebene und **Addierer** in ODER-Ebene

nur noch 1 Steuerleitung
pro Gitterbaustein

$$xy'z + x'y'z + xyz$$

$$xy'z + xyz + xyz'$$

ROMs als spezielle PLAs

ROM - Festspeicher mit

Adressen der Länge n

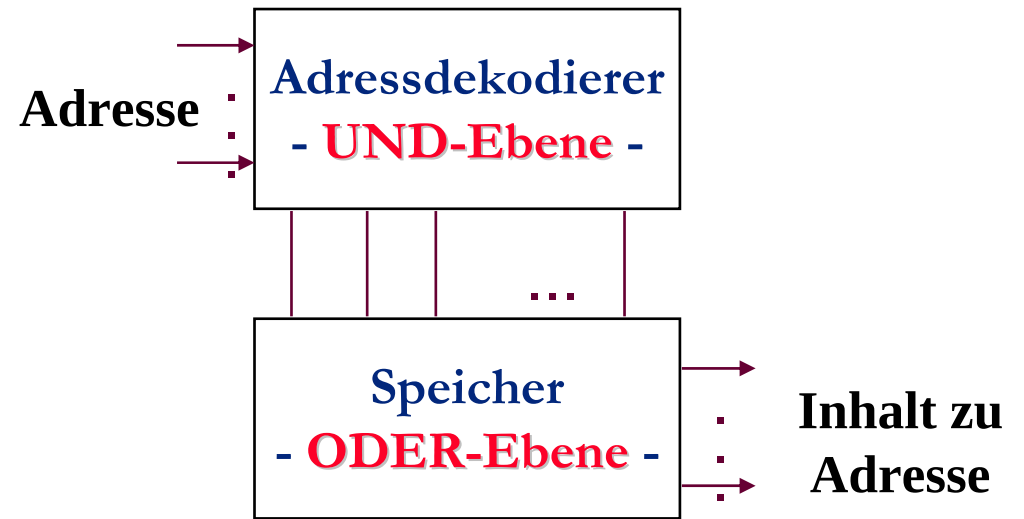
→ 2^n Werte der Länge m

→ $m \times 2^n$ Matrix, deren Spalten den Adressen 0 bis 2^n-1 entsprechen

m Bits

1	1			1
1	0			1
0	0		...	1
1	0			0
0	...			2^n-1

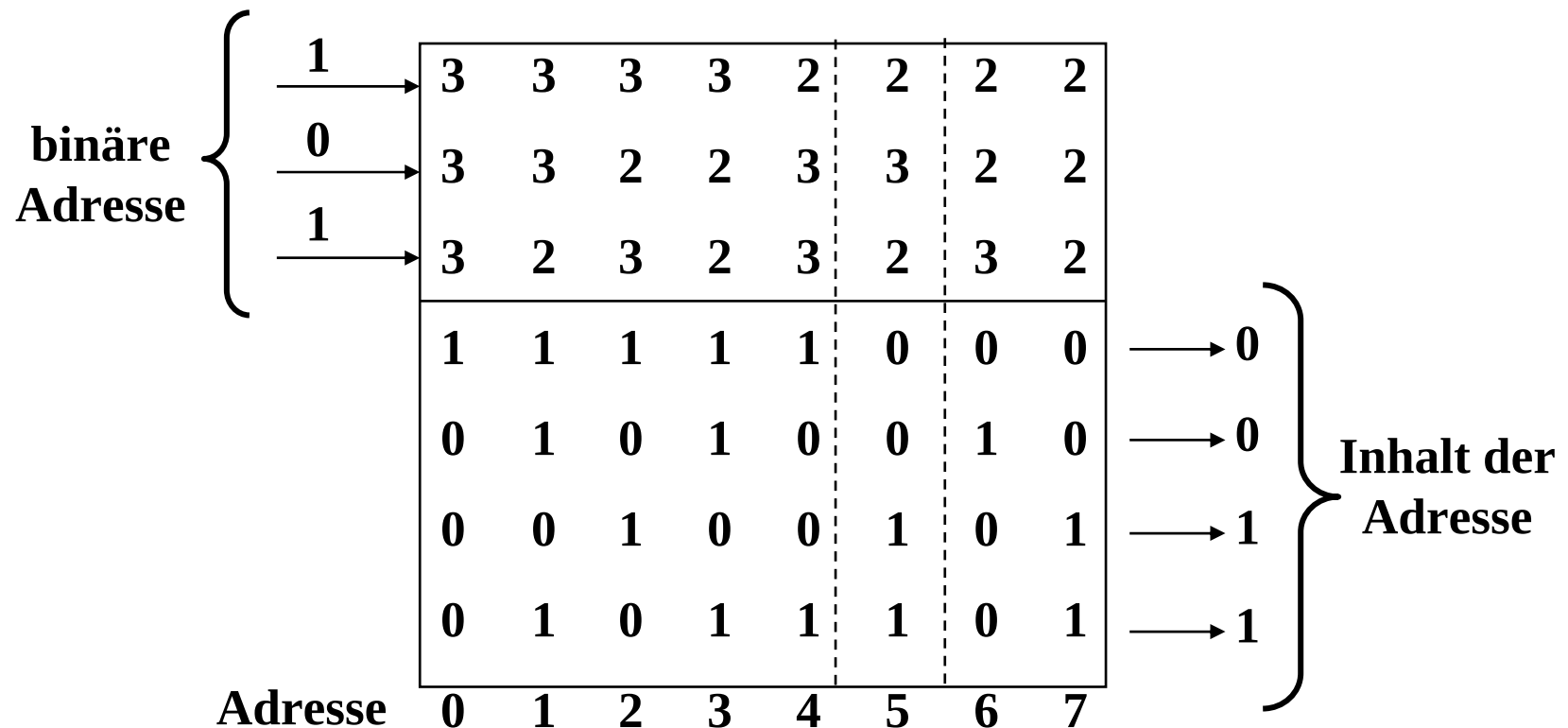
... kann als Oder-Teil eines
PLAs aufgefaßt werden.



Durch Hinzunahme eines **UND-Teils** der Dimension $n \times 2^n$ erhält man ein **PLA** mit n Eingaben, m Ausgaben und 2^n Spalten.

Die UND-Ebene realisiert einen **Adressdekodierer**, wenn in jeder Spalte genau der Minterm erzeugt wird, der diese Spalte dual codiert.

Beispiel: $n = 3, m = 4$



Bei Eingabe einer Adresse i in Dualdarstellung wird **genau in der i -ten Spalte eine 1** erzeugt, so dass der in der **Oder-Ebene** stehende Wert ausgegeben wird.

Es ist sogar nur eine Steuerleitung pro Gitterpunkt notwendig, da in der **UND-Ebene** kein Identifier vorkommt.