

---

# — Informatik I —

## Modul 3: Schaltnetze



**Universität  
Zürich<sup>UZH</sup>**



# Modul 3: Schaltnetze

---

- ❑ Formale Grundlagen logischer Beschreibungen
  - Boolesche Algebra, Schaltalgebra
- ❑ Normal- und Minimalformen
- ❑ Realisierung von Schaltnetzen auf Schalter- und Gatterebene
- ❑ Entwurf von Schaltnetzen
  - Logikminimierung, KV-Diagramme
  - Programmierung von Funktionen
- ❑ Laufzeiteffekte bei Schaltnetzen

# Schaltnetze

---

## □ Schaltnetze:

- Rein kombinatorische logische Schaltungen
- Kein Speicherverhalten
- Logische Funktionen

## □ Beispiele:

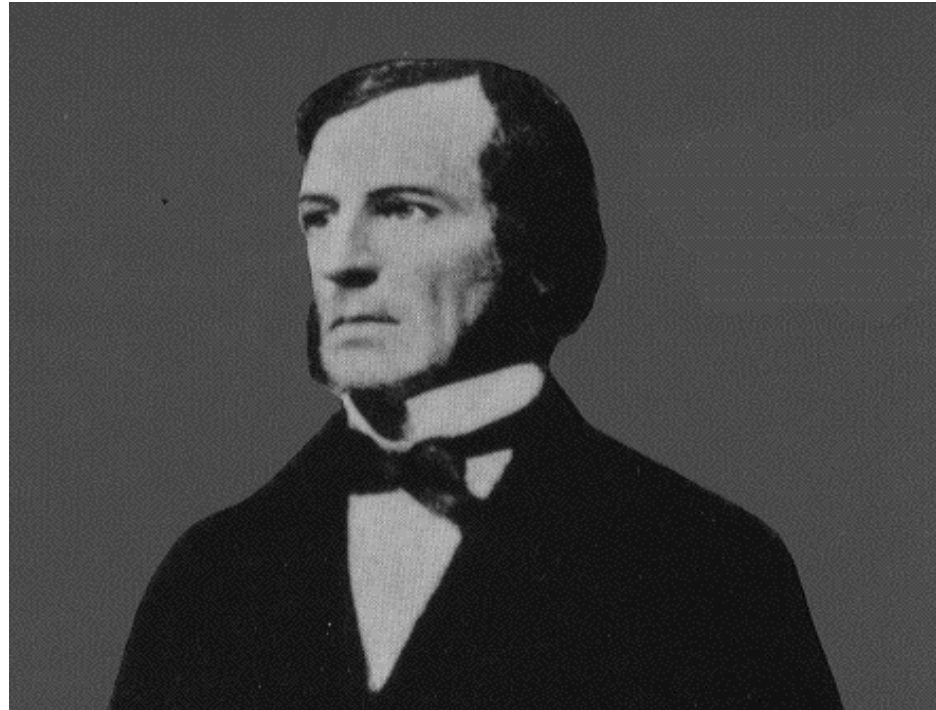
- Licht-Aus Warnung im Kraftfahrzeug
- „Motor aus“ und „Tür auf“ und „Licht an“  $\Rightarrow$  Alarm

# Formale Grundlagen

- Zur Untersuchung und Beschreibung der Eigenschaften und des Verhaltens von logischen Funktionen ist die Boolesche Algebra hervorragend geeignet.
- Entwickelt wurde sie von dem Mathematiker

George Boole  
(1815 – 1864)

als Algebra der Logik.



# Boolesche Algebra

## □ Definition:

- Als eine **Boolesche Algebra** bezeichnet man eine Menge  $V = \{a,b,c,\dots\}$ , auf der zwei zweistellige Operationen  $\oplus$  und  $\otimes$  derart erklärt sind, daß durch ihre Anwendung auf Elemente aus  $V$  wieder Elemente aus  $V$  entstehen (Abgeschlossenheit).

## □ Abgeschlossenheit: Für alle $a, b \in V$ gilt:

$$a \otimes b \in V$$

$$a \oplus b \in V$$

## □ Weiterhin müssen die vier **Huntingtonschen Axiome** gelten.

# Huntingtonsche Axiome

## □ H1 — Kommutativgesetz:

- $a \otimes b = b \otimes a$
- $a \oplus b = b \oplus a$

## □ H2 — Distributivgesetz:

- $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
- $a \oplus (b \otimes c) = (a \oplus b) \otimes (a \oplus c)$

## □ H3 — Neutrales Element:

- Es existieren zwei Elemente  $e, n \in V$ , so dass gilt:
  - $a \otimes e = a$  (e wird **Einselement** genannt)
  - $a \oplus n = a$  (n wird **Nullelement** genannt)

## □ H4 — Inverses Element:

- Für alle  $a \in V$  existiert ein Element  $\bar{a} \in V$ , so dass gilt:
  - $a \otimes \bar{a} = n$
  - $a \oplus \bar{a} = e$

# Schaltalgebra

- Die Schaltalgebra ist eine spezielle Boolesche Algebra, die durch die folgende Korrespondenztabelle definiert wird:

Boolesche Algebra	Schaltalgebra
$\vee$	$\{0,1\}$
$\oplus$	$\vee$
$\otimes$	$\wedge$
<b>n</b>	0
<b>e</b>	1
$\bar{a}$	$\bar{a}$

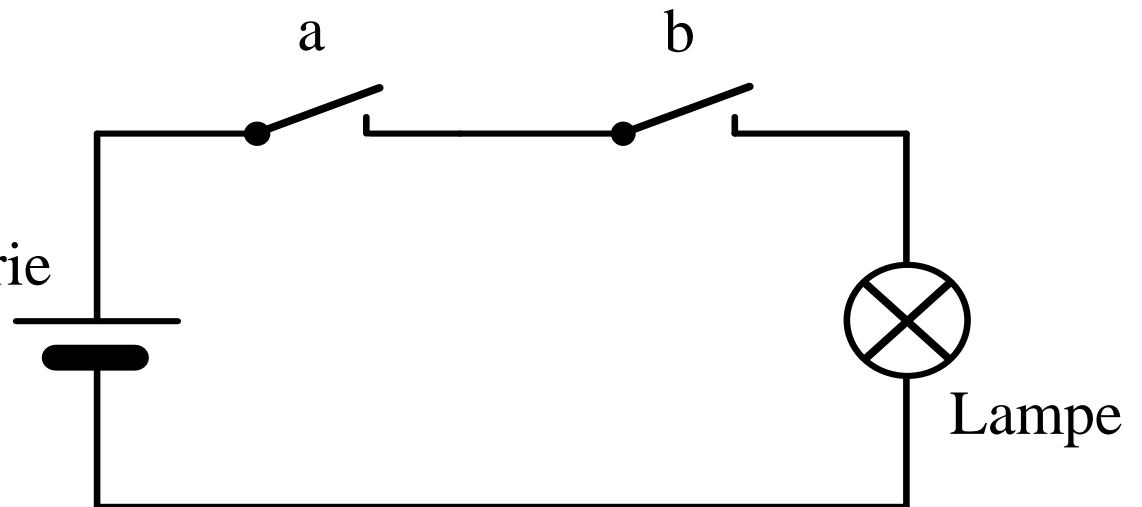
Zusätzlich alternative Schreibweise:

$a + b$  für  $a \vee b$  bzw. Benennung ODER  
 $a \& b, ab$  für  $a \wedge b$  bzw. Benennung UND

# Realisierung von logischen Verknüpfungen (1)

- Für die technische Realisierung logischer Verknüpfungen kann man sich zunächst einfache Schaltermodelle für logische Bausteine vorstellen.

- UND-Verknüpfung: Batterie

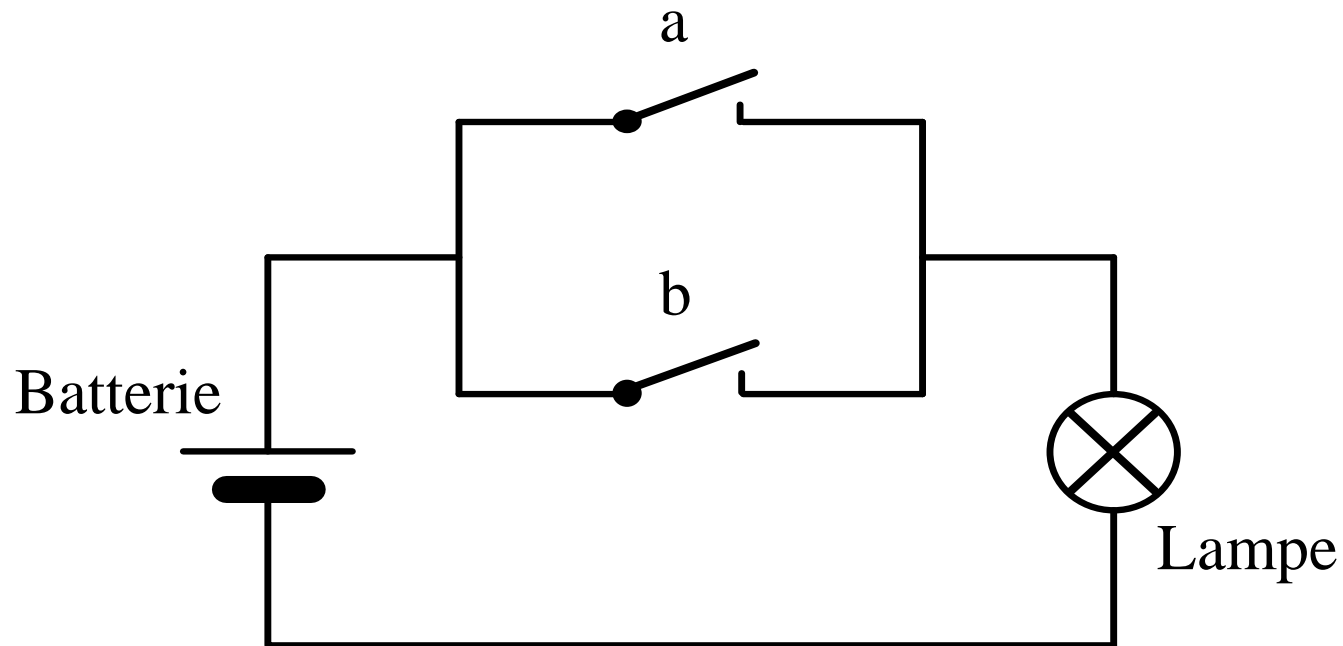


- Die Lampe brennt (Funktionswert 1) nur, wenn beide Schalter geschlossen sind (a UND b gleich 1), sonst bleibt die Lampe dunkel (Funktionswert 0).



# Realisierung von logischen Verknüpfungen (2)

- ODER-Verknüpfung:  
Die Lampe brennt, wenn einer der beiden Schalter geschlossen ist.



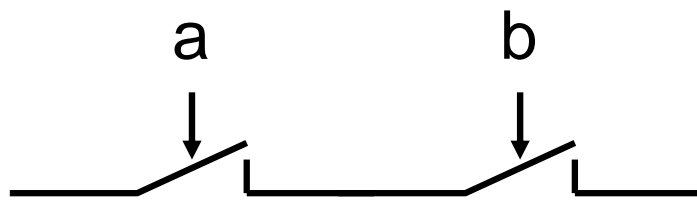
# Schaltalgebra

- Die Verknüpfungen können leicht in Funktionstabellen dargestellt werden:

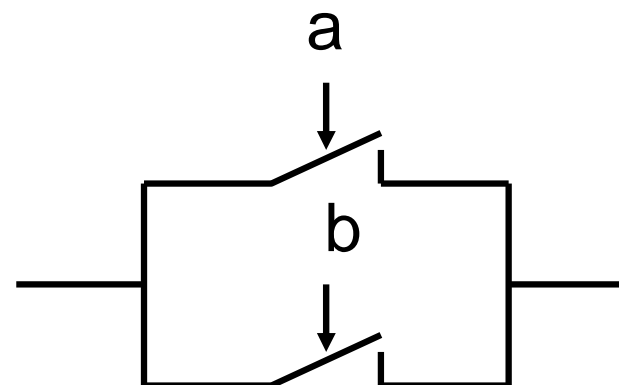
a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

a	$\bar{a}$
0	1
1	0



Technische Realisierung  
mit Schaltern



# Schaltalgebra

---

□ Huntingtonsche Axiome in der Schaltalgebra:

□ H1:             $a \vee b = b \vee a$   
                   $a \wedge b = b \wedge a$

□ H2:             $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$   
                   $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

□ H3:             $a \wedge 1 = a$   
                   $a \vee 0 = a$

□ H4:             $a \wedge \bar{a} = 0$   
                   $a \vee \bar{a} = 1$

# Schaltalgebra

- Aus den vier Huntington'schen Axiomen lassen sich weitere Sätze ableiten:

- Assoziativgesetz:  $(a \wedge b) \wedge c = a \wedge (b \wedge c)$   
 $(a \vee b) \vee c = a \vee (b \vee c)$

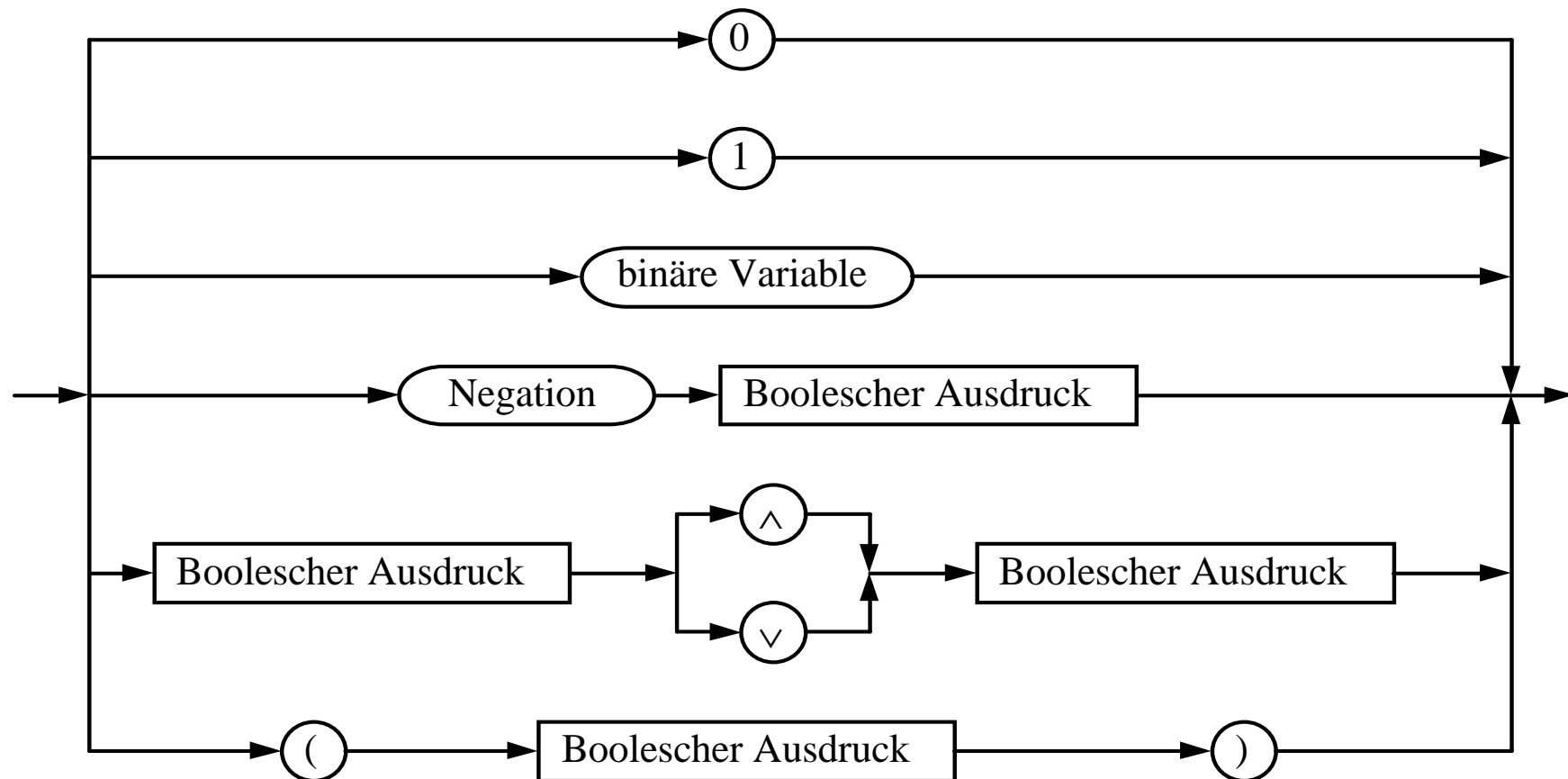
- Idempotenzgesetz:  $a \wedge a = a$   
 $a \vee a = a$

- Absorptionsgesetz:  $a \wedge (a \vee b) = a$   
 $a \vee (a \wedge b) = a$

- DeMorgan-Gesetz:  $\overline{a \wedge b} = \bar{a} \vee \bar{b}$   
 $\overline{a \vee b} = \bar{a} \wedge \bar{b}$  Auch benannt als: NAND  
NOR

# Boolescher Ausdruck (1)

- Ein Boolescher Ausdruck ist eine Zeichenfolge, die aus binären Variablen, den Operatoren  $\wedge$  und  $\vee$  und Klammern besteht und syntaktische Regeln erfüllt, die durch folgendes Syntaxdiagramm gegeben sind:



# Boolescher Ausdruck (2)

- Beispiele:
  - syntaktisch korrekte Boolesche Ausdrücke:  $a \vee b$ ,  $0$ ,  $(\bar{a} \wedge b) \vee c$
- Keine Booleschen Ausdrücke, da syntaktisch **nicht korrekt**:
  - $a \vee \vee a$ ,  $1 0$ ,  $() \vee c$
- Für die Konstanten 0 und 1 verwendet man in der Schaltalgebra manchmal auch in Anlehnung an die Aussagenalgebra die Bezeichnung **Wahrheitswerte**:
  - 0 : falsch            1 : wahr
- Ein Boolescher Ausdruck hat in der Regel zunächst keinen Wahrheitswert, da er binäre Variablen enthalten kann.
  - Erst durch Belegung der binären Variablen mit Wahrheitswerten erhält der Boolesche Ausdruck einen Wahrheitswert.

# Definitionen

---

- Die Belegung einer Menge von binären Variablen eines Booleschen Ausdrucks mit Wahrheitswerten bezeichnet man als **Interpretation**.
- Die Interpretation eines Booleschen Ausdrucks liefert eine **Aussage**, die entweder wahr oder falsch ist.
- Verschiedene Interpretationen eines Booleschen Ausdrucks können zu dem selben Wahrheitswert führen.
- Ein Boolescher Ausdruck, bei dem alle möglichen Interpretationen zum Wahrheitswert „wahr“ führen, heißt **Tautologie**.
  - Beispiel:  $\bar{a} \vee a$  ist eine Tautologie (häufig auch als T markiert).

# Boolesche Funktionen

---

- Gegeben: Tupel von binären Variablen  $(x_1, x_2, \dots, x_n)$
- Definition: Eine (n-stellige) **Boolesche Funktion** ordnet jeder möglichen Wahrheitswertbelegung dieser Variablen genau einen Wahrheitswert zu:

$$f : \{0, 1\}^n \rightarrow \{0, 1\}$$

- Wieviele Belegungen gibt es?  
 $2^n$  Belegungen

- Wieviele verschiedene n-stellige Funktionen gibt es?  
 $2^{(2^n)}$  Funktionen

(denn es gibt zu jedem Argument einer Booleschen Funktion zwei verschiedene Funktionswerte)



# Beispiele

---

- Negation:
  - Eine einstellige Boolesche Funktion

$$f : \{0,1\} \rightarrow \{0,1\}$$

die jedem Operanden aus dem Definitionsbereich  $\{0,1\}$  einen Funktionswert aus dem Wertebereich  $\{0,1\}$  zuordnet.

- $\wedge$  und  $\vee$ :
  - Zwei zweistellige Boolesche Funktionen:

$$f : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$$

# Darstellung boolescher Funktionen

1. Durch eine Funktionstabelle
2. Durch einen algebraischen Ausdruck (symbolische Form)
3. Durch einen Graphen

Funktionstabelle

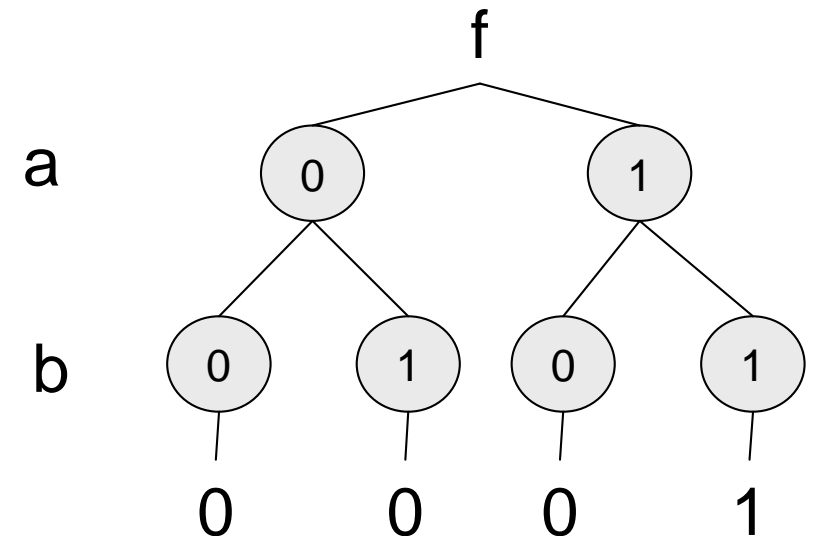
a	b	f
0	0	0
0	1	0
1	0	0
1	1	1

symbolische Form

$$f = a \wedge b$$

Graph

(z.B. Shannon-Baum)



# Boolesche Funktionen

---

- ❑ Wie kommt man von der symbolischen Darstellung zur Funktionstabelle?
- ❑ Durch rekursive Auswertung des symbolischen Ausdrucks!
- ❑ Konvention:
  - Negation vor Konjunktion und Konjunktion vor Disjunktion
  - Durch Klammern kann eine andere Reihenfolge der Auswertung festgelegt werden
- ❑ Wieviele zweistellige Funktionen gibt es?
- ❑ Wieviele dreistellige Funktionen gibt es?

# 16 mögliche zweistellige boolesche Funktionen

$x_1$	0011	verbale Form	symbolische Darstellung	Bezeichnung
$x_0$	0101			
$f_0$	0000	konstant 0	0	Kontradiktion, Symbol: $\perp$ (unerfüllbar)
$f_1$	0001	$x_1$ und $x_0$	$x_1 \wedge x_0$	Konjunktion
$f_2$	0010	nicht $x_0$ , aber $x_1$	$x_1 \wedge \bar{x}_0$	Inhibition
$f_3$	0011	identisch $x_1$	$x_1$	Identität
$f_4$	0100	nicht $x_1$ , aber $x_0$	$\bar{x}_1 \wedge x_0$	Inhibition
$f_5$	0101	identisch $x_0$	$x_0$	Identität
$f_6$	0110	$x_1$ ungleich $x_0$	$x_1 \leftrightarrow x_0$	Antivalenz, XOR
$f_7$	0111	$x_1$ oder $x_0$	$x_1 \vee x_0$	Disjunktion
$f_8$	1000	nicht ( $x_1$ oder $x_0$ )	$x_1 \bar{\vee} x_0$	NOR-Funktion, Peircescher Pfeil
$f_9$	1001	$x_1$ gleich $x_0$	$x_1 \leftrightarrow x_0$	Äquivalenz
$f_{10}$	1010	nicht $x_0$	$\bar{x}_0$	Negation
$f_{11}$	1011	wenn $x_0$ , dann $x_1$	$x_0 \rightarrow x_1$	Implikation
$f_{12}$	1100	nicht $x_1$	$\bar{x}_1$	Negation
$f_{13}$	1101	wenn $x_1$ , dann $x_0$	$x_1 \rightarrow x_0$	Implikation
$f_{14}$	1110	nicht ( $x_1$ und $x_0$ )	$x_1 \bar{\wedge} x_0$	NAND-Funktion, Shefferscher Strich
$f_{15}$	1111	konstant 1	1	Tautologie, Symbol: T (allgemeingültig)

# Vollständige Operatorensysteme

- Definition:

Ein System von Operatoren, mit dem alle booleschen Funktionen dargestellt werden können, heißt **vollständiges Operatorensystem**.

- Die Operatoren  $(\wedge, \vee, \bar{\phantom{x}})$  bilden ein vollständiges Operatorensystem.

- Beispiel:

$a \leftrightarrow b$  liefert das gleiche Ergebnis wie  $(a \wedge b) \vee (\bar{a} \wedge \bar{b})$ .

$\leftrightarrow$  läßt sich durch die Grundoperationen  $\wedge$ ,  $\vee$  und  $\bar{\phantom{x}}$  ersetzen

# Vollständige Operatorensysteme

Operatoren- system	Darstellung der ...		
	Negation	Konjunktion	Disjunktion
$(\wedge, \vee, \neg)$	$\bar{a}$	$a \wedge b$	$a \vee b$
$(\wedge, \neg)$	$\bar{a}$	$a \wedge b$	$\overline{\bar{a} \wedge \bar{b}}$
$(\vee, \neg)$	$\bar{a}$	$\overline{\bar{a} \vee \bar{b}}$	$a \vee b$
$(\bar{\wedge})$	$a \bar{\wedge} a$	$(a \bar{\wedge} b) \bar{\wedge} (a \bar{\wedge} b)$	$(a \bar{\wedge} a) \bar{\wedge} (b \bar{\wedge} b)$
$(\bar{\vee})$	$a \bar{\vee} a$	$(a \bar{\vee} a) \bar{\vee} (b \bar{\vee} b)$	$(a \bar{\vee} b) \bar{\vee} (a \bar{\vee} b)$
$(\wedge, \leftrightarrow)$	$a \leftrightarrow 1$	$a \wedge b$	$a \leftrightarrow b \leftrightarrow (a \wedge b)$

Hinweis:  $\wedge$  wird häufig weggelassen, Bsp:  $a \wedge b \leftrightarrow ab$

# Tautologie (1)

- Wann repräsentieren zwei Ausdrücke A und B dieselbe Boolesche Funktion?
- Gleichbedeutend:  
Ist  $A \leftrightarrow B$  eine Tautologie?
- Gegeben seien zwei Boolesche Funktionen:  
$$f_1(a,b) = (a \wedge b) \vee (\bar{a} \wedge \bar{b})$$
$$f_2(a,b) = (a \vee \bar{b}) \wedge (\bar{a} \vee b)$$
- Ist  $f_1$  identisch mit  $f_2$   
oder  
ist  $(a \wedge b) \vee (\bar{a} \wedge \bar{b}) \leftrightarrow (a \vee \bar{b}) \wedge (\bar{a} \vee b)$  eine Tautologie?

# Tautologie (2)

- Beweis mit Hilfe von Funktionstabellen oder mittels Umformungen von Ausdrücken unter Verwendung der algebraischen Gesetze.
- Zwei Ausdrücke sind äquivalent, falls die Ergebnisse ihrer Auswertung für alle möglichen Kombinationen von Variablenbelegungen identisch sind.

$a$ $b$	$(a \wedge b) \vee (\bar{a} \wedge \bar{b})$	$(a \vee \bar{b}) \wedge (\bar{a} \vee b)$	$x \leftrightarrow y$
0 0	1	1	1
0 1	0	0	1
1 0	0	0	1
1 1	1	1	1



# Tautologie (3)

- Mittels algebraischer Umformung:

$$(a \wedge b) \vee (\bar{a} \wedge \bar{b}) = [(a \wedge b) \vee \bar{a}] \wedge [(a \wedge b) \vee \bar{b}]$$

(Distributivgesetz)

$$= [(a \vee \bar{a}) \wedge (b \vee \bar{a})] \wedge [(a \vee \bar{b}) \wedge (b \vee \bar{b})]$$

(Distributivgesetz)

$$= [1 \wedge (b \vee \bar{a})] \wedge [(a \vee \bar{b}) \wedge 1]$$

(Inverses Element)

$$= (b \vee \bar{a}) \wedge (a \vee \bar{b})$$

(Neutrales Element)

$$= (\bar{a} \vee b) \wedge (a \vee \bar{b})$$

(Kommutativgesetz)

# Modul 3: Schaltnetze

---

- Formalen Grundlagen logischer Beschreibungen
  - Boolesche Algebra, Schaltalgebra
- Normal- und Minimalformen
- Realisierung von Schaltnetzen auf Schalter- und Gatterebene
- Entwurf von Schaltnetzen
  - Logikminimierung, KV-Diagramme
  - Programmierung von Funktionen
- Laufzeiteffekte bei Schaltnetzen

# Normalformen

---

- Eine boolesche Funktion kann durch verschiedene boolesche Ausdrücke beschrieben werden.
- Eine Standarddarstellung boolescher Funktionen im vollständigen Operatorensystem  $(\wedge, \vee, \bar{\phantom{x}})$  ist die **konjunktive (KNF)** und die **disjunktive Normalform (DNF)**.
- Definition:  
Ein **Literal**  $L_i$  ist entweder eine Variable  $x_i$  oder ihre Negation  $\bar{x}_i$   
d.h.,  $L_i \in \{x_i, \bar{x}_i\}$

# Produktterme

- Definition:

Ein **Produktterm**  $K(x_1, \dots, x_m)$  ist die **Konjunktion von Literalen**

$$\bigwedge_{i \in \{1, \dots, m\}} L_i = L_1 \wedge \dots \wedge L_m$$

oder die Konstante "0" oder "1 "

- Beispiele:  $a \wedge a \wedge b$        $\bar{x}_i \wedge x_i$

- Jeder Produktterm  $K(x_1, \dots, x_m) = \bigwedge_{i \in \{1, \dots, m\}} L_i$  kann so dargestellt werden, daß eine Variable  $x$  in höchstens einem Literal vorkommt.

# Literale und Produktterme

- Falls  $L_h = x$ ,  $L_j = x$ ,  $h \neq j$  (mehrfach bejahtes Auftauchen)
  - $\Rightarrow L_h \wedge L_j = x$
  - $\Rightarrow K(x_1, \dots, x_m) = \bigwedge_{i=1}^m L_i$
  
- Mehrfaches Auftreten von  $x$  kann nach Idempotenzgesetz gestrichen werden ( $x \wedge x = x$ ).
  
- Falls  $L_h = x$  und  $L_j = \bar{x}$  (gemischtes bejahtes und negiertes Auftreten)
  - $\Rightarrow L_h \wedge L_j = 0$
  - $\Rightarrow K(x_1, \dots, x_m) = 0$  (Produktterm wird zu 0)

# Implikant und Minterm

---

- Definition:

Ein Produktterm  $K(x_1, \dots, x_n)$  heißt **Implikant** einer Booleschen Funktion  $y(x_1, \dots, x_n)$ , wenn  $K \rightarrow y$

- Das heißt, für jede Belegung  $B \in \{0, 1\}^n$  gilt:  
Wenn  $K(B) = 1$ , dann ist auch  $y(B) = 1$

- Definition:

Ein Implikant einer Booleschen Funktion  $y(x_1, \dots, x_n)$  heißt **Minterm**, wenn ein Literal jeder Variablen  $x_i$  der Funktion  $y$  im Implikanten genau einmal vorkommt.

# Minterme

---

- Minterme einer Booleschen Funktion  $y(x_1, \dots, x_4)$ :

$$x_1 \wedge x_2 \wedge x_3 \wedge x_4$$

$$x_1 \wedge \bar{x}_2 \wedge x_3 \wedge x_4$$

- Keine Minterme der Booleschen Funktion  $y(x_1, \dots, x_4)$ :

$$x_1 \wedge x_2$$

$$x_1 \wedge \bar{x}_2 \wedge x_3 \wedge x_3 \wedge x_4$$

# Disjunktive Normalform

---

- Damit läßt sich die disjunktive Normalform definieren:
- Definition:  
Es sei eine Boolesche Funktion  $y(x_1, \dots, x_n)$  gegeben. Ein Boolescher Ausdruck heißt **disjunktive Normalform (DNF)** der Funktion  $y$ , wenn er aus einer **disjunktiven Verknüpfung aller Minterme  $K_i$**  besteht:

$$y = K_0 \vee K_1 \vee \dots \vee K_k, \quad k \leq 2^n - 1$$

- Es darf dabei keine zwei Konjunktionen  $K_i, K_j$  mit  $i \neq j$  geben, die zueinander äquivalent sind.



# Disjunktive Normalform

---

## □ Beispiele

□  $f(a,b,c) = abc \vee a\bar{b}c \vee \bar{a}\bar{b}c$   
ist in DNF.

□  $f(a,b,c) = abc \vee a\bar{b} \vee cab \vee a(bc \vee \bar{b}\bar{c})$   
ist nicht in DNF, denn:

- $a\bar{b}$  enthält nicht alle Variablen
- $abc$  und  $cab$  sind äquivalent
- $a(bc \vee \bar{b}\bar{c})$  ist keine reine Konjunktion

# Implikat

- Definition:

Es sei  $D(x_1, \dots, x_m)$  eine Disjunktion von Literalen  
 $\bigvee_{i \in \{1, \dots, m\}} L_i = L_1 \vee \dots \vee L_m$  oder die Konstante "0" oder "1"

- Der Term  $D(x_1, \dots, x_m)$  heißt **Implikat** einer Booleschen Funktion

$$y(x_1, \dots, x_m), \text{ wenn } \bar{D} \rightarrow \bar{y}$$

- Das heißt für jede Belegung  $B \in \{0,1\}^n$  gilt:  
Wenn  $D(B) = 0$ , dann ist auch  $y(B) = 0$ .

# Maxterm

---

- Definition:

Ein Implikat einer Booleschen Funktion  $y(x_1, \dots, x_m)$  heißt **Maxterm**, wenn ein Literal jeder Variable  $x_i$  der Funktion  $y$  im Implikaten genau einmal vorkommt.

- Maxterm-Beispiele für die Booleschen Funktion  $y(x_1, \dots, x_3)$ :

$$\begin{aligned} & x_1 \vee x_2 \vee x_3 \\ & x_1 \vee \bar{x}_2 \vee x_3 \end{aligned}$$

# Konjunktive Normalform

---

- Definition:  
Es sei eine Boolesche Funktion  $y(x_1, \dots, x_m)$  gegeben.
- Ein Boolescher Ausdruck heißt **konjunktive Normalform (KNF)**, wenn er aus einer konjunktiven Verknüpfung aller Maxterme  $D_i$  besteht:
- $y = D_0 \wedge D_1 \wedge \dots \wedge D_k$ ,  $k \leq 2^n - 1$
- Es darf dabei keine zwei Disjunktionen  $D_i, D_j$  mit  $i \neq j$  geben, die zueinander äquivalent sind.

# Deutung: Disjunktive/Konjunktive Normalform

---

- Jeder **Minterm** einer **DNF** entspricht einer Zeile in der Funktionstabelle, die den **Funktionswert 1** liefert.
- Jeder **Maxterm** einer **KNF** entspricht einer Zeile in der Funktionstabelle, die den **Funktionswert 0** liefert.
- Disjunktive und konjunktive Normalformen sind **eindeutige Darstellungen!**
  - Bis auf Permutationen (z.B. abc, acb, bac, bca, cab, cba sind äquivalent)

# DNF und KNF

- In einer Funktion mit n Variablen *können* bis zu  $2^n$  Minterme bzw. Maxterme auftreten. Für  $n = 3$  sind diese:

	Minterm	Maxterm
0	$\bar{a} \bar{b} \bar{c}$	$a \vee b \vee c$
1	$a \bar{b} \bar{c}$	$\bar{a} \vee b \vee c$
2	$\bar{a} b \bar{c}$	$a \vee \bar{b} \vee c$
3	$a b \bar{c}$	$\bar{a} \vee \bar{b} \vee c$
4	$\bar{a} \bar{b} c$	$a \vee b \vee \bar{c}$
5	$a \bar{b} c$	$\bar{a} \vee b \vee \bar{c}$
6	$\bar{a} b c$	$a \vee \bar{b} \vee \bar{c}$
7	$a b c$	$\bar{a} \vee \bar{b} \vee \bar{c}$

# Beispiel: DNF und KNF

Um eine Funktion zu beschreiben, reicht die Angabe aller Minterme (oder aller Maxterme) aus.

	c b a	y	Minterme	Maxterme
	0 0 0	1	$\bar{a} \bar{b} \bar{c}$	
	0 0 1	0		$\bar{a} \vee b \vee c$
	0 1 0	0		$a \vee \bar{b} \vee c$
	0 1 1	1	$a b \bar{c}$	
	1 0 0	1	$\bar{a} \bar{b} c$	
	1 0 1	0		$\bar{a} \vee b \vee \bar{c}$
	1 1 0	0		$a \vee \bar{b} \vee \bar{c}$
	1 1 1	1	$a b c$	

$$\text{DNF: } y = (\bar{a} \bar{b} \bar{c}) \vee (a b \bar{c}) \vee (\bar{a} \bar{b} c) \vee (a b c)$$

$$\text{KNF: } y = (\bar{a} \vee b \vee c) (a \vee \bar{b} \vee c) (\bar{a} \vee b \vee \bar{c}) (a \vee \bar{b} \vee \bar{c})$$

# Herkunft der Bezeichnungen

---

- Funktionen aus genau einem Minterm liefern für genau eine Belegung den Funktionswert 1, d.h., abgesehen von der trivialen Nullfunktion haben sie eine minimale Anzahl an Einsen.
- Entsprechend liefern Funktionen aus nur einem Maxterm für genau eine Belegung als Ergebnis 0, d.h., sie haben abgesehen von der Einsfunktion die maximale Anzahl an Einsen.



# DNF oder KNF aus der Funktionstabelle

---

## □ DNF:

Aus der Funktionstabelle einer Funktion erhält man die Minterme, indem man in allen Zeilen mit dem Funktionswert 1 jeweils alle Eingangsvariablen mit  $\wedge$  verknüpft und dabei Eingangsvariablen mit dem Wert 0 negiert. Durch die disjunktive Verknüpfung dieser Minterme kann ein Boolescher Funktionsausdruck in DNF hergeleitet werden.

## □ KNF:

Aus der Funktionstabelle einer Funktion erhält man die Maxterme, indem man in allen Zeilen mit dem Funktionswert 0 jeweils alle Eingangsvariablen mit  $\vee$  verknüpft und dabei Eingangsvariablen mit dem Wert 1 negiert. Durch die konjunktive Verknüpfung dieser Minterme kann ein Boolescher Funktionsausdruck in KNF hergeleitet werden.

# DNF oder KNF aus beliebiger Form

- Um Funktionen aus der DF bzw. KF in die DNF bzw. KNF zu überführen, ist der **Shannonsche Entwicklungssatz** behilflich.
- Entwicklung nach der Variablen  $x_i$ :
  - die Variable wird in der Funktion auf den Wert 1 gesetzt,
  - der entstehende Term konjunktiv mit  $x_i$  verknüpft,und  $\vee$ -verknüpft mit:
  - die Variable wird in der Funktion auf den Wert 0 gesetzt und
  - der entstehende Term konjunktiv mit  $\bar{x}_i$  verknüpft

$$y = f(x_1, \dots, x_n)$$
$$= [x_i \wedge f(x_1, \dots, x_{i-1}, \mathbf{1}, x_{i+1}, \dots, x_n)] \vee [\bar{x}_i \wedge f(x_1, \dots, x_{i-1}, \mathbf{0}, x_{i+1}, \dots, x_n)]$$

# Beispiel:

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

Shannon-Entwicklung nach  $a$  und  $\bar{a}$

$$= a [1 \bar{b} c \vee 1 \bar{b} \vee b c] \vee \bar{a} [0 \bar{b} c \vee 0 \bar{b} \vee b c]$$

H4:  $1 \bar{b} = 0$  und H3:  $x \vee 0 = x$

$$= a [\bar{b} c \vee b c] \vee \bar{a} [\bar{b} \vee b c]$$

Syntaktische Anpassung der Terme, Sortierung von  $b$  nach nicht- und negierten Literalen und H3:  $x \wedge 1 = x$

$$= a [b(c) \vee \bar{b}(c)] \vee \bar{a} [b(c) \vee \bar{b}(1)]$$

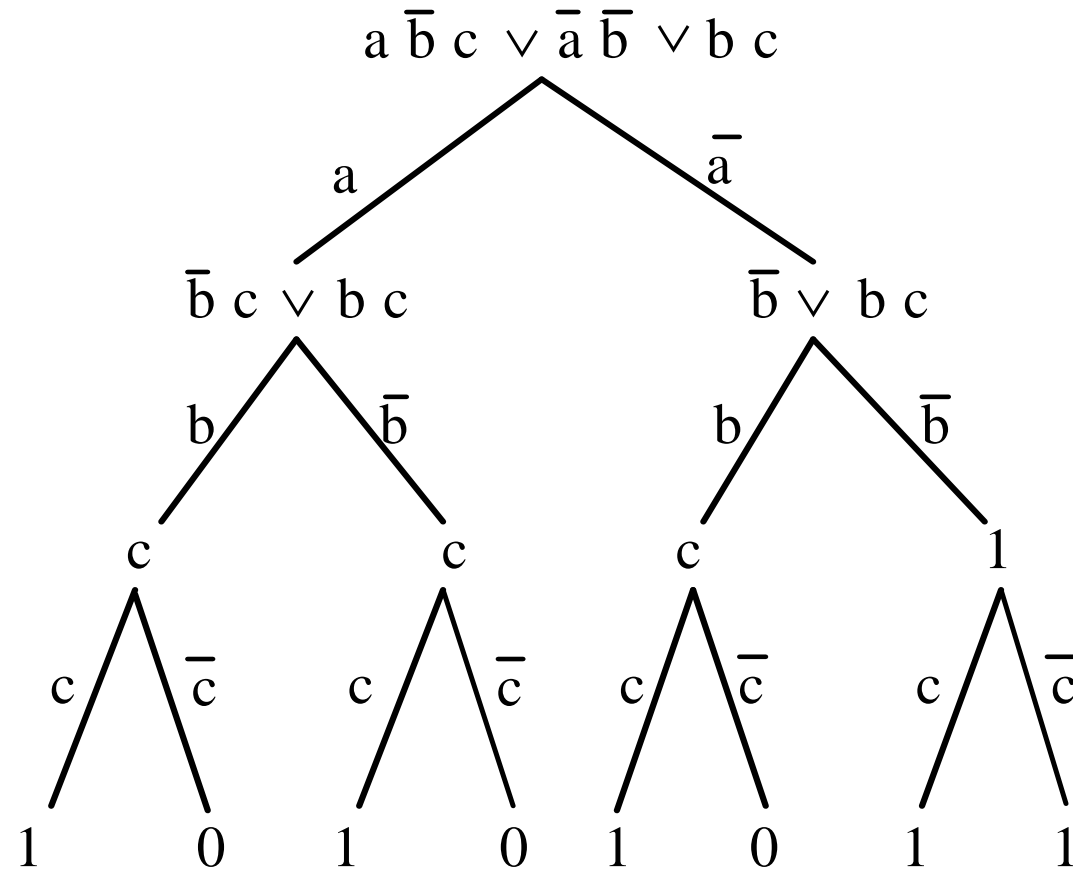
Erweiterung von  $c$  im letzten Term über H4:  $c \vee \bar{c} = 1$

$$= a [b(c) \vee \bar{b}(c)] \vee \bar{a} [b c \vee \bar{b}(c \vee \bar{c})]$$

Distributivgesetz (Ausmultiplizieren)

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

# Beispiel: Shannon-Baum



Nachdem die Funktion nach allen Variablen entwickelt wurde, können die Minterme durch Verfolgen der Äste des Baums gefunden werden, die zu einer 1 führen.

# DNF und KNF

Wiederholung:

- Disjunktive und konjunktive Normalformen sind **eindeutige Darstellungen!**
  - bis auf Permutationen (z.B. abc, acb, bac, bca, cab, cba sind äquivalent)

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

- DNF:  $y = \bar{a} \bar{b} c \vee \bar{a} b c \vee a \bar{b} \bar{c} \vee a \bar{b} c \vee a b c$
- KNF:  $y = (a \vee b \vee c) \wedge (a \vee \bar{b} \vee c) \wedge (\bar{a} \vee \bar{b} \vee c)$

# Minimalformen (1)

- Ziele:
  - „Möglichst kurze“ Boolesche Ausdrücke für eine gegebene Boolesche Funktion.
  - Technische Realisierung einer Schaltung mit möglichst geringen Kosten.
  
- Ähnlich zum Aufbau der disjunktiven und konjunktiven Normalform gibt es eine **disjunktive (DMF)** und **konjunktive (KMF) Minimalform**.
  
- Es kann mehrere disjunktive und konjunktive Minimalformen für die gleiche Funktion geben.
  - Beispiel:
    - $y = a \bar{b} \vee b \bar{c} \vee \bar{a} c$  und
    - $y = a \bar{c} \vee \bar{b} c \vee \bar{a} b$stellen dieselbe Funktion dar, beides sind disjunktive Minimalformen.

# Minimalformen (2)

---

- Das Auffinden einer Minimalform ist insbesondere für Funktionen mit einer größeren Anzahl von Variablen keine triviale Aufgabe.
- Oft können nur suboptimale Lösungen unter Verwendung von Heuristiken gefunden werden.
- Bei Minimierungsverfahren geht man in zwei Schritten vor:
  - Es wird eine Menge von Implikanten bzw. Implikate der Funktion  $y$  mit einer möglichst geringen Anzahl von Literalen gebildet.
  - Aus dieser Menge wird eine möglichst geringe Anzahl von Implikanten bzw. Implikate herausgesucht, deren Disjunktion bzw. Konjunktion die Funktion  $y$  ergeben.

# NAND/NOR-Konversion

- $(\overline{\wedge})$ -System (NAND-System) und  $(\overline{\vee})$ -System (NOR-System) sind **vollständige Operatorensysteme**
  - ⇒ beliebige disjunktive und konjunktive Ausdrücke können mit NAND- und NOR-Verknüpfungen dargestellt werden.
  
- Überführungen (vier Fälle):
  1. Fall: Funktion in disjunktiver Form  $\Rightarrow (\overline{\wedge})$ -System
  2. Fall: Funktion in disjunktiver Form  $\Rightarrow (\overline{\vee})$ -System
  3. Fall: Funktion in konjunktiver Form  $\Rightarrow (\overline{\vee})$ -System
  4. Fall: Funktion in konjunktiver Form  $\Rightarrow (\overline{\wedge})$ -System
  
- Warum sind diese Überführungen relevant?
  - ⇒ Einfache Implementierung in Hardware!
    - NANDs/NORs sind sehr einfach in Schaltungen realisierbar.



# NAND-Konvertierung (Beispiel: 1. Fall)

---

- 1. Fall:  
Funktion in disjunktiver Form  $\Rightarrow$  ( $\bar{\wedge}$ )-System

Gegeben sei eine Funktion in disjunktiver Form.

- Überführung:
  1. Doppelte Negation
  2. Anschließende Anwendung der DeMorganschen Regeln
- Dann erhält man einen Ausdruck, der nur noch NAND als Operator enthält.

# Beispielrechnung

$$\begin{aligned}y &= \bar{a} b c \vee a \bar{b} c \vee a b \bar{c} \vee \bar{a} \bar{b} \bar{c} \\ &= \overline{\overline{\bar{a} b c} \vee \overline{a \bar{b} c} \vee \overline{a b \bar{c}} \vee \overline{\bar{a} \bar{b} \bar{c}}} \\ &= \overline{\bar{a} b c} \wedge \overline{a \bar{b} c} \wedge \overline{a b \bar{c}} \wedge \overline{\bar{a} \bar{b} \bar{c}} \\ &= \text{NAND}_4(\text{NAND}_3(\bar{a}, b, c), \text{NAND}_3(a, \bar{b}, c), \\ &\quad \text{NAND}_3(a, b, \bar{c}), \text{NAND}_3(\bar{a}, \bar{b}, \bar{c}))\end{aligned}$$

Dabei ist  $\text{NAND}_k(x_1, \dots, x_k)$  eine  $k$ -stellige Funktion, für die gilt:

$$\text{NAND}_k(x_1, \dots, x_k) \begin{cases} 0 \text{ für } x_1 = \dots = x_k = 1 \\ 1 \text{ sonst} \end{cases}$$

# NAND<sub>2</sub>-Funktion

Darstellung der NAND<sub>2</sub>-Funktion durch den  $\overline{\wedge}$  Operator:

**Problem:** Die Operatoren  $\overline{\wedge}$  und  $\overline{\vee}$  sind nicht assoziativ.

$$(x_1 \overline{\wedge} x_2) \overline{\wedge} x_3 \neq x_1 \overline{\wedge} (x_2 \overline{\wedge} x_3)$$

$$(x_1 \overline{\vee} x_2) \overline{\vee} x_3 \neq x_1 \overline{\vee} (x_2 \overline{\vee} x_3)$$

$$\text{NAND}_3(x_1, x_2, x_3) = \overline{x_1 \wedge x_2 \wedge x_3} = \overline{(x_1 \overline{\wedge} x_2) \overline{\wedge} x_3}$$

$$(x_1 \overline{\wedge} x_2) \overline{\wedge} x_3 = \overline{\overline{x_1 \wedge x_2 \wedge x_3}} \neq$$

$$x_1 \overline{\wedge} (x_2 \overline{\wedge} x_3) = \overline{\overline{x_1 \wedge x_2 \wedge x_3}}$$

# Wichtige Zusammenfassung

---

## □ Schaltalgebra

- Boolesche Ausdrücke und Funktionen
- Vollständiger Operatorensysteme

## □ Darstellung

- Funktionstabelle
- Symbolische Form
- Shannon Baum (Graph)

## □ Normalformen

- Produktterm und Literal
- Disjunktive Normalform:
  - Implikant und Minterm
    - Zeile einer Funktionstabelle mit dem Funktionswert 1
- Konjunktive Normalform:
  - Implikat und Maxterm
    - Zeile einer Funktionstabelle mit dem Funktionswert 0

## □ Minimalformen

- Nicht eindeutig, aber „kurz“ in Form notwendiger Min-/Maxterme

# Modul 3: Schaltnetze

---

- Formale Grundlagen logischer Beschreibungen
  - Boolesche Algebra, Schaltalgebra
- Normal- und Minimalformen
- Realisierung von Schaltnetzen auf Schalter- und Gatterebene
- Entwurf von Schaltnetzen
  - Logikminimierung, KV-Diagramme
  - Programmierung von Funktionen
- Laufzeiteffekte bei Schaltnetzen

# Realisierung von Schaltnetzen

## Hierarchie:

- Register-Transfer-Ebene: logische Bausteine als Grundelemente (Grundlage der Programmierung)
  - Multiplexer, Schieberegister, Addierer, ... **In folgenden Modulen ein wenig diskutiert**
- Gatterebene: logische Gatter **In M3 soeben theoretisch gezeigt, jetzt in Gatter übersetzt**
  - UND, ODER, NAND,...
- Schalterebene: Transistoren als Schalter **Beispiel in M1 erwähnt**
  - Elektrotechnische Grundlagen
- Layoutebene: Transistortechnologie **Beispiel in M1 erwähnt**
  - Elektrotechnische Grundlagen

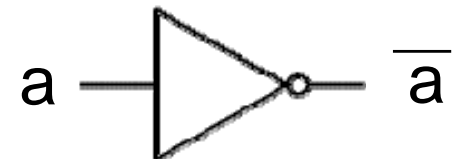
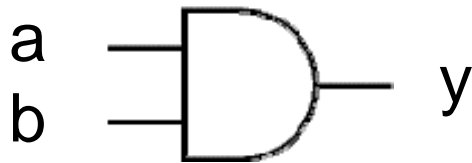
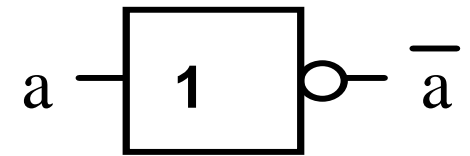
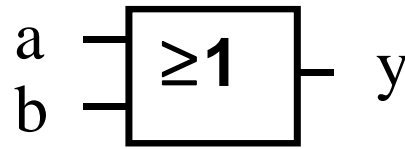
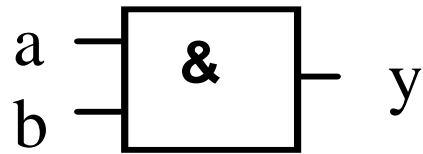
Bottom-up

# Gatterebene

---

- Abstrahierung von der internen Realisierung der Verknüpfungsbausteine
- Beschränkung auf das logische Verhalten
  - Abbildung logischer Funktionen auf Schaltungen ohne tiefergehende elektrotechnische Kenntnisse
- Verknüpfungsbausteine werden durch Schaltsymbole dargestellt

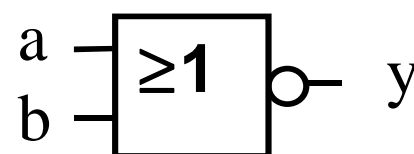
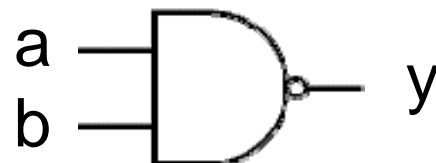
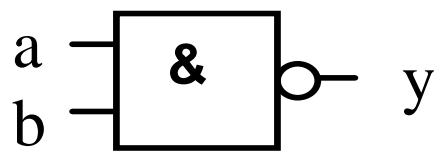
# Schaltsymbole (DIN 40900 Teil 12, ANSI/IEEE-Standard 91-1984, IEC-Standard & US-Symbole)



UND-Verknüpfung

ODER-Verknüpfung

Negation



NAND-Verknüpfung

NOR-Verknüpfung

Verknüpfungsbausteine dieser Art werden **Gatter** genannt.



# Bedeutung der Zeichen

---

& : andere Schreibweise für  $\wedge$

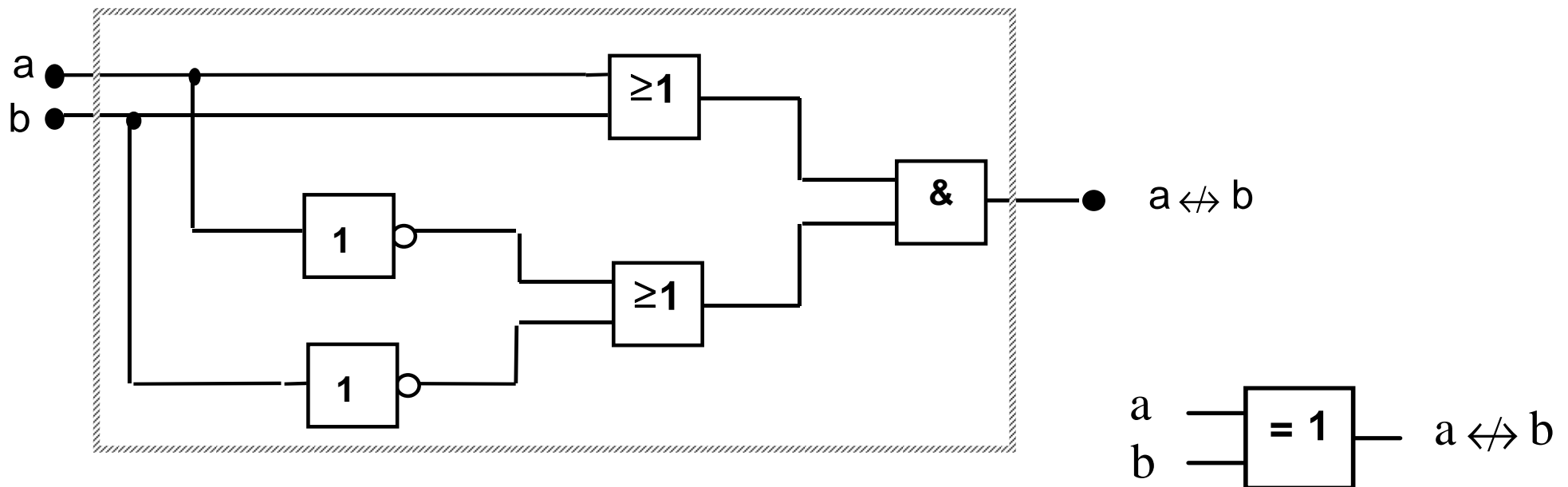
1 : der Ausgang ist genau dann 1,  
wenn an 1 Eingängen eine 1 liegt

◦ : Negation

Weitere Symbole, alte Darstellungen und die Logik hinter den Symbolen finden sich im Web!

# Antivalenz-Gatter

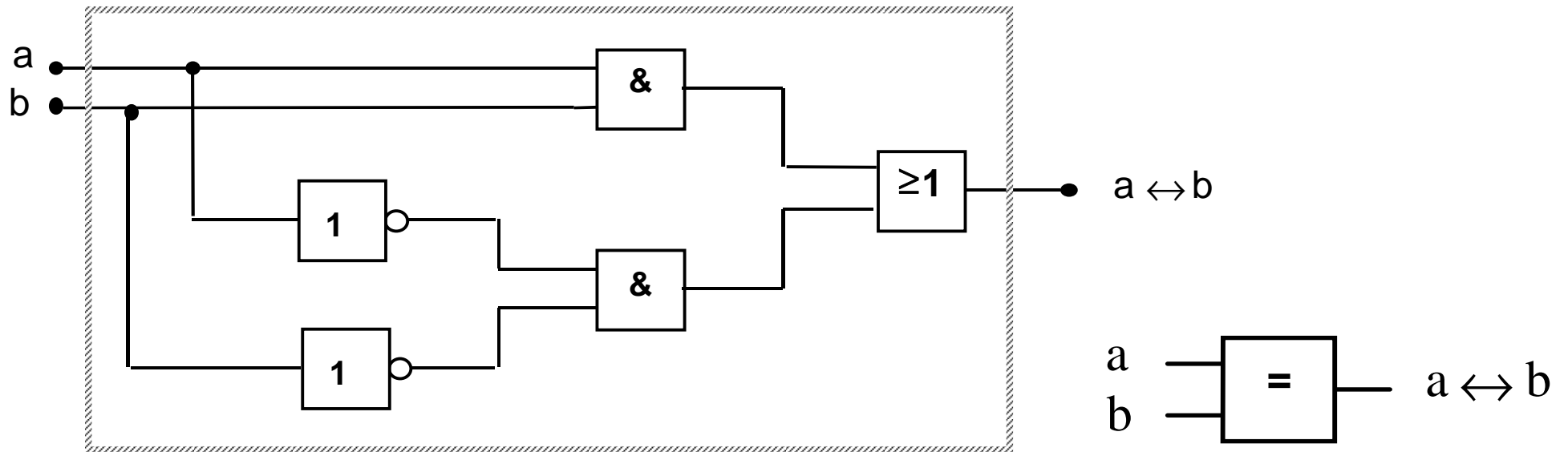
Aus einfacheren Gattern lassen sich hierarchisch komplexere Gatter aufbauen, die teilweise eigene Symbole besitzen.



Antivalenz als Komposition fünf einfacherer Schaltglieder.

Geeignet zur Überlauferkennung (siehe M2)

# Äquivalenz-Gatter



Äquivalenz als Komposition fünf einfacherer Schaltglieder.

# Modul 3: Schaltnetze

---

- Formale Grundlagen logischer Beschreibungen
  - Boolesche Algebra, Schaltalgebra
- Normal- und Minimalformen
- Realisierung von Schaltnetzen auf Schalter- und Gatterebene
- Entwurf von Schaltnetzen
  - Logikminimierung, KV-Diagramme
  - Programmierung von Funktionen
- Laufzeiteffekte bei Schaltnetzen

# Entwurf von Schaltnetzen (1)

- Praktischer Entwurf von Schaltnetzen muß beachten, daß
  - reale Gatter keine idealen Verknüpfungen sind, sondern z.B. Wärme abgeben,
  - reale Gatter Platz benötigen (Quadratmicrometer) und
  - reale Gatter Verzögerungszeiten besitzen (Microsekunden).
- Teil 1: Technische Kriterien:
  - Leistungsaufnahme, Schaltzeit, Platzbedarf, Material, Lebensdauer
  - Korrekte Realisierung unter Beachtung des statischen und dynamischen Verhaltens der verwendeten Bauelemente (Hasards und Wettläufe)
  - Berücksichtigung technischer Beschränkungen, z.B. begrenzte Anzahl von Eingängen der Logikelemente, begrenzte Belastbarkeit von Elementausgängen
- Teil 2: Ökonomische Kriterien
  - Geringe Kosten für den Entwurf (Entwurfsaufwand), z.B. Lohnkosten, Kosten für Rechnerbenutzung zur Entwurfsunterstützung
  - Geringe Kosten für die Realisierung (Realisierungsaufwand), z.B. Kosten für die eingesetzten Bauelemente (bei der Herstellung integrierter Schaltkreise wird die Realisierung auf einer möglichst kleinen Chipfläche gefordert)
  - Geringe Kosten für Inbetriebnahme, laufenden Betrieb, z.B. Test, Wartung

# Entwurf von Schaltnetzen (2)

---

- Grenzen zwischen technischen und ökonomischen Kriterien sind teilweise fließend.
- Einzelne Kriterien stehen auch im Widerspruch zueinander.
  - z.B. Erhöhung der Zuverlässigkeit  $\Rightarrow$  Erhöhung der Kosten
  - geringere Realisierungskosten  $\Rightarrow$  höhere Entwurfskosten
- ➔ Aufgabe des Entwerfers besteht darin, für eine bestimmte Problemstellung den günstigsten Kompromiss zu finden.
- **Ziel des Entwurfs:** Unter Einhaltung bestimmter technischer Kriterien vor allem einen günstigen Kompromiss bezüglich der ökonomischen Kriterien anzustreben, um so zu einem Minimum an Gesamtkosten zu gelangen.

# Minimierungsverfahren

---

- Grundlage:
  - Realisierung der Schaltnetze in zweistufiger Form
- Darstellungsform, deren Realisierung die geringsten Kosten verursacht, bezeichnet man als
  - Minimalform
- Der Vorgang der Erzeugung einer Minimalform wird als
  - Minimierung bezeichnet.
- Es gibt drei Arten von Minimierungsverfahren:
  - Algebraische Verfahren
  - Graphische Verfahren
  - Tabellarische Verfahren
- Algebraische und graphische Verfahren eignen sich nur für Funktionen mit bis zu 5/6 Variablen, danach werden sie zu unübersichtlich. Danach wendet man tabellarische Verfahren an.

# Praktische Einordnung

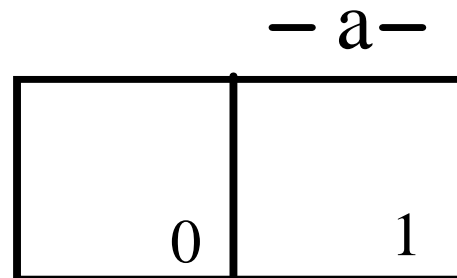
Aufgabenstellung	Primterme	Auswahl
<b>Variablenanzahl <math>\leq 6</math></b>	<b>KV-Diagramm</b>	<b>KV-Diagramm</b> <b>Überdeckungstabelle</b>
Geg. DF(KF) Ges. DMF(KMF) [Disj./Konj. Minimalform]	Consensus Quine-McCluskey	Überdeckungstabelle
Geg. DF(KF) Ges. KMF(DMF) [Disj./Konj. Form]	<b>Nelson</b>	<b>Überdeckungstabelle</b>

Für erweiterte/leistungsfähigere Verfahren wird auf die Literatur verwiesen!



# Graphische Verfahren

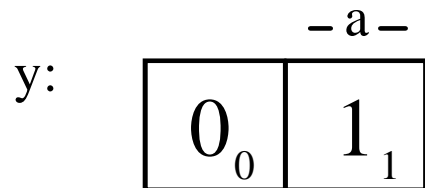
- Das KV-Diagramm (nach Karnaugh und Veith)
  - Auch oft nur Karnaugh map, Karnaugh-Diagramm
- Ausgangspunkt ist ein Rechteck, dessen rechte Hälfte der Variablen  $a$  und dessen linke Hälfte  $\bar{a}$  zugeordnet wird:



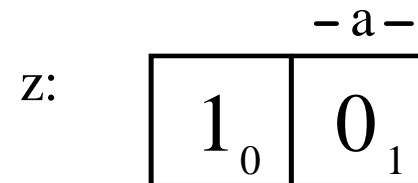
**KV-Diagramm für eine Variable**

# KV-Diagramm (1)

- Die Zahl in den Feldern gibt den Index der Variablenbelegung an:
  - Index des Minterms, der dort den Wert 1 annimmt.
- Durch Eintragen der Wahrheitswerte 0 oder 1 in die Felder des KV-Diagramms wird eine Boolesche Funktion charakterisiert.
- Das KV-Diagramm ist eine weitere Darstellungsform Boolescher Funktionen (Alternative zur Funktionstabelle).



$$y = a$$



$$z = \bar{a}$$

# KV-Diagramme (2)

- KV-Diagramme für mehrere Variable erhält man durch Spiegelung (für jede neue Variable verdoppelt sich die Anzahl der möglichen Belegungen)

a	
0	1
2	3

| b

a			
0	1	5	4
2	3	7	6

| b

c

a			
0	1	5	4
2	3	7	6
10	11	15	14
8	9	13	12

| b

c

d |

# KV-Diagramme (3)

- Jedes Feld hat eine eindeutige Variablenzuordnung, die an den Rändern abgelesen werden kann:

- Feld 11 hat die Zuordnung:  $a b \bar{c} d$ .

- Der Index in den Feldern gibt den Index der zum Feld gehörenden Variablenbelegung an (Variablen in umgekehrter alphabetischer Reihenfolge angetragen).

- Feld 11 =  $1011_2 = d \bar{c} b a$

0	1	5	4
2	3	7	6
10	11	15	14
8	9	13	12

# KV-Diagramme — Erstellung (1)

---

- Funktion sei in Tabellenform gegeben:
  - Jede Zeile der Funktionstabelle entspricht einem Feld im KV-Diagramm.
  - ⇒ Für jede Zeile der Funktionstabelle sucht man das zugehörige Feld im KV-Diagramm und trägt den Funktionswert ein.
  
- **Trick**, um das Auffinden der Felder im KV-Diagramm zu erleichtern:
  - Man schreibt die Eingangsvariablen in „umgekehrter alphabetischer Reihenfolge“ in die Tabelle.
  - Dann kann das KV-Diagramm gemäß der Indizierung seiner Felder mit den Werten ausgefüllt werden, welche die Tabelle beim Durchlaufen von oben nach unten liefert.

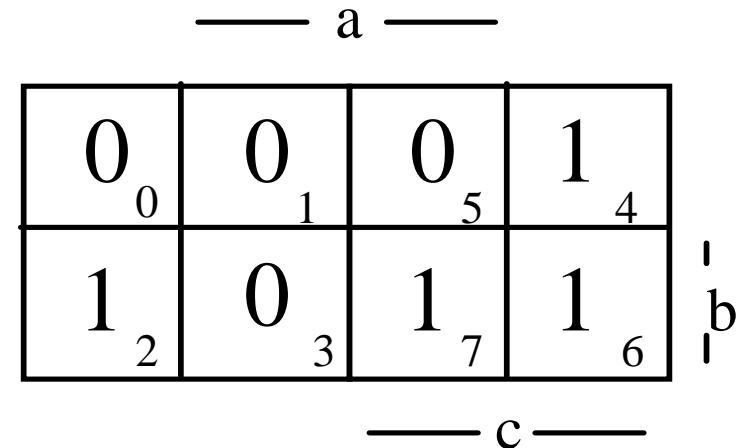
# KV-Diagramme — Erstellung (2)

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

*Funktionstabelle:*

Index	c	b	a	y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Damit ergibt sich das folgende KV-Diagramm:

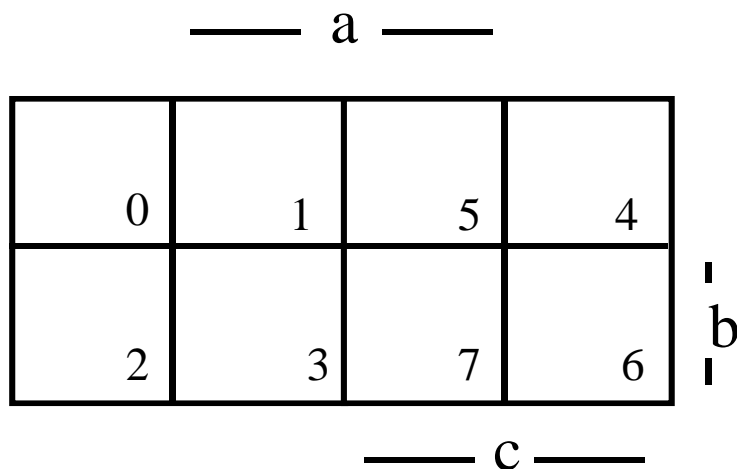


# Eigenschaften der KV-Diagramme (1)

□ Wesentliche Eigenschaft:

- Symmetrisch zu einer Achse liegende Minterme unterscheiden sich lediglich in einer Variablen.

□ Beispiel:



oder

$$\begin{aligned} \text{Minterm 0} &= \bar{c} \bar{b} \bar{a} \\ \text{Minterm 1} &= \bar{c} \bar{b} a \end{aligned}$$

oder

$$\begin{aligned} \text{Minterm 0} &= \bar{c} \bar{b} \bar{a} \\ \text{Minterm 4} &= c \bar{b} \bar{a} \end{aligned}$$

oder

$$\begin{aligned} \text{Minterm 1} &= \bar{c} \bar{b} a \\ \text{Minterm 3} &= \bar{c} b a \end{aligned}$$

# Eigenschaften der KV-Diagramme (2)

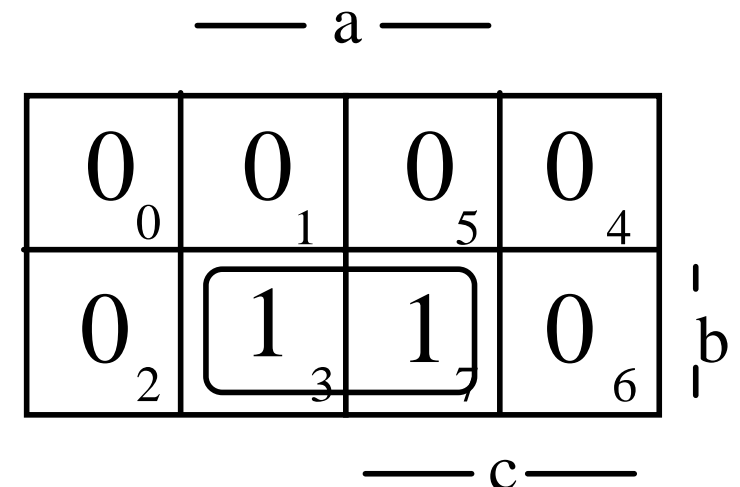
- Nach den Regeln der Booleschen Algebra lassen sich Terme, die sich nur in einer Variablen unterscheiden, zusammenfassen:

- Beispiel:

$$\begin{aligned} a b c \vee a b \bar{c} &= a b (c \vee \bar{c}) \\ &= a b \end{aligned}$$

- Es entsteht ein Term ohne diese Variable.

- Symmetrisch zu den Achsen des KV-Diagramms liegende Minterme lassen sich zu einem einfacheren Term zusammenfassen.





# Definition: Primimplikant

---

- Ein Implikant  $p$  ist **Primimplikant**, falls es keinen Implikanten  $q \neq p$  gibt, der von  $p$  impliziert wird

$$\forall q: q \neq p \Leftrightarrow \neg(p \rightarrow q)$$

- *d.h.*,  $p$  ist von **größtmöglicher Ordnung** ( $p$  umfasst einen maximal großen Einsblock).
- Es gilt:  
Jede Funktion ist als Disjunktion ihrer Primimplikanten darstellbar.

# Herauslesen der Primimplikanten

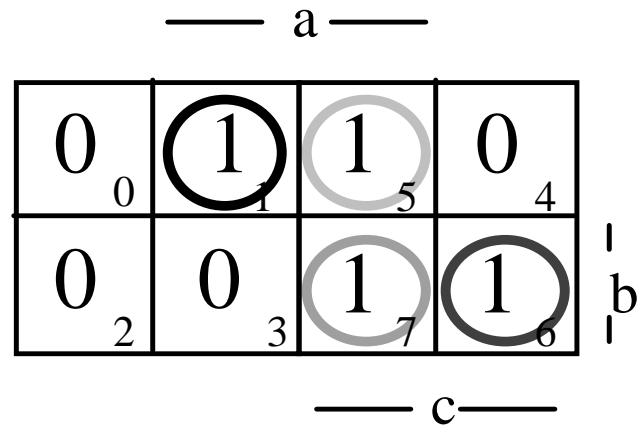
---

- Herauslesen der Primimplikanten aus dem KV-Diagramm:
  - Man versucht, möglichst große Blöcke von Einsen im Diagramm zu finden, wobei jeder Einsblock  $2^k$  Felder umfassen muß.

Beispiel:

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

# Beispiel



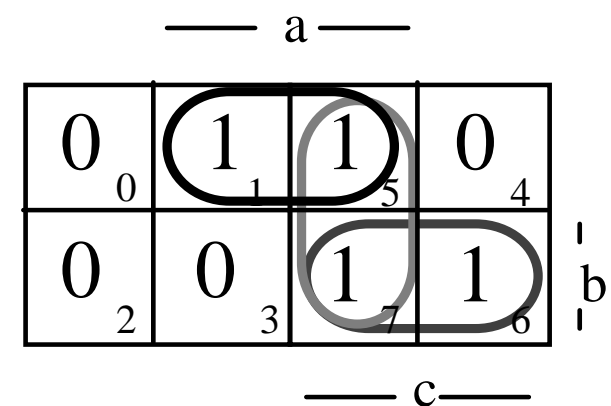
Vier Minterme:

$$(a \bar{b} \bar{c}, a \bar{b} c, a b \bar{c}, \bar{a} b c)$$

Drei Primimplikanten:

Implikanten erster Ordnung  
( $a \bar{b}$ ,  $a c$ ,  $b c$ )

Aber: ( $a \bar{b}$ ,  $b c$ ) genügen  
eigentlich!



$$\begin{aligned} \square f &= a \bar{b} \bar{c} \vee a \bar{b} c \vee a b \bar{c} \vee \bar{a} b c \\ &= a \bar{b} \vee a c \vee b c = a \bar{b} \vee b c \end{aligned}$$

# Herauslesen einer Funktion

Auffrischung:

□ Definition:

Es sei  $D(x_1, \dots, x_m)$  eine Disjunktion von Literalen

$\bigvee L_i = L_1 \vee \dots \vee L_m$  oder die Konstante „0“ oder „1“

□ Der Term  $D(x_1, \dots, x_m)$  heißt **Implikat** einer Booleschen Funktion

$y(x_1, \dots, x_m)$  [f Funktion von  $x_i$ ],

wenn  $\bar{D} \rightarrow \bar{y}$  [ $y$  Produktterm aus Literalen dieser  $x_i$ ]

□ Das heißt für jede Belegung  $B \in \{0, 1\}^n$  gilt:

Wenn  $D(B) = 0$ , dann ist auch  $y(B) = 0$ .

□  $\bar{y}$  ist **Implikant** von  $y$ , wenn  $\bar{y}$  die Funktion  $y$  impliziert, d.h.

wenn die Menge aller Einstellen von  $\bar{y}$  in der Menge aller Einstellen von  $y$  enthalten ist.

# Beispiel (1)

Beispiel:  $f = \text{Minterme } (0,2,4,5,6,7)$

f:

— a —			
1 <sub>0</sub>	0 <sub>1</sub>	1 <sub>5</sub>	1 <sub>4</sub>
1 <sub>2</sub>	0 <sub>3</sub>	1 <sub>7</sub>	1 <sub>6</sub>
— c —			

| b |

— a —			
1 <sub>0</sub>	0 <sub>1</sub>	1 <sub>5</sub>	1 <sub>4</sub>
1 <sub>2</sub>	0 <sub>3</sub>	1 <sub>7</sub>	1 <sub>6</sub>
— c —			

| b |

$g = a \bar{b} c$   
ist Implikant

— a —			
1 <sub>0</sub>	0 <sub>1</sub>	1 <sub>5</sub>	1 <sub>4</sub>
1 <sub>2</sub>	0 <sub>3</sub>	1 <sub>7</sub>	1 <sub>6</sub>
— c —			

| b |

$g = b c$   
ist Implikant

— a —			
1 <sub>0</sub>	0 <sub>1</sub>	1 <sub>5</sub>	1 <sub>4</sub>
1 <sub>2</sub>	0 <sub>3</sub>	1 <sub>7</sub>	1 <sub>6</sub>
— c —			

| b |

$g = \bar{a}$   
ist Implikant

# Beispiel (2)

Beispiel:  $f = \text{Minterme } (0,2,4,5,6,7)$

f:

— a —			
$1_0$	$0_1$	$1_5$	$1_4$
$1_2$	$0_3$	$1_7$	$1_6$
— c —			

| b |

— a —			
$1_0$	$0_1$	$1_5$	$1_4$
$1_2$	$0_3$	$1_7$	$1_6$
— c —			

| b |

$g = b$   
ist KEIN Implikant

— a —			
$1_0$	$0_1$	$1_5$	$1_4$
$1_2$	$0_3$	$1_7$	$1_6$
— c —			

| b |

$g = c$   
ist Implikant

— a —			
$1_0$	$0_1$	$1_5$	$1_4$
$1_2$	$0_3$	$1_7$	$1_6$
— c —			

| b |

$g = \bar{a} \vee c$   
ist KEIN Implikant

(Impl.)

# Implementation von Funktionen

---

- Implementation **regelmäßig wiederkehrender** Funktionen.

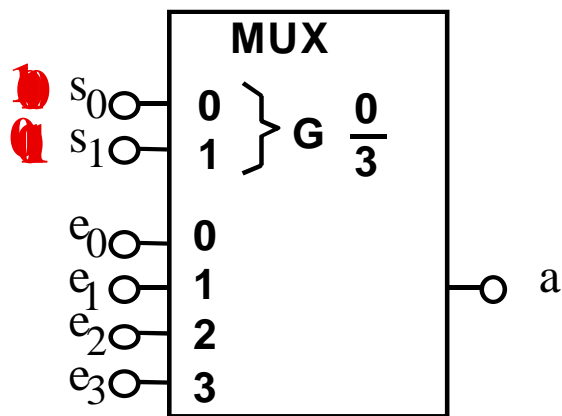
Zwei Varianten werden diskutiert:

- Über Multiplexer/Demultiplexer
- Speicher

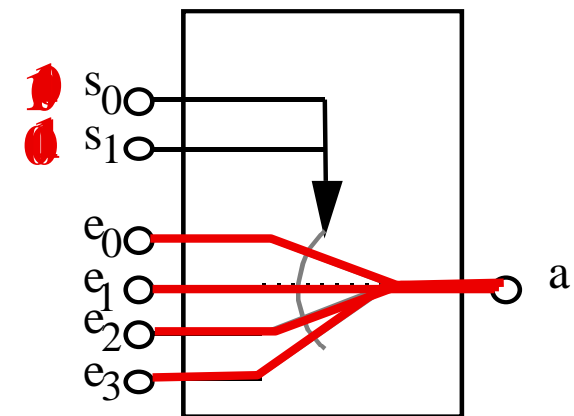
- Ein **Multiplexer** (Abkürzung: **MUX**) ist ein Baustein mit mehreren Eingängen und einem Ausgang, wobei über  $n$  Steuerleitungen einer der  $2^n$  Eingänge auf den Ausgang geschaltet wird.
- Multiplexer werden nach ihrer Größe als  $2^n:1$  - Multiplexer (alternativ als 1-aus- $2^n$ - Multiplexer) klassifiziert.

# 1-aus-4-Multiplexer

- Schaltbild und logisches Verhalten:



$s_1$	$s_0$	$a$
0	0	$e_0$
0	1	$e_1$
1	0	$e_2$
1	1	$e_3$





# Logische Funktionen mit Multiplexern

---

- Ein Multiplexer kann nicht nur zur Steuerung von Datenflüssen sondern auch zur Realisierung logischer Funktionen verwendet werden.
- Man kann mit einem  $2^n: 1$  - Multiplexer eine logische Funktion mit  $n+1$  Variablen implementieren.
- Hierzu wird die sog. **Implementierungstabelle** verwendet.

# Implementierungstabelle

---

- Die Tabelle besteht aus:
  - $2^n$  Spalten für die möglichen Belegungen der  $n$  Steuereingänge
  - 2 Zeilen für die negierte und nicht negierte  $(n+1)$ -te Variable
  
- In die Tabelle werden die Funktionswerte in Abhängigkeit von den Variablen eingetragen.
  
- Anschließend betrachtet man jede Spalte für sich und ordnet ihr eine einstellige Funktion  $g \in \{0, 1, a, \bar{a}\}$  zu, mit der dann der Eingang belegt wird, der zu der entsprechenden Steuervariablenkombination gehört.

# Beispiel

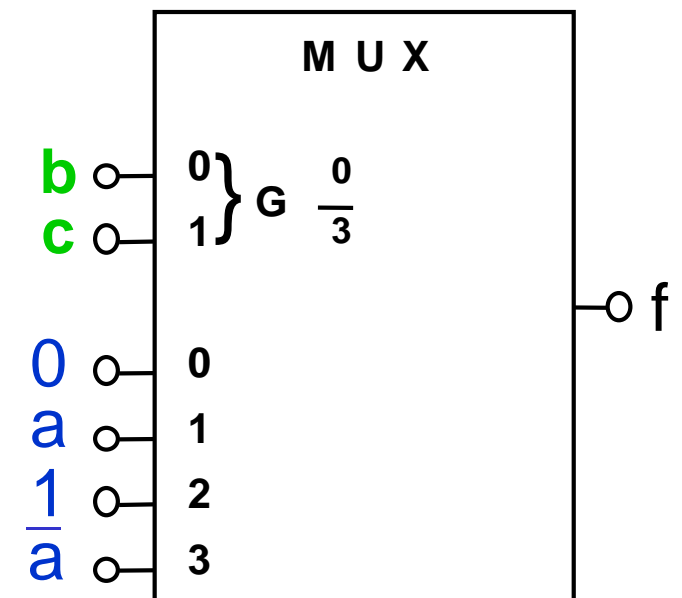
Realisierung einer Funktion mit Multiplexer:

$$f = \bar{a}c \vee \bar{b}c \vee ab\bar{c}$$

Implementierungstabelle bei Wahl von b und c als Steuereingänge:

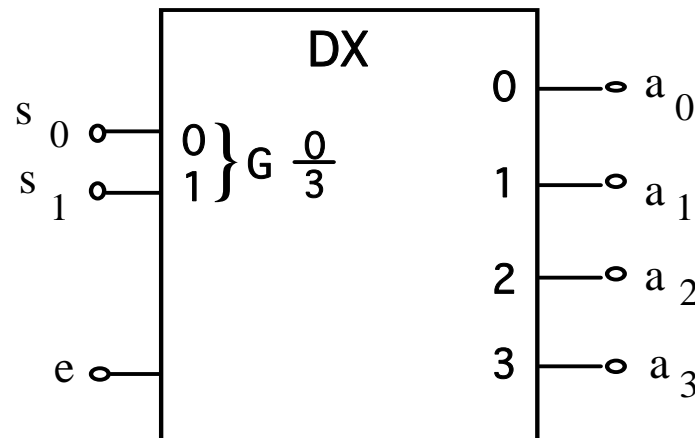
<b>c b</b>	00	01	10	11
a = 0	0	0	1	1
a = 1	0	1	1	0
f =	0	a	1	$\bar{a}$

Realisierung



# Demultiplexer/Dekoder (1)

- Der zum Multiplexer korrespondierende Baustein, der einen Eingang abhängig von  $n$  Steuerleitungen auf einen von  $2^n$  Ausgängen schaltet, heißt **Demultiplexer**.
- Beispiel:



$s_1$	$s_0$	$a_0$	$a_1$	$a_2$	$a_3$
0	0	e	0	0	0
0	1	0	e	0	0
1	0	0	0	e	0
1	1	0	0	0	e

Schaltbild und logisches Verhalten eines 1-auf-4-Demultiplexers

# Demultiplexer/Dekoder (2)

---

- Man beachte:
  - der Demultiplexer hat einen Enable-Eingang  $e$  sowie  $n$  Eingänge  $s_i$  für eine Dualzahl, die an den  $2^n$  Ausgängen  $a_j$  dekodiert bereitgestellt wird.
- Enable-Eingang  $e = 0$ , dann liegen alle Ausgänge auf 0 ansonsten wird eine 2-bit-Zahl dekodiert, z.B. wird bei Anlegen der Zahl 2 ( $s_1 = 1, s_0 = 0$ ) der Ausgang  $a_2 = 1$  und alle anderen Ausgänge bleiben 0.
- Der Demultiplexer wird deshalb auch **Dekoder** genannt.

(Speich.)

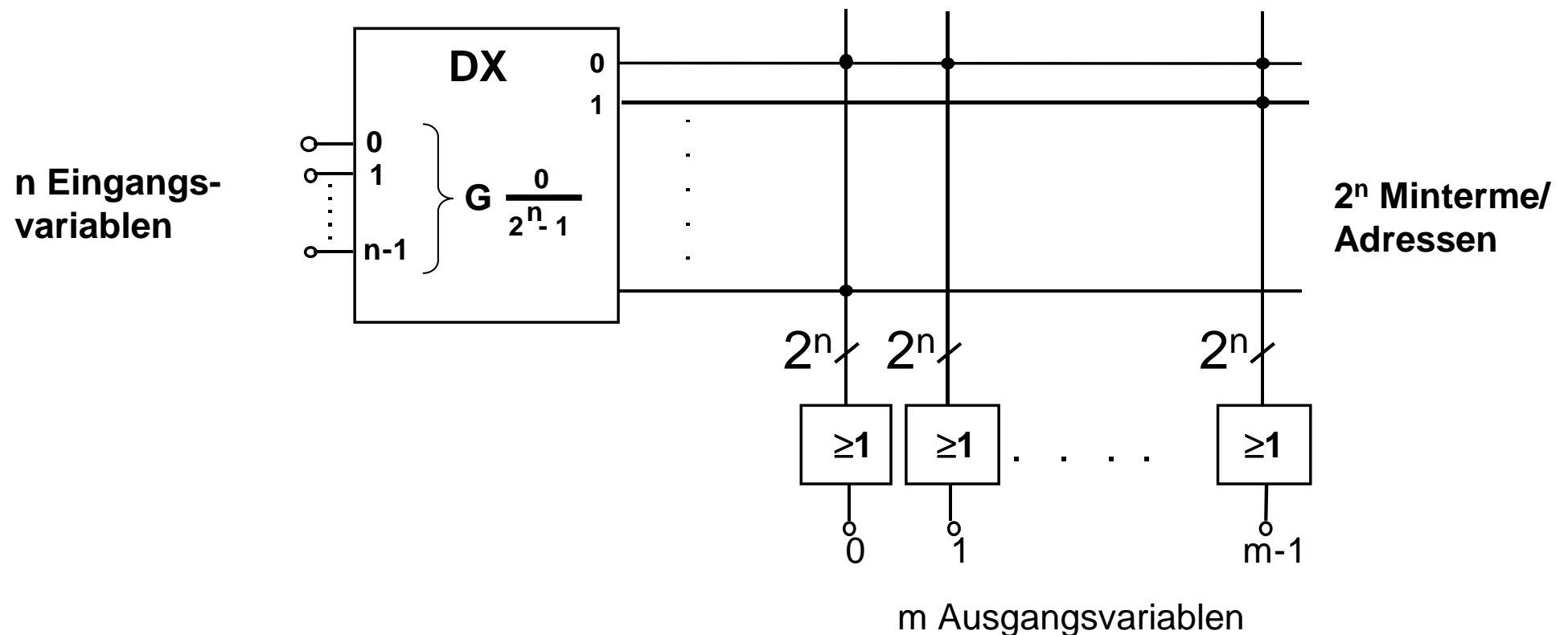
# Realisierung mittels Speicherbausteinen

---

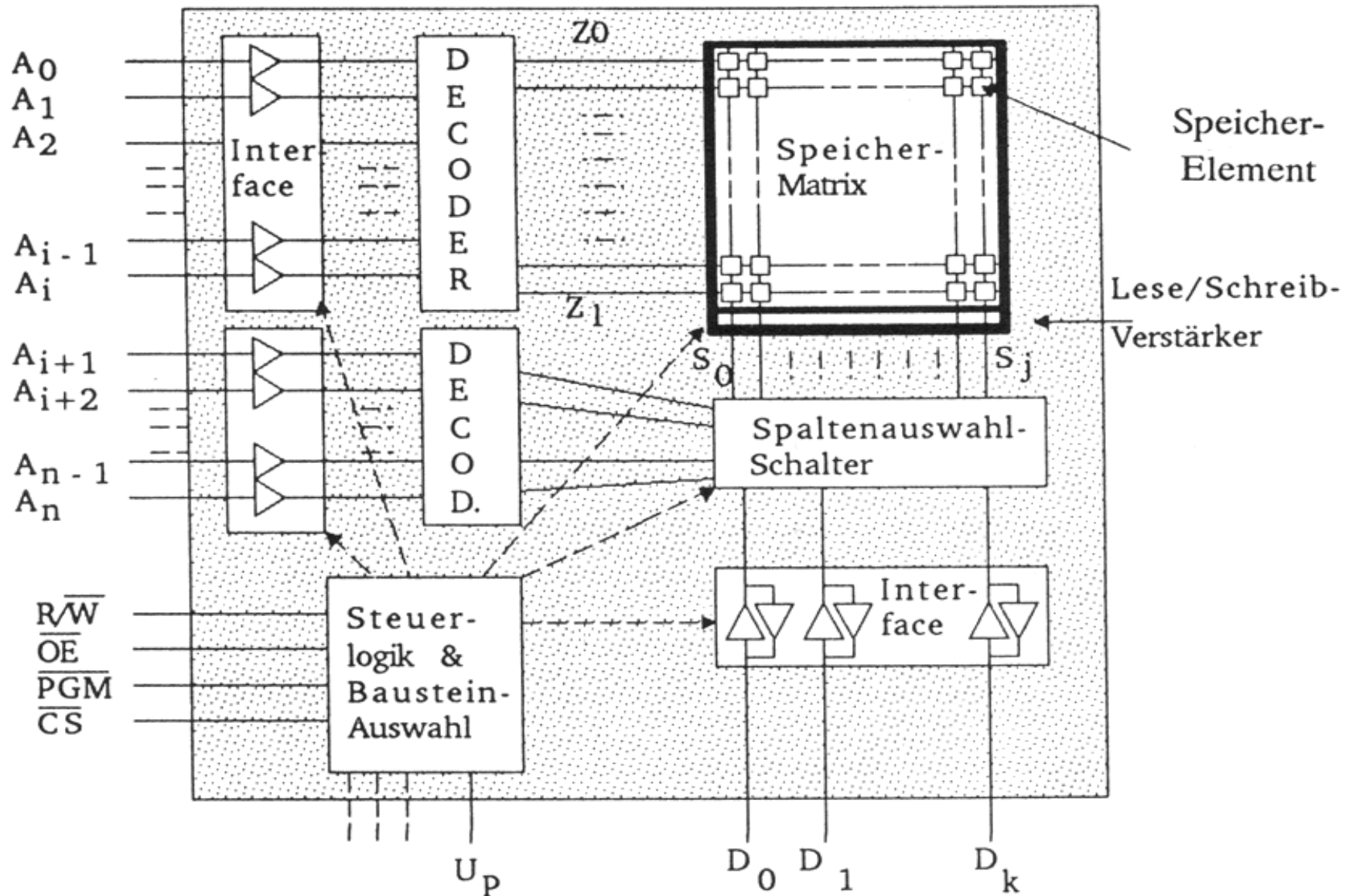
- Bei den bisher behandelten Bausteinen (Gatter, Multiplexer, Dekoder) war die Funktion fest vorgegeben.
  - festverdrahtete Logik
- Höherintegrierte Verknüpfungsbausteine müssen die Flexibilität bieten, an viele verschiedene Anwendungen anpaßbar zu sein. Diese Anpassung wird als **Personalisierung** oder als **Programmierung** bezeichnet.
  - mikroprogrammierte Logik  
(siehe M6 für entsprechende CPUs)

# Schematischer Aufbau eines Speicherbausteins

Speicheranordnung, in der beliebige Funktionstabellen abgelegt werden.



# Organisation von Speicherbausteinen





# Erläuterung zum Speicherbaustein

---

- Durch das Anlegen von Eingangssignalen wird eine Speicherzelle ausgewählt (adressiert) und der dort gespeicherte Funktionswert an den Ausgängen zur Verfügung gestellt.
- Die Leitungen, die den Dekoder verlassen, entsprechen also den Mintermen von  $n$  Eingangsvariablen, also den Zeilen der Funktionstabelle.
- Das Speichern einer 1 für eine bestimmte Ausgangsvariable  $i$  bedeutet, daß dieser Minterm in die ODER-Verknüpfung am  $i$ -ten Ausgang einbezogen wird, eine 0 heißt, daß der Minterm nicht benutzt wird.

# Speichertypen (1)

---

- Je nach Personalisierung des Speicherbausteins unterscheidet man verschiedene Speichertypen.
- ROM (Read Only Memory):
- Speicherbausteine, auf die nur lesend zugegriffen werden kann. Programmierung beim Hersteller (maskenprogrammierbare ROMs), wird während der ganzen Lebenszeit des Bausteins beibehalten.

# Speichertypen (2)

---

- PROM (Programmable Read Only Memory):
- Programmierbare ROMs, die erst vom Benutzer programmiert werden.
- Programmierung:
  - Durchbrennen von mikroskopisch kleinen Verbindungen (engl.: fusible link)
  - Auf den Baustein aufgebrachte Ladungen, die über Jahre hinweg durch physikalische Prozesse festgehalten werden (engl.: stored charge).

# Speichertypen (3)

---

- EPROM (Erasable Programmable Read Only Memory):
  - Ein benutzerprogrammierbarer Speicherbaustein, durch UV-Licht wieder löschtbar und dann neu programmierbar.
  - Quarzfenster auf der Bausteinoberfläche.
  - Es gibt auch Speicherbausteine, die elektrisch (durch das Anlegen höherer Spannungen) gelöscht werden können
    - EEPROM: Electrically Erasable PROM
    - EAROM: Electrically Alternable ROM.
    - Anwendung z.B. Speicher bei Digitalkameras (Compact Flash, Memory Stick)

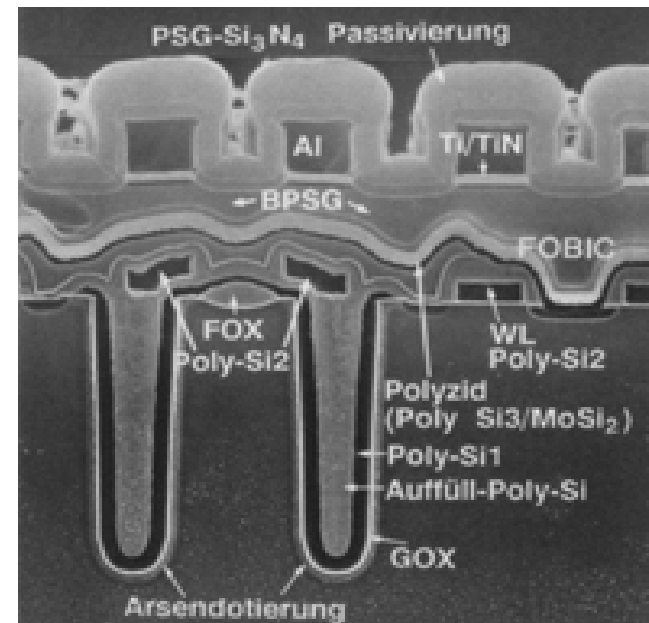
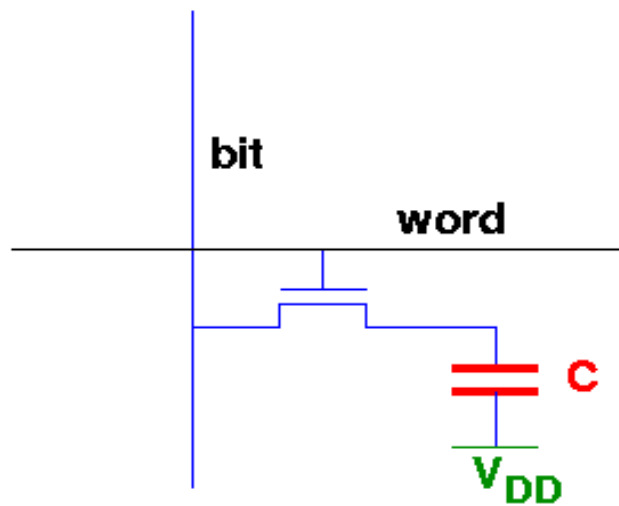
# Speichertypen (4)

---

- ❑ RAM (Random Access Memory):
  - ❑ Speicher, auf die während des Normalbetriebs lesend und schreibend zugegriffen werden kann.
  - ❑ Ein RAM-Baustein verliert seine Programmierung, wenn er von der Spannungsversorgung abgetrennt wird.
  - ❑ Man unterscheidet :
    - Dynamische RAM-Bausteine (DRAM)
    - Statische RAM-Bausteine (SRAM).

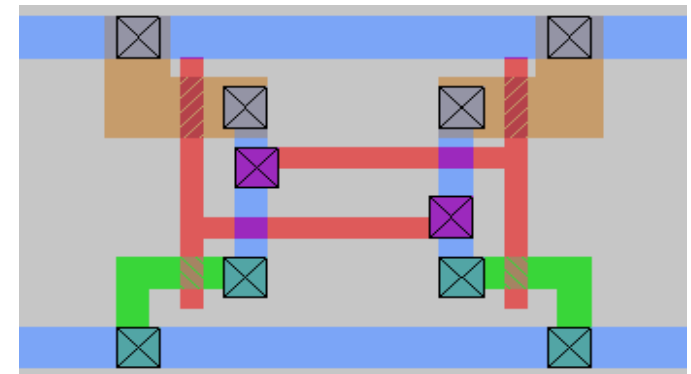
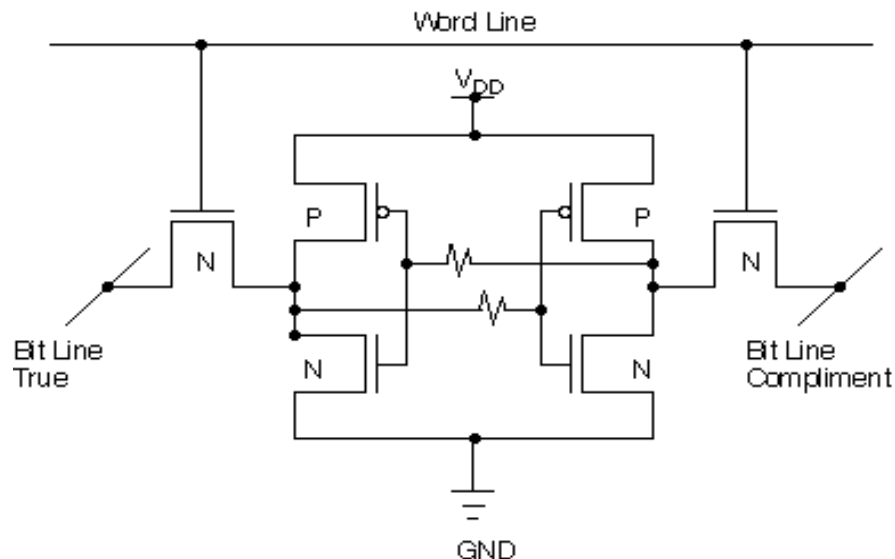
# Dynamische RAM-Bausteine (DRAM)

- Ein Kondensator dient als Ladungsspeicher, und ein Transistor wird zum Ankoppeln an diesen Ladungsspeicher benötigt.
- Der Speicherinhalt muß in regelmäßigen Zeitabständen „aufgefrischt“ werden (memory refresh).



# Statische RAM-Bausteine (SRAM)

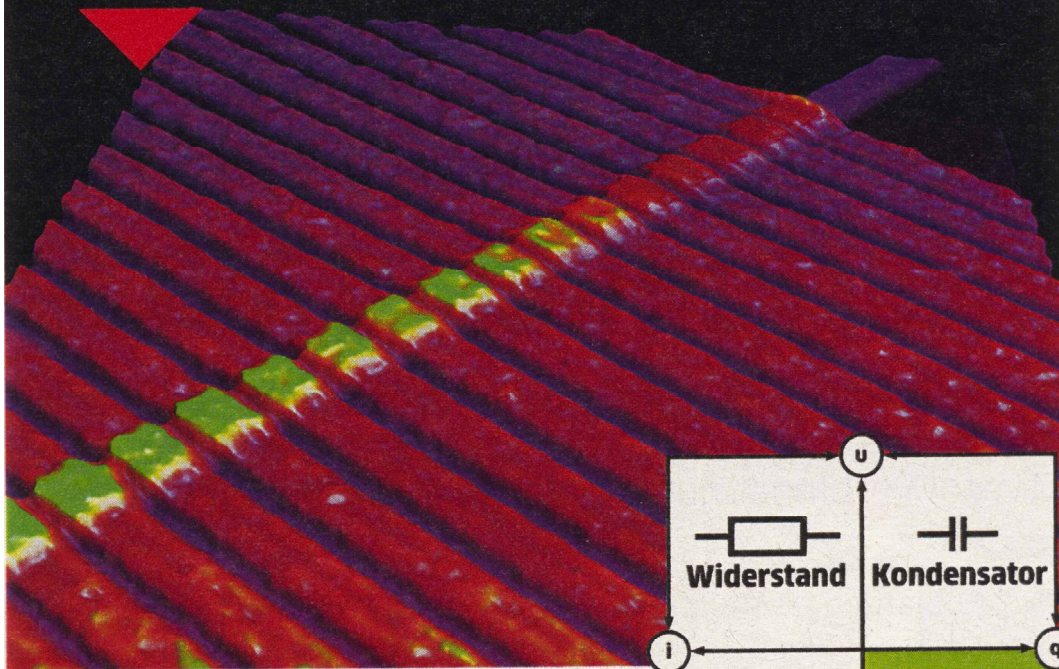
- Als Speicherzelle wird ein Flipflop verwendet (Erklärung folgt in Modul 4).
  - Die Speicherzelle hält ihre Programmierung auch ohne Regeneration.
  - Die Zugriffszeit bei einem statischen RAM ist wesentlich kürzer als bei einem dynamischen RAM.
  - Eine statische Speicherzelle benötigt etwa 6 bis 8 Transistoren, eine dynamische dagegen deutlich weniger (z.B. ein Transistor).



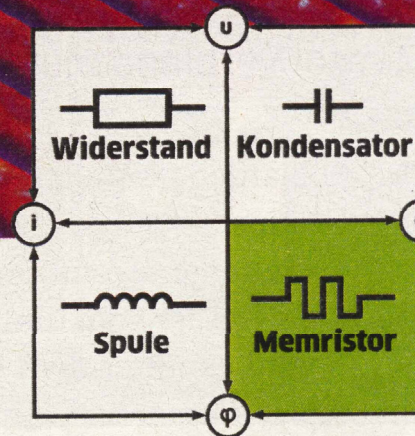


# Memresistor

2011



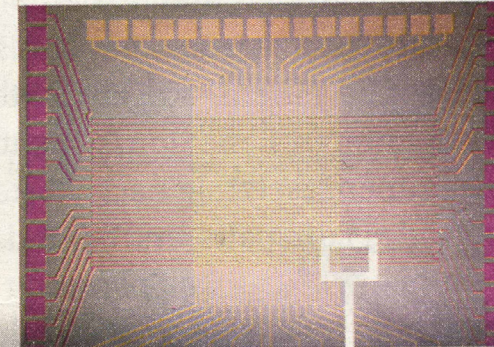
**Neuling** Der Memristor ist das 2008 eingeführte vierte elektronische Bauteil. Er ermöglicht Speicher so schnell wie RAM - die ihren Inhalt aber auch ohne Strom behalten



2011

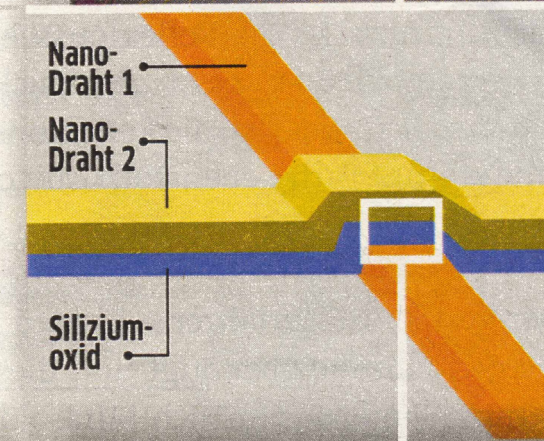
## Einfacher Aufbau

Memristoren sind wesentlich einfacher aufgebaut als etwa Flashspeicher - und so günstiger herzustellen. Sie bestehen im Prinzip aus rechtwinklig gekreuzten Leiterbahnen und einer Speicherschicht aus Silizium



## Materialvielfalt

Die Speicherschichten eines Memristors lassen sich sowohl aus diversen Metallen als auch aus nichtmetallischen Stoffen fertigen. Die Rice Universität entdeckte jüngst, dass auch Siliziumoxid als Speichermaterial taugt





# Gegenüberstellung von RAMs und ROMs

- Unterschiede zwischen RAMs und ROMs betreffen vor allem:
  - Lese-/Schreib-Möglichkeiten (RW; *read-write*)
  - Zugriffszeiten (ZZ; für R/W)
  - Speicherpermanenz ohne Spannungsversorgung (SP)
  - Realisierbare Speichergröße (SG)

	ROM	PROM	EPROM	Flash-PROM	DRAM	SRAM
RW	R	R	R ((W))	R (W)	RW	RW
ZZ	+	+	+ / -	+ / -	++	+++
SP	+	+	+	+	-	-
SG	+	+	+	+	+	+

# PLA (Programmable Logic Array)

---

- ❑ Bisher wurde die gesamte Funktionstabelle in einem Speicherbaustein abgespeichert und die Funktion durch ihre DNF (Disjunktive Normalform) realisiert.
- ❑ Verwendet man stattdessen die DMF (Disjunktive Minimalform), lassen sich Funktionen oft sehr viel kompakter darstellen.
- ❑ PLA (Programmable Logic Array):
  - ❑ Im Unterschied zum ROM werden bei PLA eingangsseitig nicht Minterme, sondern Primimplikanten der Minimalüberdeckung erzeugt.
  - ❑ Dazu wird der Dekoder durch eine UND-Matrix ersetzt.

# FPLA und PAL

---

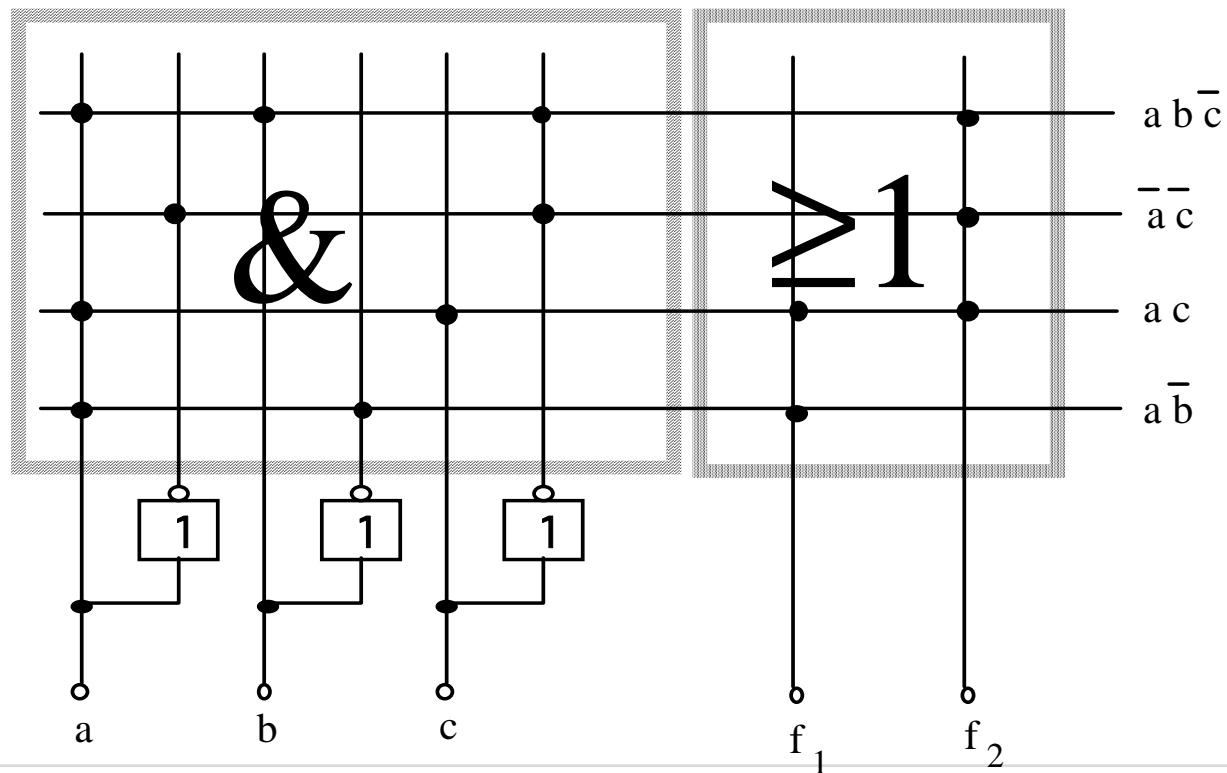
- ❑ PLAs werden ähnlich wie ROMs bereits bei der Herstellung personalisiert.
- ❑ Ein vom Benutzer zu programmierendes PLA mit fest vorgegebener Anzahl von Eingangsvariablen  $n$ , Produkttermen  $k$  und Ausgangsvariablen  $m$  wird **FPLA** (**field programmable logic array**) genannt.
- ❑ Alternativ dazu werden **PAL**-Bausteine (**programmable array logic**) angeboten, bei denen die UND- bzw. ODER-Matrix bereits in der Herstellung personalisiert wurde.

# Beispiel: PAL-Realisierung

Die (schon minimierten) Funktionen

$$f_1 = a \bar{b} \vee a c \quad \text{und} \quad f_2 = \bar{a} \bar{c} \vee a c \vee a b \bar{c}$$

sollen mit einem PAL realisiert werden:



# Modul 3: Schaltnetze

---

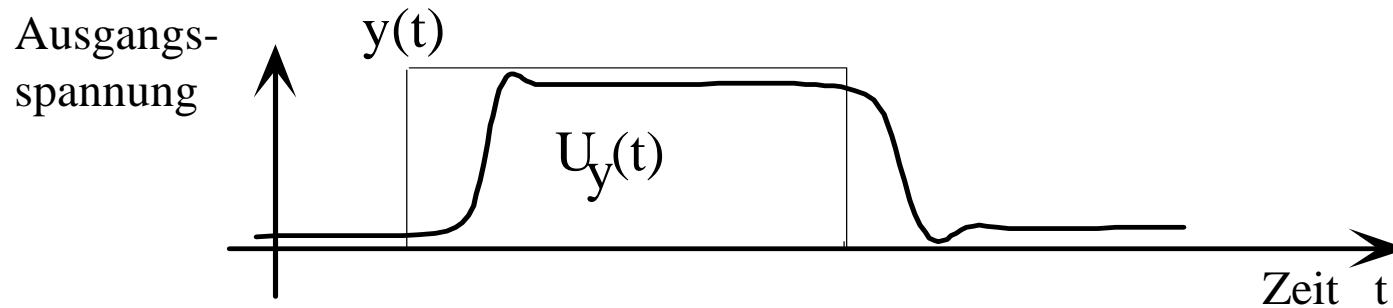
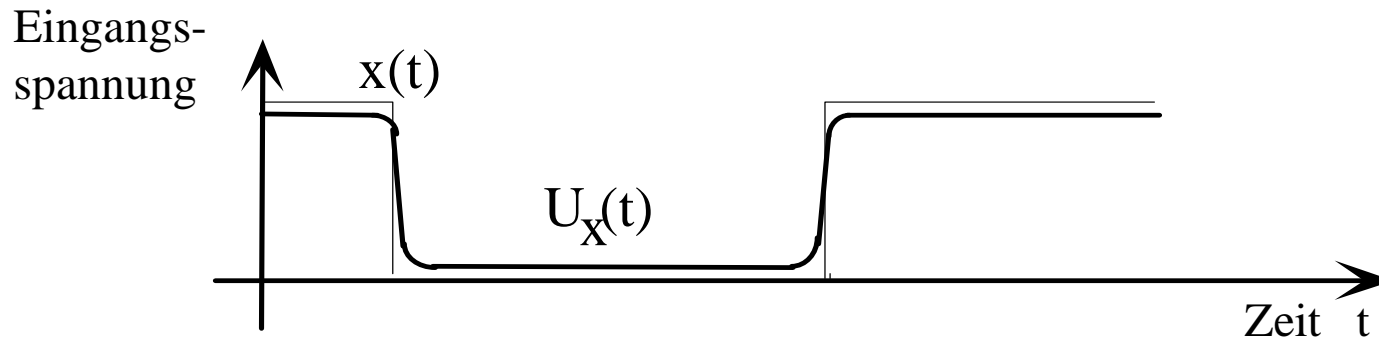
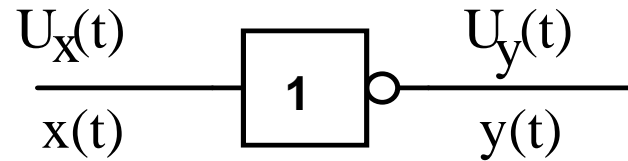
- ❑ Formale Grundlagen logischer Beschreibungen
  - Boolesche Algebra, Schaltalgebra
- ❑ Normal- und Minimalformen
- ❑ Realisierung von Schaltnetzen auf Schalter- und Gatterebene
- ❑ Entwurf von Schaltnetzen
  - Logikminimierung, KV-Diagramme
  - Programmierung von Funktionen
- ❑ Laufzeiteffekte bei Schaltnetzen

# Laufzeiteffekte

---

- ❑ Auf der Gatterebene wurden die Gatter bisher als ideale logische Verknüpfungen betrachtet.
- ❑ In der Realität werden Gatter jedoch z.B. mittels Transistoren, Widerstände, Kapazitäten realisiert (Layoutebene).
- ❑ Der zeitliche Signal-Verlauf eines realen Gatters weicht vom Verlauf der idealen booleschen Größen ab.

# Realer und idealer Signalverlauf (Inverter)



# Realistischere Beschreibung von Gattern

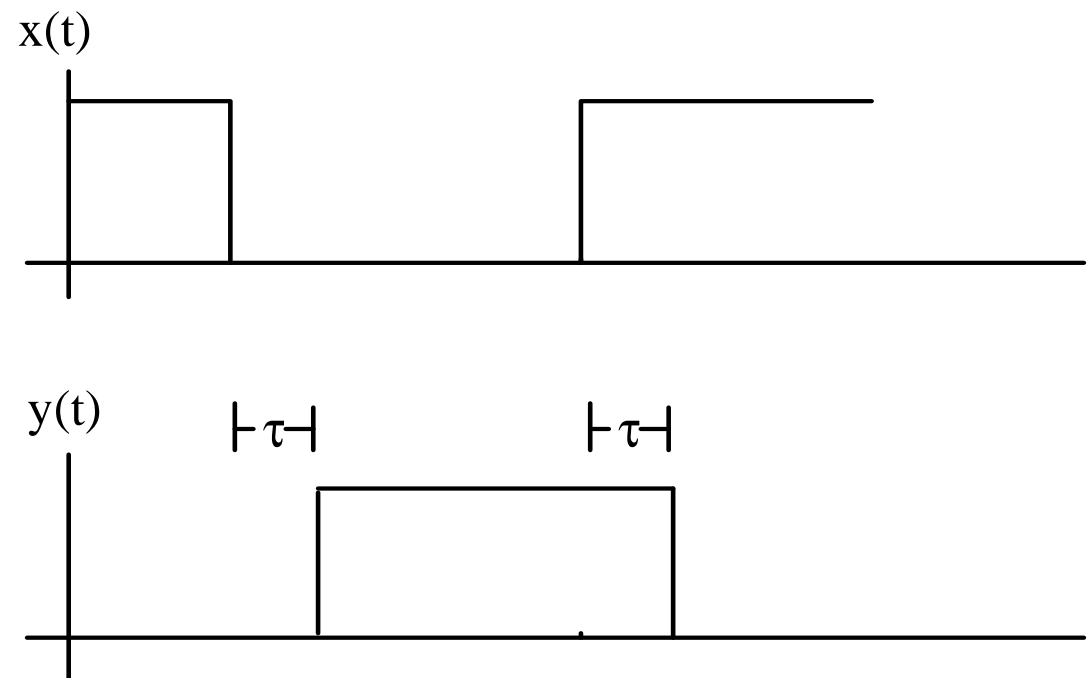
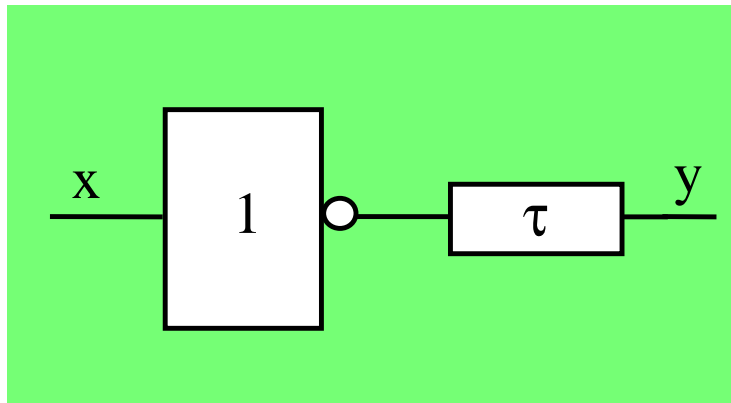
- Um die Effekte auf der Gatterebene annähernd zu beschreiben, gibt es eine Reihe verschieden komplexer Modelle.
- Einfaches Modell: **Totzeitmodell**
  - Es werden lediglich die durch Gatter und Leitungen entstehenden Totzeiten berücksichtigt.
  - Ein **reales Verknüpfungsglied** (Gatter) wird modelliert durch:
    - Ein **ideales Verknüpfungsglied** ohne Verzögerungsanteil und
    - Ein **Totzeitglied** als reines Verzögerungsglied (steht für die Schaltzeit des Gatters und ggf. für Leitungsverzögerungen).
- Das zeitliche Verhalten einer binären Größe hinter einem Totzeitglied ist dasselbe wie dasjenige vor dem Totzeitglied, aber um die Zeit  $\tau$  versetzt:





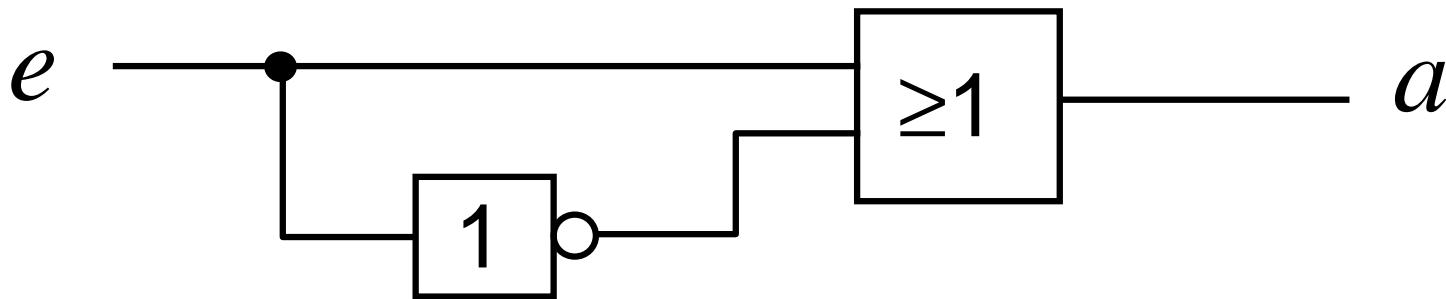
# Beispiel: Totzeitmodell eines Inverters

- Mit Hilfe dieses einfachen Modells lassen sich Laufzeiteffekte bereits sehr gut modellieren (auch wenn dieses Modell noch sehr idealisierend ist!).



# Beispiel: Inverteranwendung

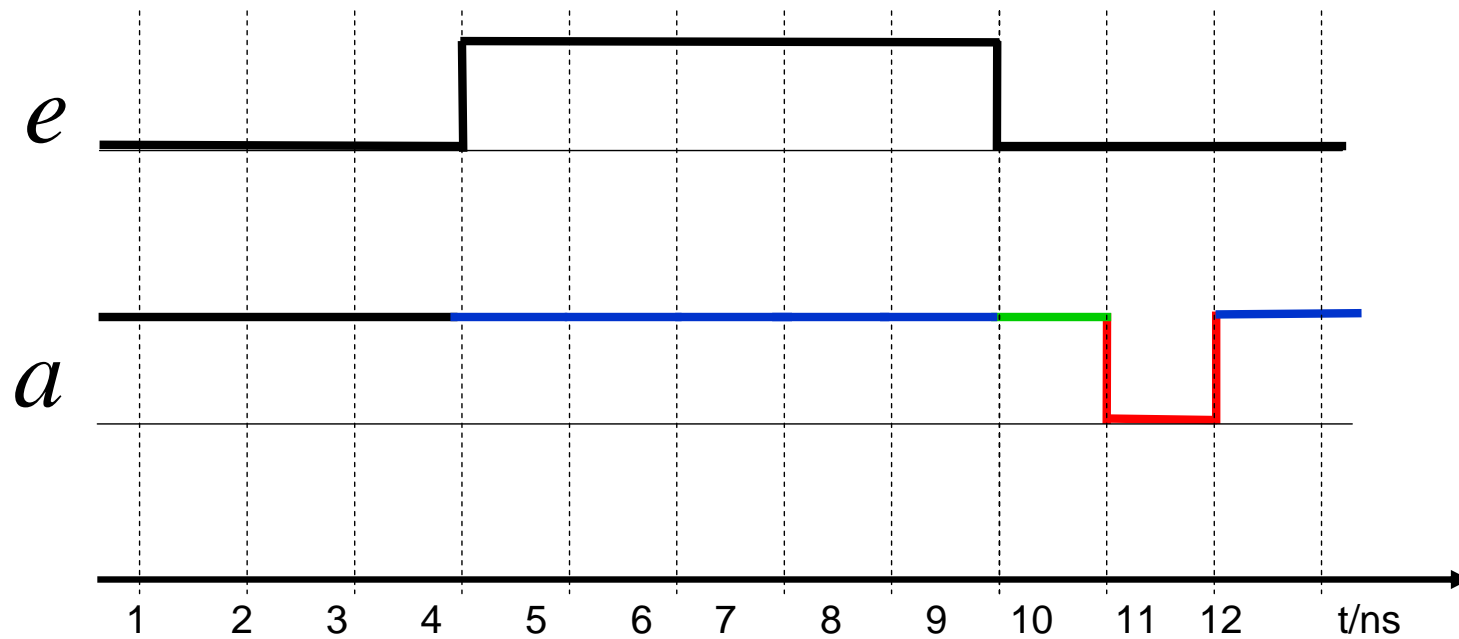
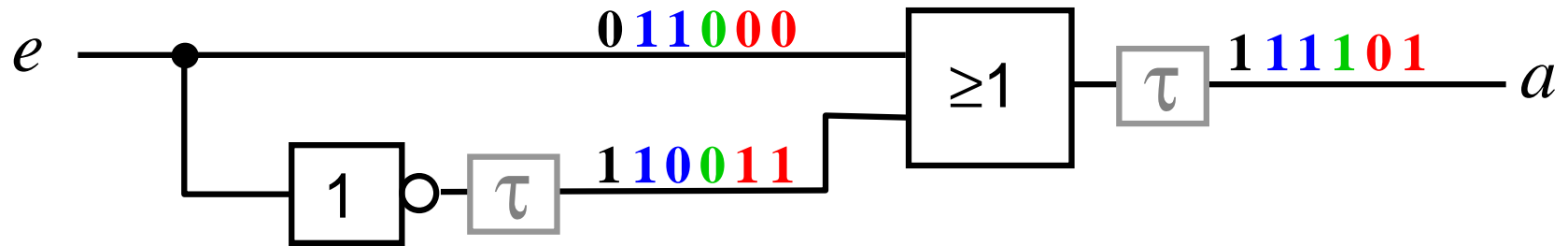
Gegeben:



$$a = e \vee \bar{e} = 1$$

Beide Gatter haben eine Verzögerungszeit von 1 ns.

# Zeit-Diagramm



# Verhalten eines Schaltnetzes bei Änderung der Eingabebelegung (1)

---

- Ideales Schaltnetz:
- Das Ausgangssignal **ändert sich nicht**, wenn alte und neue Belegung denselben logischen Verknüpfungswert liefern.
- Das Ausgangssignal **ändert sich genau einmal**, wenn alte und neue Belegung verschiedene logische Verknüpfungswerte liefern.

# Verhalten eines Schaltnetzes bei Änderung der Eingabebelegung (2)

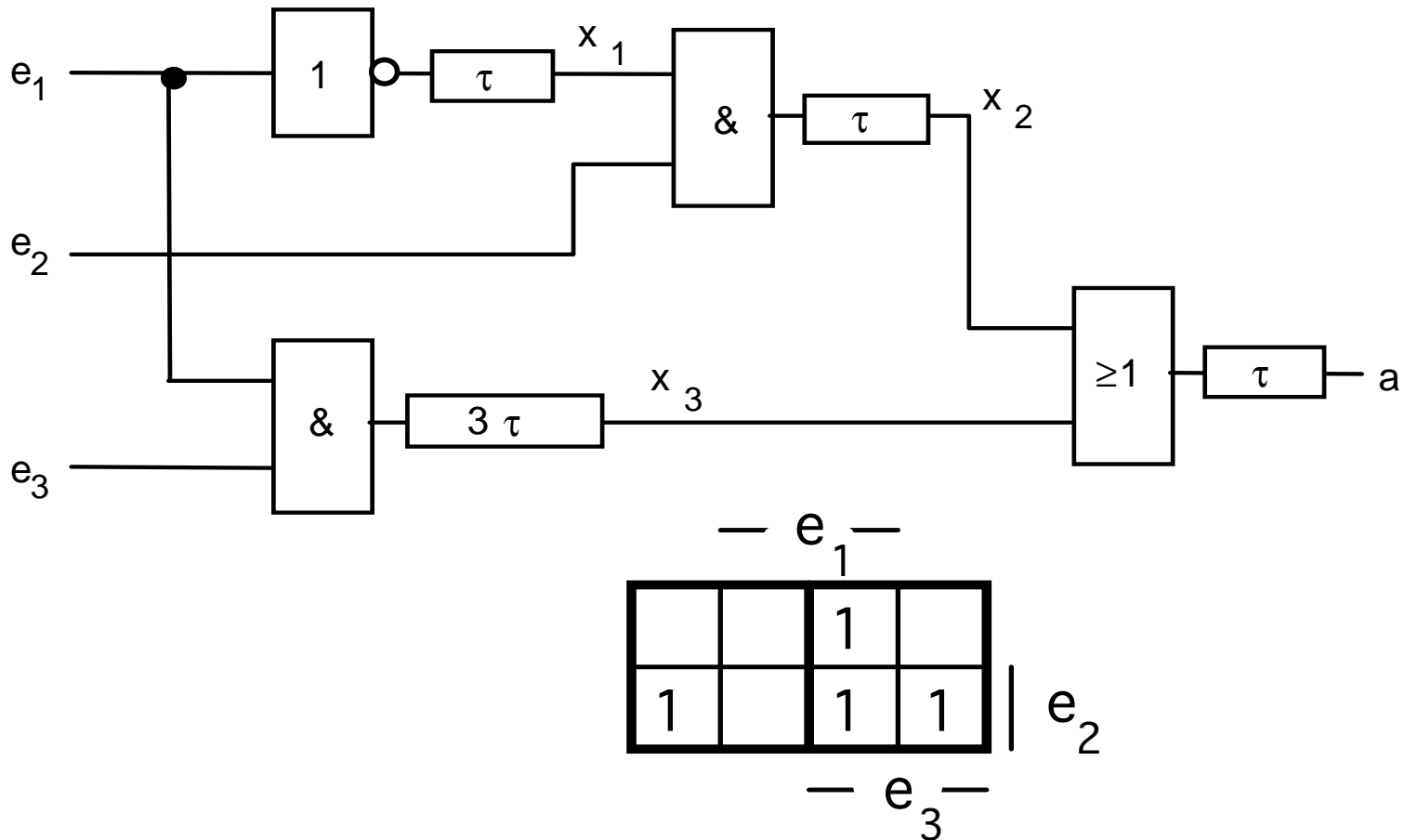
---

- Reales Schaltnetz:
- Die Änderung läuft auf verschiedenen langen Wegen mit verschiedenen Verzögerungen durch das Schaltnetz.
- Mehrfache Änderungen des Ausgangssignals sind möglich, bis sich der stabile Endwert einstellt

→ **Hasardfehler**

# Beispiel

Funktion:  $a = \bar{e}_1 e_2 \vee e_1 e_3$



# Eingabewechsel

---

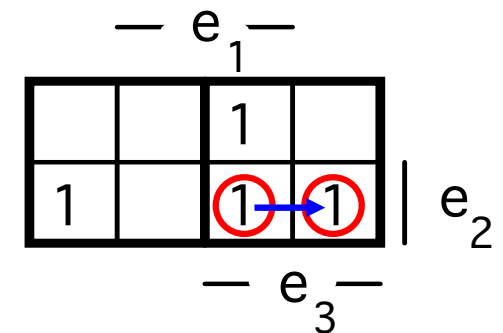
- Es sollen die folgenden Eingabewechsel betrachtet werden:
  - a) Die Eingänge  $e_2$  und  $e_3$  seien konstant 1, der Eingang  $e_1$  wechsele von 0 auf 1
  - b) Die Eingänge  $e_2$  und  $e_3$  seien konstant 1, der Eingang  $e_1$  wechsele von 1 auf 0

# Funktion: $a = \bar{e}_1 e_2 \vee e_1 e_3$

Funktionswerte bei den Übergängen:

$$(e_3, e_2, e_1) = (1, 1, 0) \quad \Rightarrow \quad a = 1$$

$$(e_3, e_2, e_1) = (1, 1, 1) \quad \Rightarrow \quad a = 1$$



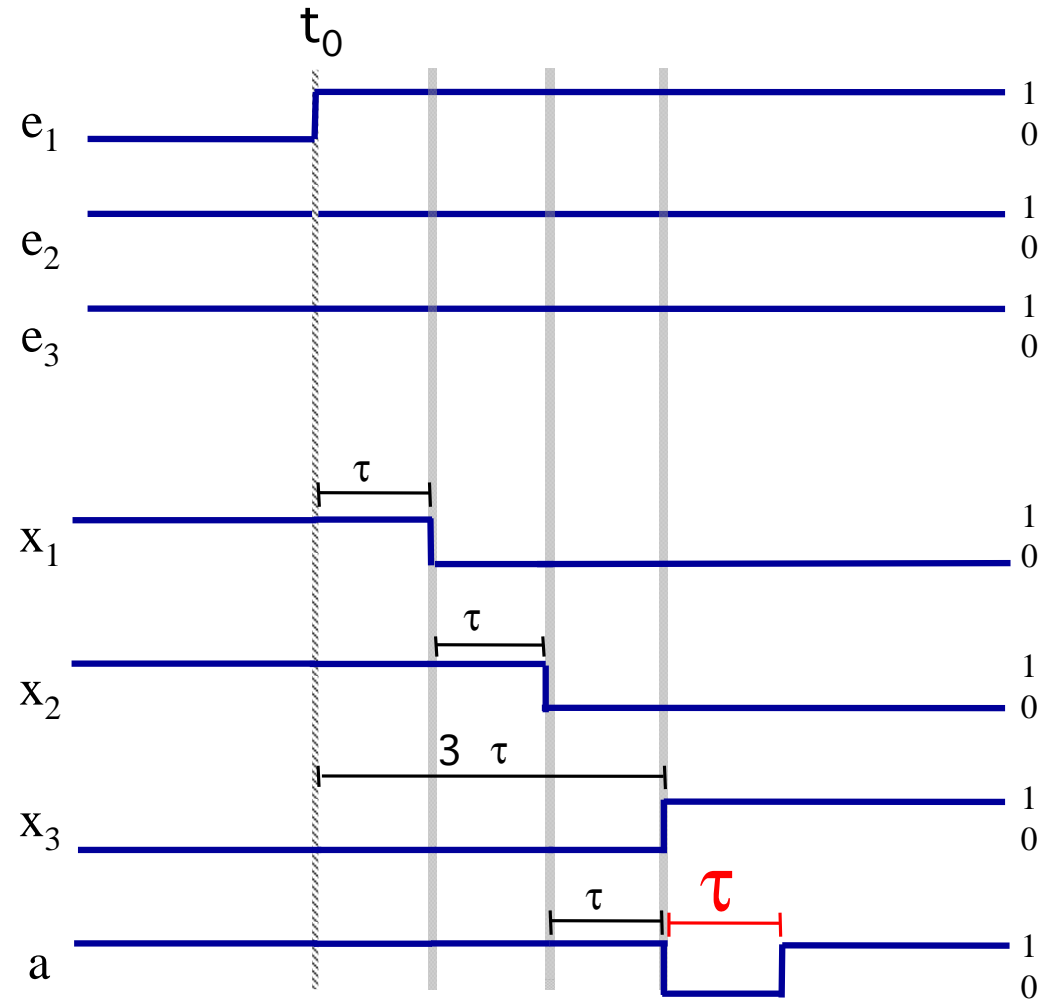
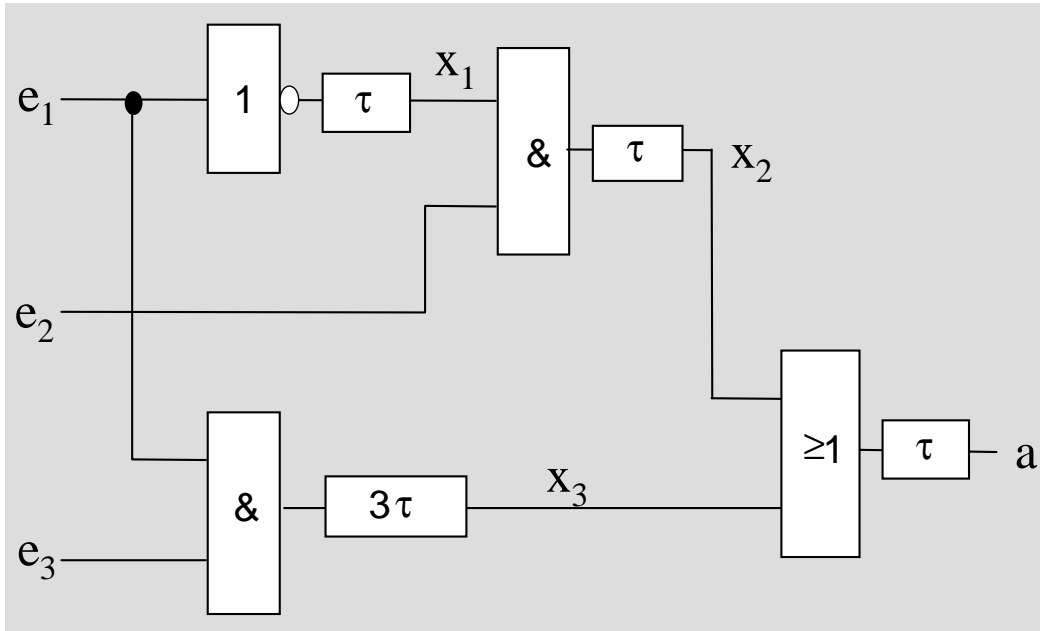
$\Rightarrow$  korrektes Verhalten bei den Übergängen.

Bei beiden Übergängen darf sich der Wert von  $a$  nicht ändern.  
Er muß **konstant 1** bleiben.

**Genau dieses Verhalten kann jedoch nicht garantiert werden !**

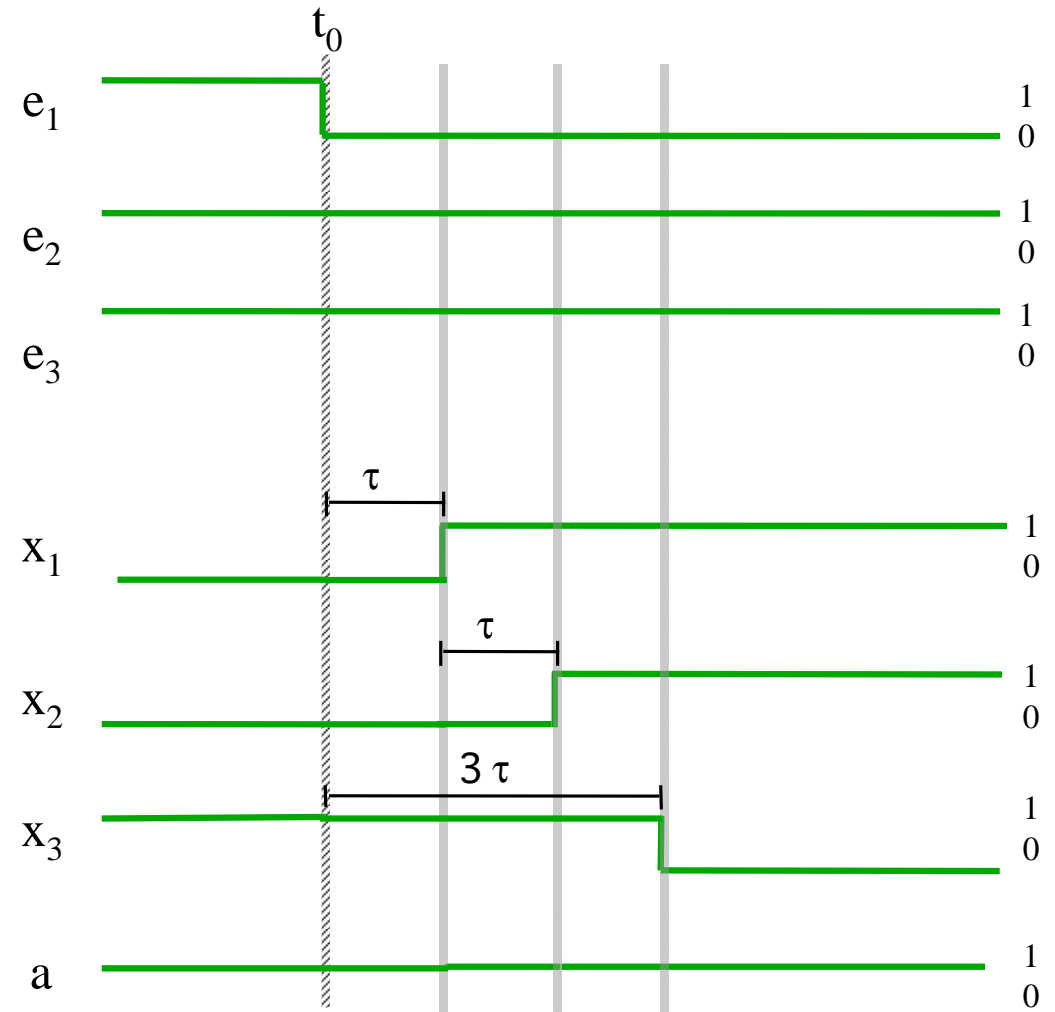
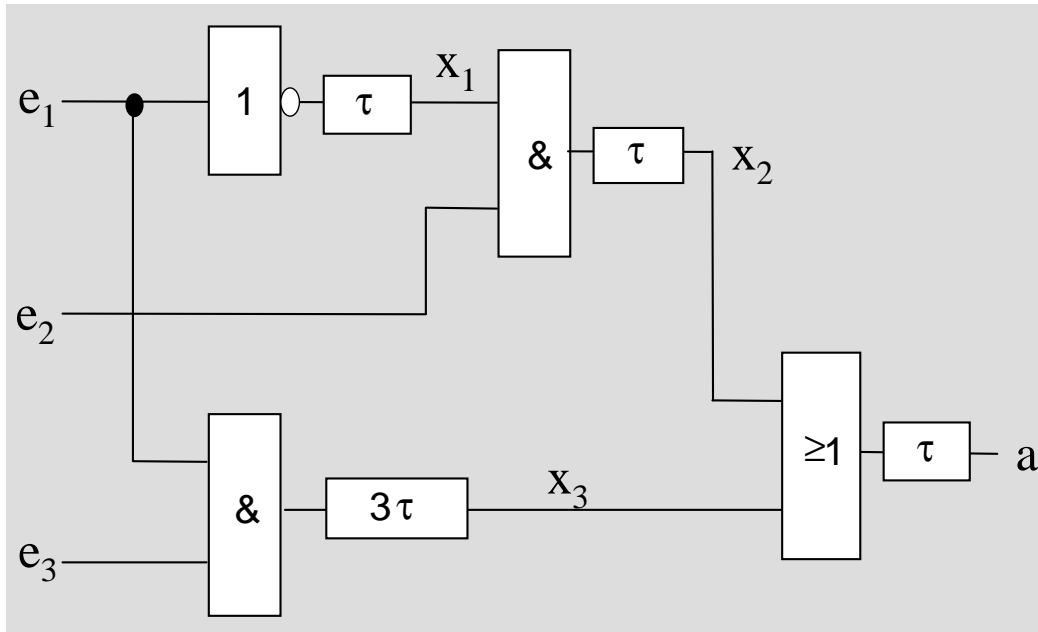


# Das Verhalten anhand des Totzeitmodells



**Hasardfehler**

# Das Verhalten anhand des Totzeitmodells



kein Hasardfehler

# Ergebnis

---

- Beim Wechsel  $e_1$  von 0 auf 1 liefert das Ausgangssignal nicht ständig den korrekten Funktionswert

→ **Hasardfehler**

- Beim Wechsel  $e_1$  von 1 auf 0 ist das Ausgangssignal hingegen korrekt

# Begriffe: Eingabewechsel, Übergang

---

- Definition:

Ein **Eingabewechsel** ist die Änderung einer oder mehrerer Eingangsvariablen zu einem bestimmten Zeitpunkt.

- Falls sich mehrere Eingangsvariablen ändern sollen, so müssen sie dies gleichzeitig tun.

- Definition:

Ein **Übergang** ist der Vorgang im Schaltnetz, der vom Eingabewechsel ausgelöst wird. Er beginnt mit dem Eingabewechsel und endet mit dem Eintreten des neuen Ruhezustandes.

# Begriffe: Hasardfehler - Hasard

---

- Definition:

Ein **Hasardfehler** ist eine mehrmalige Änderung der Ausgangsvariablen während eines Übergangs.

- Definition:

Ein **Hasard** ist die durch das Schaltnetz gegebene logisch-strukturelle Vorbedingung für einen Hasardfehler, ohne Berücksichtigung der konkreten Verzögerungswerte.

# Hasardbehaftete Übergänge (1)

---

- Jeder Hasard ist eine Eigenschaft eines bestimmten Überganges im Schaltnetz.
- Zur Betrachtung, ob ein bestimmter Übergang hasardbehaftet ist oder nicht, interessiert nur:
  - Die logische Funktion, die durch das Schaltnetz realisiert wird.
  - Die Struktur des Schaltnetzes, d.h. die Anzahl, die Verknüpfungsfunktionen und die genaue Anordnung der Gatter zur Realisierung der Funktion, nicht jedoch die tatsächlichen Verzögerungswerte der verwendeten Gatter.

# Hasardbehaftete Übergänge (2)

---

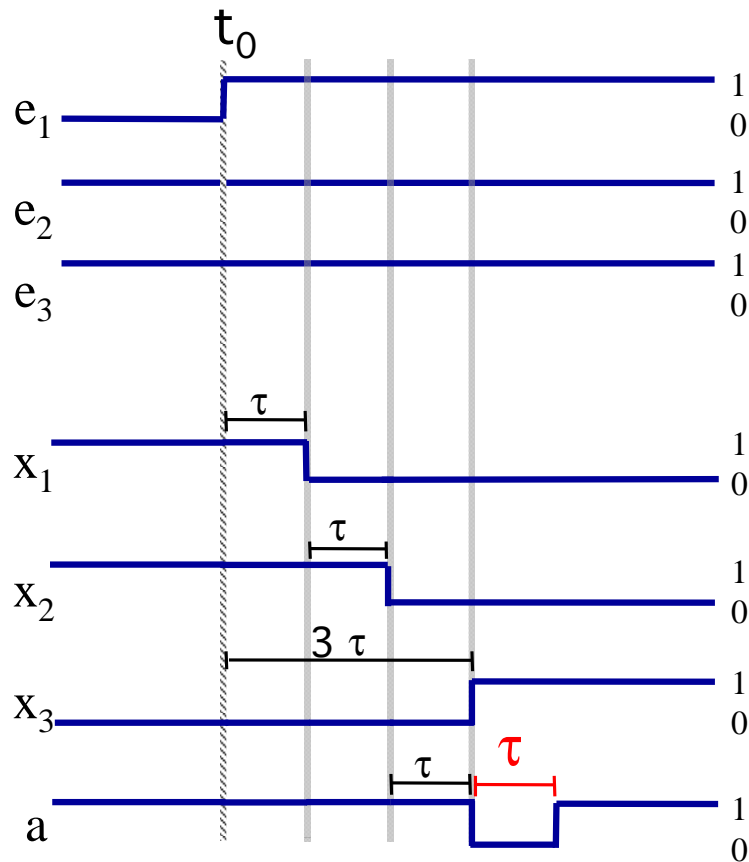
- Tritt in einem konkreten Schaltnetz bei einem bestimmten Übergang ein Hasardfehler auf, so ist dieser Übergang hasardbehaftet, also:

Hasardfehler  $\rightarrow$  Hasard

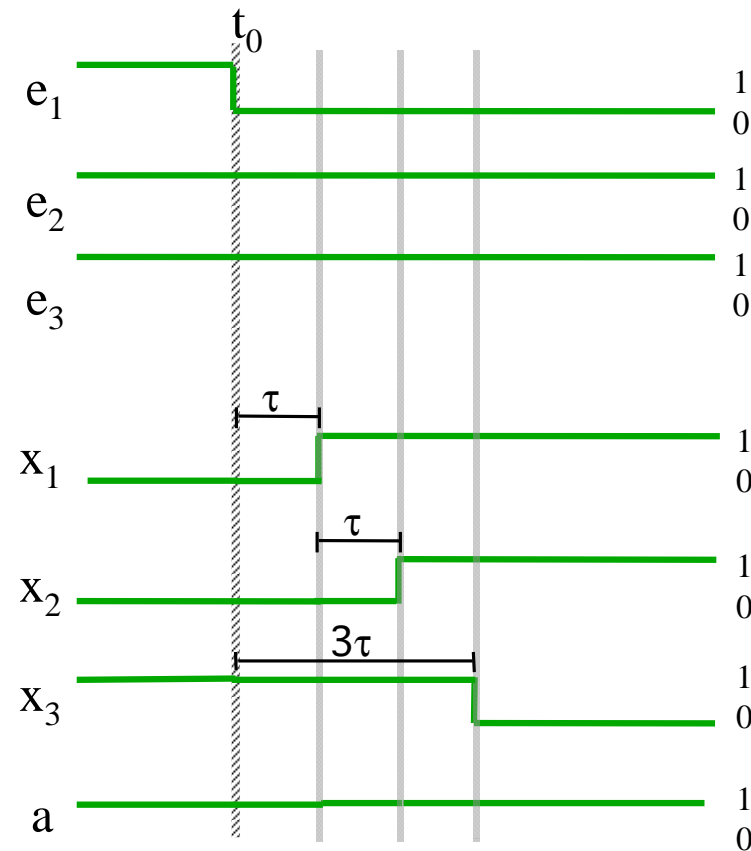
- Die Umkehrung gilt jedoch nicht: Ist ein Übergang hasardbehaftet, so folgt hieraus nicht notwendigerweise das Eintreten eines Hasardfehlers.

Hasard  $\wedge$  ungünstige Verzögerungswerte  $\rightarrow$  Hasardfehler

# Beispiel 1



**Hasardfehler**



**kein Hasardfehler**

Der Übergang  $(e_3, e_2, e_1) : (1, 1, 0) \rightarrow (1, 1, 1)$  ist hasardbehaftet, da es die Möglichkeit zu einem Hasardfehler gibt.



# Funktionshasard

---

- Definition:

Ein **Funktionshasard** ist ein Hasard, dessen Ursache in der zu realisierenden Funktion liegt.

- Er tritt in jedem möglichen Schaltnetz für diese Funktion auf. Er kann nicht behoben werden.

⇒ Für ein konkretes Schaltnetz mit Funktionshasard kann zwar der Funktionshasardfehler durch günstige Wahl der Verzögerungswerte behoben werden, nicht jedoch der Hasard selbst.

# Strukturhasard

---

- Definition:

Ein **Strukturhasard** ist ein Hasard, dessen Ursache in der Struktur des realisierten Schaltnetzes liegt.

- Ein **Strukturhasard** kann deshalb immer durch Änderung der Schaltnetzstruktur bei gleicher Schaltnetzfunktion behoben werden.

⇒ Es ist grundsätzlich möglich, ein anderes Schaltnetz zu entwerfen, welches dieselbe Funktion realisiert und den Strukturhasard beseitigt.

# Statischer 0-Hasard

---

- Analog zu den Übergängen werden die Hasards als statisch bzw. dynamisch bezeichnet, je nachdem, bei welcher Art von Übergang sie auftreten.
- Ein Hasard in einem statischen 0-Übergang heißt **statischer 0-Hasard**.

Beispiele für statische 0-Hasardfehler:



# Statischer 1-Hasard

- Ein Hasard in einem statischen 1-Übergang heißt **statischer 1-Hasard**.
- Beispiele für statische 1-Hasardfehler:

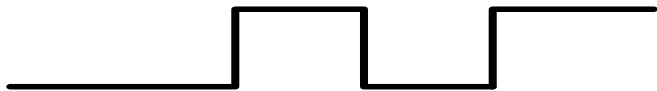


Der Übergang  $(1,1,0) \rightarrow (1,1,1)$  im Beispiel enthält also einen statischen 1-Hasard.

# Dynamischer 01-Hasard

---

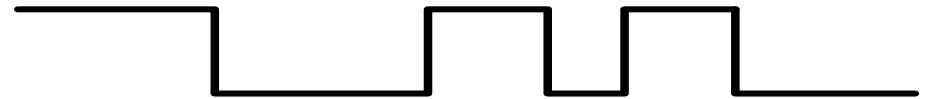
- Ein Hasard in einem dynamischen 01-Übergang heißt **dynamischer 01-Hasard**.
- Beispiele für dynamische 01-Hasardfehler:



# Dynamischer 10-Hasard

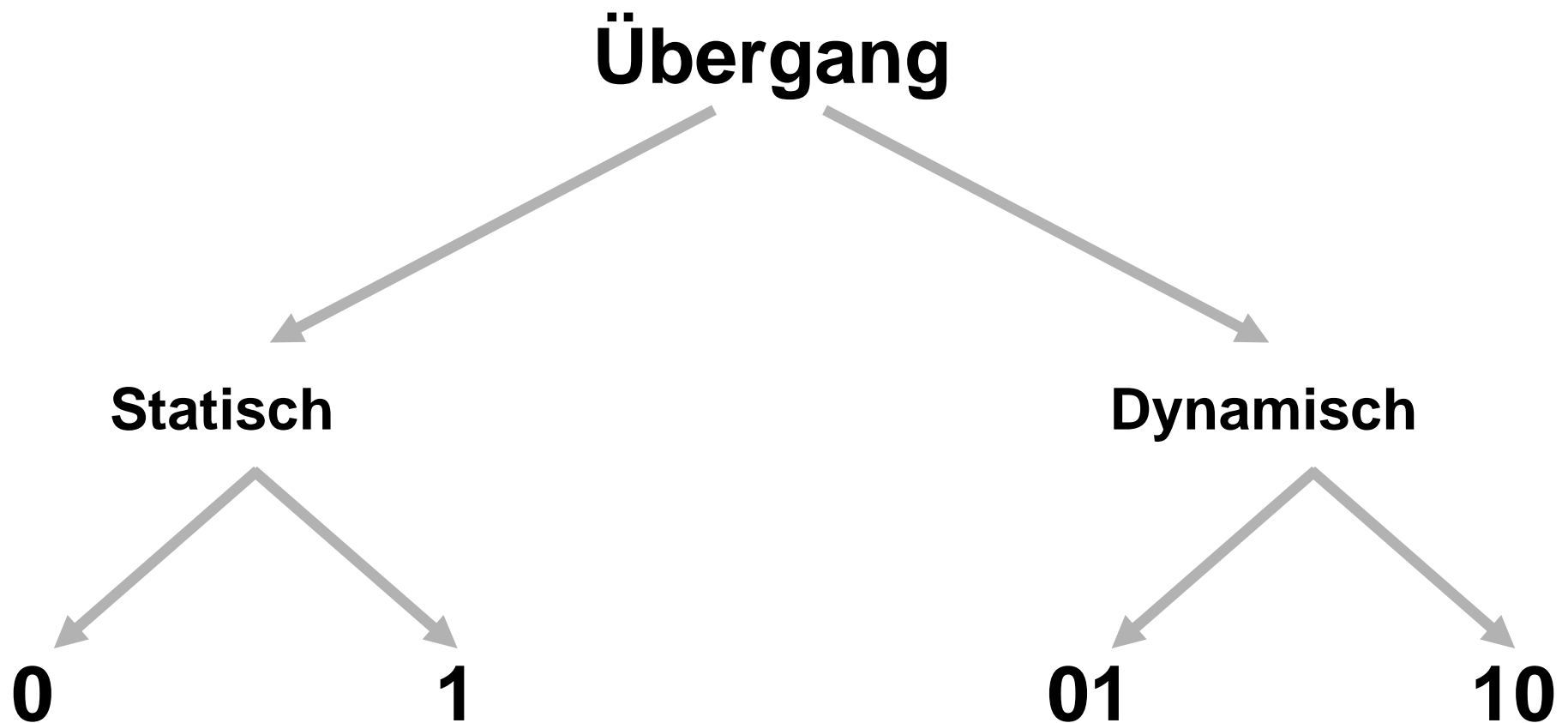
---

- Ein Hasard in einem dynamischen 10-Übergang heißt **dynamischer 10-Hasard**.
- Beispiele für dynamische 10-Hasardfehler:

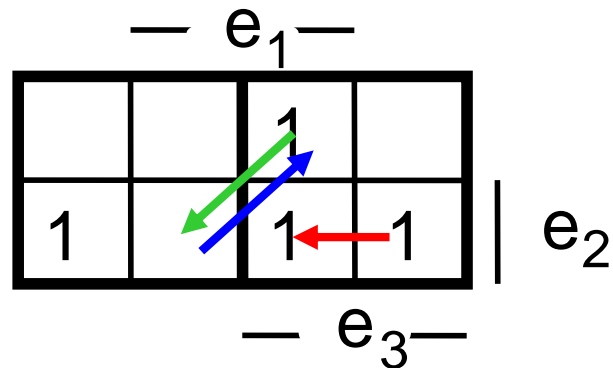


# Klassifizierung von Hasards

---



# Übergangsbeispiele



## Statischer 1-Übergang:

Übergang  $(e_3, e_2, e_1)$ :  $(1, 1, 0) \rightarrow (1, 1, 1)$

## Dynamischer 01-Übergang:

Übergang  $(e_3, e_2, e_1)$ :  $(0, 1, 1) \rightarrow (1, 0, 1)$

## Dynamischer 10-Übergang

Übergang in umgekehrter Richtung:  $(1, 0, 1) \rightarrow (0, 1, 1)$