Digitale Rechenanlagen

MARIÁN VAJTERŠIC

Fachbereich Computerwissenschaften
Universität Salzburg
marian@cosy.sbg.ac.at
Tel.: 8044-6344

28. September 2017

Kapitel 3: Logische Schaltungen

- Bisher:
  - Information
  - Binäre Codierung von Zahlen, um Rechenoperationen mit diesen Zahlen in digitalen Rechenanlagen durchführen zu können.
- Ziel:
  - Zu zeigen: Welche Bauteile man **dazu** braucht und wie diese **strukturiert** sind.
  - Dass die **numerischen** Berechnungen auf einfache **logische** Verknüpfungen zurückgeführt werden können.

Logische Operationen

Logische Operationen sind sehr einfach: Aus **wenigen** lassen sich komplexe Berechnungen erzeugen.
Diese Operationen arbeiten mit **Aussagen**.

Beispiel [Aussage(n)]

Die Aussage

Wenn die Sonne scheint und es windstill ist, fahre ich an den See.

kann in folgende **Aussagen** zerlegt werden:

A: Die Sonne scheint
B: windstill (d. h. kein Wind)
C: an den See fahren

Wahrheitswerte

Die Aussage

A: Die Sonne scheint

kann nur **zwei Wahrheitswerte** annehmen:

W (Wahr)/T (True)	wenn die Sonne scheint
F (Falsch)/F (False)	wenn die Sonne nicht scheint

- Die Aussagen dieses Typs können binär codiert werden (W=1, F=0).
- Wir können **sie** in digitalen Rechenanlagen **bearbeiten**.

Theoretische Basis

Die **theoretische Basis** zur **Aussagenverknüpfung** und deren **Bearbeitung** ist die **mathematische Logik** (formale Logik, Aussagenlogik).

Bemerkung:

- Wenn es **nicht nur <u>zwei</u> Zustände (Wahrheitswerte)** gibt → mehrwertige Logik
- Wenn die **Zustände (Wahrheitswerte) nicht genau definiert** sind → Fuzzy Logic (ungenaue Logik)

Aussagenlogik

Wahrheitswerttabelle

A: Die Sonne scheint
B: windstill
C: an den See fahren

Betrachtet man **alle Möglichkeiten** für das Beispiel, können wir eine **Wahrheitswerttabelle aufstellen**:

A	B	C	
F	F	F	
F	W	F	Sonne scheint **nicht**, windstill: **kein Seeausflug**
W	F	F	
W	W	W	Sonne scheint, windstill: **wir fahren** an den See

Diese Wahrheitswerttabelle entspricht der Konjunktion $A \& B = C$.

Aussagenlogische Operationen

- **Negation** $\neg$ (nicht, non, not): Die Negation kehrt den Wahrheitswert einer Aussage um:

A	$\neg A$
W	F
F	W

Im Beispiel: Ist die Aussage B **wahr** (also windstill, d. h. kein Wind), dann ist $\neg B$ **falsch** (d. h. Wind).

Aussagenlogische Operationen

- **Konjunktion** & $\wedge$ (logisches und/and): Konjunktion von zwei Aussagen A, B wird **nur dann wahr**, wenn A **und** B **wahr sind**:

A	B	$A\&B$
F	F	F
F	W	F
W	F	F
W	W	W

Diese Tabelle entspricht der Formalisierung unseres *See-Ausflug*-Beispiels:
$C = A\&B$, wobei A: Sonne scheint, B: windstill, C: an den See fahren.

Aussagenlogische Operationen

- **Disjunktion** $\vee$ (Adjunktion, logisches oder, or): Es genügt, dass A **oder** B wahr werden, damit die Disjunktion von A, B wahr wird. Die Wahrheitstabelle für die Disjunktion:

A	B	$A \vee B$
F	F	F
F	W	W
W	F	W
W	W	W

(gesprochen A oder B)

Sprachlich – Mathematisch

Auch bei diesen elementaren Operationen ist zu beachten, dass **die sprachliche Verwendung <u>nicht unbedingt</u> mit der mathematischen Bedeutung <u>übereinstimmen</u>** muss.

Z. B.: logisches oder

In der Sprache benutzen wir das Wort *oder* nicht nur inklusiv, das bedeutet, dass der Fall $A = W$, $B = W$, $A \vee B = W$ (die Wahrheitstabelle der Disjunktion) gilt, sondern auch exklusiv (die Wahrheitstabelle von **exclusive or**):

Aussagenlogische Operationen

- **XOR** $\oplus$ (Antivalenz, exklusives oder, exclusive OR, XOR):
  Hier gilt **entweder oder**, also im Fall $A = W$, $B = W$ ist $A \oplus B = F$.

A	B	$A \oplus B$
F	F	F
F	W	W
W	F	W
W	W	F

Beispiel [Sprache]

$$\text{Ich kaufe Käse oder Wurst} \begin{cases} \text{inklusive oder (mathematische Bedeutung).} \\ \text{exklusive oder.} \end{cases}$$

Inklusiv: Wahr, auch wenn ich beide Artikel kaufe.
Exklusiv: Mathematisch wahr, wenn ich **entweder** Käse **oder** Wurst kaufe (**d. h. wenn ich nicht beides kaufe**). Daher ist der Fall $A = W$**,** $B = W$ **falsch** (A: Ich kaufe Käse, B: Ich kaufe Wurst).

Aussagenlogische Operationen

- **Subjunktion** $\rightarrow$ (in der Literatur oft als Implikation $\Rightarrow$ bezeichnet):

A	B	$A \rightarrow B$
F	F	W
F	W	W
W	F	F
W	W	W

(wenn A dann B, aus A folgt B)

Wenn A falsch ist, ist die Subjunktion immer wahr (entspricht der Tatsache, dass man mit einer falschen Voraussetzung alles behaupten kann).

Für den Fall, dass B wahr ist, ist die Subjunktion auch immer wahr (wenn die Folgerung wahr ist, ist die Voraussetzung unerheblich).

Aussagenlogische Operationen

- **Bijunktion** $\leftrightarrow$:

  $A \leftrightarrow B$ (aus A folgt B und umgekehrt)
  (A genau dann wenn B)
  (A dann und nur dann, wenn B)

A	B	$A \leftrightarrow B$
F	F	W
F	W	F
W	F	F
W	W	W

  Bijunktion ist wahr, wenn $A = B = W$ oder $A = B = F$.

  Bemerkung:
  Bijunktion wird oft in der Literatur als Äquivalenz bezeichnet (mit Zeichen $\Leftrightarrow$ oder $\equiv$). Äquivalenz ist ein Spezialfall der Bijunktion. (Äquivalenz ist eine tautologische Bijunktion.) (Die Definition von Äquivalenz folgt auf Folie 247.)

Verknüpfungen

Wir können zwei **Aussagenvariablen** A und B (A, B repräsentieren Aussagen) auf verschiedene Arten verknüpfen und jede dieser **Verknüpfungen** mit einem Junktor (Verknüpfungszeichen) darstellen.

Bereits bekannte Junktoren sind die Zeichen $\neg$, &, $\vee$, $\oplus$ und $\rightarrow$.

Verknüpfungen

Anzahl der Verknüpfungen, also aller möglichen Junktoren aller möglichen Spalten in der Wahrheitstabelle, **bei** n **Variablen**: 2^{2^n}

- Die Wahrheitstabelle hat 2^n Zeilen.
- Diese 2^n Zeilen können mit allen möglichen Kombinationen aus 1 und 0 belegt werden $\rightarrow 2^{(2^n)}$ Möglichkeiten.

Verknüpfungen

$n = 2$, $2^{(2^n)} = 2^4 = 16$ Möglichkeiten (V1-V6)

A	B	V1	V2	V3	V4	V5
0	0	0	1	0	0	0
0	1	0	0	1	0	0
1	0	0	0	0	1	0
1	1	0	0	0	0	1
		0-mal 1	1-mal 1			

V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
1	0	0	1	0	1	0	1	1	1	1
1	1	0	0	1	0	1	0	1	1	1
0	1	1	1	0	0	1	1	0	1	1
0	0	1	0	1	1	1	1	1	0	1
2-mal 1						3-mal 1				4-mal 1

Aussageform

Definition

[Aussageform] Eine Verknüpfung von Aussagevariablen mit **ein** oder **mehreren** Junktoren nennt man **Aussageform**.

Eine Aussageform definiert eine aussagenlogische **Wahrheitsfunktion**.

Den **Wert** einer Wahrheitsfunktion (Aussageform) **berechnet** man mit einer **Wahrheitstabelle**.

Aussageform

Beispiel

Berechnung der Wahrheitsfunktion für die Aussageform $\mathcal{A}$: $(\neg A \& B) \vee (A \& \neg B)$
(**2** Aussagevariablen: A, B; **3** Junktoren: $\neg$, &, $\vee$).

Den Wert der Aussageform können wir mit folgender Wahrheitstabelle ausrechnen:

A	B	$\neg A \& B$	$A \& \neg B$	$(\neg A \& B) \vee (A \& \neg B)$
F	F	F	F	F
F	W	W	F	W
W	F	F	W	W
W	W	F	F	F

Wir sehen, dass diese Aussageform der **XOR-Operation** entspricht.
(XOR: **äquivalent** zu dieser Aussageform und **einfacher**.)

Aussagenlogische Grundbegriffe

Einige wichtige Definitionen der Aussagenlogik.

Definition

[Tautologie] Eine Aussageform $\mathcal{A}$ heißt **Tautologie**, wenn sie für jede beliebige Bewertung ihrer Variablen immer wahr ist.
(Man sagt dann: $\mathcal{A}$ ist allgemeingültig.)

Beispiel [Tautologie]

$\mathcal{A} : A \rightarrow A$

A	A	$A \rightarrow A$
F	F	W
W	W	W

Hingegen $\mathcal{A} : A \& A$ ist **nicht** allgemeingültig:

A	A	$A \& A$
F	F	**F**
W	W	W

Aussagenlogische Grundbegriffe

Definition

[Kontradiktion] Eine Aussageform $\mathcal{A}$ heißt **Kontradiktion**, wenn sie für jede beliebige Bewertung ihrer Variablen immer falsch ist.
(Man sagt dann: $\mathcal{A}$ ist ein Widerspruch.)

Beispiel [Kontradiktion]

$\mathcal{A} : A \& \neg A$

A	$\neg A$	$A \& \neg A$
F	W	F
W	F	F

Aussagenlogische Grundbegriffe

Definition

[Implikation $\Rightarrow$ (tautologische Subjunktion)**]** Eine Aussageform $\mathcal{A}$ impliziert die Aussageform $\mathcal{B}$ genau dann, wenn eine Variablenbewertung, die $\mathcal{A}$ **wahr** macht, auch $\mathcal{B}$ **wahr** macht.

Beispiel [Implikation]

$A\&B \Rightarrow A$

(hier $A\&B = \mathcal{A}$, $A = \mathcal{B}$)

A	B	$A\&B$
W	W	W
W	F	F
F	W	F
F	F	F

Aussagenlogische Grundbegriffe

Überprüfung einer Implikation:
Betrachten **Zeilen** der Wahrheitswerttabelle, **wo** $\mathcal{A}$ **wahr** ist. Wenn Implikation, dann **muss in diesen Zeilen auch** $\mathcal{B}$ **wahr** sein.

Im Beispiel:
$\mathcal{A} = A\&B$ ist **wahr** für $A = B = W$ (**erste Zeile** der Tabelle), in der **ersten Zeile** ist **auch** $\mathcal{B} = A$ **wahr**, also $\underline{\mathcal{A}} = A\&B$ **impliziert** $\underline{\mathcal{B}} = A$.

Aussagenlogische Grundbegriffe

Bemerkung
Implikation wird auch **tautologische Subjunktion** genannt.

Die Begründung:
A impliziert B ($A \Rightarrow B$) dann und nur dann, wenn $A \rightarrow B$ eine Tautologie ist.

Zu zeigen:
(i) Wenn $A \Rightarrow B$: $A \rightarrow B$ ist Tautologie
(ii) Wenn $A \rightarrow B$ ist Tautologie: $A \Rightarrow B$

Aussagenlogische Grundbegriffe

Beweis (ii):
Angenommen $A \to B$ ist eine Tautologie, d. h. für beliebige Bewertung ihrer Variablen ist sie immer wahr, d. h.:

A	B	$C = A \to B$
W	W	W
~~W~~	~~F~~	~~F~~
F	F	W
F	F	W

Weil $A \to B$ eine Tautologie ist, tritt der Fall $A = W$, $B = F$ (Zeile 2) nicht auf.
Bezeichnung: $\mathcal{A} = A$, $\mathcal{B} = B$

$\mathcal{A}$ wahr: Zeile 1. In dieser Zeile ist auch $\mathcal{B}$ wahr,
also $\mathcal{A} = A$ impliziert $\mathcal{B} = B$, d. h. $A \Rightarrow B$. $\square$

Beweis (i): analog (andere Richtung)

Aussagenlogische Grundbegriffe

Weshalb genügt es bei Überprüfung von $\mathcal{A} \Rightarrow \mathcal{B}$, **nur** die Zeilen, **wo** $\mathcal{A}$ **wahr** ist zu betrachten (ob dort auch $\mathcal{B}$ wahr ist)?
Implikation $\Rightarrow$: tautologische Subjunktion

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A} \rightarrow \mathcal{B}$	$\mathcal{A} \Rightarrow \mathcal{B}$
F	F	W	W
F	W	W	W
W	F	F	—
W	W	W	W

Aussagenlogische Grundbegriffe

1. Wenn $\mathcal{A} = F$ (Zeilen 1, 2) dann ist $\mathcal{A} \rightarrow \mathcal{B}$ Tautologie (tautologische Subjunktion).
   Also: Die Zeilen wo $\mathcal{A} = F$ **muss man nicht** überprüfen.
2. Wenn $\mathcal{A} = W$ (Zeilen 3, 4), dann ist $\mathcal{A} \rightarrow \mathcal{B}$ **Tautologie** (tautologische Subjunktion (Implikation genannt)) **nur** für $\mathcal{B} = W$.
   Also bei der Implikation muss in den Zeilen **wo** $\mathcal{A} = W$ **auch** $\mathcal{B} = W$ sein.
   (Die Zeile 3 ist für die Implikation unzulässig, weil Verstoß gegen Tautologie – die Subjunktion ist in diesem Fall keine tautologische Subjunktion.)

Also: Die Zeilen der Wahrheitstabelle wo $\mathcal{A} = W$ **muss man** (auf die Bedingung $\mathcal{B} = W$) überprüfen.
Aus 1. und 2.: Für die Überprüfung von $\mathcal{A} \Rightarrow \mathcal{B}$ genügt es, **nur** die Zeilen der Wahrheitstabelle mit $\mathcal{A} = W$ zu betrachten.

Aussagenlogische Grundbegriffe

Bemerkung
Die Implikation ist ein Spezialfall der Subjunktion (also nicht jede Subjunktion ist gleichzeitig auch eine Implikation).

Beispiel [Subjunktion aber keine Implikation]

$(A \vee B) \rightarrow (A \wedge B)$

A	B	$A \vee B$	$A \wedge B$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

$\underline{\mathcal{A} = A \vee B}$ wahr (1) : 2., 3., 4. Zeile
$\underline{\mathcal{B} = A \wedge B}$ wahr (1) : 4. Zeile

Also ist diese Subjunktion **keine** Implikation, weil die Variablenbewertungen $A = 0$, $B = 1$ (Zeile 2) und $A = 1$, $B = 0$ (Zeile 3) (**die $\mathcal{A}$ wahr machen**) **nicht** $\mathcal{B}$ wahr machen.

Aussagenlogische Grundbegriffe

Beispiel [Implikation]

$B \Rightarrow A \rightarrow B$

A	B	$A \rightarrow B$
F	F	W
F	W	W
W	F	F
W	W	W

$\underline{\mathcal{A} = B}$ wahr : Zeilen 2 und 4
$\underline{\mathcal{B} = A \rightarrow B}$ wahr : Zeile 1, 2 und 4

Also wo $\mathcal{A}$ wahr ist, ist auch $\mathcal{B}$ wahr, Implikation.

Aussagenlogische Grundbegriffe

Tautologische Bijunktion

Definition

[Äquivalenz, $\equiv$, $\Leftrightarrow$] Zwei Aussageformen $\mathcal{A}$ und $\mathcal{B}$ heißen logisch **äquivalent**, wenn **jede** Variablenbewertung $\mathcal{A}$ und $\mathcal{B}$ **gleichwertig** macht.

Eine **äquivalente Definition** zu dieser Definition:
Die Wahrheitswerte von $\mathcal{A}$ und $\mathcal{B}$ in der Wahrheitstabelle stimmen zeilenweise überein.

Aussagenlogische Grundbegriffe

Beispiel [Äquivalenz]

Überprüfung ob die **Bijunktion** $A \leftrightarrow B$ äquivalent zu $(A \rightarrow B)\&(B \rightarrow A)$ ist.

A	B	$A \leftrightarrow B$
W	W	W
F	W	F
W	F	F
F	F	W

Hier: Aussageform $\mathcal{A} = A \leftrightarrow B$, $\mathcal{B} = (A \rightarrow B)\&(B \rightarrow A)$.

Aussagenlogische Grundbegriffe

(Fortsetzung Beispiel)

Die Wahrheitstabelle zur Überprüfung der Äquivalenz:

A	B	$\underline{\mathcal{A}} = A \leftrightarrow B$	$A \rightarrow B$	$B \rightarrow A$	$\underline{\mathcal{B}} = (A \rightarrow B)\&(B \rightarrow A)$
W	W	W	W	W	W
F	W	F	W	F	F
W	F	F	F	W	F
F	F	W	W	W	W

Die Wahrheitswerte von $\mathcal{A}$ und $\mathcal{B}$ stimmen zeilenweise überein
$\rightarrow$ $\mathcal{A}$ und $\mathcal{B}$ sind logisch äquivalent.

Aussagenlogische Grundbegriffe

Beispiel [Äquivalenz]

$A \oplus B \equiv (\neg A \& B) \vee (A \& \neg B)$
(vorher bewiesen)

Beispiel [Äquivalenz]

$A \rightarrow B \Leftrightarrow \neg A \vee B$

A	B	$A \rightarrow B$	$\neg A$	$\neg A \vee B$
0	0	1	1	1
0	1	1	1	1
1	0	0	0	0
1	1	1	0	1

Aussagenlogische Grundbegriffe

Man versucht, logische Äquivalenzen zu finden, die möglichst wenige Variablen und Verknüpfungen aufweisen.
Warum?
Man will Schaltungen produzieren; wenn die Aussage kurz ist, ist auch die Schaltung nicht so kompliziert.

Aussagenlogische Grundbegriffe

Wichtige Äquivalenzen (für die Umformung von Aussageformen)

- De Morgansche Regeln
  $\neg(A \vee B) \equiv \neg A \& \neg B$
  $\neg(A \& B) \equiv \neg A \vee \neg B$
- Die Idempotenz
  $(A \& A) \equiv A \quad A \vee A \equiv A$
- Die doppelte Negation $\neg\neg A \equiv A$
- Negation (einer Aussageform) (Shannon)
  Negation einer beliebigen Aussageform erhält man dadurch, dass man die bejahten (nicht negierten) Variablen verneint (negiert), die verneinten Variablen bejaht, Konjunktionen in Disjunktionen, Disjunktionen in Konjunktionen, Bijunktionen in Antivalenzen (exklusive oder) und Antivalenzen in Bijunktionen umwandelt.

$$\overline{F(A, \overline{B}, \vee, \&, \leftrightarrow, \oplus)} = F(\overline{A}, B, \&, \vee, \oplus, \leftrightarrow)$$

Aussagenlogische Grundbegriffe

Beispiele [Anwendung Shannonsches Theorem]

- De Morgan

  $$\neg(A\&B\&C) = \overline{(A\&B\&C)} = \overline{A} \vee \overline{B} \vee \overline{C}$$

  $$\neg(A \vee B \vee C) = \overline{(A \vee B \vee C)} = \overline{A}\&\overline{B}\&\overline{C}$$

- $\overline{A \leftrightarrow B \vee C} = \overline{A} \oplus \overline{B}\&\overline{C}$

- $\overline{\overline{\overline{\overline{(\overline{A} \vee B)} \vee \overline{(\overline{C} \vee D)}}}\&\overline{(E \leftrightarrow B)}} =$ Auflösung von unten nach oben

  $$= \overline{\overline{(A\&\overline{B}) \vee (C\&\overline{D})}\&(\overline{E} \oplus \overline{B})} =$$

  $$= \overline{(\overline{A} \vee B)\&(\overline{C} \vee D)\&(\overline{E} \oplus \overline{B})} =$$

  $$= (A\&\overline{B}) \vee (C\&\overline{D}) \vee (E \leftrightarrow B)$$

Aussagenlogische Grundbegriffe

Gesetze und Regeln für die Umformung aussagenlogischer Formeln:

$$A \wedge \mathrm{F} \equiv \mathrm{F} \qquad A \vee \mathrm{F} \equiv A$$
$$A \wedge \mathrm{W} \equiv A \qquad A \vee \mathrm{W} \equiv \mathrm{W} \qquad \neg(\neg A) \equiv A$$
$$A \wedge A \equiv A \qquad A \vee A \equiv A$$
$$A \wedge \neg A \equiv \mathrm{F} \qquad A \vee \neg A \equiv \mathrm{W}$$

Kommutativgesetze

$$A \wedge B \equiv B \wedge A$$
$$A \vee B \equiv B \vee A$$

Assoziativgesetze

$$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$$
$$A \vee (B \vee C) \equiv (A \vee B) \vee C$$

Aussagenlogische Grundbegriffe

Distributivgesetze

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C) \qquad A \wedge (A \vee B) \equiv A$$
$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C) \qquad A \vee (A \wedge B) \equiv A$$

De Morgansche Gesetze

$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$
$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$

$$A \oplus B \equiv (\neg A \wedge B) \vee (A \wedge \neg B)$$

$A \rightarrow B \equiv \neg A \vee B$ (bereits bewiesen)

$$A \leftrightarrow B \equiv (A \wedge B) \vee (\neg A \wedge \neg B)$$

$A \rightarrow B \equiv \neg B \rightarrow \neg A$ (Umkehrschluss)

Normalformen

Normalformen

Bisher betrachtet:
Zwei Aussagevariablen (Aussageformen).
Aber es ist klar, dass wir Aussageformen aus beliebig vielen Variablen mit allen möglichen Junktoren bilden können.

Beispiel [Aussageform mit 4 Variablen und 5 Junktoren]

$(\neg A \oplus B) \to ((C \vee D) \& (\neg B \vee D))$

Ziel
Man sucht eine **vereinheitlichte** Darstellung von Aussageformen, um die Aussageformen **leichter systematisch** bearbeiten zu können.

Konjunktionsterm

Definition

[Konjunktionsterm] Ein **Konjunktionsterm** bezeichnet eine Aussageform, die entweder aus einer negierten oder nicht-negierten Variablen oder der **Konjunktion** zweier oder mehrerer negierter oder nicht-negierter Variablen besteht.

Beispiel [Konjunktionsterme]

$A, \neg B$ (eine negierte/nicht-negierte Variable)
$C \& A \& \neg D$ (Konjunktion mehrerer negierter/nicht-negierter Variablen)

Konjunktionsterm

Wozu Konjunktionsterme?
$\rightarrow$ Um zu Normalform zu kommen.

Definition

[Konjunktionsterm A in B enthalten] Sind A und B zwei Konjunktionsterme, so „**ist A in B enthalten**“, wenn **alle negierten und nicht-negierten Variablen von A auch in B vorkommen**.

Beispiel

Die Aussageform $A \& \neg B$ **ist in** $A \& \neg B \& C$ **enthalten**.
Aber $A \& B$ ist nicht enthalten (B fehlt).

Disjunktive Normalform

Definition

[Disjunktive Normalform (DNF)] Eine Aussageform liegt in **disjunktiver Normalform** (DNF) vor, wenn diese entweder ein Konjunktionsterm oder die **Disjunktion** von zwei oder mehreren Konjunktionstermen, wobei **keiner in einem anderen enthalten sein darf**, ist.

DNF

A, B, C
$A \& B$
$A \vee C$
$(A \& B \& \neg C) \vee (\neg A \& B)$

Vollständige DNF

Von großer Bedeutung ist der folgende Spezialfall einer DNF:

Definition

[Vollständige DNF] Eine DNF heißt **vollständig**, wenn in **jedem Konjunktionsterm jede Aussagevariable der Aussageform vorkommt**. Die Konjunktionsterme einer vollständigen DNF heißen **Minterme**.

DNF (A, B, C)

$A \& B$	ist nicht vollständig
$A \vee C$	ist nicht vollständig
$(A \& B \& \neg C) \vee (\neg A \& B)$	ist nicht vollständig
$(A \& \neg B) \vee (\neg B \& C)$	ist nicht vollständig
$(A \& \neg B) \& C \vee A \& B \& C$	ist vollständig

Vollständige DNF

Konjunktionsterme werden durch Disjunktionen voneinander getrennt.

Wichtig: Bei vollständiger DNF müssen Variablen angegeben werden.

Zum Beispiel: $(A \& B) \vee (\neg A \& B)$ ist für A, B als Variablen vollständig, nicht aber für A, B, C.

Vollständige DNF

Theorem
Jede nicht-kontradiktorische Aussageform kann auf eine vollständige DNF gebracht werden.

Sehr wichtig für die Konstruktion von logischen Schaltungen: Z. B. wenn wir eine Wahrheitsfunktion (= gewünschte Funktion einer logischen Schaltung) in Form einer Wahrheitstabelle haben, können wir daraus eine Aussageform in DNF ableiten.

Disjunktive Normalform

Bisher:
Aussageform $\rightarrow$ Wahrheitstabelle

In der Praxis tritt der umgekehrte Prozess auf:
Funktion (Wahrheitstabelle) $\rightarrow$ Aussageform

Also:

- Funktion in Wahrheitstabelle aufschreiben
- DNF ableiten
- Ausdruck vereinfachen (je kürzer der Ausdruck, desto einfacher das elektronische Bauteil und desto geringer die Kosten)

Disjunktive Normalform

Beispiel [Ableitung einer DNF aus der Wahrheitstabelle]

Realisieren Sie die folgende Wahrheitsfunktion (aus folgender Wahrheitstabelle soll eine **DNF** abgeleitet werden):

x_1	x_2	$f(x_1, x_2)$
W	W	F
F	W	W
W	F	W
F	F	W

[f: Negation der Konjunktion]

- Wir betrachten alle **Zeilen**, deren $f(x_1, x_2) = \mathbf{W}$ und **bilden** aus den Variablen **Konjunktionsterme**, wobei die **Variable negiert** wird, wenn ihr Wert **falsch ist**.

Disjunktive Normalform

(Fortsetzung Beispiel)

- Wir erhalten:

$$\left.\begin{array}{ll} \text{2. Zeile:} & \neg A \& B \\ \text{3. Zeile:} & A \& \neg B \\ \text{4. Zeile:} & \neg A \& \neg B \end{array}\right\} \rightarrow \overbrace{(\neg A \& B) \vee (A \& \neg B) \vee (\neg A \& \neg B)}^{\text{Vollständige DNF (Variablen } A,\ B\text{)}}$$

  Jeder Konjunktionsterm beschreibt eine Zeile, die wahr wird
  $\rightarrow$ damit ist die gesamte Wahrheitstabelle beschrieben.

  Die Disjunktion dieser Aussagen wird dann wahr, wenn zumindest einer der Konjunktionsterme wahr wird.

  Die Zeilen mit $f(x_1, x_2) = \mathrm{F}$ bleiben unberücksichtigt, weil diese den Wahrheitswert X der Aussageform nicht ändern ($X \vee \mathrm{F} = X$).
  Diese Terme dürfen nicht zur DNF hinzugefügt werden, weil die DNF dann nicht äquivalent zur ursprünglichen Funktion $f(x_1, x_2)$ wäre.

Konjunktive Normalform

Definition

[Konjunktive Normalform (KNF)] Eine Aussageform liegt in **konjunktiver Normalform (KNF)** vor, wenn diese entweder ein Disjunktionsterm, oder die **Konjunktion** von zwei oder mehreren Disjunktionstermen, wobei keiner in einem anderen enthalten sein darf, ist.

Definition

[Disjunktionsterm] Ein Disjunktionsterm bezeichnet eine Aussageform, die entweder aus einer negierten oder nicht-negierten Variablen oder der **Disjunktion** zweier oder mehrerer negierter oder nicht-negierter Variablen besteht.

- Die Disjunktionsterme der vollständigen KNF heißen **Maxterme**.

Konjunktive Normalform

Beispiel [KNF]

$(A \vee B) \& (B \vee C)$ (aber nicht vollständig)

Konstruktion einer KNF

Für jede Zeile der Funktionstabelle, die **false** ist, wird der zugehörige **Maxterm** wie folgt gebildet: Alle Argumente werden miteinander mit der **Oder**-Funktion verknüpft, wobei diejenigen Argumente, die den Wert **wahr** haben, **negiert** werden. Anschließend erhält man durch die konjunktive Verknüpfung (**Und**-Funktion) aller auf diese Weise gebildeter Maxterme den äquivalenten booleschen Ausdruck in konjunktiver Normalform.

DNF und KNF

Aus der 1. Zeile des Beispiels von Folie 265 (die **F** ist):

KNF: $\underbrace{\neg A \vee \neg B}_{\text{Maxterm}} \qquad (= \neg(A \& B))$

Bemerkung: Jede DNF kann in KNF umgeschrieben werden.

Bemerkung: Es muss erfüllt sein, dass beide (KNF, DNF) gleichwertig sind.

Es gilt (das Beispiel):

$(\neg A \& B) \vee (A \& \neg B) \vee (\neg A \& \neg B) \equiv \neg(A \& B)$

Primimplikant

Ein weiterer wichtiger Begriff:

Definition

[Primimplikant] Ein Konjunktionsterm ψ heißt Primimplikant einer Aussageform $\mathcal{A}$ dann und nur dann, wenn ψ die Form $\mathcal{A}$ logisch impliziert und $\mathcal{A}$ durch keinen anderen in ψ enthaltenen Konjunktionsterm impliziert wird.

Primimplikant

Beispiel [Primimplikant]

Die Aussageform $\mathcal{A} = (A\&B) \vee (A\&\neg B\&C)$ ist eine DNF, die durch den Primimplikanten $\psi = A\&C$ impliziert wird.

- Wenn der Primimplikant wahr wird, wird die Aussage wahr (egal was B ist).
  $A\&C$ ist Implikant: offensichtlich.
  Wenn Primimplikant, dann nicht mehr verkleinerbar.
  In ψ enthaltene Konjunktionsterme sind A und C:
  weder A noch C impliziert $\mathcal{A}$.
- Alle Terme in DNF sind Implikanten.
  $A\&B$: Kandidat für Primimplikant; ist es einer?
  A, B darin enthalten. Weder A noch B impliziert $\mathcal{A}$.

Daher sind $A\&B$ und $A\&C$ die beiden einzigen Primimplikanten.

Primimplikanten einer Aussageform

- Aussageform $\rightarrow$ DNF
- Vollständige DNF
- Primimplikanten [Quine-McCluskey]
- Ermittlung der wesentlichen Primimplikanten

Primimplikanten einer Aussageform

Aussageform	**logische** Umwandlungen	DNF
$\mathcal{A} = \mathcal{A}(A, B, C)$	$\equiv$	$\mathrm{DNF}(A, B, C)$

  $\mathrm{DNF}(A, B, C) = (A \& B) \vee (A \& \neg B) \& C$
- Vollständige DNF
  Obige DNF ist nicht vollständig (im ersten Term fehlt C).
  Es gilt $(C \vee \neg C) \equiv \mathrm{W}$ und $(A \& B) \& \mathrm{W} \equiv (A \& B)$.
  Also ändert die Operation $(A \& B) \& (C \vee \neg C) \equiv (A \& B)$ die Wahrheitswerte der obigen DNF nicht:

  $$\begin{aligned} \mathrm{DNF}(A, B, C) \quad &\equiv (A \& B) \& (C \vee \neg C) \vee (A \& \neg B) \& C \equiv \\ &\equiv A \& B \& C \vee A \& B \& \neg C \vee A \& \neg B \& C \end{aligned}$$

  (diese Form ist **vollständig**)
- Primimplikanten [Quine-McCluskey]
- Ermittlung der wesentlichen Primimplikanten

Quine-McCluskey

- Das Verfahren von Quine-McCluskey
  Input: **Vollständige** DNF $\Psi = \psi_1 + \psi_2 + \cdots + \psi_k$
  1. Die **Minterme** $\psi_1, \ldots, \psi_k$ werden in einer **Liste** zusammengestellt. Dabei ist es vorteilhaft, die Minterme **in Gruppen** einzuteilen, sodass benachbarte Gruppen sich **nur in einer** Variablen unterscheiden.

     Z. B.: K_0: Alle Minterme ohne negierte Variablen
     K_1: Alle Minterme mit einer negierten Variablen
     $\vdots$ etc.
  2. Wenn sich in der Liste zwei Minterme ψ_i und ψ_j **nur in einer Variablen** unterscheiden (negiert, nicht-negiert) (was nur in benachbarten Klassen möglich ist), dann schreiben wir **einen neuen Konjunktionsterm** $\psi_{i,j}$ an, der **durch Weglassen der unterschiedlichen Variablen entsteht**.
     ψ_i und ψ_j werden dann als erledigt **abgehakt**.
  3. Wir führen Punkt 2. solange aus, bis wir **nichts** mehr zusammenfassen können. (Dabei ist es auch möglich, bereits abgehakte Terme wieder neu zusammenzufassen).
  4. Die in der Liste **verbleibenden nicht abgehakten Konjunktionsterme** sind **alle Primimplikanten von** Ψ.

Quine-McCluskey

Beispiel 1: Ermittlung von Primimplikanten einer vollständigen DNF

(Quine-McCluskey-Verfahren)

Input: Vollständige DNF $\Psi = \psi_1 + \psi_2 + \cdots + \psi_k \quad (k = 7)$

$$\begin{aligned}\Psi = \quad & \neg A \& \neg B \overset{\psi_7}{\&} \neg C \& \neg D \vee \neg A \& B \overset{\psi_6}{\&} \neg C \& \neg D \vee \neg A \& B \overset{\psi_4}{\&} C \& \neg D \\ & \vee A \& \neg B \overset{\psi_1}{\&} C \& D \vee A \& B \overset{\psi_5}{\&} \neg C \& \neg D \vee A \& B \overset{\psi_2}{\&} \neg C \& D \vee A \& B \overset{\psi_3}{\&} C \& \neg D\end{aligned}$$

Quine-McCluskey

$\neg A : \overline{A}$ & : weggelassen

		Schritt (2) (erste Anwendung)	Schritt (2) (zweite Anwendung) (Fertig)
K_0:	—		
K_1:	$A\,\overline{B}\,C\,D$ (1)	$A\,\overline{B}\,C\,D$ (1)	$A\,\overline{B}\,C\,D$ (1)
	$A\,B\,\overline{C}\,D$ (2)✓	$B\,C\,\overline{D}$ (4), (3)✓	$A\,B\,\overline{C}$ (5), (2)
	$A\,B\,C\,\overline{D}$ (3)✓	$A\,B\,\overline{C}$ (5), (2)	$B\,\overline{D}$ (6), (4); (5), (3)☒
		$A\,B\,\overline{D}$ (5), (3)✓	(4, 3), (6, 5)
K_2:	$\overline{A}\,B\,C\,\overline{D}$ (4)✓	$\overline{A}\,B\,\overline{D}$ (6), (4)✓⊗	
	$A\,B\,\overline{C}\,\overline{D}$ (5)✓	$B\,\overline{C}\,\overline{D}$ (6), (5)✓	
K_3:	$\overline{A}\,B\,\overline{C}\,\overline{D}$ (6)✓	$\overline{A}\,\overline{C}\,\overline{D}$ (7), (6)	$\overline{A}\,\overline{C}\,\overline{D}$ (7), (6)
K_4:	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ (7)✓		

Quine-McCluskey

$\otimes$ (6), (4) unterscheiden sich in C, $\overline{C}$
$\overset{\text{Schritt (2)}}{\longrightarrow}$ (6), (4) werden abgehakt und durch den neuen Term $\overline{A}\,B\,\overline{D}$ ersetzt (kommt in Gruppe K_2)

$\boxtimes$ (6), (4); (5), (3): $\overline{A}\,B\,\overline{D}$ und $A\,B\,\overline{D}$ unterscheiden sich in $\overline{A}$, A
$\rightarrow$ neuer Term $B\,\overline{D}$ entsteht

Primimplikanten (nicht abgehakte Terme):
$A\,\overline{B}\,C\,D,\;\; A\,B\,\overline{C},\;\; B\,\overline{D},\;\; \overline{A}\,\overline{C}\,\overline{D}.$

Quine-McCluskey

Beispiel 2: Ermittlung aller Primimplikanten von $\mathcal{A} = (A\&B) \vee (A\&\neg B)\&C$ (keine vollständige DNF)

(1) $\underline{A\&B\&C}$ ✓	$A\&B$ ((1), (2))
(2) $A\&B\&\neg C$ ✓	$A\&C$ ((1), (3))
(3) $A\&\neg B\&C$ ✓	

Primimplikanten einer Aussageform (Fortsetzung)

- Überprüfung
  Prim $\Rightarrow$ DNF? Kein im Prim enthaltener Konjunktionsterm $\Rightarrow$ DNF?
  Weshalb reicht es, nur solche Zeilen zur Übereinstimmung von Prim und DNF (in der Wahrheitstabelle) zu betrachten, wo Prim $=$ W?
  Bereits erklärt (Folien 243–244) $\Rightarrow$: Tautologische Subjunktion

Prim	DNF	Prim $\Rightarrow$ DNF
F	F	W
F	W	W
~~W~~	~~F~~	~~F~~
W	W	W

Also: Wo Prim $=$ F ist Prim $\Rightarrow$ DNF W (also Prim $\Rightarrow$ DNF), wenn Prim $=$ W, darf nicht der Fall — (3. Zeile) auftreten, damit Prim $\Rightarrow$ DNF, also muss für Prim $=$ W die 4. Zeile gelten.

Überprüfung [Primimplikanten]

- Ob der Konjunktionsterm ein Implikant ist (d. h. ob dieser die Aussageform impliziert)
- Kein in dem Implikant enthaltener Konjunktionsterm impliziert die Aussageform

Ist $A\&C$ ein Primimplikant von $\mathcal{A} = (A\&B) \vee (A\&\neg B)\&C$ aus Beispiel 2?

Überprüfung [Primimplikanten]

A	B	C	① $A \& B$	$A \& \neg B$	② $(A \& \neg B) \& C$	$① \overset{A}{\vee} ②$	$A \& C$
W	W	W	W	F	F	W	W
W	F	W	F	W	W	W	W
F	W	W	F	F	F	F	F
F	F	W	F	F	F	F	F
W	W	F	W	F	F	W	F
W	F	F	F	W	F	F	F
F	W	F	F	F	F	F	F
F	F	F	F	F	F	F	F

Überprüfung [Primimplikanten]

- Impliziert $A \& C$ $\mathcal{A} = (A \& B) \vee (A \& \neg B) \& C$?
  - Ja, weil wo $A \& C$ wahr ist (Zeile 1, 2), ist auch $\mathcal{A}$ wahr.
- Impliziert A oder C (die Konjunktionsterme, die im Term $A \& C$ enthalten sind) $\mathcal{A}$?
  - A: Nein wegen Zeile 6: $A = \mathrm{W}$ aber $\mathcal{A} = \mathrm{F}$, also $A \nRightarrow \mathcal{A}$.
  - C: Nein wegen Zeile 3: $C = \mathrm{W}$ aber $\mathcal{A} = \mathrm{F}$, also $C \nRightarrow \mathcal{A}$.
- $A \& B$:
  - Auch Implikant: Wo $A \& B$ wahr (Zeilen 1, 5), ist auch $\mathcal{A}$ wahr.
  - A: Kein Implikant (oben bewiesen).
  - B: Kein Implikant (wegen Zeilen 3, 7).

Daher ist auch $A \& B$ Primimplikant.

Verknüpfungsbasen

Verknüpfungsbasen

Es stellt sich die Frage:
Welche und wie viele Junktoren werden zur Darstellung aller Aussageformen benötigt?

Definition

[Verknüpfungsbasis] Als Verknüpfungsbasis bezeichnen wir die Menge der Junktoren, die ausreicht, jede Wahrheitsfunktion als Aussageform darzustellen.

Verknüpfungsbasen

Aus dem Theorem auf Folie 263 folgt, dass man nur $\neg$, &, $\vee$ braucht, um eine DNF/KNF zu erstellen. Daher kann man mit $\neg$, &, $\vee$ alle Aussageformen darstellen.
$\rightarrow \{\neg, \&, \vee\}$ ist eine Verknüpfungsbasis.

Diese Erkenntnis hat wichtige praktische Bedeutung:
Man benötigt im Prinzip nur drei logische Bauteile, um alle Aussageformen in Hardware zu realisieren.
Frage: Geht es mit noch weniger Junktoren?
$\rightarrow$ Ja

Verknüpfungsbasen

- Einelementige Verknüpfungsbasen:
  Shefferbasis $\{|\}$
  Peircebasis $\{\downarrow\}$
- Shefferbasis
  Die Wahrheitstabelle für die Sheffer-Funktion $A \mid B$ („A Sheffer B")

A	B	$A \mid B$
W	W	F
F	W	W
W	F	W
F	F	W

Shefferbasis

Es gilt $A \mid B \overset{\text{Definition}}{=} \neg(A \& B) \overset{\text{De Morgan}}{=} \neg A \vee \neg B$.

Aufgrund dieser Äquivalenz wird der entsprechende logische Baustein als **NAND** (NOT AND) bezeichnet.

Also: Mit $\{|\}$ kann alles dargestellt werden – man braucht nur einen einzigen Baustein.

Frage: Wie kann man zeigen, dass die Shefferbasis eine Verknüpfungsbasis ist?

Shefferbasis

Um dies nachzuweisen, muss man die Junktoren der Verknüpfungsbasis $\{\neg, \&, \vee\}$ mit $|$ darstellen können:

$$\neg: \quad \neg A \equiv \neg(A \& A) \equiv A \mid A$$

$$\vee: \quad A \vee B \equiv \neg\neg(A \vee B) \equiv \neg(\neg A \& \neg B) \equiv \neg((A \mid A) \& (B \mid B)) \equiv (A \mid A) \mid (B \mid B)$$

$$\wedge: \quad A \& B \equiv \neg \overbrace{\neg(A \& B)}^{\text{Def}} \equiv \neg(A \mid B) \equiv (A \mid B) \mid (A \mid B)$$

Peircebasis

- Die Peircebasis

  Die Wahrheitstabelle für die **nicodsche Wahrheitsfunktion** $\downarrow$

A	B	$A \downarrow B$
W	W	F
F	W	F
W	F	F
F	F	W

  Es gilt: $A \downarrow B \overset{\text{Definition}}{=} \neg A \& \neg B \overset{\text{De Morgan}}{=} \neg(A \vee B)$,
  was den logischen Baustein die Bezeichnung **NOR** (NOT OR) ergibt.

  Dass $\{\downarrow\}$ eine Verknüpfungsbasis ist: Beweis analog wie bei $\{|\}$.

Verknüpfungsbasis

Praktische Bedeutung dieser einelementigen Verknüpfungsbasen:

Für die Realisierung **aller** logischer Operationen wird ein **einziger** Grundbaustein benötigt.
Das bedeutet **aber nicht**, dass die Verwendung eines **einzigen** Bausteins **optimal** hinsichtlich der Anzahl der Bausteine für eine Schaltung ist.

Boolesche Algebra

Boolesche Algebra

Shannon hat auf den Zusammenhang zwischen der booleschen Algebra und dem Schaltungsdesign hingewiesen.

Boolesche Algebra: begründet von George Boole (1815–1864).

Ganz allgemein in der Mathematik wird unter einer Algebra eine **Trägermenge** und eine oder mehrere **Verknüpfungen** von Elementen dieser Menge verstanden, die genau definierten **Axiomen** genügen.

Verknüpfung

Nähere Betrachtung des Begriffs Verknüpfungen:

Definition

Eine n-stellige (n-äre) Operation in einer Menge X ist jede Funktion f, die jedes n-Tupel x von X mit einem Element $y \in X$ verknüpft.
Man sagt dann auch, dass X bezüglich f **abgeschlossen** ist.
[Abgeschlossen: Ergebnis $y = f(x)$ ist wiederum in der Menge X.]

Verknüpfung

Beispiel [unäre und binäre Operationen in $\mathbb{R}$]

$\mathbb{R}$: Die Menge der reellen Zahlen

In $\mathbb{R}$ ist

- $f(x) = x + 1$ eine **unäre** Operation
- $f(x, y) = x + y$ eine **binäre** Operation.

Oder: Die **Menge** $\mathbb{R}$ **ist abgeschlossen bezüglich** f (in beiden Fällen).

Bemerkung: **Binär** bezeichnet hier nicht die Zahlendarstellung, sondern die Anzahl (2) der verknüpften Elemente (x, y).

Hingegen:

- In $\mathbb{N}$ (Menge der natürlichen Zahlen) ist $f(m, n) = m - n$ **keine binäre Operation** (weil $\mathbb{N}$ nicht abgeschlossen bezüglich f ist).

Axiomensystem der booleschen Algebra

- Axiomensystem der booleschen Algebra

Definition

[Boolesche Algebra] Als boolesche Algebra bezeichnen wir ein Sechs-Tupel **BA mit Trägermenge** B, wenn in B zwei binäre Operationen $\wedge$ und $\vee$ sowie eine unäre Operation $'$ definiert sind **und** zwei spezielle Elemente **0** und **1** existieren **und** folgende Axiome gelten:

1. $\forall x, y \in B : x \vee y = y \vee x$ Kommutativgesetz
2. $\forall x, y \in B : x \wedge y = y \wedge x$ Kommutativgesetz
3. $\forall x, y, z \in B : x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ Distributivgesetz
4. $\forall x, y, z \in B : x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ Distributivgesetz
5. $\forall x \in B : x \vee 0 = x$
6. $\forall x \in B : x \wedge 1 = x$
7. $\forall x \in B : x \vee x' = 1$
8. $\forall x \in B : x \wedge x' = 0$
9. $0 \neq 1$ also: $|B| \geq 2$; $|B|$: Kardinalität von B (= Anzahl der Elemente von B)

Axiomensystem der booleschen Algebra

$x \wedge y$: das **Produkt** von x und y
$x \vee y$: die **Summe** von x und y
x': das **Komplement** von x
0: das **Nullelement**
1: das **Einselement**

[**Boolesche Algebra** BA kennzeichnen wir als sechs-Tupel BA$= \langle B, \wedge, \vee, ', 0, 1\rangle$]

Axiomensystem der booleschen Algebra

Bemerkung

- In einigen Literaturquellen ist die boolesche Algebra BA mit der Trägermenge B identifiziert.
- Die Zeichen $\wedge$ („mal") und $\vee$ („plus") sind in der Definition von BA allgemein für die Bezeichnung der Binäroperationen zu verstehen (obwohl für eine spezielle BA (Aussagenlogik) diese Operationen identisch mit $\wedge$ (logisch und) und $\vee$ (logisch oder) sind, $'$ ist dann die Negation).

Axiomensystem der booleschen Algebra

Beispiel [keine boolesche Algebra]

$\mathbb{R}$: Die Menge der reellen Zahlen
$\wedge$: Das Produkt von reellen Zahlen
$\vee$: Die Summe von reellen Zahlen
0.0: Nullelement
1.0: Einselement

Ohne Komplement $'$ zu definieren sieht man, dass mit $\wedge$, $\vee$ das Axiom 4. verletzt ist, weil $x \vee (y \wedge z) = x + (y \cdot z) \neq (x \vee y) \wedge (x \vee z) = (x + y) \cdot (x + z)$.

Daher ist $\langle \mathbb{R}, \cdot, +, ', 0.0, 1.0 \rangle$ **keine** boolesche Algebra.

Bemerkung: In unserem Beispiel gilt auch Axiom 7. **nicht**: Sei $x' = -x$,
$x + x' = x - x = 0 \neq 1$

Axiomensystem der booleschen Algebra

Beispiel [boolesche Algebra]

$$B_0 = \langle \underbrace{\{ \overbrace{\emptyset}^{\text{Leere Menge}}, \overbrace{M}^{\text{Menge } (\neq\emptyset)} \}}_{B}, \underbrace{\cap}_{\wedge}, \underbrace{\cup}_{\vee}, \underbrace{-}_{'}, \underbrace{\emptyset}_{0}, \underbrace{M}_{1} \rangle$$

Ist eine **zweielementige** boolesche Algebra mit den bekannten Operatoren $\cap$ (Durchschnitt), $\cup$ (Vereinigung) und $-$ (Komplement).

Axiomensystem der booleschen Algebra

Beispiel (Fortsetzung)

1. $\emptyset \cup M = M \cup \emptyset$
2. $\emptyset \cap M = M \cap \emptyset$
3. $\forall x, y, z \in B$, d. h. $x = \begin{cases} \emptyset \\ M \end{cases}, y = \begin{cases} \emptyset \\ M \end{cases}, z = \begin{cases} \emptyset \\ M \end{cases}$

   $\rightarrow$ 8 Fälle

x	y	z	$x \cap (y \cup z)$	$(x \cap y) \cup (x \cap z)$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
M	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	M	$\emptyset$	$\emptyset$	$\emptyset$
M	M	$\emptyset$	M	M
$\emptyset$	$\emptyset$	M	$\emptyset$	$\emptyset$
M	$\emptyset$	M	M	M
$\emptyset$	M	M	$\emptyset$	$\emptyset$
M	M	M	M	M

Axiomensystem der booleschen Algebra

Beispiel (Fortsetzung)

4. Analog wie 3.
5. $x = M$: $M \cup \emptyset \overset{(x \vee 0) = x}{=} M$ $(= x)$; $\quad x = \emptyset$: $\emptyset \cup \emptyset = \emptyset$ $(= x)$
6. $x = M$: $M \cap M \overset{(x \wedge 1) = x}{=} M$ $(= x)$; $\quad x = \emptyset$: $\emptyset \cap M = \emptyset$ $(= x)$
7. $x = M$: $M \cup \emptyset \overset{(x \vee x') = 1}{=} M$ $(= 1)$; $\quad x = \emptyset$: $\emptyset \cup M = M$ $(= 1)$
8. $x = M$: $M \cap \emptyset \overset{(x \wedge x') = 0}{=} \emptyset$ $(= 0)$; $\quad x = \emptyset$: $\emptyset \cap M = \emptyset$ $(= 0)$
9. $\emptyset \overset{0 \neq 1}{\neq} M$

Aussagenlogik und boolesche Algebra

Theorem

Die bezüglich Konjunktion, Disjunktion und Negation abgeschlossene Menge $\mathcal{A}$ der Aussagen ist die Trägermenge der booleschen Algebra

$$BA = \{\mathcal{A}, \&, \vee, \neg, W, F\}$$

Daher: Die **gesamte Theorie der booleschen Algebra kann** für die Aussagenlogik und somit **auch für logische Schaltungen verwendet werden**.
Umgekehrt müssen **alle Gesetze**, die wir in **der Aussagenlogik** kennengelernt haben, **auch für die boolesche Algebra gültig sein** (Idempotenz, De Morgan, Doppelte Negation, ...).

Schreibweise

Wir werden in Zukunft in der Schreibweise für

- W das Element 1
- F das Element 0

benutzen und

- das Produkt (Konjunktion) mit $\cdot$
- die Summe (Disjunktion) mit $+$
- und das Komplement (Negation) mit $\bar{\ }$

bezeichnen.
Die Aussagevariablen **werden mit Indices versehen** (wie in der Mathematik), z. B. x_1, x_2, y_1.

Beispiel 1

Die Schreibweise für die DNF $(A\&\neg B) \vee (\neg C\&D)$ wäre dann $x_1\overline{x_2} + \overline{x_3}x_4$ ($\cdot$ weggelassen).

Schreibweise

Beispiel 2: Gültigkeit des Distributivgesetzes (4.)

$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
$a + bc = (a + b)(a + c)$

$$a + bc \stackrel{?}{=} (a+b)(a+c) = aa + ac + ba + bc \stackrel{1.}{=} a + ac + ba + bc \stackrel{2.}{=} a + ba + bc \stackrel{3.}{=} a + bc$$

1. $aa = a$
2. $a + ac = a(1 + c) = a \cdot 1 = a$
3. $a + ba = a(1 + b) = a \cdot 1 = a$

DNF und KNF

Ableitung einer vollständigen DNF und KNF mittels boolescher Algebra

DNF (vollständig): Überall wo im Konjunktionsterm x_i fehlt, fügt man $(x_i + \overline{x_i}) = W = 1$ als Multiplikator hinzu, danach wendet man das Distributivgesetz an:

$$\begin{aligned} a \wedge (b \vee c) &= (a \wedge b) \vee (a \wedge c) \\ a(b+c) &= ab + ac \end{aligned}$$

Beispiel 1 [vollständige DNF]

- **Aussage:** $y = (x_1 + \overline{x_3})x_2$

$$\begin{aligned} &= x_1x_2 + x_2\overline{x_3} \text{ } \textbf{(DNF)} \\ &\overset{\text{Erweiterung}}{=} x_1x_2(x_3 + \overline{x_3}) + (x_1 + \overline{x_1})x_2\overline{x_3} \\ &= x_1x_2x_3 + x_1x_2\overline{x_3} + x_1x_2\overline{x_3} + \overline{x_1}x_2\overline{x_3} \\ &\overset{(a+a=a)}{=} x_1x_2x_3 + x_1x_2\overline{x_3} + \overline{x_1}x_2\overline{x_3} \text{ } \textbf{(vollständige DNF)} \end{aligned}$$

DNF und KNF

Beispiel 2 [vollständige DNF]

- **Aussage:** $y = (x + \overline{y}z)(\overline{x} + z)$

$$
\begin{aligned}
&= x\overline{x} + xz + \overline{x}\,\overline{y}z + \overline{y}z \\
&= xz + \overline{x}\,\overline{y}z + \overline{y}z \text{ (DNF)} \\
&\overset{\text{Erweiterung}}{=} xz(y + \overline{y}) + \overline{x}\,\overline{y}z + \overline{y}z(x + \overline{x}) \\
&= xyz + x\overline{y}z + \overline{x}\,\overline{y}z + x\overline{y}z + \overline{x}\,\overline{y}z \\
&= xyz + x\overline{y}z + \overline{x}\,\overline{y}z \text{ (vollständige DNF)}
\end{aligned}
$$

DNF und KNF

KNF (vollständig): Überall wo im Disjunktionsterm x_i fehlt, wird $x_i\overline{x_i} = F = 0$ in den Term (künstlich) eingefügt (weil $(x + 0) = x$), danach wendet man das Distributivgesetz an:

$$\begin{aligned} a \vee (b \wedge c) &= (a \vee b) \wedge (a \vee c) \\ a + (bc) &= (a + b)(a + c) \end{aligned}$$

Beispiel [vollständige KNF]

- **Aussage:** $y = (x_1 + \overline{x_2}x_3)(\overline{x_1} + x_3) \overset{\text{Distributivgesetz}}{=}$
  $(x_1 + \overline{x_2})(x_1 + x_3)(\overline{x_1} + x_3)$ **(KNF)** $\overset{\text{Erweiterung}}{=}$
  $(x_1 + \overline{x_2} + x_3\overline{x_3})(x_1 + x_2\overline{x_2} + x_3)(\overline{x_1} + x_2\overline{x_2} + x_3) \overset{\text{Distributivgesetz}}{=}$
  $(x_1+\overline{x_2}+x_3)(x_1+\overline{x_2}+\overline{x_3})(x_1+x_2+x_3)(x_1+\overline{x_2}+x_3)(\overline{x_1}+x_2+x_3)(\overline{x_1}+\overline{x_2}+x_3) =$
  $(x_1 + x_2 + x_3)(x_1 + \overline{x_2} + x_3)(\overline{x_1} + x_2 + x_3)(x_1 + \overline{x_2} + \overline{x_3})(\overline{x_1} + \overline{x_2} + x_3)$
  (vollständige KNF)

Schaltnetze

Schaltnetze

Der Schritt von
Aussageformen (und davon abgeleiteten Wahrheitsfunktionen)
zu einer
logischen Realisierung dieser booleschen Funktion
(später auch **Schaltfunktionen** genannt)
benötigt
Hardware (in einer **graphischen** Repräsentation).

Schaltnetz: Nichts anderes als die **graphische Repräsentation einer logischen Aussageform**.

(Anders: Die graphische Darstellung einer Schaltfunktion nennen wir **Schaltnetz**.)

Schaltfunktion

Wie definiert man eine Schaltfunktion?

Eine **Schaltfunktion** ist eine Abbildung der Form: $f : B^n \to B^m$, wobei $B = \{0, 1\}$.

(Es werden also n Eingänge auf m Ausgänge abgebildet.)

Schaltnetz

Schaltnetz (also: graphische Darstellung einer Schaltfunktion):

Knoten: Logische Bausteine (Schaltfunktionen)

Kanten: Leitungen

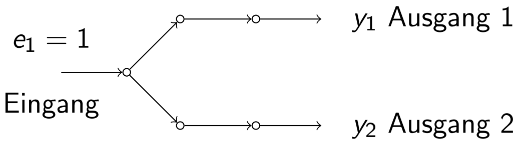
Eingang: ist eine **Kante**, an die ein logisches Signal angelegt wird

Ausgang: ist eine **Kante**, die ein logisches Signal als Resultat liefert

Schaltnetz

Beispiel [Schaltnetz]

Ein Eingangssignal wird auf 2 Ausgänge geleitet:



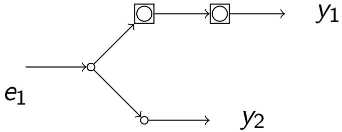
Schaltnetz

[Basisknoten]

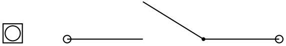
Grundelemente jeder Schaltung: Schalter

In der graphischen Darstellung: ◙

z. B.



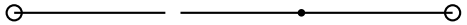
Einfache Schaltnetze

Schalter: 

Jeder Variable (Operand) der Schaltfunktion wird in der graphischen Darstellung ein Schalter zugeordnet.

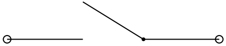
Eine logische Variable (x) kann zwei logische Werte (0, 1) annehmen, die den zwei Zuständen des Schalters entsprechen.

- Bei $x = 1$ ist der Schalter **ein**, d. h. **geschlossen**.

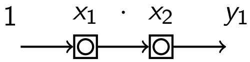
  Das Signal kommt durch (weil die Leitung hergestellt ist).

- Wenn $x = 0$ ist der Schalter **aus**, d. h. **offen**

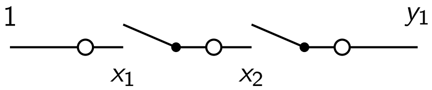
  

  und der Stromfluss wird unterbrochen.

Schaltnetz für Konjunktion



Zwei Operanden (Variablen): Zwei Schalter **seriell** geschaltet.

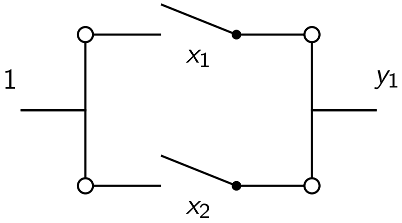


Wenn das Signal (1) des Eingangs durchkommt: $y_1 = 1$
Wenn die Leitung unterbrochen ist: $y_1 = 0$

$y_1 = 1 \Leftrightarrow$ wenn beide Schalter geschlossen $(x_1 = 1 \wedge x_2 = 1)$
$y_1 = 0 \Leftrightarrow$ wenn ein (oder beide) Schalter offen sind $(x_1 = 0 \vee x_2 = 0)$

Also: Dieses Schaltnetz realisiert die Konjunktion $(y_1 = x_1 x_2)$.

Schaltnetz für Disjunktion



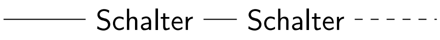
Oder: $x_1 + x_2 = y_1$

$y_1 = 1 \Leftrightarrow x_1 = 1 \vee x_2 = 1$
$y_1 = 0 \Leftrightarrow x_1 = 0 \wedge x_2 = 0$

Also: Das Signal (1) kommt nur dann nicht durch, wenn beide Schalter offen sind.

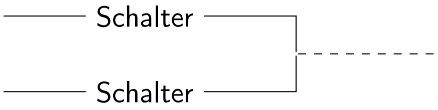
Serien- und Parallelschaltung

- Die **Serienschaltung** von Schaltern entspricht einer **Konjunktion** der Variablen:



Serienschaltung

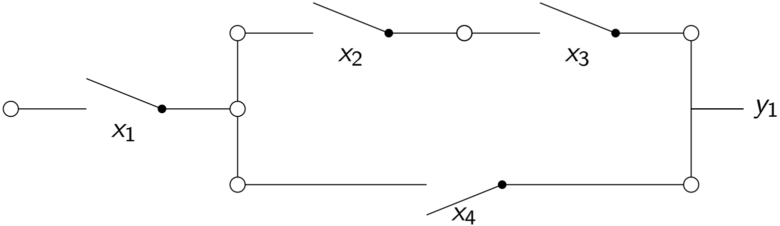
- Die **Parallelschaltung** von Schaltern entspricht einer **Disjunktion** der jeweiligen Variablen:



Parallelschaltung

Serien- und Parallelschaltung

Beispiel [Schaltnetz aus mehreren Schaltern]

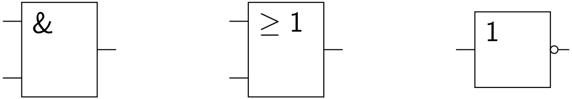


(Der Schalter für die Variable x_4 stellt eine Negation dar.)

Die dazugehörige Schaltfunktion: $y_1 = x_1(x_2x_3 + \overline{x_4})$

Logische Gatter

- **Die Gatter** repräsentieren die Schalterdarstellung **logischer Basisfunktionen**.
- Die Gatter sind Grundbausteine für die Schaltnetze.
- Die Gatter für $\wedge, \vee, \neg$ (die wichtigsten logischen Operationen — sie bilden die Verknüpfungsbasis):



Gatter für **Konjunktion** | Gatter für **Disjunktion** | Gatter für **Negation**

Logische Gatter

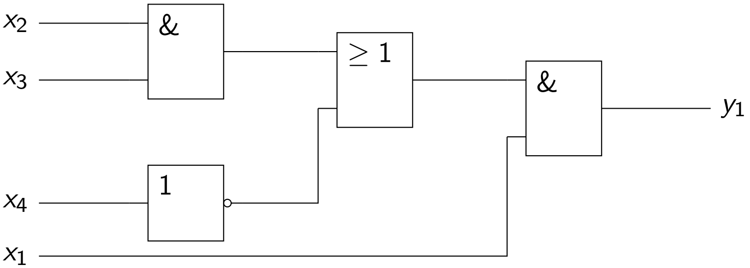
Also: Die Gatter sind Realisierungen logischer Funktionen der Form $f(x_1, x_2) = y_1$ (gilt nicht für $\neg$). Es werden also zwei boolesche Werte (Eingänge) auf einen Ausgang abgebildet.

Hat man mehrere Ausgänge, so bildet man für jeden einzelnen Ausgang eine logische Funktion (Schaltnetz) aus diesen Gattern und kombiniert diese dann zu einer einzigen Schaltung.

(Das Schaltnetz kann mit Gattern dann eleganter gezeichnet werden.)

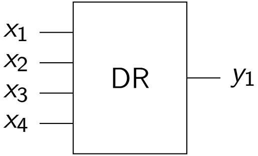
Logische Gatter

Beispiel [Gatterdarstellung des Schaltnetzes aus Folie 318]



Logische Gatter

Das Schaltnetz aus der vorigen Folie kann in einem nächsten Abstraktionsschritt als neuer Grundbaustein repräsentiert werden, wie z. B. als DR:



Logische Gatter

Durch immer **weitere Abstraktionen** kann man so immer **komplexere** Schaltungen aufbauen → **hierarchischer Hardwareaufbau**

Auch der modernste Prozessor besteht nur aus solchen einfachen Bausteinen (in sehr **großer** Anzahl).

Die Schaltungen, die mit **VLSI** (**V**ery **L**arge **S**cale **I**ntegration) Technologie erzeugt sind, besitzen 10^5 Transistoren pro mm^2.

Heutige Prozessoren (z. B. Intel Xeon Broadwell-E5) erreichen schon Transistorendichten von über 10^7 Transistoren pro mm^2.

Arithmetische Schaltnetze

Arithmetische Schaltnetze

Ziel:
Schaltnetze für fundamentale arithmetische Operationen.

Zunächst werden die Grundbausteine präsentiert.

Addition von zwei Ein-Bit-Zahlen (x_1, x_2)

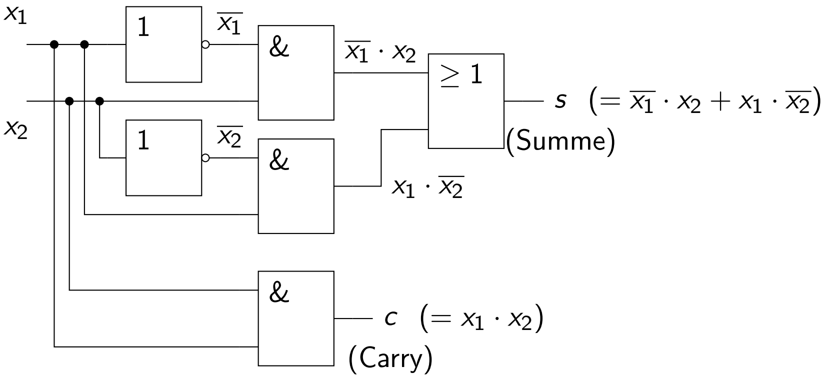
- Zuerst die Aufstellung der Wahrheitstabelle für die Addition der Bits x_1 und x_2. Berücksichtigt wird dabei, dass nicht nur eine Summe s, sondern auch ein **Übertrag** c (für Carry) auftreten.

x_1	x_2	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Zwei DNFs, abgeleitet aus der Tabelle (für die beiden Ausgänge s und c):
  $s = \overline{x_1} \cdot x_2 + x_1 \cdot \overline{x_2}$ (XOR)
  $c = x_1 \cdot x_2$

Addition von zwei Ein-Bit-Zahlen (x_1, x_2)

- Diese DNF-Darstellung der Schaltfunktionen für s und c lässt sich direkt in ein Schaltnetz mit den logischen Gattern **AND**, **OR** und **NOT** übertragen. (Z. B. bei $c = x_1 \cdot x_2$: **UND**-Baustein (x_1 und x_2 sind „verundet“))



Addition von zwei Ein-Bit-Zahlen (x_1, x_2)

- Dieses Schaltnetz nennt man Halbaddierer (die Erklärung für diese Bezeichnung folgt auf Folie 337).
- Für diese (einfache) Operation benötigt man **6 logische Gatter**.
- Die DNFs für s und c kann man vereinfachen und auch s als **Funktion** von x_1, x_2 und c ausdrücken. Durch geeignete Umformung (siehe Folien 254–255) erhalten wir

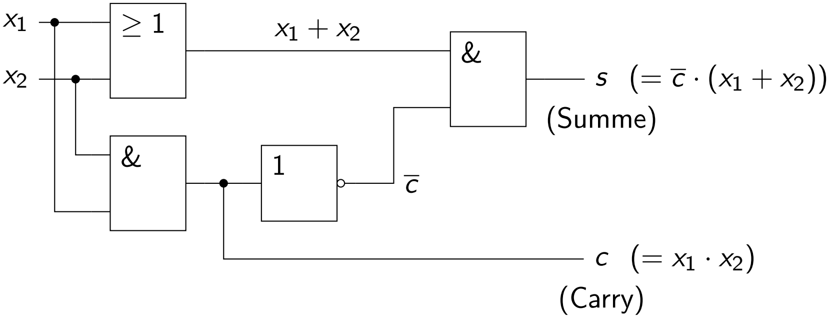
  $$s = \overline{c}(x_1 + x_2)\,.$$

  Gemeinsam mit dem Ausdruck

  $$c = x_1 x_2$$

  können wir diese Umformung in eine logische Schaltung (vereinfachter Addierer) übertragen.

Addition von zwei Ein-Bit-Zahlen (x_1, x_2)



Dieser vereinfachte Halbaddierer benötigt nur **4 Bausteine** (und damit auch weniger Leitungen), um dieselbe Funktion darzustellen (dank des theoretischen Unterbaus der booleschen Algebra).

Addition von Binärzahlen

Frage: Wie addiert man zwei n-Bit-Binärzahlen ($n > 1$)?

$$\begin{array}{llcl} \text{Die Summanden:} & x & = & (x_{n-1} \ldots x_1 \; x_0) \\ & y & = & (y_{n-1} \ldots y_1 \; y_0). \\ \text{Die Summe:} & s & = & (s_{n-1} \ldots s_1 \; s_0). \end{array}$$

Die Addition von x und y verläuft in n Schritten für $i = 0, 1, \ldots, n-1$ (im Schritt i wird die i-te Bit-Stelle behandelt).

Addition von Binärzahlen

Im Schritt i ($i = 0, \ldots, n-1$) werden die Ein-Bit-Zahlen x_i und y_i zusammen addiert mit dem Übertrag (Ein-Bit-Zahl) c_{i-1}, der bei der Addition im vorherigen Schritt $i-1$ entstanden ist (im Schritt $i = 0$ ist $c_{i-1} = 0$).

Dabei entsteht nicht nur die Ein-Bit-Zahl s_i (i-te Stelle der Summe) sondern auch der Übertrag c_i für den **nächsten** Schritt $i+1$.

(Das Resultat ist **gültig**, wenn $c_{n-1} = 0$. Sonst bekommen wir eine $(n+1)$-stellige Zahl, die außerhalb des Darstellungsintervalls liegt).

Formalisierung der Operation des i-ten Schrittes:
$f : (x_i, y_i, c_{i-1}) \to (s_i, c_i)$.

Addition von Binärzahlen

- Die Wahrheitstabelle für die Funktion f:

	x_i	y_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
$\otimes$	1	1	0	0	1
	1	1	1	1	1

Addition von Binärzahlen

- $n = 4, \quad x = 1001 = (x_3 x_2 x_1 x_0), \quad y = 0101 = (y_3 y_2 y_1 y_0)$

Schritt 0: $x_0 = 1, y_0 = 1, c_{-1} = 0 \rightarrow s_0 = 0, \; c_0 = 1$ (Zeile $\otimes$ der Tabelle)

Schritt 1: $x_1 = 0, y_1 = 0, \; c_0 = 1 \rightarrow s_1 = 1, \; c_1 = 0$

Schritt 2: $x_2 = 0, y_2 = 1, \; c_1 = 0 \rightarrow s_2 = 1, \; c_2 = 0$

Schritt 3: $x_3 = 1, y_3 = 0, \; c_2 = 0 \rightarrow s_3 = 1, \; c_3 = 0$

$c_{n-1} = c_3 = 0 \rightarrow$ kein Overflow $\rightarrow$ Das Ergebnis $s = s_3 s_2 s_1 s_0 = 1110$ ist gültig.

Addition von Binärzahlen

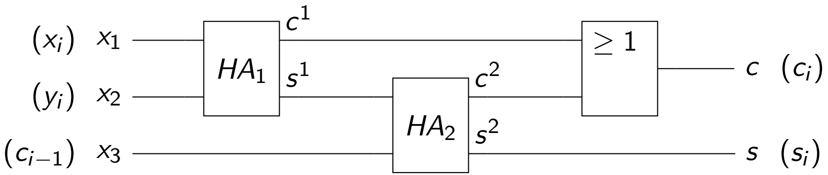
- $x = 0101, \quad y = 1101$

$$
\begin{array}{ccccc}
 & 0 & 1 & 0 & 1 \\
 & {}_{(1)}1 & {}_{(1)}1 & {}_{(0)}0 & {}_{(1)}1 \\
\hline
1 & 0 & 0 & 1 & 0
\end{array}
$$

$c_3 = 1 \rightarrow$ Überlauf (das Ergebnis ist eine 5-stellige Zahl, daher ungültig)

Halb- und Volladdierer

In jedem Schritt werden drei Ein-Bit-Zahlen (x_i, y_i, c_{i-1}) addiert. Dabei müssen **zwei** Summen von Ein-Bit-Zahlen erzeugt werden $(x_i + y_i = s', s' + c_{i-1})$. Dazu benötigt man zwei Halbaddierer (HA_1, HA_2). Aus diesen lässt sich der **Volladdierer** aufbauen.



Halb- und Volladdierer

Zur Überprüfung der Funktionalität des Volladdierers betrachten wir den Fall $x_1 = x_2 = x_3 = 1$ ($x_i = y_i = c_{c-1} = 1$) (**die letzte Zeile der Wahrheitstabelle**).
Am Ausgang soll $c = 1, s = 1$ ($c_i = 1, s_i = 1$) gelten.

Beim 1. Halbaddierer: $s^1 = 0, c^1 = 1$ (weil $x_1 + x_2 = 1 + 1$)

Beim 2. Halbaddierer: $s^2 = 1, c^2 = 0$ (weil $s^1 + x_3 = 0 + 1$)

Ausgang:
$c = 1$ (weil $c^1 + c^2 = 1 + 0$)
$s = 1$ (weil $s^2 = 1$)
ok $\checkmark$

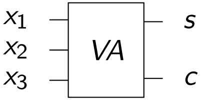
(Die übrigen Fälle kann man auf ähnliche Weise überprüfen.)

Warum die Namen Volladdierer und Halbaddierer?

Bei der Addition erzeugt der Volladdierer **zwei** Summen (von Ein-Bit-Zahlen) und der Halbaddierer nur **eine** Summe (also nur **die Hälfte** derer die bei der **Voll**addition nötig sind).

Volladdierer

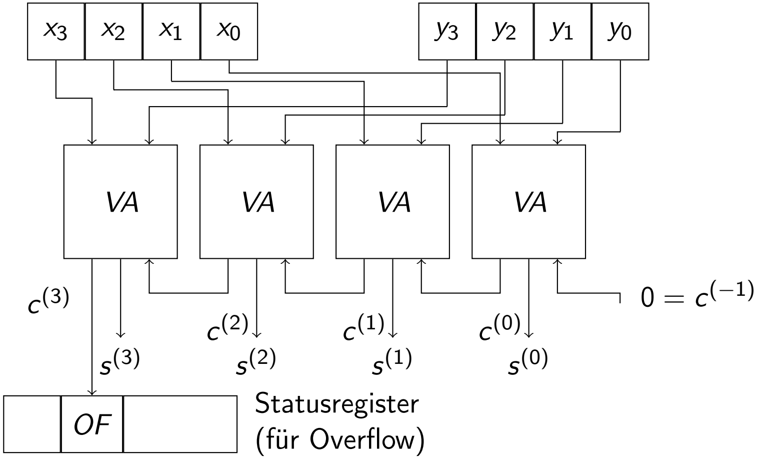
Nun haben wir einen neuen Grundbaustein „Volladdierer“ (VA) mit drei Eingängen und zwei Ausgängen:



Wenn man den Eingang x_3 **als Übertrag** c_{i-1} der Addition der vorhergehenden Stelle verwendet, kann man durch **Komposition von VAs** ein Rechenwerk aufbauen, das Binärzahlen korrekt addiert.

4-Bit-Addierer

Nun zeigen wir den Aufbau eines Rechenwerks für die Addition von 4-Bit-Binärzahlen:



4-Bit-Addierer

- Analoge Vorgangsweise **für weitere Rechenoperationen** (man kann alles in Hardware realisieren).
- Praktisches Problem dabei: Die auftretende Komplexität der Schaltungen, die bei heutigen Prozessoren nur automatisch mit CAD (Computer Aided Design) Techniken (Programmen) bewältigt werden kann.

Schaltungsminimierung

Schaltungsminimierung

Ziel:
Minimierung der Anzahl logischer Bausteine.

Bevor wir uns mit der Schaltungsminimierung beschäftigen, werden wir zunächst die alternativen Darstellungsformen von Schaltfunktionen betrachten.

(Bisher: Die Schaltfunktionen in Form logischer Schaltungen mit speziellen logischen Bausteinen.)

Andere Möglichkeiten:

- Gebietsdarstellung
- Karnaugh-Diagramm

Gebietsdarstellung

Der **Darstellung von Mengen entlehnt** und kann für die Schaltfunktionen von **bis zu drei Variablen** verwendet werden.

Es besteht aus einem **Rechteck** (das **Gesamtgebiet der Schaltfunktionen**), für **jede Variable** gibt es **einen Kreis** im Rechteck. Im Inneren des Kreises ist die Variable wahr und außerhalb falsch. Die Kreise müssen sich jeweils gegenseitig **überlappen**.

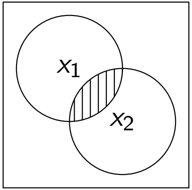
Zwei Typen von **Flächen**: Schraffiert und unschraffiert.

Die **schraffierten** Flächen entsprechen dem Wahrheitswert W der Aussage, die **unschraffierten** dem Wahrheitswert F.

Gebietsdarstellung

Beispiel [Gebietsdarstellung]

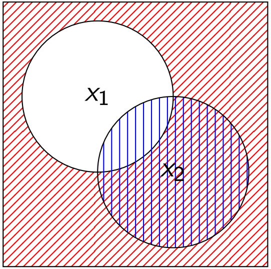
Die Konjunktion zweier Variablen x_1, x_2:



Die **Konjunktion** $y = x_1 \cdot x_2$ wird als gemeinsamer Bereich (**Schnitt**) von x_1 und x_2 **dargestellt** (also $\cap$ (Schnitt) entspricht der Konjunktion (&, $\wedge$, $\cdot$)).
Da die Konjunktion genau dann wahr ist, wenn beide Variablen wahr sind, entspricht dies der schraffierten Fläche innerhalb der Kreise.
Die **Disjunktion** $y = x_1 + x_2$ entspricht der **Vereinigung** von x_1, x_2, weil dort mindestens eine Variable wahr ist.
Das **Komplement** entspricht der **Negation**.

Gebietsdarstellung

Beispiel [Darstellung von $y = \overline{x_1} + x_2$]



 : $\overline{x_1}$

 : x_2

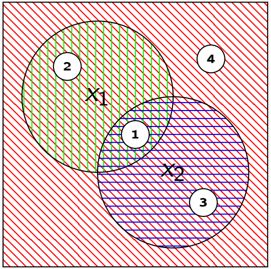
 $\oplus$  $\oplus$  : $\overline{x_1} + x_2$: **wahr**

Anmerkung: Wahrheitswert **falsch** (also $\overline{y}$) gilt nur für das unschraffierte Gebiet:

$$x_1 - x_1 x_2 = x_1(1 - x_2) = x_1\overline{x_2} = \overline{(\overline{x_1} + x2)} = \overline{y}.$$

Gebietsdarstellung

Die Gebietsdarstellung von zwei Variablen lässt genau vier Flächen unterscheiden:

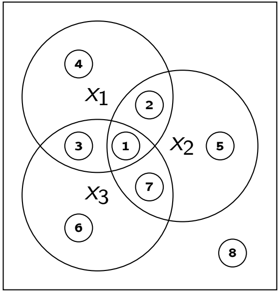


Fläche	Minterm
1	$x_1 x_2 = x_1 \cap x_2$
2	$x_1 \overline{x_2} = x_1 \cap \overline{x_2}$
3	$\overline{x_1} x_2 = \overline{x_1} \cap x_2$
4	$\overline{x_1}\,\overline{x_2} = \overline{x_1} \cap \overline{x_2}$

Diese Flächen entsprechen allen möglichen Mintermen der Variablen x_1, $\overline{x_1}$, x_2, $\overline{x_2}$.

Gebietsdarstellung

Die Gebietsdarstellung von **drei Variablen**: **8 Flächen** entsprechen $2^3 = 8$ **Mintermen**.



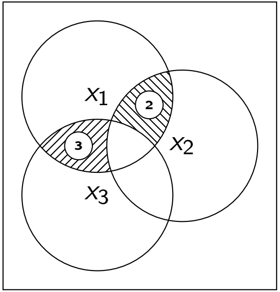
1 : $x_1\, x_2\, x_3$ nur dann wahr, wenn alle drei Variablen wahr sind, d. h. im Gebiet, wo sich alle drei Kreise überlappen

8 : $\overline{x_1}\,\overline{x_2}\,\overline{x_3}$ wahr (also schraffiert), wenn alle drei Variablen $\overline{x_i} = \text{W} \Rightarrow x_i = \text{F}$, d. h. im gemeinsamen Gebiet außerhalb der Kreise

Gebietsdarstellung

Beispiel

Wie lautet die Schaltfunktion, die dieser Gebietsdarstellung entspricht?



Schraffierte Flächen:
Vereinigung der Flächen 2 3

2 : $x_1 x_2 \overline{x_3}$ 3 : $x_1 \overline{x_2} x_3$

Also: Die Schaltfunktion ist: $x_1 x_2 \overline{x_3} + x_1 \overline{x_2} x_3$.

Gebietsdarstellung

Minterm für die Fläche F_j, $j = 1, \ldots, 2^n$ ($n = 1, 2, 3$) (n: Anzahl der Variablen)

Die Regel:
Wenn die Fläche F_j außerhalb des Kreises x_i liegt, dann tritt diese Variable als $\overline{x_i}$ auf.

Wenn F_j innerhalb von Kreis x_i liegt, dann wird sie als x_i genommen ($i = 1, 2, 3$).

Wenn die schraffierte Fläche aus mehreren Teilflächen besteht, wird diese als Disjunktion (Addition) der Teilflächen dargestellt.

Gebietsdarstellung

Über die Gebietsdarstellung können Schaltfunktionen vereinfacht werden, indem man versucht die schraffierten Gebiete alternativ zu beschreiben.

Diese Darstellung hat eher anschaulichen Charakter, weil die Anzahl der Variablen in der Schaltfunktion auf **drei** limitiert ist.

Karnaugh-Diagramm (K-Diagramm)

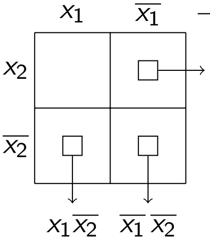
Ziel:
Zuordnung von Teilflächen zu Mintermen, um dadurch eine Vereinfachung der Schaltfunktion zu erreichen. Die Teilflächen sind in einem Rechteck geordnet.

- Die einzelnen Minterme werden hier durch **gleich große Teilflächen** repräsentiert.
- **Konstruktion**:
  - Alle Variablen sind in **negierter** und **nicht-negierter** Form an den Zeilen und Spalten des Diagramms aufgetragen.
  - Die **Kreuzungsflächen** repräsentieren dann jeweils einen **Minterm**.
  - Wesentliche Eigenschaft dabei: Die Minterme benachbarter Flächen unterscheiden sich (vertikal und horizontal) nur in einer Variablen durch Negation.
    (Das gilt auch für die erste und die letzte Fläche jeder Spalte/Zeile.)

Karnaugh-Diagramm (K-Diagramm)

Beispiel [Konstruktion von K-Diagrammen]

$n = 2$: $x_1, x_2 \to 2^2$ Teilflächen $\to$ 4 Minterme $\to 2 \times 2$-Rechteck



$\to x_1, x_2$ (negiert und nicht-negiert) aufgetragen

(Teilfläche für den Minterm $\overline{x_1}x_2$)

Die benachbarten Teilflächen unterscheiden sich nur in einer Variable durch Negation (z. B. $x_1\overline{x_2}$, $\overline{x_1}\,\overline{x_2}$).

Karnaugh-Diagramm (K-Diagramm)

Beispiel [Konstruktion von K-Diagrammen]

$n = 3$:

Variablen $\begin{array}{c} x_1, x_2, x_3 \\ \overline{x_1}, \overline{x_2}, \overline{x_3} \end{array}$ $\to 2^3$ Teilflächen $\to$ 8 Minterme $\to$ 2×4- oder 4×2-Rechteck

	x_1	$\overline{x_1}$
$x_2\overline{x_3}$	$\circ$	$\times$
x_2x_3	$\circ$	
$\overline{x_2}x_3$		
$\overline{x_2}\,\overline{x_3}$		$\times$

oder

	$\overline{x_1}\,\overline{x_2}$	$\overline{x_1}x_2$	x_1x_2	$x_1\overline{x_2}$
$\overline{x_3}$				
x_3	$\diamond$			$\diamond$

Karnaugh-Diagramm (K-Diagramm)

Beispiel [Konstruktion von K-Diagrammen (Fortsetzung)]

Es sind beliebige Permutationen von Indizes erlaubt, welche die Nachbarschaftsbedingung erfüllen.

Benachbarte Flächen:

$$\left.\begin{array}{lllll} \circ & \circ & : & x_1 x_2 \overline{x_3} & x_1 x_2 x_3 \\ \times & \times & : & \overline{x_1} x_2 \overline{x_3} & \overline{x_1}\,\overline{x_2}\,\overline{x_3} \\ \diamond & \diamond & : & \overline{x_1}\,\overline{x_2} x_3 & x_1 \overline{x_2} x_3 \end{array}\right\} \text{Unterschied jeweils in einer Variable}$$

Karnaugh-Diagramm (K-Diagramm)

Beispiel [Konstruktion von K-Diagrammen]

$n = 4$:

$\begin{array}{c} x_1, x_2, x_3, x_4 \\ \overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4} \end{array} \rightarrow 2^4$ Teilflächen $\rightarrow$ 16 Minterme $\rightarrow$ 4×4-Rechteck (es geht auch z. B. 2×8)

K_4 :

	$\overline{x_1}\,\overline{x_2}$	$\overline{x_1}x_2$	x_1x_2	$x_1\overline{x_2}$
$\overline{x_3}\,\overline{x_4}$	$\circ$			
$\overline{x_3}x_4$	$\times$			$\times$
x_3x_4				
$x_3\overline{x_4}$	$\circ$			

$\circ\ \circ$: $\overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4}\quad \overline{x_1}\,\overline{x_2}x_3\overline{x_4}$

$\times\ \times$: $\overline{x_1}\,\overline{x_2}\,\overline{x_3}x_4\quad x_1\overline{x_2}\,\overline{x_3}x_4$

Permutationen möglich!

Karnaugh-Diagramm (K-Diagramm)

Erklärung für

- Die Variablen in 2 Gruppen unterteilen: $\{x_1, x_2\}$, $\{x_3, x_4\}$
- In jeder Gruppe aus der negierten und nicht-negierten Variablen eine gereihte Folge von 2-Termen generieren, welche sich gerade in einer Position unterscheiden (ähnlich wie beim Gray-Code).
  - Aus $\{x_1, x_2\}$: $\overline{x_1}\,\overline{x_2}, \overline{x_1}x_2, x_1x_2, x_1\overline{x_2}$
    oder (**Permutation**): $x_1x_2, \overline{x_1}x_2, \overline{x_1}\,\overline{x_2}, x_1\overline{x_2}$
  - Aus $\{x_3, x_4\}$: $\overline{x_3}\,\overline{x_4}, \overline{x_3}x_4, x_3x_4, x_3\overline{x_4}$
    oder (**Permutation**): $x_3x_4, \overline{x_3}x_4, \overline{x_3}\,\overline{x_4}, x_3\overline{x_4}$

Karnaugh-Diagramm (K-Diagramm)

- Diese Reihenfolge an die **orthogonal** gelegene Seiten des 4×4-Diagramms platzieren

	$\overline{x_1}\,\overline{x_2}$	$\overline{x_1}x_2$	x_1x_2	$x_1\overline{x_2}$
$\overline{x_3}\,\overline{x_4}$				
$\overline{x_3}x_4$				
x_3x_4				
$x_3\overline{x_4}$				

Permutation

	x_1x_2	$\overline{x_1}x_2$	$\overline{x_1}\,\overline{x_2}$	$x_1\overline{x_2}$
x_3x_4				
$\overline{x_3}x_4$				
$\overline{x_3}\,\overline{x_4}$				
$x_3\overline{x_4}$				

Karnaugh-Diagramm (K-Diagramm)

Bemerkung: Wir können die Seiten des Diagramms auch mit 1-Term-Reihenfolgen zu belegen. Die Variablen $x_2, \overline{x_2}$, bzw. $x_4, \overline{x_4}$ werden auf die gegenüberliegenden Seiten gestellt.

	$\overline{x_1}$	$\overline{x_1}$	x_1	x_1	
$\overline{x_3}$					$\overline{x_4}$
$\overline{x_3}$					x_4
x_3					x_4
x_3					$\overline{x_4}$
	$\overline{x_2}$	x_2	x_2	$\overline{x_2}$	

Permutation

	x_1	$\overline{x_1}$	$\overline{x_1}$	x_1	
x_3					x_4
$\overline{x_3}$					x_4
$\overline{x_3}$					$\overline{x_4}$
x_3					$\overline{x_4}$
	x_2	x_2	$\overline{x_2}$	$\overline{x_2}$	

Karnaugh-Diagramm (K-Diagramm)

$n = 5$: $2^5 = 32$ Teilflächen

Ab einer Anzahl von 5 Variablen kann man entweder zu einer dreidimensionalen Darstellung übergehen, oder eine „hierarchische“ Struktur aus K-Diagrammen niedrigerer Dimensionen aufzubauen.

x_5	$\overline{x_5}$
K_4	$\tilde{K}_4$

$\tilde{K}_4$: Spaltenindizes von K_4 in **„reflected order“**, damit die Nachbarschaftsbedingung nicht verletzt wird.

Karnaugh-Diagramm (K-Diagramm)

x_5 | $\overline{x_5}$

	$\overline{x_1}\,\overline{x_2}$	$\overline{x_1}x_2$	x_1x_2	$x_1\overline{x_2}$	$x_1\overline{x_2}$	x_1x_2	$\overline{x_1}x_2$	$\overline{x_1}\,\overline{x_2}$
$\overline{x_3}\,\overline{x_4}$								
$\overline{x_3}x_4$								
x_3x_4								
$x_3\overline{x_4}$								

: Spiegelung (reflected order)

Karnaugh-Diagramm (K-Diagramm)

Bei der „vertikalen“ Konstruktion im zweiten (unteren K_4-Block) müssen die Zeilenindizes von K_4 in „reflected order“ aufgetragen werden:

x_5	K_4
$\overline{x_5}$	$\hat{K}_4$

Karnaugh-Diagramm (K-Diagramm)

$n = 6$: $2^6 = 64$ Teilflächen

	x_5	$\overline{x_5}$
x_6	K_4	$\tilde{K}_4$
$\overline{x_6}$	$\hat{K}_4$	$\hat{\tilde{K}}_4$

$\hat{\tilde{K}}_4$: Spalten- **und** Zeilenindizes von K_4 „reflected“

Karnaugh-Diagramm (K-Diagramm)

	x_5	$\overline{x_5}$
x_6	K_4	$\tilde{K}_4$
$\overline{x_6}$	$\hat{K}_4$	$\hat{\tilde{K}}_4$

		x_5				$\overline{x_5}$			
		$\overline{x_1}\,\overline{x_2}$	$\overline{x_1}x_2$	x_1x_2	$x_1\overline{x_2}$	$x_1\overline{x_2}$	x_1x_2	$\overline{x_1}x_2$	$\overline{x_1}\,\overline{x_2}$
x_6	$\overline{x_3}\,\overline{x_4}$								
	$\overline{x_3}x_4$								
	x_3x_4				○	○			
	$x_3\overline{x_4}$			□					
$\overline{x_6}$	$x_3\overline{x_4}$			□					
	x_3x_4								
	$\overline{x_3}x_4$	×							×
	$\overline{x_3}\,\overline{x_4}$								

Karnaugh-Diagramm (K-Diagramm)

○○	:	$x_6\, x_3\, x_4\, \underline{x_5}\, x_1\, \overline{x_2}$	$x_6\, x_3\, x_4\, \underline{\overline{x_5}}\, x_1\, \overline{x_2}$
××	:	$\overline{x_6}\, \overline{x_3}\, x_4\, \underline{x_5}\, \overline{x_1}\, \overline{x_2}$	$\overline{x_6}\, \overline{x_3}\, x_4\, \underline{\overline{x_5}}\, \overline{x_1}\, \overline{x_2}$
□□	:	$\underline{x_6}\, x_3\, \overline{x_4}\, x_5\, x_1\, x_2$	$\underline{\overline{x_6}}\, x_3\, \overline{x_4}\, x_5\, x_1\, x_2$

Unterschied jeweils nur bei __.

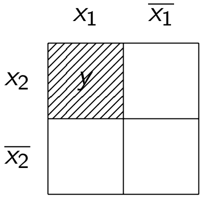
Für jedes Paar benachbarter Teilflächen Unterschied nur in einer Variable.
Ohne Spiegelung: z. B. ○ ○ hätten Unterschied in zwei Variablen ($x_5\, x_1 \quad \overline{x_5}\, \overline{x_1}$).

Karnaugh-Diagramm (K-Diagramm)

- Darstellung von Schaltfunktionen
- Die Flächen jener Minterme, die die Schaltfunktion implizieren, werden schraffiert.

Beispiel [Darstellung der Konjunktion]

Angenommen $n = 2; y = x_1 x_2$



Karnaugh-Diagramm (K-Diagramm)

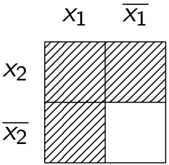
- Die Schaltfunktion ist durch **die Vereinigung der schraffierten Teilflächen** gekennzeichnet.
- Ist die **gesamte** Zeile oder Spalte **schraffiert**, so heißt das, dass **alle Minterme**, die durch die einzelnen Flächen repräsentiert werden, **zu der Variablen**, die die Zeile oder Spalte beschreibt, **zusammengefasst werden können**.
  - **Grundidee** zur Vereinfachung von Schaltungen mittels Karnaugh-Diagramm.

Karnaugh-Diagramm (K-Diagramm)

Beispiel [Darstellung und Vereinfachung von Schaltfunktionen mit Hilfe von K-Diagrammen]

$y = x_1 x_2 + x_2 \overline{x_1} + x_1 \overline{x_2}$

Darstellung:



Vereinfachung: Die Teilflächen (Minterme) $x_1 x_2$, $x_2 \overline{x_1}$ der ersten Zeile werden zur Variable x_2 zusammengefasst. Ähnlicherweise wird die erste Spalte durch x_1 repräsentiert. Also: $y = x_1 + x_2$ (Disjunktion von x_1, x_2).

Erklärung:

$y = x_1 x_2 + x_2 \overline{x_1} + x_1 \overline{x_2} = x_1 x_2 + x_1 x_2 + x_2 \overline{x_1} + x_1 \overline{x_2} = (x_1 x_2 + x_1 \overline{x_2}) + (x_1 x_2 + x_2 \overline{x_1})$
$= x_1 (x_2 + \overline{x_2}) + x_2 (x_1 + \overline{x_1}) = x_1 + x_2$

$\rightarrow$ Das obige K-Diagramm repräsentiert die Disjunktion.

Karnaugh-Diagramm (K-Diagramm)

- $y = (x_1 + x_2)(x_1 + x_3) + (x_1 x_2 x_3)$

  Darstellung:

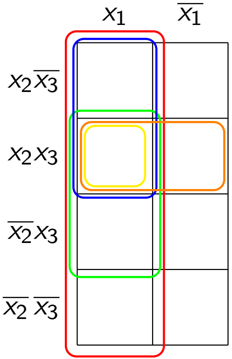
	x_1	$\overline{x_1}$
$x_2\overline{x_3}$		
$x_2 x_3$		
$\overline{x_2} x_3$		
$\overline{x_2}\,\overline{x_3}$		

$$
\begin{aligned}
y &= x_1 x_1 + x_1 x_3 + x_2 x_1 + x_2 x_3 + x_1 x_2 x_3 \\
&\qquad (\text{weil } x_1 \cdot x_1 = x_1, 1 = x_2 + \overline{x_2}, 1 = x_3 + \overline{x_3}, 1 = x_1 + \overline{x_1}) \\
&= x_1 + x_1(x_2 + \overline{x_2})x_3 + x_2 x_1 (x_3 + \overline{x_3}) \\
&\quad + x_2 x_3 (x_1 + \overline{x_1}) + x_1 x_2 x_3 \\
&= \boxed{x_1} + \boxed{(x_1 x_2 x_3 + x_1 \overline{x_2} x_3)} + \boxed{(x_1 x_2 x_3 + x_1 x_2 \overline{x_3})} \\
&\quad + \boxed{(x_1 x_2 x_3 + \overline{x_1} x_2 x_3)} + \boxed{x_1 x_2 x_3}
\end{aligned}
$$

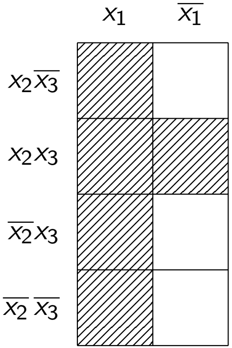
Karnaugh-Diagramm (K-Diagramm)

$$y = \boxed{x_1} + \boxed{(x_1x_2x_3 + x_1\overline{x_2}x_3)} + \boxed{(x_1x_2x_3 + x_1x_2\overline{x_3})} + \boxed{(x_1x_2x_3 + \overline{x_1}x_2x_3)} + \boxed{x_1x_2x_3}$$

Die Vereinigung aller Teilflächen ist das K-Diagramm für y



$\rightarrow$



$\rightarrow$ Die Vereinfachung

$x_1 + x_2x_3$

$\downarrow$ 1. Spalte, $\downarrow$ 2. Zeile

zusammengefasst

Vereinfachung mit K-Diagrammen

- In die Felder des Diagramms werden die entsprechenden Werte der Wahrheitstabelle eingetragen
- Im Diagramm werden die Blöcke identifiziert, die aus **benachbarten 1-Elementen** bestehen.
  (Wenn **ein** 1-er isoliert da steht, wird er einen Konjunktionsterm darstellen, wo alle Variablen (negiert oder nicht-negiert) auftreten.)

Vereinfachung mit K-Diagrammen

Beispiel ($n = 3$)

x_1	x_2	x_3	Wert
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

K_3:

	$\overline{x_1}\,\overline{x_2}$	$x_1\overline{x_2}$	x_1x_2	$\overline{x_1}x_2$
$\overline{x_3}$	0	1 [1]	1	1 [2]
x_3	1 [3]	1	0	1 [3]

0 für $x_1x_2x_3$, d. h. für $x_1 = x_2 = x_3 = 1$.

Zusammenfassung der 1-er Blöcke: (1-3):
Block 1: $x_1\overline{x_2}$ (weil x_3, $\overline{x_3}$ auftreten)
Block 2: $x_2\overline{x_3}$ (weil x_1, $\overline{x_1}$ auftreten)
Block 3: $\overline{x_1}x_3$ (weil x_2, $\overline{x_2}$ beinhaltet)

$\rightarrow$ vereinfachte Form: $x_1\overline{x_2} + x_2\overline{x_3} + \overline{x_1}x_3$

Hier: Keine isolierten 1-er in der Tabelle.

Vereinfachung mit K-Diagrammen

Die Darstellung mit K-Diagramm ist schon für **sechs** Variablen (relativ) unübersichtlich.

- Grund für die Präsentierung eines analytischen Verfahrens, das auch für eine größere Anzahl von Variablen geeignet ist.

Das Verfahren von Quine-McCluskey

Ziel:
Vereinfachung einer Schaltfunktion die in DNF vorliegt.
(**Vereinfachung**: Einzelne Variablen in speziellen Konjunktionstermen oder ganze Konjunktionsterme **weglassen**.)

Formalisierung der Vereinfachung (einer DNF)

Bezeichnung:

- Φ: DNF
- v_Φ: Die Gesamtzahl der negierten oder nicht-negierten Variablen in Φ (**jedes Auftreten** wird gezählt)
- k_Φ: Die Anzahl der Konjunktionsterme von Φ

Definition

Sind zwei DNFs Φ und Ψ gegeben, so heißt Φ einfacher als Ψ genau dann, wenn $v_\Phi \leq v_\Psi$ und $k_\Phi \leq k_\Psi$ und mindestens eine der Ungleichungen streng ($<$) gilt.

Formalisierung der Vereinfachung (einer DNF)

Definition

Die DNF Φ einer Schaltfunktion heißt disjunktive **M**inimalform (D**M**F) genau dann, wenn keine einfachere logisch äquivalente Schaltfunktion existiert.

$\rightarrow$ Die einzelnen Konjunktionsterme (Disjunktionsglieder) können nicht weiter vereinfacht werden.
Vermutung: Die Disjunktionsglieder einer DMF sind Primimplikanten.

Theorem

Jede ***DMF*** *Φ der Schaltfunktion y ist eine Disjunktion von einem oder mehreren Primimplikanten.*

$\rightarrow$ Um eine DMF aus einer DNF zu finden, muss man alle Primimplikanten der DNF ermitteln.

Die Ermittlung von Primimplikanten mit K-Diagrammen

Primimplikanten: Wir suchen möglichst große Blöcke der Größe 2^k, die sich in der schraffierten Fläche befinden.

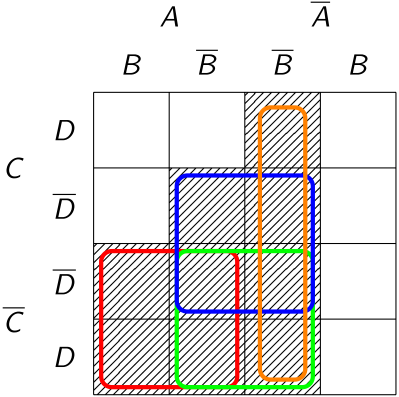
Wesentliche Primimplikanten: Entsprechen der minimalen Anzahl der Blöcke, welche die schraffierte Fläche überdecken.

Bemerkung: Diese **minimale** Überdeckung muss nicht die einzige sein.

- Es gibt mehrere DMFs (also muss DMF nicht eindeutig sein).
- Die DMF ist eindeutig, wenn eine einzige minimale Überdeckung der schraffierten Fläche des K-Diagramms existiert.

Die Ermittlung von Primimplikanten mit K-Diagrammen

Beispiel 1



Blöcke Größe $2^2 = 4$:

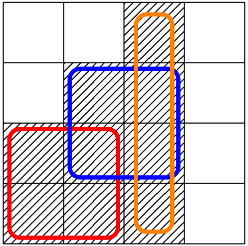
$$\left.\begin{aligned} P1 &= \overline{A}\,\overline{B} \\ P2 &= \overline{B}\,\overline{D} \\ P3 &= \overline{B}\,\overline{C} \\ P4 &= A\,\overline{C} \end{aligned}\right\}$$

Alle möglichen Blöcke Größe 4 also Primimplikanten

Die Ermittlung von Primimplikanten mit K-Diagrammen

Beispiel 1 (Fortsetzung)

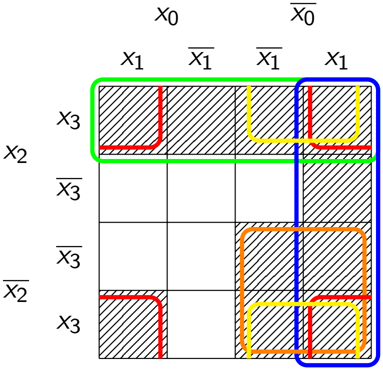
Die minimale Überdeckung ist möglich mit $P1, P2, P4$: Das sind die wesentlichen Primimplikanten.



Diese minimale Überdeckung ist die einzige.
$\rightarrow$ DMF: $P1 \vee P2 \vee P4$ ist eindeutig.

Die Ermittlung von Primimplikanten mit K-Diagrammen

Beispiel 2



Blöcke Größe $2^2 = 4$:

$\boxed{P1} = x_1 x_3$
$\boxed{P2} = x_2 x_3$
$\boxed{P3} = \overline{x_0} x_1$
$\boxed{P4} = \overline{x_0}\,\overline{x_2}$
$\boxed{P5} = \overline{x_0} x_3$

Die Primimplikanten sind also $P1$, $P2$, $P3$, $P4$ und $P5$.

Die Ermittlung von Primimplikanten mit K-Diagrammen

Beispiel 2 (Fortsetzung)

Bei der Überdeckung (minimal) ist die Fläche $P5$ überflüssig.

$\rightarrow$ Die wesentlichen Primimplikanten sind $P1$, $P2$, $P3$ und $P4$.

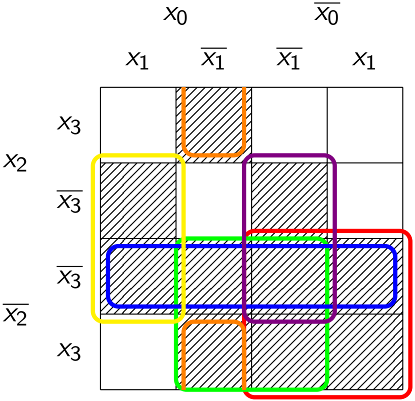
$\rightarrow$ DMF$= P1 \vee P2 \vee P3 \vee P4 = x_1x_3 + x_2x_3 + \overline{x_0}x_1 + \overline{x_0}\,\overline{x_2}$

Die Überdeckung mit Teilflächen der Größe 4 ist mit $P1$, $P2$, $P3$ und $P4$ eindeutig.

$\rightarrow$ DMF ist eindeutig.

Die Ermittlung von Primimplikanten mit K-Diagrammen

Beispiel 3



Blöcke Größe 4:

$P1 = \overline{x_0}\,\overline{x_2}$

$P2 = \overline{x_1}\,\overline{x_2}$

$P3 = \overline{x_2}\,\overline{x_3}$

Blöcke Größe 2:

$P4 = x_0\overline{x_1}x_3$

$P5 = x_0x_1\overline{x_3}$

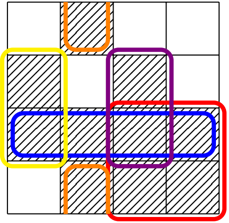
$P6 = \overline{x_0}\,\overline{x_1}\,\overline{x_3}$

Also: Primimplikanten sind: $P1, P2, \ldots, P6$

Die Ermittlung von Primimplikanten mit K-Diagrammen

Beispiel 3 (Fortsetzung)

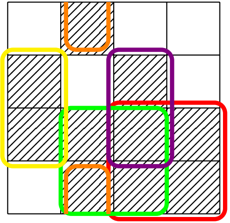
1. minimale Überdeckung



mit $P1, P3, P4, P5, P6$

Das sind die wesentlichen Primimplikanten für DMF_1:
$P1 \vee P3 \vee P4 \vee P5 \vee P6$

2. minimale Überdeckung



mit $P1, P2, P4, P5, P6$

Das sind die wesentlichen Primimplikanten für DMF_2:
$P1 \vee P2 \vee P4 \vee P5 \vee P6$

Hier gibt es zwei minimale Überdeckungen $\rightarrow$ die DMF ist nicht eindeutig

Entwurf einer logischen Schaltung

Beispiel [Entwurf einer (kleinen) logischen Schaltung: Von der Festlegung der DNF bis zur Minimierung der Schaltung]

Entwurf einer Schaltung zur Erkennung von Pseudotetraden in Zahlen, die im Binary Coded Decimal (BCD)-Format vorliegen.

(Die BCD-Darstellung von Zahlen codiert jede Ziffer einer Dezimalzahl als binäre 4-Bit-Zahl.)

$0 \to 0000, 1 \to 0001, \ldots, 9 \to 1001$

z. B. $129_{(10)} \to 000100101001_{(BCD)}$

Weil 4-stellige Binärzahlen 16 Ziffern repräsentieren können (und wir im Dezimalsystem nur 10 haben), gibt es auch 4-Tupel (genau 6), die keine Dezimalzahl repräsentieren. Diese „sinnlosen“ 4-Bit-Wörter heißen Pseudotetraden (4-Bit-Wort auch als Nibble bezeichnet).

Entwurf einer logischen Schaltung

Beispiel (Fortsetzung)

Der BCD-Code wird z. B. zur Steuerung von Digitalanzeigen verwendet.
(Da man zumeist Dezimalzahlen anzeigen will und sich die Ziffern aus einem BCD-Code einfach extrahieren lassen.)

Entwurf einer logischen Schaltung

Beispiel (Fortsetzung)

Die Aufstellung einer Wahrheitstafel, die einer korrekten BCD-Zahl $x_3x_2x_1x_0$ den Wert 0 und einer Pseudotetrade den Wert 1 zuweist:

	x_3	x_2	x_1	x_0	y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0

	x_3	x_2	x_1	x_0	y
8	1	0	0	0	0
9	1	0	0	1	0
Pseudotetrade	1	0	1	0	1
Pseudotetrade	1	0	1	1	1
Pseudotetrade	1	1	0	0	1
Pseudotetrade	1	1	0	1	1
Pseudotetrade	1	1	1	0	1
Pseudotetrade	1	1	1	1	1

Entwurf einer logischen Schaltung

Beispiel (Fortsetzung)

Aus der Tabelle leiten wir die vollständige DNF ab (die Terme mit $y = 1$ werden betrachtet):

$$DNF_{x_0,x_1,x_2,x_3}: \quad x_3\overline{x_2}x_1\overline{x_0} + x_3\overline{x_2}x_1x_0 + x_3x_2\overline{x_1}\,\overline{x_0} + x_3x_2\overline{x_1}x_0 + x_3x_2x_1\overline{x_0} + x_3x_2x_1x_0$$

Nun bestimmen wir alle Primimplikanten (der vollständigen DNF) mit dem Verfahren von Quine-McCluskey.

Die Minterme in Klassen K_0, K_1, K_2 (entsprechend der Anzahl ihrer negierten Terme) einteilen und solange zusammenfassen und bei Verwendung abhaken, bis keine Vereinfachung mehr möglich ist.

Entwurf einer logischen Schaltung

Beispiel (Fortsetzung)

$$
\begin{array}{llll}
K_0: & x_3x_2x_1x_0\ \checkmark & x_3x_2x_1\ \checkmark & x_3x_2 \\
 & & x_3x_2x_0\ \checkmark & x_3x_1 \\
 & & x_3x_1x_0\ \checkmark & x_3x_2 \\
 & & & x_3x_1 \\
K_1: & x_3x_2x_1\overline{x_0}\ \checkmark & x_3x_2\overline{x_0}\ \checkmark & \\
 & x_3x_2\overline{x_1}x_0\ \checkmark & x_3x_1\overline{x_0}\ \checkmark & \\
 & x_3\overline{x_2}x_1x_0\ \checkmark & x_3x_2\overline{x_1}\ \checkmark & \\
 & & x_3\overline{x_2}x_1\ \checkmark & \\
K_2: & x_3x_2\overline{x_1}\,\overline{x_0}\ \checkmark & & \\
 & x_3\overline{x_2}x_1\overline{x_0}\ \checkmark & &
\end{array}
$$

(Idempotenz)

$\rightarrow x_3x_1$

x_3x_2

Entwurf einer logischen Schaltung

Beispiel (Fortsetzung)

Die Primimplikanten: x_3x_1, x_3x_2

$\rightarrow$ Die DMF ist $x_3x_1 + x_3x_2 = x_3(x_1 + x_2)$

Die beiden gefundenen Primimplikanten sind wesentlich (wir brauchen beide für die DMF – keiner kann weggelassen werden um zur DMF zu gelangen).

(Es gibt auch solche, die nicht wesentlich sind $\rightarrow$ wir brauchen ein Verfahren zur Bestimmung wesentlicher Primimplikanten)

Wesentliche Primimplikanten

Theorem

Jeder Minterm der vollständigen DNF Φ enthält mindestens einen Konjunktionsterm der DMF zu Φ.

$\rightarrow$ Ein Primimplikant, der nur in einem einzigen Minterm enthalten ist, muss wesentlich sein (da ein Weglassen dieses Primimplikanten zur Verletzung des obigen Theorems führt).

Wesentliche Primimplikanten

→ Das Verfahren zur Bestimmung der wesentlichen Primimplikanten:

1. Wir weisen **jedem Minterm** φ der vollständigen DNF Φ **eine Spalte** in einer Tabelle zu. **In jede Zeile** dieser Tabelle tragen wir **einen Primimplikanten von** Φ ein.
2. **Enthält ein Minterm einen Primimplikanten**, so kennzeichnen wir die entsprechende Stelle in der Tabelle **mit einem Kreuz**.

Wesentliche Primimplikanten

3. Enthält eine Spalte **nur ein Kreuz** (wesentlicher Primimplikant), dann umgeben wir **das Kreuz** mit einem **Kreis**. **Alle anderen Kreuze der selben Zeile** umgeben wir mit einem **Quadrat. Existiert keine Spalte mit nur einem Kreuz**, dann umgeben wir **ein beliebiges Kreuz einer Spalte** mit einem **Kreis**.
4. Erscheint dann **in jeder Spalte ein Kreis oder ein Quadrat**, ist das Verfahren **beendet**.[⊛]
   Es verbleiben jene Primimplikanten, in deren Zeile **kein Kreis auftritt**, als **nicht wesentlich**. Diese bleiben dann für die DMF **unberücksichtigt**.

[⊛] Andernfalls wiederholen wir 3. für eine noch unberücksichtigte Spalte.

Wesentliche Primimplikanten

Beispiel [Wesentliche Primimplikanten]

Die vollständige DNF: $\Phi = x_1\overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3} + x_1x_2\overline{x_3} + x_1x_2x_3$

Die Primimplikanten von Φ sind x_1x_2, x_1x_3 und $x_2\overline{x_3}$ (Quine-McCluskey)

Aufstellen der Tabelle zur Bestimmung der wesentlichen Primimplikanten:

	$x_1\overline{x_2}x_3$	$\overline{x_1}x_2\overline{x_3}$	$x_1x_2\overline{x_3}$	$x_1x_2x_3$
x_1x_2			$\times$	$\times$
x_1x_3	$\times$			$\boxed{\times}$
$x_2\overline{x_3}$		$\times$	$\boxed{\times}$	

$\times$: 2. Schritt
$\bigcirc$, $\square$: 3. Schritt
$\smile$: 4. Schritt

Aus 4. Schritt $\rightarrow$ Primimplikant x_1x_2 ist **nicht wesentlich**.
Daher lautet die DMF $x_1x_3 + x_2\overline{x_3}$.

Wesentliche Primimplikanten

Mit dem Verfahren von Quine-McCluskey und der obigen Technik zur Auffindung der wesentlichen Primimplikanten ist es möglich, die **DMF** aus einer vollständigen **DNF** exakt zu bestimmen.

Beschränkung auf 10 Variablen, da der Aufwand zu hoch wird. Danach werden heuristische Methoden eingesetzt.

Eindeutigkeit der DMF

Fall $\circledast$ tritt auf, wenn in der Spalte kein $\bigcirc$ oder kein $\square$ ist (nach 4.), also wenn dort mindestens zwei $\times$ sind (wenn nur ein $\times$ dort wäre, dann würde dieses im ersten Durchlauf den $\bigcirc$ bekommen).
$\rightarrow$ Also müssen mindestens zwei $\times$ sein. Nach 3. umgeben wir ein beliebiges Kreuz dieser Spalte mit einem Kreis.
$\rightarrow$ Also ist die Anzahl der Möglichkeiten den Kreis zu machen gleich der Anzahl der $\times$ in der Spalte.
$\rightarrow$ Für **jede Wahl** des Kreises gibt es **eine Lösung** für die DMF.
$\rightarrow$ **Also ist die DMF nicht eindeutig, wenn Fall $\circledast$ auftritt.** (Der Fall, dass keine Spalte mit nur einem Kreuz existiert (siehe 3. Schritt) ist im Fall $\circledast$ inbegriffen, siehe Bemerkung).

$\rightarrow$ **Die DMF ist eindeutig, wenn der Algorithmus mit nur einem Durchlauf des 3. Schrittes endet.**

Beispiel

Beispiel 1 (siehe Folie 377)

9 Minterme

$\overline{A}\,\overline{B}\,C\,D$
$\overline{A}\,\overline{B}\,C\,\overline{D}$
$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$
$\overline{A}\,\overline{B}\,\overline{C}\,D$
$A\,\overline{B}\,C\,\overline{D}$
$A\,\overline{B}\,\overline{C}\,\overline{D}$
$A\,\overline{B}\,\overline{C}\,D$
$A\,B\,\overline{C}\,\overline{D}$
$A\,B\,\overline{C}\,D$

Quine-McCluskey
$\rightarrow$

4 Primimplikanten

$\overline{A}\,\overline{B}$	P1
$\overline{B}\,\overline{D}$	P2
$\overline{B}\,\overline{C}$	P3
$A\,\overline{C}$	P4

Beispiel

Beispiel 1 (Fortsetzung)

	$\overline{A}\,\overline{B}CD$	$\overline{A}\,\overline{B}C\overline{D}$	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$	$\overline{A}\,\overline{B}\,\overline{C}D$	$A\overline{B}C\overline{D}$	...
$\overline{A}\,\overline{B}$	$\times$	$\boxed{\times}$	$\boxed{\times}$	$\boxed{\times}$		...
$\overline{B}\,\overline{D}$		$\boxed{\times}$	$\boxed{\times}$		$\times$	...
$\overline{B}\,\overline{C}$			$\times$	$\times$		...
$A\overline{C}$						...

...	$A\overline{B}\,\overline{C}\,\overline{D}$	$A\overline{B}\,\overline{C}D$	$AB\overline{C}\,\overline{D}$	$AB\overline{C}D$
...				
...	$\boxed{\times}$			
...	$\times$	$\times$		
...	$\boxed{\times}$	$\boxed{\times}$	$\times$	$\times$

$\overline{B}\,\overline{C}$ nicht wesentlich
$\rightarrow DMF = P1 \vee P2 \vee P4$

(Übereinstimmung mit der einzigen Überdeckung mittels K-Diagramm, siehe Folie 378)

Beispiel

Beispiel 2 (siehe Folie 381)

10 Minterme

$x_0\, x_1\, x_2\, \overline{x_3}$ M1
$x_0\, x_1\, \overline{x_2}\, \overline{x_3}$ M2
$x_0\, \overline{x_1}\, \overline{x_2}\, \overline{x_3}$ M3
$x_0\, \overline{x_1}\, x_2\, x_3$ M4
$x_0\, \overline{x_1}\, \overline{x_2}\, x_3$ M5
$\overline{x_0}\, \overline{x_1}\, x_2\, \overline{x_3}$ M6
$\overline{x_0}\, \overline{x_1}\, \overline{x_2}\, \overline{x_3}$ M7
$\overline{x_0}\, \overline{x_1}\, \overline{x_2}\, x_3$ M8
$\overline{x_0}\, x_1\, \overline{x_2}\, \overline{x_3}$ M9
$\overline{x_0}\, x_1\, \overline{x_2}\, x_3$ M10

Quine-McCluskey
$\longrightarrow$

6 Primimplikanten

$\overline{x_0}\, \overline{x_2}$ P1
$\overline{x_1}\, \overline{x_2}$ P2
$\overline{x_2}\, \overline{x_3}$ P3
$x_0\, \overline{x_1}\, x_3$ P4
$x_0\, x_1\, \overline{x_3}$ P5
$\overline{x_0}\, \overline{x_1}\, \overline{x_3}$ P6

Beispiel

Beispiel 2 (Fortsetzung)

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
P1							$\boxed{\times}$	$\boxed{\times}$	$\boxed{\times}$	$\times$
P2			$\times$ 1		$\times$		$\times$	$\times$		
P3		$\times$	$\times$ 2				$\times$		$\times$	
P4				$\times$	$\boxed{\times}$					
P5	$\times$	$\boxed{\times}$								
P6						$\times$	$\boxed{\times}$			

Spalte M3: Fall $\circledast$, d. h. Wiederholung 3. Schritt $\rightarrow$ **DMF nicht eindeutig**

Beispiel

Beispiel 2 (Fortsetzung)

$\otimes^1$ *DMF_1*: Wir setzen Kreis in der Zeile P2
$\rightarrow$ **P3 nicht wesentlich**
$\rightarrow DMF_1 = P1 \vee P2 \vee P4 \vee P5 \vee P6$
$= \overline{x_0}\,\overline{x_2} + \overline{x_1}\,\overline{x_2} + x_0\overline{x_1}x_3 + x_0x_1\overline{x_3} + \overline{x_0}\,\overline{x_1}\,\overline{x_3}$
2. Überdeckung (Folie 382)

$\otimes^2$ *DMF_2*: Wir setzen Kreis in der Zeile P3
$\rightarrow$ **P2 nicht wesentlich**
$\rightarrow DMF_2 = P1 \vee P3 \vee P4 \vee P5 \vee P6$
$= \overline{x_0}\,\overline{x_2} + \overline{x_2}\,\overline{x_3} + x_0\overline{x_1}x_3 + x_0x_1\overline{x_3} + \overline{x_0}\,\overline{x_1}\,\overline{x_3}$
1. Überdeckung (Folie 382)

Graphisches Verfahren (für Pseudotetraden) mit Karnaugh-Diagrammen

Ziel:
Schaltungsminimierung mit Hilfe von K-Diagrammen

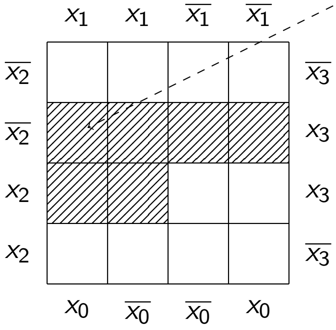
- Die folgende **Eigenschaft** wird ausgenutzt: Die benachbarten Teilflächen unterscheiden sich nur in **einer Variablen** eines **Minterms** (**benachbarte Teilflächen** in einem K-Diagramm sind jene Flächen, die **eine gemeinsame Seite** haben, wobei **das Ende** jeder Zeile/Spalte **mit dem Anfang verbunden** ist).

Graphisches Verfahren (für Pseudotetraden) mit Karnaugh-Diagrammen

Beispiel [Pseudotetraden]

$$\text{DNF: } \Phi = x_3\overline{x_2}x_1\overline{x_0} + x_3\overline{x_2}x_1x_0 + x_3x_2\overline{x_1}\,\overline{x_0} + x_3x_2\overline{x_1}x_0 + x_3x_2x_1\overline{x_0} + x_3x_2x_1x_0$$

- Eintragung in ein K-Diagramm ($n = 4$, $2^4 = 16 = 4 \times 4$ Teilflächen)



Also 6 Teilflächen (gekennzeichnet)

Graphisches Verfahren (für Pseudotetraden) mit Karnaugh-Diagrammen

Beispiel (Fortsetzung)

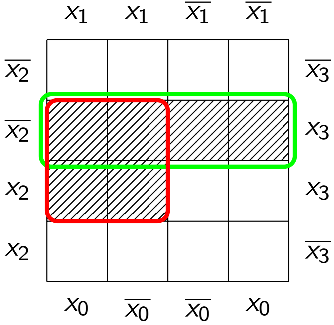
- Zusammenfassung benachbarter schraffierter Flächen zu Blöcken der Größe 2^k $(k = 1, 2, \ldots, n)$

  Jede Fläche muss **mindestens einmal** in einem Block vorkommen, die **Anzahl der Blöcke** soll **minimal** und die **Größe der Blöcke maximal** werden.

Graphisches Verfahren (für Pseudotetraden) mit Karnaugh-Diagrammen

Beispiel (Fortsetzung)

- Zusammenfassung



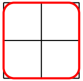
Bei uns $n = 4$
$\rightarrow$ Blöcke von Größe

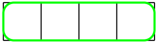
2^1,	2^2,	2^3,	2^4
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
Ja	Ja	Geht nicht	Geht nicht

(weil nur 6 Flächen)

Graphisches Verfahren (für Pseudotetraden) mit Karnaugh-Diagrammen

Beispiel (Fortsetzung)

Block 1: x_1  $x_3 \quad = x_1 x_3$

Block 2: x_2  $x_3 \quad = x_2 x_3$

Also zwei 4er-Blöcke benachbarter Flächen

$\rightarrow$ **DMF:** $x_1 x_3 + x_2 x_3$

Graphisches Verfahren (für Pseudotetraden) mit Karnaugh-Diagrammen

Beispiel (Fortsetzung)

Bemerkungen

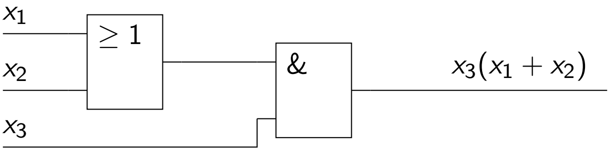
- Wir hätten auch andere Möglichkeiten (z. B. 3 2er-Blöcke), aber dann ist die Anzahl der Blöcke höher als in unserer DMF und je kleiner der Block, desto mehr Variablen.
- Die Anzahl der Variablen ist hier auf 6 beschränkt, da es darüber hinaus zu Darstellungsproblemen kommt.

Graphisches Verfahren (für Pseudotetraden) mit Karnaugh-Diagrammen

Beispiel (Fortsetzung)

Die logische Schaltung (zur DMF)

$y = x_1x_3 + x_2x_3 = (x_1 + x_2) \cdot x_3$



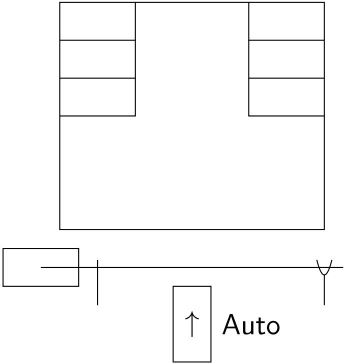
Also: Zur Erkennung von Pseudotetraden.

Schaltwerke

Schaltwerke

Schaltungen, bei denen Zeit eine Rolle spielt.

Beispiel [Automatische Parkplatzschranke]



Zwei Zustände der automatischen Schranke (die von der Zeit abhängig sind):

- Schranke soll aufgehen, wenn ein Auto kommt.
- Schranke soll geschlossen bleiben, wenn der Parkplatz voll ist.

Man braucht dazu ein Gedächtnis (Zähler), wie viele Autos hineingefahren sind.

Schaltwerke

Unterschied zu Schaltnetzen: Bei gleichem Input ergibt sich bei einem Schaltnetz immer der gleiche Output.

Also: Wenn ein Auto kommt, geht die Schranke auf (ohne Rücksicht auf die Zahl der Autos am Parkplatz $\rightarrow$ Chaos).

Schaltwerke sind Schaltnetze mit Rückkopplung (Feedback), wo es eine Möglichkeit gibt, sich an Aktionen in der Vergangenheit erinnern zu können.

$\rightarrow$ Dazu braucht man ein technisches System zur Speicherung der Aktionen
$\rightarrow$ Speicher (Memory).

Also: Für alle Formen der Rückkopplung ist Speicher nötig.

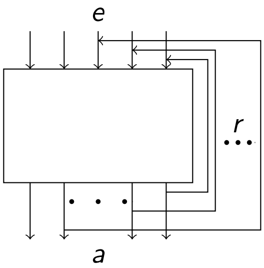
Schaltwerke

Definition

[Schaltwerk] Ein Schaltwerk ist ein gerichteter Graph, an dessen Knoten sich Schaltwerke (oder Schaltnetze) befinden.

Bemerkung: Der gerichtete Graph hier ist nicht azyklisch (wie bei einem Schaltnetz), sondern er enthält auch Zyklen, die für die Rückkopplung verantwortlich sind.

Das Schema eines Schaltwerks mit r rückgekoppelten Ausgängen:



Von insgesamt a Ausgängen sind r rückgekoppelt mit Eingängen.

Schaltwerke

- Problem bei der Rückkopplung:

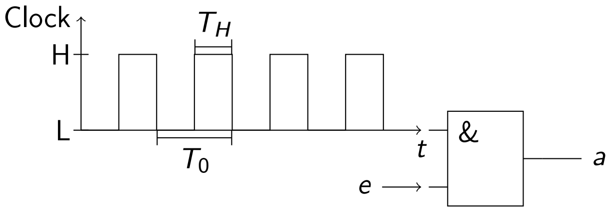
  z. B. Ausgang a_i gelangt schon an den Eingang e_j, obwohl der Ausgang a_k noch den alten Zustand an e_s überträgt.

  $\rightarrow$ Die Information am Eingang ist eine sinnlose Mischung aus **Gegenwart** und **Vergangenheit**: Dies würde zu einer **Fehlfunktion** des Schaltwerks führen.

Zeitverhalten

Für Schalt**werke** wird (zumeist) ein **Taktsignal** benötigt, das als **Zeitgeber** fungiert, sodass sich **die einzelnen Signale eindeutig gewissen Zeitspannen oder Zeitpunkten zuordnen lassen**. Diese Zuordnung heißt **Synchronisation**.

- Das **Taktsignal** (**Takt** oder **Clock**) wird aus nur zwei Zuständen aufgebaut (weil in digitalen Schaltungen logische Signale genau zwei Zustände haben).
  Ein **Rechtecksignal** erfüllt diese Anforderung. Damit kann ein Eingangssignal e mit einem Taktsignal t synchronisiert werden.



Zeitverhalten

Der Takt wird auf einen Eingang eines AND-Gatters gelegt. Ist der Taktpegel auf **L** (low), so entspricht dies einer logischen **0**. Wenn **H** (high) $\rightarrow$ logische **1**.

Das **Eingangssignal** e **erscheint nur dann am Ausgang** a**, wenn der Takt H ist.** (**Andernfalls** wird der Eingang **abgeschaltet**.)

Zeitverhalten

→ **Zustandssteuerung** (State Triggering), weil das Eingangssignal wird für eine gewisse Zeitspanne durch den **Zustand des Taktes** definiert.

- Einige Begriffe bezüglich Taktsignal:

  Die Periodendauer (eines periodischen Signals):

$$T_0 = \frac{1}{f} \quad [ns, \mu s, ms, s]$$

(f: Frequenz (in Hz))
Periodendauer $1ns \rightarrow f = 1GHz$
$1ms \rightarrow f = 1kHz$
(Frequenz, mit der eine Mikrowelle betrieben wird: $2.45GHz$)

Zeitverhalten

Definition

[Tastverhältnis]

$$\alpha_0 = \frac{T_H}{T_0} \qquad \begin{array}{ll} T_H\text{:} & \text{Zeitdauer des H-Takts} \\ T_0\text{:} & \text{Periodendauer.} \end{array}$$

Das Tastverhältnis gibt die relative Dauer des H-Pegels an.

α_0: Wichtig bei Schwankungen des Eingangssignals:

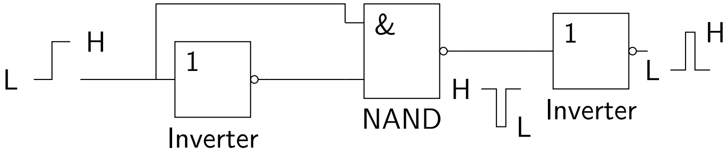
- Wenn das Taktsignal auf L gesetzt ist, kann das Eingangssignal beliebig schwanken, doch der Ausgang bleibt davon unberührt.
- In den H-Zeiten macht die Signalschwankung Schwierigkeiten.

Deswegen versucht man α_0 sehr klein zu machen, damit sich das Eingangssignal immer schärfer auf einen Zeitpunkt eingrenzen lässt.

Bei Bauteilen in digitalen Schaltungen gilt meist $\alpha_0 \approx 0.5$.

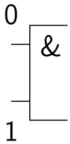
Zeitverhalten

Mit wenigen Gattern kann man aus einem solchen Takt einen sehr schmalen Impuls ableiten:



Eine 0 am Eingang eines NAND bewirkt eine 1 am Ausgang.

Zeitverhalten

Wenn der Takt **auf L** ist, ist der Eingang am NAND 

$\rightarrow$ der Ausgang ist **1**.

Springt der Takt **auf H**, dann braucht der erste Inverter **etwas Zeit**, um seinen Ausgang auf L zu setzen.

Während dieser Zeit liefert der Inverter noch immer $\overline{L} = H$ und das (neue) Takt-Signal (am oberen Eingang von NAND) ist ebenfalls H.

$\rightarrow$ während dieser Zeit ist am Ausgang des NAND **L**. Dieser Zustand dauert aber **nur sehr kurz**, weil aus dem Inverter gleich das **L**-Signal kommt und NAND wieder **auf H** springt.

Also: Die Signallaufzeit durch den Inverter bestimmt die Dauer des abgeleiteten Impulses (der aber jedenfalls sehr viel kürzer als T_H des Taktes sein wird).

Zeitverhalten

Der zweite Inverter stellt nur sicher, dass eine **positive** Flanke (Übergang **von L auf H**) einen positiven Impuls zur Folge hat.

Also: Die obige Schaltung liefert bei jeder **positiven** Flanke einen sehr kurzen Impuls (mit $\alpha_0 \ll 0.5$).

Damit kann man „Momentaufnahmen“ von logischen Signalen machen. Diese Technik nennt man Flankensteuerung (Slope Triggering).

Das Huffman-Modell

Die Beschreibung des Zeitverhaltens eines Schaltwerks

$$\mathbf{z}(t + \Delta t) = f(\mathbf{x}(t), \mathbf{y}(t))$$

$\mathbf{z}$: Der Vektor aller Ausgänge
$\mathbf{x}$: Der Vektor der Eingänge
$\mathbf{y}$: Der innere Zustand des Schaltwerks

Das Huffman-Modell eines Schaltwerks: Das Schaltwerk besteht aus zwei Blöcken:

- Kombinatorischer Schaltkreis (Schaltnetz)
- Speicher (zeitabhängiger Zustand)

In diesem Modell steuert der innere Zustand des Schaltwerks (Speicher) (der auf den Eingang rückgekoppelt wird) die Verarbeitung der Eingänge $\mathbf{x}(t)$.

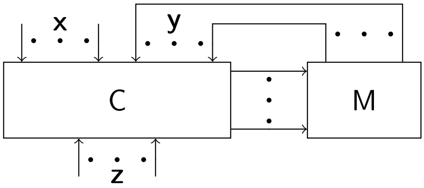
Das Huffman-Modell

Ein solches System (das auf externe Ereignisse $\mathbf{x}(t)$ je nach Zustand des Automaten unterschiedliche Aktionen setzt) heißt endlicher Automat.

(Das Modell wird vom Compilerbau über Software Engineering bis zur Robotik verwendet.)

Das Huffman-Modell

Huffman-Modell eines Schaltwerks
(mit Blöcken C (Schaltnetz) und M (Speicher, Memory))



Bemerkung: Für die Konstruktion eines Schaltwerks nach diesem Modell existieren keine analytischen Verfahren → man benutzt Heuristiken.

Für den Aufbau von C sowie auch M werden die Grundbausteine benutzt.

Flip-Flops

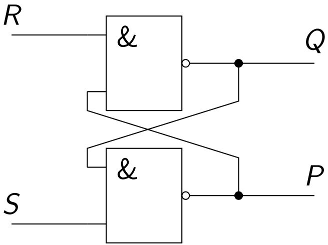
Grundelement aller logischen Bausteine mit Speichereigenschaft ist das Flip-Flop.

Die Flip-Flop-Schaltwerke sind charakterisiert durch **einen oder mehrere** (rückgekoppelte) Eingänge und **zwei komplementäre** Ausgänge (Q und $\overline{Q}$).

R-S-Flip-Flop

Das **R-S**-Flip-Flop ist das **einfachste** speichernde Bauelement:
R: Reset (rücksetzen)
S: Set (setzen)

- Als **Ein-Bit-Speicher** betrachtet. Kombination solcher Speicher-Elemente: *n*-**Bit**-Register (um *n*-Bit-Wörter zu speichern).
- Eine einfache Realisierung eines R-S-Flip-Flops mit zwei NANDs:



R-S-Flip-Flop

- **Rückkopplung** der Ausgänge (Zustände) Q, P auf die Eingänge R, S
  Die Rückkopplungen erschweren aber die Analyse der Schaltung und es entsteht dabei immer die Frage, ob sie stabile Zustände annimmt oder ob sie oszilliert.
- Der Zustand der Ausgänge Q und P wird zeitabhängig und folgendermaßen bezeichnet (Δt bezeichnet einen sehr kleinen Zeitabschnitt, der für die Änderung der Ausgänge notwendig ist):

Zeit	Zustand
t	$P_t,\ Q_t$
$t+\Delta t$	$P_{t+1},\ Q_{t+1}$
$t+2\Delta t$	$P_{t+2},\ Q_{t+2}$
$t+3\Delta t$	$P_{t+3},\ Q_{t+3}$

- Das Schaltwerk arbeitet nach folgenden Regeln:

$$\begin{aligned} Q_{t+i} &= R \text{ NAND } P_{t+i-1} \\ P_{t+i} &= S \text{ NAND } Q_{t+i-1} \end{aligned} \qquad i = 1, 2, 3, \ldots$$

R-S-Flip-Flop (Analyse)

Wahrheitstabelle (**Transitionstabelle**)

- Sie wird so genannt, weil Rückkopplungen beachtet werden müssen. Es werden darin auch zeitliche **Zustandsübergänge** eingetragen.
- Die Ableitung der kompletten Wahrheitstabelle für das R-S-Flip-Flop:

x	y	$\overline{x \& y}$ $(x$ NAND $y)$
0	0	1
0	1	1
1	0	1
1	1	0

$\rightarrow$ eine 0 am Eingang eines NAND-Gatters erzwingt eine 1 am Ausgang

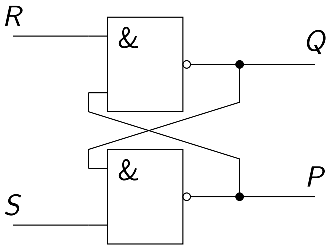
Beispiel

$R = 0, S = 1, Q_t = 1, P_t = 0$

$\rightarrow Q_{t+1} = 1, P_{t+1} = 0$

$\rightarrow Q_{t+2} = 1, P_{t+2} = 0$

$\rightarrow Q_{t+3} = 1, P_{t+3} = 0$



Zeile	R	S	Q_t	P_t	Q_{t+1}	P_{t+1}	Q_{t+2}	P_{t+2}	Q_{t+3}	P_{t+3}
1	0	0	0	0	1	1	1	1	1	1
2	0	0	0	1	1	1	1	1	1	1
3	0	0	1	0	1	1	1	1	1	1
4	0	0	1	1	1	1	1	1	1	1
5	0	1	0	0	1	1	1	0	1	0
6	0	1	0	1	1	1	1	0	1	0
7	0	1	1	0	1	0	1	0	1	0
8	0	1	1	1	1	0	1	0	1	0
9	1	0	0	0	1	1	0	1	0	1
10	1	0	0	1	0	1	0	1	0	1
11	1	0	1	0	1	1	0	1	0	1
12	1	0	1	1	0	1	0	1	0	1
13	1	1	0	0	1	1	0	0	1	1
14	1	1	0	1	0	1	0	1	0	1
15	1	1	1	0	1	0	1	0	1	0
16	1	1	1	1	0	0	1	1	0	0

R-S-Flip-Flop

- Bei $R = 0$ (**Zeilen 1–8**) ist immer $Q = 1$ (dies entspricht der Definition von NAND).
- Bei $R = 0$, $S = 1$ (**Zeilen 5–8**) herrscht ein stabiler Zustand, wobei $Q = 1$ ist und $P = 0$ (ab $t + 2$).
  Das heißt (auf den Q-Ausgang bezogen), dass eine 1 geschrieben wird (was einer **Setzen (Set)** Operation entspricht).
- Bei $R = 1$, $S = 0$ (**Zeilen 9–12**) ist die Situation umgekehrt: $P = 1$, $Q = 0$ (ab $t + 2$).
  Auf Q-Ausgang bezogen: Eine 0 wird geschrieben (was einem **Rücksetzen (Reset)** entspricht).

R-S-Flip-Flop

- Bei $R = 1$, $S = 1$ und $Q_t = P_t$ (**Zeilen 13 und 16**) ist das Schaltwerk **instabil** (da die Werte an den Ausgängen Q und P oszillieren).
- Bei $R = 1$, $S = 1$ und $Q_t = \overline{P_t}$ (**Zeilen 14 und 15**) behalten die Ausgänge Q und P ihren alten Wert.
  Auf Q-Ausgang bezogen: Der Q-Wert wird gespeichert (was einem **Speichern** Zustand des Schaltwerks entspricht).
- Die Kombination $R = 0$, $S = 0$ (**Zeilen 1–4**) muss verboten bleiben, da die Ausgänge in diesem Fall trotz Stabilität ($Q = P = 1$) nicht komplementär (Bedingung für Flip-Flop) sind.

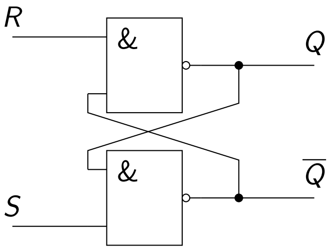
R-S-Flip-Flop

Damit entspricht die reguläre Betriebsart des R-S-Flip-Flops den Zeilen **5–8, 9–12 und 14–15**.

Für die obigen Zeilen gilt: Das Schaltwerk befindet sich (spätestens ab $t+2$, d. h. ab einer gewissen Zeit $t+\Delta t$) in einem **stabilen** (d. h. **nichtoszillierenden**) Zustand und an beiden Ausgängen Q und P liegen jeweils **komplementäre** Signale an.

Daher wird in den meisten Flip-Flop-Darstellungen der P-Ausgang direkt mit $\overline{Q}$ bezeichnet.

Im weiteren werden auch wir diese Darstellung akzeptieren.



R-S-Flip-Flop

Der Zustand, der sich hier zeitlich ändert, ist durch den Wert am Ausgang Q gegeben. Aus den Zeilen 6, 7, 10, 11, 14 und 15 können wir eine **vereinfachte Tabelle** für den Ausgang Q in Zeit t und $t + \Delta t$ ableiten:

	R	S	$Q(t)$	$Q(t+\Delta t)$
6	0	1	0	1
7	0	1	1	1
10	1	0	0	0
11	1	0	1	0
14	1	1	0	0
15	1	1	1	1

R-S-Flip-Flop

Die Zusammenfassung der Funktionen des R-S-Flip-Flops

- Für $R = S = 0$: **Unerwünschter** Zustand
- Für $R = S = 1$ ist der neue Zustand $Q(t + \Delta t)$ gleich dem alten Zustand $Q(t) \rightarrow$ Der Inhalt des Speichers bleibt unverändert, d. h. **das Schaltwerk speichert**.
- Für $R = \overline{S}$ **folgt der Ausgang** Q **immer** S (unabhängig vom Zustand), was einer **Schreiboperation** entspricht, d. h. der Speicher wird mit S beschrieben.
  - $R = 0$, $S = 1$: Es wird eine 1 geschrieben (was einem Setzen (Set) entspricht).
  - $R = 1$, $S = 0$: Es wird eine 0 geschrieben (was einem Rücksetzen (Reset) entspricht).

R-S-Flip-Flop

Die Funktion des R-S-Flip-Flops tabellarisch

R	S		
0	0	Unerwünschter Zustand	
0	1	Set (Setzen)	Schreiben
1	0	Reset (Rücksetzen)	
1	1	Speichern	

Bemerkung:
Das R-S-Flip-Flop ist die Basis aller speichernden Bausteine.
(Einen n-Bit-Speicher kann man prinzipiell als Komposition solcher 1-Bit-Speicher aufbauen.)

D-Flip-Flop

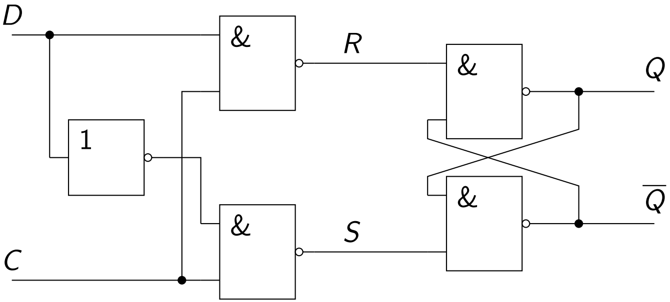
Problem beim einfachen R-S-Flip-Flop: Jede Änderung von R und S verändert sofort (innerhalb der Signallaufzeiten) den Zustand von Q.

In digitalen Rechenanlagen braucht man das Lesen oder Schreiben der Daten aber meist zu einem gewissen Zeitpunkt.

$\rightarrow$ Man braucht einen Taktgeber bzw. Clock (C), der das Flip-Flop steuert.

D-Flip-Flop

R-S-Flip-Flop erweitert um eine kleine Zusatzschaltung: D-Flip-Flop (Latch (Auffang-Flip-Flop))



D-Flip-Flop

- Ist $C = 0 \xrightarrow{\text{Eigenschaft von NAND}}$
  $R = S = 1 \xrightarrow{\text{Zeile 4 der Tabelle}}$
  D-Flip-Flop speichert, d. h. dass in diesem Zustand jegliche Änderung von D unberücksichtigt bleibt.

- Ist $C = 1 \rightarrow$ $\begin{array}{l} S{=}C \text{ NAND } \overline{D}{=}1 \text{ NAND } \overline{D} \\ R{=}C \text{ NAND } D{=}1 \text{ NAND } D \end{array} \rightarrow$
  $S = \overline{R} \rightarrow$
  Zeilen 2 und 3 der Tabelle

  Hier folgt der Ausgang Q dem Eingang $S = 1 \text{ NAND } \overline{D} = D$, d. h. das Datum vom Eingang D wird geschrieben.

Die Übersichtstabelle des R-S-Flip-Flops:

	R	S	
1	0	0	Unerwünscht
2	0	1	Set (1)
3	1	0	Reset (0)
4	1	1	Speichern

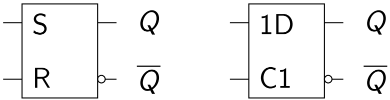
D-Flip-Flop

- Der Takt kann ein periodisches Signal (H-L) sein, aber der Takt kann auch von einem anderen Schaltwerk kommen. Dieses kann dann dem Latch signalisieren, wann es ein neues Datum übernehmen soll.

  Ist der Takt periodisch, dann wird bei jedem H-Pegel der Speicher neu beschrieben. (Der Takt kann aber auch z. B. durch einen Knopfdruck erfolgen, womit man z. B. einem Messgerät mitteilt, wann es einen neuen Messwert liefern soll.)
- Um mehrere Bits zu speichern kann das Latch einfach erweitert werden: **Parallele** Komposition von n 1-Bit-Latches, die mit dem gemeinsamen Takt synchronisiert sind, ergibt ein n-Bit-Register.

D-Flip-Flop

Die Symbole für R-S-Flip-Flop und D-Flip-Flop:



Das Symbol C1 beim Takteingang des D-Flip-Flops steht für die Datenübernahme während des H-Pegels des Takts (logische 1 der Clock).

D-Flip-Flop

Problem
Die Änderung des Speicherzustands ist bei R-S-Flip-Flop und D-Flip-Flop zustandsgesteuert. Es gibt aber Anwendungen, wo die Flankensteuerung erforderlich ist.

Dazu kann man das Latch mit der kleinen Schaltung versehen oder das **Master-Slave**-Prinzip anwenden.

D-Flip-Flop

D-Flip-Flops in **Master-Slave**-Schaltung

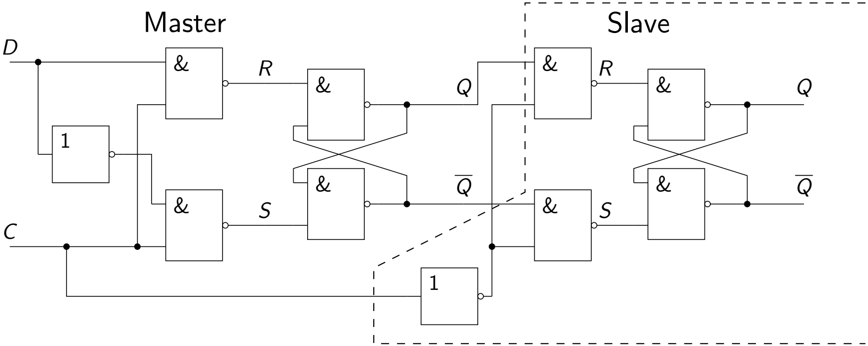
Dieses Schaltwerk besteht aus zwei D-Flip-Flops, die hintereinander geschaltet sind.

Das **vor**geschaltete D-Flip-Flop ist der Master.
Das **nach**geschaltete D-Flip-Flop ist der Slave.

Der Slave übernimmt den Ausgang vom Master **nur zu einem gewissen Zeitpunkt (Taktflanke)**.

D-Flip-Flop

D-Flip-Flops in **Master-Slave**-Schaltung



- Der **Takt** für Slave ist $\overline{C}$ (C: Takt für Master).

D-Flip-Flop

D-Flip-Flops in **Master-Slave**-Schaltung

Aufgrund der Funktionalität des D-Latches:

- Bei $C = 0 \rightarrow$ Das D-Latch vom **Master speichert**.
  Geht der Takt auf $C = 1 \rightarrow$ **Der Master übernimmt das Datum** D (Übergang $0 \rightarrow 1$: **positive Taktflanke**). Da der Takt für den **Slave** invertiert ist, ist dieser **in der Speicherstellung**, d. h. der **Ausgang von Slave bleibt unverändert**.
  (In dieser Situation kann sich der Eingang und damit der Ausgang des Masters ständig ändern, trotzdem reagiert der Slave nicht.)

- Wenn der Takt am Master **von 1 auf 0** geht (**negative Taktflanke**), geht der **Takt am Slave auf 1** und **der Slave übernimmt den Ausgang** Q **des Masters**, der sich ab diesem Zeitpunkt in Speichersituation befindet.

D-Flip-Flop

D-Flip-Flops in **Master-Slave**-Schaltung

Das globale Verhalten der Master-Slave-Schaltung:

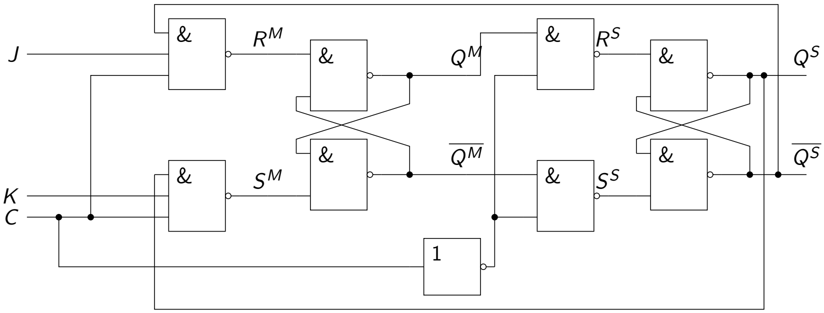
- **Übernahme des Master-Eingangs** D **auf den Slave-Ausgang** Q **mit der negativen Taktflanke**.
- Damit haben wir erreicht, dass der Speicher nur zu exakt definierten Zeitpunkten beschrieben werden kann.

J-K-Flip-Flop

Nachteil des R-S-Flip-Flops:
Die unerwünschte Eingangssituation $R = S = 0$, die die Operationsmodi dieses Flip-Flops auf Set, Reset und Speichern beschränkt.

Beim **J-K-Flip-Flop** gibt es einen zusätzlichen Modus: Kippen (Toggle) durch eine **zusätzliche Rückkopplung vom Ausgang zum Eingang** eines Master-Slave-Flip-Flops.

J-K-Flip-Flop



Der Slave-Ausgang Q^S wird auf den Master-Eingang K und $\overline{Q^S}$ auf J rückgekoppelt

$\rightarrow$ am Master wird $R^M = \overline{J\overline{Q^S}}$, $S^M = \overline{KQ^S}$ (getaktet durch C).

J-K-Flip-Flop

- Wenn $C = 1$:
  Master aktiv
  (d. h. R^M, S^M, Q^M ändern sich (oder auch nicht) in Abhängigkeit von K, J, Q^S)
  Slave speichert (weil $\overline{C} = 0 \rightarrow R^S = S^S = 1$)
  (d. h. Q^S ändert sich nicht)
- Wenn $C = 0$:
  Master speichert
  (d. h. R^M, S^M und daher auch Q^M ändern sich nicht)
  Slave aktiv (weil $\overline{C} = 1$)
  und übernimmt auf Q^S den Wert Q^M vom Master

Also: Bei der negativen Taktflanke (Übergang $C : 1 \rightarrow 0$) übernimmt der Slave den Ausgang Q^M vom Master auf seinen Ausgang Q^S.

J-K-Flip-Flop

- Wir müssen untersuchen, was **auf dem Master** bei $C = 1$ passiert (wenn dieser aktiv ist).
- Wir haben vier Fälle zur Untersuchung (abhängig von J und K):

J-K-Flip-Flop

$C = 1 \rightarrow 0$

- **Fall 1:** $J = K = 0$
  $C = 1$, $R^M = S^M = 1$ (unabhängig davon, was Q^S liefert, weil eine 0 am Eingang von NAND immer eine 1 erzwingt)
  $\rightarrow$ der **Master** ist im Speicherzustand, d. h. Q^M ändert sich nicht (unabhängig davon, was vom Slave kommt).

  $C : 1 \rightarrow 0$
  Bei $C = 0$ übernimmt der Slave also immer den gleichen unveränderten Wert Q^M

  $\rightarrow$ **Im Fall 1: Das Schaltwerk speichert.**

J-K-Flip-Flop

- **Fall 2:** $J = \overline{K} = 1$
  1. $C = 1,\ Q^S = 1$

     $R^M = \overline{J \& \overline{Q^S}} = \overline{1 \& 0} = 1,\ S^M = \overline{KQ^S} = \overline{0 \& 1} = 1$
     $\rightarrow Q^M$ bleibt unverändert $\rightarrow$ beim Übergang $C : 1 \rightarrow 0$ bleibt Q^S unverändert (d. h. $Q^S = 1$).
  2. $C = 1,\ Q^S = 0$

     $R^M = \overline{J \& \overline{Q^S}} = \overline{1 \& 1} = 0,\ S^M = \overline{KQ^S} = \overline{0 \& 0} = 1$

     $\rightarrow R^M = \overline{S^M} \overset{\text{Eigenschaft R-S-FF}}{\rightarrow} Q^M$ folgt $S^M \rightarrow Q^M = 1$
     Beim Übergang auf $C = 0$ wird $Q^M = 1$ auf Q^S übernommen $\rightarrow Q^S = 1$.

$\rightarrow$ Für **Fall 2** gilt: Der Slave-Ausgang wird immer $Q^S = 1$, **was einem Setzen des Speichers entspricht.**

J-K-Flip-Flop

- **Fall 3:** $\overline{J} = K = 1$
  $\rightarrow$ Dieser Fall ist analog zum **Fall 2** und führt zu einem **Rücksetzen** von Q^S (d. h. $Q^S = 0$).
- **Fall 4:** $J = K = 1$
  $\rightarrow$ In diesem Fall **kippt** Q^S **immer in den negierten Zustand**.

J-K-Flip-Flop

Beweis für Fall 4

$J = K = 1$

1. $C = 1,\ Q^S = 1$

   $R^M = \overline{J\overline{Q^S}} = \overline{1\&0} = 1,\ S^M = \overline{KQ^S} = \overline{1\&1} = 0$
   $\rightarrow Q^M$ folgt S^M (weil $R^M = \overline{S^M}$) $\rightarrow Q^M = 0$
   Beim Übergang auf $C = 0$ wird Q^M auf Q^S übernommen $\rightarrow$ der neue Wert $Q^S(t + \Delta t)$ wird **0**.
   Also: Q^S nimmt den **negierten** Zustand an.

2. $C = 1,\ Q^S = 0$

   $R^M = \overline{J\overline{Q^S}} = \overline{1\&1} = 0,\ S^M = \overline{KQ^S} = \overline{1\&0} = 1$
   $\rightarrow R^M = S^M \rightarrow Q^M$ folgt $S^M \rightarrow Q^M = 1$
   $Q^M = 1$ wird bei Übergang auf $C = 0$ durch den Slave übernommen $\rightarrow Q^S = 1$, was wiederum einem **negierten** Zustand entspricht.

$\rightarrow$ In diesem **Fall kippt** Q^S also immer in den negierten Zustand.

J-K-Flip-Flop

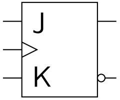
Die vereinfachte Transitionstabelle für das J-K-Flip-Flop

J	K	$Q^S(t+\Delta t)$
0	0	$Q^S(t)$
0	1	0
1	0	1
1	1	$\overline{Q^S(t)}$

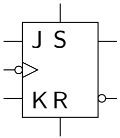
J-K-Flip-Flop

Schaltsymbole für J-K-Flip-Flop

(J/K Eingänge und der Eingang für den Takt)
(◦: Negationssymbol)



Mit positiver Flankensteuerung ohne asynchronen R/S-Eingängen



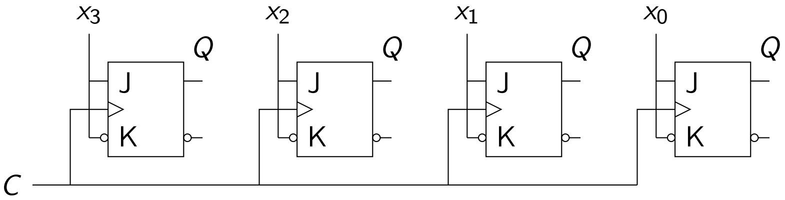
Mit negativer Flankensteuerung mit asynchronen R/S-Eingängen

Die asynchronen R/S-Eingänge ermöglichen ein Setzen/Rücksetzen des J-K-Flip-Flops unabhängig von seinem aktuellen Zustand (z. B. für die Initialisierung).

Mit J-K-Flip-Flops kann man viele Grundbausteine digitaler Rechenanlagen aufbauen.

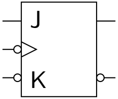
J-K-Flip-Flop

Parallelregister (zur Datenspeicherung)



4-Bit-Parallelregister (aus J-K-Flip-Flops (mit positiver Flankensteuerung) aufgebaut[2]) zur Speicherung von 4-Bit-Zahlen $(x_3x_2x_1x_0)$

Allgemein: n-Bit-Zahlen mit n J-K-Flip-Flops

[2] Ähnlicherweise können 

genommen werden, dann erfolgt die Datenübernahme mit negativer Taktflanke.

J-K-Flip-Flop

Die J-K-Flip-Flops haben $J = \overline{K} = x_i$ $(i = 3, \ldots, 0)$.

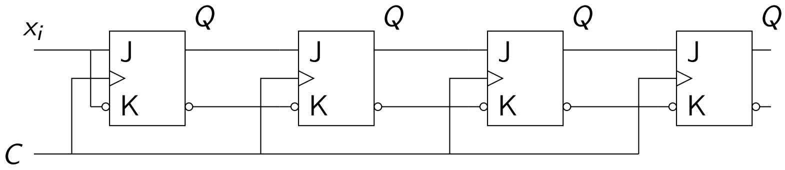
Wenn $J = \overline{K}$ $\rightarrow$ der Ausgang Q des J-K-Flip-Flops folgt dem Eingang $J \rightarrow Q = x_i$.

Mit neuer positiven Taktflanke:
$\rightarrow$ Die neue Zahl $(x_3x_2x_1x_0)$ ist am Ausgang vom Parallelregister, d. h. eine neue Zahl wird gespeichert, d. h. die Schreiboperation wurde durchgeführt.

J-K-Flip-Flop

Schieberegister (Shift Register) (Arithmetik, Codierung)

Zu jeder **Taktflanke** (siehe auch Fußnote beim Parallelregister) wandert der Eingangswert um eine Stelle nach **rechts**.



Ein 4-Bit-Schieberegister

Bei jeder Taktflanke wird der Eingang x_i in das nächste Flip-Flop geschrieben (weil wieder wie beim Parallelregister $J = \overline{K} = x_i \rightarrow Q = x_i$)

D. h. wenn sich zum Zeitpunkt t der Wert $x_i x_{i-1} x_{i-2} x_{i-3}$ im Schieberegister befindet, wird der Wert nach der nächsten (hier positiven) Taktflanke $x_{i+1} x_i x_{i-1} x_{i-2}$ sein.

J-K-Flip-Flop

Binärzähler

Um Ereignisse zu zählen (z. B. die Öffnung der Parkschranke), um Programme zu steuern, die Zeitmessung durchzuführen, etc.

Ein neues Ereignis bedeutet dabei immer binäre Addition $(+1)$ zum Zähler und dadurch entsteht ein neues Ergebnis.

$$
\begin{array}{cc}
 & \text{Zähler} \\
+ & \text{neues Ereignis} \\
\hline
= & \text{das Ergebnis}
\end{array}
\qquad
\begin{array}{cccccccccc}
 & \text{MSB} & & & & & & & & \text{LSB} \\
 & a_0 & a_1 & a_2 & \dots & a_{i-1} & a_i & a_{i+1} & \dots & a_{n-1} \\
+ & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 1 \\
\hline
 & a_0 & a_1 & a_2 & \dots & a_{i-1} & \overline{a_i} & \overline{a_{i+1}} & \dots & \overline{a_{n-1}}
\end{array}
$$

Wobei: i ist der erste Index (**von rechts**), für den $a_i = 0$ gilt.

J-K-Flip-Flop

Die Addition:

$$\begin{array}{cccccccccccc} & & & & & & i & i+1 & & \dots & n-1 \\ & a_0 & a_1 & a_2 & \dots & a_{i-1} & 0 & 1 & 1 & \dots & 1 \\ + & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 1 \\ \hline & a_0 & a_1 & a_2 & \dots & a_{i-1} & 1 & 0 & 0 & \dots & 0 \end{array}$$

Also diese Addition:

- Kippt die Bits (LSB) $a_{n-1}, a_{n-2}, \dots, a_i$ um
- Lässt die Bits (MSB) $a_{i-1}, a_{i-2}, \dots, a_0$ unverändert

Beispiel:

$a_0 a_1 \underline{a_2} a_3 a_4 a_5 = 1\,0\,\underline{\mathbf{0}}\,1\,1\,1 \to i = \underline{2}$

$$\begin{array}{ccccccc} & 1 & 0 & 0 & 1 & 1 & 1 \\ + & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline & 1 & 0 & 1 & 0 & 0 & 0 \end{array}$$

$\to$ Bits a_5, a_4, a_3, a_2 invertieren:

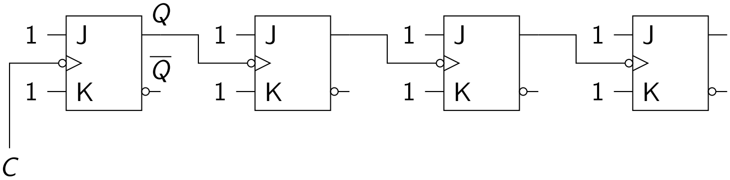
$0111 \to 1000$

und

Bits a_1, a_0 unverändert lassen: 10

J-K-Flip-Flop

Asynchroner binärer Vorwärtszähler

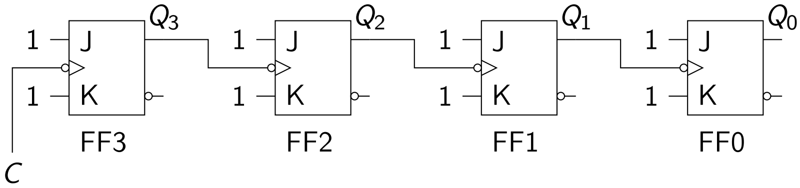


Der externe Takt C steuert nur das erste Flip-Flop.

Der Takt für die restlichen Flip-Flops wird aus den Ausgängen der vorherigen abgeleitet:

Alle Flip-Flops haben $J = K = 1$ (d. h. $Q(t + \Delta t) = \overline{Q(t)}$), also sind sie **zum Kippen vorbereitet**. Das Kippen geschieht **bei negativer Taktflanke**, d. h. beim Übergang von Q von 1 auf 0 (bei der positiven Taktflanke, d. h. wenn Q von 0 auf 1 geändert wird, passiert nichts).

J-K-Flip-Flop



	(LSB)			(MSB)
	a_3	a_2	a_1	a_0
$+$	1	0	0	0
		1. 2.	1. 2.	1. 2.
	$\overline{a_3}$	$\overline{a_2}$ oder a_2	$\overline{a_1}$ oder a_1	$\overline{a_0}$ oder a_0

FF$_i$: $a_i = Q_i(t), Q_i' = Q_i(t + \Delta t)$

(Aufgrund des Ereignisses (negative Taktflanke) wird $Q_3 = a_3$ gekippt: Also ist der neue Ausgangswert $Q_3' = \overline{a_3}$.)

J-K-Flip-Flop

1. Wenn $a_3 = Q_3 = 1 \to Q_3' = \overline{a_3} = 0$ → **negative** Taktflanke für FF2 → kippen, d. h. $Q_2' = \overline{a_2}$.
2. Wenn $a_3 = Q_3 = 0 \to Q_3' = \overline{a_3} = 1$ → **positive** Taktflanke, also kippt FF2 nicht um → der Ausgangswert $Q_2' = a_2$ (bleibt unverändert).

J-K-Flip-Flop

Beispiel $[n = 4]$

$a_0\, a_{\underline{1}}\, a_2\, a_3 = 1\,\underline{0}\,1\,1$ (also $i = \underline{1}$) $\rightarrow$ das Ergebnis ist $a_0\, \overline{a_1}\, \overline{a_2}\, \overline{a_3} = a_0\, 1\, 0\, 0 = 1\, 1\, 0\, 0$

- FF3 kippt $\rightarrow a'_3 = \overline{a_3} = \overline{1} = 0$
  Weil $Q_3 = 1$ und $Q'_3 = 0 \rightarrow$ **negative** Taktflanke $\rightarrow$
- FF2 kippt $\rightarrow Q'_2 = \overline{a_2} = \overline{1} = 0 \rightarrow$ FF2 erzeugt eine **negative** Taktflanke für FF1 (weil $Q_2 = 1$, $Q'_2 = 0$) $\rightarrow$
- FF1 kippt $\rightarrow Q'_1 = \overline{a_1} = 1 \rightarrow$ FF1 erzeugt eine **positive** Taktflanke für FF0 $\rightarrow$
- FF0 kippt **nicht** $\rightarrow$ Ergebnis ist $a_0\, \overline{a_1}\, \overline{a_2}\, \overline{a_3}$

J-K-Flip-Flop

Wenn $a_0\,a_1\,a_2\,a_3 = 1\,1\,1\,1$:

- FF3 bekommt negative Taktflanke $\rightarrow Q_3' = \overline{Q_3} = \overline{a_3} = \overline{1} = 0$
  $\rightarrow$ FF3 erzeugt negative Taktflanke für FF2
- FF2 kippt $\rightarrow Q_2' = \overline{Q_2} = \overline{1} = 0$
- FF1 kippt $\rightarrow Q_1' = 0$
- FF0 kippt $\rightarrow Q_0' = 0$

$\rightarrow$ aus dem Zählerstand $1\,1\,1\,1$ bekommt man $0\,0\,0\,0$.

J-K-Flip-Flop

Der Zähler arbeitet **asynchron**, da sich die Zählerstände der einzelnen Flip-Flops nur **nacheinander** ändern können, aber nicht gleichzeitig.
(Es kommt daher beim Kippen eines Flip-Flops kurzzeitig immer zu Fehlanzeigen des Zählers – die aber meist keine Rolle spielen.)

→ Im folgenden Beispiel zeigen wir die Konstruktion eines **synchronen** binären Vorwärtszählers:
Alle Flip-Flops werden **synchron** (unter einem **gemeinsamen** Takt) geändert.

J-K-Flip-Flop

Synchroner binärer Vorwärtszähler

Aufbau aus 4 J-K-Flip-Flops

Wegen der Synchronisierung muss der gemeinsame Takt an allen Flip-Flops anliegen. Die **J/K-Eingänge** heißen hier **Vorbereitungseingänge**, weil diese das Flip-Flop auf eine Änderung (oder eine Speicherung) des Ausgangs vorbereiten sollen.

Bemerkung:
Im Sinn des **Huffman-Modells** sind die **Flip-Flops der Speicherteil des Schaltwerks** und **das Schaltnetz**, das am Eingang den inneren Zustand (Zählerstand) erhält und daraus die Vorbereitungseingänge generiert **ist der kombinatorische Schaltkreis**.

J-K-Flip-Flop

Erzeugung einer Transitionstabelle, die alle möglichen Zählerstände (Zustandsübergänge) enthält und für jeden Übergang die benötigten Vorbereitungseingänge angeben

Vorbereitungen des J-K-Flip-Flops für Ausgangsübergänge

				Übergang					
J	K	$Q(t+\Delta t)$		$Q(t)$		$Q(t+\Delta t)$	J	K	
0	0	$Q(t)$		0	$\rightarrow$	0	0	X	$\boxtimes$
0	1	0	$\rightarrow$	0	$\rightarrow$	1	1	X	
1	0	1		1	$\rightarrow$	0	X	1	
1	1	$\overline{Q(t)}$		1	$\rightarrow$	1	X	0	

X: Beliebig, d. h. 1 oder 0

J-K-Flip-Flop

Begründung ☒:
Der Übergang von **0** zu **0** erfolgt bei einem J-K-Flip-Flop wenn

- entweder $J = K = 0$
  (dann ist $Q(t + \Delta t) = Q(t)$,
  also $\mathbf{0} = Q(t) \to Q(t + \Delta t) = \mathbf{0}$)
- oder $J = 0$ und $K = 1$
  (dann ist $Q(t + \Delta t) = 0$,
  also $\mathbf{0} = Q(t) \to Q(t + \Delta t) = \mathbf{0}$).

J-K-Flip-Flop

Die Transitionstabelle des synchronen 4-Bit Vorwärtszählers:

	t	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
	Q_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	Q_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	Q_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	Q_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
FF0	J_0	1	X	1	X	1	X	1	X	1	X	1	X	1	X	1	X
	K_0	X	1	X	1	X	1	X	1	X	1	X	1	X	1	X	1
FF1	J_1	0	1	X	X	0	1	X	X	0	1	X	X	0	1	X	X
	K_1	X	X	0	1	X	X	0	1	X	X	0	1	X	X	0	1
FF2	J_2	0	0	0	1	X	X	X	X	0	0	0	1	X	X	X	X
	K_2	X	X	X	X	0	0	0	1	X	X	X	X	0	0	0	1
FF3	J_3	0	0	0	0	0	0	0	1	X	X	X	X	X	X	X	X
	K_3	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	1

J-K-Flip-Flop

Der 4-Bit Zähler ist aus 4 J-K-Flip-Flops (FF0, FF1, FF2, FF3) aufgebaut und kann sich insgesamt in 16 Zuständen befinden, ausgedrückt durch Zahlen $Q_0Q_1Q_2Q_3 = 0000, \ldots, 1111$.

Aus der Tabelle kann man die notwendigen Vorbereitungen ablesen. Der Übergang von 12 auf 13 benötigt z. B. die folgenden Eingangsvorbereitungen von J und K auf den Flip-Flops:

$$\begin{array}{ccccc} & & & & \text{FF3} \quad \text{FF0} \\ 12 & = & Q_3Q_2Q_1Q_0 & = & 1\;1\;0\;0 \\ 13 & = & Q_3'Q_2'Q_1'Q_0' & = & 1\;1\;0\;1 \\ & & & & \text{MSB} \quad \text{LSB} \end{array} \rightarrow \text{Es folgt aus der Tabelle auf Folie 465}$$

Für Übergang

$Q_0 \rightarrow Q_0'$ (d. h. $0 \rightarrow 1$) muss auf FF0 $J_0 = 1,\ K_0 = X$ sein.
$Q_1 \rightarrow Q_1'$ (d. h. $0 \rightarrow 0$) muss auf FF1 $J_1 = 0,\ K_1 = X$ sein.
$Q_2 \rightarrow Q_2'$ (d. h. $1 \rightarrow 1$) muss auf FF2 $J_2 = X,\ K_2 = 0$ sein.
$Q_3 \rightarrow Q_3'$ (d. h. $1 \rightarrow 1$) muss auf FF3 $J_3 = X,\ K_3 = 0$ sein.

J-K-Flip-Flop

Für alle Eingänge gilt, dass wir $J = K$ setzen können (da dies die Verteilung der X ermöglicht, weil in jedem Paar von J und K ein X ist).

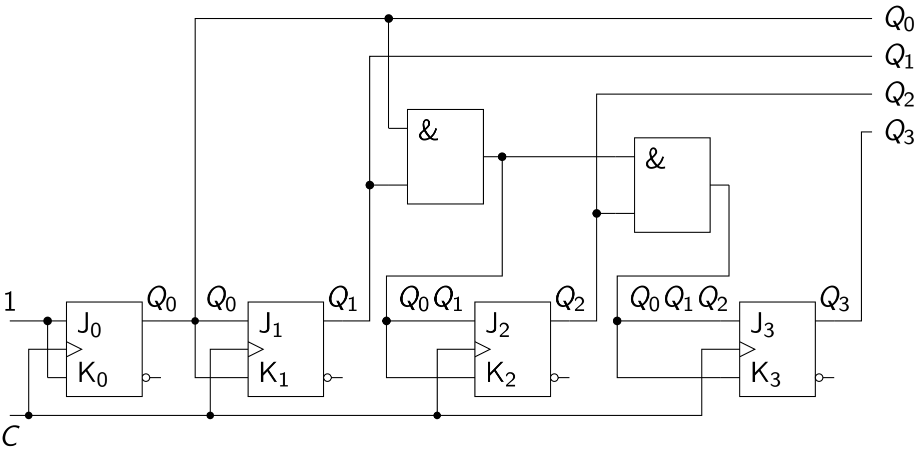
- Auf FF0 hat man daher $J_0 = K_0 = 1$, d. h. dieses Flip-Flop ist immer auf Kippen vorbereitet. (Dies ist auch an der Struktur der Übergänge von Q_0 (immer abwechselnd $0 \to 1$ oder $1 \to 0$) erkennbar – volle Analogie zum asynchronen Fall.)
- Für FF1 ergibt sich $J_1 = K_1 = Q_0$ (weil überall in der Tabelle ist Q_0 gleich entweder J_1 oder K_1 und der andere Eingang ist X.)
  $\rightarrow$ Dieses Flip Flop erhält also den Ausgang von FF0.

J-K-Flip-Flop

- Für FF2 folgt $J_2 = K_2 = Q_0 Q_1$ (einfach aus der Tabelle abzulesen, z. B. für $t = 11 : Q_0 Q_1 = 1\&1 = 1$ oder $t = 06 : Q_0 Q_1 = 0\&1 = 0$. Im Allgemeinen muss dies über die vollständige DNF und einem Minimierungsverfahren abgeleitet werden.)
- Für FF3 lässt sich $J_3 = K_3 = Q_0 Q_1 Q_2$ ableiten.

$\rightarrow$ Das Schaltwerk für den **synchronen** binären 4-Bit-Vorwärtszähler ist auf der folgenden Folie abgebildet.

J-K-Flip-Flop



J-K-Flip-Flop

Ähnlicherweise lassen sich verschiedene Zähler und andere Schaltwerke aufbauen.

Erster Schritt:
Prinzipielle Aufteilung in Schaltnetz und Speicher (Huffman-Modell).
(Also muss man unterscheiden, welche Größen den Ein-/Ausgang des Systems beschreiben. Im Spezialfall des Zählers sind Zustands- und Ausgangsvariablen identisch.)

Die Schaltnetze lassen sich dann aus den Transitionstabellen in bekannter Weise ableiten.

Die Zustandsgrößen werden durch Flip-Flops gespeichert und verändert.