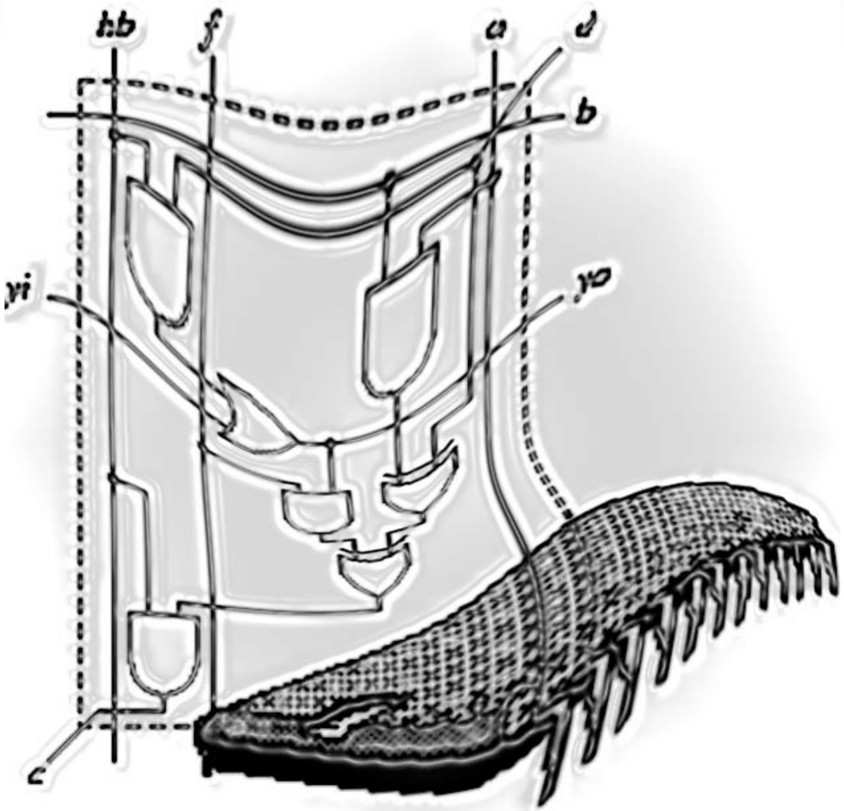
Technische Informatik I



Kapitel 2

Boolesche Algebra

Prof. Dr. Dirk W. Hoffmann

Hochschule Karlsruhe ◆ University of Applied Sciences ◆ Fakultät für Informatik

Schaltalgebra

- $\neg$, $\wedge$ und $\vee$ sind Operatoren über der Menge {0,1}

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

Konjunktion

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

Disjunktion

a	$\neg a$
0	1
1	0

Negation

- Die Operatoren erfüllen mehrere wichtige Gesetze
  - Kommutativgesetze
    - $a \wedge b = b \wedge a$ $\quad$ $a \vee b = b \vee a$
  - Distributivgesetze
    - $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ $\quad$ $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
  - Existenz von neutralen Elementen
    - $a \wedge 1 = a$ $\quad$ $a \vee 0 = a$
  - Existenz von inversen Elementen
    - $a \wedge \neg a = 0$ $\quad$ $a \vee \neg a = 1$

Boolesche Algebra

Gegeben: Menge V, Operatoren $\bullet, + : V \times V \rightarrow V$

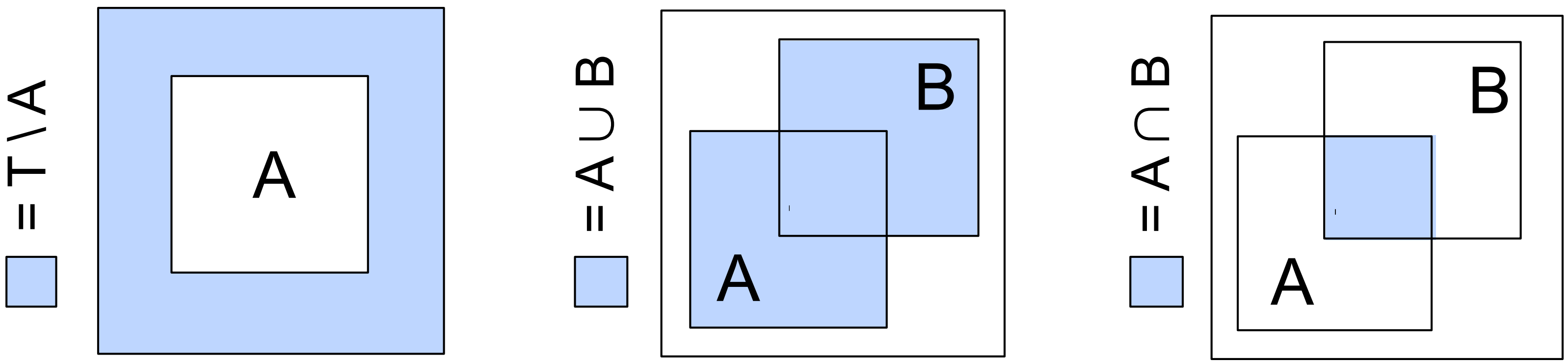
V heißt boolesche Algebra,
wenn die folgenden vier Huntington'schen Axiome gelten:

- Kommutativgesetze (K): $a \bullet b = b \bullet a$
  $a + b = b + a$
- Distributivgesetze (D): $a \bullet (b + c) = (a \bullet b) + (a \bullet c)$
  $a + (b \bullet c) = (a + b) \bullet (a + c)$
- Neutrale Elemente (N): Es existieren $e, n \in V$ mit
  $a \bullet e = a$ und $a + n = a$
- Inverse Elemente (I): Für alle $a \in V$ existiert ein a' mit
  $a \bullet a' = n$ und $a + a' = e$

Boolesche Algebra

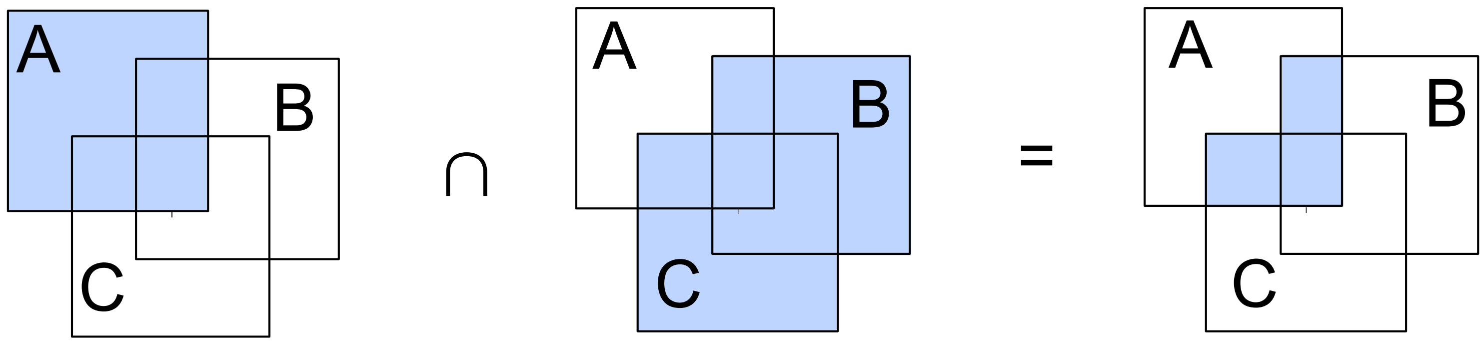
- Mengenalgebra über einer Trägermenge T

Boolesche Algebra	Mengenalgebra	
V	$\wp(T)$	Potenzmenge der Trägermenge T
•	$\cap$	Durchschnitt
+	$\cup$	Vereinigung
n	$\emptyset$	Leere Menge
e	T	Trägermenge
a‘	$T \setminus A$	Komplementärmenge

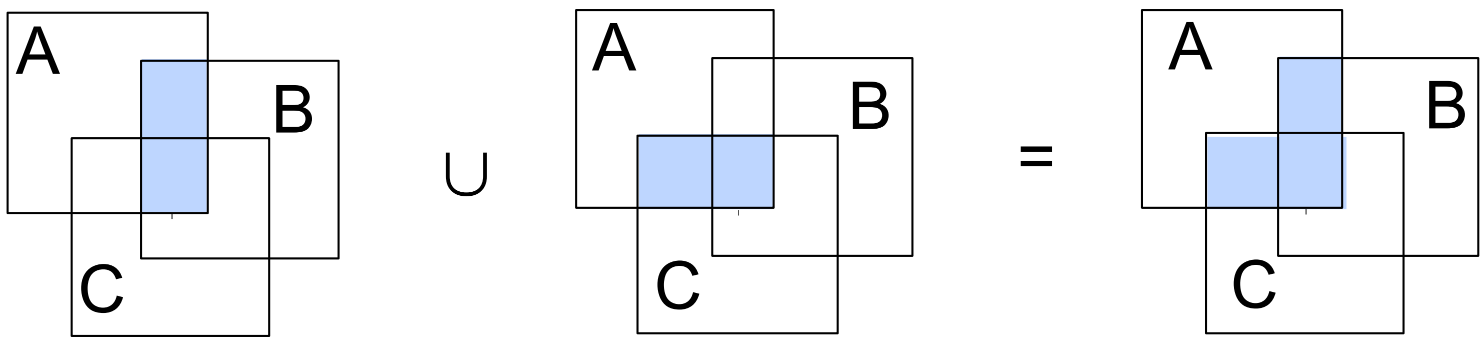


Boolesche Algebra: Beispiele

- $A \cap (B \cup C)$

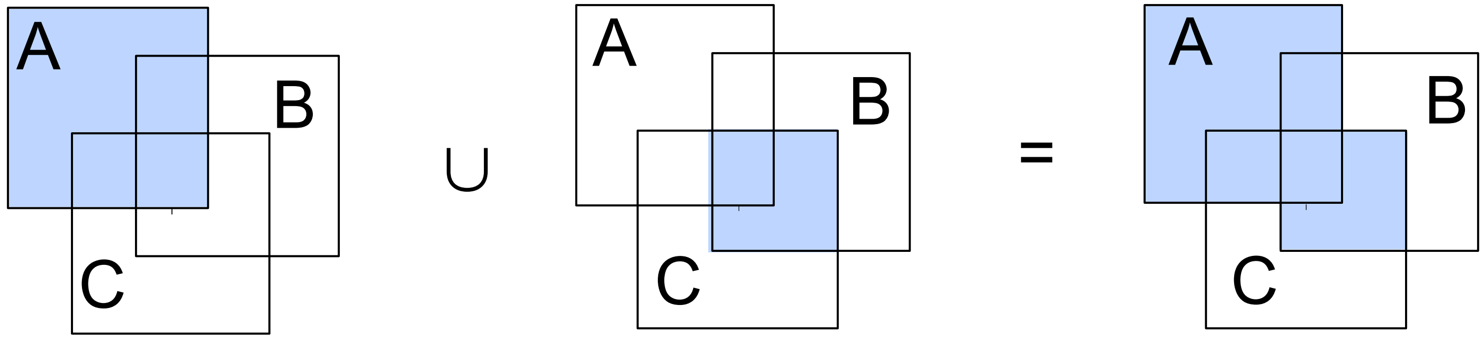


- $(A \cap B) \cup (A \cap C)$

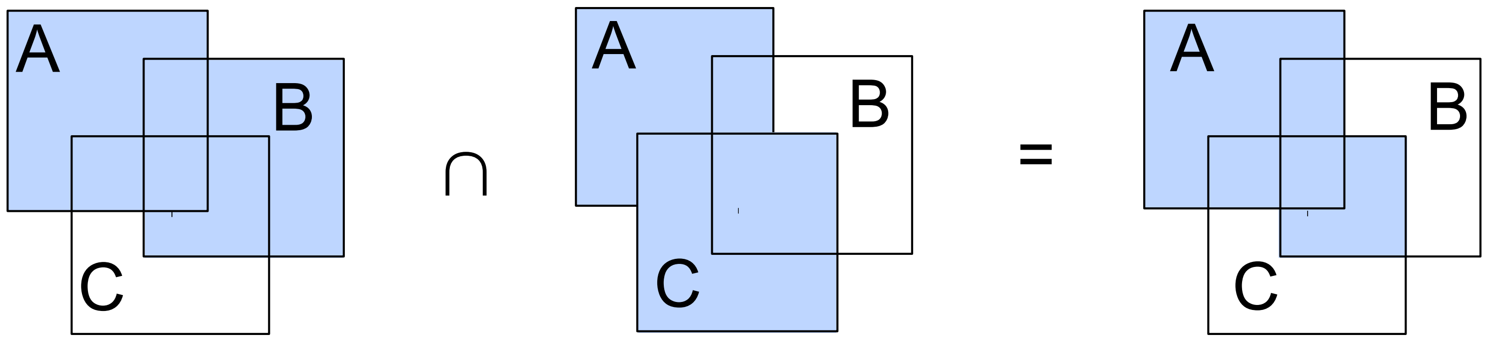


Boolesche Algebra: Beispiele

- $A \cup (B \cap C)$



- $(A \cup B) \cap (A \cup C)$



Boolesche Algebra

- Auch die Schaltalgebra ist eine boolesche Algebra

Boolesche Algebra	Schaltalgebra	
V	{ 1, 0 }	Wahrheitswerte (TRUE, FALSE)
•	∧	Konjunktion (UND-Operator)
+	∨	Disjunktion (ODER-Operator)
n	0	„Falsch“ (FALSE)
e	1	„Wahr“ (TRUE)
a‘	¬a	Negation (Verneinung)

Notation und Operatorenbindung

- Abgeleitete Operatoren („*syntactic sugar*“)

  - $(a \rightarrow b)$ für $(\neg a \vee b)$ (Implikation)
  - $(a \leftarrow b)$ für $(b \rightarrow a)$ (Inv. Implikation)
  - $(a \leftrightarrow b)$ für $(a \rightarrow b) \wedge (a \leftarrow b)$ (Äquivalenz)
  - $(a \oplus b)$ für $\neg(a \leftrightarrow b)$ (Antivalenz)

- Bezeichnungen

  - $\neg(a \vee b)$ (NOR-Operation)
  - $\neg(a \wedge b)$ (NAND-Operation)

- Boolesche Funktionen
  - ¬ ist eine einstellige boolesche Funktion

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

  - Alle anderen Operatoren sind zweistellige boolesche Funktionen

$$\wedge,\ \vee,\ \ldots : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$$

  - Wie viele zweistellige boolesche Funktionen gibt es insgesamt?

Die zweistelligen booleschen Funktionen

		$f_0 = 0$	$f_1 = a \wedge b$	$f_2 = \neg a \wedge b$	$f_3 = b$	$f_4 = \neg b \wedge a$	$f_5 = a$	$f_6 = a \oplus b$	$f_7 = a \vee b$	$f_8 = \neg(a \vee b)$	$f_9 = a \leftrightarrow b$	$f_{10} = \neg a$	$f_{11} = a \rightarrow b$	$f_{12} = \neg b$	$f_{13} = a \leftarrow b$	$f_{14} = \neg(a \wedge b)$	$f_{15} = 1$
b	a	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		Nullfunktion	Konjunktion					Antivalenz	Disjunktion	NOR	Äquivalenz		Implikation		Inverse Implikation	NAND	Einsfunktion

Notation und Operatorenbindung

- Alternative Notation der booleschen Operatoren
  - $(a \bullet b)$ bzw. (ab) anstelle $(a \wedge b)$
  - $(a + b)$ anstelle $(a \vee b)$
  - $\overline{a}$ anstelle $\neg a$
  - (a ⇹ b) anstelle $(a \oplus b)$
- Bindung der Operatoren
  - $\wedge$ bindet stärker als $\vee$
  - $\neg$ bindet stärker als $\wedge$
- Klammerung
  - Gleiche binäre Operatoren werden linksassoziativ zusammengefasst, z.B. $a \wedge b \wedge c = (a \wedge b) \wedge c$

Kommutativgesetze	$a \wedge b = b \wedge a$ $a \vee b = b \vee a$	(K)
Distributivgesetze	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$	(D)
Neutrale Elemente	$a \wedge 1 = a$ $a \vee 0 = a$	(N)
Inverse Elemente	$a \wedge \neg a = 0$ $a \vee \neg a = 1$	(I)

In jeder Booleschen Algebra, so auch in der Schaltalgebra, gelten die vier oben gezeigten Huntington‘schen Axiome

Aus den Huntington‘schen Axiomen lassen sich weitere praktische Rechenregeln ableiten…

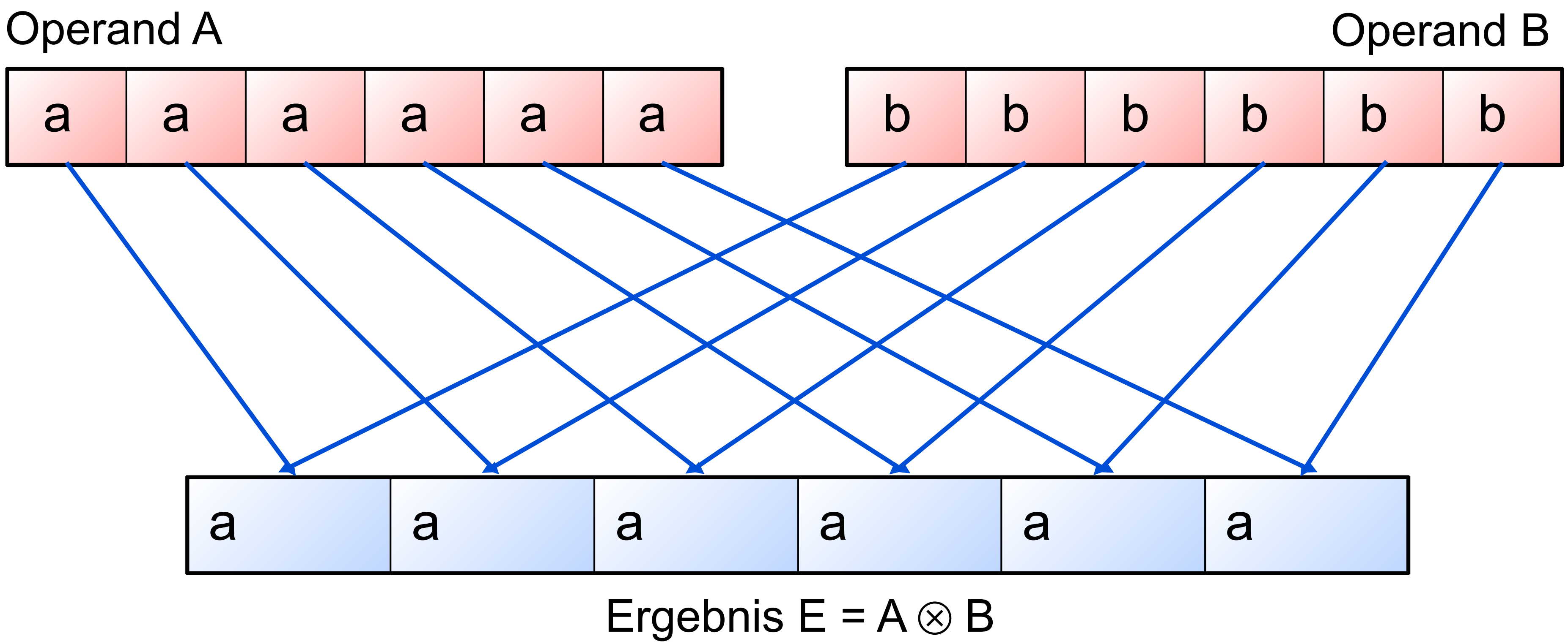
Kommutativgesetze	$a \wedge b = b \wedge a$ $a \vee b = b \vee a$	(K)
Distributivgesetze	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$	(D)
Neutrale Elemente	$a \wedge 1 = a$ $a \vee 0 = a$	(N)
Inverse Elemente	$a \wedge \neg a = 0$ $a \vee \neg a = 1$	(I)
Assoziativgesetze	$a \wedge (b \wedge c) = (a \wedge b) \wedge c = a \wedge b \wedge c$ $a \vee (b \vee c) = (a \vee b) \vee c = a \vee b \vee c$	(A)
Idempotenzgesetze	$a \wedge a = a$ $a \vee a = a$	(ID)
Absorptionsgesetze	$a \vee (a \wedge b) = a$ $a \wedge (a \vee b) = a$	(AB)
Gesetze von DeMorgan	$\neg(a \vee b) = \neg a \wedge \neg b$ $\neg(a \wedge b) = \neg a \vee \neg b$	(M)
Auslöschungsgesetze	$a \wedge 0 = 0$ $a \vee 1 = 1$	(L)
Gesetz der Doppelnegation	$\neg\neg a = a$	(DN)

Anwendung der Regeln

- Vereinfachung von Ausdrücken
  - Beispiel 1: $Y = (A \vee B) \wedge (\neg A \vee B) \wedge (A \vee \neg B)$
  - Beispiel 2: $Y = (A \rightarrow B) \rightarrow ((\neg A \rightarrow B) \rightarrow B)$

Bitweise logische Operationen

A,B seien Bitvektoren, ⊗ eine beliebige Verknüpfung



Bitweise logische Operationen

- UND, ODER und XOR wirken wie spezielle Bit-Masken

	1	1	0	1	0	0
∧	0	1	1	1	0	0
=	0	1	0	1	0	0

	1	1	0	1	0	0
∨	0	1	1	1	0	0
=	1	1	1	1	0	0

	1	1	0	1	0	0
⊕	0	1	1	1	0	0
=	1	0	1	0	0	0

UND wird verwendet, um Bits gezielt auf 0 zu setzen. Dazu hat die Maske an allen Bitpositionen, die übernommen werden sollen, eine 1 und an den Stellen, die auf 0 gesetzt werden sollen, eine 0.

ODER wird verwendet, um Bits gezielt auf 1 zu setzen. Dazu hat die Maske an allen Bitpositionen, die übernommen werden sollen, eine 0 und an den Stellen, die auf 1 gesetzt werden sollen, eine 1.

XOR wird verwendet, um Bits gezielt zu kippen. Dazu hat die Maske an allen Bitpositionen, die übernommen werden sollen, eine 0 und an den Stellen, die gekippt werden sollen, eine 1.