FH D Digitaltechnik 1

Prof. Dr. J.Wietzke

Email	j.wietzke@fbi.fh-darmstadt.de
Tel	+49 (6151) 16-8472
Fax	+49 (6151) 16-8935
Post	Haardtring 100, 64295 Darmstadt
Sprechstunde	Mi. 16.00-17.30 Uhr, 14/208

1 Einleitung

1.1 Vorwort

Das Skript wurde aus mehreren Vorlagen und eigenen Texten zusammengestellt, auch zur möglichen Vereinheitlichung der Vorlesungen, die über mehrere Züge von verschiedenen Dozenten vorgetragen werden.
Das Skript ist nicht zur Weitergabe bestimmt sondern dient nur zur eigenen Vorbereitung der Vortragenden. Teil 1 endet vor oder mit der Einführung der Flip-Flops, je nach Fortschritt der Vorlesung, dann beginnt Teil 2 mit Fokus auf Schaltwerken, Speichern und Zustandsautomaten. Da die Vorlesung Rechnerorganisation thematisch recht voll ist, wird die Datensicherung mit Mitteln der Kodierung hier vor der Bearbeitung der Massenspeicher besprochen.

1.2 Nachrichten und Information

Man unterscheidet zwischen den Begriffen "Nachricht" und Information. Eine Nachricht ist eine Zeichenfolge, die nach bestimmten (syntaktischen) Regeln gebildet wurde, während Information die Bedeutung (Semantik) der Nachricht für deren Empfänger bezeichnet. Da Information ein subjektiver Begriff ist und kaum formalisiert werden kann, wollen wir ihn hier nicht genauer definieren. Beispiel: Nachricht: Glockenschlag der Kirchturmuhr - Information: Tageszeit. Nachrichten werden mit Zeichen eines Alphabets gebildet. Dadurch entstehen Worte, dies sind Zeichenketten endlicher Länge. Aus Worten lassen sich Sätze bilden.

1.3 Bits und Bitfolgen

Die Maßeinheit der Information ist ein Bit; d.h. 1 Bit ist die kleinste mögliche Informationsmenge. Dies ist die Informationsmenge einer Nachricht, die in einer Antwort auf eine Frage, die genau zwei Antworten zuläßt und die Bedeutung der Nachricht bestimmt, enthalten ist, also in Antworten wie:

- ja oder nein,
- wahr oder falsch,
- schwarz oder weiß,
- hell oder dunkel,
- groß oder klein,
- stark oder schwach,
- links oder rechts.

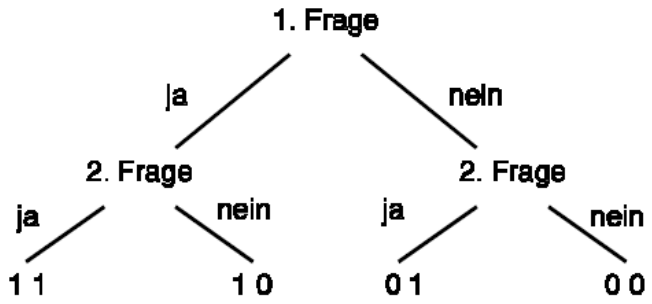
Zu einer solchen Frage läßt sich immer eine sogenannte Codierung der Antwort festlegen. Da es nur zwei mögliche Antworten gibt, reichen dazu zwei Zeichen aus. Man benutzt meist die Zeichen 0 und 1, manchmal auch, um Mißverständnisse auszuschließen, L (low) und H (high), und nennt sie Bits. Die Codierung ist nötig, weil die Information technisch dargestellt werden muß. Man benutzt dabei etwa elektrische Ladungen:
0 = ungeladen, 1 = geladen
oder elektrische Spannungen

0 entspricht einer Spannung kleiner 2 Volt,
1 entspricht einer Spannung größer 2 Volt
oder Magnetisierungen
0 = nicht magnetisiert, 1 = magnetisiert.
Auf die genaue technische Realisierung kommt es uns hier aber nicht an.

Zur Darstellung von Information werden also Nachrichten (Zeichenfolgen) verwendet. Ist die Information einer Nachricht durch eine Frage bestimmt, die mehr als zwei Antworten zuläßt, so enthält die Beantwortung der Frage mehr als 1 Bit an Information.
Die Frage etwa, aus welcher Himmelsrichtung, Nord, Süd, Ost oder West, der Wind weht, läßt vier mögliche Antworten zu. Der Informationsgehalt der Antwort auf diese Frage ist aber nur 2 Bit, denn man kann die ursprüngliche Frage in zwei elementare Fragen verwandeln, die jeweils nur zwei Antworten zulassen:

1. Weht der Wind aus einer der Richtungen Nord oder Ost (ja/nein)?
2. 2. Weht der Wind aus einer der Richtungen Ost oder West (ja/nein)?

Eine mögliche Antwort, etwa ja auf die erste Frage und nein auf die zweite Frage, läßt sich durch die beiden Bits
1 0
repräsentieren (man sagt dazu auch, die Antwort wird als Bitfolge 10 codiert). Die Bitfolge 10 besagt also diesmal, daß der Wind aus Norden weht. Ähnlich repräsentieren die Bitfolgen
0 0 « Süd 0 1 « West 1 0 « Nord 1 1 « Ost.
Offensichtlich gibt es genau vier mögliche Folgen mit 2 Bits. Mit 2 Bits können wir also Fragen beantworten, die vier mögliche Antworten zulassen.



Entscheidungsbaum

Obenstehend ist dafür der sogenannte Entscheidungsbaum angegeben. Man erhält die vorgesehene Codierung, wenn man von der 'Wurzel' des Baums zu den 'Blättern' fortschreitet und für 'ja (links)' eine 1 , für nein (rechts) eine '0' wählt.
Ein Baum ist eine in der Informatik oft wiederkehrende wichtige Datenstruktur und kann wie folgt definiert werden. Ein Baum besteht aus Knoten und Kanten; Kanten verbinden immer zwei Knoten. Die Knoten eines Baums lassen sich in Ebenen einordnen. Die Anzahl der Ebenen nennt man Tiefe des Baums. Die erste Ebene enthält nur einen Knoten, die Wurzel; die Knoten aller anderen Ebenen sind jeweils mit genau einem Knoten der nächst niedrigeren Ebene über eine Kante verbunden. Die Knoten der letzten Ebene nennt man Blätter. In einem Binärbaum sind die Knoten mit höchstens drei anderen Knoten verbunden - bis auf die Wurzel; sie ist höchstens mit zwei Knoten verbunden. Bäume sind spezielle (gerichtete) Graphen.
Lassen wir auf die Frage 'Woher weht der Wind?' auch noch die Zwischenrichtungen Südost, Nordwest, Nordost und Südwest zu, so gibt es vier weitere mögliche Antworten, also insgesamt acht. Mit einem zusätzlichen Bit, also mit insgesamt 3 Bits, können wir alle acht

möglichen Antworten codieren.
Die möglichen Folgen aus 3 Bits sind:
0 0 0, 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0, 1 1 1
und die möglichen Antworten auf die Frage nach der Windrichtung sind:
Süd, West, Nord, Ost, Südost, Nordwest, Nordost, Südwest.
Jede beliebige ein-eindeutige Zuordnung der Himmelsrichtungen zu diesen Bitfolgen können wir als Codierung der Windrichtungen hernehmen, z.B.:
0 0 0 = Süd 1 0 0 = Südost
0 0 1 = West 1 0 1 = Nordwest
0 1 0 = Nord 1 1 0 = Nordost
0 1 1 = Ost 1 1 1 = Südwest
Offensichtlich verdoppelt jedes zusätzliche Bit die Anzahl der möglichen Bitfolgen, so daß gilt:
Es gibt genau 2^N mögliche Bitfolgen (Codierungen, Codewörter) der Länge N. Sie bilden einen Code.

1.4 Präfixfreie Codes

Eine Zeichenkette Z heißt Präfix der Zeichenkette W genau dann, wenn die Länge von Z kleiner als die von W und der Anfang von W (von links) mit Z identisch ist.
Beispiel:
01 ist Präfix von 01011,
kein Präfix von 01,
kein Präfix von 1101.
kein Präfix von 0.
Definition: Eine Codierung C (ein Code) ist präfixfrei, wenn kein Codewort aus C Präfix eines anderen Codeworts aus C ist.
Beispiel: A1 Codewörter
präfixfrei:
A 010
B 011
E 101
R 11
nicht präfixfrei:
A 110
B 101
E 10
R 11
Codewörter eines präfixfreien Codes können eindeutig decodiert werden, ebenso Sequenzen solcher Codewörtern. Decodieren heißt, das Urbild des Codewortes zu bestimmen.
Eine Binärcodierung erhält man, wenn T das Binäralphabet B = {0, 1} ist. Information, die maschinell verarbeitbar sein soll, wird als Bitfolgen (Worte eines Binärcodes) dargestellt. Ein Blockcode ist ein Code, dessen Codewörter alle die gleiche Länge haben. Codewörter codieren in der Regel Daten, die als Text oder Zahlen vorliegen aber z. B. auch Maschinenbefehle.
Bevor wir uns aber überlegen, wie solche Daten am besten codiert werden, wollen wir den Begriff Information noch näher beleuchten.

1.5 Darstellung von Information

In diesem Kapitel wird die Darstellung von Texten, logischen Werten, Zahlen und Instruktionen als Daten, genauer als Bitfolgen, erläutert. Eine Bitfolge bestehend aus 8 Bits nennt man ein Byte.

1.6 Darstellung von Text

Um Texte in einem Rechner darzustellen, codiert man das Textalphabet und Satzzeichen in Bitfolgen. Für die Darstellung aller Zeichen kommt man mit 7 Bits aus; das ergibt 128 verschiedene Möglichkeiten. Man muß dann eine Tabelle erstellen, mit der man jedem Zeichen einen solchen Bitcode zuordnet. Die heute gebräuchlichste Tabelle liefert die sogenannte ASCII-Codierung. ASCII (sprich aski) steht für „American Standard Code for Information Interchange“.
Einige Zeichen mit ihrem 7-Bit-ASCII-Code:

ASCII	Zeichen		
a	1100001	0	0110000
A	1000001	1	0110001
b	1100010	9	0111001
B	1000010	CR	0001101
z	1111010	+	0101011
Z	1011010	-	0101101
?	0111111	=	0111101

Auf vielen Rechnern, darunter auch auf den IBM-kompatiblen Personal Computern, hat man den ASCII-Code um nochmals 128 Zeichen erweitert, man benötigt dazu 8 Bits, ein Byte, zur Darstellung eines Zeichens. Diesen Code nennt man auch den erweiterten ASCII-Code. Die zusätzlichen Codes benutzt man zur Darstellung sprachspezifischer Sonderzeichen. Inzwischen ist der sogenannte UNICODE mit 16 Bits gebräuchlich. Mit ihm lassen sich u.a. die Alphabete sämtlicher Sprachen der Welt codieren.

1.7 Darstellung von Wahrheitswerten

Aussagen der Aussagenlogik können zweierlei Werte annehmen: wahr oder falsch. Man verwendet zur Darstellung der Wahrheitswerte ein Binäralphabet. Die Wahrheitswerte lassen sich durch die logischen Operatoren : UND (AND,• , $\wedge$) und ODER (OR, +, $\vee$) miteinander verknüpfen: wahr UND falsch = falsch, wahr ODER falsch = wahr, etc.

1.8 Graphiken und Bilder

Auch komplexe Gebilde wie Graphiken und Bilder können als Daten in einem Computer verarbeitet und gespeichert werden. Bilder und Graphiken werden dabei z.B. in Folgen von Rasterpunkten aufgelöst. Diese Methode kann man sich veranschaulichen, wenn man ein

Zeitungsbild mit einer Lupe betrachtet. Bei ausreichender Vergrößerung sieht man, wie das Bild aus einzelnen Rasterpunkten zusammengesetzt ist. Jedem dieser Rasterpunkte (Pixel) ist ein Zahlenwert (ein Bit, ein Byte oder mehrere Bytes) zugeordnet, der einen Farbwert codieren, je nachdem, ob das Bild ein- oder mehrfarbig ist. Eine Folge solcher Codes für Rasterpunkte repräsentiert dann zusammen mit einer Vorschrift zur Umsetzung (CLUT color look up table) ein Bild oder eine Graphik.

1.9 Darstellung ganzer Zahlen

Wir sind gewohnt, Zahlen im Dezimalsystem darzustellen und verwenden dafür die zehn Ziffern (digits) 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Um diese als Bitfolgen zu codieren, benötigen wir mindestens 4 Bits, also ein halbes Byte. Ein halbes Byte nennt man ein Nibble (oder Halbbyte).
Mit einem Nibble lassen sich jedoch 24 = 16 Ziffern codieren. Deshalb führt man weitere 6 Ziffern ein. Da diese auf heutigen Tastaturen nicht vorhanden sind, verwendet man dafür die Zeichen A, B, C, D, E und F.

0000 = 0	0100 = 4	1000 = 8	1100 = C
0001 = 1	0101 = 5	1001 = 9	1101 = D
0010 = 2	0110 = 6	1010 = A	1110 = E
0011 = 3	0111 = 7	1011 = B	1111 = F

BCD-Code mit den Pseudotetraden A bis F.
Obenstehende gibt die übliche Codierung dieser 16 Ziffern als sogenannte Tetraden wieder. Man nennt diesen Code, eingeschränkt auf die Dezimalziffern, BCD-Code (binary coded decimals).
Darstellung von Maschinenbefehlen mit Hilfe von Tetraden:
Maschinenbefehle sind lange Bitfolgen, deren Länge oftmals ein Vielfaches von 4 ist. Damit diese leichter zu lesen sind, werden auch zu deren Darstellung oft die Tetraden verwendet.

Für unsere Zwecke ist aber eine andere Darstellung von Zahlen von größerer Bedeutung: die Dualzahl-Darstellung, denn mit dieser Darstellung „rechnen“ Rechner.
Zahlen(werte) werden in Positionsschreibweise dargestellt (Stellenwertsystem). Den Zahldarstellungen in allen diesen Zahlensystemen ist ein Wert zugeordnet, der durch das folgende Bildungsgesetz gewonnen wird:

$$Z = \sum_i x_i Y^i$$

Falls i ≥0 ist, erhalten wir die Darstellung einer ganzen positiven ganzen Zahl. Die Zahl Y ist die Basis des Zahlensystems und i gibt die Stellen der Ziffern an; xi ist der Wert der i-ten Ziffer. Es gibt genau Y Ziffern. Das duale Zahlensystem ist das einfachste Stellenwertsystem, das sich realisieren läßt. Es hat die Basis 2 und kommt deshalb mit zwei Ziffern 0 und 1, den Bits, aus (B = 8 Octal-, B = 16 Hexadezimalsystem). Um Mehrdeutigkeiten zu vermeiden, gibt man zusätzlich oft das Zahlensystem in Klammern an. (Um jedoch alle Mehrdeutigkeiten auszuschließen, soll man z.B. statt (10) besser (zehn) schreiben.)
Beispiele:
Dezimalsystem: Y = 10 (zehn) und x3 = 1, x2 = 0, x1 = 9, x0 = 3
1093(10)
Dualsystem: x3 = 1, x2 = 0, x1 = 1, x0 = 1 Z = 1011(2)
Z = 1011(2), dargestellt im Dezimalsystem, ist Z = 11(10), denn 1•8 + 0•4 + 1•2 + 1•1 =

11(10).
Es dürfte somit auch klar sein, warum der in der Tabelle wiedergegebene Code BCD-Code heißt.
Die Position (Stelle) einer Ziffer legt fest, wieviel sie zum Wert der darstellten Zahl beiträgt. In diesen Stellenwertsystemen wird also eine Zahl durch eine Folge von Ziffern xi beschrieben. Der Ziffernvorrat ist so groß wie die Basis. Im Hexadezimalsystem mit der Basis 16(10) benötigen wir also 16(10) Ziffern. Den Hexadezimalziffern
A, B, C, D, E und F sind die Werte
10(10), 11(10), 12(10), 13(10), 14(10) und 15(10) zugeordnet.
Mittels Konvertierungsverfahren (z.B. dem Horner Schema) lassen sich Zahlen von einem in ein anderes Zahlensystem umwandeln. Ebenso lassen sich die Rechenoperationen, wie wir sie vom Dezimalsystem her kennen - Addition, Subtraktion, Multiplikation und Division - in all diesen Zahlensystemen formulieren: Stellenweises Rechnen mit Weitergabe von Überträgen.
Negative und positive Zahlen werden durch Hinzufügen (meist durch Vorausstellung) eines Vorzeichens unterschieden, üblicherweise + und -, man kann im Dualsystem aber auch die Bits 0,1 als Vorzeichen nehmen (0 positiv, 1 negativ). Man nennt dies die „Vorzeichen mit Betrag“-Darstellung.
Beispiel: 01011(2) -> + 11(10)
11011(2) -> - 11(10)

2 Boolesche Algebra

2.1 Boolsche Algebra

Die Boolesche Algebra (Schaltalgebra) dient der Beschreibung digitaler Funktionen. Ähnlich wie in der Algebra werden Variable über Operatoren verknüpft und es lassen sich Gleichungen aufstellen und umformen. Im Unterschied zur Algebra, die wir von der Mathematikausbildung her kennen, nehmen die Variablen jedoch nur die Werte 0 und 1 an.
Die Theorie zur Booleschen Algebra wurde 1854 von dem Mathematiker George Boole entwickelt. Die Anwendung der Booleschen Algebra zur Behandlung von Schaltnetzen erkannte Claude E. Shannon um 1940. Er zeigte, daß diese Algebra besonders geeignet ist, die Eigenschaften von Serien- und Parallelschaltungen von Schaltern und Relais zu beschreiben. Schalter und Relais waren damals die Grundelemente zum praktischen Aufbau digitaler Schaltungen.

2.2 Boolsche Menge

Wir betrachten eine aus genau zwei unterscheidbaren Elementen bestehende Menge.
Historisch gibt es
“0“ und “1“ zur Beschreibung algebraischer Verknüpfungen
“F“ und “T“ zur Beschreibung logischer Verknüpfungen (TRUE,FALSE)
„“L“ und „“H“ zur Beschreibung elektrischer Verknüpfungen (Low Voltage, high Voltage)

Technologie	0	1
Relais	Schalter offen	Schalter geschlossen
Glasfaser	Licht aus	Licht an
Compact Disk	Keine Vertiefung	Vertiefung
CMOS Schaltkreis	0 – 1,5 Volt	3,5 – 5 Volt
TTL-Schaltkreis	0 – 0,8 Volt	2 – 5 Volt

Elementare boolesche Operatoren: UND, ODER, NICHT
Als elementare Operatoren der booleschen Funktionen werden üblicherweise die Operatoren UND, ODER und NICHT ($\wedge$, $\vee$, $\neg$) verwendet.
Für die Grundverknüpfungen werden häufig auch andere Operatoren verwendet,
z.B. a×b oder a&b für die UND-Verknüpfung
sowie a+b oder a$\vee$b für die ODER-Verknüpfung.
Die Negation/Inversion bindet stärker als UND und ODER.
Mit diesen Operatoren kann man alle booleschen Funktionen durch Gleichungen beschreiben.

2.3 Die Booleschen Postulate

Definition:
Grundlage der Booleschen Algebra ist die Menge A mit den Elementen 0 und 1, in der die zweielementigen Operationen "+" und "·" (bzw. "$\vee$" und "$\wedge$") definiert sind, so daß gilt:

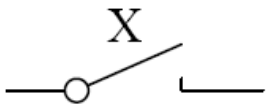
2.3.1 Postulate (Annahmen)

- P1: a = 0 oder a = 1 mit a $\in$ A
- P2: $0 \cdot 0 = 0$ $0 \wedge 0 = 0$
- P3: $1 \cdot 1 = 1$ $1 \wedge 1 = 1$
- P4: $0 + 0 = 0$ $0 \vee 0 = 0$
- P5: $1 + 1 = 1$ $1 \vee 1 = 1$
- P6: $1 \cdot 0 = 0 \cdot 1 = 0$ $1 \wedge 0 = 0$ $0 \wedge 1 = 0$
- P7: $1 + 0 = 0 + 1 = 1$ $1 \vee 0 = 1$ $0 \vee 1 = 1$
- P8: $\neg 1 = 0$ $\neg 0 = 1$

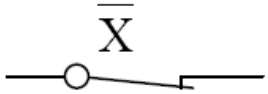
2.3.2 Null und Eins Gesetze

X *1 = X
X +1 = 1
X *0 = 0
X +0 = X

Unter Heranziehung der Booleschen Postulate (Annahmen, Voraussetzungen) können die Booleschen Theoreme (Lehrsätze) aufgestellt werden. Die Richtigkeit dieser Lehrsätze kann jederzeit mit Hilfe der Postulate überprüft werden Die Operationen lassen sich anhand von Kontaktschaltungen erläutern, die aus Schließern und Öffnern bestehen:
Schließer:



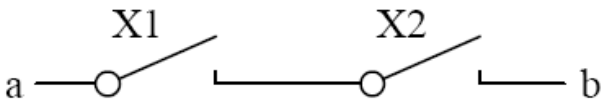
Schließer:
X=0: Schalter geöffnet
X=1: Schalter geschlossen



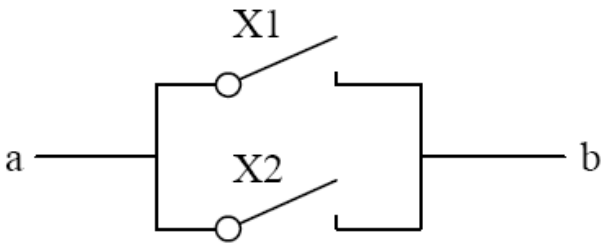
Öffner:
X=0: Schalter geschlossen
X=1: Schalter geöffnet

Eine UND-Verknüpfung besteht aus der Hintereinanderschaltung zweier Öffner, eine ODER-Verknüpfung aus der Parallelschaltung. Für eine Negation wird der Schließer verwendet.

UND-Verknüpfung



ODER-Verknüpfung



XY	X·Y
0 0	0
0 1	0
1 0	0
1 1	1

UND

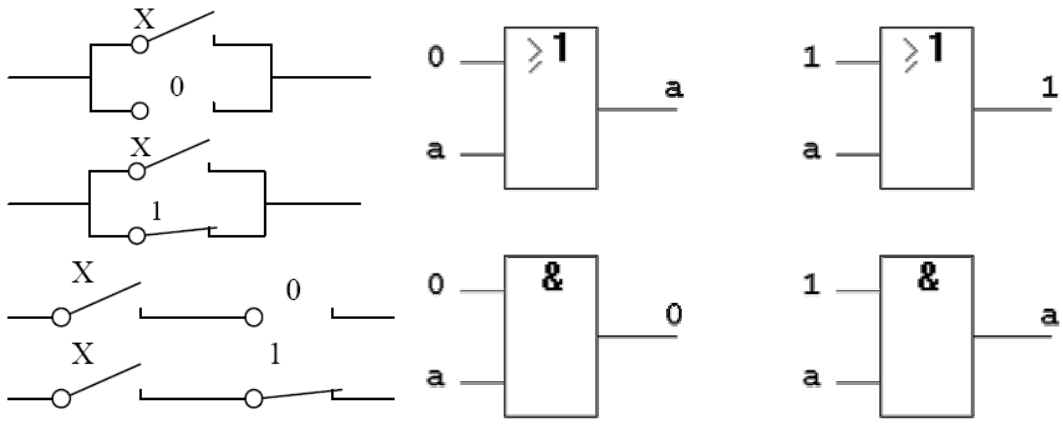
XY	X+Y
0 0	0
0 1	1
1 0	1
1 1	1

ODER

Z	$\overline{Z}$
0	1
1	0

NICHT

Es gelten nun basierend auf den Postulaten die folgenden Gesetze der Schaltalgebra, die sich anhand der Kontaktschaltungen anschaulich erläutern lassen.

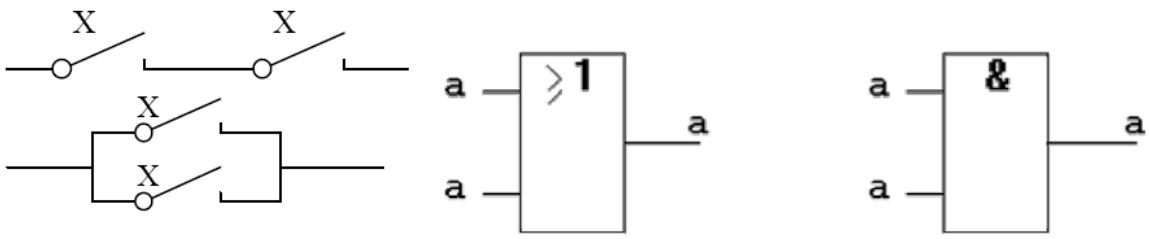


2.3.3 Doppelte Negierung

$$\overline{\overline{x}} = (\overline{\overline{x}}) = x$$

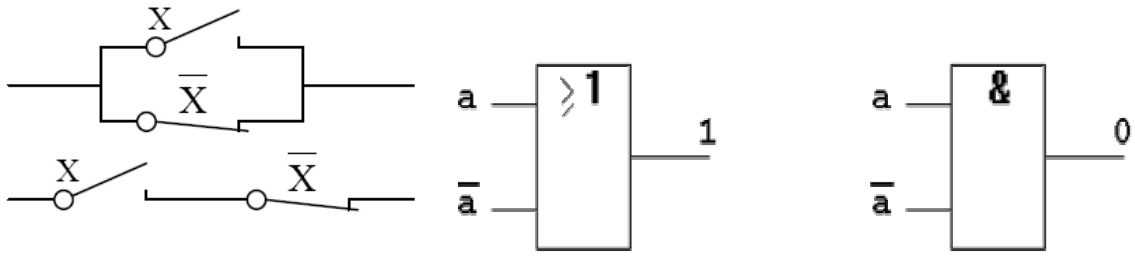
2.3.4 Idempotenzgesetze (Identitätsgesetz)

$X \wedge X = X$
$X \vee X = X$



2.3.5 Komplementgesetze

Zu jedem Wert X existiert ein Komplement X , für welches die folgenden Gesetze gelten
$X \vee \neg X = 1$
$X \wedge \neg X = 0$
$\neg\neg X = X$

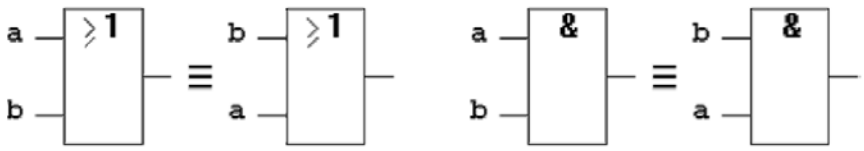


2.3.6 Kommutativgesetze (Vertauschungsgesetze)

Die UND-Verknüpfung ist kommutativ: $X1 \wedge X2 = X2 \wedge X1$

Die ODER-Verknüpfung ist kommutativ: $X1 \vee X2 = X2 \vee X1$

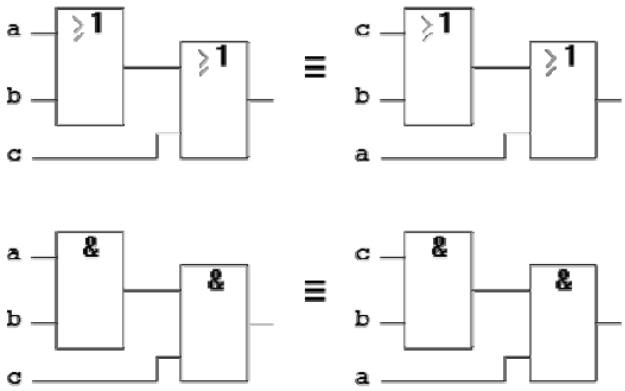
Schaltungstechnisch:



2.3.7 Assoziativgesetze(Zusammenfassungsgesetz)

Die UND-Verknüpfung ist assoziativ: $(X1 \wedge X2) \wedge X3 = X1 \wedge (X2 \wedge X3)$
Die ODER-Verknüpfung ist assoziativ: $(X1 \vee X2) \vee X3 = X1 \vee (X2 \vee X3)$

Aufgrund des Assoziativgesetzes ist die Bearbeitungsreihenfolge gleicher Operatoren $\wedge$ bzw. $\vee$ egal. Daher werden zur eindeutigen Spezifikation der Funktion die Klammern nicht benötigt. Es gilt somit
z.B.: $(X1 \wedge X2) \wedge X3 = X1 \wedge (X2 \wedge X3) = X1 \wedge X2 \wedge X3$.
Aufgrund des Kommutativgesetzes können weiterhin die Operanden in ihrer Reihenfolge vertauscht werden, so daß z.B. gilt:
$(X1 \wedge X2) \wedge X3 = X1 \wedge X2 \wedge X3 = X3 \wedge X1 \wedge X2$.



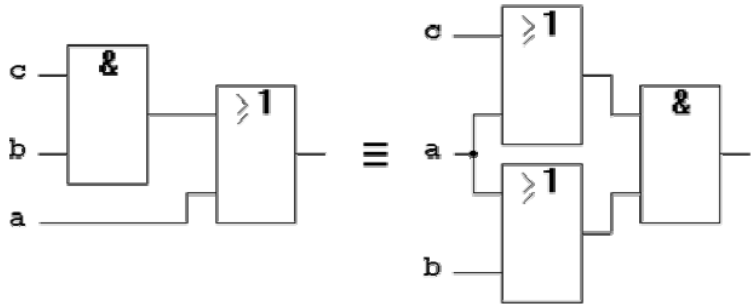
Jeder Klammerung, die in der algebraischen Form auftritt, entspricht bei der Gatterimplementierung eine Gatterebene

2.3.8 Distributivgesetze

1. Distributivgesetz: $X1 \wedge (X2 \vee X3) = (X1 \wedge X2) \vee (X1 \wedge X3)$
2. Distributivgesetz: $X1 \vee (X2 \wedge X3) = (X1 \vee X2) \wedge (X1 \vee X3)$

Aussagenlogisch kann das zweite Distributivgesetz wie folgt erläutert werden:

- Der Ausdruck X1∨(X2∧X3) ist genau dann wahr, wenn X1 wahr ist oder wenn X2 und X3 beide wahr sind.
- (X1∨X2)∧(X1∨X3) ist genau dann wahr, wenn jeder der beiden Klammerausdrücke wahr ist. Diese Ausdrücke sind wiederum wahr, wenn X1 wahr ist. Ist X1 nicht wahr, so müssen X2 und X3 beide wahr sein. Somit sind beide Ausdrücke identisch.



Das 1. Distributivgesetz kann entsprechend erläutert werden.

2.3.9 Kürzungsregeln, Absorptionsgesetze

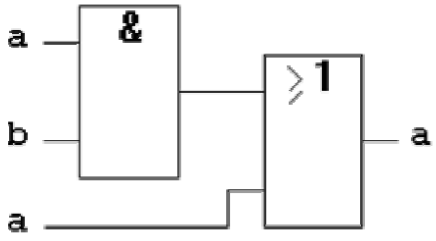
1. Kürzungsregel: X1 ∨ (X1 ∧ X2) = X1
2. Kürzungsregel: X1 ∧ (X1 ∨ X2) = X1
3. Kürzungsregel: X1 ∨ (¬X1 ∧ X2) = X1 ∨ X2
4. Kürzungsregel: X1 ∧ (¬X1 ∨ X2) = X1 ∧ X2
5. Kürzungsregel: (X1 ∧ X2) ∨ (X1 ∧ ¬X2) = X1
6. Kürzungsregel: (X1 ∨ X2) ∧ (X1 ∨ ¬X2) = X1

Der Beweis der Kürzungsregeln kann z.B. durch Umformungen oder durch Aufstellen der Wahrheitstabelle erfolgen.

Beispiel: Beweis der 1. Kürzungsregel durch Umformen
X1 ∨ (X1 ∧ X2) = (X1 ∧ 1) ∨ (X1 ∧ X2) (Null- und Eins-Gesetze)
= X1 ∧ (1 ∨ X2) (1. Distributivgesetz,)
= X1 ∧ 1 (Null- und Eins-Gesetze)
= X1 (Null- und Eins-Gesetze)

Beispiel: Beweis der 3. Kürzungsregel durch Aufstellen der Wahrheitstabelle:

X1 X2	$Y = X1 \vee (\overline{X1} \wedge X2)$
0 0	0
0 1	1
1 0	1
1 1	1

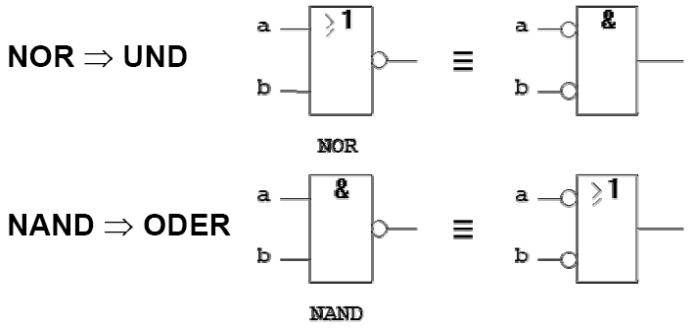


2.4 De Morgansche Gesetze

1. De Morgansches Gesetz: $\neg(X1 \wedge X2) = \neg X1 \vee \neg X2$

2. De Morgansches Gesetz: $\neg(X1 \vee X2) = \neg X1 \wedge \neg X2$

Die De Morganschen Gesetze gelten auch für mehr als 2 Variable. Ein Beweis hierzu ist mittels der Distributivgesetze möglich.



2.5 Dualitätsprinzip, Shannonsches Gesetz

Es wurde eine boolesche Algebra auf den Werten 0 und 1 und den Operatoren UND, ODER und NICHT definiert und die Verknüpfungen in Form von Postulaten vereinbart.
Diese boolesche Algebra weist eine Symmetrie auf. Zu jeder Aussage erhält man eine duale Aussage, wenn man die Werte 0 und 1 sowie die Operatoren UND und ODER vertauscht.
Die Symmetrie nennt man das Dualitätsprinzip. Es folgt natürlich unmittelbar aus dem symmetrischen Aufbau der Postulate bezüglich der entsprechenden Operatoren und Werte.
Die De Morganschen Gesetze basieren direkt auf dem Dualitätsprinzip.
Das Shannonschen Gesetz formuliert dieses Dualitätsprinzip:

$\neg f(X1, X2, ...Xn, \wedge, \vee) = f(\neg X1, \neg X2, ..., \neg Xn, \vee, \wedge)$
Es sagt aus, daß der invertierte Wert einer Funktion f gleich dem Wert ist, den die gleiche Funktion liefert, wenn man alle Operanden negiert und die Operatoren UND und ODER vertauscht.
Weiterhin kann man in Anlehnung an obige Gleichung die Eigenschaft dualer Funktionen vereinbaren. Zwei Funktionen f1 und f2 sind zueinander dual, wenn gilt:

$f1(X1, X2, ...Xn, \wedge, \vee) = \neg f2(\neg X1, \neg X2, ..., \neg Xn, \vee, \wedge)$

Zwei Funktionen sind somit zueinander dual, wenn durch Negieren der Operanden, Vertauschen der Operatoren und Negieren des Ergebnisses die eine Funktion in die zweite übergeht.

Beispiel:

X1 X2	X1∧X2
0 0	0
0 1	0
1 0	0
1 1	1

dual ⟸⟹

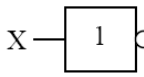
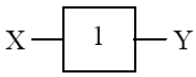
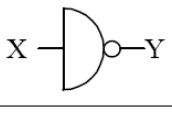
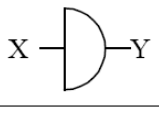
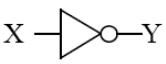
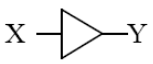
X1 X2	X1∨X2
1 1	1
1 0	1
0 1	1
0 0	0

3 Boolesche Funktionen

Mithilfe der booleschen Operatoren
UND Konjunktion
ODER Disjunktion
NICHT Negation
lassen sich alle boolesche Funktionen mit beliebig vielen Eingangsvariablen realisieren.

3.1.1 Funktionen mit einem Eingang und Ausgang

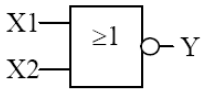
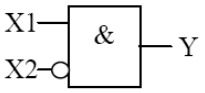
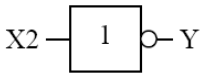
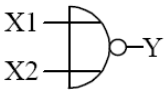
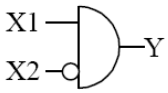
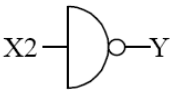
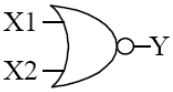
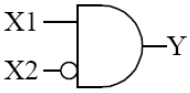
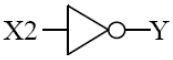
Die Tabelle gibt einen Überblick über alle Funktionen mit Hilfe der entsprechenden Wahrheitstafeln, Gleichungen und der zur Funktion zugehörige VHDL-Signalzuweisung. Für jede Funktion ist die zugehörige Bezeichnung angegeben.

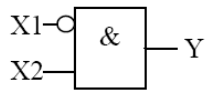
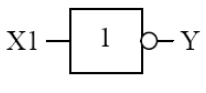
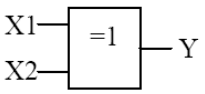
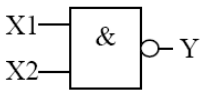
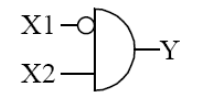
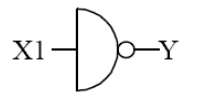
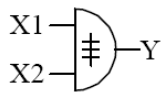
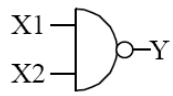
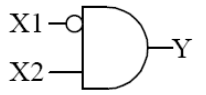
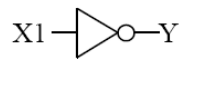
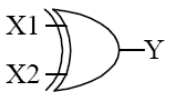
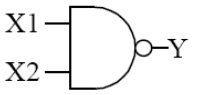
Bezeichnung		Konstante 0	NICHT, Negation, Inversion	Abbild, Treiber, Identität, Buffer	Konstante 1
Wahrheitstabelle	X	Y	Y	Y	Y
	0	0	1	0	1
	1	0	0	1	1
Gleichung		Y = 0	Y = ¬X, Y = /X, Y = $\overline{X}$	Y = X	Y = 1
VHDL		`Y <= '0';`	`Y <= not X;`	`Y <= X;`	`Y <= '1';`
Schaltsymbol DIN 40900					
DIN (alte Norm)					
ANSI / IEEE (alte Norm)					

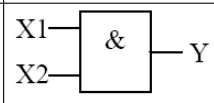
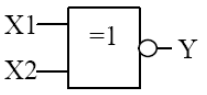
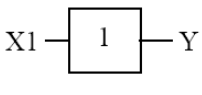
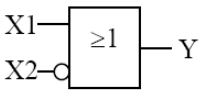
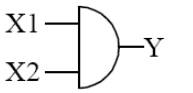
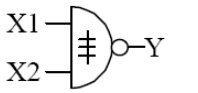
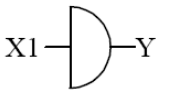
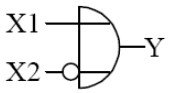
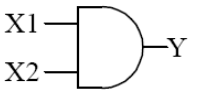
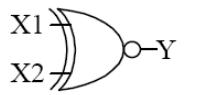
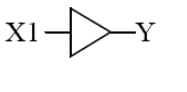
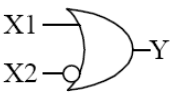
3.1.2 Funktionen mit zwei Eingängen und einem Ausgang

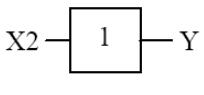
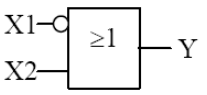
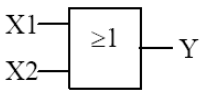
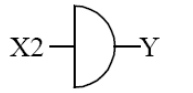
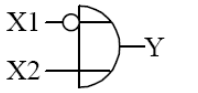
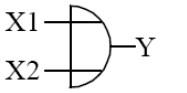
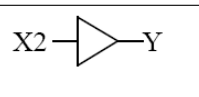
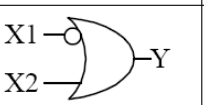
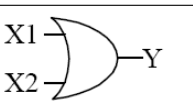
Es gibt 16 boolesche Funktionen mit zwei Eingängen und einem Ausgang. Die nachfolgende Tabelle listet alle 16 Funktionen.
Jede Funktion ist mit einem Namen bezeichnet und kann über Wahrheitstabellen/Wertetafeln beschrieben werden oder mit den Operatoren UND, ODER und NICHT. Für jede Funktion ist die zugehörige VHDL-Beschreibung gezeigt und die Schaltsymbole in den drei

gebräuchlichen Normen angegeben. Die dargestellten Beschreibungen boolscher Funktionen sind äquivalent.

Bezeichnung		Konstante 0	NOR	Inhibition	Negation (X2)
Wahrheitstabelle	X2 X1	Y	Y	Y	Y
	0 0	0	1	0	1
	0 1	0	0	1	1
	1 0	0	0	0	0
	1 1	0	0	0	0
Gleichung UND/ODER/NICHT		Y = 0	Y = ¬(X1∨X2)	Y = X1∧(¬X2)	Y = ¬X2
Operator			X1 $\overline{\vee}$ X2		¬X2
VHDL		`Y <= '0';`	`Y <= X1 nor X2;`	`Y <= X1 and not X2;`	`Y <= not X2;`
Schaltsymbol DIN 40900					
DIN (alte Norm)					
ANSI / IEEE (alte Norm)					

Bezeichnung		Inhibition	Negation (X1)	Antivalenz, XOR	NAND
Wahrheitstabelle	X2 X1	Y	Y	Y	Y
	0 0	0	1	0	1
	0 1	0	0	1	1
	1 0	1	1	1	1
	1 1	0	0	0	0
Gleichung UND/ODER/NICHT		Y = (¬X1)∧X2	Y = ¬X1	Y = (X1∧(¬X2)) ∨ ((¬X1)∧X2)	Y = ¬(X1∧X2)
Operator			¬X1	X1↮X2 , X1⊕X2	X1 $\overline{\wedge}$ X2
VHDL		`Y <= not X1 and X2;`	`Y <= not X1;`	`Y <= X1 xor X2;`	`Y <= X1 nand X2;`
Schaltsymbol DIN 40900					
DIN (alte Norm)					
ANSI / IEEE (alte Norm)					

Bezeichnung			UND, AND	Äquivalenz, XNOR	Identität (X1)	Implikation
Wahrheitstabelle	X2	X1	Y	Y	Y	Y
	0	0	0	1	0	1
	0	1	0	0	1	1
	1	0	0	0	0	0
	1	1	1	1	1	1
Gleichung UND/ODER/NICHT			$Y = X1 \wedge X2$	$Y = (X1 \wedge X2) \vee ((\neg X1) \wedge (\neg X2))$	$Y = X1$	$Y = X1 \vee (\neg X2)$
Operator			$X1 \wedge X2$, $X1 \cdot X2$, $X1X2$	$X1 \leftrightarrow X2$, $X1 \equiv X2$	X1	$X2 \rightarrow X1$
VHDL			`Y <= X1 and X2;`	`Y <= X1 xnor X2;`	`Y <= X1;`	`Y <= X1 or not X2;`
Schaltsymbol DIN 40900						
DIN (alte Norm)						
ANSI / IEEE (alte Norm)						

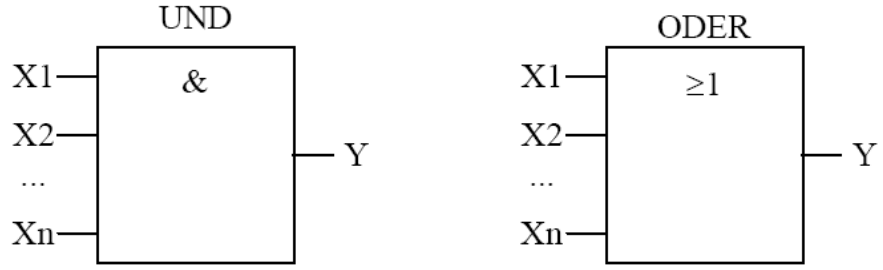
Bezeichnung			Identität (X2)	Implikation	ODER, OR	Konstante 1
Wahrheitstabelle	X2	X1	Y	Y	Y	Y
	0	0	0	1	0	1
	0	1	0	0	1	1
	1	0	1	1	1	1
	1	1	1	1	1	1
Gleichung UND/ODER/NICHT			$Y = X2$	$Y = (\neg X1) \vee X2$	$Y = X1 \vee X2$	$Y = 1$
Operator			X2	$X1 \rightarrow X2$	$X1 \vee X2$, $X1+X2$	1
VHDL			`Y <= X2;`	`Y <= not X1 or X2;`	`Y <= X1 or X2;`	`Y <= '1';`
Schaltsymbol DIN 40900						
DIN (alte Norm)						
ANSI / IEEE (alte Norm)						

3.1.3 Boolesche Funktionen mehrerer Variabler

Eine boolesche Funktion f mit n Argumenten Xi besitzt die folgende generische Wahrheitstabelle, in der jeder Funktionswert fi entweder den Wert 0 oder 1 annehmen kann:

X_{n-1}	X_{n-2}	...	X_1	X_0	$Y=f(X_{n-1}, X_{n-2}, ..., X_1, X_0)$
0	0	...	0	0	f_0
0	0	...	0	1	f_1
0	0		1	0	f_2
...	...		...	...	...
1	1	...	1	0	f_{2^n-2}
1	1	...	1	1	f_{2^n-1}

Es ist ersichtlich, daß eine boolesche Funktion mit n Argumenten durch genau 2^n Funktionswerte fi beschrieben wird. Genau auf diese Art und Weise spezifizieren Wahrheitstabellen eine Funktion. Da jeder Funktionswert fi den Wert 0 oder 1 annehmen kann, gibt es genau 2(2↑n) unterschiedliche Funktionen mit n Argumenten.
Für n=1 ergeben sich damit 4 Funktionen und für n=2 ergeben sich 8 Funktionen.
Die Digitaltechnik verwendet auch bei n >= 3 die bereits vorgestellten Schaltsymbole der UND- und ODER Funktion mit Erhöhung der Anzahl der Eingänge:



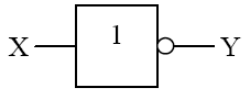
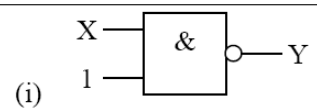
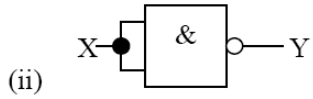
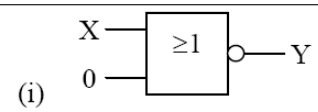
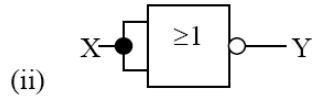
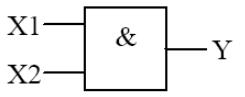
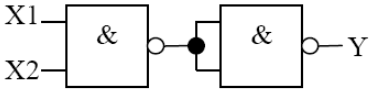
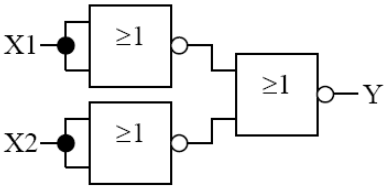
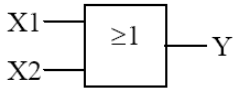
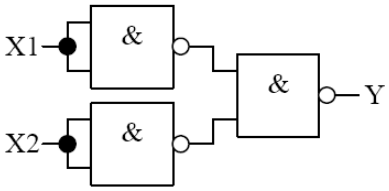
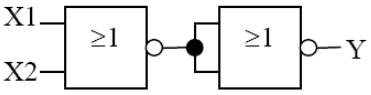
4 Vollständige Systeme aus UND, ODER, NICHT

Bisher wurden die booleschen Funktionen durch die Grundverknüpfungen UND, ODER und NICHT dargestellt. Diese Verknüpfungen besitzen eine besondere Bedeutung, da man mathematisch nachweisen kann, daß sich mit diesen drei Operationen alle booleschen Funktionen darstellen lassen. Man bezeichnet daher diese Verknüpfungen auch als vollständiges System. Für den Schaltungsentwurf hat dies die Attraktivität, daß lediglich drei Grundschaltungen benötigt werden, welche diese drei Operationen in Hardware realisieren. Durch Replizieren und Kombination dieser drei Grundschaltungen kann jede boolesche Schaltung aufgebaut werden.

5 Vollständige Systeme mit NAND- und mit NOR-Operator

Allein die NAND-Verknüpfung bildet ebenfalls ein vollständiges System. Man kann dies damit beweisen, daß man mit dem NAND-Operator die Grundverknüpfungen UND, ODER

und NICHT nachbildet. Das gleiche gilt für die NOR-Verknüpfung. Nachfolgende Tabelle zeigt die Realisierung der Grundverknüpfungen durch NAND und NOR.

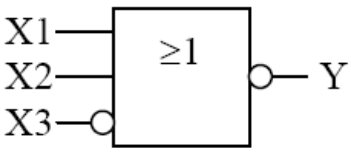
Grundverknüpfung	Realisiert mit NAND	Realisiert mit NOR
Negation, NICHT  $Y = \overline{X}$	(i)  (ii)  (i) $Y = X \overline{\wedge} 1$ (ii) $Y = X \overline{\wedge} X$	(i)  (ii)  (i) $Y = X \overline{\vee} 0$ (ii) $Y = X \overline{\vee} X$
Konjunktion, UND  $Y = X1 \wedge X2$	 $Y = (X1 \overline{\wedge} X2) \overline{\wedge} (X1 \overline{\wedge} X2)$	 $Y = (X1 \overline{\vee} X1) \overline{\vee} (X2 \overline{\vee} X2)$
Disjunktion, ODER  $Y = X1 \vee X2$	 $Y = (X1 \overline{\wedge} X1) \overline{\wedge} (X2 \overline{\wedge} X2)$	 $Y = (X1 \overline{\vee} X2) \overline{\vee} (X1 \overline{\vee} X2)$

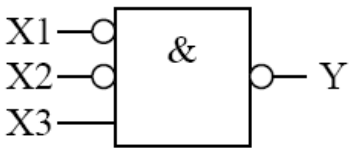
Somit reicht es für die Entwicklung einer booleschen Hardware aus, entweder eine Grundschaltung für NAND oder für NOR zur Verfügung zu stellen. Daraus kann durch Replizieren und Kombinieren jede boolesche Schaltung entwickelt werden.

5.1.1 Mehrstufige Schaltungen

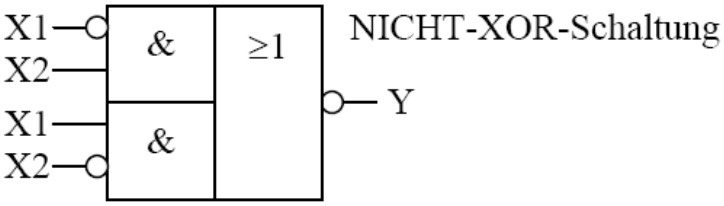
Zum praktischen Aufbau digitaler Schaltungen werden Grundschaltungen der vorgestellten Operatoren verwendet. Diese Grundschaltungen werden als Gatter oder Logikgatter bezeichnet. Eigenschaft eines Logikgatters ist, daß es die Eingangssignale gemäß der spezifizierten Funktion verknüpft und das Verknüpfungsergebnis zeitverzögert auf dem Ausgang ausgibt. Zur Realisierung komplexer boolescher Funktionen werden einfache Logikgatter kombiniert.
n-stufige Logik: Wenn bei einer aus Gattern aufgebauten Logikschaltung n Gatter zwischen Ein- und Ausgängen liegen, wird die Schaltung als n-stufige Logik bezeichnet. Mögliche Negationen an Ein- oder Ausgängen werden in der Zählung nicht berücksichtigt.

Beispiele: einstufige Logik





Beispiel: zweistufige Logik



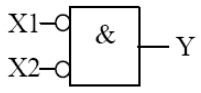
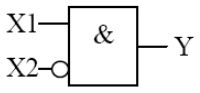
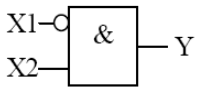
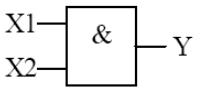
Da sich die Verzögerungszeiten der Gatter zwischen den Ein- und Ausgängen aufsummieren, sollte bei schnellen Schaltungen die Anzahl der Stufen so gering wie möglich gehalten werden. Einstufige Schaltungen ermöglichen jedoch nur die Realisierung weniger Funktionen. Daher wird für schnelle Schaltungen zweistufige Logik eingesetzt.
Ein Maß für den Aufwand bzw. Komplexität einer Schaltung ist die Anzahl der Gattereingänge. Dabei summiert man die Eingänge aller Gatter der n Schaltungsstufen.
Die Gesamtanzahl der Eingänge ergibt das Maß für die Schaltungskomplexität.

5.1.2 Normalformen

Normalformen boolescher Funktionen dienen dazu, eine beliebige Funktion in einheitlicher Form zu beschreiben. Jeder Teilausdruck (bzw. jede Klammer) der Schaltfunktion enthält alle Eingangsvariablen (ist also Min oder Maxterm).
In der booleschen Algebra sind zwei Normalformen gebräuchlich, die Disjunktive- und die Konjunktive Normalform. Diese Normalformen basieren auf Mintermen und Maxtermen.

5.1.3 Minterm(Vollkonjunktion)

Konjunktion, in der alle Variablen (bejaht oder negiert) einer Funktion vorkommen.
Für n=2 Eingangsvariable ergeben sich somit 4 Minterme:

Minterm-Bezeichnung		m_0	m_1	m_2	m_3
Funktionstabelle	X2 X1				
	0 0	1	0	0	0
	0 1	0	1	0	0
	1 0	0	0	1	0
	1 1	0	0	0	1
Gleichung		$m_0 = \overline{X1} \wedge \overline{X2}$	$m_1 = X1 \wedge \overline{X2}$	$m_2 = \overline{X1} \wedge X2$	$m_3 = X1 \wedge X2$
Schaltsymbol					

Minterme sind somit boolesche Funktionen, die genau für eine Kombination der Eingänge den Ausgangswert '1' und für alle anderen Kombinationen den Wert '0' annehmen.
Da eine Funktion mit n Eingangsvariablen genau 2^n mögliche Eingangskombinationen besitzt, existieren für n Eingangsvariable auch genau 2^n Minterme.

Bemerkung: Die Indizierung der Minterme ergibt sich aus dem Wert der Eingangskombination, bei der die Minterm-Funktion den Wert '1' annimmt. Beispielsweise wird bei n=3 der Minterm $(\neg X3)\wedge X2\wedge X1$ mit m3 bezeichnet, denn bei der Eingangskombination "011" gibt die Minterm-Funktion den Wert '1' aus. Da (011)2 = (3)10, ergibt sich der Indexwert 3 in der Minterm-Bezeichnung.

5.1.4 Maxterm(Volldisjunktion):

Disjunktion, in der alle Variablen (bejaht oder negiert) einer Funktion vorkommen.
Dualität zu den Mintermen. Die zugehörigen Maxterm-Funktionen besitzen bei genau einer Eingangskombination den Ausgangswert '0', bei allen übrigen 2^n-1 Eingangskombinationen gibt die Funktion den Wert '1' aus. Die Bezeichnung der Maxterme erfolgt mit Mi. Der Index i ergibt sich aus der Eingangskombination, welche in der Maxterm-Funktion den Ausgangswert '0' erzeugt. Bei n Eingangsvariablen gibt es 2^n unterschiedliche Maxterme (entsprechend wie auch bei den Mintermen).

Beispiel: 3 Eingänge: X3, X2 und X1, Maxterm: $X3\vee(\neg X2)\vee X1$
Funktionstabelle der Maxterm-Funktion:

X3	X2	X1	Y	
0	0	0	1	
0	0	1	1	
0	1	0	0	← Eingangskombination "010" erzeugt Ausgangswert '0'
0	1	1	1	⇒ Maxterm M_2
1	0	0	1	
1	0	1	1	
1	1	0	1	
1	1	1	1	

5.1.5 Disjunktive Normalform (DNF)

Eine beliebige boolesche Funktion lässt sich durch die disjunktive Verknüpfung geeigneter Minterme realisieren. Bei vorgegebener Funktion nimmt man genau die Minterme, welche den Wert '1' an den Stellen erzeugen, an denen die Funktion '1'-Werte ausgibt. Durch die disjunktive Verknüpfung dieser Minterme erhält man genau die vorgegebene Funktion.
Diese Form der Beschreibung einer booleschen Funktion bezeichnet man als Disjunktive Normalform.

Beispiel: DNF-Darstellung einer Funktion mit n=3 Eingangsvariablen.

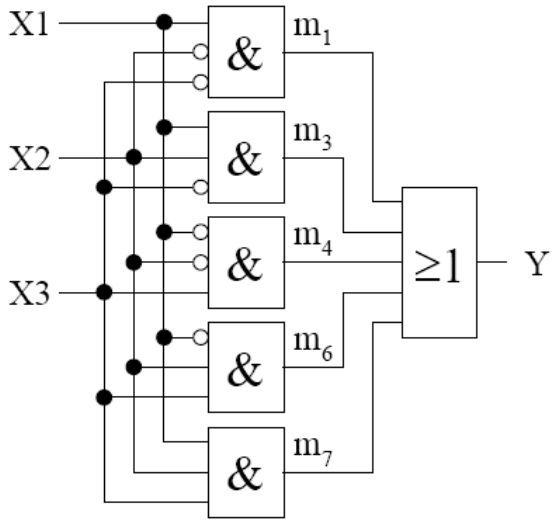
Spezifikation der Funktion durch eine Funktionstabelle:

X3	X2	X1	Y	
0	0	0	0	
0	0	1	1	←
0	1	0	0	
0	1	1	1	←
1	0	0	1	←
1	0	1	0	
1	1	0	1	←
1	1	1	1	←

Die Funktion kann somit durch die folgende Gleichung realisiert werden:

$$Y = m_1 \vee m_3 \vee m_4 \vee m_6 \vee m_7$$
$$= (\overline{X3} \wedge \overline{X2} \wedge X1) \vee (\overline{X3} \wedge X2 \wedge X1) \vee (X3 \wedge \overline{X2} \wedge \overline{X1}) \vee$$
$$(X3 \wedge X2 \wedge \overline{X1}) \vee (X3 \wedge X2 \wedge X1)$$

Die Funktion lässt sich durch die folgende Schaltung beschreiben, die eine zweistufige UND/ODERStruktur
aufweist:



5.1.6 Konjunktive Normalform

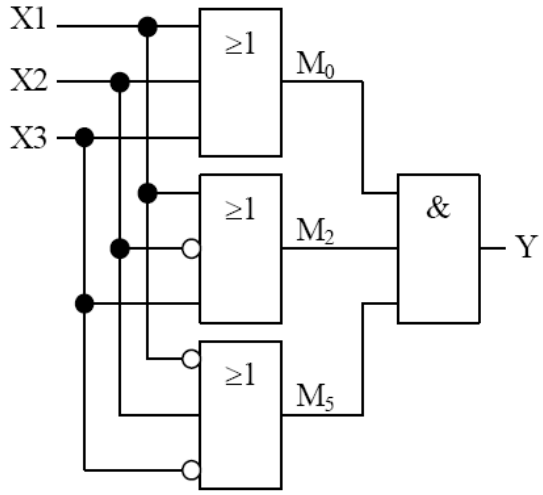
Eine beliebige boolesche Funktion lässt sich durch konjunktive Verknüpfung geeigneter Maxterme beschreiben. Für jede Eingangskombination, an der die Funktion den Ausgangswert '0' produziert, verknüpft man die zugehörigen Maxterme mittels Konjunktion. Die resultierende Beschreibung wird als Konjunktive Normalform
bezeichnet.
Beispiel: KNF-Darstellung einer Funktion (gleiche Funktion wie bei der DNF):

X3	X2	X1	Y	
0	0	0	0	←
0	0	1	1	
0	1	0	0	←
0	1	1	1	
1	0	0	1	
1	0	1	0	←
1	1	0	1	
1	1	1	1	

Die Funktion kann somit durch die folgende Gleichung realisiert werden:

$$Y = M_0 \wedge M_2 \wedge M_5$$
$$= (X3 \vee X2 \vee X1) \wedge (X3 \vee \overline{X2} \vee X1) \wedge (\overline{X3} \vee X2 \vee \overline{X1})$$

Die Funktion lässt sich durch die folgende Schaltung beschreiben, die eine zweistufige ODER/UND Struktur aufweist:



5.1.7 Minimierungsverfahren

Jede Boolsche Funktion ist algebraisch in kanonischer Form darstellbar. Diese Ausdrücke können im allgemeinen weiter vereinfacht werden, in die konjunktive oder die disjunktive Minimalform. Hierzu sind mehrere Verfahren bekannt, die alle auf der Eliminierung von Nachbarschaftsvariablen aus benachbarten Min- oder Maxtermen beruhen.

5.1.7.1 Boolesche Algebra:

Umformen einer vorgegebenen Gleichung in die Minimalform mit den vorgestellten Rechenregeln für boolesche Gleichungen. Dieses Vorgehen ist selbst für Gleichungen mit wenigen Unbekannten schwierig. Daher bietet es sich nur in Ausnahmefällen und nur für einfache Gleichungen an.

5.1.7.2 Algorithmische Verfahren:

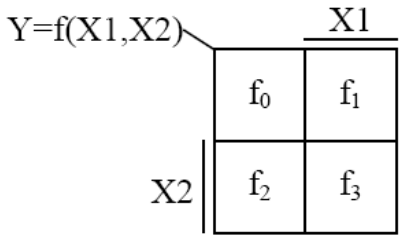
Algorithmische Verfahren fassen Min- bzw. Maxterme sukzessive zusammen, so daß die Terme der ersten Stufe vereinfacht werden. Der bekannteste Algorithmus zur Ermittlung boolescher Minimalformen ist das Quine-McCluskey-Verfahren. Algorithmische Verfahren bilden die Grundlage für die Minimierung boolescher Funktionen mittels Software. Programmpakete zum ASICEntwurf und zur Logikbeschreibung beinhalten algorithmische Verfahren, so daß der Anwender gewünschte Logikfunktionen problemnah spezifizieren kann und durch den Algorithmus eine minimale Implementierung erhält.

5.1.7.3 Grafische Verfahren:

Bei grafischen Verfahren werden boolesche Funktionen geeignet grafisch dargestellt, so daß Terme anschaulich zusammengefaßt und dabei vereinfacht werden können. Das bekannteste grafische Verfahren ist die Minimierung mittels des Karnaugh-Veitch-Diagramms, meist abgekürzt als KV-Diagramm bezeichnet. Dieses Verfahren wird im nachfolgenden Abschnitt im Detail vorgestellt.

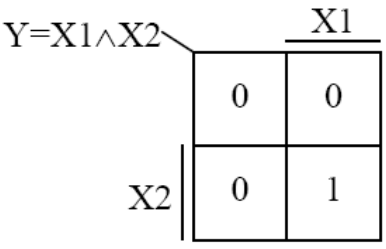
5.1.8 KV-Diagramm

KV-Diagramm für zwei Eingangsvariable
Eine Funktion zweier boolescher Variablen lässt sich in folgender Form grafisch anschaulich darstellen:



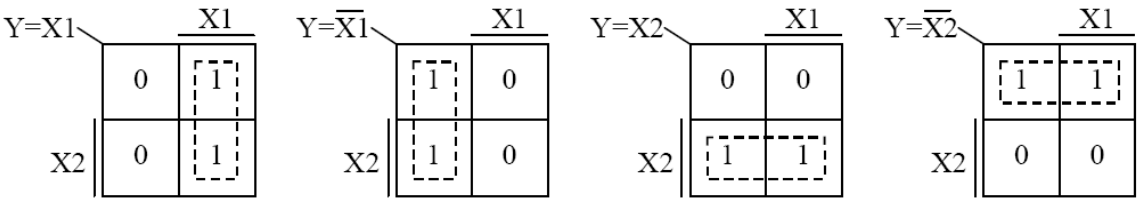
X2	X1	Y
0	0	f_0
0	1	f_1
1	0	f_2
1	1	f_3

In dem Rechteck auf der linken Seite soll in der linken Spalte X1=0 und in der rechten Spalte X1=1 gelten. Daher ist die rechte Spalte mit X1 markiert. Ebenso soll in der obersten Zeile X2=0 und in der untersten Zeile X1=1 gelten. Damit ist eine eindeutige Zuordnung der in der rechten Tabelle angeführten Funktionswerte zu den Zeilen und Spalten im Rechteck möglich. Das Rechteck stellt ein KV-Diagramm für eine boolesche Funktion mit zwei Eingängen dar.
Beispiel: Darstellung der UND-Verknüpfung im KV-Diagramm:



X2	X1	Y
0	0	0
0	1	0
1	0	0
1	1	1

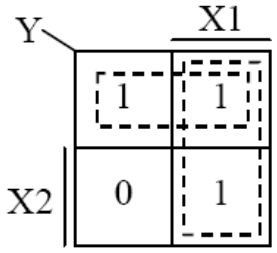
Interessant ist nun die Darstellung der Terme aus einer Eingangsvariablen im KV-Diagramm für 2 Eingangsvariable. Nachfolgende Diagramme zeigen alle Möglichkeiten von Termen aus einer Eingangsvariablen und ihre Darstellung im KV-Diagramm:



Man erkennt, daß diese Terme aus einer Eingangsvariablen zwei benachbarte Felder im KV-Diagramm belegen. Entsprechend belegt eine Funktion mit keiner Eingangsvariablen (konstanter Wert) 4 Felder im KV-Diagramm.
Die Kombination der einfachen Terme ermöglicht die Beschreibung aufwendiger Funktionen.
Beispiel: Minimalform der Funktion aus 3 Mintermen:

$Y = (\neg X1 \wedge \neg X2) \vee (X1 \wedge \neg X2) \vee (X1 \wedge X2)$ = m0+m1+m3

Aus den gezeigten Termen einer Eingangsvariablen lassen sich die aus 3 Mintermen bestehenden Funktionen zweier Eingangsvariablen mittels ODER-Verknüpfung konstruieren.

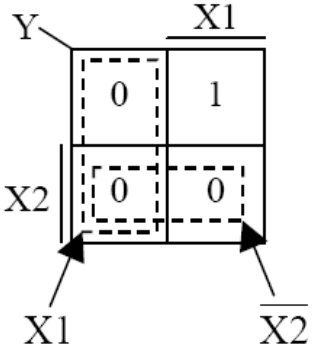


X2	X1	Y
0	0	1
0	1	1
1	0	0
1	1	1

Das Beispiel zeigt, daß die gewünschte Funktion durch ODER-Verknüpfung der Terme X1 und X2 erzielt werden kann. Somit kann die gewünschte Funktion durch die folgende Gleichung realisiert werden:

$Y = X1 \vee \overline{X2}$

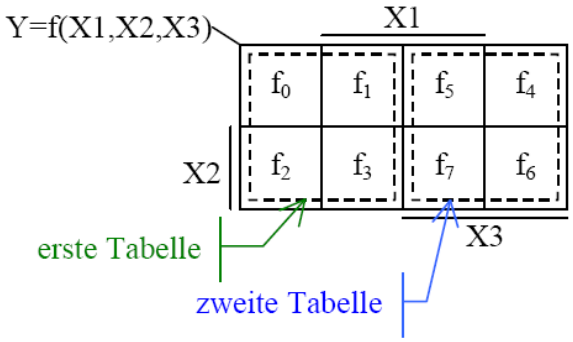
Diese Form ist die disjunktive Minimalform der Funktion.
Das KV-Diagramm kann sowohl, wie oben gezeigt, zur Konstruktion disjunktiver Minimalformen verwendet werden, ebenso aber auch zur Konstruktion konjunktiver Minimalformen. Im Falle konjunktiver Minimalformen werden jedoch die 0-Felder im KV-Diagramm zusammengefaßt.



X2	X1	Y
0	0	0
0	1	1
1	0	0
1	1	0

$Y = X1 \wedge \overline{X2}$

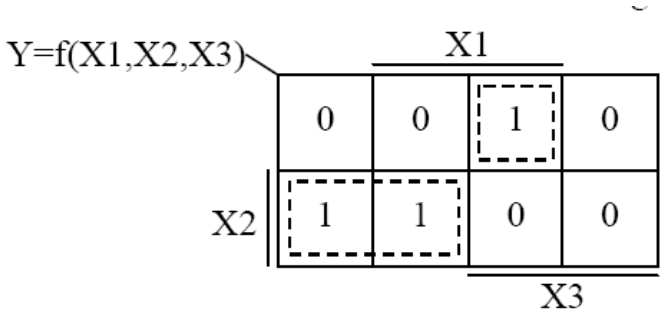
KV-Diagramm für 3 Eingangsvariable
Das KV-Diagramm soll nun auf 3 Eingangsvariable erweitert werden.
Man erreicht dies durch Kombinieren zweier Tabellen aus zwei Eingangsvariablen X1, X2. Die zweite Tabelle wird dabei gespiegelt dargestellt. Die erste Tabelle ist gültig, wenn die dritte Eingangsvariable X3 den Wert 0 annimmt, die zweite Tabelle gilt entsprechend beim Wert 1 von X3:



X3	X2	X1	Y
0	0	0	f_0
0	0	1	f_1
0	1	0	f_2
0	1	1	f_3
1	0	0	f_4
1	0	1	f_5
1	1	0	f_6
1	1	1	f_7

Beispiel: Gegeben sei die folgende Funktion aus drei Eingangsvariablen in Tabellenform. Gesucht ist die disjunktive Minimalform der Funktion:

X3	X2	X1	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



Die disjunktive Normalform der Funktion ergibt sich zu:

$$Y = (\overline{X1} \wedge X2 \wedge \overline{X3}) \vee (X1 \wedge X2 \wedge \overline{X3}) \vee (X1 \wedge \overline{X2} \wedge X3)$$

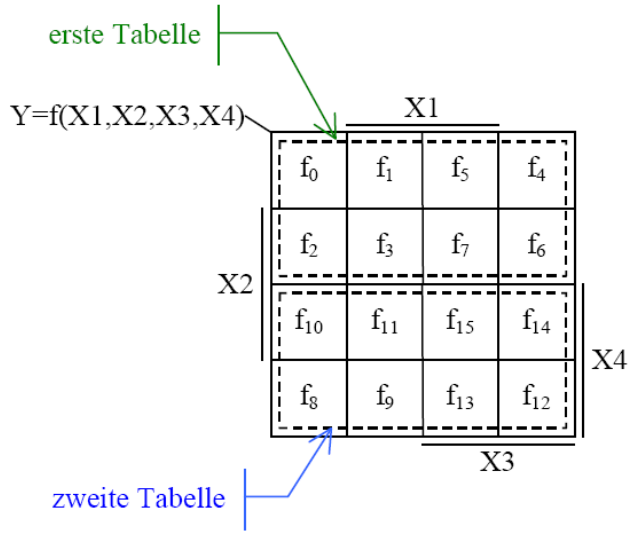
Durch Zusammenfassen

$$Y = (X2 \wedge \overline{X3}) \vee (X1 \wedge \overline{X2} \wedge X3)$$

Beispiel:
Zeichnen Sie ds KV-Diagramm für 3 Variablen mit der DNF = m7 + m3.
Gebens Sie die optimierte Version DMF an, ebenso die KMF.

5.1.8.1 KV-Diagramm für 4 Eingangsvariable

Das Verfahren zur Zusammenfassung zweier kleinerer Tabellen zu einer größeren Tabelle lässt sich fortsetzen. Man nimmt zwei KV-Diagramme mit drei Eingangsvariablen und setzt sie wieder gespiegelt zusammen:



5.1.8.2 Teilweise definierte Funktionen

Häufig liegt der Fall vor, daß eine Funktion nur für einige mögliche Wertekombinationen erklärt ist und der Funktionswert für die restlichen Kombinationen beliebig (don't care) ist. Mann setzt dann zunächst die definierten Funktionswerte in das KV-Diagramm ein und wählt die freien Werte so aus, daß möglichst große Untertafeln entstehen.
Beispiel : Zeichnen sie das KV-Diagramm für 3 Eingangsvariablen mit der DNF = m1 + m3 + m7 + d2 + d5. Welche Werte von ‚d' führen zur besten Optimierung?

5.1.8.3 Implikanten

Zur Minimierung von Funktionen bildet man zunächst geeignete Implikanten einer Funktion und faßt diese dann zu einer Gleichung zusammen.

5.1.8.4 Definitionen Implikant:

Faßt man Min- bzw. Maxterme einer Funktion so zusammen, daß deren Verbund durch einen Term geringerer Komplexität, d.h. mit einer reduzierten Anzahl von Eingangsvariablen, beschrieben werden kann, so wird der resultierende Term als Implikant bezeichnet. Die Anzahl der in einem Implikanten zusammengefaßten Min- bzw. Maxterme bildet eine 2er-Potenz. Somit können auch einzelne Min- und Maxterme als Implikanten interpretiert werden.

5.1.8.5 Primimplikant:

Ist ein Implikant einer booleschen Funktion in keinem anderen Implikanten vollständig enthalten, wird er als Primimplikant bezeichnet.

Bezeichnung für die so groß wie möglich gewählten Blöcke von „Einsen" im Karnaugh-Diagramm

5.1.8.6 Kern-Primimplikant (essentieller PI):

Enthält ein Primimplikant (PI) mindestens einen Min- oder Maxterm, der in keinem anderen Primimplikanten enthalten ist, bezeichnet man diesen als Kern-Primimplikanten.

Primimplikanten, die mindestens eine „Eins“ enthalten, die sonst von keinem anderen Block abgedeckt sind -> die minimale Lösung enthält zumindest die
essentiellen Primimplikanten

5.1.8.7 Nicht-essentielle PI:

Primimplikanten, die obige Bedingung nicht erfüllen

5.1.8.8 Redundante PI:

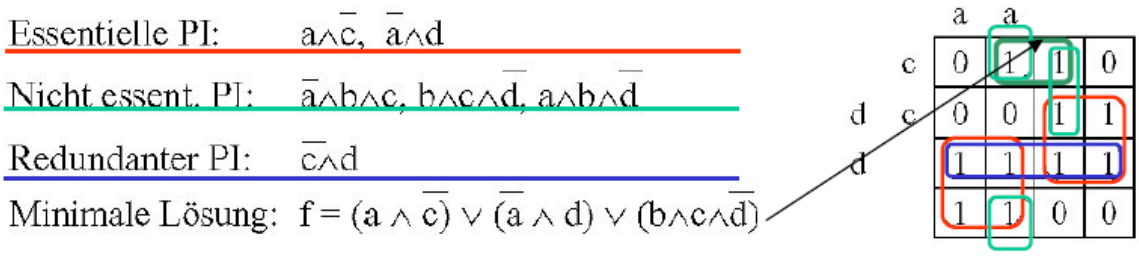
Nicht essentielle PI, die nur bereits von essentiellen PI
abgedeckte „Einsen“ bzw. „Nullen“ markieren -> überflüssig

Beispiel: Gegeben sei die nachfolgend als Tabelle vereinbarte Funktion. Gesucht werden Implikanten der
Funktion.

X3	X2	X1	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

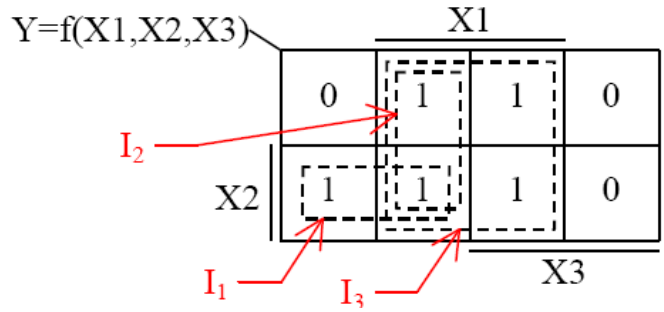
Minterm	I_1	I_2	I_3
$m_1 = (\neg X3)\wedge(\neg X2)\wedge X1$		x	x
$m_2 = (\neg X3)\wedge X2\wedge(\neg X1)$	x		
$m_3 = (\neg X3)\wedge X2\wedge X1$	x	x	x
$m_5 = X3\wedge(\neg X2)\wedge X1$			x
$m_7 = X3\wedge X2\wedge X1$			x

Faßt man die Minterme m2 und m3 zusammen, so lässt sich der resultierende Term mit
$I1 = (\neg X3)\wedge X2$
beschreiben. Dieser Term ist ein Implikant. Er besteht aus $2 = 2^1$ Mintermen. Weiter lassen sich in der vorgegebenen Funktion die Minterme m1 und m3 zu einem zweiten Implikanten
$I2 = (\neg X3)\wedge X1$
zusammenfassen, der wiederum aus 2^1 Mintermen besteht. Ebenso bilden die Minterme m1, m3, m5, m7 einem dritten Implikanten
I3 = X1,
der aus $4 = 2^2$ Mintermen besteht. Man erkennt, daß der aus m1 und m3 gebildete, zweite Implikant I2 vollständig in dem aus m1, m3, m5 und m7 gebildeten dritten Implikanten I3 enthalten ist. Der erste und der dritte Implikant I1 und I3 sind hingegen in keinem anderen Implikanten vollständig enthalten. Diese sind somit Primimplikanten. Somit kann im Beispiel die vorgegebene Funktion allein durch den ersten und dritten Implikanten beschrieben werden:
$Y = I1 \veebar I3 = ((\neg X3)\wedge X2) \veebar X1$
Weiteres Beispiel

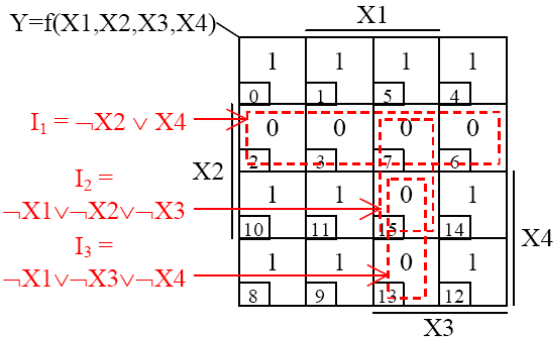


5.1.8.9 Lokalisierung von Implikanten im KV-Diagramm

Im KV-Diagramm können Implikanten sehr einfach lokalisiert werden. Sie bilden sich in kompakter Form auf benachbarte Felder ab. Wenn man somit 2^i benachbarte Felder (i beliebig) kompakt zusammenfassen kann, erhält man damit einen Implikanten. Lässt sich ein Implikant nicht mehr mit benachbarten Implikanten im KV-Diagramm zusammenfassen, ist dieser ein Primimplikant. Beispiel: Darstellung der Funktion des vorhergehenden Beispiels im KV-Diagramm und Lokalisierung von Implikanten zur Bildung der disjunktiven Minimalform:



Man erkennt im KV-Diagramm deutlich, daß I2 vollständig in I3 enthalten ist. Man benötigt somit nur I1 und I3 zur Beschreibung der Funktion. I1 und I3 sind Primimplikanten. Sie sind darüber hinaus Kern-Primimplikanten, denn der Minterm m2 ist nur in I1 und die Minterme m3, m5 und m7 sind nur in I3 enthalten.
Beispiel: Ermittlung der Primimplikanten einer Funktion mit 4 Eingangsvariablen zur Bildung der konjunktiven Minimalform:



X4	X3	X2	X1	Y	I_1	I_2	I_3
0	0	0	0	1			
0	0	0	1	1			
0	0	1	0	0	M_2		
0	0	1	1	0	M_3		
0	1	0	0	1			
0	1	0	1	1			
0	1	1	0	0	M_6		
0	1	1	1	0	M_7	M_7	
1	0	0	0	1			
1	0	0	1	1			
1	0	1	0	1			
1	0	1	1	1			
1	1	0	0	1			
1	1	0	1	0			M_{13}
1	1	1	0	1			
1	1	1	1	0		M_{15}	M_{15}

Zur der Bildung der konjunktiven Minimalform müssen die Maxterme zu Implikanten zusammengefaßt werden.
Man erkennt im KV-Diagramm, daß die vorgegebene Funktion 3 Primimplikanten enthält, die im Diagramm mit I1, I2 und I3 gekennzeichnet sind:

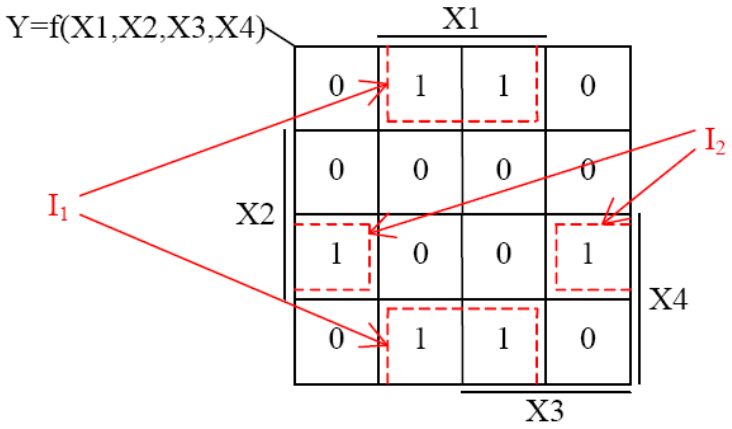
- Der Primimplikant I1 besteht aus den Maxtermen M2, M3, M6 und M7. Dabei sind die Maxterme M2, M3 und M6 nur in I1 enthalten, somit handelt es sich um einen Kern-Primimplikanten.
- Der Primimplikant I2 besteht aus den Maxtermen M7 und M15. Da M7 auch in I1 und M15 auch in I3 enthalten ist, ist I2 kein Kern-Primimplikant.
- Der Primimplikant I3 besteht aus den Maxtermen M13 und M15. Der Maxterm M13 ist nur in I3 enthalten, daher ist I3 ein Kern-Primimplikant.

Zur Darstellung aller Maxterme werden im Beispiel nur die beiden Kern-Primimplikanten I1 und I3 benötigt. Damit ergibt sich die konjunktive Minimalform zu:

$Y = I1 \wedge I3 = ((\neg X2) \vee X4) \wedge ((\neg X1) \vee (\neg X3) \vee (\neg X4))$

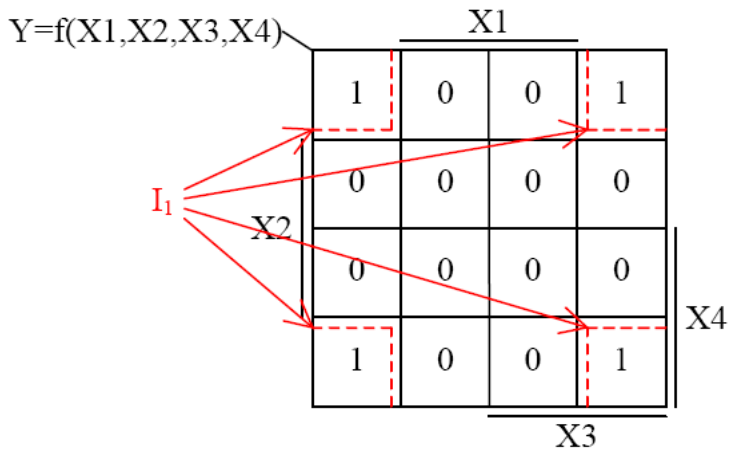
Die Nachbarschaft von Feldern basiert im KV-Diagramm auf einer Torus-Topologie. Dies bedeutet, daß Felder in der untersten Reihe Nachbarn der Felder in der obersten Reihe sind. Gleiches gilt auch für die Felder der rechten und linken Seite. Nachfolgendes Beispiel verdeutlicht diese Nachbarschaften.

Beispiel: Nachbarschaften am Rand des KV-Diagramms
Nachfolgende Funktion besitzt zwei Kern-Primimplikanten, deren Felder Nachbarschaften am Rand des gezeigten KV-Diagramms besitzen. Somit belegt der Primimplikant I1 Felder am oberen und am unteren Rand des Diagramms. Entsprechend belegt I2 Felder am linken und



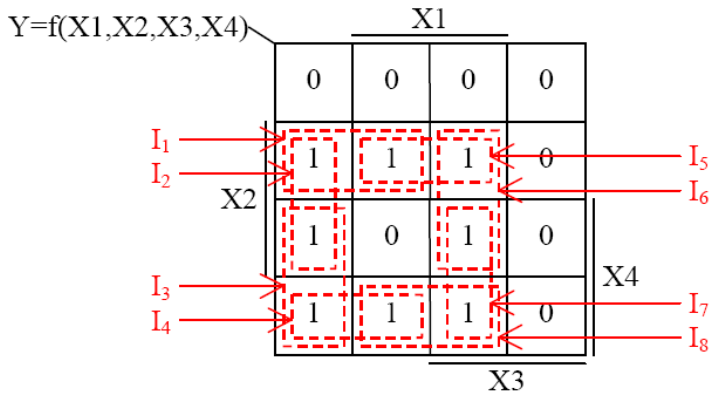
am rechten Rand.
Der Primimplikant I1 wird durch den Term (X1∧ (¬X2)) beschrieben, I2 durch den Term ((¬X1)∧X2∧X4). Damit erhält man als disjunktive Minimalform die Gleichung:
Y=(X1∧(¬X2)) ∨ ((¬X1)∧X2∧X4).

Beispiel: Nachbarschaften am Rand des KV-Diagramms
Nachfolgende Funktion besitzt nur einen Primimplikanten in den Ecken des KV-Diagramms:

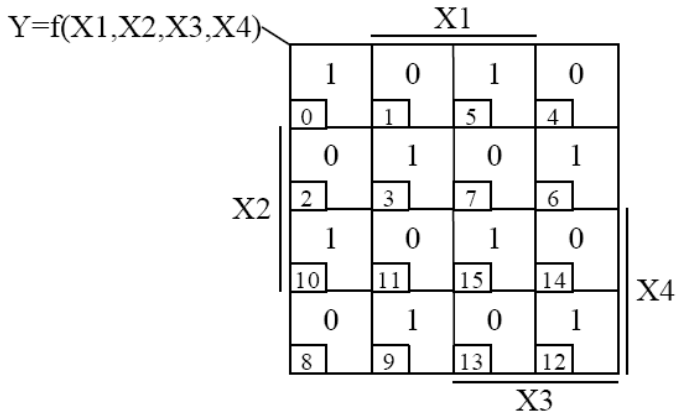


Y = ((¬X1)∧(¬X2)).

Beispiel: Boolesche Funktion ohne Kern-Primimplikanten
Die nachfolgend im KV-Diagramm spezifizierte boolesche Funktion besitzt keine Kern-Primimplikanten, wenn sie als disjunktive Minimalform realisiert werden soll.



Anwendungsbeispiel: Zur Datensicherung im Speicher soll die gerade Parität für 4 Bit Worte berechnet werden. Dies bedeutet, daß eine boolesche Funktion den Wert 1 liefert, wenn eine gerade Anzahl von Bits des betrachteten Wortes den Wert 1 annehmen. Anderenfalls soll die boolesche Funktion den Wert 0 liefern.



Beispiel: Gray-Code Inkrementierer
Der Gray-Code kodiert ganze Zahlen so in einer binären Darstellung, daß benachbarte Werte sich nur in einem einzelnen Bit unterscheiden.
Für einen 3-Bit Gray-Code soll eine Schaltung entworfen werden, welche einen Wert in Gray-Kodierung am Eingang verarbeitet und den nachfolgenden Wert in Gray-Kodierung ausgibt.
Der 2-Bit Gray-Code für die Werte 0 bis 3 ergibt sich zu:

Wert	0	1	2	3
Gray-Code	00	01	11	10

Man erkennt, daß sich die benachbarten Werte nur in einem Bit unterscheiden. Auch unterscheiden sich die Codes für den größten und den kleinsten Wert nur in einem Bit.
Aus dem 2-Bit Gray-Code kann ein 3-Bit Gray-Code entwickelt werden. Für die Werte 0 bis 3 fügt man dem Gray-Code eine 0 hinzu und erweitert ihn so auf 3 Bit. Für die Werte 4 bis 7 verwendet man die 2- Bit Gray-Codes in gespiegelter Reihenfolge und fügt eine 1 als höchstes Bit hinzu. Damit ergibt sich der 3-Bit Gray-Code zu:

Wert	0	1	2	3	4	5	6	7
Gray-Code	0 00	0 01	0 11	0 10	1 10	1 11	1 01	1 00

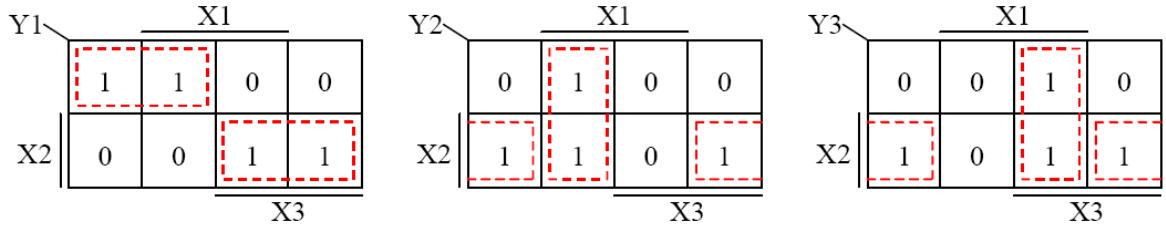
Dieses Spiegeln des Codes mit Hinzufügen der Binärziffer 0 für die untere Hälfte und 1 für die obere Hälfte lässt sich solange fortsetzten, bis man einen Gray-Code erzeugt hat, der den gewünschten Wertevorrat besitzt.
Nachfolgend ist zur Verifikation des Verständnisses noch der 4-Bit Gray-Code aufgeführt:

Wert	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Gray-Code	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000

Die Wahrheitstabelle für den geforderten 3-Bit Gray-Code Inkrementierer ergibt sich also wie folgt:

X3	X2	X1	Y3	Y2	Y1
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Dies lässt sich in drei KV-Diagrammen darstellen:



$Y1 = ((\neg X2) \wedge (\neg X3)) \vee (X2 \wedge X3)$
$Y2 = (X1 \wedge (\neg X3)) \vee ((\neg X1) \wedge X2)$
$Y3 = (X1 \wedge X3) \vee ((\neg X1) \wedge X2)$

5.1.9 Minimierungsverfahren von Quine und Mc Cluskey

Das Minimierungsverfahren von Quine und Mc Cluskey eignet sich für die Implementierung des Minimierungsalgorithmus mit einem Digitalrechner, für die anschauliche Ermittlung der Minimalform einer booleschen Gleichung per Hand ist es hingegen weniger geeignet.
Das Verfahren wird im folgenden nur für die Ermittlung der disjunktiven Minimalform vorgestellt. Es kann entsprechend aber auch zur Ermittlung der konjunktiven Minimalform eingesetzt werden.
Das Verfahren besteht aus den folgenden Schritten:
1. Ermittlung aller Minterme der Funktion.
2. Unterteilung der Minterme in Gruppen. Eine Gruppe faßt Minterme mit gleicher Anzahl negierter Eingangsvariablen zusammen.
3. Paarweise Zusammenfassen von Termen benachbarter Gruppen zu Termen geringerer Komplexität durch Anwendung der 5. Kürzungsregel. Kennzeichnung der zusammengefaßten Terme.
4. Wiederholung von Schritt 3 für die zusammengefaßten Terme, bis keine Vereinfachung mehr durchgeführt werden kann.
Ist keine Vereinfachung mehr möglich, bilden alle nicht gekennzeichneten Minterme und Terme die Primimplikanten der Funktion.
5. Ermittlung der Kern-Primimplikanten aus den gefundenen Primimplikanten. Die Kern-Primimplikanten gehören auf jeden Fall zu den Termen der gesuchten Funktion.
6. Hinzufügen von Primimplikanten zur Funktion, bis alle Minterme der Funktion berücksichtigt sind.

Beispiel: Minimierung nach Quine und Mc Cluskey
Folgende Funktion soll minimiert werden:

X4	X3	X2	X1	Y		Gruppe
0	0	0	0	1	m_0	4
0	0	0	1	0		
0	0	1	0	0		
0	0	1	1	1	m_3	2
0	1	0	0	0		
0	1	0	1	1	m_5	2
0	1	1	0	1	m_6	2
0	1	1	1	1	m_7	1
1	0	0	0	0		
1	0	0	1	0		
1	0	1	0	0		
1	0	1	1	1	m_{11}	1
1	1	0	0	0		
1	1	0	1	0		
1	1	1	0	0		
1	1	1	1	1	m_{15}	0

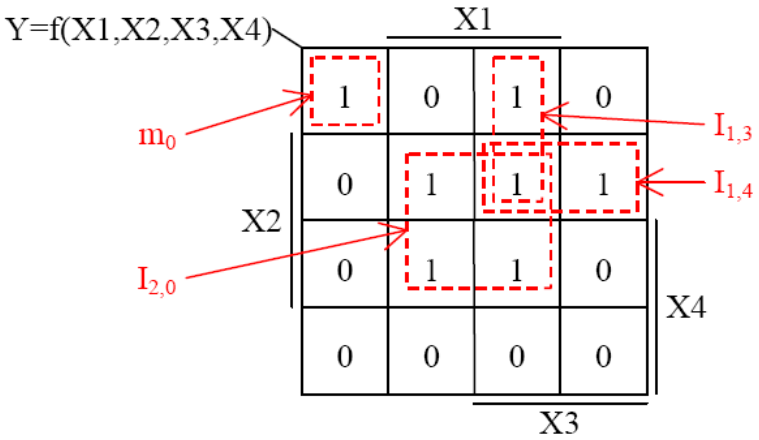
Eintragen der Minterme in die Minimierungstabelle und schrittweises Kürzen ergibt:

Minterme		1. Vereinfachung		2. Vereinfachung
Gruppe 0:				
m_{15}=X4∧X3∧X2∧X1	x	$I_{1,0}$=m_{15}∨m_7=X3∧X2∧X1 $I_{1,1}$=m_{15}∨m_{11}=X4∧X2∧X1	x x	$I_{2,0}$= $I_{1,0}$∨$I_{1,5}$=X2∧X1 ($I_{2,1}$= $I_{1,1}$∨$I_{1,2}$=$I_{2,0}$)
Gruppe 1:				
m_7=¬X4∧X3∧X2∧X1 m_{11}=X4∧¬X3∧X2∧X1	x x	$I_{1,2}$=m_7∨m_3=¬X4∧X2∧X1 $I_{1,3}$=m_7∨m_5=¬X4∧X3∧X1 $I_{1,4}$=m_7∨m_6=¬X4∧X3∧X2 $I_{1,5}$=m_{11}∨m_3=¬X3∧X2∧X1	x x	
Gruppe 2:				
m_3=¬X4∧¬X3∧X2∧X1 m_5=¬X4∧X3∧¬X2∧X1 m_6=¬X4∧X3∧X2∧¬X1	x x x			
Gruppe 3				
Gruppe 4:				
m_0=¬X4∧¬X3∧¬X2∧¬X1				

Zunächst wird die 1. Vereinfachung durchgeführt: Beim Zusammenfassen des Minterms der Gruppe 0 mit den Mintermen der Gruppe 1 entstehen die Terme I1,0 und I1,1. Da sich alle Minterme der beiden Gruppen zusammenfassen lassen, werden alle Minterme in diesen Gruppen markiert (kleines x in der Spalte rechts daneben). Ebenso können Minterme der Gruppe 1 mit Mintermen der Gruppe 2 zusammengefaßt werden, dabei entstehen die Terme I1,2, I1,3, I1,4, und I1,5. Alle Minterme der Gruppe 2 werden bei der Zusammenfassung verwendet, daher werden sie ebenfalls markiert. Minterme der Gruppe 3 sind im Beispiel nicht vorhanden. Der Minterm m0 der Gruppe 4 kann im Beispiel nicht zusammengefaßt werden, daher bleibt er unmarkiert. Damit ist dieser Minterm der erste gefundene Primimplikant der Funktion. Bei der 2. Vereinfachung können die Terme I1,0 und I1,5 zum Term I2,0, zusammengefaßt werden. Die Zusammenfassung der Terme I1,1 und I1,2 liefert den gleichen Term I2,0. Somit werden die Terme I1,0, I1,1, I1,2, und I1,5 markiert. Die unmarkierten Terme I1,3 und I1,4, die sich nicht weiter zusammenfassen lassen, sind zwei weitere Primimplikanten der Funktion. Der Term I2,0 kann nicht mehr zusammengefaßt werden, er bildet den vierten Primimplikanten der Funktion.
Damit besitzt die Funktion vier Primimplikanten: m0, I1,3, I1,4, und I2,0. Es muß nun noch entschieden werden, welche der Primimplikanten für die Minimalform der Funktion benötigt werden. Dazu werden zunächst die Kern-Primimplikanten ermittelt:

Minterm	**m_0**	**$I_{1,3}$**	**$I_{1,4}$**	**$I_{2,0}$**
m_0	x			
m_3				x
m_5		x		
m_6			x	
m_7		x	x	x
m_{11}				x
m_{15}				x

Die Tabelle zeigt, welche Minterme der vorgegebenen Funktion in welchen Primimplikanten enthalten sind. Man erkennt, daß im Beispiel alle Primimplikanten auch Kern-Primimplikanten sind und somit zur minimalen Beschreibung der Funktion benötigt werden.
Man erhält damit die disjunktive Minimalform der Funktion:
$Y = m0 \vee I1,3 \vee I1,4 \vee I2,0$
$= (\neg X4 \wedge \neg X3 \wedge \neg X2 \wedge \neg X1) \vee (\neg X4 \wedge X3 \wedge X1) \vee (\neg X4 \wedge X3 \wedge X2) \vee (X2 \wedge X1)$
Zur Verifikation ist die Funktion nachfolgend noch im KV-Diagramm dargestellt:



Weiteres Beispiel in etwas anderer Formulierung:

x4	x3	x2	x1	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Die kanonische disjunktive Normalform enthält die Minterme m0, m2, m4, m5, m6, m7, m10, m11.

m	Vektor
0	0000
2	0010
4	0100
5	0101
6	0110
7	0111
10	1010

Kombiniert man jeweils benachbarte Terme, so ergeben sich:

m	Vektor
0,2	00-0
0,4	0-00
2,0	wie oben
2,6	0-10
2,10	-010
4,0	wie oben
4,5	010-
4,6	01-0
5,4	wie oben
5,7	01-1
6,2	wie oben
6,4	wie oben
6,7	011-
7,5	wie oben
7,6	wie oben
10,2	wie oben
10,11	101-
11,10	wie oben

Es bleiben also:

m	Vektor
0,2	00-0
0,4	0-00
2,6	0-10
2,10	-010
4,5	010-
4,6	01-0
5,7	01-1
6,7	011-
10,11	101-

Auch hier wieder die Zusammenfassung der Nachbarterme:

m	Vektor
0,2,4,6	0--0
2,10	-010
4,5,6,7	01--
10,11	101-

Die optimierte Funktion lautet dann:
y= ¬x4¬x1 + ¬x3x2¬x1 + ¬x4x3 + x4¬x3x2

Im KV Diagramm kann man erkennen, daß das noch nicht das Optimum ist. Durch die zufällige Streichungsreihenfolge von oben nach unten ist ein essentieller PI in der Funktion verblieben, der bei anderer Reihenfolge gestrichen worden wäre. Durch Betrachtung der ursprünglich enthaltenen Minterme kann man besser optimieren.

m	Vektor	ursprünglich enthalten
0,2,4,6	0—0	0000 0010 0100 0110
2,10	-010	~~0100 1010~~
4,5,6,7	01--	~~0100~~ 0101 ~~0110~~ 0111
10,11	101-	1010 1011

Wenn man nun gezielt die Doppel so streicht, daß eine ganze Zeile überflüssig wird, findet man das tatsächliche Optimum. In diesem Fall würde die zweite Zeile entfallen.

y= ¬x4¬x1 + ¬x4x3 + x4¬x3x2

Das gleiche Beispiel kann mit Maxtermen geübt werden.

5.1.10 Binäre Entscheidungsdiagramme

Boolesche Ausdrücke können auch durch sogenannte geordnete binäre Entscheidungsdiagramme (OBDD Ordered Boolean Decision Diagram) dargestellt werden. Diese sind oft sehr kompakt und lassen sich algorithmisch effizient behandeln.

5.1.10.1 OBDD

Ein OBDD ist ein gerichteter azyklischer Graph mit Wurzel, Zwischenknoten und zwei Endknoten (0-Knoten und 1-Knoten) ohne Ausgangskanten. Die Zwischenknoten sind mit Booleschen Variablen beschriftet. Von jedem Zwischenknoten gehen genau zwei Kanten aus (0-Kante und 1-Kante). Die Variablen sind linear geordnet, z.B. a < b < c <.... , und tauchen in dieser Reihenfolge von der Wurzel beginnend im Graphen auf. D.h., jedem Zwischenknoten ist ein Variablenname zugeordnet, wobei gilt, daß für alle Pfade von der Wurzel des Graphen zu einem der beiden Endknoten die Variablen in der gleichen Reihenfolge auftreten. Es müssen aber nicht in jedem Pfad alle Variablennamen auftreten! (Ordnung).
OBDDs finden in vielen Bereichen der Informatik eine Anwendung. Hier soll nur ein kleines Beispiel erwähnt werden. Der Boolescher Ausdruck

$$Z = (a\bar{b}c + de) \cdot f \cdot (\bar{g} + hi)$$

läßt sich wie folgt zerlegen:

$$Z1 = a\bar{b}c$$

$$Z2 = de$$

$$Z3 = \bar{g}$$

$$Z4 = hi$$

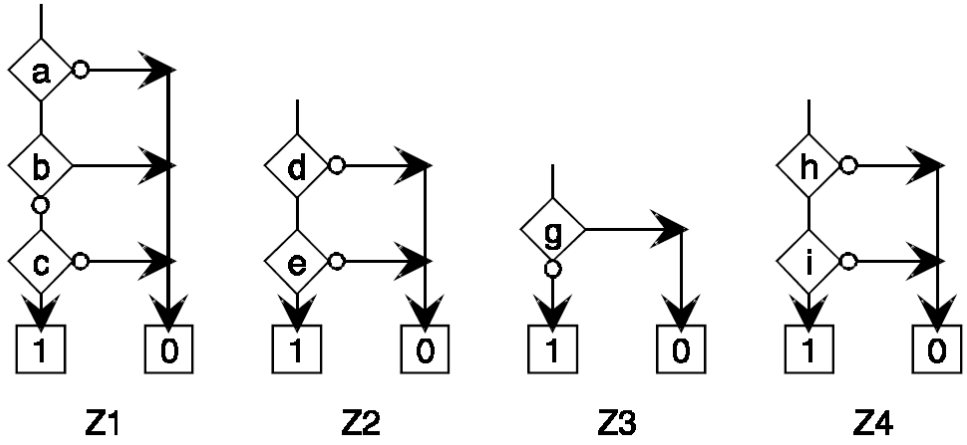
$$Z5 = Z1 + Z2$$

$$Z6 = Z3 + Z4$$

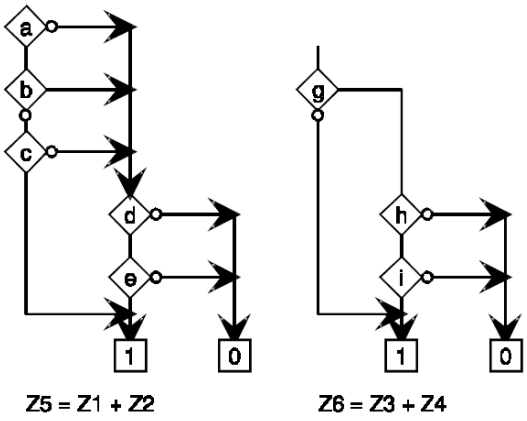
$$Z = Z5 \cdot f \cdot Z6$$

Für Z1 bis Z4 lasen sich sofort OBDDs angeben. Eine Raute steht für den Test einer binären Variablen auf Wert 0 oder 1. Wenn dieser Test (die Entscheidung) ein 1 ergibt, wird längs des Pfeiles (1-Kante); wenn sich eine 0 ergibt, wird längs der Kante mit Kreis (0-Kante) zum

nächsten Test gegangen. Um z.B. zu sehen, welchen Wert der durch den OBDD Z1 dargestellte Boolesche Ausdruck für die Variablenbelegung a = 1, b = 0 und c = 1 hat, durchlaufe man das erste Diagramm entsprechend. (0-Kante mit, 1-Kante ohne Kreis)



Teildiagramme werden wie folgt zusammen gesetzt (Bild 7.2 und 7.3):
Eine ODER-Verknüpfung bedeutet anhängen am 0-Knoten; eine UND-Verknüpfung bedeutet anhängen am 1-Knoten. Damit ist im Prinzip beschrieben, wie man für einen Booleschen Ausdruck eine OBDD von vielen möglichen erhält. In der Literatur findet man geeignetere Algorithmen dazu.



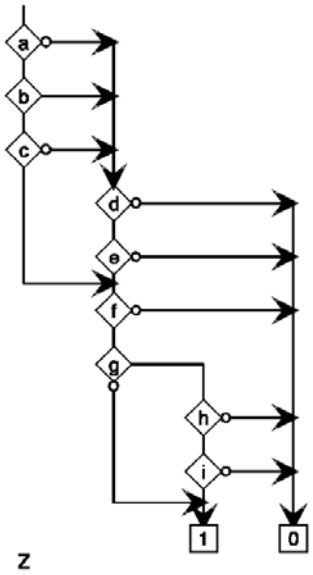
Besonders einfach wird die Komplementbildung eines Booleschen Ausdrucks dann, wenn dieser als OBDD vorliegt. Man vertausche einfach den 0- mit den 1-Knoten. Oft ist es günstig, spezielle OBDDs zu betrachten, die ROBDDs (Reduced Ordered Boolean Decision Diagram): Ein ROBDD ist ein OBDD, für den folgendes gilt:

- Es gibt keine zwei Knoten mit demselben Variablennamen, für die gilt, daß sie dieselben 0- und 1- Nachfolgeknoten haben (Reduziertheit 1).
- Bei keinem der Nicht-Endknoten ist der 0-Nachfolgeknoten identisch zu dem 1-Nachfolgeknoten (Reduziertheit 2).

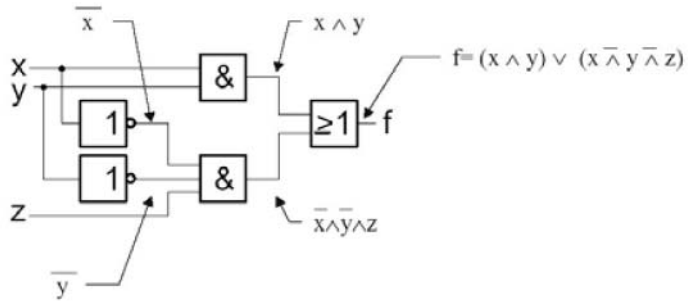
Man erhält den ROBDD eines Booleschen Ausdrucks, indem man das Entscheidungsdiagramm, oder besser noch mit Hilfe des Entwicklungsatzes die Shannonsche Normalform aufstellt, die Endknoten identifiziert und dann die beiden folgenden Reduktionsschritte wiederholt anwendet:

- Von zwei Knoten mit demselben Variablennamen, für die gilt, daß sie dieselben 0- und 1- Nachfolgeknoten haben, entferne einen und lege dessen einlaufende Kanten an den anderen Knoten an.

- Wenn bei einem der Nicht-Endknoten der 0-Nachfolgeknoten identisch zu dem 1-Nachfolgeknoten ist, überbrücke ihn und entferne ihn dann.



Kombinatorische Schaltung (Schaltnetz, Gatterschaltung):
• Ausgangswert = f (Eingangswerte)
• Enthält keine Speicherelemente
• Funktionsbeschreibung durch Funktionstabelle (Wahrheitstabelle, Wertetabelle) oder boolsche Algebra

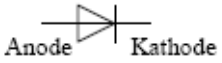


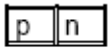
x1	x2	x3	y = f(x1,x2,x3)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

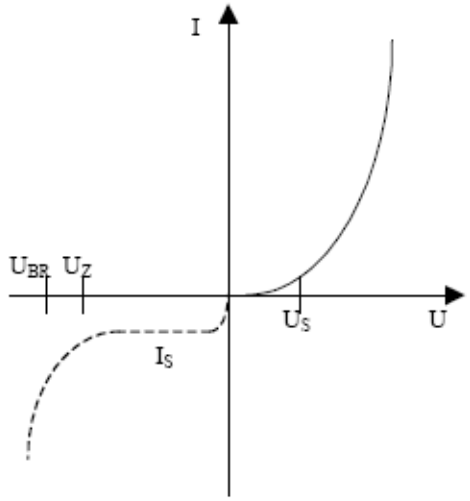
Vorgehen:
1. Problemerfassung mittels Funktionstabelle
(Wahrheitstabelle, Wertetabelle, Schaltbelegungstabelle - SBT)
2. Funktionsgleichungen aufstellen
3. Schaltungsvereinfachung (Minimierung) mit geeigneten
Verfahren (KV-Tafel, Quine-McCluskey, rechnerunterstützt)
4. Schaltungsrealisierung, Bauteileauswahl

6 Dioden als logisches Element

Halbleiterdioden besitzen einen Ventil- oder Gleichrichteffekt ihres pn-Überganges, d.h. ein Stromfluß von Anode zu Kathode ist näherungsweise ideal möglich, während ein Stromfluß umgekehrt näherungsweise versperrt bleibt. Symbol und Kennlinie siehe unten.



Aufbau (Prinzip): 



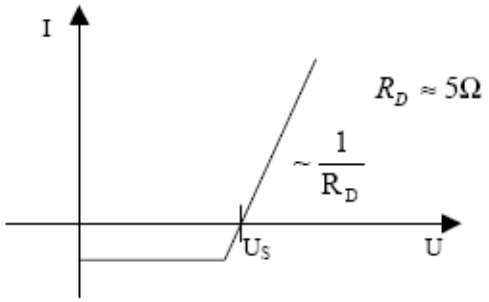
$$I = I_0 \cdot \left(e^{\frac{U}{U_S}} - 1 \right)$$

$$I = I_F - I_S$$

$$U_{SGe} \approx 0{,}3V$$

$$U_{SSi} \approx 0{,}6V$$

Ein stark vereinfachtes Ersatzbild aus Spannungsquelle, idealer Diode und Widerstand ergibt folgende Kennlinie:

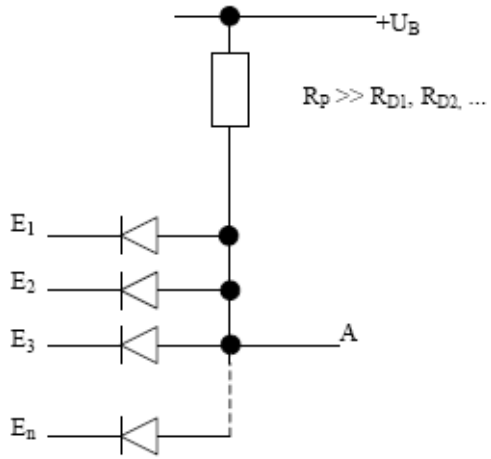


Kenndaten:

U_F : Durchlaßspannung	I_F : Durchlaßstrom
U_R : Sperrspannung	I_S : Sperrstrom
U_S : Schwellenspannung	
U_Z : Zenerspannung	

In einem Bereich U > Us ist die Diode durchlässig mit einem Widerstand RD, Durchlaßbereich.
In einem Bereich U<Us sperrt die Diode mit einem annähernd unendlichen Widerstand.

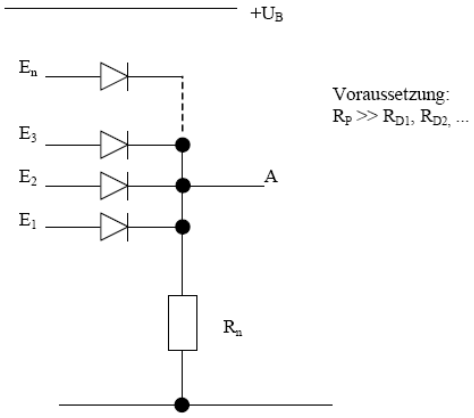
Unter Ausnutzung dieser Eigenschaften ist eine folgende UND-Schaltung realisierbar:



Nur wenn an allen Eingängen Ei die Betriebspannung anliegt oder die Eingänge offen sind, ist auch am Ausgang die Spannung hoch ,'High'=1 bei positiver Logik, es ergibt sich also eine logische UND-Schaltung.
Liegt auch nur an einem Eingang eine niedrige Spannung (~0V) an, so beträgt die unbelastete Ausgangsspannung Us=0,6V, also näherungsweise 'Low'=0 bei positiver Logik.
Bei negativer Logik 'H' = 0, ,'L' = 1 erhalten wir eine ODER-Schaltung.

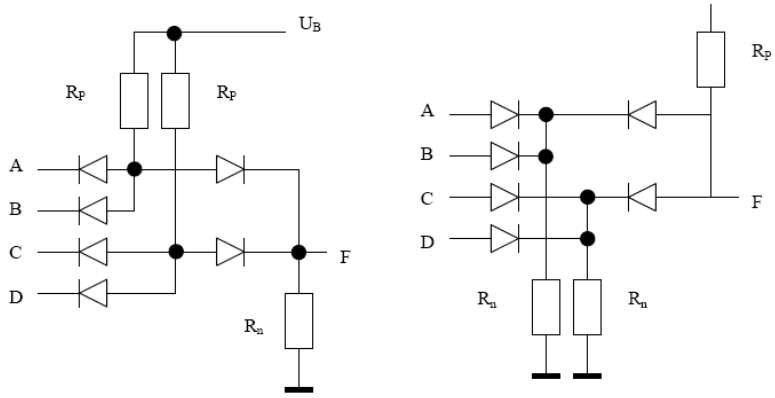
			pos. Logik "UND"					neg. Logik "ODER"			
E_1	E_2	E_3	A	E_1	E_2	E_3	A	E_1	E_2	E_3	A
L	L	L	L	0	0	0	0	1	1	1	1
L	L	H	L	0	0	1	0	1	1	0	1
L	H	L	L	0	1	0	0	1	0	1	1
L	H	H	L	0	1	1	0	1	0	0	1
H	L	L	L	1	0	0	0	0	1	1	1
H	L	H	L	1	0	1	0	0	1	0	1
H	H	L	L	1	1	0	0	0	0	1	1
H	H	H	H	1	1	1	1	0	0	0	0

Auch für positive Logik läßt sich mit Dioden eine ODER-Schaltung erzeugen:



Bereits ein Eingangssignal auf ‚H' reicht aus, um auch die Ausgangsspannung auf ‚H' zu bekommen.

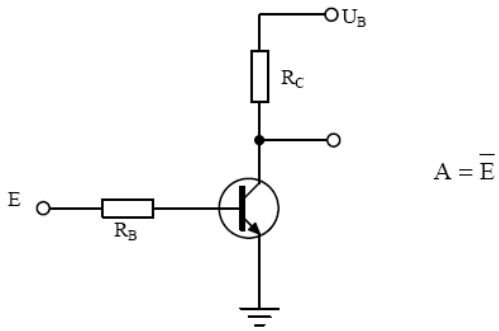
Durch Kaskadierung lassen sich Schaltnetze konstruieren:



F = (AB)+(CD)
F=(A+B)(C+D)

Negation

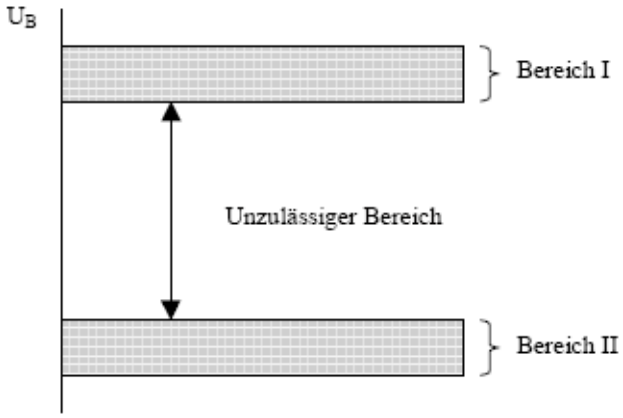
Für die Negation wird ein aktives Element benötigt, z.B. eine Inverterschaltung. Das nächste Bild zeigt einen npn-Transistor als Inverter.



Wird der Transistor am Eingang mit einer Spannung (High) größer als die Schwellspannung seiner BE-Diode angesteuert, so ‚'schaltet er durch' und wird niederohmig. Damit ergibt sich am Ausgang eine sehr kleine Spannung (Low).

7 Technologien integrierter Schaltkreise

Wie werden nun in praktischen Schaltungen die Logikpegel realisiert? Beschränken wir uns auf die binären Systeme, so müssen wir die logischen Zustände „"wahr" und „"falsch" darstellen. In der physikalischen Realisierung kann man dafür nicht feste elektrische Pegel verwenden, vielmehr muß man aus Toleranzgründen (Temperatur, Versorgungsspannung, Alterung, Last) Pegelbereiche angeben. Im allgemeinen lassen sich 3 Bereiche über der Betriebsspannung angeben.

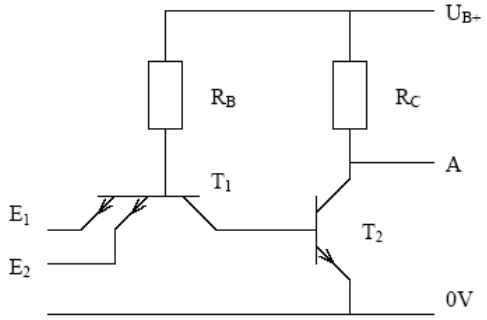


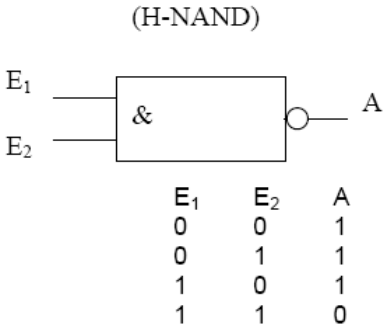
Es gibt die beiden Logikbereiche I und II und dazwischen einen undefinierten, verbotenen Bereich. Der absolute Wert der Betriebsspannung und der Bereichsgrenzen hängt dabei von den verwendeten Technologien und den gewünschten Eigenschaften ab. So erlauben hohe Betriebsspannungen auch hohe Störfestigkeit, erfordern umgekehrt aber höheren Energiebedarf und verringern die möglichen Schaltgeschwindigkeiten.
Übliche Spannungen im integrierten Bereich sind heute:

TTL, 5V
CMOS, 5-20V
BICMOS, 5V/3,3V
NMOS, 5V
ECL, -5,2V.

7.1.1 TTL-Technik

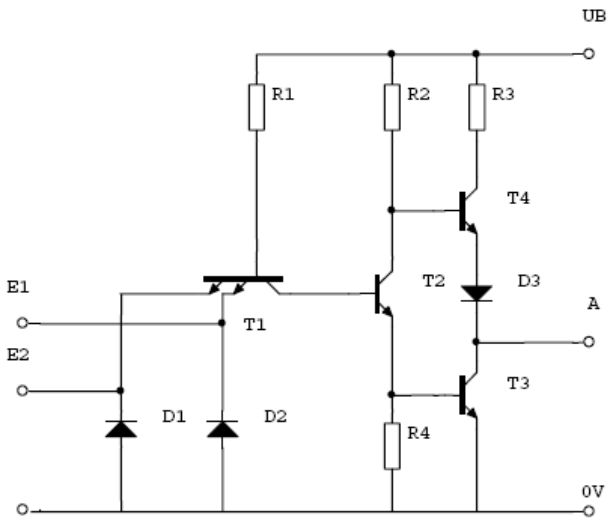
Der ursprünglich am meisten verbreitete Standard war (und ist noch) die TTL Technik. Ein Multi-Emitter-Transistor stellt das logische Element (Inversbetrieb, Basisgrundschaltung) dar. Nur wenn beide Eingänge auf 'H' liegen, öffnet die Basis-Kollektor –Diode von T1 und steuert T2 durch. Am Ausgang erscheint dann 'L'.





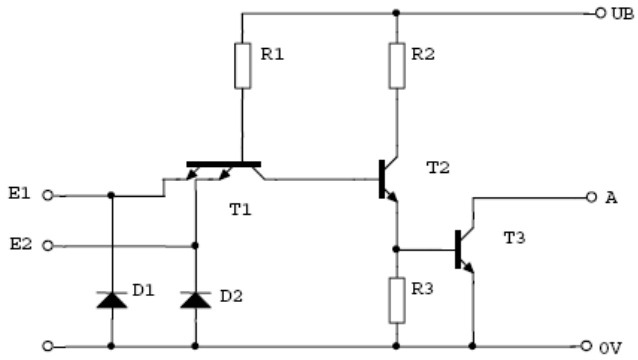
E_1	E_2	A
0	0	1
0	1	1
1	0	1
1	1	0

Der Eingangsstrom einer solchen Schaltung wird durch IeL=-UB/RB bzw. IeH=UB/2RB bestimmt und ist damit recht hoch.
Wird das statische FanOut der Ausgangsschaltung durch Anschließen zu vieler Eingangsstufen überschritten, so kann der logische Pegel nicht mehr garantiert werden.
Um das Ausgangsverhalten zu verbessern, hat man'Totem-Pole'-Ausgänge designed:



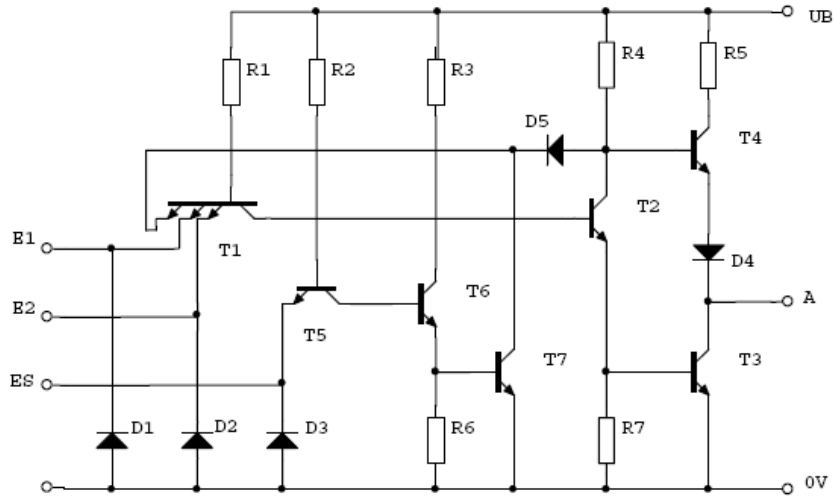
7.1.2 Open Collector Ausgangsschaltungen

Will man externe Laste, z.B. LEDs, Lampen oder Relais ansteuern, sind Open Collector Ausgangsschaltungen sinnvoll. Die Last kann hier gegen eine eigene positive Betriebsspannung in weiten Grenzen unabhängig von der Betriebsspannung des Logikbausteins betrieben werden.



7.1.3 Tri-State Ausgangsschaltung

Ein Tri-State-Ausgang kann neben den logischen Pegeln auch den Zustand X=’hochohmig’ annehmen.



Mit dem Steuereingang Es können beide Ausgangstransistoren in den Sperrbereich gesteuert werden, so daß der Ausgang schwimmt. Die ist nützlich, wenn sich mehrere Schaltungen abwechselnd eine Schaltleitung teilen müssen, z.B. bei gemultiplexten Bussystemen.

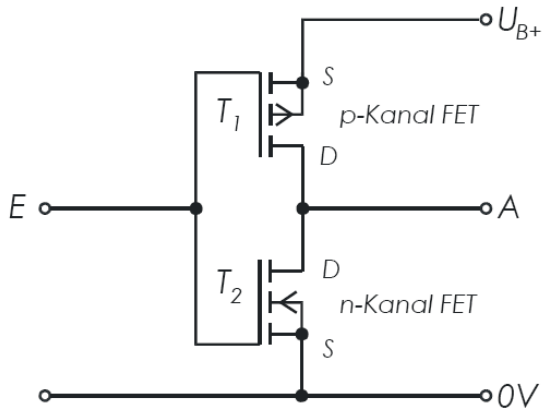
Es gibt diverse Ableitungen der ursprünglichen DTL- oder TTL-Technologie.
S (Schottky)-, F (Fast)-, und AS (Advanced Schottky)-Varianten sind besonders schnell durch Verwendung von schnellen Schottky-Dioden, haben aber auch erhöhte Verlustleistungen.

Weiterhin gibt es neuere Schaltungen mit Schwerpunkt auf niedriger Verlustleistung als LS- (Low Power Schottky) und ALS (Advanced Low Power Schottky)-Typen.

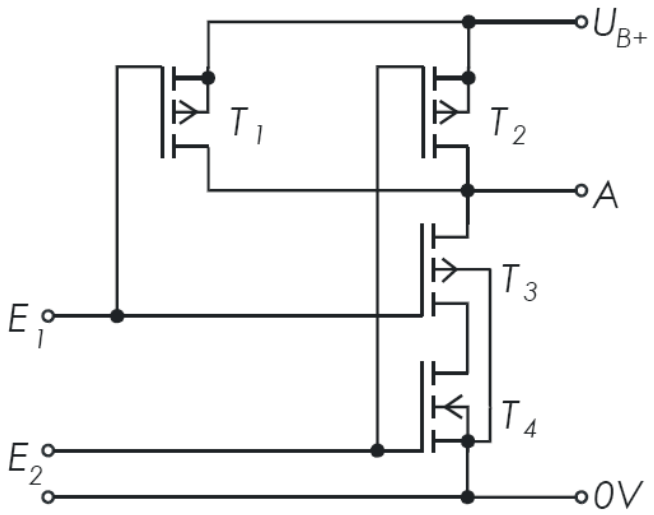
7.1.4 C-MOS-Technologie

Die CMOS-Technik (Complementary Metal Oxide Semiconductor) verwendet MOS Feldeffekt-Transistoren.
FET-Transistoren in n-Kanal CMOS-Technologie schalten näherungsweise leistungslos mit sehr geringem Eingangsstrom, können also als ideale Schalter betrachtet werden.
Durch die Verschaltung eines n-Kanal und eines p-Kanal FET-Schalters entsteht ein Inverter, bei dem eingangspegelabhängig immer ein Schalter durchgeschaltet, der andere gesperrt ist. Es fließt also nur im Moment des dynamischen Schaltens ein nennenswerter Strom. Auch die Ansteuerung über die verbundenen Gates geschieht wegen des hohen statischen Eingangswiderstandes nahezu verlustleistungslos. Die Betriebsspannung kann zwischen 5V und ca. 20 V gewählt werden, die logischen Schaltschwelle ist ungefähr bei der halben Betriebsspannung.

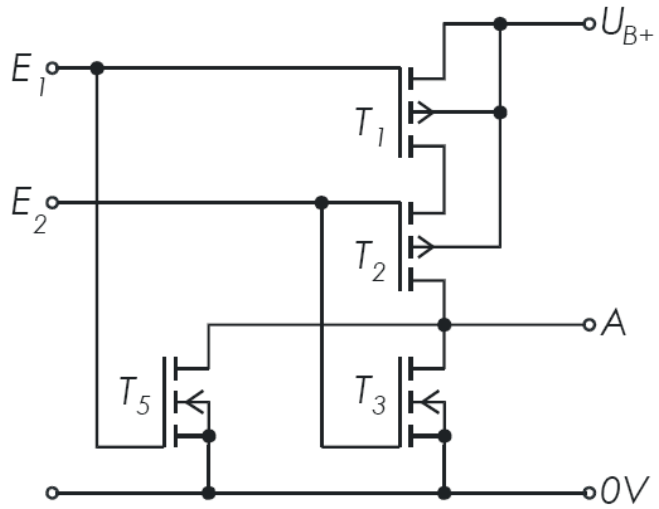
CMOS-Inverter:



Der Aufbau eines NAND-Gatters ist einfach:

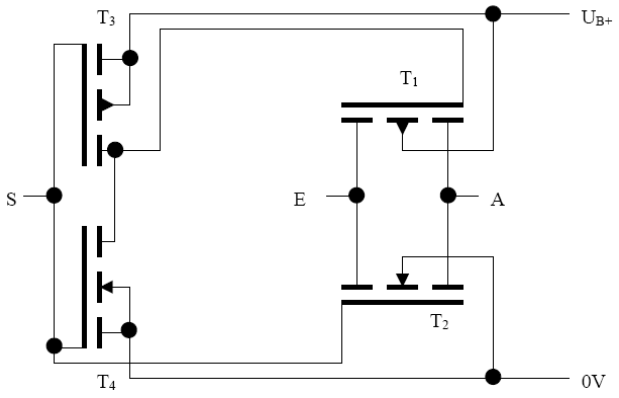


Ein entsprechendes NOR-Gatter:

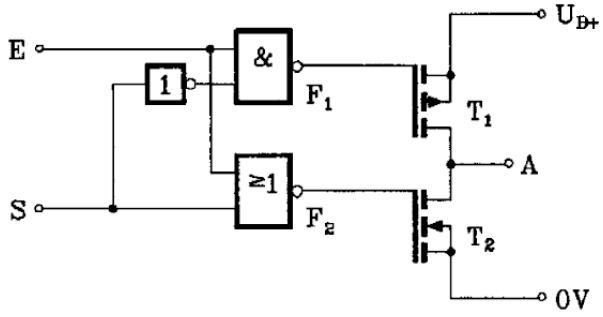


In dieser Technologie müssen insbesondere die dynamischen FanOut Bedingungen beachtet werden, da die ausgangsseitige Impedanz für H- und L-Pegel unterschiedlich und lastabhängig sind.

Analogschalter (transmission gates)
Mit nachfolgendem prinzipiell dargestellten Schalter kann ein Übergangswiderstand zwischen E und A sehr hochohmig oder sehr niederohmig gemacht werden. Die Schaltung ist gut geeignet für Schaltmatrizen und Multiplexer. Auch zur Realisierung von Tri-State-Ausgängen wird sie verwendet.



CMOS mit Tri-State-Ausgängen
Durch interne Kombinatorik können auch beide Ausgangs-FETs abgeschaltet werden:



Besonderheiten von CMOS-Schaltungen

7.1.4.1 Dynamisches Verhalten

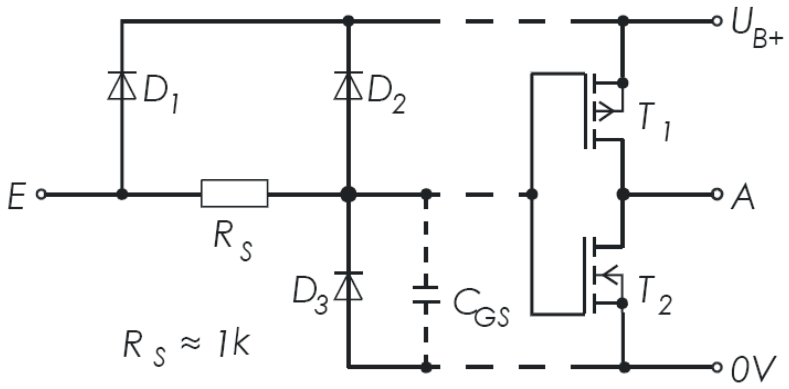
Im Moment des dynamischen Schaltens sind beide Ausgangstransistoren niederohmig, was einen hohen Querstrom verursacht. Dies erzeugt Lastspitzen auf der Stromversorgung, die sorgfältig dimensioniert werden muß. Der Versorgungsstrom bei CMOS-Schaltungen ist näherungsweise proportional zur Arbeitsfrequenz.

7.1.4.2 Latch-up

Es kann in den internen Schichten eines CMOS-Elements zur Durchschaltung eines parasitären Thyristors kommen, der dann Betriebsspannung gegen Masse kurzschließt. Dies führt zur internen Zerstörung des Elementes. Deshalb:
Unbenutzte Eingänge müssen auf feste Spannungen gelegt werden.
Eingangsspannungen dürfen nicht größer als UB werden, auch nicht beim Einschalten der Schaltung.

7.1.4.3 Empfindlichkeiten gegen statische Elektrizität

MOS-Schaltungen sind extrem hochohmig und entladen intern statische Ladungen nicht. Bereits bei statischer Aufladung >50V gibt es Zerstörungsgefahr. Besonders tückisch ist, daß unsachgemäße Handhabung zu Spätschäden führen kann, die erst in der fertig montierten und ausgelieferten Schaltung bemerkbar werden (Frühausfälle).
Zur Handhabung solcher Technologien sind MOS-Arbeitsplätze mit leitendem Bodenbelag, Kontrolle der Luftfeuchtigkeit, Anschluß der handhabenden Personen an Masse-Armbänder etc. erforderlich.
Schutzschaltungen von MOS-Schaltungen:

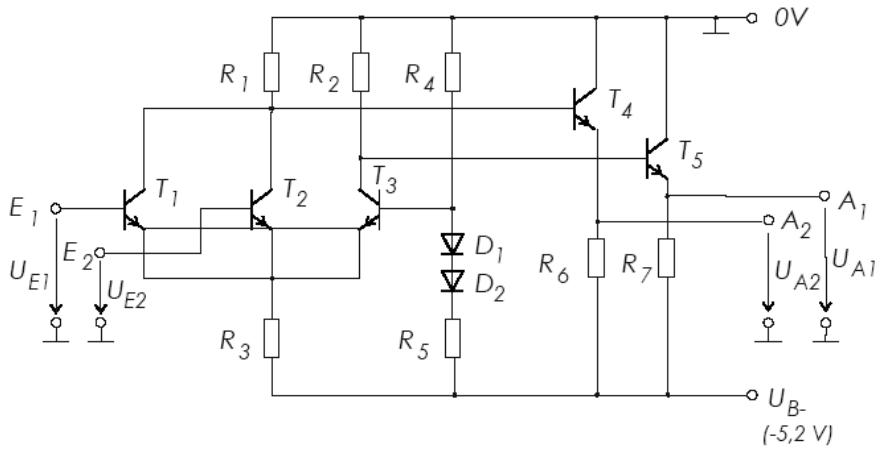


7.1.5 BICMOS-Technologie

BICMOS-Bausteine verwenden eine Kombination aus TTL-Ausgangsschaltung und CMOS-Eingangstechnik. Dies vereinigt hohe Arbeitsfrequenz mit robusten Ausgängen und ermöglicht weiterhin leichten Übergang zwischen Technologien und unterschiedlichen Betriebsspannungen. Die Schaltzeiten von BICMOS-Gattern liegen bei wenigen Nanosekunden.

7.1.6 ECL-Technik

Will man den Sub-Nanosekunden Bereich erreichen, wird man ECL-Technik verwenden (Emitter coupled Logic). Durch Vermeidung von Sättigungszuständen der Transistoren lassen sich Schaltzeiten von einigen 100 Picosekunden erreichen. Allerdings sind Integrationsdichte niedrig und Leistungsverbrauch hoch. Man wird daher diese Technologie nur im ,'Front-end' (z.B. A/D-Wandler) einsetzen und nach Reduzierung der Datenrate auf andere Technologien übergehen.



Eine Tabelle der verschiedenen Eigenschaften der Technologien schließt dieses Kapitel.

Schaltkreis-familie	$\frac{U_B}{V}$	$\frac{\overline{P_v}}{mW}$	$\frac{\overline{T_p}}{ns}$	$\frac{\overline{U_S}}{V}$	fan out	$\frac{\overline{f_{max}}}{MHz}$
DTL	5	15	25	0,7	8	10
DTLZ (LSL)	12 15	16 27	175 167	5 6,5	10 10	0,5-2 0,5-2
TTL-Standard	5	10	10	0,4	10	15
TTL-Low-Power	5	1	33	0,4	20	2,5
TTL-High-Power	5	22	6	0,4	10	35
TTL-Shottky	5	19	3	0,4	10	75
TTL-Low-Power-Shottky	5	2	9,5	0,4	20	25
TTL-Fast	5	5	2,25	0,4	20	100
ECL	-5,2	60	1	0,2	15	250
I^2L	1-15	0,01-30	10-1000	0,3	5	5
PMOS	-12	6	100	3	20	2
NMOS	10	2	15	2	20	15
CMOS	5 10 15	10^{-5}	100 50 40	1,5 3 4,5	50	2 5 7
LOCMOS	5 10 15	10^{-5}	60 30 20	1,5 3 4,5	50	8 16 24
HCMOS	5	10^{-5}	10	1,5	50	50

Folgende Tabelle gibt einen Auszug über bekannte Bausteinfamilien wieder, die darauf folgende Tabelle empfiehlt für bestimmte Technologien Ersatzfamilien

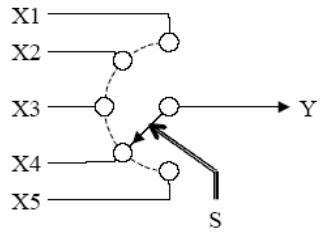
74F	Fast Logic
ABT	Advanced BiCMOS Technology
AC	Advanced CMOS Logic
ACT	Advanced CMOS Logic
AHC	Advanced High-Speed CMOS
AHCT	Advanced High-Speed CMOS
ALB	Advanced Low-Voltage BiCMOS
ALS	Advanced Low-Power Schottky Logic
ALVC	Advanced Low-Voltage CMOS Technology
ALVT	Advanced Low-Voltage BiCMOS Technology
AS	Advanced Schottky Logic
AVC	Advanced Very-Low-Voltage CMOS Logic
BCT	BiCMOS Technology
CD4000	CMOS Logic
ECL	Emitter Coupled Logic
FCT	Fast CMOS Technology
HC	High-Speed CMOS Logic
HCT	High-Speed CMOS Logic
LS	Low-Power Schottky Logic
LV-A	Low-Voltage CMOS Technology
LVC	Low Voltage CMOS Technology
LVT	Low-Voltage BiCMOS Technology
S	Schottky Logic
TTL	Transistor-Transistor Logic

Verwendete Familie	**Änderungsgrund**	**Neue Familie**
AC/ACT	Wechsel nach 3.3V	LVC
AC/ACT	Geringere Rauschleistung	AHC/AHCT
ABT	Wechsel nach 3.3V	LVT
AHC/AHCT	Wechsel nach 3.3V	LV
ALS	Wechsel von Bipolar nach CMOS; Schneller	AHC
ALS	Wechsel nach 3.3V	LV
ALS	Schneller; Wechsel nach 3.3V	LVC
ALVC	Schneller	AVC
AS	Wechsel von Bipolar nach BiCMOS; Schneller	ABT
AS	Wechsel nach 3.3V	LVT
BCT	Schneller; geringerer Stromverbrauch	ABT
74F	Geringeres Rauschen; geringerer Stromverbrauch	ABT
74F	Wechsel nach 3.3V	LVT
FCT	Wechsel nach 3.3V	LVT
HC/HCT	Wechsel nach 3.3V	LV
HC/HCT	Schneller	AHC/AHCT
HC/HCT	Viel schneller; Wechsel nach 3.3V	LVC
LS	Wechsel von Bipolar nach CMOS; Schneller	AHC
LS	Schneller; Wechsel nach 3.3V	LV
LV	Schneller	LVC
LV	Viel schneller	ALVC
LVC	Schneller	ALVC
LVQ	Schneller	LVC
LVQ	Viel schneller	ALVC
S	Wechsel von Bipolar nach CMOS; Schneller	AHC
S	Wechsel nach 3.3V	LVT
TTL	Wechsel von Bipolar nach CMOS; Schneller	AHC
TTL	Wechsel nach 3.3V	LV
TTL	Schneller; Wechsel nach 3.3V	LVC

8 Kombinatorische Grundschaltungen, Schaltnetze

8.1 Multiplexer und Demultiplexer

Funktionsweise eines Multiplexers
Multiplexer dienen dazu, aus vielen Signalen eines für die Weiterverarbeitung auszuwählen. Anschaulich kann man sich die Funktion eines Multiplexers an einem Drehschalter verdeutlichen, der genau eine Eingangsleitung auf einen Ausgang schaltet:



In der Digitaltechnik möchte man die Auswahl, welches Eingangssignal auf den Ausgang Y geschaltet wird, mit digitalen Signalen Si vornehmen. Somit erhält man eine boolesche Funktion, der als Parameter die Eingangsund die Auswahlsignale (auch als Selektionssignale bezeichnet) übergeben werden. Typischerweise wird das Auswahlsignal S als binärer Signalvektor realisiert und der selektierte Eingang binär kodiert.

8.1.1 Ein 2:1 Multiplexer

Für einen 2:1 Multiplexer ergibt sich die nachfolgende Funktionstabelle:

S	X2	X1	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Man erkennt, daß im Fall S=0 der Wert X1 und im Fall S=1 der Wert X2 auf den Ausgang gegeben wird. Damit kann die Funktionstabelle auch verkürzt geschrieben werden:

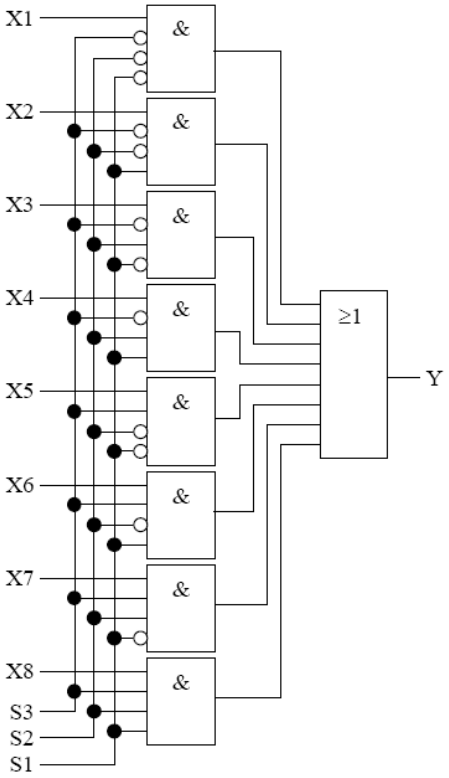
S	Y
0	X1
1	X2

Dies bedeutet, daß im Fall S=0 der Eingangswert von X1 auf dem Ausgang Y ausgegeben wird. Im Falle S=1 erfolgt die Ausgabe von X2 auf Y.

8.1.2 Ein 8:1 Multiplexer

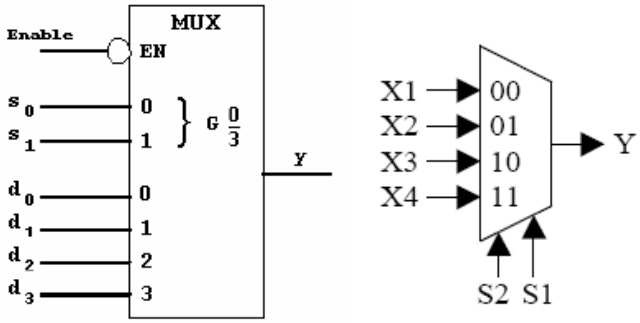
Mit der oben erläuterten verkürzten Schreibweise der Funktionstabelle kann beispielsweise auch ein 8:1- Multiplexer übersichtlich dargestellt werden:

S3	S2	S1	Y
0	0	0	X1
0	0	1	X2
0	1	0	X3
0	1	1	X4
1	0	0	X5
1	0	1	X6
1	1	0	X7
1	1	1	X8



8.1.3 DIN-Symbol

Ein Multiplexer kann nach der DIN 40900 Norm mit einem eigenen Symbol beschrieben werden. Nachfolgend ist ein Symbol für einen 4:1 Multiplexer gezeigt:



Die Steuereingänge S1 und S2 wählen aus, welcher der Eingänge X1, X2, X3 oder X4 auf den Ausgang Y geschaltet wird. Dazu muß aus den Steuereingängen ein Index berechnet werden. Die Bezeichnungen 0 und 1 der Steuereingänge geben eine Wertigkeit bei ihrer

Zusammenfassung zu einem Index an. In obigem Beispiel wird ein Indexbereich von 0 bis 3 spezifiziert, Steuereingang S1 hat die Wertigkeit 0 und S2 die Wertigkeit 1. Somit ergibt eine Ansteuerung von S2=1 und S1=0 den Index 2 und schaltet den mit 2 bezeichneten Eingang X3 auf den Ausgang Y.

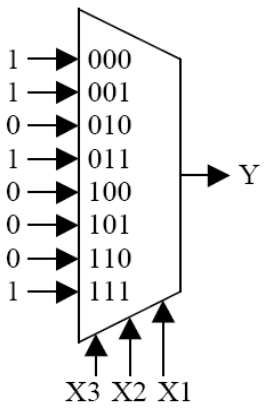
Multiplexer sind geeignet, um boolesche Funktionen zu realisieren.

8.2 Realisierung boolescher Funktionen mit Multiplexern.

8.2.1 Realisierung von Funktionen mit N Eingängen durch 2^N:1 Muxer

Am einfachsten ist die Realisierung einer Funktion mit N Eingangsvariablen durch einen 2^N:1 Multiplexer. Die Eingänge schaltet man an die Selektionseingänge, an die Dateneingänge schaltet man, je nach Vorgabe der Funktion, eine 0 oder 1.
Beispiel: Realisierung einer Funktion mit N=3 Eingängen mittels 8:1 Multiplexer. Gegeben ist die folgende boolesche Funktion:

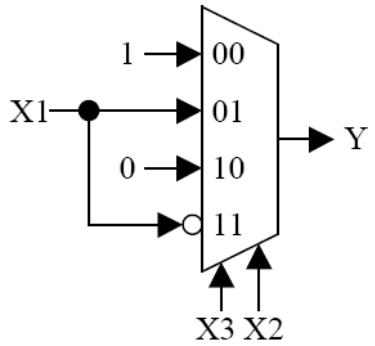
X3	X2	X1	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Beispiel: Realisierung einer Funktion mit N=3 Eingängen mittels 4:1 Multiplexer.
Gegeben ist die folgende boolesche Funktion:

X3	X2	X1	Y	
0	0	0	1	Y=1
0	0	1	1	
0	1	0	0	Y=X1
0	1	1	1	
1	0	0	0	Y=0
1	0	1	0	
1	1	0	1	Y=¬X1
1	1	1	0	

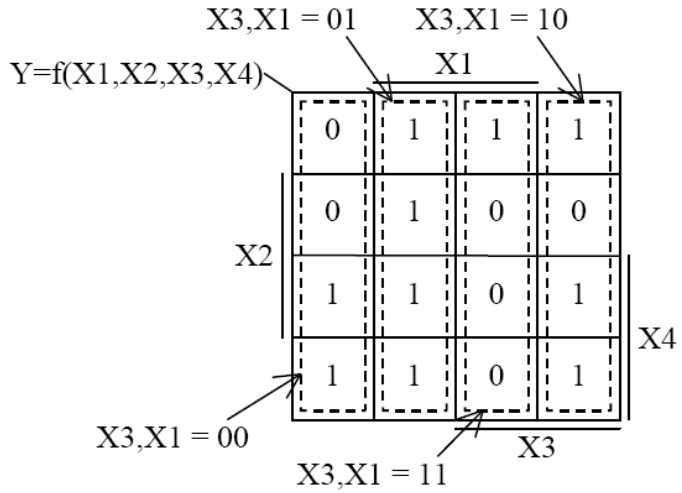
Wählt man X2 und X3 als Steuereingänge, so kann man die Funktionstabelle direkt durch die folgende Multiplexerschaltung realisieren:



Beispiel: Realisierung einer Funktion aus 4 Eingangsvariablen mit einem 4:1 Multiplexer
Gegeben sei die folgende Funktion aus 4 Eingangsvariablen:

X4	X3	X2	X1	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

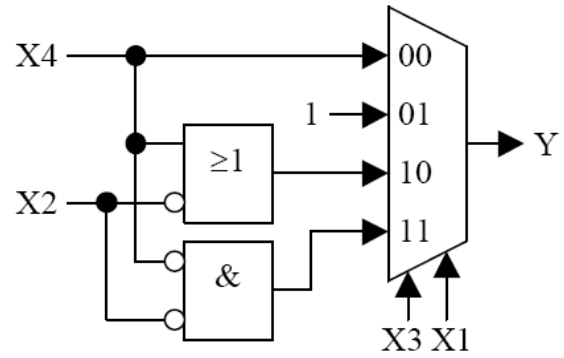
Will man diese Funktion direkt und vollständig mit einem Multiplexer realisieren, so benötigt man, wie im vorhergehenden Abschnitt gezeigt, einen 16:1 oder (wenn negierte Signale zur Verfügung stehen) einen 8:1 Multiplexer.
Im Beispiel soll die Funktion jedoch mit einem 4:1 Multiplexer realisiert werden.
Wählt man (willkürlich) die Eingangsvariablen X3 und X1 als Steuervariable des Multiplexers, so kann man im KV-Diagramm für jede Kombination der Steuereingänge ein Unterdiagramm lokalisieren und als boolesche Funktion darstellen. In nachfolgendem KV-Diagramm sind die vier Unterdiagramme markiert und mit der zugehörigen Steuerkombination des Multiplexers gekennzeichnet.



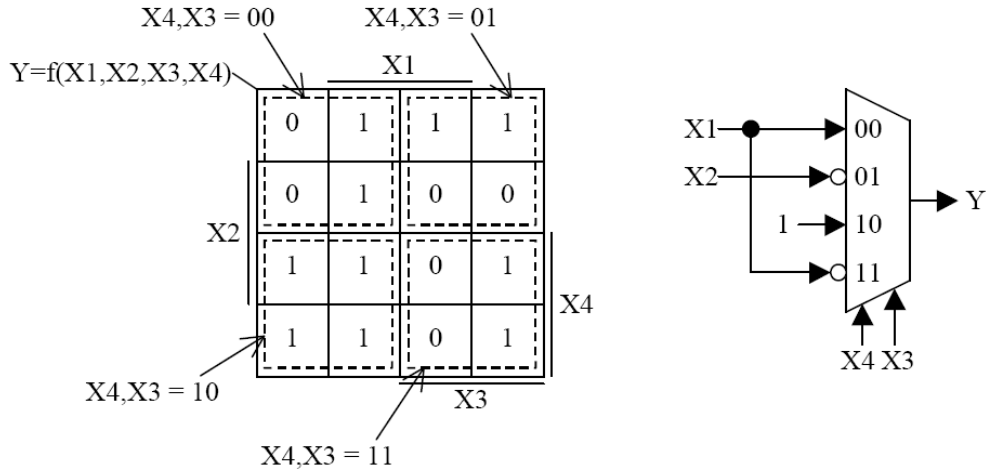
Man erhält somit für jede Steuerkombination eine boolesche Funktion über den verbleibenden Variablen X2 und X4:

Steuerkombination X3,X1	Ausgangsfunktion
00	$Y = X4$
01	$Y = 1$
10	$Y = X4 \vee (\neg X2)$
11	$Y = (\neg X4) \wedge (\neg X2)$

Damit ergibt sich die Schaltung der booleschen Funktion aus einer Kombination einfacher Minimalformen und einem Multiplexer.

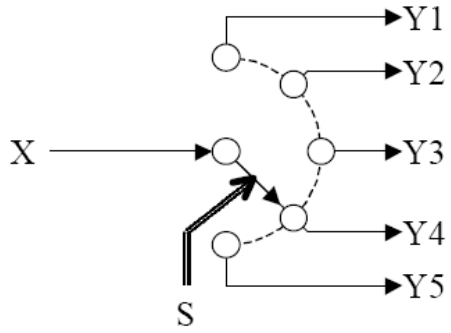


Die gefundene Lösung ist nicht optimal. Wählt man beispielsweise X4 und X3 als Ansteuersignale des Multiplexers, so erhält man vier triviale boolesche Funktionen über den verbleibenden Variablen X2 und X1 und eine Schaltung mit geringerem Aufwand.



8.2.2 Funktionsweise eines Demultiplexers

Ein Demultiplexer realisiert die umgekehrte Funktion des Multiplexers. Ein einzelner Eingang X wird auf einen von vielen Ausgängen geschaltet. Der in nachfolgender Abbildung dargestellte Drehschalter realisiert einen Demultiplexer.



Bei der digitalen Realisierung eines Demultiplexers wählt man den Ausgang Yi, mit dem der Eingang X verbunden wird, durch Steuereingänge Si aus. Mit einem Steuervektor S aus N Bits kann man einen Eingang mit einem von 2^N Ausgängen verbinden.

8.2.3 Ein 1:4 Demultiplexer

Ein 1:4 Demultiplexer besitzt einen Eingang X, zwei Steuereingänge S1, S2 und vier Ausgänge Y1, Y2, ..., Y4. Das bisher beschriebene Verhalten des Demultiplexers stellt sich in folgender Funktionstabelle dar:

S2	S1	X	Y1	Y2	Y3	Y4
0	0	0	0	*	*	*
0	0	1	1	*	*	*
0	1	0	*	0	*	*
0	1	1	*	1	*	*
1	0	0	*	*	0	*
1	0	1	*	*	1	*
1	1	0	*	*	*	0
1	1	1	*	*	*	1

Diese Tabelle lässt sich verkürzt schreiben, wenn jeder Ausgang Yi als Funktion von X dargestellt wird. Weiterhin möchte man üblicherweise, daß ein Ausgang einen konstanten Wert (0 oder auch 1) ausgibt, wenn der Ausgang nicht selektiert ist. In nachfolgender Tabelle

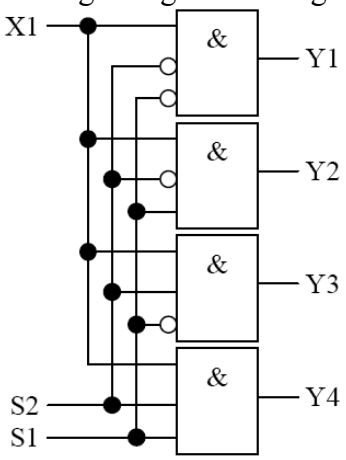
wird für einen nicht selektierten Eingang der Wert 0 angenommen. Dann erhält die Tabelle die folgende Form:

S2	S1	Y1	Y2	Y3	Y4
0	0	X	0	0	0
0	1	0	X	0	0
1	0	0	0	X	0
1	1	0	0	0	X

Damit erhält man die folgenden Gleichungen zur Beschreibung eines 1:4 Demultiplexers:

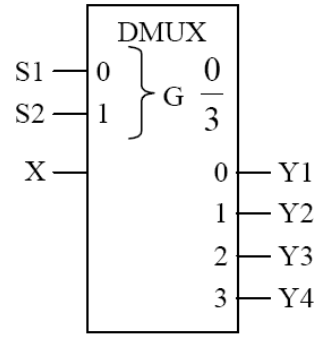
$$Y1 = X \wedge (\neg S2) \wedge (\neg S1)$$
$$Y2 = X \wedge (\neg S2) \wedge S1$$
$$Y3 = X \wedge S2 \wedge (\neg S1)$$
$$Y4 = X \wedge S2 \wedge S1$$

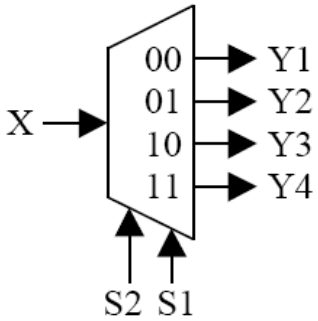
Die zugehörige Schaltung ergibt sich zu:



8.2.4 DIN-Symbol

Ein Demultiplexer wird nach der DIN 40900 Norm mit einem eigenen Symbol beschrieben. Nachfolgend ist ein Symbol für einen 1:4 Demultiplexer gezeigt:





Blockschaltbild

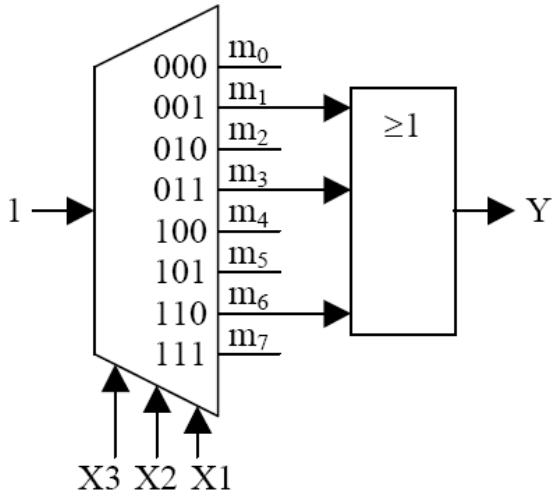
8.2.5 Realisierung boolescher Funktionen mit Demultiplexern

Legt man bei einem Demultiplexer den Eingang X auf den Wert 1, liefert jeder Ausgang eine Minterm-Funktion m_i der Steuereingänge. Disjunktive Verknüpfung der gewünschten Minterme liefert die vorgegebene Funktion. Mit einem $1:2^N$ Demultiplexer und einem maximal N Bit breiten ODER-Gatter kann man somit alle gewünschten Funktionen aus N Eingangsvariablen realisieren.

Beispiel: Realisierung einer vorgegebenen Funktion aus 3 Eingängen mit einem 1:2^3Demultiplexer

X3	X2	X1	Y	
0	0	0	0	
0	0	1	1	m_1
0	1	0	0	
0	1	1	1	m_3
1	0	0	0	
1	0	1	0	
1	1	0	1	m_6
1	1	1	0	

Die Demultiplexer-Schaltung liefert direkt alle Minterme mi aus drei Eingangsvariablen. Verbindet man die für die Funktion benötigten Minterme m1, m3 und m6 mit dem nachfolgenden ODER-Gatter, erhält man eine Schaltungsrealisierung der vorgegebenen Funktion:



8.3 Code-Umsetzer

8.3.1 Prioritäts-Encoder

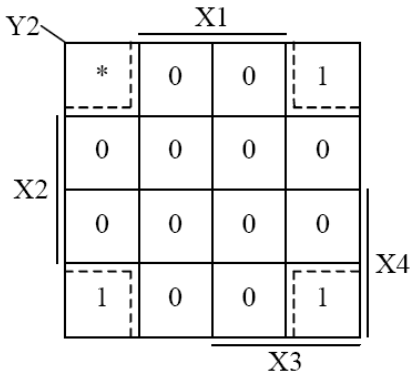
In Rechnerschaltungen werden häufig Ereignisse auf einzelnen Leitungen signalisiert, somit können Ereignisse parallel auftreten. Ein Rechner kann jedoch Ereignisse nur sequentiell nacheinander bearbeiten. Somit ist es notwendig, aus mehreren aktiven Signalen das Ereignis höchster Priorität zu detektieren und dem Rechner mitzuteilen, so daß dieser darauf reagieren kann. Prioritäts-Encoder dienen diesem Zweck. Ein Prioritäts- Encoder detektiert, ob mindestens eine seiner Eingangsleitungen aktiviert ist, und ermittelt den Index der am höchsten prioritisierten, aktivierten Leitung.
Beispiel: Ein Prioritäts-Encoder für 4 Eingangsleitungen, bei denen Eingang X1 die höchste Priorität besitzt, läßt sich durch folgende Wahrheitstabelle spezifizieren:

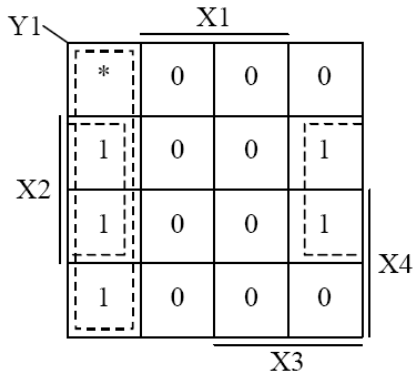
X4	X3	X2	X1	Y3	Y2	Y1
0	0	0	0	0	*	*
0	0	0	1	1	0	0
0	0	1	0	1	0	1
0	0	1	1	1	0	0
0	1	0	0	1	1	0
0	1	0	1	1	0	0
0	1	1	0	1	0	1
0	1	1	1	1	0	0
1	0	0	0	1	1	1
1	0	0	1	1	0	0
1	0	1	0	1	0	1
1	0	1	1	1	0	0
1	1	0	0	1	1	0
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	0	0

Der Ausgang Y3 nimmt dann den Wert 1 an, wenn zumindest einer der Eingänge den Wert 1 besitzt. Die Ausgänge Y2 und Y1 liefern den binärkodierten Index des niedrigsten aktiven Eingangs. Wenn Y3=0 gilt, sind die Werte von Y2 und Y1 beliebig.
Die Ermittlung der Gleichung von Y3 ist einfach:

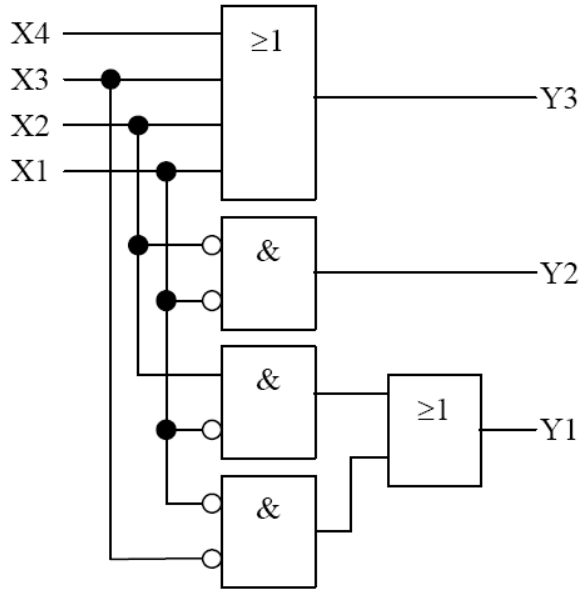
$Y3 = X1 \veebar X2 \veebar X3 \veebar X4$

Zur Ermittlung von Y1 und Y2 werden die zugehörigen KV-Diagramme aufgestellt:





Die detektierten Primimplikanten führen zu den folgenden Minimalformen für Y1 und Y2:
$Y1 = ((\neg X1) \wedge X2) \veebar ((\neg X1) \veebar (\neg X3))$
$Y2 = ((\neg X1) \wedge (\neg X2))$
Somit erhält man die folgende Schaltung für den beschriebenen Prioritäts-Encoder:



8.3.2 Beispiel: 1-aus-10-Dekoder

Aufgabe: Dekodierung und Anzeige von BCD-Zahlen
Eine von 10 Lampen, die mit den Ziffern 0 – 9 beschriftet sind, soll mit einer BCD-Zahl angesprochen werden, d.h. die entsprechende Lampe soll eingeschaltet werden.

0	1	2	3	4	5	6	7	8	9

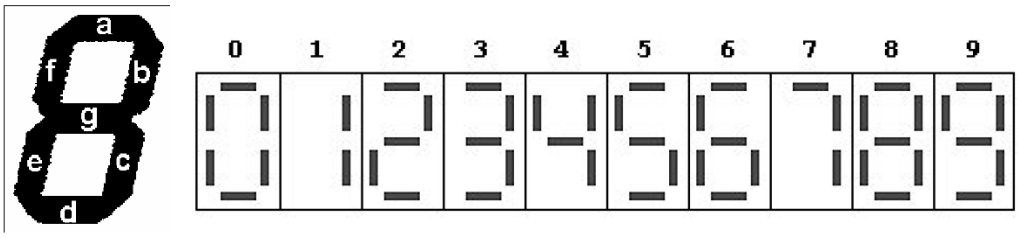
Vorgehen:
1. Funktionstabelle aufstellen
2. Disjunktive Normalform für die Ausgänge aufstellen (auch für Don‘t Care)
3. Minimierung der Funktion (z.B. KV-Diagramm)
4. Lösung aufschreiben
5. Schaltung erstellen (und Bauteilauswahl)

	Eingänge				Ausgänge									
BCD	D	C	B	A	a	b	c	d	e	f	g	h	i	j
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	1	0	0	0	0	0	0	0
3	0	0	1	1	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	1	0	0	0	0	0
5	0	1	0	1	0	0	0	0	0	1	0	0	0	0
6	0	1	1	0	0	0	0	0	0	0	1	0	0	0
7	0	1	1	1	0	0	0	0	0	0	0	1	0	0
8	1	0	0	0	0	0	0	0	0	0	0	0	1	0
9	1	0	0	1	0	0	0	0	0	0	0	0	0	1
10	1	0	1	0	×	×	×	×	×	×	×	×	×	×
11	1	0	1	1	×	×	×	×	×	×	×	×	×	×
12	1	1	0	0	×	×	×	×	×	×	×	×	×	×
13	1	1	0	1	×	×	×	×	×	×	×	×	×	×
14	1	1	1	0	×	×	×	×	×	×	×	×	×	×
15	1	1	1	1	×	×	×	×	×	×	×	×	×	×

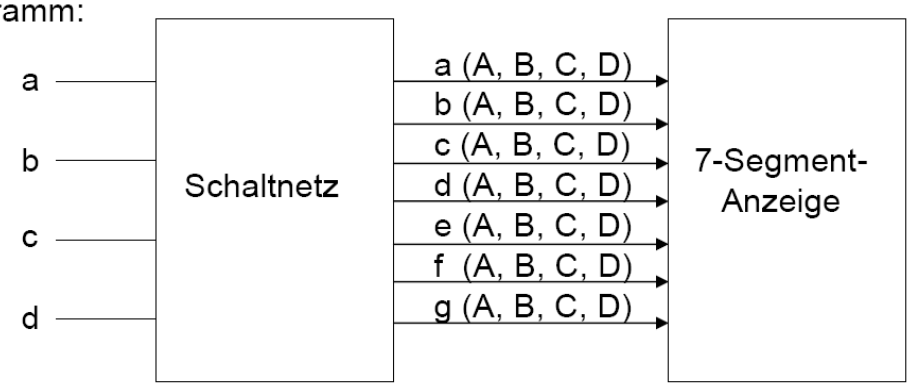
a = A‘ B‘ C‘ D‘
b = A B‘ C‘ D‘
d = A B C‘
e = A‘ B‘ C
f = A B‘ C
g = A‘ B C
h = A B C
i = A‘ D
j = A D

8.3.3 Beispiel: BCD zu 7-Segment-Dekoder

7-Segment-Anzeige:



Blockdiagramm:

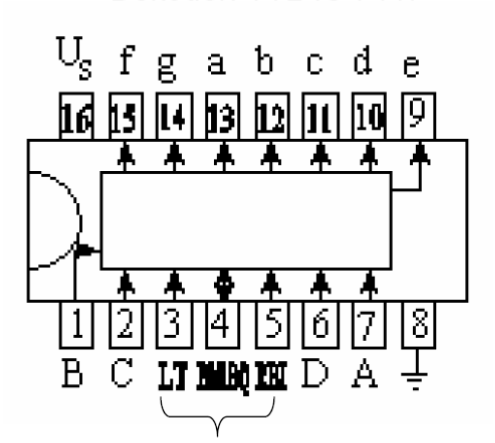


Funktionstabelle:

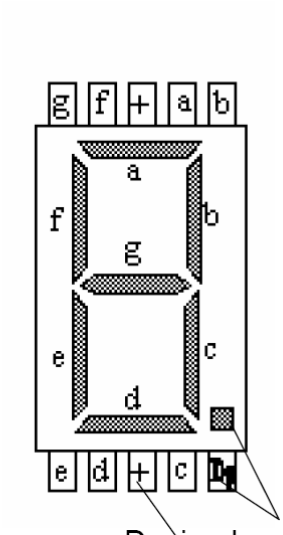
BCD-Code (Eingangsvariable)				7-Segment-Code (Ausgangsvariable)							
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	
0	0	0	1	0	1	1	0	0	0	0	
0	0	1	0	1	1	0	1	1	0	1	
0	0	1	1	1	1	1	1	0	0	1	
0	1	0	0	0	1	1	0	0	1	1	
0	1	0	1	1	0	1	1	0	1	1	
0	1	1	0	1	0	1	1	1	1	1	
0	1	1	1	1	1	1	0	0	0	0	
1	0	0	0	1	1	1	1	1	1	1	
1	0	0	1	1	1	1	1	0	1	1	
1	0	1	0	X	X	X	X	X	X	X	Pseudo-tetraden (Don‘t care)
1	0	1	1	X	X	X	X	X	X	X	
1	1	0	0	X	X	X	X	X	X	X	
1	1	0	1	X	X	X	X	X	X	X	
1	1	1	0	X	X	X	X	X	X	X	
1	1	1	1	X	X	X	X	X	X	X	

Bauelemente:

Dekoder: TTL-IC 7447



Z.B. „Lampentest“



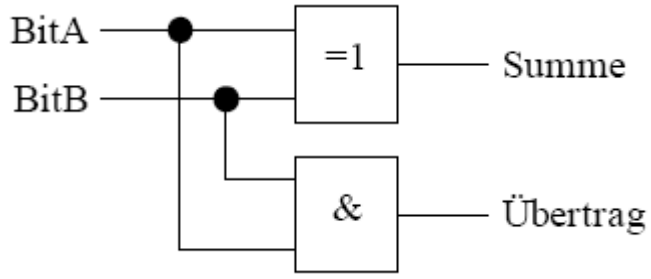
Anschlusspunkt(e); nur einmal zu beschalten

8.4 Rechenwerke

8.4.1 Beispiel: Addition

Aufgabe: Zwei Dualzahlen addieren
Beispiel: 1011
+1111
111 (Übertrag)
11010
Lösungsschritte:
Halbaddierer: Zwei einstellige Dualzahlen addieren; Ergebnis ist SummenBit und ÜbertragsBit (Carry-Bit)

Summe = BitA **xor** BitB
Übertrag = BitA **and** BitB



Blockschaltbild HA

Volladdierer: Zwei einstellige Dualzahlen und eingehendes ÜbertragsBit addieren; Ergebnis ist SummenBit und (ausgehendes) ÜbertragsBit

Addierwerk: Zwei mehrstellige Dualzahlen addieren

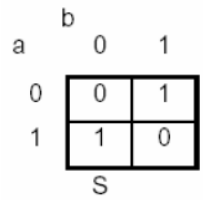
(Kombination von Volladdierern)

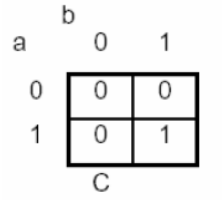
8.4.2 Beispiel: Halbaddierer

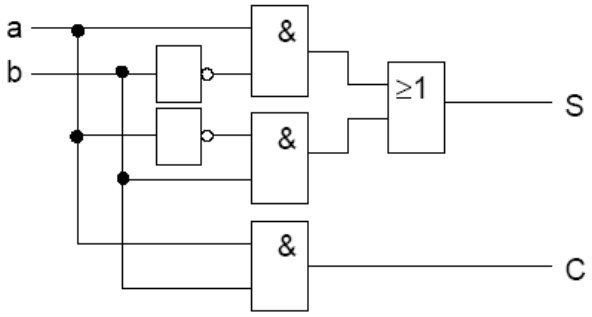
Aufgabe:
Addiert zwei einstellige Dualzahlen
Erzeugt als Ergebnis ein
SummenBit (S)
ÜbertragsBit bzw. Carry-Bit (C)

a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

DNF: S = ab‘ + a‘b
C = ab
-> Keine Vereinfachung mit KV-Diagramm möglich

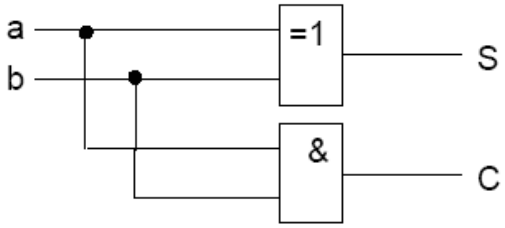






Vereinfachung mit XOR (Antivalenz):

S = ab‘ + a‘b = a ⊕ b



8.4.3 Beispiel: Volladdierer

Aufgabe:

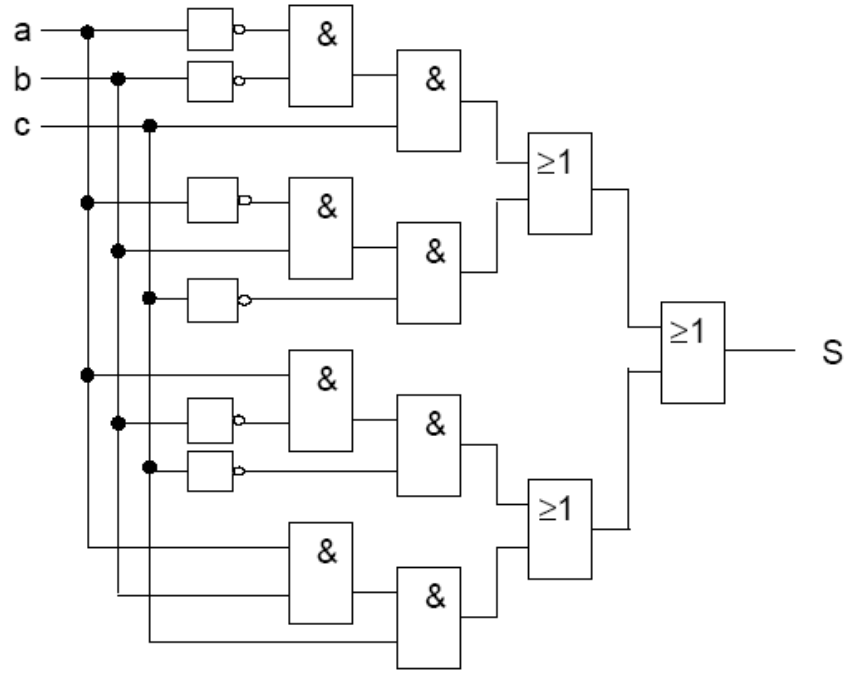
• Addiert zwei einstellige Dualzahlen mit eingehendem ÜbertragsBit (c = Carry in), d.h. es sind drei Bit zu addieren
• Erzeugt als Ergebnis ein
- SummenBit (S)
- ÜbertragsBit bzw. Carry-Bit (C)

Funktionstabelle:

a	b	c	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Betrachtungen für das SummenBit S:
DNF: S = a'b'c + a'bc' + ab'c' + abc
KV-Diagramm:
->keine Vereinfachung möglich (ebenso mit dem Verfahren nach Quine McCluskey

c \ ab	00	01	11	10
0	0	1	0	1
1	1	0	1	0



Algebraische Vereinfachung:
Ziel: Nur XOR – Gatter verwenden
S = a'b'c + a'bc' + ab'c' + abc
= a' (b'c + bc') + a (b'c' + bc)

mit x‘y + xy‘ = x ⊕ y (XOR, Antivalenz)

und x‘y‘ + xy = (x ⊕ y)‘ = x ⊙ y (XNOR, Äquivalenz)
wird

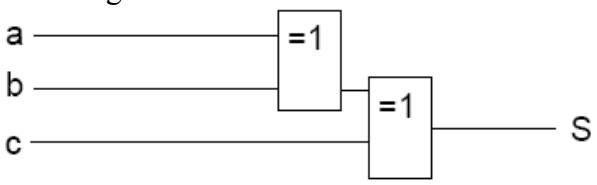
S = a‘ (b ⊕c) + a (b ⊕c)‘

= a ⊕(b ⊕c)

S = a ⊕b ⊕c

x	y	XOR	XNOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

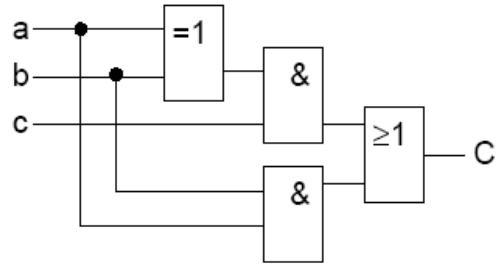
Schaltung für das SummenBit:



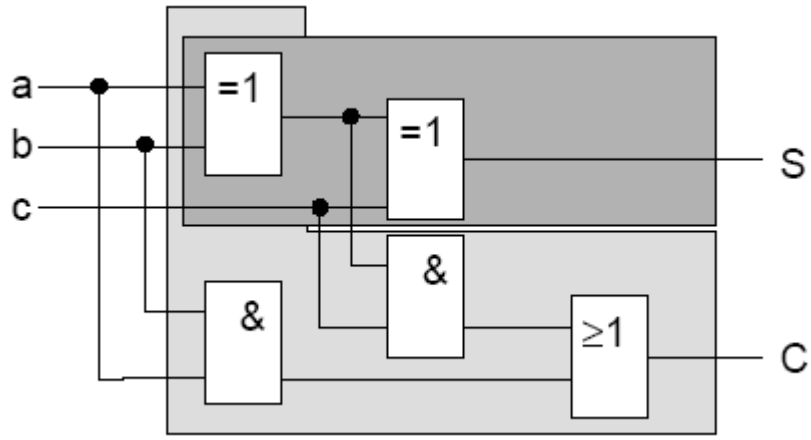
Betrachtungen für das Carry-Bit C:

C = ab‘c + ab + a‘bc
= c (ab‘ + a‘b) + ab

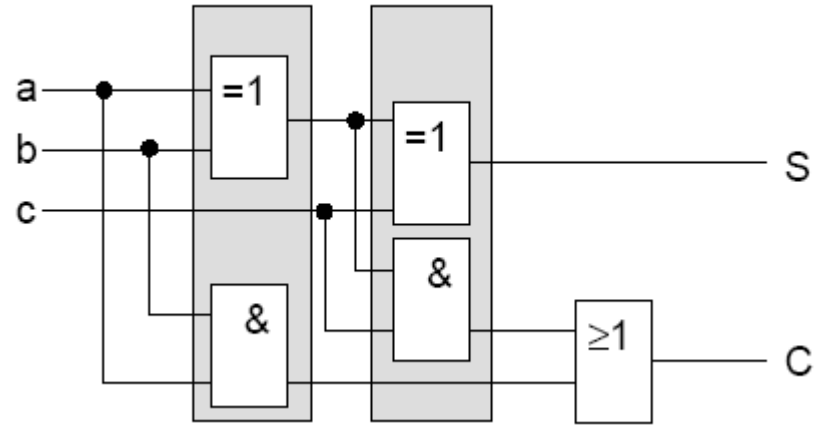
C = c (a ⊕b) + ab



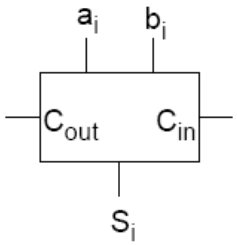
Zusammenbau der Schaltungen für SummenBit und Carry-Bit:



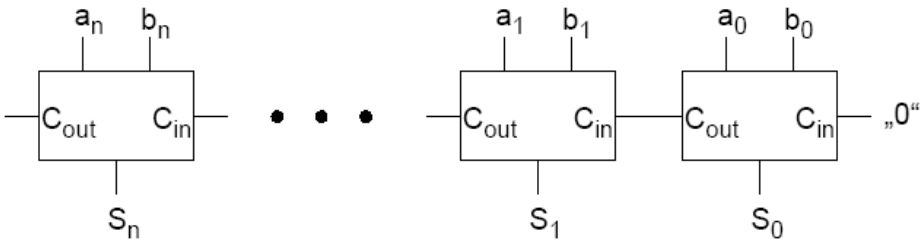
8.4.4 Vergleich mit Halbaddierer:



Der Volladdierer ist aus zwei Halbaddierern und einem ODER-Gatter aufgebaut. Andere Darstellung:



Paralleladdierwerk (Ripple Adder):



Vorteil:

- Einfache Schaltung
- Modularer Aufbau (Volladdierer an Stelle 0 erlaubt Kaskadierung – das Carry-in kann dann aber 0 oder 1 sein)
- Theoretisch beliebig ausbaubar

Nachteil: Langsam, da sich die Ausführungszeiten der einzelnen Volladdierer summieren
Lösung: Carry-Look-Ahead-Addierwerk (schneller, aber aufwendiger)
Prinzip:
• Zusammenfassung von z.B. acht Volladdierern zu einem Addierwerk
• Carry-Bit wird dem darüber liegenden Addierwerk zur Verfügung gestellt, bevor die Addition vollständig ausgeführt ist

Für das Carry-Bit des Volladdierers war:

$C = ab + c\,(a \oplus b)$

Mit $ab = G$ = „carry generate"-Signal
(wenn a,b beide 1 sind, wird ungeachtet des Carry-in ein Carry-out erzeugt)

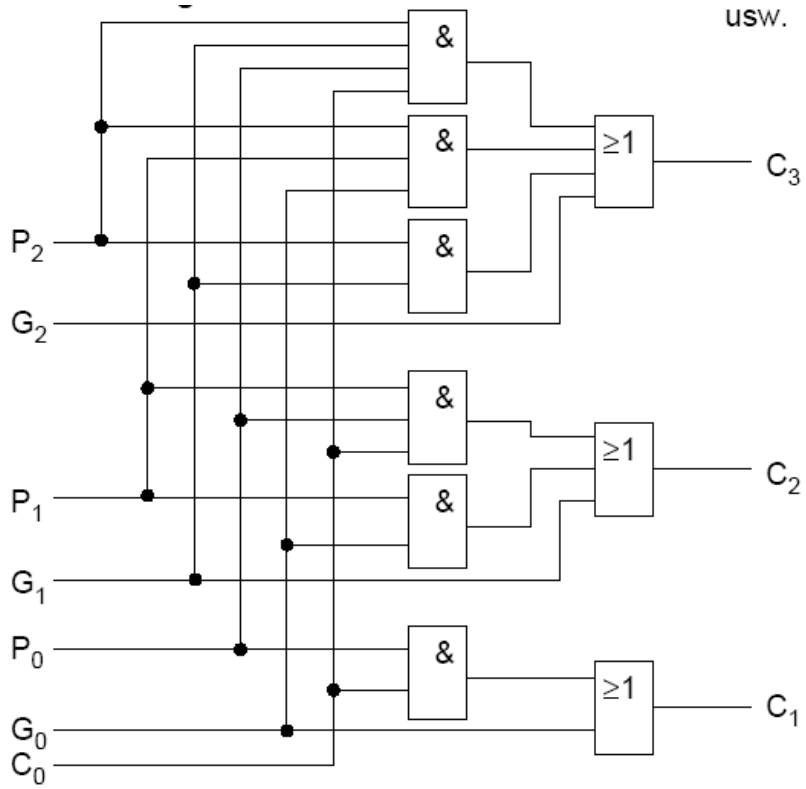
und $a \oplus b = P$ = „carry propagate"-Signal

kann man verallgemeinert schreiben

$C_{i+1} = G_i + P_i C_i$

und sukzessive entwickeln

$C1 = G0 + P0C0$
$C2 = G1 + P1C1 = G1 + P1\,(G0 + P0C0) = G1 + P1G0 + P1P0C0$
$C3 = G2 + P2C2 = G2 + P2\,(G1 + P1G0 + P1P0C0)$
$= G2 + P2G1 + P2P1G0 + P2P1P0C0$
usw.



Vorteil:
• Hohe Beschleunigung des Addierers

Nachteile:
• Aufwand nimmt mit zunehmenden Addierstufen sehr stark zu (vergl. Sukzessive Entw.)

• Teuer, z.B. für 32 oder 64 Bit Addierer

8.4.5 Beispiel: Quadrierer für 2-stellige Dualzahl

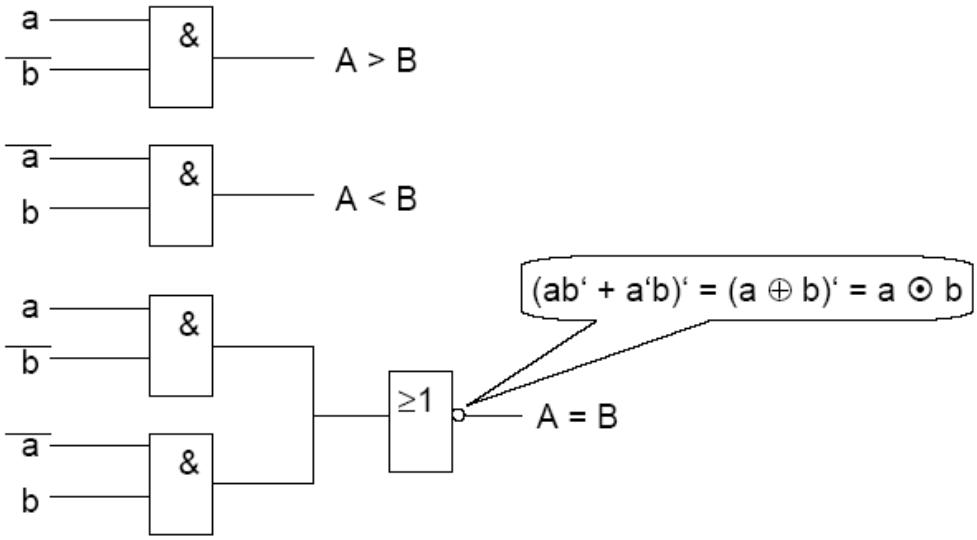
Eingänge: a0, a1
Ausgänge: b0, b1, b2, b3
Funktionstabelle:

a_1	a_0	b_3	b_2	b_1	b_0
0	0	0	0	0	0
0	1	0	0	0	1
1	0	0	1	0	0
1	1	1	0	0	1

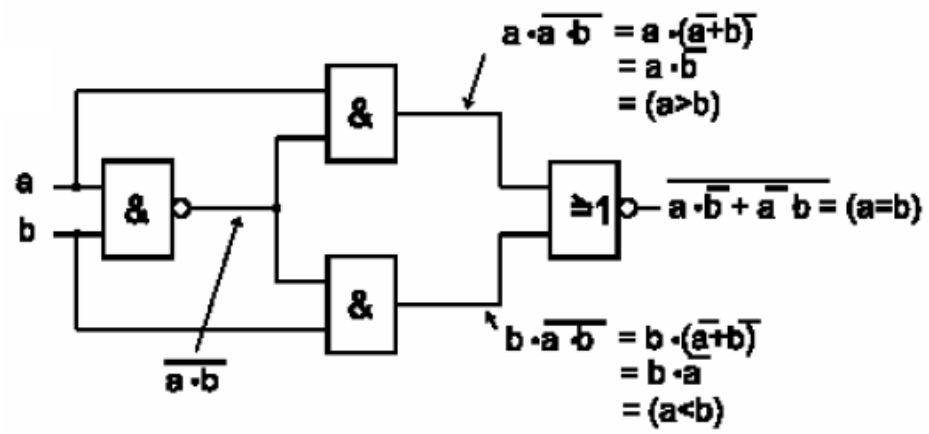
Gleichungen für die Ausgänge:
b0 = a1‘a0 + a1a0 = a0
b1 = 0
b2 = a1a0‘
b3 = a1a0

8.4.6 Beispiel: Komparator

Vergleicher: Vergleicht zwei Eingangswerte
Ergebnis:
a) = („gleich“)
b) > („größer“)
c) > („kleiner“)
Einfache binäre Vergleichsschaltungen:
a = b : y = a‘b‘ + ab = a ⊙ b = (a ⊕ b)‘
a > b : y = ab‘
a < b : y = a‘b



Schaltungsrealisierung:

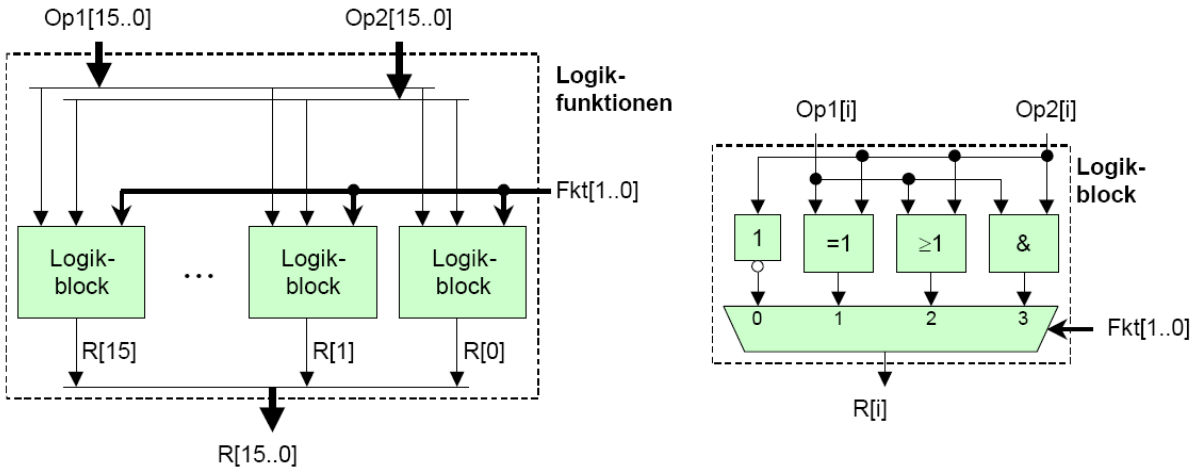


8.5 Logische Operationswerke

Neben der Arithmetik werden in Hochsprachen logische Operationen und Schiebeoperationen als Operatoren spezifiziert. Somit müssen diese Operationen in einem Rechner durch ein Rechenwerk direkt oder indirekt zur Verfügung gestellt werden. Der Abschnitt zeigt Rechenwerke für bitweise logische Operationen und für Schiebeoperationen, wie sie in Rechnern benötigt werden, um beispielsweise die entsprechenden Operatoren von C oder C++ zu realisieren.

8.5.1 Bitweise logische Verknüpfungen

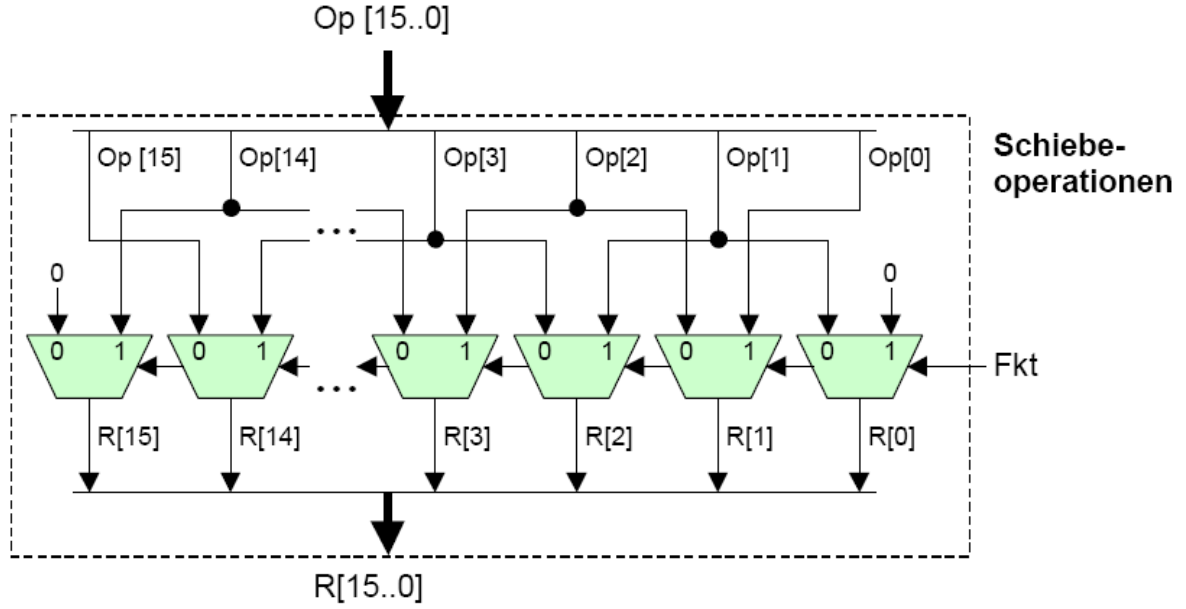
Die wichtigsten bitweisen logischen Operationen sind die Negation, UND-Verknüpfung, ODER-Verknüpfung und die EXKLUSIV-ODER-Verknüpfung. Die Funktionen lassen sich in einer Logikeinheit zusammenfassen, wie dies in folgender Abbildung dargestellt ist.



In der gezeigten Realisierung werden die zwei Operanden *OP1* und *OP2* bitweise miteinander verknüpft. Die Verknüpfung zweier Einzelbits ist in der Abbildung in einem Logikblock zusammengefasst, dessen Innenschaltung auf der rechten Seite der Abbildung im Detail gezeigt ist. Die Verknüpfung erfolgt parallel mit allen vier Logikfunktionen. Das gewünschte Ergebnis der Funktion wird mit einem nachgeschalteten Multiplexer ausgewählt.

8.5.2 Schiebeoperationen

Bei der Realisierung von Schiebeoperationen in einem Rechner muss entschieden werden, ob per Hardware beliebige Schiebeoperationen in einem Schritt, oder nur die Verschiebung um ein Bit pro Schritt durchführt werden kann.
Die erste Lösung führt zu einem „Barrel-Shifter“. Der Entwurf einer zugehörigen Architektur wird dem Leser als Übung überlassen. Im zweiten Fall müssen mehrfache Schiebeoperationen durch mehrfache Anwendung einzelner Schiebeschritte per Software realisiert werden.



Rechenwerk zum Rechts- und Linksschieben eines Operanden