Vorlesung

Technische Informatik II

Für Bachelor

- Grundzüge der Datentechnik –

Kapitel 2

2. Entwurf und Analyse kombinatorischer Schaltungen

Ein digitales System realisiert in definierter Weise eine Zuordnung von binären Eingangswerten (oder digitalen Signalen) in binäre Ausgangswerte. Digitale Systeme werden üblicherweise mit drei Gruppen von Schaltungen und deren Kombinationen realisiert:

- mit Schaltnetzen, d.h. kombinatorischen Schaltungen, die rein kombinatorisch zusammengesetzt sind und keinerlei Speicher enthalten.
- mit Schaltwerken, d.h. sequentiellen Schaltungen, die neben der Speicherung und Zeitverzögerung digitaler Signale auch die Impulserzeugung ermöglichen.
- mit Sonderschaltungen z.B. zur Anpassung von Eingangs- und Ausgangssignalen, Pegelumsetzung und Leistungsverstärkung (Pegelwandler, Treiber, Sender-/ Empfängerbausteine)

Dem eigentlichen Ziel digitaler Schaltungen und Strukturen, nämlich der Ausführung logischer Funktionen, dienen nur die Schaltnetze und Schaltwerke.

2.1 Definition von Schaltnetzen und Modulverkettung

Ein Schaltnetz ordnet jeder Kombination von binären Eingangsvariablen (x_0, ... , x_ν, ... ,x_{n-1}) eine Kombination von binären Ausgangsvariablen (y_0, ... ,y_μ, ... ,y_{m-1}) zu. Die Eingangs- und Ausgangsvariablen (x_ν,y_μ) können zwei Werte annehmen (1 oder 0).

Die Eingangsvariablen x sind als gegebene Signale, z.B. elektrische Spannungen von binären Speichern, Registern, Sensoren, Tasten oder von Ausgängen anderer Schaltnetze, anzusehen. Die Ausgangsvariablen y führen wiederum zu binären Speichern, zu anderen Schaltnetzen, zu Anzeigeelementen oder bewirken andere äußere Aktionen. In Bild 2-1 ist das Blockschaltbild eines Schaltnetzes angegeben, wobei die Ausgangsgrößen Y über die logische Funktion λ mit den Eingangsgrößen X verknüpft sind.

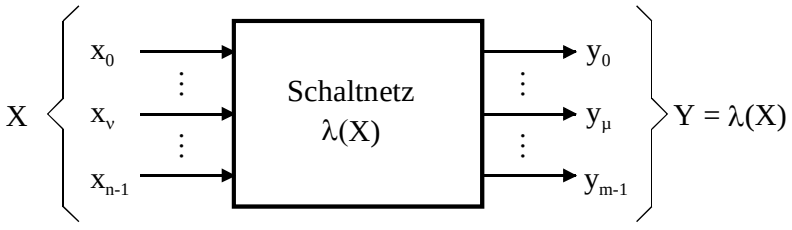


Bild 2-1: Blockschaltbild eines Schaltnetzes

Ein Eingangswort X wird gebildet aus einer Binärkombination der Komponenten $x_0 \ldots x_{n-1}$:

$$X^i = (x_{n-l}, x_{n-2, \ldots,} x_\nu, \ldots, x_0);$$

X^i ist ein bestimmtes Element der Menge X^k, wobei der Exponent k die Mächtigkeit der Menge kennzeichnet. Ein Ausgangswort Y wird in gleicher Weise gebildet wie aus einer Binärkombination der Komponenten $y_0 \ldots y_{m-1}$:

$Y^j = (y_{m-l}, y_{m-2}., \ldots, y_\mu, \ldots, y_0)$

Im Folgenden wird der Entwurf von Schaltungen bei gegebener und zu realisierender logischer Funktion λ betrachtet. Bei komplexen logischen Funktionen kann man auch den Entwurf strukturieren, in dem man das gesamte Schaltnetz der Funktion λ in eine Kombination von Teilnetzen (Modulen) λ_p zerlegt.

Bild 2-2 zeigt eine Verkettung von Modulen kombinatorischer Logik.

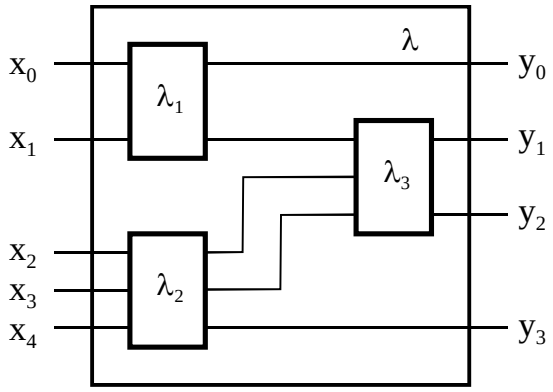


Bild 2-2: Beispiel für zulässige Modulverkettung

Zu beachten ist bei der Modulverkettung, dass keine Ausgänge von Teilnetzen direkt untereinander oder direkt mit Eingängen zusammen geschaltet werden (s. Bild 2-3). Zulässig ist aber, ein Signal auf mehrere Eingänge im Netz zu verteilen.

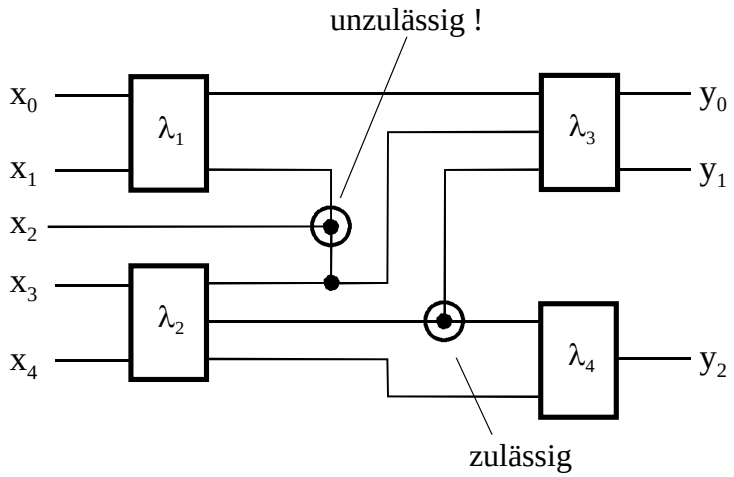


Bild 2-3: Beispiel für unzulässige Modulverkettung

Für die Schaltnetze gilt weiter, dass kein Ausgangssignal der Teilmodule auf in der Verkettung davor liegende Module gelegt werden darf (keine Rückkopplungszweige). Durch eine Rückkopplung wird aus einer kombinatorischen Logikfunktion eine sequentielle Schaltung (Bild 2-4), die im folgenden Kapitel behandelt werden.

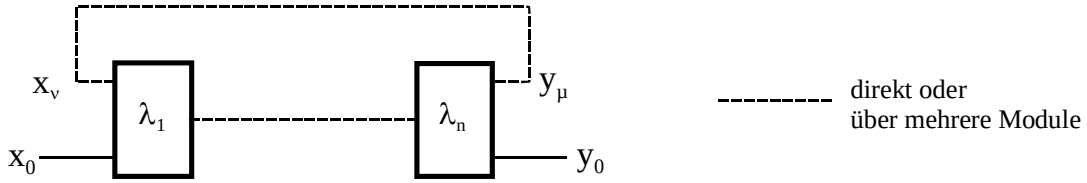


Bild 2-4: Beispiel für eine sequentielle Schaltung durch rückgekoppelte Verkettung von kombinatorischen Modulen

Für die Realisierung von Schaltnetzen kann man die logische Funktion auf Verkettung von sogenannten Elementarfunktionen zurückführen. Das hat den Vorteil für eine schaltungstechnische Realisierung, dass nur eine begrenzte Zahl von Standardmodulvarianten verfügbar sein muss, um komplexe Logikfunktionen implementieren zu können.

Bild 2-5 zeigt eine solche Implementierung als Beispiel. Die logische Funktion des Schaltnetzes ist die eines sog. Volladdierers, der zwei Binärvariablen sowie einen Übertrag (zur Verkettung mehrerer Binärstellen) addiert und daraus die Summe sowie einen Übertrag (in die nächste Binärstelle) generiert. Die Realisierung kommt mit drei Elementarfunktionen aus.

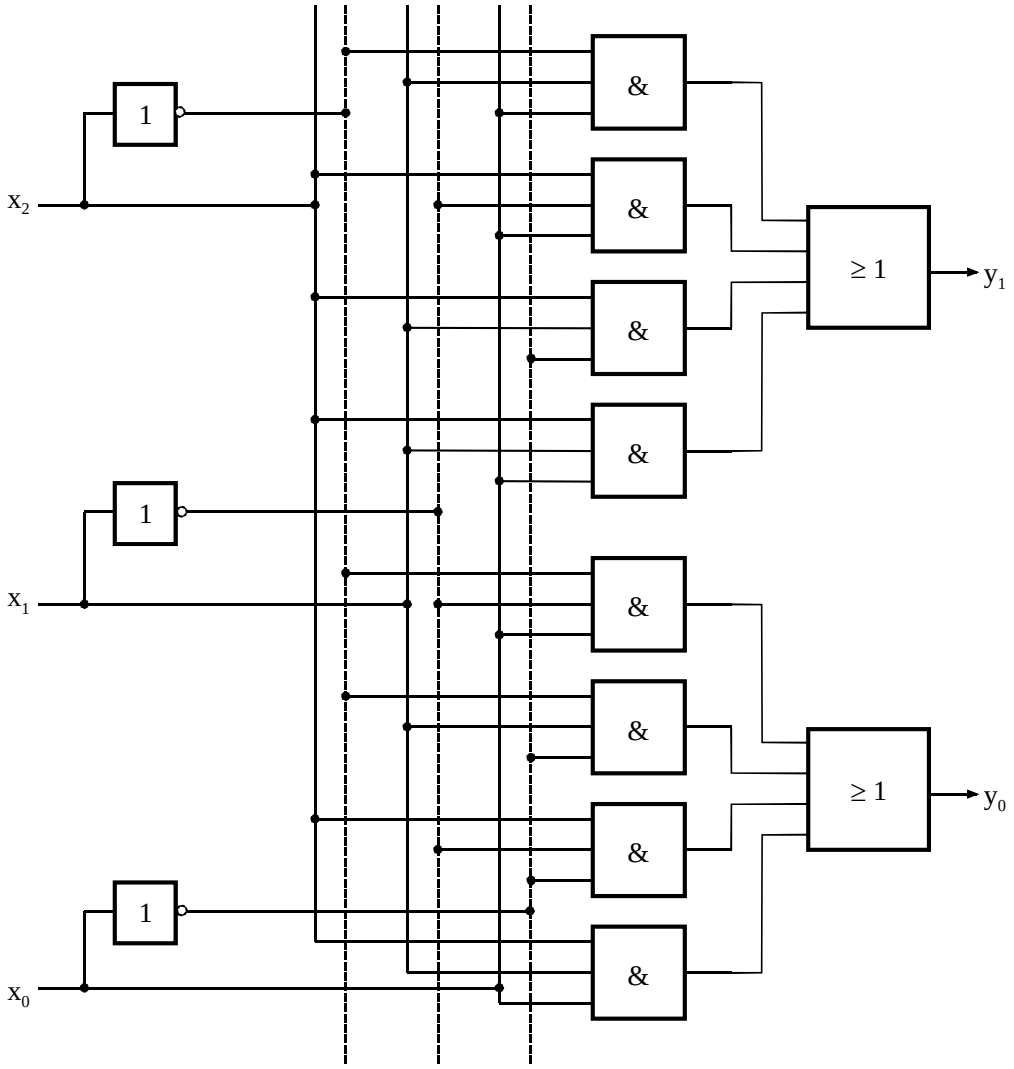


Bild 2-5: Netz von Elementarschaltungen

2.2 Beschreibung kombinatorischer Logik

2.2.1 Darstellungsformen

Die individuelle Abhängigkeit der Ausgangsgrößen von den Eingangsvariablen eines Schaltnetzes können auf verschiedene Weise dargestellt werden, z. B. als Wahrheitstabelle, als (Boolesche) Gleichung, mit einem vereinbarten Kurzsymbol, als Baumstruktur oder als ein Netz von Elementarschaltungen (Bild 2-5). Diese Darstellungsmöglichkeiten werden im Folgenden am Beispiel des Volladdierers angegeben.

1. Wahrheitstabelle

Mit Hilfe einer Wahrheitstabelle oft auch Werte- oder Zustandstabelle genannt, können die Zusammenhänge zwischen den Eingangsvariablen und den Ausgangsgrößen beschrieben werden. Wie in Bild 2-6 angegeben, sind auf der linken Seite der Tabelle alle $2^3 = 8$ möglichen Eingangskombinationen (x_2, x_1, x_0) dargestellt. Auf der rechten Seite sind die den Eingangskombinationen zugeordneten Ausgangsgrößen y_1 und y_0 aufgeführt.

Eingangsvariable x			Ausgangs-variable y	
x_2	x_1	x_0	y_1	y_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Bild 2-6: Wahrheitstabelle eines Schaltnetzes

2. Gleichung

Der Zusammenhang zwischen Eingangs- und Ausgangsvariablen lässt sich im angegebenen Beispiel auch schaltalgebraisch darstellen durch:

$$y_0 = \overline{x}_2\,\overline{x}_1\,x_0 \quad v \quad \overline{x}_2\,x_1\overline{x}_0 \quad v \quad x_2\,\overline{x}_1\overline{x}_0 \quad v \quad x_2\,x_1x_0$$

$$y_1 = \overline{x}_2\,x_1\,x_0 \quad v \quad x_2\,\overline{x}_1x_0 \quad v \quad x_2\,x_1\overline{x}_0 \quad v \quad x_2\,x_1x_0$$

(Die Funktionen, Schreibweisen und Gesetze der Schaltalgebra werden in Abschnitt 2.2.2 eingeführt.)

3. Kurzsymbol

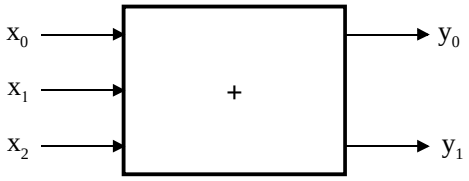


Bild 2-7: Kurzsymbol eines Schaltnetzes

Für den im Beispiel verwendeten Volladdierer kann das in Bild 2-7 gezeichnete Kurzsymbol angegeben werden. Neben den Eingangs- und Ausgangsvariablen wird die Funktion des Schaltnetzes durch das Zeichen "+" beschrieben.

4. Netz von Elementarschaltungen

Die Auflösung in Elementarschaltungen, die technischen Schaltungen entsprechen, nennt man "Realisierung". In Bild 2-5 ist der Volladdierer mit Hilfe der in Abschnitt 3.3.2 beschriebenen Schaltfunktionen dargestellt.

5. Baumstruktur

Die Wahrheitstabelle nach Bild 2-6 lässt sich auch in Form eines "Entscheidungsbaumes" beschreiben. Bild 2-8 zeigt den sog. Shannon-Baum für das Beispiel aus Bild 2-6.

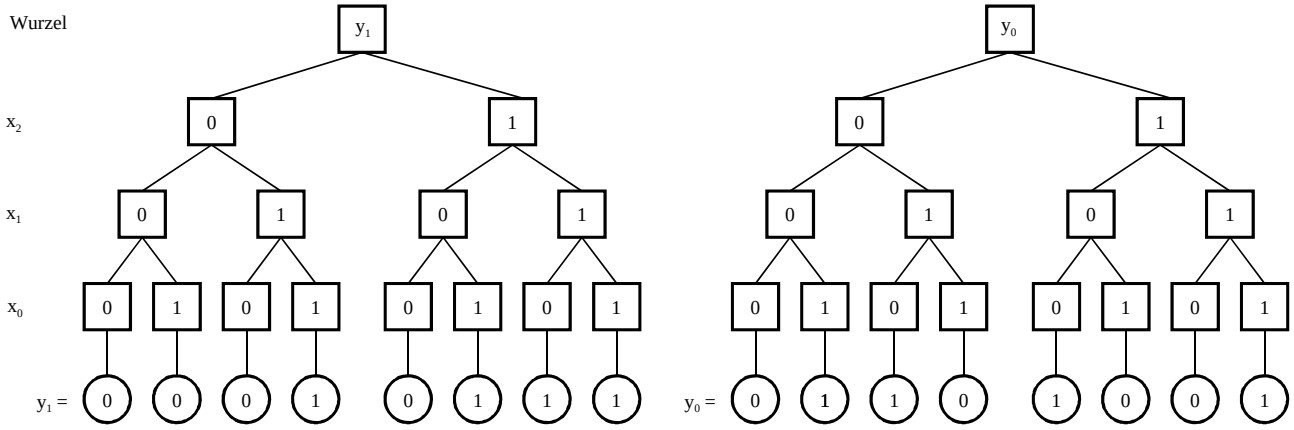


Bild 2-8: Shannon Baum

Die Darstellungsformen1., 2., und 5. sind sog. kanonische Darstellungen, da sie die vollständigen Abhängigkeiten der Ausgangsvariablen von allen Eingangskombinationen darstellen.

2.2.2 Schaltfunktion und Schaltalgebra

Eine Schaltfunktion ("logische Funktion") ist die Abhängigkeit einer Variablen y von binären Eingangsvariablen $x_0 \ldots x_{n-1}$. Sie entspricht einem Schaltnetz mit einer Ausgangskomponente.

2.2.2.1 Logische Funktionen von einer Variablen

Für eine Variable (x_0) gibt es $2^1 = 2$ Eingangskombinationen und $2^2 = 4$ mögliche Funktionen (y_0, y_1, y_2, y_3), die Null- und Einsfunktion, die Identität und die Negation (Bild 2-9). Mit logischen Funktionen von einer Variablen lassen sich noch keine verketteten Module zu Realisierung komplexer Schaltnetze aufbauen.

Eingangskombination	Eingangsvariable	Ausgangsfunktion				Bezeichnung
	x_0	y_0^1	y_1^1	y_2^1	y_3^1	Algebra. Darstellung
		0	x_0	$\overline{x}_0$	1	
x^0	0	0	0	1	1	Funktionstabelle
x^1	1	0	1	0	1	
		Nullfunktion	Identität	Negation	Einsfunktion	Benennung

Bild 2-9: Logische Funktion von einer Variablen

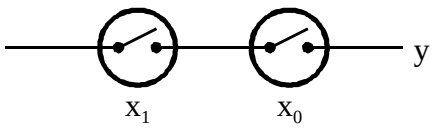
2.2.2.2 Logische Funktionen von mehreren Variablen

Da sich mit logischen Funktionen von zwei Eingangsvariablen bereits Modulverkettungen aufbauen lassen, enthalten diese bereits auch unseren benötigten Satz von Elementarfunktionen zur Realisierung komplexer Schaltnetze. Im Prinzip kommt man dabei mit zwei Grundverknüpfungen und der Negation aus. Alle weiteren logischen Funktionen lassen sich durch Modulverkettung dieser Grundverknüpfungen und der Negation aufbauen.

Logische Grundverknüpfungen von 2 Variablen $y = \lambda(x_1, x_0)$

UND-Verknüpfung

Ob in einer Leitung ein Strom fließt, hängt von einer bestimmten Wertekombination der Variablen x_0 und x_1 ab. Bei einer Reihenschaltung kann y nur dann 1 sein, wenn x_0 und x_1 gleichzeitig 1 sind, d.h. beide Schalter geschlossen sind.



x_1	x_0	y
0	0	0
0	1	0
1	0	0
1	1	1

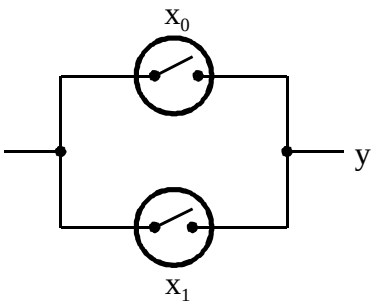
Formale Beschreibung der Bedingung:

$y = x_1 \wedge x_0 \qquad (= x_1 x_0);$

Es handelt sich dabei um eine UND-Verknüpfung (AND, Konjunktion).

ODER-Verknüpfung

Schaltet man die Elemente parallel, ergibt sich eine ODER-Verknüpfung (OR, Disjunktion), welche die Schaltfunktion y = l immer dann erfüllt, wenn eine der beiden Variablen x_0 oder x_1 den Wert 1 hat.



x_1	x_0	y
0	0	0
0	1	1
1	0	1
1	1	1

Formale Beschreibung der Bedingung:

$$y = x_1 \vee x_0;$$

Vollständiger Satz der Funktionen von zwei Variablen

Für zwei Variable gibt es $2^2 = 4$ Eingangskombinationen und $2^4 = 16$ Funktionen (y^2_0 ... y^2_{15}). Neben den Null-, Eins-, Identitäts- und Negationsfunktionen (6) gibt es acht Funktionen von 2 Variablen (x_1, x_0), die sich als Varianten von UND- oder ODER-Schaltungen mit Negationen an Ein- und Ausgängen darstellen lassen (Bild 2.10). Die verbleibenden zwei sind die "Äquivalenz" (y^2_9) und die "Antivalenz" (y^2_6). Letztere bezeichnet man heute meist als "Exklusives ODER (XOR)", sie ist auch identisch mit der "Zyklisch-ODER"-Funktion, dem Grundelement der "linearen" Schaltnetze. Bild 2.10 zeigt auch die algebraischen Funktionssymbole und die graphischen Schaltsymbole nach DIN EN 60617-12.

		Algebra. Darstellung	Funktionstabelle				Benennung	DIN EN 60617-12 Schaltsymbol
Ausgangsfunktion	y_0^2	0	0	0	0	0	Nullfunktion	0 ---------
	y_1^2	$x_1 \wedge x_0$	0	0	0	1	AND (Konjunktion)	&
	y_2^2	$x_1 \wedge \overline{x}_0$	0	0	1	0	(UND)	x_1, x_0 &
	y_3^2	x_1	0	0	1	1	(Identität)	x_1 ----------
	y_4^2	$\overline{x}_1 \wedge x_0$	0	1	0	0	(UND)	x_1, x_0 &
	y_5^2	x_0	0	1	0	1	(Identität)	x_0 ----------
	y_6^2	$x_1 \not\equiv x_0$	0	1	1	0	XOR (Antivalenz)	=1
	y_7^2	$x_1 \vee x_0$	0	1	1	1	OR (Disjunktion)	≥1
	y_8^2	$\overline{x_1 \vee x_0}$	1	0	0	0	NOR (Peirce-Fkt.)	≥1
	y_9^2	$x_1 \equiv x_0$	1	0	0	1	Äquivalenz	=
	y_{10}^2	$\overline{x}_0$	1	0	1	0	(Negation)	x_0 1
	y_{11}^2	$x_1 \vee \overline{x}_0$	1	0	1	1	Implikation	x_1, x_0 ≥1
	y_{12}^2	$\overline{x}_1$	1	1	0	0	(Negation)	x_1 1
	y_{13}^2	$\overline{x}_1 \vee x_0$	1	1	0	1	Implikation	x_1, x_0 ≥1
	y_{14}^2	$\overline{x_1 \wedge x_0}$	1	1	1	0	NAND (Sheffer-Fkt.)	&
	y_{15}^2	1	1	1	1	1	Einsfunktion	1 ----------
Eingangs-variablen	x_1		0	0	1	1		
	x_0		0	1	0	1		
Kombination der Eingangsvariablen			x^0	x^1	x^2	x^3		

Bild 2-10: Logische Funktionen von zwei Variablen

Logische Funktionen von n Variablen
Allgemein gibt es für n Eingangsvariablen

2^n Kombinationen und 2^{2^n} Funktionen.

Alle diese Funktionen können aber wie erwähnt durch Zusammenschaltung weniger Grundfunktionen realisiert werden. Die bekannteste Elementarfunktions-Gruppe besteht aus NEGATION, UND und ODER. Für verschiedene Anwendungen ist es aber zweckmäßig einen erweiterten Satz von Elementarfunktionen zu verwenden. Bild 2-11 zeigt einen gebräuchlichen Satz von elementaren logischen Funktionen.

Benennung	Formel	Schaltsymbol

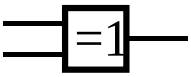
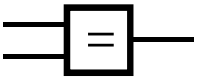
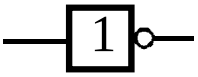
		DIN EN 60617-12
UND (AND)	$x_1 \wedge x_0$ $(x_1 x_0)$ $(x_1 \cdot x_0)$	
ODER (OR)	$x_1 \vee x_0$	
NAND	$\overline{x_1 \wedge x_0}$	
NOR	$\overline{x_1 \vee x_0}$	
XOR (Antivalenz)	$x_1 \not\equiv x_0$ $(x_1 \oplus x_0)$	
Äquivalenz	$x_1 \equiv x_0$	
Negation	$\overline{x}$	

Bild 2-11: Logische Funktionen und ihre Schaltsymbole

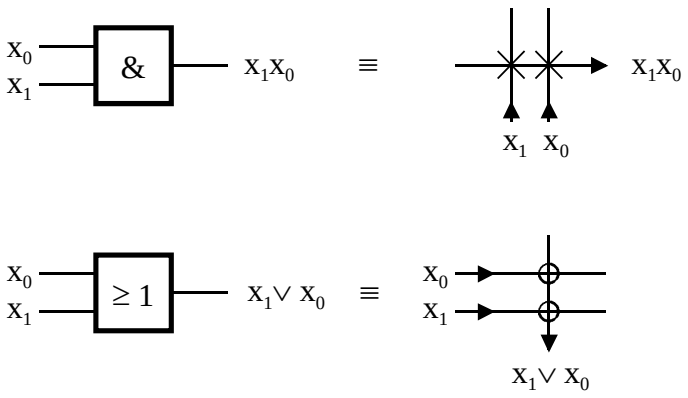


Bild 2-12: Graphische Darstellung der verdrahteten Logik (wired logic)

Verdrahtete Logik
Bild 2-12 zeigt die graphischen Darstellungen einer UND- und einer ODER-Schaltung in Form der so genannten "verdrahteten Logik" (wired logic) . Bei der UND-Schaltung entsteht eine logische "1", wenn alle angekreuzten Eingangslinien (X) auf "l" sind (wired AND).

Bei der ODER -Schaltung wird der Ausgang "l", wenn mindestens eine eingekreiste Linie (0) eine "l" zeigt (wired OR). Diese Symbolik wird im folgenden immer wieder benutzt, weil sie einfache, übersichtliche Darstellungen logischer Verknüpfungen erlaubt und auch der technischen Realisierung moderner Integrationstechnik (z.B. Programmierbare Logik, PLD) entspricht. Da diese Darstellung keine Auskunft gibt, welches die Eingänge bzw. Ausgänge sind, muss man Richtungspfeile verwenden, falls der Signalfluss nicht aus dem Kontext hervorgeht. In Bild 2-12 sind bei der verdrahteten UND-Schaltung die Ausgänge waagerecht und bei der ODER-Schaltung senkrecht gezeichnet. Dies entspricht der Anordnung von späteren Beispielen. Wir werden diese im folgenden öfter verwenden, sie erfordert kaum Pfeile für die Kennzeichnung der Wirkungsrichtung.

2.2.2.3 Rechenregeln der Schaltalgebra

Um auch komplexere logische Schaltungen systematisch entwerfen oder analysieren zu können, benötigt man Hilfsmittel. Ein solches Hilfsmittel ist die Schaltalgebra. Sie geht auf Claude E. Shannon zurück, der 1938 nachwies, dass die prinzipiellen Eigenschaften von Serien- und Parallelschaltungen aus Schaltern z.B. mit einer speziellen Booleschen Algebra, die nur zwei verschiedene Aussagen kennt, besonders gut beschrieben werden können. Shannon stützte sich dabei auf eine algebraische Struktur, die George Boole (1815 - 1864) als mathematisches Hilfsmittel zur systematischen und abstrakten Behandlung der Logik entwickelt hat.

Bei der Behandlung logischer Schaltungen mit Hilfe der Schaltalgebra oder auch Booleschen Algebra wird in einer symbolischen Beschreibungsweise lediglich die logische Funktion erfasst, ohne dabei die jeweiligen geräte- und schaltungstechnischen Ausführungen zu berücksichtigen. Die Schaltalgebra kann und soll daher auch keine Aussagen über Schaltzeiten, Signallaufzeiten, Spannungstoleranzen oder Belastbarkeit einer Schaltung geben.

Aus der Definition der UND-, ODER- und Negations-Funktionen folgen die Rechengesetze der Schaltalgebra. Sie bilden die Grundlage für Vereinfachungsverfahren zur Erzielung technischer Vorteile (Bild 2-13).

Die beiden De Morgan'schen Gesetze beschreiben die Beziehungen zwischen einer beliebigen Funktion und ihrem Komplement:

1) Das Komplement einer UND-Verknüpfung von Variablen ist gleich der ODER -Verknüpfung der Komplemente dieser Variablen

2) Das Komplement einer ODER-Verknüpfung von Variablen ist gleich der UND-Verknüpfung der Komplemente dieser Variablen.

Der Beweis der Rechengesetze kann durch Vergleich der Funktionstabellen erfolgen. Als Beispiel wird in Bild 2-14 das erste De Morgan'sche Gesetz für zwei Variable bewiesen.

Konjunktion (UND)	Disjunktion (ODER)
Rechnen mit einer Variablen und 0 und 1	
$\overline{x} \wedge \overline{x} = \overline{x}$	$\overline{x} \vee \overline{x} = \overline{x}$
$\overline{x} \wedge x = 0$	$\overline{x} \vee x = 1$
$x \wedge x = x$	$x \vee x = x$
$\overline{x} \wedge 0 = 0$	$\overline{x} \vee 0 = \overline{x}$
$\overline{x} \wedge 1 = \overline{x}$	$\overline{x} \vee 1 = 1$
$x \wedge 0 = 0$	$x \vee 0 = x$
$x \wedge 1 = x$	$x \vee 1 = 1$
Kommutatives Gesetz	
$x_0 \wedge x_1 = x_1 \wedge x_0$	$x_0 \vee x_1 = x_1 \vee x_0$
Assoziatives Gesetz	
$x_2 \wedge x_1 \wedge x_0 = x_2 \wedge (x_1 \wedge x_0)$	$x_2 \vee x_1 \vee x_0 = x_2 \vee (x_1 \vee x_0)$
Distributives Gesetz	
$x_2 \wedge (x_1 \vee x_0) = (x_2 \wedge x_1) \vee (x_2 \wedge x_0)$	$x_2 \vee (x_1 \wedge x_0) = (x_2 \vee x_1) \wedge (x_2 \vee x_0)$
Vereinfachungsregel (aus distributivem Gesetz abgeleitet)	
$(x_1 \vee x_0) \wedge (x_1 \vee \overline{x}_0) = x_1$	$(x_1 \wedge x_0) \vee (x_1 \wedge \overline{x}_0) = x_1$

De Morgansche Gesetze

$$\overline{x_2 \wedge x_1 \wedge x_0 \ldots} = \overline{x}_2 \vee \overline{x}_1 \vee \overline{x}_0 \ldots$$

$$\overline{x}_2 \wedge \overline{x}_1 \wedge \overline{x}_0 \ldots = \overline{x_2 \vee x_1 \vee x_0 \ldots}$$

Bild 2-13: Rechengesetze der Schaltalgebra

x_1	x_0	$x_1 \wedge x_0$	$\overline{x_1 \wedge x_0}$	$\overline{x}_1$	$\overline{x}_0$	$\overline{x}_1 \vee \overline{x}_0$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

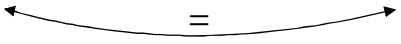


Bild 2-14: Beweis des 1. De Morgan'schen Gesetzes über die Wahrheitstabelle

2.2.3 Darstellung von Schaltfunktionen durch normierte Ausdrücke

a) Disjunktive Normalform (DNF)

Definition des Begriffes Minterm
Ein Minterm ist eine Schaltfunktion von n Eingangsvariablen, die nur für eine Kombination aller Eingangsvariablen gleich "1" wird und bei allen anderen Kombinationen von n Eingangsvariablen den Wert "0" hat.

Bei n Eingangsvariablen gibt es insgesamt 2^n Minterme (= Zahl der Eingangskombinationen). Ein Minterm ist nur darstellbar durch die konjunktive Verknüpfung aller n Eingangsvariablen bzw. Komplemente (vollständige Konjunktion oder Vollkonjunktion).

Beispiel: Die vier möglichen Minterme der Funktion $y = \lambda(x_0, x_1)$ sind $x^0 \ldots x^3$. Es gibt entsprechend 4 Mintermfunktionen $K_0 \ldots K_3$, die nur "1" für den jeweiligen Minterm werden.

	x_1	x_0	K_3	K_2	K_1	K_0
x^0	0	0	0	0	0	1
x^1	0	1	0	0	1	0
x^2	1	0	0	1	0	0
x^3	1	1	1	0	0	0

$$K_0 = \overline{x}_1 \overline{x}_0$$
$$K_1 = \overline{x}_1 x_0$$
$$K_2 = x_1 \overline{x}_0$$
$$K_3 = x_1 x_0$$

Wenn man Mintermfunktionen und damit die Minterme überlagert, d.h. disjunktiv miteinander verknüpft (ODER), kann man jede beliebige Funktion y zusammensetzen. Man nennt diese Form die **disjunktive Normalform**.

Disjunktive Normalform
Bild 2-15 zeigt das Beispiel einer Funktion mit drei Eingangsvariablen. Es werden die Minterme K_i aus den Eingängen bzw. den negierten Eingängen gebildet und zwar für jede Eingangskombination, für die die Ausgangsfunktion den Wert "l" hat:

$$K_2 = \overline{x}_2 x_1 \overline{x}_0$$
$$K_3 = \overline{x}_2 x_1 x_0$$
$$K_5 = x_2 \overline{x}_1 x_0$$
$$K_6 = x_2 x_1 \overline{x}_0$$
$$K_7 = x_2 x_1 x_0$$

Diese Konjunktionen enthalten alle Eingangsvariablen $x_0 \ldots x_2$ und werden deshalb Vollkonjunktionen genannt; sie werden auch als "Mintermfunktionen" bezeichnet. Eine ODER-Schaltung (Disjunktion) sorgt dafür, dass alle einzelnen Mintermfunktionen am Ausgang erscheinen. Die disjunktive Normalform (DNF) ist also die disjunktive Verknüpfung aller Minterme, für die die Ausgangsfunktion y den Wert 1 annimmt, d.h.:

$$y = K_2 \vee K_3 \vee K_5 \vee K_6 \vee K_7$$
$$= \overline{x}_2 x_1 \overline{x}_0 \vee \overline{x}_2 x_1 x_0 \vee x_2 \overline{x}_1 x_0 \vee x_2 x_1 \overline{x}_0 \vee x_2 x_1 x_0$$

Bild 2-16 zeigt die schaltungstechnische Realisierung der disjunktiven Normalform.

Eingangsvariable			Negation			Mintermfunktionen					Ausgangs-funktion
x_2	x_1	x_0	$\overline{x}_2$	$\overline{x}_1$	$\overline{x}_0$	K_7	K_6	K_5	K_3	K_2	y
0	0	0	1	1	1	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0	0	0	1	1
0	1	1	1	0	0	0	0	0	1	0	1
1	0	0	0	1	1	0	0	0	0	0	0
1	0	1	0	1	0	0	0	1	0	0	1
1	1	0	0	0	1	0	1	0	0	0	1
1	1	1	0	0	0	1	0	0	0	0	1

Bild 2-15: Wahrheitstabelle mit Mintermfunktionen

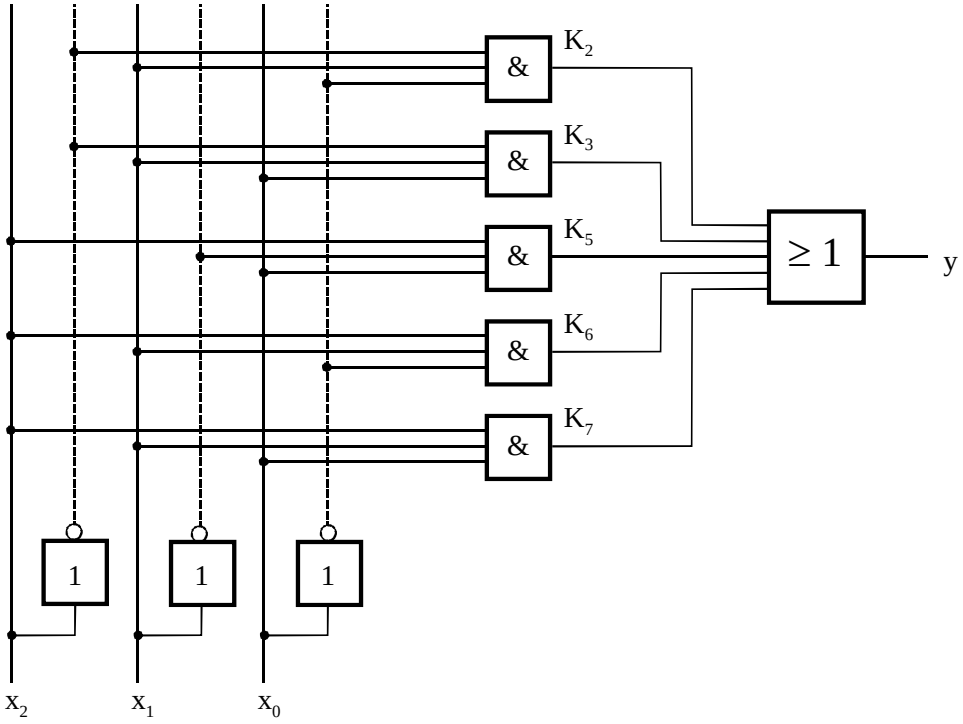


Bild 2-16: Schaltungstechnische Realisierung der disjunktiven Normalform

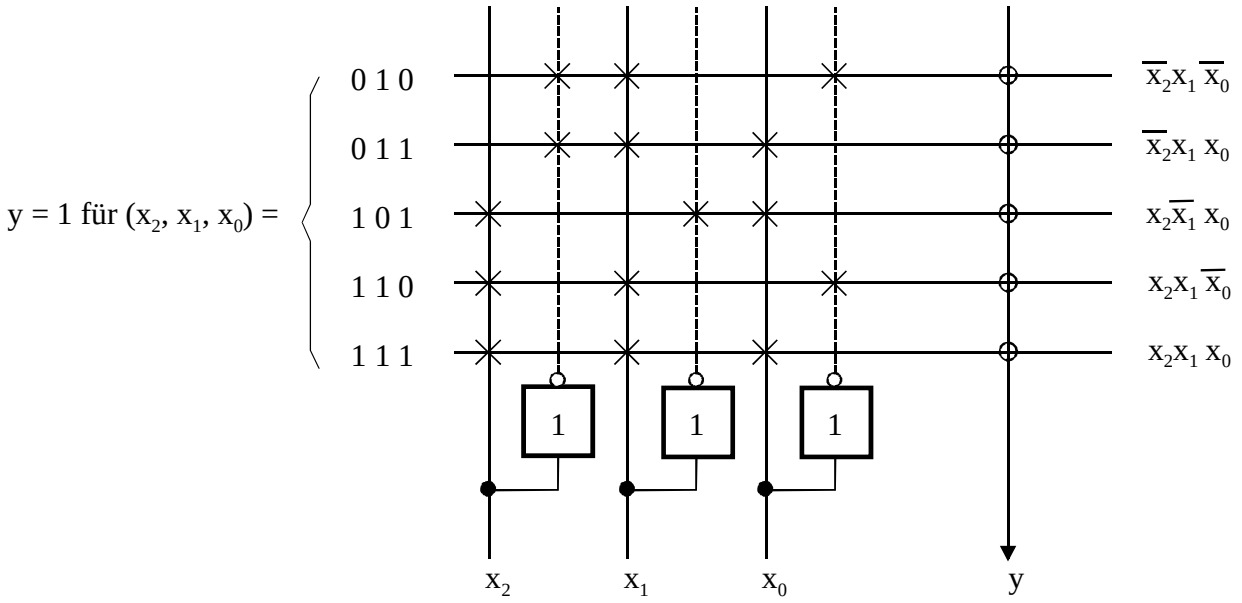


Bild 2-17: Verdrahtete Logik der disjunktiven Normalform

b) Konjunktive Normalform (KNF)

Definition des Begriffes Maxterm
Ein Maxterm ist eine Schaltfunktion von n Eingangsvariablen, die nur für eine Kombination aller n Eingangsvariablen gleich "0" wird und bei allen anderen Kombinationen der n Eingangsvariablen den Wert "l" hat.

Bei n Eingangsvariablen gibt es insgesamt 2^n Maxterme (= Zahl der Eingangskombinationen). Ein Maxterm ist darstellbar durch die disjunktive Verknüpfung aller n Eingangsvariablen bzw. ihrer Komplemente (vollständige Disjunktion oder Volldisjunktion).

Beispiel: Die vier möglichen Maxterme der Funktion $y = \lambda(x_1, x_0)$ ergeben 4 Maxtermfunktionen $D_0 \ldots D_3$.

x_1	x_0	D_3	D_2	D_1	D_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

$D_0 = x_1 \vee x_0$
$D_1 = x_1 \vee \overline{x}_0$
$D_2 = \overline{x}_1 \vee x_0$
$D_3 = \overline{x}_1 \vee \overline{x}_0$

Wenn man Maxterme überlagert, d.h. konjunktiv miteinander verknüpft (UND), kann man jede beliebige Funktion y zusammensetzen. Man nennt diese Form die **konjunktive Normalform**.

Konjunktive Normalform
Die konjunktive Normalform (KNF) ist die duale Form zur disjunktiven Normalform (DNF). In Bild 2-18 ist für das gleiche Beispiel wie im Bild 2-15 die Wahrheitstabelle angegeben. Für jede Eingangskombination, die am Ausgang eine "0" erzeugt, wird durch entsprechende Invertierung der Eingangsanschlüsse eine ODER-Funktion (Disjunktion) erzeugt.

Die vereinigende UND-Schaltung (Konjunktion) erzeugt für alle Eingangskombinationen eine "l", wenn keine "0" in den Maxtermen vorkommt. Die konjunktive Normalform (KNF) ist die konjunktive Verknüpfung aller Maxterme, für die die Ausgangsfunktion den Wert "0" annimmt, d.h.

$$\begin{aligned} y &= D_4 \wedge D_1 \wedge D_0 \\ &= (\overline{x}_2 \vee x_1 \vee x_0) \wedge (x_2 \vee x_1 \vee \overline{x}_0) \wedge (x_2 \vee x_1 \vee x_0) \end{aligned}$$

Bild 2-19 zeigt die schaltungstechnische Realisierung der konjunktiven Normalform. Der Vergleich der schaltungstechnischen Realisierungen in KNF und DNF zeigt, dass die KNF immer dann die weniger aufwendige Methode ist, wenn mehr Einsen als Nullen am Ausgang vorkommen. Im Gegensatz dazu ist die DNF günstiger bei mehr Nullen als Einsen.

Eingangsvariable			Negation			Maxtermfunktion			Ausgangs-funktion
x_2	x_1	x_0	$\overline{x}_2$	$\overline{x}_1$	$\overline{x}_0$	D_4	D_1	D_0	y
0	0	0	1	1	1	1	1	0	0
0	0	1	1	1	0	1	0	1	0
0	1	0	1	0	1	1	1	1	1
0	1	1	1	0	0	1	1	1	1
1	0	0	0	1	1	0	1	1	0
1	0	1	0	1	0	1	1	1	1
1	1	0	0	0	1	1	1	1	1
1	1	1	0	0	0	1	1	1	1

Bild 2-18: Wahrheitstabelle mit Maxtermfunktionen

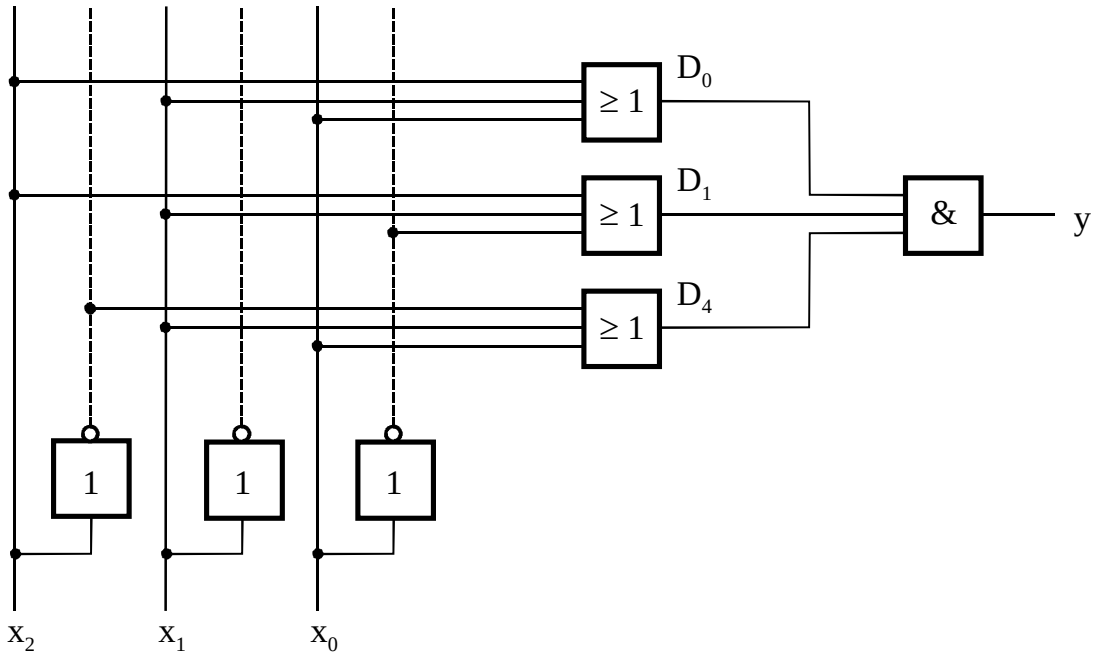


Bild 2-19: Schaltungstechnische Realisierung des Beispiels in konjunktiver Normalform

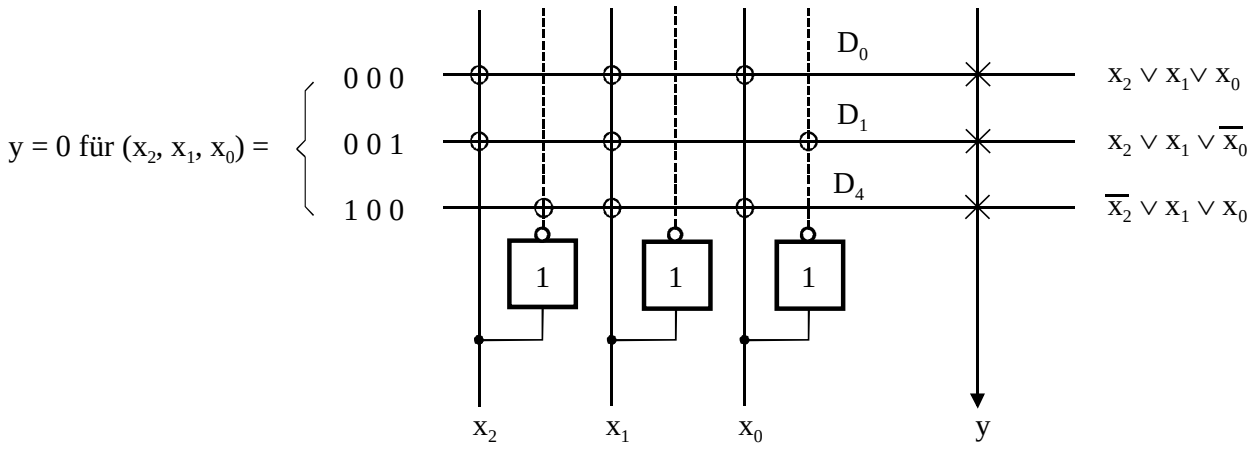


Bild 2-20: Verdrahtete Logik der konjunktiven Normalform

2.3 Minimierung von Schaltfunktionen

Eine Schaltfunktion kann durch verschiedene Ausdrücke in DNF oder KNF dargestellt werden. So stellen der Ausdruck $y = x_2x_1x_0 \vee x_1x_0 \vee x_1\overline{x}_0$ und der Ausdruck $y = x_1$ dieselbe Funktion dar. Das Aufsuchen möglichst einfacher und kurzer Lösungen zu einer gegebenen Funktion nennt man Minimierung. So ist sie eine Methode zur Realisierung einer zweistufigen Schaltfunktion mit minimalem Aufwand. "Aufwand" ist hier definiert als die Summe sämtlicher Gattereingänge einer Schaltung, da diese ganz grob den technischen Aufwand widerspiegelt, der zur Herstellung einer Schaltung notwendig ist bzw. die (Silizium-)Fläche bestimmt, die bei einer integrierten Halbleiterschaltung benötigt wird.

Zur Minimierung von Schaltfunktionen können für Problemstellungen geringer Komplexität graphische heuristische Verfahren eingesetzt werden. Für komplexe Schaltfunktionen gibt es sowohl heuristische als auch algorithmische Verfahren, wobei die algorithmischen Verfahren eine optimale Lösung im obigen Sinne liefern und rechnerunterstützt eingesetzt werden. Für diese beiden Ansätze werden im Folgenden Beispiele erläutert.

a) Minimierung mit Hilfe der Schaltalgebra

Mit den Rechenregeln der Bool'schen Algebra (vgl. Bild 2-13) können die algebraischen Ausdrücke einer Schaltfunktion minimiert werden. Insbesondere mit Hilfe der Vereinfachungsregel können Terme verkürzt oder redundante Terme eliminiert werden.

Beispiel:

$$y = \underbrace{\bar{x}_3\bar{x}_2\bar{x}_1x_0}_{(1)} \vee \underbrace{\bar{x}_3\bar{x}_2x_1\bar{x}_0}_{(2)} \vee \underbrace{\bar{x}_3\bar{x}_2x_1x_0}_{(3)} \vee \underbrace{\bar{x}_3x_2\bar{x}_1\bar{x}_0}_{(4)} \vee \underbrace{\bar{x}_3x_2\bar{x}_1x_0}_{(5)}$$

$$y = \underbrace{\bar{x}_3\bar{x}_2\bar{x}_1x_0}_{(1)} \vee \underbrace{\bar{x}_3x_2\bar{x}_1x_0}_{(5)} \vee \underbrace{\bar{x}_3\bar{x}_2x_1\bar{x}_0}_{(2)} \vee \underbrace{\bar{x}_3\bar{x}_2x_1x_0}_{(3)} \vee \underbrace{\bar{x}_3x_2\bar{x}_1\bar{x}_0}_{(4)} \vee \underbrace{\bar{x}_3x_2\bar{x}_1x_0}_{(5)}$$

$$y = \bar{x}_3\bar{x}_1x_0\underbrace{(\bar{x}_2 \vee x_2)}_{=1} \vee \bar{x}_3\bar{x}_2x_1\underbrace{(\bar{x}_0 \vee x_0)}_{=1} \vee \bar{x}_3x_2\bar{x}_1\underbrace{(\bar{x}_0 \vee x_0)}_{=1}$$

$$y = \bar{x}_3\bar{x}_1x_0 \vee \bar{x}_3\bar{x}_2x_1 \vee \bar{x}_3x_2\bar{x}_1$$

In dem Beispiel wird zunächst ein redundanter Term (5) hinzugefügt. Dies dient dazu, im Folgeschritt auch den Term (1) zu minimieren. Die Terme (1) und (5), (2) und (3), (4) und (5) lassen sich durch die Vereinfachungsregel minimieren.

Die heuristische Minimierung mit Hilfe der Schaltalgebra erfordert eine gewisse Übung im Umgang mit Schaltfunktionen.

b) Minimierung mit Hilfe von Karnaugh-Veith-Diagrammen

Das Karnaugh-Veith-Diagramm wird eingesetzt, um die Nachbarschaftsverhältnisse der algebraischen Terme graphisch leicht erkennbar zu machen. Es ist (ebenso wie eine Tabelle) eine Auflistung aller Kombinationen der Eingangsvariablen, wobei jedem Feld genau eine Kombination zugeordnet wird und von jedem Feld zu seinem Nachbarn sich genau eine Variable ändert. Bei n Variablen ergeben sich 2^n Felder, wobei die Indizierung beliebig, aber eindeutig sein muss (Bild 2-21 - Bild 2-23). Im Folgenden soll immer die Indizierung wie in den Bildern angegeben verwendet werden.

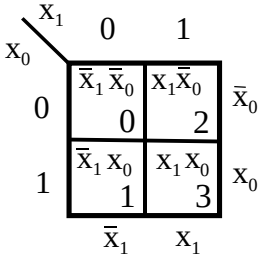


Bild 2-21: Karnaugh-Veith-Diagramm mit zwei Variablen (x_1, x_0)

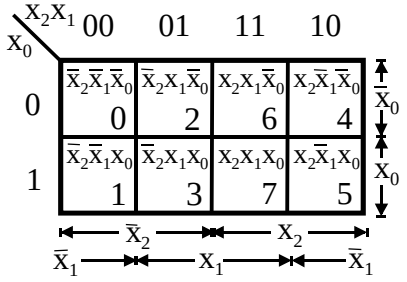


Bild 2-22: Karnaugh-Veith-Diagramm mit drei Variablen (x_2, x_1, x_0)

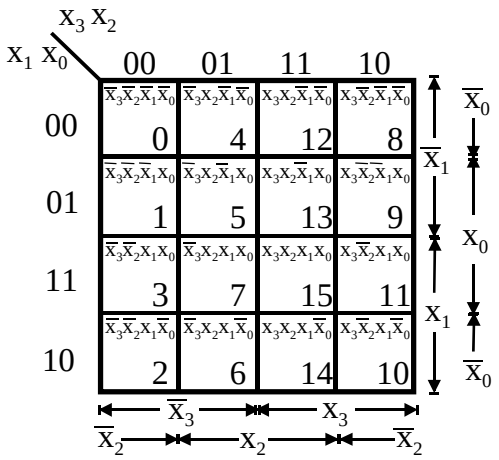


Bild 2-23: Karnaugh-Veith-Diagramm mit vier Variablen (x_3, x_2, x_1, x_0)

	x_3	x_2	x_1	x_0	y
(0)	0	0	0	0	0
(1)	0	0	0	1	1
(2)	0	0	1	0	1
(3)	0	0	1	1	1
(4)	0	1	0	0	1
(5)	0	1	0	1	1
(6)	0	1	1	0	0
(7)	0	1	1	1	0
(8)	1	0	0	0	0
(9)	1	0	0	1	0
(10)	1	0	1	0	0
(11)	1	0	1	1	0
(12)	1	1	0	0	0
(13)	1	1	0	1	0
(14)	1	1	1	0	0
(15)	1	1	1	1	0

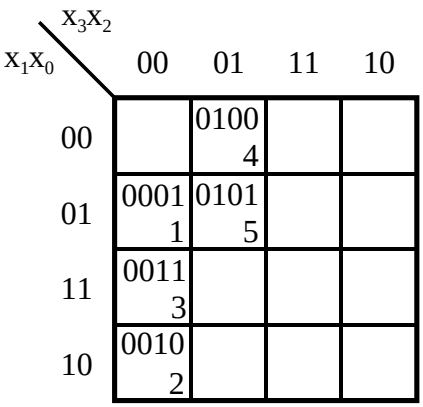


Bild 2-24: Darstellung des Beispiels in Tabelle und Karnaugh-Veith-Diagramm

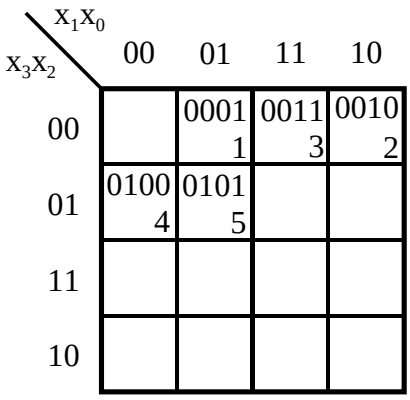


Bild 2-25: Darstellung des Beispiels aus Bild 2-24 mit vertauschter Indizierung

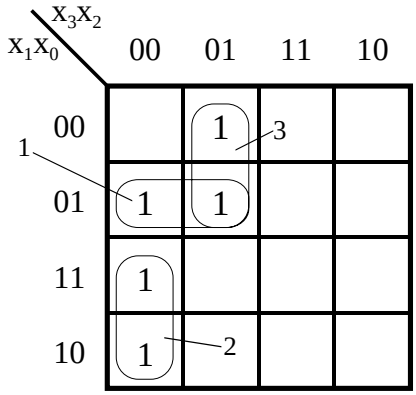
Minimierung im Karnaugh-Veith-Diagramm

Ziel der Minimierung ist wie in Beispiel a) Terme für die Anwendung der Vereinfachungsregel zu finden. Diese lässt sich immer dann anwenden, wenn in Termen eine oder mehrere Variablen komplementär zueinander sind. Im Karnaugh-Veith-Diagramm wird dies durch Nachbarschaftsverhältnisse sichtbar.

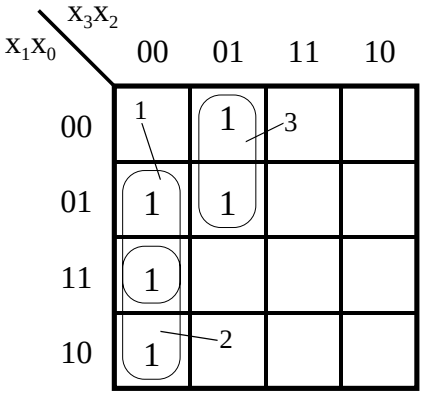
Die Minimierung erfolgt daher durch Zusammenfassung von jeweils zwei benachbarten Feldern, wobei einzelne Terme auch mehrfach erfasst werden können (entspricht dem Hinzufügen redundanter Terme wie in Beispiel a) (siehe Bild 2-26). Weiterhin kann man auch über die Ränder hinaus zusammenfassen.

Die entstehenden Schleifen fassen Terme zusammen, in denen bestimmte Variable nach den Gesetzen der Booleschen Algebra eliminiert werden. Das Beispiel in Bild 2-26 zeigt auch, dass man evtl. die einzelnen Terme auf verschiedene Weisen zusammenfassen kann. Angrenzende Zweier-Schleifen kann man durch Elimination der unterscheidenden Variable in Vierer-Schleifen zusammenfassen (Bild 2-27a, b). Entsprechendes gilt auch für weitere Zusammenfassungen (Bild 2-27c, d). Isolierte Terme können nicht minimiert werden und müssen vollständig wiedergegeben werden (siehe Beispiel Bild 2-27a, "Einerschleife").

Wegen der Dualität zwischen DNF und KNF kann die Minimierung nach Karnaugh ebenso für die konjunktive Normalform angewendet werden; bei der DNF werden die "Einsen" - bei der KNF die "Nullen" dargestellt und zusammengefasst (Bild 2-28).



Schl. 1 elim. x_2: $\overline{x}_3\overline{x}_1x_0$
Schl. 2 elim. x_0: $\overline{x}_3\overline{x}_2x_1$
Schl. 3 elim. x_0: $\overline{x}_3x_2\overline{x}_1$



Schl. 1 elim. x_1: $\overline{x}_3\overline{x}_2x_0$
Schl. 2 elim. x_0: $\overline{x}_3\overline{x}_2x_1$
Schl. 3 elim. x_0: $\overline{x}_3x_2\overline{x}_1$

1. Lösung: $y = \overline{x}_3\overline{x}_1x_0 \vee \overline{x}_3\overline{x}_2x_1 \vee \overline{x}_3x_2\overline{x}_1$
2. Lösung: $y = \overline{x}_3\overline{x}_2x_0 \vee \overline{x}_3\overline{x}_2x_0 \vee \overline{x}_3x_2\overline{x}_1$
Normalform: $y = \overline{x}_3\overline{x}_2\overline{x}_1x_0 \vee \overline{x}_3\overline{x}_2x_1\overline{x}_0 \vee \overline{x}_3\overline{x}_2x_1x_0 \vee \overline{x}_3x_2\overline{x}_1\overline{x}_0 \vee \overline{x}_3x_2\overline{x}_1x_0$

Bild 2-26: Beispiele für die Minimierung der gegebenen Funktion y mit vier Variablen (DNF) mit Hilfe eines Karnaugh-Veith-Diagramms

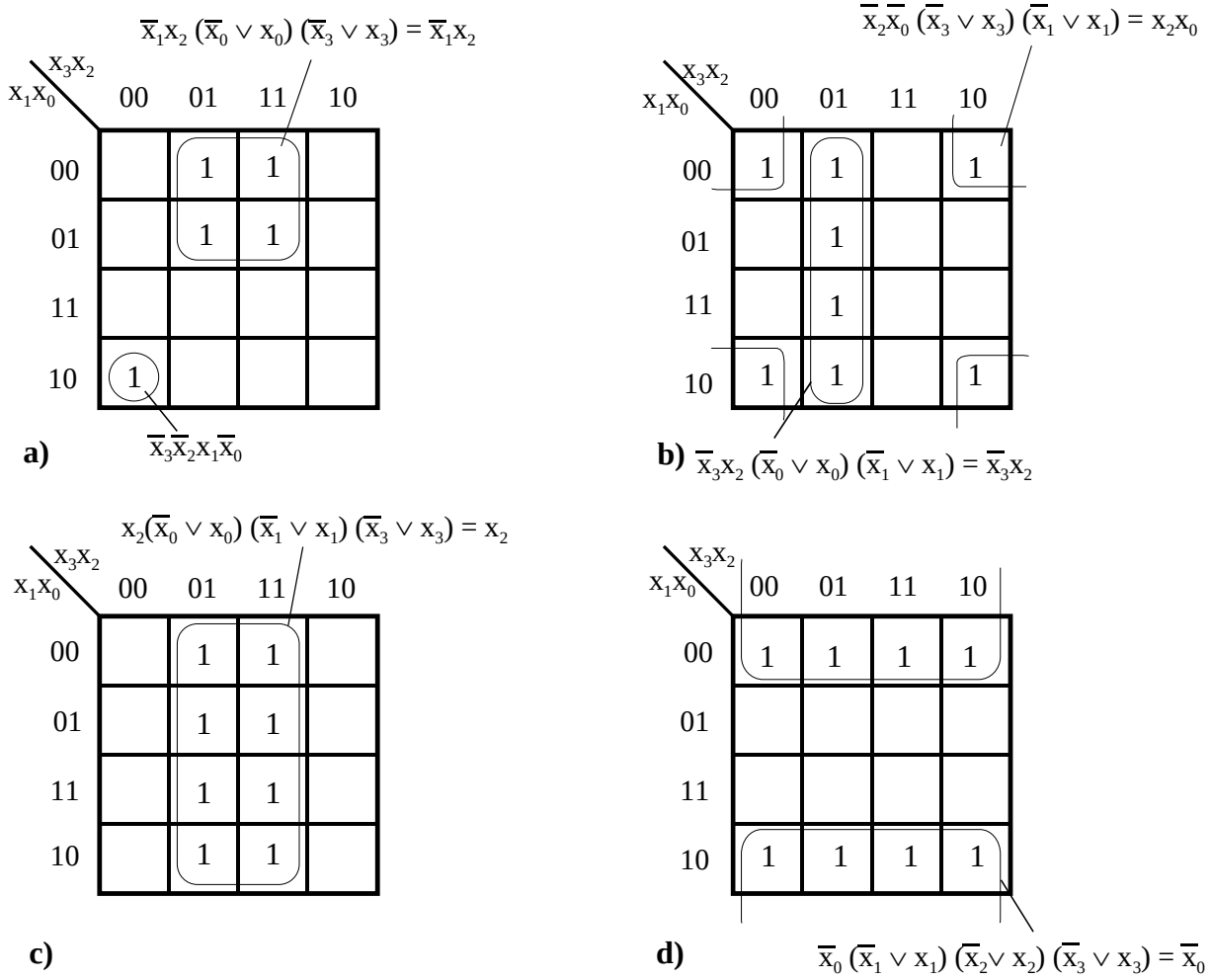
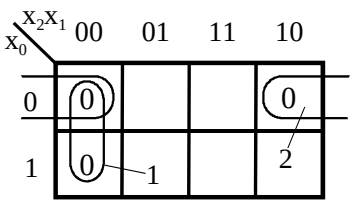


Bild 2-27: Weitere Möglichkeiten der Zusammenfassung von vier und acht Feldern

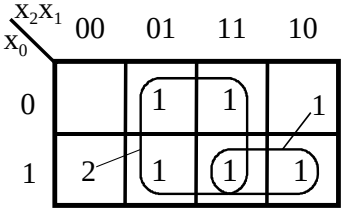
Minimierung einer KNF



EK: $0 \rightarrow x$
$1 \rightarrow \overline{x}$

Schl. 1 elim. x_0: $x_2 \vee x_1$
Schl. 2 elim. x_2: $x_1 \vee x_0$
$y_K = (x_2 \vee x_1) \wedge (x_1 \vee x_0)$

Vergleich mit der Minimierung der DNF



EK: $0 \rightarrow \overline{x}$
$1 \rightarrow x$

Schl. 1 elim. x_1: $x_0\ x_2$
Schl. 2 elim. $x_0\ x_2$: x_1
$y_D = x_2\ x_0 \vee x_1$

Ausrechnung nach dem distributiven Gesetz: $y_D = x_2x_0 \vee x_1 = (x_2 \vee x_1) \wedge (x_0 \vee x_1) = y_K$

Bild 2-28: Beispiel für die Minimierung einer KNF und der Vergleich zur DNF

Es gibt Fälle, in denen die Schaltfunktionen nicht bei allen Eingangskombinationen definiert ist, sog. partielle Funktionen. Der Zustand der Schaltfunktion y wird bei derartigen Eingangskombinationen mit einem X oder einem - Symbol gekennzeichnet, so genanntes don't care.

Für die Minimierung der Schaltfunktion können diese "don't care"-Terme jeweils so gesetzt werden, das die Schleifen maximal, d. h. die Terme minimal werden, und das sowohl für die DNF als auch die KNF.

Das folgende Beispiel zeigt die Minimierung mit Berücksichtigung der "don't cares" bei einer partiellen Funktion.

i	x_3	x_2	x_1	x_0	y
0	0	0	0	0	1
1	0	0	0	1	X
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	X
12	1	1	0	0	0
13	1	1	0	1	X
14	1	1	1	0	X
15	1	1	1	1	1

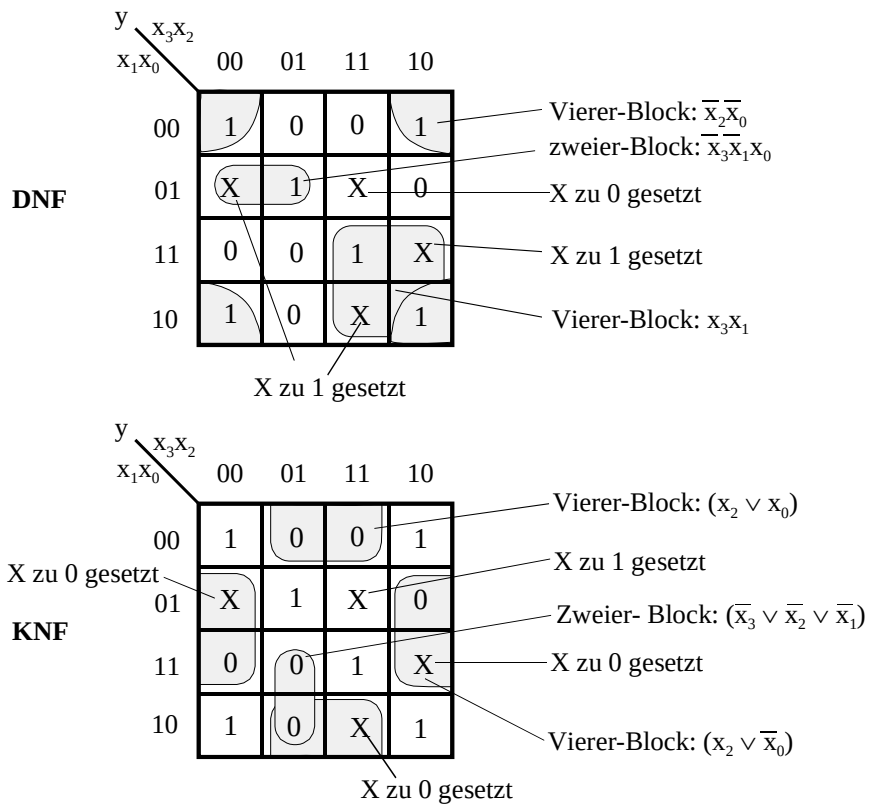


Bild 2-29: Minimierung partieller Funktionen

Als Ergebnis für das Beispiel erhalten wir mit der DNF ohne Berücksichtigung der don't cares

$$y_{min} \underset{0}{=} \overline{x}_2\overline{x}_0 \vee x_3x_2x_1x_0 \vee \overline{x}_3x_2\overline{x}_1x_0$$

mit Berücksichtigung der don't cares

$$y_{min} \underset{X}{=} \overline{x}_2\overline{x}_0 \vee x_3x_1 \vee \overline{x}_3\overline{x}_1x_0$$

c) Minimierung nach Quine-McCluskey

Das Verfahren von Quine-McCluskey ist ein Beispiel für einen algorithmischen Ansatz, bei dem die Nachbarschaften der Minterme tabellarisch gesucht werden. Es lässt sich leicht als Computerprogramm implementieren im Gegensatz zum Karnaugh-Veith-Verfahren.

Beim Quine-McCluskey-Algorithmus gibt es in der Literatur zwei grundsätzlich unterschiedliche Formen der Darstellung hinsichtlich Anschaulichkeit und mathematischem Formalismus. Der Vollständigkeit halber werden hier beide Formen beschrieben. Die Funktionstabelle aus Bild 2-29 soll in beiden Fällen als Beispiel dienen.

Verfahren mit binären Vektoren

Man kann die Eingangskombinationen, die zu einem Minterm gehören als Vektor auffassen, so dass zu dieser Funktion die Vektoren (0, 0, 0, 0), (0, 0, 1, 0), (0, 1, 0, 1), (1, 0, 0, 0), (1, 0, 1, 0) und (1, 1, 1, 1) für den Ausgangswert 1 und die Vektoren (0, 0, 0, 1), (1, 0, 1, 1), (1, 1, 0, 1) und (1, 1, 1, 0) für die don't cares gehören.

Der nächste Schritt ist es diese binären Vektoren in Gruppen zu sortieren, wobei jedes Element einer Gruppe dieselbe Anzahl von Einsen hat. Es gibt bei n Eingangsvariablen n +1 Gruppen, in diesem Beispiel die folgenden fünf:

Gruppe 0:	0000
Gruppe 1:	0001
	0010
	1000
Gruppe 2:	0101
	1010
Gruppe 3:	1011
	1101
	1110
Gruppe 4:	1111

Der Sinn der Sortierung in Gruppen ist die Algorithmisierung des Findens von Nachbarschaften. Es können nur zwei Terme bzw. 2 Vektoren miteinander kombiniert werden, wenn sie sich in genau einer Stelle unterscheiden. Die dazu notwendige Bedingung ist, dass sie sich in der Anzahl der Einsen um genau eins unterscheiden. Es handelt sich hier um eine notwendige nicht aber um eine hinreichende Bedingung, deshalb muss jeder Term einer Gruppe mit jedem Term der nächsten Gruppe verglichen werden. Wenn sie sich tatsächlich nur in einer Stelle unterscheiden, dann werden sie zu einem neuen Term kombiniert, in welchem die differierende Stelle durch eine Strich (don't care) ersetzt wird. Zum Beispiel: 0010 und 0000 sind benachbart und werden zu 00-0 kombiniert. Der Prozess, Vergleichen jedes Elements einer Gruppe mit jedem Element der nächsten Gruppe, wird solange fortgesetzt bis keine neuen Terme mehr gefunden werden.

Tabelle 2-1 zeigt die sogenannte Kürzungstabelle, wobei die Originalgruppen auf der linken Seite stehen. Die zweite Spalte enthält alle Kombinationen aus der ersten Spalte und die dritte Spalte Kombinationen aus der zweiten Spalte. Die Terme, die zu einem neuen kombiniert werden, werden als benutzt markiert. In der zweiten Spalte gibt es 4 Terme die nicht markiert sind: 000-, 0-01, -101 und 11-1. D. h. diese Terme können nicht weiter reduziert werden und werden als sogenannte Primimplikanten bezeichnet. Die dritte Spalte enthält noch zwei weitere Primimplikanten: -0-0, 1-1-. Terme die doppelt erzeugt werden können gestrichen werden.

Tabelle 2-1: Kürzungstabelle

Gruppe 0:	0000	✓	Gruppe 0/1:	000-		Gruppe 0/1/2:	-0-0	
Gruppe 1:	0001	✓		00-0	✓		~~-0-0~~	
	0010	✓		-000	✓	Gruppe 1/2/3:		
	1000	✓	Gruppe 1/2:	0-01				
Gruppe 2:	0101	✓		-010	✓	Gruppe 2/3/4:	1-1-	
	1010	✓		10-0	✓		~~1-1-~~	
Gruppe 3:	1011	✓	Gruppe 2/3:	101-	✓			
	1101	✓		-101				
	1110	✓		1-10	✓			
Gruppe 4:	1111	✓	Gruppe 3/4	1-11	✓			
				11-1				
				111-	✓			

Der nächste Schritt ist es, herauszufinden welche Minterme durch welche Primimplikanten abgedeckt werden. Dazu wird eine sogenannte Primimplikantentabelle aufgestellt (Tabelle 2-2) Jeder notwendige Minterm bekommt eine Spalte – don't care Minterme also nicht – und jeder Primimplikant eine Zeile. Es werden alle Stellen in der Tabelle markiert, an denen ein Primimplikant einen Minterm abdeckt. Da die Striche im Primimplikanten don't care bedeuten, deckt z. B. der Primimplikant -0-0 die Minterme 0000, 0010, 1000 und 1010 ab. Ziel ist es mit möglichst wenig Primimplikanten alle Minterme abzudecken.

Tabelle 2-2 Primimplikantentabelle

	0000	0010	0101	1000	1010	1111		
000-	X						redundant	$\overline{x}_3\overline{x}_1x_0$
0-01			X				wählbar	$x_2\overline{x}_1x_0$
-101			X				wählbar	
11-1						X	eingeschränkt wählbar	
-0-0	*X*	**X**		**X**	*X*		unverzichtbar	$\overline{x}_2\overline{x}_0$
1-1-					X	X	eingeschränkt wählbar	x_3x_1

Wird in der Tabelle ein Minterm nur durch einen ganz bestimmten Primimplikanten abgedeckt, dann spricht man von einem sogenannten Kernprimimplikanten. **Dies ist erkennbar dadurch, das in der Spalte nur ein Kreuz ist.**

Diese Kernprimimplikanten sind unverzichtbar. Hier gibt es nur einen Kernprimimplikanten -0-0, der für die Minterme 0010 und 1000 unverzichtbar ist. Dieser Primimplikant deckt aber auch die Minterme 0000 und 1010 ab (kursive Kreuze), so dass der Primimplikant 000- redundant ist. Für die Minterme 0101 und 1111 muss eine Auswahl getroffen werden. Der Minterm 1111 wird sowohl vom Primimplikanten 11-1 als auch von 1-1- abgedeckt. Bedeutet man die Knotenbedingung das ein Minterm mit weniger Eingängen der einfachere ist so fällt Entscheidung für 1-1-. Die beiden Primimplikanten, die den Minterm 0101 abdecken, sind gleich berechtigt, so dass es hier zwei gleichberechtigte Lösungen gibt. Die vier Minterme, die übrig bleiben, führen zu den folgenden beiden Lösungen:

$$y = \overline{x}_3\overline{x}_1x_0 \vee \overline{x}_2\overline{x}_0 \vee x_3x_1$$
$$y = x_2\overline{x}_1x_0 \vee \overline{x}_2\overline{x}_0 \vee x_3x_1$$

Kochrezeptartig kann man das Verfahren wie folgt zusammenfassen:

Quine-McCluskey Kurzanleitung

1. Binärkombination jedes Minterms (**mit** don't cares) heraussuchen.
   In Gruppen einsortieren, wobei jeder Term einer Gruppe die gleiche Anzahl von Einsen hat. Diese Gruppen in aufsteigender Reihenfolge sortieren.
2. Jeden Term einer Gruppe mit jedem Term der **nächsten** Gruppe vergleichen. Sind zwei Terme logisch benachbart, werden sie zu einem neuen Term kombiniert, wobei die Bitstelle die unterschiedlich ist durch einen Strich (don't care) ersetzt wird.
3. Markieren jedes benutzten Terms.
4. Der Prozess „Vergleichen und Markieren" wird so lange wiederholt bis alle Kombinationsmöglichkeiten erschöpft sind.

5. Aufstellen der Primimplikantentabelle. Die Minterme (**ohne** don't cares) bilden die Spalten, die Primimplikanten die Zeilen. Alle Stellen, an denen ein Primimplikant einen Minterm abdeckt, werden mit einem Kreuz markiert.
6. Heraussuchen der unverzichtbaren Primimplikanten durch Lokalisieren der Spalten mit nur einem Kreuz.
7. Auswahl eines Minimums an Primimplikanten um alle Minterme abzudecken.
8. Die unverzichtbaren und die ausgewählten Minterme bilden disjunktiv verknüpft das minimierte Ergebnis.

Verfahren mit Indizes

Bei diesem Verfahren werden die Nachbarschaften durch die Indizes einer Eingangskombination ermittelt. Da die Indizes im Dezimalsystem angegeben werden, sind die Nachbarschaften nicht mehr unmittelbar zu erkennen, wie es in der linären Vektorschreibweise möglich ist. Die möglichen Eingangskombinationen werden aufsteigend mit Indexnummern (z.B. I_j mit $0 \leq j \leq 15$ für 4 Variable) versehen. Es gilt

– benachbarte Belegungen unterscheiden sich in der Anzahl der Bits um genau 1.
Damit werden Indexgruppen gebildet mit gleicher "1"-Anzahl, hier z. B. für 4 Variablen.

Indexgruppe	Index	
0	0	(0, 0, 0, 0)
1	1, 2, 4, 8	.
2	3, 5, 6, 9, 10, 12	.
3	7, 11, 13, 14	.
4	15	(1, 1, 1, 1)

– Bei benachbarten Belegungen ist die Differenz der Indizes eine Potenz von 2, der Exponent gibt dabei die Bitstelle an:

z. B. $I_{15} - I_7 = 8 = 2^3$

	x_3	x_2	x_1	x_0		
die Worte	1	1	1	1	$\hat{=}$	I_{15}
und	0	1	1	1	$\hat{=}$	I_7

unterscheiden sich in x_3.

Zur Minimierung werden die in der Schaltfunktion gegebenen 1-Belegungen der Indizes in die Indexgruppe einsortiert und systematisch Differenzen innerhalb benachbarter Gruppen gesucht, die a) eine 2er Potenz ergeben und b) größer als Null sind, d. h. Differenzen aller Indizes k, l

für die gilt $k \in I_{j+1}, l \in I_j \quad k > l$

und $k - l = 2^p,$

diese werden in eine so genannte erste Kürzungstabelle eingetragen. Die Einträge entsprechen allen möglichen 2er-Schleifen im Karnaugh-Veith-Diagramm.

Haben Einträge in der 1. Kürzungstabelle gleiche Differenzen ergeben (die gleiche Variable lässt sich aus den beiden Termen eliminieren), muss weiter geprüft werden, ob sich eine weitere

Variable aus beiden Termen eliminieren lässt (entspr. Bildung einer 4er-Schleife). Dazu wird eine weitere Kürzungstabelle mit Indizes k, l aus der Gruppe I_j/I_{j+1} gebildet, für die gilt

$$k - 1 = m - n = 2^p$$

sowie $k - m = l - n = 2^q$ $\quad (k > l \text{ und } m > n \text{ und } p \neq q)$

Die zugehörigen Differenzen 2^p, 2^q werden in die neue Kürzungstabelle eingetragen. Diese Vorgehensweise wird solange wiederholt, bis in den Kürzungstabellen keine neuen, gleichen Differenzen mehr auftreten (sich keine größeren Schleifen mehr bilden lassen).

Das Ergebnis besteht aus den Einträgen der letzten Kürzungstabelle (≙ größte Schleife) und den Einträgen der vorigen Tabellen, die keine gleichen Differenzen geliefert haben.

Da die Kürzungstabellen alle möglichen Differenzkombinationen (≙ alle möglichen Schleifen) enthalten, müssen die Einträge auf redundante Terme in einem weiteren Schritt überprüft werden. Partielle Funktionen können wieder mit don't cares wie bei der Karnaugh-Veith-Methode behandelt werden.

Anhand des Beispiels aus (Bild 2-29) soll die Minimierung exemplarisch durchgeführt werden.

$I^1 = \{0, 2, 5, 8, 10, 15\}$ "1"-Belegungen

$I^X = \{1, 11, 13, 14\}$ don't cares (zu 1 gewählt)

1. Indexgruppe

Gruppe	Indizes j
0	0
1	1, 2, 8
2	5, 10
3	11, 13, 14
4	15

1. Kürzungstabelle

Gruppe	Indizes k, l (Differenz)	
0 / 1	1,0 (1) 2,0 (2) 8,0 (8)	P_0
1 / 2	5,1 (4) 10,2 (8) 10,8 (2)	P_2
2 / 3	11,10 (1) 13,5 (8) 14,10 (4)	P_3
3 / 4	15,11 (4) 15,13 (2) 15,14 (1)	P_1

Gleiche Differenzen ergeben sich

zwischen Gruppe 0 / 1 und 1 / 2 Terme 2,0 und 10,8 (2)
8,0 und 10,2 (8)

zwischen Gruppe 1 / 2 und 2 / 3 Terme 5,1 und 14,10 (4)
10,2 und 13,5 (8)

zwischen Gruppe 3 / 4 und 2 / 3 Terme 11,10 und 15,14 (1)
14,10 und 15,11 (4)

Die Terme 1,0 (1) aus Gruppe 1 / 2 und 15, 13 (2) aus 3 / 4 werden als nicht weiter minimierbar (2er Schleifen) markiert (P_0, P_1).

Die nächste Stufe ergibt

2. Kürzungstabelle

Gruppe	Indizes k, l, m, n (Differenzen p, q)
0 / 1 / 2	10, 8, 2, 0 (2, 8) P_4 10, 2, 8, 0 (8, 2)
1 / 2 / 3	-
2 / 3 / 4	15, 14, 11, 10 (1,4) P_5 15, 11, 14, 10 (4,1)

Die Differenzen der Gruppen 1 / 2 / 3 fallen weg, weil gilt

Diff. 14, 10, 5, 1 ergibt $9 \neq 2^q$
13, 5, 10, 2 ergibt $3 \neq 2^q$

Die Terme 5, 1 und 13, 5 können nun nicht in die Kürzungstabelle übernommen werden und müssen nachträglich als Minimalterme P_2 und P_3 in der ersten Tabelle markiert werden. Es gibt in der zweiten Kürzungstabelle keine gleichen Differenzen in benachbarten Gruppen. Es muss also keine weitere Kürzungstabelle aufgestellt werden. Die Terme P_0 – P_5 stellen wiederum alle möglichen minimierbaren Terme dar (zwei 2er- und vier 4er-Schleifen) die sogenannten Primimplikanten, welche nun auf Redundanz überprüft werden.

Auswahl:

	X		0	2	5	8	10	15	= I^1
P_0	1, 0	(1)	X						redundant
P_1	15, 13	(2)						X	eingeschränkt wählbar
P_2	5, 1	(4)			X				wählbar
P_3	13, 5	(8)			X				wählbar
P_4	10, 8, 2, 0	(2, 8)	X	**X**		**X**	X		unverzichtbar
P_5	15, 14, 11, 10	(1,4)					X	X	eingeschränkt wählbar

Nichtredundante Terme ergeben sich, wenn ein Term der Indexmenge I^1 nur von einem Primimplikanten P_i abgedeckt wird (nur ein Kreuz in einer Spalte), in diesem Fall deckt P_4 die Minterme I_2 und I_8 ab.

P_4 deckt aber auch Minterm I_0 ab, somit ist P_0 redundant.

Minterm I_{15} wird sowohl von P_1 als auch P_5 abgedeckt, da aber P_5 mehr Kreuze als P_1 hat, was bedeutet sie bilden die größeren Schleifen, wird P_5 genommen. P_2 und P_3 sind gleichwertig, d. h. es gibt zwei gleichberechtigte Lösungen. Hier wird P_2 verwendet. Die Terme P_2, P_4 und P_5 werden also verwendet und müssen in Boolesche Gleichungen übersetzt werden.

Es gilt

$P_2 = \begin{matrix} 0 & \boxed{1} & 0 & 1 \\ 0 & \boxed{0} & 0 & 1 \end{matrix} = \overline{x}_3\overline{x}_1x_0$

$P_4 = \begin{matrix} \boxed{1} & 0 & \boxed{1} & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \boxed{0} & 0 & \boxed{0} & 0 \end{matrix} = \overline{x}_2\overline{x}_0$

$P_5 = \begin{matrix} 1 & \boxed{1} & 1 & \boxed{1} \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & \boxed{0} & 1 & \boxed{0} \end{matrix} = x_3x_1$

$$y_{min} = \overline{x}_3\overline{x}_1x_0 \vee \overline{x}_2\overline{x}_0 \vee x_3x_1$$

Tabelle 2-3: Vergleich der Verfahren von Karnaugh-Veith und Quine-McCluskey

	Karnaugh-Veith	Quine-McCluskey
Variablenzahl	≤ 6	beliebig
Anzahl der Min- bzw. der Maxterme	$\leq 2^6$	beliebig
Einbeziehung von don't care	entsprechend geometrischer Nachbarschaft	alle in Kürzungstabelle einbezogen; keine in Auswahltabelle einbezogen
alle y_{min} gefunden?	nein	ja, es werden systematisch alle gefunden
Verfahren	grafisch	tabellarisch, gut geeignet für Rechnerimplementierung

Tabelle 2-3 stellt die Verfahren nach Karnaugh-Veith und Quine-McCluskey gegenüber und Bild 2-30 zeigt eine Übersicht gängiger Minimierungsverfahren.

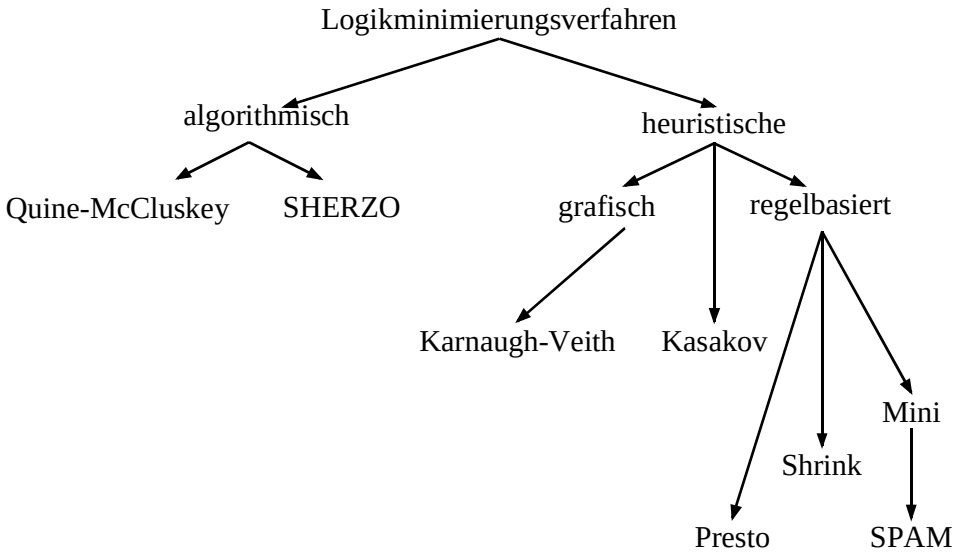


Bild 2-30: Gegenüberstellung von Minimierungsverfahren

2.4 Schaltnetzrealisierungen mit anderen Basissystemen

2.4.1 Die logischen Grundfunktionen mit NAND- und NOR-Schaltungen

Aufgrund der Rechengesetze der Schaltalgebra kann jedes der drei Grundfunktionselemente ausschließlich mit NAND- bzw. NOR-Gattern realisiert werden (Bild 2-31). Damit ist bewiesen, dass sowohl NAND- als auch NOR-Schaltungen vollständige logische Systeme darstellen, mit denen beliebige Schaltnetze verwirklicht werden können.

2.4.2 Normalformen mit NAND- und NOR-Schaltungen

Jede logische Funktion und jedes Schaltnetz in disjunktiver Normalform (DNF) lässt sich unmittelbar mit NAND-Gattern (

Bild **2-32**) realisieren. Entsprechend lässt sich die KNF mit NOR-Gattern (Bild 2-33) realisieren.

Bei diesen Umwandlungen bleiben die Topologie der Schaltung und die Eingangs- und Ausgangsvariablen erhalten, nur die Zwischenwerte erscheinen in invertierter Form.

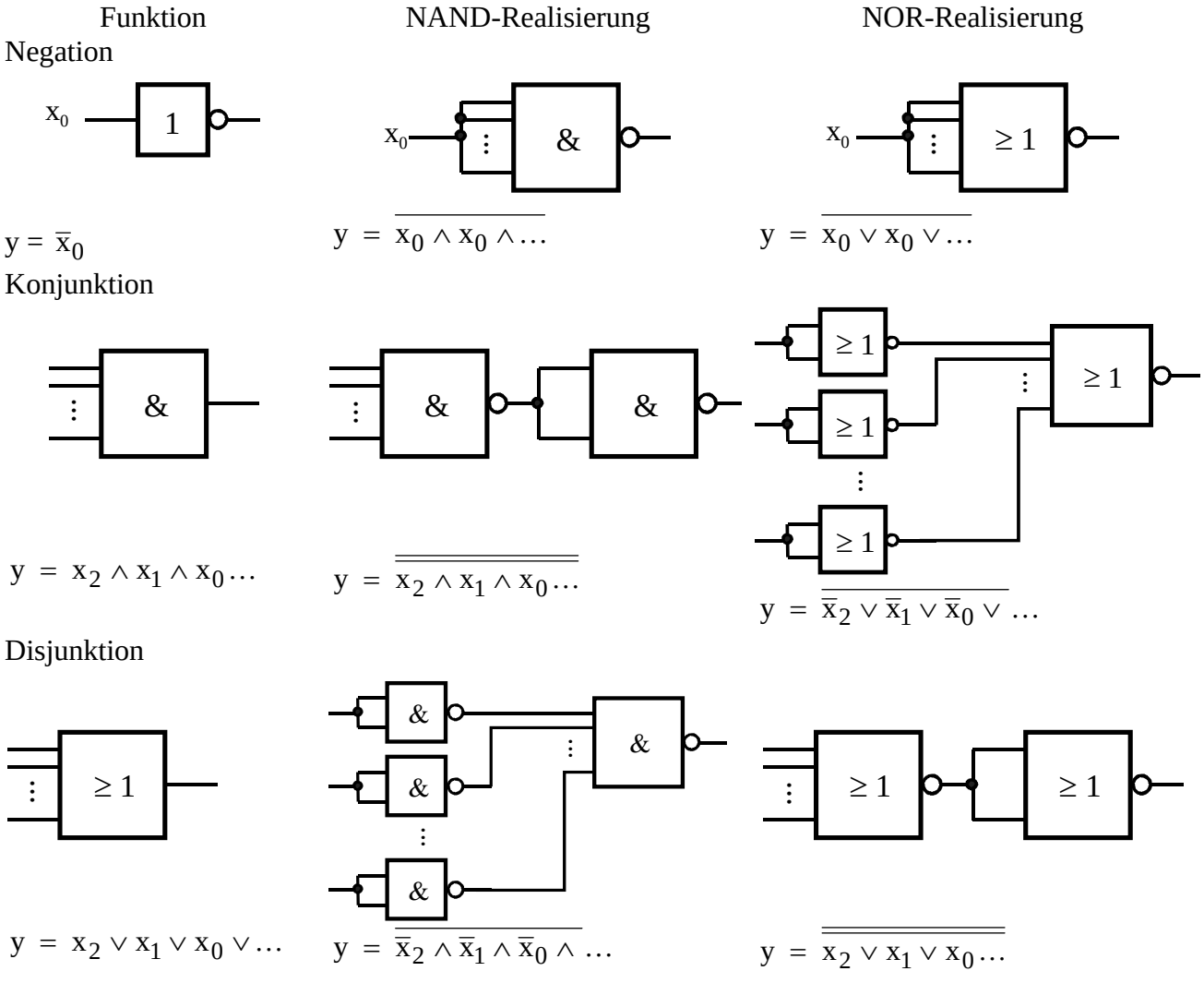


Bild 2-31: Realisierung der Grundfunktionen mit NAND- und NOR-Gattern

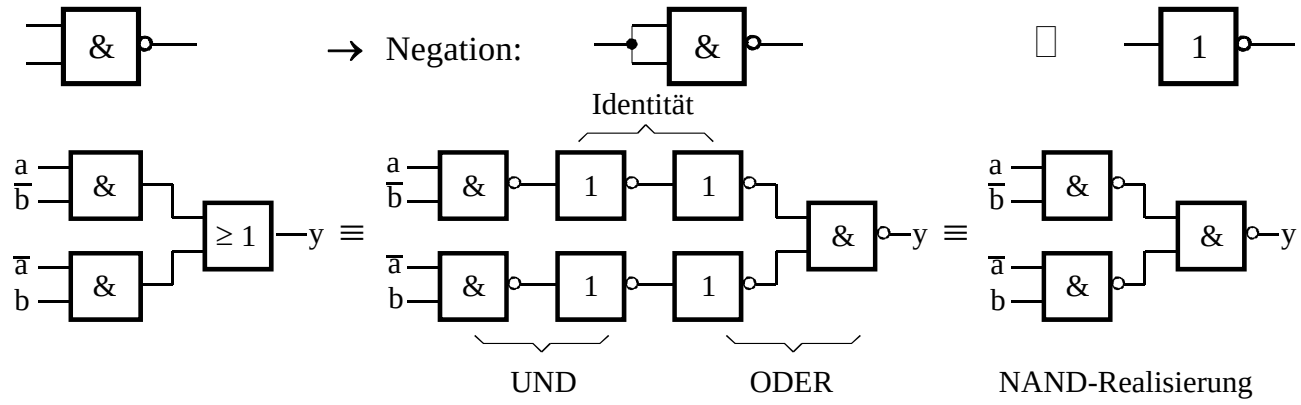


Bild 2-32: Realisierung einer Funktion in DNF mit NAND-Gattern

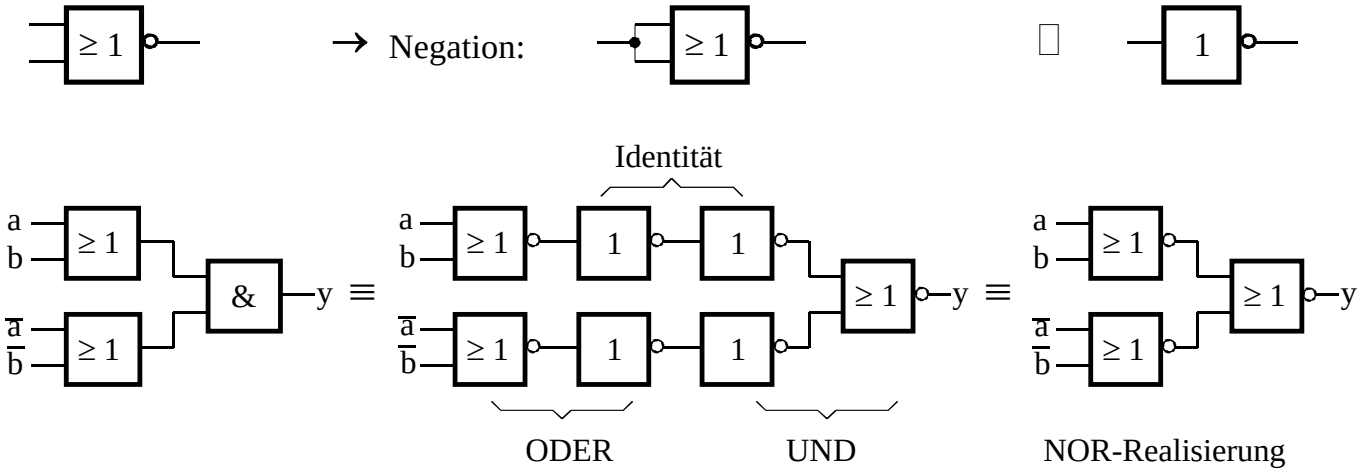


Bild 2-33: Realisierung einer Funktion in KNF mit NOR-Gattern

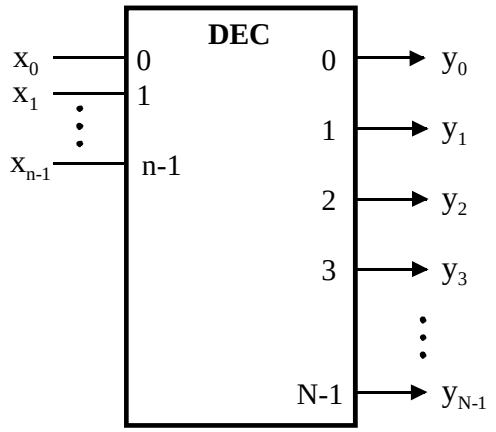
2.5 Beispiele für Schaltnetzrealisierungen

Auswahlbausteine

a) Decodierer

Bild 2-34a zeigt Schaltungssymbol und Wahrheitstabelle eines Decodierers. Aufgabe des Schaltnetzes ist es, jeder binär codierten Eingangskombination eine eindeutige Ausgangsleitung mit "1"-Belegung zuzuordnen. Damit können z. B: Zeilen bzw. Spalten im Speicherelementen, die in Matrixform angeordnet sind, selektiert werden.

Die Ausgangsfunktionen y_{n-1} … y_0 stellen den vollständigen Satz der Mintermfunktionen dar. Sie sind daher nicht minimierbar aber allein mit Konjunktionen realisierbar. Bild 2-34b zeigt die Realisierung mit UND-Gattern (n Eingänge) und als verdrahtete Logik.



$N = 2^n$

Wahrheitstabelle:

x_{n-1}	…	x_1	x_0	y_{N-1}	…	y_3	y_2	y_1	y_0
0	…	0	0	0	…	0	0	0	1
0	…	0	1	0	…	0	0	1	0
0	…	1	0	0	…	0	1	0	0
0	…	1	1	0	…	1	0	0	0
⋮			⋮	⋮			⋮	⋮	⋮
1	…	1	1	1	…	…	0	0	0

Bild 2-34a: Decodierer (Decoder) – Prinzip

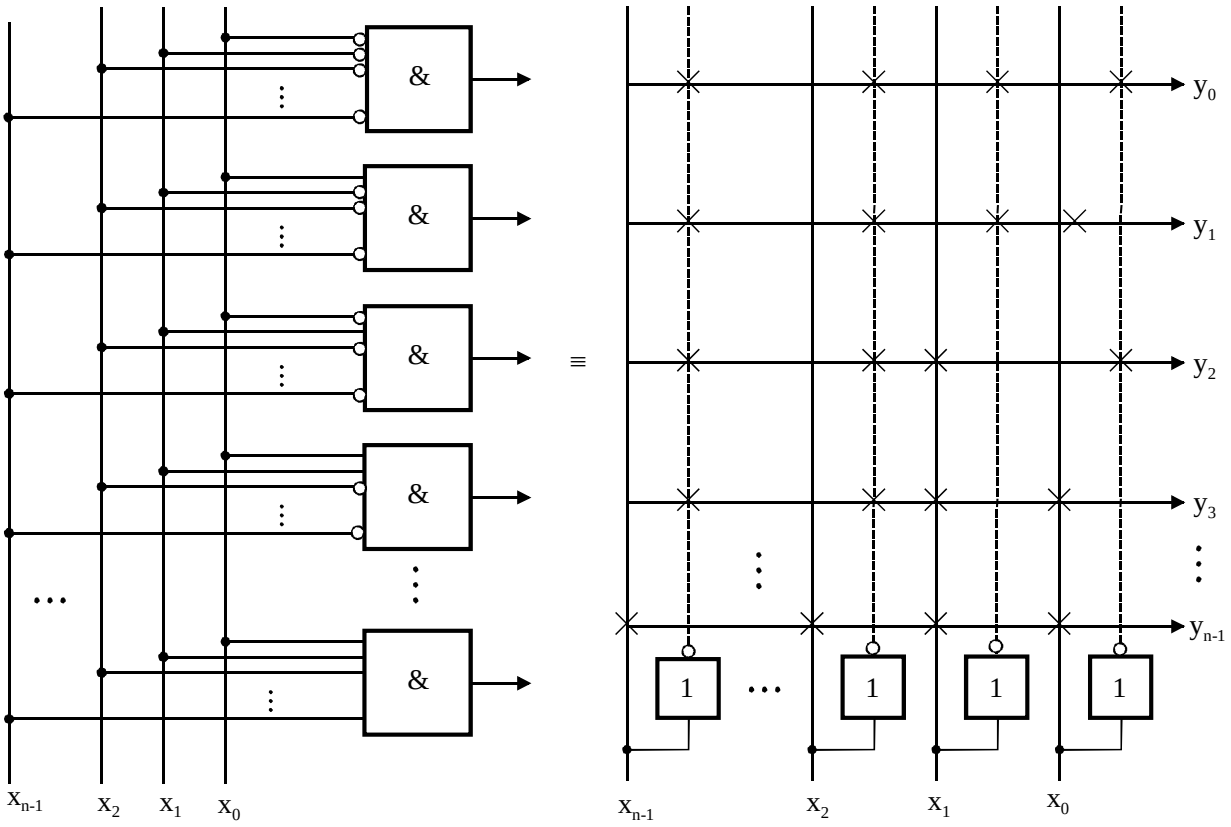


Bild 2-34b: Decodierer (Decoder) - Realisierung

b) Multiplexer

In ähnlicher Form können andere Auswahlfunktionen durch Schaltnetze realisiert werden, z. B. Multiplexer:

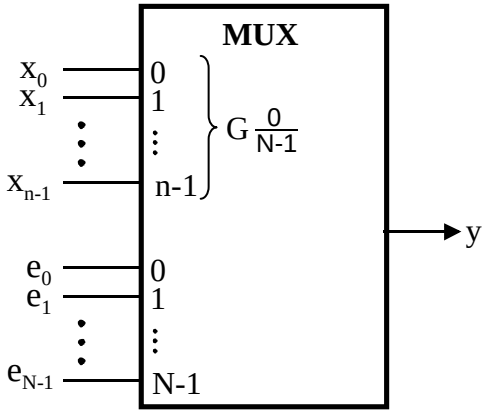


Bild 2-35: Multiplexer

Mit den "Auswahleingängen" $x_{n-1} \ldots x_0$ wird eine von $N = 2^n$ Eingangsbelegungen dem Ausgang Y zugeordnet. Realisierung wie beim Decoder mit einem zusätzlichen Eingang für e_i an jedem UND-Gatter und einer Disjunktion über alle UND-Gatter-Ausgänge.

c) De-Multiplexer

Der De-Multiplexer ordnet die durch e gegebene Eingangsbelegung eindeutig einem der Ausgänge $y_{N-1} \ldots y_0$ zu.

Realisierung wie beim Decoder mit zusätzlichem Eingang e an jedem UND-Gatter.

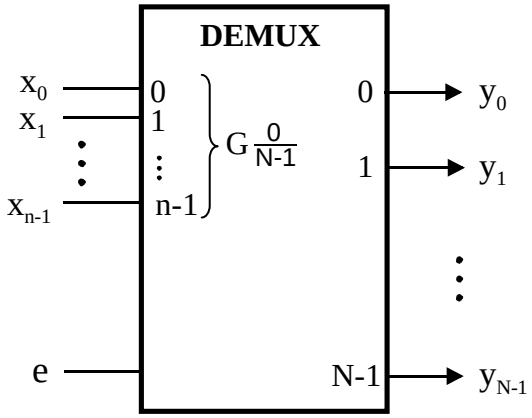


Bild 2-36: De-Multiplexer

Diese Art von Schaltnetzen benötigen bei ein- oder zweistufiger Implementierung Gatter mit sehr vielen Eingängen für größere n. Daher werden diese Funktionen oft mehrstufig als hierarchische Kaskade von Elementen realisiert.

Rechenschaltungen

a) Vergleicher
Die Wahrheitstabelle eines Binärstellenvergleichers lautet

x_1	x_0	y_2	y_1	y_0
0	0	0	0	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1

$y_0 = 1$ für $x_1 = x_0$
$y_1 = 1$ für $x_1 > x_0$
$y_2 = 1$ für $x_1 < x_0$

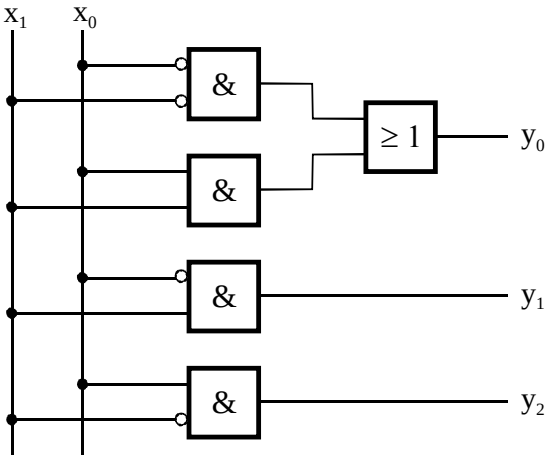


Bild 2-37: Realisierung eines Bitstellenvergleichers

Der Vergleich mehrstelliger Binärzahlen wird überlicherweise durch Subtraktion (s. Subtrahierer) umgesetzt.

b) Addierer
In Kapitel 2.1 (s. Bild 2-5) wurde bereits die Realisierung der Funktion "Volladdierer" mit Addition zweier Binärstellen und Eingangsübertrag sowie Additionsergebnis und Übertrag als Ausgängen als Beispiel vorgestellt.

Um einen mehrstelligen Addierer aufzubauen, kann man im Prinzip mehrere solcher Volladdierer kaskadieren und für die unterste Stelle den Eingangsübertrag auf "0" festlegen.

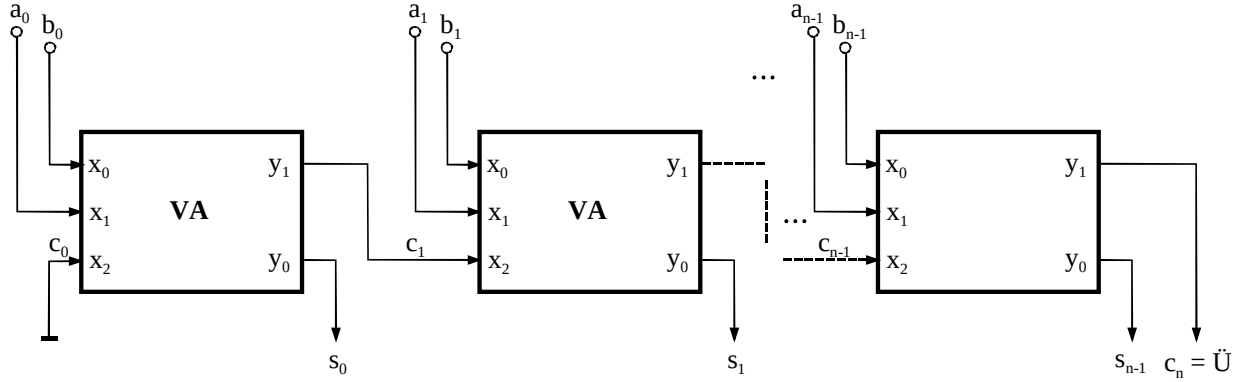


Bild 2-38: Addierer für zwei n-stellige Binärzahlen c = a + b mit Volladdierern

Der Halbaddierer realisiert die binäre Addition ohne Eingangsübertrag aber mit Ausgangsübertrag. Durch Kaskadierung von zwei Halbaddierern lässt sich wiederum der Volladdierer aufbauen.

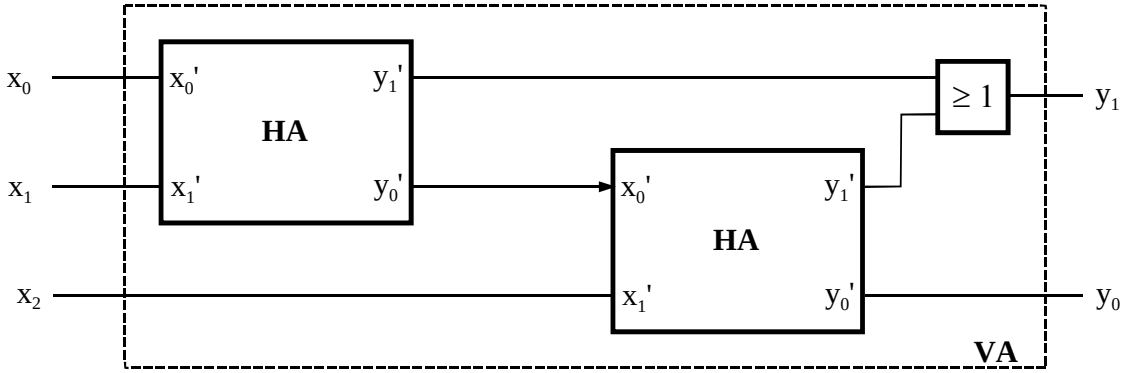


Bild 2-39: Realisierung eines Volladdierers mit zwei Halbaddierern

Der Volladdierer (DNF, minimiert) benötigt 26 Gattereingänge während der Volladdierer nach Bild 2-38 mit 22 Gattereingängen auskommt. Allerdings weist dieser eine höhere Stufenanzahl auf.

c) Subtrahierer
Subtrahieren lassen sich im Prinzip analog zum Addierer aufbauen.

Die Wahrheitstabelle des Vollsubtrahierers ist

x_2	x_1	x_0	y_1	y_0
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$x_2 = b_i$
x_1, x_0 = Eingänge
y_0 = Differenz
$y_1 = b_{i+1}$ (Borrow)

Im Vergleich zum Volladdierer muss nur die Logik für den Ausgangsübertrag (y_1) geändert werden. Das lässt sich im Prinzip auch durch einen weiteren Eingang (Umschalter Addition / Subtraktion) realisieren. Allerdings ist eine spezielle Schaltung für die Subtraktion in den meisten Fällen nicht notwendig, da die Behandlung negativer Zahlen durch das 2er-Komplement sehr einfach möglich ist (Komplementbildung und Addition).

d) Multiplikation
Die Wahrheitstabelle der einstelligen binären Multiplikation ist

x_1	x_0	y
0	0	0
0	1	0
1	0	0
1	1	1

entspricht also der logischen UND-Verknüpfung. Mehrstellige Binärzahlen lassen sich entsprechend dem Verfahren beim schriftlichen Multiplizieren im Dezimalsystem durch stellenweise

Multiplikation, Stellenverschiebung und Addition realisieren. Diese Umsetzung benötigt also Speicher für Zwischenergebnisse und wird somit erst im nächsten Kapitel bei den Schaltwerken behandelt. Das Gleiche gilt für die Division.

Programmierbare Grundstrukturen

Die Basisrealisierung der DNF in 2-stufiger AND/OR/NOT Logik mit einem Netz aus AND/NOT Verknüpfungen in der ersten Stufe und OR-Verknüpfungen in der zweiten Stufe lassen sich wie in Bild 2-40 als "generischer" Aufbau von Schaltnetzen darstellen.

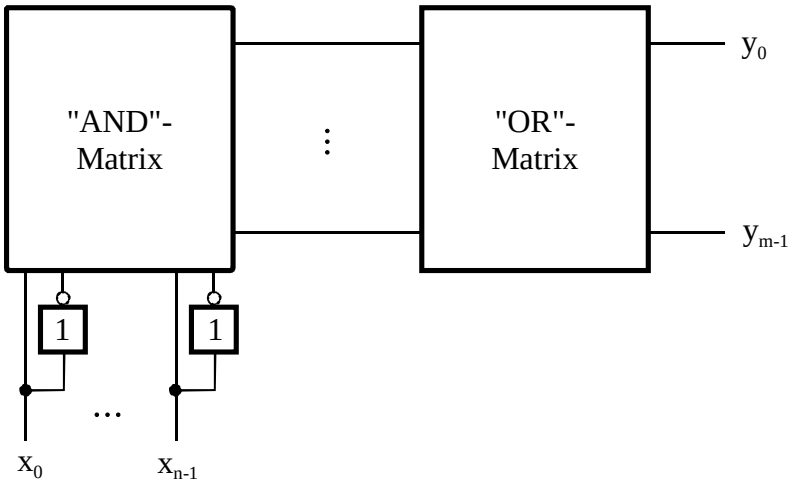


Bild 2-40: 2-stufige Basisrealisierung der DNF

Je nachdem welche der Elemente programmierbar ausgeführt sind ergeben sich drei Typen von Standardrealisierungen:

a) ROM-Realisierung
Bild 2-41a zeigt eine Realisierung mit fester UND-Matrix (vollständig ausgeführt ≙ Decodierer) in Form von verdrahteter Logik. Hier wird durch den Decodierer jeweils eine Spalte der OR-Matrix durch Anlegen einer Eingangskombination $x_{n-1} \ldots x_0$ ausgewählt. Die Zeilen der programmierbaren OR-Matrix entsprechen den jeweiligen Schaltfunktionen $y_{m-1} \ldots y_0$. Das ROM realisiert also die vollständige Wahrheitstabelle.

b) PLA-Realisierung (Bild 2-41b)
Sind beide Matrizen programmierbar, so spricht man von einem PLA (Programmable Logic Array). Dies entspricht den vorher betrachteten DNF-Realisierungen, bei denen nur diejenigen Konjunktionen der Eingänge realisiert und disjunktiv verknüpft werden, die eine "1" in der Schaltfunktion liefern sollen.

c) PAL-Realisierung (Bild 2-41c)
Ist nur die AND-Matrix programmierbar, so spricht man von einem PAL (Programmable Array Logic). Bei vielen Schaltungen ist die Zahl der wirklich benötigten Disjunktionen pro Ausgangsfunktion begrenzt. Deshalb stellt das PAL eine "vorkonfektionierte" OR-Matrix zur Verfügung.

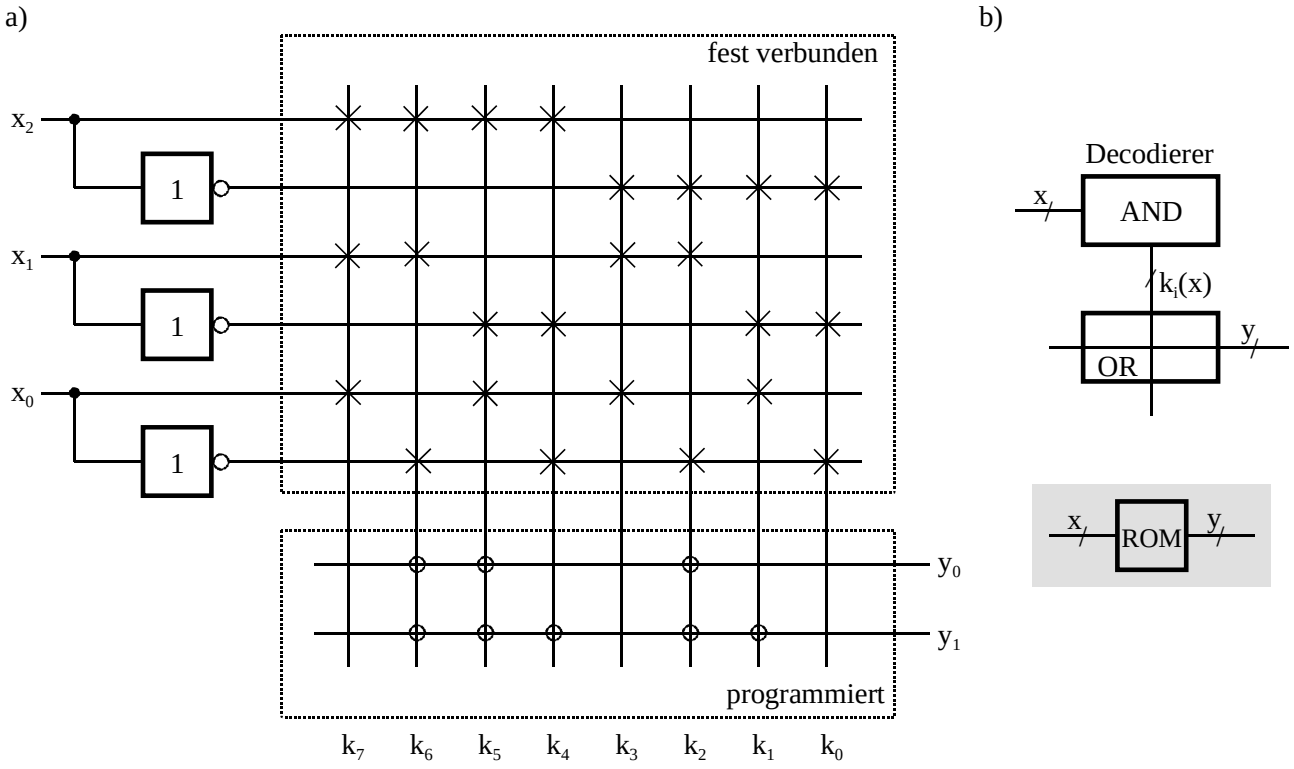


Bild 2-41a: Abstrakte ROM-Programmierung

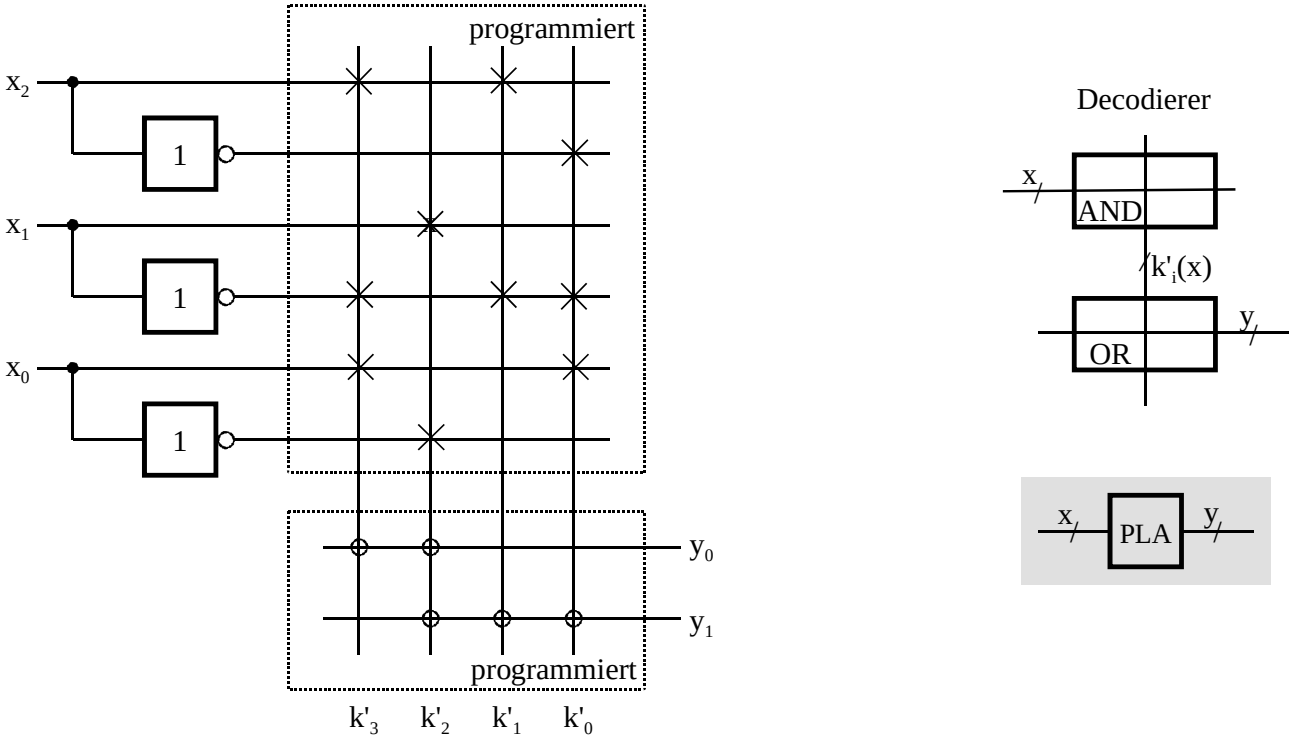


Bild 2-41b: Abstrakte PLA-Struktur

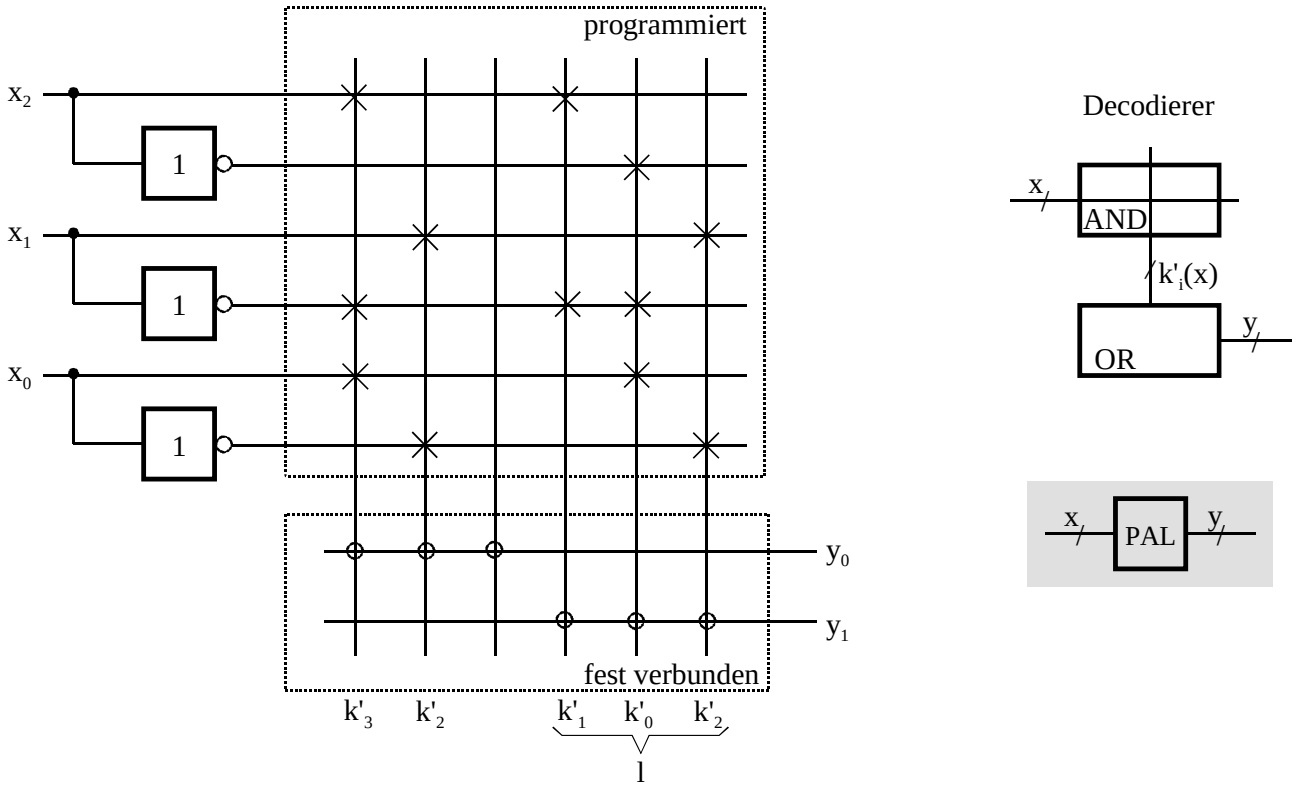


Bild 2-41c: Abstrakte PAL-Struktur

2.6 Praktische Betrachtungen zum Schaltungsentwurf

2.6.1 Implementierungsaspekte

Durch die bisherigen Ausführungen könnte der Eindruck entstanden sein, dass es beim Entwurf eines digitalen Schaltnetzes im wesentlichen damit getan ist, einige wenige Boolesche Gleichungen zu vereinfachen, diese dann in einen Schaltplan umzusetzen und durch einfaches Aneinanderreihen von logischen Gattern zu realisieren.

In der Praxis dagegen ist der Entwickler einer ganzen Reihe von zusätzlichen Zwängen ausgeliefert, die sich daraus ergeben, dass die logischen Elemente eben nicht allein durch Terme der Booleschen Algebra definiert sind, sondern als elektronische Bausteine über eine ganze Reihe zusätzlicher Charakteristika verfügen. Außerdem sucht man nicht nach irgendeiner Lösung, sondern ist bestrebt, eine der Aufgabenstellung angepasste und wirtschaftliche Lösung zu erreichen. Es soll deshalb im folgenden kurz auf einige, aus dem elektronischen Aufbau von Gattern resultierende Kenndaten und auf wirtschaftliche Überlegungen eingegangen werden.

Kennwerte von Schaltkreisen
Ein Blick auf die von den Herstellern von logischen Bauelementen herausgegebenen Datenblätter ergibt selbst für einen sehr einfachen Schaltkreis einige Seiten an Information.

Der Temperaturbereich gibt dem Entwickler an, wie kalt oder warm seine Schaltung werden darf, ohne dass dadurch die anderen Kenndaten des Bauelements unzulässig beeinflusst werden, d.h. ihrerseits die Bereichsgrenzen einhalten. Für den alltäglichen Einsatz, d.h. im Bereich von etwa 25° C, ist diese Werteangabe von untergeordneter Bedeutung; elektronische Komponenten sollen jedoch auch unter extremen klimatischen Bedingungen funktionieren.

Die verschiedenen Spannungswerte definieren die extremen Bedingungen für die Ein- bzw. Ausgänge (worst case). V_{0L} z.B. gibt die maximale Ausgangsspannung für den Fall an, dass der Ausgang auf log. "0" liegt. Bei höheren Spannungen muss man davon ausgehen, dass der Baustein defekt ist.

Sehr wichtig sind die Angaben über die verschiedenen Ströme, die zwischen einzelnen Gattern fließen können. In einem engen Zusammenhang mit diesen Strömen stehen die Angaben des FAN-IN und FAN-OUT.

FAN-OUT (FO, Ausfächerungsfaktor, Ausgangslastfaktor) ist die normierte Ausgangsbelastbarkeit. Die Normierung bezieht sich auf den Strom, den ein Eingang einer einfachen Schaltung (NAND, NOR, ...) derselben Schaltungsfamilie (z.B. ECL, TTL, CMOS, ...) benötigt, um eine "l" oder "0" richtig lesen zu können. D.h., ein FO von 10 bedeutet, dass dieser Ausgang mit bis zu maximal 10 Eingängen belastet werden darf, um auch bei ungünstigsten Voraussetzungen einwandfreies Funktionieren zu gewährleisten.

Gatter, deren Eingänge eine größere Last darstellen (XOR, komplexere Schaltungen), werden mit einem "FAN-IN" > 1 (FI, Eingangslastfaktor) gekennzeichnet. Das FAN-IN gibt an, um welchen Faktor die Belastung größer ist als bei einfachen Schaltungen. Daraus ergibt sich, dass an jedem Punkt einer Schaltung die Summe aller FI kleiner gleich dem FO des damit verbundenen Ausgangs sein muss (Bild 2-42).

Da verschiedene Schaltkreisfamilien z.T. sehr verschiedene Nennströme haben, kann auch ein einfacher Eingang für einen Ausgang einer anderen Technik ein FI von > 1, aber auch < 1 haben.

So kann z.B. bei typischen Normalausgängen und -eingängen (FO = 10, FI = 1) ein "Low-Power-Schottky-TTL" (LSTTL)-Gatter nur zwei "Schottky-TTL" (STTL)-Eingänge treiben. Umgekehrt können "starke" Ausgänge viele "schwache" Eingänge treiben (z.B. ein STTL-Ausgang 50 LSTTL-Eingänge).

Weitere wichtige Kennwerte von logischen Bauelementen sind die verschiedenen Verzögerungszeiten, die, wie wir schon an früherer Stelle gehört haben, nicht vernachlässigt werden dürfen. Bild 2-43 zeigt die Verhältnisse bei einem einfachen Inverter. Ein idealer Impuls am Eingang des Gatters wird mit deutlicher Verzögerung und Verschleifung seiner Flanken am Ausgang weitergegeben. Die Zeit t_{HL} ist die Verzögerungszeit zwischen der steigenden Flanke des Eingangsimpulses und dem Punkt, an dem der Gatterausgang den Wert V_{OL} erreicht hat. Umgedreht ist t_{LH} die Zeit zwischen der fallenden Flanke des Eingangs und dem Zeitpunkt, an dem der Ausgang den Wert V_{OH} erreicht hat.

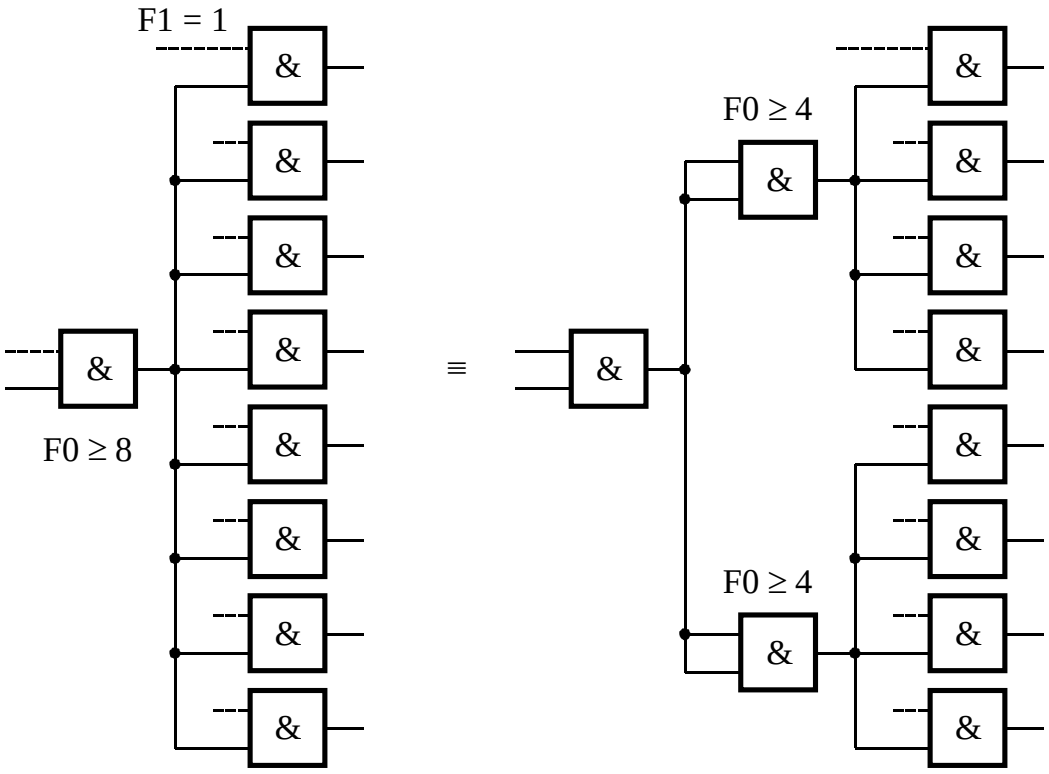


Bild 2-42: Beispiel einer Schaltnetzrealisierung mit Begrenzung auf FAN-Out = 4

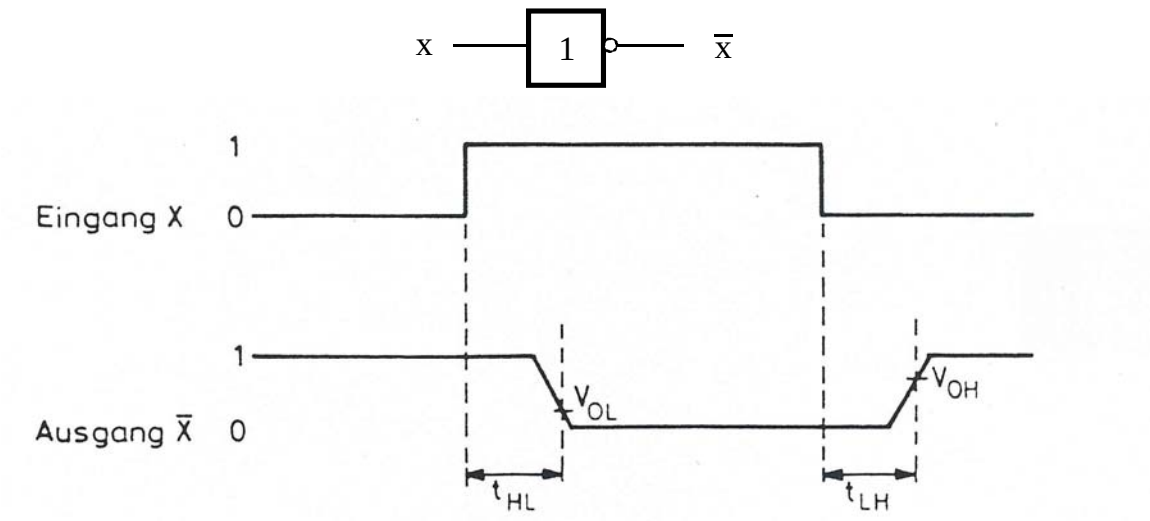


Bild 2-43: Die Verzögerungszeiten bei einem Inverter (Negator)

Wirtschaftliche Überlegungen

Wirtschaftliche Überlegungen bei der Realisierung von logischen Schaltungen hängen eng mit der vorgesehenen Anwendung zusammen. Während z.B. bei Schaltungen für Satellitenmissionen Gewicht, Zuverlässigkeit und Leistungsverbrauch entscheidend sind, wird eine Schaltung für eine Waschmaschine allein nach den Kosten beurteilt.

Schaltnetze werden heute üblicherweise nicht mehr durch diskrete Schaltkreis sondern in programmierbarer Logik wie ROM, PAL, PLA-Realisierung, Kunden-programmierbaren Gate Array oder sog. FPGAs realisiert.

In den genannten Anwendungsfällen hilft der Einsatz integrierter Lösungen, die unterschiedlichen Anforderungen optimal zu realisieren (hohe Integration höhere Zuverlässigkeit wegen geringerer Bauteilanzahl, geringerer Leistungsverbrauch, geringere Kosten).

2.6.2 Dynamische Effekte in kombinatorischen Schaltungen

Dynamische Effekte, sog. Hazards entstehen durch

a) nicht zeitgleiches Ändern von Eingangssignalen

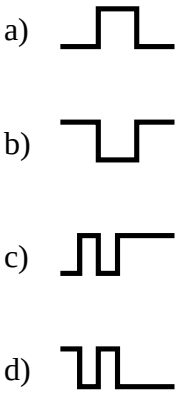
b) Laufzeitunterschiede im Schaltnetz für Signale die unterschiedliche Anzahlen von Gattern durchlaufen.

Entsprechend unterscheidet man

a) Funktionshazards, die durch Belegungswechsel von zwei oder mehreren Eingangsvariablen möglich sind.

b) Strukturhazards, die bei Schaltungen mit mehreren Schaltstufen möglich sind.

Beide Arten von Hazards können im Prinzip vier verschiedene Arten von kurzzeitigen Änderungen im Ausgangssignal hervorrufen, je nach Ausgangs- und Endzustand des Signals.



Funktionshazards

In Bild 2-44 sind die Möglichkeiten für statische Funktionshazards an einem Beispiel im Karnaugh-Veith-Diagramm dargestellt.

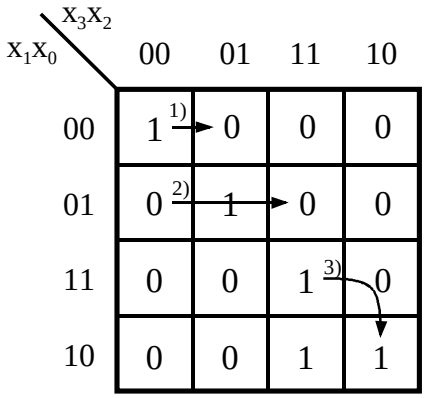


Bild 2-44: Funktionshazards im KV-Diagramm

Belegungswechsel

1) von x^0 (0,0,0,0) zu x^4 (0, 1, 0, 0) $\rightarrow$ kein Hazard möglich.

2) von $x^1 \rightarrow x^5 \rightarrow x^{13}$ statischer 0-Hazard

3) von $x^{15} \rightarrow x^{11} \rightarrow x^{10}$ statischer 1-Hazard

Die Reihenfolge der Belegungswechsel ist hier also dafür entscheidend, ob ein Hazard auftritt oder nicht. In ähnlicher Weise gilt dies für dynamische 0 - 1 und 1 - 0 Hazards.

Vermeidung von Funktionshazards ist möglich, wenn

a) nur Belegungswechsel von genau einer Eingangsvariablen möglich sind → lässt sich für allgemeine Schaltnetzrealisierungen nicht garantieren.

b) Die Belegungswechsel am Eingang bzw. die Gültigkeit der Ausgangsvariablen zeitlich mit Hilfe eines Taktes synchronisiert werden.

Die Synchronisation von Signalen wird im folgenden Kapitel weiter behandelt.

Strukturhazards

Statische Strukturhazards können in zwei- oder mehrstufigen Strukturen auftreten, wenn ein Eingangssignal auf mehrere Logikblöcke mit unterschiedlichen Laufzeiten geht, deren Ausgänge dann wiederum logisch verknüpft werden.

Das folgende Beispiel verdeutlicht dies:

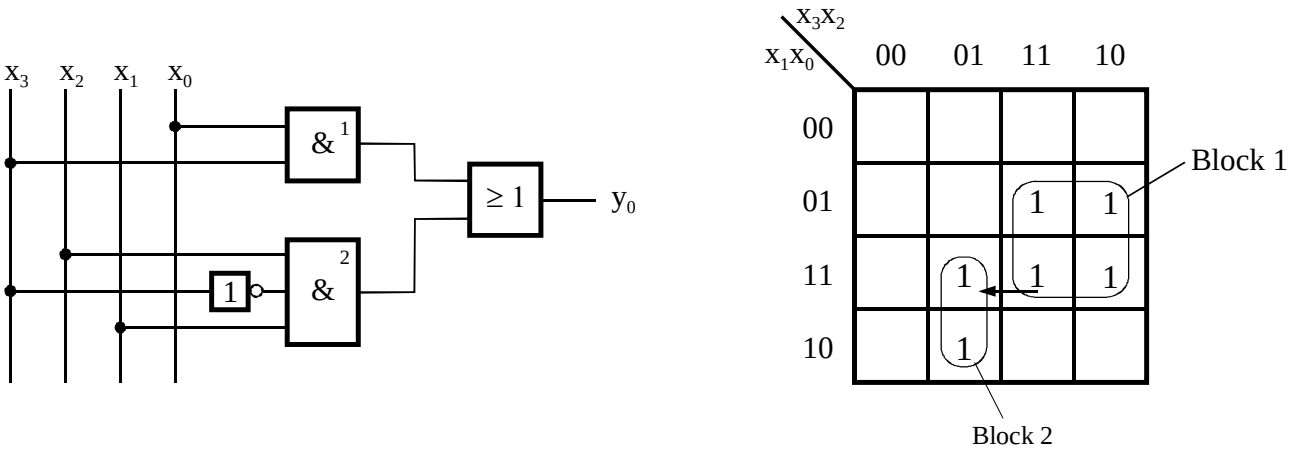


Bild 2-45: Statische Strukturhazards im KV-Diagramm

Das Signal x_3 liegt wegen des Inverters mit unterschiedlichen Laufzeiten an den Blöcken 1 und 2.

Beim Wechsel von x^{15} zu x^7 wird zunächst Block 1 auf "0" gehen bevor Block 2 auf "1" geht, weil das Signal $\overline{x}_3$ noch durch den Inverter verzögert wird.

Dadurch sind beide UND-Gatter kurzzeitig auf "0" und damit auch der Ausgang y.

Kritisch sind also Belegungswechsel zwischen Logikblöcken, die im Karnaugh-Veith-Diagramm durch Schleifen dargestellt sind.

Durch Hinzufügen eines redundanten Terms kann ein solcher StrukturHazard vermieden werden, wie Bild 2-46 zeigt.

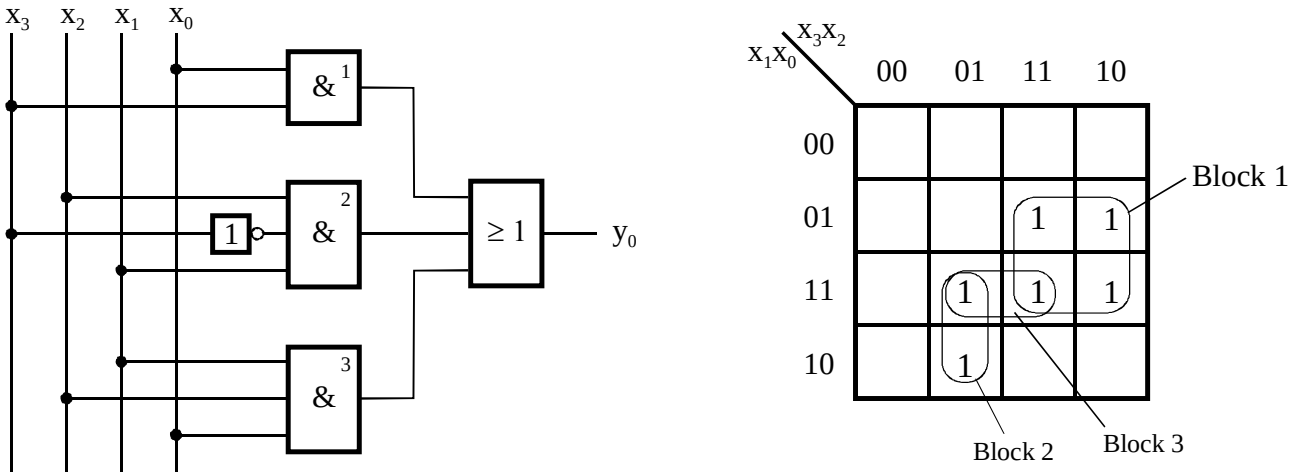


Bild 2-46: Vermeidung von Strukturhazards durch Redundanz

Hier hält der Block 3 den Ausgang auf "1", wenn zwischen x^{15} und x^{7} gewechselt wird.

Dynamische Strukturhazards entstehen in mehrstufigen Schaltungen durch Überlagerung von Strukturhazards in vorherigen Stufen.

Vermeidung von Strukturhazards:

a) durch redundante Terme (Analyse notwendig)

b) Taktsynchronisation (siehe sequentielle Schaltungen)