2 Schaltalgebra

Sonderfall der endlichen Booleschen Algebra ($B, \wedge, \vee, \bar{}, 0, 1$) mit nur zwei Elementen in B:

- B: Trägermenge = {0,1}
- $\wedge$ zweistelliger Operator der Kunjunktion (UND)
- $\vee$ zweistelliger Operator der Disjunktion (ODER)
- $\bar{}$ einstelliger Operator Negation (NOT)
- 0: neutrales Element der Disjunktion
- 1: neutrales Element der Konjunktion

technische Bedeutung:
Binäre Variable (Schaltvariable) können genau zwei Werte annehmen:
Wahrheitswert: falsch/wahr,
Schalter: aus/ein,
Spannungen bzw. Pegelbereiche: Low/High,
duale Ziffern: 0/1

Sie können mit den Grundverknüpfungen $\wedge$, $\vee$ und $\bar{}$ verknüpft werden (Schaltfunktion)
Es gelten die Gesetze der Booleschen Algebra.

Anwendung sowohl für logische als auch für arithmetische Verknüpfungen.

logische Verknüpfungen →	technische Realisierung von Gesetzen der Logik (Aussagenlogik, boolesche Algebra, Schaltalgebra)
arithmetische Verknüpfungen →	technische Realisierung von Rechenregeln in Zahlensystemen, (binär codierte Zahlen, mathematische Verknüpfungen)

2.1 logische Verknüpfungen

logische Elementarfunktionen:

			HDL (Hardware Description Language)
Negation	-		NOT
UND	$\wedge$	auch: ·, &	AND
ODER	$\vee$	auch: +, |, #	OR

Postulate der Schaltalgebra:

Eindeutigkeit:
- x = 0 dann ist x ≠ 1 — X <= '1';
- x = 1 wenn x ≠ 0 — X <= '0';
- $\bar{0} = 1$ — $\bar{1} = 0$
- $\bar{\bar{x}} = x$

Dualität: Vertauschen von $\wedge \Leftrightarrow \vee$ sowie $0 \Leftrightarrow 1$ ergibt äquivalente duale Beziehung (Shannonsches Theorem)

	ODER	**UND**
neutr. Element:	$0 \vee 0 = 0$	$1 \wedge 1 = 1$
	$0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1$	$0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0$
Variable:	$x \vee 0 = x$	$x \wedge 0 = 0$
	$x \vee 1 = 1$	$x \wedge 1 = x$
	$x \vee x \vee x \ldots = x$	$x \wedge x \wedge x \ldots = x$
	$x \vee \bar{x} = 1$	$x \wedge \bar{x} = 0$

Rechengesetze:

	ODER $\vee$, +, | ,#	**UND $\wedge$, · , &**
Kommutativgesetz	$x_1 \vee x_2 = x_2 \vee x_1$	$x_1 \wedge x_2 = x_2 \wedge x_1$
	$x_1 + x_2 = x_2 + x_1$	$x_1 \cdot x_2 = x_2 \cdot x_1$

Assoziativgesetz

$x_1 \vee (x_2 \vee x_3) = (x_1 \vee x_2) \vee x_3$ $\quad$ $x_1 \wedge (x_2 \wedge x_3) = (x_1 \wedge x_2) \wedge x_3$

$x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3$ $\quad$ $x_1 \cdot (x_2 \cdot x_3) = (x_1 \cdot x_2) \cdot x_3$

Distributivgesetz

$(x_1 \vee x_2) \wedge (x_1 \vee x_3) = x_1 \vee (x_2 \wedge x_3)$ $\quad$ $(x_1 \wedge x_2) \vee (x_1 \wedge x_3) = x_1 \wedge (x_2 \vee x_3)$

$(x_1 + x_2) \cdot (x_1 + x_3) = x_1 + (x_2 \cdot x_3)$ $\quad$ $(x_1 \cdot x_2) + (x_1 \cdot x_3) = x_1 \cdot (x_2 + x_3)$

(gilt nicht in der konv. Algebra!)

Theorem von De´Morgan

$\overline{x_1 \vee x_2} = \bar{x}_1 \wedge \bar{x}_2$ $\quad$ $\overline{x_1 \wedge x_2} = \bar{x}_1 \vee \bar{x}_2$

$\overline{x_1 + x_2} = \bar{x}_1 \cdot \bar{x}_2$ $\quad$ $\overline{x_1 \cdot x_2} = \bar{x}_1 + \bar{x}_2$

Kürzungsregeln:

$x_1 \vee (x_1 \wedge x_2) = x_1$ $\quad$ $x_1 \wedge (x_1 \vee x_2) = x_1$

$x_1 + (x_1 \cdot x_2) = x_1 \cdot (1 + x_2) = x_1$ $\quad$ $x_1 \cdot (x_1 + x_2) = x_1 + (0 \cdot x_2) = x_1$

$(x_1 \wedge x_2) \vee (x_1 \wedge \bar{x}_2) = x_1$ $\quad$ $(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) = x_1$

$(x_1 \cdot x_2) + (x_1 \cdot \bar{x}_2) = x_1 \cdot (x_2 + \bar{x}_2) = x_1$ $\quad$ $(x_1 + x_2) \cdot (x_1 + \bar{x}_2) = x_1 + (x_2 \cdot \bar{x}_2) = x_1$

$x_1 \vee (\bar{x}_1 \wedge x_2) = x_1 \vee x_2$ $\quad$ $x_1 \wedge (\bar{x}_1 \vee x_2) = x_1 \wedge x_2$

$x_1 + (\bar{x}_1 \cdot x_2) = x_1 + x_2$ $\quad$ $x_1 \cdot (\bar{x}_1 + x_2) = x_1 \cdot x_2$

Vorrangregeln:

1) Negation
2) UND (Konjunktion)
3) ODER (Disjunktion)

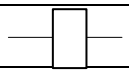
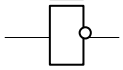
In HDL Beschreibungen hat NOT Vorrang, AND und OR sind jedoch gleichrangig!

binäre Verknüpfungen - Schaltfunktionen

Analog zur gewöhnlichen Algebra läßt sich eine Abbildung vom Typ $f : B^n \rightarrow B$ ($n \in N$) definieren.
Eine Abbildung $f(x_1, x_2, \ldots x_n) = f(x) = y$ heißt n-stellige Schaltfunktion (SF)
Der Definitionsbereich $B^n=\{0,1\}^n$ der Schaltfunktion f umfaßt 2^n Elemente.
Der Wertebereich wird durch die Menge B={0,1} gebildet und enthält 2 Elemente.
Die Menge aller n-stelligen Schaltfunktionen entspricht der Menge aller Abbildungen einer 2^n-elementigen Menge in eine 2-elementige Menge.
Hierfür gibt es 2^{2^n} Möglichkeiten, die Anzahl der möglichen Schaltfunktionen.

n = 1: 4 Funktionen
n = 2: 16 Funktionen
n = 3: 256 Funktionen
n = 4: 65536 Funktionen

Bsp: n = 1:

Eingang x_0 / Ausgang	0	1	Gleichung	Symbol	Bezeichnung	HDL (Hardware Description Language)
y_0	0	0	$y_0 = 0$	0	Konstante 0	`Y0 <= '0';`
y_1	0	1	$y_1 = x_0$		Identität	`Y1 <= X0;`
y_2	1	0	$y_2 = \bar{x}_0$		Negation	`Y2 <= NOT X0;`
y_3	1	1	$y_3 = 1$	1	Konstante 1	`Y3 <= '1';`

Technische Umsetzung:

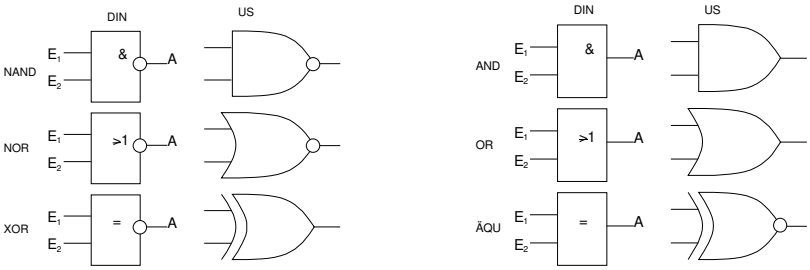
y_0, y_3: konstante Eingangspegel 1, 0
y_1: Verbindung, technisch oft auch Verstärkung (nichtnegierender “Treiber”)
y_2: Die Negationsfunktion ist mit Halbleiterbauelementen (Bipolar- oder Feldeffekttransistoren) sehr gut realisierbar. Grundgatter: NEGATOR

Bsp: n = 2:

Eingang x_0 / x_1 / Ausgang	0 0	0 1	1 0	1 1	Gleichung	Symbol	Bezeichnung	HDL
y_0	0	0	0	0	$y_0 = 0$		Konstante 0	`Y0 <= '0';`
y_1	0	0	0	1	$y_1 = x_0 \wedge x_1$ $y_1 = x_0 \cdot x_1$	&	AND (UND) Konjunktion	`Y1 <= X0 AND X1;`
y_2	0	0	1	0	$y_2 = x_0 \wedge \bar{x}_1$ $y_2 = x_0 \cdot \bar{x}_1$	&	Inhibition II	`Y2 <= X0 AND NOT X1;`
y_3	0	0	1	1	$y_3 = x_0$	x_0	Identität x_0	`Y3 <= X0;`
y_4	0	1	0	0	$y_4 = \bar{x}_0 \wedge x_1$ $y_4 = \bar{x}_0 \cdot x_1$	&	Inhibition I	`Y4 <= NOT X0 AND X1;`
y_5	0	1	0	1	$y_5 = x_1$	x_1	Identität x_1	`Y5 <= X1;`
y_6	0	1	1	0	$y_6=(\bar{x}_0 \wedge x_1) \vee (x_0 \wedge \bar{x}_1)$ $y_6 = x_0 \oplus x_1$	=1	XOR Antivalenz	`Y6 <= X0 XOR X1;`
y_7	0	1	1	1	$y_7 = x_0 \vee x_1$ $y_7 = x_0 + x_1$	≥1	OR (ODER) Disjunktion	`Y7 <= X0 OR X1;`
y_8	1	0	0	0	$y_8 = \overline{x_0 \vee x_1}$ $y_8 = \overline{x_0 + x_1}$	≥1	NOR	`Y8 <= NOT (X0 OR X1);`
y_9	1	0	0	1	$y_9=(\bar{x}_0 \wedge \bar{x}_1) \vee (x_0 \wedge x_1)$ $y_9 = x_0 \leftrightarrow x_1$	=	EQU (ÄQU) Äquivalenz	`Y9 <= NOT (X0 XOR X1);`
y_{10}	1	0	1	0	$y_{10} = \bar{x}_1$	x_1	NEG Negation x_1	`Y10 <= NOT X1;`
y_{11}	1	0	1	1	$y_{11} = x_0 \vee \bar{x}_1$ $y_{11} = x_0 + \bar{x}_1$	≥1	Implikation II	`Y11 <= X0 OR NOT X1;`
y_{12}	1	1	0	0	$y_{12} = \bar{x}_0$	x_0	NEG Negation x_0	`Y12 <= NOT X0;`
y_{13}	1	1	0	1	$y_{13} = \bar{x}_0 \vee x_1$ $y_{13} = \bar{x}_0 + x_1$	≥1	Implikation I	`Y13 <= NOT X0 OR X1;`
y_{14}	1	1	1	0	$y_{14} = \overline{x_0 \wedge x_1}$ $y_{14} = \overline{x_0 \cdot x_1}$	&	NAND	`Y14 <= NOT (X0 AND X1);`
y_{15}	1	1	1	1	$y_{15} = 1$		Konstante 1	`Y15 <= '1';`

Häufig realisierte Grundfunktionen werden durch 2-Eingangs-Grundgatter realisiert

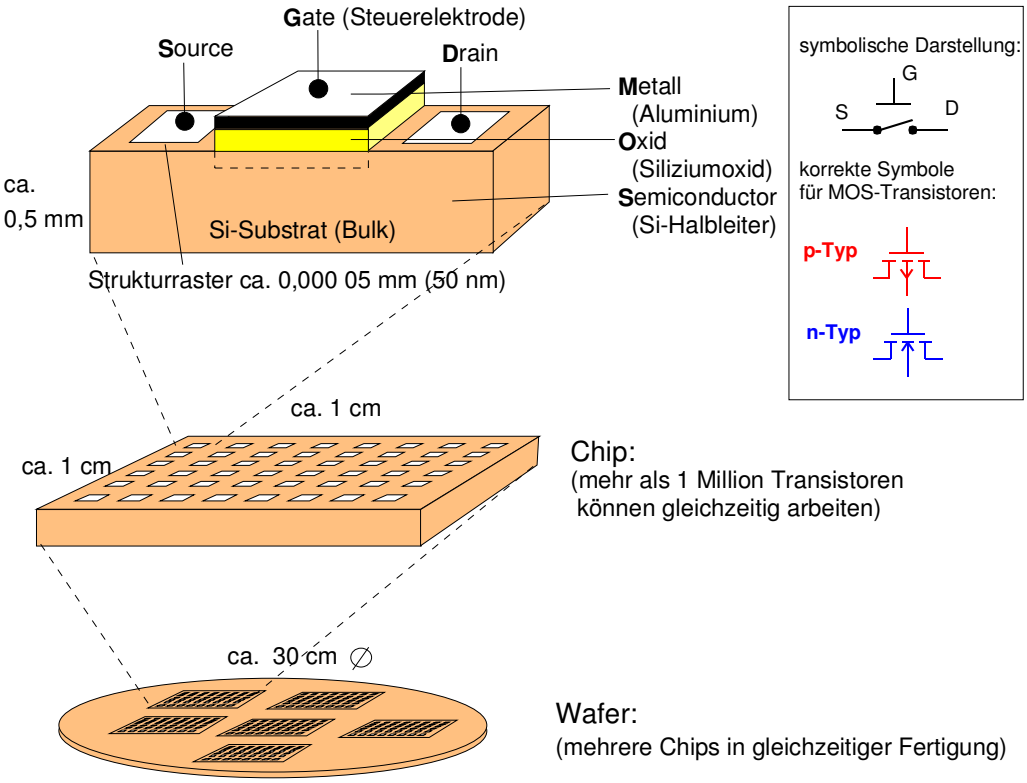
AND	$y_1 = x_0 \wedge x_1$	`Y1 <= X0 AND X1;`
XOR	$y_6 = x_0 \oplus x_1$	`Y6 <= X0 XOR X1;`
EQU	$y_6 = x_0 \leftrightarrow x_1$	`Y6 <= NOT (X0 XOR X1);`
OR	$y_7 = x_0 \vee x_1$	`Y7 <= X0 OR X1;`
NOR	$y_8 = x_0 \vee x_1$	`Y8 <= NOT (X0 OR X1);`
NAND	$y_{14}= x_0 \wedge x_1$	`Y14 <= NOT (X0 AND X1);`



Gegenüberstellung häufig verwendeter Schaltsymbole der DIN- und US-Norm

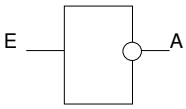
Realisierung der Grundfunktionen

NOT, NAND, NOR usw. werden zumeist in CMOS (Complementary Metal Oxide Semiconductor) Technologie gefertigt. MOS Transistoren sind einfach aufgebaut und automatisierbar herstellbar:

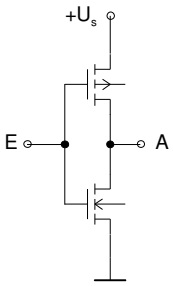


NOT (Negator): enthält 2 komplementäre (also p-und n-Typ) Transistoren: CMOS

Symbol



Innenschaltung



Tabelle

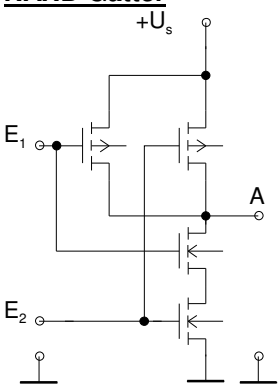
E	A
0	1
1	0

boolesche Algebra,
HDL Beschreibung

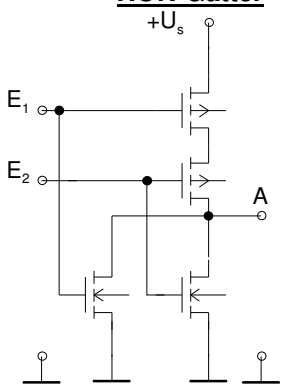
```
A = E

A <= NOT E;      -- HDL
```

NAND Gatter



NOR-Gatter



… enthalten jeweils 4 dieser Transistoren mit geeigneter „Verdrahtung“. Für weitere Details im Gatteraufbau sei auf die einführende Übung oder Literatur verwiesen, z.B. Becker, Drechsler, Molitor: Technische Informatik: Eine Einführung, Pearson 2005.

Schaltbelegungstabelle

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

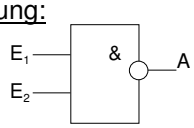
E_1	E_2	A
0	0	1
0	1	0
1	0	0
1	1	0

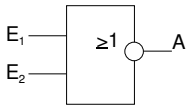
Schaltfunktion:

$A = \overline{E_1 \wedge E_2}$ (NAND)

$A = \overline{E_1 \vee E_2}$ (NOR)

Symboldarstellung:





HDL (Hardware Description Language):

A <= NOT (E1 AND E2); A <= NOT (E1 OR E2);

Darstellung von Schaltfunktionen:

- logische Aussagen
- Schaltbelegungstabelle ST (auch Wahrheitstabelle, Logiktabelle)
- Schaltfunktion SF (auch boolesche Funktion)
- Logikplan LP (auch Schaltplan, schematic)

Schaltungssynthese: ST → SF → LP

Schaltungsanalyse: LP → SF → ST

Schaltbelegungstabelle ST: für n Eingangsvariable und 1 Ausgangsvariable: n+1 Spalten, 2^n Zeilen

Schaltfunktion SF: $y = f(x_0, x_1, \ldots x_{n-1})$ mit konjunktiven und disjunktiven Verknüpfungen. Oft gibt es mehrere äquivalente Schaltfunktionen,

z.B. $y = \overline{x_1} \wedge x_0 \vee x_1 \wedge \overline{x_0} \vee x_2 \wedge x_1$

$y = x_1 \wedge x_0 \vee x_1 \wedge \overline{x_0} \vee x_2 \wedge x_0$

Vollkonjunktion: UND-Verknüpfung aller Variablen (negiert oder nichtnegiert):

z.B. $y_3 = \bar{x}_3 \wedge \bar{x}_2 \wedge x_1 \wedge x_0$

Kanonisch disjunktive Normalform KDN: Alle die Vollkonjunktionen, die den Wert 1 ergeben, werden disjunktiv verknüpft: (Kanonisch: vollständig, d.h. alle Variablen sind enthalten)

$y = y_0 \vee y_1 \vee .... \vee y_{m-1}$ mit $y_i = \wedge (x_0, x_1, \ldots x_{n-1})$; $m=2^n$;

alle $y_i = 1$ sind "Minterme"

Volldisjunktion: ODER-Verknüpfung aller Variablen (negiert oder nichtnegiert):

z.B. $y_6 = \bar{x}_3 \vee x_2 \vee x_1 \vee \bar{x}_0$

Kanonisch konjunktive Normalform KKN: Alle die Volldisjunktionen der negierten Eingangsvariablen, die den Wert 0 ergeben, werden konjunktiv verknüpft:

$y = y_0 \wedge y_1 \wedge .... \wedge y_{m-1}$ mit $y_i = \vee (x_0, x_1, \ldots x_{n-1})$; $m=2^n$

alle $y_i = 0$ sind "Maxterme"

Logikplan LP: grafische Umsetzung der Schaltfunktion SF bzw. der Schaltbelegungstabelle ST.

Die technische Realisierung der KDN oder KKN ist mit NOT, AND und OR – Gattern immer möglich - wenn auch nicht mit minimalem Aufwand.

Beispiel:
Äquivalenz-Funktion $y = x_0 \leftrightarrow x_1$:
1. Beschreibung durch disjunktive Normalform

x_1 \ x_0	0	1
0	1	0
1	0	1

$\vee$ n / $\wedge$	x_0	x_1	y_n	Min-term
0	0	0	1	$\bar{x}_0 \wedge \bar{x}_1$
1	0	1	0	
2	1	0	0	
3	1	1	1	$x_0 \wedge x_1$

Vollkonjunktionen (Minterme):

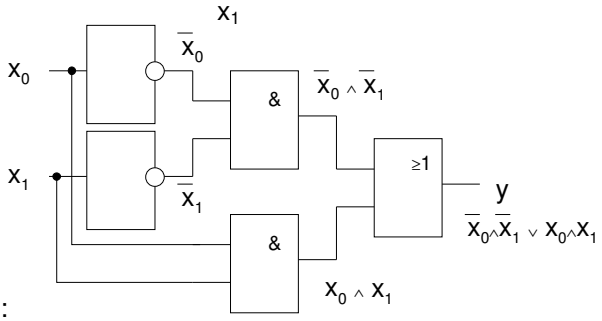
$y_0 = \bar{x}_0 \wedge \bar{x}_1$

x_1 \ x_0	0	1
0	y_0=1	y_0=0
1	y_0=0	y_0=0

$y_3 = x_0 \wedge x_1$

x_1 \ x_0	0	1
0	y_3=0	y_3=0
1	y_3=0	y_3=1

Kanonisch disjunktive Normalform KDN: ODER-Verknüpfung der Minterme ergibt: $y = \bar{x}_0 \wedge \bar{x}_1 \vee x_0 \wedge$



x_1 \ x_0	0	1
0	1	0
1	0	1

Schaltplan:

HDL: `Y <= (NOT X0 AND NOT X1) OR (X0 AND X1);`
oder: `Y <= NOT (X0 XOR X1);`

2. Beschreibung durch konjunktive Normalform

$\wedge$ n / $\vee$	x_0	x_1	$\bar{x}_0$	$\bar{x}_1$	y_n	Max-Term
0	0	0	1	1	1	
1	0	1	1	0	0	$x_0 \vee \bar{x}_1$
2	1	0	0	1	0	$\bar{x}_0 \vee x_1$
3	1	1	0	0	1	

Volldisjunktionen (Maxterme):

$y_1 = x_0 \vee \bar{x}_1$

x_1 \ x_0	0	1
0	y_1=1	y_1=1
1	y_1=0	y_1=1

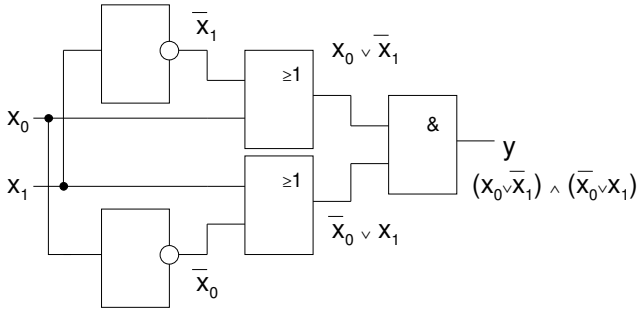
$y_2 = \bar{x}_0 \vee x_1$

x_1 \ x_0	0	1
0	y_2=1	y_2=0
1	y_2=1	y_2=1

Kanonisch konjunktive Normalform: UND-Verknüpfung der Maxterme ergibt $y = (x_0 \vee \bar{x}_1) \wedge (\bar{x}_0 \vee x_1)$

x_1 \ x_0	0	1
0	1	0
1	0	1

Schaltplan:



HDL: `Y <= (X0 OR NOT X1) AND (NOT X0 OR X1);`

2.2 binäre Zahlensysteme

Stellenwertsysteme: Positionssysteme, Stelle der Ziffer ist für ihre Wertigkeit entscheidend.
Basis kann jede ganze Zahl B>2 sein

$Z = c_{n-1} \cdot B^{n-1} + c_{n-2} \cdot B^{n-2} + ... + c_0 \cdot B^0 + c_{-1} \cdot B^{-1} + c_{-2} \cdot B^{-2} + c_{-m} \cdot B^{-m}$

$$Z = \sum_{\nu=-m}^{n-1} c_\nu B^\nu$$ mit B: Basis, c: Koeffizient (Ziffer),

n Vorkommastellen, m Nachkommastellen

Beispiele:

Dezimalsystem: Basis 10, Ziffern 0 .. 9,
$2010 = 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 0 \cdot 10^0$

Dualsystem: Basis 2, Ziffern 0, 1
$10110 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 22_D$

Oktalsystem: Basis 8, Ziffern 0 .. 7,
$26_O = 2 \cdot 8^1 + 6 \cdot 8^0 = 22_D$

Hexadezimalsystem: Basis 16, Ziffern 0 .. 9, A .. F
$16 = 1 \cdot 16^1 + 6 \cdot 16^0 = 22_D$

Die ersten 16 Dualzahlen, ihre Dezimal- und Hexadezimalentsprechung

Dualzahl	Dezimal-zahl	Hexa-dezimal	**Dualzahl**	Dezimal-zahl	Hexa-dezimal
0	0	0	**1000**	8	8
1	1	1	**1001**	9	9
10	2	2	**1010**		A
11	3	3	**1011**		B
100	4	4	**1100**		C
101	5	5	**1101**		D
110	6	6	**1110**		E
111	7	7	**1111**		F

Schlußfolgerungen:

- Dualzahlen lassen sich unmittelbar mit zweiwertigen Elementen (binary digit, bit) darstellen. Pro Stelle wird ein Bit zur Kodierung von 0 oder 1 genutzt.
- Dezimalzahlen sind ebenfalls binär codierbar, Code 0, 1, 10, .. 1001
  BCD-Zahl, **B**inär **C**odierte **D**ezimalzahl,
  Codes 1010 ... 1111 bleiben ohne Zuordnung
- Hexadezimalzahlen sind mit vollständiger Zuordnung 0, 1, 10, ... 1111 binär mit 4 Symbolen (4 bit) pro Ziffer codierbar
- Für große Zahlen ist die Darstellung in Hexadezimalziffern oder die Nutzung von Abkürzungen, die sich auf das dezimale Zahlensystem beziehen, sinnvoll.
  1 K bit= 1024 bit, 1 M bit = 1.048.576 bit ≈ 10^6 bit, 1 G bit ≈ 10^9 bit, 1 T bit ≈ 10^{12} bit
- Jede Hexadezimalziffer wird mit 4 Binärstellen codiert.
- Die Gruppierung von 8 bit ergibt ein Byte. Auch hier gibt es Vielfache, z.B.:
  1 K byte = 1024 byte = 1024 · 8 bit

- Ein Byte (8 bit) kann 2^8 = 256 verschiedene Zeichen aufnehmen,
  Beispiel: 8-Bit-PC-Zeichensatz, Textdarstellung, Code 'd' = 100_D
- Ein Byte kann ebenso 2 Hexadezimalziffern aufnehmen.
  Beispiel: Hexadezimale Darstellung von Texten (character) 'd' = $100_D = 64_H$
- Es bedarf Vereinbarungen, wie eine Information binär codiert ist
  → Kodierungsvorschriften
  Beispiele: Dualcode, BCD-Code, Zweierkomplementcode für Zahlendarstellungen, ASCII, ANSI, IEC-Zeichensätze
- Zur Darstellung negativer Zahlen bieten sich auch binäre Größen (0,1) an. Dazu muß eine zusätzliche Binärstelle (Vorzeichenbit) vereinbart werden oder eine Codierung genutzt werden, die positive und negative Zahlen gleichermaßen einschließt.

weitere Codes zur Darstellung von Dezimalziffern:

8421-Code (bereits als BCD-Code vorgestellt)
2421-Code (Aiken-Code): negationssymmetrisch
Drei-Excess-Code (Stibitz-Code):
negationssymmetrisch, unter Vermeidung von 0000 bzw. 1111, die im Fehlerfall mit höherer Wahrscheinlichkeit auftreten.
Gray-Code: einschrittiger Code, auch zur Codierung von Hexadezimalzahlen geeignet

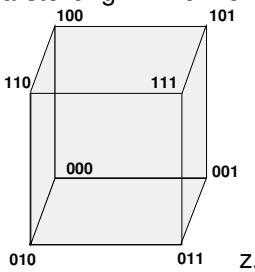
Codierung	8421		8 4 2 1	2 4 2 1	9 8 7 6 5 4 3 2 1 0
Hexadezimal	Dual	Gray	BCD	Aiken	1 aus 10
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0 0 0 0 0 1
1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 0 0 0 0 1 0
2	0 0 1 0	0 0 1 1	0 0 1 0	0 0 1 0	0 0 0 0 0 0 1 0 0
3	0 0 1 1	0 0 1 0	0 0 1 1	0 0 1 1	0 0 0 0 0 1 0 0 0
4	0 1 0 0	0 1 1 0	0 1 0 0	0 1 0 0	0 0 0 0 1 0 0 0 0
5	0 1 0 1	0 1 1 1	0 1 0 1	1 0 1 1	0 0 0 0 1 0 0 0 0
6	0 1 1 0	0 1 0 1	0 1 1 0	1 1 0 0	0 0 0 1 0 0 0 0 0
7	0 1 1 1	0 1 0 0	0 1 1 1	1 1 0 1	0 0 1 0 0 0 0 0 0
8	1 0 0 0	1 1 0 0	1 0 0 0	1 1 1 0	0 1 0 0 0 0 0 0 0
9	1 0 0 1	1 1 0 1	1 0 0 1	1 1 1 1	1 0 0 0 0 0 0 0 0
A	1 0 10	1 1 1 1			
B	1 0 1 1	1 1 1 0			
C	1 1 0 0	1 0 1 0			
D	1 1 0 1	1 0 1 1			
E	1 1 1 0	1 0 0 1			
F	1 1 1 1	1 0 0 0			

weitere Codeeigenschaften:

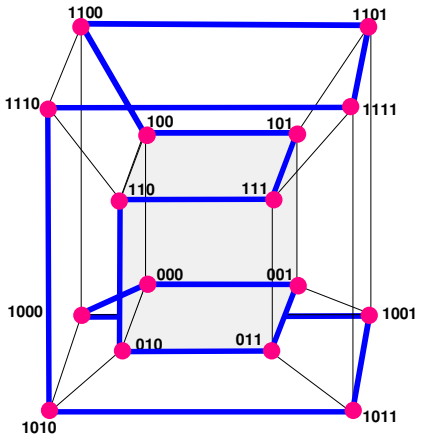
negationssymmetrisch: Ziffern für 6 .. 9 entsprechen den Ziffern von 0 .. 4, wenn 0 und 1 in der Codierung vertauscht werden. Z.B. : Aiken-Code

einschrittig: aufeinanderfolgende Codewärter unterscheiden sich nur in genau einem Bit.
→ keine Fehler beim Zählübergang. Bsp: Gray-Code

Code-Darstellung im n-dimensionalen Koordinatensystem (Boolescher Würfel)



z.B. n=3

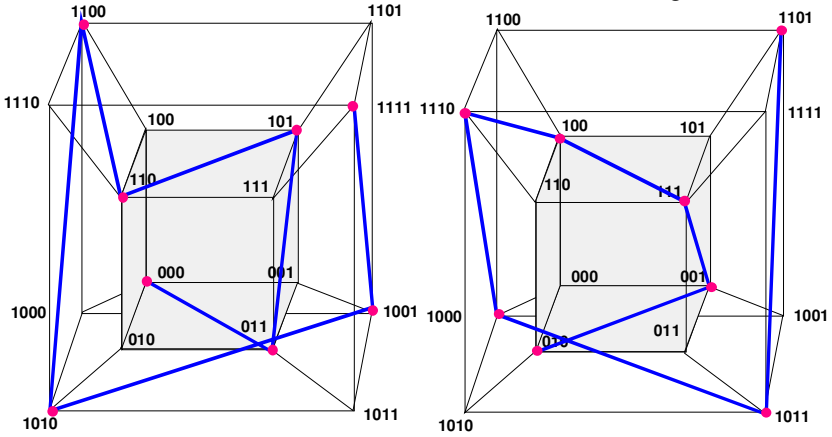


Darstellung des einschrittigen Gray-Codes D im Hypercube (n=4: vierdimensionaler Boolescher Würfel)

Fehlererkennende und fehlerkorrigierende Codes:

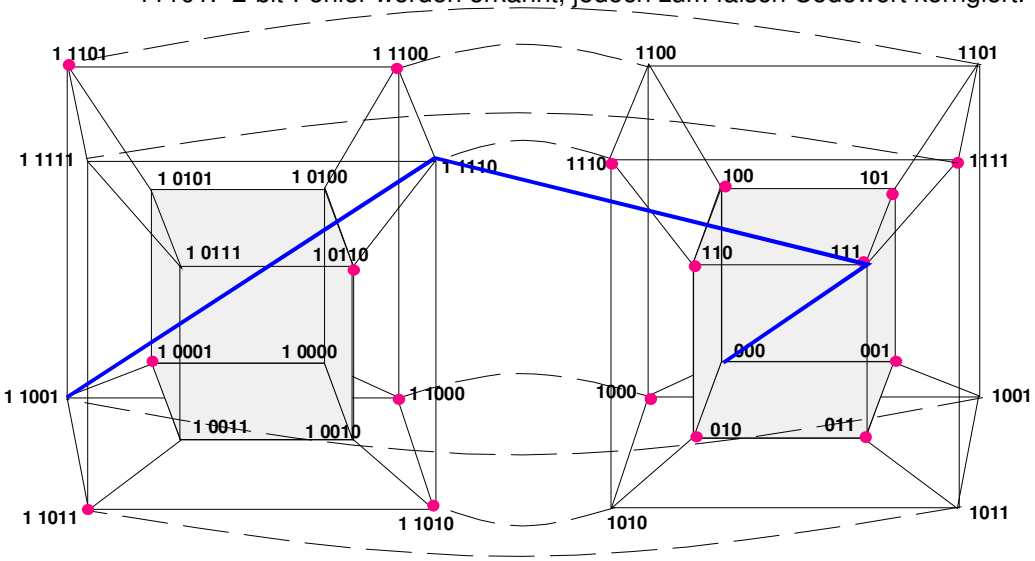
Hamming-Abstand: minimale Anzahl der Veränderungen zwischen zwei Codezeichen. einschrittige Codes besitzen eine Hamming-Distanz von 1.

Beispiel 1: Code mit Hamming-Abstand 2: zwei Codezeichen unterscheiden sich in mindestens 2 Bit, das Auftreten von 1-bit-Fehlern führt zu ungültigen Codezeichen → Fehlererkennung



gerade Anzahl 1 (even Parity) ungerade Anzahl 1 (odd Parity)

Beispiel 2: Code mit Hamming-Distanz 3: zwei beliebige Codezeichen unterscheiden sich in mindestens 3 Bit, das Auftreten von 1-bit-Fehlern (rote Kennzeichnung) führt zu ungültigen Codezeichen, die jedoch eindeutig einem gültigen Code zugeordnet werden können → Fehlerkorrektur, z.B. wird 01111 dem Nachbarcode 00111 zugeordnet, 01110 dagegen 11101. 2-bit-Fehler werden erkannt, jedoch zum falsch Codewort korrigiert.



Für den Aufbau solcher Hamming-Codes werden den m Informationsbits k Korrekturbits zugefügt, so daß die Bitlänge n=m+k beträgt. Mit k Prüfbits können 2^k Fehlerzustände gekennzeichnet werden, die sowohl die Informationsbits m als auch die Prüfbits k betreffen können: $2^k > m+k$
k wächst nur logarithmisch mit n, d.h. bei großen Bitbreiten ist der Aufwand an Korrekturbits relativ gering.

m	k
1	2
4	**3**
11	4
26	5
57	6
120	7
247	8

Codes zur alphanumerische Zifferndarstellung

Bsp. 7-bit-Code (z.B. im ASCII-Code, ISO-7-Bit-Code,)

Beispielcode (Hexadezimal) für „Text 1“: 54 65 78 74 20 31

Hex. $x_6x_5x_4$ $x_3x_2x_1x_0$	2	3	4	5	6	7
0	Space	**0**	**@**	**P**	**`**	**p**
1	**!**	**1**	**A**	**Q**	**a**	**q**
2	**"**	**2**	**B**	**R**	**b**	**r**
3	**#**	**3**	**C**	**S**	**c**	**s**
4	**$**	**4**	**D**	**T**	**d**	**t**
5	**%**	**5**	**E**	**U**	**e**	**u**
6	**&**	**6**	**F**	**V**	**f**	**v**
7	**'**	**7**	**G**	**W**	**g**	**w**
8	**(**	**8**	**H**	**X**	**h**	**x**
9	**)**	**9**	**I**	**Y**	**i**	**y**
A	*	**:**	**J**	**Z**	**j**	**z**
B	**+**	**;**	**K**	**[**	**k**	**{**
C	**,**	**<**	**L**	\	**l**	|
D	**-**	**=**	**M**	**]**	**m**	**}**
E	**.**	**>**	**N**	**^**	**n**	~
F	/	**?**	**O**	**_**	**o**	

Zyklische Codes: Bei zyklischen Codes wird das nachfolgende Codezeichen aus dem Vorgänger durch eine Verknüpfung (Polynom) generiert. Zyklische Vertauschung der Codebits (d.h. links- oder rechtsverschiebung) führt auf ein gültiges Codewort.

Beispiel einer Bildungsvorschrift eines n-Bit-Codes:

$G = \{g_{n-1}, \ldots g_0\}$: Polynomkoeffizienten (Schlüssel)

Hilfsgröße: $X = (x_{n-1}\cdot 2^{n-1} + x_{n-2}\cdot 2^{n-2} + \ldots + x_0\cdot 2^0)$

$X \neq 0$

$x_n = (g_{n-1}\cdot x_{n-1} + g_{n-2}\cdot x_{n-2} + \ldots + g_0\cdot x_0) \bmod 2$ (Modulo-Division → Rest)

$X = (x_{n-2}\cdot 2^{n-1} + \ldots + x_0\cdot 2^1 + x_n\cdot 2^0)$ (Links schieben, 2^n te Stelle verwerfen)

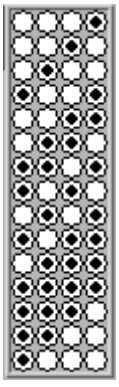
Beispiel1: n=4, G={1,1,0,0}, Anfangsbedingung: X={0,0,0,1}

$x_4 = (1\cdot x_3 + 1\cdot x_2 + 0\cdot x_1 + 0\cdot x_0) \bmod 2 = 0$

$X = 2$

fortlaufende Rechnung ergibt die (periodische) Codefolge (s. Bild):

1, 2, 4, 9, 3, 6, D, A, 5, B, 7, F, E, C, 8, ...



Beispiel2: n=4, G={1,0,1,0} liefert einen kürzeren zyklischen Code

1, 2, 5, A, 4, 8, ...

2.3 arithmetische Verknüpfungen

Grundrechenvorschriften: (Rechenregeln für binäre Zahlendarstellungen)

Addition:		Multiplikation:	
	0 + 0 = 0		0 · 0 = 0
	0 + 1 = 1		0 · 1 = 0
	1 + 0 = 1		1 · 0 = 0
	1 + 1 = 10 *)		1 · 1 = 1
Subtraktion:	0 - 0 = 0	Division:	0 : 0 = n.d.
	0 - 1 = -1 **)		0 : 1 = 0
	1 - 0 = 1		1 : 0 = n.d.
	1 - 1 = 0		1 : 1 = 1

*) in manchen Entwurfssystemen wird + und · bzw. + und & als logische Verknüpfung ∧ (UND) bzw. ∨ (ODER) genutzt, es ist jedoch nicht immer dem arithmetischen + (plus) und · (minus) identisch. Beispiel: 1 + 1 = 10; 1 ∧ 1 = 1

**) Die Darstellung eines Vorzeichens kann auch mit den binären Werten 0,1 erfolgen, in der Regel wird - durch 1 dargestellt. Beispiel: 0 - 1 = 11. Es sind besondere Vereinbarungen zur Unterscheidung von der positiven $3_D = 11$ notwendig.

Komplementzahlen: Das **B-Komplement** K_B ist die Ergänzung einer Zahl mit fester Stellenbreite zur nächst höheren Potenz der Basis des Zahlensystems B

$$K_B(z)=B^n-z \qquad (1)$$

Beispiele:

Zehnerkomplement einer n-stelligen Zahl z: $K_{10}(z) = 10^n - z$

Zweierkomplement einer n-stelligen Zahl z: $K_2(z) = 2^n - z$

Beispiele:

B=10 Dezimalzahl,	n=2,	$z=82_D$	$K_{10}(82) = 18_D$
B=10 Dezimalzahl,	n=1,	$z=6_D$	$K_{10}(6) = 4_D$
B=2, Dualzahl,	n=4,	z= 0110	$K_2(0110) =1010$

Das **(B-1) Komplement** ist das um 1 verminderte B-Komplement

$$K_{B-1}(z) = K_B(z) - 1 = B^n-z-1 \qquad (2)$$

Beispiele:

Neunerkomplement einer n-stelligen Zahl z: $K_9(z) = 10^n - z -1$

Einerkomplement einer n-stelligen Zahl z: $K_1(z) = 2^n - z -1$

B=10 Dezimalzahl,	n=2,	$z=82_D$	$K_9(82) = 17_D$
B=10 Dezimalzahl,	n=1,	$z=6_D$	$K_9(6) = 3_D$
B=2, Dualzahl,	n=4	z= 0110	$K_1(0110) =1001$

Zweierkomplement, Einerkomplement: Das Einerkomplement ist das um 1 verminderte Zweierkomplement, es ist im Dualsystem einfach durch **Invertierung** aller Binärstellen von z ermittelbar.

$$K_1(z)=(K_2(z)-1) = \bar{z} \qquad (3)$$

Beispiel: z = 0 1 0 1 (5_D) $(K_2(z)-1)$= 1 0 1 0 (10_D)

$K_2(z)$ = 1 0 1 1 (11_D)

Dezimal	Dual	B_1 Einer-komplement	B_2 Zweier-komplement
15	1 1 1 1	0 0 0 0	0 0 0 1
14	1 1 1 0	0 0 0 1	0 0 1 0
13	1 1 0 1	0 0 1 0	0 0 1 1
12	1 1 0 0	0 0 1 1	0 1 0 0
11	1 0 1 1	0 1 0 0	0 1 0 1
10	1 0 1 0	0 1 0 1	0 1 1 0
9	1 0 0 1	0 1 1 0	0 1 1 1
8	1 0 0 0	0 1 1 1	1 0 0 0
7	0 1 1 1	1 0 0 0	1 0 0 1
6	0 1 1 0	1 0 0 1	1 0 1 0
5	0 1 0 1	1 0 1 0	1 0 1 1
4	0 1 0 0	1 0 1 1	1 1 0 0
3	0 0 1 1	1 1 0 0	1 1 0 1
2	0 0 1 0	1 1 0 1	1 1 1 0
1	0 0 0 1	1 1 1 0	1 1 1 1
0	0 0 0 0	1 1 1 1	0 0 0 0

Anwendung von Komplementzahlen zur Subtraktion

D = M - S		
= M - S	$+ 2^n - 2^n$	Erweiterung um 2^n
= M + (2^n - S)	$- 2^n$	Voraussetzung: $z<2^n$
= M + **K_2(S)**	$- 2^n$	vgl. (1) mit $K_B(z)=K_{B-1}(z) + 1$
= M + K_1(S) + 1	$- 2^n$	vgl. (2), im Dualsystem gilt: $K_1(z) = \bar{z}$
= M + **$\bar{S}$ +1**	$- 2^n$	vgl. (3), $\bar{S}$: Bitweise Negation von S
D = Z	$- 2^n$	mit Additionsergebnis Z=M+S+1

Schlußfolgerung:

1. die Subtraktion im Dualsystem ist durch Addition der Zweierkomplementzahl des Subtrahenden mit anschließender Korrektur -2^n möglich.
2. die Zweierkomplementzahl $K_2(S)$ ist durch bitweise Negation und Addition von 1 technisch sehr einfach zu realisieren $K_2(S) =(\bar{S} + 1)$.
3. die anschließender Korrektur von Z mit -2^n zur Darstellung von D ist

a) trivial lösbar, wenn $Z \geq 2^n$ (weglassen der Binärstelle 2^n)
D = Z "+Z"
b) durch Komplementbildung lösbar, wenn $Z < 2^n$
das Ergebnis ist in dem Fall $D = -(2^n - Z)$ "-Z"
und damit $D = -(K_2(Z))$
auch hier kann $K_2(Z)$ durch bitweise Negation und Addition von 1 gebildet werden
$D = -(Z + 1)$

Nutzt man für negative Zahlen grundsätzlich die Komplementdarstellung $-D = (K_2(Z))$, entfällt die Korrektur nach Punkt 3.b). Da die Codes dieser negativen Zahlen sich von den positiven Zahlen unterscheiden müssen, halbiert sich der positive Zahlenbereich in einen positiven und einen negativen Bereich. Es ergeben sich einfache Rechenvorschriften für die Addition und Subtraktion vorzeichenbehafteter Zahlen:

(+A) + (+B)	Addition			
(+A) + (-B)	Addition			
(-A) + (-B)	Addition			
(+A) - (+B)	2er Komplementbildung von B	und	(+A) + (-B)	Addition
(-A) - (-B)	2er Komplementbildung von B	und	(-A) + (+B)	Addition

Ein negatives Vorzeichen ist an der führenden 1 in der Zweierkomplementdarstellung erkennbar.
n-Bit-Zweierkomplementzahlen haben einen festen Zahlenbereich von -2^{n-1} ... $2^{n-1}-1$.

Eine Erweiterung (Verringerung) des Zahlenbereiches ist durch
- Vorsetzen (Streichen) von Nullen bei positiven Zahlen bzw.
- Vorsetzen (Streichen) von Einsen bei negativen Zahlen möglich.

Offset-Binär-Code: Dieser Code wird durch Addition eines Offsets (z.B. von 2^{n-1}) zur vorzeichenbehafteten Zahl erzeugt. Es entstehen positive Zahlen. Gegenüber der Zweierkomplementdarstellung ist das führende Bit invertiert. Der Code wird häufig bei Digitalisierern (Analog-Digital-Umsetzern) oder zur Exponentendarstellung von Gleitkommazahlen (hier ist der Offset $2^{n-1}-1$) genutzt.

Dezimal	Dual, Vorzeichen-Betragsdarstellung	Zweierkomplement-Darstellung	Zweierkomplement Hexadezimal	Offset-Binär
9	0 1 0 0 1			
8	0 1 0 0 0			
7	0 0 1 1 1	0 1 1 1	7	1 1 1 1
6	0 0 1 1 0	0 1 1 0	6	1 1 1 0
5	0 0 1 0 1	0 1 0 1	5	1 1 0 1
4	0 0 1 0 0	0 1 0 0	4	1 1 0 0
3	0 0 0 1 1	0 0 1 1	3	1 0 1 1
2	0 0 0 1 0	0 0 1 0	2	1 0 1 0
1	0 0 0 0 1	0 0 0 1	1	1 0 0 1
0	0 0 0 0 0	0 0 0 0	0	1 0 0 0
-1	1 0 0 0 1	1 1 1 1	F	0 1 1 1
-2	1 0 0 1 0	1 1 1 0	E	0 1 1 0
-3	1 0 0 1 1	1 1 0 1	D	0 1 0 1
-4	1 0 1 0 0	1 1 0 0	C	0 1 0 0
-5	1 0 1 0 1	1 0 1 1	B	0 0 1 1
-6	1 0 1 1 0	1 0 1 0	A	0 0 1 0
-7	1 0 1 1 1	1 0 0 1	9	0 0 0 1
-8	1 1 0 0 0	1 0 0 0	8	0 0 0 0
-9	1 1 0 0 1			

Beispiele in der Programmiersprache C:

unsigned int 16 bit vorzeichenlos
$0 .. 65535_D$ bzw. $0 .. FFFF_H$

int 16 bit Zweierkomplementdarstellung
Dezimal: - 32768, .. , -1, 0, 1, .. ,+ 32767
Hexadezimal: 8000, .. , FFFF, 0000, 0001, .. ,7FFF

long 32 bit Zweierkomplementdarstellung
$-2.147.483.648_D$ ... -1, 0, 1 ... $2.147.483.647_D$
8000 0000 ... FFFF FFFF, 0000 0000, 0000 0001 ... 0FFF FFFF

2.4 Minimierungsverfahren

Ziel:
Gegebene Schaltfunktionen sollen unter gegebenen Randbedingungen mit möglichst geringem technischen Aufwand realisiert werden.

technischer Aufwand:	Anzahl der Gatter (Chipfläche, Anzahl der Schaltkreise) Verknüpfungsart (Wahl der Grundgatter z.B. Negator, NAND, XOR) Anzahl der Eingänge (abhängig von der Verknüpfungsart)
Randbedingungen:	Technologie (TTL, CMOS, ...) erforderliche Verknüpfungszeit (Geschwindigkeit) Parallelität: mehrstufige Anordnungen benötigen i.A. wenig Gatter bzw. wenig Eingänge, besitzen jedoch lange Signaldurchlaufzeiten (sind ”langsam”), zweistufige Anordnungen benötigen i.A. viele Gatter bzw. viele Eingänge, haben kurze Signallaufzeiten (sind ”schnell”)
Ansatz:	Minimierung der Schaltfunktion nach bestimmten Kriterien, z.B. minimale Anzahl von Verknüpfungen (=Gatteranzahl), minimale Anzahl von OR-Verknüpfungen (Minterme bzw. Produktterme in PLD´s), minimale Stufenanzahl (=max. Geschwindigkeit).
vereinfachter Ansatz:	einfache Schaltfunktionen ergeben meist auch einfache Realisierungen
Ausgangspunkt:	kanonisch disjunktive Normalform KDN

Vereinfachung der Schaltfunktion:

1. Vereinfachung mit Theoremen der Schaltalgebra:

Kriterium: minimale Anzahl der Symbole (Verknüpfungen, Variablen)
Bsp. KDN mit zwei Mintermen: $x_1 \wedge x_2 \vee x_1 \wedge \bar{x}_2$ (7 Symbole)
$= x_1 (x_2 \vee \bar{x}_2) = x_1$ (1 Symbol)

2. Grafische Verfahren:

Karnaugh-Veitch (KV)-Diagramme:

Für jede mögliche Vollkonjunktion - d.h. für jede Tabellenzeile in der ST - wird ein Tabellenplatz in einem Rechteck reserviert und der Wert der ST eingetragen. Die Tabellenplätze sind so angeordnet, daß sich zwei benachbarte Tabellenplätze nur in ***einer*** Eingangsvariablen, d.h. bezüglich x_i und $\bar{x}_i$ unterscheiden. (Bei n < 5 ist das geometrisch einfach möglich)
Ziel: quadratische oder rechteckige Zusammenfassung benachbarter Plätze mit dem Wert 1 (sogenannte Zweier-, Vierer-, Achtergruppen ..).
Innerhalb dieser Gruppen treten eine, zwei, drei, .. Variablen sowohl negiert als auch nichtnegiert auf, sind also eliminierbar.
Beispiel für eine Variable x_i: $(a \wedge x_i) \vee (a \wedge \bar{x}_i) = a$

Beispiele für KV-Diagramme:

(Die Zahlen in den Feldern entsprechen den Zeilen in der Schaltbelegungstabelle bzw. der dualcodierten Zahl $x_n \ldots x_1 x_0$)

n=2 ST Schaltbelegungstabelle

x_0	x_1	y
0	0	0
0	1	1
1	0	2
1	1	3

KV-Diagramm

x_1 \ x_0	0	1
0	0	1
1	2	3

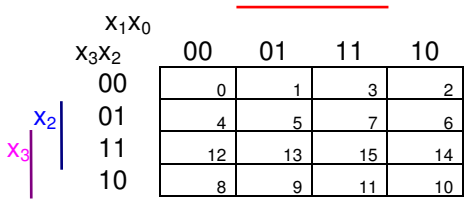
n=3 KV-Diagramm

x_2 \ x_1x_0	00	01	11	10
0	0	1	3	2
1	4	5	7	6

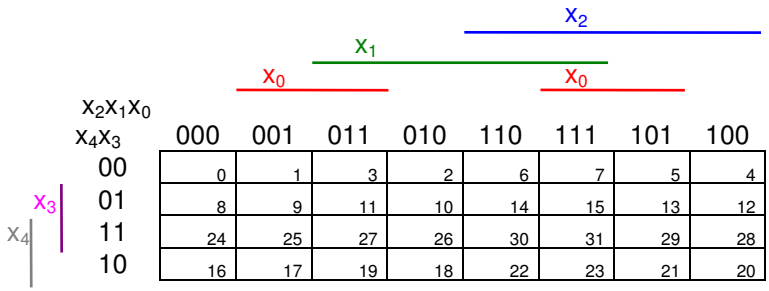
x_1, x_0, x_2

n=4

x_1, x_0



n=5



(hier sind auch Plätze der rechten Hälfte der Tabelle mit den spiegelbildlich gelegenen linken Plätzen benachbart, z.B. Platz 9 und 13 oder 11 und 15)

Minimierungsverfahren:

- Aufstellen einer Tabelle entsprechend der Anzahl der Eingänge
- Eintragung aller 0 und 1 Belegungen von y_i
- Zusammenfassung möglichst **weniger** und **großer** quadratischer oder rechteckiger Bereiche mit 1-Belegungen. Die Bereiche dürfen sich teilweise überlappen und über die Ränder hinaus gehen.
- Jeder Block bildet einen **Primimplikanten**, er wird aus der konjunktiven Verknüpfung der sich **nicht** ändernden Variablen gebildet. Die Variable werden dazu negiert ($\bar{x}$) erfaßt, wenn sie im Block konstant 0 sind, und nichtnegiert (x) erfaßt, wenn sie konstant 1 sind.
- die Primimplikanten werden **disjunktiv** verknüpft → minimierte Schaltfunktion

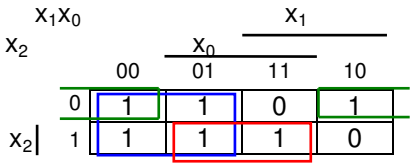
Modifikation:

- es kann ebenso die negierte Ausgangsfunktion $\bar{y}$ durch Zusammenfassung der 0-Bereiche gebildet und durch anschließende Negation bzw. nach dem de Morganschen Theorem in eine Ausgangsfunktion y umgewandelt werden.
- Wenn nicht alle Tabellenplätze definiert belegt wurden (unvollständige Beschreibung), kann an diese Stelle ein – oder × eingetragen werden und je nach günstigerer Blockbildung als 0 oder 1 gewertet werden.

Beispiel:

$$y = \bar{x}_2\,\bar{x}_1\,\bar{x}_0 \vee \bar{x}_2\,x_1\,\bar{x}_0 \vee \bar{x}_2\,\bar{x}_1\,x_0 \vee x_2\,\bar{x}_1\,x_0 \vee x_2\,\bar{x}_1\,\bar{x}_0 \vee x_2\,x_1\,x_0$$

n=3



Viererbereich: $\bar{x}_1$

Zweierbereiche: $x_2\,x_0$

$\bar{x}_2\,\bar{x}_0$

Ergebnis: $y = \bar{x}_1 \vee x_2\,x_0 \vee \bar{x}_2\,\bar{x}_0$

3. Numerische Verfahren

Verfahren nach *Quine und McCluscey*

Ausgangspunkt: kanonisch disjunktive Normalform KDN
Grundgedanke: systematischer Vergleich der einzelnen Vollkonjunktionen (Minterme), um Variablen nach der Beziehung $(x_i \vee \bar{x}_i) = 1$ zu eliminieren.
Systematisierung: Erstellung von Rubriken mit jeweils gleicher Anzahl von nichtnegierten, "bejahenden" also 1-Variablen.
Realisierung: Vergleich zwischen benachbarten Rubriken auf Unterscheidung in **genau einer** Variablen und Elimination dieser Variablen.
Beispiel: $\bar{x}_3\,\bar{x}_2\,x_1\,x_0 \vee \bar{x}_3\,\bar{x}_2\,x_1\,\bar{x}_0 = \bar{x}_3\,\bar{x}_2\,x_1$ (Elimination von x_0)
Mehrfachauftretende Terme werden gestrichen. Es ergibt sich eine Vereinfachung um 1 Stufe.

Die Fortsetzung des Verfahrens erfolgt solange, bis keine Unterscheidung mehr in einer Variablen auftritt. Alle nicht weiter zu vereinfachenden Terme bilden Primimplikanten (Primterme), die disjunktiv miteinander verknüpft werden.

Redundanzkontrolle: es müssen nur die Primterme zusammengefaßt werden, welche disjunktiv verknüpft alle Vollkonjunktionen (Minterme) erfüllen.

Beispiel: $y = \bar{x}_2\,\bar{x}_1\,\bar{x}_0 \vee \bar{x}_2\,x_1\,\bar{x}_0 \vee \bar{x}_2\,\bar{x}_1\,x_0 \vee x_2\,\bar{x}_1\,x_0 \vee x_2\,\bar{x}_1\,\bar{x}_0 \vee x_2\,x_1\,x_0$
0 2 1 5 4 7

	disjunktive Normalform			1. Vereinfachung			2. Vereinfachung		
0	$\bar{x}_2\,\bar{x}_1\,\bar{x}_0$	√	0,2	$\bar{x}_2$ - $\bar{x}_0$	p_0	0,1/4,5	- $\bar{x}_1$ -	p_2
2	$\bar{x}_2\,x_1\,\bar{x}_0$	√	0,4	- $\bar{x}_1\,\bar{x}_0$	√	0,4/1,5	- ~~$\bar{x}_1$~~ -	
4	$x_2\,\bar{x}_1\,\bar{x}_0$	√	0,1	$\bar{x}_2\,\bar{x}_1$ -	√			
1	$\bar{x}_2\,\bar{x}_1\,x_0$	√	4,5	$x_2\,\bar{x}_1$ -	√			
5	$x_2\,\bar{x}_1\,x_0$	√	1,5	- $\bar{x}_1\,x_0$	√			
7	$x_2\,x_1\,x_0$	√	5,7	x_2 - x_0	p_1			

Primimplikanten: $p_0 = \bar{x}_2\,\bar{x}_0$; $p_1 = x_2\,x_0$; $p_2 = \bar{x}_1$
Ergebnis: $y = \bar{x}_2\,\bar{x}_0 \vee x_2\,x_0 \vee \bar{x}_1$

Kontrolle:

		000 0	001 1	010 2	100 4	101 5	111 7
p_0	$\bar{x}_2\,\bar{x}_0$	×		×			
p_1	$x_2\,x_0$					×	×
p_2	$\bar{x}_1$	×	×		×	×	

Schlußfolgerung: in diesem Beispiel sind alle 3 Primimplikanten für die Minterme in den Spalten 1, 2, 4 und 7 unbedingt erforderlich (markierten Bereiche)
→ **wesentlich**e Primimplikanten
Treten zusätzlich unwesentliche Primimplikanten auf, d.h. werden die Spalten von mehreren Primimplikanten erfüllt, so wird eine minimale Anzahl gesucht, um alle Minterme zu überdecken.
"Test" aller Kombinationsmöglichkeiten, Anordnungsproblem!

2. Beispiel:

$$y = \underset{0}{\bar{x}_2\,\bar{x}_1\,\bar{x}_0} \vee \underset{2}{\bar{x}_2\,x_1\,\bar{x}_0} \vee \underset{1}{\bar{x}_2\,\bar{x}_1\,x_0} \vee \underset{6}{x_2\,x_1\,\bar{x}_0} \vee \underset{5}{x_2\,\bar{x}_1\,x_0} \vee \underset{7}{x_2\,x_1\,x_0}$$

	disjunktive Normalform	
0	$\bar{x}_2\,\bar{x}_1\,\bar{x}_0$	√
2	$\bar{x}_2\,x_1\,\bar{x}_0$	√
1	$\bar{x}_2\,\bar{x}_1\,x_0$	√
6	$x_2\,x_1\,\bar{x}_0$	√
5	$x_2\,\bar{x}_1\,x_0$	√
7	$x_2\,x_1\,x_0$	√

	1. Vereinfachung	
0,2	$\bar{x}_2$ - $\bar{x}_0$	p_0
0,1	$\bar{x}_2$ $\bar{x}_1$ -	p_1
2,6	- $x_1\,\bar{x}_0$	p_2
1,5	- $\bar{x}_1\,x_0$	p_3
6,7	x_2 x_1 -	p_4
5,7	x_2 - x_0	p_5

Vorläufiges Ergebnis:

$y = p_0 \vee p_1 \vee p_2 \vee p_3 \vee p_4 \vee p_5$

$y = \bar{x}_2\;\bar{x}_0 \vee \bar{x}_2\;\bar{x}_1 \vee x_1\,\bar{x}_0 \vee \bar{x}_1\,x_0 \vee x_2\,x_1 \vee x_2\,x_0$

Kontrolle :

		000 0	001 1	010 2	110 6	101 5	111 7
p_0	$\bar{x}_2\;\bar{x}_0$	×		×			
p_1	$\bar{x}_2\;\bar{x}_1$	×	×				
p_2	$x_1\,\bar{x}_0$			×	×		
p_3	$\bar{x}_1\,x_0$		×			×	
p_4	$x_2\;x_1$				×		×
p_5	$x_2\,x_0$					×	×

Von den vorhandenen unwesentlichen Primimplikanten wird eine minimale Anzahl gesucht, die alle Minterme überdecken.

p_0, p_3, p_4 : Variante 1 Primimplikanten mit bestmöglicher Überdeckung
p_1, p_2, p_5 : Variante 2 Primimplikanten mit bestmöglicher Überdeckung

Ergebnis:

$y = \bar{x}_2\;\bar{x}_0 \vee \bar{x}_1\,x_0 \vee x_2\,x_1$

$y = \bar{x}_2\;\bar{x}_1 \vee x_1\,\bar{x}_0 \vee x_2\,x_0$ (zwei gleichwertige Lösungen)

Weitere Varianten:

Weitere Varianten ergeben sich aus Identität/Exkluxiv-Oder (EQU / XOR) Extraktion aus der 1. Vereinfachung:

Vergleich $p_0\,p_5$: $x_2 \equiv x_0$
Vergleich $p_1\,p_4$: $x_2 \equiv x_1$
Vergleich $p_2\,p_3$: $x_1 \oplus x_0$

		000 0	001 1	010 2	110 6	101 5	111 7
$p_0\,p_5$	$x_2 \equiv x_0$	×		×		×	×
$p_1\,p_4$	$x_2 \equiv x_1$	×	×		×		×
$p_2\,p_3$	$x_1 \oplus x_0$		×	×	×	×	

Weitere mögliche Ergebnisse:

Es ergeben sich weitere 3 Lösungen unter Zuhilfenahme der EQU/XOR Verknüpfung:

$y = (x_2 \equiv x_0) \vee (x_2 \equiv x_1)$

$y = (x_2 \equiv x_0) \vee (x_1 \oplus x_0)$

$y = (x_2 \equiv x_1) \vee (x_1 \oplus x_0)$

ob diese Lösung *besser* oder *schlechter* als die Vorige ist, hängt von der Effizienz der Realisierung der EQU/XOR-Gatter ab. In CMOS-Technologie lassen sich z.B. sehr kleine uns schnelle XOR/EQU Gatter realisieren.