GTI-Zusammenfassung

6.4.2022 - WS 2021/22

1. Achtung!:

Diese Zusammenfassung ist (absichtlich!) nicht vollständig. Ich finde, gewisse Teile sind mir recht gut gelungen, deswegen stelle ich sie öffentlich zur Verfügung. Ich habe aber bewusst bestimmte Dinge ausgelassen, die ich (für mich) zu einfach, zu trivial oder nicht klausurrelevant fand.

Dementsprechend werden nicht alle Teile der Vorlesung behandelt und ich empfehle zusätzlich folgende Zusammenfassung anzuschauen: https://gitlab.cs.fau.de/ik15ydit/latexandmore/-/jobs/artifacts/autobuild/raw/GTI/gti_zusammenfassung.pdf?job=GTI

2. Inhaltsverzeichnis

3. CODIERUNG

3.1 DEFINITIONEN

Analogsignal

Zeit- und wertkontinuierlich (z. B. physikalische Größen).

Diskretisierung

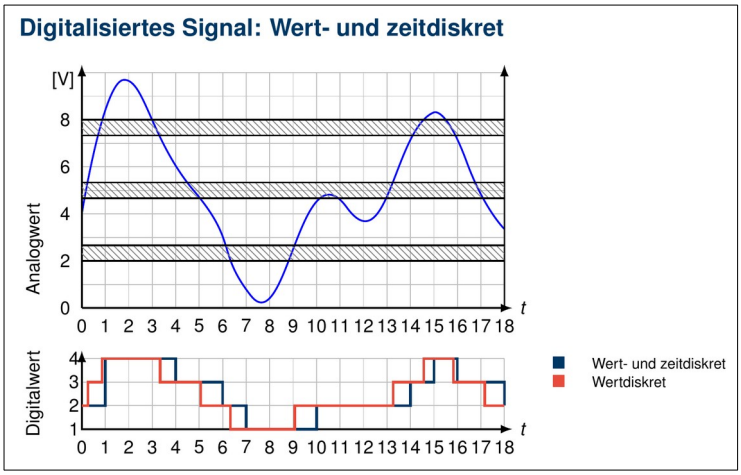
Einschränken der Menge möglicher Werte auf eine endliche Anzahl.

Undefinierter Bereich

Weiche Diskriminationsgrenze zwischen zwei Intervallen. D. h. in diesem Bereich wird der alte Wert beibehalten und erst beim Eintreten in den nächsten Bereich geändert.

Digitalsignal

Zeit- und wertdiskretes Signal.



Informationsgehalt $I(x)$

$$I(x) = ld\,\frac{1}{p(x)} = -ld\,p(x)\ \ [bit]$$

Je seltener ein Zeichen auftritt, desto höher ist sein Informationsgehalt.

Entropie H

Durchschnittlicher Informationsgehalt eines Alphabets mit N Zeichen.

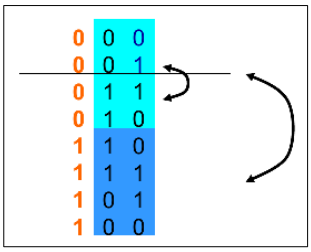
$$H = \sum_{i=1}^{N} p(x_i) \cdot I(x_i) = \sum_{i=1}^{N} p(x_i) \cdot ld\,\frac{1}{p(x_i)}$$

3.2 GRAY-CODES

Ein Gray-Code ist ein einschrittiger Code, bei dem sich nur ein Bit pro Wort ändert (Hamming-Distanz = 1).

Bildung eines Gray-Codes

1. Start: Schreibe eine 0 über eine 1.
2. Spiegele jeweils den gesamten geschriebenen Block.
3. Schreibe vor jedes Element des bisherigen Blocks eine 0 und vor jedes der Spiegelung eine 1.



3.3 Fehlererkennung mit Hamming-Distanz

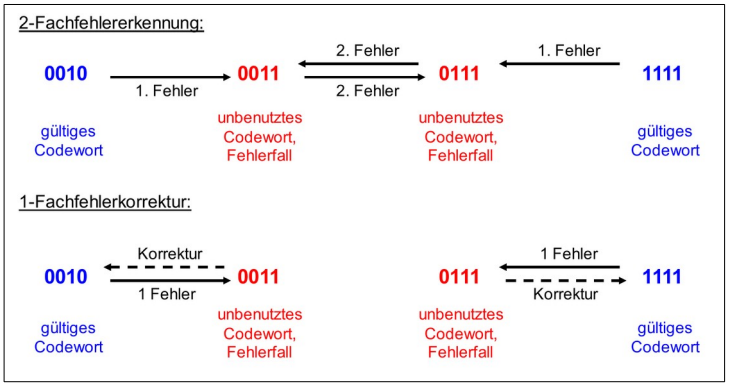
Hamming-Distanz (HD)

Anzahl der unterschiedlichen Binärstellen zwischen zwei gleich langen Codewörtern.

Minimale Hamming-Distanz (HD_{min})

Die kleinste Hamming-Distanz zwischen allen Codewörtern.

Eine minimale Hamming-Distanz d bedeutet, dass sich zwischen jedem Codewort minimal d Binärstellen ändern.



Erkennung von Einzelfehlern

Es können $(d-1)$-Fehler erkannt werden:

Ist ein $(d-1)$-Fehler aufgetreten, so ändern sich zwischen den richtigen und falschen Codewörtern weniger als d Stellen und das fehlerhafte Codewort kann erkannt werden.

Korrektur von Einzelfehlern

Es können $\frac{(d-1)}{2}$-Fehler korrigiert werden:

Ist ein $\frac{(d-1)}{2}$-Fehler aufgetreten, kann der erkannte Fehler $(d-1)$-sicher dem ursprünglichen Codewort zugeordnet werden, da nur zu diesem die Fehlerdistanz $\frac{(d-1)}{2}$ besteht.

Paritätsbit

An jedes Codewort wird ein zusätzliches Bit angehängt, das je nach Art der Parität (gerade/ungerade) bei einer geraden/ungeraden Anzahl an Einsen im Codewort den Wert 0 hat.

Blocksicherung

Erweiterung der Paritätssicherung auf zwei Dimensionen.
Die Parität wird spalten- und zeilenweise überprüft.

Ziffer	Codewörter mit gerader Parität				
5	0	1	0	1	0
4	0	1	0	0	1
1	0	0	1	1	1
3	0	0	1	1	0
9	1	0	0	1	0
8	1	0	0	0	1
Prüfwort	0	0	1	0	1

3.4 HAMMING-CODES

Anzahl an Paritätsbits: $2^k - k - 1 \geq m$

k: Anzahl Paritätsbits; m: Anzahl Informationsstellen

Bei Hexadezimalzahlen $(m = 4)$ also:

$2^k - k - 1 \geq 4 \rightarrow k = 3 \;\rightarrow\; m + k = 4 + 3 = 7$

Die natürlichen Zahlen (der Gesamtanzahl an Bits) werden binär in absteigender Reihenfolge angeordnet $(m + k)$.

7	6	5	4	3	2	1
1	1	1	(1)	0	0	0
1	1	0	0	1	(1)	0
1	0	1	0	1	0	(1)

Spalten mit genau einer 1 werden einem Paritätsbit zugeordnet.

x_4	x_3	x_2	y_3	x_1	y_2	y_1
1	1	1	1	0	0	0
1	1	0	0	1	1	0
1	0	1	0	1	0	1

Jedes y ist ein Ergebnis eines XORs $(\bigoplus)$ aller $x = 1$-Komponenten seiner Einserreihe

x_4	x_3	x_2	y_3	x_1	y_2	y_1
[1]	[1]	[1]	(1)	0	0	0
[1]	[1]	0	0	[1]	(1)	0
[1]	0	[1]	0	[1]	0	(1)

$y_1 = x_1 \oplus x_2 \oplus x_4$
$y_2 = x_1 \oplus x_3 \oplus x_4$
$y_3 = x_2 \oplus x_3 \oplus x_4$

3.5 OPTIMALE CODES

Minimierung der durchschnittlichen Codewortlänge m :

$\overline{m} = \sum_{i=1}^{N} p(x_i) \cdot m(x_i)$; m: Länge eines Wortes

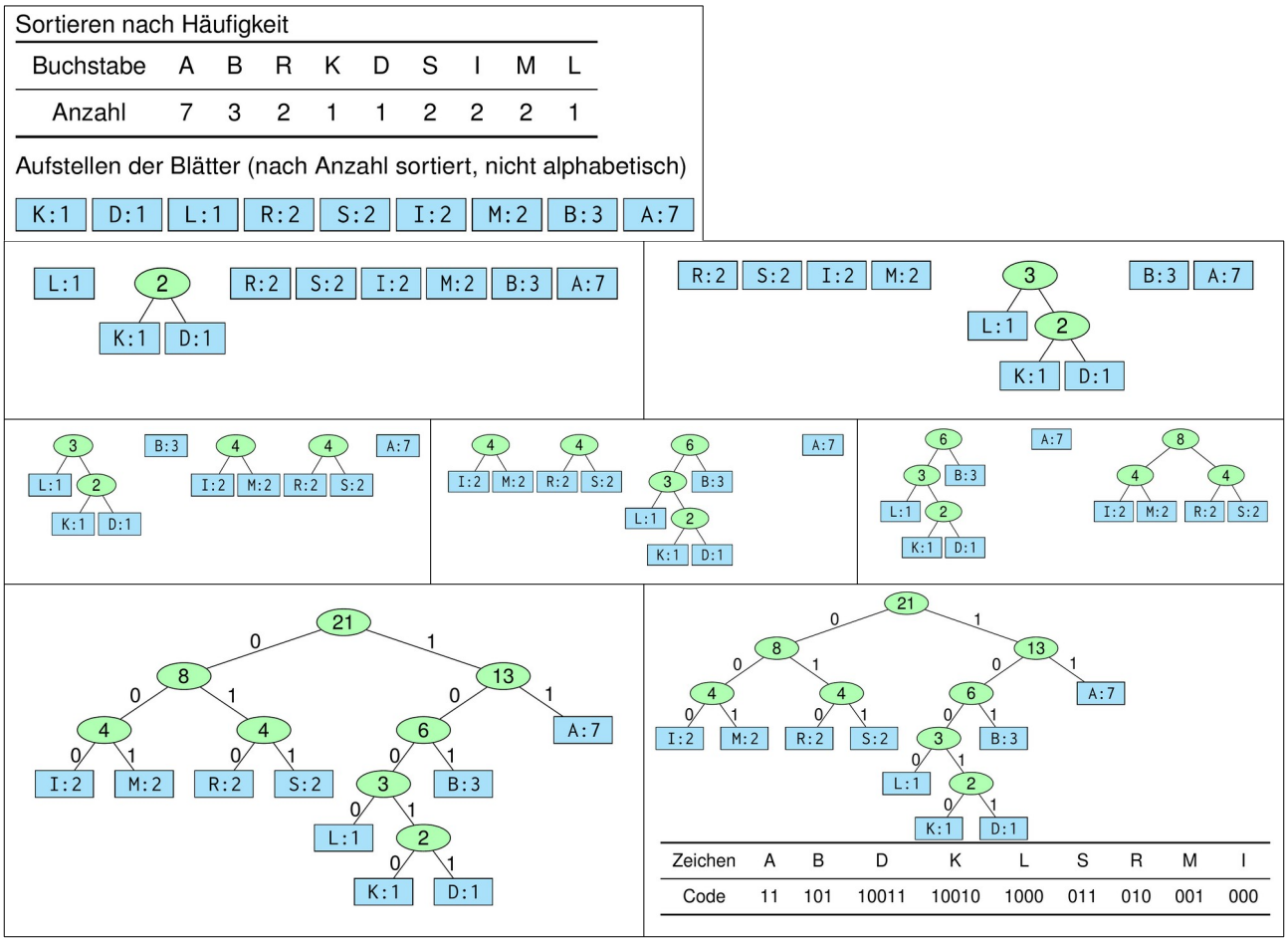
Präfixfreiheit

Kein Codewort ist Präfix eines anderen Codewortes.

3.6 Huffmann-Code (Präfix-Frei)

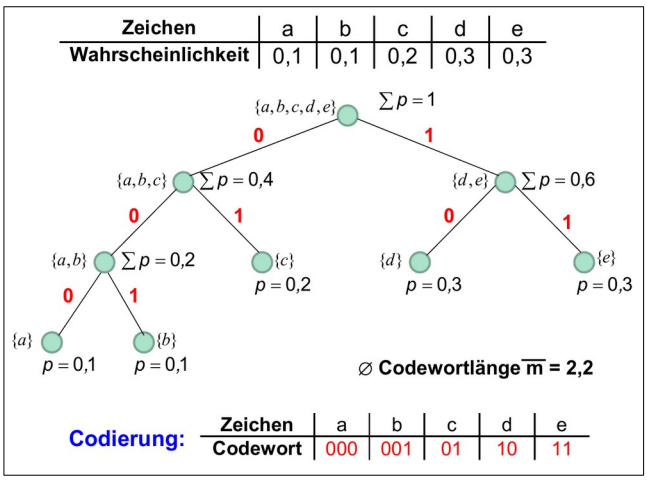
Effiziente Zuweisung von Codewörtern variabler Länge unter Ausnutzung ihres Informationsgehalts:

1. Buchstaben der Häufigkeit nach sortieren.
2. Buchstaben mit der geringsten Häufigkeit zusammenfassen.
3. Sukzessive die Knoten mit der geringsten Häufigkeit zum einem Baum zusammenfügen.
4. Beim fertigen Baum an jeden linken Ast eine 0, an jeden rechten Ast eine 1 schreiben.
5. Codewörter als Pfad des Baumes von der Wurzel zu den Blättern aus auslesen.



3.7 Shannon-Fano-Code (Präfix-Frei)

1. Zeichenvorrat nach aufsteigender Wahrscheinlichkeit linear sortieren.
2. Zwei Teilmengen so konstruieren, dass die Summenwahrscheinlichkeiten der beiden Teilmengen möglichst gleich groß sind.
3. Mit den jeweils resultierenden Teilmengen rekursiv in gleicher Weise fortfahren.
4. Beim fertigen Baum an jeden linken Ast eine 0, an jeden rechten Ast eine 1 schreiben.



4. ZAHLENSYSTEME

	Dezimalsystem	Binärsystem	Sedezimalsystem („Hexadezimalsystem“)	Oktalsystem
Basis:	10	2 (1 Bit (2^1))	16 (4 Bit (2^4))	8 (3 Bit (2^3))
Sequenz:	$r_i \in \{0,1,2,3,4,5,6,7,8,9\}$	$r_i \in \{0,1\}$	$r_i \in \{0,1,2,3,4,5,6,7$ $8,9,\mathrm{A},\mathrm{B},\mathrm{C},\mathrm{D},\mathrm{E},\mathrm{F}\}$	$r_i \in \{0,1,2,3,4,5,6,7\}$
Kennzeichnung:	$1303_{(10)}$	$1011_{(2)}, \underline{0\mathrm{b}}1001$	$\mathrm{BABE}_{(16)}, \underline{0\mathrm{x}}\mathrm{BABE}$	$1303_{(8)}, \underline{0}1303$

4.1 UMRECHNUNG VON ZAHLENSYSTEMEN

N		b	Quotient N'	Rest R
4867	/	16	304	3
304	/	16	19	0
19	/	16	1	3
1	/	16	0	1

$\rightsquigarrow 4867_{(10)} = 1303_{(16)}$

Modulo-Methode (Dezimalsystem(!) → Beliebiges System):

1. Dezimal(!)zahl N ganzzahlig durch Basis b teilen
   → Ganzzahliger Quotient N' und Rest R
2. R als Ziffer **an den Anfang** des Ergebnisses anhängen
3. Falls $N' \neq 0$: Mit N' als neue Zahl bei Schritt 1 wieder anfangen

Trick bei besonderen Zahlensystemen:

Zwei Zahlensysteme mit Basen a und b und es gilt $a^e = b$ für ein $e \in \mathbb{N}$. Dann können e Ziffern des Zahlensystems mit Basis a in eine Ziffer des Zahlensystems mit Basis b umgewandelt werden. Z. B. entspricht eine Hex-Ziffer 4 Binärziffern (2^4).

0010	1010
↕	↕
2	A

$\rightsquigarrow 101010_{(2)} = 2\mathrm{A}_{(16)}$

(4 Binärziffern = 1 Hex-Ziffer)

4.2 NEGATIVE ZAHLEN

Ab sofort: Zahlen mit fester Länge (hier: 8 Bit)

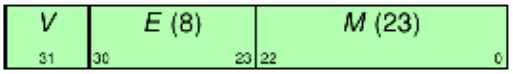
Vorzeichen-Betrags-Darstellung	Einerkomplement-Darstellung	Zweierkomplement-Darstellung
Vorderstes Bit gibt Vorzeichen an, restliche Bits den Betrag	Für negative Zahlen alle Bits des Betrags invertieren	Für negative Zahlen alle Bits des Betrags invertieren und 1 addieren (Von rechts kommend ab der ersten 1 alle Ziffern auf der linken Seite invertieren)
$+13_{(10)} \mapsto \underline{0}0001101_{(2)} \mapsto$ 0x0D $-13_{(10)} \mapsto \underline{1}0001101_{(2)} \mapsto$ 0x8D → Problem: Zwei Nuller (±0)	$+13_{(10)} \mapsto 00001101_{(2)} \mapsto$ 0x0D $-13_{(10)} \mapsto 11110010_{(2)} \mapsto$ 0xF2 → Problem: Zwei Nuller (±0)	$+13_{(10)} \mapsto 00001101_{(2)} \mapsto$ 0x0D $-13_{(10)} \mapsto 11110010_{(2)} + 1 = 11110011_{(2)} \mapsto$ 0xF3 → Nur noch eine Null
Erstes Bit, das sog. *Most Significant Bit* (MSB/„Vorzeichen-Bit“): 0 = Positiv; 1 = Negativ		

4.3 BINARY CODED DECIMAL (BCD)

Jede Dezimalstelle wird durch vier Binärstellen codiert ($2^3 < 10 < 2^4$). Sehr ineffiziente Speicherung, aber leicht rekonstruierbar.

Binär	i) Hexadezimal	ii) Oktal	Dezimal	iii) BCD
10	2	2	2	0010
1 0100	14	24	20	0010 0000
11 1111	3F	77	63	0110 0011
100 0000	40	100	64	0110 0100
1111 1111	FF	377	255	0010 0101 0101
1111 1110	FE	376	254	0010 0101 0100

4.4 GLEITKOMMAZAHLEN (IEEE)



- V: Vorzeichenbit (0 = positiv, 1 = negativ)
- M: Mantisse mit Aufbau $1, M$ (Die 1 vor dem Komma ist festgelegt und wird nicht mitgespeichert)
- E: Biased-Exponent. Der reale Exponent wird durch die Subtraktion eines *BIAS* (im IEEE-Format: 127) berechnet (also 2^{E-BIAS} bzw. 2^{E-127}). Dadurch ist E immer positiv.
- $BIAS$: Höchste Zahl, die sich mit 1 Bit weniger als E darstellen lässt $(2^{8-1} = 2^7 = 127)$

Insgesamt: $(-1)^V \cdot (1, M) \cdot 2^{E-BIAS}$

Spezialwerte des BIAS

Biased Exponent E	Mantisse M	Wert	
0 < E < 255	**M**	**(- 1)^V·2^{E-127} ·(1,M)**	
255	≠ 0	ungültiger Wert (NAN)	z.B. Division durch 0
255	0	(- 1)^V · ∞ (±unendlich)	Wert außerhalb des Zahlenbereichs
0	≠ 0	(- 1)^V·2^{-126} ·(0,M)	
0	0	(- 1)^V · 0	es gibt +0 und -0

Beispiele für Umrechnung Dezimal → Gleitkommazahl

Gegeben: $Z = (-1)^{VB} \cdot (1 + M) \cdot 2^{E-BIAS}$
Gesucht: Dezimalzahl zu 0 1001 1001 1001 1001 1001 0000 0000 000

Lösung:
Vorzeichenbit: 0_2
Charakteristik: $10011001_2 = 153_{10}$
Mantisse: $10011001100100000000000_2 = \frac{2457_{10}}{4096_{10}}$
BIAS: $B = 2^{n-1} - 1 = 2^{8-1} - 1 = 127_{10}$

$$Z = (-1)^0 \cdot (1 + \frac{2457}{4096}) \cdot 2^{153-127} = 107364352_{10}$$

Hinweis
$10011001100100000000000_2 \rightarrow 100110011001_2 = 2457_{10}$
Teilen durch $2^{Stellen}$: $\frac{2457_{10}}{(2^{12})_{10}} = \frac{2457_{10}}{4096_{10}}$

Gegeben: $Z = (-1)^{VB} \cdot (1 + M) \cdot 2^{E-BIAS}$
Gesucht: Dezimalzahl zu 1 0001 1001 1001 1001 0000 0000 0000 000.

Lösung:
Vorzeichenbit: 1_2
Charakteristik: $00011001_2 = 25_{10}$
Mantisse: $10011001000000000000000_2 = \frac{153_{10}}{256_{10}}$
BIAS: $B = 2^{n-1} - 1 = 2^{8-1} - 1 = 127_{10}$

$$Z = (-1)^1 \cdot (1 + \frac{153}{256}) \cdot 2^{25-127} = -0{,}3151 \cdot 10_{10}^{-30}$$

Hinweis
$10011001000000000000000_2 \rightarrow 10011001_2 = 153_{10}$
Teilen durch $2^{Stellen}$: $\frac{153_{10}}{(2^8)_{10}} = \frac{153_{10}}{256_{10}}$

Gegeben: $-6{,}25_{10} \cdot 10^{-3} = -0{,}00000\overline{0011}_2$.
Gesucht: IEEE-Darstellung $Z = (-1)^{VB} \cdot (1 + M) \cdot 2^{E-BIAS}$.

Lösung:
Vorzeichenbit: negativ, also 1_2
Mantisse: Angegeben ist keine Darstellung 1,M deshalb muss normalisiert werden (Shift nach links bzw. rechts)

$0{,}00000\overline{0011}_2 = 1{,}1\overline{0011} \cdot 2^{-8}$ (hier: Links-Shift → negativer Exponent). Daraus lassen sich Mantisse und der reale Exponent herleiten.

Lösung (fortgesetzt): $Z = (-1)^{VB} \cdot (1 + M) \cdot 2^{E-BIAS}$

$$0{,}00000\overline{0011}_2 = 1,\underbrace{1\overline{0011}}_{\text{Mantisse}} \cdot \underbrace{2^{-8}}_{\text{Realer Exponent}}$$

Mantisse: $10011001100110011001101_2$ (genau 23 Bit wählen)
$e_{real} = E - BIAS \Rightarrow E = e_{real} + BIAS = -8 + 127 = 119_{10}$
Charakteristik: $119_{10} = 01110111_2$

Zusammengesetzt: $-6{,}25_{10} \cdot 10^{-3} \approx 1$ 0111 0111 1001 1001 1001 1001 1001 101$_2$

Gegeben: $3{,}14159_{10} = 11{,}0010\ 0100\ 0011\ 1111\ 0011\ 11(1\ldots)_2$.
Gesucht: IEEE-Darstellung $Z = (-1)^{VB} \cdot (1 + M) \cdot 2^{E-BIAS}$.

Lösung:
Vorzeichen: 0, da positiv
Normalisierung: $1{,}10010\ 0100\ 0011\ 1111\ 0011\ 11(1\ldots)_2 \cdot 2^1$ (Shift nach rechts)
Mantisse: $10010010000111111010000_2$ (genau 23 Bit wählen)
Charakteristik: $E = e_{real} + BIAS = 1 + 127 = 128_{10} = 2^7 = 10000000_2$

Zusammengesetzt: $3{,}14159_{10} \approx 0$ 1000 0000 1001 0010 0001 1111 1010 000$_2$

4.5 GRUNDRECHENARTEN

Addition

Funktioniert wie in der Grundschule mit Übertrag.

	1 1 1 0 1 0 1 0 0 1 1 0	(3750_{10})
+	0 1 0 1 0 1 1 1 1 1 1 0	(1406_{10})
+	1 1 1 1 1 1 1 1 1 1 1 1	
=	1 0 1 0 0 0 0 1 0 0 1 0 0	(5156_{10})

Addition mit BCD-Zahlen

Bei jedem Übertrag und jeder Pseudotetrade $(x > 9)$ muss $6\ (0110_2)$ addiert werden.

Die Addition funktioniert prinzipiell wie die Binäraddition, mit folgenden Abweichungen:

- Stellen (je 4 Bit) werden getrennt addiert und der Übertrag in die nächste Stelle gezogen.
- Pseudotetraden müssen korrigiert werden.
- Überläufe müssen korrigiert werden, wenn sie nicht durch einen Korrekturschritt verursacht wurden.
- Korrekturschritt ist die Addition von $6\ (0110_2)$.

$775_{10} + 498_{10} = 1273_{10}$

			0 1 0 1		Erste Dezimalstelle
+			1 0 0 0		
=			1 1 0 1		Pseudotetrade
+			0 1 1 0		Korrekturaddition
=		1	0 0 1 1		
+		0 1 1 1			Zweite Dezimalstelle
+		1 0 0 1			
=	1	0 0 0 1			Überlauf
+		0 1 1 0			Korrekturaddition
=	1	0 1 1 1			
+	0 1 1 1				Dritte Dezimalstelle
+	0 1 0 0				
=	1 1 0 0				Pseudotetrade
+	0 1 1 0				Korrekturaddition
= 1	0 0 1 0	0 1 1 1	0 0 1 1		
= 1273_{10}					

Subtraktion

Abbilden der Subtraktion auf die Addition (nur Zweierkomplement): $a + (-b) = c$

	0 0 0 1 1 1 0 1 1 0	(118_{10})
+	1 1 0 1 1 0 0 1 1 1	(-153_{10})
+	1 1 1 1	
=	(0) 1 1 1 1 0 1 1 1 0 1	(-35_{10})

Multiplikation

Jede Ziffer der linken Zahl wird mit der rechten Zahl multipliziert und, um ihre jeweilige Stelle verschoben, unter die rechte Zahl geschrieben und dann aufaddiert.

	1101 · 1001
	1101
+	0000
+	0000
+	1101
Übertrag	0010000
Produkt	1110101

Addition/Subtraktion von Gleitkommazahlen

1. Rechtsschieben der Mantisse (implizite 1 nicht vergessen: $1, M$) der Zahl mit dem kleineren Exponenten, bis die Exponenten gleich sind (um $|E_1 - E_2|$ Stellen).
2. Addition/Subtraktion der Mantissen mit Vorzeichenbit (je nach Vorzeichen(-bit) noch das Zweierkomplement bilden): $V1, M \pm V1, M$
3. Normalisierung des Ergebnisses: Rechts-/Linksschieben (ohne das Vorzeichenbit!) bis wieder eine implizite 1 vor dem Komma steht. Bei einem negativen Ergebnis vorher das Zweierkomplement bilden (Mantissen sind vorzeichenlos).

(Mathematisch: $N_1 + N_2 = 2^{E_1 - BIAS} \cdot M_1 + 2^{E_2 - BIAS} \cdot M_2 = 2^{E_1 - BIAS} \cdot (M_1 + 2^{E_2 - E_1} \cdot M_2)$)

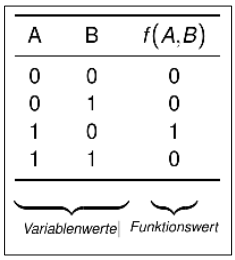
Multiplikation von Gleitkommazahlen

1. Vorzeichen getrennt behandeln
2. Addition der Exponenten: $E_1 + E_2 - BIAS$
3. Multiplikation der Mantissen: $1, M_1 \cdot 1, M_2$
4. Normalisierung des Ergebnisses (s. Addition)

(Mathematisch: $N_1 \cdot N_2 = 2^{(E_1 + E_2 - BIAS) - BIAS} \cdot M_1 \cdot M_2$)

5. Schaltfunktionen und Schaltalgebra

5.1 Definitionen



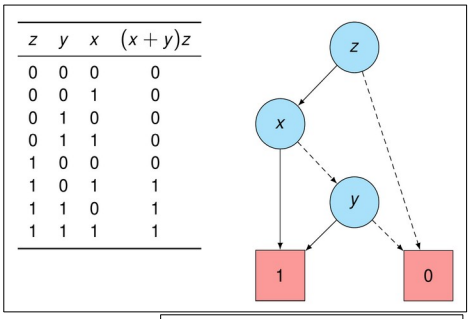
A	B	$f(A,B)$
0	0	0
0	1	0
1	0	1
1	1	0

Variablenwerte | Funktionswert

Funktionstabelle

- Enthält die Funktionswerte für alle Permutationen der Variablenwerte.
- Jede Variable kann entweder wahr $(=1)$ oder falsch sein $(=0)$.
- Es gibt somit bei n Variablen 2^n verschiedene Kombinationen (= Zeilen in der Tabelle).

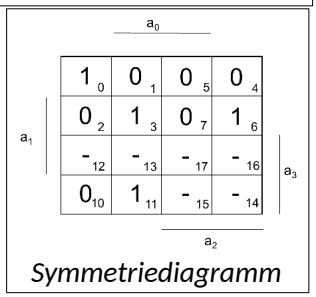
Binary Decision Diagrams (BDD)



z	y	x	$(x+y)z$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

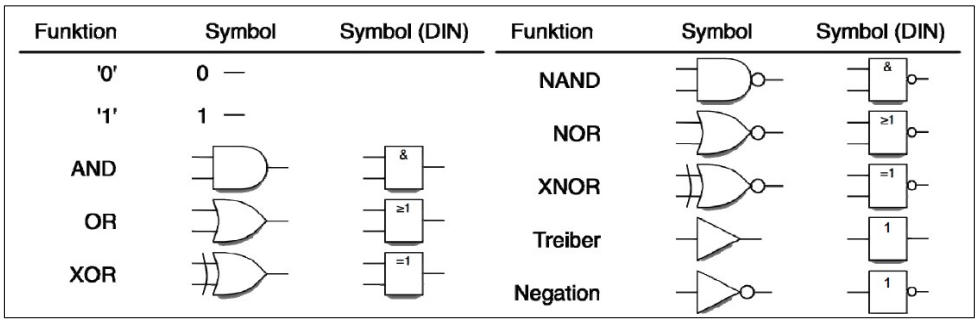
- Grafische Darstellung von Funktionstabellen.
- Erzeugung einer Baumstruktur mit Fallunterscheidung der Variablen.
- Pfad von der Wurzel bis zur $0/1$-Senke repräsentiert eine Ableitung mit Funktionswert $0/1$.

Symmetriediagramm (KV-Diagramm) → Siehe Rechts



Symmetriediagramm

5.2 Logik-Gatter



Funktion	Symbol	Symbol (DIN)	Funktion	Symbol	Symbol (DIN)
'0'	0		NAND		&
'1'	1		NOR		≥1
AND		&	XNOR		=1
OR		≥1	Treiber		1
XOR		=1	Negation		1

5.3 Bool'sche Algebra

Gesetze

Kommutativgesetz	$A \cdot B = B \cdot A$	$A + B = B + A$
Assoziativgesetz	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$	$(A + B) + C = A + (B + C)$
Distributivgesetz	$(A + B) \cdot C = A \cdot C + B \cdot C$	$(A \cdot B) + C = (A + C) \cdot (B + C)$
De Morgan	$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$

Rechenregeln

XOR	$A \bigoplus B = (A \cdot \overline{B}) + (\overline{A} \cdot B)$	
Idempotenz	$A + A = A$	$A \cdot A = A$
Absorption	$A \cdot (A + B) = A$	$A + (A \cdot B) = A$
Reduktion	$(A \cdot B) + (A \cdot \overline{B}) = A$	$(A + B) \cdot (A + \overline{B}) = A$
Konstanten	$\overline{A} + A = 1 \ \ \& \ \ \overline{A} \cdot A = 0$	

5.4 Normal-/Minimalformen

Minterme $(\cdot)$	**Maxterme** $(+)$
Konjunktion (Verundung) aller Literale einer Funktion in negierter oder nicht negierter Form. Ein Minterm entspricht einer 1 der Funktion.	Disjunktion (Veroderung) aller Literale einer Funktion in negierter oder nicht negierter Form. Ein Maxterm entspricht einer 0 der Funktion.
Disjunktive Normalform (DNF)	**Konjunktive Normalform (KNF)**
Disjunktion (Veroderung) aller Minterme (1en) einer Funktion. Beispiel: $x\bar{y}\bar{z} + xy\bar{z}$	Konjunktion (Verundung) aller Maxterme (Negierung der 0en) einer Funktion. Beispiel: $(x + \bar{y} + \bar{z}) \cdot (x + y + \bar{z})$
Disjunktive Minimalform (DMF)	**Konjunktive Minimalform (KMF)**
Disjunktion von, durch Konjunktion verknüpften Literalen, die (ohne diese Form zu verletzen) nicht weiter vereinfacht werden kann. Also eine vereinfachte DNF.	Konjunktion von, durch Disjunktion verknüpften Literalen, die (ohne diese Form zu verletzen) nicht weiter vereinfacht werden kann. Also eine vereinfachte KNF.

Gesucht: Disjunktive Form von $f_3(w,x,y,z) = (\bar{w}+z)\cdot(\bar{w}+x+y)\cdot(x+\bar{y}+z)$

Lösung:

$$
\begin{aligned}
f_3 &= (\bar{w}+z)(\bar{w}+x+y)(x+\bar{y}+z) \\
&\equiv (\bar{w}+z)(\bar{w}x+\bar{w}\bar{y}+\bar{w}z+x+x\bar{y}+xz+xy+y\bar{y}+yz) && \text{(Distribution)} \\
&\equiv (\bar{w}+z)(\bar{w}\bar{y}+\bar{w}z+x+y\bar{y}+yz) && \text{(Absorption)} \\
&\equiv (\bar{w}+z)(\bar{w}\bar{y}+\bar{w}z+x+yz) && \text{(Unerfüllbarkeit)} \\
&\equiv (\bar{w}+z)(\bar{w}\bar{y}+x+yz) && \text{(Konsensus)} \\
&\equiv \bar{w}\bar{y}+\bar{w}x+\bar{w}yz+\bar{w}\bar{y}z+xz+yz && \text{(Distribution)} \\
&\equiv \bar{w}\bar{y}+\bar{w}x+xz+yz && \text{(Absorption)}
\end{aligned}
$$

Gesucht: Konjunktive Form von $f_4(x,y,z) = xyz + \bar{x}y + \bar{y}z$

Lösung: Anwenden des Distributivgesetzes: $a + bc = (a+b)(a+c)$

$$
\begin{aligned}
f_4 &= xyz + \bar{x}y\bar{z} + \bar{x}yz + \bar{x}\bar{y}z + x\bar{y}z \\
&\equiv xyz + \bar{x}yz + \bar{x}yz + \bar{x}y\bar{z} + \bar{x}\bar{y}z + x\bar{y}z && \text{(Idempotenz)} \\
&\equiv (x+\bar{x})yz + (z+\bar{z})\bar{x}y + (x+\bar{x})\bar{y}z && \text{(Distribution)} \\
&\equiv \bar{x}y + \bar{y}z + yz && \text{(Immer wahr)} \\
&\equiv \bar{x}y + (\bar{y}+y)z && \text{(Distribution)} \\
&\equiv \bar{x}y + z && \text{(Immer wahr)} \\
&\equiv (\bar{x}+z)(y+z) && \text{(Distribution)}
\end{aligned}
$$

5.5 Umwandlung in NAND/NOR

Die disjunktive/konjunktive Form doppelt negieren und anschließend mit De-Morgan umformen.

6. Minimierungsverfahren

6.1 Definitionen

Primterm

Minimale Terme einer DMF bzw. KMF. Also maximale Einsen- bzw. Nullenstellenüberdeckung im Symmetriediagramm.

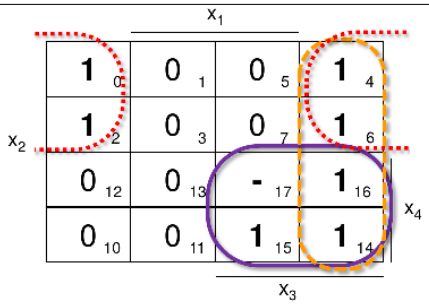
Primplikant

Primterm einer DMF. Also maximale Einsenüberdeckung inkl. *Don't Cares* im Symmetrie-Diagramm.

Primimplikat

Primterm einer KMF. Also maximale Nullenüberdeckung inkl. *Don't Cares* im Symmetrie-Diagramm.

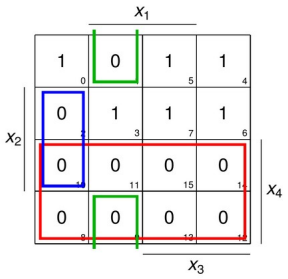
6.2 Minimierung am Symmetriediagramm



Primimplikanten:

$p_1 = x_4 x_3$ $p_2 = x_3 \overline{x_1}$ $p_3 = \overline{x_4}\,\overline{x_1}$

1. Auswahl derjenigen **Primimplikanten**, die **allein eine Einsstelle** überdecken (diese müssen genommen werden) → sogenannte ***Kerne***
2. Sind durch die Kerne alle Einsstellen überdeckt → DMF
3. sonst: Auswahl weiterer Primimplikanten notwendig



Negiert auslesen!

$KMF(y_1) = \overline{x_4} \cdot (x_3 + \overline{x_2} + x_1) \cdot (x_3 + x_2 + \overline{x_1})$

6.3 Nelson/Petrick-Verfahren

Nelson-Verfahren

1. Behandlung aller Freistellen (*Don't cares*) als Einsstellen!
2. Bestimmung einer konjunktiven Form mittels Nullblocküberdeckung.
3. Umformen auf eine disjunktive Form mittels Distribution und Absorption.
4. Streichen aller Terme, die nur Freistellen (*Don't cares*) überdecken.

Nun wurden alle Primterme gefunden, diese können sich aber noch überlappen → Es muss die beste Auswahl an Primtermen mit möglichst wenig Überlappung (Kosten) gefunden werden (Überdeckungsproblem).

Petrick-Verfahren

Jeder Primterm wird als eigenständige (Präsenz-)Variable (p_k) aufgefasst. Der sog. Patrick-Ausdruck (PA) ist eine disjunktive Normalform (DNF) aller Präsenzvariablen. D. h. für jede Einsstelle werden alle Variablen der überdeckenden Primterme miteinander verodert und dann insgesamt verundet.

Anschließend wird der Patrick-Ausdruck in die konjunktive Minimalform (KMF) umgewandelt.

Jeder Teilterm des Endergebnisses bildet dann eine mögliche Lösung des Überdeckungsproblems.

Die optimale Lösung ist der Teilterm mit den niedrigsten Kosten.

Der Patrick-Ausdruck muss immer $= 1$ liefern und ist somit eine Gleichung!

(Das Petrick-Verfahren kann auch bei zyklischen Resttabellen verwendet werden!)

Beispiel:

k	PI	j: 0	2	4	11	12	14	16	p_i	c_i
1	x_4x_2					x		x	p_1	c_1
2	$\overline{x_3}x_2$		x			x			p_2	c_2
3	$x_4x_3\overline{x_1}$						x	x	p_3	c_3
4	$x_4\overline{x_3}x_1$				x				p_4	c_4
5	$\overline{x_4}x_2\overline{x_1}$	x		x					p_5	c_5
6	$\overline{x_4}\overline{x_3}\overline{x_1}$	x	x						p_6	c_6
7	$x_3\overline{x_2}\overline{x_1}$			x			x		p_7	c_7

$$PA = (p_5 + p_6) \cdot (p_2 + p_6) \cdot (p_5 + p_7) \cdot p_4 \cdot (p_1 + p_2) \cdot (p_3 + p_7) \cdot (p_1 + p_3) \overset{!}{=} 1$$

$$PA = p_1p_4p_6p_7 + p_2p_3p_4p_5 + p_1p_3p_4p_5p_6 + p_2p_3p_4p_6p_7 + p_1p_2p_4p_5p_7 = 1$$

Überdeckungstabelle

Start: Für jeden Primterm wird angegeben, welche Einsstellen er überdeckt sowie dessen Kosten.

Wiederhole solange, bis keine Schritte mehr anwendbar sind:

1. Kerne finden und alle von Kernen vollständig überdeckten Spalten streichen.
2. Spaltendominanzen finden und dominierende Spalten streichen.
3. Zeilendominanzen finden und dominierte Zeilen streichen.

Oft endet das Verfahren mit einer sog. zyklischen Resttabelle (wenn keine der Regeln mehr anwendbar ist, aber noch keine Überdeckung gefunden wurde).

Kernimplikant (Kernterm):

Ein Primterm, der eine 1 überdeckt, die kein anderer Primterm überdeckt. Sind in der Tabelle durch nur einen Eintrag in einer Spalte erkennbar und müssen in die Überdeckungslösung aufgenommen werden.

Primimplikant	0	2	3	5	c_k
1			×		2
2	×		×	×	2
3		×		×	3
4	×			×	1

Spaltendominanz

Spalten, die andere Spalten dominieren, können gestrichen werden

Beispiel rechts: 3 wird von 5 dominiert, 0 wird von 5 dominiert

Primimplikant	0	2	3	5	c_k
1			×	×	2
2	×		×	×	2
3		×			3
4	×			×	1

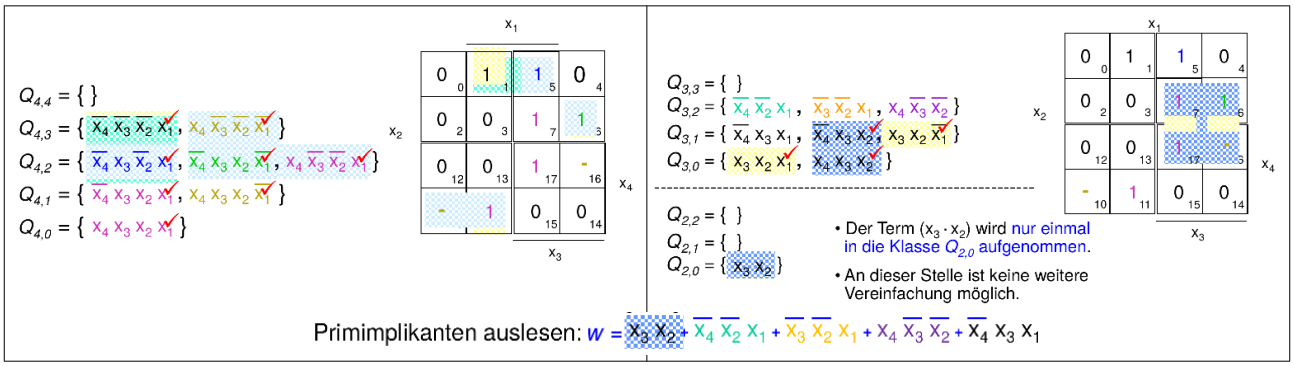
Zeilendominanz:

Eine dominierte Zeile kann gestrichen werden, falls ihre Kosten mindestens so hoch wie die der dominierenden Zeile sind und keine Zeilen existieren, welche die restlichen Einsstellen überdecken und in Summe mit der dominierten Zeile geringere Kosten aufweisen.

Primimplikant	0	2	3	5	c_k
1			×	×	2
2	×		×	×	4
3		×			3
4	×			×	1

6.4 QUINE/MCCLUSKEY-VERFAHREN

1. Bildung der Disjunktiven Normalform (alle Freistellen werden zu 1 gewählt und mitberücksichtigt).
2. Fasse die Implikanten zu Klassen $Q_{i,j}$ zusammen (i: Anzahl der Literale, j: Anzahl negativer Literale).
3. Implikanten benachbarter Klassen $Q_{i,j}$ und $Q_{i,j-1}$ werden mittels Distributivgesetz zu neuer Klasse $Q_{i-1,j-1}$ zusammengefasst und markiert.
4. Alle nicht markierten Terme sind Primimplikanten.
5. Überdeckt ein Primimplikant nur Elemente der Redundanzmenge, wird er verworfen.

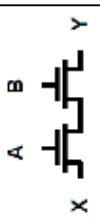


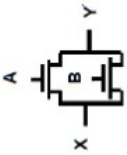
6.5 ZUSAMMENFASSUNG

Bestimmung der Primimplikanten	Kostenminimale Auswahl der Primimplikante (Lösung des Überdeckungsproblems)
• Symmetrie-Diagramm (grafisch) • Nelson-Verfahren (algebraisch) • Quine/McCluskey-Verfahren (tabellarisch)	• Symmetrie-Diagramme (grafisch) • Petrick-Verfahren (algebraisch) • Überdeckungstabelle (tabellarisch)

7. SCHALTNETZE

7.1 TRANSISTOR-SCHALTUNGEN





Logisches UND	Logisches ODER
Hintereinander	Parallel

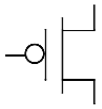
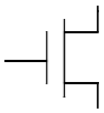
Transistor

Transfer Resistor, ein elektronischer Schalter.

MOS (Metal Oxid Semiconductor)

Konkrete Transistortechnologie mit folgenden Anschlüssen:

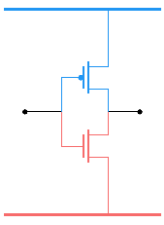
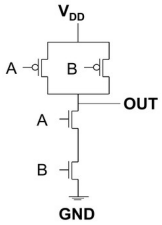
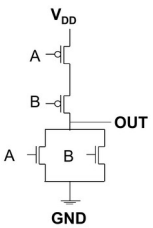
- Source: Quelle (Eingang) des Transistorsr
- Gate: Steuerkanal
- Drain: Senke (Ausgang) des Transistors

PMOS – Pull-up (PUN)	NMOS – Pull-down (PDN)
Öffner, n-dotiert → Leitet bei 0 	Schließer, p-dotiert → Leitet bei 1 
Nur negierte Literale Umformen: $PUN(\overline{A+B}) = \overline{A} \cdot \overline{B}$	Nur nicht-negierte Literale Schaltfunktion negieren: $PDN(\overline{A+B}) = \overline{\overline{A+B}} = A+B$

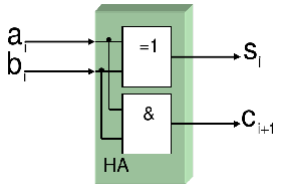
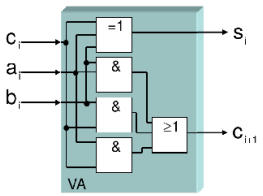
CMOS (Complementary MOS)

- Zusammensetzung aus komplementärem NMOS-Schaltnetz (Pull-Down-Netzwerk, da mit Erdung verbunden) und PMOS-Schaltnetz (Pull-Up-Netzwerk, da mit Versorgunsspannung verbunden).
- Vorteil: Kein Energieverbrauch im festen Schaltzustand, weil PDN und PUN komplementär sind und somit nur beim Schalten sehr kurz gleichzeitig leiten.
- Nachteil: Mehr Fläche auf Chip notwendig.

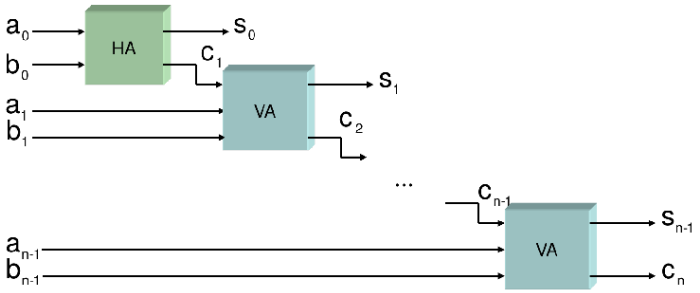
V_{DD} (Spannungsquelle)
↓
PMOS / PUN
↓
NMOS / PDN
↓
GND (Erdung)

Inverter	NAND	NOR
$f(x) = \overline{x}$ PUN: $\overline{x}$ PDN: $\overline{\overline{x}} = x$	$f(x,y) = \overline{x \cdot y}$ PUN: $\overline{x \cdot y} = \overline{x} + \overline{y}$ PDN: $\overline{\overline{x \cdot y}} = x \cdot y$	$f(x,y) = \overline{x+y}$ PUN: $\overline{x+y} = \overline{x} \cdot \overline{y}$ PDN: $\overline{\overline{x+y}} = x+y$
		

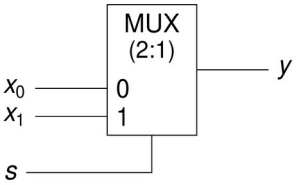
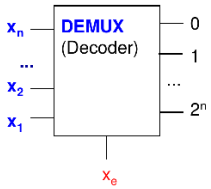
7.2 ADDIERER

Halbaddierer (HA)	Volladdierer (VA)
	

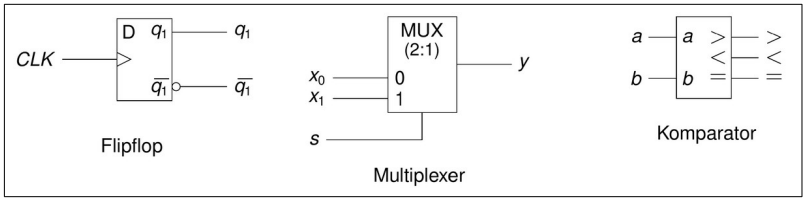
n-Bit Ripple-Carry-Addierer



7.3 MULTIPLEXER

Multiplexer	Demultiplexer
Bekommt n Leitungen und eine binäre Zahl (s) als Eingabe und leitet dann die entsprechende Leitung zum Ausgang durch.	Bekommt n Leitungen und eine weitere spezielle Leitung $(x_?)$. Anschließend wird der Index dieser speziellen Leitung ausgegeben.
	

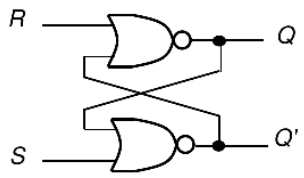
Weitere Schaltbausteine



7.4 Latches (Pegelgesteuert)

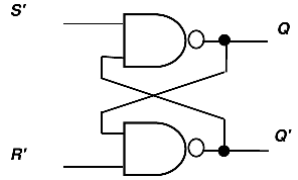
RS-Latch

Active-High



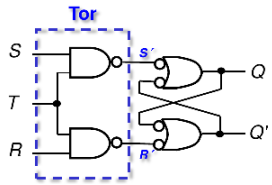
	S	R	Q	Q'
Set:	1	0	1	0
Hold:	0	0	1	0
Reset:	0	1	0	1
	0	0	0	1
	1	1	0	0

Active-Low

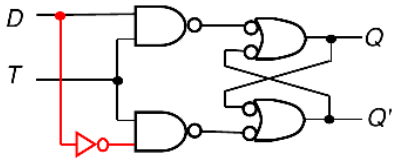


	S'	R'	Q	Q'
Reset:	1	0	0	1
Hold:	1	1	0	1
Set:	0	1	1	0
	1	1	1	0
	0	0	1	1

Getaktetes RS-Latch

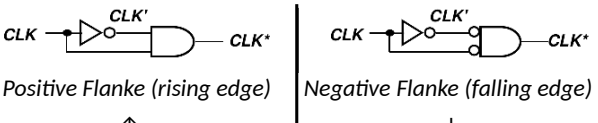


D-Latch (kein ungültiger Zustand)

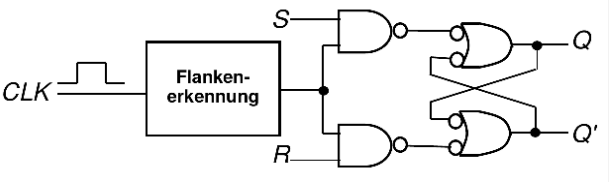


7.5 FlipFlops (Flankengesteuert)

Flankenerkennung:

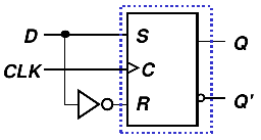


RS-FlipFlop



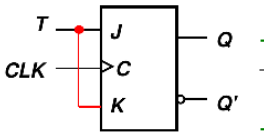
S	R	CLK	Q(t+1)	Zustand
0	0	X	Q(t)	Kein Wechsel
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	?	ungültig

D-FlipFlop



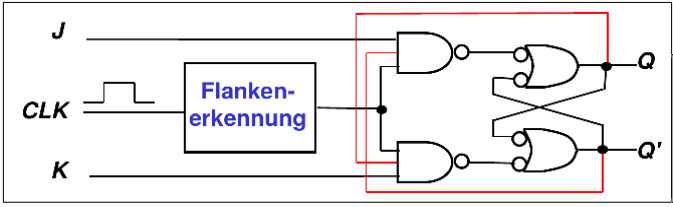
D	CLK	Q(t+1)	Zustand
1	↑	1	Set
0	↑	0	Reset

T-FlipFlop (Toggle)



T	CLK	Q(t+1)	Zustand
0	↑	Q(t)	halten
1	↑	Q(t)'	wechseln

JK-FlipFlop

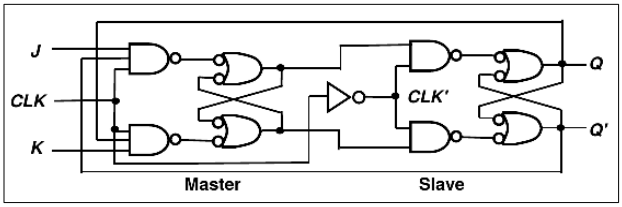


J	K	Takt	Q(t+1)	Zustand
0	0	↑	Q(t)	halten
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	Q(t)'	wechseln

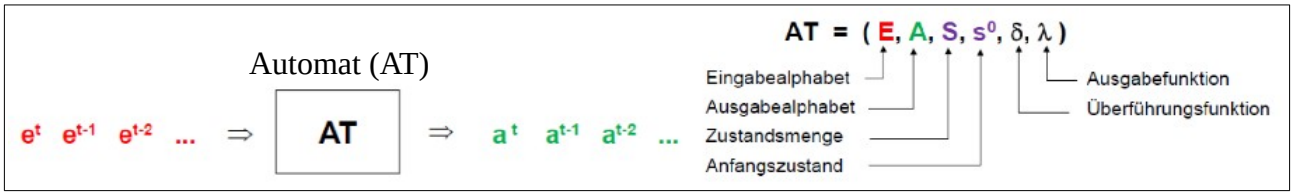
Master-Slave-JK-FlipFlop

Master ist aktiviert bei steigender Flanke, Slave ist aktiviert bei fallender Flanke → Verzögerte Weiterleitung des Masterwertes an den Ausgang.

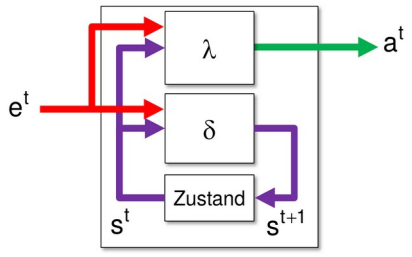
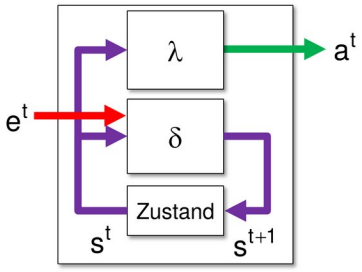
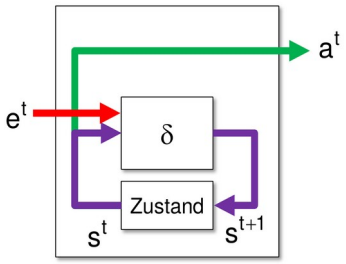
Die Wertetabelle entspricht dem JK-FlipFlop.



8. AUTOMATEN

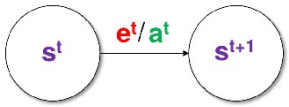


8.1 AUTOMATENARTEN

Mealy-Automat	Moore-Automat	Medwedew-Automat
Ausgabe hängt vom jeweiligen Zustand und der Eingabe ab	Ausgabe hängt alleine vom Zustand ab	Ausgabe ist der Zustand selbst
$a^t = \lambda(e^t, s^t)$	$a^t = \lambda(s^t)$	$a^t = s^t$
		

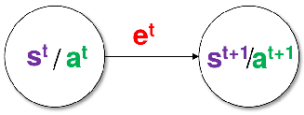
8.2 ZUSTANDSDIAGRAMM & AUTOMATENTAFEL

Mealy-Automat



s^t	s^{t+1} / a^t e_1	… e_i …	e_n
…	…		
s_k	…	s_j/a_m	…
	…		

Moore-Automat



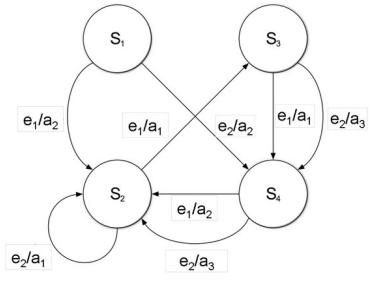
s^t	s^{t+1} e_1	… e_i …	e_n	a^t
…	…			
s_k	…	s_j	…	a_m
	…			

Darstellung

Konzept	Darstellung	Beispiel
Zustände (Q)	Kreise mit eindeutiger Bezeichnung	S_0
Anfangszustand (q_0)	Pfeil ohne Quelle	S_0
Übergangsfunktion	Zustände verbindender Pfeil	S_0 —b→ S_1

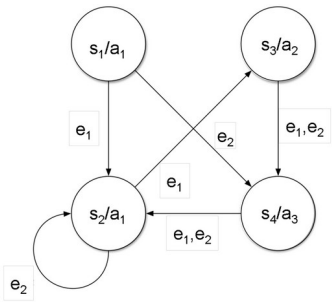
Mealy-Automat (Bsp)

s^t	s^{t+1} / a^t e_1	e_2
s_1	s_2/a_2	s_4/a_2
s_2	s_3/a_1	s_2/a_1
s_3	s_4/a_1	s_4/a_3
s_4	s_2/a_2	s_2/a_3



Moore-Automat (Bsp)

s^t	s^{t+1} e_1	e_2	a^t
s_1	s_2	s_4	a_1
s_2	s_3	s_2	a_1
s_3	s_4	s_4	a_2
s_4	s_2	s_2	a_3



9. VHDL

9.1 Syntax

Bezeichner (Namensgebung)

- Am Ende jeder Code-Zeile muss ein Semikolon stehen (einzige Ausnahme ist die letzte Zeile der Port-List bei `ENTITY`)!
- Groß-/Kleinschreibung wird nicht beachtet.
- Immer mit einem Buchstaben anfangen, keine Sonderzeichen und nie mehr als zwei Unterstriche hintereinander.

Kommentar

Beginnt mit `--` und endet mit dem Zeilenende: `--Dies ist ein Kommentar`

Konstanten

```
CONSTANT name : datatype := value
    CONSTANT bus_width : INTEGER := 8;
```

Variablen (:=)

- Sind nur ein gedankliches Konstrukt und werden nachher in der Hardware in Signale umgewandelt.
- Nur in `PROCESS`-Blöcken deklarier- und benutzbar.
- Nur zum temporären Speichern geeignet.
- Die Wertzuweisung erfolgt sofort, wenn die Variablenzuweisung ausgeführt wird.

```
VARIABLE name{, name2} : datatype [:= value];
    VARIABLE row, column : INTEGER RANGE 0 TO 31;
    VARIABLE tmp : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
```

Signale (<=)

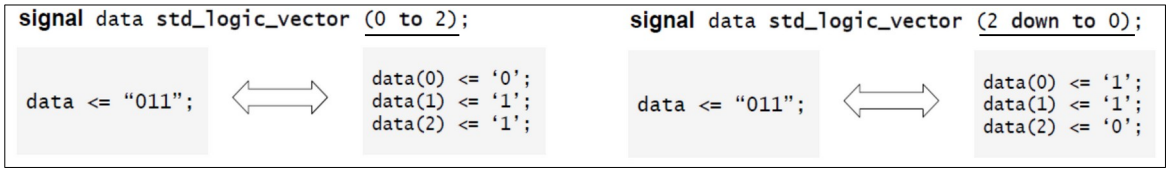
- Sind vergleichbar mit einer physikalischen Leitung oder einem Register.
- Können nicht in `PROCESS`-Blöcken deklariert werden.
- Die Wertzuweisung erfolgt erst, wenn der Prozess angehalten hat (bei `END PROCESS`) und nur mit dem zuletzt zugewiesenen Wert.
- Initialwerte für Signale werden von Synthesewerkzeugen ignoriert!

```
SIGNAL name{, name2} : datatype [<= value];
    SIGNAL clk, reset : BIT <= '0';  --Initialwerte werden ignoriert
    SIGNAL counter : INTEGER RANGE 0 TO 31;
```

(Eckige Klammern [] bedeuten die Angabe ist optional. Geschweifte Klammern {} bedeuten die Angabe kann beliebig oft wiederholt werden.)

Datentypen

```
• INTEGER RANGE 0 TO 100;
• STD_LOGIC;
• STD_LOGIC_VECTOR (0 TO 2);      --Bitvektor(Array), Index von li nach re
• STD_LOGIC_VECTOR (2 DOWNTO 0);  --Bitvektor(Array), Index von re nach li
• SIGNED (3 DOWNTO 0);  --Bit-Arrays (std_logic_vector), mit Rechenopera-
• UNSIGNED (3 DOWNTO 0) --tionen; Cast mit: std_logic_vector(s1)
```



Aufzählungsdatentypen

```
TYPE name IS (s0, s1, s2);
    TYPE state_type IS (IDLE, READY, ERROR, SEND, ...);
    SIGNAL present_state: state_type;
    CASE present_state IS WHEN IDLE => ...
```

Operatoren

- Logische Operatoren: `and`, `or`, `nand`, `nor`, `xor`
- Relationale Operatoren: `=`, `/=`, `<`, `<=`, `>`, `>=`
- Addition und Konkatenation: `+`, `-`, `&`
- Vorzeichen: `+`, `-`
- Multiplikation: `*`, `/`, `mod`, `rem`
- Exponent, Absolutbetrag, Komplement: `**`, `abs`, `not`

Flankenerkennung

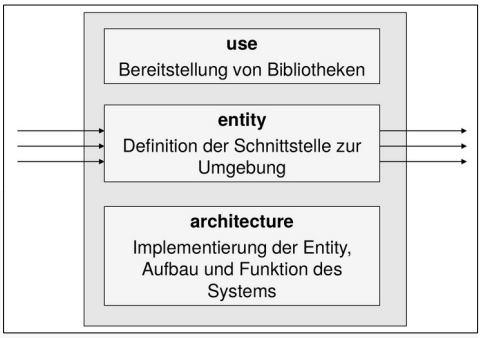
```
IF(rising_edge(clk)) THEN ...          IF(falling_edge(clk)) THEN ...
```

9.2 KONTROLLSTRUKTUREN

If-clause	Switch-case

`IF condition THEN`
`  --Code`
`{ELSIF condition THEN`
`  --Code }`
`[ELSE`
`  --Code ]`
`END IF;`
	`CASE expression IS`
	`  WHEN choice =>`
	`  --Code`
	`  {WHEN choice =>`
	`  --Code }`
	`  [WHEN OTHERS =>`
	`  --Code ]`
	`END CASE;`

9.3 Aufbau



Entity

Beschreibt „Blackbox-artig“ die Inputs und Outputs eines Moduls.

```
ENTITY entity_name IS
    --Portliste:
    PORT(
        input1      : IN STD_LOGIC;  --Eingangsport
        output1     : OUT SIGNED(7 DOWNTO 0);  --Ausgangsport
        output2     : OUT STD_LOGIC  --in der letzten Zeile kein Semikolon!
    );
END [ENTITY] entity_name;  --Eckige Klammern bedeuten, die Angabe ist optional
```

Architecture

Beschreibt den inneren/internen Aufbau eines **ENTITY**. Alle Prozesse werden nebenläufig ausgeführt.

```
ARCHITECTURE architecture_name OF entity_name IS
    --Interne Signale:
    SIGNAL intern : STD_LOGIC;    SIGNAL output : INTEGER RANGE 0 TO 5;
    --Importieren anderer Entitys:
    COMPONENT entityA
        PORT (a0 : IN STD_LOGIC;
              a1 : OUT INTEGER RANGE 0 TO 5);
    END COMPONENT;
BEGIN
    --Umbenennen der Portliste der import. Entitys in Signale der Architecture:
    EA: entityA port map(a0 => intern, a1 => output);
    --Verschiedene nebenläufige Prozesse (PROCESS), s. unten
END [ARCHITECTURE] architecture_name;
```

Process

Beschreibt den eigentlichen sequentiellen Code. Die Sensitivitätsliste beschreibt alle das Ergebnis beeinflussenden Signale (der **ENTITIY** und **ARCHITECTURE**). Der **PROCESS** wird auch nur aufgerufen wenn sich eines dieser Signal ändert. Hat keinen „Rückgabewert“.

```
PROCESS (signal0, signal1) IS
    --Lokale Variablen:
    VARIABLE tmp : SIGNED(3 DOWNTO 0) = "0100";
BEGIN
    --Code
END PROCESS;
```

10. ANHANG

10.1 BINÄRE DIVISION

Restoring Division

In jedem Schritt wird versuchsweise eine Subtraktion ausgeführt und bei negativem Rest rückgängig gemacht.

Restoring Division (am Beispiel 436_{10}: $15_{10} = 29_{10}$ R 1_{10})

	0	1	1	0	1	1	0	1	0	0	|	-	-	-	-	$R^{(0)}$, $Q^{(0)}$
0	1	1	0	1	1	0	1	0	0	-	|	-	-	-	-	linksschieben, subtrahieren
	0	1	1	1	1											
0	0	1	1	0	0	0	1	0	0	-	|	-	-	-	1	positiver Rest
0	1	1	0	0	0	1	0	0	-	-	|	-	-	1	-	linksschieben, subtrahieren
	0	1	1	1	1											
0	0	1	0	0	1	1	0	0	-	-	|	-	-	1	1	positiver Rest
0	1	0	0	1	1	0	0	-	-	-	|	-	1	1	-	linksschieben, subtrahieren
	0	1	1	1	1											
0	0	0	1	0	0	0	0	-	-	-	|	-	1	1	1	positiver Rest
0	0	1	0	0	0	0	-	-	-	-	|	-	1	1	1	linksschieben, subtrahieren
	0	1	1	1	1											
1	1	1	0	0	1	0	-	-	-	-	|	-	1	1	1	0	negativer Rest addieren
	0	1	1	1	1											
0	0	1	0	0	0	0	-	-	-	-	|	-	1	1	1	0	
0	1	0	0	0	0	-	-	-	-	-	|	1	1	1	0	-	linksschieben, subtrahieren
	0	1	1	1	1											
R	0	0	0	0	1	-	-	-	-	-	|	Q	1	1	1	0	1	positiver Rest

Non-performing Division

Der Partialrest wird nur dann durch die Differenz ersetzt, wenn diese nicht-negativ war.

Non-performing Division (am Beispiel 436_{10}: $15_{10} = 29_{10}$ R 1_{10})

	0	1	1	0	1	1	0	1	0	0	|	-	-	-	-	$R^{(0)}$, $Q^{(0)}$
0	1	1	0	1	1	0	1	0	0	-	|	-	-	-	-	linksschieben, subtrahieren
	0	1	1	1	1											
0	0	1	1	0	0	0	1	0	0	-	|	-	-	-	-	1	positiver Rest
0	1	1	0	0	0	1	0	0	-	-	|	-	-	-	1	-	linksschieben, subtrahieren
	0	1	1	1	1											
0	0	1	0	0	1	1	0	0	-	-	|	-	-	-	1	1	positiver Rest
0	1	0	0	1	1	0	0	-	-	-	|	-	-	1	1	-	linksschieben, subtrahieren
	0	1	1	1	1											
0	0	0	1	0	0	0	0	-	-	-	|	-	-	1	1	1	positiver Rest
0	0	1	0	0	0	0	-	-	-	-	|	-	1	1	1	-	linksschieben, subtrahieren
	0	1	1	1	1											
1	1	1	0	0	1	0	-	-	-	-	|	-	1	1	1	0	negativer Rest
0	0	1	0	0	0	0	-	-	-	-	|	-	1	1	1	0	alten Rest kopieren
0	1	0	0	0	0	-	-	-	-	-	|	1	1	1	0	-	linksschieben, subtrahieren
	0	1	1	1	1											
R	0	0	0	0	1	-	-	-	-	-	|	Q	1	1	1	0	1	positiver Rest

Non-Restoring Division

Nach einer Subtraktion wird mit einem eventuell entstehenden negativen Rest weitergearbeitet.

Statt Subtraktionen werden in diesem Fall Additionen vorgenommen, bis der Partialrest dadurch wieder nichtnegativ geworden ist.

Endet das Verfahren mit einem negativen Partialrest, muss daraus durch einen Korrekturschritt der entsprechende positive Rest berechnet werden.

Non-Restoring Division

Beispiel mit Korrekturschritt am Ende ($90_{10} : 7_{10} = 12_{10}$ R 6_{10})

Schritt	Partialrest								Quotient				*Beschreibung*
0	0	1	0	1	1	0	1	0	-	-	-	-	$R^{(0)}$>=0, $Q^{(0)}$
	1	0	1	1	0	1	0	0	-	-	-	-	2 * $R^{(0)}$, $Q^{(0)}$
	0	1	1	1	0	0	0	0	-	-	-	-	B´ subtrahieren => q_0= 1
1	0	1	0	0	0	1	0	0	-	-	-	1	$R^{(1)}$>=0, $Q^{(1)}$ = 2 * $Q^{(0)}$ + q_0
	1	0	0	0	1	0	0	0	-	-	-	1	2 * $R^{(1)}$, $Q^{(1)}$
	0	1	1	1	0	0	0	0	-	-	-	1	B´ subtrahieren => q_1= 1
2	0	0	0	1	1	0	0	0	-	-	1	1	$R^{(2)}$>=0, $Q^{(2)}$ = 2 * $Q^{(1)}$ + q_1
	0	0	1	1	0	0	0	0	-	-	1	1	2 * $R^{(2)}$, $Q^{(2)}$
	0	1	1	1	0	0	0	0	-	-	1	1	B´ subtrahieren => q_2= 0
3	1	1	0	0	0	0	0	0	-	1	1	0	$R^{(3)}$<0, $Q^{(3)}$ = 2 * $Q^{(2)}$+ q_2
	1	0	0	0	0	0	0	0	-	1	1	0	2 * $R^{(3)}$, $Q^{(3)}$
	0	1	1	1	0	0	0	0	-	1	1	0	B´ addieren -> q_3= 0
4 = l	1	1	1	1	0	0	0	0	1	1	0	0	$R^{(l)}$<0, $Q^{(l)}$ = 2 * $Q^{(3)}$+ q_3
	1	1	1	1	0	0	0	0	1	1	0	0	Nicht schieben: Nur
	0	1	1	1	0	0	0	0	1	1	0	0	B´ **zur Korrektur** addieren
	0	1	1	0	0	0	0	0	1	1	0	0	$R^{(l+1)}$, $Q^{(l)}$

<u>Wiederhole für jede Stelle von b:</u>

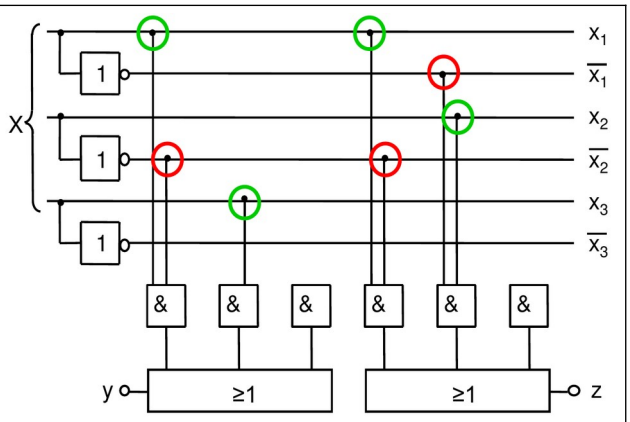
1. Schiebe a und q um eins nach links.
2. $R_i = \begin{cases} a-(-B'), & \text{wenn } carry = 1 \\ a-B', & \text{wenn } carry = 0 \text{ (Korrektur)} \end{cases}$
3. Speichere das $carry$ als niedrigstes Bit im Quotienten q.

Ist das letzte $carry = 0$: Addiere B' als Korrektur.

10.2 SCHALTNETZSTRUKTUREN

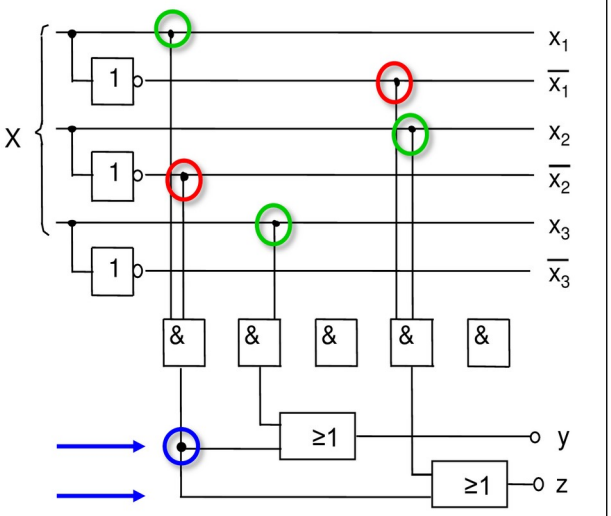
PAL (Programmable Array Logic)

- Freie UND-Matrix
- Feste ODER-Matrix



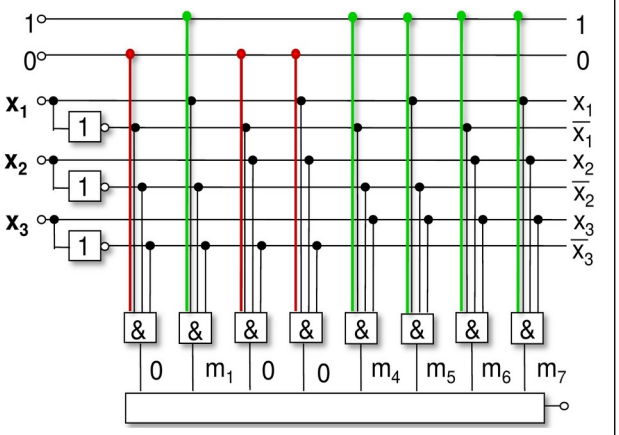
PLA (Programmable Logic Array)

- Freie UND-Matrix
- Freie ODER-Matrix



ULA (Universal Logic Array)

- Feste UND-Matrix
- Feste ODER-Matrix
- Auswahl der 2^n Minterme programmierbar



ROM (Read Only Memory)

- Feste UND-Matrix
- Freie ODER-Matrix