1. Grundlagen digitaler Systeme

Sie studieren Informatik. Die Informatik geht mit Rechnern um. Rechner sind heute fast ausschließlich Digitalrechner. Digitalrechner sind digitale Systeme, aber nicht jedes digitale System ist ein Rechner.

Jede vernünftige Wissenschaft setzt an den Anfang eine Reihe von Definitionen und Begriffsbestimmungen. Je mehr das sind, desto unverdaulicher ist eine Wissenschaft.

Auch wenn man nichts von Definitionen hält und einen eher pragmatischen Zugang zu einer Wissenschaft sucht, ist es hilfreich, die verwendeten Begriffe zu hinterfragen. In aller Regel bürgern sich nämlich nur solche wissenschaftlichen Termini ein, die einen semantischen, sprich: inhaltlichen, Bezug zu dem Gegenstand haben, den sie bezeichnen. Ein ehemaliger Kollege formulierte das immer so: "Ein Putzlappen heißt Putzlappen, weil es ein Putzlappen ist". Wir werden also gelegentlich Begriffe linguistisch deuten, um hinter deren Bedeutung zu kommen.

In den Unterlagen Prof. Monjaus finden Sie eine Definition des Begriffes **System**:

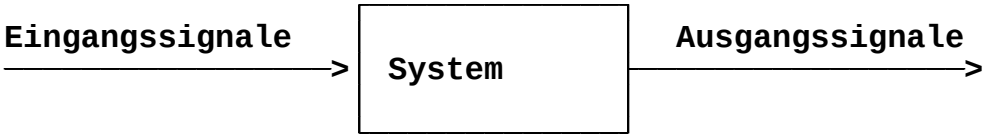
*Ein **System** ist ein nach bestimmten Gesichtspunkten abgegrenzter Bereich der objektiven Realität sowie jedes seiner **Abbilder** (**Modelle**). Der nicht zum System gehörende Teil heißt **Umwelt** (**Umgebung, environment**). System und Umwelt werden durch den **Systemrand** (**Schnittstelle, interface**) voneinander getrennt.*

*Ein System ist aus kleineren Einheiten, den **Elementen**, zusammengesetzt. Zwischen den Elementen können **Beziehungen** (**Relationen**) bestehen. Ebenso können Beziehungen zwischen dem System und der Umgebung bestehen.*

*Die Relationen bestimmen zusammen mit den Elementen die **Struktur** (structure) des Systems. Das System als Ganzes wie auch die Elemente haben ein bestimmtes **Verhalten** (**behaviour**), d. h. sie erfüllen eine bestimmte Funktion.*

Ich schlage Ihnen aber vor, diese Definition zunächst mal beiseite zu lassen. Wenn Sie dann eine ganze Reihe konkreter Systeme kennengelernt haben, können Sie sich leicht davon überzeugen, daß die erwähnte Definition tatsächlich alle diese konkreten Systeme überdeckt. Wir behelfen uns zunächst mit einer vereinfachten Darstellung:

Def.:



Ein System verarbeitet Signale.

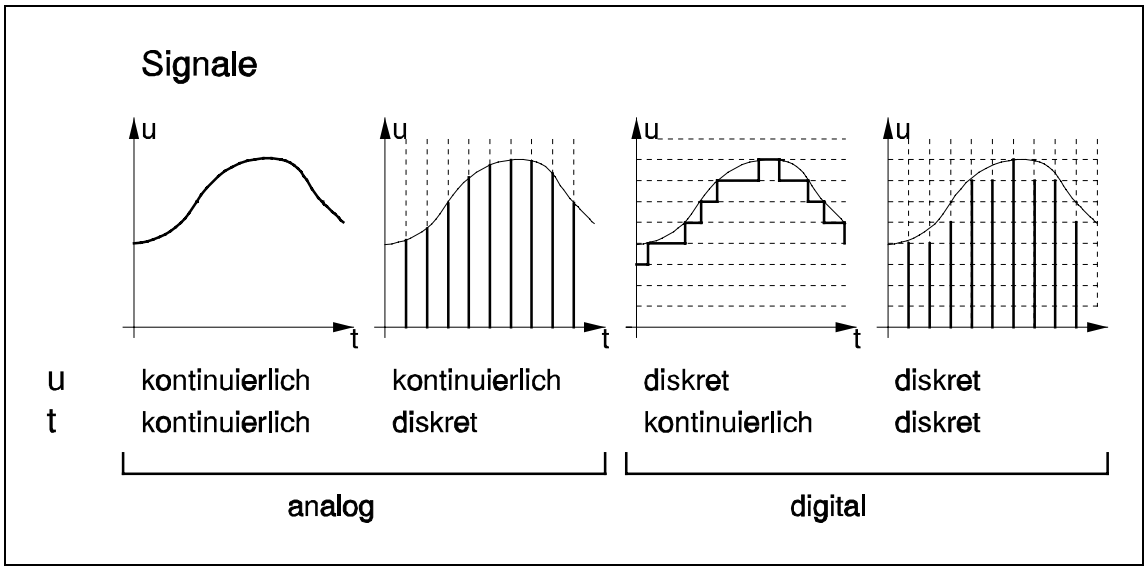
Kommen wir zum Begriff **digital**. Digital ist ein Allerweltswort geworden. Was bedeutet es? Die ursprüngliche Bedeutung "Finger" oder "Zehe" bringt uns nicht allzuweit. Die Bedeutung "Ziffer" eigentlich auch nicht.

Def.: **Ein System heißt dann digital, wenn es digitale Signale verarbeitet.**

Was ist ein Signal und wann ist es digital?

Def.: **Ein Signal ist eine zeitlich veränderliche physikalische Größe (z. B. elektrische Spannung, Druck, Temperatur, Wasserstand, ...), die eine auf sie abgebildete Information trägt.**

Sowohl der Wert der physikalischen Größe als auch ihr zeitlicher Verlauf können kontnuierlich (stetig) oder diskontinuierlich (unsteig, diskret) sein. Man unterscheidet danach vier Signaltypen:

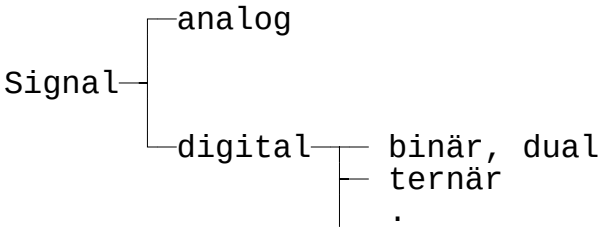


Analoge Signale können innerhalb bestimmter Grenzen theoretisch **beliebig viele Werte** annehmen. Ein analoges Signal hat theoretisch eine unendlich hohe Auflösung. In der Praxis sind ihr jedoch Grenzen gesetzt (Rauschen!).

Digitale Signale können innerhalb bestimmter Grenzen nur **endlich viele, wohlunterscheidbare Werte** annehmen. Abhängig davon, wie-

viele unterschiedliche Werte definiert sind, spricht man von **binären** (dualen, zweiwertigen), **ternären** (dreiwertigen) usw. **(Digital-)signalen**.

Wenn Sie das äußere Erscheinungsbild einer Uhr mit Zeiger (analog) und einer Uhr mit Ziffernanzeige (digital) vergleichen, wird Ihnen der Unterschied sofort klar: Die Genauigkeit, mit der Sie die Zeit von einer Analoguhr ablesen können, ist theoretisch unbegrenzt hoch. Bei einer Digitaluhr ist die Genauigkeit auf die niederwertigste Stelle der Anzeige begrenzt. Bei elektrischen Meßinstrumenten (Spannung, Strom, Widerstand, ...) ist die Situation die gleiche. Analoge Zeigerinstrumente liefern Werte, deren Genauigkeit nur durch Ablesefehler (Parallaxe, ..., dagegen Spiegelskala) und div. systematische Fehler (Klasse des Instruments) begrenzt wird. Digitale Meßinstrumente liefern Werte, deren Genauigkeit durch die Anzahl Ziffern der Anzeige (und div. systematische Fehler) begrenzt ist.



Analoge Signale sind die historisch älteren. Die klassische Nachrichtentechnik und die klassische Unterhaltungselektronik (Telefon, Rundfunk, Fernsehen, ...) sind Beispiele. Analogrechner wurden vor allem für wissenschaftliche Zwecke eingesetzt.

Mit dem Siegeszug des Digitalrechners und den heutigen Möglichkeiten integrierter digitaler Schaltkreise sind analoge Verfahren weitestgehend zurückgedrängt worden. In Massenbedarfsartikeln wie Fernsehempfängern halten sie sich oft nur noch deshalb, weil entsprechende digitale Verfahren zwar bekannt sind, die Standardisierung aber noch andauert und der Übergang nicht finanzierbar ist. Auf den Fachmessen (Funkausstellung!) kann man sich anschauen, was schon alles entwickelt worden ist.

In der modernen digitalen Schaltungstechnik verwendet man fast ausschließlich **binäre** Digitalsignale. Die Gründe dafür sind vorwiegend technischer Natur. Wir werden später auf diese Problematik näher eingehen.

Das adäquate mathematische Handwerkszeug ist die **BOOLEsche Algebra**. Die BOOLEsche Algebra ist eng verwandt mit der sog. Aussagenlogik. Über beide mathematische Methoden werden Sie bereits in der Schule gehört haben. Sie hören auch in Ihrer Mathematikausbildung von der BOOLEschen Algebra. Wir werden hier die Anwendung der BOOLEsche Algebra beim Entwurf digitaler Systeme genauer kennenlernen.

2. Zahlendarstellung

2.1. Positionssysteme

Was muß man sich merken?

Basis $b \in \{2,3,...\}$

Zahl $z = d_{m-1}d_{m-2}...d_1d_0.d_{-1}d_{-2}...d_{-n}{}_b$

mit **Ziffer** (**Digit**) d_p auf **Position** p

und $d_p \in \{0,1,...,b-1\}$

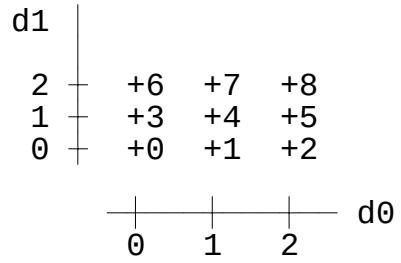
Die Ziffern (Digits) werden der Eindeutigkeit wegen häufig mit Bezeichnungen belegt, aus denen die Basis b erkennbar wird:

b = 2 => bit (**b**inary dig**it**)
b = e => nit (**n**atural dig**it**), fällt aus dem Rahmen!
b = 10 => dit (**d**ecimal dig**it**)

Wert $w = \sum_{p=-n}^{m-1} d_p * b^p$

Für jeden Wert gibt es genau eine Darstellung in einem Positionssystem (Orthogonalität).

Beispiel 2.1: zweistellige Ternärzahl $z_3 = d_1d_0{}_3$



Beispiel 2.2: hinterhältig und völlig nutzlos

Gesucht ist eine Zahl z zur Basis b, für die gilt

$z_b = \sqrt{121_b}$

Lösung:

Was ist zunächst zur Basis b zu sagen?

Aus $d \in \{0,1,...,b-1\}$ folgt $b \geq 3$.

Wie geht es weiter?

$$\text{Aus } w = \sum_{p=-n}^{m-1} d_p * b^p \quad \text{folgt}$$

$$z_b = \sqrt{1*b^2 + 2*b^1 + 1*b^0} = \sqrt{b^2 + 2b + 1}$$

$$= b + 1 = 1*b^1 + 1*b^0 = \mathbf{11}_b$$

Von einiger Bedeutung sind neben dem Dezimalsystem nur das Dual-, das Oktal- und das Hexadezimalsystem. Die übrigen Zahlensysteme haben kaum eine praktische Bedeutung. Nichtsdestoweniger wird die Konvertierung zwischen beliebigen Zahlensysteme gern und ausgiebig geübt.

Auf das Dezimalsystem muß ich nicht näher eingehen, es ist uns geläufig. Im Zusammenhang mit (Binär-)Rechnern haben das Dual- oder Binärsystem sowie das Oktal- und das Hexadezimalsystem eine große Bedeutung erlangt.

Dualsystem $b = 2$, $d \in \{0,1\}$
Oktalsystem $b = 8$, $d \in \{0,1,2,3,4,5,6,7\}$
Hexadezimalsystem $b = 16$, $d \in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

Das Dualsystem ist das mathematische Abbild der technischen Gegebenheiten in Binärrechnern. Die technische Basis jedes Binärrechners sind digitale (genauer: binäre) Schaltkreise. Sie verarbeiten nur zwei technisch gut voneinander unterscheidbare Pegel, die als Werte einstelliger Binärzahlen interpretiert werden => Binärsystem.

Binärzahlen sind schlecht lesbar, d. h. es gelingt kaum, "mit einem Blick" den Wert einer hinreichend langen Binärzahl zu erfassen. Wegen $8 = 2^3$ bzw. $16 = 2^4$ können jeweils Gruppen von 3 bzw. 4 Bits als eine Oktal- bzw. Hexadezimalzahl beschrieben werden, wohlgemerkt nur **beschrieben**. Es gibt keinen Oktal- oder Hexadezimalrechner. Oktal- bzw. Hexadezimalzahlen dienen nur der kompakteren Darstellung von Binärzahlen und damit in erster Linie der bequemeren Dokumenation.

Problematisch wird es erst, wenn beide Systeme parallel verwendet werden (IBM <--> DEC).

2.2. Konvertierung von Zahlen

Es wird die Konvertierung von Positionszahlen im Zahlensystem zur Basis b in wertgleiche Positionszahlen im Zahlensystem zur Basis B betrachtet

2.2.1. Konvertierung ganzer Zahlen

Fall 1

$b = 2^i$, $B = 2^j$, $i,j \in \{1,2,...\}$, $i \neq j$, d. h. b und B sind ganze (ungleiche) Zweierpotenzen

Fall 1 ist (fast) der einzige Fall, der in der praktischen Arbeit am Rechner Bedeutung hat!

Korrespondenztabelle (die sollte man im Kopf haben)

DUAL	OKTAL	HEXADEZIMAL
0000	00	0
0001	01	1
0010	02	2
0011	03	3
0100	04	4
0101	05	5
0110	06	6
0111	07	7
1000	10	8
1001	11	9
1010	12	A
1011	13	B
1100	14	C
1101	15	D
1110	16	E
1111	17	F

Fall 1.1

i = 1, j ≠ 1, d. h. Konvertierung **aus dem Dualsystem**

Vom Dezimalpunkt aus nach links (und rechts) werden jeweils j Bit zu einer Gruppe zusammengefaßt. Führende (und endende) Nullen sind ggf. zu ergänzen. Jede j-Bit-Gruppe wird durch das entsprechende Ziffernsymbol aus der Basis B gemäß Korrespondenztabelle ersetzt.

Beispiel 2.3:

$101011000110_2 = 101\_011\_000\_110_2 = 5306_8$

$= 1010\_1100\_0110_2 = AC6_{16}$

$11011001110101001_2 = o11\_011\_001\_110\_101\_001_2 = 331651_8$

$= ooo1\_1011\_0011\_1010\_1001_2 = 1B3A9_{16}$

Fall 1.2 i ≠ 1, j = 1, d. h. Konvertierung **ins Dualsystem**

Jedes Ziffernsymbol wird durch seine duale Repräsentation gemäß Korrespondenztabelle ersetzt.

Beispiel 2.4:

$1573_8 = 001\_101\_111\_011_2 = 1101111011_2$

$A748_{16} = 1010\_0111\_0100\_1000_2 = 1010011101001000_2$

Fall 1.3 i,j ≠ 1, i ≠ j, **der allgemeine Fall**

Die Quellzahl wird zunächst nach Fall 1.2 in eine Dualzahl und danach nach Fall 1.1 in die Zielzahl umgeschrieben.

Beispiel 2.5:

$3A_{16} = x_4$

$= 0011\_1010_2 = 111010_2 = 11\_10\_10_2 = 322_4$

Fall 2

B = 10, d. h. Konvertierung **ins Dezimalsystem**

Es wird der dezimale Wert nach der o. a. Formel berechnet.

Beispiel 2.6:

$$101100_2 = x_{10} = 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0$$
$$= 32_{10} + 8_{10} + 4_{10}$$
$$= 44_{10}$$

Eine Vereinfachung kann das sog. HORNER-Schema bringen. Wer das HORNER-Schema ohnehin kennt, sollte es anwenden. Lernen sollte er es dafür nicht. Aus der Wertformel (für ganze Zahlen)

$$\textbf{Wert} \quad w = \sum_{p=0}^{m-1} d_p * b^p$$

folgt $w = d_0 + d_1 b + d_2 b^2 + \ldots + d_{m-2} b^{m-2} + d_{m-1} b^{b-1}$

$= d_0 + b(d_1 + \ldots + b(d_{m-2} + b d_{m-1}) \ldots))$

d. h. Ausklammern von b befreit von der expliziten Berechnung der Potenzen von b.

Beispiel 2.7:

$$1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0_{\,2}$$

$$((((1*2 + 0)*2 + 1)*2 + 1)*2 + 0)*2 + 0$$

$$((((\ \ 2\ \ )*2 + 1)*2 + 1)*2 + 0)*2 + 0$$
$$(((\ \ \ \ 5\ \ \ \ )*2 + 1)*2 + 0)*2 + 0$$
$$((\ \ \ \ \ \ 11\ \ \ \ \ \ )*2 + 0)*2 + 0$$
$$(\ \ \ \ \ \ \ \ 22\ \ \ \ \ \ \ \ )*2 + 0$$
$$44_{10}$$

Fall 3

b = 10, d. h. Konvertierung **aus dem Dezimalsystem**

Hier empfiehlt sich die "Methode des scharfen Hinsehens". Die Wertformel liefert wieder den Schlüssel. Es wird von oben nach unten geprüft, wie oft B^p in der Quellzahl "steckt".

Beispiel 2.8:

$179_{10} = x_2$

alle Werte $2^8 = 256$ und größer "stecken" 0-mal in 179.
$2^7 = 128$ "steckt" **1**-mal in 179.
179 - 1*128 = 51.
$2^6 = 64$ "steckt" **0**-mal in 51.
$2^5 = 32$ "steckt" **1**-mal in 51.
51 - 1*32 = 19.
$2^4 = 16$ "steckt" **1**-mal in 19.
19 - 16 = 3.
$2^3 = 8$ "steckt" **0**-mal in 3.
$2^2 = 4$ "steckt" **0**-mal in 3.
$2^1 = 2$ "steckt" **1**-mal in 3.
3 - 2 = 1.
$2^0 = 1$ "steckt" **1**-mal in 1.
1 - 1 = 0. <—— Abbruchkriterium
fertig! ——————> 10110011_2

Ein aus dem HORNER-Schema abgeleitetes Divisionsverfahren kann die Lösung vereinfachen.

Beispiel 2.9:

$179_{10} = x_2$

```
179
 :2 = 89 Rest 1 <—— LSB !
      :2 = 44 Rest 1
           :2 = 22 Rest 0
                :2 = 11 Rest 0
                     :2 =  5 Rest 1
                          :2 =  2 Rest 1
                               :2 =  1 Rest 0
                                    :2 =  0 Rest 1 <——  MSB !
                                          |
                                          └— Abbruchkriterium !
——————> 10110011₂
```

Hinweis: Je größer die Basis des Zielsystems ist, desto schneller konvergiert das Verfahren. Falls das Zielsystem das Dualsystem ist, empfiehlt es sich deshalb, zunächst ins Hexadezimalsystem (oder ins Oktalsystem) zu konvertieren und das so gewonnene Zwischenergebnis nach Fall 1.2 in eine Dualzahl umzuschreiben. Das gilt sowohl für die "Methode des scharfen Hinsehens" als auch für das Divisionsverfahren.

Beispiel 2.10:

```
179
:16 =  11 Rest 3  <—— LSB
      :16 = 0 Rest 11  <—— MSB
            |
            └— Abbruchkriterium
```

——————> $B3_{16} = 1011\_0011_2 = 10110011_2$

Fall 4

Der allgemeine Fall ist praktisch nur lösbar, wenn man über das Dezimalsystem konvertiert, d. h. Fall 2 und Fall 3 kombiniert.

Beispiel 2.11

$2210020_3 = z_5$

$2210020_3 = x_{10} = z_5$

Fall 2 Fall 3

ohne HORNER-Schema:

$\longrightarrow 2210020_3 = 2*3^6 + 2*3^5 + 1*3^4 + 2*3^1$

$= 2*729 + 2*243 + 1*81 + 2*3$

$= 1458 + 486 + 81 + 6 = 2031_{10}$

$\longrightarrow = 3*625 + 1*125 + 1*25 + 1*5 + 1*1$

$= 3111_5$

2.2.2. Konvertierung gebrochener Zahlen

Der ganze Teil und der gebrochene Teil werden getrennt konvertiert, der ganze Teil nach den oben behandelten, der gebrochene Teil nach den unten folgenden Regeln.

Fall 2

B = 10, d. h. Konvertierung **ins Dezimalsystem**

Gegeben sei eine gebrochene Zahl zur Basis b mit m Stellen nach dem Punkt

$$0.d_{-1}d_{-2}...d_{-m_b}$$

Dann erfolgt die Konvertierung in drei Schritten:

1. Verschieben des Punkts um m Stellen nach rechts (d. h. Multiplikation mit b^m)

   $$0.d_{-1}d_{-2}...d_{-m_b} \times b^m = d_{-1}d_{-2}...d_{-m_b}$$

2. Konvertieren der so entstandenen ganzen Zahl in eine wertgleiche Dezimalzahl
3. Division dieser Dezimalzahl durch b^m_{10}

Beispiel 2.12

$0.8A7_{16} = x_{10}$

1. Schritt $0.8A7_{16} \times 16^3 = 8A7_{16}$

2. Schritt $8x16^2 + 10x16 + 7 = 2215_{10}$

3. Schritt $2215 : 4096 = 0.54077..._{10}$

Beispiel 2.13

$0.4247_8 = x_{10}$

1. Schritt $0.4247_8 \times 8^4 = 4247_8$

2. Schritt $4x8^3 + 2x8^2 + 4x8 + 7 = 2215_{10}$

3. Schritt $2215 : 4096 = 0.54077..._{10}$

Fall 3

b = 10, d. h. Konvertierung **aus dem Dezimalsystem**

Auch hier hilft die "Methode des scharfen Hinsehens". Eleganter ist ein aus dem HORNER-Schema abgeleitetes Verfahren.

Beispiel 2.14

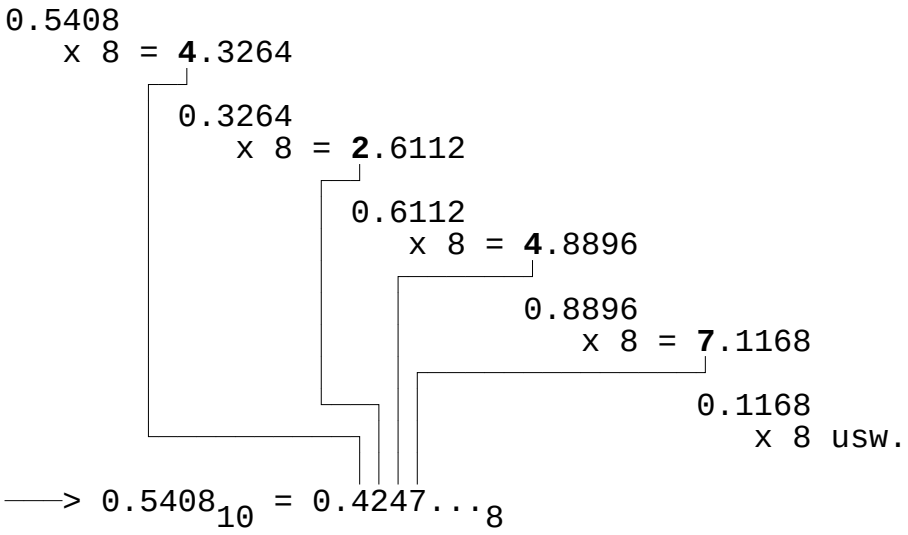
$0.5408_{10} = x_{16}$

```
0.5408
  x 16 = 8.6528
         0.6528
           x 16 = 10.4448
                  0.4448
                    x 16 = 7.1168
                           0.1168
                             x 16 usw.
```

$\longrightarrow 0.5408_{10} = 0.8A7..._{16}$

Beispiel 2.15

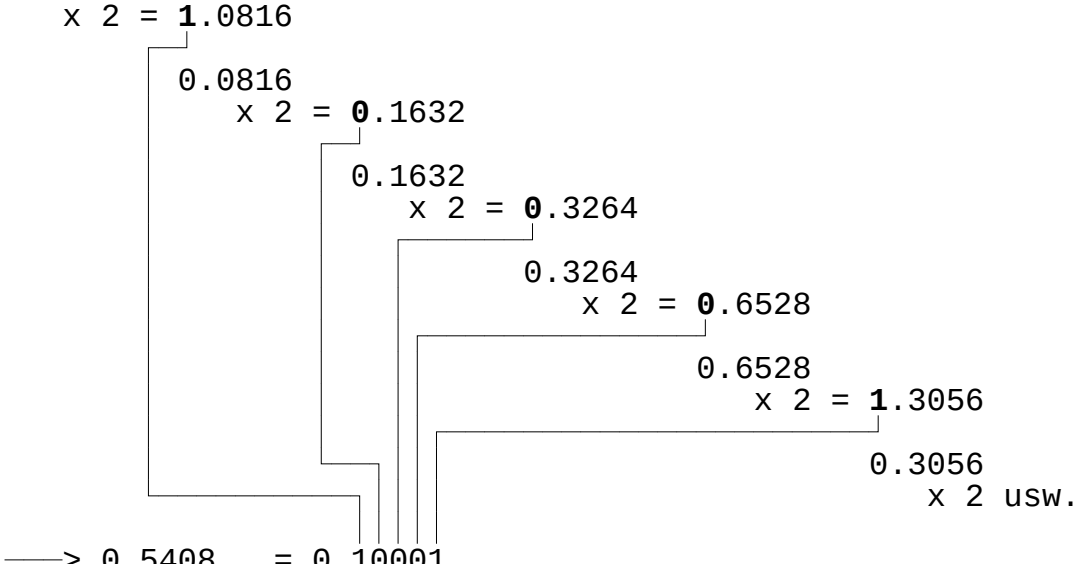
$0.5408_{10} = x_8$



$\longrightarrow 0.5408_{10} = 0.4247..._8$

Beispiel 2.16

$0.5408_{10} = x_2$



$\longrightarrow 0.5408_{10} = 0.10001..._2$

Auch hier führt der "Umweg" über das Hexadezimalsystem oder das Oktalsystem zu erheblichen Einsparungen:

$0.8A7_{16} = 0.1000\_1010\_0111_2$ (vgl. Beispiel 2.14)
$0.4247_8 = 0.100\_010\_100\_111_2$ (vgl. Beispiel 2.15)

2.3. Darstellung negativer Zahlen

Aus der "normalen" Mathematik kennen wir die Darstellung vorzeichenbehafteter Zahlen als Kettung aus

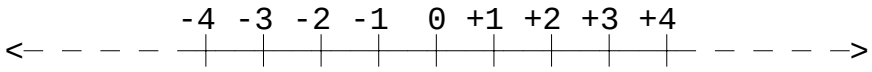
"**Vorzeichen**" und "**Betrag**".

Diese Form der Zahlendarstellung wird in Rechnern eher selten verwendet. Der Hauptgrund dafür ist, daß man bei arithmetischen Operationen (Addition, Subtraktion, ...) mit so dargestellten Zahlen "Vorzeichen" und "Betrag" getrennt voneinander und auf unterschiedliche Weise behandeln muß. Das führt zu erhöhtem Aufwand bei der technischen Realisierung arithmetischer Operationen. Dieser Nachteil und die Tatsache, daß der Darstellungsraum eines Rechners stets endlich ist, hat zur Konsequenz, daß in der **Festkommaarithmetik (s. u.)** fast ausschließlich die sog. **Komplement**darstellung Anwendung findet.

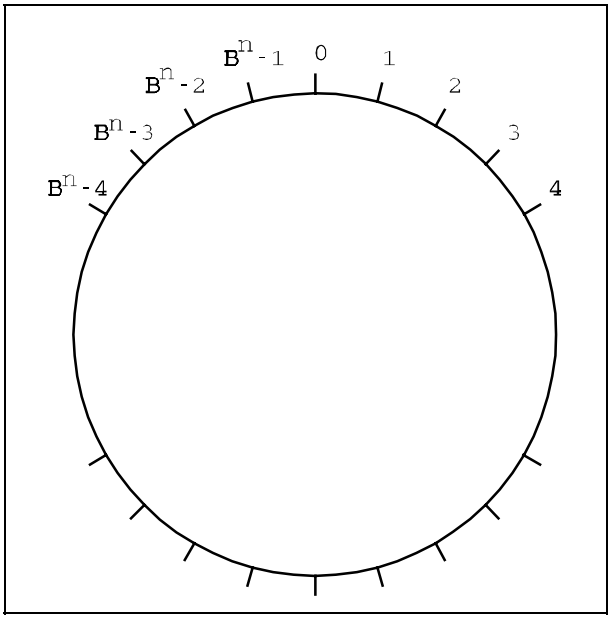
Aus Abschnitt 1 der Vorlesung folgt, daß in einem Positionssystem der Basis B mit einer n-stelligen Zahl genau B^n Zahlen darstellbar sind. Bisher haben wir ausschließlich positive (vorzeichenlose) Zahlen betrachtet. Der im Unterschied zur "normalen" Mathematik endliche Darstellungsraum

$0, 1, \ldots, B^n-1$

legt nahe, anstelle des (nach links und rechts unbegrenzten) **Zahlenstrahls**



den **Zahlenkreis** zur Verdeutlichung der Zusammenhänge heranzuziehen.



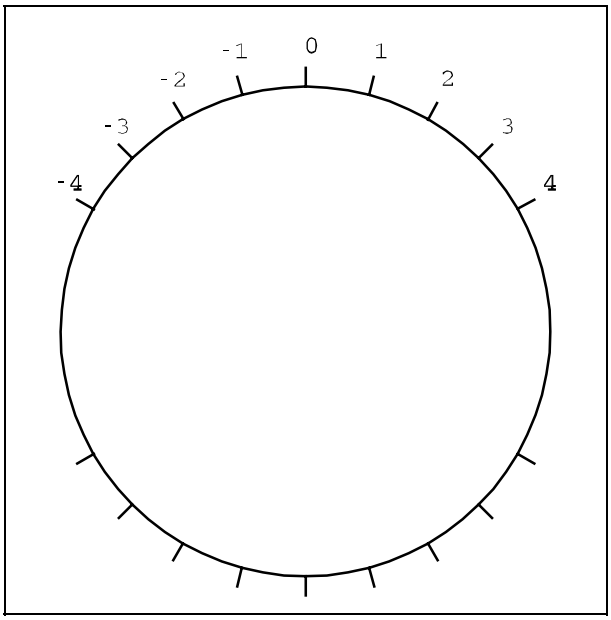
Bei n = 3 folgt z. B. für das Dezimalsystem der Darstellungsraum 000, 001, ... , 999 und für das Binärsystem der Darstellungsraum 000, 001, ... , 111.

Die **Addition** ist im Zahlenkreis als **Bewegung in Uhrzeigerrichtung**, die **Subtraktion** als **Bewegung entgegen der Uhrzeigerrichtung** darstellbar. Zu beachten ist, daß aus der Endlichkeit des Darstellungsraums folgt

$$B^n - 1 \; + \; 1 \; \Rightarrow \; 0.$$

Mathematisch formuliert handelt es sich bei dieser Operation um eine **Addition mod B^n** (s. u.).

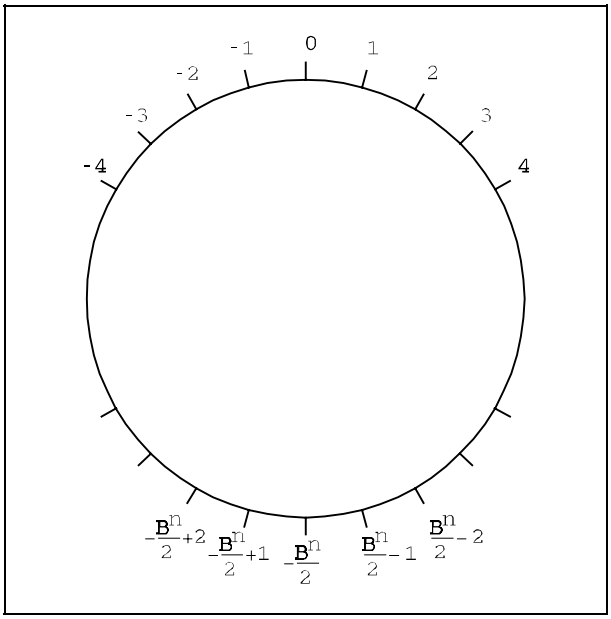
Wenn bei gleicher Mächtigkeit des Darstellungsraums positive **und** negative Zahlen dargestellt werden sollen, dann liegt nahe, den Darstellungraum möglichst symmetrisch auf die positiven und negativen Zahlen aufzuteilen, d. h. man möchte erreichen, daß jede darstellbare Zahl sowohl als positive als auch als negative Zahl darstellbar ist. Man teilt also den Zahlenkreis in zwei (gleichgroße) Hälften:



Interessant ist der untere Scheitelpunkt! In einem Stellenwertsystem mit **gerader Basis B** (nur mit solchen System haben wir es zukünftig zu tun: B = 2, 4, 8, 10, 16) ergibt sich der Darstellungsraum

$$-\frac{B^n}{2},\ -\left(\frac{B^n}{2}-1\right),\ \ldots,\ -1,\ 0,\ +1,\ \ldots,\ +\left(\frac{B^n}{2}-2\right),\ +\left(\frac{B^n}{2}-1\right)$$

$B^n/2$ negative Zahlen | $B^n/2$ positive Zahlen

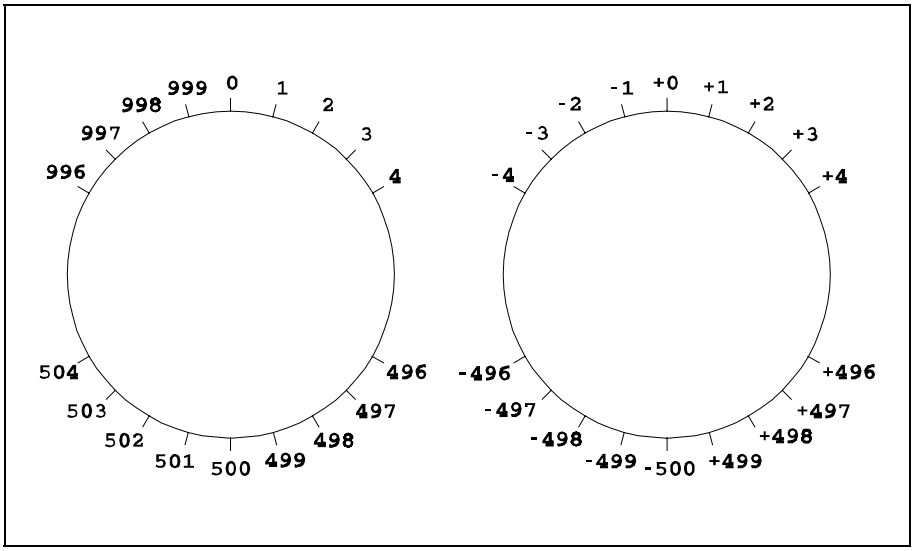


Mit all diesen Überlegungen ist nun immer noch nicht klar geworden, warum man in diesem Zusammenhang von **Komplement**darstellung spricht. Die Sache wird klar, wenn man die beiden Darstellungsräume

$$R_{vorzeichenlos} = \{0, 1, ..., B^n-1\} \quad \text{und}$$

$$R_{vorzeichenbehaftet} = \{-\frac{B^n}{2}, ..., -1, 0, +1, ..., \frac{B^n}{2} -1\}$$

miteinander vergleicht. Für eine 3-stellige Dezimalzahl ergibt sich der Zahlenkreis



d. h. die Zahl -1 wird als 999 dargestellt, die Zahl -2 als 998, die Zahl -499 als 501, die Zahl -500 als 500 usw.

Allgemein könnte man zunächst schreiben

Eine Zahl $-z_1$ wird als eine Zahl $+z_2$ dargestellt.

Beim genaueren Hinschauen sieht man aber, daß stets gilt

$|z_1| + |z_2| = 1000 = 10^3$ oder

$|z_2| = 10^3 - |z_1|$

und es gilt tatsächlich allgemein:

In einem n-stelligen Stellenwertsystem der Basis B wird

die **negative Zahl -z** als **positive Zahl B^n - |-z|**

dargestellt. Damit wird dann auch der Begriff **Komplement** (= Ergänzung zu B^n) klar. Genauer unterscheidet man zwischen B- und B-1-Komplement und versteht unter B-1-Komplement die Differenz $B^n - 1 - |-z|$.

Beiden Komplementen ist eigen, daß - wie bei dem üblichen (unären) Minusoperator - die zweifache Anwendung auf eine Zahl wieder diese Zahl liefert:

$$B^n - (B^n - z) = z$$

$$B^n - 1 - (B^n - 1 - z) = z$$

Das Komplement einer positiven Zahl ist die betragsgleiche negative Zahl, das Komplement einer negativen Zahl ist die betragsgleiche positive Zahl.

Im **Binärsystem** (B = 2) geht die Halbierung $2^n / 2$ über in 2^{n-1}, und die Darstellungsräume nehmen dann die Form

$R_{vorzeichenlos} = \{0, 1, ..., 2^n-1\}$ und

$R_{vorzeichenbeh.} = \{-2^{n-1}, ..., -1, 0, +1, ..., 2^{n-1}-1\}$

an.

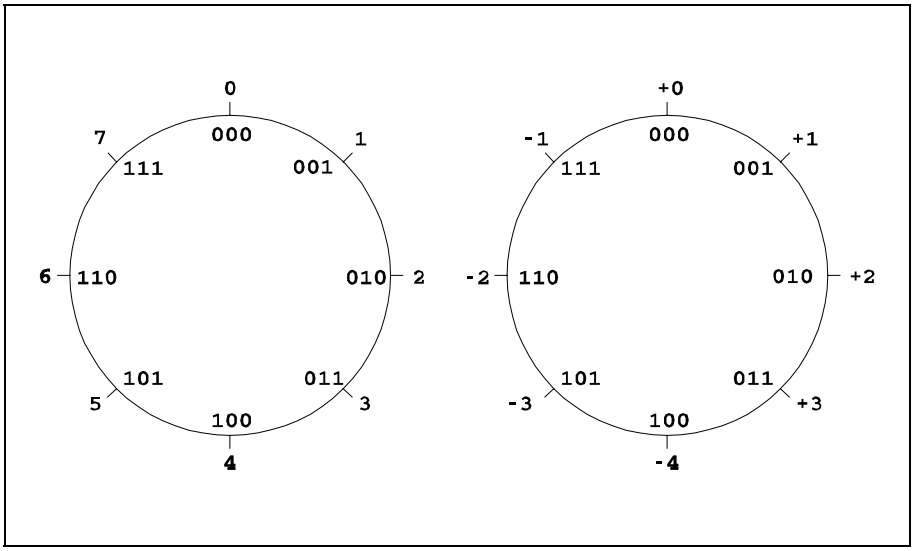
Beispiel 2.17

Mit 3 Bit lassen sich die Werte

$0, ..., 2^3-1 = 0 ... 7$ oder

$-2^2, ..., 2^2-1 = -4 ... +3$

darstellen. Am Zahlenkreis stellt sich das Ergebnis wie folgt dar:



Es ist zu erkennen, daß das höchstwertige Bit (MSB) bei positiven Zahlen stets den Wert 0 und bei negativen Zahlen stets den Wert 1 hat. Das gilt allgemein für vorzeichenbehaftete Binärzahlen!!! **Das MSB ist aber nicht das Vorzeichen**, es hat einen numerischen Wert, wie aus den allgemeinen Betrachtungen zum Komplement deutlich geworden ist.

Für gängige Verarbeitungsbreiten ergibt sich also z. B.

n	vorzeichenlos	vorzeichenbehaftet
8	0 ... 255	-128 ... +127
16	0 ... 65535	-32768 ... +32767

Die häufig anzutreffende, umgekehrte Fragestellung **"Wieviele Bits benötige ich zur Darstellung einer bestimmten Zahl?"** ist also für positive Zahlen ohne eine Aussage dazu, in welchem Darstellungsraum wir uns bewegen, nicht eindeutig beantwortbar.

Beispiel 2.18

Wieviele Bits sind zur Darstellung der Dezimalzahl 6 erforderlich?

Für den Darstellungsraum der vorzeichenlosen Zahlen ergibt sich

$$2^n - 1 \geq 6$$
$$2^n \geq 7$$
$$n \geq \text{ld } 7, \text{ ganz}$$
$$n = 3$$
$$6_{10} = 110_2$$

Für den Darstellungsraum der vorzeichenbehafteten Zahlen folgt

$$2^{n-1} - 1 \geq 6$$
$$2^{n-1} \geq 7$$
$$n - 1 \geq \text{ld } 7, \text{ ganz}$$
$$n \geq \text{ld } 7 + 1, \text{ ganz}$$
$$n = 4$$
$$6_{10} = 0110_2$$

Um Ihnen den Begriff des Komplements nahezubringen, habe ich alle Berechnungen im Dezimalsystem ausgeführt. Praktisch bildet man das Einer- und das Zweierkomplement von Binärzahlen aber meist binär. Zwei Verfahren sind gleichermaßen gut geeignet:

Verfahren 1:
- Bitweises Negieren der Quellzahl
- Addition von 1 in der niederwertigsten Stelle

Verfahren 2:
- von rechts beginnend alle Bits bis einschließlich der ersten 1 unverändert übernehmen
- alle links von der ersten 1 liegenden Bits negiert übernehmen

Beispiel 2.19

Gegeben sei die Darstellung der Zahl + 62 im 8-Bit-Format, Gesucht die Darstellung der Zahl - 62 im gleichen Format.

Verfahren 1

Bitweises Negieren und Addition von 1 in der niederwertigsten Stelle.

```
  00111110   + 62
  --------
  11000001          bitweise Negation
+        1   + 1    Addition von 1 in der niederwertigsten Stelle
--------------
  11000010   - 62
==========
```

Die bitweise Negation liefert zunächst $11000001_2 = C1_{16} = 193_{10}$. $193_{10} = 255_{10} - 62_{10}$ ist das sog. Einerkomplement, d. h. das Komplement zu $2^8 - 1 = 255$. Die anschließende Addition von 1 in der niederwertigsten Stelle liefert dann $11000010_2 = C2_{16} = 194_{10}$, wobei $194_{10} = 256_{10} - 62_{10}$ das Zweierkomplement ist, d. h. das Komplement zu $2^8 = 256$.

Verfahren 2

Von rechts nach links jede Binärziffer bis einschließlich der ersten gefundenen 1 nichtnegiert und danach jede Binärziffer negiert übernehmen.

```
<-------------
0 0 1 1 1 1 1 0   + 62
| | | | | | | |
v v v v v v v v
1 1 0 0 0 0 1 0
===============
```

Wir wollen uns noch davon überzeugen, daß die erneute Anwendung dieser Verfahren auf das Ergebnis wieder die Zahl + 62 liefert:

Verfahren 1

```
  11000010   - 62
  --------
  00111101
+        1   + 1
--------------
  00111110   + 62
==========
```

Verfahren 2

```
1 1 0 0 0 0 1 0   - 62
| | | | | | | |
v v v v v v v v
0 0 1 1 1 1 1 0   + 62
```

2.4. Festkommarithmetik

Festkommadarstellung heißt zunächst: **Das Komma steht immer an der gleichen Stelle**. Wir vereinbaren hier aber einschränkend, daß wir unter Festkommadarstellung eine m-stellige **ganze** Binärzahl verstehen wollen:

$$b = 2,\ d_p \in \{0,1\},\ p \in \{0,1,...,m-2,m-1\},\ z = d_{m-1}d_{m-2}...d_1d_0.$$

d. h. rechnerintern arbeiten wir ausschließlich mit ganzen Zahlen. (Konsequenz für die Vearbeitung nichtganzer Zahlen? ⟶ Sache des Programmierers!)

Wegen der Endlichkeit des Darstellungsraums verdeutlichen wir uns die Verhältnisse wieder mit Hilfe des Zahlenkreises.

2.4.1. Addition

Für die Addition zweier einstelliger Binärzahlen x und y zur Summe s gilt

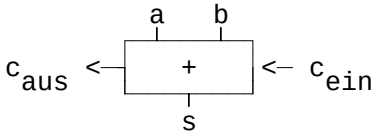
x + y	c	s
0 + 0 =	0	
0 + 1 =	1	
1 + 0 =	1	
1 + 1 = 1	0	(1 + 1 = 0 merke 1)

Da die Summe drei unterschiedliche Werte annehmen kann, sind zu ihrer Darstellung 2 Bits nötig. Da Bit mit dem Gewicht 2^1 ist in Analogie zur dezimalen schriftlichen Addition der **Übertrag (Carry) in die nächsthöhere Stelle** ("auslaufender" Übertrag).

Im allgemeinen Fall kann aber auch ein **Übertrag aus der nächstniedrigen Stelle** auftreten ("einlaufender" Übertrag), so daß die Addition in einer Binärstelle vollständig so zu beschreiben ist:

a	b	c_{ein}		c_{aus}	s
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

Vorschau auf den RC-Adder:



Mehrstellige Binärzahlen werden analog zur dezimalen schriftlichen Addition addiert. Wir wollen zunächst den Fall der vorzeichenlosen 4-stelligen Binärzahlen betrachten. Der Darstellungsraum ergibt sich zu R = {0,1,...,14,15}.

Beispiel 2.20 Addition vorzeichenloser Binärzahlen

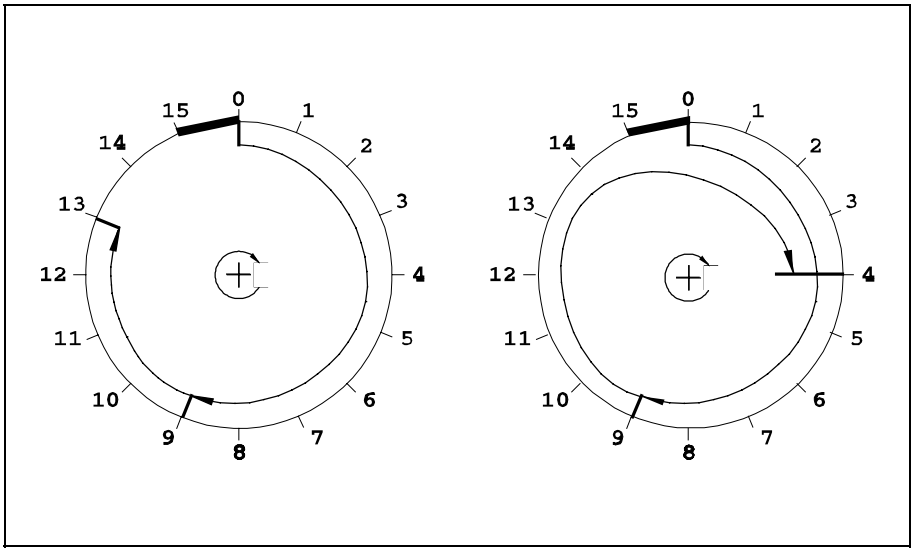
```
    1 0 0 1       9            1 0 0 1       9
+   0 1 0 0     + 4        +   1 0 1 1     + 11
+ 0 0 0 0 - (Ü)              + 1 0 1 1 - (Ü)
  ---------   ----           ---------   -----
  1 1 0 1      13            0 1 0 0       4
  =========   ====           =========   =====
    |                          |
    └──> kein Übertrag, korrekt  └──> Übertrag, falsch
```

Mathematisch korrekt wird hier eine **Addition MODULO 16** ausgeführt. Man versteht darunter den ganzzahligen Rest nach ganzzahliger Division durch 16

(9 + 4) MOD 16 = 13 MOD 16 => 0 Rest **13**
(9 + 11) MOD 16 = 20 MOD 16 => 1 Rest **4**

Anm.: Für m-stellige ganze Binärzahlen gilt allgemein **Addition MODULO 2^m**.

Falsch wird ein Ergebnis immer dann, wenn bei Addition (oder Subtraktion) der Übergang von 2^m-1 nach 0 (oder von 0 nach 2^m-1) überschritten wird. **Am auslaufenden Übertrag ist erkennbar, daß das Ergebnis falsch ist**.



Wir wollen nun den Fall der vorzeichenbehafteten 4-stelligen Binärzahlen beleuchten. Der Darstellungsraum ergibt sich zu R = {-8,-7,...,-1,0,+1,...,+6,+7}

Beispiel 2.21 Addition vorzeichenbehafteter Binärzahlen

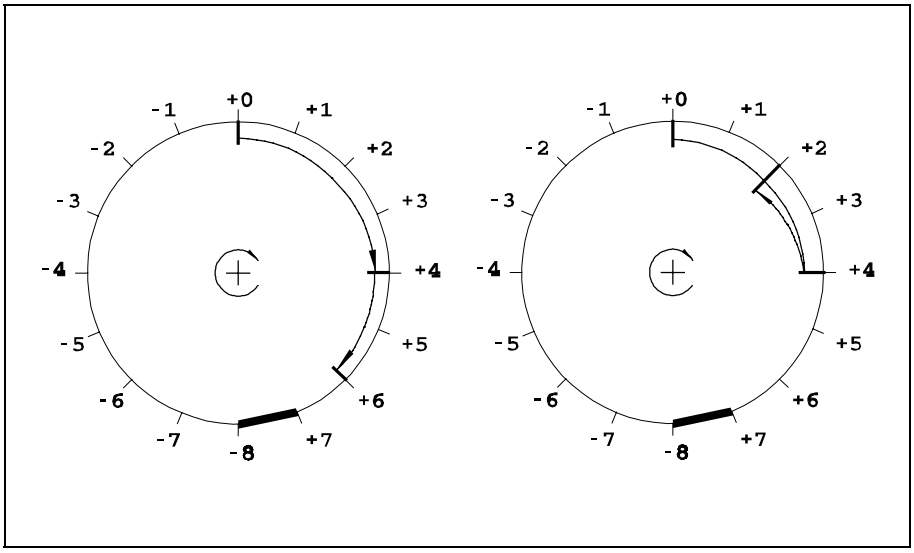
Negative Zahlen werden in Form des 2er-Komplements der betragsgleichen positiven Zahl dargestellt. Die Addition erfolgt in der gleichen Weise, wie für vorzeichenlose Binärzahlen gezeigt. Der Vorteil des Zweierkomplements ist es, daß das MSB keiner Sonderbehandlung unterzogen werden muß (s. oben; das MSB ist nicht das Vorzeichen, es hat einen numerischen Wert!).

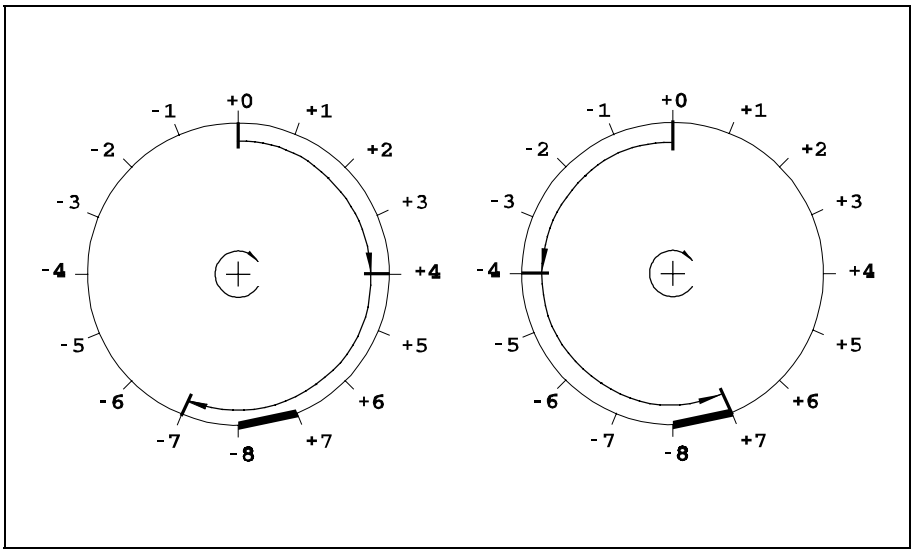
```
    0 1 0 0       (+4)
+   0 0 1 0     + (+2)
+ 0 0 1 1 - (Ü)
  ┌─┘
  │ ─────────   ──────
  │ 0 1 1 0       (+6)
  │ ═════════   ══════
  └──> beide Überträge gleich,
       kein Überlauf, korrekt
```

```
    0 1 0 0       (+4)
+   1 1 1 0     + (-2)
+ 1 1 0 1 - (Ü)
  ┌─┘
  │ ─────────   ──────
  │ 0 0 1 0       (+2)
  │ ═════════   ══════
  └──> beide Überträge gleich,
       kein Überlauf, korrekt
```

```
    0 1 0 0       (+4)
+   0 1 0 1     + (+5)
  0 1 0 0 - (Ü)
  ┌─┘
  │ ─────────   ──────
  │ 1 0 0 1       (-7)
  │ ═════════   ══════
  └──> Überträge ungleich,
       Überlauf, falsch
```

```
    1 1 0 0       (-4)
+   1 0 1 1     + (-5)
  1 0 0 0 - (Ü)
  ┌─┘
  │ ─────────   ──────
  │ 0 1 1 1       (+7)
  │ ═════════   ══════
  └──> Überträge ungleich,
       Überlauf, falsch
```





Wann ist das Ergebnis falsch? Woran ist erkennbar, daß das Ergebnis falsch ist? Am **Übertrag (Carry)** offensichtlich nicht!

Es gibt 4 Fälle, bei denen das Ergebnis falsch wird:

	a	b	c
Addition a + b = c	positiv	positiv	negativ
	negativ	negativ	positiv
Subtraktion a - b = c	positiv	negativ	negativ
	negativ	positiv	positiv

Falsch wird ein Ergebnis immer dann, wenn bei Addition (oder Subtraktion) der Übergang $+2^{m-1}-1$ nach -2^{m-1} (oder von -2^{m-1} nach $+2^{m-1}-1$ überschritten wird. Zur Erkennung dieser Fälle wird ein spezielles Signal - **Überlauf (Overflow)** - gebildet. Die Bildungsvorschrift kann aus der o. a. Tabelle abgeleitet werden.

$$OF = ADD\ (/a_{m-1}\ /b_{m-1}\ c_{m-1} \vee a_{m-1}\ b_{m-1}\ /c_{m-1}) \vee$$
$$SUB\ (/a_{m-1}\ b_{m-1}\ c_{m-1} \vee a_{m-1}\ /b_{m-1}\ /c_{m-1})$$

Einfacher ist die Auswertung der beiden höchstwertigen Übertragsbits, **des Übertrags in die höchstwertige Stelle hinein** und **des Übertrags aus der höchstwertigen Stelle heraus**. Ein Überlauf tritt immer dann auf, wenn diese beiden Übertragsbits ungleich sind (unabhängig davon, ob addiert oder subtrahiert wird:

$$OF = \overline{ü_{m-1}}\ ü_m \vee ü_{m-1}\ \overline{ü_m}$$

2.4.2. Subtraktion

Die Subtraktion wird üblicherweise auf die Addition des 2er-Komplements zurückgeführt. Auf die Demonstration anhand des Zahlenkreises verzichte ich hier.

Beispiel 2.22 Subtraktion vorzeichenloser Binärzahlen

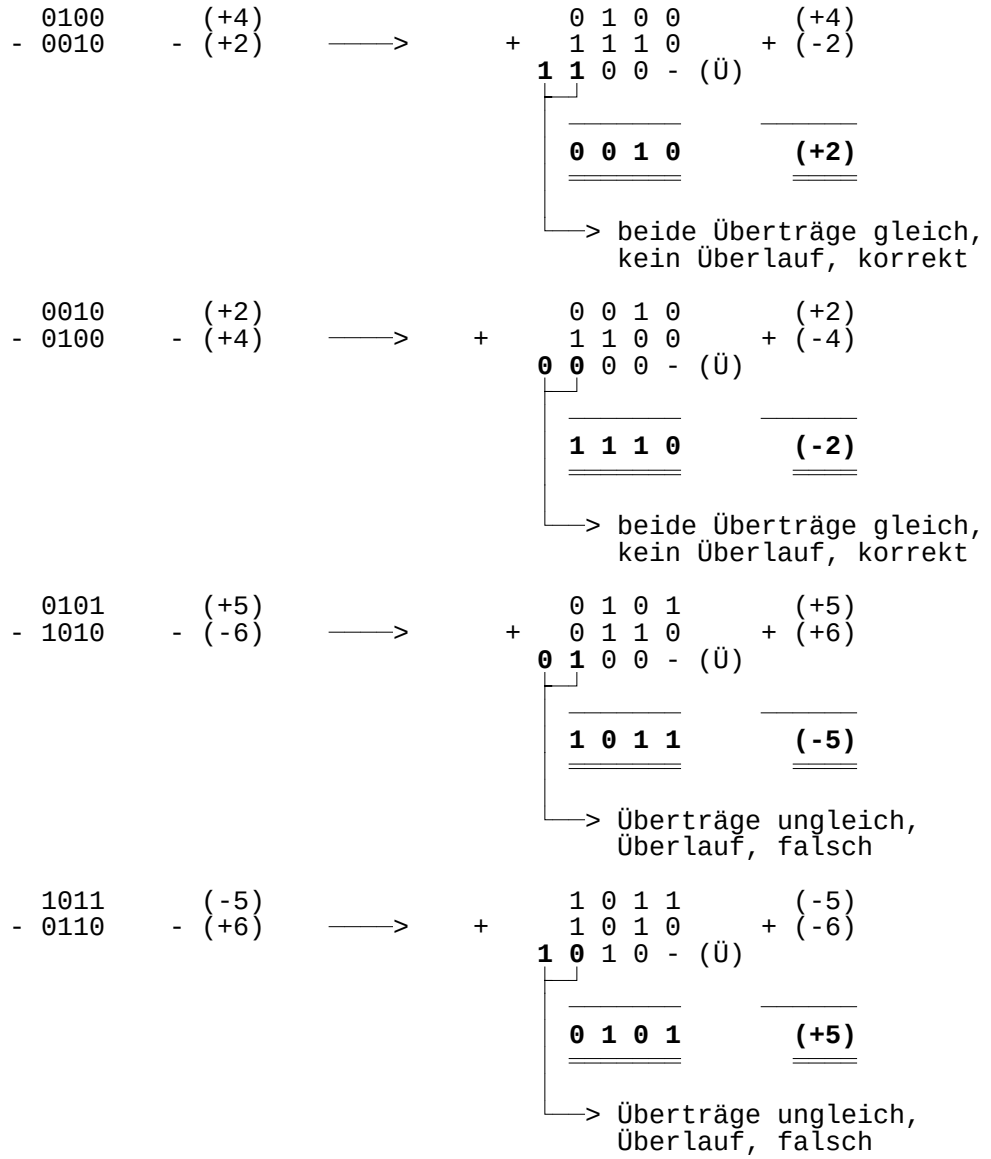
```
  0100      (+4)                 0 1 0 0       (+4)
- 0010    - (+2)    ------>   +   1 1 1 0     + (-2)
                              + 1 1 0 0 - (Ü)
                                ---------     -------
                                0 0 1 0       (+2)
                                =========
                                |
                                +------> Übertrag, korrekt

  0010      (+2)                 0 0 1 0       (+2)
- 0100    - (+4)    ------>   +   1 1 0 0     + (-4)
                              0 0 0 0 - (Ü)
                                ---------     -------
                                1 1 1 0       (+14)
                                |
                                +------> kein Übertrag, falsch
```

Bei der Subtraktion vorzeichenloser Zahlen ist ein falsches Ergebnis daran erkennbar, daß **kein auslaufender Übertrag** auftritt!

Anm.: Diese Aussage kann zu Mißverständnissen führen. Bei üblichen Prozessoren ist nach einer arithmetischen Operation (Addition, Subtraktion) am sog. Carry-Flag (CF) der Wert des Übertrags ablesbar. Um unabhängig von der Operation aus CF = 1 auf

ein falsches Ergebnis schließen zu können, wird im Falle der Addition der "echte" (nichtnegierte) Wert des Übertrags und im Falle der Subtraktion der negierte Wert des Übertrags in das Carry-Flag eingetragen!

Beispiel 2.23 Subtraktion vorzeichenbehafteter Binärzahlen

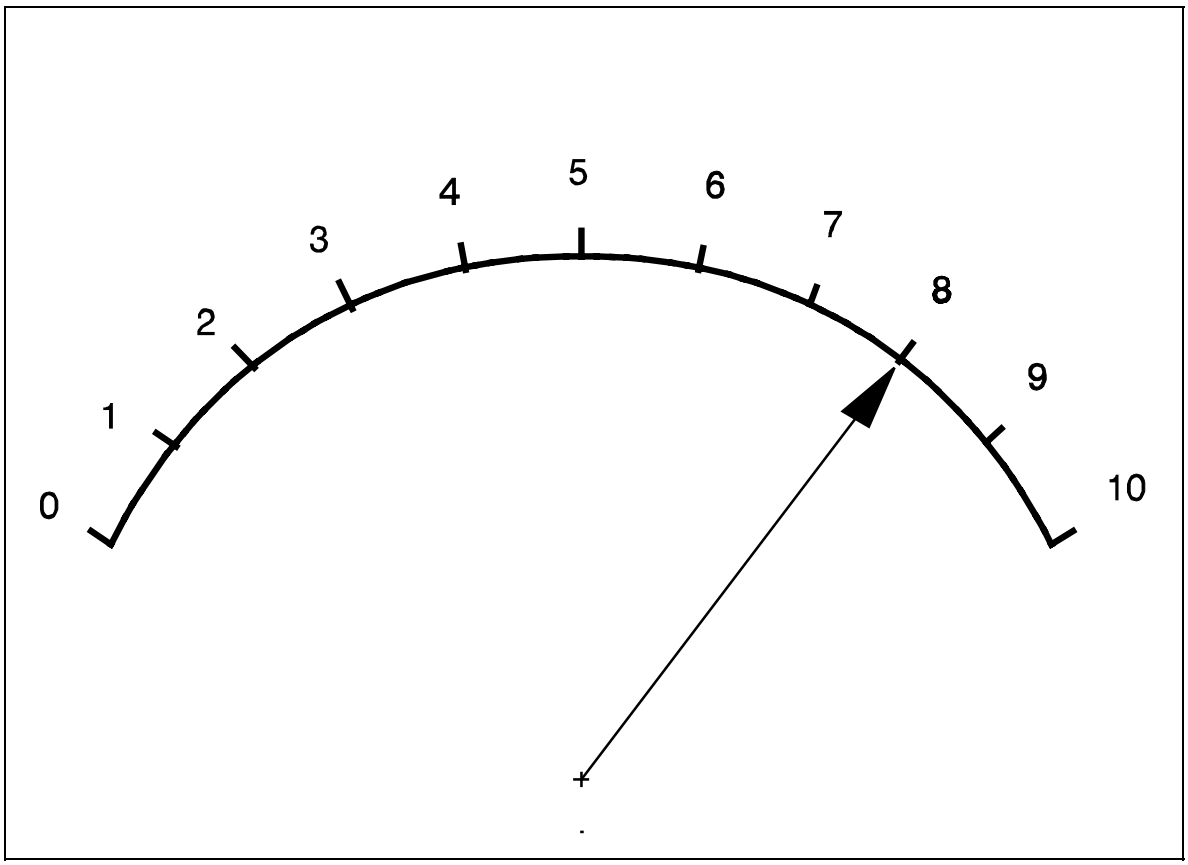


Für die Subtraktion vorzeichenbehafteter Binärzahlen gilt wieder - wie bei der Addition - daß das Ergebnis immer dann falsch wird, wenn der untere Scheitel des Zahlenkreises übersprungen wird, d. h. wenn ein **Überlauf (Overflow)** auftritt. Erkennbar ist das Auftreten eines Überlaufs daran, daß - wie bei der Addition - die beiden höchstwertigen Übertragsbits ungleich sind.

2.5. Gleitkommaarithmetik

Bei vorgegebener Länge m des Kodeworts (der rechnerinternen Darstellung) lassen sich nur 2^m verschiedene Werte darstellen. In der Mehrzahl der Fälle ist das zu wenig. Ein Ausweg ist die Vergrößerung von m (vgl. 8-, 16-, 32- 64-Bit-Prozessoren und die entsprechenden Datentypen höherer Programmiersprachen, z. B. short_integer, integer, long_integer, ...).

Es gibt aber noch einen anderen Ausweg aus dem Dilemma. Sie hatten bestimmt schon mal ein **analoges** Voltmeter (Zeigerinstrument) mit mehreren Meßbereichen in der Hand. Ein Charakteristikum eines solchen Meßinstruments ist, daß mit ein und derselben Skala alle Meßbereiche überdeckt werden.



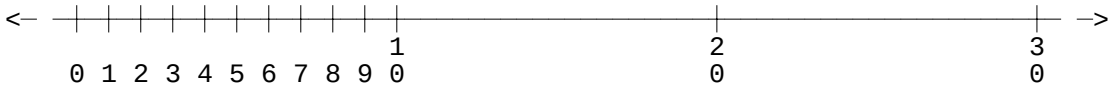
Nehmen wir weiter an, daß wir beim Ablesen **nicht interpolieren sondern** immer zum nächstgelegenen Skalenwert hin **runden** (digitalisieren!), dann ergeben sich die möglichen Meßwerte

im Meßbereich 10 V: {0, 1, 2, ..., 10}
im Meßbereich 100 V: {0, 10, 20, ..., 100}
im Meßbereich 1000 V: {0, 100, 200, ...,1000}
usw.

> Anm.: Da der Meßfehler bei analogen Meßinstrumenten in Prozent des Endausschlags angegeben wird, ist der absolute Meßfehler immer dann besonders hoch, wenn der Meßwert ins erste Drittel der Skala fällt. Deshalb verwendet man in der Praxis meist zwei Skalen und staffelt die Meßbereiche 1 V, 3 V (3,16

V), 10 V, 30 V (31,6 V), ... oder auch drei Skalen mit den Meßbereichen 1V, 2 V (2,16 V), 5 V (4,66 V), 10 V, 20 V, 50 V, ... Die Staffelung leitet sich aus der Überlegung ab, daß der Quotient der Endausschläge benachbarter Meßbereiche (näherungsweise) konstant sein soll. Für die Musiker untere Ihnen: Bekanntermaßen zerfällt eine Oktave in 12 Halbtonschritte. Das Frequenzverhältnis einer Oktave ist konstant 2 : 1 (z. B. a'' = 880 Hz, a' = 440 Hz). Das Frequenzverhältnis eines Halbtonschritts ist ebenfalls konstant und zwar 12. Wurzel aus 2 : 1.

Auf dem Zahlenstrahl nimmt sich das also so aus:



Wenn wir uns bei der rechnerinternen Zahlendarstellung von diesen Überlegungen leiten lassen, dann dann würde das heißen, daß je größer die Zahl ist, desto schlechter darf die nötige "Auflösung" sein: Die Lösung heißt **Gleitkommadarstellung**.

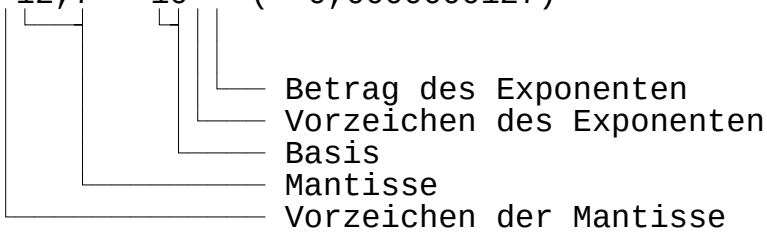
Jede Zahl z kann in der Form

$z = m * b^e$ mit m = Mantisse, b = Basis, e = Exponent

dargestellt werden.

Beispiel 2.24

Dezimalsystem $+12,7 * 10^{-9}$ (= 0,0000000127)



Binärsystem $-1.011 * 2^{+101} = -101100$ (= 44)

Aus diesen Beispielen ist erkennbar:

- Lage des Punktes und Größe des Exponenten stehen miteinander in einem bestimmten Zusammenhang
- Positiver (negativer) Exponent heißt, Mantissenpunkt um Exponent Stellen nach rechts (links) verschieben (falls Mantisse und Exponent im gleichen Zahlensystem beschrieben sind)

Für die rechnerinterne Darstellung steht nun wieder das Problem der Endlichkeit des Darstellungsraums, sowohl Exponent als auch Mantisse haben eine feste Länge.

Ziel: **Jede Zahl so genau wie möglich darstellen**
(d. h. den kleinstmöglichen Meßbereich wählen, der die Zahl gerade noch erfaßt)

Bei fester Mantissenlänge ist die Genauigkeit der Zahlendarstellung dann am größten, wenn die höchstwertige Mantissenziffer nicht den Wert Null hat.

Beispiel 2.25: Zur Genauigkeit der Zahlendarstellung

Des besseren Verständnisses wegen verwende ich das Dezimalsystem. Es soll die Zahl 123,45 so genau wie möglich dargestellt werden. Die Mantisse habe 4 Stellen der Exponent 1 Stelle. Mögliche Darstellungsformen sind:

Komma steht rechts von der niederwertigsten Mantissenziffer (ganzzahlige Mantisse)

$0001 * 10^{2} = 100$
$0012 * 10^{1} = 120$
$0123 * 10^{0} = 123$
$\mathbf{1}234 * 10^{-1} = 123,4$ **<——— so genau wie möglich**

oder Komma steht links von der höchstwertigen Mantissenziffer (gebrochene Mantisse)

$[0.]0001 * 10^{6} = 100$
$[0.]0012 * 10^{5} = 120$
$[0.]0123 * 10^{4} = 123$
$[0.]\mathbf{1}234 * 10^{3} = 123,4$ **<——— so genau wie möglich**

Die Hundertstelstelle ist in diesem Format überhaupt nicht darstellbar.

Die zuletzt angegebene Darstellung nennt man **normalisiert.**

Def.: Normalisiert ist eine Gleitkommazahl dann, wenn der Punkt links vor der höchstwertigen Mantissenfiffer (MSD) steht und die höchstwertige Mantissenziffer nicht den Wert 0 hat.

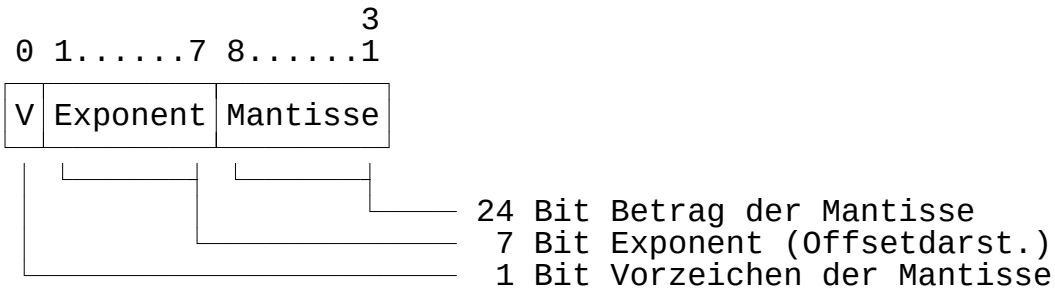
Beispiel 2.26 Normalisierte Darstellung

$+.127 * 10^{-7}$ $-.1011 * 2^{+1001}$

Die Mantisse ist also stets kleiner als Eins!

Für die rechnerinterne Darstellung verwendet man abweichend von der Festkommadarstellung zur Darstellung negativer Zahlen nicht das 2er-Komplement.

Beispiel 2.27 IBM Mainframe (360, 370, 3090 usw.); Datentyp SHORT REAL



Die **Basis** hat hier den Wert 16.

Der Wertebereich des **Exponenten** beträgt

$\{-2^6, ...,+2^6-1\}$ (= {-64, ...,+63}.

Kodiert wird aber im Feld Exponent die Summe

Exponent' = Exponent + 64,

d. h. der Wertebereich

$\{0, ..., +2^7-1\} = \{0, ..., 127\}$.

Man nennt diese Darstellung **Offsetdarstellung**. Für die Exponentenlänge n Bit gilt allgemein:

$\{-2^{n-1}, ..., +2^{n-1}-1\} \longrightarrow \{0, ..., 2^n-1\}$

mit dem Offset 2^{n-1}.

Die **Mantisse** wird **hexadezimal** aufgefaßt. D. h. von links beginnend werden jeweils 4 Bit zu einer Tetrade zusammengefaßt. Die Zahl ist dann normalisiert, wenn die linke Tetrade nicht den Wert 0000_2 hat.

Was leistet diese Darstellung?

Mit b_i als dem i-ten Bit, i = 0,1,...,31, kann man schreiben

$$z = (-1)^{b_0} * (b_8*2^{-1} + ... + b_{31}*2^{-24}) * 16^{(b_1*2^6 + ... + b_7*2^0 - 2^6)}$$

$(b_8*2^{-1} + ... + b_{31}*2^{-24})$: Mantisse, normalisiert

$(b_1*2^6 + ... + b_7*2^0 - 2^6)$: Exponent in Offsetdarstellung - Offset

b_0: Vorzeichen der Mantisse (0 pos., 1 neg.)

Daraus kann der Darstellungsraum berechnet werden.

Minimum des Betrags:

$$.0001\_0000\_0000\_0000\_0000\_0000 * 16^{0000000 - 64}$$

$$= 2^{-4} * 16^{-64} = 16^{-1} * 16^{-64} = 16^{-65}$$

Maximum des Betrags:

$.1111\_1111\_1111\_1111\_1111\_1111 = 16^{1111111 - 64}$

wenn man in der niederwertigsten Stelle eine 1 (also 2^{-24} oder 16^{-6}) addiert, dann ergibt sich die Summe zu 1, d. h. die größtmögliche Mantisse hat den Wert $1 - 2^{-24}$ oder $1 - 16^{-6}$.

$= (1 - 16^{-6}) * 16^{63} \approx 16^{63}$

Betrag:

$16^{-65} <= z <= 16^{63}$ oder $5,4 * 10^{-79} <= z <= 7,2 * 10^{75}$

Darstellunsraum:

$- 7,2 * 10^{75} <= z <= + 7,2 * 10^{75}$

Arithmetische Operationen müssen den üblichen Potenzgesetzen gehorchen:

Multiplikation: $\mathbf{m_1 * b^{e1} * m_2 * b^{e2} = m_1 * m_2 * b^{(e1+e2)}}$

Division: $\mathbf{m_1 * b^{e1} / m_2 * b^{e2} = (m_1 / m_2) * b^{(e1-e2)}}$

Für die Strichoperationen müssen zunächst die Exponenten angeglichen werden. Wir nehmen an, daß bei $e_2 \neq e_1$ der Exponent e_2 an e_1 angeglichen wird:

$e_2 \longrightarrow e_2 + (e_1 - e_2) \longrightarrow e_1$

Die Angleichung darf aber den Wert der Gleitkommazahl nicht verändern, d. h. die Mantisse m_2 muß entsprechend korrigiert werden

$m_2 \longrightarrow m_2'$

Die Addition von $e_1 - e_2$ im Exponenten entspricht einer Multiplikation der Mantisse mit

$b^{(e1-e2)}$,

die - um den Wert der Gleitkommazahl nicht zu verändern - durch Verschieben des Punktes in der Mantisse um $e_1 - e_2$ Stellen nach links ($e_1 \geq e_2$) bzw. um $e_2 - e_1$ Stellen nach links ($e_1 < e_2$) kompensiert werden muß.

Addition: $\mathbf{m_1 * b^{e1} + m_2 * b^{e2} \longrightarrow m_1 * b^{e1} + m_2' * b^{e1} = (m_1 + m_2') * b^{e1}}$

Subtraktion: $\mathbf{m_1 * b^{e1} - m_2 * b^{e2} \longrightarrow m_1 * b^{e1} - m_2' * b^{e1} = (m_1 - m_2') * b^{e1}}$

Anschließend ist in jedem Falle zu normalisieren.

3. Kodierung

Wir wollen Kodierung nicht als Verschlüsselung zum Zwecke der Geheimhaltung auffassen, sondern als Mittel zur Darstellung von Sachverhalten so, daß eine Rechner mit diesen Sachverhalten umgehen kann. D. h. Kodierung wird für uns wohl meist Kodierung im Binärsystem sein. In diesem Sinne ist alles, was wir bisher unter dem Gliederungspunkt 1.1. Zahlendarstellung besprochen haben, Kodierung und Umgang mit kodierten Zahlen. Die Kodierungstheorie ist eine spezielle Sparte der Theoretischen Informatik und wird dort ausführlich behandelt. Wir wollen hier nur noch einige Zusammenhänge und Fakten nachtragen.

$X = \{x_1, x_2, ...\}$ sei ein **Symbolalphabet** eines Kodes. In diesem Kode sind **$card(X) = |X|$** Sachverhalte darstellbar

Beispiel 3.1:

Im Dezimalsystem ist $D = \{0,1,2,3,4,5,6,7,8,9\}$, im Binärsystem $B = \{0,1\}$ die Menge der Ziffern. Es sind $|D| = 10$ bzw. $|B| = 2$ Sachverhalte darstellbar.

$W_k = \underbrace{X^k = X \times X \times \ldots \times X}_{\text{k-faches Kreuzprodukt}} = \{w_1, w_2, ...\}$ sei die Menge der

k-stelligen Wörter $w_i = x_1 x_2 ... x_k$, die mit diesem Symbolalphabet bildbar sind. Es sind

$$|W| = |X|^k$$

k-stellige Wörter bildbar und damit $|X|^k$ Sachverhalte darstellbar.

Beispiel 3.2:

Mit einer 3-stelligen Dezimalzahl sind 1000 unterschiedliche Zahlen darstellbar:

$X = D = \{0,1,2,3,4,5,6,7,8,9\}$, $|D| = 10$

$W = D^3 = D \times D \times D = \{000, 001, 002, ..., 999\}$, $|W| = 1000$

Mit einer 16-stelligen Binärzahl sind $2^{16} = 65536$ Sachverhalte darstellbar.

Wenn eine bestimmte Anzahl Sachverhalte dargestellt (kodiert) werden soll, hat man also die Wahl, entweder das Symbolalphabet entsprechend groß zu machen oder hinreichend lange Kodewörter zu bilden. Da wir es in der Regel mit binären Digitalrechnern zu tun haben, ist mit $B = \{0,1\}$ das Symbolalphabet fest, und wir haben nur die Möglichkeit die Länge der binären Kodewörter zu beeinflussen. Aus der Anzahl der darzustellenden Sachverhalte ist die Länge des Kodewortes berechenbar:

Aus $|W| = |X|^k$ folgt allgemein

$$k >= \frac{\log_b |W|}{\log_b |X|}$$

Hier ist die Basis b des Logarithmus zunächst noch beliebig. Mit $b = |X|$ und $\log_{|X|} |X| = 1$ geht die Gleichung über in

$$k >= \log_{|X|} |W|.$$

Beispiel 3.3:

Für das Dezimalsystem gilt k >= lg |W|, für das Binärsystem gilt k >= ld |W| (logarithmus dualis).

Beispiel 3.4:

Wieviele Bits benötigen wir für die binäre Kodierung des deutschen Alphabets?

W = {A,...,Z,Ä,Ö,Ü,a,...,z,ä,ö,ü,ß}

|W| = 26 + 3 + 26 + 4 = 59

k >= ld 59

59 liegt zwischen $32 = 2_5$ und 64 2^6, d. h. man benötigt mindestens 6 Bits.

Im Vorlesungsskript Prof. Monjaus finden Sie eine ganze Reihe unterschiedlicher Kodes. Auch in den Übungen werden verschiedene Kodes behandelt. Diese Kodes eignen Sie sich bitte im Selbststudium an. Von einiger Bedeutung sind:

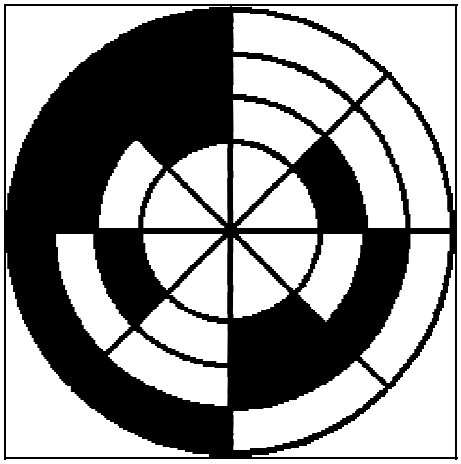
Gray-Kode

dezimal	Binärkode	Gray-Kode
	b_2 b_1 b_0	g_2 g_1 g_0
0	0 0 0	0 0 0
1	0 0 1	0 0 1
2	0 1 0	0 1 1
3	0 1 1	0 1 0
4	1 0 0	1 1 0
5	1 0 1	1 1 1
6	1 1 0	1 0 1
7	1 1 1	1 0 0
0	0 0 0	0 0 0

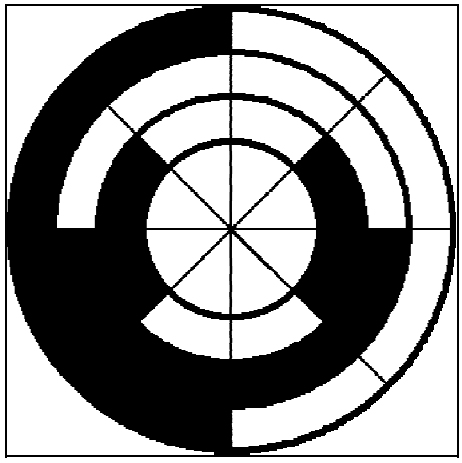
Der Sinn des Gray-Kodes erschließt sich erst, wenn zusätzlich zur bloßen Zuordnung Sachverhalt <——> Kodewort eine feste Reihenfolge der Kodewörter vereinbart ist. Aufeinanderfolgende Kodewörter unterscheiden sich beim Gray-Kode jeweils nur in einem einzigen Bit. Für den Gray-Kode kann ein allgemeines Bildungsgesetz angegeben werden (Vorgriff auf die BBOLEsche Algebra):

$$g_i = b_{i+1} \not+ b_i \; ; \; i = 0,...,n-1; \; b_n = 0$$

Wir werden den Gray-Kode später im sog. KARNAUGH-Plan bei der Logikminimierung verwenden. Eine technische Anwendung des Gray-Kodes ist die Kodierscheibe für die Winkelkodierung:



Kodierscheibe im Binärkode



Kodierscheibe im Gray-Kode

Worin besteht der Vorteil der Kodierscheibe im Gray-Kode gegenüber der Kodierscheibe im Binärkode?

m-aus-n-Kode

n = Länge der Kodewörter
m = Anzahl der Einsen pro Kodewort

Alle n-stelligen Kodewörter mit m Einsen sind gültige Kodewörter

Beispiele:

1-aus-4-Kode: W = {0001,0010,0100,1000}
2-aus-3-Kode: W = {011,101,110}

m-aus-n-Kodes haben eine Bedeutung als fehlererkennende Kodes. Der 1-aus-n-Kode wird uns später beim Logikentwurf noch begegnen.

ASCII-Kode

ASCII = **A**merican **S**tandard **C**ode for **I**nformation **I**nterchanging
КОИ = Код обмена информацией

usw.

Der ASCII-Kode ist - wie der Name schon sagt - ein Kode, der für den Austausch von Information zwischen z. B. einem Rechner und einem peripheren Gerät vereinbart (standardisiert) wurde. Der ASCII-Kode spielt aber auch eine dominierende Rolle bei Darstellung (Speicherung) von Information innerhalb des Rechners.

Wir wollen von folgendem **Scenario** ausgehen

Rechner <------> peripheres Gerät

Ausgabe: Rechner ----> peripheres Gerät (z. B. Drucker)
Eingabe: Rechner <---- peripheres Gerät (z. B. Chipkartenleser)

und uns auf die Ausgabe auf einen Drucker konzentrieren. Die Information, die von einem Rechner an einen Drucker gegeben werden muß, läßt sich grob in zwei Kategorien teilen (stellen Sie sich bitte anstelle eines Druckers eine Schreibmaschine und anstelle des Rechners sich selbst vor):

1. zu druckende Zeichen

2. gewisse Steuerinformationen für den Drucker, z. B.

   - Wagenrücklauf
   - Zeilenvorschub
   - Tabulator
   - Seitenwechsel (Vorschub auf den Anfang der nächsten Seite)

Zu jeder besseren Rechnerdokumentation und zu jeder Einführung in maschinenorientierte (d. h. Assembler-) Programmierung gehört eine ASCII-Kode-Tabelle. Wir wollen uns den Kode anhand einer solchen ASCII-Kode-Tabelle erarbeiten:

DEZ	HX	ASC	DEZ	HX	ASC	DEZ	HX	ASC	DEZ	HX	ASC	DEZ	HX	ASC
0	00	NUL	48	30	0	96	60	`	128	80	Ç	176	B0	▒
1	01	SOH	49	31	1	97	61	a	129	81	ü	177	B1	▓
2	02	STX	50	32	2	98	62	b	130	82	é	178	B2	│
3	03	ETX	51	33	3	99	63	c	131	83	â	179	B3	┤
4	04	EOT	52	34	4	100	64	d	132	84	ä	180	B4	┤
5	05	ENQ	53	35	5	101	65	e	133	85	à	181	B5	Á
6	06	ACK	54	36	6	102	66	f	134	86	å	182	B6	Â
7	07	BEL	55	37	7	103	67	g	135	87	ç	183	B7	À
8	08	BS	56	38	8	104	68	h	136	88	ê	184	B8	©
9	09	HT	57	39	9	105	69	i	137	89	ë	185	B9	╣
10	0A	LF	58	3A	:	106	6A	j	138	8A	è	186	BA	║
11	0B	VT	59	3B	;	107	6B	k	139	8B	ï	187	BB	╗
12	0C	FF	60	3C	<	108	6C	l	140	8C	î	188	BC	╝
13	0D	CR	61	3D	=	109	6D	m	141	8D	ì	189	BD	¢
14	0E	SO	62	3E	>	110	6E	n	142	8E	Ä	190	BE	¥
15	0F	SI	63	3F	?	111	6F	o	143	8F	Å	191	BF	┐
16	10	DLE	64	40	@	112	70	p	144	90	É	192	C0	└
17	11	DC1	65	41	A	113	71	q	145	91	æ	193	C1	┴
18	12	DC2	66	42	B	114	72	r	146	92	Æ	194	C2	┬
19	13	DC3	67	43	C	115	73	s	147	93	ô	195	C3	├
20	14	DC4	68	44	D	116	74	t	148	94	ö	196	C4	─
21	15	NAK	69	45	E	117	75	u	149	95	ò	197	C5	┼
22	16	SYN	70	46	F	118	76	v	150	96	û	198	C6	ã
23	17	ETB	71	47	G	119	77	w	151	97	ù	199	C7	Ã
24	18	CAN	72	48	H	120	78	x	152	98	ÿ	200	C8	╚
25	19	EM	73	49	I	121	79	y	153	99	Ö	201	C9	╔
26	1A	SUB	74	4A	J	122	7A	z	154	9A	Ü	202	CA	╩
27	1B	ESC	75	4B	K	123	7B	{	155	9B	ø	203	CB	╦
28	1C	FS	76	4C	L	124	7C	|	156	9C	£	204	CC	╠
29	1D	GS	77	4D	M	125	7D	}	157	9D	Ø	205	CD	═
30	1E	RS	78	4E	N	126	7E	~	158	9E	×	206	CE	╬
31	1F	US	79	4F	O	127	7F	DEL	159	9F	ƒ	207	CF	¤
32	20	SP	80	50	P				160	A0	á	208	D0	ð
33	21	!	81	51	Q				161	A1	í	209	D1	Ð
34	22	"	82	52	R				162	A2	ó	210	D2	Ê
35	23	#	83	53	S				163	A3	ú	211	D3	Ë
36	24	%	84	54	T				164	A4	ñ	212	D4	È
37	25	&	85	55	U				165	A5	Ñ	213	D5	ı
38	26	’	86	56	V				166	A6	ª	214	D6	Í
39	27	(	87	57	W				167	A7		215	D7	Î
40	28	)	88	58	X				168	A8		216	D8	Ï
41	29	*	89	59	Y				169	A9		217	D9	┘
42	2A	+	90	5A	Z				170	AA		218	DA	┌
43	2B	,	91	5B	[				171	AB		219	DB	█
44	2C	-	92	5C	\				172	AC		220	DC	▄
45	2D	.	93	5D	]				173	AD		221	DD	▌
46	2E	/	94	5E	^				174	AE		222	DE	▐
47	2F		95	5F	_				175	AF		223	DF	▀

DEZ	HX	ASC
224	E0	Ó
225	E1	ß
226	E2	Ô
227	E3	Ò
228	E4	õ
229	E5	Õ
230	E6	µ
231	E7	þ
232	E8	Þ
233	E9	Ú
234	EA	Û
235	EB	Ù
236	EC	ý
237	ED	Ý
238	EE	¯
239	EF	´
240	F0	
241	F1	±
242	F2	=
243	F3	¾
244	F4	¶
245	F5	§
246	F6	÷
247	F7	¸
248	F8	°
249	F9	¨
250	FA	·
251	FB	¹
252	FC	³
253	FD	²
254	FE	■
255	FF	

Aufbau der Kodetabelle:

3 Spalten **DEZ HX ASC**, d. h. jedes Zeichen (druckbares Zeichen oder Steuerzeichen) wird in seiner **dezimalen** und **hexadezimalen** Repräsentation und der Darstellung des **Zeichen**s selbst (druckbares Zeichen) oder einer **mnemonische**n **Bezeichnung** des Zeichens (Steuerzeichen) gegenüber gestellt.

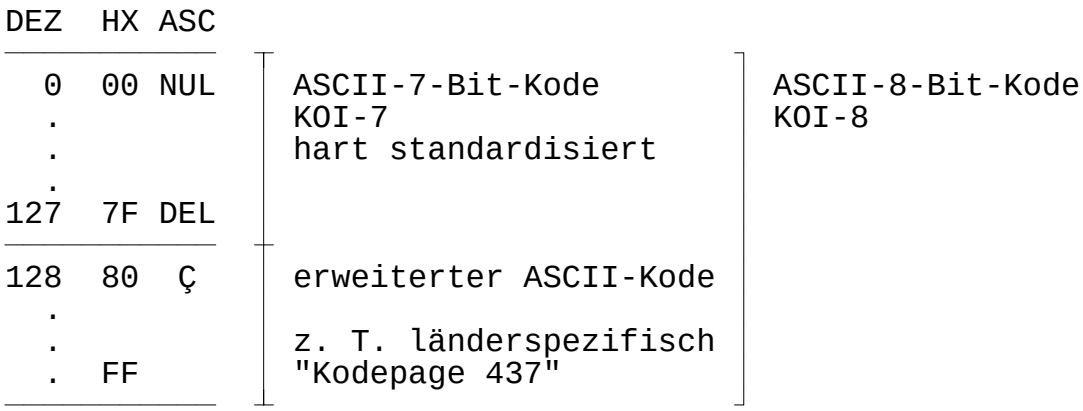
Warum das alles?

HX <--> ASC wird verständlich, wenn man sich vorstellt, daß es gelegentlich nötig wird, ein Zeichen in einem HEX-Dump wiederzufinden.

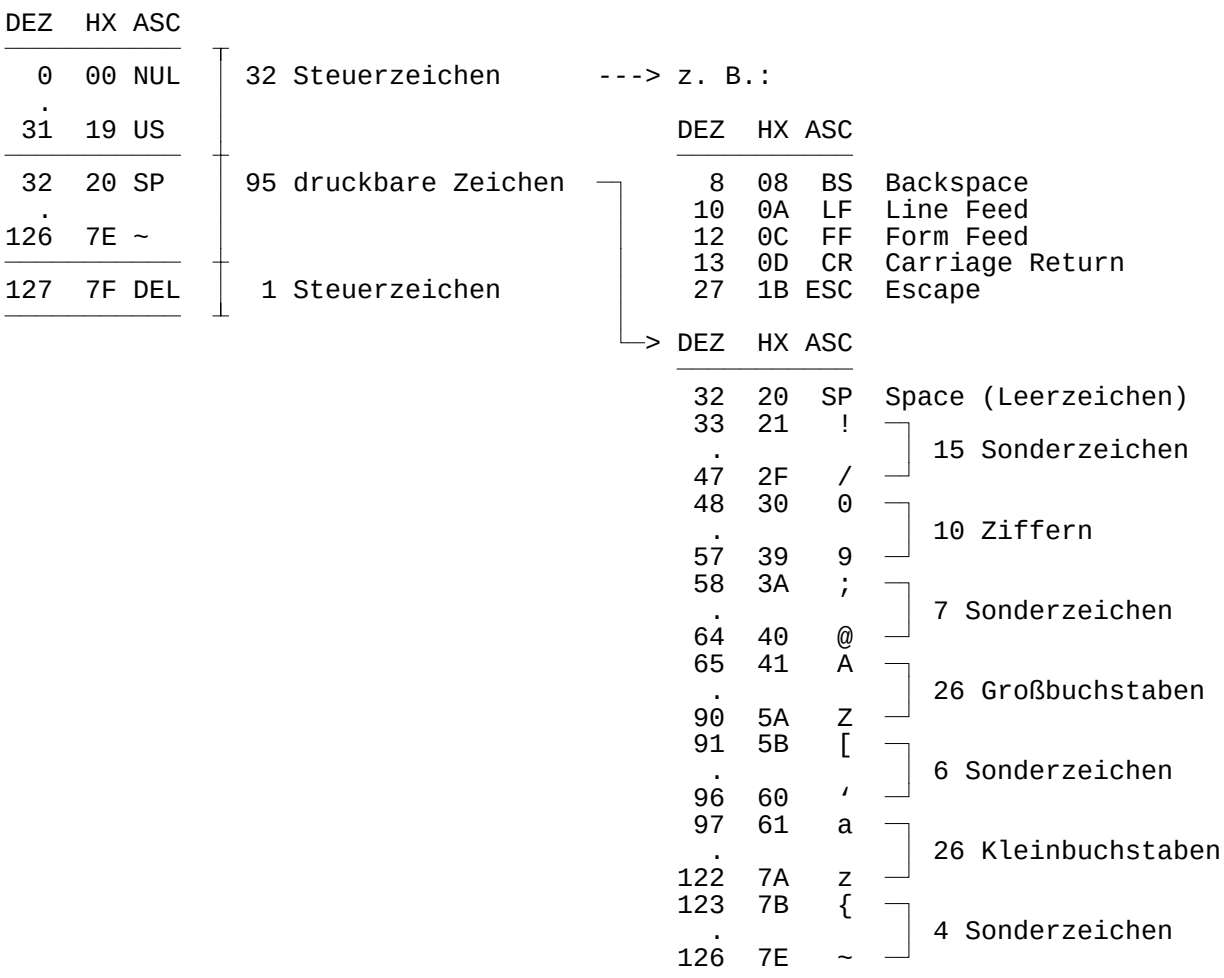
DEZ <--> ASC ist gut, wenn man mit der Tastatur alle 256 Zeichen eingeben können will, z. B. <Alt-2-1-9> = ▄, Ziffern im Zifferblock.

Man kann aber in gewissen typfreien Programmiersprachen auch mit ASCII-Zeichen "rechnen", z. B. ’A’ + 1 = ’B’.

Grobeinteilung in zwei Zeilenmengen



Feineinteilung des ASCII-7-Bit-Kodes



An der jeweils ersten Hexaziffer bzw. (Binär-) Tetrade (der sogenannten **Zone**) ist ganz grob und nicht hundertprozentig genau zu erkennen, um welche Art von Zeichen es sich handelt: **0 bis 1** - (nichtdruckbare) Steuerzeichen, **2 bis 7** - druckbare Zeichen mit **3** - Ziffern

Feineinteilung des erweiterten ASCII-Kodes

s. Kode-Pages, ASCII-Grafik, länderspezifische Buchstaben, div. weitere nichtländerspezifische druckbare Zeichen, z. B. ², ³, µ.

Zahlendarstellung

Wie bereits dargestellt, werden die Ziffern 0 bis 9 im ASCII-Kode als 30_{16} bis 39_{16} oder 0011_0000_2 bis 0011_1001_2 dargestellt. Der numerische Wert der Ziffer ist jeweils der zweiten Hexaziffer bzw. der zweiten (Binär-) Tetrade entnehmbar. Die Zone hat stets den Wert 3_{16} bzw. 0011_2. Man nennt diese Darstellung die **entpackte oder ungepackte Darstellung**.

Da - wenn man einmal weiß, daß es sich bei einer Zeichenkette um eine Zahl handelt - die Zonen keine Information tragen, kennt man daneben auch die **gepackte** Darstellung, bei der die Zonen entfallen und die numerischen Hexaziffern bzw. (Binär-) Tetraden dicht hintereinander angeordnet sind.

4. Technische Realisierung

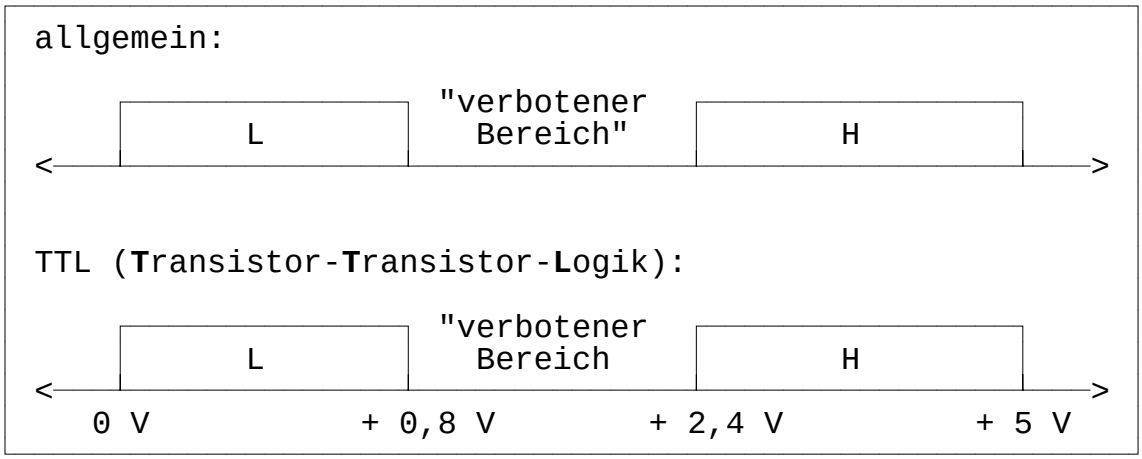
Sie erinnern sich:

Ein Signal ist eine zeitlich veränderliche physikalische Größe, die eine auf sie abgebildete Information trägt.

Hier:

physikalische Größe = elektrische Spannung als Informationsträger
Information = binär kodiert

Es wäre denkbar, jedem Symbol der Menge B = {0,1} einen bestimmten, wohlunterscheidbaren Spannungs**wert** zuzuordnen. Technische Systeme haben es aber an sich, Werte nur innerhalb gewisser Grenzen konstant halten zu können. Aus diesem Grunde ordnet man jedem Symbol einen bestimmten, wohlunterscheidbaren Spannungs**bereich** zu, der auch unter extremen Verhältnissen (Temperatur, Betriebsspannung, Last,...) eingehalten werden kann. Der mathematisch kleinere der beiden Bereiche heißt "LOW" ("L"), der mathematisch größere "HIGH" ("H").

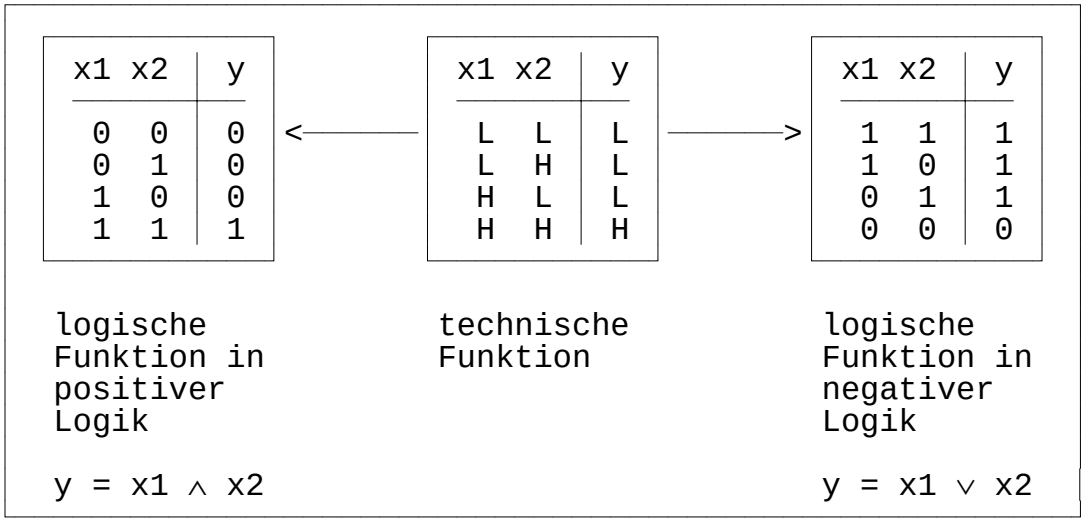


Man unterscheidet zwei Arten der Zuordung der Symbole der Menge B = {0,1} zu den Symbolen der Menge U = {L,H}:

positive Logik:	B ⟷ U	negative Logik:	B ⟷ U
	0 ⟷ L		0 ⟷ H
	1 ⟷ H		1 ⟷ L

Beispiel 4.1 (Vorgriff auf **Logikgatter**)

Gegeben sei ein Logikgatter der Funktion $y = f(x_1,x_2)$. Die technische Funktion $y = f_t(x_1,x_2)$ mit $y,x_1,x_2 \in \{L,H\}$ sei durch die unten in der Mitte angegebene Wahrheitstabelle beschrieben.



Interpretiert man die technische Funktion in positiver Logik, so ergibt sich die linke Wahrheitstabelle. Wählt man die Interpretation in negativer Logik, folgt daraus die rechte Wahrheitstabelle. Auslesen der Wahrheitstabellen (Vorgriff auf die BOOLEsche Algebra) liefert für die positive Logik die Konjunktion (AND) $y = x_1 \wedge x_2$ und die negative Logik die Disjunktion (OR) $y = x_1 \vee x_2$.

Ein Logikgatter, das - in positiver Logik betrieben - ein AND-Gatter ist, ist - in negativer Logik betrieben - ein OR-Gatter, und umgekehrt. Konjunktion und Disjunktion bilden ein **duales Funktionspaar** (vgl. Dualität der Wellen- und Teilchennatur des Lichts).

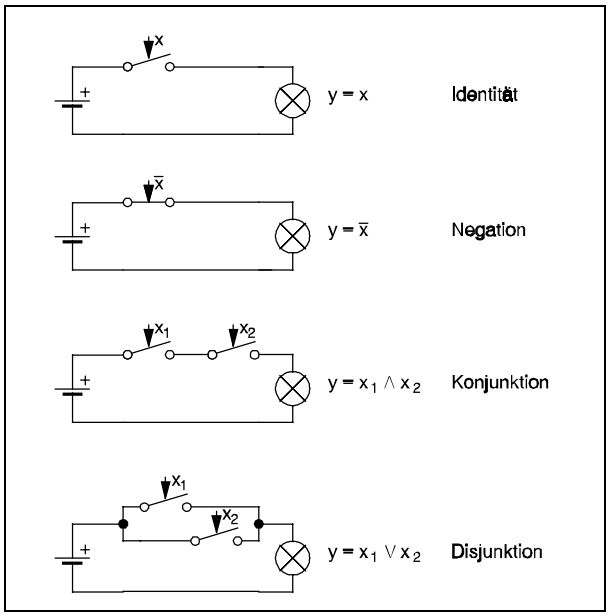
Bauelementehersteller, die nicht wissen können, in welcher Logik der Anwender ein Gatter betreiben wird, geben die Wahrheitstabellen deshalb stets in technischen Pegeln U = {L,H} an, obwohl für jede Schaltkreisreihe eine Vorzugszuordnung definiert ist (für TTL ist das die positive Logik). Anwender (und wir in der Lehre) verwenden so gut wie immer die logischen Pegel B = {0,1}. Im Hardwarepraktikum werden wir uns etwas genauer mit TTL-Gattern befassen.

Noch eine Bemerkung zum "Verbotenen Bereich". Ein Logikgatter wird dann korrekt betrieben, wenn der Pegel nicht in den verbotenen Bereich kommt. Die Gatter "bringen" das aber (Hinweis für die "Bastler").

Eine ältere Form der technischen Realisierung ist die mittels elektrischer Schaltkontakte. Diese Realisierung hat den Vorteil, daß sie gut verständlich ist. Die BOOLEsche Algebra wurde früher überhaupt nur "Schaltalgebra" genannt, da die Beschreibung von Relaisschaltungen das erste technisch relevante Anwendungsgebiet der BOOLEschen Algebra war. Informationsträger ist hier der

elektrische Strom. Den Symbolen B = {0,1} werden die Stromwerte $I = \{0, I_1\}$ zugeordnet. Meist verwendet man die positive Logik.

Beispiel 4.2:



5. Funktionen und Entwurf digitaler Schaltungen
5.1. Kombinatorische Schaltungen
5.1.1. Grundlagen

Wir sind bisher - eher intuitiv - bereits mit kombinatorischen Systemen umgegangen und wollen nun beginnen, die Betrachtungen theoretisch zu umtermauern und weiterzuführen.

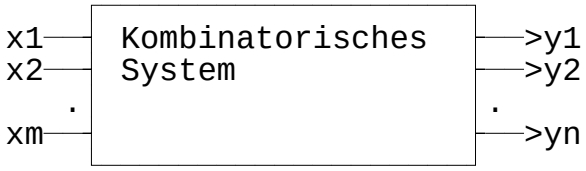
Kombinatorische Systeme sind Systeme, bei denen die Belegung

$$\underline{y} = y_1 y_2 \ldots y_n$$

an den Ausgängen ausschließlich von den Belegungen an den Eingängen

$$\underline{x} = x_1 x_2 \ldots x_m$$

zum gleichen Zeitpunkt abhängen.



Das mathematische Mittel zur Beschreibung kombinatorischer Systeme ist die BOOLEsche Algebra (George BOOLE, 1815 - 1864). Ich werde eine auf die technische Anwendung der BOOLEschen Algebra reduzierte Einführung bringen. Sie haben gewise Grundkenntnisse aus der Schule. Außerdem wird die BOOLEsche Algebra auch im Rahmen der Mathematikausbildung behandelt.

5.1.1.1. BOOLEsche Funktionen

Eine BOOLEsche (binäre) Funktion ist eine eindeutige Abbildung

$$f: B^k \rightarrow B^1,$$

die jedem Binärvektor

$$\underline{x} \in B^k = \{\underline{x} \mid \underline{x} = (x_1, \ldots, x_k),\ x_i \in \{0,1\},\ i = 1, \ldots, k\}$$

eines der beiden Elemente der Menge

$$B^1 = \{0,1\}$$

zuordnet. Da B^k genau 2^k Elemente besitzt und die Zuordnung der Werte 0 und 1 zu den Binärvektoren unabhängig voneinander geschehen kann, gibt es 2^{2^k} voneinander verschiedene Funktionen. In diese Zahl gehen auch alle Entartungen (Funktionen von k-1, ... , 0 Variablen) ein.

x_1	01		Bezeichnung
f_1	00	0	Konstante Null
f_2	01	x_1	Identität
f_3	10	$\overline{x_1}$	Negation
f_4	11	1	Konstante Eins

Die binären Funktionen über $B^1 = \{\underline{x} \mid (x_1),\ x_1 \in \{0,1\}\}$

x_1	0011		Bezeichnung
x_2	0101		
f_1	0000	0	Konstante 0
f_2	0001	$x_1 \wedge x_2$	Konjunktion, AND
f_3	0010	$x_1 \wedge \overline{x_2}$	Inhibition
f_4	0011	x_1	Identität
f_5	0100	$\overline{x_1} \wedge x_2$	Inhibition
f_6	0101	x_2	Identität
f_7	0110	$x_1 \not\sim x_2 = \overline{x_1} \wedge x_2 \vee x_1 \wedge \overline{x_2}$	Antivalenz, XOR
f_8	0111	$x_1 \vee x_2$	Disjunktion, OR
f_9	1000	$\overline{x_1} \wedge \overline{x_2} = \overline{x_1 \vee x_2}$	NOR
f_{10}	1001	$x_1 \sim x_2 = \overline{x_1} \wedge \overline{x_2} \vee x_1 \wedge x_2$	Äquivalenz
f_{11}	1010	$\overline{x_2}$	Negation
f_{12}	1011	$x_2 \rightarrow x_1 = \overline{x_2} \vee x_1$	Implikation
f_{13}	1100	$\overline{x_1}$	Negation
f_{14}	1101	$x_1 \rightarrow x_2 = \overline{x_1} \vee x_2$	Implikation
f_{15}	1110	$\overline{x_1} \vee \overline{x_2} = \overline{x_1 \wedge x_2}$	NAND
f_{16}	1111	1	Konstante 1

Die binären Funktionen über $B^2 = \{\underline{x} \mid (x_1,x_2),\ x_1,x_2 \in \{0,1\}\}$

In den dritten Spalten der oben angegebenen Tabellen sind die Funktionen in einer Form dargestellt, bei der neben den Konstanten 0 und 1 nur die Operatoren $\overline{\ }$ (Negation), $\wedge$ (Konjunktion) und $\vee$ (Disjunktion) verwendet werden. Die Negation, die Konjunktion und die Disjunktion bilden zusammen ein funktionell vollständiges System BOOLEscher Funktionen ("Algebra der Logik"), d. h. alle BOOLEsche Funktionen sind mit ihrer Hilfe darstellbar. Es gibt eine ganze Reihe solcher funktionell vollständiger Systeme BOOLEscher Funktionen:

- Konjunktion, Disjunktion, Negation ("Algebra der Logik")
- Konjunktion, Negation
- Disjunktion, Negation
- Implikation, Negation
- Inhibition, Negation
- NAND
- NOR
- Antivalenz, Konjunktion, Konstante ’1’ ("Shegalkin-Algebra")
- Äquivalenz, Disjunktion, Konstante ’0’

Streng genommen, ist die Algebra der Logik nicht vollständig, da sie nicht minimal ist, d. h. es existiert wenigstens eine Untermenge der Funktionen der Algebra der Logik, die ihrerseits vollständig ist. Von besonderer technischer Bedeutung sind die NAND- und die NOR-Funktion. Die beiden zuletzt aufgeführten Systeme nennt man auch "schwach vollständig", da sie neben BOOLEschen Funktionen auch eine BOOLEsche Konstante enthalten, ohne die nicht alle BOOLEschen Funktionen darstellbar wären.

5.1.2. Grundgesetze der BOOLEschen Algebra und Rechenregeln

Auf die mathematisch korrekte Einführung der BOOLEschen Algebra kann ich verzichten, da das in Ihrer Mathematikausbildung ausführlich behandelt wird. Ich stelle Ihnen zunächst die wichtigsten Rechenregeln vor. Nachfolgend werden wir dann eine Reihe von Anwendungen auf kombinatorische Schaltungen kennenlernen.

Die BOOLEsche ALgebra kennt eine ganze Menge Regeln, die Sie - wenn Sie einigermaßen erfolgreich mit der BOOLEschen Algebra umgehen wollen - alle kennen sollten. Einige davon haben Sie bereits kennengelernt.

Der Umgang mit der BOOLEschen Algebra erfordert einige Routine. Eine Besonderheit der BOOLEschen Algebra besteht darin, daß die sture Anwendung der Regeln meist nicht zum Ziel führt. Man muß im allgemeinen sehr konkrete Vorstellungen davon haben, wie das Ergebnis aussehen soll, und dafür geeignete Regeln auswählen.

Eine BOOLEsche Funktion (s. o.) ist eine eindeutige Abbildung

$$f: B^k \longrightarrow B$$

$$\text{mit } B = \{0,1\}$$

$$\text{und } B^k = \underbrace{B \times B \times \ldots \times B}_{k\text{-mal}}$$

$$= \{\underline{x} \mid \underline{x} = (x_1, x_2, \ldots, x_k)\},\ x_i \in B,\ i = 1, \ldots, k$$

$$= \{0\ldots00, 0\ldots01, 0\ldots10, 0\ldots11, \ldots, 1\ldots11\}$$

die jedem Binärvektor $\underline{x} \in B^k$ eines der beiden Elemente der Menge B zuordnet.

Für die Menge B gelte die naheliegende Ordnungsrelation $0 < 1$, woraus folgt

$$0 \leq x \text{ für alle } x$$

$$x \leq 1 \text{ für alle } x$$

Wir wollen die Konsequenzen daraus hier nicht weiter vertiefen. Für gewisse Begriffsbildungen ist es aber nützlich, sich diesen Sachverhalt einzuprägen.

Von den oben bereits eingeführten Funktionen einer und zweier Variablen werden wir im weiteren nur eine Teilmenge benötigen und näher betrachten.

Negation

Schreibweisen $y = f(x) = \overline{x} = /x = \neg x = \text{NOT } x$

Wahrheitstabelle

x	y
0	1
1	0

Regeln

$\overline{0} = 1$

$\overline{1} = 0$

$\overline{\overline{0}} = 0$

$\overline{\overline{1}} = 1$

$\overline{\overline{x}} = x$

Konjunktion

Schreibweisen $y = f(x_1,x_2) = x_1 \wedge x_2 = x_1 x_2 = x_1 \& x_2 = x_1 \bullet x_2$

$= x_1 \text{ AND } x_2 = \min(x_1,x_2)$

Wahrheitstabelle

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Regeln

$0 \wedge x = 0$

$1 \wedge x = x$

$x \wedge x = x$ (Idempotenz)

$x \wedge \overline{x} = 0$ (Komplement)

$x_1 \wedge x_2 = x_2 \wedge x_1$ (Kommutativität)

$x_1 \wedge x_2 \wedge x_3 = (x_1 \wedge x_2) \wedge x_3 = x_1 \wedge (x_2 \wedge x_3)$ (Assoziativität)

Disjunktion

Schreibweisen $y = f(x_1,x_2) = x_1 \vee x_2 = x_1 + x_2$

$= x_1 \text{ OR } x_2 = \max(x_1,x_2)$

Wahrheitstabelle

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Regeln

$0 \vee x = x$

$1 \vee x = 1$

$x \vee x = x$ (Idempotenz)

$x \vee \overline{x} = 1$ (Komplement)

$x_1 \vee x_2 = x_2 \vee x_1$ (Kommutativität)

$x_1 \vee x_2 \vee x_3 = (x_1 \vee x_2) \vee x_3 = x_1 \vee (x_2 \vee x_3)$ (Assoziativität)

Antivalenz

Schreibweisen

$y = f(x_1, x_2) = x_1 \not\vee x_2 = x_1 \oplus x_2 = x_1 \neq x_2$

$= x_1$ XOR x_2

Wahrheitstabelle

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

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

Regeln

$0 \not\vee x = x$

$1 \not\vee x = \overline{x}$

$x \not\vee x = 0$

$x \not\vee \overline{x} = 1$

$x_1 \not\vee x_2 = x_2 \not\vee x_1$ (Kommutativität)

$x_1 \not\vee x_2 \not\vee x_3 = (x_1 \not\vee x_2) \not\vee x_3 = x_1 \not\vee (x_2 \not\vee x_3)$ (Assoziativität)

Äquivalenz

Schreibweisen

$y = f(x_1, x_2) = x_1 \sim x_2 = x_1 \equiv x_2$

$= x_1$ XNOR $x_2 = x_1$ EQU x_2

Wahrheitstabelle

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

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

Regeln

$0 \sim x = \overline{x}$

$1 \sim x = x$

$x \sim x = 1$

$x \sim \overline{x} = 0$

$x_1 \sim x_2 = x_2 \sim x_1$ (Kommutativität)

$x_1 \sim x_2 \sim x_3 = (x_1 \sim x_2) \sim x_3 = x_1 \sim (x_2 \sim x_3)$ (Assoziativität)

Weitere Rechenregeln

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

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

$(x_1 \wedge x_2) \nsim (x_1 \wedge x_3) = x_1 \wedge (x_2 \nsim x_3)$ (Distributivität)

$(x_1 \vee x_2) \sim (x_1 \vee x_3) = x_1 \vee (x_2 \sim x_3)$ (Distributivität)

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

$x_1 \vee (\overline{x}_1 \wedge x_2) = x_1 \vee x_2$ (Absorption)

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

$x_1 \wedge (\overline{x}_1 \vee x_2) = x_1 \wedge x_2$ (Absorption)

$\overline{x_1 \wedge x_2} = \overline{x_1} \vee \overline{x_2}$ (DeMorgan)

$\overline{x_1 \vee x_2} = \overline{x_1} \wedge \overline{x_2}$ (DeMorgan)

$\overline{x_1 \wedge \ldots \wedge x_n} = \overline{x_1} \vee \ldots \vee \overline{x_n}$ (DeMorgan, generalisiert)

$\overline{x_1 \vee \ldots \vee x_n} = \overline{x_1} \wedge \ldots \wedge \overline{x_n}$ (DeMorgan, generalisiert)

$\overline{f(x_1,\ldots,x_n,\wedge,\vee)} = f(\overline{x_1},\ldots,\overline{x_2},\vee,\wedge)$ (Shannon)

Der Shannonsche Satz läßt sich noch weiter verallgemeinern. Wegen der Dualität von Konjunktion und Disjunktion einerseits

und Äquivalenz und Antivalenz andererseits gilt z. B. auch

$$\overline{f(x_1,...,x_n,\wedge,\not\sim)} = f(\overline{x_1},...,\overline{x_2},\vee,\sim)$$

Für den Umgang mit Äquivalenz und Antivalenz sind noch die nachfolgenden Regeln hilfreich.

$$x_1 \not\sim x_2 = \overline{x_1} \not\sim \overline{x_2}$$

$$\mathbf{x_1 \sim x_2} = \overline{x_1} \sim \overline{x_2}$$

$$\overline{\mathbf{x_1 \not\sim x_2}} = \overline{\overline{x_1} \not\sim \overline{x_2}} = \overline{x_1} \not\sim x_2 = x_1 \not\sim \overline{x_2} = \mathbf{x_1 \sim x_2}$$

$$\overline{\mathbf{x_1 \sim x_2}} = \overline{\overline{x_1} \sim \overline{x_2}} = \overline{x_1} \sim x_2 = x_1 \sim \overline{x_2} = \mathbf{x_1 \not\sim x_2}$$

Die vier letztgenannten Regeln gelten allgemein nur für gerade Anzahlen von Variablen. Für ungerade Anzahlen von Variablen gilt z. B.

$$\mathbf{x_1 \not\sim x_2 \not\sim x_3 = x_1 \sim x_2 \sim x_3} \quad \mathbf{!!!}$$

Die beiden letzten Regeln werden wir nur selten und dann nur in einem einzigen Zusammenhang verwenden:

Entwickeln einer Funktion $f(\underline{x})$ nach einer Variablen x_i:

$$\begin{aligned} f(x_1,...,x_i,...,x_n) &= x_i \wedge f(\underline{x})\Big|_{x_i=1} \vee \overline{x_i} \wedge f(\underline{x})\Big|_{x_i=0} \\ &= x_i \wedge f(x_1,...,1,...,x_n) \vee \\ &\quad \overline{x_i} \wedge f(x_1,...,0,...,x_n) \end{aligned}$$

Ableiten einer Funktion $f(\underline{x})$ nach einer Variablen x_i:

$$\begin{aligned} \frac{\delta f(\underline{x})}{\delta x_i} = \frac{\delta f(x_1,...,x_i,...,x_n)}{\delta x_i} &= f(\underline{x})\Big|_{x_i=0} \not\sim f(\underline{x})\Big|_{x_i=1} \\ &= f(x_1,...,0,...,x_n) \not\sim \\ &\quad f(x_1,...,1,...,x_n) \\ &= f(x_1,...,\overline{x_i},...,x_n) \not\sim \\ &\quad f(x_1,...,x_i,...,x_n) \end{aligned}$$

5.1.3. Wahrheitstabelle und Normalformen

Wahrheitstabellen haben wir bereits kennengelernt und verwendet. Sie sind eines der grundlegenden Beschreibungsmittel BOOLEscher Funktionen. Im Kopf der Wahrheitstabelle werden auf der linken Seite die Variablen x_i in einer geeigneten (d. h. an das konkrete Problem angepaßten) Reihenfolge aufgetragen, auf der rechten Seite der Funktionswert $f(\underline{x})$. In der Tabelle selbst trägt man auf der linken Seite alle denkbaren Belegungen der Variablen untereinander ein (ebenfalls in einer geeigneten, problemangepaßten Reihenfolge!). Auf der rechten Seite ordnet man jeder Belegung der Variablen den entsprechenden Funktionswert zu.

x_1 x_2 ... x_n	$y = f(x_1,x_2,...,x_n)$
0 0 ... 0	f(0,0,...,0)
0 0 ... 1	f(0,0,...,1)
. . .	.
. . .	.
1 1 ... 1	f(1,1,...,1)

Aus der Wahrheitstabelle kann folgende BOOLEsche Funktion abgelesen werden:

$$y = f(x_1,x_2,...,x_n) = \overline{x_1}\wedge\overline{x_2}\wedge...\wedge\overline{x_n}\wedge f(00...0) \vee \overline{x_1}\wedge\overline{x_2}\wedge...\wedge x_n\wedge f(00...1) \vee \vdots \vee x_1\wedge x_2\wedge...\wedge x_n\wedge f(11...1)$$

Wir werden später diese ausführliche Schreibweise noch verwenden. Für die unmittelbar folgenden Betrachtungen verwenden wir eine vereinfachte Schreibweise. Da die Funktionswerte $f(\underline{x})$ nur die Werte 0 oder 1 annehmen können und da außerdem gilt $0\wedge x = 0$ und $0\vee x = x$, können alle diejenigen Konjunktionen weggelassen werden, die den Funktionswert 0 liefern. Bei den Konjunktionen, die den Funktionswert 1 liefern, lassen wir darüber hinaus den Funktionswert weg.

Beispiel 5.1

Unter "2.4.1. Addition" haben wir eine Wahrheitstabelle verwendet, dort allerdings für ein Funktionsbündel, da auf der rechten Seite zwei Funktionswerte aufgeführt waren. Wir beschränken uns hier auf den auslaufenden Übertrag c_{aus}

a b c_{ein}	c_{aus}	a b c_{ein}	c_{aus}
0 0 0	0	1 0 0	0
0 0 1	0	**1 0 1**	**1**
0 1 0	0	**1 1 0**	**1**
0 1 1	**1**	**1 1 1**	**1**

und lesen aus der Tabelle zunächst ausführlich ab:

$$c_{aus} = \bar{a}\wedge\bar{b}\wedge\overline{c_{ein}}\wedge 0 \vee \bar{a}\wedge\bar{b}\wedge c_{ein}\wedge 0 \vee \bar{a}\wedge b\wedge\overline{c_{ein}}\wedge 0 \vee \bar{a}\wedge b\wedge c_{ein}\wedge 1 \vee$$

$$a\wedge\bar{b}\wedge\overline{c_{ein}}\wedge 0 \vee a\wedge\bar{b}\wedge c_{ein}\wedge 1 \vee a\wedge b\wedge\overline{c_{ein}}\wedge 1 \vee a\wedge b\wedge c_{ein}\wedge 1$$

Weglassen aller Konjunktionen mit dem Funktionswert 0 und Weglassen des Funktionswerts bei den verbleibenden Konjunktionen liefert schließlich die vereinfachte Schreibweise:

$$c_{aus} = \bar{a}\wedge b\wedge c_{ein} \vee a\wedge\bar{b}\wedge c_{ein} \vee a\wedge b\wedge\overline{c_{ein}} \vee a\wedge b\wedge c_{ein}$$

Oben in der Tabelle sind die entsprechenden Konjunktionen fett markiert, und es wird klar sein, daß der "Umweg" über die ausführliche Schreibweise zukünftig nicht mehr nötig sein wird.

Diese Schreibweise (Notationsform) einer BOOLEschen Funktion heißt Kanonische disjunktive Normalform (KDNF).

Def.: Kanonische disjunktive Normalform (KDNF)

Die Kanonische disjunktive Normalform (KDNF) einer BOOLEschen Funktion f(<u>x</u>) ist die disjunktive Verknüpfung einer Anzahl von Konjunktionen K_j

$$\mathbf{f(\underline{x}) = f(x_1,x_2,...,x_i,...,x_m) = K_1 \vee K_2 \vee ... \vee K_j \vee ... \vee K_n,}$$

wobei jede der Konjunktionen K_j eine konjunktive Verknüpfung <u>aller</u> Variablen x_i (echt oder negiert) ist.

Beispiel 5.2.

Wie bleiben bei der Wahrheitstabelle aus Beispiel 5.1.

a	b	c_{ein}	c_{aus}	a	b	c_{ein}	c_{aus}
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**
0	**0**	**1**	**0**	1	0	1	1
0	**1**	**0**	**0**	1	1	0	1
0	1	1	1	1	1	1	1

Wenn man aus der Tabelle die Zeilen ausliest, in denen c_{aus} den Wert Null erhält (fett markiert), ergibt sich analog zu Beispiel 5.1. zunächst

$$\overline{c_{aus}} = \bar{a}\wedge\bar{b}\wedge\overline{c_{ein}} \vee \bar{a}\wedge\bar{b}\wedge c_{ein} \vee \bar{a}\wedge b\wedge\overline{c_{ein}} \vee a\wedge\bar{b}\wedge\overline{c_{ein}}$$

und weiter

$$c_{aus} = \overline{\bar{a}\wedge\bar{b}\wedge\bar{c}_{ein}} \vee \bar{a}\wedge\bar{b}\wedge c_{ein} \vee \bar{a}\wedge b\wedge\bar{c}_{ein} \vee a\wedge\bar{b}\wedge c_{ein}.$$

Die Anwendung einer der DeMorganschen Regeln liefert dann

$$c_{aus} = \overline{\bar{a}\wedge\bar{b}\wedge\bar{c}_{ein}} \wedge \overline{\bar{a}\wedge\bar{b}\wedge c_{ein}} \wedge \overline{\bar{a}\wedge b\wedge\bar{c}_{ein}} \wedge \overline{a\wedge\bar{b}\wedge c_{ein}}$$

$$= \underline{\underline{(a\vee b\vee c_{ein})\wedge(a\vee b\vee\bar{c}_{ein})\wedge(a\vee\bar{b}\vee c_{ein})\wedge(\bar{a}\vee b\vee c_{ein})}}$$

Diese Schreibweise (Notationsform) einer BOOLEschen Funktion heißt Kanonische konjunktive Normalform (KKNF).

Def.: Kanonische konjunktive Normalform (KKNF)

Die Kanonische konjunktive Normalform (KKNF) einer BOOLEschen Funktion $f(\underline{x})$ ist die konjunktive Verknüpfung einer Anzahl von Disjunktionen D_j

$f(\underline{x}) = f(x_1,x_2,...,x_i,...,x_m) = D_1 \wedge D_2 \wedge ... \wedge D_j \wedge ... \wedge D_n,$

wobei jede der Disjunktionen D_j eine disjunktive Verknüpfung <u>aller</u> Variablen x_i (echt oder negiert) ist.

Beispiel 5.3. (Vorgriff auf die Minimierung)

Ausgehend vom Ergebnis des Beispiels 5.1.

$$c_{aus} = \bar{a}\wedge b\wedge c_{ein} \vee a\wedge\bar{b}\wedge c_{ein} \vee a\wedge b\wedge\bar{c}_{ein} \vee a\wedge b\wedge c_{ein}$$

$$= \quad K_1 \quad \vee \quad K_2 \quad \vee \quad K_3 \quad \vee \quad K_4$$

findet man durch "scharfes Hinsehen"

$$K_1 \vee K_4 = \bar{a}\wedge b\wedge c_{ein} \vee a\wedge b\wedge c_{ein} = (\bar{a} \vee a)\wedge b\wedge c_{ein} = b\wedge c_{ein}$$

$$K_2 \vee K_4 = a\wedge\bar{b}\wedge c_{ein} \vee a\wedge b\wedge c_{ein} = a\wedge(\bar{b} \vee b)\wedge c_{ein} = a\wedge c_{ein}$$

$$K_3 \vee K_4 = a\wedge b\wedge\bar{c}_{ein} \vee a\wedge b\wedge c_{ein} = a\wedge b\wedge(\bar{c}_{ein} \vee c_{ein}) = a\wedge b$$

und schließlich

$$c_{aus} = b\wedge c_{ein} \vee a\wedge c_{ein} \vee a\wedge b.$$

Diese Schreibweise (Notationsform) einer BOOLEschen Funktion heißt Disjunktive Normalform (DNF).

Def.: Disjunktive Normalform (DNF)

Die Disjunktive Normalform (DNF) einer BOOLEschen Funktion $f(\underline{x})$ ist die disjunktive Verknüpfung einer Anzahl von Konjunktionen K_j

$$f(\underline{x}) = f(x_1, x_2, ..., x_i, ..., x_m) = K_1 \vee K_2 \vee ... \vee K_j \vee ... \vee K_n,$$

wobei jede der Konjunktionen K_j <u>eine einzige</u> Variable x_i (echt oder negiert) oder die konjunktive Verknüpfung <u>mehrerer oder aller</u> Variablen x_i (echt oder negiert) sein kann.

Beispiel 5.4. (Vorgriff auf die Minimierung)

Ausgehend vom Ergebnis des Beispiels 5.2.

$$c_{aus} = (a \vee b \vee c_{ein}) \wedge (a \vee b \vee \overline{c_{ein}}) \wedge (a \vee \overline{b} \vee c_{ein}) \wedge (\overline{a} \vee b \vee c_{ein})$$

$$= \quad D_1 \quad \wedge \quad D_2 \quad \wedge \quad D_3 \quad \wedge \quad D_4$$

findet man durch "scharfes Hinsehen"

$$D_1 \wedge D_2 = (a \vee b \vee c_{ein}) \wedge (a \vee b \vee \overline{c_{ein}}) = a \vee b \vee (c_{ein} \wedge \overline{c}_{ein}) = a \vee b$$

$$D_1 \wedge D_3 = (a \vee b \vee c_{ein}) \wedge (a \vee \overline{b} \vee c_{ein}) = a \vee (b \wedge \overline{b}) \vee c_{ein} = a \vee c_{ein}$$

$$D_1 \wedge D_4 = (a \vee b \vee c_{ein}) \wedge (\overline{a} \vee b \vee c_{ein}) = (a \wedge \overline{a}) \vee b \vee c_{ein} = b \vee c_{ein}$$

und schließlich

$$c_{aus} = (a \vee b) \wedge (a \vee c_{ein}) \wedge (b \vee c_{ein}).$$

Diese Schreibweise (Notationsform) einer BOOLEschen Funktion heißt Konjunktive Normalform (KNF).

Def.: Konjunktive Normalform (KNF)

Die Konjunktive Normalform (KNF) einer BOOLEschen Funktion $f(\underline{x})$ ist die konjunktive Verknüpfung einer Anzahl von Disjunktionen D_j

$$f(\underline{x}) = f(x_1, x_2, ..., x_i, ..., x_m) = D_1 \wedge D_2 \wedge ... \wedge D_j \wedge ... \wedge D_n,$$

wobei jede der Disjunktionen D_j eine einzige Variable x_i (echt oder negiert) oder die disjunktive Verknüpfung mehrerer oder aller Variablen x_i (echt oder negiert) sein kann.

Für jede BOOLEsche Funktion $f(\underline{x})$ existiert - abgesehen von Vertauschungen wegen der Kommutativität der Konjunktion und der Disjunktion - genau eine KDNF und genau eine KKNF.

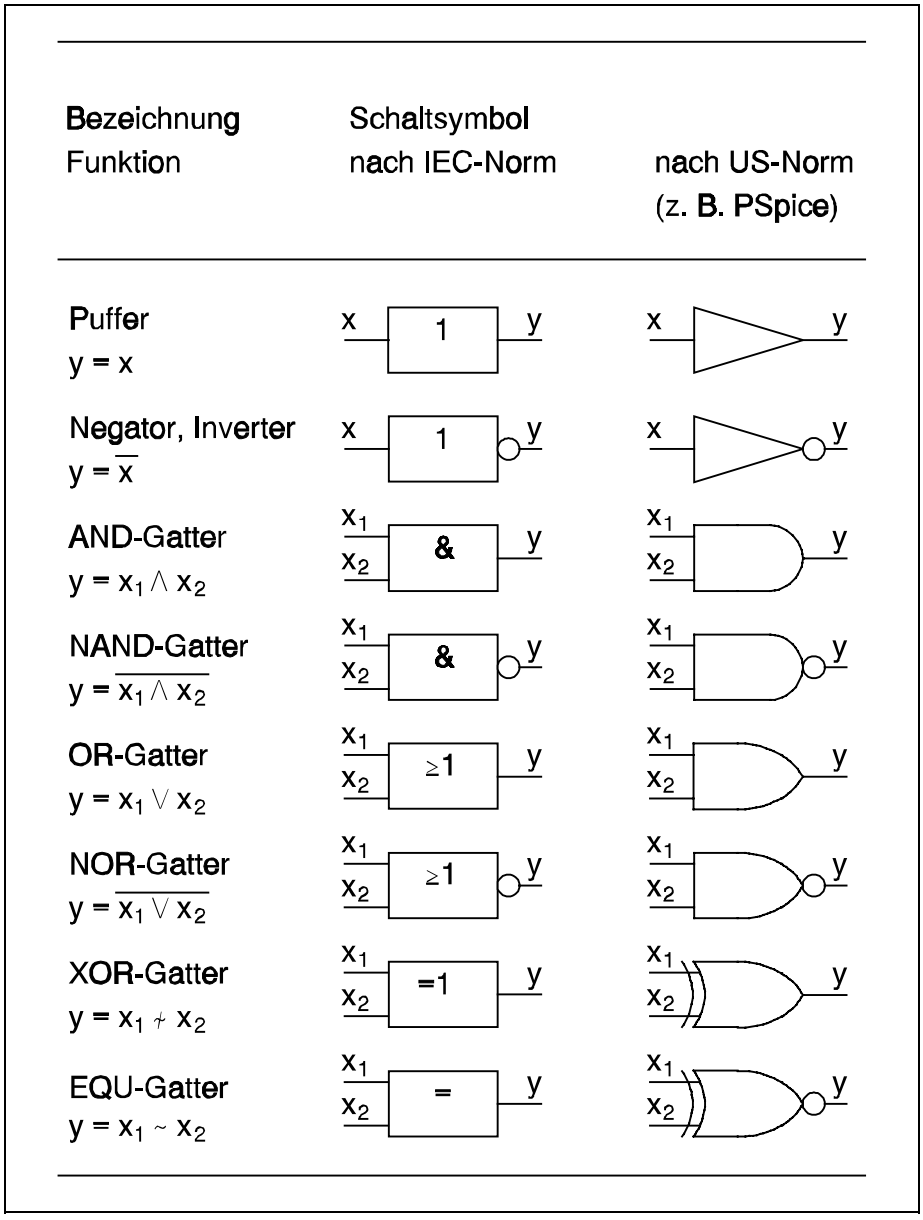
DNF und KNF kann es für ein und dieselbe BOOLEsche Funktion mehrere unterschiedliche geben.

Es gibt weitere Normalformen, die wir hier nicht verwenden werden.

5.1.4. Gatter und Gatternetze

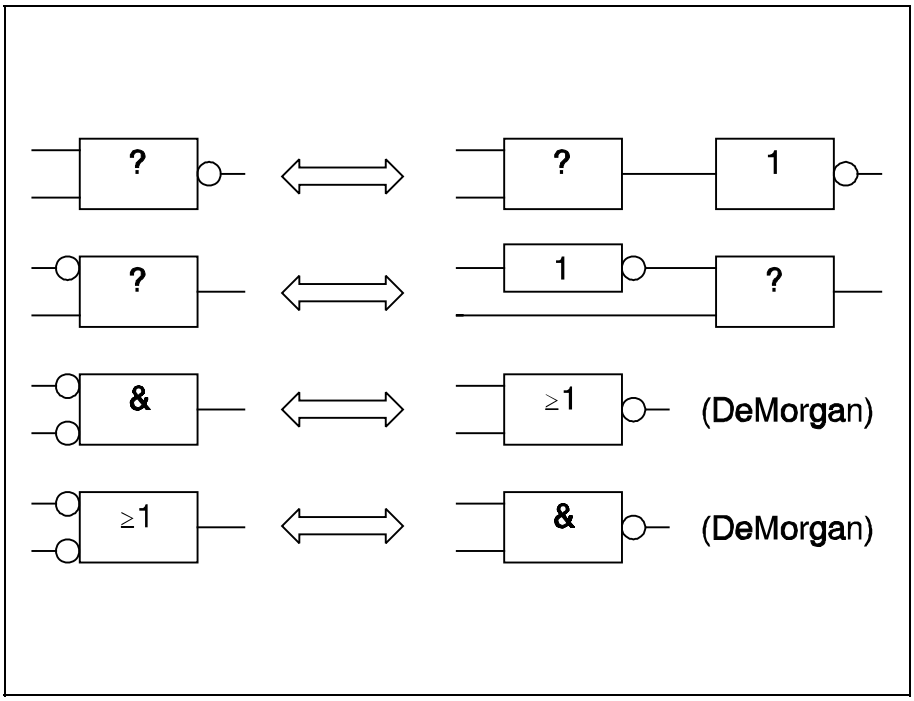
Gatter seien technische Realisierungen binärer Grundfunktionen (Operatoren); Gatternetze Zusammenschaltungen von Gattern in Form gerichteter, schleifenfreier (aber nicht zwingend maschenfreier) Netze zur Realisierung komplexerer binärer Funktionen.

Einige besonders häufig eingesetzte Gattertypen sind nachfolgend mit ihrer Bezeichnung, ihrer Funktion und ihren Schaltsymbolen zusammengestellt:



Anmerkung 1:

Für beide Normen gilt: Ein Kreis an einem Ausgang (oder auch an einem Eingang) steht für die Negation, d. h. (hier nur für die IEC-Norm dargestellt) z. B.:



Anmerkung 2:

In der Übersicht oben sind nur ein- und zweieingängige Gatter dargestellt. Wegen der Assoziativität der Konjunktion, der Disjunktion, der Antivalenz und der Äquivalenz (s. 5.1.2.) sind aber auch Gatter mit mehr als zwei Eingängen möglich.

Anmerkung 3:

Die in der IEC-Norm verwendeten Bezeichnungen für die Gattertypen sind:

’1’ - Identität

’&’ - Konjunktion

’≥1’ - Disjunktion (der Ausgang y führt dann den Wert 1, wenn ein oder mehrere Eingänge x_i den Wert 1 führen)

’=1’ - Antivalenz (der Ausgang y führt dann den Wert 1, wenn genau ein Eingang x_i den Wert 1 führt; bei Gattern mit mehr als zwei Eingängen ist diese Aussage falsch (Warum?))

’=’ - Äquivalenz (der Ausgang y führt dann den Wert 1, wenn alle Eingänge den gleichen Wert führen; bei Gattern mit mehr als zwei Eingängen ist diese Aussage falsch (Warum?))

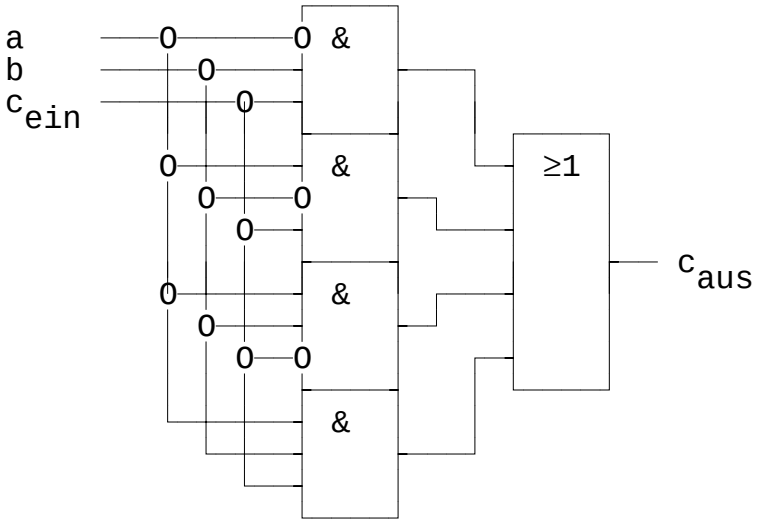
Anmerkung 4:

Das US-Symbol für das EQU-Gatter gilt exakt nur für das zweieingängige Gatter (Warum?)

Ein Gatternetz, das eine bestimmte binäre Funktion realisieren soll, werden wir grundsätzlich als Abbildung einer Schreibweise (Notationsform) dieser Funktion auffassen.

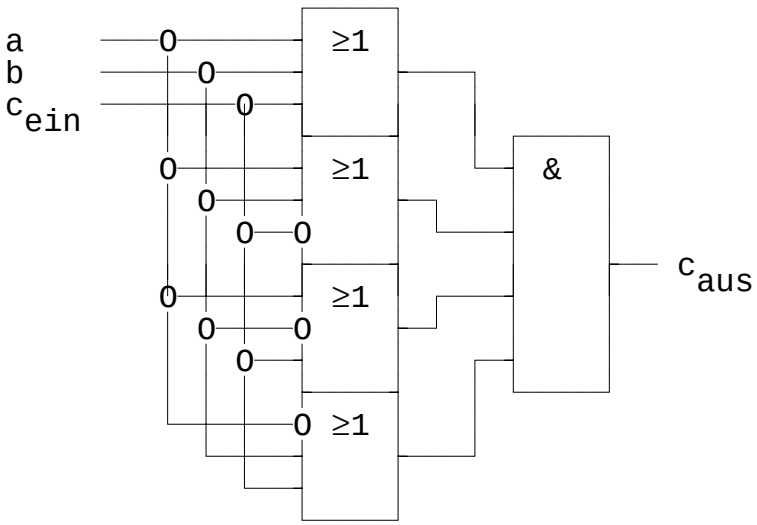
Beispiel 5.5. (vgl. Beispiel 5.1.)

$$c_{aus} = \bar{a} \wedge b \wedge c_{ein} \vee a \wedge \bar{b} \wedge c_{ein} \vee a \wedge b \wedge \overline{c_{ein}} \vee a \wedge b \wedge c_{ein}$$



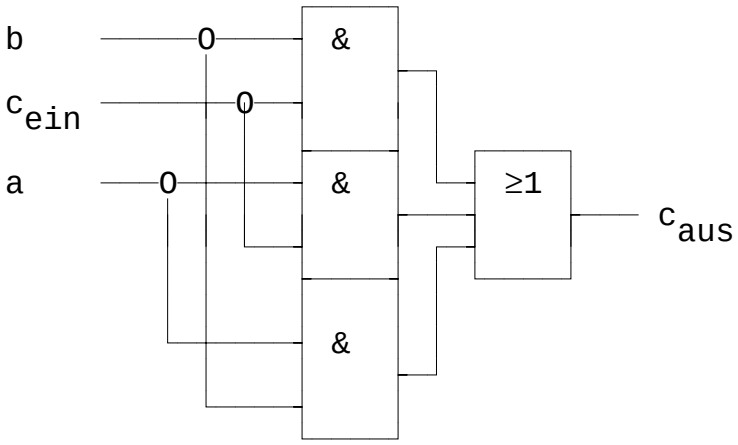
Beispiel 5.6. (vgl. Beispiel 5.2.)

$$c_{aus} = (a \vee b \vee c_{ein}) \wedge (a \vee b \vee \overline{c_{ein}}) \wedge (a \vee \bar{b} \vee c_{ein}) \wedge (\bar{a} \vee b \vee c_{ein})$$



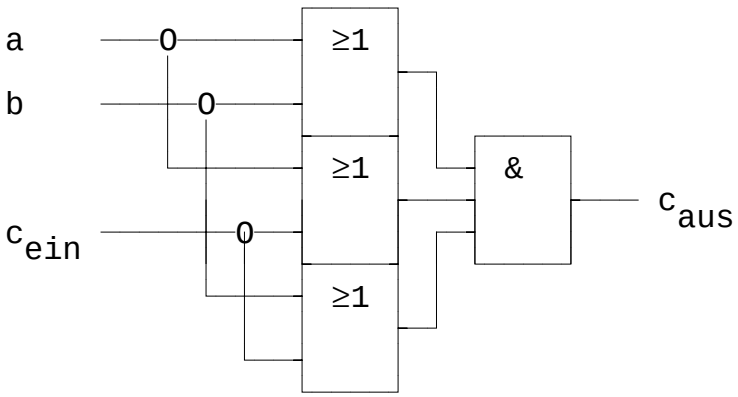
Beispiel 5.7. (vgl. Beispiel 5.3.)

$c_{aus} = b \wedge c_{ein} \vee a \wedge c_{ein} \vee a \wedge b$



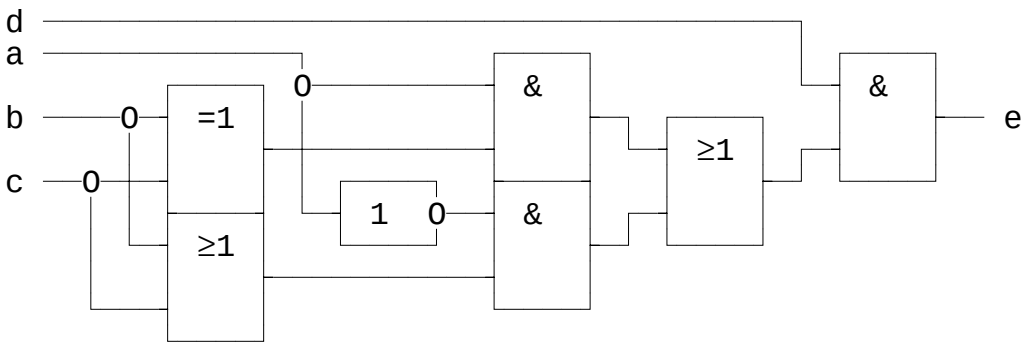
Beispiel 5.8. (vgl. Beispiel 5.4.)

$c_{aus} = (a \vee b) \wedge (a \vee c_{ein}) \wedge (b \vee c_{ein})$



Beispiel 5.9.

$e = d(a(b \not\leftrightarrow c) \vee \bar{a}(b \vee c))$



Beispiel 5.10.

Die BOOLEsche Funktion aus Beispiel 5.9. läßt sich so umschreiben:

e = d(a(b ⊻ c) ∨ ā(b ∨ c))

"Definition" der Antivalenz

= d(a(b̄c ∨ bc̄) ∨ ā(b ∨ c))

Distributivgesetz

= d(ab̄c ∨ abc̄ ∨ āb ∨ āc)

Komplement und x∧1 = x

= d(ab̄c ∨ abc̄ ∨ āb(c̄ ∨ c) ∨ ā(b̄ ∨ b)c)

Distributivgesetz

= d(ab̄c ∨ abc̄ ∨ ābc̄ ∨ ābc ∨ āb̄c ∨ ābc)

Idempotenz und Kommutativgesetz

= d(abc̄ ∨ ab̄c ∨ ābc ∨ ābc̄ ∨ āb̄c)

6 5 3 2 1 <—— sortiert!

Distributivgesetz

= abc̄d ∨ ab̄cd ∨ ābcd ∨ ābc̄d ∨ āb̄cd (KDNF)

D B 7 5 3 <—— sortiert!

Man sollte bei der Notation von KDNF grundsätzlich die Reihenfolge der Variablen in den Konjunktionen (hier: alphabetisch) und die Reihenfolge der Konjunktionen in der Disjunktion (hier: absteigend) einheitlich wählen! Für KKNF gilt diese Aussage analog. Die geordnete Notation erhöht deren Übersichtlichkeit und verhindert wirksam Irrtümer.

Die KDNF ist Ausgangsnotation für die weitere Vereinfachung. Die geschickte (hier: paarweise) Zusammenfassung von Konjunktionen liefert z. B.

$$= ab\bar{c}d \vee \bar{a}b\bar{c}d \vee a\bar{b}cd \vee \bar{a}\bar{b}cd \vee \bar{a}bcd \vee \bar{a}\bar{b}cd$$

D 5 B 3 7 3

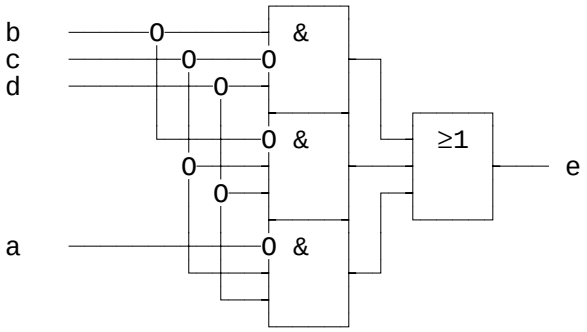
Distributivgesetz

$$= (a \vee a)b\bar{c}d \vee (a \vee a)\bar{b}cd \vee \bar{a}(b \vee \bar{b})cd$$

Komplement und $x \wedge 1 = x$

$$= b\bar{c}d \vee \bar{b}cd \vee \bar{a}cd$$

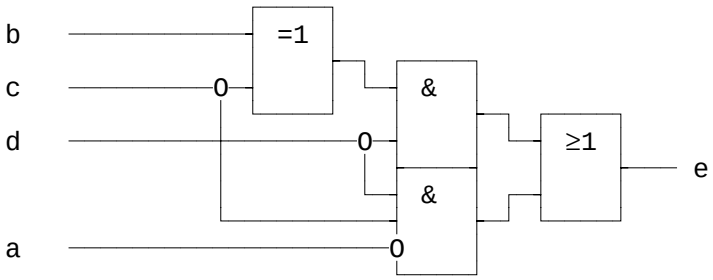
was zu folgendem Gatternetz führt:



Man könnte aber auch schreiben

$$e = (b \not\vee c)d \vee \bar{a}cd$$

und erhielte das Gatternetz



Die Vorgehensweise kann nicht befriedigen. Es bleiben zu viele Fragen offen. Ist man wirklich "geschickt" vorgegangen? Gibt es ein noch "einfacheres" Gatternetz? Wann ist ein Gatternetz "einfacher" als ein anderes?——> Wir suchen einen systematischen Lösungsansatz!

5.1.5. Minimierung

Gegeben sei eine BOOLEsche Funktion in einer beliebigen Schreibweise, gesucht die funktionsgleiche DNF (KNF), der das "einfachste", d. h. kostengünstigste zweistufige AND-OR-Gatternetz (OR-AND-Gatternetz) entspricht.

5.1.5.1. Kosten

Um die Kosten für ein Gatternetz messen zu können, führen wir den sog. Kostenfaktor ein.

Def.: Kostenfaktor K = Summe der Anzahlen der Eingänge der verwendeten AND- und OR-Gatter; ggf. erforderliche Negatoren werden nicht mitgezählt.

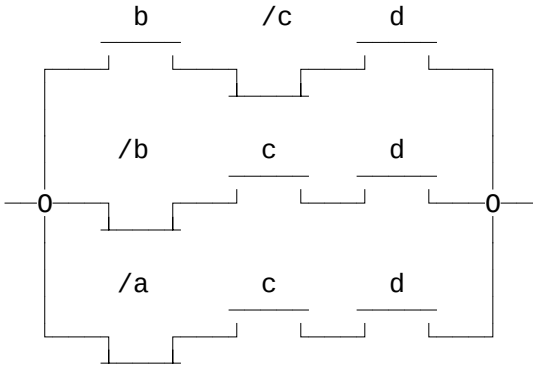
Anm.: Die gewählte Definition geht zurück auf die Kontaktnetze der Schaltalgebra (s. Beispiel 4.2.). Man zählt die Anzahl der Pfade und die Anzahl der Kontakte pro Pfad (DNF) bzw. die Anzahl der Ebenen und die Anzahl der Kontakte pro Ebene (KNF). Da Arbeits- und Ruhekontakte etwa gleichviel kosten, muß zwischen ihnen nicht unterschieden werden.

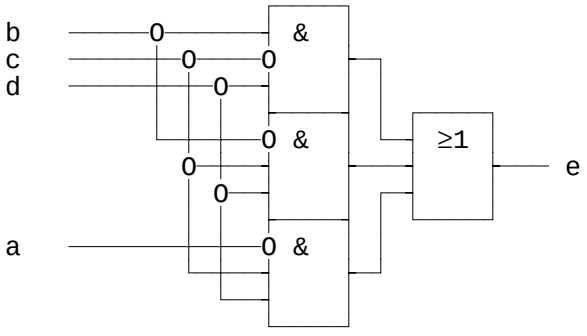
Beispiel 5.11.

In Beispiel 5.10. haben wir die DNF

$e = b\bar{c}d \vee \bar{b}cd \vee \bar{a}cd$

gefunden. Kontakt- und Gatternetz sehen so aus





Der Kostenfaktor ergibt sich zu

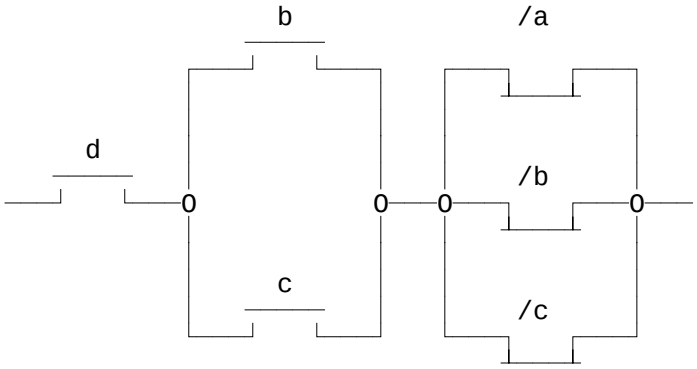
K_{DNF} = 3 Pfade +
3 Kontakte im oberen Pfad +
3 Kontakte im mittleren Pfad +
3 Kontakte im unteren Pfad

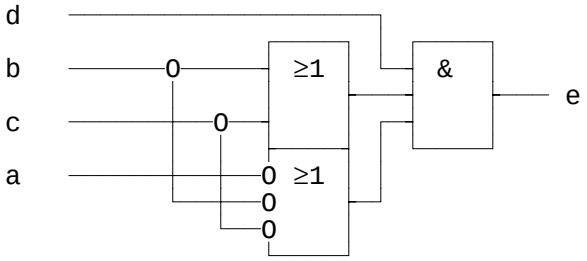
= 3 OR-Eingänge + 3*3 AND-Eingänge = 12

Auf analoge Weise (s. Beispiel 5.2.) findet man die KNF

$e = d(b \vee c)(\bar{a} \vee \bar{b} \vee \bar{c})$

Kontakt- und Gatternetz sehen so aus





Der Kostenfaktor ergibt sich zu

K_{KNF} = 3 Ebenen +
1 Kontakt in der linken Ebene +
2 Kontakte in der mittleren Ebene +
3 Kontakte in der rechten Ebene = 9

bzw. für das Gatternetz

K_{KNF} = 3 OR-Eingänge + 2 AND-Eingänge + 3 AND-Eingänge = 8

Beispiel 5.12.

Analog zu Beispiel 5.11. ergeben sich die Kostenfaktoren für die Gatternetze aus den Beispielen 5.5. bis 5.8. zu

$K_{5.5} = K_{5.6} = 3 + 3 + 3 + 3 + 4 = 16$
$K_{5.7} = K_{5.8} = 2 + 2 + 2 + 3 = 9$

Wir halten fest:

Die Umsetzung der KDNF (KKNF) führt zum "teuersten" Gatternetz.

Die Kosten für das KDNF-Gatternetz und für das KKNF-Gatternetz sind nicht notwendig gleichgroß.

Die Umsetzung einer (nichtkanonischen) DNF (KNF) führt zu einem Gatternetz, das "billiger" als das entsprechende KDNF-Gatternetz (KKNF-Gatternetz) ist.

Die Kosten für das "billigste" DNF-Gatternetz und für das "billigste" KNF-Gatternetz sind ebenfalls nicht notwendig gleichgroß.

Eine mögliche Strategie für das Finden des "billigsten" Gatternetzes könnte so aussehen, daß zunächst getrennt die Kosten für das KDNF- und für das KKNF-Gatternetz ermittelt werden. Anschließend vereinfacht man so gut wie möglich das "billigere" der beiden Gatternetze und erhält so das "billigste" Gatternetz überhaupt. Die Allgemeingültigkeit dieser Strategie darf aber bezweifelt werden. Es ist nicht ausgeschlossen, daß das "billigste" Gatternetz ein DNF-Gatternetz (KNF-Gatternetz) ist, obwohl das KDNF-Gatternetz (KKNF-Gatternetz) "teuerer" ist als das KKNF-Gatternetz (KDNF-Gatternetz). Man wird also nicht umhinkommen, sowohl das "billigste" DNF-Gatternetz als auch das "billigste" KNF-Gatternetz zu ermitteln, um mit dem "billigsten" davon das "billigste" Gatternetz überhaupt zu finden.

5.1.5.2. KARNAUGH-Plan

Der KARNAUGH-Plan hilft, zu einer gegebenen KDNF (KKNF) die minimale (d. h. kostengünstigste) funktionsgleiche DNF (KNF) zu finden.

In Beispiel 5.10 haben wir kennengelernt, daß KDNF durch "geschicktes" Zusammenfassen von Elementarkonjunktionen zu DNF vereinfacht werden können. Das Verfahren bestand darin, Paare von Elementarkonjunktionen K_i und K_j zu suchen, die sich in nur genau einer Variablen x_k unterscheiden, und zu einer (nichtelementaren) Konjunktion zusammenzufassen.

$$\ldots \vee K_i \vee K_j \vee \ldots =$$

$$\ldots \vee \ddot{x}_1 \ldots \ddot{x}_{k-1} \mathbf{x_k} \ddot{x}_{k+1} \ldots \ddot{x}_n \vee \ddot{x}_1 \ldots \ddot{x}_{k-1} \bar{\mathbf{x}}_{\mathbf{k}} \ddot{x}_{k+1} \ldots \ddot{x}_n \vee \ldots =$$

$$\ldots \vee (\mathbf{x_k} \vee \bar{\mathbf{x}}_{\mathbf{k}}) \ddot{x}_1 \ldots \ddot{x}_{k-1} \ddot{x}_{k+1} \ldots \ddot{x}_n \vee \ldots =$$

$$\ldots \vee \ddot{x}_1 \ldots \ddot{x}_{k-1} \ddot{x}_{k+1} \ldots \ddot{x}_n \vee \ldots$$

Anm.: $\ddot{x}_j$ steht für x_j oder $\bar{x}_j$, aber in beiden Konjunktionen gleich!

Das Problem dabei besteht darin, keines dieser Paare zu übersehen. KARNAUGH und VEITCH haben unabhängig voneinander ein Diagramm angegeben, in dem alle möglichen Minterme (Maxterme) einer KDNF (KKNF) so angeordnet sind, daß Minterme (Maxterme), die sich in nur einer Variablen unterscheiden, unmittelbar benachbart sind. Auf diese Weise lassen sich Paare (Quartette, Oktette, ...) von Mintermen (Maxtermen), die sich zusammenfassen lassen, leichter erkennen.

Wir werden hier immer von KARNAUGH-Plan sprechen. Üblich sind auch die Bezeichnungen VEITCH-Diagramm oder KV-Diagramm. Obwohl der KARNAUGH-Plan sowohl zur Darstellung der KDNF als auch der KKNF geeignet ist, werden wir ihn hier immer nur als Darstellungsmittel für die KDNF verwenden.

Eine sehr gründliche und theoretisch exakte Einführung in die Minimierung finden Sie in den Folien Dr. Königs (Folien 4-1 bis 4-133). Wir werden hier die Minimierung anhand eines schon mehrfach verwendeten Beispiels kennenlernen.

Beispiel 5.13.

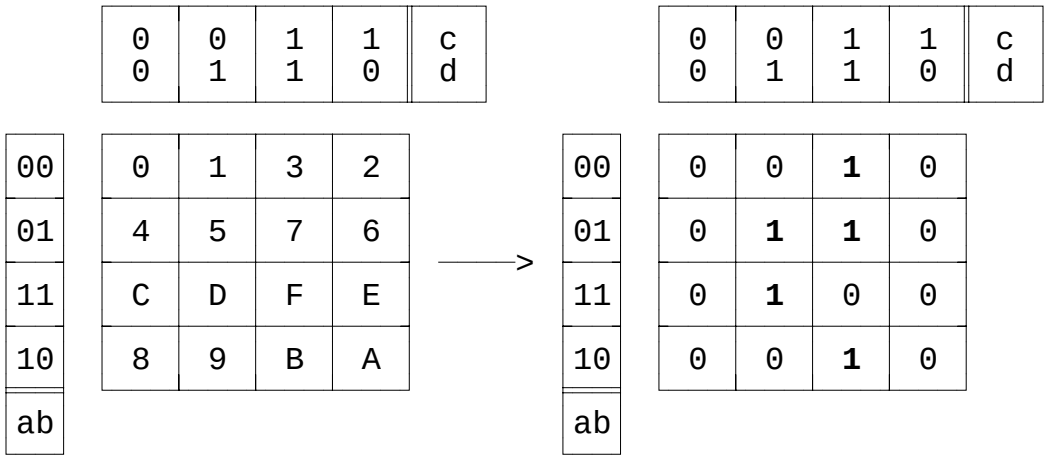
In Beispiel 5.10. haben wir die KDNF

$$\mathbf{e = ab\bar{c}d \vee a\bar{b}cd \vee \bar{a}bcd \vee \bar{a}b\bar{c}d \vee \bar{a}\bar{b}cd}$$
$$\mathbf{= K_D \vee K_B \vee K_7 \vee K_5 \vee K_3}$$

zur DNF

$$\mathbf{e = b\bar{c}d \vee \bar{b}cd \vee \bar{a}cd}$$

vereinfacht. Im KARNAUGH-Plan spiegelt sich das so wieder:



Der KARNAUGH-Plan besteht aus einem zweidimensionalen Feld mit 2^n Plätzen, wenn die darzustellende Funktion $f(\underline{x})$ eine Funktion von n Variablen ist. Jedem Platz ist genau eine der 2^n möglichen Elementarkonjunktionen zugeordnet. Die Zuordnung erfolgt so, daß sich unmittelbar benachbarte Elementarkonjunktionen in genau

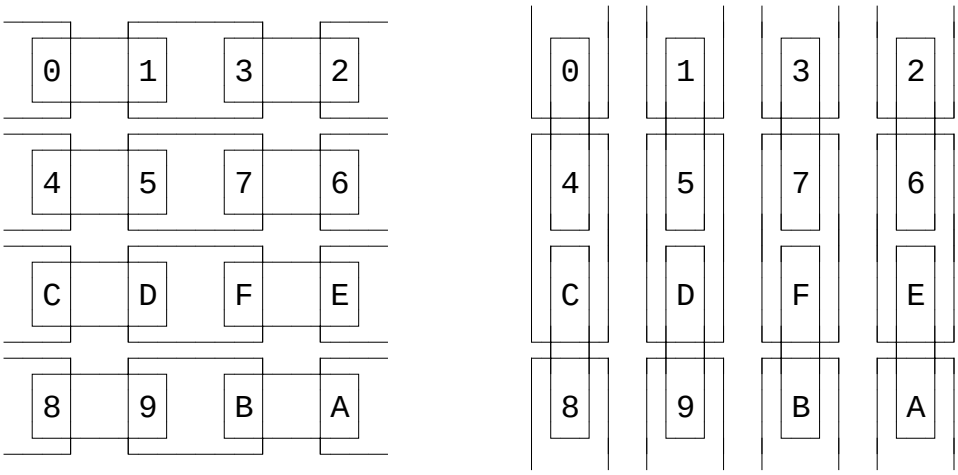
einer Variablen unterscheiden. Dazu wird die Menge der Eingangsvariablen in zwei etwa gleichgroße Teilmengen aufgeteilt, und die möglichen Belegungen der Eingangsvariablen der Teilmengen werden als Randbeschriftung im GRAY-Kode aufgetragen.

Für n = 2, 3 und 4 ist der KARNAUGH-Plan sehr übersichtlich. Ab n = 5 wird er immer unübersichtlicher. In der Literatur finden Sie KARNAUGH-Pläne bis n = 6. Wir werden hier den KARNAUGH-Plan nur bis n = 4 verwenden.

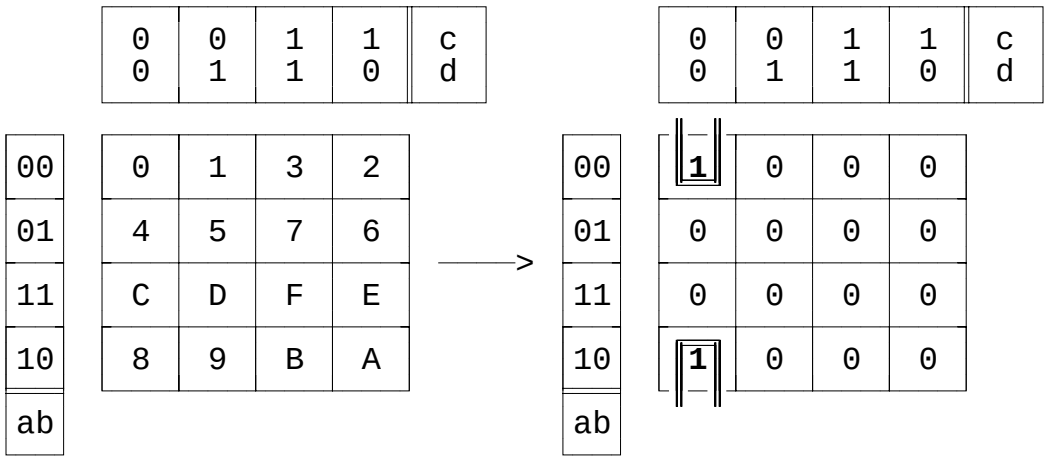
Im konkreten Fall ist n = 4, und der KARNAUGH-Plan hat $2^4 = 16$ Felder. Die Menge der Eingangsvariablen {a,b,c,d} wird in zwei gleichgroße Teilmengen {a,b} und {c,d} aufgeteilt, und die möglichen Belegungen der Eingangsvariablen der beiden Teilmengen werden im GRAY-Kode als Randbeschriftung aufgetragen, z. B. in der Reihenfolge 00, 01, 11, 10.

Der oben links dargestellte Plan dient nur der besseren Verständigung. Wir werden ihn später nicht mehr benötigen. In diesem Plan sind die 16 Plätze mit dem Hexadezimaläquivalent der jeweiligen Eingangsbelegung abcd gekennzeichnet (das Hexadezimaläquivalent der Eingangsbelegung, die eine bestimmte Konjunktion erfüllt, wurde bereits in Beispiel 5.10. als Ordnungskriterium verwendet!). Der oben rechts dargestellte Plan unterscheidet sich vom linken Plan dadurch, daß in den 16 Plätzen die Funktionswerte der ihnen zugeordneten Elementarkonjunktionen eingetragen sind (vgl. die "ausführliche" Schreibweise der KDNF!). Nur diesen Plan werden wir zukünftig brauchen.

Benachbart sind die in der gleichen Reihe unmittelbar nebeneinander befindlichen Plätze, z. B. 0-1, 1-3, 3-2 aber auch 2-0! Benachbart sind weiterhin die in der gleichen Spalte unmittelbar unter-/übereinander befindlichen Plätze, z. B. 0-4, 4-C, C-8 aber auch 8-0!



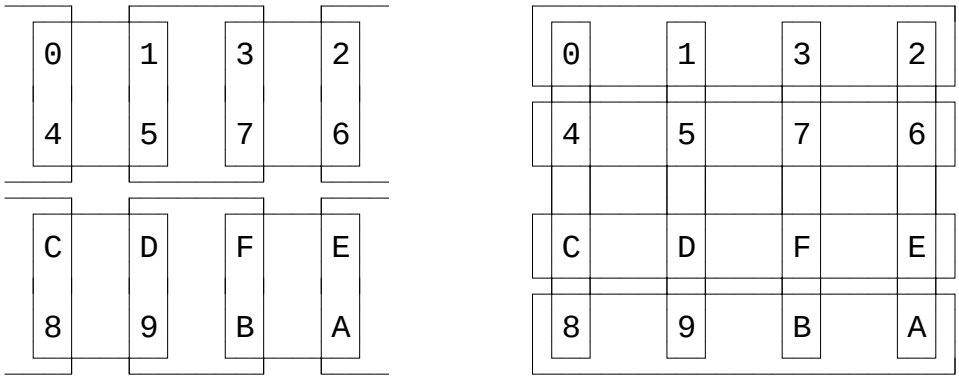
Zusammenfassen lassen sich benachbarte Plätze dann, wenn sie (im rechten Plan) den Funktionswert 1 tragen, also etwa

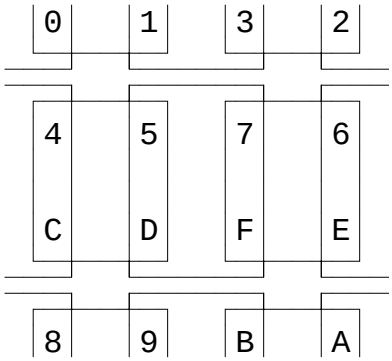


$K_8 \vee K_0 = a\bar{b}\bar{c}d \vee \bar{a}\bar{b}\bar{c}d = (a \vee \bar{a})\bar{b}\bar{c}d = \bar{b}\bar{c}d$

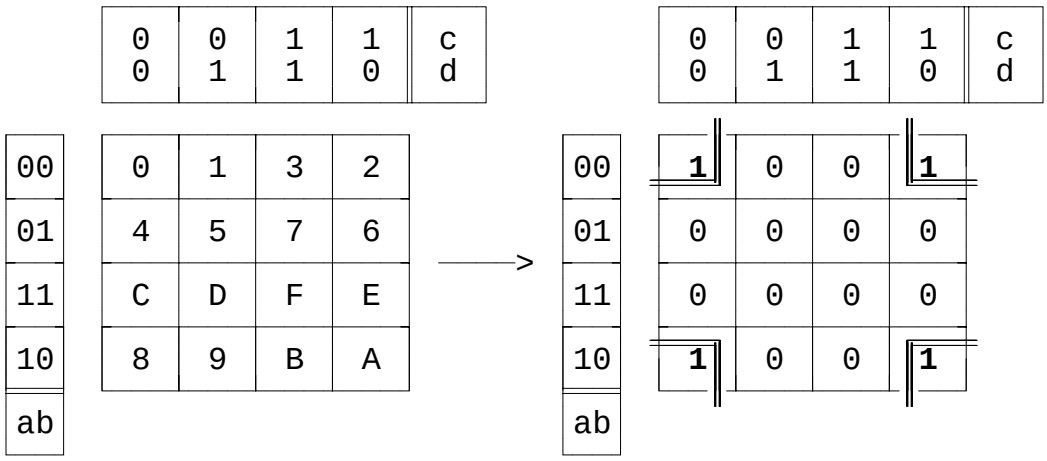
Neben Paaren benachbarter Plätze sind im KARNAUGH-Plan aber auch Quartett, Oktette und ggf. weitere Mengen benachbarter Plätze erkennbar, wobei jeweils nur Platzmengen M benachbart sein können, für die gilt $|M| = 2^i$, i = 1,2,3,...

Wir erkennen etwa die Quartette 0-1-4-5, 1-3-5-7, 2-3-6-7 aber auch 0-2-4-6 und sogar 0-2-8-A.





Das Quartett 0-2-8-A wollen wir genauer betrachten:



$K_A \vee K_8 \vee K_2 \vee K_0 = a\bar{b}cd \vee a\bar{b}\bar{c}d \vee \bar{a}\bar{b}cd \vee \bar{a}\bar{b}\bar{c}d$

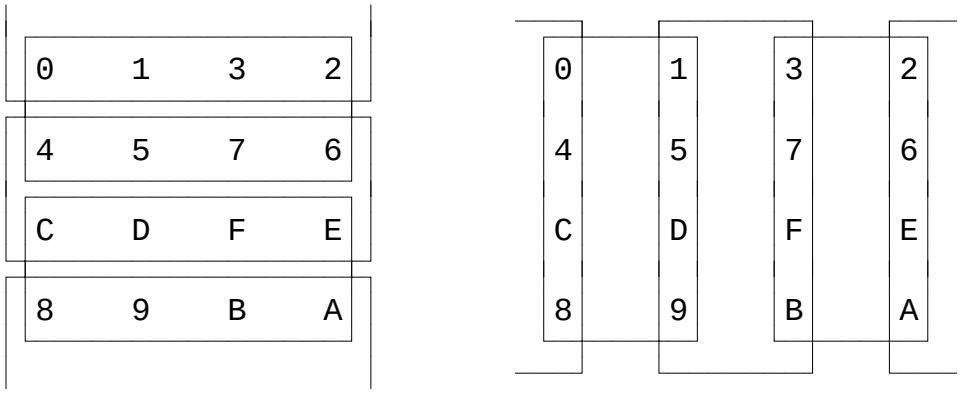
$= \bar{b}d(ac \vee a\bar{c} \vee \bar{a}c \vee \bar{a}\bar{c})$

$= \bar{b}d(a(c \vee \bar{c}) \vee \bar{a}(c \vee \bar{c}))$

$= \bar{b}d(a \vee \bar{a})(c \vee \bar{c})$

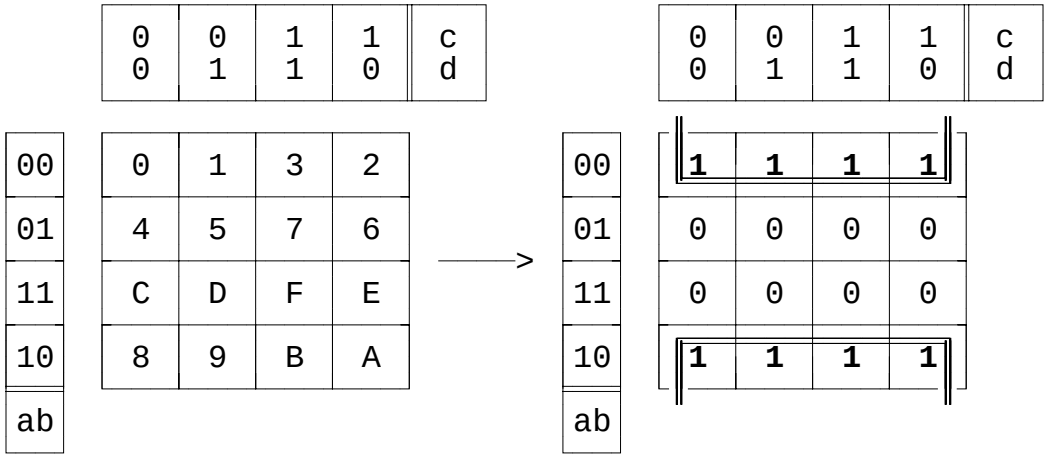
$= \bar{b}d$

Analog dazu finden wir die möglichen Oktette



z. B.

0-1-2-3-4-5-6-7, 0-1-4-5-8-9-C-D aber auch 0-1-2-3-8-9-A-B.



$K_B \vee K_A \vee K_9 \vee K_8 \vee K_3 \vee K_2 \vee K_1 \vee K_0 =$

$a\bar{b}cd \vee a\bar{b}c\bar{d} \vee a\bar{b}\bar{c}d \vee a\bar{b}\bar{c}\bar{d} \vee \bar{a}\bar{b}cd \vee \bar{a}\bar{b}c\bar{d} \vee \bar{a}\bar{b}\bar{c}d \vee \bar{a}\bar{b}\bar{c}\bar{d} =$

$\bar{b}(acd \vee ac\bar{d} \vee a\bar{c}d \vee a\bar{c}\bar{d} \vee \bar{a}cd \vee \bar{a}c\bar{d} \vee \bar{a}\bar{c}d \vee \bar{a}\bar{c}\bar{d}) =$

$\bar{b}(a(cd \vee c\bar{d} \vee \bar{c}d \vee \bar{c}\bar{d}) \vee \bar{a}(cd \vee c\bar{d} \vee \bar{c}d \vee \bar{c}\bar{d})) =$

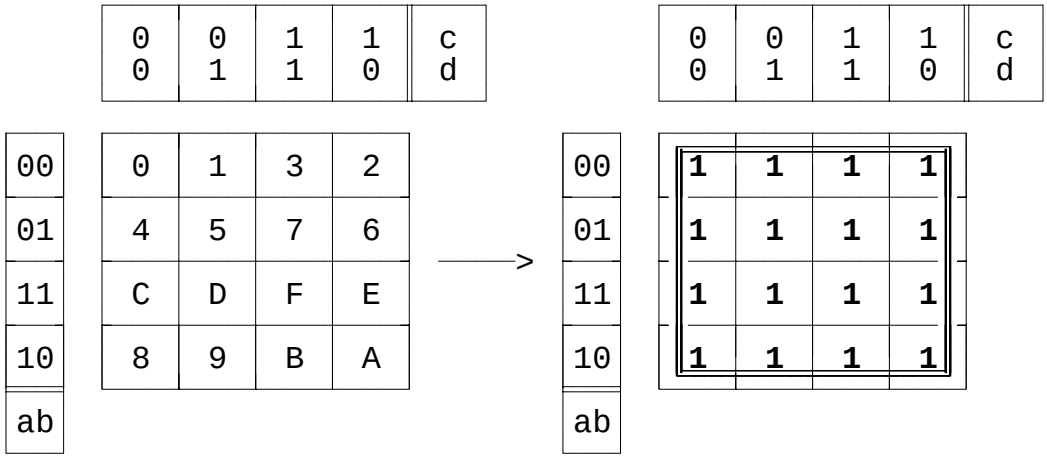
$\bar{b}(a(c(d \vee \bar{d}) \vee \bar{c}(d \vee \bar{d})) \vee \bar{a}(c(d \vee \bar{d}) \vee \bar{c}(d \vee \bar{d}))) =$

$\bar{b}(a(c \vee \bar{c})(d \vee \bar{d}) \vee \bar{a}(c \vee \bar{c})(d \vee \bar{d})) =$

$\bar{b}(a \vee \bar{a})(c \vee \bar{c})(d \vee \bar{d}) =$

$\bar{b}$

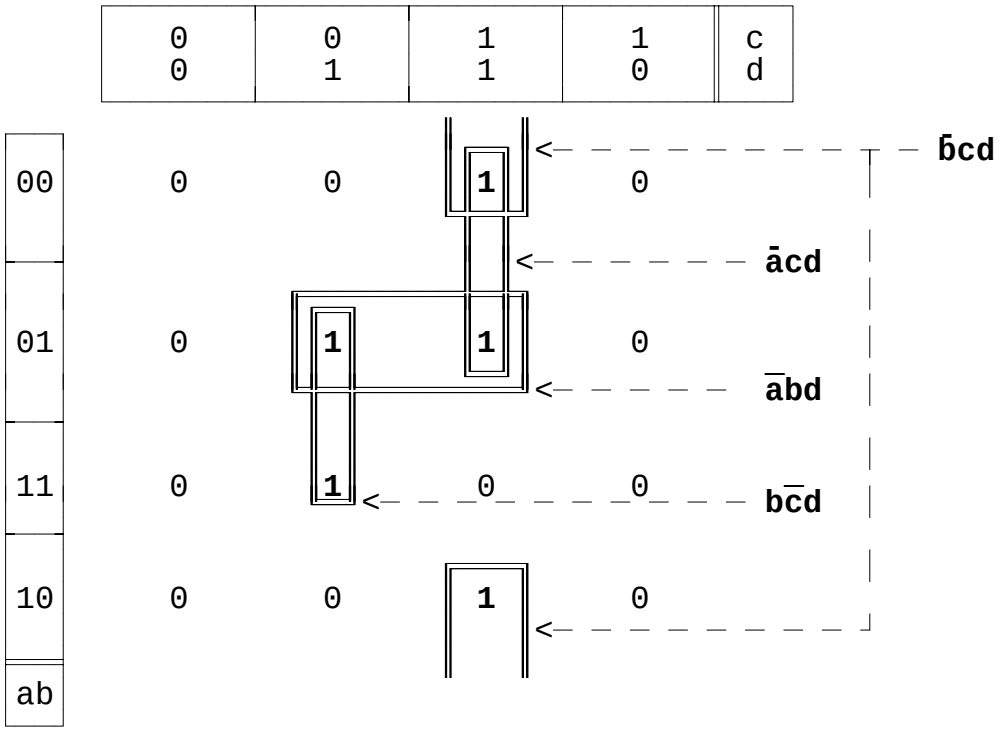
und schließlich können auch alle 16 Felder den Funktionswert 1 tragen und zusammengefaßt werden:



$K_F \vee \ldots \vee K_0 = 1$

Wir halten fest: Je mehr Elementarkonjunktionen zu einer (nicht-elementaren) Konjunktion zusammengefaßt werden können, desto "billiger" ist deren Realisierung. Anstelle der Bezeichnung "Zusammenfassung" verwendet man theoretisch exakter auch die Bezeichnung "Implikant". Größtmögliche Zusammenfassungen heißen auch "Primimplikanten".

Wir wollen nun die gewonnenen Erkenntnisse auf des oben angegebene Beispiel anwenden und finden vier zusammenfaßbare Paare:



e = bc̄d ∨ b̄cd ∨ ācd ∨ **ābd**

Die letzte, fett markierte Konjunktion haben wir bei der oben, durch "geschicktes" Zusammenfassen ermittelten Lösung nicht gefunden! Haben wir sie übersehen?

Da für jede BOOLEsche Funktion nur genau eine KDNF existiert, sind die beiden Lösungen dann korrekt, wenn sie sich in ein und dieselbe KDNF überführen lassen. Prüfen Sie das nach!

Es gibt offensichtlich Zusammenfassungen, die nicht zwingend in die Lösung übernommen werden müssen. Welche Zusammenfassungen müssen übernommen werden, welche dürfen weggelassen werden?

Jede aus einer KDNF abgeleitete DNF ist funktionsgleich der KDNF, wenn die Konjunktionen der DNF alle Elementarkonjunktionen der KDNF überdecken. Das ist eine notwendige Bedingung!

Um die minimale DNF zu finden, muß diejenige Überdeckung gesucht werden, die mit einer minimalen Anzahl von maximal großen Zusammenfassungen alle Elementarkonjunktionen überdeckt.

In der Mehrzahl der Fälle führt folgende Strategie zum Ziel:

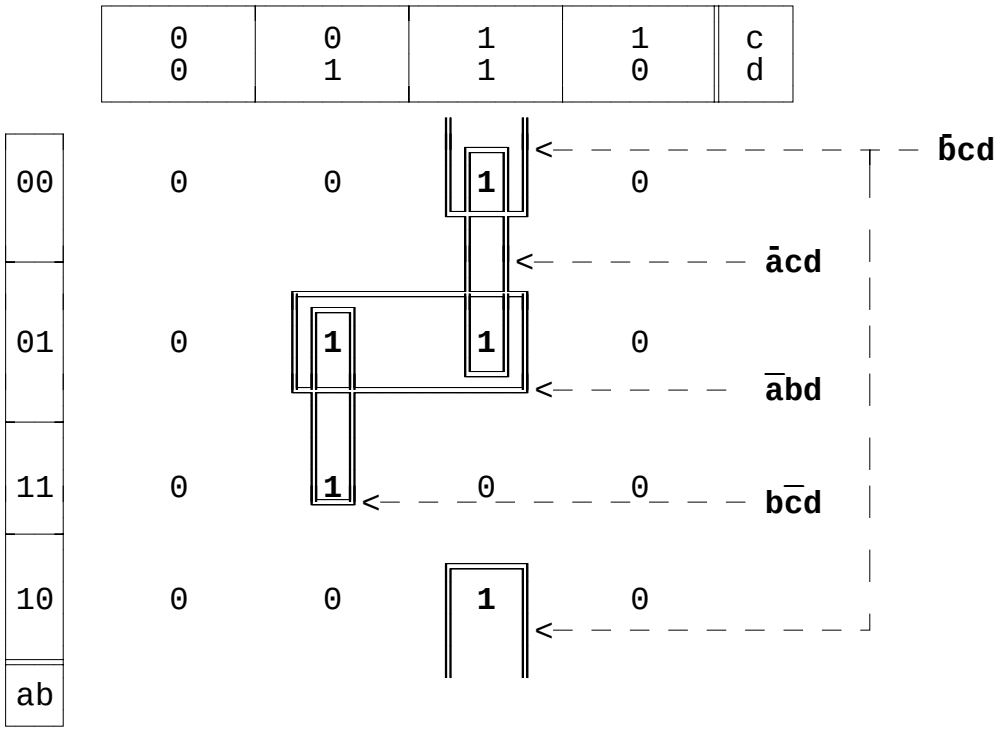
1. Für jede Elementarkonjunktion Eintragen aller größtmöglichen Zusammenfassungen in den KARNAUGH-Plan.

2. Suchen solcher Elementarkonjunktionen, die nur durch eine einzige Zusammenfassung überdeckt werden und Übernehmen dieser Zusammenfassungen ("KPI, Kernprimimplikanten") in die Lösung.

Falls damit bereits alle Elementarkonjunktionen überdeckt sind, ist die Lösung gefunden.

3. Für die noch nicht überdeckten Elementarkonjunktionen Auswählen einer minimalen Überdeckung.

Für unser Beispiel ergibt sich nach Schritt 1 der bereits angegebene KARNAUGH-Plan:



Schritt 2 liefert:

K_D = abc̄d wird überdeckt durch bc̄d

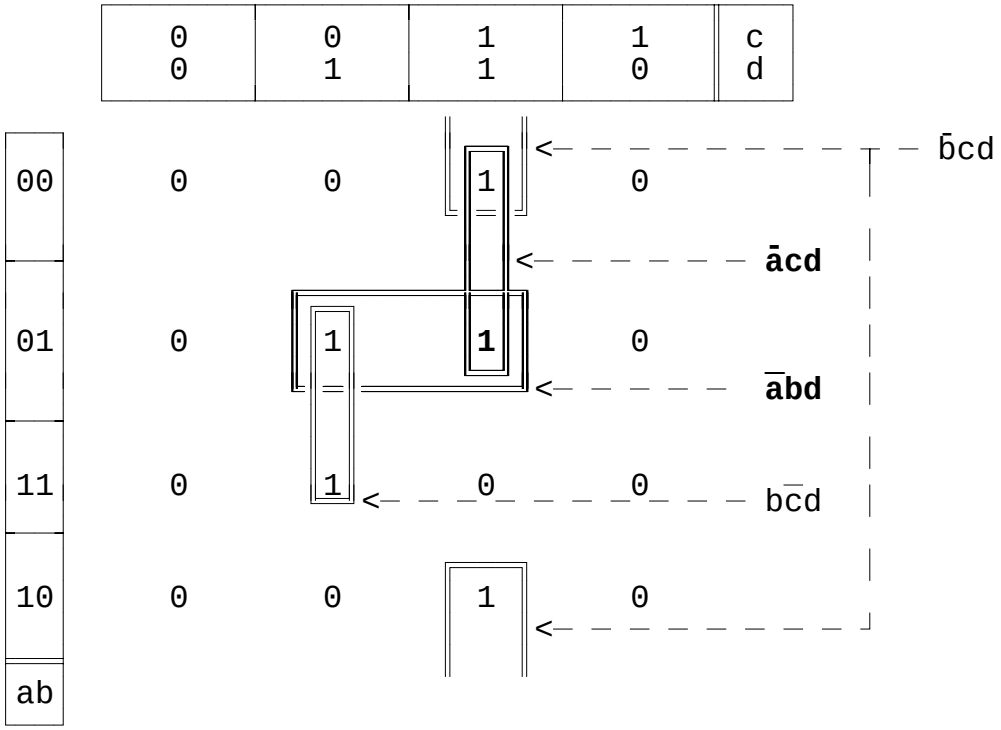
K_B = ab̄cd wird überdeckt durch b̄cd

K_7 = ābcd wird überdeckt durch ācd und ābd

K_5 = ābc̄d wird überdeckt durch ābd und bc̄d

K_3 = āb̄cd wird überdeckt durch ācd und b̄cd

Die Elementarkonjunktionen K_D und K_B werden nur durch jeweils eine einzige Zusammenfassung überdeckt. Diese Zusammenfassungen müssen in die Lösung übernommen werden (nicht mehr fett markiert). Die beiden Zusammenfassungen überdecken die Elementarkonjunktionen K_D, K_B, K_5 und K_3 (nicht mehr fett markiert), um die wir uns fortan nicht mehr kümmern müssen. Die Elementarkonjunktion K_7 ist als einzige Elementarkonjunktion noch nicht überdeckt.



In Schritt 3 haben wir die Wahl zwischen zwei Zusammenfassungen, die gleichermaßen die Konjunktion K_7 überdecken, und erhalten zwei gleichgute Lösungen (Kostenfaktor = 12):

$e = \bar{a}bd \vee \bar{b}cd \vee b\bar{c}d$

$e = \bar{a}cd \vee \bar{b}cd \vee b\bar{c}d$

Schritt 3 ist nicht in jedem Falle so trivial. Es kommt vor, daß in Schritt 1 überhaupt keine Elementarkonjunktion gefunden wird, die nur durch eine einzige Zusammenfassung überdeckt wird. Dann muß die gesamte Lösung in Schritt 3 gefunden werden. PETRICK hat dafür ein algebraisches Verfahren angegeben.

Der PETRICK-Ausdruck PA ist eine aussagenlogische Formulierung, die angibt, welche Zusammenfassungen in die Lösungen übernommen werden müssen bzw. können. Man bezeichnet dazu die Zusammenfassungen geeignet und stellt die Verhältnisse tabellarisch dar. Für unser Beispiel ergibt sich:

		K_D	K_B	K_7	K_5	K_3
		$ab\bar{c}d$	$a\bar{b}cd$	$\bar{a}bcd$	$\bar{a}b\bar{c}d$	$\bar{a}\bar{b}cd$
p_1	$\bar{a}bd$			x	x	
p_2	$\bar{a}cd$			x		x
p_3	$\bar{b}cd$		x			x
p_4	$b\bar{c}d$	x			x	

Ein Kreuz in der Tabelle (Zeile p_i, Spalte K_j) bedeutet, daß der Primimplikant p_i die Elementarkonjunktion K_j überdeckt.

PETRICK definiert für jeden Primimplikanten p_i eine BOOLEsche Variable e_i mit

$$e_i = \begin{cases} 1 & \text{wenn } p_i \text{ eine Elementarkonjunktion } K_j \text{ überdeckt} \\ 0 & \text{wenn } p_i \text{ eine Elementarkonjunktion } K_j \text{ nicht überdeckt} \end{cases}$$

und gibt für jede Elementarkonjunktion die Alternativen der sie überdeckenden Primimplikanten in Form eines aussagenlogischen Ausdrucks (PETRICK-Ausdruck) an. Für unser Beispiel ergibt sich:

$PA_{K_D} = e_4$ - um K_D zu überdecken, muß p_4 in die Lösung übernommen werden

$PA_{K_B} = e_3$ - um K_B zu überdecken, muß p_3 in die Lösung übernommen werden

$PA_{K_7} = e_1 \vee e_2$ - um K_7 zu überdecken, müssen entweder p_1 oder p_2 in die Lösung übernommen werden

$PA_{K_5} = e_1 \vee e_4$ - um K_5 zu überdecken, müssen entweder p_1 oder p_2 in die Lösung übernommen werden

$PA_{K_3} = e_2 \vee e_3$ - um K_3 zu überdecken, müssen entweder p_2 oder p_3 in die Lösung übernommen werden

Da alle Primimplikanten K_j überdeckt werden müssen, folgt für den PETRICK-Ausdruck der gesamten Funktion die Konjunktion der PETRICK-Ausdrücke der Elementarkonjunktionen. Im Beispiel ergibt sich:

$$PA = PA_{K_D} \wedge PA_{K_B} \wedge PA_{K_7} \wedge PA_{K_5} \wedge PA_{K_3} =$$

$$e_4 \wedge e_3 \wedge (e_1 \vee e_2) \wedge (e_1 \vee e_4) \wedge (e_2 \vee e_3) =$$

$$e_4 \wedge e_3 \wedge (e_1e_2 \vee e_1e_3 \vee e_2e_4 \vee e_2e_3e_4) =$$

$$e_1e_2e_3e_4 \vee e_1e_3e_4 \vee e_2e_3e_4$$

Es gibt drei Lösungen

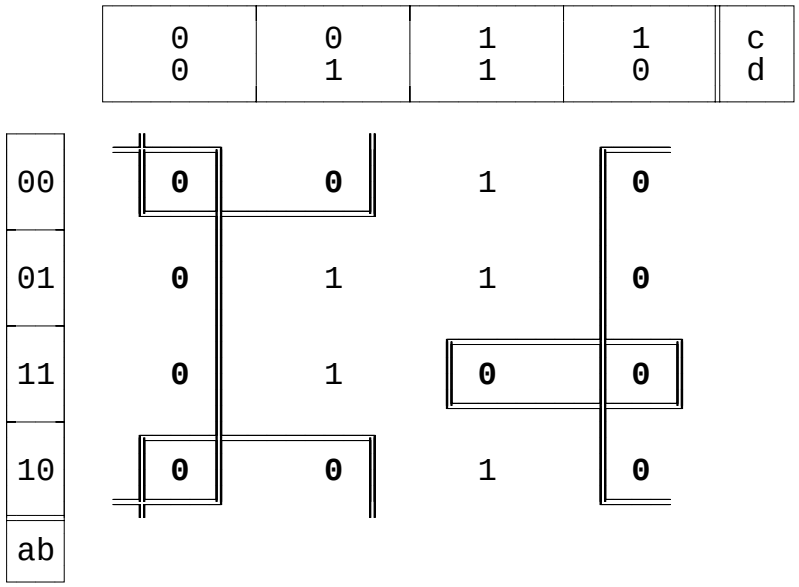
$e = p_1 \vee p_2 \vee p_3 \vee p_4 = \bar{a}bd \vee \bar{a}cd \vee \bar{b}cd \vee b\bar{c}d$ (K = 16)

$e = p_1 \vee p_3 \vee p_4 = \bar{a}bd \vee \bar{b}cd \vee b\bar{c}d$ (K = 12)

$e = p_2 \vee p_3 \vee p_4 = \bar{a}cd \vee \bar{b}cd \vee b\bar{c}d$ (K = 12)

die kostenmäßig zu bewerten sind. Der Kostenfaktor nimmt für die erste Lösung den Wert 16 an, für die beiden anderen den Wert 12. Wenn keine weiteren Kriterien zu beachten sind, sind die beiden letzten Lösungen gleichgut und billiger als die erste Lösung. Man wird folglich eine der beiden letzten Lösungen auswählen.

Wir wollen abschließend die ganze Prozedur noch für die minimale KNF exerzieren. Wir gehen so vor, daß wir die minimale DNF für den negierten Funktionswert ermitteln und diese dann mit Hilfe der DeMorganschen Regel in die minimale KNF umschreiben.



$\bar{e} = p_1 \vee p_2 \vee p_3 = \bar{d} \vee \bar{b}\bar{c} \vee abc$

Den Aufwand für den PETRICK-Ausdruck können wir uns sparen. Die drei Primimplikanten sind KPI (Kernprimimplikanten), da jeder von ihnen mindestens eine Elementarkonjunktion exklusiv überdeckt:

p : K , K , K , K
 1   2   4   5   C

p : K , K
 2   1   9

p : K
 3   F

Es gibt nur eine minimale Lösung:

$e = \overline{\bar{d} \vee \bar{b}\bar{c} \vee abc} = d(b \vee c)(\bar{a} \vee \bar{b} \vee \bar{c})$ (K = 8)

5.1.6. Technische Realisierung von Gatternetzen

Gegeben sei eine BOOLEsche Funktion in einer beliebigen Schreibweise, gesucht ein funktionsgleiches Gatternetz aus vorgegebenen Gattern.

5.1.6.1. Gatternetze aus NAND-Gattern

Schritt 1 - Umwandeln der gegebenen Schreibweise in eine (vorzugsweise minimale) DNF

Schritt 2 - Zweifaches Negieren der DNF

Schritt 3 - Auflösen der unteren Negation nach einer der DeMorganschen Regeln

Schritt 4 - Abbilden der gefundenen Schreibweise auf ein Gatternetz aus NAND-Gattern

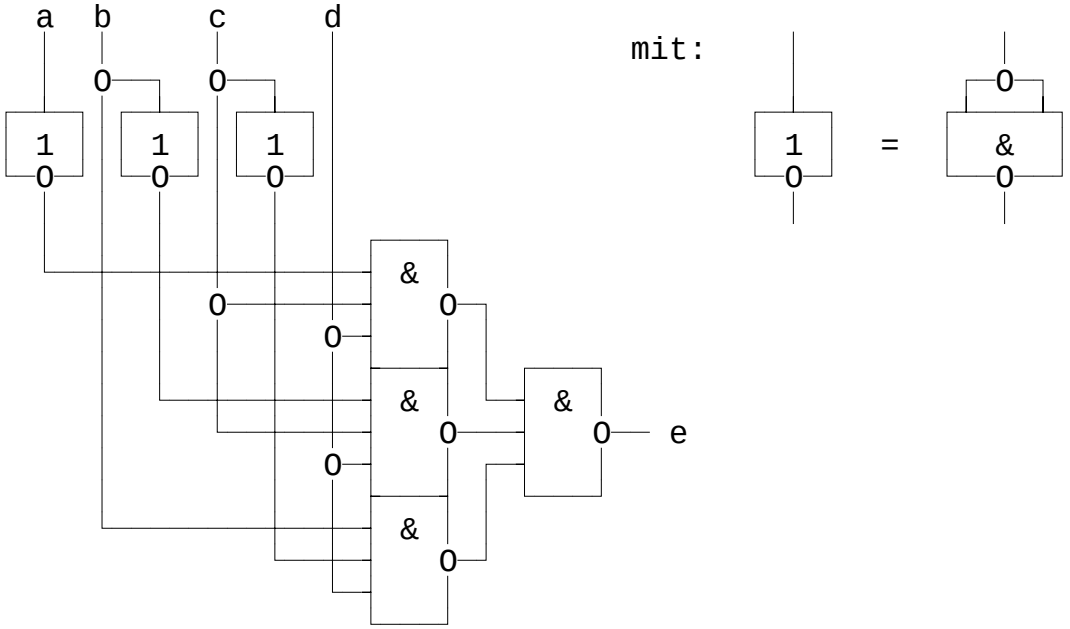
Für Beispiel 5.13. ergibt sich:

Schritt 1 $e = \bar{a}cd \vee b\bar{c}d \vee b\bar{c}\bar{d}$ (eine der beiden Lösungen)

Schritt 2 $e = \overline{\bar{a}cd \vee b\bar{c}d \vee b\bar{c}\bar{d}}$

Schritt 3 $e = \overline{\bar{a}cd} \wedge \overline{b\bar{c}d} \wedge \overline{b\bar{c}\bar{d}}$

Schritt 4



5.1.6.2. Gatternetze aus NOR-Gattern

Schritt 1 - Umwandeln der gegebenen Schreibweise in eine (vorzugsweise minimale) KNF

Schritt 2 - Zweifaches Negieren der KNF

Schritt 3 - Auflösen der unteren Negation nach einer der DeMorganschen Regeln

Schritt 4 - Abbilden der gefundenen Schreibweise auf ein Gatternetz aus NOR-Gattern

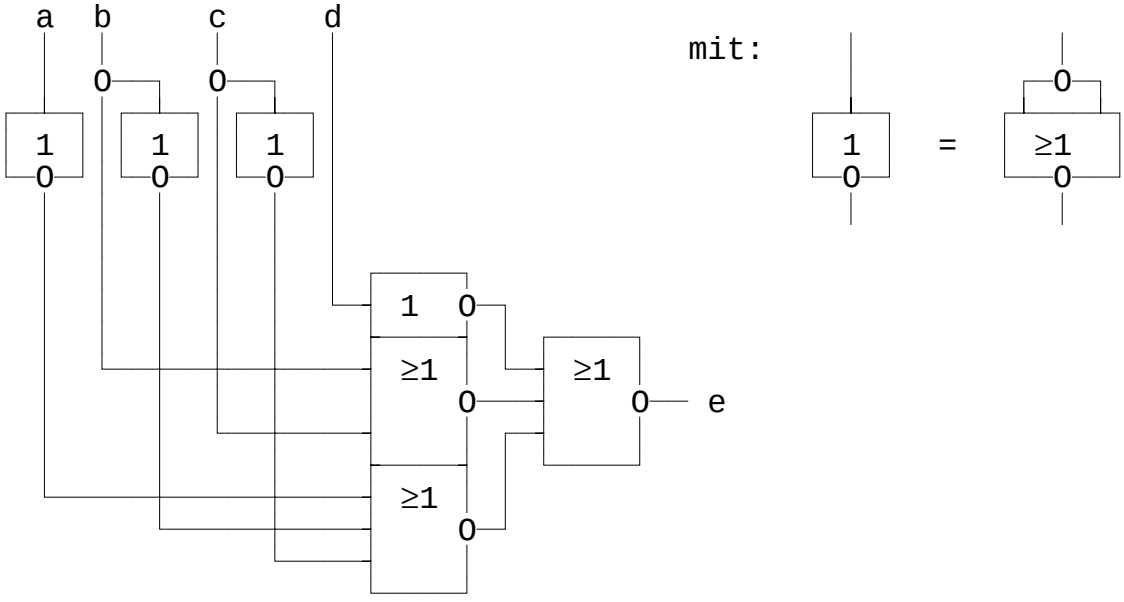
Für Beispiel 5.13. ergibt sich:

Schritt 1 $e = \overline{\overline{d(b \vee c)(\bar{a} \vee \bar{b} \vee \bar{c})}}$

Schritt 2 $e = \overline{\overline{d(b \vee c)(\bar{a} \vee \bar{b} \vee \bar{c})}}$

Schritt 3 $e = \overline{\bar{d} \vee \overline{b \vee c} \vee \overline{\bar{a} \vee \bar{b} \vee \bar{c}}}$

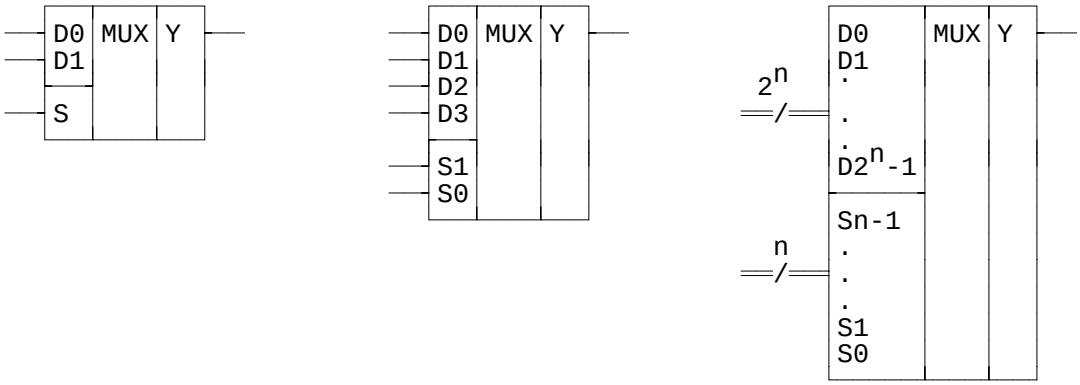
Schritt 4



5.1.6.3. Gatternetze aus 2-zu-1-Multiplexern

Ein 2^n-zu-1-Multiplexer oder 2^n-zu-1-Datenselektor (n = 1,2,...) ist eine kombinatorische Schaltung, mit deren Hilfe eine von 2^n Eingangsvariablen d nach Maßgabe einer n Bit breiten Adresse s ausgewählt und auf den Ausgang der Schaltung durchgereicht wird.

2-zu-1-Multiplexer **4-zu-1-Multiplexer** ... **2^n-zu-1-Multiplexer**



$$y_2 = \bar{s}d_0 \vee sd_1 \qquad y_4 = \bar{s}_1\bar{s}_0d_0 \vee \bar{s}_1s_0d_1 \vee s_1\bar{s}_0d_2 \vee s_1s_0d_3$$

$$y_{2^n} = \overline{s_{n-1}...s_1}\bar{s}_0d_0 \vee \overline{s_{n-1}...s_1}s_0d_1 \vee ... \vee s_{n-1}...s_1s_0d_{2^n-1}$$

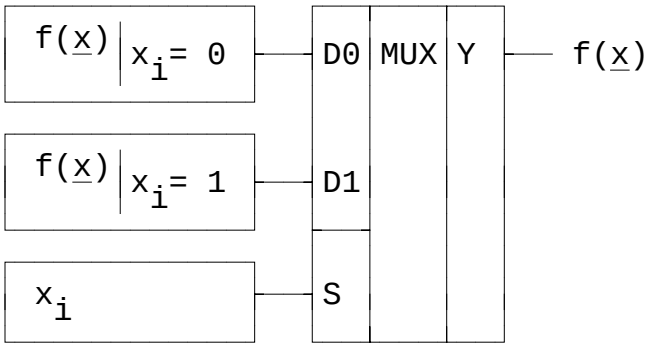
$0.....01 \wedge$!!!

Der Entwicklungssatz der BOOLEschen Algebra ähnelt in seiner Struktur der Funktionsbeschreibung eines 2-zu-1-Multiplexers

Kofaktor Kofaktor

$$f(\underline{x}) = \overline{x_i} \wedge f(\underline{x})\big|_{x_i=0} \vee x_i \wedge f(\underline{x})\big|_{x_i=1}$$

$$y = \bar{s} \wedge d_0 \vee s \wedge d_1$$



Die beiden Kofaktoren lassen sich rekursiv solange nach den anderen Variablen entwickeln, bis eine Abbildung der BOOLEschen Funktion auf ein Gatternetz aus ausschließlich 2-zu-1-Multiplexern möglich wird.

Das gewonnene Gatternetz ist abhängig von der Reihenfolge der Variablen, nach denen entwickelt wird.

Für Beispiel 5.13. ergibt sich für die Entwicklungsreihenfolge a, b, c, d folgendes Gatternetz:

Schritt 1 - eine der beiden oben gefundenen Lösungen

$e = \bar{a}cd \vee \bar{b}cd \vee b\bar{c}d$

Schritt 2 - Entwickeln der Funktion nach a

$e = \bar{a}(bd \vee cd) \vee a(\bar{b}cd \;\Sigma\; b\bar{c}d)$

Schritt 3 - Entwickeln der Kofaktoren nach b

$e = \bar{a}(\bar{b}\wedge cd \vee b\wedge d) \vee a(\bar{b}\wedge cd \vee b\wedge\bar{c}d)$

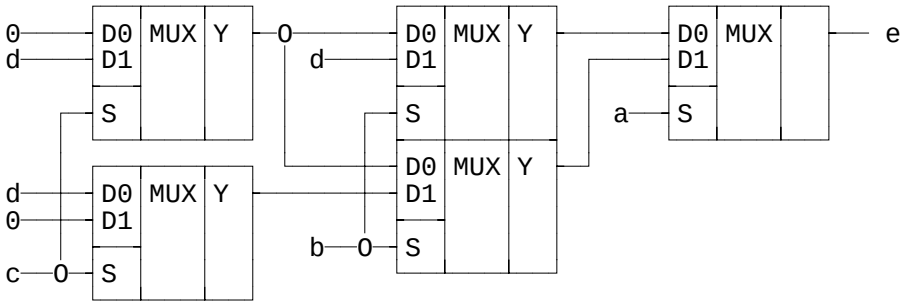
Schritt 4 - Entwickeln der Kofaktoren nach c

$e = \bar{a}(\bar{b}(\bar{c}\wedge 0 \vee c\wedge d) \vee b\wedge d) \vee a(\bar{b}(\bar{c}\wedge 0 \vee c\wedge d) \vee b(\bar{c}\wedge d \vee c\wedge 0))$

Schritt 5 - Entwickeln der Kofaktoren nach d

entfällt

Schritt 6 - Gatternetz



Wählt man die Entwicklungsreihenfolge d, c, b, a ergibt sich

Schritt 2 - Entwickeln der Funktion nach d

$e = \bar{d}\wedge 0 \vee d(\bar{a}c \vee \bar{b}c \vee b\bar{c})$

Schritt 3 - Entwickeln der Kofaktoren nach c

$e = \bar{d}\wedge 0 \vee d(\bar{c}\wedge b \vee c(\bar{a} \vee \bar{b}))$

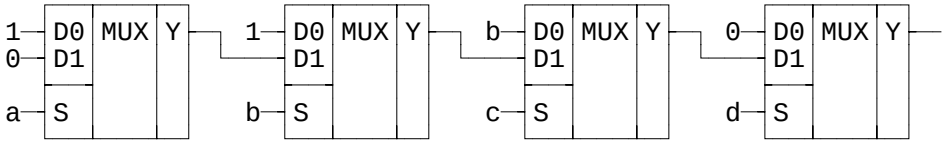
Schritt 4 - Entwickeln der Kofaktoren nach b

$e = \bar{d}\wedge 0 \vee d(\bar{c}\wedge b \vee c(\bar{b}\wedge 1 \vee b\vee\bar{a}))$

Schritt 5 - Entwickeln der Kofaktoren nach a

$e = \bar{d}\wedge 0 \vee d(\bar{c}\wedge b \vee c(\bar{b}\wedge 1 \vee b(\bar{a}\wedge 1 \vee a\wedge 0)))$

Schritt 6 - Gatternetz



5.1.6.4. Realisierung einer n-stelligen BOOLEschen Funktion mit einem 2^n-zu-1-Multiplexer

Die ausführliche Schreibweise der KDNF einer n-stelligen BOOLEschen Funktion hat Ähnlichkeit mit der Funktionsbeschreibung eines 2^n-zu-1-Multiplexers.

BOOLEsche Funktion	**Multiplexer**
$y = \bar{x}_1\bar{x}_2\ldots\bar{x}_n f(00\ldots0) \vee$	$y = \overline{s_{n-1}}\,\overline{s_{n-2}}\ldots\bar{s}_0 d_0 \vee$
$\bar{x}_1\bar{x}_2\ldots x_n f(00\ldots1) \vee$	$\overline{s_{n-1}}\,\overline{s_{n-2}}\ldots s_0 d_1 \vee$
$\vdots$	$\vdots$
$x_1x_2\ldots x_n f(11\ldots1)$	$s_{n-1}s_{n-2}\ldots s_0 d_{2^n-1}$

Mit der Zuordnung

$s_{n-1} = x_1,\ s_{n-2} = x_2,\ \ldots\ ,\ s_0 = x_n$

$d_0 = f(00\ldots0),\ d_1 = f(00\ldots1),\ \ldots\ ,\ d_{2^n-1} = f(11\ldots1)$

gelingt es offenbar, dem Multiplexer die gewünschte BOOLEsche Funktion "beizubringen", d. h. ihn so zu "programmieren".

Für die 4-stellige BOOLEsche Funktion aus Beispiel 5.13. benötigen wir einen 2^4-zu-1-Multiplexer, d. h. einen 16-zu-1-Multiplexer.

BOOLEsche Funktion	**Multiplexer**
$e = \bar{a}\bar{b}\bar{c}\bar{d}\wedge0 \vee$	$y = \bar{s}_3\bar{s}_2\bar{s}_1\bar{s}_0\wedge d_0 \vee$
$\bar{a}\bar{b}\bar{c}d\wedge0 \vee$	$\bar{s}_3\bar{s}_2\bar{s}_1 s_0\wedge d_1 \vee$
$\bar{a}\bar{b}c\bar{d}\wedge0 \vee$	$\bar{s}_3\bar{s}_2 s_1\bar{s}_0\wedge d_2 \vee$
$\bar{a}\bar{b}cd\wedge1 \vee$	$\bar{s}_3\bar{s}_2 s_1 s_0\wedge d_3 \vee$
$\bar{a}b\bar{c}\bar{d}\wedge0 \vee$	$\bar{s}_3 s_2\bar{s}_1\bar{s}_0\wedge d_4 \vee$
$\bar{a}b\bar{c}d\wedge1 \vee$	$\bar{s}_3 s_2\bar{s}_1 s_0\wedge d_5 \vee$
$\bar{a}bc\bar{d}\wedge0 \vee$	$\bar{s}_3 s_2 s_1\bar{s}_0\wedge d_6 \vee$
$\bar{a}bcd\wedge1 \vee$	$\bar{s}_3 s_2 s_1 s_0\wedge d_7 \vee$

$a\bar{b}\bar{c}\bar{d} \wedge 0 \vee$ $s_3\bar{s}_2\bar{s}_1\bar{s}_0 \wedge d_8 \vee$

$a\bar{b}\bar{c}d \wedge 0 \vee$ $s_3\bar{s}_2\bar{s}_1 s_0 \wedge d_9 \vee$

$a\bar{b}c\bar{d} \wedge 0 \vee$ $s_3\bar{s}_2 s_1\bar{s}_0 \wedge d_{10} \vee$

$a\bar{b}cd \wedge 1 \vee$ $s_3\bar{s}_2 s_1 s_0 \wedge d_{11} \vee$

$ab\bar{c}\bar{d} \wedge 0 \vee$ $s_3 s_2\bar{s}_1\bar{s}_0 \wedge d_{12} \vee$

$ab\bar{c}d \wedge 1 \vee$ $s_3 s_2\bar{s}_1 s_0 \wedge d_{13} \vee$

$abc\bar{d} \wedge 0 \vee$ $s_3 s_2 s_1\bar{s}_0 \wedge d_{14} \vee$

$abcd \wedge 0$ $s_3 s_2 s_1 s_0 \wedge d_{15}$

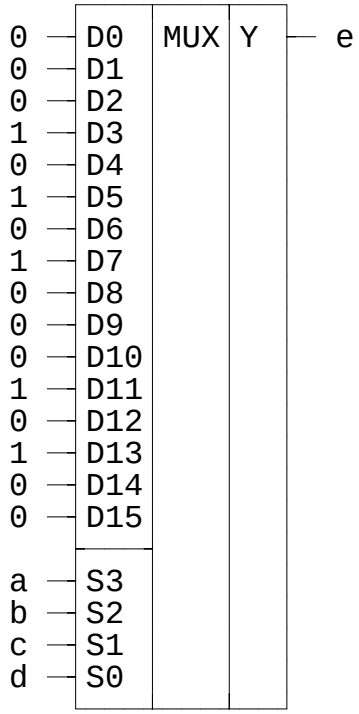
Aus der Gegenüberstellung der beiden BOOLEschen Funktionen ist ableitbar, daß mit der Zuordnung

$s_3 = a,\ s_2 = b,\ s_1 = c,\ s_0 = d$

$d_3 = d_5 = d_7 = d_{11} = d_{13} = 1$

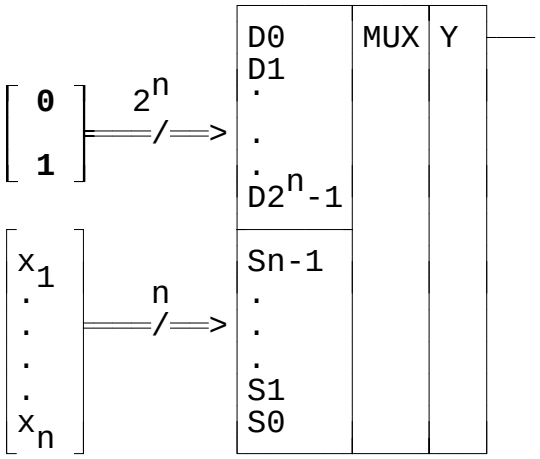
$d_0 = d_1 = d_2 = d_4 = d_6 = d_8 = d_9 = d_{10} = d_{12} = d_{14} = d_{15} = 0$

der Multiplexer so "programmiert" ist, daß er die gewünschte Funktion realisiert. Es ergibt sich die Schaltung:



Die Zuordnung der Variablen zu den Adreßeingängen des Multiplexers ist beliebig. Die Belegung der Dateneingänge ist abhängig von der gewählten Zuordnung der Variablen zu den Adreßeingängen und von der zu realisierenden Funktion.

Als allgemeines Beschaltungsschema ergibt sich



5.1.6.5. Realisierung einer n-stelligen BOOLEschen Funktion mit einem 2^{n-1}-zu-1-Multiplexer

Die ausführliche Schreibweise der KDNF einer n-stelligen BOOLEschen Funktion hat auch Ähnlichkeit mit der Funktionsbeschreibung eines 2^{n-1}-zu-1-Multiplexers.

BOOLEsche Funktion		**Multiplexer**
$y = \bar{x}_1\bar{x}_2\ldots\overline{x_{n-1}}\wedge\bar{x}_n f(00..0\mathbf{0}) \vee$	⟶	$y = \overline{s_{n-2}}\,\overline{s_{n-3}}\ldots\bar{s}_0 d_0 \vee$
$\bar{x}_1\bar{x}_2\ldots\overline{x_{n-1}}\wedge x_n f(00..0\mathbf{1}) \vee$		
$\bar{x}_1\bar{x}_2\ldots x_{n-1}\wedge\bar{x}_n f(00..1\mathbf{0}) \vee$	⟶	$\overline{s_{n-2}}\,\overline{s_{n-3}}\ldots\bar{s}_0 d_1 \vee$
$\bar{x}_1\bar{x}_2\ldots x_{n-1}\wedge x_n f(00..1\mathbf{1}) \vee$		
⋮		⋮
$x_1x_2\ldots x_{n-1}\wedge\bar{x}_n f(11..1\mathbf{0}) \vee$	⟶	$s_{n-2}s_{n-3}\ldots s_0 d_{2^{n-1}-1}$
$x_1x_2\ldots x_{n-1}\wedge x_n f(11..1\mathbf{1})$		

Mit der Zuordnung

$s_{n-2} = x_1,\ s_{n-3} = x_2,\ \ldots\ ,\ s_0 = x_{n-1}$

$d_0 = \bar{x}_n f(00..0\mathbf{0}) \vee x_n f(00..0\mathbf{1})$

$d_1 = \bar{x}_n f(00..1\mathbf{0}) \vee x_n f(00..1\mathbf{1})$

$d_{2^{n-1}-1} = \bar{x}_n f(11..1\mathbf{0}) \vee x_n f(11..1\mathbf{1})$

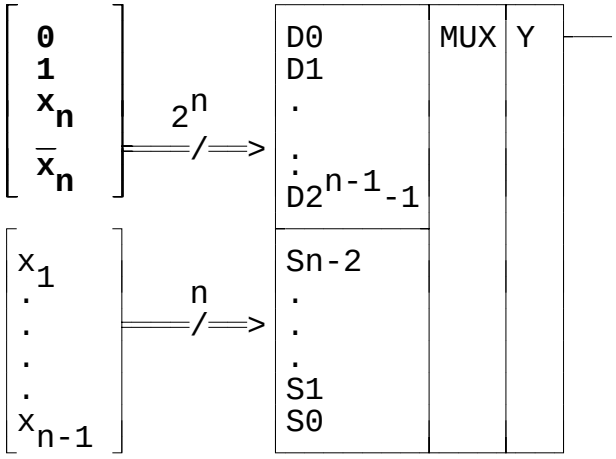
gelingt es offenbar auch, dem Multiplexer die gewünschte BOOLEsche Funktion "beizubringen". Wenn man die Ausdrücke

$d_i = \overline{x}_n f(.....\mathbf{0}) \vee x_n f(.....\mathbf{1})$

genauer untersucht, ergibt sich für die 4 möglichen Belegungen der Funktionswerte f(.....**0**) und f(.....**1**)

f(.....**0**)	f(.....**1**)	$d_i = \overline{x}_n f(.....\mathbf{0}) \vee x_n f(.....\mathbf{1})$
0	0	$\overline{x}_n \wedge 0 \vee x_n \wedge 0 = 0 \vee 0 = \mathbf{0}$
0	1	$\overline{x}_n \wedge 0 \vee x_n \wedge 1 = 0 \vee x_n = \mathbf{x_n}$
1	0	$\overline{x}_n \wedge 1 \vee x_n \wedge 0 = \overline{x}_n \vee 0 = \mathbf{\overline{x}_n}$
1	1	$\overline{x}_n \wedge 1 \vee x_n \wedge 1 = \overline{x}_n \vee x_n = \mathbf{1}$

Daraus folgt das allgemeine Beschaltungsschema



Aus den n Variablen der zu realisierenden Funktion sind n-1 beliebige Variablen auszuwählen und in beliebiger Weise den n-1 Adreßeingängen des Multiplexers zuzuordnen. Davon und von der zu realiserenden Funktion ist die Belegung der Dateneingänge abhängig, wobei für jeden Dateneingang aus der Beschaltung mit 0, 1, x_n oder $/x_n$ zu wählen ist.

Für die 4-stellige BOOLEsche Funktion aus Beispiel 5.13. benötigen wir einen 2^3-zu-1-Multiplexer, d. h. einen 8-zu-1-Multiplexer.

BOOLEsche Funktion **Multiplexer**

$e = \bar{a}\bar{b}\bar{c}\mathbf{d} \wedge \mathbf{0} \vee \bar{a}\bar{b}\bar{c}\mathbf{d} \wedge \mathbf{0} \vee \;\longrightarrow\; y = \overline{s}_2\overline{s}_1\overline{s}_0 \wedge \mathbf{d_0} \vee$

$\bar{a}\bar{b}c\bar{\mathbf{d}}\wedge\mathbf{0} \vee$
$\bar{a}\bar{b}c\mathbf{d}\wedge\mathbf{1} \vee$ ⟶ $\bar{s}_2\bar{s}_1s_0\wedge\mathbf{d_1} \vee$

$\bar{a}b\bar{c}\bar{\mathbf{d}}\wedge\mathbf{0} \vee$
$\bar{a}b\bar{c}\mathbf{d}\wedge\mathbf{1} \vee$ ⟶ $\bar{s}_2s_1\bar{s}_0\wedge\mathbf{d_2} \vee$

$\bar{a}bc\bar{\mathbf{d}}\wedge\mathbf{0} \vee$
$\bar{a}bc\mathbf{d}\wedge\mathbf{1} \vee$ ⟶ $\bar{s}_2s_1s_0\wedge\mathbf{d_3} \vee$

$a\bar{b}\bar{c}\bar{\mathbf{d}}\wedge\mathbf{0} \vee$
$a\bar{b}\bar{c}\mathbf{d}\wedge\mathbf{0} \vee$ ⟶ $s_2\bar{s}_1\bar{s}_0\wedge\mathbf{d_4} \vee$

$a\bar{b}c\bar{\mathbf{d}}\wedge\mathbf{0} \vee$
$a\bar{b}c\mathbf{d}\wedge\mathbf{1} \vee$ ⟶ $s_2\bar{s}_1s_0\wedge\mathbf{d_5} \vee$

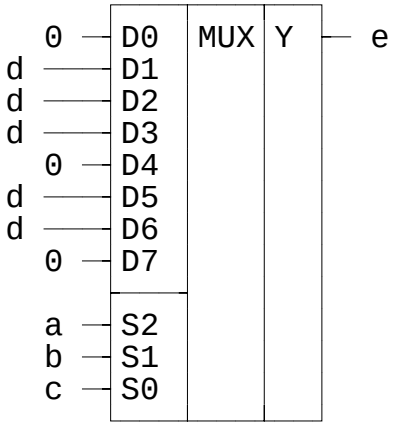
$ab\bar{c}\bar{\mathbf{d}}\wedge\mathbf{0} \vee$
$ab\bar{c}\mathbf{d}\wedge\mathbf{1} \vee$ ⟶ $s_2s_1\bar{s}_0\wedge\mathbf{d_6} \vee$

$abc\bar{\mathbf{d}}\wedge\mathbf{0} \vee$
$abc\mathbf{d}\wedge\mathbf{0}$ ⟶ $s_2s_1s_0\wedge\mathbf{d_7}$

Aus der Gegenüberstellung der beiden BOOLEschen Funktionen ist ableitbar, daß mit der Zuordnung

$s_2 = a,\ s_1 = b,\ s_0 = c$
$d_0 = d_4 = d_7 = 0$
$d_1 = d_2 = d_3 = d_5 = d_6 = d$

der Multiplexer so "programmiert" ist, daß er die gewünschte Funktion realisiert. Es ergibt sich die Schaltung:



Der Fall 2^{n-1}-zu-1-Multiplexer ist besonders interessant, da hier (wenn überhaupt) nur ein einziger Negator zusätzlich nötig ist.

5.1.6.6. Realisierung einer n-stelligen BOOLEschen Funktion mit einem 2^{n-2}-zu-1-Multiplexer

Wenn man die in 5.1.6.4. und 5.1.6.5. gefundenen allgemeinen Beschaltungsschemata vergleicht, erkennt man eine Systematik, die dabei hilfreich ist, das allgemeine Beschaltungsschema für die Realisierung einer beliebige n-stellige BOOLEsche Funktion mit dem nächstkleineren Multiplexer, d. h. einem 2^{n-2}-zu-1-Multiplexer, zu ermitteln. Im Fall 2^n-zu-1-Multiplexer ist die Beschaltung jedes der Dateneingänge aus der Funktionenmenge {Konstante_0,Konstante_1}, d. h. aus der **Menge der Funktionen keiner Variablen**, zu wählen; im Fall 2^{n-1}-zu-1-Multiplexer aus der Funktionenmenge {Konstante_0,Konstante_1,Identität,Negation}, d. h. aus der **Menge der Funktionen einer Variablen**. Wir dürfen also erwarten, daß im Fall 2^{n-2}-zu-1-Multiplexer die Beschaltung jedes der Dateneingänge aus der **Menge der Funktionen zweier Variablen** zu wählen sein wird.

Für die 4-stellige BOOLEsche Funktion aus Beispiel 5.13. benötigen wir einen 2^2-zu-1-Multiplexer, d. h. einen 4-zu-1-Multiplexer.

BOOLEsche Funktion		**Multiplexer**
$e = \bar{a}\bar{b}\bar{c}\mathbf{d}\wedge\mathbf{0} \vee$	⎫ ⟶	$y = \overline{s}_1\overline{s}_0\wedge\mathbf{d_0} \vee$
$\bar{a}\bar{b}\bar{c}\mathbf{d}\wedge\mathbf{0} \vee$	⎪	
$\bar{a}\bar{b}\mathbf{c}\bar{\mathbf{d}}\wedge\mathbf{0} \vee$	⎪	
$\bar{a}\bar{b}\mathbf{cd}\wedge\mathbf{1} \vee$	⎭	
$\bar{a}b\bar{c}\mathbf{d}\wedge\mathbf{0} \vee$	⎫ ⟶	$\overline{s}_1 s_0\wedge\mathbf{d_1} \vee$
$\bar{a}b\bar{c}\mathbf{d}\wedge\mathbf{1} \vee$	⎪	
$\bar{a}b\mathbf{c}\bar{\mathbf{d}}\wedge\mathbf{0} \vee$	⎪	
$\bar{a}b\mathbf{cd}\wedge\mathbf{1} \vee$	⎭	
$a\bar{b}\bar{c}\mathbf{d}\wedge\mathbf{0} \vee$	⎫ ⟶	$s_1\overline{s}_0\wedge\mathbf{d_2} \vee$
$a\bar{b}\bar{c}\mathbf{d}\wedge\mathbf{0} \vee$	⎪	
$a\bar{b}\mathbf{c}\bar{\mathbf{d}}\wedge\mathbf{0} \vee$	⎪	
$a\bar{b}\mathbf{cd}\wedge\mathbf{1} \vee$	⎭	
$ab\bar{c}\mathbf{d}\wedge\mathbf{0} \vee$	⎫ ⟶	$s_1 s_0\wedge\mathbf{d_3}$
$ab\bar{c}\mathbf{d}\wedge\mathbf{1} \vee$	⎪	
$ab\mathbf{c}\bar{\mathbf{d}}\wedge\mathbf{0} \vee$	⎪	
$ab\mathbf{cd}\wedge\mathbf{0}$	⎭	

Aus der Gegenüberstellung der beiden BOOLEschen Funktionen ist ableitbar, daß mit der Zuordnung

$s_1 = a,\ s_0 = b,$

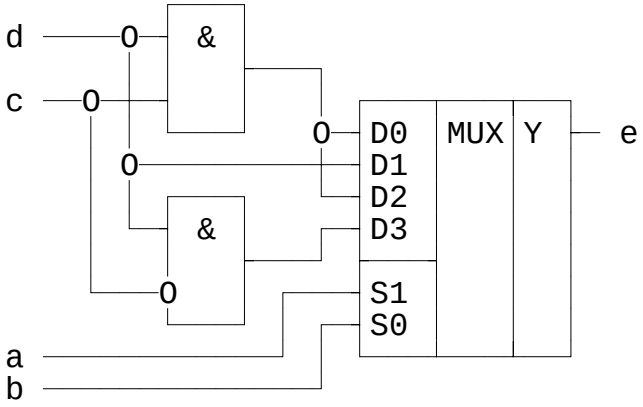
$d_0 = \bar{c}\bar{d} \wedge 0 \vee \bar{c}d \wedge 0 \vee c\bar{d} \wedge 0 \vee cd \wedge 1 = cd$

$d_1 = \bar{c}\bar{d} \wedge 0 \vee \bar{c}d \wedge 1 \vee c\bar{d} \wedge 0 \vee cd \wedge 1 = \bar{c}d \vee cd = d$

$d_2 = \bar{c}\bar{d} \wedge 0 \vee \bar{c}d \wedge 0 \vee c\bar{d} \wedge 0 \vee cd \wedge 1 = cd$

$d_3 = \bar{c}\bar{d} \wedge 0 \vee \bar{c}d \wedge 1 \vee c\bar{d} \wedge 0 \vee cd \wedge 0 = \bar{c}d$

der Multiplexer so "programmiert" ist, daß er die gewünschte Funktion realisiert. Es ergibt sich die Schaltung:



5.2. Sequentielle Schaltungen

5.2.1. Grundlagen

5.2.1.1. Begriff

Sie erinnern sich:

Kombinatorisch heißt ein System dann, wenn seine Ausgangsbelegung in einem Zeitpunkt t_n ausschließlich von der **Eingangsbelegung** im gleichen Zeitpunkt t_n bestimmt wird.

Im Gegensatz dazu:

Sequentiell heißt ein System dann, wenn seine Ausgangsbelegung in einem Zeitpunkt t_n von **der Folge der Eingangsbelegungen** in (nicht notwendig allen) zurückliegenden Zeitpunkten $...,t_{n-2},t_{n-1}$ und dem Zeitpunkt t_n bestimmt wird.

Ein sequentielles System braucht ein **"Gedächtnis"**, das sich die Eingangsbelegungen in den zurückliegenden Zeitpunkten merkt.

5.2.1.2. Der endliche deterministische Automat

Nichts ist so praktisch wie eine gute Theorie!

Das Automatenmodell wird in der Informatik breit angewendet. Sie werden es in den verschiedensten Zusammenhängen wiederfinden, auch in abgewandelter Form. Die Form, die ich Ihnen hier vorstelle, ist die Grundlage für die einheitliche Beschreibung aller sequentiellen Systeme und folglich von immenser Bedeutung.

Def.: Ein Automat A ist ein Quintupel

$$A = (X,Y,Z,f,g)$$

mit

X = Menge der Eingabesymbole (= Eingabealphabet)
Y = Menge der Ausgabesymbole (= Ausgabealphabet)
Z = Menge der Zustände (= Zustandsalphabet)
f = Überführungsfunktion
g = Ergebnisfunktion (Ausgabefunktion, Resultatfunktion)

Da die **Menge der Zustände** (= Gedächtnis des Automaten) **endlich** ist, heißt der Automat **endlicher** Automat.

mit

$f: X \times Z \longrightarrow Z$
$g: X \times Z \longrightarrow Y$

f und g sind Mengenrelationen oder Abbildungen. Jeder Kombination aus einem Symbol des Eingabealphabets und einem Zustand ("alter" Zustand) aus dem Zustandsalphabet wird durch f genau ein Zustand ("neuer" Zustand) aus dem Zustandsalphabet und durch g genau ein Symbol aus dem Ausgabealphabet zugeordnet. Man kann f und g als Mengen geordneter Tripel (Eingabesymbol,alter_Zustand,neuer_Zustand) bzw. (Eingabesymbol,alter_Zustand,Ausgabesymbol) beschreiben.

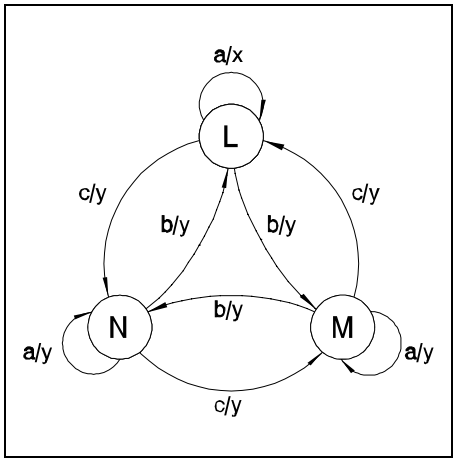
Da immer **genau ein Folgezustand** und genau ein Ausgabesymbol zugeordnet werden, heißt der Automat **deterministischer** Automat.

Beispiel 5.14.

```
X = {a,b,c}
Y = {x,y}
Z = {L,M,N}

f = {(a,L,L),(b,L,M),(c,L,N),
     (a,M,M),(b,M,N),(c,M,L),
     (a,N,N),(b,N,L),(c,N,M)}

g = {(a,L,x),(b,L,y),(c,L,y),
     (a,M,y),(b,M,y),(c,M,y),
     (a,N,y),(b,N,y),(c,N,y)}
```

Diese Darstellung ist zwar korrekt aber wenig übersichtlich. Man stellt die Verhältnisse entweder in einem **Automatengraphen** oder in einer **Automatentabelle** dar:

Automatengraph



Automatentabelle

Ergebnisfunktion g

Überführungsfunktion f

x	z	z'	y
a	L	L	x
b	L	M	y
c	L	N	y
a	M	M	y
b	M	N	y
c	M	L	y
a	N	N	y
b	N	L	y
c	N	M	y

Der Automatengraph ist gut geeignet, einen Überblick zu geben. Für Berechnungen aller Art eignet sich besser die Automatenta-

belle. Wir werden beide Darstellungsformen nebeneinander verwenden.

Für den Fall des binären Automaten sind die Eingabesymbole, die Ausgabesymbole und die Zustände **binär codiert.**

Ein **Eingabesymbol** ist eine **Belegung der Eingänge**, ein **Ausgabesymbol eine Belegung der Ausgänge** und ein **Zustand eine Belegung des Zustandsspeichers**.

Man sagt dann besser:

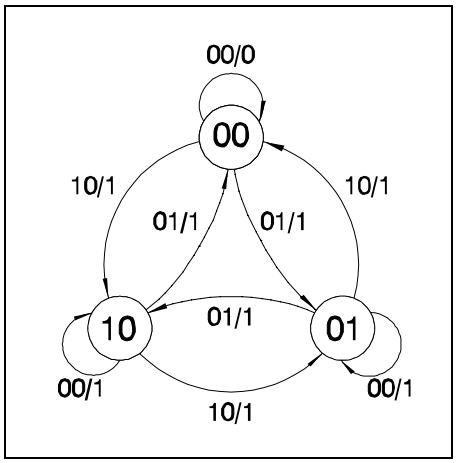
X = Menge der Eingangsbelegungen
Y = Menge der Ausgangsbelegungen
Z = Menge der Zustände
f = Überführungsfunktion
g = Ergebnisfunktion (Ausgabefunktion, Resultatfunktion)

Mit

a = 00, b = 01, c = 10, L = 00, M = 01, N = 10, x = 0, y = 1

geht unser Beispiel über in

Automatengraph



Automatentabelle

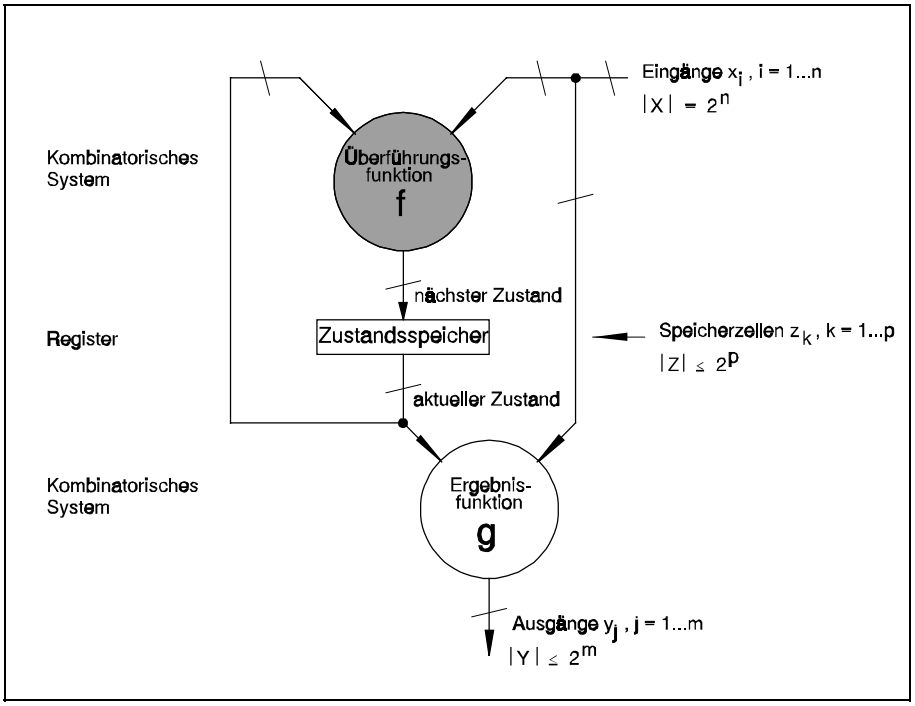
Ergebnisfunktion g

Überführungsfunktion f

x z	z'	y
00 00	00	0
01 00	01	1
10 00	10	1
00 01	01	1
01 01	10	1
10 01	00	1
00 10	10	1
01 10	00	1
10 10	01	1

Sie ahnen wohl sicher schon, daß uns die Tatsache, daß wir die Kodierungen x = 11 und z = 11 nicht verwenden, noch beschäftigen wird.

Eine weitere Darstellungsmöglichkeit, die uns den Weg für die schaltungstechnische Realisierung öffnet, ist eine Art Ersatzschaltung des Automaten:



Das Automatenmodell, welches wir bisher verwendet haben, ist ein sogenannter MEALY-Automat. Außerdem kennen wir noch dem MOORE- und den MEDVEDEV-Automaten. Sie unterscheiden sich nur in der Ausgabefunktion:

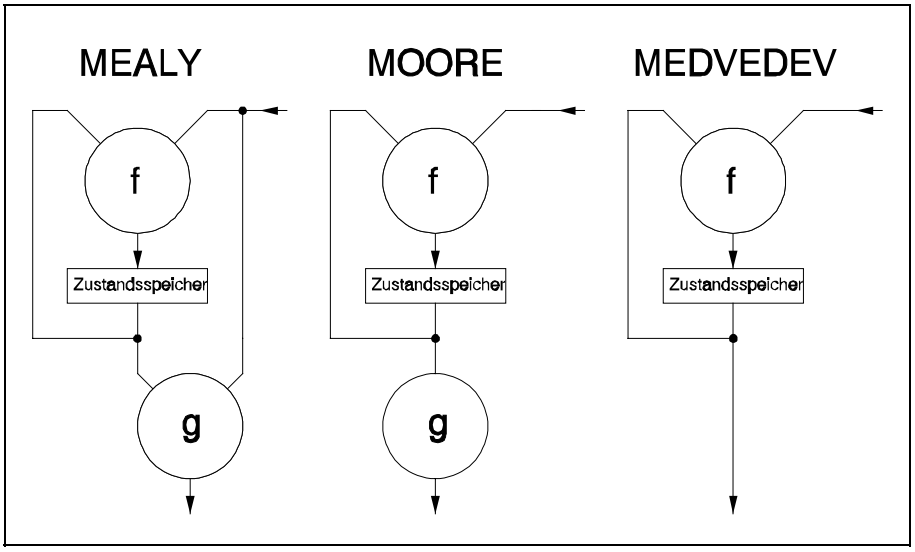
MEALY g: X x Z ⟶ Y

MOORE g: Z ⟶ Y

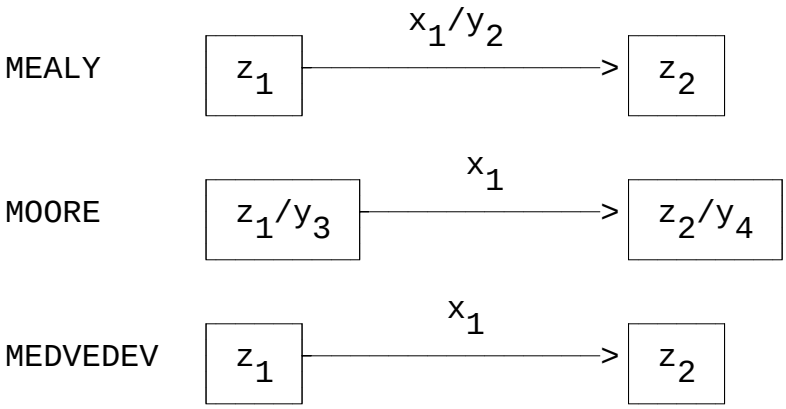
MEDVEDEV g: Z = Y

mit $z_i = y_i$ für alle i, d. h. die Speicherzellen sind direkt herausgeführt!

In der Ersatzschaltung zeigt sich das so:



Die Zustands- und Kantenbeschriftung im Automatengraphen ist für die drei Automatentypen unterschiedlich:



In der Automatentabelle fehlt beim MEDVEDEV-Automaten die Spalte für die Ausgangsbelegungen (y).

Beispiel 5.15.

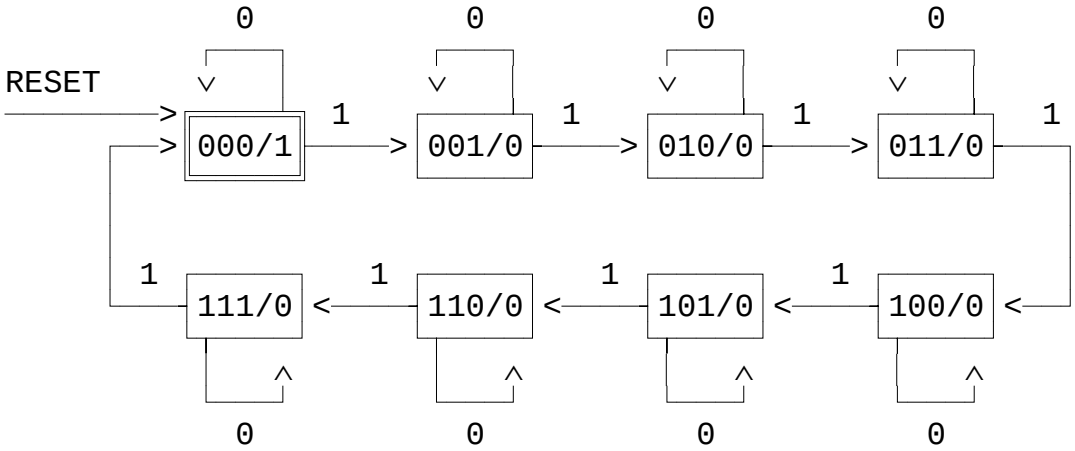
Ein 3-Bit-Binärzähler soll bei x = 1 in Schritten von 1 vorwärtszählen und bei x = 0 nicht zählen. Bei z = 000 soll er y = 1 ausgeben, sonst y = 0. Der Zustand z = 000 sei der Initialzustand (Anfangszustand), in den der Zähler auf geeignete, hier nicht näher betrachtete Weise voreinstellbar (rücksetzbar) ist.

Aus der Aufgabenstellung leiten wir ab

Z = {000,001,010,011,100,101,110,111}
X = {0,1}
Y = {0,1}

Da die Ausgabe unabhängig von der Eingabe ist, liegt ein MOORE-Automat vor.

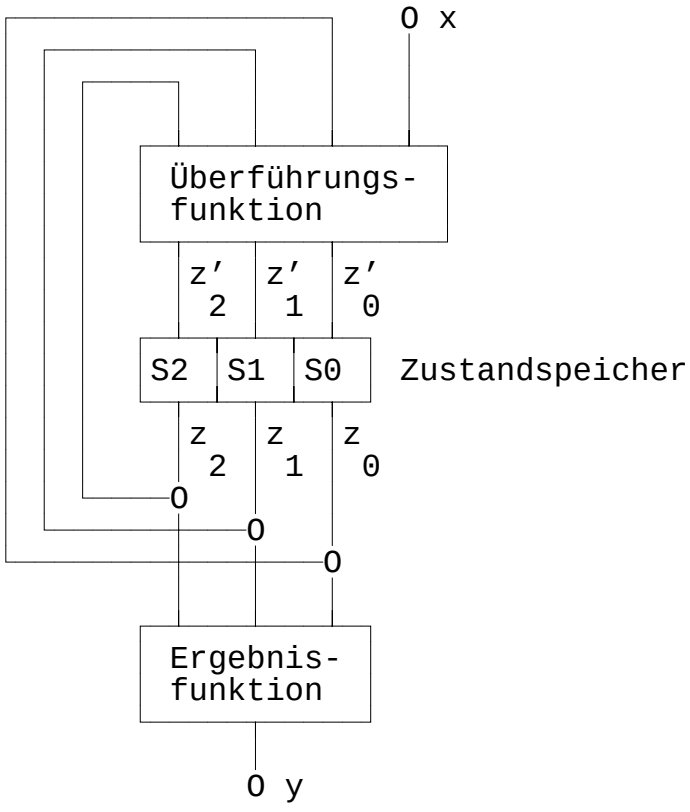
Automatengraph



Automatentabelle

x	$z_2 z_1 z_0$	$z'_2 z'_1 z'_0$	y
0	0 0 0	0 0 0	1
0	0 0 1	0 0 1	0
0	0 1 0	0 1 0	0
0	0 1 1	0 1 1	0
0	1 0 0	1 0 0	0
0	1 0 1	1 0 1	0
0	1 1 0	1 1 0	0
0	1 1 1	1 1 1	0
1	0 0 0	0 0 1	1
1	0 0 1	0 1 0	0
1	0 1 0	0 1 1	0
1	0 1 1	1 0 0	0
1	1 0 0	1 0 1	0
1	1 0 1	1 1 0	0
1	1 1 0	1 1 1	0
1	1 1 1	0 0 0	0

Ersatzschaltung



Vorgriff auf den Automatenentwurf:

Es war bereits betont worden, daß Überführungsfunktion und Ergebnisfunktion durch kombinatorische Systeme zu realisieren sind. Es sollte also möglich sein, die BOOLEschen Funktionen $z_2' = f(x,z_2,z_1,z_0)$, $z_1' = f(x,z_2,z_1,z_0)$, $z_0' = f(x,z_2,z_1,z_0)$ und $y = f(x,z_2,z_1,z_0)$ aus der Automatentabelle abzulesen und zu minimieren und damit entsprechende Gatternetze zu erhalten.

z_2'	0011	z_1
	0110	z_0
00	0000	
01	1111	
11	1101	
10	0010	
xz_2		

$\mathbf{z_2'} = \bar{x}z_2 \vee z_2\bar{z}_1 \vee z_2\bar{z}_0 \vee x\bar{z}_2z_1z_0 = \mathbf{z_2 \not\vee xz_1z_0}$

z_1'	0011	z_1
	0110	z_0
00	0011	
01	0011	
11	0101	
10	0101	
xz_2		

$\mathbf{z_1'} = z_1\bar{z}_0 \vee \bar{x}z_1 \vee x\bar{z}_1z_0 = \mathbf{z_1 \not\vee xz_0}$

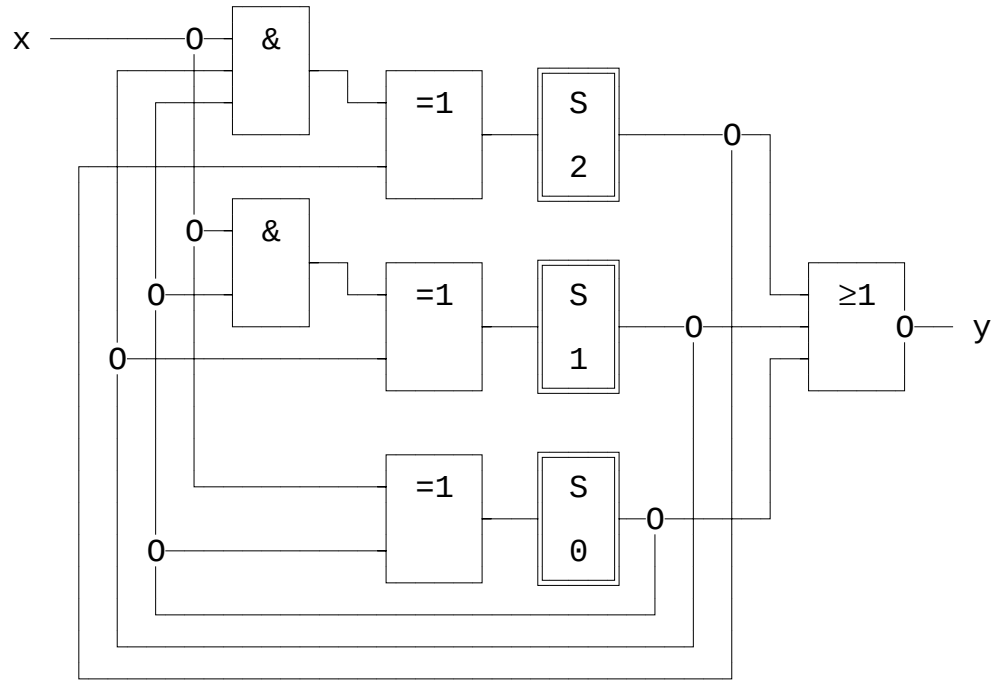
z_0'	0011	z_1
	0110	z_0
00	0110	
01	0110	
11	1001	
10	1001	
xz_2		

$\mathbf{z_0'} = \bar{x}z_0 \vee x\bar{z}_0 = \mathbf{z_0 \not\vee x}$

y	0011	z_1
	0110	z_0
00	1000	
01	0000	
11	0000	
10	1000	
xz_2		

$\mathbf{y} = \bar{z}_2\bar{z}_1\bar{z}_0 = \overline{\mathbf{z_2 \vee z_1 \vee z_0}}$

Antivalenz und NOR bieten sich hier an und führen zu einer sehr übersichtlichen Realisierung. Die resultierende Schaltung sähe dann so aus:



Das könnte tatsächlich bereits ein funktionstüchtiger Automat sein!

Wir haben eine Vorstellung vom Entwurf der Gatternetze für die beiden Funktionen. Unklar ist noch die Realisierung des Zustandsspeichers. Auch auf die oben vereinbarte Rücksetzbarkeit in einen Initialzustand sind wir noch nicht eingegangen. Und schließlich ist noch unklar, was die Zustandsübergänge wann auslöst. In den nächsten Vorlesungen werden wir uns deshalb auf den Zustandsspeicher konzentrieren.

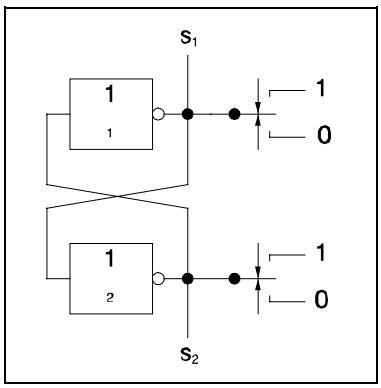
5.2.2. Zustandsspeicher

Ein Zustandsspeicher ist ein eindimensionales Array aus 1-Bit-Speicherzellen, den sog. Flipflops. Ein solches eindimensionales Array heißt auch (Speicher-)Register.

5.2.2.1. Flipflop

5.2.2.1.1. Experiment

Zum besseren Verständnis der Funktion eines Flipflops stellen wir ein Gedankenexperiment voran.



Wir betrachten zwei kreuzgekoppelte Negatoren, einen sog. Flipflop-Kern. An den Ausgängen s_1 und s_2 erkennen Sie zwei Taster. Jeder Taster hat drei Stellungen: 0, 1 und die hier gezeichnete Mittelstellung. Wenn sich ein Taster in der Stellung 0 (1) befindet, dann soll der Ausgang, an den der Taster angeschlossen ist, den Pegel 0 (1) annehmen. Befindet sich der Taster in Mittelstellung, bleibt der Ausgang sich selbst überlassen.

Die beiden Taster sollen unabhängig voneinander jede der drei Stellungen einnehmen können. Damit ist es möglich, den Flipflop-Kern beliebig zu initialisieren (s_1s_2 = 00|01|10|11), ihn anschließend sich selbst zu überlassen und zu beobachten, was er dann tut. Wir verwenden dazu ein Simulationsprogramm, das Sie im nächsten Semester noch genauer kennenlernen werden (VHDL).

In einem ersten Experiment nehmen wir an, daß der Flipflop-Kern symmetrisch ist, d. h. beide Negatoren sollen exakt die gleiche Verzögerungszeit aufweisen (hier: 10 ns).

t	s1 s2	s1 s2	s1 s2	s1 s2	
0 ns	**0 0**	**0 1**	**1 0**	**1 1**	**<— Initialbelegung**
10 ns	1 1	. .	. .	0 0	
20 ns	0 0	. .	. .	1 1	
30 ns	1 1	. .	. .	0 0	
40 ns	0 0	. .	. .	1 1	

In den Initialisierungsfällen s_1s_2 = 00 und s_1s_2 = 11 "schwingt" der Flipflop-Kern, in den Initialisierungsfällen s_1s_2 = 01 und s_1s_2 = 10 behält der Flipflop-Kern die Initialbelegungen bei, er "speichert" sie.

In einem zweiten Experiment nehmen wir an, daß der Flipflop-Kern unsymmetrisch ist, d. h. die beiden Negatoren sollen unterschiedliche Verzögerungszeiten aufweisen (Negator 1: 10 ns, Negator 2: 20 ns).

t	s1 s2	s1 s2	s1 s2	s1 s2	
0 ns	**0 0**	**0 1**	**1 0**	**1 1**	**<— Initialbelegung**
10 ns	1 0	. .	. .	0 1	
20 ns	. .	. .	. .	. .	
30 ns	. .	. .	. .	. .	
40 ns	. .	. .	. .	. .	

In den Initialisierungsfällen s_1s_2 = 00 und s_1s_2 = 11 "gewinnt" der schnellere Negator, d. h. der Negator, der als erster den neuen Ausgangspegel bereitstellt, in den Initialisierungsfällen s_1s_2 = 01 und s_1s_2 = 10 behält der Flipflop-Kern die Initialbelegungen bei, er "speichert" sie.

In einem dritten Experiment nehmen wir an, daß Negator 1 der langsamere (20 ns) und Negator 2 der schnellere (10 ns) ist.

t	s1 s2	s1 s2	s1 s2	s1 s2	
0 ns	**0 0**	**0 1**	**1 0**	**1 1**	**<— Initialbelegung**
10 ns	0 1	. .	. .	1 0	
20 ns	. .	. .	. .	. .	
30 ns	. .	. .	. .	. .	
40 ns	. .	. .	. .	. .	

Auch hier "gewinnt" wieder in den Initialisierungsfällen s_1s_2 =

00 und s_1s_2 = 11 der schnellere Negator, und auch hier "speichert" wieder der Flipflop-Kern in den Initialsierungsfällen s_1s_2 = 01 und s_1s_2 = 10 die Initialsierungsbelegungen.

Wenn man die Ergebnisse der drei Experimente anders zusammenstellt, wird die Aussage noch deutlicher:

t	t1<t2		t1=t2		t1>t2		
	s1	s2	s1	s2	s1	s2	
0 ns	**0**	**0**	**0**	**0**	**0**	**0**	**<— Initialbelegung**
10 ns	1	0	1	1	0	1	
20 ns	.	.	0	0	.	.	
30 ns	.	.	1	1	.	.	
40 ns	.	.	0	0	.	.	

t	t1<t2		t1=t2		t1>t2		
	s1	s2	s1	s2	s1	s2	
0 ns	**0**	**1**	**0**	**1**	**0**	**1**	**<— Initialbelegung**
10 ns	.	.	.	.	.	.	
20 ns	.	.	.	.	.	.	
30 ns	.	.	.	.	.	.	
40 ns	.	.	.	.	.	.	

t	t1<t2		t1=t2		t1>t2		
	s1	s2	s1	s2	s1	s2	
0 ns	**1**	**0**	**1**	**0**	**1**	**0**	**<— Initialbelegung**
10 ns	.	.	.	.	.	.	
20 ns	.	.	.	.	.	.	
30 ns	.	.	.	.	.	.	
40 ns	.	.	.	.	.	.	

t	t1<t2		t1=t2		t1>t2		
	s1	s2	s1	s2	s1	s2	
0 ns	**1**	**1**	**1**	**1**	**1**	**1**	**<— Initialbelegung**
10 ns	0	1	0	0	1	0	
20 ns	.	.	1	1	.	.	
30 ns	.	.	0	0	.	.	
40 ns	.	.	1	1	.	.	

In den Initialsierungsfällen s_1s_2 = 01 und s_1s_2 = 10 "speichert" der Flipflop-Kern die Initialbelegung unabhängig davon, ob er symmetrisch oder unsymmetrisch ist. In den Initialisierungsfällen s_1s_2 = 00 und s_1s_2 11 hängt das Ergebnis davon ab, ob der Flipflop-Kern symmetrisch ist (dann "schwingt" er) oder ob und auf welche Art er unsymmetrisch ist (dann "gewinnt" der schnellere Negator).

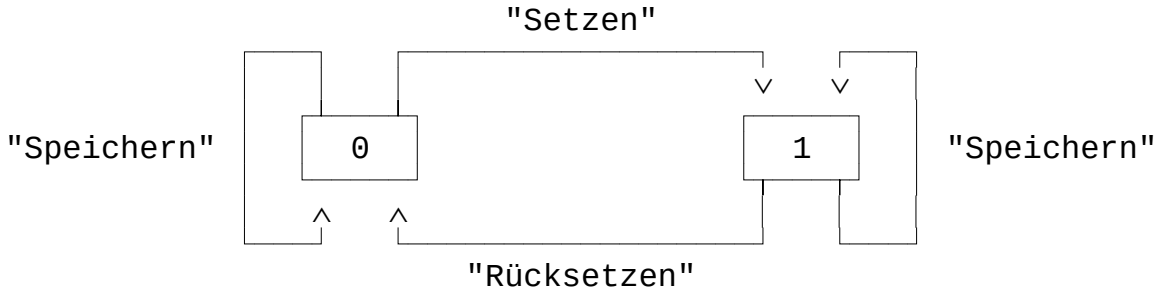
In der technischen Realität ist einerseits die exakte Symmetrie

nicht erreichbar und andererseits ist die Art der Unsymmetrie schwer vorhersagbar. Die Initialisierungsfälle $s_1s_2 = 00$ und $s_1s_2 = 11$ sind deshalb ungeeignet für den Einsatz des Flipflop-Kerns als Speicher. Die Initialisierungsfälle $s_1s_2 = 01$ und $s_1s_2 = 10$ hingegen eignen sich offenbar hervorragend.

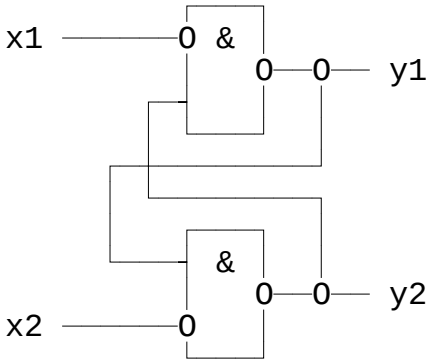
Unklar ist noch, wie die zu speichernde Information in den Flipflop-Kern hineinkommt, da ja kaum davon ausgegangen werden kann, daß die in unserem Gedankenexperiment verwendeten Taster zum Einsatz kommen. Die weiteren Überlegungen werden sich also mit der schaltungstechnischen Realisierung von Flipflops befassen müssen.

5.2.2.1.2. Grund-Flipflop

Das Grund-Flipflop (GFF) ist ein Miniautomat vom Typ MEDVEDEV. Der Automat hat zwei Zustände, Z = {0,1}.



Wir kennen zwei unterschiedliche GFFs, das NAND-Grund-Flipflop (NAND-GFF) und das NOR-Grund-Flipflop (NOR-GFF), und werden aber, obwohl dafür kein zwingender Grund besteht, nur das NAND-GFF näher betrachten und verwenden.



Was leistet das NAND-GFF? Wir analysieren das Flipflop zunächst als kombinatorisches System und ermitteln die Wahrheitstabelle.

x_1	x_2	y_1	y_2	
1	1	1	1	
1	0	1	0	
0	1	0	1	
0	0	?	?	<—— Mit $x_1x_2 = 00$ bleibt sich das Flipflop selbst überlassen!

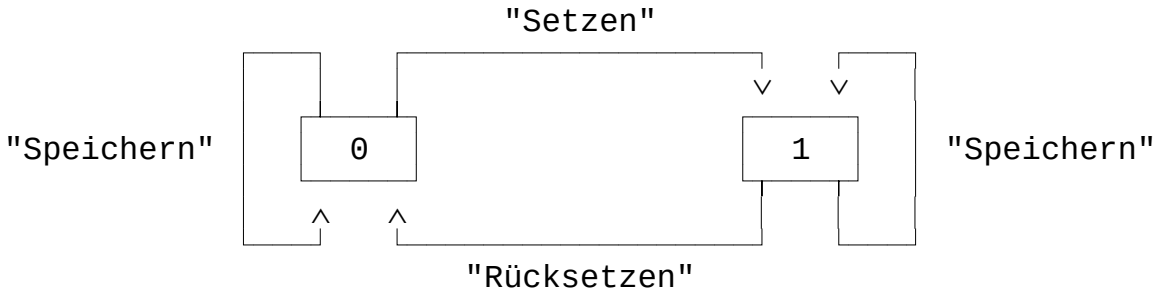
Was passiert genau bei $x_1x_2 = 00$? Wir vermuten, daß die vorherge-

hende Belegung von Bedeutung sein wird, und beziehen sie in die Analyse ein.

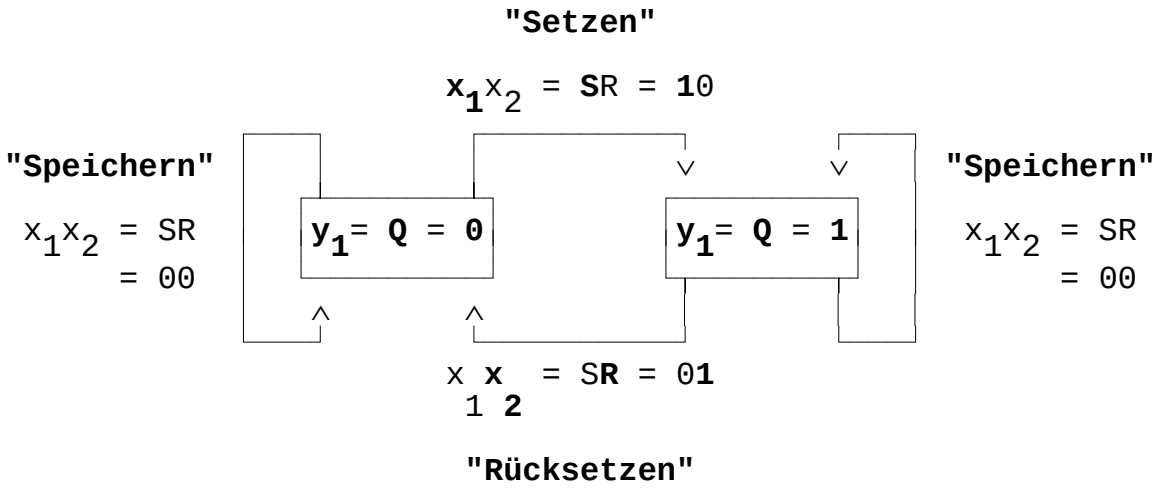
	x_1x_2	y_1y_2	x_1x_2	y_1y_2	x_1x_2	y_1y_2	x_1x_2	y_1y_2
t-1	0 0	? ?	0 1	**0 1**	1 0	**1 0**	1 1	1 1
t	0 0	? ?	0 0	**0 1**	0 0	**1 0**	0 0	x x

? "speichert" "schwingt"

Im Vergleich mit dem o. a. Automatengraphen



gelingt die Deutung

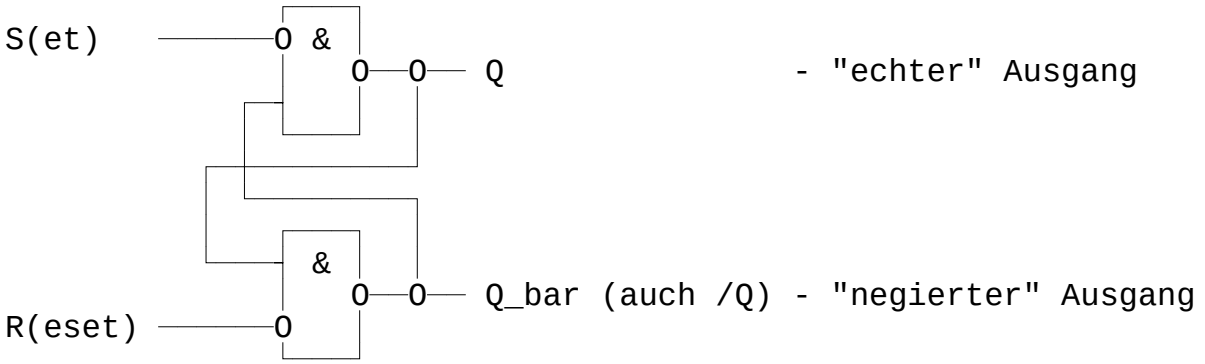


$\overline{S} \wedge \overline{R} = 1$ —-> Weder Setzen noch Rücksetzen ——> Speichern

$\overline{S} \wedge R = 1$ —-> Nicht Setzen, aber Rücksetzen ——> Rücksetzen

$S \wedge \overline{R} = 1$ —-> Setzen, aber nicht Rücksetzen ——> Setzen

$S \wedge R = 1$ —-> Sowohl Setzen als auch Rücksetzen ——> "verboten"



Wir wollen dieses Flipflop in den weiteren Betrachtungen **NAND - RS-Grund-Flipflop (NAND-RS-GFF)** nennen.

Die Belegung SR = 11 ist nicht eigentlich verboten, da sie ja zur definierten Ausgangsbelegung QQ_bar = 11 führt (Q_bar = /Q ist dabei zwar nicht erfüllt, aber das ist eher ein zu verschmerzender Mangel). Wenn jedoch auf SR = 11 unmittelbar SR = 00 ("Speichern") folgt, ist die Ausgangsbelegung des Flipflops nicht vorhersagbar (s. 5.2.2.1.1. Experiment).

Bevor wir die zugehörige Automatentabelle angeben, sehen wir uns noch eine erweiterte Automatentabelle an, in der neben Q und Q' auch die negierten Ausgänge Q_bar und Q_bar' aufgeführt sind.

	S R	Q Q_bar	Q' Q_bar'	Bemerkung
	(0 0	0 0	? ?)	QQ_bar = 00 nicht einstellbar
——>	0 0	0 1	0 1	Speichern
——>	0 0	1 0	1 0	Speichern
	0 0	1 1	? ?	Q'Q_bar' nicht vorhersagbar
	(0 1	0 0	0 1)	QQ_bar = 00 nicht einstellbar
——>	0 1	0 1	0 1	Rücksetzen
——>	0 1	1 0	0 1	Rücksetzen
	0 1	1 1	0 1	Rücksetzen
	(1 0	0 0	1 0)	QQ_bar = 00 nicht einstellbar
——>	1 0	0 1	1 0	Setzen
——>	1 0	1 0	1 0	Setzen
	1 0	1 1	1 0	Setzen
	(1 1	0 0	1 1)	QQ_bar = 00 nicht einstellbar
	1 1	0 1	1 1	"verboten"
	1 1	1 0	1 1	"verboten"
	1 1	1 1	1 1	"verboten"

In die eigentliche Automatentabelle werden für dieses Flipflop nur diejenigen Zeilen aufgenommen, für die **Q_bar = /Q** und **S∧R = 0** gilt (mit "——>" markiert). Die Spalten Q_bar und Q_bar' werden weggelassen.

	S R	Q	Q'	Bemerkung
——>	0 0	0	0	Speichern
——>	0 0	1	1	Speichern
——>	0 1	0	0	Rücksetzen
——>	0 1	1	0	Rücksetzen
——>	1 0	0	1	Setzen
——>	1 0	1	1	Setzen

!
S∧R = 0

Gelegentlich findet man allerdings auch Automatentabellen, bei denen die Belegung SR = 11 zusätzlich angegeben ist. Obwohl SR = 11 zu Q'Q_bar'= 11 führt, nimmt man den schlechtesten Fall an, d. h. auf SR = 11 folgt SR = 00 ("Speichern"), und schreibt

	S R	Q	Q'	Bemerkung
——>	0 0	0	0	Speichern
——>	0 0	1	1	Speichern
——>	0 1	0	0	Rücksetzen
——>	0 1	1	0	Rücksetzen
——>	1 0	0	1	Setzen
——>	1 0	1	1	Setzen
	1 1	**0**	**X**	**"verboten"**
	1 1	**1**	**X**	**"verboten"**

Aus dieser Automatentabelle kann die BOOLEsche Funktion Q' = f(S,R,Q) ausgelesen werden. Eintragen in den KARNAUGH-Plan und Minimieren liefert die sog. Charakteristische Gleichung dieses Flipflops. Beim Minimieren dürfen die mit "X" markierten Felder nicht mit in Primimplikanten erfaßt werden.

Q'	0	1	Q
00	0	1	
01	0	0	
11	X	X	
10	1	1	
SR			

Charakteristische Gleichung:

$$\mathbf{Q' = S \vee \bar{R}Q} \quad (\text{mit } S \wedge R \overset{!}{=} 0)$$

Die **Automatentabelle**

S R	Q	Q'	Bemerkung
0 0	0	0	Speichern
0 0	1	1	Speichern
0 1	0	0	Rücksetzen
0 1	1	0	Rücksetzen
1 0	0	1	Setzen
1 0	1	1	Setzen

$S \wedge R \overset{!}{=} 0$

kann vereinfacht werden zur **Flipflop-Tabelle**

S R	Q'	Bemerkung
0 0	Q	Speichern
0 1	0	Rücksetzen
1 0	1	Setzen

$S \wedge R \overset{!}{=} 0$

und aus der Automatentabelle kann die dritte mögliche Tabelle,

die **Substitutionstabelle** (zur Begriffsbildung s. u.), abgeleitet werden

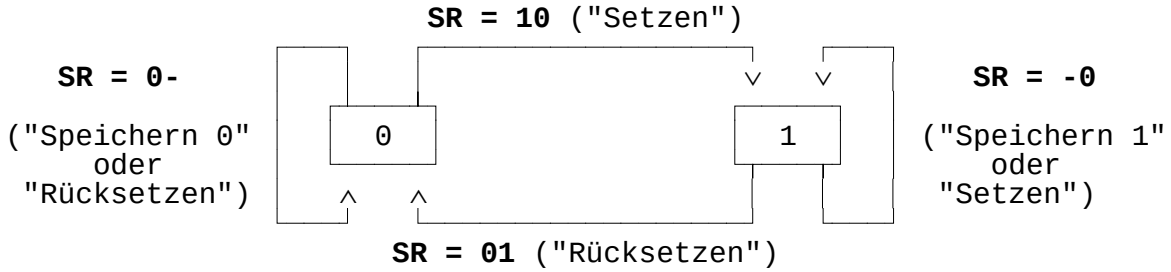
Q Q’	S R	Bemerkung
0 0	0 -	Speichern 0 oder Rücksetzen
1 1	- 0	Speichern 1 oder Setzen
1 0	0 1	Rücksetzen
0 1	1 0	Setzen

$S \wedge R \overset{!}{=} 0$

"-" steht für "Don’t Care" und bedeutet, daß jeder beliebige, gültige Pegel möglich ist (- = 0|1).

Alle drei angegebenen Tabellen enthalten die gleiche Information. Die Information wird nur unterschiedlich präsentiert. Jede der drei Tabellen hat ihre Existenzberechtigung und ist bei der Beantwortung gewisser Fragestellungen hilfreich.

Die **Flipflop-Tabelle** beschreibt in kompakter Form die Funktionsweise des Flipflops selbst. Die **Automatentabelle** ist dann hilfreich, wenn ein Flipflop als Automat aufgefaßt werden muß, z. B. bei der sog. Flipflop-Substitution, bei der ein Flipflop unter Verwendung eines Flipflops anderen Typs (zu Flipflop-Typen s. u.) realisiert werden soll. Die **Substitutionstabelle** beantwortet die Frage: Welche Eingangsbelegung SR muß man anlegen, wenn die Ausgangsbelegung des Flipflops von Q nach Q’ wechseln soll (mit QQ’ = 00|01|10|11)? Sie wird im Automatenentwurf (s. u.) verwendet. Aus ihr kann auch die Kantenbeschriftung des **Automatengraphen** abgelesen werden.

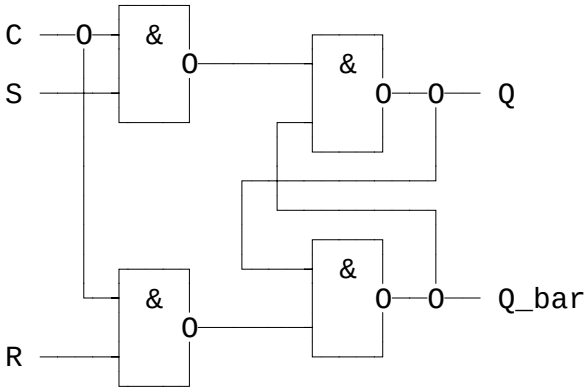


Beachten Sie die Unterschiede in der Kantenbeschriftung im Vergleich zu dem Automatengraphen, der am Anfang der Überlegungen stand! Die zuletzt angegebene Form werden wir in der Folge ausschließlich verwenden!

5.2.2.1.3. Taktzustandsgesteuerte (statische) Flipflops

Für den Einsatz der Flipflops in Zustandsspeichern von Automaten ist es wünschenswert, mit **einem einzigen**, von außen zugeführten Synchronisations- oder Taktsignal ("clock") zwischen Zeitbereichen, in denen Daten in das Flipflop eingetragen werden ("Setzen", "Rücksetzen"), und Zeitbereichen, in denen Daten im Flipflop gehalten werden ("Speichern"), unterscheiden zu können. Diese Forderung führt zunächst zum Konzept des taktzustandsgesteuerten (statischen) Flipflops.

5.2.2.1.3.1. Statisches RS-Flipflop

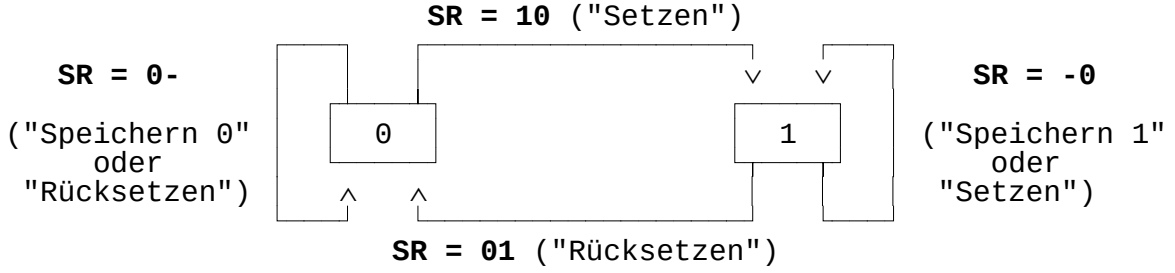


Die (ausführliche) Flipflop-Tabelle ergibt sich zu

C S R	Q'	Bemerkung
0 - -	Q	Speichern
1 0 0	Q	Speichern
1 0 1	0	Rücksetzen
1 1 0	1	Setzen

$S \wedge R \overset{!}{=} 0$

Üblicherweise läßt man aber die Zeile, in der das Verhalten bei inaktiver Taktbelegung (hier: C = 0) beschrieben wird, in der Automatentabelle weg und nimmt diese Zeile auch nicht in den Automatengraphen auf. Die Automatentabelle und der Automatengraph des taktzustandsgesteuerten RS-Flipflops sind folglich identisch mit der Automatentabelle und dem Automatengraphen des NAND-RS-GFF.

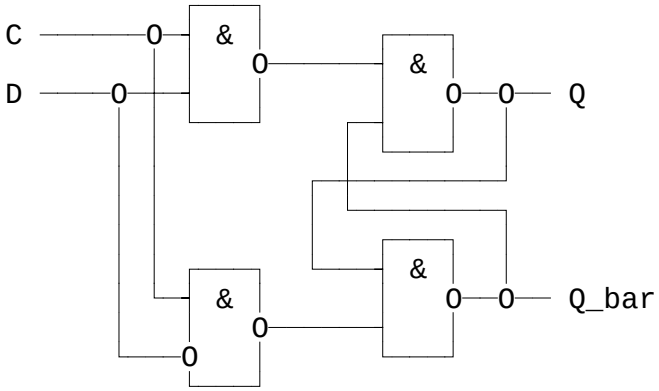


Man vereinbart, daß eine Kante dann und nur dann durchlaufen wird, wenn das Taktsignal aktiv ist (hier: C = 1). Bei inaktivem Taktsignal verharrt das Flipflop in seinem "alten" Zustand.

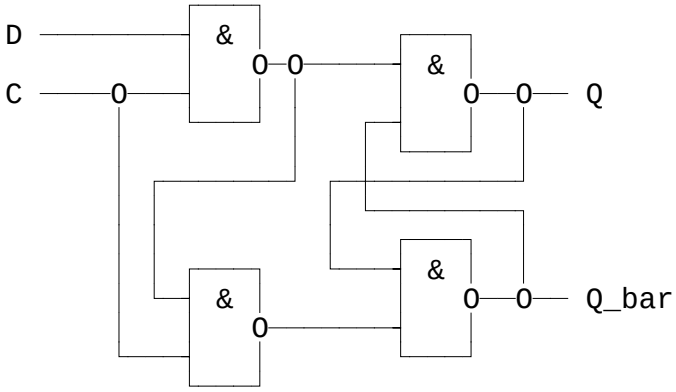
5.2.2.1.3.2. Statisches D-Flipflop

Da einerseits die "verbotene" Belegung (SR = 11) scheinbar nicht sinnvoll genutzt werden kann und andererseits Speichern (SR = 00) bei taktzustandsgesteuerten Flipflops auch mit der inaktiven Taktbelegung realisierbar ist, kann man auf die Belegungen SR =

11|00 auch gänzlich verzichten. Mit S = D und R = /S = /D folgt daraus das statische D-FF (**D** von **d**ata oder auch **d**elay)



Häufiger findet man allerdings eine kostengünstigere Realisierung (Überzeugen Sie sich von der logischen Identität beider Realisierungen!)

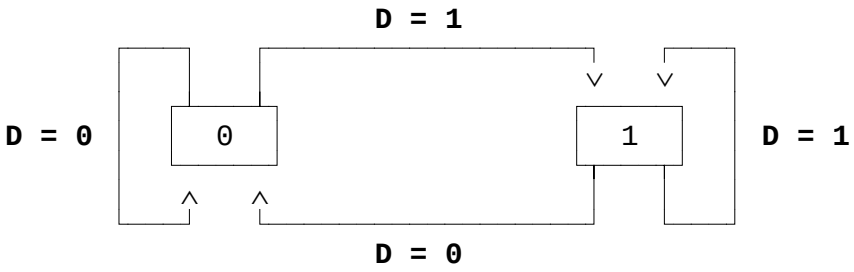


Die Tabellen und der Automatengraph fallen deutlich einfacher aus.

D	Q’
0	0
1	1

D	Q	Q’
0	0	0
0	1	0
1	0	1
1	1	1

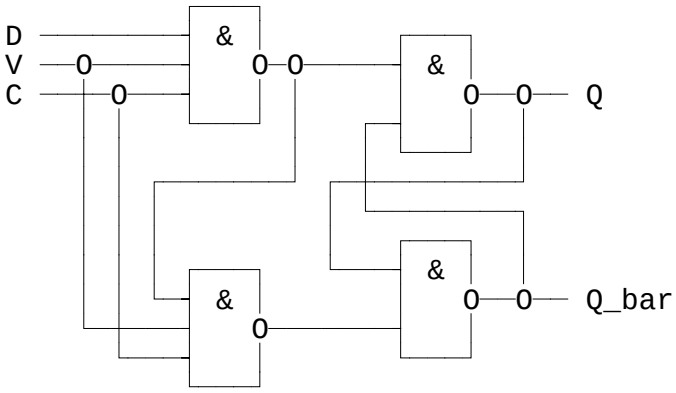
Q	Q’	D
0	0	0
0	1	1
1	0	0
1	1	1



Man könnte der Auffassung sein, daß im Zustandspeicher des Automatem aus Beispiel 5.15. genau solche statischen D-FFs zum Einsatz kommen könnten. Warum nicht, wird uns später klar werden.

5.2.2.1.3.3. Statisches DV-Flipflop

Wenn man Speichern nicht nur mit der inaktiven Taktbelegung realisieren können will, greift man zum DV-FF (**V** von **v**alid).

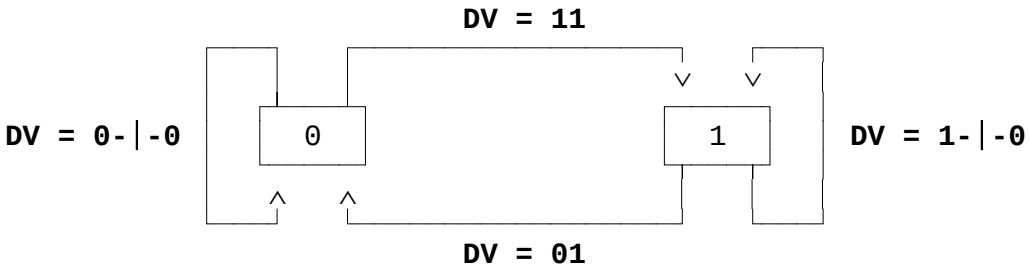


mit

D V	Q'
0 0	Q
0 1	0
1 0	Q
1 1	1

D V	Q	Q'
0 0	0	0
0 0	1	1
0 1	0	0
0 1	1	0
1 0	0	0
1 0	1	1
1 1	0	1
1 1	1	1

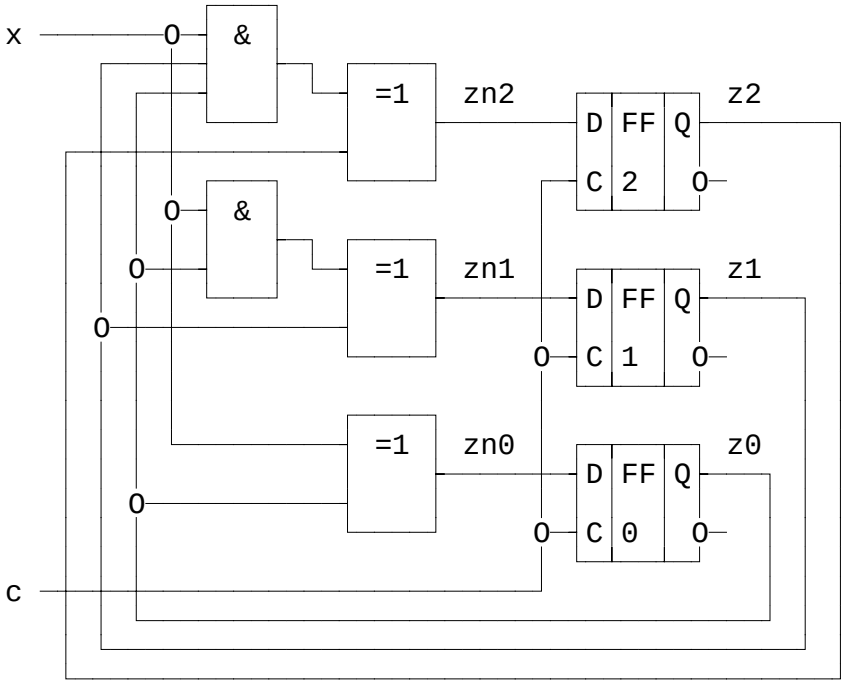
Q Q'	D V	
0 0	0 -	<——
0 0	- 0	<——
0 1	1 1	
1 0	0 1	
1 1	1 -	<——
1 1	- 0	<——



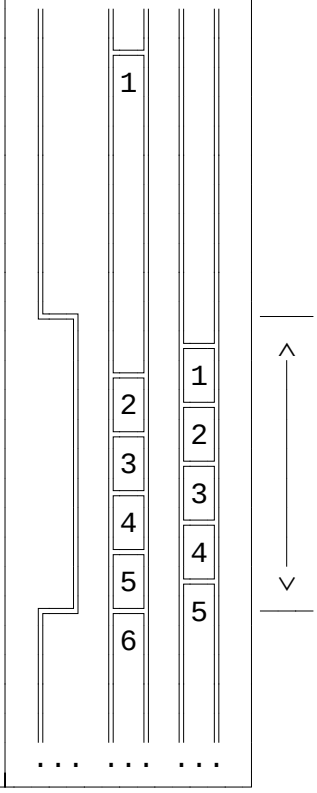
5.2.2.1.3.4. Nachteile statischer Flipflops

Wir betrachten den Automaten aus Beispiel 5.15. Der Zustandsspeicher sei aus statischen D-FFs aufgebaut. Wir nehmen an, daß die Überführungsfunktion und die Flipflops jeweils eine Verzögerungszeit von 5 ns aufweisen. Das Taktsignal habe eine Frequenz von 10 MHz (T_{clock} = 100 ns) und ein Tastverhältnis von 50%. Der Eingang x sei mit 1 belegt.

Wir simulieren das Verhalten des Automatem mit Hilfe eines geeigneten Simlators (VHDL, PSpice, ...).



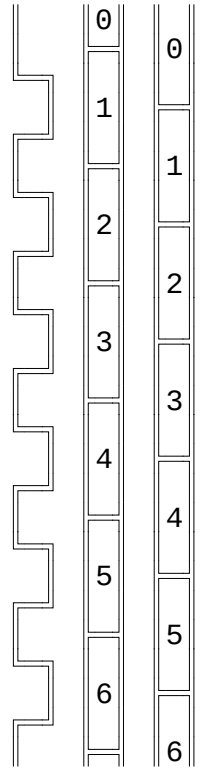
ns	c	zn 210	z 210
0	0	000	000
5	0	001	000
10	0	001	000
15	0	001	000
20	0	001	000
25	0	001	000
30	0	001	000
35	0	001	000
40	0	001	000
45	0	001	000
50	**1**	001	000
55	**1**	001	001
60	**1**	010	001
65	**1**	010	010
70	**1**	011	010
75	**1**	011	011
80	**1**	100	011
85	**1**	100	100
90	**1**	101	100
95	**1**	101	101
100	0	110	101
105	0	110	101
110	0	110	101
115	0	110	101
120	0	110	101
...	.	...	...



Der Automat "läuft durch", solange C = 1 anliegt. Er "schwingt".

Wie läßt sich das "Durchlaufen" verhindern? Das Taktsignal muß offenbar den Wert 0 annehmen, bevor die Überführungsfunktion den nächsten Zustand bereitstellt. Andererseits muß das Taktsignal mindestens solange den Wert 1 annehmen, bis der Zustandsspeicher den nächsten Zustand sicher übernommen hat. Wir wählen T_{clock} = 16 ns und erhalten:

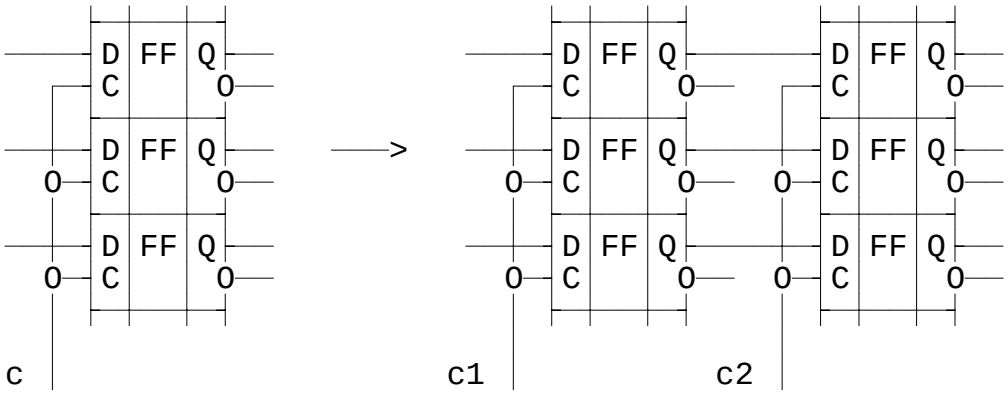
ns	c	zn 210	z 210
0	0	000	000
5	0	001	000
8	1	001	000
13	1	001	001
16	0	001	001
18	0	010	001
24	1	010	001
29	1	010	010
32	0	010	010
34	0	011	010
40	1	011	010
45	1	011	011
48	0	011	011
50	0	100	011
56	1	100	011
61	1	100	100
64	0	100	100
66	0	101	100
72	1	101	100
77	1	101	101
80	0	101	101
82	0	110	101
88	1	110	101
93	1	110	110
96	0	110	110
98	0	111	110
..	.	...	...

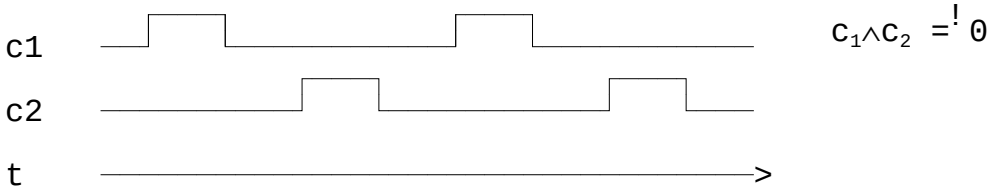


Jetzt gefällt uns das Verhalten des Automaten! Mit jedem Taktimpuls wird der Wert des Zählers um 1 erhöht.

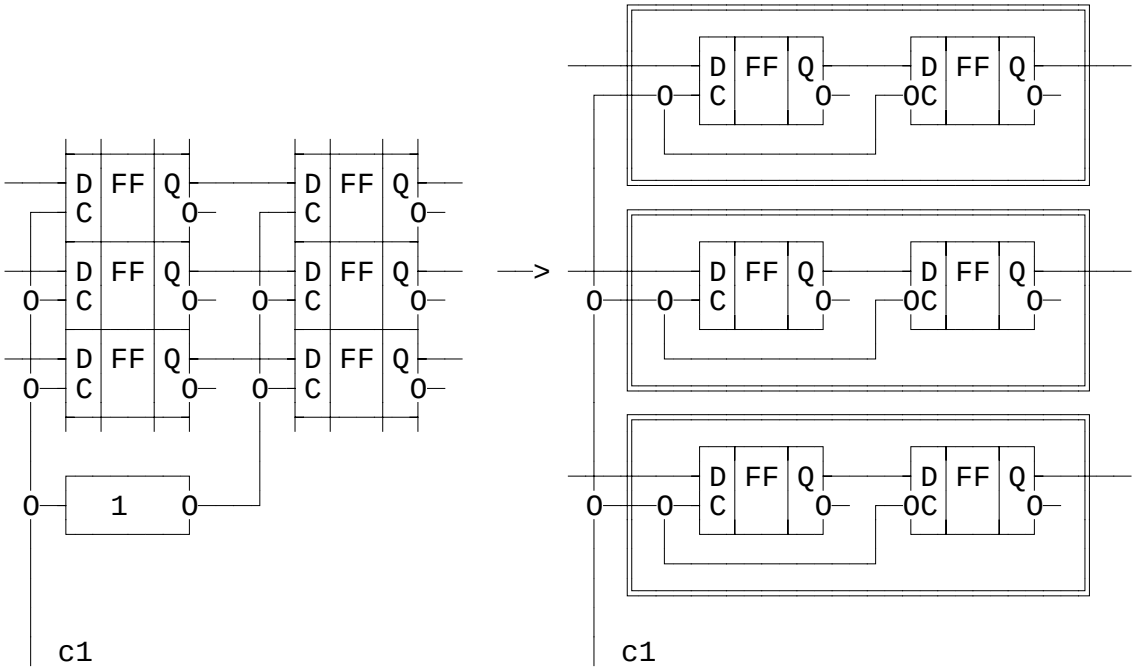
Das Problem der statischen Flipflops ist offenbar ihre Eigenschaft, daß bei aktiver Taktbelegung Änderungen an den Eingangsbelegungen "sofort" auf den Ausgang der Flipflops "durchschlagen" können. Das macht sie für Automaten nur begrenzt geeignet. Kurze Taktimpulse, wie oben verwendet, führen zu erheblichen elektrischen Problemen, und sind als generelle Lösung ungeeignet. Wünschenswert ist ein Verhalten der Flipflops, bei dem unabhängig von der Taktdauer der Automat nicht "durchläuft".

Eine früher oft praktizierte Methode ist das Doppeln des Registers des Zustandsspeichers und das Takten der beiden Register mit orthogonalen Takten.





Die Orthogonalitätsbedingung ist gerade noch erfüllt, wenn gilt $c_2 = /c_1$. Dann können auch zwei in einem Datenstrang liegende statische Flipflops zu einem Doppel-Flipflop zusammengefaßt werden.



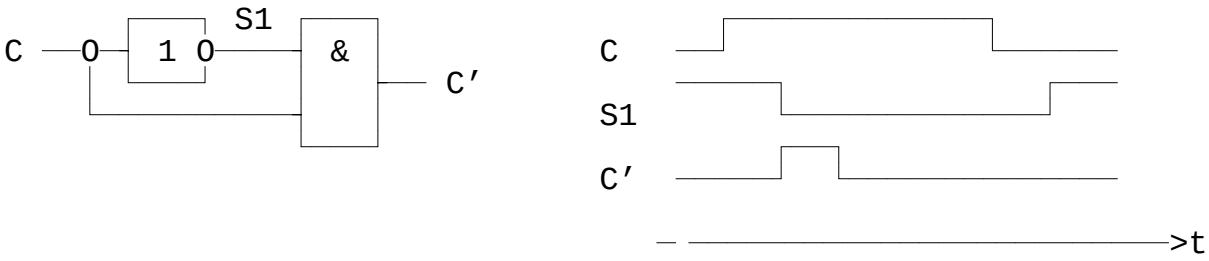
Da das jeweils rechte Teil-Flipflop eines solchen Doppel-Flipflops die Information aus dem jeweils linken Teil-Flipflop "sklavisch" übernimmt, nennt man des rechte Teil-Flipflop "Slave", das linke Teil-Flipflop Master und das gesamte Doppel-Flipflop "Master-Slave-Flip-flop" (MS-FF).

Die Tabellen und der Automatengraph eines MS-FF unterscheiden sich nicht von denen des entsprechenden taktzustandsgesteuerten Flipflops. Man vereinbart, daß eine Kante dann und nur dann durchlaufen wird, wenn das Taktsignal einen vollen Zyklus durchläuft (hier: C = 010). Sonst verharrt das Flipflop in seinem "alten" Zustand.

Eine prinzipiell andere Lösung sind taktflankengesteuerte (dynamische) Flipflops.

5.2.2.1.4. Taktflankengesteuerte (dynamische) Flipflops

Wir hatten in 5.2.2.1.3.4 festgestellt, daß bei hinreichend kurzen Taktsignalen statische Flipflops in Automaten eingesetzt werden können. Die Idee des taktflankengesteuerten Flipflops besteht nun darin, im Flipflop selbst aus der steigenden oder fallenden Flanke (rising edge, falling edge) des Taktsignals durch Differentiation einen hinreichend kurzen Taktimpuls zu gewinnen und damit das Flipflop zu takten.



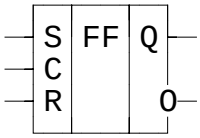
Die Tabellen und der Automatengraph eines taktflankengesteuerten Flipflops unterscheiden sich nicht von denen des entsprechenden taktzustandsgesteuerten Flipflops. Man vereinbart, daß eine Kante dann und nur dann durchlaufen wird, wenn das Taktsignal eine aktive Flanke (hier: steigende Flanke, C = 01) durchläuft. Sonst verharrt das Flip-flop in seinem "alten" Zustand.

Mit der schaltungstechnischen Realisierung der vorgestellten Flipflops befassen Sie sich im Hardwarepraktikum.

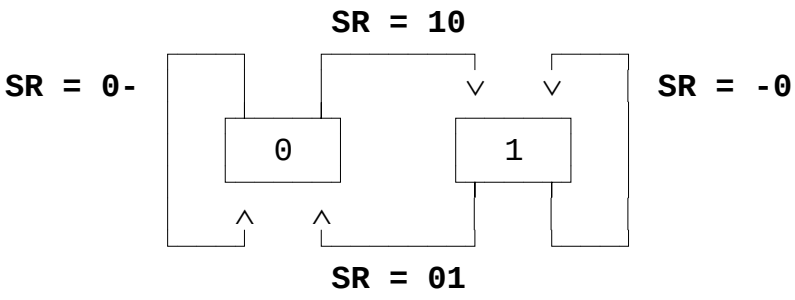
5.2.2.1.5. Zusammenstellung gängiger Flipflop-Typen

Die Art der Taktierung spiegelt sich weder in der Charakteristischen Gleichung, noch in den Tabellen, noch in den Automatengraphen wider. Mit Ausnahme des T- (von engl. to **t**oggle) und des JK-Flipflops (von engl. to **j**ump, to **k**ill oder dem Erfinder **J**ack **K**ilby), die nur als MS-Flipflop oder taktflankengesteuertes Flipflop realisierbar sind (Warum ?), gelten die nachfolgenden Angaben also für alle Varianten der Taktierung. Für jedes Flipflop sind Schaltsymbol, Charakteristische Gleichung, Automatengraph, Flipflop-Tabelle, Automatentabelle und Substitutionstabelle aufgeführt.

RS-FF



$\mathbf{Q' = S \vee \bar{R}Q}$ (mit $S \wedge R \stackrel{!}{=} 0$)

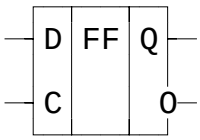


S R	Q'
0 0	Q
0 1	0
1 0	1

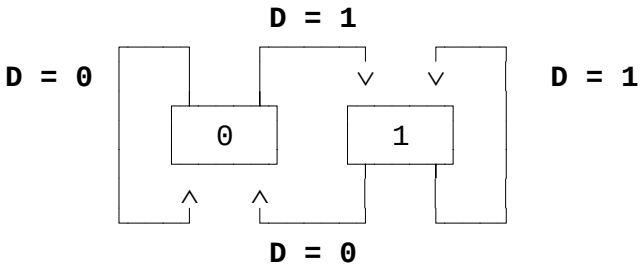
S R	Q	Q'
0 0	0	0
0 0	1	1
0 1	0	0
0 1	1	0
1 0	0	1
1 0	1	1

Q Q'	S R
0 0	0 -
1 1	- 0
1 0	0 1
0 1	1 0

D-FF



$Q' = D$

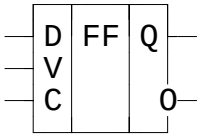


D	Q′
0	0
1	1

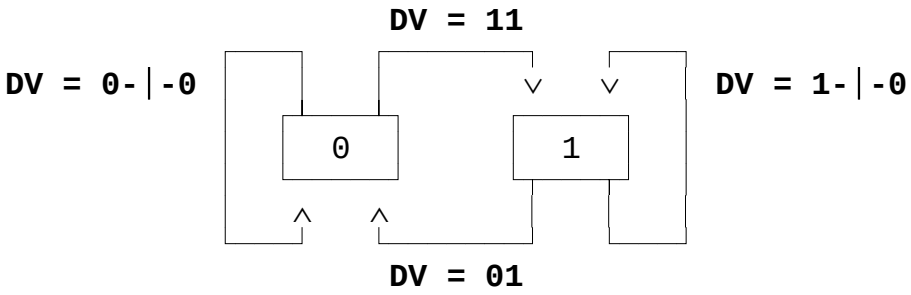
D	Q	Q′
0	0	0
0	1	0
1	0	1
1	1	1

Q	Q′	D
0	0	0
0	1	1
1	0	0
1	1	1

DV-FF



$Q' = DV \vee \overline{V}Q$

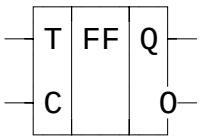


D	V	Q′
0	0	Q
0	1	0
1	0	Q
1	1	1

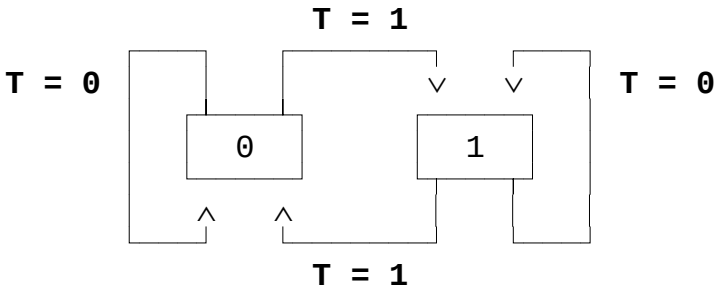
D	V	Q	Q′
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Q	Q′	D	V
0	0	0	-
0	0	-	0
0	1	1	1
1	0	0	1
1	1	1	-
1	1	-	0

T-FF



$Q' = T \not+ Q$

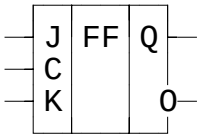


T	Q′
0	Q
1	/Q

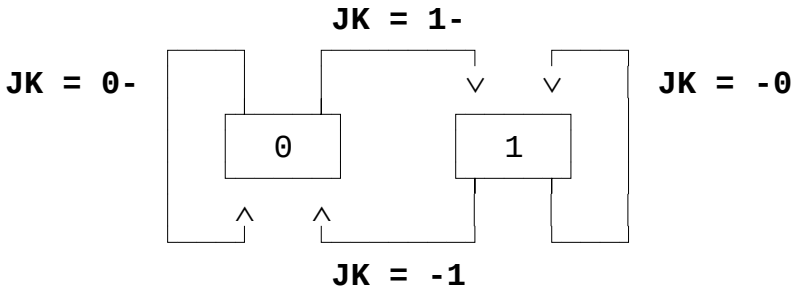
T	Q	Q′
0	0	0
0	1	1
1	0	1
1	1	0

Q	Q′	T
0	0	0
0	1	1
1	0	1
1	1	0

JK-FF



$Q' = J\overline{Q} \vee \overline{K}Q$



J	K	Q′
0	0	Q
0	1	0
1	0	1
1	1	/Q

J	K	Q	Q′
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

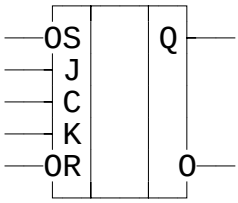
Q	Q′	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

Alle vorgestellten Flipflop-Typen werden Ihnen begegnen. In Automaten werden wir in erster Linie taktflankengesteuerte und Master-Slave-Flipflops einsetzen. Im Hardwarepraktikum werden Sie kennenlernen, daß taktflankengesteuerte und Master-Slave-Flipflops über weite Strecken identisches Verhalten zeigen, daß sie aber durchaus auch Unterschiede aufweisen, die insbesondere bei der Gestaltung des Taktsignals zu beachten sind.

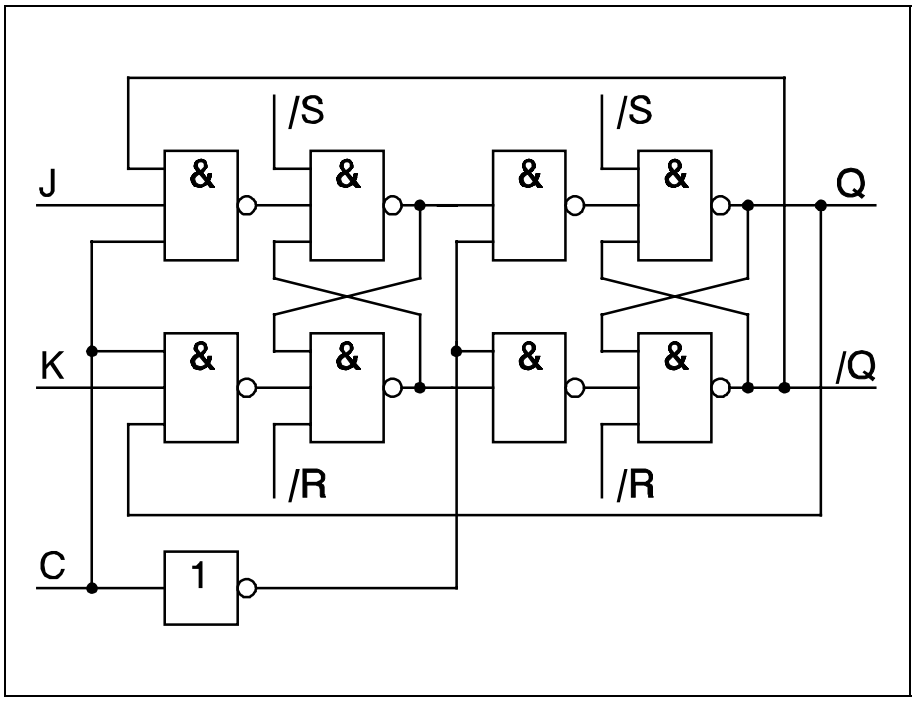
5.2.2.1.6. Handelsübliche Flipflops

Handelsübliche Flipflops, etwa das taktflankengesteuerte D-FF SN7474 oder das JK-MS-FF SN7472, sind zudem statisch setz- und rücksetzbar. Sie vereinen in sich die Eigenschaften eines NAND-RS-GFF und eines taktflankengesteuerten bzw. Master-Slave-Flipflops. Die statische Setz- und Rücksetzbarkeit wird dabei oft zur Einstellung eines definierten Anfangszustands (Initialisierung) des Flipflops verwendet.

Für das JK-MS-FF SN7472 gelten z. B. das Schaltsymbol, die (erweiterte) Flipflop-Tabelle und die "Ersatzschaltung"



/S	/R	C	J	K	Q'	Q_bar'
0	1	-	-	-	1	0
1	0	-	-	-	0	1
0	0	-	-	-	1	1
1	1	⊓	0	0	Q	Q_bar
1	1	⊓	1	0	1	0
1	1	⊓	0	1	0	1
1	1	⊓	1	1	/Q	/Q_bar

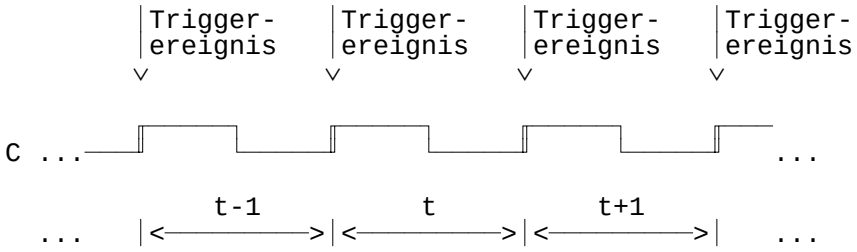


5.2.3. Entwurf synchroner Automaten

5.2.3.1. Grundlagen

Mit der Einführung eines (periodischen) Taktsignals kann die oben angeführte Automatendefinition etwas modifiziert werden. Wir betrachten hier Automaten aus mit der steigenden Taktflanke (rising edge) gesteuerten (getriggerten) Flipflops. Die Überlegungen gelten aber gleichermaßen für mit der fallenden Taktflanke (falling edge) getriggerte Flipflops und Master-Slave-Flipflops.

Das periodische Taktsignal teilt die Zeitskala in gleichlange Zeitabschnitte (Phasen), die wir mit ...,t-1,t,t+1,... bezeichnen. Die Phasen werden getrennt durch das gewählte Triggerereignis (event) - hier die steigende Flanke des Taktsignals.



Die Überführungsfunktion eines Automaten

$f: X \times Z \longrightarrow Z$

gibt, wie bereits besprochen, für jede Eingangsbelegung aus X und jeden "alten" Zustand aus Z an, welcher "neue" Zustand aus Z folgt. "Alter" und "neuer" Zusatnd sind auf der durch das Taktsignak diskretiserten Zeitskale die Zustände des Automaten in zwei unmittelbar aufeinanderfolgenden Phasen, also ewta in den Phasen t-1 ("alt") und t ("neu") oder t ("alt") und t+1 ("neu"). Damit kann man die beiden Funktionen aus der Automatendefinition (hier für den MEALY-Automaten) z. B. so schreiben:

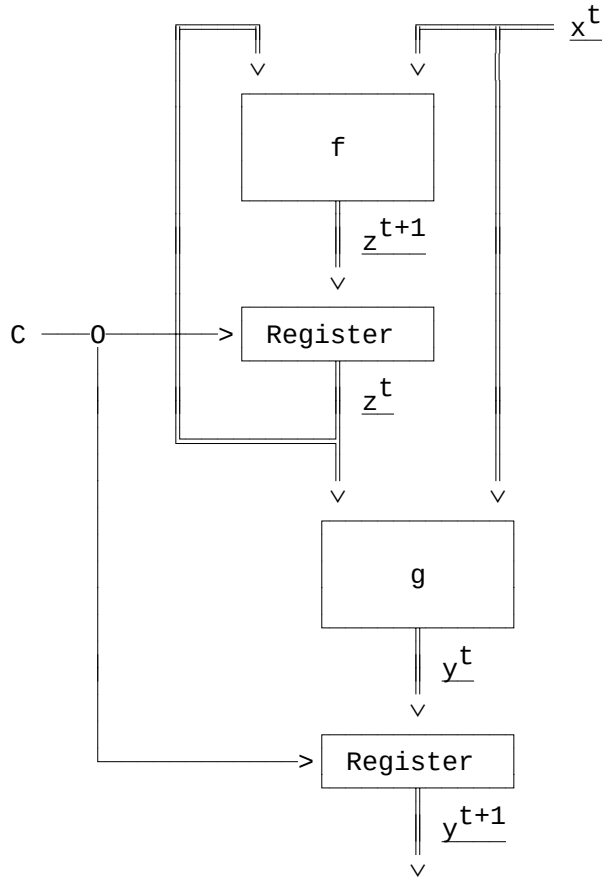
Überführungsfunktion $f: X^t \times Z^t \longrightarrow Z^{t+1}$

Ergebnisfunktion $g: X^t \times Z^t \longrightarrow Y^t$

Beachten Sie, daß in der Überführungsfunktion ein Zusammenhang zwischen zwei aufeinanderfolgenden Phasen hergestellt wird. X^t und Z^t bezeichnen die Eingansgbelegung und den Zustand **vor** dem Triggerereignis, Z^{t+1} den Zusatnd **nach** dem Triggerereignis. Demgegenüber beschreibt die Ergebnisfunktion die Abhängigkeit der Ausgangsbelegung von der Eingangsbelegung und dem Zustand in ein und derselben Phase (vgl. Definition Kombinatorisches System). Die Konsequenz davon ist, daß bei einem MEALY-Automaten durch die Änderung der Eingangsbelegung allein (d. h. ohne Triggerereignis) die Änderung der Ausgangsbelegung verursacht werden kann. Man findet deshalb gelegentlich auch Automatendefinitionen mit

Ergebnisfunktion $g: X^t \times Z^t \longrightarrow Y^{t+1}$

Was bedeutet das? Hier wird offenbar - wie bei der Überführungsfunktion - ein Zusammenhang hergestellt zwischen der Eingangsbelegung und dem Zustand **vor** dem Triggerereignis und der Ausgangsbelegung **nach** dem Triggerereignis. In der Realisierung heißt das, die Ausgangsbelegung muß zwischengespeichert werden.



Wir werden in der Folge die hier eingeführte Indizierung auch auf Flipflops übertragen und anstelle von Q und Q' ab und an auch Q^t und Q^{t+1} bzw. Q^{t-1} und Q^t schreiben.

Def.: Ein Automat heißt synchron, wenn sich bei einem Zustandswechsel alle Flipflops seines Zustandsspeichers - wenn überhaupt - gleichzeitig, d. h. synchron, ändern.

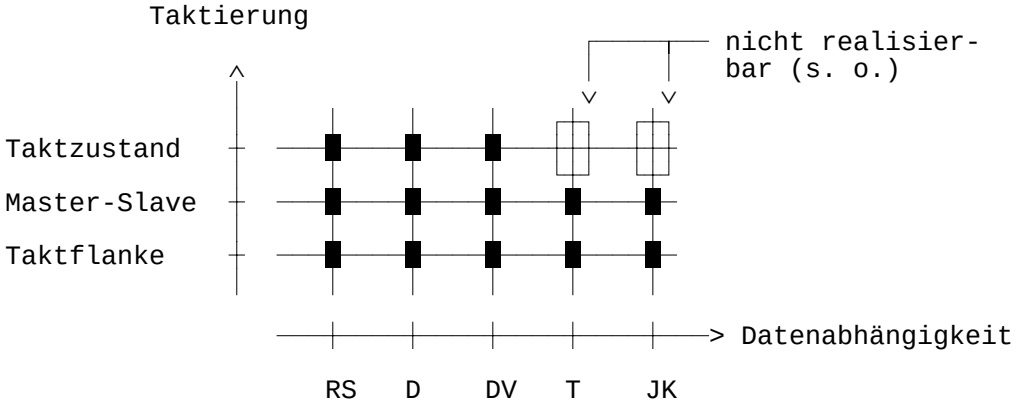
Anm.: Das ist nur möglich, wenn alle Flipflops vom gleichen (von außen zugeführten) Taktsignal und mit dem gleichen Triggerereignis getriggert werden.

5.2.3.2. Flipflop-Substitution

Ein Flipflop ist - wie bereits für das GFF erwähnt - ein Miniautomat vom Typ MEDVEDEV. Der "Entwurf" eines Flipflops ist folglich der Entwurf eines - zugegebenermaßen - sehr kleinen Automaten.

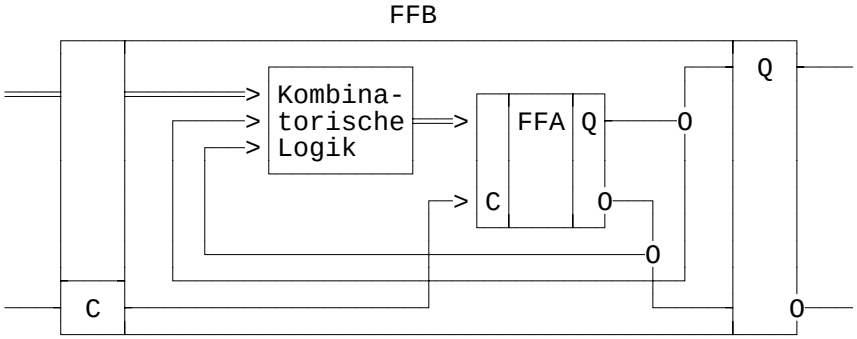
Gegeben sei ein Flipflop vom Typ A, gesucht eine kombinatorische Beschaltung dieses Flipflops so, daß sich die Gesamtschaltung wie ein Flipflop vom Typ B verhält. Das Flipflop B soll also durch ein Flipflop A ersetzt (substituiert) werden.

Läßt sich jedes Flipflop durch jedes andere Flipflop ersetzen? Die oben eingeführten Flipflops sind nach zwei orthogonalen Kriterien kategorisierbar, nach der Art der Taktierung und nach der Art der Abhängigkeit des Schaltverhaltens von den Dateneingängen.



Streng genommen lassen sich nur alle diejenigen Flipflops ineinander überführen (durcheinander ersetzen), die auf die gleiche Weise getaktet werden, in der vorstehenden Grafik also in der gleichen Zeile stehen. Eine Ausnahme sei erwähnt. Wenn ein RS-Flipflop ersetzt werden soll, dann wird das resultierende Flipflop keine "verbotene" Belegung haben können, da außer dem RS-Flipflop keines der anderen Flipflops diese Belegung kennt. Außerdem verhalten sich Master-Slave-Flipflops und taktflankengetriggerte Flipflops über weite Strecken gleich und können also ineinander überführt werden. Die Grenzen sind Gegenstand eines Versuchs des Hardwarepraktikums.

Ein MEDVEDEV-Automat, der sich wie ein Flipflop Typ B verhält und aus einem Flipflop Typ A aufgebaut ist, hat die Struktur:



Anm.: Ggf. muß in die Taktleitung ein Negator eingeschleift werden. Unter welcher Bedingung?

Entwurfsgegenstand ist die kombinatorische Logik. Als "Hand-

werkszeug" benötigen wir die Automatentabelle des Flipflops Typ B und die Substitutionstabelle des Flipflops Typ A.

5.2.3.2.1. JK-FF aus D-FF

FF Typ B = JK-FF

J^t	K^t	Q^t	Q^{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

FF Typ A = D-FF

Q^t	Q^{t+1}	D^t
0	0	0
0	1	1
1	0	0
1	1	1

Schritt 1

Ergänzen der Automatentabelle des JK-FFs durch eine Spalte D^t. Die Werte für D^t ergeben sich für jedes Zustandspaar Q^tQ^{t+1} der Automatentabelle des JK-FFs nach Maßgabe der Substitutionstabelle des D-FFs.

J^t	K^t	Q^t	Q^{t+1}	D^t
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

Anm.: Da für das D-FF gilt $D^t = Q^{t+1}$, kann hier natürlich auf die Spalte für D^t verzichtet werden!

Schritt 2

Eintragen der Funktion $D^t = f(J^t,K^t,Q^t)$ in den KARNAUGH-Plan und Auslesen der minimierten DNF.

D^t		0 1	Q^t
0	0	0 1	
0	1	0 0	
1	1	1 0	
1	0	1 1	
J^t	K^t		

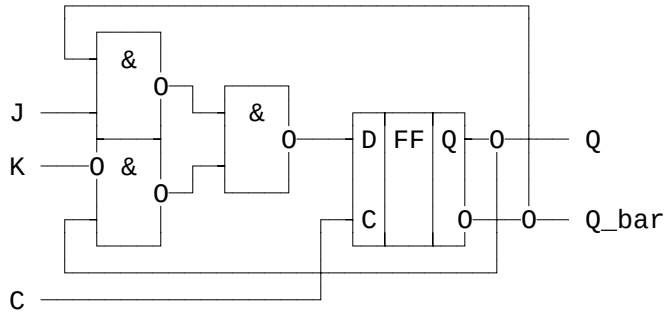
$$D^t = J^t\overline{Q^t} \vee \overline{K^t}Q^t$$

Diese Funktion kommt uns bekannt vor. Sie entspricht der Charakteristischen Gleichung des JK-FFs.

Merke: Bei einem D-FF als Realisierungsbasis (FF Typ A = D-FF)

ergibt sich die Beschaltung des D-FFs stets direkt aus der Charakteristischen Gleichung des zu realisierenden Flipflops (FF Typ B).

Schritt 3 - Schaltung

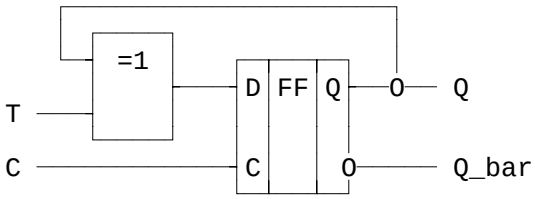


5.2.3.2.2. T-FF aus D-FF

Wir nutzen die in 5.2.3.2.1. gewonnene Erkenntnis und finden sofort

$$D^t = T^t \not+ Q^t$$

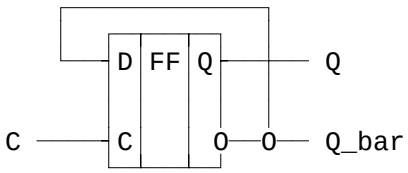
und die Schaltung



In Zählern und Frequenzteilern ist häufig ein Flipflop anzutreffen, das immer toggelt, h. h. T = 1. Dafür ergibt sich

$$D^t = 1 \not+ Q^t = \overline{Q^t}$$

und die Schaltung



Was tut diese Schaltung, wenn für das D-FF ein taktzustandsgesteuertes Flipflop eingesetzt wird?

5.2.3.2.3. JK-FF aus RS-FF

FF Typ B = JK-FF

J^t	K^t	Q^t	Q^{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

FF Typ A = RS-FF

Q^t	Q^{t+1}	S^t	R^t
0	0	0	-
0	1	1	0
1	0	0	1
1	1	-	0

Schritt 1

Ergänzen der Automatentabelle des JK-FFs durch die Spalten S^t und R^t. Die Werte für S^t und R^t ergeben sich für jedes Zustandspaar Q^tQ^{t+1} der Automatentabelle des JK-FFs nach Maßgabe der Substitutionstabelle des RS-FFs.

J^t	K^t	Q^t	Q^{t+1}	S^t	R^t
0	0	0	0	0	-
0	0	1	1	-	0
0	1	0	0	0	-
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	-	0
1	1	0	1	1	0
1	1	1	0	0	1

Schritt 2

Eintragen der Funktionen $S^t = f(J^t,K^t,Q^t)$ und $R^t = f(J^t,K^t,Q^t)$ in den KARNAUGH-Plan und Auslesen der minimierten DNF.

S^t	0	1
0 0	0	-
0 1	0	0
1 1	1	0
1 0	1	-

Q^t

J^t K^t

$S^t = J^t\overline{Q^t}$

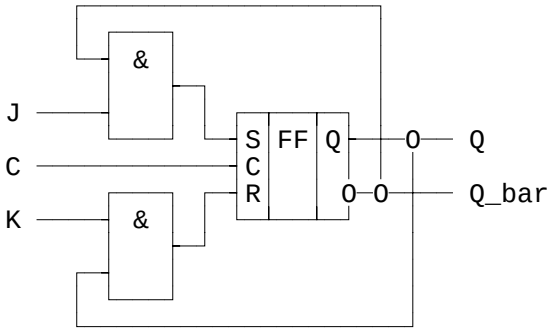
R^t	0	1
0 0	-	0
0 1	-	1
1 1	0	1
1 0	0	0

Q^t

J^t K^t

$R = K^tQ^t$

Schritt 3 - Schaltung



Die gefundene Schaltung entspricht der in 5.2.2.6. angegebenen Ersatzschaltung für das handelsübliche JK-MS-FF SN7472.

5.2.3.2.4. RS-FF aus T-FF

FF Typ B = RS-FF

S^t	R^t	Q^t	Q^{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

FF Typ A = T-FF

Q^t	Q^{t+1}	T^t
0	0	0
0	1	1
1	0	1
1	1	0

Schritt 1

Ergänzen der Automatentabelle des RS-FFs durch eine Spalte T^t. Die Werte für T^t ergeben sich für jedes Zustandspaar Q^tQ^{t+1} der Automatentabelle des RS-FFs nach Maßgabe der Substitutionstabelle des T-FFs.

S^t	R^t	Q^t	Q^{t+1}	T^t
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	X	?
1	1	1	X	?

Für die Zustandsübergänge Q^tQ^{t+1} = 0X|1X finden sich in der Substitutionstabelle keine Eintragungen. Man ist hier gezwungen, die Folgezustände Q^{t+1} geeignet zu wählen. Wählt man etwa 0X ⟶ 01 und 1X ⟶ 10, dann erhält man ein JK-FF. Wählt man dagegen 0X ⟶ 00 und 1X ⟶ 11, dann funktioniert man die "verbotene"

Belegung um zu "Speichern". Das wollen wir hier tun und erhalten

S^t	R^t	Q^t	Q^{t+1}	T^t
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	0

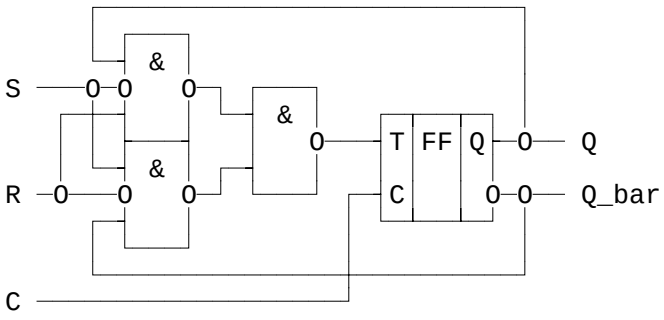
Schritt 2

Eintragen der Funktion $T^t = f(S^t,R^t,Q^t)$ in den KARNAUGH-Plan und Auslesen der minimierten DNF.

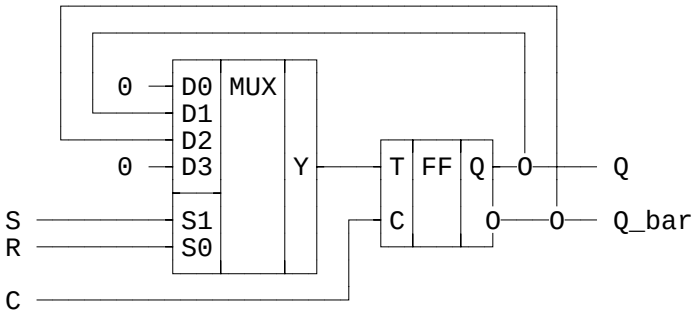
T^t		Q^t
	0 1	
0 0	0 0	
0 1	0 1	
1 1	0 0	
1 0	1 0	
S^t R^t		

$$T^t = \overline{S^t}\, R^t\, Q^t \;\vee\; S^t\, \overline{R^t}\, \overline{Q^t}$$

Schritt 3 - Schaltung



oder vielleicht auch



5.2.3.3. Automatenentwurf

Wir wollen den Automatenentwurf anhand der Beispiele 5.14. und 5.15. kennenlernen.

zu Beispiel 5.14.

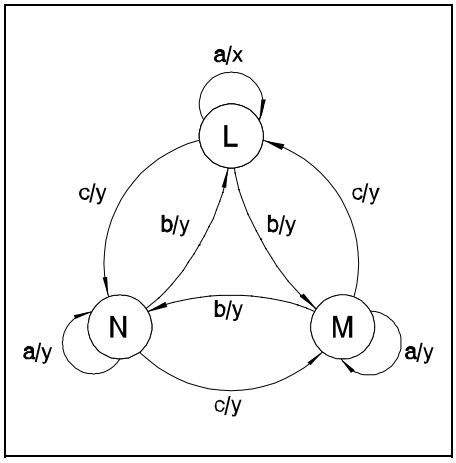
Gegeben ist ein MEALY-Automat mit:

X = {a,b,c}
Y = {x,y}
Z = {L,M,N}

f = {(a,L,L),(b,L,M),(c,L,N),
(a,M,M),(b,M,N),(c,M,L),
(a,N,N),(b,N,L),(c,N,M)}

g = {(a,L,x),(b,L,y),(c,L,y),
(a,M,y),(b,M,y),(c,M,y),
(a,N,y),(b,N,y),(c,N,y)}

Automatengraph



Automatentabelle

x^t	z^t	z^{t+1}	y^t
a	L	L	x
a	M	M	y
a	N	N	y
b	L	M	y
b	M	N	y
b	N	L	y
c	L	N	y
c	M	L	y
c	N	M	y

Gesucht ist eine binäre Realisierung des Automaten.

Schritt 1 - Binäre Kodierung

Binär kodiert werden die Mengen X, Y und Z. Die Kodierung erfolgt in zwei Schritten. Im ersten Schritt wird die Breite der entsprechenden Binärvektoren ermittelt, im zweiten Schritt erfolgt die eigentliche Kodierung der Elemente der Mengen.

ld |X| = ld 3 ——> Breite des Eingangsvektors = 2 Bit

ld |Y| = ld 2 ——> Breite des Ausgangsvektors = 1 Bit

ld |Z| = ld 3 ——> Breite des Zustandsvektors = 2 Bit

Die eigentliche Kodierung kann beliebig gewählt werden. Es kann aber Nebenbedingungen geben, die eine bestimmte Kodierung nahelegen. Wenn etwa der Zustand L als Anfangszustand (Initialzustand) gewählt wird und wenn dieser Zustand mit dem bei handelsüblichen Flipflops häufig möglichen, statischen Rücksetzen er-

reicht werden soll, dann wird man sinvoll L = 00 kodieren.

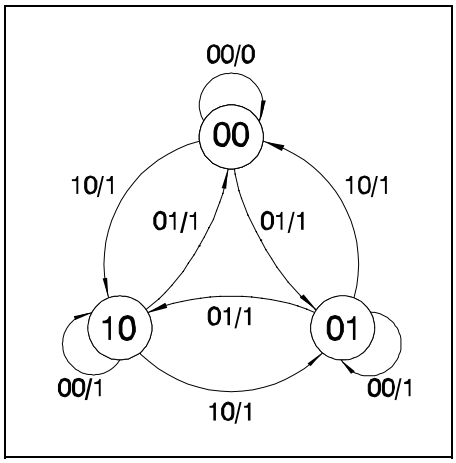
Wir wählen für die Elemente der Menge X a = 00, b = 01, c = 10, für die Elemente der Menge Y x = 0, y = 1 und für die Elemente der Menge Z L = 00, M = 01, N = 10 und erhalten

X = {00,01,10}
Y = {0,1}
Z = {00,01,10}

f = {(00,00,00),(01,00,01),(10,00,10),
(00,01,01),(01,01,10),(10,01,00),
(00,10,10),(01,10,00),(10,10,01)}

g = {(00,00,0),(01,00,1),(10,00,1),
(00,01,1),(01,01,1),(10,01,1),
(00,10,1),(01,10,1),(10,10,1)}

Automatengraph



Automatentabelle

x^t z^t	z^{t+1}	y^t
00 00	00	0
00 01	01	1
00 10	10	1
01 00	01	1
01 01	10	1
01 10	00	1
10 00	10	1
10 01	00	1
10 10	01	1

Die Kodierungen x = 11 und z = 11 sind möglich, werden aber nicht benötigt und folglich (zunächst) nicht benutzt. Welche Konsequenzen hat das?

1. Wenn man die Eingangsbelegung x = 11 an den Automaten anlegt, "weiß" der Automat nicht, was er tun soll. Zwei Auswege sind vorstellbar. Entweder man verbietet in der Spezifikation des Automaten die Eingangsbelegung x = 11 (dann kann sie aber immer noch im Fehlerfalle auftreten!) oder man definiert, was der Automat bei x = 11 tun soll. Man könnte z. B. definieren, daß der Automat bei x = 11 immer in den Initialzustand z = 00 geht. Man könnte aber z. B. auch definieren, daß der Automat bei x = 11 das Gleiche tut, wie bei x = 10. Wir würden also kodieren c = 10|11 = 1-. So wollen wir die Automatendefinition für den weiteren Entwurf modifizieren.

2. Kann der Zustand z = 11 in der binären Realisierung des Automaten überhaupt vorkommen? Scheinbar nicht!? Doch, er kann vorkommen! Unmittelbar nach Anlegen der Stromversorgung an ein Flipflop und vor dem ersten Rücksetzen bzw. Setzen befindet sich das Flipflop im Speicherzustand und nimmt nicht vorhersehbar einen der beiden möglichen Zustände QQbar = 01|10 ein. Unser Automat kann also nach dem

Anlegen der Stromversorgung nicht vorhersehbar einen der vier Zustände z = 00|01|10|11 einnehmen. Zwei Auswege sind vorstellbar. Entweder man macht den Automaten aus jedem Zustand heraus rücksetzbar (statisch oder dynamisch) oder man definiert, in welchen Zustand der Automat (unabhängig von der Eingangsbelegung) gehen soll, wenn er sich im Zustand z = 11 befindet. Wir wollen hier annehmen, daß er in den Zustand z = 01 gehen soll. Es bleibt noch die Frage offen, was der Automat ausgeben soll, wenn er sich im Zustand z = 11 befindet. Wir definieren hier, daß er unabhängig von der Eingangsbelegung y = 1 ausgibt.

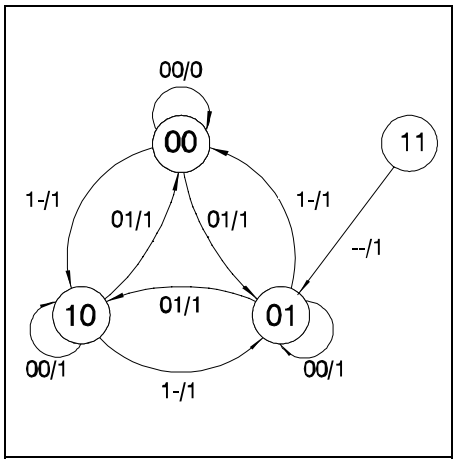
Mit diesen Überlegungen erhalten wir

X = {00,01,1-}
Y = {0,1}
Z = {00,01,10,11}

f = {(00,00,00),(01,00,01),(1-,00,10),
(00,01,01),(01,01,10),(1-,01,00),
(00,10,10),(01,10,00),(1-,10,01),
(--,11,01)}

g = {(00,00,0),(01,00,1),(1-,00,1),
(00,01,1),(01,01,1),(1-,01,1),
(00,10,1),(01,10,1),(1-,10,1),
(--,11,1)}

Automatengraph



Automatentabelle

x^t z^t	z^{t+1}	y^t
00 00	00	0
00 01	01	1
00 10	10	1
00 11	01	1
01 00	01	1
01 01	10	1
01 10	00	1
01 11	01	1
10 00	10	1
10 01	00	1
10 10	01	1
10 11	01	1
11 00	10	1
11 01	00	1
11 10	01	1
11 11	01	1

Schritt 2 - Bereitstellen des "Handwerkszeugs"

Wir wollen den Automaten mit Hilfe von mit der steigenden Taktflanke getriggerten JK-FFs realisieren und benötigen folglich die Substitutionstabelle des JK-FFs.

Q^t	Q^{t+1}	J^t	K^t
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

Schritt 3 - Ergänzen der Automatentabelle

Da der Zustandsvektor 2 Bit breit ist, benötigen wir zwei JK-FFs FF1 und FF0 mit den Eingängen J1, K1 und J0, K0.

x_1^t	x_0^t	z_1^t	z_0^t	z_1^{t+1}	z_0^{t+1}	j_1^t	k_1^t	j_0^t	k_0^t	y^t
0	0	0	0	0	0	0	-	0	-	0
0	0	0	1	0	1	0	-	-	0	1
0	0	1	0	1	0	-	0	0	-	1
0	0	1	1	0	1	-	1	-	0	1
0	1	0	0	0	1	0	-	1	-	1
0	1	0	1	1	0	1	-	-	1	1
0	1	1	0	0	0	-	1	0	-	1
0	1	1	1	0	1	-	1	-	0	1
1	0	0	0	1	0	1	-	0	-	1
1	0	0	1	0	0	0	-	-	1	1
1	0	1	0	0	1	-	1	1	-	1
1	0	1	1	0	1	-	1	-	0	1
1	1	0	0	1	0	1	-	0	-	1
1	1	0	1	0	0	0	-	-	1	1
1	1	1	0	0	1	-	1	1	-	1
1	1	1	1	0	1	-	1	-	0	1

Die Werte für $j_1^t k_1^t$ ergeben sich aus den Wertepaaren $z_1^t z_1^{t+1}$, die Werte für $j_0^t k_0^t$ aus den Wertepaaren $z_0^t z_0^{t+1}$ jeweils nach Maßgabe der Substitutionstabelle des JK-FFs.

Schritt 4 - Ansteuerfunktionen

Eintragen der Funktionen

$$j_1^t = f(x_1^t, x_0^t, z_1^t, z_0^t)$$

$$k_1^t = f(x_1^t, x_0^t, z_1^t, z_0^t),$$

$$j_0^t = f(x_1^t, x_0^t, z_1^t, z_0^t)$$

$$k_0^t = f(x_1^t, x_0^t, z_1^t, z_0^t)$$

in KARNAUGH-Pläne und Auslesen der minimierten DNF.

j_1	0 0 1 1	z_1
	0 1 1 0	z_0
0 0	0 0 - -	
0 1	0 1 - -	
1 1	1 0 - -	
1 0	1 0 - -	
x_1x_0		

k_1	0 0 1 1	z_1
	0 1 1 0	z_0
0 0	- - 1 0	
0 1	- - 1 1	
1 1	- - 1 1	
1 0	- - 1 1	
x_1x_0		

$j_1 = x_1\overline{z_0} \vee \overline{x_1}x_0z_0$ $\qquad$ $k_1 = x_1 \vee x_0 \vee z_0$

j_0	0 0 1 1	z_1
	0 1 1 0	z_0
0 0	0 - - 0	
0 1	1 - - 0	
1 1	0 - - 1	
1 0	0 - - 1	
x_1x_0		

k_0	0 0 1 1	z_1
	0 1 1 0	z_0
0 0	- 0 0 -	
0 1	- 1 0 -	
1 1	- 1 0 -	
1 0	- 1 0 -	
x_1x_0		

$j_0 = \overline{x_1}x_0\overline{z_1} \vee x_1z_1$ $\qquad$ $k_0 = x_0\overline{z_1} \vee x_1\overline{z_1}$

Schritt 5 - Ergebnisfunktion

Aus der Automatentabelle ist direkt ablesbar

$$\overline{y^t} = \overline{x_1^t}\ \overline{x_0^t}\ \overline{z_1^t}\ \overline{z_0^t} \quad \longrightarrow \quad y^t = x_1^t \vee x_0^t \vee z_1^t \vee z_0^t$$

Schritt 6 - Schaltung

Anstelle der Schaltung gebe ich hier eine VHDL-Verhaltensbeschreibung an. VHDL ist eine gut lesbare Sprache, die auch dann verständlich ist, wenn man die Sprache nicht oder nur lückenhaft kennt. Im 2. Semester lernen Sie die Sprache im Fach Rechnerorganisation genauer kennen.

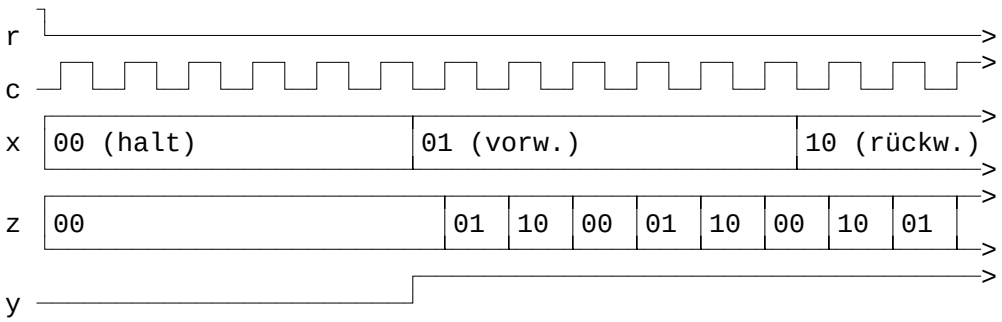
```
entity test is
end test;

architecture test of test is
  signal x,j,k,z : bit_vector(1 downto 0);
  signal r,c,y : bit;
begin
  r <= '1','0' after 5 ns;
  c <= not c after 10 ns;
  x <= "00","01" after 120 ns,"10" after 240 ns,"11" after 360 ns;
```

```
  process (r,c) begin
    if r = '1' then
       z <= "00";
    elsif c'event and c = '1' then
       z(0) <= (j(0) and not z(0)) or (not k(0) and z(0));
       z(1) <= (j(1) and not z(1)) or (not k(1) and z(1));
    end if;
  end process;
  j(1) <= (x(1) and not z(0)) or (not x(1) and x(0) and z(0));
  k(1) <= x(1) or x(0) or z(0);
  j(0) <= (not x(1) and x(0) and not z(1)) or (x(1) and z(1));
  k(0) <= (x(0) and not z(1)) or (x(1) and not z(1));
  y    <= x(1) or x(0) or z(1) or z(0);
end test;
```

Die Simulation ergibt

ns	r	c	x	z	y	ns	r	c	x	z	y	ns	r	c	x	z	y	ns	r	c	x	z	y
0	1	0	00	00	0																		
5	0	0	00	00	0	120	0	0	01	00	1	240	0	0	10	00	1	360	0	0	11	00	1
10	0	1	00	00	0	130	0	1	01	01	1	250	0	1	10	10	1	370	0	1	11	10	1
20	0	0	00	00	0	140	0	0	01	01	1	260	0	0	10	10	1	380	0	0	11	10	1
30	0	1	00	00	0	150	0	1	01	10	1	270	0	1	10	01	1	390	0	1	11	01	1
40	0	0	00	00	0	160	0	0	01	10	1	280	0	0	10	01	1	400	0	0	11	01	1
50	0	1	00	00	0	170	0	1	01	00	1	290	0	1	10	00	1	410	0	1	11	00	1
60	0	0	00	00	0	180	0	0	01	00	1	300	0	0	10	00	1	420	0	0	11	00	1
70	0	1	00	00	0	190	0	1	01	01	1	310	0	1	10	10	1	430	0	1	11	10	1
80	0	0	00	00	0	200	0	0	01	01	1	320	0	0	10	10	1	440	0	0	11	10	1
90	0	1	00	00	0	210	0	1	01	10	1	330	0	1	10	01	1	450	0	1	11	01	1
100	0	0	00	00	0	220	0	0	01	10	1	340	0	0	10	01	1	460	0	0	11	01	1
110	0	1	00	00	0	230	0	1	01	00	1	350	0	1	10	00	1	470	0	1	11	00	1

oder

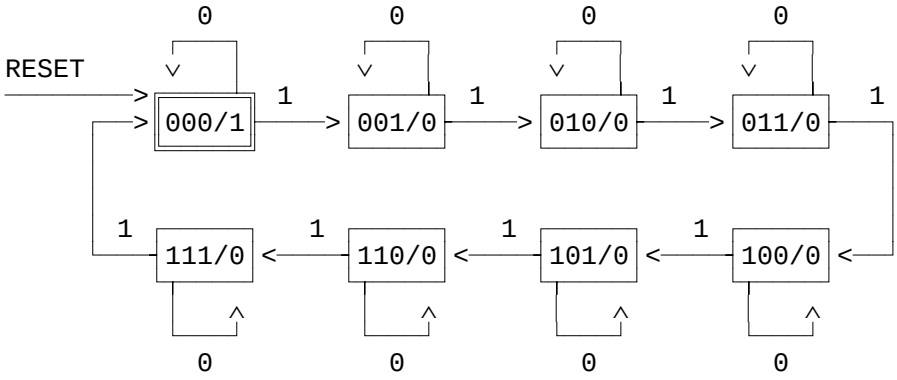


zu Beispiel 5.15.

Ein 3-Bit-Binärzähler soll bei x = 1 in Schritten von 1 vorwärtszählen und bei x = 0 nicht zählen. Bei z = 000 soll er y = 1 ausgeben, sonst y = 0. Der Zustand z = 000 sei der Initialzustand (Anfangszustand). Es liegt ein MOORE-Automat vor.

X = {0,1}, Y = {0,1}, Z = {000,001,010,011,100,101,110,111}

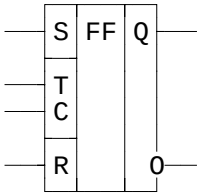
Automatengraph



Automatentabelle

x^t	$z_2^t z_1^t z_0^t$	$z_2^{t+1} z_1^{t+1} z_0^{t+1}$	y^t
0	0 0 0	0 0 0	1
0	0 0 1	0 0 1	0
0	0 1 0	0 1 0	0
0	0 1 1	0 1 1	0
0	1 0 0	1 0 0	0
0	1 0 1	1 0 1	0
0	1 1 0	1 1 0	0
0	1 1 1	1 1 1	0
1	0 0 0	0 0 1	1
1	0 0 1	0 1 0	0
1	0 1 0	0 1 1	0
1	0 1 1	1 0 0	0
1	1 0 0	1 0 1	0
1	1 0 1	1 1 0	0
1	1 1 0	1 1 1	0
1	1 1 1	0 0 0	0

Gesucht ist eine Realisierung aus statisch setz- und rücksetzbaren T-MS-Flipflops.



S R	C	T	Q’	Q_bar’
0 1	-	-	0	1
1 0	-	-	1	0
1 1	-	-	1	1
0 0	⊔	0	Q	Q_bar
0 0	⊔	1	/Q	/Q_bar

Schritt 1 - binäre Kodierung

entfällt, da die Automatenbeschreibung bereits binär kodiert vorliegt

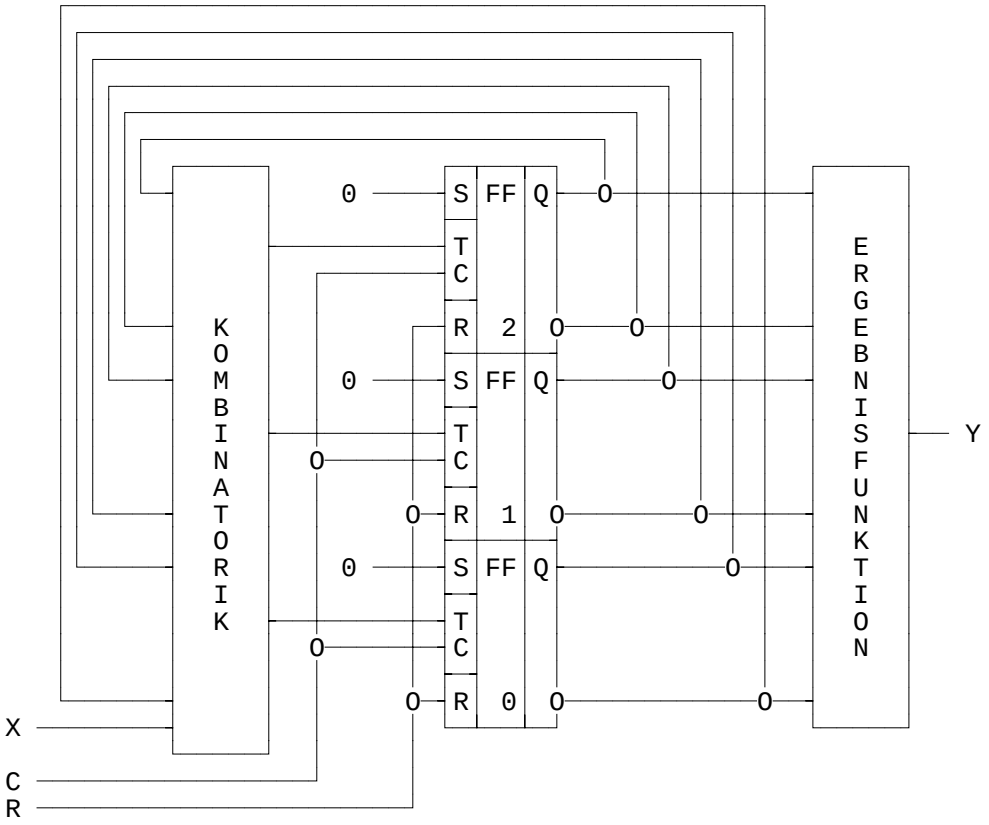
Schritt 2 - Bereitstellen des "Handwerkszeugs"

Substitutionstabelle des T-FFs

Q^t	Q^{t+1}	T^t
0	0	0
0	1	1
1	0	1
1	1	0

Schritt 3 - Ergänzen der Automatentabelle

Gegenstand des Automatenentwurfs ist nur die Ermittlung der Ansteuerfunktionen für die T-Eingänge. Die statische Rücksetzbarkeit in den Zustand z = 000 wird mit SR = 01 realisiert und dem Entwurfsergebnis "überlagert" - Superpositionsentwurf (s. Hardwarepraktikum, Versuch Sequentielle Systeme 2).



Die Werte für die t^t ergeben sich aus den Wertepaaren $z^t z^{t+1}$ für i = 0,1,2 jeweils nach Maßgabe der Substitutionstabelle des T-FFs.

x^t	$z_2^t z_1^t z_0^t$	$z_2^{t+1} z_1^{t+1} z_0^{t+1}$	$t_2^t\ t_1^t\ t_0^t$	y^t
0	0 0 0	0 0 0	0 0 0	1
0	0 0 1	0 0 1	0 0 0	0
0	0 1 0	0 1 0	0 0 0	0
0	0 1 1	0 1 1	0 0 0	0
0	1 0 0	1 0 0	0 0 0	0
0	1 0 1	1 0 1	0 0 0	0
0	1 1 0	1 1 0	0 0 0	0
0	1 1 1	1 1 1	0 0 0	0
1	0 0 0	0 0 1	0 0 1	1
1	0 0 1	0 1 0	0 1 1	0
1	0 1 0	0 1 1	0 0 1	0
1	0 1 1	1 0 0	1 1 1	0
1	1 0 0	1 0 1	0 0 1	0
1	1 0 1	1 1 0	0 1 1	0
1	1 1 0	1 1 1	0 0 1	0
1	1 1 1	0 0 0	1 1 1	0

Schritt 4 - Ansteuerfunktionen

Eintragen der Funktionen

$t_2^t = f(x^t, z_2^t, z_1^t, z_0^t)$

$t_1^t = f(x^t, z_2^t, z_1^t, z_0^t),$

$t_0^t = f(x^t, z_2^t, z_1^t, z_0^t)$

in KARNAUGH-Pläne und Auslesen der minimierten DNF.

t_2	0 0 1 1	z_1
	0 1 1 0	z_0
0 0	0 0 0 0	
0 1	0 0 0 0	
1 1	0 0 1 0	
1 0	0 0 1 0	
x	z_2	

t_1	0 0 1 1	z_1
	0 1 1 0	z_0
0 0	0 0 0 0	
0 1	0 0 0 0	
1 1	0 1 1 0	
1 0	0 1 1 0	
x	z_2	

$t_2 = xz_1z_0$

$t_1 = xz_0$

t_0	0 0 1 1	z_1
	0 1 1 0	z_0
0 0	0 0 0 0	
0 1	0 0 0 0	
1 1	1 1 1 1	
1 0	1 1 1 1	
x	z_2	

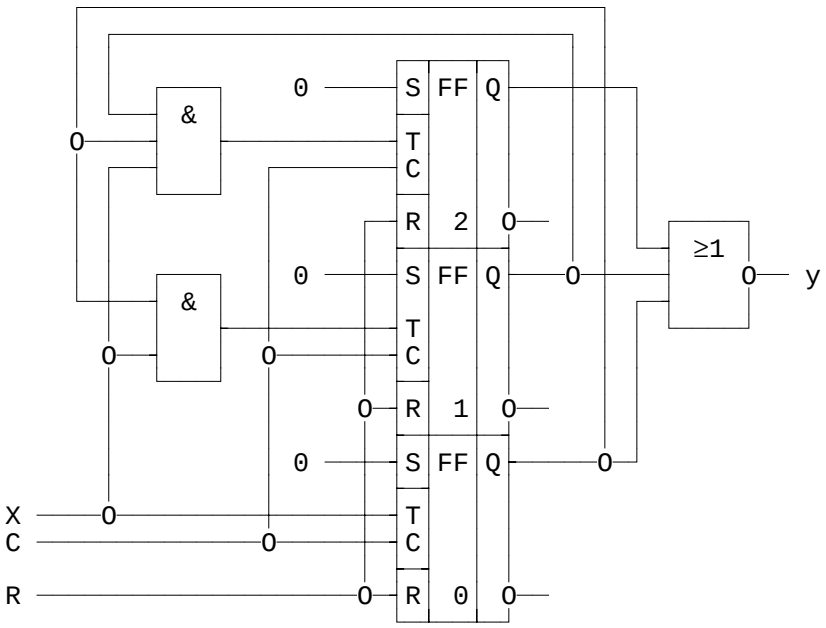
$t_0 = x$

Schritt 5 - Ergebnisfunktion

y	0 0 1 1	z_1
	0 1 1 0	z_0
0 0	1 0 0 0	
0 1	0 0 0 0	
1 1	0 0 0 0	
1 0	1 0 0 0	
x z_2		

$$y = \overline{z_2}\ \overline{z_1}\ \overline{z_0} = \overline{\overline{z_2}\ \overline{z_1}\ \overline{z_0}} = \overline{z_2 \vee z_1 \vee z_0}$$

Schritt 6 - Schaltung



VHDL-Verhaltensbeschreibung

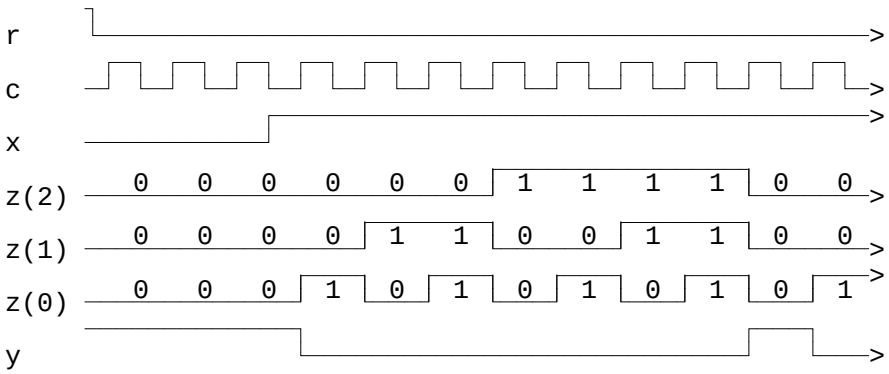
```
entity test is
end test;

architecture test of test is
  signal t,z : bit_vector(2 downto 0);
  signal r,c,x,y : bit;
begin
  r <= '1','0' after 5 ns;
  c <= not c after 10 ns;
  x <= '0','1' after 60 ns;
```

```
  process (r,c) begin
    if r = '1' then
       z <= "000";
    elsif c'event and c = '1' then
       z(2) <= (t(2) xor z(2));
       z(1) <= (t(1) xor z(1));
       z(0) <= (t(0) xor z(0));
    end if;
  end process;
  t(2) <= x and z(1) and z(0);
  t(1) <= x and z(0);
  t(0) <= x;
  y    <= not (z(2) or z(1) or z(0));
end test;
```

Simulationsergebnisse

```
 ns r c x   z y

  0 1 0 0 000 1
  5 0 0 0 000 1
 10 0 1 0 000 1
 20 0 0 0 000 1
 30 0 1 0 000 1
 40 0 0 0 000 1
 50 0 1 0 000 1
 60 0 0 1 000 1
 70 0 1 1 001 0
 80 0 0 1 001 0
 90 0 1 1 010 0
100 0 0 1 010 0
110 0 1 1 011 0
120 0 0 1 011 0
130 0 1 1 100 0
140 0 0 1 100 0
150 0 1 1 101 0
160 0 0 1 101 0
170 0 1 1 110 0
180 0 0 1 110 0
190 0 1 1 111 0
200 0 0 1 111 0
210 0 1 1 000 1
220 0 0 1 000 1
230 0 1 1 001 0
240 0 0 1 001 0
```

oder

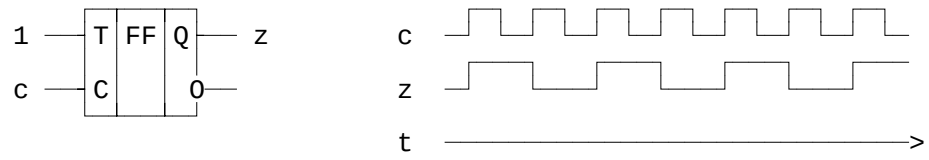


5.2.4. Entwurf asynchroner Automaten

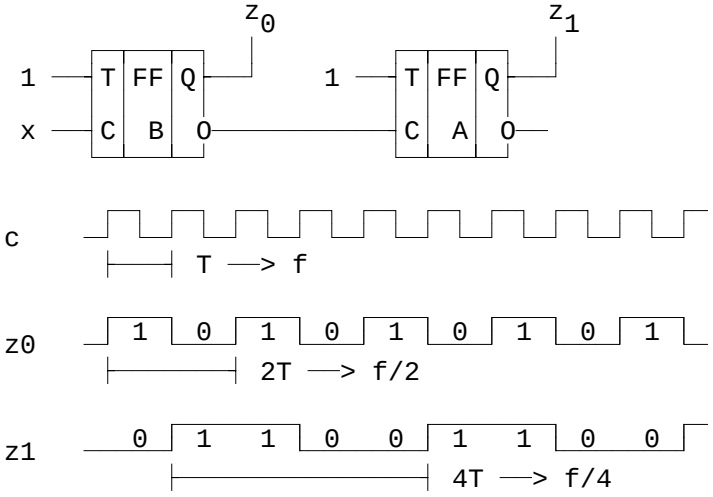
Die kombinatorische Logik, die in einem synchronen Automaten die Belegungen der Dateneingänge der Flipflops bereitstellt, muß **für jedes Flipflop bei jedem Triggerereignis** entscheiden, was das Flipflop tun soll (Setzen, Rücksetzen, Speichern, Toggeln).

Nötig ist das Triggerereignis für ein bestimmtes Flipflop aber nur bei all den Zustandsübergängen eines Automaten, bei denen sich der Zustand dieses Flipflops ändert. Soll ein Flipflop bei einem Zustandsübergang des Automaten in seinem vorherigen Zustand verharren, bestünde auch die Möglichkeit, dieses Flipflop bei diesem Zustandsübergang des Automaten überhaupt nicht mit dem Triggerereignis zu beaufschlagen. Die kombinatorische Logik, die die Belegungen der Dateneingänge der Flipflops bereitstellt, müßte für dieses Flipflop bei diesem Zustandsübergang des Automaten keine Entscheidung fällen und könnte vermutlich einfacher ausfallen.

Aus dem Entwurf synchroner Automaten wissen wir, daß sich die Ausgangsbelegung eines Flipflops höchstens immer dann ändert, wenn das Triggerereignis auftritt. Sie ändert sich also grundsätzlich seltener, als das Taktsignal selbst.



Wenn es gelingt, in einem Automaten ein Flipflop B zu finden, dessen Ausgangsbelegung sich mindestens immer dann ändert, wenn ein anderes Flipflop A getriggert werden muß, wenn sich also dessen Ausgangsbelegung ändern soll, dann kann Flipflop A mit der Ausgangsbelegung von Flipflop B getriggert werden.



Die vorstehende Schaltung kann entweder als Modulo-4-Binärzähler oder als Frequenzteiler aufgefaßt werden, der aus einer Frequenz f_c eine Frequenz $f_{z0} = f_c/2$ und eine Frequenz $f_{z1} = f_c/4$ erzeugt. Bei in dieser Weise kaskadierten n T-FFs kann am letzten (n-ten) Flipflop ein symmetrisches Signal der Frequenz $f/2^n$ abgenommen werden. In Digitaluhren teilt man so etwa eine Quarzfrequenz von $2^{16}\ s^{-1} = 65536\ s^{-1}$ mit 16 Flipflops auf die Frequenz $1\ s^{-1}$ herunter, mit der die (mechanische oder elektronische) Zeitanzeige angesteuert werden kann.

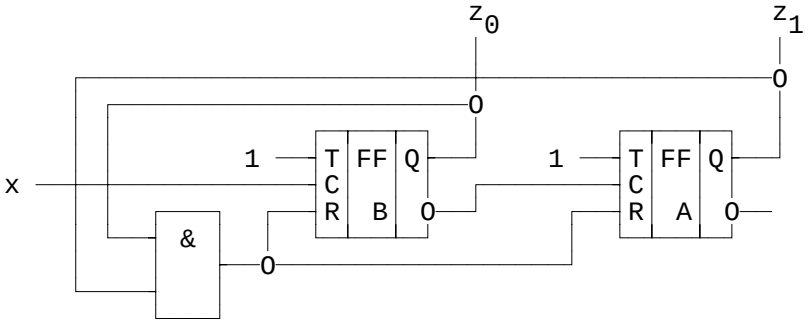
Die konsequente Weiterführung dieser Überlegungen führt zum asynchronen Automaten.

Def.: Ein Automat heißt asynchron, wenn sich bei einem Zustandswechsel die Flipflops - wenn überhaupt - nicht zwingend gleichzeitig ändern.

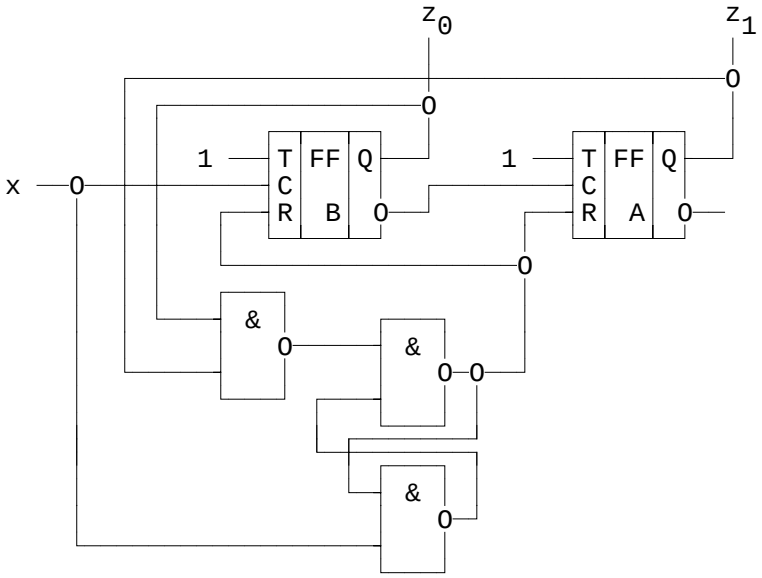
Anm.: Das ist immer dann gegeben, wenn mindestens eines der Flipflops des Automaten durch ein anderes Flipflop des gleichen Automaten getriggert wird und folglich das Triggerereignis "später" bekommt.

Ausgehend von der oben gezeigten T-Flipflop-Kaskade lassen sich auch Modulo-m-Binärzähler bzw. Frequenzteiler realisieren, die aus einer Frequenz f eine Frequenz f/m ableiten, wobei in beiden Fällen $m \neq 2^n$ gelten kann.

Für m = 3 ergibt sich z. B. die nachfolgende Schaltung. Das zusätzliche 2er-AND-Gatter generiert ein statisches Rücksetzsignal, wenn der Zähler den Zustand $z_1z_0 = 11$ erreicht. Der Zähler geht also vom Zustand $z_1z_0 = 10$ in den Zustand $z_1z_0 = 11$, verläßt ihn aber sofort wieder und geht in den Zustand $z_1z_0 = 00$.



Diese Schaltung ist nicht unbedingt zu empfehlen, da der Rücksetzimpuls unter Umständen so kurz wird, daß er den Zähler nicht sicher in den Zustand $z_1z_0 = 00$ überführen kann. In diesem Fall "hilft" ein statisches RS-Flipflop, das den Rücksetzimpuls definiert verlängert.

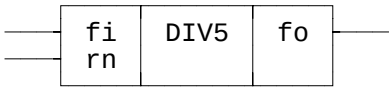


Die beiden zuletzt gezeigten Lösungen erwecken den Eindruck von "Bastellösungen". Ein seriöseres ENtwurfsverfahren für asynchrone Automaten sehen wir uns an einem Beispiel an, das Ihnen in der Übung und später im Hardwarepraktikum (Versuch Sequentielle Systeme 3) wiederbegegnen wird.

Beispiel 5.16.

Zu entwerfen sei ein Frequenzteiler 5:1 aus handelsüblich JK-MS-Flipflops SN7472.

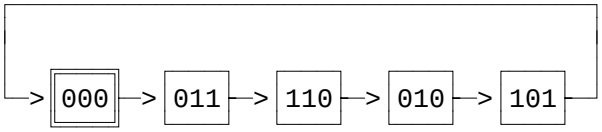
Schaltsymbol



Automatentabelle

Q^t 210	Q^{t+1} 210
000	011
011	110
110	010
010	101
101	000

Automatengraph

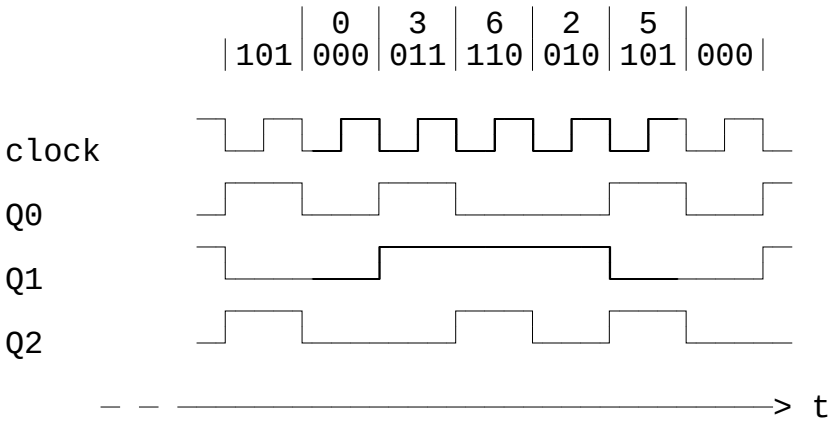


Das Syntheseverfahren hat zum Ziel, möglichst viele FFs nicht durch den von außen zugeführten Takt sondern durch ein anderes FF zu takten. Dazu ist für jedes FF A zu prüfen, ob es ein FF B gibt, das als Taktquelle für das FF A geeignet ist.

Regel 1: Ein FF B ist dann als Taktquelle für ein FF A geeignet, wenn das FF B an seinem Ausgang wenigstens immer dann die gleiche Flanke ("aktive Taktflanke", s. u.) produziert, wenn das FF A schaltet.

Regel 2: Falls mehrere FFs FF B als Taktquelle für ein FF A geeignet sind, ist dasjenige zu wählen, das die geringste Anzahl aktiver Taktflanken produziert (???).

Der Sachverhalt wird deutlich, wenn wir aus Automatentabelle oder Automatengraph das Impulsdiagramm ableiten.



Die Analyse des Schaltverhaltens der FFs ergibt:

Q –> Q'	clock	Q2	Q1	Q0
000 –> 011	F		**R**	**R**
011 –> 110	F	R		F
110 –> 010	F	F		
010 –> 101	F	R	**F**	**R**
101 –> 000	F	F		F

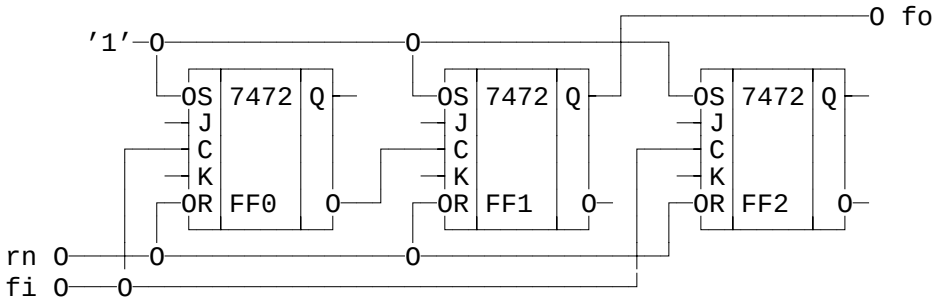
Jedes der FFs schaltet (**R**(ising edge), **F**(alling edge)) - wenn überhaupt - dann, wenn der von außen zugeführte Takt clock, eine fallende Flanke (**F**) aufweist. Das ist der Fall des synchronen Automaten.

FF 1 schaltet (**R**, **F**) - wenn überhaupt - aber auch dann, wenn FF 0 eine steigende Flanke (**R**) produziert. Nach Regel 1 ist folglich FF 0 als Taktquelle für FF 1 geeignet. Die steigende Flanke (**R**) ist die **aktive Taktflanke** der Taktquelle.

Anm.: Hier schaltet FF 1 **bei jeder** von FF 0 produzierten steigenden Flanke. Regel 1 läßt aber zu, daß FF 1 **nicht bei jeder** von FF 0 produzierten steigenden Flanke zwingend schaltet!

Da hier jedoch die FFs mit der fallenden Taktflanke schalten, muß FF 1 mit dem negierten Ausgang von FF 0 getaktet werden. Das Ergebnis dieser Überlegungen ist die noch unvollständige Be-

schaltung der FFs (Takt- und asynchrone Setz- und Rücksetzeingänge).



Für die vollständige Beschaltung der FFs fehlt noch die Beschaltung der J- und K-Eingänge. Bei allen mit dem von außen zugeführten Takt getakteten FFs (den synchronen FFs FF 0 und FF 2) wird die Beschaltung der J- und K-Eingänge so ermittelt, wie es vom Entwurf synchroner Automaten bekannt ist.

Q Q Q 2 1 0	Q'Q'Q' 2 1 0	**J K** **2 2**	J K 1 1	**J K** **0 0**
0 0 0	0 1 1	**0 -**	1 -	**1 -**
0 1 1	1 1 0	**1 -**	- 0	**- 1**
1 1 0	0 1 0	**- 1**	- 0	**0 -**
0 1 0	1 0 1	**1 -**	- 1	**1 -**
1 0 1	0 0 0	**- 1**	0 -	**- 1**

J2	0011 0110	1 0
0	0-**11**	
1	**----**	
2		

J2 = Q1

K2	0011 0110	1 0
0	**----**	
1	**-1-1**	
2		

K2 = '1'

J0	0011 0110	1 0
0	**1--1**	
1	---0	
2		

J0 = /Q2

K0	0011 0110	1 0
0	**--1-**	
1	**-1--**	
2		

K0 = '1'

Bild 7. Beschaltung der J- und K-Eingänge der synchronen FFs

Um die Beschaltung der J- und K-Eingänge des asynchronen FFs FF1 ermitteln zu können, ist die vollständige Automatentabelle um die Taktquelle FF0 zu reduzieren:

1. Streichen der Spalten Q0 und Q0'
2. Streichen aller Zeilen außer der Zeilen, die der sog. aktiven Taktflanke der Taktquelle entsprechen (hier: R(ising edge), d. h. Q0 = '0' und Q0' = '1')

Für das Beispiel ergibt sich

Streichen **Streichen**

Q Q **Q**	Q'Q'**Q'**	J K	J K	J K	
2 1 **0**	2 1 **0**	2 2	1 1	0 0	
0 0 **0**	0 1 **1**	0 -	1 -	1 -	
0 1 1	**1 1 0**	**1 -**	**- 0**	**- 1**	**<— Streichen**
1 1 0	**0 1 0**	**- 1**	**- 0**	**0 -**	**<— Streichen**
0 1 **0**	1 0 **1**	1 -	- 1	1 -	
1 0 1	**0 0 0**	**- 1**	**0 -**	**- 1**	**<— Streichen**

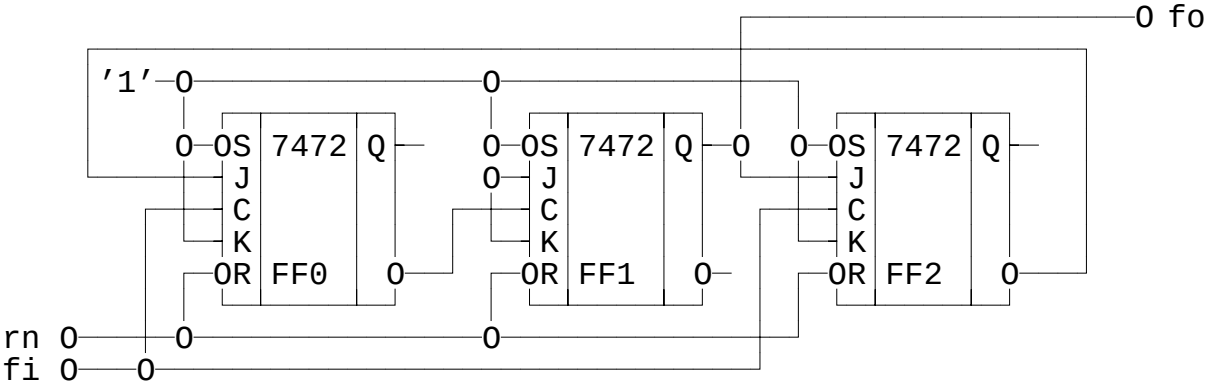
Q Q	Q'Q'	J K	**J K**	J K
2 1	2 1	2 2	**1 1**	0 0
0 0	0 1	0 -	**1 -**	1 -
0 1	1 0	1 -	**- 1**	1 -

Bild 8. Reduzieren der Automatentabelle

Die weitere Behandlung erfolgt, wie bekannt. Hier kann direkt aus der Automatentabelle abgelesen werden:

J1 = K1 = '1'.

Die vollständige Beschaltung der FFs ergibt sich zu



6. Ausgewählte Funktionsblöcke von Digitalrechnern

6.1. Verbindungseinrichtungen

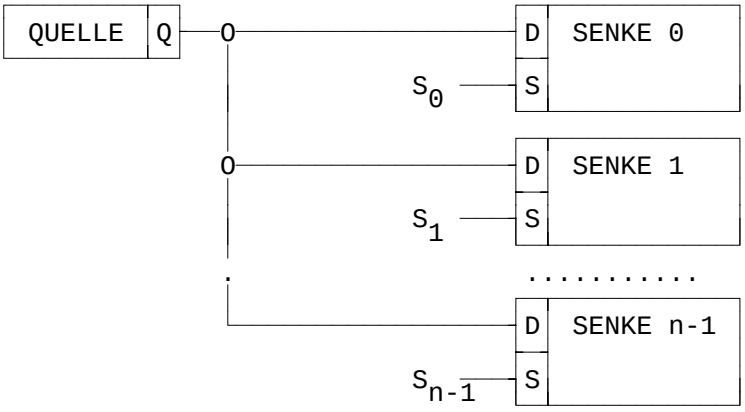
Unter Verbindungs- oder Koppeleinrichtungen wollen wir Einrichtungen verstehen, die Funktionsblöcke untereinander verbinden. Das können selbst wieder Funktionsblöcke oder auch "nur" Verdrahtungsvarianten zwischen Funktionsblöcken sein. Falls nicht anderes gesagt ist, sehen wir uns die Verhältnisse am Beispiel von TTL-Schaltkreisen an.

6.1.1. 1:1-Verbindung

Der Datenausgang Q einer Quelle ist direkt (d. h. galvanisch) mit dem Dateneingang D einer Senke verbunden. Die Senke kann häufig über ein Steuersignal S (Enable, Valid, Strobe, Select, ...) entscheiden, ob sie die anliegenden Daten übernimmt oder nicht.

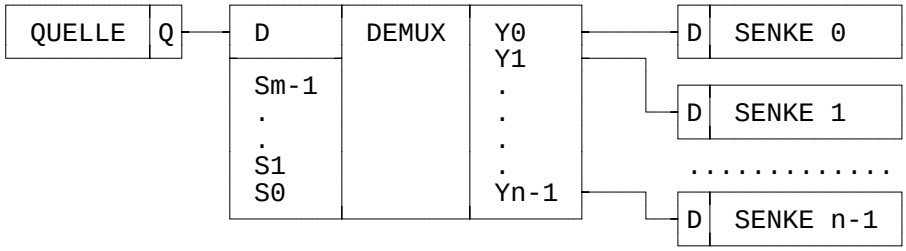
6.1.2. 1:n-Verbindung

Der Datenausgang Q einer Quelle ist direkt (d. h. galvanisch) mit den Dateneingängen D aller n Senken verbunden. Jede Senke kann in der Regel über ein Steuersignal S entscheiden, ob sie die anliegenden Daten übernimmt oder nicht.

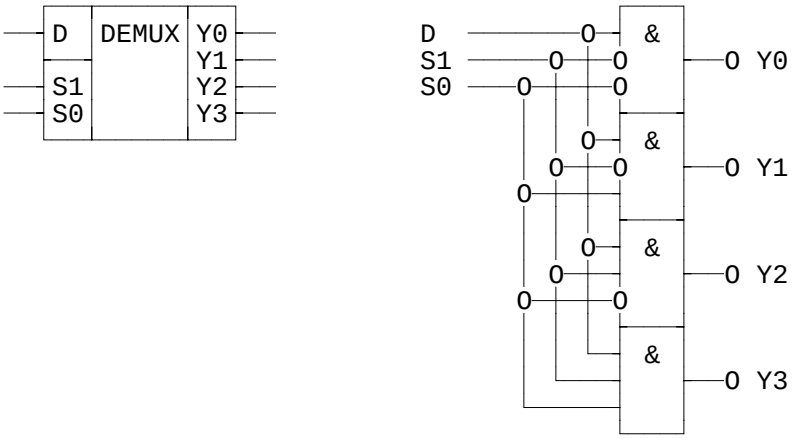


Anm.: Der Maximalwert von n ist durch die elektrischen Eigenschaften der in einer konkreten technischen Realisierung eingesetzten Schaltkreisbaureihe festgelegt. Bei TTL-Schaltkreisen der Standardbaureihe beträgt etwa der sog. Ausgangslastfaktor (fan-out) 10, d. h. an einen TTL-Ausgang können maximal 10 TTL-Eingänge angeschlossen werden. Was kann man tun, wenn mehr als 10 TTL-Eingänge mit einem TTL-Ausgang verbunden werden sollen?

Eine weitere Möglichkeit ist der sog. Demultiplexer - eine Funktionseinheit, die nach Maßgabe einer im Binärkode vorliegenden "Adresse" S den Ausgang der Quelle mit dem Eingang der durch die "Adresse" ausgewählten Senke verbindet.



Beispiel 1-zu-4-Demultiplexer



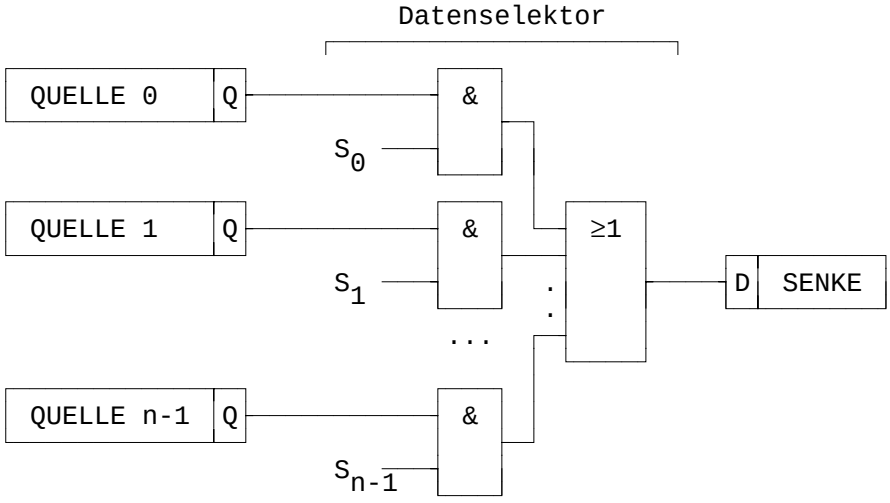
6.1.3. n:1-Verbindung

Die Datenausgänge Q von n Quellen speisen den Dateneingang D einer einzigen Senke so, daß in einem Zeitpunkt nur genau eine Quelle relevant ist. Da eine Senke in der Regel die Auswahl der relevanten Quelle nicht selbst vornehmen kann, ist hier ein Funktionsblock Datenselektor erforderlich.

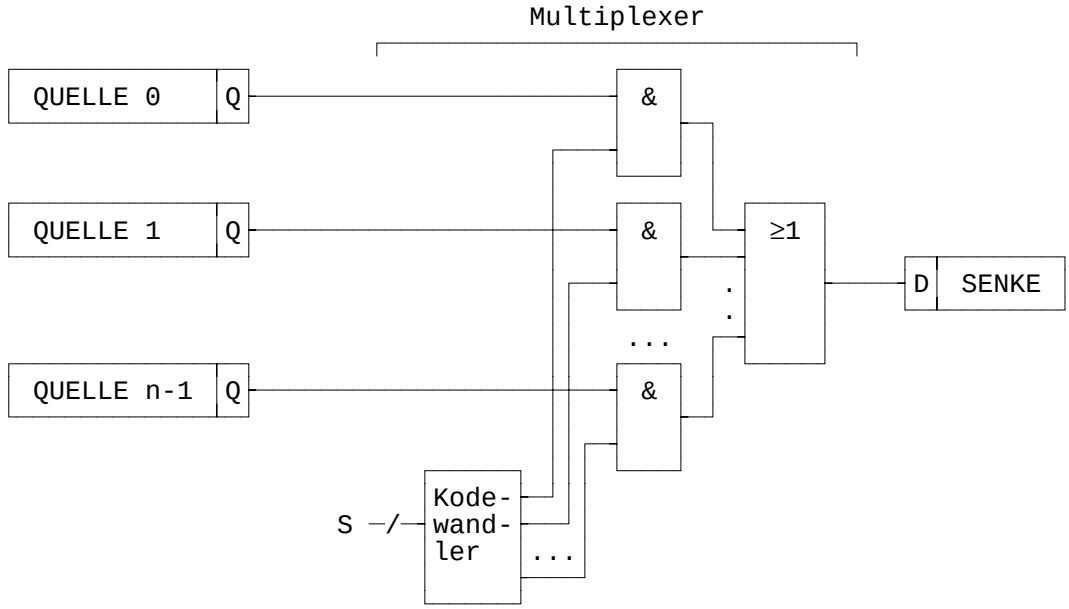
Die nachfolgende Schaltung enthält als Datenselektor ein sog. "gesteuertes ODER" mit

$D = S_0Q_0 \vee S_1Q_1 \vee ... \vee S_{n-1}Q_{n-1}$,

das die Funktion des Datenselektors nur dann korrekt erfüllt, wenn in einem Zeitpunkt nur genau ein S_i (i = 0,1,...,n-1) den Wert 1 annimmt (und dann die "relevante" Quelle Q_i mit der Senke verbindet), d. h. S muß im 1-aus-n-Kode anliegen.



Einen anderen Datenselektor haben wir bereits kennengelernt, den Multiplexer (s. o.). Im Unterschied zum "gesteuerten ODER" wird beim Multiplexer S im Binärkode angelegt und intern in den 1-aus-n-Kode gewandelt.



Anm.: Welcher Datenselektor zum Einsatz kommt, hängt davon ab, in welchem Kode S vorliegt. Liegt S bereits binär kodiert vor, wird man zum Multiplexer greifen. Sind die S_i mehr oder weniger unabhängig voneinander, wird man das "gesteuerte ODER" wählen und auf geeignete Weise sichern, daß S im 1-aus-n-Kode bereitgestellt wird, z. B. mit Hilfe einer Prioritätslogik. Wird S durch ein "Steuerwerk" (einen endlichen Automaten) bereitgestellt, "programmiert" man das Steuerwerk oft so, daß S mit Sicherheit im 1-aus-n-Kode vorliegt, und verwendet dann ebenfalls ein "gesteuertes ODER" als Datenselektor.

Beispiel: Prioritätslogik

Es liegen drei voneinander unabhängige Steuerspannungen S_0, S_1 und S_2 vor. S_0 habe Priorität vor S_1 und S_1 vor S_2.

S_0	S_1	S_2	S'_0	S'_1	S'_2
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

$S'_0 = S_0$

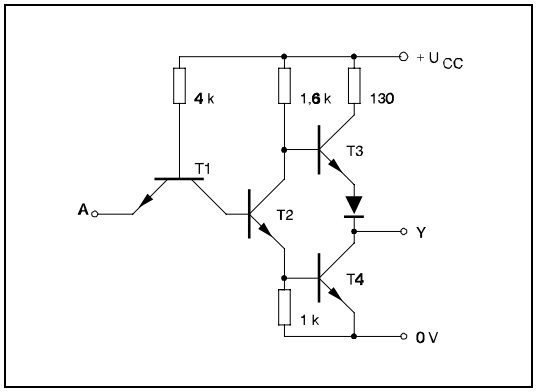
$S'_1 = \overline{S_0} \wedge S_1$

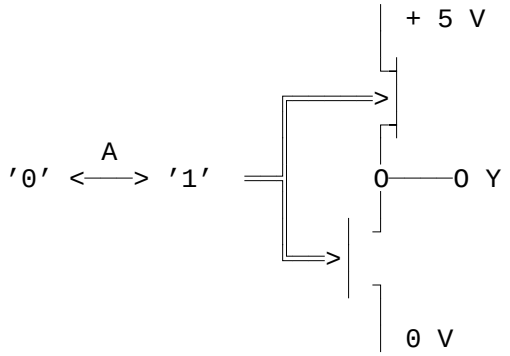
$S'_2 = \overline{S_0} \wedge \overline{S_1} \wedge S_2$

Völlig befriedigen können die bis jetzt gezeigten Lösungen kaum, da der Aufwand für die Datenselektion nicht unerheblich ist.

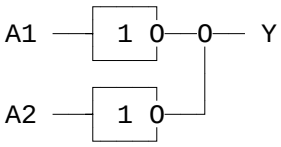
Eine Lösung, bei der man alle Quellen direkt (d. h. galvanisch) mit der Senke verbindet, scheint zunächst technisch nicht realisierbar. Wir wollen uns die elektrischen Verhältnisse an einem stark vereinfachten Modell des Ausgangs eines TTL-Standardgatters klar machen und

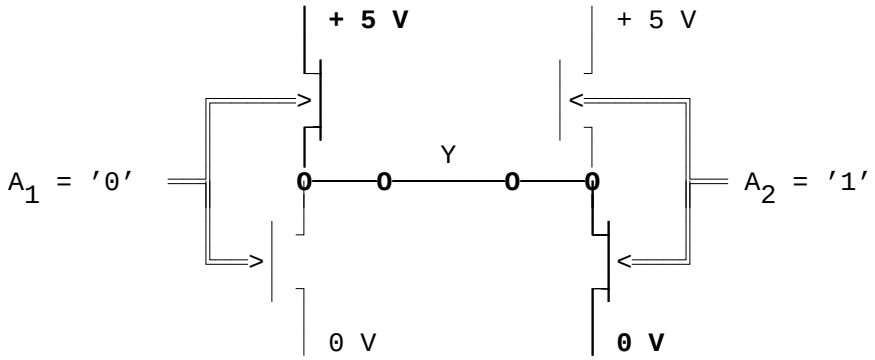
betrachten zunächst einen Negator mit Totem-Pole-Ausgang. Genauer wird die Problematik im Hardwarepraktikum beleuchtet.





Schaltet man zwei solche Negatoren ausgangsseitig parallel und belegt die Eingänge unterschiedlich, entsteht ein Kurzschluß, der zur Zerstörung wenigstens eines der beiden Negatoren führen kann. Man nennt diesen Zustand "Konflikt". Der Wert des Ausgangs Y ist nicht vorhersehbar.

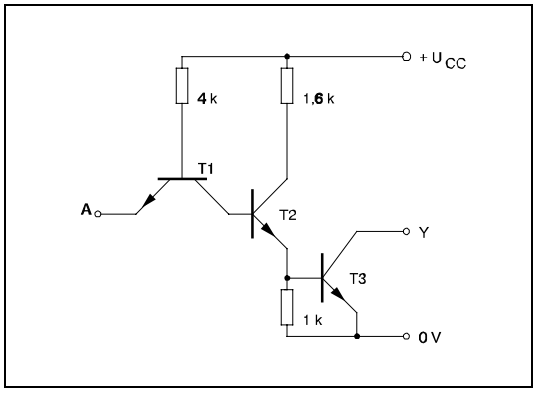


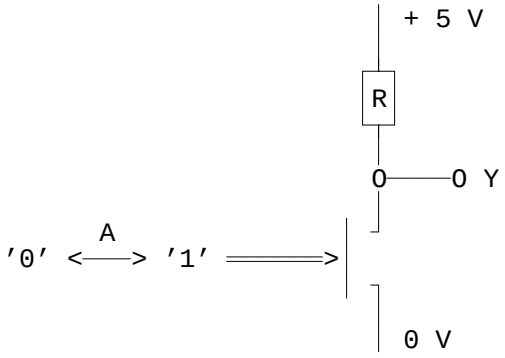


Die vollständige Wahrheitstabelle sieht so aus:

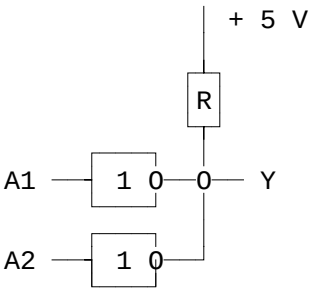
A1	A2	Y
0	0	1
0	1	X (Konflikt)
1	0	X (Konflikt)
1	1	0

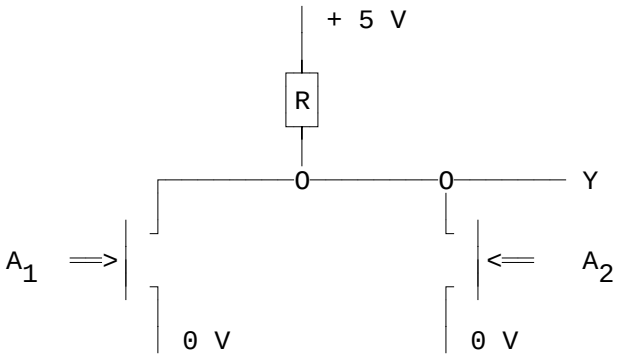
In TTL gibt es neben Gattern mit Totem-Pole-Ausgang aber auch Gatter mit Open-Collector-Ausgang, bei denen nur der untere der beiden "Schalter" existiert und der Ausgang über einen ohmschen Widerstand (Pull-up-Widerstand) mit der Spannungsquelle + 5 V verbunden wird.





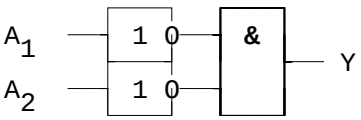
Schaltet man zwei solche Negatoren ausgangsseitig parallel, kann kein Kurzschluß auftreten. Sobald wenigstens einer der beiden Eingänge den Wert 1 führt, führt Y den Wert 0.



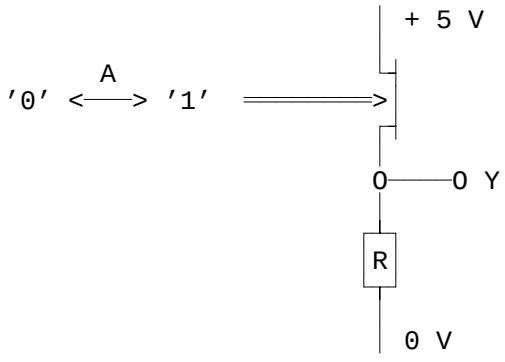


$\overline{y} = a_1 \vee a_2 \quad \longrightarrow \quad y = \overline{a_1 \vee a_2} = \overline{a_1} \wedge \overline{a_2}$

Die Parallelschaltung der beiden Negatorausgänge hat eine logische Funktion! Man nennt sie "wired AND".

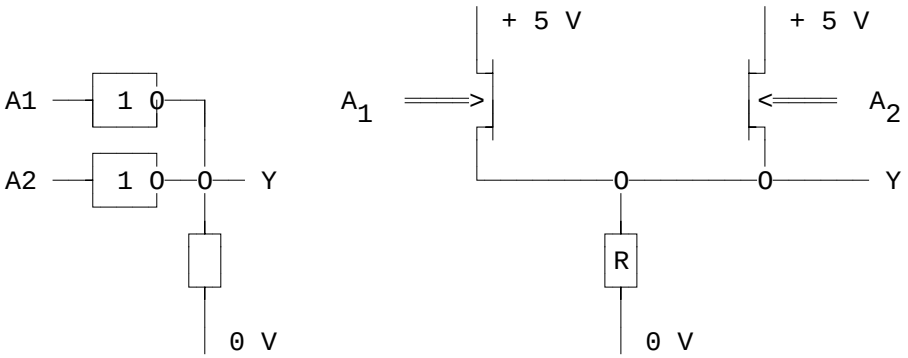


In ähnlicher Weise lassen sich Gatter mit Open-Emitter-Ausgang konstruieren (aber nicht bei TTL!), bei denen nur der obere der beiden "Schalter" existiert und der Ausgang über einen ohmschen Widerstand (Pull-down-Widerstand) mit 0 V verbunden wird.



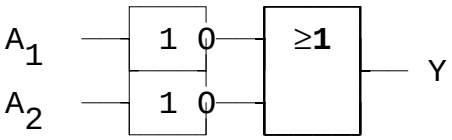
Schaltet man zwei solche Negatoren ausgangsseitig parallel, kann

ebenfalls kein Kurzschluß auftreten. Sobald wenigstens einer der beiden Eingänge den Wert 0 führt, führt Y den Wert 1:



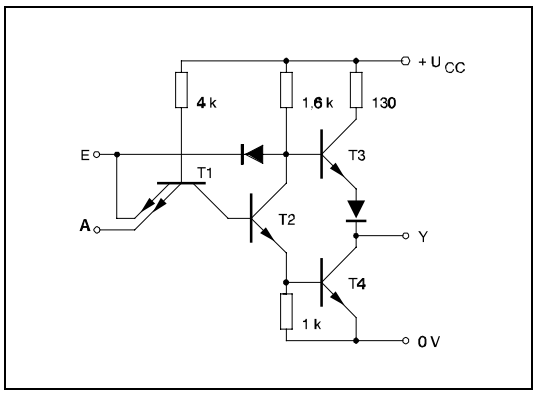
$y = \overline{a_1} \vee \overline{a_2}$

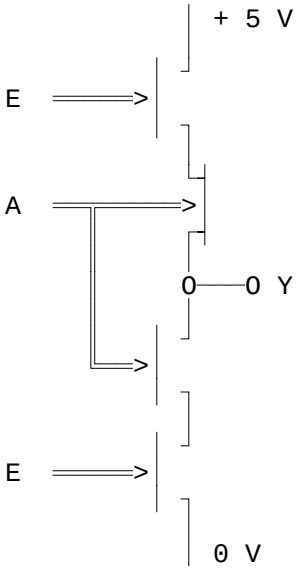
Die Parallelschaltung der beiden Negatorausgänge hat wieder eine logische Funktion! Man nennt sie "wired OR".

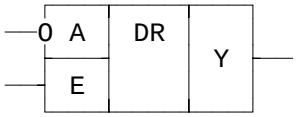


Mit Open-Collector- oder Open-Emitter-Gattern lassen sich also mehrere Quellen parallelschalten. Der resultierende Pegel hängt davon ab, welche logische Funktion die Parallelschaltung hat. Bei "wired AND" gewinnt die Null, bei "wired OR" die Eins.

Ein Nachteil der Open-Collector- und Open-Emitter-Schaltungen besteht darin, daß die erforderlichen Pull-up- bzw. Pull-down-Widerstände zusammen mit den unvermeidlichen Schaltungskapazitäten zu nicht unbeträchtlichen Umladezeiten führen und damit die Datenübertragungsrate deutlich verringern. Der Ausweg sind Gatter mit Tri-state-Ausgang. Gatter mit Tri-state-Ausgang arbeiten wie Gatter mit Totem-Pole-Ausgang, lassen sich jedoch über einen zusätzlichen Enable-Eingang E auch vollständig vom Ausgang abtrennen ("deaktivieren", "hochohmig machen").

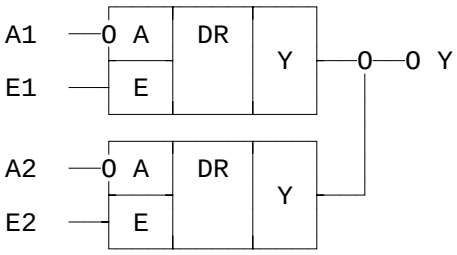






E A	Y
0 0	Z (hochohmig)
0 1	Z (hochohmig)
1 0	1
1 1	0

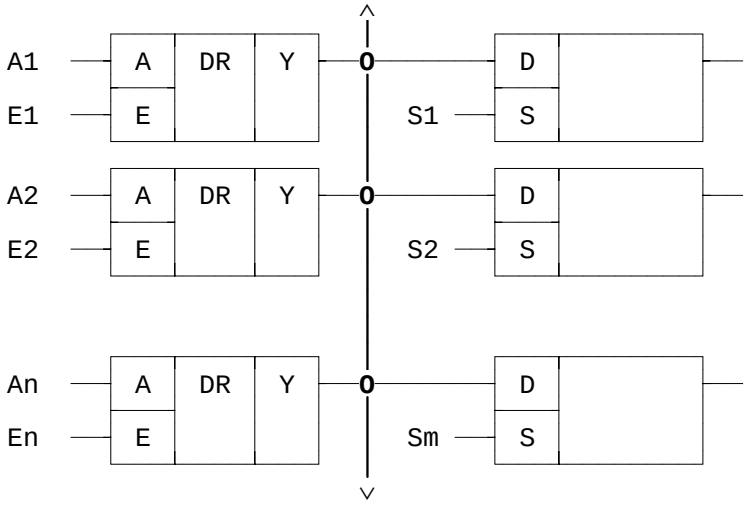
Schaltet man zwei solche Negatoren ausgangsseitig parallel, ergibt sich folgende Wahrheitstabelle:



E1 E2	A1 A2	Y
0 0	0 0	**Z (hochohmig)**
0 0	0 1	**Z (hochohmig)**
0 0	1 0	**Z (hochohmig)**
0 0	1 1	**Z (hochohmig)**
0 1	0 **0**	**1**
0 1	0 **1**	**0**
0 1	1 **0**	**1**
0 1	1 **1**	**0**
1 0	**0** 0	**1**
1 0	**0** 1	**1**
1 0	**1** 0	**0**
1 0	**1** 1	**0**
1 1	0 0	1
1 1	**0 1**	**X (Konflikt)**
1 1	**1 0**	**X (Konflikt)**
1 1	1 1	0

6.1.4. n:m-Verbindung

Abgesehen von speziellen Verbindungseinrichtungen, die wir hier nicht besprechen wollen, werden n:m-Verbindungen als Kombination aus n:1- und 1:m-Verbindungen realisiert. In Digitalrechnern trifft man vorzugsweise sog. Busse an. Unter einem Bus versteht man eine Leitung, die alle Quellen und Senken direkt (d. h. galvanisch) miteinander verbindet, wobei die Quellen - wie besprochen - Gatter mit Open-collector-, Open-emitter- oder Tri-state-Ausgang sein können und die Senken die auf dem Bus liegenden Daten übernehmen können oder nicht.



Rechnerintern trifft man in erster Linie Tri-state-Busse an. Ein Hauptproblem besteht darin, Bus-Konflikte sicher zu verhindern. Die dazu verwendeten Verfahren würden den Rahmen dieser Lehrveranstaltung sprengen.

6.1.5. Anmerkung zur Modellierung von Bussen in VHDL

Entwurfsbeschreibungssprachen lassen zunächst nur reguläre Leitungsnetze zu. Regulär ist ein Leitungsnetz dann, wenn es genau eine Quelle und beliebig viele Senken verbindet. Bei Leitungsnetzen mit mehr als einer Quelle muß ein Mechanismus existieren, der aus den Pegeln, die die einzelnen Quellen schreiben, den resultierenden Pegel, den die Senken lesen können, ermittelt.

In VHDL heißt dieser Mechanismus "Resolution". Ordnet man einem Leitungsnetz den Datentyp **std_logic** zu, dann dürfen Leitungsnetze mit mehr als einer Quelle beschrieben werden, und bei jeder Änderung des Ausgangspegels einer dieser Quellen wird implizit der daraus folgende, ggf. geänderte resultierende Pegel des gesamten Leitungsnetzes berechnet.

Der Datentyp **std_logic** beschreibt eine 9-wertige Logik,

('U','X','0','1','Z','W','L','H','-')

was zunächst verwirren muß, da letztendlich doch binäre, also 2-wertige Systeme modelliert werden sollen. Die Motivation für die 9-wertige Logik ist vielgestaltig.

Bei der Behandlung der Flipflops haben wir bereits kennengelernt, daß Flipflops nach dem Zuschalten der Stromversorgung und vor dem ersten Setzen bzw. Rücksetzen nichtvorhersehbar entweder auf '0' oder auf '1' stehen - sie sind (noch) nicht initialisiert. Dieser Fall heißt im Datentyp **std_logic** 'uninitialized' und wird mit 'U' bezeichnet. Beim RS-Flipflop haben wir außerdem erfahren, daß ein Flipflop - obwohl es bereits initialisiert worden ist - durch "Fehlbedienung" in einen Zustand gebracht werden kann, der nichtvorsehbar '0' oder '1' ist. Dieser Fall heißt "forcing 0 or 1" und wird mit 'X' bezeichnet.

Die Pegel '0' ("forcing 0"), '1' ("forcing 1)" und 'X' ("forcing 0 or 1") bilden eine 3-wertige Logik, die wesentlich ist für das Grundverständnis der Logiksimulation. Mit 'X' = '0'|'1' ergibt sich für

die Wahrheitstabellen der Negation, der Konjunktion und der Disjunktion

Negation	Konjunktion	Disjunktion
$y = \overline{x}$	$y = x_1 \wedge x_2$	$y = x_1 \vee x_2$

x	y
0	1
1	0
X	X

x1	x2	y
0	0	0
0	1	0
0	X	0
1	0	0
1	1	1
1	X	X
X	0	0
X	1	X
X	X	X

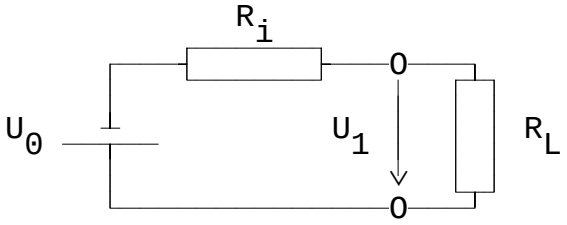
x1	x2	y
0	0	0
0	1	1
0	X	X
1	0	1
1	1	1
1	X	1
X	0	X
X	1	1
X	X	X

Besonders beachtenswert ist die eingeschränkte Gültigkeit der Komplementregeln:

x	$x \wedge \overline{x}$	$x \vee \overline{x}$	
0	0	1	
1	0	1	
X	**X**	**X**	**!!!**

Man spricht deshalb bei der 3-wertigen Simulation auch von "pessimistischer" Simulation.

Wenn mehrere Quellen ausgangsseitig parallelgeschaltet sind und - was der Regelfall ist - unterschiedliche Pegel liefern können, fehlt uns bisher ein allgemeingültiges Kriterium, nach dem entscheidbar wäre, welche Quelle "gewinnt". Wir haben aber bereits kennengelernt, daß sich bei Open-collector-Gattern der Pegel ’0’ gegenüber dem Pegel ’1’ und bei Open-emitter-Gattern der Pegel ’1’ gegenüber dem Pegel ’0’ durchsetzt. Bei Totem-Pole-Gattern hingegen ist nicht entschieden, welcher Pegel sich durchsetzt. Es gibt offensichtlich qualitative Unterschiede der Pegel in Abhängigkeit von der Art des Gatterausgangs. Eine Betrachtung am Grundstromkreis liefert den Schlüssel.



Eine Spannungsquelle mit der Leerlaufspannung U_0 und dem Innenwiderstand R_i werde durch einen Lastwiderstand R_L belastet. Die Spannungsteilerregel liefert:

$$\frac{U_1}{U_0} = \frac{R_L}{R_i + R_L} = \frac{1}{1 + \frac{R_i}{R_L}}$$

Für $R_i << R_L$ ergibt sich $U_1 \approx U_0$, für $R_i = R_L$ folgt $U_1 = U_0/2$ und für $R_i >> R_L$ gilt $U_1 \approx 0$. Die Ausgangsspannung U_1 "bricht" bei gleicher Belastung R_L umso mehr zusammen, je größer der Innenwiderstand R_i der Spannungsquelle ist. Man spricht auch von "harten" und "weichen" Spannungen, in Abhängigkeit davon, ob der Innenwiderstand der Spannungsquelle niedrig oder hoch ist. Werden zwei Spannungsquellen zusammengeschaltet, "gewinnt" immer die "härtere" Spannung! Diesen Zusammenhang nutzt VHDL beim Datentyp **std_logic** und ordnet jedem der 9 Werte eine Signalstärke (signal strength) $s \sim 1/R_i$ zu. Es gelten die Ungleichung

$$s('U') > s('X') > \begin{matrix} s('0') \\ s('1') \end{matrix} > s('W') > \begin{matrix} s('L') \\ s('H') \end{matrix} > s('Z') > s('-')$$

und mit dem Resolution-Operator RES die Regeln

'0' RES '1' = 'X'
'L' RES 'H' = 'W'.

Die Pegel '0' ("forcing 0"), '1' ("forcing 1)" und 'X' ("forcing 0 or 1") bilden die bereits besprochene 3-wertige, "harte" Logik. Die Pegel 'L' ("weak 0"), 'H' ("weak 1") und 'W' ("weak 0 or 1") bilden eine weitere, jedoch "weichere" 3-wertige Logik. Der Pegel 'Z' steht für "high impedance" und '-' für das imperative "don't care", das für die Simulation ohne Bedeutung ist.

Ein Totem-pole-Gatter liefert die Ausgangspegel '0' und '1', ein Open-collector-Gatter die Ausgangspegel '0' und 'H', ein Open-emitter-Gatter 'L' und '1' und ein Tri-state-Gatter die Ausgangspegel '0', '1' und 'Z'. Werden mehrere Gatter ausgangsseitig parallelgeschaltet und liefern sie unterschiedliche Ausgangspegel, dann gewinnt der "härteste" Pegel nach Maßgabe der o. a. Ungleichung und Regeln und bestimmt den Pegel des gesamten Leitungsnetzes.

6.2. Programmierbare kombinatorische Logik

Wir haben bereits kennengelernt, daß sich Multiplexer so beschalten ("programmieren") lassen, daß sie - in bestimmten Grenzen - beliebige BOOLEsche Funktionen realisieren. Von dieser Möglichkeit macht man aber eher selten Gebrauch. Es ist eine Vielzahl spezieller Schaltkreise entwickelt worden, von denen wir zwei betrachten wollen: **ROM** (**R**ead **O**nly **M**emory) und **PLA** (**P**rogrammable **L**ogic **A**rray).

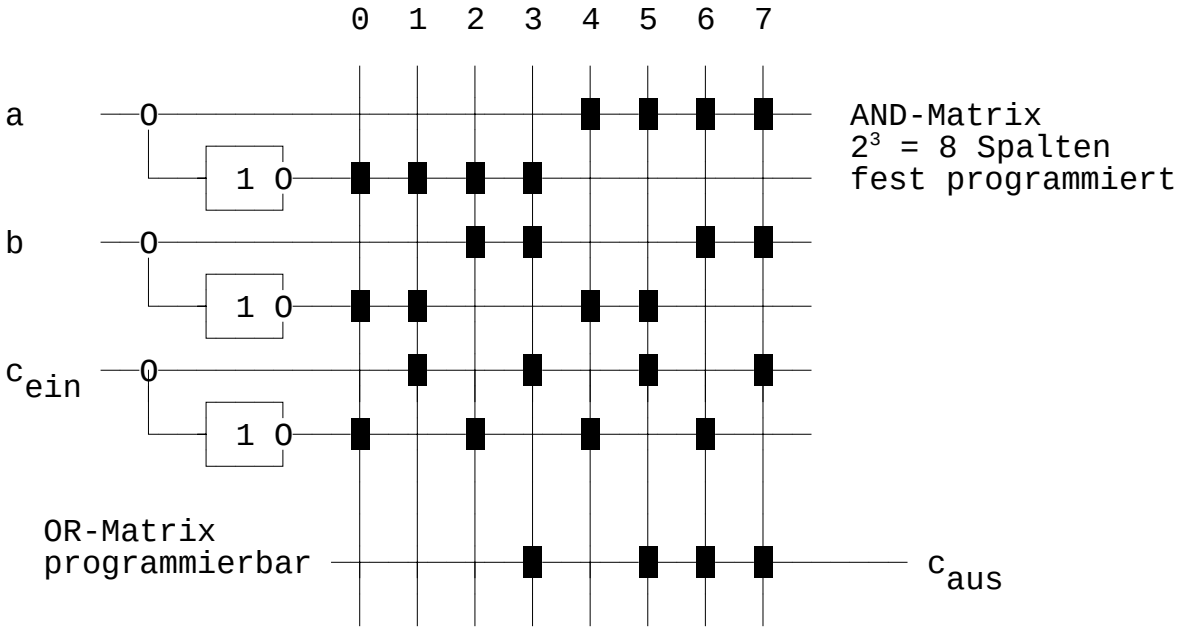
ROM und PLA sind prinzipiell gleich aufgebaut. Wir machen uns die Wirkungsweise an einem Beispiel klar, dem Volladder (s. 2.4.1.), und beschränken uns zunächst auf c_{aus}.

a b c_{ein}	c_{aus} s
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

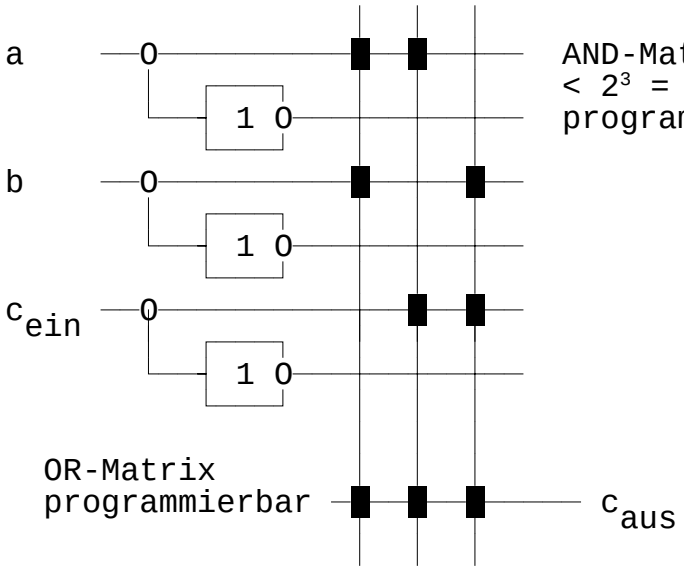
$$c_{aus} = \underbrace{\bar{a}\ b\ c_{ein}}_{3} \vee \underbrace{a\ \bar{b}\ c_{ein}}_{5} \vee \underbrace{a\ b\ \overline{c_{ein}}}_{6} \vee \underbrace{a\ b\ c_{ein}}_{7}$$

$$= a\ b \vee a\ c_{ein} \vee b\ c_{ein}$$

ROM-Realisierung (8x1-ROM)



PLA-Realisierung



Wir erkennen: Ausgangspunkt für eine **ROM**-Realisierung einer BOOLEschen Funktion ist deren **KDNF**, Ausgangspunkt für eine **PLA**-Realisierung einer BOOLEschen Funktion ist deren **minimierte DNF**.

Die OR-Matrizen realer ROMs und PLAs besitzen natürlich mehr als eine Zeile. Damit sind ROMs und PLAs grundsätzlich geeignet, beliebige BOOLEsche Funktionsbündel zu realisieren.

Die Spaltenzahl der AND-Matrix einer PLA mit n Eingängen ist stets kleiner als 2^n. Um diesen Nachteil wettzumachen, legt man die AND-Matrix einer PLA - wie die OR-Matrix bei ROMs und PLAs -ebenfalls programmierbar aus.

Die Minimierung eines Funktionsbündels muß das Ziel haben, mit so wenig wie möglich Spalten (d. h. Implikanten) auszukommen. Es sind also solche Implikanten zu suchen, die in möglichst vielen BOOLEschen Funktionen des zu realisierenden BOOLEschen Funkti-onsbündels verwendet werden können.

Die (globale) Bündeloptimierung läßt sich in der Regel nicht durch die (lokale) Minimierung der einzelnen BOOLEschen Funktionen des Bündels erreichen. In den Folien Dr. Königs ist die Bündeloptimierung umfassend erläutert.

6.3. Adder

Die Arbeitsgeschwindigkeit (≈ max. Taktfrequenz) eines Digitalrechners wird u. a. durch die Arbeitsgeschwindigkeit der verwendeten ALU (engl. arithmetic logic unit) bestimmt. Ein charakteristisches Maß für die Arbeitsgeschwindigkeit einer ALU ist die Additionszeit t_{add}, die Zeit, die mindestens vergeht, um zwei n-Bit-Binärzahlen a und b korrekt zu addieren. Die n-Bit-Binärzahlen a und b werden so dargestellt:

$$a = a_{n-1}*2^{n-1} + a_{n-2}*2^{n-2} +...+ a_i*2^i +...+ a_1*2^1 + a_0*2^0 \qquad (1)$$

$$b = b_{n-1}*2^{n-1} + b_{n-2}*2^{n-2} +...+ b_i*2^i +...+ b_1*2^1 + b_0*2^0 \qquad (2)$$

Für die Summe s folgt:

$$s = s_{n-1}*2^{n-1} + s_{n-2}*2^{n-2} +...+ s_i*2^i +...+ s_1*2^1 + s_0*2^0 \qquad (3)$$

Eine technische Realisierung eines binären Paralleladders ist der Ripple-Carry-Adder (engl. to ripple = rieseln; der Übertrag "rieselt" durch die Schaltung):

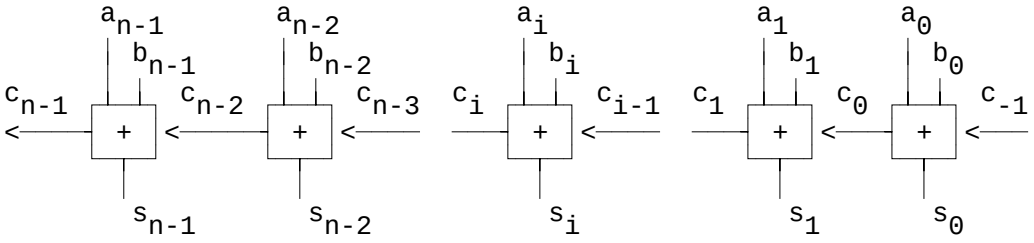
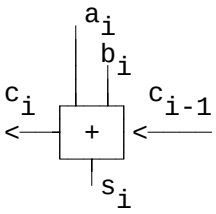


Bild 1. Ripple-Carry Adder

Die Funktion des (Voll-)Adders leitet sich aus der Wahrheitstabelle ab:



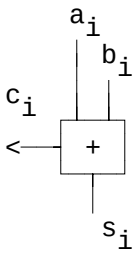
c_{i-1}	b_i	a_i	c_i	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Bild 2. Adder und Wahrheitstabelle

$$c_i = a_i \wedge b_i \vee a_i \wedge c_{i-1} \vee b_i \wedge c_{i-1} \qquad (4)$$

$$s_i = a_i \not+ b_i \not+ c_{i-1} \qquad (5)$$

Der Adder kann aus zwei Halbaddern zusammengesetzt werden, wobei ein Halbadder die Addition zweier Bits ohne einlaufenden Übertrag realisiert:



b_i	a_i	c_i	s_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Bild 3. Halbadder und Wahrheitstabelle

$$c_i = a_i \wedge b_i \tag{6}$$

$$s_i = a_i \not\vee b_i \tag{7}$$

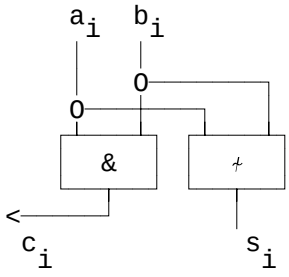


Bild 4. Struktur des Halbadders

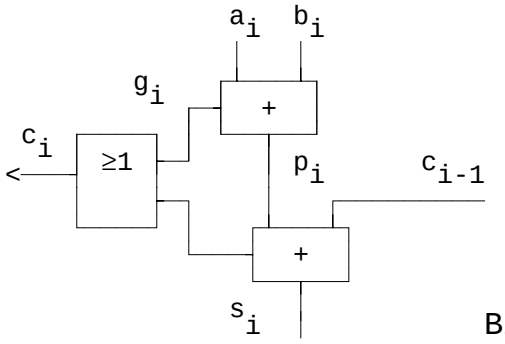


Bild 5. Struktur des Adders aus Halbaddern

$$c_i = g_i \vee p_i \wedge c_{i-1} \tag{8}$$

$$= a_i \wedge b_i \vee (a_i \wedge \overline{b}_i \vee a_i \wedge b_i \overline{)} \wedge c_{i-1}$$

$$= a_i \wedge b_i \vee a_i \wedge c_{i-1} \vee b_i \wedge c_{i-1}$$

Anm.: $g_i = a_i \wedge b_i$ ist von "generate" (engl. to generate = erzeugen) abgeleitet und gibt an, unter welchen Bedingungen in der Stufe i ein Übertrag erzeugt wird. $p_i = a_i \not\vee b_i$ ist von "propagate" (engl. to propagate = verbreiten) abgeleitet

und gibt an, unter welchen Bedingungen ein einlaufender Übertrag c_{i-1} auf c_i durchgereicht wird.

$$s_i = p_i \not+ c_{i-1} \qquad (9)$$

$$= a_i \not+ b_i \not+ c_{i-1}$$

Eine weitere, vor allem bei komplexeren ALUs gern genutzte Möglichkeit ist:

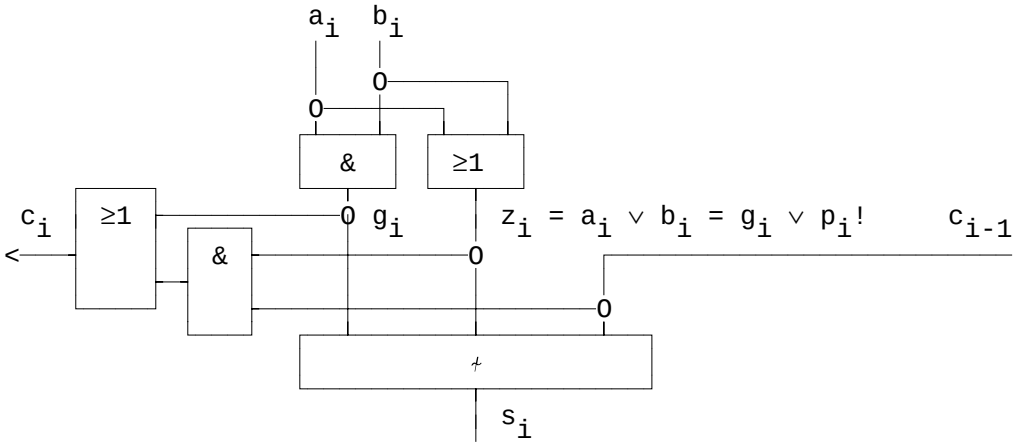


Bild 6. Struktur des Adders (weitere Möglichkeit)

$$c_i = g_i \vee z_i \wedge c_{i-1} \qquad (10)$$

$$= a_i \wedge b_i \vee (a_i \vee b_i) \wedge c_{i-1}$$

$$= a_i \wedge b_i \vee a_i \wedge c_{i-1} \vee b_i \wedge c_{i-1}$$

$$s_i = g_i \not+ z_i \not+ c_{i-1} \qquad (11)$$

$$= (a_i \wedge b_i) \not+ (a_i \vee b_i) \not+ c_{i-1}$$

$$= a_i \not+ b_i \not+ c_{i-1}$$

Für die Bestimmung der Additionszeit t_{add} des n-Bit-Ripple-Carry-Adders sollen folgende Voraussetzungen gelten:

- Die Signale a_i und b_i (i = 0,1, ... ,n-1) und das Signal c_{-1} liegen zum Zeitpunkt t = 0 an.
- Für den (Voll-)Adder gilt: Die Signale c_i und s_i liegen eine Zeiteinheit t_{vadd} nachdem das letzte der Signale a_i, b_i und c_{i-1} anliegt an. Beim n-Bit-Ripple-Carry-Adder ist das in jedem Falle das Signal c_{i-1}, so daß hier gilt: Die Signale c_i und s_i (i = 0,1,...,n-1) liegen eine Zeiteinheit t_{vadd} nachdem das Signal c_{i-1} (i = 0,1,...,n-1) anliegt an. Die Additionszeit wird also durch die Ausbreitungszeit des Übertrags bestimmt.

Damit gilt:

$$t_{add} = n * t_{vadd}. \qquad (12)$$

Es handelt sich dabei um den "schlechtesten" Fall, bei dem ein einlaufender Übertrag c_{-1} über alle Stufen des Ripple-Carry-Adders hinweg auf c_{n-1} durchgereicht wird, d. h. es muß gelten:

$$p_i = a_i \not+ b_i = 1, \text{ für alle } i = 0,1, \ldots ,n-1. \qquad (13)$$

Um die Additionszeit zu verringern, faßt man mehrere (k) benachbarte Stellen zu einer Gruppe zusammen, für die man den Übertrag parallel erzeugt und aus denen man den Ripple-Carry-Adder zusammensetzt:

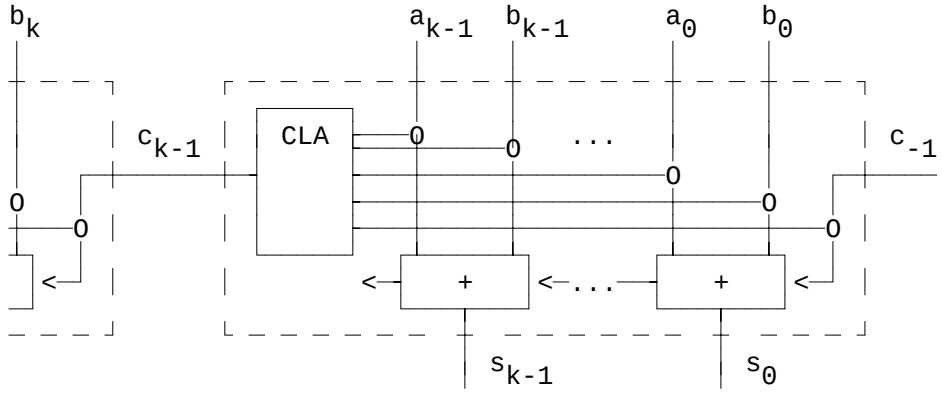


Bild 7. Gruppenbildung

Die Funktionseinheit CLA realisiert die vorausschauende (parallele) Übertragsbildung ("carry look ahead", engl. to look ahead = vorausschauen), d. h. der Übertrag c_{k-1} liegt an, bevor die Summe s_{k-1} s_{k-2}, ...,s_0 anliegt. Wenn man der Einfachheit halber annimmt, daß die Funktionseinheit CLA eine Verzögerungszeit von einer Zeiteinheit t_{vadd} aufweist, liegen c_{k-1} nach t_{vadd} und s_{k-1} nach $k*t_{vadd}$ an (ein zeitlicher Gewinn ergibt sich überhaupt nur, wenn die Funktionseinheit CLA eine Verzögerungszeit aufweist, die kleiner als $k*t_{vadd}$ ist). Für den Ripple-Carry-Adder gilt dann

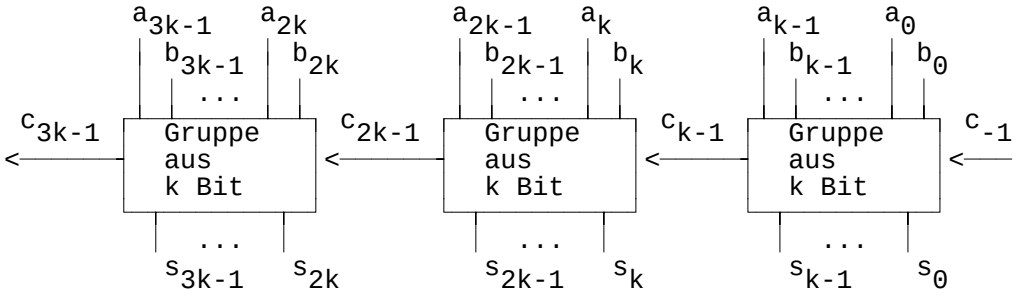


Bild 8. Ripple-Carry-Adder aus Gruppen zu k Bit

allgemein:

$$t_{add} = \left\lceil k + \frac{n}{k} - 1 \right\rceil * t_{vadd}. \qquad (14)$$

Wie groß ist k zu wählen, um die Additionszeit zu minimieren?

$$\frac{d}{dk}\left\lceil \frac{t_{add}}{t_{vadd}} \right\rceil = \frac{d}{dk}\left\lceil k + \frac{n}{k} + 1 \right\rceil = 1 - \frac{n}{k^2} = 0 \qquad (15)$$

$$k_{opt} = \sqrt{n} \qquad (16)$$

$$t_{addmin} = (2\sqrt{n} - 1) * t_{vadd} \qquad (17)$$

Für n = 24 ergibt sich z. B.:

k	1	2	3	4	(4,9)	6	8	12	24
t_{add} / t_{vadd}	24	13	10	9	(8,8)	9	10	13	24

Bild 9. $t_{add} / t_{vadd} = f(k)$, Beispiel

Achtung! Diese Aussagen gelten korrekt nur unter den getroffenen Voraussetzungen (s. o.).

Für die technische Realisierung der CLA-Logik gibt es mehrere Möglichkeiten. Bei vollständig paralleler Übertragsbildung gilt zunächst:

$$\begin{aligned} c_{k-1} = {} & a_{k-1} \wedge b_{k-1} \vee \\ & a_{k-2} \wedge b_{k-2} \wedge (a_{k-1} \not\wedge b_{k-1}) \vee \\ & | \\ & a_0 \wedge b_0 \wedge (a_{k-1} \not\wedge b_{k-1}) \wedge \ldots \wedge (a_1 \not\wedge b_1) \vee \\ & c_{-1} \wedge (a_{k-1} \not\wedge b_{k-1}) \wedge \ldots \wedge (a_1 \not\wedge b_1) \wedge (a_0 \not\wedge b_0). \end{aligned} \qquad (17)$$

Auf der Grundlage dieser Beziehung ist die "schnellste" aber auch aufwendigste CLA-Logik realisierbar. Setzt man den (Voll-)Adder aus zwei Halbaddern zusammen und führt die internen Signale g_i und p_i nach außen, dann vereinfacht sich (17) zu

$$c_{k-1} = g_{k-1} \vee \quad (18)$$

$$g_{k-2} \wedge p_{k-1} \vee$$

$$|$$

$$g_0 \wedge p_{k-1} \wedge \ldots \wedge p_1 \vee$$

$$c_{-1} \wedge p_{k-1} \wedge \ldots \wedge p_1 \wedge p_0,$$

was sehr einfach aussagenlogisch interpretierbar ist: Ein auslaufender Übertrag c_{k-1} tritt auf, wenn ein Übertrag in der Stufe k-1 entsteht oder ... oder wenn ein Übertrag in der Stufe 0 entsteht und über die Stufen 1,2, ... und n-1 durchgereicht wird oder wenn ein einlaufender Übertrag c_{-1} auftritt und über die Stufen 0,1, ... und n-1 durchgereicht wird.

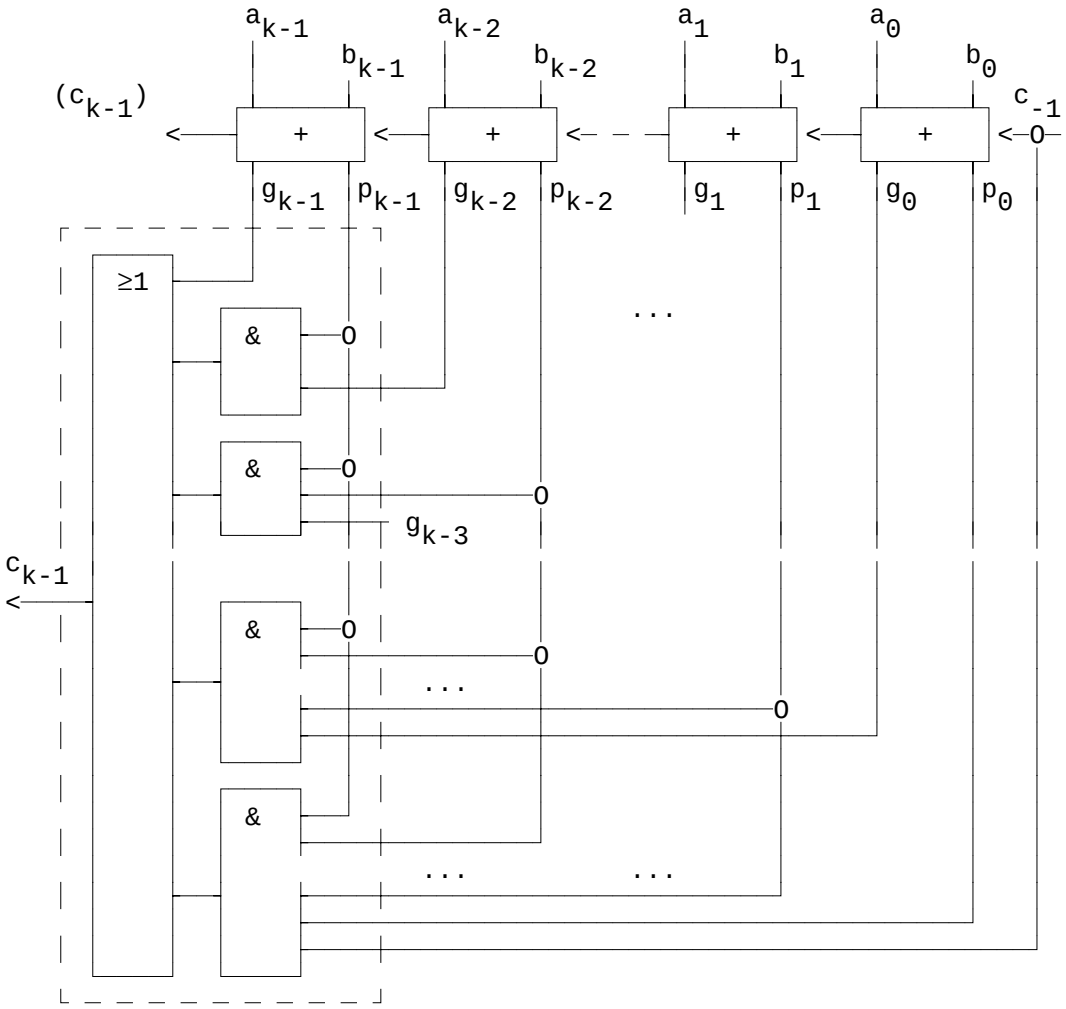


Bild 10. CLA-Logik (s-Ausgänge nicht dargestellt)

Analog dazu ergibt sich für die vorzugsweise bei komplexeren ALUs verwendete Lösung:

$$c_{k-1} = g_{k-1} \vee g_{k-2} \wedge z_{k-1} \vee \mid g_0 \wedge z_{k-1} \wedge \ldots \wedge z_1 \vee c_{-1} \wedge z_{k-1} \wedge \ldots \wedge z_1 \wedge z_0. \quad (19)$$

Bei der Realisierung sehr breiter n-Bit-Binäradder ist es sinnvoll, den Adder aus mehreren Gruppen zusammenzusetzen, die ihrerseits wieder aus Gruppen benachbarter Bits bestehen. Die bisherigen Betrachtungen erlauben aber nur den Aufbau von Ripple-Carry-Addern aus Gruppen, die aus einer Anzahl von (Voll-)Addern bestehen und für die der auslaufende Übertrag mit einer CLA-Logik vorausschauend (parallel) erzeugt wird. Für Gruppen aus Gruppen steht noch kein geeigneter Mechanismus zur Verfügung. Das soll jetzt nachgeholt werden. Um über mehrere Gruppen hinweg den auslaufenden Übertrag vorausschauend ermitteln zu können, müßten diese Gruppen zwei Signale g’ und p’ (oder g’ und z’) bereitstellen. Aus (8) und (18) folgt:

$$c_{k-1} = g'_{k-1} \vee p'_{k-1} \wedge c_{-1} \quad (20)$$

$$g'_{k-1} = g_{k-1} \vee g_{k-2} \wedge p_{k-1} \vee \ldots \vee g_0 \wedge p_{k-1} \wedge p_{k-2} \wedge \ldots \wedge p_1 \quad (21)$$

$$p'_{k-1} = p_{k-1} \wedge p_{k-2} \wedge \ldots \wedge p_1 \wedge p_0. \quad (22)$$

Aus (10) und (19) folgt analog:

$$c_{k-1} = g'_{k-1} \vee z'_{k-1} \wedge c_{-1} \quad (23)$$

$$g'_{k-1} = g_{k-1} \vee g_{k-2} \wedge z_{k-1} \vee \ldots \vee g_0 \wedge z_{k-1} \wedge z_{k-2} \wedge \ldots \wedge z_1 \quad (24)$$

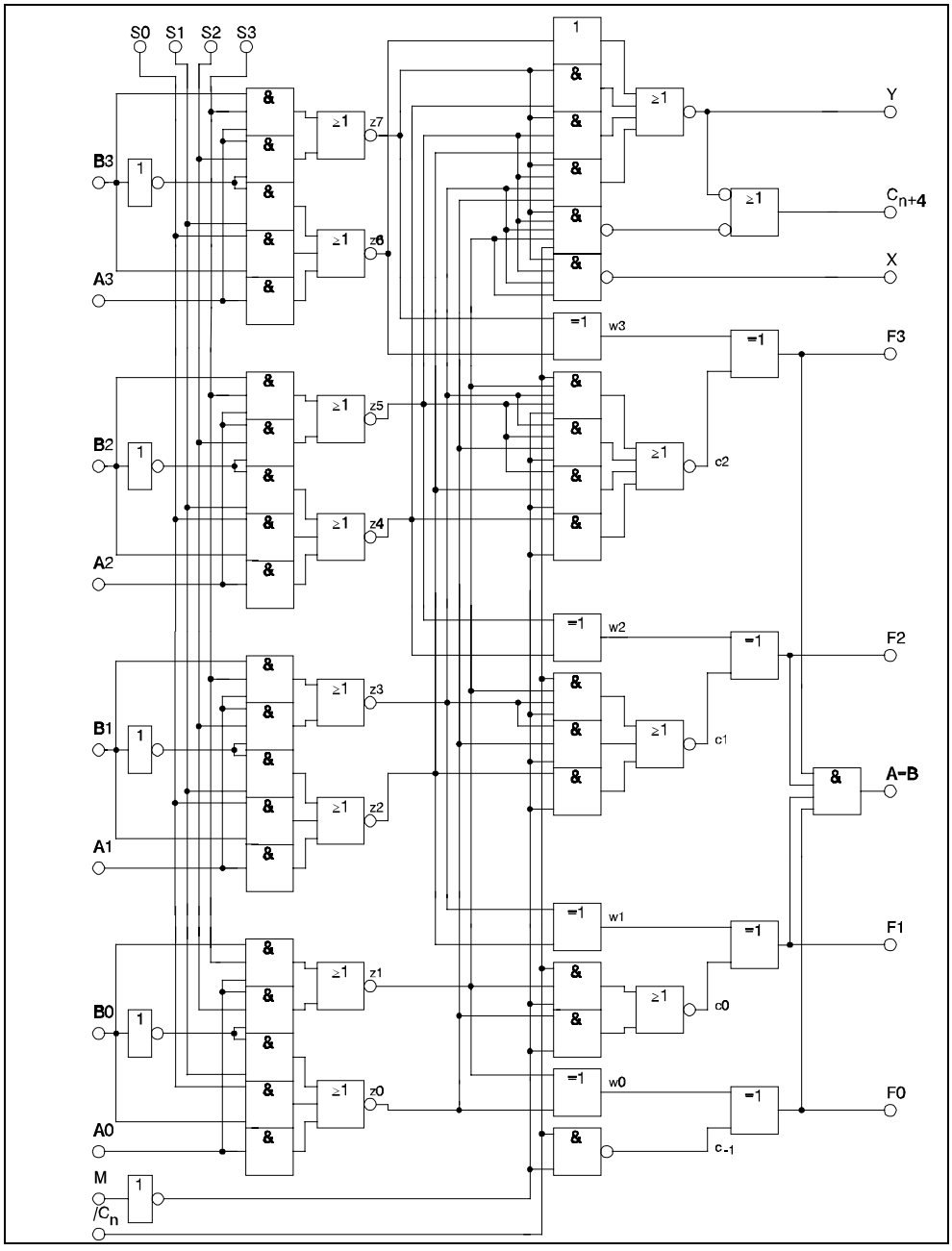
$$z'_{k-1} = z_{k-1} \wedge z_{k-2} \wedge \ldots \wedge z_1 \wedge z_0. \quad (25)$$

Die Zusammenschaltung mehrerer Gruppen erfolgt analog zu Bild 10. Es wird die gleiche CLA-Logik verwendet.

6.4. ALU

Ein Rechner vom von-Neumann-Typ (s. u. und LV Rechnerorganisation im 2. Semester)) "tut" zu einem Zeitpunkt nur "genau ein Was". Diese Eigenschaft legt nahe, eine Funktionseinheit zu entwickeln, die alle logischen, alle arithmetischen und - wenn möglich - alle sonstigen Operationen so ausführen kann, daß zu einem Zeitpunkt nur genau eine dieser Funktionen "aktiv" ist - eine sog. ALU (engl. **a**rithmetic **l**ogic **u**nit).

Wir schauen uns exemplarisch die 4-Bit-ALU SN74181 an.



SN74181, Schaltung

SELECTION	ACTIVE-HIGH DATA
S3 S2 S1 S0	M = L, ARITHMETIC FUNCTIONS
	$\overline{C}_n$ = H (no carry)
L L L L	F = A
L L L H	F = A + B
L L H L	F = A + $\overline{B}$
L L H H	F = MINUS 1 (2's COMPL)
L H L L	F = A PLUS A$\overline{B}$
L H L H	F = (A + B) PLUS A$\overline{B}$
L H H L	F = A MINUS B MINUS 1
L H H H	F = A$\overline{B}$ MINUS 1
H L L L	F = A PLUS AB
H L L H	F = A PLUS B
H L H L	F = (A + $\overline{B}$) PLUS AB
H L H H	F = AB MINUS 1
H H L L	F = A PLUS A
H H L H	F = (A + B) PLUS A
H H H L	F = (A + $\overline{B}$) PLUS A
H H H H	F = A MINUS 1

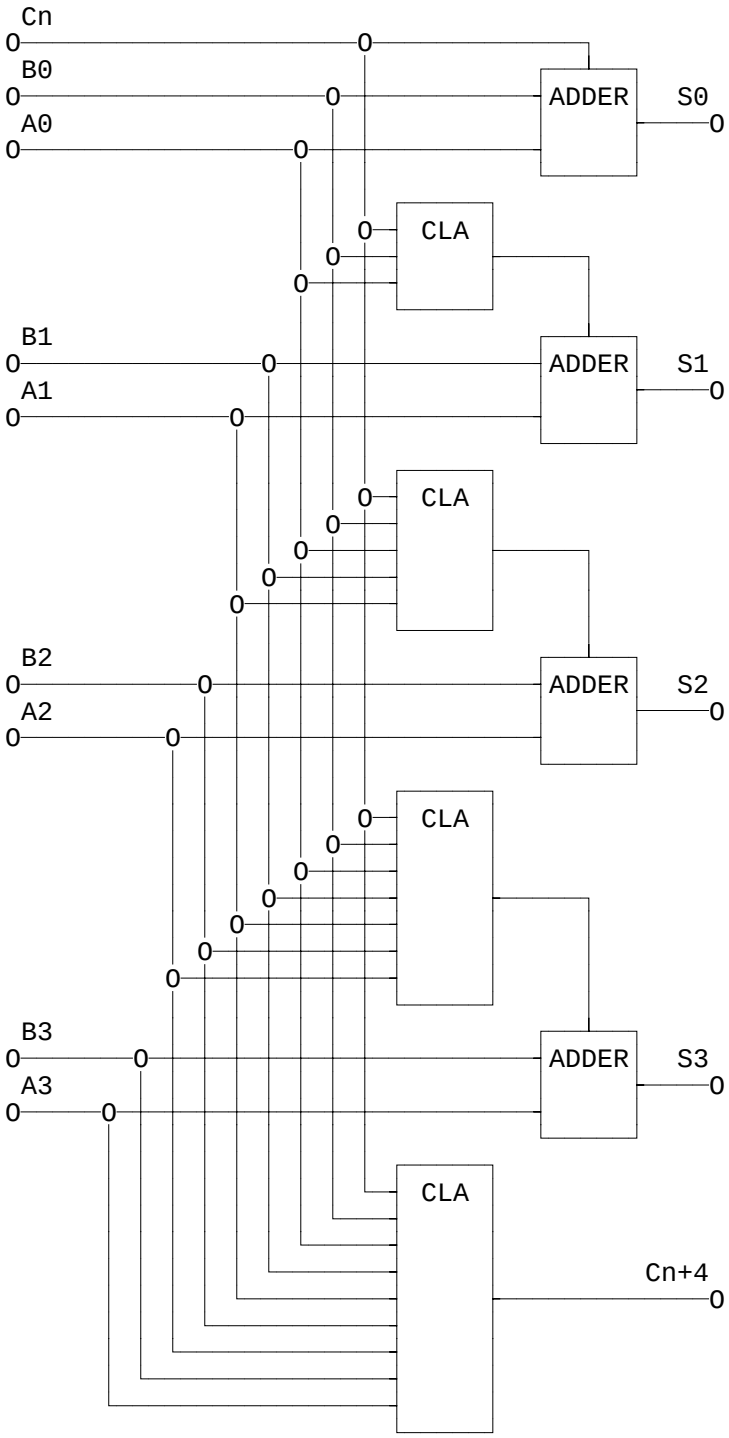
SELECTION	ACTIVE-HIGH DATA
S3 S2 S1 S0	M = L, ARITHMETIC FUNCTIONS
	$\overline{C}_n$ = L (with carry)
L L L L	F = A LPUS 1
L L L H	F = (A + B) PLUS 1
L L H L	F = (A + $\overline{B}$) PLUS 1
L L H H	F = ZERO
L H L L	F = A PLUS A$\overline{B}$ PLUS 1
L H L H	F = (A + B) PLUS A$\overline{B}$ PLUS 1
L H H L	F = A MINUS B
L H H H	F = A$\overline{B}$
H L L L	F = A PLUS AB PLUS 1
H L L H	F = A PLUS B PLUS 1
H L H L	F = (A + $\overline{B}$) PLUS AB + 1
H L H H	F = AB
H H L L	F = A PLUS A PLUS 1
H H L H	F = (A + B) PLUS A PLUS 1
H H H L	F = (A + $\overline{B}$) PLUS A PLUS 1
H H H H	F = A

SELECTION				ACTIVE-HIGH DATA
S3	S2	S1	S0	M = L, LOGIC FUNCTIONS
L	L	L	L	$F = \overline{A}$
L	L	L	H	$F = \overline{A + B}$
L	L	H	L	$F = \overline{A}B$
L	L	H	H	F = 0
L	H	L	L	$F = \overline{AB}$
L	H	L	H	$F = \overline{B}$
L	H	H	L	$F = A \oplus B$
L	H	H	H	$F = A\overline{B}$
H	L	L	L	$F = \overline{A} + B$
H	L	L	H	$F = \overline{A \oplus B}$
H	L	H	L	F = B
H	L	H	H	F = AB
H	H	L	L	F = 1
H	H	L	H	$F = A + \overline{B}$
H	H	H	L	F = A + B
H	H	H	H	F = A

SN74181, logische Funktionen

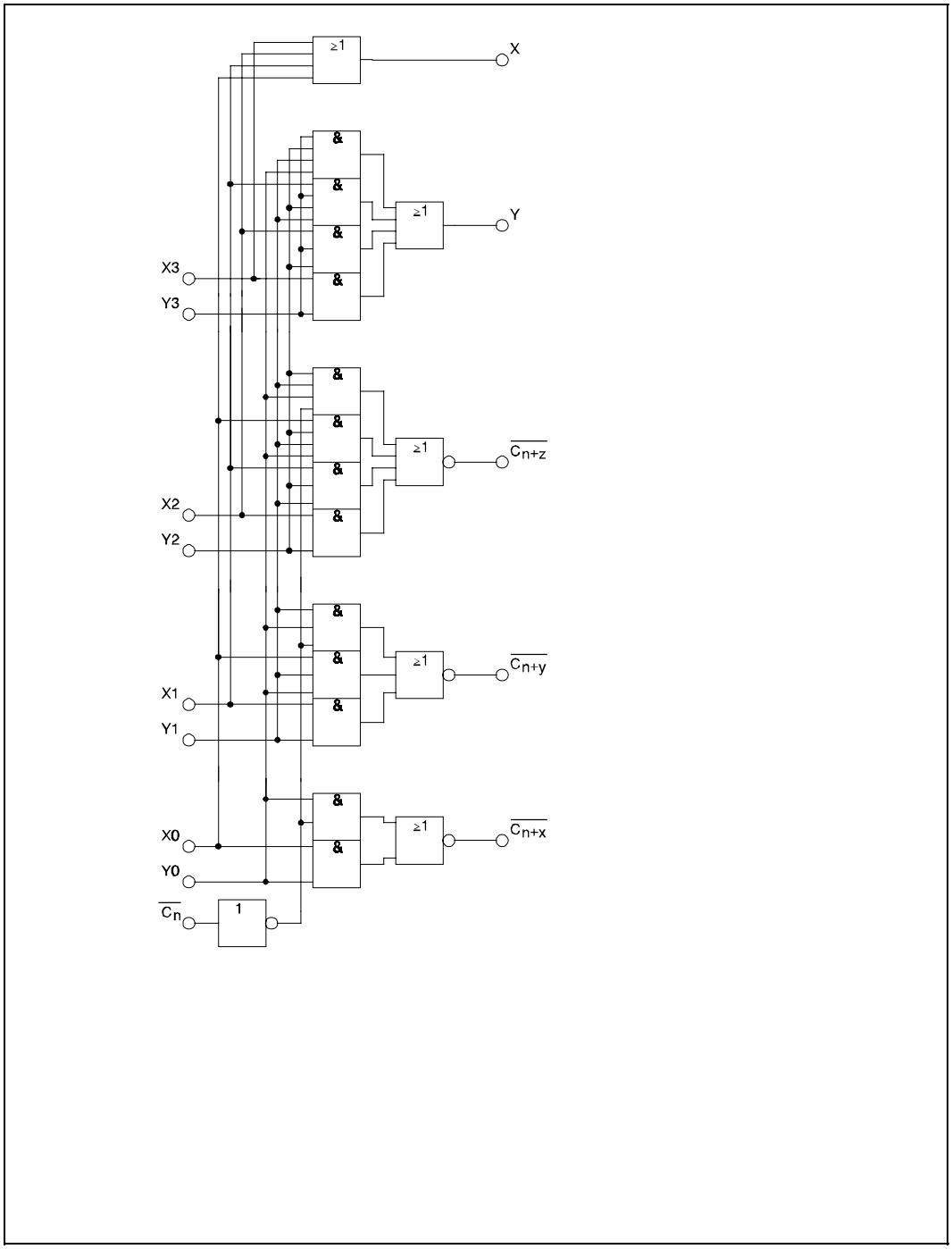
Bei der Analyse der Struktur der ALU SN74181 fällt auf, daß abweichend von der in 6.3. angegebenen Form der Übertragsbildung für jedes Summenbit und den auslaufenden Übertrag getrennte CLA-Schaltungen existieren. Das Summenbit S_0 liegt nach Ablauf von t_{hadd} an, die Summenbits S_1 bis S_3 nach $t_{cla} + t_{hadd}$ und der auslaufende Übertrag C_4 nach t_{cla}. Die Additionszeit beträgt also unabhängig von der Anzahl der Stellen $t_{cla} + t_{hadd}$.

Man bezeichnet einen solchen Adder im engeren Sinne als **Carry-Lookahead-Adder** im Gegensatz zum in 6.3. vorgestellten **Carry-Skip-Adder** (Das Übertragssignal "überspringt" eine Gruppe von k Bits). In der Literatur finden Sie weitere Verfahren und gute Übersichtsarbeiten, die die Addertheorie umfassend behandeln.



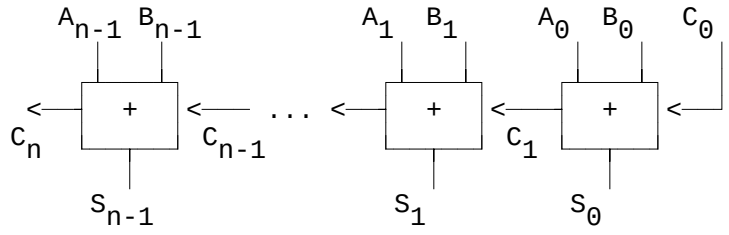
SN74181, Prinzip des Adders

Die Ausgänge X und Y dienen der vorausschauenden Übertragsberechnung für den Fall, daß mehrere 4-Bit-ALUs vom Typ SN74181 zu eine breiteren ALU zusammengeschaltet werden sollen. Der CLA-Schaltkreis SN74182 erlaubt die vorausschauende Übertragsberechnung für ALUs bis max. 16 Bit Verarbeitungsbreite. Für noch breitere ALUs können die CLA-Schaltkreise SN74182 kaskadiert werden.



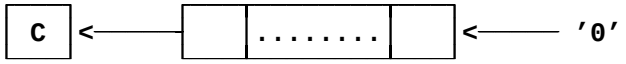
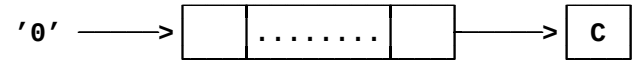
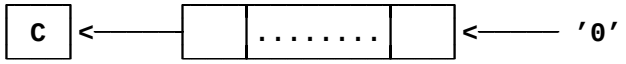
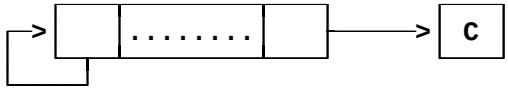
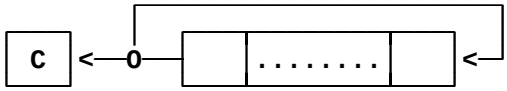
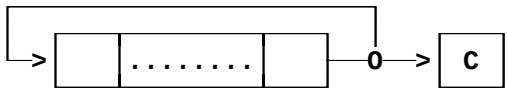
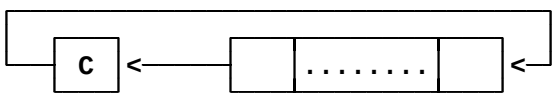
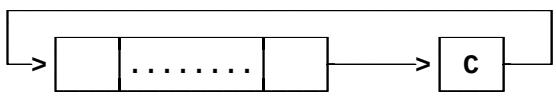
SN74182, Schaltung

In einer ALU werden neben dem gewünschten Verknüpfungsergebnis auch die sog. Flags erzeugt. Flags nehmen Informationen über gewisse Eigenschaften des Verknüpfungsergebnisses auf, die mit bedingten Sprungbefehlen (s. u. und LV Rechnerorganisation im 2. Semester) ausgewertet werden können. Darunter sind auch solche Informationen, die am Verknüpfungsergebnis selbst nicht mehr ablesbar sind.



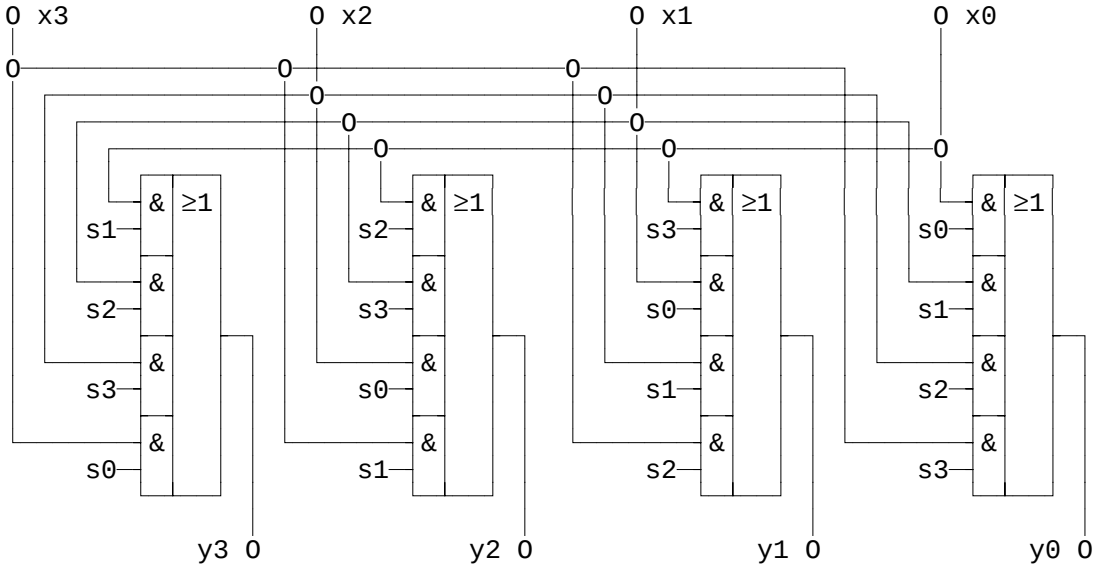
Flag	BOOLEsche Funktion
ZERO	$\overline{S_{n-1} \wedge \ldots \wedge S_1 \wedge S_0} = \overline{S_{n-1} \vee \ldots \vee S_1 \vee S_0}$
CARRY	C_n
OVERFLOW	$C_n \not+ C_{n-1}$
SIGN	S_{n-1}
PARITY	$S_{n-1} \not+ \ldots \not+ S_1 \not+ S_0$

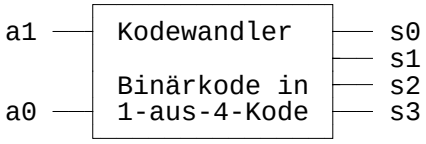
Unter den Maschinenbefehlen eines Digitalrechners finden sich auch sog. Schiebe- und Rotationsbefehle (engl. shift, rotate), z. B. bei den Mikroprozessoren INTEL 80x86:

SHL	Shift left	
SHR	Shift right	
SAL	Shift arithmetic left	
SAR	Shift arithmetic right	
ROL	Rotate left	
ROR	Rotate right	
RCL	Rotate left through carry	
RCR	Rotate right through carry	

Eine einfache aber langsame technische Realisierung erhält man bei Verwendung eines Schieberegisters (s. u.). Schneller aber aufwendiger sind die kombinatorischen Barrel-Shifter und Barrel-Rotator (von engl. to barrel = packen ?)

Ich gebe nachfolgend eine Schaltung für einen 4-Bit-Barrel-Rotator auf der Grundlage von 4-zu-1-Multiplexern an. In der technischen Realität werden oft sog. Transfer-Gatter eingesetzt, die eine deutlich einfachere Realisierung erlauben, die wir aber hier nicht behandeln wollen.





a1 a0	s0 s1 s2 s3		y3 y2 y1 y0
0 0	1 0 0 0	keine Rotation	x3 x2 x1 x0
0 1	0 1 0 0	Rotation um 1 Bit	x0 x3 x2 x1
1 0	0 0 1 0	Rotation um 2 Bit	x1 x0 x3 x2
1 1	0 0 0 1	Rotation um 3 Bit	x2 x1 x0 x3

6.5. Prüfung kombinatorischer Systeme

Die Korrektheit der technischen Realisierung einer BOOLEschen Funktion über B^k ist dann nachgewiesen, wenn die Abbildungsvorschrift

$$f: B^k \rightarrow B^1$$

für alle 2^k Belegungen des Binärvektors $\underline{x} = (x_1, \ldots, x_k)$ bestätigt ist. Eine kombinatorische Schaltung mit z. B. 10 Eingängen würde danach eine Testfolge aus $2^{10} = 1024$ unterschiedlichen 10-Bit-Testvektoren benötigen.

Ein empirisches Verfahren, das mit wesentlich weniger Testvektoren auskommt (und dafür aber auch weniger vollständig ist!), geht von der (sehr groben) Annahme aus, daß es für den Nachweis der Korrektheit hinreichend ist, Einfach-Stuck-at-Fehler am Rand der technischen Realisierung auszuschließen. Ein Stuck-at-Fehler an einem Eingang oder Ausgang der technischen Realisierung ist ein Fehler, der so wirkt, als würde der Pegel an diesem Ein- oder Ausgang unabhängig vom Testvektor fest auf 0 (Stuck-at-0-Fehler, SA0-Fehler) oder fest auf 1 (Stuck-at-1-Fehler, SA1-Fehler) liegen. Bei TTL-Gattern entsprechen z. B. der Kurzschluß einer Leitung zu 0 Volt (Masse) dem SA0-Fehler an dieser Leitung, der Kurzschluß einer Leitung zu +5 Volt (Betriebsspannung) dem SA1-Fehler an dieser Leitung und eine Leitungsunterbrechung dem SA1-Fehler an allen damit "offenen" Eingängen. Die Einschränkung auf Einfachfehler bedeutet, daß weiter vereinfachend angenommen wird, daß in einer technischen Realisierung zu einem Zeitpunkt - wenn überhaupt - nur genau ein Stuck-at-Fehler auftritt.

Dem Nachweis der Korrektheit liegt folgendes Szenario zugrunde:

- Die Menge der Fehler ist abzählbar (Anzahl der Fehlerorte * Anzahl der Fehlerarten). Für jeden Fehler ist wenigstens ein Testvektor zu bestimmen, der geeignet ist, diesen Fehler zu erkennen.

- Um einen bestimmten Fehler zu erkennen, ist an die Eingänge der zu verifizierenden technischen Realisierung ein dafür geeigneter Testvektor anzulegen. Am Ausgang (oder an den Ausgängen) der technischen Realisierung sind der sich einstellende Istpegel und der Sollpegel laut Abbildungsvorschrift zu vergleichen. Stimmen die Pegel nicht überein, so liegt einer der Fehler vor, die der angelegte Testvektor geeignet ist zu erkennen (es gelingt häufig nicht, Testvektoren zu finden, die nur genau einen Fehler erkennen!).

- Sind die Testvektoren für alle betrachteten Fehler angelegt und traten dabei keine Abweichungen zwischen den

Ist- und den Sollausgangspegeln auf, so kann mit Sicherheit nur ge-schlossen werden, daß in der technischen Realisierung keiner der betrachteten Fehler enthalten ist. Jeder nichtbetrachtete Fehler kann durchaus noch enthalten sein.

Ein Testvektor, der geeignet ist, einen SA0-(SA1-)-Fehler an einem Eingang der technischen Realisierung zu erkennen, muß zwei Bedingungen genügen:

- Provokationsbedingung: Am Fehlerort ist der Pegel 1(0) anzulegen.
  (Um das Vorhandensein z. B. eines Kurzschlusses zu Masse an einem Eingang (SA0-Fehler) an einem Ausgang beobachten zu können, muß der Eingang mit 1 belegt werden, um im Falle des Kurzschlusses die Verfälschung des Pegels zu 0 überhaupt registrieren zu können. Der Fehler muß "provoziert" werden)

- Transportbedingung: Die Belegung der anderen Eingänge ist so zu wählen, daß der Pegel mindestens eines Ausgangs vom Pegel am Fehlerort abhängt.
  (Falls sich der Pegel am Fehlerort ändert (beim Auftreten des Fehlers), muß sich der Pegel mindestens eines Ausgangs ebenfalls ändern. Der im Fehlerfalle registrierte Pegelwechsel wird auf diesen Ausgang "transportiert")

Daraus folgt die Berechnungsvorschrift für die Menge der Testvektoren, die geeignet sind, einen SA0-(SA1-)-Fehler an einem Eingang x_i der technischen Realisierung einer BOOLEschen Funktion f über B^k zu erkennen.

SA0-Defekt an x_i:

$$T_{x_i \equiv 0} = x_i \wedge \left(f(\underline{x})\Big|_{x_i=0} \not\leftrightarrow f(\underline{x})\Big|_{x_i=1}\right) \stackrel{!}{=} 1$$

SA1-Defekt an x_i:

$$T_{x_i \equiv 1} = \overline{x_i} \wedge \left(f(\underline{x})\Big|_{x_i=0} \not\leftrightarrow f(\underline{x})\Big|_{x_i=1}\right) \stackrel{!}{=} 1$$

Da der Ausgang stets beobachtbar ist, entfällt für Fehler am Ausgang die Transportbedingung:

$$T_{f \equiv 0} = f \stackrel{!}{=} 1$$

$$T_{f \equiv 1} = \overline{f} \stackrel{!}{=} 1.$$

Beispiel:

Es sei für den 2-zu-1-Multiplexer eine Testfolge zu berechnen.

$f = \bar{s} \wedge d_0 \vee s \wedge d_1$

Berechnung der Transportbedingungen:

$f|_{s=0} \not\leftrightarrow f|_{s=1} = d_0 \not\leftrightarrow d_1$

$f|_{d_0=0} \not\leftrightarrow f|_{d_0=1} = \bar{s}$

$f|_{d_1=0} \not\leftrightarrow f|_{d_1=1} = s$

Berechnung der Mengen der Testvektoren:

$T_{s \equiv 0} = s \wedge (d_0 \not\leftrightarrow d_1) \stackrel{!}{=} 1$

$\Rightarrow (d_0,d_1,s) = (0,1,1)|(1,0,1)$

$T_{s \equiv 1} = \bar{s} \wedge (d_0 \not\leftrightarrow d_1) \stackrel{!}{=} 1$

$\Rightarrow (d_0,d_1,s) = (0,1,0)|(1,0,0)$

$T_{d_0 \equiv 0} = d_0 \wedge \bar{s} \stackrel{!}{=} 1$

$\Rightarrow (d_0,d_1,s) = (1,0,0)|(1,1,0)$

$T_{d_0 \equiv 1} = \overline{d_0} \wedge \bar{s} \stackrel{!}{=} 1$

$\Rightarrow (d_0,d_1,s) = (0,0,0)|(0,1,0)$

$T_{d_1 \equiv 0} = d_1 \wedge s \stackrel{!}{=} 1$

$\Rightarrow (d_0,d_1,s) = (0,1,1)|(1,1,1)$

$T_{d_1 \equiv 1} = d_1 \overline{\wedge} s = 1^{!}$

$\Rightarrow (d_0,d_1,s) = (0,0,1)|(1,0,1)$

$T_{f \equiv 0} = \bar{s} \wedge d_0 \vee s \wedge d_1 \stackrel{!}{=} 1$

$\Rightarrow (d_0,d_1,s) = (0,1,1)|(1,0,0)|(1,1,0)|(1,1,1)$

$$T_{f\equiv 1} = \bar{s}\wedge\bar{d}_0 \vee s\wedge\bar{d}_1 \stackrel{!}{=} 1$$

$$\Rightarrow (d_0,d_1,s) = (0,0,0)\,|\,(0,0,1)\,|\,(0,1,0)\,|\,(1,0,1)$$

Tabellarisch ist das Ergebnis in der Testmatrix (Bild 5) zusammengefaßt. TV steht für "Testvektor", δ(TV) für das Dezimaläquivalent des Testvektors. In die Matrix ist das Attribut

$e_{i,j}$ mit: i = δ(TV),

j ∈ Menge der betrachteten Fehler,

e { = 0, falls der Testvektor i nicht geeignet ist, den Fehler j zu erkennen
= 1, falls der Testvektor i geeignet ist, den Fehler j zu erkennen

eingetragen (der besseren Übersicht halber nur die Einsen).

Fehler	0	1	2	3	4	5	6	7	δ(TV)
	(000)	(001)	(010)	(011)	(100)	(101)	(110)	(111)	TV
s ≡0				1		1			
s ≡1			1		1				
d_0≡0					1		1		
d_0≡1	1		1						
d_1≡0				1				1	
d_1≡1		1				1			
f ≡0				1	1		1	1	
f ≡1	1	1	1			1			

Bild 5. Testmatrix

Das Ziel der weiteren Betrachtungen besteht darin, die Menge der Testfolgen zu ermitteln, die alle betrachteten Fehler erkennen. Für jede dieser Testfolgen gilt offensichtlich

$$\bigwedge_j \left(\bigvee_i e_{i,j} \right) \stackrel{!}{=} 1,$$

d. h. in ihr muß für jeden betrachteten Fehler wenigstens ein Testvektor enthalten sein, der diesen Fehler erkennt.

Für das Beispiel folgt

$$(e_{3,s\equiv 0} \vee e_{5,s\equiv 0}) \wedge (e_{2,s\equiv 1} \vee e_{4,s\equiv 1}) \wedge \ldots \stackrel{!}{=} 1,$$

wobei der zweite Index für diese Berechnung entfallen kann

$$(e_3 \vee e_5) \wedge (e_2 \vee e_4) \wedge (e_4 \vee e_6) \wedge (e_0 \vee e_2) \wedge (e_3 \vee e_7) \wedge (e_1 \vee e_5) \wedge$$
$$\wedge (e_3 \vee e_4 \vee e_6 \vee e_7) \wedge (e_0 \vee e_1 \vee e_2 \vee e_5) \stackrel{!}{=} 1.$$

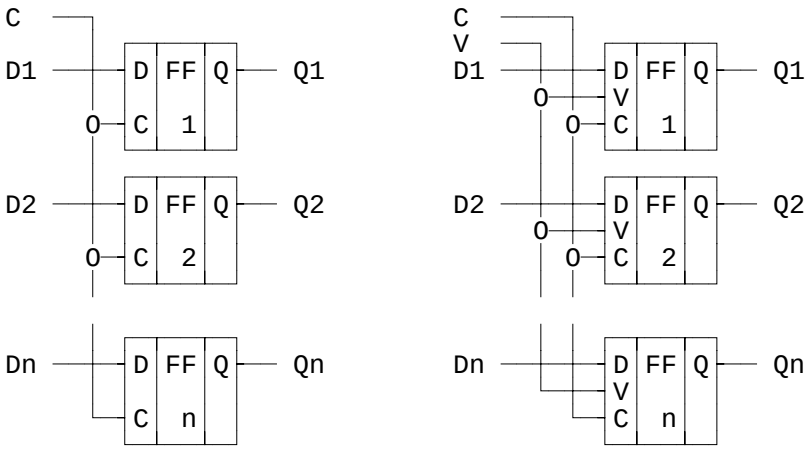
Ausmultiplizieren liefert

$$e_0 \wedge e_1 \wedge e_3 \wedge e_4 \vee e_0 \wedge e_3 \wedge e_4 \wedge e_5 \vee e_0 \wedge e_4 \wedge e_5 \wedge e_7 \vee e_1 \wedge e_2 \wedge e_3 \wedge e_4 \vee$$
$$\vee e_2 \wedge e_3 \wedge e_4 \wedge e_5 \vee e_2 \wedge e_4 \wedge e_5 \wedge e_7 \vee e_1 \wedge e_2 \wedge e_3 \wedge e_6 \vee e_2 \wedge e_3 \wedge e_5 \wedge e_6 \vee$$
$$\vee e_2 \wedge e_5 \wedge e_6 \wedge e_7 \stackrel{!}{=} 1,$$

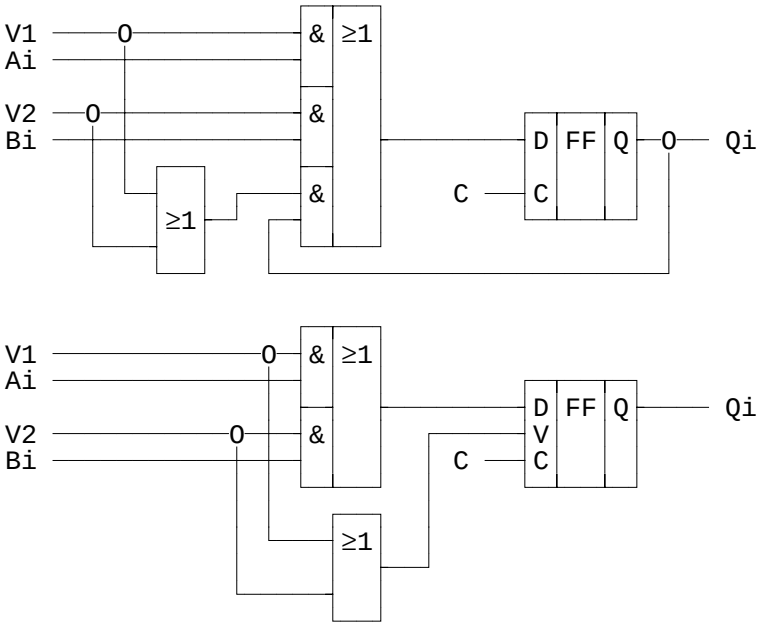
d. h. es gibt neun mögliche unterschiedliche Testfolgen, die gleichermaßen geeignet sind, alle betrachteten Fehler zu erkennen. In diesem Falle sind sie alle gleich lang, sie bestehen aus jeweils vier Testvektoren, aus den Testvektoren 0, 1, 3 und 4 oder den Testvektoren 0, 3, 4 und 5 usw. oder den Testvektoren 2, 5, 6 und 7.

6.6. Register

Register sind eindimensionale Arrays aus Flipflops, vorzugsweise des gleichen Typs. Beim Automatenentwurf (s. 5.2.2.) haben wir bereits einen Einsatzfall kennengelernt - den Zustandsspeicher. In Digitalrechnern werden Register außerdem vielfältig zur (temporären) Speicherung von Daten (Operanden, Zwischenergebnisse, Resultate, Befehle, ...) verwendet. Hier kommen meist Register aus D- oder DV-Flipflops zum Einsatz.

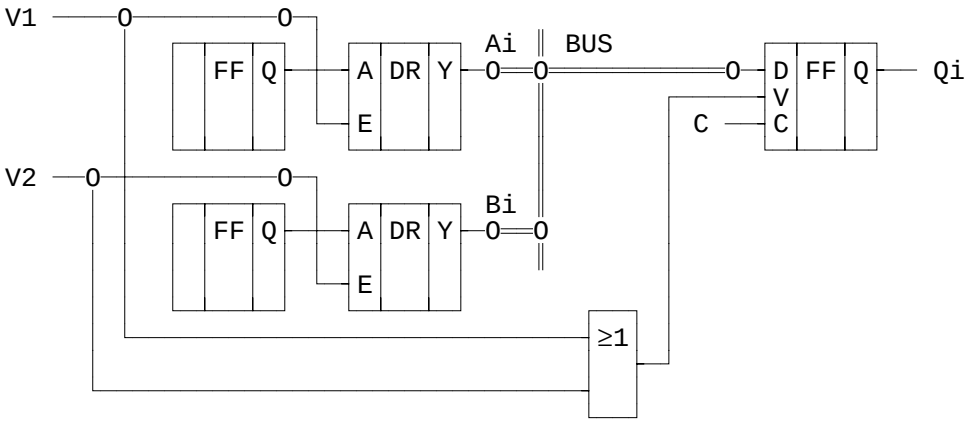


Ein Schaltungsbeispiel soll den Vorteil des DV-Regsiters gegenüber dem D-Register verdeutlichen. Nach Maßgabe der beiden Steuerspannungen V_1 und V_2 sollen entweder der Vektor A (V_1V_2 = 10) oder der Vektor B (V_1V_2 = 01) ins Register übernommen werden, oder das Register soll seinen vorhergehenden Wert behalten (V_1V_2 = 00). Es ist jeweils nur ein Flipflop dargestellt.

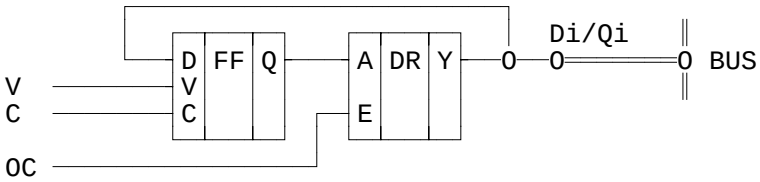


Anstelle der gesteuerten ODER (s. 6.1.3.) könnten auch Multiplexer verwendet werden. Register mit Tri-State-Ausgängen ver-

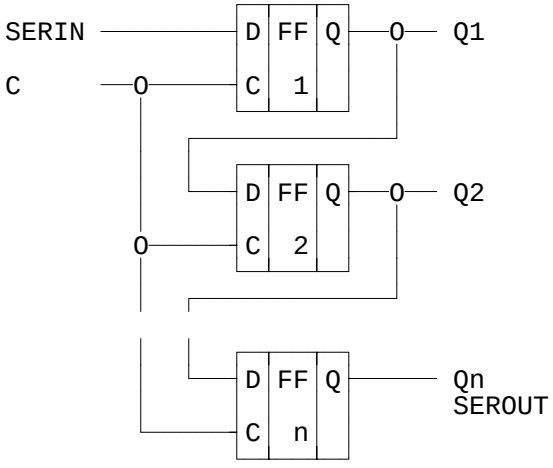
einfachen die Schaltung weiter.



Schließlich werden auch Register mit bidirektionalen Tri-State-Ein-/Ausgängen verwendet.



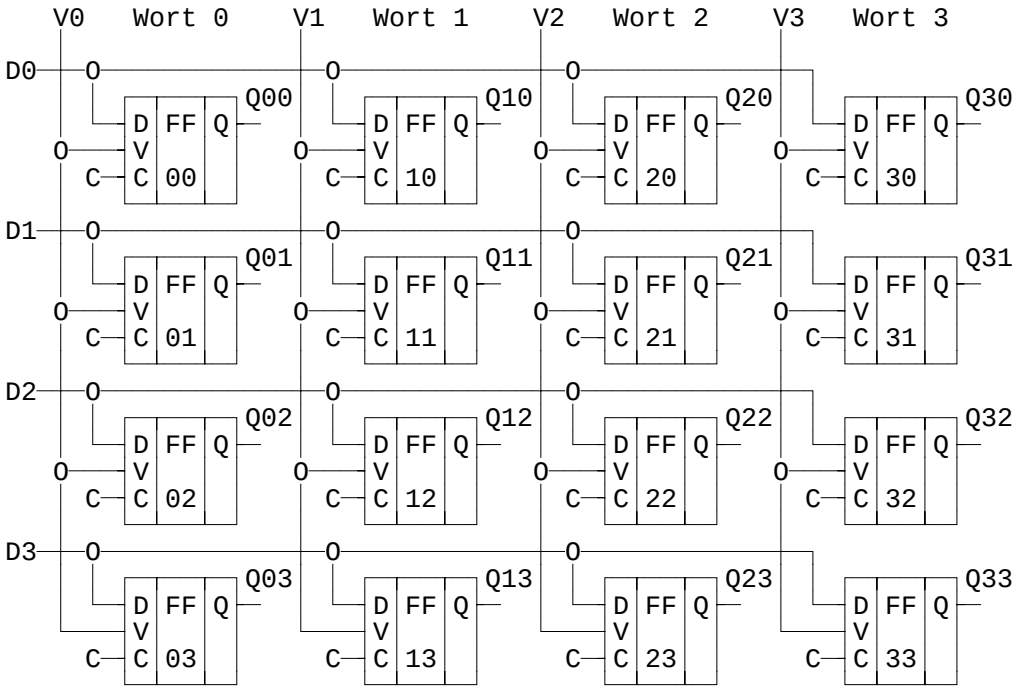
Eine spezielle Art von Registern sind Schieberegister (Umlaufspeicher) im Gegensatz zu den bisher behandelten Parallelregistern. Sie können zur Realisierung der Schiebe- und Rotationsbefehle (s. 6.4.) und der Serien-/Parallel- und Parallel-/Serien-Wandlung verwendet werden. Auch Zähler (Johnsonzähler, s. u.) lassen sich realisieren. Rückgekoppelte Schieberegister spielen eine Rolle bei der Berechnung der Signatur bei der Datenübertragung (CRC-Polynom). Schieberegister werden im prüfgerechten Entwurf (Scan and Set Design) eingesetzt.

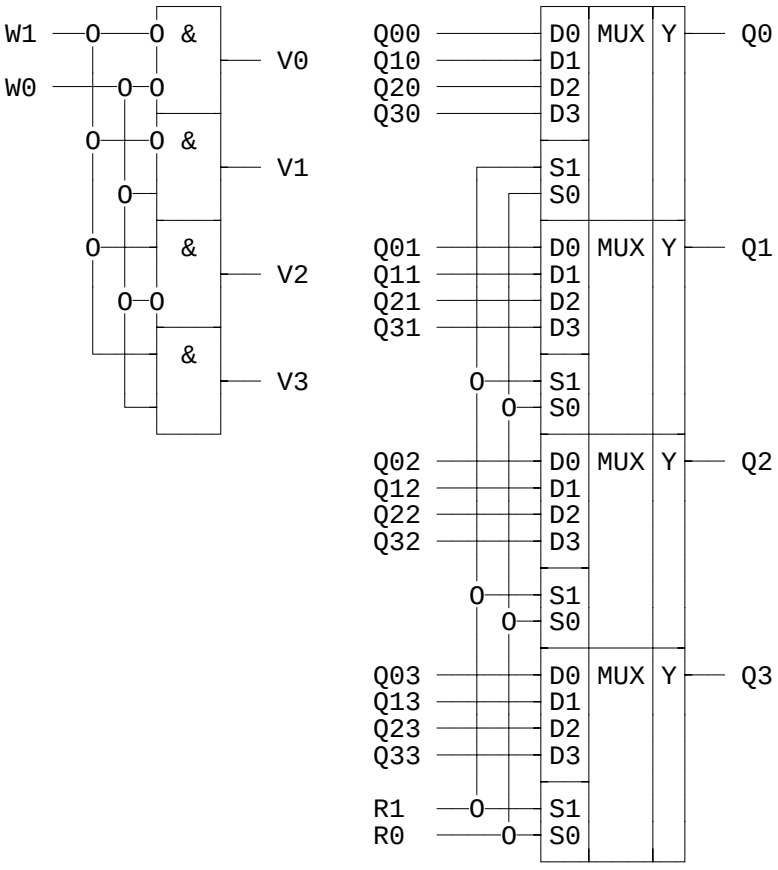


Reale Schaltkreise sind häufig multifunktionale Register, die sowohl als Parallel- wie auch als Schieberegister (shift left, shift right, rotate left, rotate right) betrieben werden kön-

nen.

Register können zu Registerbänken oder -sätzen zusammengeschaltet werden, z. B. einem 4x4-Registerblock.



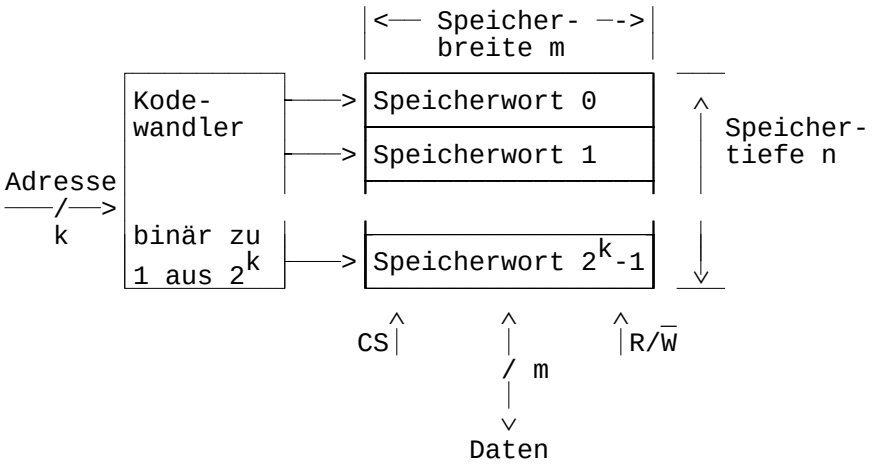


Registerblöcke verfügen meist über getrennte, globale Enable-Signale für das Lesen und das Schreiben (hier nicht dargestellt). In der Regel wird dem Registerblock nur eine Adresse zugeführt (W = R), von der gelesen wird und auf die gleichzeitig (mit C = 1 bei statischen Flipflops) geschrieben werden kann. Im Beispiel oben kann gleichzeitig auf die Adresse W geschrieben und von einer ggf. anderen Adresse R gelesen werden. Solche Registerblöcke heißen deshalb auch Dual-Port-Memory (DPM) oder allgemein Multiple-Port-Memory (MPM). Um mehrere solcher Registerblöcke zu größeren Registerblöcken zusammenschalten zu können, versieht man Registerblöcke häufig mit Tri-State-Ausgängen oder bidirektionalen Tri-State-Ein-/Ausgängen(s. o.).

6.7. Speicher

Im engeren Sinne soll hier von RAMs die Rede sein (RAM = random access memory, Schreib-Lese-Speicher mit wahlfreiem Zugriff). Man unterscheidet eine ganze Reihe von unterschiedlichen RAM-Typen (SRAM, DRAM, SDRAM, ...) und unterschiedlichen Organisationsprinzipien, die zu behandeln den Rahmen der Lehrveranstaltung sprengen würde.

Wir beschränken uns deshalb auf das SRAM, für das der oben behandelte Registerblock Vorbild ist, und wollen nur einige grundlegende Begriffe und Bezeichnungen kennenlernen.



Adreßbreite	k Bit
Speicherbreite (Aufrufbreite)	m Bit
Speichertiefe	$n = 2^k$
Speicherkapazität	m*n Bit
Bezeichnung	m*n-Bit-RAM

Anm.: mit 8 Bit = 1 Byte werden die Speicherbreite und die

Speicherkapazität häufig in Byte und Vielfachen davon (1 K = 2^{10}, 1 M = 2^{20}, 1 G = 2^{30}, ...) angegeben.

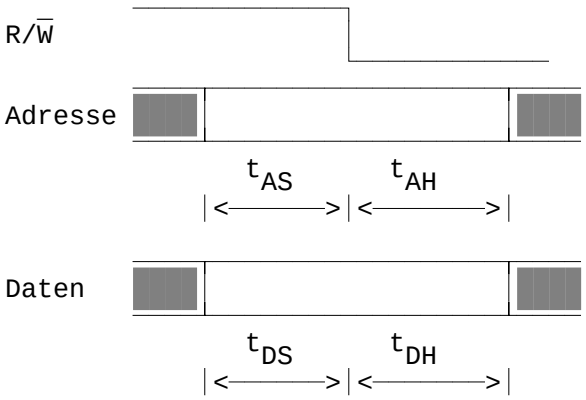
Die Steuerspannung CS (Chip Select) aktiviert einen Speicherschaltkreis bei Zusammenschaltung mehrerer Speicherschaltkreise zu einem Speicher größerer Kapazität. Die Steuerspannung

$R/\overline{W}$ wählt aus, ob gelesen oder geschrieben werden soll.

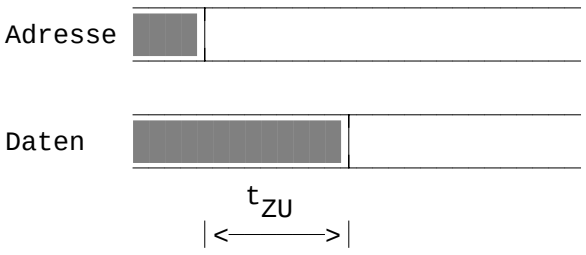
Die dynamischen Eigenschaften eines Speichers bestimmen wesentlich die Arbeitsgeschgwindigkeit des digitalen Systems, in dem er eingesetzt ist. Wir wollen deshalb noch einige wesentliche dynamischen Kennwerte des Speichers anführen.

Der Schreibvorgang werde durch die fallende Flanke des Signals

$R/\overline{W}$ ausgelöst. Um den Schreibvorgang korrekt ausführen zu können, müssen die Adresse t_{AS} (**Adreß-Setup-Zeit**) und die Daten t_{DS} (**Daten-Setup-Zeit**) früher anliegen und dürfen erst nach t_{AH} (**Adreß-Hold-Zeit**) bzw. nach t_{DH} (**Daten-Hold-Zeit**) wieder geändert werden.



Unter **Zugriffszeit** versteht man beim Lesevorgang die Zeit, die zwischen dem Anlegen der Adresse und dem Anliegen des Inhalts der adressierten Speicherzelle (= Speicherwort) vergeht.



Oft gibt man die **Zykluszeit** an und versteht darunter das minimal zulässige Zeitintervall zwischen zwei aufeinanderfolgenden Speicherzugriffen. Da die gelesenen Daten auch sicher in das am Speicherausgang angeschlossene Datenregister übernommen werden müssen, ist die Zykluszeit etwas größer als die Zugriffszeit. Der Kehrwert der Zykluszeit ist die sog. **maximale Datenrate**.

6.8. Zähler und Frequenzteiler

In 5.2. haben wir bereits Zähler und Frequenzteiler entworfen. Grundsätzliche Unterschiede haben wir nicht erkannt. Folglich müßte sich ein Zähler als Frequenzteiler und umgekehrt ein Frequenzteiler auch als Zähler einsetzen lassen.

Gemeinsam ist zunächst, daß ein Zähler mit n Zuständen und ein Frequenzteiler, der aus einer Impulsfolge der Frequenz f eine Impulsfolge der Frequenz f/n bildet, durch ein und denselben Automatengraphen beschrieben werden können.

Bei genauerer Betrachtung ergibt sich aber:

1. Ein Zähler muß initialisierbar sein, da das Zählergebnis nur in Bezug auf einen definierten Anfangswert sinnvoll interpretierbar ist. Bei einem Frequenzteiler kann auf das Initialisieren verzichtet werden, da von ihm "nur" erwartet wird, daß er über einen großen Zeitraum hinweg die Frequenz f/n liefert.

2. Bei einem Zähler muß der aktuelle Zustand auswertbar sein, da dieser das Zählergebnis repräsentiert. Bei einem Frequenzteiler genügt ein einziger Ausgang. Das kann bei geschickter Kodierung der Ausgang eines der Flipflops sein.

Zähler werden als Frequenzteiler einsetzbar sein, Frequenzteiler als Zähler nur bedingt. Wenn die sequentielle Schaltung aus handelsüblichen TTL-Schaltkreisen geringen Integrationsgrads (SSI = small scale integration) aufgebaut wird, hat es der Entwerfer immer in der Hand, die für den Einsatzfall geforderten Bedingungen einzuhalten. Setzt er hingegen fertig konfektionierte Frequenzteiler höheren Integrationsgrads ein, dann wird er diese in aller Regel nicht als Zähler betreiben können.

In der Literatur finden Sie eine Vielzahl von Zähler- und Frequenzteilerschaltungen. Vollständigkeit ist kaum erreichbar, obwohl es durchaus sehr umfangreiche Schaltungssammlungen gibt. Wir können sie (fast) alle entwerfen, sobald wir wissen, was sie tun sollen. Das soll aber nicht heißen, daß ich Ihnen empfehle, solche Schaltungssammlungen zu ignorieren. Dort finden sich häufig Lösungen, die Sie nur nach längerem Nachdenken finden würden. Andererseits sollten Sie Schaltungen aus Schaltungssammlungen auch nicht kritiklos übernehmen.

Ich trage an dieser Stelle noch einige Beispiele nach.

6.8.1. Johnson-Zähler

Ich habe bereits erwähnt, daß sich Schieberegister auch als Zähler einsetzen lassen. Wir sehen uns einen 4-Bit-Johnson-Zähler an und verwenden D-Flipflops. Wir streben eine möglichst einfache Realisierung an, und gehen davon aus, daß der Initialzustand Q = 0000 statisch einstellbar ist.

Automatentabelle

Q3	Q2	Q1	Q0	Q3′	Q2′	Q1′	Q0′
0	0	0	0	1	0	0	0
1	0	0	0	1	1	0	0
1	1	0	0	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1
0	1	1	1	0	0	1	1
0	0	1	1	0	0	0	1
0	0	0	1	0	0	0	0
0	0	1	0	-	-	-	-
0	1	0	0	-	-	-	-
0	1	0	1	-	-	-	-
0	1	1	0	-	-	-	-
1	0	0	1	-	-	-	-
1	0	1	0	-	-	-	-
1	0	1	1	-	-	-	-
1	1	0	1	-	-	-	-

Eintragen in KARNAUGH-Pläne liefert:

Q3′	0 0	1 1	Q1
	0 1	1 0	Q0
0 0	**1** 0	0 **-**	
0 1	**-** -	0 **-**	
1 1	**1** -	0 **1**	
1 0	**1** -	- **-**	
Q3 Q2			

Q3′ = /Q0

Q2′	0 0	1 1	Q1
	0 1	1 0	Q0
0 0	0 0	0 -	
0 1	- -	0 -	
1 1	**1** -	**1 1**	
1 0	**1** -	- **-**	
Q3 Q2			

Q2′ = Q3

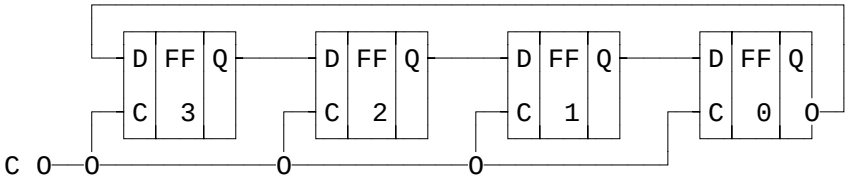
Q1′	0 0	1 1	Q1
	0 1	1 0	Q0
0 0	0 0	0 -	
0 1	**-** -	**1** -	
1 1	**1** -	**1 1**	
1 0	0 -	- -	
Q3 Q2			

Q1′ = Q2

Q0′	0 0	1 1	Q1
	0 1	1 0	Q0
0 0	0 0	**1** -	
0 1	- -	**1** -	
1 1	0 -	**1 1**	
1 0	0 -	**-** -	
Q3 Q2			

Q0′ = Q1

Die Realisierung führt auf ein Schieberegister.

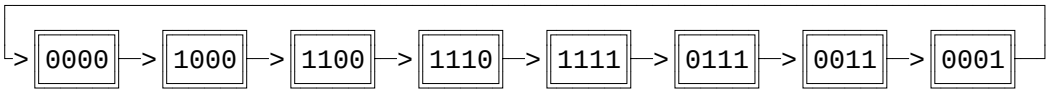


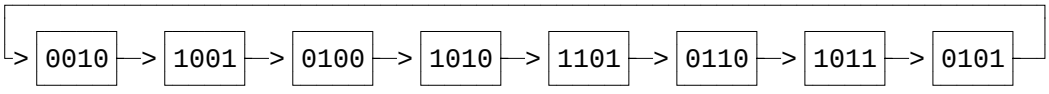
Die vielen redundanten Zustände geben trotz der vereinbarten Initialisierbarkeit Anlaß zur Sorge. Was passiert, wenn sich der Zähler in einen solchen redundanten Zustand "verirrt"? Mit der gefundenen Überführungsfunktion kann die Automatentabelle vervollständigt werden.

Automatentabelle, vervollständigt

Q3	Q2	Q1	Q0	Q3′	Q2′	Q1′	Q0′
0	0	0	0	1	0	0	0
1	0	0	0	1	1	0	0
1	1	0	0	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1
0	1	1	1	0	0	1	1
0	0	1	1	0	0	0	1
0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0
0	1	1	0	1	0	1	1
1	0	0	1	0	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	0	1	0	1
1	1	0	1	0	1	1	0

Zum nicht geringen Schrecken ergibt sich, daß der Zustandsgraph in zwei disjunkte Teilgraphen zerfällt. Wenn sich der Zähler einmal "irrt", verbleibt er in dem falschen Zählmodus.





Was tun? Wenn man ein fertiges Schieberegister verwenden will, kommt nur in Frage, D3 = Q3′ zu modifizieren. In der Literatur findet sich die Lösung

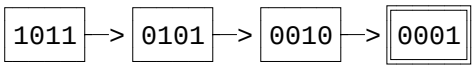
$$Q_3' = \overline{Q_1}\,\overline{Q_0} \vee Q_3\overline{Q_0}.$$

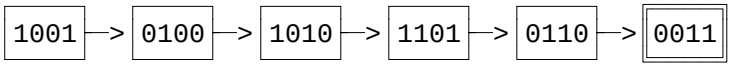
Mit dieser Funktion kann die Automatentabelle korrigiert werden.

Automatentabelle, 2. Hälfte, korrigiert

Q3	Q2	Q1	Q0	Q3’	Q2’	Q1’	Q0’
0	0	1	0	**0**	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0
0	1	1	0	**0**	0	1	1
1	0	0	1	0	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	0	1	0	1
1	1	0	1	0	1	1	0

Es ergeben sich die nachfolgenden Übergänge. Der Zähler findet, falls er sich mal "irrt", selbständig in die korrekte Zustandsfolge zurück.

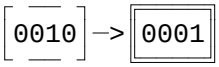


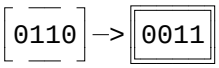


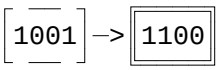
Unschön ist, daß der Zähler im Extremfall 5 Taktzyklen braucht, bis er wieder zurückfindet. Wir wollen den Versuch unternehmen, die minimal mögliche Anzahl "falscher" Übergänge zu finden. Da nur D3 = Q3’ modifiziert werden kann, suchen wir zunächst solche Zustände (Zustandsmenge 1), die bei Negation des Bits Q3’ unmittelbar in einen regulären Zustand übergehen, und finden die Übergänge

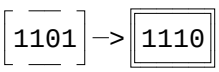
Automatentabelle, 2. Hälfte, korrigiert, Variante 2, Schritt 1

Q3	Q2	Q1	Q0	Q3’	Q2’	Q1’	Q0’
0	**0**	**1**	**0**	**0**	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0
0	**1**	**1**	**0**	**0**	0	1	1
1	**0**	**0**	**1**	**1**	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	0	1	0	1
1	**1**	**0**	**1**	**1**	1	1	0







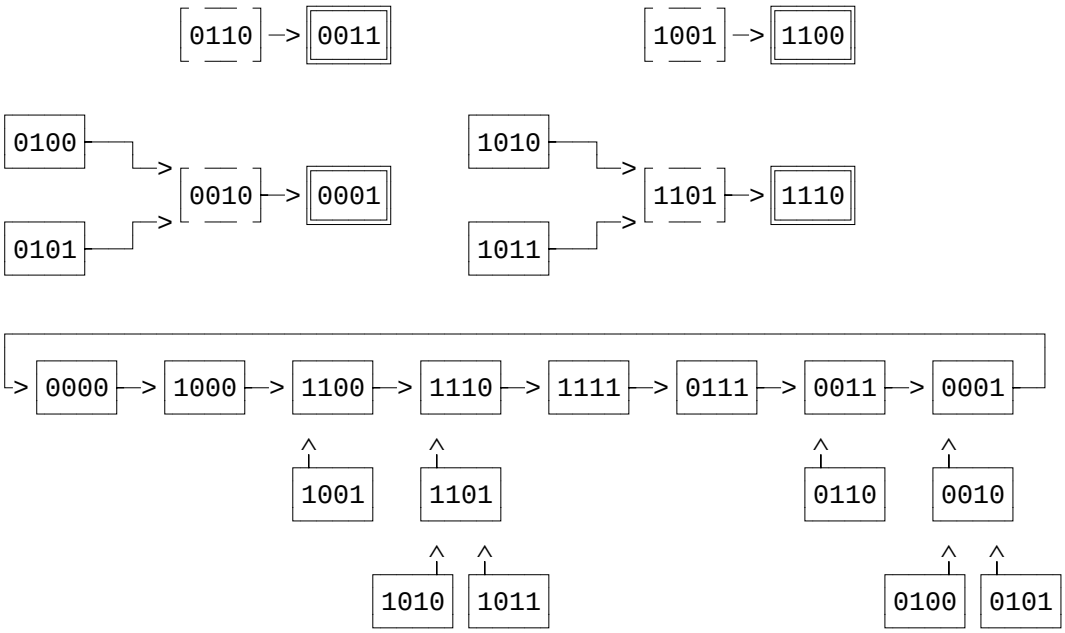


Die verbleibenden Zustände untersuchen wir danach, ob sie bereits jetzt oder nach Negation des Bits Q3’ in einen Zustand der Zustandsmenge 1 übergehen, und finden die Übergänge

Automatentabelle, 2. Hälfte, korrigiert, Variante 2, Schritt 2

Q3	Q2	Q1	Q0	Q3′	Q2′	Q1′	Q0′
0	0	1	0	0	0	0	1
0	**1**	**0**	**0**	**0**	0	1	0
0	**1**	**0**	**1**	0	0	1	0
0	1	1	0	0	0	1	1
1	0	0	1	1	1	0	0
1	**0**	**1**	**0**	1	1	0	1
1	**0**	**1**	**1**	**1**	1	0	1
1	1	0	1	1	1	1	0

Insgesamt ergibt sich



Eintragen in den KARNAUGH-Plan liefert:

Q3′ \ Q1 Q0	0 0	0 1	1 1	1 0
0 0	1	0	0	0
0 1	0	0	0	0
1 1	1	1	0	1
1 0	1	1	1	1

Q3 Q2

$$Q_3' = \overline{Q_2}\ \overline{Q_1}\ \overline{Q_0} \vee Q_3\ \overline{Q_0} \vee Q_3\ \overline{Q_1} \vee Q_3\ \overline{Q_2}$$

6.8.2. Binary Rate Multiplier (BRM)

Wir haben bislang nur Frequenzteiler mit ganzzahligen Teilerfaktoren entworfen. Mit einem BRM lassen sich auch nichtganzzahlige

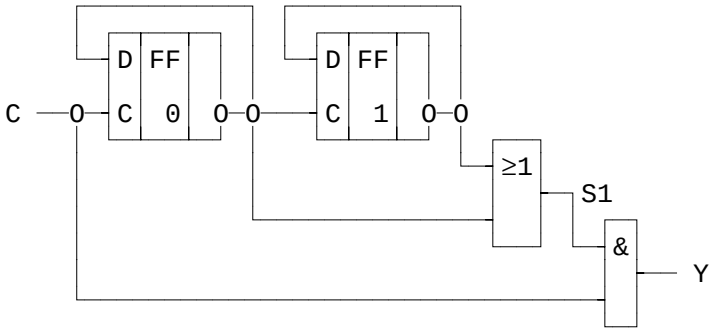
Faktoren realisieren. Dazu muß man jedoch von der Vorstellung abgehen, die gesuchte Impulsfolge müsse periodisch sein. Mit der etwas weitherzigeren Definition

Frequenz = mittlere Anzahl Impulse pro Zeiteinheit

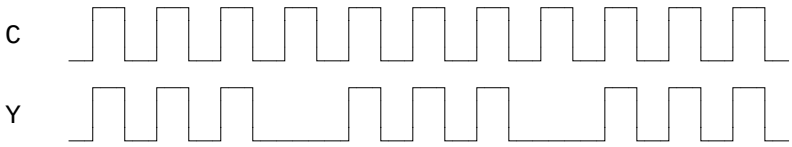
lassen wir auch nichtperiodische Impulsfolgen zu.

Das Prinzip soll am Beispiel eines Frequenzteilers erläutert werden, der aus einer Impulsfolge der Frequenz f eine Impulsfolge der Frequenz f/n = 0,75*f ableitet, d.h. n = 4/3.

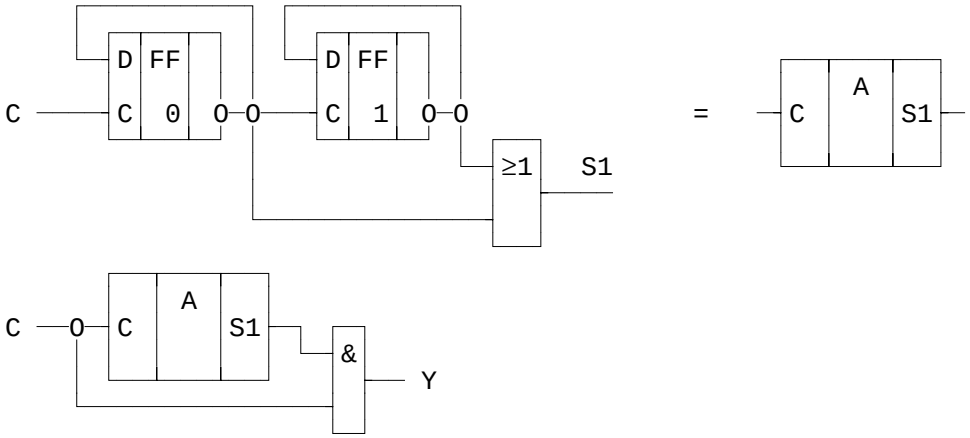
Bei einem (hier asynchronen) 2-Bit-Binärzähler werden die Zustände Q_1Q_0 = 01|10|11 ausgewählt und konjunktiv mit dem Taktsignal verknüpft.



Die Ausgangsspannung Y hat die Form



Streng genommen ist ein BRM kein Automat, wie wir ihn bisher entworfen haben, da hier das Taktsignal in die Ergebnisfunktion eingeht! Wir zerlegen den BRM in zwei Teile, einen "sauberen" Automaten und die Verknüpfung des Ausgangssignals dieses Automaten mit dem Taktsignal. Der "saubere" Automat wird wie bekannt entworfen.



7. Ein einfacher von-Neumann-Rechner

In der noch verbleibenden Zeit werden wir einen einfachen von-Neumann-Rechner - den LC1 - kennenlernen. In der ebenfalls in diesem Semester von mir für die Studiengänge Ergänzungsstudium Lehramt Informatik und Wirtschaftsinformatik gehaltenen Vorlesung Rechnerorganisation/Rechnerarchitektur wird der LC1 sehr breit behandelt. Im Internetangebot der Professur Technische Informatik finden Sie umfangreiche Materialien dazu, auf die ich zurückgreife.

Wir werden hier einen etwas anderen Schwerpunkt wählen. Wir wollen uns anschauen, wo und mit welcher Funktion die bisher behandelten digitalen Systeme in einem Digitalrechner eingesetzt werden.