Lerntext Informatik I - Technische Grundlagen der Informatik

Mladen Ivkovic
mladen.ivkovic@hispeed.ch

Herbstsemester 2014

Anmerkung des Autoren

- Dieses Dokument dient zur Präsenzübungsvorbereitung für meine Kommilitonen und Kommilitoninnen. Ich habe mich stark an der Vorlesung, gehalten von Prof. Dr. Burkhart Stiller, und den Vorlesungsfolien orientiert. Jegliche nicht-kommerzielle Weitergabe ist von meiner Seite her gestattet.
- Dieses Dokument ist nach bestem Wissen und Gewissen erstellt. Unvollständigkeiten und Fehler sind möglich und wahrscheinlich. (Grammatik und Orthographie sowieso. lol)
- Alle Abbildungen stammen aus den Vorlesungsfolien, falls nicht anders vermerkt.
- Dieses Dokument beinhaltet nicht den gesamten Stoff der Vorlesung, insofern könnte der an der Prüfung(en) abgefrage Stoff nicht Teil dieses Dokumentes sein. Ich habe nach bestem Wissen und Gewissen den Weizen von der Spreu getrennt.

1 Einleitung und Entwicklung

- '**Informatik**' ist eine Begriffsverschmelzung aus 'Information' und 'Automatik'. Sie umfasst allgemein die automatisierte Informationsverarbeitung in Natur, Technik und Gesellschaft. Anfangs waren hauptsächlich Rechenmaschinen zur Zahlenverarbeitung gefragt, heutige Maschinen verarbeiten beliebige Informationen.

- Ein **Algorithmus** ist eine Verarbeitungsvorschrift, die von einer Maschine oder auch von einem Menschen durchgeführt werden kann. Aus der Präzision der sprachlichen Darstellung des Algorithmus' muss die Abfolge der einzelnen Verarbeitungsschritte eindeutig hervorgehen.

- Mitte der 1940er Jahre entwickelte John von Neumann die **Fundamentalprinzipien** einer Rechenanlage. (Siehe Abb. 1.1) Es beinhaltet ein Rechenwert, ein Steuerwerk, Ein- und Ausgabe und Verbindungen. Programme und Daten sind im Speicher. Die Bearbeitung von Befehlen erfolgt Schritt für Schritt. Es hat bedingte Sprünge und Verzweigungen.

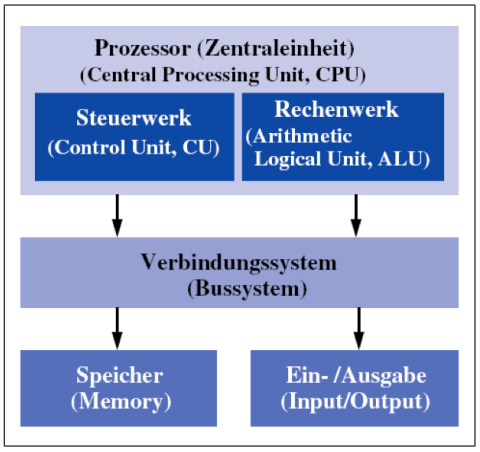

Abb. 1.1: Fundamentalprinzip einer Rechenanlage

- Die erste Generation von Rechnern waren Röhrenrechner, die zweite Transistorrechner, die dritte und vierte Generation basierten auf Mikrochips mit hoch- und höchstintegrierten Schaltkreisen. Die Leistung steigt, während der Stromverbrauch sinkt.

- **'Mooresches Gesetz'**: Die Anzahl der Transistoren pro (Prozessor-)Chip verdoppelt sich alle zwei Jahre. Die Verarbeitungsleistung der Hochleistungsprozessoren verdoppelt sich alle 18 Monate. Für den gleichen Preis liefert die Mikroelektronik die doppelte Leistung in weniger als zwei Jahren.

- **Rechnerstrukturen** legen die Grundlagen für den Bereich der Technischen Informatik (Hardware) und deckten den Bereich von der Logik zu einfachsten CPUs ab. **Rechnerorganisation** (und deren Nutzung) zeigt Systemarchitekturen (wie) und Befehlssätze (was). Sie umfasst Betriebssystemfunktionen, Organisation von Computern und Kommunikation.

2 Rechnerarithmetik

- Die **Rechnerarithmetik** soll als Beispiel eingeführt werden, wie grössere Informationseinheiten elektronisch verarbeitet werden. Sie bildet gleichzeitig die entscheidende Grundlage, beliebige Informationen und Informationseinheiten zu behandeln (berechnen, umwandeln, speichern, kommunizieren).

- Die gängigste Form der **Zahlensysteme** sind **Stellenwertsysteme**. Die Zahlen werden in Form einer Reihe von Ziffern z_i mit dazugehöriger Potenz der Basis b^i dargestellt. Der Wert der Zahl ergibt sich dann als Summe der Werte aller Einzelstellen $z_i b^i$.

- **Umrechnung** in andere Zahlensysteme: Gegeben sei Zahl Z, umzuwandeln in System mit Basis b.

  Euklidisches System: Suche Potenz p, so dass $b^p \leq Z < b^{p+1}$. Dividiere Z durch b^p. Der Rest der Division ist die erste (vorderste) Ziffer, den Quotienten wieder durch b^{p-1} dividieren, biss alle Stellen durch sind. (Nach Komma: p wird negativ, also wird mit b^{-p} multipliziert.)

  Horner Schema: Dividiere Z durch b. Der Rest dieser Division ist die letzte Stelle (Einerstelle). Wiederhole Divisionen, bis kein Rest mehr.

- Das Dualsystem ist das wichtigste Zahlensystem im Rechner. Oktal- und Hexadezimalsystem werden benutzt, da sie leicht ins Dualsystem umwandelbar und besser zu verstehen als lange 0-1-Kolonnen sind. Eine einzelne Binärstelle (0 oder 1), die ein Rechner speichert, wird als **Bit** (Binary digit) bezeichnet.

- Für die Darstellung negativer Zahlen in Rechnern werden vier verschiedene Formate benutzt:
  - **Darstellung mit Betrag und Vorzeichen:** Das am weitesten links stehende Bit (MSB, most significant bit) wird als Vorzeichenbit benutzt. Z.B. 0 = positive Zahl, 1 = negative Zahl. Die Nachteile sind, dass bei Addition und Subtraktion die Vorzeichen der Operanden gesondert betrachtet werden müssen und dass es zwei Repräsentationen der Zahl 0 (also ± 0) gibt.
  - **Stellenkomplement/Einerkomplement**: Stellenkomplement der entsprechenden positiven Zahl. **Um eine Zahl zu negieren**, wird jedes Bit der Zahl komplementiert, also Nullen in Einsen und umgekehrt verwandelt. Negative Zahlen sind wiederum durch ein gesetztes Bit in der ersten Stelle charakterisiert. Zwar muss die erste Stelle bei Addition/Subtraktion nicht gesondert betrachtet werden, aber es gibt weiterhin zwei Darstellungen der Null.

  - **Zweierkomplement-Darstellung**: Man addiert nach der Stellenkomplementierung noch eine 1. Das hat eine eindeutige Null, aber der Zahlenbereich ist unsymmetrisch.
  - **Offset-Dual- (Exzess)-Darstellung**: Die Darstellung einer Zahl erfolgt in Form ihrer Charakteristik (bias). Der gesamte Zahlenbereich wird durch Addition einer Konstanten (Exzess, Offset) so nach oben verschoben, dass die kleinste (negative) Zahl die Darstellung 0...0 erhält. Der Zahlenbereich ist hier auch asymmetrisch.

- Für **Fest- und Gleitkommazahlen** braucht es spezielle Vereinbarungen für die Darstellung von Vorzeichen und Komma im Rechner. Für die Darstellung des Kommas gibt es zwei Möglichkeiten:
  - Bei der **Festkommadarstellung** sitzt das Komma innerhalb des Maschinenwortes, das eine Dualzahl enthalten soll, an einer festen Stelle. Meist setzt man das Komma hinter die letzte Stelle. Andere Zahlen können durch entsprechende massstabsfaktoren in die gewählte Darstellungsform überführt werden. Negative Zahlen: Meist Zweierkomplement-Darstellung. Die Festkomma-Darstellungen werden heute hardwareseitig nicht mehr verwendet, jedoch bei der Ein- oder Ausgabe.
  - Zur Darstellung von Zahlen, die betragsmässig sehr gross oder sehr klein sind, verwendet man die **Gleitkommadarstellung**. Sie entspricht einer halblogarithmischen Form $X = \pm \text{Mantisse} \cdot b^{Exponent-bias}$.

    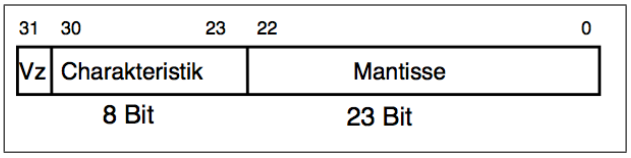

    Abb. 2.1: Gleitkommazahl nach IEEE Standard

    Die Basis ist fest und braucht nicht mehr explizit repräsentiert zu werden. Werden mit Betrag und Vorzeichen dargestellt. Bei der Mantisse ist die Lage des Kommas durch eine Vereinbarung festgelegt. Der Exponent ist eine Ganze Zahl, in in Form ihrer Charakteristik dargestellt wird. Für Charakteristik und Mantisse wird im Rechner eine feste Anzahl von Speicherstellen festgelegt.
  - Die Differenz zweier aufeinanderfolgenden Zahlen wächst bei Gleitkomma-Zahlen exponentiell mit der Grösse der Zahlen, während sie bei Festkomma-Zahlen konstant ist. Damit ergibt sich eine hohe Ungenauigkeit bei der Darstellung grosser Zahlen. Die Gesetzmässigkeiten, die für reelle Zahlen gelten, werden **für Maschinendarstellungen verletzt**. Da beliebig viele Möglichkeiten existieren, unterschiedliche Gleitkommaformate zu definieren, wurde eine Normierung erforderlich: Der **IEEE-P 754-Floating-Point-Standard**. Dieser definiert mehrere Darstellungsformen: single mit 32 Bit, double mit 64 Bit und extended mit 80 Bit. Siehe Abb. 2.1 für single-Darstellung.

- **Konversion** Dezimalzahl - IEEE754 single: Zahl z_{10} ins Binärsystem konvertieren. (Ziffern vor dem Komma durch 2 teilen, nach dem Komma mit 2 Multiplizieren.)

Das Vorzeichenbit ist 1 falls negativ, 0 falls positiv. Der *bias* für singles ist 8 bit lang: 01111111, d.h. unsere momentane Zahl wäre $VZ \cdot 2^{exponent-01111111} \cdot z_2$. Der anfängliche Exponent ist genau gleich wie der *bias*, da unsere Zahl $z_2 = z_2 \cdot 1 = z_2 \cdot 2^0 = z_2 \cdot 2^{exp-bias}$, weswegen gelten muss $exp = bias$. Erhöhen wir den Exponenten um 1, verschiebt sich die Kommastelle um eine Stelle nach links. Wir erhöhen den Exponenten nun so lange, bis das Komma hinter der ersten Stelle (einer 1) ist. Diese 1 wird nicht geschrieben, da sie für alle so dargestellten Zahlen immer dort ist.

- Für die duale Addition gilt allgemein:

0 + 0	= 0
0 + 1	= 1
1 + 0	= 1
1 + 1	= 0 Übertrag 1
1 + 1 + 1 (vom Übertrag)	= 1 Übertrag 1

- Subtraktion von Dualzahlen basiert auf der Addition von Festkomma-Dualzahlen, denn $a - b = a + (-b)$.

- Ganzzahlige **Multiplikation** bzw. **Division** wird im Rechner allgemein mittels wiederholter Addition durchgeführt. Bei Zahlen, die mit Betrag und Vorzeichen dargestellt sind, ergeben sich keine Probleme: Beträge werden miteinander multipliziert, Vorzeichen des Produktes wird nach der Regel sign$(x \cdot y)$ =sign(x)XOR sign(y) bestimmt. Vorzeichenbehaftete Zahlen können grundsätzlich in die Darstellung mit Vorzeichen und Betrag gebracht werden. Zur Multiplikation von Gleitkommazahlen muss man Mantissen beider Zahlen multiplizieren und ihre Exponenten addieren. Die Division von Dualzahlen folgt denselben Prinzipien. Bei Division durch Null muss ein Ausnahmezustand erkannt und an den Prozessor weitergemeldet werden.

  Schriftliche Additionen, Subtraktionen, Multiplikationen und Divisionen funktionieren in anderen Zahlensystemen genau wie im Dezimalsystem, nur ist die 'Obergrenze' entsprechend anders gesetzt.

- **Zeichendarstellung**:
  - **ASCII Code**: Klein-/Grossbuchstaben, arabische Ziffern, Sonderzeichen. Kodierung erfolgt in einem Byte (8Bits) → 128 mögliche Zeichen, da erstes Bit nicht genutzt wird. Zur Speicherung von Texten werden einzelne Bytes einfach hintereinander abgespeichert, so dass man einen String erhält. Ende des Strings: Länge der Zeichenkette in den ersten Bytes vor dem String abgespeichert, oder Ende wird durch ein besonderes, nicht darzustellendes Zeichen gekennzeichnet.
  - **UniCode**: Heute 4-Byte-Schema, praktisch alle bekannten Schriftkulturen und Zeichensysteme kodierbar. Die Formate für Speicherung und Übertragung sind unterschiedlich.

3 Schaltnetze

3.1 Formale Grundlagen logischer Beschreibungen

- Schaltnetze sind rein kombinatorische logische Schaltungen. Es sind logische Funktionen, welche kein Speicherverhalten besitzen. Zu deren Untersuchung und Beschreibung der Eigenschaften und des Verhaltens ist die Boolesche Algebra hervorragend geeignet.

- Als **Boolesche Algebra** bezeichnet man eine abgeschlossene Menge V ($\forall y, b \in V$: $a \otimes b \in V$, $a \oplus b \in V$), auf der für zwei zweistellige Operationen $\oplus$ und $\otimes$ das Kommutativgesetz und das Distributivgesetz gelten und ein Neutrales und Inverses Element existieren (Huntingtonsche Axiome):

$$\begin{aligned}
\text{Kommutativgesetz: } & a \otimes b = b \otimes a \\
& a \oplus b = b \oplus a \\
\text{Distributivgesetz: } & a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes b) \\
& a \oplus (b \otimes c) = (a \oplus b) \otimes (a \oplus c) \\
\text{Neutrales Element } & a \otimes e = a \\
& a \oplus n = a \\
\text{Inverses Element } & a \otimes \bar{a} = n \\
& a \oplus \bar{a} = e
\end{aligned}$$

- In der Schaltalgebra entspricht die Menge V $0, 1$, die Addition (bisher: $\oplus$) $\vee$ bzw. $+$, (ODER) mit dem neutralen Element 0, die Multiplikation (bisher: $\otimes$) $\wedge$ bzw. & (UND) mit dem neutralen Element 1. Das inverse Element von a wird weiterhin mit $\overline{a}$ bezeichnet.

  Kleine Übersichtstabelle[1]:

Disjunktion	$\oplus$	$+$	$\vee$	ODER
Konjunktion	$\otimes$	&	$\wedge$	UND

[1] Kleine, etwas weit hergeholte Merkhilfe: "DOV und AUK". **D**isjunktion **O**der $\vee$ und $\wedge$ **U**nd **K**onjution

Aus den Huntingtonschen Axiomen lassen sich weitere Sätze ableiten:

$$
\begin{aligned}
\text{Assoziativgesetz } & (a \wedge b) \wedge c = a \wedge (b \wedge c) \\
& (a \vee b) \vee c = a \vee (b \vee c) \\
\text{Idempotenzgesetz } & a \wedge a = a \\
& a \vee a = a \\
\text{Absorptionsgesetz } & a \wedge (a \vee b) = a \\
& a \vee (a \wedge b) = a \\
\text{DeMorgan-Gesetz } & \overline{a \wedge b} = \bar{a} \vee \bar{b} \text{ (NAND)} \\
& \overline{a \vee b} = \bar{a} \wedge \bar{b} \text{ (NOR)}
\end{aligned}
$$

- Verknüpfungen:
  UND: $= 1$ falls $a = b = 1$, sonst $= 0$. (Gibt nur 1 raus, falls a und b 1 sind.)
  ODER: $= 0$ falls $a = b = 0$, sonst $= 1$. (Gibt 1 raus, falls a oder b oder beide 1 sind.) Das Inverse von 0 ist 1 und umgekehrt.

- In der Schaltalgebra verwendet man manchmal auch die Bezeichnung **Wahrheitswerte** für 0 und 1: 0 = falsch, 1 = wahr. Ein Boolescher Ausdruck hat i.d.R. zunächst keinen Wahrheitswert, da er binäre Variablen enthalten kann. Durch die Belegung der binären Variablen mit Wahrheitswerten (=Interpretation) erhält der Boolesche Ausdruck einen Wahrheitswert. Ein Boolescher Ausdruck, bei dem alle Belegungen der Variablen mit Wahrheitswerten zu 'wahr' führen, heisst **Tautologie.**

- Eine n-stellige Boolesche Funktion ordnet jeder möglichen Wahrheitswertbelegung eines Tupels von Variablen $(x_1, ..., x_n)$ genau einen Wahrheitswert zu: $f : 0, 1^n \rightarrow \{0, 1\}$. Die Darstellung boolescher Funktionen erfolgt durch eine Funktionstabelle, einen algebraischen Ausdruck oder durch einen Graphen. Für eine Tabelle 16 möglicher zweistelliger boolescher Funktionen, siehe Abb. 7.1.

- Ein System von Operatoren, mit dem alle booleschen Funktionen dargestellt werden können, heisst vollständiges Operatorensystem. Die Operatoren $(\wedge, \vee, ^-)$ bilden ein vollständiges Operatorensystem. Für weitere vollständige Operatorensysteme, siehe Abb. 7.2

3.2 Normal- und Minimalformen

- Eine boolesche Funktion kann durch verschiedene Boolesche Ausdrücke beschrieben werden. Eine Standarddarstellung boolescher Funktionen im vollständigen Operatorensystem $(\wedge, \vee, ^-)$ ist die **konjunktive (KNF)** und **disjunktive Normalform (DNF)**.

- Def **Literal**: $L_i \in \{x_i, \overline{x_i}\}$
  Def: Ein Produktterm $K(x_1, ..., x_m)$ ist die Konjunktion von Literalen $\bigwedge_{i \in 1,...,m} L_i = L_1 \wedge ... \wedge L_m$
  Jeder Produktterm kann so dargestellt werden, dass eine Variable x in höchstens einem Literal vorkommt, da $(x \wedge x) = x$ und $(x \wedge \overline{x}) = 0 \rightarrow K(x_1, ..., x_m) = 0$

- Ein Produktterm $K(x_1, ..., x_n)$ heisst **Implikant** einer Booleschen Funktion $y(x_1, ..., x_n)$, wenn $K \rightarrow y$, d.h. für jede Belegung $B \in \{0, 1\}^n$ gilt: Wenn $K(B) = 1 \Rightarrow y(B) = 1$. D.h. $K(B)$ ist ein Implikant, wenn jede Belegung von n Variablen genau dann eine 1 liefert, wenn $y(B)$ auch eine 1 liefert, aber nicht zwangsweise umgekehrt. Ein Implikant einer Booleschen Funktion y heisst **Minterm**, wenn ein Literal jeder Variablen x_i der Funktion y im Implikanten genau einmal vorkommt, sprich jede Variable oder ihre Inverse muss im Produktterm auftreten.

- Es sei eine Boolesche Funktion $y(x_1, ..., x_n)$ gegeben. Ein boolescher Ausdruck heisst **Disjunktive Normalform** der Funktion y, wenn er aus einer disjuntkiven Verknüpfung aller Minterme K_i besteht: $y = K_0 \vee K_1 \vee ... \vee K_k, k = 2^n - 1$. Es darf dabei keine zwei Konjunktionen $K_i, K_j, i \neq j$ geben, die zueinander äquivalent sind.

- Sei $D(x_1, ..., x_m)$ eine Disjunktion von Literalen $\bigvee^{i \in 1,...,m} L_i = L_1 \vee ... \vee L_m$. Der Term $D(x_1, ..., x_m)$ heisst **Implikat** einer Booleschen Funktion $y(x_1, ..., x_m)$, wenn $\overline{D} \rightarrow \overline{y}$, d.h. für jede Belegung $B \in 0, 1^n$ gilt: Wenn $D(B) = 0 \Rightarrow y(B) = 0$. D.h. $D(B)$ ist Implikat von $y(B)$, wenn jede Belegung eine 0 liefert, wenn auch y eine 0 liefert. Ein Implikat einer Booleschen Funktion $y(x_1, ..., x_m)$ heisst **Maxterm**, wenn ein Literal jeder Variable x_i der Funktion y im Implikaten genau einmal vorkommt.

- Sei eine Boolesche Funktion $y(x_1, ..., x_m)$ gegeben. Ein Boolescher Ausdruck heisst **konjunktive Normalform**, wenn er aus einer konjunktiven Verknüpfung aller Maxterme D_i besteht: $y = D_0 \wedge D_1 \wedge ... \wedge D_k, k = 2^m - 1$. Es darf dabei keine zwei Disjunktionen $D_i, D_j, i \neq j$ geben, die zueinander äquivalent sind.

- Jeder Minterm einer DNF entspricht einer Zeile in der Funktionstabelle, die den Funktionswert 1 liefert. Jeder Maxterm einer KNF entspricht einer Zeile in der Funktionstabelle, die den Funktionswert 0 liefert. Die DNF und KNF sind eindeutige Darstellungen. In einer Funktion mit n Variablen können bis zu 2^n Minterme bzw. Maxterme auftreten. Um eine Funktion zu beschreiben, reicht die Angabe aller Minterme (oder aller Maxterme) aus.

- Überführung von Funktionen aus DF bzw. KF in DNF bzw. KNF:

  - Shannonscher Entwicklungssatz:

    Zuerst nach der Variable x_i entwickeln: Die Variable wird in der Funktion auf 1 gesetzt und der entstehende Term konjunktiv mit x_i verknüpft. Der gesamte

Term wird $\vee$-verknüpft mit der Funktion, in welcher x_i auf 0 gesetzt wurde und dieser Term wird konjunktiv mit $\overline{x_i}$ verknüpft.

$$y = f(x_1, ..., x_n) =$$
$$= [x_i \wedge f(x_1, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)] \vee [\overline{x_i} \wedge f(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n)]$$

Schlussendlich ist es das Ziel, einen Ausdruck von Mintermen zu erhalten, damit es eine Normalform ist. Also muss in jedem "Term"der Funktion jede Variable oder ihre Inverse vorkommen. Z.T. muss man erweitern mit $b = b \wedge 1$ und $1 = c \vee \bar{c}$. Nachdem die Funktion nach allen Variablen entwickelt wurde, können die Minterme durch Verfolgen der Äste des Baums gefunden werden, die zu einer 1 führen.

– Aus Funktionentabelle:

DNF: Aus der Funktionstabelle einer Funktion erhält man die Minterme, indem man in allen Zeilen mit dem Funktionswert 1 jeweils alle Eingangsvariablen mit $\wedge$ verknüpft und dabei Eingangsvariablen mit dem Wert 0 negiert. Durch die disjunktive[2] Verknüpfung dieser Minterme kann ein Boolescher Funktionsausdruck in DNF hergeleitet werden.

KNF: Aus der Funktionstabelle einer Funktion erhält man die Maxterme, indem man in allen Zeilen mit dem Funktionswert 0 jeweils alle Eingangsvariablen mit $\vee$ verknüpft und dabei Eingangsvariablen mit dem Wert 1 negiert. Durch die konjunktive Verknüpfung dieser Maxterme kann ein Boolescher Funktionsausdruck in KNF hergeleitet werden.

- Ähnlich zum Aufbau der disjunktiven und konjunktiven Normalform gibt es eine disjunktive (DMF) und konjunktive (KMF) Minimalform. Das Ziel ist, möglichst kurze Boolesche Ausdrücke für eine gegebene Boolesche Funktion zu erhalten. Minimalformen sind nicht eindeutig. Bei Minimierungsverfahren geht man in zwei Schritten vor: zuerst wird eine Menge von Implikanten bzw. Implikate der Funktion y mit einer möglichst geringen Anzahl von Literalen gebildet. Aus dieser Menge wird eine möglichst geringe Anzahl von Implikanten bzw. Implikate herausgesucht, deren Disjunktion bzw. Konjuktion die Funktion y ergeben. Oft können nur suboptimale Lösungen unter Verwendung von Heuristiken gefunden werden.

- **NAND/NOR Konversionen:** Das ($\overline{\wedge}$)-System und das ($\overline{\vee}$)-System sind vollständige Operatorensysteme - beliebige disjunktive und konjunktive Ausdrücke können mit NAND- und NOR-Verknüpfungen dargestellt werden. Diese Überführungen sind wichtig, da sie eine einfache Implementierung in die Hardware besitzen. Die Überführung geschieht durch Doppelte Negation und anschliessende Anwendung der DeMorganschen Regeln. Ein Problem dabei ist, dass die Operatoren $\overline{\wedge}$ und $\overline{\vee}$ nicht assoziativ sind. Beispielrechnung siehe Abb. 7.3.

[2]Eselsbrücke für Konjunktion/Disjunktion: Alphabetisch. **d**isjunktion = **o**der, **k**onjunktion = **u**nd.

3.3 Realisierung von Schaltnetzen auf Schalter- und Gatterebene

- Auf der **Gatterebene** (welche über der Schalterebene, wo Transistoren als Schalter benutzt werden, und über der Layoutebene, welche auf Transistortechnologie basiert, liegt) sind die logischen Gatter wie UND, ODER, NAND etc. Es ist eine Abstraktion von der internen Realisierung der Verknüpfungsbausteine. Es wird auf das logische Verhalten beschränkt, die logischen Funktionen werden auf Schaltungen ohne tiefergehende elektrotechnische Kenntnisse abgebildet. Die Verknüpfungsbausteine. werden durch Schaltsymbole dargestellt (Sie Abb. 7.4). Verknüpfungsbausteine dieser Art werden **Gatter** genannt.

- Aus einfacheren Gattern lassen sich hierarchisch komplexere Gatter aufbauen, die teilweise eigene Symbole besitzen, wie das Antivalenz-Gatter oder das Äquivalenz-Gatter.

3.4 Entwurf von Schaltnetzen

- Der praktische Entwurf von Schaltnetzen muss beachten, dass reale Gatter keine idealen Verknüpfungen sind, sondern z.B. Wärme abgeben; dass reale Gatter Platz benötigen und dass sie Verzögerungszeiten besitzen. Technische Kriterien wären Leistungsaufnahme, Schaltzeit, Platzbedarf, Material, Lebensdauer etc., Korrekte Realisierung und Berücksichtigung technischer Beschränkungen. Ökonomische Kriterien wären geringe Kosten für den Entwurf und für die Realisierung. Das Ziel des Entwurfs ist es, unter Einhaltung bestimmter technischer Kriterien vor Allem einen günstigen Kompromiss anzustreben, um so zu einem Minimum an Gesamtkosten zu gelangen.

- Die Darstellungsform, deren Realisierung die geringsten Kosten verursacht, bezeichnet man als **Minimalform**. Der Vorgang der Erzeugung einer Minimalform nennt man **Minimierung.** Es gibt algebraische Verfahren, Graphische Verfahren und Tabellarische Verfahren. Algebraische und graphische Verfahren eignen sich nur für bis zu 5-6 Variablen, danach werden sie zu unübersichtlich.

- Das **KV-Diagramm** (nach Karnaugh und Veith) ist ein graphisches Verfahren. Ausgangspunkt ist ein Rechteck, dessen rechte Hälfte der variablen a und dessen linke Hälfte $\bar{a}$ zugeordnet wird. Die Zahl in den Feldern gibt den Index der Variablenbelegung an. Durch Eintragen der Wahrheitswerte 0 oder 1 in die Felder des KV-Diagramms wird eine Boolesche Funktion charakterisiert. Das KV-Diagramm für mehrere Variablen erhält man durch Spiegelung des bestehenden Rechtecks. Jedes Feld hat eine eindeutige Variablenzuordnung, die an den Rändern abgelesen werden kann. In Abb. 7.5 hat das Feld 11 die Zuordnung $ab\bar{c}d$.

  Ist die Funktion in Tabellenform gegeben, so entspricht jede Zeile der Funktionstabelle einem Feld im KV-Diagramm und für jede Zeile muss das zugehörige

Feld gesucht und der entsprechende Wert eingetragen werden. Trick: Schreibt man die Variablen in umgekehrter alphabetischer Reihenfolge in die Funktionentabelle, kann man von oben nach unten die Werte in die Plätze im KV-Diagramm der Reihe nach beim richtigen Index eintragen.

Symmetrisch zu einer Achse liegende Minterme (1sen) in einem KV-Diagramm unterscheiden sich lediglich in einer Variablen. Terme, die sich nur in einer Variablen unterscheiden, lassen sich zusammenfassen. Es entsteht ein Term ohne diese Variable. Aus dem KV-Diagramm können Primimplikanten herausgelesen werden. Man versucht, möglichst grosse Blöcke von Einsen im Diagramm zu finden, wobei jeder Einsblock 2^k Felder umfassen muss. Ein **Primimplikant** p ist ein Implikant, falls es keinen Implikanten q gibt, der von p impliziert wird. Mit anderen Worten, p muss 1sen erhalten, welche q nicht hat. p ist also von grösstmöglicher Ordnung, umfasst einen maximal grossen Einsblock. Jede Funktion ist als Disjunktion ihrer Primimplikanten darstellbar.

- Zur Implementation regelmässig wiederkehrender Funktionen seien hier zwei Versionen betrachtet:

  – Ein **Multiplexer** (Abkürzung: MUX) ist ein Baustein mit mehreren Eingängen und einem Ausgang, wobei über n Steuerleitungen einer der 2^n Eingänge auf den Ausgang geschaltet wird. Multiplexer werden nach ihrer Grösse als $2^n : 1$ - Multiplexer (alternativ als 1-aus-2^n- Multiplexer) klassifiziert. Man kann mit einem $2^n : 1$ - Multiplexer eine logische Funktion mit $n+1$ Variablen implementieren. Hierzu wird die sog. Implementierungstabelle verwendet. Die Tabelle besteht aus 2^n Spalten für die möglichen Belegungen der n Steuereingänge und 2 Zeilen für die negierte und nicht negierte $(n + 1)$-te Variable. In die Tabelle werden die Funktionswerte in Abhängigkeit von den Variablen eingetragen.

    Der zum Multiplexer korrespondierende Baustein, der einen Eingang abhängig von n Steuerleitungen auf einen von 2^nAusgängen schaltet, heisst Demultiplexer, auch Dekoder genannt.

- Bei den bisher behandelten Bausteinen (Gatter, Multiplexer, Dekoder) war die Funktion fest vorgegeben. (festverdrahtete Logik). Mittels mikroprogrammierter Logik auf Speichern lässt sich Verknüpfungsbausteine anpassen (programmieren). Durch das Anlegen von Eingangssignalen wird eine Speicherzelle ausgewählt (adressiert) und der dort gespeicherte Funktionswert an den Ausgängen zur Verfügung gestellt. Das Speichern einer 1 für eine bestimmte Ausgangsvariable i bedeutet, dass dieser Minterm in die ODER-Verknüpfung am i-ten Ausgang einbezogen wird, eine 0 heisst, dass der Minterm nicht benutzt wird. Je nach Personalisierung des Speicherbausteins unterscheidet man verschiedene Speichertypen:

  ROM (Read Only Memory) sind Speicherbausteine, auf die nur lesend zugegriffen werden kann. Programmierung wird während der ganzen Lebenszeit des Bausteins beibehalten.

PROM (Programmable Read Only Memory) sind programmierbare ROMs, die erst vom Benutzer programmiert werden.

EPROM (Erasable Programmable Read Only Memory) sind benutzerprogrammierbare Speicherbausteine, welche durch UV-Licht wieder löschbar und dann neu programmierbar sind.

RAM (Random Access Memory) sind Speicher, auf die während des Normalbetriebs lesend und schreibend zugegriffen werden kann. Ein RAM-Baustein verliert seine Programmierung, wenn er von der Spannungsversorgung abgetrennt wird. Man unterscheidet dynamische- (DRAM) und statische (SRAM) RAM-Bausteine. Bei dynamischen RAM dient ein Kondensator als Ladungsspeicher und ein Transistor wird zum Ankoppeln an diesen Ladungsspeicher benötigt. Der Speicherinhalt muss in regelmässigen Zeitabständen 'aufgefrischt' werden (memory refresh). Bei statischen RAM wird als Speicherzelle ein Flipflop verwendet. Sie hält ihre Programmierung auch ohne Regeneration und die Zugriffszeit ist wesentlich kürzer als bei einem dynamischen RAM, hingegen benötigt sie etwa 6 bis 8 Transistoren, eine dynamische dagegen deutlich weniger (z.B. ein Transistor).

Bei **PLA (Programmable Logic Array)** benutzt man statt der Normalform die disjunktive Minimalform, womit sich Funktionen viel kompakter darstellen lassen. Im Unterschied zum ROM werden bei PLA eingangsseitig nicht Minterme, sondern Primimplikanten der Minimalüberdeckung erzeugt.

3.5 Laufzeiteffekte bei Schaltnetzen

- Auf der Gatterebene wurden die Gatter bisher als ideale logische Verknüpfungen betrachtet. In der Realität werden Gatter jedoch z.B. mittels Transistoren, Widerstände, Kapazitäten realisiert (Layoutebene). Der zeitliche Signal-Verlauf eines realen Gatters weicht vom Verlauf der idealen booleschen Grössen ab.

- Bei einem **idealen Schaltnetz** ändert sich das Ausgangssignal nicht, wenn die alte und neue Belegung denselben logischen Verknüpfungswert liefern. Wenn die alte und neue Belegung verschiedene logische Verknüpfungswerte liefern, ändert sich das Ausgangssignal genau einmal.

  Bei einem **realen Schaltnetz** läuft die Änderung auf verschieden langen Wegen mit verschiedenen Verzögerungen durch das Schaltnetz. Mehrfache Änderungen des Ausgangssignals sind möglich, bis sich der stabile Endwert einstellt (**Hasardfehler**).

- Um die Effekte auf der Gatterebene annähernd zu beschreiben, gibt es eine Reihe verschieden komplexer Modelle. Ein einfaches Modell ist das **Totzeitmodell**: Es werden lediglich die durch Gatter und Leitungen entstehenden Totzeiten berücksichtigt. Ein reales Verküpfungsglied (Gatter) wird modelliert durch ein ideales Verknüpfungsglied ohne Verzögerungsanteil und ein Totzeitglied als reines Verzögerungsglied (steht für die Schaltzeit des Gatters und ggf. für Leitungsverzögerungen). Das zeitliche Verhalten einer binären Grösse hinter einem Totzeitglied ist

dasselbe wie dasjenige vor dem Totzeitglied, aber um die Zeit τ versetzt. Mit Hilfe dieses einfachen Modells lassen sich Laufzeiteffekte bereits sehr gut modellieren, auch wenn dieses Modell noch sehr idealisierend ist.

- Ein **Eingabewechsel** ist die Änderung einer oder mehrerer Eingangsvariablen zu einem bestimmten Zeitpunkt. Falls sich mehrere Eingangsvariablen ändern sollen, so müssen sie dies gleichzeitig tun. Ein **Übergang** ist der Vorgang im Schaltnetz, der vom Eingabewechsel ausgelöst wird. Er beginnt mit dem Eingabewechsel und endet mit dem Eintreten des neuen Ruhezustandes.

- Ein **Hasardfehler** ist eine mehrmalige Änderung der Ausgangsvariablen während eines Übergangs. Ein **Hasard** ist die durch das Schaltnetz gegebene logisch-strukturelle Vorbedingung für einen Hasardfehler, ohne Berücksichtigung der konkreten Verzögerungswerte. Jeder Hasard ist eine Eigenschaft eines bestimmten Überganges im Schaltnetz. Zur Betrachtung, ob ein bestimmter Übergang hasardbehaftet ist oder nicht, interessiert nur die logische Funktion, die durch das Schaltnetz realisiert wird und die Struktur des Schaltnetzes, nicht jedoch die tatsächlichen Verzögerungswerte der verwendeten Gatter. Tritt in einem konkreten Schaltnetz bei einem bestimmten Übergang ein Hasardfehler auf, so ist dieser Übergang hasardbehaftet. Die Umkehrung gilt jedoch nicht: Ist ein Übergang hasardbehaftet, so folgt hieraus nicht notwendigerweise das Eintreten eines Hasardfehlers.

  Ein **Funktionshasard** ist ein Hasard, dessen Ursache in der zu realisierenden Funktion liegt. Er tritt in jedem möglichen Schaltnetz für diese Funktion auf. Er kann nicht behoben werden. Für ein konkretes Schaltnetz mit Funktionshasard kann zwar der Funktionshasardfehler durch günstige Wahl der Verzögerungswerte behoben werden, nicht jedoch der Hasard selbst. Ein **Strukturhasard** ist ein Hasard, dessen Ursache in der Struktur des realisierten Schaltnetzes liegt. Ein Strukturhasard kann deshalb immer durch Änderung der Schaltnetzstruktur bei gleicher Schaltnetzfunktion behoben werden. Es ist grundsätzlich möglich, ein anderes Schaltnetz zu entwerfen, welches dieselbe Funktion realisiert und den Strukturhasard beseitigt.

4 Schaltwerke

4.1 Formale Grundlagen

- Bei **Schaltnetzen** (kombinatorische Schaltungen) hängt die Ausgabe lediglich von den Werten der Eingangsvariablen zum gleichen Zeitpunkt ab. Für diese Unterscheidung seien Laufzeitverzögerungen vernachlässigt. Bei **Schaltwerken** (sequentielle Schaltungen) hängen die Ausgabewerte auch von Belegungen der Eingangsvariablen zu vergangenen Zeitpunkten ab.

  Man fasst alle Abhängigkeiten von Werten der Vergangenheit in einem sogenannten **Zustand** zusammen. Das Schaltwerk erzeugt damit seine Ausgabe in Abhängigkeit von den augenblicklichen Eingangsvariablen und seinem Zustand; diese Grössen beeinflussen auch den nächsten Zustand des Schaltwerks.

- Ein 6-Tupel $M = (E, A, Z, \delta, \omega, z_0)$ heisst **Automat**, wenn E, A und Z nichtleere Mengen sind, wobei:
  - E: Menge der Eingangsbelegungen e
  - A: Menge der Ausgangsbelegungen a
  - Z: Menge der Zustände z.
  - δ: Überführungsfunktion. $\delta : Z \times E \to Z$

    δ ist eine auf der Menge $Z \times E$ definierte Funktion, deren Werte in Z liegen.
  - ω: Ausgabefunktion. $\omega : Z \times E \to A$

    ω ist eine auf der Menge $Z \times E$ definierte Funktion, deren Werte in A liegen.
  - z_0: Grundzustand

  Die Zustandsmenge Z ermöglicht die Speicherung von Wissen über Eingangsbelegungen der Vergangenheit. Die aktuelle Ausgabebelegung wird durch die Funktion ω, der neue Zustand durch die Funktion δ aus den aktuellen Eingangsbelegungen und dem alten Zustand erzeugt.

- Hängt der Ausgabewert lediglich vom augenblicklichen Zustand ab, spricht man in diesem Falle von einem **Moore-Automaten** (Moore-Schaltwerk). Die Ausgabebelegung ist unabhängig von der Eingabebelegung, sie kann sich nur nach einem Zustandswechsel ändern.

- Geht auch die Eingabebelegung in die Berechnung des Ausgabewertes ein, erhält man in diesem Falle einen **Mealy-Automaten** (Mealy-Schaltwerk). Die Ausgabewerte können sich sofort mit der Änderung von Eingabevariablen ändern.

Bei einem **Mealy-Automat 1. Art** wird zunächst aus der neu anliegenden Eingabebelegung die Ausgabebelegung gebildet und dann in den Folgezustand gewechselt. Bei einem **Mealy-Automat 2. Art** wird zunächst aus der neu anliegenden Eingabebelegung der Folgezustand und dann mit der noch anliegenden Eingabebelegung die Ausgabebelegung gebildet.

- Zur **formalisierten Beschreibung des Verhaltens** eines Automaten gibt es vier Möglichkeiten:
  - **Zeitdiagramm**: Es wird eine beispielhafte Folge von Eingabebelegungen dargestellt. Dient zur Veranschaulichung des Problems.
  - **Ablauftabelle**: Sind die Zustände bekannt, so kann die Überführungsfunktion $z_{k+1} = \delta(z_k, x)$ und die Ausgabefunktion $(y_0, y_1) = \omega(z_k, x)$ in einer Ablauftabelle dargestellt werden.
  - **Automatentabelle**
  - **Automatengraph**
- Zur **Speicherung** vergangener Informationen ist ein Zustandsspeicher erforderlich. Die einfachste Form dieses Zustandsspeichers ist Rückkopplung (siehe Abb. 7.6). Durch die Rückkopplung lassen sich die in den Eingangsvariablen nicht mehr repräsentierten Informationen wieder am Eingang zur Verfügung stellen. Als ideales Gatter betrachtet, ist die Schaltung unzulässig. In der Realität hat jede Schaltung eine Verzögerungszeit > 0. Somit hebt sich der Widerspruch $z^{neu} = \bar{z}$ (nach dem Gatter) und $z = z^{neu}$ (Rückkopplung) mit der Verzögerung Δ: $z^{neu}(t) = \bar{z}(t)$ und $(t + \Delta) = z^{neu}(t + \Delta)$. Das Δ-Verzögerungsglied entspricht in diesem Fall dem Speicher. Um ein von Verzögerungszeiten unabhängiges Verhalten zu erreichen, wird der Speicher so aufgebaut, dass z seinen Wert so lange beibehält, bis es explizit durch ein externes Signal T auf z^{neu} gesetzt wird.

4.2 Asynchrone Schaltwerke, Flip-Flops

- Werden alle Zustandsspeicher von einem oder mehreren zentralen Synchronisationssignal(en) **T (Takt)** gesteuert, so spricht man von einem **synchronen Schaltwerk**. Anderenfalls spricht man von einem **asynchronen Schaltwerk**. Die Synchronisation über einen Takt kann **flankengesteuert** und **pegelgesteuert** sein.
- Bei der **Pegelsteuerung** ist der Speicher während einer Takthälfte transparent, während der anderen speichert er. Die Eingänge wirken sich nur auf den Zustand aus, wenn der Takt z.B. den Wert 1 hat. Ist der Takt 0, wird der Zustand gespeichert. Nachteil: Die Eingangssignale können sich während der aktiven (transparenten) Taktperiode mehrfach ändern. Einfachste Realisierung: Konjunktive Verknüpfung jeder Eingangsvariablen mit dem Takt. Pegelgesteuerte Zustandsspeicher werden auch Latches genannt.

- Bei der **Flankensteuerung** wird nur während der positiven ($0 \rightarrow 1$) oder der negativen ($1 \rightarrow 0$) Taktflanke die Eingabewerte in den Speicher übernommen. Der Vorteil dabei ist, dass Eingänge nur für eine sehr kurze Zeitspanne gültig sein müssen (und nicht über eine ganze Takthälfte wie bei der Pegelsteuerung), somit sind die Auswertezeitpunkte definiert.

- Mittlere und grössere Schaltwerke werden fast immer als synchrone Schaltwerke entworfen, weil sie leichter zu analysieren und zu entwerfen sind, denn sie sind unabhängig von Verzögerungszeiten. Wird die Dauer des Taktes grösser als die maximale Verzögerungszeit im Schaltnetz gewählt, so haben sich die Ausgänge der Schaltnetze δ und ω stabilisiert, bevor sie sich auf z^{neu} auswirken. Zur Analyse und Synthese eines synchronen Schaltwerks muss man also die Schaltnetze δ und ω betrachten und die Schaltung kann an den Stellen aufgetrennt werden, an denen die Speicherelemente sitzen.

- Die in synchronen Schaltwerken benutzten Speicherbausteine sind selbst kleine asynchrone Schaltwerke. Immer schneller werdende Bausteine zwingen zu teilweise asynchronen Entwurfstechniken; Werden die Verzögerungszeiten der verwendeten Bausteine kleiner als die Signallaufzeiten auf der Schaltungsplatine/auf dem Chip (ca. 20-30 cm/ns), dann ist der Takt ist nicht länger synchron, da er die einzelnen Bausteine je nach Entfernung zu für die Bausteine unterschiedlichen wahrnehmbaren Zeitpunkten erreicht.

- **Asynchrone Schaltwerke** arbeiten ohne einen zentralen Takt: Sie reagieren sofort auf jede Änderung der Eingangs- und Zustandsvariablen und sind sehr störempfindlich, auch auf Hasardfehler in den Übergangs-Schaltnetzen. Hasardfehler können ebenfalls falsche Zustandsübergänge und Oszillationen verursachen. Zur weiteren Verringerung des Störrisikos arbeiten asynchrone Schaltwerke darüber hinaus meist im sogenannten **normal fundamental mode**. Hierbei darf sich maximal eine Eingangsvariable gleichzeitig ändern und ein Eingabewechsel kann erst dann erfolgen, wenn alle internen Änderungen abgeklungen sind.

- Synchrone Schaltwerke benötigen taktgesteuerte Zustandspeicher. Hierfür werden **Flipflops** verwendet. (Siehe Abb. 7.11) Beim Entwurf synchroner Schaltwerke sind Zustand und gewünschter Folgezustand bekannt. Gesucht sind die Werte der Ansteuervariablen der Flipflops. Diese lassen sich leicht mit Hilfe der sog. Ansteuertabelle eines Flipflops bestimmen. Diese gibt den Zustandsübergang eines Flipflops unter den verschiedenen Eingabebelegungen wieder.

- Bei einem **RS-Flipflop** (Siehe Abb. 7.7) soll der Eingang s den Speicher setzen (s=1 $\rightarrow$ Ausgang q=1). Der Eingang r soll den Speicher rücksetzen (r=1 $\rightarrow$ q=0) Zum Speichern: r und s beide $0 \rightarrow q$ behält letzten Wert. Verboten: r und s gleichzeitig $1 \rightarrow$ die Ausgänge p und q sind komplementär. Die Zustandsvariable q und ihre Negation $\bar{q}$ ($= p$) stehen am Ausgang zur Verfügung. Um das RS-Flipflop in einem synchronen Schaltwerk verwenden zu können, muss ein Taktsignal eingeführt

werden, welches die Änderung der Zustandsvariablen in der inaktiven Taktphase verhindert. Dieses ist leicht zu erreichen, indem man die beiden Eingänge durch je ein UND-Gatter mit diesem Takt verknüpft.

- Bei einem RS-Flipflop ist stets die Nebenbedingung ($r \wedge s = 0$) zu beachten. Führt man eine Eingangsvariable d bejaht zum S-Eingang und negiert zum R-Eingang, ist diese Bedingung stets erfüllt. Damit erhält man ein sogenanntes **D-Latch**. Der anliegende Eingabewert wird in allen Fällen als Flipflopzustand übernommen und einen Takt lang gespeichert. Das Eingangssignal wird um eine Taktperiode verzögert am Ausgang zur Verfügung gestellt. (Daher der Name D-Latch von 'to delay' = verzögern).

  D-Flipflops sind die am einfachsten zu realisierenden flankengesteuerten Speicherelemente. Sie sind wegen des geringen Flächenbedarfs die in integrierten Schaltungen am häufigsten verwendeten Speicherglieder. Im Schaltsymbol wird die Taktflankensteuerung durch ein Dreieck am Takteingang spezifiziert. Bei einer Steuerung mit der negativen Taktflanke wird ein Negationszeichen vor das Dreieck gesetzt. (Kreis vor dem Eingang)

- Ein taktflankengesteuertes D-Flipflop erhält man durch die Zusammenschaltung zweier D-Latches, die mit komplementären Taktpegeln gesteuert werden. Das erste Latch wird Master-Latch, das zweite Slave-Latch genannt. Ein solches Flipflop wird auch als **Master-Slave-Flipflop** (Siehe Abb. 7.8) bezeichnet. Während $T = 0$ folgt das erste Latch den Änderungen des Eingangssignals d, während das zweite Latch verriegelt ist. Ändert sich T von 0 auf 1 (positive Taktflanke), wird das erste Flipflop verriegelt. Unabhängig von den nun auftretenden Änderungen von d bleibt der Ausgabewert q_1 gleich dem Wert von d, der beim 0-1-Wechsel des Taktes anlag. Dieser Wert wird in das zweite Latch übernommen und dort auch weiter gespeichert, wenn T wieder auf 0 zurückgeht.

- Beim RS-Flipflop war die Eingangsvariablen-Kombination $r = s = 1$ verboten. Als vierte Funktion neben 'speichern', 'setzen' und 'rücksetzen' soll nun bei Eingangskombination $r = s = 1$ der Flipflop-Inhalt komplementiert werden. Bezeichnung: j - resultierender Setzeingang; k - resultierender Rücksetzeingang $\rightarrow$ **JK-Flipflop** (Abb. 7.9)

- Ein **T-Flipflop** ("to toggle", kippen; Abb. 7.10) hat nur einen Eingang. Liegt an diesem Eingang eine 1, kippt das Flipflop mit jedem Taktimpuls in einen anderen Zustand, hat die Eingangsvariable den Wert 0, behält das Flipflop seinen alten Zustand bei. Durch geeignete Eingangsbeschaltung eines JK-Flipflops lässt sich leicht das Verhalten eines T-Flipflops erzeugen. Ein synchrones Setzen oder Rücksetzen des T-Flipflops ist mit dem Eingang e nicht möglich. Um das Flipflop in einen definierten Grundzustand zu bringen, ist daher ein zusätzlicher Setz- oder Rücksetzeingang notwendig.

4.3 Synchrone Schaltwerke

Entwurf synchroner Schaltwerke am Beispiel des **Serienaddierers**:

- **Aufstellen des Automatengraphen**: Zwei Eingabevariablen-Folgen (Zahl x und Zahl y), sowie eine Folge von Ausgabewerten, die Summe s. In einem gegebenen Takt muss von der Vorgeschichte des Schaltwerks lediglich der Übertrag aus dem vorhergehenden Takt bekannt sein. Es reichen zwei Zustände aus: Ü - ein Übertrag aus der vorhergehenden Stelle ist zu berücksichtigen; kÜ: kein Übertrag.
- **Blockschaltnetz des Addierers**:

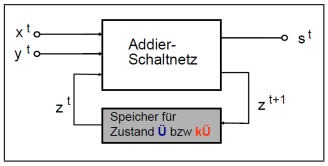

Abb. 4.1: Blockschaltnetz des Addierers

- **Automatengraph**: Es ist ein Mealy-Schaltwerk nötig, da die Ausgabe von den aktuellen Werten der Eingabevariablen abhängen soll. Im Automatengraphen werden deshalb die Ausgabebelegungen an die Kanten geschrieben.
- **Automatentafel des Addierers**: Aus dem Automatengraphen lässt sich die Automatentafel ableiten. Bei synchronen Schaltwerken werden stabile Zustände nicht gesondert markiert, da angenommen wird, dass alle Zustände bis zum nächsten Taktzyklus stabil sind.

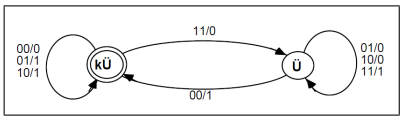

Abb. 4.2: Automatengraph des Addierers

- **Zustandskodierung**: Bei asynchronen Schaltwerken ist die Zustandskodierung sehr kritisch, denn die Wahl einer geeigneten Zustandskodierung ist für das Funktionieren des Schaltwerks entscheidend. Bei synchronen Schaltwerken ist die Zustandskodierung unkritisch, denn ein synchrones Schaltwerk funktioniert grundsätzlich mit jeder eindeutigen Zustandskodierung, doch kann eine gute Zustandskodierung bei synchronen Schaltwerken jedoch den Schaltungsaufwand reduzieren. Hier ist sie trivial, da wir nur zwei Zustände Ü und kÜ und somit nur eine Zustandsvariable ü haben.

z	z^+ / s			
	x y =			
	00	01	10	11
kÜ	kÜ /0	kÜ /1	kÜ /1	Ü /0
Ü	kÜ /1	Ü /0	Ü /0	Ü /1

Abb. 4.3: Automatentafel des Addierers

- **Kodierte Ablauftabelle**: Durch Einsetzen der Zustandskodierung in die Automatentafel halten wir die kodierte Ablauftabelle. Aus dieser lassen sich bereits die Funktionsausdrücke der Ausgabeschaltnetze ableiten. Zur Erzeugung der Schaltnetze der Überführungsfunktion muss zuvor noch der verwendete Flipflop-Typ festgelegt werden. Jeder Flipflop-Typ muss anders angesteuert werden.

4.4 Spezielle Schaltwerke, ALU

- Ein **Register** ist eine lineare Anordnung von Flipflops zur Speicherung mehrerer Bits (Bitvektor). Die Flipflops werden mit einem gemeinsamen Takt angesteuert.

Im allgemeinen werden die Flipflops durch zusätzliche gemeinsame Steuersignale beeinflusst.

- Ein **Schieberegister** ist eine Kette von in Reihe geschalteten Registern oder D-Flipflops. Der Ausgang eines Speicherelements ist jeweils mit dem Eingang des nächsten verbunden. Interpretiert man die Bitfolge als Dualzahl, entspricht ein Rechtsschieben einer Division durch 2 (ohne Rest). Schiebt man die Bitfolge ein Bit nach links (mit 0 als neuem letztem Bit), erhält man eine Multiplikation mit 2.

  Verbindet man den seriellen Ausgang eines Schieberegisters mit seinem seriellen Eingang, erhält man einen Umlaufspeicher (Ringzähler), der eine Bitfolge beliebig lange zwischenspeichern kann und dabei im Kreise schiebt.

- **Zähler** können Impulse abzählen und aufeinanderfolgende Adressen eines Speichers adressieren (z.B. bei Programmzählern) oder aufeinanderfolgende Arbeitsschritte kontrollieren (bei Steuerwerken). Eine vorgegebene Impulsfolge lässt sich in der Frequenz reduzieren, der Zähler wirkt als Frequenzteiler.

  Die Grösse des Ansteuerschaltnetzes wächst mit zunehmender Bitzahl stark an. Aus diesem Grund sind asynchrone Zähler attraktiv. Diese Schaltung ist langsamer, da jedes Flipflop erst dann reagiert, wenn das vorhergehende Flipflop von 1 nach 0 umgeschaltet hat. Ausserdem ändern sich die Ausgänge der Schaltung nicht gleichzeitig.

- **Arithmetisch-logische Einheit ALU**: Das ALU ist ein Rechenwerk, der funktionale Kern eines Digitalrechners zur Durchführung logischer und arithmetischer Verknüpfungen. Eingangsdaten: Daten und Steuersignalen vom Prozessor. Ausgangsdaten der ALU: Ergebnisse und Statussignale an den Prozessor.

5 Rechnerarchitektur - und Organisation

5.1 Aufbau und Funktionsweise

- Die **Zentraleinheit** besteht im wesentlichen aus den Komponenten der Hauptplatine (Mainboard oder Motherboard):
  - **Mikroprozessor** (CPU)): Ausführung der Programme, Steuerung und Verwaltung der Hardware.
  - **RAM-Arbeitsspeicher**: enthält Programme, die gerade ausgeführt werden, und verwendete Daten.
  - **ROM-Speicher**: enthält BIOS bzw. Programm, das beim Einschalten Hardware prüft und bootet vom Speichermedium)
  - **Busse und Schnittstellen**: Kommunikation zwischen einzelnen Bestandteilen des Mainboards, zum Anschluss von Peripheriegeräten
  - **Chipsatz**: fest auf dem Mainboard untergebrachte Schaltkreise, z.B. Für die Steuerung sämtlicher Anschlüsse des Mainboards.
- **Mikroprozessoren** sind integrierte elektronische Schaltkreise. Besteht aus:
  - **ALU**: Rechenwerk, das mathematische Operationen und logische Verknüpfungen durchführt.
  - **Register**: spezielle Speicherplätze innerhalb des Prozessorkerns. ALU rechnet mit den Werten, die sich in den Arbeitsregistern befinden.
  - **Steuerwerk**: übernimmt mittels zweier spezieller Register Kontrolle über die Ausführung des Programmcodes und initiiert andere Steuerungsfunktionen.
  - **Busse** (Datenleitungen) verbindet Prozessor mit den Komponenten. Datenbus: Austausch mit dem Arbeitsspeicher. Adressbus: Übertragen der zugehörigen Speicheradressen. Steuerbus: Ansteuerung der Peripherie-Anschlüsse.
- **Register** sind prozessorinterne Speicherplätze, die jeweils ein (binäres) Datum bestimmter Länge (z. B. 32-Bit) aufnehmen können. Sie besitzen sehr enge Verbindungen zu anderen Prozessorkomponenten. Nur mit den Daten in den Arbeitsregistern können direkte logische Operationen durchgeführt werden.

- **Binäre Maschinenbefehle** (im Befehlsregister verarbeitet) sind für den Menschen praktisch unlesbar. Daher wurde zur Erleichterung der Programmierung eine symbolische Schreibweise für Maschinenbefehle eingeführt (Assembler-Befehle). Diese werden dann vom Assembler in Maschinensprache übersetzt. Maschinenbefehle bestehen aus mehreren Teilen. Diese umfassen im Allgemeinen den eigentlichen Befehl, einen Operandenteil mit Angabe der Adressierungsart und einen Operandenwert oder eine Adresse.

- Die Adressierungsarten bieten verschiedene Möglichkeiten eines Prozessors, die Adresse eines Operanden oder eines Sprungziels im Speicher zu berechnen.

- Zwei Arten von **Speicher** existieren in einem Rechner: Der Arbeitsspeicher ("Kurzzeitgedächtnis", memory,Daten die jederzeit sofort zur Verfügung stehen müssen) und Peripheriespeicher (Massenspeicher, "Langzeitgedächtnis")

  Der Arbeitsspeicher ist eine lineare Liste von Speicherworten (Maximale Anzahl von Speicherelementen, die in einem Buszyklus zwischen Mikroprozessor und Speicher übertragen werden können). Eine Möglichkeit, Massenspeicher zu unterscheiden, ist deren physikalisches Schreib- und Leseverfahren (Magnetisch, optisch, magneto-optisch).

  Unter einem Cache-Speicher versteht man allgemein einen kleinen, schnellen Pufferspeicher, der vor einen langsamen, grösseren Speicher geschaltet wird, um dessen Zugriffszeit zu verbessern. Pufferspeicher hält Kopien derjenigen Teile des Arbeitsspeichers bereit, auf die aller Wahrscheinlichkeit nach von der CPU als nächstes zugegriffen wird.

5.2 Organisation

- Ein Digitalrechner besteht aus **Hardware** (mechanische und elektronische Bauelemente), **Software** (Programme, die auf dem Rechner ablaufen) und **Firmware** (Mikroprogramme in ROMs, Mittelstellung zwischen Hardware und Software).

- Das **BIOS**(Basic Input/Output System) ist ein Chip, der sich (bei IBM-PCs) auf dem Mainboard befindet und Firmware enhält, die hier die Basis-Steuerlogik für den Start des Rechners beinhaltet. Bei jedem Start eines Rechners wird zuerst ein Programm im BIOS ausgeführt, das bestimmte Tests durchführt, dazugehörige Kontrollmeldungen anzeigt und danach ein Betriebssystem von der Festplatte lädt und startet.

- Der **Chip-Satz** ist das Bindeglied zwischen den einzelnen Komponenten eines Computersystems und legt fest, welche Komponenten verwendet werden können.

- **Busse und Schnittstellen** werden sowohl zur Kommunikation zwischen den Bestandteilen des Mainboards als auch zum Anschluss aller Arten von Peripheriegeräten benötigt. Dem Transport von Daten zwischen den Einheiten auf Mainboard,

Prozessor, Arbeitsspeicher und Ausgabe/Eingabe dient ein internes Bussystem. Aus Geschwindigkeitsgründen werden darauf mehrere Bits parallel übertragen. Der **Datenbus** dient der bidirektionalen Übertragung von Daten zwischen den Einheiten. Der **Adressbus** dient der unidirektionalen Übermittlung von Adressen zum Speicher bzw. Ein/Ausgabeeinheiten. Der **Steuerbus** dient zur Koordination exklusiver Zugriffe auf den Daten- und Adressbus.

- Für die **Kommunikation** zwischen Prozessor, den einzelnen Bestandteilen des Mainboards und den Periperiegeräten werden verschiedene Techniken eingesetzt. Beim Polling von Daten fragt der Prozessor in bestimmten Zeitabständen nach, ob Daten zur Übertragung anstehen. Bei Interrupt Requests kann Kommunikation von einem Gerät durch Auslösung bestimmter Signale begonnen werden. Ein Systemsteuerbaustein wird benötigt, wenn mehrere Komponenten selbstständig auf den Systembus zugreifen. Für den Systembus-Zuteilungsverfahren gibt es zentrale Verfahren (unabhängige Anforderung) und dezentrale Verfahren (Kette). Möglichkeiten zur Übertragung der Kommunikationsdateien sind Programmed I/O, Memory Mapped I/O und DMA-Kanäle. Anschlüsse für Erweiterungskarten sind der PCI-, AGP- und PCMCIA-Anschluss. Für Laufwerke wie Festplatten, CD-ROM und andere Massenspeicher sind die EIDE und SCSI-Schnittstellen wichtig. Weitere Peripherieanschlüsse sind der USB-Anschluss und die IEEE-1394-Schnittstelle (Firewire). Diese unterstützen das Hot-Plugging-Verfahren und können im laufenden Betrieb angeschlossen und entfernt werden.

- Als **Peripherie** bezeichnet man Eingabe-, Ausgabe- und Speichergeräte, die an einen Rechner angeschlossen sind.

6 Betriebs- und Kommunikationssysteme

6.1 Überblick von Betriebssystemen

- Ein **Betriebssystem** umfasst die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen. Ziel eines Betriebssystems ist es, die semantische Lücke zwischen Anwendung (wünscht möglichst hohe Operationen) und Hardware (bietet elementare Operationen) zu verkleinern. Es erweitert die Fähigkeiten der Hardware.

- Das Betriebssystem bildet die Schnittstelle zwischen Mensch und Hardware. Dessen **Aufgabe** ist Verwaltung aller peripheren Betriebsmittel des Rechnersystems (Festplatte, CD, Drucker...), Verwaltung der Prozesse und Threads, Speichers, aller Einzelteile eines komplexen Systems (Prozessoren, Arbeitsspeicher etc), Geräte und Treiber.

6.2 Auftrags- und Speicherverwaltung

- Das Schlüsselkonzept moderner Betriebssysteme ist ein Prozesskonzept. Anwendungsprogramme werden in Form von **Prozessen** verwaltet. Ein Prozess ist die Ausführung eines Algorithmus, der durch ein ausführbares Programm beschrieben ist. Die Auswahl des nächsten zu bearbeitenden Prozesses und Umschaltung erfolgt durch Scheduler.

- Für statische Systeme mit immer gleichen Aufgaben wird statisches Scheduling (à la Zugfahrplan) eingesetzt. Für grössere Systeme mit beliebig start- und anhaltbaren Programmen werden dynamische Schedulingverfahren benötigt.

- Ein **Mutex** wird verwendet, wenn nur der Benutzer, welcher den Mutex hat, eine Veränderung machen darf. Eine **Semaphore** verleiht eine bestimmte Anzahl von Zugriffsrechten. Mutex und Semaphore werden mittels Tokens (Recht, eine bestimmte Aktion ausführen zu dürfen) implementiert. Ein Prozess, der eine Aktion ausführen will, muss ein Token aus einem Behälter nehmen und dieses nach Gebrauch wieder zurücklegen. Ist der Behälter leer, muss der Prozess warten, bis ein Token zurückgelegt wurde.

- Greift die CPU auf einen Hauptspeicher, welcher Arbeitsmengen beinhaltet, der mit einem Hintergrundspeicher, wo die Programme und Daten sind, kommuniziert, so bleibt Vorgang dem Anwender völlig verborgen, d.h. Arbeitsspeicher erscheint dem Anwender wesentlich grösser, als er ist. Ein nach diesem Konzept verwalteter Speicher heisst **virtueller Speicher**. Die Hauptaufgabe der virtuellen Speicherverwaltung ist die Umsetzung viruteller (logischer) Adressen in physikalische Adressen.

  Der Benutzer kennzeichnet Objekte (Programme, Variabeln etc) durch Namen. Der Compiler übersetzt diese Namen in virtuelle (logische) Adressen. Die virtuelle Speicherverwaltung wandelt diese Adressen in physikalische Adressen (wirklicher Ort des Objekts im Hauptspeicher) um.

- Zur virtuellen Speicherverwaltung existieren zwei grundlegende Verfahren. Beim **paging** (Aufteilung in Seiten) wird der logische und der physikalilsche Adressraum in Segmente fester Länge (Seiten) unterteilt. Ein Prozess wird auf viele dieser Seiten verteilt, es gibt keine logischen Zusammenhänge. Beim **swapping** (Aufteilung in Segmente) wird der virtuelle Adressraum in Segmente verschiedener Länge zerlegt und jedem Prozess ein oder mehrere Segmente zugeordnet. Die einzelnen Segmente enthalten logisch zusammenhängende Informationen und können rel. gross sein.

6.3 Einlagerung, Zuweisung, Ersetzung

- Beim Austausch von Daten zwischen Arbeits- Hintergrundspeicher ergeben sich drei Problemkreise: Der **Einlagerungszeitpunkt** (wann werden Segmente/Seiten in Arbeitssp. eingelagert?), das **Zuweisungsproblem** (wo werden Seiten/Segmente eingelagert?) und das **Ersetzungsproblem** (welche Segmente oder Seiten müssen ausgelagert werden, um Platz für neu benötigte Daten zu schaffen?)

- Beim Einlagerungszeitpunkt ist die Einlagerung auf Anforderung ein gängiges Verfahren. Die Daten werden eingelagert, sobald auf sie zugegriffen wird, sie sich aber nicht im Arbeitsspeicher befinden. Das Zuweisungsproblem ist bei Seitenwechselverfahren nicht existent, da alle Seiten gleich gross sind und somit immer passende Lücken nutzbar sind. Bei Segmentierungsverfahren muss eine ausreichend grosse Lücke im Arbeitsspeicher gefunden werden. Für das Ersetzungsproblem wird bei Segmentierungsverfahren meist die Anzahl von einem Prozess gleichzeitig benutzbaren Segmente limitiert und bei Einlagerung eines neuen Segments ein für diesen Prozess benutztes Segment ausgelagert. Bei Seitenwechselverfahren gibt es viele Möglichkeiten wie FIFO (first in first out), LIFO (Last in first out) etc.

- Befindet sich eine Seite/Segment nicht im Hauptspeicher, löst der Prozessor eine Unterbrechung aus, um die Seite/Segment durch das Betriebssystem zu laden (**Seiten-/Segmentfehler**).

- Das Betriebsystem führt über die freien Speicherbereiche Buch und lagert bei nicht ausreichendem Freiplatz im Hauptspeicher diejenigen Speicherinhalte auf den Hintergrundspeicher aus, die gegenüber ihren Originalen auf dem Hintergrundspeicher verändert worden sind.

7 Anhang

x_1 / x_0	0011 / 0101	verbale Form	symbolische Darstellung	Bezeichnung
f_0	0000	konstant 0	0	Kontradiktion, Symbol: $\bot$ (unerfüllbar)
f_1	0001	x_1 und x_0	$x_1 \wedge x_0$	Konjunktion
f_2	0010	nicht x_0, aber x_1	$x_1 \wedge \overline{x_0}$	Inhibition
f_3	0011	identisch x_1	x_1	Identität
f_4	0100	nicht x_1, aber x_0	$\overline{x_1} \wedge x_0$	Inhibition
f_5	0101	identisch x_0	x_0	Identität
f_6	0110	x_1 ungleich x_0	$x_1 \not\leftrightarrow x_0$	Antivalenz, XOR
f_7	0111	x_1 oder x_0	$x_1 \vee x_0$	Disjunktion
f_8	1000	nicht (x_1 oder x_0)	$x_1 \overline{\vee} x_0$	NOR-Funktion, Peircescher Pfeil
f_9	1001	x_1 gleich x_0	$x_1 \leftrightarrow x_0$	Äquivalenz
f_{10}	1010	nicht x_0	$\overline{x_0}$	Negation
f_{11}	1011	wenn x_0, dann x_1	$x_0 \rightarrow x_1$	Implikation
f_{12}	1100	nicht x_1	$\overline{x_1}$	Negation
f_{13}	1101	wenn x_1, dann x_0	$x_1 \rightarrow x_0$	Implikation
f_{14}	1110	nicht (x_1 und x_0)	$x_1 \overline{\wedge} x_0$	NAND-Funktion, Shefferscher Strich
f_{15}	1111	konstant 1	1	Tautologie, Symbol: T (allgemeingültig)

Abb. 7.1: 16 mögliche zweistellige boolesche Funktionen

Operatoren-system	Darstellung der ...		
	Negation	Konjunktion	Disjunktion
$(\wedge, \vee, \neg)$	$\overline{a}$	$a \wedge b$	$a \vee b$
$(\wedge, \neg)$	$\overline{a}$	$a \wedge b$	$\overline{\overline{a} \wedge \overline{b}}$
$(\vee, \neg)$	$\overline{a}$	$\overline{\overline{a} \vee \overline{b}}$	$a \vee b$
$(\overline{\wedge})$	$a \overline{\wedge} a$	$(a \overline{\wedge} b) \overline{\wedge} (a \overline{\wedge} b)$	$(a \overline{\wedge} a) \overline{\wedge} (b \overline{\wedge} b)$
$(\overline{\vee})$	$a \overline{\vee} a$	$(a \overline{\vee} a) \overline{\vee} (b \overline{\vee} b)$	$(a \overline{\vee} b) \overline{\vee} (a \overline{\vee} b)$
$(\wedge, \not\leftrightarrow)$	$a \not\leftrightarrow 1$	$a \wedge b$	$a \not\leftrightarrow b \not\leftrightarrow (a \wedge b)$

Abb. 7.2: Vollständige Operatorensysteme

$$
\begin{aligned}
y &= \bar{a}\,b\,c \vee a\,\bar{b}\,c \vee a\,b\,\bar{c} \vee \bar{a}\,\bar{b}\,\bar{c} \\
&= \overline{\overline{\bar{a}\,b\,c \vee a\,\bar{b}\,c \vee a\,b\,\bar{c} \vee \bar{a}\,\bar{b}\,\bar{c}}} \\
&= \overline{\overline{\bar{a}\,b\,c} \wedge \overline{a\,\bar{b}\,c} \wedge \overline{a\,b\,\bar{c}} \wedge \overline{\bar{a}\,\bar{b}\,\bar{c}}} \\
&= \mathrm{NAND}_4(\mathrm{NAND}_3(\bar{a}, b, c), \mathrm{NAND}_3(a, \bar{b}, c), \\
&\qquad \mathrm{NAND}_3(a, b, \bar{c}), \mathrm{NAND}_3(\bar{a}, \bar{b}, \bar{c}))
\end{aligned}
$$

Abb. 7.3: Beispiel Konversion in NAND-Operatorensystem

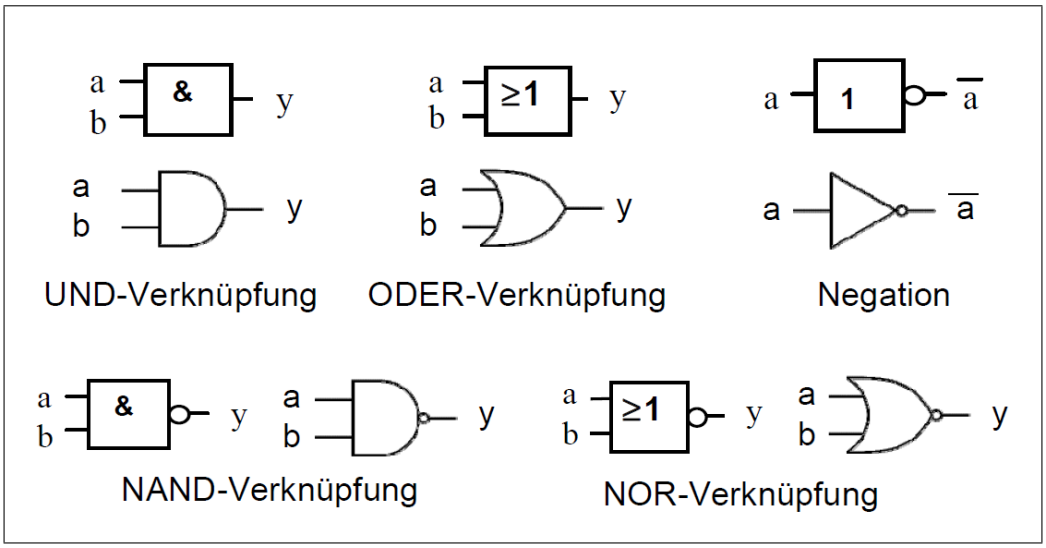

Abb. 7.4: Schaltsymbole (DIN 40900 Teil 12, ANSI/IEEEStandard 91-1984, IEC-Standard & US-Symbole

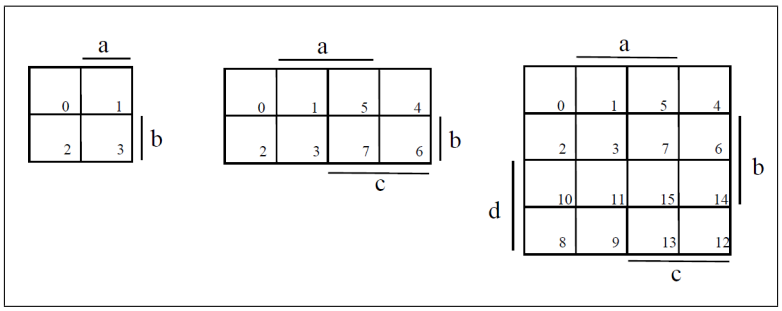

Abb. 7.5: KV-Diagramme

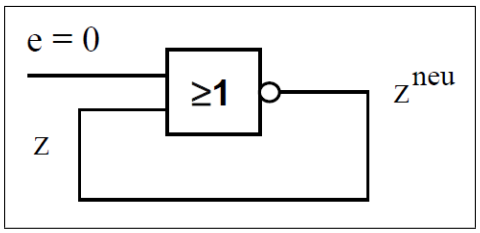

Abb. 7.6: Beispiel: Rückgekoppeltes NOR-Gatter

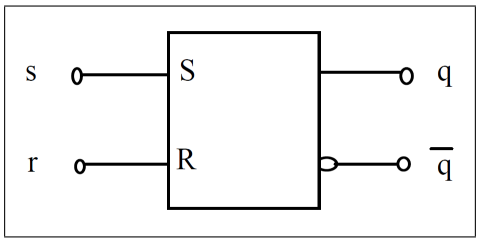

Abb. 7.7: Schaltsymbol RS-Flipflop

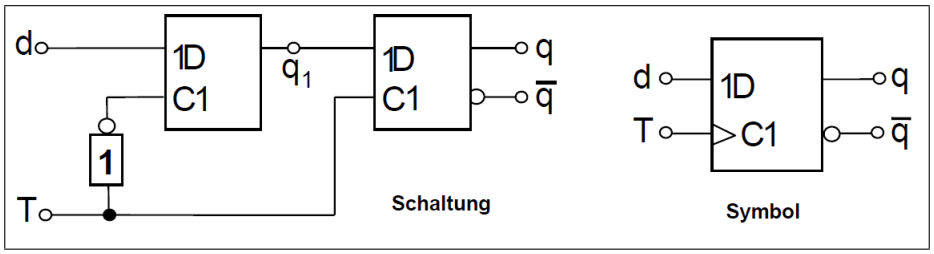

Abb. 7.8: Schaltsymbol und Schaltung Master-Slave-Flipflop

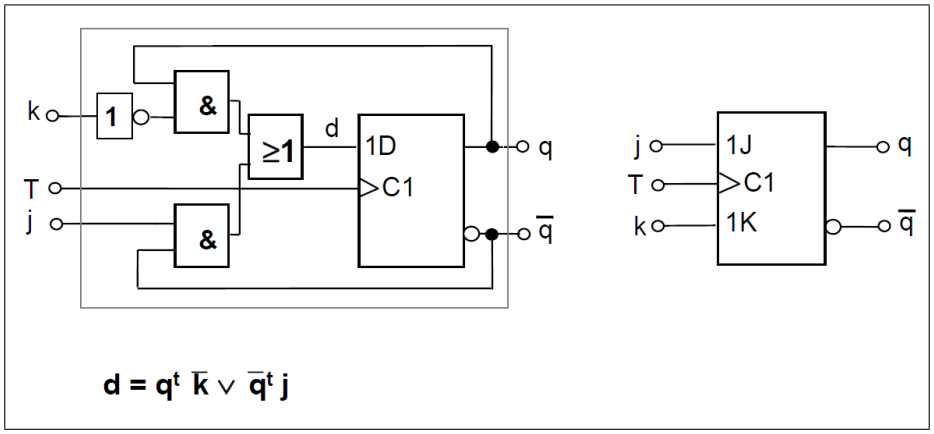

Abb. 7.9: Schaltsymbol und Schaltung JK-Flipflop

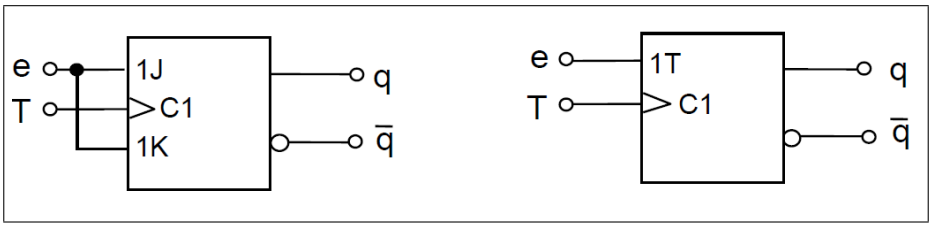

Abb. 7.10: Schaltsymbol und Schaltung Synchroner T-Flipflop

- RS-Flipflop (asynchron): r=s=1 verboten
  → RS-Latch (getaktet, pegelgesteuert)
- D-Flipflop, D-Latch: r und s =0 immer beachtet
- Taktflankengesteuertes D-Flipflop durch Zusammenschaltung zweier D-Latches
- JK-Flipflop: r und s = 1 → Ausgang komplementieren
- T-Flipflop: Eingang 1 → Ausgang komplementieren, sonst speichern

Abb. 7.11: Zusammenfassung Flipflops

RS-Flipflop

q^t	q^{t+1}	r	s
0	0	-	0
0	1	0	1
1	0	1	0
1	1	0	-

D-Flipflop

q^t	q^{t+1}	d
0	0	0
0	1	1
1	0	0
1	1	1

JK-Flipflop

q^t	q^{t+1}	j	k
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

T-Flipflop

q^t	q^{t+1}	e
0	0	0
0	1	1
1	0	1
1	1	0

Abb. 7.12: Ansteuertabellen