Universität Bielefeld

Interner Bericht der
Technischen Fakultät
Abteilung Informationstechnik

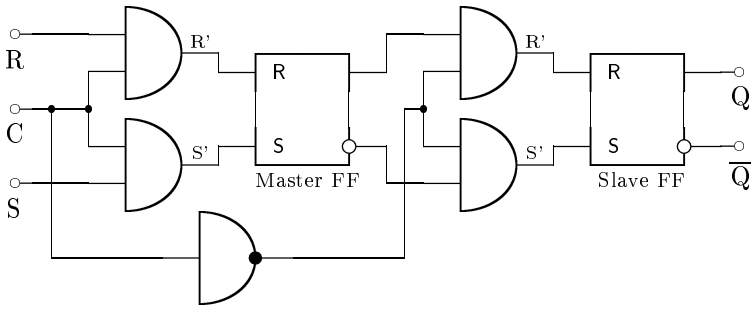
Skript zur Vorlesung

Technische Informatik I

Mirco Hilbert

Sommersemester 2001

(Stand: 1. April 2001)



Universität Bielefeld • Postfach 10 01 31 • 33501 Bielefeld

Vorwort

Dieses Skript basiert auf der Vorlesung „Technische Informatik II“, die Prof. Alois Knoll im Wintersemester 1998/99 gehalten hat, und auf einer Mitschrift von Frank Bettenworth vom Wintersemester 1996/97.

Dieses Skript wird noch überarbeitet und erweitert und erhebt somit keinen Anspruch auf Korrektheit und Vollständigkeit. Insbesondere der Abschnitt 2.1 „Zahldarstellungen und Rechenwerke“ wurde bis auf die Einteilung in Unterabschnitte noch gar nicht überarbeitet. Kritik, Fehlerkorrekturen sowie sonstige Verbesserungsvorschläge sind daher immer erwünscht. Die aktuellste Version befindet sich immer im Netz der Technischen Fakultät unter `/homes/mhilbert/pub/` oder im Internet unter `http://www.Mickel.MySite.de`.

Ich danke Frank Bettenworth, der den Grundstein für dieses Skript gelegt hat, Marco Balke und allen die noch zur Fehlerkorrektur des 98er Skriptes beigetragen haben, Peter Ladkin, der mich dazu überredet hat, dieses Skript noch einmal zu überarbeiten, und Alois Knoll dessen geistiges Eigentum der Inhalt des Skriptes ist. Außerdem möchte ich Sebastian Tannert danken, der zusammen mit Andreas Tille das LaTeX-Paket CIRC geschrieben hat, mit dem ich die Schaltungen gesetzt habe, und der mir zuletzt noch geholfen hat, das bestehende Symbolinventar um ein benötigtes Symbol zu erweitern.

Bielefeld, im April 2001

*Mirco Hilbert**

*eMail: `mhilbert@TechFak.Uni-Bielefeld.de`

Inhaltsverzeichnis

1 Informationsverarbeitung: Schaltwerke und Schaltnetze **1**

1.1 Bool'sche Algebra . . . 1

1.2 Schaltfunktionen . . . 3

1.2.1 Anzahl der Schaltfunktionen . . . 3

1.2.2 Ziele für das Folgende . . . 3

1.3 Normalformen . . . 3

1.3.1 Definitionen . . . 4

1.3.2 Fundamentaler Satz für boolsche Normalform . . . 4

1.3.2.1 Beweis für alle n über Induktion . . . 4

1.3.2.2 Beispiel . . . 5

1.3.2.3 Bemerkung . . . 5

1.3.3 Vereinfachungsregeln . . . 5

1.3.4 Kanonisierungsregeln . . . 6

1.4 Systematische Verfahren zur Vereinfachung von Schaltfunktionen . . . 6

1.4.1 Karnaugh-Diagramm (K-Diagramm) . . . 6

1.4.1.1 Beispiel (für $n = 2$) . . . 6

1.4.1.2 Zweites Beispiel (für $n = 3$) . . . 7

1.4.2 Vereinfachung von Schaltfunktionen mit Hilfe von Quine-McClusky . . . 7

1.4.2.1 Grundprinzip . . . 7

1.4.2.2 Beispiel . . . 7

1.4.2.3 Systematisches Vorgehen . . . 7

1.4.2.4 Beispiel . . . 8

1.5 Synthese von Schaltwerken . . . 9

1.5.1 Kombinatorischer Automat (Schaltnetz) . . . 9

1.5.1.1 Vorbemerkung . . . 9

1.5.1.2 Definitionen . . . 9

1.5.1.3 Sätze . . . 10

1.5.1.4 Beispiele für kombinatorische Automaten . . . 10

1.5.2 Sequentielle Automaten (Schaltwerke) bzw. sequentielle Schaltnetze . . . 11

1.5.2.1 Bemerkung zur Taktsteuerung . . . 11
1.5.2.2 Drei Arten der Taktsteuerung . . . 12
1.5.2.3 Speicher und Rückführung . . . 12
1.5.2.4 Allgemeiner sequentieller Automat . . . 16
1.5.2.5 Möglichkeit zur Realisierung der Automaten . . . 16
1.5.2.6 Darstellung des Verhalten von Automaten . . . 17
1.6 Speicherbauelemente . . . 20
1.6.1 Einfaches Flip-Flop: RS-Flip-Flop . . . 20
1.6.1.1 Realisierungsmöglichkeiten . . . 20
1.6.2 Taktzustandsgesteuertes RS-Flip-Flop . . . 21
1.6.3 Taktflankengesteuertes RS-Flip-Flop . . . 21
1.6.4 Master-Slave Flip-Flop . . . 22
1.6.5 Jump/Kill Flip-Flop . . . 22
1.6.6 Jump/Kill-Master-Slave Flip-Flop . . . 22
1.7 Elektrische Realisierung von Gattern . . . 23
1.7.1 Realisierung mit elektromechanischen Schaltern (Relais) . . . 23
1.7.1.1 Beispiel: ODER-Gatter . . . 23
1.7.2 Realisierung mit elektronischen Schaltern (Transistoren) . . . 24
1.7.2.1 Grundschaltung eines Transistorschalters . . . 24
1.7.2.2 Bestimmung der Arbeitspunkte für „ein“ und „aus“ . . . 25
1.7.2.3 Konkrete Realisierungen . . . 27
1.7.2.4 TTL-Technik: Transistor-Transistor-Logik . . . 30

2 Zahldarstellungen und Informationsrepräsentation 34
2.1 Zahldarstellungen und Rechenwerke . . . 34
2.1.1 Zahldarstellungen . . . 34
2.1.1.1 Ganzzahlige-Darstellung im Detail . . . 35
2.1.1.2 Darstellung nach Vorzeichen und Betrag . . . 35
2.1.1.3 Zweikomplementdarstellung . . . 35
2.1.1.4 Einerkomplement . . . 36
2.1.1.5 Binärkodierte Dezimalzahlen (Dualzahlen) . . . 36
2.1.1.6 Sedezimalzahlen (Hexadezimalzahlen) . . . 36
2.1.1.7 Fließkommazahlen . . . 37
2.1.2 Schaltnetze zur Addition . . . 38
2.1.2.1 Addition von zwei n-bit-Zahlen . . . 38
2.1.2.2 Addition von Zweikomplementzahlen . . . 39
2.1.3 Verfahren zur Division von Dualzahlen . . . 40
2.2 Informationsrepräsentation: Kodierungstheorie . . . 40

2.2.1 Kodierung von Informationen . . . 40
2.2.1.1 Kodierung für interne Repräsentation . . . 40
2.2.1.2 Kodierung für Informationsübertragung und Störung . . . 41
2.2.1.3 Anforderungen . . . 41
2.2.1.4 Einige Definitionen . . . 41
2.2.2 Codes fester Länge . . . 42
2.2.2.1 Beispiel: Direkter Code für die Dezimalziffern mit den Gewichten $g_i = 2^i$ 42
2.2.2.2 Definition: Hammingabstand . . . 42
2.2.2.3 Beispiel: Einschrittiger Code (Gray-Code) für die Ziffern 0 ... 9 . . . 43
2.2.2.4 Kodierung zur Fehlererkennung und -korrektur . . . 43
2.2.2.5 Binäre Block-Codes . . . 44
2.2.2.6 Lineare systematische Codes . . . 45
2.2.2.7 Zyklische binäre Codes . . . 47
2.2.3 Codes variierender Länge . . . 48
2.2.3.1 Serienwortkodierung . . . 48
2.2.3.2 Kodierung unter Berücksichtigung der Wahrscheinlichkeit . . . 49
2.2.3.3 Abschließende Bemerkungen zur Kodierung . . . 54
2.2.3.4 Lexikon-basierte Kodierung . . . 54

Literaturliste **56**

Stichwortverzeichnis **57**

Abbildungsverzeichnis

1.1 Synthese von Schaltwerken . . . 9
1.2 Schaltung: Exklusiv-Oder als Beispiel für einen kombinatorischen Automaten . . . 10
1.3 Schaltung: Kombinatorischer Automat aus m Einzelautomaten . . . 10
1.4 Schaltung: Halbaddierer . . . 11
1.5 Schaltung: Taktsteuerung . . . 12
1.6 Impulsdiagramm zur Taktsteuerungsschaltung (Schaltzeit z. B. 10 ns) . . . 13
1.7 Impulsdiagramm: Zweiflankensteuerung . . . 14
1.8 Schaltung: Takthalbierer . . . 14
1.9 Impulsdiagramm: Takthalbierer . . . 14
1.10 Schaltung: Schaltnetz mit Speichern . . . 15
1.11 Schaltung: kombinatorisches Netzwerk unter Verwendung von Speichern . . . 17
1.12 Ampelschaltung . . . 18
1.13 Schaltung: Steuerung der Ampelanlage . . . 19
1.14 Automatengraph zur Ampelschaltung . . . 20
1.15 Schaltung: RS-Flip-Flop . . . 20
1.16 Schaltung: Zwei NOR-Gatter mit gekreuzten Eingängen . . . 21
1.17 Schaltung: Taktzustandsgesteuertes RS-Flip-Flop . . . 21
1.18 Schaltung: Taktflankengesteuertes RS-Flip-Flop . . . 21
1.19 Schaltung: Master-Slave Flip-Flop . . . 22
1.20 Schaltung: Jump/Kill Flip-Flop . . . 22
1.21 Schaltung: Jump/Kill-Master-Slave Flip-Flop . . . 23
1.22 Schaltung: Durch Relais realisiertes ODER-Gatter . . . 23
1.23 Schalterstrom-Kennlinie . . . 24
1.24 Schalterstrom-Kennlinie . . . 24
1.25 Schaltung: Transistorschalter . . . 25
1.26 Schaltcharakteristik . . . 25
1.27 Arbeitsgerade mit den beiden Arbeitspunkten für „ein“ und „aus“ . . . 25
1.28 Kennlinienfeld des Transistors . . . 26
1.29 Schaltung: Inverter in Widerstands-Transistor-Logik . . . 28

1.30 Schaltung: Ersatzschaltbild wenn Q = $High$ 28
1.31 Schaltung: Ersatzschaltbild wenn Q = Low 28
1.32 Schaltung: ODER-Gatter ohne Verstärker 29
1.33 Schaltung: ODER-Gatter mit Verstärker 29
1.34 Schaltung: Passives UND-Gatter 30
1.35 Schaltung: NPN-Transistor kann durch zwei Dioden nachgebildet werden 30
1.36 Schaltung: Passives UND-Gatter in DTL-Technik und „aktives“ UND-Gatter in TTL-Technik 30
1.37 Schaltung: Verwendung des Multi-Emitter-Transistors zum Aufbau eines UND-Gatters . . 31
1.38 Schaltung: Ersatzschaltbild wenn $Q \approx U_{CE\,\mathrm{sat}} = Low$ 31
1.39 Schaltung: Ersatzschaltbild wenn $Q = High$ 32
1.40 Schaltung: Um einen Verstärker erweitertes UND-Gatter 32
1.41 Schaltung: Gegentakt-Endstufe 33
1.42 Ersatzschaltbild wenn $Q = High$ 33
1.43 Ersatzschaltbild wenn $Q = Low$ 33

2.1 Informationsverarbeitendes System 41
2.2 Kodierung für Informationsübertragung 41
2.3 gesendete Bitfolge wird durch eine Störung verfälscht 43
2.4 Codewürfel 44
2.5 Entropie-Wahrscheinlichkeits-Diagramm 50
2.6 ausgeglichener Codebaum (in Bezug auf die Wahrscheinlichkeitsmassen) 51

Kapitel 1

Informationsverarbeitung: Schaltwerke und Schaltnetze

Informationsverarbeitungsvorgänge benötigen neben der logischen eine physikalische Darstellung. Der Verarbeitungsvorgang ist identisch mit einer Umformung der physikalischen Repräsentation.

Viele physikalische Repräsentationen sind denkbar: Stärke eines hydraulischen oder elektrischen Stroms, Farbe, Intensität oder Phasenlage von Licht, stufenlos veränderbare Spannungspegel usw.

Beim heutigen Stand der Technik ist die Darstellung von Information durch die zweielementige Menge der boolschen Werte $\mathbb{B} = \{0, L\}$ bzw. durch Binärworter (= Tupel von boolschen Werten) oder durch Sequenzen von solchen Wörtern üblich. Damit ist die Informationsverarbeitung beschreibbar durch Funktionen auf Binärwörtern bzw. Folgen davon. Die elektrischen Schaltwerke und -netze realisieren unter Umständen sehr komplexe boolsche Funktionen und bestehen ihrerseits aus Verknüpfungen elementarer boolscher Funktionen: „NICHT“, „UND“ und „ODER“.

Im folgenden Behandlung von boolscher Algebra, boolschen Funktionen, Verknüpfungen solcher Funktionen und Vereinfachung, sowie Umsetzung in elektrischen Schaltungen.

1.1 Bool'sche Algebra

Es werden zweistellige Operatoren „$\wedge$“ und „$\vee$“ betrachtet, die

- dem Kommutativgesetz:
  $a \wedge b = b \wedge a$
  $a \vee b = b \vee a$
- dem Assoziativgesetz:
  $(a \wedge b) \wedge c = a \wedge (b \wedge c)$
  $(a \vee b) \vee c = a \vee (b \vee c)$
- dem Absorptionsgesetz:
  $a \wedge (a \vee b) = a$
  $a \vee (a \wedge b) = a$
- dem Idempotenzgesetz:
  $a \wedge a = a$
  $a \vee a = a$

} Verband

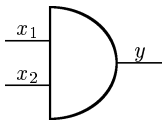
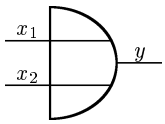
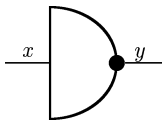
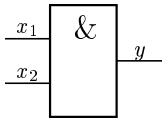
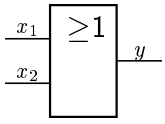
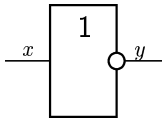
genügen. Außerdem muß das Distributivgesetz gelten:
$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

} distributiver Verband

Für eine boolsche Algebra wird zusätzlich gefordert, daß ein einstelliger Negationsoperator („$\neg$“ oder „$\overline{a}$“) definiert ist und das Neutralitätsgesetz sowie das Gesetz von De Morgan gelten:

- neutrales Element:
  $a \wedge e = a$
  $a \vee n = a$
- Negation:
  $a \wedge \overline{a} = n$
  $a \vee \overline{a} = e$
- Gesetz von De Morgan:
  $\overline{a \wedge b} = \overline{a} \vee \overline{b}$
  $\overline{a \vee b} = \overline{a} \wedge \overline{b}$

Eine boolsche Algebra kann für die beiden Elemente $\mathbb{B} = \{0, L\}$ definiert werden (oder auch für Binärwörter gleicher Länge, oder auch für Funktionen, die boolsche Werte als Resultate liefern).
Im Falle der boolschen Algebra über der zweielementigen Trägermenge $\mathbb{B} = \{0, L\}$ spricht man von einer Schaltalgebra.
Die elementaren Verknüpfungsoperationen werden in Schaltnetzen, z. B. durch elektronische Schaltungen, realisiert. Dafür werden **Schaltzeichen** eingeführt und die Resultate der Verknüpfungen werden zweckmäßigerweise als **Wertetabellen** dargestellt.

Schaltzeichen	UND (AND) $\wedge$	ODER (OR) $\vee$	NICHT (NOT) $\neg$
– alte Norm:			
– neue Norm:			
boolsche Funktion:	$y = f(x_1, x_2) = x_1 \wedge x_2$	$y = f(x_1, x_2) = x_1 \vee x_2$	$y = f(x) = \neg\, x$

Wertetabelle:

x_1	x_2	y
0	0	0
0	L	0
L	0	0
L	L	L

x_1	x_2	y
0	0	0
0	L	L
L	0	L
L	L	L

x	y
0	L
L	0

Alle möglichen Kombinationen von zwei boolschen Variablen lassen sich wie folgt auf die elementaren Verknüpfungen zurückführen:

		L	L	0	0	x_1
		L	0	L	0	x_2
	Name					$y = f(x_1, x_2)$
	Null	0	0	0	0	0
→	NOR	0	0	0	L	$\overline{x_1 \vee x_2}$
→	Inhibition	0	0	L	0	$\overline{x_1} \wedge x_2$
→	Negation	0	0	L	L	$\overline{x_1}$
	Inhibition	0	L	0	0	$x_1 \wedge \overline{x_2}$
	Negation	0	L	0	L	$\overline{x_2}$
→	Antivalenz (XOR)	0	L	L	0	$(\overline{x_1} \wedge x_2) \vee (x_1 \wedge \overline{x_2})$
→	NAND	0	L	L	L	$\overline{x_1 \wedge x_2}$
→	AND	L	0	0	0	$x_1 \wedge x_2$
→	Äquivalenz	L	0	0	L	$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$
	Transfer	L	0	L	0	x_2
	Implikation	L	0	L	L	$\overline{x_1} \vee x_2$
	Transfer	L	L	0	0	x_1
	Implikation	L	L	0	L	$x_1 \vee \overline{x_2}$
→	OR	L	L	L	0	$x_1 \vee x_2$
	Eins	L	L	L	L	L

→: technisch relevant.

1.2 Schaltfunktionen

Eine Abbildung $f : \mathbb{B}^n \to \mathbb{B}$ mit $f(x_1, \ldots, x_n) = y$ und $(x_1, \ldots, x_n) \in \mathbb{B}^n$ sowie $y \in \mathbb{B} = \{0, L\}$ heißt n-stellige **Schaltfunktion**. Sie wird definiert durch einen boolschen Term aus den Variablen $x_1 \ldots x_n$, den Konstanten 0 und L, sowie den Operatoren $\wedge$, $\vee$ und $\neg$, oder aber durch eine Wertetabelle, d. h. die Angabe aller Resultate für alle möglichen Belegungen der Variablen.

1.2.1 Anzahl der Schaltfunktionen

Sei f eine Schaltfunktion $f : \mathbb{B}^n \to \mathbb{B}$ mit n frei belegbaren Variablen. Dann gibt es 2^n mögliche Darstellungen (Kombinationen) von Wertebelegungen für die n Variablen. Eine bestimmte Funktion f ist für jede dieser $w = 2^n$ Eingangskombinationen definiert, sie produziert also w Ergebnisse (jeweils 0 oder L). Durch die Wahl unterschiedlicher Schaltfunktionen $f : \mathbb{B}^n \to \mathbb{B}$ lassen sich für diese w Ergebnisse 2^w Ergebniskombinationen vorschreiben. Es existieren also genau 2^{2^w} boolsche Funktionen, die alle möglichen Wertekombinationen der Eingangsvariablen auf alle Kombinationen von Resultaten abbilden.

1.2.2 Ziele für das Folgende

1. Überführung einer ein bestimmtes Problem beschreibenden Wertetabelle (mit endlich vielen Elementen) in eine zu denselben Resultaten führende Schaltfunktion und danach in ein elektrisches Schaltnetz.
2. Vereinfachung dieser Schaltfunktion; je weniger Elemente notwendig sind, um die Funktion zu realisieren, desto geringer ist der Schaltungsaufwand (Kosten, Zeiten).

1.3 Normalformen

Bei eindeutiger Normalformdarstellung läßt sich die Frage entscheiden, wann zwei Terme die gleiche Schaltfunktion darstellen.

1.3.1 Definitionen

1. **Term**:
   (a) Die Konstanten 0 und L sowie die Variablen $x_1, \ldots, x_n$ sind Terme.
   (b) Ist t ein Term, so ist auch $\neg t$ ein Term.
   (c) Sind t_1 und t_2 Terme, dann auch $t_1 \vee t_2$ und $t_1 \wedge t_2$
2. **Minterm**: Term, der aus der UND-Verknüpfung aller betrachteten Variablen besteht. Jede Variable kommt darin genau einmal vor, und zwar in negierter oder nicht-negierter Form. Also:
   minterm $(b_1, \ldots, b_n) = (v_1 \wedge \ldots \wedge v_n)$ mit $v_i = \begin{cases} x_i \text{ falls } b_i = L \\ \overline{x_i} \text{ falls } b_i = 0 \end{cases}$
   Beispiel: minterm $(L, 0) = (x_1 \wedge \overline{x_2})$
   Beobachtung: minterm $(L, 0)$ ist nur „L“ für die Belegung $(L, 0)$ der beiden Eingangsvariablen.
3. **Maxterm**: Analog zum Minterm, aber für die ODER-Verknüpfung:
   maxterm $(b_1, \ldots, b_n) = (v_1 \vee \ldots \vee v_n)$ mit $v_i = \begin{cases} x_i \text{ falls } b_i = L \\ \overline{x_i} \text{ falls } b_i = 0 \end{cases}$

1.3.2 Fundamentaler Satz für boolsche Normalform

Jede n-stellige Schaltfunktion kann durch einen Term in **disjunktiver Normalform** (DNF) repräsentiert werden. Disjunktive Normalform liegt vor, wenn ein Term entweder 0 oder L ist oder die folgende Gestalt besitzt: $t = (t_1 \vee t_2 \vee \ldots \vee t_n)$ mit $t_i = (v_{i1} \wedge v_{i2} \wedge \ldots \wedge v_{ik})$
Der Satz wird plausibel, wenn die DNF in der folgenden Weise durch Hinschreiben aller Möglichkeiten konstruiert wird. Beispiel (für $n = 2$)

$$\begin{aligned} f(x_1, x_2) = \; & (f(0,0) \wedge (\overline{x_1} \wedge \overline{x_2})) \\ \vee \; & (f(0,L) \wedge (\overline{x_1} \wedge x_2)) \\ \vee \; & (f(L,0) \wedge (x_1 \wedge \overline{x_2})) \\ \vee \; & (f(L,L) \wedge (x_1 \wedge x_2)) \end{aligned}$$

Dies ist die „Erweiterung“ der Funktion f mit konkreten Werten jeweils um eine UND-Verknüpfung, die bei der jeweiligen Wertebelegung der Variablen (und nur da) zu L führt. Die Funktion selbst kann bei der jeweiligen Belegung nur den Wert L (dann bleibt minterm erhalten) oder 0 aufweisen (dann fällt minterm sofort weg).

1.3.2.1 Beweis für alle n über Induktion

Allgemein gilt:
$f(x_1, x_2, \ldots, x_n) = \bigvee_{b_1 \ldots b_n \in \mathbb{B}^n} (f(b_1, b_2, \ldots, b_n) \wedge \text{minterm}(b_1, \ldots, b_n))$
$b_1 \ldots b_n \in \mathbb{B}^n$ sind hierbei alle Variationen mit Wiederholung bzw. alle „Binärzahlen“ $0 \ldots 2^{n-1}$.
Damit Normdarstellung gefunden, allerdings nicht unbedingt kürzeste Form. Die disjunktive Verknüpfung aller Minterme wird **vollständige** oder **kanonische disjunktive Normalform** (KDNF) genannt. Bei der kanonischen DNF kommen alle Variablen in jedem der Konjunktionsterme genau einmal vor.
Häufig kann die DNF weiter vereinfacht werden, wenn Teilterme der Form
$\ldots \vee (t_1 \wedge \ldots \wedge t_{i-1} \wedge x_i \wedge t_{i+1} \wedge \ldots \wedge t_n) \vee \ldots \vee (t_1 \wedge \ldots \wedge t_{i-1} \wedge \overline{x_i} \wedge t_{i+1} \wedge \ldots \wedge t_n) \vee \ldots$
auftreten. Dann spielt x_i keine Rolle und die beiden Teilterme können zusammengefaßt werden zu einem Term der Form $\ldots \vee (t_1 \wedge \ldots \wedge t_{i-1} \wedge t_{i+1} \wedge \ldots \wedge t_n) \vee \ldots$

1.3.2.2 Beispiel

Gegeben sei eine Wertetabelle der Form:

Dezimal	x_3	x_2	x_1	$f(x_1,x_2,x_3)$	minterm(x_1,x_2,x_3)
0	0	0	0	L	$\overline{x_3} \wedge \overline{x_2} \wedge \overline{x_1}$
1	0	0	L	0	$\overline{x_3} \wedge \overline{x_2} \wedge x_1$
2	0	L	0	0	$\overline{x_3} \wedge x_2 \wedge \overline{x_1}$
3	0	L	L	0	$\overline{x_3} \wedge x_2 \wedge x_1$
4	L	0	0	0	$x_3 \wedge \overline{x_2} \wedge \overline{x_1}$
5	L	0	L	L	$x_3 \wedge \overline{x_2} \wedge x_1$
6	L	L	0	0	$x_3 \wedge x_2 \wedge \overline{x_1}$
7	L	L	L	L	$x_3 \wedge x_2 \wedge x_1$

Umformung in Termdarstellung:

$$
\begin{aligned}
f(x_1,x_2,x_3) = {} & (L \wedge \overline{x_3} \wedge \overline{x_2} \wedge \overline{x_1}) && {}^{0}\\
\vee\ & (0 \wedge \overline{x_3} \wedge \overline{x_2} \wedge x_1) && {}^{1}\\
\vee\ & (0 \wedge \overline{x_3} \wedge x_2 \wedge \overline{x_1}) && {}^{2}\\
\vee\ & (0 \wedge \overline{x_3} \wedge x_2 \wedge x_1) && {}^{3}\\
\vee\ & (0 \wedge x_3 \wedge \overline{x_2} \wedge \overline{x_1}) && {}^{4}\\
\vee\ & (L \wedge x_3 \wedge \overline{x_2} \wedge x_1) && {}^{5}\\
\vee\ & (0 \wedge x_3 \wedge x_2 \wedge \overline{x_1}) && {}^{6}\\
\vee\ & (L \wedge x_3 \wedge x_2 \wedge x_1) && {}^{7}\\
\Rightarrow f(x_1,x_2,x_3) = {} & (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}) \vee \underbrace{(x_1 \wedge \overline{x_2} \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)}_{(x_1 \wedge x_3)}
\end{aligned}
$$

Dadurch erhält man die vereinfachte DNF:
$f(x_1,x_2,x_3) = (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}) \vee (x_1 \wedge x_3)$
Hier kommen nicht mehr in jedem Term alle Variablen vor.

1.3.2.3 Bemerkung

Es gibt Wertetabellen, bei denen nicht für jede Belegung der Eingangsvariablen ein bestimmtes Ereignis definiert ist (bzw. vorgeschrieben werden muß). Dies kann bei der Vereinfachung von Schaltfunktionen nützlich sein.
Hier wäre es z. B. hilfreich, wenn der Term $(x_1 \wedge \overline{x_2} \wedge \overline{x_3})$ hinzugenommen werden würde (das bedeutet in Zeile 1 ist das Ergebnis ebenfalls „L“).

1.3.3 Vereinfachungsregeln

Die DNF ist nicht mehr weiter zu vereinfachen, wenn keine der folgenden Regeln mehr anwendbar ist:

Distributivgesetz:	$x_1 \wedge (x_2 \vee x_3) \rightarrow (x_1 \wedge x_2) \vee (x_1 \wedge x_3)$	$x_1 \vee (x_2 \wedge x_3) \rightarrow (x_1 \vee x_2) \wedge (x_1 \vee x_3)$
De Morgan :	$\overline{x_1 \wedge x_2} \rightarrow \overline{x_1} \vee \overline{x_2}$	$\overline{x_1 \vee x_2} \rightarrow \overline{x_1} \wedge \overline{x_2}$
Doppelte Negation:	$\overline{\overline{x_1}} \rightarrow x_1$	
Assoziativität:	$(x_1 \wedge x_2) \wedge x_3 \rightarrow x_1 \wedge (x_2 \wedge x_3)$	$(x_1 \vee x_2) \vee x_3 \rightarrow x_1 \vee (x_2 \vee x_3)$
Idempotenz:	$x_1 \wedge x_1 \rightarrow x_1$	$x_1 \vee x_1 \rightarrow x_1$
Negation:	$x_1 \wedge \overline{x_1} \rightarrow 0$	$x_1 \vee \overline{x_1} \rightarrow L$
neutrales Element:	$x_1 \wedge L \rightarrow x_1$	$x_1 \vee 0 \rightarrow x_1$

1.3.4 Kanonisierungsregeln

Kommt in der boolschen Funktion eine Variable z vor, die in der DNF nicht mehr auftaucht, so kann sie durch die folgende Ersetzung eingeführt werden:
$t \to (t \wedge z) \vee (t \wedge \overline{z})$ (Vereinfachte DNF $\to$ KDNF).

1.4 Systematische Verfahren zur Vereinfachung von Schaltfunktionen

1.4.1 Karnaugh-Diagramm (K-Diagramm)

Graphische Darstellung einer Schaltfunktion f, äquivalent zur Wertetabelle oder Normalform. Speziell geeignet, um Schaltfunktionen mit bis zu $n = 4$ Variablen zu vereinfachen. Hierzu werden die Ergebnisse von f für alle Wertekombinationen der Eingangsvariablen in ein Diagramm mit 2^n Feldern eingetragen.

Fall $n = 2$:

x_2 \ x_1	0	L
0	$f(0,0)$	$f(L,0)$
L	$f(0,L)$	$f(L,L)$

Minterme für die entsprechenden Variablenkombinationen:
$f(0,0)$: $\overline{x_1} \wedge \overline{x_2}$ $\quad$ $f(L,0)$: $x_1 \wedge \overline{x_2}$
$f(0,L)$: $\overline{x_1} \wedge x_2$ $\quad$ $f(L,L)$: $x_1 \wedge x_2$

Fall $n = 3$:

x_3 \ x_2, x_1	$0,0$	$0,L$	L,L	$L,0$
0	$f(0,0,0)$	$f(L,0,0)$	$f(L,L,0)$	$f(0,L,0)$
L	$f(0,0,L)$	$f(L,0,L)$	$f(L,L,L)$	$f(0,L,L)$

Fall $n = 4$:

x_4, x_3 \ x_2, x_1	$0,0$	$0,L$	L,L	$L,0$
$0,0$	$f(0,0,0,0)$	$f(L,0,0,0)$	$f(L,L,0,0)$	$f(0,L,0,0)$
$0,L$	$f(0,0,L,0)$	$f(L,0,L,0)$	$f(L,L,L,0)$	$f(0,L,L,0)$
L,L	$f(0,0,L,L)$	$f(L,0,L,L)$	$f(L,L,L,L)$	$f(0,L,L,L)$
$L,0$	$f(0,0,0,L)$	$f(L,0,0,L)$	$f(L,L,0,L)$	$f(0,L,0,L)$

1.4.1.1 Beispiel (für $n = 2$)

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

Wertetabelle:

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	L	0
L	0	L
L	L	L

Eintrag in das K-Diagramm:

x_2 \ x_1	0	L
0	0	L
L	0	L

$\begin{matrix} L \\ L \end{matrix}$ entspricht x_1, d. h. diese beiden L werden ebenso durch die Funktion $f(x_1, x_2) = x_1$ erzeugt, diese Funktion ist also äquivalent.
Im K-Diagramm entspricht dies einem senkrechten Block über die Grenze von x_2 hinweg, x_2 wird also für die Ergebniserzeugung nicht benötigt.

1.4.1.2 Zweites Beispiel (für $n = 3$)

$f(x_1, x_2, x_3) = (\overline{x_1} \wedge x_2 \wedge \overline{x_3}) \vee (\overline{x_1} \wedge x_2 \wedge x_3) \vee (x_1 \wedge \overline{x_2} \wedge \overline{x_3}) \vee (x_1 \wedge \overline{x_2} \wedge x_3)$

Eintrag in das K-Diagramm:

x_3 \ x_2, x_1	$0,0$	$0,L$	L,L	$L,0$
0	0	L	0	L
L	0	L	0	L

Die beiden $\begin{matrix} L \\ L \end{matrix}$ werden von der DNF $f(x_1, x_2) = (\overline{x_2} \wedge x_1) \vee (x_2 \wedge \overline{x_1})$ erzeugt. Diese ist offensichtlich einfacher, als die Ausgangsfunktion. Größer ist der „Reduktionserfolg“, wenn Viererblöcke oder gar Achterblöcke zusammengefaßt werden können.
Die Vereinfachung der Schaltfunktion mit dem K-Diagramm kann auch dadurch erfolgen, daß die Nullen zu Blöcken zusammengefaßt werden (Vertauschung der Rollen von $\wedge$ und $\vee$).
Wenn eine Wertetabelle die Schaltfunktion nicht vollständig spezifiziert, dann wird für die unspezifizierten Werte in die entsprechenden Felder ein „X“ eingetragen. Diese Felder können dann für die Blockbildung mit „0“ oder „L“ belegt werden.

1.4.2 Vereinfachung von Schaltfunktionen mit Hilfe von Quine-McClusky

Problem bei K-Diagrammen: unübersichtlich bei mehr als 4 Variablen. Quine-McClusky arbeitet mit Tabellen und ist daher leicht auf den Rechner übertragbar. Sinnvolle Anwendung bis $n = 12$ Variablen.

1.4.2.1 Grundprinzip

Suche nach Mintermen in der Wertetabelle, die sich um die Negation einer Variable x_i unterscheiden. Diese Variable wird elemeniert gemäß der Vereinfachungsregel
$(x_1 \wedge \ldots \wedge x_i \wedge \ldots \wedge x_n) \vee (x_1 \wedge \ldots \wedge \overline{x_i} \wedge \ldots \wedge x_n) = (x_1 \wedge \ldots \wedge x_{i-1} \wedge x_{i+1} \wedge \ldots \wedge x_n)$

1.4.2.2 Beispiel

x_1	x_2	x_3	$f(x_1, x_2, x_3)$	
L	0	L	L	$\Rightarrow L\ 0\ -$
L	0	0	L	

Das entspricht also der Schaltfunktion:
$f(x_1, x_2, x_3) = (x_1 \wedge \overline{x_2} \wedge x_3) \vee (x_1 \wedge \overline{x_2} \wedge \overline{x_3}) = (x_1 \wedge \overline{x_2})$

1.4.2.3 Systematisches Vorgehen

1. Aus der Wahrheitstabelle wird eine Tabelle derart aufgebaut, daß Gruppen aus Mintermen entstehen, die sich nicht in der Anzahl nicht-negierter Variablen unterscheiden.
2. Verschmelzung jeweils zweier Minterme aus unterschiedlichen Gruppen, die sich nur um die Negation einer Variablen unterscheiden.
3. Kennzeichnung nicht weiter zusammenfaßbarer Terme (**Primimplikanden**).
4. Wiederholung von 2. und 3., bis keine weitere Vereinfachung mehr möglich ist.
5. Disjunktive Verknüpfung aller Primimplikanden ergibt die vereinfachte Schaltfunktion.

1.4.2.4 Beispiel

Wahrheitstabelle:

Dez.	x_4	x_3	x_2	x_1	f
0	0	0	0	0	L
1	0	0	0	L	L
2	0	0	L	0	L
3	0	0	L	L	L
4	0	L	0	0	0
5	0	L	0	L	0
6	0	L	L	0	0
7	0	L	L	L	L
8	L	0	0	0	L
9	L	0	0	L	L
10	L	0	L	0	L
11	L	0	L	L	0
12	L	L	0	0	0
13	L	L	0	L	0
14	L	L	L	0	L
15	L	L	L	L	L

1. Schritt: Umordnung der Wahrheitstabelle nach Gruppen:

0	0	0	0	0	keine nicht negierte Variable
1	0	0	0	L	eine nicht negierte Variable
2	0	0	L	0	
8	L	0	0	0	
3	0	0	L	L	zwei nicht negierte Variablen
9	L	0	0	L	
10	L	0	L	0	
7	0	L	L	L	drei nicht negierte Variablen
14	L	L	L	0	
15	L	L	L	L	vier nicht negierte Variablen

2. Schritt: Verschmelzung A:

Dez.	x_4	x_3	x_2	x_1
0, 1	0	0	0	–
0, 2	0	0	–	0
0, 8	–	0	0	0
1, 3	0	0	–	L
1, 9	–	0	0	L
2, 3	0	0	L	–
2, 10	–	0	L	0
8, 9	L	0	0	–
8, 10	L	0	–	0
3, 7	0	–	L	L
10, 14	L	–	L	0
7, 15	–	L	L	L
14, 15	L	L	L	–

3. Schritt: Verschmelzung B:

0, 1, 2, 3	0	0	–	–
0, 1, 8, 9	–	0	0	–
0, 2, 8, 10	–	0	–	0
3, 7	0	–	L	L
10, 14	L	–	L	0

$7, 15$	$-$	L	L	L
$14, 15$	L	L	L	$-$

Hieraus ergibt sich die Normalform. Die Anzahl der Terme entspricht der Anzahl der Zeilen.
Also Schaltfunktion:
$f(x_1, x_2, x_3, x_4) = (\overline{x_4} \wedge \overline{x_3}) \vee (\overline{x_3} \wedge \overline{x_2}) \vee (\overline{x_3} \wedge \overline{x_1}) \vee (\overline{x_4} \wedge x_2 \wedge x_1) \vee (x_4 \wedge x_2 \wedge \overline{x_1}) \vee (x_3 \wedge x_2 \wedge x_1) \vee (x_4 \wedge x_3 \wedge x_2)$
Weitere Vereinfachung mit sogenannten Primimplikatorentabellen (siehe Literatur).

1.5 Synthese von Schaltwerken

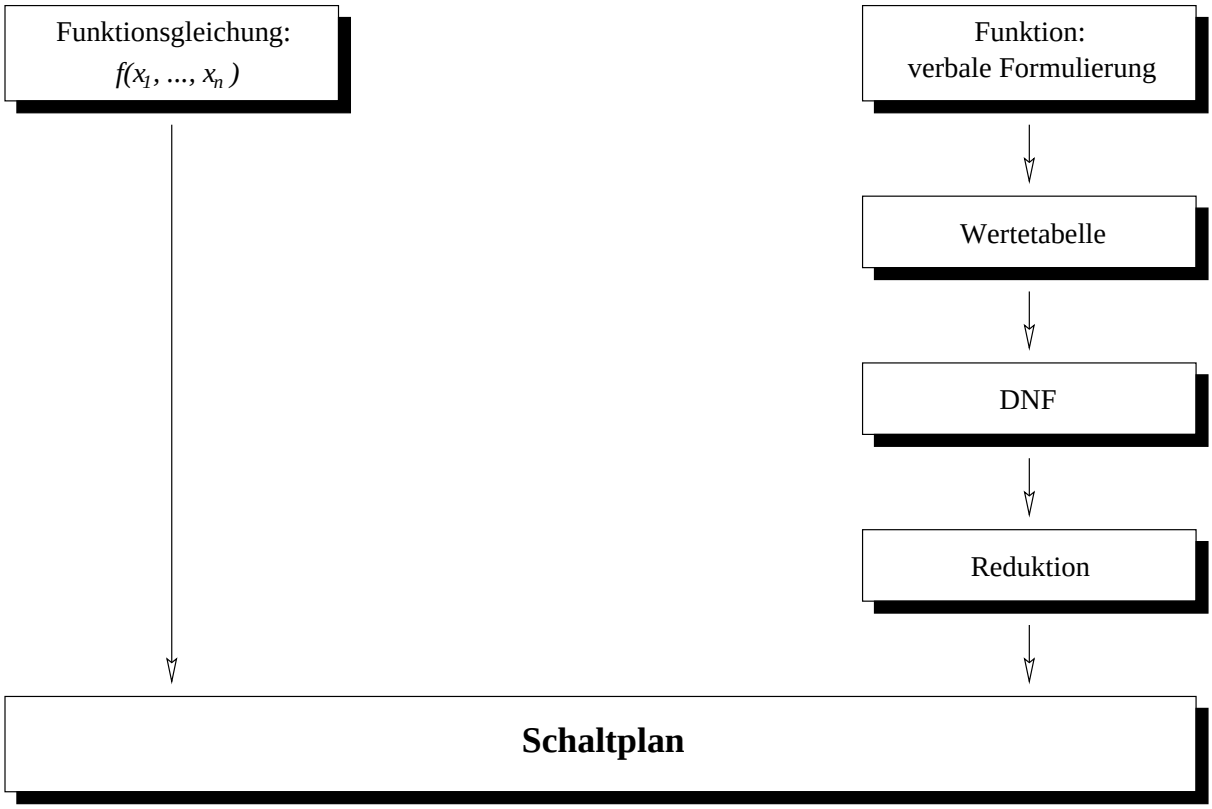


Abbildung 1.1: Synthese von Schaltwerken

1.5.1 Kombinatorischer Automat (Schaltnetz)

1.5.1.1 Vorbemerkung

Schaltnetze, die eine bestimmte Menge von Eingangswertekombinationen in eine Menge von Ausgangskombinationen abbilden, heißen **kombinatorische Automaten**. Die Verknüpfungen $\vee, \wedge, \neg$ heißen **elementare kombinatorische Automaten**.

1.5.1.2 Definitionen

1. Die Menge $X = \mathbb{B}^n = \{0, L\}^n$ mit den Buchstaben $x = (x_1, \ldots, x_n)$ mit $x_i \in \{0, L\}$ und $i = 1 \ldots n$ heißt **Eingabealphabet**.

2. Die Menge $Y = \mathbb{B}^m$ mit den Buchstaben $y = (y_1, \ldots, y_m)$ mit $y_i \in \{0, L\}$ und $i = 1 \ldots m$ heißt **Ausgabealphabet**.
3. Die Abbildung $\Phi : X \to Y,\ \Phi(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$ heißt **Alphabetabbildung**. Sie ordnet jedem $x \in X$ ein $y \in Y$ zu.
4. Das Eingabealphabet $X = \mathbb{B}^n$ und das Ausgabealphabet $Y = \mathbb{B}$ zusammen mit der einfachen Alphabetbildung $\varphi(x_1, \ldots, x_n) = y$ heißt **einfacher kombinatorischer Automat** $A = (x, y, \varphi)$.
5. $X = \mathbb{B}^n$ und $Y = \mathbb{B}^m$ zusammen mit Φ heißt **kombinatorischer Automat** $A = (X, Y, \Phi)$.

1.5.1.3 Sätze

1. Jeder einfache kombinatorische Automat (x, y, φ) kann durch Zusammenschaltung von Elementarautomaten ($\wedge, \vee, \neg$-Gatter) realisiert werden (Beweis über DNF).
2. Jeder kombinatorische Automat (X, Y, Φ) kann durch m einfache Automaten erzeugt werden.

1.5.1.4 Beispiele für kombinatorische Automaten

1.5.1.4.1 Exklusiv-Oder (Antivalenz)

Eingabealphabet $X = \mathbb{B}^2$, Ausgabealphabet $Y = \mathbb{B}^1$
$y = \Phi(x_1, x_2) = \varphi(x_1, x_2) = (x_1 \wedge \overline{x_2}) \vee (\overline{x_1} \wedge x_2)$

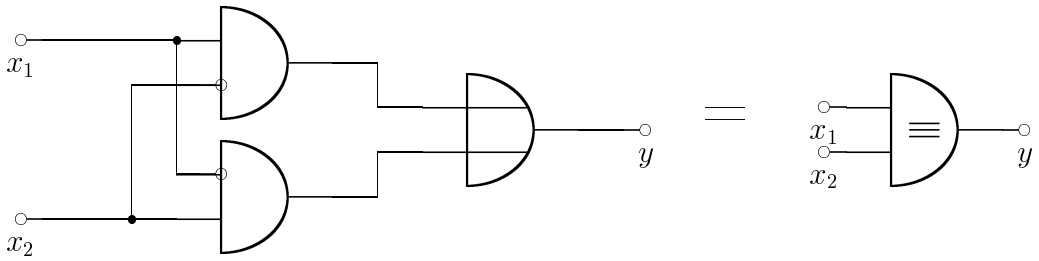


Abbildung 1.2: Exklusiv-Oder als Beispiel für einen kombinatorischen Automaten

1.5.1.4.2 Kombinatorischer Automat aus m Einzelautomaten

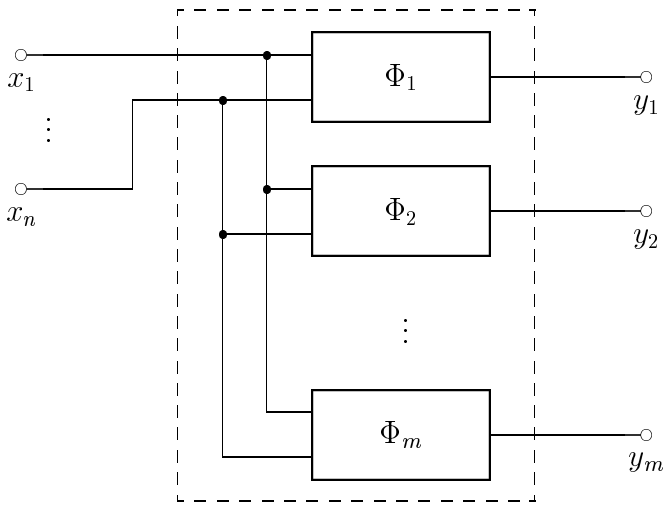


Abbildung 1.3: Kombinatorischer Automat aus m Einzelautomaten

1.5.1.4.3 Halbaddierer: Schaltung zur Addition zweier Dualzahlen mit Übertragerzeugung

x_2	x_1	y_2	y_1
0	0	0	0
0	L	0	L
L	0	0	L
L	L	L	0

Also Schaltung: $\Phi : \mathbb{B}^2 \to \mathbb{B}^2$, mit $\Phi(x_1, x_2) = (y_1, y_2)$

Man erkennt sofort aus Wertetabelle:
$y_1 = (x_1 \wedge \overline{x_2}) \vee (\overline{x_1} \wedge x_2)$ und $y_2 = (x_1 \wedge x_2)$

Die zugehörige Schaltung ist:

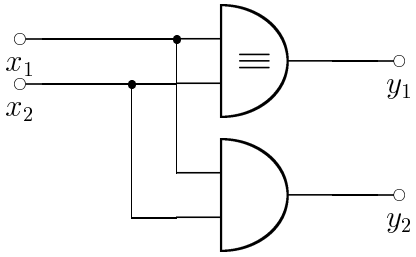


Abbildung 1.4: Halbaddierer

1.5.2 Sequentielle Automaten (Schaltwerke) bzw. sequentielle Schaltnetze

Beim kombinatorischen Automaten ist die Ausgabe $y(t)$ zu einem bestimmten Zeitpunkt t nur abhängig von der Eingabe $x(t)$ zum selben Zeitpunkt (abgesehen von Signallaufzeiten). Feste Zuordnung von y zu x. Erweiterung der möglichen Abbildungen $x \to y$ beim sequentiellen Automaten dadurch, daß $y(t)$ nicht nur von der Eingabe $x(t)$, sondern außerdem noch von allen Eingaben zu früheren Zeitpunkten $x(t-T)$, $x(t-2T)$, ... abhängt.
Dazu ist es erforderlich, daß der sequentielle Automat sich die früheren Eingaben merkt, also über **Speicher** verfügt. Das Ausgabewort zum Zeitpunkt t hängt damit nicht nur von der Eingabegröße, sondern auch von den Zuständen, d. h. den Inhalten der Speicher zu diesem Zeitpunkt ab.

1.5.2.1 Bemerkung zur Taktsteuerung

Prinzipiell ist für T jeder Wert > 0 denkbar. Normalerweise wird T deutlich größer gewählt als die Einspeicherungszeit der Bauelemente in der Schaltung. Außerdem bleibt T konstant (feste Taktfolge). Damit erfolgen alle Zustandsänderungen in einem fest definierten Zeitraster (synchron).

1.5.2.1.1 Vorteile

1. Störungen, welche außerhalb des Taktrasters eintreffen (nur dort sind Zustandswechsel möglich), haben keinen Einfluß.
2. Störeffekte, die innerhalb der Schaltung durch unterschiedliche Laufzeiten auftreten, haben ebenfalls keinen Einfluß.

1.5.2.1.2 Beispiel

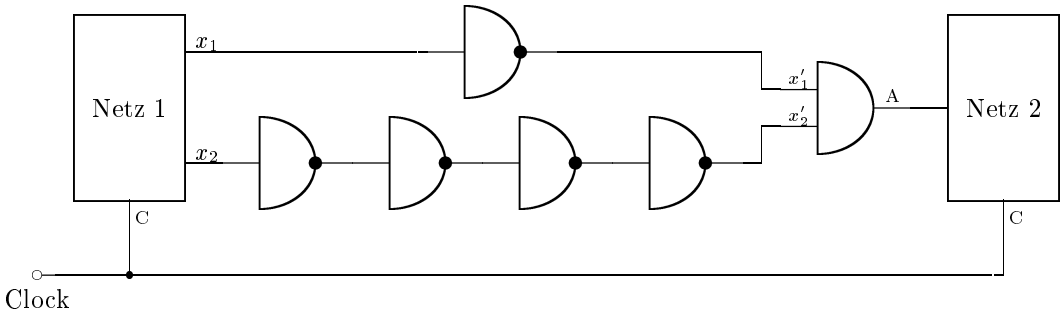


Abbildung 1.5: Taktsteuerung

Impulsdiagramm ab dem Zeitpunkt $t = 0$, wo $\left.\begin{array}{l} x_1 : L \to 0 \\ x_2 : L \to 0 \end{array}\right\}$ Übergänge zum Zeitpunkt $t = 0$:

Offensichtlich: Ein System, daß mit aufsteigender Taktflanke arbeitet, wird nicht gestört. Schaltwerk hat einen Speicher im Gegensatz zu Schaltnetzen.

1.5.2.2 Drei Arten der Taktsteuerung

1. Taktzustandssteuerung
2. Taktflankensteuerung
3. Zweizustands- oder Zweiflankensteuerung

Bei der Taktzustandssteuerung konjunktive Verknüpfung vom Taktsignal C und der Eingangsvariablen: Variablenänderungen nur während $C = L$.
Um die Zeit zu verkürzen, wird Zustandswechsel nur akzeptiert, wenn C den Pegel ändert: Taktflankensteuerung.
Wenn die Ausgänge eines getakteten Gliedes schalten, reicht unter Umständen die Zeit zur Übernahme nicht aus: Zweiflankensteuerung mit zwei zeitversetzten Takten, oder S_1 wird mit aufsteigender, S_2 mit absteigender Flanke getriggert.

1.5.2.3 Speicher und Rückführung

Das Ausgabewort des elementaren Speichers ist das um einen Takt verzögerte Eingabewort. Der Speicher merkt sich also die Eingabe über die Dauer T.
$y(k) = x(k-1)$, wobei k der Zeitindex $t = k \cdot T$ ist und $f = \frac{1}{T}$ die sogenannte Taktfrequenz bezeichnet.

$\frac{t}{T} = k$	0	1	2	$\cdots$	k	$k+1$
x	$x(0)$	$x(1)$	$x(2)$	$\cdots$	$x(k)$	$x(k+1)$
y	$y(0)$	$x(0)$	$x(1)$	$\cdots$	$x(k-1)$	$x(k)$

Wegen seines Verhaltens wird der Speicher in Schaltwerken auch als Verzögerungsglied bezeichnet.
Durch dieses verzögernde Verhalten des Speichers sind nun Rückkopplungen möglich, die bei asynchronen Netzen zu Instabilitäten (sogenannten „Flimmerschaltungen“) führen würden.

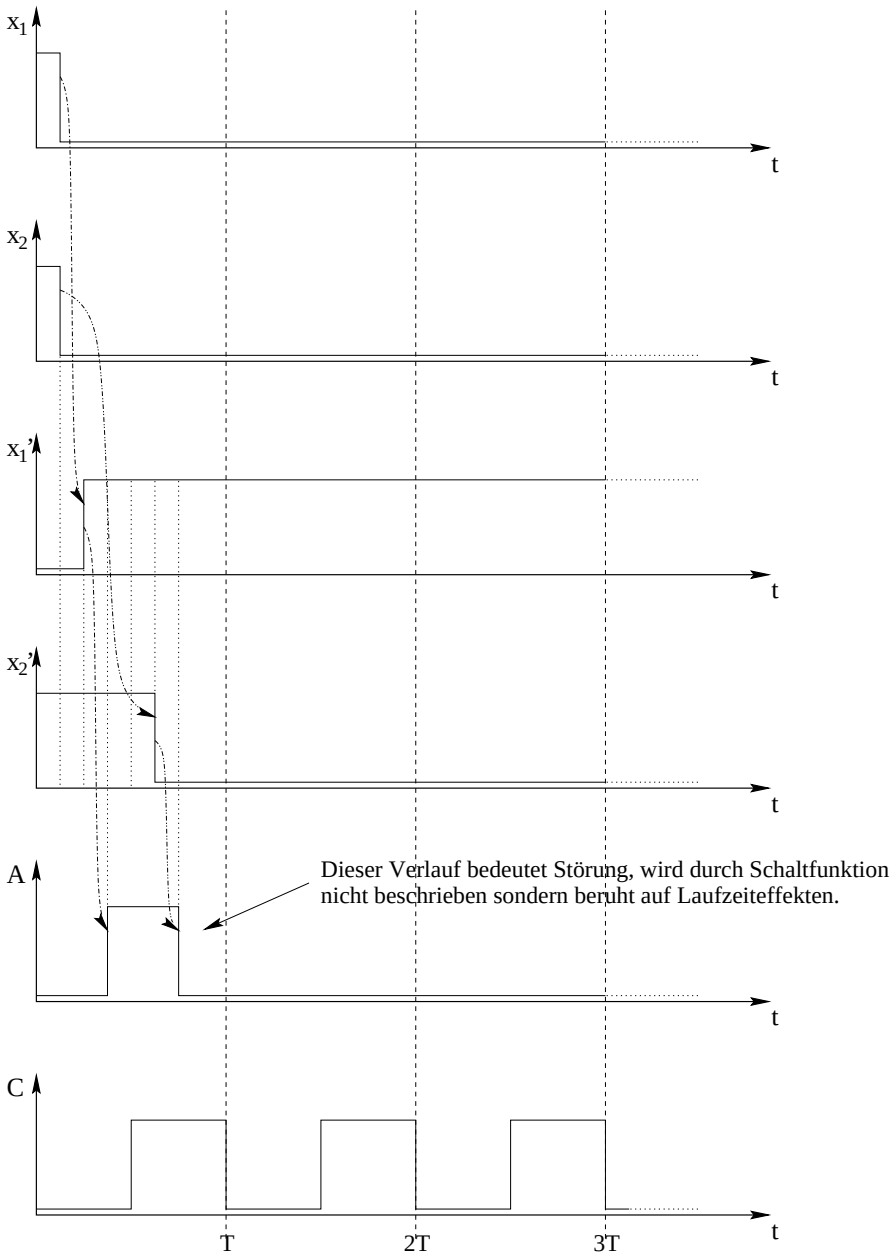


Abbildung 1.6: Impulsdiagramm zur Taktsteuerungsschaltung (Schaltzeit z. B. 10 ns)

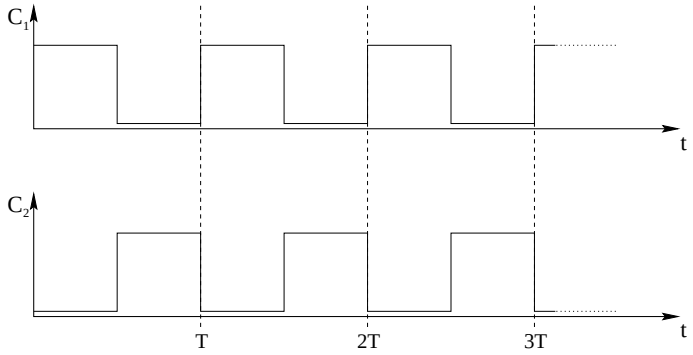


Abbildung 1.7: Impulsdiagramm: Zweiflankensteuerung

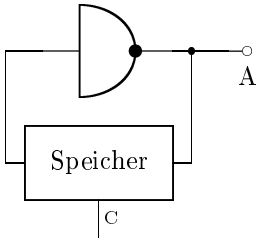


Abbildung 1.8: Takthalbierer

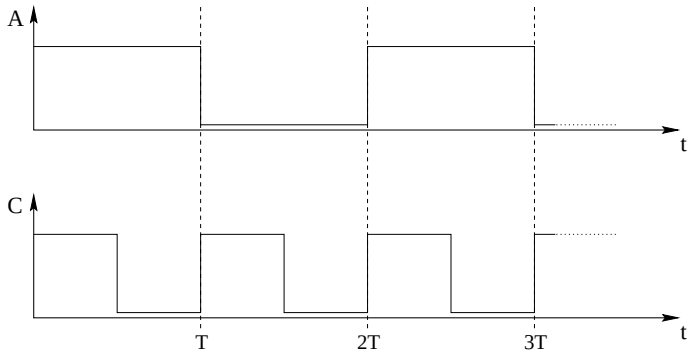


Abbildung 1.9: Impulsdiagramm: Takthalbierer

1.5.2.3.1 Beispiel „Takthalbierer“

Wird ein Schaltnetz mit Speichern kombiniert, dann ist es im allgemeinen nicht möglich, sofort einen direkten Zusammenhang zwischen Eingang und Ausgang anzugeben.
Grund: Innere Rückführungen über Speicherglieder.

1.5.2.3.2 Beispiel: Bestimmung des Ausgangswortes der folgenden Schaltung

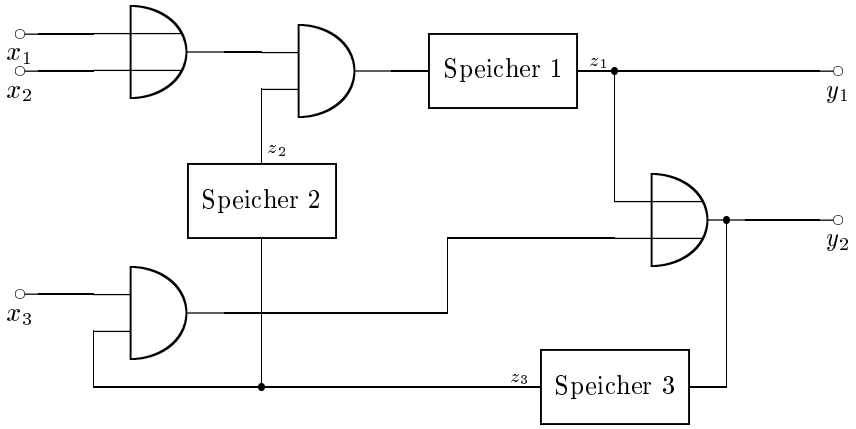


Abbildung 1.10: Schaltnetz mit Speichern

Eingabewort: $\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$, Ausgabewort: $\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$, Zustandswort: $\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$

Zur Analyse der Schaltung: Einführung von Hilfsvariablen z_i an jedem Speicherausgang. Damit kann man sofort das folgende Gleichungssystem aus der Schaltung ablesen:

$$\begin{array}{ll} z_1(k+1) & = z_2(k) \wedge (x_1(k) \vee x_2(k)) \\ z_2(k+1) & = z_3(k) \\ z_3(k+1) & = z_1(k) \vee (x_3(k) \wedge z_3(k)) \\ \hline y_1(k) & = z_1(k) \\ y_2(k) & = z_1(k) \vee (x_3(k) \wedge z_3(k)) \end{array}$$

Sei beispielsweise die Folge von Eingangswörtern $\begin{pmatrix} L \\ L \\ 0 \end{pmatrix}$, $\begin{pmatrix} 0 \\ 0 \\ L \end{pmatrix}$, $\begin{pmatrix} 0 \\ L \\ L \end{pmatrix}$ und die Speicherinhalte für $k = 0$ (Anfangsbedingungen) $z = \begin{pmatrix} 0 \\ L \\ 0 \end{pmatrix}$, dann läßt sich aus dem Gleichungssystem die folgende Tabelle erzeugen:

k	0	1	2	3	$\cdots$
x_1	L	0	0	$\cdots$	
x_2	L	0	L	$\cdots$	
x_3	0	L	L	$\cdots$	
z_1	0	L	0	$\cdots$	
z_2	L	0	0	$\cdots$	
z_3	0	0	L	$\cdots$	
y_1	0	L	0	$\cdots$	
y_2	0	L	L	$\cdots$	

1.5.2.4 Allgemeiner sequentieller Automat

Im allgemeinen hat ein sequentieller Automat n Eingänge, m Ausgänge und q Speicher. Er wird durch das folgende Gleichungssystem beschrieben:

$$\begin{aligned}
z_1(k+1) &= f_1(z_1(k),\ldots,z_q(k);x_1(k),\ldots,x_n(k)) \\
&\vdots \qquad\qquad\qquad\qquad \vdots \\
z_q(k+1) &= f_q(z_1(k),\ldots,z_q(k);x_1(k),\ldots,x_n(k)) \\
y_1(k) &= g_1(z_1(k),\ldots,z_q(k);x_1(k),\ldots,x_n(k)) \\
&\vdots \qquad\qquad\qquad\qquad \vdots \\
y_m(k) &= g_m(z_1(k),\ldots,z_q(k);x_1(k),\ldots,x_n(k))
\end{aligned}$$

Damit ist die vollständige Angabe des Folgezustands und der Ausgabe eines endlichen binären Zustandsautomaten, d. h. eines Automaten mit endlich vielen Zuständen (2^q Zustände), möglich durch die Angabe

- seines Zustandswortes $z(k) = (z_1(k),\ldots,z_q(k)) \in \{0,L\}^q$
- seines Eingabewortes $x(k) = (x_1(k),\ldots,x_n(k)) \in \{0,L\}^n$
- seiner Ausgabefunktionen $g_j : \mathbb{B}^q \times \mathbb{B}^n \to \mathbb{B}$ mit endlich vielen $g_j(z,x) = y_j$ und $j = 1\ldots m$ und
- seiner Übergangsfunktion $f_i : \mathbb{B}^q \times \mathbb{B}^n \to \mathbb{B},\ z_i' = f_i(z,x),\ i = 1\ldots q$ mit $z_i' = z_i(k+1)$

Allgemein wird die Menge aller Eingangsworte $X = \mathbb{B}^n$ (Eingabealphabet) zusammen mit dem Zustandsalphabet $Z = \mathbb{B}^q$ und dem Ausgabealphabet $Y = \mathbb{B}^m$ mit den Abbildungen $f_i : Z \times X \to Z$ mit $f_i(x,z) = z_i',\ i = 1\ldots q$ und $g_i : Z \times X \to Y$ mit $g_i(x,z) = y_i$ als **binärer endlicher Zustandsautomat** (X, Y, Z, f, g) bezeichnet, wo f und g für die Menge aller Übergangs- bzw. Ausgabefunktionen stehen. Ein solcher Automat wird auch als binärer **Mealy-Automat** bezeichnet. Ein wichtiger Spezialfall ist der **Moore-Automat**, bei dem die Ausgabe nicht direkt von der Eingabe beeinflußt wird, sondern nur von dem Zustand, bei dem $g_i = f(z_1,\ldots,z_q)$ gilt.

1.5.2.5 Möglichkeit zur Realisierung der Automaten

Direkte Umsetzung in kombinatorisches Netzwerk unter Verwendung von Speichern.

„Normalform“ der Mealy-Automaten: f und g sind kombinatorische Automaten aus q bzw. m Schaltfunktionen. Jeder sequentielle Automat kann durch zwei kombinatorische Automaten mit den Abbildungsfunktionen f und g, sowie q Speichern realisiert werden.

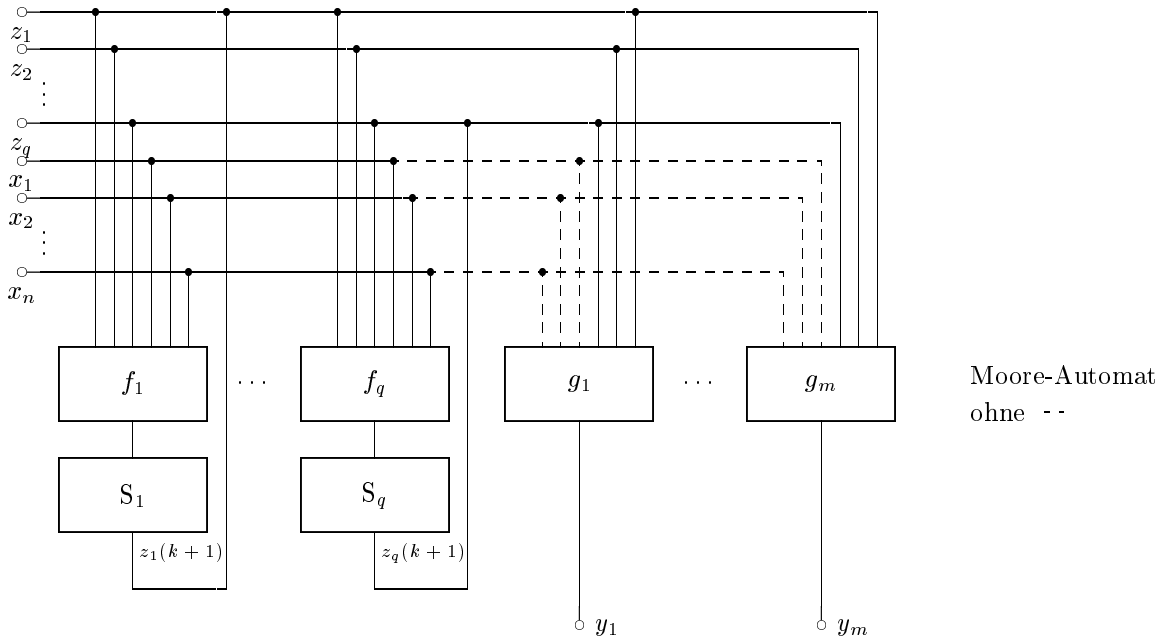


Abbildung 1.11: kombinatorisches Netzwerk unter Verwendung von Speichern

1.5.2.6 Darstellung des Verhalten von Automaten

1.5.2.6.1 Wertetabelle

Für jeden Zustand z werden in 2^n Zeilen alle möglichen Wertekombinationen von Eingangsvariablen eingetragen. In jeder Zeile werden ferner die q Folgezustände und die m Ausgaben notiert.

z	x	y	z'
0	0	$y_1(z,x),\ldots,y_m(z,x)$	$f_1(z,x),\ldots,f_q(z,x)$
0	1		
⋮	⋮	⋮	⋮
0	2^n-1		
⋮	⋮	⋮	⋮
2^q-1	0		
⋮	⋮	⋮	⋮
2^q-1	2^n-1		

1.5.2.6.2 Beispiel: Ampelschaltung

System soll realisiert werden mit zwei Zustandsspeichern für Fahrtrichtung 1 und Fahrtrichtung 2.

1.5.2.6.2.1 Verhalten

Normalerweise sind beide Fahrtrichtungen blockiert und Fußgänger haben grün. Wenn ein Fahrzeug auf einer der beiden Kontaktschleifen erkannt, wird die jeweilige Ampel für eine Taktphase auf grün geschaltet, danach wieder auf rot.

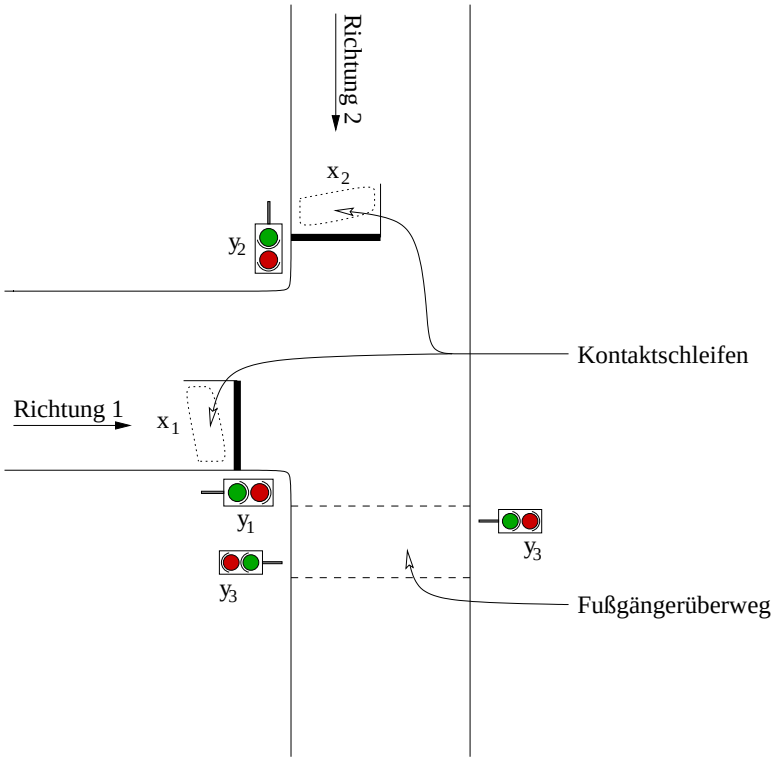


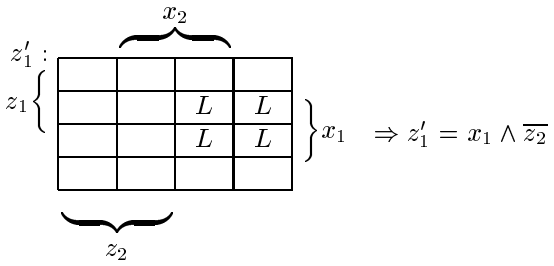
Abbildung 1.12: Ampelschaltung (für y_1, y_2 und y_3 gelte: rot $= 0$, grün $= L$)

Zustandswort: $\begin{pmatrix} \text{Richtung 1 frei} \\ \text{Richtung 2 frei} \end{pmatrix}$, Eingabewort: $\begin{pmatrix} \text{Fahrzeug auf Schleife 1} \\ \text{Fahrzeug auf Schleife 2} \end{pmatrix}$, Ausgabewort: $\begin{pmatrix} \text{1 grün} \\ \text{2 grün} \\ \text{Fußgänger grün} \end{pmatrix}$

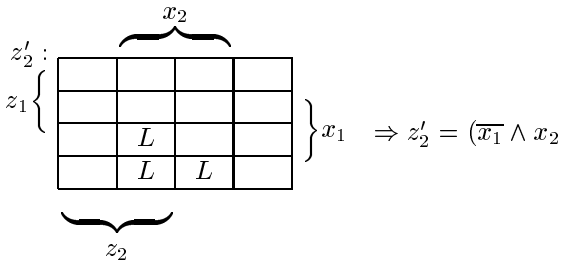
z		x		y			z'	
z_1	z_2	x_1	x_2	y_1	y_2	y_3	$z'_1 = f_1(x,z)$	$z'_2 = f_2(x,z)$
0	0	0	0	0	0	L	0	0
0	0	0	L	0	0	0	0	L
0	0	L	0	0	0	0	L	0
0	0	L	L	0	0	0	L	0
0	L	0	0	0	L	0	0	0
0	L	0	L	0	L	0	0	L
0	L	L	0	0	L	0	0	0
0	L	L	L	0	L	0	0	L
L	0	0	0	L	0	0	0	0
L	0	0	L	L	0	0	0	0
L	0	L	0	L	0	0	L	0
L	0	L	L	L	0	0	L	0

$z = \begin{pmatrix} L \\ L \end{pmatrix}$ ist aus Sicherheitsgründen verboten.

1.5.2.6.2.2 Automatenentwurf durch Berechnung der jeweiligen Übergangsfunktion



$\Rightarrow z_1' = x_1 \wedge \overline{z_2}$



$\Rightarrow z_2' = (\overline{x_1} \wedge x_2 \wedge \overline{z_1}) \vee (x_2 \wedge \overline{z_1} \wedge z_2)$

Schließlich ist offensichtlich: $y_3 = \overline{x_1} \wedge \overline{x_2} \wedge \overline{z_1} \wedge \overline{z_2}$

1.5.2.6.2.3 Schaltung für die Steuerung der Ampelanlage

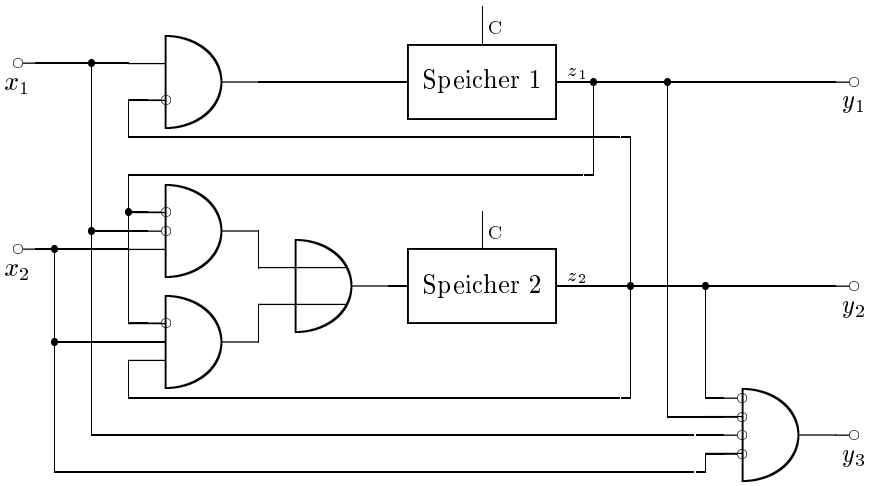


Abbildung 1.13: Schaltung für die Steuerung der Ampelanlage

1.5.2.6.2.4 Automatengraph

- Jedem Zustand z wird ein Kreis zugeordnet, in den das zugehörige Zustandswort geschrieben wird.
- Befindet sich der Automat in z und wird durch Eingabe von x in den Zustand z' überführt, dann wird z mit z' durch einen Pfeil verbunden und x an die Pfeilkante geschrieben.

Beispiel Ampel:

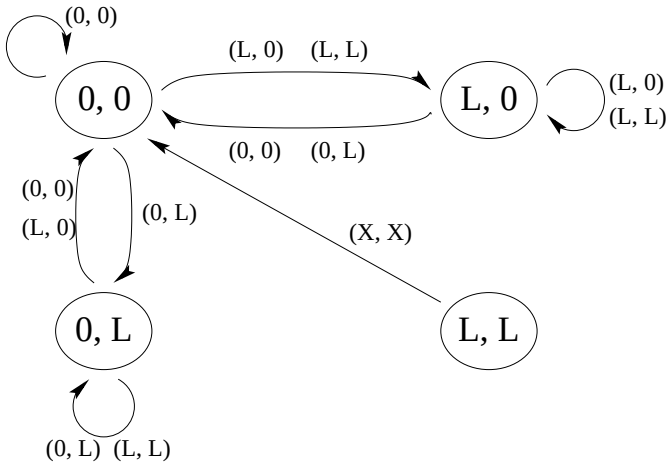


Abbildung 1.14: Automatengraph zur Ampelschaltung

1.6 Speicherbauelemente

Schaltkreise, die zwei Zustände speichern können, werden Flip-Flops (oder bistabile Bauelemente) genannt.

1.6.1 Einfaches Flip-Flop: RS-Flip-Flop

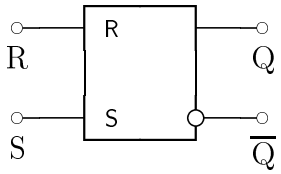


Abbildung 1.15: RS-Flip-Flop

S	R	Q	Q^+	
0	0	0	0	
0	L	0	0	
L	0	0	L	
L	L	0	$\times$	$\leftarrow$ undefiniert
0	0	L	L	
0	L	L	0	
L	0	L	L	
L	L	L	$\times$	$\leftarrow$ undefiniert

Das FF wird durch Anlegen von L an S gesetzt ($Q = L$) und durch Anlegen von L an R zurückgesetzt ($Q = 0$). Gleichzeitiges Aktivieren von $S = L$ und $R = L$ ist verboten und führt zu undefiniertem Zustand $Q = \times$.

1.6.1.1 Realisierungsmöglichkeiten

Zwei NOR-Gatter mit gekreuzten Eingängen.

Asynchrones Verhalten: Bauelement kann jederzeit seinen Zustand wechseln.

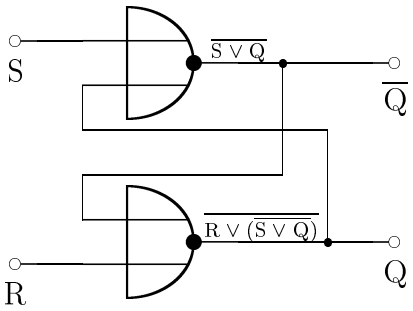


Abbildung 1.16: Zwei NOR-Gatter mit gekreuzten Eingängen

1.6.2 Taktzustandsgesteuertes RS-Flip-Flop

Eigenschaft: Nur wenn C = L, ist Zustandswechsel möglich.

Schaltung:

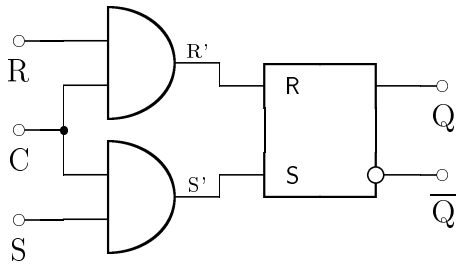


Abbildung 1.17: Taktzustandsgesteuertes RS-Flip-Flop

Kein Wechsel, solange C = 0 (Störunterdrückung).

1.6.3 Taktflankengesteuertes RS-Flip-Flop

Beim flankengesteuerten RS-Flip-Flop wird die Zustandsänderung nur während der aufsteigenden Flanke von C akzeptiert.
Realisierung über **Laufzeiten** von Gattern:

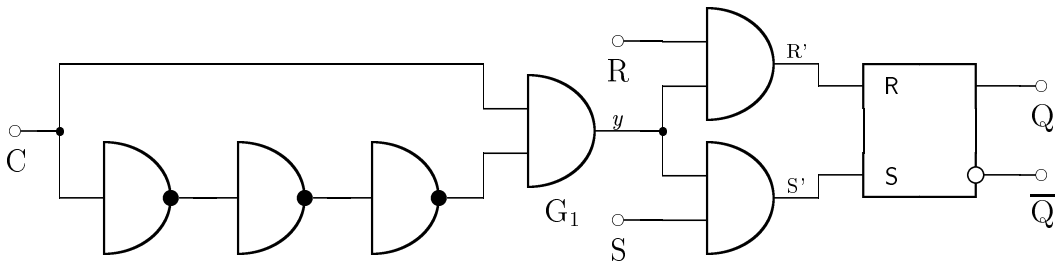


Abbildung 1.18: Taktflankengesteuertes RS-Flip-Flop

Wenn C von 0 nach L wechselt, dann erscheint an beiden Eingängen von G_1 kurzzeitig L mit der Folge $y = L$. Dies dauert an, bis der Pegel L über die drei lediglich zur Verzögerung dienenden Inverter gelaufen ist. Nur in dieser Zeit werden Zustandswechsel akzeptiert.

1.6.4 Master-Slave Flip-Flop

Bei Zweiflankensteuerung ist die Entkopplung von Zustandsübernahme und -ausgabe notwendig. In diesem Fall werden zwei zustands- oder flankengetriggerte Flip-Flop hintereinander geschaltet (master und Slave). Das Master-FF übernimmt z. B. mit ansteigender, das Slave-FF mit abfallender Flanke. Im Fall der Zustandssteuerung:

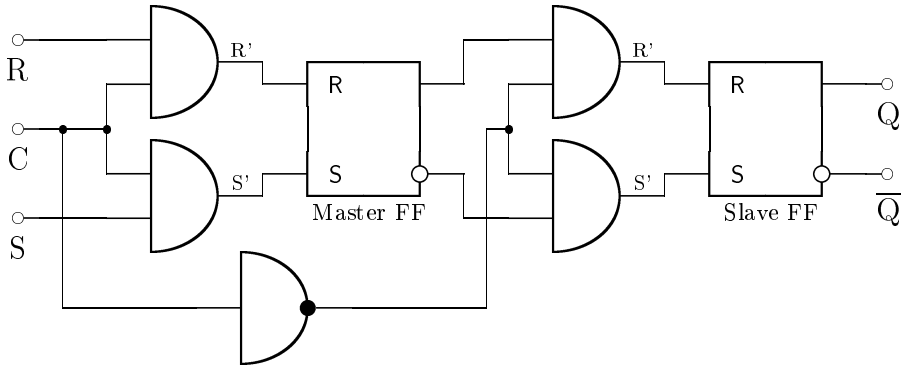


Abbildung 1.19: Master-Slave Flip-Flop

1.6.5 Jump/Kill Flip-Flop

Das JK (Jump/Kill)-FF ist das am häufigsten benutzte Flip-Flop. Es besteht aus einem RS-Flip-Flop, dessen Eingänge so gegeneinander verriegelt sind, daß J = K = L erlaubt ist. In diesem Fall erfolgt bei jedem Takt ein Kippen in die jeweils andere Lage.

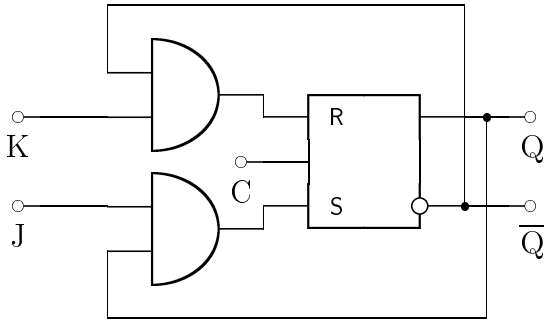


Abbildung 1.20: Jump/Kill Flip-Flop

Entsprechend kann ein Jump/Kill-Master-Slave Flip-Flop aufgebaut werden für Zweiflankensteuerung, bei der die Übernahme des Zustands zeitlich von der Änderung des Ausgangs entkoppelt ist.

1.6.6 Jump/Kill-Master-Slave Flip-Flop

Tabelle für die Zustandswechsel:

J	K	Q	Q^+
0	0	q_n	q_n
0	L	×	0
L	0	×	L
L	L	q_n	$\overline{q_n}$

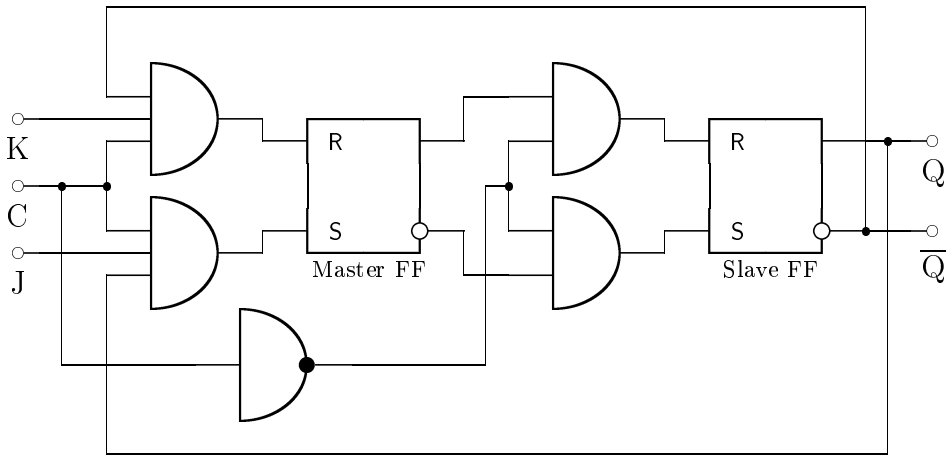


Abbildung 1.21: Jump/Kill-Master-Slave Flip-Flop

q_n: Pegel von Q zum Zeitpunkt n.

1.7 Elektrische Realisierung von Gattern

Alle für kombinatorische oder sequentielle Netzwerke erforderlichen Komponenten lassen sich über die bereits eingeführten elementaren Gatterfunktion realisieren.
Wie lassen sich Gatterfunktionen so realisieren, daß

- Spannungspegel für 0 und L gut unterscheidbar sind,
- eine hohe Geschwindigkeit erreicht wird und
- die Leistungsaufnahme gering ist?

1.7.1 Realisierung mit elektromechanischen Schaltern (Relais)

1.7.1.1 Beispiel: ODER-Gatter

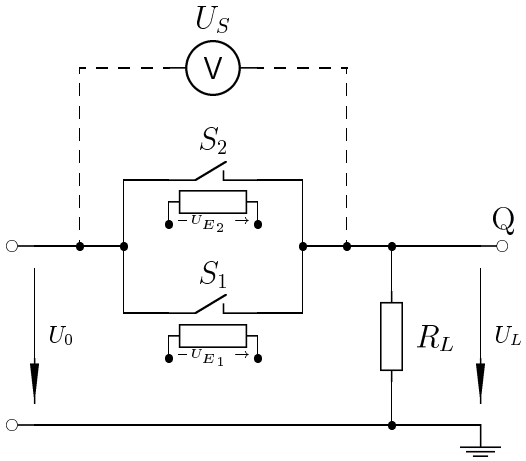


Abbildung 1.22: Durch Relais realisiertes ODER-Gatter

Betriebsspannung U_0, Laststrom $I_L = \frac{U_L}{R_L}$ mit $U_L = U_0 - U_S$, Lastwiderstand R_L (Widerstand des am Ausgang Q angeschlossenen Verbrauchers), Schalter S_1 und S_2 repräsentieren Eingangsvariablen (können mechanisch oder über Magnetspulen betätigt werden), Betriebsspannungen für Relais U_{E_1} und U_{E_2}.

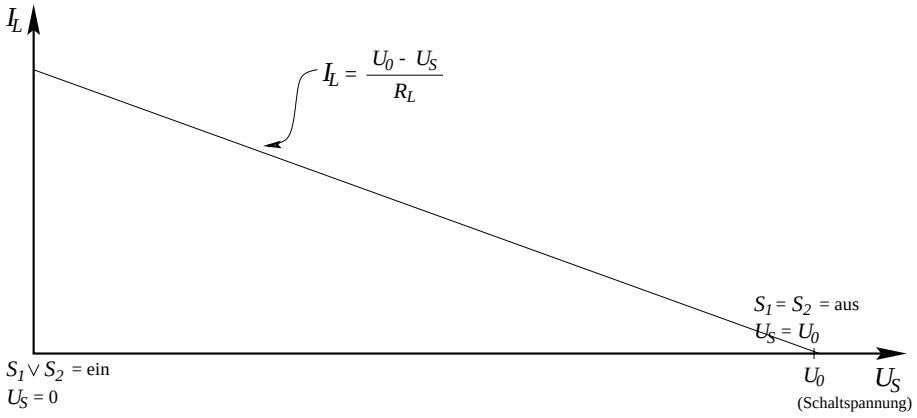


Abbildung 1.23: Schalterstrom-Kennlinie

Verhältnis hier $R_{\text{ein}} = 0$, $R_{\text{aus}} = \infty$, im realen Fall $R_{\text{ein}} > 0 \Rightarrow U_{S\,\text{ein}} > 0$, $R_{\text{aus}} < \infty \Rightarrow U_{S\,\text{aus}} < U_0$.

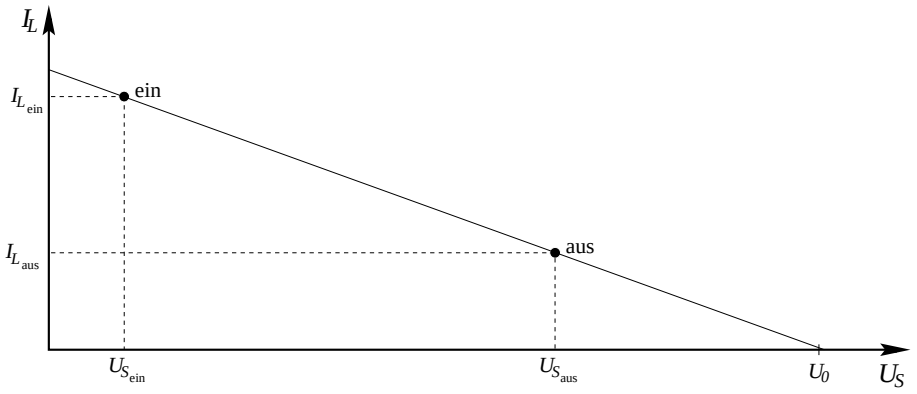


Abbildung 1.24: Schalterstrom-Kennlinie
Im realen Fall ist $I_{L\,\text{ein}}$ kleiner und $I_{L\,\text{aus}}$ größer als im idealen Fall.

1.7.2 Realisierung mit elektronischen Schaltern (Transistoren)

Transistoren können über einen kleinen Steuerstrom einen großen Ausgangsstrom schalten, sind also **Verstärker**. Diese sind außerdem sehr schnell. Allerdings sieht ihre Schaltcharakteristik alles andere als ideal aus ($R_{\text{ein}} \gg 0\,\Omega$ und $R_{\text{aus}} \ll \infty$).

1.7.2.1 Grundschaltung eines Transistorschalters

Eingangsgröße U_E, Ausgangsgröße U_S.

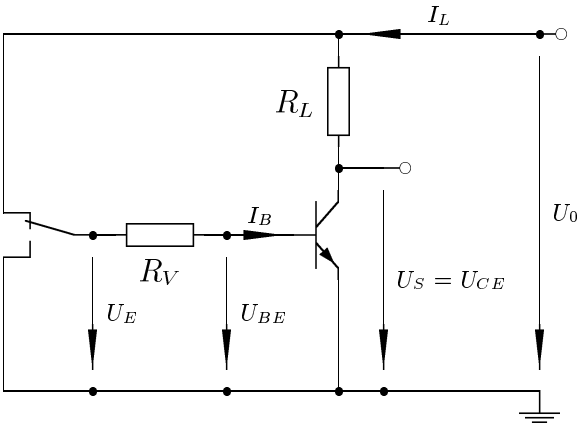


Abbildung 1.25: Grundschaltung eines Transistorschalters

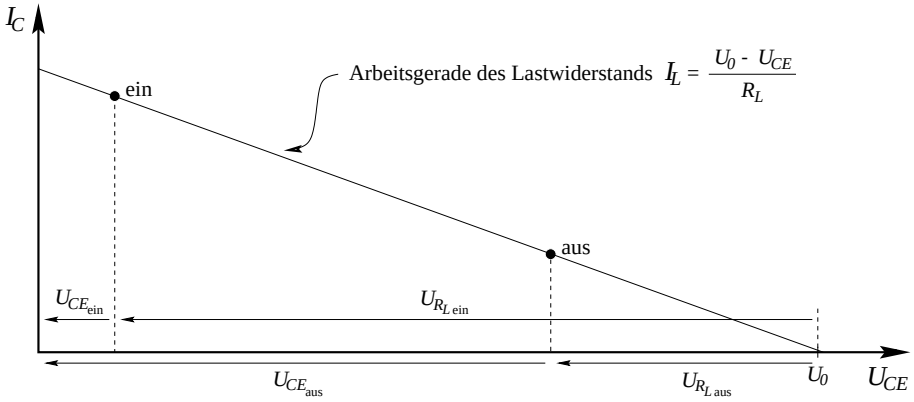


Abbildung 1.26: Schaltcharakteristik

1.7.2.2 Bestimmung der Arbeitspunkte für „ein“ und „aus“

Wenn der Transistor wie der Widerstand ein lineares Bauelement wäre, wenn also ein linearer Zusammenhang zwischen U_{CE} und I_L mit einem konstanten Proportionalitätsfaktor bestünde, dann könnten die beiden Arbeitspunkte durch Schnitt der Arbeitsgeraden von R_L mit den Widerstandsgeraden des Transistors mit den Steigungen $\frac{1}{R_{\text{ein}}}$ und $\frac{1}{R_{\text{aus}}}$ bestimmt werden:

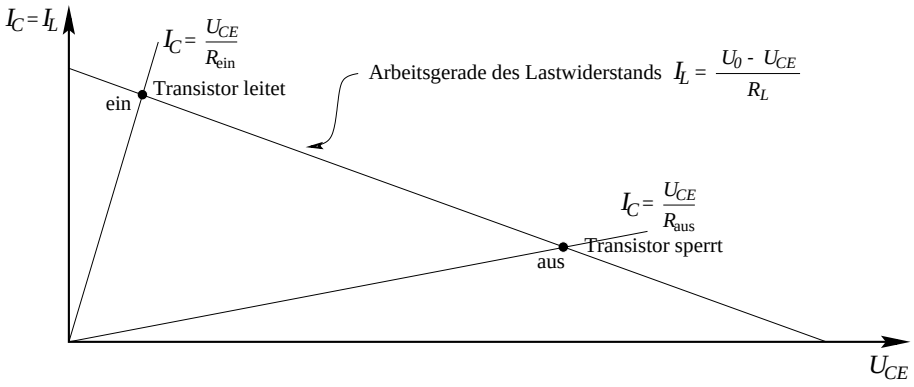


Abbildung 1.27: Arbeitsgerade mit den beiden Arbeitspunkten für „ein“ und „aus“

Man könnte in diesem Fall auf die graphische Ermittlung verzichten. Wenn aber – wie beim Transistor der Fall – ein nichtlinearer Zusammenhang zwischen Kollektorspannung und Kollektorstrom durch das Bauteil besteht, ist nur graphische Ermittlung der Arbeitspunkte möglich. Zuhilfenahme des Kennlinienfelds des Transistors $I_C = f(U_{CE})$.

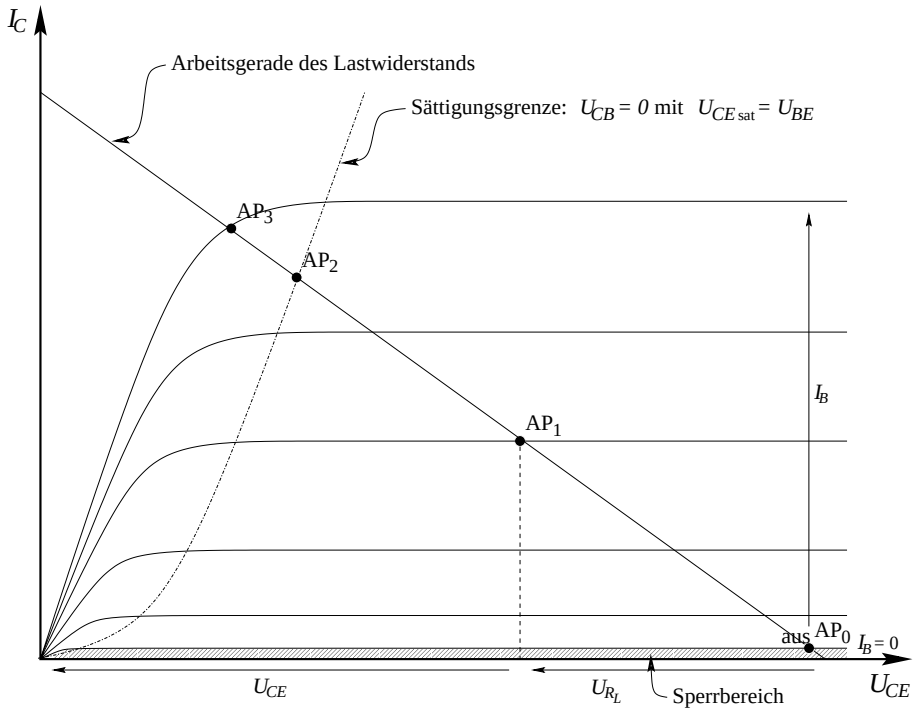


Abbildung 1.28: Kennlinienfeld des Transistors

Übersteuerungsbereich (links von der Sättigungsgrenze): Bei weiterer Vergrößerung von I_B steigt I_C nicht mehr an. Die Basis ist mit Ladungsträgern überschwemmt („gesättigt“), $U_{CE} < U_{CE\,\mathrm{sat}}$.
Aktiver Bereich (rechts von der Sättigungsgrenze): Alle Ladungsträger in der Basis werden vom Kollektor abgesaugt, $U_{CE} > U_{CE\,\mathrm{sat}}$.

Je nach Wahl des Basisstroms I_B ergeben sich unterschiedliche Kollektorströme / Kollektor-Emitter-Spannungen, also unterschiedliche Arbeitspunkte:

AP$_0$: $I_B = 0$
Keine Ladungsträger fließen in die Basis, der Transistor sperrt, es fließt nur Sperrstrom.
$I_{C0} \approx 0,\ U_{CE} \approx U_0$.

AP$_1$: $I_B > 0,\ U_{CE} > U_{CE\,\mathrm{sat}}$
Bei Erhöhung des Basisstroms I_B nimmt I_C ein Stück weit linear zu (mit dem Proportionalitätsfaktor B), d. h. der Arbeitspunkt wandert auf der Arbeitsgerade des Lastwiderstands nach links oben. Der Transistor leitet.
$I_{C0} \ll I_C,\ U_{CE} \ll U_0$.

AP$_2$: $I_B > 0,\ U_{CE} = U_{CE\,\mathrm{sat}}$
Bei weiterer Erhöhung des Basisstroms nimmt U_{CE} immer weiter ab bis $U_{CE} = U_{CE\,\mathrm{sat}}$. Der Transistor ist gesättigt.
$U_{CB} = 0$ und $U_{CE} = U_{CE\,\mathrm{sat}}$.

AP$_3$: $I_B > 0,\ U_{CE} < U_{CE\text{sat}}$
Bei weiterer Erhöhung des Basisstroms nimmt I_C nur noch unwesentlich zu, der Transistor ist übersteuert. U_{CE} sinkt unter $U_{CE\text{sat}}$, es ist $U_{CE} = U_{CE\text{Rest}}$.

Transistoren im Schalterbetrieb werden an der Sättigungsgrenze betrieben, gelegentlich auch im Übersteuerungsbereich.

Vorteil: Viele Ladungsträger kommen beim Einschalten in die Basis → Schneller Abbau der B-C-Sperrschicht durch Kompensation der dafür verantwortlichen (endlich vielen) Ladungsträger. Außerdem geringere Verlustleistung, weil U_{CE} klein.
Aber: Ausschaltungsvorgang relativ langsam, weil es einige Zeit dauert bis die Ladungsträger-überschwemmte Basiszone freigeräumt ist. Entscheidend für den sich einstellenden Arbeitspunkt ist die Größe des Basisvorwiderstandes R_V. R_V sollte möglichst groß sein, damit Ansteuerungsleistung gering. Andererseits muß R_V klein genug sein, um Basisstrom zu erlauben, der den Transistor in den Sättigungszustand führt. $U_{CE} \leq U_{CE\text{sat}}$ (Sättigungszustand) bzw. $U_{CE} = U_{CE\text{Rest}} < U_{CE\text{sat}}$ (Übersteuerungszustand).
An der Sättigungsgrenze gilt:

$$I_B = \frac{U_0 - U_{BE}}{R_V} = \frac{U_0 - U_{CE\text{sat}}}{R_V} \Longrightarrow R_V = \frac{U_0 - U_{CE\text{sat}}}{I_B}$$

mit $I_C = B \cdot I_B$ wird daraus:

$$R_V = B \cdot \frac{U_0 - U_{CE\text{sat}}}{I_C}$$

Nun ist andererseits $U_{CE\text{sat}} = U_0 - R_C \cdot I_C$ bzw. $I_C = \frac{U_0 - U_{CE\text{sat}}}{R_C}$
Eingesetzt in vorige Gleichung ergibt dies:

$$R_V \leq B \cdot R_C$$

Die Verhältnisse werden im Übersteuerungsfall ($R_V < B \cdot R_C$) etwas schwieriger; wir verzichten auf die Berechnung.

1.7.2.3 Konkrete Realisierungen

1.7.2.3.1 Inverter in Widerstands-Transistor-Logik[1]

Der Transistor wird so betrieben, daß $U_{BE} \approx 0,7V$, $U_{CE\text{Rest}} = 0,1V$; R_L hat einen Wert von $10k\Omega$.
Falls $U_{\text{ein}} = 12V$, dann $U_A = 0,1V$ (Low),
Falls $U_{\text{ein}} = 0V$, dann $U_A = 12V$ $(High)$.

Problem: Werden an den Ausgang die Eingänge nachfolgender Stufen geschaltet, dann verändert sich die Spannung am Punkt Q.
1. Fall: Q führt hohes Potential $High$ (Transistor sperrt). Dann wirken die nachfolgenden Stufen wie ein Ableitwiderstand R_A gegen Massenpotential. Es ergibt sich folgendes Ersatzschaltbild:

U_A ist am Punkt Q $U_A = U_0 \cdot \frac{R}{R_L + R_A}$
Sei z. B. $R_A = 1k\Omega$, dann ist mit $U_0 = 12V$ und $R_L = 330\Omega$ $\quad U_A = 12V \cdot \frac{1k\Omega}{1,33k\Omega} \approx 9V$.

Je mehr Folgestufen am Punkt Q angeschlossen werden, desto kleiner wird der diese Stufen repräsentierende Ableitwiderstand R_A. Es ist deshalb bei gegebener Belastung zu überprüfen, ob der für $High$-Potential

[1] Resistor-Transistor-Logic (RTL)

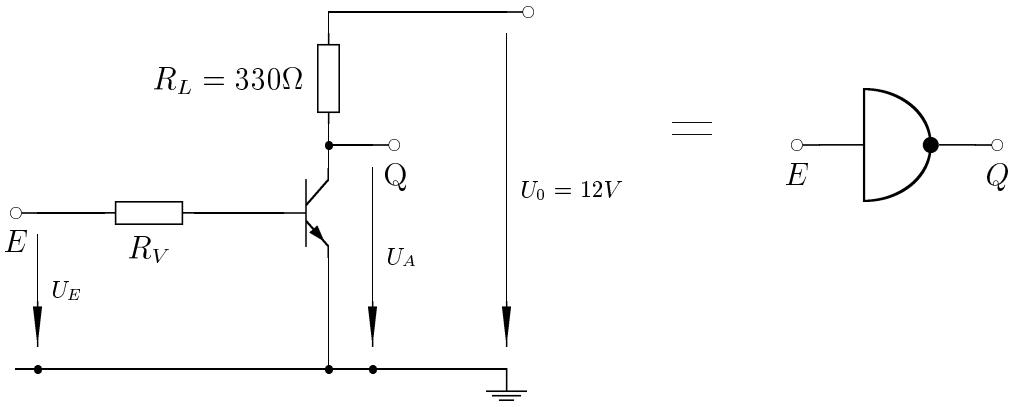


Abbildung 1.29: Inverter in Widerstands-Transistor-Logik

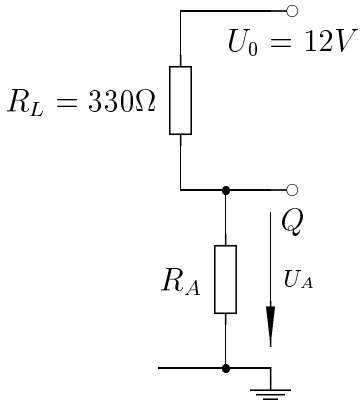


Abbildung 1.30: Ersatzschaltbild wenn Q = *High*

($Q = High$) vereinbarte Pegel noch erreicht wird.
2. Fall: Es kann der Fall eintreten, daß eine nachfolgende Stufe zu einer Belastung in der Form führt, daß sie an Q keinen Strom „herauszieht“, sondern einen Strom „hereindrückt“. Dann wird bei $Q = Low$ unter Umständen U_A zu groß.

Ersatzschaltbild:

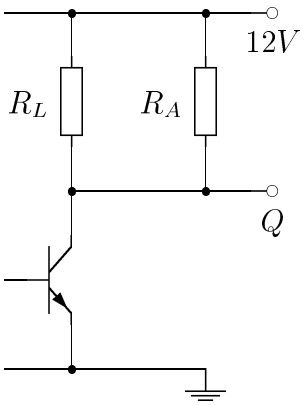


Abbildung 1.31: Ersatzschaltbild wenn Q = *Low*

Problem: I_A hebt das Potential an Q an.

1.7.2.3.2 ODER-Gatter

1.7.2.3.2.1 Ohne Verstärker (rein passive Schaltung)

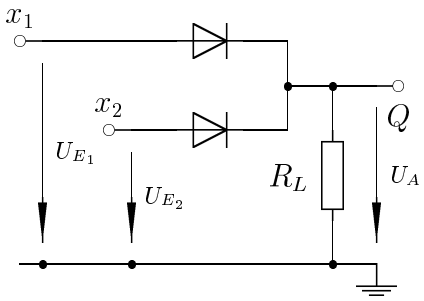


Abbildung 1.32: ODER-Gatter ohne Verstärker

Funktion:
Für $U_{E_1} = High$ und $U_{E_2} = Low$ leitet die Diode D_1, $U_A = High$, Diode D_2 sperrt.
Für $U_{E_1} = Low$ und $U_{E_2} = High$ leitet die Diode D_2, $U_A = High$, Diode D_1 sperrt.
Für $U_{E_1} = High$ und $U_{E_2} = High$ leiten die Dioden D_1 und D_2, $U_A = High$.
Der Widerstand R_L ist erforderlich, damit bei offenen Eingängen auf jeden Fall $U_A = Low$ ist.

1.7.2.3.2.2 Mit Verstärker

Um geringe Eingangsbelastung zu erzielen und höhere Ausgangsbelastungen zu ermöglichen, werden der passiven ODER-Stufe zwei verstärkende NICHT-Glieder nachgeschaltet: Dioden-Transistor-Logik (DTL-Technik).

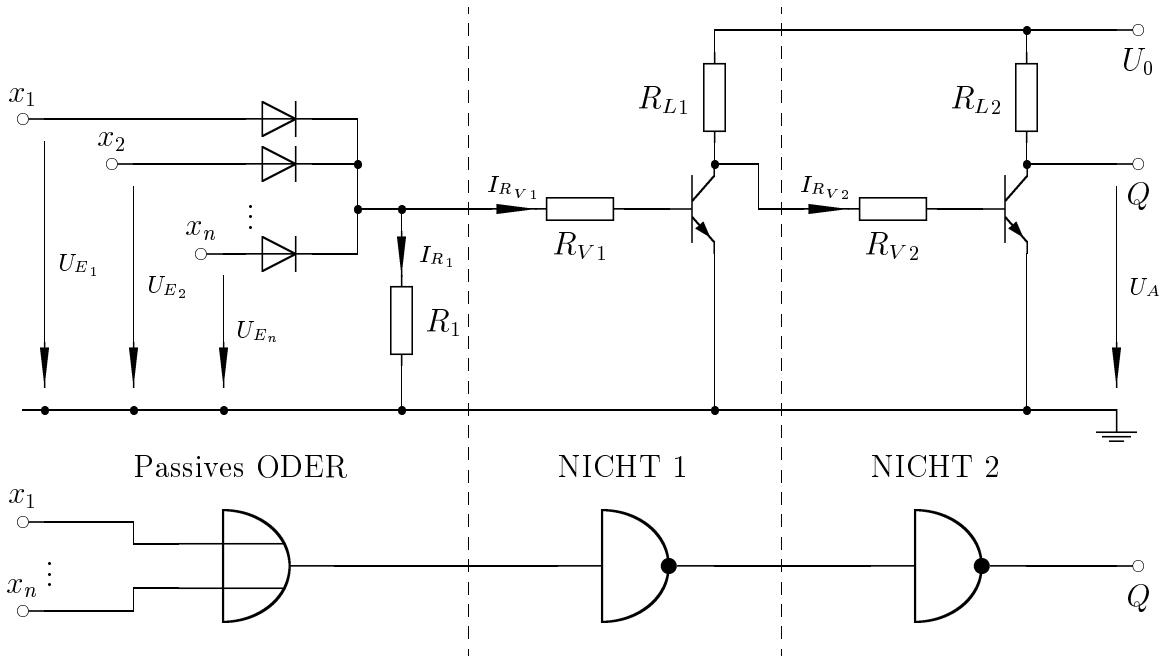


Abbildung 1.33: ODER-Gatter mit Verstärker

1.7.2.3.3 Passives UND-Gatter

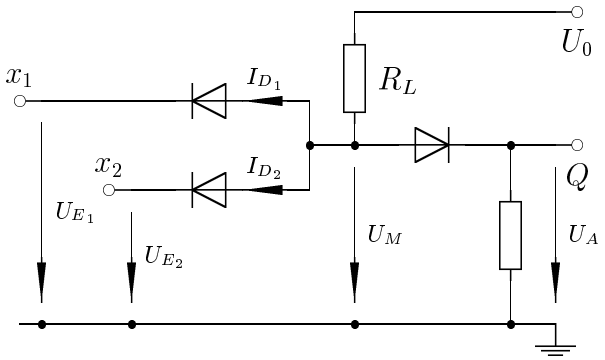


Abbildung 1.34: Passives UND-Gatter

Funktion

- $U_{E_1} = Low \wedge U_{E_2} = Low \Rightarrow D_1$ und D_2 leiten, über beide Dioden fließt I_{D1} nach Masse, $U_M \approx 0$.
- $(U_{E_1} = Low \wedge U_{E_2} = High) \vee (U_{E_1} = High \wedge U_{E_2} = Low) \Rightarrow U_M$ wird über eine Diode auf Low gezogen.
- Nur wenn $U_{E_1} = High \wedge U_{E_2} = High$ dann wird $U_M \approx U_0$.

DTL-Technik wurde für diskreten Aufbau entwickelt. Für integrierten Aufbau eignet sich TTL-Technik besser.

1.7.2.4 TTL-Technik: Transistor-Transistor-Logik

Entscheidender Entwicklungsschritt: Doppel-Emitter-Transistor.
NPN-Transistor kann im Ersatzschaltbild durch zwei Dioden nachgebildet werden:

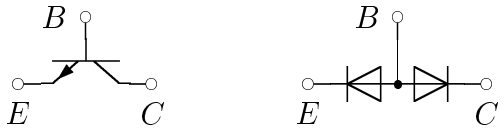


Abbildung 1.35: NPN-Transistor kann durch zwei Dioden nachgebildet werden

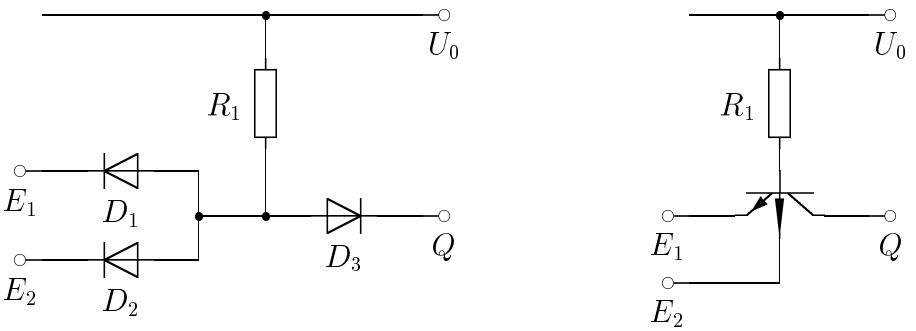


Abbildung 1.36: Passives UND-Gatter in DTL-Technik und „aktives“ UND-Gatter in TTL-Technik

Multi-Emitter-Transistor benötigt deutlich weniger Chipfläche als zwei Dioden.

1.7.2.4.1 Verwendung des Multi-Emitter-Transistors zum Aufbau eines UND-Gatters

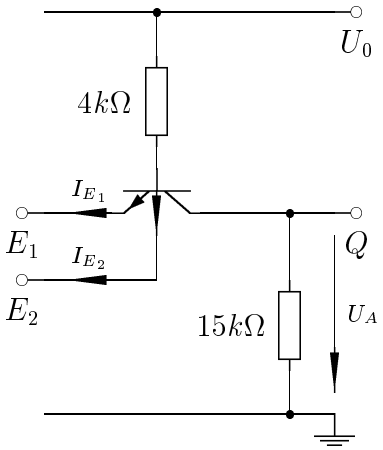


Abbildung 1.37: Verwendung des Multi-Emitter-Transistors zum Aufbau eines UND-Gatters

Funktion: Analog zu der UND-Schaltung in DTL-Technik.
1. Fall: $U_{E_1} = Low$ (und (zusätzlich) $U_{E_2} = Low$).

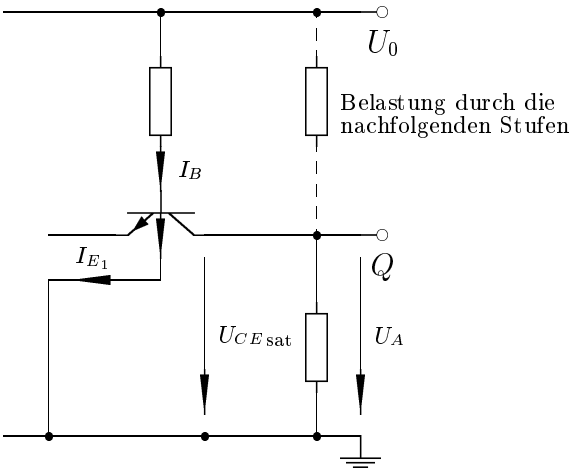


Abbildung 1.38: Ersatzschaltbild wenn $Q \approx U_{CE\mathrm{sat}} = Low$

Der Transistor arbeitet im Normalbetrieb, Basisstrom I_B fließt über I_E ab, Ausgang $Q \approx U_{CE\mathrm{sat}} = Low$

2. Fall:

Der Transistor leitet ebenfalls, aber er arbeitet im Inversbetrieb, d. h. E liegt auf höherem Potential als C, der Basisstrom fließt über R ab, $\Rightarrow Q = High$.
Der Multi-Emitter-Transistor wird also ständig vom Basisstrom durchflossen. Dies führt zu kurzen Umschaltzeiten, weil beim Umschalten der beiden Betriebszustände die Basisladung nicht ausgeräumt werden muß.

1.7.2.4.2 Erweiterung des UND-Gatters um einen Verstärker

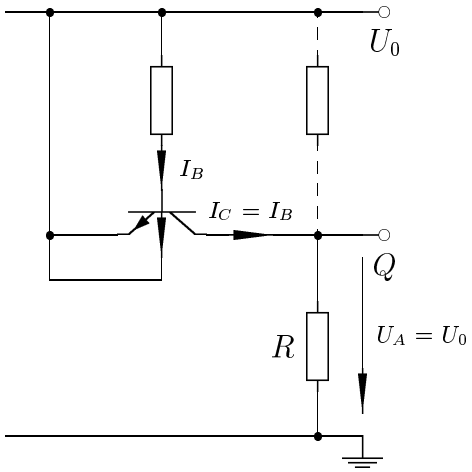


Abbildung 1.39: Ersatzschaltbild wenn $Q = High$

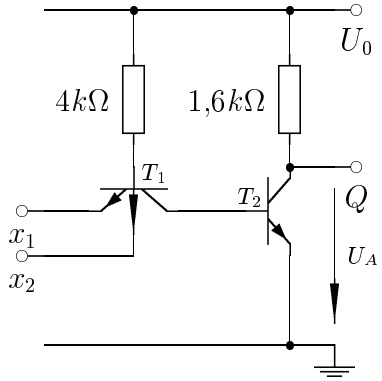


Abbildung 1.40: Um einen Verstärker erweitertes UND-Gatter

T_2 führt zu Verstärkung und zu Potentialumkehr. ⇒ Grundschaltung der TTL-Technik: NAND-Gatter.
Problem: Am Ausgang liegende Last-Kapazitäten müssen beim Pegelwechsel schnell umgeladen werden. Im Falle eines Wechsels von Q : $Low \rightarrow High$ muß dies über den Widerstand erfolgen. Dies ist sehr langsam.
Lösung: Aufbau einer sogenannten Gegentakt-Endstufe.

Funktion:
1. Fall: Beide oder ein Eingang auf *Low*. Dann hat Basis von T_2 das Potential $U_{CE\mathrm{Rest}\,T_1}$, welches sich auswirkt, um T_2 durchzusteuern.

Damit liegt die Basis von T_3 über R_2 auf High, T_3 leitet, $Q = High$.

2. Fall: Beide Eingänge auf $High$. Nach Rechnung: $U_{BET_2} \approx 1,4V$. T_2 und T_4 steuern durch, Q wird $Q = U_{CE\mathrm{Rest}\,T_4} \leq Low$.

Da T_2 in der Sättigung und damit $U_{CET_2} \approx 0,2V$ liegt die Basis von T_3 auf einen um $0,2V$ höheren Potential, als die Basis von T_4. Um zu vermeiden, daß T_3 ebenfalls durchsteuert, wird das Emitterpotential von T_3 durch die Diffusionsspannung von D in etwa $0,7V$ angehoben, T_3 sperrt, weil $U_{BET_3} < 0V$.

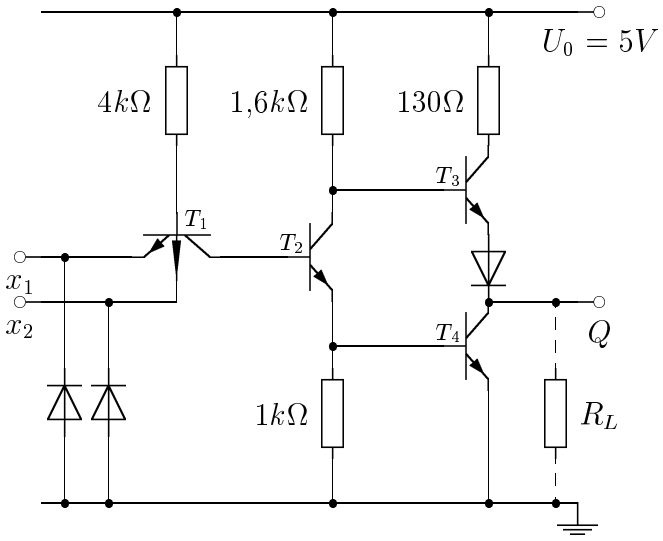


Abbildung 1.41: Gegentakt-Endstufe

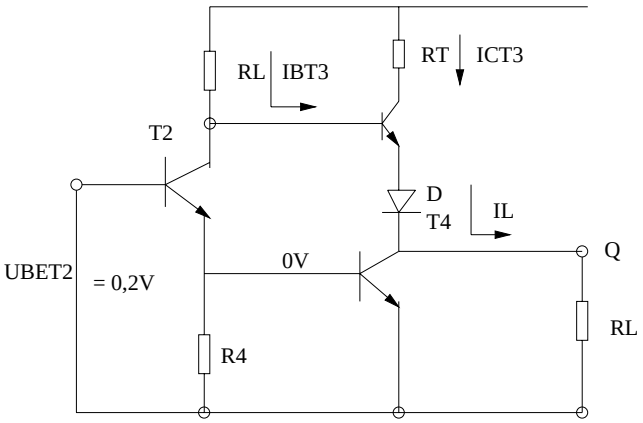


Abbildung 1.42: Ersatzschaltbild wenn $Q = High$

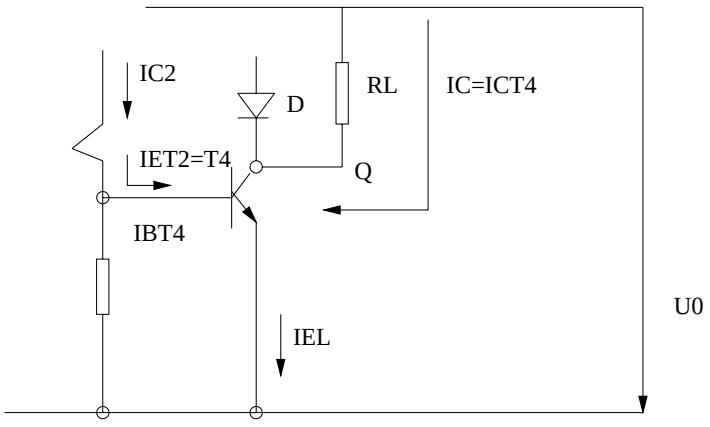


Abbildung 1.43: Ersatzschaltbild wenn $Q = Low$

Kapitel 2

Zahldarstellungen und Informationsrepräsentation

2.1 Zahldarstellungen und Rechenwerke

2.1.1 Zahldarstellungen

Neben der Ausführung von logischen Verknüpfungen zwischen Variablen (und anschließend bedingtem Sprung) ist das Rechnen mit Zahlen die andere wichtige Klasse von Operationen die Universalrechner ausführen können müssen.
Je nach Anwendung unterschiedliche Anforderung an Rechengeschwindigkeit, Aufwand und Geschwindigkeit:

Beispiel:

	Genauigkeit	Geschwindigkeit	Aufwand
DSP	niedrig 4 ... 16 bit	hoch bis sehr hoch (Co, HPN)	niedrigst (Consumer Anw.)
Arithmetik auf PC's	mittel 16 ... 64bit	niedrig bis mittel	mittel je Preisklasse
wiss. techn. Rechner	hoch ... 128bit	hoch bis sehr hoch(> 1 GFLOPs)	hoch bis sehr hoch
kaufmänn. Rechner	mittel bis hoch 32 ... 64 bit	niedrig bis mittel	mittel (Main Frame)

Diese drei Kriterien entscheiden über die Wahl der Zahldarstellung und über den Grad der (parallelen) Ausführungen von Operationen direkt in Hardware.

Grundsätzliche Unterscheidung in 2 Klassen:

Ganzzahl-Darstellung (Fließkomma); Vorzeichenbehaftet (INTEGER), Vorzeichenlos (CARDINAL):

- Vorteil: Operation einfach durchzuführen, billig.
- Nachteil: Begrenzter Wertebereich („geringe Dynamik“) setzt genauere Problemanalyse voraus. Der auftretende Diskretierungsfehler Δ (Abstand zwischen zwei Zahlen) ist absolut konstant.

Fließkomma Zahlen: (REAL, Annäherung an reelle Zahlen). Einführung einer zusätzlichen Zahl ex, die die Stellung des Kommas angibt (halblogarithmische Darstellung).
Beispiel:

$$x = \underbrace{375}_{Matisse} \cdot \underbrace{10}_{Basis}^{\underbrace{ex}_{Exponent}} = \begin{cases} 0,375 & für \quad ex = -3, \Delta = 0,001 \\ 375 & für \quad ex = 0, \Delta = 1, \\ 375.000 & für \quad ex = 3, \Delta = 1000. \end{cases}$$

Vorteil: Mit gleicher Anzahl von Bits läßt sich ein größerer Wertebereich als bei ganzzahliger Darstellung überstreichen, relativer Diskretierungsfehler Δ/X bleibt etwa konstant.
Nachteil: Höherer Aufwand, durch getrennte Behandlung von Matisse und Exponent, immer geringere Genauigkeit, als bei Ganzzahl gleicher Stellenzahl.

2.1.1.1 Ganzzahlige-Darstellung im Detail

Positive n-stellige ganze Zahl mit dem Wert x zur Basis b wird durch Folge von n Ziffern $X_i \epsilon [0, b-1]$ repräsentieren. Dabei ist $x = \sum_{i=1}^{n} X_i \cdot b^{i-1}$, Darstellung als Ziffernfolge $(X_n, X_{n-1}, \ldots, X_1)_b$. Gilt ebenso für gebrochene binäre Zahlen:
$(X_n, X_{n-1}, \ldots, X_1, X_0, X_{-1}, \ldots, X_{k-1}) = \sum_{i=-k}^{n} X_i b^{i-1}$
Die Folge der n Ziffern zur Darstellung der Zahl mit dem Wert x werden mit $X[n]$ bezeichnet.

2.1.1.2 Darstellung nach Vorzeichen und Betrag

Vereinbarung eines Vorzeichensbits V für die Zahl mit dem Wert x, so daß Vorzeichenzahl (V, $X[n]$) wie folgt definiert:

$$[X_n] = |x|;\ v = \begin{cases} 0 & für \quad x \geq 0 \\ 1 & für \quad x <_{(-)} 0\text{„}negativeNull\text{“} \end{cases}$$

Dann ist $x = (\overline{V} - V) \cdot x[n]$ mit $\overline{V}$ als Komplement (Rest zu 1) von V.
Diese Darstellung ist gut geeignet für Multiplikation und Division, nicht aber für Addition, Subtraktion. Bei letzteren ist Zweikomplementdarstellung günstiger, weil dabei keine getrennte Behandlung von Vorzeichen nötig ist.

2.1.1.3 Zweikomplementdarstellung

Zur häufigsten gewählten Darstellung für positive und negative Dualzahlne. Duale Werte der Zweikomplementzahl stimmt mit x überein, wenn $x \geq 0$. Ist x negativ, dann wird $|x|$ von 2^n abgezogen, damit wieder positiver Wert entsteht. Es ist :

$$x[n] = \begin{cases} x & für \quad x \geq 0 \\ 2^n - |x| = 2^n + x & für \quad x < 0. \end{cases}$$

Also Repräsentation der negativen Zahlen in der oberen Hälfte ($x_n = 1$) des Wertebereichs einer n-stelligen Zahl.
X_n sieht man, ob Zahl positiv oder negatives X_n wird in Rechnung direkt einbezogene Rückabbildung:

$$x = x[n] - x_n 2^n = \begin{cases} x[n] & für \quad x_n = 0 \\ x[n] - 2^n & für \quad x_n = 1 \end{cases}$$

Veranschaulichung an Zahlenring für 3-bit-Zahl:

Bemerkung: Wird eine n-stellige Zweikomplementzahl um p-Stellen erweitert, dann muß das Sign-Bit x_n p-mal links angehängt werden. (Sign-Extension, z. B. wird beim Übergang von $n = 3$ auf $n = 5$ aus der Repräsentation 101 (entspricht -3) die Repräsentation 11101). Frage nach dem Vorzeichenwechsel: Welche Operationen sind nötig, um $x[n]$ so umzufassen, damit sie statt $+x$ den Wert $-x$ darstellt. Den

negativen Wert repräsentieren $X'[n]$ mit $X'[n] = 2^n - |x|$ also $X'[n] = 2^n - X[n]$.
Es gilt: $X'[n] = \overline{X[n]} + 1$ wo $\overline{X[n]}$ alle Stellen invertiert.
Beweis: Offensichtlich ist $X[n] + \overline{X[n]} = 11\ldots1 = 2^n - 1$.
Mit $X'[n] = 2^n - [(2^n - 1) - \overline{x}[n]] = \overline{X[n]} + 1$.
Also Vorzeichenwechsel durch Negation aller Bits und Addition von 1. Serielles Vorgehen („Bit für Bit"): Von rechts her bleiben alle Nullen unverändert und die erste eins, danach werden alle Stellen bitweise invertiert.
Achtung: Die negativste Zahl -2^{n-1} darf nicht komplementiert werden, weil sie um eins (betragsmäßig) größer ist, als die größte positive.

2.1.1.4 Einerkomplement

Wie Zweierkomplement, aber Komplement wird nicht gegen 2^n, sondern gegen $2^n - 1$ gebildet:

$$X[n] = \begin{cases} x & für \quad x \geq 0 \\ 2^n - 1 + x & für \quad x < 0 \end{cases}$$

und Rückabbildung

$$x = \begin{cases} X[n] & für \quad x_n = 0 \\ X[n] - 2^n + 1 & für \quad x_n = 1 \end{cases}$$

Hier ist $X'[n] = \overline{X[n]}$, der Vorzeichenwechsel ist beim Einerkomplement also besonders einfach, dafür ist aber bei Rückabbildung für $x_n = 1$ eine Korrektur um 1 erforderlich.

2.1.1.5 Binärkodierte Dezimalzahlen (Dualzahlen)

Problem: Obwohl Dualzahlen rechnerisch einfach zu handhaben, scheiden sie aus, wenn sie keine exakte Repräsentation ermöglichen. underlineAußerdem: Menschen wollen Eingabe/Ausgabe in Dezimalform vornehmen.
Mögliche Lösung: Konversion zwischen Zahlensystem. Dabei jedoch Genauigkeitsverlust. Die rationale Zahl 0,1 ist z. B. nicht mit endlichen Stellenzahl in Dualsystem darstellbar.
Deshalb: Besonders im kaufmännischen Bereich wird auch rechnerintern mit Dualzahlen operiert. Jede Dezimalstelle wird in 4 Bits kodiert, die Vierergruppen heißen „Tetranden" die Bitkombinationen, die die unerlaubten Ziffern „10 ... 15"kodieren, heißen „Pseudotetranden".
Beispiel: 1015_{10} entspricht $\underbrace{0001}_{1}\ \underbrace{0000}_{0}\ \underbrace{0001}_{1}\ \underbrace{0101}_{5}$
Darstellung von negativen BCD-Zahlen in neuner Komplement.

2.1.1.6 Sedezimalzahlen (Hexadezimalzahlen)

Rechnerinterne Dualzahlen sind für Menschen unhandlich (nicht einprägsam). Deshalb häufig als Zahl zur Basis 16. Dualzahl als Hexadezimalzahl darstellbar durch Gruppierung von jeweils vier Bits, wobei die im Falle der BCD-Darstellung verbotenen Pseudotetraden hier durch die Buchstaben $A \ldots F$ repräsentiert werden.
Zur Kennzeichnung, daß es sich um eine Hexadezimalzahl handelt, wird ein „H" angehängt.
Beispiel $\underbrace{1011}_{B}\ \underbrace{0110}_{6}\ \underbrace{0011}_{3}\ \underbrace{1100}_{C} = B63CH$
Stellenwerte sind Potenzen zur Basis 16, es ergibt sich in Dezimaldarstellung:
$4096 \cdot 11 + 256 \cdot 6 + 16 \cdot 3 + 1 \cdot 12 = 46652$
Anwendung hauptsächlich zur Ausgabe von Zahlenwert auf Registerebene des Prozessors.

2.1.1.7 Fließkommazahlen

Fließkommanzahl x besteht aus Martisse mx zur Basis b und Exponent ex zur Basis:
$x = mx \cdot b^{ex}$
Offensichtlich ist diese Darstellung nicht eindeutig:
Beispiel:
$x = 40_{10} = 101_2 \cdot 2^3 = 1010_2 \cdot 2^2 = 10,12 \cdot 2^4$
Deshalb Definition einer normalisierten Darstellung:
$x = v \cdot \underbrace{mx_{norm} \cdot b^k}_{mx'_{norm}} \cdot b^{ex}$
mx_{norm} : positive, normalisierte Martisse.
V: Vorzeichen $\{+1, -1\}$ oder $\{0, 1\}$
K: ganzzahlige Konstante.
Bei Darstellung der Martisse durch r-stellige Zahl zur Basis b eignet sich am besten die folgende Normalisierungsbedingung:
$\frac{1}{b} \leq mx_{norm} \leq 1 - \frac{1}{b^r} < 1$
Das bedeutet, daß das Komma vor der ersten Stelle (höchstwertige Ziffer) der Martisse steht.
Beispiel:

$b = 10$, $r = 3$, $mx_{norm} \epsilon [0.100, \ldots, 0.999]$
$b = 2$, $r = 3$, $mx_{norm} \epsilon [0.500, \ldots, 0875]$

Damit wird der Wertebereich für darstellbare Zahlen:
$\frac{1}{b} \cdot b^{ex_{min}} \leq x \leq (1 - \frac{1}{b^r}) \cdot b^{ex_{max}}$

(**Achtung:** Null nicht enthalten !!)
Der Abstand zwischen zwei aufeinanderfolgende Zahlen wird $\Delta x = \frac{\Delta}{b^r} \cdot b^{ex}$
Für die Darstellung des Exponenten eignet sich (da Ganzzahl) die Zweikomplementdarstellung. Besser jedoch: Darstellung als Charakteristik. $EX[n] = ex + 2^{n-1}$; n Stellenzahl zur Darstellung von ex.
Entspricht Zweikomplementdarstellung mit negierten Vorzeichen. Vorteil: EX wächst monoton mit der Größe von x, deshalb ist Betragsvergleich zweier Fließkommazahlen besonders einfach (in einem Schritt.)
Dafür notwendige und gebräuchliche Darstellung:

(hier für 32−bit Wort).
Größer-/Kleinerrelation für Zahlen gilt ebenso für ihre Darstellung.
Wichtigste Norm: IEEE-Standard $754 - 1985$ definiert 4 verschiedene Formate; das gebräuchlichste ist das 32 bit single procision format.

$EX\{8\} = 2^{8-1} + ex \underbrace{-1}_{wegen Kontrollcodes} = ex + 127$ für $ex = -126 \ldots 127$

Die Martisse wird auf Werte zwischen 1 und < 2 normiert (Multiplikation der Normalisierungsbedingung mit $2 = b^k; k = 1$). Da die Stelle vor dem Komma immer Eins, wird sie nicht dargestellt; es verbleiben 23 Stellen für den gebrochenen Anteil (Fractal FX).
Damit: $x = (-1)^r \cdot 2^{EX-127} - (1, < FX >)$.
Erlaubter Exponentenbereich $-126 \ldots 127 \Leftrightarrow EX = 1 \ldots 254$.
$EX = 0$ wird zur Darstellung der Null verwendet.
$EX = 255$ zur Darstellung con ∞ bzw. weiterer in dem Martissenbits kodierter Steuercodes.
Beispiel für Darstellung: Zu repräsentieren sei die Zahl $x = 47,953125$. Umwandlungsprozedur:

Also entspricht die exakte darstellbare Zahl 47,953125 der Dualzahl $101111,111101 = 1,01111111101 \cdot 2^5$
Damit:

2.1.2 Schaltnetze zur Addition

2.1.2.1 Addition von zwei n-bit-Zahlen

Direkter Ansatz: Addierer für jede Stelle und Übertragsweiterleitung von Stelle zu Stelle.

	X_n	$X_{n-1} \ldots$	X_1
	Y_n	$Y_{n-1} \ldots$	Y_1
S_{n+1}	S_n	$S_{n-1} \ldots$	S_1

Mit Halbaddierer
und Volladdierer

$* = (x_i \wedge y_i) \vee (x_i \wedge c_i) \vee (y_i \wedge c_i)$
$= \underbrace{(x_i \wedge y_i)}_{y_i} \vee [\underbrace{(x_i \vee y_i)}_{(P_i \wedge c_i)} \wedge c_i]$
y_i ist Carry Generate und $(P_i \wedge c_i)$ ist Carry Propagate.
ergibt sich sofort Schaltnetz:

Problem: Vor Gültigkeit der Ergebnisse ist Durchlaufen des Übertrags durch alle Stufen erforderlich. Jeweils zwei Gatterlaufzeiten pro VA.
Lösung: Reduktion der Zeit, bis Übertrag gültig durch Übertragsvorausberechnung (Carry-Look-Ahead). Dazu Einsetzen der Gleichung für c_{i+1} in Gleichung für c_{i+2}: Damit läßt sich c_{i+2} berechnen, ohne daß c_{i+1} berechnet.
Erläuterung am Beispiel der Kaskadierung (Hintereinanderschaltung) von 4 VA
„Geradeaus Fall“:

Ergebnis gültig nach $8T$ mit T: Gatterlaufzeit.

Nun für Carry-Look-Ahead:

$C_2 = G_1 \vee (P_1 \wedge C_1)$
$C_3 = Y_2 \vee (P_2 \wedge C_2) = G_2 \vee (P_2 \wedge [Y_1 \vee\ (P_1 \wedge C_1)]) = Y_2 \vee (P_2 \wedge G_1) \vee\ (P_2 \wedge P_1 \wedge C_1)$
$C_4 = Y_3 \vee (P_3 \wedge C_3) = Y_3 \vee (P_3 \wedge Y_2) \vee\ (P_3 \wedge P_2 \wedge Y_1) \vee (P_3 \wedge P_2 \wedge P_1 \wedge C_1)$
„P bzw. G-Verknüpfung“: 1 Takt
„Und-Verknüpfung : 1 Takt
„Oder-Verknüpfung: 1 Takt und 2 Takte im letzten Addierer $= 5T \Leftarrow 40\%$ Ersparnis !!

2.1.2.2 Addition von Zweikomplementzahlen

Vorgehen wie bei Addition, allerdings müssen je nach Zahlenwert der beiden Summanden X und Y verschiedene Fälle unterschieden werden. Dies deshalb weil Ergebnis unter Umständen nicht mehr in vorgeg. Stellenzahl darstellbar.

1. Die beiden Operanden haben unterschiedl. Vorzeichen (zu erkennen am MSB). Addition analog zum Ganzzahl-Fall; Carry-Bit hat nur für Kaskadierung eine Bedeutung.

2. Operanden haben gleiches Vorzeichen
   $(X_n = 0 \wedge Y_n = 0) \vee (X_n = 1 \wedge Y_n = 1)$. Dann besteht Gefahr desÜberlaufs, d. h. Ergebnis ist nicht mehr mit Stellenzahl der Operanden darstellbar. Überlauf bei zwei positiven Zahlen, wenn $S_n = 1$ (positive und positive Zahl ergibt negative Zahl).
   Überlauf bei zwei negativen Zahlen, wenn $S_n = 0$ (C_{n+1} ist dabei 1).

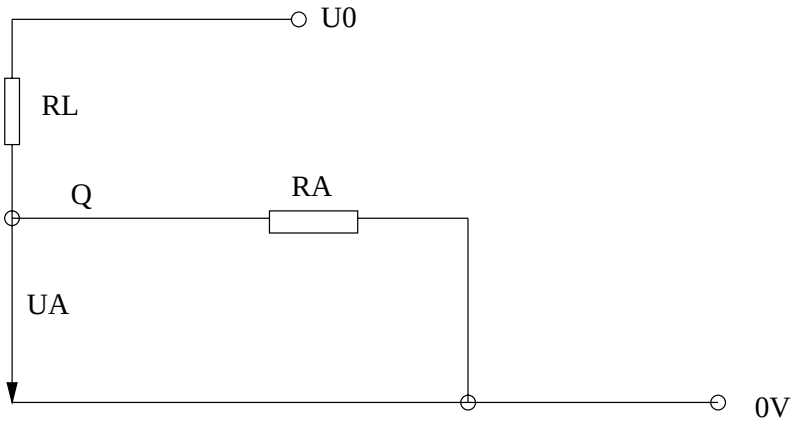
Beispiele für Überlauffälle: 4-bit Zweikomplementzahlen.

+7	0111
+6+	0110
0	1101

$0 \leftarrow C_5$
1101 entspricht -3 statt 13.

-7	1001
-6	1010
1	0011

$1 \leftarrow C_5$ +3 statt -13 Überlauf-Bedingung wird angezeigt durch Overflow-Bit OV; dieses Bit zeigt an, daß die Summe nicht mehr mit Stellenzahl der Operanden darstellbar.
$OV = (X_n \wedge Y_n \wedge \overline{S_n}) \vee (\overline{X_n} \wedge \overline{Y_n} \wedge S_n) = (X_n \equiv Y_n) \wedge (C_{n+1} \not\equiv S_n)$
Wenn Zahlen mit $n+1$-Stellen weiterverbreitet werden kann, zeigt OV an, daß C_{n+1}-Bit als $n+1$-te Stelle verwendet werden kann. Somit ist OV-Bit Fehlerindikator.



2.1.3 Verfahren zur Division von Dualzahlen

Betrachtung des ganzzahligen Falles.
$\frac{Dividend[2n]}{Divisor}$ = Quotient$[n]$ + Rest$[n]$.
Ermittlung der Quotientenziffer durch schrittweise Subtraktion des gegen den Dividenden verschobenen Divisors und Vergleich (zur Feststellung, ob verschobener Divisor in Dividend enthalten). Vor der Division muß geprüft werden, ob das Ergebnis mit n-Stellen darstellbar ist. Das ist der Fall, wenn Differenz zwischen Dividend und ganz nach links geschoben Divisor neg. ist.
Beispiel für Überlauf im Dezimalsystem:
Ist $13 \div 4$ mit einer Dezimalstelle darstellbar ?
Überlauftest:

	13	
	-4	
P	-3	< 0:darstellbar

2.2 Informationsrepräsentation: Kodierungstheorie

Es gibt unterschiedliche Repräsentationsformen von Informationen, z. B. für die

Tatsache, daß die Ampel rot ist:

- Ton bestimmter Höhe
- Licht bestimmter Farbe
- Zeichenfolge: „Die Ampel ist rot“.

Üblicher Fall: Repräsentation einer endlichen Grundmenge von Information durch Zeichenfolgen aus einem endlichen Zeichensatz.

Problem: Einfache und ökonomische Repräsentation von Informationen.
Lösung: Kodierung, d. h. Übergang von einem Repräsentationssystem in ein anderes.

„Die Ampel ist rot“ $\rightarrow AR$.
„Die Ampel ist grün“$\rightarrow AG$.

Zwei Ziele bei der Wahl des Codes:
Einerseits möglichst kurze Codewörter (ökonomische Übertragung), andererseits gewisses Maß an Redundanz, um Störungen korrigieren zu können.

2.2.1 Kodierung von Informationen

2.2.1.1 Kodierung für interne Repräsentation

- Kodierer wandelt Zeichenfolge in andere Zeichenfolge um, die adäquat für die interne Repräsentation ist, d. h. für die Verarbeitung durch den Prozessor.

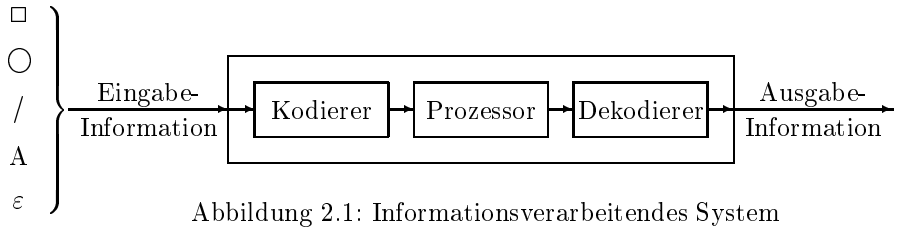


Abbildung 2.1: Informationsverarbeitendes System

- Prozessor arbeitet auf Zahlen- bzw. Zeichenfolgen dieser Repräsentation.
- Dekodierer wandelt das Resultat in die gewünschte „ausgebbare“ Form um.

2.2.1.2 Kodierung für Informationsübertragung und Störung

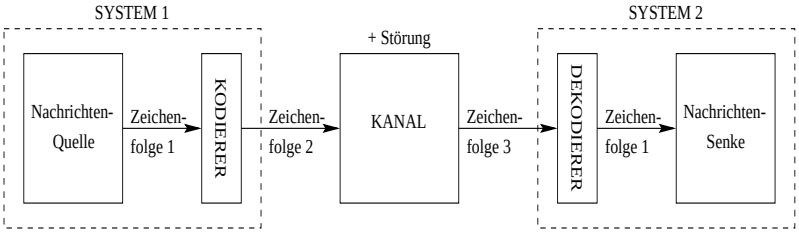


Abbildung 2.2: Kodierung für Informationsübertragung: Umwandlung einer Zeichenfolge 1 in eine Zeichenfolge 2, die adäquat über den Kanal übertragen werden kann.

- **Parallelübertragung**: Ein oder mehrere Zeichen werden parallel übertragen (z. B. 8 Bit gleichzeitig).
- **Serienübertragung**: Ein Zeichen wird in mehrere Untereinheiten zerlegt, die nacheinander übertragen werden (z. B. 7 Bit/Zeichen seriell).

2.2.1.3 Anforderungen

- Ökonomie der Darstellung für die Verarbeitung und Speicherung (wichtig für 2.2.1.1).
- Fehlersicherheit, d. h. Eindeutigkeit der Dekodiertheit auch nach Überlagerung von Störungen.

2.2.1.4 Einige Definitionen

- **Zeichenmenge** A: Zeichenvorrat, bei linearer Ordnung auch **Alphabet** genannt.
- **Elementarer Zeichenvorrat**: $\mathbb{B} = \{0, L\}$, zwei Binärzeichen oder binary digits (Bit).
- Menge A^* der **Zeichenfolgen** über Zeichenvorrat A enthält die **Wörter** über A.
  Z. B. ist die Menge $\mathbb{B}^* = \{0, L\}^*$ die Menge der Binärwörter (z. B. ist „L00L0L“ ein 6-bit Binärwort, die Menge $\mathbb{B}^n$ ist die Menge aller n-bit Binärwörter).
- Ein **Code** ist die Abbildungsvorschrift
  $c : A \to \mathbb{B}$ für einzelne Zeichen oder
  $c : A^* \to \mathbb{B}^*$ für Abbildung einer Zeichenfolge in eine Zeichenfolge.

Im Zusammenhang mit der Rechentechnik werden zumeist Binärkodierungen für Alphabete betrachtet, also Kodierungen der Form $c : A \to \mathbb{B}^*$, wobei A ein bestimmtes Alphabet sei.

Beispiele für Alphabete: Alle Großbuchstaben, alle Ziffern, alle chinesischen Zeichen.
Beispiele für Codes: ASCII[1], EBCDIC[2], ISO-Latin 1, Unicode.

2.2.2 Codes fester Länge

Rechnerintern steht eine feste Anzahl von n Bits zur Darstellung eines Codewortes CW zur Verfügung. Daher zumeist Verwendung von Codes fester Länge $c : A \to \mathbb{B}^n$.

Normalerweise werden für die Kodierung „direkte“ Codes verwendet, bei denen eine Gewichtsfunktion g_i für die einzelnen Binärstellen zum Einsatz kommt.

Dabei sind die Gewichte von Binärzeichen $w(L) = 1$ und $w(0) = 0$.

Wenn für jede Stelle i (mit $0 \leq i \leq n-1$) ein Gewicht g_i vorgegeben ist, dann wird dem Binärwort $d = \langle d_{n-1} \dots d_0 \rangle$ mit $d_i \in \{L, 0\}$ eine Zahl z zugeordnet durch $z = \sum_{i=0}^{n-1} g_i \; w(d_i)$

2.2.2.1 Beispiel: Direkter Code für die Dezimalziffern mit den Gewichten $g_i = 2^i$

4-bit-Binärwort:

2^3	2^2	2^1	2^0
8	4	2	1
$i = 3$	$i = 2$	$i = 1$	$i = 0$

Damit Zuordnung zu den ersten zehn Ziffern des Dezimalsystems:
$c_D : \{0 \dots 9\} \to \mathbb{B}^4$

a	$c_D(a)$
0	0000
1	000L
2	00L0
⋮	⋮
9	L00L

Codewörter unterscheiden sich in der Anzahl von Nullen und Einsen. Charakterisierung des „Unterschieds“ zwischen Wörtern $a = \langle a_1 \dots a_n \rangle$ und $b = \langle b_1 \dots b_n \rangle$ durch den sogenannten **Hammingabstand**. Je mehr sich zwei Wörter unterscheiden, desto größer ist der Hammingabstand h. Je größer h, desto bessere Möglichkeiten hat man, die übertragene Information im Fehlerfall zu erkennen.

2.2.2.2 Definition: Hammingabstand

$h : \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{N}$ mit $h(\langle a_1 \dots a_n \rangle, \langle b_1 \dots b_n \rangle) = \sum_{i=1}^{n} d_i$, mit $d_i = \begin{cases} 1 & \text{falls } a_i \neq b_i \\ 0 & \text{sonst} \end{cases}$
oder einfach die Anzahl von Bitstellen, in denen sich die Wörter voneinander unterscheiden.

Der Hammingabstand einer Kodierungsabbildung ist der minimale Abstand zwischen zwei Codewörtern.

[1] American Standard Code for Information Interchange
[2] Extended Binary Coded Decimal Interchange Code

2.2.2.3 Beispiel: Einschrittiger Code (Gray-Code) für die Ziffern 0 ... 9

a	$c_{Gray}(a)$	a	$c_{Gray}(a)$
0	0000	5	$0LLL$
1	$000L$	6	$0L0L$
2	$00LL$	7	$0L00$
3	$00L0$	8	$LL00$
4	$0LL0$	9	$L000$

2.2.2.4 Kodierung zur Fehlererkennung und -korrektur

Problem: Bei der Übertragung von Codewörtern auf einem störungsanfälligen Kanal kann es zu Fehldekodierungen beim Empfänger kommen.

2.2.2.4.1 Direkter Code, 2-bit-Worte für die Ziffern 0 ... 3

a	$c_D(a)$
0	00
1	$0L$
2	$L0$
3	LL

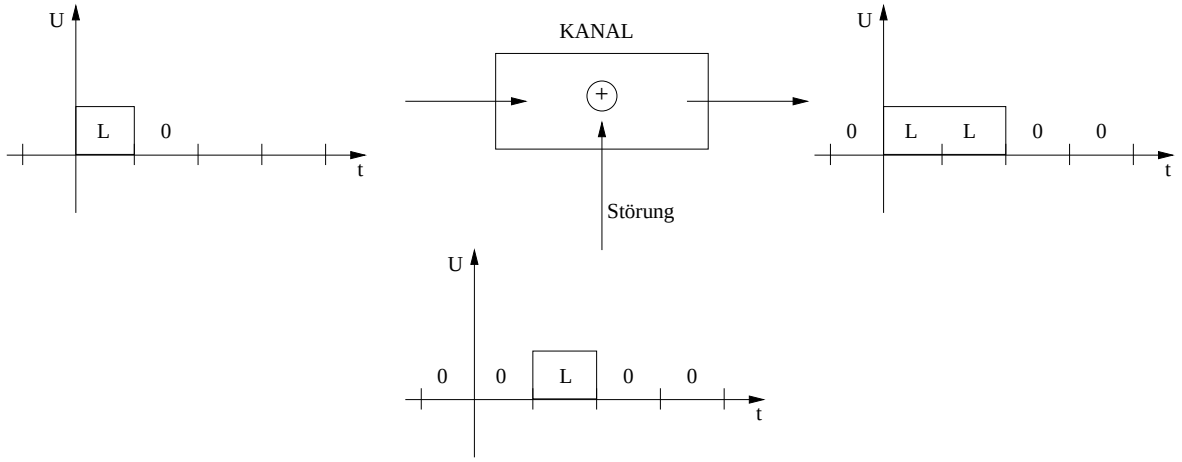


Abbildung 2.3: Es soll die Folge L0 gesendet werden. Als Störung werde die Folge 0L überlagert. Statt der Ziffer „2“ wird die Ziffer „3“ dekodiert.

Da jede Bitstelle der Codewörter mit „Nutzinformationen“ belegt ist ($h = 1$), bedeutet schon ein einzelner Bitfehler eine Verfälschung.
Abhilfe durch den Einbau von **Redundanz** (z. B. 3-bit-Codewort (CW)).

a	$C_P(a)$	Paritätsbit[3]
0	00	0
1	$0L$	L
2	$L0$	L
3	LL	0

2.2.2.4.2 Codewürfel

[3] 0 bei gerader Anzahl von L's, L bei ungerader

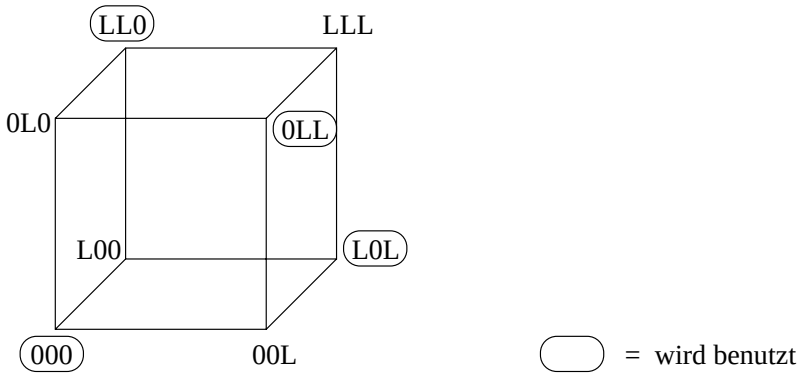


Abbildung 2.4: Codewürfel: Darstellung im (hier 3-dimensionalen) Coderaum

Zwischen zwei verwendeten Codewörtern liegen (mindestens) 2 Kanten des Würfels ($h = 2$). Im Falle eines 1-Bit Fehlers tritt ein illegales Codewort auf. Damit ist die Erkennung von 1-bit-Fehlern möglich, allerdings keine Erkennung von 2-Bit-Fehlern. Dies ist erst realisierbar, wenn die Redundanz (Hammingabstand) weiter vergrößert wird.

Im folgenden Behandlung von linearen Block-Codes, linearen systematischen Codes und zyklischen Codes.

2.2.2.4.3 Vorbemerkung

Da die Bitstellen im folgenden nicht nur für „Ja/Nein"- Entscheidungen stehen, sondern mathematische Operationen mit ihnen ausgeführt werden, benutzen wir zu ihrer Darstellung die Ziffern 0 und 1 (entspricht 0 und L).

Wir beschränken diese Operationen auf die modulo-2-Rechnung, d. h. es gibt nur 2 Reste (Elemente {0,1}). Bei dieser Rechnung sind alle auf den rationalen Zahlen möglichen Operationen definiert:

Multiplikation:	$0 \cdot 0 = 0$	$0 \cdot 1 = 0$	$1 \cdot 0 = 0$	$1 \cdot 1 = 1$
Addition:	$0 + 0 = 0$	$0 + 1 = 1$	$1 + 0 = 1$	$1 + 1 = 0$
Subtraktion:	$0 - 0 = 0$	$0 - 1 = 1$	$1 - 0 = 1$	$1 - 1 = 0$

Da aus -1 nach modulo-2 Operation 1 wird, gibt es keinen Unterschied zwischen Addition und Subtraktion.

In der üblichen Weise sind die Operationen auf Vektoren und Polynomen definiert.

Beispiel $X = (x_1\ x_2\ x_3\ x_4) = (1010)$
$Y = (y_1\ y_2\ y_3\ y_4) = (0011)$
$Z = X + Y = (x_1 + y_1\ x_2 + y_2\ x_3 + y_3\ x_4 + y_4) = (1001)$

2.2.2.5 Binäre Block-Codes

- Folge der zu überarbeitenden Symbole wird in einzelne Blöcke zerlegt, für die jeweils einzeln Fehlererkennung durch Auswertung redundanter Information erfolgt.
- Darstellung des Codewortes als Vektorraum, Beschreibung des Codes durch Zuordnungsliste: Systematik üblich, aber nicht erforderlich.
- Gebräuchliches Verfahren zur Berechnung redundanter Information bei Sender und Empfänger: **Paritätsprüfung**. Dabei wird an das CW eine Prüfstelle (k=1) angehängt, die die Anzahl von Einsen im CW auf eine gerade Zahl (**even parity**) ergänzt.

2.2.2.5.1 Beispiel: Direkter 4-bit-Code mit Paritätsbit

n=5 CW-Stellen, m=4 Nachrichtenstellen, k=1 Kontrollstellen

$$c_D p(a) = c_D(a) \circ \mathrm{par}(c_D(a)), \text{ mit } \mathrm{par}(a) = \begin{cases} 1 & \text{falls ungerade Anzahl von Bitstellen mit einer 1} \\ 0 & \text{sonst} \end{cases}$$

Damit vollständiger Code mit 5-bit-Wörtern:

	$c_D(a)$	$\mathrm{par}(c_D(a))$	$c_D p(a)$
0	0 0 0 0	0	0 0 0 0 0
1	0 0 0 1	1	0 0 0 1 1
2	0 0 1 0	1	0 0 1 0 1
3	0 0 1 1	0	0 0 1 1 0
4	0 1 0 0	1	0 1 0 0 1
5	0 1 0 1	0	0 1 0 1 0
6	0 1 1 0	0	0 1 1 0 0
7	0 1 1 1	1	0 1 1 1 1
8	1 0 0 0	1	1 0 0 0 1
9	1 0 0 1	0	1 0 0 1 0

Das Muster 00100 (illegal) wird erzeugt durch Verfälschung von Bit 2 bei „0“, Bit 0 bei „2“, Bit 1 bei „3“ oder Bit 3 bei „6“.

Erkennung von 1-bit Fehlern, aber keine Korrektur. Anwendung dieser Art von Paritätsprüfung bei V.24-Kopplungen (Terminal-Rechner-Schnittstellen) oder im Arbeitsspeicher des IBM-PC.

Andere Darstellung für die Berechnung der Paritätsbits:
$\mathrm{par}(x) = x \cdot P$, wobei x ein CW mit m Nachrichtenstellen als Spaltenvektor dargestellt und P ein Spaltenvektor der selben Dimension mit lauter Einsen ist.

Berechnung für „6“:

$$\mathrm{par}(x) = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = 0 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 = 0 + 1 + 1 + 0 = 0$$

2.2.2.6 Lineare systematische Codes

Auf eine gegebene Kombination von Nachrichtenstellen wird eine Rechenvorschrift (Prüfschema) angewendet und im Kodierer so der Wert der Prüfstelle $k \geq 1$ ermittelt. Im Dekodierer kommt das gleiche Schema zur Anwendung.

2.2.2.6.1 Beispiel für Prüfschema

(entspricht der Funktion par() bei den binären Block-Codes)

Nachrichtenstellen				Prüfstellen		
x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

Codewort wird hier eingesetzt und jeweils durch Vektormultiplikation durchgeführt. Das Prüfschema besteht aus k Prüfzeilen und $n = m + k$ Prüfspalten. In jeder Zeile gerade Anzahl von Einsen.
$\Rightarrow$ Die Anwendung des Prüfschemas ist Paritätskontrolle mit unterschiedlichen Testmustern.
Das Prüfschema stellt lineares Gleichungssystem dar, jede Prüfzeile entspricht einer linearen Gleichung.

Hier:
1. Zeile: $1 \cdot x_1 + 1 \cdot x_2 + 1 \cdot x_3 + 0 \cdot x_4 + 1 \cdot x_5 + 0 \cdot x_6 + 0 \cdot x_7 = 0 \Leftrightarrow \; x_1 + x_2 + x_3 = x_5$
2. Zeile: $1 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3 + 1 \cdot x_4 + 0 \cdot x_5 + 1 \cdot x_6 + 0 \cdot x_7 = 0 \Leftrightarrow \; x_1 + x_2 + x_4 = x_6$
3. Zeile: $1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 + 0 \cdot x_5 + 0 \cdot x_6 + 1 \cdot x_7 = 0 \Leftrightarrow \; x_1 + x_3 + x_4 = x_7$

2.2.2.6.2 Kompaktere Darstellung

Prüfdaten werden zu Vektoren $P_1 \ldots P_n$ zusammengefaßt. Dann wird ein Gleichungssystem aufgestellt:
$x_1 P_1 + x_2 P_2 + \ldots + x_n P_n = 0$
Für die Berechnung der Prüfstellen werden die Nachrichtenstellen eines zu übertragenden CWs in die obigen Gleichungen eingesetzt. Bedingung für die eindeutige Bestimmung der Kontrollstellen aus den Nachrichtenstellen: Die k „Kontrollstellenvektoren“ $\overline{P_{n-k+1} \ldots P_n}$ müssen zusammengefaßt die Einheitsmatrix bilden.

2.2.2.6.3 Beispiel für die Berechnung der Prüfstellen aus gegebenem Nachrichtenteil

$(x_1\ x_2\ x_3\ x_4) = (1001) \Rightarrow (x_5\ x_6\ x_7) = (100) \Rightarrow$ CW: (1001100)

Der Dekodierer setzt das empfangene CW in das Gleichungssystem ein, alle Gleichungen müssen erfüllt sein, sonst liegt ein Fehler vor.

2.2.2.6.4 Erkennbarkeit von Fehlern

- Fehler wird ausgedrückt durch überlagerte Fehlermuster $F = (f_1\ f_2 \ldots f_n)$, wobei
  $f_i = \begin{cases} 1 & \text{falls Fälschung an der Stelle } i \text{ aufgetreten} \\ 0 & \text{falls Stelle } i \text{ korrekt} \end{cases}$
  Also: empfangenes CW: $x' = (x'_1 \ldots x'_n) = x + F = (x_1 + f_1\ x_2 + f_2 \ldots x_n + f_n)$
- Empfänger prüft CW durch Einsetzen in das Gleichungssystem, dabei resultiert ein Ergebnisvektor:
  $Z = x'_1 P_1 + x'_2 P_2 + \ldots + x'_n P_n$
- 1. Wenn kein Fehler vorliegt, so ist $x' = x$ und $Z = 0$.
  2. Wenn $Z \neq 0$, dann Fehler, aber auch möglich: $Z = 0$, obwohl $F \neq 0$
- Umformung:
  $$\begin{aligned} Z &= x'_1 P_1 + x'_2 P_2 + \ldots + x'_n P_n \\ &= (x_1 + f_1) P_1 + (x_2 + f_2) P_2 + \ldots + (x_n + f_n) P_n \\ &= \underbrace{(x_1 P_1 + x_2 P_2 + \ldots + x_n P_n)}_{0} + (f_1 P_1 + f_2 P_2 + \ldots + f_n P_n) \\ &= 0 + (f_1 P_1 + f_2 P_2 + \ldots + f_n P_n) \end{aligned}$$
- Anwendung der Fehlerüberprüfung auf gefälschtes CW führt auf gleiches Ergebnis wie Anwendung des Prüfschemas auf Fehlermuster direkt. $\Longrightarrow$ Für Erkennbarkeit des Fehlers ist das CW x ohne Bedeutung. $\Longrightarrow$ F ist dann nicht erkennbar, wenn Übereinstimmung mit möglichem Codewort.
- Wenn gewährleistet ist, daß zwei Fehlermuster $F_i \neq F_j$ auf verschiedene Fehler-Syndrome $Z_i \neq Z_j$ führen, dann ist in der Tat Fehlerkorrektur möglich, z. B. über Zuordnungstabellen.
- Aufstellung eines Prüfschemas für eine bestimmte Hammingdistanz möglich, aber aufwendig.

Bedeutung linearer Codes heutzutage gegenüber den zyklischen Codes gering.

2.2.2.7 Zyklische binäre Codes

- Codeworte x_i werden durch Linearkombinationen von Generatorworten erzeugt.
- Generatorworte G_i (m Nachrichten-, k Kontroll- und $n = m + k$ Gesamtstellen) werden durch **Stellenverschiebung** aus einem **Generatormuster** abgeleitet.

2.2.2.7.1 Beispiel

			1	0	1	1	Generatormuster
x_7	x_6	x_5	x_4	x_3	x_2	x_1	
0	0	0	1	0	1	1	G_1
0	0	1	0	1	1	0	G_2
0	1	0	1	1	0	0	G_3
1	0	1	1	0	0	0	G_4
u^6	u^5	u^4	u^3	u^2	u^1	u^0	

2.2.2.7.2 Formalisierung

- Zuordnung von Wertigkeiten $u^0, u^1 \ldots, u^{n-1}$ zu den einzelnen Stellen $x_1, \ldots, x_n$.
- Damit rein formaler Übergang auf Polynome
  für Generatormuster $G(u) = g_k u^k + g_{k-1} u^{k-1} + \ldots + g_1 u^1 + g_0$ (Generatorpolynom mit Grad k)
  und Codeworte $X(u) = x_n u^{n-1} + x_{n-1} u^{n-2} + \ldots + x_2 u^1 + x_1 u^0$ (Codewortpolynom mit Grad $n-1$)
  Interpretation der Polynomkoeffizienten als Zahlenfolgen.
- Polynom für verschiedene Anwendungsfälle von der ITU[4] genormt.

2.2.2.7.3 Obiges Beispiel in Polynomdarstellung

(Generatorpolynom $G(u) = u^3 + u^1 + 1$):

u^6	u^5	u^4	u^3	u^2	u^1	u^0	Wertigkeiten
			1	0	1	1	$G(u)$
		1	0	1	1		$G(u) \cdot u^1$
	1	0	1	1			$G(u) \cdot u^2$
1	0	1	1				$G(u) \cdot u^3$

Codewort ist z. B. die Summe aus zweiter und vierter Zeile: $X(u) = G(u)(u^3 + u^1) = (1001110)$

Wichtig ist: Generatorworte haben die Form $G(u) \cdot u^i$; da alle Generatorworte $G(u)$ als Faktor enthalten, ist $G(u)$ auch in jedem CW von $X(u)$ enthalten. $G(u)$ ist also Teiler aller $X(u)$: $\frac{X(u)}{G(u)} = Q(u)$
$\Rightarrow$ Jedes zyklisch verschobene (rotierte) CW stellt wieder ein CW dar, daher der Name dieser Technik.
Erzeugung eines Codewortes bei beliebiger Wahl der Nachrichtenstellen:
Seien die Nachrichtenstellen beliebig gewählt, dann ergeben sich die k Kontrollstellen durch Polynomdivision.

Im Beispiel:
$G(u) = u^3 + u + 1$ (Grad $k = 3$)
$m = 4$ Nachrichtenstellen $\Rightarrow m + k = 7$ Gesamtstellen
Nachrichtenstellen seien gewählt: (1001)
Gesucht: Gesamtcodewort: (1001???)
Vorgehen:

[4] International Telecommunications Union

- Zunächst Annahme, daß $X = (1001\underline{000})$ sei
- Ergibt sich bei der Division X/G ein Rest, so wird dieser von X abgezogen (vgl. Vorgehen bei der Division ohne Rest bei den natürlichen Zahlen („`div`")).
- Das Resultat ist dann durch $G(u)$ ohne Rest teilbar und damit Codewort.

2.2.2.7.4 Polynomdivision (Horner-Schema)

$\overbrace{(u^6\ u^5\ u^4\ u^3\ u^2\ u^1\ u^0)}^{X(u)}$								$\overbrace{(u^3\ u^2\ u^1\ u^0)}^{G(u)}$					$\overbrace{(u^3\ u^2\ u^1\ u^0)}^{Q(u)}$			
1	0	0	1	0	0	0	/	1	0	1	1	=	1	0	1	0
1	0	1	1													
	0	1	0	0												
	0	0	0	0												
		1	0	0	0											
		1	0	1	1											
			0	1	1	0										
			0	0	0	0										
				1	1	0	⟵ Rest der Division									

Damit ist dann das Codewort: $X = (1001000) - (0000110) = (1001110)$
Wichtig ist, daß bei der Division nur der Rest betrachtet werden muß, $Q(u)$ spielt keine Rolle. Dadurch kann im Falle von Implementierungen erheblich Hardware eingespart werden

- Das Generatorpolynom spielt hier die gleiche Rolle wie das Prüfschema bei den linearen Codes, ist aber viel kompakter.
- Im Dekodierer wird ankommendes Wort mit Hilfe von Polynomdivision durch Generatorpolynom geteilt, falls Rest: Fehler !!
- Auch hier (wie bei den linearen Codes): Fehlersyndrom kann bei Analyse auf gefälschte Nachrichtenstelle führen.
- Kanalsicherung durch zyklische Codes (CRC[5]) sehr verbreitet auf Übertragungsleitungen, aber auch auf Datenträgern (Festplatten, Floppies).

2.2.3 Codes variierender Länge

Bei Codes variierender Länge werden die zu kodierenden Zeichen auf Codewörter unterschiedlicher Länge abgebildet.
Vorteil: Häufig vorkommende Zeichen (z. B. der Buchstabe „E" in deutschen Texten) können mit kurzem CW kodiert werden, seltener auftetende mit einem langen CW. Länge des kodierten Textes geringer als unkodiert.
Nachteil: Bei Serienwortkodierung bzw. -übertragung ist unter Umständen die Trennung der Einzelwörter schwierig. Insbesondere wenn auf der Übertragungsleitung Störungen auftreten oder die Synchronisation zwischen Sender und Empfänger verloren ist.

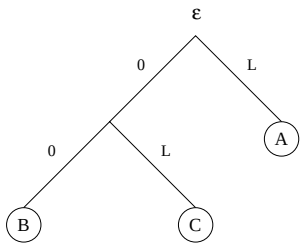
2.2.3.1 Serienwortkodierung

$c^*(\langle a_1 ... a_n \rangle) = c(a_1) \circ c(a_2) \circ \ \ldots \circ c(a_n)$

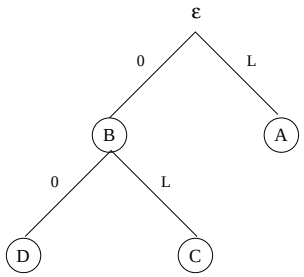
[5] Cyclic Redundancy Check

2.2.3.1.1 Beispiele

1. $A \to L$
   $B \to 00$
   $C \to 0L$
   Zu kodieren sei die Zeichenfolge: AABCA
   $\longrightarrow$ $LL000LL$
   Problem: Start bei der zweiten „0“:
   Dekodierung der Zeichenfolge BAA



2. $A \to L$
   $B \to 0$
   $C \to 0L$
   $D \to 00$
   Zu kodierende Zeichenfolge: AABCDA
   $\longrightarrow$ $LL00L00L$
   Bei Dekodierung möglich:
   AABBABBA
   AADADA



Im zweiten Beispiel ist eine (hinreichende) Voraussetzung für die Eindeutigkeit der Kodierung bei Serienwortübertragungen verletzt: Kein Codewort darf identisch mit dem Anfang eines anderen CW sein. Codewörter sind nur an den Blättern, nicht jedoch im Inneren des Codebaums erlaubt.

2.2.3.1.2 Fano-Eigenschaft

Ist die Bedingung verletzt, dann ist es nicht mehr entscheidbar, ob ein CW bereits eigenständiges Wort oder nur Anfang eines CW ist. Die Fano-Bedingung hinreichend, aber nicht notwendig.

2.2.3.1.3 Beispiel

$A \to L00$
$B \to L0$
Kodierung der Zeichenfolge: AB $\to \underbrace{L0}_{B?}\, 0L0$
Da kein Wort mit „0“ beginnt, kann zurückgeschlossen werden, daß hier A kodiert wurde.

2.2.3.2 Kodierung unter Berücksichtigung der Wahrscheinlichkeit

Häufig wird das Ziel verfolgt, eine gegebene Nachricht einer bestimmten Quelle durch eine Folge von CW zu kodieren, deren Gesamtlänge möglichst kurz ist (Minimierung der Länge der zu kodierenden Nachrichten).
Plausibles Vorgehen: Berücksichtigung der Auftrittswahrscheinlichkeit der von der Quelle gesendeten Zeichen. Ist über die Auftrittswahrscheinlichkeit nichts bekannt, so bleibt nur die Möglichkeit, jedem Zeichen gleichlanges CW zuzuordnen (z. B. ASCII: 2 Zeichen $\Rightarrow$ ld 256 = 8 bit/Zeichen).
Allgemein gilt: Um ein Zeichen aus dem Zeichenvorrat A mit M Elementen zu kodieren, werden bei Binärkodierung ld M bit benötigt.

2.2.3.2.1 Begriffsdefinitionen

1. Die **Auftrittswahrscheinlichkeit** eines Zeichens a ist die Anzahl n der Auftritte des Zeichens in einer zufälligen Zeichenfolge der Länge N für $N \to \infty$, d. h. $p_a = \lim_{N\to\infty} \frac{n}{N}$.

2. Der **Informationsgehalt** oder **Entscheidungsgehalt** eines Zeichens ist der Kehrwert seiner Auftrittswahrscheinlichkeit. Es wird definiert:
   $I_a = \text{ld } \frac{1}{p_a} = -\text{ld } p_a$ in bit.

3. Der von einem von der Quelle ausgegebenen Zeichen zu erwartende mittlere Informationsgehalt wird als **Entropie** („Überraschungswerthaltigkeit“) der Wahrscheinlichkeitsquelle bezeichnet und ist der Mittelwert
   $H = E\{I_i\} = -\sum_{i=1}^{M} p_i \text{ ld } p_i$.
   Je höher die Entropie der Quelle, desto größer ist der Übertragungswert der von ihr ausgegebenen Nachrichten. Die Entropie erreicht ihr Maximum, wenn alle Zeichen gleich wahrscheinlich sind; dann ist $H_0 = I_i = \text{ld } M$. Die Entropie ist niedrig, wenn die Quelle zumeist das gleiche Zeichen aussendet, welches nur selten von einem anderen unterbrochen wird. In diesem Fall reichen also wenige Bits, um die Nachrichten zu repräsentieren.

2.2.3.2.2 Beispiel

Quelle mit den Zeichen „1“ und „0“
Mit der Auftrittswahrscheinlichkeit p wird „1“ gesendet, mit der Auftrittswahrscheinlichkeit $1 - p$ wird „0“ gesendet. Dann ist die Entropie dieser Quelle:
$H = -p \text{ ld } p - (1 - p) \text{ ld } (1 - p)$
Für $p = 0,5$ ist die Entropie $H = 0,5 \cdot 1 + 0,5 \cdot 1 = 1$ bit, aber für $p = 0,25$ ist sie $H = 0,8$ bit.

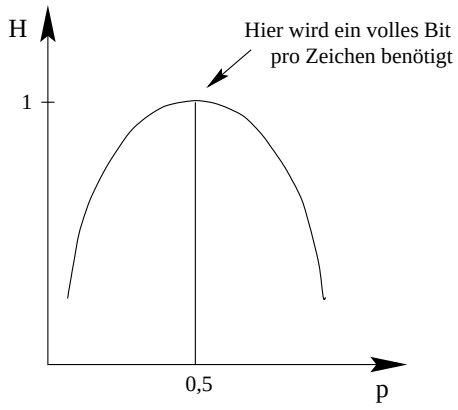


Abbildung 2.5: Entropie-Wahrscheinlichkeits-Diagramm

Die Länge der CW wird klein, wenn jedes Bit (eines CW) den maximalen Informationsgehalt transportiert, wenn also bei jedem Bit die Wahrscheinlichkeit für „1“ bei $p = 0,5$ und für „0“ ebenfalls bei $p = 0,5$ liegt. $\Rightarrow$ Maximaler „Überraschungswert“ für jedes Bit.
Bei der Serienübertragung engt jedes Bit den „Spielraum“ für noch mögliche Wörter aus der Menge der CW weiter ein. Bei einem (näherungsweise) optimalen Code muß daher jedes weitere Bit die noch verbleibende Menge an CW in näherungsweise gleichwahrscheinliche Teilmengen zerlegen.

2.2.3.2.3 Beispiel

8 Zeichen seien zu kodieren.

Zeichen i	$\diamond$	$\triangle$	$\square$	$\bigcirc$	$\wedge$	$\vee$	$\rightarrow$	$\leftarrow$
p_i	$0,5$	$0,1$	$0,05$	$0,03$	$0,09$	$0,08$	$0,07$	$0,08$

2.2.3.2.3.1 Möglichkeit zur Kodierung

Konstruktion eines Codebaums von der Wurzel her („top down"), bei dem sich die Wahrscheinlichkeit an jedem Knoten gleichmäßig auf die unteren Teilbäume aufspaltet. Zuordnung der CW so, daß ihre Auftrittswahrscheinlichkeit möglichst nahe an die aufgeteilte Wahrscheinlichkeits-Masse kommt.

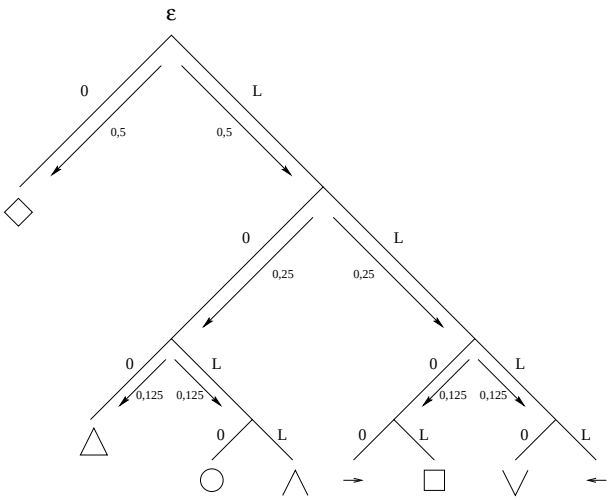


Abbildung 2.6: ausgeglichener Codebaum (in Bezug auf die Wahrscheinlichkeitsmassen)

Damit Codetabelle:

Zeichen	p_i	CW	$p_i \cdot L_i$
$\diamond$	$0,5$	0	$0,5$
$\triangle$	$0,1$	$L00$	$0,3$
$\square$	$0,05$	$LL0L$	$0,2$
$\bigcirc$	$0,03$	$L0L0$	$0,12$
$\wedge$	$0,09$	$L0LL$	$0,36$
$\vee$	$0,08$	$LLL0$	$0,32$
$\rightarrow$	$0,07$	$LL00$	$0,28$
$\leftarrow$	$0,08$	$LLLL$	$0,32$

L_i: Länge des CW in bits

$\sum = 2,4$ bit/Zeichen

Gegenüber Gleichverteilung[6] (3 bit/Zeichen) Einsparung von 20%.
Unbefriedigend: Abweichung der geforderten Wahrscheinlichkeiten an den Blättern von der tatsächlichen Auftrittswahrscheinlichkeit läßt letzte Sicherheit über die tatsächliche Erreichung des Optimums vermissen.

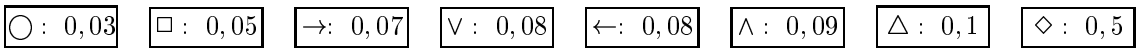
[6] Gleichverteilungsannahme: einzige Möglichkeit bei unbekannter Auftrittswahrscheinlichkeit

2.2.3.2.3.2 Deshalb bessere Möglichkeit

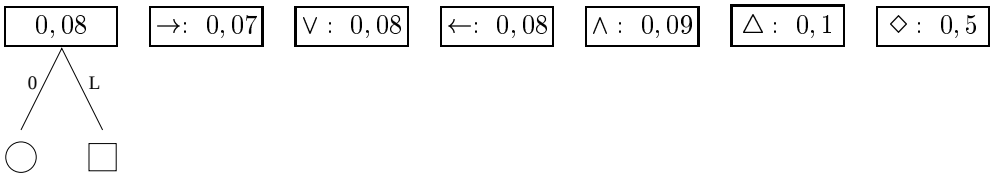
Konstruktion des Codebaumes von den Blättern her („bottom up“), ausgehend von der Beobachtung, daß die am seltensten auftretenden Zeichen am weitesten unten im Codebaum angeordnet sind.
Einfache Vorschrift zur Konstruktion des Baumes: Bei jedem Teilschritt werden die Teilbäume, die jeweils die geringste summierte Zeichenhäufigkeit aufweisen, zusammengefaßt.

Illustration am Beispiel:

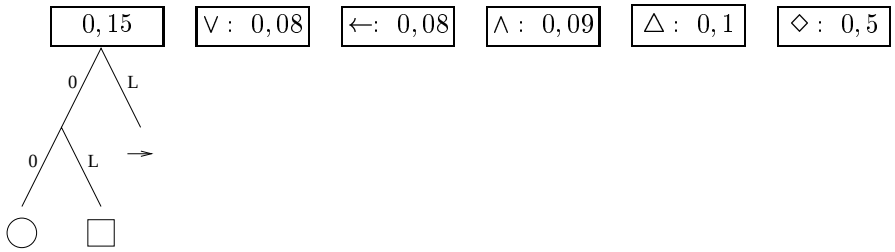
1. Schritt: Liste mit Zeichen, sortiert nach Häufigkeiten



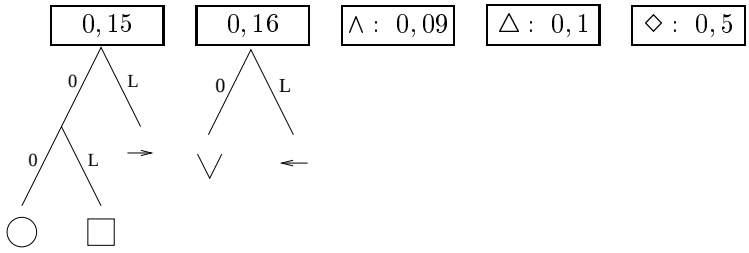
2. Schritt: Zusammenfassung der beiden geringsten Häufigkeiten



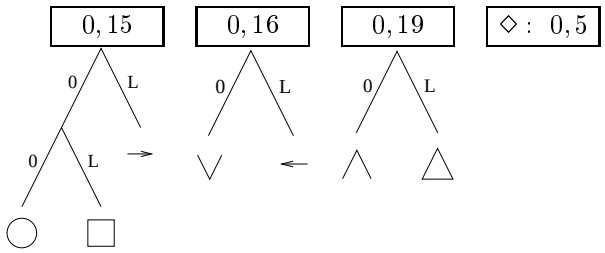
3. Schritt: Zusammenfassung der beiden geringsten Häufigkeiten



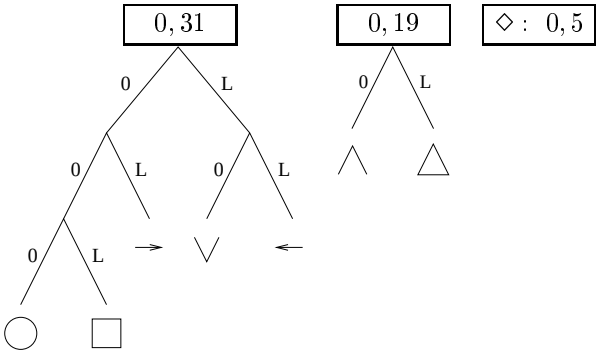
4. Schritt: Zusammenfassung der beiden geringsten Häufigkeiten



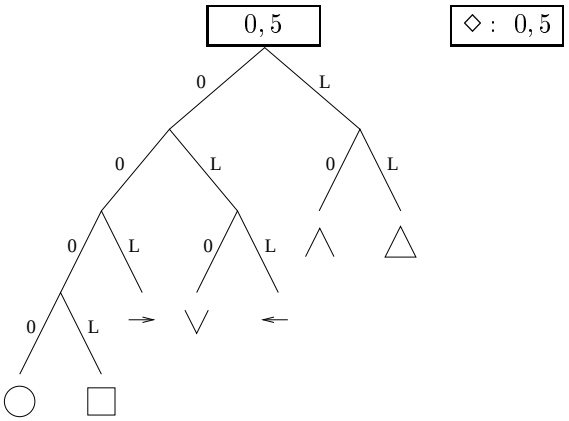
5. Schritt: Zusammenfassung der beiden geringsten Häufigkeiten



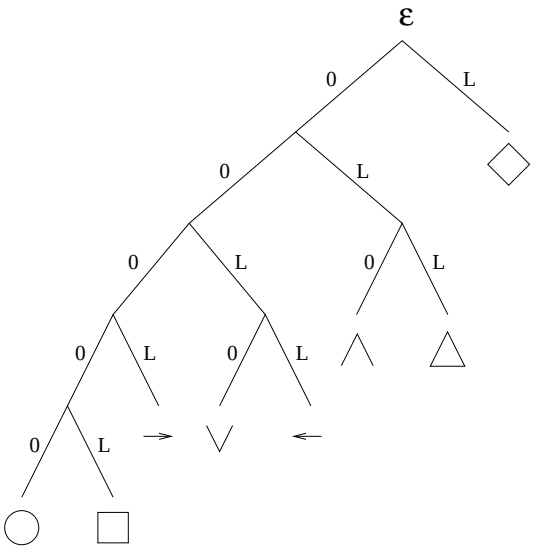
6. Schritt: Zusammenfassung der beiden geringsten Häufigkeiten



7. Schritt: Zusammenfassung der beiden geringsten Häufigkeiten



8. Schritt: Endgültiger Codebaum



Damit Codetabelle:

Zeichen	p_i	CW	$p_i \cdot L_i$	$p_i \cdot \mathrm{ld}\ p_i$
$\diamond$	$0,5$	L	$0,5$	$-0,5$
$\triangle$	$0,1$	$0LL$	$0,3$	$-0,332$
$\square$	$0,05$	$0000L$	$0,25$	$-0,216$
$\bigcirc$	$0,03$	00000	$0,15$	$-0,152$
$\wedge$	$0,09$	$0L0$	$0,27$	$-0,313$
$\vee$	$0,08$	$00L0$	$0,32$	$-0,292$
$\rightarrow$	$0,07$	$000L$	$0,28$	$-0,269$
$\leftarrow$	$0,08$	$00LL$	$0,32$	$-0,292$
			$\sum = 2,39$	$\sum = 2,364$

Prinzip des **Huffmann-Coding** (1952): Verfahren erzeugt bei gegebenen Auftrittswahrscheinlichkeiten Optimum bezüglich der Länge der Kodierung. Vielfach technische Anwendung.

2.2.3.3 Abschließende Bemerkungen zur Kodierung

- statische Methoden
- „Lexikon-basierte" Methoden

Notwendig für Lexikon-basierte Methoden: Modellierung der Quelle:

- statische Methoden: gleiches Modell für alle Texte
- semiadaptive Methoden: Bestimmung eines Modells durch Analyse (eines Teils) des Textes. Problem: Text muß vor Beginn der Kodierung komplett beim Kodierer bekannt sein.
- adaptive Methoden: ausgehend von einem Initialmodell (z. B. alle Zeichen gleich wahrscheinlich) adaptieren Kodierer und Dekodierer schritthaltend nach dem selben Algorithmus ihr Modell. Problem: Nur sinnvoll auf fehlerfreier Übertragungsstrecke.

Außerdem oft nützlich: Berücksichtigung des Kontexts.
Beispiel Modellbildung ohne Kontext:
„u" hat eine Wahrscheinlichkeit von $2,4\% \Rightarrow 5,38$ bits; „u" nach „q" hat allerdings eine Wahrscheinlichkeit von $99\% \Rightarrow 0,014$ bits.
Quelle:
Bell et. al., „Modeling for Text Compression"
ACM Computing Survey, Vol. 21, No. 4, Dec. 89.

2.2.3.4 Lexikon-basierte Kodierung

Ersetzung von Zeichengruppen durch einen Zeiger auf „Codebuch"-Eintrag.

- Erste Entwurfsentscheidung: Welche Zeichengruppen kommen ins Lexikon?
- Zweite Entscheidung: Wie wird der zu kodierende Text in diese Zeichengruppen aufgeteilt?
  - 1. Möglichkeit: Kodierer sucht den jeweils längsten Codebucheintrag, der mit dem nächsten Zeichen übereinstimmt (nicht unbedingt optimal, aber schnell).
  - 2. Möglichkeit: LFF (Longest Fragment First): Hier sucht der Kodierer nach dem längsten Substring, der mit einem Eintrag im Codebuch übereinstimmt, dann nach dem zweitlängsten usw., bis der Text komplett passiert ist.

2.2.3.4.1 Beispiel

Lexikon: $M = \{a, b, c, aa, aab, ab, baa, bccb, bccba\}$
9 Lexikoneinträge $\Rightarrow$ jeder String kann in 4 Bits gepackt werden.
Text: $'aaabccbaaaa' = 11 \cdot 8 = 88$ bits bei ASCII
Nach Parsierung: Unterteilung in 5 Fragmente (Zeichengruppen):

$$\underbrace{aa}_{2}\ \underbrace{a}_{4}\ \underbrace{bccba}_{1}\ \underbrace{aa}_{3}\ \underbrace{a}_{5}$$

$5 \cdot 4 = 20$ bits

Häufigst angewendetes Verfahren zur systematischen Kodierung:

2.2.3.4.2 Lempel-Ziv-Kompression (LZC)

Familie von verschiedenen Algorithmen für unterschiedliche Anforderungen:
LZxx (xx bedeutet Veröffentlichungsjahr), z. B.
LZ77: Grundlage für gzip,
LZ78: Ziv and Lempel Compression of individual sequences via variable rate coding, IEEE Transactions on Information Theory, IT-24, No. 5, Sept. 78

2.2.3.4.2.1 Vorgehen

Text wird in Gruppen zerlegt, eine Gruppe K aus N Zeichen wird repräsentiert als Zeiger auf die Gruppe J aus $N-1$ Zeichen, die mit den ersten $N-1$ Zeichen von K übereinstimmt und einem zusätzlichen Zeichen. J bezeichnet man als „Präfix“ von K. Darstellung aller Präfixe eines Codebuchs in einem Baum für schnellen Zugriff.

2.2.3.4.2.2 Beispiel

Text: $'aaabbabaabaaabab'$
Lineare Parsierung von links:

	$\underbrace{a}$	$\underbrace{aa}$	$\underbrace{b}$	$\underbrace{ba}$	$\underbrace{baa}$	$\underbrace{baaa}$	$\underbrace{bab}$
Gruppen-Nr.:	1	2	3	4	5	6	7
Ergebnis:	$(\perp, a)$	$(1, a)$	$(\perp, b)$	$(3, a)$	$(4, a)$	$(5, a)$	$(4, b)$

$16 \cdot 8 = 128$ bits

- Zeiger benötigt bei Parsierung von p Gruppen ld p bits. Jeder Einzelbuchstabe benötigt $q = 8$ bits. Im Beispiel: 7 Gruppen $\Rightarrow$ 3 bit/Zeiger + 8 bit/Zeichen für jede Gruppe, also $7 \cdot 11 = 77$ bits für den Gesamtstring.
- Hohe Kompressionsrate bei großer Gruppenlänge (linearem Textzuwachs steht logarithmischer Zeigerlängenzuwachs gegenüber).
- Problem: Lexikon benötigt immer mehr Speicherplatz bei fortschreitender Traversierung des Textes. Vermeidungsstrategie: Wenn allokierter Speicherplatz gefüllt, wird das Lexikon gelöscht und vollständig neu aufgebaut (Adaptivität bleibt erhalten).

Literaturliste

[Schi92] Schiffmann, Schmitz
Technische Informatik I, II
Springer-Verlag, 1992

[Broy91] M. Broy
Informatik I, II
Springer-Verlag, 1991

[Ober90] Oberschlepp, Vossen
Rechnerarchitektur
Oldenbourg-Verlag, 1990

[Hoff93] P. Hoffmann
Rechnerentwurf, 3. Aufl.
Oldenbourg-Verlag, 1993

[Schn85] G. M. Schneider
The Principles of Computer Organization
Wiley, 1985

[Hama90] Hamacher, Vranesie, Zuky
Computer Organization
McGraw-Hill, 1990

[Patt93] Patterson, Hennessy
Computer Organization and Design
Morgan Kaufmann, 1993

[Flic94] Flick, Liebig
Mikroprozessortechnik, 4.Aufl.
Springer-Verlag, 1994

[Baue91] Bauer, Wösner
Informatik I, II
Springer-Verlag, 1991

[Catt88] K. Cattermole
Statistische Analyse und Struktur von Information
Weinheim: VCH-Verlagsgesellschaft, 1988

[Heis95] W. Heise, P. Quattrocchi
Informations- und Kodierungstheorie
Springer-Verlag, 1995

Stichwortverzeichnis

A
Absorptionsgesetz 1
aktiver Bereich 26
Alphabet 41
Alphabetabbildung 10
Ampelschaltung 18
AND 3
Antivalenz 3
Arbeitsgerade 25
Arbeitspunkt 25
Assoziativgesetz 1, 5
Auftrittswahrscheinlichkeit 50
Ausgabealphabet 10, 16
Ausgabefunktion 16
Automatengraph 19

B
Basis 26
Betriebsspannung 24
bistabiles Bauelement 20
Block-Codes 44
Bool'sche Algebra 1
bottom up 52

C
Code 41
Codebaum 49, 51, 53
Codes
    Block-Codes 44
    lineare systematische 45
    zyklische binäre 47
Codewürfel 43

D
De Morgan 2, 5
Diode 29
Distributivgesetz 1, 5
DNF 4
DTL-Technik 29

E
Eingabealphabet 9, 16
Eingabewort 16
Eins 3
elementarer Zeichenvorrat 41
Emitter 26
Entropie 50
Entscheidungsgehalt 50
Erkennbarkeit von Fehlern 46
even parity 44

F
Fano-Eigenschaft 49
Fehlererkennbarkeit 46
Flimmerschaltung 12
Flip-Flop 20
    einfaches: RS-Flip-Flop 20
    Jump/Kill 22
    Jump/Kill-Master-Slave 22
    Master-Slave 22
    taktflankengesteuertes 21
    taktzustandsgesteuertes 21

G
Generatormuster 47
Generatorwort 47
Gesetz von De Morgan 2, 5
Gray-Code 43

H
Hammingabstand 42, 44
Horner-Schema 48
Huffmann-Coding 54

I
Idempotenzgesetz 1, 5
Implikation 3
Impulsdiagramm 13
Informationsgehalt 50
Inhibition 3

K
K-Diagramm 6
Kanonisierungsregel 6
Karnaugh-Diagramm 6
KDNF 4
Kodierung
    adaptive Methode 54
    Lexikon-basierte Kodierung 54
    Lexikon-basierte Methode 54
    nach Wahrscheinlichkeit 49
    semiadaptive Methode 54
    Serienwortkodierung 48
    statische Methode 54
Kollektor 26
kombinatorischer Automat 9, 10

einfacher 10
elementarer 9
Kommutativgesetz 1
Kontrollstellenvektoren 46

L

Laststrom 24
Lastwiderstand 24
Laufzeit 13, 21
Lempel-Ziv-Kompression 55
Lexikon 54
Logik
Dioden-Transistor-Logik 29
Transistor-Transistor-Logik 30
Widerstands-Transistor-Logik 27

M

Maxterm 4
Mealy-Automat 16
Minterm 4
Moore-Automat 16

N

NAND 3
Negation 2, 3, 5
doppelte 5
Negationsoperator 2
neutrales Element 2, 5
NOR 3
Normalform 3
disjunktive 4
kanonische disjunktive 4
vollständige disjunktive 4
Null 3

O

OR 3

P

Polynomdivision 48
Primimplikand 7

Q

Quine-McClusky 7

R

Redundanz 40, 43
Relais 23
RTL 27

S

Schalter 24
elektromechanischer 23
elektronischer 24
Schaltfunktion 3
Schaltzeichen 2
Speicher 11
Stellenverschiebung 47
Steuerstrom 24

T

Takthalbierer 15
Taktsteuerung 12
Taktflankensteuerung 12
Taktzustandssteuerung 12
Zweiflankensteuerung 12, 22
Zweizustandssteuerung 12
Term 4
top down 51
Transfer 3
Transistor 24
Doppel-Emitter-Transistor 30
Multi-Emitter-Transistor 30
TTL-Technik 30

U

Überraschungswerthaltigkeit 50
Übertragung
in Serie 41
parallel 41

V

Verband 1
distributiver 1
Vereinfachungsregeln 5
Verzögerungsglied 12

W

Wertetabelle 2
Wort 41

X

XOR 3

Z

Zeichenfolge 41
Zeichenmenge 41
Zeichenvorrat
elementarer 41
Zustandsalphabet 16
Zustandswort 16
Zweiflankensteuerung 12, 22
Zweizustandssteuerung 12