

# **Grundlagen der Informatik und Technische Informatik**

FH-Hagenberg und JKU Linz

Thomas Müller-Wipperfürth

---

# Vorbemerkung

Diese Unterlagen dienen dazu, den in der Vorlesung gebrachten Stoff zu unterstützen, ersetzen aber keinesfalls die Vorlesung. Die Inhalte wurden von Thomas Müller-W. verfaßt und zusammengestellt und von Christian Steinmayr mittels L<sup>A</sup>T<sub>E</sub>X aufbereitet. Kapitel 1 wurde im Rahmen einer Seminararbeit von Studierenden erstellt. Die Unterlagen werden laufend verbessert, Hinweise auf noch enthaltene Fehler und Verbesserungsvorschläge sind jederzeit willkommen.

# Inhaltsverzeichnis

<b>1 Einführung</b>	<b>1</b>
1.1 Geschichte des Computers . . . . .	1
1.1.1 Mechanische Computer . . . . .	1
1.1.2 Die ersten elektronischen Computer . . . . .	2
1.1.3 Die ersten vier Generationen von Computern . . . . .	3
1.1.4 Die fünfte Generation und darüber hinaus . . . . .	3
1.2 Digitale Systeme . . . . .	3
1.2.1 Digitale versus analoge Systeme . . . . .	3
1.2.2 Organisation eines digitalen Computers mit Speicher . . . . .	5
1.3 Entwurf Digitaler Systeme . . . . .	5
1.3.1 Abstraktionsebenen . . . . .	5
<b>2 Zahlensysteme und Codes</b>	<b>7</b>
2.1 Einführung . . . . .	7
2.2 Überblick . . . . .	7
2.2.1 Definition von Zahlen . . . . .	7
2.2.2 Die Stellenschreibweise . . . . .	8
2.2.3 Das Polyadische Zahlensystem (Basis r) . . . . .	8
2.2.4 Das Stellenwertsystem . . . . .	8
2.2.5 Zahlensysteme von Bedeutung . . . . .	9
2.2.6 Zahlensysteme im Vergleich . . . . .	9
2.2.7 Beispiele zum Umgang mit Zahlensystemen . . . . .	10
2.3 Zahlenkonvertierung . . . . .	10
2.3.1 Beispiele für die Umwandlung von Zahlen in verschiedene Systeme . . . . .	10
2.3.2 Beispiel Oktalsystem ( $r=8$ ) . . . . .	11
2.3.3 Beispiel Hexadezimalsystem ( $r=16$ ) . . . . .	11
2.4 Umrechnungsverfahren . . . . .	12
2.4.1 Umrechnung von Zahlen der Basis A nach B . . . . .	12
2.4.2 Sukzessive Multiplikation mit Addition . . . . .	12
2.4.3 Sukzessive Division mit Rest . . . . .	13
2.4.4 Sukzessive Multiplikation (nach Horner-Schema) . . . . .	14
2.4.5 Allgemeines Verfahren zur Konvertierung von Zahlen der Basis A nach B . . . . .	15
2.5 Rechnen in beliebigen Zahlensystemen . . . . .	16
2.5.1 Rechnen im Dezimalsystem . . . . .	16
2.5.2 Rechnen im Dualsystem . . . . .	17
2.5.3 Rechnen im Hexadezimalsystem . . . . .	20
2.5.4 Rechnen im Oktalsystem . . . . .	22

2.6	Darstellung negativer Zahlen . . . . .	23
2.6.1	Vorzeichen-Betrag-Darstellung (Signed-Magnitude) . . . . .	23
2.6.2	Exzeß-Darstellungen (Excess-Codes) . . . . .	24
2.6.3	Komplement-Darstellung . . . . .	25
2.6.4	Rechnen mit Einser-Komplement-Darstellung . . . . .	27
2.6.5	Rechnen mit Zweier-Komplement-Darstellung . . . . .	28
2.6.6	Vergleich der Darstellungsarten . . . . .	29
2.7	Gleitkomma-Darstellung . . . . .	30
2.7.1	Einführung in die Gleitkomma-Darstellung . . . . .	30
2.7.2	Allgemeine Darstellung einer Gleitkommazahl F . . . . .	31
2.7.3	Gleitkomma-Formate (Floating-Point Formate) . . . . .	32
2.7.4	Gleitkomma Multiplikation . . . . .	34
2.7.5	Gleitkomma Addition . . . . .	36
2.8	Zeichen- und Zahlendarstellung mit Codes . . . . .	38
2.8.1	ASCII . . . . .	38
2.8.2	BCD . . . . .	38
2.8.3	7-4-2-1 Code . . . . .	39
2.8.4	Übersicht von Zahlencodes . . . . .	39
2.9	Fehlererkennende und Fehlerkorrigierende Codes . . . . .	40
2.9.1	Erweiterung mit Paritätsbits . . . . .	40
2.9.2	Hamming-Code-Tabelle . . . . .	40
<b>3</b>	<b>Boolesche Algebra und logische Schaltungen</b> . . . . .	<b>41</b>
3.1	Schaltalgebra (Boolesche Algebra) . . . . .	41
3.1.1	Definition . . . . .	41
3.1.2	Schaltalgebra . . . . .	42
3.1.3	Mengenlehre . . . . .	42
3.1.4	Beweis, dass die Schaltalgebra eine Boolesche Algebra ist . . . . .	43
3.1.5	Rechenregeln . . . . .	44
3.1.6	Weitere Boolesche Algebren . . . . .	44
3.1.7	Eigenschaften von 0 und 1 . . . . .	44
3.2	Schaltausdruck (bzw. Schaltformen) . . . . .	44
3.3	Prinzip der Dualität . . . . .	45
3.4	Grundgatter und deren Darstellungen . . . . .	45
3.4.1	Darstellung der Grundgatter mit NOR bzw. NAND . . . . .	46
3.5	Schaltfunktionen (Boolesche Funktionen) . . . . .	47
3.5.1	Realisierung durch Gatterschaltungen . . . . .	47
3.6	Schaltnetze . . . . .	48
3.6.1	Realisierung von Schaltnetzen mit NAND/NOR-Gattern . . . . .	49
3.7	Normalformen (Standard Forms) . . . . .	51
3.7.1	Kanonische Normalformen (Canonical Forms) . . . . .	52
3.7.2	Erstellen einer DKNF (disjunktiv kanonische Normalform) . . . . .	52
3.7.3	Erstellen einer KKNF (konjunktiv kanonische Normalform) . . . . .	54
3.7.4	Komplementierung einer kanonischen Normalform . . . . .	56
3.8	Beispiele . . . . .	57
3.9	Vereinfachung von Schaltfunktionen . . . . .	59

3.9.1	Minimierung mittels KV-Diagrammen . . . . .	61
3.9.2	Vereinfachung . . . . .	66
3.9.3	Weitere Beispiele . . . . .	67
<b>4</b>	<b>Kombinatorische Grundschaltungen</b>	<b>71</b>
4.1	Decodierer (Decoder) . . . . .	71
4.2	Codierer mit Priorität (Priority Encoders) . . . . .	73
4.3	Multiplexer (Selektoren) . . . . .	74
4.4	Demultiplexer (Distributor) . . . . .	76
4.5	Ripple-Carry Adder . . . . .	76
4.5.1	Was versteht man unter einem Ripple-Carry Adder? . . . . .	76
4.5.2	Carry-Look-Ahead Adder (CLA) . . . . .	79
4.5.3	Addierer/Subtrahierer . . . . .	80
4.6	Logische Einheit (Logic Unit, LU) . . . . .	82
4.6.1	Arithmetische logische Einheit . . . . .	83
4.7	Busse . . . . .	85
<b>5</b>	<b>Sequentielle Logik</b>	<b>87</b>
5.1	Endliche Automaten (Finite State Machines) . . . . .	87
5.1.1	Begriffserklärung . . . . .	87
5.1.2	Beispiele . . . . .	88
5.2	Latches und FlipFlops . . . . .	92
5.2.1	RS-Latch . . . . .	92
5.2.2	RS-Latch mit Enable . . . . .	96
5.2.3	D-Latch . . . . .	97
5.2.4	Master-Slave-D-Flip-Flop . . . . .	98
5.2.5	JK-Flipflop . . . . .	99
5.2.6	Toggle Flipflop . . . . .	99
5.2.7	Vergleich: D-Latch, D-FF, JK-FF . . . . .	100
5.3	Zähler (Counter) . . . . .	102
5.3.1	Asynchrone (Serielle) Zähler . . . . .	102
5.3.2	Synchrone (Parallele) Zähler . . . . .	103
5.3.3	Setup- und Holdzeit . . . . .	104
<b>6</b>	<b>Schaltwerks-Entwurf</b>	<b>105</b>
6.1	Beispiele . . . . .	105
6.2	Beispiel Roboter . . . . .	107
6.2.1	Codierung von A,B,Q . . . . .	107
6.2.2	Wahrheitstabellen für $\delta$ und $\lambda$ . . . . .	107
6.2.3	Schaltwerk . . . . .	107
6.2.4	Aufspulen von $\delta_S$ und $\lambda_S$ in Schaltfunktionen . . . . .	108
6.2.5	Schaltwerk (Gatterebene) . . . . .	110
6.3	Weitere Beispiele . . . . .	111



# Abbildungsverzeichnis

1.1	Abacus . . . . .	1
1.2	Addiermaschine von Blaise Pascal (1623-1662) . . . . .	1
1.3	Struktur eines klassischen von Neumann-Rechners . . . . .	2
1.4	analoges Signal . . . . .	4
1.5	Sample-and-Hold Darstellung eines analogen Signals . . . . .	4
1.6	digitales Signal . . . . .	4
1.7	Beispiele auf verschiedenen Abstraktionsebenen . . . . .	6
2.1	Vorzeichen-Betrag-System . . . . .	23
2.2	Einserkomplement . . . . .	27
2.3	Zweierkomplement . . . . .	28
3.1	Venn-Diagramme: Vereinigung, Durchschnitt, Komplement . . . . .	42
3.2	Logische Funktionen und Schaltsymbole der Grundgatter . . . . .	45
3.3	Wahrheitstabellen der Grundgatter . . . . .	46
3.4	Realisierung der Grundgatter mittels NAND- und NOR-Gatter . . . . .	46
3.5	Schaltfunktion . . . . .	47
3.6	Gatterschaltbild einer Schaltfunktion . . . . .	47
3.7	Schaltnetz . . . . .	48
3.8	Gatterschaltbild eines Schaltnetzes . . . . .	48
3.9	KV-Diagramme für AND, OR und XOR . . . . .	61
3.10	KV-Diagramme für $f(a,b,c)$ . . . . .	61
3.11	KV-Diagramm mit zwei Variablen . . . . .	62
3.12	KV-Diagramm mit drei Variablen . . . . .	63
3.13	KV-Diagramm mit vier Variablen . . . . .	63
3.14	KV-Diagramm mit fünf Variablen . . . . .	64
3.15	KV-Diagramm mit sechs Variablen . . . . .	64
4.1	Schaltbild und Symbol eines Decoders . . . . .	71
4.2	3-zu-8 Decodierer mit 1-zu-2 Decodierern aufgebaut . . . . .	72
4.3	3-zu-8 Decodierer mit 2-zu-4 Decodierern aufgebaut . . . . .	72
4.4	Schaltbild und Symbol eines Codierers . . . . .	73
4.5	Symbol eines 4-zu-2 Codierers . . . . .	73
4.6	Schaltbild und Symbol eines Multiplexers . . . . .	75
4.7	8-zu-1 Selektor mit einem Decodierer . . . . .	75
4.8	Schaltbild und Wertetabelle eines Demultiplexers . . . . .	76
4.9	Demultiplexer mit Enable-Signal . . . . .	76

4.10	Serielle Datenübertragung . . . . .	76
4.11	Ripple-Carry Adder: KV-Diagramm für $s_i$ und $c_{i+1}$ . . . . .	77
4.12	Schaltbild eines 1-Bit Volladdierers . . . . .	77
4.13	Schaltbild eines 8-Bit Volladdierers . . . . .	78
4.14	Schaltbild eines 4-Bit Ripple-Carry Adders . . . . .	78
4.15	Schaltbild eines 4-Bit CLA Generators . . . . .	80
4.16	Symbol eines 4-Bit Addierers mit CLA Generatoren . . . . .	80
4.17	Schaltbild eines 16-Bit Addierers mit 2-Ebenen CLA Generator . . . . .	81
4.18	Schaltbild eines 8-Bit Addierer/Substrahierer . . . . .	81
4.19	Schaltbild einer 1-Bit logischen Einheit . . . . .	82
4.20	Schaltbild eines AR (arithmetic extenders) . . . . .	83
4.21	Schaltbild eines LE (logic extenders) . . . . .	84
4.22	Schaltbild für eine 4-Bit ALU . . . . .	85
4.23	Schaltbild eines tristate drivers . . . . .	85
4.24	Bus mit 4 Eingängen . . . . .	86
5.1	Mealy- und Moore-Typ . . . . .	87
5.2	Beispiel Mealy-FSM: Erkennen von 'ab' . . . . .	88
5.3	Beispiel Moore-FSM . . . . .	88
5.4	Cola-Automat . . . . .	89
5.5	Fahrtscheinautomat . . . . .	90
5.6	Beispiel Roboter . . . . .	91
5.7	Ablaufdiagramm . . . . .	91
5.8	Schaltung und Schaltsymbol - RS-Latch . . . . .	92
5.9	RS-Latch: S=0, R=0 . . . . .	92
5.10	RS-Latch: S=0→1, R=0, Fall1 . . . . .	92
5.11	RS-Latch: S=0→1, R=0, Fall2 . . . . .	93
5.12	RS-Latch: S=1→0, R=0 . . . . .	93
5.13	RS-Latch: S=0, R=0→1 . . . . .	93
5.14	RS-Latch: S=0, R=1→0 . . . . .	94
5.15	RS-Latch: S=1, R=1 ('verbotene' Eingangswerte) . . . . .	94
5.16	RS-Latch: S=1, R=1→0 (kein Problem) . . . . .	94
5.17	RS-Latch: S=1→0, R=1→0 (instabile Ausgabewerte) . . . . .	95
5.18	Zustandsdiagramm und Schaltsymbol eines RS-Latches . . . . .	95
5.19	Zeitdiagramm - RS-Latch (NOR-Gatter) . . . . .	96
5.20	Schaltung und Schaltsymbol - RS-Latch mit Enable . . . . .	96
5.21	Schaltung und Schaltsymbol - D-Latch . . . . .	97
5.22	Zeitdiagramm - D-Latch . . . . .	97
5.23	Schaltung und Schaltsymbol - D-FF . . . . .	98
5.24	Zeitdiagramm - MS-D-FF . . . . .	98
5.25	Flankengesteuertes JK-Flipflop (Schaltung und Schaltsymbol) . . . . .	99
5.26	Schaltsymbol eines Toggle Flipflop . . . . .	99
5.27	Realisierung von Toggle Flipflops ohne Enable . . . . .	99
5.28	Realisierung von Toggle Flipflops mit Enable . . . . .	99
5.29	D-Latch, D-FF, JK-FF . . . . .	100
5.30	D-Latch, D-FF, JK-FF (mit Lösung) . . . . .	101

5.31 Setup- und Holdzeit . . . . .	104
6.1 Beispiel Mealy . . . . .	106
6.2 Schaltwerk . . . . .	107
6.3 Ausgabefunktion . . . . .	108
6.4 Überführungsfunktion . . . . .	108
6.5 Vereinfachung der Robotersteuerung mit Hilfe von KV . . . . .	109
6.6 Schaltwerk für Robotersteuerung in Gatterebene . . . . .	110
6.7 Mensch . . . . .	111
6.8 Ein Zähler in Moore-FSM . . . . .	111
6.9 Erkennen von '011' . . . . .	112
6.10 Beispiel: Erkennen von 'bbbb' (Mealy-FSM) . . . . .	112
6.11 Beispiel: Erkennen von 'aba' (Mealy-FSM) . . . . .	112
6.12 Beispiel: x . . . . .	113



# Tabellenverzeichnis

1.1	Abstraktionsebenen	6
2.1	Übersicht von Zahlensystemen	9
2.2	Addition im Dezimalsystem	16
2.3	Multiplikation im Dezimalsystem	16
2.4	Addition im Dualsystem	17
2.5	Multiplikation im Dualsystem	17
2.6	Verfahren zur Multiplikation im Dualsystem	18
2.7	Addition im Hexadezimalsystem	20
2.8	Multiplikation im Hexadezimalsystem	20
2.9	Addition im Oktalsystem	22
2.10	Multiplikation im Oktalsystem	22
2.11	Übersicht gängiger Excess-Codes	24
2.12	Übersicht von Codes für negative Zahlen	29
2.13	Probleme der Gleitkomma-Darstellung	30
2.14	Exponential-Darstellung	30
2.15	IEEE Floating-Point Formate	33
2.16	Floating-Point Formate von drei Maschinen	33
2.17	ASCII	38
2.18	BCD Code	38
2.19	7-4-2-1 Code	39
2.20	Übersicht von Zahlencodes	39
2.21	BCD Code mit Paritätsbit	40
2.22	Hamming-Code-1	40
3.1	Rechenregeln	44
3.2	Eigenschaften von 0 und 1	44
3.3	Wertetabelle	48
4.1	Wertetabelle für einen 4-zu-2 Coder (mit Priorität)	74
4.2	Wertetabelle für einen Volladdierer	77
4.3	Vergleich der Verzögerungszeiten verschiedener Addierer-Techniken	81
4.4	bool'sche Funktionen mit zwei Variablen	82
4.5	Funktionstabelle des AE	83
4.6	Funktionstabelle des LE	84
5.1	Zusammenfassung RS-Latch	95

5.2 Wertetabelle des RS-Latch (links: Realisierung mit NOR-Gatter, rechts: Realisierung mit NAND-Gatter) . . . . .	96
5.3 Wahrheitstabelle - RS-Latch mit Enable . . . . .	97
5.4 Wahrheitstabelle - D-Latch . . . . .	97
5.5 Wahrheitstabelle - MS-D-FF . . . . .	98
5.6 Wahrheitstabelle - JK-D-FF . . . . .	99

# Kapitel 1

## Einführung

### 1.1 Geschichte des Computers

#### 1.1.1 Mechanische Computer

Der erste Computer ist wahrscheinlich der Abacus (siehe Abbildung 1.1), der seit über 3000 Jahren im Orient benutzt wird.

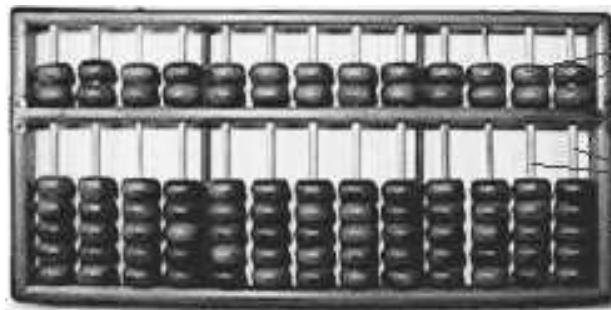


Abbildung 1.1: Abacus

Erst um das Jahr 1600 (also ca. 2600 Jahre nach der Entwicklung des Abacus) nutzte John Napier Logarithmen als Basis für ein Gerät, welches Zahlen multiplizieren konnte. Sein Gerät führte zur Erfindung des *Rechenschiebers*. 1642 baute Blaise Pascal (1623-1662) eine Addiermaschine mit geschalteten Rädern (siehe Abbildung 1.2), ähnlich dem Prinzip des modernen *Odometers*.

Langenscheidt: Odometer [eu'damIte] s mot. Am. Meilenzähler m.

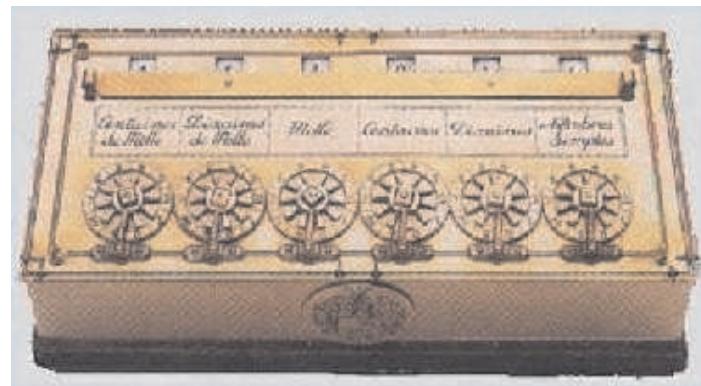


Abbildung 1.2: Addiermaschine von Blaise Pascal (1623-1662)

1830 entwickelte Charles Babbage die 'Analytical Engine'. Das ist das erste Gerät, welches Prinzipien des modernen Computers benutzt. Der Betrieb bisheriger mechanischer Rechenmaschinen bestand aus einem Eingabemechanismus für die Daten, einem Kurbelantrieb, einer Ablesevorrichtung für Zwischenergebnisse und einer Vorrichtung zum Ordnen von Ergebnissen.

Das Konstruktionsprinzip von Babbage sah nun bereits folgende Komponenten vor:

- *Arbeitsspeicher* (zur Speicherung von Eingabedaten, Zwischen- und Endergebnissen),
- *Rechenwerk* (zur Durchführung arithmetischer Operationen),
- *Steuerwerk* (zur Steuerung der Reihenfolge der Rechenprozesse und des Datentransports) sowie
- *Ein- und Ausgabeelemente*

(vgl. Von-Neumann-Architektur)

Abbildung 1.3 zeigt die Struktur eines klassischen von Neumann-Rechners

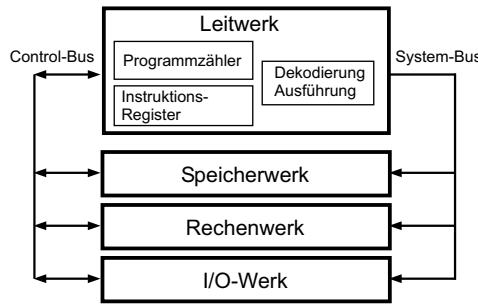


Abbildung 1.3: Struktur eines klassischen von Neumann-Rechners

Da aber zu dieser Zeit eine Massenproduktion von Präzisionszahnrädern nicht möglich war, und die Zahnradgetriebe bei ihren Bewegungen einen so grossen Energiebedarf gehabt hätten, dass die Abgabe von Kraft und Drehmoment an weitere Getriebe zu klein geworden wäre, konnte die 'Analytical Engine' nicht in Masse produziert werden.

### 1.1.2 Die ersten elektronischen Computer

Der erste Schritt in Richtung digitale Computer gelang Howard Aiken von der Harvard Universität und George Slabitz von Bell Telephone Laboratories. Sie entwickelten in den späten 1930er Jahren eine automatische Rechenmaschine auf der Basis von Relaisnetzwerken.

Während des 2. Weltkrieges wurden noch andere Relaismaschinen für militärische Zwecke entwickelt. Doch alle diese hatten den Nachteil, dass sie relativ langsam und sehr gross waren.

In den frühen 1940er Jahren bauten John Mauchly und J. Presper Eckert von der Universität Pennsylvania auf der Basis von Vakuum-Röhren einen Computer, den sie **electronic numerical integrator and calculator (ENIAC)** nannten. ENIAC bestand aus 18.000 Elektronenröhren, welche einen enormen Energieverbrauch hatten. Außerdem hatte er eine hohe Fehlerrate und war mittels Stechbrett nur schwer zu programmieren.

In den folgenden Jahren wurden drei Entdeckungen gemacht, mit denen die rapide Entwicklung zum heutigen Computer begann:

1. **John von Neumann** schlug vor, das Programm im Speicher des Computers zu halten, wodurch es nach Belieben verändert werden konnte.
2. **1947** wurde von den Herren Bardeen, Brattain und Scheckley der **Transistor** erfunden. Dieser reduzierte den Strom- und Platzverbrauch gegenüber der Elektronenröhre gewaltig.
3. J. W. Forrester und seine Kollegen vom Massachusetts Institute of Technology (MIT) entwickelten den **magnetic core memory**, wodurch erstmals ein Speicher mit grösserer Kapazität realisierbar wurde.

### 1.1.3 Die ersten vier Generationen von Computern

1. ENIAC und andere Computer mit **Elektronenröhren**, die in den späten 40er und während der 50er Jahren gebaut wurden, zählt man zur ersten Generation digitaler Computer.
2. Mit dem Siegeszug des Transistors in den späten 50er Jahren kam die **zweite Generation** digitaler Computer auf den Markt. Diese Computer, die **Transistoren** statt den grossen energiefressenden Röhren nutzten, waren wesentlich kleiner und schneller als ihre Vorgänger.
3. In den späten 60er und während den 70er Jahren etablierte sich die **dritte Generation** von digitalen Computern auf dem Markt. Diese Maschinen nutzten **integrierte Schaltkreise** (integrated circuits IC) anstatt einer Menge einzelner Transistoren. In einem IC ist eine Menge von Transistorschaltkreisen integriert. Diese Integration brachte eine weitere drastische Reduzierung der Computergrösse.

Mit der Entwicklung der dritten Generation tauchten erstmals in den späten 60er Jahren die **Minicomputer** auf. Zusätzlich zu den komplexen Grosscomputern (auch Mainframes genannt) brachten viele Hersteller diese kleineren Computer mit geringerer Kapazität auf den Markt. Die Minicomputer, die ihren Namen von der geringeren Grösse und den geringeren Kosten hatten, waren es, die den Einsatz des Computers auf breiter Basis ermöglichen. Computergesteuerte Prozesse und Anlagen wurden in der Industrie zum Standard.

4. Mit der fortwährenden Weiterentwicklung der ICs kam Ende der 70er und Anfang der 80er Jahre eine weitere Generation zum Durchbruch. Bei der **vierten Generation** digitaler Computer sind die Hardwarekomponenten in **large scale integrated (LSI)** und **very large scale integrated (VLSI) circuits** Technologie realisiert.

Mit Hilfe der VLSI Technologie wurde es möglich, kleine aber leistungsstarke Computer zu bauen. Bekannt sind diese Computer als Personal Computer (PC) oder als Workstations. Die zentrale Komponente dieser Maschinen ist der Prozessor. Der Prozessor (central processing unit, kurz: CPU) ist in einem einzigen VLSI-Baustein realisiert. Die Entwicklung des Mikroprozessors ist von zwei führenden Herstellern geprägt: Intel Corporation und Motorola. In letzter Zeit ist AMD auf der Überholspur.

Die Vermarktung der PCs, wie der IBM Personal Computer (Intel Prozessor) und der Apple Macintosh (Motorola Prozessor) trug mehr als alles andere dazu bei, dass die Computer heutzutage diesen enormen Stellenwert haben. Bevor der PC auf so breiter Basis zur Anwendung kam, wurden die meisten Computer ausschliesslich von Computerexperten benutzt. Heutzutage werden die Computer auch von Nichtexperten genutzt. Man braucht kein Spezialist sein, um einen Computer zu bedienen. Gleicher gilt für Computernetzwerke, die auch erst durch die vierte Generation realisierbar wurden. Durch diese Netzwerke wurden auch völlig neue Anwendungen (E-mail, Internet, ...) möglich.

### 1.1.4 Die fünfte Generation und darüber hinaus

Wann beginnt die fünfte Computergeneration? Oder hat sie schon begonnen? Gebrauchen wir die klassische Methode einen Generationswechsel zu identifizieren, nämlich den Sprung zu einer neuen Hardwaretechnologie, so ist die Antwort 'nein'. Aber ist die verwendete Hardware der einzige Indikator einer neuen Technologie? Wahrscheinlich nicht.

Es ist klar, dass Fortschritte in der Softwaretechnologie genauso tiefgründige Effekte in der Nutzung von Computern hervorrufen, wie die Hardwareentwicklung. Neue Benutzerschnittstellen wie Sprachaktivierung, oder neue Paradigmen wie parallel processing und Neuronale Netze, können genauso die nächste Generation von Computern charakterisieren.

Wo auch immer die Grenze zur nächsten Generation liegen mag, es scheint sicher, dass parallel processing, künstliche Intelligenz, optical processing, visuelles Programmieren und leistungsstarke Netzwerke Schlüsselrollen in Computersystemen der Zukunft spielen werden.

## 1.2 Digitale Systeme

### 1.2.1 Digitale versus analoge Systeme

Bei einem digitalen System oder Gerät wird die Information in **diskreter** Form dargestellt und verarbeitet. Das bedeutet, dass das System mit einer fix vorgegebenen endlichen Anzahl an Werten arbeitet. Im Gegensatz dazu arbeiten analoge Systeme mit Information in kontinuierlicher Form. Bei analogen Systemen ist der Wertebereich zwar auch eingeschränkt, jedoch ergibt sich eine praktisch unendliche Anzahl an unterschiedlichen Werten, da die

Abstufung von einem Wert zum nächsten unendlich klein ist (bei digitalen Systemen ist dies jeweils der kleinste darstellbare Wert).

**Diskrete Zahlenwerte:** Zahlenwerte, die durch endliche Intervalle voneinander getrennt stehen (Math., Phys.).  
(c) Dudenverlag.

Eine Uhr, welche die Zeit mittels Stunden, Minuten und Sekundenzeiger anzeigt, ist ein Beispiel für ein analoges System. Wohingegen eine Uhr mit einer Anzeige in Ziffernform ein digitales Gerät darstellt. Die Musik auf traditionellen Audiokassetten ist in analoger Form aufgenommen, dagegen ist die Musik auf CDs in digitaler Form gespeichert. Eine modernere Art von Audiokassetten (MiniDisk) speichern die Musik ebenfalls in digitaler Form.

Abbildung 1.4 zeigt ein analoges Signal wie es z.B. auf einer herkömmlichen Audiokassette vorkommt.

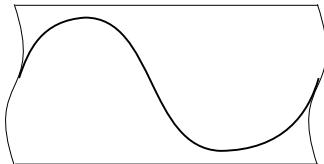


Abbildung 1.4: analoges Signal

Abbildung 1.5 zeigt das gleiche Signal, wenn es in einheitlichen Zeitabständen abgetastet wird und der Wert bis zum nächsten Abtasten gehalten wird (Sample and Hold).

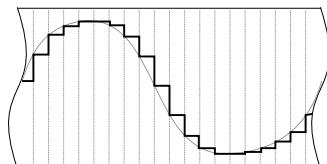


Abbildung 1.5: Sample-and-Hold Darstellung eines analogen Signals

Jedem Wert aus Abbildung 1.5 wird eine binäre Nummer zugewiesen (digitalisiert). Diese Werte werden vertikal auf das Band gespeichert (siehe Abbildung 1.6). Das gleiche Signal in digitaler Form wird in Abbildung 1.6 dargestellt, wobei jeder Wert aus Abbildung 1.5 durch eine binäre Nummer repräsentiert wird und vertikal auf das Band geschrieben wird.

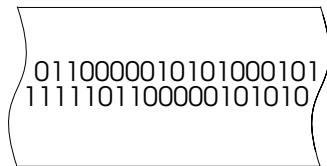


Abbildung 1.6: digitales Signal

Der Computer ist zwar das klassische Beispiel eines digitalen Systems, jedoch gibt es auch viele andere Beispiele wie digitale Uhren, Verkehrsampeln, Taschenrechner usw. Alle diese Geräte mit Ausnahme des Computers sind Systeme mit fixer Funktionalität, die vom Benutzer nicht modifiziert werden können. Im Gegensatz dazu ist der Computer ein programmierbares System, d.h. sein Verhalten kann vom Benutzer verändert werden. Mit anderen Worten, der Computer ist ein Universalsystem, während die anderen genannten Beispiele anwendungsspezifische Systeme darstellen.

In letzter Zeit werden in Autos, bei elektronischen Spielen oder diversen Steuerungen immer öfter Computer anstatt anwendungsspezifischer Schaltkreise (ASICs) eingesetzt. Zu diesem Zweck wird ein Programm mit dem gewünschten Verhalten entwickelt, ein Mikrocontroller damit programmiert und dieser fertig programmierte Controller in das Produkt integriert. Auf diese Art werden immer mehr spezifische Schaltkreise durch Mikrocontroller

ersetzt, da es wesentlich einfacher ist, für eine Anwendung ein neues Mikrocontrollerprogramm zu entwickeln, als eine gänzlich neue Schaltung. Gewöhnlich ist dies auch mit einer erheblichen Kostensparnis verbunden. Analoge Computer und andere analoge Systeme waren schon in Gebrauch lange bevor digitale Schaltkreise perfektioniert wurden.

Warum dennoch digitale Systeme die analogen auf immer mehr Gebieten verdrängen, hat mehrere Gründe:

- Generell bieten digitale Techniken mehr Flexibilität als analoge, da digitale Systeme leicht programmiert werden können und somit verschiedenste Algorithmen ausführen.
- Numerische Information kann digital mit grösserer Präzision über einen weiteren Bereich dargestellt werden als mit analogen Signalen.
- Speichern und wieder Auslesen von Information ist in digitaler Form wesentlich leichter handzuhaben als in analoger Form.
- Digitale Techniken bieten Möglichkeiten zur Fehlererkennung und Fehlerkorrektur (z.B. Fehlerkorrekturcodes in der Übertragungstechnik).
- Digitale Systeme können leichter miniaturisiert werden als analoge.

### 1.2.2 Organisation eines digitalen Computers mit Speicher

Die Grundstruktur eines digitalen Computers besteht aus der **arithmetisch/logischen Einheit (ALU)**, der **Kontrolleinheit (CPU)** und der **Eingabe/Ausgabe Einheit (I/O)**.

Jedes Computersystem kennt eine gewisse Menge von Befehlen, die sogenannten Maschinenbefehlen. Üblicherweise wird jedes Programm in die Maschinensprache umgewandelt, welche nur aus Maschinenbefehlen besteht. Diese Befehle spezifizieren Operationen, welche von der arithmetisch/logischen Einheit an den Daten angewandt werden als auch Interaktionen zwischen der arithmetisch/logischen Einheit, der E/A Einheit und dem Speicher.

## 1.3 Entwurf Digitaler Systeme

Man versteht darunter nicht nur eine Technik zum Hardwareentwurf, sondern einen gesamten Prozess, in dem mehrere Personen (Projektteam) zu verschiedenen Aspekten des Endprodukts beitragen. Alle Mitarbeiter haben ihre eigenen Fachkenntnisse, sehen so das Produkt aus ihrer Sicht und verwenden zur Realisierung verschiedene Werkzeuge. Gemeinsam setzen sie ein Konzept in ein reales Produkt um.

### 1.3.1 Abstraktionsebenen

Elektronische Systeme können auf verschiedenen Abstraktionsebenen betrachtet und entworfen werden. Diese reichen von der Systemebene, in der keine Hardwaredetails spezifiziert sind, bis zur physikalischen Sicht eines Produkts. Tabelle 1.1 zeigt eine Zusammenfassung aller wichtigen Ebenen und gibt Beispiele dafür an.

Tabelle 1.1 lässt sich auch graphisch darstellen, wie Abbildung 1.7 zeigt. Eine Spezifikation eines Systems stellt einen Punkt in dieser Ebene dar. Je weiter aussen man sich befindet, desto weniger Detailwissen ist erforderlich. Jedoch können dann grössere Komplexitäten beschrieben werden.

#### Definition des Verhaltens

Das spätere Produkt wird zunächst als 'Black Box' betrachtet. Die Spezifikation konzentriert sich auf das Verhalten der Ausgänge in Abhängigkeit der Eingänge und der Zeit. Man beschreibt die Funktionalität, also die Reaktion des Produkts auf alle möglichen Eingaben, nicht die Struktur eines vorgegebenen Designs. Verzichtet wird auf eine Angabe wie die einzelnen Komponenten implementiert werden sollen. Häufig verwendete Darstellungsformen sind z.B. Algorithmen, Formeln und einfache Diagramme.

#### Beschreibung der Struktur

Das Produkt wird als Kombination einzelner Komponenten und deren Verbindung untereinander beschrieben. Spezifiziert wird nicht das Verhalten, sondern die Implementierung ohne explizite Referenz auf die Funktionalität. Letztere kann in einigen Fällen durchscheinen, da man erahnen kann, wie die Komponenten aufeinander wirken. Allerdings ist das äusserst unwahrscheinlich wenn die Anzahl der Komponenten eine bestimmte Grenze (z.B. 10.000

Ebene	Verhaltensebene	Strukturebene	Layout
System	ausführbare Spezifikationen, Programme	Prozessoren, Steuerungen, Speicher, ASICs	Multichipmodule
Register	Algorithmen, Ablaufdiagramme, endliche Automaten	Addierer, Vergleichsoperatoren, Register, Registersätze, Datenpfade	Mikrochips
Gatter	boolesche Gleichungen	Gatter, Flip-Flops	Module
Transistor	Differentialgleichungen	Transistoren, Widerstände, Kondensatoren	Analog- und Digitalzellen

Tabelle 1.1: Abstraktionsebenen

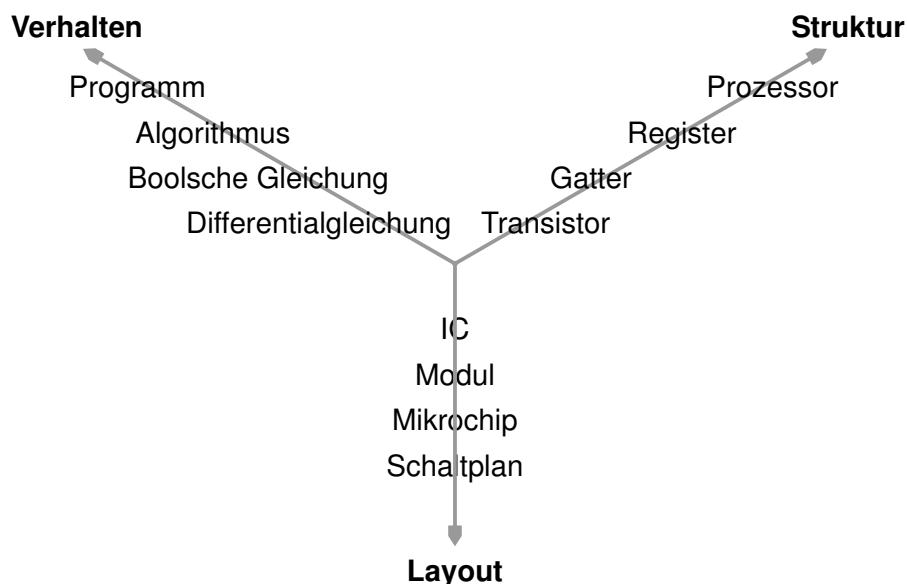


Abbildung 1.7: Beispiele auf verschiedenen Abstraktionsebenen

Stück) erreicht oder überschreitet. Vor allem auch deshalb, weil es die heutige Technologie erlaubt, Mikrochips mit mehreren Millionen Transistoren zu konstruieren.

### Beschreibung des Layouts

Spezifiziert werden die physikalischen Charakteristiken des Produkts, also die Dimensionen und Positionen der einzelnen Komponenten, sowie deren Verbindungen. Während bei einer strukturellen Beschreibung nur angegeben wird, welche Teile miteinander verbunden sind, werden hier die Relationen genau definiert. Diese Repräsentation dient auch dazu, das Produkt nach dessen Herstellung zu beschreiben und Angaben bezüglich Gewicht, Größe, Hitzeempfindlichkeit, Energieaufnahme und Position der Pins machen zu können.

# Kapitel 2

## Zahlensysteme und Codes

### 2.1 Einführung

Zahlen spielen in der Informationsverarbeitung eine wichtige Rolle, wie folgende Beispiele verdeutlichen sollen:

- Lösen von linearen Gleichungssystemen ( $y = kx + d$ )
- näherungsweise Lösung von Differentialgleichungen
- numerische Integration
- Standardfunktionen ( $\sin(x)$ ,  $\cos(x)$ ,  $\log(x)$ , . . . )
- digitale Signalverarbeitung (FFT, ...)
- Sprach-/Bilderkennung und -verarbeitung

Deshalb ist es notwendig, Zahlen computergerecht darzustellen, wobei an diese Darstellungsweise besondere Anforderungen gestellt werden:

- Arithmetische Operationen (ADD, SUB, MUL, DIV) sollten leicht durchführbar sein.
- Die elektronische Realisierung sollte einfach sein.
- Der Zahlenbereich soll genügend gross sein.
- Die Darstellung soll der Gewohnheit (Dezimalsystem) entsprechen.
- Fehler in der Darstellung einer Zahl sollen erkennbar, unter Umständen auch korrigierbar sein.

Da sich diese Anforderungen zum Teil einander ausschliessen oder einschränken, wird man bei der Auswahl einer Zahlendarstellung Kompromisse schliessen müssen.

### 2.2 Überblick

#### 2.2.1 Definition von Zahlen

**Definition 1:** Zahlen sind Worte über einem endlichen Alphabet von Symbolen, die eindeutig einen bestimmten Zahlenwert repräsentieren.

**Definition 2:** Ein Zahlensystem ist die Gesamtheit der zur Darstellung einer Zahl verwendeten Zahlzeichen (Ziffern) und Regeln für deren Zusammensetzung.

### 2.2.2 Die Stellenschreibweise

$$N = (a_{n-1}a_{n-2}\dots a_1a_0, a_{-1}a_{-2}\dots a_{-m})_r \quad (2.1)$$

$a_i$	Ziffern
,	Komma (Radixpunkt), teilt gebrochenen Anteil vom ganzzahligen Anteil
$r$	Radix, Basiszahl
$a_{n-1}$	MSD most significant digit
$a_{-m}$	LSD least significant digit

*Beispiel 1:*

123,35 entspricht  $(123, 35)_{10}$

### 2.2.3 Das Polyadische Zahlensystem (Basis r)

Bei dem uns vertrauten Zahlensystem - dem Dezimalsystem - handelt es sich um ein polyadisches Zahlensystem. Als Zeichenvorrat (Ziffern) stehen die Zahlen 0 bis  $r - 1$  zur Verfügung. Jede Zahl wird durch eine Ziffernfolge repräsentiert, wobei jede Ziffer aufgrund ihrer Position gewichtet wird. Der Wert einer Zahl ergibt sich dann aus der Summe aller gewichteten Ziffern. Die Basis  $r$  (= Radix) des Zahlensystems gibt an, wie gross diese Gewichtung ist und wie viele Ziffern (= Zeichenvorrat) verfügbar sind.

Eine Zahl mit  $n$  Vorkommastellen und  $m$  Nachkommastellen hat folgendes Aussehen (allgemeine Darstellung einer Zahl als Ziffernfolge zur Basis  $r$ ):

$$\begin{aligned} N &= \sum_{i=-m}^{n-1} a_i r^i \\ &= a_{n-1} r^{n-1} + a_{n-2} r^{n-2} + \dots + a_1 r + a_0 + a_{-1} r^{-1} + a_{-2} r^{-2} + \dots + a_{-m} r^{-m} \end{aligned} \quad (2.2)$$

*Beispiel 2:*

$$(123, 35)_{10} = 1 * 10^2 + 2 * 10^1 + 3 * 10^0 + 3 * 10^{-1} + 5 * 10^{-2}$$

### 2.2.4 Das Stellenwertsystem

Jeder Stelle wird ein (frei wählbarer) Wert zugeordnet.

$$N = \sum_{i=-m}^{n-1} a_i w_i \quad (2.3)$$

Verglichen mit dem polyadischen Zahlensystem können hier die einzelnen Stellen unterschiedlich gewichtet werden.

### 2.2.5 Zahlensysteme von Bedeutung

- $r = 2$  ... Binärzahlen, Dualzahlen
- $r = 8$  ... Oktalzahlen
- $r = 10$  ... Dezimalzahlen
- $r = 16$  ... Hexadezimalzahlen

Im **Dualsystem** ist (wie der Name schon sagt) die Basis **r** gleich **2**. Daher gibt es im Dualsystem für die Zifferndarstellung nur zwei Symbole: 0 und 1. Der Wert N einer gegebenen Dualzahl mit den Ziffern  $a_i$  lässt sich entsprechend Formel 2.4 folgendermassen berechnen.

$$N = \sum_{i=-m}^{n-1} a_i 2^i \quad (2.4)$$

Das Dualsystem ( $r=2$ ) ist eines Sonderfall des Binärsystem: Unter einem Binärsystem versteht man ein Zahlensystem mit einem Zeichenvorrat von 2 Zeichen.

#### Kurzbezeichnung:

Bit = Binary Digit

### 2.2.6 Zahlensysteme im Vergleich

Tabelle 2.1 zeigt eine Übersicht der Zahlensysteme mit Basis 10, 2, 8 und 16.

Basiszahl	Dezimal r=10	Dual r=2	Oktal r=8	Hexadezimal r=16
Ziffern	0,1,2,...,9	0,1	0,1,...,7	0,1,...,9, A,B,...,F
	0	0000	0	0
	1	0001	1	1
	2	0010	2	2
	3	0011	3	3
	4	0100	4	4
	5	0101	5	5
	6	0110	6	6
	7	0111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	D
	14	1110	16	E
	15	1111	17	F
	16	10000	20	10
	17	10001	21	11
	.	.	.	.

Tabelle 2.1: Übersicht von Zahlensystemen

### 2.2.7 Beispiele zum Umgang mit Zahlensystemen

*Beispiel 3:*

$$(10101)_2 = 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 21_{10}$$

*Beispiel 4:*

$$\begin{aligned} (10,101)_2 &= 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} = \\ &= 2_{10} + 0 + 0,5_{10} + 0 + 0,125_{10} = \\ &= 2,625_{10} \end{aligned}$$

*Beispiel 5:*

$$\begin{aligned} (0,1111)_2 &= 1 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3} + 1 * 2^{-4} = \\ &= 0,5_{10} + 0,25_{10} + 0,125_{10} + 0,0625_{10} = \\ &= 0,9375_{10} \end{aligned}$$

## 2.3 Zahlenkonvertierung

Unter Zahlenkonvertierung versteht man die Umwandlung einer Zahl von einem Zahlensystem in die gleichwertige Zahl eines anderen Zahlensystems.

### 2.3.1 Beispiele für die Umwandlung von Zahlen in verschiedene Systeme

*Beispiel 6:* Umwandlung vom Dualsystem ins **Dezimalsystem**

$$\begin{aligned} 10011,01_2 &=?_{10} \\ &= 1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} = \\ &= 16 + 2 + 1 + 0,25 = 19,25_{10} \end{aligned}$$

*Beispiel 7:* Umwandlung vom Dezimalsystem ins **Dualsystem**

$$\begin{aligned} 25,5_{10} &=?_2 \\ 25 - 16 &= 9 \\ 9 - 8 &= 1 \\ 1 - 4 &= - \\ 1 - 2 &= - \\ 1 - 1 &= 0 \\ &\Rightarrow 11001,1_2 \end{aligned}$$

*Beispiel 8:* Umwandlung vom Hexadezimalsystem ins **Oktalsystem**

- 2D<sub>16</sub> = ?<sub>8</sub>
1. direkt ... siehe Tabelle 2.1: Zahlensysteme
  2. Umweg über Dezimalsystem  

$$\begin{aligned} 2D_{16} &= 2 * 16^1 + 13 * 16^0 = 45_{10} \\ 45_{10} &= 5 * 8^1 + 5 * 8^0 = 55_8 \end{aligned}$$
  3. Umweg über Dualsystem  
 aus Tabelle: 2<sub>16</sub> = 0010<sub>2</sub>, D<sub>16</sub> = 1101<sub>2</sub>  
 $\Rightarrow: 2D_{16} = 00|10'1|101_2 = 55_8$

### 2.3.2 Beispiel Oktalsystem ( $r=8$ )

Nachfolgend soll eine Dualzahl in eine Oktalzahl umgewandelt werden. Dabei ersetzt jede Oktalziffer eine **Dreiergruppe** von Dualziffern. Man erhält also folgende Ersetzungstabelle:

Dual	000	001	010	011	100	101	110	111
Oktal	0	1	2	3	4	5	6	7

*Beispiel 9:* Umwandlung einer Dualzahl in eine **Oktalzahl**

$$11010110001_2 = ?_8$$

$$11'010'110'001_2 = 3261_8$$

$$11101110, 11001_2 = ?_8$$

$$11'101'110, 110'01_2 = 356, 62_8$$

Will man eine Oktalzahl in eine entsprechende Dualzahl konvertieren, ersetzt man einfach jede Oktalziffer durch die entsprechende 3-stellige Dualzahl.

*Beispiel 10:* Umwandlung einer Oktalzahl in eine **Dualzahl**

$$7353_8 = ?_2$$

$$7353_8 = 111'011'101'011_2$$

$$53, 21_8 = ?_2$$

$$53, 21_8 = 101'011, 010'001_2$$

### 2.3.3 Beispiel Hexadezimalsystem ( $r=16$ )

Das Hexadezimalsystem wird auch **Sedezimalsystem** genannt.

In den folgenden Ausführungen soll eine Dualzahl in eine Hexadezimalzahl umgewandelt werden. Die Vorgangsweise ist prinzipiell dieselbe wie bei der Umwandlung ins Oktalsystem. Der einzige Unterschied ist, dass statt Dreiergruppen nun **Vierergruppen** von Dual ersetzt werden.

Dual	0000	0001	0010	0011	0100	0101	0110	0111
Hexadezimal	0	1	2	3	4	5	6	7
Dual	1000	1001	1010	1011	1100	1101	1110	1111
Hexadezimal	8	9	A	B	C	D	E	F

*Beispiel 11:* Umrechnen **ins** Hexadezimalsystem

$$10'1110'0101, 0110'11_2 = 2E5,6C_{16}$$

*Beispiel 12:* Umrechnen **vom** Hexadezimalsystem

$$\text{CAFFEE}_{16} = 1100'1010'1111'1111'1110'1110_2$$

## 2.4 Umrechnungsverfahren

Nicht immer muss die Basis des Zahlensystems eine Zweierpotenz sein. In diesem Kapitel werden Verfahren zur Umwandlung einer Zahl eines beliebigen Zahlensystems in die entsprechende Zahl eines anderen Zahlensystems vorgestellt.

### 2.4.1 Umrechnung von Zahlen der Basis A nach B

**Rechnen in Basis B**

**Verfahren:** Aufstellen der gewichteten Summe und Auswertung der Stellen in Basis B

*Beispiel 13:*  $10100_2 = ?_{10}$

$$1 * 2^4 + 1 * 2^2 = 16_{10} + 4_{10} = 20_{10}$$

*Beispiel 14:*  $274_8 = ?_{10}$

$$2 * 8^2 + 7 * 8^1 + 4 * 8^0 = 128_{10} + 56_{10} + 4_{10} = 188_{10}$$

*Beispiel 15:*  $FF_{16} = ?_{10}$

$$15 * 16^1 + 15 * 16^0 = 240_{10} + 15_{10} = 255_{10}$$

*Beispiel 16:*  $1101,011_2 = ?_8$

$$1 * 2^3 + 1 * 2^2 + 1 * 2^0 + 1 * 2^{-2} + 1 * 2^{-3} = \mathbf{10}_8 + 4_8 + 1_8 + 0,2_8 + 0,1_8 = 15,3_8$$

**Nachteil:** Bilden der Potenzen der Basis A.  $\Rightarrow$  Sukzessive Multiplikation mit Addition

### 2.4.2 Sukzessive Multiplikation mit Addition

**Anmerkung:**

Auch bei diesem Verfahren erfolgen die Zwischenrechnungen im Zahlensystem der Ergebnisbasis. Die Berechnung aller Potenzen der Basis A wird durch wiederholtes Multiplizieren des Zwischenergebnisses mit der Basis A ersetzt.

**Rechnen mit der Basis B**

**Verfahren:** Wiederholtes Multiplizieren mit der Basis A und Addition einer Ziffer pro Rechenschritt.

Wichtig: **gerechnet wird in der Ergebnisbasis.**

$$((a_{n-1} * r + a_{n-2}) * r + a_{n-3}) * r + \dots + a_0 \quad (2.5)$$

*Beispiel 17:*

$$1011011_2 = ?_{10}$$

$$\begin{array}{rcl} & 1 & \\ 2 * 1 + 0 & = 2 & \\ 2 * 2 + 1 & = 5 & \\ 2 * 5 + 1 & = 11 & \\ 2 * 11 + 0 & = 22 & \\ 2 * 22 + 1 & = 45 & \\ 2 * 45 + 1 & = 91 & \\ & \Rightarrow 91_{10} & \end{array}$$

### 2.4.3 Sukzessive Division mit Rest

Umwandlung von Zahlen der Basis A nach B mit Hilfe des **Horner-Schemas**.

**Rechnen in Basis A**

**Anmerkung:** Bei diesem Verfahren erfolgen die Zwischenrechnungen im Zahlensystem der umzurechnenden Zahl.

Wie wir schon gesehen haben, lässt sich der Wert einer **natürlicher Zahl** folgendermaßen berechnen:

Der Ausdruck

$$N = \sum_{i=0}^{n-1} a_i r^i = a_{n-1} r^{n-1} + a_{n-2} r^{n-2} + \dots + a_1 r^1 + a_0 \quad (2.6)$$

lässt sich auch folgendermaßen darstellen:

$$(((a_{n-1}r + a_{n-2})r + a_{n-3})r + \dots)r + a_1)r + a_0 \quad (2.7)$$

*Beispiel 18:*

$$234_{10} = ?_8$$

$$234_{10} = ?_{16}$$

$$234_{10} = ?_2$$

Bei der Division der umzuwendelnden Zahl durch die Basis B ist der entstehende Rest eine Ziffer des Ergebniszahlensystems. Teilt man die umzuwendende Zahl durch r, so bleibt  $a_0$  als Rest. Bei der Division des Quotienten bleibt  $a_1$  als Rest, u.s.w.

*Beispiel 19:*  $234_{10} = ?_8$

$$\begin{aligned} 234_{10}/8_{10} &= 29_{10} & \text{Rest } 2 \Rightarrow a_0 = 2 \\ 29_{10}/8_{10} &= 3_{10} & \text{Rest } 5 \Rightarrow a_1 = 5 \\ 3_{10}/8_{10} &= 0_{10} & \text{Rest } 3 \Rightarrow a_2 = 3 \\ && \Rightarrow 234_{10} = 352_8 \end{aligned}$$

*Beispiel 20:*  $234_{10} = ?_{16}$

$$\begin{aligned} 234_{10}/16_{10} &= 14_{10} & \text{Rest } 10 \Rightarrow a_0 = A \\ 14_{10}/16_{10} &= 0_{10} & \text{Rest } 14 \Rightarrow a_1 = E \\ && \Rightarrow 234_{10} = EA_{16} \end{aligned}$$

*Beispiel 21:*  $30_{10} = ?_2$

$$\begin{aligned} 30_{10}/2_{10} &= 15_{10} & \text{Rest } 0 \Rightarrow a_0 = 0 \\ 15_{10}/2_{10} &= 7_{10} & \text{Rest } 1 \Rightarrow a_1 = 1 \\ 7_{10}/2_{10} &= 3_{10} & \text{Rest } 1 \Rightarrow a_2 = 1 \\ 3_{10}/2_{10} &= 1_{10} & \text{Rest } 1 \Rightarrow a_3 = 1 \\ 1_{10}/2_{10} &= 0_{10} & \text{Rest } 1 \Rightarrow a_4 = 1 \\ && \Rightarrow 30_{10} = 1'1110_2 \end{aligned}$$

Um Zahlen mit gebrochenem Anteil umwandeln zu können, ist eine Fallunterscheidung zu treffen:

- ganzahliger Anteil
- gebrochener Anteil

### 2.4.4 Sukzessive Multiplikation (nach Horner-Schema)

Die Sukzessive Multiplikation mit Hilfe des Horner-Schemas erlaubt die Umwandlung von Nachkommastellen der Basis A nach Basis B.

**Rechnen in Basis A**

Der gebrochene Zahlenanteil lässt sich wie folgt mittels des Hornerschemas darstellen:

$$r^{-1}(a_{-1} + r^{-1}(a_{-2} + r^{-1}(a_{-3} + r^{-1}(\dots + r^{-1}a_{-m})))) \quad (2.8)$$

Durch Multiplikation mit r ergibt sich ein Anteil vor dem Komma (zB  $a_{-1}$ ) und ein weiterer gebrochener Anteil. Die fortgesetzte Multiplikation mit r erzeugt eine Folge von ganzzahligen Werten, die die gesuchten(n) Ziffern darstellen.

Die Umrechnung von **Zahlen mit Vor- und Nachkommastellen** führen wir auf die zwei oben genannten Algorithmen zurück, indem wir die umzurechnende Zahl durch ganzzahligen Anteil und Nachkommanteil darstellen und getrennt die Darstellung in der Ergebnisbasis berechnen.

*Beispiel 22:*  $46,375_{10} = ?_2$

$$\begin{aligned} 46_{10}/2_{10} &= 23_{10} & \text{Rest } 0 \Rightarrow a_0 = 0 \\ 23_{10}/2_{10} &= 11_{10} & \text{Rest } 1 \Rightarrow a_1 = 1 \\ 11_{10}/2_{10} &= 5_{10} & \text{Rest } 1 \Rightarrow a_2 = 1 \\ 5_{10}/2_{10} &= 2_{10} & \text{Rest } 1 \Rightarrow a_3 = 1 \\ 2_{10}/2_{10} &= 1_{10} & \text{Rest } 0 \Rightarrow a_4 = 0 \\ 1_{10}/2_{10} &= 0_{10} & \text{Rest } 1 \Rightarrow a_5 = 1 \\ &\Rightarrow 46_{10} = 10'1110_2 \\ 0,375_{10} * 2_{10} &= 0,75_{10} & \Rightarrow a_{-1} = 0 \\ 0,75_{10} * 2_{10} &= 1,50_{10} & \Rightarrow a_{-2} = 1 \\ 0,50_{10} * 2_{10} &= 1,00_{10} & \Rightarrow a_{-3} = 1 \\ && \Rightarrow 0,375_{10} = ,011_2 \\ && \Rightarrow 46,375_{10} = 101110,011_2 \end{aligned}$$

*Beispiel 23:*  $0,1285_{10} = ?_8$

$$\begin{aligned} 0,1285_{10} * 8_{10} &= 1,0280_{10} & \Rightarrow a_{-1} = 1 \\ 0,0280_{10} * 8_{10} &= 0,2240_{10} & \Rightarrow a_{-2} = 0 \\ 0,2240_{10} * 8_{10} &= 1,7920_{10} & \Rightarrow a_{-3} = 1 \\ 0,7920_{10} * 8_{10} &= 6,3360_{10} & \Rightarrow a_{-4} = 6 \\ 0,3360_{10} * 8_{10} &= 2,6880_{10} & \Rightarrow a_{-5} = 2 \\ 0,6880_{10} * 8_{10} &= 5,5040_{10} & \Rightarrow a_{-6} = 5 \\ &\Rightarrow 0,1285_{10} = ,101625\dots_8 \end{aligned}$$

*Beispiel 24:*  $0,2_{10} = ?_2$

$$\begin{aligned} 0,2_{10} * 2_{10} &= 0,4_{10} & \Rightarrow a_{-1} = 0 \\ 0,4_{10} * 2_{10} &= 0,8_{10} & \Rightarrow a_{-2} = 0 \\ 0,8_{10} * 2_{10} &= 1,6_{10} & \Rightarrow a_{-3} = 1 \\ 0,6_{10} * 2_{10} &= 1,2_{10} & \Rightarrow a_{-4} = 1 \\ 0,2_{10} * 2_{10} &= 0,4_{10} & \Rightarrow a_{-5} = 0 \\ 0,4_{10} * 2_{10} &= 0,8_{10} & \Rightarrow a_{-6} = 0 \\ 0,8_{10} * 2_{10} &= 1,6_{10} & \Rightarrow a_{-7} = 1 \\ 0,6_{10} * 2_{10} &= 1,2_{10} & \Rightarrow a_{-8} = 1 \\ &\Rightarrow 0,2_{10} = ,00110011\dots_2 \end{aligned}$$

*Beispiel 25:*  $0,5_{10} = ?_2$

$$\begin{aligned} 0,5_{10} * 2_{10} &= 1,0_{10} \Rightarrow a_{-1} = 1 \\ &\Rightarrow 0,5_{10} = 0,1_2 \end{aligned}$$

*Beispiel 26:*  $0,75_{10} = ?_2$

$$\begin{aligned} 0,75_{10} * 2_{10} &= 1,5_{10} \Rightarrow a_{-1} = 1 \\ 0,50_{10} * 2_{10} &= 1,0_{10} \Rightarrow a_{-2} = 1 \\ &\Rightarrow 0,75_{10} = 0,11_2 \end{aligned}$$

#### 2.4.5 Allgemeines Verfahren zur Konvertierung von Zahlen der Basis A nach B

**Schritt 1:** Aufstellen der Summenformel und Transformation nach  $r=10$  (Rechnen in Basis  $r=10$ )

**Schritt 2:** Sukzessive Divison/Multiplikation nach Horner-Schema (Rechnen in Basis  $r=10$ )

*Beispiel 27:*  $22_3 = ?_7$

$$\begin{aligned} \text{Schritt 1: } 2 * 3^1 + 2 * 3^0 &= 8_{10} \\ \text{Schritt 2: } 8_{10} / 7_{10} &= 1_{10} \quad \text{Rest}1 \Rightarrow a_0 = 1 \\ &1_{10} / 7_{10} = 0_{10} \quad \text{Rest}1 \Rightarrow a_1 = 1 \\ \text{Lösung: } 22_3 &= 11_7 \end{aligned}$$

**Sonderfälle:**

- $A = B^K$  ... zB:  $16 \rightarrow 2$
- $B = A^K$  ... zB:  $2 \rightarrow 16$

*Beispiel S1:*  $(1|011|011|, 101|011|1)_2 = ?_8$   
 $\Rightarrow 133,534_8$

*Beispiel S2:*  $(AF, 16C)_{16} = ?_2$   
 $\Rightarrow 1010\ 1111, 0001\ 0110\ 1100_2$

*Beispiel S3:*  $(AF, 16C)_{16} = ?_8$   
 $\Rightarrow 10|101|111|, 000|101|101|100_2$   
 $\Rightarrow 257,0554_8$

## 2.5 Rechnen in beliebigen Zahlensystemen

### 2.5.1 Rechnen im Dezimalsystem

#### Addition im Dezimalsystem

Tabelle 2.2 zeigt bekannte die Additionsmatrix für das Dezimalsystem.

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

Tabelle 2.2: Addition im Dezimalsystem

#### Multiplikation im Dezimalsystem

Tabelle 2.3 zeigt die Multiplikationsmatrix für das Dezimalsystem.

x	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	10	12	14	16	18
3	0	3	6	9	12	15	18	21	24	27
4	0	4	8	12	16	20	24	28	32	36
5	0	5	10	15	20	25	30	35	40	45
6	0	6	12	18	24	30	36	42	48	54
7	0	7	14	21	28	35	42	49	56	63
8	0	8	16	24	32	40	48	56	64	72
9	0	9	18	27	36	45	54	63	72	81

Tabelle 2.3: Multiplikation im Dezimalsystem

### 2.5.2 Rechnen im Dualsystem

#### Addition im Dualsystem

Tabelle 2.4 zeigt die Additionsmatrix für das Dualsystem.

+	0	1
0	0	1
1	1	10

Tabelle 2.4: Addition im Dualsystem

Beispiel 28: Addition 1:

$$1_2 + 1_2 + 1_2 = 10_2 + 1_2 = 11_2$$

Beispiel 29: Addition 2:

$$\begin{array}{r} 10_2 \\ + 01_2 \\ \hline 11_2 \end{array} \quad \begin{array}{r} 2 \\ + 1 \\ \hline 3 \end{array}$$

Beispiel 30: Addition 3:

$$\begin{array}{r} 10\ 1101_2 \\ + 11\ 0101_2 \\ \hline 110\ 0010_2 \end{array} \quad \begin{array}{r} 45 \\ + 53 \\ \hline 98 \end{array}$$

Beispiel 31: Addition 4:

$$\begin{array}{r} 10\ 1101_2 \\ + 11\ 0101_2 \\ + 00\ 1101_2 \\ + 01\ 0001_2 \\ \hline 1000\ 0000_2 \end{array} \quad \begin{array}{r} 45 \\ + 53 \\ + 13 \\ + 17 \\ \hline 128 \end{array}$$

#### Multiplikation im Dualsystem

Tabelle 2.5 zeigt die Multiplikationsmatrix für das Dualsystem.

x	0	1
0	0	0
1	0	1

Tabelle 2.5: Multiplikation im Dualsystem

**Verfahren A:** Abarbeitung des Multiplikators beginnend bei  $a_{n-1}$ .

Beispiel 32: Multiplikation im Dualsystem

$$23_{10} * 11_{10} = 253_{10}$$

$$10111_2 * 1011_2 = ?_2$$

$$\begin{array}{r} 23 * 11 \\ \hline 23 \\ 23 \\ \hline 253 \text{ dez} \end{array} \quad \begin{array}{r} 10111 * 1011 \\ \hline 10111 \\ 00000 \\ 10111 \\ \hline 11111101 \text{ bin} \end{array}$$

**Nachteil:** viele Zwischenergebnisse brauchen viel Speicherplatz (z.B. Register)  
komplexe Addition

**Ausweg:** Sofortige Summenbildung  
Bilden von Zwischenergebnissen

**Problem:** Bestimmung des Stellenwertes der ersten Ziffer

**Lösung:** Verfahren B

**Verfahren B:** Umgekehrte Abarbeitung der Stellen des Multiplikators (beginnend bei  $a_0$ ).

*Beispiel 33:* Multiplikation im Dualsystem nach Verfahren B

$$\begin{array}{r} 23_{10} * 11_{10} = 253_{10} \\ 10111_2 * 1011_2 = ?_2 \end{array}$$

$$\begin{array}{r} <-- \\ 10111 * 1011 \\ \hline 10111 \\ 10111 \\ 00000 \\ 10111 \\ \hline 11111101 \end{array}$$

Tabelle 2.6 zeigt Verfahren B unter Verwendung von nur 2 Registern (Multiplikation mittels Add-and-Shift):

Ergebnis Multiplikant	0 0000 0000 0 0001 0111	Multiplikator $d_0 = 1$	1011 $\leftarrow$
Ergebnis Multiplikant	0 0001 0111 0 0010 1110	$d_1 = 1$	Addieren + Schieben
Ergebnis Multiplikant	0 0100 0101 0 0101 1100	$d_2 = 0$	Addieren + Schieben
Ergebnis Multiplikant	0 0100 0101 0 1011 1000	$d_3 = 1$	Schieben
Ergebnis Multiplikant	0 1111 1101		Addieren

Tabelle 2.6: Verfahren zur Multiplikation im Dualsystem

### Subtraktion im Dualsystem

Regeln für die Subtraktion im Dualsystem:

$$1_2 - 0_2 = 1_2$$

$$1_2 - 1_2 = 0_2$$

$$0_2 - 0_2 = 0_2$$

$$0_2 - 1_2 = 1_2 \dots \text{mit 1 geborgt } (10_2 - 1_2 = 1_2)$$

Beispiel 34: Subtraktion 1:

$$\begin{array}{r} 100\ 1101_2 \\ - 1\ 0111_2 \\ \hline 011\ 0110_2 \end{array} \quad \begin{array}{r} 77 \\ - 23 \\ \hline 54 \end{array}$$

Beispiel 35: Subtraktion 2:

$$\begin{array}{r} 11\ 1000_2 \\ - 0\ 0001_2 \\ \hline 11\ 0111_2 \end{array} \quad \begin{array}{r} 56 \\ - 1 \\ \hline 55 \end{array}$$

### Division im Dualsystem

Beispiel 36: Division im Dualsystem

$$\begin{array}{r} 186 : 14 = 13 \\ \hline -14 \\ -- \\ 46 \\ -42 \\ -- \\ 4 \text{ Rest} \end{array} \quad \begin{array}{r} 10111010 : 1110 = 1101 \\ \hline -1110 \quad | \\ ----- \\ 10010 \quad | \\ -1110 \quad | \\ ----- \\ 010010 \\ -1110 \\ ----- \\ 100 \text{ Rest} \end{array}$$

### 2.5.3 Rechnen im Hexadezimalsystem

#### Addition im Hexadezimalsystem

Tabelle 2.7 zeigt die Additionsmatrix für das Hexadezimalsystem.

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	19	19	1A	1B	1C	1D	1E

Tabelle 2.7: Addition im Hexadezimalsystem

Beispiel 37: Addition im Hexadezimalsystem

$$\begin{array}{r}
 2A58_{16} \\
 + 71D0_{16} \\
 \hline
 9C28_{16}
 \end{array}
 \quad
 \begin{array}{r}
 10840_{10} \\
 + 29136_{10} \\
 \hline
 39976_{10}
 \end{array}$$

#### Multiplikation im Hexadezimalsystem

Tabelle 2.8 zeigt die Multiplikationsmatrix für das Hexadezimalsystem.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Tabelle 2.8: Multiplikation im Hexadezimalsystem

*Beispiel 38:* Multiplikation im Hexadezimalsystem

$$\begin{array}{r} 5C2A * 71D0 \\ \hline \end{array}$$

28526

5C2A

4AE220

-----

28F96C20

### Subtraktion im Hexadezimalsystem

*Beispiel 39:* Subtraktion im Hexadezimalsystem

$$\begin{array}{r} 9F1B_{16} & 40731_{10} \\ - 4A36_{16} & -18998_{10} \\ \hline 54E5_{16} & 21733_{10} \end{array}$$

### Division im Hexadezimalsystem

*Beispiel 40:* Division im Hexadezimalsystem

$$\begin{array}{r} 27FCA : 3E = A51 \\ \hline \end{array}$$

26C

---

13C

136

---

006A

3E

----

2C Rest

### 2.5.4 Rechnen im Oktalsystem

#### Addition im Oktalsystem

Tabelle 2.9 zeigt die Additionsmatrix für das Oktalsystem.

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Tabelle 2.9: Addition im Oktalsystem

#### Multiplikation im Oktalsystem

Tabelle 2.10 zeigt die Multiplikationsmatrix für das Oktalsystem.

x	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Tabelle 2.10: Multiplikation im Oktalsystem

## 2.6 Darstellung negativer Zahlen

### 2.6.1 Vorzeichen-Betrag-Darstellung (Signed-Magnitude)

Diese Darstellung zeichnet sich durch die **Trennung von Vorzeichen und Betrag** aus.

$$N = (s \ a_{n-1} \ a_{n-2} \dots \ a_1 \ a_0, \ a_{-1} \ a_{-2} \dots \ a_{-m})_r \quad (2.9)$$

$s$	Vorzeichen (Sign) +/ -
$a_i$	Ziffern
,	Komma (Radixpunkt), teilt gebrochenen Anteil vom ganzzahligen Anteil
$r$	Radix, Basiszahl
$a_{n-1}$	MSD most significant digit
$a_{-m}$	LSD least significant digit

wobei

$s = 0$ , wenn  $N \geq 0$

$s = r-1$ , wenn  $N \leq 0$  (beim Dualsystem: (Basis  $r=2$ )  $\Rightarrow$  Sign  $s=1$ )

$$+0_{10} = 0\ 000_{sm2}$$

$$-0_{10} = 1\ 000_{sm2}$$

$\Rightarrow$  Es gibt **zwei** Darstellungen für den Wert 0.

Abbildung 2.1 zeigt das Vorzeichen-Betrag-System in Kreisform.

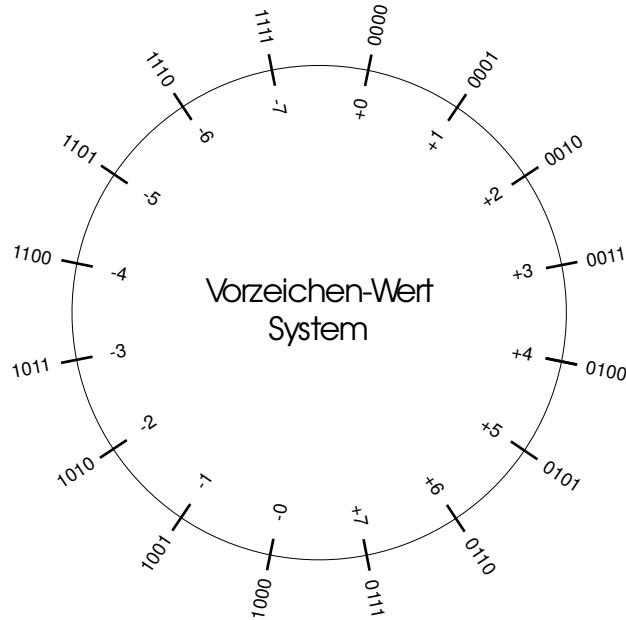


Abbildung 2.1: Vorzeichen-Betrag-System

*Beispiel 41:* Signed-Magnitude Darstellung im Dualsystem

$$+14_{10} = 01110_{sm2}$$

$$-14_{10} = -(1110)_2 = 11110_{sm2}$$

*Nachteile:*

- zwei Darstellungen für den Wert 0 (-0, +0)
- Rechenoperationen benötigen viele Fallunterscheidungen

## 2.6.2 Exzeß-Darstellungen (Excess-Codes)

*Prinzip:*

- Der kleinste darstellbare Zahlenwert wird einer Exzeß-darstellung wird durch 00...0 repräsentiert.
- Der Wert 0 wird um eine positive Zahl (Offset, Exzeß) verschoben.

*Vorteile:*

- Vorzeichenbit wird nicht benötigt
- Vereinfachung der Addition und Subtraktion gegenüber der Vorzeichen-Betrag-Darstellung (SM-Darstellung)

*Anwendung:*

- Exponenten in Gleitkommadarstellung

*Beispiel 42:* Excess-4 Code (Excess- $2^{n-1}$ )

$$n = 3$$

$$\text{Offset} = 2^{n-1} = 2^{3-1} = 4$$

$$\text{Bereich: } -2^{n-1}, \dots, 2^{n-1} - 1 = -4..3$$

Excess-4	
-4	000
-3	001
-2	010
-1	011
0	100 ← 4 = Offset (1 gefolgt von Nullen)
1	101
2	110
3	111

*Beispiel 43:* Excess-3 Code (Excess- $2^{n-1} - 1$ )

$$n = 3$$

$$\text{Offset} = 2^{n-1} - 1 = 2^{3-1} - 1 = 3$$

$$\text{Bereich: } -(2^{n-1} - 1), \dots, 2^{n-1} - 1 = -3..4$$

Excess-3	
-3	000
-2	001
-1	010
0	011 ← 3 = Offset (0 gefolgt von Einsen)
1	100
2	101
3	110
4	111

### Excess-Codes Übersicht

Die Tabelle 2.11 bietet eine Übersicht der gängigen Excess-Codes.

Excess-Code	n	Offset	0-Wert
Excess-3	3	$2^{3-1} - 1 = 3$	011
Excess-4	3	$2^{3-1} = 4$	100
Excess-127	8	$2^{8-1} - 1 = 127$	0111 1111
Excess-128	8	$2^{8-1} = 128$	1000 0000
Excess-1023	11	$2^{11-1} - 1 = 1023$	011 1111 1111
Excess-1024	11	$2^{11-1} = 1024$	100 0000 0000

Tabelle 2.11: Übersicht gängiger Excess-Codes

### 2.6.3 Komplement-Darstellung

#### Radix-Komplement (r-Komplement)

*Definition:*

Das Radix-Komplement  $\overline{N_r}$  einer Zahl  $N_r$  ist definiert durch

$$\overline{N_r} = r^n - N_r \quad (2.10)$$

wobei  $n$  die Anzahl der Stellen von  $N$  beschreibt.

Das Radix-Komplement  $\overline{N_r}$  wird zur Darstellung von  $-N_r$  verwendet.

*Beispiel 44:*

$$n = 3$$

$$r = 2$$

$$r^n = 2^3 = 8_{10} = 1000_2$$

	$N_r$	$r^n - N_r$	$\overline{N_r}$	$\overline{N_r}_{10}$
$0_{10}$	000	$8-0 = 8$	1000	$+8_{10}$
$1_{10}$	001	$8-1 = 7$	111	$-1_{10}$
$2_{10}$	010	$8-2 = 6$	110	$-2_{10}$
$3_{10}$	011	$8-3 = 5$	101	$-3_{10}$
$4_{10}$	100	$8-4 = 4$	100	$-4_{10}$

↑                              ↑

$\leftarrow$  Problem: 100 ist demnach  $+4_{10}$  und  $-4_{10}$  zugleich!

Wertebereich:

- Grösste positive Zahl:  $r^{n-1} - 1 = 2^2 - 1 = 3$
- Kleinste negative Zahl:  $-r^{n-1} = -2^2 = -4$

**Das Radix-Komplement für  $r=2$   
heißt Zweier-Komplement.**

*Beispiel 45:* Berechne das Radix-Komplement von  $5_{10}$  im Dualsystem

*Angaben zum Radix-Komplement:*

$$n = 4$$

$$r = 2$$

*Berechnung:*

$$\begin{aligned} N &= 5_{10} = 0101_2 \\ \overline{N_r} &= r^n - N = 2^4 - 0101_2 = 1011_{2K} \end{aligned}$$

*Zwischenrechnung:*

$$\begin{array}{r} 10000_2 \\ -0101_2 \\ \hline 1011_2 \end{array} = 11_{10}$$

*Kontrolle im Dezimalsystem:*

$$\overline{N_r} = 16_{10} - 5_{10} = 11_{10}$$

*Kontrolle im Dualsystem:*

$$\begin{array}{r} 0101 \\ +1011 \\ \hline 10000 \end{array} \quad \text{Übertrag wird nicht berücksichtigt, da } n=4.$$

**Vereinfachte Berechnung:**

1. Ersetze jede Ziffer  $a_i$  durch  $(r - 1) - a_i$
2. Addiere die Zahl 1

*Beispiel 46:* Vereinfachte Berechnung:

$$\begin{array}{r} 0101 \\ \downarrow \\ 1010 \\ +1 \\ \hline 1011 \end{array} \quad (\Rightarrow \text{2er-Komplement})$$

$\overline{N_r}$  kann verwendet werden, um  $-N_r$  darzustellen.  
 $N_r + \overline{N_r} = 0$

$$\begin{array}{r} +1 & 0001 \\ \downarrow \\ 1110 \\ +1 \\ \hline -1 & \overline{1111} & 2K \end{array}$$

*Beispiel 47:*

$$-8_{10} = 1000_{2K}$$

$$-1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 0 * 2^0 = -8$$

$$-1_{10} = 1111_{2K}$$

$$-1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = -8 + 4 + 2 + 1 = -1$$

$$+7_{10} = 0111_{2K}$$

$$-0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 4 + 2 + 1 = +7$$

Das erste Bit stellt den negativen Anteil dar,  
die restlichen Bits den positiven Anteil.

**r-1 Komplement**

Das r-1 - Komplement heißt für r=2 Einserkomplement.

*Definition:*  $\overline{N_{r-1}} = r^n - 1 - N_r$

*Einfache Berechnung:*

Ersetze jede Stelle  $a_i$  durch  $r - 1 - a_i$ , d.h. ersetze bei  $r = 2$  die Ziffer 0 durch 1 und die Ziffer 1 durch 0.

*Beispiel 48:*

$$0101 \rightarrow 1010_{1K}$$

### 2.6.4 Rechnen mit Einser-Komplement-Darstellung

siehe Abbildung 2.2.

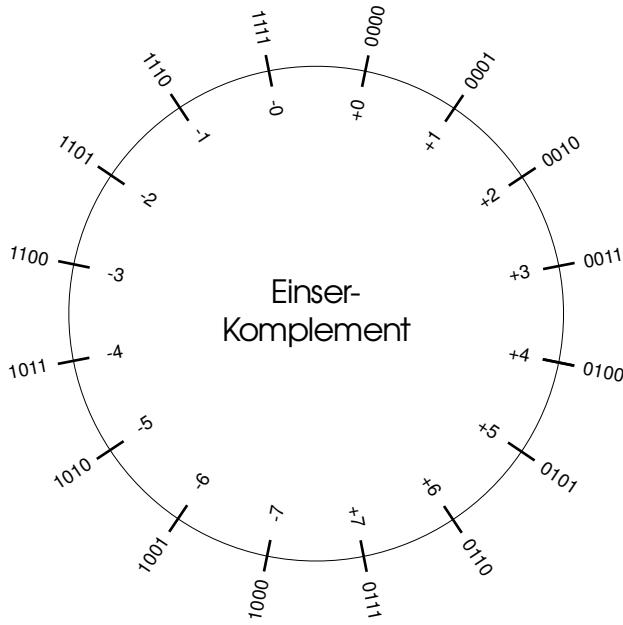


Abbildung 2.2: Einserkomplement

Beispiel 49:  $3-3 = 0$

$$\begin{array}{r} +3 \quad 011 \\ -3 \quad 100 \\ \hline 111 \end{array} \quad 0 ! \Rightarrow 2 \text{ Nullen}$$

Beispiel 50:  $9-4 = 5$

$$\begin{array}{r} +4 \quad 00100 \\ \downarrow \\ -4 \quad 11011 \\ \\ +9 \quad 01001 \\ -4 \quad 11011 \\ \hline 100100 \quad 4 ! \\ +1 \quad 1 \\ \hline 00101 \quad 5 \rightarrow \text{ok} \end{array}$$

Die Addition des Überlaufs ist notwendig, um korrekte Ergebnisse zu erhalten.

### 2.6.5 Rechnen mit Zweier-Komplement-Darstellung

siehe Abbildung 2.3

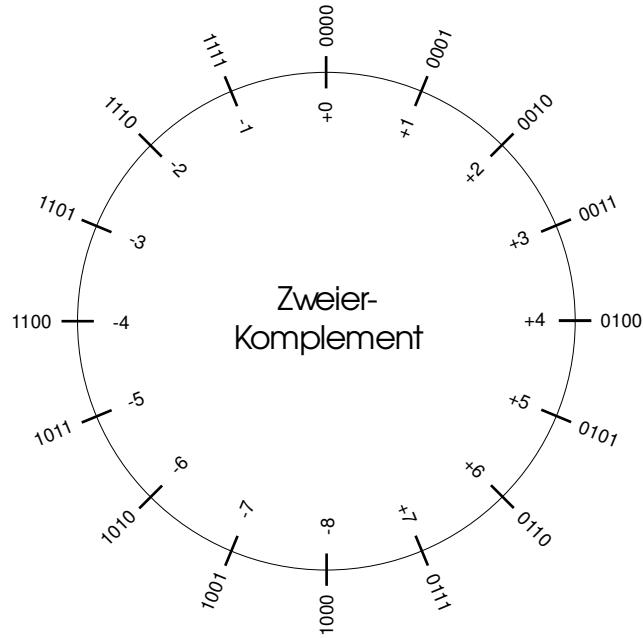


Abbildung 2.3: Zweierkomplement

Beispiel 51:  $3-3 = 0$

$$\begin{array}{r}
 +3 \quad 011 \\
 \downarrow \\
 100 \\
 +1 \quad \quad 1 \\
 -3 \quad \underline{101} \quad 2K
 \end{array}$$

$$\begin{array}{r}
 +3 \quad 011 \\
 -3 \quad \underline{101} \\
 \hline 1000 \quad 2K \Rightarrow \text{ok}
 \end{array}$$

Überlauf:  
Falls Übertrag in das VZ-Bit  $\neq$  Übertrag aus dem VZ-Bit

### 2.6.6 Vergleich der Darstellungsarten

Tabelle 2.12 zeigt eine Übersicht von Codes für negative Zahlen.

Decimal	Two's Complement	One's Complement	Signed-Magnitude	Excess $2^{m-1}$
-8	1000	—	—	0000
-7	1001	1000	1111	0001
-6	1010	1001	1110	0010
-5	1011	1010	1101	0011
-4	1100	1011	1100	0100
-3	1101	1100	1011	0101
-2	1110	1101	1010	0110
-1	1111	1110	1001	0111
0	0000	1111 or 0000	1000 or 0000	1000
1	0001	0001	0001	1001
2	0010	0010	0010	1010
3	0011	0011	0011	1011
4	0100	0100	0100	1100
5	0101	0101	0101	1101
6	0110	0110	0110	1110
7	0111	0111	0111	1111

Tabelle 2.12: Übersicht von Codes für negative Zahlen

## 2.7 Gleitkomma-Darstellung

### 2.7.1 Einführung in die Gleitkomma-Darstellung

#### Definition des Wertebereichs:

Der *Wertebereich (Range)* einer Zahlendarstellung beschreibt das Intervall der darstellbaren Zahlen von der kleinsten bis zur grössten Zahl.

Mit zB. 4 Ziffern (Stellen) lassen sich im Dezimalsystem beispielsweise folgende Wertebereiche darstellen:

- von 0000, bis 9999,
- von 000,0 bis 999,9
- von ,0000 bis ,9999

Diese Darstellungen werden als **Festkomma-Darstellungen** bezeichnet, da für alle Zahlen im Wertebereich die Position des Kommas fest ist.

**Nachteile:** In der ganzzahligen Darstellung von 0000 bis 9999 kann die Zahl 99000 nicht dargestellt werden, obwohl nur 2 Ziffern von Null verschieden sind.

**Ausweg:** Verschieben des Kommas (= *Floating Point Representation*).

*Beispiel 52:* Von 4 Ziffern beschreiben 2 Ziffern die Position des Kommas und 2 Ziffern dienen zur Darstellung der Zahl:

Zahl	Signifikante Ziffern	Position des Kommas
0,99	99	00
9,9	99	01
99,0	99	02
9500,0	95	04
99000	99	05
Problem:		
9501	95 !!!	04
0,0099	99	-02
-99,0	-99	02

Tabelle 2.13: Probleme der Gleitkomma-Darstellung

Zusätzlicher Aufwand für die Verwendung von *Vorzeichen* und *Zahl* für die *Position des Kommas*.

$$\begin{aligned}
 & 0,99 \dots 0,99 * 10^0 \\
 & 9,9 \dots 0,99 * 10^1 \\
 & 99 \dots 0,99 * 10^2 \\
 & 990 \dots 0,99 * 10^3 \\
 & 9900 \dots 0,99 * 10^4 \\
 & 99000 \dots 0,99 * 10^5 \\
 & 0,099 \dots 0,99 * 10^{-1} \\
 & 0,0099 \dots 0,99 * 10^{-2}
 \end{aligned}$$

Tabelle 2.14: Exponential-Darstellung

#### Vorteile:

- Darstellbare Zahlen sind:  $-0,99 * 10^{99} \dots -0,01 * 10^{-99}$ ,  $0, +0,01 * 10^{-99} \dots +0,99 * 10^{99}$ .
- Einfacher Vergleich
- Vereinfachte Normalisierung von Exponenten (vorzeichenlose Darstellung).

**Nachteil:**

- Der Wertebereich wird nicht mehr gleichmäßig abgedeckt:

$$\begin{array}{r}
 1 \dots 0,1 * 10^1 \\
 2 \dots 0,2 * 10^1 \\
 9 \dots 0,9 * 10^1 \\
 \hline
 10 \dots 0,1 * 10^2 \\
 20 \dots 0,2 * 10^2 \\
 90 \dots 0,9 * 10^2 \\
 \hline
 100 \dots 0,1 * 10^3 \\
 200 \dots 0,2 * 10^3
 \end{array}$$

**2.7.2 Allgemeine Darstellung einer Gleitkommazahl F**

$$F = (-1)^s * M * r^E$$

M	Mantisse (Betrag)
E	Exponent (inkl. Vorzeichen)
r	Radix, Basiszahl
s	Vorzeichenbit der Mantisse

1	e Stellen	n+m Stellen
s	Exponent E	Mantisse M

Das **Vorzeichen** des Exponenten wird durch Verwendung eines *Exzeß-Codes* dargestellt.

Die **Mantisse** ist *normalisiert*, d.h. das Komma befindet sich ganz links, die linke Ziffer ist ungleich Null!  
 $\Rightarrow$  eindeutige Darstellung einer Zahl. Zum Beispiel Normalisieren von  $0,01 * 10^0 \rightarrow 0,1 * 10^{-1}$ .

*Beispiel 53:*

Gegeben:  $N = 101101,101_2 (= 45,625_{10})$

Gefragt ist die Darstellung von N im Gleitkomma-Format mit  $e = 5$  und  $(n + m = 10)$ .

Der Exponent ist im Exzeß-16-Code darzustellen. (Exzeß- $2^{n-1}$ -Code).

Um wieviel Stellen wandert das Komma nach vor?

$$\begin{aligned}
 101101,101 &= \\
 &= 10110,1101 * 2^1 = (1 \text{ x verschoben}) \\
 &= 1011,01101 * 2^2 = (2 \text{ x verschoben}) \\
 &= 101,101101 * 2^3 = (3 \text{ x verschoben}) \\
 &= 10,1101101 * 2^4 = (4 \text{ x verschoben}) \\
 &= 1,01101101 * 2^5 = (5 \text{ x verschoben}) \\
 &= ,101101101 * 2^6 = (6 \text{ x verschoben}) \dots \text{Das Komma ist also 6 mal zu verschieben.}
 \end{aligned}$$

Der Exponent  $6_{10}$  im Exzeß-16-Code codiert 10110.

Dezimal	Exzeß-16
-16	0 0000
-15	0 0001
-1	0 1111
0	1 0000
5	1 0101
6	1 0110
15	1 1111

→ Gleitkomma-Format: | 0 | 1 0110 | 10 1101 1010 |

0	1 0110	10 1101 1010
Vorzeichen	Exponent	Mantisse
positive Mantisse	= $6_{10}$ in Exzeß-16-Code	

Beispiel 54: Verschieben um n Stellen (= Multiplikation mit  $r^n$ )

$$\begin{aligned} 101,11 &= 5,75_{10} \\ 101,11 &= 0,10111 * 2^3 = ? \end{aligned}$$

$$\begin{array}{r} 0,1011 \quad 0,5 \\ \quad 0,125 \\ \quad 0,0625 \\ \quad 0,03125 \\ \hline 0,71875_{10} \end{array}$$

$$\text{Kontrolle: } 0,71875_{10} * 2^3 = 5,75_{10}$$

### 2.7.3 Gleitkomma-Formate (Floating-Point Formate)

Gleitkomma-Formate unterscheiden sich durch:

- unterschiedliche Darstellung des Exponenten
- unterschiedliche Darstellung der Mantisse
- unterschiedliche Basis
- unterschiedliche Normalisierung

#### IEEE-Gleitkomma-Standard 754-1985: allgemein

VZ	Vorzeichenbehafteter Exponent E	Mantissenwert M (auch <i>Signifikant</i> )
----	------------------------------------	---

VZ ... Mantissen-Vorzeichen

$$E = 0, \quad M = 0 \quad \text{Darstellung der NULL}$$

$$E = 255, \quad M = 0 \quad \pm\infty$$

$$E = 255, \quad M \neq 0 \quad \text{NAN} (= \text{not a number})$$

→ Wertebereich nicht leicht einsichtig!

**Achtung:** 1 Bit (VZ) + 8 Bit (Exponent) + 24 Bit (Mantisse) = 33 Bit !!!

**Hidden Bit:** Durch Normalisieren ist das 1. Bit immer '1'. Dieses Bit wird aus Platzgründen nicht abgespeichert.

#### IEEE-Gleitkomma-Standard 754-1985: Einfache Genauigkeit (*single precision*):

0	1	8	9	31
VZ	Exzeß-127		normalisierter Mantissenwert M	

#### IEEE-Gleitkomma-Standard 754-1985: Doppelte Genauigkeit (*double precision*):

0	1	11	12	63
VZ	Exzeß-1023		normalisierter Mantissenwert M	

Tabelle 2.15 zeigt eine Gegenüberstellung von einfacher- und doppelter Genauigkeit im IEEE-Gleitkomma-Standard 754-1985.

	Single	Double
Word length	32 bits	64 bits
Fraction + hidden bit m	23 + 1 bits	52 + 1 bits
Exponent e	8 bits	11 bits
Bias	127	1023
Approximate range	$2^{128} \approx 3.8 * 10^{38}$	$2^{1024} \approx 9 * 10^{307}$
Smallest normalized number	$2^{-126} \approx 10^{-38}$	$2^{-1022} \approx 10^{-308}$
Approximate precision	$2^{-23} \approx 10^{-7}$	$2^{-52} \approx 10^{-15}$

Tabelle 2.15: IEEE Floating-Point Formate

### Beispiele für weitere Floating-Point Formate

	IBM/370	DEC/VAX	Cyber 70
Word length (double)	32 (64) bits	32 (64) bits	60 bits
Significand + (hidden bit)	24 (56) bits	23+1 (55+1) bits	48 bits
Exponent	7 bits	8 bits	11 bits
Bias	64	128	1024
Base	16	2	2
Range of M	$\frac{1}{16} \leq M < 1$	$\frac{1}{2} \leq M < 1$	$1 \leq M < 2$
Representation of M	Signed-magnitude	Signed-magnitude	One's complement
Approximate range	$16^{63} \approx 7 * 10^{75}$	$2^{127} \approx 1,9 * 10^{38}$	$2^{1023} \approx 1 * 10^{307}$
Approximate precision	$2^{-24} \approx 10^{-7}(10^{-17})$	$2^{-24} \approx 10^{-7}(10^{-17})$	$2^{-48} \approx 10^{-14}$

Tabelle 2.16: Floating-Point Formate von drei Maschinen

### 2.7.4 Gleitkomma Multiplikation

Zwei Zahlen  $F_1$  und  $F_2$  werden multipliziert indem (1) die Mantissen multipliziert und (2) die Exponenten addiert werden.

$$\begin{aligned}F_1 &= (-1)^{S_1} * M_1 * 2^{E_1} \\F_2 &= (-1)^{S_2} * M_2 * 2^{E_2}\end{aligned}$$

**Achtung:** Bei der Addition von  $E_1 + E_2$  (in der Exzeß-Darstellung) enthält sowohl  $E_{1(E16)}$  als auch  $E_{2(E16)}$  den Offset.

*Beispiel 55:* Multipliziere  $2^5$  mit  $2^3$

*Berechnung der Exponenten*

$$E_{1(E16)} = E_{1(10)} + 16 = 5 + 16 = 21_{(E16)}$$

$$E_{2(E16)} = E_{2(10)} + 16 = 3 + 16 = 19_{(E16)}$$

*Berechnung des Ergebnis-Exponenten  $E_3$*

$$E_3 = 21 + 19 = 40 !!!$$

$$E_3 = 5 + 3 + 16 = 24 \text{ oder}$$

$$E_3 = 21 + 19 - \mathbf{16} = 24$$

Werden 2 Zahlen in einer Exzeß-Darstellung addiert, so muss vom Ergebnis der Offset einmal abgezogen werden.

$$F_3 = F_1 * F_2$$

$$F_3 = F_1 * F_2 = (-1)^{S_3} * M_3 * 2^{E_3}$$

$$S_3 = (S_1) \text{ XOR } (S_2)$$

XOR	0	1
0	0	1
1	1	0

$$M_3 = M_1 * M_2$$

$$E_{3(E16)} = E_{1(E16)} + E_{2(E16)} - \text{Offset}$$

⇒ Normalisieren

**Sonderfälle:**

- Wenn  $E_3 \geq$  dem maximalen Exponenten ist, muss ein **Überlauf-Fehler** angezeigt werden.  
 $+\infty, -\infty$  wird erzeugt!
- Wenn  $E_3 \leq$  dem minimalen Exponenten ist, muss ein **Unterlauf-Fehler** angezeigt werden.  
0 wird erzeugt!

*Beispiel 56:* Multiplikation von  $3,5_{10}$  und  $5,25_{10}$  mit ( $n+m=8$ ,  $e=5$ , Exzeß-16-Code)

$$F_1 = 3,5_{10} = 11,1_2$$

$$F_2 = 5,25_{10} = 101,01_2$$

$$F_3 = 3,5_{10} * 5,25_{10} = 18,375_{10}$$

$$\begin{aligned} F_1 &= 0,11100000 * 2^{2_{10}} = \\ &= 0,11100000 * 2^{18_{10}-16_{10}} = \\ &= 0,11100000 * 2^{(10010)_{E16}} \\ &= |0|10010|11100000| \end{aligned}$$

$$\begin{aligned} F_2 &= 0,10101000 * 2^{3_{10}} = \\ &= 0,10101000 * 2^{19_{10}-16_{10}} = \\ &= 0,10101000 * 2^{(10011)_{E16}} \\ &= |0|10011|10101000| \end{aligned}$$

1. Schritt: Multiplikation der Mantissen

$$\begin{array}{r} 11100000 * 10101000 \\ \hline 111|0000|00 \quad (\text{eine Null anhängen}) \\ 1|1100|0000 \quad (\text{eine Null anhängen}) \\ | 111|0000|0000 \quad (\text{drei Nullen anhängen}) \\ \hline 1001|0011|0000|0000 \end{array}$$

$$1110'0000 * 1010'1000 = 1001'0011'0000'0000 \ (\Rightarrow 16 \text{ Stellen})$$

2. Schritt: Addition der Exponenten - bias/excess

$$\begin{array}{r} 10010 \\ +10011 \\ -10000 \\ \hline 10101 \end{array} \quad 10101_{E16} = 5_{10}$$

3. Schritt: Normalisieren (hier nicht nötig)

$$M_3 = 1001\ 0011$$

$$E_3 = 10101$$

$$S_3 = 0$$

Zwischenergebnis: | 0 | 10101 | 10010011 |

$$\begin{aligned} \text{Probe: } E_3 &= (10101)_{E16} = 21 - 16 = 5_{10} \\ F_3 &= 0,10010011 * 2^5 = 10010,011 = 18,375_{10} \end{aligned}$$

$$5_{10} = 21_{10} - 16_{10} = (10101)_{E16}$$

Ergebnis: | 0 | 10101 | 10010011 |

### 2.7.5 Gleitkomma Addition

**Voraussetzung:** Mantissen dürfen nur bei gleichem Exponenten (= gleicher Stelle des Kommas) addiert werden.

*Beispiel 57:*

$$x = 0,01_{10} = 0,1 * 10^{-1}$$

$$y = 1,00_{10} = 0,1 * 10^1$$

Ohne Berücksichtigung der Exponenten würde hier bei der Addition der Mantissen ein Fehler entstehen.

Aus dieser Überlegung ergeben sich folgende 4 Schritte:

1. Schritt: Exponenten-Abgleich
2. Schritt: Addition der ausgerichteten Mantissen
3. Schritt: Exponent des Ergebnisses ist Exponent des größeren Operanden
4. Schritt: Normalisieren der Mantisse und Anpassen des Exponenten

Verschieben der Mantisse der **kleineren Zahl** nach rechts bei gleichzeitiger Erhöhung des Exponenten.

*Beispiel 58:*  $1,0 + 0,01$

$$F_1 = 1_{10} = 0,1_{10} * 10^{+1}$$

$$F_2 = 0,01_{10} = 0,1_{10} * 10^{-1} = 0,001_{10} * 10^{+1}$$

Addition:

$$0,1_{10} * 10^{+1} + 0,001_{10} * 10^{+1} = 0,101_{10} * 10^{+1} = 1,01_{10}$$

**Mathematisch (für  $E_1 \geq E_2$ ):**

$$F_1 \pm F_2 = [(-1)^{S1} M_1 \pm (-1)^{S2} M_2 * r^{-(E_1 - E_2)}] * r^{E_1}$$

*Beispiel 59:*

$$13,25_{10} + 3,5_{10}$$

$m+n = 10$ ,  $e = 5$ , Exzeß-16-Code

Vom Dezimal ins Dualsystem mit Normalisierung

$$13,25_{10} = 1101,01_2 = 0,110101 * 2^4$$

$$3,5_{10} = 11,1_2 = 0,111 * 2^2$$

Darstellung der Exponenten im Exzeß-16-Code

$$4 + 16 = 20 \rightarrow 10100$$

$$2 + 16 = 18 \rightarrow 10010$$

Addition:

$$F_1 = |0|10100|1101010000|$$

$$F_2 = |0|10010|1110000000|$$

$F_2$  ist die kleinere Zahl (Exponent ist kleiner)

$$\begin{array}{r}
 F_2 = 0 \quad 10010 \quad 11 \ 1000 \ 0000 \\
 F'_2 = 0 \quad 10011 \quad 01 \ 1100 \ 0000 \\
 \hline
 F''_2 = 0 \quad 10100 \quad 00 \ 1110 \ 0000 \\
 F_1 = + 0 \quad 10100 \quad 11 \ 0101 \ 0000 \\
 \hline
 & & 1 \mid 00 \ 0011 \ 0000
 \end{array}$$

$$F_3 = (-1)^0 * 1,000011 * 2^4 = (-1)^0 * 0,1000011 * 2^5$$

Darstellung des Exponenten im Exzeß-16-Code für  $F_3$

$$5 + 16 = 21 \rightarrow 10101$$

Lösung:

$$F_3 : |0|10101|1000011000|$$

Probe:

$$10101_{E16} = 5_{10}$$

$$0,1000011 * 2^5 = 10000,11_2 = 16,75_{10}$$

## 2.8 Zeichen- und Zahlendarstellung mit Codes

### 2.8.1 ASCII

ASCII (= American Standard Code for Information Interchange)

$b_3 b_2 b_1 b_0$	$b_6 b_5 b_4$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p	
0001	SOH	DC1	!	1	A	Q	a	q	
0010	STX	DC2	”	2	B	R	b	r	
0011	ETX	DC3	#	3	C	S	c	s	
0100	EOT	DC4	\$	4	D	T	d	t	
0101	ENQ	NAK	%	5	E	U	e	u	
0110	ACK	SYN	&	6	F	V	f	v	
0111	BEL	ETB	‘	7	G	W	g	w	
1000	BS	CAN	(	8	H	X	h	x	
1001	HAT	EM	)	9	I	Y	i	y	
1010	FL	SUB	*	:	J	Z	j	z	
1011	VT	ESC	+	;	K	[	k	{	
1100	FF	FS	,	<	L	\	l		
1101	CR	GS	-	=	M	]	m	}	
1110	SOH	RS	.	>	N	^	n	~	
1111	SI	US	/	?	O	-	o	DEL	

Tabelle 2.17: ASCII

#### Abkürzungen für Steuerungscode (Control code abbreviations):

NUL: null, SOH: start of heading, STX: start of text, ETX: end of text, EOT: end of transmission, ENQ: enquiry, ACK: acknowledge, BEL: bell, BS: backspace, HAT: horizontal tab, FL: line feed, VT: vertical tab, FF: form feed, CR: carriage return, SO: shift out, SI: shift in, DLE: data link escape, DC1: device control 1, DC2: device control 2, DC3: device control 3, DC4: device control 4, NAK: negative acknowledgment, SYN: synchronize, ETB: end transmission block, CAN: cancel, EM: end of medium, SUB: substitute, ESC: escape, FS: file separator, GS: group separator, RS: record separator, US: unit separator, SP: space, DEL: delete or rubout.

### 2.8.2 BCD

BCD (= Binary Coded Decimals)

Dez	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Tabelle 2.18: BCD Code

### 2.8.3 7-4-2-1 Code

Dez	7-4-2-1
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	1000 ←
8	1001
9	1010

Tabelle 2.19: 7-4-2-1 Code

### 2.8.4 Übersicht von Zahlencodes

Gewicht	Dual 8421	BCDIC (8421) *	Exzeß-3	Aiken 2421	Gray	2-aus-5 (74210) *	Biquinär 50 43210
0	0000	1010	0011	0000	0000	11000	01 00001
1	0001	0001	0100	0001	0001	00011	01 00010
2	0010	0010	0101	0010	0011	00101	01 00100
3	0011	0011	0110	0011	0010	00110	01 01000
4	0100	0100	0111	0100	0110	01001	01 10000
5	0101	0101	1000	1011	0111	01010	10 00001
6	0110	0110	1001	1100	0101	01100	10 00010
7	0111	0111	1010	1101	0100	10001	10 00100
8	1000	1000	1011	1110	1100	10010	10 01000
9	1001	1001	1100	1111	1101	10100	10 10000

Tabelle 2.20: Übersicht von Zahlencodes

\* Beim BCDIC und 2-aus-5 fällt die Darstellung der Ziffer 0 aus der Gewichtung!

## 2.9 Fehlererkennende und Fehlerkorrigierende Codes

### 2.9.1 Erweiterung mit Paritätsbits

Hez	Dez	BCD	Gerade Parität Even Parity	Ungerade Parität Odd Parity
00	0	0000	0	1
01	1	0001	1	0
02	2	0010	1	0
03	3	0011	0	1
04	4	0100	1	0
05	5	0101	0	1
06	6	0110	0	1
07	7	0111	1	0
08	8	1000	1	0
09	9	1001	0	1

Tabelle 2.21: BCD Code mit Paritätsbit

### 2.9.2 Hamming-Code-Tabelle

*Eigenschaften:*

- $d_{min} = 3$
- $p_2 : i_3 \oplus i_2 \oplus i_1$
- $p_1 : i_3 \oplus i_2 \oplus i_0$
- $p_0 : i_3 \oplus i_1 \oplus i_0$

*Zur Kontrolle, ob und welche(s) Bit(s) gekippt ist/sind:*

- Vermischen von Informationsbits und Paritätsbits:  
 $(i_3 \ i_2 \ i_1 \ p_2 \ i_0 \ p_1 \ p_0) = (b_7 \ b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1)$
- Gewichten der Paritätsbits

$p_1 : 4$

$p_1 : 2$

$p_0 : 1$

Hex	Dez	Hamming-Code-1					$\Rightarrow$	zur Kontrolle										
		data			parity			$i_3 \ i_2 \ i_1 \ i_0$	$p_2 \ p_1 \ p_0$	$b_7 \ b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1$	$0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$	$0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1$	$0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1$	$0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0$	$0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0$	$0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1$	$0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1$	$0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0$
00	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0
01	1	0	0	0	1	0		0	1	1	0	0	0	0	0	0	0	1
02	2	0	0	1	0	1		0	1	0	1	0	0	0	0	0	1	0
03	3	0	0	1	1	0		1	1	0	0	1	0	0	0	0	1	1
04	4	0	1	0	0	0		1	1	0	0	0	0	0	0	0	1	0
05	5	0	1	0	1	0		1	0	1	0	0	0	0	0	0	1	1
06	6	0	1	1	0	0		0	1	1	1	0	0	0	0	0	1	1
07	7	0	1	1	1	0		0	0	0	0	0	0	0	0	0	1	0
08	8	1	0	0	0	0		1	1	1	0	0	0	0	0	0	0	0
09	9	1	0	0	1	0		1	0	0	0	0	0	0	0	0	0	1
0A	10	1	0	1	0	0		0	1	0	0	0	0	0	0	0	1	0
0B	11	1	0	1	1	0		0	0	1	0	0	0	0	0	0	1	0
0C	12	1	1	0	0	0		0	0	1	0	0	0	0	0	0	1	0
0D	13	1	1	0	1	0		0	1	0	0	0	0	0	0	0	1	1
0E	14	1	1	1	0	0		1	0	0	0	0	0	0	0	0	1	1
0F	15	1	1	1	1	0		1	1	1	0	0	0	0	0	0	1	1

Tabelle 2.22: Hamming-Code-1

# Kapitel 3

## Boolesche Algebra und logische Schaltungen

**Al'ge'bra:** [österr. ...gebra; arab.-roman.] die; -, ...ebren: 1. (ohne Plural) Lehre von den Beziehungen zwischen math. Größen und den Regeln, denen sie unterliegen. 2. = algebraische Struktur.

**al'ge'bra'isch:** die Algebra betreffend; -e Struktur: eine Menge von Elementen (Rechenobjekten) einschließlich der zwischen ihnen definierten Verknüpfungen.

### 3.1 Schaltalgebra (Boolesche Algebra)

*G. Boole:* (1815-1864)

*C. Shannon* entwickelt daraus 1938 die Schaltalgebra um eine systematische Behandlung von elektrischen Logikschaltungen zu ermöglichen. Die Schaltalgebra ist ein Spezialfall einer Booleschen Algebra.

#### 3.1.1 Definition

Eine Boolesche Algebra besteht aus einer Menge  $B$  und zwei zweistelligen Verknüpfungen  $+$  und  $\cdot$ , die folgende Gesetze erfüllen:

Gegeben:  $x, y, z \in B$

##### 1. Abgeschlossenheit von $B$ bezüglich $+$ und $\cdot$

heißt, dass kein mögliches Ergebnis außerhalb von  $B$  liegt.

Für alle  $x \in B$  und  $y \in B$  gilt:

$$(x + y) \in B$$

$$(x \cdot y) \in B$$

##### 2. Neutrale Elemente

$B$  hat ein neutrales Element bezüglich  $\cdot$  das mit 1 (Einselement) bezeichnet wird.

$B$  hat ein neutrales Element bezüglich  $+$  das mit 0 (Nullelement) bezeichnet wird.

Für alle  $x \in B$  gilt:

$$x + 0 = x$$

$$x \cdot 1 = x$$

##### 3. Kommutativität

Für alle  $x \in B$  und  $y \in B$  gilt:

$$x + y = y + x$$

$$x \cdot y = y \cdot x$$

#### 4. Distributivitat

Fur alle  $x \in B$ ,  $y \in B$  und  $z \in B$  gilt:

$$\begin{aligned}x \cdot (y + z) &= (x \cdot y) + (x \cdot z) \\x + (y \cdot z) &= (x + y) \cdot (x + z)\end{aligned}$$

#### 5. Komplementares Element

Fur alle  $x \in B$  existiert ein  $\bar{x}$ , sodass gilt:

$$\begin{aligned}x + \bar{x} &= 1 \\x \cdot \bar{x} &= 0\end{aligned}$$

#### 6. $B$ muss mindestens zwei Elemente besitzen

##### 3.1.2 Schaltalgebra

Die Schaltalgebra ist eine spezielle Boolesche Algebra mit  $B = \{0, 1\}$ .

ODER	+	0	1
	0	0	1
	1	1	1

UND	·	0	1
	0	0	0
	1	0	1

##### 3.1.3 Mengenlehre

Abbildung 3.1 zeigt Venn-Diagramme fur **Vereinigung** ( $a + b$ ), **Durchschnitt** ( $a \cdot b$ ) und **Komplement** ( $\bar{a}$ ).

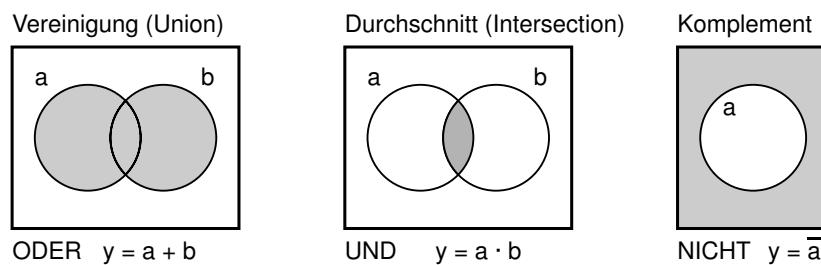


Abbildung 3.1: Venn-Diagramme: Vereinigung, Durchschnitt, Komplement

### 3.1.4 Beweis, dass die Schaltalgebra eine Boolesche Algebra ist

#### 1. Abgeschlossenheit:

Die Tabellen für ODER, UND enthalten nur 0 und 1.

#### 2. Neutrale Elemente:

$$+: \begin{array}{rcl} 1 + 0 & = & 1 \\ 0 + 0 & = & 0 \\ \uparrow x & \uparrow \text{neutrales Element} & \Rightarrow x + 0 = x \quad \text{für alle } x \end{array}$$

$$\cdot : \begin{array}{rcl} 1 \cdot 1 & = & 1 \\ 0 \cdot 1 & = & 0 \\ \uparrow x & \uparrow \text{neutrales Element} & \Rightarrow x \cdot 1 = x \quad \text{für alle } x \end{array}$$

#### 3. Kommutativität:

Ergibt sich aus der Symmetrie der Tabellen für + und ·.

#### 4. Distributivität $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ :

Beweis durch perfekte Induktion (acht Möglichkeiten, davon alle angeführt).

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

$\uparrow (=)$        $\uparrow (=)$

#### 5. Komplementäre Elemente:

0 und 1 sind gegeneinander komplementäre Elemente.

$$\begin{array}{ll} x + \bar{x} = 1 & 0 + \bar{0} = 0 + 1 = 1 \\ & 1 + \bar{1} = 1 + 0 = 1 \end{array}$$

$$\begin{array}{ll} x \cdot \bar{x} = 0 & 0 \cdot \bar{0} = 0 \cdot 1 = 0 \\ & 1 \cdot \bar{1} = 1 \cdot 0 = 0 \end{array}$$

#### 6. :

Die Schaltalgebra besitzt genau zwei unterschiedliche Werte: 0 und 1.

### 3.1.5 Rechenregeln

Tabelle 3.1 gibt eine Übersicht von weiteren Rechenregeln.

	+	.
Idempotenz	$x + x = x$	$x \cdot x = x$
Beschränktheit	$x + 1 = 1$	$x \cdot 0 = 0$
Absorption	$x + (x \cdot y) = x$	$x \cdot (x + y) = x$
(Involution)		$\overline{\overline{x}} = x$
Assoziativität	$(x + y) + z = x + (y + z)$	$(x \cdot y)z = x \cdot (y \cdot z)$
Consensus	$(x \cdot y) + (\overline{x} \cdot z) + (y \cdot z) = (x \cdot y) + (\overline{x} \cdot z)$	$(x + y) \cdot (\overline{x} + z) \cdot (y + z) = (x + y) \cdot (\overline{x} + z)$
DeMorgan	$\overline{(x + y)} = \overline{x} \cdot \overline{y}$	$\overline{(x \cdot y)} = \overline{x} + \overline{y}$

Tabelle 3.1: Rechenregeln

### 3.1.6 Weitere Boolesche Algebren

**Potenzmenge** über einer beliebigen Menge  $S$  ( $P(S), \cap, \cup$ ):

$$\begin{array}{lll} \{\} & \dots & 0\text{-Element} \\ S & \dots & 1\text{-Element} \end{array}$$

$\equiv$  Menge aller Teilmengen einer Menge  $S$ .

### 3.1.7 Eigenschaften von 0 und 1

Tabelle 3.2 verdeutlicht die Eigenschaften von 0 und 1.

+	.	$\overline{x}$
ODER	UND	NICHT
$x + 0 = x$	$x \cdot 0 = 0$	$\overline{0} = 1$
$x + 1 = 1$	$x \cdot 1 = x$	$\overline{1} = 0$

Tabelle 3.2: Eigenschaften von 0 und 1

## 3.2 Schaltausdruck (bzw. Schaltformen)

Jede Schaltkonstante (0, 1) oder Schaltvariable ist ein Schaltausdruck.

Sind  $T_1$  und  $T_2$  Schaltausdrücke, dann sind auch  $\overline{T_1}$ ,  $\overline{T_2}$ ,  $(T_1 + T_2)$  und  $(T_1 \cdot T_2)$  Schaltausdrücke.

Eine Schaltvariable oder deren Komplement wird auch als *Literal* bezeichnet.

### 3.3 Prinzip der Dualität

Zu jedem Schaltausdruck  $T$  existiert ein dualer Schaltausdruck  $T^D$ .

$T^D$  wird gebildet, in dem man in  $T$

- jedes + durch ·,
- jedes · durch +,
- jede 0 durch 1 und
- jede 1 durch 0 ersetzt.

Gilt  $T_1 = T_2$ , dann gilt auch  $T_1^D = T_2^D$ :

	$T_1 = T_2$	$T_1^D = T_2^D$
Beispiel 1:	$x + (x \cdot y) = x$	$x \cdot (x + y) = x$
Beispiel 2:	$x + 1 = 1$	$x \cdot 0 = 0$

### 3.4 Grundgatter und deren Darstellungen

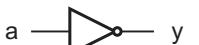
Funktion	log. Funktion	DIN 40900 Teil 12	ANSI/IEEE 91.1984
AND	$y = a \cdot b$	 a ————— & ————— y	 a ————— o ————— y
NAND	$y = \overline{a \cdot b}$	 a ————— & ————— y	 a ————— o ————— y
OR	$y = a + b$	 a ————— ≥1 ————— y	 a ————— o ————— y
NOR	$y = \overline{a + b}$	 a ————— ≥1 ————— y	 a ————— o ————— y
Buffer	$y = a$	 a ————— 1 ————— y	 a ————— △ ————— y
NOT	$y = \overline{a}$	 a ————— 1 ————— y	 a ————— o ————— y
XOR	$y = (\overline{a} \cdot b) + (a \cdot \overline{b}) = (a \oplus b)$	 a ————— -1 ————— y	 a ————— o ————— y
XNOR	$y = (\overline{a} \cdot \overline{b}) + (a \cdot b) = (\overline{a \oplus b})$	 a ————— -1 ————— y	 a ————— o ————— y

Abbildung 3.2: Logische Funktionen und Schaltsymbole der Grundgatter

Je nachdem welche Technologie man verwendet, kann der Aufbau der Grundgatter unterschiedlich sein. Daher einigt man sich auf abstrakte Schaltsymbole. Es gibt zwei wichtige Normen zur Darstellung der Grundgatter: die Norm DIN 40900 Teil 12 und die Norm ANSI/IEEE 91-1984. In Abbildung 3.2 werden die beiden Normen gegenübergestellt. Hier wird nur die DIN-Norm verwendet.

Abbildung 3.3 enthält die Wahrheitstabellen der Grundgatter.

AND	NAND	OR	NOR	Buffer	NOT	XOR	XNOR
a   b   y	a   b   y	a   b   y	a   b   y	a   y	a   y	a   b   y	a   b   y
0   0   0	0   0   1	0   0   0	0   0   1	0   0	0   1	0   0   0	0   0   1
0   1   0	0   1   1	0   1   1	0   1   0	1   1	1   0	0   1   1	0   1   0
1   0   0	1   0   1	1   0   1	1   0   0			1   0   1	1   0   0
1   1   1	1   1   0	1   1   1	1   1   0			1   1   0	1   1   1

Abbildung 3.3: Wahrheitstabellen der Grundgatter

### 3.4.1 Darstellung der Grundgatter mit NOR bzw. NAND

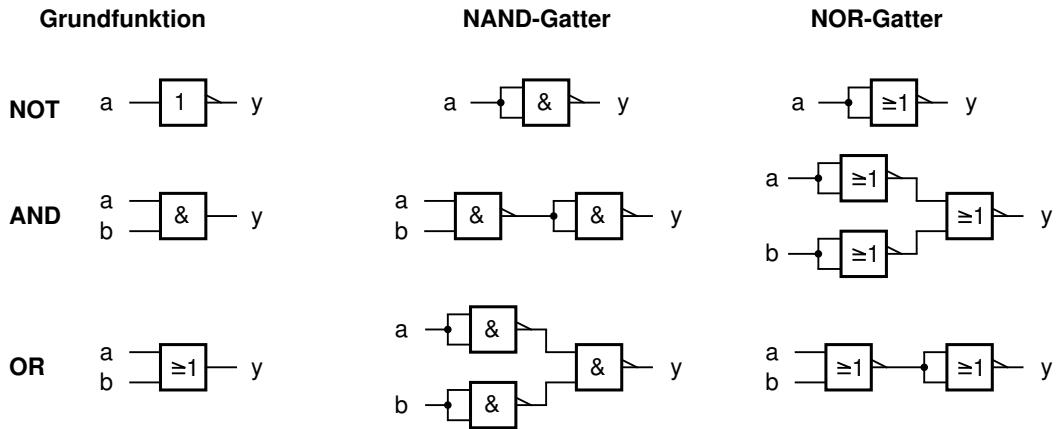


Abbildung 3.4: Realisierung der Grundgatter mittels NAND- und NOR-Gatter

Alle Grundgatter können mittels NOR- bzw. NAND-Gatter erzeugt werden. Momentan stellt sich die Frage wozu man alle Funktionen aus NAND bzw. NOR erzeugen soll. Dies ist in der CMOS-Technologie sehr wichtig, da diese beiden Funktionen sehr leicht zu produzieren sind. Abbildung 3.4 zeigt wie die jeweilige Grundfunktion (linke Spalte) mit NAND (mittlere Spalte) beziehungsweise NOR Gatter (rechte Spalte) realisiert werden kann.

### 3.5 Schaltfunktionen (Boolesche Funktionen)

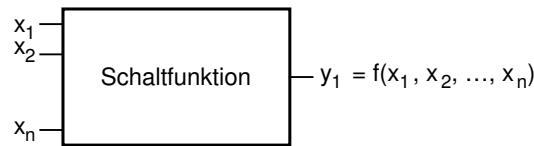


Abbildung 3.5: Schaltfunktion

Abbildung 3.5 stellt eine Schaltfunktion als BlackBox dar.

$$f : B^n \rightarrow B : (x_1, x_2, \dots, x_n) \mapsto f(x_1, x_2, \dots, x_n) := y$$

*Beispiel 1:*

$$f : B^3 \rightarrow B : (x_1, x_2, x_3) \mapsto f(x_1, x_2, x_3) := y$$

$$y := x_1 + \overline{x_2}x_3$$

$x_1$	$x_2$	$x_3$	$\overline{x_2}x_3$	$x_1 + \overline{x_2}x_3$
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	0	0	1
1	1	1	0	1

#### 3.5.1 Realisierung durch Gatterschaltungen

Abbildung 3.6 zeigt eine Schaltfunktion als Gatterschaltbild.

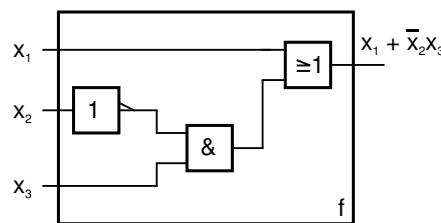


Abbildung 3.6: Gatterschaltbild einer Schaltfunktion

## 3.6 Schaltnetze

Die Verallgemeinerung einer Schaltfunktion heißt Schaltnetz.

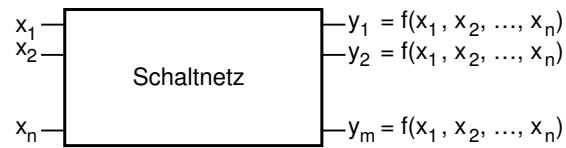


Abbildung 3.7: Schaltnetz

Abbildung 3.7 stellt ein Schaltnetz als Blackbox dar.

$$f : B^n \rightarrow B^m : (x_1, x_2, \dots, x_n) \mapsto f(x_1, x_2, \dots, x_n) := (y_1, \dots, y_m)$$

*Beispiel 2:*

$$\begin{aligned} f : B^3 &\rightarrow B^2 \\ f(x_1, x_2, x_3) &:= (y_1, y_2) = (x_1 + \overline{x_2}, \overline{x_1 x_3}) \end{aligned}$$

*Gatterschaltbild*

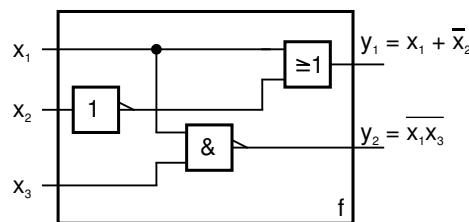


Abbildung 3.8: Gatterschaltbild eines Schaltnetzes

Abbildung 3.8 zeigt ein Schaltnetz als Gatterschaltbild.

*Wertetabelle*

$x_1$	$x_2$	$x_3$	$x_1 + \overline{x_2}$	$(x_1 x_3)$	$(\overline{x_1} \overline{x_3})$	$(y_1, y_2)$
0	0	0	1	0	1	(1, 1)
0	0	1	1	0	1	(1, 1)
0	1	0	0	0	1	(0, 1)
0	1	1	0	0	1	(0, 1)
1	0	0	1	0	1	(1, 1)
1	0	1	1	1	0	(1, 0)
1	1	0	1	0	1	(1, 1)
1	1	1	1	1	0	(1, 0)

Tabelle 3.3: Wertetabelle

Tabelle 3.3 zeigt die Wertetabelle des Schaltnetzes.

### 3.6.1 Realisierung von Schaltnetzen mit NAND/NOR-Gattern

$x$	$y$	AND	NAND	OR	NOR
0	0	0	1	0	1
0	0	0	1	1	0
0	1	0	1	1	0
0	1	1	0	1	0

Beispiel 3: Realisierung mit NAND-Gattern

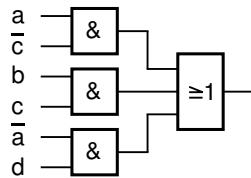
Gegeben ist folgende Schaltfunktion:  $f(a, b, c, d) = a\bar{c} + bc + \bar{a}d$

#### Lösung 1: Gatterschaltbild wie Schaltfunktion

Schaltfunktion

$$f(a, b, c, d) = a\bar{c} + bc + \bar{a}d$$

Gatterschaltbild



#### Lösung 2: Mit NAND2- und NAND3-Gattern

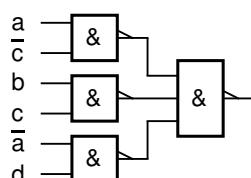
Schaltfunktion

$$f(a, b, c, d) = a\bar{c} + bc + \bar{a}d$$

$$= \overline{\overline{a\bar{c}} + bc + \overline{\bar{a}d}}$$

$$= \overline{\overline{a\bar{c}}} \cdot \overline{bc} \cdot \overline{\overline{\bar{a}d}}$$

Gatterschaltbild



**Lösung 3: Mit nur NAND2-Gattern**

Schaltfunktion

$$f(a, b, c, d) = a\bar{c} + bc + \bar{a}d$$

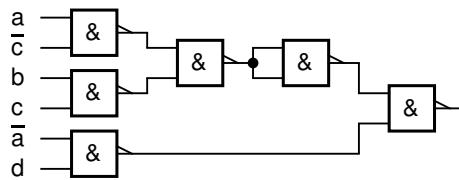
$$= \overline{\overline{a\bar{c}} + bc} + \bar{a}d$$

$$= \overline{a\bar{c} \cdot \overline{bc}} + \bar{a}d$$

$$= \overline{\overline{a\bar{c}} \cdot \overline{\overline{bc}}} + \bar{a}d$$

$$= \overline{\overline{a\bar{c}} \cdot \overline{bc} \cdot \overline{\bar{a}d}}$$

Gatterschaltbild



$$f(a, b, c, d) = a\bar{c} + bc + \bar{a}d = \overline{\overline{a\bar{c}} \cdot \overline{bc} \cdot \overline{\bar{a}d}}$$

Wertetabelle (Vergleich von Lösung 1 und Lösung 3)

abcd				y <sub>1</sub>					e	f	e · f	y <sub>3</sub> = e · f
	a	c	b		a	c	b	e				
0000	0	0	0	0	1	1	1	1	1	1	0	0
0001	0	0	1	1	1	1	1	0	0	0	1	1
0010	0	0	0	0	1	1	1	1	1	1	0	0
0011	0	0	1	1	1	1	1	0	0	0	1	1
0100	0	0	0	0	1	1	1	1	1	1	0	0
0101	0	0	1	1	1	1	1	0	0	0	1	1
0110	0	1	0	1	1	0	0	1	0	1	0	1
0111	0	1	1	1	1	0	0	0	0	0	1	1
1000	1	0	0	1	0	1	0	1	0	0	1	1
1001	1	0	0	1	0	1	0	1	0	0	1	1
1010	0	0	0	0	1	1	1	1	1	1	0	0
1011	0	0	0	0	1	1	1	1	1	1	0	0
1100	1	0	0	1	0	1	0	1	0	0	1	1
1101	1	0	0	1	0	1	0	1	0	1	0	1
1110	0	1	0	1	1	0	0	1	0	1	0	1
1111	0	1	0	1	1	0	0	1	0	1	0	1

## 3.7 Normalformen (Standard Forms)

### Begriffserklärung

*Literal:* Variable oder negierte Variable

*Disjunktion:* Summe

*Konjunktion:* Produkt

*Disjunktive Normalform:* Disjunktion von Konjunktionen  $\Rightarrow$  Summe von Produkten

*Konjunktive Normalform:* Konjunktion von Disjunktionen  $\Rightarrow$  Produkt von Summen

Eine *Konjunktion von Literalen* wird nur für höchstens eine Belegung mit Werten '1'.

$x_1$	$x_2$	$x_3$	$x_1\bar{x}_2x_3$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Eine *Disjunktion von Literalen* wird nur für höchstens eine Belegung mit Werten '0'.

$x_1$	$x_2$	$x_3$	$x_1 + \bar{x}_2 + x_3$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Sei  $f : B^n \rightarrow B$ , dann heißt eine Disjunktion/Konjunktion vollständig, wenn jedes  $x_i$  (wobei  $1 \leq i \leq n$ ) in negierter oder nicht negierter Form enthalten ist.

*Beispiel 4:*

$$f : B^3 \rightarrow B$$

- $x_1\bar{x}_2x_3$  ... vollständige Konjunktion
- $x_1x_3$
- $\bar{x}_1 + x_2 + x_3$  ... vollständige Disjunktion
- $\bar{x}_1$

### 3.7.1 Kanonische Normalformen (Canonical Forms)

**DKNF:** Disjunktive kanonische Normalform

Summe von vollständigen Produkten. zB  $x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3$

**KKNF:** Konjunktive kanonische Normalform

Produkt von vollständigen Summen. zB  $(\bar{x}_1 + x_2 + x_3) \cdot (x_1 + \bar{x}_2 + x_3)$

### 3.7.2 Erstellen einer DKNF (disjunktiv kanonische Normalform)

Beispiel 5: Erstellen einer DKNF aus der Wertetabelle,  $f : B^3 \rightarrow B$

$x_1$	$x_2$	$x_3$	$f(x_1, x_2, x_3)$		
0	0	0	1	←	$m_0$
0	0	1	1	←	$m_1$
0	1	0	0		
0	1	1	0		
1	0	0	1	←	$m_4$
1	0	1	0		
1	1	0	0		
1	1	1	1	←	$m_7$

$a \in \text{OnMenge}: f(a) = 1$

OnMenge:  $f^{-1}(1) = \{000, 001, 100, 111\}$

Jedes Element  $\mathbf{a}$  der OnMenge wird als Produkt codiert. Falls die Stelle  $i$  von  $a$  gleich '1' ist ( $a_i = 1$ ), wird  $a_i$  durch  $x_i$  ersetzt, falls  $a_i = 0$ , wird  $a_i$  durch  $\bar{x}_i$  ersetzt.

$$\begin{aligned} 000 &: \bar{x}_1 \bar{x}_2 \bar{x}_3 \\ 001 &: \bar{x}_1 \bar{x}_2 x_3 \\ 100 &: x_1 \bar{x}_2 \bar{x}_3 \\ 111 &: x_1 x_2 x_3 \end{aligned}$$

$$\text{DKNF}(f) = \bar{x}_1 \bar{x}_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 x_2 x_3$$

Die Produkte heißen **Minterme** von  $f$ .

Jeder Minterm ist nur für eine Belegung mit Werten gleich '1'.

Für jede Schaltfunktion  $f : B^n \rightarrow B : (x_1, \dots, x_n) \mapsto f(x_1, \dots, x_n)$  ist die zugehörige DKNF definiert durch:

$$\text{DKNF}(f) = \sum_{a \in \text{OnMenge}(f)} (y_1, \dots, y_n)_a \quad (3.1)$$

wobei gilt:

$$y_i := \begin{cases} x_i, & \text{wenn } a_i=1 \\ \bar{x}_i, & \text{wenn } a_i=0 \end{cases}$$

**Abgekürzte Schreibweise:**

$$\text{DKNF}(f) = m_0 + m_1 + m_4 + m_7$$

Beispiel 6: Bringe  $f(x, y, z) = x + yz$  in die DKNF

Möglichkeit 1: algebraisch

$$\begin{aligned} x + yz &= x(y + \bar{y})(z + \bar{z}) + (x + \bar{x})yz \\ &= (xy + x\bar{y})(z + \bar{z}) + (x\bar{y}z + \bar{x}yz) \\ &= (xyz + xy\bar{z} + x\bar{y}z + x\bar{y}\bar{z}) + (xyz + \bar{x}yz) \end{aligned}$$

$$\Rightarrow OnMenge = \{111, 110, 101, 100, 011\}$$

$$DKNF(f) = m_3 + m_4 + m_5 + m_6 + m_7$$

Möglichkeit 2: mit Wertetabelle

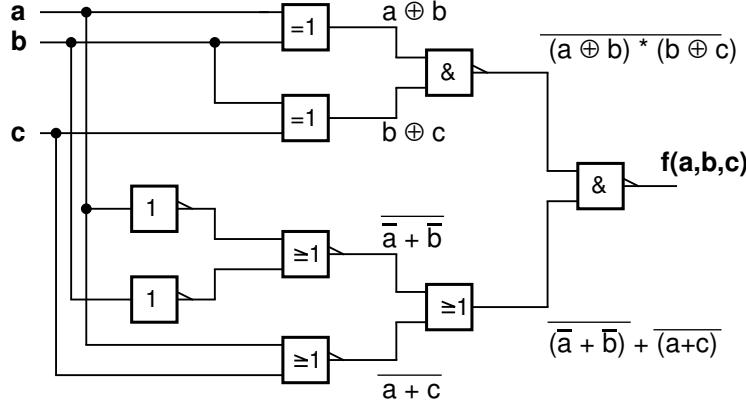
$xyz$	$yz$	$x + yz$
000	0	0
001	0	0
010	0	0
011	1	1 $\leftarrow m_3$
100	0	1 $\leftarrow m_4$
101	0	1 $\leftarrow m_5$
110	0	1 $\leftarrow m_6$
111	1	1 $\leftarrow m_7$

$$\Rightarrow OnMenge = \{011, 100, 101, 110, 111\}$$

$$DKNF(f) = \bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z + xy\bar{z} + xyz$$

Beispiel 7: Ermittle die DKNF für folgende Schaltung

Gatterschaltbild



Wertetabelle

$a \ b \ c$	1	2	$3 = \overline{1 \cdot 2}$	$\bar{a}\bar{b}$	4	5	$6 = 4 + 5$	$f = \overline{3 \cdot 6}$
0 0 0	0	0	1	1 1	0	1	1	0
0 0 1	0	1	1	1 1	0	0	0	1 $\leftarrow m_1$
0 1 0	1	1	0	1 0	0	1	1	1 $\leftarrow m_2$
0 1 1	1	0	1	1 0	0	0	0	1 $\leftarrow m_3$
1 0 0	1	0	1	0 1	0	0	0	1 $\leftarrow m_4$
1 0 1	1	1	0	0 1	0	0	0	1 $\leftarrow m_5$
1 1 0	0	1	1	0 0	1	0	1	0
1 1 1	0	0	1	0 0	1	0	1	0

$$\Rightarrow OnMenge = \{001, 010, 011, 100, 101\}$$

$$DKNF(f) = m_1 + m_2 + m_3 + m_4 + m_5$$

$$DKNF(f) = \bar{a}\bar{b}c + \bar{a}b\bar{c} + \bar{a}bc + a\bar{b}\bar{c} + ab\bar{c}$$

### 3.7.3 Erstellen einer KKNF (konjunktiv kanonische Normalform)

Beispiel 8: Erstellen einer KKNF aus der Wertetabelle,  $f : B^3 \rightarrow B$

$x_1$	$x_2$	$x_3$	$f(x_1, x_2, x_3)$	
0	0	0	1	
0	0	1	1	
0	1	0	0	← $M_2$
0	1	1	0	← $M_3$
1	0	0	1	
1	0	1	0	← $M_5$
1	1	0	0	← $M_6$
1	1	1	1	

$$a \in \text{OffMenge}: f(a) = 0$$

$$\text{OffMenge}: f^{-1}(0) = \{010, 011, 101, 110\}$$

Jedes Element  $\mathbf{a}$  der OffMenge wird als Summe codiert. Falls die Stelle  $i$  von  $a$  gleich '1' ist ( $a_i = 1$ ), wird  $a_i$  durch ' $\bar{x}_i$ ' ersetzt, falls  $a_i = 0$ , wird  $a_i$  durch ' $x_i$ ' ersetzt.

- 010 ( $M_2$ ):  $x_1 + \bar{x}_2 + x_3$
- 011 ( $M_3$ ):  $x_1 + \bar{x}_2 + \bar{x}_3$
- 101 ( $M_5$ ):  $\bar{x}_1 + x_2 + \bar{x}_3$
- 110 ( $M_6$ ):  $\bar{x}_1 + \bar{x}_2 + x_3$

$$\text{KKNF}(f) = (x_1 + \bar{x}_2 + x_3) \cdot (x_1 + \bar{x}_2 + \bar{x}_3) \cdot (\bar{x}_1 + x_2 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_2 + x_3)$$

Die Summen heißen **Maxterme** von  $f$ .

Jeder Maxterm ist nur für eine Belegung mit Werten gleich '0'.

Für jede Schaltfunktion  $f : B^n \rightarrow B : (x_1, \dots, x_n) \mapsto f(x_1, \dots, x_n)$  ist die zugehörige KKNF definiert durch:

$$\text{KKNF}(f) = \prod_{a \in \text{OffMenge}(f)} (y_1, \dots, y_n)_a \quad (3.2)$$

wobei gilt:

$$y_i := \begin{cases} \bar{x}_i, & \text{wenn } a_i=1 \\ x_i, & \text{wenn } a_i=0 \end{cases}$$

**Abgekürzte Schreibweise:**

$$\text{KKNF}(f) = M_2 \cdot M_3 \cdot M_5 \cdot M_6$$

Beispiel 9: Bringe  $f(x, y, z) = \bar{x}\bar{y} + xz$  in die KKNF

Möglichkeit 1: algebraisch

$$\begin{aligned}\bar{x}\bar{y} + xz &= (\bar{x}\bar{y} + x) \cdot (\bar{x}\bar{y} + z) = \\ &= (\bar{x} + x) \cdot (\bar{y} + x) \cdot (\bar{x} + z) \cdot (\bar{y} + z) = \\ &= 1 \cdot \underbrace{(\bar{x} + y)}_{(1)} \cdot \underbrace{(\bar{x} + z)}_{(2)} \cdot \underbrace{(\bar{y} + z)}_{(3)}\end{aligned}$$

Jeder Summe fehlt eine Variable  $\Rightarrow$  hinzufügen von  $(x\bar{x})$ ,  $(y\bar{y})$  bzw.  $(z\bar{z})$ .

$$\begin{aligned}(1) : (x + \bar{y}) &= (x + \bar{y}) + (z\bar{z}) = (x + \bar{y} + z) \cdot (x + \bar{y} + \bar{z}) \\ (2) : (\bar{x} + z) &= (\bar{x} + z) + (y\bar{y}) = (\bar{x} + z + y) \cdot (\bar{x} + z + \bar{y}) \\ (3) : (\bar{y} + z) &= (\bar{y} + z) + (x\bar{x}) = (\bar{y} + z + x) \cdot (\bar{y} + z + \bar{x})\end{aligned}$$

Nach streichen von doppelten Summen erhält man ...

$$\text{KKNF}(f) = (x + \bar{y} + z) \cdot (x + \bar{y} + \bar{z}) \cdot (\bar{x} + y + z) \cdot (\bar{x} + \bar{y} + z)$$

$$\text{KKNF}(f) = M_2 \cdot M_3 \cdot M_4 \cdot M_6$$

Möglichkeit 2: mit Wertetabelle

x	y	z	$\bar{x}\bar{y}$	$xz$	$\bar{x}\bar{y} + xz$
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	0	0	0
1	1	1	0	1	1

$$\Rightarrow \text{OffMenge} = \{010, 011, 100, 110\}$$

$$\text{KKNF}(f) = (x + \bar{y} + z) \cdot (x + \bar{y} + \bar{z}) \cdot (\bar{x} + y + z) \cdot (\bar{x} + \bar{y} + z)$$

$$\text{KKNF}(f) = M_2 \cdot M_3 \cdot M_4 \cdot M_6$$

### 3.7.4 Komplementierung einer kanonischen Normalform

Gegeben:

$$f(x, y, z) = x + yz$$

Das Komplement  $\bar{f}$  lautet:

$$\begin{aligned}\bar{f}(x, y, z) &= \overline{x + yz} = \\ &= \overline{x} \cdot \overline{y} \overline{z} = \\ &= \overline{x} \cdot (\overline{y} + \overline{z}) = \\ &= (\overline{x} \cdot \overline{y}) + (\overline{x} \cdot \overline{z})\end{aligned}$$

$xyz$	$yz$	$f(x, y, z)$	$\bar{x}$	$\bar{y}$	$\bar{z}$	$\bar{x}y$	$\bar{x}z$	$\bar{f}(x, y, z)$
		$x + yz$						$\bar{x}\bar{y} + \bar{x}\bar{z}$
000	0	0	1	1	1	1	1	1
001	0	0	1	1	0	1	0	1
010	0	0	1	0	1	0	1	1
011	1	1	1	0	0	0	0	0
100	0	1	0	1	1	0	0	0
101	0	1	0	1	0	0	0	0
110	0	1	0	0	1	0	0	0
111	1	1	0	0	0	0	0	0

Also:

$$f(x, y, z) = x + yz$$

$$\text{DKNF}(f) = m_3 + m_4 + m_5 + m_6 + m_7$$

$$\text{KKNF}(f) = M_0 \cdot M_1 \cdot M_2$$

$$\bar{f}(x, y, z) = \bar{x}\bar{y} + \bar{x}\bar{z}$$

$$\text{DKNF}(\bar{f}) = m_0 + m_1 + m_2$$

$$\text{KKNF}(\bar{f}) = M_3 \cdot M_4 \cdot M_5 \cdot M_6 \cdot M_7$$

### 3.8 Beispiele

*Beispiel 10:* Überprüfe mit Hilfe von Wertetabellen folgende Aussage:

$$\overline{x+y} = \overline{x} \cdot \overline{y} \text{ (DeMorgan)}$$

x	y	$\overline{x}\overline{y}$	$x+y$	$\overline{x+y}$	=?=?	$\overline{x} \cdot \overline{y}$
0	0	1	1	0		1
0	1	0	1	0		0
1	0	0	1	0		0
1	1	0	0	1		0

Ergebnis: Aussage ist **richtig!**

*Beispiel 11:* Überprüfe mit Hilfe von Wertetabellen folgende Aussage:

$$\overline{x} \cdot \overline{y} + x \cdot y = \overline{(x \cdot \overline{y} + \overline{x} \cdot y)}$$

x	y	$\overline{x} \cdot \overline{y}$	$x \cdot y$	$\overline{x} \cdot \overline{y} + x \cdot y$	=?=?	$x \cdot \overline{y}$	$\overline{x} \cdot y$	$(x \cdot \overline{y} + \overline{x} \cdot y)$	$\overline{(x \cdot \overline{y} + \overline{x} \cdot y)}$
0	0	1	0	1		0	0	0	1
0	1	0	0	0		0	1	1	0
1	0	0	0	0		1	0	1	0
1	1	0	1	1		0	0	0	1

Ergebnis: Aussage ist **richtig!**

*Beispiel 12:* Überprüfe mit Hilfe von Wertetabellen folgende Aussage:

$$x \cdot y + \overline{x} \cdot z + y \cdot z = x \cdot y + \overline{x} \cdot z \text{ (Consensus)}$$

x	y	z	$x \cdot y$	$\overline{x} \cdot z$	$y \cdot z$	$x \cdot y + \overline{x} \cdot z + y \cdot z$	=?=?	$x \cdot y$	$\overline{x} \cdot z$	$x \cdot y + \overline{x} \cdot z$
0	0	0	0	0	0	0		0	0	0
0	0	1	0	1	0	0		0	1	1
0	1	0	0	0	0	0		0	0	0
0	1	1	0	1	1	1		0	1	1
1	0	0	0	0	0	0		0	0	0
1	0	1	0	0	0	0		0	0	0
1	1	0	1	0	0	1		1	0	1
1	1	1	1	0	1	1		1	0	1

Ergebnis: Aussage ist **richtig!**

*Beispiel 13:* Überprüfe mit Hilfe von Wertetabellen folgende Aussage:

$$\overline{\bar{a} + \bar{b}} \cdot \overline{c \cdot d} = a \cdot b \cdot \bar{c} + a \cdot b \cdot d$$

a b c d	$\bar{a} + \bar{b}$	$\overline{\bar{a} + \bar{b}}$	$\overline{c \cdot d}$	$\overline{\bar{a} + \bar{b}} \cdot \overline{c \cdot d}$
0 0 0 0	1	0	1	0
0 0 0 1	1	0	1	0
0 0 1 0	1	0	1	0
0 0 1 1	1	0	0	0
0 1 0 0	1	0	1	0
0 1 0 1	1	0	1	0
0 1 1 0	1	0	1	0
0 1 1 1	1	0	0	0
1 0 0 0	1	0	1	0
1 0 0 1	1	0	1	0
1 0 1 0	1	0	1	0
1 0 1 1	1	0	0	0
1 1 0 0	0	1	1	1
1 1 0 1	0	1	1	1
1 1 1 0	0	1	1	1
1 1 1 1	0	1	0	0

=? =

$a \cdot b \cdot \bar{c}$	$a \cdot b \cdot d$	$a \cdot b \cdot \bar{c} + a \cdot b \cdot d$
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
1	0	1
1	1	1
0	0	0
0	1	1

Ergebnis: Aussage ist **falsch!**

### 3.9 Vereinfachung von Schaltfunktionen

**Literal:** Schaltvariable bzw. negierte (komplementierte) Schaltvariable

**Kanonische Formen:** eindeutige Darstellung, aber viele Operatoren und Literale

**Normalform:** nicht eindeutige Darstellung, aber weniger Operatoren

**Nicht normalisierte Ausdrücke:** weniger Operatoren und Literale durch gemeinsame Nutzung  
Beispiel:  $(a + b)cd = acd + bcd$

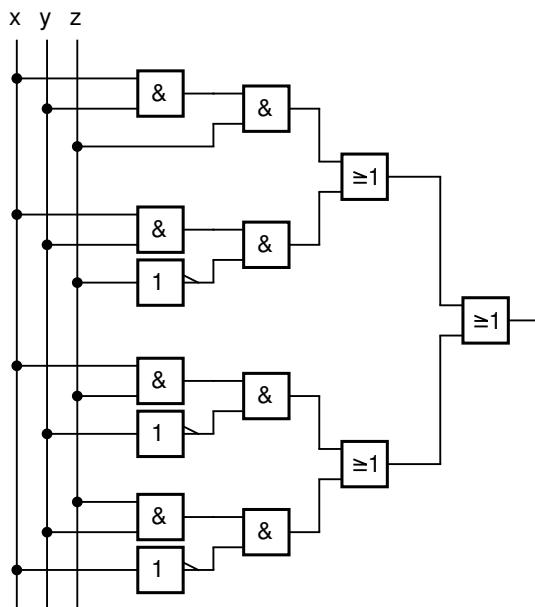
**Verzögerungszeit:** Die Verzögerungszeit bei Gatterrealisierungen wird bestimmt durch den *kritischen Pfad* (= Maximum der Verzögerungszeiten aller Wege durch die Schaltung).

*Beispiel 14:*

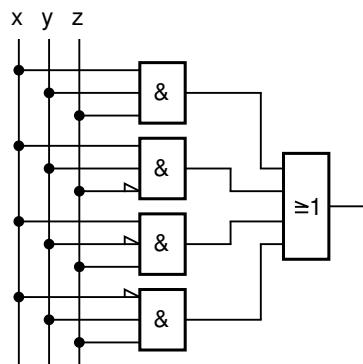
- Gegeben sei folgende Funktion:  $f(x, y, z) = xyz + xy\bar{z} + x\bar{y}z + \bar{x}yz$  (DKNF)

Realisierung:  $\Rightarrow 8 \text{ AND}, 3 \text{ OR}$

Realisierung mit mehrstufiger Logik (8 AND, 3 OR):



Realisierung mit zweistufiger Logik:



**2.** Zusammenfassen nach der Regel:  $ab + a\bar{b} = a(b + \bar{b}) = a \cdot 1 = a$

$$\begin{aligned} f(x, y, z) &= \mathbf{xyz} + xy\bar{z} + \mathbf{xyz} + x\bar{y}z + xyz + \bar{x}yz = \\ &= xy(z + \bar{z}) + xz(y + \bar{y}) + yz(x + \bar{x}) = \\ &= xy + xz + yz \end{aligned}$$

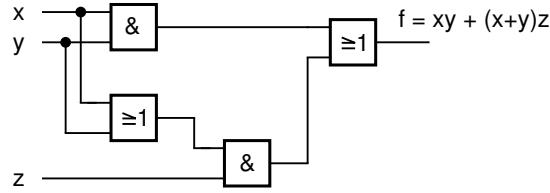
Realisierung:  $\Rightarrow 3 \text{ AND}, 2 \text{ OR}$

**3.** Weitere mögliche Zusammenfassung:

$$f(x, y, z) = xy + (x + y)z$$

Realisierung:  $\Rightarrow 2 \text{ AND}, 2 \text{ OR}$

Realisierung mit Gattern:



#### Vorteil der Normalformen:

Direkt umsetzbar in zweistufige Logik.

Günstige für die Verwendung von PLA (=Programmable Logic Array) → schnell.

### 3.9.1 Minimierung mittels KV-Diagrammen

#### Karnaugh & Veith Verfahren:

Graphisches Verfahren zur Vereinfachung von Schaltausdrücken bis zu vier (sechs) Variablen.

*Beispiel 15:* KV-Diagramme für AND, OR und XOR

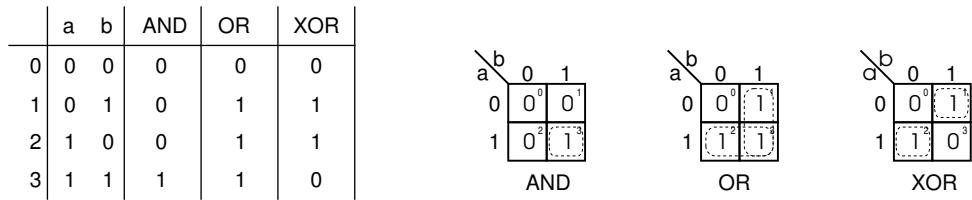


Abbildung 3.9: KV-Diagramme für AND, OR und XOR

*Beispiel 16:* KV-Diagramme für  $f(a,b,c)$

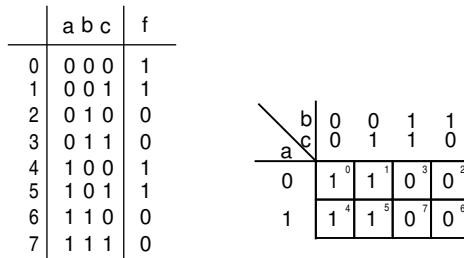
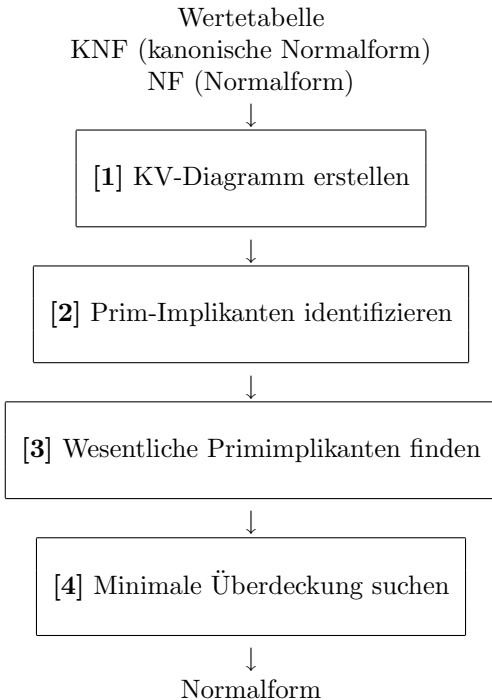


Abbildung 3.10: KV-Diagramme für  $f(a,b,c)$

#### Vorgehensweise



[1]: Zum Beispiel durch Ablesen aus Wertetabelle.

[2]: Gruppe aus 2, 4, 8, 16, ... - Mintermen, die in keiner anderen Gruppe enthalten ist  
 → *grösste Gruppe für jeden Minterm*.

[3]: Primimplikant, der mindestens einen Minterm enthält, der in sonst keinem Primimplikanten enthalten ist.

[4]: Abdeckung **aller** Minterme durch:

1. wesentliche Primimplikanten,
2. ausgewählte Primimplikanten, sodass die Gesamtanzahl der Literale minimal ist.

- Gegeben sei  $f : B^n \rightarrow B : (x_1, x_2, \dots, x_n) \rightarrow f((x_1, x_2, \dots, x_n))$
- Eine Konjunktion  $g$  einer Teilmenge der negierten oder nicht negierten Variablen  $x_i$  heißt **Implikant** von  $f$  wenn gilt:
 
$$\bigwedge_{a \in B^n} g(a) = 1 \Rightarrow f(a) = 1$$
- $g$  heißt **Primimplikant** von  $f$  wenn gilt:
  - $g$  ist Implikant von  $f$  und
  - Durch Entfernen eines Literals aus  $g$  entsteht kein neuer Implikant ( $g$  kann nicht verkürzt werden).
- $g$  heißt **wesentlicher Primimplikant** von  $f$  wenn ein Minterm  $m$  von  $f$  existiert, sodass gilt:
  - Durch Verkürzen von  $m$  kann nur der Primimplikant  $g$  (aber kein anderer Primimplikant von  $f$ ) erzeugt werden.
  - Der Minterm  $m$  wird ausschließlich von  $g$  abgedeckt.

## 1. Erstellen von KV-Diagrammen

*Prinzip:* Je zwei benachbarte Felder unterscheiden sich nur in einem einzigen Literal, so dass zB immer zwei Felder  $xy$  und  $x\bar{y}$  zu  $x(y + \bar{y}) = x$  zusammengefasst werden können.

### KV-Diagramm mit zwei Variablen

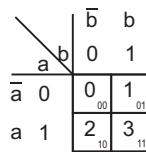


Abbildung 3.11: KV-Diagramm mit zwei Variablen

### KV-Diagramm mit drei Variablen

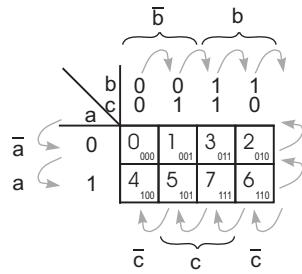


Abbildung 3.12: KV-Diagramm mit drei Variablen

### KV-Diagramm mit vier Variablen

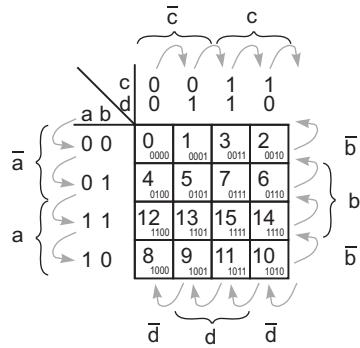


Abbildung 3.13: KV-Diagramm mit vier Variablen

### Eckpunkte im KV-Diagramm:

	c	0	0	1	1
a	b	0	1	1	0
0 0		1	.	.	1
0 1		.	.	.	.
1 1		.	.	.	.
1 0		1	.	.	1

Vereinfachung mittels Schaltalgebra:

$$\bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}c\bar{d} + a\bar{b}\bar{c}\bar{d} + a\bar{b}c\bar{d} =$$

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

$$\bar{a}\bar{b}\bar{d} + a\bar{b}\bar{d} =$$

$$\bar{b}\bar{d}(a + \bar{a}) =$$

$$\bar{b}\bar{d}$$

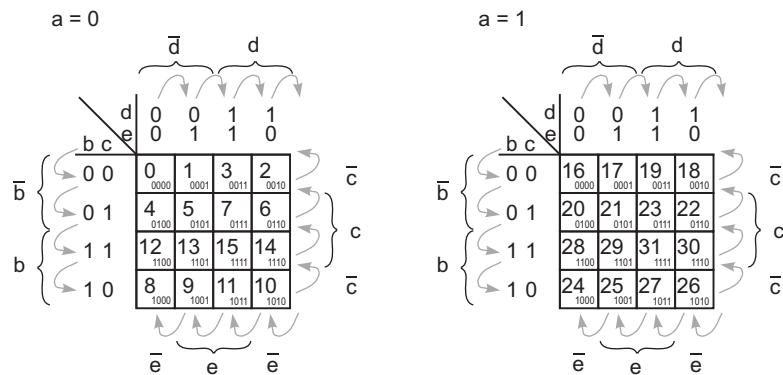
**KV-Diagramm mit fünf Variablen**


Abbildung 3.14: KV-Diagramm mit fünf Variablen

**KV-Diagramm mit sechs Variablen**

The diagram shows four KV-diagrams for six variables (a, b, c, d, e, f) with 64 states each.

		a = 0				a = 1			
		b = 0				b = 1			
		c	b	f	e	c	b	f	e
0 0		0	0	1	1	0	16	48	0
0 1									1
1 1							31		1
1 0									0
0 0		32							1
0 1									1
1 1					47				1
1 0									0

Abbildung 3.15: KV-Diagramm mit sechs Variablen

## 2. Primimplikanten identifizieren

**Beispiel 17:**  $f(a, b, c) = \bar{a}bc + \bar{a}b\bar{c}$

gegeben:  
 $f = \bar{a}bc + \bar{a}b\bar{c}$

vereinfacht:  
 $f = \bar{a}b(c + \bar{c}) = \bar{a}b$

	b	0	0	1	1	1
c	0	0	1	1	1	0
a	0	0	0	1	1	1
0	0	0	1	1	1	1
1	0	0	1	0	0	0

$\bar{a}b$  markiert die Primimplikante  $\bar{a}b$ .

**Beispiel 18:**  $f(a, b, c) = abc + ab\bar{c} + \bar{a}bc + \bar{a}b\bar{c}$

gegeben:  
 $f = abc + ab\bar{c} + \bar{a}bc + \bar{a}b\bar{c}$

vereinfacht:  
 $f = ab(c + \bar{c}) + \bar{a}b(c + \bar{c})$   
 $f = ab + \bar{a}b = b(a + \bar{a}) = b$

	b	0	0	1	1	1
c	0	0	1	1	1	0
a	0	0	0	1	1	1
0	0	0	1	1	1	1
1	0	0	1	0	0	0

$b$  markiert die Primimplikante  $b$ .

**Beispiel 19:**  $f(a, b, c) = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + a\bar{b}\bar{c} + a\bar{b}\bar{c}$

gegeben:  
 $f = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + a\bar{b}\bar{c} + a\bar{b}\bar{c}$

vereinfacht:  
 $f = \bar{a}\bar{c}(b + \bar{b}) + a\bar{c}(b + \bar{b})$   
 $f = \bar{a}\bar{c} + a\bar{c} = \bar{c}(a + \bar{a}) = \bar{c}$

	b	0	0	1	1	0
c	0	1	0	0	1	1
a	0	1	0	1	1	0
0	1	0	0	0	1	1
1	1	0	0	1	0	1

$\bar{c}$  markiert die Primimplikante  $\bar{c}$ .

### 3.9.2 Vereinfachung

**Beispiel 20:** Vereinfachung von  $f(A, B, C, D) = \overline{A} \overline{B} \overline{C} D + \overline{A} \overline{B} C \overline{D} + \overline{A} B \overline{C} \overline{D} + \overline{A} B C \overline{D} + A \overline{B} \overline{C} D$

Ausgangssituation:

$$\begin{array}{rccccc} 0001 & & 0010 & & 0100 & & 0110 \\ \text{1} & & \text{2} & & \text{4} & & \text{6} \\ \overline{A} \overline{B} \overline{C} D & + & \overline{A} \overline{B} C \overline{D} & + & \overline{A} B \overline{C} \overline{D} & + & \overline{A} B C \overline{D} \\ & & & & & & \\ & & & & & & \text{9} \\ & & & & & & A \overline{B} \overline{C} D \end{array}$$

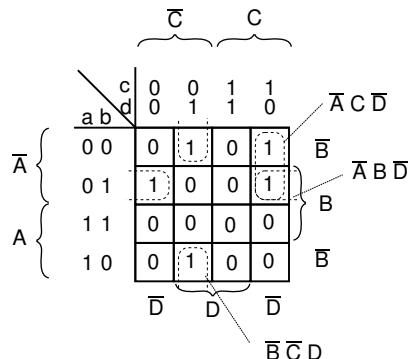
Vereinfachung mittels boolescher Algebra

$$\begin{aligned} & \overline{B} \overline{C} D (\overline{A} + A) + \overline{A} \overline{B} C \overline{D} + \overline{A} B \overline{C} \overline{D} + \overline{A} B C \overline{D} \\ & \downarrow \\ & \overline{B} \overline{C} D (\overline{A} + A) + \overline{A} C \overline{D} (\overline{B} + B) + \overline{A} B \overline{C} \overline{D} + \overline{A} B C \overline{D} \\ & \downarrow \\ & \overline{B} \overline{C} D (\overline{A} + A) + \overline{A} C \overline{D} (\overline{B} + B) + \overline{A} B \overline{D} (\overline{C} + C) \\ & \downarrow \\ & \overline{B} \overline{C} D + \overline{A} C \overline{D} + \overline{A} B \overline{D} \end{aligned}$$

Vereinfachung mittels KV-Diagramm

1. Aufstellen des KV-Diagrammes

Gegebene Minterme:  $\overline{A} \overline{B} \overline{C} D$ ,  $\overline{A} \overline{B} C \overline{D}$ ,  $\overline{A} B \overline{C} \overline{D}$ ,  $\overline{A} B C \overline{D}$  und  $A \overline{B} \overline{C} D$



2. Primimplikanten:  $\overline{A} B \overline{D}$ ,  $\overline{A} C \overline{D}$ ,  $\overline{B} \overline{C} D$

3. Wesentliche Primimplikanten:  $\overline{A} B \overline{D}$ ,  $\overline{A} C \overline{D}$ ,  $\overline{B} \overline{C} D$

4. Minimale Überdeckung:  $\overline{A} B \overline{D} + \overline{A} C \overline{D} + \overline{B} \overline{C} D$

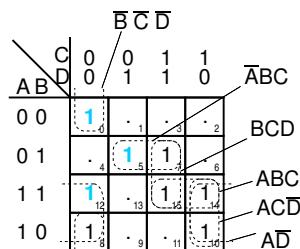
### 3.9.3 Weitere Beispiele

*Beispiel 21: f(A,B,C,D)*

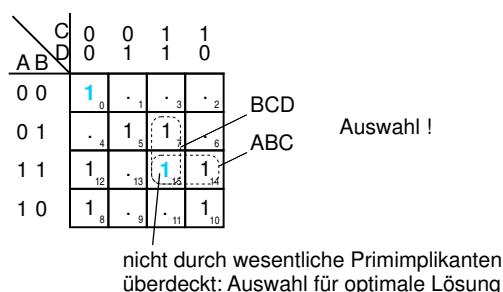
ABCD	f
0 0 0 0	1
1 0 0 1	0
2 0 0 1 0	0
3 0 0 1 1	0
4 0 1 0 0	0
5 0 1 0 1	1
6 0 1 1 0	0
7 0 1 1 1	1
8 1 0 0 0	1
9 1 0 0 1	0
10 1 0 1 0	1
11 1 0 1 1	0
12 1 1 0 0	1
13 1 1 0 1	0
14 1 1 1 0	1
15 1 1 1 1	1

**Minterme:**  $m_0, m_5, m_7, m_8, m_{10}, m_{12}, m_{14}, m_{15}$   
 $\overline{ABC}\overline{D}, \overline{ABC}D, \overline{ABC}D, \dots$

**Primimplikanten:**  $\overline{B}\overline{C}\overline{D}, BCD, \overline{A}BD, A\overline{D}, ABC$



**Wesentliche Primimplikanten:**  $\overline{B}\overline{C}\overline{D}, A\overline{D}, \overline{A}BD$



**Zwei gleichwertige Lösungen:**

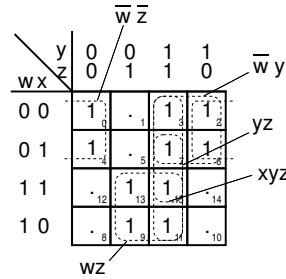
**Lösung 1:**  $\overline{B}\overline{C}\overline{D} + A\overline{D} + \overline{A}BD + BCD$

**Lösung 2:**  $\overline{B}\overline{C}\overline{D} + A\overline{D} + \overline{A}BD + ABC$

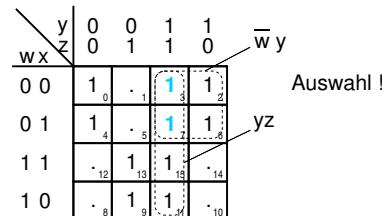
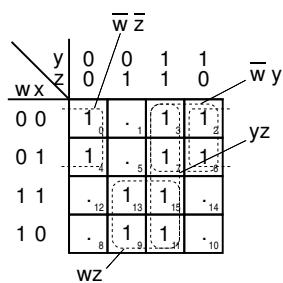
Beispiel 22:  $f(w,x,y,z)$

w	x	y	z	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	0	1	1
4	0	1	0	1
5	0	1	1	0
6	0	1	0	1
7	0	1	1	1
8	1	0	0	0
9	1	0	1	1
10	1	0	0	0
11	1	0	1	1
12	1	1	0	0
13	1	1	1	1
14	1	1	0	0
15	1	1	1	1

Primimplikanten:  $\bar{w}\bar{z}$ ,  $yz$ ,  $wz$ ,  $\bar{w}y$



Wesentliche Primimplikanten:  $\bar{w}\bar{z}$ ,  $wz$



Zwei gleichwertige Lösungen

Lösung 1:  $F_1 = wz + \bar{w}\bar{z} + yz$

Lösung 2:  $F_2 = wz + \bar{w}\bar{z} + \bar{w}y$

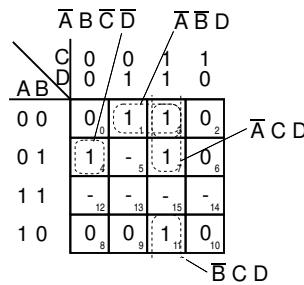
**Beispiel 23:** Unvollständig definierte Schaltfunktion (don't care)

A	B	C	D	f
0	0	0	0	0
1	0	0	1	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	1
5	0	1	0	-
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	1	0
10	1	0	1	0
11	1	0	1	1
12	1	1	0	-
13	1	1	0	-
14	1	1	1	-
15	1	1	1	-

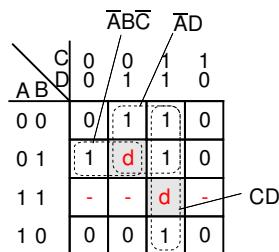
$$m_1 + m_3 + m_4 + m_7 + m_{11}$$

don't cares: (5, 12, 13, 14, 15)

Lösung 1:  $F_1 = \overline{A} \overline{B} D + \overline{A} C D + \overline{A} B \overline{C} \overline{D} + \overline{B} C D$



Lösung 2:  $F_2 = \overline{A} D + \overline{A} B \overline{C} + C D$



**Lösung 3:**  $F_3 = \overline{AD} + B\overline{C} + CD$

		C	B	$\overline{C}$	$\overline{A}$	D
		0	0	1	1	0
AB	D	0	0	1	1	0
0 0	0	0	1	1	0	
0 1	1	1	d	1	0	
1 1	d	d	d	d	d	CD
1 0	0	0	1	0		

Lösung 3 realisiert folgende Schaltfunktion:

A	B	C	D	f
0	0	0	0	0
1	0	0	1	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1 (-)
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	1	0
10	1	0	1	0
11	1	0	1	1
12	1	1	0	1 (-)
13	1	1	0	1 (-)
14	1	1	1	0 (-)
15	1	1	1	1 (-)

# Kapitel 4

## Kombinatorische Grundschaltungen

### 4.1 Decodierer (Decoder)

Beispiele für Decoder sind: BCD-zu-Dezimal-Decoder, Display-Decoder und Address-Decoder.

Adressdecodierer werden meist in grössere Einheiten eingebunden, mit der Aufgabe, aus n Komponenten eine auszuwählen bzw. zu aktivieren. Ist dies der Fall, kann jeder Komponente ein Index zwischen 0 und n-1 zugewiesen werden, der durch eine Binäradresse A repräsentiert wird. Um nun eine Komponente auszuwählen, wird A in n Auswahlleitungen decodiert, bei denen nur eine den Wert 1 hat. Generell besitzt ein m-zu-n Decodierer  $m = \log_2(n)$  Eingabeleitungen  $A_{m-1}..A_0$  und n Ausgangsleitungen  $C_{n-1}..C_0$ , sowie einen Kontroll-Eingang E, der die Ausgangsleitungen steuert.

$E = 0$ : alle Ausgangsleitungen 0

$E = 1$ : nur jene Ausgangsleitungen  $C_i$  ist 1,  
dessen i den Binärwert der Eingangsleitungen  $A_{m-1}..A_0$  entspricht

Abbildung 4.1 zeigt einen 1-zu-2 Decodierer. Dieser hat eine Adressleitung  $A_0$ , eine Kontrollleitung E und zwei Ausgangsleitungen,  $C_1$  und  $C_0$ .

E	$A_0$	$C_1$	$C_0$
1	0	0	1
1	1	1	0
0	X	0	0

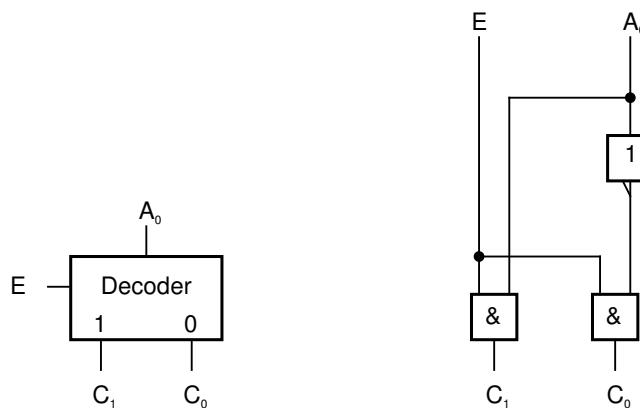


Abbildung 4.1: Schaltbild und Symbol eines Decoders

Falls nötig, kann der 1-zu-2 Decodierer zu einem 2-zu-4 Decodierer erweitert werden. Dafür werden dann zwei Adressleitungen,  $A_1$  und  $A_0$ , und vier Ausgangsleitungen benötigt,  $C_3$ ,  $C_2$ ,  $C_1$  und  $C_0$ . Unter Verwendung von 1-zu-2 bzw. 2-zu-4 Decodierer ist es möglich, grössere m-zu-n Decodierer zu erstellen. Folgende Abbildung zeigt einen 3-zu-8 Decodierer, der auf beide Arten (mittels 1-zu-2,

2-zu-4 Decodierer) realisiert wurde.

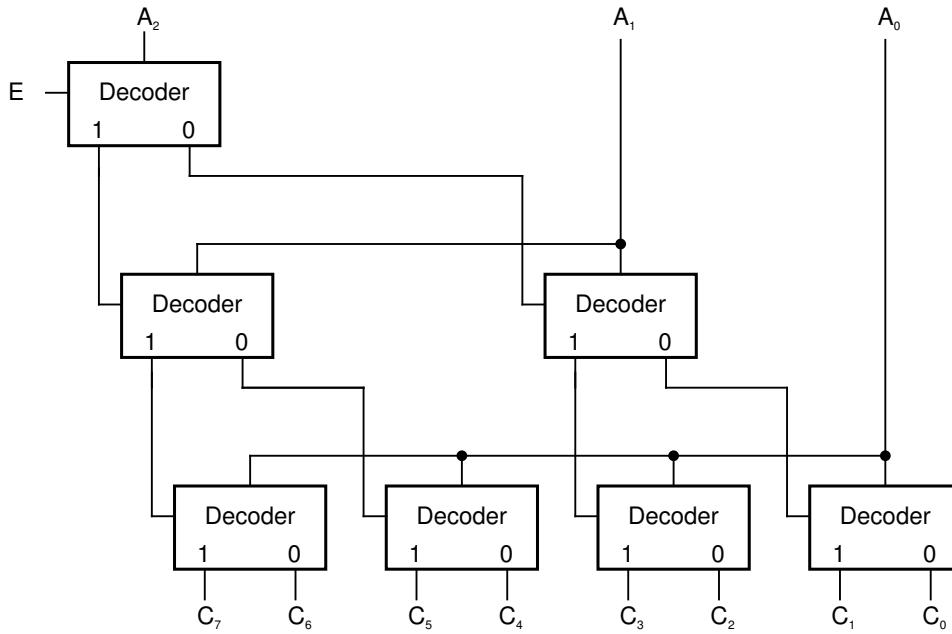


Abbildung 4.2: 3-zu-8 Decodierer mit 1-zu-2 Decodierern aufgebaut

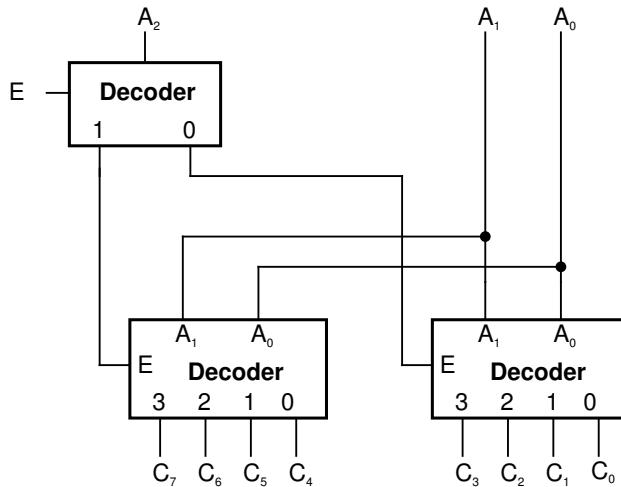


Abbildung 4.3: 3-zu-8 Decodierer mit 2-zu-4 Decodierern aufgebaut

Werden 1-zu-2 Decodierer verwendet, werden  $\log_2(n)$ , im Beispiel also  $\log_2(8) = 3$  Ebenen von Decodierern benötigt. Jede dieser 3 Ebenen decodiert ein Adress-Bit. Das MSB (most significant bit) der Adress-Bits wird sozusagen von einem Decodierer bearbeitet, das nächst wichtigste Bit von zwei Decodierer, ..., bis das letzte Bit von  $n/2$  Decodierer bearbeitet wurde. Jeder Ausgang der Decodierer einer Ebene setzt die Kontrollleitung der darunterliegenden Ebene. Daraus resultiert die Verdopplung der Anzahl der Decodierer jeder Ebene.

Das selbe Prinzip gilt auch für die Verwendung von 2-zu-4 Decodierer, wobei jedoch diese Decodierer zwei Bits gleichzeitig decodieren können. Somit werden nur mehr halb so viele Ebenen und Decodierer als bei einer Realisierung mit 1-zu-2 Decodierer benötigt.

## 4.2 Codierer mit Priorität (Priority Encoders)

Ein Codierer erzeugt ein eindeutiges Ausgabecodewort für jede mögliche Belegung seiner Eingänge. Ein Priority-Encoder hat  $n$  Eingänge ( $D_{n-1}..D_0$ ) und  $m = \log_2(n)$  Ausgänge ( $A_{m-1}..A_0$ ). Somit gilt  $n = 2^m$ .

Ein Priority-Encoder besitzt weiters eine Ausgangsleitung Any, bei der 1 anliegt, falls irgendeine der Eingangsleitungen von 0 verschieden ist. Die Ausgangsleitungen geben den Index des signifikantesten Eingangs-Bits  $D_i$  das den Wert 1 hat aus. Die Priorität wird also durch die Indizes der Eingangsleitungen bestimmt (sei  $n - 1 \leq i \leq 0$ , dann gilt: je grösser  $i$  desto grösser die Priorität der Leitung  $D_i$ ).

Im folgenden ist der einfachste dieser Codierer dargestellt:

$D_1$	$D_0$	$A_0$	Any
0	0	0	0
0	1	0	1
1	x	1	1

$$A_0 = D_1$$

$$\text{Any} = D_0 + D_1$$

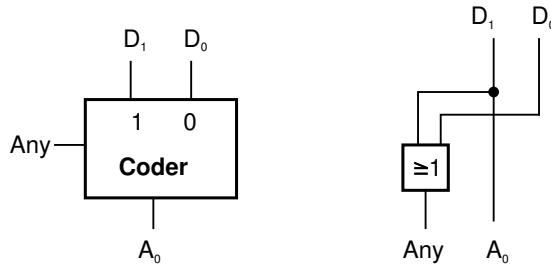


Abbildung 4.4: Schaltbild und Symbol eines Codierers

Nützlicher sind 4-zu-1 Codierer, die vier Eingangsleitungen  $D_3$  bis  $D_0$  und neben dem Any Ausgang zwei Adressausgänge  $A_1$  und  $A_0$  besitzen.

Die Bedeutung von 2-zu-1 und 4-zu-1 Codierern liegt darin, dass durch Kombination dieser Bausteine mit Selektoren eine grössere Anzahl von Eingangs-Bits bewältigt werden kann.

Abbildung 4.5 zeigt einen 4-zu-2 Codierer.

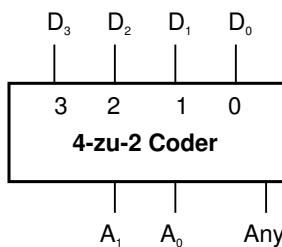


Abbildung 4.5: Symbol eines 4-zu-2 Codierers

Die folgende Wertetabelle veranschaulicht die Funktion des 'Any'-Ausgangs.

D3	D2	D1	D0	Encoder		Priority-Encoder		
				A1	A0	A1	A0	Any
0	0	0	0			0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	1	0	1	1
0	0	1	1			0	1	1
0	1	0	0	1	0	1	0	1
0	1	0	1			1	0	1
0	1	1	0			1	0	1
0	1	1	1			1	0	1
1	0	0	0	1	1	1	1	1
1	0	0	1			1	1	1
1	0	1	0			1	1	1
1	0	1	1			1	1	1
1	1	0	0			1	1	1
1	1	0	1			1	1	1
1	1	1	0			1	1	1
1	1	1	1			1	1	1

Tabelle 4.1: Wertetabelle für einen 4-zu-2 Coder (mit Priorität)

## 4.3 Multiplexer (Selektoren)

Oft kommt es vor, dass Komponenten zu bestimmten Zeiten Daten benötigen, die von verschiedenen Quellen stammen. Um diese Aufgabe zu lösen werden Selektoren (Multiplexer) eingesetzt. Selektoren sind Bausteine, die eine von verschiedenen Datenquellen auswählen können, die zum Beispiel als Operanden für eine ALU verwendet werden, zum Speichern gedacht sind oder über einen Bus transportiert werden sollen.

$S_1$	$S_0$	$\mathbf{Y}$
0	0	$D_0$
0	1	$D_1$
1	0	$D_2$
1	1	$D_3$

$$Y = \overline{S_1} \overline{S_0} D_0 + \overline{S_1} S_0 D_1 + S_1 \overline{S_0} D_2 + S_1 S_0 D_3$$

Im Allgemeinen haben Selektoren  $n$  Eingänge, einen Ausgang und  $\log_2 n$  Auswahlleitungen, die eine der  $n$  Eingänge zum Ausgang durchschalten. Theoretisch lassen sich beliebige  $n$ -zu-1 Selektoren erstellen, in der Praxis haben sich aber 2-zu-1 und 4-zu-1 Selektoren als Grundbausteine bewährt. Aus diesen Selektoren lassen sich leicht andere bauen, die eine grössere Anzahl an Eingängen haben.

Die Abbildung 4.6 zeigt einen 4-zu-1 Selektor, der jenen von vier Eingängen selektiert, der über die beiden Auswahlleitungen,  $S_1$  und  $S_0$  bestimmt wird. Für einen  $n$ -zu-1 Selektor, wobei  $n$  ein Vielfaches von 2 darstellt, werden  $\log_2(n)$  Auswahlleitungen und  $\log_2(n)$  Ebenen von 2-zu-1 Selektoren benötigt. Dabei dient jede Auswahlleitung als Kontrolleingang für alle Selektoren einer Ebene. Auf der ersten Ebene wählt jeder Selektor aus zwei Datenquellen, in den darunterliegenden Ebenen wählen die Selektoren aus zwei Ausgängen der Selektoren aus der darüberliegenden Ebene.

Als Alternative Vorgehensweise lässt sich zum Beispiel ein 8-zu-1 Selektor mit logischen Gattern realisieren. Im folgenden Fall wird ein 3-zu-8 Decodierer benutzt, um die 3 Auswahlleitungen zu decodieren.

Dies lässt sich zwar leicht realisieren, hat aber den Nachteil, dass man es nicht einfach erweitern kann. Die Anzahl und Grösse der Gatter im Decodierer und die Grösse des OR-Gatters würde mit

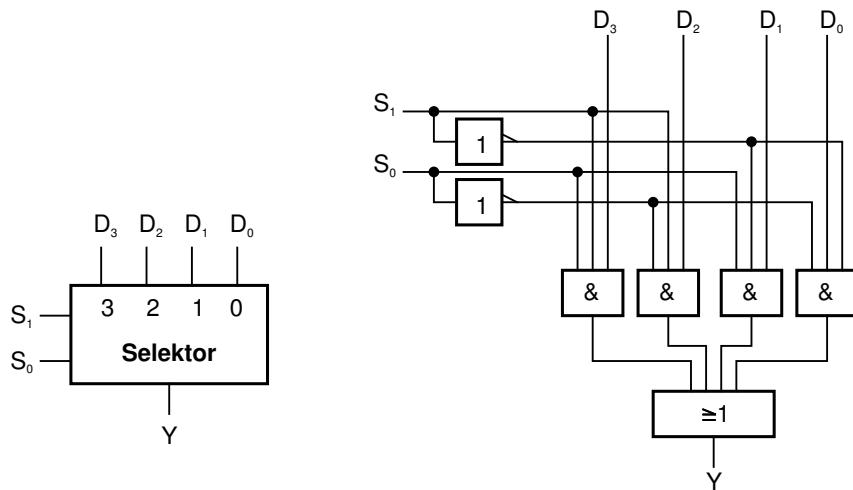


Abbildung 4.6: Schaltbild und Symbol eines Multiplexers

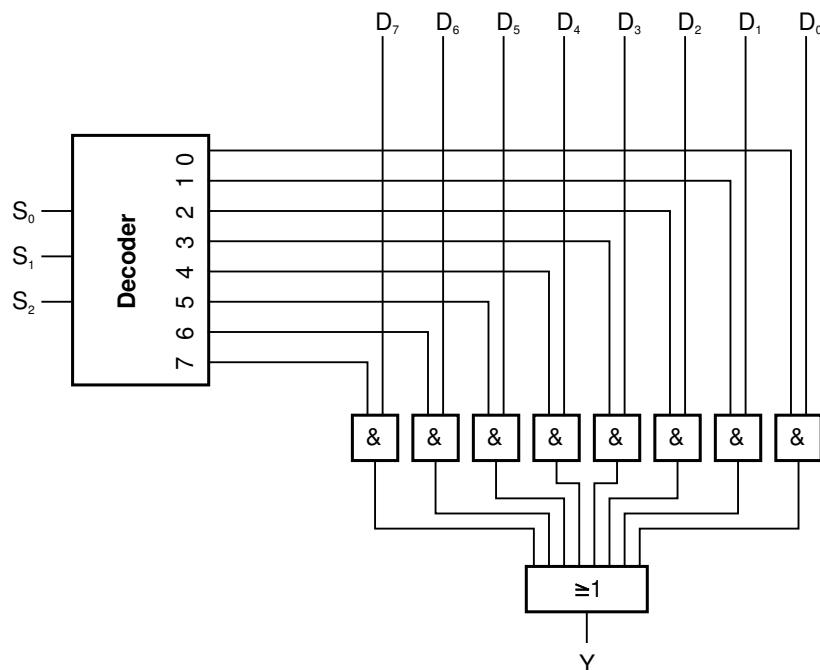


Abbildung 4.7: 8-zu-1 Selektor mit einem Decodierer

der der Eingangsleitungen steigen. Diese grösseren Gatter müssten als Baumstruktur von Gattern realisiert werden, was signifikante Folgen auf das Zeitverhalten des Selektors mit sich bringen würde. Daher ist die Bauweise mittels Gattern nur für kleine Werte von  $n$  praktikabel.

## 4.4 Demultiplexer (Distributor)

Funktion: Umschalter, Auswahl eines Ausgangs und Durchschalten des Eingangs auf diesen Ausgang.

Demultiplexer:

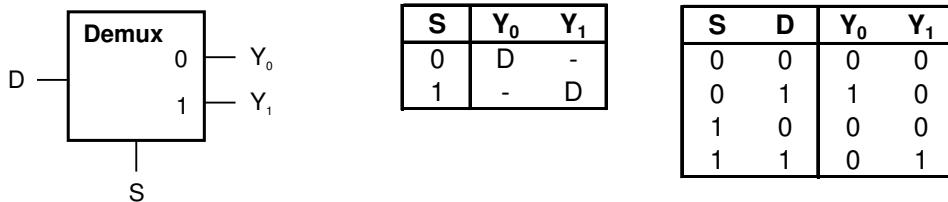


Abbildung 4.8: Schaltbild und Wertetabelle eines Demultiplexers

Demultiplexer mit Enable:

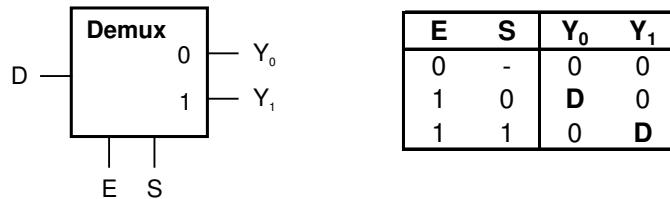


Abbildung 4.9: Demultiplexer mit Enable-Signal

Serielle Datenübertragung mit Multiplexer und Demultiplexer:



Abbildung 4.10: Serielle Datenübertragung

## 4.5 Ripple-Carry Adder

### 4.5.1 Was versteht man unter einem Ripple-Carry Adder?

Im folgendem Abschnitt werden die arithmetischen Komponenten beschrieben, die für die binäre Addition benötigt werden.

Zwei Binärzahlen  $x = x_{n-1} \cdots x_0$  und  $y = y_{n-1} \cdots y_0$ , deren MSBs (most significant bits) sich an der linken Stelle befinden, werden addiert, indem man jedes Bit-Paar  $x_i$  und  $y_i$  mit XOR verknüpft und dabei ein sogenanntes Carry-Bit  $c_i$  mitführt, welches den Überlauf der vorhergehenden Verknüpfung repräsentiert. Abbildung 4.11 zeigt die KV-Diagramme für  $s_i$  und  $c_{i+1}$  für einen Ripple-Carry Adder.

$s_i$	$x$	0	0	1	1	0
$c_i$	$y$	0	1	1	1	0
0		1		1		
1		1		1		

$c_{i+1}$	$x$	0	0	1	1	0
$c_i$	$y$	0	1	1	1	0
0		1		1		
1		1		1		

Abbildung 4.11: Ripple-Carry Adder: KV-Diagramm für  $s_i$  und  $c_{i+1}$ 

Das i-te Summen-Bit lässt sich ableiten:

$$s_i = (x_i \text{ } XOR \text{ } (y_i \text{ } XOR \text{ } (c_i))) \quad (4.1)$$

Das darauffolgende Überlauf-Bit ergibt sich:

$$c_{i+1} = (x_i \text{ } AND \text{ } y_i) \text{ } OR \text{ } (c_i \text{ } AND \text{ } (x_i \text{ } XOR \text{ } y_i)). \quad (4.2)$$

Ein Überlauf bei  $c_{i+1}$  tritt folglich dann auf, wenn beide Werte  $x_i$  und  $y_i$  den Wert 1 haben oder einer der beiden Werte und das Zugehörige (zuvor berechnete) Carry-Bit 1 sind.

$x_i$	$y_i$	$c_i$	$c_{i+1}$	$s_i$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabelle 4.2: Wertetabelle für einen Volladdierer

Unter Benutzung der Wertetabelle in Tabelle 4.2 lässt sich folgende Gatter-Schaltung in Bild 4.12 für einen 1-Bit Volladdierer (Full Adder, FA) ableiten, der im weiteren Verlauf durch VA (Volladdierer) dargestellt wird.

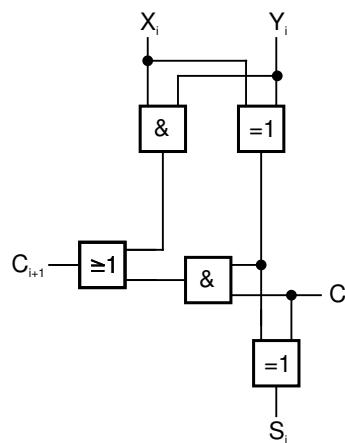


Abbildung 4.12: Schaltbild eines 1-Bit Volladdierers

Jeder Binäraddierer kann prinzipiell als Aneinanderreihung von VAs realisiert werden, wobei das Überlauf-Bit  $c_{i+1}$  eines VAs als das Überlauf-Bit  $c_i$  im nächst höheren VA verwendet wird.

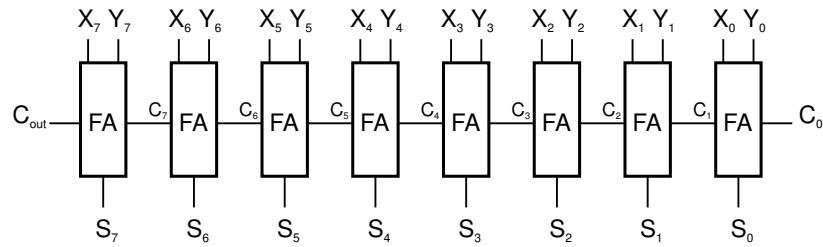


Abbildung 4.13: Schaltbild eines 8-Bit Volladdierers

Ein 8-Bit Addierer (Adder) besteht zum Beispiel aus 8 VAs, die wie in Bild 4.13 abgebildet, verbunden werden. So lassen sich Addierer für beliebig grosse Binärzahlen erstellen, wobei jedoch ein signifikanter Nachteil entsteht. Je mehr VAs in Serie geschaltet werden, desto länger benötigt die Schaltung, bis das Ergebnis am Ausgang anliegt, weil der nächste VA auf das jeweilige Carry-Bit des vorigen warten muss.

Diese Abhängigkeit war sozusagen namesgebend für den Ripple-Carry Adder, da das Carry-Bit alle VAs durchlaufen (durch "ripple'n") muss. Bild 4.14 zeigt einen 4-Bit Addierer im Ripple-Carry Design. Die Zahlangaben neben den Gattersymbolen stellen die Verzögerungszeiten der entsprechenden Gatter dar.

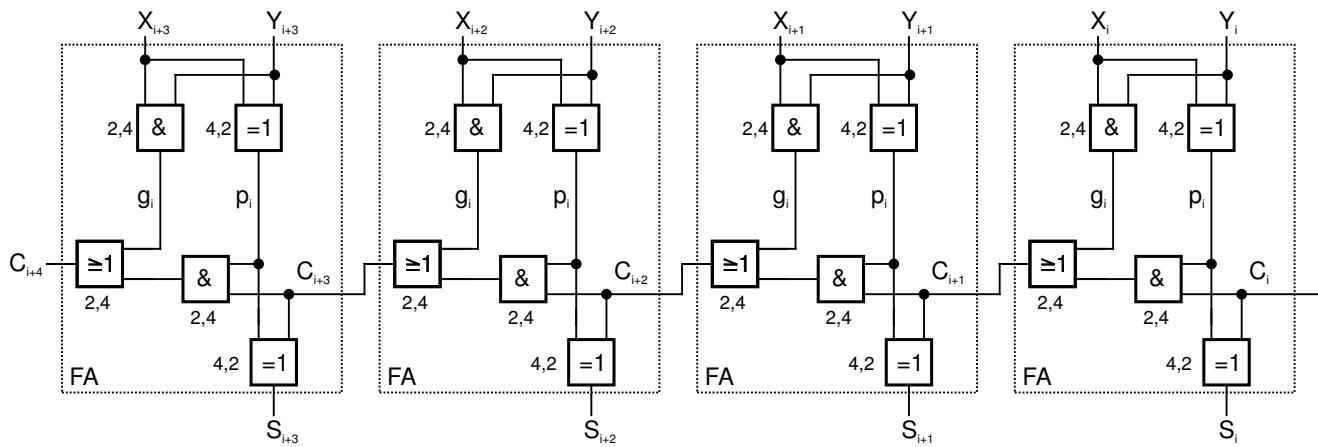


Abbildung 4.14: Schaltbild eines 4-Bit Ripple-Carry Adders

### 4.5.2 Carry-Look-Ahead Adder (CLA)

Um dem Nachteil der Verzögerung des Ripple-Carry Adders Herr zu werden, wurde das Konzept des Carry-Look-Ahead (CLA) eingeführt. Diese Technik beruht auf der Tatsache, dass der grösste Teil der Berechnung für jedes abhängige Carry-Bit vorverarbeitet werden kann. Dieses Konzept kann anhand eines 4-Bit Addierers gut veranschaulicht werden:

Dazu wird eine Funktion  $g_i = x_i \text{ AND } y_i$  zur Generierung des Carry-Bits und eine Transportfunktion  $p_i = x_i \text{ XOR } y_i$  für selbiges definiert. Die vier Carry-Bits  $c_{i+1..c_{i+4}}$  können durch

$$c_{i+1} = g_i + p_i c_i \quad (4.3)$$

$$c_{i+2} = g_{i+1} + p_{i+1} c_{i+1} \quad (4.4)$$

$$c_{i+3} = g_{i+2} + p_{i+2} c_{i+2} \quad (4.5)$$

$$c_{i+4} = g_{i+3} + p_{i+3} c_{i+3} \quad (4.6)$$

ausgedrückt werden ( $p_i c_i \rightarrow p_i \text{ AND } c_i; g_i + p_i c_i \rightarrow g_i \text{ OR } p_i c_i$ ).

Durch Vorwärts-Substitution lassen sich alle Gleichungen mit  $c_i$  ausdrücken, so dass keine Abhängigkeiten zwischen diesen mehr bestehen. Das hat den Vorteil, dass aus dem ersten Überlauf-Bit  $c_i$  sofort alle nachfolgenden berechnet werden können:

$$c_{i+1} = g_i + p_i c_i \quad (4.7)$$

$$c_{i+2} = g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_i \quad (4.8)$$

$$c_{i+3} = g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i c_i \quad (4.9)$$

$$c_{i+4} = g_{i+3} + p_{i+3} g_{i+2} + p_{i+3} p_{i+2} g_{i+1} + p_{i+3} p_{i+2} p_{i+1} g_i + p_{i+3} p_{i+2} p_{i+1} p_i c_i \quad (4.10)$$

Zwar hängt nun keine Gleichung mehr von  $c_{i+1}, c_{i+2}, c_{i+3}$  oder  $c_{i+4}$  ab, jedoch kann diese Technik schwer auf n-Bit erweitert werden, da die Gatterbausteine Beschränkungen bezüglich der Anzahl der Ein- und Ausgabe-Ports unterliegen. Für die Berechnung von  $c_{i+n}$  würden  $n+1$  Eingabe-Ports für AND und OR benötigt werden, da die Anzahl der Produkt- und Additionsterme für jedes zusätzliche Bit um eins zunimmt! Dies ließe sich zwar durch hierarchische Aneinanderreihung von Gattern realisieren, würde jedoch im Endeffekt in einer grösseren Verzögerungszeit des Addierers resultieren und dem Ziel der Look-Ahead Technik widersprechen. Da die Anzahl der Ausgangs-Ports ebenfalls mit der Variablenanzahl steigt, ist in der Praxis n auf 4 limitiert.

**Carry-Look-Ahead Generator:** Um CLAs mit höherer Bit-Breite zu generieren, werden mehrere 4-Bit CLA Generatoren verwendet, die sich aus der oben gezeigten Gleichung  $c_{i+4}$ , Formel 12, wie folgt ausdrücken lassen:

$$c_{i+4} = g_{(i,i+3)} + p_{(i,i+3)} c_i \quad (4.11)$$

$$g_{(i,i+3)} = g_{i+3} + p_{i+3} p_i + p_{i+3} p_{i+2} q_{i+1} + p_{i+3} p_{i+2} p_{i+1} g_i \quad (4.12)$$

$$p_{(i,i+3)} = p_{i+3} p_{i+2} p_{i+1} p_i \quad (4.13)$$

$g_{(i,i+3)}$  stellt die Generierungsfunktion der Carry-Bits dar, und  $p_{(i,i+3)}$  die Transportfunktion des initialen.  $c_i$ .

Abbildung 4.15 zeigt schemenhaft einen solchen CLA-Generator, welcher für die Berechnung der Carry-Bits von Addierern verwendet werden kann (siehe Abbildung 4.16).

Der Vergleich des CLA-Generators mit dem Ripple-carry Addierer macht klar, dass mit einem 4-Bit CLA-Generator ein schnellerer 4-Bit Addierer erzeugt werden kann, da alle vier Carry-Bits parallel berechnet werden. Somit lässt sich die Geschwindigkeit grösserer Addierer durch Aneinanderreihung von CLA-Generatoren (bei einem 16-Bit Addierer: vier 4-Bit-Generatoren), wesentlich steigern. Bei dieser Technik wird jeder 4-Bit Addierer mit einem 4-Bit CLA-Generator realisiert, wobei nur die Carry-Bits  $c_4, c_8, c_{12}$  und  $c_{16}$  jeweils zum nächst höher- signifikanten CLA-Generator weitergereicht (durchge"rippled") werden müssen.

Ein weiterer Geschwindigkeitsgewinn lässt sich erzielen, indem  $c_4, c_8, c_{12}$  und  $c_{16}$  durch einen eigenen CLA Generator erzeugt werden.

$$c_4 = g_{(0,3)} + p_{(0,3)} c_0 \quad (4.14)$$

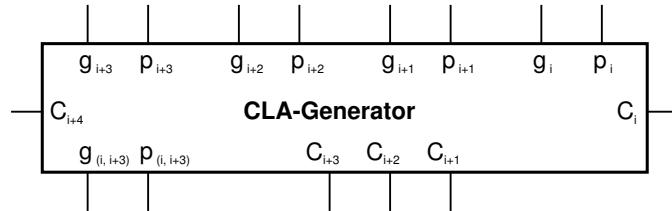


Abbildung 4.15: Schaltbild eines 4-Bit CLA Generators

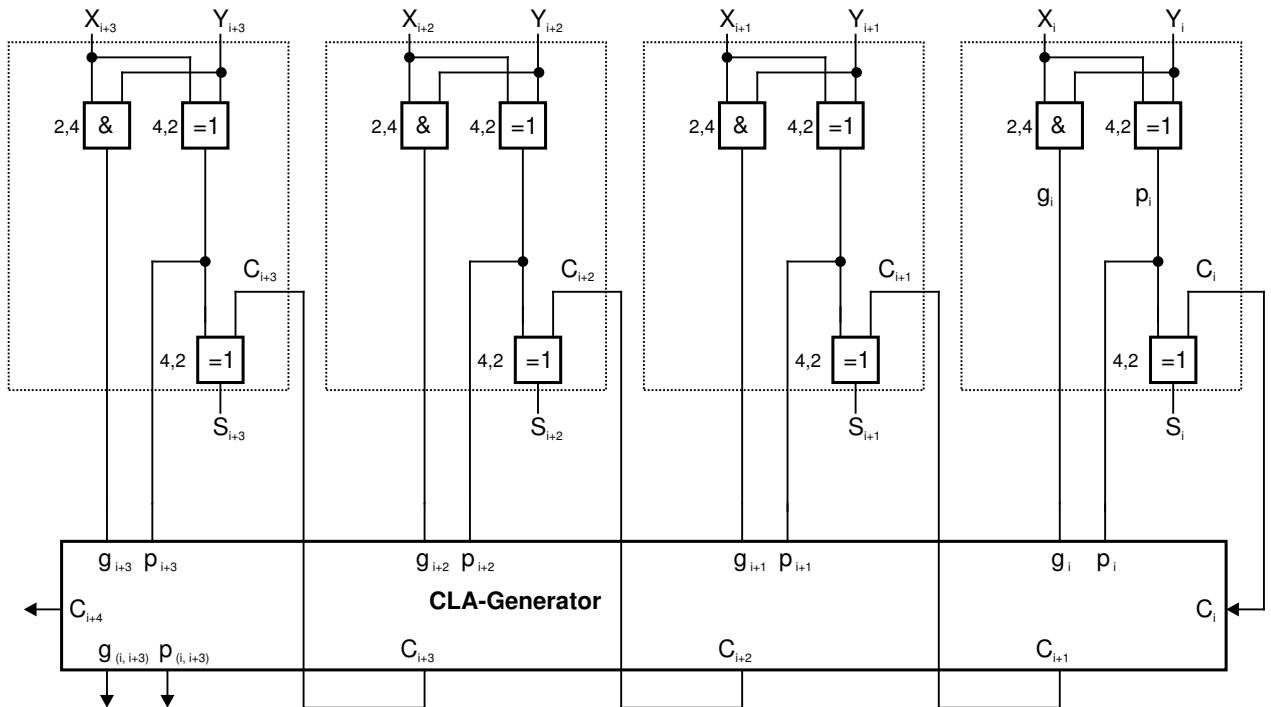


Abbildung 4.16: Symbol eines 4-Bit Addierers mit CLA Generatoren

$$c_8 = g_{(4,7)} + p_{(4,7)}c_4 \quad (4.15)$$

$$c_{12} = g_{(8,11)} + p_{(8,11)}c_8 \quad (4.16)$$

$$c_{16} = g_{(12,15)} + p_{(12,15)}c_{12} \quad (4.17)$$

Durch diese zweite Ebene können die Carry-Bits, die die einzelnen Generatoren weiterreichen, ignoriert werden, da diese der zusätzliche CLA-Generator bereitstellt. Somit kann auch das Weiterreichen der Carry-Bits  $c_4, c_8, c_{12}$  und  $c_{16}$  eingespart werden. In der Abbildung 4.17 ist ein solcher Addierer dargestellt.

Analog lassen sich 32- und 64-Bit Addierer erzeugen, wobei die Anzahl der Ebenen logarithmisch steigt. Im Allgemeinen werden für einen n-Bit Addierer ( $\log_2(n)$ ) Ebenen benötigt.

Vergleicht man nun den Zeitaufwand für das Weiterreichen des Carry-Bits bei einem 16-Bit Ripple-Carry Adder mit einem entsprechenden CLA Adder mit einer bzw. zwei Ebenen, so ist ein grosser Geschwindigkeitsunterschied feststellbar. Die Werte in Tabelle 4.3 beziehen sich auf die in Bild 4.14 dargestellten Gatterlaufzeiten.

### 4.5.3 Addierer/Subtrahierer

Die binäre Subtraktion wird wie die Addition realisiert, mit dem Unterschied, dass zuvor das Zweier-Komplement des Subtrahenden gebildet wird. Das Zweier-Komplement wird berechnet, indem jedes Bit invertiert und zu diesem Wert danach 1 addiert wird. Diese Addition lässt sich leicht durch Setzen des initialen Carry-Bits auf 1 durchführen.

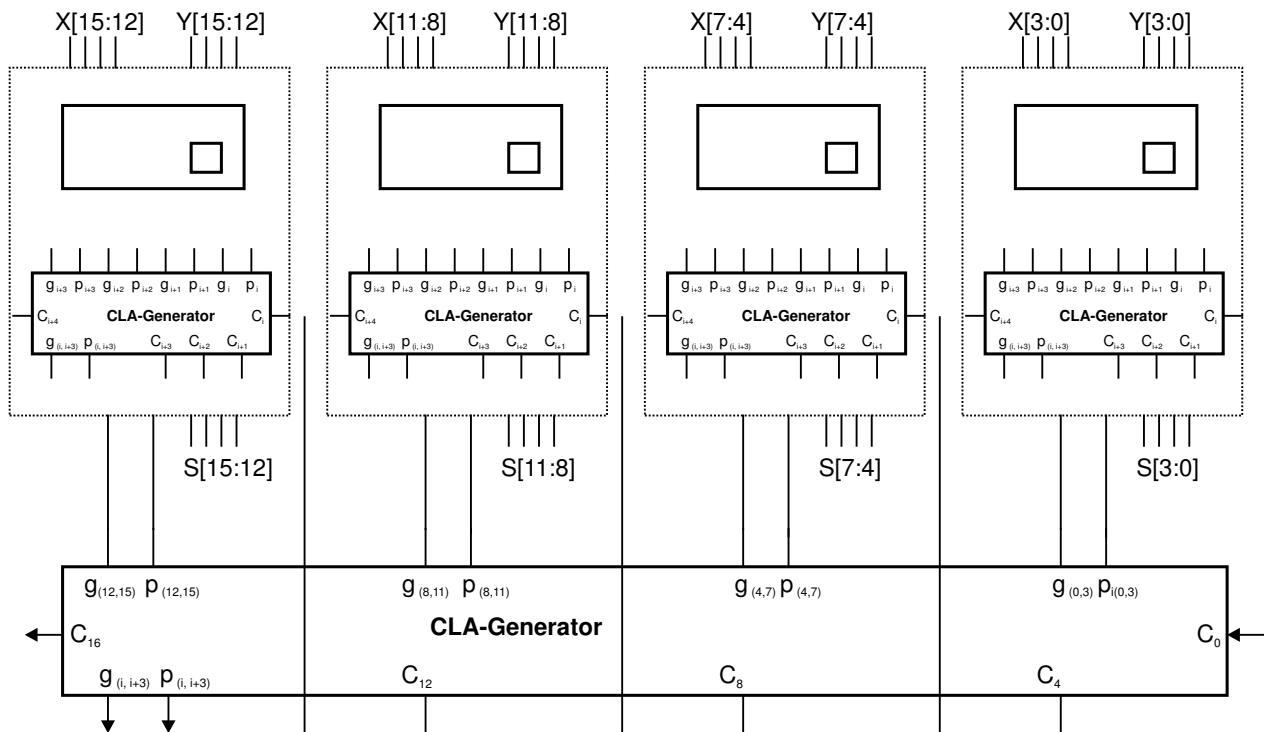


Abbildung 4.17: Schaltbild eines 16-Bit Addierers mit 2-Ebenen CLA Generator

Technik	Berechnung der Carry-Bits (ns)	Berechnung der Resultats (ns)
Ripple-Carry Adder	76.8	81.0
1-Ebenen CLA	19.2	27.4
2-Ebenen CLA	4.8	19.4

Tabelle 4.3: Vergleich der Verzögerungszeiten verschiedener Addierer-Techniken

Wegen dieser Ähnlichkeit von Addition und Subtraktion ist es sinnvoll, eine Funktionseinheit zu konstruieren, die beides beherrscht (siehe Abbildung 4.18).

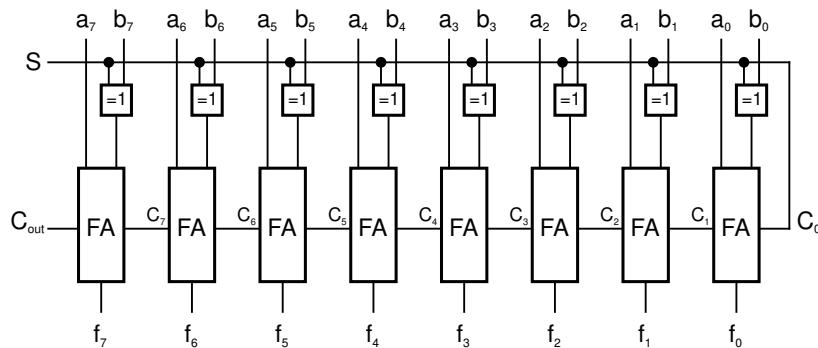


Abbildung 4.18: Schaltbild eines 8-Bit Addierer/Substrahierer

Für diese Funktionalität führt man eine weitere Eingangsleitung (SELECT) ein, bei der für Addition 0 und für Subtraktion 1 anliegt. Durch eine 'XOR'-Verknüpfung der SELECT Leitung mit den Eingangs-Bits des Subtrahenden erfolgt bei  $\text{SELECT} = 1$  die Invertierung. Ebenfalls wird durch die SELECT-Leitung das Carry-Bit  $c_0$  entsprechend initialisiert.

Bild 4.18 zeigt einen Addierer und Subtrahierer im Ripple-Carry Design. Eine Realisierung mittels CLA ist genauso gut möglich.

## 4.6 Logische Einheit (Logic Unit, LU)

Im Allgemeinen ermöglicht eine logische Einheit (logic unit, LU), welche mehrere bool'sche Funktionen  $f_i$  zur Verfügung stellt, die Auswertung einer solchen Funktion, die auf zwei Operanden,  $X = x_{n-1}..x_0$  und  $Y = y_{n-1}..y_0$ , angewendet wird. Durch Selektion einer Funktion  $f_i$  wird diese auf die beiden Operanden angewandt und das Ergebnis  $S = f_i(X, Y) = f_i(x_{n-1}, y_{n-1})..f_i(x_0, y_0)$  berechnet.

Da eine logische Einheit, die alle 16 bool'schen Funktionen mit zwei Variablen auswertet, nicht sehr komplex ist, lässt sich an einer solchen die Vorgehensweise zur Realisierung einer beliebigen logischen Einheit gut demonstrieren.

Tabelle 4.4 zeigt alle möglichen Funktionen für zwei Variablen:

	xi	yi	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	
m0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	S0
m1	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	S1
m2	1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	S2
m3	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	S3

Tabelle 4.4: bool'sche Funktionen mit zwei Variablen

Um eine dieser 16 Funktionen auswählen zu können, benötigt man 4 SELECT-Leitungen ( $S_0, S_1, S_2, S_3$ ). Jede Selektionsvariable  $S_i$  ( $0 \leq i \leq 3$ ) wird auf 1 gesetzt, wenn (und nur wenn) die Funktion für den Minterm  $m_i$  1 liefert.

Zum Beispiel muss für die Funktion  $f_{14}$  (OR) bei den SELECT-Leitungen  $S_3, S_2$  und  $S_1$  1 anliegen, bei  $S_0$  0. Betrachtet man die Binärzahl  $S_3S_2S_1S_0 = 1110$ , so entspricht diese der Zahl 14 im Dezimalsystem, die den Index der ausgewählten Funktion widerspiegelt.

Da jede SELECT-Leitung genau einen Minterm steuert, ergibt sich für eine 1-Bit logische Einheit folgende Gleichung und das folgende Schaltbild:

$$z_i = S_0 m_0 + S_1 m_1 + S_2 m_2 + S_3 m_3 \quad (4.18)$$

$$z_i = S_0 \bar{x}_i \bar{y}_i + S_1 \bar{x}_i \bar{y}_i + S_2 \bar{x}_i \bar{y}_i + S_3 \bar{x}_i \bar{y}_i \quad (4.19)$$

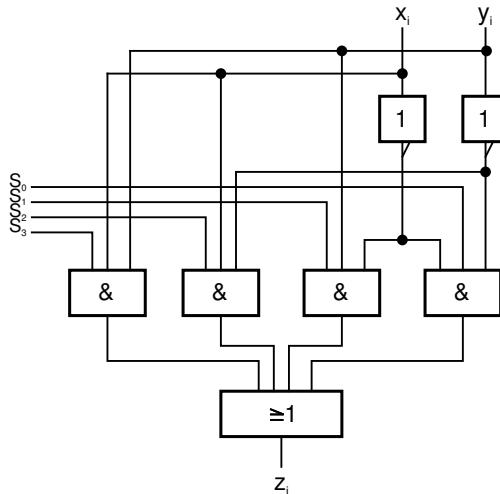


Abbildung 4.19: Schaltbild einer 1-Bit logischen Einheit

Für eine n-Bit LU werden n solcher 1-Bit LUs parallel geschaltet.

### 4.6.1 Arithmetische logische Einheit

#### *Arithmetic-Logic Unit, ALU*

Wie der Name bereits sagt, verarbeitet eine arithmetische logische Einheit (ALU) die grundlegenden arithmetischen und logischen Operationen in einem Mikroprozessor. Da all diese arithmetischen Operationen wie Addition, Subtraktion, Increment und Decrement auf der Addition basieren, kann man eine ALU einfach durch Veränderung der Eingabe eines Ripple-Carry oder CLA Adders konstruieren. Die Einheiten für die Anwendung der arithmetischen und logischen Operationen werden arithmetic extender (AE) bzw. logic extender (LE) genannt. Eine oder beide dieser Einheiten ist mit dem Eingang des Addierers verbunden.

#### Erstellung des arithmetic extenders (AE)

Da die ALU 4 arithmetische und 4 logische Operationen verarbeiten soll, wird eine Variable M (Modus) verwendet, die zwischen arithmetischen ( $M=1$ ) und logischen ( $M=0$ ) Operationen auswählt. Weiters benötigt man zwei SELECT-Leitungen,  $S_1$  und  $S_0$ , die die jeweils auszuführende Operation bestimmen.

M	S1	S0	Funktionsname	F	X	Y	$c_0$
1	0	0	Decrement	A-1	A	alle 1	0
1	0	1	Addition	A+B	A	B	0
1	1	0	Subtraktion	A+B'	A	B'	1
1	1	1	Increment	A+1	A	alle 0	1

Tabelle 4.5: Funktionstabelle des AE

Die obige Tabelle beinhaltet auch den Wert des ALU Ausgangs F und jene von  $c_0$  (Carry-Bit) und der beiden Adder-Eingänge X und Y, die für die Berechnung für F benötigt werden. Durch Aufspalten des Y-Einganges in  $b_i$  und  $y_i$  ergibt sich der bool'sche Ausdruck

$$y_i = M \overline{S_1} b_i + M S_0 b_i \quad (4.20)$$

der die Schaltung des AE realisiert.

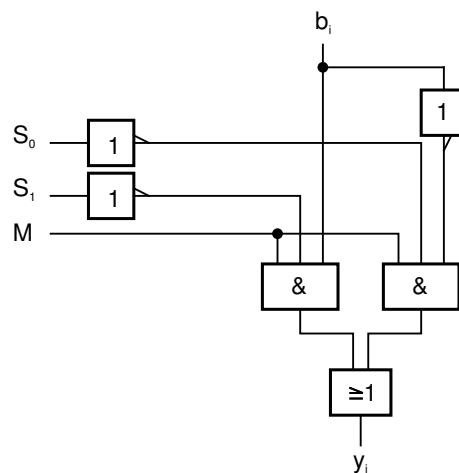


Abbildung 4.20: Schaltbild eines AR (arithmetic extenders)

### Erstellung des logic extenders (LE)

M	S1	S0	Funktionsname	F	X	Y	c <sub>0</sub>
0	0	0	Komplement	A'	A'	0	0
0	0	1	AND	A AND B	A AND B	0	0
0	1	0	Identität	A	A	0	0
0	1	1	OR	A OR B	A OR B	0	0

Tabelle 4.6: Funktionstabelle des LE

Anhand der Tabelle 4.6 erkennt man, dass Y und c<sub>0</sub> für alle logischen Operationen 0 sind. X hingegen benötigt für jede dieser Operationen einen anderen bool'schen Ausdruck. Daraus lässt sich auch hier der bool'sche Ausdruck und somit die Schaltung des LE ableiten:

$$x_i = \overline{M} \overline{S_1} \overline{S_0} \overline{a_i} + \overline{M} S_1 S_0 b_i + S_0 a_i b_i + S_1 a_i + M a_i \quad (4.21)$$

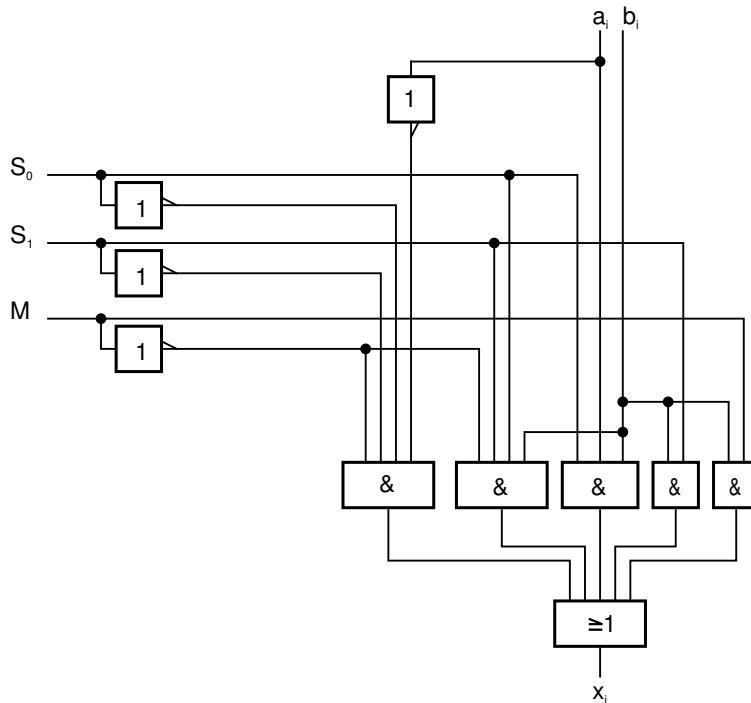


Abbildung 4.21: Schaltbild eines LE (logic extender)

#### 4-Bit ALU

Nachdem die Realisierung des AE und LE bekannt ist, besteht die nächste Aufgabe darin, diese mit einem Adder zu verbinden, um eine komplette ALU zu bekommen. Dies kann anhand einer 4-Bit ALU veranschaulicht werden.

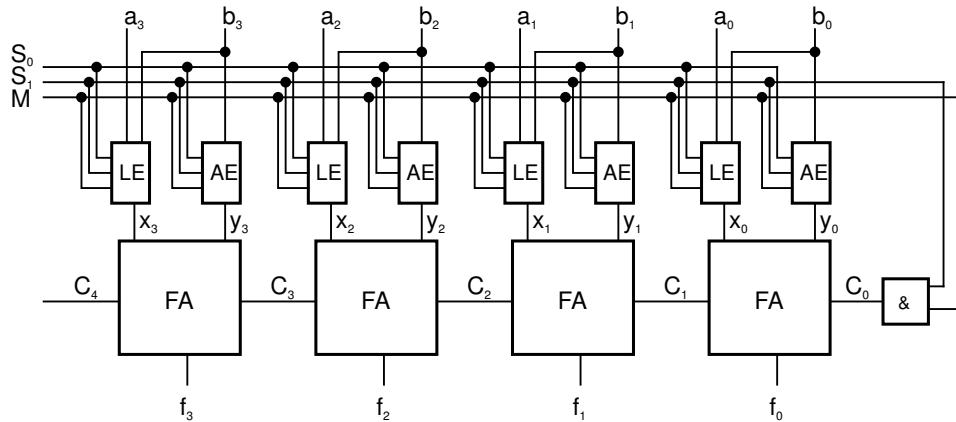


Abbildung 4.22: Schaltbild für eine 4-Bit ALU

Falls nötig, lässt sich die ALU auf n-Bit erweitern, indem ein n-Bit Addierer in Verbindung mit n AEs und n LEs verwendet wird.

Die meisten ALUs werden in der Praxis auf diese Weise erstellt. Unterscheidungen gibt es meist nur in der Anzahl und dem Typ ihrer arithmetischen und logischen Operationen und der Realisierung der Berechnung der Überlauf-Bits.

## 4.7 Busse

Obwohl die in dem vorigen Abschnitt beschriebenen Selektoren häufig verwendet werden, sind sie, wenn eine grosse Anzahl von Eingangsleitungen erforderlich ist, schwierig herzustellen, da der Verdrahtungsaufwand dann hoch ist. Es gibt jedoch eine elegante Lösung für solche Probleme: Busse.

Für die Konstruktion eines solchen benutzt man einen Baustein, genannt 'tristate driver'. Dessen Ausgang kennt die drei Zustände 0, 1 und Z. Z repräsentiert den hochohmigen Zustand (=high-impedance), der angibt, dass eine am Bus angeschlossene Komponente nicht als Quelle von Daten in Frage kommt.

E	Y
0	Z
1	D

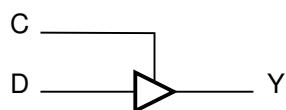


Abbildung 4.23: Schaltbild eines tristate drivers

Wie aus obiger Abbildung ersichtlich ist, hat ein tristate driver eine Datenleitung D, eine Kontrollleitung E und eine Ausgangsleitung Y. Immer, wenn an E der Wert 1 anliegt, wird die Datenleitung

auf Y durchgeschaltet. Ansonsten erhält Y den Wert Z.

Jeder Bus besitzt einen tristate driver pro Quelle. Da jedoch nur eine Quelle gleichzeitig aktiv sein darf, werden die Kontrollleitungen der Treiber als Quelladresse codiert und können somit effizient übertragen und gespeichert werden. Also ergeben sich für  $2^n$  Datenquellen n Adressleitungen, durch die nach Decodierung die n Kontrollleitungen angesprochen werden.

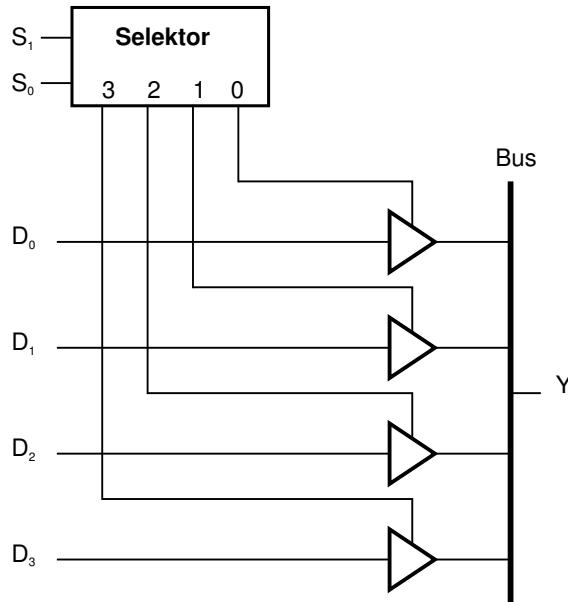


Abbildung 4.24: Bus mit 4 Eingängen

Der Bus ist in seiner Funktionsweise einem Selektor gleichwertig, da auch bei ihm von n Input Datenleitungen nur eine aktiv sein darf.

Aus obiger Abbildung geht hervor, dass aus den Leitungen  $S_1$  und  $S_0$  nach Decodierung die 4 Kontrollleitungen angesprochen werden. Busse sind prinzipiell einfach herzustellen und an die Anforderungen (z.B.: Anzahl der Datenquellen) anzupassen. Deshalb finden sie oft Verwendung in lokalen Netzwerken, an die viele Quellen auf engem Raum (selbes Gebäude, selber Raum) verbunden werden müssen.

# Kapitel 5

## Sequentielle Logik

### 5.1 Endliche Automaten (Finite State Machines)

Der Zustand eines endlichen Automaten  $M$  enthält zu jeder Zeit sämtliche Informationen über die Vergangenheit von  $M$ , die für das zukünftige Verhalten von  $M$  relevant sind.

#### 5.1.1 Begriffserklärung

$$M = (I, O, S, \delta, \lambda) \text{ bzw.}$$

$$M = (A, B, Q, \delta, \lambda)$$

I bzw. A: Endliche Menge von Inputs

O bzw. B: Endliche Menge von Outputs

S bzw. Q: Endliche Menge von Zuständen

$\delta$ :  $S \times I \rightarrow S$  Zustandsübergangsfunktion

$\lambda$ :  $S \times I \rightarrow O$  Ausgabefunktion

$\delta(s, i)$ : Folgezustand von Zustand  $s \in S$  und Input  $i \in I$

$\lambda(s, i)$ : Ausgabe bei Zustand  $s \in S$  und Input  $i \in I$

also:  $s_{t+1} = \delta(s_t, i_t)$  aber  $o_t = \lambda(s_t, i_t)$

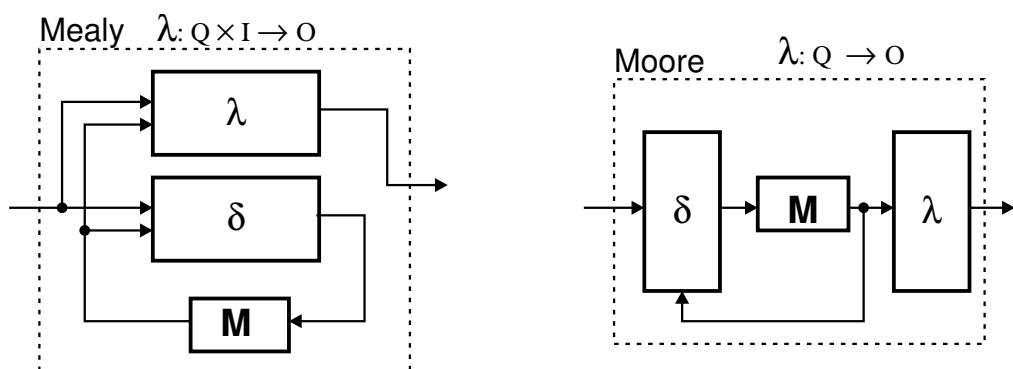


Abbildung 5.1: Mealy- und Moore-Typ

### 5.1.2 Beispiele

**Beispiel 1:** Ein Mealy Automat zur Erkennung der Folge 'ab' in beliebig langen Wörtern über dem Alphabet  $\{a, b\}$

Input-Menge :  $I = \{a, b\}$   
 Output-Menge :  $O = \{0, 1\}$   
 State-Menge :  $S = \{A, B\}$

t	0	1	2	3	4	5	6	7	8	9	10
Eingabe	a	a	b	a	a	b	b	a	b	a	b
Ausgabe	0	0	1	0	0	1	0	0	1	0	1

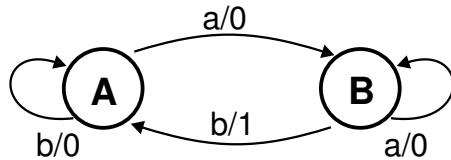


Abbildung 5.2: Beispiel Mealy-FSM: Erkennen von 'ab'

**Beispiel 2:** Ein Moore-Automat

Input-Menge :  $I = \{a, b\}$   
 Output-Menge :  $O = \{0, 1\}$   
 State-Menge :  $S = \{A, B, C\}$

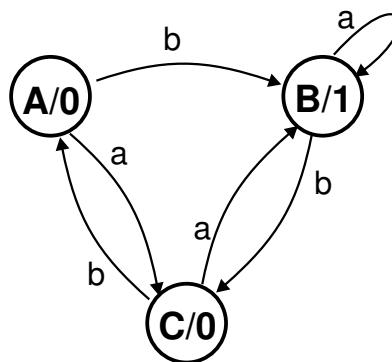


Abbildung 5.3: Beispiel Moore-FSM

Die Ausgabe ist nur vom Zustand abhängig!

Zustand	Folgezustand		Ausgabe
	a	b	
A	C	B	0
B	B	C	1
C	B	A	0

Input/Output-Experiment:

- Eingabewort: abbaba, Anfangszustand: A
- Ausgabewort: 00110, Endzustand: B

t	0	1	2	3	4	5	
Eingabe	a	b	b	a	b	a	
Zustand	A	C	A	B	B	C	B
Ausgabe	0	0	0	1	1	0	

**Beispiel 3:** Modellierung des Verhaltens eines Cola-Automaten als Mealy-FSM

Vereinfacht wird angenommen, dass der Cola-Automat nur ATS 5,- bzw. ATS 10,- akzeptiert. Ein Cola kostet ATS 20,-. Der Automat soll den eingeworfenen Betrag in ATS anzeigen bzw. eine Flasche Cola freigeben. Restgeld wird nicht berücksichtigt!

Input-Menge :  $I = \{5, 10\}$   
 Output-Menge :  $O = \{A0, A5, A10, A15, Cola\}$   
 State-Menge :  $S = \{S0, S5, S10, S15\}$

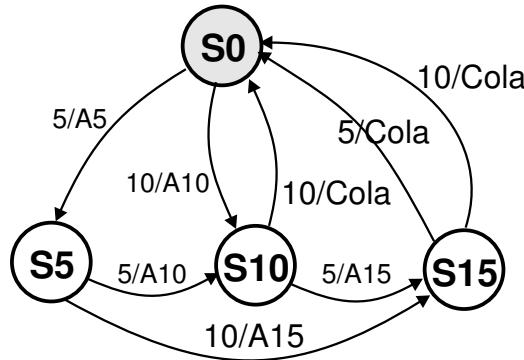


Abbildung 5.4: Cola-Automat

Tabellen für  $\delta$  und  $\lambda$

	$\delta$		$\lambda$	
	5	10	5	10
S0	S5	S10	S0	A5 A10
S5	S10	S15	S5	A10 A15
S10	S15	S0	S10	A15 Cola
S15	S0	S0	S15	Cola Cola

Input/Output-Experiment:

Eingabewort: 5 5 10 10 10 5

Anfangszustand: S0

t	0	1	2	3	4	5	
Zustand	S0	S5	S10	S0	S10	S0	S5
Eingabe	5	5	10	10	10	5	
Ausgabe	A5	A10	Cola	A10	Cola	A5	

Endzustand: S5

Ausgabewort: A5 A10 Cola A10 Cola

**Beispiel 4:** Fahrscheinautomat

Eingabe: 2 Tasten (TK, TL) und Geld (ATS 1,-)

Ausgabe: 2 Fahrscheine (FK, FL)

Anfangszustand: w (warten)

Input-Menge :  $I = \{TK, TL, 1\}$   
 Output-Menge :  $O = \{FK, FL, 1\}$   
 State-Menge :  $S = \{W, K, L, L1\}$

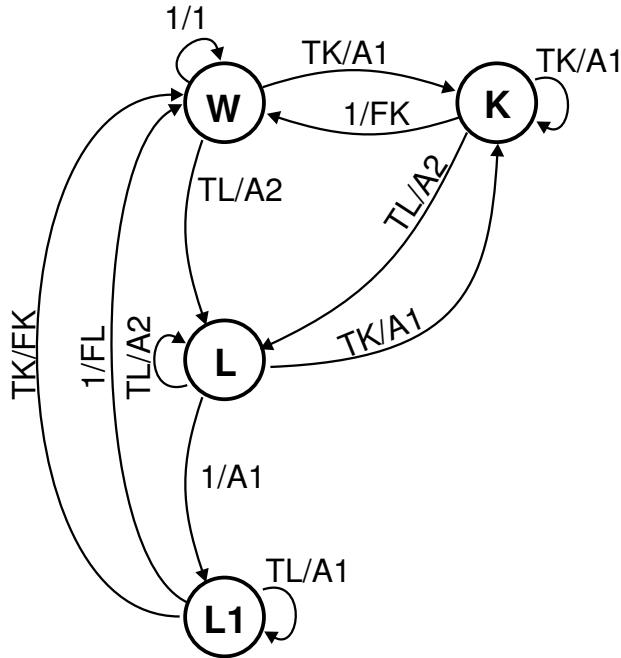


Abbildung 5.5: Fahrscheinautomat

Zustand	Folgezustand			Ausgabe		
	TK	TL	1	TK	TL	1
W	K	L	W	A1	A2	1
K	K	L	W	A1	A2	FK
L	K	L	L1	A1	A2	A1
L1	W	L1	W	FK	A1	FL

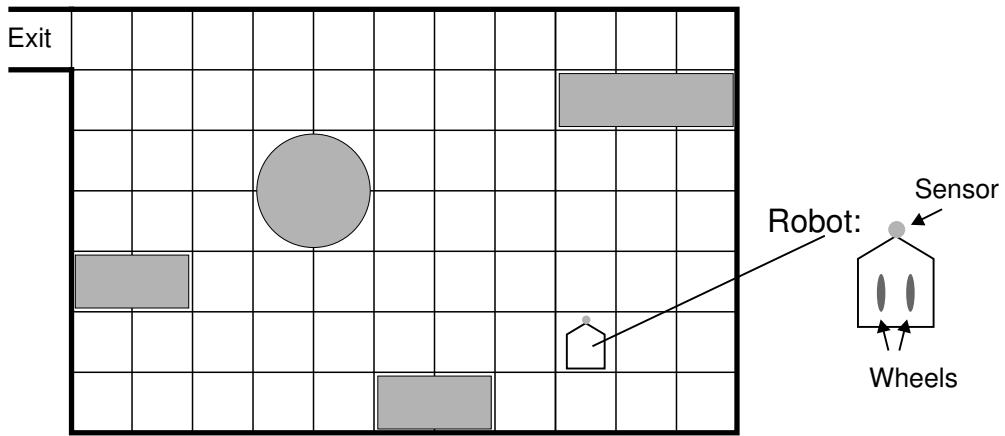
**Beispiel 5: Beispiel Roboter**

Abbildung 5.6: Beispiel Roboter

$$M = (I, O, S, \delta, \lambda)$$

$I = \{0, 1\}$	<b>0:</b> kein Hindernis
	<b>1:</b> Hindernis vorhanden
$O = \{G, L, R\}$	<b>G:</b> gerade
	<b>L:</b> links
	<b>R:</b> rechts
$S = \{U, V, W, X\}$	<b>U:</b> Weg frei, letzte Drehung: L
	<b>V:</b> Weg blockiert $\rightarrow$ Rechtsdrehung
	<b>W:</b> Weg frei, letzte Drehung: R
	<b>X:</b> Weg blockiert $\rightarrow$ Linksdrehung

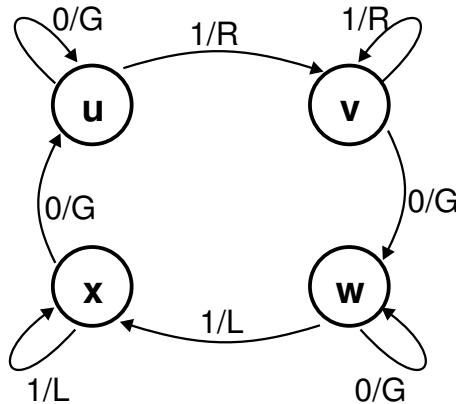


Abbildung 5.7: Ablaufdiagramm

$\delta$	a	b	$\lambda$	a	b
<b>u</b>	<b>u</b>	<b>v</b>	<b>u</b>	<b>G</b>	<b>R</b>
<b>v</b>	<b>w</b>	<b>v</b>	<b>v</b>	<b>G</b>	<b>R</b>
<b>w</b>	<b>w</b>	<b>x</b>	<b>w</b>	<b>G</b>	<b>L</b>
<b>x</b>	<b>u</b>	<b>x</b>	<b>w</b>	<b>G</b>	<b>L</b>

## 5.2 Latches und FlipFlops

### 5.2.1 RS-Latch

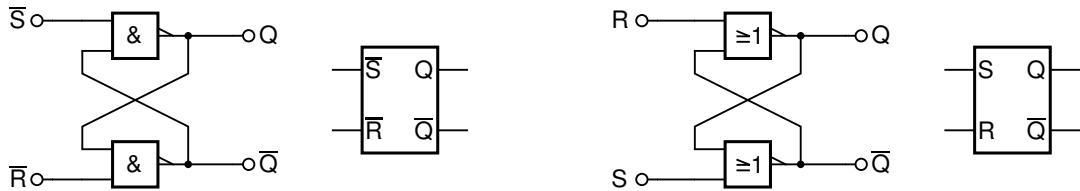


Abbildung 5.8: Schaltung und Schaltsymbol - RS-Latch

Ein RS-Latch ist ein 1-bit Speicher, d.h. es kann einen binären Wert 0 oder 1 am Ausgang speichern. Diese Funktion wird durch zwei verkreuzt rückgekoppelte Gatter erreicht, wie in Abbildung 5.8 ersichtlich ist. Es existieren grundsätzlich zwei Implementierungsmöglichkeiten mit den einfachen Gattern NAND und NOR, die sich auch in den Schaltsymbolen unterscheiden.

**RS-Latch: S=0, R=0 (Zwei Fälle von stabilen Zuständen möglich):**

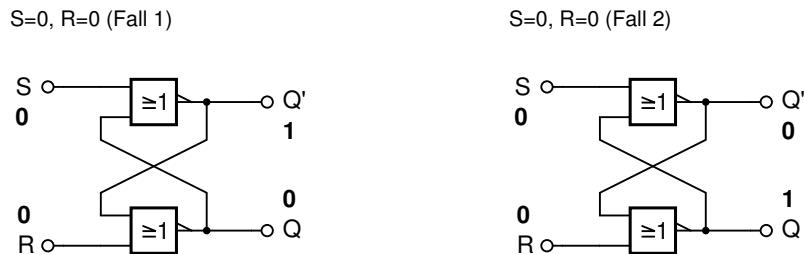
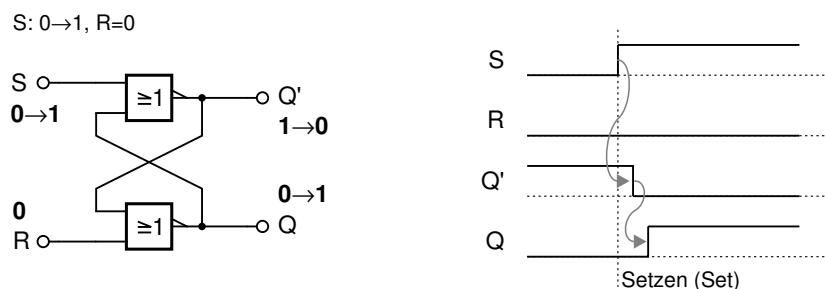


Abbildung 5.9: RS-Latch: S=0, R=0

**RS-Latch: S=0 $\rightarrow$ 1, R=0 (Bewirkt in beiden Fällen dieselben Werte für Q und Q'):**

Abbildung 5.10: RS-Latch: S=0 $\rightarrow$ 1, R=0, Fall1

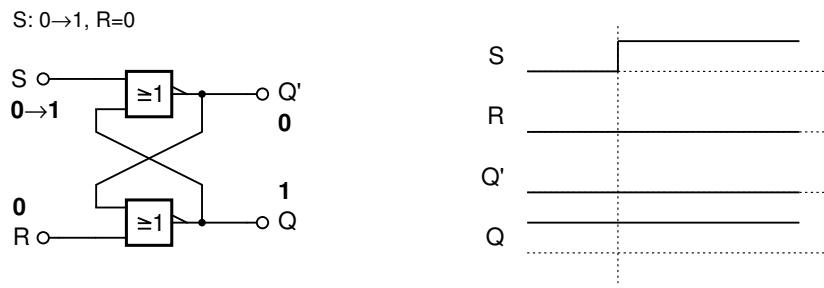


Abbildung 5.11: RS-Latch: S=0→1, R=0, Fall2

**RS-Latch: S=1→0, R=0 (Die Ausgänge Q und Q' behalten ihre Werte):**

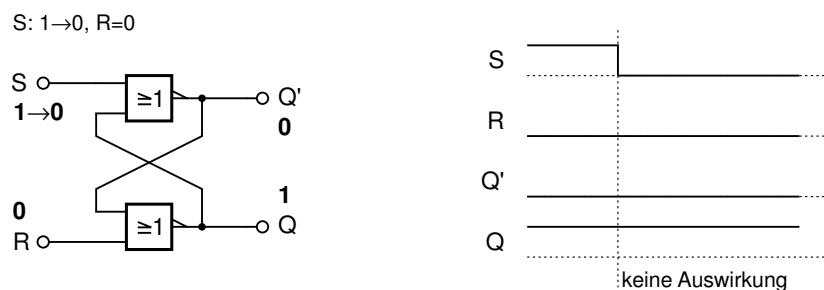


Abbildung 5.12: RS-Latch: S=1→0, R=0

**RS-Latch: S=0, R=0→1 (Bewirkt wieder Veränderung der Ausgabewerte):**

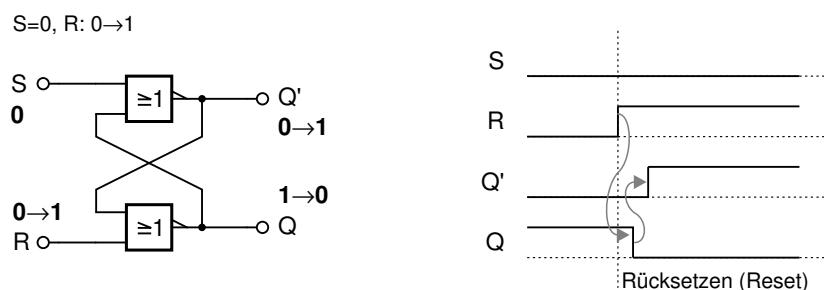


Abbildung 5.13: RS-Latch: S=0, R=0→1

**RS-Latch: S=0, R=1→0 (Ausgänge Q und Q' behalten ihre Werte):**

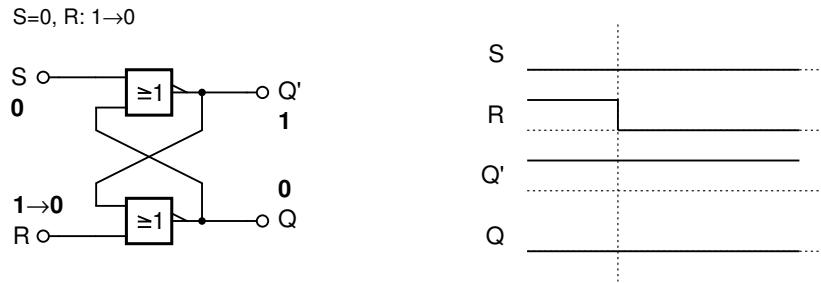


Abbildung 5.14: RS-Latch: S=0, R=1→0

**RS-Latch: S=1, R=1 ('verbotene' Eingangswerte):**

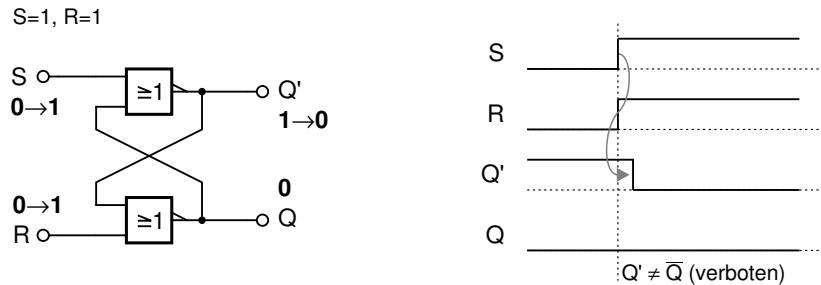


Abbildung 5.15: RS-Latch: S=1, R=1 ('verbotene' Eingangswerte)

**RS-Latch: S=1, R=1→0 (kein Problem):**

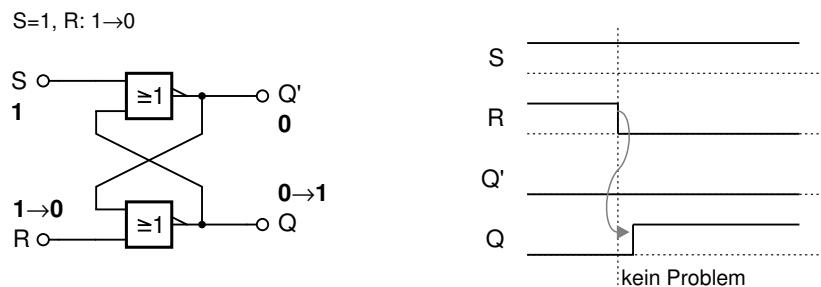


Abbildung 5.16: RS-Latch: S=1, R=1→0 (kein Problem)

**RS-Latch:  $S=1 \rightarrow 0, R=1 \rightarrow 0$  (führt zu instabilen Ausgabewerten):**

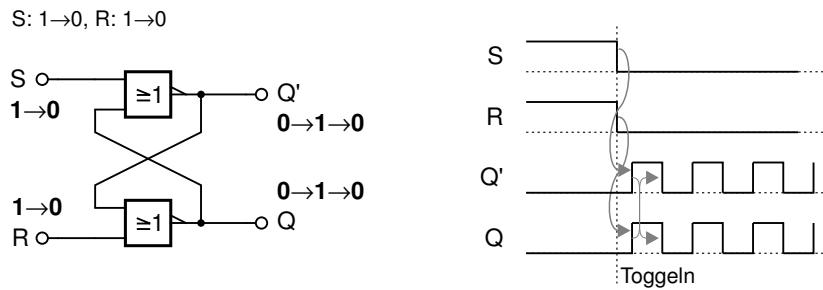


Abbildung 5.17: RS-Latch:  $S=1 \rightarrow 0, R=1 \rightarrow 0$  (instabile Ausgabewerte)

**RS-Latch: Zusammenfassung (gesamt):**

$S$	$R$	$Q_t$	$Q_{t+1}$	Bemerkung
0	0	0	0	stabil, hold
0	0	1	1	stabil, hold
0	1	0	0	Reset
0	1	1	0	Reset
1	0	0	1	Set
1	0	1	1	Set
1	1	0	0	nicht erlaubt
1	1	1	0	nicht erlaubt

Tabelle 5.1: Zusammenfassung RS-Latch

**RS-Latch: Zustandsdiagramm und Schaltsymbol:**

$$S = \{0, 1\}, I = \{00, 01, 10\}, O = \{01, 10\}.$$

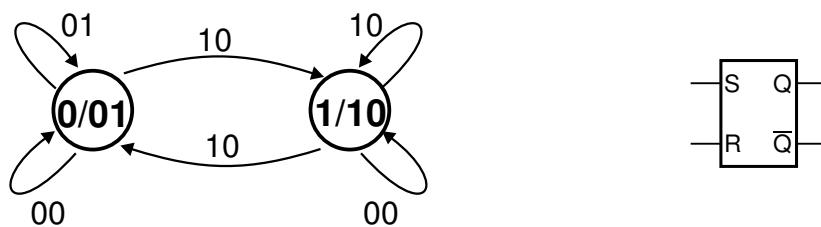


Abbildung 5.18: Zustandsdiagramm und Schaltsymbol eines RS-Latches

Im regulären Betrieb stellen sich die beiden Ausgänge  $Q$  und  $\bar{Q}$  auf zueinander komplementäre Werte ein. Die Funktion der Schaltung erklärt sich aus der Funktion der Einzelgatter, in diesem Fall NAND bzw. NOR und der kreuzweisen Rückkopplung, und wird durch die angegebene Wertetabelle (siehe Tabelle 5.2 – “ $Q_{-1}$ ” bedeutet, daß  $Q$  den gleichen Wert behält wie zum Zeitpunkt  $t-1$ ) beschrieben. Die Funktion der Eingänge kann als Setzen (S) und Rücksetzen (R) die werden.

Aus der Funktion der Einzelgatter resultiert auch die unerlaubte Belegung von  $S$  und  $R$ , in dem  $S$  und  $R$  auf '1' gesetzt sind (bzw.  $\bar{S}$  und  $\bar{R}$  auf low), und die Ausgänge  $Q$  und  $\bar{Q}$  nicht mehr zueinander komplementär sind. Werden von dieser 'verbotenen' Eingangsbelegung beide Eingänge  $S$  und  $R$  gleichzeitig auf '0' gesetzt, kommt es zu einem undefinierten Verhalten des Flipflops. Im Zeitdiagramm (Abbildung 5.19) ist dies durch die Linie zwischen den Werten '1' und '0' dargestellt.

$S$	$R$	$Q$	$\bar{Q}$	$\bar{S}$	$\bar{R}$	$Q$	$\bar{Q}$
0	0	$Q_{-1}$	$\bar{Q}_{-1}$	0	0	1	1
0	1	0	1	0	1	1	0
1	0	1	0	1	0	0	1
1	1	0	0	1	1	$Q_{-1}$	$\bar{Q}_{-1}$

Tabelle 5.2: Wertetabelle des RS-Latch (links: Realisierung mit NOR-Gatter, rechts: Realisierung mit NAND-Gatter)

Das Zeitdiagramm (siehe Abbildung 5.19) wurde aus Gründen des besseren Verständnisses ohne Verzögerungszeiten und mit idealen Flanken dargestellt.

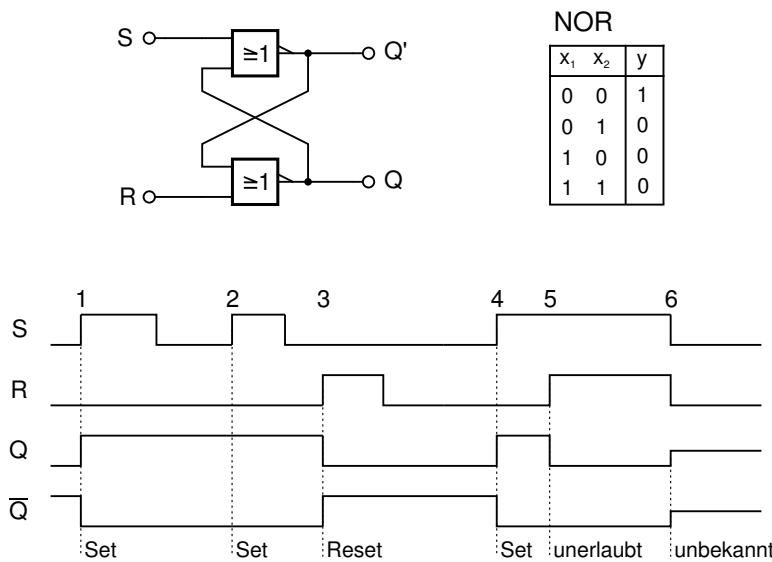


Abbildung 5.19: Zeitdiagramm - RS-Latch (NOR-Gatter)

### 5.2.2 RS-Latch mit Enable

Dieses Latch (siehe Abbildung 5.20) ist ein um den Eingang *Enable* erweitertes RS-Latch. Mit dem Eingang *Enable* ( $E$ ) kann zwischen den zwei Betriebsarten transparent und speichern umgeschaltet werden. Um das Latch transparent zu machen, muß der Eingang  $E$  auf den '1' gesetzt werden. Jede Änderung der Eingänge  $R$  und  $S$  wird direkt am Ausgang  $Q$  übernommen.

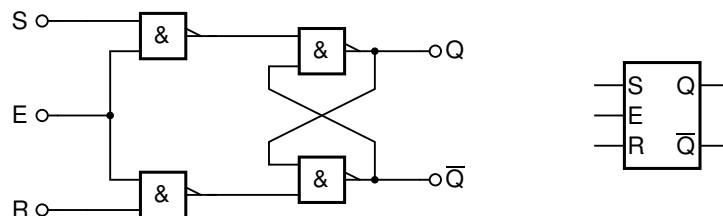


Abbildung 5.20: Schaltung und Schaltsymbol - RS-Latch mit Enable

Das RS-Latch verhält sich wie eine Leitung mit Verzögerung. Im speichernden Zustand, wenn Enable auf '0' gesetzt ist, wird die Eingabe nicht mehr am Ausgang übernommen, der Pegel am Ausgang bleibt unverändert, wird gespeichert (siehe Tabelle 5.3 – “-“ steht für einen beliebigen Wert).

E	S	R	Q	$\bar{Q}$	
0	-	-	$Q_{-1}$	$\bar{Q}_{-1}$	hold
1	0	0	$Q_{-1}$	$\bar{Q}_{-1}$	hold
1	0	1	0	1	reset
1	1	0	1	0	set
1	1	1	1	1	nicht erlaubt

Tabelle 5.3: Wahrheitstabelle - RS-Latch mit Enable

### 5.2.3 D-Latch

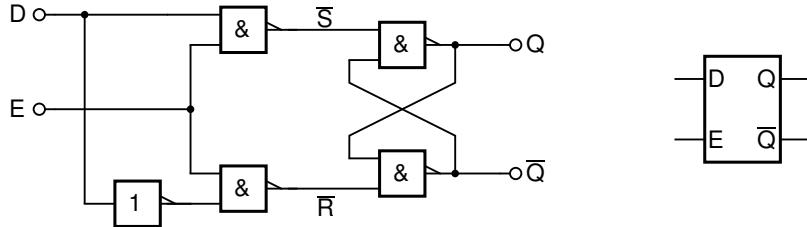


Abbildung 5.21: Schaltung und Schaltsymbol - D-Latch

E	D	Q	$\bar{Q}$
0	0	$Q_{-1}$	$\bar{Q}_{-1}$
0	1	$Q_{-1}$	$\bar{Q}_{-1}$
1	0	0	1
1	1	1	0

Tabelle 5.4: Wahrheitstabelle - D-Latch

Das D-Latch (siehe Abbildung 5.21) baut auf der Realisierung des RS-Latch mit Enable-Eingang auf (siehe Abbildung 5.20). Der Vorteil liegt darin, daß der Eingang  $R$  aus dem Eingang  $S$  durch Negation gewonnen wird. Dadurch kann es nicht zum unerlaubten Belegung von  $R$  und  $S$  kommen. Der Eingang wird als Data (D) bezeichnet. Tabelle 5.4 beschreibt die Funktion.

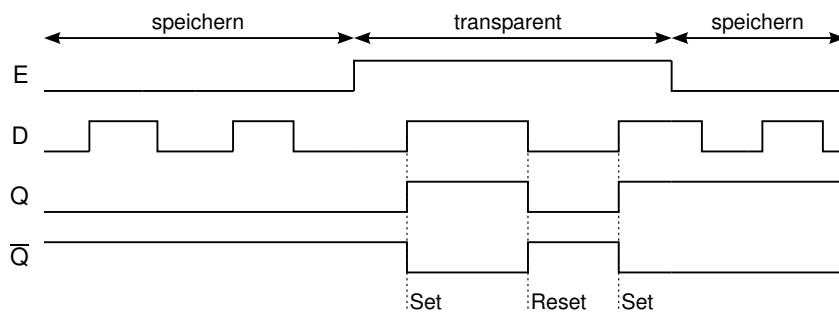


Abbildung 5.22: Zeitdiagramm - D-Latch

Auch das Zeitdiagramm (siehe Abbildung 5.22) zeigt, dass erst durch '1' am Eingang E der Wert von D übernommen wird. Die Ausgänge ändern sich nur im Zustand *transparent*.

### 5.2.4 Master-Slave-D-Flip-Flop

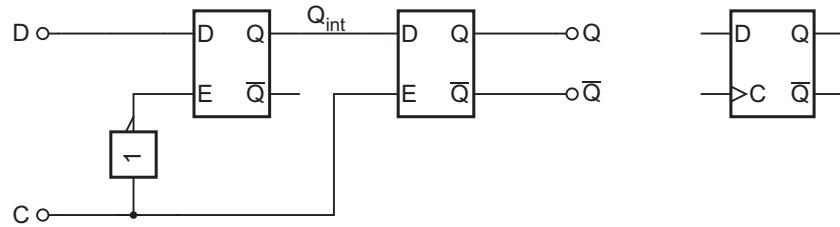


Abbildung 5.23: Schaltung und Schaltsymbol - D-FF

Analog zum D-Latch können auch beim D-Flip-Flop die unerlaubten Eingaben verhindert werden. Beim D-Flip-Flop (siehe Abbildung 5.23) wird die Eingabe D nur zur steigenden Flanke am Ausgang übernommen. Während der '0'-Phase des Taktsignals ist das erste D-Latch transparent, das Eingangssignal D liegt also am Ausgang von Latch 1 an. Bei steigender Flanke wird der Zustand am Ausgang des ersten Latches gespeichert und am zweiten D-Latch durchgeschaltet (siehe Tabelle 5.5 und Abbildung 5.24).

D	C	$Q$	$\bar{Q}$
-	0	$Q_{-1}$	$\bar{Q}_{-1}$
-	1	$Q_{-1}$	$\bar{Q}_{-1}$
0	↑	0	1
1	↑	1	0

Tabelle 5.5: Wahrheitstabelle - MS-D-FF

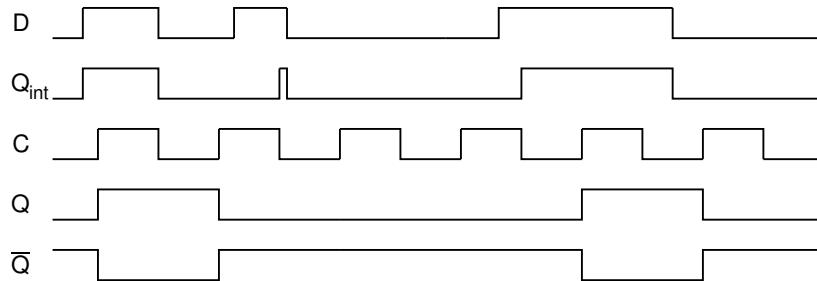


Abbildung 5.24: Zeitdiagramm - MS-D-FF

### 5.2.5 JK-Flipflop

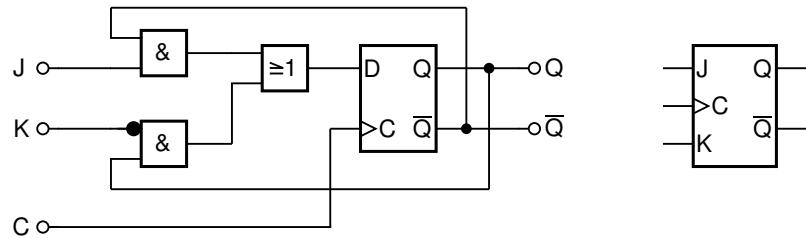


Abbildung 5.25: Flankengesteuertes JK-Flipflop (Schaltung und Schaltsymbol)

J	K	C	Q	$Q^*$
0	0	↑	0	0
0	0	↑	1	1
0	1	↑	1	0
1	0	↑	0	1
1	1	↑	0	1
1	1	↑	1	0

Tabelle 5.6: Wahrheitstabelle - JK-D-FF

### 5.2.6 Toggle Flipflop

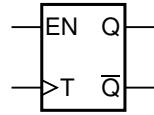


Abbildung 5.26: Schaltsymbol eines Toggle Flipflop



Abbildung 5.27: Realisierung von Toggle Flipflops ohne Enable

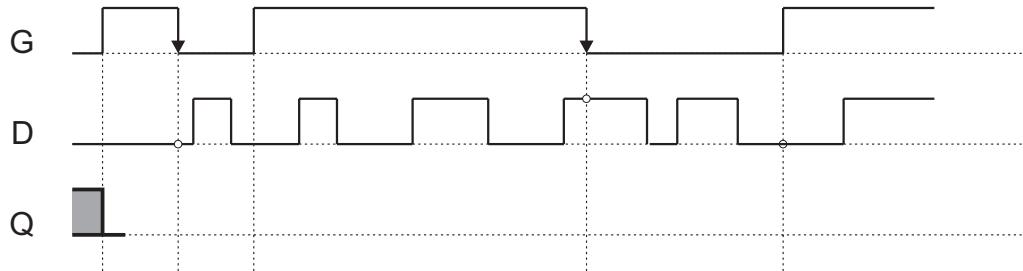


Abbildung 5.28: Realisierung von Toggle Flipflops mit Enable

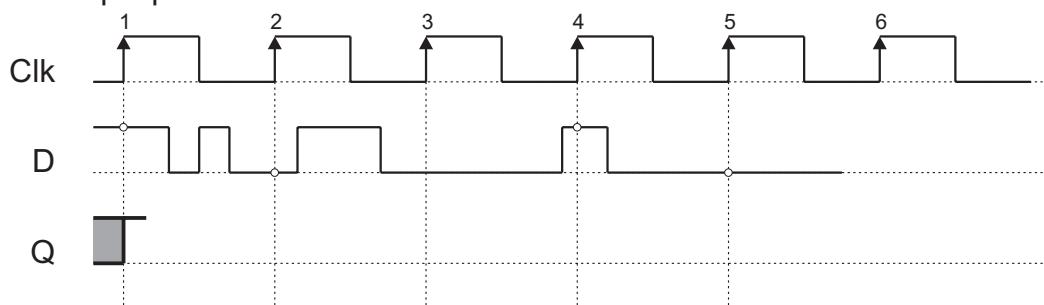
### 5.2.7 Vergleich: D-Latch, D-FF, JK-FF

Aufgabenstellung: Abbildung 5.29 ...

D-Latch



D-Flipflop



JK-Flipflop

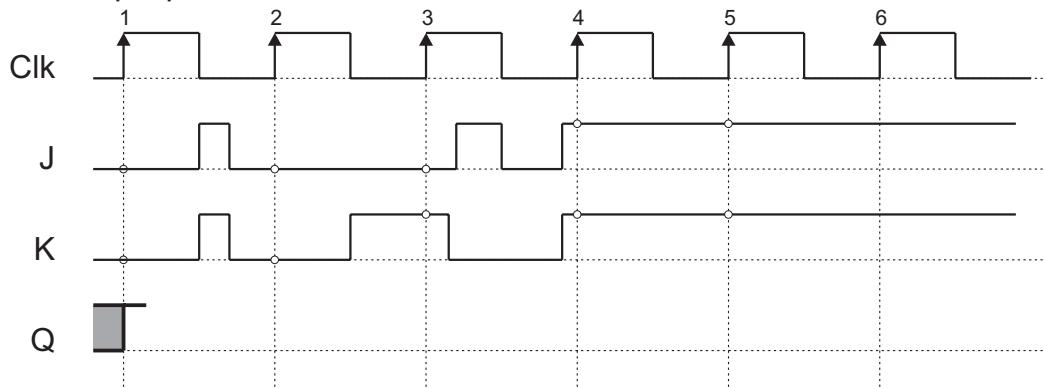
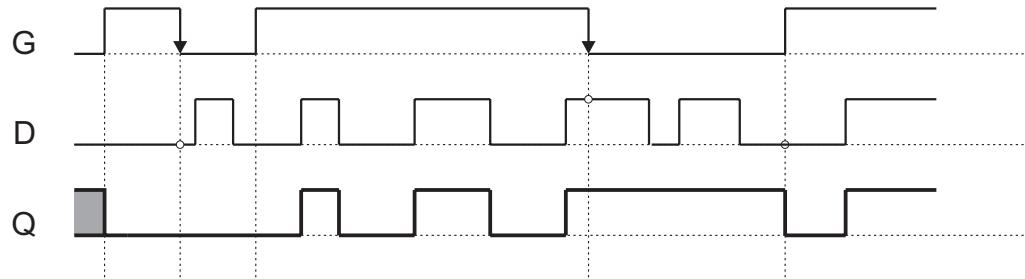


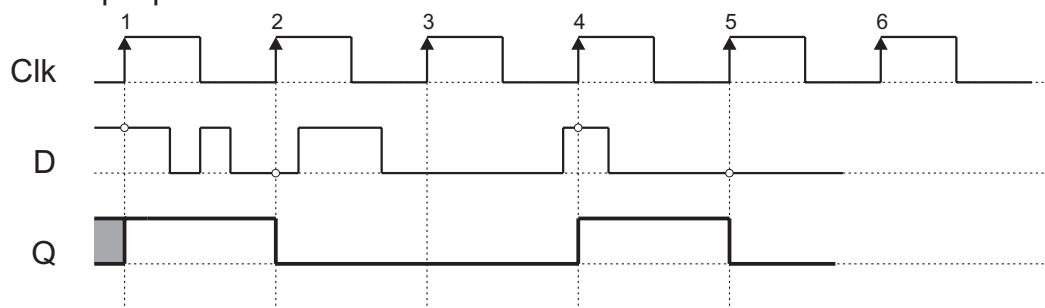
Abbildung 5.29: D-Latch, D-FF, JK-FF

Lösung: Siehe Abbildung 5.30

D-Latch



D-Flipflop



JK-Flipflop

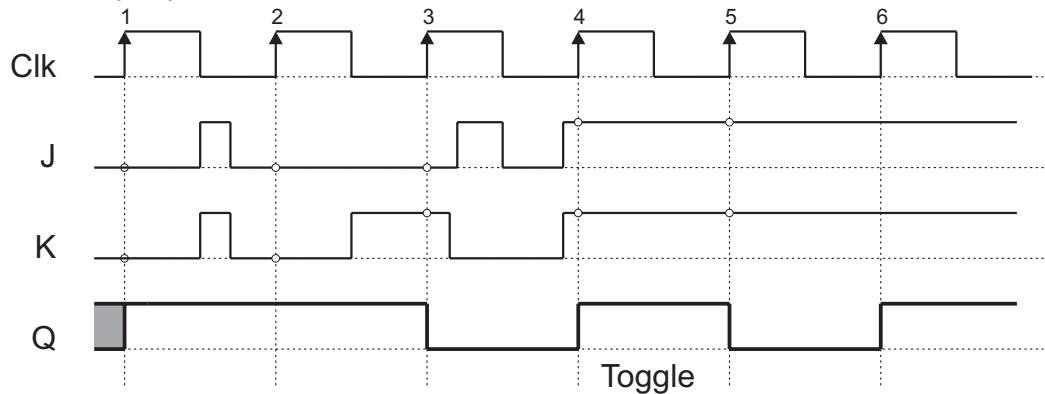
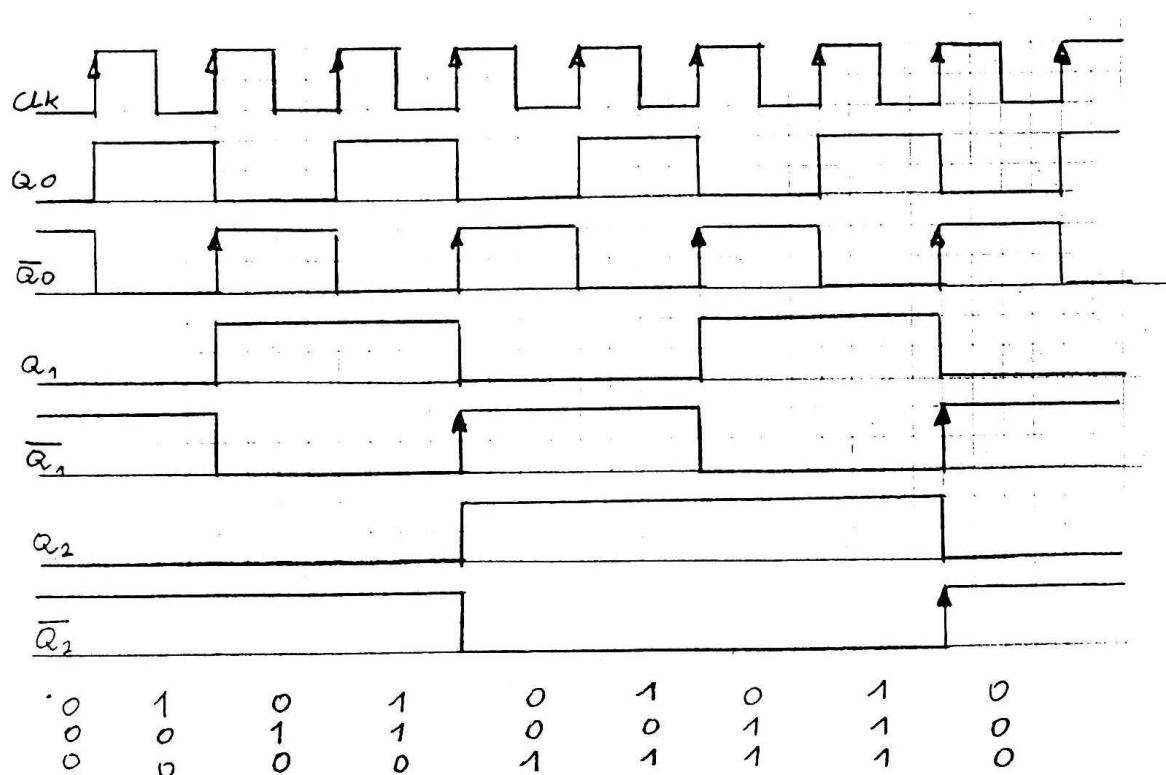
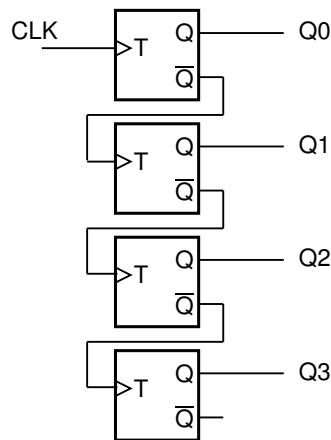


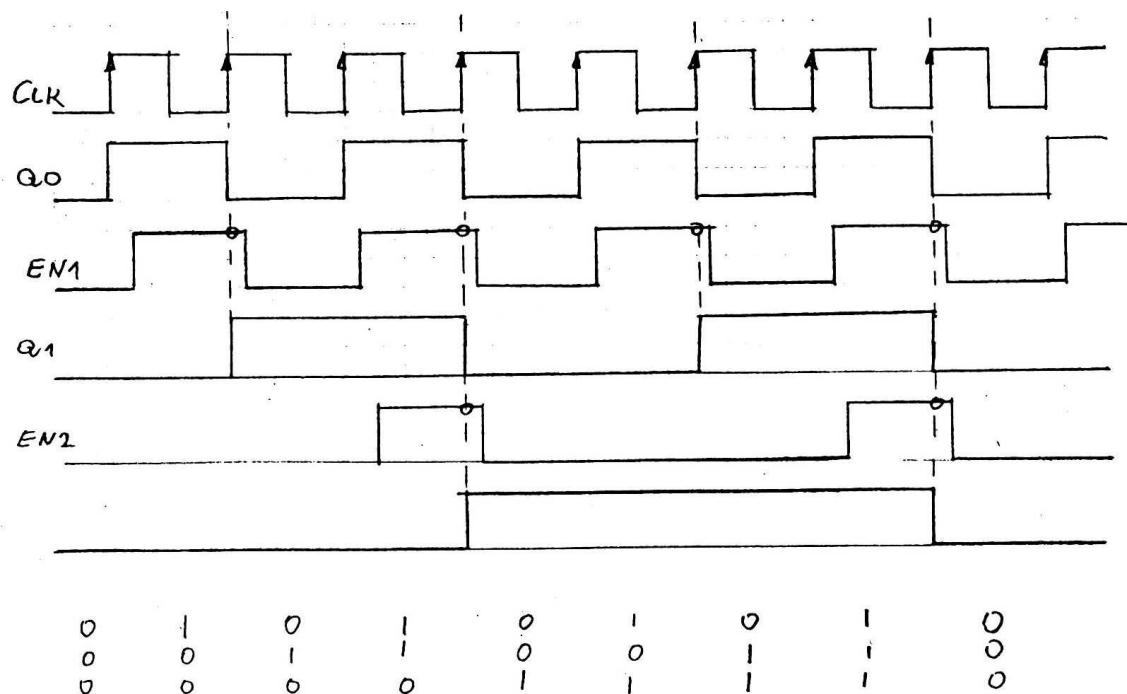
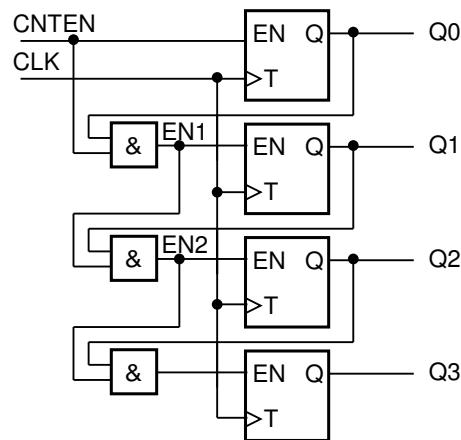
Abbildung 5.30: D-Latch, D-FF, JK-FF (mit Lösung)

## 5.3 Zähler (Counter)

### 5.3.1 Asynchrone (Serielle) Zähler



### 5.3.2 Synchrone (Parallele) Zähler



$$CNTEN = 1$$

### 5.3.3 Setup- und Holdzeit

Das angelegte Eingangssignal muß bereits einige Zeit vor Übernahme am Eingang anliegen. Diese Zeit bezeichnet man als Setupzeit. Andererseits muß ein angelegtes Eingangssignal auch eine bestimmte Zeit lang am Eingang angelegt sein. Die Holdzeit gibt die Zeit an, wie lange sich das Signal nach der Übernahme durch das sequentielle Schaltelement nicht ändern darf. In Abbildung 5.31 werden diese beiden Zeiten anhand eines D-Flip-Flops gezeigt.

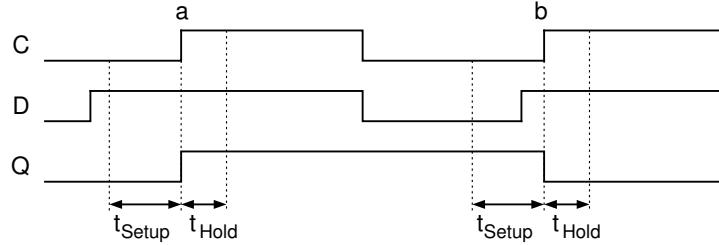


Abbildung 5.31: Setup- und Holdzeit

Im Fall a werden die Zeiten eingehalten, der Eingang D ist konstant. Jedoch im Fall b ändert D seinen Wert während der Setupzeit. Das führt zu undefiniertem Verhalten des Flip-Flops. Typische Werte sind 0.5ns – 5ns für  $t_{Setup}$  und 0ns – 1ns für  $t_{Hold}$ .

## Literatur

- [Böhm90] E. Böhmer. *Elemente der angewandten Elektronik*. F. Vieweg & Sohn, Braunschweig, 1990.
- [Gajski97] D. D. Gajski. *Principles of Digital Design*. Prentice-Hall Internat., Inc., 1997.
- [Hage95] R. Hagelauer. *Skriptum zur Vorlesung Nachrichtentechnik 1*. 1995.
- [Hage96] R. Hagelauer. *Skriptum zur Vorlesung Elektronische Grundlagen der technischen Informatik* 1996.
- [Hage97] R. Hagelauer. *Skriptum zur Vorlesung Nachrichtentechnik 2*. 1997.
- [rech97] P. Rechenberg, G. Pomberger. *Informatik Handbuch*. Hanser Verlag 1997.
- [Volk94] J. Volkert. *Skriptum zur Vorlesung Technische Informatik 1*. 1994.

# Kapitel 6

## Schaltwerks-Entwurf

### 6.1 Beispiele

*Beispiel 1:* Ein endlicher Automat

Zustände:  $Q = 1, 2, 3$

Eingaben:  $A = a, b$

Ausgaben:  $B = u, v, w$

Zustandüberführung:  $\delta$

Ausgabefunktion:  $\lambda$

Nächster Zustand			Ausgabe		
$q_i$	a	$q_{i+1}$	$q_i$	a	b
1	1a	2	1	1a	u
1	1b	3	1	1b	v
2	2a	1	2	2a	w
2	2b	2	2	2b	w
3	3a	3	3	3a	u
3	3b	2	3	3b	v

⇒ Daraus ergibt sich Zustandsüberführungs- ( $\delta$ ) und Ausgabefunktion ( $\lambda$ ).

$\delta$	a	b	$\lambda$	a	b
1	2	3	1	u	v
2	1	3	2	w	w
3	3	2	3	u	v

**Beispiel 2:** Mealy-Automat

Gesucht ist ein endlicher Automat der in einer beliebig langen Folge, bestehend aus a und b, jeweils das Auftreten von ... anzeigt.

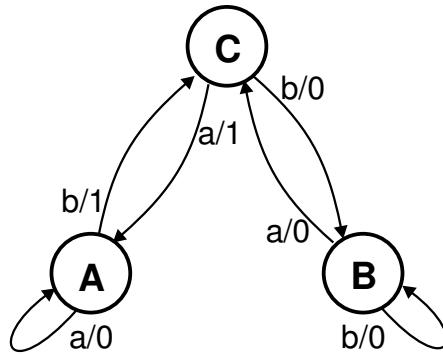


Abbildung 6.1: Beispiel Mealy

Die Ausgabe ist abhängig vom Zustand und der Eingabe!

Zustand	Folgezustand		Ausgabe	
	a	b	a	b
A	B	C	1	0
B	B	A	0	1
C	A	C	0	0

Input/Output-Experiment

Mögliche Eingabe:  $x = abbaba$

Anfangszustand: A

t	0	1	2	3	4	5	
Zustand	A	B	A	C	A	C	A
Eingabe	a	b	b	a	b	a	
Ausgabe	1	1	0	0	0	0	
Folgezustand	B	A	C	A	C	A	

Endzustand: A

## 6.2 Beispiel Roboter

### 6.2.1 Codierung von A,B,Q

**Umschreiben der Tabellen:**

$$S = (A_s, B_s, Q_s, \delta_s, \lambda_s)$$

$$A_s = B = 0, 1$$

$$B_s = B^2 = 00, 01, 10, 11$$

$$Q_s = B^2 = 00, 01, 10, 11$$

wobei:

$$u : 00$$

$$v : 01$$

w : 11 **Achtung!** (führt zu einfacheren Schaltfunktionen)

x : 10 **Achtung!** (führt zu einfacheren Schaltfunktionen)

und

$$G : 00$$

$$L : 10$$

$$R : 01$$

### 6.2.2 Wahrheitstabellen für $\delta$ und $\lambda$

Dabei wird die Codierung von A, B und Q verwendet.

### 6.2.3 Schaltwerk

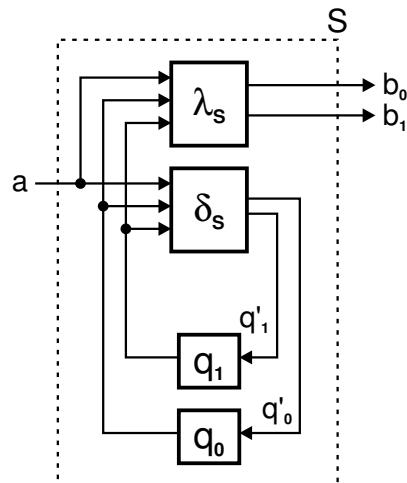


Abbildung 6.2: Schaltwerk

### 6.2.4 Aufspulen von $\delta_S$ und $\lambda_S$ in Schaltfunktionen

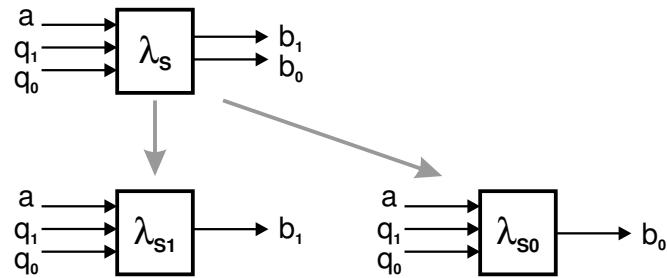


Abbildung 6.3: Ausgabefunktion

$$q'_0 = \overline{q_0} q_1 \bar{a} + q_0 q_1 \bar{a} + q_0 q_1 a + q_0 \overline{q_1} a$$

$$q'_1 = \overline{q_0} \overline{q_1} a + \overline{q_0} q_1 \bar{a} + \overline{q_0} q_1 a + q_0 q_1 \bar{a}$$

**Vereinfachung:**

$$\lambda_{S0} : b_0 = q_0 a$$

$$\lambda_{S1} : b_1 = \overline{q_0} a$$

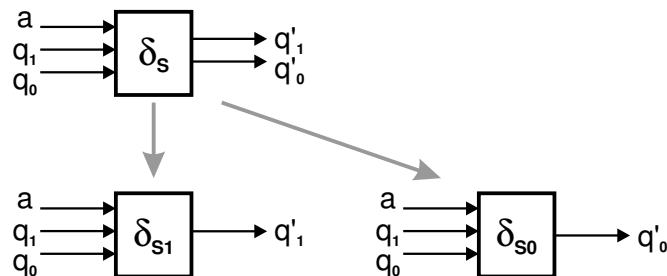


Abbildung 6.4: Überführungsfunction

$$b_0 = q_0 q_1 a + q_0 \overline{q_1} a$$

$$b_1 = \overline{q_0} \overline{q_1} a + \overline{q_0} q_1 a$$

**Vereinfachung:**

$$\delta_{S0} : q'_0 = q_0 \bar{a} + \overline{q_1} a$$

$$\delta_{S1} : q'_1 = q_0 \bar{a} + q_1 a$$

## Vereinfachung mit Hilfe von KV-Diagrammen

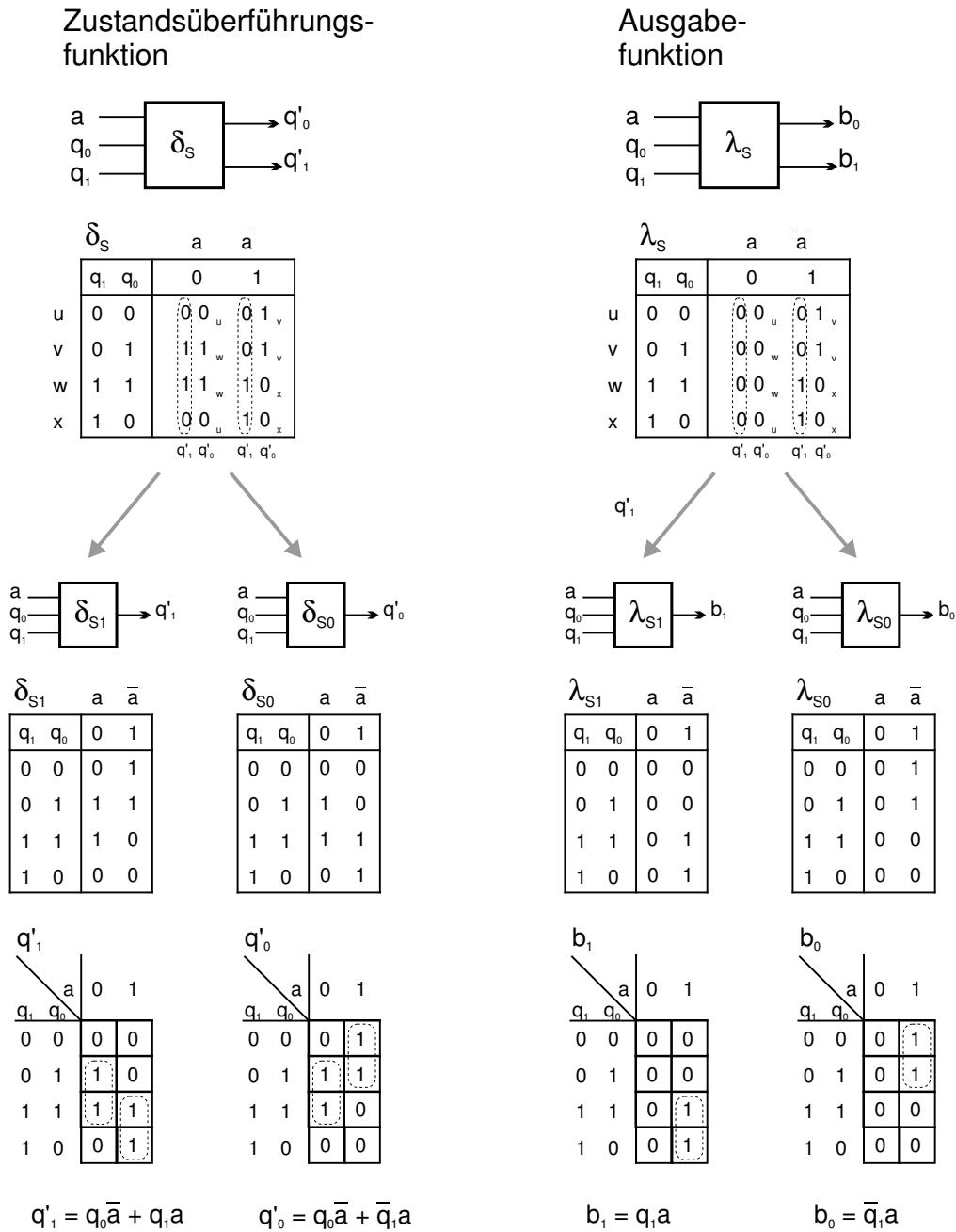


Abbildung 6.5: Vereinfachung der Robotersteuerung mit Hilfe von KV

### 6.2.5 Schaltwerk (Gatterebene)

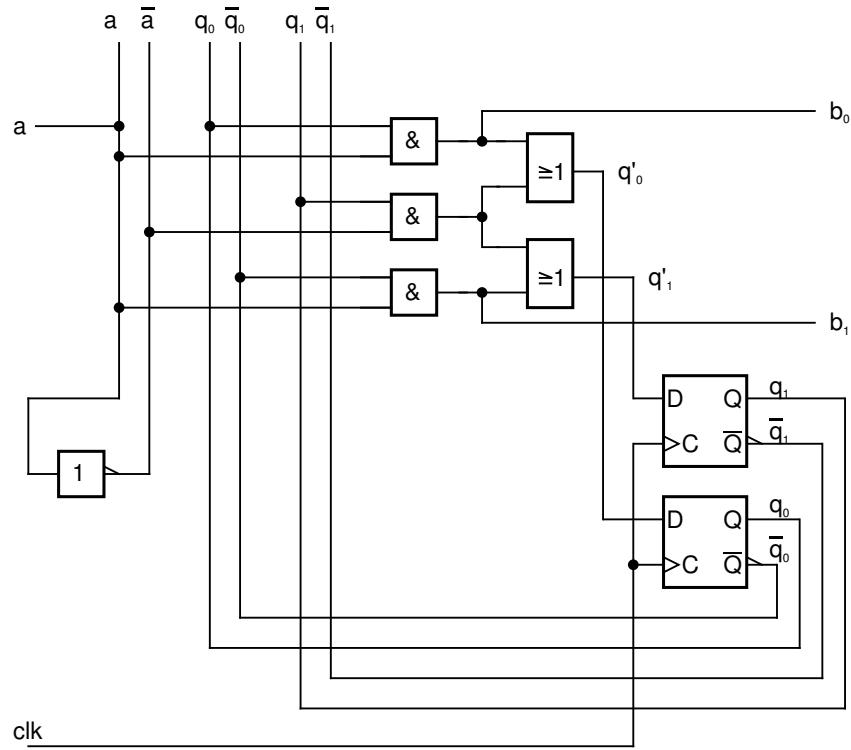


Abbildung 6.6: Schaltwerk für Robotersteuerung in Gatterebene

### 6.3 Weitere Beispiele

#### *Beispiel 3: Parkscheinautomat*

Stell Dir einen Parkscheinautomaten vor

- Du drückst einen Knopf und bekommst einen Parkschein
- Du wirfst ATS 5,- ein und bekommst denselben Parkschein
- Du wirfst ATS 5,- und nochmal ATS 5,- ein und bekommst wieder denselben Parkschein
- Du veränderst durch deine Handlung den Zustand des Automaten und dieser soll/muß unterschiedliche Reaktionen zeigen.

#### *Beispiel 4: Mensch mit Wecker*

schläft/ist wach (zwei Zustände)

Zustandsänderungen durch Wecker bedingt Unmutsäußerung.

Schalter — Wecker läutet / schimpfen — wach sein

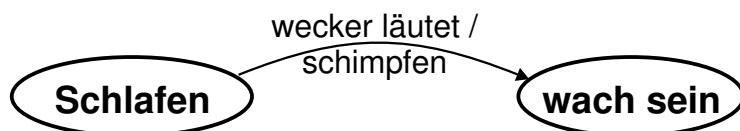


Abbildung 6.7: Mensch

#### *Beispiel 5: Zähler*

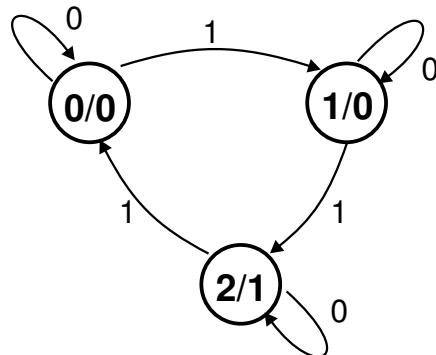


Abbildung 6.8: Ein Zähler in Moore-FSM

**Beispiel 6:** Modulo-5-Zähler

...

**Beispiel 7:** Erkennen von '011'

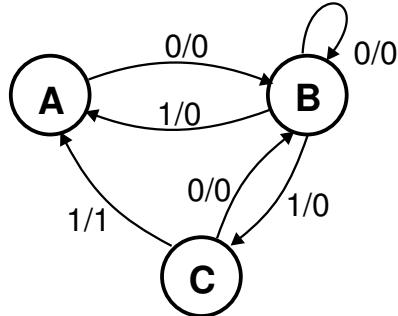


Abbildung 6.9: Erkennen von '011'

**Beispiel 8:** Erkennung von bbbb (auch ineinander verzahnt)

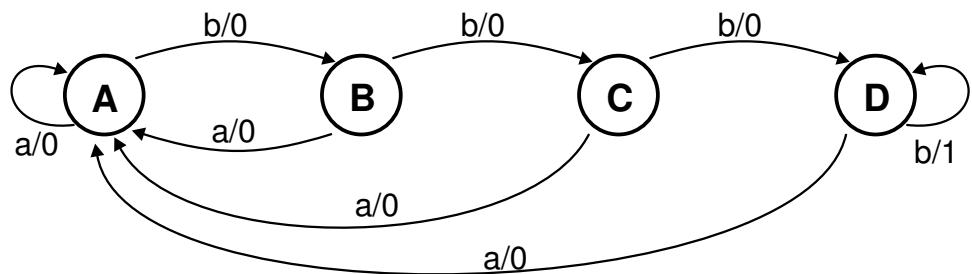


Abbildung 6.10: Beispiel: Erkennen von 'bbbb' (Mealy-FSM)

**Beispiel 9:** Erkennung von aba (auch überlappend)

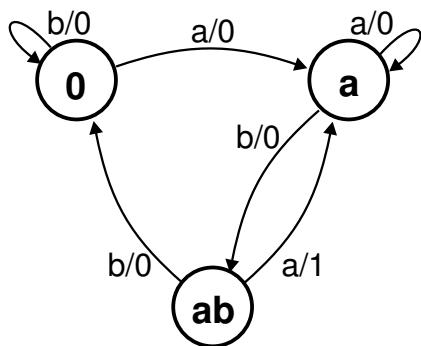


Abbildung 6.11: Beispiel: Erkennen von 'aba' (Mealy-FSM)

**Überlappendes Auftreten erkennen:** ababa

**Beispiel 10: Input/Output-Experiment****Eingabe:**  $x = abba$ **Anfangszustand:** 1

$t$	0	1	2	3	
q Zustand	1	2	3	2	1
a Eingabe	a	b	b	a	
b Ausgabe	u	w	v	w	

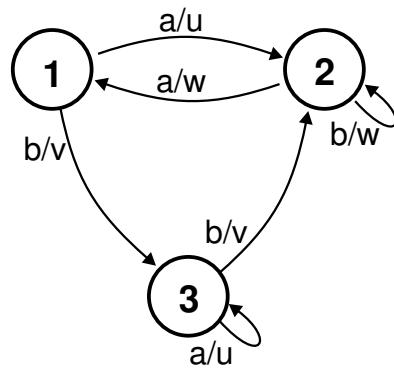
**Endzustand:** 1

Abbildung 6.12: Beispiel: x

