

Prof. Dr.-Ing. Jürgen Roßmann

Grundgebiete der Informatik II

Prinzipien des Digitalrechners



Das vorliegende Skriptum zur Vorlesung *Prinzipien des Digitalrechners* behandelt Grundlagen des Aufbaus und der Funktion digitaler Rechenanlagen.

Es entspricht dem Charakter dieses Skriptums als Lehrunterlage im Grundstudium, dass der betrachtete Stoff in vielen anderen Texten ebenfalls behandelt wird. Unterschiede ergeben sich bei der Setzung von Schwerpunkten. Als wesentliche Quellen werden die im Literaturverzeichnis aufgeführten Lehrbücher genannt, deren Studium zur Vertiefung von Teilespekten empfohlen wird. Weiterhin wurde verschiedenen Firmenunterlagen Information entnommen. Auf ausführliche Zitate im Text wird im Hinblick auf Kürze und Lesbarkeit verzichtet.

Das Skriptum wurde in seiner ursprünglichen Form von Prof. Dr.-Ing. K.-F. Kraiss und seinen Mitarbeitern zusammengestellt. In der aktuellen Version wurden die sich „nicht ändernden Wahrheiten“ unverändert übernommen. Verschiedene Kapitel wurden aktualisiert, um den aktuellen Stand der äußerst rasanten Entwicklungen auf dem Gebiet der Digitalrechner zu reflektieren. Insbesondere das Kapitel über die Assembler-Programmierung erfuhr eine grundlegende Überarbeitung.

Prof. Dr.-Ing. J. Roßmann

und

Mitarbeiter



Inhaltsverzeichnis

1	Historischer Überblick	1
2	Aufbau und Funktion eines Digitalrechners	11
2.1	Aufbau eines Arbeitsplatzrechners	11
2.2	Kennwerte eines Digitalrechners	13
3	Informationsdarstellung und Codierung	15
3.1	Codierung	15
3.2	Informationsgehalt einer Nachricht	17
3.3	Wichtige Codes	20
3.3.1	Optimalcodes (Codeworte unterschiedlicher Länge)	20
3.3.2	Block-/Wortcodes (Codeworte fester Länge)	22
3.4	Erkennung und Korrektur von Übertragungsfehlern	28
3.4.1	Zur Rolle der Hamming-Distanz	28
3.4.2	Quer- und Längsparitätsprüfung	30
4	Zahlendarstellung	32
4.1	Polyadische Zahlensysteme	32
4.2	Umwandlung in Zahlensysteme mit anderer Basis	35
4.2.1	Umwandlung in Dezimalzahlen	35
4.2.2	Umwandlung von Dezimalzahlen	35
4.2.3	Dual-, Oktal- und Hexadezimalzahlen	37
4.3	Zahlendarstellung im Digitalrechner	37
4.3.1	Darstellung ganzer Zahlen	37
4.3.2	Festkomma-Darstellung	41
4.3.3	Gleitkomma-Darstellung	42
5	Schaltungslogik	49
5.1	Zwecke und Ziele	49
5.2	Boolesche Algebra	49
5.3	Beispiele Boolescher Algebren	50
5.3.1	Mengenalgebra	50
5.3.2	Schaltalgebra	52
5.4	Boolesche Funktionen	54
5.4.1	Darstellung von Booleschen Funktionen	55
5.4.2	Minimierung Boolescher Funktionen	60
5.4.3	Behandlung unvollständig definierter Schaltfunktionen	64

6 Logische Schaltungen	69
6.1 Technische Realisierung Logischer Funktionen	69
6.2 Standard-Schaltnetze	69
6.2.1 Addierer	69
6.2.2 Schaltketten	74
6.2.3 Codeumsetzer	79
6.2.4 Datenwähler	83
6.2.5 Kreuzschienenverteiler (crossbar switch)	86
6.3 Speicherglieder	87
6.3.1 RS-Flipflop (Latch)	87
6.3.2 RS-Flipflop mit Zustandssteuerung	87
6.3.3 D-Flipflop mit Zustandssteuerung	89
6.3.4 RS-Flipflop mit Zwei-Zustandssteuerung	90
6.3.5 J-K-Flipflop	90
6.3.6 Zähler	93
6.3.7 Schieberegister	94
6.4 Programmierbare Logik	96
6.4.1 PROM	97
6.4.2 PAL-Bausteine (Programmable Array Logic)	99
6.4.3 (Field) Programmable Logic Array (FPLA)	100
6.4.4 Makrozellen (Programmierbare IC-Bausteine):	101
7 Automaten	102
7.1 Einführung	102
7.2 Das Quintupel des Automaten	103
7.3 Darstellungsweisen von Automaten	105
7.3.1 Automatengraphen	105
7.3.2 Automatentabellen	105
7.3.3 Automatenband	105
7.4 Automatentypen	106
7.5 Umwandlung zwischen Moore- und Mealy-Automat	109
7.6 Äquivalenz und Zustandsreduktion	111
7.6.1 Äquivalente Zustände	111
7.7 Technische Realisierung von Automaten	114
7.7.1 Realisierung des Modellautomaten als Mealy-Automat	115
8 Aufbau und Funktion einer Zentraleinheit	120
8.1 Der Von-Neumann-Rechner	120
8.2 Rechenwerk	123
8.3 Steuerwerk (Control Unit)	130

Inhaltsverzeichnis

8.4	Mikroprogrammierung	132
8.5	Kern der Zentraleinheit (CPU)	133
8.6	Abweichungen vom von Neumann-Konzept	133
8.6.1	Harvard-Architektur	133
8.6.2	Pipelining	134
8.6.3	Sprungvorhersage (Branch Prediction)	136
8.6.4	Parallelrechner-Architekturen	136
8.6.5	Rechenwerke mit Vektoreinheit	137
8.7	Gleitkomma-Koprozessoren	137
8.8	Superskalarität	138
8.9	Register Renaming	140
8.10	CISC- versus RISC-Maschinen	141
8.11	VLIW-Prozessoren	142
8.12	Multithreading-Architekturen	142
9	Maschinensprache und Assembler	145
9.1	Zugrunde gelegte Hardware	146
9.2	Register	148
9.2.1	Das Statusregister	149
9.2.2	General Purpose Working Register	149
9.2.3	Stack-Pointer	150
9.3	Assembler	151
9.3.1	Assemblerbefehle	152
9.3.2	Ablauf eines Assemblerprogramms	152
9.4	Untergliederung der Assemblerbefehle	153
9.4.1	Datentransferbefehle	154
9.4.2	Arithmetische Operationen	156
9.4.3	Logische Operationen	157
9.4.4	Vergleichsbefehle (CP und CPI)	158
9.4.5	Bitbefehle (SBR, CBR, BSET und BCLR)	158
9.4.6	Verzweigebefehle	159
9.4.7	Programmierung von Schleifen	161
9.4.8	Rekursion	161
10	Organisation der Ein-/Ausgabe	164
10.1	Ein-/Ausgabe-Hardware	164
10.1.1	Grafikkarten	164
10.1.2	Drucker	168
10.1.3	Maus und Trackball	172
10.2	Busse und Schnittstellen	174

10.2.1	Einführung	174
10.2.2	Grundlagen der Bussysteme	175
10.2.3	Beispiele für Peripherie-Busse	182
10.2.4	Beispiele für E/A-Busse	184
10.2.5	Schnittstellen für Punkt-zu-Punkt Verbindungen	190
10.2.6	Drahtlose Kommunikation	194
10.3	Ein-/Ausgabetechniken	195
10.3.1	Ein-/Ausgabe über den CPU-Bus	195
10.3.2	Ein-/Ausgabe über Direct Memory Access (DMA)	197
10.4	Ein-/Ausgabe von Analogdaten	199
10.4.1	Analogeingabe	199
10.4.2	Analogausgabe	199
11	Spechertechnik	201
11.1	Speichermerkmale	201
11.2	Halbleiterspeicher	202
11.2.1	Halbleiterbauelemente	202
11.2.2	Integrierte Schaltungen	204
11.2.3	Schreib-Lese-Speicher (RAM)	205
11.2.4	Assoziativspeicher	208
11.2.5	Festwertspeicher (ROM)	208
11.3	Magnetische Massenspeicher	210
11.4	Optische Massenspeicher	213
11.5	Magneto-optische Massenspeicher	217
11.6	Speicherorganisation	217
11.6.1	Cache-Speicher	218
11.6.2	Virtueller Speicher	221
11.6.3	Interleaving	221
12	Rechneraufbau am konkreten Beispiel	223
12.1	Pentium-Familie	223
12.1.1	Übersicht	223
12.1.2	Beispiel: Pentium 4	225
12.2	PowerPC-Familie	227
12.2.1	Übersicht	227
12.2.2	Beispiel: PowerPC 7400	228
12.3	Leistungsbewertung von Rechnersystemen	230
12.3.1	Benchmarkverfahren	230
12.3.2	Die SPEC CPU-Suite	231
12.4	Entwicklungsperspektiven bei Speicherkapazität und Rechengeschwindigkeit	233

Literaturverzeichnis	235
A Übersicht über die Befehle des Atmel AVRmega8	237
B Weiterführende Links zum Thema Assembler	241
Index	242



Kapitel 1

Historischer Überblick

3. Jh.a.C. Abakus (chin. Suan Pan)

Biquinäre Darstellung, basierend auf dem 5-Finger-Abzählsystem des Menschen
(siehe Abb. 1.1)

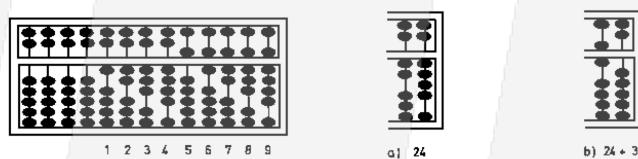


Abbildung 1.1: Rechnen mit dem Abakus

1614 John Napier (1550–1617) entdeckt die Logarithmen.

1623 W. Schickard (1592–1635) entwickelt eine fünfstellige Dezimalrechenmaschine (Abb. 1.3) für die astronomischen Berechnungen Keplers. Sie besteht aus 17 Zahnrädern. Die Unterlagen wurden erst 1957 wiederentdeckt. Das Zählwerk mit Übertrag (Abb. 1.2) wird noch heute benutzt.

1641 Blaise Pascal (1623–62) entwickelt eine Additionsmaschine für 8 Dezimalstellen.

1650 Partridge: Erfindung des Rechenschiebers

1673 Gottfried Wilhelm Leibnitz (1646–1716) entwickelt eine 4 Species-Maschine (Abb. 1.4) mit Staffelwalze (verschiebbare Zahnräder).

1679 Leibnitz verwendet binäre Zahlen.

18. Jh. J.O. de la Mettrie (1709–51) veröffentlicht „L' homme machine“. Zur gleichen Zeit werden verschiedene Androiden (schreibende, zeichnende, Klavier und Schach spielende Puppen) hergestellt.

1728 Falcon entwickelt Webstuhlsteuerungen über Lochstreifen.

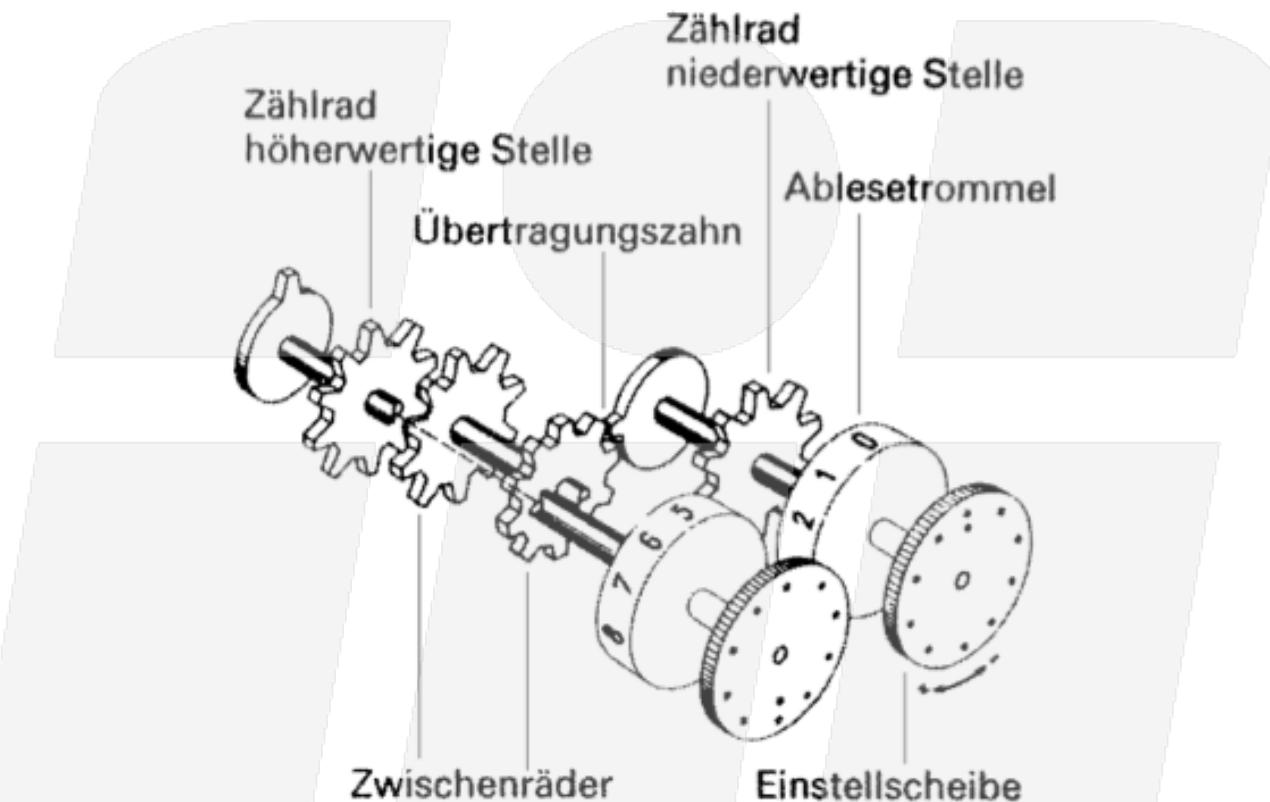


Abbildung 1.2: Zählwerk mit Dezimalübertrag

- 1805 J.M. Jacquard (1752–1834) verbessert die Webstuhlsteuerungen von Falcon.
- 1833 Charles Babbage (1792–1871) entwickelt die „Difference Engine“ sowie das Konzept der „Analytical Engine“. Die Analytical Engine sollte für ballistische Berechnungen eingesetzt werden und über ein Rechenwerk, einen Speicher, eine Programmsteuerung über Lochkarten bzw. -streifen und einen Drucker verfügen. Ihre Realisierung scheitert jedoch an den zu jener Zeit unzureichenden technischen Möglichkeiten.
- 1847 George Boole (1815–1864) begründet die Algebra der Logik.
- 1890 Herrmann Hollerith (1860–1929) entwickelt eine Lochkartenmaschine für die US-Volkszählung. Er gründet die Firma Tabulating Machine Comp., später International Business Machines (IBM).
- 1897 F. Braun entwickelt die Kathodenstrahlröhre.
- ab 1939 W. Schottky leistet theoretische Vorarbeiten zu Halbleitern.
- 1938 Konrad Zuse (1910-1995) baut die Rechenanlage Z1 und ein Jahr später die Z2.

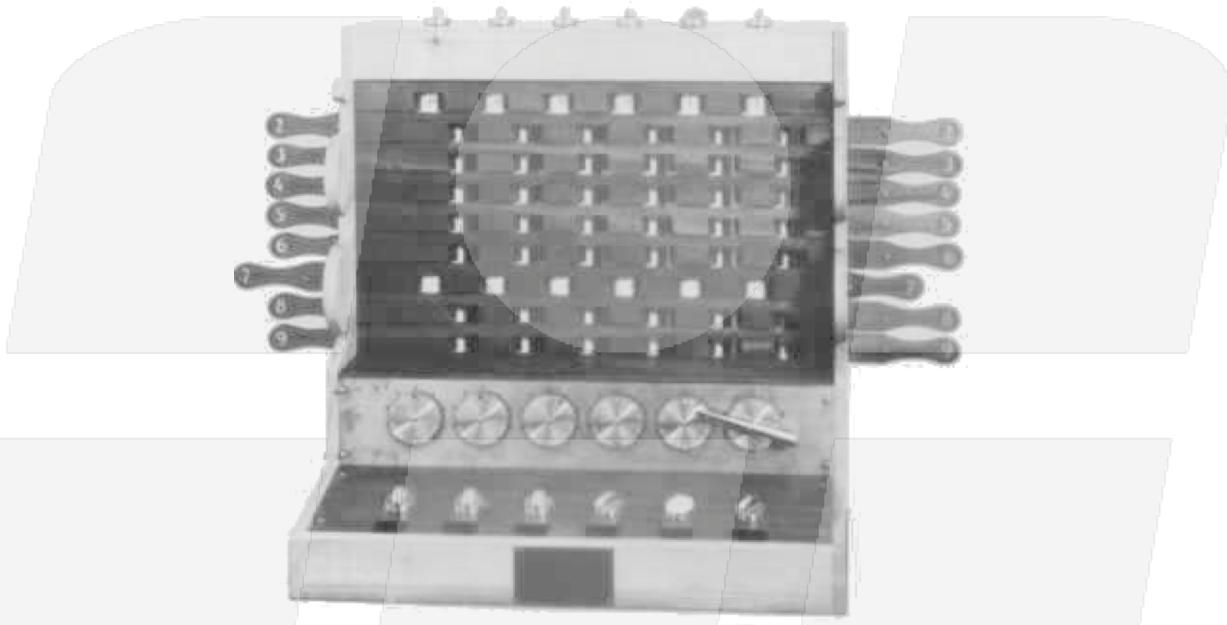


Abbildung 1.3: Schickards Rechenmaschine

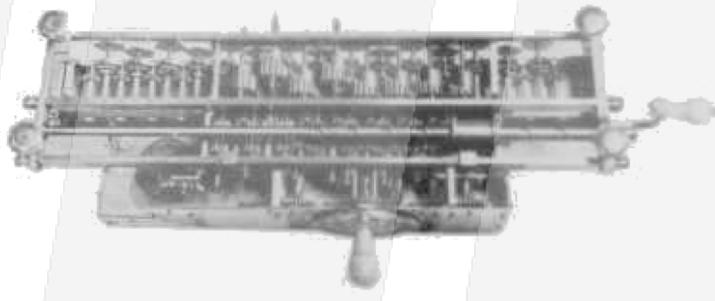


Abbildung 1.4: Rechenmaschine von Leibnitz

1941

Zuse baut die Z3 (siehe Abb. 1.5), eine frei programmierbare Rechenmaschine mit 600 Relais für Logik und 2000 Relais für den Speicher. 64 Binärzahlen mit 22 Stellen (7 Dezimalstellen), Gleitkommadarstellung, 4 Spezies und Radizieren. Addition in 5 ms, Multiplikation in 4–5 s.

1944

Howard Hathaway Aiken (1900–73) entwickelt den ASCC (Automated Sequence Controlled Calculator, Abb. 1.6), auch Harvard Mark 1 genannt. Programmierbar über Stecktafel, relaisgesteuert. 700000 Einzelteile, 15 m lang, 2,5 m hoch, 3,5 Tonnen schwer. Addition in 300 ms, Multiplikation in 2–6 s, Division in 11 s.

Alan Turing (1912–54) baut die Dechiffriermaschine Colossus zur Dechiffrierung von geheimen Nachrichten des deutschen Militärs.

1946

J.P. Eckert und J.W. Mauchly bauen den ersten Röhrenrechner ENIAC (Electronic Numerical Integrator and Computer). Der Rechner besitzt 18000 Röhren, einen Takt

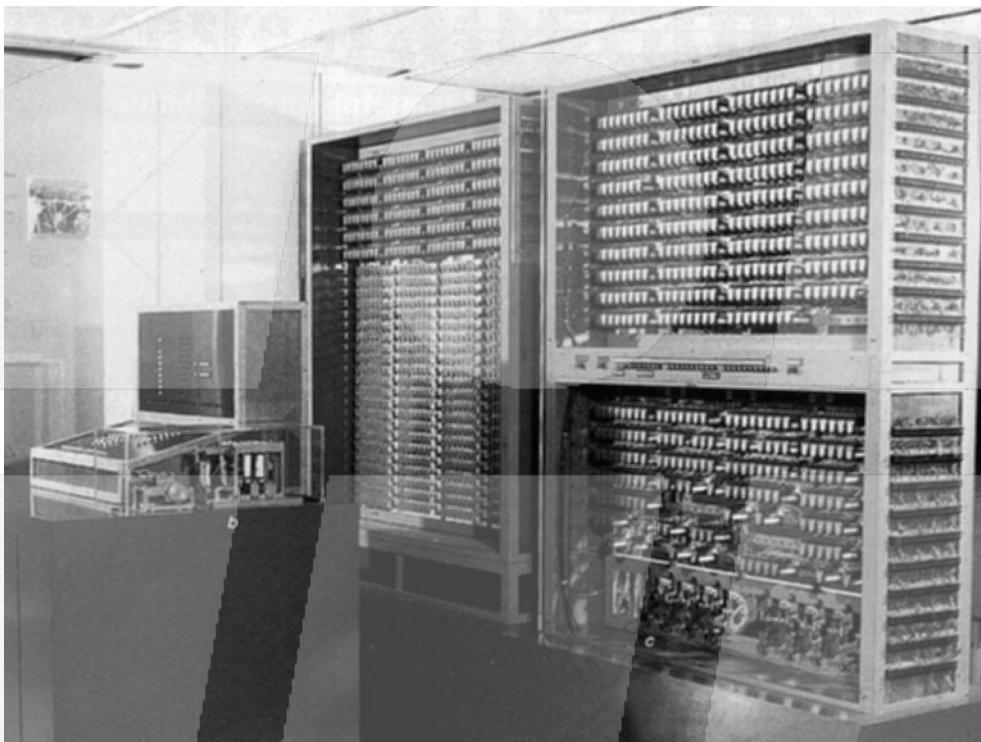


Abbildung 1.5: Rekonstruktion der Z3 von Konrad Zuse



Abbildung 1.6: IBM ASCC (Automatic Sequence Controlled Calculator)

von 10 kHz und arbeitet dezimal. Programmierung über Schalttafel, kein Programm- speicher. Multiplikation in 2,8 ms.

John von Neumann (1910–57) schlägt den EDVAC (Electronic Discrete Variable Automatic Computer) vor, welcher über einen einheitlichen Programm- und Datenspeicher verfügt. Er arbeitet binär. Eine datenabhängige Programmverzweigung ist mög lich. Der Rechner wird 1949 fertiggestellt.

- 1948 In Cambridge, UK wird mit dem EDSAC der erste Rechner mit von Neumann– Architektur fertiggestellt.

- 1945 Einsatz von Analogrechnern, die bis in die späten 70er Jahre populär bleiben. Einsatz zur Simulation, zur Berechnung von Differentialgleichungssystemen und zu Steuerungszwecken („Command and Control“).
- 1948 Claude Shannon entwickelt die Informationstheorie.
Norbert Wiener: „Cybernetics“
W. Shockley, W. Brattain und J. Bardeen erfinden den Transistor.
- 1949 Entwicklung von externen Speichermedien (Magnetband, -platte und -trommel)
- 1956 Der Rechner Z22 (500 Röhren) wird an der RWTH Aachen installiert. Er ist mit 140 kHz getaktet, die Wortlänge beträgt 38 Bits. Addition in 0,6 ms, Multiplikation in 10 ms.
Der erste Transistorrechner TX-0 wird am MIT von Wesley Clarke und Ken Olsen, dem späteren Gründer von DEC, gebaut. Bis zu diesem Zeitpunkt waren Rechner teure Einzelstücke ohne große Stückzahlen. Erst danach erfolgt eine kommerzielle Auswertung und Standardisierung.
- 1958 Frank Rosenblatt entwickelt den Neurocomputer „Mark I Perceptron“ zur Mustererkennung.
- 1958-61 J. Kilby entwickelt die ersten integrierten Schaltungen (Integrated Circuits, IC's).
- 1965 Einsatz von SSI-Chips (Small Scale Integration, weniger als 100 Transistoren je Chip)
- ab 1966 Entwicklung von Mainframes mit Dialogstationen
- 1970 Einsatz von MSI-Chips (Medium Scale Integration, weniger als 1000 Transistoren je Chip)
- 1975 Einsatz von LSI-Chips (Large Scale Integration, weniger als 10000 Transistoren je Chip)
Entwicklung von Vektorrechnern (z. B. Cray 1)
Entwicklung der Mikrocomputer mit 8 Bit-Mikroprozessor (z. B. Apple II)
- 1980 Einsatz von VLSI-Chips (Very Large Scale Integration, mehr als 10000 Transistoren je Chip)
Entwicklung der Mikrocomputer mit 16 Bit-Mikroprozessor (z. B. IBM PC, Apple Macintosh)
- 1985 Rumelhart und Hinton entwickeln den Backpropagation-Algorithmus zum Training mehrschichtiger neuronaler Netze.
- 1986 Es entstehen Thermo-magnetooptische Disks.

- 1987 5. Computergeneration beginnt.
IBM entwickelt eine 3 1/2-Zoll Festplatte mit einer Kapazität von 10 Milliarden Bits.
- 1988 EISA (ein Standardbus, der den ISA-Bus auf 32 Bit erweitert) taucht auf: Kreiert als Antwort auf IBMs hauseigenes Micro-Channel-Interface, dient EISA in erster Linie dazu, ältere ISA (PC-AT)-Erweiterungskarten zu unterstützen.
- 1989 Intels neuer Prozessor, der 80486, verfügt über 1,2 Millionen Transistoren, das sind vier mal mehr als beim Vorgänger. Der mathematische Coprozessor ist ab sofort in den Hauptprozessor integriert. Auch ein 8 KB großer Cache für Daten ist erstmals auf dem 486er integriert (bislang waren diese als separate Bausteine realisiert). Der Prozessor taktet anfangs mit 25 MHz und wird bis 1992 zum 80486DX2 mit 66 MHz und 1994 zum DX4 mit 100 MHz weiterentwickelt.
- 1990 Der Parallelcomputer SUPRENUM ist nun serienreif.
- 1991 Linus Torvalds entwickelt auf der Basis von „UNIX“ das Betriebssystem „LINUX“. IBM brachte den ersten PC auf den Markt, mit einem 8088 Prozessor, der auf 77 MHz getaktet ist.
Eine spezifizierte CPU-Architektur wurde durch ein US-amerikanisches Konsortium der Unternehmen Apple, IBM und Motorola (auch kurz AIM genannt) entwickelt.
Geburt des PCI-Anschlusses: Der Peripheral Component Interconnect (kurz PCI), entwickelt von Intel, erleichtert PC-Herstellern und -Nutzern das Leben beachtlich. PCI (der Vorläufer von PCI Express) erlaubt die automatische Konfiguration von Interfacekarten, verbessert die Geschwindigkeit und erweitert die generelle Kompatibilität.
- 1992 IBM führt das erste Think Pad ein, das sofort zur Designikone wird.
- 1993 Intel überrascht die Konkurrenz: Um den Clonern eins auszuwischen, nennt Intel seinen neuen Chip nicht 80586, sondern Pentium. Auf ihm sind über 3,1 Millionen Transistoren integriert. Der Pentium taktet intern anfangs mit 60 bzw. 66 MHz.
Es werden allgemeine Standards für das "World Wide Web" festgelegt.
- 1994 Die erste Power Macintosh-Reihe wurde im März 1994 eingeführt und stellte für die Anwender einen nahtlosen Übergang auf eine völlig neue Art von Prozessor dar: Die konventionellen Prozessoren der Motorola-68k-Baureihe wurden abgelöst durch den RISC-Prozessor PowerPC 601.
- 1995 Die ersten bedeutenden 3D-Chips: Die ersten 3D-Beschleuniger, die auch wirklich für PC-Gaming genutzt werden konnten - Rendition Vérité 1000 und 3dfx Voodoo - markieren eine neue Ära von Grafikkarten.
Der Pentium Pro wird offiziell vorgestellt. Den Intel-Prozessor der sechsten Generation gibt es mit 150, 166, 180 und 200 MHz interner Taktrate. Die Zahl der Transistoren beträgt 5,5 Millionen.

-
- 1996 Ein japanischer Rechner schafft 1 Gflops (ein Maß für die Leistungsfähigkeit von Computern oder Prozessoren und bezeichnet die Anzahl der Gleitkommazahl-Operationen (Additionen oder Multiplikationen), die von ihnen pro Sekunde ausgeführt werden können).
USB taucht auf: Im Gegensatz zu Serial- und Parallel-Ports, bedeutet USB den Schritt hin zu selbstkonfigurierenden Geräten und macht das Anschließen von Tastaturen, Mäusen und Kameras an den PC zu einem echten Kinderspiel. Zudem entsteht ein komplett neuer Markt für USB-Flash-Speicher.
- 1997 Der Großrechner „BigBlue“ von IBM schlägt den Schachweltmeister Garry Kasparov. „Deep Blue“ war in der Lage 200 Millionen mögliche Schachzüge pro Sekunde zu berechnen und schaffte es so, den damaligen Weltmeister in der ersten Partie zu besiegen.
Der Marktführer Intel präsentiert den Pentium MMX. Die MMX-Technik erweitert die Prozessorarchitektur um 57 neue Befehle für den Grafik-, Video- und Audio-Bereich. Bei herkömmlichen Anwendungen sind die MMX-Prozessoren um 10 bis 15 Prozent schneller. Bei speziell für MMX entwickelten Multimedia-Programmen kann der Leistungsgewinn dagegen bis zu 87 Prozent ausmachen. Die neue Pentium-MMX-CPU taktet zunächst mit 166 oder 200 MHz. Sie besitzt 4,5 Millionen Transistoren.
Intels Pentium II kommt auf den Markt: Der neue Chip mit Taktfrequenzen von 233, 266 und 300 MHz vereint die Vorteile von Pentium Pro und MMX. Er taktet extern mit 66 MHz und verfügt über 7,5 Millionen Transistoren.
- 1998 Die ersten funktionsfähigen CNTFETs (englisch: Carbon Nanotube Field-Effect Transistor) wurden von einer Gruppe der Universität Delft vorgestellt.
- 2000 Der Athlon von AMD übertrifft Intel's Pentiums III Prozessor in Geschwindigkeit und Stabilität. Schnellster Computer der Welt ist der IBM Rechner ASCI White mit 4938 Gflops
- 2001 Intel bringt den Pentium III-Prozessor auf den Markt, der mit einer Taktfrequenz von 1000MHz arbeitet. Im Juni des gleichen Jahres kommt der erste 64-Bit-Prozessor von Intel in die Serienproduktion. Er bekommt den Namen „ITANIUM“. Er ist mit den Taktfrequenzen 733 MHz und 800 MHz erhältlich.
- 2002 Entwicklung von Earth Simulator, der schnellste Supercomputer von 2002 bis 2004. Als Earth Simulator sind zwei japanische Supercomputer bekannt, die auf Vektorprozessoren basieren. Das erste Parallelrechnersystem wurde 2002 am Yokohama Institute for Earth Sciences (YES) von der Firma NEC in Betrieb gesetzt.
- 2003 AMD bringt die erste 64-bit-x86-CPU: AMD kommt Intel zuvor und bringt als erster Konzern die Athlon-64-CPU auf den Markt, die sowohl 64-bit-Register, als auch 64-bit-Speicher in ihrer Architektur bietet. Microsoft startet daraufhin die Entwicklung einer Windows-Version mit 64-bit-Unterstützung - Intel bringt sein eigenes x86-64-Produkt erst zwei Jahre später auf den Markt.

- Das stereoskopische 3D-Display wurde von A.C.T. Kern entwickelt.
- 2006 Intel bringt Core 2 Duo auf den Markt. Core 2 Duo braucht deutlich weniger Energie und produziert weniger Abwärme als AMD-Prozessoren.
- 2007 Das mit Multi-Touch-Steuerung ausgestattete iPhone von Apple verwischt die Grenze zwischen Handy und Computer, die Ära der Smartphones beginnt.
Blue Gene ist ein Projekt zum Entwurf und Bau einer High-End-Computertechnik. Diese soll laut IBM sowohl zur Erforschung der Grenzen des Supercomputing in der Computerarchitektur, zur Entwicklung der für die Programmierung und Kontrolle massiv paralleler Systeme nötigen Software und zur Nutzung von Rechenkraft für ein besseres Verständnis biologischer Prozesse wie etwa der Proteinfaltung dienen.
- 2008 Der IBM Roadrunner-Supercomputer hat die Leistung von 1 PetaFLOPS (Quadrillion Operationen pro Sekunde) übertroffen und ist zum schnellsten Computer der Welt geworden. Das System ging am 31. März 2013 endgültig außer Betrieb. Begründet wurde dieses Vorgehen mit seiner schlechten Energieeffizienz, insbesondere im Vergleich zu anderen Supercomputern.
- 2009 Der Supercomputer Cray XT5 (Jaguar) war das produktivste Computersystem der Welt.
- 2012 Der Titan ist ein Supercomputer von Cray. Mit einer Rechenleistung von 17,59 PetaFLOPS galt er von November 2012 bis April 2013 als der leistungsstärkste Supercomputer der Welt.
- 2013 Der Tianhe-2 ist ein chinesischer Supercomputer am Nationalen Supercomputer-Zentrum in Guangzhou. Er erreichte in seiner ersten Ausbaustufe (Juni 2013) eine Rechenleistung von 33,86 PetaFLOPS und war fast doppelt so schnell wie der bis dahin leistungsstärkste Supercomputer, der US-amerikanische Titan (17,59 PFLOPS). Die theoretische Maximalleistung des Systems wird mit 55 PetaFLOPS angegeben.
- 2015 Vom US-Energieministerium betriebene Trinity-Supercomputer wurde gestartet. Auch hier kommen in den Nodes Intels Xeon 7250 mit 68 Kernen zum Einsatz. Mit insgesamt 979,968 Kernen kommt das System auf 14,14 PetaFLOPS.
- 2016 Der Sunway TaihuLight ist (Stand März 2019) der dritt schnellste Supercomputer der Welt. Mit 93,0146 PetaFLOPS war er von Juni 2016 bis Juni 2018 der mit großem Abstand schnellste Supercomputer der Welt.
- 2018 Der Summit (auch OLCF-4 genannt) ist ein Supercomputer des Unternehmens IBM, der im Oak Ridge National Laboratory in den USA im Juni 2018 in Betrieb genommen wurde. Zum Zeitpunkt der Inbetriebnahme war der Summit der nach dem Linpack-Benchmark mit einer Spitzenleistung von rund 122 PetaFLOPS schnellste Supercomputer der Welt und führt die TOP500-Liste seit Juni 2018 an.

Sierra ist IBM- Supercomputer am Lawrence Livermore National Laboratory der USA. Er nahm im Juni den Betrieb auf und ist der "militärische Zwilling" des Summit-Supercomputers, der auf gleicher Hybrid-Architektur aus POWER9-Prozessoren und Nvidia Tesla Volta Rechenbeschleunigern besteht. Während Summit für zivile Forschung genutzt wird, soll Sierra die Atomwaffenforschung unterstützen.

2019 IBM Q System One ist der weltweit erste kommerzielle Quantencomputer, der auf Schaltkreisen basiert.

heute Einsatz von Mikrocomputern mit 32 oder 64-Bit Mikroprozessoren sowie Entwicklung und Einsatz von Parallelrechnern

¹siehe Kapitel 8.10

²ehemalige Intel Mitarbeiter

³ehemalige Motorola Mitarbeiter; MOS Technology wurde später von Commodore aufgekauft.

Tabelle 1.1: Entwicklung von Standard-Mikroprozessoren in den letzten 35 Jahren (Auszug)

Jahr	Prozessor	Typ ¹	Hersteller	Wortbreite
1971	i4004	CISC	Intel	4 Bit
1972	i8008	CISC	Intel	8 Bit
1973	i8080	CISC	Intel	8 Bit
1974	2650		Valvo	8 Bit
1975	Pace		National Semiconductors	16 Bit
1976	Z80	CISC	Zilog ²	8 Bit
	6502	CISC	MOS Technology ³	8 Bit
1978	i8088	CISC	Intel	8 Bit
1979	i8086	CISC	Intel	16 Bit
	MC 68000	CISC	Motorola	32 Bit
1982	i80286	CISC	Intel	16 Bit
1984	MC 68020	CISC	Motorola	32 Bit
1985	i80386	CISC	Intel	32 Bit
1988	MC 68030	CISC	Motorola	32 Bit
1989	i80486	CISC	Intel	32 Bit
1990	MC 68040	CISC	Motorola	32 Bit
1991	AM386DX	CISC	AMD	32 Bit
1993	Pentium	CISC	Intel	32 Bit
	R4000	RISC	MIPS	32 Bit
	SPARC-ECL	RISC	SUN	32 Bit
	MPC601 (PowerPC)	RISC	Motorola, Apple, IBM	32 Bit
1994	MPC603	RISC	Motorola, Apple, IBM	32 Bit
	Alpha	RISC	DEC	64 Bit
1995	R10000	RISC	MIPS	64 Bit
	MPC604	RISC	Motorola, Apple, IBM	32 Bit
1997	MPC750	RISC	Motorola, Apple, IBM	32 Bit
	PentiumII	CISC	Intel	32 Bit
	UltraSPARC II	RISC	Sun	64 Bit
1998	R12000	RISC	SGI	64 Bit
1999	Pentium III	CISC	Intel	32 Bit
	MPC7400	RISC	Motorola, IBM	32 Bit
	Athlon	CISC	AMD	32 Bit
2000	Pentium IV	CISC	Intel	32 Bit
2001	IA64 (Itanium)	CISC	Intel	64 Bit
	MPC7450	RISC	Motorola, IBM	32 Bit
2002	Itanium2	CISC	Intel	64 Bit
	MPC970	RISC	IBM, Apple	64 Bit
	Alpha-EV7	RISC	Compaq	64 Bit
2003	UltraSPARC III	RISC	Sun	64 Bit
	Pentium M	CISC	Intel	32 Bit
	AMD64	CISC	AMD	64 Bit
2004	Xeon	CISC	Intek	64 Bit
	PA-8800	RISC	HP	64 Bit
	MPC970FX	RISC	IBM, Apple	64 Bit
	UltraSPARC IV	RISC	Sun	64 Bit
2005	Pentium D	CISC	Intel	64 Bit
	Athlon64X2	CISC	AMD	64 Bit
	MPC970MP	RISC	IBM, Apple	64 Bit
	UltraSPARC T1	CISC	Sun	64 Bit
2006	Core2 Duo	RISC	Intel	64 Bit
	Montecito	RISC	Intel	64 Bit

Kapitel 2

Aufbau und Funktion eines Digitalrechners

2.1 Aufbau eines Arbeitsplatzrechners

Abbildung 2.1 zeigt den Aufbau eines modernen Arbeitsplatzrechners. Konkret wird der Aufbau eines PCs auf Intel-Basis dargestellt. Das Prinzip ist jedoch übertragbar auf alle anderen Arbeitsplatzrechner (Mac, Workstation, etc.).

Den Kern des Systems stellt die CPU (Zentraleinheit) dar. Sie arbeitet eine Folge von Maschinenbefehlen ab. Sie ist über einen so genannten CPU-Bus mit einem externen schnellen Zwischenspeicher verbunden. In diesem Zwischenspeicher werden Daten und Befehle zwischengespeichert, die häufig benötigt werden.

Der CPU-Bus besitzt eine Anbindung an die so genannte *Host-Brücke*, die die Datenübermittlung von und zu Speicherbus (für Zugriffe auf den Hauptspeicher), Peripherie-Bus (für die Kommunikation mit Peripheriegeräten) und, in manchen Systemen, den Grafik-Bus, der eigentlich nur ein speziell für die bei Grafikanwendungen benötigten hohen Übertragungsraten angepasster Peripheriebus ist, realisiert.

In PC-Systemen und Macs ist der Peripheriebus in der Regel ein PCI-Bus. Über diesen Bus findet die Kommunikation der CPU mit den Peripheriegeräten und den LAN-Controller statt. Als Peripheriegeräte bezeichnet man z. B. Massenspeicher, Tastatur, Monitor, etc.

Das vorliegende Skript dient dazu, die Grundlagen des Aufbaus und der Funktionsweise des Digitalrechners insgesamt und seiner Komponenten zu vermitteln. Dabei enthalten die Kapitel 3, 4, 5 und 6 Grundlagen, die für das Verständnis sämtlicher Teile notwendig sind. Kapitel 7 beschreibt Grundlagen der Automatentheorie. Sowohl die CPU als auch die Kontroller-Bausteine für die Bussysteme sind prinzipiell Automaten. Die eigentliche CPU ist Inhalt von Kapitel 8. Eine CPU besteht schematisch aus Rechenwerk, Steuerwerk und internem Speicher. Sie ist in der Lage, Maschinenprogramme auszuführen, die aus Sequenzen von Binärzahlen bestehen. Um die Programmierung zu erleichtern, gibt es Hilfsprogramme, die diese Programme aus sog. Assemblerprogrammen erzeugen (Kapitel 9).¹ Ein- und Ausgabe werden in Kapitel 10 behandelt. Kapitel 11 beschreibt die Grundlagen der Speichertechnik, sowohl innerhalb des Rechners

¹Noch einfacher ist die Programmierung in so genannten Hochsprachen (z. B. Modula oder C++), die ebenfalls von Übersetzern in Maschinencode übersetzt werden. Dies ist jedoch nicht Gegenstand der Vorlesung.

(Hauptspeicher, Caches) als auch die Verfahren zur externen Datenspeicherung (Massenspeicher).

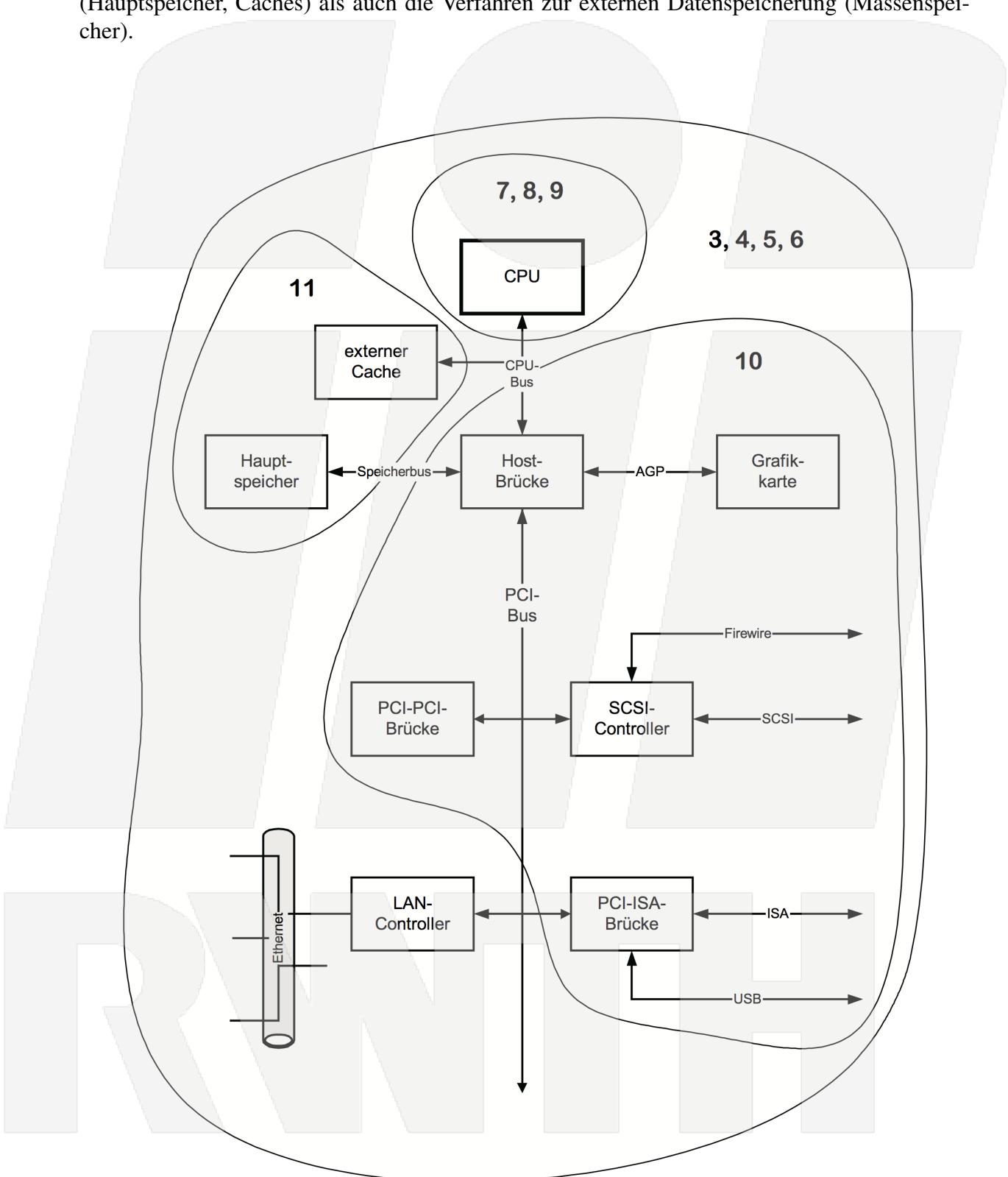


Abbildung 2.1: Skizze des Aufbaus eines Arbeitsplatzrechners (PC). Die Zahlen geben die Kapitel an, die den angegebenen Bestandteil beschreiben.

2.2 Kennwerte eines Digitalrechners

Charakteristische Kenngrößen für die Leistungsfähigkeit eines Digitalrechners sind u. a. Rechengeschwindigkeit des Prozessors bzw. der Prozessoren und die Kapazität des Hauptspeichers. Je nach Anwendung können auch andere Kenngrößen wie z. B. Datensicherheit von großer Bedeutung sein. Tabelle 2.1 zeigt typische Kenngrößen von verschiedenen Rechnerarten.

- Rechengeschwindigkeit

Die Geschwindigkeit eines Prozessors kann in MIPS (Million Instructions Per Second) gemessen werden. Wenn es, wie bei technisch-wissenschaftlichen Anwendungen, im wesentlichen auf die Durchführung von arithmetischen Operationen mit reellen Zahlen ankommt, kann die Geschwindigkeit des Rechners auch in FLOPS (Floating Point Operations Per Second) angegeben werden.

In Kapitel 8 wird gezeigt, dass eine Angabe von MIPS bzw. MFLOPS die Geschwindigkeit eines Rechners nur unzureichend beschreiben kann. Aus diesem Grund wurden verschiedene, genormte Testprogramme entwickelt, deren Ausführungszeit die Rechengeschwindigkeit charakterisieren soll (Benchmarktests).

- Kapazität des Hauptspeichers

Die kleinste Informationseinheit im Digitalrechner wird als 1 Bit (Binary Digit) bezeichnet und kennzeichnet die beiden Zustände 0 und 1 bzw. Strom ein / Strom aus.² Größere Informationseinheiten sind wie folgt definiert:

1	Byte	=	8	Bit
1	KBit (Kilobit)	=	1024	Bit
1	KB (Kilobyte)	=	1024	Byte
1	MB (Megabyte)	=	1024	KB
1	GB (Gigabyte)	=	1024	MB
1	TB (Terabyte)	=	1024	GB

- Preis

Die in der Abbildung genannten Zahlenwerte sind lediglich Anhaltspunkte. Für ein konkretes Rechnermodell kann die eine oder andere Kenngröße beträchtlich von der hier genannten abweichen.

Tabelle 2.1: Einige Kenngrößen verschiedener Rechnerarten

Rechnerart	Rechengeschw.	Speicherkap.	typische Hersteller
Personalcomputer	20 – 2000 MIPS	32 – 512 MB	IBM, Intel, Apple DM
Workstations	1000 – 10000 MIPS	0,5 – 8 GB	Sun, HP, Compaq, SGI, IBM
Mainframes	1000 – 10000 MIPS	> 2 GB	IBM, Amdahl
Supercomputer	< 15 TFLOPS	> 1 TB	IBM, Intel, Hitachi, Cray

²Die Einheit „bit“ steht im Gegensatz dazu für „basic indissoluble information unit“ und ist die Einheit für den Informationsgehalt einer Nachricht (siehe auch Kapitel 3).

Die fünfhundert schnellsten Supercomputer werden etwa halbjährlich in der so genannten TOP 500-Liste veröffentlicht (<http://www.top500.org/>). Die Spitzenreiter (Stand: Juni 2015) sind in Tabelle 2.2 aufgelistet. Bei Aufgaben, die sich sehr gut in kleinere Teile aufteilen lassen, kann auch durch die Zusammenschaltung sehr vieler Rechner über das Internet eine Rechenleistung erzielt werden, die der Leistung der schnellsten Supercomputer entspricht (Beispiel: Das SETI@home-Projekt, bei dem über 2 Millionen Standard-Rechner beteiligt sind, die zusammen eine theoretische Maximalleistung von 12 Tflops erreichen).

Tabelle 2.2: Die zehn schnellsten Supercomputer (Stand: Juni 2015)

Hersteller	Computer	Leistung (TFLOPS)	Prozessoren	Speicher (TB)	Installationsort und -jahr
NUDT	Tianhe-2 (Milkyway-2) TH-IVB-FEP	33.862,7	3.120.000	1024	National Supercomputer Zentrum, China 2013
Cray Inc.	Titan - Cray XK7	17.590	560.640	710	DOE/SC/Oak Ridge National Laboratory, USA 2012
IBM	Sequoia BlueGene/Q	17.173	1.572.864	1572	DOE/NNSA/LLNL, USA, 2011
Fujitsu	K Computer SPARC64	10.510	705.024	1.410	RIKEN Advanced Institute for Computer Science, Japan 2011
IBM	Mira BlueGene/Q	8.587	786.432	k.A.	DOE/SC/Argonne National Laboratory, USA, 2012
Cray Inc.	Piz Daint - Cray XC30	6.271	786.432	k.A.	SSwiss National Supercomputing Centre , Switzerland, 2012
Cray Inc.	Shaheen II - Cray XC40	5.537	115.984	k.A.	King Abdullah University of Science and Technology, Saudi Arabia, 2015
Dell	Stampede - PoerEdge C8220	5.168	462.462	192	Texas Advanced Computing Center, USA, 2012
IBM	JUQUEEN - BlueGene/Q	5.009	458.752	458	Forschungszentrum Jülich, Germany, 2012
IBM	Vulcan BlueGene/Q	4.293	393.216	393	DOE/NNSA/LLNL, USA, 2012

Kapitel 3

Informationsdarstellung und Codierung

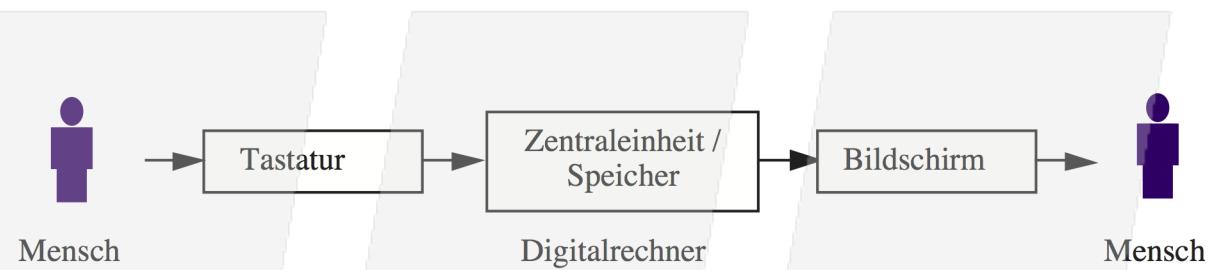


Abbildung 3.1: Informationsverarbeitung

Abbildung 3.1 zeigt eine vereinfachte Sicht der Informationsverarbeitung mit einem Digitalrechner. Der Benutzer gibt die zur Verarbeitung notwendigen Informationen und die Verarbeitungsvorschrift (Programm) über die Tastatur ein. Der Digitalrechner verarbeitet die Informationen dem Programm entsprechend und gibt die Ergebnisse auf den Bildschirm aus, wo der Benutzer sie ablesen kann.

Bevor Informationen im Digitalrechner verarbeitet werden, müssen sie in einer dem Rechner verständliche Form umgesetzt werden. Im gewählten Beispiel erfolgt die Umsetzung mittels Tastatur. Während der Verarbeitung im Digitalrechner oder einer Übertragung zwischen mehreren Digitalrechnern werden die Nachrichten wiederum umgewandelt. Schließlich werden die Ergebnisse der Informationsverarbeitung in einer dem Menschen verständlichen Form zur Ausgabe auf den Bildschirm gebracht.

3.1 Codierung

In der Informatik unterscheidet man zwischen den Begriffen *Information* und *Nachricht*. Unter einer Information versteht man die Bedeutung eines Sachverhalts. Eine Nachricht dient der Darstellung einer Information, sie enthält somit die Information. Nachrichten sind z.B.

- Worte → bestehend aus Buchstaben
- Bilder → bestehend aus Bildelementen (Pixel)

Einer Information können mehrere Nachrichten zugeordnet sein (z.B. Computer - Rechner), ebenso können einer Nachricht mehrere Informationen zugeordnet sein (z. B. Bank; je nach

Kontext bezeichnet die Nachricht ein Geldinstitut oder eine Sitzgelegenheit). Der Fall der mehrdeutigen Nachricht soll im Weiteren für die Codierung ausgeschlossen werden.

Allgemein formuliert sind Nachrichten Worte, bestehend aus Elementen (Zeichen, Buchstaben oder Ziffern genannt) einer vereinbarten, endlichen Menge. Eine solche Menge von Zeichen nennt man Alphabet.

Alphabet: endliche Menge von Zeichen

$$\begin{aligned}
 A &= \{a_1, \dots, a_n\}; \\
 A_{bin} &= \{0, 1\}; \quad \text{Binärziffern} \\
 A_{dez} &= \{0, 1, \dots, 9\}; \quad \text{Dezimalziffern} \\
 A_{morse} &= \{\cdot, -\}; \quad \text{Zeichen des Morsealphabets ohne Zwischenraum} \\
 A_{\alpha-num} &= \{A, \dots, Z, 0, \dots, 9, \text{Sonderzeichen}\}; \quad \text{alphanumerisches Alphabet}
 \end{aligned}$$

Wort: Folge verketteter, nicht notwendig verschiedener Zeichen.

$$\begin{aligned}
 A^m &= \text{Menge der Worte der Länge } m \text{ über } A, m \in \mathbb{N} \\
 A^* &= \text{Menge aller Worte über } A
 \end{aligned}$$

Ein Wort $a \in A^*$ der Länge $n, n \in \mathbb{N}$, schreibt man auch als

$$a = a_1 a_2 \dots a_n$$

$a_i \in A$ bezeichnet das i -te Zeichen des Wortes a mit $1 \leq i \leq n$.

Konkatenation: Verkettung zweier Worte $a \circ a'$ über A^* .

Sei a ein Wort der Länge n und a' ein Wort der Länge m über A^* , dann ist

$$(a \circ a') = \begin{cases} a_i & \text{für } 1 \leq i \leq n \\ a'_{i-n} & \text{für } n+1 \leq i \leq n+m \end{cases}$$

Kardinalität: Anzahl verschiedener Worte A^m .

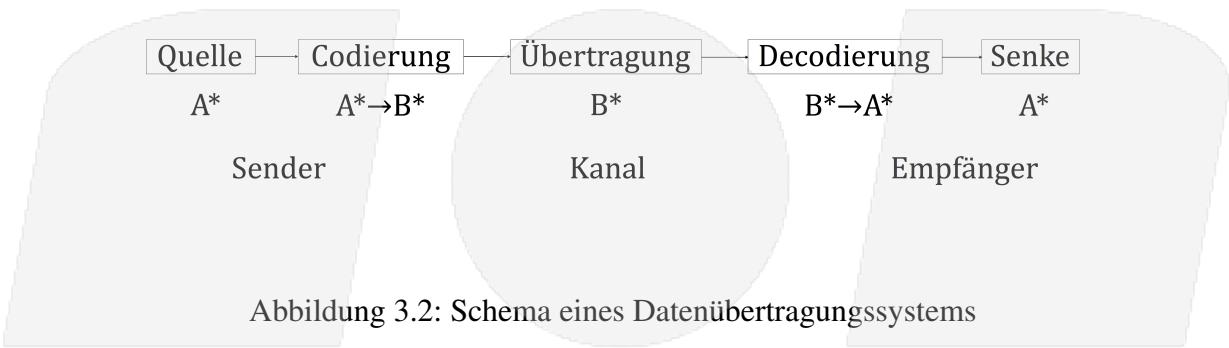
$$\begin{aligned}
 A &= \{a_1, \dots, a_n\}; \\
 \text{card}A^m &= (\text{card}A)^m = n^m
 \end{aligned}$$

Beispiel: Gesucht ist die Anzahl der 8 Bit langen Binärworte.

$$\begin{aligned}
 A_{bin} &= \{0, 1\}; \\
 \text{card}A_{bin}^8 &= (\text{card}A_{bin})^8 = 2^8
 \end{aligned}$$

Es können 256 verschiedene 8 Bit lange Worte gebildet werden.

Abbildung 3.2 zeigt ein typisches Codierungsbeispiel, die Datenübertragung. Quelle und Senke benutzen das Alphabet A, und die Übertragung schreibt das Alphabet B vor. Codierer und Decodierer leisten die Umsetzung von A nach B und umgekehrt.



Codierung: Unter einer direkten Codierung κ versteht man eine injektive (eindeutige) Abbildung

$$\kappa: A \rightarrow B,$$

die jedem Zeichen $z \in A$ ein Zeichen $\kappa(z) \in B$ zuordnet.

Eine Codierung κ^* , die jedem Wort $a = z_1 z_2 \dots z_n \in A^*$ ein Wort $\kappa(a) \in B^*$ zuordnet, beschreibt man wie folgt: Seien A und B nichtleere Alphabete. Eine Codierung κ^* ist eine injektive Abbildung:

$$\begin{aligned} \kappa^*: A^* &\rightarrow B^* \\ \text{mit } \kappa^*(z_1 \circ z_2 \circ \dots \circ z_n) &= \kappa(z_1) \circ \kappa(z_2) \circ \dots \circ \kappa(z_n) \end{aligned}$$

Eine direkte Umsetzung von einem Zeichen des einen Alphabets in ein einzelnes Zeichen eines anderen Alphabets ist nur selten anzutreffen. Der häufigere Fall ist die Codierung eines Zeichens durch ein Wort.

$$\kappa: A \rightarrow B^*$$

Beispiel: Morse-Codierung

$$\kappa: \{A, \dots, Z, 0, \dots, 9\} \rightarrow \{\cdot, -\}^*$$

Code: DIN 44300

- Eine Vorschrift für die eindeutige Zuordnung (Codierung) der Zeichen eines Zeichenvorrats zu denjenigen eines anderen Zeichenvorrats (Bildmenge).
- Zeichenvorrat der Bildmenge

3.2 Informationsgehalt einer Nachricht

Betrachtet man das Codierungsbeispiel der Datenübertragung, so ist die Wahrscheinlichkeit, mit der eine bestimmte Nachricht an der Senke erwartet wird, eine Aussage über die Bedeutung der

Nachricht für den Empfänger. Nachrichten, die mit hoher Wahrscheinlichkeit auftreten, beinhalten wenig neue Information. Nachrichten mit geringer Auftrittswahrscheinlichkeit stellen eine Überraschung dar, sie beinhalten viel neue Information.

Die Auftrittswahrscheinlichkeiten der einzelnen Zeichen eines Alphabets reichen zur Beschreibung einer Quelle mit statistisch unabhängigen Zeichenfolgen aus. (Quellen mit Gedächtnis werden hier nicht betrachtet.)

Seien $a_1, a_2, \dots, a_n \in A$ Zeichen, die mit den Wahrscheinlichkeiten p_1, p_2, \dots, p_n auftreten und treten keine anderen Zeichen auf

$$\sum_{j=1}^n p_j = 1$$

dann berechnet sich der *Informationsgehalt* I (s. Abb. 3.3) eines Zeichens a_j zu:¹

$$I(a_j) = \log\left(\frac{1}{p_j}\right) = -\log p_j$$

Die relative Gewichtung des Informationsgehalts einzelner Zeichen einer Nachricht nach ihrer Auftrittswahrscheinlichkeit führt zur *Entropie* H , als Maß für die Ungewissheit hinsichtlich des Verhaltens der Quelle, d. h. für den Informationsgehalt einer Nachricht.

Die Entropie H eines Alphabets berechnet sich nach der Shannon'schen Formel:

$$H = \sum_{j=1}^n p_j \cdot \log \frac{1}{p_j}$$

$$H = - \sum_{j=1}^n p_j \cdot \log p_j (*)$$

Der Verlauf der Unsicherheitsfunktion (*) ist in Abbildung 3.3 dargestellt.

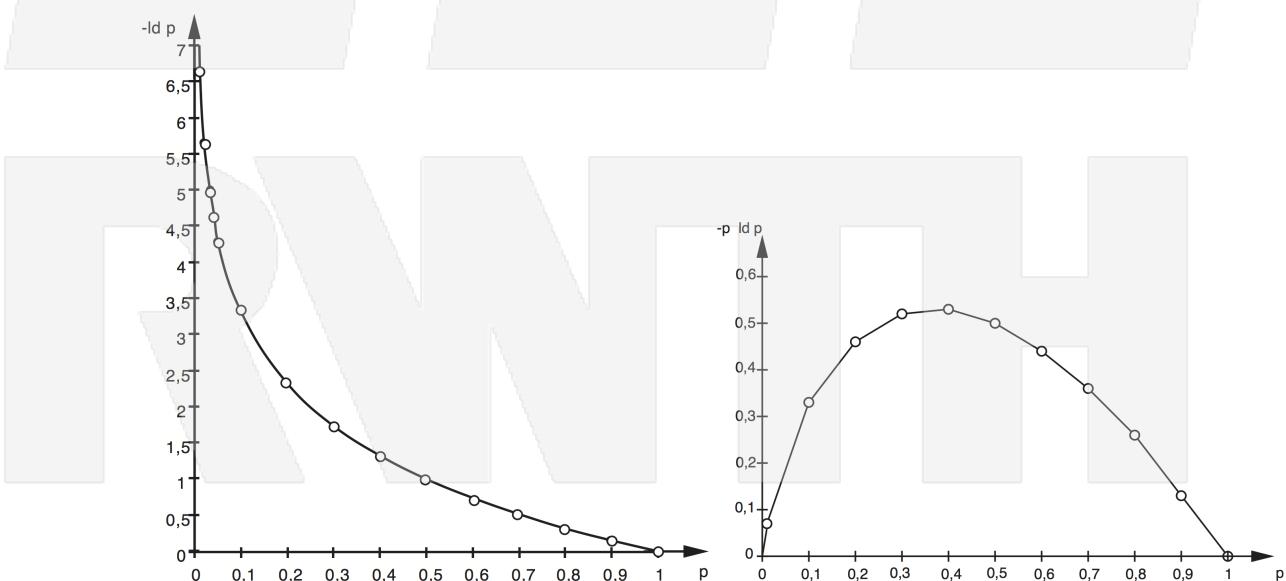


Abbildung 3.3: Informationsgehalt (links) und Unsicherheitsfunktion (rechts)

¹Die Einheit des Informationsgehalts ist durch die Basis des Logarithmus bestimmt (Basis 2 ergibt die Einheit bit; eine andere Basis ergibt dem entsprechend eine andere Einheit).

Die maximale Entropie H_{max} ergibt sich, wenn alle n Zeichen gleich wahrscheinlich sind ($p = \frac{1}{n}$).

$$H_{max} = H_0 = - \sum_{j=1}^n \frac{1}{n} \cdot \log \frac{1}{n} = \log n$$

Die Wahl des Logarithmus ist zunächst beliebig. Eine Umrechnung ist möglich:

$$\log_a x = \frac{\log_b x}{\log_b a}$$

Beispiel:

$$\text{ld } 32 = \frac{\log_{10} 32}{\log_{10} 2} = \frac{1.505}{.301} = 5$$

Im Hinblick auf Ja/Nein-Entscheidungen hat sich der Zweierlogarithmus für die Entropieberechnung durchgesetzt. Als Einheit wird ein *bit* (basic indissoluble information unit) verwendet.²

Als Redundanz wird der Anteil einer Nachricht bezeichnet, der keine Information vermittelt. In der Nachrichtenübertragung wird zur Fehlererkennung die Nachricht absichtlich um redundante Anteile wie z.B. Paritätsbits erweitert. Die Redundanz R eines Codes errechnet sich aus der Differenz zwischen maximal möglicher und tatsächlich verwandelter Entropie:

$$\begin{aligned} R &= H_0 - H \\ R &= H_0 - \left(- \sum_{j=1}^n p_j \cdot \text{ld } p_j \right) \\ R &= H_0 + \sum_{j=1}^n p_j \cdot \text{ld } p_j \\ R &= \text{ld } n + \sum_{j=1}^n p_j \cdot \text{ld } p_j \end{aligned}$$

Relative Redundanz r :

$$r = \frac{R}{H_0} = \frac{R}{\text{ld } n}$$

oder

$$r = 1 + \frac{1}{\text{ld } n} \sum_{j=1}^n p_j \cdot \text{ld } p_j$$

Beispiel:

Gegeben sei das Alphabet

$$A = \{a, b, c, d, e, f, g, h\}$$

mit den zugehörigen Auftrittswahrscheinlichkeiten:

$$p_a = 0.0625, p_b = 0.0625, p_c = 0, p_d = 0.0625, p_e = 0.5, p_f = 0.125, p_g = 0.125, p_h = 0.0625$$

²Die Ähnlichkeit zur Einheit „Bit“ ist nicht zufällig: Zur Darstellung von n bit in einem Digitalrechner benötigt man mindestens n Bit.

$$\begin{aligned}
 H &= - \sum_{j=1}^n p_j \cdot \text{ld } p_j \\
 &= \sum_{j=1}^n p_j \text{ld } \frac{1}{p_j} \\
 &= 4 \cdot 0.0625 \cdot \text{ld } 16 + 0 + 1 \cdot 0.5 \cdot \text{ld } 2 + 2 \cdot 0.125 \cdot \text{ld } 8 \\
 &= 0.25 \cdot \text{ld } 16 + 0.5 \cdot \text{ld } 2 + 0.25 \cdot \text{ld } 8 \\
 H &= \underline{\underline{2.25 \text{ bit}}}
 \end{aligned}$$

Wenn zur Zeichendarstellung 3 Bit benutzt werden, folgt die Redundanz zu:

$$R = H_0 - H = 3 \text{ bit} - 2.25 \text{ bit} = \underline{\underline{0.75 \text{ bit}}}$$

Relative Redundanz:

$$r = \frac{0.75 \text{ bit}}{3 \text{ bit}} = \underline{\underline{0.25}}$$

3.3 Wichtige Codes

3.3.1 Optimalcodes (Codeworte unterschiedlicher Länge)

$$\kappa: A \rightarrow B^*$$

Ziel des Optimalcodes ist die Minimierung der mittleren Wortlänge:

$$\bar{l} = \sum_{j=1}^n p_j l_j$$

Morse-Code

Morsecodierung und Häufigkeit von Buchstaben (Tab. 3.1 u. Abb. 3.4)

Tabelle 3.1: Morse-Alphabet

A	· -	J	· - - - -	S	· · ·	1	· - - - -
B	- · · ·	K	- - · -	T	-	2	· · - - -
C	- - · - ·	L	· - - ..	U	· · -	3	· · · - -
D	- - ..	M	- -	V	· · · -	4	· · · · -
E	.	N	- - .	W	· - -	5	· · · · ·
F	· · - -	O	- - - -	X	- · · -	6	- · · · ·
G	- - - .	P	· - - - -	Y	- - - -	7	- - - - -
H	· · · ·	Q	- - - - -	Z	- - - ..	8	- - - - - -
I	..	R	· - - .	0	- - - - - -	9	- - - - - - -

Längenklassen der Morsezeichen:

l_j	a_j
1	{E,T}
2	{A,I,M,N}
3	{D,G,K,O,R,S,U,W}
4	{B,C,F,H,J,L,P,Q,V,X,Y,Z}
5	{0,1,2,3,4,5,6,7,8,9}

Reihenfolge der Häufigkeit von Buchstaben in englischen Texten (aus 1,7 Mio. Zeichen):

E, T, A, O, I, N, S, R, H, L, C, D, U, M, P, F, Y, G, W, B, V, K, X, J, Q, Z

Reihenfolge der Häufigkeit von Buchstaben in deutschen Texten (s. auch Tab. 3.2):

E, N, R, I, S, T, D, H, A, U, L, C, G, M, O, B, Z, W, F, K, V, Ü, P, Ä, Ö, J, Y, Q, X

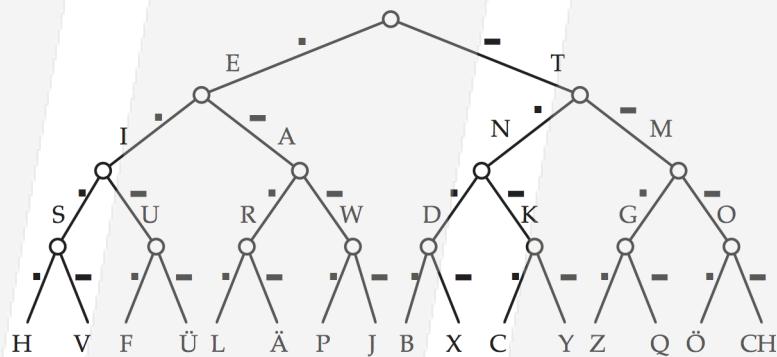


Abbildung 3.4: Codebaum des Morse-Alphabets

Das Erkennen von Beginn und Ende eines Zeichens im Morse-Code erfolgt durch eine Pause (Zwischenraum) zwischen den Zeichen.

Fano-Code

Im *Fano-Code* (Tab. 3.2) wird das Erkennen von Beginn und Ende eines Zeichens durch die **Fano-Bedingung** gewährleistet:

Kein Wort des Codes darf Anfang eines anderen Codewortes sein.

Tabelle 3.2: Buchstabenhäufigkeit in der deutschen Sprache und Fano-Code

a_j	p_j	Fano-Code	l_j	a_j	p_j	Fano-Code	l_j
Zw	0,15149	000	3	O	0,01772	111001	6
E	0,14700	001	3	B	0,01597	111010	6
N	0,08835	010	3	Z	0,01423	111011	6
R	0,06858	0110	4	W	0,01420	111100	6
I	0,06377	0111	4	F	0,01360	111101	6
S	0,05388	1000	4	K	0,00956	1111100	7
T	0,04731	1001	4	V	0,00735	1111101	7
D	0,04385	1010	4	Ü	0,00580	11111100	8
H	0,04355	10110	5	P	0,00499	11111101	8
A	0,04331	10111	5	Ä	0,00491	11111110	8
U	0,03188	11000	5	Ö	0,00255	111111110	9
L	0,02931	11001	5	J	0,00165	1111111110	10
C	0,02673	11010	5	Y	0,00017	11111111110	11
G	0,02667	11011	5	Q	0,00015	111111111110	12
M	0,02134	111000	6	X	0,00013	111111111111	12

3.3.2 Block-/Wortcodes (Codeworte fester Länge)

$$\kappa: A \rightarrow B^m$$

Codierung alphanumerischer Zeichen

Für die Verarbeitung von Text ist der folgende Zeichenumfang erforderlich:

je 26 Groß- und Kleinbuchstaben	10 Ziffern	1 Leerzeichen
ca. 16 Sonderzeichen	ca. 10 Steuerzeichen	

insgesamt ca. 89 Zeichen

Die Binär-Darstellung erfordert eine Mindestwortlänge von $m = 7$

$$\text{ld } 89 < \text{ld } 128 = 7$$

American Standard Code for Information Interchange (ASCII)

Der ASCII-Code (Tab. 3.3) ist für die Nachrichtentechnik entwickelt worden. Die beiden ersten Spalten enthalten die Steuerzeichen (Tab. 3.4). Die zusätzlichen Zeichen dienen der Darstellung nationaler Sonderzeichen und Grafiksymbole.

Unicode

Der ASCII-Code ist kulturell auf den Westen beschränkt. Angestrebt wird der 16-Bit-*Unicode*, der es erlaubt 65536 Zeichen darzustellen, unter anderem auch die 27000 chinesischen Han-

Tabelle 3.3: ASCII: Die Spalte gibt die vorderen 3 Bits, die Zeile die hinteren 4 Bits an

	0	1	2	3	4	5	6	7	
0	NUL	DLE	(leer)	0	@	P	'	p	0000
1	SOH	DC1	!	1	A	Q	a	q	0001
2	STX	XON	"	2	B	R	b	r	0010
3	ETX	DC3	#	3	C	S	c	s	0011
4	EOT	XOF	\$	4	D	T	d	t	0100
5	ENQ	NAK	%	5	E	U	e	u	0101
6	ACK	SYN	&	6	F	V	f	v	0110
7	BEL	ETB	,	7	G	W	g	w	0111
8	BS	CAN	(8	H	X	h	x	1000
9	HT	EM)	9	I	Y	i	y	1001
A	LF	SUB	*	:	J	Z	j	z	1010
B	VT	ESC	+	;	K	[k	{	1011
C	FF	FS	,	<	L	\	l		1100
D	CR	GS	-	=	M]	m	}	1101
E	SO	RS	.	>	N	^	n	~	1110
F	SI	US	/	?	O	_	o	DEL	1111
	000	001	010	011	100	101	110	111	

Tabelle 3.4: ASCII: Abkürzungen und Vollnamen der Steuerzeichen

NUL	alle Leit. NUL	VT	Vertical Tab	SYN	SYNchron. idle
SOH	Start Of Heading	FF	Form Feed	ETB	End Transm. Block
STX	Start of TeXt	CR	Carriage Return	CAN	CANcel
ETX	End of TeXt	SO	Shift Out	EM	End of Medium
EOT	End Of Transm.	SI	Shift In	SUB	SUBstitute
ENQ	ENquiry	DLE	Data Link Escape	ESC	ESCAPE
ACK	ACKnowledged	DC1	Device Control 1	FS	File Separator
BEL	BELL	XON	XON-protocol	GS	Group Separator
BS	BackSpace	DC3	Device Control 3	RS	Record Separator
HT	Horizontal Tab	XOF	XOFF-protocol	US	Unit Separator
LF	Line Feed	NAK	Not AcKnowl.	DEL	DElete

Zeichen, sowie die japanischen Zeichenalphabete. Tabelle 3.5 zeigt die Aufteilung der Unicode-Tabelle. Die ersten 255 Zeichen entsprechen dem ASCII-Zeichensatz, wobei die ersten 32 Zeichen unbenutzt bleiben. Dort stehen im ASCII Zeichensatz die Steuerzeichen, welche sich im UNICODE im Bereich 2400-2424 befinden.

Tabelle 3.5: Aufteilung der UNICODE Zeichentabelle. Die einzelnen Gruppen sind jeweils in weitere Blöcke untergliedert.

Adresse (hex)	Beschreibung
0020-02a8	'Westliche' Zeichen (in drei Blöcke aufgeteilt)
02b0-02e9	Modifikatoren
0300-0345	raumfreie Symbole (z.B. Akzente)
0374-1ffe	Symbole verschiedener Sprachen
2000-237a	Sonderzeichen und mathematische Symbole
2400-2424	Steuerzeichen
2440-3020	Sonderzeichen
3021-ffff	31 Blöcke mit Symbolen verschiedener Sprachen und Sonderzeichen

Binärcodes für Dezimalzahlen

Tetradencodes Zur binären Darstellung der zehn Dezimalziffern benötigt man $n = 4$ Bit. Mit 4 Bit können 16 verschiedene Zeichen dargestellt werden, von denen 6 (Pseudotetraden, Abk. PT) nicht genutzt werden (Tab. 3.6). Die Redundanz, die zur Fehlererkennung genutzt werden kann, beträgt: $R = (4 - 3.322) \text{ bit} = 0.678 \text{ bit}$

Tabelle 3.6: Verschiedene Tetradencodes (PT gekennzeichnet)

	BCD-Code	Aiken-Code	Exzess-3-Code
0000	0	0	
0001	1	1	
0010	2	2	
0011	3	3	0
0100	4	4	1
0101	5		2
0110	6	Pseudo-	3
0111	7		4
1000	8		5
1001	9	tetraden	6
1010			7
1011		5	8
1100	Pseudo-	6	9
1101		7	
1110	tetraden	8	tetraden
1111		9	

Vor- und Nachteile der verschiedenen Tetradencodes sind Tabelle 3.7 zu entnehmen.

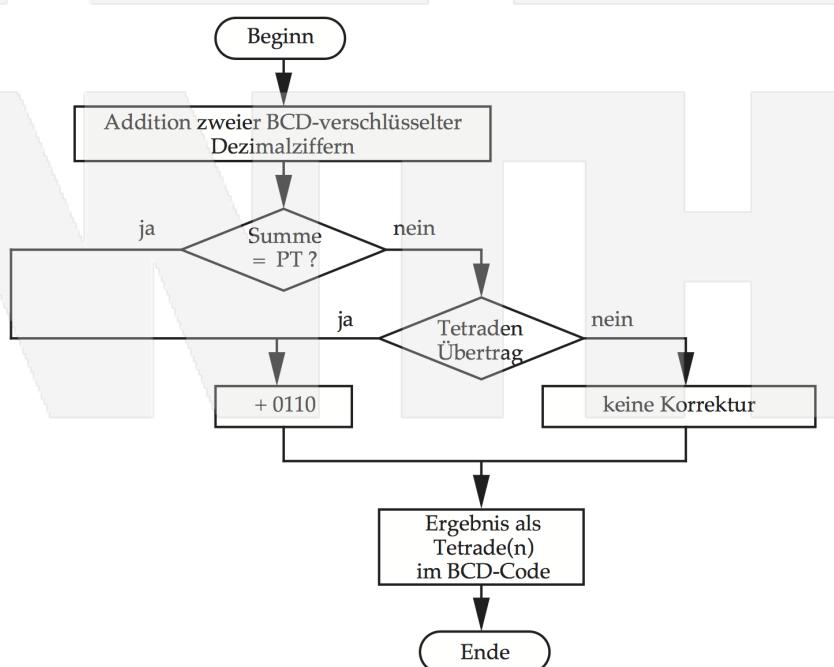
Tabelle 3.7: Vergleich der Tetradencodes

	BCD-Code	Aiken-Code	Exzess-3-Code
Rechenwerk für Addition	kompliziert, da das Auftreten von 6 verschiedenen Pseudotetraden festgestellt werden muss		einfach, da die Korrektur nur vom Übertrag abhängt
Komplementbildung d.h. Subtraktion	schwierige Umrechnung	einfache Umrechnung durch Vertauschung von 1 und 0	

$$\begin{array}{r}
 28936: \quad 0010 \quad 1000 \quad 1001 \quad 0011 \quad 0110 \\
 +48283: \quad 0100 \quad 1000 \quad 0010 \quad 1000 \quad 0011 \\
 \text{Übertrag: } \quad \underline{\quad} \leftarrow \underline{\quad} \leftarrow \underline{\quad} \leftarrow \underline{11\quad} \leftarrow \underline{\quad} \leftarrow \underline{\quad} \\
 \text{1. Summation: } \quad 0111 \quad 0001 \quad 1100 \quad 1011 \quad 11 \\
 \text{Pseudotetraden: } \quad \text{nein} \quad \text{nein} \quad \text{ja} \quad \text{ja} \quad \text{nein} \\
 \text{Tetraden-Übertrag: } \quad \text{nein} \quad \text{ja} \quad * \quad * \quad \text{nein} \\
 \text{Korrektur: } \quad - \quad +0110 \quad +0110 \quad +0110 \quad - \\
 \text{Übertrag: } \quad \underline{\quad} \quad \underline{\quad} \quad \underline{11} \quad \underline{\quad} \quad \underline{\quad} \\
 \text{2. Summation: } \quad 0111 \quad 0111 \quad 0010 \quad 0001 \quad 1001 \\
 = 77219: \quad 7 \quad 7 \quad 2 \quad 1 \quad 9
 \end{array}$$

Abbildung 3.5: Addition im BCD-Code (spaltenweise von rechts nach links). Der "*" hat die Bedeutung eines Don't care, denn da bereits festgestellt wurde, dass es sich um eine Pseudotetraden handelt, wird in jedem Fall korrigiert, egal ob ein Tetradenübertrag vorhanden ist oder nicht.

Verarbeitung von BCD-Code



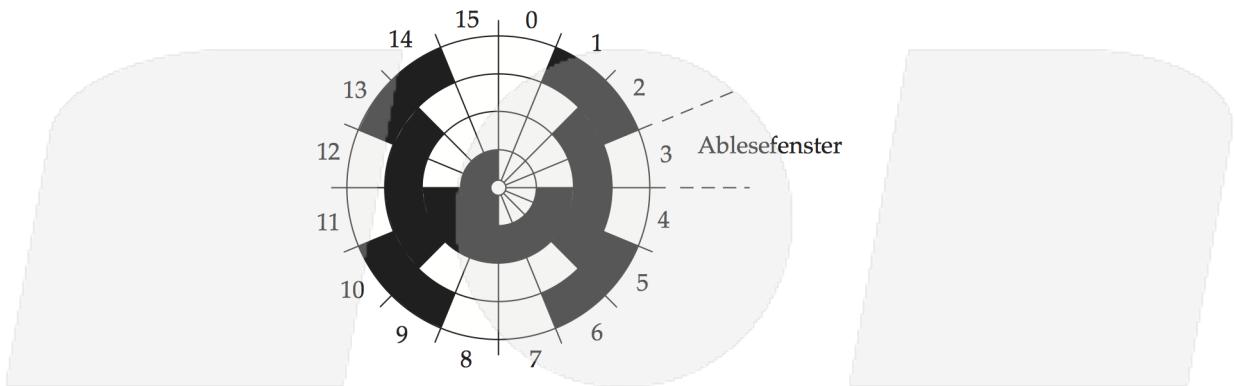


Abbildung 3.6: Codierscheibe (Gray-Code)

Gray-Code

Aufeinanderfolgende Zahlen unterscheiden sich nur in einer Bitstelle (Tab. 3.8). Dies ist bedeutsam für Anwendungen in der Messtechnik. Hierbei kommen häufig Codierscheiben (Abb. 3.6) zum Einsatz.

Tabelle 3.8: Umwandlung Gray-Code \leftrightarrow Dualzahl \leftrightarrow Dezimalzahl

Gray-Code	Dualzahl	Dezimalzahl
0000	0000	0
0001	0001	1
0011	0010	2
0010	0011	3
0110	0100	4
0111	0101	5
0101	0110	6
0100	0111	7
1100	1000	8
1101	1001	9
1111	1010	10
1110	1011	11
1010	1100	12
1011	1101	13
1001	1110	14
1000	1111	15

m-aus-n-Codes

n = Wortlänge;

m = Anzahl der '1'-en im Codewort für den gesamten Ziffernbereich

Zwei-aus-fünf-Code

Dezimalziffer	$\binom{5}{2}$ -Code
0	11000
1	00011
2	00101
3	00110
4	01001
5	01010
6	01100
7	10001
8	10010
9	10100

Für den Fall, dass alle Ziffern gleich wahrscheinlich sind, ergibt sich eine Redundanz von

$$R = (5 - 3.322) \text{ bit} = 1.678 \text{ bit}$$

Bei diesem Code ist eine einfache Fehlererkennung möglich.

Eins-aus-zehn-Code

Dezimalziffer	$\binom{10}{1}$ -Code
0	0000000001
1	0000000010
2	0000000100
3	0000001000
4	0000010000
5	0000100000
6	0010000000
7	0100000000
8	1000000000
9	1000000000

Redundanz:

$$R = (10 - 3.322) \text{ bit} = 6.678 \text{ bit}$$

Dieser Code wird zur Ansteuerung von Ziffernanzeigen verwendet.

Biquinär-Code

Dezimalziffer	Biquinär-Code	
0	00001	01
1	00010	01
2	00100	01
3	01000	01
4	10000	01
5	00001	10
6	00010	10
7	00100	10
8	01000	10
9	10000	10

Redundanz:

$$R = (7 - 3.322) \text{ bit} = 3.678 \text{ bit}$$

Im binären und quinären Teil darf jeweils nur eine '1' vorhanden sein.
Der Biquinär-Code entspricht der Codierung der Zahlen beim Abakus.

3.4 Erkennung und Korrektur von Übertragungsfehlern

3.4.1 Zur Rolle der Hamming-Distanz

Als *Hamming-Distanz* d wird die Anzahl verschiedener Bitstellen in zwei n -Bit langen Worten bezeichnet:

$$\begin{array}{rcl} w_1 & = & 1 \quad 0 \quad 1 \\ w_2 & = & 0 \quad 0 \quad 0 \\ & * & * \end{array}$$

$$d(w_1, w_2) = 2$$

Die Hamming-Distanz d_m eines Codes ist der kleinste auftretende Bitabstand d unter allen gültigen Codeworten (Nutzworten).

Abbildung 3.7 zeigt die grafische Darstellung der Hamming-Distanz für die gültigen Codeworte 000, 110, 101, 011. Hier ist die Hamming-Distanz $d_m = 2$, d.h. zwischen zwei gültigen Codeworten liegt mindestens ein ungültiges. Die Veränderung einer Bitstelle bewirkt, dass beim Empfänger ein ungültiges Codewort (Pseudowort) ankommt. Ein 1-Bit-Fehler ist also erkennbar, aber nicht korrigierbar.

Die Zahl e der erkennbaren Fehler ist $e = (d_m - 1)$. Die Zahl k der korrigierbaren Fehler ist $k = \lfloor \frac{(d_m - 1)}{2} \rfloor$. Dabei bezeichnen die so genannten Gaußklammern „[“ und „]“ die in ihnen enthaltene Zahl, soweit sie ganz ist, oder sonst die nächst kleinere ganze Zahl.

Knotenmodelle dienen der 2-dimensionalen, grafischen Darstellung der Hamming-Distanz (Abb. 3.8 u. 3.9).

Bedeutung der minimalen Hamming-Distanz d_m eines Codes:

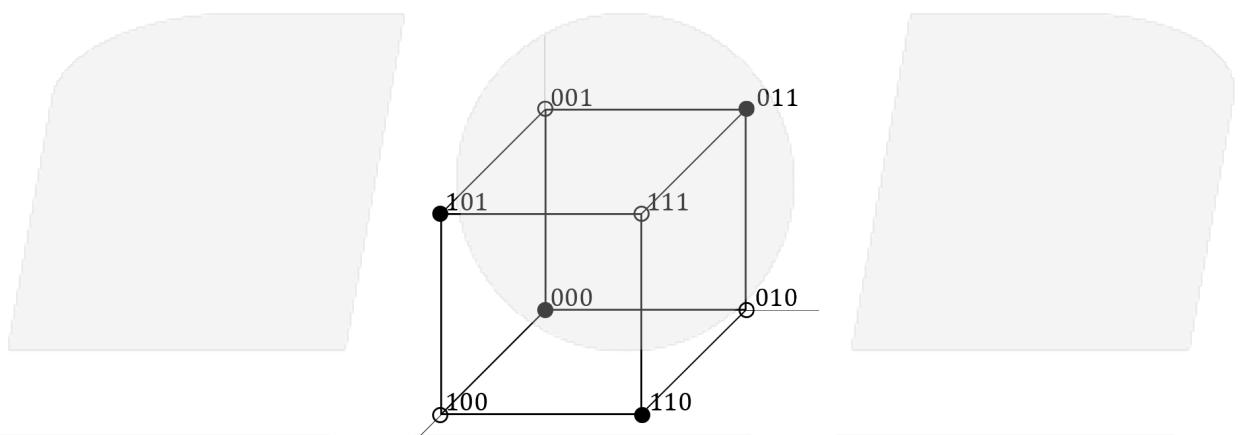


Abbildung 3.7: Darstellung des Binärcodes im dreidimensionalen Vektorraum

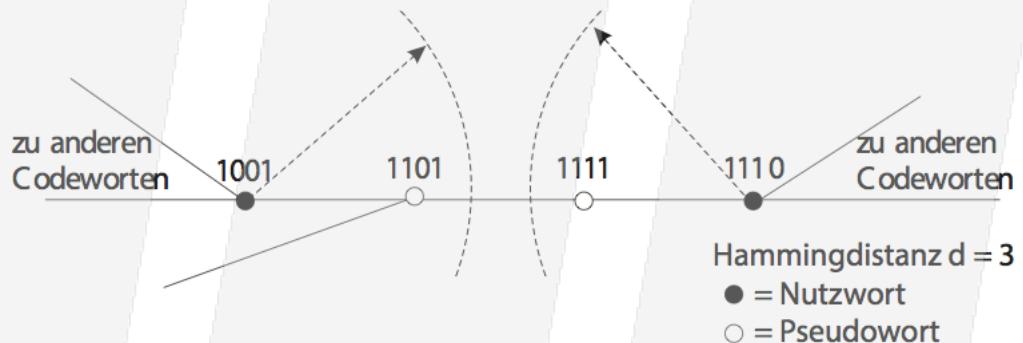


Abbildung 3.8: Knotenmodell für zwei Codeworte mit $d = 3$

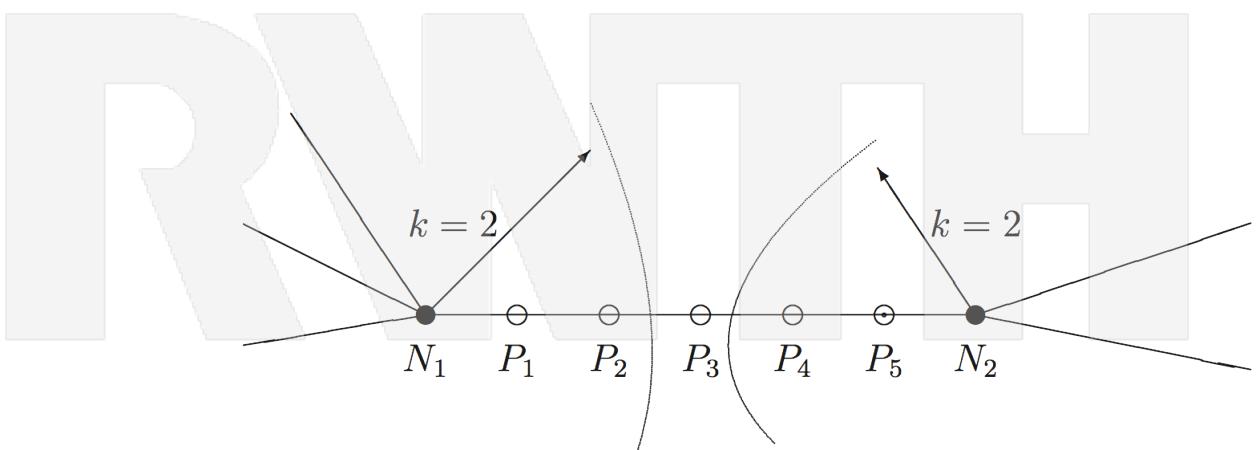


Abbildung 3.9: Fehlerkorrektur $k = 2$ (N=Nutzwort, P=Pseudowort) $d = 6$

$d_m = 1$	Eindeutigkeit
$d_m = 2$	Einzelfehlererkennung
$d_m = 3$	Doppelfehlererkennung / Einzelfehlerkorrektur
$d_m = 4$	Dreifachfehlererkennung / Einzelfehlerkorrektur
$d_m = 5$	Vierfachfehlererkennung / Doppelfehlerkorrektur

Wahrscheinlichkeit für das Auftreten eines $k \leq m$ -fach Fehlers in einem m -stelligen Wort

Sei p die Wahrscheinlichkeit für das Auftreten eines Fehlers an einer Bitstelle. Dann folgt nach der Binomialverteilung:

kein Fehler: $P(0) = (1 - p)^m$

1-fach Fehler: $P(1) = mp(1 - p)^{m-1}$

2-fach Fehler: $P(2) = \frac{1}{2}m(m-1)p^2(1-p)^{m-2}$

.

.

.

k -fach Fehler: $P(k) = \binom{m}{k} p^k (1-p)^{m-k} \quad 0 \leq k \leq m$

mit $p = 0,1\%$ folgt für die Wortlänge $m = 10$:

$P(0) = 0,999^{10} \approx 99\%$

$P(1) = 10 \cdot 0,001 \cdot 0,999^9 \approx 0,01 = 1\%$

$P(2) = \frac{1}{2}10 \cdot 9 \cdot 0,000001 \cdot 0,999^8 \approx 0,000045 = 0,0045\%$

.

.

.

$P(10) = 0,001^{10} = 10^{-30}$

d. h., dass auch bei einer erheblichen Wahrscheinlichkeit für einen Einzelfehler das Auftreten von mehr als einem Fehler pro Wort sehr unwahrscheinlich ist. Wenn also Einzelfehler erkannt werden können, dann ist das Risiko unerkannter Mehrbitfehler sehr gering und u. U. vernachlässigbar.

3.4.2 Quer- und Längsparitätsprüfung

Das Hinzufügen eines Bits verdoppelt die Anzahl der möglichen Codeworte. Zwischen zwei gültigen Codeworten liegt dann immer ein ungültiges, die Redundanz ist 1 bit.

gerade und ungerade Parität

01110110|1 gerade Parität

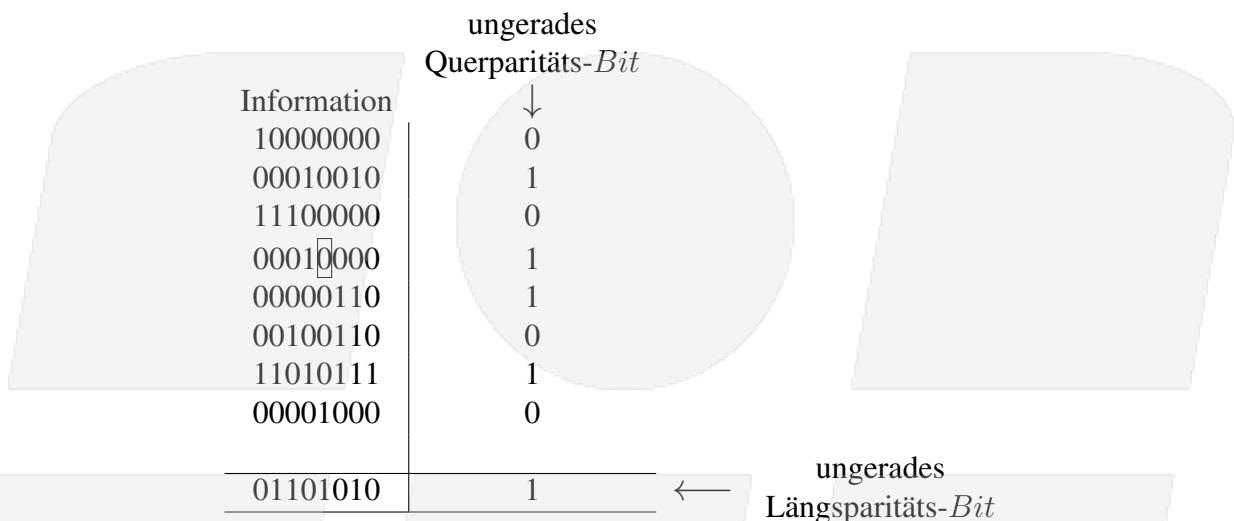
Einsen zu einer geraden Anzahl ergänzen

01110110|0 ungerade Parität

Einsen zu einer ungeraden Anzahl ergänzen

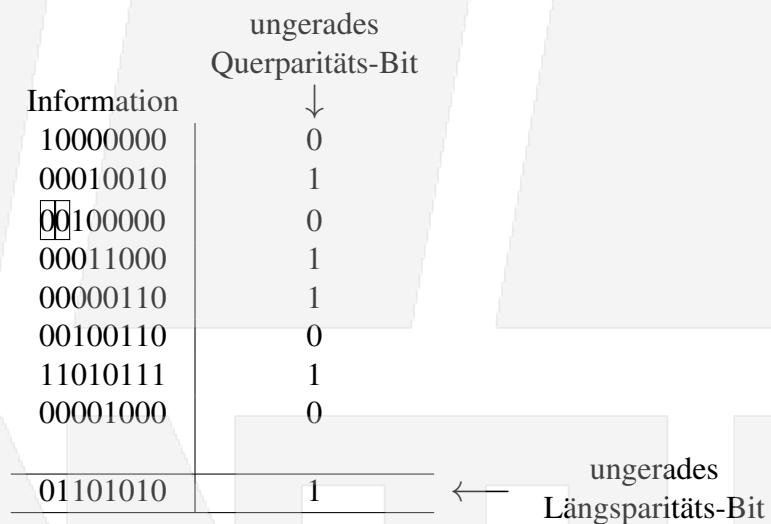
Blockprüfung mit Längs- und Querparitäts-Bit

Einzelfehlerkorrektur



Die eingerahmte Bitstelle wurde bei der Übertragung gestört. Durch die erneute Berechnung der Längs- und Querparitäts-Bits ist die Stelle eindeutig zu identifizieren. Der Einzelfehler kann somit korrigiert werden.

Doppelfehlererkennung



Die eingerahmten Bitstellen wurden bei der Übertragung gestört. Durch die erneute Berechnung der Längs- und Querparitäts-Bits sind die beiden Spalten eindeutig zu identifizieren. Die zugehörige Zeile kann allerdings nicht bestimmt werden. Die Doppelfehler können somit erkannt, aber nicht korrigiert werden.

Bit 9 der Längsparitätsbits gibt die Parität der vorangegangenen Querparitätsbits an. Das letzte übertragene Wort eines Blockes enthält somit ausschließlich Längsparitäten der vorangegangenen Worte.

Kapitel 4

Zahlendarstellung

4.1 Polyadische Zahlensysteme

In Polyadischen Zahlensystemen (Stellenwertsystemen) wird jede Zahl Z nach Potenzen einer Basis $B \in \mathbb{N}$ zerlegt. Ein polyadisches Zahlensystem basiert auf dem B -elementigen Alphabet $\{0, 1, \dots, B - 1\}$. Die Elemente $z_i, 0 \leq i < B$ des Alphabets heißen Ziffern. Abbildung 4.1 zeigt einige Zahlensysteme zusammen mit ihren Alphabeten.

In einem Zahlensystem kann eine ganze Zahl $Z, 0 \leq Z < B^n, n \in \mathbb{N}$, eindeutig durch folgende Summe dargestellt werden:

$$\begin{aligned} Z &= \sum_{i=0}^{n-1} z_i B^i, \quad z_i \in \{0, 1, \dots, B - 1\} \\ &= z_{n-1} \cdot B^{n-1} + z_{n-2} \cdot B^{n-2} + \dots + z_2 \cdot B^2 + z_1 \cdot B^1 + z_0 \cdot B^0 \\ &= z_{n-1} \cdot B^{n-1} + z_{n-2} \cdot B^{n-2} + \dots + z_2 \cdot B^2 + z_1 \cdot B + z_0 \end{aligned}$$

Zur Vereinfachung der Darstellung dient die so genannte Zifferschreibweise:

$$Z = (z_{n-1} z_{n-2} \dots z_2 z_1 z_0)_B$$

Führende Nullen werden in der Regel weggelassen. Außerdem kann die Angabe der Basis B entfallen, wenn sie aus dem Kontext hervorgeht.

Zahlensystem	Basis	Alphabet
Dualsystem	2	{0, 1}
Ternärsystem	3	{0, 1, 2}
Quinärsystem	5	{0, 1, 2, 3, 4}
Oktalsystem	8	{0, 1, 2, 3, 4, 5, 6, 7}
Dezimalsystem	10	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Hexadezimalsystem	16	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

Abbildung 4.1: Polyadische Zahlensysteme

Eine Variante der Darstellung liefert das sog. Hornerschema, welches im nächsten Abschnitt als Hilfsmittel zur Umwandlung von Zahlen eingesetzt wird:

$$Z = (\dots (z_{n-1}B + z_{n-2})B + \dots + z_1)B + z_0$$

In Polyadischen Zahlensystemen werden negative Zahlen in der Regel mit dem Vorzeichen „-“ versehen. Für die Darstellung von negativen Dualzahlen wird oft eine andere Darstellungsart gewählt, auf die im Zusammenhang mit der Dualzahlenarithmetik näher eingegangen wird.

Für die Darstellung rationaler Zahlen bzw. endlicher Brüche kann die Summenschreibweise wie folgt erweitert werden:

$$\begin{aligned} Z &= \sum_{i=-k}^{n-1} z_i B^i, \quad z_i \in \{0, 1, \dots, B-1\}, k \in \mathbb{N}_0 \\ &= z_{n-1} \cdot B^{n-1} + \dots + z_1 \cdot B^1 + z_0 \cdot B^0 + z_{-1} \cdot B^{-1} + \dots + z_{-k} \cdot B^{-k} \end{aligned}$$

Ziffernschreibweise:

$$Z = (z_{n-1} z_{n-2} \dots z_2 z_1 z_0, z_{-1} z_{-2} \dots z_{-k})_B$$

Hornerschema:

$$\dots + z_0 + (z_{-1} + \dots + (z_{-(k-1)} + z_{-k} B^{-1}) B^{-1} \dots) B^{-1}$$

Beispiel:

$$\begin{aligned} 160,464 &= (160,464)_{10} \\ &= 1 \cdot 10^2 + 6 \cdot 10^1 + 0 \cdot 10^0 + 4 \cdot 10^{-1} + 6 \cdot 10^{-2} + 4 \cdot 10^{-3} \\ &= (1 \cdot 10 + 6)10 + 0 + (4 + (6 + 4 \cdot 10^{-1})10^{-1})10^{-1} \end{aligned}$$

In der Informatik spielen das Dualsystem, das Oktalsystem und das Hexadezimalsystem eine besondere Rolle. Aufgrund der Zweiwertigkeit des Dualsystems ist es in Digitalrechnern besonders leicht technisch zu realisieren (Strom an / Strom aus). Mit Hilfe des Oktal- bzw. des Hexadezimalsystems lassen sich drei bzw. vier Dualziffern sehr einfach zu einer einzigen (Oktal-) Hexadezimalziffer umwandeln. Diese Umwandlung ist deshalb so einfach, weil es sich bei den Basen 8 bzw. 16 um Zweierpotenzen handelt. Dualzahlen können also vor allem durch Hexadezimalzahlen wesentlich kompakter dargestellt werden. Aufgrund der Bedeutsamkeit der drei Basen 2, 8 und 16 sind einige ihrer Potenzen in der Abbildung 4.2 aufgeführt.

Ähnlich wie im Dezimalsystem sind auch für große Dualzahlen Abkürzungen gebräuchlich, die in der Größenordnung 10^3 gestaffelt sind. Man beachte den Zusammenhang $2^{n \cdot 10} \sim 10^{n \cdot 3}$:

$$\begin{array}{rclcl} 1 \text{ K} & = & 1024 & = & 2^{10} \sim 10^3 \quad (\text{in Worten: 1 Kilo}) \\ 1 \text{ M} & = & 1048576 & = & 2^{20} \sim 10^6 \quad (\text{in Worten: 1 Mega}) \\ 1 \text{ G} & = & 1073741824 & = & 2^{30} \sim 10^9 \quad (\text{in Worten: 1 Giga}) \\ 1 \text{ T} & = & 1099511627776 & = & 2^{40} \sim 10^{12} \quad (\text{in Worten: 1 Tera}) \end{array}$$

Abbildung 4.3 zeigt noch einmal den Vergleich zwischen den im Zusammenhang mit Digitalrechnern verwendeten Zahlensystemen. Man beachte, dass der BCD-Code keine *Zahlen* kodiert, sondern *Ziffern* von Dezimalzahlen.

Potenz zur Basis B	$B = 2$	$B = 8$	$B = 16$
B^0	1	1	1
B^1	2	8	16
B^2	4	64	256
B^3	8	512	4096
B^4	16	4096	65536
B^5	32	32768	1048576
B^6	64	262144	
B^7	128	2097152	
B^8	256	16777216	
B^9	512	134217728	
B^{10}	1024	1073741824	
B^{11}	2048	8589934592	
B^{12}	4096		
B^{13}	8192		
B^{14}	16384		
B^{15}	32768		
B^{16}	65536		
B^{17}	131072		
B^{18}	262144		
B^{19}	524288		
B^{20}	1048576		
B^{21}	2097152		

Abbildung 4.2: Potenzen von 2, 8 und 16

dezimal	dual	oktal	hexadezimal	BCD
0	0000	0	0	0000
1	0001	1	1	0001
2	0010	2	2	0010
3	0011	3	3	0011
4	0100	4	4	0100
5	0101	5	5	0101
6	0110	6	6	0110
7	0111	7	7	0111
8	1000	10	8	1000
9	1001	11	9	1001
10	1010	12	A	0001 0000
11	1011	13	B	0001 0001
12	1100	14	C	0001 0010
13	1101	15	D	0001 0011
14	1110	16	E	0001 0100
15	1111	17	F	0001 0101
16	10000	20	10	0001 0110

Abbildung 4.3: Die Zahlen 0 bis 16

4.2 Umwandlung in Zahlensysteme mit anderer Basis

Grundsätzlich kann jede Zahl, die in einem bestimmten polyadischen Zahlensystem dargestellt ist, in einem anderen polyadischen Zahlensystem dargestellt werden. In der Regel wandelt man die Zahl zunächst in eine Dezimalzahl um, die dann wiederum in eine Zahl des gewünschten Zielsystems umgewandelt wird:

$$\text{Zahl zur Basis } B_1 \longleftrightarrow \text{Zahl zur Basis } 10 \longleftrightarrow \text{Zahl zur Basis } B_2$$

4.2.1 Umwandlung in Dezimalzahlen

Die *Umwandlung von Zahlen* eines beliebigen polyadischen Zahlensystems in eine Dezimalzahl kann durch einfaches Anwenden der Summenformel $Z = \sum_{i=-k}^{n-1} z_i B^i$ erfolgen.

4.2.2 Umwandlung von Dezimalzahlen

Hier soll lediglich die Umwandlung von Dezimalzahlen in Dualzahlen besprochen werden. Die Umwandlung von Dezimalzahlen in beliebige andere Zahlensysteme funktioniert jedoch prinzipiell gleich.

Der ganzzahlige Anteil und der gebrochene Anteil der Dezimalzahl werden getrennt behandelt.

Umwandlung des ganzzahligen Anteils

Zur Umwandlung des ganzzahligen Anteils bieten sich zwei Methoden an: Die erste Methode basiert auf einer Division durch 2^n und anschließender Subtraktion von 2^n , falls der ganzzahlige Anteil des Quotienten 1 ergab. 2^n wird dabei als größtmögliche Potenz von 2 gewählt, die gerade noch kleiner oder gleich der umzuandelnden Zahl ist. Das Verfahren wird wiederholt, bis $n = 0$ ist. Die zuerst berechnete Ziffer ist dann die höchswertige Ziffer des gesamten Ergebnisses.

Beispiel:

$$\begin{array}{r}
 437 : 256 = 1 \\
 - \underline{256} \\
 181 : 128 = 1 \\
 - \underline{128} \\
 53 : 64 = 0 \\
 53 : 32 = 1 \\
 - \underline{32} \\
 21 : 16 = 1 \\
 - \underline{16} \\
 5 : 8 = 0 \\
 5 : 4 = 1 \\
 - \underline{4} \\
 1 : 2 = 0 \\
 1 : 1 = 1
 \end{array}$$

Ergebnis: $437_{10} = 110110101_2$

Die zweite Methode basiert auf dem Horner-Schema. Hier ist die zuletzt berechnete Ziffer die höchstwertige Ziffer. Das Ergebnis muss also von unten nach oben gelesen werden (div ist die Ganzzahldivision, mod ist die Modulo-Operation):¹

$$\begin{array}{ll}
 Z = Z^{(0)} & = z_0 + B(z_1 + B(z_2 + Bz_3)) \\
 Z^{(0)} \text{ div } B = Z^{(1)} & = z_1 + B(z_2 + Bz_3) \\
 Z^{(1)} \text{ div } B = Z^{(2)} & = z_2 + Bz_3 \\
 Z^{(2)} \text{ div } B = Z^{(3)} & = z_3 \\
 Z^{(3)} \text{ div } B = Z^{(4)} & = 0
 \end{array}
 \quad
 \begin{array}{l}
 Z^{(0)} \text{ mod } B = z_0 \\
 Z^{(1)} \text{ mod } B = z_1 \\
 Z^{(2)} \text{ mod } B = z_2 \\
 Z^{(3)} \text{ mod } B = z_3
 \end{array}$$

Beispiel:

$$\begin{array}{ll}
 437 \text{ div } 2 & = 218, \quad 437 \text{ mod } 2 = 1 \\
 218 \text{ div } 2 & = 109, \quad 218 \text{ mod } 2 = 0 \\
 109 \text{ div } 2 & = 54, \quad 109 \text{ mod } 2 = 1 \\
 54 \text{ div } 2 & = 27, \quad 54 \text{ mod } 2 = 0 \\
 27 \text{ div } 2 & = 13, \quad 27 \text{ mod } 2 = 1 \\
 13 \text{ div } 2 & = 6, \quad 13 \text{ mod } 2 = 1 \\
 6 \text{ div } 2 & = 3, \quad 6 \text{ mod } 2 = 0 \\
 3 \text{ div } 2 & = 1, \quad 3 \text{ mod } 2 = 1 \\
 1 \text{ div } 2 & = 0, \quad 1 \text{ mod } 2 = 1
 \end{array}$$

Ergebnis: $437_{10} = 110110101_2$

Umwandlung des gebrochenen Anteils

Die Umwandlung des gebrochenen Anteils R_Q (Wert der Nachkommaziffernfolge) ist nicht immer exakt möglich. Im Zielsystem können unendlich lange Brüche entstehen, so dass lediglich gefordert wird:

$$R_Q = R_Z + \varepsilon; \quad \varepsilon \geq 0, \text{ minimal}$$

Die Umwandlungsmethode basiert ebenfalls auf dem Horner-Schema. Durch fortgesetzte Multiplikation des gebrochenen Anteils mit der Zielbasis und Abspaltung der jeweils vordersten Ziffer erhält man die Zieldarstellung. Das Ergebnis ist von oben nach unten zu lesen.

$$\begin{aligned}
 R_Q &= (z_{-1} + (z_{-2} + z_{-3}B^{-1})B^{-1})B^{-1} + \varepsilon && \rightarrow z_{-1} \\
 R_Q \cdot B &= z_{-1} + (z_{-2} + z_{-3}B^{-1})B^{-1} + B\varepsilon && \rightarrow z_{-2} \\
 ((z_{-2} + z_{-3}B^{-1})B^{-1} + B\varepsilon) \cdot B &= z_{-2} + z_{-3}B^{-1} + B^2\varepsilon && \rightarrow z_{-3} \\
 (z_{-3}B^{-1} + B^2\varepsilon) \cdot B &= z_{-3} + B^3\varepsilon
 \end{aligned}$$

Beispiel:

¹ Allgemein lässt sich das Verfahren auch so formulieren:

$$\begin{aligned}
 Z^{(0)} &= Z \\
 Z^{(i)} &= \left\lfloor \frac{Z^{(i-1)}}{B} \right\rfloor, \quad \text{für } i = 1, \dots, n \quad \text{mit } n = \min\{i | Z^{(i)} = 0\} \\
 z_{i-1} &= B \left(\frac{Z^{(i-1)}}{B} - Z^{(i)} \right) \text{ für } i = 1, \dots, n
 \end{aligned}$$

0,38	·	2	=	0,76	→	0
0,76	·	2	=	1,52	→	1
0,52	·	2	=	1,04	→	1
0,04	·	2	=	0,08	→	0
0,08	·	2	=	0,16	→	0
0,16	·	2	=	0,32	→	0
0,32	·	2	=	0,64	→	0
0,64	·	2	=	1,28	→	1
						Abbruch

Ergebnis: $0,38_{10} = 0,01100001_2$

4.2.3 Dual-, Oktal- und Hexadezimalzahlen

Die Umwandlung von Dualzahlen in Oktal- oder Hexadezimalzahlen und umgekehrt ist ohne den Umweg über das Dezimalsystem möglich, da die Basen 8 bzw. 16 Zweierpotenzen sind:

Bei der Umwandlung von (Oktal-)Hexadezimalzahlen in Dualzahlen wird jede Ziffer der (Oktal-)Hexadezimalzahl durch genau 3 bzw. 4 Dualziffern ersetzt. Umgekehrt können je 3 bzw. 4 Ziffern einer Dualzahl zu genau einer (Oktal-)Hexadezimalziffer zusammengefasst werden. Dabei ist zu beachten, dass die Gruppierung von Dualziffern am rechten Ende der Dualzahl, also an der niedrigstwertigen Dualziffer beginnen muss. Um vollständige Dreier- bzw. Vierergruppen zu erhalten, müssen ggf. Nullen ergänzt werden. Beispiel:

$$\begin{array}{rcl} 110\ 110\ 101 & , & 100 = 665,4_8 \\ 0001\ 1011\ 0101 & , & 1000 = 1B5,8_{16} \end{array}$$

4.3 Zahlendarstellung im Digitalrechner

Im Digitalrechner sind Zahlen in der Regel als Dualzahlen fester Länge bzw. fester Stellenzahl n gespeichert. Gebräuchlich sind Stellenzahlen von 8 Bits (1 Byte), 16 Bits oder 32 Bits. Für kaufmännische Anwendungen ist auch eine Zahlendarstellung denkbar, in der Dezimalzahlen Ziffer für Ziffer kodiert sind (z. B. BCD-Code). Diese Variante wurde schon bei der Codierung behandelt und wird hier nicht weiter verfolgt.

4.3.1 Darstellung ganzer Zahlen

Mit einer festen Wortlänge von n Bits lassen sich 2^n verschiedene ganze Zahlen darstellen, z. B. der Wertebereich $[0, 2^n - 1]$ oder $[-2^{n-1}, 2^{n-1} - 1]$. Für die Darstellung negativer ganzer Zahlen im Digitalrechner gibt es verschiedene Möglichkeiten:

- Darstellung mit Hilfe des Vorzeichens:

Die naheliegendste Methode zur Darstellung negativer Dualzahlen besteht darin, das linke bzw. höchstwertige Bit als Vorzeichen (z. B. 0: positiv, 1: negativ) zu interpretieren. Diese Darstellungsart hat jedoch zwei Nachteile: Zum einen hat die Zahl 0 mit dieser Methode zwei Darstellungen (+0 und -0), zum anderen muss der Digitalrechner über Hardware bzw. ein Rechenwerk verfügen, welches sowohl addieren als auch subtrahieren kann.

- Darstellung im Einerkomplement:

Mit Hilfe des *Einerkomplements* $K_1(Z)$ kann die Subtraktion auf die Addition zurückgeführt werden. Das Einerkomplement einer Dualzahl Z ergibt sich durch Subtraktion dieser Zahl von $2^n - 1$:

$$K_1(Z) = (2^n - 1) - Z = (11 \dots 1)_2 - Z$$

Die Umwandlung einer Zahl in ihr Einerkomplement kann technisch sehr einfach durch bitweises Komplementieren erreicht werden:

$$K_1(Z) = K_1(\sum_{i=0}^{n-1} z_i 2^i) = \sum_{i=0}^{n-1} (1 - z_i) 2^i = \sum_{i=0}^{n-1} \bar{z}_i 2^i$$

Beispiel: $K_1(00110101) = 11001010$

Auch im Einerkomplement existieren zwei Darstellungen für die Zahl 0, nämlich $(00 \dots 0)_2$ und $(11 \dots 1)_2$.

- Darstellung im Zweierkomplement:

Das *Zweierkomplement* einer Dualzahl Z ergibt sich durch Subtraktion dieser Zahl von 2^n :

$$K_2(Z) = 2^n - Z = K_1(Z) + 1$$

Beispiel: $K_2(00110101) = K_1(00110101) + 1 = 11001010 + 1 = 11001011$

Die Umwandlung einer Zahl in ihr Zweierkomplement geschieht also durch Umwandlung in ihr Einerkomplement und der anschließenden Addition einer 1. Im Zweierkomplement ist jede Darstellung einer Zahl eindeutig.

Man beachte, dass das Zweierkomplement eines Zweierkomplements wieder die Zahl in Normaldarstellung ergibt:

$$K_2(K_2(Z)) = K_2(2^n - Z) = 2^n - K_2(Z) = 2^n - (2^n - Z) = Z$$

Abbildung 4.4 stellt die verschiedenen Darstellungsmöglichkeiten für ein Byte lange Dualzahlen noch einmal zusammen.

Ganzzahlenarithmetik

- **Addition**

Die duale Addition wird prinzipiell genauso wie die dezimale Addition durchgeführt. Abbildung 4.5 zeigt eine Übersicht aller möglichen Fälle, die bei der Addition von zwei Dualziffern unter Berücksichtigung von Überträgen auftreten können.

Beispiel:

$$\begin{array}{r}
 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & \hat{=} & 53 \\
 + & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \hat{=} & 28 \\
 \text{Übertrag} & 1 & 1 & 1 & 1 \\
 \hline
 \text{Summe} & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & \hat{=} & 81
 \end{array}$$

- **Subtraktion**

Abbildung 4.6 zeigt eine Übersicht aller möglichen Fälle, die bei der Subtraktion von zwei Dualziffern unter Berücksichtigung der Überträge auftreten können.

Beispiel:

$$\begin{array}{r}
 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & \hat{=} & 53 \\
 - & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \hat{=} & 28 \\
 \text{Borger} & & & 1 & 1 & & & & & \\
 \hline
 \text{Summe} & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & \hat{=} & 25
 \end{array}$$

- **Subtraktion durch Addition des Zweierkomplements**

Es wurde schon darauf hingewiesen, dass die Subtraktion unter Verwendung des Zweierkomplements durchgeführt werden kann:

$$Z_1 - Z_2 = Z_1 - Z_2 + 2^n = Z_1 + (2^n - Z_2) = Z_1 + K_2(Z_2)$$

Man beachte, dass die Addition von 2^n das Ergebnis nicht verändert, da 2^n links vom höchstwertigen Bit steht. Ein positives Ergebnis ist am Übertrag zu erkennen. Ist das Ergebnis negativ, fehlt der Übertrag und das höchstwertige Bit ist gleich 1. Ein negatives

Dezimalzahl	Vorzeichen	Einerkomplement	Zweierkomplement
+128	n. darstellbar	n. darstellbar	n. darstellbar
+127	0111 1111	0111 1111	0111 1111
+93	0101 1101	0101 1101	0101 1101
+1	0000 0001	0000 0001	0000 0001
+0	0000 0000	0000 0000	0000 0000
-0	1000 0000	1111 1111	0000 0000
-1	1000 0001	1111 1110	1111 1111
-93	1101 1101	1010 0010	1010 0011
-127	1111 1111	1000 0000	1000 0001
-128	n. darstellbar	n. darstellbar	1000 0000

Abbildung 4.4: Ganzzahlige Dualzahlen in verschiedenen Darstellungen

erster Summand	zweiter Summand	Übertrag von rechts	Summe	Übertrag nach links
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Abbildung 4.5: Addition von Dualziffern

Ergebnis kann durch erneute Bildung des Zweierkomplements in Normaldarstellung gebracht werden.

Beispiel für ein positives Ergebnis:

$$\begin{array}{r}
 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
 + & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
 \text{Übertrag} & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
 \hline
 (1) & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1
 \end{array}
 \stackrel{\hat{=}}{\quad} 53$$

$$\begin{array}{r}
 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
 \hline
 & 25
 \end{array}
 \stackrel{\hat{=}}{\quad} -28$$

Beispiel für ein negatives Ergebnis:

$$\begin{array}{r}
 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
 + & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
 \hline
 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1
 \end{array}
 \stackrel{\hat{=}}{\quad} 28$$

$$\begin{array}{r}
 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
 \hline
 & Z
 \end{array}
 \stackrel{\hat{=}}{\quad} -53$$

$$\begin{array}{r}
 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
 + & & & & & & & 1 \\
 \hline
 & K_1(Z)
 \end{array}$$

$$\begin{array}{r}
 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & K_2(Z) & = & 25
 \end{array}$$

$$\Rightarrow Z = -25$$

Zum besseren Verständnis sei erwähnt, dass die Subtraktion durch Addition des Komplements nicht auf Dualzahlen beschränkt ist. Auch bei der dezimalen Subtraktion kann diese Methode angewendet werden (10-er Komplement):

$$\begin{array}{rcl}
 53 - 28 & = & 53 + (100 - 28) = 53 + 72 = (1)25 \\
 28 - 53 & = & 28 + (100 - 53) = 28 + 47 = (0)75 \quad K_{10}(75) = -25
 \end{array}$$

• Multiplikation

Die Multiplikation erfolgt durch fortgesetzte Addition und Verschiebung.

Minuend	Subtrahend	Übertrag von rechts	Differenz	Borger von links
0	0	0	0	0
1	0	0	1	0
0	1	0	1	1
1	1	0	0	0
0	0	1	1	1
1	0	1	0	0
0	1	1	0	1
1	1	1	1	1

Abbildung 4.6: Subtraktion von Dualziffern

Beispiel:

$$\begin{array}{r}
 1\ 1\ 0\ 1 \cdot 1\ 0\ 1\ 1 \\
 \hline
 & 1\ 1\ 0\ 1 \\
 & 1\ 1\ 0\ 1 \\
 0\ 0\ 0\ 0 \\
 \hline
 1\ 1\ 0\ 1 \\
 \hline
 1\ 0\ 0\ 0\ 1\ 1\ 1
 \end{array}$$

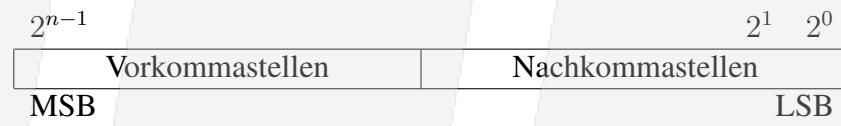
- **Division**

Die Division erfolgt durch fortgesetzte Subtraktion und Verschiebung. Beispiel:

$$\begin{array}{r}
 1\ 1\ 0\ 0\ 1\ 0 : 1\ 0\ 1 = 1\ 0\ 1\ 0 \\
 1\ 0\ 1 \\
 \hline
 1\ 0\ 1 \\
 1\ 0\ 1 \\
 \hline
 0\ 0 \\
 0 \\
 \hline
 0
 \end{array}$$

4.3.2 Festkomma-Darstellung

In der Festkomma-Darstellung wird eine reelle Zahl durch eine n -stellige Dualzahl repräsentiert, wobei das Komma an beliebiger, fester Stelle steht:



Die Kürzel LSB und MSB stehen für least bzw. most significant bit. Die Darstellung ganzer Zahlen ist ein Spezialfall der Festkommadarstellung, in der das Komma rechts vom LSB steht. Negative Zahlen können als Komplement oder auch mit Vorzeichen dargestellt werden. Mit Vorzeichen ergibt sich folgende Zahlendarstellung:



Abbildung 4.7 liefert eine Übersicht über die Zahlenbereiche für 16 Bit lange Festkomma-Darstellungen mit Vorzeichen.

Festkommaarithmetik

Die Addition und die Subtraktion verläuft in der gleichen Weise wie bei der Ganzzahlenarithmetik. Bei der Multiplikation und der Division muss die Kommaverschiebung berücksichtigt werden.

	Komma	
	rechts vom LSB	links vom MSB
größte pos. Zahl	$2^{15} - 1 = 32767$	$0,999969482 = (1 - 2^{-15})$
kleinste pos. Zahl	$2^0 = 1$	$0,000030517 = (2^{-15})$
kleinste Zahl	$-(2^{15} - 1) = -32767$	$-0,999969482 = (-(1 - 2^{-15}))$

Abbildung 4.7: Zahlenbereiche für n=16 Bit lange Festkomma-Darstellung mit VZ

Probleme der Festkomma-Darstellung

Mit Zahlen in Festkommadarstellung kann zwar sehr schnell gerechnet werden, da die Hardware eines Digitalrechners Festkommaoperationen direkt ausführen kann, sie besitzt jedoch den gravierenden Nachteil, dass der darstellbare Zahlenbereich sehr beschränkt ist. So ist z. B. in einem Festkomma-Format der Länge 16, in dem das Komma links neben dem viertletzten Bit angenommen wird, die Zahl $(0,0001111)_2$ nicht mehr hinreichend genau darstellbar, obwohl eigentlich 16 Bit für ihre Darstellung ausreichen würden. Wegen der festen Position des Kommas wird sie jedoch als $(000000000000,0001)_2$ dargestellt. Der meiste Platz wird hier für die führenden Nullen benötigt.

4.3.3 Gleitkomma-Darstellung

Im Gleitkomma-Format wird der Zahlenbereich durch eine halblogarithmische Darstellung erweitert. Die Gleitkomma-Darstellung einer Zahl Z hat die Form:

$$Z = M \cdot B^E$$

M bezeichnet die Mantisse, E den Exponenten der Gleitkommazahl Z . B ist eine beliebige Basis. Im Digitalrechner gilt in der Regel $B = 2$. Die Darstellung einer Zahl im Gleitkomma-Format ist nicht eindeutig, wie folgende Beispiele zeigen:

$$\begin{array}{rcl} 6,75_{10} & = & 6,75_{10} \cdot 10^{0_{10}} = 0,675_{10} \cdot 10^{1_{10}} = 675,0_{10} \cdot 10^{-2_{10}} \\ 110,11_2 & = & 110,11_2 \cdot 2^{0_2} = 0,11011_2 \cdot 2^{11_2} = 11011,0_2 \cdot 2^{-10_2} \end{array}$$

Um eine eindeutige Darstellung zu erhalten, kann eine Gleitkommazahl normalisiert werden. Es existieren mehrere Möglichkeiten der Normalisierung. Man kann sich z. B. darauf einigen, dass für die Mantisse $0 \leq M < 1$ gelten soll und ihre erste Nachkommastelle von Null verschieden sein muss. Bezogen auf obige Beispiele sind also die Darstellungen in der vorletzten Spalte normalisiert.

Im Digitalrechner können Gleitkommazahlen z. B. in folgendem Format abgespeichert werden:

2^{n-1}	$2^1 \quad 2^0$	
VZ	Betrag der Mantisse	Charakteristik

Das Komma der Mantisse und die Basis werden nicht explizit gespeichert. Während für die Basis üblicherweise $B = 2$ gilt, steht das Komma wie bei der Festkomma-Darstellung an beliebiger, aber fester Stelle. VZ bezeichnet das Vorzeichen der Mantisse.

Statt des Exponenten wird in den meisten Computern die sog. Charakteristik C benutzt. Die Charakteristik ergibt sich aus der Addition des Exponenten E mit einem konstanten Offset, der sich aus der Anzahl c der für die Charakteristik reservierten Bits errechnen lässt:

$$C = E + \left(\frac{1}{2}B^c - 1\right)$$

Durch die Verwendung der Charakteristik erreicht man, dass im Exponenten lediglich mit positiven Zahlen gerechnet werden muss. Im Gegensatz zum Zweierkomplement kann so ein bitweiser Vergleich zweier Zahlen realisiert werden.

Das IEEE-Gleitkomma-Format 754

Am Beispiel des weit verbreiteten IEEE-Formats 754 soll der Aufbau einer Gleitkomma-Darstellung noch einmal verdeutlicht werden:

- Das Format hat eine Länge von 32 Bits, die wie folgt aufgeteilt sind:

31	30	23	22	0
v	c c c c c c c	m m		Mantisse

VZ Char.

- $B = 2$
- $C = E + (2^7 - 1)$
- $C = 0$ und $M = 0$ signalisieren, dass der Zahlenwert Null ist
- $C = 255$ signalisiert, dass der Zahlenwert ungültig ist (NAN, „not a number“, oder $\pm\infty$)
- Das Komma der Mantisse steht links vom MSB der Mantisse, also links von Bit 22.
- Vorzeichen der Mantisse: VZ=0 entspricht +, VZ=1 entspricht –
- Normalisierung:

Die normalisierte Darstellung besitzt eine Mantisse der Form $(1, \dots)_2$, d.h. die Mantisse beginnt immer mit einer 1. Deshalb kann diese 1 in der Darstellung weggelassen werden (hidden bit). Eine Sonderstellung nimmt die Zahl 0 ein. Definitionsgemäß besteht die Darstellung der Gleitkommazahl 0 ausschließlich aus Nullen.

Zusammenfassend ergibt sich der Wert einer im IEEE-Format abgespeicherten Gleitkommazahl zu

$$Z = (-1)^{VZ} \cdot (1, 0 + M) \cdot 2^{C-(2^7-1)}.$$

Die Tatsache, dass das IEEE-Format mit einer Charakteristik statt mit Exponenten arbeitet — in Verbindung mit der Anordnung Vorzeichen-Charakteristik-Mantisse (Reihenfolge abnehmender Signifikanz) — führt zu der interessanten Beobachtung, dass ein Vergleich zweier Zahlen einfach bitweise von links nach rechts erfolgen kann. Abbildung 4.8 enthält die Gleitkomma-darstellung einiger Zahlen im IEEE-Format. Die ersten beiden Zeilen enthalten die größte bzw. kleinste positive Zahl, die man im IEEE-Format darstellen kann.

Bemerkung:

Die Interpretation der Darstellung einer Gleitkommazahl im Digitalrechner als normale ganze Zahl wird als „Maschinenwort“ bezeichnet und häufig in hexadezimaler Schreibweise angegeben.

hexadez.	Maschinenwort dual	Gleitkommazahl	
		dual	dezimal
7FFF FFFF	0 11111111 <u>11...1</u> <u>23</u>	NAN	NAN
0000 0000	0 00000000 <u>00...0</u> <u>23</u>	0	0
7F7F FFFF	0 11111110 <u>11...1</u> <u>23</u>	1,1 <u>11...1</u> · 2^{127}	$\approx 3,4 \cdot 10^{38}$
3F80 0000	0 01111111 <u>00...0</u> <u>23</u>	1,0 <u>00...0</u> · 2^0	1
BF80 0000	1 01111111 <u>00...0</u> <u>23</u>	-1,0 <u>00...0</u> · 2^0	-1
40D8 0000	0 10000001 10110 <u>0...0</u> <u>19</u>	1,10110 <u>0...0</u> · 2^2	6,75

Abbildung 4.8: Gleitkommazahlen im IEEE-Format 754

Gleitkommaarithmetik

- **Addition und Subtraktion**

Bei der Addition/Subtraktion von Gleitkommazahlen muss darauf geachtet werden, dass die Exponenten der beiden Summanden gleich sind:

1. Angleichung der Exponenten durch Verschiebung des Exponenten des betragmäßig kleineren Summanden
2. Addition bzw. Subtraktion der Mantissen
Die Subtraktion kann wie in der Ganzzahlenarithmetik mit Hilfe der Addition des Zweierkomplements durchgeführt werden.
3. Normalisierung des Ergebnisses

- **Multiplikation und Division**

1. Multiplikation bzw. Division der Mantissen
2. Addition bzw. Subtraktion der Exponenten
3. Vorzeichen setzen
4. Normalisierung des Ergebnisses

- **Beispiel**

Gegeben sei ein Gleitkomma-Format der Länge 32, das wie folgt aufgeteilt ist: Bits 0 - 22 Mantisse, Bits 23 - 30 Charakteristik (= Exponent + $(2^7 - 1)$), Bit 31 Vorzeichen der Mantisse. Das Komma der Mantisse M steht links von Bit 22, $0 \leq M < 1$. In der normalisierten Darstellung ist Bit 22 für von Null verschiedene Zahlen stets gesetzt². Die Subtraktion soll über die Addition des Zweierkomplements durchgeführt werden.

² Anders als beim IEEE-754-Format steht die Normalisierungs-Eins in diesem Formatbeispiel immer an der ersten Nachkommastelle. Links vom Komma stets damit stets eine Null, damit $0 \leq M < 1$ erfüllt ist.

Folgender arithmetischer Ausdruck soll nun berechnet werden:

$$\frac{2}{9} \cdot 36 - 8$$

1. Konvertierung und Normalisierung der Dezimalzahlen 2 und 9:

VZ	Char.	Mantisse
$2_{10} = 10_2 = 0,1_2 \cdot 2^2$	0	100 0000 1 100 0000 0000 0000 0000 0000
$9_{10} = 1001_2 = 0,1001_2 \cdot 2^4$	0	100 0001 1 100 1000 0000 0000 0000 0000

2. Division:

Division der Mantissen

$$0,1 : 0,1001 = 0,111000111000 \dots$$

$$\begin{array}{r}
 10000 \\
 1001 \\
 \hline
 1110 \\
 1001 \\
 \hline
 1010 \\
 1001 \\
 \hline
 1
 \end{array}$$

$$\text{Subtraktion der Exponenten: } 2 - 4 = -2$$

Das Ergebnis liegt schon in normalisierter Form vor.

VZ	Char.	Mantisse
$0,111000 \dots \cdot 2^{-2}$	0	011 1110 1 111 0001 1100 0111 0001 1100

3. Konvertierung und Normalisierung der Dezimalzahl 36:

VZ	Char.	Mantisse
$36_{10} = 100100_2 = 0,1001_2 \cdot 2^6$	0	100 0010 1 100 1000 0000 0000 0000 0000

4. Multiplikation:

Multiplikation der Mantissen (durch fortgesetzte Addition)

$$0,11100011100011100011100 \cdot 0,1001$$

$$\begin{aligned}
 &= 0,11100011100011100011100 \cdot 2^{-1} \\
 &\quad + 0,11100011100011100011100 \cdot 2^{-4} \\
 &= 0,11100011100011100011100 \cdot 2^{-1} \\
 &\quad + 0,00011100011100011100011 \cdot 2^{-1} \\
 &= 0,111111111111111111111111 \cdot 2^{-1}
 \end{aligned}$$

$$\text{Addition der Exponenten: } -2 + 6 - 1 = 3$$

Das Ergebnis liegt schon in normalisierter Form vor.

VZ	Char.	Mantisse
$0,111111 \dots \cdot 2^3$	0	100 0001 0 111 1111 1111 1111 1111 1111

5. Konvertierung und Normalisierung der Dezimalzahl 8:

VZ	Char.	Mantisse
$8_{10} = 1000_2 = 0,1_2 \cdot 2^4$	0	100 0001 1 100 0000 0000 0000 0000 0000

6. Subtraktion:

Angleichung des Exponenten des Zwischenergebnisses

VZ	Char.	Mantisse
0,111... $\cdot 2^3$	0	100 0001 1 011 1111 1111 1111 1111

Bildung des Zweierkomplements der Mantisse von 8

$$K_2(1000...) = K_1(1000...) + 1 = 0111... + 1 = 1000...$$

In diesem speziellen Fall ist das Zweierkomplement also mit dem Ausgangswert identisch.

Addition der Mantissen

$$\begin{array}{r} 0,01111111... \\ +0,10000000... \\ \hline 0,11111111... \end{array}$$

Das Ergebnis ist negativ (zu erkennen am fehlenden Übertrag), also muss erneut das Zweierkomplement gebildet werden.

$$K_2(1111...) = K_1(1111...) + 1 = 0000... + 1 = 0000...01$$

$$\underline{-2^{-19}} = -0,000...01 \cdot 2^4$$

VZ	Char.	Mantisse
1	100 0001 1	000 0000 0000 0000 0000 0001

Normierung:

VZ	Char.	Mantisse
1	011 0110 1	100 0000 0000 0000 0000 0000

Probleme der Gleitkomma-Darstellung

- Obiges Beispiel weist bereits auf ein elementares Problem der Gleitkomma-Darstellung hin. Man erwartet, dass der Computer als Ergebnis der Berechnung des Ausdrucks eine 0 ausgibt. Stattdessen ist das Resultat eine, wenn auch betragsmäßig sehr kleine, so doch von 0 verschiedene Zahl. Die Ursache dieser Ungenauigkeit liegt darin, dass der Digitalrechner nur endlich viele reelle Zahlen darstellen kann. Im Beispiel ist das Ergebnis der Division, also die Zahl $2/9$, nicht mit endlich vielen Bits als Gleitkommazahl darstellbar.

In Computerprogrammen sollten deshalb Gleitkommazahlen nie auf Gleichheit, z.B. „ $Z = 0?$ “ abgefragt werden, sondern in der Form „ $|Z| < \varepsilon$ “¹. Dabei muss ε eine kleine, im verwendeten Gleitkomma-Format gerade noch darstellbare Zahl sein.

- Weitere Ungenauigkeiten bei der Gleitkommaarithmetik ergeben sich, wenn Operationen auf zwei Zahlen sehr unterschiedlicher Größe ausgeführt werden. Die Addition einer sehr großen mit einer sehr kleinen Zahl führt z.B. dazu, dass bei der Angleichung der Exponenten signifikante Stellen der kleinen Zahl verloren gehen („Auslösungsproblem“). Dadurch können unerwartete Effekte auftreten. So ist das Assoziativgesetz $(Z_1 + Z_2) + Z_3 = Z_1 + (Z_2 + Z_3)$ für Gleitkommazahlen nicht mehr uneingeschränkt gültig. Man betrachte z.B. die Summe

$$\sum_{i=1}^n \frac{1}{i}$$

für hinreichend großes n . Die Berechnung von $\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{2} + 1$ liefert ein wesentlich genaueres Ergebnis als die Berechnung $1 + \frac{1}{2} + \dots + \frac{1}{n-1} + \frac{1}{n}$, da im letzteren Fall die anfänglichen Summanden sehr groß sind und die Addition der kleinen Summanden das Ergebnis nicht mehr ändert.

- **Beispiel** Die Genauigkeitsproblematik bei der Gleitkommaarithmetik soll anhand der Addition der Zahlen $-23, 25_{10}$ und 113_{10} vorgeführt werden. Dabei sei das folgende Maschinenwortformat gegeben:

VZ	Mantisse	Charakteristik
15	14 7	6 0

$VZ = 0 \rightarrow$ positiv

$VZ = 1 \rightarrow$ negativ

Das Komma steht vor Bit 14

Charakteristik = Exponent + $2^6 - 1$

Es gibt keine Normalisierung, d. h. keine Vorschrift bzgl. der Mantissendarstellung

1. Konvertierung und Normalisierung der Dezimalzahlen:

$$113_{10} = 0111\ 0001_2 = 0,1110\ 001 \cdot 2^7$$

$$\text{Charakteristik} = \text{Exponent} + 2^6 - 1 = 7_{10} + 2^6 - 1 = 100\ 0110$$

Darstellung der Dezimalzahl 113_{10} als Maschinenwort:

VZ	Mantisse	Charakteristik
0	111 0001 0	100 0110

$$-23, 25_{10} = -1\ 0111, 01_2 = -0,1011\ 101 \cdot 2^5$$

Angleichung der Exponenten durch Verschiebung der Mantisse des Summanden mit kleinerem Exponenten ergibt:

$$-0,1011\ 101 \cdot 2^5 = -0,\underbrace{0010\ 1110}_\text{8BitMantisse}\ 1 \cdot 2^7$$

Da nur 8 Bit für die Mantisse zur Verfügung stehen, wird hier die letzte Ziffer abgeschnitten. Darstellung der Dezimalzahl $-23, 25_{10}$ als Maschinenwort:

VZ	Mantisse	Charakteristik
1	001 0111 0	100 0110

2. Addition der Mantissen

Die Addition von $-23, 25_{10}$ wird mit Hilfe des Zweierkomplements durchgeführt.

- Bildung des Zweierkomplements der Dualdarstellung von $-23, 25_{10}$:

$$K_1(0010\ 1110) = 1101\ 0001$$

$$K_2 = K_1 + 1 = 1101\ 0010$$

- Addition:

$$\begin{array}{r}
 1110\ 0010 \\
 +1101\ 0010 \\
 \hline
 (1)1011\ 0100
 \end{array}$$

Der Überlauf zeigt ein positives Ergebnis an.

3. Darstellung des Ergebnisses

VZ	Mantisse	Charakteristik
0	101 1010 0	100 0110

Das Ergebnis liegt bereits in normalisierter Form vor.

$$0,1011\ 0100 \cdot 2^7 = 101\ 1010,0_2 = 90_{10}$$

Die mit absoluter Genauigkeit durchgeführte dezimale Addition liefert:

$$113,0_{10} - 23,25_{10} = 89,75_{10}$$

Der Fehler entsteht durch das Abschneiden der letzten Ziffer bei der Anpassung des Exponenten.

$$-23,25_{10} = -0,0010\ 1110 \underbrace{1}_{\cdot 2^7} \cdot 2^7$$

Kapitel 5

Schaltungslogik

5.1 Zwecke und Ziele

Die Schaltungslogik bildet die theoretische Grundlage für den Schaltungsentwurf. Die Schaltungslogik lässt sich aufteilen in die Boolesche Algebra und die auf ihr basierenden Verfahren zur Minimierung boolscher Funktionen. Ohne diese Verfahren wäre keine effiziente Entwicklung logischer Schaltungen möglich.

5.2 Boolesche Algebra

Sei B eine Menge von Elementen, über der zwei zweistellige Operationen Φ und Ψ erklärt sind. Das Quadrupel (B, Φ, Ψ, \neg) ist genau dann eine *Boolesche Algebra* (George Boole, 1815 – 1864, englischer Mathematiker), wenn für beliebige Elemente $a, b, c \in B$ folgende Axiome (die sog. Huntingtonschen Axiome) gelten:

1. Kommutativgesetz

$$a\Phi b = b\Phi a$$

$$a\Psi b = b\Psi a$$

2. Distributivgesetz

$$a\Phi(b\Psi c) = (a\Phi b)\Psi(a\Phi c)$$

$$a\Psi(b\Phi c) = (a\Psi b)\Phi(a\Psi c)$$

3. Neutrale Elemente

Die Menge B enthält ein Nullelement (n) und ein Einselement (e), d.h. für alle $a \in B$ gilt:

$$a\Phi e = a$$

$$a\Psi n = a$$

4. Komplement

Die Menge B enthält zu jedem Element $a \in B$ auch das Komplement $\bar{a} \in B$

$$a\Psi \bar{a} = e$$

$$a\Phi \bar{a} = n$$

Dieses Axiomensystem gilt zunächst ohne inhaltliche Interpretation. Aus den Axiomen können folgende Sätze abgeleitet werden:

Assoziativgesetz

$$\begin{aligned}(a\Phi b)\Phi c &= a\Phi(b\Phi c) \\ (a\Psi b)\Psi c &= a\Psi(b\Psi c)\end{aligned}$$

Idempotenzgesetz

$$\begin{aligned}a\Phi a &= a \\ a\Psi a &= a\end{aligned}$$

Absorptionsgesetz

$$\begin{aligned}a\Psi(a\Phi b) &= a \\ a\Phi(a\Psi b) &= a\end{aligned}$$

DeMorgansche Gesetze

$$\begin{aligned}\overline{(a\Psi b)} &= \bar{a}\Phi\bar{b} \\ \overline{(a\Phi b)} &= \bar{a}\Psi\bar{b}\end{aligned}$$

In einer Booleschen Algebra gilt das Dualitätsprinzip, d.h. zu jeder Aussage, die sich aus den vier Axiomen ableiten lässt, erhält man eine duale Aussage durch Vertauschen der Operationen Φ und Ψ sowie der neutralen Elemente n und e .

5.3 Beispiele Boolescher Algebren

5.3.1 Mengenalgebra

Die Potenzmenge einer Menge M bildet eine Boolesche Algebra, wenn man die Operation Ψ als die Vereinigung \cup und die Operation Φ als den Durchschnitt \cap interpretiert. Die leere Menge \emptyset und die Allmenge M bezeichnen dann das Null- bzw. Einselement.

Beispiel:

Sei $M = \{e_1, e_2, e_3\}$. Dann ist mit

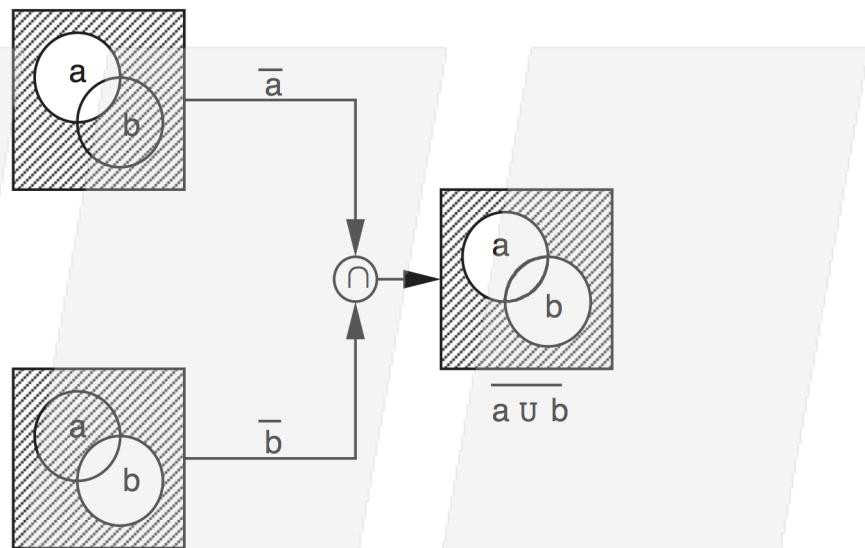
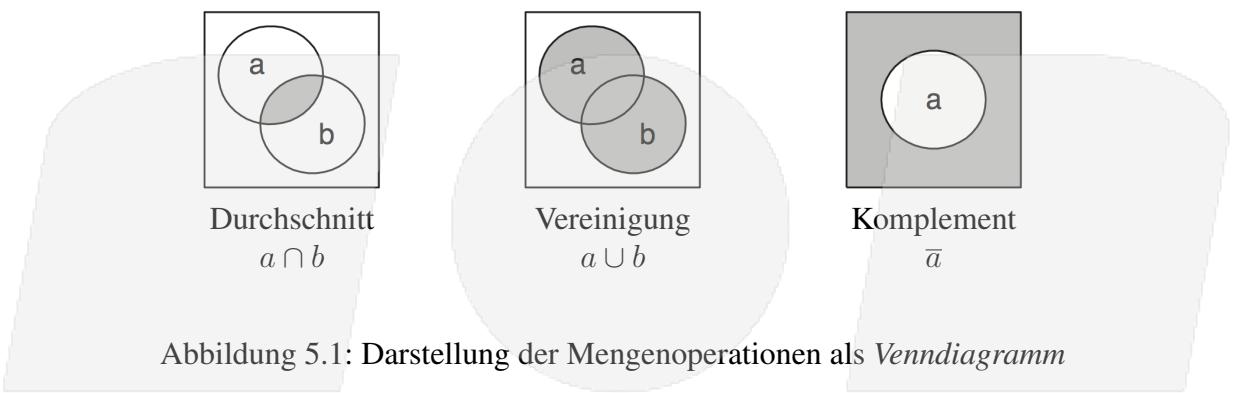
$$\begin{aligned}B &= \{\emptyset, \{e_1\}, \{e_2\}, \{e_3\}, \{e_1, e_2\}, \{e_1, e_3\}, \{e_2, e_3\}, \{e_1, e_2, e_3\}\} \\ &= \{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7\}\end{aligned}$$

das Quadrupel (B, Φ, Ψ, \neg) eine Boolesche Algebra. Es gilt:

$$\begin{aligned}\overline{m_0} &= m_7 = e & \overline{m_1} &= m_6 & \overline{m_2} &= m_5 & \overline{m_3} &= m_4 \\ \overline{m_4} &= m_3 & \overline{m_5} &= m_2 & \overline{m_6} &= m_1 & \overline{m_7} &= m_0 = n\end{aligned}$$

Die Gültigkeit der Gesetze, z.B. des Distributivgesetzes, kann exemplarisch nachgewiesen werden:

$$\begin{aligned}m_2 \cap (m_4 \cup m_6) &= \{e_2\} \cap (\{e_1, e_2\} \cup \{e_2, e_3\}) \\ &= \{e_2\} \cap \{e_1, e_2, e_3\} \\ &= \{e_2\} \\ &= \{e_2\} \cup \{e_2\} \\ &= (\{e_2\} \cap \{e_1, e_2\}) \cup (\{e_2\} \cap \{e_2, e_3\}) \\ &= (m_2 \cap m_4) \cup (m_2 \cap m_6)\end{aligned}$$



Sätze der Mengenalgebra können mit Hilfe von Venndiagrammen bewiesen werden (J. Venn, 1834 – 1923, englischer Logiker):

Da die Elemente von B ebenfalls Mengen sind, können diese durch Flächenstücke dargestellt werden. Vereinigung und Durchschnitt lassen sich bei einer geringen Anzahl von Eingangsvariablen übersichtlich anschaulich zuordnen (Abb. 5.1). Für die Komplementbildung sei die volle Menge oder Allmenge durch ein Quadrat repräsentiert ($a \Psi \bar{a} = e$ bzw. $a \cup \bar{a} = M$!). Abbildungen 5.2 und 5.3 zeigen die Beweise der DeMorganschen Regel und des Assoziativgesetzes mit Hilfe von Venndiagrammen.

Man beachte, dass Beweise mit Hilfe von Venndiagrammen nicht nur für Mengenalgebren, sondern für jede Boolesche Algebra Gültigkeit besitzen, denn zu jeder Booleschen Algebra existiert auch eine Mengenalgebra mit derselben Anzahl von Elementen, in der für die Operationen \cup und \cap dieselben Gesetze gelten wie für die Operationen Ψ und Φ der Booleschen Algebra (Isomorphie).

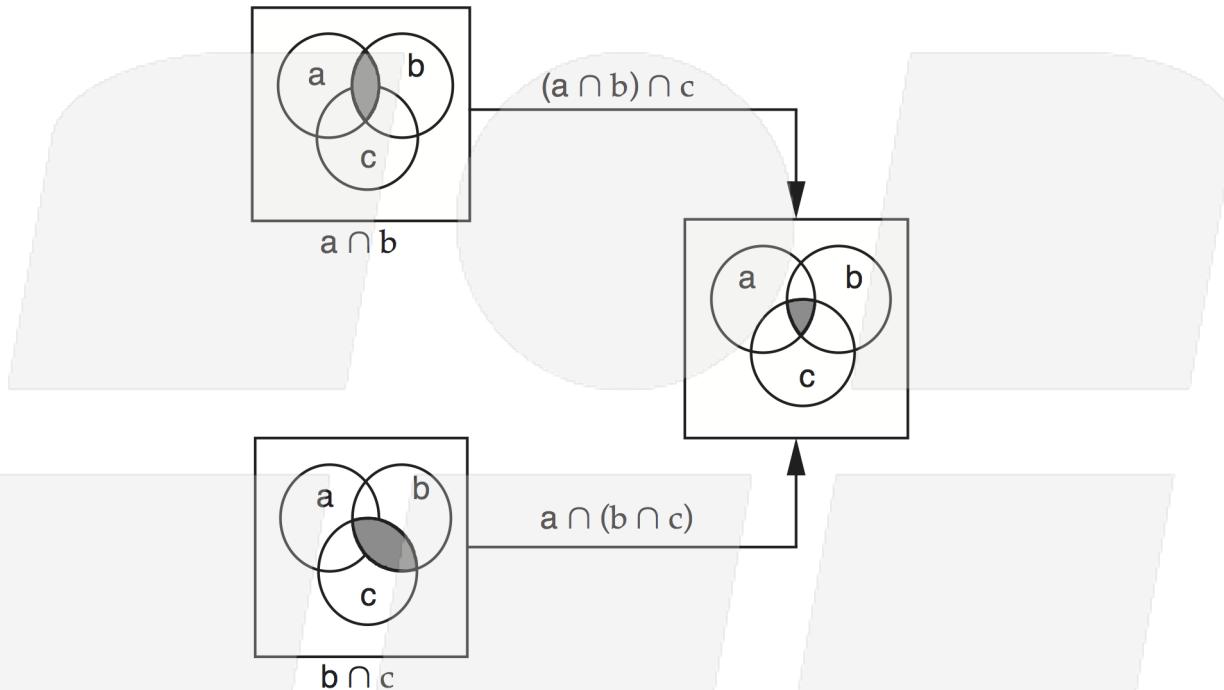


Abbildung 5.3: Beweis des Assoziativgesetzes mit Venndiagrammen

5.3.2 Schaltalgebra

Die Aussagenalgebra, im technischen Kontext *Schaltalgebra* genannt, ist eine spezielle Boolesche Algebra, die die zweielementige Menge $B = \{0, 1\}$ benutzt. Die Operation Ψ wird dabei als Oder-Verknüpfung bzw. Disjunktion ($+$, \vee) und die Operation Φ als Und-Verknüpfung bzw. Konjunktion (\cdot , \wedge) erklärt. Die beiden Elemente $e = 1$ und $n = 0$ werden als Wahrheitswerte „wahr“ bzw. „falsch“ beliebiger Aussagen interpretiert. Die Komplementbildung bezeichnet man in der Schaltalgebra auch als Negation (\bar{a} , $\neg a$).

Die Huntingtonschen Axiome für die Schaltalgebra lassen sich wie folgt schreiben:

Kommutativgesetz

$$a \cdot b = b \cdot a$$

$$a + b = b + a$$

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

$$a + (b \cdot c) = (a + b) \cdot (a + c)$$

$$a \cdot 1 = a$$

$$a + 0 = a$$

Distributivgesetz

Neutrale Elemente

Komplement

$$a + \bar{a} = 1$$

$$a \cdot \bar{a} = 0$$

Aus den Axiomen abgeleitete Sätze:

Assoziativgesetz

$$\begin{aligned}(a \cdot b) \cdot c &= a \cdot (b \cdot c) \\ (a + b) + c &= a + (b + c)\end{aligned}$$

Idempotenzgesetz

$$\begin{aligned}a \cdot a &= a \\ a + a &= a\end{aligned}$$

Absorptionsgesetz

$$\begin{aligned}a + (a \cdot b) &= a \\ a \cdot (a + b) &= a\end{aligned}$$

DeMorgansche Gesetze

$$\begin{aligned}\overline{a + b} &= \overline{a} \cdot \overline{b} \\ \overline{a \cdot b} &= \overline{a} + \overline{b}\end{aligned}$$

Shannon-Gesetze

$$\begin{aligned}\overline{a + b + c + \dots + d} &= \overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \dots \cdot \overline{d} \\ \overline{a \cdot b \cdot c \cdot \dots \cdot d} &= \overline{a} + \overline{b} + \overline{c} + \dots + \overline{d}\end{aligned}$$

Reduktionsgesetze

$$\begin{aligned}a \cdot (\overline{a} + b) &= a \cdot b \\ a + \overline{a} \cdot b &= a + b \\ (a + b) \cdot (\overline{a} + c) &= a \cdot c + \overline{a} \cdot b\end{aligned}$$

Doppelte Negation

$$\overline{\overline{a}} = a$$

Die Vereinbarung „Punktrechnung vor Strichrechnung“ hilft Klammern sparen:

$$a + (b \cdot c) = a + b \cdot c = a + bc$$

Im Folgenden werden die Huntingtonschen Axiome und die Gesetze der Schaltalgebra zum Beweis der Aussagen $a + 1 = 1$ und $a \cdot 0 = 0$ genutzt:

Behauptung:

$$a + 1 = 1$$

Beweis:

$$\begin{aligned}a + 1 &= (a + 1) \cdot 1 && \text{(Neutrales Element)} \\ &= (a + 1) \cdot (a + \overline{a}) && \text{(Komplement)} \\ &= a \cdot a + a \cdot \overline{a} + 1 \cdot a + 1 \cdot \overline{a} && \text{(Distributivität)} \\ &= a + \overline{a} && \text{(Idempotenz/Neutrales Element/Komplement)} \\ &= 1 && \text{(Komplement)}\end{aligned}$$

Der Beweis von $a \cdot 0 = 0$ erfolgt nach dem Dualitätsprinzip in der gleichen Weise.

5.4 Boolesche Funktionen

Eine Funktion

$$f : \underbrace{B \times B \times \dots \times B}_{n-mal} \rightarrow \underbrace{B \times B \times \dots \times B}_{m-mal}$$

heißt *Schaltfunktion*. Kurz: $f : B^n \rightarrow B^m$

Beispiel:

$$f : B^4 \rightarrow B^2 \text{ mit } f(a, b, c, d) = (a + b \cdot \bar{d}, b \cdot \bar{c})$$

Eine Schaltfunktion $f : B^n \rightarrow B$ heißt (n -stellige) *Boolesche Funktion*. Für n unabhängige, binäre Variablen gibt es 2^{2^n} Boolesche Funktionen. Abbildungen 5.4 und 5.5 führen sämtliche einstelligen bzw. zweistelligen Booleschen Funktionen auf.

Name der Funktion	Operationssymbol	Formel	Funktionstabelle		
			a	0	1
Nullfunktion	–	0		0	0
Reproduktion	–	a		0	1
Negation	$\neg a$	\bar{a}		1	0
Einsfunktion	–	1		1	1

Abbildung 5.4: Die 4 einstelligen Booleschen Funktionen

	Name der Funktion	Operationssymbol	Formel	Funktionstabelle
				a 0 0 1 1 b 0 1 0 1
F_0	Nullfunktion	–	0	0 0 0 0 0
F_1	Konjunktion (AND)	–	$a \cdot b$	0 0 0 0 1
F_2	Inhibition	–	$a \cdot \bar{b}$	0 0 1 0 0
F_3	Identität von a	–	a	0 0 1 1 1
F_4	Inhibition	–	$\bar{a} \cdot b$	0 1 0 0 0
F_5	Identität von b	–	b	0 1 0 1 1
F_6	Antivalenz (XOR)	$a \oplus b$	$a \cdot \bar{b} + \bar{a} \cdot b$	0 1 1 0 0
F_7	Disjunktion (OR)	–	$a + b$	0 1 1 1 1
F_8	Peirce-Funktion (NOR)	$a \downarrow b$	$\bar{a} \cdot \bar{b}$	1 0 0 0 0
F_9	Äquivalenz	$a \leftrightarrow b$	$a \cdot b + \bar{a} \cdot \bar{b}$	1 0 0 1 1
F_{10}	Negation von b	$\neg b$	\bar{b}	1 0 1 0 0
F_{11}	Implikation aus b	$b \rightarrow a$	$\bar{b} + a$	1 0 1 1 1
F_{12}	Negation von a	$\neg a$	\bar{a}	1 1 0 0 0
F_{13}	Implikation aus a	$a \rightarrow b$	$\bar{a} + b$	1 1 0 1 1
F_{14}	Sheffer-Funktion(NAND)	$a b$	$\bar{a} + \bar{b}$	1 1 1 1 0
F_{15}	Einsfunktion	–	1	1 1 1 1 1

Abbildung 5.5: Die 16 zweistelligen Booleschen Funktionen

5.4.1 Darstellung von Booleschen Funktionen

Es gibt verschiedene Möglichkeiten, Boolesche Funktionen darzustellen:

Darstellung durch eine Funktionstabelle

Die Darstellung von Booleschen Funktionen mit Hilfe von Funktionstabellen wurde in den Abbildungen 5.4 und 5.5 bereits eingeführt. Funktionstabellen können u.a. zum Beweis von Sätzen verwendet werden. Abbildung 5.6 zeigt den Beweis für die Regel von DeMorgan ($a + b = \bar{a} \cdot \bar{b}$) anhand einer solchen Tabelle.

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

Abbildung 5.6: Beweis der deMorgan-Regel mit einer Funktionstabelle

Man beachte außerdem, dass jede Boolesche Funktion unter alleiniger Verwendung des Sheffer- oder des Peirce-Operators dargestellt werden kann (Abb. 5.7). Für die UND-Verknüpfung ist dies in Abbildung 5.8 exemplarisch bewiesen.

	NOR	NAND
$a \cdot b$	$(a \downarrow a) \downarrow (b \downarrow b)$	$(a b) (a b)$
$a + b$	$(a \downarrow b) \downarrow (a \downarrow b)$	$(a a) (b b)$
\bar{a}	$a \downarrow a$	$a a$

Abbildung 5.7: Ersatz von OR, AND und \neg durch den Sheffer- bzw. Peirce-Operator

a	b	$a \cdot b$	$a \downarrow a$	$b \downarrow b$	$(a \downarrow a) \downarrow (b \downarrow b)$
			$\overline{a + a}$	$\overline{b + b}$	
0	0	0	1	1	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	1	0	0	1

Abbildung 5.8: Beweis, dass die UND- durch die Peirce-Funktion darstellbar ist.

Darstellung mit Schalterdiagrammen und Schaltnetzen

Boolesche Funktionen lassen sich aufgrund der Zweiwertigkeit ihrer Eingangsvariablen und der Funktionswerte besonders einfach durch Schaltungen technisch realisieren. Die Wahrheitswerte 1 und 0 der Schaltalgebra können durch „Spannung/keine Spannung“, „Strom/kein Strom“,

„Kontakt/kein Kontakt“ etc. technisch dargestellt werden. So besteht z. B. die Möglichkeit, Eingabevervariablen einer Booleschen Funktion als Schalter zu realisieren. Beträgt der Wert einer Eingabevervariablen 1, so sind die Kontakte des zugeordneten Schalters geschlossen, sonst geöffnet. Für negierte Eingabevervariablen ergibt sich natürlich die umgekehrte Situation. Boolesche Funktionen können also als so genannte Schalterdiagramme dargestellt werden. Aus den Beispielen in Abbildung 5.9 wird ersichtlich, wie Schalterdiagramme aufgebaut sind. Anhand von Schalterdiagrammen lassen sich Theoreme der Schaltalgebra sehr anschaulich beweisen. Abbildung 5.10 zeigt den Beweis eines der Reduktionsgesetze mit Hilfe von Schalterdiagrammen.

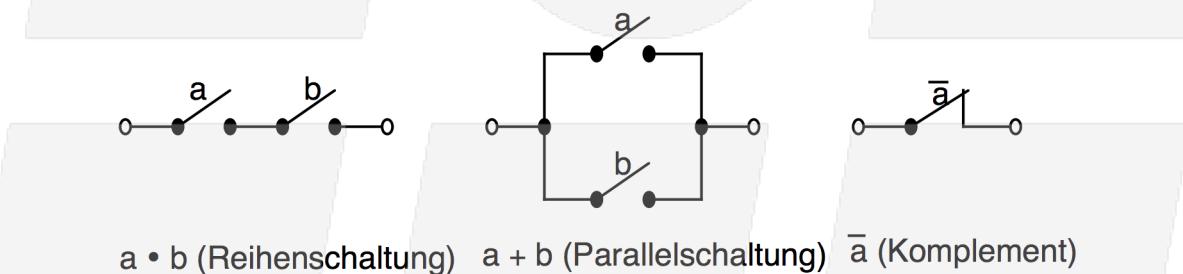


Abbildung 5.9: Darstellung von Booleschen Funktionen durch Schalterdiagramme

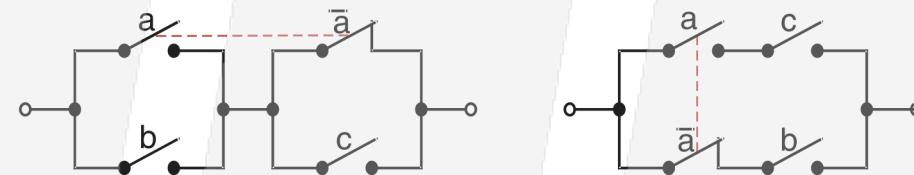


Abbildung 5.10: Beweis des Reduktionsgesetzes $(a + b) \cdot (\bar{a} + c) = a \cdot c + \bar{a} \cdot b$

Darstellung durch Venndiagramme und Programme

Dass eine Boolesche Funktion mit Hilfe eines Venndiagramms dargestellt werden kann, wurde bereits in Abschnitt 5.3.1 erläutert.

Des Weiteren kann eine Boolesche Funktion auch durch die Angabe eines Programms zur Berechnung ihrer Funktionswerte bestimmt werden. Diese Darstellungsmöglichkeit wird hier jedoch nicht weiter betrachtet und sei nur der Vollständigkeit halber erwähnt.

Darstellung durch eine Formel

Für Boolesche Funktionen mit vielen Eingangsvariablen ist eine Darstellung mittels Funktions-tabelle nicht mehr praktikabel. Stattdessen kann die Boolesche Funktion durch eine Formel, in der Regel unter Verwendung der Operatoren $+$, \cdot und \neg , beschrieben werden. Die Formeldarstellung einer Booleschen Funktion ist nicht eindeutig. Es existieren zwei kanonische Formen der Darstellung Boolescher Funktionen, die disjunktive Normalform und die konjunktive Normalform:

Disjunktive Normalform

Eine Boolesche Funktion f ist in disjunktiver Form (*DF*), wenn sie ausschließlich aus Konjunktionen $K_i, i = 1 \dots n$, besteht, die ihrerseits disjunktiv verknüpft sind:

$$f = K_1 + K_2 + \dots + K_n$$

Eine Konjunktion ist dabei ein Term, in dem einfache oder negierte Variablen ausschließlich konjunktiv verknüpft sind. Eine Konjunktion heißt *Minterm* von f , wenn sie alle Eingangsvariablen von f enthält.¹

f ist in disjunktiver Normalform (*DNF*), wenn sie ausschließlich aus disjunktiv verknüpften Mintermen besteht.

Beispiele:

$f(a, b, c) = a \cdot (ab + bc)$ ist weder in DF noch in DNF.

$f(a, b, c) = abc + a \cdot (\bar{b} + c)$ ist weder in DF noch in DNF.

$f(a, b, c) = a\bar{b}c + a\bar{b}\bar{c}$ ist in DNF.

$f(a, b, c) = a\bar{b}c + a\bar{c}$ ist in DF, nicht jedoch in DNF.

Konjunktive Normalform

Eine Boolesche Funktion f ist in konjunktiver Form (*KF*), wenn sie ausschließlich aus Disjunktionen $D_i, i = 1 \dots n$, besteht, die ihrerseits konjunktiv verknüpft sind:

$$f = D_1 \cdot D_2 \cdot \dots \cdot D_n$$

Eine Disjunktion ist dabei ein Term, in dem einfache oder negierte Variablen ausschließlich disjunktiv verknüpft sind. Eine Disjunktion heißt *Maxterm* von f , wenn sie alle Eingangsvariablen von f enthält.² f ist in konjunktiver Normalform (*KNF*), wenn sie ausschließlich aus konjunktiv verknüpften Maxtermsen besteht.

Beispiele:

$f(a, b, c) = a + bc$ ist weder in KF noch in KNF.

$f(a, b, c) = (a + \bar{b} + c) \cdot (a + \bar{b} + \bar{c})$ ist in KNF.

$f(a, b, c) = (a + \bar{b} + c) \cdot (a + \bar{c})$ ist in KF, nicht jedoch in KNF.

Jede Boolesche Funktion lässt sich durch die Anwendung der Huntingtonschen Axiome und der daraus ableitbaren Sätze in ihre (eindeutige) disjunktive bzw. konjunktive Normalform überführen.

Diese lassen sich direkt aus der Funktionstabelle ablesen.

¹Der Begriff *Minterm* kommt daher, dass ein Minterm die *minimal* mögliche Anzahl von Einsen in der Wahrheitstabelle erzeugt (nämlich in genau einer Zeile).

²Der Begriff *Maxterm* kommt daher, dass ein Maxterm die *maximal* mögliche Anzahl von Einsen in der Wahrheitstabelle erzeugt (nämlich in allen außer einer Zeile).

Beispiel:

a	b	c	Q
0	0	0	0
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	0

Die DNF lautet:

$$Q = \bar{a} \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot c + a \cdot \bar{b} \cdot \bar{c} + a \cdot \bar{b} \cdot c + a \cdot b \cdot \bar{c}$$

Damit erhält man die DNF wie folgt:

Ist der Wert des Ausgangs Q aufgrund der Wertekombination an den Eingängen eine 0, überinge man die Tabellenzeile.

Ist der Wert des Ausgangs Q indessen eine 1, ist ein Minterm zu bilden. Dabei entsprechen Nullen Negierungen und Einsen Identitäten der Eingangsvariablen.

Die KNF lautet:

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

Damit erhält man die KNF wie folgt:

Ist der Wert des Ausgangs Q aufgrund der Wertekombination an den Eingängen eine 1, überinge man die Tabellenzeile.

Ist der Wert des Ausgangs Q indessen eine 0, ist ein Maxterm zu bilden. Dabei entsprechen Einsen Negierungen und Nullen Identitäten der Eingangsvariablen.

Darstellung durch ein Karnaugh-Veitch-Diagramm

Diese grafische Darstellungsform beruht auf der Eintragung der Funktionswerte einer Booleschen Funktion in Quadrate. Die Darstellung mit Hilfe von Karnaugh-Veitch-Diagrammen (**KV-Diagrammen**) spielt bei der Minimierung von Booleschen Funktionen eine wichtige Rolle (siehe Abschnitt 5.4.2).

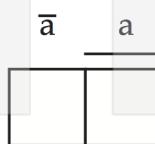


Abbildung 5.11: KV-Diagramm für einstellige Funktionen

Das einfachste KV-Diagramm ist ein solches für einstellige Funktionen. Abbildung 5.11 zeigt, dass es aus zwei aneinandergesetzten Quadranten besteht, deren Flächen den Funktionswerten der Variablen a bzw. deren Komplement $\neg a$ zugeordnet sind.

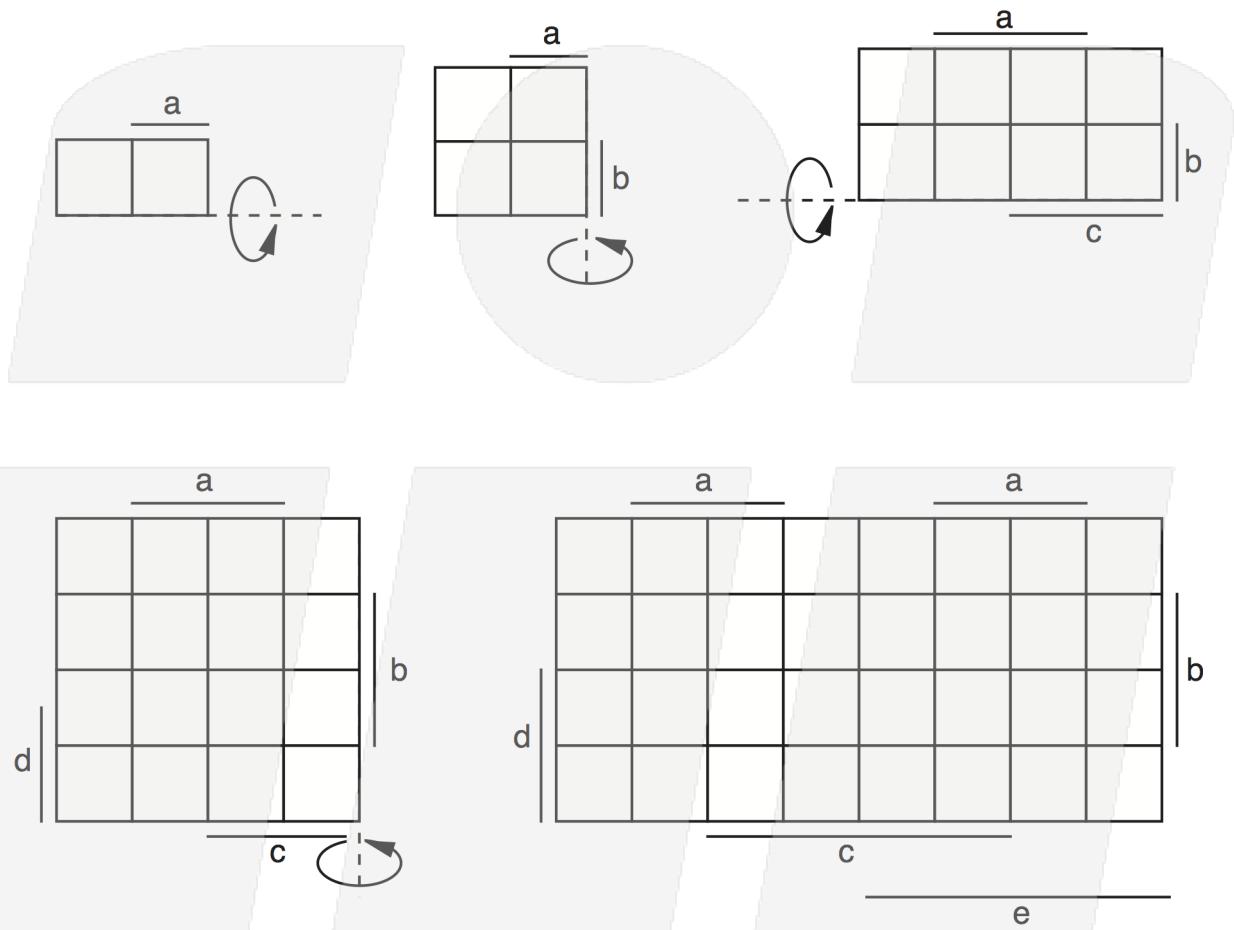


Abbildung 5.12: KV-Diagramme für ein- bis fünfstellige Funktionen

Das Quadrat für die nicht-negierte Variable wird durch einen Balken markiert. Ausgehend von diesem Diagramm können Diagramme für Funktionen mit mehr als einer Variablen erstellt werden. Ein KV-Diagramm für $n + 1$ Variablen entsteht, indem das KV-Diagramm für n Variablen an der unteren Kante gespiegelt wird, falls n ungerade ist, bzw. an der rechten seitlichen Kante gespiegelt wird, falls n gerade ist. Abbildung 5.12 zeigt die Konstruktion von KV-Diagrammen mit bis zu 5 Variablen.

Genau genommen handelt es sich bei der Spiegelung um eine Konstruktionsvorschrift nach Veitch, während das Verfahren von Karnaugh auf einem einfachen Aneinandersetzen der Felder beruht. Die Spiegelungsmethode von Veitch hat den Vorteil, dass in Diagrammen mit bis zu 4 Variablen die Bereiche der positiven bzw. negierten Variablen in übersichtlicher Form zusammenhängen, so dass im Folgenden lediglich diese Methode zur Anwendung kommt. Es wird jedoch weiterhin allgemein von KV-Diagrammen gesprochen.

Eine konkrete Boolesche Funktion kann nun in einem KV-Diagramm dargestellt werden, indem die Funktionswerte dieser Funktion in die Quadrate der entsprechenden Variablenkombination eingetragen werden. Abbildung 5.13 zeigt ein KV-Diagramm für vierstellige Funktionen, in dessen Quadrate die Dezimal-Äquivalente eingetragen sind, die der UND-Verknüpfung der Variablen im Binärkode für die Anordnung (d, c, b, a) entspricht.

Man beachte, dass jedes Quadrat im KV-Diagramm genau einem Minterm, also einer Konjunktion von allen Eingabeveriablen (negiert oder nicht-negiert) entspricht, so dass sich die

				a
0	1	5	4	
2	3	7	6	b
10	11	15	14	
8	9	13	12	c
d				

Abbildung 5.13: Dezimal-Äquivalente

DNF direkt aus einem KV-Diagramm angeben lässt. In der Abbildung 5.14 sind die Booleschen Funktionen $f_1(a) = \bar{a}$, $f_2(a, b) = a + b$ und $f_3(a, b, c, d) = ad + b + \bar{a}\bar{c}\bar{d}$ dargestellt. Es genügt, lediglich die Einsen einzutragen, wenn leere Quadrate als Variablenkombination mit dem Funktionswert 0 interpretiert werden. Hier ist eine Schreibweise gewählt, in der mit 1 zu belegende Felder schraffiert sind.

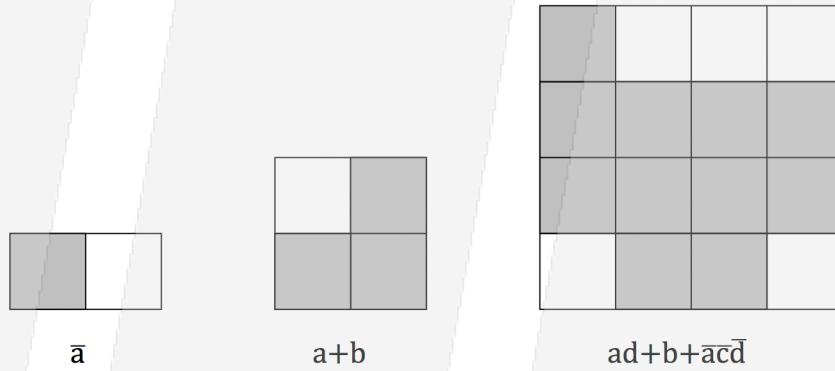


Abbildung 5.14: Beispiele für Boolesche Funktionen im KV-Diagramm

5.4.2 Minimierung Boolescher Funktionen

Jede Boolesche Funktion kann in eindeutiger Weise in der DNF (oder auch KNF) dargestellt werden. Die meisten Funktionen können jedoch auch durch eine Formel beschrieben werden, die kürzer ist als ihre DNF. Es ist offensichtlich, dass der Aufwand für die schalttechnische Realisierung einer Booleschen Funktion abnimmt, wenn die zugrundegelegte Formel kürzer ist. Setzt man eine Realisierung durch ein Schaltnetz mit AND- und OR-Gattern voraus, so führt der Wegfall von Variablen in einem Minterm zu AND-Gattern mit weniger Eingängen. Kann sogar ein ganzer Minterm wegfallen, ohne dass sich die Boolesche Funktion dadurch ändert, wird ein komplettes AND-Gatter eingespart, und das OR-Gatter kommt mit einem Eingang weniger aus. Wird die Boolesche Funktion ausschließlich mit Hilfe von zweistelligen Gattern realisiert, so führt jeder Wegfall einer Variablen auch zum Wegfall mindestens eines Gatters. Im Rahmen der

Vorlesung werden zwei Verfahren zur systematischen Verkürzung der DNF einer Booleschen Funktion vorgestellt, ein auf den KV-Diagrammen basierendes graphisches Verfahren und ein algebraisches Verfahren von Quine und McCluskey.

Beide Verfahren beruhen auf dem Prinzip, dass zwei Konjunktionen, die bis auf die Negation genau einer Variablen identisch sind, zu einer einzigen Konjunktion zusammengefasst werden können, in der diese Variable weggelassen wird. Genauer:

$$\begin{aligned} abc + ab\bar{c} &= ab \cdot (c + \bar{c}) && (\text{Distributivgesetz}) \\ &= ab \cdot 1 && (\text{Komplement}) \\ &= ab && (\text{Neutrales Element}) \end{aligned}$$

In dieser Umformung wurden das Kommutativ- und das Assoziativgesetz implizit verwendet.

Minimierung mit KV-Diagrammen

Durch die geschickte graphische Anordnung der Minterme in KV-Diagrammen, genauer in Veitch-Diagrammen für bis zu 4 Variablen, sind diejenigen Konjunktionen benachbart, auf die sich das o. a. Prinzip der Vereinfachung anwenden lässt. Um die Darstellung einer Booleschen Funktion zu vereinfachen, fasst man deshalb zwei benachbarte Quadrate bzw. Minterme zu einem sog. Zweierblock zusammen. Abbildung 5.15 zeigt einen solchen Vereinfachungsschritt an einem Beispiel.

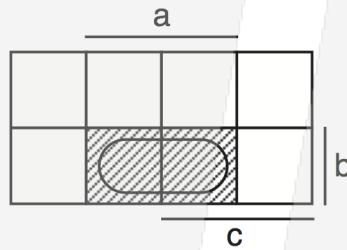


Abbildung 5.15: Vereinfachung der Funktion $f(a, b, c) = abc + ab\bar{c}$ zu $f(a, b, c) = ab$

Dieses Verfahren kann fortgesetzt werden, indem man zwei Zweierblöcke zu einem Viererblock, zwei Viererblöcke zu einem Achterblock etc. vereinfacht. In der Abbildung 5.16 wird ein Viererblock aus zwei Zweierblöcken gebildet.

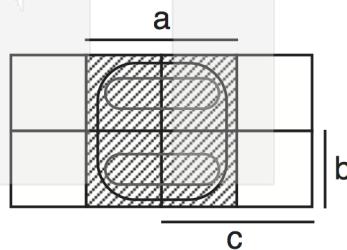


Abbildung 5.16: Vereinfachung der Funktion $f(a, b, c) = ab + a\bar{b}$ zu $f(a, b, c) = a$

Statt den Umweg über die zwei jeweils nächst kleineren Blöcke zu gehen, können Einserblöcke bzw. Minterme direkt zu möglichst großen Blöcken zusammengefasst werden. Im Beispiel bedeutet dies, dass der Viererblock direkt aus den vier Einserblöcken gebildet werden kann.

Die Minimierung von Booleschen Funktionen mit Hilfe von KV-Diagrammen geschieht also in der Weise, dass Minterme bzw. Einserblöcke zu möglichst großen Blöcken zusammengefasst werden. Abbildung 5.17 zeigt mehrere Beispiele, aus denen noch einige Details sichtbar werden:

- Die Nachbarschaft von Blöcken setzt sich über die Ränder des Diagramms zyklisch fort (wrap around, Bsp. 1, 2 und 3).
- Kleinere Blöcke können (aufgrund des Idempotenzgesetzes) mehrfach zur Bildung von größeren Blöcken benutzt werden (Bsp. 4).
- Es gibt i. A. verschiedene, gleichwertige Möglichkeiten, Einserblöcke zu größeren Blöcken zusammenzufassen. Die Vereinfachung einer Booleschen Funktion ist deshalb nicht eindeutig (Bsp. 5 und 6).



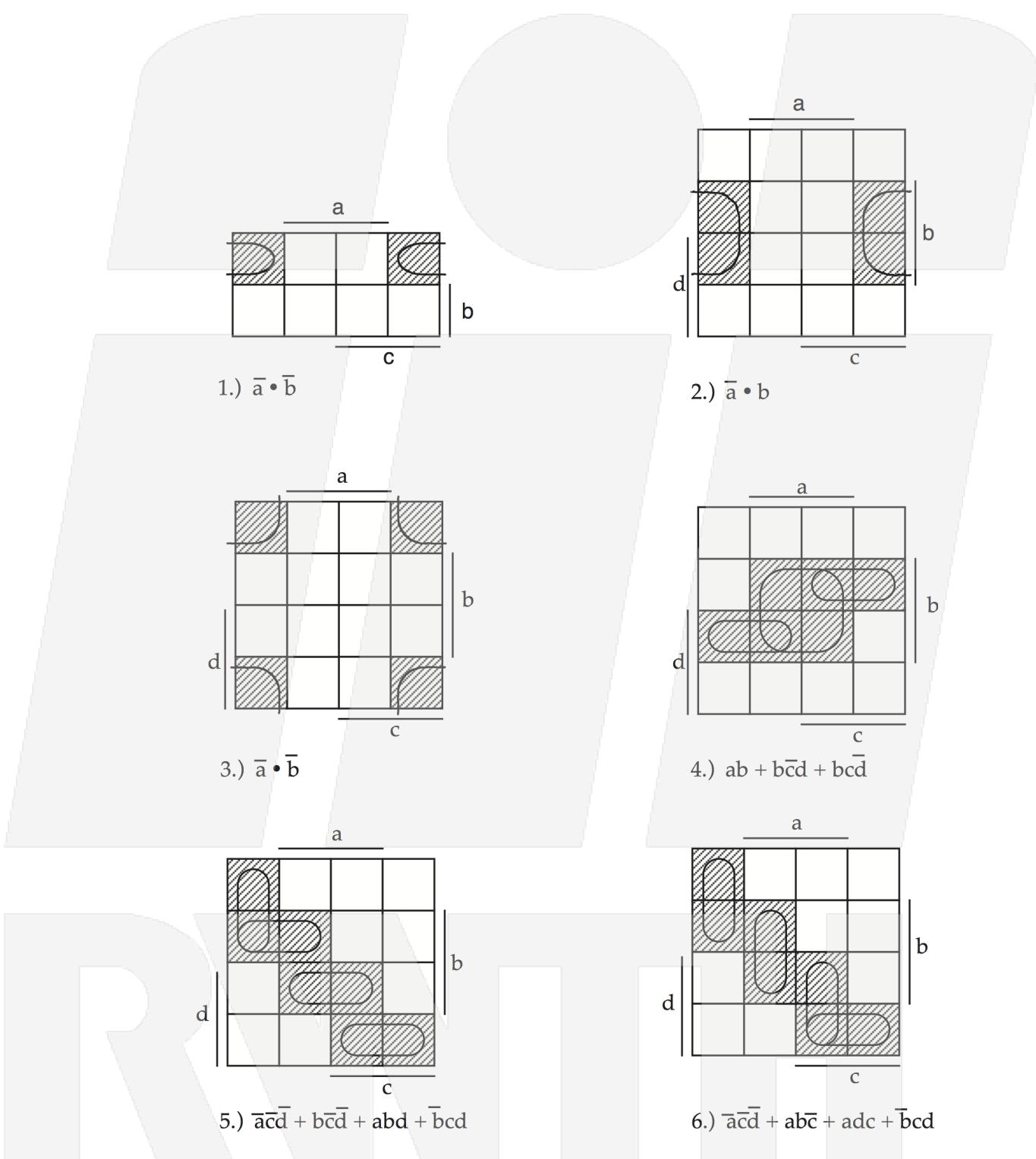


Abbildung 5.17: Beispiele für die Minimierung mit KV-Diagrammen

5.4.3 Behandlung unvollständig definierter Schaltfunktionen

Es gibt Eingangskombinationen x_1, x_2, \dots, x_n denen kein Funktionswert $f(x_1, x_2, \dots, x_n)$ zugeordnet ist. Falls es gleichgültig ist, welche Ausgangssignale die Booleschen Funktionen bei diesen Eingangskombinationen erzeugt, können die mit * gekennzeichneten „don't care“-Felder zur Schaltungsvereinfachung herangezogen werden.

Beispiel: Decodierung der Dezimalzahlen 1, 2, 3, 4, 5 aus einem 4-stelligen Dualcode. Abbildung 5.18 zeigt das dazugehörige KV-Diagramm und die Funktionsgleichung ohne Don't Care-Felder.

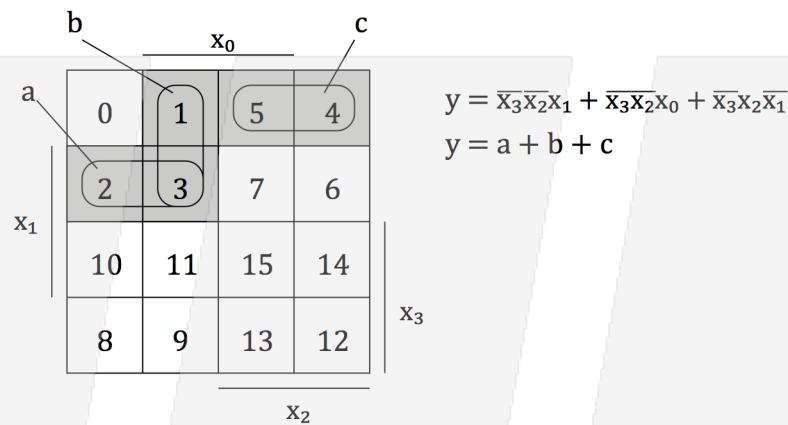


Abbildung 5.18: Dezimalzifferndecodierung ohne d.c.-Felder

Abbildung 5.19 zeigt das KV-Diagramm bei Berücksichtigung der Don't Care-Fälle für die Ziffern 12, 13, 14, 15.

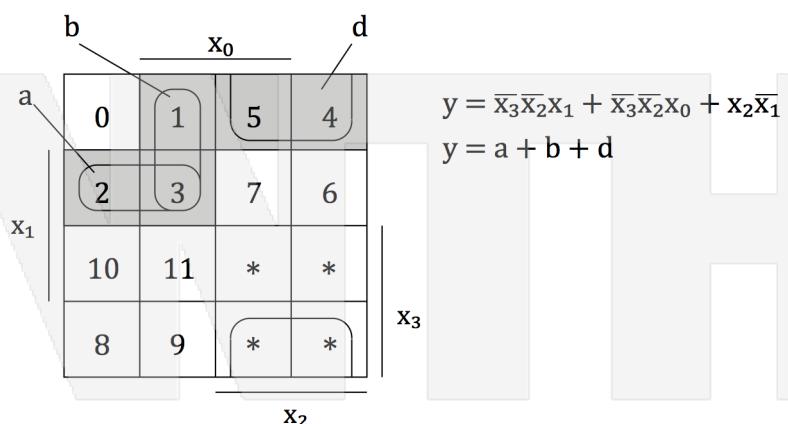


Abbildung 5.19: Dezimalzifferndecodierung mit d.c.-Feldern

Treten die don't care - Eingangskombinationen auf, so liefern 12, 13 eine „1“ und 14, 15 eine „0“.

Das Quine-McCluskey-Verfahren

Im Quine-McCluskey-Verfahren werden die Minterme der DNF in einer Tabelle in Klassen K_i geordnet. Der Index i bezeichnet die Anzahl der nicht-negierten Variablen. Man versucht nun, die Minterme benachbarter Klassen K_i zusammenzufassen. Minterme, die derart zusammengefasst werden können, werden abgehakt. Mit den zusammengefassten Termen verfährt man dann genauso wie mit den Mintermen selbst, d. h. sie werden in Klassen eingeteilt und, falls möglich, zusammengefasst und abgehakt. Das Verfahren wird solange fortgesetzt, bis es keine zusammenfassbaren Terme mehr gibt. Die nicht abgehakten Terme, die so genannten *Primimplikanten*, entsprechen dann denjenigen Blöcken im Karnaugh-Verfahren, welche nicht mehr vergrößert werden können. Abbildung 5.20 zeigt die Anwendung des Verfahrens anhand der Beispielfunktion

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

Um nachhalten zu können, welche Terme miteinander verschmolzen wurden, sind in Klammern die entsprechenden Dezimaläquivalente (vgl. Abb. 5.13) angegeben. Die resultierende Darstellung der Funktion f (vgl. Abb. 5.21) lautet somit

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

Ein Blick auf das dieser Formel entsprechende KV-Diagramm (Abb. 5.21) lässt sofort erkennen, dass einige Blöcke überflüssig sind, da alle ihre Einsen bereits durch andere Blöcke überdeckt werden.

Für die Formel bedeutet dies, dass sie durch Weglassen von Konjunktionen, die diesen überflüssigen Blöcken entsprechen, weiter verkürzt werden kann. Während für bis zu vier Variablen die Bestimmung einer minimalen Menge von Konjunktionen bzw. Blöcken, die alle Minterme bzw. Einsen erfasst (überdeckt), mit Hilfe eines KV-Diagramms sehr einfach zu ermitteln ist, muss für mehr als vier Variablen ein anderes Verfahren zur Lösung des so genannten Überdeckungsproblems gewählt werden:

- Zunächst stellt man eine Tabelle bzw. Matrix auf, in der jedem Minterm eine Spalte und jedem Primimplikanten eine Zeile zugeordnet ist. Jedes Feld der Tabelle, das durch einen Primimplikanten und einen darin enthaltenen Minterm definiert ist, wird markiert (Abb. 5.22).
- Ausgehend von dieser Tabelle bestimmt man die so genannten *wesentlichen Primimplikanten*. Ein wesentlicher Primimplikant überdeckt einen Minterm, der sonst von keinem anderen Primimplikanten überdeckt wird. In der Tabelle verweisen Spalten, die nur eine Markierung enthalten, auf wesentliche Primimplikanten. Im Beispiel stellt sich u.a. der Primimplikant $\overline{c}d$ als wesentlich heraus, da er als einziger den Minterm $a\overline{b}cd$ (Dezimaläquivalent 11) überdeckt.
- Die in den wesentlichen Primimplikanten nicht enthaltenen Minterme und die nicht-wesentlichen Primimplikanten fasst man in einer Restmatrix zusammen (Abb. 5.23).
- In dieser *Restmatrix* muss nun eine Auswahl von Zeilen getroffen werden, so dass die daraus resultierende Formel eine minimale Länge besitzt. Dabei ist darauf zu achten, dass in dieser Auswahl alle restlichen Minterme enthalten sein müssen. Es sei darauf hingewiesen, dass dieser Schritt u.U. hohen Aufwand mit sich bringt, da im schlimmsten Fall

sämtliche Teilmengen der verbleibenden Primimplikanten untersucht werden müssen. Im Beispiel gestaltet sich die Auswahl sehr einfach. Es ergeben sich vier gleichwertige Restüberdeckungen.

Klasse	Minterme mit Dezimal-Äquivalenten	Terme mit 3 Variablen	Terme mit 2 Variablen	
K_0	$\bar{d}\bar{c}\bar{b}\bar{a}$ (0) ✓	$\bar{d}\bar{c}\bar{b}$ (0 - 1) $\bar{d}\bar{b}\bar{a}$ (0 - 4) $\bar{c}\bar{b}\bar{a}$ (0 - 8)	✓ ✓ ✓	$\bar{d}\bar{b}$ (0,1 - 4,5) (0,4 - 1,5) $\bar{c}\bar{b}$ (0,1 - 8,9) (0,8 - 1,9)
K_1	$\bar{d}\bar{c}\bar{b}a$ (1) ✓ $\bar{d}c\bar{b}\bar{a}$ (4) ✓ $d\bar{c}\bar{b}\bar{a}$ (8) ✓	$\bar{d}\bar{b}a$ (1 - 5) $\bar{c}ba$ (1 - 9) $\bar{d}cb$ (4 - 5) $\bar{d}ca$ (4 - 6) $\bar{d}c\bar{a}$ (8 - 10) $d\bar{c}\bar{b}$ (8 - 9)	✓ ✓ ✓ ✓ ✓ ✓	$\bar{d}c$ (4,5 - 6,7) (4,6 - 5,7) $d\bar{c}$ (8,10 - 9,11) (8,9 - 10,11)
K_2	$\bar{d}cba$ (5) ✓ $\bar{d}cb\bar{a}$ (6) ✓ $d\bar{c}b\bar{a}$ (10) ✓ $d\bar{c}\bar{b}a$ (9) ✓	$\bar{d}ca$ (5 - 7) $\bar{d}cb$ (6 - 7) $cb\bar{a}$ (6 - 14) $d\bar{c}b$ (10 - 11) $db\bar{a}$ (10 - 14) $d\bar{c}a$ (9 - 11)	✓ ✓ ○ ✓ ○ ✓	
K_3	$\bar{d}cba$ (7) ✓ $d\bar{c}ba$ (11) ✓ $dc\bar{b}\bar{a}$ (14) ✓			
K_4				

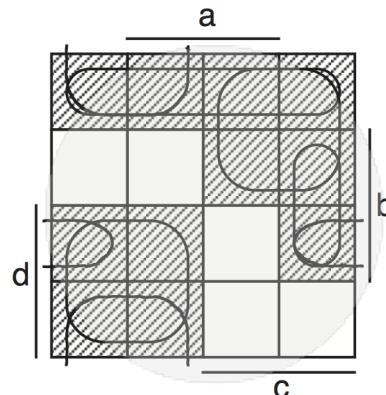
Abbildung 5.20: Das Quine-McCluskey-Verfahren in einem Beispiel
(...) – Dezimaläquivalente der jeweiligen Terme

Die resultierende Darstellung der Funktion f setzt sich somit zusammen aus den wesentlichen Primimplikanten und einer der vier minimalen Restüberdeckungen:

$$f(a, b, c, d) = cd + \bar{c}d + \begin{cases} \bar{a}bc + \bar{b}\bar{d} \\ \bar{a}bc + \bar{b}\bar{c} \\ \bar{a}bd + \bar{b}\bar{d} \\ \bar{a}bd + \bar{b}\bar{c} \end{cases}$$

Berücksichtigung von Don't-Care-Fällen

Bei der Bestimmung der Primimplikanten werden Don't-Care-Fälle wie Fälle gewertet, bei denen der Funktionswert gleich Eins ist. Bei der Bestimmung der wesentlichen Primimplikan-


 Abbildung 5.21: Das KV-Diagramm der verkürzten Darstellung von f

Primimplikanten	Minterme										
	0	1	4	5	6	7	8	9	10	11	14
$\bar{a}bc$					x						x
$\bar{a}bd$									x		x
$\bar{b}\bar{d}$	x	x	x	x							
$\bar{b}\bar{c}$	x	x					x	x			
cd (wesentlich)			x	x	x	\otimes					
$\bar{c}d$ (wesentlich)							x	x	x		\otimes

 Abbildung 5.22: Matrix zur Auswahl der wesentlichen Primimplikanten - mit \otimes markiert

ten jedoch werden die Primimplikanten ausgeschlossen, die nur Don't-Care-Fälle überdecken. Auch in der Restmatrix brauchen solche Fälle nicht berücksichtigt zu werden.

Angenommen, die Funktion f aus dem letzten Abschnitt besitze beim Minterm $dcb\bar{a}$ einen Don't-Care-Fall, also

$$f(dcb\bar{a}) = \text{Don't Care}$$

Dann verläuft die Bestimmung der Primimplikanten genau wie in Abbildung 5.20 gezeigt. Auch die Bestimmung der wesentlichen Primimplikanten bleibt wie in Abbildung 5.22. Bei der Restmatrix sieht man jedoch, dass die Terme $\bar{a}bc$ und $\bar{a}bd$ nur noch den Minterm 14, also genau den Don't-Care-Fall, erzeugen müssen. Damit sind diese Terme nicht notwendig, und die minimier-

Primimplikanten	Minterme		
	0	1	14
$\bar{a}bc$		x	
$\bar{a}bd$			x
$\bar{b}\bar{d}$	x	x	
$\bar{b}\bar{c}$	x	x	

Abbildung 5.23: Restmatrix

te Funktion ist

$$f(a, b, c, d) = c\bar{d} + \bar{c}d + \begin{cases} \bar{b}\bar{d} \\ \bar{b}\bar{c} \end{cases}$$

Kapitel 6

Logische Schaltungen

6.1 Technische Realisierung Logischer Funktionen

In logischen Schaltungen werden einfache Boolesche Funktionen wie z.B. OR und AND durch Verknüpfungsglieder, so genannte *Gatter*, technisch realisiert. In Abbildung 6.1 sind die wichtigsten Gattersymbole aufgeführt. Durch die Verknüpfung mehrerer solcher Gatter lassen sich komplexere Boolesche Funktionen als sogenannte *Schaltnetze* darstellen.

6.2 Standard-Schaltnetze

Unter einem *Schaltnetz* versteht man eine Gatterschaltung (Abb. 6.2 rechts), bei der die Ausgangssignale nur von den Eingangssignalen abhängen (keine Rückführungen). Es besteht eine Zuordnung von n Eingangsvariablen zu m Ausgangsvariablen.

6.2.1 Addierer

Wie in Kapitel 4 gezeigt wurde, können die arithmetischen Grundoperationen mit Hilfe der Komplementbildung auf die Addition zurückgeführt werden. Ein Addierer stellt somit eine wesentliche Komponente des Rechenwerks eines Digitalrechners dar.

Halbaddierer

In der Funktionstabelle 6.1 für die Addition 1-stelliger Dualzahlen bezeichnet s die Funktion der Summe und \ddot{u} die Funktion des Übertrags in die nächste Bitstelle.

Bildet man die DNF für die Summe und den Übertrag, so ergeben sich die folgenden Gleichungen:

$$s = \overline{x_1}x_0 + x_1\overline{x_0} \quad (6.1)$$

$$\ddot{u} = x_1x_0 \quad (6.2)$$

Mit Hilfe der Gleichungen 6.1 und 6.2 bildet man das in Abbildung 6.2 wiedergegebene Schaltnetz.

Formel	Schaltsymbole nach DIN 40700 Teil 14	
	bis 1976	seit 1976
$a + b + c + \dots$		
$a \cdot b \cdot c \cdot \dots$		
\bar{a}		
$\overline{a \cdot b \cdot c \cdot \dots}$		
$\overline{a + b + c + \dots}$		
$a \oplus b \oplus c \oplus \dots$		
$\bar{a} \cdot b$		

Abbildung 6.1: Gattersymbole für logische Verknüpfungen.

Tabelle 6.1: Halbaddierer

x_1	x_0	s	\ddot{u}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Gleichung 6.1 entspricht dem eXklusiven OdeR:

$$s = x_1 \oplus x_0 \quad (6.3)$$

Mit Hilfe der Gleichungen 6.2 und 6.3 entwickelt man das in Abbildung 6.3 gezeigte gleichwertige Schaltnetz.

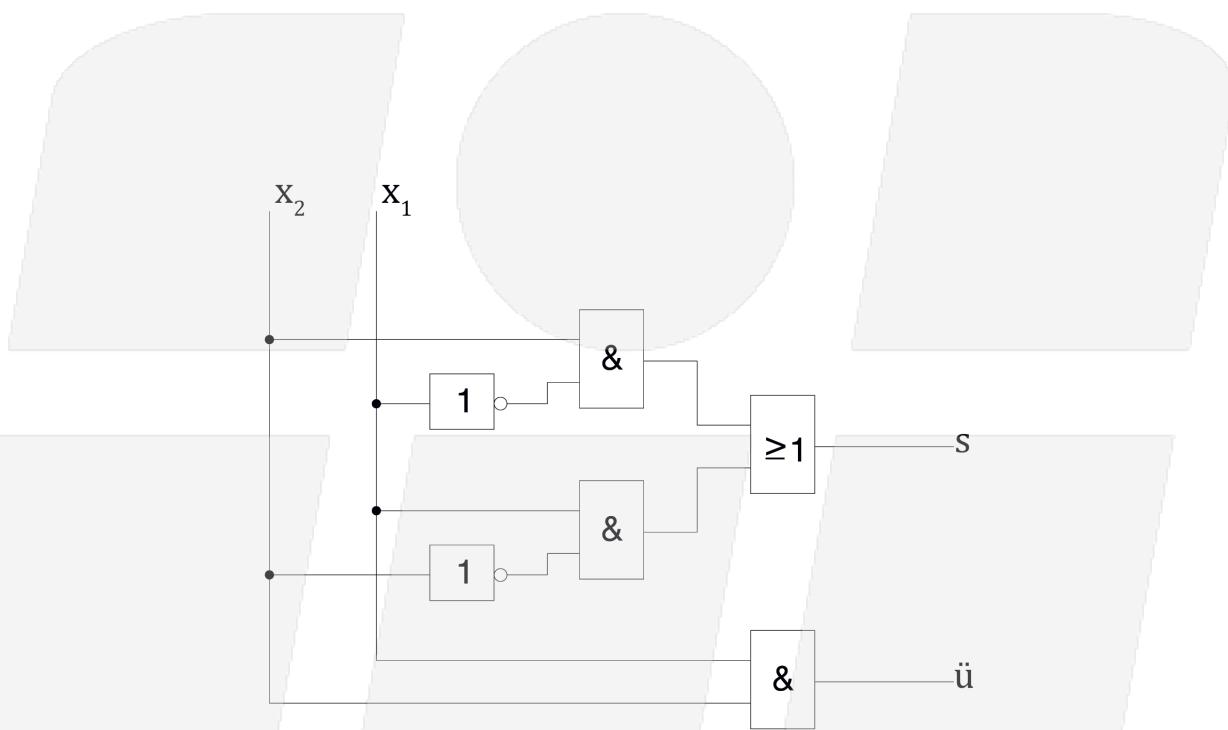


Abbildung 6.2: Schaltnetz eines Halbaddierers

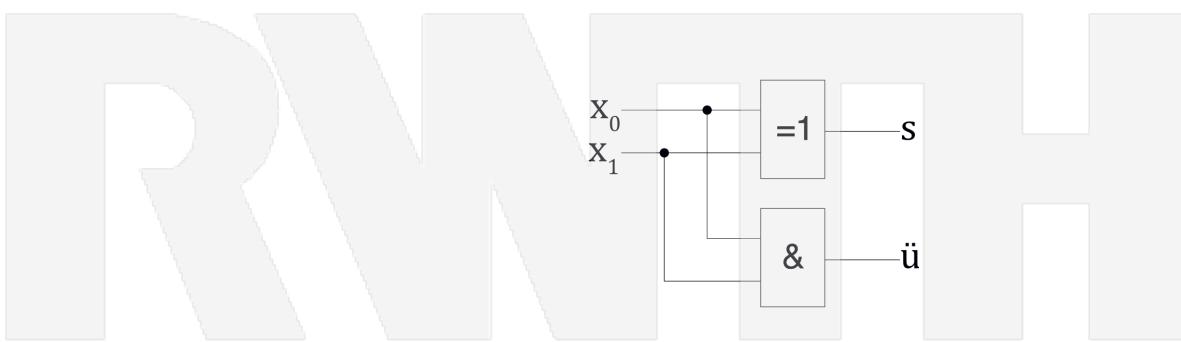


Abbildung 6.3: Aufbau eines Halbaddierers aus einem XOR-Gatter und einem UND-Gatter

Soll die Verwendung des XOR-Gatters vermieden werden, so kann Gleichung 6.1 wie folgt vereinfacht werden:

$$\begin{aligned}
 s &= \overline{x_1}x_0 + x_1\overline{x_0} \\
 &= (\overline{x_1}x_0 + x_1)(\overline{x_1}x_0 + \overline{x_0}) \\
 &= [(\overline{x_1} + x_1)(x_0 + x_1)][(\overline{x_1} + \overline{x_0})(x_0 + \overline{x_0})] \\
 &= (x_0 + x_1)(\overline{x_1} + \overline{x_0}) \\
 &= (x_0 + x_1)\overline{x_1 \cdot x_0} \\
 &= (x_0 + x_1)\bar{u}
 \end{aligned} \tag{6.4}$$

Abbildung 6.4 gibt das zugehörige Schaltnetz und das wieder.

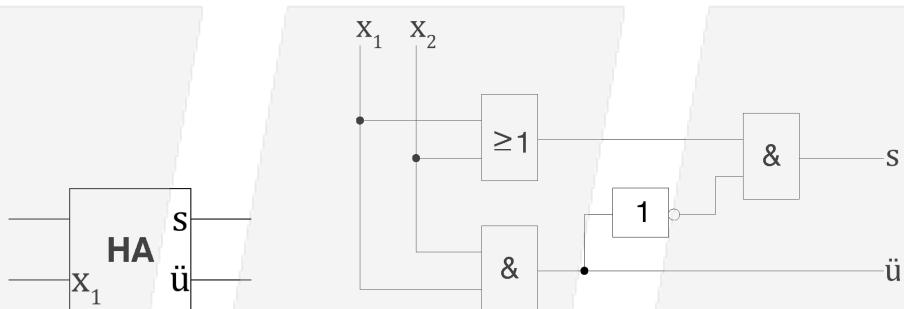


Abbildung 6.4: Schaltzeichen und vereinfachtes Schaltnetz eines Halbaddierers

Volladdierer

Zur Addition mehrstelligiger Dualzahlen muss der Übertrag vorangehender Bitstellen (in der Funktionstabelle 6.2 als \bar{u}' bezeichnet) berücksichtigt werden.

Tabelle 6.2: Volladdierer

x_1	x_0	\bar{u}'	s	\bar{u}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Bildet man die DNF für die Summe und den Übertrag, so ergeben sich die folgenden Gleichungen:

$$s = \overline{x_1}\overline{x_0}\bar{u}' + \overline{x_1}x_0\overline{\bar{u}'} + x_1\overline{x_0}\overline{\bar{u}'} + x_1x_0\bar{u}' \tag{6.5}$$

$$\bar{u} = \overline{x_1}x_0\bar{u}' + x_1\overline{x_0}\bar{u}' + x_1x_0\overline{\bar{u}'} + x_1x_0\bar{u}' \tag{6.6}$$

Algorithmische Vereinfachung der DNF:

$$\begin{aligned}
 s &= \overline{x_1} \overline{x_0} \ddot{u}' + \overline{x_1} x_0 \ddot{u}' + x_1 \overline{x_0} \ddot{u}' + x_1 x_0 \ddot{u}' \\
 &= x_1 x_0 \ddot{u}' + \overline{x_1} \overline{x_0} \ddot{u}' + x_1 \overline{x_0} \ddot{u}' + \overline{x_1} x_0 \ddot{u}' \\
 &= (x_1 x_0 + \overline{x_1} \overline{x_0}) \ddot{u}' + (x_1 \overline{x_0} + \overline{x_1} x_0) \ddot{u}' \\
 &= (x_1 \leftrightarrow x_0) \ddot{u}' + (x_1 \oplus x_0) \ddot{u}' \\
 &\quad \text{Äquivalenz Antivalenz}
 \end{aligned} \tag{6.7}$$

mit $(x_1 \leftrightarrow x_0) = \overline{(x_1 \oplus x_0)}$ folgt

$$\begin{aligned}
 s &= \overline{(x_1 \oplus x_0)} \ddot{u}' + (x_1 \oplus x_0) \ddot{u}' \\
 &= (x_1 \oplus x_0) \oplus \ddot{u}' \tag{6.8}
 \end{aligned}$$

$$\begin{aligned}
 \ddot{u} &= \overline{x_1} x_0 \ddot{u}' + x_1 \overline{x_0} \ddot{u}' + x_1 x_0 \ddot{u}' + x_1 x_0 \ddot{u}' \\
 &= (\overline{x_1} x_0 + x_1 \overline{x_0}) \ddot{u}' + x_1 x_0 \\
 &= (x_1 \oplus x_0) \ddot{u}' + x_1 x_0 \tag{6.9}
 \end{aligned}$$

Mit Hilfe der Gleichungen 6.8 und 6.9 bildet man das Schaltnetz eines Volladdierers für eine Binärstelle (Abb. 6.5).

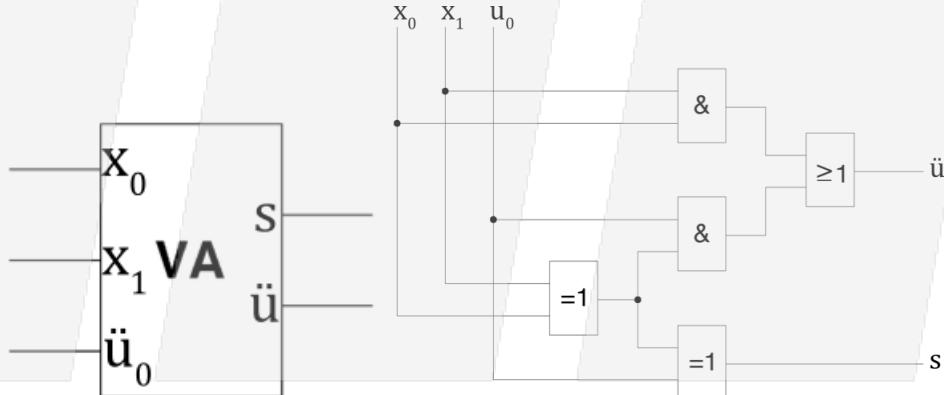


Abbildung 6.5: Schaltzeichen und Schaltnetz eines Volladdierers

Abbildung 6.6 zeigt die Entwicklung eines Volladdierers aus zwei Halbaddierern. Sei s' die Summe und \ddot{u}' der Übertrag des 1. Halbaddierers, \ddot{u}'' der Übertrag des 2. Halbaddierers und \ddot{u}''' der Übertragseingang des Schaltnetzes. Der Übertrag \ddot{u} des Volladdierers berechnet sich wie folgt:

$$s' = x_1 \oplus x_0 \tag{6.10}$$

$$\ddot{u}' = x_1 x_0 \tag{6.11}$$

$$\begin{aligned}
 \ddot{u}'' &= s' \ddot{u}''' \\
 &= (x_1 \oplus x_0) \ddot{u}''' \tag{6.12}
 \end{aligned}$$

$$\ddot{u} = \ddot{u}' + \ddot{u}'' \tag{6.13}$$

$$s = s' \oplus \ddot{u}''' \tag{6.14}$$

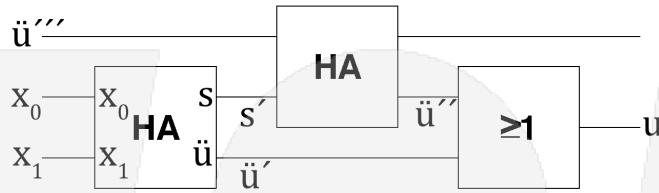


Abbildung 6.6: Aufbau eines Volladdierers aus zwei Halbaddierern

6.2.2 Schaltketten

Unter einer *Schaltkette* versteht man mehrstufige Schaltnetze, in denen mehrere Teilschaltnetze gleicher Struktur kettenartig hintereinander geschaltet sind (Abb. 6.7). Nicht nur die Eingangssignale einer Stufe, sondern auch Übergangssignale der vorhergehenden Stufen sind zu beachten. Als Beispiele werden im Weiteren ein Vergleicher und ein Addierer betrachtet.

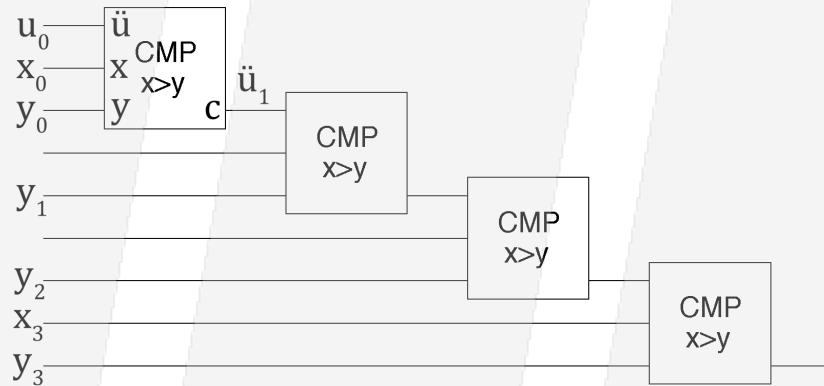


Abbildung 6.7: Struktur einer Schaltkette am Beispiel eines 4-stelligen Vergleichers (CMP)

Vergleicher

Zwei n -stellige Binärzahlen sollen stellenweise verglichen werden. Es ist ein Schaltnetz zu entwerfen, welches ein Ausgangssignal c liefert, mit:

$$\begin{aligned} x &= (x_{n-1}, \dots, x_0) \\ y &= (y_{n-1}, \dots, y_0) \end{aligned}$$

$$\begin{aligned} c &= \ddot{u}_n \\ &= x > y \end{aligned}$$

Die Funktionswerte für ein beliebiges \ddot{u}_{i+1} sind Tabelle 6.3 zu entnehmen.

Vereinfachung der DNF:

$$\begin{aligned} \ddot{u}_{i+1} &= \overline{\ddot{u}_i} x_i \overline{y_i} + \ddot{u}_i \overline{x_i} \overline{y_i} + \ddot{u}_i x_i \overline{y_i} + \ddot{u}_i x_i y_i \\ &= x_i \overline{y_i} + \ddot{u}_i x_i + \ddot{u}_i \overline{y_i} \\ &= x_i \overline{y_i} + \ddot{u}_i (x_i + \overline{y_i}) \end{aligned} \tag{6.15}$$

Mit Hilfe der Gleichung 6.15 entwickelt man das in Abbildung 6.8 wiedergegebene Schaltnetz eines Vergleichers für eine Binärstelle.

Eine Vergleicher-Schaltkette für vier Binärstellen ist in Abbildung 6.9 dargestellt.

Tabelle 6.3: Funktionswerte für eine Binärstelle beim Vergleich $x > y$

\ddot{u}_i ankommender Übertrag	x_i	y_i	\ddot{u}_{i+1} abgehender Übertrag
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

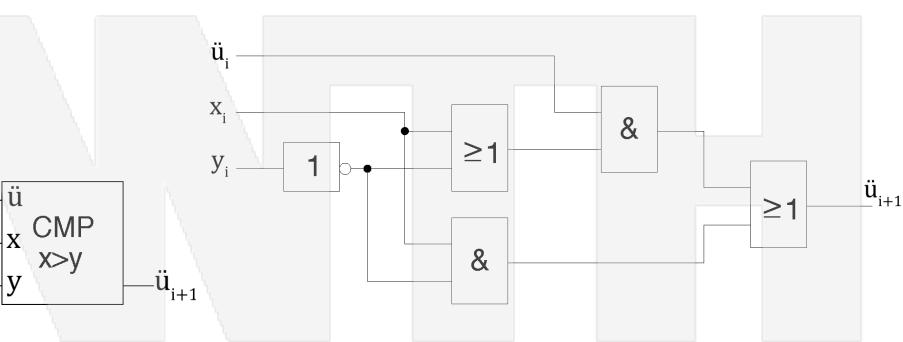


Abbildung 6.8: Schaltzeichen und Schaltnetz eines Vergleichers

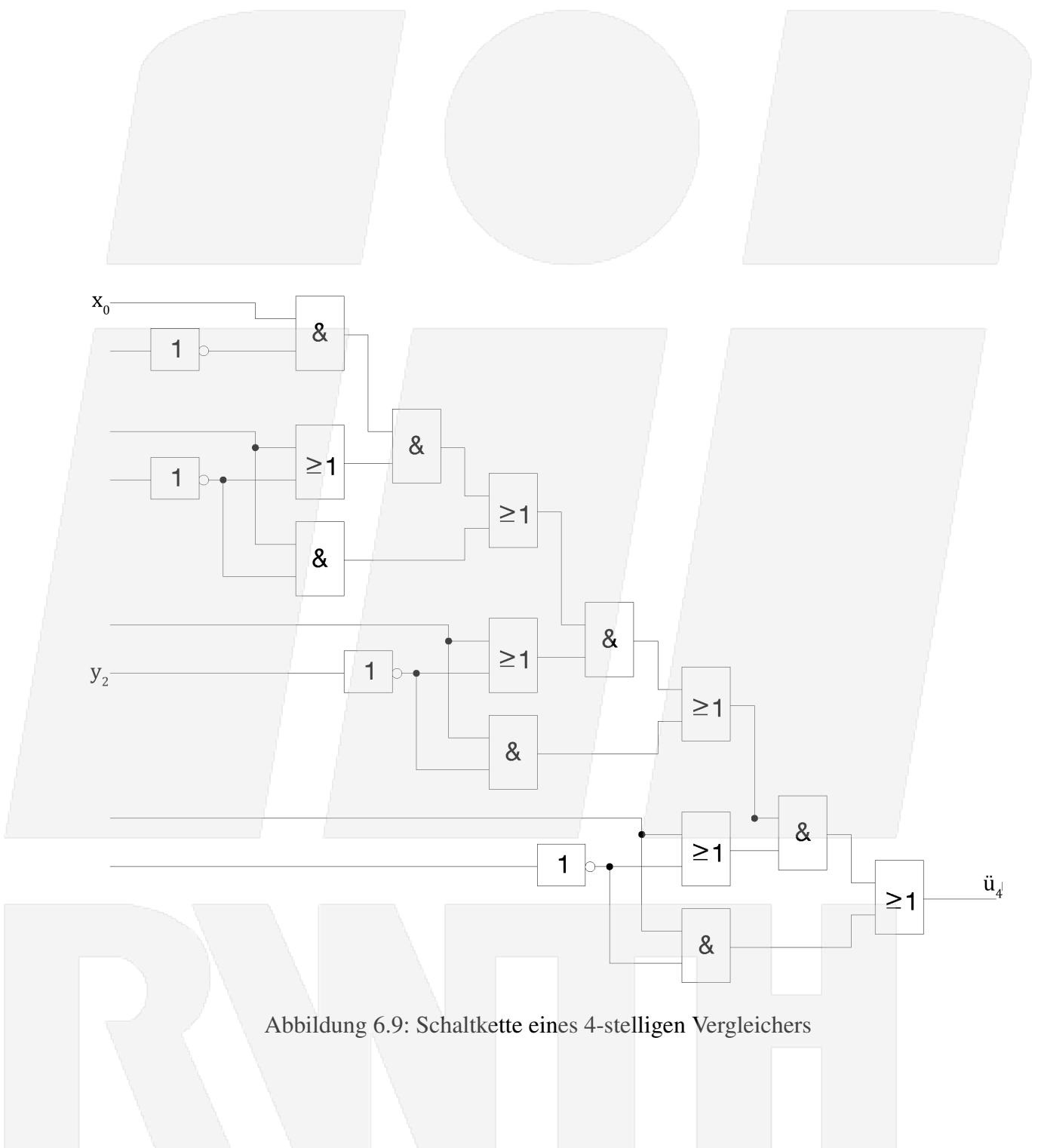


Abbildung 6.9: Schaltkette eines 4-stelligen Vergleichers

Addierer

Zwei n -stellige Dualzahlen x und y mit

$$\begin{aligned}x &= (x_{n-1}, \dots, x_0) \\y &= (y_{n-1}, \dots, y_0)\end{aligned}$$

sollen in einem Schaltnetz addiert werden. Das Ergebnis ist eine $(n + 1)$ - stellige Dualzahl s :

$$s = (s_n, s_{n-1}, \dots, s_0)$$

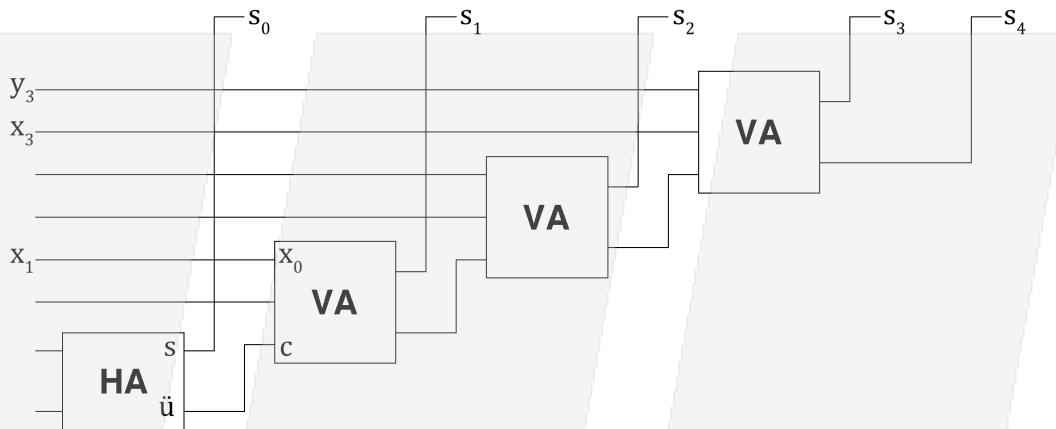


Abbildung 6.10: Schaltkette zur Addition 4-stelliger Dualzahlen x und y

Die Schaltkette wird aus Volladdierern zusammengesetzt. Für die geringstwertige Stelle genügt ein Halbbaddierer (Abb. 6.10).

Laufzeiten bei Schaltketten

Die *Laufzeit* von Schaltketten berechnet sich nach folgender Formel:

mit	t_l	=	$(n_k \cdot n_s \cdot t_s)$
	: Laufzeit		
	: Anzahl der Kettenglieder (Stufen)		
	: Stufigkeit eines Kettenglieds (Anzahl der hintereinandergeschalteten Gatter)		
	: Laufzeit eines Gatters		

Eine Laufzeitverkürzung lässt sich erreichen durch:

- Zusammenfassen von Kettengliedern (Abb. 6.11)
- Umformung von Kettengliedern in die (aufwändigeren) 2-stufige Form.

Die Funktionsgleichungen zusammengefasster Kettenglieder ergeben sich durch Einsetzen.

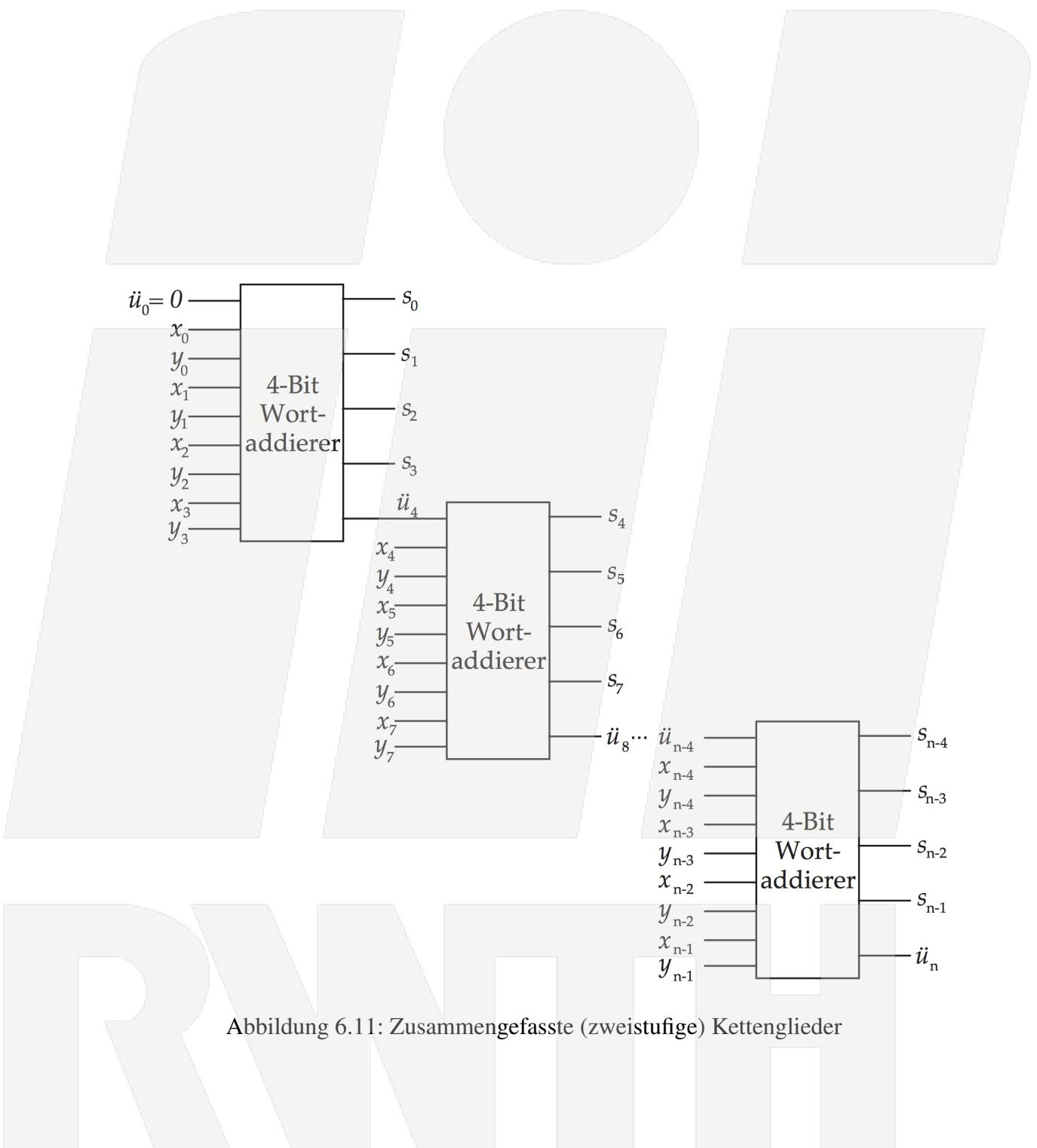


Abbildung 6.11: Zusammengefasste (zweistufige) Kettenglieder

Beispiel: Zusammenfassung zweier Volladdierer (vergl. Gleichung 6.8 und 6.9)

$$\begin{aligned}
 \ddot{u}_{i+2} &= x_{i+1}y_{i+1} + (x_{i+1} \oplus y_{i+1})\ddot{u}_{i+1} \\
 \ddot{u}_{i+1} &= x_iy_i + (x_i \oplus y_i)\ddot{u}_i \\
 \ddot{u}_{i+2} &= x_{i+1}y_{i+1} + (x_{i+1} \oplus y_{i+1})(x_iy_i + (x_i \oplus y_i)\ddot{u}_i) \\
 s_i &= (x_i \oplus y_i) \oplus \ddot{u}_i \\
 s_{i+1} &= (x_{i+1} \oplus y_{i+1}) \oplus \ddot{u}_{i+1} \\
 s_{i+1} &= (x_{i+1} \oplus y_{i+1}) \oplus (x_iy_i + (x_i \oplus y_i)\ddot{u}_i)
 \end{aligned}$$

Die Gleichungen s_i , s_{i+1} und \ddot{u}_{i+2} müssen noch in ihre zweistufige Form umgeformt werden. Man erhält:

$$\begin{aligned}
 \ddot{u}_{i+2} &= x_{i+1}y_{i+1} + x_{i+1}\bar{y}_{i+1}\ddot{u}_{i+1} + \bar{x}_{i+1}y_{i+1}\ddot{u}_{i+1} \\
 s_i &= \bar{x}_iy_i\ddot{u}_i + x_i\bar{y}_i\ddot{u}_i + \bar{x}_i\bar{y}_i\ddot{u}_i + x_iy_i\ddot{u}_i \\
 s_{i+1} &= \bar{x}_{i+1}y_{i+1}\bar{\ddot{u}}_{i+1} + x_{i+1}\bar{y}_{i+1}\bar{\ddot{u}}_{i+1} + \bar{x}_{i+1}\bar{y}_{i+1}\ddot{u}_{i+1} + x_{i+1}y_{i+1}\ddot{u}_{i+1}
 \end{aligned}$$

6.2.3 Codeumsetzer

Exzess-3 → Dezimal

Tabelle 6.4 gibt die Funktionswerte für die Codeumsetzung wieder.

Nachfolgend sind die Funktionsgleichungen für die Exzess-3 → Dezimal-Codeumsetzung aufgeführt. Der Ausgang y_{10} dient der Pseudotetradenerkennung. Abbildung 6.12 zeigt das KV-Diagramm für die Ausgänge y_0 bis y_9 .

$$\begin{aligned}
 y_0 &= \bar{x}_2 \bar{x}_3 \\
 y_1 &= \bar{x}_0 \bar{x}_1 \bar{x}_3 \\
 y_2 &= x_0 \bar{x}_1 \bar{x}_3 \\
 y_3 &= \bar{x}_0 x_1 x_2 \\
 y_4 &= x_0 x_1 x_2 \\
 y_5 &= \bar{x}_0 \bar{x}_1 \bar{x}_2 \\
 y_6 &= x_0 \bar{x}_1 \bar{x}_2 \\
 y_7 &= \bar{x}_0 x_1 \bar{x}_2 \\
 y_8 &= x_0 x_1 x_3 \\
 y_9 &= x_2 x_3 \\
 y_{10} &= \bar{x}_0 \bar{x}_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 \bar{x}_3 + x_0 x_2 x_3 + x_1 x_2 x_3
 \end{aligned}$$

Abbildung 6.13 gibt das zugehörige Schaltnetz wieder.

Tabelle 6.4: Exzess-3 → Dezimal

i	x_3	x_2	x_1	x_0	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	0	0	1	*	*	*	*	*	*	*	*	*	*
1	0	0	0	1	1	*	*	*	*	*	*	*	*	*	*
2	0	0	1	0	1	*	*	*	*	*	*	*	*	*	*
3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
4	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
5	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0
6	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0
7	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0
8	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0
10	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0
11	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0
12	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0
13	1	1	0	1	1	*	*	*	*	*	*	*	*	*	*
14	1	1	1	0	1	*	*	*	*	*	*	*	*	*	*
15	1	1	1	1	1	*	*	*	*	*	*	*	*	*	*

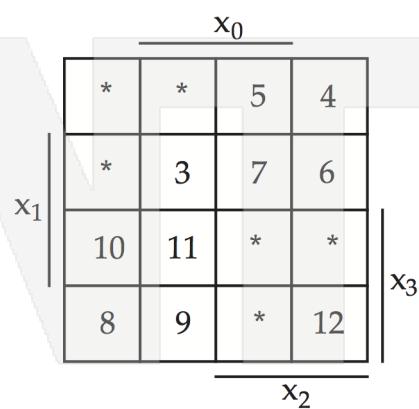


Abbildung 6.12: KV-Diagramm mit *don't care* Feldern

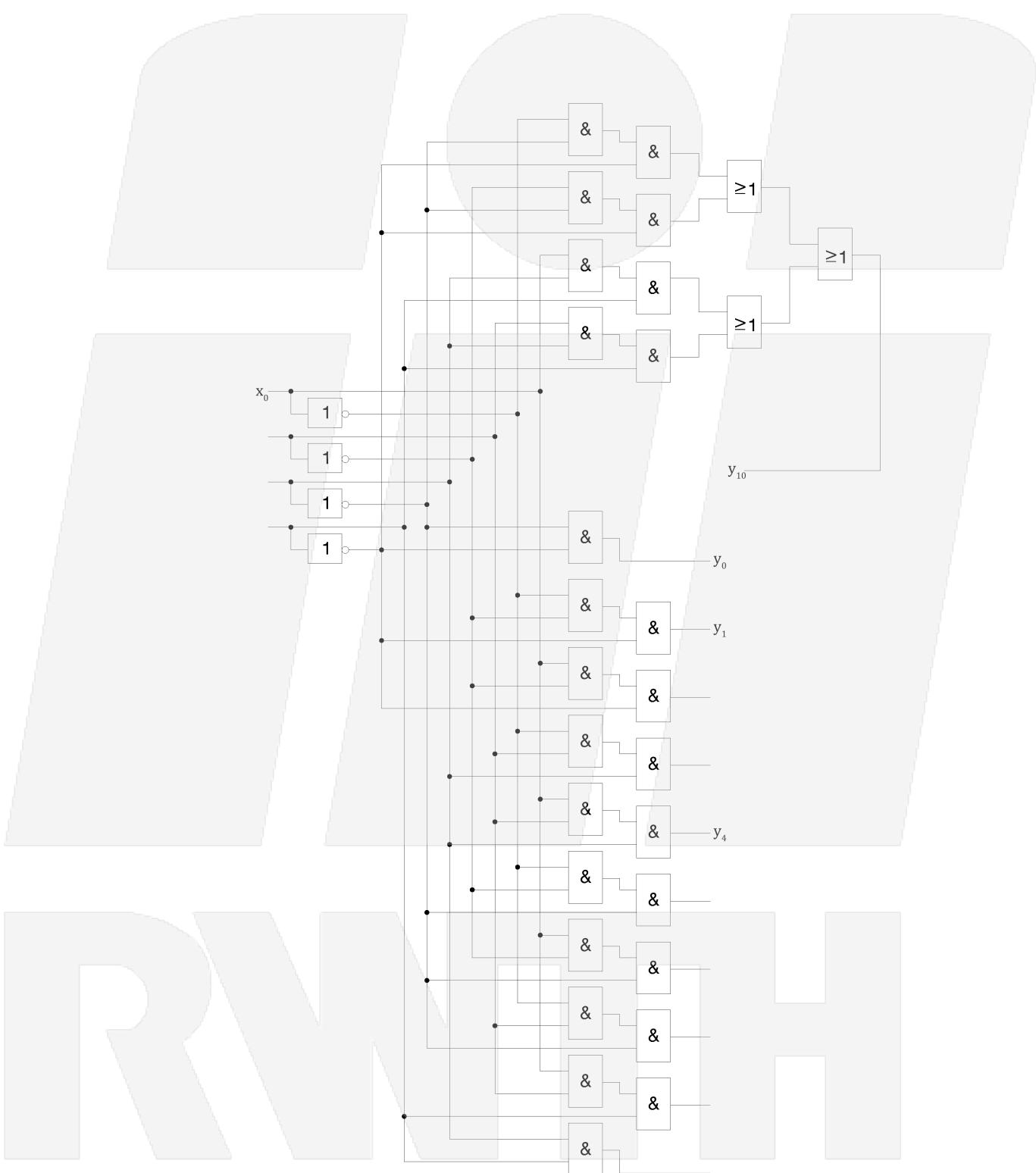


Abbildung 6.13: Exzess-3 → Dezimal-Codeumsetzer mit Pseudotetradenerkennung

Dual-Code → Gray-Code

Tabelle 6.5 zeigt die Zuordnung der Dezimalziffern zum Gray- und Dual-Code.

Tabelle 6.5: Gray-Code → Dual-Code → Dezimalziffern

Gray	Dual	Dezimal
0000	0000	0
0001	0001	1
0011	0010	2
0010	0011	3
0110	0100	4
0111	0101	5
0101	0110	6
0100	0111	7
1100	1000	8
1101	1001	9
1111	1010	10
1110	1011	11
1010	1100	12
1011	1101	13
1001	1110	14
1000	1111	15

Das Wandlungskonzept für n -stellige Zahlen nutzt die folgenden Zusammenhänge:

Sei D_i die i -te Ziffer der Dualzahl und
 G_i die i -te Ziffer der Zahl im Gray-Code. Dann gilt:

$$\begin{array}{l}
 \text{Gray-Code} \rightarrow \text{Dual-Code} \\
 \begin{aligned}
 D_n &= G_n \\
 D_i &= \overline{D_{i+1}} \cdot G_i + D_{i+1} \cdot \overline{G_i} \\
 &= D_{i+1} \oplus G_i
 \end{aligned}
 \end{array}$$

$$\begin{array}{l}
 \text{Dual-Code} \rightarrow \text{Gray-Code} \\
 \begin{aligned}
 G_n &= D_n \\
 G_i &= D_i \cdot \overline{D_{i+1}} + \overline{D_i} \cdot D_{i+1} \\
 &= D_i \oplus D_{i+1}
 \end{aligned}
 \end{array}$$

Die Schaltkette für die Code-Umsetzung in beide Richtungen ist in Abbildung 6.14 dargestellt.

Die Umsetzung erfolgt in Abhängigkeit von s.

$$\begin{array}{ll} s = 0 & \\ \text{Gray-Code} \rightarrow \text{Dual-Code} & \\ s = 1 & \\ \text{Dual-Code} \rightarrow \text{Gray-Code} & \end{array}$$

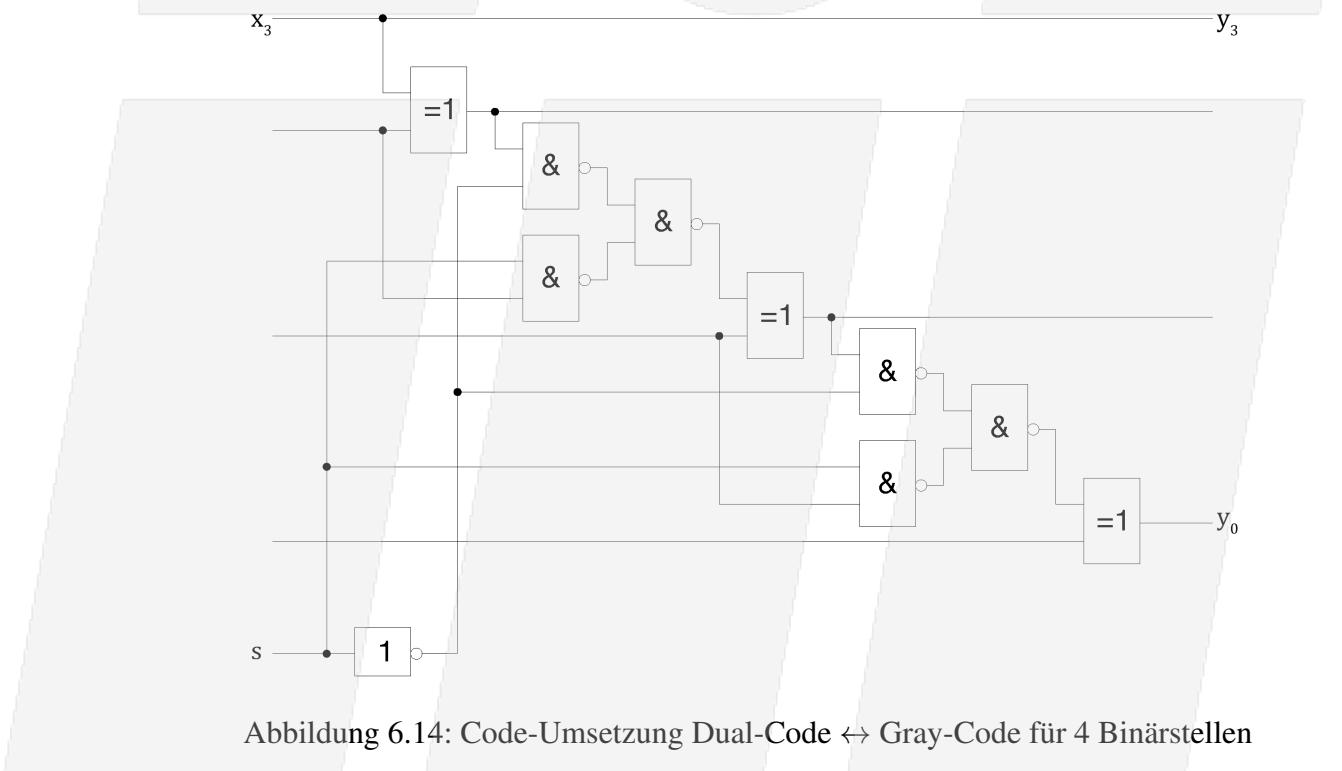


Abbildung 6.14: Code-Umsetzung Dual-Code \leftrightarrow Gray-Code für 4 Binärstellen

6.2.4 Datenwähler

Demultiplexer

Ein *Demultiplexer* (DEMUX) schaltet ein Eingangssignal auf eine von 2^n Ausgangsleitungen.

s_1	s_0	y_0	y_1	y_2	y_3
0	0	x	0	0	0
0	1	0	x	0	0
1	0	0	0	x	0
1	1	0	0	0	x

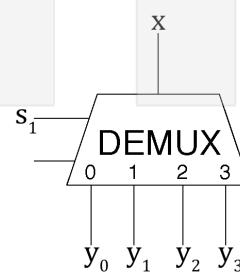


Abbildung 6.15: Funktionstabelle und Schaltzeichen DEMUX

Abbildung 6.15 gibt das Schaltzeichen und die Funktionstabelle und Abbildung 6.16 das Schaltnetz eines Demultiplexers wieder.

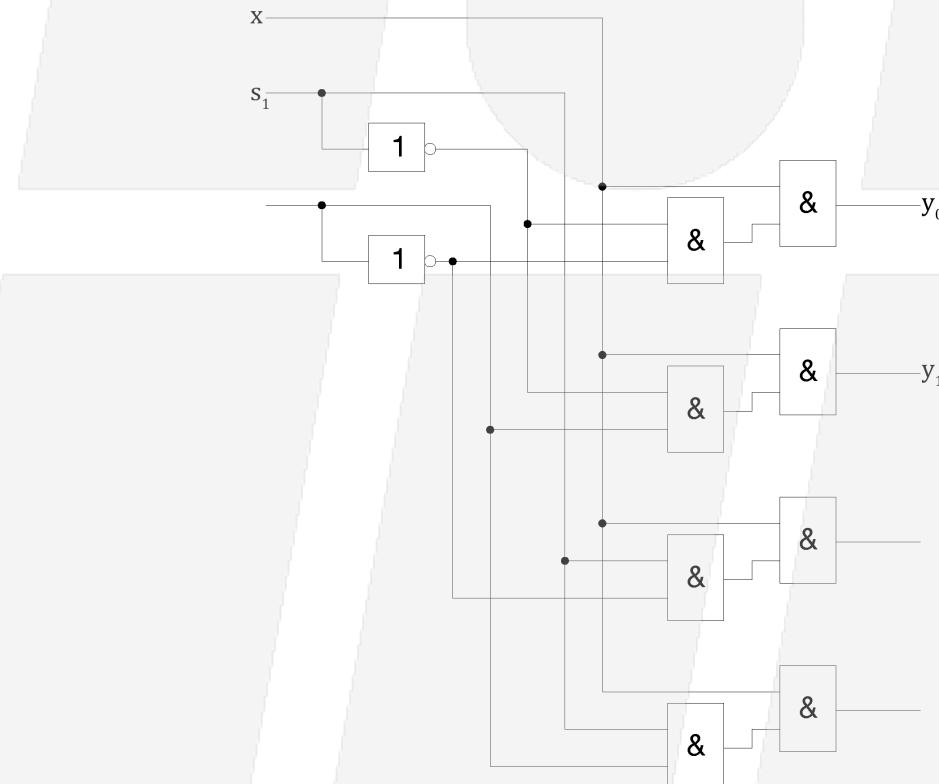


Abbildung 6.16: 1-auf-4-DEMUX

Multiplexer

Ein *Multiplexer (MUX)* schaltet eines von 2^n Eingangssignalen über ein n -bit langes Steuerwort auf einen Ausgang. Funktionstabelle und Schaltzeichen sind Abbildung 6.17 zu entnehmen.

s_1	s_0	y
0	0	x_0
0	1	x_1
1	0	x_2
1	1	x_3

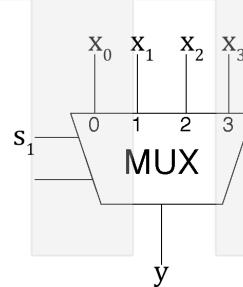


Abbildung 6.17: Funktionstabelle und Schaltzeichen MUX

Abbildung 6.18 gibt das Schaltnetz eines Multiplexers wieder.

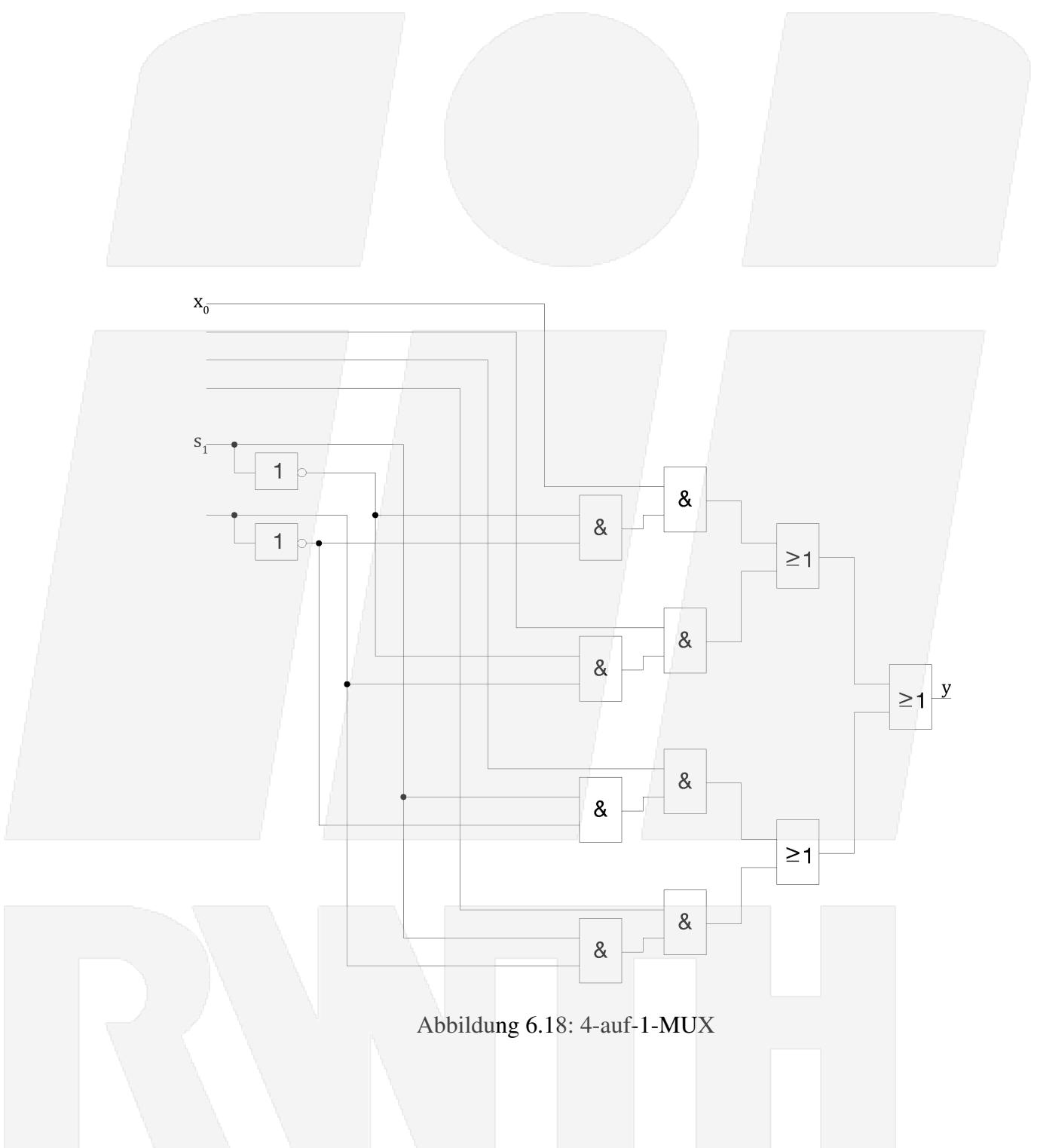


Abbildung 6.18: 4-auf-1-MUX

6.2.5 Kreuzschienenverteiler (crossbar switch)

Kreuzschienenverteiler (Abb. 6.19) werden zur beliebigen Verbindung von Eingangs- mit Ausgangsleitungen verwendet (z. B. rekonfigurierbarer Rechnervernetzungen).

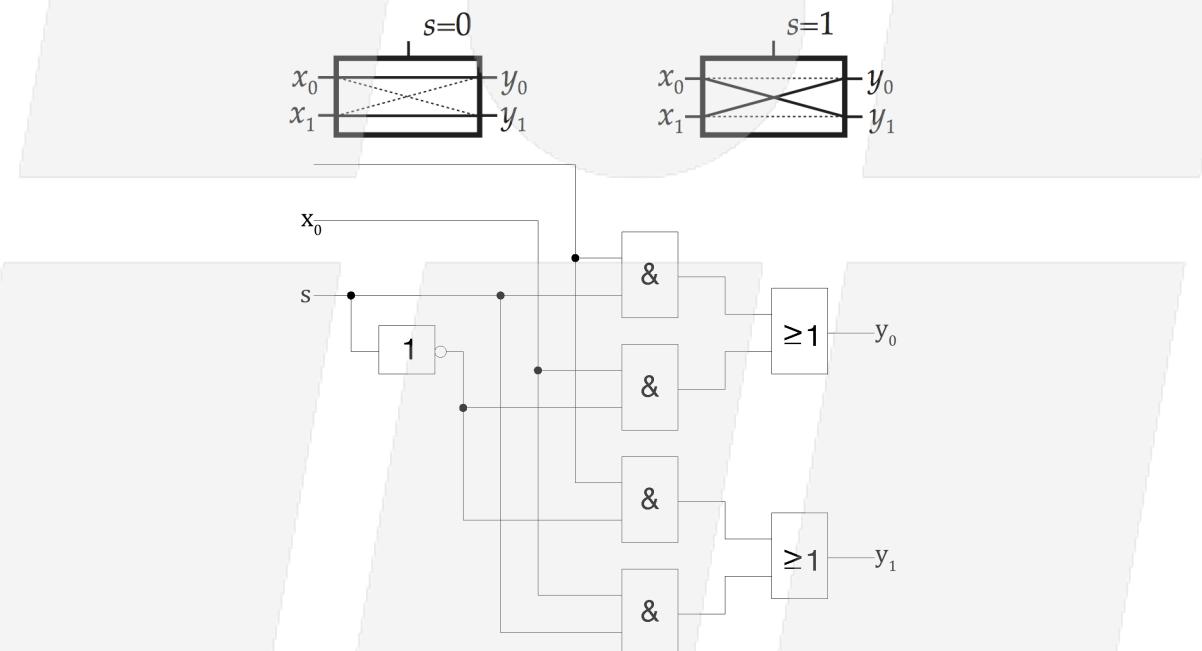


Abbildung 6.19: Schaltzeichen und Schaltnetz eines Kreuzschienenverteilers

Eine Zusammenschaltung von Kreuzschienenverteilern wie in Abbildung 6.20 nennt man Permutationsnetzwerk.

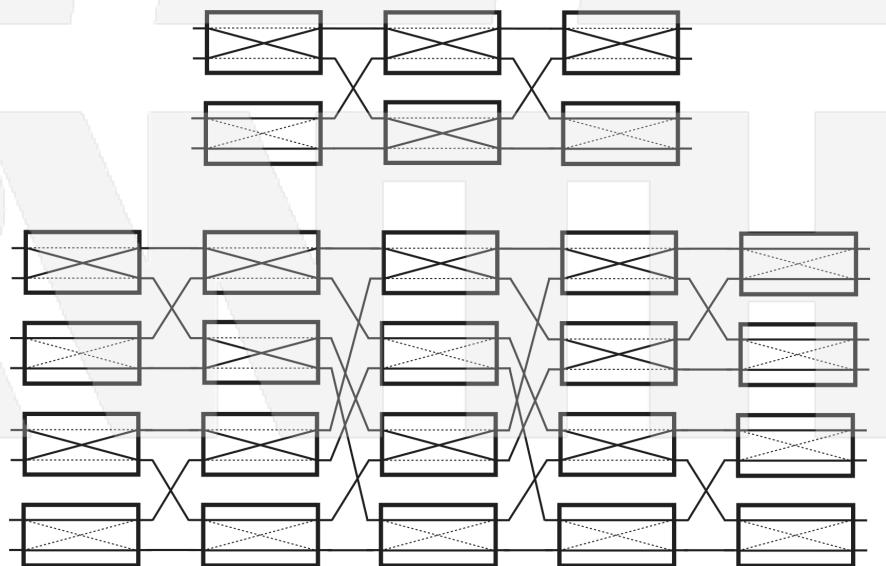


Abbildung 6.20: 4- und 8-fach-Permutationsnetzwerk

6.3 Speicherglieder

Speicherglieder dienen der Speicherung von Daten auf unbegrenzte Zeit (sofern die Stromversorgung gewährleistet ist). Die Daten sollen beliebig abgefragt und geändert werden können. Das RS-Flipflop ist ein einfaches Speicherglied, das diese Anforderungen erfüllt. Die Abkürzung RS bezieht sich auf die beiden Zustände R = reset für Löschen und S = set für Setzen.

6.3.1 RS-Flipflop (Latch)

Abbildung 6.21 zeigt einen asynchronen Oszillator aus einem rückgekoppelten NAND-Gatter. Die Tabelle gibt die zugehörigen Zustände wieder. Δt bezeichnet die Gatterlaufzeit

x	$f_{t+\Delta t}$	$f_{t+2\Delta t}$	$f_{t+3\Delta t}$
0	1	1	1
1	$\neg f_t$	f_t	$\neg f_t$

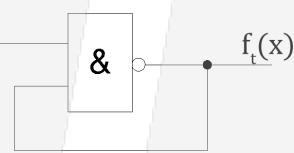


Abbildung 6.21: Zustandstabelle und Schaltung eines asynchronen Oszillators

Die Schaltung eines *RS-Flipflops* aus NOR-Gattern ist in Abbildung 6.22 dargestellt¹. Das zugehörige Impulsdiagramm ist Abbildung 6.23 zu entnehmen. Bistabile Kippschaltungen aus rückgekoppelten Gattern können zwei stabile Zustände annehmen.

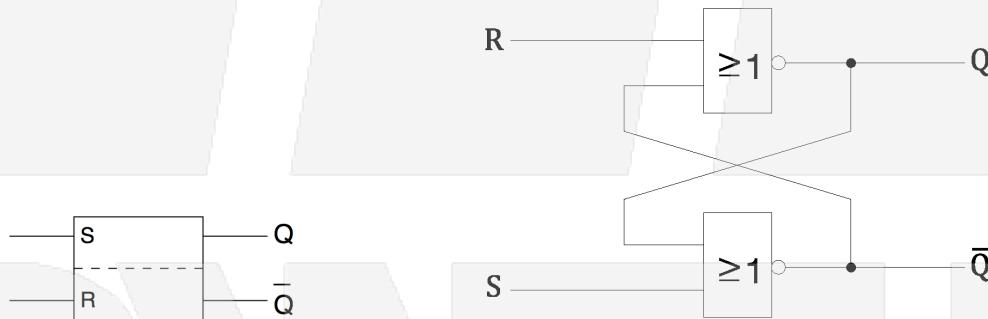


Abbildung 6.22: RS-Flipflop: Schaltzeichen und Schaltnetz aus zwei rückgekoppelten NOR-Gattern

Die zugehörigen Zustände sind Tabelle 6.6 zu entnehmen.

6.3.2 RS-Flipflop mit Zustandssteuerung

Beim asynchronen RS-Flipflop wird die anliegende Ansteuerung sofort wirksam. Bei synchronen (getakteten) Schaltungen (Abb. 6.24) wird der Zeitpunkt, zu dem die Eingangsvariablen wirksam werden, durch einen gemeinsamen Grundtakt festgelegt.

¹Es ist auch möglich, Flipflops anders aufzubauen; im Folgenden beschränken wir uns jedoch auf Flipflops mit dem vorgestellten Aufbau aus NOR-Gattern.

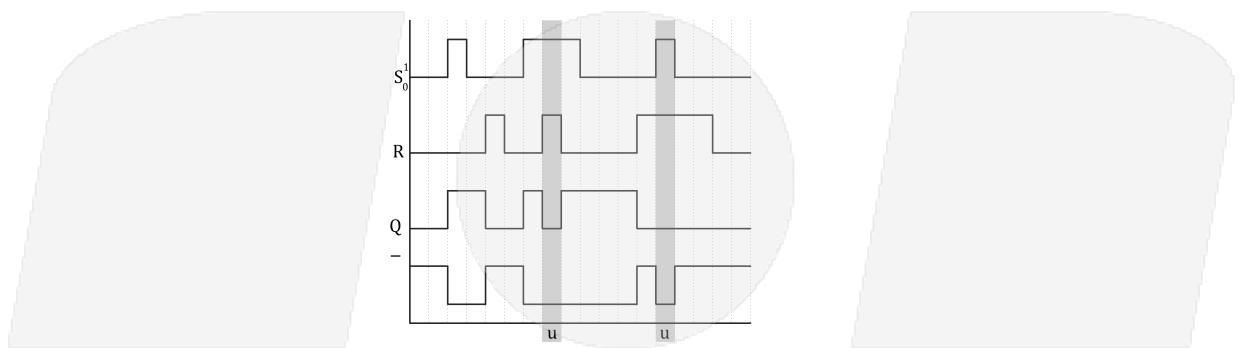


Abbildung 6.23: Impulsdiagramm zum RS-Flipflop (u = unzulässig)

Tabelle 6.6: Zustände des RS-Flipflops

S	R	Q	
0	0	speichern (wie vorher)	
0	1	0	
1	0	1	
1	1	(0)	unzulässig

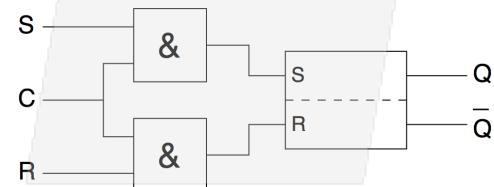


Abbildung 6.24: Schaltzeichen und Schaltnetz des RS-Flipflops mit Zustandssteuerung

Tabelle 6.7: Zustände des RS-Flipflops mit Zustandssteuerung

C	S	R	Q_{n+1}	
0	0	0	Q_n	
0	0	1	Q_n	keine Änderung des Ausgangszustandes
0	1	0	Q_n	
0	1	1	Q_n	
1	0	0	Q_n	Speichern
1	0	1	0	Rücksetzen
1	1	0	1	Setzen
1	1	1	—	unzulässig

Aus Tabelle 6.7 und Abb. 6.25 geht hervor, dass Zustandsänderungen nur bei $C = 1$ möglich sind. Wenn beide Eingänge ($R = S = 1$) eines RS-Flipflops aus NOR-Gattern und zusätzlich die Clock gesetzt sind, dann sind Q und \bar{Q} beide auf Low. Sobald die Clock auf Low wechselt, kann man keine Aussage mehr über Q und \bar{Q} treffen. Die Belegung ist zufällig und schwingt bei idealen Flipflops (identische Gatterlaufzeiten in beiden Zweigen) zwischen High und Low. Diese Reaktion ist unzulässig.

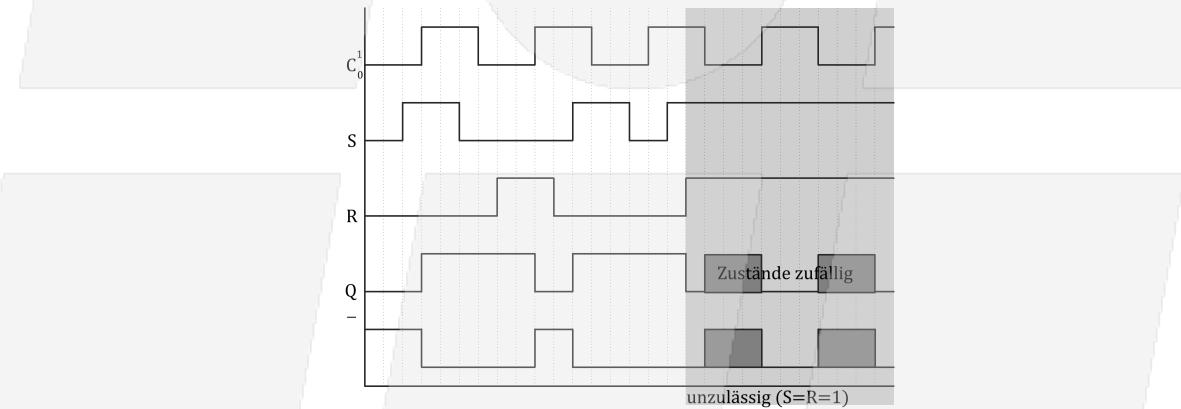


Abbildung 6.25: Impulsdiagramm des RS-Flipflops mit Zustandssteuerung

6.3.3 D-Flipflop mit Zustandssteuerung

Beim D-(Delay)-Flipflop wird die unzulässige Eingangskombination $R = S = 1$ durch die Verknüpfung beider Eingänge über einen Inverter vermieden (Abb. 6.26 u. Tab. 6.8). Die Eingangsvariable D wird mit der Clockvorderflanke übernommen.

Tabelle 6.8: Zustände des D-Flipflops mit Zustandssteuerung

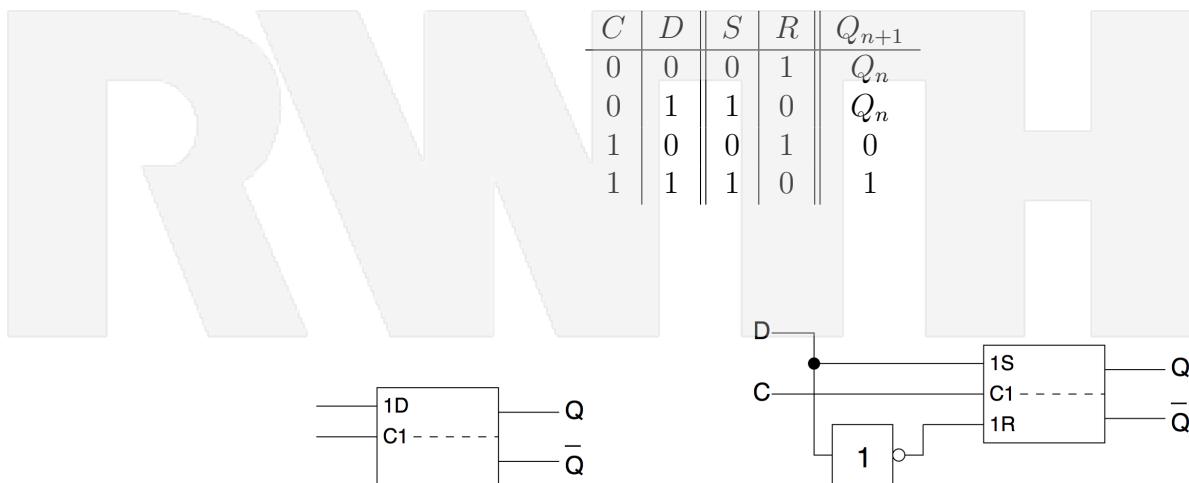


Abbildung 6.26: Schaltzeichen und Schaltnetz des D-Flipflops mit Zustandssteuerung

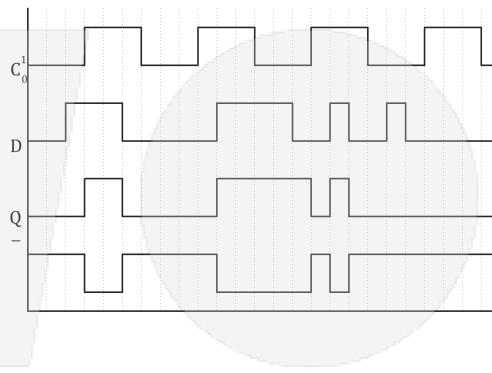


Abbildung 6.27: Impulsdiagramm zum D-Flipflop mit Zustandssteuerung

6.3.4 RS-Flipflop mit Zwei-Zustandssteuerung

Ein RS-Flipflop mit Zwei-Zustandssteuerung besteht aus zwei hintereinandergeschalteten RS-Flipflops (Abb. 6.28). Es wird auch *Master-Slave-Flipflop* genannt. Bei einem Master-Slave-Flipflop sind während des Taktzustands $C = 1$ die Eingangsvariablen für das Master-Flipflop wirksam. Die Übernahme in das Slave-Flipflop erfolgt jedoch erst beim Übergang des Taktes von $C = 1$ nach $C = 0$ (Abb. 6.29). Durch diese Maßnahme wird vermieden, dass bei hintereinandergeschalteten Flipflops, wie etwa in Registern und Zählern, ein Eingangssignal über den gemeinsamen Takt direkt durch alle Glieder durchgeschaltet wird. Eine solche bei abfallender Taktflanke durchschaltende Anordnung nennt man *rückflankengekenngetriggert*. Die Wahrheitstabelle entspricht dabei der des Flipflops mit Zustandssteuerung (Tabelle 6.7).

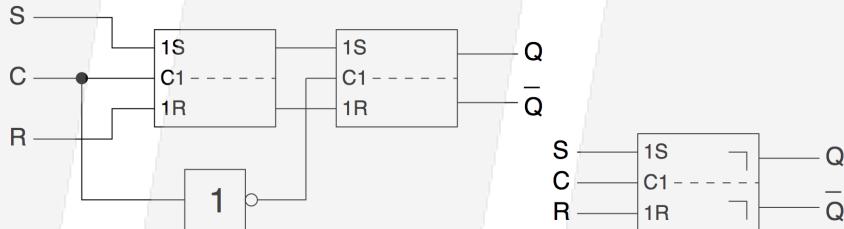


Abbildung 6.28: Schaltung des RS-Flipflops mit Zwei-Zustandssteuerung (Master-Slave-Flipflop)

Abbildung 6.30 zeigt ein D-Flipflop mit Rückflankentriggierung, das als Verzögerungsglied eingesetzt werden kann. Abbildung 6.31 zeigt das zugehörige Impulsdiagramm. Die Wahrheitstabelle entspricht Tabelle 6.8.

6.3.5 J-K-Flipflop

Das *J-K-Flipflop* (Abb. 6.32) wurde aus dem RS-Flipflop entwickelt, um den nicht definierten Zustand $R = S = 1$ zu umgehen (Abb. 6.25). Anders als beim D-Flipflop werden beide Eingänge verwendet. Das Flipflop besitzt die folgenden Eigenschaften:

- Es wechselt den Zustand am Ausgang nach einem Clocksignal, wenn an beiden Eingängen eine 1 anliegt.
- Die Set-Funktion wird nur dann ausgelöst, wenn \bar{Q} logisch 1 ist.
- Die Reset-Funktion wird nur dann ausgelöst, wenn Q logisch 1 ist.

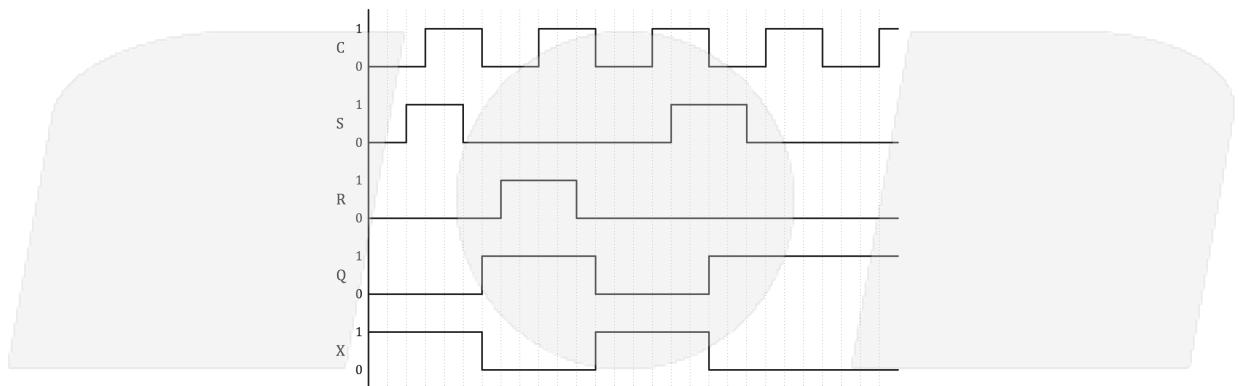


Abbildung 6.29: Impulsdiagramm zum Master-Slave-Flipflop

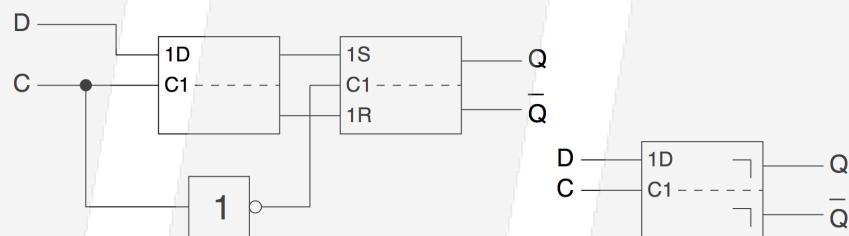


Abbildung 6.30: Schaltung des D-Flipflops mit Rückflankentriggerung

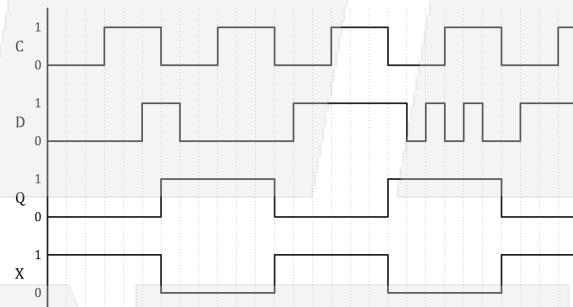


Abbildung 6.31: Impulsdiagramm zum D-Flipflop mit Rückflankentriggerung

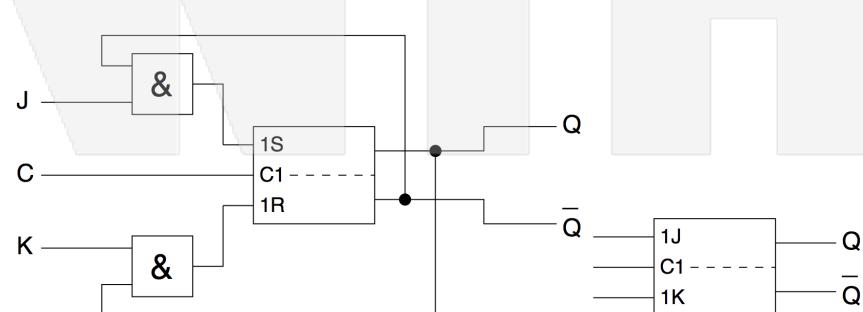


Abbildung 6.32: Schaltung des J-K-Flipflops

J-K-Flipflop mit Rückflankensteuerung

Um zu verhindern, dass die Schaltung im Zustand $C = J = K = 1$ oszilliert (Zustandswechsel an Q bzw. \bar{Q}), wird die Master-Slave Anordnung aus Abbildung 6.33 verwendet. Der Zustandswechsel (Invertieren der gespeicherten Information) erfolgt dann bei $J = K = 1$ jeweils zum Zeitpunkt der abfallenden Flanke (Abb. 6.34).

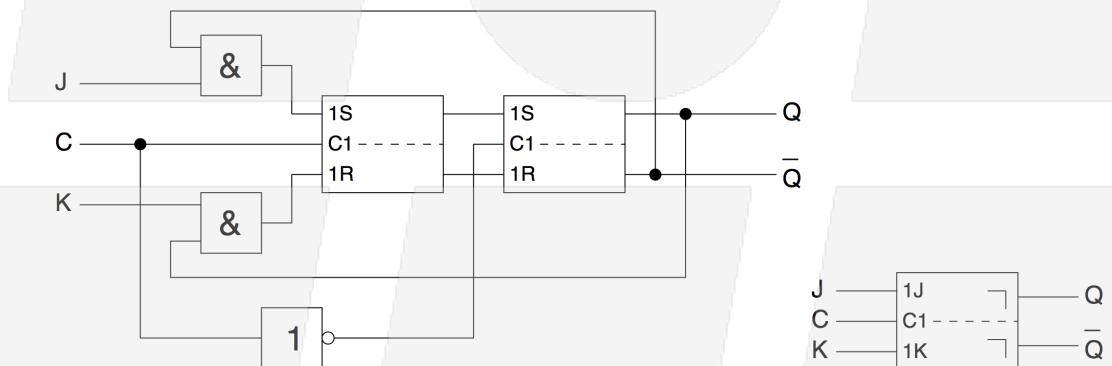


Abbildung 6.33: Schaltung des J-K-Flipflops mit Rückflankentriggerung

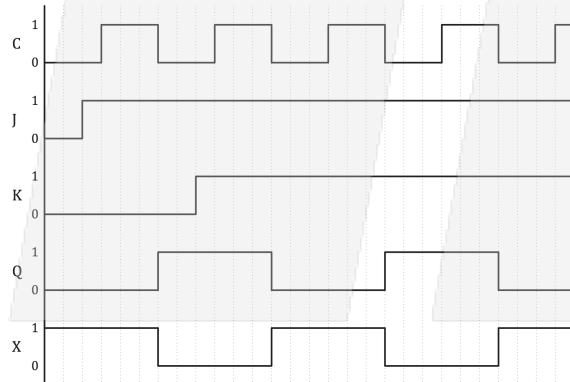


Abbildung 6.34: Impulsdiagramm zum J-K-Flipflop mit Rückflankentriggerung

Tabelle 6.9: Zustands- und Synthesetabelle des JK-Flipflops. '*' bezeichnet don't care Bedingungen bei Zustandswechseln.

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

$Q_n \rightarrow Q_{n+1}$	J	K
$0 \rightarrow 0$	0	*
$0 \rightarrow 1$	1	*
$1 \rightarrow 0$	*	1
$1 \rightarrow 1$	*	0

6.3.6 Zähler

Die Grundbausteine binärer *Zähler* sind Flipflops, deren geeignete Verschaltung das für das Zählen notwendige fortlaufende Speichern und Addieren ermöglicht. Dabei wird eine vorher festgelegte Sequenz von Zuständen abhängig von Impulsen an einem *Clock*-Eingang durchlaufen. Werden alle Flipflops bei einem Zählvorgang gleichzeitig geschaltet, so spricht man von Synchronzählern. Erfolgt das Schalten der Zählerflipflops in einer zeitlichen Abfolge, dann handelt es sich um Asynchronzählern.

Asynchrone Zähler

Als ein wichtiger Vertreter asynchroner Zähler wird ein Aufwärtszähler betrachtet. Dabei ist in Abbildung 6.35 der Zähler auf 4 Bit beschränkt. Die Zählerkette kann jedoch problemlos durch Anfügen weiterer Flipflops beliebig erweitert werden. Es wird davon ausgegangen, dass

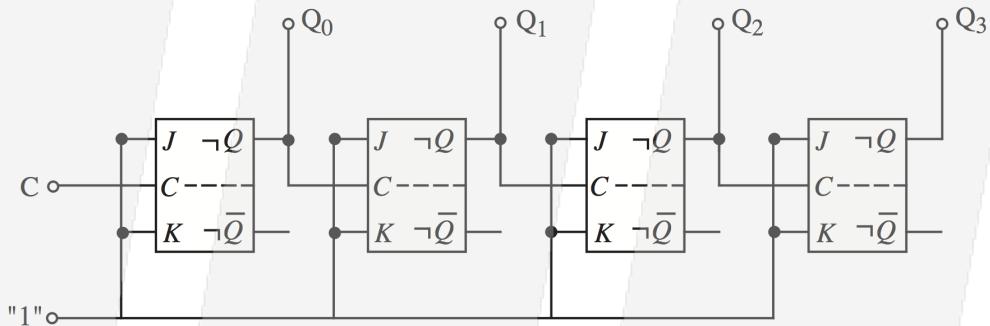


Abbildung 6.35: 4-Bit Asynchronzähler

alle Flipflops zu Beginn zurückgesetzt sind, d.h. alle Ausgänge Q logisch 0 sind. Trifft nun der erste zu zählende Impuls am Eingang des ersten Flipflops ein, dann schaltet dieses mit der abfallenden Flanke um. Der Ausgang Q_0 wird logisch 1. Mit der nächsten negativen Flanke wechselt Q_0 von logisch 1 auf logisch 0, was ein Impuls für das zweite Flipflop darstellt. Dies lässt sich analog für die weiteren Flipflops fortführen. Abbildung 6.36 zeigt das zugehörige Impulsdigramm.

Synchrone Zähler

Bei einem synchronen Zähler werden alle Flipflops gleichzeitig durch einen gemeinsamen Impuls weitergeschaltet. Daraus ergibt sich, dass der Wechsel der einzelnen Stufen zwischen logisch 0 und logisch 1 durch die Eingänge J und K erfolgen muss. Bei einem 4-Bit Synchronaufwärtszähler erfolgt die Ansteuerung der J und K Eingänge über zwei UND-Gatter. Abbildung 6.37 zeigt einen 4-Bit Synchronaufwärtszähler. Der Zähler kann nicht durch Anfügen von zusätzlichen Flipflops beliebig erweitert werden. Die Setz- und Rücksetzbedingungen für weitere Flipflops müssen jeweils explizit ermittelt werden.

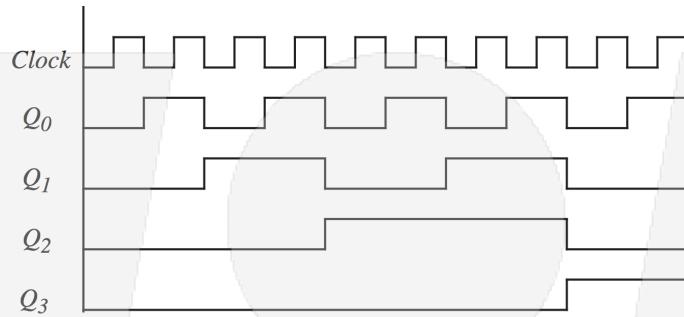


Abbildung 6.36: Impulsdiagramm eines 4-Bit Asynchronzählers

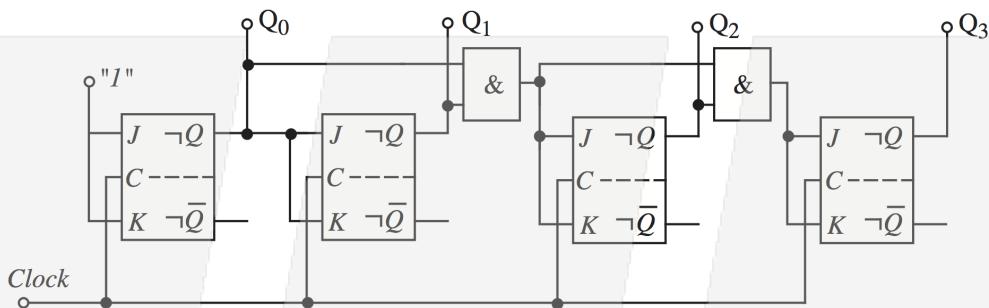


Abbildung 6.37: 4-Bit Synchronzähler

6.3.7 Schieberegister

Schieberegister nehmen Informationen bitweise auf, speichern diese und geben sie auf Abruf wieder ab. Wie die im vorherigen Abschnitt beschriebenen Zähler werden auch Schieberegister aus Flipflops zusammengesetzt.

Schieberegister mit serieller Ein- und Ausgabe

Ein einfaches 4-Bit Schieberegister mit serieller Ein- und Ausgabe ergibt sich durch Hintereinanderschalten von vier D-Flipflops (Abbildung 6.38).

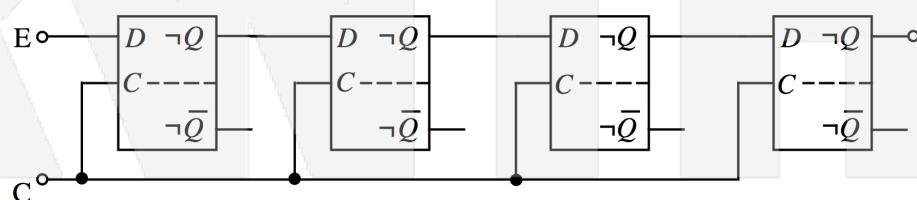


Abbildung 6.38: 4-Bit Schieberegister für serielle Ein- und Ausgabe

Durch den gemeinsamen *Clock*-Eingang wird bewirkt, dass bei jedem Impuls die Information um genau eine Stufe (ein Flipflop) weitergeschaltet wird. Bei dem hier verwendeten 4-Bit

Schieberegister ist die Information nach vier Takten komplett in das Schieberegister eingelesen. Ein Abschalten des Taktes bewirkt das Speichern der Information. Bleibt der Takt anliegen bzw. wird dieser erneut zugeschaltet, so wird die Information nach dem FIFO-Prinzip (First In First Out) seriell ausgegeben.

Schieberegister mit serieller Ein- und paralleler Ausgabe

Viele Schieberegister verfügen zusätzlich zur seriellen Ausgabe auch über die Möglichkeit einer parallelen Ausgabe. Abbildung 6.39 zeigt ein solches Schieberegister aus rückflankengesteuerten RS-Flipflops.

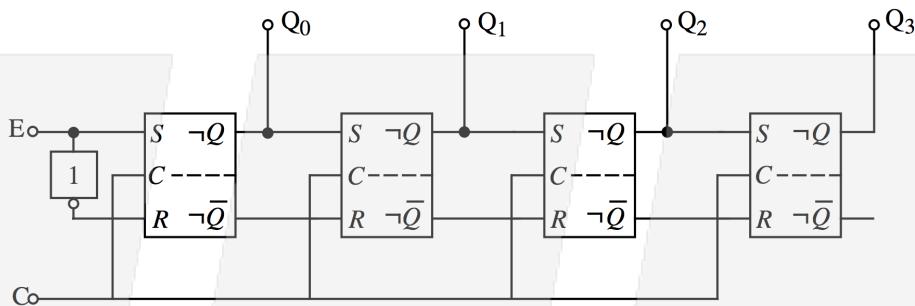


Abbildung 6.39: 4-Bit Schieberegister mit paralleler Ausgabe

Die hier verwendeten RS-Flipflops benötigen gegenüber den in Abbildung 6.38 verwendeten D-Flipflops eine modifizierte Art der Rücksetzung.

Während der Parallelausgabe sind weitere Schiebetakte zu vermeiden, da diese die parallel ausgegebene Information verfälschen würden. Aus diesem Grund wird das Schieberegister mit einer zusätzlichen Verriegelungsschaltung versehen (s. Abbildung 6.40).

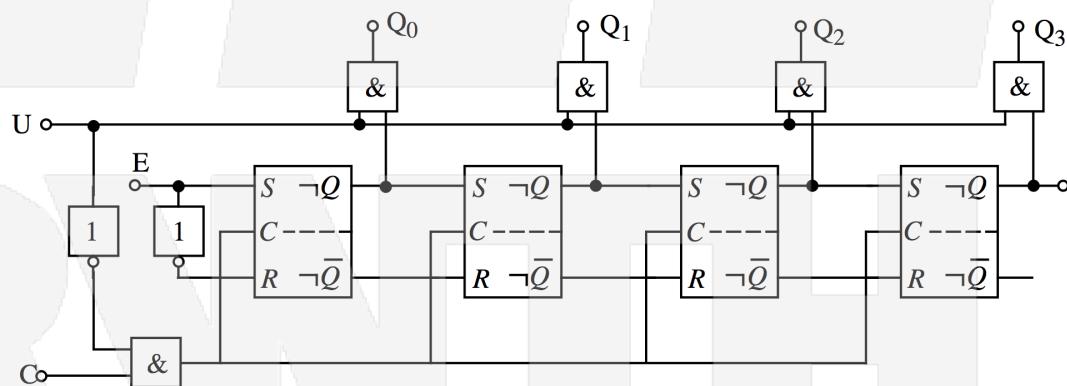


Abbildung 6.40: Schieberegister mit Parallelausgabe und Verriegelung

Liegt am Eingang U logisch 0 an, so verhält sich das Schieberegister in Abbildung 6.40 analog zu dem aus Abbildung 6.39 (getakteter Betrieb). Bei einer logischen 1 am Eingang U wird der Takt T über ein UND-Gatter gesperrt und das Schieberegister kann parallel ausgelesen werden (Parallelausgabe).

Abbildung 6.41 zeigt ein Zeitablaufdiagramm für das Abspeichern der Dualzahl 1010 in einem 4-Bit Schieberegister. Es ist deutlich zu sehen, wie die Bitfolge „durchgeschoben“ wird.

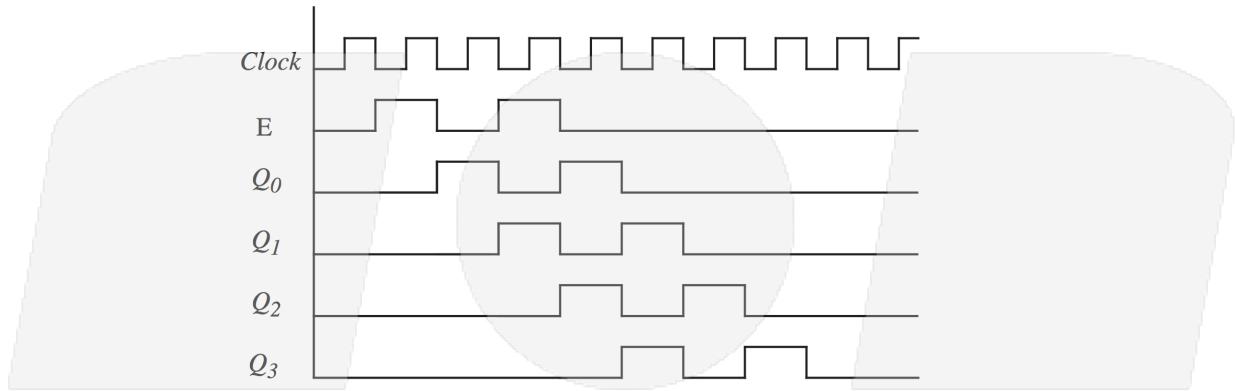


Abbildung 6.41: Impulsdiagramm für ein 4-Bit Schieberegister mit paralleler Ausgabe

Schieberegister mit umschaltbarer Schieberichtung

Neben den oben bereits vorgestellten Schieberegistervarianten werden auch Schieberegister mit umschaltbarer Schieberichtung verwendet. Abbildung 6.42 zeigt eine Prinzipschaltung für ein solches Register.

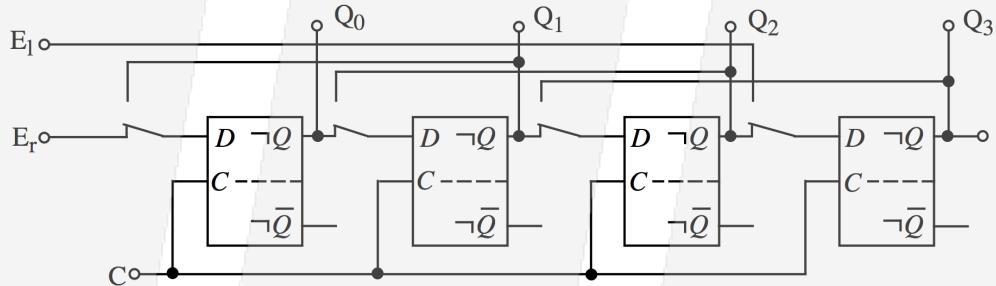


Abbildung 6.42: Schieberegister mit umschaltbarer Schieberichtung

Die in Abbildung 6.42 angezeigte Schalterstellung gilt dabei für Rechtsschieben *E_r*. Die hier eingezeichneten Schalter werden in der Praxis durch eine entsprechende Logik ersetzt.

6.4 Programmierbare Logik

Halbleiterspeicher werden eingeteilt in Nur-Lese-Speicher (**ROM** = Read Only Memory) und in Schreib-Lese-Speicher (**RAM** = Random Access Memory). Beide Speicher erlauben einen wahlfreien Zugriff, allerdings enthält ein ROM im Gegensatz zum RAM einen festen bzw. nicht ohne weiteres veränderlichen Speicherinhalt, der nur noch gelesen werden kann. Der Nur-Lese-Speicher (Festwertspeicher) wird noch weiter unterteilt in:

- ROM, dies sind beim Herstellungsprozess programmierte Nur-Lese-Speicher.
- PROM (Programmable ROM), sind durch den Anwender selbst zu programmierende Festwertspeicher.
- REPROM (Reprogrammable ROM), sind ROMs, die nach einer Löschung erneut programmiert werden können. Sie werden weiter unterteilt in:
 - EPROM (Erasable PROM), bei denen die Information mit Hilfe von UV-Licht lösbar ist, sowie
 - EEPROM (Electrically erasable ROM), bei denen der Speicherinhalt elektrisch gelöscht werden kann.

Die Programmierung der Speicherzellen beruht darauf, die Verbindung zwischen der Adress- und Leseleitung aufzubauen oder zu zerstören (siehe auch 11.2.5). ROMs werden z.B. bei der Code-Konvertierung (etwa Binär- in Gray-Codierung), der Zeichengenerierung oder der Verschlüsselung von Daten verwendet.

6.4.1 PROM

Am Beispiel der PROMs soll die Darstellung integrierter Hardware-Normalformen erläutert werden.

Tabelle 6.10: Codeumsetzung Dualcode ($e_0 \dots e_3$) → Graycode ($f_0 \dots f_3$)

e_3	e_2	e_1	e_0	f_3	f_2	f_1	f_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Der Chip, ein so genanntes AND/OR-Array enthält 16 festverdrahtete Konjunktionen (Minterme) und 4 programmierbare 16-stellige Disjunktionen. Im Beispiel wird die Codeumsetzung Dual-Code → Gray-Code realisiert (Abb. 6.43). PROM-Realisierungen verlangen stets die Konstruktion aller Minterme, ob sie verwendet werden oder nicht. Die Realisierung einer Funktion mit n Variablen benötigt also ein PROM mit 2^n Speicherworten.

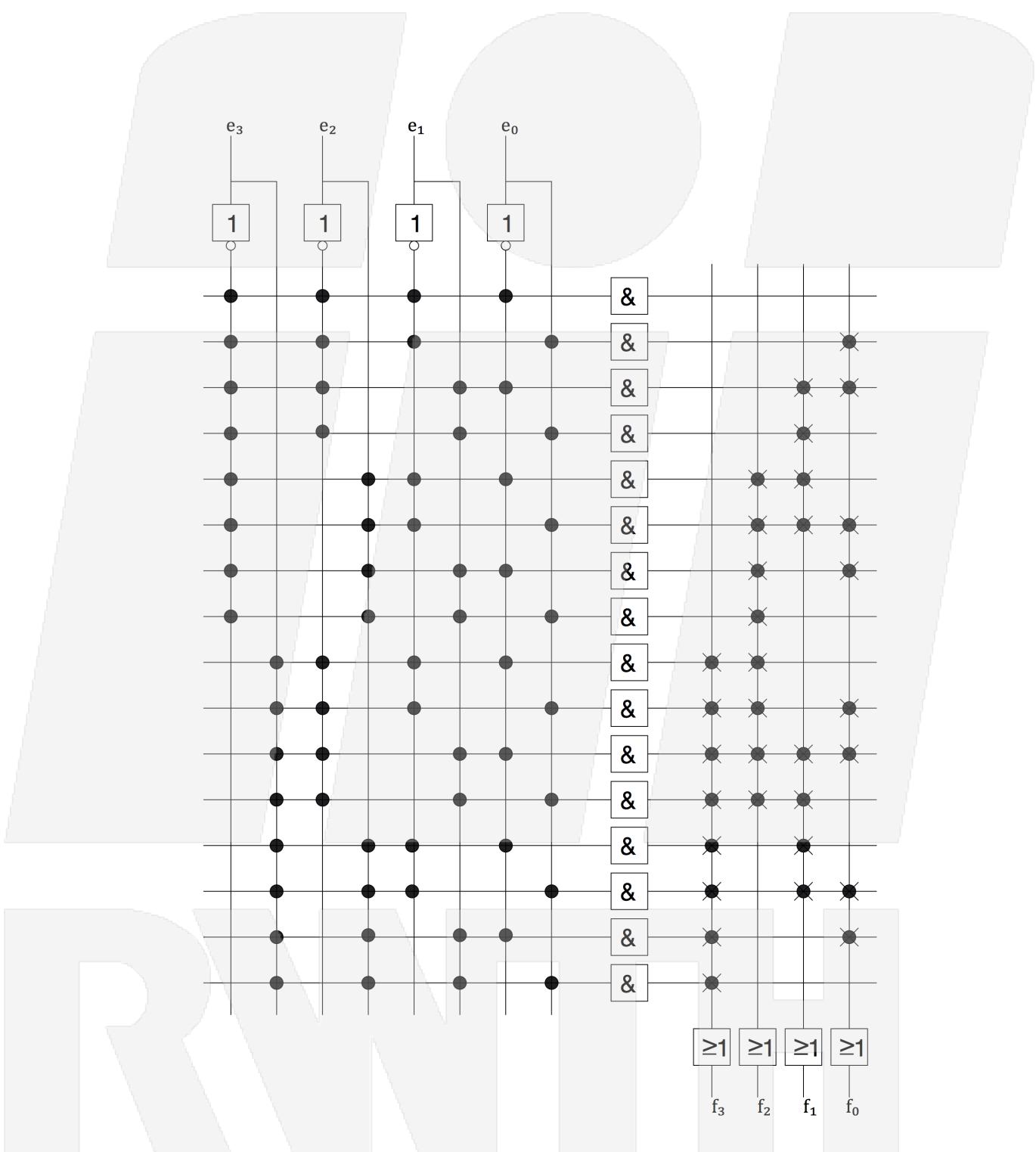


Abbildung 6.43: PROM zur Darstellung der tabellarisch aufgelisteten Funktion $f_{0 \dots 3}(e_0 \dots e_3)$

6.4.2 PAL-Bausteine (Programmable Array Logic)

Eine Funktion wird als Summe von Produkten (Konjunktionen) dargestellt (SOP), d. h. das AND-Feld wird programmiert (Abb. 6.44).

PAL-Baustein PAL 14 L4

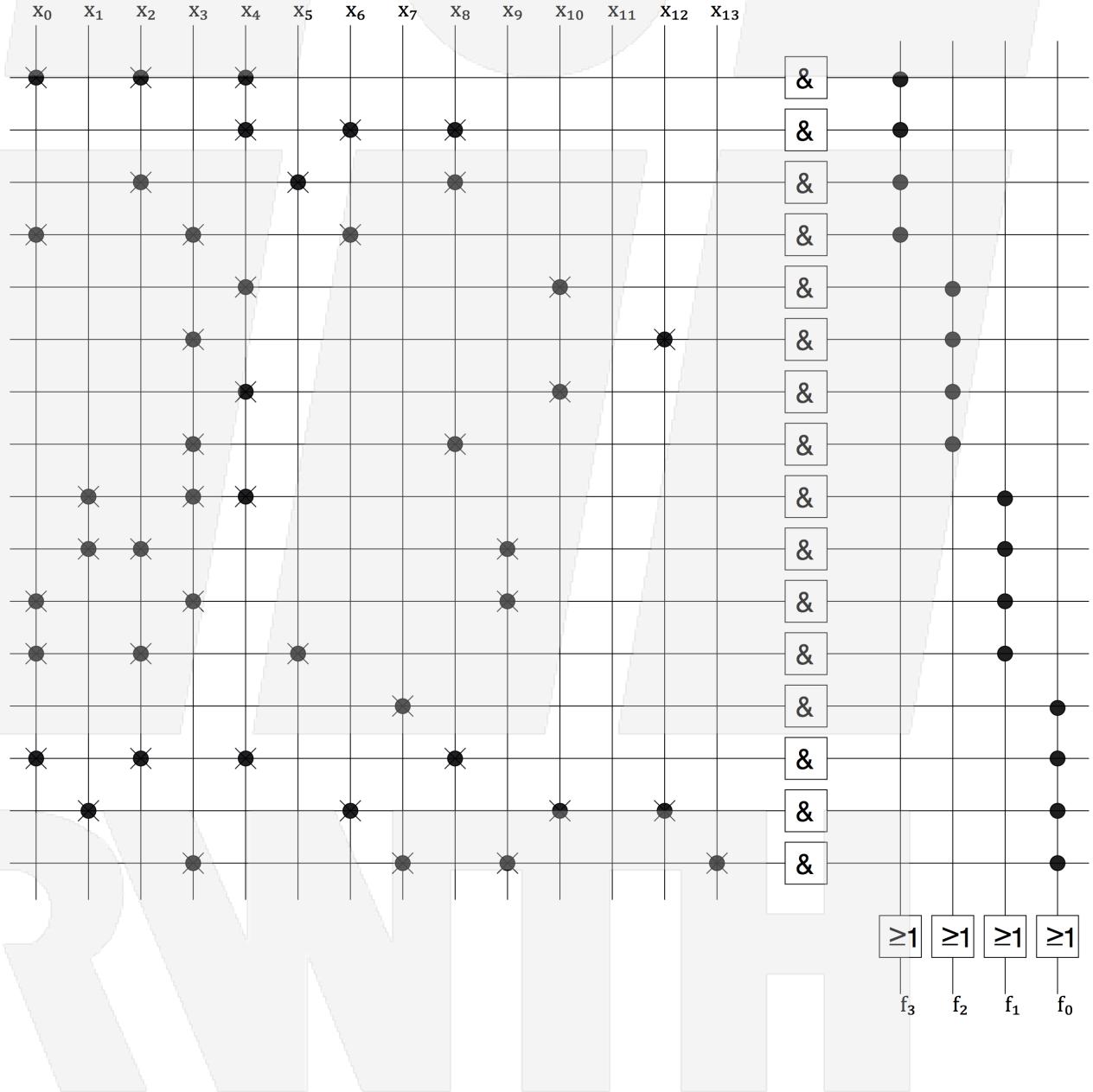


Abbildung 6.44: PAL 14 L4: Jeder Ausgang f_i ist die festverdrahtete Disjunktion von 4 programmierbaren Konjunktionen der 14 Eingänge x_0, \dots, x_{13}

6.4.3 (Field) Programmable Logic Array (FPLA)

Hier sind Konjunktionen und Disjunktionen programmierbar. Dies ergibt größere Flexibilität. Falls eine Rückführung des Ausgangs vorhanden ist, können sequentielle Funktionen erzeugt werden (PLS: Programmable Logic Based Sequencer (Abb. 6.45)). Die Ausgänge der ODER-Matrix werden in einem Register zwischengespeichert.

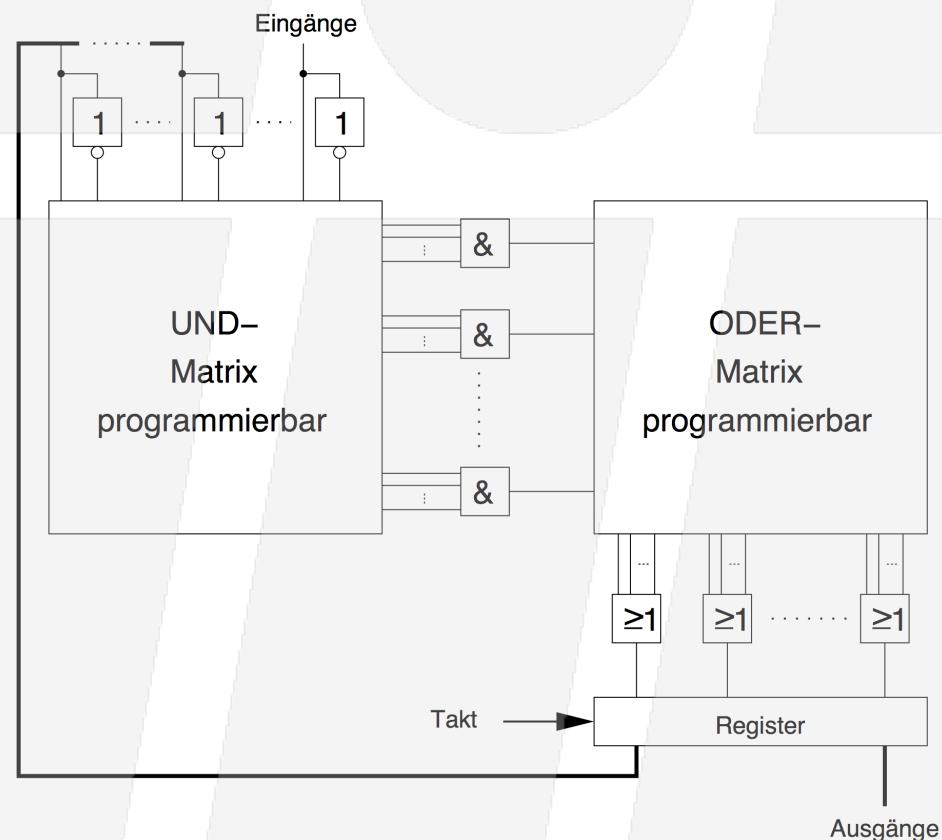


Abbildung 6.45: Struktur eines Programmable Logic Based Sequencers

6.4.4 Makrozellen (Programmierbare IC-Bausteine):

Makrozellen bieten Erweiterung der AND/OR-Felder um Register und E/A-Blöcke (Abb. 6.46).

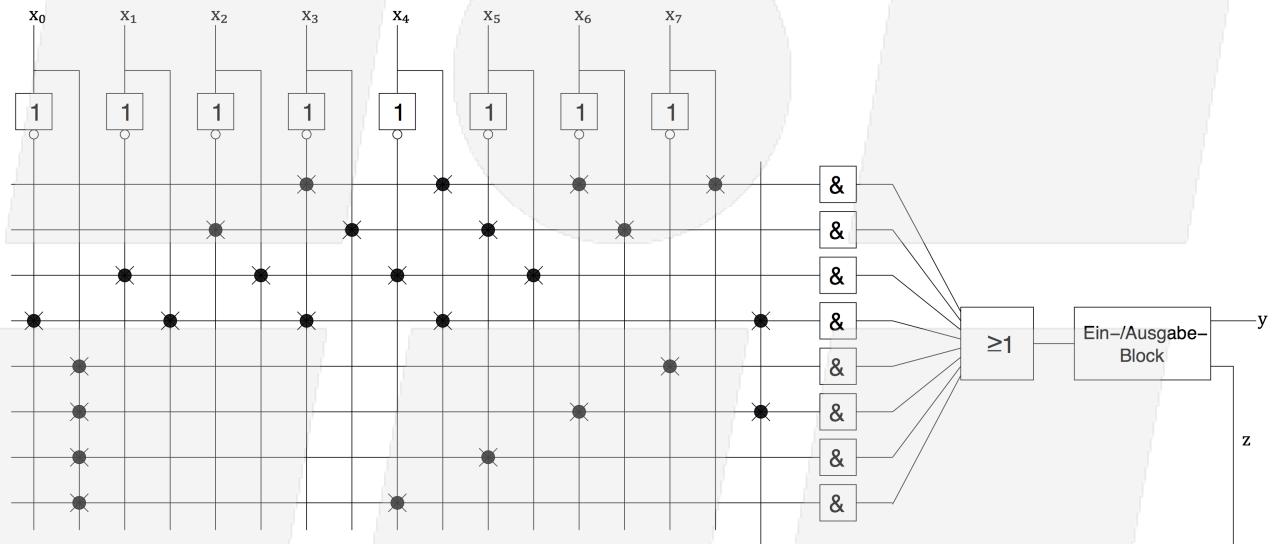


Abbildung 6.46: Makrozelle mit 8 intern programmierbaren Produkttermen, der Ausgang kann rückgekoppelt werden. Jede Makrozelle führt in einen Ein-/Ausgabeblock.

Makrozellen mit bis zu 9000 Gatteräquivalenzen² sind z. Z. verfügbar. Intern vorgegebene Strukturen schränken jedoch die Verbindungsmöglichkeiten ein.

²Mit Gatteräquivalenz ist ein NAND-Gatter mit 2 Eingängen gemeint.

Kapitel 7

Automaten

7.1 Einführung

Die Automatentheorie befasst sich mit der mathematischen Beschreibung von Systemen. Dazu gehören sowohl technische als auch biologische oder soziale Systeme, deren funktionelles Verhalten durch ein einheitliches abstraktes Modell beschrieben wird. In der Digitaltechnik werden Schaltnetze und Schaltwerke von der Automatentheorie abgedeckt. Aber auch im täglichen Leben sind Automaten häufig anzutreffen, wie folgendes Beispiel zeigt.

In einen Getränkeautomaten wird zunächst ein Geldstück eingeworfen. Durch Drücken einer entsprechenden Taste kann der Bediener eines von zwei möglichen Getränken wählen oder den eingeworfenen Geldbetrag zurückfordern. Ist der Geldbetrag nicht ausreichend, ertönt ein Signalton.

Mit Hilfe dieses Beispiels lassen sich charakteristische Eigenschaften von Automaten finden. Zunächst gibt es für jeden Automaten offensichtlich eine Menge möglicher Eingaben X (Geld-einwurf, Tastendruck) und eine Menge möglicher Ausgaben Y (Getränk, Signalton). Allerdings ist die Ausgabe nicht nur von der Eingabe abhängig, sondern auch von dem inneren Zustand Z des Automaten (eingeworfener Geldbetrag), der einer Menge möglicher Zustände Z angehört. Dabei spiegelt der aktuelle innere Zustand die vorangegangenen Eingaben wider. Auf eine Eingabe hin kann ein neuer innerer Zustand angenommen werden (Zustandswechsel) und die Ausgabe kann sich ändern. Für eine Beschreibung eines Automaten muss daher noch definiert sein, nach welchen Regeln diese Wechsel erfolgen. Dazu dienen zwei Funktionsbündel. Die Übergangsfunktion f beschreibt die Zustandswechsel, die Ausgangsfunktion g die Ausgabewechsel. Ein Automat A ist somit mit fünf Angaben, dem so genannten Quintupel des Automaten, vollständig beschrieben.

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

Im Folgenden wird ausschließlich von getakteten Automaten die Rede sein. Dies bedeutet, dass der Automat zu diskreten Zeitpunkten $t, t+1, t+2, \dots$ die anliegende Eingabe übernimmt und Zustands- sowie Ausgabewechsel durchführt, bevor sich eine neue Eingabe auswirkt. Bei getakteten Automaten spielen Signallaufzeiten keine Rolle und die einzelnen Zustände müssen nicht stabil sein.

7.2 Das Quintupel des Automaten

Eine präzisere Beschreibung der Grundelemente eines Automaten erfolgt mit Hilfe des in Abbildung 7.1 dargestellten Modellautomaten. Zur Eingabe besitzt er drei Eingabetaster X_0 bis X_2 . Intern kann der Automat drei verschiedene Zustände Z_0 bis Z_2 annehmen. Die Ausgabe besteht aus einer zweistelligen Dezimalanzeige, deren erste Ziffer den Zustand vor und deren zweite Ziffer den Zustand nach Betätigung eines Eingabeschalters anzeigt, also insgesamt den Zustandswechsel. Bei der Eingabe X_0 soll der Automat seinen Zustand nicht ändern. Bei X_1 sollen die Zustände Z zyklisch vorwärts und bei X_2 zyklisch rückwärts durchlaufen werden.

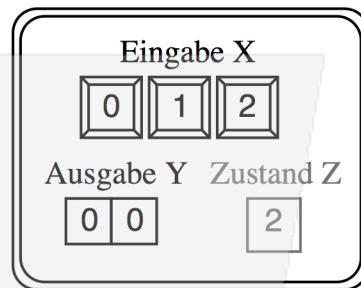


Abbildung 7.1: Modellautomat

In Tabelle 7.1 ist abgebildet, wie der Modellautomat auf eine Eingabe X^{n+1} reagiert. In der Tabelle sind den Eingabemöglichkeiten die Zustände Z^n (vor Betätigung der Eingabetaste) und Z^{n+1} (nach Betätigung der Eingabetaste) aufgeführt, sowie die Ausgabe Y^{n+1} , die der Automat dabei generiert.

Tabelle 7.1: Verhalten des Modellautomaten auf Eingaben

X^{n+1}	Z^n	Z^{n+1}	Y^{n+1}
0	0	0	00
1	0	1	01
1	1	2	12
1	2	0	20
2	0	2	02
2	2	1	21
2	1	0	10
0	0	0	00
1	0	1	01
0	1	1	11
1	1	2	12
0	2	2	22

Die Eingangsmenge X

Die *Eingangsmenge X* ist die Menge aller vorkommenden Eingaben, mit denen das Verhalten des Automaten beeinflusst wird. Sie setzt sich aus den Eingabeelementen zusammen. Für den Modellautomaten lautet die Eingangsmenge:

$$X = \{X_0, X_1, X_2\}$$

X_0, X_1 und X_2 sind nur die Kurzbezeichnungen für die drei Eingangselemente. Schaltungstechnisch werden Sie durch binäre Eingangsvariablen x_0 bis x_n codiert.

Die Ausgangsmenge Y

Die *Ausgangsmenge Y* ist die Menge aller vorkommenden Ausgaben. Sie setzt sich aus den Ausgabeelementen zusammen. Für den Modellautomaten lautet die Ausgabemenge:

$$Y = \{Y_{00}, Y_{01}, Y_{02}, Y_{10}, Y_{11}, Y_{12}, Y_{20}, Y_{21}, Y_{22}\}$$

Die Zustandsmenge Z

Die *Zustandsmenge* ist die Menge der inneren *Zustände*, die der Automat annehmen kann. Bei dem Modellautomaten besteht sie aus 3 Elementen:

$$Z = \{Z_0, Z_1, Z_2\}$$

Die Übergangsfunktion f

Die *Übergangsfunktion f* ist ein Bündel von Funktionen, das für jede Kombination von aktuellem Zustand und Eingabe den Folgezustand definiert.

$$Z^{n+1} = f(X^{n+1}, Z^n)$$

Die inneren Zustände vor der Eingabe liegen im Zeitabschnitt n , die neuen im Zeitabschnitt $n+1$. Daher kennzeichnet man die alten Zustände mit Z_i^n , die neuen mit Z_i^{n+1} . Analog heißen die Eingangselemente, mit denen der neue Zeitabschnitt beginnt X_i^{n+1} .

Für den Modellautomaten gilt folgende Übergangsfunktion:

$$\begin{aligned} Z_0^{n+1} &= \{X_0^{n+1} \cdot Z_0^n + X_1^{n+1} \cdot Z_2^n + X_2^{n+1} \cdot Z_1^n\} \\ Z_1^{n+1} &= \{X_0^{n+1} \cdot Z_1^n + X_1^{n+1} \cdot Z_0^n + X_2^{n+1} \cdot Z_2^n\} \\ Z_2^{n+1} &= \{X_0^{n+1} \cdot Z_2^n + X_1^{n+1} \cdot Z_1^n + X_2^{n+1} \cdot Z_0^n\} \end{aligned}$$

Die oberste Gleichung bedeutet, dass der Folgezustand Z_0 genau in drei Fällen angenommen wird: Wenn in Zustand Z_0 die Eingabe X_0 erfolgt, wenn in Zustand Z_1 die Eingabe X_2 erfolgt und wenn in Zustand Z_2 die Eingabe X_1 erfolgt.

Die Übergangsfunktion f lässt sich auch als Abbildung schreiben:

$$f : X^{n+1} \times Z^n \rightarrow Z^{n+1}$$

Diese Schreibweise bedeutet, dass für jede Kombination X^{n+1} und Z^n die Funktion f einen Wert Z^{n+1} bestimmt. Das Kreuzprodukt $X^{n+1} \times Z^n$ besteht aus allen Paaren (X^{n+1}, Z^n) mit $X^{n+1} \in X$ und $Z^n \in Z$.

Die Ausgangsfunktion g

Die *Ausgangsfunktion g* ist ein Funktionsbündel, das für jede Kombination aus altem Zustand und Eingabe die neue Ausgabe definiert. Für den Modellautomaten gilt:

$$\begin{array}{lll} Y_{00}^{n+1} = X_0^{n+1} \cdot Z_0^n & Y_{01}^{n+1} = X_1^{n+1} \cdot Z_0^n & Y_{02}^{n+1} = X_2^{n+1} \cdot Z_0^n \\ Y_{10}^{n+1} = X_2^{n+1} \cdot Z_1^n & Y_{11}^{n+1} = X_0^{n+1} \cdot Z_1^n & Y_{12}^{n+1} = X_1^{n+1} \cdot Z_1^n \\ Y_{20}^{n+1} = X_1^{n+1} \cdot Z_2^n & Y_{21}^{n+1} = X_2^{n+1} \cdot Z_2^n & Y_{22}^{n+1} = X_0^{n+1} \cdot Z_2^n \end{array}$$

Analog zur Ausgangsfunktion schreibt man:

$$Y^{n+1} = g(X^{n+1}, Z^n)$$

oder

$$g : X^{n+1} \times Z^n \rightarrow Y^{n+1}$$

7.3 Darstellungsweisen von Automaten

Abgesehen von dem Quintupel des Automaten gibt es noch weitere Möglichkeiten, das Verhalten eines Automaten zu beschreiben. Die verschiedenen Darstellungsweisen besitzen jeweils spezifische Eigenschaften bezüglich ihrer Verwendbarkeit für Analyse und Synthese.

7.3.1 Automatographen

Graphen sind eine sehr anschauliche Art Automaten zu beschreiben. Ein Graph besteht aus Knoten und gerichteten Kanten. Die Kanten entspringen und enden in genau einem Knoten. Jedem Knoten ist ein innerer Zustand zugeordnet, jeder Kante ein Zustandsübergang. In Abbildung 7.2 ist der Modellautomat aus Abschnitt 7.2 als Graph dargestellt. Die Kanten sind mit den Eingangselementen (X_i), die den Zustandsübergang bewirken und mit dem im neuen Zustand angezeigten Ausgabeelement (Y_{jk}) bezeichnet.

7.3.2 Automatentabellen

Die Übergangs- und Ausgangsfunktion lassen sich auch in Form einer Tabelle schreiben. Dabei ordnet man den alten inneren Zuständen die Zeilen und den Eingangselementen die Spalten zu. So entsteht für jede mögliche Kombination aus altem Zustand und Eingabe ein Kreuzungspunkt, in den man für die *Übergangstabelle* den neuen Zustand und für die *Ausgangstabelle* die neue Ausgabe einträgt. In der Automatentabelle, die den kompletten Automaten beschreibt, sind Übergangs- und Ausgangstabelle kombiniert. In Tabelle 7.2 sind die drei Tabellen für den Modellautomaten abgebildet.

7.3.3 Automatenband

Das *Automatenband* (Tabelle 7.3) stellt das Verhalten eines Automaten auf eine beliebige Folge von Eingaben dar. Für eine komplett Beschreibung müssen alle möglichen Zustandsübergänge mindestens einmal vorkommen, was mit steigender Komplexität des Automaten schnell unpraktikabel wird. Das Hauptanwendungsbereich dieser Darstellungsweise liegt in der Analyse.

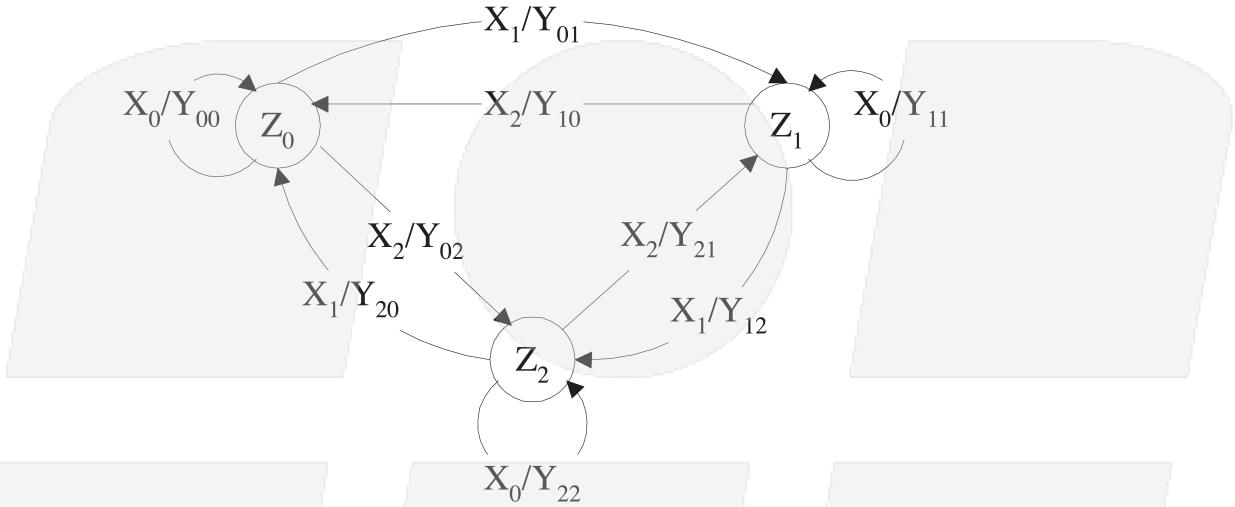


Abbildung 7.2: Der Modellautomat als Graph

Tabelle 7.2: Übergangs-, Ausgangs- und Automatentabelle für den Modellautomaten

n	n + 1		
	X ₀	X ₁	X ₂
Z ₀	Z ₀	Z ₁	Z ₂
Z ₁	Z ₁	Z ₂	Z ₀
Z ₂	Z ₂	Z ₀	Z ₁

n	n + 1		
	X ₀	X ₁	X ₂
Z ₀	Y ₀₀	Y ₀₁	Y ₀₂
Z ₁	Y ₁₁	Y ₁₂	Y ₁₀
Z ₂	Y ₂₂	Y ₂₀	Y ₂₁

n	n + 1		
	X ₀	X ₁	X ₂
Z ₀	Z ₀ /Y ₀₀	Z ₁ /Y ₀₁	Z ₂ /Y ₀₂
Z ₁	Z ₁ /Y ₁₁	Z ₂ /Y ₁₂	Z ₀ /Y ₁₀
Z ₂	Z ₂ /Y ₂₂	Z ₀ /Y ₂₀	Z ₁ /Y ₂₁

7.4 Automatentypen

Verschiedene Klassen von Automaten ergeben sich, wenn man für die Mengen X, Y und Z oder das Funktionsbündel g gewisse Voraussetzungen macht. Einen Überblick über diese Einteilung gibt Tabelle 7.4.

Mealy-Automat

Der *Mealy-Automat* ist der allgemeinste Automat. Das Automatenquintupel unterliegt keinen Einschränkungen. Bei ihm hängen sowohl Ausgabe als auch Folgezustand von der Eingabe und dem aktuellen Zustand ab. Abbildung 7.2 stellt ein Beispiel für den Graphen eines Mealy-Automaten dar, während Abbildung 7.3 das Blockschaltbild eines Mealy-Automaten zeigt.

Tabelle 7.3: Automatenband des Modellautomaten

...	X ₀	X ₁	X ₁	X ₁	X ₂	X ₂	X ₂	X ₀	X ₁	X ₀	X ₁	X ₀
Z ₀	Z ₀	Z ₁	Z ₂	Z ₀	Z ₂	Z ₁	Z ₀	Z ₀	Z ₁	Z ₁	Z ₂	Z ₂
...	Y ₀₀	Y ₀₁	Y ₁₂	Y ₂₀	Y ₀₂	Y ₂₁	Y ₁₀	Y ₀₀	Y ₀₁	Y ₁₁	Y ₁₂	Y ₂₂

Tabelle 7.4: Automatentypen

Voraussetzung	Typ
keine	Mealy-Automat; Übergangsautomat
$g : Z^n \rightarrow Y^n$	Moore-Automat; Zustandsautomat
$Y = \{Y_1\}$	Halbautomat
$X = \{X_1\}$	Autonomer Automat
$Z = \{Z_1\}$	Zuordner; kombinatorische Schaltung

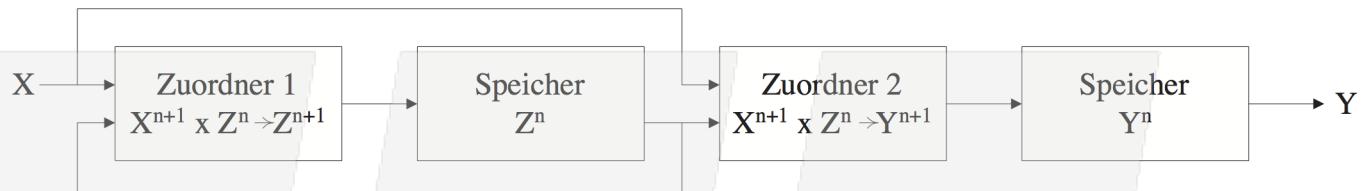


Abbildung 7.3: Blockschaltbild eines Mealy-Automaten

Moore-Automat

Ein Automat heißt *Moore-Automat*, wenn seine Ausgangsfunktion g nur von dem Zustand Z und nicht von X abhängt. Abbildung 7.4 zeigt das zugehörige Blockschaltbild.



Abbildung 7.4: Blockschaltbild eines Moore-Automaten

Der Modellautomat wird beispielsweise zu einem Moore-Automaten, wenn nur noch der aktuelle Zustand angezeigt wird und nicht mehr der Zustandswechsel. Dann reduziert sich die Ausgabemenge auf die drei Elemente Y_0 , Y_1 und Y_2 , die jeweils einem Zustand fest zugeordnet sind. Bei einem als Graph dargestellten Moore-Automaten kann man dann die Ausgabeelemente innerhalb der Knoten bezeichnen. Ein Beispiel dafür zeigt Abbildung 7.5.

Auch die Automatentafel vereinfacht sich, wie in Tabelle 7.5 dargestellt. Da jedem Zustand ein Ausgabeelement fest zugeordnet ist, muss es nur noch für jede Zeile einmal angegeben werden.

Zuordner

Bei einem *Zuordner* (Abbildung 7.6) besteht die Zustandsmenge nur aus einem Element $\{Z = Z_0\}$. Damit kann das Verhalten des Automaten nur von der aktuellen Eingabe und nicht mehr von dem alten Zustand abhängen. Der Zuordner bildet also nur eine Eingabe auf eine Ausgabe ab.

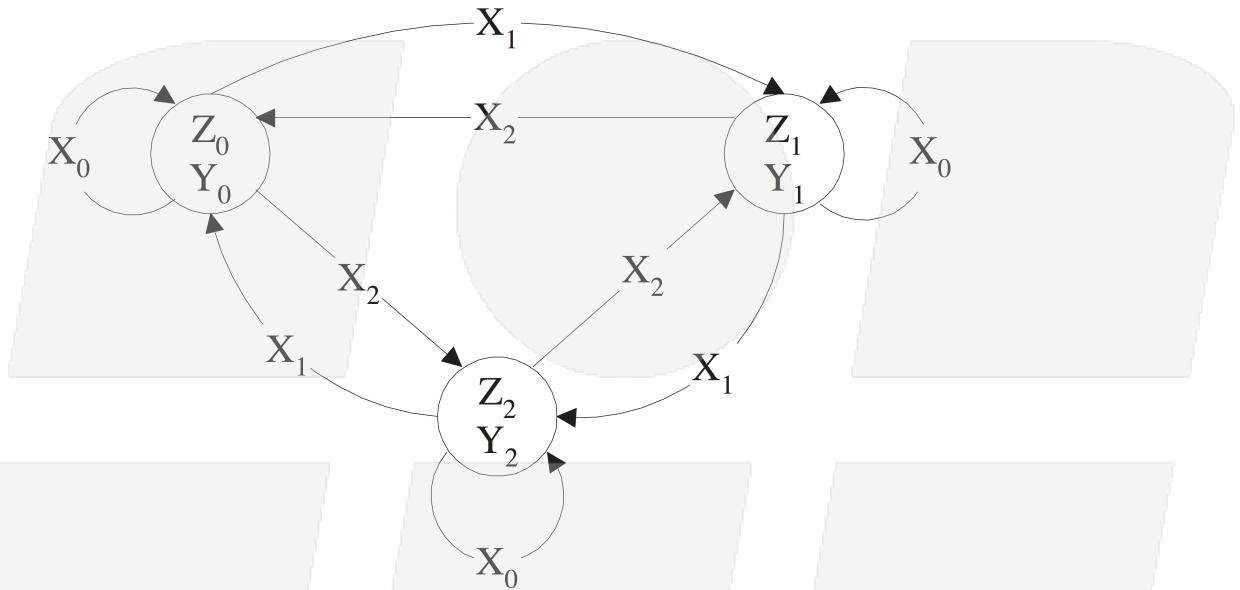


Abbildung 7.5: Graph eines Moore-Automaten

Tabelle 7.5: Automatentafel eines Moore-Automaten

n	n + 1			n
	X0	X1	X2	
Z0	Z0	Z1	Z2	Y0
Z1	Z1	Z2	Z0	Y1
Z2	Z2	Z0	Z1	Y2

Autonomer Automat

Von einem autonomen Automaten (Abbildung 7.7) spricht man, wenn nur ein Eingabeelement existiert $\{X = X_0\}$ oder anders formuliert, wenn Übergangs- und Ausgangsfunktion von X unabhängig sind. Dann ist das Verhalten des Automaten von außen nicht beeinflussbar und es wird ständig eine bestimmte Zustandsfolge durchlaufen. Einfache Zähler sind autonome Automaten.

Halb-Automat

Besteht die Ausgangsmenge Y eines Automaten nur aus einem Element oder ist keine Ausgangsfunktion definiert, spricht man von einem Halb- oder Medwedjew-Automaten. Dieser Typ hat zwar praktisch keine Bedeutung, eine Reihe theoretisch interessanter Fragestellungen, wie

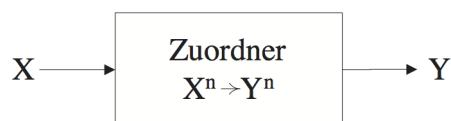


Abbildung 7.6: Blockschaltbild eines Zuordners



Abbildung 7.7: Blockschaltbild eines autonomen Automaten

die Minimierung der Anzahl innerer Zustände, lassen sich aber an ihm einfach behandeln. Abbildung 7.8 zeigt das zugehörige Blockschaltbild eines Halb-Automaten.

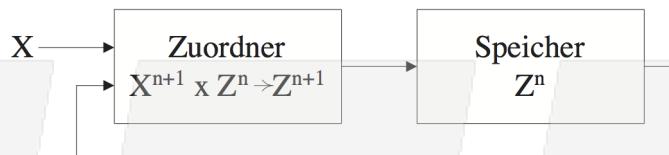


Abbildung 7.8: Blockschaltbild eines Halb-Automaten

Akzeptor

Ein weiterer wichtiger Automatentyp ist der *Akzeptor*. Dies ist ein Automat, für den eine Menge von Zuständen als akzeptierend definiert ist. Diese Eigenschaft ist nicht über eine Einschränkung des Automaten-Quintupels definierbar. Ein Akzeptor kann zwischen akzeptierten und zurückgewiesenen Eingabefolgen unterscheiden. Ein Beispiel für einen Akzeptor ist ein Automat, welcher als Eingabe Ziffern und Buchstaben erhalten kann und prüft, ob eine eingegebene Zeichenfolge eine Zahl darstellt.

7.5 Umwandlung zwischen Moore- und Mealy-Automat

Betrachtet man einen Automaten von außen, ist nicht unterscheidbar, ob es sich um einen Moore- oder einen Mealy-Automaten handelt. Man kann beweisen, dass jeder Mealy- in einen Moore-Automaten überführbar ist und umgekehrt.

Die Umwandlung von einem Moore- in einen Mealy-Automaten ist sehr einfach, da die Anzahl der Zustände und somit auch die Übergangsfunktion unverändert bleiben kann. Die Ausgangsfunktion ist direkt ablesbar. Abbildung 7.9 zeigt den Moore-Automaten aus Abbildung 7.5 als Mealy-Automaten, während in Tabelle 7.6 die entsprechenden Automatentabellen dargestellt sind. Zur Konstruktion wird jede Kante mit dem Ausgabeelement bezeichnet, das in dem Knoten steht, auf den sie zeigt. Anschließend werden die Ausgabeelemente aus den Knoten entfernt.

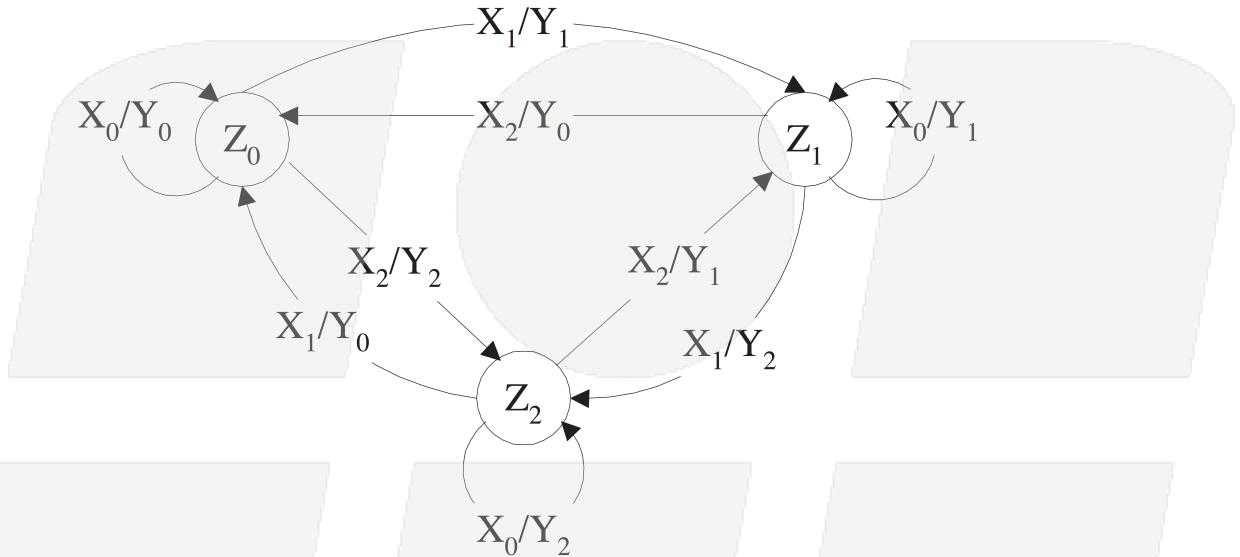


Abbildung 7.9: In Mealy-Automat umgewandelter Moore-Automat

Tabelle 7.6: Umwandlung eines Moore-Automaten (links, s. auch Tabelle 7.5) in einen äquivalenten Mealy-Automaten (rechts)

Moore				Mealy			
n	n + 1			n	n + 1		
	X ₀	X ₁	X ₂		Z ₀ /Y ₀	Z ₁ /Y ₁	Z ₂ /Y ₂
Z ₀	Z ₀	Z ₁	Z ₂	Y ₀			
Z ₁	Z ₁	Z ₂	Z ₀	Y ₁			
Z ₂	Z ₂	Z ₀	Z ₁	Y ₂			

Bei der Umwandlung von einem Mealy- in einen Moore-Automaten kann die Zustandsmenge nicht beibehalten werden, denn für den Mealy-Automaten gilt:

$$X^{n+1} \times Z_{Mealy}^n \rightarrow Y^{n+1}$$

Das heißt, jedem Zustandsübergang kann ein Ausgabeelement zugeordnet werden. Für den Moore-Automaten gilt dagegen:

$$Z_{Moore}^{n+1} \rightarrow Y^{n+1}$$

Also existiert nur ein Ausgabeelement pro Zustand. Da die Ausgabeelemente für jede Eingabefolge aber identisch sein sollen, muss gelten:

$$X^{n+1} \times Z_{Mealy}^n = Z_{Moore}^{n+1}$$

Das bedeutet, dass jedem Kreuzprodukt aus neuem Eingangselementen und altem Zustand des Mealy-Automaten ein Zustand des äquivalenten Moore-Automaten entspricht.

Als Beispiel soll der in Tabelle 7.7 links dargestellte Mealy-Automat in einen äquivalenten Moore-Automaten umgewandelt werden. Dazu wird jeder Kombination aus Mealy-Zustand und Eingabe ein Moore-Zustand zugeordnet. So wird beispielsweise Z₀/X₀ zum neuen Zustand Z₀₀.

Tabelle 7.7: Umwandlung eines Mealy-Automaten (links) in einen äquivalenten Moore-Automaten (rechts)

Mealy				
n	n + 1			
	X ₀	X ₁	X ₀	X ₁
Z ₀	Z ₁ /Y ₂	Z ₂ /Y ₁	Z ₁₀	Z ₁₁
Z ₁	Z ₀ /Y ₃	Z ₀ /Y ₄	Z ₂₀	Z ₂₁
Z ₂	Z ₂ /Y ₀	Z ₀ /Y ₅	Z ₀₀	Z ₀₁

n	n + 1		n
	X ₀	X ₁	
Z ₀₀ = (Z ₀ × X ₀)	Z ₁₀	Z ₁₁	Y ₂
Z ₀₁ = (Z ₀ × X ₁)	Z ₂₀	Z ₂₁	Y ₁
Z ₁₀ = (Z ₁ × X ₀)	Z ₀₀	Z ₀₁	Y ₃
Z ₁₁ = (Z ₁ × X ₁)	Z ₀₀	Z ₀₁	Y ₄
Z ₂₀ = (Z ₂ × X ₀)	Z ₂₀	Z ₂₁	Y ₀
Z ₂₁ = (Z ₂ × X ₁)	Z ₀₀	Z ₀₁	Y ₅

Mit zwei Eingabeelementen und drei Zuständen des Mealy-Automaten ergeben sich $2 \cdot 3 = 6$ Moore-Zustände. Die neuen Zustände werden in die erste Spalte der zu konstruierenden Moore-Automatentabelle eingetragen. Der Folgezustand Z_{10} in der ersten Zeile und ersten Spalte ergibt sich, indem in der Mealy-Tabelle zunächst der Folgezustand zu $Z_0 \times X_0$ gesucht wird. Es ist Z_1 . Nun wird in der Moore-Tabelle die Kombination $Z_1 \times X_0$ gesucht. Sie ist mit Z_{10} bezeichnet. Alle weiteren Folgezustände ergeben sich entsprechend. Schließlich wird jedem Moore-Zustand das Ausgangselement zugewiesen, das beim Mealy-Automaten dem zugehörigen Kreuzprodukt zugeordnet ist.

7.6 Äquivalenz und Zustandsreduktion

Zwei Automaten, die auf eine beliebige Eingabefolge identisch reagieren, sind für einen außen stehenden Beobachter nicht unterscheidbar. Trotzdem kann der Komplexitätsgrad ihres internen Aufbaus sehr unterschiedlich sein. Wenn man einen Automaten implementieren möchte, ist es von großer praktischer Bedeutung zu einem gegebenen Automaten A einen äquivalenten Automaten A' zu bestimmen, der mit einer minimalen Anzahl an Zuständen realisierbar ist.

7.6.1 Äquivalente Zustände

Zunächst folgen einige in diesem Zusammenhang benötigte Begriffsdefinitionen:

Definition 7.1 Zwei Zustände Z_i und Z_j eines Automaten sind äquivalent, wenn der Automat auf eine beliebige Eingangsfolge stets mit derselben Ausgangsfolge reagiert, gleichgültig ob im Zustand Z_i oder Z_j begonnen wird.

Definition 7.2 Zwei Zustände Z_i und Z_j eines Automaten sind k -äquivalent, wenn der Automat auf eine beliebige Eingangsfolge der Länge k stets mit derselben Ausgangsfolge reagiert, gleichgültig ob im Zustand Z_i oder Z_j begonnen wird.

Definition 7.3 Ein Automat heißt *zustandsminimal* oder *reduziert*, wenn er keine äquivalenten Zustände besitzt.

Definition 7.4 Die zu einander k -äquivalenten Zustände eines Automaten werden zu k -Äquivalenzklassen K_i^k zusammengefasst. Dabei stellt i einen Zählindex dar, durch den verschiedene k -Äquivalenzklassen unterschieden werden können.

Soll zu einem Automaten A der zustandsminimierte Automat A' ermittelt werden, geschieht dies durch Zusammenfassen äquivalenter Zustände.

Als Beispiel soll eine Mausefalle dienen, die entweder gespannt oder nicht gespannt ist. Ein Stück Speck kann als Köder eingesetzt werden. Zu einem betrachteten Zeitpunkt kommt die Maus in Reichweite der Feder oder nicht. Abhängig davon wird die Maus gefangen.

Der Automat hat die vier Zustände $Z_{00}, Z_{01}, Z_{10}, Z_{11}$, wobei der erste Index angibt, ob Speck vorhanden ist und der zweite, ob die Mausefalle gespannt ist. Die Eingabe lautet Maus in Reichweite der Feder (X_1) oder nicht (X_0) und die Ausgabe Maus gefangen (Y_1) oder nicht (Y_0). Die Automatentabelle ist in Tabelle 7.8 angegeben.

Tabelle 7.8: Übergangs-, Ausgangs- und Automatentabelle für eine Mausefalle mit Speck

n	$n + 1$	
	X_0	X_1
Z_{00}	Z_{00}	Z_{00}
Z_{01}	Z_{01}	Z_{00}
Z_{10}	Z_{10}	Z_{00}
Z_{11}	Z_{11}	Z_{10}

n	$n + 1$	
	X_0	X_1
Z_{00}	Y_0	Y_0
Z_{01}	Y_0	Y_1
Z_{10}	Y_0	Y_0
Z_{11}	Y_0	Y_1

n	$n + 1$	
	X_0	X_1
Z_{00}	Z_{00}/Y_0	Z_{00}/Y_0
Z_{01}	Z_{01}/Y_0	Z_{00}/Y_1
Z_{10}	Z_{10}/Y_0	Z_{00}/Y_0
Z_{11}	Z_{11}/Y_0	Z_{10}/Y_1

Anhand der Ausgangstabelle erkennt man direkt, dass die Zustände Z_{00} und Z_{10} 1-äquivalent sind, da sie aufgrund einer einzelnen Eingabe für einen außen stehenden Betrachter nicht unterscheidbar sind. Daher werden sie in der 1-Äquivalenzklasse K_0^1 zusammengefasst. Genauso fasst man Z_{01} und Z_{11} zur 1-Äquivalenzklasse K_1^1 zusammen. Nun wird die Automatentabelle so umsortiert, dass die Zustände einer Äquivalenzklasse direkt untereinander stehen (Tabelle 7.9, links). Der Index der Äquivalenzklasse ist für jeden Folgezustand in der 1-Äquivalenzklassentabelle nach dem „–“-Zeichen angegeben. Auf diese Weise lassen sich leicht höherwertige Äquivalenzklassen erkennen, denn es gilt:

Definition 7.5 Eine k -Äquivalenzklasse enthält alle Zustände, die $(k-1)$ -äquivalent sind und deren Folgezustände für eine beliebige Eingabe $X_i \in X$ in denselben Äquivalenzklassen liegen.

Offensichtlich sind die gefundenen 1-Äquivalenzklassen ebenfalls 2-Äquivalent und mit vollständiger Induktion auch äquivalent. Damit sind die Endklassen gefunden, die direkt den Zuständen des reduzierten Automaten entsprechen. Der reduzierte Automat ist in Tabelle 7.9 abgebildet.

Das Ergebnis ist auch anschaulich einsichtig. Das Vorhandensein von Speck erhöht zwar die Wahrscheinlichkeit eine Maus zu fangen, beeinflusst aber nicht das Verhalten der Falle auf eine konkrete Eingabe. Entsprechend sind alle Zustände äquivalent, die sich nur in dem Attribut *Speck vorhanden bzw. kein Speck vorhanden* unterscheiden.

Als weiteres Beispiel soll der durch die Automatentabelle 7.10 beschriebene redundante Moore-Automat reduziert werden. Im Unterschied zum Mealy-Automaten gibt es hier auch

Tabelle 7.9: 1-Äquivalenzklassen der Mausefalle mit Speck und reduzierter Automat

	X_0	X_1	
Z_{00}	$Z_{00}/Y_0 - K_0^1$	$Z_{00}/Y_0 - K_0^1$	K_0^1
Z_{10}	$Z_{10}/Y_0 - K_0^1$	$Z_{00}/Y_0 - K_0^1$	
Z_{01}	$Z_{01}/Y_0 - K_1^1$	$Z_{00}/Y_1 - K_0^1$	K_1^1
Z_{11}	$Z_{11}/Y_0 - K_1^1$	$Z_{10}/Y_1 - K_0^1$	

	X_0	X_1
Z'_0	Z'_0/Y_0	Z'_0/Y_0
Z'_1	Z'_1/Y_0	Z'_0/Y_1

0-äquivalente Zustände. Das sind solche, denen die gleiche Ausgabe zugeordnet ist. Im Beispiel gibt es drei verschiedene Ausgaben, also auch drei 0-Äquivalenzklassen. Die Ermittlung der äquivalenten Zustände erfolgt im Prinzip wie beim Mealy-Automaten. Zunächst wird die Tabelle umsortiert, so dass die 0-äquivalenten Zustände untereinander stehen. Dann wird neben jedem Folgezustand der Index der 0-Äquivalenzklasse notiert, der er angehört (Tabelle 7.10, rechts). Nun lassen sich leicht die 1-Äquivalenzklassen finden, denn wiederum gilt, dass eine k -Äquivalenzklasse alle Zustände enthält, die $(k-1)$ -äquivalent sind und deren Folgezustände für eine beliebige Eingabe $X_i \in X$ in denselben Äquivalenzklassen liegen.

Also müssen die Äquivalenzklassen weiter unterteilt werden, deren Folgezustände nicht für jede Eingabe in der gleichen Äquivalenzklasse liegen. Dies ist nur für die 0-Äquivalenzklasse K_2^0 der Fall, die in drei Klassen aufgespalten wird. Durch Umsortieren und Eintrag des Indizes der 1-Äquivalenzklasse neben jedem Folgezustand entsteht Tabelle 7.11.

Tabelle 7.10: Automatentabelle und 0-Äquivalenzklassen eines redundanten Moore-Automaten

	X_0	X_1	X_2	
Z_1	Z_{10}	Z_1	Z_6	Y_2
Z_2	Z_{11}	Z_8	Z_6	Y_2
Z_3	Z_1	Z_7	Z_6	Y_1
Z_4	Z_9	Z_2	Z_3	Y_2
Z_5	Z_{10}	Z_1	Z_6	Y_1
Z_6	Z_8	Z_2	Z_{11}	Y_2
Z_7	Z_8	Z_2	Z_{11}	Y_2
Z_8	Z_6	Z_{11}	Z_5	Y_3
Z_9	Z_4	Z_3	Z_5	Y_3
Z_{10}	Z_3	Z_9	Z_7	Y_2
Z_{11}	Z_1	Z_6	Z_4	Y_1

	X_0	X_1	X_2		
Z_3	$Z_1 - K_2^0$	$Z_7 - K_2^0$	$Z_6 - K_2^0$	K_1^0	Y_1
Z_5	$Z_{10} - K_2^0$	$Z_1 - K_2^0$	$Z_6 - K_2^0$		
Z_{11}	$Z_1 - K_2^0$	$Z_6 - K_2^0$	$Z_4 - K_2^0$		
Z_1	$Z_{10} - K_2^0$	$Z_1 - K_2^0$	$Z_6 - K_2^0$	K_2^0	Y_2
Z_2	$Z_{11} - K_1^0$	$Z_8 - K_3^0$	$Z_6 - K_2^0$		
Z_4	$Z_9 - K_3^0$	$Z_2 - K_2^0$	$Z_3 - K_1^0$		
Z_6	$Z_8 - K_3^0$	$Z_2 - K_2^0$	$Z_{11} - K_1^0$	K_3^0	Y_3
Z_7	$Z_8 - K_3^0$	$Z_2 - K_2^0$	$Z_{11} - K_1^0$		
Z_{10}	$Z_3 - K_1^0$	$Z_9 - K_3^0$	$Z_7 - K_2^0$		
Z_8	$Z_6 - K_2^0$	$Z_{11} - K_1^0$	$Z_5 - K_1^0$		
Z_9	$Z_4 - K_2^0$	$Z_3 - K_1^0$	$Z_5 - K_1^0$		

Dieses Verfahren wird solange wiederholt, bis keine Äquivalenzklasse mehr unterteilt werden muss. Dabei ist zu beachten, dass für jeden Schritt alle Klassen zu prüfen sind, also auch solche, die im vorangegangenen Schritt nicht verändert wurden. Im Beispiel ist der Endzustand nach dem Ermitteln der 2-Äquivalenzklassen erreicht.

Der in Tabelle 7.12 rechts dargestellte reduzierte Automat ergibt sich wie zuvor aus den End-

Tabelle 7.11: 1-Äquivalenzklassen des redundanten Moore-Automaten

	X_0	X_1	X_2			
Z_3	$Z_1 - K_2^1$	$Z_7 - K_4^1$	$Z_6 - K_4^1$	K_1^1	Y_1	
Z_5	$Z_{10} - K_3^1$	$Z_1 - K_2^1$	$Z_6 - K_4^1$			
Z_{11}	$Z_1 - K_2^1$	$Z_6 - K_4^1$	$Z_4 - K_4^1$			
Z_1	$Z_{10} - K_3^1$	$Z_1 - K_2^1$	$Z_6 - K_4^1$	K_2^1	Y_2	
Z_2	$Z_{11} - K_1^1$	$Z_8 - K_5^1$	$Z_6 - K_4^1$	K_3^1		
Z_{10}	$Z_3 - K_1^1$	$Z_9 - K_5^1$	$Z_7 - K_4^1$			
Z_4	$Z_9 - K_5^1$	$Z_2 - K_3^1$	$Z_3 - K_1^1$	Y_3		
Z_6	$Z_8 - K_5^1$	$Z_2 - K_3^1$	$Z_{11} - K_1^1$		K_4^1	
Z_7	$Z_8 - K_5^1$	$Z_2 - K_3^1$	$Z_{11} - K_1^1$			
Z_8	$Z_6 - K_4^1$	$Z_{11} - K_1^1$	$Z_5 - K_1^1$	K_5^1	Y_3	
Z_9	$Z_4 - K_4^1$	$Z_3 - K_1^1$	$Z_5 - K_1^1$			

Tabelle 7.12: 2-Äquivalenzklassen des redundanten Moore-Automaten und zustandsreduzierter Automat

	X_0	X_1	X_2		
Z_3	$Z_1 - K_3^2$	$Z_7 - K_5^2$	$Z_6 - K_5^2$	K_1^2	Y_1
Z_{11}	$Z_1 - K_3^2$	$Z_6 - K_5^2$	$Z_4 - K_5^2$		
Z_5	$Z_{10} - K_4^2$	$Z_1 - K_3^2$	$Z_6 - K_5^2$		
Z_1	$Z_{10} - K_4^2$	$Z_1 - K_3^2$	$Z_6 - K_5^2$	K_3^2	Y_2
Z_2	$Z_{11} - K_1^2$	$Z_8 - K_6^2$	$Z_6 - K_5^2$		
Z_{10}	$Z_3 - K_1^2$	$Z_9 - K_6^2$	$Z_7 - K_5^2$		
Z_4	$Z_9 - K_6^2$	$Z_2 - K_4^2$	$Z_3 - K_1^2$	K_5^2	Y_3
Z_6	$Z_8 - K_6^2$	$Z_2 - K_4^2$	$Z_{11} - K_1^2$		
Z_7	$Z_8 - K_6^2$	$Z_2 - K_4^2$	$Z_{11} - K_1^2$		
Z_8	$Z_6 - K_5^2$	$Z_{11} - K_1^2$	$Z_5 - K_2^2$	K_6^2	Y_3
Z_9	$Z_4 - K_5^2$	$Z_3 - K_1^2$	$Z_5 - K_2^2$		

	X_0	X_1	X_2	
Z'_1	Z'_3	Z'_5	Z'_5	Y_1
Z'_2	Z'_4	Z'_3	Z'_5	Y_1
Z'_3	Z'_4	Z'_3	Z'_5	Y_2
Z'_4	Z'_1	Z'_6	Z'_5	Y_2
Z'_5	Z'_6	Z'_4	Z'_1	Y_2
Z'_6	Z'_5	Z'_1	Z'_2	Y_3

klassen.

7.7 Technische Realisierung von Automaten

Die grundsätzliche Vorgehensweise zur schaltungstechnischen Umsetzung eines Automaten ist wie folgt:

Ausgangspunkt für die technische Realisierung eines Automaten ist die Zustands- und die Ausgangstabelle bzw. deren Kombination, die Automatentabelle. Damit ist das Verhalten des Automaten vollständig beschrieben.

Um den Aufwand für die Schaltung möglichst gering zu halten, wird der Automat in einem ersten Schritt auf redundante Zustände getestet und falls erforderlich minimiert.

Als nächstes werden die *Codierungstabellen* festgelegt. Die Codierung der Elemente eines getakteten Automaten ist im Prinzip beliebig, da sie keinen Einfluss auf die Funktionalität hat. Sie bestimmt allerdings zum Teil den Realisierungsaufwand. Leider gibt es für eine günstige Codierung nur heuristische Verfahren (Daumenregeln), so dass hier die Erfahrung des Schaltungsdesigners eine wesentliche Rolle spielt.

Dann muss festgelegt werden, wie Übergangs- und Ausgangsfunktion in einer Schaltung umgesetzt werden sollen. Als Schaltungstypen kommen Festwertspeicher oder festverdrahtete Logikschaltungen in Frage. Von dieser Entscheidung ist auch die Wahl der Speicherelemente abhängig. Bei festverdrahteter Logik sind in der Regel JK-Flipflops zur Umsetzung am besten geeignet, da ihre Zustandsübergangs-Tabelle verglichen mit anderen Flipflops die meisten Redundanzen aufweist, was günstig für eine Minimierung der Ansteuerfunktionen ist. Für die Realisierung von Festwertspeichern eignen sich häufig D-Flipflops, da sie nur einen Informationseingang – und somit nur eine Ansteuerungsfunktion – besitzen.

Damit sind die Grundlagen für die Bestimmung der Schaltfunktion gelegt. Durch Umsortieren der Automatentabelle und Anwendung der Codierungstabellen erhält man eine Darstellung, aus der sich Ansteuerfunktionen und Übergangsfunktion einfacher ermitteln lässt.

Die Vorgehensweise ist nochmals in Tabelle 7.10 zusammengefasst. Im Folgenden soll beispielhaft der Modellautomat aus Abschnitt 7.2 realisiert werden.

7.7.1 Realisierung des Modellautomaten als Mealy-Automat

Übergangsteil

Zunächst soll die Übergangsfunktion des Modellautomaten in festverdrahteter Logik mit JK-Flipflops realisiert werden. Die Codierung von Eingangsvariablen und Zuständen erfolgt dual, entsprechend der dezimalen Indizes wie in Tabelle 7.13 links dargestellt. Um die drei inneren Zustände zu repräsentieren, werden zwei JK-Flipflops mit den Zuständen q_0 und q_1 benötigt. Für die drei möglichen Eingaben sind zwei binäre Eingangsvariablen x_0 und x_1 erforderlich. Rechts in Tabelle 7.13 ist aufgelistet, wie ein JK-Flipflop anzusteuern ist, um die vier möglichen Zustandsübergänge zu realisieren.

Tabelle 7.13: Codierung der Eingangs- und Zustandselemente des Modellautomaten und Zustandsübergangs-Tabelle für das JK-Flipflop

	x_1	x_0
X_0	0	0
X_1	0	1
X_2	1	0
–	1	1

	q_1	q_0
Z_0	0	0
Z_1	0	1
Z_2	1	0
–	1	1

Übergang $q_i^n \rightarrow q_i^{n+1}$	J	K
$0 \rightarrow 0$	0	*
$0 \rightarrow 1$	1	*
$1 \rightarrow 0$	*	1
$1 \rightarrow 1$	*	0

Zum Ermitteln der Ansteuerungsfunktionen für die JK-Flipflops dient Tabelle 7.14. Links ist eine umsortierte Übergangstabelle des Modellautomaten abgebildet. In dieser Darstellung sind

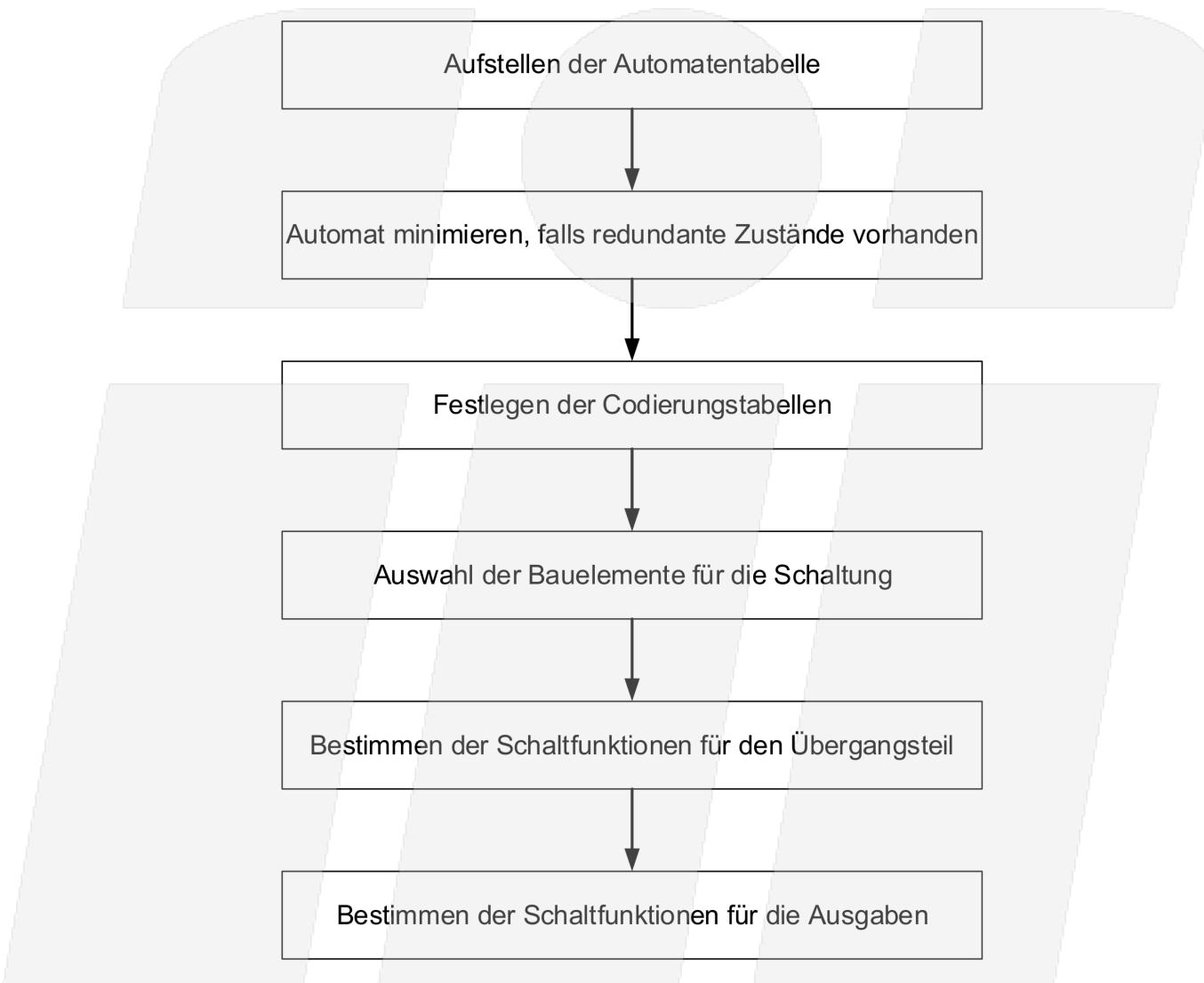


Abbildung 7.10: Lösungsweg für die technische Realisierung eines Automaten

die Spalten mit den Folgezuständen untereinander angeordnet. Die Codierung der Elemente gemäß Tabelle 7.13 ergibt die ersten sechs Spalten der rechten Produktionstabelle. Anhand der Zustandsübergänge und der Zustandsübergangs-Tabelle wird dann für die J- und K-Eingänge beider Flipflops für jeden Zustandsübergang ermittelt, wie sie anzusteuern sind. Die so entstandenen Spalten lassen sich direkt in ein KV-Diagramm (Abbildung 7.11) eintragen, wobei nicht definierte Variablenkombinationen mit „don't care“-Zeichen zu füllen sind.

Die minimierten Ansteuerfunktionen sind dann wie üblich ermittelbar. Sie lauten:

$$\begin{aligned} J_1 &= \overline{q_0}x_1 + q_0x_0 & K_1 &= x_0 + x_1 \\ J_0 &= q_1x_1 + \overline{q_1}x_0 & K_0 &= x_0 + x_1 \end{aligned}$$

Die sich ergebende Schaltung für den Übergangsteil ist in Abbildung 7.12 dargestellt.

Tabelle 7.14: Links: Umsortierte Übergangstabelle des Modellautomaten. Rechts: Produktions-tabelle zur Ermittlung der Ansteuerung der JK-Flipflops

X^{n+1}	Z^n	Z^{n+1}
X_0	Z_0	Z_0
X_0	Z_1	Z_1
X_0	Z_2	Z_2
X_1	Z_0	Z_1
X_1	Z_1	Z_2
X_1	Z_2	Z_0
X_2	Z_0	Z_2
X_2	Z_1	Z_0
X_2	Z_2	Z_1

X^{n+1}	Z^n	Z^{n+1}							
x_1	x_0	q_1	q_0	q_1	q_0	J_1	K_1	J_0	K_0
0	0	0	0	0	0	0	*	0	*
0	0	0	1	0	1	0	*	*	0
0	0	1	0	1	0	*	0	0	*
0	1	0	0	0	1	0	*	1	*
0	1	0	1	1	0	1	*	*	1
0	1	1	0	0	0	*	1	0	*
1	0	0	0	1	0	1	*	0	*
1	0	0	1	0	0	0	*	*	1
1	0	1	0	0	1	*	1	1	*

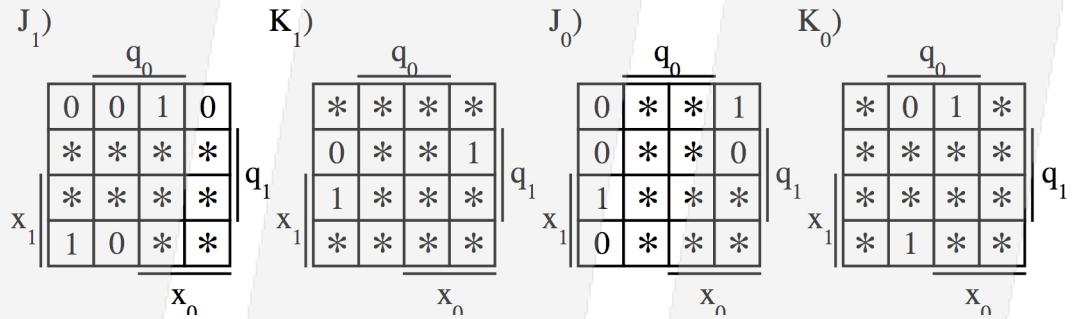


Abbildung 7.11: KV-Diagramme

Ausgangsteil

Nun soll noch der Ausgangsteil des Modellautomaten aus Abschnitt 7.2 unter Verwendung eines Festwertspeichers (PROM) und D-Flipflops realisiert werden. Dazu betrachten wir zunächst

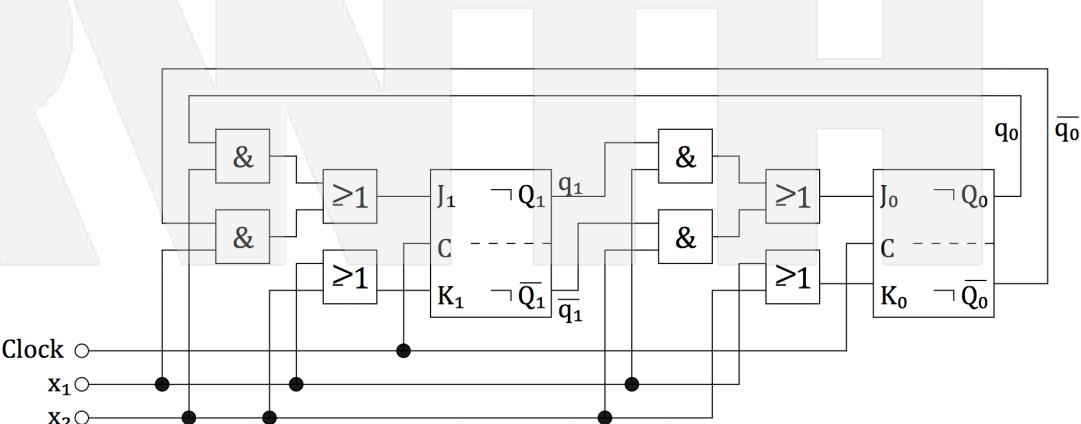


Abbildung 7.12: Schaltbild des Übergangsteils des Modellautomaten

noch einmal die Ausgangsfunktion eines Mealy-Automaten:

$$Y^{n+1} = g(X^{n+1}, Z^n)$$

Diese Funktion verknüpft den alten Zustand mit der neuen Eingabe zu der neuen Ausgabe. Die Eingabe darf also erst mit dem Takt wirksam werden. Um eine korrekte Ausgabe zu erhalten, werden, wie bereits in Abbildung 7.3 dargestellt, mit der Ausgangsfunktion Speicher angesteuert. Diese müssen synchron zu den Speichern des Übergangsteils getaktet sein. Das Blockschaltbild in Abbildung 7.13 zeigt nochmals einen Mealy-Automaten, in dem nun die Speicher als Flipflops realisiert sind.

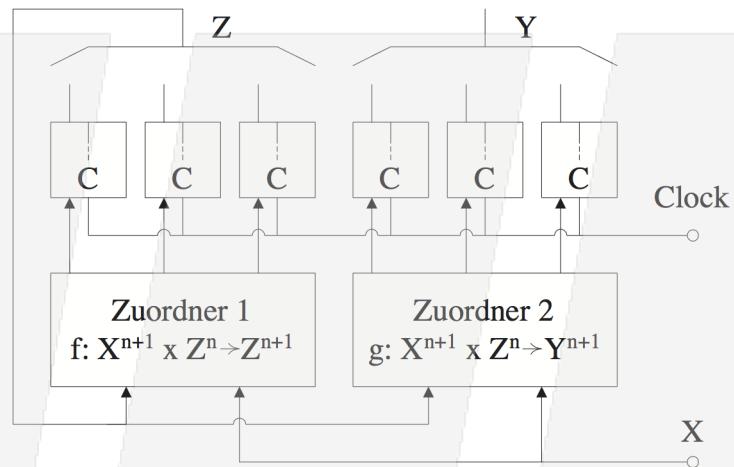


Abbildung 7.13: Blockschaltbild zur Ansteuerung der Speicher eines Mealy-Automaten

Zur Realisierung der Ausgangsfunktion wird zunächst eine Codierung der Ausgangselemente gemäß Tabelle 7.15 links definiert. Um die neun verschiedenen Ausgangselemente darzustellen, werden mindestens 4 binäre Variablen benötigt. Die dezimalen Indizes sind ziffernweise dual codiert.

Tabelle 7.15 mitte zeigt die umsortierte Ausgangstabelle des Modellautomaten. Rechts sind die Elemente gemäß Tabelle 7.13 und Tabelle 7.15 links codiert. Die erste Spalte beinhaltet die vier Eingangsvariablen für die Ansteuerfunktion als Hexadezimalzahl.

Aus Tabelle 7.15 rechts lassen sich die Ansteuerfunktion für die D-Flipflops direkt ablesen, da der neue Wert eines D-Flipflops nicht wie bei JK-Flipflops von dem zuvor gespeicherten Wert abhängt. Da ein PROM als Festwertspeicher benutzt werden soll und somit alle Minterme einzeln programmiert werden müssen, ist eine Minimierung nicht erforderlich. „Don't care“-Zeichen sind einfach durch Nullen zu ersetzen. Abbildung 7.14 zeigt die Schaltung, wobei nur die programmierbaren Verbindungen des PROMs explizit dargestellt sind. Die gesetzten Verbindungen sind direkt aus den letzten vier Spalten der Tabelle 7.15 rechts zu übernehmen. Der Takt und die vier Eingangsvariablen können direkt am Übergangsteil (Abbildung 7.12) abgegriffen werden.

Tabelle 7.15: Links: Codierung der Ausgangselemente des Modellautomaten. Mitte: Umsortierte Ausgangstabelle. Rechts: Codierte Ausgangstabelle.

	X^{n+1}	Z^n	Y^{n+1}		X^{n+1}	Z^n	Y^{n+1}					
				Hex	x_1	x_0	q_1	q_0	y_3	y_2	y_1	y_0
Y_{00}	X_0	Z_0	Y_{00}	0	0	0	0	0	0	0	0	0
Y_{01}	X_0	Z_1	Y_{11}	1	0	0	0	1	0	1	0	1
Y_{02}	X_0	Z_2	Y_{22}	2	0	0	1	0	1	0	1	0
				-	0	0	-	-	*	*	*	*
Y_{10}	X_1	Z_0	Y_{01}	4	0	1	0	0	0	0	0	1
Y_{11}	X_1	Z_1	Y_{12}	5	0	1	0	1	0	1	1	0
Y_{12}	X_1	Z_2	Y_{20}	6	0	1	1	0	1	0	0	0
Y_{20}				-	0	1	-	-	*	*	*	*
Y_{21}	X_2	Z_0	Y_{02}	8	1	0	0	0	0	0	1	0
Y_{22}	X_2	Z_1	Y_{10}	9	1	0	0	1	0	1	0	0
				A	1	0	1	0	1	0	0	1
				-	1	0	-	-	*	*	*	*

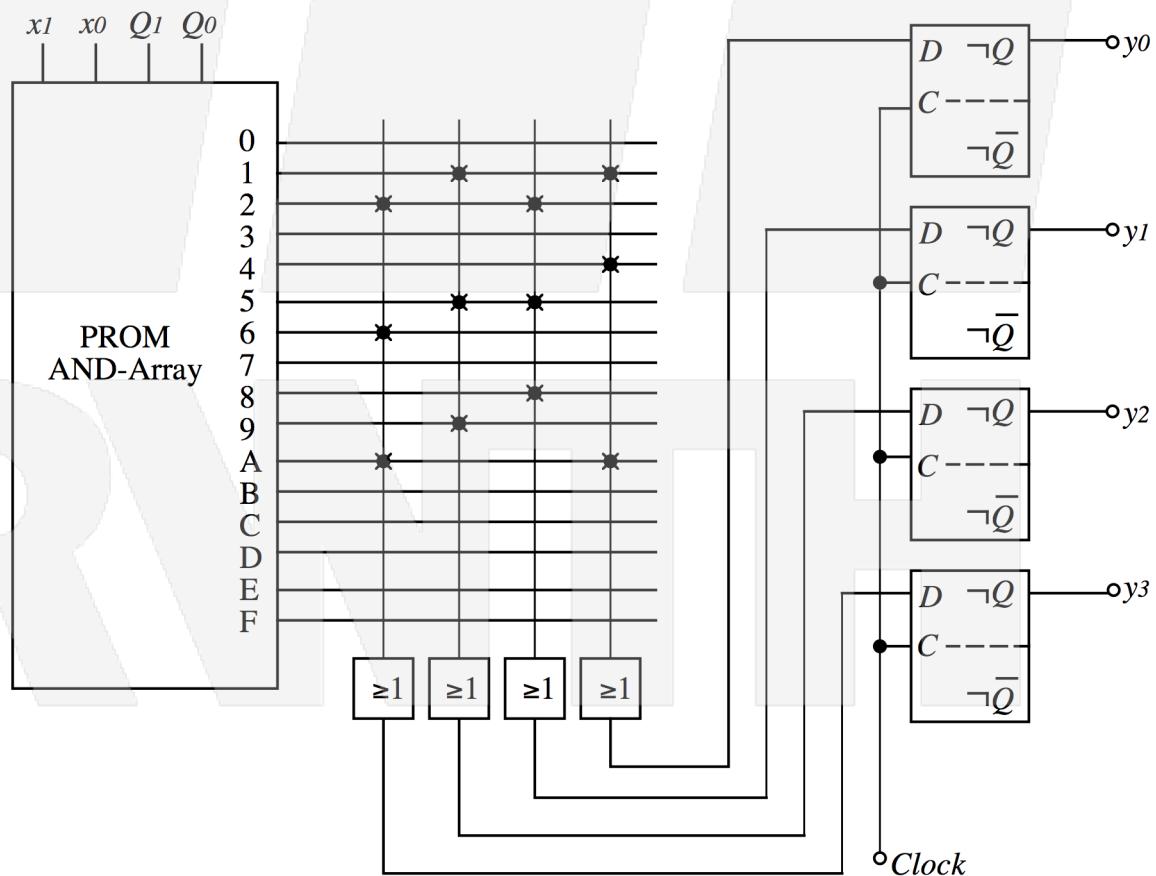


Abbildung 7.14: Schaltbild des Ausgangsteils des Modellautomaten

Kapitel 8

Aufbau und Funktion einer Zentraleinheit

Der Aufbau eines Digitalrechners folgt meist dem so genannten Von-Neumann-Prinzip. Dabei werden Programme und Daten in einem gemeinsamen Speicher gehalten. Daraus ergeben sich bestimmte Aufgaben, die eine Zentraleinheit erfüllen muss. Im Folgenden wird darum zunächst der prinzipielle Aufbau eines Von-Neumann-Rechners erläutert.

8.1 Der Von-Neumann-Rechner

Ein Von-Neumann-Rechner ist durch die Speicherung von Daten und Programm in einem gemeinsamen Speicher gekennzeichnet. Er besteht in der Regel aus folgenden Komponenten:

- Zentraleinheit (CPU: Central Processing Unit), bestehend aus
 - Steuerwerk (CU: Control Unit)
 - Rechenwerk (ALU: Arithmetical and Logical Unit)
 - Speicher
 - * Programm- oder Befehlszähler (BZ)
 - * Befehlsregister (BR)
 - * Speicheradressregister (SAR)
 - * Speicherdatenregister (SDR)
 - * mehrere Arbeitsregister
- Hauptspeicher
- Adressbus
- Datenbus
- Steuerleitungen
- E/A-Geräte

Abbildung 8.1 zeigt den prinzipiellen Aufbau eines Digitalrechners, während Abbildung 8.2 die Anordnung der Komponenten innerhalb der Zentraleinheit darlegt.

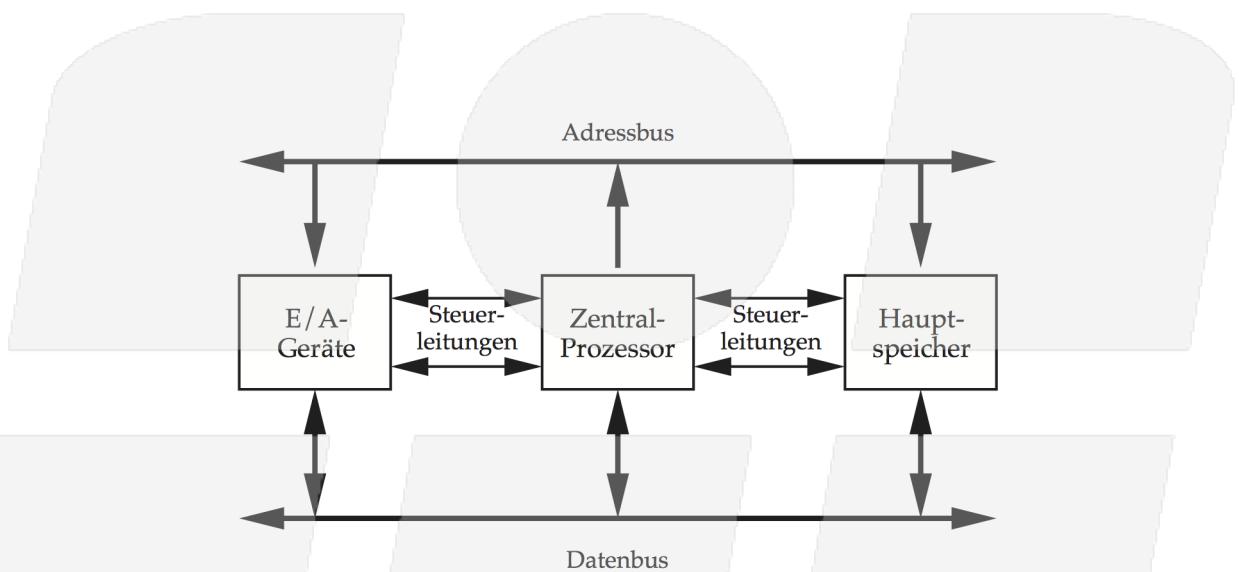


Abbildung 8.1: Schematischer Aufbau eines einfachen Rechners

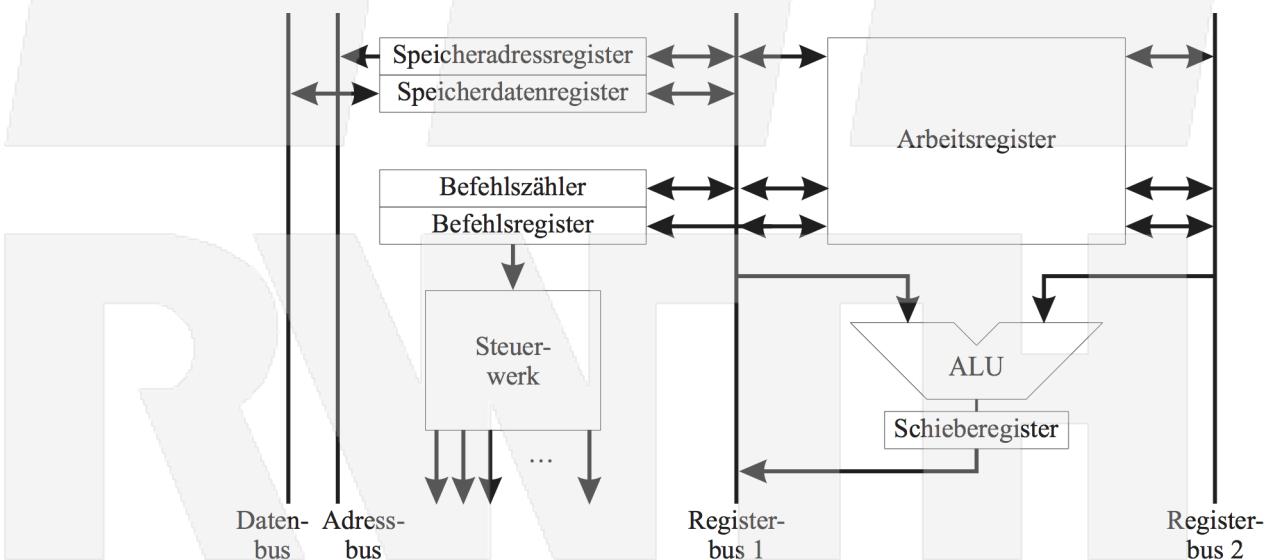


Abbildung 8.2: Zentralprozessor (Central Processing Unit, CPU) mit Arbeitsregistern und zwei internen Bussen

Abbildung 8.3 erläutert, wie die Abarbeitung eines Befehls im Digitalrechner abläuft:

- Das Programm und die Daten befinden sich zunächst im Hauptspeicher.
- Das Steuerwerk initiiert das Laden eines gespeicherten Befehls. Der Befehlszähler (BZ) adressiert dazu den Hauptspeicher über das Speicheradressregister (SAR) und lädt den gespeicherten Befehl über das Speicherdatenregister (SDR) in das Befehlsregister (BR).
- Das Steuerwerk erzeugt eine dem Befehl entsprechende Anweisung für den Prozessor, z. B.:
 - Ausführung einer arithm. oder log. Operation durch die ALU (siehe Abb. 8.3)
 - Bereitstellen von Operanden für die ALU
 - Laden/Speichern von Arbeitsregistern
 - Erhöhen des BZ (durch den Inkrementierer)
 - Laden von Sprungbefehlen

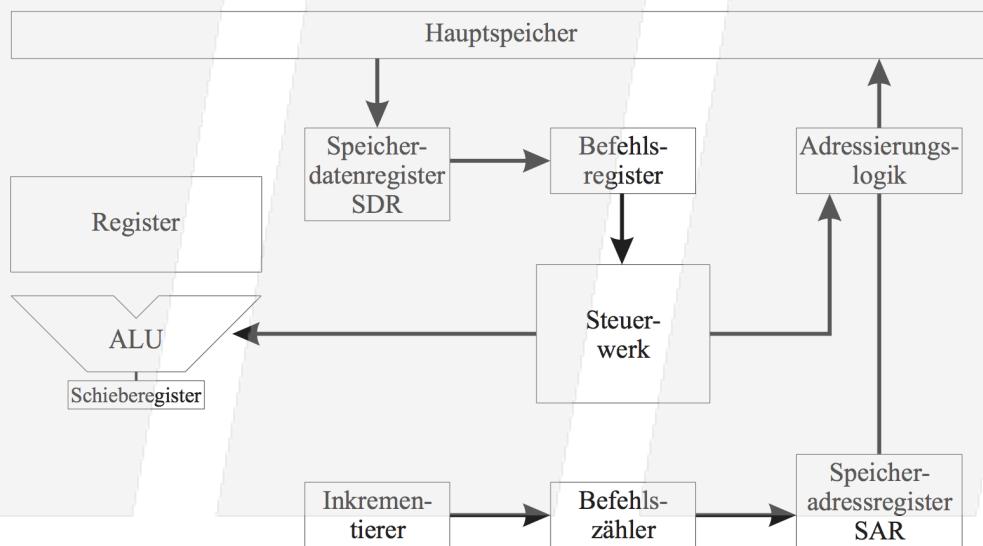


Abbildung 8.3: Befehlswege im Rechner

Der oben skizzierte Rechner arbeitet nach dem klassischen von Neumann-Prinzip:

- Der Rechner enthält zumindest Speicher, Rechenwerk, Steuerwerk und Ein-/ Ausgabegeräte.
- Die Zentraleinheit arbeitet taktgesteuert.
- Die intern verwendete Signalmenge ist binär codiert.
- Im Rechner werden Worte fester Länge parallel verarbeitet.
- Der Hauptspeicher besteht aus fortlaufend adressierten Speicherworten.
- Der Inhalt eines Speicherwortes wird über seine Adresse angesprochen.
- Der Rechner verarbeitet extern eingegebene Programme, die intern gespeichert werden.

- Programmbefehle und Datenworte werden im einheitlichen Hauptspeicher gespeichert.
- Der Rechner verarbeitet Programmbefehle und Datenworte sequentiell (Single Instruction - Single Data, SISD).
- Die normale Verarbeitung der Programmbefehle geschieht fortlaufend in der Reihenfolge der Speicherung der Programmbefehle. Diese sequentielle Verarbeitung kann durch Sprungbefehle oder datenbedingte Verzweigungen verändert werden.

Vorteile der von Neumann Architektur:

- minimaler Hardware-Aufwand (heute nicht mehr bedeutsam)

Nachteile der von Neumann Architektur:

- physikalischer Flaschenhals:
Kommunikation zwischen CPU und Hauptspeicher.
 - Eine Operation wird jeweils nur auf den Inhalt der angegebenen Speicherzelle angewendet. Es kann nicht erkannt werden, ob der Speicherzelleninhalt ein Befehl, ein Datum oder eine Adresse ist.
- intellektueller Flaschenhals:
 - Es ist notwendig, die Lösung eines Anwendungsproblems in eine Folge sequentieller Lösungsschritte umzusetzen.

8.2 Rechenwerk

Das *Rechenwerk* besteht aus einer ALU (Arithmetic and Logical Unit) und einem Satz von Arbeitsregistern.

Aufgaben des Rechenwerks:

- Arithmetische Operationen wie Addition, Inkrement, Dekrement, Komplementierung.
- Logische Operationen wie bitweise Negation, Konjunktion, Disjunktion, XOR-Bildung.
- Verschiebungsoperationen (zyklische- und nichtzyklische Links- und Rechtsshifts).
- Vergleichsoperationen zwischen Zahlen.

Die Abarbeitung kann seriell oder parallel erfolgen.

Eine ALU besteht aus Wortaddierer, Schieberegister und parametrischer Eingangsschaltung. Die parametrische Eingangsschaltung (Abb. 8.4) liefert über Steuerleitungen alle 4 einstelligen Booleschen Funktionen einer Eingangsgröße.

Konstruktion einer einfachen arithmetischen Einheit: Wortaddierer mit vorgeschaltetem parametrischem Eingang und wählbarem Überlaufbit c_0 , Steuerleitungen a, b, c (Abb. 8.5).

Zur Erzeugung logischer Funktionen wird der Übertrag von einer Zelle zur nächsten durch eine zusätzliche Steuerleitung ($d = 0$) unterbrochen (Abb. 8.6). Als logische Operationen werden nach der untenstehenden Tabelle 8.1 die Identität, Antivalenz, Äquivalenz und Negation gewonnen. Es fehlen für eine vollständige logische Basis die Disjunktion und Konjunktion.

a	b	$g(a, b, y)$
0	0	0
0	1	\bar{y}
1	0	y
1	1	1

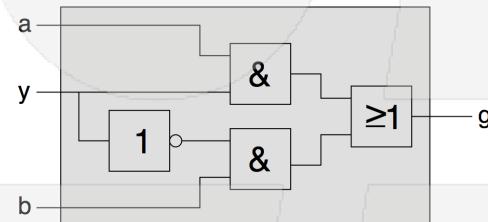
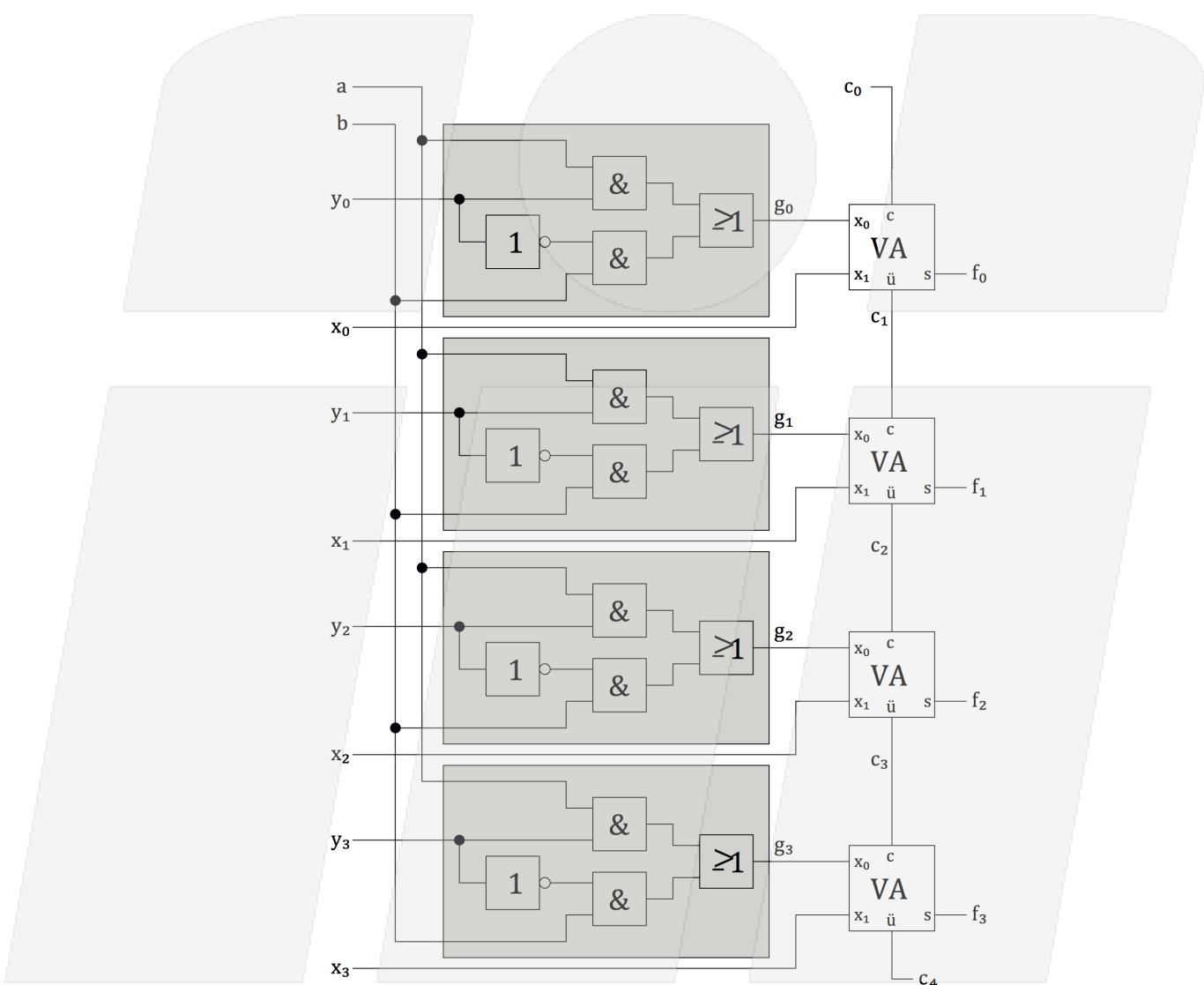


Abbildung 8.4: Schaltung zur Erzeugung der 1-stelligen Schaltfunktionen

Tabelle 8.1: Logische Funktionen der abgebildeten ALU

a	b	c_0	d	$g(a, b, y)$	$f(a, b, c, d, x, y)$
0	0	—	0	0	x
0	1	—	0	\bar{y}	$x \oplus \bar{y} \iff (x \leftrightarrow y)$
1	0	—	0	y	$x \oplus y$
1	1	—	0	1	$x \oplus 1 \iff \bar{x}$



a	b	c_0	$g(a, b, y)$	$f(a, b, c_0, x, y)$	
0	0	0	0	x	
0	0	1	0	$x + 1$	
0	1	0	\bar{y}	Subtr. von $x - y$ im Einerkomplement	$(= x + 2^{n+1} - 1 - y)$
0	1	1	\bar{y}	Subtr. von $x - y$ im Zweierkomplement	$(= x + 2^{n+1} - y)$
1	0	0	y	$x + y$	
1	0	1	y	$x + y + 1$	
1	1	0	1	Subtr. von $x - 1$ im Zweierkomplement	$(= x + 2^{n+1} - 1)$
1	1	1	1	x	$(= x + 2^{n+1})$

Abbildung 8.5: Funktionen und Schaltnetze einer arithmetischen Einheit

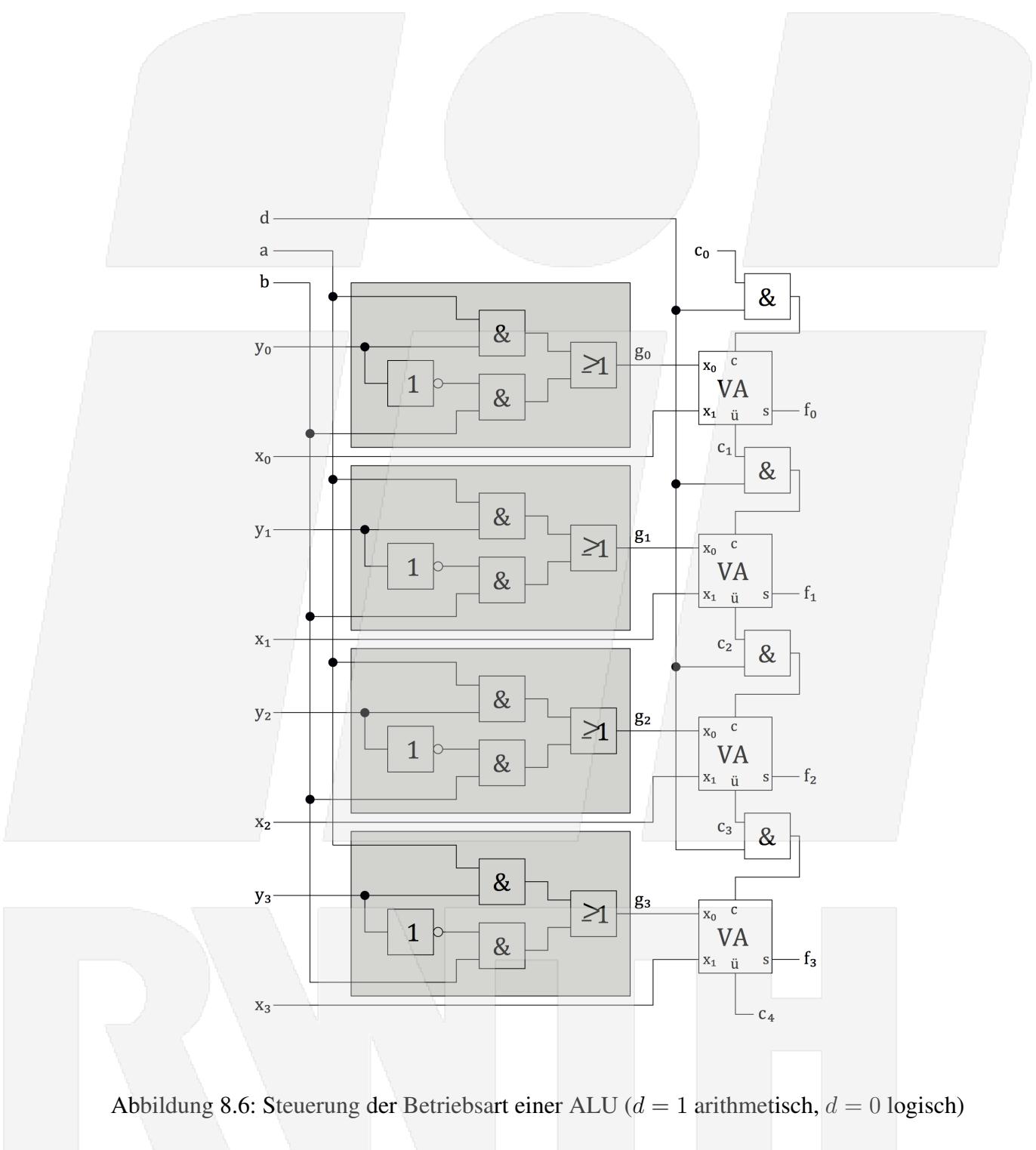


Abbildung 8.6: Steuerung der Betriebsart einer ALU ($d = 1$ arithmetisch, $d = 0$ logisch)

Kernzelle einer vollständigen ALU

Es wird eine schaltungstechnische Erweiterung durchgeführt, um die noch benötigte Konjunktion und Disjunktion zu realisieren (Abb. 8.7). Dazu bietet sich an, auf die Identität ($a = b = 0$) sowie auf die Äquivalenz ($a = 0, b = 1$) im logischen Betrieb ($d = 0$) zu verzichten. Die Identität ist bereits im arithmetischen Betrieb vorhanden und die Äquivalenz ist bei Bedarf als Komplement der Antivalenz ($a = 1, b = 1$) darstellbar.

Tabelle 8.2 sind die Funktionen der erweiterten ALU in beiden Betriebsarten zu entnehmen. Die Funktion $h(a, b, d, x, y)$ ergibt sich dabei aus:

$$\begin{aligned} h(a, b, d, x, y) &= x + \bar{a} \bar{b} \bar{d} y + \bar{a} b \bar{d} \bar{y} \\ &= x + \bar{a} \bar{d} (\bar{b}y + b\bar{y}) \\ &= x + \bar{a} \bar{d}(b \oplus y) \end{aligned}$$

Tabelle 8.2: Funktionen der ALU Kernzelle

a	b	c_0	d	\parallel	$g(a, b, y)$	\parallel	$h(a, b, d, x, y)$	\parallel	Fkt. $f(a, b, c, d, x, y)$	\parallel	Bezeichnung
Logische Funktionen											
0	0	—	0		0		$x + y$		$(x + y) \oplus 0 = x + y$		Disjunktion
0	1	—	0		\bar{y}		$x + \bar{y}$		$(x + \bar{y}) \oplus \bar{y} = xy$		Konjunktion
1	0	—	0		y		x		$x \oplus y$		Antivalenz
1	1	—	0		1		x		$x \oplus 1 = \bar{x}$		Einerkomplement
Arithmetische Funktionen											
0	0	0	1		0		x		x		Identität (Transfer)
0	0	1	1		0		x		$x + 1$		Inkrement
0	1	0	1		\bar{y}		x		$x - y$ im Einerkompl.		Subtraktion
0	1	1	1		\bar{y}		x		$x - y$ im Zweierkompl.		Subtraktion
1	0	0	1		y		x		$x + y$		Addition
1	0	1	1		y		x		$x + y + 1$		Addition mit Übertrag
1	1	0	1		1		x		$x - 1$		Dekrement
1	1	1	1		1		x		x		Identität (Transfer)

Die Multiplikation von Dual- und Festkommazahlen kann bei einer solchen ALU mit Hilfe der fortgesetzten Addition durchgeführt werden (Tabelle 8.3). Eine mögliche Hardwareumsetzung zeigt Abbildung 8.8. Für die Multiplikation zweier n -stelligen Bitzahlen kommt ein Schieberegister mit $2n$ -Bitstellen zum Einsatz. Der Multiplikant wird in das Register X und der Multiplikator in die untere Hälfte des Registers Y ($y_{n-1} \dots y_0$) geladen. Die obere Hälfte des Registers Y ist mit 0 vorbesetzt. Nun wird die obere Hälfte des Registers Y mit dem Register X addiert, falls das niederwertigste Bit des Multiplikators (Y_0) 1 ist. Bei $Y_0 = 0$ wird statt X lediglich eine 0 addiert. Nach diesem Schritt wird das gesamte Register Y nach rechts geschoben und die Addition erfolgt aufs Neue. Dieser Vorgang wird n -mal wiederholt. Das Ergebnis der Multiplikation steht danach im Register Y .

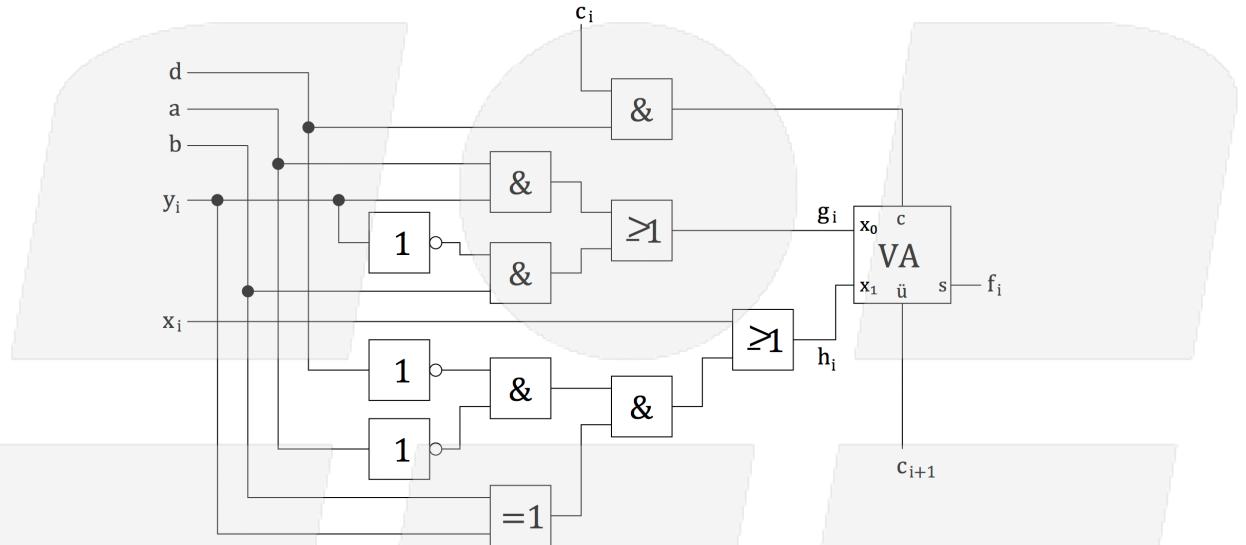


Abbildung 8.7: Kernzelle einer ALU

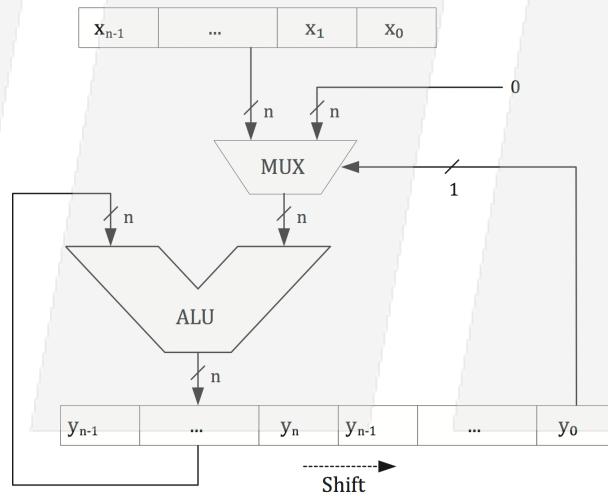


Abbildung 8.8: Multiplikation mit einer ALU

Tabelle 8.3: Multiplikationsschema für Dual- und Festkommazahlen

x_6	x_5	x_4	x_3	x_2	x_1	x_0	*	y_6	y_5	y_4	y_3	y_2	y_1	y_0
								x_6y_0	x_5y_0	x_4y_0	x_3y_0	x_2y_0	x_1y_0	x_0y_0
								x_6y_1	x_5y_1	x_4y_1	x_3y_1	x_2y_1	x_1y_1	x_0y_1
					x_6y_2	x_5y_2		x_4y_2	x_3y_2	x_2y_2	x_1y_2	x_0y_2		
					x_6y_3	x_5y_3	x_4y_3	x_3y_3	x_2y_3	x_1y_3	x_0y_3			
					x_6y_4	x_5y_4	x_4y_4	x_3y_4	x_2y_4	x_1y_4	x_0y_4			
					x_6y_5	x_5y_5	x_4y_5	x_3y_5	x_2y_5	x_1y_5	x_0y_5			
					x_6y_6	x_5y_6	x_4y_6	x_3y_6	x_2y_6	x_1y_6	x_0y_6			
z_{13}	z_{12}	z_{11}	z_{10}	z_9	z_8	z_7		z_6	z_5	z_4	z_3	z_2	z_1	z_0

ALU mit Statusregister für Zahlenvergleiche

Statusregisterinhalt:

C = Carry (Übertragsbit), N = Negative (Ergebnis ist negativ)

Z = Zero (Ergebnis besteht nur aus Nullen), V = Overflow (Überlauf)

Dabei ergibt sich das Overflow-Bit aus der XOR-Verknüpfung $V = c_n \oplus c_{n-1}$.

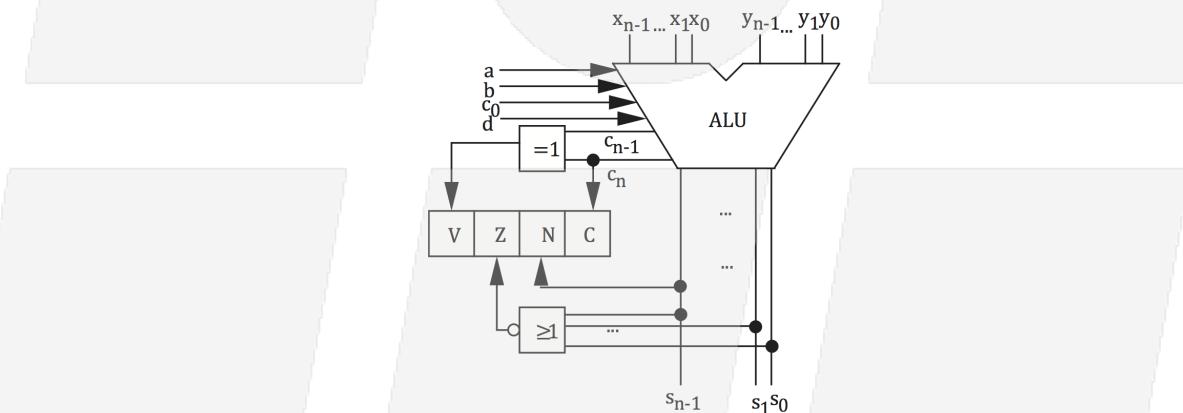


Abbildung 8.9: Statusregister zur Anzeige von Vergleichsoperationen in der ALU

Tabelle 8.4: Statusregisterbelegung beim Vergleich zweier Binärzahlen

Relation	Statusregisterbelegung
$x = y$	$Z = 1$
$x \neq y$	$Z = 0$
$x \geq y$	$N = V$
$x < y$	$N \neq V$
$x > y$	$N = V$ und $Z = 0$
$x \leq y$	entweder $N \neq V$ oder $Z = 1$

Beispiel:

Vergleich der Zahlen $x = -8$ (1000) und $y = +7$ (0111) für eine 4-Bit ALU

$$\begin{array}{r}
 1 \ 0 \ 0 \ 0 \ -8 \quad x \\
 + \ 1 \ 0 \ 0 \ 1 \ -7 \quad -y \text{ (Zweierkomplement)} \\
 \hline
 1 \ 0 \ 0 \ 0 \ 0 \quad c \\
 \hline
 (1) \ 0 \ 0 \ 0 \ 1 \quad s
 \end{array}$$

Entsprechend der Abbildung 8.9 ist $c_{n-1} = 0$ und $c_n = C = 1$ und damit $V = 1$. Es liegt also ein Überlauf vor, d.h. die Zahl $(-8) + (-7) = -15$ ist im Zweierkomplement durch s mit 4 Datenleitungen nicht darstellbar. Mit $s_{n-1} = 0$ gilt $N = 0$. Damit ist entsprechend dem Fall $x < y$ Tabelle 8.4 $N \neq V$ erfüllt. Mit $s_0 = 1$ ergibt sich über die Peirce-Verknüpfung der s -Datenleitungen $Z = 0$. Z zeigt somit die Ungleichheit von x und y an.

8.3 Steuerwerk (Control Unit)

Unabhängig davon, wie komplex ein Rechenwerk aufgebaut ist und welche Operationen es durchführen kann, es ist auf das Erledigen von atomaren Operationen (Rechnungen, logische Verknüpfungen, etc.) beschränkt. Es kann jedoch seine internen Abläufe nicht selbst beeinflussen, sondern muss durch eine externe Einheit kontrolliert werden. Diese Einheit muss den Ablauf der Rechnung steuern und die Operanden bereitstellen.

In der allgemeinsten Form ist ein Steuerwerk durch einen Automaten realisiert, der zu jedem Zeitpunkt für die gesamte CPU festlegt, was in welcher Funktionseinheit zu tun ist.

Aufgaben:

- Befehlsdecodierung (Abb. 8.10)
- Koordination der zeitlichen Abläufe im Rechner
 - Befehl holen
 - Vorbereitung der Adressierung des nächsten Befehls
 - Befehl interpretieren (Decodieren)
 - Operanden holen (falls erforderlich)
 - Operation ausführen
 - Wegschaltung der Busse

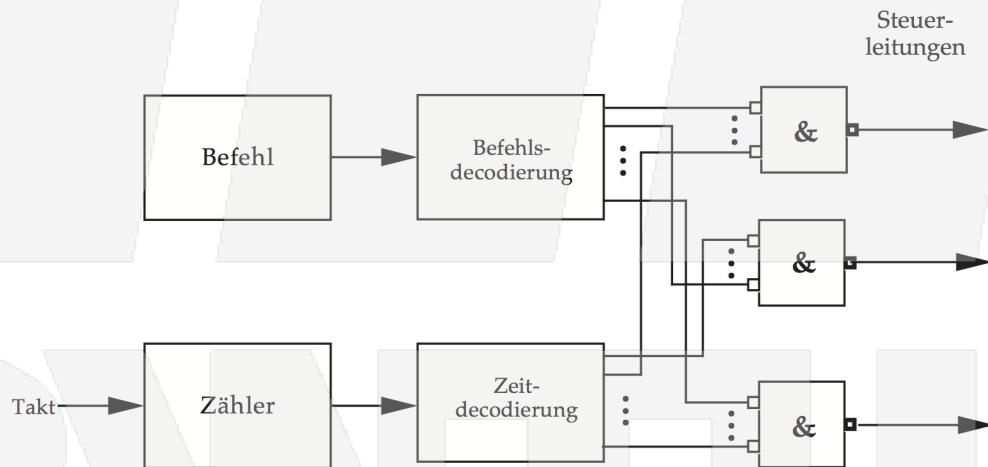
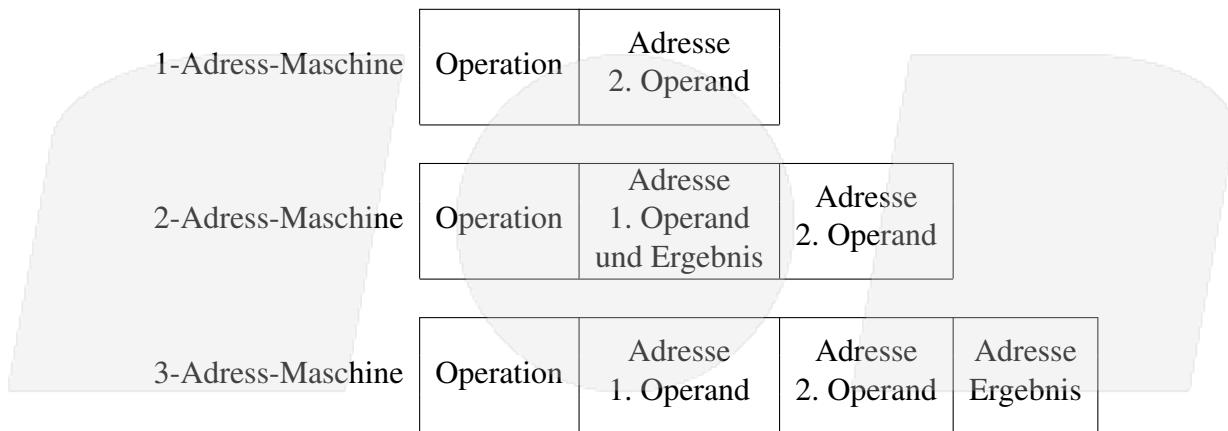


Abbildung 8.10: Schema der Befehlsdecodierung

Operandenadressierung:

Die Adressierung der Operanden hängt davon ab, wie viele Adressangaben die Maschine erlaubt. Die 1-Adress-Maschine kann pro Operation nur eine Adresse verarbeiten, aus der der zweite Operand geladen wird. Der erste Operand steht immer im Akkumulator, in den auch das Ergebnis geschrieben wird. Die 2-Adress-Maschine erlaubt die gleichzeitige Adressierung zweier Operanden, wobei in einer der beiden angegebenen Adressen auch das Ergebnis gespeichert wird (im Beispiel unten an der Adresse des 1. Operanden). Eine 3-Adress-Maschine erlaubt darüber hinaus die Adressierung zweier Operanden sowie die explizite Angabe der Adresse für den Ergebniswert.



Als Beispiel zeigt Abbildung 8.11 eine 16 Bit 3-Adress-ALU mit 7 Arbeitsregistern. Der Aufbau eines Steuerwortes (hier 16 Bit breit) ist wie folgt:

- X Auswahl eines Quellregisters als x-Operand der ALU
- Y Auswahl eines Quellregisters als y-Operand der ALU
- F Funktion der ALU
- S Schieberegistersteuerung
- R Auswahl des ZielRegisters

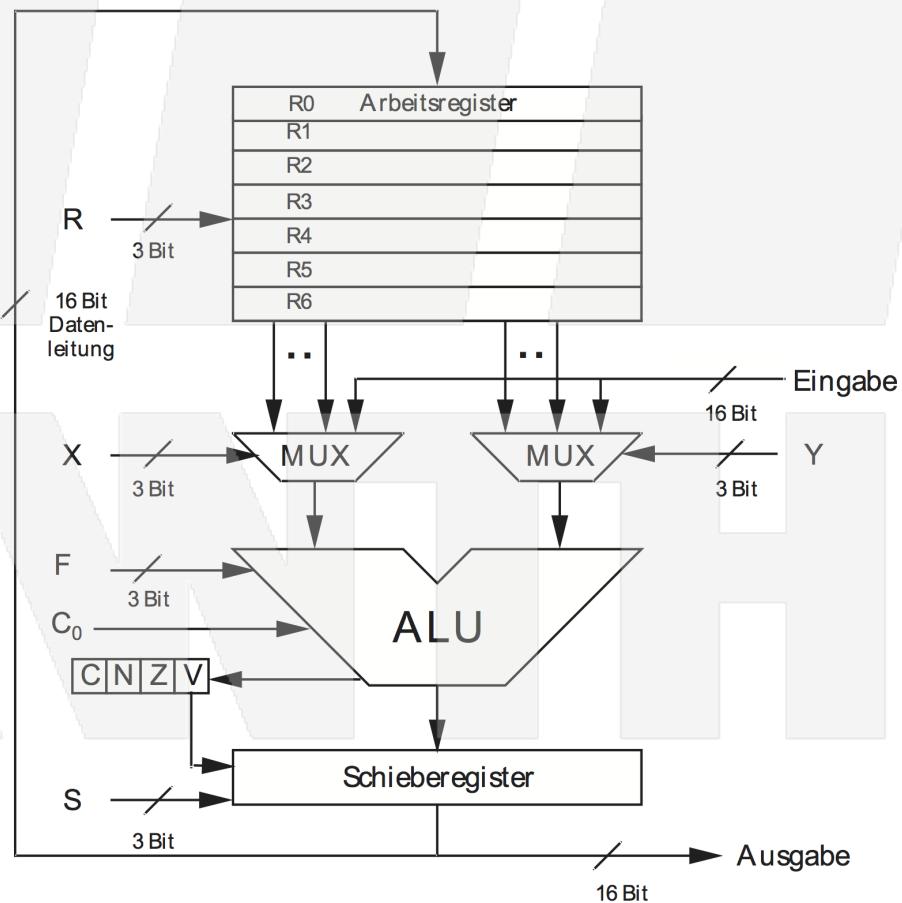


Abbildung 8.11: Schaltung einer 16 Bit 3-Adress-ALU

Steuerworte stehen in ROM- oder RAM-Speichern. Steuerwortbreiten sind z. B. bei:

VAX 11/750 → 80 Bit
 VAX 11/780 → 99 Bit

8.4 Mikroprogrammierung

Ein Maschinenbefehl initiiert unter Umständen eine Folge von Mikrobefehlsschritten. Bei jedem Schritt wird ein Mikroprogrammwort aus dem Mikroprogrammspeicher gelesen (Abb. 8.12).

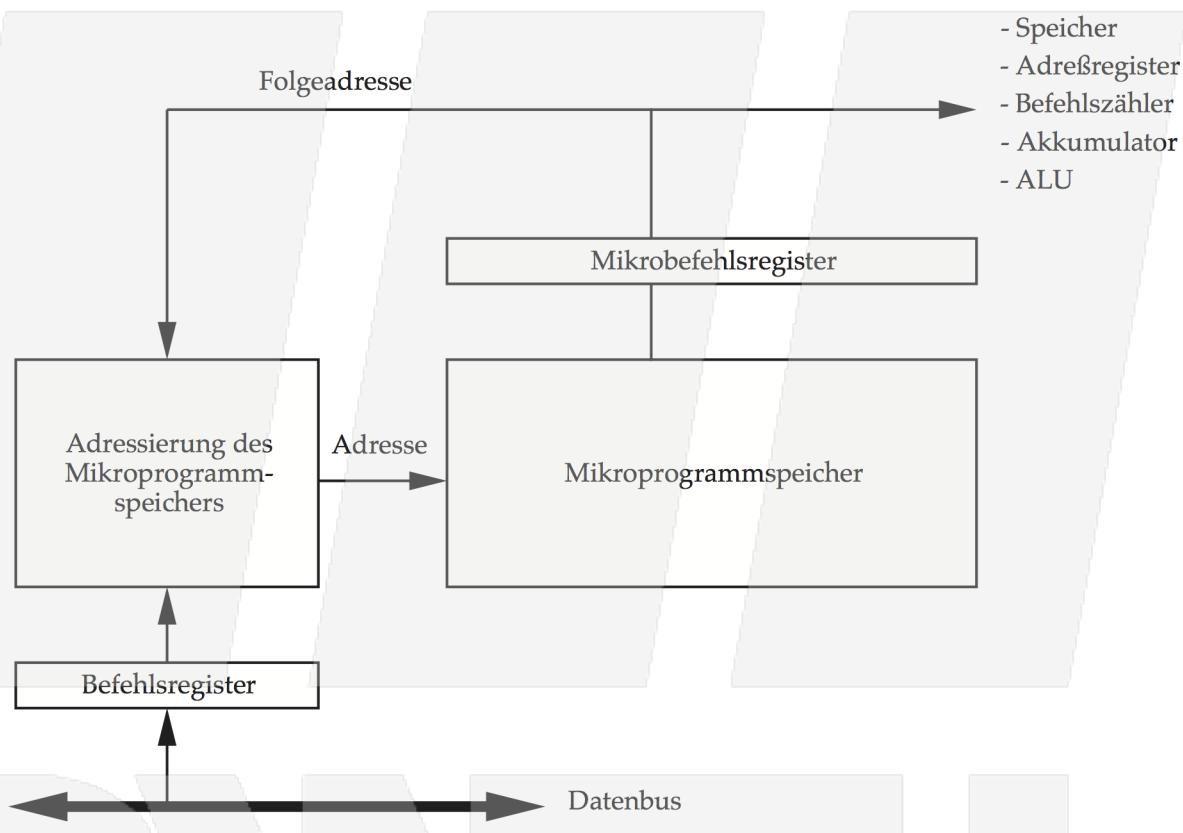


Abbildung 8.12: Mikroprogrammiertes Steuerwerk

Der Maschinenbefehl *Addiere Operand zum Akkumulator* wird z.B. rechnerintern durch folgendes Mikrogramm realisiert:

1. Mikrobefehl: Befehl lesen → Befehlsregister
2. Mikrobefehl: Operandenadresse lesen → Adressregister
3. Mikrobefehl: Operand lesen, Operand+Akkumulator → Akku

8.5 Kern der Zentraleinheit (CPU)

Eine *CPU* besteht aus einer ALU, einem Steuerwerk und Registern (Abb. 8.13).

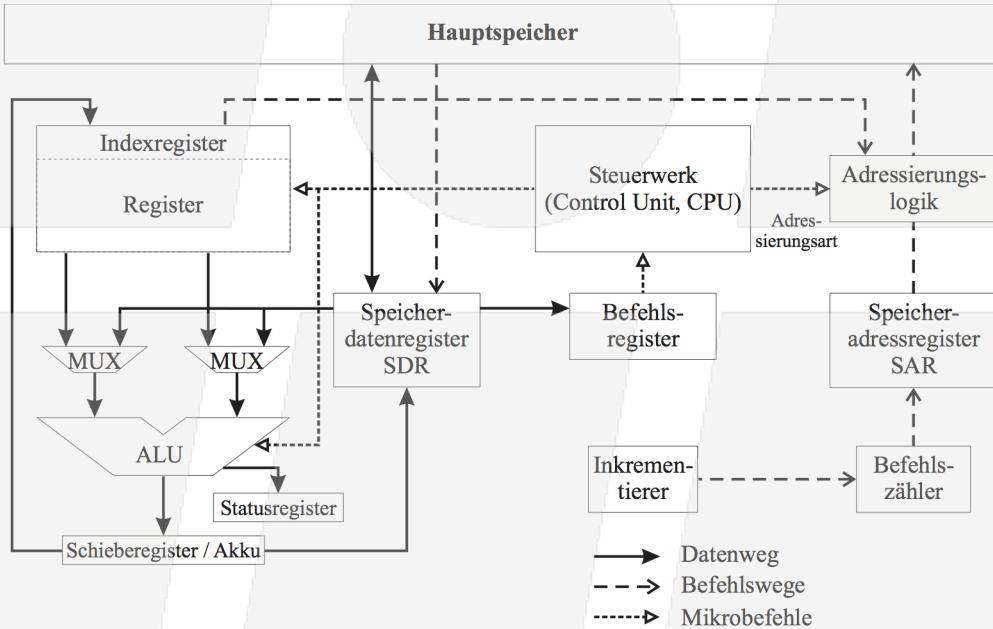


Abbildung 8.13: Zentraleinheit und Hauptspeicher

Ein-/Ausgabeverwaltung: Jedem Gerät wird eine Speicheradresse zugeteilt. Damit wird auf das Gerät wie auf eine Speicherzelle zugegriffen (Memory Mapped I/O).

Fetch-Execute Zyklus (die Datenflüsse sind in Abb. 8.13 dargestellt):

- nächsten Maschinenbefehl lesen (gestrichelt)
- Befehl ausführen (schwarz)

8.6 Abweichungen vom von Neumann-Konzept

8.6.1 Harvard-Architektur

Bei der *von Neumann-Architektur* eines Rechners sind Daten und Programm in einem gemeinsamen Speicher abgelegt. Rechner mit diesem Organisationsprinzip sind einfach aufzubauen, allerdings können Befehle und Daten nur nacheinander geholt werden. Im Gegensatz dazu werden bei Rechnern mit *Harvard-Architektur* getrennte Daten- und Programmspeicher verwendet. Der von H. Aiken von 1939 bis 1944 an der Harvard-Universität entwickelte *Mark I* war einer der ersten Computer dieser Architektur. Für seinen Daten-Speicher wurden elektromechanisch angetriebene Drehwähler der Fernmeldetechnik verwendet und sein Programm-Speicher bestand aus einem Lochstreifen. In letzter Zeit werden Prozessoren (vor allem RISC) mit getrenntem Daten- und Befehlsbus ausgestattet. Dies bringt sowohl einen höheren Hardwareaufwand als auch eine Verdoppelung der Speicherbandbreite mit sich. Zur Begrenzung des Aufwands wird oft der gemeinsame (große) Hauptspeicher beibehalten und nur momentan oft benötigte Programm- und Datenblöcke in getrennten Caches bereithalten (Abb. 8.14).

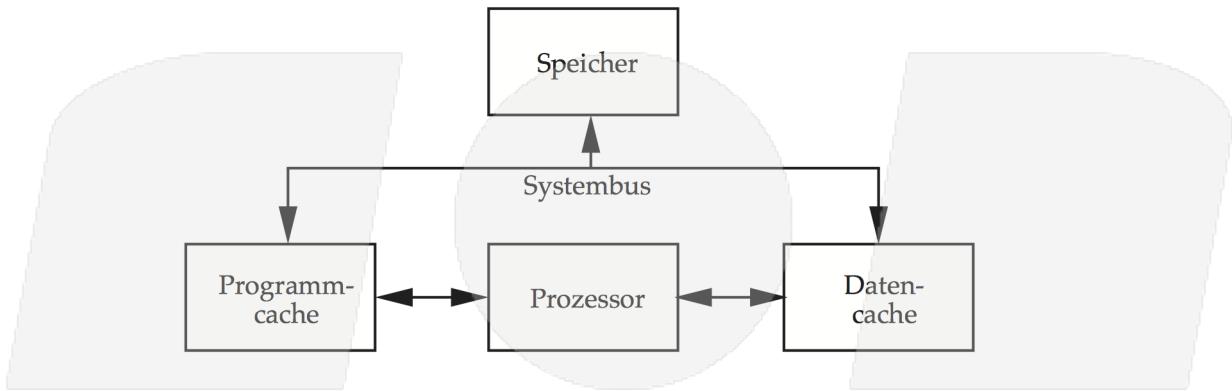


Abbildung 8.14: Prozessor mit HARVARD-Architektur und großem Hauptspeicher

8.6.2 Pipelining

Die Abarbeitung eines Befehls im von Neumann-Rechner läuft in mehreren Phasen ab, so dass auch die zur Abarbeitung benötigte Hardware in verschiedene Segmente unterteilt werden kann (Instruktionssegmentierung):

- S₁: Laden eines Befehls in das BR
- S₂: Befehlsdecodierung durch das Steuerwerk
- S₃: Bereitstellen der Operanden
- S₄: Ausführung des Befehls durch die ALU
- S₅: Speichern des Resultats

Das Diagramm 8.15 zeigt auf, zu welchen Zeitpunkten die Segmente in einem von Neumann-Rechner mit der Abarbeitung eines Befehls beschäftigt sind. Es wird noch einmal deutlich, dass der klassische von Neumann-Prozessor streng sequentiell arbeitet, d.h. zu jedem Zeitpunkt wird genau eine Aktion ausgeführt.

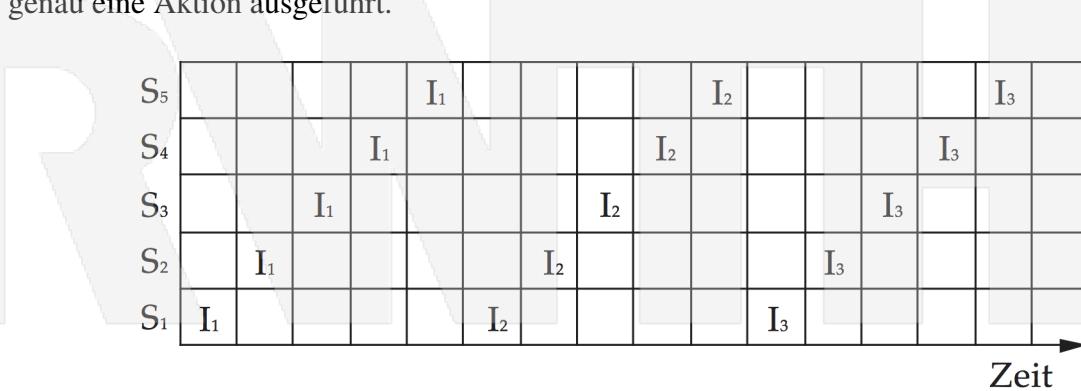


Abbildung 8.15: Raum-Zeit-Diagramm für einen von Neumann-Prozessor

In der Abbildung 8.16 sind die Segmente nun als sog. Instruktionsspipeline angeordnet. Mit Pipelining ergibt sich das Raum-Zeit-Diagramm nach Abbildung 8.17.

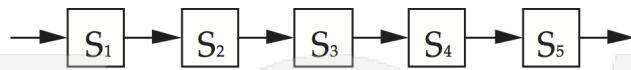


Abbildung 8.16: Instruktionspipeline

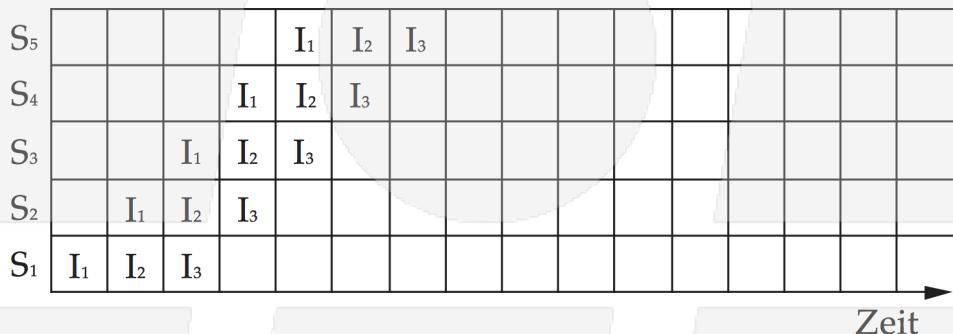


Abbildung 8.17: Raum-Zeit-Diagramm mit Pipelining

In der Decodierungsphase des ersten Befehls wird der nächste Befehl schon in das BR geladen. Während die Operanden für I₁ bereitgestellt werden, kann der zweite Befehl decodiert und der dritte Befehl geladen werden. Auf diese Weise wird der physikalische Flaschenhals umgangen.

Bei k Segmenten beträgt der Zeitbedarf zur Ausführung von n Instruktionen

- $T = n \cdot k$ für einen von Neumann-Prozessor
- $T_P = n + k - 1$ für einen Prozessor mit Instruktionspipeline

$$\text{Speedup} := \frac{T}{T_P} = \frac{k \cdot n}{n+k-1} \rightarrow k \quad (n \rightarrow \infty)$$

Bei einer Pipeline mit 5 Segmenten ergibt sich also ein Speedup von ebenfalls 5. Nach 4 Schritten (Startup-Phase der Pipeline) wird in jedem Zeitschritt ein Resultat geliefert. Pipelining funktioniert natürlich nur solange, wie die Befehle eines Maschinenprogramms nacheinander abgearbeitet werden. Nach der Ausführung eines Sprungbefehls (vgl. Kapitel 9) muss die Startup-Phase hingegen wieder neu durchlaufen werden.

Arithmetisches Pipelining

Genau wie die Ausführung eines Befehls in verschiedene Teilschritte unterteilt werden kann, so lassen sich auch Gleitkommaoperationen aufteilen.

Beispiel: Segmentierung der Gleitkommaaddition

- S1: Exponentenvergleich
- S2: Mantissenverschiebung
- S3: Mantissenaddition
- S4: Normalisierung

8.6.3 Sprungvorhersage (Branch Prediction)

Innerhalb eines Programmes besitzen Programmverzweigungen eine gewisse Wahrscheinlichkeit. Dies lässt sich ausnutzen, um eine Sprungvorhersage durchzuführen. In einer Tabelle (Branch Target Buffer) sind dabei die Adressen der Sprungbefehle und deren Verzweigungen gespeichert (Abbildung 8.18). Die „Branch History Table“ enthält für jeden ausgeführten bedingten Sprungbefehl einen Zähler, der die Anzahl ausgeführter Sprünge angibt. In Abhängigkeit dieses Wertes wird bei der Sprungvorhersage entweder „Sprung durchgeführt“ angenommen oder nicht. Durch die Vorhersage eines Sprunges lassen sich die Befehle der wahrschein-

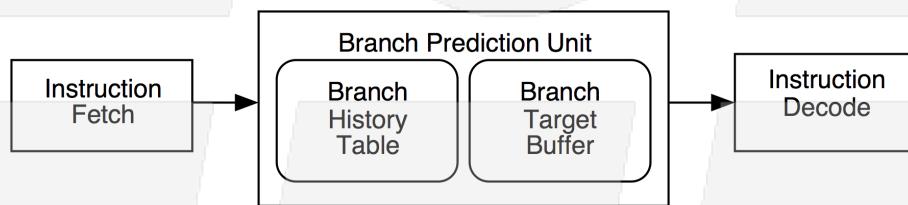


Abbildung 8.18: Prinzip der Sprungvorhersage

lich genommenen Verzweigung schon laden bevor das Ergebnis der Zielberechnung feststeht. Man nennt dies spekulative Ausführung („speculative execution“). Die Sprungvorhersageeinheit steuert die eigentliche Befehlsdecodierung so an, dass die Befehle des wahrscheinlich genommenen Sprungziels geladen und ausgeführt werden.

Wird das unwahrscheinlichere Sprungziel genommen, so müssen die bereits in der Instruktionspipeline befindlichen Operationen aus dieser gelöscht, bzw. deren Ergebnisse verworfen werden.

Ziel dieses Verfahrens ist, ähnlich wie bei der Superskalarität (vgl. 8.8), eine möglichst gute Auslastung der Prozessorpipelines.

8.6.4 Parallelrechner-Architekturen

Bei *Parallelrechnern* unterscheidet man zwischen SIMD- und MIMD-Rechnern.

In einem *SIMD*-Rechner (Single instruction – Multiple Data) kann eine Operation für mehrere Operanden gleichzeitig ausgeführt werden. Dementsprechend müssen mehrere Rechenwerke vorhanden sein. Der Hauptspeicher ist in der Regel aus mehreren Speichermodulen aufgebaut. Damit die Rechenwerke mit Daten versorgt werden können, verbindet ein Netzwerk jede ALU mit jedem Speichermodul. Abbildung 8.19(links) zeigt den prinzipiellen Aufbau eines SIMD-Rechners.

SIMD-Rechner sind besonders zur Durchführung von Operationen aus der linearen Algebra geeignet. So kann z.B. eine Vektoraddition in nur einem Schritt ausgeführt werden, wenn die Vektorlänge die Anzahl der Rechenwerke nicht überschreitet.

In einem *MIMD*-Rechner (Multiple Instruction – Multiple Data) können mehrere, im Allgemeinen verschiedene Befehle gleichzeitig für ebenfalls verschiedene Operanden ausgeführt werden. Deshalb existiert für jede ALU eine eigene Control Unit (CU, dt.: Steuerwerk). Ein MIMD-Rechner verfügt also über mehrere unabhängig voneinander arbeitende Prozessoren.

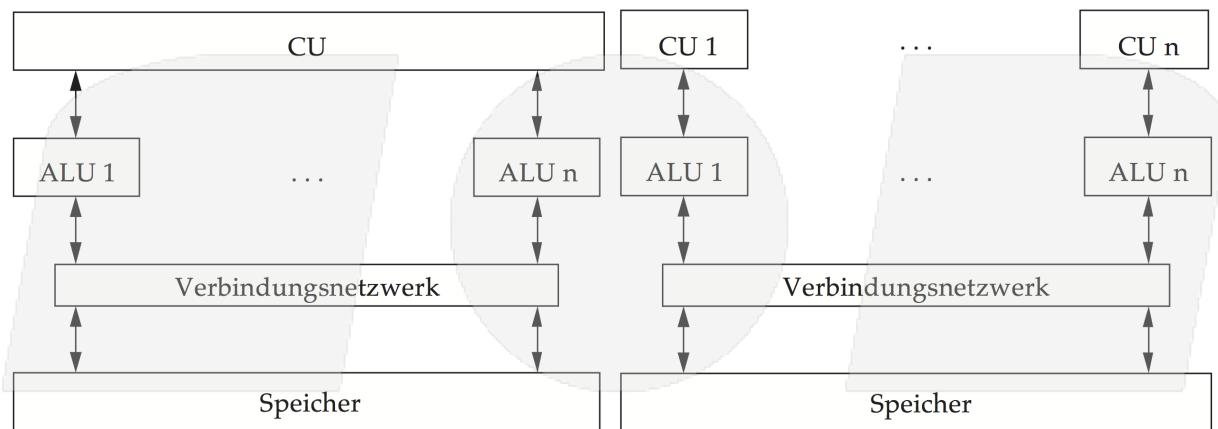


Abbildung 8.19: Aufbau eines SIMD-Rechners (links) und eines MIMD-Rechners (rechts)

Abbildung 8.19 (rechts) zeigt einen MIMD-Rechner mit globalem Speicher, auf den alle Prozessoren über ein Verbindungsnetzwerk zugreifen können. Eine Variante dieses Konzepts verwendet ausschließlich lokale Speicher, d.h. jeder Prozessor besitzt einen eigenen Speicher, auf den nur er zugreifen kann. Das Netzwerk verbindet dann nicht die Prozessoren mit dem Speicher, sondern die Prozessoren untereinander. In der Literatur findet man diese Variante unter dem Namen „lose gekoppelte Prozessoren“, „verteiltes System“ oder auch „Multicomputer“.

8.6.5 Rechenwerke mit Vektoreinheit

Eine *Vektoreinheit* verfügt über (arithmetische) Pipelines zur Durchführung von Gleitkomma-Operationen (Addition, Multiplikation). Das Prinzip ist mit dem des Instruktion-Pipelining (vgl. Abschnitt 8.6.2) identisch.

Mit Hilfe von arithmetischen Pipelines können Operationen auf Vektoren entscheidend beschleunigt werden. Rechner, die über eine Vektoreinheit verfügen, werden *Vektorrechner* (Abb. 8.20) genannt. Die in den Hochleistungsrechenzentren installierten so genannten *Supercomputer* sind in der Regel Vektorrechner, deren hohe Leistungsfähigkeit im wesentlichen auf dem Prinzip des arithmetischen Pipelining beruht.

Da die Durchführung von arithmetischen Operationen auf Skalaren mit Hilfe von Pipelines ineffizient ist, verfügt das Rechenwerk neben der Vektoreinheit auch über eine Skalareinheit. Das Steuerwerk (CU) erkennt bei der Befehlsdecodierung, ob eine Skalar- oder Vektoroperation durchzuführen ist und steuert die entsprechende Einheit an.

Aufgrund sinkender Kosten für MIMD-Parallelrechner verlieren die klassischen Vektorrechner mehr und mehr an Bedeutung.

8.7 Gleitkomma-Koprozessoren

Sollen Gleitkommaoperationen mit einer für ganze Zahlen oder Festkommazahlen ausgelegten ALU durchgeführt werden, so ist es erforderlich, die einzelnen Schritte der Operationen nacheinander auszuführen und durch Software zu steuern, z. B. durch ein Maschinensprache- oder

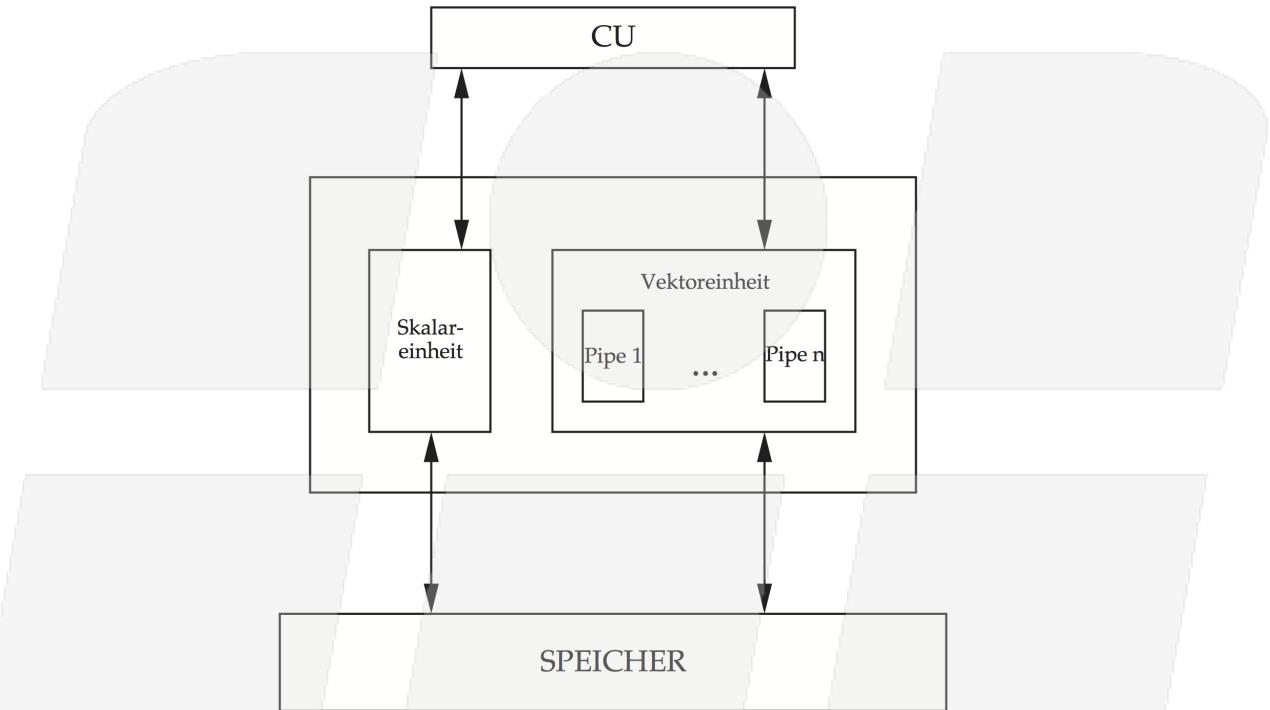


Abbildung 8.20: Prinzipieller Aufbau eines Vektorrechners

Mikroprogramm. Um dies zu beschleunigen, wird der Zentralprozessor um Funktionseinheiten zur Gleitkommaverarbeitung erweitert. Dazu gehören eine Gleitkomma-Arithmetikeinheit, Gleitkommaregister sowie die entsprechenden Steuer- und Statusregister.

Beispiele für (alte) Gleitkommaprozessoren sind der Intel 80x87, Motorola MC 68881 und der MC 68882.

Die Mantissen- und Exponentenoperationen können nacheinander oder parallel in Festkommaprozessoren durchgeführt werden. (Abb. 8.21)

Bei modernen Prozessoren werden die Einheiten zur Verarbeitung von Gleitkomma-Zahlen in den Prozessorkern integriert.

8.8 Superskalarität

Mit *Superskalarität* bezeichnet man die gleichzeitige Ausführung zweier oder mehrerer Instruktionen durch zwei oder mehrere Ausführungseinheiten (Abbildung 8.22). Nach der Decodierung der Instruktion (Instruction Decode) übernimmt die entsprechende Einheit die Ausführung der Instruktion. Die Superskalarität ist daher eng mit dem Aufbau der Prozessorpipelines verknüpft.

Da die Ausführung der verschiedenen Instruktionen keine konstante Anzahl von Taktzyklen beansprucht, kann es vorkommen, dass die Instruktionen in der falschen Reihenfolge beendet werden, z.B. ein Fließkommabefehl der nach einer Festpunktinstruktion im Programm steht, vor dieser beendet wird. Daher ist den parallel arbeitenden Einheiten noch eine weitere nachgeschaltet, die die Ergebnisse wieder in die richtige Reihenfolge bringt (Instruction Reorder).

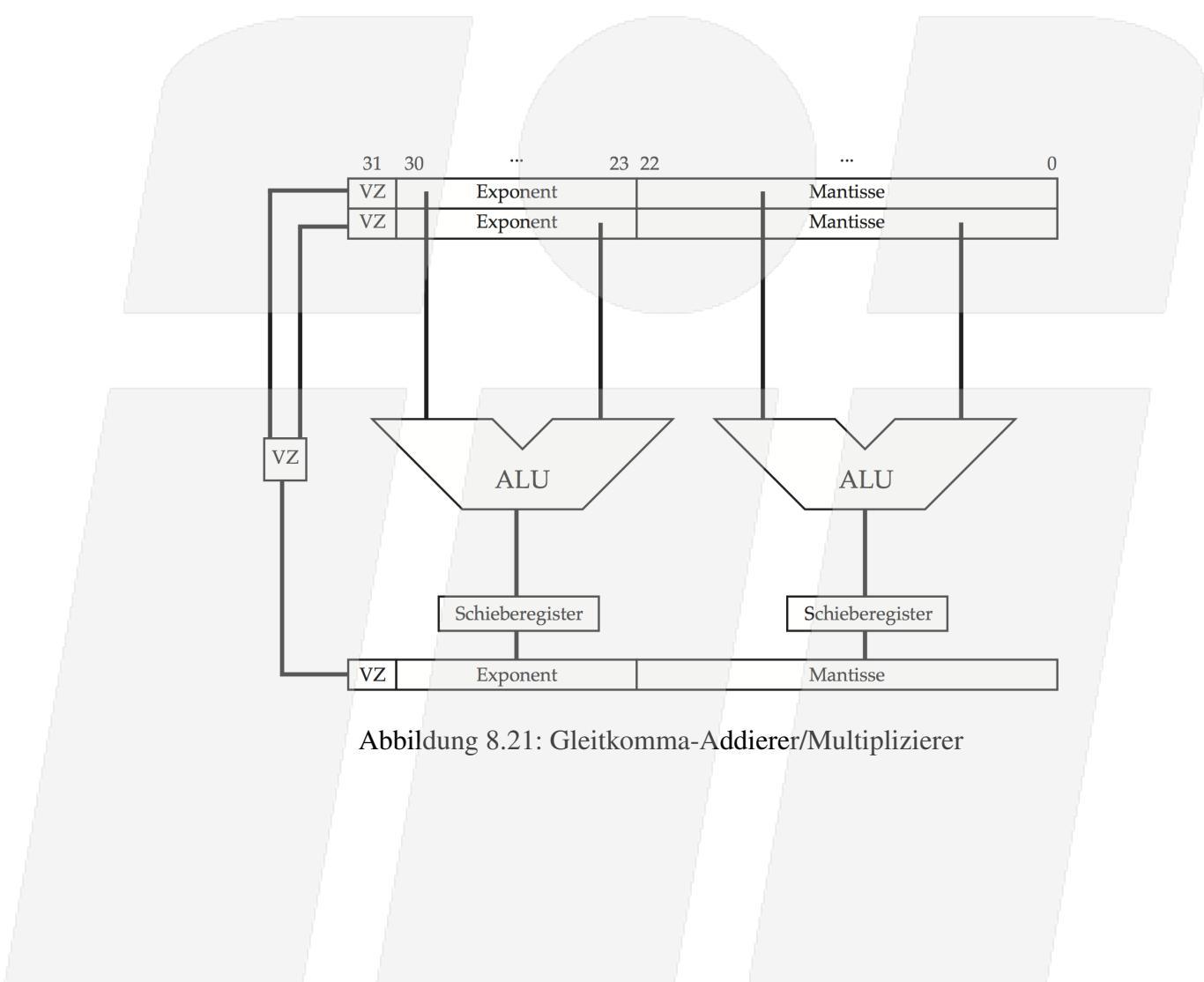


Abbildung 8.21: Gleitkomma-Addierer/Multiplizierer

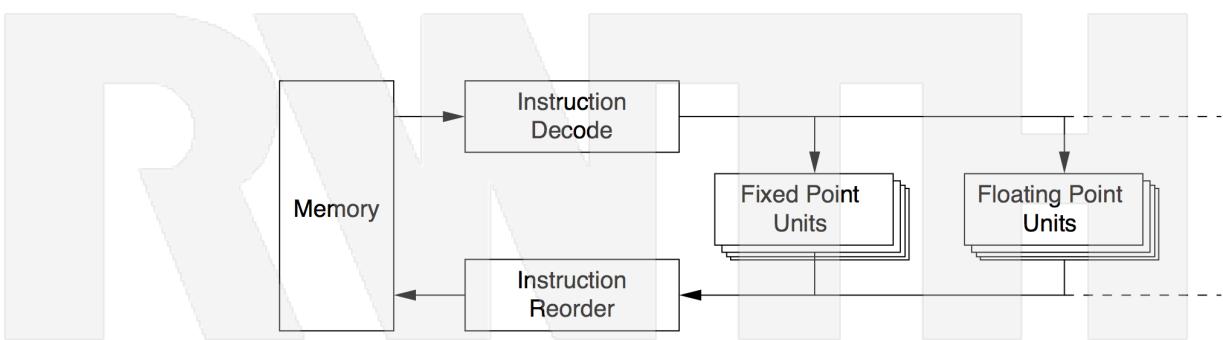


Abbildung 8.22: Prinzip der Superskalaren Architektur

Eine Weiterentwicklung des Instruction Reorder ist die so genannte „*Out of Order Execution*“, bei der die aus dem Speicher geladenen Instruktionen während der Decodierung so umgeordnet werden, dass die parallel arbeitenden Einheiten möglichst ausgelastet sind.

8.9 Register Renaming

Während der spekulativen Ausführung und ‘Out Of Order Execution’ von Befehlen kommt es vor, dass Ergebnisse in ein und dasselbe Register geschrieben werden sollen. Um dies zu ermöglichen, existieren bei Prozessoren die diese Verfahren unterstützen so genannte Register Files, die eine Sammlung von Registern darstellen, welche die Ergebnisse von Operationen aufnehmen. Die einzelnen Register besitzen keine festen Namen, sondern bekommen logische Namen, die denen der im Assembler verwendeten Register entsprechen, zugeteilt. So kann im Verlauf einer ‘Out Of Order’ Ausführung ein logisches Register mehrfach im Register File vorhanden sein, weil beispielsweise mehrere Additionen ihr Ergebnis in das Register *R0* schreiben wollen. Abbildung 8.23 zeigt den Vorgang des Register Renamings an einem Beispiel.

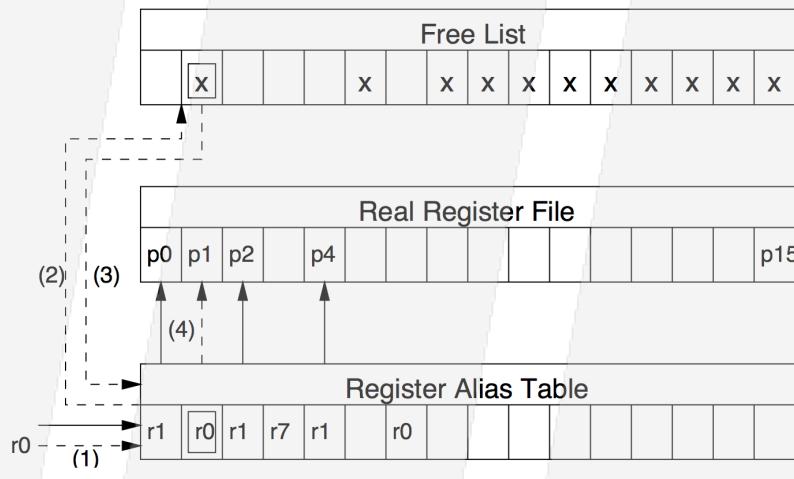


Abbildung 8.23: Register Renaming

Die Logischen Register *r0*, *r1* und *r7* sind bereits auf physikalische Register abgebildet. In der Register Alias Table wird die Zuordnung von logischen zu physikalischen Registern verwaltet. Im dargestellten Zustand (vor dem mit (1) bezeichneten Schritt) sind beispielsweise drei Operationen aktiv, die das Register *r1* benutzen. Das Register *r1* ist daher über die Register Alias Table drei physikalischen Registern (*p0*, *p2*, *p4*) zugeordnet.

Nun kommt eine Operation zur Ausführung, die das Register *r0* benötigt (Schritt (1)). Die Register Alias Table fordert nun aus der Free List, in der die freien physikalischen Register verwaltet werden (im Bild mit einem *x* gekennzeichnet), ein physikalisches Register an (Schritt (2)). Aus der Free List wird das nächste freie Register entnommen und in die Register Alias Table eingetragen (Schritt (3)). Nun kann der Zugriff auf das Register erfolgen (Schritt (4)). In der Register Alias Table wird außerdem vermerkt, welcher Instruktion die Register jeweils zugeordnet sind.

Ist eine Instruktion vollständig ausgeführt, so wird der Eintrag in der Register Alias Table gelöscht und in der Free List das entsprechende physikalische Register wieder als unbenutzt markiert.

Die Anzahl gleichzeitig bearbeitbarer Instruktionen ist somit von der Tiefe der Instruktionspipelines, der Anzahl der vorhandenen Ausführungseinheiten sowie der Größe des Register Files abhängig.

8.10 CISC- versus RISC-Maschinen

Zunächst wurden immer komplexere Befehlssätze unter Verwendung der Mikroprogrammierung entwickelt (*CISC, Complex Instruction Set Computer*). Ziel waren dabei die bessere Nutzung der Prozessorleistung und einfachere Compiler. Ein extremes Beispiel ist der Intel iAPX-432 mit 222 Maschinenbefehlen, die zwischen 6 und 321 Bits lang sind, und einem 64 KBitgroßen Mikroprogrammspeicher.

Statistisch werden jedoch nur wenige Befehle tatsächlich häufig verwendet. Untersuchungen am IBM 360-Rechner haben z.B. ergeben, dass 99% des Maschinencodes aus nur 30 verschiedenen Befehlen besteht, obwohl der Rechner über einen Befehlsvorrat von 200 Befehlen verfügt. Wählt man aus diesen 30 die 10 häufigsten aus, so machen diese immerhin noch 80% des Maschinencodes aus. Aus diesem Grund wurden einfache Prozessoren mit elementarem Befehlssatz entwickelt (*RISC, Reduced Instruction Set Computer*).

Zur Einordnung eines bestimmten Rechners als RISC oder CISC darf nicht nur die Anzahl der Maschinenbefehle berücksichtigt werden. Es handelt sich eindeutig um eine *RISC*-Architektur, wenn 5 der folgenden 8 Kriterien erfüllt sind:

- weniger als 50 Maschinenbefehle
- weniger als 4 Adressierungsarten
- weniger als 4 Befehlsformate
- Speicherzugriff nur über LOAD/STORE-Befehle, d.h. alle Operationen außer dem Laden und Speichern von Befehlen und Daten finden nur in Prozessorregistern statt.
- mehr als 32 Prozessorregister
- fest verdrahtete Maschinenbefehle (keine Mikroprogrammierung)
- Befehlausführung (meist) in einem Taktzyklus
- Verfügbarkeit optimierender Compiler für höhere Programmiersprachen

Tabelle 8.5: Beispiele für CISC- und RISC-Prozessoren

	RISC	CISC
Beispiele	Intel i860 PowerPC SPARC Alpha	Intel i80x86 Motorola 680x0
Befehlsumfang	50-100	200-400
Zyklen pro Befehl	1,2-1,7	4-10

Um einen Leistungsvergleich zwischen CISC- und RISC-Prozessoren durchzuführen, reicht es nicht aus, MIPS (Millionen Instruktionen pro Sekunde) als Maßzahl heranzuziehen. Da die Ausführung eines CISC-Befehls i.a. länger dauert als die eines RISC-Befehls (vgl. Tabelle 8.5),

schniedet ein RISC-Prozessor beim MIPS-Vergleich wesentlich besser ab. Man muss jedoch berücksichtigen, dass ein CISC-Programm in der Regel kürzer ist als ein RISC-Programm für dieselbe Aufgabe. Es bietet sich deshalb folgende Formel zur Berechnung der Leistung P eines Prozessors an:

$$P = \frac{\tau}{\omega \cdot k}$$

mit τ = Taktrate, ω = Taktzyklen pro Befehl und k = Zahl der benötigten Befehle für eine bestimmte, möglichst repräsentative Testaufgabe. Untersuchungen haben ergeben, dass ein RISC-Prozessor 20 - 40% mehr Befehle benötigt als ein CISC-Prozessor, aber dafür 3 - 6 mal schneller rechnet, so dass sich eine relative Leistungssteigerung von 1,7 bis 6,5 ergibt. Tabelle 8.6 zeigt die Untersuchungsergebnisse für drei konkrete Prozessoren.

Tabelle 8.6: CISC- und RISC-Prozessoren im Vergleich

Prozessor	k	$\tau[\text{MHz}]$	ω	P
MC 68030 (CISC)	1,0	33	5,2	6,34
i80386 (CISC)	1,1	33	4,4	6,82
SPARC (RISC)	1,2	33	1,2	21,15

8.11 VLIW-Prozessoren

Das Gegenstück zu den RISC-Prozessoren bilden die *VLIW* (*Very Long Instruction Word*) Prozessoren. Sie haben, wie die RISC-Prozessoren auch, Befehlsworte fester Länge, die aber im Gegensatz zu den RISC Befehlen sehr lang sind. Der 1985 entwickelte Multiflow Trace 28/200 Chip besaß Instruktionsworte von 1024 bit Länge.

Das Ziel ist dabei, mehrere Operationen und Operanden so zu gruppieren, dass stets alle verfügbaren Verarbeitungseinheiten der CPU aktiv sind. Die Länge des Instruktionswortes hängt also direkt mit der Anzahl der parallel arbeitenden Einheiten der CPU (z. B. mehrere Fixed Point Units) zusammen.

Der Vorteil dieser Architektur gegenüber den RISC Prozessoren liegt darin, dass sich die Steuerwerke der Prozessoren wesentlich einfacher gestalten lassen. Bei RISC Prozessoren muss das Steuerwerk die Befehle umordnen (Abschnitt 8.8), um möglichst viele Verarbeitungseinheiten der CPU zu beschäftigen. Bei VLIW steht die Reihenfolge der Befehle schon vor dem Programmstart fest. Die Festlegung der Instruktionsworte ist hier Aufgabe des Assemblers (Kapitel 9), bzw. des Compilers.

Beispiele für VLIW Prozessoren sind: Die TI C6xx von Texas Instruments, der Phillips TriMedia Prozessor und der Itanium von Intel und Hewlett Packard.

8.12 Multithreading-Architekturen

Multithreading ist ein Konzept, das heute zu den Standard-Techniken in der Softwareentwicklung gehört. Ein *Thread* (Faden, Strang) ist ein unabhängiger Kontrollfluss innerhalb eines Programms. Der Unterschied zwischen einem Prozess und einem Thread liegt darin, dass Prozesse

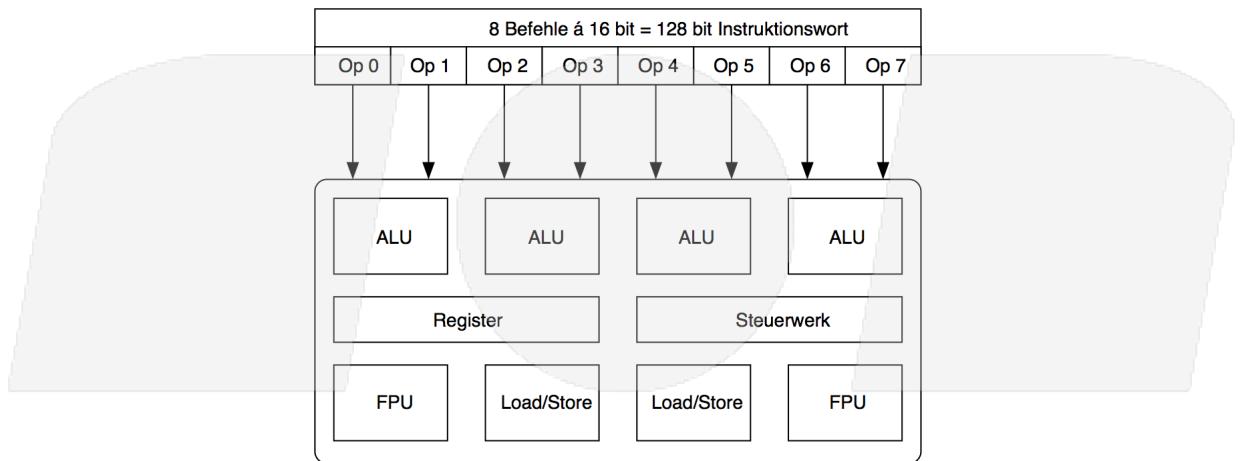


Abbildung 8.24: Very Long Instruction Word Prozessor. Eine Instruktion enthält mehrere Befehle, die auf die verfügbaren Verarbeitungseinheiten der CPU verteilt werden.

jeweils eigene Adressräume besitzen während Threads (als Teil eines Prozesses) parallel innerhalb des gleichen Adressraums ablaufen. Das Betriebssystem kann daher sehr viel schneller zwischen verschiedenen Threads als zwischen verschiedenen Prozessen umschalten.

Die Thread-Technik lässt sich im Prinzip auf allen Mikroprozessoren implementieren. Allerdings ist die hardwaremäßige Unterstützung dieses Konzepts von besonderem Interesse, da ein Prozessor immer dann, wenn ein Thread warten muss (z. B. wegen Synchronisationsbefehlen), hardwaremäßig auf einen anderen Thread umschalten kann.

Eine Multithreading-CPU besitzt mehrere Rechenwerke, aber nur ein Steuerwerk. Der Unterschied zu superskalaren Architekturen besteht im wesentlichen darin, dass die parallel ausgeführten Befehle unterschiedlichen Kontrollflüssen entstammen. Deswegen muss das Steuerwerk die Zustände aller bearbeiteten Threads kennen. Weiterhin müssen alle Register sowie der Befehlszähler mehrfach vorhanden sein.

Grundsätzlich lassen sich drei Ansätze unterscheiden:

- Bei *Blocked Multithreading* stellt der Prozessor den Threads zusätzliche Register zur Verfügung, jedoch keine zusätzlichen Funktionseinheiten. Es wird immer nur ein Thread gleichzeitig ausgeführt. Muss ein Thread anhalten, wird auf einen anderen umgeschaltet.
- *Interleaved Multithreading* wählt in jedem Taktzyklus einen Befehl aus einem ausführbereiten Thread aus. Muss ein Befehl warten, wird der Thread aus der Liste der ausführbereiten Threads ausgebündet.
- *Simultaneous Multithreading* ist die heute favorisierte Variante, die interleaved Multithreading mit superskalaren Architekturen kombiniert. Es werden also pro Taktzyklus mehrere, aus unterschiedlichen Threads stammende, Befehle ausgeführt.

Der Vorteil gegenüber normalen Superskalaren-Architekturen liegt darin, dass durch die Unabhängigkeit der Threads wesentlich weniger Abhängigkeiten zwischen Daten auftreten, die in unterschiedlichen Funktionseinheiten verwendet werden.

Ein früher Computer mit Multithreading-Architektur war der Denelcor HEP (1982), dem ersten kommerziell vertriebenen (aber nicht erfolgreichen) MIMD-Rechner. Alle großen Prozessor-

Hersteller entwickeln mittlerweile Multithreading-Prozessoren. Intel bietet ab dem Pentium4 sowie dem Itanium das simultaneous Multithreading mit dem Namen „*Hyperthreading*“, welches die Bearbeitung von zwei Threads gleichzeitig unterstützt. Während Hyperthreading auf einem CPU-Kern basiert, besitzen aktuelle CPUs wie der Intel Core2 Duo oder AMD Athlon64X2 zwei Prozessorkerne und gewährleisten so echtes Multithreading. Durch die Kopplung von Doppelkernprozessor und Hyperthreading können bis zu vier Threads gleichzeitig bearbeitet werden, wie es bei der Intel Xeon 7000er-Reihe sowie der Pentium Extreme Edition 840 und 955 der Fall ist. Sun stellt mit den UltraSPARC T1 einen Prozessor vor, der über bis zu acht Kerne mit vierfach Multithreading verfügt, und so bis zu 32 Threads gleichzeitig verarbeiten kann.



Kapitel 9

Maschinensprache und Assembler

Maschinensprache ist charakterisiert durch Befehlsformat und -umfang. Sie ist hardwareabhängig, wenn auch weitgehend konsistent innerhalb einer Rechnerfamilie (z.B. Atmel AVR; IBM 360/370/390; Intel 8080, 80x86; Motorola 680x0).

Im von *Neumann-Rechner* befinden sich *Maschinenbefehle* zusammen mit den zur Programmausführung benötigten Daten im Hauptspeicher, codiert als 0-1-Folgen. Während der Programmausführung werden die Maschinenbefehle in das *Befehlsregister* der CPU geladen und von der *Steuereinheit* (Control Unit) interpretiert.

Im Vergleich dazu existiert die *Harvard-Architektur*. Bei ihr sind *Programmspeicher* sowie *Datenspeicher* strikt voneinander getrennt. Dies hat den Vorteil, dass innerhalb eines einzigen Taktzyklus Befehle und ihre dazugehörigen Daten aus separaten Speichern gelesen werden können. Die ursprüngliche von Neumann-Architektur benötigt für die gleiche Operation zwei Taktzyklen.

Ein weiterer Vorteil dieser Architektur ist, dass die *Wortbreite* für Daten und Befehle nicht mehr einheitlich sein muss. Dies ermöglicht eine effizientere Gestaltung des Befehlssatzes.

Von besonderem Interesse ist diese Architektur daher für *Embedded Systems* und *Mikrocontroller*.

Bei dem im folgenden betrachteten und den Assemblerbeispielen zu Grunde gelegten Mikrocontroller handelt es sich um einen Vertreter der Harvard-Architektur, und zwar um den *Atmel AVRmega8*.

Grundsätzlich besteht ein Maschinenprogramm aus einer Folge von Zahlen in dualer oder bestenfalls hexadezimaler Darstellung und ist deshalb nur schwer zu lesen oder gar zu erstellen. Aus diesem Grund wird statt der reinen Maschinensprache in der Regel die sog. Assemblersprache verwendet. Die *Assemblersprache* verbessert die Lesbarkeit von Maschinenprogrammen durch eine mnemotechnische Darstellung des BefehlsCodes und die Möglichkeit, Symbole für Speicheradressen und Daten zu verwenden. Vor der Ausführung muss ein Assemblerprogramm natürlich in ein reines Maschinenprogramm umgewandelt werden. Das hierzu benötigte Übersetzungsprogramm wird Assembler genannt. Man beachte, dass der Begriff *Assembler* auch häufig als Kürzel für die Assemblersprache selbst benutzt wird.

9.1 Zugrunde gelegte Hardware

Die in diesem Kapitel betrachteten Befehle und Strukturen beziehen sich auf den Mikrocontroller AVRmega8 der Firma Atmel und dessen Assembler. Hierbei handelt es sich um einen *8-Bit-RISC-Prozessor* mit geringer Energieaufnahme.

Die technischen Daten im Überblick:

- 130 Assemblerbefehle
- 32 8-Bit-Register
- 8k interner Speicher
- 512 Bytes EEPROM
- 1k SRAM
- zwei 8-Bit-Timer / Counter
- ein 16-Bit-Timer / Counter
- sechs Kanal 10-Bit-A/D-Wandler (Analog / Digital \rightarrow Wandler)
- Echtzeitzähler mit separatem Oszilator
- drei PWM-Kanäle
- analog Comparator
- serielles Interface

Die einzelnen Komponenten können im folgenden Blockdiagramm (siehe Abbildung 9.2) identifiziert werden.

Der 8kB interne Speicher ist als *SRAM* realisiert und interagiert unter anderem mit dem *8-bit Datenbus*.

Das *Steuerwerk (Instruction Decoder)* beinhaltet das Leitwerk des Mikroprozessors. Hier werden die einzelnen Befehle des Programms dekodiert und, entsprechend des Befehls, die Operanden geladen, auf die der Befehl angewandt werden soll.

Die arithmetisch-logische Einheit (*ALU*) stellt das *Rechenwerk* dar und dient beispielsweise der Addition.

Um eine Kommunikation mit externen und internen Komponenten zu ermöglichen, verfügt der Mikroprozessor über eine Vielfalt von *Ein-* und *Ausgabeports*. Diese sind im Blockdiagramm unter anderem mit PORTB, PORTC und PORTD bezeichnet.

Beim *Analog-Digitalwandler* (ADC) handelt es sich um einen Konverter, der das analoge Eingangssignal (eine Spannung) in ein digitales wandelt, welches dann vom Mikrocontroller weiterverarbeitet werden kann.

Der *Stack-Pointer* stellt ein Hilfsregister dar. In ihm ist der Zeiger auf den zuletzt belegten Speicher des *Stacks* (Stapel) abgelegt. (Eine ausführliche Erläuterung des Stacks wird im Abschnitt 9.2.3 gegeben.)

Der interne *Timer* (TIMER/COUNTER) ist ein integrierter Schaltkreis des Chips, der beim

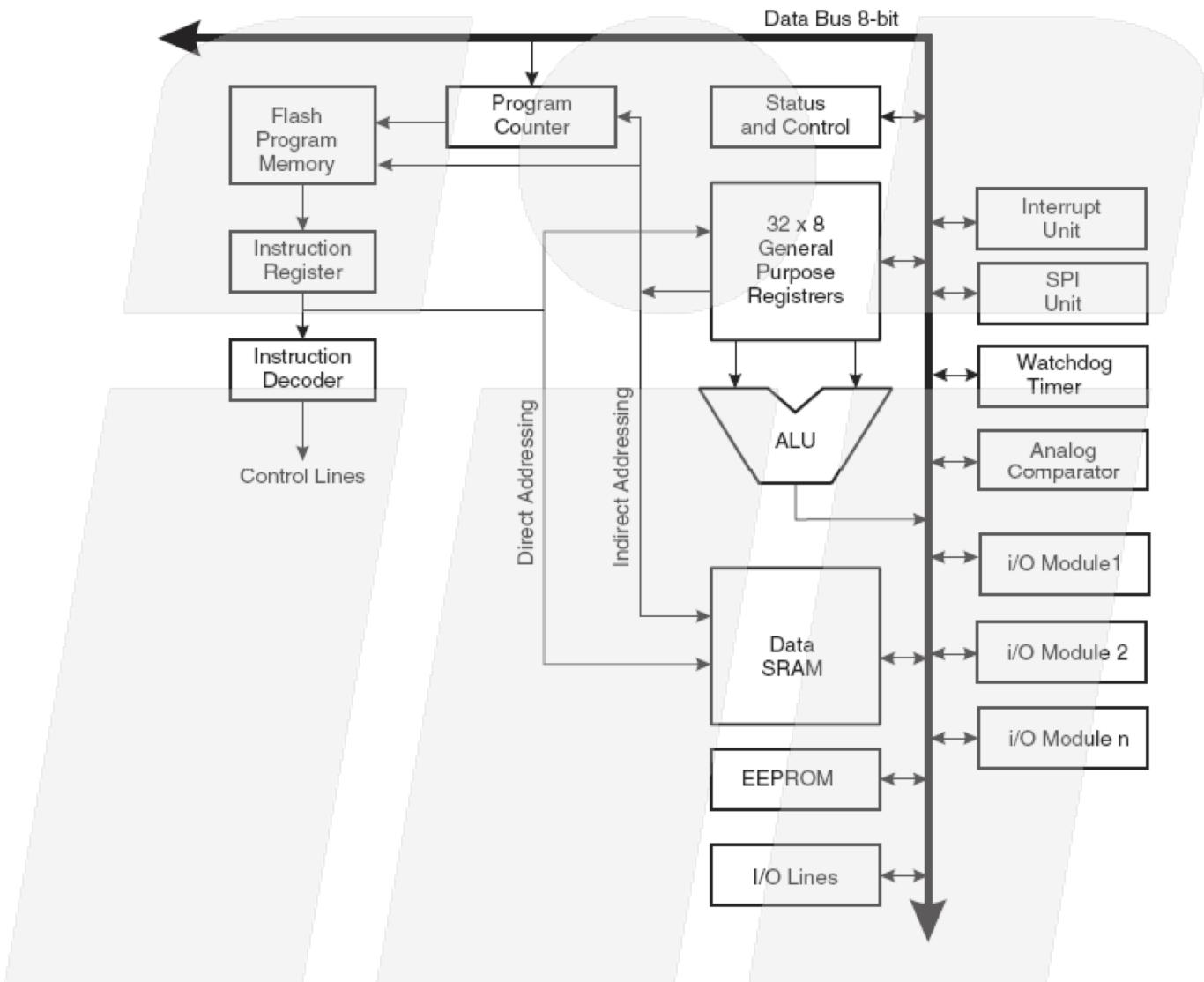


Abbildung 9.1: Die AVR-Mikrocontroller-Architektur im Überblick

Messen von Zeitabständen, beim wiederkehrenden Ausführen von Programmteilen oder beim Zählen von Ereignissen zum Einsatz kommt. Der *Universal Asynchronous Receiver Transmitter* (kurz UART) versieht den seriellen digitalen Datenstrom mit einem fixen Rahmen. Dazu zählt unter anderem ein Start- und ein Stopp-Bit.

Beim *Inter-Integrated Circuit* (I^2C) handelt es sich um einen seriellen 2-Draht-Datenbus zur Kommunikation digitaler Schaltkreise. Ein weiteres Bussystem stellt das *Serial Peripheral Interface* (SPI) dar.

Der *Komparator* (COMPARATOR) ermöglicht den Vergleich zweier analoger Werte auf Identität.

Alle oben genannten Komponenten können im folgenden Blockdiagramm 9.2 identifiziert werden.

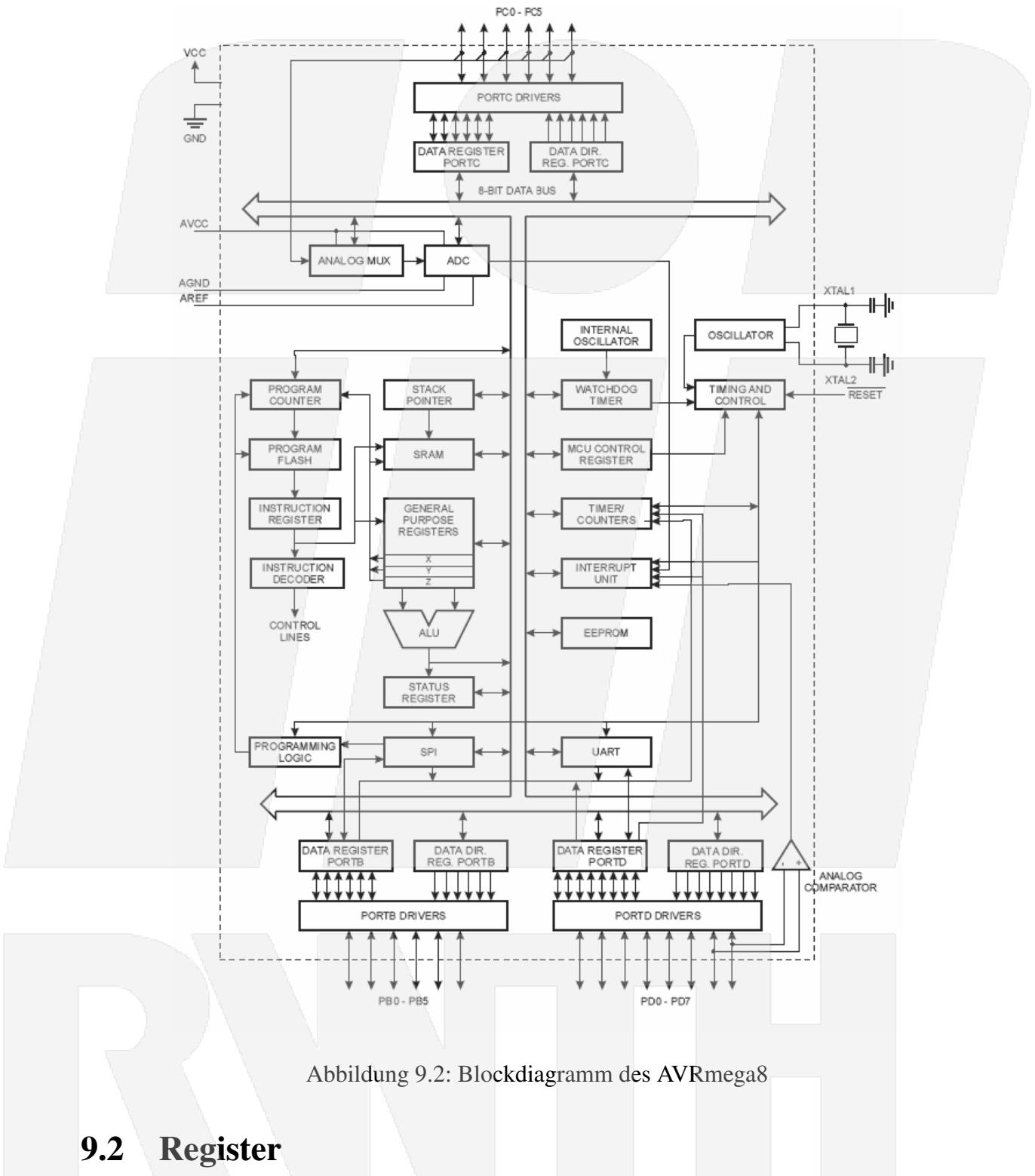


Abbildung 9.2: Blockdiagramm des AVRmega8

9.2 Register

Als Register bezeichnet man zu Gruppen zusammengefasste Speicherzellen innerhalb des Speichers eines Prozessors. Dabei stellen die Register die schnellste Art der *Datenmanipulation* dar. Gleichzeitig können Informationen dort aber nur vorübergehend gespeichert werden. Register werden benötigt, da eine Vielzahl von Befehlen nur unter Zuhilfenahme dieser ausgeführt werden können.

Der Aufbau, die Größe und die Art der Register sind dabei abhängig vom eingesetzten Prozessortyp. Im folgenden soll anhand der realen Hardware des Atmel AVRmega8 der Aufbau der verschiedenen Register näher gebracht werden.

9.2.1 Das Statusregister

Als *Statusregister* wird ein Register im *Steuerwerk* bezeichnet. Über die dort gesetzten Bits sind Rückschlüsse auf die letzten Rechenoperationen der *ALU* möglich. So kann zum Beispiel geprüft werden, ob ein Ergebnis gleich 0 war. In diesem Fall ist das Zero Flag gesetzt. Dies kann dann wiederum Einfluß auf den Ablauf des Programms haben (siehe hierzu das Beispiel aus Abschnitt 9.4.7).

Da die einzelnen Bits logisch voneinander unabhängig sind, können sie in einem Register zusammen gefasst werden.

Der Aufbau des Statusregisters des Atmel AVRmega8 ist dabei wie folgt:

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 9.3: Statusregister des Atmelmega8

Dabei handelt es sich um die in Tabelle 9.1 aufgelisteten Bits bzw. Flags:

C	Carry Flag (Übertragsbit)
Z	Zero Flag (Ergebnis war 0)
N	Negative Flag (negatives Ergebnis der letzten Operation)
V	Two's Complement Flag
S	Sign-Bit ($S=N \oplus V$)
H	Half Carry
T	Bit Copy Storage (dient als Speicher für einzelne Bits)
I	Global Interrupt enable (steuert die Unterbrechungsverarbeitung)

Tabelle 9.1: Bits bzw. Flags des Statusregisters des Atmelmega8

9.2.2 General Purpose Working Register

Wie bereits einleitend erwähnt, verfügt der Mikrocontroller AVRmega8 über 32 8-Bit-*Register* zur internen Speicherung von Daten. Diese werden als Register R0 bis R31 bezeichnet. Um im Einzelfall auf ein 16-Bit-Register zurückgreifen zu können, sind sechs der 8-Bit-Register zusammenfassbar zu drei 16-bit-Registern.

Dies veranschaulichen die Abbildungen 9.4 und 9.5.

Dabei werden die Register R26 bis R31 zu 16-bit-Registern zusammengezogen. Bezeichnet werden sie anschließend als *X-, Y- und Z-Register* und setzen sich aus einem hohen und einem niedrigen Byte zusammen.



Abbildung 9.4: Register des Atmel AVRmega8

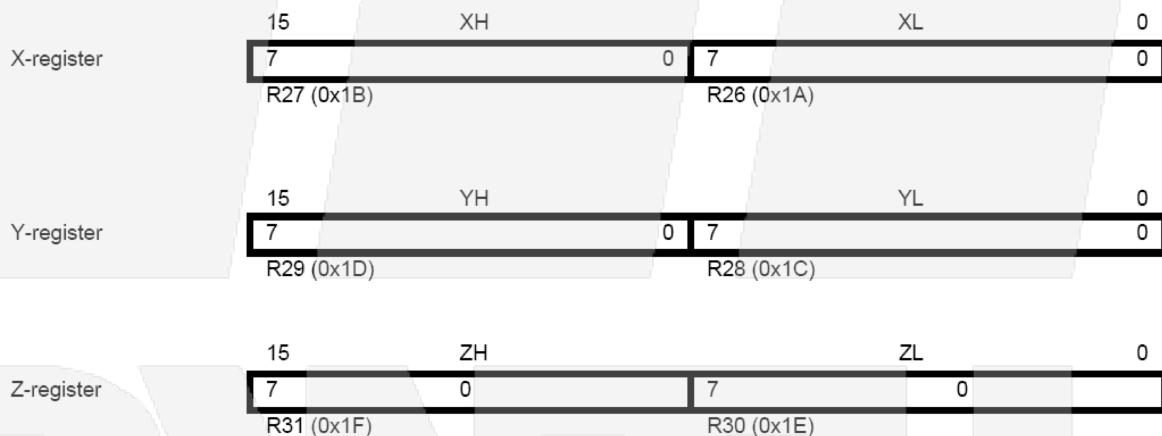


Abbildung 9.5: Die drei 16-bit-Register des Atmel AVRmega8 mit hohem und niedrigen Byte

9.2.3 Stack-Pointer

Der *Kellerspeicher* (auch: *Stack*, *Keller* oder *Stapel* bezeichnet) ist ein spezieller Speicher oder Speicherbereich, der nach dem *LIFO*-Prinzip (*Last In, First Out*) aufgebaut ist: Eine Leseoperation (*POP*) bezieht sich immer auf das sog. oberste Kellerelement, d.h. auf denjenigen Wert, der zuletzt (mit dem *PUSH*-Befehl) in diesen Speicher geschrieben wurde. Bei der Durchführung des *POP*-Befehls wird der gelesene Wert automatisch aus dem Keller entfernt, so dass der vorletzte in den *Keller* geschriebene Wert zum obersten Kellerelement wird.

Der Stack wird dabei zum Beispiel genutzt, um beim Aufruf einer *Subroutine* die *Rücksprungadresse* sichern zu können. Ohne einen Stack könnte keine Unterfunktionen aufgerufen und

anschließend aus dieser zurückgekehrt werden.

Darüber hinaus sind auf dem Stack auch temporäre Daten abgelegt.

Um mit dem Stack arbeiten zu können, benötigt man den so genannten *Stack-Pointer*, ein Zeiger der auf die Stapsel spitze und somit auf das zuletzt hinzugefügte Element zeigt. Von diesem ausgehend kann man dem Stapel ein Element hinzufügen (PUSH) oder eines herunternehmen (POP).

Bit	15	14	13	12	11	10	9	8	SPH
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPL
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Abbildung 9.6: Der Stack-Pointer

Da der Mikrocontroller AVRmega8 über einen Speicher größer 256 Byte verfügt, reicht ein Register zur Adressierung nicht mehr aus. Hier wird ein zweites hinzugefügt, welches mit SPH (für high) bezeichnet wird. Das erste ist entsprechend das niedrige Stack-Pointer-Register. Eine korrekte Initialisierung des Stackpointers erfolgt, indem jeweils das Lowbyte und das Hibyte von RAMEND geladen werden und in die entsprechenden Register geschrieben werden.

Abbildung 9.7 zeigt die korrekte Initialisierung des Stack-Pointers.

```
.include "m8def.inc"

LDI r16, LOW(RAMEND) ; Lowbyte von RAMEND laden
OUT SPL, r16           ; in Stackpointer schreiben
LDI r16, HIGH(RAMEND) ; Hibyte von RAMEND laden
OUT SPH, r16           ; in Stackpointer schreiben
```

Abbildung 9.7: Assemblercode zur Initialisierung des Stack-Pointers

9.3 Assembler

Assemblersprache stellt eine spezielle Programmiersprache dar, in welcher die *Maschinensprache* einer speziellen *Prozessorarchitektur* in einer für den Menschen lesbaren Form angezeigt wird. Dabei stellen die einzelnen Befehle der Assemblersprache *Symbole des Maschinencodes* dar. Die Abbildung zwischen Befehlsworten (*Mnemonic*) und Maschinencodes ist hierbei üblicherweise bijektiv.

Im Falle des Atmel-Assemblers ist dies aber nicht der Fall.

Da unterschiedliche Prozessoren unterschiedliche Befehlssätze unterstützen, ist bei Assembler nicht grundsätzlich die Möglichkeit gegeben, ein in einer Assemblersprache geschriebenes Programm für andere Prozessoren zu nutzen. Das heißt, dass für jede *Prozessorfamilie* ein eigenes

Assemblerprogramm geschrieben werden muss.

Assembler ist somit keine *Hochsprache*. Da es aber sehr nah an der *Maschinenebene* angesiedelt ist, ermöglicht es, die Arbeitsweise eines Prozessors besser zu verstehen und dient dem Erlernen effizienter Programmiermethoden.

9.3.1 Assemblerbefehle

Wie bereits zum Ende des letzten Abschnitts erwähnt, ist ein Assembler immer nur für einen Prozessor oder eine Prozessorfamilie gültig. Somit unterscheiden sich die einzelnen *Befehlsworte* für die einzelnen Prozessoren.

Exemplarisch soll nun anhand des Befehlssatzes des Atmel AVRmega8 eine Einführung in die Assemblerbefehle gegeben werden.

Die einzelnen Befehle des Assemblers des Atmel-Mikroprozessors sind hierbei 16-Bit lang. Die Position und Länge des *Codeanteils* unterscheidet sich und ist abhängig vom einzelnen Befehl. Dabei kann ein solcher maximal zwei *Parameter* enthalten.

Da ein Mikrocontroller auf einen dauerhaften Betrieb ausgelegt ist, verfügt er zwar über Befehle, die ihn in einen Warte- oder *Sleep-Modus* versetzen, doch über keine Befehle für ein *Programmende*.

Einen einführenden Überblick gibt Abbildung 9.8.

Maschinencode	Assembler
1110 1000 0000 0000	ldi r16, 128
0000 1101 0000 0000	add r16, r0
0001 1101 0001 0001	adc r17, r1
0010 0011 0000 0001	and r16, r17
1001 0101 0000 0101	asr r16
0000 0000 0000 0000	

Abbildung 9.8: Gegenüberstellung von Maschinencode und Assembler

9.3.2 Ablauf eines Assemblerprogramms

Ein Assemblerprogramm läuft nach dem folgenden Schema ab:

Der *Programmcounter* ist ein Zeiger, der auf den nächsten auszuführenden Befehl verweist. Zu Beginn des Programms steht dieser auf 0000.

Im Folgenden wird der Befehl gelesen (*fetch*). Im zweiten Schritt wird der Befehl ausgeführt (*execute*). Der Programmcounter wird anschließend um eins erhöht und zeigt auf den nächsten Befehl. Bei einem linearen Programm wäre dies der Befehl 0001.

Der nächste Befehl wird bereits zur *Ausführungszeit* des vorhergehenden Befehls gelesen. Hierdurch können Performancegewinne erzielt werden, doch ist es ebenso möglich, dass Verzögerungen auftreten, wenn bei einer Verzweigung der falsche Befehl im Voraus geladen wurde. In diesem Fall muss ein *Takt* eingefügt werden, um den fehlenden Befehl nachzuladen.

Abbildung 9.9 zeigt den Ablauf eines Programms mit Lade- und Ausführungszyklen.

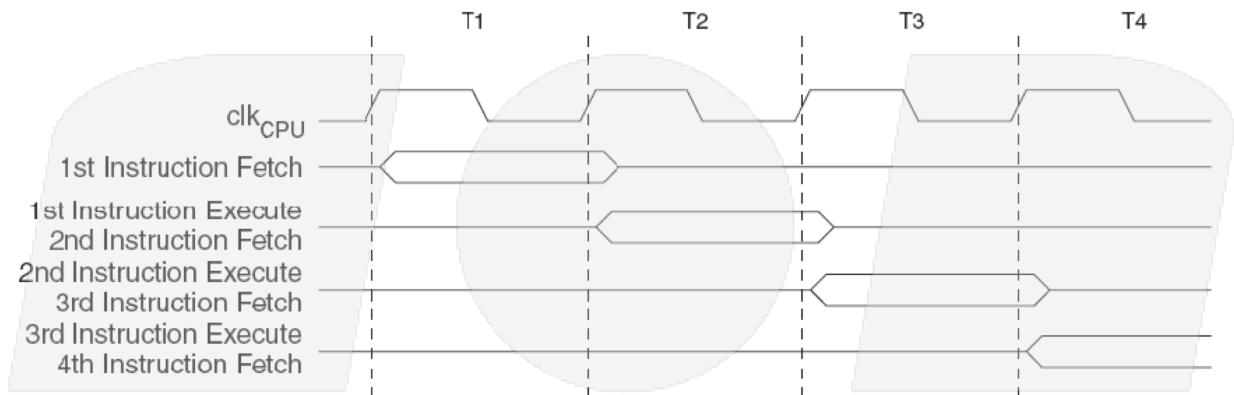


Abbildung 9.9: Taktzyklen eines Programms

Betrachtet man dabei einen Takt genauer, stellt man fest, dass innerhalb der Ausführungszeit eines solchen Befehls die *Register fetch*, ALU-Operation und write back sequentiell ausgeführt werden.

Zu Beginn werden die benötigten Operanden in die Register geladen, dann die *ALU-Berechnung* durchgeführt und anschließend das Ergebnis in das entsprechende Register zurückgeschrieben. (Dies ist abhängig vom verwendeten Befehl; siehe hierzu Abschnitt 9.4.) Die Abbildung 9.10 zeigt den Ablauf grafisch.

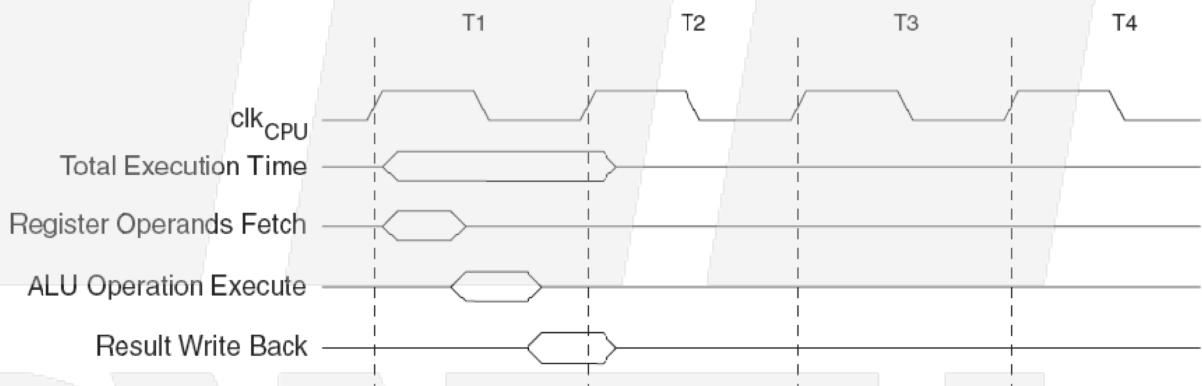


Abbildung 9.10: Sequentieller Ablauf von fetch, ALU-Operation und write back innerhalb eines Taktes

9.4 Untergliederung der Assemblerbefehle

In einem *Digitalrechner* sind verschiedene Arten von Maschinenbefehlen implementiert:

- *Datentransferbefehle*
Befehle zum Transport von Daten zwischen Speicherzellen des Hauptspeichers oder anderer Speicher (z.B. des sog. *Kellerspeichers*) und *Registern* der *CPU*.
- *Arithmetische Operationen*
Addition, Subtraktion, Multiplikation, Division.

- *Logische Operationen*

Bitweise logische Verknüpfung (z.B. *OR*, *AND*) von zwei Operanden

- *Vergleichsbefehle*

Testen des Inhalts zweier Register auf identischen Inhalt (*Compare*)

- *Bit-Befehle*

Setzen, Löschen und Lesen einzelner Bits im Akku sowie *bitweises Vergleichen*.

- *Status-Befehle*

Setzen, Löschen und Lesen einzelner Bits des Statuswortes.

- *Verzweigebefehle*

Beeinflussung des Programmablaufs durch bedingte und *unbedingte Sprünge* (Jump, Call, Skip).

- Befehle zur *Programmablaufsteuerung*

Befehle wie der Warte-Befehl gehören zu dieser Kategorie.

- Befehle zur *Unterbrechungsverarbeitung*

Verarbeitung von Unterbrechungen durch innere Ereignisse (programmierte Unterbrechung) oder äußere Ereignisse (Interruptsignal). *Interrupts* spielen bei der Steuerung des Datenflusses von und zu externen Geräten eine wichtige Rolle. Der AVR unterstützt diverse Interrupt-Arten. Diese werden aber im Rahmen der Vorlesung nicht betrachtet.

Wie bereits erwähnt, orientiert sich die Darstellung über Assembler an der realen Hardware des Mikrocontrollers Atmel AVRmega8 und dessen Assembler. Im folgenden sollen die verschiedenen *Befehlstypen* näher erläutert werden.

9.4.1 Datentransferbefehle

Mittels *Datentransferbefehlen* können Wertzuweisungen zwischen einzelnen Registern, aber auch zwischen Registern und angegliedertem Speicher und der I/O stattfinden.

Dabei unterstützt der AVRmega8 die folgenden direkten und indirekten *Adressierungsarten*.

Datenverschiebung zwischen Registern (**MOV**)

Um eine Information von einem Register in ein anderes zu verschieben, muss der Assemblerprogrammierer den Assemblerbefehl *MOV* nutzen. Dieser erfordert noch zwei weitere Parameter, nämlich aus welchem Register gelesen (*Rr*) und in welches geschrieben (*Rd*) werden soll.

MOV Rd, Rr $Rd \leftarrow Rr$

Unmittelbares Laden (LDI)

Mittels des Assemblerbefehls *LDI* ist es möglich, unmittelbar eine *Konstante* in ein Register zu laden. Hierzu müssen die Parameter *K* für die definierte Konstante und *Rd* für das *Zielregister* bestimmt werden. Dieses muss beim Befehl *LDI* eines der Register *R16-R31* sein. Die Folge ist ein sofortiges Laden in das Register *Rd*.

LDI Rd, K $Rd \leftarrow K$

Direktes Laden aus dem Speicher (LDS)

Zudem ist es möglich, ein Datum direkt aus dem *Speicher* zu laden. Hierzu wird als Parameter des Befehls *LDS* die Speicheradresse *k* benötigt, aus der gelesen werden soll, sowie das Register *Rd*, in das geschrieben werden soll.

LDS Rd, k $Rd \leftarrow (k)$

Indirektes Laden (LD)

Darüber hinaus unterstützt der betrachtete *Mikrocontroller* zwei Arten des indirekten Ladens. Beim ersten Befehl wird der Inhalt eines Registers in ein anderes verschoben. Hierbei handelt es sich um eine *Speicheradresse* *Z* des Datenspeichers, die ausgewertet wird und deren Inhalt dann in das vorgesehene Register *Rd* geschrieben wird.

LD Rd, Z $Rd \leftarrow Z$

Indirektes Laden mit Offset (LDD)

Beim zweiten Befehl wird auf die ermittelte Speicheradresse ein Wert aufaddiert. Dies ermöglicht das Lesen von Daten ab einer bestimmten Adresse.

Das Datum der auf diese Weise ermittelten Speicheradresse (*Z+q*) wird dann in das Register *Rd* übertragen.

LDD Rd, Z+q $Rd \leftarrow (Z+q)$

Speichern (STS, ST und STD)

Da Daten aber nicht nur gelesen werden, existieren auch Befehle zum Speichern. Diese verhalten sich analog zu den oben genannten Befehlen zum Lesen.

Zu nennen sind hierbei die Befehle *STS*, *ST*, *STD*:

- *direktes Speichern*

STS k, Rr $(k) \leftarrow Rr$

- *indirekten Speichern*

ST Z, Rr (Z) \leftarrow Rr

- indirekten Speichern mit Verschiebung

STD Z+q, Rr (Z+q) \leftarrow Rr

Lesen und Schreiben auf Ports (*IN* und *OUT*)

Sofern man mittels des Assemblers für den Atmel Mikrocontroller *Ports* auslesen möchte, so geschieht dies mit dem Befehl *IN*. Hierbei wird der Wert des Ports in das Register Rd übertragen.

IN Rd, P Rd \leftarrow P

Umgekehrt kann man auch den Wert eines Registers in einen Port schreiben. Hierzu ist der Assemblerbefehl *OUT* zu nutzen.

OUT P, Rr P \leftarrow Rr

Ports werden in diesem Zusammenhang benötigt, um Zugriff auf *E/A-Funktionen* zu erlangen . Zudem geschieht der Datenaustausch des *Timers*, des *Comparators*, des *A/D-Wandlers* und des *Stackpointers* über Ports.

Schreiben auf und Lesen vom Stack *PUSH* und *POP*

Um auf den Stack zu schreiben, wird der Assemblerbefehl *PUSH* genutzt, der als Parameter das zu sichernde Register Rr enthält. Dieses wird auf dem Stack abgelegt. Anschließend wird der *Stackpointer* dekrementiert. Dieser zeigt nun auf die nächst niedrigere Adresse.

PUSH Rr Stack \leftarrow Rr, Stackpointer \leftarrow Stackpointer - 1

Um vom Stack zu lesen, wird der Assemblerbefehl *POP* genutzt. Hier wird zunächst der Stackpointer inkrementiert und im Anschluss das Element vom Stack in das angegebene Register Rd verschoben.

POP Rd Stackpointer \leftarrow Stackpointer + 1, Rd \leftarrow Stack,

Beachtet werden muss aber in jedem Fall, dass vor der Verwendung des Stackpointers zuerst seine *Initialisierung* stattfinden muss, damit dieser einen gültigen Wert hat und nicht durch die *Stackoperationen* andere gespeicherten Werte (Variablen etc.) überschrieben werden. Zudem muss der Programmierer dafür Sorge tragen, dass vor jedem Aufruf des Befehls *POP* der Befehl *PUSH* ausgeführt wurde.

9.4.2 Arithmetische Operationen

Um arithmetische Operationen durchzuführen, sind im Falle des Atmel-Mikroprozessors unter anderem folgende Operationen als Befehle implementiert.

Addition (ADD)

Der Assemblerbefehl *ADD* summiert die Register *Rd* und *Rr* und speichert das Ergebnis in *Rd*.

ADD Rd, Rr $Rd \leftarrow Rd + Rr$

Subtraktion (SUB)

Um ein Register *Rr* von einem Register *Rd* zu subtrahieren, wird der Befehl *SUB* genutzt. Dieser speichert das Ergebnis im Register *Rd*.

SUB Rd, Rr $Rd \leftarrow Rd - Rr$

Multiplikation (MUL)

Die Multiplikation (*MUL*) verhält sich analog zur Addition und Subtraktion. Allerdings wird das Ergebnis der Multiplikation nicht in *Rd* gespeichert, sondern in *R1 : R0*.

MUL Rd, Rr $R1 : R0 \leftarrow Rd \times Rr$

Linksshift des Registers *Rd* (LSL)

Um den Inhalt eines Registers nach links zu verschieben, stellt der Assembler des Mikrocontrollers den Befehl *LSL* bereit. Die *Bit-Stelle* einer jeden Stelle des Registers wird um eins erhöht. Die Bit-Stelle 0 wird mit einer 0 gefüllt.

LSL Rd $Rd(n + 1) \leftarrow Rd(n), Rd(0) \leftarrow 0$

Rechtsshift des Registers *Rd* (LSR)

Der Rechtsshift mittels *LSR* verläuft analog. Jede Bit-Stelle des Registers wird auf die nächst kleinere umgesetzt. Das führende Bit *Rd* (7) wird mit einer 0 gesetzt.

LSR Rd $Rd(n) \leftarrow Rd(n + 1), Rd(7) \leftarrow 0$

Inkrementieren und Dekrementieren des Registers (INC und DEC)

Mittels dieser Befehle ist eine einfache Manipulation der Werte des Registers möglich. Soll der Wert um eins erhöht werden, so geschieht dies durch den Aufruf des Befehls *INC*. Umgekehrt ist eine Dekrementierung durch Verwendung des Befehls *DEC* möglich.

INC Rd $Rd \leftarrow Rd + 1$
DEC Rd $Rd \leftarrow Rd - 1$

9.4.3 Logische Operationen

Neben den *arithmetischen Verknüpfungen* können auch logische zur Manipulation des Inhalts eines Registers zum Einsatz kommen.

Bitweise Und-Verknüpfung (AND)

Soll eine bitweise Und-Verknüpfung zweier 8-Bit-Register stattfinden, wird der Befehl **AND** genutzt. So werden jeweils entsprechende Bit-Stellen miteinander verknüpft. Bei zwei Einsen einer Bit-Stelle wird eine 1 im Register **Rd** gesetzt, ansonsten wird eine 0 gesetzt.

AND Rd, Rr $Rd \leftarrow Rd \text{ AND } Rr$

Bitweise Oder-Verknüpfung/Exklusiv-Oder-Verknüpfung (OR und EOR)

Analog geschieht die bitweise *Oder-Verknüpfung* und *Exklusiv-Oder-Verknüpfung*.

OR Rd, Rr $Rd \leftarrow Rd \text{ OR } Rr$
EOR Rd, Rr $Rd \leftarrow Rd \text{ XOR } Rr$

Bitweises Negieren (COM)

Das bitweise Negieren eines Registers **Rd** geschieht in diesem Zuge durch *Invertieren* der einzelnen Bits. Hierzu dient der Befehl **COM**.

COM Rd $Rd \leftarrow \$FF - Rd$

9.4.4 Vergleichsbefehle (CP und CPI)

Um zu prüfen, ob in zwei Registern gleich große Werte enthalten sind, kann dies über den Befehl **CP** des Assemblers abgefragt werden. Somit wird intern der Wert des Registers **Rr** vom Wert des Registers **Rd** subtrahiert und auf 0 geprüft.

CP Rd, Rr $Rd - Rr$

Zudem kann der Wert eines Registers auch mit einer *Konstante K* verglichen werden. Dies geschieht analog zum Vergleich mit einem zweiten Register über den Befehl **CPI**.

CPI Rd, K $Rd - K$

Zu beachten ist in beiden Fällen, dass es sich beim Compare um eine Subtraktion handelt, deren Ergebnis nicht gespeichert wird, sondern nur die entsprechenden Statusregister gesetzt werden.

9.4.5 Bitbefehle (SBR, CBR, BSET und BCLR)

Assemblerbefehle dieser Gruppe ermöglichen das Setzen und Löschen einzelner Bits in Registern.

So kann zum Beispiel durch den Befehl **SBR** ein einzelnes oder mehrere Bits im Register gesetzt werden indem eine OR-Verknüpfung mit einer entsprechenden Variablen durchgeführt wird.

SBR Rd, K $Rd \leftarrow Rd \text{ OR } K$

Das Löschen entsprechender Bits im Register geschieht durch Verwendung des Befehls *CBR*.

CBR Rd, K $Rd \leftarrow Rd \text{ AND } (0xFF - K)$

Um speziell im *Statusregister* Bits zu setzen oder zu löschen, müssen die Befehle *BSET* und *BCLR* verwendet werden.

Beim Befehl *BSET* wird über den Befehl *SREG* an der Stelle *s* eine 1 gesetzt; zum Löschen wird an der Stelle *s* eine 0 gesetzt.

BSET s $SREG(s) \leftarrow 1$
BCLR s $SREG(s) \leftarrow 0$

9.4.6 Verzweigebefehle

Um neben rein *sequenziellen* Programmen auch solche mit *Verzweigungen* zu unterstützen, bietet der Assembler des Mikrocontrollers der Firma Atmel auch *Verzweigebefehle*. Dabei unterscheidet man zwischen zwei Arten.

Eine Befehlssequenz kann als Unterprogramm aus dem eigentlichen (Haupt-)Programm ausgelagert werden. Das Unterprogramm kann dann von beliebiger Stelle des *Hauptprogramms* aus aufgerufen werden.

Die Verwendung von Unterprogrammen macht das Hauptprogramm übersichtlicher. Außerdem wird Speicherplatz gespart, falls das Unterprogramm mehrfach aufgerufen wird.

Beim Sprung in das Unterprogramm muss der Wert des *Befehlszählers* zwischengespeichert werden. Das Zwischenspeichern wird automatisch vom Sprungbefehl (z.B. *RCALL*) durchgeführt. Beim Verlassen des Unterprogramms wird dieser Wert dann wieder in den Befehlszähler geschrieben, damit das *Hauptprogramm* an der richtigen Stelle fortgesetzt wird. Den *Rücksprung* und das Setzen des Befehlszählers erledigt der *RET*-Befehl.

Für das Zwischenspeichern des Befehlszählers verwendet der *RCALL*-Befehl den *Stack*. Durch das *LIFO*-Konzept des Kellers sind dann auch geschachtelte *Unterprogrammaufrufe* möglich, d.h. ein Unterprogramm kann wiederum ein Unterprogramm aufrufen.

Dabei ist es auch möglich, dass sich ein Unterprogramm wiederholt selbst aufruft. Dies nennt man *Rekursion* (vgl. Abschnitt 9.4.8).

Darüber hinaus ist es möglich, im Programmablauf durch Sprünge zu verzweigen. Hierbei wird aber nicht, wie oben beschrieben, die Rücksprungadresse gesichert. Es ist also somit nur möglich, zu einer definierten *Sprungmarke* bzw. zu einem Befehl, auf den der *Programmcounter* dann zeigt, zu gelangen.

Sprungmarken sind hierbei Speicherzellen mit einem symbolischen Namen, der definiert und einer Speicherzelle zugeordnet wurde.

Relativer Aufruf eines Unterprogramms (*RCALL*)

Um den Aufruf eines Unterprogramms zu ermöglichen, wird der Programmcounter inkrementiert und auf dem Stack gesichert. Anschließend wird ein relativer Sprung (siehe relativer Sprung, Abschnitt 9.4.6) durchgeführt.

RCALL k Stack \leftarrow PC + 1, PC \leftarrow PC + k + 1

Rücksprung aus dem Unterprogramm (RET)

Um anschließend wieder aus dem Unterprogramm zurückzukehren und den „sequenziellen“ Programmablauf aufzunehmen, wird der gesicherte *Programmcounter* wiederhergestellt. Dies geschieht mittels des Befehls *RET*. Der Programmcounter zeigt hierbei auf die nächste auszuführende Befehlszeile.

Der Programmierer muss aber darauf achten, dass er vor dem Aufruf des Befehls *RET* den Befehl *RCALL* oder *ICALL* ausgeführt hat.

RET PC \leftarrow Stack

Relativer Sprung (RJMP)

Mit dem Befehl *RJMP* kann ein relativer Sprung durchgeführt werden. Der Programmcounter wird in diesem Fall nicht nur um 1 inkrementiert, sondern zusätzlich um den Wert k erhöht.

Mit relativer Adressierung können Programme so geschrieben werden, dass sie beliebig im Speicher verschiebbar sind und die Adressangaben trotzdem immer stimmen. Diese Eigenschaft wird für Programmteile benötigt, die als Teil anderer Programme (die in beliebigen Speicherbereichen liegen können) ablaufen sollen.

RJMP k PC \leftarrow PC + k + 1

Bedingter Sprung (BREQ und BRLO)

Darüber hinaus ist es möglich, im Programmablauf zu springen, wenn ein Registerwert gleich 0 ist.

Dies ist mit dem Befehl *BREQ* möglich, der testet, ob das *Zero-Flag* gesetzt wurde (Gleichheit). In diesem Fall wird ein relativer Sprung durchgeführt.

BREQ k if (Z = 1) then PC \leftarrow PC + k + 1

Analog kann auch über das *Carry-Flag* (Übertragsbit) getestet werden, ob ein Wert kleiner ist als ein anderer und in diesem Fall verzweigen. Diese Funktionalität wird über den Assemblerbefehl *BRLO* bereitgestellt.

BRLO k if (C = 1) then PC \leftarrow PC + k + 1

Indirekter Sprung (IJMP)

Beim *indirekten Sprung* wird aus dem Register Z der neue Wert des Programmcounters gelesen. Hierbei handelt es sich um eine Speicheradresse des Programmspeichers. Diese wird gelesen und das Programm wird an dieser Stelle fortgeführt. Das heißt, dass bei der indirekten Adressierung der Inhalt der im Maschinenbefehl angegebenen Adresse nicht als der *Operand* selbst, sondern wiederum als Adresse interpretiert wird, in der dann der Operand steht.

IJMP PC \leftarrow Z

Indirekter Aufruf eines Unterprogramms (ICALL)

Mittels des Befehls *ICALL* kann ein Unterprogramm indirekt angesprungen werden. Der Programmcounter wird auf dem Stack gesichert und ein indirekter Sprung (s.o.) durchgeführt. Nach Ablauf des Unterprogramms kann der Programmcounter aus dem *Stack* geholt und für den weiteren Programmablauf gesetzt werden.

ICALL $\text{Stack} \leftarrow \text{PC} + 1, \text{PC} \leftarrow Z$

9.4.7 Programmierung von Schleifen

In höheren Programmiersprachen können Schleifen in der Form

WHILE <Bedingung> *DO* <Anweisung>

programmiert werden.

In Assembler können diese *Schleifen* mit Hilfe von bedingten Sprüngen und *Marken* realisiert werden. Hierzu wird während des Programmablaufs ein Wert in ein Register geschrieben, der anschließend in einer *Schleife* beispielsweise dekrementiert werden soll.

Die Schleife selbst ist durch eine entsprechende *Sprungmarke* gekennzeichnet.

Soll, wie bereits erwähnt, der Wert dekrementiert werden, geschieht dies über den Befehl DEC. Der Inhalt des Registers wird um eins erniedrigt und das Z-Flag (*Zero-Flag*) berechnet. Sofern das Z-Flag gleich 1 ist (d.h. der Registerwert ist gleich 0), wird die Schleife durch den anschließenden Befehl BREQ mit der entsprechenden Sprungmarke (im unteren Beispiel end; siehe Abbildung 9.11) verlassen. Ist der Wert größer 0, wird ein relativer Sprung zum Beginn der Schleife durchgeführt und erneut durchlaufen. Dies geschieht so lange bis das Z-Flag gleich 1 ist, d.h. der Wert des Registers gleich 0.

Dann wird der Code der Sprungmarke ausgeführt, über die die Schleife verlassen wurde. Im Beispiel NOP. Die Abbildung 9.11 zeigt eine solche Schleife und die Sprungmarken.

```

        LDI r16,10 ; lade 10 in R16
loop:   NOP      ; Schleifeninhalt ←
        DEC r16    ; Verringere R16 um 1
        BREQ end   ; Falls 0: zu end springen
        RJMP loop  ; zu loop springen
        NOP      ; Code nach der Schleife
end:    NOP

```

Abbildung 9.11: Beispielhafte Implementierung einer Schleife in Assembler

9.4.8 Rekursion

Wie bereits in Kapitel 9.4.6 erwähnt, ist es mit Assembler möglich ein *Unterprogramm* von einem anderen Unterprogramm aufrufen zu lassen. Einen speziellen Fall stellt hierbei der Aufruf eines Unterprogramms durch sich selbst dar.

Definiert der Assemblerprogrammierer nun zur Berechnung eines (mathematischen) Problems

das Unterprogramm so, dass der *Funktionswert* $f(n)$ durch die Berechnung des Funktionswertes von $f(n - 1)$ abhängt und dies fortgeführt wird, bis eine gegebene Abbruchbedingung dieser Verschachtelung erreicht ist, spricht man von *Rekursion*.

Das Unterprogramm wird somit solange erneut aufgerufen, bis ein durch den Programmierer definierter Wert zur Berechnung herangezogen werden kann.

Erfolgt keine korrekte Definition der *Abbruchbedingung*, so berechnet das Programm zu viele Durchläufe des Unterprogramms und beendet unter Umständen die Berechnung nie. Daher ist auf eine korrekte Abbruchbedingung zu achten, die erreicht werden kann.

Soll zum Beispiel das Produkt der Zahlen von 1 bis n berechnet werden, so ist dies mathematisch definiert durch:

$$fak(n) = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \dots \cdot (n-1) \cdot n$$

Die *rekursive Beschreibung* der Produktfunktion (Fakultät) sieht vor, dass das Produkt aller Zahlen von 1 bis n durch das Produkt aller vorhergehenden, also des Produktes der Zahlen von 1 bis n-1, multipliziert mit der aktuellen Zahl erhalten werden kann.

$$fak(n) = fak(n-1) \cdot n$$

Um eine korrekte Abbruchbedingung zu erhalten wird der *Rekursionsanfang* als $fak(1) = 1$ definiert.

Somit ist die Produktfunktion rekursiv definiert als:

$$f(n) = \begin{cases} 1, & \text{wenn } n = 1 (\text{Rekursionsanfang}) \\ Prod(n - 1) \cdot n, & \text{wenn } n > 1 (\text{Rekursionsschritt}) \end{cases} \quad (9.1)$$

Das Produkt der Zahlen von 1 bis 4 berechnet sich beispielsweise wie folgt:

$$\begin{aligned} fak(4) &= fak(3) \cdot 4 \\ &= fak(2) \cdot 3 \cdot 4 \\ &= fak(1) \cdot 2 \cdot 3 \cdot 4 \\ &= 1 \cdot 2 \cdot 3 \cdot 4 \\ &= 24 \end{aligned}$$

Um eine gleichwertige Beschreibung der Produktfunktion in Assembler zu erhalten, muss der Code so aufgebaut sein, dass innerhalb einer Unterfunktion die Zerlegung des Problems in ein *Teilproblem* geschieht ($fak(n) = fak(n-1) \cdot n$). Dies darf aber nur so lange geschehen, bis der Rekursionsanfang erreicht ist.

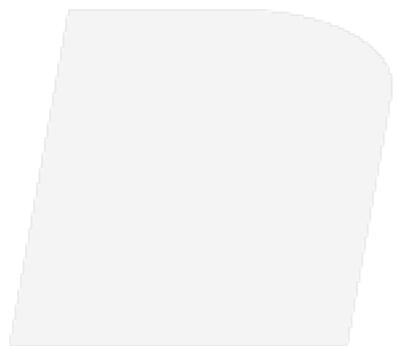
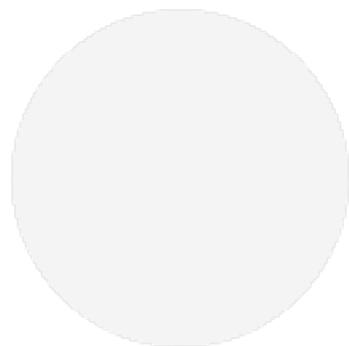
In diesem Fall ist das Unterprogramm durch einen Sprung zu verlassen. Die ermittelten Faktoren werden dann miteinander multipliziert.

Nachfolgend ist ein solches Assemblerprogramm für den Atmel AVRmega8 aufgelistet:

```
.include "m8def.inc"

LDI r16, LOW(RAMEND)      ; LOW-Byte der obersten RAM-Adresse
OUT SPL, r16
LDI r16, HIGH(RAMEND)     ; HIGH-Byte der obersten RAM-Adresse
OUT SPH, r16
```

```
LDI r16,4  
RCALL fak  
NOP  
fak:  
    PUSH r16  
    CPI r16,1  
    BREQ done  
    DEC r16  
    RCALL fak  
done:  
    POP r17  
    MUL r16,r17  
    MOV r16,r0  
    RET
```



Kapitel 10

Organisation der Ein-/Ausgabe

10.1 Ein-/Ausgabe-Hardware

10.1.1 Grafikkarten

Abbildung 10.1 zeigt den Aufbau einer *Grafikkarte*. Über den Busanschluß (siehe Kap. 10.2) gelangen die Daten in den Bildspeicher (Video-RAM). Von dort werden sie vom Frame Buffer Controller (FBC) gelesen. Der FBC besteht aus einem ASIC-Baustein mit Registern zur Erzeugung der horizontalen und vertikalen Synchronisationssignale zur Ansteuerung eines Kathodenstrahlmonitors.

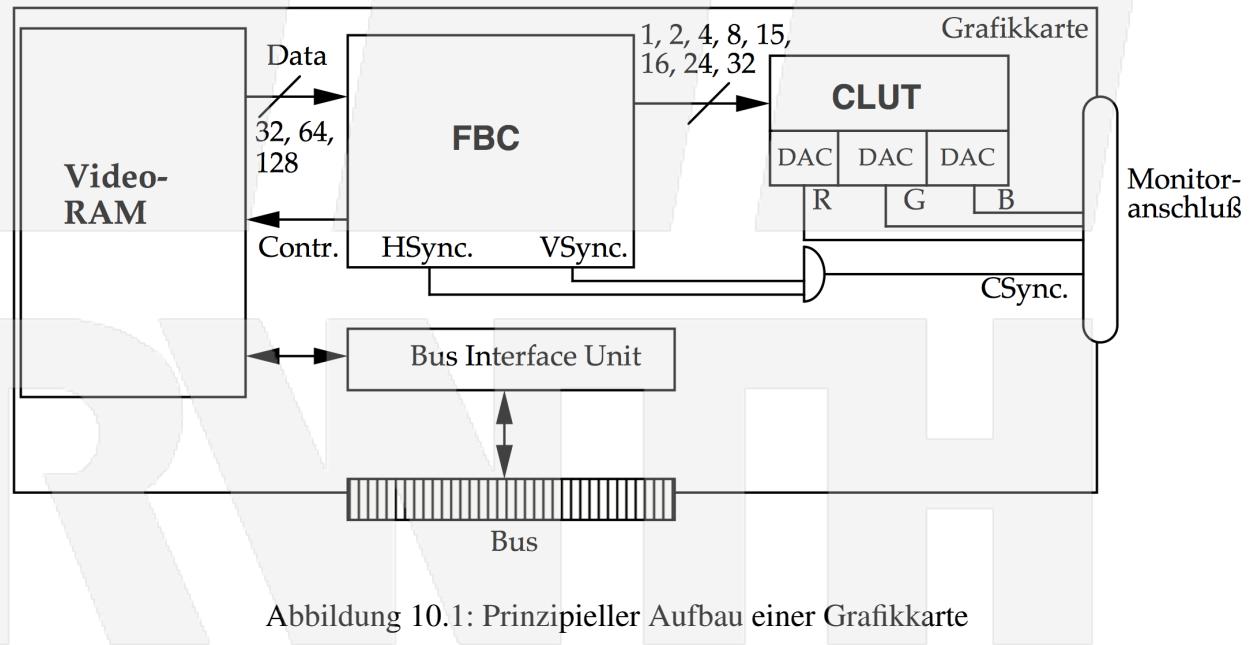


Abbildung 10.1: Prinzipieller Aufbau einer Grafikkarte

Die CLUT-Einheit (Color Look Up Table) enthält eine Tabelle, die eine bestimmte Farbnummer aus dem Video-RAM in einen Farbwert, bestehend aus Rot-, Grün- und Blauanteil, bestimmt. Diese Tabelle ist nicht erforderlich, wenn eine Farbtiefe von 15, 16, 24 oder 32 Bits pro Pixel (Pixel = Picture Element, Bildpunkt) eingestellt ist. Dann erfolgt eine direkte Aufteilung der Daten aus dem Video RAM auf die Digital Analog Wandler (DAC) (z.B. 8 bit für jede Grundfarbe bei 24 Bits pro Pixel). Diese digitalen Farbwerte werden durch Digital-/Analog-Wandler in

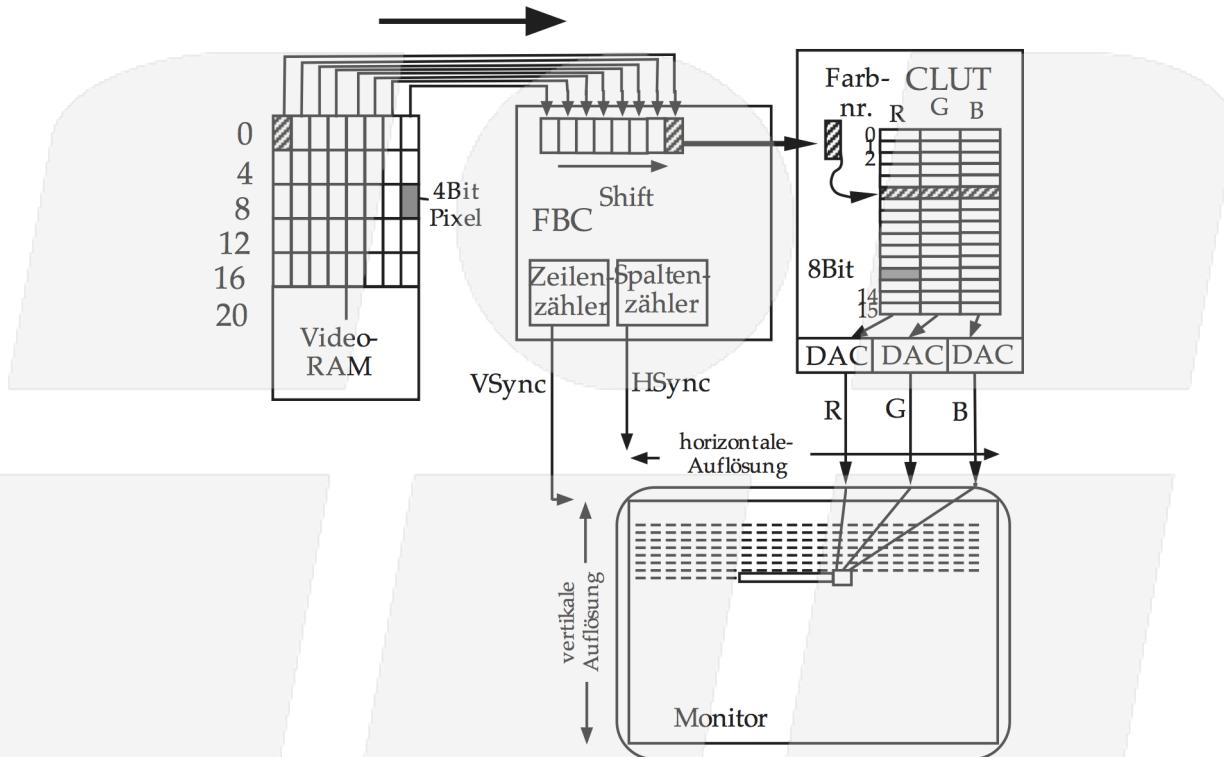


Abbildung 10.2: Schema des Bildaufbaus auf einem Monitor

Analogwerte zur Ansteuerung der RGB-Kanonen des Monitors umgesetzt. CLUT und die Digital/Analog Wandler befinden sich meist in einem Baustein der als *RAMDAC* bezeichnet wird. Der FBC berechnet dabei die für die Darstellung benötigte Information über die horizontale und vertikale Position. Abbildung 10.2 zeigt den Vorgang des Bildaufbaus auf dem Monitor im Detail.

Die *Synchronisationssignale* sind erforderlich, damit der Elektronenstrahl des Monitors wieder an den Anfang einer Zeile (HSync-Signal) bzw. wieder an den Anfang des Bildes (VSync-Signal) zurückkehren kann.

Die aktuelle horizontale Frequenz (Zeilenrate) und vertikale Frequenz (Bildwiederholrate) ergeben sich aus der Anzahl der Spalten, bzw. Zeilen, der Dauer des Strahlrücklaufs und der Zeitdauer, die für die Darstellung eines Pixels benötigt wird (Pixeltakt). Für die Horizontalfrequenz und Vertikalfrequenz erhält man damit:

$$hfreq = (t_{hsync} + spalten \cdot t_{pixel})^{-1}$$

$$vfreq = \frac{hfreq}{zeilen_sichtbar + zeilen_vsync}$$

Dabei steht t_{hsync} für die Dauer des HSync-Signals, t_{pixel} für die zur Darstellung eines Pixels benötigte Zeit. Die Größen $zeilen_sichtbar$ und $zeilen_vsync$ geben an, wie viele sichtbare Zeilen es gibt bzw. wie viele Zeilen während des VSync-Signals dargestellt werden könnten. Die Vertikalfrequenz und die Dauer des vertikalen Synchronimpulses werden dabei in Abhängigkeit der Zeilenfrequenz angegeben. Richtwerte für die Dauer des Horizontal- und Vertikalimpulses sind ca. 10% der Länge einer Zeile (in Pixeln) bzw. der Höhe des Bildes (in Zeilen).

Beschleunigung der Grafikausgabe

Das bisher dargestellte Prinzip einer Grafikkarte erfordert, dass die CPU sämtliche Aufgaben der Grafikerzeugung übernimmt. Das bedeutet, dass die CPU jeden Bildpunkt explizit ansprechen muss. Dadurch entsteht eine hohe CPU-Last, die zu Lasten der auszuführenden Programme geht.

Deswegen wurden in den letzten Jahren Grafikkarten entwickelt, die, neben der Integration des RAMDACs in den FBC zur Reduktion des Verdrahtungsaufwandes, in der Lage sind, so genannte Grafikprimitive wie z.B. das Zeichnen von Linien, Kreisen, Polygonzügen oder das Füllen von Flächen unabhängig von der CPU durchführen können. Von der CPU wird lediglich ein Kommando an den Grafikchip gesandt, der die Anweisung für die Grafikprimitive enthält. Dadurch reduziert sich der Rechenaufwand für die Darstellung eines Bildes beträchtlich. Die CPU-Leistung steht somit den Anwendungsprogrammen zur Verfügung.

Man unterteilt diese *Grafikbeschleuniger* in zwei Klassen:

1. 2D oder GUI Beschleuniger

Diese sind in der Lage zweidimensionale Objekte beschleunigt darzustellen, wodurch sich insbesondere die Darstellung der Elemente einer grafischen Benutzeroberfläche (GUI, Grafical User Interface) beschleunigen lässt.

2. 3D Beschleuniger

Insbesondere beim computergestützten Entwurf (CAD, Computer Aided Design), den Anwendungen der Virtuellen Realität (VR) und dem Unterhaltungssektor kommen in der letzten Zeit verstärkt Grafikbeschleuniger zur Darstellung dreidimensionaler Objekte zum Einsatz. Ein wesentlicher Unterschied zu 2D Beschleunigern ist ein zusätzlicher Speicher, Z-Buffer genannt, der die Tiefeninformation der darzustellenden Objekte enthält, wodurch sich teilweise verdeckte Flächen sehr schnell berechnen lassen.

Digitale Ansteuerung von Bildschirmen

Moderne Bildwiedergabesysteme wie Flachbildschirme oder Digitalprojektoren benötigen im Gegensatz zu Röhrengeräten eine digitale Darstellung der Bilddaten. Bei der bisher üblichen Datenübertragung per VGA-Kabel werden die im Computer digital vorliegenden Daten in analoge RGB-Signale gewandelt, zum Digitalmonitor geschickt und dort wieder digitalisiert. Um diese überflüssigen Schritte zu umgehen wurden verschiedene Standards zur digitalen Bildübertragung entworfen, die im Folgenden kurz vorgestellt werden. Sie unterscheiden sich im Wesentlichen nur durch die Art ihrer Steckverbindungen und der zusätzlichen Daten, die außer dem Bildsignal übertragen werden können. Alle Standards verwenden das von Silikon Image entwickelte und von der DDWG (Digital Display Working Group) spezifizierte *TMDS-Protokoll* (Transition-Minimized Differential Signaling), auch *PanelLink* genannt. Hierbei werden die digitalen Farbwerte für Rot, Grün und Blau getrennt über drei parallele Leitungspaare seriell übertragen. Die Übertragung erfolgt differentiell und symmetrisch in jedem Leitungspaar, wodurch sich eine hohe Störsicherheit ergibt, da äußere Störeinflüsse sich auf beide Leitungen gleichermaßen auswirken. Die DDWG ist ein Konsortium von Unternehmen, das 1998 mit dem Ziel gegründet wurde, einen Standard für die digitale Bildverarbeitung zu schaffen.

P&D (Plug and Display)

Die erste Version dieses Standards wurde 1997 von der VESA (Video Electronics Standards Organization) vorgestellt. In der 30+4 - poligen Steckverbindung sind außer den TMDS-Bilddaten



auch Datenleitungen für analoge Bildübertragung, USB- und Firewire-Signale integriert. Aufgrund dieses Aufwands hat sich dieser Standard als zu teuer und für die Praxis unbrauchbar erwiesen und konnte sich nicht durchsetzen.

DFP (Digital Flat Panel)

1999 von der Digital Flat Panel Group (angeführt von Compaq) vorgestellt, handelt es sich hierbei im Wesentlichen um eine vereinfachte Version des P&D, die sich auf die Übertragung der digitalen Bilddaten per TMDS-Protokoll beschränkt und mit einer 20-poligen Steckverbindung auskommt. Wie schon bei P&D beträgt die maximale übertragbare Auflösung 1280x1024 Pixel, was neben der fehlenden Möglichkeit, auch Analogbildschirme weiterhin ansteuern zu können, der Hauptgrund ist, dass sich auch dieses Verfahren nicht durchsetzen kann.

DVI (Digital Visual Interface)

Im Gegensatz zu den obigen Standards bietet diese 1999 von der DDWG vorgestellte Digitalverbindung einen zusätzlichen TMDS-Link, der parallel genutzt werden kann, wodurch sich die Datenrate verdoppelt. So beträgt die maximal übertragbare Bildauflösung im sog. *Single Link* 1600x1200 Pixel, im *Dual Link* jedoch 2048x1536 Pixel. Es gibt zwei Versionen von DVI-Steckverbindungen: DVI-D überträgt nur digitale Signale, während DVI-I („integrated“) zusätzlich drei analoge RGB-Kanäle plus Sync besitzt, wodurch mit einem entsprechenden VGA-Adapterkabel auch analoge Monitore weiterhin angesteuert werden können. Die 24-polige Steckverbindung ist folgendermaßen belegt: 12 TMDS-Leitungen (2 x 3 Paare), 2 Taktleitungen, 2 DDC-Leitungen, 4 Abschirmungen sowie 4 Leitungen für Stromversorgung und V-Sync. Bei dem DDC-Standard (Digital Data Channel) handelt es sich um einen von der VESA definierten Informationskanal zur Übertragung von zusätzlichen Daten wie Plug&Play und Energiesparfunktionen. DVI-Anschlüsse werden zunehmend in Digitalbildschirme und Grafikkarten eingebaut und sind momentan der de-facto-Standard für die digitale Bildübertragung. Ein Nachteil von DVI ist, dass die Kabellängen gemäß Norm auf 5m begrenzt sind. Dieses kann nur durch die Verwendung sehr hochwertiger oder optischer DVI-Verbindungen umgangen werden.

HDMI (High-Definition Multimedia Interface)

Hierbei handelt es sich um eine Weiterentwicklung des DVI-Standards für die Unterhaltungselektronik, um sowohl digitale Video- als auch Audiodaten über einen 19-poligen Miniatursteckkontakt zu übertragen. Der Standard wurde von allen namhaften Herstellern der Unterhaltungselektronik in Zusammenarbeit mit der Filmindustrie entwickelt. Wichtig war hierbei die Umsetzung des digitalen HDCP-Kopierschutzes (High Bandwidth Digital Content Protection), um Filmdaten vor digitaler Kopie zu schützen. Der gleiche Kopierschutz wird auch bei der DVI-Übertragung von digitalen Videoinhalten, wie sie bereits einige DVD-Spieler bieten, verwendet. Die per HDMI erreichbaren Datenraten betragen für Video wie auch bei DVI 165 MHz, was für

HDTV-Signale ausreichend ist, und für Audio bis zu 192 kHz bei 24 Bit Auflösung mit bis zu 8 Kanälen. HDMI ist zu DVI abwärtskompatibel, d.h. dass auch DVI-Signale per Adapter über HDMI übertragen werden können. HDMI-Kabelverbindungen sind im Gegensatz zu DVI auch für längere Strecken geeignet.

10.1.2 Drucker

Drucker sind Ausgabegeräte für Texte und Grafiken. Sie unterscheiden sich durch verschiedene Druckverfahren, d.h. die Art und Weise, wie Farben auf den Träger (Papier, Folien) aufgebracht werden. Abbildung 10.4 gibt eine Übersicht über verschiedene Druckverfahren. Folgende Druckertypen sollen nun näher behandelt werden:

- Nadeldrucker
- Tintenstrahldrucker
- Laserdrucker
- Farbdrucker

Diese Druckertypen haben gemeinsam, dass die Daten in Form von Grafikpunkten (*Pixel, Picture Elements*), meist zu Matrizen zusammengefasst, aufgetragen werden.

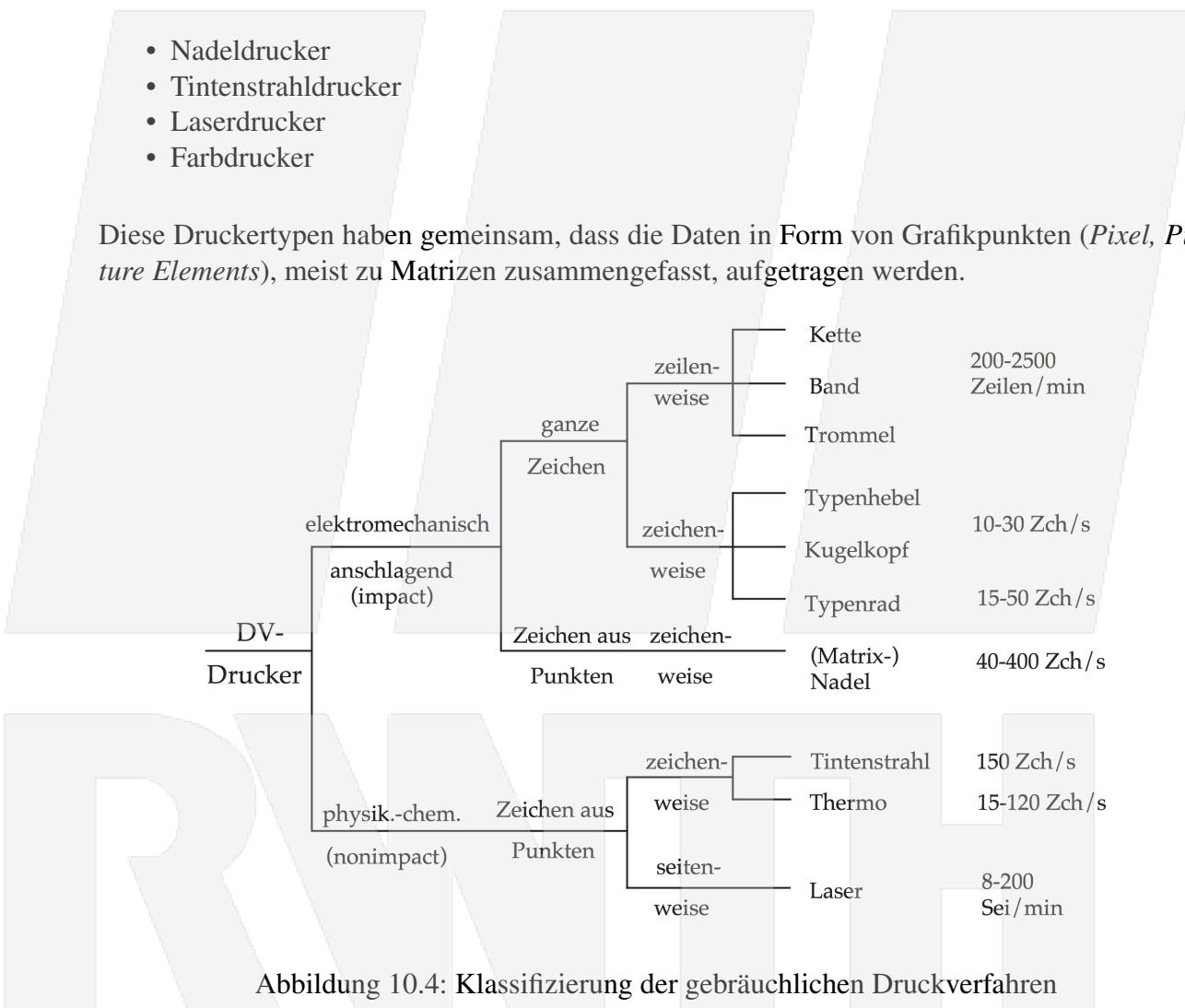


Abbildung 10.4: Klassifizierung der gebräuchlichen Druckverfahren

Nadeldrucker

Nadeldrucker, auch Matrix- oder *Impact*-Drucker genannt, weisen durch ihre Technik einige Vorteile gegenüber Tintenstrahl- oder Laserdruckern auf.

Das Funktionsprinzip des Nadeldruckers veranschaulicht Abbildung 10.5. Es beruht auf einem Druckkopf mit 9, 24 oder gar 48 Nadeln, die von der Druckerelektronik einzeln angesteuert werden können. Soll ein Punkt gedruckt werden, wird durch einen Stromstoß in einer Spule ein Magnetfeld erzeugt, das die Nadel in Richtung der Papierwalze beschleunigt. Das vorgelagerte Farbband wird so (wie bei einer Schreibmaschine) gegen das Papier gedrückt.

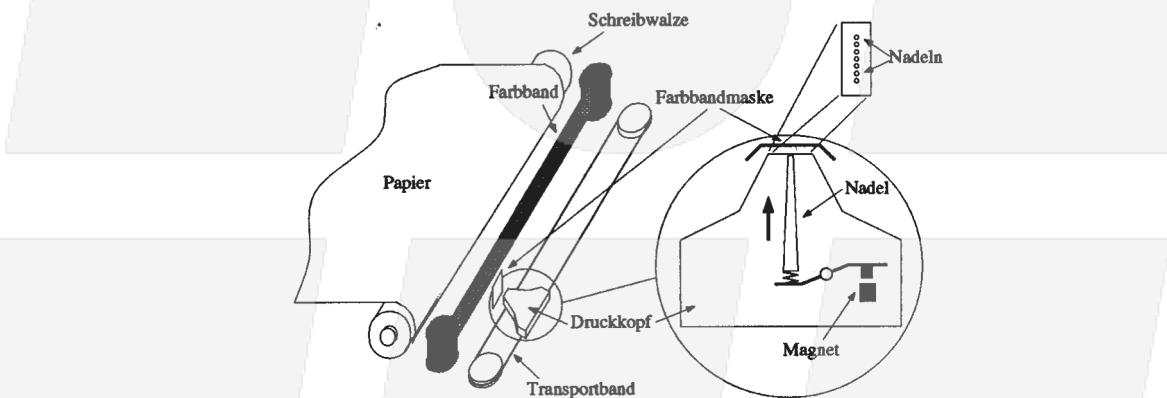


Abbildung 10.5: Funktionsweise eines Nadeldruckers

Durch entsprechend feine Nadeln (0,2 mm beim 24-Nadel-Drucker) lassen sich vertikale Auflösungen bis zu 360 dpi (*dots per inch*, Punkte pro Zoll) erzielen. Feine Schrittmotoren bewegen den Druckkopf zeilenweise am Papier vorbei und lassen horizontale Auflösungen von 180 bis 360 dpi zu. Nach dem Druck einer Zeile wird das an der Gummiwalze vorbeigeführte Papier weiter geschoben (*Line Feed*, Zeilenvorschub) und der Druckkopf an den Anfang der nächsten Zeile zurückgeschoben (*Carriage Return*, Wagenrücklauf).

Nadeldrucker besitzen eine eigene CPU, die die Daten von einer seriellen oder parallelen Schnittstelle einliest, puffert und dann interpretiert. So lassen moderne Geräte zum Beispiel den bidirektionalen Druck zu, wobei das „Spiegeln“ der Druckdaten für den Rückweg des Druckkopfes automatisch im Drucker vorgenommen wird. Neben der Möglichkeit, die deutschen Schriftzeichen und einige Sonderzeichen zu drucken (*ASCII-Zeichensatz*), verstehen die meisten Nadeldrucker bestimmte Steuersequenzen zur Druckkopfpositionierung, zum Grafikdruck und zur Betriebsmodus-Auswahl. So können z.B. unterschiedliche Zeichensätze (*Fonts*), deren Punktmaßnahmen als Matrix im internen Festspeicher (ROM, vgl. Kapitel 10) vorliegen, oder Textattribute (wie Fettdruck, Kursivdruck, Unterstreichen) angewählt werden. Ein Satz solcher Steuersequenzen, der *ESCP*-Standard, wurde von der Firma EPSON entwickelt und wird von vielen Nadeldruckern emuliert.

Andere Druckertypen orientieren sich in ihrer Ausstattung und Funktionsweise häufig an der des Nadeldruckers, der als Pioniergerät der digitalen Drucktechnologie anzusehen ist.

Wegen der sehr niedrigen Druckkosten je Seite (bis 0,5 Cent/Seite) werden Nadeldrucker vor allem dann eingesetzt, wenn große Datenmengen zu drucken sind und die Druckqualität nicht im Vordergrund steht. Ein weiterer Vorteil der Anschlagtechnik liegt in der Möglichkeit, mehrere Durchschläge zu erzeugen (Formulardruck), wozu die Entfernung der Druckeinheit von der Walze variiert werden kann.

Als nachteilig werden mittlerweile die niedrige Druckgeschwindigkeit und die Druckgeräusche angesehen, was zu einer Verdrängung des Nadeldruckers geführt hat. Als Protokoll-Drucker auf

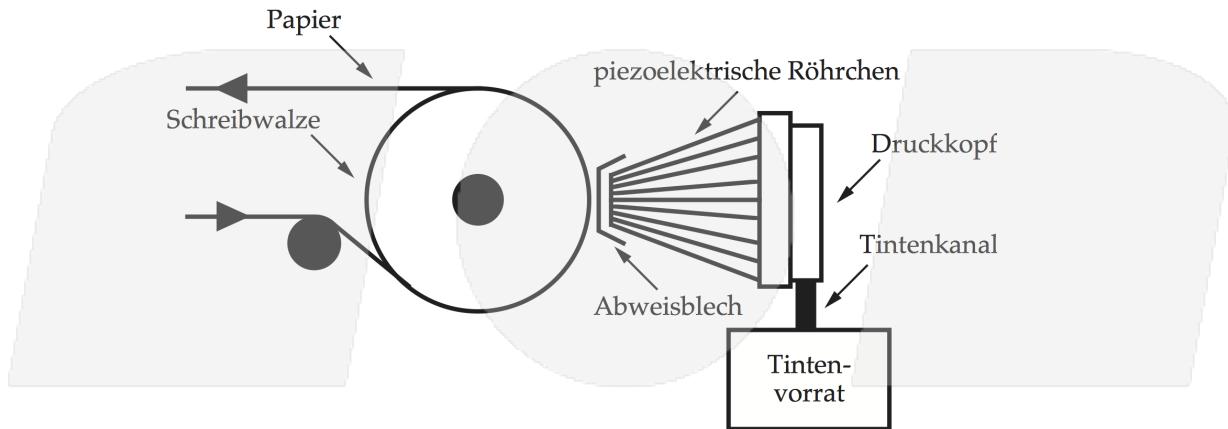


Abbildung 10.6: Prinzipskizze eines Tintenstrahldruckers

Endlospapier für die Industriemaschinen-Überwachung ist ein Nadeldrucker jedoch auch heute noch wegen seiner hohen Zuverlässigkeit gefragt.

Tintenstrahldrucker

Tintenstrahldrucker gewinnen durch günstige Anschaffungspreise, geringe Betriebsgeräusche und die erzielbare Druckqualität immer mehr Marktanteile. Sie sind eine Alternative zu den teureren Laserdruckern. Abbildung 10.6 zeigt die Prinzipskizze des Tintenstrahldruckers.

Bei der Tintenstrahltechnik wird das Druckbild durch feinste Tropfen einer flüssigen Tinte erzeugt, die durch bis zu 48 Düsen auf das Papier geschleudert werden. Bei Tintenstrahldruckern werden zwei verschiedene Verfahren angewandt, um einen Überdruck in der Tintendüse zu erzeugen. Bei der Bubble-Jet-Technik befindet sich am Ende der Düse ein Heizelement. Ein Stromstoß setzt so viel Hitze frei, dass ein Teil der Tinte verdampft. Der entstehende Überdruck schleudert den Tintentropfen auf das Papier. Beim Piezo-Verfahren erzeugt ein Piezoplättchen, das sich beim Anlegen einer Spannung verformt, den notwendigen Überdruck in der Düse.

Die übrige Mechanik ist mit der eines Nadeldruckers nahezu identisch, wenngleich die Ausführung (aufgrund geringerer Kräfte im System) weniger robust und damit leichter und kleiner (portabel) sein kann.

Tintenstrahldrucker werden überall dort eingesetzt, wo sehr leise, kleine und leichte Drucker benötigt werden. Durch die aufwändige Druckkopf-Technik (wobei der Druckkopf häufig auch die Tinte enthält, so dass bei erschöpftem Tintenvorrat der gesamte Druckkopf gewechselt werden muss) sind Druckkosten von bis zu 6 Cent pro Seite möglich.

Laserdrucker

Der *Laserdrucker* arbeitet nach einem Verfahren, das ursprünglich für Kopiergeräte entwickelt wurde. Abbildung 10.7 zeigt eine Prinzipskizze.

Die Bildtrommel, ein mit einem fotoelektrischen Material beschichteter Metallzylinder, wird elektrostatisch aufgeladen. Ein Laserstrahl schreibt das negative Bild der darzustellenden Seite zeilenweise auf die geladene Trommel, die an dieser Belichtungseinheit vorbeigedreht wird. Auf diese Weise belichtete Flächen werden durch den Laser entladen. An den verbleibenden

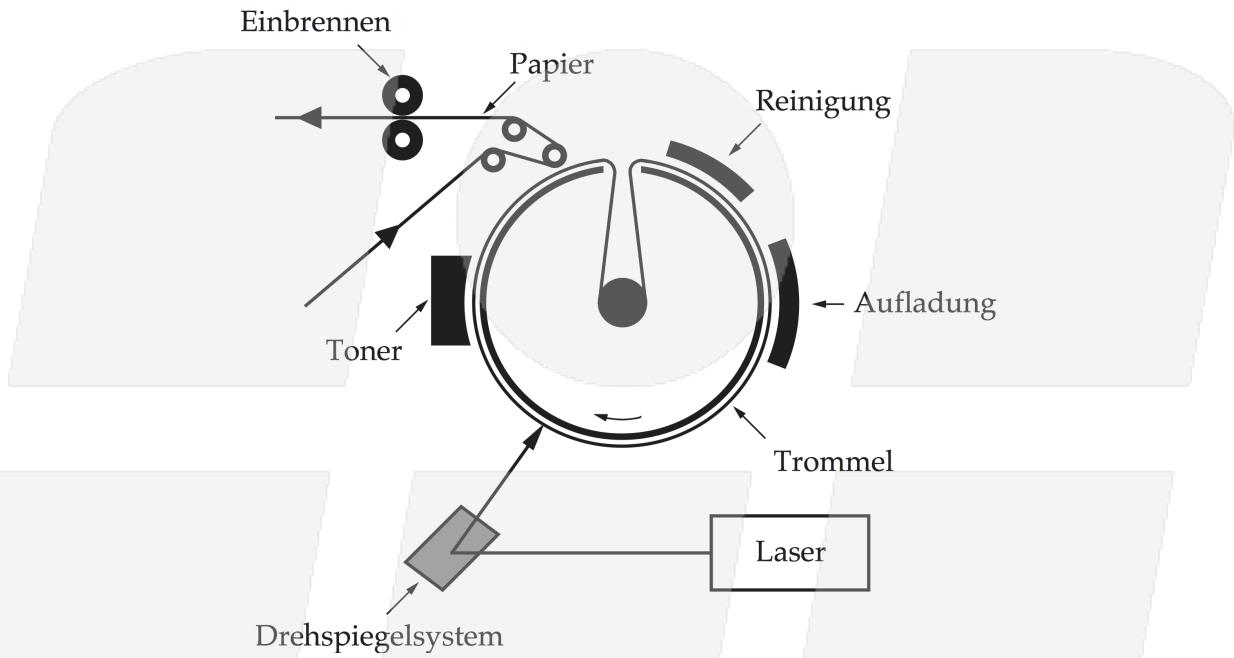


Abbildung 10.7: Prinzipskizze eines Laserdruckers

(noch geladenen) Stellen bleibt so genannter *Toner*, ein harzhaltiges Farbpulver, haften und wird in einem zweiten Prozess auf das Papier übertragen. Durch Hitzeeinwirkung schmilzt das Harz und fixiert so die Farbpunkte auf dem Papier.

Für den Vorgang, der sich in 1 bis 2 Sekunden abspielt, müssen die Daten einer ganzen Seite bereits vorher in einem Speicher gesammelt und für die Druckeinheit aufbereitet werden, um einen kontinuierlichen Datenstrom zu erhalten. Die minimale Größe dieses Speichers beträgt 1 MByte. Geräte mit 512 KByte eignen sich nicht für reinen Grafikdruck, sondern lediglich für Textanwendungen, bei denen der Laserdrucker noch während des Belichtungsvorganges die Bitmuster der Schriftzeichen aus einem eingebauten ROM ausliest.

Um die Datenmengen, die zwischen Drucker und Computer übertragen werden, gering zu halten, hat die Firma Adobe die hardwareunabhängige, vektororientierte Seitenbeschreibungssprache *Postscript* entwickelt. Statt in einer Punktmatrix werden hier nur die Umrisse (*Outlines*) der Objekte in Form von Vektorkoordinaten an den Drucker gesandt. Diese Technik hat unter anderem den Vorteil, dass die Vektorobjekte skaliert werden können, ohne dass ein Qualitätsverlust bei der Darstellung eintritt. Erst nach der Vergrößerung durch skalares Umrechnen der Vektorkoordinaten wird eine „Rastergrafik“ im Speicher des Druckers erzeugt. Für diesen Prozess muss ins ROM des Laserdruckers (für seine schnelle CPU) ein *Postscript*-Interpreter eingebaut sein, d.h. der Rechner des Anwenders braucht sich um die Aufbereitung der Daten nicht zu kümmern. Wegen Lizenzgebühren sind *Postscript*-fähige Laserdrucker teuer. Viele der heute verkauften Systeme sind jedoch bereits für die spätere Nachrüstung eines *Postscript*-Moduls vorbereitet.

Die Kosten für einen Laserdrucker sind mit ca. 5 Cent pro Seite relativ hoch. Dies liegt weniger an dem (ebenfalls teuren) *Toner*, sondern vielmehr an den hohen Wartungskosten. Die mit dem empfindlichen Material Selen beschichtete Bildtrommel verschleißt bereits nach wenigen 10.000 Blatt und muss dann für einige hundert Euro ausgetauscht werden. Der koreanische

Hersteller Kyocera hat jedoch einen völlig wartungsfreien Laserdrucker mit einer keramisch beschichteten Bildtrommel entwickelt, dessen etwas teurere Technologie wirtschaftlich ist und sich bereits durchsetzt.

Farbdrucker

Bereits für wenig Geld sind Farb-Tintenstrahldrucker erhältlich, mit denen sich jedoch keine Ergebnisse erzielen lassen, die den Ansprüchen der Farbbildreproduktion im computerunterstützten, semiprofessionellen Grafikergewerbe (*Desktop Publishing*) genügen könnten. In diesem Bereich werden daher Drucker eingesetzt, die mit der Thermo-Transfer- oder Thermo-Sublimationstechnik arbeiten.

Beim Thermo-Transfer-Verfahren werden durch punktweises Erhitzen einer farbigen Wachsträgerfolie in einer 4×4 oder 5×5 Matrix feine Farbpunkte in den Farben Gelb, Magenta („Rosa“) und Cyan („Türkis“) auf Spezialpapier aufgeschmolzen. Dies geschieht in drei bzw. in vier Durchgängen (falls Schwarz als eigene Trägerfolie verwendet wird). Die verschiedenen Farbtöne ergeben sich durch die Verteilung der 3 bzw. 4 Grundfarben innerhalb der Matrix (additive, bei Überlappung auch subtraktive Farbmischung). Dieses Verfahren wird *Dithering* genannt. Es hat den Nachteil, dass für eine annehmbare Anzahl verschiedener Farbtöne die Dithermatrix entsprechend groß sein muss, was zu einer Verringerung der Auflösung führt.

Das Thermo-Sublimations-Verfahren funktioniert ähnlich wie das Thermo-Transfer-Verfahren. Die Farben werden jedoch nicht von dem Farbträger heruntergeschmolzen, sondern verdampft, so dass sie auf dem Spezialpapier kondensieren. Die aufgetragene Farbmenge ist bei diesem Verfahren variabel: die Erhitzung eines Punktes lässt sich in 256 Stufen unterteilen, so dass ebensoviele Helligkeitsabstufungen einer Grundfarbe erzeugt werden können. Mit den drei Grundfarben ergibt sich durch subtraktive Farbmischung eine Farbpalette von 16,7 Millionen Farben bei voller Druck-Auflösung. Das menschliche Auge unterscheidet nur etwa 3,5 Millionen Farben, so dass sich diese (bisher sehr teure) Technik zur Reproduktion von digitalen Fotografien eignet.

10.1.3 Maus und Trackball

Mit der Einführung von grafischen Benutzeroberflächen hat sich die so genannte Maussteuerung als Ergänzung zur Tastatursteuerung bei der Bedienung von Anwendungsprogrammen durchgesetzt. Prinzipiell unterscheidet man Mäuse bzgl. der Umsetzungsmethode der Mausbewegung in elektrische Information sowie der Art des Anschlusses an den Computer.

Beim opto-mechanischen Prinzip wird die Bewegung einer Kugel im Inneren der Maus, die durch ein Loch im Boden Kontakt mit der (ebenen) Arbeitsfläche hat, mittels Achsen über zwei Rollen auf Locheisen übertragen (Abbildung 10.8). Jeweils zwei Lichtschranken (für die horizontale (x-) und vertikale (y-) Richtung) tasten deren Bewegung ab. Dabei entstehen zwei Rechteck-Signale pro Richtung, über deren Periodenanzahl die (relative) Position (Weiterbewegung) ermittelt wird. Über die Periodendauer kann die Geschwindigkeit bestimmt werden, was die Programmierung einer dynamischen Maussteuerung (d.h. bei schnellen Bewegungen springt der Mauszeiger auf dem Bildschirm in größeren Schritten) ermöglicht. Die Richtung (z.B., ob nach oben oder nach unten bei der vertikalen Bewegung) wird aus der Phasenlage ($\pm 90^\circ$) der beiden Rechtecksignale gewonnen (Abbildung 10.9). Dazu werden die beiden Lichtschranken

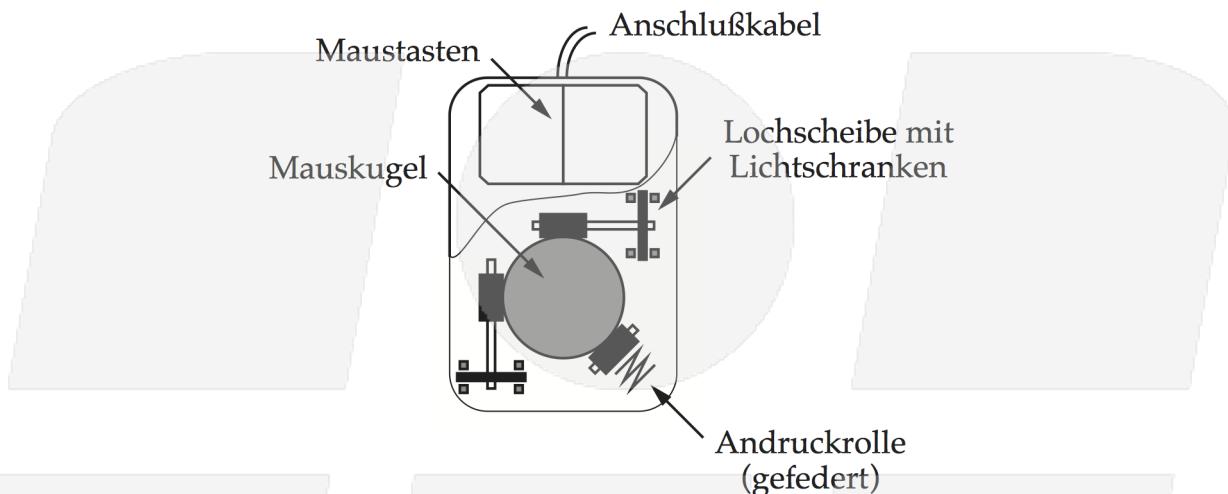


Abbildung 10.8: Aufbau einer opto-mechanischen Maus

einer Richtung um einen halben Lochabstand versetzt an den Lochscheiben angeordnet (Abbildung 10.10).

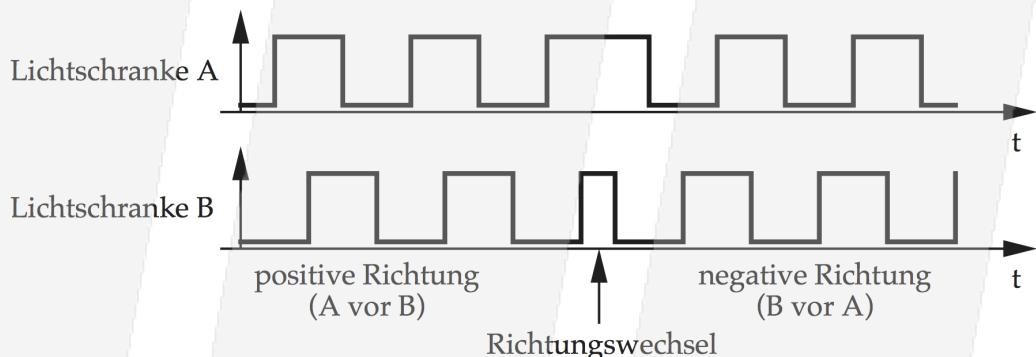


Abbildung 10.9: Signalverlauf an den beiden Fotosensoren einer Bewegungsrichtung

Die Auswertung der vier Signale würde den Hauptprozessor stark beladen, weshalb dafür ein eigener Mikrocontroller eingesetzt wird. Dieser befindet sich entweder in der Maus selbst, die dann an eine serielle Schnittstelle angeschlossen wird (serielle Maus) oder auf einer Platine, die über einen Steckplatz an den CPU-Bus des Computers angeschlossen wird (Busmaus). Die serielle Maus versorgt über die bei den meisten Rechnern vorhandene serielle Schnittstelle eine Steueroftware, den so genannten Maustreiber, mit den relativen Koordinaten der Mausbewegung.

Die ersten optischen Mäuse benötigten im Gegensatz zur opto-mechanischen Maus eine spezielle Unterlage (*Mousepad*), auf der ein Gittermuster aus horizontalen und vertikalen Linien aufgedruckt ist. Beim Bewegen der Maus wird dieses Raster abgetastet. Zwei versetzt angeordnete Fotosensoren detektieren die Anzahl der bei einer Verschiebung der Maus in horizontale oder vertikale Richtung überfahrenen Linien. Die Auswertung der Signale erfolgt dann wie bei der seriellen Maus. Heutzutage benutzen optische Mäuse eine kleine Kamera um ca. 1500 Bilder pro Sekunde aufzunehmen. Um auf nahezu jeder Unterlage funktionieren zu können, besitzt die Maus zur Beleuchtung des Untergrundes eine rote LED. Ein CMOS-Sensor (Complementary Metal-Oxide Semiconductor) nimmt die Bilder auf, die von einem DSP (Digital Signal

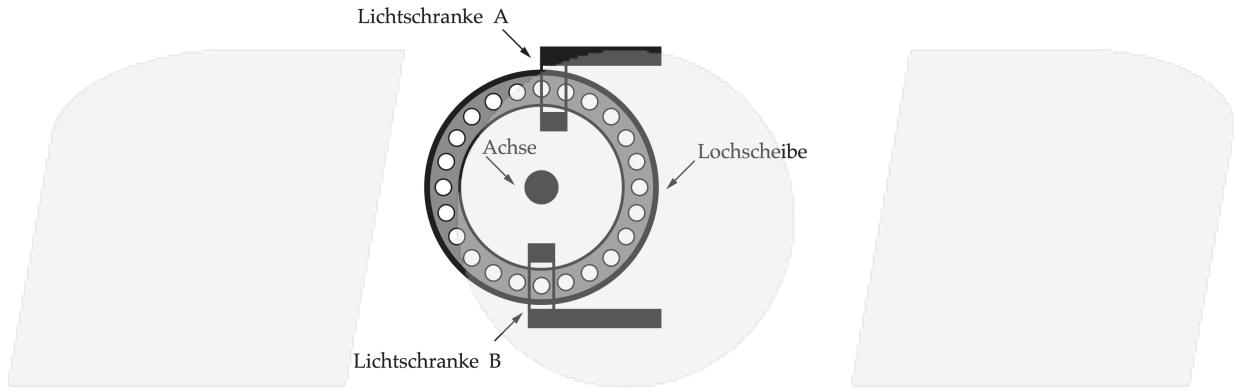


Abbildung 10.10: Optische Abtastung an einer der beiden Lochscheiben

Processor) verarbeitet werden. Der DSP (arbeitet mit ca. 18 MIPS) erkennt Muster und vergleicht die Verschiebung der Muster zu vorhergehenden Bildern. Daraus errechnet der DSP die Richtung und Strecke, um die die Maus bewegt wurde. Der Vorteil dieser Mäuse im Gegensatz zu Mäusen mit einer mechanischen Kugel liegt darin, dass die Maus auf nahezu jedem Untergrund zuverlässig funktioniert und dabei nicht schmutzanfällig ist. Durch den hohen Rechenaufwand liegt die Stromaufnahme optischer Mäuse mit ca. 70 mA jedoch auch deutlich über der herkömmlicher Mäuse (ca. 10 mA).

Eine Kenngröße für Mäuse ist deren Auflösung. Sie wird, wie bei Druckern, in Punkten pro Zoll angegeben (*dpi*), und äußert sich durch den kleinstmöglichen Bereich, auf dem die Maus bewegt werden kann, um eine bestimmte Anzahl Punkte auf dem Bildschirm zu überstreichen.

Beispiel: Eine Maus mit 400 dpi kann auf einer Arbeitsfläche von minimal $2'' \times 1,5''$ (ca 5 cm \times 4 cm) einen Bildschirm von 800×600 Punkten ansprechen. Die hohe Empfindlichkeit kann durch den Maustreiber beliebig herabgesetzt werden, indem erst mehrere Impulse eine Bewegung des Mauszeigers um einen Punkt auf dem Bildschirm bewirken (d.h. die Arbeitsfläche wird größer). Mäuse mit hoher Auflösung erfordern eine besonders präzise Mechanik, hochwertige Fotosensoren und schnelle Mikrocontroller.

Trackballs finden ihren Einsatz überall dort, wo der Platz für eine Navigation mit der Maus nicht ausreicht. Der mechanische und elektronische Aufbau gleicht dem einer umgedrehten Maus.

10.2 Busse und Schnittstellen

10.2.1 Einführung

Dieser Abschnitt befasst sich sowohl mit *Bussen*, die zur Informationsübertragung zwischen mehreren Geräten oder zwischen einzelnen Baugruppen eines Systems dienen, als auch mit *Schnittstellen* (*Interfaces*), die für den physikalischen Zugriff auf ein Bussystem erforderlich sind.

Diese Kommunikationsstruktur ist schematisch in Abbildung 10.11 dargestellt. Alle Komponenten sind über entsprechende Schnittstellen mit einem aus Bussen bestehenden Verbindungsnetz miteinander verbunden. Das Verbindungsnetz ist gegeben durch eine physikalische Möglichkeit zur Übertragung der Daten und ein *Protokoll*. Dieses besteht aus einem Satz von Regeln,

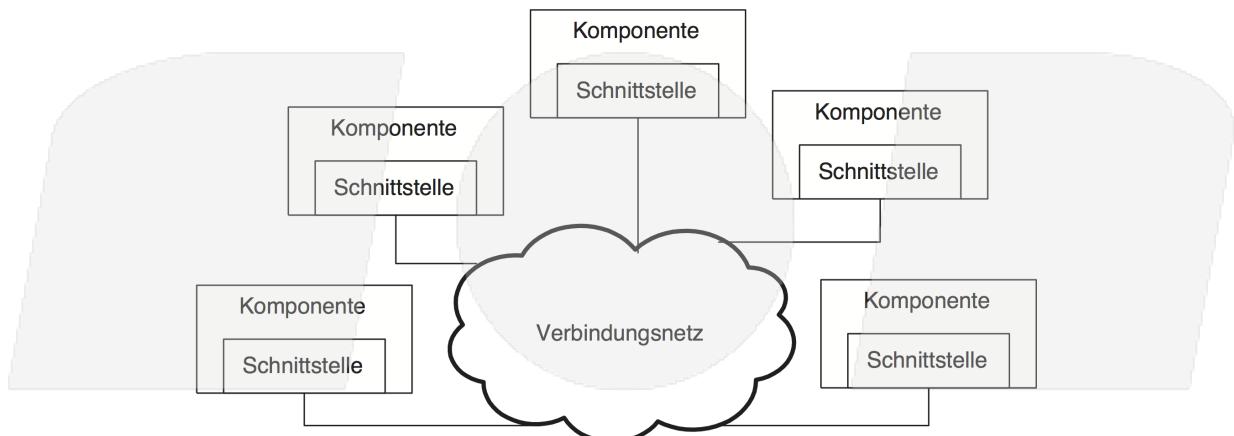


Abbildung 10.11: Kommunikation der Komponenten eines Arbeitsplatzrechners

die die Art und Weise der Übertragung der Daten festlegen. Jedes Bussystem verwendet ein spezifisches Protokoll, das von den angeschlossenen Systemkomponenten verstanden wird. Das heißt, die Schnittstellen übersetzen die vom Bus übertragenen Daten für den internen Gebrauch der Komponenten. Verschiedene Busse werden über Koppelmodule miteinander verbunden, die sowohl die Hardware-Umsetzung als auch die Übersetzung der Protokolle gewährleisten. Auf verschiedene Verbindungen zwischen gleichen und unterschiedlichen Bussystemen wird im weiteren Verlauf dieses Abschnitts näher eingegangen.

Physikalisch kann das Verbindungsnetz auf verschiedenste Arten realisiert werden. Dedizierte Verbindungswege zwischen allen Komponenten (point-to-point) und ein einziger, gemeinsamer Verbindungsweg für alle Komponenten bilden dabei die Extrema. Letzteres wird auch *physikalischer Bus* genannt und besteht im einfachsten Fall aus einer größeren Anzahl Leitungen zur Übertragung von Adressen, Daten und Steuerinformationen. Bei vielen Buskomponenten führt diese Struktur jedoch zu Engpässen. Verallgemeinerte Busse, die dieses Problem lösen sollen, werden als *logische Busse* bezeichnet. Sie sind gekennzeichnet durch ein gemeinsames Verbindungsnetz für alle Komponenten, eine eindeutige Adressierung der Kommunikationspartner sowie einem Verfahren zur Vermeidung von Konflikten beim Zugriff auf eine Komponente. Eine bestimmte Topologie ist nicht in der Definition logischer Busse inbegriffen. Im Folgenden wird auf einige Aspekte der logischen Busse detaillierter eingegangen.

10.2.2 Grundlagen der Bussysteme

Überblick der Busarten

Wie bereits erwähnt, dient ein Bus zur Verbindung mehrerer Systemkomponenten mittels Leitungen. Dabei kann zu jeder Zeit nur eine Komponente Daten senden. Diese können hingegen von genau einer, mehreren (multi-cast) oder allen anderen (broad-cast) Komponenten empfangen werden. Die beiden grundsätzlichen, physikalischen Konfigurationen der dazu verwendeten Bussysteme heißen *bit-parallel* und *bit-seriell*.

Abbildung 10.12 zeigt die Struktur eines so genannten *parallelen Busses*. Er überträgt Signale getrennt auf drei verschiedenen Teilbussen: Adressbus, Datenbus und Steuerbus, die entsprechend zur Übertragung von Adressen, Daten und Steuersignalen dienen. Weiterhin werden die

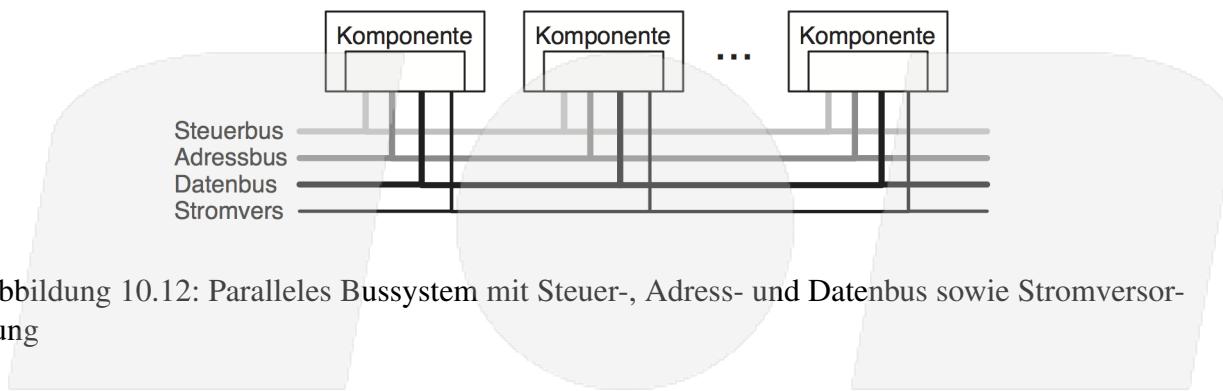


Abbildung 10.12: Paralleles Bussystem mit Steuer-, Adress- und Datenbus sowie Stromversorgung

Komponenten über gesonderte Leitungen mit Strom versorgt. Der Datenbus besteht typischerweise aus 4, 8, 16, 32 oder 64 parallelen Datenleitungen. Um Anschlüsse und Leitungen zu sparen, werden Adressen und Daten häufig im Zeitmultiplex auf einem Bus übertragen, d. h. sie werden nacheinander über die gleichen Leitungen gesendet. Man spricht dann von einem *Multiplexbus*.

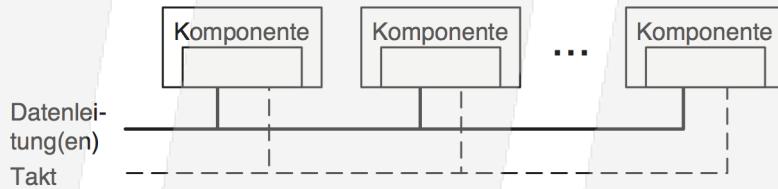


Abbildung 10.13: Serielles Bussystem mit einer oder mehreren Datenleitungen sowie optionaler Taktleitung

Der in Abbildung 10.13 dargestellte *serielle Bus* kommt im einfachsten Fall mit zwei Leitungen aus: Einer Datenleitung, über die sowohl Adresse, Daten als auch Steuerinformationen gesandt werden und einer Masseleitung. Die Datenleitungen können auch nach Funktionen getrennt sein, eine Leitung für Daten, eine für Adressen, eine für Kontrollinformationen, etc. Wird ohne eine besondere Taktleitung gearbeitet, spricht man von einem *asynchronen Bus*. Die Synchronisation zwischen Sender und Empfänger eines Datenpaketes muss hier über Steuerinformationen geregelt werden. Beim *synchronen Bus* geschieht die Synchronisation entweder physikalisch, d. h. durch eine zusätzliche Taktleitung oder durch einen in den Daten enthaltenen Takt. Bei dieser zweiten Methode werden die Daten in so genannten Paketen verschickt, die außer den Daten noch Steuerinformationen, Sender- und Empfängeradresse und Prüfzeichen enthalten.

Systemkomponenten, die über einen Bus Daten senden oder empfangen können, werden als Busteilnehmer oder Knoten (nodes) bezeichnet. Dabei ist zwischen Knoten, die selbständige Buszugriffe durchführen können, den sogenannten *aktiven Knoten (Master)*, und den *passiven Knoten (Slave)*, deren Buszugriffe von einem Master gesteuert werden, zu unterscheiden. An einem *Multimaster-Bus* sind mehrere Master erlaubt, deren Buszugriffe von einem Busarbiter geregelt werden. Der Auslöser einer Datenübertragung wird dabei Initiator, die Kommunikationspartner Target genannt.

Eingangs ist bereits erwähnt worden, dass das Bussystem ein Flaschenhals der Systemleistung in modernen Arbeitsplatzrechnern darstellt. Diese Situation soll durch ein hierarchisch strukturiertes Bussystem verbessert werden. Ein typisches solches Bussystem ist in Abbildung 10.14

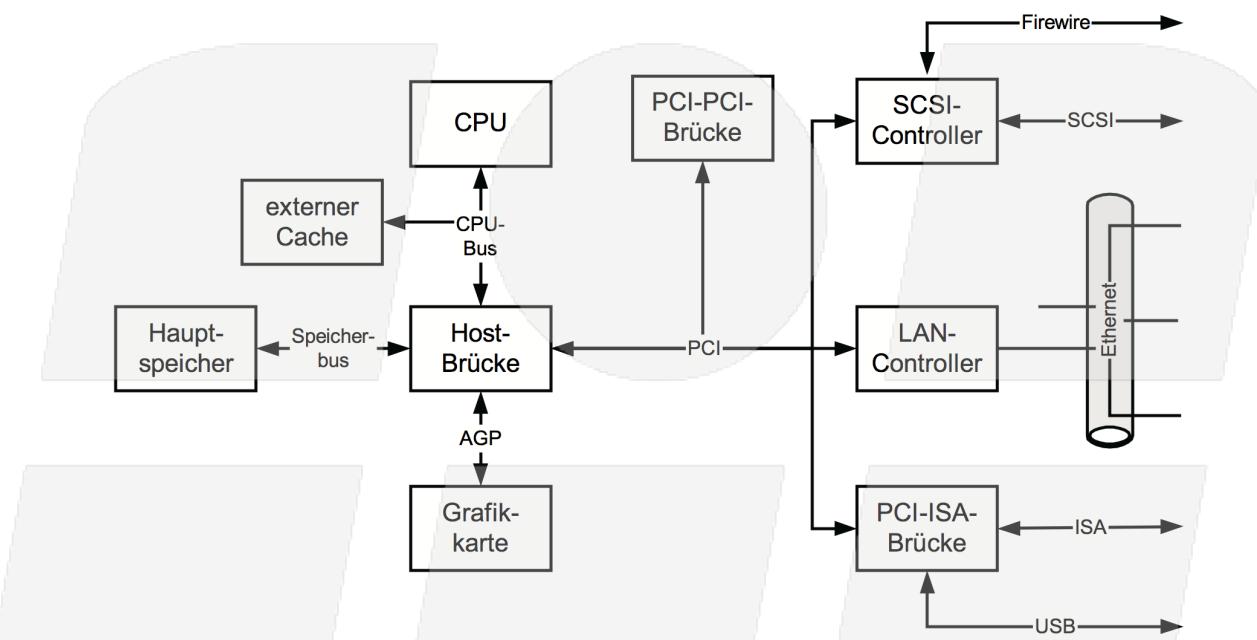


Abbildung 10.14: Typisches hierarchisches Bussystem.

dargestellt. Die verschiedenen Busarten werden nach ihrer Verwendung im System bezeichnet:

- *Register-Bus*
Sehr schneller Bus innerhalb der CPU (on-chip), der Register, ALU und CU miteinander verbindet.
- *System-Bus* (auch: *CPU-Bus*)
Parallelbus mit direktem Anschluss an die CPU-Signale, gesteuert durch den Prozessor, einen Cache- oder DMA-Controller. Dieser Bus ist zwar sehr schnell, erlaubt jedoch nur sehr kleine Distanzen bei kleiner kapazitiver Belastung.
- *Speicherbus*
Schneller Parallelbus zur Verbindung von CPU und Hauptspeicher. Arbeitet häufig auch mit der gleichen Datenübertragungsrate wie der CPU-Bus. Der Speicherbus ist jedoch nicht direkt an die CPU-Signale angeschlossen, sondern über die Host-Brücke.
- *Peripheriebus* (z. B. PCI-Bus, AGP, vgl. Abschnitt 10.2.3)
Dient der Anbindung interner und externer Systemkomponenten an den CPU-Bus. Durch die größere Reichweite sinkt die Datenübertragungsrate im Vergleich zum CPU-Bus. Er verbindet Komponenten, die oft sehr unterschiedliche Bandbreiten benötigen. Die Kopplung zwischen CPU- und Peripherie-Bus erfolgt durch die *Host-Brücke*.
- *Ein-/Ausgabebus* (z. B. USB, SCSI, ... vgl. Abschnitt 10.2.4)
Dienen zum Anschluss von Ein-/Ausgabegeräten sowie Massenspeichern an das System. Die geläufigsten Typen sind: ISA, MCA, SCSI, USB, FireWire. Alle sind über Brücken oder Controller mit dem PCI-Bus verbunden.

Bustopologien

Abbildung 10.15 zeigt die grundlegenden Bustopologien. Der bereits beschriebene physikalische Bus ist die in Arbeitsplatzrechnern am häufigsten eingesetzte Topologie (Strang). Alle Teilnehmer sind hier an die gleiche Signalleitung angeschlossen und nur ein Teilnehmer kann jeweils Daten übertragen. Durch das Zusammenschließen mehrerer physikalischer Busse (gleich oder verschieden) mittels Repeatern bzw. Brücken entsteht ein so genannter *segmentierter Bus* (Die Funktionsweisen der Koppeleinheiten zwischen Bussen sind in Tabelle 10.1 kurz aufgelistet).

Bei der *Ringtopologie* sind alle Knoten unidirektional mit ihren direkten Nachbarn verbunden. Jeder Teilnehmer dient dabei als Repeater, d. h. das Eingangssignal wird verstärkt und resynchronisiert an die Nachbarn weitergeleitet. Bei einem *Stern* ist jeder Teilnehmer einzeln an einen Vermittlungsknoten angeschlossen. Die Kommunikation findet über den entweder passiven oder aktiven Vermittlungsknoten statt. Im ersten Fall sind die Komponenten für die Steuerung der Verbindungen zuständig. Ein aktiver Knoten übernimmt die Steuerung selbst und führt Zusatzfunktionen wie z. B. die Zwischenspeicherung der Daten aus. Vermittlungsknoten können nur eine oder auch mehrere Verbindungen herstellen. Im zweiten Fall spricht man von *Switches*. Die Stern-Strang-Topologie verbindet mehrere Vermittlungsknoten von Sternen miteinander. Eine *Baum-Topologie* verbindet die Komponenten, die hier als Blätter des Baumes zu verstehen sind, bidirektional über sogenannte Hubs.

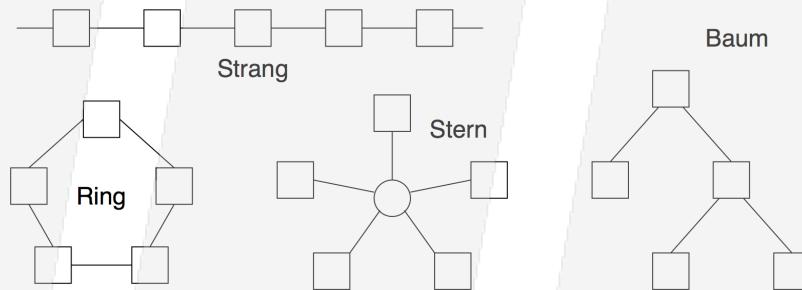


Abbildung 10.15: Grundlegende Bustopologien

Koppeleinheiten

Tabelle 10.1 gibt einen Überblick über die verschiedenen Koppeleinheiten.

Die einfachste Art der Kopplung ist die zwischen zwei identischen Bussen. Die hierzu verwendete Einheit heißt *Repeater*. Ein Repeater verstärkt das Signal elektrisch, regeneriert es zeitlich und stellt die korrekte Synchronisation sicher, ohne die Daten dabei zu puffern. Auf beiden Seiten des Repeaters werden gleiche Adressierung und gleicher Adressraum verwendet.

Ein *Hub* dient zum Aufbau von Baum-Topologien. Wie der Repeater verbindet er identische Busse mit gleicher Adressierung und gleichem Adressraum. Hier wird jedoch ein übergeordneter Bus (Upstream) mit darunterliegenden Bussen (Downstream) verbunden. Die Übertragung der Daten wird von einem Hub-Controller organisiert. Vom Upstream- zu den Downstream-Ports werden die Daten im Broadcast übertragen. In umgekehrter Richtung ist zusätzlich die alleinige Weiterleitung der Daten an den Upstream-Bus möglich (Point-to-Point). Durch gezieltes An- und Abschalten sowie Zurücksetzen von Ports baut das Hub selbstständig Verbindungen auf oder bricht diese ab. Weiterhin registriert es den Anschluss bzw. das Abtrennen von

Tabelle 10.1: Funktionen einiger Koppeleinheiten

Koppeleinheit	Funktion
Repeater	Verbindung gleicher Busse, Verstärkung und Resynchronisation des Signals
Hub	Repeater, der auf einer Seite mehrere Anschlüsse besitzt (z. B. für Baum-Topologie)
Brücke	Verbindung gleicher oder unterschiedlicher Busse, ggf. Umsetzung des Protokolls und der Adressierung für den anderen Bus
Switch	Ähnlich der Brücke, kann mehrere Verbindungen gleichzeitig aufbauen.

Busteilnehmern während des Betriebs (Hot Plugging) sowie das Auftreten von Fehlern. Diese Informationen werden an die zuständige Netzkomponente weitergegeben.

Die *Brücke (Bridge)* ist die komplexeste Koppeleinheit zwischen gleichen oder verschiedenen Bussen. Ihre Hauptbestandteile sind die busspezifischen Schnittstellen, Pufferspeicher und eine Steuerungseinheit. In den Pufferspeichern findet eine zeitliche und insbesondere eine protokollspezifische Anpassung der Signale statt. Dies bezieht sich überwiegend auf eine Anpassung der Adressierung sowie von Synchronisierungs- und Kontrolldaten. Über eine Brücke werden dabei nur Daten übertragen, die für dahinterliegende Knoten bestimmt sind. Auf beiden Seiten der Brücke können unterschiedliche Adressräume verwandt werden. Zusätzliche Komponenten einer Brücke können sein: Cache-Controller, Speicher-Controller und weitere Schnittstellen.

Ein *Switch* zeichnet sich dadurch aus, dass er mehrere Verbindungen zwischen Komponenten oder Bussen gleichzeitig verwalten kann. Ein Beispiel ist der Vermittlungsknoten eines sternförmigen Busses.

Adressierung der Busteilnehmer

Je nach verwendeter Busarchitektur werden verschiedene Verfahren zur Selektion von Targets durch einen aktiven Knoten verwandt. Bei einfachen Mikroprozessor-Systemen, in denen der Prozessor die einzige aktive Komponente ist, oder bei Brücken (z. B. PCI, vgl. 10.2.3) wird jede ausgegebenen Adresse durch einen zentralen Adressdecoder ausgewertet (Abb. 10.16a). Dieser selektiert die den Adressen entsprechenden Komponenten. Eine Variante dieses Verfahrens ist der dezentrale Adressdecoder (Abb. 10.16b). Hier besitzt jede Komponente einen eigenen Dekoder, der den entsprechenden Adressraum kennt. Bei Multiplexbussen kann jeder Komponente eine der Datenleitungen als Selektionsleitung zugewiesen werden (Abb. 10.16c).

In seriellen Bussen enthalten die über den Bus verschickten Datenpakete auch die Empfängeradresse. Diese wird in allen Komponenten decodiert. Adressierungen sind im Point-to-Point-, Multicast- und Broadcast-Modus möglich.

Bei Baum-Topologien ist zu unterscheiden, ob Kommunikation nur zwischen einer Komponente und dem Wurzelknoten (z. B. USB) oder zwischen allen Komponenten möglich ist (z. B. FireWire). Im ersten Fall versendet der Wurzelknoten die Daten als Broadcast und die angefragte Komponente quittiert den Empfang (Acknowledge). Die anderen Knoten senden ih-

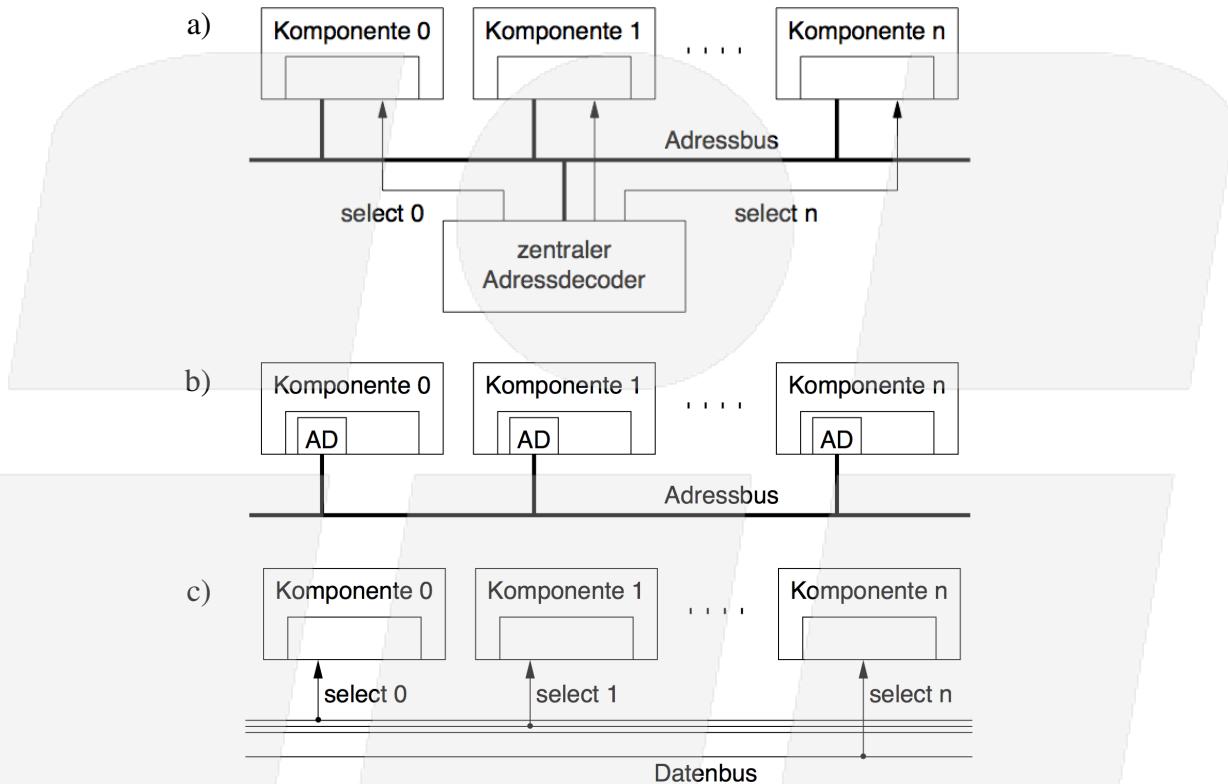


Abbildung 10.16: Adressierungsarten der an Parallelbussen angeschlossenen Komponenten: a) mit zentralem Adressdecoder; b) mit dezentraler Adressdecodierung (AD = Adressdecoder); c) Datenleitungen als Selektionsleitung

re Daten ausschließlich zum Wurzelknoten. Im zweiten Fall verschickt jeder beliebige Knoten Nachrichten im Broadcast an seinen Vaterknoten und seine Söhne, die das gleich Verhalten zeigen. Damit wird die Nachricht als Broadcast über den gesamten Baum versandt.

Busarbitrierung

Unter *Busarbitrierung* versteht man die Auflösung von Konflikten bei gleichzeitigem Zugriff mehrerer Busmaster auf ein Bussystem. Im Allgemeinen schicken Komponenten, die auf den Bus zugreifen wollen eine Anforderung (*Bus Request*, BRQ) an den *Arbiter* („Schiedsrichter“), der nach bestimmten Regeln einen der Bewerber auswählt und ihm eine Zugriffsgenehmigung (*Bus Grant*, BGR) erteilt. Der Bus darf jedoch erst verwendet werden, wenn ein BUSY-Signal anzeigt, dass keine Komponente mehr auf ihn zugreift.

Bei dem Verfahren der *unabhängigen Anforderung* (Abb. 10.17a) entscheidet ein zentraler Arbiter, der mit jeder Komponente über eine BRQ- und eine BGR-Leitung verbunden ist. Bei einem *dezentralen Arbiter* besitzt jede Komponente einen Arbiter, die untereinander mit BRQ-Leitungen verbunden sind (Abb. 10.17b). Alle Komponenten, die nicht die höchste Priorität besitzen, ziehen ihre Anforderung zurück. So erhält die Komponente mit höchster Priorität automatisch den Zugriff. Dieses Verfahren ist sehr aufwändig und wird daher nur für Systeme mit wenigen Komponenten verwendet.

Das so genannte *Daisy-Chain-Verfahren* (Abb. 10.17c) arbeitet mit einem zentralen Arbiter, der in einen Busmaster (typischerweise der Prozessor oder eine Brücke) integriert sein kann. Alle Komponenten senden über eine Leitung ihre Anforderungen, worauf der Arbiter ein BGR-Signal an die erste Komponente sendet. Will diese nicht zugreifen, reicht sie das Signal an ihre „rechte“ Nachbarkomponente weiter. Will auch diese nicht zugreifen, reicht sie das Signal wiederum an ihre rechte Nachbarkomponente weiter, usw. Die Priorität der Komponenten hängt also von ihrer Entfernung zum Arbiter ab. Eine Variante verzichtet ganz auf den Arbiter: Der ersten Komponente wird ständig ein BGR gesandt, und ggf. weitergereicht. Die Zugriffe werden dann maßgeblich durch Abfrage der BUSY-Leitung geregelt (Abb. 10.17d).

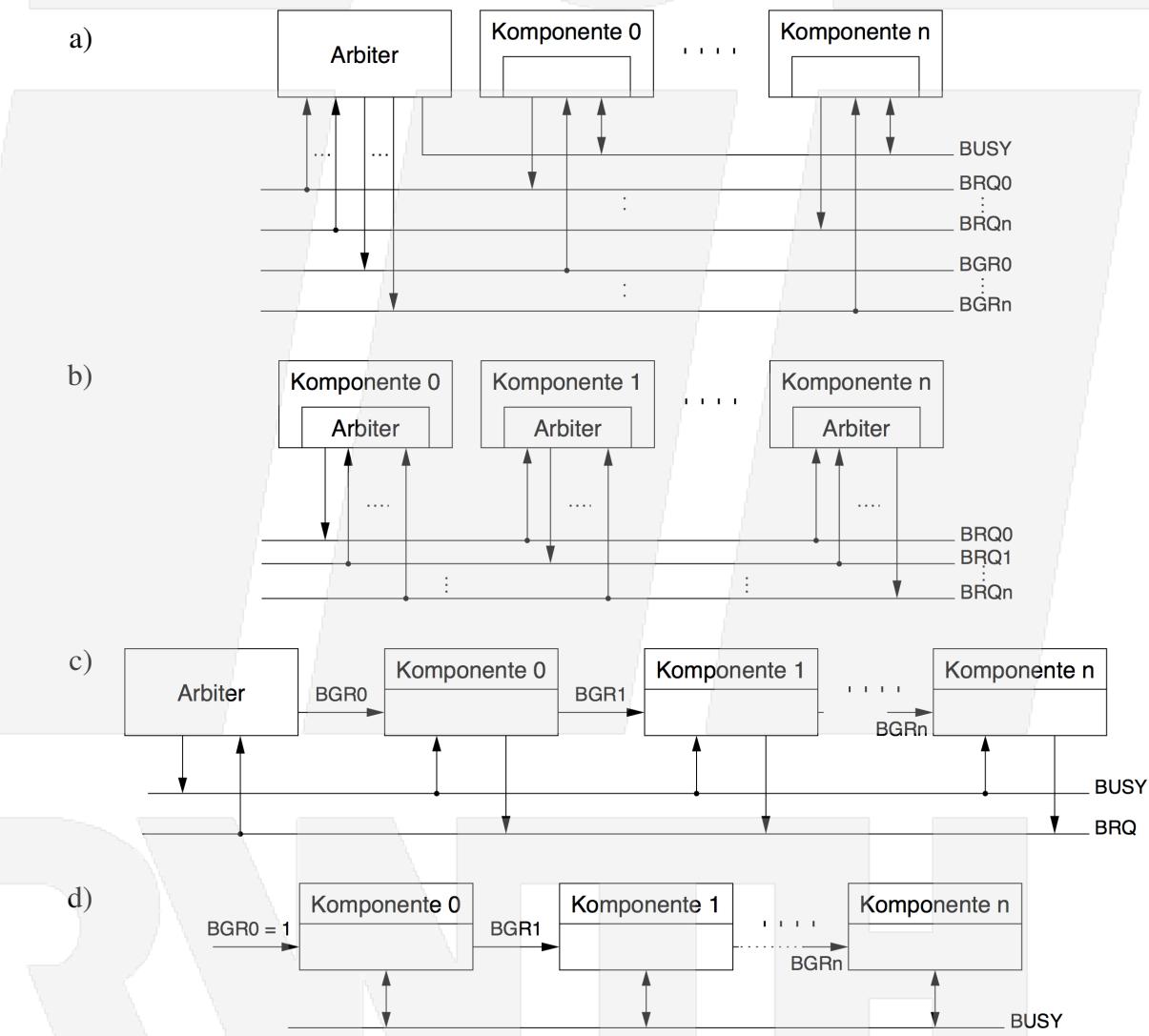


Abbildung 10.17: Verschiedene Arten der Busarbitrierung: a) mit unabhängiger Anforderung; b) mit dezentralem Arbiter; c) Daisy-Chain-Verfahren; d) Daisy-Chain-Verfahren ohne Arbiter

Bei Baum-Topologien gibt es wiederum zwei Verfahren: Hat lediglich der Wurzelknoten Zugriff auf den Bus, wird das so genannte *Polling-Verfahren* verwendet. Dabei versendet der Host-Knoten regelmäßig Anfragen nach Zugriffswünschen an alle Komponenten, auf deren Antwort

er dann entsprechend reagiert. Bei Baumnetzen, in denen alle Komponenten Buszugriff besitzen, senden diese Zugriffsanforderungen an den Hostknoten, der diese erteilt oder ablehnt.

10.2.3 Beispiele für Peripherie-Busse

PC-AT-Bus (ISA, Industrie-Standard-Architektur)

Der *ISA-Bus* wurde 1985 von IBM aus der Taufe gehoben und stellt seitdem den Industriestandard für auf Intel x86 Prozessoren basierende Systeme dar.

Der ISA-Bus ist ein mit 8 MHz getakteter synchroner Bus mit 16 Daten und 24 Adressleitungen. Somit lassen sich Daten in einem Adressraum von 16 MByte mit theoretisch 8 MByte/s (2 Takte pro 16bit Wort) übertragen. Tatsächlich beträgt die maximale Datenrate jedoch nur ca. 6 MByte/s. Dies wird dadurch bedingt, dass die Kompatibilität zum Vorgänger, dem XT-Bus (8Bit, 4.77 MHz Takt) beibehalten werden muss.

Mit dem Aufkommen der 32 Bit Prozessoren (Intel 386 und Nachfolger) zeigte sich das neben der niedrigen Datenübertragungsrate wesentliche Handicap des ISA Busses: Daten die von einem ISA-Busteilnehmer in einen Adressbereich außerhalb der ersten 16 MByte übertragen werden sollten, benötigten dafür eine zusätzliche Übertragung die nur von der CPU durchgeführt werden konnte. Dadurch verringerte sich die effektive Übertragungsrate des Busses abermals.

PCI-Bus (Peripheral Component Interconnect)

Da weder der ISA-Bus noch der daraus entwickelte EISA-Bus (Enhanced ISA) leistungsfähig genug für die Pentium Prozessoren war, entwarf Intel den *PCI-Bus*. Das Schema eines PCI-basierten Rechnersystems zeigt Abb. 10.18.

Der PCI-Bus zählt zu den Peripheriebussen und ist vom CPU-Bus über die CPU-Bridge entkoppelt. Ziele dabei sind Prozessorunabhängigkeit, Anwenderfreundlichkeit und Erweiterbarkeit. PCI-Busse werden nicht nur in Pentium-basierten Systemen eingesetzt, sondern sind auch Bestandteil der CHRP-Spezifikation für PowerPC-Systeme.

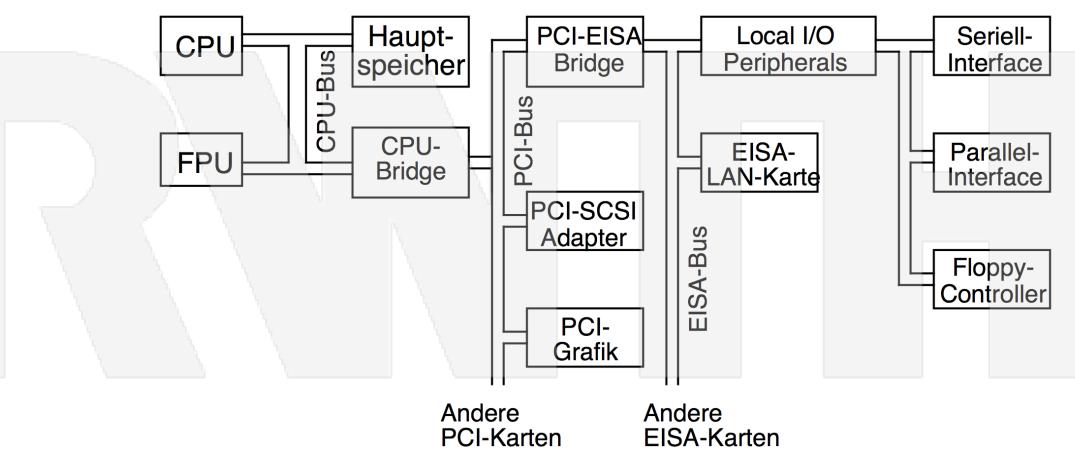


Abbildung 10.18: Schema eines PCI-Bus basierten Rechnersystems

Topologie:

Der PCI-Bus ist normalerweise in eine komplexe Struktur von Bussen eingebunden (Abb.

10.18). Die wichtigste Koppeleinheit ist dabei die Host-Brücke, die den PCI-Bus mit dem CPU-Bus verbindet. Die Brücke enthält außerdem den zentralen Busarbiter. Es ist außerdem möglich, den PCI-Bus über eine Standardbus-Brücke mit anderen Peripheriebussen wie (E)ISA, Micro-channel oder andere PCI-Busse anzukoppeln. Schließlich lassen sich über spezielle Brücken noch E/A-Busse wie SCSI oder dem VME-Bus ankoppeln.

Adressierung der Busteilnehmer:

Es gibt zwei mögliche Modi zur Adressdecodierung, mit denen ein Teilnehmer arbeiten kann. Normalerweise arbeiten Komponenten im so genannten *positiven Decodiermodus*. Sie erkennen selbstständig, ob sie angesprochen sind (dezentrale Adressdecodierung). Allerdings kann es maximal eine Einheit geben, die im so genannten *subtraktiven Decodiermodus* arbeitet. Diese reagiert auf alle Adressen, mit denen keine andere PCI-Komponente angesprochen wird.

Busarbitrierung:

Der PCI-Bus arbeitet mit dem Verfahren der unabhängigen Anforderung. Genaueres ist nicht spezifiziert, nur dass eine „faire“ Zuteilung realisiert werden muss und keine Blockaden auftreten dürfen.

Erweiterung für Grafikanwendungen:

Bei Anwendungen in der Computergrafik werden sehr hohe Übertragungsraten benötigt, die einen PCI-Bus sehr stark belasten. Deshalb gibt es eine auf diese Anwendungen spezialisierte Erweiterung, den *Accelerated Graphics Port*, kurz *AGP*. Er ist in die Host-Brücke integriert.

Ein PCI-Bus verfügt über 32 oder 64 Leitungen, über die 32 oder 64 Bit breite Adress- und Datenwörter im Multiplexverfahren übertragen werden. Die Breite der Datenwörter ist dabei unabhängig von der Anzahl der Leitungen. Der Bus kann im Burst-Modus arbeiten und erreicht dann bei einer Taktrate von 33 MHz und 32 Bits Busbreite eine Übertragungsrate von 132 MByte/s. Die in der Revision 2.1 des PCI-Standards definierte Busbreite von 64 Bits bei max. 66 MHz Takt erlaubt sogar eine Übertragungsrate von max. 528 MByte/s, der Adressraum vergrößert sich auf 9 TByte.

Weiterhin ist der PCI-Bus in der Lage, Transfers ohne das Zutun der CPU durchzuführen (Multimasterfähigkeit). So kann die CPU während eines PCI-PCI Transfers weiterarbeiten und z.B. auf den Hauptspeicher zugreifen.

PCI-Express

Mit stetig wachsenden Anforderungen an das Bussystem durch schnelle Netzwerkkarten, hochwertige Soundkarten und leistungsfähige Grafikkarten gelangt der PCI-Bus allmählich an seine Grenzen. Mit den neuen Intel-Chipsatzserien i915 und i925 soll der *PCI-Express* genannte Nachfolger eingeführt werden und sowohl den bisherigen PCI-Bus als auch die AGP-Schnittstelle für Grafikkarten ablösen. Dabei werden einige grundlegende Veränderungen vorgenommen. So erfolgt hier die Datenübertragung seriell über sog. „Lanes“ anstatt wie bisher parallel. Dabei handelt es sich um Leitungspaare aus zwei unidirektionalen Leitungen für Senden und Empfangen, wodurch Voll-Duplex Betrieb mit jedem angeschlossenen Peripheriegerät ermöglicht wird. Die verschiedenen Versionen des PCI-Express-Busses definieren sich über die Anzahl paralleler Lanes, ihre Bezeichnungen sind PCI E x1, x4, x8 und x16 mit 1 bis 16 Lanes. Bei mehreren parallelen Lanes wird das serielle Datensignal sequenziell auf die einzelnen Lanes aufgeteilt, wodurch sich die Datenrate entsprechend vervielfacht. PCI E x1 soll den bisherigen PCI-Bus ablösen, während PCI E x16 als neue Grafikschnittstelle (*PEG - PCI-Express for Graphics*) vorgesehen ist. Die Grundfrequenz des PCI E - Systems beträgt 2,5 GHz. Da die Daten

aber zwecks Synchronisation von Sender und Empfänger kodiert übertragen werden (8 Bit auf 10 Bit Umsetzung) ergibt sich eine nutzbare Datenrate von $2\text{Gbit/s} = 250\text{ MB/s}$ pro Richtung und Lane. Daraus folgt eine maximale Datenrate von 500 MB/s in beide Richtungen bei PCI E x1, was circa das Vierfache gegenüber herkömmlichem 32 Bit PCI bedeutet. Für Grafikwendungen bietet PCI E x16 somit bis zu 4 GB/s sowohl hin als auch zurück. Zum Vergleich: Die Datenrate von AGP 8x beträgt ca. 2 GB/s , was allerdings bisher auch für anspruchsvolle Anwendungen in der Regel völlig ausreichend ist. Weitere Neuerungen bei PCI-Express sind eine Erhöhung der maximalen Leistungsübertragung der Stromversorgung von 25 W bei AGP 8x auf 75 W sowie eine Vergrößerung des Konfigurationsspeichers von 256 Byte auf 4096 Byte . PCI-Express ist voll systemkompatibel zu PCI und AGP, es bietet also unter anderem identische Funktionen für die Initialisierungsphase des Betriebssystems, wodurch die Einführung unproblematisch ist.

PCMCIA

Die **PCMCIA** (Personal Computer Memory Card International Association) wurde 1989 mit dem Ziel gegründet, einen einheitlichen Standard für Computerkarten mit integrierten Schaltungen zu schaffen, der insbesondere Anforderungen an das *mobile Computing* (z. B. wenig Stromverbrauch, geringe Größe und Stoßfestigkeit) berücksichtigt. So wurde der als „**PCMCIA**“ bekannte Standard für PC-Steckkarten entwickelt, die eine Größe von $85,6 * 54\text{ mm}$ besitzen und mit einem 68-Pin-Connector ausgestattet sind. Der Standard ermöglicht sowohl die Anbindung von zusätzlichen Speicher als auch von Peripheriegeräten. Es sind 3 Typen definiert, die sich auf die Dicke der Karte beziehen (Typ I: $3,5\text{ mm}$, Typ II: 5 mm und Typ III: $10,5\text{ mm}$). Typ I-Karten sind in der Regel Speicherkarten (RAM, Flash oder SRAM), Typ II-Karten meistens I/O-Karten (Modem, LAN), während Typ III-Karten häufig für rotierende Massenspeicher (Festplatten) verwendet werden. Tabelle 10.2 erläutert die Daten des PCMCIA-Standards.

Tabelle 10.2: Kennwerte des PCMCIA-Standards

Busmastering	32Bit, bis zu 33MHz
DMA	direkte Nutzung von DMA möglich
eXecute In Place (XIP)	auf der Karte gespeicherte Software direkt ausführbar
Stromaufnahme	$3,3\text{V}/5\text{V}$ ($3,3\text{V}$ -Karten sind vor Überspannung geschützt)
Hot Plugging	Karten können zur Laufzeit eingesteckt bzw. abgezogen werden. Eine (De-)Installation erfolgt automatisch
Zoomed Video (ZV)	die Karte kann direkt auf den VGA-Controller zugreifen

10.2.4 Beispiele für E/A-Busse

SCSI-Bus (Small Computer System Interface)¹

Der **SCSI-Bus** dient dem Anschluss von Peripheriegeräten (Festplatten, CD-ROM, Scanner, Bandlaufwerken, ...) und kann, je nach verwendeter Norm 8 bzw. 16 verschiedene Geräte

¹gesprochen: „Skassi“

verwalten. Den Anschluss an die Hauptplatine übernimmt ein Hostadapter, der als SCSI-Gerät zählt. Die einzelnen Geräte besitzen eine SCSI-ID (0-7 bzw. 0-15), über die sie ansprechbar sind.

Der SCSI-Bus kann sowohl synchron als auch asynchron betrieben werden. Wegen der höheren Datenübertragungsrate ist allerdings der Synchronbetrieb üblich.

Der SCSI-Bus arbeitet kommandoorientiert. Dabei sendet ein Initiator ein Kommando (z. B. zum Lesen von Daten) an ein Target. Mit der Einführung des SCSI-2 Standards wurden Kommandomengen für die verschiedenen SCSI-Gerätetypen (Festplatten, Wechselmedien, Drucker, etc.) definiert. Diese Kommandos werden von einem SCSI-Controller interpretiert. Um die Gesamtleistung des Bussystems zu erhöhen, können die Busteilnehmer mehrere Kommandos speichern und in der Reihenfolge umsortieren. Dadurch kann z. B. eine Festplatte Daten, die bereits im Zwischenspeicher der Festplatte vorliegen, bevorzugt bearbeiten. Abbildung 10.19 verdeutlicht dies im vereinfachten Zustandsdiagramm. Ein Kommando des Initiators beeinflusst dabei zwei Zustandsautomaten des Targets. Nachdem das Kommando gepuffert wurde, aktiviert der Empfangsautomat den Bearbeitungsautomaten.

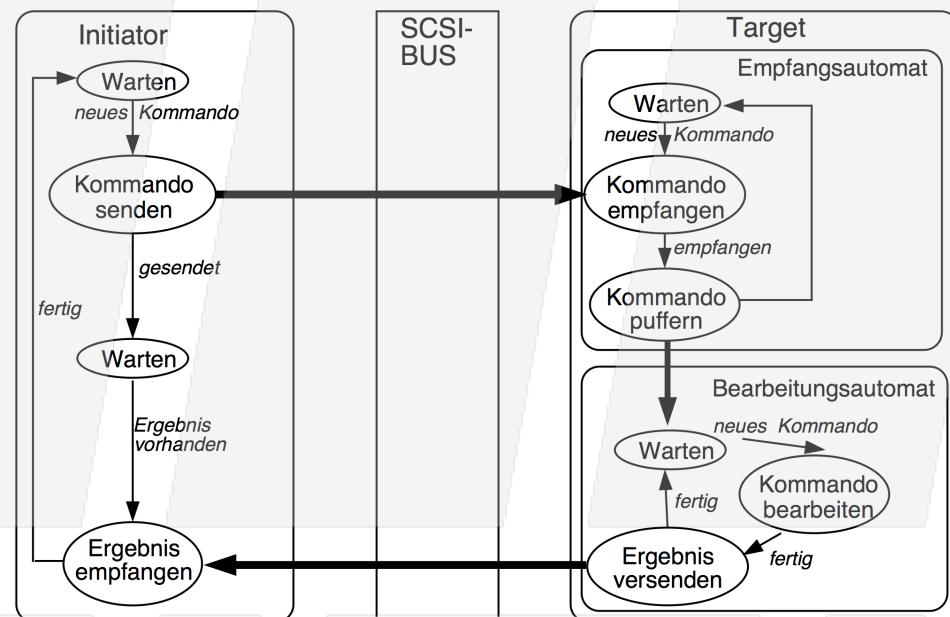


Abbildung 10.19: Kommandobearbeitung am SCSI-Bus

Während der Bearbeitung eines Kommandos kann das SCSI-Gerät den Bus freigeben, so dass der Bus anderen Busteilnehmern zur Verfügung steht. Obwohl verschiedene SCSI-Standards existieren (Tab. 10.3) können mittels Steckeradapters Geräte einer Norm an Hostadapters einer anderen Norm betrieben werden. Die SCSI-Geräte stimmen dazu untereinander mittels asynchroner Übertragung die Protokolle ab (Sync-Negotiation).

Die SCSI-1 Norm findet sich heute nur noch vereinzelt. Am meisten verbreitet sind Geräte der SCSI-2 Norm in den Ausprägungen „Fast“ (10MHz Takt im synchronen Betrieb) und „Ultra“ (20 MHz) mit jeweils 8 Bit oder 16 Bit breiten Bussen.

Topologie:

Ein SCSI-Bus ist ein Strang, an dem bis zu 16 Teilnehmer (inklusive Controller) angeschlossen sein können. Die Ankopplung an den Peripheriebus erfolgt über eine Brücke, den so genannten SCSI-Controller.

Tabelle 10.3: Kennwerte der verschiedenen SCSI Normen

Busbreite	SCSI-1	Transferrate (MByte/s)			
		SCSI-2			
		Fast	Ultra	Ultra 2	Ultra 3
8 (normal)	5	10	20	40	80
16 (Wide16)	10	20	40	80	160

Adressierung der Busteilnehmer:

In der Adressierungsphase („Selektion“) setzt das Gerät, das die Arbitrierung gewonnen hat, die der gewünschten ID und der eigenen ID entsprechenden Datenleitungen. Damit wird dem angesprochenen Gerät mitgeteilt, welches Gerät die Verbindung anfordert.

Busarbitrierung:

Die Arbitrierung erfolgt dezentral anhand der ID der angeschlossenen Geräte. Die höchste Priorität besitzt die ID 7 (der Controller). Bei 8 Bit breiten Bussen hat die ID 0 die niedrigste Priorität. Bei 16 Bit breiten Bussen ist die Prioritätenfolge 7, ..., 0, 15, ..., 8. Alle Geräte, die den Bus belegen wollen, setzen die ihrer ID entsprechende Datenleitung. Anschließend prüfen alle Geräte, ob es Busteilnehmer mit höherer Priorität gibt. Ist dies der Fall, wird die Anfrage zurückgezogen. Das Gerät mit der höchsten Priorität erhält den Bus.

USB-Bus (Universal Serial Bus)

Der **USB** hat zum Ziel, den Anschluss von Peripheriegeräten niedriger Datenrate wie Tastatur, Maus, Modems, Scanner, etc. zu vereinheitlichen und zu vereinfachen. Insbesondere werden kostengünstige Schnittstellen und Stecker verwendet. Das Kabel ist abgeschirmt und vier-adrig. Der USB-Stecker ist schematisch in Abbildung 10.20 dargestellt.

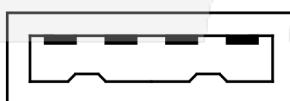


Abbildung 10.20: USB-Stecker

Der Anschluss der Geräte erfolgt über eine einheitliche Schnittstelle. Im Gegensatz zu den gebräuchlichen Bussystemen für diese Geräte (RS232, etc. siehe Kap. 10.2.5) können am USB bis zu 127 Geräte betrieben werden. Außerdem ist das Einfügen und Herausnehmen von Geräten im laufenden Betrieb möglich (Hot Plugging). Im Gegensatz dazu lässt sich eine einmal von einem PC abgezogene Tastatur erst nach einem Neustart wieder ansprechen. Der USB verfügt über zwei Datenübertragungsraten: Standard (Full Speed) bei 12 Mbit/s und langsam (Low Speed) bei 1,5 Mbit/s. Für die höhere Datenrate, müssen die Datenleitungen verdrillt sein. Geräte mit geringer Leistungsaufnahme können über den Bus mit Strom versorgt werden.

Topologie:

Die Busteilnehmer am USB sind in drei Gerätetklassen aufgeteilt:

- **Host:** Der Rechner, der die Kontrolle über den Bus hat und den Datenfluss steuert. In den meisten PCI Chipsätzen ist bereits ein Controller integriert.

- **Function:** Das an den USB angeschlossene Peripheriegerät. Jedes Peripheriegerät benötigt ebenfalls einen USB Controller.
- **Hub:** Ein Vermittlungsknoten, über den sich weitere USB Geräte an den Bus anschließen lassen. Hubs arbeiten als Zwischenverstärker (repeater) für die Signale und können Geräte mit Strom versorgen.

Diese Geräteklassen bilden zusammen einen logischen Bus mit Baumstruktur (vgl. Abbildung 10.21). Der Host fungiert als Wurzel-Knoten (root-node) und ist zuständig für die Initialisierung, Nummerierung und Adressierung der anderen Geräte. Die Hubs sind die inneren Knoten des Baums. Sie besitzen einen Upstream-Port zur Kommunikation Richtung Host und mehrere Downstream-Ports, an die weitere Hubs oder Endgeräte angeschlossen sein können. Die Functions schließlich sind die Blätter des USB-Baums. Sie bestehen aus mehreren logischen Endpunkten, wie z. B. für Gerätesteuerung, Interruptlogik und Ein-/Ausgabe. Die Schicht des Hosts nicht mitgezählt, darf der USB-Baum maximal sieben Schichten (Tiers) umfassen. Mit einer Maximallänge von 5 m pro Kabel ist die größte zulässige Entfernung zwischen Root-Knoten und einer Komponente gleich 35 m.

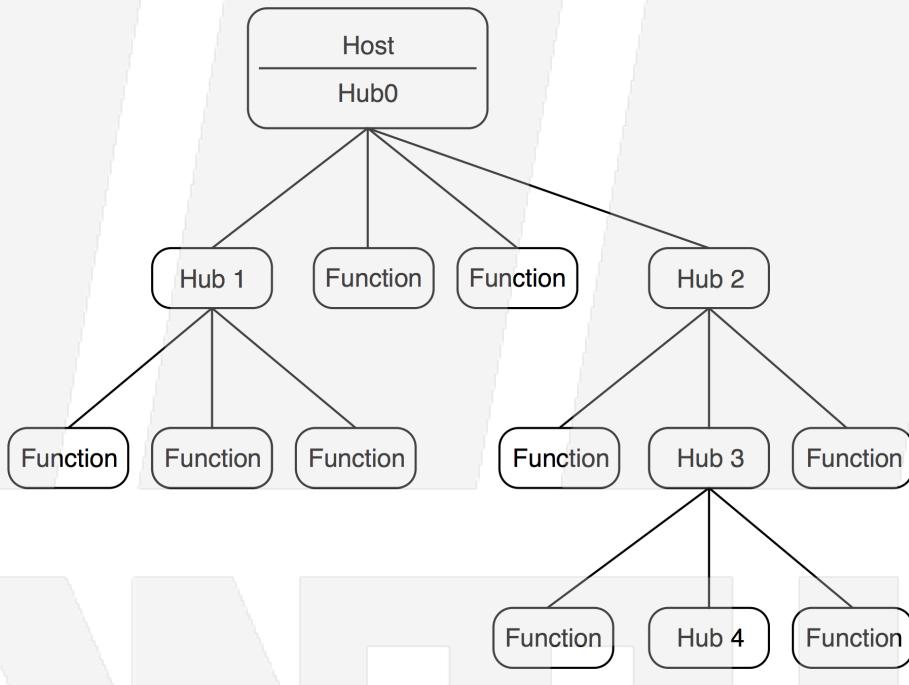


Abbildung 10.21: Struktur einer USB Hierarchie

Adressierung der Busteilnehmer:

Eine USB-Adresse besteht aus 11 Bits. Die ersten 7 Bits sind die Geräteadresse (Function-address); die weiteren 4 Bits dienen zur Adressierung der logischen Endpunkte der Function. Jedes Gerät besitzt nach Einschalten bzw. Anschließen die Adresse 0. Eindeutige Nummern werden danach im so genannten Enumeration Process vom Host vergeben.

Busarbitrierung:

Wie bereits in Abschnitt 10.2.2 erwähnt wurde, kommt beim USB das Polling-Verfahren zur Anwendung. Das heißt, dass der Host in regelmäßigen Abständen die Endgeräte nach Zugriffswünschen fragt und diese nach Prioritäten gewährt. Weiterhin kann der Host Teilnehmern feste Zeitkanäle pro Rahmen gewähren.

Kommunikation:

Die Kommunikation zwischen Host und Endgeräten geschieht beim USB durch zehn verschiedene Typen von Paketen. Der Typ wird durch eine stark redundante, 8-bit-weite Paketkennung ausgedrückt. Die verschiedenen Pakettypen werden im Folgenden kurz beschrieben:

- *Start-of-Frame-Paket (SOF)*: Kennzeichnung eines neuen Zeitrahmens von 1 ms. 11-bit Rahmennummer und 5-bit Prüfzeichen.
- *Token-Paket*: Übertragung der Adresse des Kommunikationspartners durch den Host sowie Art der Kommunikation (OUT, IN, SETUP) in den unteren 4 Bits der Adresse. Ebenfalls 5-bit Prüfzeichen.
- *Datenpaket*: Abhängig von der Übertragungsart können bis zu 1023 Datenbytes übertragen werden. Zusätzlich sind 16-bit Prüfzeichen enthalten. Die richtige Reihenfolge mehrerer Datenpakete wird durch abwechselnde Kennzeichnung sichergestellt.
- *Handshake-Pakete*: Es gibt drei Arten, die den Empfang eines Paketes quittieren:
ACK: Fehlerfreier Empfang. Bei umfangreichen Daten nur nach dem letzten Paket.
NAK: Momentan kein Senden und Empfangen eines Datenpakets möglich.
STALL: Endpunkt außer Betrieb. Intervention des Hosts erforderlich.
Zusätzlich kann ein SETUP-Paket geschickt werden, das eine Function initialisiert und sie zum Zurücksenden von Statusinformationen veranlasst.

Je nach Anforderungen der Endgeräte wird eine von fünf Übertragungsarten des USB verwendet. Diese teilen sich die Gesamtzeit eines Rahmens. Folgende Übertragungsarten werden unterstützt:

- *Isochrone Übertragung*: Garantiert eine konstante Datenrate durch Reservierung eines gleichgroßen Abschnitts in jedem Rahmen. Bei dieser Art der Übertragung findet keinerlei Fehlererkennung oder -korrektur statt. Sie wird hauptsächlich zur Übertragung von Audio-, Video und Multimediadaten verwendet. Die maximale Bandbreite unter Abzug von Steuerinformationen beträgt 8,2 Mbit/s.
- *Bulk-Übertragung*: Große Datenmengen (z. B. bei Drucker, Scanner, Modem) werden nach diesem Verfahren übertragen. Bei Erkennung eines Fehlers mittels Prüfzeichen werden die Daten erneut gesendet. Der Bulk-Übertragung stehen nur die von anderen Typen nicht belegten Bereiche eines Rahmens zur Verfügung, maximal 9,7 Mbit/s.
- *Interrupt*: Nicht-periodisch auftretende Daten (z. B. Maus, Tastatur) mit geringen Datenmengen werden bei Bedarf übertragen. Auch hier findet Fehlererkennung und ggf. Wiederholung statt. Diesem Übertragungstypen ist eine feste Bedienzeit aber kein konkretes Zeitverhalten zugesichert. Bei 8, 16, 32 oder 64 Bytes pro Paket kann eine maximale Bandbreite von 9,7 Mbit/s erreicht werden.
- *Steuerdaten*: Für die Übertragung von Daten zur Konfiguration und Überwachung von Functions und Hubs sind mindestens 10 % eines Rahmens reserviert. Bei 8, 16, 32 oder 64 Byte Daten pro Paket ist eine maximale Bandbreite von 6,6 Mbit/s möglich.
- *Low Speed*: Maximal ein Paket pro Rahmen darf mit langsamer Geschwindigkeit (1,5 Mbit/s) übertragen werden.

Hot Plugging:

Beim Anschließen eines Endgerätes oder Hubs während des Betriebs gibt der Hub, an den das neue Gerät angeschlossen wird, eine Statusmeldung an den Host ab. Dieser fordert darauf vom Hub die Portnummer des neuen Gerätes und aktiviert den entsprechenden Port. Über den Steuerungskanal des neuen Gerätes spricht der Host das neue Gerät an, erhält Informationen über dieses als Antwort und vergibt eine eindeutige USB-Adresse. Ist eine Function hinzugekommen, wird diese im Host eingetragen und der entsprechende Treiber geladen. Bei Anschluss eines Hubs wird die oben beschriebene Prozedur für alle an den Hub angeschlossenen Geräte wiederholt. Nach Bestimmung der verlangten Datenraten und Latenzen der neu angeschlossenen Geräte durch den Host sind diese betriebsbereit. Können die geforderten Bedingungen nicht erfüllt werden, ist das Hot Plugging gescheitert. Das Entfernen von Geräten geschieht analog: Das Übergeordnete Hub teilt die Änderung dem Host mit, der dann die betroffenen Geräte abmeldet.

FireWire (IEEE-1394-Bus)

Der IEEE-1394 ist eine Weiterentwicklung des von Apple entwickelten *FireWires*. Die beiden Begriffe werden jedoch in der Regel synonym verwendet. Das Bussystem wird sowohl in industriellen Systemen als auch als serieller Hochgeschwindigkeitsbus bei Arbeitsplatzrechnern und Unterhaltungsgeräten (z. B. Sonys i.Link) verwendet. Hier wird nur auf die zweite Variante eingegangen. Der FireWire hat viele Parallelen zum USB: Er unterstützt *asynchrone Übertragung* und Übertragung mit fester Bandbreite (*isochrone Übertragung*), Hot Plugging, und kann als Spannungsversorgung für Geräte niedriger Leistungsaufnahme dienen. Bei der verwendeten Hardware (Kabel, Stecker usw.) hat man sich ebenfalls auf eine einheitliche, kostengünstige Realisierung geeinigt. Die Datenraten beim FireWire sind jedoch mit 100, 200 oder 400 Mbit/s wesentlich höher als bei USB. Derzeit wird sogar an Geräten mit Datenraten bis zu 4,8 Gbit/s entwickelt. Durch die hohe Übertragungsgeschwindigkeit und die Hot Plugging Fähigkeit eignet sich dieser Bustyp auch für *Device Bays* (Einschubsschächte für Peripheriegeräte im Rechner).

Topologie:

Der FireWire-Bus hat eine Baum-Topologie (vgl. z. B. Abbildung 10.22). Als Knoten sind alle Geräte mit FireWire Anschluss zugelassen, also Rechner, Koppeleinheiten und andere Endgeräte. Pro Bus sind 63 Knoten erlaubt; über Brücken können bis zu 1023 Busse gekoppelt werden. Daraus ergibt sich eine Anzahl von maximal 64449 Geräten an einem IEEE-1394-Bus. Wie in Abbildung 10.22 angedeutet können auf den verschiedenen Verbindungsleitungen unterschiedliche Übertragungsraten verwendet werden. Anders als bei USB ist das Vorhandensein eines Rechners nicht notwendig. Das Bussystem bestimmt automatisch einen der Teilnehmer zum Root-Knoten. Somit ist auch die Kommunikation direkt zwischen zwei Endgeräten möglich.

Jeder Teilnehmer kann maximal 27 Ports für weitere Teilnehmer zur Verfügung stellen. Zwischen zwei miteinander kommunizierenden Knoten dürfen maximal 16 Kabelabschnitte liegen. Die Länge der Kabelabschnitte ist bei Low-cost-Kabeln auf 4,5 m, bei POF-Kabeln (Plastic Optical Fibre) auf 50–100m begrenzt. Daraus ergeben sich als maximale Entfernung zweier Knoten 72 m bzw. 1600 m. Die Daten werden in Paketen übertragen, die denen des USB entsprechen.

Adressierung der Busteilnehmer:

Von den insgesamt 64 Bits einer FireWire-Adresse werden 10 Bits zur Adressierung des jeweiligen Teilbusses und 6 Bits zur Selektion der Komponente verwendet. Die übrigen 48 Bits

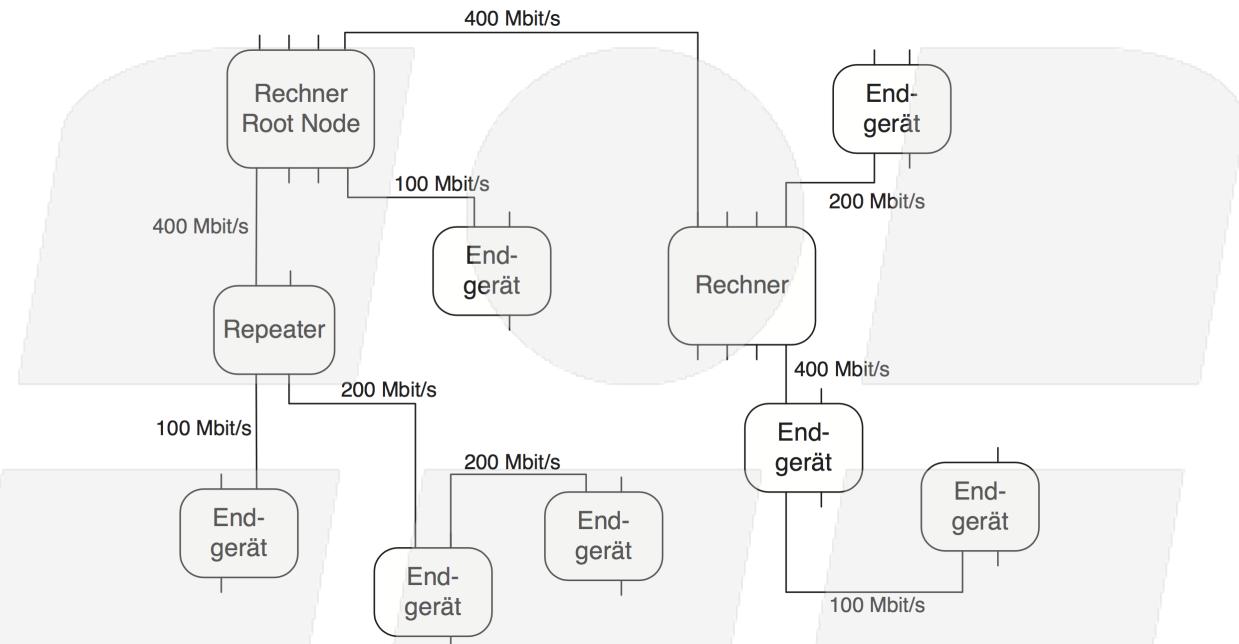


Abbildung 10.22: Beispiel zur Struktur eines FireWire-Systems

ermöglichen die Adressierung von 256 TByte pro Knoten.

Busarbitrierung:

Wie bereits in Abschnitt 10.2.1 erläutert, erfolgt die Arbitrierung per Request von den Teilnehmern und Zuweisung durch den Root-Knoten.

Kommunikation:

Der asynchrone Übertragungsmodus dient hauptsächlich der sicheren Übertragung von Daten. Wenn einem Knoten Zugriff auf den Bus erteilt wird, sendet dieser die Daten in einzelnen Paketen zum Empfänger. Dieser muss den korrekten Empfang bestätigen. Bei Fehlern wird das Paket neu verschickt. Im Gegensatz dazu hält die isochrone Übertragung eine bestimmte Bandbreite ein. Dadurch können bei fehlerhafter Übertragung jedoch die Daten nicht neu geschickt werden. Diese Übertragungsart wird besonders für die Echtzeitübertragung von Audio-, Video- und Multimediadaten verwendet.

10.2.5 Schnittstellen für Punkt-zu-Punkt Verbindungen

Als *Schnittstelle* bezeichnet man die Übergangsstelle zwischen den Geräten eines Systems, an der Daten und Signale ausgetauscht werden. Schnittstellen sind z.B. für den physikalischen Zugriff eines Geräts auf ein Bussystem erforderlich. Wie schon bei den Bussen muss man auch hier grundsätzlich zwischen parallelen und seriellen Schnittstellen unterscheiden.

Bei den parallelen Schnittstellen werden die Daten in ein oder mehrere Bytes breiten Worten parallel übertragen, wogegen die seriellen Schnittstellen für eine bitweise serielle Übertragung der Daten konzipiert sind. Parallele Schnittstellen ermöglichen hohe Übertragungsraten durch kurze, vieladrigie Kabelverbindungen, während serielle Schnittstellen lange (kostengünstige) Kabelverbindungen (z.B. Telefonnetze) bei niedrigeren Übertragungsraten und wenigen Leitungsadern erlauben.

Damit ein Computer mit der Außenwelt kommunizieren kann, muss er Daten mit den verschiedensten Peripheriegeräten austauschen können. Um Geräte unterschiedlicher Herstellung miteinander verbinden zu können, müssen die Schnittstellen der Geräte genormt oder zumindest ähnlich sein. Deshalb gibt es für die Übertragung der Daten verschiedene Schnittstellennormen bzw. Standards (Herstellerempfehlungen), die nicht nur die physikalische Realisation der Schnittstelle (Pinbelegung), sondern z.B. auch die Protokolle für den synchronen oder asynchronen Datenverkehr in beide Richtungen festlegen. Zwei dieser so genannten Standardschnittstellen sollen im Folgenden vorgestellt werden.

V.24 (RS232)-Schnittstelle

Die V.24-Schnittstelle ist eine asynchrone, serielle Schnittstelle. Die einzelnen Zeichen werden jeweils als Folge von Einzelbits übertragen. Jede Folge wird von einem Startbit mit 'L'-Pegel angeführt und von einem oder zwei Stopbits mit 'H'-Pegel abgeschlossen. Als letztes Datenbit kann optional ein Paritätsbit (siehe Kapitel 3) zur Fehlererkennung eingefügt werden (Abb. 10.23). Der 'H'-Pegel entspricht einer Spannung von $+3V$ - $+15V$, der 'L'-Pegel einer Spannung von $-3V$ bis $-15V$. Sinn und Zweck dieser Norm, die im Wesentlichen dem amerikanischen RS232-Standard entspricht, ist es, eine einheitliche Basis für die Datenfernübertragung zu schaffen. Sie beruht auf einer von der CCITT im Jahre 1968 verabschiedeten Empfehlung zur Gestaltung von Schnittstellen für die Datenübertragung zwischen Datenendeinrichtungen (DEE, z.B. Computer, Drucker) und Datenübertragungseinrichtungen (DÜE, z.B. MODEM). Die Empfehlung sieht ca. 50 Schnittstellenleitungen vor, die alle denkbaren Anwendungsfälle abdecken sollen. So existieren neben der Masseleitung und den Datenleitungen noch eine ganze Reihe von Leitungen, die den Verkehr zwischen DEE und DÜE steuern.

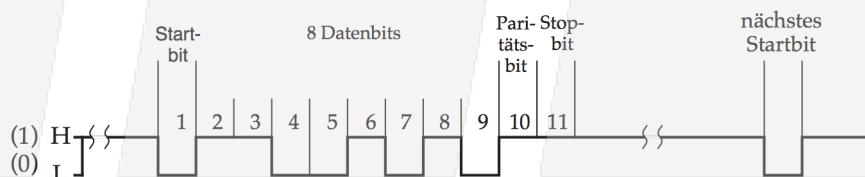


Abbildung 10.23: Serielle Übertragung eines ASCII-Zeichens mit ungeradem Paritätsbit und einem Stopbit

Die Gesamtzahl der von der Norm vorgesehenen Schnittstellenbelegungen wird jedoch nur von den wenigsten Anwendern genutzt. Für konkrete Anwendungsfälle können geeignete Leitungen ausgewählt werden. So reichen 3 Leitungen (TD, RD, GND) völlig aus, um z. B. den Verkehr zwischen Computer und Terminal aufrechtzuerhalten. Tabelle 10.4 zeigt eine Auswahl nach der Vorschrift DIN 66020. Sie sieht die Nutzung von 9 Pins des genormten 25-poligen V.24 Steckers (Abb. 10.24) vor.

Die Übertragungsgeschwindigkeit (Baudrate) beträgt bis zu 19 200 Baud (Bit/s), die Reichweite bis zu 15 m. Sie ist innerhalb eines vorgegebenen Rasters frei wählbar. Damit es nicht zu Fehlinterpretationen von Daten kommt, müssen beide Geräte, sofern sie eigene Taktgeber besitzen, auf eine einheitliche Baudrate eingestellt werden.

Da der Einsatz von DÜE für Entferungen von wenigen Metern unsinnig ist, können DEE'en auch direkt miteinander verbunden werden (z.B. Computer und Drucker). Da bei V.24-Schnittstellen allgemein gilt, dass die Ausgangsleitungen der DEE immer mit den zugehörigen

Tabelle 10.4: Pinbelegung der RS232-Schnittstelle

Pin	Signal	DIN Bez.	Funktion
2	Data transmit - TD	D1	Senden von Daten an die DÜE
3	Data receive - RD	D2	Senden von Daten an die DEE
4	Request to send - RTS	S2	Aufforderung an die DÜE zum Senden
5	Clear to send - CTS	M2	DÜE ist sendebereit
6	Data set ready - DSR	M1	DÜE ist betriebsbereit
7	Signal Ground - GND	E2	Betriebserde
8	Carrier detect - DCD	M5	Empfangssignalpegel der DÜE innerhalb des Toleranzbereiches
17	Receive clock - RC	T4	Empfangsschritt-Takt
20	Data terminal ready DTR	S1.2	Betriebsbereitschaft der DEE
22	Ring Indicator - RI	M3	Ankommender Ruf
24	Transmit clock - TC	T1	Sendeschritt-Takt an die DÜE

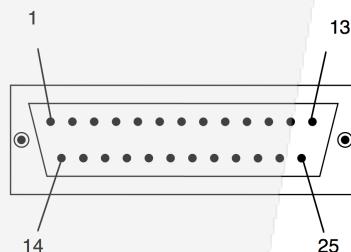


Abbildung 10.24: Steckerbelegung der 25-poligen D-Subminiatur Steckergarnitur

Eingangsleitungen der DÜE zu verbinden sind, gelten für diese Fälle einige Einschränkungen. So müssen beispielsweise einige Leitungen über Kreuz miteinander verbunden werden (Null-Modem). Abbildung 10.25 zeigt eine solche Anordnung.

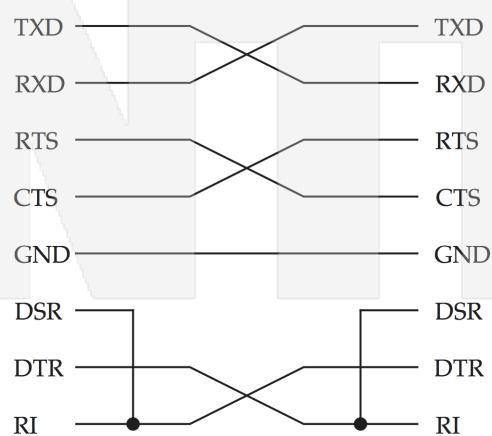


Abbildung 10.25: Verbindung von zwei DEE' en mit einem Null-Modem

Centronics-Schnittstelle

Ein weitverbreiteter Standard für parallele Datenübertragung ist die *Centronics-Schnittstelle*. Sie wurde erstmalig von der Firma Centronics zur Druckeransteuerung eingesetzt. Bei ihr handelt es sich im Gegensatz zur V.24-Schnittstelle nicht um eine amtliche Normierung. Dennoch ist sie im PC-Bereich als Schnittstelle zur Druckeransteuerung weit verbreitet. Da diese Schnittstelle mit TTL (Transistor-Transistor-Logik)-Pegeln arbeitet, muss die Leitungslänge unter 2 m bleiben und jede Signalleitung mit eigener Masseleitung verdrillt werden (twisted pairs). Im Gegensatz zur seriellen Schnittstelle werden alle 8 Datenbits gleichzeitig, also parallel, über 8 getrennte Leitungen übertragen.

Die Centronics-Schnittstelle ist nur für die Datenübertragung in eine Richtung (unidirektional), z.B. vom Computer zum Drucker, ausgelegt. In der Regel wird druckerseitig ein 36-poliger Amphenol-Stecker verwendet (AMP 57-30360, Abb. 10.26), während rechnerseitig meist eine 25-polige SUB-D-Buchse eingesetzt wird.

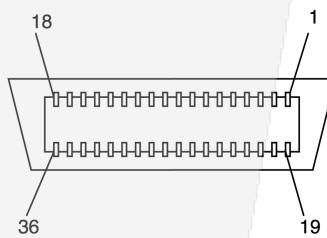


Abbildung 10.26: Amphenol-Stecker für die Centronics-Schnittstelle

Die Steckerbelegung der wichtigsten Leitungen wird im Folgenden erläutert und in Tabelle 10.5 noch einmal zusammengefasst:

Leitung 1 (STROBE) führt normalerweise den logischen Pegel 'H', wechselt jedoch auf 'L', um dem Peripheriegerät zu signalisieren, dass ein Datenbyte bereitsteht.

Leitungen 2 - 9 (DATA 0 - DATA 7) dienen der byteweisen Datenübertragung.

Leitung 10 (ACKNLG - acknowledge=bestätigen) liegt im Ruhezustand auf 'H' und wird vom Peripheriegerät auf 'L' gesetzt, um dem Sender den Empfang übertragener Daten zu signalisieren (Quittierung).

Leitung 11 (BUSY - busy=beschäftigt, besetzt) dient der Peripherie dazu, dem Computer Empfangsbereitschaft zu signalisieren. 'L' bedeutet hierbei: Gerät ist empfangsbereit, 'H' bedeutet: Gerät kann momentan keine weiteren Daten verarbeiten.

Leitung 16 (GND - ground=Erde, Masse) ist das Bezugspotenzial aller Signale. Sie liegt immer auf 'L'.

Leitung 31 (INIT - Initialisierung) liegt normalerweise auf 'H', ein Impuls mit dem Pegel 'L' initialisiert das Peripheriegerät (entspricht dem Aus- und Wiedereinschalten des Gerätes).

Leitung 32 (ERROR - Fehler) liegt normalerweise auf 'H', wechselt auf 'L', falls am Peripheriegerät ein Fehler auftritt (z.B. Papiermangel im Drucker).

Die Datenübertragung läuft nach den Regeln des Quittungsbetriebes (Handshake-Betrieb) folgendermaßen ab (Abb. 10.27): Kurz nachdem ein Zeichen von 8 Bit Breite auf den Datenleitungen anliegt, gibt das datensendende Gerät (z.B. Computer) ein Übernahmesignal (STROBE auf 'L') von kurzer Dauer ab. Der Empfänger (Drucker) bestätigt dies, indem er ACKNL kurz auf 'L' setzt. Während der darauffolgenden Verarbeitung der Daten wird BUSY vom Empfänger solange auf 'H' gesetzt, bis er wieder bereit ist, Daten zu empfangen.

Tabelle 10.5: Pinbelegung der Centronics-Schnittstelle

Pin	Bezeichnung
1	Strobe
2 – 9	Data 0 – 7
10	Acknowledge
11	Busy
12	GND
13	Select
14	Auto-linefeed
15	unbenutzt
16 – 17	GND
18	+5V
19 – 30	GND
31	Init
32	Error
33 - 36	unbenutzt

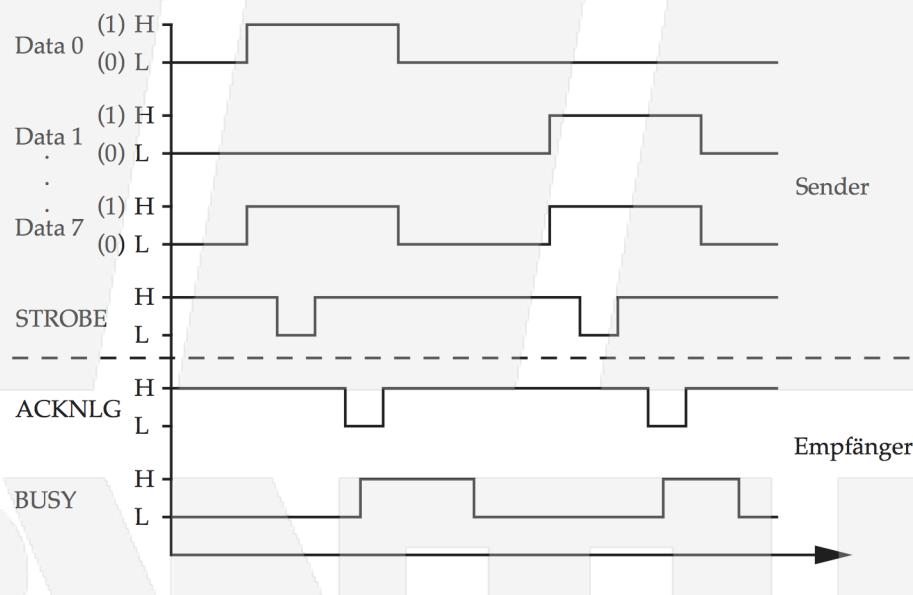


Abbildung 10.27: Schema des Signalverlaufs bei der Übertragung von zwei Zeichen über eine Centronics-Schnittstelle

10.2.6 Drahtlose Kommunikation

Neben Schnittstellen, die mit Hilfe einer physikalischen Verbindung realisiert sind, existieren noch diverse drahtlose Schnittstellen und Kommunikationsprotokolle, die z. B. über Funkwellen oder Infrarotlicht miteinander kommunizieren. Zwei verbreitete Standards, nämlich IrDA (Infrared Data Association) und Bluetooth, werden im Folgenden beschrieben.

IrDA

IrDA (Infrared Data Association) ist ein Schnittstellenstandard zur Kommunikation per Infrarotlicht. Es wird eine Punkt-zu-Punkt-Verbindung zwischen zwei maximal 1m entfernten Geräten realisiert. Die Kommunikationsgeschwindigkeit wird zu Beginn jeder Übertragung von den beiden Kommunikationspartnern ausgehandelt und beträgt bis zu 4 MBit/s, mindestens jedoch 9,6 KBit/s. Die Position der beiden Partner wird über die Sende- bzw. Empfangskegel definiert. Beide besitzen einen Öffnungswinkel von 30°. Eine Kommunikation ist möglich, wenn sich zwischen Sender und Empfänger eine freie Strecke befindet und diese Strecke sowohl im Sende- als auch im Empfangskegel liegt.

Bluetooth

Der Name *Bluetooth* („Blauzahn“) geht auf den Wikinger-König Harald „Blåtand“ von Dänemark (940–986) zurück, der Dänemark und Teile Norwegens unter seiner Krone vereinigte. Bluetooth ist ein einheitlicher Standard für den Aufbau lokaler Funknetze. Ein einfaches Bluetooth-Netz (*Piconet*) besteht aus einer Master Unit und bis zu sieben aktiven Slave Units. Zusätzlich können noch bis zu 256 so genannte Parked Devices an ein Piconet angebunden sein. Diese sind zwar mit dem Master synchronisiert und können schnell aktiviert werden, sind aber an keiner Übertragung beteiligt. Ist eine Einheit an mehreren Piconets beteiligt, so bilden diese Netze ein *Scatternet*.

Es gibt drei verschiedene Konfigurationen eines Bluetooth-Übertragungskanals; als Sprachmedium mit drei synchronen Kanälen zu je 64 KBit/s in beiden Richtungen, als symmetrischer Datenkanal mit je 433.9 KBit/s in beide Richtungen, und als asymmetrischer Datenkanal mit 723.2 KBit/s in eine Richtung und 57.6 KBit/s in die andere Richtung. In das Bluetooth-Übertragungsprotokoll ist ein Verschlüsselungsverfahren integriert. Diese Eigenschaft besitzt sonst keines der verbreiteten Schnittstellenprotokolle.

Der Bluetooth-Standard arbeitet mit Funkfrequenzen im 2,4 GHz-Band und definiert drei mögliche Leistungsstufen für die Übertragung, nämlich 1 mW Sendeleistung (Klasse 3), 2,5 mW (Klasse 2) und 100 mW (Klasse 1). Damit lassen sich Entferungen bis zu 100 m überbrücken. Die typische Distanz liegt jedoch zwischen 1 m und 10 m.

10.3 Ein-/Ausgabetechniken

10.3.1 Ein-/Ausgabe über den CPU-Bus

Die *Ein-/Ausgabe* gehört zu den wichtigsten und häufigsten Operationen von informationsverarbeitenden Systemen, weshalb die Wahl eines geeigneten Ein-/Ausgabeverfahrens im Zeitalter der Vernetzung und Informationssysteme zunehmend an Bedeutung gewinnt.

Ein-/Ausgabe-Operationen lassen sich in zwei Klassen einteilen:

- programmgesteuerte Ein-/Ausgabe
- unterbrechungsgesteuerte Ein-/Ausgabe

In beiden Klassen wird zwischen der direkten und der gepufferten Ein-/Ausgabe unterschieden.

Direkte Ein-/Ausgabe

Bei der direkten Ein-/Ausgabe wird die Ansteuerung des Peripheriegerätes in allen Funktionen durch direkte Adressierung von Daten- und Steuerleitungen durchgeführt. Eingabedaten müssen im Moment des Anliegens am CPU-Bus gelesen (und interpretiert) werden, Ausgabedaten werden im Moment des Anliegens am CPU-Bus geschrieben.

Die direkte Ein-/Ausgabe eignet sich nur für den langsamen Datenverkehr, solange die CPU zusätzlich mit anderen Prozessen beschäftigt ist. Die direkte programmgesteuerte Ausgabe erfolgt dann nur selten (zyklisch zwischen den Prozessen des Hauptprogramms), wobei durch Abfrage der entsprechenden Steuerleitung geprüft werden muss, ob der (direkte) Ausgabekanal zur Verfügung steht. Ist das nicht der Fall, muss entweder auf die Freigabe zum Senden gewartet werden (was möglicherweise unvorhersehbar lange dauert) oder bis zum nächsten Durchlauf der Ausgaberoutine gewartet werden.

Die direkte programmgesteuerte Eingabe ist (bei asynchronen Übertragungen) wenig sinnvoll, da das zyklische Abfragen (*Polling*) der Steuerleitungen („liegt ein Datenwort an?“) und Auslesen des Datenbusses sehr schnell geschehen muss, damit keine Daten verloren gehen. Andernfalls muss das Programm solange angehalten werden, bis das erwartete Ereignis eintritt, was unter Umständen gar nicht passiert. Während dieser Wartephase kann der Rechner keine anderen Aufgaben übernehmen.

Bessere Möglichkeiten bietet die unterbrechungsgesteuerte direkte Ein-/Ausgabe. Hier wird durch die Ereignisse „Sendekanal ist frei“ bzw. „Ereignis liegt auf Empfangskanal an“ über den Steuerbus eine Unterbrechungsanforderung (*Interrupt Request*) an die CPU gestellt. Diese bricht, sofern der Interrupt zugelassen ist, die Bearbeitung des laufenden Prozesses ab, um in einer Unterbrechungsroutine die Ein- bzw. Ausgabe durchzuführen. Danach wird mit der Abarbeitung des Hauptprogramms fortgefahrene.

Gepufferte Ein-/Ausgabe

Die gepufferte Ein-/Ausgabe ermöglicht hohe Übertragungsgeschwindigkeiten. Häufig wird die Ein- bzw. Ausgabe von einem eigenen Micro-Controller am CPU-Bus übernommen.

Bei der Eingabe wird das empfangene Datenwort in einem Puffer des Micro-Controllers gespeichert und solange bereitgehalten, bis es (entweder programm- oder unterbrechungsgesteuert) von der Eingaberoutine abgeholt wird. Dazu müssen vom Micro-Controller oder von der Ein-/Ausgaberoutine Vorkehrungen getroffen werden, dass ein Datenwort nicht durch das nachfolgende überschrieben wird, solange es nicht aus dem Puffer abgeholt wurde.

Bei der Ausgabe wird entsprechend das Datenwort zunächst in einen Puffer (Register) des Micro-Controllers geschrieben, von wo es (bei Freigabe des Übertragungskanals) übernommen und übertragen wird.

Abbildung 10.28 zeigt das Prinzipschaltbild für die gepufferte, programmgesteuerte Ausgabe über das sog. *Handshake-Verfahren*. Das Programm im Rechner fragt zyklisch den Ausgang P des Flip-Flops solange ab, bis dieser von der Peripherie auf LOW (P=0) gesetzt worden ist (polling). Damit zeigt die Peripherie an (*Handshake*), dass sie den Inhalt des Ausgaberegisters empfangen hat, so dass der Inhalt durch ein neues Datenwort überschrieben werden kann. Dann wird über einen Adressdecodierer das Ausgaberegister adressiert und über den Datenbus mit dem neuen Datenwort geladen. Der Ausgang des Flip-Flops P wird auf HIGH (P=1) gesetzt. Dadurch wird der Peripherie mitgeteilt (*Handshake*), dass ein neues Datenwort abgeholt werden

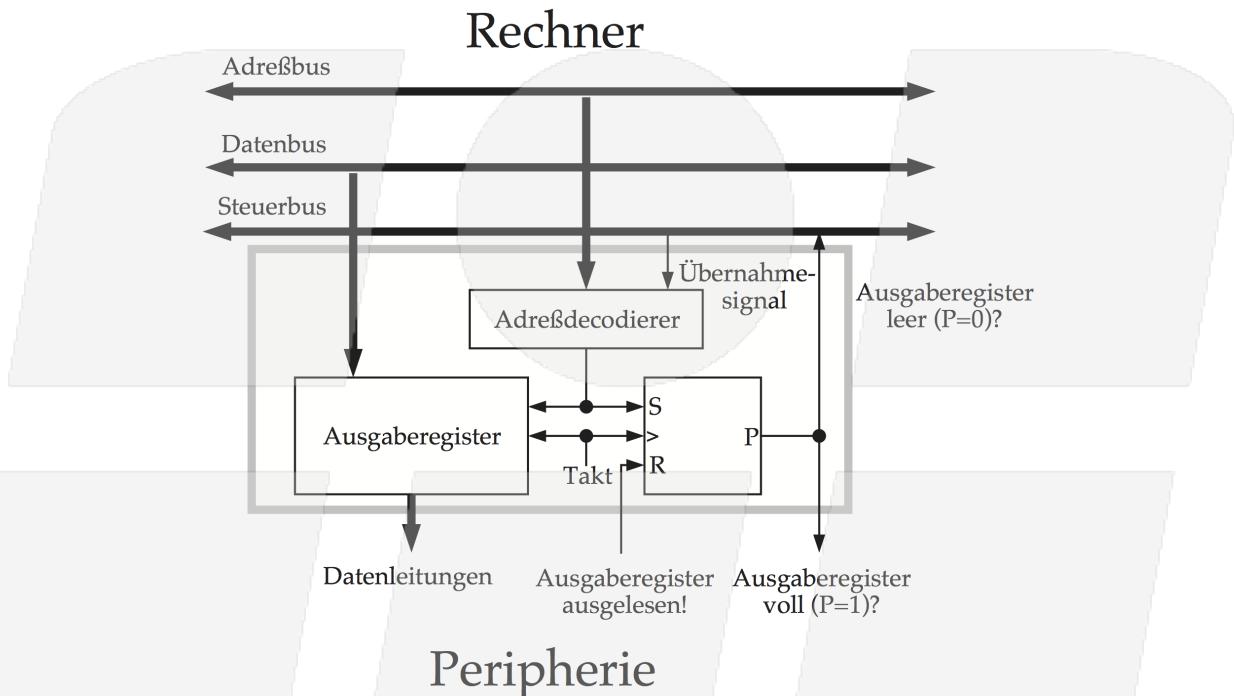


Abbildung 10.28: Prinzipschaltbild für die gepufferte, programmgesteuerte Ausgabe

kann. Bis das geschieht, braucht die CPU nicht angehalten zu werden, da das Datenwort im Register gespeichert (gepuffert) ist.

Abbildung 10.29 zeigt das entsprechende Prinzipschaltbild für die gepufferte, interruptgesteuerte Ausgabe. Der Controller ist um ein zusätzliches (Interrupt-) Register erweitert worden. Sobald die Peripherie ein neues Datenwort empfangen kann, setzt sie den Ausgang P des Flip-Flops auf LOW und löst damit einen Interrupt im Prozessor des Rechners aus. Der Prozessor unterbricht daraufhin den normalen Programmablauf und lädt das Interruptregister mit der Startadresse des zugehörigen Unterbrechungsprogramms, welches dann die eigentliche Ausgabe vornimmt (Laden des Ausgaberegisters und Setzen des Flip-Flops).

Häufig wird von dem Betriebssystem eines Rechners eine weitere Pufferung auf Software-Basis der auf diese Weise empfangenen bzw. zu sendenden Daten durchgeführt. Diese Routinen (für Schreib- und Leseoperationen), deren Puffergröße durch Parameterübergabe eingestellt werden kann, übernehmen dann die Kommunikation mit der Hardware (unterbrechungsgesteuert, durch Micro-Controller gepuffert) und können von anderen Programmen aufgerufen werden.

10.3.2 Ein-/Ausgabe über Direct Memory Access (DMA)

Mit Hilfe des *DMA* (Abb. 10.30) kann ein (blockweiser) Datentransfer zwischen Hauptspeicher und Peripherie durchgeführt werden. Der DMA-Controller erhält die Startadresse und Länge eines zu übertragenden Blocks entweder programm- oder interruptgesteuert durch die CPU.

Während des DMA-Transfers ist die CPU im einfachsten Fall völlig stillgelegt. Beim so genannten *Cycle Stealing* „stiehlt“ der DMA-Controller einzelne Arbeitstakte der CPU, ohne diese völlig abzuschalten. Der DMA-Zugriff erfolgt immer dann, wenn die CPU keinen Speicherzugriff ausführt. Beim DMA-Transfer muss die Priorität des Buszugriffs durch CPU und DMA-Controller verwaltet werden. Generell hat der DMA höhere Priorität und die CPU muss warten,

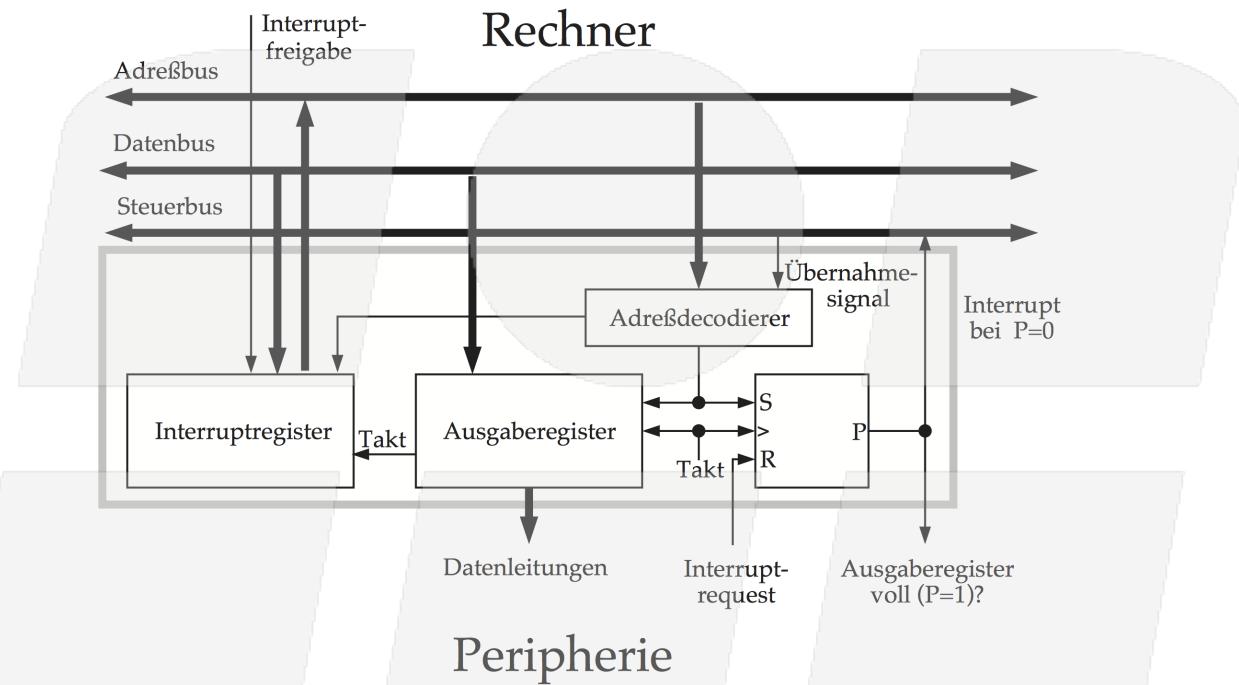


Abbildung 10.29: Prinzipschaltbild für die gepufferte, interruptgesteuerte Ausgabe

bis der Bus freigegeben ist.

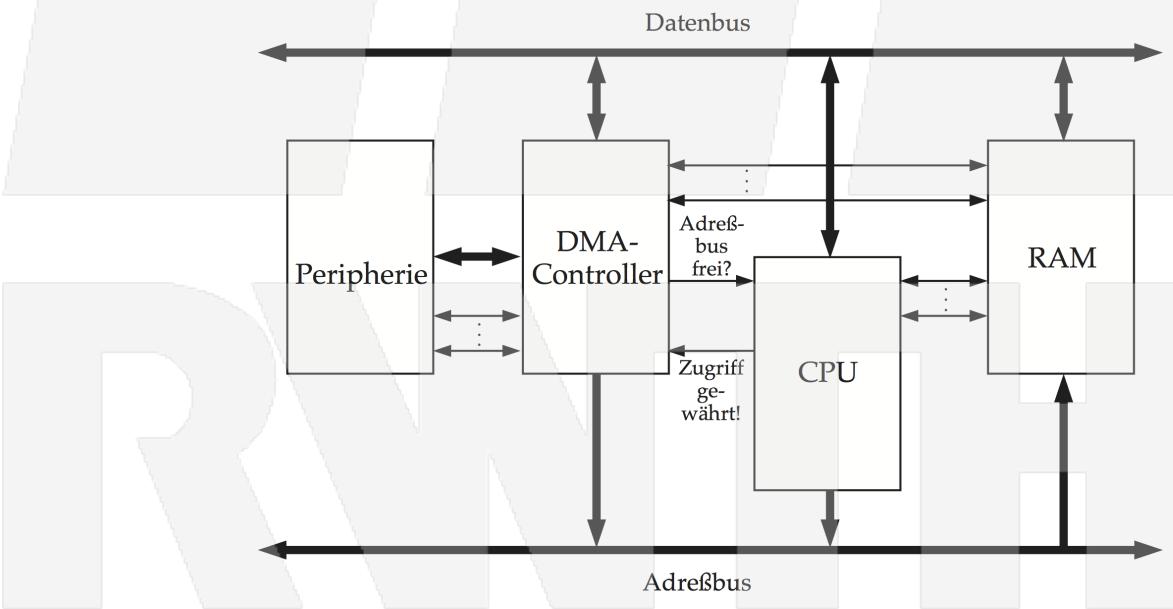


Abbildung 10.30: Prinzipschaltbild für den Direct Memory Access (DMA)

Beim PCI-Bus sowie bei Crossbar-Bussystemen ist es allerdings möglich, dass die CPU weiterarbeitet während ein DMA-Transfer durchgeführt wird. Lediglich der Zugriff auf die beiden Teilnehmer am DMA Transfer ist für die CPU gesperrt.

10.4 Ein-/Ausgabe von Analogdaten

Oft müssen physikalische Größen eines Prozesses wie Temperatur, Druck oder elektrischer Widerstand von einem Rechner gemessen bzw. gesteuert werden. Die Messdaten liegen in der Regel als Strom- oder Spannungswerte vor, aus denen die gewünschten physikalischen Größen berechnet werden können. Meist liegt als Analogwert eine elektrische Spannung im Bereich von 10^{-3} V bis 10^3 V vor, die von einem Analog-/Digitalumsetzer in einen digitalen Wert umgewandelt werden muss. Soll ein Analogwert ausgegeben werden, so wird die digitale Information des Rechners mit Hilfe eines Digital/Analog-Wandlers in eine Analogspannung umgesetzt.

10.4.1 Analogeingabe

Sägezahnspannung

Ein Generator erzeugt einen linearen, sägezahnartigen Spannungsanstieg. Die Zeit bis zum Erreichen des Pegels der Messspannung wird mit einem Zähler gemessen. Der Zählerstand ist proportional zur Messspannung.

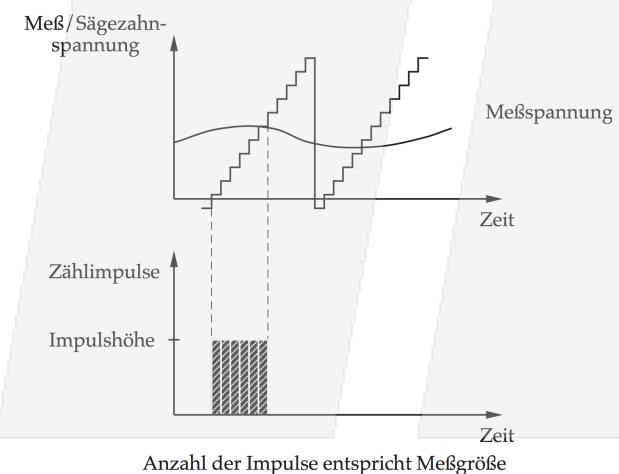


Abbildung 10.31: Zeitdiagramm der Sägezahnumsetzung

Stufenumsetzung/Wägeverfahren

Schnell (bis zu 100000 Wandlungen/sec), da die Wandlungszeit nur proportional zum Logarithmus der Anzahl der Vergleichsgrößen steigt. Die Messspannung wird mit einer Reihe binär abgestufter Messpegel verglichen und dabei schrittweise eingegrenzt. Das Ergebnis einzelner Vergleiche wird in einem Register gespeichert. Abbildung 10.32 zeigt den Näherungsvorgang für eine achtstellige digitale Ausgabe.

10.4.2 Analogausgabe

Die Bitwerte eines Ausgaberegisters werden zum Zu- und Abschalten binär abgestufter Widerstände benutzt (Abbildung 10.33). In Verbindung mit einer eingeprägten Stromquelle und

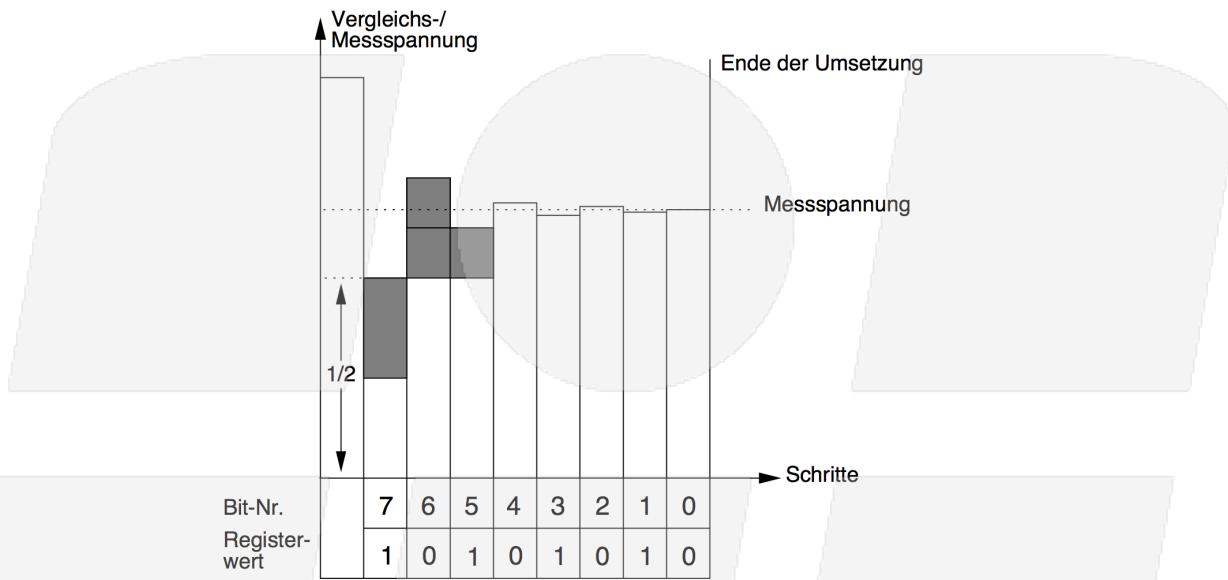


Abbildung 10.32: Zeitdiagramm zur Stufenumsetzung

einem Operationsverstärker ergibt sich die dem Digitalwert entsprechende Analogspannung zu

$$U_a = R \cdot U_{REF} \sum_{k=0}^{N-1} 2^k G_0 S_k ; S_k = \begin{cases} 0, & \text{Bit ungesetzt} \\ 1, & \text{Bit gesetzt} \end{cases}$$

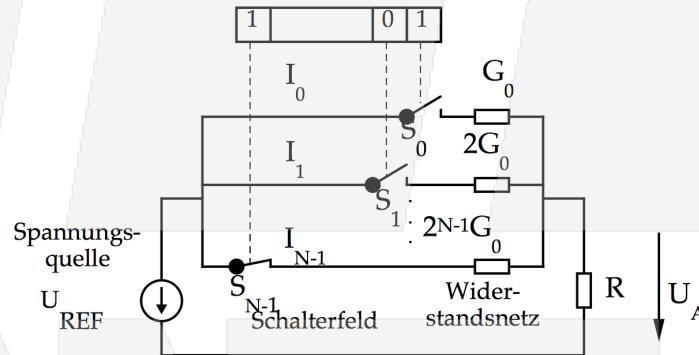


Abbildung 10.33: D/A-Wandler mit Stromsummation im Summierv verstärker

Kapitel 11

Speichertechnik

11.1 Speichermerkmale

Der *Speicher* beschreibt die Funktionseinheit des Rechners, in dem die Daten und Programme aufbewahrt werden. In digitalen Speichern geschieht dies in Form von Bits, die durch Speicherelemente repräsentiert werden, welche einen von zwei erlaubten physikalischen Zuständen annehmen können. Um die Speicherelemente zu lokalisieren, ordnet man einer Gruppe von Speicherelementen eine *Adresse* zu. Diese kleinste adressierbare Einheit wird mit *Speicherzelle* bezeichnet. Im allgemeinen entspricht eine Speicherzelle einem Speicherwort. Durch die Angabe der Adresse einer Speicherzelle kann ihr Inhalt entweder gelesen oder geschrieben werden. Diesen Vorgang bezeichnet man mit *Zugriff*. Speicher können nach unterschiedlichen Merkmalen wie Zugriffszeit, Zugriffsart, Zykluszeit, Speicherkapazität oder Speicherprinzip unterschieden werden. Außerdem unterscheidet man zwischen statischem und dynamischem Speicher. Diese Merkmale sollen kurz erläutert werden:

Mit *Zugriffszeit* bezeichnet man die Dauer, die zum Lesen bzw. Schreiben einer Speicherzelle benötigt wird. Zugriffszeiten schwanken abhängig vom verwendeten Typ zwischen wenigen Nanosekunden und mehreren Sekunden.

Die *Zugriffsart* beschreibt die Methode, mit der auf den Speicher zugegriffen wird. Man unterscheidet zwischen:

- Speicher mit wahlfreiem Zugriff (*Random Access Memory - RAM*):
Auf jede Speicherzelle kann unabhängig von ihrer Position auf die gleiche Weise mit dem gleichen Aufwand zugegriffen werden.
- Festwertspeicher (*Read Only Memory - ROM*):
Auf die Speicherzellen kann ebenfalls direkt zugegriffen werden, allerdings nur, um ihren Inhalt zu lesen. Wird die Versorgungsspannung ausgeschaltet, bleiben ihre Inhalte erhalten.
- Speicher mit sequentiellem Zugriff:
Auf die Speicherzellen kann nur nacheinander zugegriffen werden (z.B. Magnetbandspeicher).
- Speicher mit zyklischem oder halbdirektem Zugriff:
Die Speicherzellen rotieren und sind deshalb nur in periodischen Zeitabschnitten erreich-

bar. Beispiele für diese Speicherart sind Magnetplattenspeicher und Magnettrommelspeicher.

Mit *Zykluszeit* wird die Zeitdauer bezeichnet, die vom Beginn eines Speichervorgangs bis zum Beginn des nächsten Speichervorgangs mindestens vergeht.

Aus der *Speicherkapazität* geht die Anzahl der verfügbaren Speicherzellen hervor. Eine Speicherzelle entspricht einem Speicherelement. $1024 = 2^{10}$ Byte entsprechen einem Kilobyte = 1KByte, 1024KByte einem Megabyte = 1MByte, 1024MByte einem Gigabyte = 1GByte usw. (siehe Kapitel 2).

Statische Speicher behalten ihren Inhalt bei, solange die Versorgungsspannung anliegt. *Dynamische Speicher* dagegen verlieren ihren Inhalt nach einem bestimmten Zeitintervall, so dass sie regelmäßig nachgeladen werden müssen. Diesen Vorgang bezeichnet man mit Refreshing.

Aus dem *Speicherprinzip* gehen die Funktionweise und der Aufbau des Speichers hervor. Es besagt, wie die Bits in den Speicherzellen gespeichert werden. Die Speicherprinzipien variieren stark mit dem verwendeten Baumaterial und den physikalischen Formen, durch die die Bits repräsentiert werden. So gibt es Halbleiterspeicher, Magnetspeicher, optische sowie mechanische Speicher. Die verschiedenen Speicherarten unterscheiden sich hinsichtlich Geschwindigkeit (Zugriffs- und Zykluszeit) und Preis. Aus diesem Grund kommen in Computern in der Regel mehrere Speicherprinzipien zum Einsatz, die bzgl. ihrer Geschwindigkeit hierarchisch angeordnet werden können (Abb. 11.1). Während zur Realisierung des Hauptspeichers beispielsweise der schnelle, aber teure Halbleiterspeicher eingesetzt wird, verwendet man die preisgünstigeren, langsameren Speicherarten als Massenspeicher bzw. zur Archivierung großer Datenmengen (Backup).

11.2 Halbleiterspeicher

Halbleiterspeicher werden hauptsächlich als Haupt- und Festwertspeicher von Computersystemen eingesetzt. Zunehmend dienen sie auch als schnelle Zwischenspeicher für Massenspeicher mit höherer Zugriffsdauer. Ferner werden auch Datenregister von Mikroprozessoren durch Halbleiterspeicherelemente (Flipflops, siehe Kapitel 7) realisiert. Die Vorteile des Halbleiter-speichers liegen in seinen kurzen Zugriffszeiten, die ebenso wie die Prozessorleistung für eine hohe Leistungsfähigkeit des Gesamtsystems erforderlich sind.

11.2.1 Halbleiterbauelemente

Beim Halbleiterspeicher sind die Speicherelemente durch Halbleiterbauelemente realisiert. Halbleiter sind Stoffe, deren elektrische Leitfähigkeit zwischen der von Metallen und der von Isolatoren liegt. Bekannte Halbleiter sind das Silizium (*Si*), das Germanium (*Ge*) und das Galliumarsenid (*GaAs*). Durch gezielte Verunreinigung (Dotierung) mit anderen chemischen Elementen (Störstellenatome) können die elektrischen Eigenschaften des Halbleiters geändert werden. Die Störstellenatome bewirken entweder als Elektronenspender einen Elektronenüberschuss (n-Halbleiter) oder als Elektronenempfänger einen Elektronenmangel, der so genannte Löcher erzeugt, die als positive Ladungsträger aufgefasst werden können (p-Halbleiter). Halbleiterbauelemente entstehen durch geschickte Kombination dieser unterschiedlich dotierten Halbleiter. Die wichtigsten Halbleiterbauelemente sind Dioden und Transistoren.

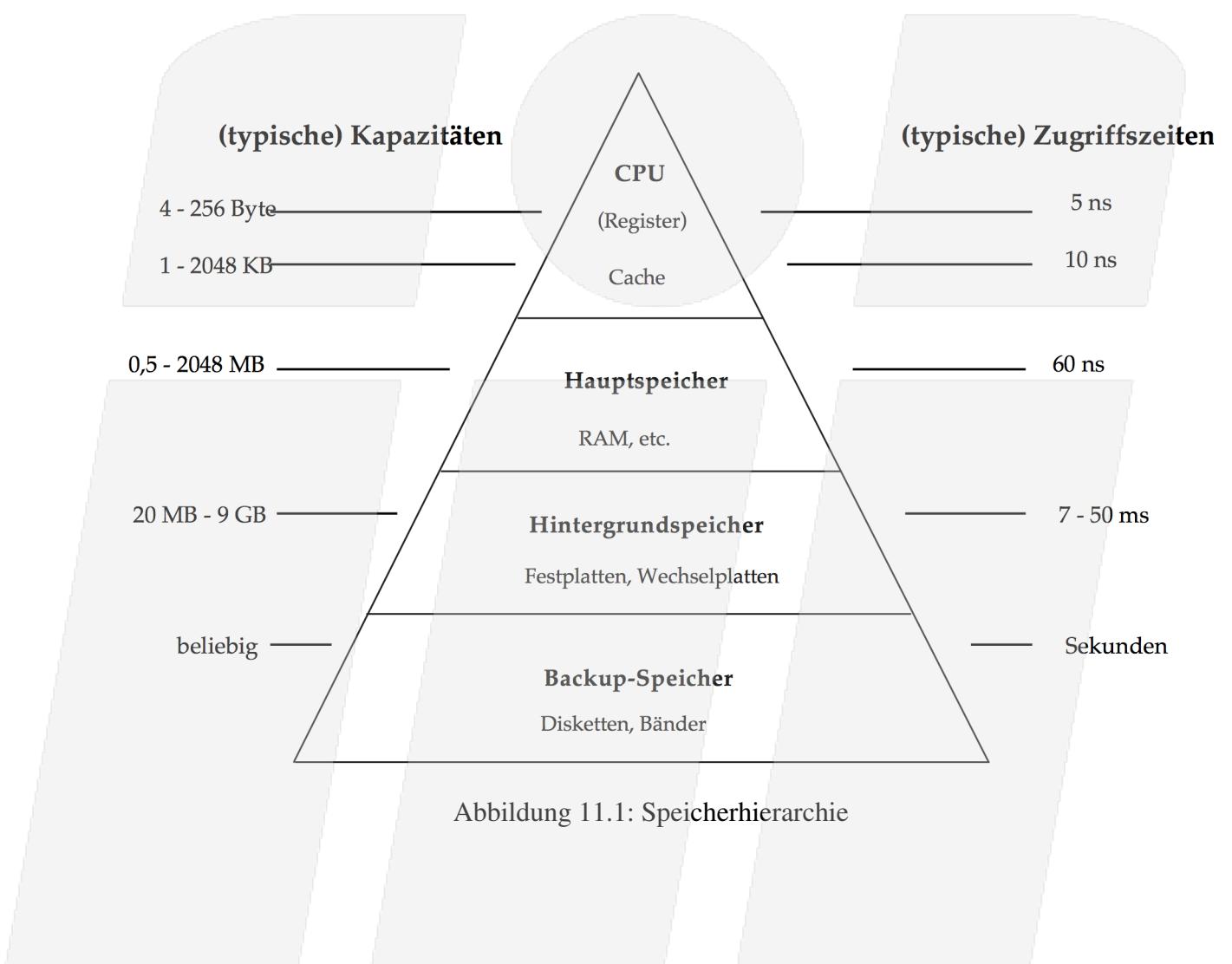


Abbildung 11.1: Speicherhierarchie

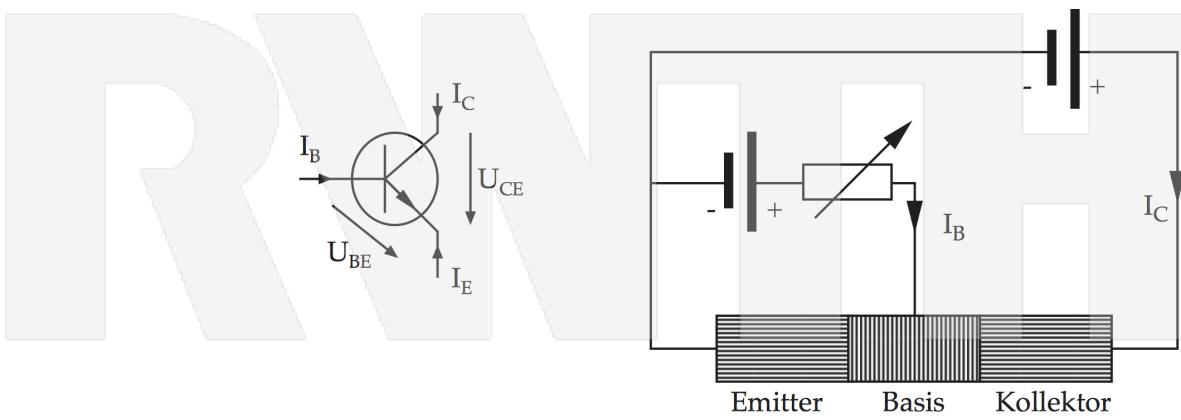


Abbildung 11.2: Schaltzeichen und Aufbau eines Bipolartransistors.

Bei den Transistoren unterscheidet u. a. die folgenden Bauarten: Bipolartransistoren und Feldeffekttransistoren.

Ein *Bipolartransistor* besteht aus drei unterschiedlich dotierten Halbleiterschichten, die Basis, Emitter und Kollektor genannt werden. Je nach Reihenfolge der Dotierungen spricht man von npn- und pnp-Transistoren. Die Abbildung 11.2 zeigt den Aufbau und das Schaltsymbol eines Bipolartransistors.

Im Folgenden soll das Verhalten des npn-Transistors kurz erläutert werden: Nach Anlegen einer Spannung $U_{BE} > 0$ zwischen Basis und Emitter fließt ein Basisstrom I_B . Liegt zwischen Kollektor und Emitter eine Spannung U_{CE} an, so fließt vom Kollektor zum Emitter ein zu I_B proportionaler Kollektorstrom I_C . Ein Transistor kann als elektronischer Schalter aufgefasst werden: Liegt zwischen Basis und Emitter eine Spannung U_{BE} an, so ist die Kollektor-Emitter-Strecke leitend. Liegt keine Spannung an, so ist sie sperrend.

Abbildung 11.3 zeigt den Aufbau eines *Feldeffekttransistors (FET)*. Er besteht aus einem

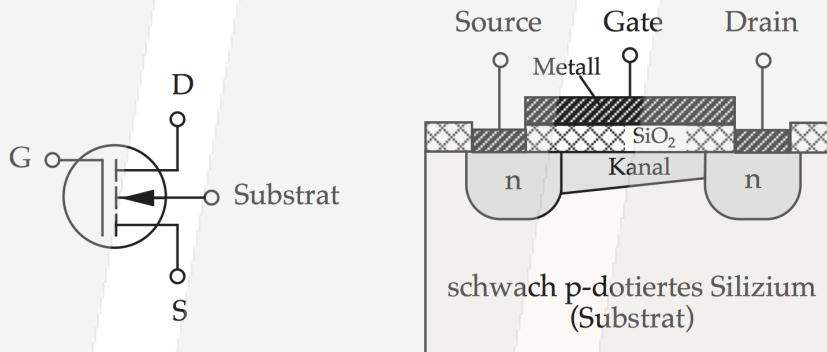
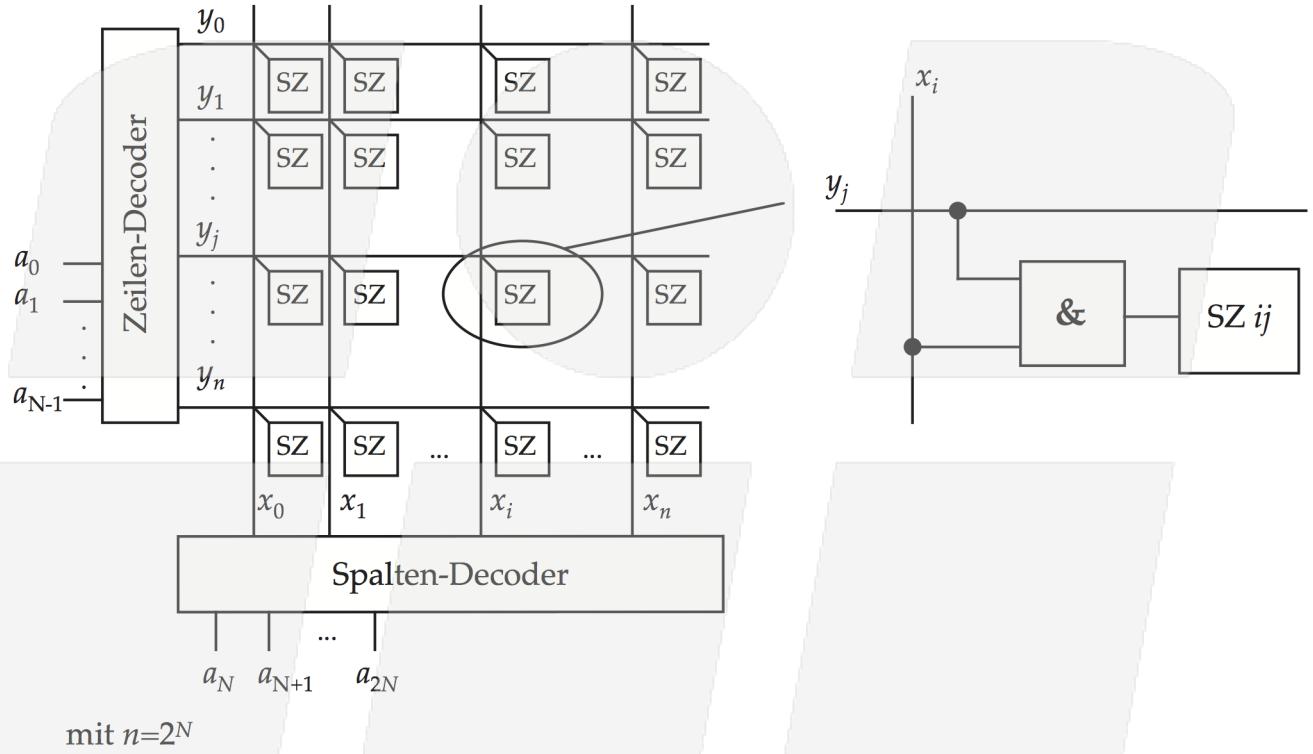


Abbildung 11.3: Schaltzeichen und Aufbau eines n-Kanal-MOS-FETs.

schwach p-dotierten Siliziumkristall, in das zwei n-dotierte Inseln eingebracht sind. Über dem Bereich zwischen den Inseln liegt eine isolierende Oxidschicht (meist SiO_2), die von einer Metallschicht abgeschlossen wird. Die beiden Inseln werden ebenfalls von einer Metallschicht überdeckt. Eine solche Struktur aus Metall, Oxid und Halbleiter nennt man auch MOS-Struktur (*Metal Oxid Semiconductor*). Deshalb heißen so aufgebaute Feldeffekttransistoren auch MOS-FETs. Die beiden Inseln bezeichnet man mit *Source (S)* und *Drain (D)*, die isolierte Metallschicht mit *Gate (G)*. Liegt zwischen Gate und Source eine Spannung ($U_{GS} > U_S$, U_S : Schwellspannung) an, so bildet sich unterhalb des Oxids eine n-leitende Inversionsschicht, die beide n-dotierte Inseln leitend miteinander verbindet. Liegt zwischen Drain und Source eine Spannung U_{DS} , so fließt ein Drainstrom I_D . Ein MOSFET dieser Bauart heißt deshalb n-Kanal-FET. Daneben gibt es auch p-Kanal-FETs. Ihr Aufbau ist entsprechend umgekehrt. Da durch das isolierte Gate kein Strom fließt, ist die Steuerung des FETs leistungslos. Ein MOSFET kann ebenso wie ein Bipolartransistor als elektronischer Schalter verwendet werden.

11.2.2 Integrierte Schaltungen

Werden Schaltungen aus Halbleiterbauelementen auf ein einzelnes Siliziumplättchen (Chip) aufgebracht, so erhält man integrierte Schaltungen (IC - integrated circuits). Je nach Struktur der verwendeten Bauelemente (Bipolartransistoren oder MOS-FETs) unterscheidet man zwischen bipolarer Technologie und MOS-Technologie. Beide Technologieformen lassen sich weiter in



mit $n=2^N$

SZ: Speicherzelle

Abbildung 11.4: Aufbau eines RAMs

Schaltkreisfamilien unterteilen. Schaltkreisfamilien sind gekennzeichnet durch eine bestimmte Zusammenschaltung gleichartiger Bauelemente im Baustein. Bausteine einer Schaltkreisfamilie sind untereinander verträglich und lassen sich leicht zu größeren Systemen zusammenschalten. So würden z. B. die in Kapitel 6 beschriebenen Schaltnetze und Schaltwerke aus Gattern und Flipflops einer Schaltkreisfamilie realisiert werden. Die wichtigsten Schaltkreisfamilien und deren Eigenschaften sind in Abbildung 11.5 aufgeführt. Abbildung 11.6 zeigt die Realisierung von Gatterbausteinen in TTL und CMOS.

11.2.3 Schreib-Lese-Speicher (RAM)

Bei einem RAM-Speicher kann nach Vorlage der Adresse auf jede Speicherzelle direkt zugegriffen werden. Die einzelnen Speicherzellen sind matrixförmig angeordnet. Abbildung 11.4 zeigt eine solche Anordnung. Eine bestimmte Speicherzelle kann durch die Angabe der Adresse mit einem Spalten- und einem Zeilendecoder decodiert werden. Der Zugriff erfolgt, wenn die Adressenbedingung $x_i = y_j$ erfüllt ist. Die Adresseneingänge müssen also so decodiert werden, dass immer genau eine Speicherzelle selektiert wird. Ferner verfügt ein RAM-Baustein über eine Schreib-Leseumschaltung und einen Chip-Select-Anschluss. Dieser Anschluss dient der Auswahl unter mehreren Speicherbausteinen, die an einem gemeinsamen Bus betrieben werden.

Bipolare Familien	<i>RTL</i>	- Resistor-Transistor-Logic
	<i>DTL</i>	- Diode-Transistor-Logic
	<i>TTL</i>	- Transistor-Transistor-Logic
	<i>ECL</i>	- Emitter-Coupled-Logic
	<i>I²L</i>	- Integrated-Injection-Logic
MOS-Familien	<i>PMOS</i>	- P-Channel-MOS
	<i>NMOS</i>	- N-Channel-MOS
	<i>CMOS</i>	- Complementary-MOS

Verlustleistung pro Gatter in mW

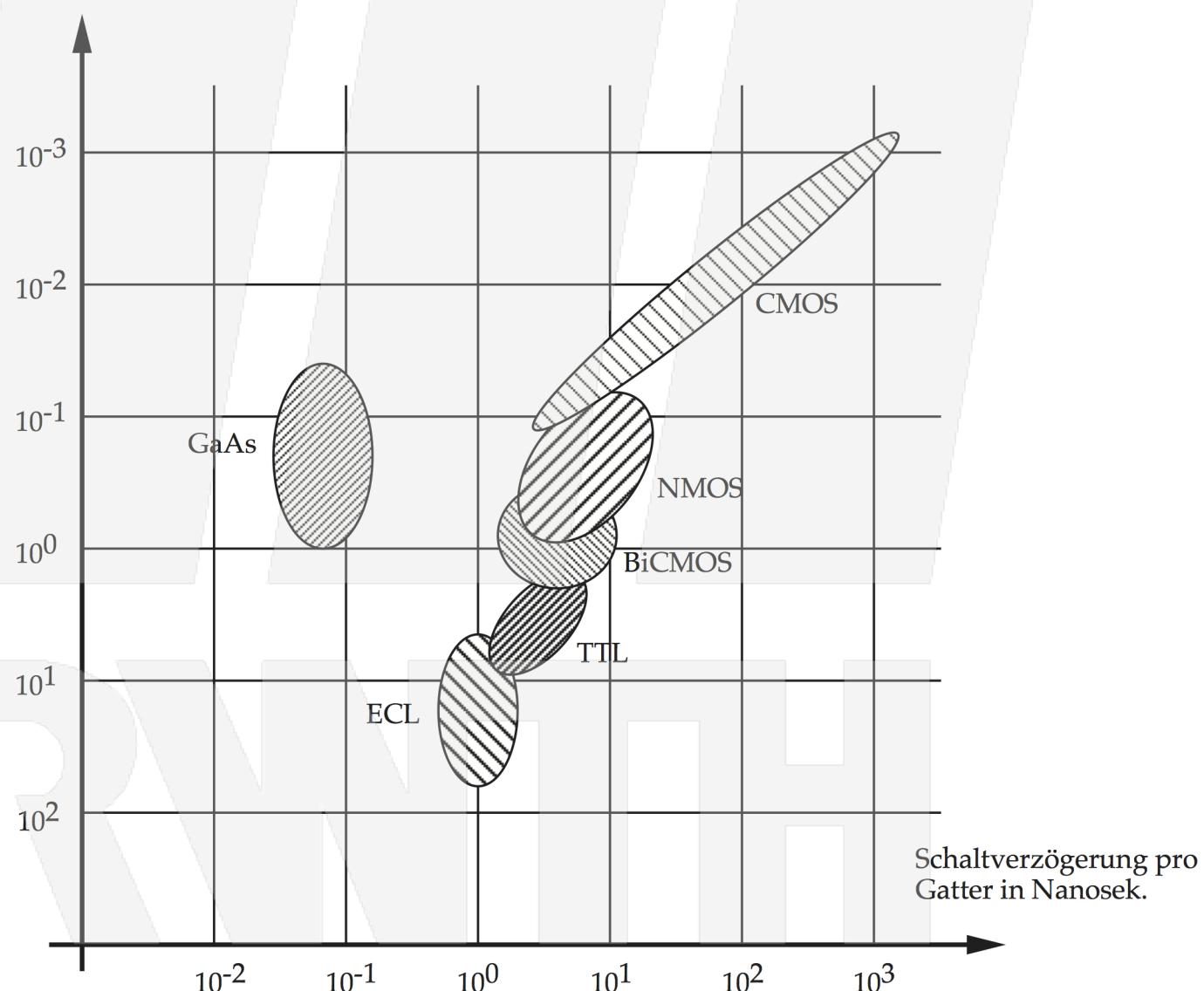


Abbildung 11.5: Schaltkreisfamilien und deren Eigenschaften

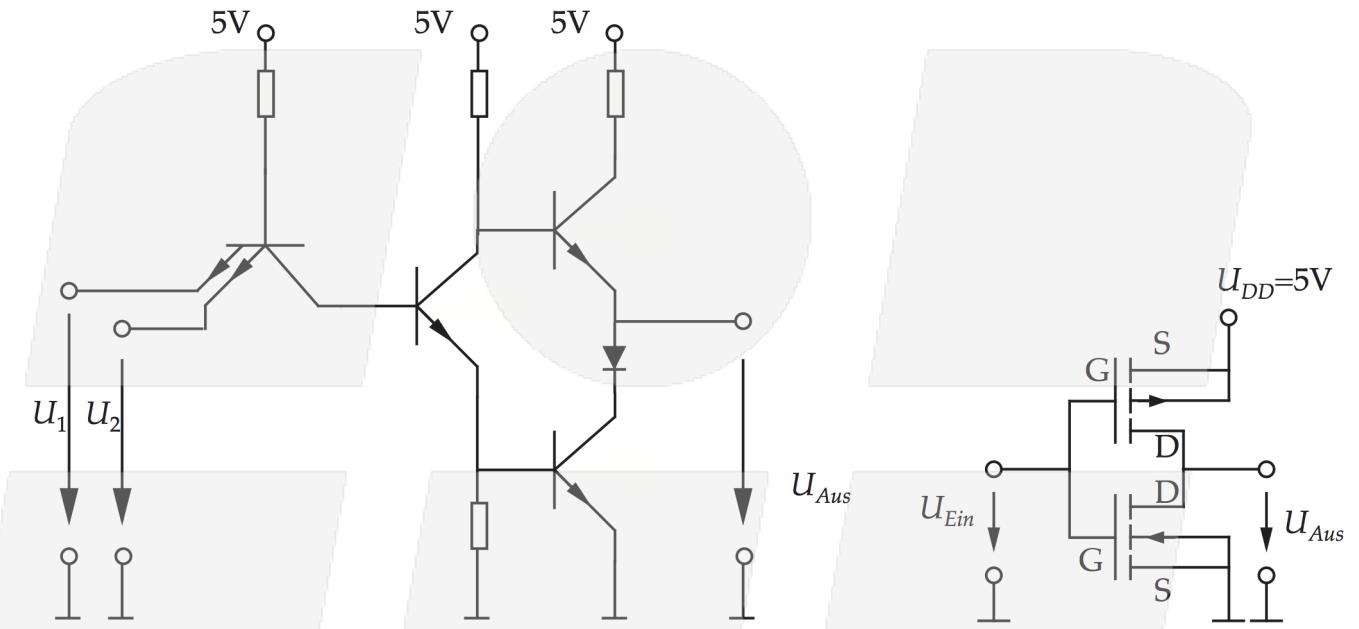


Abbildung 11.6: Realisierung eines NAND-Gatters in TTL und eines Inverters in CMOS

Statische RAMs

Bei statischen RAMs (SRAMs) bleibt der Speicherinhalt solange erhalten, bis er durch einen Schreibvorgang geändert wird. Abbildung 11.7 zeigt eine statische Speicherzelle in TTL-Technik.

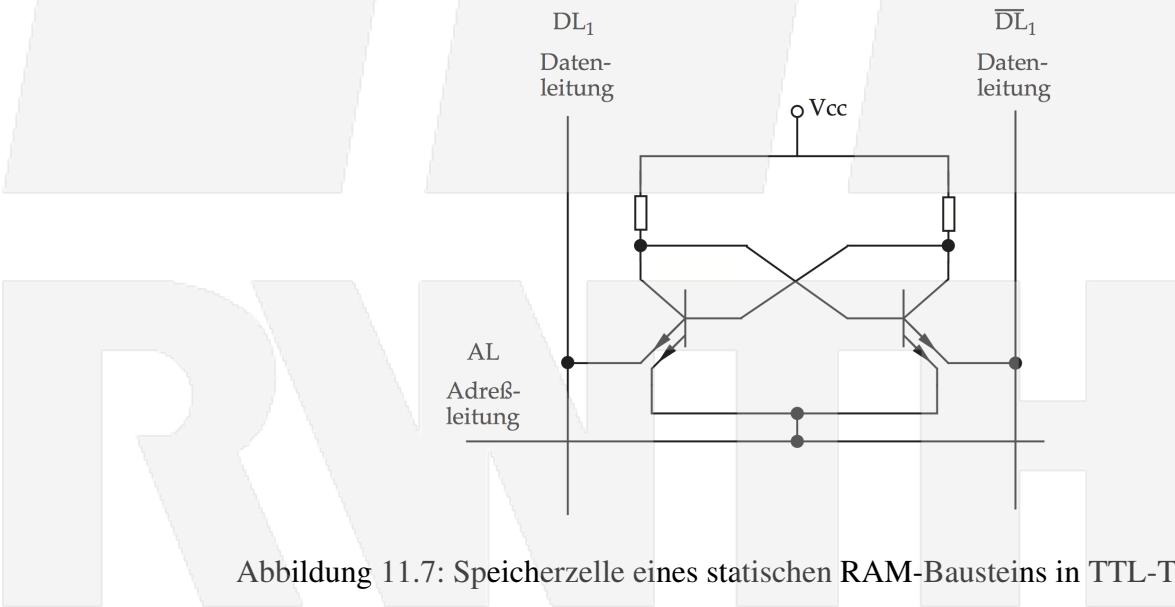


Abbildung 11.7: Speicherzelle eines statischen RAM-Bausteins in TTL-Technik

Dynamische RAMs

In dynamischen RAMs (DRAMs) werden die Bits als Kondensatorladungen gespeichert. Abbildung 11.8 zeigt die Schaltung einer dynamischen Speicherzelle in CMOS - Technik. Da sich der Kondensator aufgrund unvermeidlicher Leckströme im Lauf der Zeit entlädt, muss er in

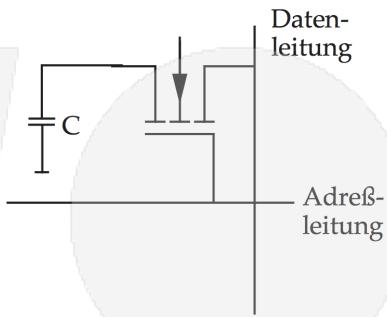


Abbildung 11.8: Schaltung einer dynamischen Speicherzelle

kurzen Zeitabständen (alle 2 - 100 ms) mit einem Refresh-Impuls nachgeladen werden. Beim Lesen der Daten wird der Speicherinhalt ebenfalls zerstört. Zum Feststellen des ursprünglichen Zustands und zum Auffrischen des Speicherinhalts sind deshalb zusätzliche Schaltungen, die im Dynamic RAM Controller zusammengefasst sind, nötig. Die Leistungsaufnahme dynamischer RAM-Bausteine ist deshalb höher als bei statischen RAMs. Andererseits benötigen dynamische Speicherbausteine weniger Bauelemente pro Speicherzelle und erreichen so gegenüber statischen Bausteinen bei gleicher Integrationsdichte und gleichen Kosten ungefähr die vierfache Speicherkapazität.

11.2.4 Assoziativspeicher

Dies ist ein Speicher mit wahlfreiem Zugriff, bei dem der Zugriff nicht nur auf eine bestimmte Speicherzelle erfolgt, sondern gleichzeitig auf alle Speicherplätze, deren Inhalt einer bestimmten Vorgabe genügt. Abbildung 11.9 zeigt den Aufbau eines Assoziativspeichers mit Suchregister, Maskenregister, Vergleichslogik und Vergleichsindikator.

Zur Adressierung werden zunächst im Maskenregister Bitpositionen gesetzt, die dem gesuchten Zelleninhalt entsprechen. Der Inhalt des Suchregisters wird dann für diese Bitpositionen gleichzeitig mit allen Speicherzellen verglichen. Im Vergleichsindikator zeigen die mit 1 belegten Stellen an, welche Speicherzellen die Inhaltsspezifikation erfüllen.

Assoziativspeicher sind wegen der aufwändigen parallelen Vergleichslogik teuer und werden deshalb nur für Spezialzwecke eingesetzt, bei denen es auf hohe Geschwindigkeit ankommt (Speicherverwaltung).

11.2.5 Festwertspeicher (ROM)

Auf einen *ROM*-Speicher kann wie bei RAM-Speichern auf jede Speicherzelle direkt zugegriffen werden. Allerdings kann ihr Inhalt nicht mehr vom Programm aus geändert werden und bleibt auch nach Abschalten der Betriebsspannung erhalten. Neben den klassischen ROMs, die schon vom Hersteller programmiert werden, gibt es auch vom Anwender programmierbare Festwertspeicher, die je nach Bauart ein oder mehrmals beschrieben werden können. Die wichtigsten ROM-Varianten (vgl. Tabelle 11.1) sollen im Folgenden kurz erläutert werden:

- **Maskenprogrammierte ROMs (MROMs):**

Bei den MROMs wird der Speicherinhalt bei der Herstellung mit einer speziellen Maske

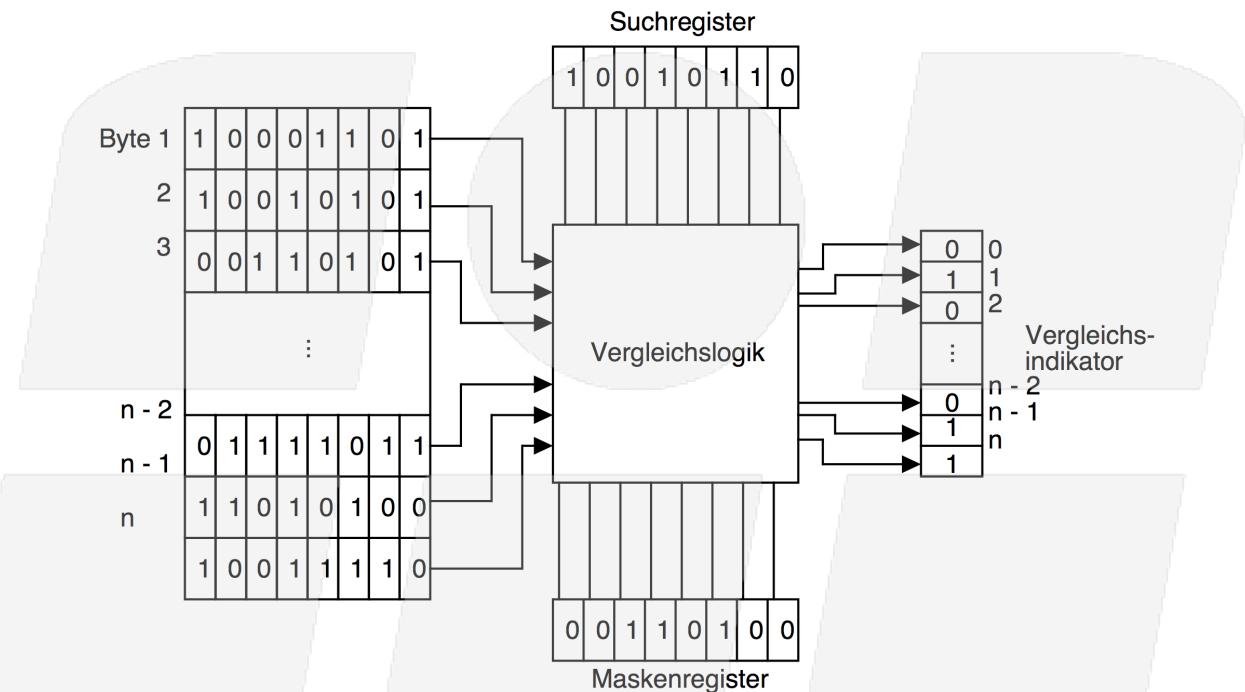


Abbildung 11.9: Assoziativspeicher

ingegeben, durch die der Siliziumchip belichtet wird, nachdem er mit einer fotoempfindlichen Schicht versehen wurde. Dieses Verfahren lohnt sich nur bei einer Produktion von großen Stückzahlen. (Bsp: Computer-ROMs mit BIOS oder Betriebssystemroutinen.)

- **Programmable ROM - PROM:**

Bei PROMs kann der Inhalt einmal vom Anwender mit Hilfe eines speziellen Programmiergeräts festgelegt und danach nicht mehr geändert werden. Abbildung 11.10 zeigt die Realisierung einer PROM-Speicherzelle. Als programmierbares Bauelement wird eine zerstörbare Verbindungsleitung verwendet.

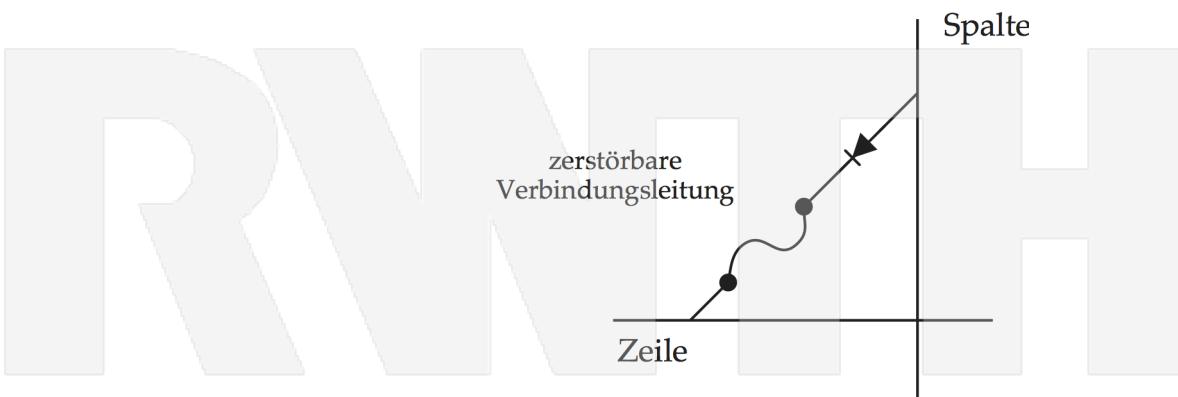


Abbildung 11.10: Speicherzelle eines PROMs

- **UV-löscherbarer Festwertspeicher (EPROM):**

Ein EPROM (*Erasable ROM*) lässt sich vom Anwender programmieren und mit UV-Licht wieder löschen. Hierfür hat das Gehäuse an seiner Oberseite ein verschließbares Fenster,

Tabelle 11.1: Vergleich der Speichervarianten

	RAM	ROM			
		MROM	PROM	EPROM	EEPROM
Schreiben Schreibdauer	unbegrenzt 10-200 ns	1mal bei Herstellung	1mal Minuten	ca 100mal Minuten	$10^4 - 10^5$ mal Millisekunden
Lesen Lesedauer	unbegrenzt 10-200ns	unbegrenzt ca. 100ns	unbegrenzt 10-300ns	unbegrenzt 30-300ns	unbegrenzt 30-300ns

durch das der Schaltkreis von der Strahlung erreicht werden kann. Anschließend kann das EPROM neu programmiert werden. EPROMs werden vorzugsweise in der Entwicklung eingesetzt.

- **Elektrisch lösbarer Festwertspeicher (EEPROM):**

Ein EEPROM (*Electrically Erasable PROM*) kann durch elektrische Signale gelöscht werden. Dabei sind der Spannungswandler zur Erzeugung der Programmierspannung und der Timer zur Festlegung der Impulsdauer auf dem Baustein integriert. Es ist daher kein separates Programmiergerät nötig.

11.3 Magnetische Massenspeicher

Data Cartridge

Ein *Data Cartridge* ist eine Magnetbandkassette, die sich in einem Kunststoff-Metallgehäuse befindet. Da diese Cartridges und die Data-Cartridge-Laufwerke weltweit dem QIC-Standard (Quarter Inch Committee) entsprechen und neu entwickelte QIC-Standards abwärtskompatibel sind, können Data Cartridges innerhalb von QIC-Laufwerken gleicher Leistungsstufe ausgetauscht werden. Die Datentransferrate beträgt bis zu 1200 KByte/s. Die maximale Aufzeichnungskapazität beträgt 4 GByte.

8 mm Data Tape

Die Firma Exabyte entwickelte speziell zur Datensicherung professionelle Video8-Laufwerke und Datenbandkassetten, die wie Heim-Videorecorder das Helical-Scan Aufzeichnungsverfahren (Schrägspuraufzeichnung) nutzen. Je nach verwendeter Schnittstelle können bis zu 5 MByte/s übertragen werden. Die maximale Kapazität beträgt 24 GByte.

4mm Data Tape (DAT)

DAT-Tapes (Digital Audio Tape) wurden entwickelt, um Musik in digitaler Qualität aufzunehmen. Das Verfahren wurde aus der 8 mm-Videotechnik abgeleitet. Mit einer Datentransferrate von max 5 MByte/s können bis zu 12 GByte, bei Datenkompression bis zu 24 GByte auf einer Kassette gespeichert werden.

VHS-Videobänder

VHS-Videobänder werden ebenfalls als Speichermedium benutzt. Sie haben eine Kapazität von ca. 14 GByte und transferieren pro Sekunde bis zu 2 MByte. Ein Bandroboter für 600 VHS-Videokassetten wird an der Universität Tübingen als 8,5 TByte-Fileserver eingesetzt.

Disketten

Der Einsatz von *Disketten* ist mit der z.Zt. maximalen Speicherkapazität von 2,88 MByte (3,5" ED) begrenzt. Die gängige 3,5" HD Diskette hat formatiert 1,44 MByte Kapazität. Es gibt zwar bereits Laufwerkentwicklungen mit 4 MByte Kapazität (2,88 MByte formatiert), die auch 3,5" DD- und HD-Disketten bearbeiten können, jedoch sind die 4 MByte Disketten nicht auf den heute verbreiteten Laufwerken einsetzbar.

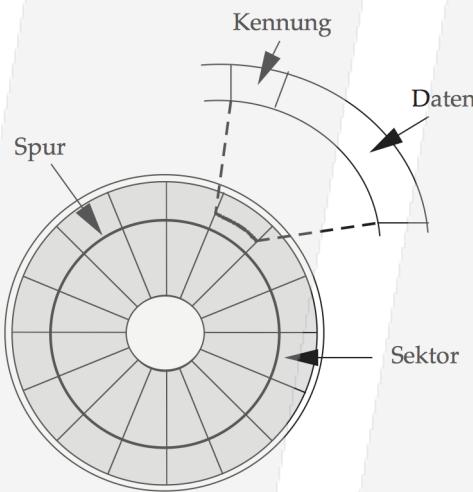


Abbildung 11.11: Organisation einer Platten- bzw. Diskettenseite

Der Name Floppy Disk leitet sich von der flexiblen Scheibe, die aus Kunststofffolie besteht, ab. Der Überzug mit einer Lackschicht enthält magnetisierbares Eisenoxyd. Ein Indexloch auf der Diskette dient der Erkennung des Spuranfangs mittels Lichtschranke. Die Rotationgeschwindigkeit beträgt 300 U/min.

Die Datenspeicherung erfolgt auf konzentrischen, magnetisierbaren Spuren (Abb. 11.11), die in „tortenstückartige“ Sektoren unterteilt sind. Ein Sектор ist die kleinste adressierbare Einheit (128, 256 oder 512 Bytes). Er nimmt einen Datensatz auf, der die Nutzdaten und ein Adressfeld mit Spuradresse, Kopfadresse (bei 2 Diskettenoberflächen), Sektornummer und Länge des Datenfeldes (bei variabler Sektorlänge) enthält.

Festplattenspeicher (Winchesterplatte)

Die *Festplatte* ist der für alle Standardanwendungen am häufigsten eingesetzte Massenspeicher, obwohl sie nicht auswechselbar ist. Der übliche Kapazitätsbereich liegt bei 3,5" Festplatten bis 75 GByte. Kleinere Festplatten mit 2,5" werden mit bis zu 25 GByte angeboten. Die kleinste derzeit erhältliche Festplatte hat einen Durchmesser von 1,3" und eine Speicherkapazität von 1 GByte.

Bei konventionellen Platten- und Diskettenspeichern sind die Lese-/Schreibköpfe und die Plattenoberflächen durch Staubkörner sehr gefährdet (siehe Abb. 11.13). Die Winchestertechnologie zeichnet sich durch eine staubdichte Kapselung aus, die Platten und Laufwerk in einem Gehäuse integriert.

Im Gegensatz zur Diskette, die nur bei Zugriff angetrieben wird, rotiert die Harddisk kontinuierlich (3000-10000 U/min). Meistens werden mehrere Platten zu einem Plattenstapel vereint, so dass jede Plattenoberfläche mindestens einen eigenen Schreib-/Lesekopf hat. Bei manchen Platten ist eine Oberfläche zur präzisen Steuerung der Platte reserviert.

Die Abtasttechnik ist ähnlich wie bei der Wechselplatte. Ein Mikroprozessor steuert die am Magnetkopfkamm befindlichen Schreib-/Leseköpfe (vgl. Abb. 11.12). Die Adressierung erfolgt über Zylinder, Spur und Sektor.

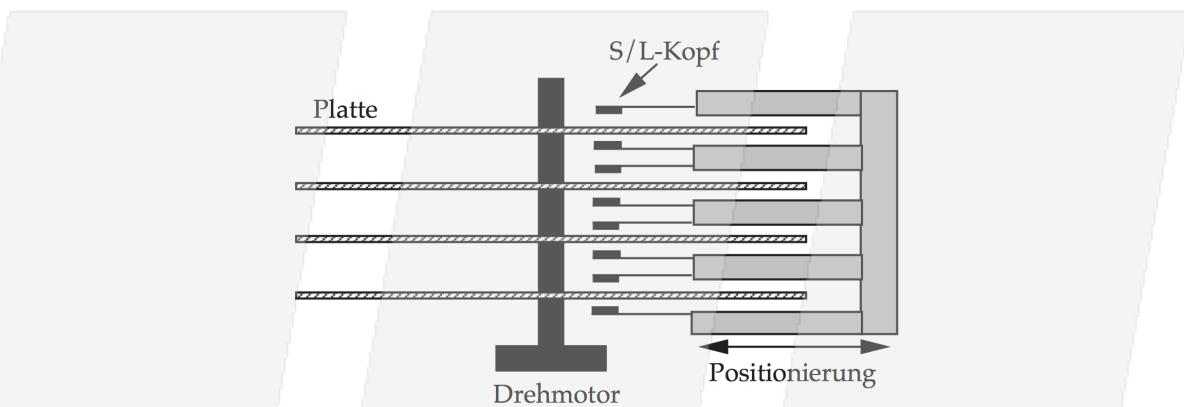


Abbildung 11.12: Seitenansicht einer Festplatte

Beim Lesen und Schreiben von Sektoren entstehen Zeitverluste durch CPU/DMA-Umschaltung oder eine zu langsame CPU. Dies wird ausgeglichen, indem die logisch aufeinanderfolgenden Sektoren physikalisch über mehrere Sektoren hinweg auseinandergelegt werden. Wieviele physische Sektoren bis zum nächsten logischen Sektor zu überspringen sind, legt der Interleave-Faktor fest. Ist er geeignet eingestellt kann ein kontinuierliches Lesen der Daten von der Platte erfolgen, ohne dass dazu unnötige Umdrehungen der ständig rotierenden Platte erforderlich sind.

Die meisten Controller verfügen über Halbleiterspeicher zum Puffern von Sektoren oder ganzen Tracks.

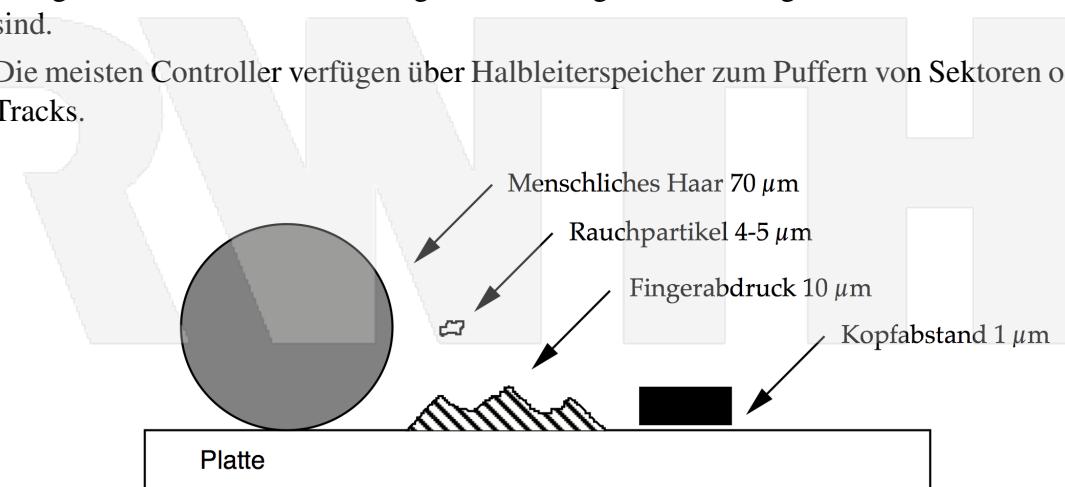


Abbildung 11.13: Oberflächenabstand eines Schreib-/Lesekopfes

Rahmenfestplatte

Hier befindet sich eine normale Winchesterplatte in einem Rahmen mit Kontaktleiste und kann leicht komplett gewechselt werden. Leider sind bei häufigem Wechseln die Kontaktleisten sehr störanfällig.

11.4 Optische Massenspeicher

CD-ROM

Wie die Audio-CD wird die Compact-Disk bei der Herstellung beschrieben und ist dann nur noch lesbar. Sie wird hauptsächlich dann eingesetzt, wenn größere feststehende Datenmengen mehreren Benutzern in nur lesbarer Form zur Verfügung stehen sollen, z. B. für Software oder Manuals. Die CD wird mit Hilfe eines Lasers ausgelesen.

In dem Trägermaterial der CD-Oberfläche sind Vertiefungen („Pits“) eingestanzt. Die Flächen zwischen Pits werden „Lands“ genannt. Scheint der Laser auf ein Land (Vertiefung im Trägermaterial), wird das Licht reflektiert und von einer Photozelle wahrgenommen. Ein Pit zerstreut das einfallende Laserlicht, so dass die Photozelle keine Reflektion wahrnimmt. Ein Wechsel zwischen Pit und Land repräsentiert eine 1, kein Wechsel (also ein Pit bzw. ein Land) wird als 0 interpretiert.

Abbildung 11.14 zeigt eine Skizze der Informationsdarstellung auf einer CD-ROM sowie das Prinzip der CD-ROM-Leseoptik.

Die Daten werden vom Hersteller nicht durch einen Laser aufgebracht, sondern im Mastering-Verfahren (wie bei der Vinyl-Platte) aufgestanzt. Die bei allen optischen Speichermedien typische hohe Schreibdichte ermöglicht hier eine Speicherkapazität von bis zu 1 GByte. Dabei handelt es sich nicht um die reine physische Speichergröße, da Informationen auf der CD mit Hilfe von zahlreichen Paritätsbits derart abgespeichert werden, so dass Lesefehler bemerkt und teilweise sogar korrigiert werden können.

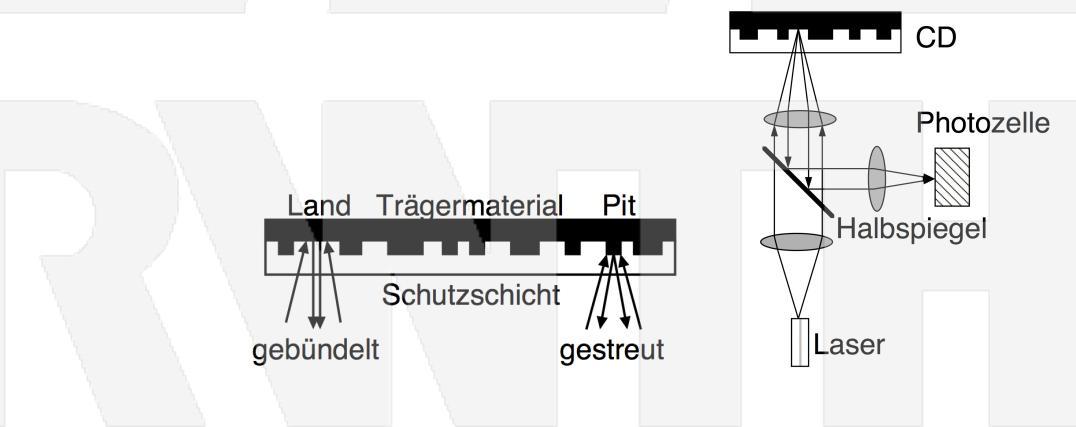


Abbildung 11.14: Informationsdarstellung auf einer CD-ROM (siehe auch Tab. 11.2)

CD-R/CD-RW

Die CD-R (CD-Recordable) ist eine einmal beschreibbare CD-ROM, die in jedem CD-ROM Laufwerk gelesen werden kann. Ursprünglich als Photo-CD entwickelt ist sie nun als allgemein

Tabelle 11.2: Kennwerte einer CD-ROM (bei 1-/44-facher Geschwindigkeit)

Kapazität	656	MByte
Mittl. Zugriffszeit	400/85	ms
Max. Zugriffszeit	1,2	sec
Datentransfer	150/6600	KByte/sec
Rotationsgeschw.	230/8000	U/min
Schreibdichte	16 000	tpi
Schreibweise		spiralig

beschreibbares Medium mit Datenkapazitäten zwischen 180 MByte (8cm-CD) und 800 MByte (12cm-CD) für den Anwender erhältlich. Die Produktionszeit verkürzt sich wesentlich, da die Mastererstellung wegfällt.

Im Gegensatz zur gestanzten CD-ROM mit einer Trägerschicht besitzt die CD-R eine Schicht mit einem lichtempfindlichen organischen Farbstoff sowie einer zusätzlichen Goldschicht. Der lichtempfindliche Farbstoff besitzt die gleichen Brechungseigenschaften wie eine leere CD. Auf ihr sind keine Pits vorhanden sondern nur Lands. Zum Schreiben wird die Intensität des Lasers erhöht, welcher die reflektierende und die Farbstoffschicht erhitzt. Dadurch kommt es zu einer chemischen Reaktion und die Reflektionseigenschaften der erwärmeden Stellen ähneln denen der Pits. Liest ein CD Laufwerk eine CD-R, so erkennt es an den „gebrannten“ Stellen ein Pit und an den ungebrannten Stellen Land. Die CD-R verhält sich jetzt wie eine CD-ROM. Allerdings ist bei einer CD-R der Kontrast zwischen Pit und Land deutlich schlechter als bei einer gepressten CD-ROM.

Eine CD-RW (CD-Rewritable) besitzt eine reflektierende Schicht (z. B. aus einer Silber-Indium-Antimonium-Tellurium-Legierung). Diese Schicht wechselt bei unterschiedlichen Temperaturen zwischen einer kristallinen und amorphen Struktur (*Phase Change*). Die kristalline Struktur (entsteht bei ca. 700°C) hat die reflektierenden Eigenschaften von Lands, während die amorphe Struktur (entsteht bei ca. 200°C) das Laserlicht absorbieren.

WORM (Write Once Read Many)

Dieses Speichermedium lässt sich ebenfalls vom Anwender nur einmal beschreiben, ist aber ergänzbar und kann beliebig oft gelesen werden. Die in unterschiedlichen Formaten (5,25" bis 14") erhältlichen optischen Speicherplatten haben derzeit eine Kapazität von bis zu 6 GByte.

Bei der optischen WORM-Aufzeichnungstechnik erhitzt ein Laser eine Schicht unter der Plattenoberfläche und erzeugt kleine Blasen (Abbildung 11.15). Diese Veränderung kann mit demselben Laserstrahl aufgrund der geänderten Lichtreflexion erfasst werden.

Digital Versatile Disc — DVD

Die DVD hat dieselben Ausmaße wie die CD-ROM: 12 cm Durchmesser und 1,2 mm Dicke. Im Gegensatz zur CD-ROM sind jedoch zwei jeweils 0,6 mm dicke Discs zusammengeklebt, so dass beide Seiten einer DVD Daten enthalten können. Jede Seite kann ihrerseits zwei informationstragende Schichten enthalten (Dual Layer Technologie).

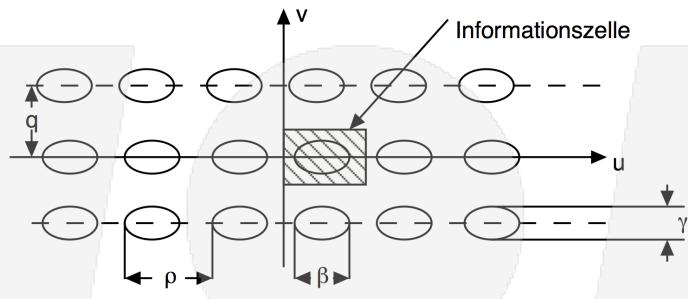


Abbildung 11.15: Informationszellen einer optischen Platte, die Abmessungen der einzelnen Elemente beschreibt Tabelle 11.3

Tabelle 11.3: Kenngrößen einer optischen Platte

Kerbentiefe	0,5	μm
Kerbenlänge := β	0,4-1	μm
Kerbenbreite := γ	0,4	μm
Spurabstand := q	1,6	μm
Kapazität	2	GByte

Zum Lesen der einzelnen Schichten wird die Brennweite der Laseroptik reguliert. Ein Wechsel zwischen den beiden Speicherschichten findet innerhalb von 5 ms statt. Dabei ändert sich die Brennweite um 40 μm . Somit erhält man die in Tabelle 11.4 aufgeführten Speicherkapazitäten.

Die im Vergleich zur CD-ROM erhöhte Speicherkapazität pro Schicht ist bedingt durch einen geringeren Abstand zwischen den Spuren und eine geringere Mindestlänge der Pits, ermöglicht durch den Umstand, dass der Laser nur noch eine 0,6mm dicke Schicht des Polycarbonat-Trägers bis zur Informationsschicht durchdringen muss. Durch Verwendung kurzwelligerer Laserstrahlen lässt sich eine bessere Fokussierung des Lasers erreichen.

Bei der Dual-Layer DVD muss die jeweils äußere Schicht halbdurchlässig sein, damit der Laser die Information der zweiten (inneren) Schicht lesen kann. Die Herstellung erfolgt so, dass auf dem Kunststoffträger aus Polycarbonat, auf den mit einem Stempel (Master) die erste Informati-

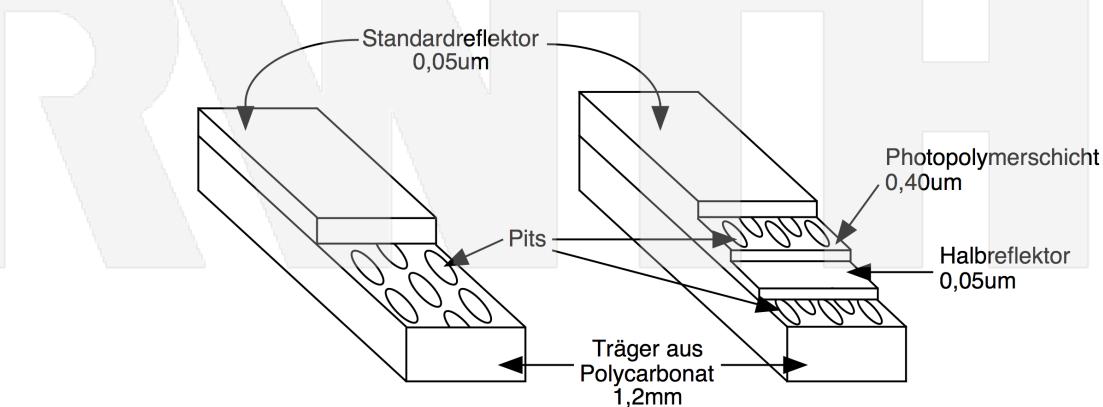


Abbildung 11.16: Vergleich des Aufbaus von Standard CD (links) und DVD Dual Layer (rechts).

Tabelle 11.4: Speicherkapazitäten in GByte der verschiedenen DVD-Formate

	Single Layer	Dual Layer
Single Side	4,7	8,5
Double Side	9,4	17

onsschicht geprägt ist, der Halbreflektor (Reflexionsvermögen 20-40 Prozent) aufgetragen wird. Durch eine weitere Stempelprägung entsteht in der danach aufgetragenen Photopolymerschicht die zweite Speicherschicht.

Wieder beschreibbare DVD

Es gibt verschiedene Typen wieder beschreibbarer DVDs. Die *DVD-R* benutzt einen Schreib-Laser zum Brennen der Information in ein organisches, gefärbtes Material. DVD-Rs können nur einmal beschrieben werden (analog zur CD-R). Im Gegensatz dazu kann eine *DVD-RW* etwa 1000mal neu beschrieben werden, da dort ein Phase-Change-Verfahren zum Einsatz kommt (analog zur CD-RW). Ein alternatives Verfahren ist *DVD+RW*, das eine ähnliche Technik benutzt.

Die *DVD-RAM* benutzt eine Mischung zwischen einem Phase-Change-Verfahren und dem bei MODs verwendeten Technik. Die Medien können mehr als 100000mal beschrieben werden und sind speziell für den Einsatz als Datenträger konzipiert. Allerdings sind die Medien derzeit normalerweise nicht mit DVD- oder DVD-R(W)-Laufwerken les- oder schreibbar. Alle Medien bieten die Kapazität einer Single-Layer, Single-Side DVD, also 4,7 GByte.

Blu-ray Disc

Im Februar 2002 wurde die Spezifikation der *Blu-ray Disc* verabschiedet. Im Gegensatz zur CD und DVD arbeitet die Blu-ray Disc mit einem blauen Laser. Blaues bzw. blau-violette Laserlicht hat eine deutlich kleinere Wellenlänge (405 nm) gegenüber rotem Laserlicht (630 nm). Dadurch kann der Abstand der Pits verringert und die Datendichte deutlich erhöht werden. So ist bei gleicher Mediengröße (12 cm) eine Speicherkapazität von 27 GByte (single Side, single Layer) bzw. 50 GByte (single Side, double Layer) möglich. Da die Blu-ray Disc zudem MPEG-2 unterstützt, können so bis zu 13 Stunden Film gespeichert werden.

Ebenso wie bei der DVD können sich die Herstellerfirmen auch auch bei diesem Format nicht auf einen Standard einigen. So existiert neben dem Blu-Ray-Konsortium (u.a. mit Sony und Philips) eine Gruppe unter Führung von Toshiba und NEC, die für die HD-DVD (früher Advanced Optical Disc-AOD) wirbt, die auf der gleichen Technik basiert. Da die blauen Laserdioden nicht die Lebensdauer von roten Laserdioden erreichen (ca. 1/5) und daher die Produktionsausbeuten von blauen Laserdioden noch sehr niedrig sind, liegt der Preis der momentan verfügbaren Laufwerke im Vergleich zu DVD- bzw. CD-Brennern sehr hoch.

Die Abbildung 11.17 zeigt den Trend der Entwicklung bei der Speicherdichte von CD-ROM, DVD und Blu-ray Produkten bei Einsatz unterschiedlicher Lasertechnologien. Die gestrichelte Linie gibt die Kapazität bei Verwendung von Dual-Layer Technologie wieder.

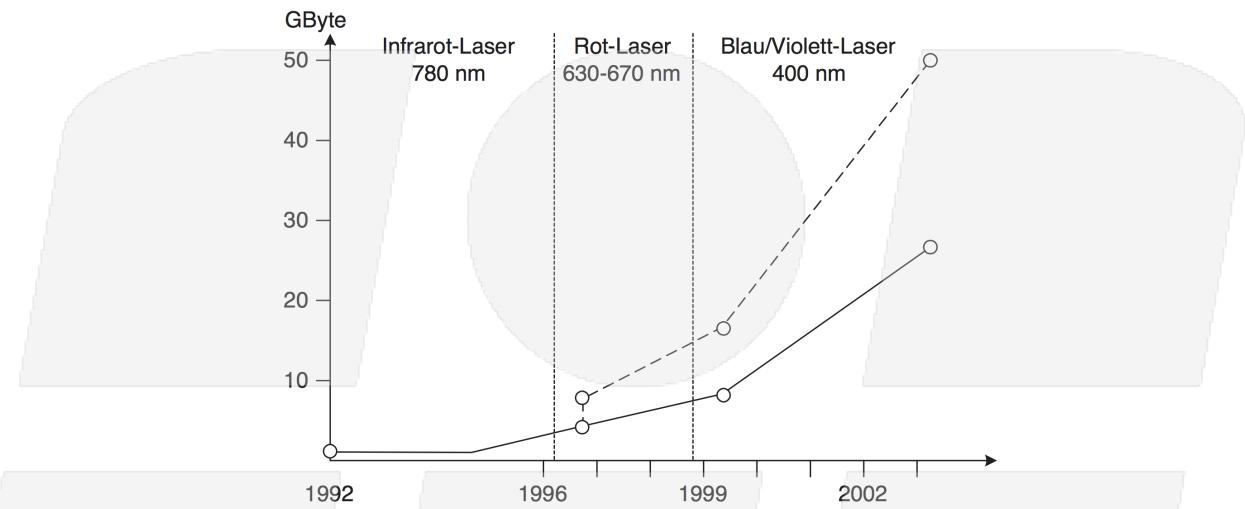


Abbildung 11.17: Entwicklung der Speicherkapazität bei optischen Massenspeichern (CD-ROM bis 1996, DVD bis 2003 und danach Blu-ray) bei Verwendung unterschiedlicher Lasertechnologien (in GByte).

11.5 Magneto-optische Massenspeicher

Rewritable Optical Disk

Die Technologie der *MOD (Magneto-Optical Disc)* nutzt sowohl Bestandteile der magnetischen als auch der optischen Speichertechnologie. Das Medium kann bis zu 1 Million mal beschrieben und wieder gelöscht werden und hat damit gegenüber anderen Plattspeichertechnologien momentan die längste Lebensdauer. Die z.Zt. angebotenen MO-Disks haben eine Kapazität von 2.4 GByte (5,25"), 640 MByte oder 256 MByte (3,5").

Beim Schreibvorgang wird die Platten-Magnetbeschichtung punktuell mit Laserstrahlen des Schreib-/Lesekopfes auf 150 Grad Celsius erhitzt. Dadurch kann die Polarisierung der Beschichtung entsprechend des vom Schreib-/Lesekopf angelegten Magnetfeldes umgekehrt werden. Die Polarisierung wird beim Lesevorgang durch die unterschiedlichen Reflektionen des Laserstrahls wieder festgestellt.

11.6 Speicherorganisation

Die unterschiedlichen Speichertechniken werden in drei Kategorien eingeordnet: den Primär-, Sekundär- und Tertiärspeicher. Sie unterscheiden sich insbesondere in der Art der Zugriffsmöglichkeiten (wahlfrei, sequentiell, zyklisch) und in der Zugriffsgeschwindigkeit (Abbildung 11.19). Zudem sind sie — wegen Preisunterschieden — unterschiedlich groß. Eine geringere Größe von Primär- und Sekundärspeicher ist jedoch oft ausreichend, da sowohl der Datenzugriff als auch die Befehlsfolgen eines Programms nicht über den gesamten Adressraum verstreut auftreten, sondern sich meist nur wenig ändern, d. h. sie bewegen sich innerhalb einer gewissen Lokalität. Die '80/20-Regel' besagt, dass in 80% der Fälle lediglich 20% der Daten benötigt werden.

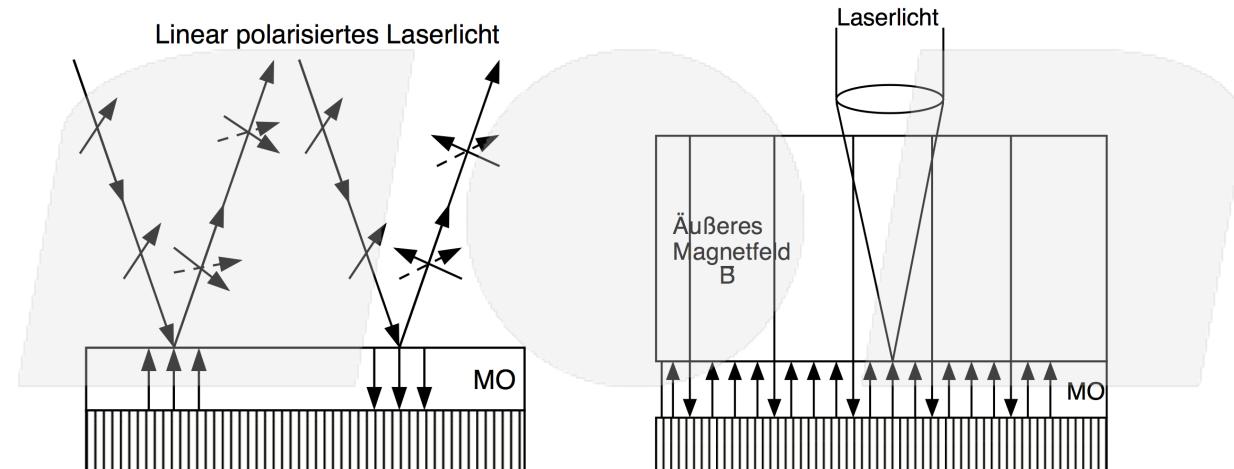


Abbildung 11.18: Lesen (li.) und Schreiben (re.) eines magnetooptischen Speichers

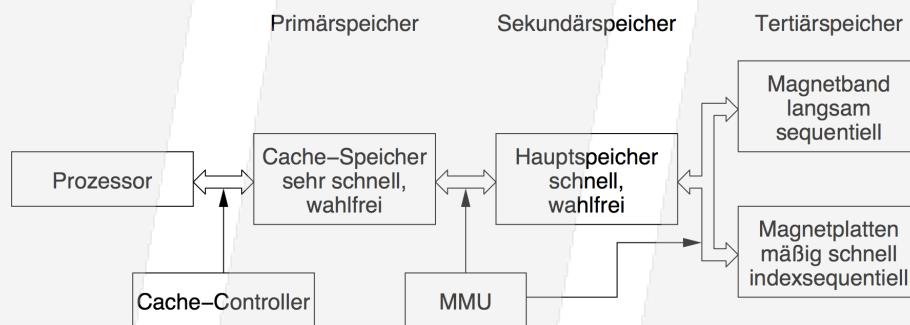


Abbildung 11.19: Hierarchische Speicherverwaltung

11.6.1 Cache-Speicher

Da sich die Adressierung beim Programmablauf innerhalb eines begrenzten Adressraumes bewegt, können eine Reihe von Operationen und Speicherzugriffen in dem kleineren und schnelleren Puffer-/Cache-Speicher ablaufen, ohne auf den langsameren Hauptspeicher zugreifen zu müssen (Abbildung 11.20). *Cache-Speicher* gibt es separat für Daten und Befehle (Harvard-Architektur) oder auch kombiniert (Unified Cache). Heute ist es üblich, eine zweistufige Cache-Architektur zu verwenden. Der First-Level Cache ist oftmals als getrennter Programm/Daten-Cache realisiert, während der Second-Level Cache als Unified Cache Programm und Daten im selben Speicherbereich aufnimmt.

Befinden sich die Daten bzw. Befehle nicht im Cache, so müssen sie aus dem Hauptspeicher nachgeladen werden. Die Trefferrate (Cache-Hitrate) hängt dabei von der Größe des Caches und der Strategie ab, mit der die im Cache bereit gehaltenen Auszüge des gesamten Adressraums ausgewählt werden.

Der First Level Cache liegt grundsätzlich am CPU Bus, für den Second Level Cache gibt es unterschiedliche Topologien, die in Abbildung 11.20 dargestellt werden.

Look-aside-Cache Cache und Hauptspeicher liegen am selben Bus. Der Cache wird im Gegensatz zum Hauptspeicher ohne Wartezyklen angesprochen. Sein Durchsatz wird durch

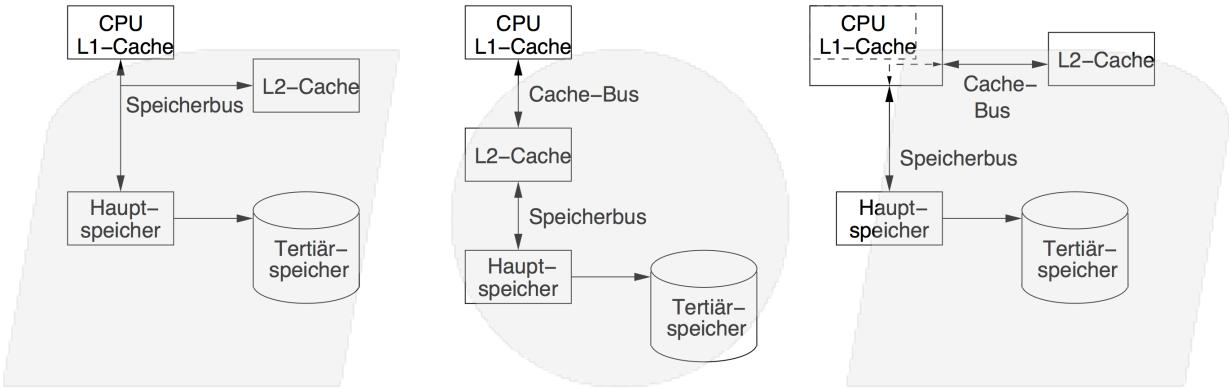


Abbildung 11.20: Cache-Topologien: Look-aside-Cache (links), Inline-Cache (mitte), Backside-Cache (rechts)

den Bustakt bestimmt. Diese Cache-Topologie findet sich auf den meisten IBM-PC kompatiblen Boards

Inline-Cache Der von Apple bei den PowerPC601 basierten Systemen verwendete Inline-Cache ist dem Hauptspeicher (aus Sicht der CPU) vorgeschaltet und von dessen Bustakt entkoppelt. Er kann daher mit einem Vielfachen der Bustaktfrequenz des Hauptspeichers betrieben werden. Um den L1-Cache mit neuen Daten zu versorgen, müssen diese zuerst in den L2-Cache geladen werden.

Backside-Cache Die PowerPC-Prozessoren der G3-Serie sowie der PentiumPro und PentiumII verwenden einen durch dedizierte Anschlüsse komplett vom Systembus getrennten Cache. Der Cache-Controller ist dabei auf dem Prozessor integriert. Diese schaltungstechnisch aufwändige Topologie bietet das größte Datendurchsatzpotenzial, da der Cache-Bus mit prinzipiell jeder Taktfrequenz betrieben werden kann, bis hinauf zur Taktfrequenz der CPU. Lädt der Prozessor eine neue Seite vom Hauptspeicher, so wird gleichzeitig der L1-Cache und der L2-Cache mit den neuen Daten versorgt (Abb. 11.20, rechte Seite).

Cache-Organisation

Ein Cache ist prinzipiell wie in Abbildung 11.21 aufgebaut: Er besitzt einen Tag-Bereich (*Tag-RAM*) und einen Datenbereich. Der Datenbereich wiederum besteht aus einer Menge von C Einträgen, den so genannten *Cache-Lines*. Diese Cache-Lines haben immer eine feste Größe von M Bytes, die *Line Size*. Es können immer nur komplette Cache Lines aus dem oder in den Hauptspeicher transferiert werden. Jeder Cache Line ist ein so genanntes *Tag* zugeordnet, das Informationen darüber beinhaltet, wo sich der zugehörige Block im Hauptspeicher befindet. Hat der Hauptspeicher die Größe S Bytes, so wird er für die Cache-Verwaltung in $B = S/M$ Blöcke eingeteilt. Ist eine Adresse k im Hauptspeicher gegeben, so ist die Nummer des dazugehörigen Blocks $b = \lfloor k/M \rfloor$. Die Art der in den Tags gespeicherten Informationen hängt dabei von der Organisation des Caches, also der Zuordnung von Speicherblöcken zu Cache-Lines, ab:

Vollständig assoziativ: Bei einem vollständig assoziativen Cache enthält das Tag die Nummer der Cache-Line im Hauptspeicher. Ein Tag hat also $\log_2 B$ Bits. Damit kann jeder Hauptspeicherblock auf jeden beliebigen Cache-Eintrag abgebildet werden. Ein Cache-Hit wird

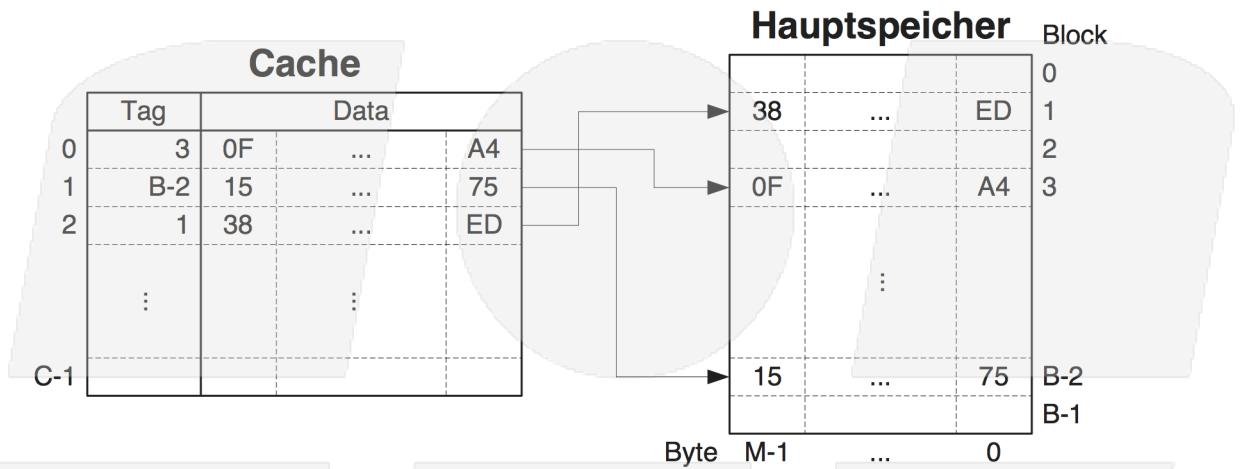


Abbildung 11.21: Struktur eines (vollständig assoziativen) Caches

durch einen parallel durchgeführten Vergleich der oberen $\log_2 B$ Bits der Adresse mit allen Tags entdeckt (Assoziativspeicher). Diese Organisationsform ist die leistungsfähigste, allerdings ist die benötigte Vergleichslogik aufwändig und daher teuer.

Direct mapping: Dies stellt eine Alternative zum Assoziativspeicher bei der Cache-Verwaltung dar. Dabei wird der Hauptspeicher in aufeinanderfolgende, gleich große Segmente eingeteilt. Die Segmentgröße entspricht der Größe C des Caches. Jeder Eintrag im Cache entspricht einem festen Block innerhalb eines Segments. Das Tag-RAM enthält lediglich die Nummer des Segments und entspricht den oberen $\log_2(B/C)$ Bits einer Adressen innerhalb des Speicherblocks. Der Schaltungsaufwand dafür ist gering, allerdings kann es vorkommen, dass sich zwei Blöcke im Wechsel verdrängen. Daher sind Direct Mapping Caches nicht so leistungsfähig wie assoziative Caches.

n-fach assoziativ: Ein n-fach assoziativer Cache besteht im Prinzip aus n parallelen direct mapped Caches. Das Tag-RAM sieht genauso aus wie beim direct mapped Cache. Beim Zugriff werden die n Tag-Einträge mit dem entsprechenden Teil der Adresse verglichen, um festzustellen, in welchem der n Teil-Caches sich die korrekte Cache-Line befindet. Dieser Cache-Typ verbindet Leistungsfähigkeit mit geringen Kosten, so dass er heute die am meisten verbreitete Organisationsform darstellt.

Schreibstrategien

Eine weitere Unterscheidungsmöglichkeit ist die Strategie, nach der veränderte Einträge im Cache in den Hauptspeicher geschrieben werden. Beim *Write-Through* Cache werden Daten, die verändert wurden sofort in den Hauptspeicher zurückgeschrieben, wodurch sich die Performance des Caches verringert. Die *Write-Back* Strategie schreibt die Cache-Line erst dann zurück, wenn sie verdrängt wird, oder beispielsweise ein Peripheriegerät einen Wert aus dem entsprechenden Block des Hauptspeichers per DMA anfordert. Dazu wird ein zusätzliches Bit pro Eintrag im Tag-RAM benötigt, das anzeigt, ob die Daten im Cache verändert wurden („Dirty-Bit“).

Bei einem DMA-Zugriff auf einen Block im Hauptspeicher wird anhand dieses Bits zuerst überprüft, ob der entsprechende Eintrag im Cache verändert worden ist. Stimmen Inhalt im Cache

und im Hauptspeicher nicht überein, wird die Cache-Line erst vom Cache in den Hauptspeicher zurückgeschrieben ('write-back'), bevor der DMA Zugriff erfolgen darf. Der Schaltungsaufwand ist gegenüber dem Write-Through Verfahren höher.

11.6.2 Virtueller Speicher

Von *virtuellem Speicher* spricht man, wenn die Festplatte als Hauptspeicherersatz fungiert. Einer Software wird dazu ein größerer Hauptspeicher vorgetäuscht als physikalisch vorhanden ist. Damit wird Software, die einen größeren Adressenraum beansprucht als physikalisch durch Hauptspeicher abgedeckt werden kann, lauffähig. Aktuelle Datenanforderungen werden durch Nachladen von Daten von der Festplatte in den Hauptspeicher befriedigt. Gegebenenfalls wird der dafür notwendige Speicherplatz durch Auslagern nicht benötigter Daten auf die Festplatte freigemacht.

Man unterscheidet dabei zwischen *Swapping* und *Paging*. Ersteres bezeichnet das Auslagern von entweder kompletten Speichersegmenten oder dem gesamten Speicherbereich eines Prozesses auf die Festplatte. Dieser Vorgang ist zeitaufwändig und wird heute kaum noch verwendet. Wesentlich effizienter ist die zweite Methode. Dabei wird der Speicher in so genannte Seiten oder Pages fester Größe eingeteilt (meist zwischen 1 KByte und 16 KByte). Beim Start eines Programms werden nur die gerade benötigten Seiten geladen. Wird der zur Verfügung stehende Speicher knapp, so werden, ebenfalls seitenweise, nach bestimmten Ersetzungsstrategien Seiten vom Hauptspeicher auf die Festplatte geschrieben.

Eine optimale Strategie hat eine minimalen Zahl von Seitenwechseln. Um Platz für die einzuladende Seite zu erhalten wird hierbei diejenige Seite entfernen, die in der Zukunft am längsten nicht mehr benötigt wird (LFD, Longest-Forward-Distance). Da eine exakte Voraussage der Zukunft unmöglich ist, orientiert man sich an dem bisherigen Adressen-/Seitenzugriff (in einem unmittelbar vorangegangenen Zeitbereich). Zwei weitere Seitenwechsel-Strategien sind z. B. FIFO (First In First Out) und LRU (Least Recently Used).

Bei Verwendung von virtuellem Speicher ist zwischen logischen Adressen und physikalischen Adressen zu unterscheiden. Erstere sind die Adressen, die von Programmen verwendet werden können. Letztere sind die Adressen des tatsächlich vorhandenen RAMs. Die Umsetzung zwischen beiden Typen erfolgt durch die *Memory Management Unit (MMU)*. Dabei wird eine Adresse zunächst in Seitennummer und Offset aufgeteilt. Zur Zuordnung von logischer zu physikalischer Seitennummer wird ein *Translation-Lookaside Buffer (TLB)* verwendet, der ähnlich wie ein vollständig assoziativer Cache (Abschnitt 11.6.1) organisiert ist: Zu einer logischen Seitennummer (\triangleq Tag) wird die physikalische Seitennummer (\triangleq Cache-Line) gespeichert.¹ Wird eine Seite nicht im TLB gefunden, so wird ein Page Fault ausgelöst, der vom Betriebssystem abgefangen werden muss. Dieses lädt die benötigte Seite in den Hauptspeicher, wofür eine in Software implementierte Hash-Tabelle benutzt wird. Diese ordnet einer Seitennummer einen Bereich auf der Festplatte zu.

11.6.3 Interleaving

Die Zykluszeiten von Hauptspeicher und Prozessoren stehen selbst bei Verwendung sehr schneller Komponenten im Verhältnis 4:1 bis 12:1. Wartezyklen des Prozessors können durch

¹Dies ist eine vereinfachte Darstellung, in der Realität ist es leider viel komplizierter; mehr Details stehen z. B. in [Ma01].

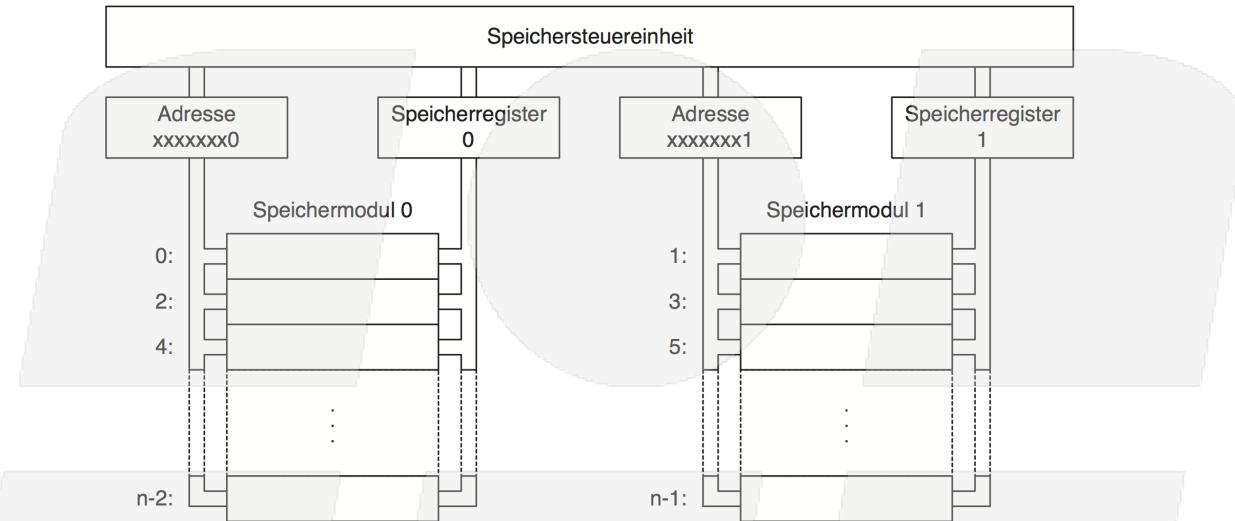


Abbildung 11.22: Speichermodulverschränkung (Interleaving)

die Verteilung aufeinanderfolgender Daten auf mehrere Speichermodule vermieden werden. Durch diese *Speichermodulverschränkung (Interleaving)* ist ein Zugriff auf mehrere Worte möglich, ohne mehrfach auf ein Speichermodul zuzugreifen und dafür die Zykluszeit des verwendeten Speicherbausteins abzuwarten. Bei Großrechnern werden bis zu 16 Module verschärkt (z. B. BASF/Hitachi 7/90 Prozessor). Abbildung 11.22 zeigt die parallele Speicherorganisation mit zwei Speicherblöcken.



Kapitel 12

Rechneraufbau am konkreten Beispiel

Die Prozessoren verschiedener Hersteller sind zumeist in verschiedene Prozessor-Familien eingeteilt. Die wesentlichen Vertreter sind dabei:

- Pentium, Hersteller: Intel Corporation
- Athlon, Hersteller: Advanced Micro Devices
- PowerPC, Hersteller: Motorola, IBM
- SPARC, Hersteller: Sun Microsystems
- MIPS, Hersteller: Silicon Graphics
- PA-RISC, Hersteller: Hewlett-Packard
- Alpha, Hersteller: Digital Equipment Corporation

Zwei Prozessor-Familien werden im Folgenden näher betrachtet.

12.1 Pentium-Familie

12.1.1 Übersicht

Die Pentium-Familie entstand 1993 als Fortsetzung der 486er Prozessoren mit dem Ziel, eine Architektur zu schaffen, die zum einen die Kompatibilität zum x86 Befehlssatz besitzt, zum anderen aber für die Ausführung von 32 bit Applikationen besser geeignet ist. Die Pentium-Familie stellt keine reine RISC oder CISC Architektur dar, da nur wenige Befehle vereinfacht wurden.

Ausgehend vom Pentium (P5), entstand 1995 der PentiumPro, der neben Optimierungen in der Prozessorarchitektur, einen mit vollem CPU-Takt betriebenen Backside-Cache (s. Kap. 11.6) auf dem Chip enthält, wodurch sich die Performance gegenüber dem Pentium wesentlich steigern ließ.

Im Zuge der Verbreitung von Multimedia und grafikintensiven Applikationen wurde der Pentium-Befehlssatz um Multimedia-Befehle erweitert, die die Fließkommaeinheit des Prozessors mit benutzen, aber nur mit Ganzzahlen arbeiten (MMX-Befehlssatz). Die Benutzung dieser Befehle setzt aber deren Programmierung in Assembler voraus, da Hochsprachencompiler diese nicht direkt unterstützen.

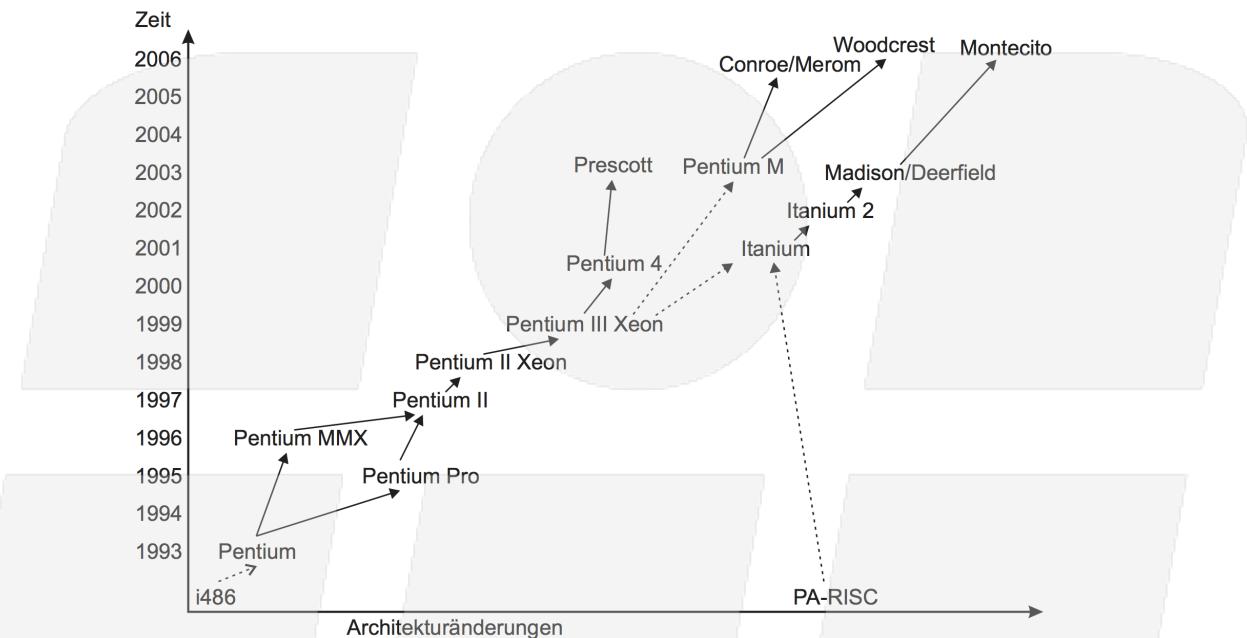


Abbildung 12.1: Übersicht über die Pentium- und Itanium-Familien

Als Verbindung der PentiumPro und der MMX Technologie entstand 1997 der Pentium II. Trotz höherer Taktfrequenz als der Pentium Pro ist dessen Leistung in einem ähnlichen Bereich angesiedelt, da sich hier der Cache nicht mehr auf dem Chip befindet und mit dem halben CPU Takt betrieben wird. Die Weiterentwicklung des Pentium II stellt der 1999 vorgestellte Pentium III dar, der als erstes Mitglied der Pentium-Familie eine SIMD-Einheit für Gleitkommazahlen besaß und in Kupfertechnologie mit einer Strukturbreite von $0.18\mu\text{m}$ gefertigt wurde.

Im Jahr 2000 erschien der Pentium 4, der zahlreiche Verbesserungen zum Pentium III mit sich brachte. Dazu zählte eine verbesserte Mikroarchitektur sowie das Hyper-Threading (s. Kap. 12.1.2). Mit dem 2001 erstmals vorgestellten *Itanium* bietet Intel auch endlich einen 64-Bit Prozessor an. Dieser Prozessor basiert auf der PA-RISC Architektur von Hewlett-Packard aus den 80'er Jahren und wurde von beiden Firmen gemeinsam entwickelt. 32-Bit-Prozessoren konnten aufgrund der 32 Bit Adressbreite lediglich einen Adressraum von 4 GByte ansprechen. Mit der Entwicklung der 64-Bit Prozessoren fällt diese Schranke, rein rechnerisch ist nun ein Adressraum von 18 Milliarden GByte ansprechbar. Problematisch ist dabei die Kompatibilität von Programm, die für 32-Bit Prozessoren entwickelt wurden, da sich z. B. Probleme mit der Länge von Variablen ergeben. Alle aktuellen 64-Bit Desktop-Prozessoren sind daher 32-Bit kompatibel.

Die Weiterentwicklung des Itanium2 stellt der Madison dar (Servervariante Deerfield), der 2006 von dem Montecito abgelöst wurde.

2003 erschien der Pentium M, welcher eine Weiterentwicklung des Pentium III darstellte. Durch die Abkehr von der Netburst-Mikroarchitektur (s. Kap. 12.1.2) erreicht die PentiumM-Familie höhere Leistung mit deutlich geringeren Taktraten. Mit der Einführung des *Woodcrest* für Server und *Conroe/Merom* im Desktop- und Mobilbereich stellte Intel die neue *Core*-Mikroarchitektur vor.

12.1.2 Beispiel: Pentium 4

Mit dem Pentium 4 führte Intel die so genannte Netburst-Mikroarchitektur ein, die die P6-Mikroarchitektur vorangegangener Prozessoren (PentiumPro bis Pentium III) ablöst. Sie wurde mit dem Ziel entwickelt, höchste Taktraten zu ermöglichen, eine besonders schnelle Ausführung einfacher Prozessorbefehle zu gewährleisten und die Kapazität der Execution Units so gut wie möglich auszunutzen.

Ein Blockschaltbild der *Netburst-Architektur* zeigt Abb. 12.2. Der Prozessor ist über das *Bus Interface Unit* an den System Bus angeschlossen. Dieses Interface hat eine Breite von 64 bit und je nach Prozessorversion eine effektive Taktrate von 400, 533 oder 800 MHz, wodurch Datenraten von 3.2, 4.2 oder 6.4 GByte/s erreicht werden. Die tatsächliche Taktrate beträgt nur ein Viertel der genannten, da die Datenübertragung „quad-pumped“ abläuft, wodurch in einem Takschritt die vierfache Datenmenge übertragen werden kann.

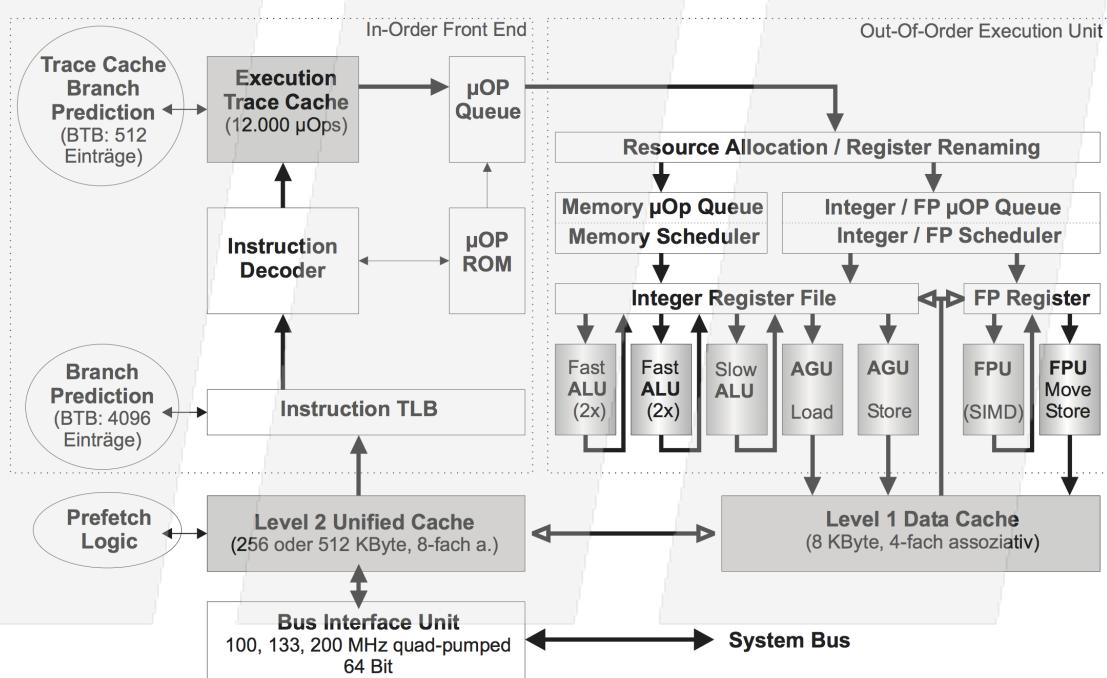


Abbildung 12.2: Blockschaltbild des Pentium 4

Der Pentium 4 besitzt eine zweifache Cache-Topologie. Der als Backside-Cache realisierte *Level 2 Cache* hat eine Größe von 256 oder 512 KByte und enthält sowohl Daten als auch Befehle (Unified Cache, siehe Kap. 11.6). Er ist 8-fach assoziativ, die Latenzzeit beim Lesen beträgt 7 Takte. Die Cache-Lines besitzen eine Länge von 128 Byte in zwei Hälften, d.h. es werden immer mindestens 64 Byte zusammenhängend aus dem Hauptspeicher gelesen. Zur Verminderung von Cache Misses ist dem L2-Cache eine *Hardware Prefetch Logik* zugeordnet, die durch Erkennen von Daten-Zugriffsmustern und Beobachten vorangegangener Cache-Misses versucht, den tatsächlichen Cache-Anforderungen um 256 Byte voraus zu sein. Bis zum Pentium III konnte solch ein Daten-Prefetch nur über die Software erreicht werden.

Enthielt der Pentium III noch jeweils einen 16 KByte Level 1 Daten- und einen genauso großen Befehlscache (Harvard-Architektur), so geht Intel beim Pentium 4 einen etwas anderen Weg: Der *Level 1 Daten Cache* ist hier nur noch 8 KByte groß, 4-fach assoziativ und besitzt Cache-Lines von 64 Byte Länge. Vorteil: Mit nur 2 Takten Latenzzeit (Lesen von Integer-Daten, 6

Takte zum Lesen von Fließkomma-Daten) ist er extrem schnell. Der L1-Daten-Cache ist über eine 256 Bit breite Verbindung mit Prozessor-Taktfrequenz mit dem L2-Cache verbunden, was eine Datenrate von z.B. 48 GByte/s bei 1,5 GHz oder 102 GByte/s bei 3,2 GHz ermöglicht. Er ist ein „write-through“ Cache, d.h. Schreibbefehle werden immer auch auf dem L2-Cache ausgeführt.

Anstelle eines L1-Befehls-Cache für IA-32 Befehle besitzt der Pentium 4 einen sogenannten *Execution Trace Cache* für 12.000 Mikrooperationen (μ Ops). Dabei handelt es sich um Befehls-worte fester Länge, die im Gegensatz zu den oft komplexen und in der Länge variablen IA-32 Maschinenbefehlen einfacher von der Execution Unit verarbeitet werden können. Die MaschinenSprache im Prozessor stellt somit selbst noch eine „Hochsprache“ dar, die in einfachere Untereinheiten übersetzt wird. Dieses geschieht im *Instruction Decoder*, wobei die meisten IA-32 Befehle in ein bis drei μ Ops zerlegt werden, komplexere Befehle wie z.B. zum Kopieren von Strings aber auch aus mehreren tausend μ Ops bestehen können. Die Mikrocode-Befehlsfolgen sind im *Mikrocode ROM* gespeichert. Um Speicherplatz im Trace Cache zu sparen, werden dort bei langen Befehlsketten ($> 4 \mu$ Ops) nur die Pointer zu dem entsprechenden Eintrag im Mikrocode ROM abgelegt. Die Positionierung des Befehls-Cache nach der Dekodierungs-Einheit verringert Dekodierungs-Engpässe bei komplexen Befehlen oder fehlerhaften Sprungvorhersagen. Zudem müssen bei Schleifen nicht mehr wiederholt die gleichen Befehle dekodiert werden, da sie hier bereits fertig dekodiert im Trace Cache liegen. Dieser Ansatz ermöglicht auch eine Sprungvorhersage auf μ Op-Ebene, wozu ein Branch Target Buffer (BTB) mit 512 Einträgen zur Verfügung steht. Die IA-32 Befehle gelangen in den Instruction Decoder aus dem L2-Cache über den *Instruction Translation Look-aside Buffer* (ITLB), der virtuelle Instruction Pointer Adressen in physikalische Adressen des L2-Cache übersetzt. Hier kommt außerdem eine aus-gfeilte Sprungvorhersage-Logik auf Programmebene zum Einsatz, die über einen mit 4096 Einträgen achtmal so großen BTB wie der Pentium III verfügt, wodurch laut Intel fehlerhafte Vorhersagen um ein Drittel gesenkt werden konnten.

Zur weiteren Verarbeitung gelangen bis zu drei Mikrooperationen pro Takt schritt über eine Queue (kleiner FIFO-Buffer zur Stabilisierung) in die *out-of-order Execution Unit*. Die out-of-order Ausführung von Befehlen gewährleistet, dass die ALUs so intensiv wie möglich genutzt werden. Zunächst werden für die eintreffenden μ Ops freie Ressourcen in den verschiedenen benötigten Buffern und Tabellen lokalisiert (*Resource Allocator*), sowie die verwendeten logischen Prozessor-Register auf die physikalischen 128 Integer- und 128 Fließkomma-Register abgebildet (*Register Renaming*). Die μ OPs werden dann in Queues noch in Programmreihenfolge abgelegt und daraus von vier verschiedenen *Schedulern* je nach Verfügbarkeit der Recheneinheiten sowie der Eingangsvariablen out-of-order ausgewählt und verarbeitet. Die Scheduler sind nach Befehlsklassen aufgeteilt: 1. Memory Scheduler, 2. Fast ALU Scheduler, 3. Slow ALU / General FP Scheduler und 4. Simple FP Scheduler.

Der Pentium 4 besitzt folgende Execution Units: Zwei AGUs (Address Generation Units) arbeiten im Prozessortakt und sind für Lese- und Schreibbefehle im L1-Cache zuständig. Beim Pentium 4 hat sich Intel ganz besonders auf die extrem schnelle Ausführung einfacher Integer-Befehle (wie ADD, SUB, AND, OR ...) konzentriert, da diese meist den Großteil eines Programms ausmachen. So finden sich zwei parallele *Fast ALUs*, die zudem „double-pumped“ arbeiten, also zwei Befehle pro Takt schritt ausführen können. Alle übrigen, komplexeren Integer-Befehle (MULT, DIV, SHIFT ...) werden von der *Slow ALU* verarbeitet. Für Fließkommazahlen gibt es zwei *FPU*s (Floating Point Units), wovon eine lediglich dem Datenzugriff dient. Um hier die Leistungsfähigkeit zu steigern, hat Intel die *SIMD-Befehle* (Single Instruction, Multiple

Data) vorangegangener Prozessoren um 144 Instruktionen erweitert (*SSE2 - Streaming SIMD Extensions 2*). Diese können mehrere Daten parallel verarbeiten, die in den acht 128-Bit XMM-Registern abgelegt werden. Dabei sind folgende Kombinationen möglich: 4 einfachgenaue oder 2 doppeltgenaue Fließkommazahlen, 16 Byte-Werte, 8 Word-Werte, 4 Double-Word-Werte, 2 Quad-Word-Werte oder ein 128-Bit Integer Wert. Diese Befehle ermöglichen eine erhebliche Leistungssteigerung von Programmen, werden aber bisher nur von wenigen Hochsprachencompilern wie dem Intel C++ Compiler unterstützt.

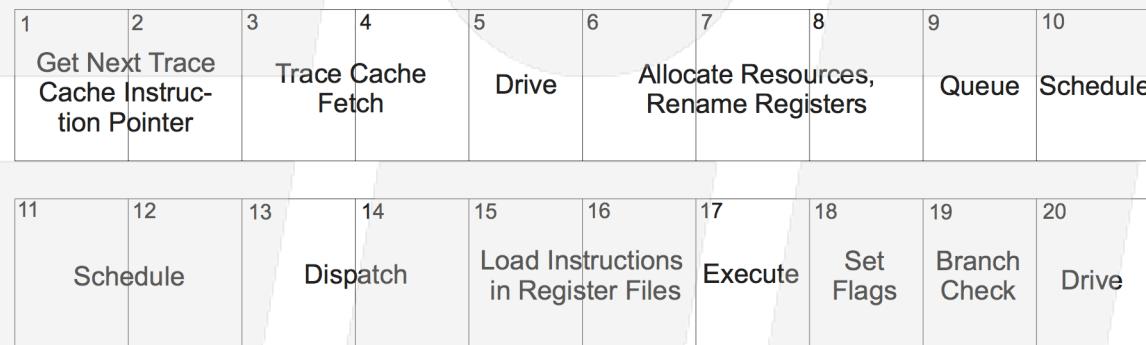


Abbildung 12.3: Execution Pipeline des Pentium 4

Zum Erreichen der extrem hohen Taktraten des Pentium 4 musste die bisher 10-stufige Execution Pipeline (z.B. Pentium III) weiter unterteilt werden und besteht nun aus 20 Stufen (siehe Abb. 12.3). Interessant ist hierbei, dass erstmals auch Pipeline-Stufen für die Laufzeit eines Signals bis zum nächsten Prozessorteil berücksichtigt werden mussten (*Drive*).

Ein neues Feature, dass einige Versionen des Pentium 4 bieten, ist die Hyper Threading *Technologie*(HT). Wie bei der out-of-order Ausführung ist auch hier das Ziel, die vorhandenen Ressourcen möglichst intensiv zu nutzen. Ein Pentium 4 Prozessor mit HT verhält sich wie zwei parallele Prozessoren und wird auch so von einem Betriebssystem gesehen, das HT unterstützt. Dadurch können zwei Threads (Programmteile) gleichzeitig auf einem Prozessor laufen, wobei natürlich nicht die Geschwindigkeit von zwei physikalischen Prozessoren erreicht wird. Technisch müssen hierfür nur wenige Teile des Prozessors doppelt vorhanden sein, die den Prozessorstatus beschreiben, sowie eine erweiterte Auswahllogik für die wartenden Befehle. Ein Pentium 4 Prozessor mit HT ist lediglich 5% größer als einer ohne diese Eigenschaft.

12.2 PowerPC-Familie

12.2.1 Übersicht

Anders als Intel, die mit der Pentium Familie die Kompatibilität zur alten x86-Architektur bewahren wollten, entstand der PowerPC aus einer Kooperation zwischen Apple, Motorola und IBM mit dem Ziel, eine RISC-Architektur zu schaffen, mit der sich ein breites Leistungsspektrum vom Embedded Controller (in industriellen Anwendungen) bis hin zu Höchstleistungsrechenanlagen (Supercomputer) abdecken lässt.

Der PowerPC-Architektur liegt die von IBM entwickelte Power 2 Architektur zugrunde, bei der die im PowerPC integrierten Einheiten auf mehrere Chips verteilt waren (Multichip Module).

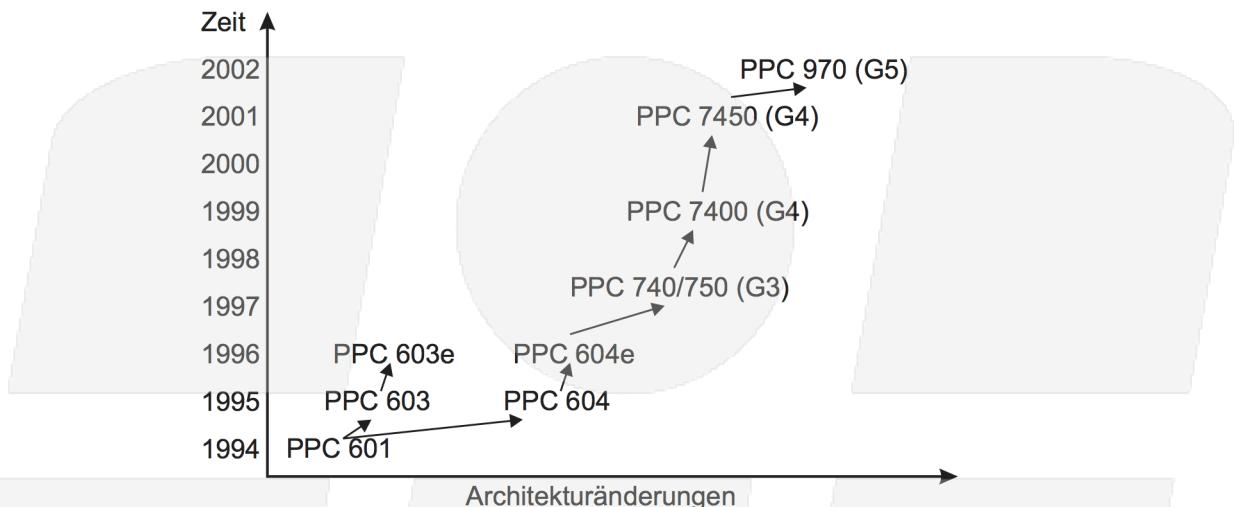


Abbildung 12.4: Übersicht über die PowerPC Familie

Erstes Mitglied dieser Familie war der PowerPC 601, aus dem der PowerPC 603 als energiesparende Version mit verbesserter Integer-Einheit hervorging. Der gleichzeitig entstandene PowerPC 604 besitzt eine höhere Fließkommaleistung und mehr parallel arbeitende Integer-Einheiten. Die 'e'-Versionen dieser Chips besitzen einen größeren Cache (2*32 KByte) und können mit höheren Taktfrequenzen (bis 350 MHz) betrieben werden.

Die 1997 vorgestellten Prozessoren PowerPC 740 und PowerPC 750 stellen die ersten Mitglieder der G3-Serie dar (PowerPC 601: G1, PowerPC 603(e)/4(e): G2). 1999 wurde das erste Mitglied der G4-Serie vorgestellt, der PowerPC 7400. Er zeichnet sich durch eine SIMD-Einheit aus (AltiVec) sowie einige interne Verbesserungen gegenüber dem PowerPC 750.

Im Oktober 2002 erschien mit dem PowerPC 970 die fünfte Prozessor-Generation. Der PPC 970 ist der erste 64-Bit-Prozessor, der abwärtskompatibel zu bestehenden 32-Bit-Anwendungen bleibt.

Daneben existieren noch Mikrocontroller auf Basis des PowerPC Kerns (ohne FPU), bei denen neben der CPU weitere Funktionalitäten, wie z. B. eine serielle Schnittstelle oder ein A/D-Wandler, auf dem Chip integriert sind.

12.2.2 Beispiel: PowerPC 7400

Der *PowerPC 7400* unterstützt einen bis zu 2 MByte großen zweifach assoziativen Unified Backside L2-Cache. Der Cache befindet sich, anders als beim Pentium III, nicht auf dem Prozessormodul. Lediglich das (wesentlich kleinere) Tag-RAM ist aus Geschwindigkeitsgründen mit integriert. Dadurch sinken die Herstellungskosten und der Cache lässt sich flexibler konfigurieren.

Die Level 1 Caches sind je 32 KByte groß und 8-fach assoziativ. Von dort gelangen die Befehle und Daten in die einzelnen Ausführungseinheiten. Als reiner RISC-Prozessor besitzt der PowerPC je einen Satz von 32 General Purpose Registern (GPR, 32 bit) und 32 Floating Point Registern (FPR, 64 bit IEEE-754 einfache und doppelte Genauigkeit). Weiterhin besitzt er 32 Vektor-Register (VR, 128 bit).

Im Blockschaltbild (Abb. 12.5) ist zu erkennen, dass der PowerPC, da ein reiner RISC Prozessor, keine μ Ops wie der Pentium III generieren muss. Zusätzlich enthält er zwei parallel arbei-

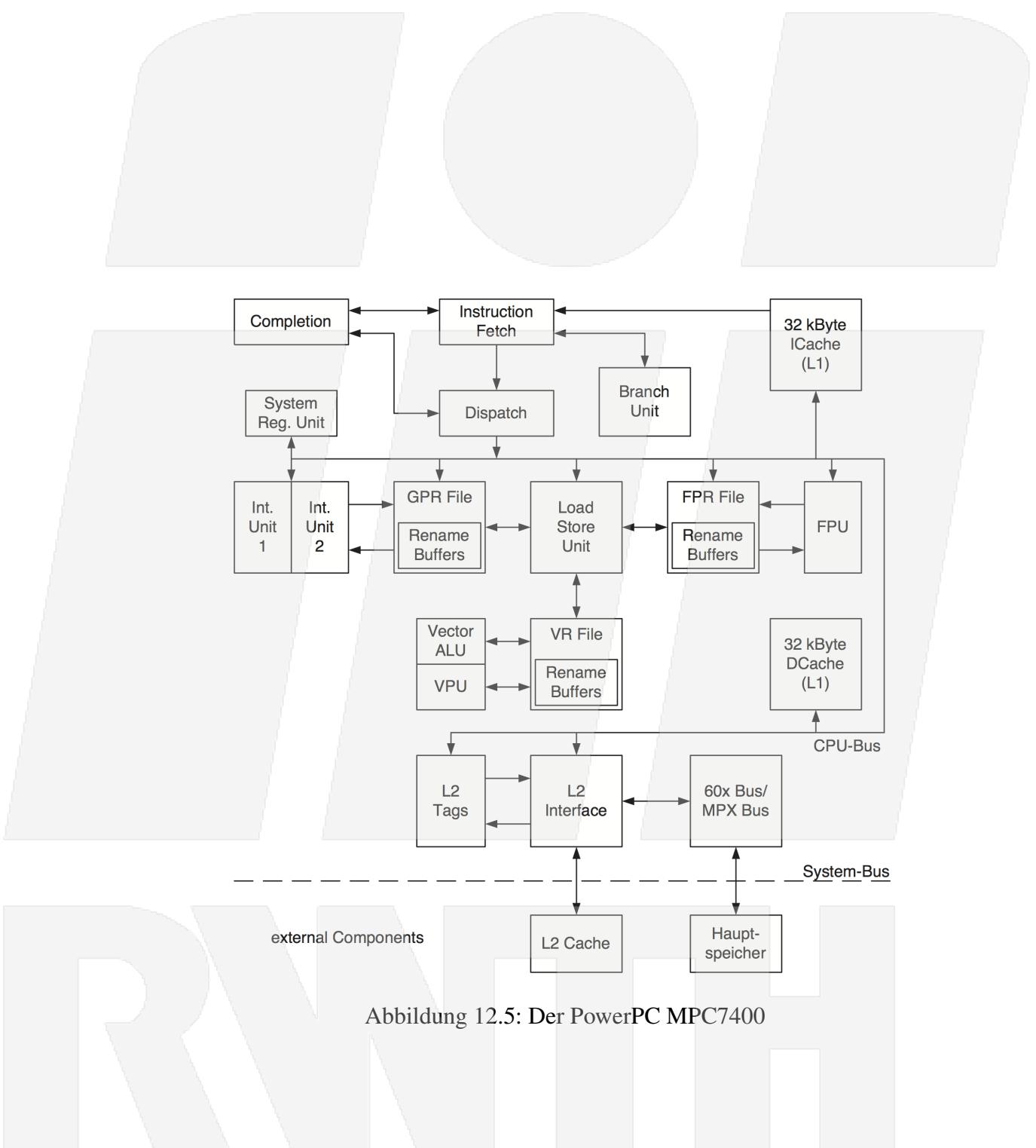


Abbildung 12.5: Der PowerPC MPC7400

tende Integer Execution Units. Wie beim Pentium III existiert eine Branch Unit. Die geladenen Befehle werden in Instruction Fetch, Completion und Dispatch in eine optimierte Reihenfolge gebracht und dann an die einzelnen Verarbeitungseinheiten weitergegeben. Die Integer- und Fließkommaregister können ebenfalls umbenannt werden, so dass anstatt die Inhalte einzelner Register zu tauschen nur deren Namen getauscht werden. Hierdurch steigt die Performance der Verarbeitungseinheiten.

Eine Besonderheit des PowerPC Prozessors ist die Realisierung eines Fließkommabefehls, der direkt eine Additions-Multiplikations-Operation für drei Fließkommaregister ausführen kann. Hiermit lassen sich numerische Algorithmen, wie z. B. Vektoroperationen oder eine Fouriertransformation, sehr effizient implementieren.

Der PowerPC 7400 wird mit einer Strukturbreite von $0.15\mu m$ gefertigt. Dadurch verringert sich der Stromverbrauch und somit der Kühlungsbedarf. Bei einer Taktfrequenz von 400 MHz verbraucht der PowerPC 7400 typischerweise etwa 6 W, der Pentium III bei 450 MHz etwa 25 W.

12.3 Leistungsbewertung von Rechnersystemen

Es existieren verschiedene Möglichkeiten, um Rechnersysteme zu bewerten:

- Hardwarekonfiguration (Größe des Hauptspeichers, Prozessor, Taktrate des Prozessors, des Caches, der Busse, verwendete Bussysteme, verfügbare Schnittstellen, etc).
- Rechenleistung in *MFlops* (Million Floating Point Operations per Second). Diese Zahl ist bei modernen Architekturen jedoch als Leistung zu sehen, die niemals überschritten werden kann. Sie sagt jedoch nichts darüber aus, ob diese Leistung real erreichbar ist.
- MFlops pro Dollar („Bang for the Buck“), als Maß für das Preis/Leistungs-Verhältnis. Diese Zahl findet sich oft bei der Bewertung von Großrechnersystemen.
- Maßzahlen standardisierter Benchmarkverfahren.

12.3.1 Benchmarkverfahren

Ein *Benchmark* ist ein Programm oder eine Sammlung von Programmen, deren Ausführungszeit gemessen und daraus eine Maßzahl gebildet wird. Diese Maßzahl dient dann zum Vergleich verschiedener Rechnersysteme auf denen der Benchmark durchgeführt wurde. Es gibt verschiedene Möglichkeiten einen Benchmark aufzubauen:

Synthetische Benchmarks Hier wird die Ausführungszeit einfacher Algorithmen, wie z. B. Suchverfahren oder eine Matrixmultiplikation gemessen. Ein Beispiel dafür ist der Linpack-Benchmark, der lineare Gleichungssysteme löst.

Benchmark-Suites Eine Sammlung einzelner Benchmarks deren Ergebnisse gewichtet zusammengefasst werden. Der unten detaillierter betrachtete SPEC-Benchmark gehört dazu.

Applikations-Benchmarks Eine Applikation, oder Teile davon werden gemessen. Applikations-Benchmarks sind in der Regel Teil einer Benchmark-Suite, z. B. SYSmark.

12.3.2 Die SPEC CPU-Suite

Die *SPEC CPU Benchmark Suite* ist ein kommerzieller Benchmark, der von der Standard Performance Evaluation Corporation entwickelt und vertrieben wird. Etwa alle drei Jahre wird dieser Benchmark überarbeitet um mit der Entwicklung der Rechnertechnologie Schritt zu halten.

Jeweils kurz nach Einführung einer neuen Version wird der Vertrieb der alten Version eingestellt und es werden nur noch mit der neuen Version erzielte Ergebnisse auf der offiziellen Benchmarkliste veröffentlicht.

Die SPEC CPU2000-Benchmark Suite ist in vier Gruppen aufgeteilt (Tab. 12.1).

Tabelle 12.1: Aufteilung der SPEC2000 Benchmark Suite

	basis	maximum
Integer	SPECint_base2000	SPECint2000
Floating Point	SPECfp_base2000	SPECfp2000

Bei den *_base2000*-Werten handelt es sich um Messergebnisse bei denen nur eine fest definierte Zahl von Optimierungen am Programmcode erlaubt ist, die bei allen am Benchmark beteiligten Programmen gleich sein muss. Bei den in der Spalte *maximum* aufgeführten Werten sind beliebige, auf die jeweiligen Aufgaben abgestimmte Optimierungen erlaubt.

Die Messwerte stellen eine relative Maßzahl bezogen auf eine Workstation von Sun Microsystems (Ultra 5) dar, einer mit 300 MHz getakteten UltraSPARC-II-CPU mit 256 KByte Level 2 Cache und einem Hauptspeicher von 256 MByte. Die SPEC-Werte dieses Rechners wurden auf 100 festgelegt.

Aufbau der Benchmark-Suite

Die Integer-Benches sind in ANSI-C oder C++ geschrieben und bestehen aus den in Tabelle 12.2 aufgeführten Programmen.

Die Floating Point-Benches sind in FORTRAN77, FORTRAN90 und C geschrieben. Viele Programme sind stark vektorisierbar (Tabelle 12.3).

Manche dieser Benchmarks bestehen aus mehreren Programmdurchläufen, z. T. mit unterschiedlichen Parametern. Das geometrische Mittel der Ergebnisse der einzelnen Programme ergibt dann die entsprechende Maßzahl. Die Ergebnisse dieses Benchmarks sind nur zusammen mit der genauen Maschinenkonfiguration und den verwendeten Compileroptionen zu veröffentlichen.

Ergebnisse

In Tabelle 12.4 sind die SPEC2000-Ergebnisse für jeweils ein Rechnersystem der verbreiteten Architekturen aufgeführt.

Um möglichst gute SPEC-Werte zu erzielen setzen viele Hersteller Systeme ein, die mit zusätzlichem Cache-Speicher oder anderen Architekturoptimierungen realisiert sind. Dies führt dazu,

Tabelle 12.2: Die Bestandteile des Integer-Teils der SPEC CPU2000-Suite (auch CINT2000 genannt)

Programm	Spr.	Aufgabe	Speicher
164.gzip	C	Datenkompression	213 MByte
175.vpr	C	Anordnung und Verbindung von Funktionsblöcken in einem FPGA (kombinatorische Optimierung)	48 MByte
176.gcc	C	Übersetzung und Optimierung von C-Code	25–160 MByte
181.mcf	C	Kostenoptimierung von Transportwegen	190 MByte
186.crafty	C	Schachprogramm	48 MByte
197.parser	C	Verarbeitung natürlicher Sprache	77 MByte
252.eon	C++	Ray-Tracing	10 MByte
253.perlbmk	C	Abarbeitung verschiedener Perl-Programme	5–150 MByte
254.gap	C	verschiedene Rechnungen mit Gruppentheorie	10 MByte
255.vortex	C	Transaktionen mit einer objektorientierten Datenbank	40–75 MByte
256.bzip2	C	Datenkompression	180–190 MByte
300.twolf	C	Layout integrierter Schaltkreise	<10 MByte

Tabelle 12.3: Die Bestandteile des Fließkomma-Teils der SPEC CPU2000-Suite

Programm	Spr.	Aufgabe	Speicher
168.wupwise	F77	Lösung von inhomogenen Verbands-Dirac-Gleichungen	175 MByte
171.swim	F77	Berechnung eines Wasserwellenmodells	190 MByte
172.mgrid	F77	Mehrfachgitter-Gleichungslöser im dreidimensionalen Potenzialfeld	55 MByte
110.applu	F77	Lösung einer parabolischen/elliptischen partiellen Differentialgleichung	180 MByte
177.mesa	C	Generierung eines 3D-Modells aus Daten mit OpenGL	10 MByte
178.galgel	F90	Analyse der oszillatorischen Instabilität einer Flüssigkeit	140–150 MByte
179.art	C	Bilderkennung mit ART2-Neuronalen Netzen	<10 MByte
183.eqquake	C	Simulation von seismischen Wellen mit Finite Elemente Methoden	50 MByte
187.facerec	F90	Gesichtserkennung	15 MByte
188.ammp	C	Simulation von Molekülbewegungen	20 MByte
189.lucas	F90	Testet Zahlen der Form $2^p - 1$ auf Primalität	145 MByte
191.fma3d	F90	Crashtestsimulation mit Finite Elemente Methoden	105 MByte
200.sixtrack	F77	Simulation der Strahlstabilität eines Partikelbeschleunigers	30 MByte
301.apsi	F77	Liefert Lösungen bzgl. Temperatur, Wind, Ausbreitungsgeschwindigkeit und Verteilung von Umweltverschmutzungen	190 MByte

Tabelle 12.4: Ergebnisse der SPEC CPU2000 Suite

Modell (Hersteller)	CPU	Takt [MHz]	1st Level Cache (KByte)	2nd Level Cache (KByte)	CINT2000		CFP2000	
					base	max	base	max
HP Superdome 64-way	PA-RISC 8700+	875	768I+1536D	—	394	413	267	288
SGI Origin 3200 1X	R14000	600	32I+32D	8192 (I+D)	483	500	499	529
AlphaServer GS80	Alpha 21264C	1001	64I+64D	8192 (I+D)	561	621	585	756
68/1001 (Compaq)								
A7N8X (Asus)	Athlon XP3200+	2200	64I+64D	512 (I+D)	1044	1080	873	982
Blade 2000 (SUN)	UltraSPARC-IIICu	1200	32I+64D	8192 (I+D)	642	722	953	1118
D875PBZ (Intel)	Pentium 4p	3200	12I+8D	512 (I+D)	1221	1261	1252	1267
Server pSeries 690 Turbo (IBM)	Power4+	1700	64I+32D	1536 (I+D)	1077	1113	1598	1699

dass die Leistung zwar im Labor gemessen, in der Realität aber nicht erreicht wird, da diese Maschinen oftmals nicht als Produkte verfügbar sind.

Die Ergebnisse des SPEC-Benchmarks sollten daher, wie bei allen anderen Benchmarks auch, nur dann als Vergleichsmaß in Betracht gezogen werden, wenn die Ergebnisse für die zu vergleichenden Maschinen vorliegen. Die Extrapolation der Ergebnisse auf eine andere Architektur führt in der Regel zu Fehlinterpretationen.

12.4 Entwicklungsperspektiven bei Speicherkapazität und Rechengeschwindigkeit

1965 stellte Gordon Moore, einer der Gründer der Intel Corporation, die Faustregel auf, dass sich die Schaltkreiskomplexität alle 12 bis 18 Monate verdoppelt. Diese Regel ist als Moores' Law (Moore'sches Gesetz) bekannt geworden und bis heute gültig. Abbildung 12.6 zeigt den Verlauf der Entwicklung seit Mitte der sechziger Jahre.

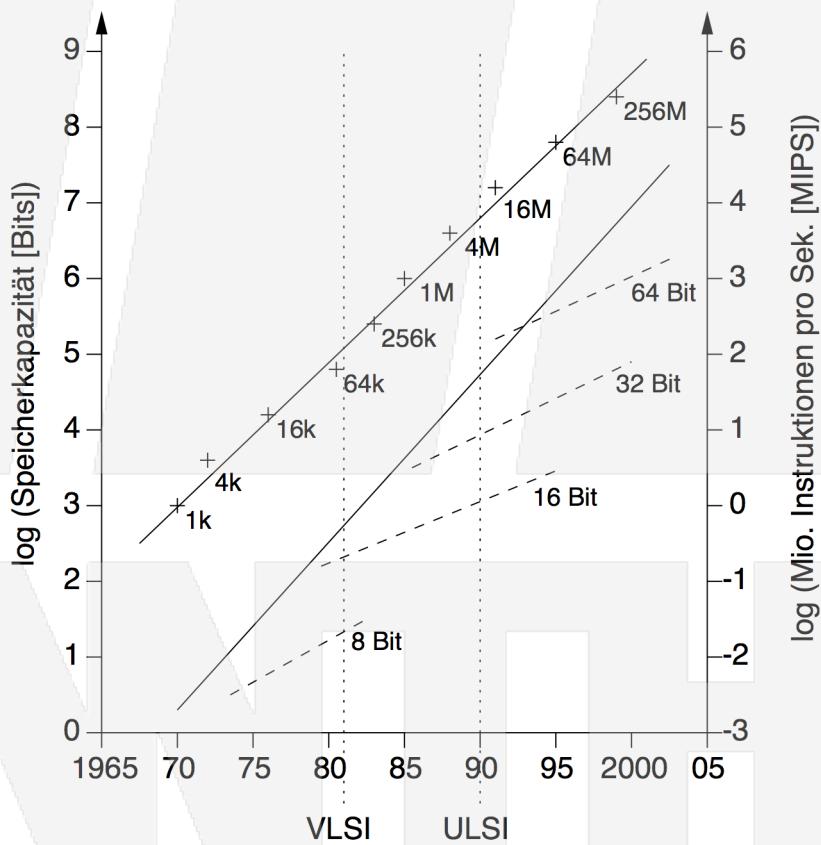


Abbildung 12.6: Entwicklung von Rechenleistung und Speicherkapazität seit Mitte der sechziger Jahre.

Tabelle 12.5 zeigt diesen Prozess an der Entwicklung der Intel Mikroprozessoren nochmals auf. Der seit 2000 erhältliche Itanium Prozessor stellt den ersten 64 Bit Prozessor von Intel dar. Im Gegensatz zu RISC Prozessoren verwendet der Itanium eine Very Long Instruction Word Architektur, bei der in einem Befehlswort Instruktionen für mehrere Ausführungseinheiten des

Prozessoren enthalten sind. Dadurch unterscheidet sich der Itanium grundlegend von den Prozessoren der Pentium Familie und deren Vorgängern. Programme die für die Pentium Prozessoren entwickelt wurden, können auf dem Itanium deshalb nur in einer Emulationsumgebung ablaufen.

Tabelle 12.5: Technische Daten der Intel Mikroprozessoren

	Vorgestellt am	Takt MHz	Bus-breite	Transistorzahl (Strukturbr. in μm)	Adressierbarer Speicher	Weiterentwicklung zum Vorgänger
4004	15.11.71	0,108	4 Bit	2300 (10)	640 Byte	erster Mikroprozessor, arithm. Berechnungen
8008	01.04.72	0,108	8 Bit	3500	16 KByte	Daten- und Zeichenmanipulationen
8080	01.04.74	2	8 Bit	6000 (6)	64 KByte	10 mal schneller als 8008
80286	01.02.82	8-12	16 Bit	134000 (1,5)	16 MByte	3-6 mal schneller als 8086
386DX	17.10.85	16-33	32 Bit	275000 (1)	4 GByte	32 Bit Architektur, L1 Cache
486DX	10.04.89	25-100	32 Bit	1,2 Mio. (0,8)	4 GByte	Integrierte FPU
Pentium	22.03.93	60-266	32 Bit	3,1 Mio. (0,5)	4 GByte	Superskalare Architektur
PentiumPro	27.03.95	150-200	32 Bit	5,5 Mio. (0,35)	4 GByte	Dynamic Execution Technology
Pentium II	07.05.97	233-450	32 Bit	7 Mio. (0,25-0,35)	64 GByte	Modulbauweise
Pentium III	26.02.99	450-1300	32 Bit	8,2 Mio. (0,18-0,25)	64 GByte	SIMD FPU
Pentium IV	2001	1300-3800	32 Bit	42 Mio. (0,13-0,18)	64 GByte	
Itanium	2001	733-800	64 Bit	25 Mio. (0,18)	64 TByte	64 Bit Architektur, Very Long Instruction Word (VLIW)
Itanium2	2002	1000-1300	64 Bit	221 Mio. (0,18)	64 TByte	
Itanium2 (Madison)	2003	1300-1500	64 Bit	410 Mio. (0,13)	64 TByte	
Pentium M	2003	1000-2266	32 Bit	77 Mio. (0,13) - 140 Mio. (0,09)	64 GByte	
Core2 Duo	2006	1666-2933	64 Bit	167 Mio. - 291 Mio. (0,065)	64 TByte	Core-Mikroarchitektur
Montecito	2006	1400-1600	64 Bit	1,72 Mrd. (0,09)	64 TByte	

Literaturverzeichnis

- [Am90] Ameling, W.
Digitalrechner – Grundlagen und Anwendungen. Technische Informatik 1
Vieweg, Braunschweig
- [Bl92] Blieberger, J., Schildt, G.-H., Schmid, U., Stöckler, S.
Informatik
Springer-Verlag, Wien New York
- [Bo96] Borucki, L.
Digitaltechnik
B. G. Teubner, Stuttgart, 4. Auflage
- [Co92] Coy, W.
Aufbau und Arbeitsweise von Rechenanlagen. Eine Einführung in Rechnerarchitektur und Rechnerorganisation für das Grundstudium der Informatik
Vieweg, Braunschweig, 2. Auflage
- [Kl83] Klar, R.
Digitale Rechenautomaten
Sammlung Göschen 2050, de Gruyter, Berlin
- [Ma01] Märtin, C.
Rechnerarchitekturen
Fachbuchverlag Leipzig/Carl Hanser Verlag, München
- [Me82] Mendelson, E.
Boolesche Algebra und logische Schaltungen — Theorie und Anwendung
Schaums Outline, McGraw-Hill, Hamburg
- [Mu99] Müller-Schloer, C., Schallenger, B., et al.
Vom Arbeitsplatzrechner zum ubiquitären Computer
VDE Verlag, Berlin
- [Re87] Rembold, U.
Einführung in die Informatik für Naturwissenschaftler und Ingenieure
Hanser Verlag, München
- [Sc92] Schiffmann, W., Schmitz, R.
Technische Informatik 1: Grundlagen der digitalen Elektronik
Springer-Lehrbuch, Springer-Verlag Berlin
- [Sc92a] Schiffmann, W., Schmitz, R.
Technische Informatik 2: Grundlagen der Computertechnik
Springer-Lehrbuch, Springer-Verlag Berlin

Literaturverzeichnis

- [Ti93] Tietze, U., Schenk, Ch.
Halbleiter-Schaltungstechnik
Springer-Verlag Berlin, 10. erweiterte Auflage
- [Wa84] Waldschmidt, E., Walter K.
Grundzüge der Informatik I
Bibliographisches Institut, Mannheim
- [Wa84a] Waldschmidt, E., Walter K.
Grundzüge der Informatik II
Bibliographisches Institut, Mannheim



Anhang A

Übersicht über die Befehle des Atmel AVRmega8

Die nachfolgende Tabelle gibt noch einmal zusammenfassend einen Überblick über die vorgestellten Befehle des Atmel AVRmega8. Sie sind hierbei alphabetisch geordnet.

Die Tabelle listet die Syntax, die Operation, den Wertebereich der Operanden, die Veränderung des Programmcounters sowie optional des Stackpointers und den 16-bit Opcode auf.



Anhang A. Übersicht über die Befehle des Atmel AVRmega8

Syntax	Operation 16-bit Opcode	Operands	Programmcounter (opt. Stack-Pointer)
ADD Rd, Rr	$Rd \leftarrow Rd + Rr$ 0000 11rrrr dddd rrrr	$0 \leq d \leq 31, 0 \leq r \leq 31$	$PC \leftarrow PC + 1$
AND Rd, Rr	$Rd \leftarrow Rd \bullet Rr$ 0010 00rrrr dddd rrrr	$0 \leq d \leq 31, 0 \leq r \leq 31$	$PC \leftarrow PC + 1$
BCLR s	$SREG(s) \leftarrow 0$ 1001 0100 lsss 1000	$0 \leq s \leq 7$	$PC \leftarrow PC + 1$
BREQ k	$if Rd = Rr (Z = 1) then PC \leftarrow PC + k + 1, else PC \leftarrow PC + 1$ 1111 00kk kkkk k001	$-64 \leq k \leq +63$	$PC \leftarrow PC + 1$
BRLO k	$if Rd < Rr (C = 1) then PC \leftarrow PC + k + 1, else PC \leftarrow PC + 1$ 1111 00kk kkkk k000	$-64 \leq k \leq +63$	$PC \leftarrow PC + k + 1; PC \leftarrow PC + 1, if condition is false$
BSET s	$SREG(s) \leftarrow 1$ 1001 0100 0sss 1000	$0 \leq s \leq 7$	$PC \leftarrow PC + 1$
CBR Rd, K	$Rd \leftarrow Rd \bullet (SFF \cdot K)$ 0111 \overline{KKKK} dddd \overline{KKKK}	$16 \leq d \leq 31, 0 \leq K \leq 255$	$PC \leftarrow PC + 1$
COM Rd	$Rd \leftarrow \$FF \cdot Rd$ 1001 010d dddd 0000	$0 \leq d \leq 31$	$PC \leftarrow PC + 1$
CP Rd, Rr	$Rd \leftarrow Rr$ 0001 01rd dddd rrrr	$0 \leq d \leq 31, 0 \leq r \leq 31$	$PC \leftarrow PC + 1$
CPI Rd, K	$Rd \leftarrow K$ 0011 KKKK dddd KKKK	$16 \leq d \leq 31, 0 \leq K \leq 255$	$PC \leftarrow PC + 1$
DEC Rd	$Rd \leftarrow Rd - 1$ 1001 010d dddd 1010	$0 \leq d \leq 31$	$PC \leftarrow PC + 1$
EOR Rd, Rr	$Rd \leftarrow Rd \oplus Rr$ 0010 01rd dddd rrrr	$0 \leq d \leq 31, 0 \leq r \leq 31$	$PC \leftarrow PC + 1$
ICALL	$PC(15:0) \leftarrow Z(15:0)$ 1001 0101 0000 1001		See Operation, Stack: Stack $\leftarrow PC + 1; SP \leftarrow SP - 2$
IJMP	$PC(15:0) \leftarrow Z(15:0)$ 1001 0100 0000 1001		See Operation.

Syntax	Operation 16-bit Opcode	Operands	Programmcounter (opt. Stack-Pointer)
IN Rd, A	Rd \leftarrow I/O(A) 1011 0AA dddd AAAA	0 \leq d \leq 31, 0 \leq A \leq 63	PC \leftarrow PC + 1
INC Rd	Rd \leftarrow Rd + 1 1001 010d dddd 0011	0 \leq d \leq 31	PC \leftarrow PC + 1
LD Rd, Z	Rd \leftarrow (Z) 1000 000d dddd 0000	0 \leq d \leq 31	PC \leftarrow PC + 1
LDD Rd, Z+q	Rd \leftarrow (Z+q) 10q0 qq0d dddd 0qqq	0 \leq d \leq 31, 0 \leq q \leq 63	PC \leftarrow PC + 1
LDI Rd, K	Rd \leftarrow K 1110 KKKK dddd KKKK	16 \leq d \leq 31, 0 \leq K \leq 255	PC \leftarrow PC + 1
LDS Rd, k	Rd \leftarrow (k) 1001 000d dddd 0000 kkkk kkkk kkkk kkkk (Doppelwort)	0 \leq d \leq 31, 0 \leq k \leq 65535	PC \leftarrow PC + 2
LSL Rd	Rd(n+1) \leftarrow Rd(n), Rd(0) \leftarrow 0 0000 11dd dddd (intern ADD Rd, Rd)	0 \leq d \leq 31	PC \leftarrow PC + 1
LSR Rd	Rd(n) \leftarrow Rd(n + 1), Rd(7) \leftarrow 0 1001 010d dddd 0110	0 \leq d \leq 31	PC \leftarrow PC + 1
MOV Rd, Rr	Rd \leftarrow Rr 0010 1rrd dddd rrrr	0 \leq d \leq 31, 0 \leq r \leq 31	PC \leftarrow PC + 1
MUL Rd, Rr	R1:R0 \leftarrow Rd \times Rr 1001 11rd dddd rrrr	0 \leq d \leq 31, 0 \leq r \leq 31	PC \leftarrow PC + 1
NOP	0000 0000 0000 0000		PC \leftarrow PC + 1
OR Rd, Rr	Rd \leftarrow Rd \vee Rr 0010 10rd dddd rrrr	0 \leq d \leq 31, 0 \leq r \leq 31	PC \leftarrow PC + 1
OUT A, Rr	I/O(A) \leftarrow Rr 1011 1AA rrrr AAAA	0 \leq r \leq 31, 0 \leq A \leq 63	PC \leftarrow PC + 1
POP Rd	Rd \leftarrow Stack 1001 000d dddd 1111	0 \leq d \leq 31	PC \leftarrow PC + 1; SP \leftarrow SP + 1

Syntax	Operation 16-bit Opcode	Operands	Programmcounter (opt. Stack-Pointer)
PUSH R _r	Stack \leftarrow R _r	0 \leq r \leq 31	PC \leftarrow PC + 1; SP \leftarrow SP - 1
	1001 001rrrrr 11111		
RCALL k	PC \leftarrow PC + k + 1	-2048 \leq k \leq 2048	PC \leftarrow PC + k + 1; SP \leftarrow SP - 2; Stack \leftarrow PC + 1
	-1101 kkkkkkkk kkkkk		
RET	PC(15:0) \leftarrow Stack		See Operation; SP \leftarrow SP + 2
	1001 0101 00000 1000		
RJMP k	PC \leftarrow PC + k + 1	-2048 \leq k \leq 2048	PC \leftarrow PC + k + 1
	1100 kkkkkkkk kkkkk		
SBR Rd, K	Rd \leftarrow Rd \vee K	16 \leq d \leq 31, 0 \leq K \leq 255	PC \leftarrow PC + 1
	0110 KKKKK dddd KKKKK		
ST Z, R _r	(Z) \leftarrow R _r	0 \leq r \leq 31	PC \leftarrow PC + 1
	1000 001rrrrr 0000 (intern: STD Z+0, R _r)		
STD Z+q, R _r	(Z+q) \leftarrow R _r	0 \leq r \leq 31, 0 \leq q \leq 63	PC \leftarrow PC + 1
	10q0 qq1rrrrr 0qqqq		
STS k, R _r	(k) \leftarrow R _r	0 \leq r \leq 31, 0 \leq k \leq 65535	PC \leftarrow PC + 2
	1001 001rrrrr 00000 kkkkk kkkkk kkkkk (Doppelwort)		
SUB Rd, R _r	Rd \leftarrow Rd - R _r	0 \leq d \leq 31, 0 \leq r \leq 31	PC \leftarrow PC + 1
	0001 10rd dddd rrrr		

Tabelle A.1: Übersicht über alle vorgestellten Assemblerbefehle des Atmel AVRmega8

Anhang B

Weiterführende Links zum Thema Assembler

Um einen tieferen Einblick in die Programmierung mit Assembler (von AVR) zu erhalten, bieten sich die folgenden Links an:

- Tutorial:
 - <http://www.avr-asm-tutorial.net/>
 - <http://www.mikrocontroller.net/articles/AVR-Tutorial>
- Chip-Dokumentation:
 - http://www.atmel.com/dyn/resources/prod_documents/doc2486.pdf
- Sprach-Dokumentation:
 - http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf
- Development-Umgebung:
 - http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725
- Dokumentation zur Entwicklungsumgebung:
 - http://www.atmel.com/dyn/resources/prod_documents/novice.pdf
 - http://www.atmel.com/dyn/resources/prod_documents/doc2510.pdf

Index

Übergangsfunktion, 98, **100**
Übergangstabelle, **101**
Übertragung
 asynchrone, **185**
 isochrone, **185**
8-Bit-RISC-Prozessor, **142**
8-Bit-Register, **154**
80/20-Regel, 213

A/D-Wandlers, **152**
Abbruchbedingung, **158**
Absorptionsgesetz, **46**, 49
Accelerated Graphics Port, **179**
ADD, **153**
Addierer, **73**
Addition, **149**, **153**
Adresse, **197**
Adressierungsarten, **150**
AGP, **173**, **179**
Akzeptor, **105**
Alphabet, **12**
ALU, **116**, **119**, **123**, **142**, **145**, 222
ALU-Berechnung, **149**
ALU-Operation, **149**
Analog-Digitalwandler, **142**
Analogausgabe, **195**
Analogeingabe, **195**
AND, **150**, **154**
Arbiter, **176**
 dezentral, **176**
Arithmetische Operationen, **149**
arithmetischen Verknüpfungen, **153**
ASCII, **18**
Assembler, **141**
Assemblersprache, **141**
Assoziativgesetz, **46**, 49
Assoziativspeicher, **204**, 216
Atmel AVR, **141**
Atmel AVRmega8, **141**
Ausführungszeit, **148**
Auszaben, 98, 100
Ausgabeports, **142**
Ausgangsfunktion, 98, **101**

Ausgangsmenge, **100**
Ausgangstabelle, **101**
Aussagenalgebra, 48
Automatenband, **101**
Automatengraphen, **101**
Automatentabelle, **101**, 110
Autonomer Automat, **104**
AVR, **237**

Backside-Cache, **215**
BCLR, **155**
Befehlsregister, **141**
Befehlstypen, **150**
Befehlsworte, **148**
Befehlszähler, **116**, **155**
Benchmark, **226**
Bipolartransistor, **200**
Bit-Befehle, **150**
bit-parallel, **171**
bit-seriell, **171**
Bit-Stelle, **153**
Bitweise logische Verknüpfung, **150**
Bitweise Und-Verknüpfung, **154**
Bitweises Negieren, **154**
bitweises Vergleichen, **150**
Blocked Multithreading, **139**
Blockprüfung, **26**
Blu-ray Disc, **212**
Bluetooth, **191**
Boolesche Algebra, **45**, **46**, 48
Boolesche Funktion, **50**
Boolesche Funktionen, 65
Brücke, **175**
Branch Prediction, **132**
BREQ, **156**
Bridge, **175**
BRLO, **156**
BSET, **155**
Bus, **170**
 asynchron, **172**
 logisch, **171**
 multiplex, **172**
 parallel, **171**

- physikalischer, **171**
segmentiert, **174**
seriell, **172**
synchron, **172**
Bus Grant, **176**
Bus Request, **176**
Busarbiter, **172**
Busarbitrierung, **176**
Cache-Lines, **215**
Cache-Speicher, **214**
Carry-Flag, **156**
CBR, **155**
CD-R, **209**
CD-ROM, **209**
CD-RW, **209**
Centronics-Schnittstelle, **189**
CISC, **137**, **219**
Code, **13**
Codeanteil, **148**
Codeumsetzung, **75**
Codierung, **13**
Codierungstabellen, **111**
COM, **154**
Comparator, **152**
Compare, **150**
Complex Instruction Set Computer, **137**
Conroe/Merom, **220**
Control Unit, **126**, **141**
Core, **220**
CP, **154**
CPI, **154**
CPU, **116**, **129**, **141**, **149**
CPU-Bus, **173**
Cycle Stealing, **193**
D-Flipflop, **85**, **86**
DAT, **206**
Data Cartridge, **206**
Datenbus, **142**
Datenmanipulation, **144**
Datenspeicher, **141**
Datentransferbefehle, **149**, **150**
DEC, **153**
DeMorgansche Gesetze, **46**, **49**
Demultiplexer, **79**
DEMUX, **79**
Device Bays, **185**
Dezimalcode, **75**
Dezimalsystem, **28**
DF, **53**
Digital Audio Tape, **206**
Digitalrechner, **149**
Direkte Ein-/Ausgabe, **192**
direktes Speichern, **151**
Disjunktive Normalform, **53**
Disketten, **207**
Division, **149**
DMA, **193**
DNF, **53**, **54**
Doppelfehlererkennung, **27**
Doppelte Negation, **49**
Dotierung, **198**
Drucker, **164**
 Farb-, **168**
 Laser-, **166**
 Nadel-, **164**
 Tintenstrahl-, **166**
Dual-Code, **78**
Dualsystem, **28**
DVD, **210**
DVD+RW, **212**
DVD-R, **212**
DVD-RAM, **212**
DVD-RW, **212**
E/A-Funktionen, **152**
EEPROM, **93**, **206**
Ein-/Ausgabe, **191**
 über DMA, **193**
 direkte, **192**
 gepufferte, **192**
 von Analogdaten, **195**
Ein-/Ausgabebus, **173**
Einerkomplement, **34**
Eingaben, **98**, **100**
Eingangsmenge, **100**
Einzelfehlerkorrektur, **26**
Embedded Systems, **141**
Entropie, **14**
EPROM, **93**, **205**
execute, **148**
Exklusiv-Oder-Verknüpfung, **154**
Exzess-3-Code, **75**
Fano-Bedingung, **17**

Fano-Code, **17**
Farbdrucker, 164, **168**
Feldeffekttransistor, **200**
Festkommaarithmetik, **37**
Festplatte, **207**
FET, **200**
fetch, **148**
Field Programmable Logic Array, **96**
FIFO, **217**
FireWire, **185**
FPLA, **96**
Funktionswert, **158**

Ganzzahlenarithmetik, **34**
Gatter, **65**
Gattersymbole, **66**
Gepufferte Ein-/Ausgabe, **192**
Gleitkommaarithmetik, **40**
Grafikbeschleuniger, **162**
Grafikkarte, **160**
Gray-Code, **22**, 78

Halb-Automat, **104**
Halbaddierer, **65**
Hamming-Distanz, **24**
Harvard-Architektur, **129**, **141**, 214, 221
Hauptprogramm, **155**
Hauptspeicher, 9
Hexadezimalsystem, **28**
Hochsprache, **148**
HornerSchema, 29, **29**
Host-Brücke, **173**
Hub, 174, **174**
Huntingtonsche Axiome, **45**, 48
Hyper Threading, **223**
Hyperthreading, **140**

ICALL, **157**
Idempotenzgesetz, **46**, 49
IEEE-Gleitkomma-Format 754, **39**
IN, **152**
INC, **153**
indirekten Speichern, **151**
indirekten Speichern mit Verschiebung, **152**
indirekten Sprung, **156**
Informationsgehalt, **14**
Infrared Data Association, **191**
Initialisierung, **152**

Inline-Cache, **215**
Instruction Decoder, **142**
Inter-Integrated Circuit, **143**
Interfaces, **170**
Interleaved Multithreading, **139**
Interrupts, **150**
Invertieren, **154**
IrDA, **191**
ISA-Bus, **178**
Itanium, **220**, 229

J-K-Flipflop, **86**
mit Rückflankensteuerung, **88**

Kardinalität:, **12**
Karnaugh-Veitch-Diagramm, **54**
Keller, **146**
Kellerspeicher, **146**
Kellerspeichers, **149**
KF, **53**
KNF, **53**, 54
Knoten
 aktiver, **172**
 passiver, **172**
Komparator, **143**
Konjunktive Normalform, **53**
Konkatenation, **12**
Konstante, **151**, **154**
Koprozessoren
 Gleitkomma-, **133**
KV-Diagramm, **54**

Längsparität, **27**
Laserdrucker, 164, **166**
Last In, First Out, **146**
Laufzeit, **73**
LDI, **151**
LFD, **217**
LIFO, **146**, **155**
Line Size, **215**
Linksshift, **153**
Logische Operationen, **150**
Look-aside-Cache, **214**
LRU, **217**
LSL, **153**
LSR, **153**

m-aus-n-Codes, **23**

- Magneto-Optical Disc, **213**
Makrozellen, **97**
Marken, **157**
Maschinenbefehle, **141**
Maschinenebene, **148**
Maschinensprache, **141, 147**
Master, **172**
Maxterm, **53**
Mealy-Automat, **102**
Memory Management Unit, **217**
Mengenalgebra, **46**
MFlops, **226**
Mikrocode, **222**
Mikrocontroller, **141, 151**
Mikroprogrammierung, **128**
MIMD, **132**
Minimierung mit KV-Diagrammen, **57**
Minterm, **53**
MMU, **217**
MMX, **219**
Mnemonic, **147**
MOD, **213**
Moore-Automat, **103**
Moores' Law, **229**
Morse-Code, **16**
Morse-Codierung, **13**
MOSFET, **200**
MOV, **150**
MROM, **204**
MUL, **153**
Multimaster-Bus, **172**
Multiplexer, **80**
Multiplikation, **149, 153**
Multithreading, **138**
MUX, **80**
Nadeldrucker, **164, 164**
Netburst-Architektur, **221**
Oder-Verknüpfung, **154**
Oktalsystem, **28**
Operand, **156**
OR, **150**
OUT, **152**
Out of Order Execution, **136**
Paging, **217**
PAL, **95**
Parallelrechner, **132**
Parameter, **148**
Parität, **26**
PCI-Bus, **173, 178**
PCI-Express, **179**
PCMCIA, **180**
Pentium, **219**
Pentium 4, **220, 221**
Pentium II, **220**
Pentium III, **220**
Peripheriebus, **173**
Peripheriegeräte, **7**
Phase Change, **210**
Piconet, **191**
Pipelining, **130, 131, 133, 134**
Polyadische Zahlensysteme, **28**
POP, **146, 152**
Ports, **152**
PowerPC, **223**
PowerPC 7400, **224**
Primimplikanten, **61**
 wesentlich, **61**
Programmablaufsteuerung, **150**
Programmable Array Logic, **95**
Programmcounter, **148, 155, 156**
Programmende, **148**
Programmspeicher, **141**
PROM, **93, 205**
Protokoll, **170**
Prozessorarchitektur, **147**
Prozessorfamilie, **147**
PUSH, **146, 152**
Querparität, **27**
Quinärsystem, **28**
Quine-McCluskey-Verfahren, **61**
R16, **151**
R31, **151**
Rücksprung, **155**
Rücksprungadresse, **146**
Rahmenfestplatte, **209**
RAM, **92, 197**
 dynamisch, **203**
 statisch, **203**
RAMDAC, **161, 162**
Random Access Memory, **92, 197**
Rd, **150, 154**

Read Only Memory, 92, **197**

Rechengeschwindigkeit, 9

Rechenwerk, 116, **119, 142**

Rechtsshift, **153**

Reduced Instruction Set Computer, **137**

Reduktionsgesetze, 49

Register, **145, 149**

Register Renaming, **136**

Register-Bus, **173**

Registern, **149**

Rekursion, **155, 158**

Rekursionsanfang, **158**

Rekursionsschritt, **158**

rekursive Beschreibung, **158**

Repeater, **174**

REPROM, **93**

Restmatrix, **61**

RET, **156**

RISC, **137, 138, 219, 223**

RJMP, **156**

ROM, **92, 197, 204**

electrically erasable, 93

erasable, 93

programmable, 93

reprogrammable, 93

Rr, **150**

RS-Flipflop, **83, 86**

mit Zustandssteuerung, **83**

mit Zwei-Zustandssteuerung, **86**

RS232, **187**

Sägezahnspannung, **195**

SBR, **154**

Scatternet, **191**

Schaltalgebra, **48**

Schaltfunktion, **50**

Schaltkette, **70**

Schaltnetz, **65**

Schieberegister, 90, **90, 91, 92**

Schleife, **157**

Schleifen, **157**

Schnittstelle, **186**

Centronics, **189**

Infrarot-, **191**

parallele, 189

serielle, **187**

Schnittstellen, **170**

SCSI, **173**

SCSI-Bus, **180**

sequenziellen Programm, **155**

Serial Peripheral Interface, **143**

Shannon'sche Formel, **14**

Shannon-Gesetze, 49

SIMD, **132**

Simultaneous Multithreading, **139**

Slave, **172**

Sleep-Modus, **148**

SPEC, **227**

Speicher, **151, 197**

Assoziativ-, **204**

dynamisch, **198**

statisch, **198**

virtueller, **217**

Speicheradresse, **151**

Speicherbus, **173**

Speicherglieder, **83**

Speicherkapazität, **198**

Speicherzelle, **197**

Sprünge, **155**

Sprungmarke, **155, 157**

Sprungvorhersage, **132, 222**

SRAM, **142**

ST, **151**

Stack, **142, 146, 155, 157**

Stack-Pointer, **142, 147**

Stackoperationen, **152**

Stackpointer, **152**

Stapel, **146**

Statusregister, **145, 155**

STD, **151**

Steuereinheit, **141**

Steuerwerk, 116, **126, 142, 145**

STS, **151**

Stufenumsetzung, **195**

SUB, **153**

Subroutine, **146**

Subtraktion, **149, 153**

Supercomputer, **133**

Superskalarität, 132, **134**

Swapping, **217**

Switch, **175**

Switches, **174**

Symbole des Maschinencodes, **147**

Synchronisationssignale, **161**

System-Bus, **173**

Tag, **215**

Tag-RAM, **215**

Takt, **148**

Teilproblem, **158**

Ternärsystem, **28**

Tetradencodes, **20**

Thread, **138**

Timer, **142**

Timers, **152**

Tintenstrahldrucker, **164, 166**

TLB, **217**

Topologie

Baum, **174**

Baum-, **174**

Ring, **174**

Stern, **174**

Trackball, **170**

Translation-Lookaside Buffer, **217**

Umwandlung von Zahlen, **31**

unbedingte Sprünge, **150**

Unicode, **18**

Unified Cache, **214**

Universal Asynchronous Receiver Transmitter, **143**

Unterbrechungsverarbeitung, **150**

Unterprogramm, **155, 157**

Unterprogrammaufrufe, **155**

USB, **173, 182**

V.24-Schnittstelle, **187**

Vektoreinheit, **133**

Vektorrechner, **133**

Venndiagramm, **47**

Vergleicher, **70**

Vergleichsbefehle, **150**

Very Long Instruction Word, **138**

Verzweigebefehle, **150, 155**

Verzweigung, **155**

VHS-Videobänder, **207**

VLIW, **138**

Volladdierer, **68**

von Neumann-Architektur, **129**

von Neumann-Rechner, **141**

Vorzeichen-Betrag-Darstellung, **33**

Wägeverfahren, **195**

WHILE <Bedingung> DO <Anweisung>, **157**

Winchesterplatte, **207**

Woodcrest, **220**

WORM, **210**

Wort, **12**

Wortbreite, **141**

write back, **149**

Write-Back, **216**

Write-Through, **216**

X-, Y- und Z-Register, **145**

Z, **156**

Zähler, **89**

asynchrone, **89**

synchrone, **89**

Zentraleinheit, **116, 129**

Zero-Flag, **156, 157**

Zielregister, **151**

Ziffernschreibweise, **28**

Zugriffsart, **197**

Zugriffszeit, **197**

Zuordner, **103**

Zustände, **98, 100**

Zustandsmenge, **100**

Zweierkomplement, **34**

Zykluszeit, **198**