Grundlagen der Digitaltechnik
Zur Einführung und Wiederholung

Ausgabestand 1.1

- Nur zur Information -

1. Grundlagen

1.1. Analoge, digitale und binäre Arbeitsweise

Wollen wir irgendeine informationsverarbeitende Einrichtung bauen, so müssen wir technische Kenngrößen als Träger der Information festlegen. Das können elektrische Spannungen sein, aber auch Druck-Werte in einem pneumatischen Gerät, der Drehwinkel einer Welle oder der Verschiebe-Weg eines beweglichen Stabes - die Erfinder sind hier auf die tollsten Ideen gekommen. Im folgenden wollen wir uns naheliegenderweise auf elektrische Kenngrößen beschränken.

Bei einer *analogen* Informationsdarstellung bzw. Arbeitsweise wird der gesamte Wertebereich der technischen Kenngröße (Spannung, Strom, Frequenz, Phasenlage usw.), die das informationstragende Signal bildet, lückenlos ausgenutzt.

Hingegen handelt es sich um eine *digitale* Arbeitsweise, wenn nur eine festgelegte Anzahl einzelner Werte zur Informationsdarstellung genutzt wird. (Gleichbedeutend zum Begriffspaar "analog" - "digital" werden auch die Begriffe "kontinuierlich" und "diskret" verwendet.)

Nutzt man nur zwei Werte, so spricht man von der *binären* Arbeitsweise.

"Binär" und "digital" werden üblicherweise als Synonyme verwendet, obwohl sie nicht immer dasselbe bedeuten (so ist die Tonfrequenzwahl beim Telefon digital, aber nicht binär). Der Sprachgebrauch ist aber so weit verbreitet, daß wir uns ihm anschließen müssen.

Der wesentliche Vorteil der digitalen Arbeitsweise
Die Welt ist analog, Informationsverarbeitung findet aber - beim derzeitigen Stand der Technik - vorwiegend digital statt. Das ist im wesentlichen auf einen einzigen Sachverhalt zurückzuführen:

- eine Analogdarstellung kann nicht unbegrenzt genau sein. Die Genauigkeit der Informationsdarstellung wird vielmehr wesentlich davon abhängen, wie genau die betreffende Einrichtung gefertigt werden kann, und letzten Endes durch physikalische Tatsachen (z. B. Wärmedehnung) begrenzt sein.
- bei der digitalen Informationsdarstellung haben wir hingegen nur eine feste Anzahl von Werten (im binären Fall sogar nur zwei).

Dies ermöglicht es, für jeden dieser Werte einen Werte*bereich* (ein Toleranzfeld) anzugeben. Zwischen diesen Wertebereichen werden Lücken gelassen ("verbotene" Bereiche"). Demgemäß müssen wir die informationsverarbeitenden Einrichtungen nur so genau fertigen, daß sie in der Lage sind, jeden Wert im Rahmen seines Toleranzfeldes zu halten - es muß lediglich ausgeschlossen werden, daß die Signalwerte in die verbotenen Bereiche kommen. Die binäre Arbeitsweise hat hierbei natürlich unschlagbare Vorteile, denn es sind nur zwei Bereiche (Toleranzfelder) für die Informationswerte vorzusehen, zwischen denen ein einziger verbotener Bereich liegt. Abbildung 1.1 veranschaulicht dies am Beispiel einer zweiwertigen, dreiwertigen und zehnwertigen Informationsdarstellung im Rahmen eines Spannungsbereichs von 0 bis 10 V.

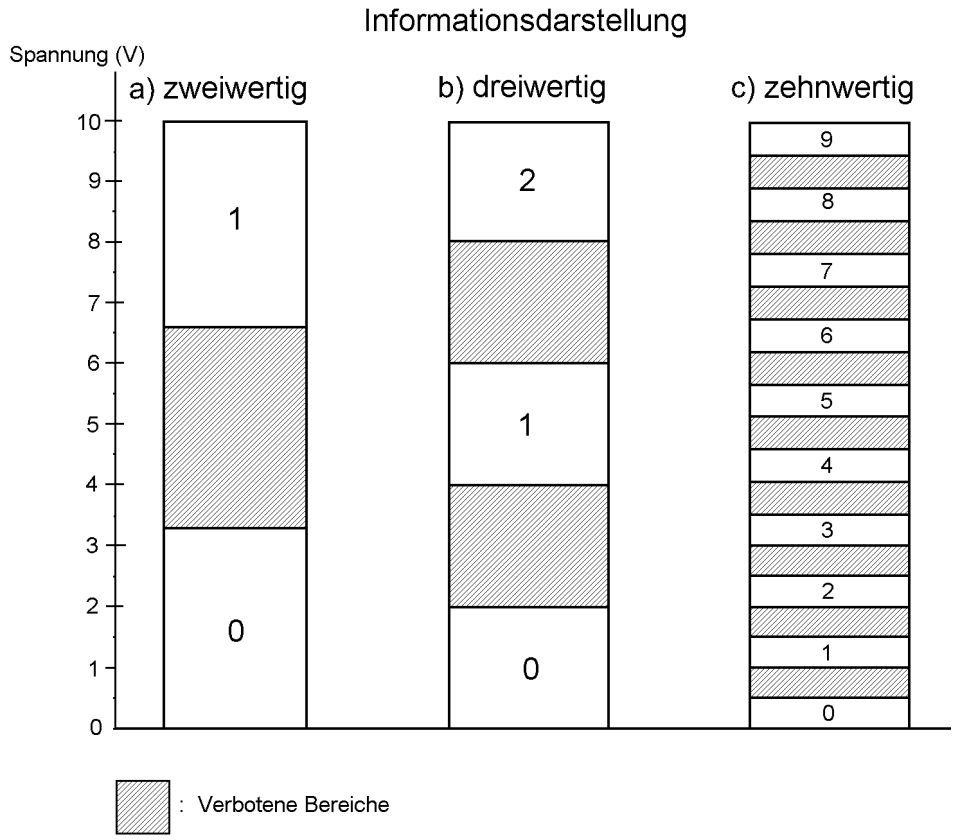


Abbildung 1.1 Vergleich verschiedener digitaler Informationsdarstellungen

Es leuchtet sofort ein, daß es wesentlich aufwendiger ist, die Genauigkeits-Anforderungen für eine zehnwertige Darstellung zu erfüllen als für eine zweiwertige. Auch sind mehrwertige Darstellungen empfindlicher gegen Störungen. Betrachten wir als Beispiel ein Signal mit einem Nennwert von 1 V. Es sei von Störungen überlagert, so daß der eigentliche Wert unregelmäßig zwischen 0,5 und 2,5 V schwankt. Gemäß Abbildung 1.1a repräsentiert dieses Signal, mag es schwanken wie es will, stets eine Null. Gemäß Abbildung 1.1c hingegen nimmt es bisweilen den Wert 1, bisweilen den Wert 2 an und kommt auch gelegentlich in einen verbotenen Bereich. Das betreffende System wäre unter diesen Bedingungen nicht arbeitsfähig.

Zur weiteren Verdeutlichung ein Blick in die Geschichte: Charles Babbage hatte für den ersten Computer überhaupt die zehnwertige (dezimale) Zahlendarstellung des gewöhnlichen Rechnens beibehalten. Er hatte mit enormen Schwierigkeiten hinsichtlich der Fertigungsgenauigkeit zu kämpfen, mußte eigens Sondermaschinen entwickeln und konnte sein Gerät nicht fertigstellen. Hingegen ist es Konrad Zuse gelungen, bereits einen ersten Versuchsaufbau, der ebenfalls rein mechanisch, aber auf binärer Grundlage arbeitete (Z1), zum durchaus befriedigenden Funktionieren zu bringen, obwohl die Einzelteile von Hand (mittels einer elektrischen Laubsäge) aus Blech zugeschnitten wurden.

Seit die binäre Informationsdarstellung zum Stand der Technik geworden ist, haben sich mehrwertige Darstellungsweisen nie mehr durchsetzen können, obwohl sie durchaus ihre Vorteile haben (so braucht man für die Übertragung einer natürlichen Zahl zwischen 0 und 9 bei zweiwertiger Darstellung 4 Signalleitungen, bei zehnwertiger Darstellung hingegen nur eine).

Anwendungsbeispiele mehrwertiger Signaldarstellungen:

- Datenübertragung über Modems (in jedem elementaren Zeitschritt (Baud) wird mehr als ein Bit übertragen),
- Auslösen von Sonderfunktionen (Programmieren, Einstellen besonderer Testzustände usw.) in an sich binären Schaltkreisen (z. B. in EPROMs und programmierbarer Logik),
- Speicherschaltkreise mit extremer Speicherdichte (2 Bits je Speicherzelle in verschiedenen Flash-ROMs und DRAMs (z. B. 4-GBit-Typen),
- Datenübertragung über Hochleistungsinterfaces. Beispiel: Quad Rambus Signaling Logic QRSL (4 Signalpegel ≙ 2 Bits je Taktflanke und Anschluß).

Groß in Mode: die Digitalisierung
Die Idealvorstellung: alles, was es an Information gibt, wird digitalisiert, das heißt in binäre Signale umgesetzt. Solche Signale können, einmal gewandelt, infolge der extremen Unempfindlichkeit gegen Störeinflüsse (vgl. Abbildung 1.1), praktisch immer wieder 100%-ig identisch (also fehlerfrei) *reproduziert* werden.

Hinzu kommen die allgemeinen Vorzüge der digitalen Schaltungstechnik:

- digitale Information kann man in praktisch unbegrenztem Umfang nahezu unbegrenzt lange speichern*),
- digitale Information kann man an sich beliebigen Wandlungen unterziehen (Rechenoperationen, Datenkompression, Verschlüsselung usw.),
- digitale Angaben lassen sich beliebig sortieren und nach bestimmten Mustern durchsuchen,
- die binäre Codierung ist eine universelle und einheitliche Form der Informationsdarstellung (für Zahlenwerte, Zeichenketten, Steuerungsangaben (Programme/Befehle), Sprache, Musik, Fernsehbilder usw.),
- Fehlererkennung und Fehlerkorrektur sind mit annehmbaren Aufwand durchführbar,
- digitale Systeme lassen sich in hochintegrierten Schaltungen verwirklichen und kostengünstig fertigen.

*): es bereitet beachtliche Schwierigkeiten, analoge Signale zu speichern (technische Mittel hierfür sind u. a. Magnetbänder und Ultraschall-Verzögerungsleitungen).

Wie gut ist eigentlich "digital"?
Von der digitalen Speicherung und Verarbeitung eigentlich analoger Signale werden Wunder erwartet - und es werden nicht selten auch Wunder versprochen. Tatsächlich gilt aber: "digital" bedeutet keineswegs "100% fehlerfrei".

Es gibt 2 grundsätzliche Fehlerquellen:

1. die Wandlung des einzelnen analogen Signalwertes in eine digitale Darstellung (Digitalisierung),
2. die Wiedergabe des zeitlichen Signalverlaufs durch einzelne aufeinanderfolgende digitale Signalwerte (Signalabtastung).

1.2. Wie werden digitale Systeme realisiert?

1.2.1. Überblick

Im folgenden geht es darum, wie sich Aufgabenstellungen der Digitaltechnik mit Einrichtungen lösen lassen, die auf modernen Halbleiter-Schaltkreisen beruhen. Auch wenn wir nicht selbst entwickeln: in der Servicepraxis können wir es mit jedem möglichen Lösungsansatz zu tun bekommen. Deshalb müssen wir diese Ansätze wenigstens im Überblick kennen. Abbildung 1.2 veranschaulicht den grundsätzlichen Aufbau einer digitalen informationsverarbeitenden Einrichtung.

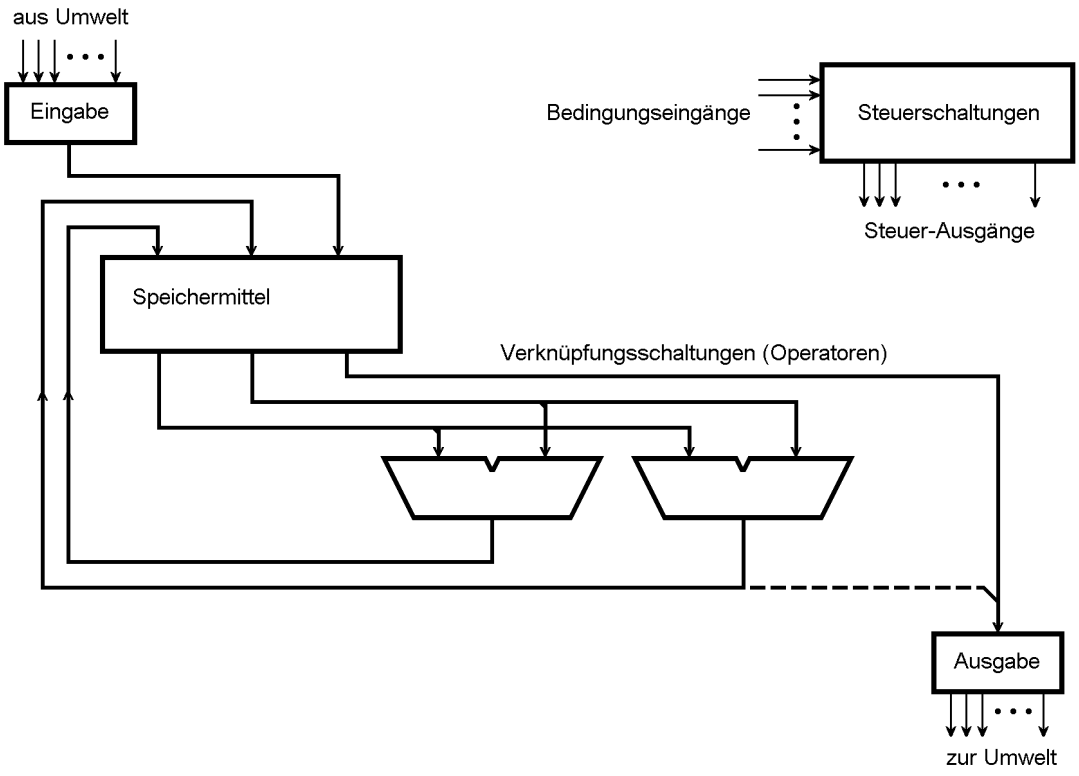


Abbildung 1.2 Digitale informationsverarbeitende Einrichtung (grundsätzlicher Aufbau)

Ein- und Ausgabe
Einrichtungen ohne jegliche Verbindung zur Um- bzw. Außenwelt sind anwendungsseitig sinnlos. Die Um- bzw. Außenwelt kann sowohl durch Meßfühler, Geber, Stellglieder, Signalwandler usw. einer "zu steuernden" Einrichtung gegeben sein, aber auch durch Bedien- und Anzeigemittel, die vom Menschen direkt genutzt werden, wie Tastaturen und Bildschirme.

Informationen aus der jeweiligen Umwelt werden in Eingabeschaltungen in die interne Informationsdarstellung gewandelt. Ergebnisse der Informationsverarbeitungsvorgänge werden über Ausgabeschaltungen in der Umwelt zur Wirkung bzw. zur wahrnehmbaren Darstellung gebracht (sichtbare Anzeige, akustische Signale, Sprachausgabe usw.).

Verarbeitung
Die Verarbeitungsvorgänge selbst laufen über Verknüpfungsschaltungen ab. Verknüpfungen zu verarbeitender Daten werden auch als Operationen bezeichnet. Demgemäß ist für die entsprechenden Schaltmittel auch die Bezeichnung "Operationswerk" in Gebrauch,

Speichermittel
Speicher haben vielfältige Angaben aufzunehmen: Daten, Zwischenwerte der Verarbeitung, Maschinenzustände, Steuerangaben (Maschinenbefehle) usw.

Steuerschaltungen
Die Steuerschaltungen wirken über Steuersignale auf alle anderen Schaltmittel ein. In einfachen Fällen ist nur eine zeitstarre Folge von Eingabe-, Verknüpfungs-, Speicher- und Ausgabevorgängen zu steuern. Zumeist müssen die Steuerschaltungen aber auf Bedingungen reagieren; deshalb sind in Abbildung 1.12 auch Bedingungssignale dargestellt.

Hinweis:
Wir sollten uns nicht nur *ein* Operationswerk, *einen* Speicher usw. vorstellen (in Entsprechung zum Aufbau eines einfachen Universalrechners (Prozessors, Mikrocontrollers), sondern durchaus auch Anordnungen, die mehrere Operationswerke, Speicher usw. enthalten (und das womöglich in ganz trickreichen Verschaltungen).

Im folgenden wollen wir die Möglichkeiten erläutern, die heutzutage zur Wahl stehen, wenn es gilt, solche Verarbeitungsaufgaben zu lösen.

1.2.2. Fertige Hardware-Plattformen

Es wird ein fertiges, am Markt erhältliches Computer-System verwendet, und die konkrete Aufgabe wird im wesentlichen durch Programmierung (= “mit Software”) gelöst (gelegentlich kann es notwendig sein, zu Anpassungszwecken aufgabenspezifische Hardware vorzusehen).

Solche fertigen Hardware-Plattformen gibt es in vielfältigen Abstufungen nach Preis, Funktionalität und Leistungsvermögen. Damit das Konzept (Hardware fertig kaufen und im wesentlichen nur programmieren) überhaupt funktioniert, sind gewisse standardisierte Schnittstellen notwendig (und seien es Standards der jeweiligen Anbieter). Für diese Standardisierung gibt es im wesentlichen drei Zugänge: den Prozessor, den Bus und die Systemumgebung.

Standardisierung durch den Prozessor
Das ist besonders im unteren Preis- und Leistungsbereich üblich. Es gibt fertige Moduln mit weithin verbreiteten Mikrocontrollern bzw. -prozessoren, die man noch durch eigene Anpassungs- und Zusatzhardware beschalten muß. Auch ist es nicht besonders schwierig, Mikrocontroller und kleine Prozessoren als Schaltkreise in eigene Entwürfe einzubauen. Seit den 80er Jahren man hat sich viel Mühe gegeben, die Schnittstellen so auszulegen, daß sich die Schaltkreise auf einfache Weise mit Speichern, E-A-Einrichtungen usw. zusammenschalten lassen (meistens genügt Einsatz “nach Kochbuch”). Die Schwierigkeiten wachsen mit der “Dicke” des Prozessors[*)] und mit der Taktfrequenz (Tabelle 1.1).

*): ein salopper Ausdruck, der sowohl die Zugriffsbreite und die Taktfrequenz als auch die Kompliziertheit des Interfaces einschließt. IA-32-Prozessoren (vom 386 an aufwärts) sind *nicht* einfach zu handhaben.

Taktfrequenz	Technologie	Entwurfsregeln
bis ca. 8...10 MHz*)	(noch) Zweiebenen-Leiterplatten	gesunder Menschenverstand**), Faustformeln
bis ca. 33 MHz	Mehrebenen-Leiterplatten	Faustformeln noch anwendbar, Leiterplattenentwicklung (Placieren, Routen) noch auf empirischer Grundlage möglich
> 33 MHz	Mehrebenen-Leiterplatten	Faustformeln nicht mehr ausreichend, statt dessen Analogsimulation von Bauelementen und Leiterplatte auf Grundlage möglichst genauer Ersatzschaltungen

*): bei mäßiger Flankensteilheit und Treibfähigkeit (bei steilen Flanken und hoher Treibfähigkeit gilt die nächste Zeile); **): mit brauchbaren Kenntnissen der Impulstechnik angereichert - Impulse sind nicht einfach zerhackter Gleichstrom...

Tabelle 1.1 Realisierungsprobleme in Abhängigkeit von der Taktfrequenz

Standardisierung durch den Bus
Das ist gleichsam das klassische Prinzip. Man entscheidet sich für ein Bussystem und wählt die passenden Steckkarten mit Prozessoren bzw. kompletten Rechnern, Speichern, E-A-Baugruppen usw. aus. Damit kann man vom Prinzip her auch höchsten Leistungsanforderungen gerecht werden.

Standardisierung durch die Systemumgebung
Während bei der Bus-Orientierung Prozessoren, Betriebssysteme usw. an sich beliebig gewählt werden können (und am Markt entsprechende Wahlmöglichkeiten angeboten werden), steht hier eine komplette Systemarchitektur (Hardware + Software-Plattform) im Vordergrund. Die Verbreitung der Wintel-PCs läßt diesen Ansatz mehr und mehr an Bedeutung gewinnen. So gibt es Industrie-PCs verschiedener Ausführungsformen, PC-Moduln, die bei kleinstmöglichen Abmessungen voll zum "Industriestandard" kompatibel sind, entsprechende Steckkarten für die gängigen Bussysteme usw. Die wirtschaftlichen Vorteile - auf Hard- und Software des PC-Massenmarktes zurückgreifen zu können - sind offensichtlich so groß, daß Plattformen auf PC-Grundlage in vielen Bereichen die bisher typischen Sonderentwicklungen verdrängen - auch dort, wo besondere Forderungen hinsichtlich Leistungsfähigkeit und Zuverlässigkeit bzw. hinsichtlich geringster Kosten gestellt werden (Maschinensteuerungen, Telekommunikation, Unterhaltungselektronik und Haushaltsgeräte*)).

*): vor allem im Zusammenhang mit dem unvermeidlichen Internetanschluß (Fachbegriffe: Set-top Boxes (Vorsatzgeräte für Fernsehapparate) und Internet Appliances - z. B. Kühlschränke, die selbsttätig Whisky und Butter nachbestellen). Die Plattform ist typischerweise ein kleiner, aus wenigen Schaltkreisen bestehender PC. Als Prozessoren kommen u. a. “uralte” Typen der P5-Klasse zum Einsatz (vgl. Abbildung 1.4).

Hinweis:
Heutzutage ist es oftmals das einzig Vernünftige, gar keine Hardware zu entwickeln, sondern sich seine Plattform aus PC-Teilen zusammenzustellen (wenn es etwas Solideres sein soll, könnte man z. B. CompactPCI-Baugruppen nehmen). Ganz sorglos kann man damit aber auch nicht leben - denken wir nur an den Umfang und an die bekannte Zuverlässigkeit der gängigen Systemsoftware. Vor allem aber: der eigentliche PC-Markt ist im wörtlichen Sinne ein Massenmarkt, und zudem ein weltweiter. Wer einen Schaltkreis, ein Motherboard usw. entwickelt, hat wenigstens sechsstellige Stückzahlen im Sinn. Und er bringt - der Mode folgend - alle halbe Jahre etwas Neues heraus. Die paar hundert oder tausend Stück, die womöglich irgend ein Maschinenfabrikant abnimmt, interessieren dabei gar nicht... (Manche Anbieter lösen das Problem, die Ersatzteilversorgung auch über Jahre hinweg aufrechtzuerhalten, auf einfache Weise, nämlich durch entsprechende Vorratshaltung.) Hinzu kommt der Wechsel auf seiten der Software.

1.2.3. Selbstentwickelte Computersysteme

Derzeit wird wohl kaum jemand einen Prozessor von Grund auf selbst entwickeln*). Es kann aber gelegentlich noch sinnvoll sein, keine fertigen Plattformen zu beziehen, sondern nur die Schaltkreise, und das System nach eigenen Vorstellungen aufzubauen. (Szenarium: Massenfabrikation, wo es auf Minimierung der Hardwarekosten ankommt. Die Embedded Systems in Staubsaugern, Waschmaschinen, Autos usw. werden meist vom gegebenen Schaltkreis ausgehend in genauer Anpassung an den jeweiligen Einsatzfall entwickelt.)

*): wo es aber Sinn hat: Prozessoren für spezielle Verarbeitungsaufgaben (Signalverarbeitung, Kryptographie, Spracherkennung, Graphik usw.) bis hin zu Prozessoren mit dynamisch änderbaren Funktionseigenschaften (die von Zeit zu Zeit in Mode kommen).

1.2.4. Klassische Schaltungsentwicklung

Hierunter verstehen wir, daß für einen bestimmten Anwendungszweck eine Hardware von Grund auf entwickelt wird.

Standardisierte elementare Schaltfunktionen (Off-the-Shelf-Schaltkreise)
Schaltungen aus einfachen Elementen von Grund auf aufzubauen ist nach wie vor notwendig: zu Anpassungszwecken, für eher einfache Aufgaben (wofür ausgewachsene Computer zu teuer sind), für Einzelanfertigungen usw. Des weiteren finden wir solche Schaltungen natürlich auch in Funktionseinheiten und Systemen, die in großen Stückzahlen gefertigt werden. Entsprechende Schaltkreise können gleichsam sofort aus dem Regal (Off the Shelf; sprich: Off se Schellf) entnommen werden.

Elementare digitale Schaltkreise gibt es seit den 60er Jahren; es handelt sich also gleichsam um einen gesättigten Stand der Technik. Die Hersteller fertigen die Bauelemente, die nachgefragt werden. (Deshalb werden auch bestimmte “Uralt-Typen” noch in riesigen Stückzahlen hergestellt.)

Moderne Entwicklungen betreffen:

- verringerte Versorgungsspannungen (von 3,3 V an abwärts bis hin zu Werten unter 1 V),
- die Ausrichtung auf 2 Typensortimente: (1) Bustreiber und andere Interface-Koppelstufen, (2) Grundfunktionen zum Aufbauen von sog. Restlogik (Glue Logic; vgl. Abbildung 1.2),
- miniaturisierte Gehäuse (Anschlußabstände z. B. 1,25; 0,625; 0,5 und 0,4 mm),

Zum Integrationsgrad
Bei den hier in Rede stehenden Schaltkreisen unterscheidet man folgende Stufen:

- SSI (Small Scale Integration): ein Schaltkreis, der nur wenige Gatterfunktionen*) enthält (nach Texas Instruments: maximal 12 Gatterfunktionen). Die einzelnen Gatter oder Flipflops sind direkt an die Schaltkreisanschlüsse geführt (Zusammenschaltung zu komplizierteren Funktionen außerhalb des Schaltkreises).
- MSI (Medium Scale Integration): ein Schaltkreis, der eine mittlere Anzahl an Gatterfunktionen enthält (nach Texas Instruments: zwischen 13 und 99 Gatterfunktionen). Der Schaltkreis stellt typischerweise eine kompliziertere Funktion dar (Zähler, Decoder, Multiplexer usw.).
- LSI (Large Scale Integration): damit bezeichnen wir alle noch komplizierteren Schaltkreise (von 100 Gatterfunktionen an aufwärts).

*): ein "Gatter" - Gate - ist eine Schaltung, die eine elementare logische Verknüpfung realisiert. Mehr in Kapitel 2.

Programmierbare Schaltkreise
Auch programmierbare Schaltkreise können typischerweise aus dem Regal genommen werden (Off the Shelf). Ein solcher Schaltkreis ist an sich *universell* nutzbar. Er enthält keine bestimmten Funktionen, sondern vielseitig nutzbare Elementarstrukturen (Zellen, schaltbare Verbindungen usw.). Erst durch *Programmieren* erhält man die jeweils gewünschten Funktionen. Die Schaltkreise unterscheiden sich u. a. hinsichtlich der Auslegung, Anzahl und Anordnung der programmierbaren Strukturen (Tabelle 1.2), und hinsichtlich des Programmierverfahrens. Hierbei kommt es vor allem darauf an, in welcher Umgebung und wie oft der Schaltkreis programmiert werden muß bzw. kann (Tabelle 1.3).

Bezeichnung	Erklärung
Programmable Logic Device (PLD)	der übliche Oberbegriff für programmierbare Logikschaltkreise
Programmable Logic Array (PLA)	kombinatorische UND-ODER-Strukturen; sowohl UND als auch ODER programmierbar. Ein PLA-Schaltkreis enthält eine bestimmte Anzahl von UND-Gattern mit programmierbaren Eingängen sowie ODER-Gatter, die den UND-Gattern nachgeschaltet werden können. Diese Verbindungen sind ebenfalls programmierbar. Veraltet
Programmable Array Logic (PAL)	kombinatorische UND-ODER-Strukturen; UND programmierbar, ODER fest verschaltet. Ein PAL-Schaltkreis enthält eine bestimmte Anzahl von UND-Gattern mit programmierbaren Eingängen. Die UND-Gatter sind fest mit ODER-Gattern verbunden. PAL-Schaltkreise werden zumeist mittels Durchschmelzverfahren (Fusible Link) programmiert
Generic Array Logic (GAL)	elektrisch programmier- und löschbare, um Makrozellen (mit universell nutzbaren Flipflops) erweiterte PAL-Strukturen
EPLD	bezeichnet üblicherweise eine elektrisch (mehrmals) programmierbare und durch UV-Licht löschbare Schaltung
Complex Programmable Logic Device (CPLD)	üblicherweise faßt man unter diesem Begriff all das zusammen, was komplexer ist als eine einfache UND-ODER-Makrozellen-Struktur, aber nicht so komplex wie ein FPGA. (Solche Schaltkreise enthalten, verglichen mit GALs, (1) komplexere Makrozellen und (2) Koppelnetzwerke zum Verbinden der Zellen untereinander.)*)
Field Programmable Gate Array (FPGA)	Sammelbegriff für Schaltkreise, die es ermöglichen, eine Vielzahl programmierbarer Zellen freizügig untereinander zu verbinden, so daß - wenigstens näherungsweise - ein ähnlicher Grad der Schaltungsintegration erreicht werden kann wie bei Nutzung kundenspezifischer Gate-Array-Schaltkreise*)

*): ein CPLD hat vergleichweise wenige komplexe, ein FPGA vergleichsweise viele einfache Zellen

Tabelle 1.2 Programmierbare Schaltkreise

Programmierverfahren	Ausführung	Änderbarkeit	Bemerkungen
Maskenprogrammierung	beim Halb-leiterhersteller	im einzelnen Schaltkreis nicht mehr änderbar	nur bei extremen Stückzahlen von praktischer Bedeutung
Durchschmelzprinzip (Fuse)	beim Anwender*)	im einzelnen Schaltkreis praktisch nicht mehr änderbar	alle Verbindungen sind vor-gefertigt, die nicht benötigten werden beim Programmieren getrennt
Aufschmelzprinzip (Antifuse)	beim Anwender*)	im einzelnen Schaltkreis praktisch nicht mehr änderbar	alle Verbindungsstellen sind zunächst getrennt, benötigte Verbindungen werden beim Programmieren hergestellt

Programmierverfahren	Ausführung	Änderbarkeit	Bemerkungen
Ladungsspeicherung mit UV-Löschung	beim Anwender*)	durch Löschen und Neuprogrammieren	Löschen durch UV-Licht; erfordert Quarzglasfenster im Schaltkreis (es gibt auch preisgünstige Ausführungen ohne Fenster; diese kann man nicht mehr löschen)**)
Ladungsspeicherung mit elektrischer Löschung (EEPROM, Flash)	beim Anwender*)	durch Löschen und Neuprogrammieren (auch: in der Anwendungsschaltung (In System Programming)	Löschen durch elektrische Impulse
RAM-Zellen	beim Anwender*) bzw. während des Betriebs	durch Umladen; auch während des normalen Betriebs beliebig oft möglich	Halten der Information in Flipflops bzw. Latches; nach jedem Einschalten ist erneutes Laden erforderlich

*): Anwender = Hersteller des Gerätes bzw. der Funktionseinheit; **) Bezeichnung: OTP (One Time Programmable)

Tabelle 1.3 Programmierverfahren (Übersicht)

Anwendungsspezifische Schaltkreise (ASICs)

Geht es um höchsten Integrationsgrad, um höchste Geschwindigkeit und - vor allem - um hohe Stückzahlen, so sind anwendungs- bzw. kundenspezifische Schaltkreise (Application Specific Integrated Circuits ASICs) in Betracht zu ziehen. Im folgenden geben wir einen Überblick über die wichtigsten Prinzipien (Abbildung 1.3, Tabelle 1.4).

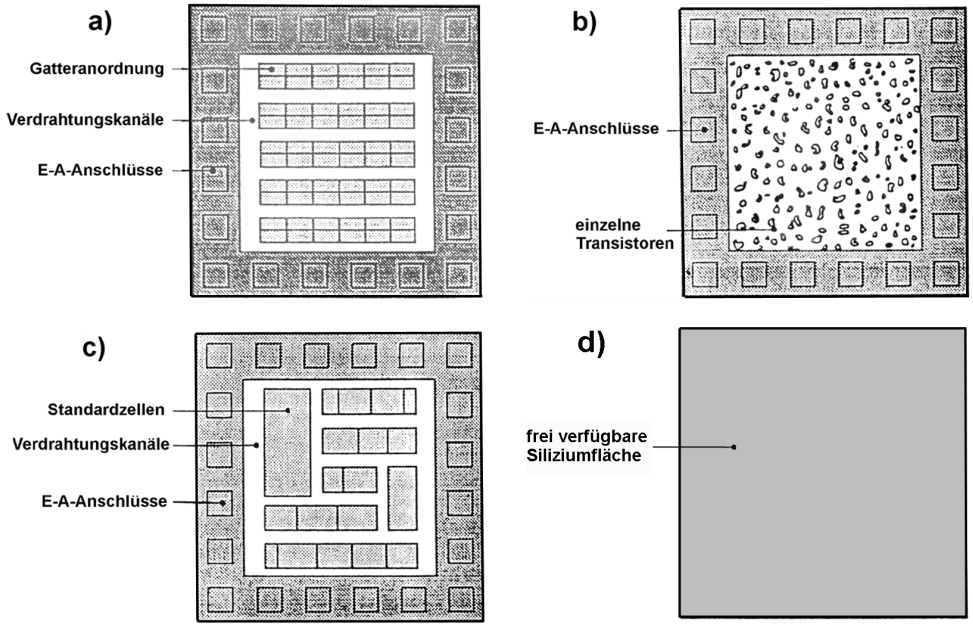


Abbildung 1.3 Gate-Array-Schaltkreise (Oki)

Gate Array (Abbildung 1.3a)
Der Schaltkreis hat eine Grundstruktur aus einzelnen Gattern, die in Form einer Matrix angeordnet sind. Diese Matrix ist am Schaltkreisrand von fest angeordneten Ein- und Ausgangsstufen umgeben. Zwischen den Gatter-Reihen sind Verdrahtungskanäle freigelassen, in denen die Gatter auf anwendungsspezifische Weise untereinander verbunden werden.

Sea of Gates (Abbildung 1.3b)
Die wörtliche Übersetzung gibt die Besonderheit nicht genau wieder: es handelt sich nicht um Gatter, sondern um eine große Anzahl von Transistoren, die am Schaltkreisrand von fest angeordneten Ein- und Ausgangsstufen umgeben sind. Besondere Verdrahtungskanäle sind nicht vorgesehen. Die Transistoren werden nach Bedarf zu Gattern und anderen Funktionseinheiten verschaltet. Verbindungsleitungen werden über Transistoren hinweggeführt (so überdeckte Transistoren können nicht funktionell ausgenutzt werden).

Standardzellenschaltkreise (Standard Cells; Abbildung 1.3c)
Diese Schaltkreise sind nicht von vornherein mit irgendwelchen Schaltelementen belegt. Vielmehr kann ein solcher Schaltkreis mit Funktionseinheiten (Macro- und Megazellen) gefüllt werden, die der Entwickler aus einer entsprechenden Zellenbibliothek auswählt. Die Ein- und Ausgangsschaltungen sowie Verdrahtungskanäle werden dabei gemäß der Zellenauswahl und -anordnung bestimmt. Als Zellen sind Gatter, Auswahlschaltungen (Multiplexer), Decoder, Zähler usw. bis hin zu kompletten Mikrocontrollern und Prozessoren nutzbar - vom 8051 bis zum Hochleistungsprozessor (Abbildung 1.4). Viele Zellenbibliotheken (Cell Libraries) enthalten Standardschaltungen, die den verbreiteten TTL- und CMOS-Baureihen entsprechen. Auch die Motherboard-Schaltkreise der PCs sind zumeist aus Standardzellen aufgebaut (DMA-Schaltungen, Zeitgeberschaltungen, Interrupt- Controller, serielle und parallele Schnittstellen usw.).

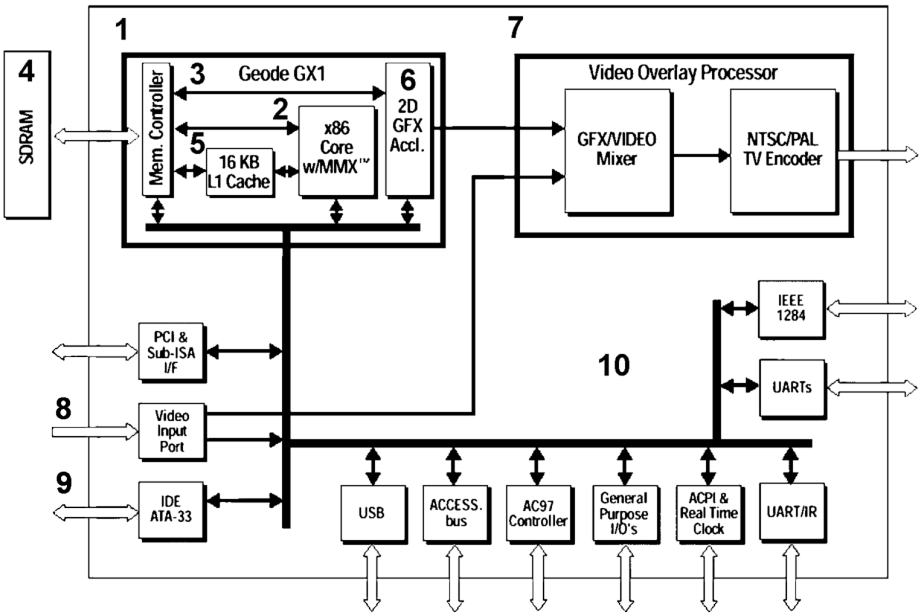


Abbildung 1.4 Auch ein Standardzellenschaltkreis (National Semiconductor)

Erklärung zu Abbildung 1.4:
Es handelt sich um ein (nahezu) komplettes System auf einem Schaltkreis (Geode SC1200). Anwendung: in Fernseh-Vorsatzgeräten (Set-top Boxes). 1 - Prozessormodul Geode GX1 (besteht seinerseits aus mehreren Standardzellen); 2 - der eigentliche Prozessorkern (P5-Klasse mit MMX, 266 MHz); 3 - Speichersteuerung; 4 - Arbeitsspeicher (SDRAMs); 5 - Cache; 6 - Graphikbeschleuniger; 7 - Videoüberlagerung und fernsehgerechte Codierung (NTSC/PAL); 8 - Videoeingang; 9 - Festplatteninterface; 10 - Anschlußsteuerungen für PC-typische Schnittstellen.

Full Custom (Abbildung 1.3d)
Bei solchen Schaltkreisen ist nichts vorgefertigt oder vordefiniert. Vielmehr wird der gesamte Schaltkreis von Grund auf für die jeweils gewünschte Funktion entwickelt.

	Gate Array	**Sea of Gates**	**Standard Cell**	**Full Custom**
Chip-Abmessungen	festgelegt (abgestuft mit unterschiedlicher Gatteranzahl)	festgelegt (abgestuft mit unterschiedlicher Transistoranzahl)	direkt von der Schaltung abhängig	direkt von der Schaltung abhängig
Anzahl der kundenspezifischen Masken	1 oder 2	2	voller Maskensatz (11...14); keine Vorfertigung möglich	voller Maskensatz; keine Vorfertigung möglich
Zeit für Fertigungsanlauf	Serie nach ca. 6 Monaten	Serie nach ca. 6 Monaten	Serie nach 8...12 Monaten	Serien nach 1½ Jahren
sinnvolle Stückzahlen*)	unter 30 000	unter 30 000	über 30 000	über 0,5 Mio
Bemerkungen		erlaubt bei gleicher Chipgröße höhere Komplexität als ein Gate Array	Flächenausnutzung besser (ca. 20%) als bei Gate Array; preisgünstiger	bestmögliche Flächenausnutzung (50 % besser im Vergleich zu Gate Array); bei hohen Stückzahlen am preisgünstigsten

*): siehe Hinweis im Text

Tabelle 1.4 ASIC-Prinzipien (Oki)

Hinweis:
Mit der Weiterentwicklung der programmierbaren Schaltkreise verschieben sich die Stückzahlen, bei denen sich der Übergang zum ASIC lohnt, weiter nach oben. Es gibt Hersteller, die auch für Stückzahlen von 500 000 und mehr programmierbare Schaltkreise bevorzugen, vor allem Typen, die man in der Anwendungsschaltung programmieren kann. Die Vorteile:

- programmierbare Schaltkreise ermöglichen Fertigung eines Hardware-Sortiments auf Grundlage einer einzigen "Plattform" (ein einziger Grund-Entwurf, womöglich sogar ein einziger Leiterplatten-Typ, der selektiv bestückt und programmiert wird),

- man kann auf dem Markt schnell reagieren (Änderungen, Modernisierungen, Ausbügeln von Fehlern, Anpassungen an neue Standards). Das betrifft vor allem die PC-Technik (Graphikkarten, Modems usw.).

1.3. Zugänge zur Digitaltechnik: Aussagenlogik und Boolesche Algebra

Zu den Grundlagen der binären Informationsverarbeitung gibt es zwei Zugänge: die klassische Aussagenlogik und die Boolesche Algebra.

Weshalb so theoretisch?
Um den Durchblick zu behalten. Natürlich könnte man auch auf rein intuitiver Grundlage entwickeln. Man würde aber damit nicht allzu weit kommen. Bereits die ersten Pioniere der Rechentechnik (u. a. Babbage und Zuse) hatten die Notwendigkeit erkannt, formalisierte Notationsweisen zu schaffen, denn es war kaum noch möglich, die Entwürfe mit herkömmlichen Dokumentationsmitteln (vor allem: in technischen Zeichnungen) in allen Einzelheiten darzustellen (zu aufwendig, zu kompliziert, unüberschaubar).

Ein Beispiel:
Stellen wir uns vor, wir seien Erfinder der alten Schule. Wir haben irgendwo gelesen, daß man Zahlen auch im Binärsystem darstellen kann. Nur zwei Werte: Null und Eins? - das schreit doch geradezu danach, Rechenmaschinen mit elektromagnetischen Relais aufzubauen. Somit könnte man die hochkomplizierten Mechanismen aus Zahnrädern, Zahnstangen usw. der bisherigen Rechenmaschinen durch eine Anzahl von Relais aus der Massenfertigung ersetzen. Fangen wir mit etwas Einfachem an: wie könnte man zwei Binärziffern zueinander addieren? Damit es ganz einfach wird, lassen wir einen ggf. einlaufenden Übertrag erst einmal weg. Die Rechenregeln sind in Tabelle 1.5 zusammengestellt.

Belegung	Summand A	Summand B	Summe S	Übertrag C
1	0	0	0	0
2	0	1	1	0
3	1	0	1	0
4	1	1	0	1

Tabelle 1.5 Einfachste Rechenregeln in einer Binärstelle: A + B = S + Ausgangsübertrag (C)

Die technische Darstellung der Null und der Eins ist naheliegend:

- Null = nicht erregtes Relais = keine Spannung = unterbrochener Stromweg,
- Eins = erregtes Relais = anliegende Spannung = geschalteter Stromweg.

Nun fangen wir mit dem Tüfteln an und nehmen uns die einzelnen Belegungen nacheinander vor:

- Belegung 1: alles Null, also passiert gar nichts. Trivial.
- Belegungen 2 und 3: ist A = 1, so ist das A-Relais erregt. Dies muß einen Stromweg zum Summenausgang S schaffen. Also ist ein Arbeitskontakt vorzusehen. Die gleiche Überlegung gilt für B = 1. Es ergeben sich also zwei parallelgeschaltete Arbeitskontakte (Abbildung 1.5a).
- Belegung 4: sind beide Relais A, B erregt, so muß die Summe Null werden, also muß der Stromweg zum Summenausgang S unterbrochen werden. Naheliegend: einen vom jeweils anderen Relais betätigten Ruhekontakt in die Stromwege gemäß Abbildung 1.15a einfügen. Zudem ist ein Stromweg zum Übertragsausgang C zu schalten. Die Lösung: zwei in Reihe geschaltete Arbeitskontakte (Abbildung 1.5b).

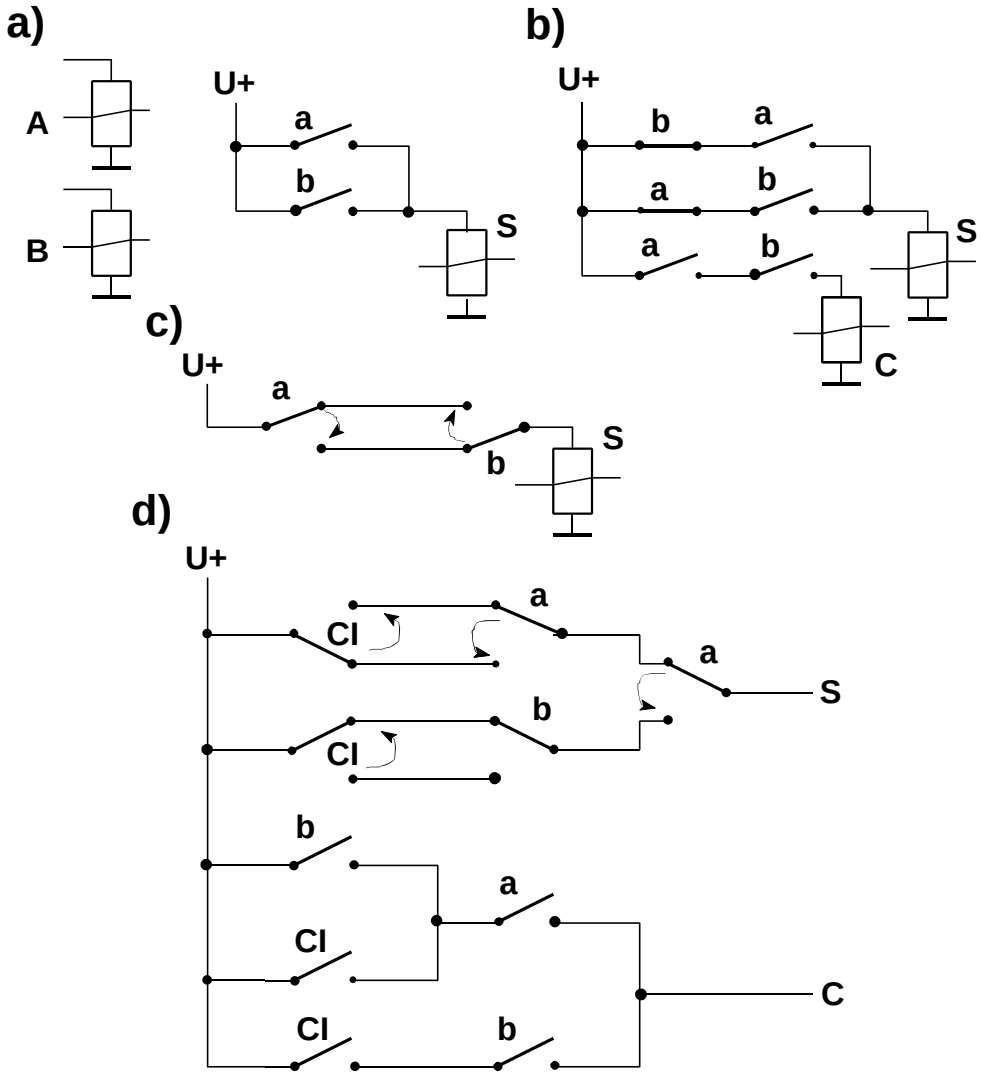


Abbildung 1.5 Wir erfinden ein binäres Addierwerk

Erklärung zuu Abbildung 1.5:
A, B - Summanden; S - Summe; C - Ausgangsübertrag; CI - Eingangsübertrag. Die Kästchen mit der geknickten Querlinie sind die Schaltsymbole der Relais. U+ - Versorgungsspannung. Kleinbuchstaben (a, b) kennzeichnen die zu den jeweiligen Relais gehörenden Kontakte. Alle Kontakte sind in Ruhelage dargestellt. a) - nur die Belegungen 2 und 3 werden berücksichtigt; b) - Belegung 4 wird eingearbeitet (1 + 1 ergibt S = 0, C = 1); c) - eine geniale Eingebung läßt uns darauf verfallen, die in Reihe geschalteten Kontakte a, b der Summenbildung durch Wechselkontakte zu ersetzen; d) - ein weiterentwickeltes Addierwerk, das auch den einlaufenden Übertrag (CI) berücksichtigt (die zugehörigen Relais sind hier nicht dargestellt).

Den Schritt von der Aufgabenstellung (Tabelle 1.5) zur Schaltung (Abbildung 1.5b, c) kann man durchaus noch nachvollziehen, Mit Abbildung 1.15d wird es schon schwieriger, und offensichtlich ist es hoffnungslos, auf die geschilderte Weise etwa einen ganzen Prozessor bewältigen zu wollen.

1.3.1. Aussagenlogik

Traditionell ist die Logik ein Teilgebiet der Philosophie. Die Aussagenlogik wiederum bildet gleichsam die Grundstufe der Logik. Sie beschränkt sich auf die Untersuchung von Aussagen, die nur wahr oder falsch sein können. Hierbei wurde erkannt, daß sich beliebig komplizierte Verknüpfungen von Aussagen auf wenige grundlegende Verknüpfungen (bzw. Aussagefunktionen) zurückführen lassen. Aus dem umgangssprachlichen Gebrauch heraus unmittelbar einsichtig ist die Rückführung auf folgende drei elementare Funktionen: UND (Konjunktion), ODER (Disjunktion), NICHT (Negation).

Konjunktion bzw. UND-Verknüpfung
Die Verknüpfung "Aussage 1 UND Aussage 2" ist nur dann wahr, wenn beide Aussagen wahr sind, in allen anderen Fällen ist sie falsch.

Disjunktion bzw. ODER-Verknüpfung
Die Verknüpfung "Aussage 1 ODER Aussage 2" ist dann wahr, wenn wenigstens eine der Aussagen wahr ist. Sie ist nur dann falsch, wenn beide Aussagen falsch sind.

Negation bzw. NICHT-Funktion
Die Negation einer Aussage ("NICHT Aussage") ist nur dann wahr, wenn die Aussage falsch ist und umgekehrt.

Diese Aussagen lassen sich in Form von Wahrheitstabellen vollständig darstellen (Abbildung 1.6). Eine Wahrheitstabelle enthält die Wahrheitswerte aller möglichen Aussagenkombinationen sowie den Wahrheitswert der resultierenden Aussage. Wie wir die beiden Wahrheitswerte bezeichnen, ist an sich gleichgültig. Wir haben in Abbildung 1.16 sowohl die Bezeichnungen F und W (für "falsch" und "wahr") sowie 0 und 1 gewählt.

UND-Verknüpfung (Konjunktion)

R = A UND B

A	B	R
F	F	F
F	W	F
W	F	F
W	W	W

A	B	R
0	0	0
0	1	0
1	0	0
1	1	1

ODER-Verknüpfung (Disjunktion)

R = A ODER B

A	B	R
F	F	F
F	W	W
W	F	W
W	W	W

A	B	R
0	0	0
0	1	1
1	0	1
1	1	1

NICHT-Funktion (Negation)

R = NICHT A

A	R
F	W
W	F

A	R
0	1
1	0

Abbildung 1.6 Wahrheitstabellen der elementaren Aussagefunktionen.
A, B - Aussagen; R - Resultat

1.3.2. Boolesche Algebra

George Boole hatte in der Mitte des 19. Jahrhunderts eine der elementaren Mathematik entsprechende symbolische Darstellung aussagenlogischer Funktionen angegeben. Hierzu hatte er die Wahrheitswerte mit 0 und 1 bezeichnet. Beispielsweise leuchtet die Analogie von Konjunktion und Multiplikation sofort ein:

$0 \cdot 0 = 0$; $0 \cdot 1 = 0$; $1 \cdot 0 = 0$; $1 \cdot 1 = 1$.

Ausgehend davon verwenden die Mathematiker Bezeichnungen wie Boolesche Algebra, Boolesche Variable, Boolescher Raum, Boolesche Funktion usw. In der modernen Mathematik werden Grundlagen allerdings kaum noch auf der unmittelbaren Anschauung aufgebaut. Der Zugang stützt sich vielmehr auf abstrakte Strukturen, die wir im folgenden kurz erläutern wollen.

Boolesche Algebren
Grundsätzlich ist eine “Algebra” definiert durch eine Menge und bestimmte Operationen über die Elemente dieser Menge. Die einer Booleschen Algebra zugrunde liegende Menge B hat nur 2 Elemente, nämlich 0 und 1: $B = \{0, 1\}$. Je nachdem, welche Operationen über die Elemente dieser Menge vorgesehen werden, ergeben sich verschiedene Boolesche Algebren. U. a. gibt es eine Boolesche Algebra, die der oben skizzierten klassischen Aussagenlogik entspricht. In ihr sind die Operationen UND, ODER und NICHT definiert.

Boolesche Räume
Einer Boolesche Variable kann jeweils eines der Elemente (0 oder 1) der Menge B zugewiesen werden. Wir betrachten k Boolesche Variable $a_1, a_2, ...a_k \in \{0, 1\}$. Insgesamt können k Variable 2^k mögliche Belegungen annehmen. Die Menge dieser Belegungen bildet einen Booleschen Raum B^k. Ein solcher Raum ist ein abstraktes Gebilde, das aus 2^k einzelnen (diskreten) Punkten besteht (Punktraum).

Boolesche Funktionen
Eine Boolesche Funktion ist eine Abbildung $B^k \rightarrow B$, die jedem Punkt des Booleschen Raumes B^k einen der Werte 0 oder 1 zuweist. Da es bei k Variablen 2^k Punkte gibt und jedem Punkt wiederum einer von zwei Wahrheitswerten zugeordnet werden kann, gibt es insgesamt:

2^{2^k} mögliche Boolesche Funktionen von k Binärvariablen.

Die Tabellen 1.6 und 1.7 geben alle Booleschen Funktionen von einer und von zwei Binärvariablen an. (Jede Zeile enthält die Wahrheitstabelle der betreffenden Funktion. Die Symbolik erläutern wir weiter unten in Tabelle 1.12 (Seite 40).

Funktionswerte bei		**Bezeichnung**
a = 1	**a = 0**	
0	0	Festwert 0
0	1	Negation ($\bar{a}$)
1	0	Identität (a)
1	1	Festwert 1

Tabelle 1.6 Alle 4 Booleschen Funktionen von einer Binärvariablen

Funktionswerte bei [a,b] =				Bezeichnung
1,1	**1,0**	**0,1**	**0,0**	
0	0	0	0	Festwert 0
0	0	0	1	NOR; $\overline{a \vee b}$
0	0	1	0	$\bar{a}$ & b; $\bar{a}b$ (Inhibition)[1)]
0	0	1	1	$\bar{a}$ (Negation)[2)]
0	1	0	0	a & $\bar{b}$; $a\bar{b}$ (Inhibition)
0	1	0	1	$\bar{b}$ (Negation)
0	1	1	0	Exclusiv-ODER; a $\oplus$ b (Ungleichheit, Antivalenz, XOR)[3)]
0	1	1	1	NAND; $\overline{a \,\&\, b}$; $\overline{ab}$[4)]
1	0	0	0	UND; a & b (Konjunktion, AND)
1	0	0	1	Äquivalenz; a $\equiv$ b (Gleichheit, Exclusiv-NOR, XNOR)[5)]
1	0	1	0	b (Identität, Pufferstufe)
1	0	1	1	$\bar{a} \vee b$ (Implikation a $\rightarrow$ b)[6)]
1	1	0	0	a (Identität, Pufferstufe)
1	1	0	1	$a \vee \bar{b}$ (Implikation b $\rightarrow$ a)
1	1	1	0	ODER; a $\vee$ b (Disjunktion, OR)[7)]
1	1	1	1	Festwert 1

1)...7): siehe die nachfolgenden Aussprachebeispiele

Tabelle 1.7 Alle 16 Booleschen Funktionen von zwei Binärvariablen

Wie sind die Tabellen zu lesen? - zwei Beispiele:

1. Negation (Tabelle 1.6): wir weisen dem Variablenwert 0 den Funktionswert 1 zu und dem Variablenwert 1 den Funktionswert 0,
2. ODER-Verknüpfung (Disjunktion; Tabelle 1.7): den Variablenwertkombinationen 1,1; 1,0 und 0,1 weisen wir jeweils den Funktionswert 1 zu. Der Variablenwertkombination 0,0 wird hingegen der Funktionswert 0 zugewiesen.

Aussprachebeispiele:

1) a quer b (auch: nicht a und b),
2) a quer (auch: nicht a),
3) a antivalent b (auch: a xor b),
4) a b quer (auch: a und b nicht; a nänd b),
5) a äquivalent b (auch: a xnor b),
6) a quer oder b (auch: wenn a, so b),
7) a oder b.

Wichtige elementare Verknüpfungen

Nicht alle Verknüpfungen der Tabellen 1.6 und 1.7 sind von gleicher Wichtigkeit. Im folgenden geben wir einen Überblick über jene Funktionen, die in der Digitaltechnik von besonderer Bedeutung sind. In Tabelle 1.8 sind die Wahrheitstabellen dieser Funktionen zusammengefaßt.

Negation (Verneinung, Invertierung)
Der Wahrheitswert wird invertiert. Aus 0 wird 1, aus 1 wird 0.

Identität (Bejahung)
Der Wahrheitswert der Variablen wird unverändert beibehalten. Diese (an sich triviale) Funktion ist technisch im Sinne eines Verstärkers bzw. Treibers nutzbar.

UND (Konjunktion, AND)
Die UND-Verknüpfung (sprich: Ännd) nimmt nur dann den Wahrheitswert 1 an, wenn alle Variablen den Wahrheitswert 1 haben. Wahrheitswert 0: es genügt, daß eine der Variablen den Wahrheitswert 0 hat.

ODER (Disjunktion, OR)
Die ODER-Verknüpfung nimmt dann den Wahrheitswert 1 an, wenn wenigstens eine der Variablen den Wahrheitswert 1 hat. Wahrheitswert 0: alle Variablen müssen den Wahrheitswert 0 haben.

NAND
Es ist eine negierte UND-Verknüpfung (NOT-AND; sprich: Nännd). Sie nimmt nur dann den Wahrheitswert 0 an, wenn alle Variablen den Wahrheitswert 1 haben. Wahrheitswert 1: es genügt, daß eine der Variablen den Wahrheitswert 0 hat.

NOR
Es ist eine negierte ODER-Verknüpfung (NOT-OR). Sie nimmt nur dann den Wahrheitswert 1 an, wenn alle Variablen den Wahrheitswert 0 haben. Wahrheitswert 0: es genügt, daß eine der Variablen den Wahrheitswert 1 hat.

Antivalenz (Exklusiv-ODER, XOR, Ungleichheit)
Diese Funktion hat immer dann den Wahrheitswert 1, wenn sich die Wahrheitswerte der beiden Variablen unterscheiden, das heißt, wenn entweder Variable a wahr und Variable b falsch ist oder umgekehrt (mit anderen Worten: bei Ungleichheit der Wahrheitswerte von a und b). Auf

Grund dieses Entweder-Oder-Verhaltens heißt die Funktion auch ausschließendes*) oder Exclusiv-ODER (XOR). Wahrheitswert 0: wenn beide Variablen den gleichen Wahrheitswert haben (0, 0 oder 1, 1).

*): die Belegung 1, 1 wird ausgeschlossen.

Äquivalenz (Exklusiv-NOR, XNOR, Gleichheit)
Diese Funktion hat immer dann den Wahrheitswert Eins, wenn die Wahrheitswerte der beiden Variablen gleich sind (0, 0 oder 1, 1). Wahrheitswert 0: wenn beide Variablen unterschiedliche Wahrheitswerte haben (0, 1 oder 1, 0).

Die Äquivalenz ist die Negation der Antivalenz und umgekehrt.

a	b	Negation*)	Identität*)	UND	ODER	NAND	NOR	Antivalenz	Äquivalenz
0	0	1	0	0	0	1	1	0	1
0	1	1	0	0	1	1	0	1	0
1	0	0	1	0	1	1	0	1	0
1	1	0	1	1	1	0	0	0	1

*): nur Variable a

Tabelle 1.8 Wahrheitstabellen wichtiger elementarer Verknüpfungen

1.3.3. Wie kann man mit "Logik" rechnen und steuern?

In der Fachsprache ist immer wieder die Rede von “logischen” Schaltungen und “Logik”signalen, von “wahr” (true; sprich: truh) und “falsch” (false; sprich: fohrls) usw. Aber nur wenige Entwurfsaufgaben haben offensichtlich etwas mit “Logik” im eigentlichen Sinne zu tun. Wie kann man mit Aussagenlogik (oder Boolescher Algebra) Information speichern, Befehle abarbeiten und Rechenoperationen ausführen? Tatsächlich handelt es sich um einen nicht besonders exakten Sprachgebrauch. Genaugenommen sind mehrere Ansätze im Verbund zu betrachten, die eines gemeinsam haben: das Prinzip der Zweiwertigkeit (Tabelle 1.9).

Wissensgebiet bzw. Gedankenkreis	zweiwertige Grundgrößen
Aussagenlogik	Wahrheitswerte “wahr” und “falsch”
Boolesche Algebra	Menge B = {0, 1}
binäre Zahlendarstellung	Ziffern 0 und 1
Programmsteuerung mit binären Steuerangaben in diskreten Zeitschritten	Steuerzustände “Ein” und “Aus” (Lochpositionen in Jacquardkarten, Nocken in Schaltwalzen usw.), Weiterschaltung in einzelnen Zeitschritten (Zahnradgetriebe, Schrittschaltwerke usw.)
technische Lösungen zum Verknüpfen und Speichern zweiwertiger Angaben	Ausnutzung verschiedener Wirkprinzipien mit nichtlinearem bzw. Schwellwertverhalten (Relais, Kippschaltungen, Transistor-Schaltstufen usw.)

Tabelle 1.9 Ansätze der zweiwertigen (binären) Informationsverarbeitung

Infolge des gemeinsamen Prinzips “Zweiwertigkeit” lassen sich verschiedenen Ansätze ineinander überführen bzw. nützliche Analogien ausnutzen (Tabelle 1.10).

Ausnutzung des Prinzips der Zweiwertigkeit	**Anwendung**
Abbildung der Binärzahlen 0 und 1 auf die Wahrheitswerte “falsch” und “wahr” bzw. in die Menge B = {0, 1}	▪ das numerische Rechnen (Addieren, Subtrahieren usw.) kann durch aussagenlogische bzw. Boolesche Funktionen beschrieben werden
Abbildung der Wahrheitswerte “falsch” und “wahr” bzw. der Elemente der Menge B = {0, 1} auf die Funktions- bzw. Schaltzustände zweiwertig arbeitender technischer Einrichtungen	▪ Schaffung technischer Einrichtungen, die die elementaren Verknüpfungen der Aussagenlogik bzw. der Booleschen Algebra verwirklichen, ▪ Realisierung des numerischen Rechnens auf Grundlage binärer Zahlendarstellungen, ▪ Realisierung von Informationsverknüpfungen, die sich auf umgangssprachlich-logische Zusammenhänge zurückführen lassen, ▪ Beschreibung und Optimierung der technischen Einrichtungen (Schaltalgebra)

Tabelle 1.10 Zur Ausnutzung des Prinzips der Zweiwertigkeit

Die Analogien bzw. wechselseitigen Überführungen sind nichts anderes als Vereinbarungssache*). Weitergehende Deutungen, wie sie die Umgangssprache gelegentlich nahelegt, führen gelegentlich zu irrtümlichen Auffassungen. So ist eine binäre Null nicht etwa falsch; ebensowenig ist eine binäre Eins wirklich wahr. Von Wahrheitswerten, von falschen und wahren Signalen usw. zu sprechen ist aber weithin üblich**). Deshalb müssen auch wir diesem Sprachgebrauch folgen.

*): es kann allerdings exakt bewiesen werden, daß diese Vereinbarungen tatsächlich berechtigt sind.

**): im Englischen tut man das besonders gern. Beim Übersetzen aufpassen - *false* heißt in unserem Zusammenhang nichts weiter als “logisch Null” oder “invertiert” oder “inaktiv”; es bedeutet keinesfalls “falsch” im Sinne von “inkorrekt” oder “fehlerhaft” (letzteres heißt im Englischen u. a. *erroneous* oder *faulty*).

Logik und Schaltungstechnik
Betrachten wir die Analogie anhand ganz einfacher Beispiele (Abbildung 1.7).

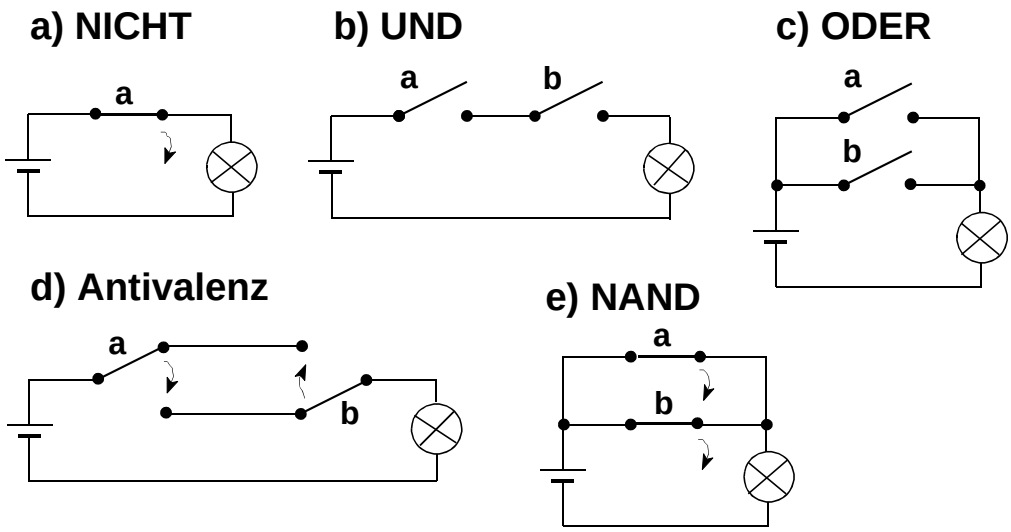


Abbildung 1.7 Kontaktschaltungen als technische Entsprechung elementarer Boolescher Funktionen (Auswahl)

Erklärung:
Es handelt sich um einfache Stromkreise aus Batterie (jeweils links) und Glühlampe (jeweils rechts), die Kontakte (Schalter, Taster, Relaiskontakte) enthalten. Die Wirkungsweise dieser Schaltungen kann durch elementare Boolesche Funktionen bzw. Aussagefunktionen beschrieben werden. Unsere Entsprechungen: nicht betätigter Kontakt = nicht leuchtende Lampe = Wahrheitswert Falsch = 0; betätigter Kontakt = leuchtende Lampe = Wahrheitswert Wahr = 1. Die Abbildung zeigt alle Stromkreise im Ruhezustand (alle Kontakte unbetätigt = Wahrheitswert Falsch = 0).

a) Modell der Negation (NICHT-Funktion). Stromkreis ist im Ruhezustand geschlossen; Lampe leuchtet. Betätigung des Kontakts trennt Stromkreis auf, so daß Lampe verlischt. Lampe leuchtet, wenn Kontakt a NICHT betätigt.

b) Modell der Konjunktion (UND-Verknüpfung). Um den Stromkreis zu schließen, sind beide Kontakte zu betätigen. Lampe leuchtet, wenn Kontakt a UND Kontakt b betätigt.

c) Modell der Disjunktion (ODER-Verknüpfung). Um den Stromkreis zu schließen, genügt es, einen der beiden Kontakte zu betätigen. Lampe leuchtet, wenn Kontakt a ODER Kontakt b (oder beide) betätigt.

d) Modell der Antivalenz. Der Stromkreis wird nur dann geschlossen, wenn entweder nur der Kontakt a oder nur der Kontakt b betätigt wird. Lampe leuchtet, wenn Kontakt a betätigt und Kontakt b nicht ODER Kontakt b betätigt und Kontakt a nicht.

e) Modell der NAND-Verknüpfung. Um den Stromkreis zu trennen (so daß die Lampe verlischt), sind beide Kontakte zu betätigen. Lampe leuchtet, wenn Kontakt a ODER Kontakt b (oder beide) NICHT betätigt; Lampe leuchtet NICHT, wenn Kontakt A UND Kontakt B betätigt..

Mit Logik rechnen
Nichts einfacher als das - wir müssen nur gleichsetzen: Binärziffer 0 = Wahrheitswert Falsch = "logisch 0", Binärziffer 1 = Wahrheitswert Wahr = "logisch 1".

Unser *Beispiel*
Gehen wir auf dieser Grundlage unser binäres Addierwerk (Seite 16) nochmals an (Tabelle 1.11).

Rechnung (A + B)	Summe S	Übertrag C
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

Tabelle 1.11 Rechnen in einer Binärstelle (ohne einlaufenden Übertrag)

Es ist sehr einfach: man sieht auf den ersten Blick (ggf. anhand des Vergleichs mit Tabelle 1.8), daß die Summe S der Antivalenz entspricht und der Ausgangsübertrag der Konjunktion:

- $S = A \oplus B$,
- $C = A \cdot B$.

Anwendungsprobleme lösen
Der Entwickler geht genauso vor wie wir es soeben kurz erläutert haben. Er ordnet den einzelnen Variablen Wahrheitswerte zu und versucht, sich die Zusammenhänge klarzumachen. Eine Möglichkeit, das Problem zu erfassen, besteht tatsächlich darin, in Listen oder Tabellen jeder möglichen Eingangsbelegung die jeweiligen Ausgangsbelegungen zuzuweisen (vgl. auch Abbildung 3.25). Alternativ dazu kann man versuchen, das, was eigentlich geleistet werden soll, unter Nutzung der Begriffe UND, ODER, NICHT usw. so genau wie möglich zu formulieren. Das führt letztlich zur Problembeschreibung durch Boolesche Gleichungen bzw. Schaltgleichungen.

Zur Technikgeschichte:
Was die Digitaltechnik und im besonderen den modernen Computer von den Grundlagen her ermöglicht hat, ist die Kombination folgender Gedankenkreise:

- der Aussagenlogik (Aristoteles),
- des formalen Rechnens mit Wahrheitswerten 0 und 1 (Boole),
- des binären Zahlensystems (Leibnitz),
- der Programmsteuerung (Jacquard, Babbage),
- der zweiwertigen Schaltelemente, zunächst auf mechanischer bzw. elektromagnetischer Grundlage (Zuse).

Daß man Relaisschaltungen und andere Anordnungen aus zweiwertig arbeitenden Funktionselementen mit den formalen Mitteln der Aussagenlogik beschreiben kann (Schaltalgebra), ist in den 30er Jahren etwa gleichzeitig von Konrad Zuse und Claude Shannon erkannt worden. Im Gegensatz zu Shannon (der an den Bell-Laboratorien tätig war) hat Zuse - als freier Erfinder - seinerzeit nichts publiziert, so daß die wissenschaftliche Priorität zumeist Shannon zugesprochen wird. Darüber hinaus hat Zuse die technische Anwendbarkeit des Binärsystems entdeckt: wenn es nur zwei Ziffern gibt, kann man jede Ziffer einem der beiden Wahrheitswerte gleichsetzen und so alle Rechenregeln auf Verknüpfungen von Wahrheitswerten zurückführen.

1.4. Schaltalgebra

Die Schaltalgebra läßt sich als technisch orientierte Abwandlung der Booleschen Algebra ansehen. Sie liefert Verfahren zur Realisierung von Schaltfunktionen mit einem gegebenen Bauelementesortiment, zur Aufwandsminimierung, zum Berechnen von Testbelegungen (um die Schaltungen prüfen zu können) usw.

1.4.1. Darstellung von Schaltfunktionen

"Schaltfunktion" ist nur eine andere Bezeichnung für Boolesche Funktion. Jede Schaltfunktion bezieht sich auf bestimmte binäre Variable (Eingangsvariable). Sie ordnet jeder Belegung dieser Eingangsvariablen einen der Wahrheitswerte 0 oder 1 zu (Funktionswert). Um diese Zuordnung darzustellen, gibt es verschiedene Möglichkeiten. Im besonderen haben sich Wahrheitstabellen, Schaltgleichungen, Belegungslisten, binäre Entscheidungsdiagramme und Schaltpläne bewährt

Variablenbezeichnungen
In der Theorie und beim Lernen ist Kürze von Vorteil. Deshalb werden Variable oft (so auch hier) mit einzelnen Buchstaben bezeichnet.

In der Schaltungspraxis hat man es hingegen meistens mit sehr vielen Variablen zu tun, und Eindeutigkeit ist das oberste Gebot. Anstelle von Buchstaben, wie a, b, usw. finden wir deshalb Variablenbezeichnungen wie DATA7, ADRS31, CARRY_IN, ShiftLeft usw., also eine Kennzeichnung, welche Bedeutung die einzelne Variable hat, manchmal aber auch "sinnlose" Kombinationen aus Buchstaben, Ziffern und Sonderzeichen (dann handelt es sich meistens um Bezeichnungen, die von der Entwurfssoftware erzeugt wurden).

Hinweise:

1. Oft wird in solchen längeren Variablenbezeichnungen auf Leerzeichen verzichtet. Der Grund: heutzutage läuft der Schaltungsentwurf weitgehend rechnergestützt ab. Zumeist verbieten die formalen Regeln Leerzeichen im einzelnen Namen. Aus Gründen der Übersichtlichkeit behilft man sich gelegentlich mit Bildungen wie COUNT_HOR_ADRS oder MemoryOutputEnable.
2. Gleichbedeutend zum Begriff "Binärvariable" spricht man in der Praxis auch vom Signalbezeichner (Signal Identifier), Anschlußbezeichner oder Leitungsbezeichner bzw. kurz von Signal, Anschluß oder Leitung.

Wahrheitstabellen

In einer Wahrheitstabelle (Truth Table; sprich: Truhs*) Tehbl) sind alle möglichen Kombinationen der Eingangsvariablen zusammen mit dem jeweiligen Funktionswert angegeben (Abbildung 1.8). Eine Wahrheitstabelle für n Eingangsvariable hat 2^n Einträge (Zeilen).

*): selbstverständlich dürfen Sie sich auch um eine wirklich korrekte Aussprache des Tieetsch bemühen...

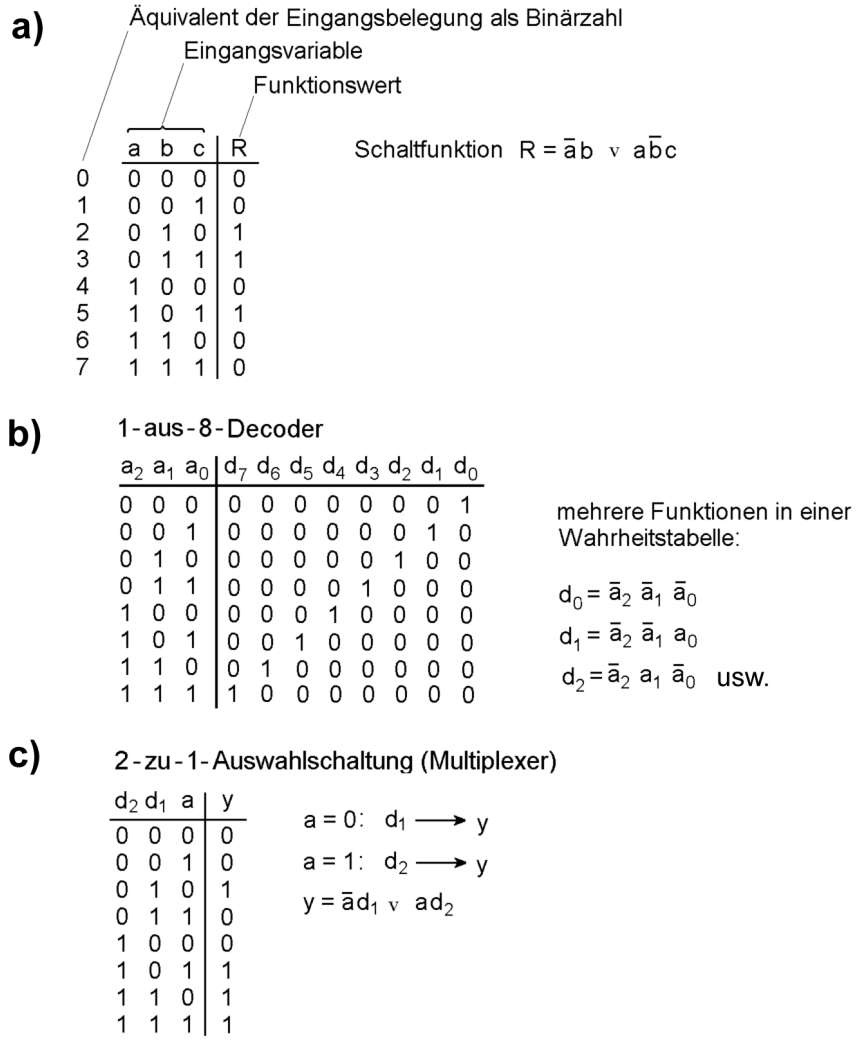


Abbildung 1.8 Wahrheitstabellen (Beispiele)

Erklärung:

a) grundsätzlicher Aufbau einer Wahrheitstabelle,

b) Wahrheitstabelle mit mehreren Funktionen, die von den gleichen Eingangsvariablen abhängen,

c) Wahrheitstabelle einer weiteren kombinatorischen Elementarschaltung.

Reihenfolge der Variablenbelegungen
Aus Gründen der Übersichtlichkeit sollten die Variablenbelegungen in regulärer Folge angegeben werden. Allgemein üblich ist die binäre Zählweise mit der Belegung 00...0 am Anfang und 11...1 am Ende, das heißt, die Belegung in der i-ten Zeile (i von 1 bis 2^n) entspricht der aus den Variablen gebildeten Binärzahl i-1. Bereich der Binärzahlen: 0 bis 2^n - 1 (vgl. Abbildung 1.8a).

Wahrheits- und Funktionstabellen in Datenblättern
Sie enthalten typischerweise weitere Symbole, die u. a. Signalflanken, wirkungslose Signale usw. kennzeichnen. Abbildung 3.4 zeigt ein Beispiel.

Schaltgleichungen (Boolesche Gleichungen)
In Schaltgleichungen sind die Variablenbezeichner mit Symbolen verknüpft, die die elementaren logischen Funktionen repräsentieren (Tabelle 1.12).

Funktion	Symbole	Beispiele
Konjunktion (UND, AND)	&	a & b
	*	a * b
	·	a · b
	∧	a ∧ b
	kein Symbol [1]	ab
Disjunktion (ODER, OR)	v	a ∨ b
	+	a + b
	#	a # b
Antivalenz (Ungleichheit, Exclusiv-ODER, XOR)	⊕	a ⊕ b
	≠	a ≠ b
Äquivalenz (Gleichheit, XNOR)	≡	a ≡ b
Negation (NICHT, Invert, NOT)[4)]	/	/a; /(a ∨ b)
	¬	¬ a
	'	a'; (a ∨ b)'
	–	$\bar{a}$; $\overline{a \vee b}$
	! [2)]	!a; !(a ∨ b)
	# [3)]	a#

1)...4): siehe Anmerkungen im Text

Tabelle 1.12 Symbole (Verknüpfungsoperatoren) in Schaltgleichungen

Anmerkungen zu Tabelle 1.12:

1) nur anwendbar, wenn 1 Zeichen je Variable,
2) typischer Negationsoperator in Schaltgleichungen, die programmierbare Logik (PLDs, GALs usw.) beschreiben bzw. die zum Eingeben in rechnergestützte Entwicklungssysteme bestimmt sind*),
3) bezeichnet ein invertiert wirkendes Signal (Negationssymbol, aktiv-Low-Anzeige),
4) bei den Negationszeichen ist gelegentlich achtzugeben, denn es gibt zwei Interpretationen:

   - es handelt sich lediglich um ein Negations*symbol* - also gleichsam nur um schmückendes Beiwerk (Beispiel: FRAME#)*),
   - es handelt sich um einen Negations*operator*. Und der zeigt an, daß tatsächlich negiert werden soll (Beispiel: !FRAME#)*).

*): vgl. Abbildung 3.8.

Belegungslisten
Es werden nur jene Variablenbelegungen in eine Liste eingetragen, die einen bestimmten Funktionswert (entweder 0 oder 1) ergeben (Abbildung 1.9).

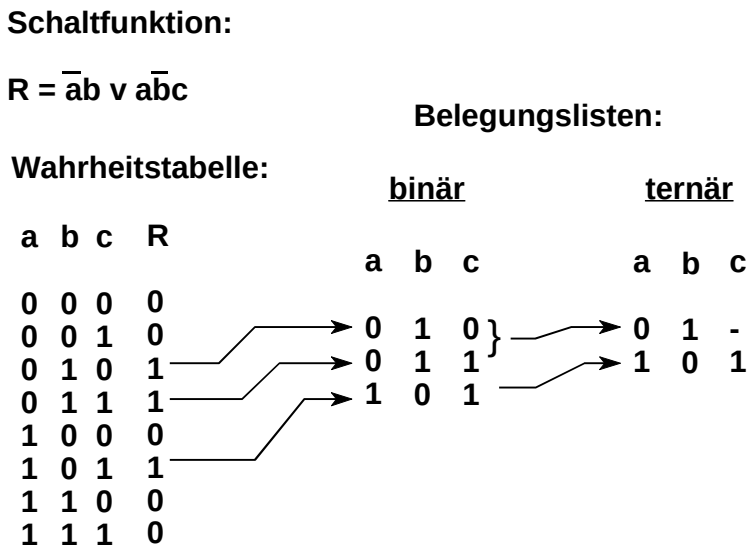


Abbildung 1.9 Schaltfunktion, Wahrheitstabelle und Belegungslisten

Erklärung:
Schaltfunktion und Wahrheitstabelle entsprechen Abbildung 1.8a. Die Belegungsliste des Funktionswertes R = 1 ist gleichsam ein Auszug aus der Wahrheitstabelle, wobei nur jene Variablenbelegungen berücksichtigt werden, die den Funktionswert R = 1 ergeben.

Es gibt 2 Arten von Belegungslisten:

- die *binäre* (zweiwertige) Belegungsliste: sie entspricht unmittelbar dem Auszug aus der Wahrheitstabelle. Ihre Einträge können somit nur die Werte 0 und 1 annehmen.
- die *ternäre* (dreiwertige) Belegungsliste: jeder Eintrag kann einen von 3 möglichen Werten annehmen: 0, 1 und - ("Strichelement").

Das Strichelement
Der Strich (-; gelegentlich auch ein **x** oder *****) steht gleichsam als Abkürzung für beide binären Werte 0 oder 1 (Abbildung 1.10).

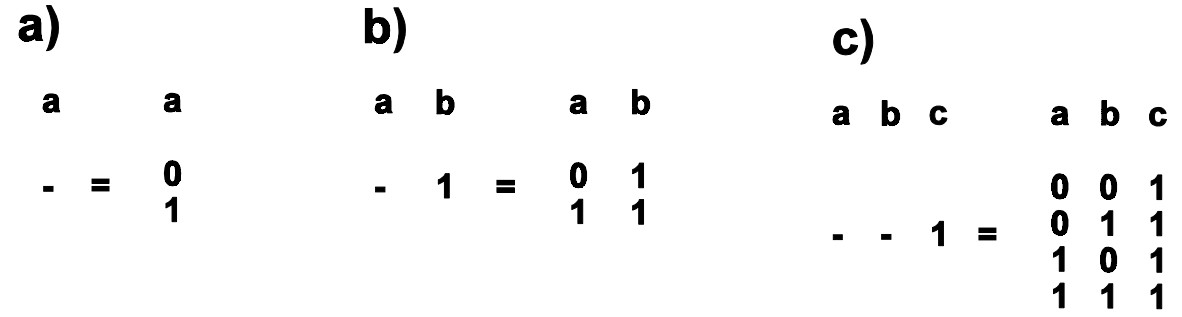


Abbildung 1.10 Die Bedeutung des Strichelements

Erklärung:
Die Abbildung zeigt 3 Belegungslisten:

a) nur eine Variable (a): das Strichelement steht für beide Belegungen 0, 1,
b) zwei Variablen (a, b): der eine Eintrag (- 1) steht für zwei binäre Einträge (0 1) und (1 1),
c) drei Variablen (a, b, c) mit 2 Strichelementen: da jedes Strichelement für beide Belegungen einer Variablen steht, ergeben sich bei 2 mit Strichelementen belegtenVariablen $2^2 = 4$ binäre Einträge: (- - 1) = (0 0 1), (0 1 1), (1 0 1), (1 1 1).

Grundsätzlich entspricht eine ternäre Variablenbelegung mit n Strichelementen 2^n binären Variablenbelegungen.

Abbildung 1.9 veranschaulicht, wie sich 2 binäre Variablenbelegungen in eine ternäre überführen lassen. Die beiden Belegungen (0 1 0) und (0 1 1) unterscheiden sich offensichtlich nur in einer Variablenposition (nämlich in c). Sie können somit zu (0 1 -) zusammengefaßt werden.

Die naheliegende Interpretation: eine mit einem Strich belegte Variable ist in der betreffenden Belegung bedeutungslos (sie muß deshalb gar nicht weiter beachtet werden - und wird deshalb im englischen Fach-Slang als *Don't Care* (sprich: Dohnt Kehr) bezeichnet).

Anwendung: zur *Minimierung* von Schaltfunktionen: eine Variable, die bedeutungslos ist, muß man auch bei der technischen Realisierung der Schaltfunktion nicht berücksichtigen.

Binäre Entscheidungsdiagramme

Ein binäres Entscheidungsdiagramm (Binary Decision Diagram BDD) ist ein gerichteter Graph, dessen Knoten Variable und dessen Kanten Variablenwerte repräsentieren (Abbildung 1.11).

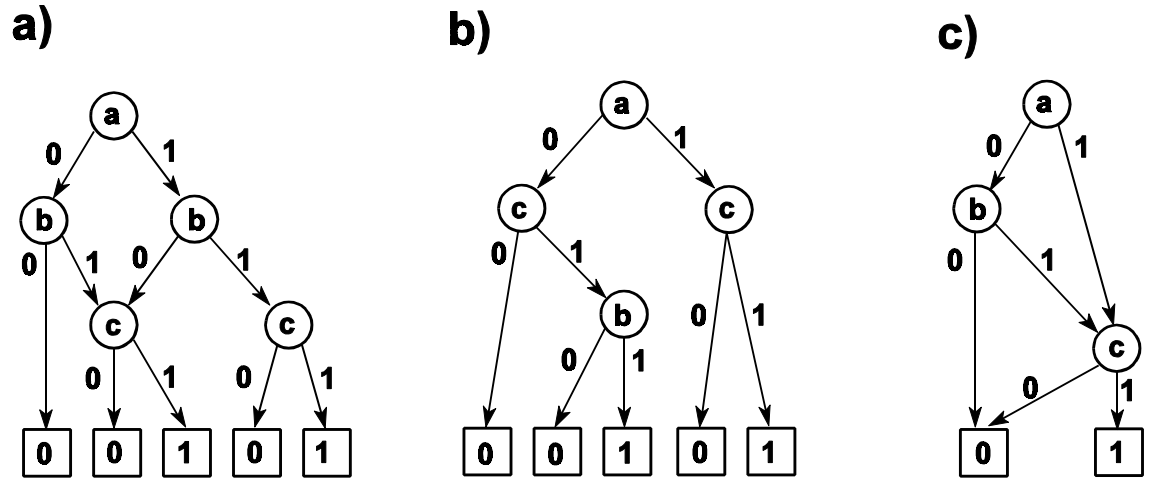


Abbildung 1.11 Binäre Entscheidungsdiagramme (BDDs)

Erklärung:

Alle 3 Entscheidungsdiagramme beschreiben die Schaltfunktion (a ∨ b) · c. Ein solches Diagramm ermöglicht es, den Funktionswert aus den Belegungen der einzelnen Variablen zu bestimmen. Es entspricht praktisch einer Art Programmablaufplan, in dem jeder Knoten die Abfrage einer Variablen mit Verzweigung in zwei Richtungen kennzeichnet. Beispiel (Abbildung 1.21a): wenn a = 0, so frage b ab; wenn b = 0, so ist der Funktionswert = 1, wenn b = 1, so frage c ab usw. Die Struktur eines solchen Diagramms hängt davon ab, in welcher Reihenfolge die Variablen abgefragt werden. Man spricht deshalb von *geordneten* Entscheidungsdiagrammen (Ordered Binary Decision Diagrams OBDDs).

a) OBDD mit der Variablen-Reihenfolge *a, b, c,*

b) OBDD mit der Variablen-Reihenfolge *a, c, b,*

c) *reduziertes* geordnetes Entscheidungsdiagramm (Reduced Ordered Binary Decision Diagram ROBDD) mit der Variablen-Reihenfolge a, b, c. “Reduziert” heißt hier, daß alle überflüssigen Knoten beseitigt sind.

Anwendung:

- zur rechnerinternen Darstellung von Schaltfunktionen (in Form von Listen- bzw. Zeigerstrukturen),
- zur technischen Verwirklichung von Schaltfunktionen. Der Vorteil: alle Schaltfunktionen können auf Grundlage einer einzigen Elementarfunktion aufgebaut werden, nämlich der 2-zu-1-Auswahl (in Abhängigkeit von einem Auswahlsignal wird eines von zwei Eingangssignalen zum Ausgang durchgeschaltet). In der Grundlagenforschung arbeitet man daran, bestimmte physikalische Effekte auszunutzen, um diese Elementarfunktion zu verwirklichen. Aber auch in eher herkömmlichen programmierbaren Schaltkreisen wird dieses Prinzip angewendet.

Schaltpläne
Schaltpläne sind zeichnerische Darstellungen, aus denen die Struktur der Schaltung hervorgeht. Näheres in Abschnitt 2.1. und in Kapitel 3.

1.4.2. Mit Schaltgleichungen rechnen

Im Service müssen Sie kaum selbst rechnen. Sie sollten aber in der Lage sein, Schaltgleichungen zu verstehen. Wichtige Fertigkeiten:

- das Hineindenken in Funktionszusammenhänge auf Grundlage von Schaltgleichungen (die oft so ähnlich aussehen wie jene in Abbildung 3.8),
- das Ausrechnen von Schaltfunktionen (wie im folgenden Praxisbeispiel beschrieben),
- die Wirkung von Festwerten erkennen (in der Praxis werden gelegentlich Schaltkreiseingänge mit Festwerten beschaltet),
- ein gewisses Verständnis elementarer Tricks (die vor allem aus der Ausnutzung der DeMorganschen Regeln beruhen.

Ein Praxisbeispiel
Eine Schaltfunktion ist in der Dokumentation folgendermaßen angegeben:

!RAS# = (MEM_ACCESS * RAM_DECODE * RAS_PULSE) + (REFRESH_ACCESS * REFRESH_CYCLE)

Sie messen folgende Werte: RAS# = 1; MEM_ACCESS = 1; RAM_DECODE = 1; RAS_PULSE = 1; REFRESH_ACCESS = 1; REFRESH_CYCLE= 0.

Ist die Schaltung in Ordnung? - Um dies beurteilen zu können, müssen Sie die Werte in die Gleichung einsetzen und den Funktionswert (für RAS#) ausrechnen. Es handelt sich um eine Disjunktion aus zwei Konjunktionen. Die erste Konjunktion ergibt den Wert 1. RAS# ist aber ein invertiertes Signal. Demgemäß müßte die Schaltfunktion den Wert 0 liefern. Die Schaltung ist also fehlerverdächtig.

1.4.3. Normalformen

Eine Normalform ist die Darstellung einer Schaltfunktion als Schaltgleichung, die nach bestimmten Regeln aufgebaut ist. In der Schaltalgebra kennt man verschiedene Normalformen. Jede beliebige Schaltgleichung ist in jeder Normalform darstellbar.

Hinweis:
Normalformen sind an sich etwas Akademisches. Schaltfunktionen, in die wir uns gelegentlich doch hineindenken müssen, sehen oftmals ziemlich “wild” aus (mit Klammern, mit negierten Klammerausdrücken usw.)[*)]. Die Bedeutung der Normalformen für den kleinen Mann der Schaltalgebra liegt in folgendem:

- im direkten Zusammenhang zwischen Normalform und zweistufigen Gatternetzwerken (UND-ODER-Strukturen). Anwendungen: (1) technische Verwirklichung von Schaltfunktionen mit Gatterschaltkreisen, (2) in programmierbaren Logikschaltkreisen.
- im Gebrauch einschlägiger Begriffe. So ist in Handbüchern, Anwendungsschriften usw. gelegentlich von Implikanden, von S.O.P-Netzwerken usw. die Rede.

Wir wollen deshalb nicht rechnen, sondern uns mit einem Überblick begnügen.

*): Beispiel: Abbildung 3.8.

Aufbau von Normalformen
Wir betrachten eine Boolesche Funktion, die von den Variablen $x_1, x_2,...x_n$ abhängt. Normalformen bestehen aus "Bausteinen" (Termen), die als konjunktive oder disjunktive Verknüpfung aller Variablen aufgebaut sind, wobei die Variablen entweder nicht negiert oder *einzeln* negiert auftreten:

Konjunktionsterme (Produktterme, Minterme):

$K_i = x_1 \& x_2 \&...\& x_n = x_1x_2...x_n$ ($x_1, x_2,...x_n$ einzeln negiert oder nicht negiert).

Beispiel: $K_i = \overline{x_1}\ \overline{x_2}\ x_3$. Unzulässig: $K_i = \overline{x_1x_2}x_3$.

Disjunktionsterme (Maxterme):

$D_i = x_1 \vee x_2 \vee...\vee x_n$ ($x_1, x_2,...x_n$ einzeln negiert oder nicht negiert).

Beispiel: $D_i = x_1 \vee \overline{x_2} \vee \overline{x_3}$. Unzulässig: $D_i = x_1 \vee \overline{x_2 \vee x_3}$

Mit diesen Termen lassen sich verschiedene Normalformen bilden. In der Praxis ist vor allem die disjunktive Normalform (DNF) von Bedeutung, gelegentlich auch die konjunktive Normalform (KNF):

- *disjunktive Normalform (DNF):* $f = K_1 \vee K_2 \vee...\vee K_n$
- *konjunktive Normalform (KNF):* $f = D_1 \& D_2 \&...\& D_n$

Summen und Produkte
Gelegentlich bezeichnet man - mit Bezug auf naheliegende Analogien - die Disjunktion als Summe und die Konjunktion als Produkt (vgl. Seite 29). Auch bestimmte symbolische Darstellungen (Tabelle 1.12) suggerieren diese Analogie (Disjunktion: $a + b$; Konjunktion: $a \cdot b$ oder ab).

Sum of Products (SOP oder S.O.P.) ist eine übliche Bezeichnung (sprich: Samm off Prodackts) für disjunktive Normalformen (= disjunktive Verknüpfung von Konjunktionen).

Product of Sums (POS oder P.O.S.) ist eine übliche Bezeichnung (sprich: Prodackt off Samms) für konjunktive Normalformen (= konjunktive Verknüpfung von Disjunktionen).

Produktterme (P-Terme), Minterme
Weitere Bezeichnungen für Konjunktionsterme. *Minterm* erklärt sich daher, weil es in einer disjunktiven Normalform (DNF, SOP) genügt, daß ein einziger Konjunktionsterm erfüllt ist, damit sich ein Funktionswert = 1 ergibt.

Maxterme
Eine weitere Bezeichnung für Disjunktionsterme. Sie erklärt sich daher, weil es in einer konjunktiven Normalform (KNF, POS) erforderlich ist, daß alle Disjunktionsterme erfüllt sind, damit sich ein Funktionswert = 1 ergibt.

Implikanden
Dies ist eine andere Bezeichnung für die Konjunktionsterme der disjunktiven und für die Disjunktionsterme der konjunktiven Normalformen.

Elementarkonjunktionen, Elementardisjunktionen
Dies sind Implikanden (Konjunktions- bzw. Disjunktionsterme), die *alle* Variablen (negiert oder nicht negiert) enthalten.

Kanonische Normalformen
Als "kanonisch" bezeichnet man Normalformen "im eigentlichen Sinne", nämlich solche, in denen *jeder* der Konjunktionsterme (DNF) oder der Disjunktionsterme (KNF) tatsächlich *alle* Variablen (negiert oder nicht negiert) enthält - mit anderen Worten: Normalformen, die ausschließlich aus Elementarkonjunktionen oder aus Elementardisjunktionen aufgebaut sind.

Reduzierte Normalformen
Eine Normalform heißt "reduziert", wenn *nicht* jeder der Konjunktionsterme (DNF) oder der Disjunktionsterme (KNF) *alle* Variablen (negiert oder nicht negiert) enthält (wenn also in wenigstens einem der Terme wenigstens eine Variable nicht vorkommt).

Hinweise:

1. Alle Schaltfunktionen lassen sich als kanonische Normalformen darstellen, aber nicht alle Schaltfunktionen als reduzierte.
2. Die Bezeichnung "reduziert" wird meistens weggelassen; man unterscheidet dann zwischen kanonischen und "gewöhnlichen" (nicht näher bezeichneten) Normalformen.

Disjunktive Normalformen (S.O.P.) und Belegungslisten
Die praktische Bedeutung disjunktiver Normalformen ergibt sich aus 2 Tatsachen:

1. sie hängen auf einfache Weise mit anderen Darstellungsformen zusammen (Abbildung 1.12),
2. sie entsprechen direkt zweistufigen UND-ODER- bzw. NAND-NAND-Schaltungsstrukturen.

Wahrheitstabelle:

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

Kanonische disjunktive Normalform (KDNF):

$R = \bar{a}\,b\,\bar{c} \vee \bar{a}\,b\,c \vee a\,\bar{b}\,c$

Belegungsliste (binär):

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

Disjunktive Normalform (DNF, S.0.P.-Darstellung):

$R = \bar{a}\,b \vee a\,\bar{b}\,c$

Belegungsliste (ternär):

a	b	c
0	1	–
1	0	1

Abbildung 1.12 Darstellungsweisen einer Schaltfunktion

Implikanden und Belegungen
Jeder Implikand einer disjunktiven Normalform entspricht direkt einer Variablenbelegung, die die Boolesche Funktion erfüllt:

- jede nichtnegiert vorkommende Variable entspricht einer Eins,
- jede negiert vorkommende Variable entspricht einer Null,
- jede fehlende (nicht vorkommende) Variable ist bedeutungslos (Don't Care) und entspricht einem Strichelement (-).

Beispiele:
Wir betrachten eine Funktion von 3 Variablen a, b, c.

1. Beispiel: $\bar{a}\ b\ \bar{c} \triangleq a = 0, b = 1, c = 0,$

2. Beispiel: $\bar{a}\ b \triangleq a = 0, b = 1, c = -.$

Umkehrung: Strichbelegungen können beim Bilden des Implikanden weggelassen werden.

Beispiel: - 01 $\triangleq \bar{b}\ c$.

Kanonische disjunktive Normalform aus Wahrheitstabelle
Wir suchen in der Wahrheitstabelle eine Zeile nach der anderen auf, in der das Ergebnis = 1 ist.

Die entsprechenden Variablenbelegungen werden in Elementarkonjunktionen gewandelt, die disjunktiv miteinander verknüpft werden.

Wahrheitstabelle aus kanonischer disjunktiver Normalform
Wir ordnen zunächst alle (2^n möglichen) Variablenbelegungen in Form einer Wahrheitstabelle an. Dann suchen wir in der Normalform eine Elementarkonjunktion nach der anderen auf und wandeln sie in eine Variablenbelegung um. Diese Variablenbelegungen suchen wir in der Wahrheitstabelle auf und setzen dort das Ergebnis auf Eins. Alle verbleibenden Variablenbelegungen (jene, die keiner der Elementarkonjunktionen entsprechen) belegen wir mit dem Ergebnis Null.

Belegungslisten und disjunktive Normalformen
Jeder Konjunktionsterm entspricht direkt einer Zeile der Belegungsliste (und umgekehrt). Die kanonische disjunktive Normalform (KDNF) entspricht einer binären, die gewöhnliche (reduzierte) DNF einer ternären Belegungsliste.

1.4.4. Schaltungsoptimierung

Optimierungskriterien
Es gibt ausgesprochen schaltalgebraische Optimierungskriterien und ausgesprochen praxisbezogene. Das klassische Optimierungsziel der Schaltalgebra ist die *Minimalform* der jeweils zu realisierenden Schaltfunktion, wobei es sich darum handelt, mit möglichst wenigen Verknüpfungen auszukommen, in die sowenige Variable wie möglich einbezogen sind. Praxisbezogene Optimierungsziele betreffen ganz allgemein Aufwendungen bzw. Kosten. So kann es wichtig sein, zwischen verschiedenen Steckkarten nur eine möglichst geringe Anzahl von Verbindungen vorzusehen, nur eine bestimmte Fläche auf einer Leiterplatte zu belegen, die Schaltfunktion in einem bestimmten CPLD oder FPGA unterzubringen usw., wobei es keine Rolle spielt, ob dabei Minimalformen im Sinne der Schaltalgebra verwirklicht werden oder nicht.

Die Optimierungen, mit denen wir es in der heutigen Praxis zu tun bekommen können, betreffen vergleichsweise einfache Funktionen in der Restlogik (Glue Logic). Die Optimierungsziele sind meist wirtschaftlicher Art: möglichst wenige Schaltkreise zu bestücken[*)], mit bestimmten Typen auszukommen, Fläche auf der Leiterplatte freizuhalten, Leiterzüge einzusparen usw. Hierbei verfallen die Entwickler gelegentlich auf die haarsträubendsten Tricks. Im folgenden wollen wir zwei oft verwendete Optimierungstricks kurz vorstellen.

*): was durchaus auch vorkommen kann: man setzt *mehr* Schaltkreise ein, als eigentlich nötig wären. Aber was bleibt anderes übrig, wenn man - beispielsweise - die gleiche Funktion einmal in der linken unteren Ecke des Motherboards und einmal in der rechten oberen Ecke braucht - ein quer über das ganze Board zu verlegender Leiterzug macht oft richtig Ärger (nicht selten gelingt es gar nicht, ihn zu verlegen), ein weiterer Schaltkreis hingegen kostet nur ein paar Cents...

Zwei typische Optimierungstricks

1. Herauslösen von Teilschaltfunktionen
Wir suchen nach Verknüpfungen von Variablen, die mehrfach vorkommen. Es liegt dann nahe, solche Verknüpfungen herauszulösen und jeweils nur einmal aufzubauen.

Beispiel:

$$f = a\,\overline{b}\,c\,d\,e \vee \overline{a}\,\overline{b}\,c\,d\,\overline{e} \vee a\,b\,\overline{c}\,\overline{d}\,e \vee a\,b\,\overline{c}\,d\,e \vee a\,b\,c\,\overline{d}\,e$$

Der geübte Entwickler erkennt sofort, daß die Verknüpfung *a b e* in 3 Implikanden vorkommt. Demzufolge läßt sich die Gleichung umformen:

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

Den eingeklammerten Ausdruck kann man vereinfachen, indem man dessen Negation (*c d*) verwirklicht und diese nochmals negiert:

$$f = a\,\overline{b}\,c\,d\,e \vee \overline{a}\,\overline{b}\,c\,d\,\overline{e} \vee a\,b\,e\,\overline{c\,d}$$

Es kann sein, daß weitere Umformungen zu weiteren Vereinfachungen führen (ob dies der Fall ist oder nicht, hängt vom jeweils verfügbaren Bauelementesortiment ab):

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

$$f = \overline{b}\,c\,d\,(\overline{a \oplus e}) \vee a\,b\,e\,\overline{c\,d}.$$

(Daß $(a\,e \vee \overline{a}\,\overline{e})$ eigentlich eine Äquivalenzverknüpfung darstellt, haben Sie sicherlich sofort erkannt... In den gängigen Baureihen gibt es aber keine Äquivalenz-, sondern nur Antivalenzgatter. Also nehmen wir ein solches und negieren dessen Ausgang.)

2. Nebenbedingungen (Don't Cares)
In die Schaltfunktion werden Signalbelegungen einbezogen, die in der Schaltung grundsätzlich nie auftreten können (Don't-Care-Belegungen). Eben weil sie nie auftreten können, schaden sie nichts - müssen also in den Schaltfunktionen auch nicht ausdrücklich ausgeschlossen werden. Daraus ergeben sich gelegentlich Vereinfachungen. Abbildung 1.13 zeigt ein Beispiel, das sich auf Abbildung 1.12 bezieht.

Ursprüngliche disjunktive Normalform:

$R = \bar{a}b \vee a\bar{b}c$

Nebenbedingung (Don't Care):

Der Term abc ist nie erfüllt (a, b, c sind nie gleichzeitig Eins).
Man kann also schreiben:

$R = \bar{a}b \vee a\bar{b}c \vee abc$

Belegungsliste (ternär):

a	b	c	
0	1	–	I
1	0	1	} Läßt sich zusammenfassen zu 1 - 1
1	1	1	}

Damit ergibt sich die minimierte DNF:

$R = \bar{a}b \vee ac$

Abbildung 1.13 Minimierung einer Schaltfunktion

Erklärung:
Wir gehen von der Schaltfunktion aus, die in Abbildung 1.22 dargestellt ist und nehmen an, die Umgebung, in der die Schaltung zum Einsatz kommt, wäre so beschaffen, daß die Belegung 1, 1, 1 nie auftritt. Dann können wir diese Belegung als Konjunktionsterm *a b c* (ausführlich geschrieben *a & b & c*) mit in die Schaltfunktion aufnehmen. Diese weitere Variablenbelegung (1 1 1) läßt sich mit der Variablenbelegung 1 0 1 zusammenfassen (zu 1 - 1), so daß sich der Term *a b c* zu *a c* vereinfachen läßt (eine 2-fache Konjunktion statt einer 3-fachen).

Probleme:

1. Ist eine solche Schaltung gegeben (z. B. beim Einarbeiten oder Fehlersuchen) und wissen wir nicht, welche Nebenbedingungen der Optimierung zugrunde gelegt wurden, so ist die Funktionsweise nicht immer leicht zu verstehen.
2. Wenn die betreffenden Nebenbedingungen nicht erfüllt sind (weil also z. B. Don't Cares tatsächlich am Schaltungseingang anliegen), verhält sich die Schaltung anders als erwartet - und gelegentlich richtig fehlerhaft.

Typische Optimierungsziele im Überblick
Worauf hin optimiert wird, hängt wesentlich von der jeweiligen Schaltungstechnologie ab:

Herkömmliche Technologie (SSI/MSI off the Shelf)
Wichtigstes Kriterium (Faustregel): sowenig Schaltkreisgehäuse wie möglich - gleichgültig, was sie enthalten (Minimum Package Count). Optimierungstricks (auch die soeben beschriebenen) werden ausgiebig angewendet.

CPLDs, FPGAs und ASICs:
Wichtigstes Kriterium (Faustregel): mit dem kostengünstigsten Schaltkreis auskommen (heißt typischerweise: (1) möglichst wenige Anschlüsse, (2) möglichst wenige Zellen). Haben wir einen Schaltkreistyp erst einmal ausgewählt, so haben wir dessen Zellen alle bezahlt, dürfen sie also auch ausnutzen.

Wichig ist, wie die Zellen aufgebaut sind und was sich in der einzelnen Zelle unterbringen läßt. Daraus ergeben sich besondere Optimierungsziele. Beispiel: CPLD Xilinx 9500: in eine einzige Zelle paßt jede Schaltfunktion, die - abhängig vom Schaltkreistyp - nicht mehr als 36 oder 54 Variable hat und deren DNF nicht mehr als 5 Konjunktionsterme (Implikanden) enthält. Unsere Optimierungsbeispiele betreffen weit weniger Variable und nicht mehr als 5 Konjunktionsterme. Die auf den Seiten 49 und 50 gezeigten Optimierungen sind hier also sinnlos. Die Bemühungen eines fleißigen Entwicklers, Teilschaltfunktionen herauszulösen, sind zumeist umsonst - denn die Entwurfssoftware wird typischerweise Schaltfunktionen, wie wir sie in unserem ersten Optimierungsbeispiel (S. 37) erhalten haben, wieder in zweistufige UND-ODER-Strukturen umformen.

2. Einführung in die Schaltungstechnik

2.1. Die technische Ausführung elementarer Schaltfunktionen

2.1.1. Schaltfunktion, Schaltelement und Schaltbild

Schaltfunktionen werden durch Schaltelemente technisch verwirklicht. Einfachste Schaltelemente sind *Negatoren* (Negators, Inverters) und *Gatter* (Gates; sprich: Gehts). Negatoren verwirklichen die NICHT-Funktion, Gatter verwirklichen elementare aussagenlogische Verknüpfungen. Die einfachsten Gatter haben zwei Eingänge und einen Ausgang. Wie Schaltelemente zusammengeschaltet sind, wird in *Schaltplänen* dargestellt.

Abbildung 2.1 veranschaulicht *Schaltsymbole* zur Darstellung von Negatoren und Gattern (die Symbolik sollte zunächst als gegeben hingenommen werden; Näheres in Abschnitt 3.1.1.). Zusammengesetzte Schaltungen entstehen, indem solche Schaltsymbole ein- und ausgangsseitig so verbunden werden, wie dies die jeweilige Schaltfunktion erfordert (Abbildung 2.2).

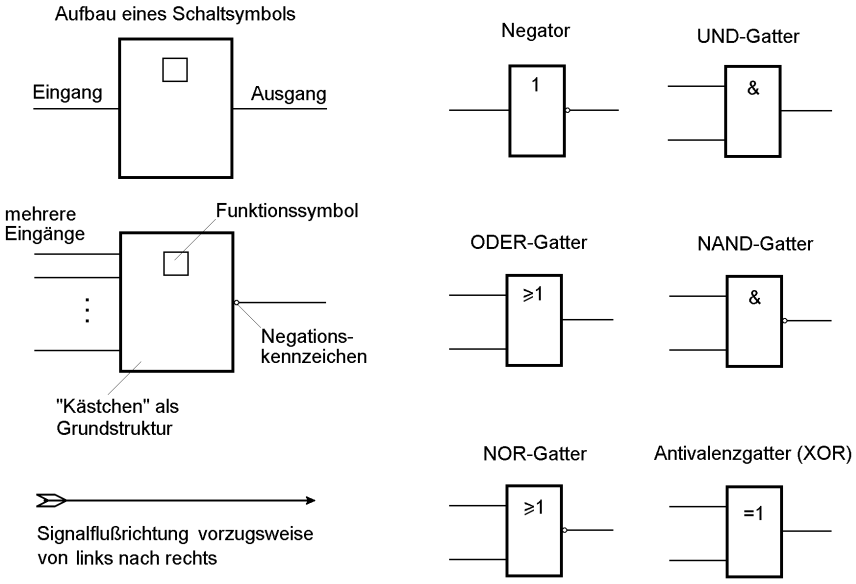


Abbildung 2.1 Schaltsymbole

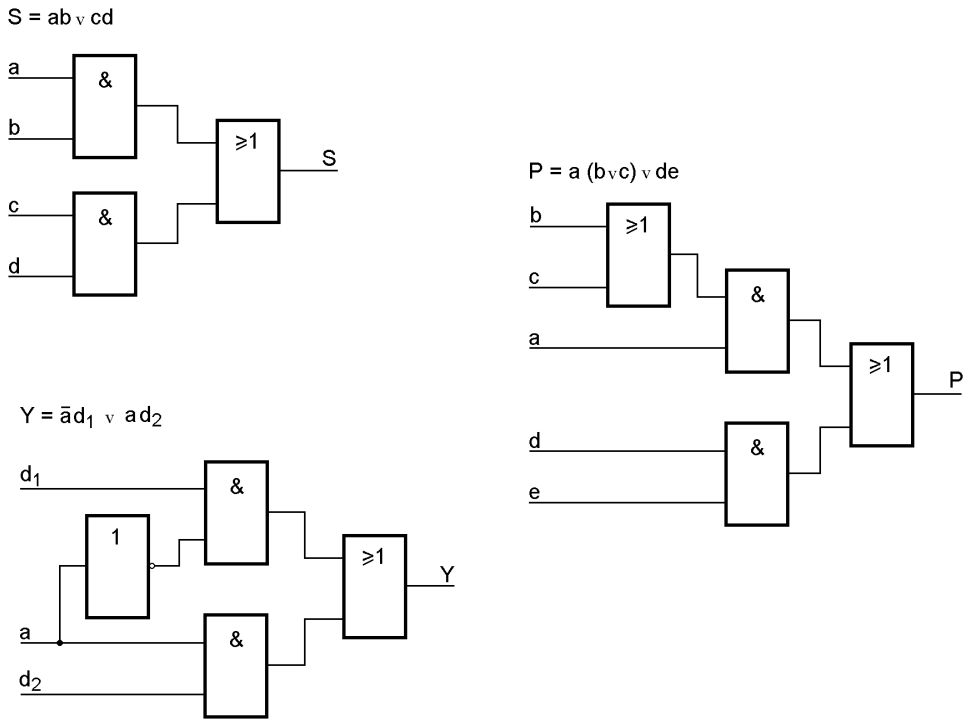


Abbildung 2.2 Beispiele einfacher Schaltpläne

2.1.2. Die vollständige Realisierungsbasis

Welche elementaren Schaltfunktionen sind notwendig, um aus einer Anzahl entsprechender Schaltelemente (Gatter) jede beliebige Schaltfunktion in Hardware ausführen zu können?

Jeder Satz elementarer Schaltfunktionen, der dies ermöglicht, heißt eine *vollständige Realisierungsbasis*.

Sicher ist, daß wir mit den drei Funktionen UND, ODER, NICHT auskommen. Jede andere Sammlung von Funktionen können wir daraufhin prüfen (ob sie eine vollständige Realisierungsbasis bildet oder nicht), indem wir versuchen, diese drei Elementarfunktionen damit aufzubauen.

Es gibt zwei Elementarfunktionen, die jeweils für sich allein als vollständige Realisierungsbasis ausreichend sind: NAND bzw. NOR. Abbildung 2.3 zeigt, daß man mit solchen Elementarfunktionen tatsächlich die Funktionen UND, ODER, NICHT verwirklichen kann.

Wir merken uns: mit nur einem einzigen Gattertyp - NAND oder NOR - kann man beliebig komplizierte logische Schaltungen aufbauen.

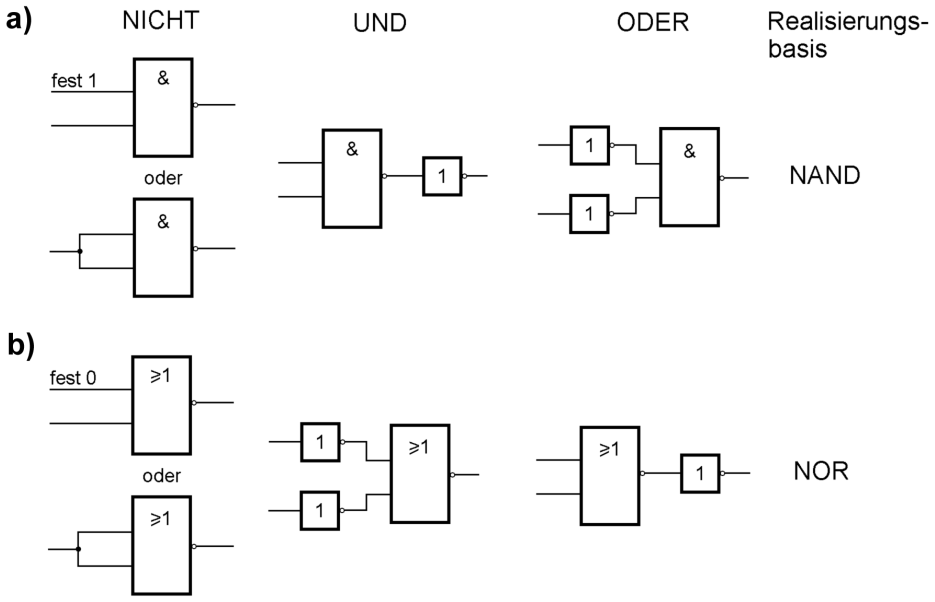


Abbildung 2.3 Elementare aussagenlogische Verknüpfungen mit NAND bzw. NOR. a) vollständige Realisierungsbasis mit NAND, b) vollständige Realisierungsbasis mit NOR

2.1.3. Gatter und Schalter

Wir haben an Beispielen gesehen, daß man mit ganz einfachen Kontakten logische Verknüpfungen realisieren kann (vgl. die Abbildungen 1.15 und 1.17). Zwischen Gattern und Kontaktanordnungen gibt es aber wichtige Unterschiede (Abbildungen 2.4 und 2.5, Tabelle 2.1).

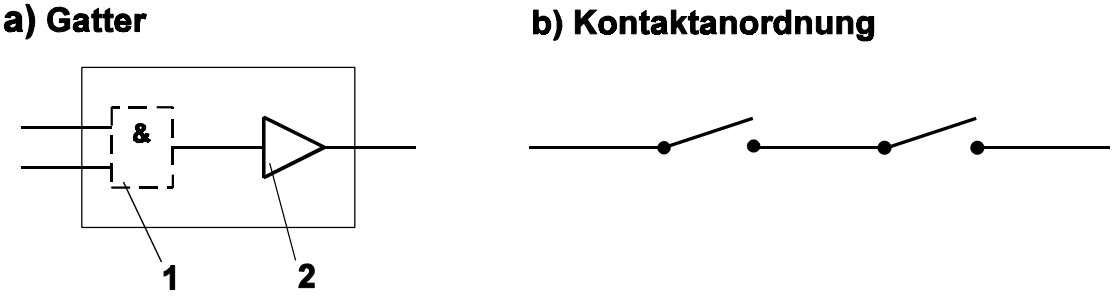


Abbildung 2.4 Gatter und Kontaktanordnung

Erklärung:
Beide Anordnungen leisten auf den ersten Blick das gleiche, nämlich (in unserem Beispiel) eine UND-Verknüpfung. Die Unterschiede ergeben sich aus dem jeweils grundsätzlich anderen Aufbau:

a) Gatter. Es handelt sich um elektronische Schaltungen, die aus der jeweiligen Belegung der Eingänge die entsprechende Ausgangsbelegung bilden. Gatter bestehen typischerweise aus der eigentlichen Verknüpfungsschaltung (1) und einer nachgeordneten Treiberstufe (2). Unser Beispiel (UND-Gatter): sind beide Eingänge mit Signalen belegt, die dem Wahrheitswert 1 entsprechen, so wird die Treiberstufe derart erregt, daß sie ein Ausgangssignal mit Wahrheitswert 1 abgibt. Signalflüsse in die Gegenrichtung (vom Ausgang zu den Eingängen) gibt es offensichtlich nicht: das Gatter ist rückwirkungsfrei.

b) Kontaktanordnung. Kontakte stellen Stromwege her oder trennen Stromwege. Es ist die Struktur (Topologie) der Kontaktanordnung, die die logische Funktion bestimmt. Unser Beispiel (UND-Verknüpfung): der Stromweg ist nur dann geschaltet, wenn der erste UND der zweite Kontakt geschlossen ist. In welche Richtung der Strom tatsächlich fließt, ist offensichtlich gleichgültig.

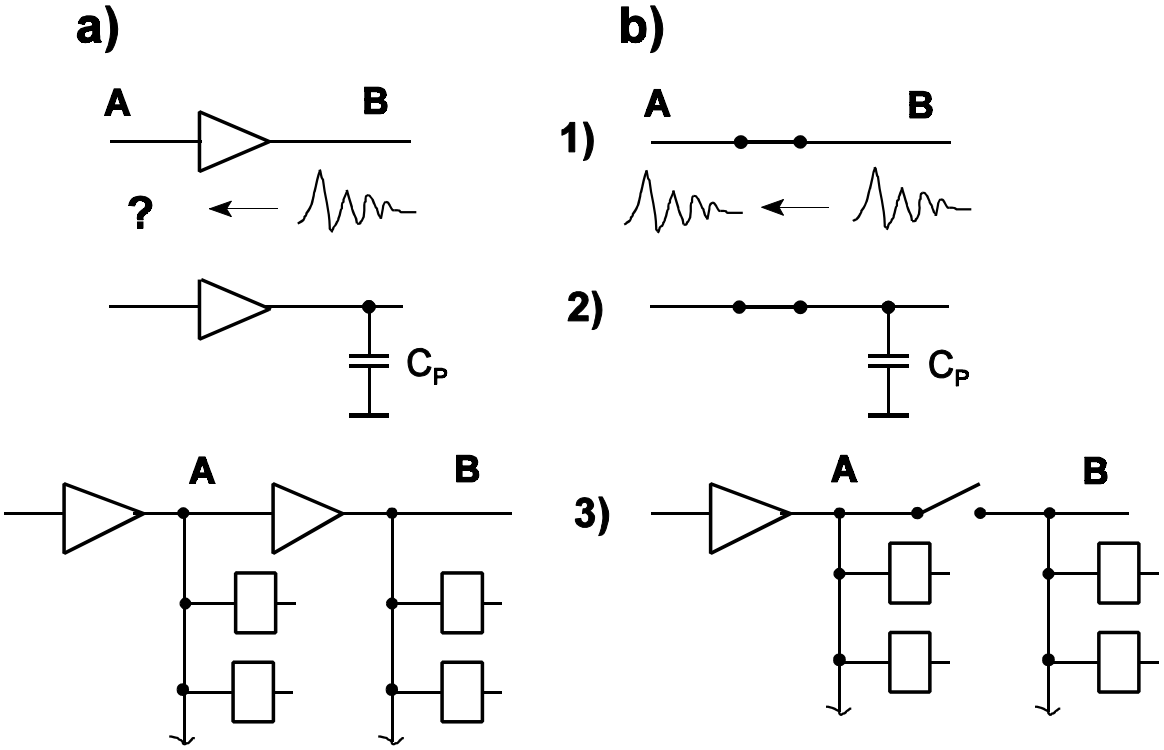


Abbildung 2.5 Gatter und Schalter im Vergleich

Erklärung:
Wir betrachten einen Signalfluß von Seite A nach Seite B.

a) eine Treiberstufe (z. B. die Ausgangsstufe eines Gatters) entkoppelt beide Seiten elektrisch voneinander (Seite B wird aktiv getrieben und wirkt somit nicht auf Seite A zurück).

b) ein geschlossener Schalter ist hingegen praktisch das gleiche wie ein Stück Draht:

   1) Seite A sieht alle Störungen auf Seite B,

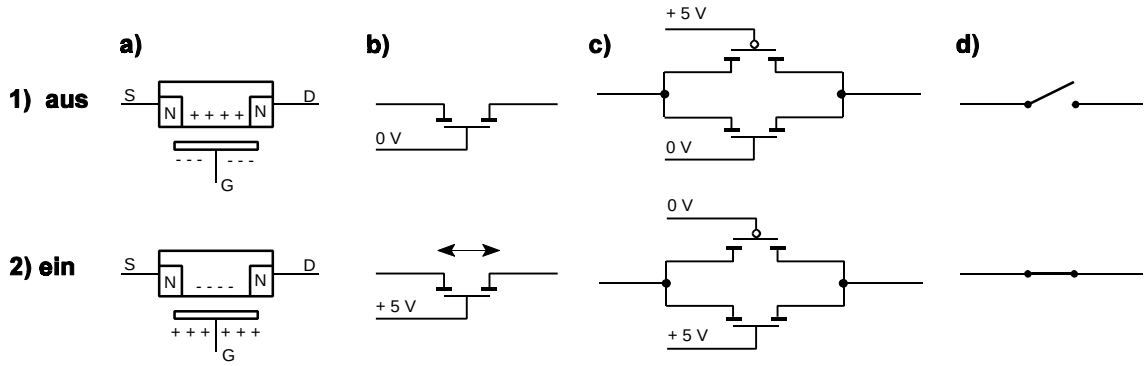
   2) Seite A sieht die parasitäre Kapazität C_p der Seite B,

   3) wird der Schalter geschlossen, so muß die Treiberstufe auf Seite A auch die Lasten von Seite B mittreiben.

Feldeffekttransistoren (FETs) als Schalter
Die Source-Drain-Strecke eines Feldeffekttransistors (FETs) wird leitend, wenn man an das Gate eine positive Spannung legt. Ein solcher Feldeffekttransistor verhält sich also wie ein Schalter bzw. ein Relais mit Arbeitskontakt (Abbildung 2.6). FET-Schalter gibt es in zwei Grundformen:

1. als sog. Transfer Gates im Innern hochintegrierter Schaltkreise,
2. als digitale Schalterbauelemente in Form von Busschaltern, Multiplexern usw. Derartige Bauelemente werden unter verschiedenen Handelsnamen angeboten (Crossbar, Quick-Switch usw.).

Abbildung 2.6 Der Feldeffekttransistor (FET) als Schalter



Erklärung:

a) Feldeffekttransistor als Schalter. S - Source, D - Drain, G - Gate. Der Zustand der Source-Drain-Strecke wird durch die Ladung auf dem Gate bestimmt: negative Ladung = aus, positive Ladung = ein.

b) Feldeffekttranistor im Schaltsymbol. Liegt eine positive Spannung am Gate, so wird die Source-Drain-Strecke leitend.

c) Transfer (Transmission) Gate in CMOS-Technologie. Wirkung ähnlich b), nur mit komplementärem Transistorpaar.

d) herkömmlicher Schalter (zum Vergleich).

Hinweis:
Solche Bauelemente spielen in manchen neumodischen Anwendungsfällen eine wichtige Rolle: Stromsparen, Austauschen von Steckkarten und Funktionseinheiten bei laufendem Betrieb (Hot Plugging), Abtrennen zeitweise nicht genutzter Busleitungen zwecks Verminderung der kapazitiven Belastung usw. Transfer Gates werden auch im Innern von Logikschaltkreisen verwendet.

Gatter	FET-Schalter
▪ aktive Ausgangsstufe, ▪ rückwirkungsfrei, ▪ längere Durchlaßverzögerung (typischerweise > 1 ns), ▪ benötigt vergleichsweise viel Siliziumfläche	▪ passive Durchreiche, ▪ nicht rückwirkungsfrei (sondern bidirektionaler Signalfluß), ▪ vernachlässigbare Durchlaßverzögerung (typischerweise < 250 ps), ▪ benötigt sehr wenig Siliziumfläche

Tabelle 2.1 Gatter- und Schalterbauelemente

2.2. Kombinatorische und sequentielle Schaltungen

Kombinatorische Schaltungen sind *speicher- und rückwirkungsfrei*. Sie werden durch Zusammenschalten von Gattern und Negatoren aufgebaut, wobei es nicht vorkommt, daß irgendein Gatterausgang auf Eingänge irgendeines vorgeschalteten Gatters zurückgeführt ist. Kombinatorische Schaltungen haben keine Speicherelemente, also kein Gedächtnis; wenn wir irgendeine Kombination von Eingangswerten anlegen, erhalten wir immer dieselbe Ausgangsbelegung.

Im Gegensatz dazu enthalten *sequentielle* Schaltungen *Speicherelemente bzw. Rückführungen*. Ihr Verhalten wird somit nicht nur von der jeweilige Eingangsbelegung, sondern auch von der "Vorgeschichte" abhängen. (Diese ergibt sich aus den vorausgegangenen Eingangsbelegungen.) Hier ist also das *Zeitverhalten* von Bedeutung. Elementare Speicherschaltungen heißen *Latches* und *Flipflops*. Eine solche Speicherschaltung kann jeweils ein Bit aufnehmen.

Eine bestimmte Belegung der Speichermittel einer sequentiellen Schaltung bezeichnen wir als deren *Zustand* (State). Die Funktion einer sequentiellen Schaltung wird im wesentlichen durch die *Zustandsübergänge* (State Transitions) bestimmt.

Abbildung 2.7 veranschaulicht den grundsätzlichen Unterschied zwischen kombinatorischen und sequentiellen Schaltungen.

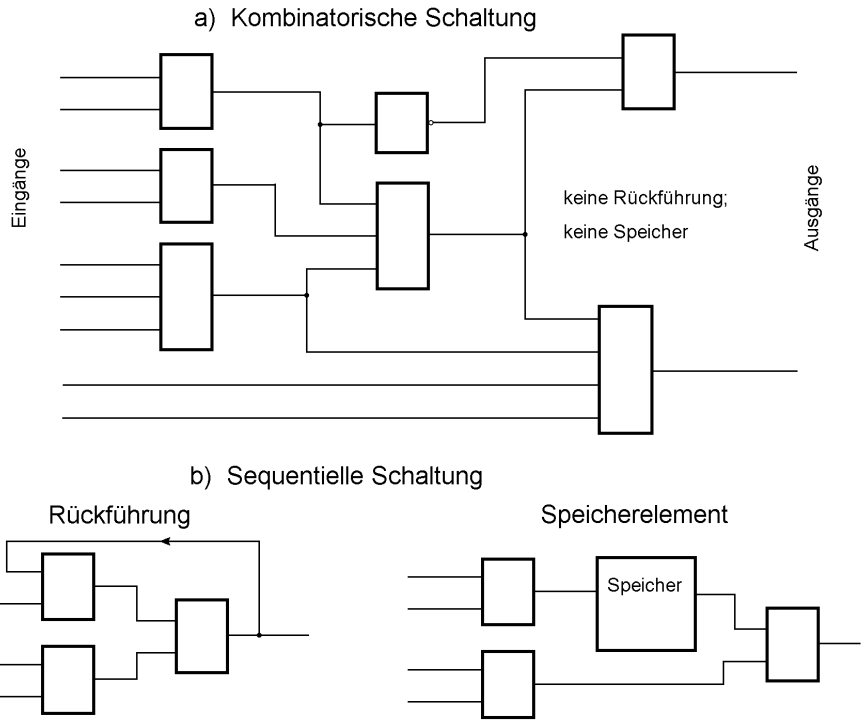


Abbildung 2.7 Kombinatorische und sequentielle Schaltungen

Register

Der Begriff gibt es sowohl in der Recherarchitektur als auch in der Digitaltechnik. Er hat in jedem Gebiet eine eigene Bedeutung:

a) Rechnerarchitektur: eine architekturseitig definierte Speicheranordnung, die programmseitig angesprochen werden kann,
b) Digitaltechnik: eine Aneinanderreihung gleichartiger Speicherelemente (Latches oder Flipflops) mit gemeinsamem Takt und gemeinsamen Steuersignalen.

RTL: Register + Kombinatorik

RTL = Register-Transfer-Ebene (Register Transfer Level). Auf dieser Ebene kann man eine Digitalschaltung in funktioneller Hinsicht vollständig dokumentieren. Hierzu fassen wir die Speicherglieder (Flipflops und Latches) zu Registern zusammen. Alle kombinatorischen Verknüpfungen werden funktionell beschrieben (mit Booleschen Gleichungen, Wahrheitstabellen o. dergl.). Wie diese Verknüpfungen im einzelnen (z. B. als Gatternetzwerk) realisiert werden, ist dabei gleichgültig. In der zeichnerischen Darstellung (Abbildung 2.7) sind die kombinatorischen Netzwerke lediglich Blocksymbole (Funktionszuordner). Komplizierte Digitalschaltungen werden typischerweise auf der RTL-Ebene entworfen (hierzu gibt es eigens Entwurfsbeschreibungssprachen). Der Entwickler beschreibt lediglich die Register und gibt die kombinatorischen Funktionen an. Die Umsetzung in Gatterstrukturen überläßt er der Entwurfssoftware.

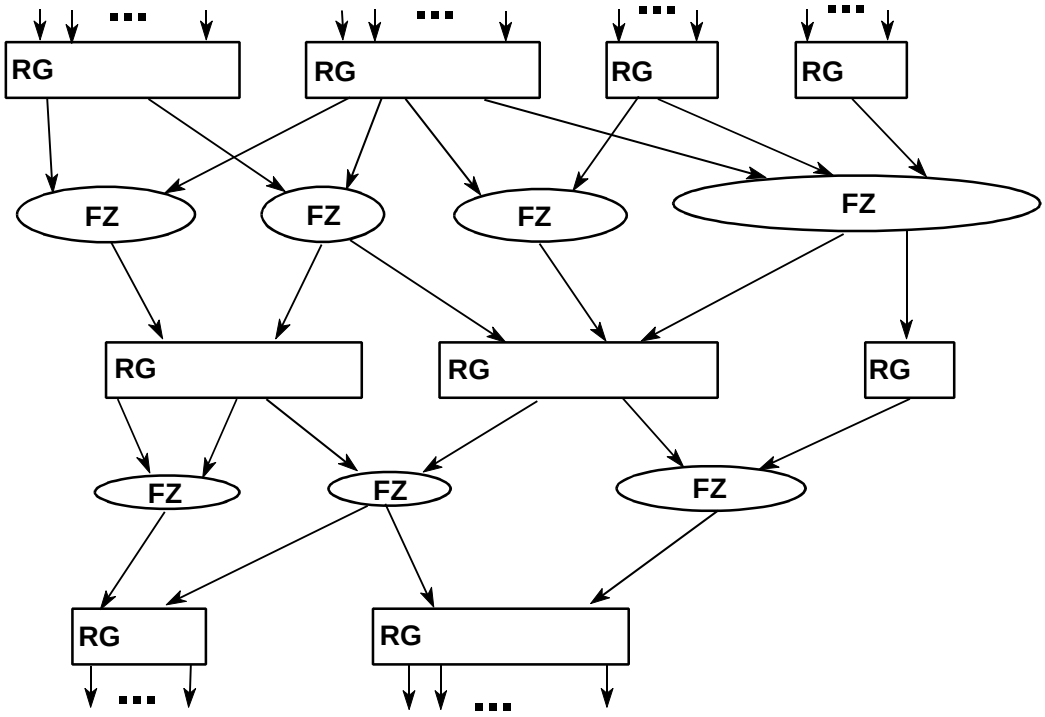


Abbildung 2.8 Zum Prinzip der RTL-Darstellung

Erklärung:
RG - Register; FZ - Funktionszuordner (Kombinatorik). In einer exakten RTL-Darstellung sind beschrieben: (1) alle Register bis auf's einzelne Flipflop, (2) alle Funktionszuordner durch die Schaltfunktionen, die die jeweiligen Beziehungen zwischen Eingangs- und Ausgangsbelegungen angeben (z. B. durch Boolesche Gleichungen oder Wahrheitstabellen).

State Machines
Die State Machine (sprich: Steht Mäschien) ist ein einleuchtendes, naheliegendes Modell für Steuerschaltungen, das auf den Grundlagen der Automatentheorie beruht.

Man geht von den funktionell notwendigen *Maschinenzuständen* (States) aus, ordnet jedem Zustand eine bestimmte Belegung der Speichermittel (Flipflops) zu (Zustandscodierung) und gestaltet die kombinatorischen Verknüpfungen so, daß sie zu jedem Taktzeitpunkt aus aktuellem Zustand und aktueller Eingangsbelegung den Folgezustand und die geforderte Ausgangsbelegung bestimmen (Funktionszuordner). Abbildung 2.9 zeigt den grundsätzlichen Aufbau einer solchen State Machine.

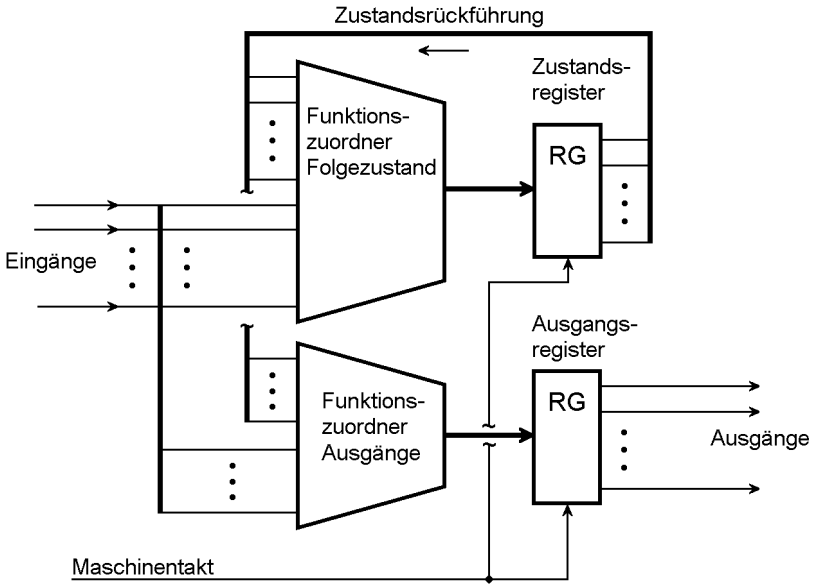


Abbildung 2.9 State Machine

Eine Tendenz im modernen Schaltungsentwurf besteht darin, Steuerschaltungen in überschaubare State Machines zu zerlegen und diese über Zustandsgraphen und die Schaltgleichungen der Funktionszuordner zu dokumentieren. Wir müssen uns also in derartigen Darstellungen zurechtfinden (vgl. die Abbildungen 1.7 bis 1.9).

2.3. Elementare Kennwerte

Wie andere Gebilde der Technik werden auch Digitalschaltungen (logische bzw. Logikschaltungen) durch zahlenmäßige Angaben genauer gekennzeichnet. Solche Wertangaben heißen allgemein Kennwerte oder Parameter.

Statische Kennwerte
Statische Parameter (engl. auch DC*) Parameters) sind *zeitunabhängige* Angaben. Dazu gehören die logischen Pegel, die Lastfaktoren, die Schaltungstiefe und die Anzahl der Zustände. Die Zeit spielt bei derartigen Werten keine Rolle.

Dynamische Kennwerte
Dynamische Parameter (engl. auch AC*) Parameters) sind *zeitabhängige* Angaben, wie Verzögerungszeiten und Schaltzeiten.

*): sprich: dieh-sie, eh-sie. DC = Gleichstrom (Direct Current), AC = Wechselstrom (Alternating Current.

2.3.1. Logische Werte und elektrische Pegel

Die beiden logischen Werte (Wahrheitswerte) wollen wir weiterhin mit 0 (falsch, invertiert bzw. negiert) und 1 (wahr) bezeichnen.

Beide logischen Werte müssen in der Praxis durch Werte einer elektrischen Kenngröße dargestellt werden. Diese Werte bezeichnet man als *Signalpegel* (logische oder Logikpegel bzw. kurz als Pegel, engl. Level (sprich: Läffl)).

Low und High
Die beiden Pegel werden üblicherweise mit *Low* (L; LO)*) und *High* (H; HI) bezeichnet. (Genauer gesagt: es gibt nicht zwei einzelne Werte, sondern zwei Wertebereiche bzw. Toleranzfelder; vgl. Abbildung 1.11.)

Mit dieser Bezeichnungsweise will man die absoluten Werte der betreffenden (physikalischen) Kenngröße zum Ausdruck bringen: *Low* ist der niedrigere Wert (die geringere Spannung oder der geringere Strom); *High* der höhere (die höhere Spannung, der höhere Strom). Ganz genau formuliert: der Wertebereich, der näher an -∞ (minus Unendlich) liegt, ist *Low*, jener, der näher an +∞ (plus Unendlich) liegt, ist *High*.

Aktiv - inaktiv
Ein Signal ist *aktiv* (erregt, asserted, valid, manchmal auch: true*)) wenn es seine jeweilige Wirkung ausübt, ansonsten ist es *inaktiv* (nicht erregt, in Ruhe, deasserted, invalid, manchmal auch: false*)). Findet die jeweilige Wirkung statt, wenn das Signal auf High-Potential liegt, so nennt man es "aktiv High", andernfalls "aktiv Low".

*): true und false erweisen sich hier als häßliche Angewohnheiten. Beispiel: die ATA-Standards. Dort setzt man true = asserted. Ein Signal, das aktiv Low wirkt, ist also bei 0-Belegung *true*, bei 1-Belegung *false*...

Die Wirkung kommt üblicherweise im Signalbezeichner - mehr oder weniger zutreffend - zum Ausdruck. So wirkt ein Signal MREQ als Speicheranforderung (Memory Request), ein Signal READY als Fertigmeldung, ein Signal GRANT als Bestätigung usw. Aktiv Low wirkende Signale werden in Schaltbild und Signalbezeichnung üblicherweise durch das jeweilige Negationssymbol (Tabelle 1.12) gekennzeichnet (ein aktiv Low wirkendes Speicheranforderungssignal heißt dann beispielsweise MREQ#).

Wirkungen in beiden Signalbelegungen (Low und High)
Das betrifft vor allem Auswahl- und Steuersignale. Ist ein solches Signal mit Low belegt, so hat es die eine Wirkung, ist es mit High belegt, die andere. Gelegentlich kommen in den Signalbezeichnern beide Wirkungen zum Ausdruck.

Hinweis:
In den Handbüchern der Schaltkreishersteller werden die jeweiligen Bezeichnungsregeln (Signal Conventions) erklärt. Beispielsweise wendet die Fa. Intel ziemlich konsequent die Regel an, in den betreffenden Signalbezeichnern *beide* Wirkungen auszudrücken; zuerst jene, die bei High, und dann jene, die bei Low auftritt.

Beispiel: was bedeutet *M/IO#*?
M steht für Speicherzugriff (Memory Access), *IO* steht für E-A-Zugriff (I/O Access). Das Negationssymbol nach IO kennzeichnet, daß diese Wirkung bei Low zustande kommt. Also bedeuten: M/IO# = 1: Speicherzugriff, M/IO# = 0: E-A-Zugriff.

Positive und negative Logik
Die beiden Wahrheitswerte 0 und 1 müssen den beiden Signalpegeln Low und High irgendwie zugeordnet werden. Hierfür gibt es zwei Möglichkeiten (Tabelle 2.2). Je nach Zuordnung spricht man von positiver oder von negativer Logik.

Signalpegel	Positive Logik	Negative Logik
Low	0	1
High	1	0

Tabelle 2.2 Positive und negative Logik

An sich kann man die Zuordnung beliebig treffen und auch - z. B. zu Optimierungszwecken - mitten in der Schaltung wechseln.

Hinweis:
Beim Wechseln gelten die DeMorganschen Regeln (Anhang): UND wird zum ODER, ODER zum UND; NAND zum NOR und NOR zum NAND.

Heutzutage vorherrschend: die positive Logik
Die positive Logik hat sich weitgehend durchgesetzt*). Wir werden deshalb ausschließlich die positive Logik verwenden.

*): die Entwicklungsgeschichte, die dazu geführt hat, in Stichworten: Silizum als vorherrschendes Halbleitermaterial → NPN-Transistoren sind die kostengünstigsten Bauelemente → deshalb positive Speisespannung → NANDs in positiver Logik sind die kostengünstigsten Gatterstrukturen → NAND-NAND entspricht UND-ODER (= naheliegende Realisierung kombinatorischer Verknüpfungen (DNF)). Diese Tradition wurde - von den 60er Jahren an - bis in die heutige Zeit weitergeführt.

2.3.2. Wichtige Kennwerte kombinatorischer Schaltungen

Abbildung 2.10 gibt einen Überblick über Parameter, die wir in Zusammenhang mit kombinatorischen Schaltungen unbedingt benötigen.

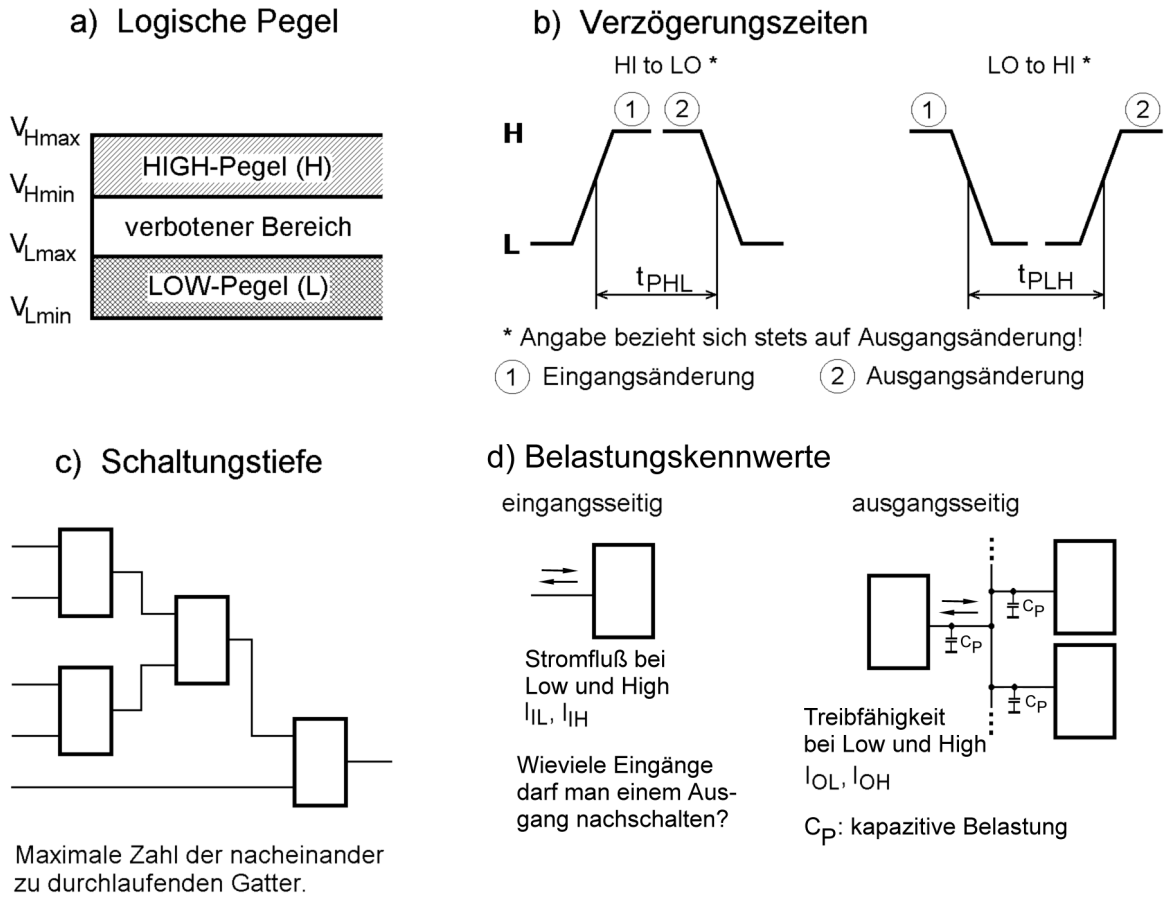


Abbildung 2.10 Wichtige Kennwerte kombinatorischer Schaltungen

a) Logische Pegel
Für die Pegel *Low* und *High* ist jeweils ein Toleranzfeld definiert. Beide Toleranzfelder sind durch einen verbotenen Bereich voneinander getrennt. Jedes Toleranzfeld wird durch einen Kleinstwert (Minimalwert) und einen Größtwert (Maximalwert) gekennzeichnet. Die Pegelangaben sind statische Parameter. Sie werden wesentlich von der Versorgungsspannung und von der technologischen Grundlage der Schaltkreise bestimmt. Seit längerem ist eine Versorgungsspannung von (+) 5 V allgemein üblich. Für moderne stromsparende Systeme bevorzugt man geringere Spannungswerte (von 3,3 V an abwärts).

b) Verzögerungszeiten
Wenn wir an den Eingängen einer kombinatorischen Schaltung nichts ändern, wird auch am Ausgang keine Änderung auftreten. Eine derart betriebene Schaltung wäre jedoch ziemlich nutzlos. In der Praxis haben wir es also mit Signalbelegungen zu tun, die sich zeitlich ändern (von Low nach High oder umgekehrt). Die Zeit, die vergeht, bis sich eine Signaländerung an einem Eingang durch eine Signaländerung am Ausgang bemerkbar macht, bezeichnen wir als Verzögerungszeit (Propagation Time t_p, Propagation Delay). Für das einzelne Gatter liegt sie üblicherweise im Bereich einiger Nanosekunden. Die Verzögerungszeiten sind dynamische Parameter.

c) Schaltungstiefe
Die Schaltungstiefe s ist eine Zahl, die angibt, wieviele Gatter nacheinander durchlaufen werden müssen, bis sich eine Eingangsänderung am Ausgang auswirkt. Dabei wird der ungünstigste Fall betrachtet. Im Beispiel ist s = 3. Bestimmung: abzählen, wieviele Gatter zwischen Ein- und Ausgang hintereinandergeschaltet sind. Hierbei ist jeder Eingang zu berücksichtigen. Die größte dieser Zahlen ergibt die Schaltungstiefe.

Durchlaufverzögerung
Ein Gatternetzwerk der Schaltungstiefe s hat - roh gerechnet - eine Durchlaufverzögerung von $s \cdot t_p$. Die Verbindungen zwischen den Gattern können wir zunächst vernachlässigen (Überschlagsrechnung). Bei extremen Anforderungen sind solche Vereinfachungen allerdings nicht mehr zulässig. (Das betrifft u. a. die modernen Bussysteme und Speichersubsysteme.)

d) Belastungskennwerte
Diese Angaben ermöglichen es, zu bestimmen, wieviele Eingänge (nachfolgender Schaltkreise) man einem bestimmten Ausgang nachschalten darf. Sie betreffen die von den Eingängen bewirkten Stromflüsse, die Treibfähigkeit der Ausgänge und die kapazitive Belastung.

2.3.3. Wichtige Kennwerte von Impulsen und sequentiellen Schaltungen

Impulse
Als *Impuls* bezeichnen wir einen Signalverlauf, der von einem logischen Wert (einem Signalpegel) zum jeweils anderen logischen Wert übergeht und nach einer gewissen Zeit (der *Impulsdauer*) zum ursprünglichen Logikpegel zurückkehrt. Abbildung 2.11 veranschaulicht Impulsverläufe und wichtige Kennwerte in der Gegenüberstellung von tatsächlichem Signalverlauf und linearisierter Darstellung.

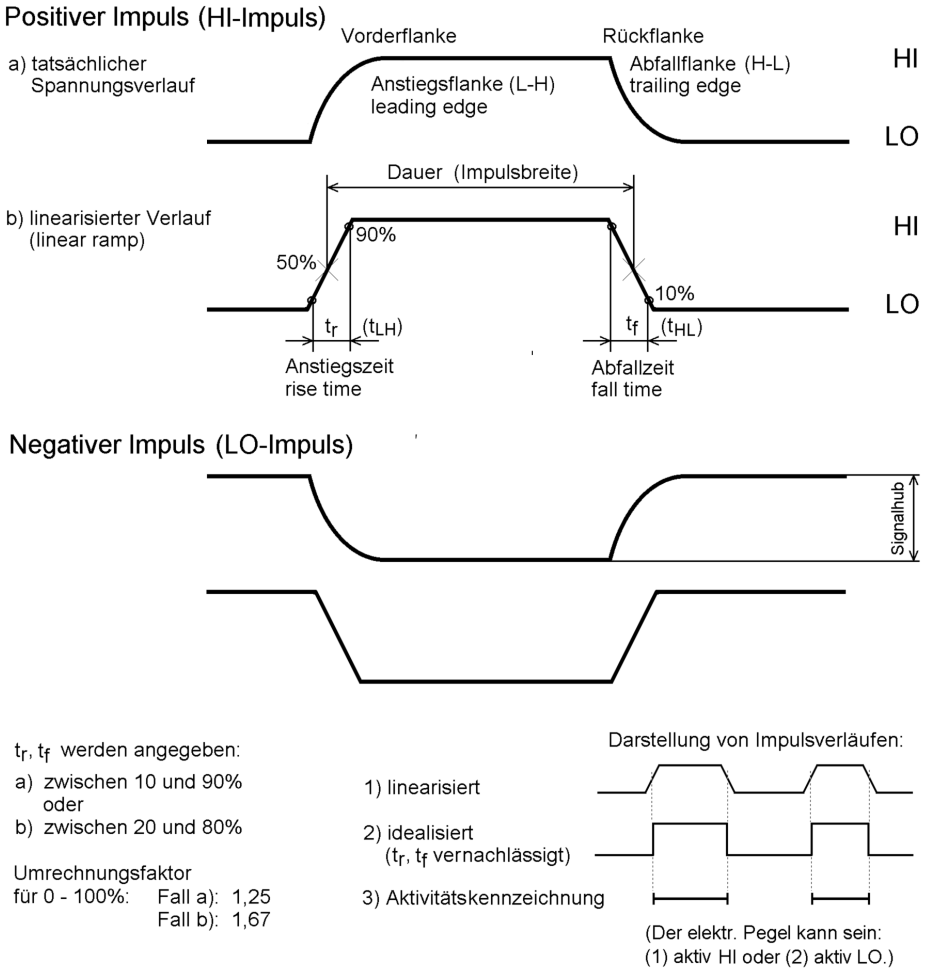


Abbildung 2.11 Impulsverläufe und wichtige Kennwerte

Impulsdauer (Impulsbreite)
Dieser Kennwert wird zwischen beiden Flanken gemessen, wenn jeweils 50% des Signalhubes erreicht sind.

Impulsflanken
Die Flanken (Edges) begrenzen den Impuls; sie bilden die Übergänge zwischen den beiden logischen Pegeln. Auf Grund elektrischer Sachverhalte sind die Flanken stets gekrümmt. Die Krümmung ist in der Praxis zumeist vernachlässigbar, so daß ein *linearisierter* Verlauf (Linear Ramp) die Übergänge zwischen den logischen Pegeln mit hinreichender Genauigkeit beschreibt.

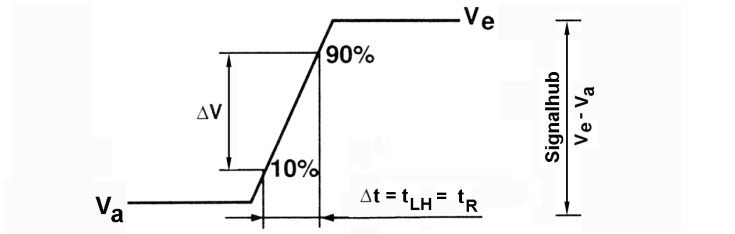
Die Bezeichnungen Anstiegs- und Abfallflanke, steigende bzw. fallende Flanke oder Low-High- bzw. High-Low-Flanke beziehen sich direkt auf die Richtung der Pegeländerung. Hingegen bezeichnen die Begriffe Vorderflanke (Leading Edge) bzw. Rückflanke (Trailing Edge) die Signalwechsel am Anfang bzw. am Ende des Impulses ohne direkten Bezug zur Richtung der Signaländerung (die Unterscheidungen sind gelegentlich beim Lesen von Datenblättern oder Funktionsbeschreibungen von Bedeutung).

Anstiegs- und Abfallzeiten
Diese Zeitangaben kennzeichnen die Dauer der jeweiligen Flanke:

- die Anstiegszeit betrifft die Low-High-Flanke (Rise Time t_R oder t_{LH}),
- die Abfallzeit betrifft die High-Low-Flanke (Fall Time t_F oder t_{HL}).

Die Zeitkennwerte werden oft zwischen 10% und 90% des jeweiligen Signalhubes gemessen, manchmal auch zwischen 20% und 80%. Gelegentlich braucht man die Zeit zwischen 0% und 100%. Abbildung 2.11 enthält die Umrechnungsfaktoren.

Flankensteilheit
Die "Steilheit" einer Signalflanke ist anschaulich klar. Es gibt aber verschiedene Angaben, um sie zu kennzeichnen. Die Bezeichnungen in der Literatur sind allerdings nicht immer eindeutig. Wir wollen hier diese Angaben als Anstiegszeit, als Anstiegsgeschwindigkeit oder Flankensteilheit (im eigentlichen Sinne) und als Anstiegsrate bezeichnen (Abbildung 2.12). Weitere gängige Fachbegriffe: Slew Rate, Edge Rate.



V_a = Anfangswert, V_e = Endwert. Signalhub = $V_e - V_a \triangleq 100\%$.
Die einschlägigen Festlegungen im Datenmaterial (s. Text) beachten.

Anstiegsgeschwindigkeit bzw. Flankensteilheit = $\frac{\Delta V}{\Delta t}$: Maßeinheit $\frac{V}{\mu s}$

Faustformel : $\frac{\Delta V}{\Delta t} \approx \frac{80\%\ V_e}{t_R} = \frac{0{,}8\ V_e}{t_R}$; $t_R \approx 0{,}8\ Ve \cdot \frac{\Delta t}{\Delta V}$

Anstiegsrate = $\frac{\Delta t}{\Delta V}$ = 1 : Anstiegsgeschwindigkeit; Maßeinheit $\frac{\mu s}{V}$ oder $\frac{ns}{V}$

Abbildung 2.12 Kennwerte von Signalflanken

Flankensteilheit bzw. Anstiegsgeschwindigkeit
Beide Begriffe werden oft in gleicher Bedeutung verwendet. Der Wert wird in Volt/Zeiteinheit (z. B. in V/µs) angegeben. Sie ist wie jede andere Geschwindigkeit (in der Physik) definiert: Anstiegsgeschwindigkeit = Spannungsdifferenz zu Zeitdifferenz. Eine Anstiegsgeschwindigkeit bzw. Flankensteilheit von 1 V/µs bedeutet, daß sich der Signalpegel innerhalb einer Mikrosekunde um 1 V ändert. Je geringer die Anstiegszeit, desto größer die Anstiegsgeschwindigkeit (Flankensteilheit).

Anstiegsrate
Die Anstiegsrate (Rise or Fall Rate) ist der Kehrwert der Anstiegsgeschwindigkeit, also das Verhältnis von Zeitdifferenz zu Spannungsdifferenz. Eine Anstiegsrate von 1 µs/V bedeutet, daß es eine Mikrosekunde dauert, bis sich der Signalpegel um 1 V geändert hat. Je geringer die Anstiegsrate, desto steiler die Flanke.

Typische Kennwerte von Signalflanken:

- langsamere Logiksignale (LS, ALS, HC). Typische Flankenanstiegszeiten: 5...10 ns, typische Flankensteilheiten um 0,2 V/ns.
- schnelle Logiksignale (AC/ACT, BiMOS, Schottky-TTL-Baureihen). Typische Flankenanstiegszeiten: 2...3 ns; typische Flankensteilheiten 1...2 V/ns.

Signalhub (Spannungshub)
Der Signalhub oder Spannungshub (Signal oder Voltage Swing) ist die Differenz zwischen Endwert und Anfangswert. Bei einer Low-High-Flanke ist der Anfangswert ein Low-Pegel und der Endwert ein High-Pegel.

Worauf beziehen sich die Anstiegs- und Abfallzeiten?
Die Anfangs- und Endwerte sind aus dem jeweiligen Datenmaterial ersichtlich. Beispiele:

- TTL: Low = 0 V, High = 3 V,
- CMOS (5-V-Baureihen): Low = 0 V, High = 4,5 V.

Worauf beziehen sich die Kennwerte der Flankensteilheit?
Auch hier ist das Datenmaterial maßgebend. Typischerweise legt man die Anfangs- und Endwerte so fest, daß gerade noch ein sicheres Umschalten zwischen Low und High gewährleistet ist. Beispiel Low-High-Flanke: Anfangswert = Low-Maximalwert V_{Lmax}, Endwert = High-Minimalwert V_{Hmin} (beides lt. Datenblatt).

Rechteckimpulse
Ein Rechteckimpuls ist ein *idealisierter* Impuls mit Anstiegs- und Abfallzeit Null. Diese Vereinfachung findet man oft in einführender Literatur und in Funktionsbeschreibungen.

Die Idealisierung ist gerechtfertigt, wenn die Impulsdauer deutlich größer ist als die Umschaltzeiten (Anstiegs- und Abfallzeit).

Beim Messen treten Impulse oft als rechteckförmige Signalverläufe in Erscheinung. Das betrifft Impulsdiagramm-Anzeigen auf Logikanalysatoren und Signaldarstellungen auf dem Oszilloskop.

Aber Achtung: Die idealisierte Darstellung ist nur dann verläßlich (d. h. uns entgeht nichts Wesentliches), wenn die Flanken in ihrem zeitlichen Verlauf tatsächlich den jeweiligen Anforderungen entsprechen (Flankensteilheit in der geforderten Größenordnung, keine Schwingungen, kein "Prellen" (Abbildung 2.13)).

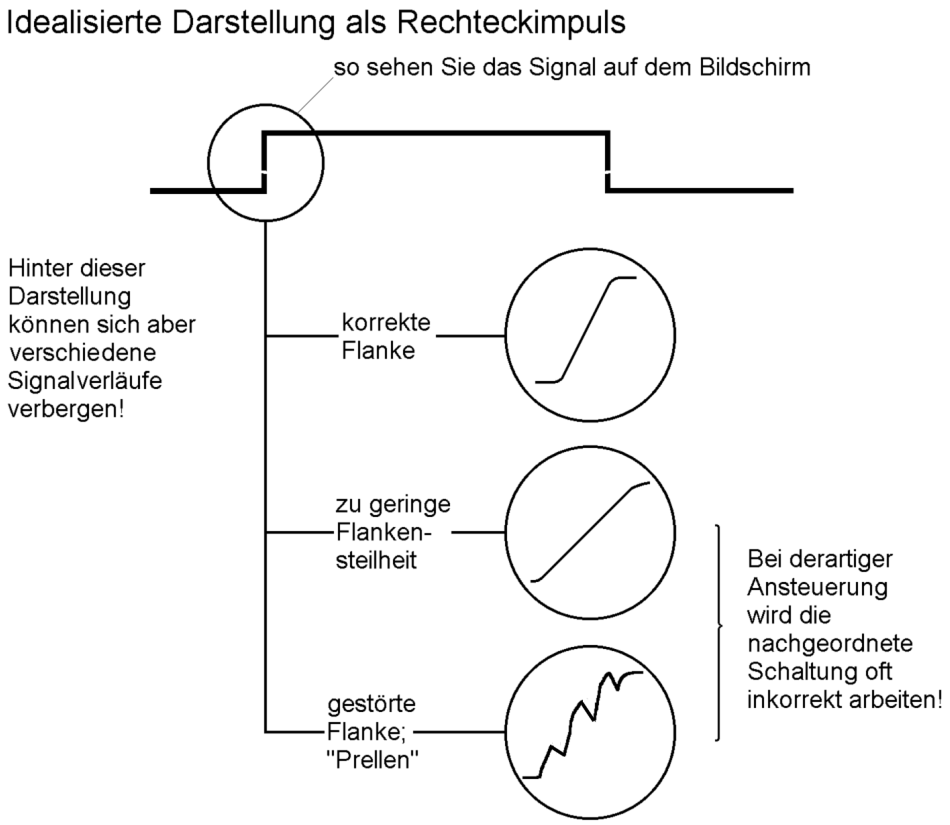


Abbildung 2.13 Idealisierte Impulsdarstellung und tatsächliche Impulsverläufe

Impulsfolgen

In der Praxis haben wir es zumeist mit Impulsfolgen zu tun. Dabei ist neben der Impulsdauer der Abstand zwischen den einzelnen Impulsen von Bedeutung. Es gibt unregelmäßige Impulsfolgen, bei denen der Abstand zwischen den Einzelimpulsen sich ständig ändert (vielfach ändert sich auch die Impulsdauer). Solche Impulsfolgen finden wir auf z. B. Daten- und Adreßleitungen und bei vielen Steuersignalen. Demgegenüber bleiben bei regelmäßigen Impulsfolgen Impulsdauer und Impulsabstand jeweils gleich.

Taktimpulse

Die in der Praxis wichtigsten regelmäßigen Impulsfolgen sind die Takte. Takte bestimmen gleichsam den Arbeitsrhythmus sequentieller Schaltungen (auch Prozessoren und ganze Motherboards sind sequentielle Schaltungen, wenngleich ausgesprochen komplizierte). Abbildung 2.14 veranschaulicht wichtige Parameter regelmäßiger Impulsfolgen, die insbesondere bei Taktsignalen von Bedeutung sind.

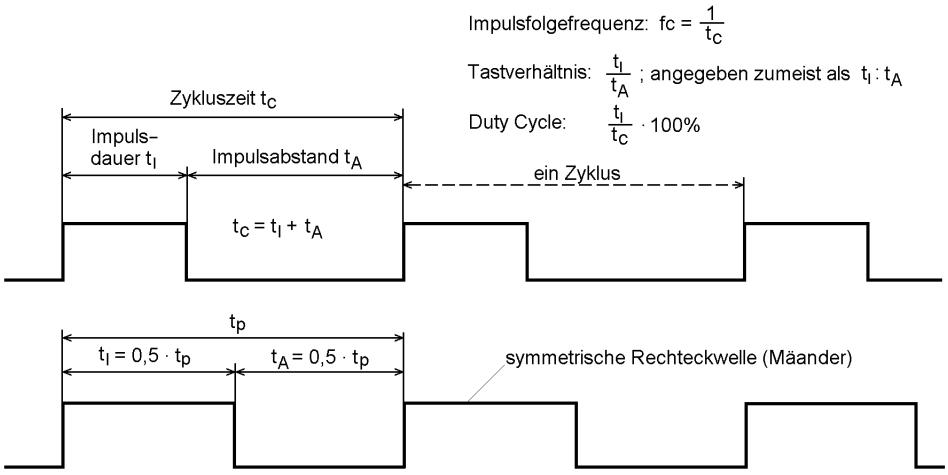


Abbildung 2.14 Parameter regelmäßiger Impulsfolgen

Zykluszeit
Die Zykluszeit (t_c) setzt sich zusammen aus der Impulsdauer (t_I) und dem Impulsabstand (t_A):

$$t_c = t_I + t_A.$$

Impulsfolgefrequenz
Die Impulsfolgefrequenz (Taktfrequenz) f_c ergibt sich als Kehrwert der Zykluszeit:

$$f_c = \frac{1}{t_c}$$

Tastverhältnis und Duty Cycle
Das Tastverhältnis ist das Verhältnis zwischen Impulsdauer (t_I) und Impulsabstand (t_A):

Tastverhältnis = $t_I : t_A$.

Der Duty Cycle (sprich: Djutieh Seikl) wird in Prozent angegeben, wobei die Impulsdauer (t_I) auf die gesamte Zykluszeit (t_c) bezogen wird. Tabelle 2.3 enthält einige Beispiele.

$$\text{Duty Cycle} = \frac{t_I}{t_c} \cdot 100\ \% = \frac{t_I}{t_I + t_A} \cdot 100\ \%$$

Impulsdauer t_I	Impulsabstand t_A	Tastverhältnis	Duty Cycle
10 ns	40 ns	10 : 40 = 1 : 4	20%
25 ns	25 ns	25 : 25 = 1: 1	50%
40 ns	10 ns	40 : 10 = 4 : 1	80%

Tabelle 2.3 Beispiele zu Tastverhältnis und Duty Cycle. Zykluszeit t_c = 50 ns

Symmetrische Rechteckwelle (Mäander)
Von besonderer praktischer Bedeutung - namentlich als Taktsignale - sind Impulsfolgen mit einem Tastverhältnis von 1:1 bzw. 50% Duty Cycle. Solche Impulsfolgen bezeichnet man auch als Rechteckwellen oder Mäander (nach einem Fluß in Griechenland, der - auf einer Landkarte passenden Maßstabs - einen ähnlichen Verlauf zeigt).

Gültigkeitsimpulse
Gültigkeitsimpulse kennzeichnen allgemein die Gültigkeit anderer Signale, sie bilden gleichsam die Bezugsbasis für deren Auswertung bzw. Verarbeitung. Taktimpulse sind - in modernen Systemen - die anwendungspraktisch wichtigsten Gültigkeitsimpulse (Stichwort: vollsynchrone Arbeitsweise).

Strobe-Impulse
Als Strobe-Impuls bezeichnet man ein Gültigkeitssignal, das nur im Bedarfsfall auftritt (während es sich bei Takten zumeist um regelmäßige, ständig durchlaufende Impulsfolgen handelt).

Sowohl bei Takt- als auch bei Strobe-Impulsen kommt es darauf an, wie deren Zeitverhältnisse in bezug auf jene Signale festgelegt sind, deren Gültigkeit sie steuern. Abbildung 2.15 veranschaulicht wichtige Begriffe und Kennwerte.

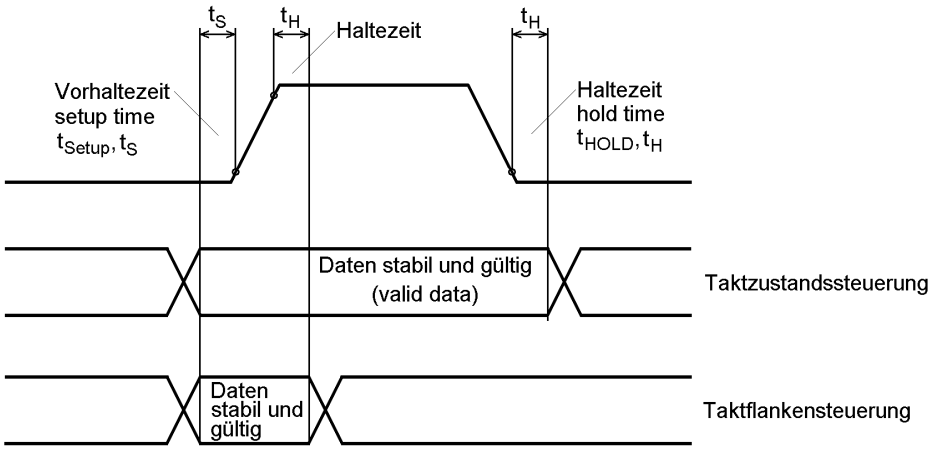


Abbildung 2.15 Gültigkeitsimpuls (Takt, Strobe) in bezug auf Daten

Wichtige Zeitkennwerte: Vorhaltezeit und Haltezeit

Vorhaltezeit (Setup Time t_S)
Um diese Zeit müssen die auf den Gültigkeitsimpuls bezogenen Signale (z. B. Datenbusbelegungen) *vor* der jeweiligen Flanke des Gültigkeitsimpulses gültig sein, also stabil anliegen.

Haltezeit (Hold Time t_H)
Nach der betreffenden Flanke des Gültigkeitsimpulses müssen die auf ihn bezogenen Signale weiterhin eine gewisse Haltezeit stabil gehalten werden; sie dürfen sich gegenüber dem Wert, den sie zur Vorhaltezeit hatten, nicht ändern.

Die zulässigen Mindestwerte der Setup- und Hold-Zeiten sind im jeweiligen Datenblatt vermerkt.

Taktflankensteuerung
Beide Zeitangaben beziehen sich auf ein und dieselbe Flanke; während der sonstigen Dauer des Gültigkeitsimpulses dürfen die Signale beliebig umschalten.

Taktzustandssteuerung
Die Vorhaltezeit bezieht sich auf die Vorderflanke; die Haltezeit auf die Rückflanke. Für die Dauer des Gültigkeitsimpulses dürfen die auf ihn bezogenen Signale nicht umschalten.

3. Dokumentation digitaler Schaltungen

Die industrielle Herstellung beliebiger technischer Güter ist ohne exakte Dokumentation nicht denkbar. Eine technische Einrichtung exakt dokumentieren heißt, sie in Funktion und Struktur so genau zu beschreiben, daß man sie fertigen, prüfen, aufstellen, warten und reparieren kann (hinzu kommt das Zerlegen zwecks Entsorgung bzw. Wiederverwertung). Dafür sind verschiedene Darstellungsweisen bzw. Beschreibungsmittel nutzbar (Tabelle 3.1). In jedem Gebiet der Technik haben sich bestimmte Beschreibungsmittel bewährt, obwohl andere aus "akademischer" Sicht dasselbe leisten könnten. So könnte man ein Haus oder einen Dieselmotor durchaus im Stil einer Programmiersprache exakt dokumentieren. In der Praxis sind aber *Zeichnungen* das bevorzugte Darstellungsmittel, ergänzt um Tabellen und Listen (Datenblätter, Stücklisten usw.) und Beschreibungen in Textform (Vorschriften- und Normenwerke, Betriebs- und Wartungsanleitungen usw.). Systeme und Geräte werden zumeist in *Handbüchern* (Reference Manuals) und in *Zeichnungssätzen* dokumentiert, Bauelemente in *Datenblättern* bzw. Datenbüchern (Data Sheets, Data Books).

allgemeine Dokumentation	**Fertigungsdokumentation**	**Servicedokumentation**
▪ Datenmaterial, ▪ Applikationsschriften, ▪ Standards, ▪ Normen, ▪ Vorschriften, ▪ Lastenhefte, ▪ Betriebsanleitungen (für Entwicklungssysteme usw.), ▪ Sprachbeschreibungen	▪ Zeichnungssätze, ▪ Entwurfs- und Fertigungsdateien, ▪ Programm-Listings, ▪ Fertigungsanweisungen, ▪ Prüf- und Abnahmevorschriften, ▪ Vorgaben zu Verpackung, Transport, Lagerung, Wiederverwertung usw. ▪ Testdaten, ▪ beschreibende Dokumentation (ähnlich Servicedokumentation)	▪ Systembeschreibung, ▪ Betriebsanleitung, ▪ Wartungsvorschrift, ▪ Installations- bzw. Inbetriebnahmevorschrift, ▪ Beschreibung des konstruktiven Aufbaus, ▪ Beschreibung der Wirkungsweise (Theory of Operation), ▪ Fehlersuchanleitung (Troubleshooting Manual), ▪ für alle Baugruppen: Stromlaufpläne (Serviceschaltpläne), Bestückungspläne und Stücklisten

Tabelle 3.1 Zur Dokumentation elektronischer Geräte - eine Übersicht

Gibt es ein optimales Beschreibungsmittel?
Zumindest gibt es nicht *das* beste Beschreibungsmittel. Vielmehr wird eine gute Dokumentation immer eine Kombination verschiedenartiger Darstellungen sein (das sind Schaltpläne, Ansichts- und perspektivische Zeichnungen, Impulsdiagramme, Listen, Tabellen, Beschreibungen in Textform usw.). *Optimal* ist eine Dokumentation, wenn sie *verständlich* ist, wenn sie es ermöglicht, daß sich der Nutzer eine klare Vorstellung von Aufbau und Funktionsweise der dokumentierten Einrichtung bilden kann (und wenn sie zudem nicht umfangreicher ist als unbedingt nötig).

Womit wir es in der Praxis zu tun bekommen können
Begnügen wir uns - für den Anfang - mit einigen Beispielen (Abbildungen 3.1 bis 3.10).

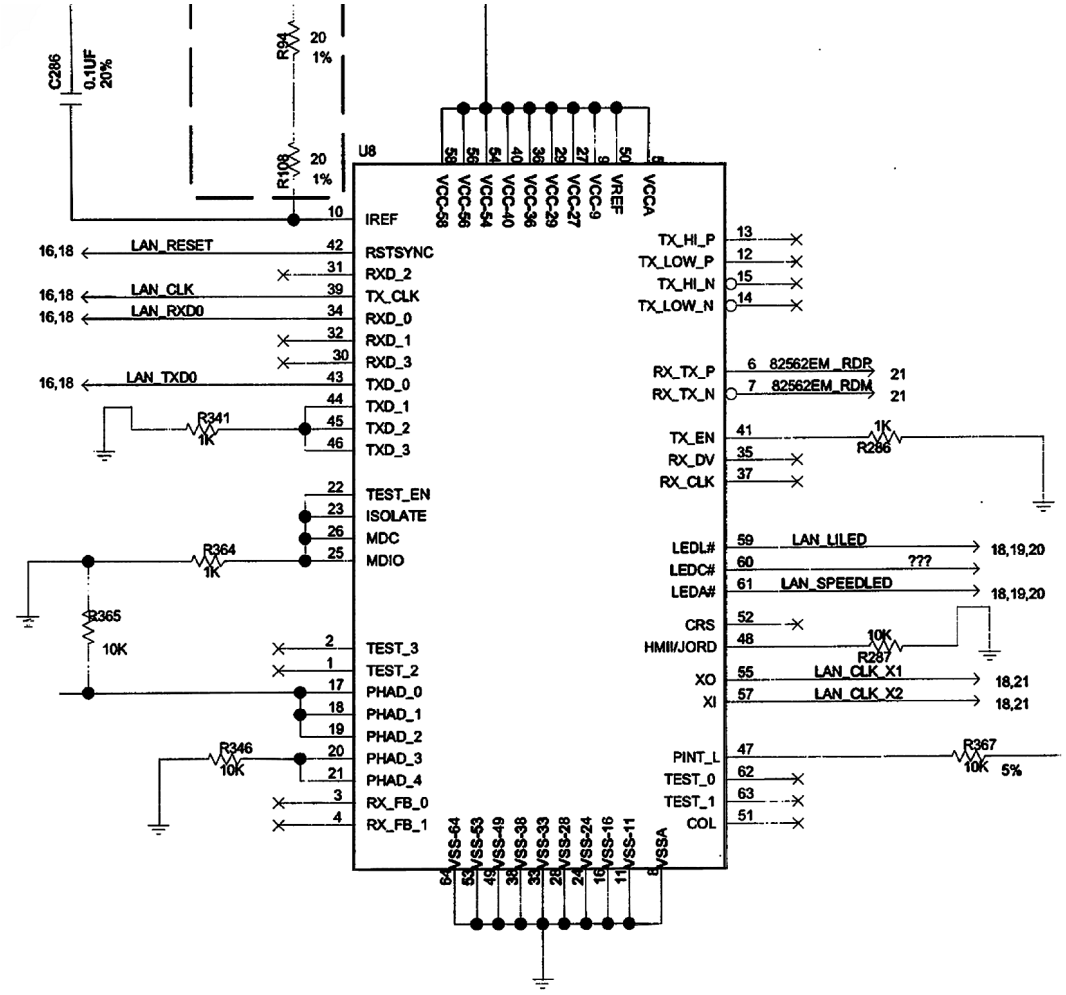


Abbildung 3.1 Auszug aus dem Schaltplan eines Motherboards (Intel)

Erklärung:
Die betreffenden Funktionen (hier: die LAN-Anschlußsteuerung) sind in einem einzigen Schaltkreis untergebracht. Wirklich kompliziert sieht das Schaltbild gar nicht aus - der Schaltkreis, auf den es ankommt, ist einfach ein Kästchen mit vielen Linien drumherum. (Richtig "dicke" Schaltkreise - Prozessor, Verteiler (Hubs) usw. - sind jeweils auf mehreren Blättern dargestellt.)

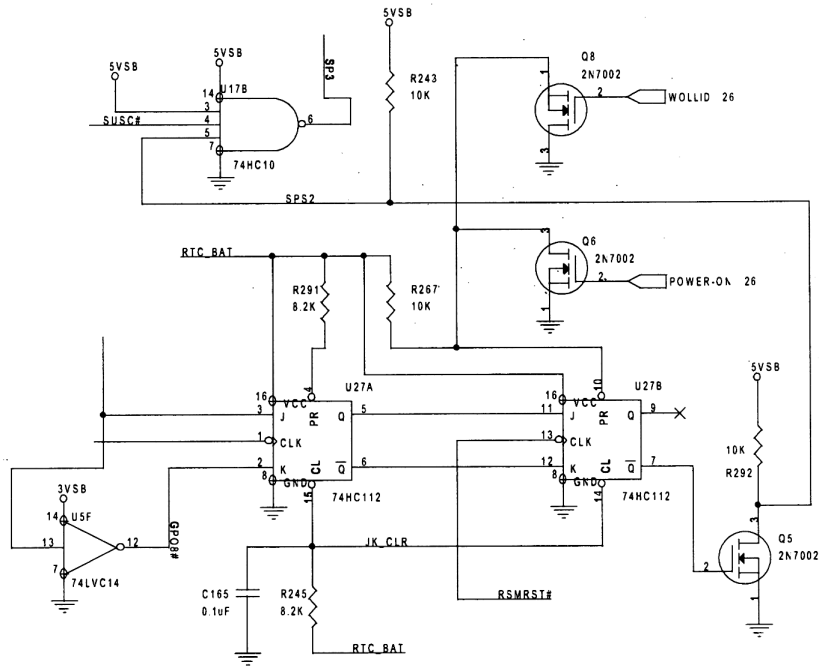


Abbildung 3.2 Ein weiterer Schaltplanauszug (Intel)

Erklärung:
Kein Großschaltkreis, sondern viel Kleinzeug. Eine auszugsweise Darstellung der Glue Logic eines Motherboards.

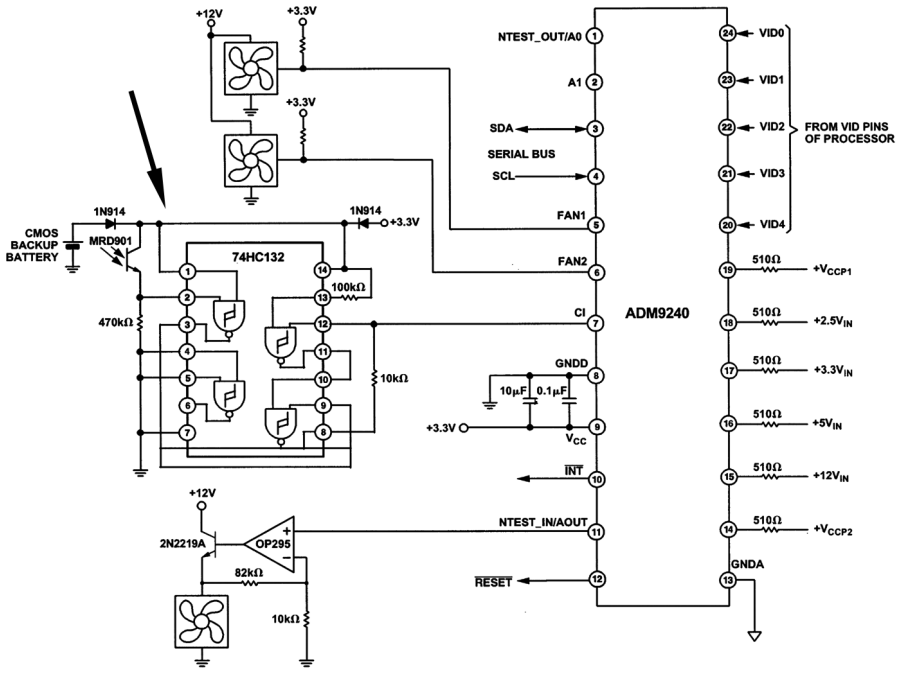


Abbildung 3.3 Überwachungsschaltkreis (Hardware Monitor) in Anwendungsschaltung (AMD)

Erklärung zu Abbildung 3.3:
Der Überwachungsschaltkreis ist hier in Verbindung mit den zu überwachenden Lüftern, Speisespannungen usw. dargestellt (dabei hat sich der Zeichner viel Narrenfreiheit genommen - wie ein streng normgerechter Schaltplan sieht es nicht gerade aus...). Ersichtlicherweise sind noch einige Kleinteile erforderlich. Der Pfeil zeigt auf eine Kunstschaltung, die das Öffnen des Gehäuses erkennen soll (anhand des Lichteinfalls auf einen Phototransistor), und zwar auch dann, wenn der PC ausgeschaltet ist (deshalb die Stützbatterie).

Truth Table(1)

$\overline{OE}$	$\overline{WE}$	$\overline{CE1}$	CE2	I/O	MODE
X	X	H	X	Hi-Z	Standby
X	X	X	L	Hi-Z	Standby
L	H	L	H	D_{OUT}	Read
X	L	L	H	D_{IN}	Write
H	H	L	H	Hi-Z	Output Disable

NOTE: 1. H = V_{IH}, L = V_{IL}, X = DON'T CARE

Abbildung 3.4 Datenblattauszug (1): Wahrheitstabelle (Paradigm)

Erklärung:
Wahrheits- oder Funktionstabellen (Truth oder Function Tables) beschreiben die Funktion eines Schaltkreises durch tabellarische Darstellungen der Eingangs- und Ausgangsbelegungen.

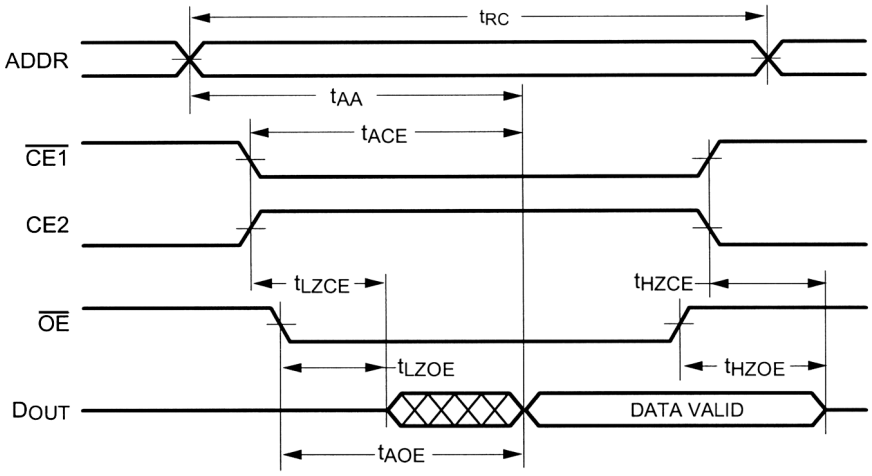


Abbildung 3.5 Datenblattauszug (2): Impulsdiagramm (Paradigm)

AC Electrical Characteristics

Description		-10(7)		-12(7)		-15		
READ Cycle	**Sym**	**Min.**	**Max.**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**
READ cycle time	t_{RC}	10		12		15		ns
Address access time	t_{AA}		10		12		15	ns
Chip enable access time	t_{ACE}		10		12		15	ns
Output hold from address change	t_{OH}	3		3		3		ns
Chip enable to output in low Z(1,3)	t_{LZCE}	5		5		5		ns
Chip disable to output in high Z(1,2,3)	t_{HZCE}		6		6		7	ns
Chip enable to power up time(3)	t_{PU}	0		0		0		ns
Chip disable to power down time(3)	t_{PD}		10		12		15	ns
Output enable access time	t_{AOE}		6		6		6	ns
Output enable to output in low Z (1,3)	t_{LZOE}	0		0		0		ns
Output disable to output in high Z(1,3)	t_{HZOE}		6		6		6	ns

Abbildung 3.6 Datenblattauszug (3): Kennwerttabelle (Paradigm)

Erklärung zu den Abbildungen 3.5 und 3.6:
Viele Abläufe werden anhand von Impulsdiagrammen dokumentiert. Die zugehörigen Zeitkennwerte sind in Tabellen zusammengefaßt. Wie sind solche Darstellungen zu lesen? - Alles Übungssache. Wir werden uns nach und nach herantasten.

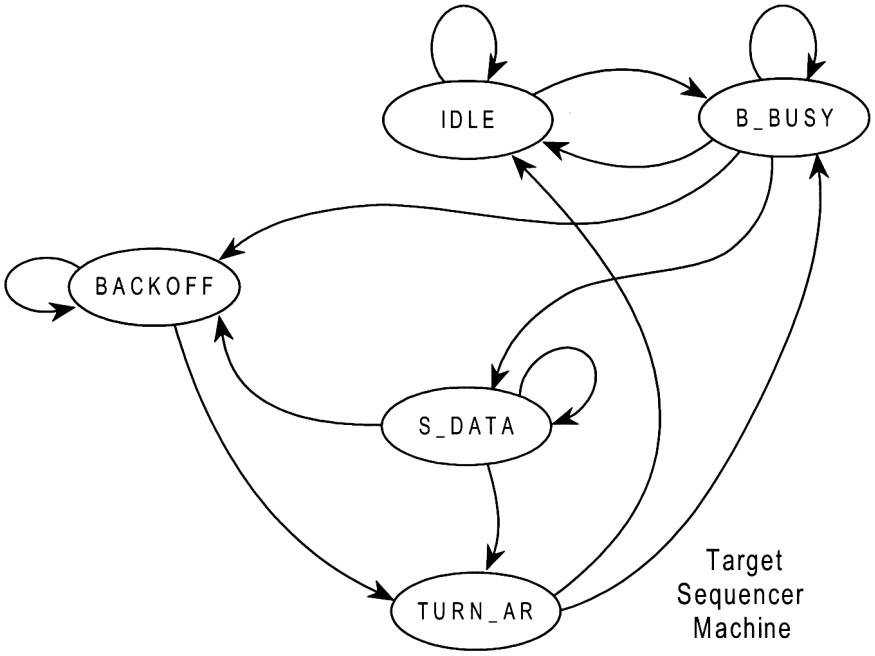


Abbildung 3.7 Zustandsgraph (PCI SIG)

IDLE or TURN_AR -- Idle condition or completed transaction on bus.

goto IDLE	if **FRAME#**
goto B_BUSY	if !**FRAME#** * !Hit

B_BUSY -- Not involved in current transaction.

goto B_BUSY	if (!**FRAME#** + !D_done) * !Hit
goto IDLE	if **FRAME#** * D_done + **FRAME#** * !D_done * !**DEVSEL#**
goto S_DATA	if (!**FRAME#** + !**IRDY#**) * Hit * (!Term + Term * Ready) * (FREE + LOCKED * L_lock#)
goto BACKOFF	*if (!**FRAME#** + !**IRDY#**) * Hit * (Term * !Ready + LOCKED * !L_lock#)

S_DATA -- Agent has accepted request and will respond.

goto S_DATA	if !**FRAME#** * !**STOP#** * !**TRDY#** * **IRDY#** + !**FRAME#** * **STOP#** + **FRAME#** * **TRDY#** * **STOP#**
goto BACKOFF	if !**FRAME#** * !**STOP#** * (**TRDY#** + **!IRDY#)**
goto TURN_AR	if **FRAME#** * (!**TRDY#** + !**STOP#**)

BACKOFF -- Agent busy unable to respond at this time.

goto BACKOFF	if !**FRAME#**
goto TURN_AR	if **FRAME#**

Abbildung 3.8 Boolesche Gleichungen, die die Zustandsübergänge in Abbildung 3.7 beschreiben (PCI SIG)

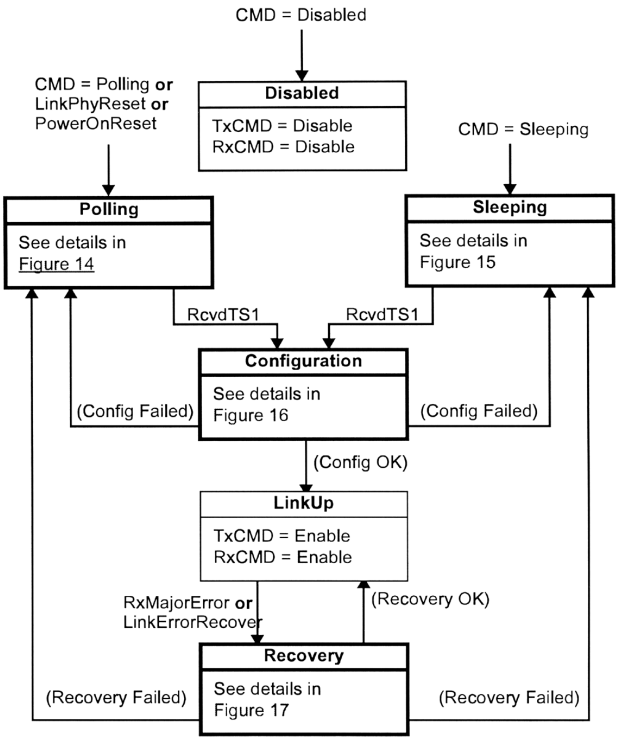


Abbildung 3.9 Ein weiterer Zustandsgraph (InfiniBand Trade Association)

Erklärung zu den Abbildungen 3.7 bis 3.9:
Es handelt sich um Auszüge aus Standards (PCI, InfiniBand). Hier geht es uns lediglich darum, verschiedene Beispiele der Dokumentation von Zustandsgraphen bzw. Zustandsdiagrammen zu zeigen. Näheres in Abschnitt 3.2.2.

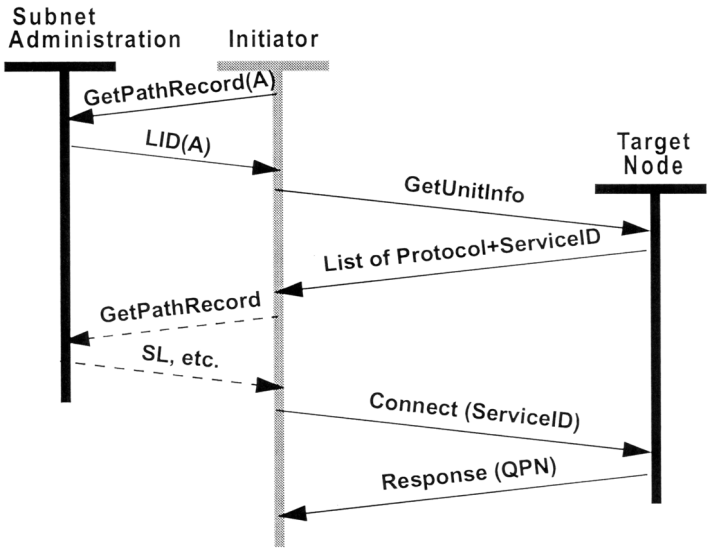


Abbildung 3.10 Aktivitätsliste (InfiniBand Trade Association)

Erklärung:
Solche Darstellungen dienen dazu, das Zusammenwirken verschiedener Einrichtungen zu veranschaulichen. Mehr in Abschnitt 3.2.4.

3.1. Ausdrucksmittel der Strukturbeschreibung

Das typische, seit Jahrzehnten bewährte Mittel der Strukturbeschreibung ist der *Schaltplan*. Im Entwicklungsbereich wird er mehr und mehr durch *Hardware-Beschreibungssprachen* ergänzt.

Zum Fertigen, Messen und Fehlersuchen braucht man neben der Schaltungsstruktur auch noch Angaben zur Lage der einzelnen Bauelemente, der Steckkarten usw. Die einzelne Leiterplatte wird üblicherweise durch einen *Bestückungsplan* dokumentiert. Komplexe Geräte erfordern zudem *Übersichtszeichnungen*, die die Lage der einzelnen Funktionseinheiten angeben. Die einzelnen Bauelemente sind üblicherweise in *Stücklisten* nach Art und Anzahl zusammengestellt.

3.1.1. Schaltsymbole und Schaltpläne

Ein Schaltplan (Schaltbild; Circuit Diagram) ist die zeichnerische Darstellung einer elektrischen Schaltung. Die einzelnen Funktions- bzw. Bauelemente, wie Schaltkreise, Gatter, Flipflops, Widerstände usw. werden durch Schaltsymbole (Schaltzeichen) dargestellt, die elektrischen Verbindungen (Leitungen) durch Linien.

Begriffserklärung
Als *Funktionselement* bezeichnen wir eine in sich unteilbare Elementarschaltung. Ein *Bauelement* ist ein körperlich gegebenes auswechselbares Funktionselement oder ein entsprechender Komplex von Funktionselementen (z. B. ein Schaltkreis). Die Unterscheidung ist deshalb wesentlich, weil viele Elementarschaltungen, wie z. B. Widerstände oder Transistoren, sowohl als Funktionselemente auf Schaltkreisen als auch als gesonderte Bauelemente vorkommen.

Logische und elektrische Schaltpläne
Jede gute technische Darstellung muß sowohl exakt als auch verständlich und anschaulich sein. Es ist deshalb nicht nur Ansichtssache, sondern geradezu eine Notwendigkeit, Einzelheiten wegzulassen, die zum Verständnis der Zusammenhänge oder zu bestimmten Tätigkeiten (z. B. zum Fertigen oder zum Fehlersuchen und Reparieren) nicht unbedingt nötig sind.

Moderne Digitalschaltungen bestehen vorwiegend aus logischen Funktionselementen. Um die Schaltung zu verstehen, sind Kenntnissse des elektrisches "Innenlebens" gar nicht erforderlich. Deshalb werden Digitalschaltungen vorwiegend durch Schaltsymbole für logische Funktionseinheiten (vom Gatter bis zum Prozessorschaltkreis) dargestellt. Manchmal kommt es allerdings vor, daß elektrische Einzelheiten bis hin zum einzelnen Widerstand oder Kondensator dargestellt werden müssen (eben dann, wenn diese Elemente für das Funktionieren der Schaltung von Bedeutung sind). Man findet deshalb oft eine “gemischte" Darstellung mit Symbolen logischer und elektrischer Funktions- und Bauelemente (vgl. Abbildung 3.3).

Schaltsymbole (Schaltzeichen)
Trotz vieler Bemühungen um Standardisierung gibt es keine weltweit einheitliche Symbolik. In Deutschland ist DIN 40 900 der grundlegende Standard. Abbildung 3.11 stellt verschiedene Schaltsymbole logischer Funktionselemente gegenüber. Abbildung 3.12 zeigt wichtige Schaltzeichen "elektrischer" Bauelemente.

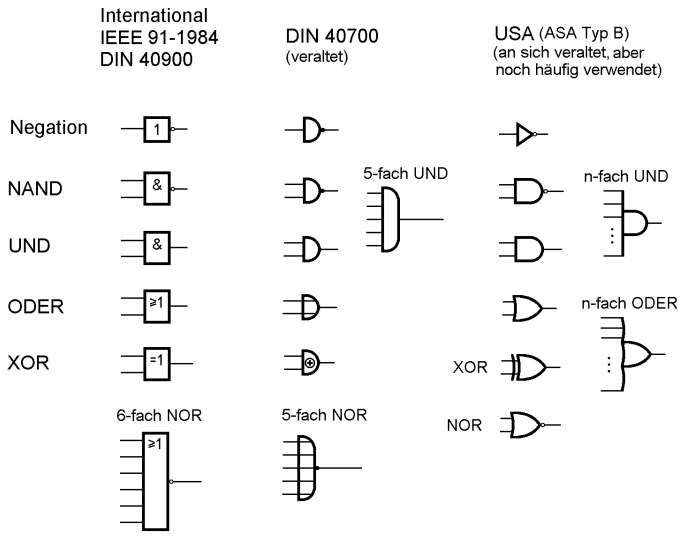


Abbildung 3.11 Schaltsymbole für Gatter

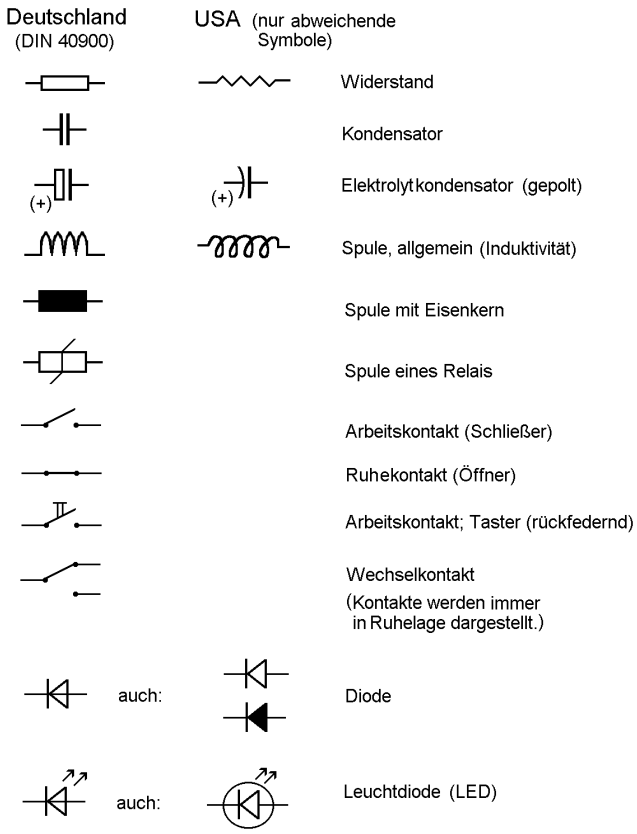


Abbildung 3.12 Wichtige Schaltzeichen

Hinweis:
Für Widerstände, Kondensatoren usw. und für Gatter werden wir die DIN-Symbolik verwenden. In vielen Schaltplänen der Praxis werden wir allerdings die alten US-Schaltsymbole antreffen (vgl. die Abbildungen 3.1 bis 3.3).

Achtung:
Verwechseln Sie nicht das US-Widerstandssymbol (die Zickzacklinie) mit dem Spulensymbol nach DIN.

Kombinatorische und sequentielle Schaltungen
In der modernen Symbolik gibt es keine Besonderheiten, wodurch sich Schaltplandarstellungen kombinatorischer und sequentieller Schaltungen auf den ersten Blick voneinander unterscheiden (alle Symbole haben Kästchenform). Hingegen erkennt man in den eher traditionellen Darstellungen Funktionselemente, die Speicherschaltungen enthalten, an der Kästchenform, während Gatter als halbkreisförmige, bogenförmige oder abgerundete Symbole erkennbar sind.

Der Standard ANSI/IEEE 91-1984 für Logiksymbole (DIN 40900, Teil 12)
Es hat viele Jahre gedauert, bis dieser Standard ausgearbeitet war. Anfänglich wurde lediglich die Kästchen-Symbolik für alle logischen Funktionseinheiten, bis hin zum Gatter, eingeführt. (Beiläufig: Ein wichtiger Grund für die Kästchen war seinerzeit die Ausgabe von Schaltplänen mit den Schnelldruckern der klassischen EDV-Anlagen. Diese konnten nur zeilenweise drucken und hatten einen recht beschränkten Zeichensatz. Die Ausgabe war aber wesentlich schneller als

über Plotter. Deshalb war es wichtig, die Schaltsymbole aus druckbaren Zeichen, wie Sternchen oder senkrechten und waagerechten Strichen, zusammenstückeln zu können.Vgl. auch Abbildung 3.18.) Im Laufe der Zeit hatte man sich ein bedeutend ehrgeizigeres Ziel gestellt: eine symbolische Darstellung auch komplizierter Funktionseinheiten, aus der die Funktionsweise eindeutig hervorgeht, ohne die Innenschaltung oder eine besondere Funktionsbeschreibung angeben zu müssen.

Die Symbole haben grundsätzlich rechteckige Form. Wenn nötig, sind allen Funktionselementen gemeinsame Steuerungs- bzw. Ausgangsschaltungen (Common Control Blocks, Common Output Elements) vom Datenteil abgesetzt ("Einschnürung", doppelte Trennlinie). Besondere Symbole (Qualifying Symbols) kennzeichnen die Wirkung bzw. Nutzungsweise von Ein- und Ausgängen sowie die funktionellen Abhängigkeiten (Abhängigkeitsnotation; Dependency Notation). Die Abbildungen 3.13 bis 31.5 sollen einen ersten Eindruck von dieser Symbolik vermitteln.

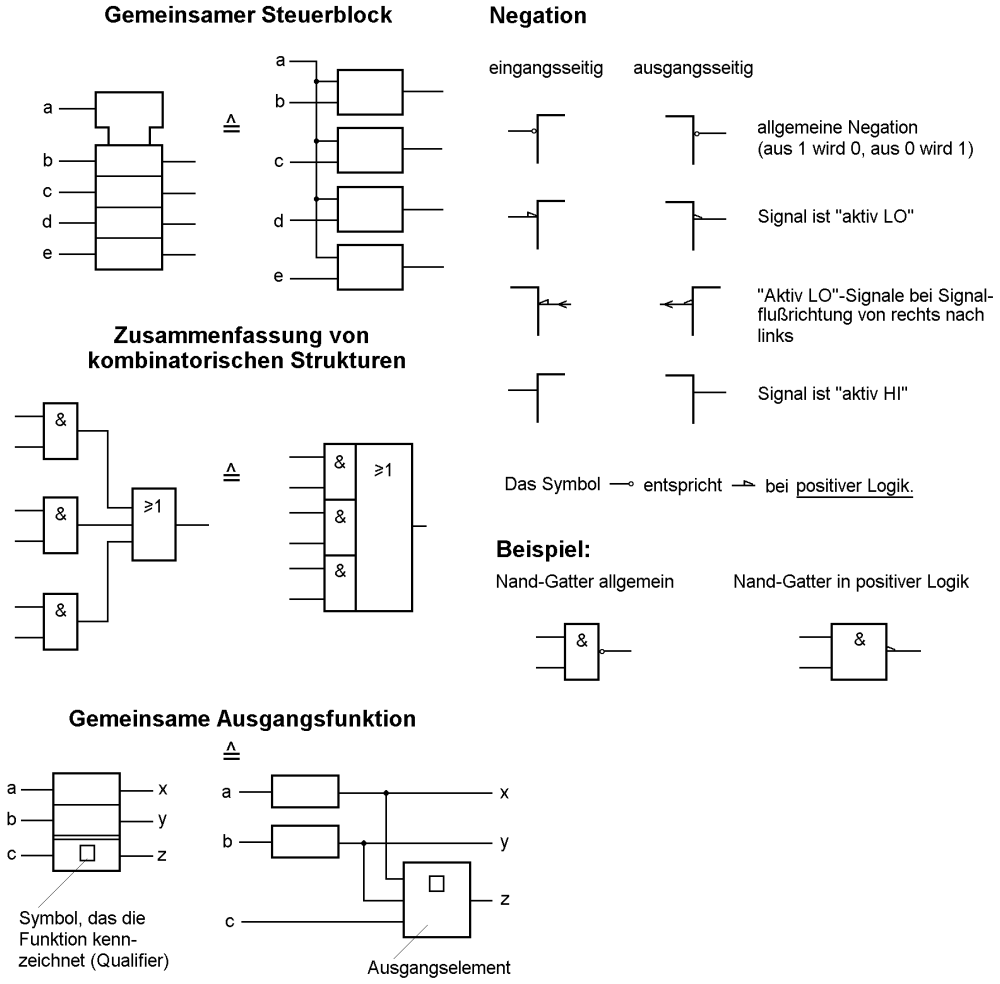


Abbildung 3.13 Wichtige Gestaltungsprinzipien der Schaltsymbole nach DIN 40900 bzw. IEEE 91-1984

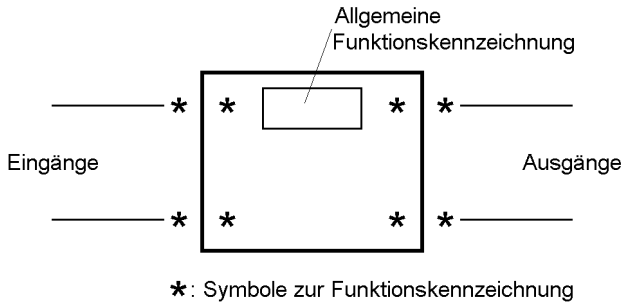


Abbildung 3.14 Anordnung der Funktionskennzeichnungen (Qualifiers) im Schaltsymbol

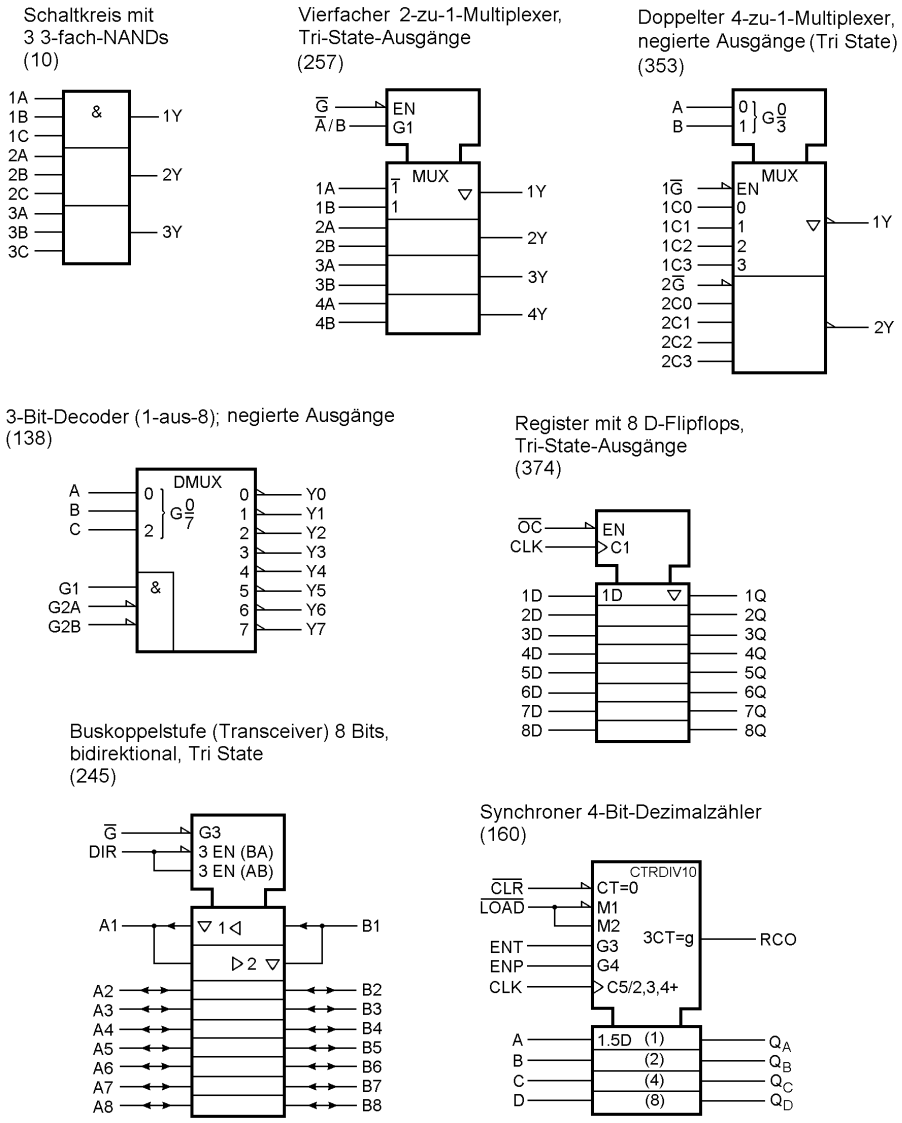


Abbildung 3.15 Beispiele für Schaltsymbole nach DIN 40900 bzw. IEEE 91-1984 (nach: Texas Instruments). Zahlenangaben wie 10, 257, 353 usw. kennzeichnen den Schaltkreistyp.

Die Praxis

Das Ziel, funktionelle Abhängigkeiten symbolisch *exakt* anzugeben, ist im Laufe der Zeit von der Entwicklung der Schaltkreistechnologie überholt worden. Die Notation gestattet nur die Wiedergabe vergleichsweise einfacher Funktionszusammenhänge, etwa im Falle eines Decoders, Zählers oder Schieberegisters. Wie will man aber die Funktionsweise eines Prozessors, eines Videoschaltkreises oder eines Motherboard-Steuerschaltkreises darstellen? Jeder derartige Schaltkreis erfordert ein Datenblatt bzw. Handbuch von vielleicht 300 Seiten und mehr!

Deshalb ist in vielen modernen Schaltplänen kaum die Notation des Standards IEEE 91-1984 zu finden. Vielmehr werden komplexe Schaltkreise einfach durch Kästchen mit den entsprechenden Ein- und Ausgängen dargestellt. Diese Vereinfachung wird oft auch auf Funktionselemente ausgedehnt, die an sich ohne weiteres gemäß dem Standard darstellbar wären, wie z.B. Buskoppelstufen. Manche Entwicklungssysteme liefern eine gemischte Darstellung: allgemein übliche Funktionselemente (z. B. Gatter, Multiplexer, Bustreiber usw.), die sich gemäß IEEE 91-1984 noch überschaubar darstellen lassen, werden auch gemäß diesem Standard wiedergegeben. Alle anderen Funktionselemente (programmierbare Logik, Prozessoren, Speicher usw.) werden hingegen als einfache Kästchen dargestellt. Meistens finden wir aber auch in ganz modernen Schaltplänen (vgl. Abbildung 3.2) noch die alte US-amerikanische Gattersymbolik (was immerhin den Vorteil hat, Gatter, Negatoren, Treiber usw. von den komplexeren Funktionselementen auf den ersten Blick unterscheiden zu können).

Ein Privatstandard

Für unsere eigenen Darstellungen verwenden wir gelegentlich eine Vereinfachung des IEEE-Standards, die sich auch in der Industriepraxis bewährt hat. Es gibt nur rechteckige Kästchen, ohne abgesetzte Steuerblöcke. Eingänge liegen links, Ausgänge rechts. Bidirektionale Anschlüsse (die sowohl als Eingang wie auch als Ausgang arbeiten) werden je nach Zweckmäßigkeit rechts oder links angeordnet. Kennzeichnende Symbole sehen wir nur für die Taktflanke und für die Negation vor.

Wenn man von Hand zeichnet, hat man gelegentlich Schwierigkeiten, die Anschlußbezeichnungen jeweils rechts und links eindeutig zuzuordnen und die zusätzlich erforderlichen Angaben (z.B. über den Schaltkreistyp) unterzubringen. (Den an sich erforderlichen Schriftgrad bekommt man von Hand eben nur schwer hin.) Hierfür hat sich folgende Abhilfe bewährt: Das Kästchen wird senkrecht in drei Streifen geteilt. Der linke Streifen enthält die Eingangskennzeichnung, der mittlere den Typ und ggf. weitere Angaben (wie etwa die Position auf der Leiterplatte) und der rechte die Ausgangskennzeichnung. Eingänge und Ausgänge können gruppenweise voneinander abgesetzt und im jeweiligen Kennzeichnungsstreifen durch dünne waagerechte Linien getrennt werden (so lassen sich z. B. Daten-, Steuer- und Taktsignale voneinander absetzen). Für die Kennzeichnung werden möglichst suggestive, ohne weiteres verständliche Abkürzungen verwendet. Abbildung 3.16 gibt einen Überblick über diesen "privaten" Standard, und zwar anhand von Beispielen (die Sie mit Abbildung 3.15 vergleichen sollten).

Wird mit moderner Software gezeichnet, so verzichtet man typischerweise auf die Hilfslinien, da es hierbei möglich ist, durch Wahl verschiedener Linienstärken, Schriftgrade usw. die Symbole ansehnlich und übersichtlich zu gestalten.

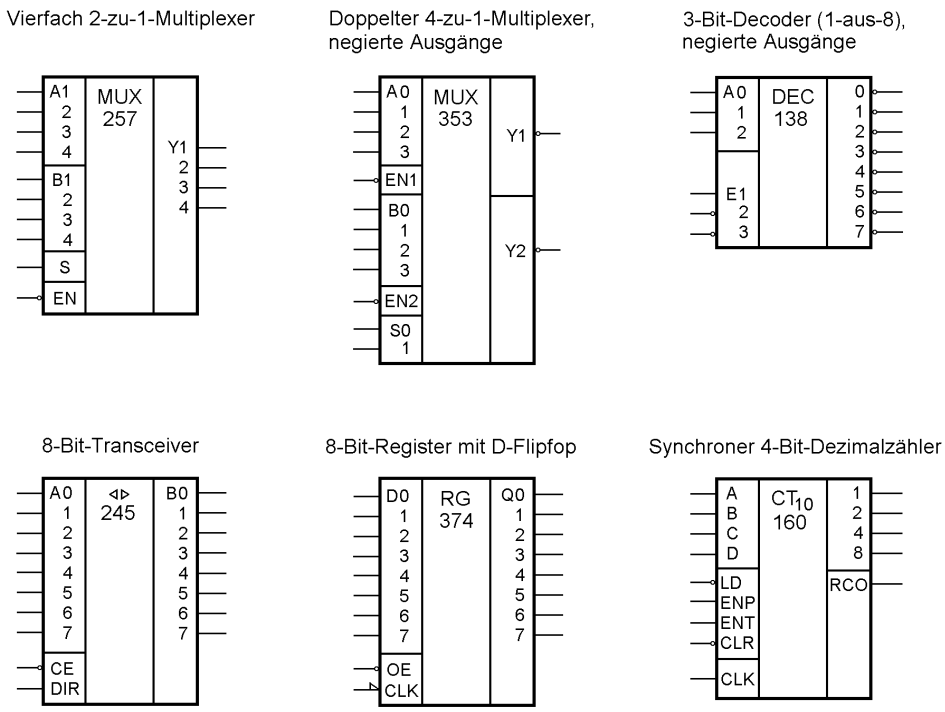


Abbildung 3.16 Ein Privatstandard (anhand von Beispielen)

Verbindungen

Verbindungen (Abbildung 3.17) werden als dünne Vollinien ausgeführt. Abzweigungen werden durch einen vollen dicken Punkt gekennzeichnet ("Lötstelle"). Dünne Vollinien, die sich lediglich kreuzen, sind voneinander unabhängig, also nicht miteinander verbunden. Verbindungen werden nur waagerecht und senkrecht geführt; Ausnahmen sind nur in besonderen Fällen üblich (z.B. um die Rückführungen in Latches zu kennzeichnen). Wenn es aus Gründen der Übersichtlichkeit zweckmäßig ist, wird man manche Verbindungen nicht insgesamt darstellen, sondern die Linie unterbrechen und die Verbindung durch Verweis kennzeichnen. Ist der Schaltplan auf verschiedene Blätter aufgeteilt, so ist die Verbindung auf jedem Blatt durch einen eindeutigen Signalbezeichner (Signal Identifier) gekennzeichnet, oft auch noch durch zusätzliche Verweisangaben.

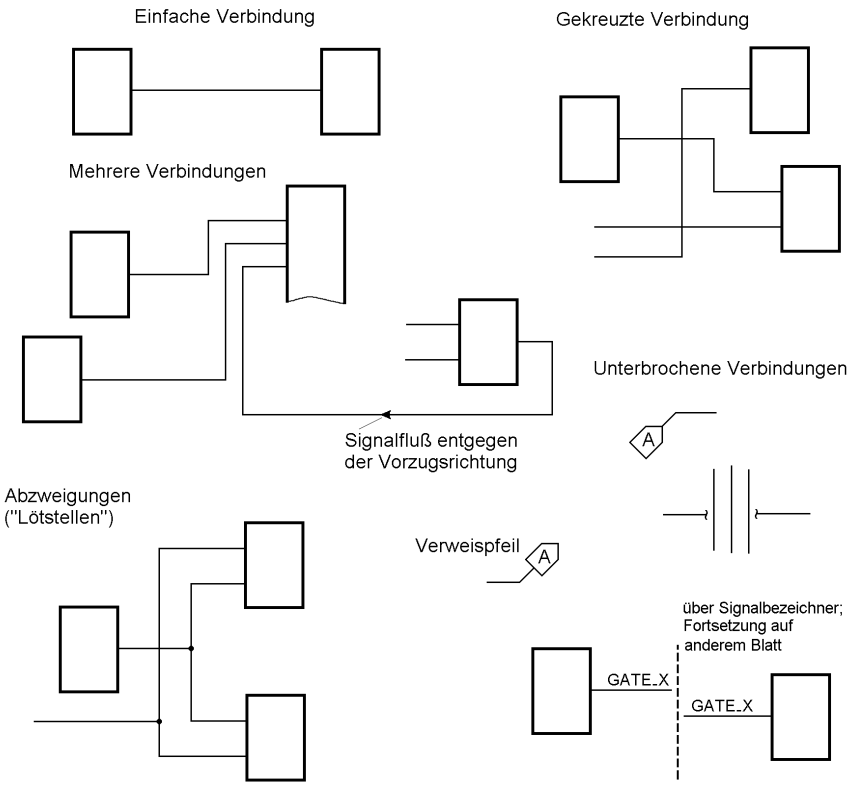


Abbildung 3.17 Verbindungen in Schaltplänen

Signalflußrichtungen
Heutzutage gilt grundsätzlich die *Vorzugsrichtung* von links nach rechts (und - wenn nötig - in der Senkrechten von oben nach unten). Signalflüsse in Gegenrichtung werden grundsätzlich durch Pfeile besonders gekennzeichnet. Namentlich dann, wenn Schaltkreise mit hundert und mehr Anschlüssen darzustellen sind, weicht man öfter von der Vorzugsrichtung ab (und nutzt bisweilen alle vier Seiten des Kästchen-Symbols), so daß in der Praxis auch vorwiegend senkrecht (von unten nach oben oder umgekehrt) orientierte Signalflüsse vorkommen.

Signalbezeichner
Signalbezeichner (Signal Identifiers) entsprechen den Variablennamen der Schaltalgebra, die wir aus Kapitel 1 kennen. In der Praxis haben nicht alle Verbindungen Signalbezeichner. Am Rand des Schaltplanes bzw. des einzelnen Blattes hat typischerweise jede hinein- oder herausführende Leitung einen Bezeichner. Mitten in der Zeichnung benennt man hingegen oft nur besonders wichtige Signale.

Masse und Speisespannungen
In "logischen" Schaltplänen verzichtet man oft auf die Darstellung der Spannungsversorgung. Wenn überhaupt, sind deren Besonderheiten, z. B. Stützkondensatoren, zusammengefaßt dargestellt (üblicherweise auf einem besonderen Blatt (Abbildung 3.18) oder in irgendeiner Ecke). Manchmal muß man aber auch solche Verbindungen in der Logik kenntlich machen. Abbildung 3.19 veranschaulicht die entsprechende Symbolik.

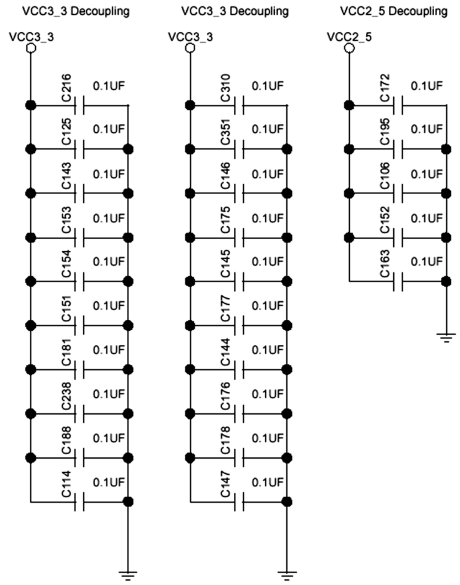


Abbildung 3.18 Zusammengefaßte Darstellung (auszugsweise) von Stützkondensatoren auf einem Motherboard (Intel)

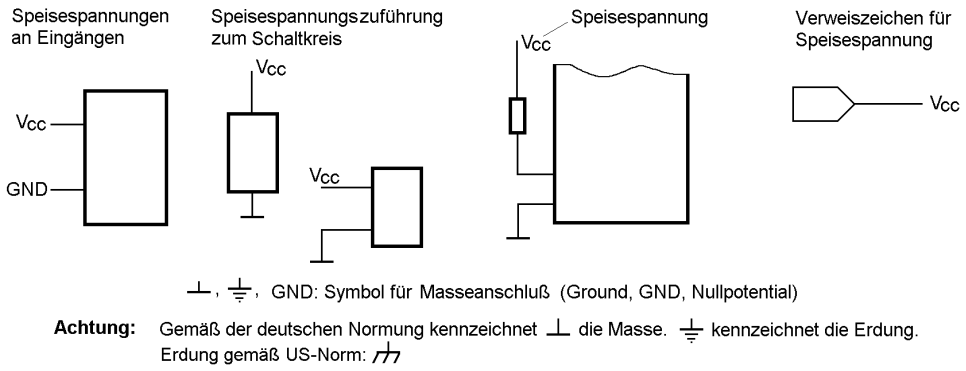


Abbildung 3.19 Masse und Speisespannungen in Logik-Schaltplänen

Masse und Erde

Wir merken uns zunächst: alle mit einem Massekennzeichen (z. B. mit einem kurzen waagerechten Strich oder einem Dreiecksymbol) versehenen Anschlüsse sind untereinander verbunden und liegen auf einem festen Pegel von 0 V. Genauer betrachtet haben die Masseverbindungen aber verschiedene Aufgaben:

- gemeinsame Rückleitung für alle Speiseströme[*)],
- gemeinsame Rückleitung für alle Signalströme[*)] (Ausnahme: differentielle Signalübertragung),
- gemeinsames Bezugspotential (0 V) für alle Signalpegel,
- Verbindung zum Erdpotential (Berührungsschutz (Schutzerdung), Potentialausgleich, Ableitung von Störungen).

Wegen dieser Mehrfachnutzung gibt es gelegentlich Probleme.

*): wir merken uns: (fast) alles, was in einer Schaltung irgendwo an Strom fließt, muß über die Masse zurück (die Masseverbindung schließt die Stromkreise).

Kabelbaumdarstellung

Im besonderen digitale Schaltungen sind oft durch komplizierte Verbindungen und durch viele einzelne Signalwege gekennzeichnet (Beispiel: ein Prozessor mit 64-Bit-Daten- und 32-Bit-Adreßbus). Solche Signalwege werden oft als "Kabelbaum" dargestellt (Abbildung 3.20). Der Kabelbaum selbst ist als dicke Linie gezeichnet, in die alle eingehenden Signale ein- und von der alle abgehenden ausmünden. Manchmal zeichnet man anstelle einer dicken - voll geschwärzten - Linie einen durch zwei dünne Linien begrenzten "Schlauch". In manchen Schaltplänen hat man sich bemüht, die Anzahl der Leitungen durch die Dicke des Kabelbaums und das Aus- und Einmünden von Leitungen bildhaft zu veranschaulichen (Abbildung 3.20 rechts), in manchen gibt man sich hingegen mit einer mehr schematischen Darstellung zufrieden (Abbildung 3.20 links). Die Signale sind entweder durchnumeriert oder einzeln mit Signalnamen bezeichnet. Um den Signalfluß im Detail zu erkennen, müssen wir den Kabelbaum insgesamt sowie die jeweiligen Leitungsnummern verfolgen.

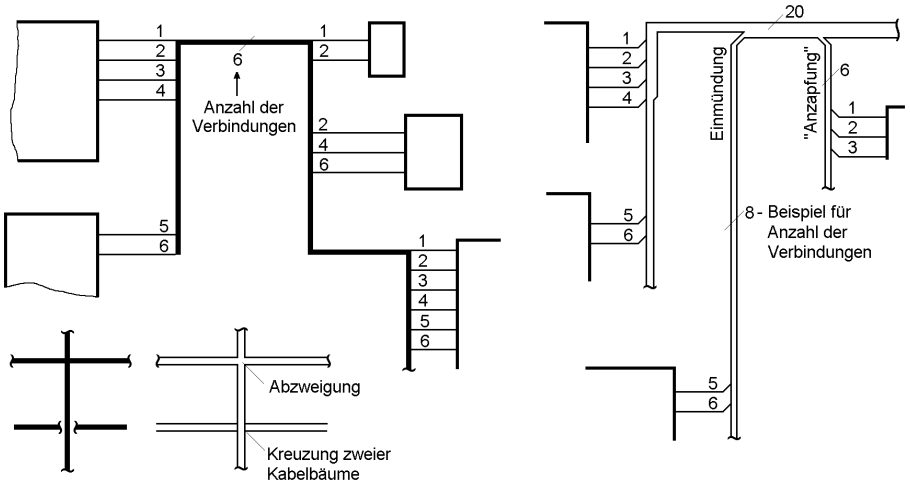


Abbildung 3.20 Kabelbaumdarstellungen

Hinweis:

Achten Sie darauf, ob Kabelbäume miteinander verbunden sind oder nicht. In nicht verbundenen (unabhängigen) Kabelbäumen werden auch die einzelnen Leitungen unabhängig voneinander durchnumeriert. Sind beispielsweise 2 Kabelbäume A, B *nicht* miteinander verbunden, so hat die Leitung Nr. 1 in A nichts mit der Leitung Nr. 1 in B zu tun (Abbildung 3.21).

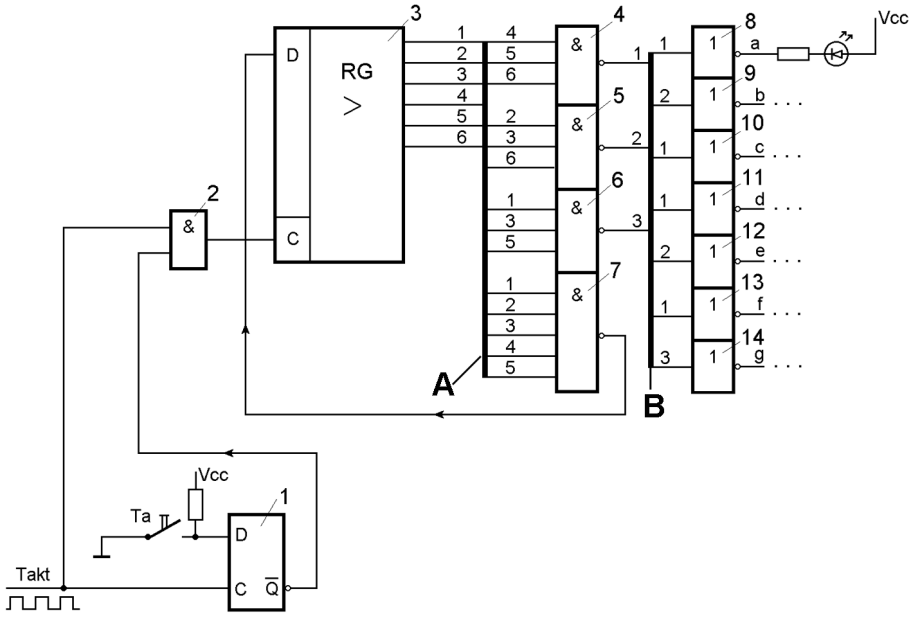


Abbildung 3.21 Schaltplan-Beispiel mit 2 Kabelbäumen

Erklärung:
In Kabelbaum A verbindet beispielsweise die Leitung Nr. 1 Funktionselement 3 mit den Funktionselementen 6 und 7; die Leitung Nr. 1 in Kabelbaum B verbindet hingegen Funktionselement 4 mit den Funktionselementen 8, 10, 11 und 13.

Bau-, Funktions- und Serviceschaltpläne
Ein *Bauschaltplan* soll die Verbindungen der einzelnen Bauelemente untereinander sowie erforderlichenfalls auch technische Einzelheiten erkennen lassen. Der *Funktionsschaltplan* (Stromlaufplan) ist dazu vorgesehen, die Funktionsweise der Einrichtung deutlich zu machen. Deshalb wird dort von technischen Einzelheiten abgesehen, beispielsweise von der gemeinsamen Unterbringung mehrerer Funktionselemente (z. B. von Gattern) in einem Schaltkreis. Im *Serviceschaltplan* sollen alle Angaben, die für das Verständnis der Funktionsweise sowie für Fehlersuche und Reparatur notwendig sind, in einer einzigen Darstellung zusammengefaßt sein. Einfache Funktionselemente werden dabei zumeist einzeln oder aufgelöst dargestellt, das heißt beispielsweise als einzelne Gatter und nicht als Teil eines Schaltkreises. Welches Funktionselement in welchem Schaltkreis enthalten ist, erkennt man aus der Schaltkreisbezeichnung, die entweder im Schaltsymbol oder außerhalb (daneben) zu finden ist.

Funktionelle (logische) und technische Anschlußbezeichnungen
Wenn wir uns in die Funktionsweise hineindenken wollen, brauchen wir die funktionellen (bzw. logischen) Anschlußbezeichnungen. Wir müssen, um den Signalfluß verfolgen zu können, Eingänge von Ausgängen unterscheiden. Darüber hinaus ist es eine große Erleichterung, wenn wir auf den ersten Blick Funktionen wenigstens grob erkennen können, wie Taktzuführung, Rücksetzen, Funktionsauswahl usw. Beim Messen brauchen wir hingegen die Position des jeweiligen Schaltkreis-Anschlusses am Gehäuse (die Pin-Nummer).

Diese wird meist außen am Schaltsymbol direkt über der betreffenden Leitung angegeben, gelegentlich aber auch im Schaltsymbol an dessen Rand. Die Numerierung und Zählweise ist eindeutig durch die Gehäuseform gegeben; sie ist entweder standardisiert oder muß aus dem betreffenden Datenblatt entnommen werden. Gelegentlich ist ein Schaltkreis auch nicht rein funktionell-symbolisch, sondern gemäß seiner Gehäuseform und Anschlußbelegung dargestellt (vgl. den 74HC132 in Abbildung 3.3 (Pfeil)).

Vertauschungsfälle
Einen hochintegrierten Schaltkreis kann man eigentlich gar nicht anders einsetzen als dies im Datenblatt bzw. im Anwendungshandbuch des Herstellers empfohlen wird ("Einsatz nach Kochbuch"). Hingegen müssen wir uns bei einfacheren Schaltkreisen durchaus auf Überraschungen gefaßt machen. Betrachten wir z. B. einen Bustreiber. Dessen Eingänge seien mit D1, D2 usw. durchnumeriert. Man erwartet nun, daß die Signale in der konkreten Schaltung so angeschlossen sind, wie es die geradezu suggestive Anschlußbezeichnung im Schaltsymbol nahelegt, daß also beispielsweise das Datenbit 0 an D1 angeschlossen ist, das Datenbit 1 an D2 usw. Es verhält sich aber nicht immer so! Schließlich ist dem Schaltkreis die Anschlußreihenfolge völlig gleichgültig. Und davon wird gelegentlich Gebrauch gemacht, um eine einfachere Leiterzugführung auf der Leiterplatte zu erreichen. Das Vertauschen ist auch in weniger einsichtigen Fällen möglich, so an Multiplexern, Decodern und Vergleichern. Man muß dann nur funktionell zusammengehörende Signale auch zusammen tauschen. Wenn man also beispielsweise an einem Multiplexer die Reihenfolge der Dateneingänge vertauscht, müssen die Auswahlsignale entsprechend mitgedreht werden.

3.1.2. Blockschaltbilder

Solche Schaltbilder sollen einen Überblick über Aufbau und Funktionsweise vermitteln. Die Symbolik ist kaum standardisiert. Im einfachsten Fall sind die Blöcke (Funktionseinheiten) in Kästchenform dargestellt und durch dünne Vollinien untereinander verbunden. Aus der Verbindungsdarstellung ist nicht ersichtlich, wieviele Signale die jeweilige Verbindung im einzelnen umfaßt. Soll das Blockschaltbild nähere Einzelheiten veranschaulichen, so deutet man oft die Anzahl der Signale durch die Dicke der verbindenden Linien an (ähnlich der Kabelbaumdarstellung). Auch unterscheidet man häufig zwischen rein kombinatorisch und sequentiell wirkenden Funktionsblöcken (kombinatorische Schaltungen: Trapezform; sequentielle Schaltungen und Speicheranordnungen: Rechteckform). Insbesondere in Funktionsbeschreibungen und Systemhandbüchern (Theory of Operation Manuals) finden wir oft recht detaillierte Blockschaltbilder, die sogar einzelne Gatter und andere Funktionselemente enthalten (das, worauf es ankommt, ist bis aufs Gatter aufgelöst, die verbleibende Hardware ist nur grob in Blockform angedeutet). Manchmal ist aus solchen Darstellungen auch die genaue Anzahl von Signalleitungen ersichtlich. Abbildung 3.22 zeigt verschiedene Darstellungselemente von Blockschaltbildern.

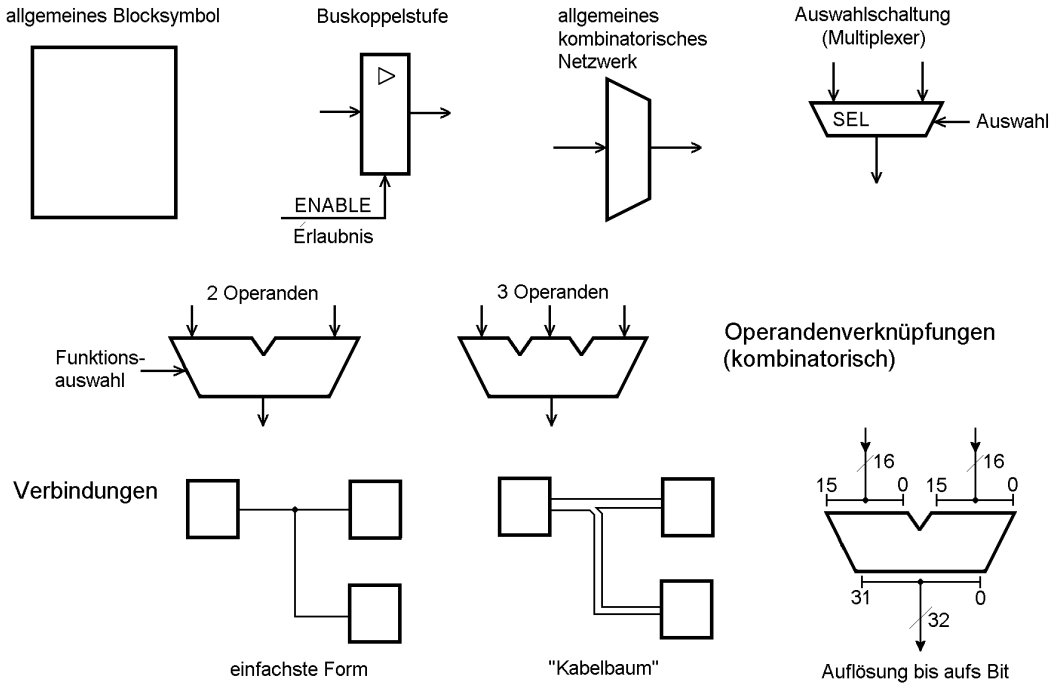


Abbildung 3.22 Darstellungselemente von Blockschaltbildern

3.1.3. Bestückungsplan und Stückliste

Zum Fertigen, Fehlersuchen und Reparieren brauchen wir eine eindeutige Kennzeichnung der Bauelemente. Diese Angaben sind im allgemeinen im Bestückungsplan*) und in der Stückliste*) enthalten (Abbildungen 3.23, 3.24). Der Bestückungsplan ist eine zeichnerische (oder auch photographische) Darstellung der Leiterplatte, aus der die Lage der einzelnen Bauelemente erkennbar ist.

*): typische englischsprachige Fachbegriffe: Bestückungsplan = Component Locator, Stückliste = Parts List oder Bill of Materials (BOM).

Gelegentlich finden wir überblicksmäßige Bestückungspläne, die nur die Lage jener Bauelemente zeigen, auf die es im betreffenden Zusammenhang gerade ankommt. Beispiele: die typischen Übersichtsdarstellungen von Motherboards und Steckkarten.

Gute, aber nicht allgemein übliche Alternativen zum Bestückungsplan sind auf die Leiterplatte gedruckte Bezugszeichen für alle Bauelemente (die allerdings auch im bestückten Zustand sichtbar sein sollten...) bzw. aufgedruckte waagerechte und senkrechte Koordinaten, beispielsweise in einer Teilung von 2,54 mm (= 0,1"). Man kann so für jedes Bauelement seine Koordinaten-Position auf der Leiterplatte im Schaltplan angeben, z. B. bezogen auf die linke obere Ecke des jeweiligen Bauelements. Dann läßt es sich leicht finden, ohne extra einen Bestückungsplan heranziehen oder auf der Leiterplatte herumsuchen zu müssen.

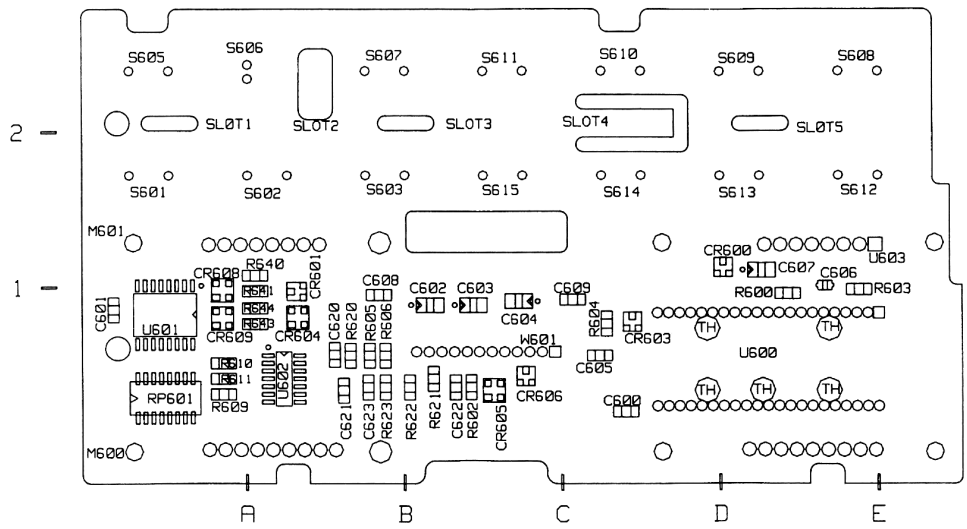


Abbildung 3.23 Auszug aus einem Bestückungsplan (Agilent Technologies)

1	2	3	4	5	6
Reference Designation	**HP Part Number**	**Qty**	**Part Description**	**Mfr. Code**	**Mfr. Part Number**
C602	0160-5945	3	Capacitor-Fxd 0.01uF 50 V	04222	08055C103KAT A
C603-C605	0160-6497	10	Capacitor-Fxd 0.1uF 25 V	04222	12065C104KAT A
R614-R617	0699-1344	4	Resistor 10 ±1% .125W TKF TC=0±100	2M627	MCR18-F-X-10R0
R618	0699-1378	1	Resistor 2.61K 1%	2M627	MCR18-F-X-2611
R619	0699-1391		Resistor 10K ±1% .125W TKF TC=0±100	2M627	MCR18-F-X-1002
U601	1826-2264	1	IC-Power Management	04713	MC34064D-5
U602	34401-88813	1	Programmed 1820-8905 8-Bit MCU	28480	34401-88813
U603	1820-5330	1	IC-75518 Interface Driver, Bipolar	01295	SN75518FN
U604	1820-5562	1	IC-MC74HC02AD	01295	SN74HC02D
U605	1820-4966	1	IC-MC74HC74AD	01295	SN74HC74D
U606	2090-0296	1	Display - Vacuum Fluorescent	11908	CP3033A
U607	1826-1528	1	IC 339 Comparator	27014	LM339M
U608	1820-6756	1	74HC299-Shift Register 8 Bit Parallel	04713	MC74HC299D
U609	1826-1402	1	IC-Voltage Regulator 4.8/5.2V	04713	MC78L05ACD
W601	34401-61602	1	Cable Assembly, 8.8L	28480	34401-61602
Y601	0410-4009	1	Crystal 12MHz +1-0.8%	00830	PBRC-12.0BRN07

Abbildung 3.24 Auszug aus einer Stückliste (Agilent Technologies)

Erklärung:
1 - Bezugszeichen (u. a. zum Schaltplan und Bestückungsplan); 2 - Bezeichnung; 3 - firmeninterne Teilnummer; 3 - Stückzahl (Qty = Quantity); 4 - Bezeichnung; 5 - Lieferanten- bzw. Herstellercode*); 6 - Teilnummer des Lieferanten bzw. Herstellers (gelegentlich wichtig (Ersatzteilbestellung)).

*): Mfr. = Manufacturer = Hersteller des jeweiligen Teils. Die Codeangabe (Pos. 5) bezieht sich auf ein Verzeichnis, in dem die Lieferanten bzw. Hersteller der Einzelteile vermerkt sind.

Bezugszeichen der Bau- und Funktionselemente
Bezugszeichen (Component Identifiers, Reference Designators) sind oft herstellerspezifisch (laufende Nummern, Buchstaben-Ziffern-Kombinationen, mnemonische Bezeichnungen).

Die gebräuchlichste Verfahrensweise: man kennzeichnet jedes Element entsprechend seiner Art durch Kennbuchstaben und numeriert die Elemente mit gleichen Kennbuchstaben fortlaufend durch. Insbesondere im englischen Sprachraum sind die Kennbuchstaben gemäß Tabelle 3.2 üblich.

Kennbuchstabe	Element
A	Baugruppe (Assembly)
BT	Batterie
C	Kondensator
D, CR	Diode
F	Sicherung (Fuse)
FL	Filter
J	Steckverbinder (Jack; dtsch. auch: Bu oder B) bzw. Steckbrücke (Jumper)
L	Spule, Induktivität
Q	Transistor; dtsch. auch T, Tr, Trs
R	Widerstand
S	Schalter
T	Transformator
TP	Prüf- bzw. Meßpunkt (Test Point; dtsch. auch MP)
U, IC	Integrierter Schaltkreis
V	Elektronenröhre (z. B. Bildröhre; diese auch: CRT)
Y	Quarz

Tabelle 3.2 Gebräuchliche Kennbuchstaben für Bau- und Funktionselemente

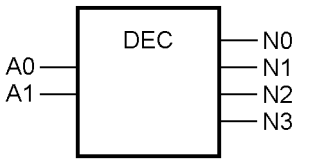
3.2. Ausdrucksmittel der Funktionsbeschreibung

Die Struktur der Digitalschaltungen wird üblicherweise durch Schaltpläne beschrieben. Ebenso wichtig sind aber die Wirkungsweise und - vor allem bei sequentiellen Schaltungen - das zeitliche Verhalten. Wir geben im folgenden einen Überblick über entsprechende Darstellungsmittel.

3.2.1. Gleichungen und Listen

Wahrheitstabellen
Wahrheitstabellen (Truth Tables) werden in Datenblättern und Funktionsbeschreibungen verwendet, um vergleichsweise einfache kombinatorische Schaltungen darzustellen. Der Vorteil: für sämtliche Eingangsbelegungen ist das Verhalten sofort ersichtlich. Der Nachteil: Eine Schaltung mit n Eingangssignalen erfordert eine Wahrheitstabelle mit 2^n Einträgen. Bei mehr als ca. 6 Eingangssignalen ist dieses Darstellungsmittel kaum mehr praktikabel. Neben der üblichen Tabellenform (vgl. Kapitel 1) gibt es auch Wahrheitstabellen in Darstellungsweisen, die an Programmiersprachen angelehnt sind (Abbildung 3.25).

Beispiel: 2-Bit-Decoder (1-aus-4)




```
field Input  = (A1 . . 0)
field Output = (N3 . . 0)
table Input  =>  Output
{
  'b' 0 0  =>  'b' 0 0 0 1;
  'b' 0 1  =>  'b' 0 0 1 0;
  'b' 1 0  =>  'b' 0 1 0 0;
  'b' 1 1  =>  'b' 1 0 0 0;
}
```

Abbildung 3.25 Wahrheitstabellen-Darstellung in einem Entwicklungssystem

Belegungslisten
In Belegungslisten sind die Eingangsbelegungen einer kombinatorischen Schaltung aufgeführt, die eine bestimmte Ausgangsbelegung (0 oder 1) bewirken. Es gibt (vgl. Kapitel 1) binäre und ternäre Belegungslisten. Solche Listen werden in der Dokumentation recht selten verwendet. Sie eignen sich aber gut, um - beim Erproben oder beim meßtechnischen Fehlersuchen - das Verhalten von Schaltungen zu erfassen. Entsprechende Listen- oder Tabellendarstellungen werden u. a. von Logikanalysatoren, Entwicklungssystemen und Simulationsprogrammen geliefert.

Zustands- oder Automatentabellen
Zustands- oder Automatentabellen sind gleichsam Wahrheitstabellen für sequentielle Schaltungen, die den *Folgezustand* angeben, und zwar in Abhängigkeit von Eingangsbelegung und aktuellem Zustand. Der Folgezustand wird dabei durch eine hochgesetzte Eins, durch einen Apostroph, durch die Angabe (t+1) oder auf ähnliche Weise gekennzeichnet. Beispiele für Variable, die einen Folgezustand repräsentieren: x^1, y', z(t+1)..

Boolesche Gleichungen
Boolesche Gleichungen (vgl. Kapitel 1) werden gelegentlich verwendet, um kombinatorische Schaltungen zu beschreiben (das betrifft im besonderen programmierbare Schaltungen). Sie werden oft in einer herstellerspezifischen Symbolik angegeben. (Eine verbreitete Symbolik: "&" = UND, "#" = ODER, "!" = NICHT. Vgl auch Tabelle 1.12. Dort sind jedoch nicht alle Symbole erfaßt, die in der Praxis vorkommen können.)

Zustands- bzw. Automatengleichungen
Das sind Boolesche Gleichungen mit zeitabhängigen Variablen. Sie beschreiben, welche Belegung die Speicherelemente im jeweils nächsten Takt einnehmen (Folgezustand; siehe auch unter "Zustands- oder Automatentabellen"). Als Beispiel vgl. Abbildung 3.8.

3.2.2. Zustandsgraphen (Zustandsdiagramme, State Diagrams)

Abbildung 3.26 veranschaulicht das Prinzip: jeder Zustand wird durch einen *Knoten* (Node) dargestellt, und die Zustandsübergänge werden durch gerichtete Verbindungen zwischen den Knoten (*Kanten;* Edges) angegeben. Typischerweise ist an jeder Verbindung (Kante) die Bedingung angetragen, die den jeweiligen Zustandswechsel auslöst (vgl. auch Abbildung 1.9). Solche Bedingungen werden als Ereignisse (Events) bezeichnet. Ist keine Bedingung eingezeichnet, so wird der Übergang in der jeweils nächsten Taktperiode zwangsläufig ausgeführt. Eine Rückführung auf denselben Knoten bedeutet, daß der aktuelle Zustand beibehalten wird, solange kein Ereignis wirksam ist, um ihn zu verlassen.

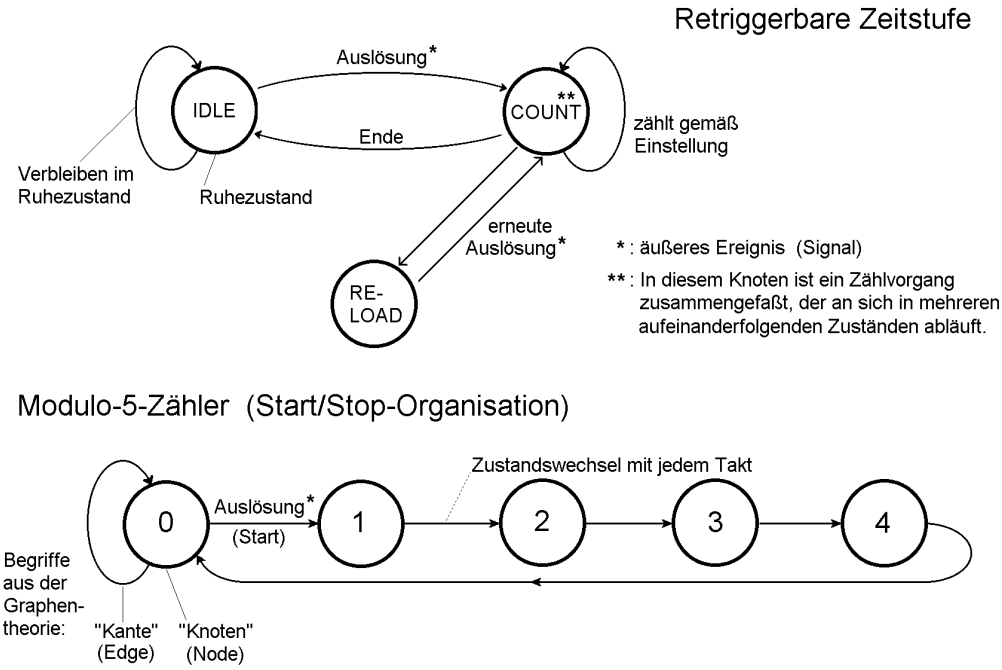


Abbildung 3.26 Zustandsgraphen (Beispiele). Vgl. weiterhin die Abbildungen 3.7 bis 3.9

Zustandsgraphen werden gern verwendet, um das Verhalten einfacher bis mittelmäßig komplizierter Schaltungen zu dokumentieren. Wirklich exakte Zustandsgraphen werden bald unübersichtlich. Deshalb sind Vereinfachungen üblich: So kann man fortlaufende Zustandswechsel (taktweises Zählen, Schieben usw.) in einem einzigen Knoten zusammenfassen (wie auch in Abbildung 3.16 gezeigt). Weiterhin bietet es sich gelegentlich an, einen komplexen, unübersichtlichen Zustandsgraphen in mehrere einfachere zu zerlegen. Komplizierte Zustandsübergänge beschreibt man oft mit gesondert angegebenen Booleschen Gleichungen (vgl. die Abbildungen 3.7 und 3.8).

Hinweis:
Abbildung 3.8 bezieht sich auf den jeweils aktuellen Zustand gemäß Abbildung 3.7. Die einzelnen Gleichungen geben an, unter welchen Bedingungen die State Machine in welchen Folgezustand übergeht. Beispiel: die State Machine befindet sich derzeit im Zustand S_DATA.

Es sind insgesamt 3 Zustandsübergänge möglich:

1. Verbleiben im Zustand S_DATA, wenn *goto S_DATA* = 1,
2. Übergang in den Zustand BACKOFF, wenn *goto BACKOFF* = 1
3. Übergang in den Zustand TURN_AR, wenn *goto TURN_AR* = 1.

3.2.3. Flußdiagramme (Flowcharts, Programmablaufpläne)

Flußdiagramme werden gelegentlich in Funktionsbeschreibungen verwendet, um Abläufe zu veranschaulichen. Abbildung 3.27 gibt einen Überblick über die übliche Symbolik. In der Vergangenheit wurden sie manchmal als exaktes Beschreibungsmittel in Standards, Wartungshandbüchern usw. eingesetzt (Abbildung 3.28). Das ist aber aus der Mode gekommen (Arbeitsaufwand, Platzbedarf, Überschaubarkeit - die heutzutage üblichen komplizierten Funktionen würden, ähnlich Abbildung 3.28 dokumentiert, eine Vielzahl von Seiten erfordern).

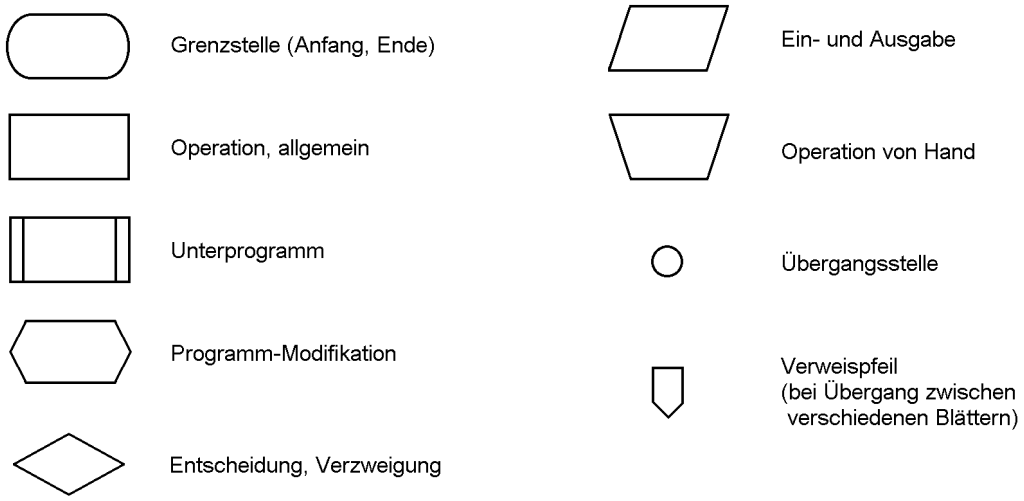


Abbildung 3.27 Symbole in Flußdiagrammen (Auswahl)

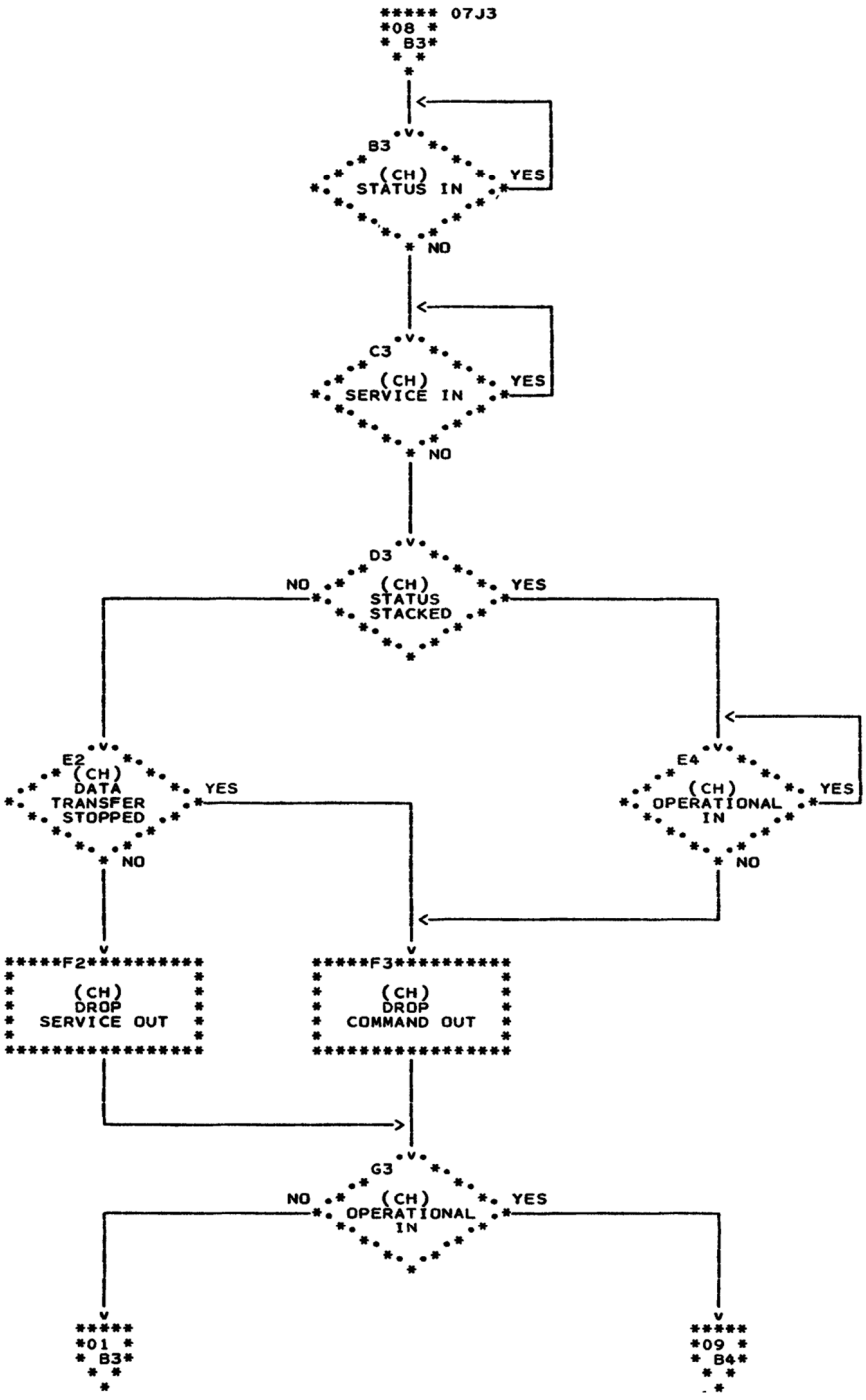


Abbildung 3.28 Ausschnitt aus einem Flußdiagramm (IBM)

Erklärung:
Dieser Ausschnitt dokumentiert eines der Signalspiele am E-A-Interface des Systems /360. In den Verzweigungskästchen werden Interfacesignale und Maschinenzustände abgefragt, in den Operationskästchen werden Signale geschaltet und Maschinenzustände verändert.

Hinweis:
Diese Abbildung wurde - in den 60er Jahren - mittels Schnelldrucker erzeugt. Dabei wurden die Symbole aus Sternchen zusammengestückelt.

3.2.4. Aktivitätsdiagramme und -listen

Aktivitätsdiagramme und Aktivitätslisten dienen dazu, das Zusammenarbeiten mehrerer Einrichtungen zu veranschaulichen, z. B. zwischen einem Prozessor, einer Interfacesteuerung und einem peripheren Gerät. Der grundsätzliche Aufbau (Abbildung 3.29): für jede der beteiligten Einrichtungen ist eine Spalte vorgesehen. Die Zeitachse verläuft von oben nach unten. Die einzelnen Aktivitäten sind jeweils eingetragen, und es ist durch Pfeile gekennzeichnet, welche nachfolgenden Aktivitäten in den jeweils anderen Einrichtungen ausgelöst werden. So lassen sich funktionelle Abhängigkeiten gut darstellen (Abbildung 3.10 zeigt ein Praxisbeispiel). Zum Dokumentieren von Einzelheiten sind solche Listen allerdings weniger geeignet.

Hinweis:
Interpretieren Sie in ein solches Diagramm nicht zuviel hinein: Sie dürfen keineswegs Längen oder Abstände messen bzw. untereinander vergleichen und daraus auf zeitliche Größenordnungen schließen!

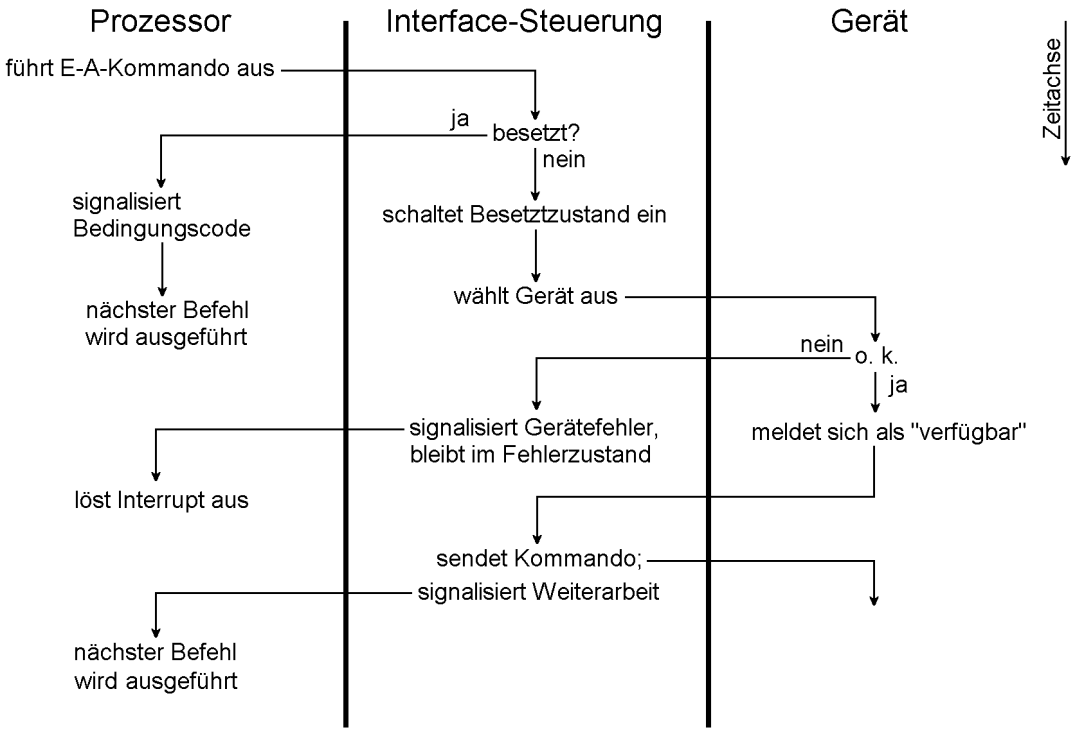


Abbildung 3.29 Aktivitätsliste (Beispiel)

3.2.5. Impulsdiagramme (Taktdiagramme)

Impulsdiagramme (Pulse Diagrams, Timing Diagrams) geben den zeitlichen Verlauf von Signalen mehr oder weniger symbolisch wieder (Abbildung 3.30; vgl. auch Abbildung 2.11).

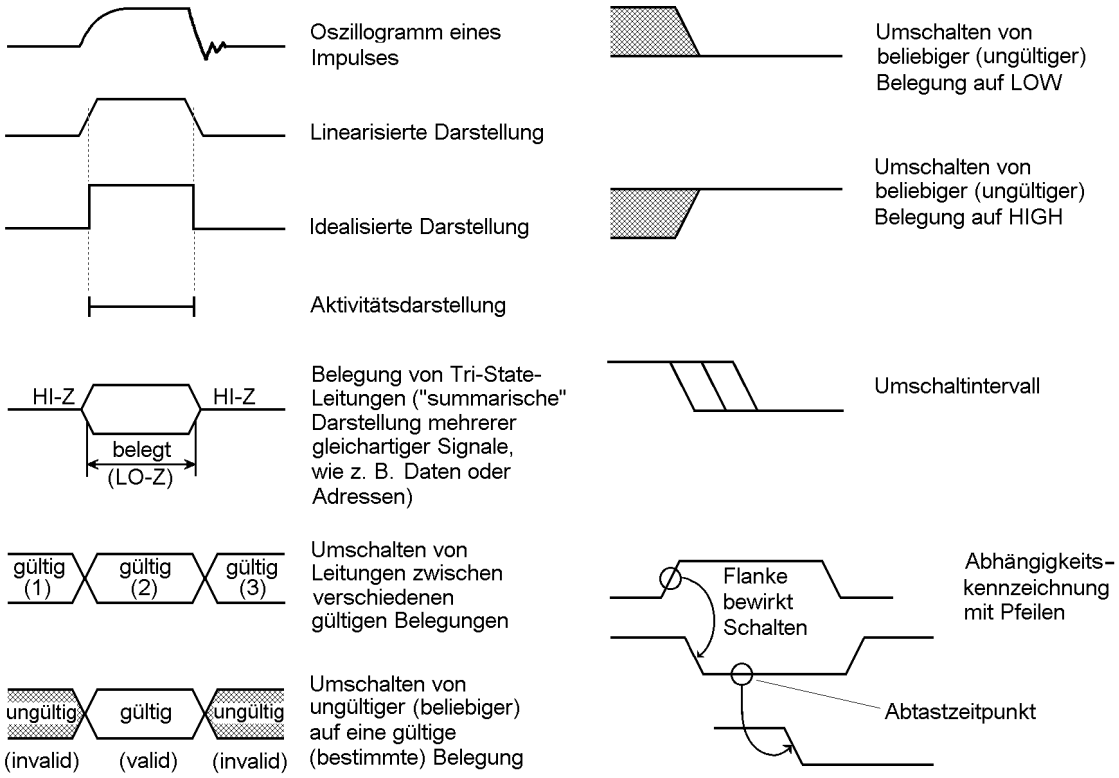


Abbildung 3.30 Symbolische Darstellungen in Impulsdiagrammen

Oszillogrammdarstellung
Das Oszillogramm zeigt den genauen Signalverlauf, wie er in der Schaltung tatsächlich mit einem Oszilloskop gemessen werden kann. Solche Oszillogrammbilder sind seit langem in der Servicedokumentation üblich. So enthalten Serviceschaltpläne von Fernsehgeräten vielfach Oszillogrammbilder besonders wichtiger Signale. Wenn es darauf ankommt, Einzelheiten des elektrischen Verhaltens zu zeigen, sind solche Darstellungen unentbehrlich. In der Digital- und Computertechnik würde eine vollständige Dokumentation auf dieser Grundlage allerdings sehr aufwendig werden, und es ist, zumindest für den weniger Geübten, auch nicht immer leicht, die charakteristischen Signalverläufe von exemplarabhängigen Abweichungen, geringfügigen Störungen (die nicht schaden) usw. zu unterscheiden. Deshalb beschränkt man sich auf besonders typische Signale (Takte, Ansteuersignale von DRAMs usw.).

Linearisierte Darstellung
Die linearisierte Darstellung setzt für den Anstieg und Abfall von Signalen (die Signalflanken) schräge Linien an. Diese Darstellungsweise ist vor allem in Datenblättern gebräuchlich.

Idealisierte Darstellung
Die idealisierte Darstellung vernachlässigt die Anstiegs- und Abfallzeiten; Flanken werden als senkrechte Linien dargestellt, Impulse sind Rechteckimpulse. Solche Impulsdiagramme werden z. B. von Logik-Simulationsprogrammen geliefert, man kann sie aber auch meßtechnisch aufnehmen (Oszilloskop, Logikanalysator).

Aktivitätsdarstellung
Die Aktivitätsdarstellung kennzeichnet *aktive* Signale durch eine (üblicherweise dickere) Linie. Aus den anderen Darstellungsformen ist der Signalwechsel zwischen High und Low direkt ersichtlich (High ist weiter oben, Low weiter unten). Hingegen ist die Aktivitätsdarstellung typischerweise so ausgelegt, daß die *aktiven* Belegungen der jeweiligen Signale hervorgehoben werden: ist das Signal aktiv, so erscheint die Linie, ist es inaktiv, so fehlt sie. Ist beispielsweise ein Signal als "aktiv Low" definiert, so erscheint eine entsprechende Linie in all den Zeitabschnitten, in denen es mit Low-Pegel belegt ist.

Impulsdiagramme in Datenblättern und Standards
Die Impulsdiagramme enthalten Zeitkennwerte im Sinne von Maßangaben (ähnlich einer technischen Zeichnung). Die zugehörigen Zahlenwerte sind typischerweise in Kennwerttabellen zusammengefaßt. Die Abbildungen 3.5 und 3.6 mögen als erste Beispiele dienen.

Hinweise:

1. Das Signalverhalten auf mehreren gleichartigen Leitungen (z. B. für Daten oder Adressen) wird oft summarisch dargestellt, wobei es nur darum geht, zu welchen Zeiten gültige bzw. ungültige Belegungen vorliegen.
2. Erschließen Sie sich genaue Zeitverhältnisse *nie* allein aus dem Impulsdiagramm! Impulsdiagramme sind fast immer (wenn nicht ausdrücklich anders angegeben) *unmaßstäbliche* Darstellungen. Zeitangaben nur aus den jeweiligen Kennwerttabellen entnehmen! (Es ist hier ähnlich wie in der allgemeinen Werkstattpraxis, wo allein die Bemaßungen gelten und es verpönt ist, mit dem Zollstock in der Zeichnung herumzumessen.)

Anhang 1: Rechenregeln der Schaltalgebra

Verwendung von Klammern

1. Assoziativität: bei gleichartigen Verknüpfungen ist die Klammeranordnung bzw. die Reihenfolge der Ausführung gleichgültig:

   $a \cdot (b \cdot c) = a \cdot b \cdot c = (a \cdot b) \cdot c$. In Kurzform: $a(bc) = abc = (ab)c$

   $a \vee (b \vee c) = a \vee b \vee c = (a \vee b) \vee c$

2. Kommutativität: bei gleichartigen Verknüpfungen ist die Reihenfolge der Variablen gleichgültig:

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

   $a \vee b = b \vee a$

3. Distributivität: eine Funktion vor einer Klammer ist auf alle Variablen in der Klammer anzuwenden; die jeweiligen Ergebnisse sind gemäß der eingeklammerten Funktion zu verknüpfen (vgl. das schulmäßige Ausmultiplizieren von Klammerausdrücken, beispielsweise a (b + c) = ab + ac):

   $a \cdot (b \vee c) = (a \cdot b) \vee (a \cdot c)$. In Kurzform: $a(b \vee c) = ab \vee ac$

   $a \vee (b \cdot c) = (a \vee b) \cdot (a \vee c)$. In Kurzform: $a \vee (bc) = (a \vee b)\,(a \vee c)$

Allgemein (wobei die Symbole *, # wahlweise für UND, ODER, Antivalenz bzw. Äquivalenz ($\cdot$, $\vee$,$\oplus$, $\equiv$) stehen):

- für gleichartige Verknüpfungen:
  - a * (b * c) = a * b * c = (a * b) * c (Assoziativität),
  - a * b = b * a (Kommutativität),
- für verschiedenartige Verknüpfungen: a * (b # c)= (a * b) # (a * c) (Distributivität).

Vorrang der Konjunktion
Bei Schreibweise ohne Konjunktionssymbol hat die Konjunktion Vorrang, sofern nicht durch Klammern eine andere Reihenfolge zum Ausdruck gebracht wird:

$abc \vee def = (a \cdot b \cdot c) \vee (d \cdot e \cdot f)$

Negation durch Überstreichen
Eine lückenlose Überstreichung mehrerer Symbole entspricht einer Einklammerung:

$\overline{a \vee b \vee c} = /(a \vee b \vee c)$

Doppelte Negation

$\bar{\bar{a}} = a$ (Negation der Negation = Bejahung (Identität))

Verknüpfungen mit sich selbst (Idempotenz)

$a \cdot a = a;\ a \vee a = a$

Verknüpfungen mit Festwerten

$\bar{0} = 1;\ \bar{1} = 0$

$0 \cdot 0 = 0;\ 0 \cdot 1 = 0;\ 1 \cdot 0 = 0;\ 1 \cdot 1 = 1;\ 0 \cdot a = 0;\ 1 \cdot a = a$

$0 \vee 0 = 0;\ 0 \vee 1 = 1;\ 1 \vee 0 = 1;\ 1 \vee 1 = 1;\ 0 \vee a = a;\ 1 \vee a = 1$

$0 \oplus 0 = 0;\ 0 \oplus 1 = 1;\ 1 \oplus 0 = 1;\ 1 \oplus 1 = 0;\ 0 \oplus a = a;\ 1 \oplus a = \bar{a}$

$0 \equiv 0 = 1;\ 0 \equiv 1 = 0;\ 1 \equiv 0 = 0;\ 1 \equiv 1 = 1;\ 0 \equiv a = \bar{a};\ 1 \equiv a = a$

Reduktion auf Festwerte

$a \cdot \bar{a} = 0;\ a \vee \bar{a} = 1;\ a \oplus \bar{a} = 1;\ a \equiv \bar{a} = 0;\ a \oplus a = 0;\ a \equiv a = 1$

$a \oplus a \oplus a = 1;\ a \oplus a \oplus a \oplus a = 0$ (Variablenanzahl ungerade: 1, gerade: 0)

$a \equiv a \equiv a = 0;\ a \equiv a \equiv a \equiv a = 1$ (Variablenanzahl ungerade: 0, gerade: 1)

Antivalenz-Rechenregeln

$\bar{a} \oplus b = a \oplus \bar{b} = \overline{a \oplus b} = a \equiv b$

$\bar{a} \oplus \bar{b} = a \oplus b$

Einsetzungsregel
Anstelle einer Variablen in einer Schaltfunktion kann eine beliebige Schaltfunktion eingesetzt (substituiert) werden:

$f_1(a, b, c,...) = f_1(a, f_2, c,...)$

Hier wurde anstelle der Variablen *b* die Schaltfunktion f_2 eingesetzt. f_2 kann sowohl von Variablen abhängen, die auch in f_1 vorkommen, als auch von weiteren Variablen.

Eine Schaltfunktion in einer Schaltfunktion heißt auch Teilschaltfunktion. Man kann Teilschaltfunktionen und einzelne Variable wechselseitig einsetzen.

Ersetzungsregel
Jede Teilschaltfunktion f_i in einer Schaltfunktion f kann durch eine äquivalente Teilschaltfunktion f_i^* ersetzt werden. Zwei Schaltfunktionen f_i und f_i^* sind äquivalent, wenn sie die gleiche Wahrheitstabelle haben.

$f(a, b, f_1, f_2, f_3) = f(a, b, f_1^*, f_2^*, f_3^*)$; wenn gilt $f_i = f_i^*$

DeMorgansche Regeln
Diese beschreiben die Dualität zwischen UND und ODER. Sie gelten für einzelne Variable sowie für Schaltfunktionen gleichermaßen (Einsetzungs- und Ersetzungsregel):

1. DeMorgansche Regel: negiertes UND = ODER der einzeln negierten Variablen.

$\overline{f_1 f_2} = \overline{f_1} \vee \overline{f_2}$; $\quad \overline{\overline{f_1}\,\overline{f_2}} = f_1 \vee f_2$

2. DeMorgansche Regel: negiertes ODER = UND der einzeln negierten Variablen.

$\overline{f_1 \vee f_2} = \overline{f_1}\,\overline{f_2}$; $\quad \overline{\overline{f_1} \vee \overline{f_2}} = f_1 f_2$

Kürzungsregeln

1. Kürzungsregel: $f_1 \vee f_1 f_2 = f_1$

$f_1 \cdot (f_1 \vee f_2) = f_1$

2. Kürzungsregel: $f_1 f_2 \vee f_1 \overline{f_2} = f_1$

$(f_1 \vee f_2) \cdot (f_1 \vee \overline{f_2}) = f_1$

3. Kürzungsregel: $f_1 \vee \overline{f_1} f_2 = f_1 \vee f_2$

$f_1 \cdot (\overline{f_1} \vee f_2) = f_1 f_2$

(f_1, f_2 können Variable oder Teilschaltfunktionen sein (Einsetzungs- und Ersetzungsregel).)

Auf Variable angewandte Kürzungsregeln (Auswahl):

$a \cdot (a \vee b) = a$; $\quad a \cdot (\overline{a} \vee b) = a \cdot b$

$\overline{a} \cdot (a \vee b) = \overline{a} \cdot b$; $\quad \overline{a} \cdot (\overline{a} \vee b) = \overline{a} \vee b$

$a \vee (a \cdot b) = a$; $\quad a \vee (\overline{a} \cdot b) = a \vee b$;

$\overline{a} \vee (a \cdot b) = \overline{a} \vee b$; $\quad \overline{a} \vee (\overline{a} \cdot b) = \overline{a} \vee b$

Elementarfunktionen mit mehr als 2 Variablen

UND
$a \cdot b \cdot c \cdot ... = a \cdot (b \cdot (c \cdot ...))$. Wahrheitswert 1: wenn alle Variable a, b, c... den Wahrheitswert 1 haben. Wahrheitswert 0: hierzu genügt es, daß eine der Variablen den Wahrheitswert 0 hat.

ODER
$a \vee b \vee c... = a$ v $(b \vee (c \vee ...))$. Wahrheitswert 0: wenn alle Variable a, b, c... den Wahrheitswert 0 haben. Wahrheitswert 1: hierzu genügt es, daß eine der Variablen den Wahrheitswert 1 hat.

NAND
$\overline{a \cdot b \cdot c \cdot ...}$. Wahrheitswert 0: wenn alle Variable a, b, c... den Wahrheitswert 1 haben. Wahrheitswert 1: hierzu genügt es, daß eine der Variablen den Wahrheitswert 0 hat.

NOR
$\overline{a \vee b \; c \vee ...}$. Wahrheitswert 1: wenn alle Variable a, b, c... den Wahrheitswert 0 haben. Wahrheitswert 0: hierzu genügt es, daß eine der Variablen den Wahrheitswert 1 hat.

Antivalenz
$a \oplus b \oplus c \oplus ... = a \oplus (b \oplus (c \oplus ...))$. Wahrheitswert 0: wenn die Anzahl der Einsen gerade ist. Wahrheitswert 1: wenn die Anzahl der Einsen ungerade ist.

Äquivalenz
$a \equiv b \equiv c... = a \equiv (b \equiv (c \equiv ...))$.Wahrheitswert 0: wenn die Anzahl der Einsen ungerade ist. Wahrheitswert 1: wenn die Anzahl der Einsen gerade ist.

Anhang 2: Internet-Adressen

1. Hersteller von Logikschaltkreisen

http://www.idt.com

http://www.fairchildsemi.com (Fairchild Semiconductor)

http://www.onsemi.com (ON Semiconductor; führt das Bauelementeprogramm von Motorola weiter (ECL, CMOS))

http://www.pericom.com

http://www.ti.com (Texas Instruments)

2. Weitere Schaltkreishersteller

http://www.cypress.com (U. a. Taktschaltkreise und programmierbare Logik)

http://www.nsc.com (National Semiconductor. Die Logikbaureihen wurden von Fairchild Semiconductor übernommen und werden dort weitergeführt, aber breites Angebot an Treiber- und Wandlerschaltkreisen)

http://www.okisemi.com (Oki; GaAs-Schaltkreise, ASICs)

3. Hersteller programmierbarer Logikschaltkreise

http://www.actel.com

http://www.altera.com

http://www.latticesemi.com

http://www.xilinx.com

4. Standardisierungsgremien

http://www.jedec.org

http://www.ecma.ch (ECMA-Standards u. a. zur Aufzeichnung auf CDs und DVDs)