Heft 01
Grundlagen der Digitaltechnik

Stand: 1.03

- gedruckt am 14. 11. 01 -

1. Analoge, digitale und binäre Arbeitsweise

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. (Wir werden uns diesem üblichen Sprachgebrauch anschließen.)

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 Wertebereich (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, das heißt, daß ausgeschlossen ist, daß irgendeine Wertdarstellung jemals einen der "verbotenen" Bereiche belegt. 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 10-wertigen Informationsdarstellung im Rahmen eines Spannungsbereichs von 0 bis 10 V. Es leuchtet sofort ein, daß es wesentlich aufwendiger ist, die Genauigkeits-Anforderungen für eine

10-wertige Darstellung zu erfüllen als für eine zweiwertige. Seit die binäre Informationsdarstellung zum Stand der Technik geworden ist, haben sich deshalb 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, hingegen bei der 10-wertigen Darstellung 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,
- Speicherschaltkreise mit extremer Speicherdichte (2 Bits je Speicherzelle in verschiedenen Flash-ROMs und DRAMs (z. B. 4-GBit-Typen).

Abbildung 1.1 Vergleich verschiedener digitaler Informationsdarstellungen

Binäre und mehrwertige Informationsdarstellung:

- eine n-wertige Informationsdarstellung entspricht ceil (ld n) Bits,
- um n Bits mehrwertig darzustellen, sind 2^n Werte erforderlich (2 Bits erfordern eine 4-wertige Darstellung, 3 Bits eine 8-wertige usw.).

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).

Digitalisierung

Zunächst ist mit allgemeinen Fehlermechanismen (Rauschen, Spannungsschwankungen, Temperaturgang, Fertigungstoleranzen) zu rechnen, wenn Signale abgetastet und gewandelt werden (und zwar sowohl bei der Analog-Digital- als auch bei der

Digital-Analog-Wandlung). Ein analoges Signal kann, genau genommen, unendlich viele Zwischenwerte annehmen, wir haben aber nur endlich viele digitale Werte (2^n bei n Bits) zur Verfügung. Das heißt, wir können den Signalwert nur in den jeweils nächstliegenden codierten Wert wandeln (Abbildung 1.2). Das ist eine Art Rundungsvorgang, der immer dann problematisch wird, wenn der zu rundende Wert genau in der Mitte liegt. Nehmen wir an, wir könnten Signalamplituden in Abständen von 1 V binär codieren. Dann werden abgetastete 5,2 V zu 5 V; 7,8 V zu 8 V usw. In beiden Beispielen haben wir uns einen Fehler von 0,2 V eingehandelt. Was aber tun mit 6,5 V? - Ob wir auf 7 V oder auf 6 V gehen, bleibt sich gleich; der Fehler beträgt jedesmal 0,5 V.

Das heißt: bei jeder Digitalisierung müssen wir mit einem *systematischen Quantisierungsfehler* rechnen. Er kann grundsätzlich nicht geringer sein als ± der halbe Wert, der in der niedrigsten Stelle codiert ist (übliche Redeweise: ± ½ Digit). Bei kleinen Signalwerten kann der Quantisierungsfehler recht beträchtlich sein.

Abbildung 1.2 Digitalisierung (das Beispiel zeigt eine Quantisierung in 16 Stufen)

Signalabtastung

Das Ziel besteht in einer möglichst getreuen und exakten Wiedergabe des tatsächlichen Signalverlaufs. Dem stehen folgende Tatsachen entgegen:

- die Analog-Digital-Wandlung ist (siehe oben) grundsätzlich mit einem Quantisierungsfehler behaftet,
- abgetastet wird nur zu festen ("diskreten" Zeitpunkten); den Signalverlauf zwischen den Abtastzeitpunkten können wir nicht beobachten,

Abtastfrequenz und Wiedergabetreue

Abbildung 1.3 veranschaulicht die einleuchtende Tatsache, daß, wenn wir einen Signalverlauf nicht oft genug abtasten, Einzelheiten verlorengehen. Die Dichte der Abtastungen wird durch die Abtastfrequenz bzw. Abtastrate (Sampling Rate) f_s gemessen (als Impulsfolgefrequenz der Abtastimpulse).

Abbildung 1.3 Signalwiedergabe durch Abtastung. a) bei hoher, b) bei niedriger Abtastfrequenz. Oben jeweils der tatsächliche Signalverlauf, darunter die Darstellung aus abgetasteten Punkten

Das Abtasttheorem (Nyquist-Kriterium) beschreibt den Zusammenhang zwischen der höchsten Frequenz (Grenzfrequenz f_g) eines bandbreitenbeschränkten periodischen Signals und der Abtastfrequenz (Abtastrate) f_s, die mindestens nötig ist, um den Signalverlauf exakt rekonstruieren zu können:

$$f_s \geq 2\, f_g.$$

Das heißt, es ist wenigstens mit dem Doppelten der oberen Grenzfrequenz abzutasten. Dieser Zusammenhang ist weithin bekannt, wird aber nicht immer zutreffend interpretiert. Abbildung 1.4 soll die Zusammenhänge veranschaulichen.

Abbildung 1.4 Zum Abtasttheorem

Zusammenhänge:

Folgefrequenz des periodischen Signalverlaufs: $f_0 = \frac{1}{t_p}$ = Grundwelle (1. Harmonische)

Der Gesamt-Signalverlauf ergibt sich als unendliche Summe von Sinusschwingungen (Fourier-Reihe):

$$u(t) = a_1 \cdot \sin \omega t + a_2 \cdot \sin 2\omega t + a_3 \cdot \sin 3\omega t + \ldots = \sum_{i=1}^{\infty} a_i \cdot \sin i\omega t \qquad \text{mit } \omega = 2\pi f_0 = \frac{2\pi}{t_p}$$

Erklärung zu Abbildung 1.4:

a) hier ist der Signalverlauf von Abbildung 1.3 als periodische Folge dargestellt. Aus der Periodendauer t_p ergibt sich eine Grund- bzw. Folgefrequenz $f_0 = 1/t_p$. Jeder periodische Ablauf kann grundsätzlich durch eine Überlagerung von Sinusschwingungen nachgebildet werden, deren Frequenz ein ganzzahliges Vielfaches der Grundfrequenz f_0 ist (also f_0, 2 f_0, 3 f_0 usw. (Fourier-Zerlegung)). In mathematisch exakter Form hat eine solche Fourier-Reihe unendlich viele Glieder.

b) da es keine unendlich hohen Frequenzen gibt, muß man die Reihe nach einer gewissen Anzahl von Gliedern abbrechen. Der Signalverlauf wird also durch eine endliche Anzahl von Sinusschwingungen näherungsweise wiedergegeben. Hier ist am Beispiel der Rechteckschwingung gezeigt, daß bereits wenige Sinusschwingungen gute Näherungen ergeben. (Hinweis: Die Fourierreihe einer symmetrischen Rechteckschwingung enthält nur die ungeraden Harmonischen.) Allein die Hinzunahme der 3. Harmonischen gibt bereits brauchbare Impulse, die man durchaus einer üblichen Logikschaltung anbieten könnte. Gehen wir bis zur 11. Harmonischen, so entstehen wirklich annehmbare Impulse mit steilen Flanken und nur noch geringfügig welligen Dächern.

c) eine Sinusschwingung kann man aus 2 Abtastwerten, die im Abstand einer halben

Periodendauer vorliegen, genau rekonstruieren - man muß aber wissen, daß die Abtastwerte eine Sinusschwingung der Frequenz f_g repräsentieren. Das heißt: aus Werten, die wir mit einer Frequenz $f_s = 2\ f_g$ abgetastet haben, können wir hinsichtlich der Frequenz f_g höchstens eine Sinusschwingung rekonstruieren, nicht aber einen beliebigen Signalverlauf.

d) dies ist das "technische Umfeld", in dem das Abtasttheorem gilt: wir begnügen uns damit, den Signalverlauf durch eine endliche Zahl von Sinusschwingungen (Harmonischen) näherungsweise nachzubilden. Die Frequenz der letzten Harmonischen, die wir "mitnehmen wollen", wird zur Grenzfrequenz f_g. Demzufolge müssen wir mit einer Abtastfrequenz $f_s \geq 2\ f_g$ abtasten. Höhere Signalfrequenzen als f_g dürfen wir nicht verarbeiten. Deshalb muß das Signal "bandbreitenbeschränkt" werden. Dazu schicken wir es durch einen Tiefpaß mit der Grenzfrequenz f_g, der dem Abtast- und Halteglied vorgeschaltet ist. Die Abtastwerte können wir dann auf beliebige Weise speichern, verarbeiten usw. Um das Signal wieder zu rekonstruieren, führen wir die Abtastwerte im Rhythmus der Abtastfrequenz f_s einem Digital-Analog-Wandler zu. Dessen Ausgang ist wiederum - im Sinne der Bandbreitenbeschränkung - ein Tiefpaß mit der Grenzfrequenz f_g nachgeschaltet. An dessen Ausgang wird dann der ursprüngliche Signalverlauf wieder erscheinen, allerdings in der Form, die sich am Ausgang des eingangsseitigen Tiefpasses ergeben hatte. Nehmen wir an, wir wollten eine symmetrische Rechteckwelle mit $f_0 = 1$ MHz abtasten und dabei noch die 11. Harmonische berücksichtigen. Das heißt, $f_g = 11$ MHz. Unser Tiefpaß müßte also nach 11 MHz "dichtmachen", und wir müßten mit $f_s = 2\ f_g$, also 22 MHz, abtasten. Am Ausgang würde dann ein Signalverlauf erscheinen, wie er im Abbildungsteil b) rechts dargestellt ist.

Aliasing

Wenn wir eine Sinusschwingung der Frequenz f_g mit einer Abtastfrequenz $f_s < 2\ f_g$ abtasten (oder - in anderer Sichtweise - die Bandbreite des abzutastenden Signals nicht auf $f_g \leq 0{,}5\ f_s$ beschränken), so erscheint am Ausgang der Anordnung von Abbildung 1.4d eine Sinusschwingung mit der sogenannten Alias-Frequenz $f_s - f_g$. - Und die schafft auch kein ausgangsseitiger Tiefpaß aus der Welt!

Oversampling

Dies ist der gängige Begriff, um hervorzuheben, daß ein bandbreitenbeschränktes Signal (obere Grenzfrequenz f_g) mit einer Abtastrate abgetastet wird, die über der unbedingt notwendigen liegt: $f_s > 2\ f_g$. Die Oversampling-Rate ist dann das Verhältnis von tatsächlicher zu notwendiger Abtastrate, also $f_s : 2\ f_g$. Beispiel: Ein Signal mit einer Grenzfrequenz $f_g = 100$ MHz wird mit einer Abtastrate $f_s = 500$ MSamples/s abgetastet. Dann spricht man von einem 2,5-fachen Oversampling ($500 : 2 \cdot 100$).

Zusammenfassung:

1. das Abtasttheorem besagt lediglich, daß aus Abtastwerten, die mit einer Abtastrate f_s gewonnen wurden, sich eine *Sinus*schwingung der Frequenz $f_s/2$ rekonstruieren läßt - und nicht etwa ein beliebiger periodischer Signalverlauf mit der Folgefrequenz $f_s/2$.
2. das Abtasttheorem gilt nur dann exakt, wenn das abzutastende Signal einer exakten Bandbreitenbeschränkung unterworfen wird, wenn es also vor der Abtastung über einen *idealen* Tiefpaß der Grenzfrequenz $f_g = f_s/2$ geführt wurde. Ein idealer Tiefpaß verfälscht aber alle Signale, die nicht sinusförmig sind (Abbildung 1.5), ein nicht idealer Tiefpaß hat hingegen unweigerlich Alias-Frequenzen zur Folge.
3. ein beliebiger periodischer Signalverlauf kann exakt aus einer unendlichen, näherungsweise aus einer endlichen Summe von Sinusschwingungen rekonstruiert werden. Diese *Fourier-Zerlegung* ist aber eine mathematische Idealisierung, die nur unter der Bedingung gilt, daß die einzelnen Sinusschwingungen nicht zueinander phasenverschoben sind (vgl. Abbildung 1.4b): wenn die Grundschwingung durch Null geht, haben auch alle Harmonischen einen Nulldurchgang). - Die Sinusschwingungen in Abbildung 1.4b) stehen auf dem Papier und lassen sich leicht zur resultierenden näherungsweisen Rechteckwelle addieren; jedes wirkliche Signal braucht aber *Zeit*, um wirkliche Signalwege (Leitungen und Schaltungen) zu durchlaufen. Eine Zeitverzögerung ist aber nichts anderes als eine Phasenverschiebung zwischen dem ein- und dem ausgangsseitigen Signalverlauf (Abbildung 1.6)

Abbildung 1.5 Frequenzgang und Verhalten im Zeitbereich

Erklärung:

a) ein idealer Impuls am Eingang eines idealen Tiefpasses führt zu Überschwingern am Ausgang. Die mathematische Behandlung ergibt sogar ausgangsseitige Schwingungen *vor* der Impulsflanke - eine physikalische Unmöglichkeit, die sich daraus erklärt, daß hier 2 Idealisierungen zusammenfallen. (Es ist ersichtlich, daß auch der ideale Tiefpaß auf eine ideale Flanke (Anstiegszeit 0) mit einer Flanke antwortet, die eine endliche Anstiegszeit (Eigenanstiegszeit t_{re}) hat.)

b) ein realer Frequenzgang mit weitgehender Annäherung an das ideale Tiefpaßverhalten. Auch diese Auslegung führt zu Überschwingern.

c) Frequenzgang mit flacherem Abfall. Je nachdem, wie die Kurve im einzelnen aussieht, gibt es entweder nur ein geringes Überschwingen oder gar keines. (Bei zu flachem Abfall wird aber die Eigenanstiegszeit zu groß.)

Damit das Abtasttheorem exakt gilt, ist (siehe oben) ein idealer Tiefpaß erforderlich. Verwirklicht man diese Forderung näherungsweise (um Alias-Schwingungen auszuschließen), so verfälscht der Tiefpaß den Signalverlauf (indem er Überschwinger hinzufügt; Abbildung 1.5a, b).

Abbildung 1.6 Signalverzerrung durch Phasenverschiebung (Beispiel)

Erklärung:

a) näherungsweise Darstellung einer Rechteckschwingung durch Überlagerung zweier Sinusschwingungen. Keine Phasenverschiebung = korrekte Darstellung (vgl. auch Abb. 1.4b).

b) die Sinusschwingungen sind gegeneinander phasenverschoben. Das führt zu einer stark verzerrten Darstellung.

Hinweis:
Damit solche Verzerrungen nicht eintreten, ist zu gewährleisten, daß Sinusschwingungen aller Frequenzen beim Passieren der Meßschaltung die gleiche zeitliche Verzögerung Δt erfahren. Phasenwinkel φ, Frequenz f und zeitliche Verzögerung Δt hängen aber folgendermaßen zusammen: $\varphi = 2\pi \cdot \Delta t \cdot f$. Mit $\Delta t = \text{const}$ ergibt sich somit die Abhängigkeit $\varphi = \text{const} \cdot f$. Das bedeutet, die Phasenverschiebung muß linear mit der Frequenz zunehmen.

Worin liegen die Vorteile der Digitalisierung?
Digitale Signale können, einmal gewandelt, infolge des extremen Störabstandes der digitalen Verarbeitung (vgl. Abbildung 1.1)[*)], praktisch immer wieder 100%-ig identisch (also fehlerfrei) *reproduziert* werden. (Im Innern digitaler Systeme kann man systematische Fehler beim Transportieren und Speichern praktisch ausschließen.)

Hinzu kommen die allgemeinen Vorzüge der digitalen Schaltungstechnik:

- digitale Information kann man in praktisch unbegrenztem Umfang unbegrenzt lange speichern[**)],
- digitale Information kann man an sich beliebigen Wandlungen unterziehen (z. B. Rechen- und Durchmusterungsalgorithmen),
- 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 akzeptablen Aufwand durchführbar,
- digitale Systeme lassen sich leichter in hoch integrierten Schaltungen verwirklichen und so kostengünstig fertigen.

*): der Begriff erschließt sich intuitiv aus den jeweils zulässigen Wertebereichen und den Abständen zwischen ihnen. Genaueres in Heft 19.

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

2. Realisierung von Aufgabenstellungen der Informationstechnik

2.1. Überblick

Im folgenden geht es darum, wie sich Aufgabenstellungen der Informationselektronik - vorzugsweise der Digitaltechnik - mit Einrichtungen lösen lassen, die auf modernen Halbleiter-Schaltkreisen und dem Stand der Technik entsprechenden Schaltungsprinzipien beruhen.

Der Stoff läßt sich systematisch - und dabei ganz allgemeingültig - gliedern, wenn man dem Signalfluß von der Signalquelle bis zur letztlich ausgelösten Wirkung folgt (Abbildung 2.1).

Abbildung 2.1 Der Signalfluß von der Quelle zur Wirkung (nach: Texas Instruments)

In der Abbildung sind alle Funktionskomplexe dargestellt, die - entsprechend dem Stand der Technik - am Signalfluß beteiligt sein können, von der Signalaufbereitung über eine erste Signalwandlung (analog → digital), die digitale Verarbeitung und Signalübertragung, eine zweite Signalwandlung (digital → analog) bis zum Erbringen der Wirkung in den Leistungsstufen*). In der Praxis sind die Signalflüsse oft abgekürzt (es sind nicht alle genannten Stufen vorhanden) und funktionell vereinfacht (so beschränkt sich die Analog-Digital-Wandlung im Lesesignalweg eines Plattenlaufwerks auf eine Impulsformung (Begrenzung) des verstärkten Lesesignals).

*): hinzu kommen die - unumgänglichen - Vorkehrungen der Stromversorgung.

Abbildung 2.2 veranschaulicht den grundsätzlichen Aufbau einer *digitalen* informationsverarbeitenden Einrichtung.

Abbildung 2.2 Digitale informationsverarbeitende Einrichtung (grundsätzlicher Aufbau)

Einrichtungen ohne jegliche Verbindung zur Um- bzw. Außenwelt sind anwendungsseitig sinnlos. Die Um- bzw. Außenwelt kann sowohl durch Meßfühler, Geber, Stellglieder, Signalwandler (Abbildung 2.1) 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[1)]. Ergebnisse der Informationsverarbeitungsvorgänge werden über Ausgabeschaltungen in der Umwelt zur Wirkung bzw. zur wahrnehmbaren Darstellung gebracht (sichtbare Anzeige, akustische Signale, Sprachausgabe usw.)[1)]. Die Verarbeitungsvorgänge selbst laufen über Verknüpfungsschaltungen ab. Verknüpfungen zu verarbeitender Daten werden auch als Operationen bezeichnet[2), 3)]. Durch eine reine Informationswandlung ausschließlich über Verknüpfungsschaltungen werden sich nur sehr wenige anwendungspraktisch wichtige Aufgaben unmittelbar lösen lassen[4)]. Speichermittel[3)] sowie Steuerschaltungen werden somit in den meisten Fällen unbedingt erforderlich sein. Die Steuerschaltungen wirken über Ausgänge (Steuersignale bzw. -leitungen) auf alle anderen Schaltmittel. 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 2.2 auch Bedingungssignale bzw. -leitungen dargestellt.

Anmerkungen:

1) erforderlichenfalls können Schaltmittel der Signalwandlung und Leistungsstufen vor- bzw. nachgeschaltet sein (Abbildung 2.1),
2) demgemäß ist für die entsprechenden Schaltmittel auch die Bezeichnung "Operationswerk" (manchmal kurz "Operator") in Gebrauch,
3) man sollte sich hierbei nicht nur *ein* Operationswerk, *einen* Speicher usw. vorstellen (in Entsprechung zum allgemein bekannten Aufbau eines einfachen Universalrechners (Prozessors, Mikrocontrollers), sondern durchaus auch Anordnungen, die mehrere Operationswerke, Speicher usw. enthalten,
4) ob es überhaupt gelingen kann, eine Informationswandlung mit Verknüpfungsschaltungen technisch zu verwirklichen, ist ein naheliegendes Kriterium bei der Entscheidung, ob man die Aufgabe mit Hardware oder mit Software löst.

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

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.

2.3.2. Standardisierung durch den Prozessor

Das ist besonders im unteren Preis- und Leistungsbereich üblich. Es gibt fertige Moduln mit weithin verbreiteten Mikrocontrollern bzw. -prozessoren (wie z. B. 8051, Z 80[1)] usw.), die man noch durch eigene Anpassungs- und Zusatzhardware beschalten (in die Anwendung einbetten) muß. Auch ist es nicht besonders schwierig, Mikrocontroller und "kleine" Prozessoren[2), 3)] unmittelbar in eigene Entwürfe einzubauen.

Anmerkungen:

1) ausgerechnet "Uralt-Typen" werden noch gern verwendet (sie werden sogar in riesigen Stückzahlen gefertigt - und gehen in Einrichtungen, denen man es von außen gar nicht ansieht (Bügeleisen, Toaster, Faxgeräte usw.)),
2) namentlich bei neueren Typen (seit den 80er Jahren) man hat sich viel Mühe gegeben, die Interfaces auszulegen, daß sich die Schaltkreise auf einfache Weise mit Speichern, E-A-Einrichtungen usw. zusammenschalten lassen (meistens genügt ein Einsatz "nach Kochbuch"),
3) die Schwierigkeiten wachsen mit zunehmender Taktfrequenz (Tabelle 2.1).

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, stattdessen 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 2.1 Realisierungsprobleme in Abhängigkeit von der Taktfrequenz

2.3.3. 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. Im mittleren und oberen Leistungsbereich dürfte derzeit der VME-Bus vorherrschend sein, von dem es verschiedene Ausführungsformen gibt. Was mehr und mehr Marktanteile gewinnt: die Nutzung von Bussystemen der PC-Technik (ISA und PCI*); künftig auch USB und IEEE 1394).

*): Ausführungsbeispiele: AT-96, PC-104, PC-104 plus, CompactPCI, IndustrialPCI. Auch der ISA-Bus stirbt in diesen Anwendungsbereichen noch lange nicht aus!

2.3.4. 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 System-Architektur im Vordergrund. Die Verbreitung der IBM-kompatiblen Personalcomputer 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 verbreiteten Bussysteme usw.

2.3. Selbstentwickelte Computersysteme

Derzeit wird wohl kaum noch 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 Schaltkreis ausgehend in genauer Anpassung an den jeweiligen Einsatzfall entwickelt.)

*): wo es aber Sinn hat - und eher noch an Bedeutung zunimmt: Prozessoren für spezielle Verarbeitungsaufgaben (Signalverarbeitung, Kryptographie, Spracherkennung, Graphik usw.) - bis hin zu Prozessoren mit dynamisch änderbaren Funktionseigenschaften (auf FPGA-Grundlage).

2.4. Klassische Schaltungsentwicklung

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

2.5.2. 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 Hardware-Baugruppen bzw. -Systemen, die in großen Stückzahlen gefertigt werden. Entsprechende Schaltkreise können gleichsam sofort aus dem Regal (Off the Shelf) 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 (z. B. der 7407 - ein Open-Collector-Treiber, der u. a. in der Tastaturschnittstelle von PCs eingesetzt wird).)

Moderne Entwicklungen betreffen:

- verringerte Versorgungsspannungen (von 3,3 V (als modernem “Industriestandard”) bis herab zu 1,8 V),
- die Ausrichtung auf 2 Typensortimente: (1) Bustreiber und andere Interface-Koppelstufen, (2) Grundfunktionen zum Aufbauen von sog. “Restlogik” (Glue Logic)[*)],
- miniaturisierte Gehäuse (Anschlußabstände 1,25, 0,625, 0,5 und 0,4 mm),
- was nach und nach aussterben wird: (1) herkömmliche Baureihen (Standard-TTL, LS-TTL usw.), (2) Funktionen mittleren Integrationsgrades (MSI), wie Zähler, Addierer usw.

*): das betrifft u. a. einzelne Gatter in extrem kleinen Gehäusen (Micro bzw. Pico Gates). Der Zweck: obwohl “fast alles” in hochintegrierten Schaltkreisen enthalten ist; braucht man gelegentlich doch hier eine einzelne Negation, dort eine NAND-Verknüpfung o. dergl. Miniaturisierte Einzelgatter können bei geringstem Platzbedarf jeweils an Ort und Stelle auf der Leiterplatte angeordnet werden (es müssen keine Leiterzüge zu einer abgesetzten “Restlogik” geführt werden).

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 Gatterfunktionen sind typischerweise einzeln verfügbar.
- MSI (Medium Scale Integration): ein Schaltkreis, der eine "mittlere" Anzahl an Gatterfunktionen enthält (nach Texas Instruments: zwischen 12 und 100 Gatterfunktionen). Der Schaltkreis stellt typischerweise eine kompliziertere Funktion dar (Zähler, Decoder, Multiplexer usw.); die Gatterfunktionen sind nicht einzeln nutzbar.
- LSI (Large Scale Integration): damit bezeichnen wir alle noch komplizierteren Schaltkreise (von 100 Gatterfunktionen an aufwärts).

Hinweise:

1. Bei Neuentwicklungen unbedingt nachsehen, was noch lieferbar ist (auch an den späteren Service beim Kunden denken! - Stichwort: Lebensdauer). Quellen: die Typenlisten (Selection Guides) der Schaltkreishersteller sowie deren WWW-Seiten. Wenn es wirklich um Stückzahlen geht: schriftlich nachfragen, ggf. Lieferbereitschaft bestätigen lassen!
2. Exotische Typen nicht verwenden (betrifft vor allem LS-TTL-MSI; erkennbar an folgenden: (1) nur in wenigen bzw. einer einzigen (und zudem älteren) Baureihe angeboten, (2) deutlich höhere Katalogpreise ("Seltenheitswert") als andere Typen vergleichbaren Integrationsgrades).
3. Ausweg: MSI-Funktionen mit programmierbaren Schaltkreisen nachbauen (dabei vom Datenmaterial anregen lassen - alte Datenbücher nicht wegwerfen!). Für entsprechende Ersatzbestückungen ist ein umfangreiches (wenngleich nicht gerade billiges) Sortimen an Adaptern, Zwischenleiterplatten usw. am Markt.

2.5.3. 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 bestimmte 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 (Abbildung 2.3,

Tabelle 2.2). 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 2.3).

Abbildung 2.3 Programmierbare Schaltkreise: eine Übersicht (Texas Instruments)

Begriffsbildungen

Die Begriffe werden teils herstellerspezifisch verwendet, teils hat sich ein allgemeiner Gebrauch durchgesetzt (obwohl es sich manchmal im eigentlichen Sinne um Warenzeichen handelt).

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. Gilt heutzutage als 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, komplexere Netzwerke und Makrozellen.)*)
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 2.2 Programmierbare Schaltkreise

Programmierverfahren	Ausführung	Änderbarkeit	Bemerkungen
Maskenprogrammierung	beim Halbleiterher-steller	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 vorge-fertigt, die nicht benötigten wer-den 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
Ladungsspeicherung mit UV-Löschung	beim Anwender*)	durch Löschen und Neuprogrammieren	Löschen durch UV-Licht; erfordert Quarzglasfenster im Schaltkreis (es gibt auch preisgün-stige Ausführungen ohne Fenster; diese kann man nicht mehr löschen)**)
Ladungsspeicherung mit elektrischer Löschung (EEPROM, Flash)	beim Anwender*)	durch Löschen und Neuprogrammieren (auch: in der Anwendungs-schaltung (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 2.3 Programmierverfahren (Übersicht)

Hinweise:

1. Der wichtigste Vorteil programmierbarer Schaltkreise: der Integrationsgrad ist durchaus mit jenem “richtiger” anwendungsspezifischer Schaltkreise (ASICs) vergleichbar, es sind aber keine anwendungsspezifischen Halbleiter-Fertigungsschritte erforderlich (Verfügbarkeit "aus dem Regal").
2. Die Besonderheit im Vergleich zu herkömmlichen “Off-the-Shelf”-Schaltkreisen: man braucht eine Entwicklungsumgebung (Software, ggf. Programmierhardware) - eine Frage der Kosten (und auch der Gewöhnung bzw. Einarbeitung (wichtig bei “engen” Terminvorgaben)).
3. Es gibt auch programmierbare *Analog*schaltungen.
4. Zu Einzelheiten siehe Heft 05.

2.5.4. 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 (Tabelle 2.4).

Gate Array
Der Schaltkreis (Abbildung 2.4) hat eine Grundstruktur aus einzelnen Gattern, die in Form einer Matrix gleichförmig angeordnet sind (ein "Gatter" - Gate - ist eine Schaltung, die eine elementare logische Verknüpfung realisiert). Diese Matrix ist am Schaltkreisrand von ebenfalls gleichförmig angeordneten Ein- und Ausgangsschaltungen umgeben. Zwischen den Gatter-Reihen sind Verdrahtungskanäle freigelassen, in denen die Gatter auf anwendungsspezifische Weise untereinander verbunden werden.

Abbildung 2.4 Gate-Array-Schaltkreis (Oki)

Sea of Gates
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 gleichförmig fest angeordneten Ein- und Ausgangsschaltungen umgeben sind (Abbildung 2.5). Besondere Verdrahtungskanäle sind nicht vorgesehen. Die Transistoren werden nach Bedarf zu Gattern und anderen Funktionseinheiten verschaltet. Verbindungsleitungen werden über Transistoren hinweggeführt (die so überdeckten Transistoren können nicht funktionell ausgenutzt werden).

Abbildung 2.5 Sea-of-Gates-Schaltkreis (Oki)

Standardzellenschaltkreise (Standard Cells)
Diese Schaltkreise sind nicht von vornherein mit irgendwelchen Schaltelementen belegt. Vielmehr kann ein solcher Schaltkreis (Abbildung 2.6) 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 Mikroprozessoren nutzbar. Bei Mikroprozessoren kann man auf populäre Typen, wie 8051, 80286 oder Motorola 68 000 zurückgreifen. Viele Zellenbibliotheken enthalten Standardschaltungen, die den verbreiteten TTL- und CMOS-Baureihen entsprechen. Auch die Motherboard-Schaltkreise der PCs sind vielfach aus Standardzellen aufgebaut (wie DMA-Schaltungen, Zeitgeberschaltungen, Interrupt- Controller, serielle und parallele Schnittstellen usw.).

Abbildung 2.6 Standardzellenschaltkreis (Oki)

Full Custom

Bei solchen Schaltkreisen ist nichts vorgefertigt oder vordefiniert, vielmehr wird der gesamte Schaltkreis von Grund auf für die jeweils gewünschte Funktion entwickelt (Abbildung 2.7).

Abbildung 2.7 Voll kundenspezifischer Schaltkreis (Full Custom; Oki)

	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 1 1/2 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 2.4 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. (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). Schnelles Reagieren auf dem Markt (Änderungen, Modernisierungen, Ausbügeln von Fehlern, Anpassungen an neue Standards). Das betrifft vor allem die PC-Technik (Graphikkarten, Modems usw.).)

2.5. Grundsatzentscheidungen beim Entwerfen

Entwerfen heißt, eine bestimmte Aufgabe mit zuhandenen technischen Mitteln auf wirtschaftlich und anwendungspraktisch sinnvolle Weise zu lösen.

2.6.2. Analyse der Entwurfsaufgabe

Es gilt, sich über folgende Problempunkte Klarheit zu verschaffen:

1. über die Funktionalität: was ist eigentlich zu tun? (Mit anderen Worten: welche Informationswandlungen sind auszuführen?)
2. über das Realzeitraster: in welcher Zeit sind die Informationswandlungen auszuführen?
3. über die Voraussetzungen und Nebenbedingungen: welche Vorschriften, Termine und Kostenvorgaben sind einzuhalten, auf welche Technologien wollen (bzw. müssen) wir zurückgreifen?

Wir müssen die Anforderungen und Lösungsmöglichkeiten wenigstens in grober Näherung sozusagen vorsortieren und bewerten (Rückgriff auf Erfahrungen, Analogieschlüsse (zu bereits bekannten und bewährten Lösungen), Probeentwürfe, Simulation).

Die Funktionalität
Sich darüber klar zu werden, *was* man eigentlich will, muß grundsätzlich jedem zielgerichteten technischen Handeln vorgeordnet werden. Das heißt: eine Entwicklungsaufgabe nie ohne genau ausgearbeitete (und kritisch geprüfte!) Spezifikation beginnen. Spezifikationsfehler sind vor Entwicklungsbeginn vergleichsweise einfach, später aber nur mit sehr viel Mühe zu beheben.

Hinweise:

1. Es ist entscheidend, sich eine klare, deutliche Vorstellung vom zu lösenden Problem zu erarbeiten. Hierzu sind - ohne Rücksicht auf irgendwelche akademischen Lehren - die Darstellungsmittel so wählen, wie dies jeweils zweckmäßig ist (die Auswahl ist teils subjektiv bedingt, teils durch die Natur des Problems). So wird man das eine Vorhaben sofort durch einen Schaltplan beschreiben, das andere zunächst durch Boolesche Gleichungen, ein weiteres durch Zustandsgraphen und wiederum ein anderes durch ein Programm in einer höheren Programmiersprache.

2. Grundsätzlich ist gegen eine irrtümliche Problemauffassung kein Kraut gewachsen. Die *formale* Korrektheitsprüfung oder die Simulation nützt *hier* nicht viel: sie wird typischerweise nicht mehr leisten, als zu bestätigen, daß die fehlerhafte Spezifikation korrekt (d. h. einschließlich der Spezifikationsfehler!) in eine Schaltung oder in ein Programm umgesetzt wurde.
3. *Tip:* keinesfalls auf eine ordentliche *umgangssprachliche Beschreibung* verzichten ("ordentlich" = verständliche Begriffe, logische Gliederung, klare und knappe Darstellung). Auch wenn es naheliegt, sofort eine Schaltung zu skizzieren oder gleich ein C-Programm in den Computer zu tippen - Sie werden sich wundern, auf wieviele Probleme Sie stoßen, wenn Sie versuchen, das, was Sie wollen, in verständlicher Alltagssprache zu formulieren. Es ist geradezu ein Alarmzeichen, wenn es an irgendeiner Stelle nicht gelingt, die Entwurfsabsicht klar auszudrücken: man hat dann die zu lösende Aufgabe nicht richtig verstanden oder irgend etwas übersehen oder man sieht (gar nicht selten) "den Wald vor lauter Bäumen nicht". Auf einer solchen Grundlage entwickelte Schaltungen, Programme usw. enthalten oft überdurchschnittlich viele Fehler. (Nicht umsonst steht gerade in hochentwickelten Qualitätssicherungssystemen die umgangssprachlich formulierte Spezifikation am Anfang des Entwicklungsprozesses.)

Das Realzeitraster

Unter diesem Begriff wollen wir alle Anforderungen an das zeitliche Verhalten zusammenfassen. Solche Anforderungen betreffen Gesamt-Ausführungszeiten, Latenzzeiten (von der auslösenden Anforderung bis zum Beginn des Reagierens), Takt-Zykluszeiten usw. Wir wollen zunächst annehmen, daß es eine “kritische” Zeitvorgabe t gibt (Tabellen 2.5, 2.6). Die Aufgabe der Systementwicklung besteht dann darin, zu gewährleisten, daß die jeweils geforderten Informationswandlungen in der vorgegebenen Zeit t mit *kostenoptimalen* Mitteln erledigt werden (Kosten über alles = über den gesamten Lebenszyklus = Total Cost of Ownership). Je geringer die Zeitvorgabe t und je größer die Kompliziertheit der auszuführenden Informationswandlungen, um so größer der Aufwand.

Typische Fragen:

- in welcher Zeit ist die Aufgabe zu erledigen?
- einmalige oder zyklisch wiederholte Ausführung?
- sind Latenzzeit-Vorgaben (von der Anforderung bis zum Beginn der Reaktion) einzuhalten?

Zeitraster	Beispiele	technische Mittel
10 ns	Befehlsausführung in Prozessoren, Speicheransteuerung, Videodarstellung	Hardware, Gatter-Ebene (auch: Transistor-Ebene)
100 ns	Gerätesteuerung	Hardware auf Gatter-Ebene, State Machines, Mikroprogrammierung
1 µs	Gerätesteuerung	Hardware auf Gatter-Ebene, State Machines, Mikroprogrammierung, Maschinenprogrammierung (was sich mit 10...50 Maschinenbefehlen erledigen läßt)
1 ms	Gerätesteuerung, Regelungsaufgaben (Closed Loop)	Realzeit-Software (was sich mit wenigen tausend Maschinenbefehlen erledigen läßt)
10 ms	Gerätesteuerung, Regelungsaufgaben (Closed Loop), Bedienung/Anzeige	Realzeit-Software
100 ms	Regelungsaufgaben (Closed Loop), Bedienung/Anzeige, Prozeßsteuerung, Informationsbeschaffung (Retrieval)	komplexe Realzeit-Software, Vernetzung
1 s	Prozeßsteuerung, Informationsbeschaffung (Retrieval)	Realzeit- bzw. beliebige Software (je nach Hardware-Plattform)
kommt nicht drauf an	Fertigungssteuerung, allgemeine Verwaltung	beliebige Software (auch Windows etc.), Vernetzung (incl. Internet)

Tabelle 2.5 Größenordnungen des Realzeitrasters

Beispiele	Größenordnung der Zeitvorgabe t	zu erbringende Funktionen[1)]
serielle Hochgeschwindigkeits-Bussysteme (Fibre Channel, FDDI, Escon, Gigabit Ethernet, IEEE 1394)	Taktfrequenzen 250 MHz... > 1 GHz; t zwischen 4 und < 1 ns	Serialisierung und Deserialisierung[2)]
serien-parallele Hochgeschwindigkeits-Bussysteme (Direct Rambus, SLIO, AGP)	Taktfrequenzen bis ca. 500 MHz; t um 5...2 ns	Serialisierung und Deserialisierung, Steuerung der Zugriffsabläufe bzw. Zustandsübergänge[2)]
Befehlsablaufsteuerung in Prozessoren	Taktfrequenzen bis > 400 MHz; t bis zu 2 ns	komplizierte Zustandsübergänge, Datenverknüpfungen, Beobachten/Erkennen von Nebenläufigkeiten und Sonderbedingungen
Videosignale	zwischen ca. 5 (Fernsehen) und 250 MHz; t bis zu 4 ns	Serialisierung und D-A-Wandlung
parallele Bussysteme (Prozessor-Bussysteme, PCI)	typische Taktfrequenzen: 33, 66, 100 MHz; t zwischen 30 und 10 ns	Steuerung der Zugriffsabläufe bzw. Zustandsübergänge
Steuern und Regeln technischer Prozesse	ergibt sich gemäß Abtasttheorem; t typischerweise im ms-Bereich[3)]	anwendungsspezifisch; typischerweise hohe algorithmische Kompliziertheit
Mensch-Maschine-Interfaces (Bedientafeln, Tastaturen, Anzeigen, Bedienoberflächen)	ergibt sich aus psychologischen Erfahrungstatsachen (“was wird als befriedigend empfunden”) und aus physiologischen Grenzen (Informationswahrnehmung, Geschwindigkeit von Bedienhandlungen); t typischerweise zwischen 20 und 200 ms[3)]	anwendungsspezifisch; typischerweise mittlere...hohe algorithmische Kompliziertheit

1): hier sind die im Zeitraster t (z. B. mit einer bestimmten Taktfrequenz) auszuführenden leistungsentscheidenden Funktionen gemeint; 2): typischerweise werden beide Taktflanken ausgenutzt; 3): “Überbieten” der Zeitanforderungen, die sich aus der Anwendung ergeben, ist sinnlos

Tabelle 2.6 Beispiele für Realzeitanforderungen. Oberhalb der Doppellinie: Hardwarelösungen unumgänglich, unterhalb: typische Fälle für Softwarelösungen

Eine schnelle Vor-Entscheidung:

- wenn t groß (Richtwert: > 1 ms): das kostengünstigste zuhandene Mittel aussuchen und prüfen, ob es ausreicht,
- wenn t extrem klein (Richtwert: < 1 µs): es kommen typischerweise nur schaltungstechnische Lösungen in Frage. Gelegentlich sind erfinderische Bemühungen notwendig (Aufgabe ist nicht immer mit Rückgriff auf Stand der Technik lösbar).

Hinweis:
Komplexität und Kompliziertheit sind keine Wechselworte*):

- *Komplexität* beschreibt Ressourcen-Anforderungen z. B. einer Schaltung oder eines Algorithmus in Abhängigkeit von der Problemgröße. Beschreibung hat die Form O (n) (Order of... = Größenordnung von...). *Komplex* = mehr als linear mit der Problemgröße wachsende Anforderungen (Schaltungsaufwand, Verarbeitungsleistung, Speicherplatz).
- *Kompliziertheit* = Spitzfindigkeit, Verwickeltheit, Vielfalt der Funktionen. *Kompliziert* = verwickelt (sophisticated). Läßt sich näherungsweise durch Umfang der Problembeschreibung (= funktionelle Spezifikation) kennzeichnen.

*): siehe auch Anhang 1. Im allgemeinen Sprachgebrauch hält man sich allerdings nicht immer an die Unterscheidung (auch wir können uns diesen Gepflogenheiten nicht immer entziehen).

Voraussetzungen und Nebenbedingungen

- Zuhandenheit,
- Infrastruktur,
- Kosten,
- Zeitbedarf (Einordnung in eine Terminplanung, Time to Market),
- sonstige Vorgaben (Normen und anderweitige Vorschriften (u. a. zu Sicherheit, Qualität, Produkthaftung, Umweltbeeinflussung usw.)),
- Akzeptanz.

Zuhandenheit
Der Begriff soll klarstellen, daß wir nur auf das zurückgreifen können, was tatsächlich zur Hand (= praktisch verfügbar) ist. Technologien, Verfahren usw., die es zwar gibt, über die wir aber nicht verfügen können (nicht ausgereift, nicht beschaffbar, zu teuer) nützen uns auch nichts.

Die Zuhandenheit betrifft:

- Technologien,
- Beschreibungs - und Entwicklungsmittel (auch: Entwicklungssysteme und -umgebungen),
- Fachwissen (State of the Art), gedankliche Ansätze (Paradigmata), Problemverständnis,
- Verifizierungsmittel*): Test, Simulation, Prüfen/Messen, Fehlersuchen (Debugging).

*): *Achtung:* was man nicht prüfen kann (Funktionsnachweis), kann man eigentlich gar nicht realisieren.

Ebenen der Zuhandenheit:

- Basistechnologien (Si, GaAs usw.),
- Fertigungstechnologien (z. B. 0,35 µm CMOS),
- Bauelemente und fertige Schaltungslösungen:
  - auf Gatter-Ebene,
  - auf Register-Transfer-Ebene,
  - als Funktionsblöcke (Building Blocks),
- Hardware-Plattformen,
- Entwicklungsumgebungen,
- Systemumgebungen.

Auf welcher Ebene der Zuhandenheit aufsetzen?
Das hängt von der Realzeitraster-Vorgabe (t) ab und ist deshalb ein fließendes Ziel in Abhängigkeit vom Stand der Technik:

- $t < 1$ ns: Basis- (Halbleiter-) Technologie,
- t = wenige ns: Fertigungstechnologie (zuhandene Halbleiter-Technologien, auch in Kombination (Mixed Signal (= analog + digital), Logik + DRAM usw.),
- $t < 1$ µs: Bauelementetechnologie (zuhandene Schaltkreise),
- $t \geq 1$ ms (vom ms-Bereich an): zuhandene Funktionseinheiten...fertige Systemumgebungen.

Kostenoptimierung: nicht zuviel Overkill für die Problemlösung, sondern gerade soviel, um Reserven für Änderungen zu haben (den gesamten Lebenszyklus bedenken!).

Paradigmata
Mit welchen geistigen und handwerklichen Voraussetzungen der Entwickler an die Aufgabe herangeht, wird oft nicht beachtet - es ist aber nicht selten von entscheidender Bedeutung. Uns geht es hier nicht um das subjektive Können, sondern um die Wahl der grundsätzlichen Lösungsansätze, d. h. sozusagen um die “Philosophie” der Problemlösung bzw. des Systementwurfs (Tabelle 2.6).

Hinweis:
Wichtig ist, jeweils einen *zweckmäßigen* Ansatz zu wählen - eine Binsenweisheit, die aber gelegentlich übersehen wird. Auch “Entwicklungsphilosophien” sind der Mode unterworfen

(erkennbar anhand von Zeitschriften, Kongressen, Seminaren, Lehrplänen usw.). Davon nicht beeindrucken lassen! - Aber auch nicht allzu konservativ/beschränkt bleiben: ein professioneller Entwickler sollte in der Lage und willens sein, sich, wenn es sein muß, auch zügig in ungewohnte Verfahren und Prinzipien einzuarbeiten.

grundlegende Lösungsansätze und Beschreibungsmittel	technische Lösungsansätze	Programmier-"Philosophien"
• Boolesche Gleichungen, • Kontaktpläne (≙ herkömmliche Steuerungstechnik), • Schaltpläne, • Zustandsgraphen (State Machines, abstrakte Automaten), • Entscheidungstabellen, • Hardware-Beschreibungssprachen, • Petri-Netze, • analoge Signalverarbeitung (≙ Regelungstechnik, rückgekoppelte Systeme, Laplace-Transformation), • digitale Signalverarbeitung (≙ z-Transformation), • neuronale Netze, • Fuzzy-Logik	• kombinatorische Zuordnung, • Table Lookup, • State Machines (abstrakte Automaten), • Sequencer (einfache Folge-, Zeitplan- und Ablauf-steuerungen), • Mikrocontroller, • universelle Prozessoren, • spezielle Prozessoren, • Multiprozessorsysteme, • hochparallele Systeme (auch: zelluläre und systolische)	• herkömmliche Programmierung (Programmablauf, Flowchart- bzw. if-then-goto-Paradigma) • strukturierte Programmierung, • ereignisgesteuerte Programmabläufe • Multitasking, • abstrakte Datentypen, • objektorientierte Programmierung , • relationale Ansätze (Datenbasis; Memory Based Reasoning), • regelbasierte Ansätze (Expertensysteme, Prolog usw.)

Tabelle 2.7 Lösungsansätze im Systementwurf (Auswahl)

2.6.3. Software oder Hardware?

Im strengen Sinne gibt es die Alternative nicht, da jede Software eine Hardwareplattform braucht, um laufen zu können. Im praktischen Sinne handelt es sich darum, zu entscheiden, ob man seine Aufgabe auf Grundlage einer fertigen Hardware-Plattform (also "nur" durch Software (Programmierung)) lösen oder ob man eigene Hardware entwickeln soll - von einfachen Anpassungs- und Zusatzschaltungen bis hin zum vollständigen System "aus einem Guß".

Zunächst sind folgende Gesichtspunkte zu berücksichtigen, die sich aus dem Stand der Technik ergeben:

- an die Funktionseigenschaften einer Neuentwicklung werden hohe Anforderungen gestellt. (Damit, daß die jeweilige Grundfunktion “gerade so” erfüllt wird, dürfte sich wohl kein Kunde mehr zufriedengeben.)
- das Programmier-Paradigma hat sich weitgehend durchgesetzt. Somit wird praktisch jedem Lösungsansatz für auch nur halbwegs komplexe funktionelle Anforderungen irgendeine Form der Programmierbarkeit zugrunde gelegt. (Kaum jemand baut noch hochkomplizierte Spezialschaltungen, die nicht irgendwie programmierbar bzw. durch Programmierung steuerbar sind.)

Es geht also in der Praxis nicht um den extremen Gegensatz: fertige Systemumgebung oder Einzweck-Spezialschaltung - sondern um kostenoptimierte Verbundlösungen.

Das Entscheidungsproblem (Hardware oder Software) stellt sich in der Praxis dann, wenn:

- extreme Realzeitanforderungen zu erfüllen sind,
- ein Funktionsumfang verwirklicht werden soll, der im jeweiligen Kostenrahmen bisher nicht implementiert werden konnte,
- Entwicklungszeiten verringert werden sollen (Time to Market).

Weshalb ersetzen wir Hardware durch Software?

- der herkömmliche Trend seit den 60er Jahren,
- Rückgriff auf ein höheres Niveau der Zuhandenheit (“weniger selbst tun”),
- Zuhandenheit hochleistungsfähiger Technologien (extreme Integrationsgrade, Speicherkapazitäten, Taktfrequenzen),
- Verkürzung der Entwicklungszeit (“es ist nur zu programmieren” = Time to Market),
- Verringerung der Hardwarekosten (durch Einsatz von Hardware aus der Massenfertigung),
- Beherrschung von Kompliziertheit (durch Zerlegen in handliche Programmstücke),
- Ersparen der Hardware-Entwicklung (stattdessen Rückgriff auf zuhandene funktionstüchtige Plattformen),
- kurze Änderungszyklen,
- flexible Entwicklungswerkzeuge,

- funktionelle Flexibilität allgemein (wir *dürfen* programmieren - es läßt sich "alles" programmieren),
- Software kann nicht ausfallen (verschleißen)*).

*): gleichwohl kann Software Fehler enthalten (und auch veralten - wenn es weit und breit keine Hardware-Plattform mehr gibt, auf der sie laufen könnte).

Weshalb ersetzen wir Software durch Hardware?

- ein (wieder mal) neuerer Trend - angeregt durch Verfügbarkeit hochintegrierter programmierbarer Schaltkreise (FPGAs, CPLDs),
- mehr Leistung,
- Kontrolle über den gesamten Lebenszyklus (nicht mehr auf Zulieferungen, fremde Standards und fremde Schutzrechte angewiesen sein),
- alle Funktionsfehler sind *unsere* Fehler - das mag gelegentlich das Selbstbewußtsein beeinträchtigen, hat aber für sich, daß wir sie auch beseitigen *dürfen*,
- Kostenoptimierung = Verbilligung der Hardware-Plattform, kein Software-Overhead (besserer Wirkungsgrad),
- Lösung der Aufgabe mit weniger Silizium, Strom und Störabstrahlung (langsamst-mögliche Taktierung),
- besseres Realzeitverhalten,
- kürzere Latenzzeiten,
- echter Parallelismus (der dem Problem überhaupt inhärente Parallelismus kann voll ausgenutzt werden, Schaltungsstruktur ≙ Problemstruktur ≙ Datenfluß, höchster Wirkungsgrad),
- unmittelbare (evidente) Vergegenständlichung der Informationswandlungen (wir *müssen nicht* programmieren: Programme sind gelegentlich unangemessen und unübersichtlich ("semantische Lücke" zwischen Problembeschreibung und Programmablauf, zuviel Overhead)).

2.6. Lösungsansätze im Vergleich

Die dem Stand der Technik entsprechenden Ansätze, mit denen Aufgaben der Digitaltechnik gelöst werden können (vgl. die Abschnitte 2.2. bis 2.4.), sind hinsichtlich wichtiger Vor- und Nachteile in Tabelle 2.9 zusammengestellt. Tabelle 2.8 gibt einen Überblick über typische Entwurfsaufgaben und die jeweils in Frage kommenden Lösungsansätze.

Aufgaben	Lösungsansätze
problemspezifische Anpassungen[1)]	• Off-the-Shelf-Schaltkreise, • programmierbare Schaltkreise, • einfache programmierbare Plattformen (z. B. Mikrocontroller)[2)]
Sondervorhaben (Einzelstücke oder geringe Stückzahlen)	• Off-the-Shelf-Schaltkreise, • programmierbare Schaltkreise, • bei höherer Kompliziertheit: fertige Hardware-Plattformen (Mikrocontroller, Industrie-PCs oder PC-Moduln, VME-Bus-Systeme usw.)[2)]
Produktinnovation und "High Volume Products"	• ASICs, • programmierbare Schaltkreise, • selbstentwickelte programmierbare Systeme (Embedded Systems)[2)], • fertige Hardware-Plattformen[2)]

1): vor allem am standardisierte Schnittstellen; 2): bedeutet: Problem wird vorzugsweise mit Software gelöst

Tabelle 2.8 Typische Entwurfsaufgaben

Wichtige Einflußgrößen:

- Kosten,
- Termine,
- Lieferzeiten,
- Paßfähigkeit (Kompatibilität, Standards usw.),
- Erfordernisse der Fertigung,
- Erfordernisse des Service,
- Abmessungen, Strombedarf, Kühlung usw.,
- verfügbare Ausrüstung,
- Budget für Investitionen,
- Placierung am Markt,
- anzustrebende Alleinstellungsmerkmale,
- Patent- und Urheberrecht, Lizenzfragen,
- Zukunftssicherheit der eigenen Lösung,
- Zukunftssicherheit der Zulieferungen.

Ansatz	Vorteile	Nachteile	sinnvolle Nutzung
fertige Hardware-Plattformen	• kaum eigene Hardware- Entwicklung, • durch Software beliebig hohe funktionelle Komplexität (es ist "alles" programmierbar), • Änderungen unproblematisch, • sofortige Verfügbarkeit der Hardware	• vergleichsweise hohe Kosten; • Leistungs-Probleme bei zeitkritischen Funktionen	• "gewöhnliche" Datenverarbeitung, • komplexe Steuerungsaufgaben, • Einzelanwendungen
selbstentwickelte Computersysteme	• Leistungs- und Kostenoptimierung möglich, • durch Software beliebig hohe funktionelle Komplexität, • Änderungen unproblematisch	• besonders hohe Entwicklungsaufwendungen (es ist Hard-***und*** Software zu entwickeln!)	• Embedded Systems in höheren Stückzahlen
standardisierte elementare Schaltfunktionen	• keine Software-Entwicklung erforderlich (die Schaltung ***ist*** die Funktion), • auch mit einfachen (werkstattüblichen) Mitteln realisierbar	• vergleichsweise geringe funktionelle Komplexität (Einzweck-Hardware), • Änderung schwierig	• vergleichsweise einfache Systeme in eher geringen Stückzahlen bzw. als Einzelstücke; Versuchsaufbauten; Interface-Schaltungen, Adapter, “Restlogik” zwischen hochintegrierten Schaltkreisen usw.
programmierbare Schaltkreise	• hohe funktionelle Komplexität und hohe Leistung möglich, • Hardware auch mit einfachen (werkstattüblichen) Mitteln realisierbar , • Änderungen (meistens) unproblematisch	• Entwicklungssoftware bzw. Entwicklungsumgebung erforderlich, • Fehlersuche schwieriger als bei herkömmlichen Schaltkreisen, gelegentlich Probleme der Lieferbarkeit*)	• Hardware aller Art (vom Einzelstück an bis zur Massenfertigung)
anwendungsspezifische Schaltkreise (ASICs)	• hohe funktionelle Komplexität und hohe Leistung möglich, • kleinste Abmessungen der Hardware (geringste Schaltkreisanzahl für eine bestimmte Aufgabe)	• bei geringen Stückzahlen zu teuer, • hohe Entwicklungsaufwendungen, • Änderungen sehr kostenaufwendig	• Systeme aller Art in hohen Stückzahlen, • besonders leistungskritische Systeme

*): z. B. wenn Hersteller das Produkt vom Markt nimmt und es keinen Zweitlieferanden gibt

Tabelle 2.9 Lösungsansätze der Digitaltechnik: eine Übersicht

Entwurfsebenen

Abhängig davon, welche Mittel gegeben bzw. als zuhanden anzusehen sind, kann man zwischen verschiedenen Ebenen unterscheiden (Tabelle 2.10).

Ebene	zuhandene Mittel	typische Entwurfsaufgaben
Systeme	komplette Systeme als Verbund von Hard- und Software	Bewerten, Auswählen, Konfigurieren, Anpassen, Anwendungsprogrammierung
Geräte	PCs, SPS, Meßgeräte, Drucker, Modems usw.	Auswählen, Systemkonfiguration, Verkoppeln, Anpassen, Programmierung
Funktionseinheiten (Modul- bzw. Leiterplattenebene, Modular/PCB Level)	Steckkarten, Funktionsmoduln, andere austauschbare Baugruppen	Auswählen, Systemkonfiguration Zusammenstellen und Verkoppeln, Programmierung,
Schaltkreise und "diskrete" Bauelemente (Bauelementeebene, Component Level)	Prozessoren, RAMs, Widerstände, Transistoren usw.	Entwerfen von Funktionseinheiten und Geräten
Elementarfunktionen (vorzugsweise in Schaltkreisen) (Gatterebene, Gate Level)	Gatter, Flipflops, Treiberstufen usw.	Entwerfen von Schaltkreisen und Funktionseinheiten
elektrische Auslegung und IC-Layout (elektrische bzw. Schaltkreisebene, Circuit Level)	Halbleiterstrukturen und Metallisierungsebenen	Entwerfen von Schaltkreisen und anderen Bauelementen auf Grundlage zuhandener Wirkprinzipien und Technologien
physikalische Effekte	-	technische Ausnutzung der Effekte, Entwerfen von Schaltkreisen und anderen Bauelementen auf Grundlage neuartiger Wirkprinzipien und Technologien

Tabelle 2.10 Entwurfsebenen

Eine Ebene ist um so höher, je komplexer die zuhandene Funktionalität ist, auf der man aufsetzen kann. Es liegt nahe, die jeweils höchsten zugänglichen Ebene auszunutzen (Arbeits- und Zeitersparnis). Steht auf einer gewissen (höheren) aber keine ausreichende Funktionalität zur Verfügung (bzw. ist diese zu teuer), so ist es erforderlich, den Entwurf auf einer niederen Ebene (also gleichsam von Grund auf) zu beginnen.

Die hier in Rede stehenden Entwurfsaufgaben betreffen vorzugsweise die Bauelemente- und Gatterebene (Component Level, Gate Level).

Zur Entwurfsmethodik

Beim Entwerfen auf Bauelemente- und Gatterebene kann man zwei Vorgehensweisen unterscheiden (Tabelle 2.11).

Vorgehensweise	Beschränkungen	die meiste Mühe bereiten...	bedeutsame Fehlerquellen
Bau- bzw. Funktionselemente (Prozessoren, synchrone Speicher usw.) auswählen und zusammenschalten	man muß sich an die jeweiligen Applikationsschriften halten; vergleichsweise wenig Narrenfreiheit	das Einarbeiten in die Architektur- und Wirkprinzipien*) sowie in die Applikationsrichtlinien	unzulängliche Einarbeitung, irrtümliche Auffassung des Datenmaterials, Übersehen von Spitzfindigkeiten
Entwerfen vom Gatter an aufwärts	praktisch keine (extreme Narrenfreiheit)	das Ausarbeiten aller Einzelheiten von Grund auf sowie der Funktionsnachweis (Verifizierung)	allgemeine Entwurfs- und Flüchtigkeitsfehler

*): das können ohne weiteres einige tausend Seiten sein - die wirklich durchgearbeitet werden wollen

Tabelle 2.11 Vorgehensweisen beim Entwerfen auf Bauelemente- und Gatterebene

Herkömmlicherweise handelt es sich um einen von Intuition geleiteten, iterativen Prozeß, der mit dem Studium des zu lösenden Problems und der zuhandenen Mittel beginnt*).

*): und zwar anhand von Datenblättern, Applikationsschriften usw. Will (oder muß) man rechnergestützt entwerfen, kommt noch das Einarbeiten in die Entwicklungsumgebung dazu.

Entwurfsunterstützungssysteme erleichtern und beschleunigen zwar die Arbeit (bzw. erlauben es überhaupt erst, größere Projekte durchzuziehen), bewirken aber keine grundsätzliche Änderung der Methodik: es ist nach wie vor intuitiv zu entwerfen, aber gestützt auf leistungsfähigere Werkzeuge. Die Anbieter einschlägiger Systeme bemühen sich daher auch, das traditionelle Entwerfen weiterhin zu unterstützen. Hierzu gehören folgende Vorkehrungen:

- Eingabe von Schaltplänen (Schematic Entry), Booleschen Gleichungen und Zustandsgraphen,
- Verfügbarkeit der herkömmlichen Funktionselemente mittleren Integrationsgrades (MSI). Die aus den verschiedenen traditionellen Logikbaureihen bekannten Multiplexer, Register, Zähler usw. stehen als Bibliotheks-Elemente (Soft Macros) zur Verfügung und können so in Entwürfen von CPLDs, FPGAs, und ASICs weiterverwendet werden.

Praxistips:

1. Neue Entwicklungsumgebungen zunächst erproben, und zwar anhand wenigstens eines nicht-trivialen Vorhabens, bei dem man weiß, was herauskommen muß (z. B. indem man einen bereits bewährten Entwurf auf dem neuen System nochmals durcharbeitet).
2. Herkömmliche Funktionselemente sind nach wie vor keineswegs altmodisch. Der Rückgriff auf bekannte Zähler (z. B. ‘163), Multiplexer (z. B. ‘151), Register (z. B. ‘574) wird von modernen Entwicklungssystemen ausdrücklich unterstützt und spart doch viel Arbeit (es ist dann nicht notwendig, Zähler, Multiplexer usw. vom Gatter an selbst zu entwerfen).
3. Älteres Datenmaterial (das Innenschaltungen von solchen Bauelementen enthält) ist eine gute Quelle von Anregungen (falls ein gerade passendes Funktionselement nicht in der Bibliothek enthalten ist, kann man es gleichsam nachempfinden - infolge des Alters ist auch nicht mit patentrechtlichen Schwierigkeiten zu rechnen).

Die meisten der heutzutage anstehenden Entwurfsaufgaben betreffen die erste der Vorgehensweisen nach Tabelle 2.11: es ist so schnell wie möglich eine funktionsfähige Lösung durch Kombinieren von zuhandenen (möglichst kostengünstigen) Bauelementen und Funktionseinheiten zu finden. Hierbei verbleiben typischerweise “Restarbeiten”, die gemäß der zweiten Vorgehensweise zu erledigen sind (mit anderen Worten: komplizierte Hardware nach Kochbuch zusammenschalten; als Rest bleibt das Lösen von Anpassungs- und Schnittstellenproblemen*) mit elementaren Mitteln (Gattern, Flipflops usw.)).

*): die bisweilen recht spitzfindig sein können - “klein” heißt nicht immer “trivial”.

3. Allgemeine Grundlagen

3.1. Zugangsweisen zur Digitaltechnik: Aussagenlogik und Boolesche Algebra

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

3.2.2. 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 3.1). 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 3.1 sowohl die Bezeichnungen F und W (für "falsch" und "wahr") sowie 0 und 1 gewählt.

Abbildung 3.1 Wahrheitstabellen der elementaren Aussagefunktionen. A, B - Aussagen; R - Resultat

3.2.3. 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 ein System abstrakter 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
Eine 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 3.1 und 3.2 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 Abschnitt 3.2.1.4.)

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

Tabelle 3.1 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	$\overline{a}b$ (Inhibition)
0	0	1	1	$\overline{a}$ (Negation)
0	1	0	0	$a\overline{b}$ (Inhibition)
0	1	0	1	$\overline{b}$ (Negation)
0	1	1	0	Exclusiv-ODER; $a \oplus b$ (Antivalenz, XOR)
0	1	1	1	NAND; $\overline{a \,\&\, b}$
1	0	0	0	UND; a & b (Konjunktion, AND)
1	0	0	1	Äquivalenz; $\overline{a \oplus b}$ (Gleichheit, XNOR)
1	0	1	0	b (Identität, Pufferstufe)
1	0	1	1	$\overline{a} \vee b$ (Implikation $a \rightarrow b$)
1	1	0	0	a (Identität, Pufferstufe)
1	1	0	1	$a \vee \overline{b}$ (Implikation $b \rightarrow a$)
1	1	1	0	ODER; $a \vee b$ (Disjunktion, OR)
1	1	1	1	Festwert 1

Tabelle 3.2 Alle 16 Booleschen Funktionen von zwei Binärvariablen

Wichtige elementare Verknüpfungen
Nicht alle Verknüpfungen der Tabellen 3.1 und 3.2 sind von gleicher Wichtigkeit. Die Funktionen UND, ODER, NICHT (Konjunktion bzw. AND; Disjunktion bzw. OR; Negation bzw. NOT) wurden bereits in Abschnitt 3.1.3. beschrieben. Im folgenden wollen wir die anderen wichtigen Funktionen betrachten:

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

NAND
Es ist eine UND-Verknüpfung mit nachfolgender Negation (NOT-AND). Sie nimmt nur dann den Wahrheitswert 0 an, wenn alle Variable den Wahrheitswert 1 haben.

NOR
Es ist eine ODER-Verknüpfung mit nachfolgender Negation (NOT-OR). Sie nimmt nur dann den Wahrheitswert 1 an, wenn alle Variable den Wahrheitswert 0 haben.

Antivalenz
Diese Funktion hat immer dann den Wahrheitswert Eins, wenn sich die Wahrheitswerte der beiden Variablen unterscheiden, das heißt, wenn entweder Variable a oder Variable b wahr ist (d. h., bei Ungleichheit der Wahrheitswerte von a und b). Auf Grund dieses Entweder-Oder-Verhaltens heißt die Funktion auch ausschließendes*) oder Exclusiv-ODER (XOR).

*): die Belegung 1, 1 wird ausgeschlossen.

Äquivalenz
Diese Funktion hat immer dann den Wahrheitswert Eins, wenn die Wahrheitswerte der beiden Variablen gleich sind.

3.2.4. 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) und “falsch” (false) 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 3.3.)

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 für das Verknüpfen und Speichern zweiwertiger Angaben	Ausnutzung verschiedener Wirkprinzipien mit nichtlinearem bzw. Schwellwertverhalten (Relais, Kippschaltungen, Transistor-Schaltstufen usw.)

Tabelle 3.3 Ansätze der zweiwertigen (binären) Informationsverarbeitung

Infolge des gemeinsamen Prinzips “Zweiwertigkeit” lassen sich verschiedenen Ansätze ineinander überführen bzw. nützliche Analogien ausnutzen (Tabelle 3.4).

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[1)]
Abbildung der Wahrheitswerte “falsch” und “wahr” bzw. der Elemente der Menge B = {0, 1} auf die Funktions- bzw. Schaltzustände zweiwertig arbeitender technischer Einrichtungen	• Realisierung des numerischen Rechnens auf Grundlage binärer Zahlendarstellungen[2)], • Realisierung elementarer Informationsverknüpfungen, die sich auf umgangssprachlich-logische Zusammenhänge zurückführen lassen[1)], • Beschreibung und Optimierung der technischen Einrichtungen (Schaltalgebra)[3)]

1)...3): siehe die folgenden Hinweise im Text

Tabelle 3.4 Zur Ausnutzung des Prinzips der Zweiwertigkeit

Hinweise:

1. Siehe Heft 03.
2. Zu den grundlegenden Datenstrukturen siehe Heft 02.
3. Siehe den folgenden Abschnitt 3.2.

Achtung:

Die Analogien bzw. wechselseitigen Überführungen sind typischerweise nichts anders als Vereinbarungssache[*)]. Weitergehende Deutungen, wie sie die Umgangssprache gelegentlich nahelegt, führen gelegentlich zu irrtümlichen bzw. fehlerhaften Auffassungen. So ist eine binäre Null nicht etwa falsch; ebensowenig ist eine binäre Eins wirklich wahr.

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

Hinweis 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 im allgemeinen 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.

3.2. 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, zur Berechnung von Testbelegungen (um die Schaltungen prüfen zu können) usw.

3.3.2. Darstellungsmöglichkeiten für 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

3.2.1.1. Erfüllung einer Schaltfunktion

Der Begriff ergibt sich daraus, daß Schaltfunktionen zu technischen Zwecken gebraucht werden, daß man also mit ihnen eine Absicht verfolgt. 1. Beispiel: ein Adreßdecoder soll dann den Wert 1 liefern, wenn an seinen Eingängen eine Adresse aus dem vorgesehenen Adreßbereich anliegt. 2. Beispiel: eine Überwachungsschaltung soll den Wert 0 liefern, wenn bestimmte - als zulässig angesehene - Belegungen an ihren Eingängen anliegen. In solchen Fällen wird man nur die Belegungen aufstellen bzw. die funktionellen Zusammenhänge formulieren, die den gewünschten Funktionswert ergeben sollen (wobei es sich von selbst versteht, daß alle anderen Belegungen zum jeweils inversen Funktionswert führen). Typische Schaltfunktionen sehen also strenggenommen so aus:

f(a, b, c,...) = 1 bzw. g(a, b, c,...) = 0.

Die Funktion f ist *erfüllt* (satisfied), wenn eine aktuelle Belegung der Variablen a, b, c,... den Wert 1 liefert; die Funktion g ist hingegen erfüllt, wenn eine aktuelle Variablenbelegung den Wert 0 liefert.

In der Technik schreibt man Schaltfunktionen üblicherweise so, daß sie einem *Ausgangssignal* einen Wert zuweisen, also z. B.:

R = f(a, b, c,...).

Beachten Sie, daß ein solcher Ausdruck strenggenommen so aussieht:

f(a, b, c,..) = 1 =: R. Das heißt, daß die Funktion f(...) dem Ausgangssignal R den Wert 1 zuweist. Anders ausgedrückt: Die Funktion f(...) ist erfüllt, wenn die aktuelle Variablenbelegung eine "1"-Belegung des Ausgangssignals *R* ergibt.

Will man hingegen ausdrücken, daß die Schaltfunktion dem Ausgangssignal eine "0"-Belegung zuweist, so wird das Ausgangssignal mit einem *Negationssymbol* (siehe weiter unten Tabelle 3.5) versehen. Beispiel:

$\overline{P}$ = g(a,b,c,...) bzw. strenggenommen: g(a,b,c,...) = 0 =: $\overline{P}$

3.2.1.2. 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 vom Entwurfsunterstützungssystem automatisch erzeugt wurden).

Hinweise:

1. Oft wird in solchen längeren Variablenbezeichnungen auf Leerzeichen verzichtet. Der Grund: heutzutage läuft der Schaltungsentwurf weitgehend rechnergestützt ab. Oftmals 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.

3.2.1.3. Wahrheitstabellen

In einer Wahrheitstabelle (Truth Table) sind alle möglichen Kombinationen der Eingangsvariablen zusammen mit dem jeweiligen Funktionswert angegeben (Abbildungen 3.1, 3.2). Eine Wahrheitstabelle für n Eingangsvariable hat 2^n Einträge (Zeilen).

Abbildung 3.2 Wahrheitstabellen (Beispiele)

Erklärung:

a) grundsätzlicher Aufbau einer Wahrheitstabelle,

b) Wahrheitstabelle mit mehreren Funktionen, die von den gleichen Eingangsvariablen abhängen (zum dargestelllten Beispiel vgl. Heft 03),

c) Wahrheitstabelle einer weiteren kombinatorischen Elementarschaltung (vgl. Heft 03).

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 (vgl. Abbildung 3.2a).

Gelegentlich sind andere Reihenfolgen zweckmäßig (z. B. die gemäß dem Gray-Code; vgl. Heft 02).

3.2.1.4. Schaltgleichungen (Boolesche Gleichungen)

In Schaltgleichungen sind die Variablenbezeichner mit Symbolen verknüpft, die die elementaren logischen Funktionen repräsentieren. Die üblichen Symbole (Verknüpfungsoperatoren) sind aus Tabelle 3.5 ersichtlich.

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

1)...3): siehe Erklärung im Text

Tabelle 3.5 Symbole (Verknüpfungsoperatoren) in Schaltgleichungen

Erklärung:

1) nur anwendbar, wenn 1 Zeichen je Variable,

2) insbesondere in Schaltgleichungen, die programmierbare Logik (PLDs, GALs usw.) beschreiben,

3) bezeichnet ein "invertiert" wirkendes Signal.

Hinweis:
Auch Ausgangssignale (die auf einer Seite der Schaltgleichung allein stehen), können mit einem Negationssymbol versehen sein. Beispiel:

$\overline{P} = g(a,b,c,...)$.

Dies bedeutet: die Gleichung soll den Funktionswert Null liefern. (Man kann sie wie eine Gleichung behandeln, die den Funktionswert 1 liefern soll, und dann entsprechend negieren. Gelegentlich lohnt sich auch ein "echtes" Negieren: $P = \overline{g}(a,b,c...)$).

3.2.1.5. Belegungslisten

Es werden nur jene Variablenbelegungen in eine Liste eingetragen, für die ein bestimmter Funktionswert (entweder 0 oder 1) erfüllt ist (Abbildung 3.3).

Abbildung 3.3 Schaltfunktion, Wahrheitstabelle und Belegungslisten

Erklärung:
Schaltfunktion und Wahrheitstabelle entsprechen Abbildung 3.2a. 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. (Da der Funktionswert somit bekannt ist, wird er nicht in die Belegungsliste aufgenommen.- Vgl. aber auch den folgenden Hinweis 2.) Es gibt 2 Arten von Belegungslisten:

- die *binäre* Belegungsliste: sie entspricht unmittelbar dem Auszug aus der Wahrheitstabelle. Ihre Einträge können somit nur die Werte 0 und 1 annehmen.
- die *ternäre* 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 3.4).

Abbildung 3.4 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 Variable 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 3.3 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 (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* bezeichnet).

Anwendung: zur *Minimierung* von Schaltfunktionen (Abschnitt 3.2.5.): eine Variable, die bedeutungslos ist, muß man auch bei der technischen Realisierung der Schaltfunktion nicht berücksichtigen.

Hinweise:

1. Die Belegungsliste der negierten (invertierten) Schaltfunktion besteht aus allen Belegungen der Wahrheitstabelle, die den Funktionswert R = 0 ergeben.
2. Wie beschrieben wird der (bekannte) Funktionswert (1 oder 0) in der Belegungsliste selbst nicht mitgeführt. Er gehört aber unbedingt zur Liste. In rechnerinternen Darstellungen muß er in geeigneter Weise codiert sein (z. B. in beschreibenden Angaben) oder sich aus dem Gebrauch der Liste ergeben (indem man z. B. nach Funktionswert (1 oder 0) getrennte Listenverarbeitungs-Unterprogramme vorsieht). In bildhaften Darstellungen muß er irgendwo notiert sein oder sich eindeutig aus dem Kontext ergeben.
3. Die Interpretation des Strichelements wird in der Literatur häufig auf Don't Care beschränkt. Genaugenommen steht aber der Strich dafür, daß die betreffende Variable beide Belegungen (0, 1) annehmen kann (n Striche beschreiben einen Unterraum des jeweiligen Booleschen Raumes B^k). Entsprechend gekennzeichnete Variable sind meistens, aber nicht immer "vernachlässigbar" (= Don't Care).
4. Belegungslisten eignen sich gut für die rechentechnische Behandlung Boolescher Funktionen, sind aber durchaus auch geeignet, um Schaltfunktionen aus der Aufgabenstellung heraus zu erstellen. (Indem man mit einer spaltenförmigen Anordnung der Variablen beginnt und in dieses Schema nacheinander alle Belegungen aufnimmt, die die betreffende Schaltfunktion erfüllen sollen. Variablen, die in einer bestimmten Belegung gar keine Rolle spielen, werden mit Strichen gekennzeichnet (hier handelt es sich um "echte" Don't cares - die sich aus dem Problemverständnis heraus unmittelbar ergeben).)

3.2.1.6. Binäre Entscheidungsdiagramme

Ein binäres Entscheidungsdiagramm (Binary Decision Diagram BDD) ist ein gerichteter Graph, dessen Knoten Variable und dessen Kanten Variablenwerte repräsentieren (Abbildung 3.5).

Abbildung 3.5 Binäre Entscheidungsdiagramme (BDDs)

Erklärung:
Alle 3 Entscheidungsdiagramme beschreiben die Schaltfunktion $(a \vee b) \wedge c$. 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 3.5a): 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 von der Reihenfolge der Variablen ab, in der es aufgebaut wird. 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 Redundanzen (d. h. überflüssigen Knoten (= Entscheidungen)) beseitigt sind. Beispielsweise gibt es in Abbildung 3.5a 2 gleichartige (isomorphe) Subgraphen (*). (Wenn a = 1, so kommt man sowohl für b = 0 als auch für b = 1 auf eine Abfrage der Variablen c, die bei c = 0 den Funktionswert 0 und bei c = 1 den Funktionswert 1 liefert. Somit ist es bei a = 1 die Abfrage der Variablen b überflüssig (= redundant = don’t care). Bei a = 0 ist hingegen b abzufragen. Da b = 1 auf die gleiche Abfrage von c führt wie a = 1, genügt es, für die Abfrage von c nur einen einzigen Knoten vorzusehen.

Hinweise:

1. Für jede Schaltfunktion und Variablen-Reihenfolge gibt es genau ein reduziertes Entscheidungsdiagramm (ROBDD).
2. Binäre Entscheidungsdiagramme eignen sich praktisch nur als rechnerinterne Darstellung (in Form von Listen- bzw. Zeigerstrukturen).
3. Speicher- und Rechenzeitbedarf werden maßgeblich von der gewählten Variablenreihenfolge bestimmt (siehe hierzu auch Abschnitt 3.2.5.4.).

3.2.1.7. Schaltpläne

Schaltpläne sind zeichnerische Darstellungen, aus denen die Struktur der Schaltung eindeutig hervorgeht. Näheres in Abschnitten 3.3.1. und Kapitel 4.

3.2.2. Rechenregeln der Schaltalgebra

Verwendung von Klammern

Assoziativität: a & (b & c) = a & b & c = (a & b) & c bzw. a(bc) = abc = (ab)c

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

Kommutativität: a & b = b & a; a ∨ b = b ∨ a

Distributivität: a & (b ∨ c) = (a & b) ∨ (a & c) bzw. a(b∨ c) = ab ∨ ac

$$a \vee (b \& c) = (a \vee b) \& (a \vee c) \text{ bzw. } a \vee (bc) = (a \vee b)(a \vee c)$$

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

abc ∨ def = (a & b & c) ∨ (d & e & f)

Identität:
$a \cdot a = a$; $a \vee a = a$; $a \oplus a = 0$; $a \equiv a = 1$

Doppelte Negation:
$\overline{\overline{a}} = a$

Allgemein (wobei die Symbole *, # wahlweise für UND, ODER, Antivalenz bzw. Äquivalenz (&, ∨,⊕, ≡) stehen) gilt:

- 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).

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

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

Rechnen mit Festwerten

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

$0 \& 0 = 0; 0 \& 1 = 0; 1 \& 0 = 0; 1 \& 1 = 1; 0 \& a = 0; 1 \& a = a$

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

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

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

Reduktion auf Festwerte:

$f \& \overline{f} = 0; f \vee \overline{f} = 1; f \oplus \overline{f} = 1; f \equiv \overline{f} = 0; f \oplus f = 0; f \equiv f = 1$

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.

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

Zwei Schaltfunktionen f_i und f_i^* sind äquivalent, wenn sie die gleiche Wahrheitstabelle haben.

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

$\overline{f_1 \, f_2} = \overline{f_1} \vee \overline{f_2}$ (negiertes UND = ODER der einzeln negierten Variablen)

$\overline{f_1 \vee f_2} = \overline{f_1} \, \overline{f_2}$ (negiertes ODER = UND der einzeln negierten Variablen)

Kürzungsregeln

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

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

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

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

3.2.3. 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.

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 (Minterme):

$K_i = x_1 \& x_2 \& ... \& x_n = x_1 x_2 ... x_n$ (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 ' $\overline{x_1 x_2} x_3$.

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:

a) *disjunktive Normalform:*

$f = K_1 \vee K_2 \vee ... \vee K_n$

b) *konjunktive Normalform:*

$f = D_1 \,\&\, D_2 \,\&\, ... \,\&\, D_n$

c) *Antivalenz-Normalform:*

$f = K_1 \oplus K_2 \oplus ... \oplus K_n$

d) *Äquivalenz-Normalform:*

$f = D_1 \equiv D_2 \equiv ... \equiv D_n$

In der Praxis ist vor allem die disjunktive Normalform (DNF) von Bedeutung, gelegentlich auch die konjunktive Normalform (KNF).

Begriffsbildungen

Summen und Produkte
Gelegentlich bezeichnet man - mit Bezug auf naheliegende Analogien (vgl. die Wahrheitstabellen in Abbildung 3.1) - die Disjunktion als Summe und die Konjunktion als Produkt. Auch bestimmte symbolische Darstellungen (Tabelle 3.5) suggerieren diese Analogie (Disjunktion: $a + b$; Konjunktion: $a \cdot b$ oder ab).

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

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

Minterme
Diese Bezeichnung für Konjunktionsterme 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
Diese Bezeichnung für Disjunktionsterme 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 Konjunktions- bzw. Disjunktionsterme.

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 wenigstes eine Variable fehlt - mit anderen Worten: wenn wenigstens ein Implikand keine Elementarkonjunktion oder Elementardisjunktion ist).

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 und Belegungslisten
Die praktische Bedeutung disjunktiver Normalformen ergibt sich aus 2 Tatsachen:

1. sie hängt auf einfache Weise mit anderen Darstellungsformen zusammen (Abbildung 3.6),
2. sie entspricht direkt einer zweistufigen UND-ODER- bzw. NAND-NAND-Schaltungsstruktur (Heft 03).

Abbildung 3.6 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 (R) = 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 R = 1. Alle verbleibenden Variablenbelegungen (jene, die keiner der Elementarkonjunktionen entsprechen) belegen wir mit R = 0.

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.

Von der reduzierten zur kanonischen disjunktiven Normalform (KDNF)
Der Weg entspricht jenem von der ternären zur binären Belegungsliste (vgl. Abschnitt 3.2.1.5.):

- Implikanden, die bereits Elementarkonjunktionen sind, werden direkt in die KDNF übernommen,
- jeder weitere Implikand wird zunächst in die entsprechend ternäre Variablenbelegung gewandelt; dann werden die Strichelemente aufgelöst (vgl. Abbildung 3.4). Die resultierenden binären Variablenbelegungen werden dann wieder in Elemenarkonjunktionen gewandelt. So werden aus einem Implikanden, dem die einzige Variable x_i fehlt, 2 Implikanden (der erste mit x_i und der zweite mit $\overline{x_i}$). Aus einem Implikanden, in dem 2 Variable x_i, x_j fehlen, werden 4 Implikanden (so daß alle Kombinationen der Variablenbelegung vorkommen:
  $(\overline{x_i}, \overline{x_j};\ \overline{x_i}, x_j;\ x_i, \overline{x_j};\ x_i, x_j)$ usw.

Beispiel: aus a b wird a b $\overline{c}$ $\vee$ a b c.

Von der kanonischen zur reduzierten disjunktiven Normalform
Der einfachste Fall: zwei Elementarkonjunktionen unterscheiden sich nur in einer Variablen voneinander (die in einem Implikanden negiert und im anderen nichtnegiert vorkommt): dann können diese beiden Implikanden durch einen einzigen Implikanden ersetzt werden, in dem diese Variable fehlt.

Beispiel: a b c $\vee$ a b $\overline{c}$ = a b.

Dies entspricht dem Zusammenfassen von 2 binären Variablenbelegungen, die sich nur in einer Variablenposition voneinander unterscheiden, zu einer einzigen ternären Belegung, die in dieser Position einen Strich enthält (vgl. Abbildung 3.3).

Es entspricht sinngemäß der Anwendung der 2. Kürzungsregel (Abschnitt 3.2.2.). Im

Beispiel: a b = f_1; c = f_2; (a b) c $\vee$ (a b) $\overline{c}$ = a b.

Grundsätzlich handelt es sich hierbei um das Problem der *Minimierung* von Schaltfunktionen (Abschnitt 3.2.5.).

3.2.4. Gleichungssysteme

Mit Booleschen Gleichungen kann man ebenso umgehen wie mit den üblichen algebraischen Gleichungen: die Gleichung wird nicht verändert, wenn man auf beide Seiten die gleiche Operation anwendet.

Die allgemeine Form einer Booleschen Gleichung
Zwei Boolesche Funktionen *f(x), g(y)* werden einander gleichgesetzt:

$f(\underline{x}) = g(\underline{y})$.

Hierbei bezeichnen *x, y* beliebige Tupel Boolescher Variabler:

$\underline{x} = \{x_1, x_2,...x_n\}$, $\underline{y} = \{y_1, y_2,...y_m\}$.

Homogenisierung
Jede beliebige Boolesche Gleichung läßt sich durch Antivalenzverknüpfung in eine homogene Gleichung (deren rechte Seite Null ist) umwandeln:

$f(\underline{x}) \oplus g(\underline{y}) = g(\underline{y}) \oplus g(\underline{y})$; es gilt aber (vgl. Abschnitt 3.2.2.): $g(\underline{y}) \oplus g(\underline{y}) = 0$. Also folgt:

$f(\underline{x}) \oplus g(\underline{y}) = 0$.

Gleichungssysteme
Gelegentlich ergeben sich mehrere zusammengehörende Boolesche Gleichungen, mit anderen Worten: ein Gleichungssystem:

$f_1(\underline{x}_1) = g_1(\underline{y}_1)$,

$f_2(\underline{x}_2) = g_2(\underline{y}_2)$,

$f_3(\underline{x}_3) = g_3(\underline{y}_3)$ usw.

Die einzelnen Gleichungen kann man zunächst homogenisieren:

$f_1(\underline{x}_1) \oplus g_1(\underline{y}_1) = 0$,

$f_2(\underline{x}_2) \oplus g_2(\underline{y}_2) = 0$,

$f_3(\underline{x}_3) \oplus g_3(\underline{y}_3) = 0$ usw.

Zusammenfassung zu einer einzigen homogenen Gleichung
Mehrere homogene Boolesche Gleichungen lassen sich zu einer einzigen homogenen Booleschen Gleichung zusammenfassen, indem man sie miteinander disjunktiv verknüpft (die resultierende Gleichung ist nur dann = 0, wenn jede der disjunktiv verknüpften Gleichungen = 0 ist):

$(f_1(\underline{x}_1) \oplus g_1(\underline{y}_1)) \vee (f_2(\underline{x}_2) \oplus g_2(\underline{y}_2)) \vee (f_3(\underline{x}_3) \oplus g_3(\underline{y}_3)) \vee ... = 0$

Zur praktischen Bedeutung
Durch Homogenisieren und Zusammenfassen lassen sich viele Problemstellungen, die auf Booleschen Gleichungen beruhen, dadurch lösen, daß man die Nullstellen einer einzigen Booleschen Gleichung untersucht.

Hinweis:
Die Nullstellen einer Booleschen Gleichung ergeben sich unmittelbar aus der Wahrheitstabelle*) (indem man nur jene Variablenbelegungen betrachtet, die dem Funktionswert 0 entsprechen) oder aus einem binären Entscheidungsdiagramm (indem man die mit 0 belegten Endknoten (in Abbildung 3.5: unten) aufsucht und sich von dort aus die jeweiligen Variablenbelegungen erschließt).

*): das läuft darauf hinaus, alle Variablenbelegungen durchzuprobieren. Siehe auch Abschnitt 3.2.5.4.

Auflösen von Booleschen Gleichungen
Viele Boolesche Gleichungen (aber nicht alle!) lassen sich nach ihren einzelnen Variablen auflösen. Auflösen nach einer Variablen x_i heißt, eine gegebene Gleichung so umzustellen, daß auf der einen Seite nur die Variable x_i und auf der anderen eine Boolesche Funktion steht, die x_i nicht enthält:

Gegeben: $f(x_1, x_2,..., x_i,...x_n) = 0$; *gesucht:* $x_i = h(x_1, x_2,...x_n)$

Die Funktion $h(x_1, x_2...x_n)$ muß so bestimmt werden, daß, wenn diese Funktion anstelle von x_i in die ursprüngliche Gleichung eingesetzt wird, sich wiederum 0 ergibt:

$f(x_1, x_2,..., h(x_1, x_2,...x_n) ...x_n) = 0.$

Beispiel:
Die Gleichung $a\,\overline{b} \vee \overline{a}\,b = 0$ ist nach b aufzulösen. Hier sieht man die Lösung fast auf den ersten Blick: $\underline{\underline{b = a}}$.

Beweis durch Einsetzen: $a\,\overline{a} \vee \overline{a}\,a = 0$.

3.2.5. Schaltungsoptimierung

3.2.5.1. Optimierungskriterien

Es gibt ausgesprochen "schaltalgebraische" Optimierungskriterien und ausgesprochen praxisbezogene. Das klassische Optimierungskriterium der Schaltalgebra ist die *Minimalform* der jeweils zu realisierenden Schaltfunktion, wobei es sich darum handelt, mit möglichst wenigen Verknüpfungen auszukommen. Praxisbezogene Optimierungskriterien 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. (Ein typisches praxisbezogenes Optimierungskriterium: die minimale Anzahl der Ausdrücke (Variable + Verknüpfungszeichen), aus denen die Schaltfunktion gebildet wird.)

3.2.5.2. Optimierungs-Tricks (Auswahl)

Inverse Schaltfunktion
Will man beispielsweise eine Schaltfunktion f (a, b, c,...) = 1 verwirklichen, so kann man ausprobieren, ob die inverse bzw. negierte Schaltfunktion $\bar{f}$ (a, b, c...) = 0 sich nicht einfacher aufbauen läßt. Es reicht dann aus, einen Negator nachzuschalten.

Hinweis:
Es liegt nahe, die inverse Schaltfunktion dann zu verwenden, wenn - bei n Variablen - die Anzahl der Belegungen, die die Schaltfunktion erfüllen (= Anzahl der Elementarkonjunktionen in der KDNF) > 2^{n-1} ist (d. h., wenn mehr als die Hälfte aller überhaupt möglichen Variablenbelegungen die Gleichung erfüllen). Durch das Negieren verringert sich dann auf jeden Fall die Anzahl der Elementarkonjunktionen (bzw. die Eingangszahl der ODER-Verknüpfung einer 2-stufigen UND-ODER-Struktur). Es kann aber vorkommen, daß sich die Implikanden “weniger gut” verkürzen lassen (man hat dann zwar weniger, dafür aber längere Implikanden (mit mehr Termen (bedeutet technisch: höhere Eingangszahlen der UND-Verknüpfungen[*)]).

*): bei programmierberan Schaltkreisen kann dies durchaus vorteilhaft sein: die UND-Verknüpfungen sind ohnehin für alle in Frage kommenden Eingangsvariablen ausgelegt, ODER-Verknüpfungen sind aber gelegentlich eine kostbare Ressource).

Umgekehrte Logik bzw. Realisierungsbasis
Üblicherweise wird heutzutage die positive Logik verwendet und nur ausnahmsweise die negative Logik (vgl. Abschnitt 3.3.2.1.). Manchmal ergeben sich günstigere Lösungen,

wenn man - entgegen der Gewohnheit - die Schaltung für die jeweils andere Logik entwirft bzw. - auf Grundlage der DeMorganschen Regeln - die Realisierungsbasis wechselt (z. B. NAND statt NOR) und an passenden Schnittstellen die Variablen entsprechend negiert.

Herauslösen von Teilschaltfunktionen
Wir suchen nach Verknüpfungen von Variablen, die mehrfach vorkommen. Es liegt dann nahe, solche Verknüpfungen herauszulösen und jeweils nur einmal aufzubauen (Schaltungsbeispiele in Heft 03).

Beispiel:

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

Wir erkennen, daß die Verknüpfung *a b e* in 3 Implikanden vorkommt. Demzufolge läßt sich die Gleichung umformen:

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

Zudem könnte den eingeklammerten Ausdruck weiter 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):

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

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

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 3.7 zeigt ein Beispiel, das sich auf Abbildung 3.6 bezieht.

Abbildung 3.7 Minimierung einer Schaltfunktion

Erklärung:
Wir gehen von der Schaltfunktion aus, die in Abbildung 3.6 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.
1. 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.

*): dies kann z. B. durch einen technischen Fehler (Störung, Defekt usw.) hervorgerufen werden.

Hinweise:

1. Beim Fehlersuchen zunächst nachprüfen, ob die Schaltung korrekt angesteuert wird (unzulässige Eingangsbelegungen ausschließen).
2. Wenn unzulässige Eingangsbelegungen (1) in Fehlerfällen tatsächlich auftreten und (2) dabei nicht vernachlässigbare Folgefehler nach sich ziehen können, müssen entsprechende Überwachungs- bzw. Kontrollschaltungen vorgesehen werden.

3.2.5.3. DNF-Minimierung nach Quine-McCluskey

Das Minimierungsverfahren hat das Ziel, aus einer gegebenen kanonischen disjunktiven Normalform eine äquivalente disjunktive Normalform zu bestimmen, die (1) eine minimale Anzahl an Implikanden aufweist und deren Implikanden (2) so kurz wie möglich sind (d. h. so wenige Variable wie möglich enthalten)*).

*): das entspricht technisch einer 2-stufigen UND-ODER- bzw. NAND-NAND-Schaltung mit minimalem Aufwand (Heft 03).

Das Verfahren besteht aus 2 Schritten:

1. Schritt - Bestimmung der Primimplikanden:

1. die kanonische disjunktive Normalform (KDNF) wird in eine binäre Belegungsliste überführt,
2. es wird versucht, deren Einträge zu ternären Belegungen (Primimplikanden) zusammenzufassen. Abbildung 3.8 zeigt die Zusammenfassungsregeln.

2. Schritt - Auswahl eines minimalen Satzes an Primimplikanden:
Aus den im ersten Schritt bestimmten Primimplikanden werden jene herausgesucht, die notwendig und hinreichend sind, um die Schaltfunktion zu erfüllen.

Begriffserklärung: Primimplikand
Ein Primimplikand ist ein Implikand, der sich nicht mehr gemäß Abbildung 3.8 (gleichbedeutend: gemäß der 2. Kürzungsregel (Abschnitt 3.2.2)) mit anderen Implikanden zusammenfassen läßt.

Abbildung 3.8 Zusammenfassungsregeln beim Verfahren nach Quine-McCluskey

Erklärung:

a) erste Stufe der Zusammenfassung (binär → ternär): zwei binäre Belegungen werden zu einer einzigen ternären zusammengefaßt, wenn sie sich nur in einer einzigen Variablenposition unterscheiden. Diese wird dann mit einem Strich belegt (vgl. auch Abschnitt 3.2.1.5.).

b) Zusammenfassung ternärer Belegungen: zwei ternäre Belegungen lassen sich zusammenfassen, wenn sie (1) Striche nur in gleichen Variablenpositionen haben*) und sich (2) ansonsten nur in einer Variablenposition unterscheiden (diese wird - als Ergebnis der Zusammenfassung - mit einem weiteren Strich belegt).

*): mit anderen Worten: alle Strich-Belegungen müssen übereinstimmen.

Hinweis:
Hier wird deutlich, daß "-" nicht ohne weiteres "Don't Care" bedeutet (mit der gedanklichen Assoziation: "kann vernachlässigt werden"). Vielmehr beschreiben Strich-Belegungen jeweils einen Unterraum des durch alle Variablen gegebenen Booleschen Raums B^k (es sind alle Kombinationen der mit Strichen belegten Variablen zu berücksichtigen - vgl. Abschnitt 3.2.1.5.). Und nur Variablenbelegungen, die den gleichen Unterraum betreffen, dürfen miteinander verglichen werden.

Der *1. Schritt des Minimierungsablaufs* beruht auf einer Durchmusterung der Belegungsliste, wobei versucht wird, Belegungen zusammenzufassen, und zwar zunächst gemäß Abbildung 3.8a und nachfolgend gemäß Abbildung 3.8b. Dabei entsteht jeweils eine neue Liste. Belegungen, die sich nicht zusammenfassen lassen, verbleiben als Primimplikanden und werden in die jeweils nächste Liste übernommen. Der Ablauf endet, wenn eine Liste entstanden ist, deren Belegungen sich nicht mehr zusammenfassen lassen

Beispiel:
Folgende, durch ihre KDNF gegebene Schaltfunktion ist zu minimieren:

$$a\,\overline{b}\,\overline{c}\,\overline{d} \vee a\,b\,\overline{c}\,\overline{d} \vee a\,\overline{b}\,\overline{c}\,d \vee a\,b\,\overline{c}\,d \vee \overline{a}\,b\,c\,\overline{d} \vee \overline{a}\,\overline{b}\,\overline{c}\,d \vee \overline{a}\,\overline{b}\,\overline{c}\,\overline{d}$$

Tabelle 3.6 veranschaulicht die einzelnen Teil-Schritte des Schrittes 1 nach Quine-McCluskey.

Tabelle 3.6 Minimierung nach Quine-McCluskey: Schritt 1 (Ablaufbeispiel)

-- in gesonderter Datei TAB0136 --

Erklärung der einzelnen Spalten:

1) die einzelnen Elementarkonjunktionen der KDNF,
2) diese Spalte enthält die ursprüngliche (binäre) Belegungsliste. Im Interesse der Übersichtlichkeit sind die Einträge gemäß der Anzahl der Einsen sortiert[*)]. Hiermit beginnt die Durchmusterung der Liste. Sie wird "von unten abgebaut". Die unterste Belegung (1101) wird aus der Liste gestrichen, und alle anderen Belegungen werden mit der gestrichenen daraufhin verglichen, ob ein Zusammenfassen gemäß Abbildung 3.8a möglich ist[**)].
3) die Belegungen, die mit 1101 zusammengefaßt werden konnten, bilden die ersten Belegungen der 2. (unteren) Liste. Oben: die gemäß 2) verkürzte Liste, wovon wiederum die unterste Belegung (1001) gestrichen wird.
4) die 2. (untere Liste) wurde um die Belegungen erweitert, die mit 1001 zusammengefaßt werden konnten. Oben: die gemäß 3) verkürzte Liste, wovon wiederum die unterste Belegung (0110) gestrichen wird.
5) die Belegung 0110 läßt sich mit keiner anderen Belegung zusammenfassen. Sie bleibt somit als Primimplikand erhalten und wird direkt in die 2. (untere) Liste aufgenommen. Oben: die gemäß 4) verkürzte Liste, wovon wiederum die unterste Belegung (1100) gestrichen wird.
6) die Belegung, die mit 1100 zusammengefaßt werden konnte, wird in die 2. (untere) Liste aufgenommen. Oben: die gemäß 5) verkürzte Liste, wovon wiederum die unterste Belegung (1000) gestrichen wird.

7) die Belegung, die mit 1000 zusammengefaßt werden konnte, wird in die 2. (untere) Liste aufgenommen. Oben: die gemäß 6) verkürzte Liste, wovon wiederum die unterste Belegung (0001) gestrichen wird.

8) die beiden letzen Belegungen der 1. (oberen) Liste (vgl. Spalte 7) lassen sich offensichtlich zusammenfassen. Das Ergebnis wird in die 2. (untere) Liste übernommen, die somit komplett ist. Nun wird die 2. (untere) Liste daraufhin untersucht, ob sich Belegungen gemäß Abbildung 3.8b zusammenfassen lassen. Es sind nur Belegungen miteinander zu vergleichen, die Striche ausschließlich an gleichen Positionen haben.

9) was sich offensichtlich nicht zusammenfassen läßt, ist der verbliebene Primimplikand (a). Mit b...e kennzeichnen wir Belegungen, die sich zusammenfassen lassen.

10) die zusammengefaßten Belegungen bilden das Ergebnis. Zusammenfassungen, die jeweils das gleiche Ergebnis liefern (wie b und c bzw. d und e) werden jeweils nur einmal aufgenommen[***)]. In der so gebildeten Liste sind keine weiteren Möglichkeiten des Zusammenfassens zu erkennen, so daß die Minimierung beendet ist.

11) so sieht die minimierte (ternäre) Belegungsliste aus (die entsprechende Schaltfunktion steht unter der Tabelle).

*): hat eine Belegung eine bestimmte Anzahl von Einsen, so muß man diese lediglich mit allen anderen Belegungen vergleichen, die die gleiche Anzahl an Einsen oder eine Eins weniger enthalten (Ablaufbeschleunigung). Deshalb auch das "Abbauen von unten" (= mit den Belegungen, die die meisten Einsen enthalten).

**): die Belegungen, die sich zusammenfassen lassen, sind in der Tabelle *kursiv* dargestellt.

***): grundsätzlich heißt das: nach jeder Zusammenfassung nachsehen, ob die gleiche Belegung schon in der Ergebnis-Liste (hier: in Spalte 10) steht.

Hinweise:

1. Tabelle 3.6 soll den Optimierungsablauf Schritt für Schritt darstellen. Beim praktischen Optimieren von Hand wird man üblicherweise jede Liste nur einmal aufstellen. Wichtig ist hierbei, (1) die bereits "erledigten" Variablenbelegungen deutlich zu kennzeichnen und (2) keine Belegung zu übersehen.
2. Was auch vorkommen kann[*)]: daß ganze Spalten ausschließlich mit Strichen belegt sind. Solche Variablen sind wirklich "Don't Care" (= redundant) und können weggelassen werden.

*): im Entwurfsprozeß (bei der Überführung der Aufgabenstellung in eine Schaltungslösung) wird man zunächst intuitiv Boolesche Variable einführen und damit Schaltfunktionen aufstellen. Es kann sich dann - als Ergebnis der Minimierung - durchaus ergeben, daß einige Variable gar nicht erforderlichsind.

Im *2. Schritt* handelt es sich darum, nachzusehen, welche Primimplikanden welche Variablenbelegungen erfüllen, um einen minimalen Satz an Primimplikanden zu bestimmen. Grundlage hierfür ist eine Primimplikandentabelle (Tabelle 3.7). Diese besteht aus einer ersten Spalte mit allen in Schritt 1 (Tabelle 3.6) ermittelten Primimplikanden (entsprechend Spalte 11 von Tabelle 3.6) sowie aus aus jeweils einer weiteren Spalte für jede der Variablenbelegungen, die die Schaltfunktion erfüllen soll. Für jeden Primimplikanden wird jede Variablenbelegung, die er erfüllt, im entsprechenden Kreuzungspunkt von Zeile und Spalte mit einem X gekennzeichnet. (Hierzu ist es notwendig, die Strichelemente aufzulösen (vgl. Abbildung 3.4) und die entsprechenden Variablenbelegungen mit jenen der obersten Zeile zu vergleichen. Bei Übereinstimmung ist ein X einzutragen.)

abcd	0000	0001	1000	1100	0110	1001	1101
0110					X		
-00-	X	X	X			X	
1-0-				X		X	X

Tabelle 3.7 Minimierung nach Quine-McCluskey. Schritt 2: Primimplikandentabelle

Der Minimierungsablauf:

1. Wir suchen nach Spalten, die *nur ein einziges X* enthalten. Die Primimplikanden den jeweiligen Zeilen sind unbedingt erforderlich (*essentielle Primimplikanden*).
2. Es ist zu prüfen, welche anderen Variablenbelegungen von den essentiellen Primimplikanden gleichsam mit erledigt (abgedeckt) werden.
3. Bleiben Variablenbelegungen übrig, die nicht von essentiellen Primimplikanden abgedeckt werden, so ist nach einer minimalen Zahl von Primimplikanden zu suchen, die diese Variablenbelegungen erfüllen.
4. Das Endziel: alle Variablenbelegungen müssen erfüllt (abgedeckt (covered)) werden, und zwar mit einer minimalen Anzahl an Primimplikanden.

Im bisher behandelten Beispiel (Tabelle 3.7) sind alle Primimplikanden essentiell, denn ersichtlicherweise gibt es für jeden der Primimplikanden wenigstens eine Spalte, in der ein einziges X steht. Wir betrachten deshalb ein weiteres Beispiel (Tabelle 3.8).

abcd	0100	1000	0101	0110	1001	1010	1101
0-00 (1)	X						
-000 (2)		X					
100- (3)		X			X		
10-0 (4)		X				X	
1-01 (5)					X		X
01-- (6)	X		X	X			
-1-1 (7)			X				X

Tabelle 3.8 Eine weitere Primimplikandentabelle. (1)...(7): laufende Nummern der Primimplikanden (s. die folgende Erklärung)

Erklärung:
Die 7 Primimplikanden sollen die 7 Variablenbelegungen erfüllen. Ersichtlicherweise sind die Primimplikanden 4 und 6 essentiell, denn sie haben in jeweils einer Spalte nur ein einziges X. Wir markieren die betreffenden Zeilen (in der Tabelle durch entsprechende Hinterlegung gekennzeichnet) und suchen in diesen Zeilen die weiteren X-Einträge auf. Gibt es weitere Primimplikanden, die in den betreffenden Spalten X-Einträge haben, so sind diese überflüssig, da die betreffenden Variablenbelegungen schon vom jeweiligen essentiellen Primimplikanden abgedeckt werden.

Im einzelnen:

- Primimplikand 4 (10-0) deckt auch die Variablenbelegung 1000 ab, so daß Primimplikand 2 (-000) gar nicht mehr benötigt wird und Primimplikand 3 (100-) entfallen könnte, falls die weitere von ihm abgedeckte Belegung 1001 anderweitig abgedeckt wird (was tatsächlich der Fall ist)
- Primimplikand 6 (01--) deckt auch die Variablenbelegungen 0100 und 0101 ab. Damit könnten die Primimplikanden 1 (0-00) und 7 (-1-1) entfallen.
- Was verbleibt, sind die Variablenbelegungen 1001 und 1101. Diese werden ersichtlicherweise durch den Primimplikanden 5 (1-01) abgedeckt. Somit sind auch sämtliche von den Primimplikanden 3 und 7 abgedeckten Belegungen “erledigt”, also können diese Primimplikanden tatsächlich entfallen.
- Es verbleibt ein minimaler Satz aus den Primimplikanden 4, 5 und 6. (10-0, 1-01, 01--). Als Boolesche Gleichung: $a\,\overline{b}\,\overline{d} \vee a\,\overline{c}\,d \vee \overline{a}\,b$.

3.2.5.4. Anmerkungen zur rechentechnischen Behandlung Boolescher Gleichungen

Herkömmlicherweise wird eine "analytische" Problemlösung (z. B. das Auflösung einer Gleichung durch systematisches Anwenden von Operationen auf beiden Seiten, "Umstellen" usw.) als einzig exakte Methode angesehen - im Gegensatz zum kombinatorischen Verfahren, nämlich dem Durchprobieren aller Möglichkeiten. Der *rechentechnische* Umgang mit Booleschen Gleichungen läuft aber praktisch immer auf kombinatorische Prinzipien hinaus: es handelt sich darum, die Lösungsmenge[1)] zu ermitteln und auf bestimme Kriterien hin zu durchmustern. Bevorzugte rechnerinterne Darstellungsformen hierfür sind Belegungslisten und binäre Entscheidungsdiagramme.

Der Ansatz liegt nahe, weil die Lösungsmenge endlich ist: schlimmstenfalls sind bei n Variablen 2^n Möglichkeiten durchzuprobieren[2)]. Das bedeutet aber auch, daß alle einschlägigen Algorithmen schlimmstenfalls exponentielle Komplexität haben können (Rechenzeit- und Speicherbedarf wachsen mit der Variablenzahl gemäß $O(a^n)$ (mit $a \geq 2$))[3)]. Man ist deshalb bemüht - durch Optimierung der rechnerinternen Darstellungen und Algorithmen - wenigstes auf eine polynominale Komplexität zu kommen ($O(n^a)$ mit a = const.)[4)].

1) den elementaren Booleschen Funktionen entsprechen isomorphe Operationen über Lösungsmengen: UND ≙ Durchschnitt ($\wedge \triangleq \cap$); ODER ≙ Vereinigung ($\vee \triangleq \cup$); Antivalenz ≙ symmetrische Differenz ($\oplus \triangleq \triangle$); Negation ≙ Komplement.

2) die Lösungsmenge einer Booleschen Funktion mit n Variablen kann praktisch immer auf höchstens 2^{n-1} Belegungen gebracht werden (ist die Anzahl der Belegungen, die zur Lösungsmenge gehören $> 2^{n-1}$, so liegt es nahe, die *negierte* Funktion zu untersuchen).

3) das bedeutet: Boolesche Probleme werden von nicht einmal allzu hohen Variablenzahlen an technisch unlösbar, auch wenn die Algorithmen an sich geradezu trivial sind. Zu den Grundbegriffen der Komplexitätstheorie siehe Anhang 1.

4) für viele anwendungspraktisch wichtige Algorithmen ist das auch gelungen; polynominale Komplexität ist aber nicht für alle Booleschen Funktionen unter allen Umständen zu garantieren. Beispielsweise hängt der Rechenzeitbedarf der Verfahren auf Grundlage binärer Entscheidungsdiagramme maßgeblich von der gewählten Variablenreihenfolge ab. (Bei gewissen Schaltfunktionen führt eine ungünstige Festlegung der Variablenreihenfolge unweigerlich zu exponentiellem Rechenzeitbedarf. Das betrifft u. a. auch Schaltfunktionen, die zur Beschreibung von Additionsnetzwerken verwendet werden.)

Die meisten Booleschen Funktionen, die sich aus der Entwurfspraxis heraus ergeben, sind allerdings nicht "bösartig". Das liegt zum Teil schon in der Natur des Entwerfens begründet:

ein Entwerfer zerlegt sich sein Problem ohnehin in Häppchen, die er übersehen kann, und ein Mensch ist ohnehin kaum in der Lage, "aus dem Stand" Funktionen mit -zig Variablen hinzuschreiben. So hat es sich gezeigt, daß selbst in ziemlich anspruchsvollen Digitalschaltungen (z. B. Hochleistungsprozessoren) die weitaus meisten Funktionen weniger als 16 Variable haben. Trotzdem kann es zu Schwierigkeiten kommen. Häufige Ursachen sind:

- harmlos aussehende Funktionen, deren Lösungsmenge sehr viele Belegungen umfaßt. Beispiel: die Antivalenz (XOR-Verknüpfung). Eine Antivalenzverknüpfung von n Variablen hat als Lösungsmenge unweigerlich 2^{n-1} Belegungen (nämlich alle Belegungen mit einer ungeraden Anzahl von Einsen). Hier nützt auch das Negieren nichts!
- an sich einfache Funktionen führen in Abhängigkeit vom gewählten Rechenverfahren, von der Variablenreihenfolge usw. zu exponentiellem Rechenzeit- bzw. Speicherplatzbedarf (vgl. obigen Punkt 3).
- die einzelnen Funktionen sehen so, wie sie sich der Entwerfer ausgedacht hat, zwar harmlos aus, die Entwicklungssoftware erzeugt aber durch Rekursion, Zusammenfassung usw. Funktionen mit geradezu riesigen Variablenzahlen (so hängt z. B. der Ausgangsübertrag eines Addierwerkes von allen Eingangsvariablen ab, auch wenn der Entwerfer die Schaltung in kleine überschaubare Stücke zerlegt hat).

Hinweis:
Die rechentechnische Behandlung Boolescher Gleichungen ist eine Wissenschaft für sich (mit eigener Spezialliteratur). Im Schaltungsentwurf hat man damit praktisch nichts zu tun; die Nutzeroberfläche schirmt gleichsam den Entwerfer von der rechnerinternen Darstellung ab. Gelegentlich wäre es aber sinnvoll, zu wissen, welche Darstellungsweisen und Algorithmen verwendet werden - nämlich um in Problemfällen (zu lange Rechenzeit, Aufgabe nicht lösbar usw.) das eigentliche Problem zu erkennen (wenn man weiß, "was los ist", kann man sich oftmals behelfen, z. B. indem man kritische Schaltfunktionen in einzeln handhabbare Teilfunktionen zerlegt bzw. entsprechende Funktionseinheiten abschnittsweise zunächst einmal für sich entwirft und erprobt).

3.3. Grundbegriffe der Schaltungstechnik

3.4.2. Die technische Ausführung elementarer Schaltfunktionen

3.3.1.1. Schaltfunktion, Schaltelement und Schaltbild

Schaltfunktionen werden durch Schaltelemente technisch verwirklicht. Einfachste Schaltelemente sind *Negatoren* (Negators, Inverters) und *Gatter* (Gates). 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 3.9 veranschaulicht *Schaltsymbole* zur Darstellung von Negatoren und Gattern (die Symbolik sollte zunächst als gegeben hingenommen werden; Näheres in Abschnitt 4.32.1.). Zusammengesetzte Schaltungen entstehen, indem solche Schaltsymbole ein- und ausgangsseitig so verbunden werden, wie dies die jeweilige Schaltfunktion erfordert (Abbildung 3.10).

Abbildung 3.9 Schaltsymbole

Abbildung 3.10 Beispiele für einfache Schaltpläne

3.3.1.2. Die vollständige Realisierungsbasis

Wieviele elementare Schaltfunktionen sind notwendig, um aus einer (aus theoretischer Sicht beliebigen) Anzahl entsprechender Schaltelemente (Gatter) jede beliebige Schaltfunktion in Hardware ausführen (implementieren, "vergegenständlichen") 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 3.11 zeigt, daß man mit solchen Elementarfunktionen tatsächlich die Funktionen UND, ODER, NICHT verwirklichen kann.

Das heißt: mit nur einem einzigen Gattertyp kann man beliebig komplizierte logische Schaltungen aufbauen!

Da NAND bzw. NOR ausreicht, bildet der Verbund UND + NICHT bzw. ODER + NICHT ebenfalls jeweils eine vollständige Realisierungsbasis.

Abbildung 3.11 Elementare aussagenlogische Verknüpfungen mit NAND bzw. NOR. a) vollständige Realisierungsbasis mit NAND, b) vollständige Realisierungsbasis mit NOR

3.3.1.3. 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 3.12 veranschaulicht den Unterschied zwischen kombinatorischen und sequentiellen Schaltungen. Näheres zu kombinatorischen Schaltungen in Heft 03 und zu sequentiellen Schaltungen in Heft 04.

Abbildung 3.12 Kennzeichen kombinatorischer und sequentieller Schaltungen

3.3.2. Elementare Kennwerte

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

Statische Kennwerte
Statische Parameter 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 sind *zeitabhängige* Angaben, wie Verzögerungszeiten und Schaltzeiten.

3.3.2.1. 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 aber 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).

Sie 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 1.1.)

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.

Die beiden Wahrheitswerte 0 und 1 müssen irgendwie den beiden Signalpegeln L und H zugeordnet werden. Dafür gibt es zwei Möglichkeiten (Tabelle 3.7). Je nach Zuordnung spricht man von positiver oder von negativer Logik.

Signalpegel	Positive Logik	Negative Logik
Low	0	1
High	1	0

Tabelle 3.9 Positive und negative Logik

An sich kann man die Zuordnung beliebig treffen und auch innerhalb eines Systems wechseln; es ist eine reine Bequemlichkeitsfrage.

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

Grundsätzlich hat sich die positive Logik weitgehend durchgesetzt. Wir werden deshalb im folgenden (und in den anderen Heften) ausschließlich die positive Logik verwenden.

3.3.2.2. Wichtige Kennwerte kombinatorischer Schaltungen

Abbildung 3.13 gibt einen Überblick über Parameter, die wir in Zusammenhang mit kombinatorischen Schaltungen unbedingt benötigen. Mit Ausnahme der Schaltungstiefe sind diese Parameter auch für sequentielle Schaltungen von Bedeutung.

Abbildung 3.13 Wichtige Kennwerte kombinatorischer Schaltungen

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. Diese Werte werden wesentlich von der Versorgungsspannung und von der technologischen Grundlage der Schaltkreise bestimmt. Heutzutage ist eine Versorgungsspannung von (+) 5 \V allgemein üblich. Für moderne stromsparende Systeme bevorzugt man mehr und mehr eine Versorgungsspannung von 3,3 \V und gelegentlich auch noch geringere Spannungswerte. Hinsichtlich der technologischen Grundlagen haben wir es im wesentlichen mit TTL- und mit CMOS-Schaltkreisen zu tun. Tabelle 3.8 in Abschnitt 3.3.3. gibt einen Überblick über die üblichen Logikpegel. Diese Pegelangaben sind statische Parameter.

Verzögerungszeiten
Wenn wir an den Eingängen eines Negators, eines Gatters oder einer beliebig komplizierten kombinatorischen Schaltung nichts ändern, wird auch am Ausgang bzw. an den Ausgängen 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 an einem Ausgang bemerkbar macht, bezeichnen wir als Verzögerungszeit (Propagation Time, Propagation Delay). Für ein Gatter bzw. einen Negator liegt sie üblicherweise im Bereich einiger Nanosekunden (Tabelle 3.8 in Abschnitt 3.3.3.). Die Verzögerungszeiten sind dynamische Parameter.

*): oftmals haben beide Signaländerungen die gleiche Verzögerungszeit ($t_{PHL} = t_{PLH} = t_P$), aber nicht immer. *Faustregel:* als allgemeine Verzsögerungszeit t_P stets den größeren der beiden Werte ansetzen.

Schaltungstiefe
Die Schaltungstiefe ist eine Zahl, die angibt, wieviele Gatter bzw. andere kombinatorische Schaltelemente zwischen dem jeweils betrachteten Ausgang und jenen Eingängen liegen, von denen die Ausgangsbelegung logisch abhängt (das klingt kompliziert, ist aber - vgl. Abbildung 3.13 - ganz einfach: im Beispiel hat die Schaltungstiefe den Wert 3; wir müssen nur nachzählen, wieviele Gatter "hintereinanderhängen").

Durchlaufverzögerung
Ein Gatternetzwerk der Schaltungstiefe s hat - roh gerechnet - eine Durchlaufverzögerung von $s \cdot t_p$. Wenn wir für t_p einen großzügigen Wert ansetzen (vgl. obige Faustregel), können wir die Verbindungen zwischen den Gattern vernachlässigen. (Bei extremen Anforderungen sind solche Vereinfachungen allerdings nicht mehr zulässig.)

Lastfaktoren
Ein Lastfaktor besagt, wieviele Eingänge (nachfolgender Schaltelemente) man einem Ausgang nachschalten darf (ausgangsseitiger Lastfaktor, Fan Out) bzw. wie ein bestimmter Eingang den vorgeschalteten Ausgang anteilig belastet (eingangsseitiger Lastfaktor, Fan In). Der jeweils konkrete Wert hängt im einzelnen von der Schaltkreis-Technologie ab.

Hinweise zur vorläufigen Orientierung:

1. In TTL-Schaltungen kann man üblicherweise von einem Ausgang aus wenigstens 10 nachfolgende Schaltkreiseingänge ansteuern kann (in diesem Fall ist der Fan Out = 10). Der Lastfaktor wird hier durch die Ausgangs- und Eingangsströme bestimmt.
2. In CMOS-Schaltungen fließt bei statischer Ansteuerung praktisch kein Strom. Wieviele Eingänge man einem Ausgang nachschalten kann, wird typischerweise durch die kapazitive Belastung bestimmt. Je größer die kapazitive Belastung, desto höher die Schaltzeiten. An übliche CMOS-Ausgänge kann man typischerweise 3...5, an Treiberstufen 10 und mehr Eingänge anschließen, ohne daß die Schaltzeiten merklich ansteigen.

Besondere Begriffsbildungen und Ausdrucksweisen
Im Fachjargon - namentlich im US-amerikanischen - sind oft bildhafte Ausdrücke üblich, um Signalwirkungen zu bezeichnen. Im strengen Sinne exakt ist das zwar nicht, man kann sich aber schnell hineinfinden. Es folgen einige elementare Beispiele:

Aktiv - inaktiv
Ein Signal ist *aktiv* (erregt, asserted, valid) wenn es seine jeweilige Wirkung ausübt, ansonsten ist es *inaktiv* (nicht erregt, in Ruhe, deasserted, invalid). (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.) Findet die jeweilige Wirkung statt,

wenn das Signal auf High-Potential liegt, so nennt man es "aktiv High", andernfalls "aktiv Low". "Aktiv Low" wirkende Signale werden in Schaltbild und Signalbezeichnung üblicherweise durch das jeweilige Negationssymbol (Tabelle 3.5) gekennzeichnet (ein aktiv Low wirkendes Speicheranforderungssignal heißt dann beispielsweise MREQ#) .

Manche Signale haben eine auswählende Wirkung; sowohl Low als auch High bewirken irgend etwas. Gelegentlich kommen in den Signalbezeichnern beide Wirkungen zum Ausdruck.

Hinweis:
In den Handbüchern der Schaltkreishersteller finden wir oft entsprechende Erklärungen für die jeweils gewählte Bezeichnungsweise (Signal Conventions). Beispielsweise wendet die Fa. Intel in ihrem neueren Datenmaterial ziemlich konsequent die Regel an, in Signalbezeichnern alle Wirkungen auszudrücken. *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-Belegung zustande kommt. Das heißt: M/IO# = 0: E-A-Zugriff; M/IO# = 1: Speicherzugriff.

3.3.2.3. 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 3.14 veranschaulicht Impulsverläufe und wichtige Kennwerte in der Gegenüberstellung von tatsächlichem Signalverlauf und linearisierter Darstellung.

Abbildung 3.14 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 (gemäß einer Exponentialfunktion). 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 L-H- bzw. H-L-Flanke beziehen sich direkt auf die Richtung des Pegel-Übergangs. 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. Sie werden oft zwischen 10% und 90% des jeweiligen Signalhubes gemessen, manchmal auch zwischen 20% und 80%. Gelegentlich brauchen Sie die Zeit zwischen 0% und 100%. Abbildung 3.14 enthält die Umrechnungsfaktoren.

Flankensteilheit bzw. Anstiegsgeschwindigkeit
Dieser Kennwert gibt die Pegeländerung bezogen auf die Zeit an. Sie wird in V/ns bzw. V/µs gemessen. Die verschiedenen Logikbaureihen erfordern bestimmte Mindest-Flankensteilheiten.

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 deshalb die Angaben als Anstiegszeit, als Anstiegsgeschwindigkeit oder Flankensteilheit (im eigentlichen Sinne) und als Anstiegsrate bezeichnen. Die Definitionen und die jeweiligen Umrechnungen gehen aus Abbildung 3.15 hervor.

Abbildung 3.15 Kennwerte von Signalflanken

Anstiegs- und Abfallzeit
Strenggenommen ist die Anstiegszeit die Zeit, die für einen Low-High-Übergang erforderlich ist (Rise Time t_R oder t_{LH}). Die Zeit, die ein High-Low-Übergang benötigt, heißt Abfallzeit (Fall Time t_F oder t_{HL}). Oft spricht man aber nur von der Anstiegszeit (auch von der Anstiegsgeschwindigkeit usw.) und meint damit beide Flanken.

Anstiegszeit
Die Anstiegszeit kennzeichnet die Zeit, die für einen Signalwechsel zwischen zwei definierten Pegeln erforderlich ist. In der Praxis bezieht man die Anstiegszeit üblicherweise auf einen Bereich zwischen 10% und 90% des jeweiligen Endwertes. (Beachten Sie, daß die Endwerte hier die durch den maximalen Low- und den minimalen High-Pegel gegeben sind.)

Anstiegsgeschwindigkeit
"Anstiegsgeschwindigkeit" und "Flankensteilheit" werden vielfach gleichbedeutend 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 ∨ 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 (BCT/ABT), Schottky-TTL-Baureihen). Typische Flankenanstiegszeiten: 2...3 ns; typische Flankensteilheiten 1...2 V/ns.

Hinweise:

1. In der Praxis ist darauf zu achten, daß die einschlägigen Anforderungen auch eingehalten werden. "Langsamere" Flanken sind nur dann zulässig, wenn die betreffenden Schaltkreis-Eingänge dafür ausgelegt sind (z. B. im Falle von Schmitt-Trigger-Eingängen; vgl. Heft 03). Sind Fehler zu suchen, sollte man sich aber auch ein solches Signal daraufhin ansehen, ob die geringe Flankensteilheit tatsächlich funktionell begründet ist*).
2. Manchmal ist es sogar von Vorteil, Flanken zu "verschleifen", also die Flankensteilheit zu vermindern (Reduktion von Störungen; vgl. Heft 17).

*): ein nicht seltener Fehler, der zu “verschliffenen” Flanken führt: falsch bestückte (d. h. mit Logik-Signalen verbundene) Stützkondensatoren.

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).

Hinweise:

1. Abbildung 3.14 zeigt u. a. auch gebräuchliche Darstellungen von Impulsverläufen. Auf solche *Impulsdiagramme* werden wir in Abschnitt 4.3.3.2. genauer eingehen.
2. Auch meßtechnisch treten Impulse oft als rechteckförmige Signalverläufe in Erscheinung. Das betrifft Impulsdiagramm-Anzeigen auf Logikanalysatoren und Signaldarstellungen auf dem Oszilloskop, wenn die zeitliche Auflösung nicht allzu fein gewählt wird

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, kein "Prellen"). Siehe dazu die Abbildungen 3.16 und 3.17.

Abbildung 3.16 Idealisierte Impulsdarstellung und tatsächliche Impulsverläufe

Abbildung 3.17 Impulsflanken in einer TTL-Schaltung (Beispiele)

Erklärung:

1) zwar etwas "lahm", aber noch o. k.,
2) o. k.,
3) o. k.,
4) nicht steil genug. Nur akzeptabel, wenn auf Schmitt-Trigger-Eingänge geführt. An “gewöhnlichen" TTL-Eingängen fehlerhaft!
5) zu starkes Unter- und Überschwingen. Überschwingen muß nicht stören, Unterschwingen wird üblicherweise in den Schaltkreis-Eingangsstufen begrenzt. Die gezeigte Größenordnung (> 1 V) ist aber zuviel! (Im besonderen an den Eingängen von Speicherschaltkreisen.)
6) High-Pegel bei ungefähr 2,5 ∨ ist für TTL o. k. (und somit auch für HCT bzw. ACT). Für "gewöhnliche" CMOS-Eingänge (4000, HC, AC) ist das aber zuwenig (es sind mindestens 3,5 ∨ erforderlich).
7) für TTL vollauf akzeptabel (2,4 ∨ High-Pegel wird schnell erreicht),
8) Fehler! Zu geringer High-Pegel. (Ausgang überlastet, z. B. durch Schluß mit einem anderen Signal oder Buskonflikt.)
9) Fehler! Low-Pegel wird nicht erreicht. (Ausgang überlastet, z. B. durch Schluß mit einem anderen Signal oder Buskonflikt.)

Impulsfolgen
In der Praxis haben wir es weniger mit Einzelimpulsen, sondern 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 3.18 veranschaulicht wichtige Parameter regelmäßiger Impulsfolgen, die insbesondere bei Taktsignalen von Bedeutung sind.

Abbildung 3.18 Parameter regelmäßiger Impulsfolgen

Zykluszeit
Ein Zyklus (Taktzyklus) wird durch zwei gleichgerichtete (ansteigende oder abfallende) Flanken begrenzt. 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, Duty Cycle
Das Tastverhältnis ist das Verhältnis zwischen Impulsdauer und Impulsabstand. Es wird üblicherweise als "echtes" Verhältnis $t_I : t_A$ angegeben (die Division wird nicht ausgeführt). Gleichbedeutend ist die Prozentangabe des Duty Cycle, wobei die Impulsdauer (t_I) auf die gesamte Zykluszeit (t_c) bezogen wird.

Rechengang: $\frac{t_A}{t_c} \cdot 100\%$

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).

Hinweis:
Taktsignale sind - auch in ihren Feinheiten - ganz wichtig für das zuverlässige Arbeiten moderner Hardware. Heutzutage sind Taktfrequenzen von 100 MHz und mehr üblich. (Das entspricht den Trägerfrequenzen des UKW-Rundfunks bzw. des Fernsehens im UHF-Bereich!) Alle Parameter müssen aufs genaueste eingehalten werden, und zwar an allen Stellen, wo der Takt benötigt wird. Wir haben es also auch mit dem Problem der *Taktverteilung* zu tun (Einzelheiten in Heft 20).

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; vgl. Heft 04).

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 3.19 veranschaulicht wichtige Begriffe und Kennwerte von Gültigkeitsimpulsen.

Abbildung 3.19 Gültigkeitsimpuls (Takt, Strobe) in Bezug auf Daten

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. In manchen Fällen kann (für eine der beiden Zeitangaben) ein Mindestwert von 0 spezifiziert sein.

- Vorhaltezeit = 0: die Daten dürfen gleichzeitig mit der betreffenden Flanke des Gültigkeitsimpulses ankommen,
- Haltezeit = 0: die Daten dürfen ungültig werden (= sich ändern), sobald die betreffende Flanke des Gültigkeitsimpulses eintrifft (eine wichtige Anwendung: in diesem Fall darf man den Ausgang des Speicherelementes (Flipflos) unbedenklich auf seinen Eingang zurückführen).

Welche Konsequenzen die Nichteinhaltung (Verletzung der Spezifikation) haben kann, wird in Heft 04 erklärt (Stichwort: Metastabilität).

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

Zustandssteuerung
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.

Hinweis:
In letzter Konsequenz wirken die Gültigkeitsimpulse an den Takteingängen von Latches und Flipflops. Nähere Einzelheiten behandeln wir deshalb an Ort und Stelle, also in Heft 04.

3.3.3. Logikbaureihen

Um "logische" Funktionen zu verwirklichen, kann man beliebige Effekte technisch ausnutzen (aussagenlogische Verknüpfungen und das Speichern von Bits gelingen mit mechanischen Mitteln, mit pneumatischen, optischen usw.). Dem Stand der Technik gemäß haben wir es jedoch fast ausschließlich mit integrierten Schaltkreisen zu tun, die aus der Sicht der Digitaltechnik je nach Technologie und interner Schaltungsauslegung verschiedenen *Logikbaureihen* angehören bzw. zu diesen kompatibel sind. Tabelle 3.8 gibt einen Überblick über herkömmliche Logikbaureihen.

Technologie	Baureihe	Speise-spannung (V_{CC} /V_{DD})	max. Low-Pegel (V_{Lmax})	min. High-Pegel (V_{Hmin})	max. Eingangsstrom		min. Ausgangsstrom		Typische Verzögerungs-zeiten
					Low (I_{IL})	High (I_{IH})	Low (I_{OL})	High (I_{OH})	
TTL	74	5 V	0,7 V	2,4 V	1,6 mA	40 µA	16 mA	0,4 mA	12 - 22 ns
	74 LS				0,4 mA	20 µA	8 mA	0,4 mA	10 - 15 ns
	74 S				2 mA	50 µA	20 mA	1 mA	3 - 5 ns
	74 AS				0,5 mA	20 µA	20 mA	2 mA	1 - 4,5 ns
	74 F				0,6 mA	20 µA	20 mA	1 mA	3,7 - 5 ns
	74 ALS				0,1 mA	20 µA	8 mA	0,4 mA	3 - 11 ns
CMOS	4000	3 - 15 V	≈ 20% V_{DD}	≈ V_{DD}	1 µA		0,4-3 mA		20 - 60 ns
	74 HC	5 V	1,3 V	3,5 V			4 mA		8 - 15 ns
	74 HCT		0,7 V	2,4 V			4 mA		14 - 18 ns
	74 AC		1,3 V	3,5 V			24 mA		5 - 10 ns
	74 ACT		0,7 V	2,4 V			24 mA		7 - 12 ns
	74 LV	3,3 V	0,6 V	2,1 V			8 mA		9 ns
	74 LVC		0,6 V	2,1 V			24 mA		4 ns
	74 LVT		0,7 V	2,4 V			64 mA	32 mA	3 ns
BiCMOS	74 BCT	5 V	0,7 V	2,4 V	< 1mA		64 mA	15 mA	7 ns
	74 ABT				1 µA		64 mA	32 mA	5 ns
ECL	10k	- 5,2 V	- 1,8 V	- 0,8 V	0,5 µA	350 µA	25 mA		2 ns
	100k	- 4,5 V							0,5 - 1,5 ns

TTL-Pegel, TTL-Kompatibilität

Wenn allgemein von "TTL-Pegeln" bzw. von “TTL-Kompatibilität" die Rede ist, meint man damit, daß entsprechende Schnittstellensignale folgende Parameter einhalten:

- Low-Pegel, eingangsseitig: 0...0,8 V,
- High-Pegel, eingangsseitig: 2 V...V_{CC} (d. h., ein Eingang sieht von 2 ∨ an ein High-Signal),
- Low-Pegel, ausgangsseitig: 0...0,4 ∨ (typisch: 0,2 V; das heißt, das Low-Signal überschreitet nie 0,4 ∨ und liegt meistens um 0,2 V).
- High-Pegel, ausgangsseitig: 2,4 V...V_{CC} (typisch: 3,4 V; das heißt, ein Ausgang liefert unter Betriebslast wenigstens 2,4 V, meistens aber ≥ 3,4 V),

Herkömmliche und moderne Logikfamilien

Der herkömmliche “Industriestandard” für Logik-Speisespannungen: V_{CC} = 5 ∨ (Bereich 4,5 ... 5,5 V), der modernere: V_{CC} = 3,3 V (Bereich 2,7...3,6 V). Moderne Logikbaureihen sind typischerweise TTL-kompatibel, unterscheiden sich aber in folgenden Punkten: (1) was halten die Schaltkreise an ihren Anschlüssen aus (maximale Eingangsspannungen)?, (2) was liefern sie tatsächlich?, (3) wo liegen die Schaltschwellen zwischen Low und High (wichtig für die Störsicherheit)? Siehe dazu Abbildung 3.20. Die verschiedenen Kopplungsmöglichkeiten zwischen 5-V- und 3,3-V-Familien sind in Tabelle 3.9 zusammengestellt.

Abbildung 3.20 Logikpegel. V_{il}, V_{ih}: Eingangsspannungen, V_{ol}, V_{oh}: Ausgangsspannungen, V_{th}: Schaltschwelle. (Nach: Texas Instruments)

Quelle (Ausgang)	Verbraucher (Eingang)	Kopplung
5 ∨ TTL	3,3 V	o.k. (Vorsicht bei LV - zulässige Eingangsspannung max. V_{CC} + 0,5 V)
5 ∨ CMOS	3,3 V	ohne Maßnahmen zur Pegelwandlung nur auf LVC und LVT
3,3 V	5 V TTL	o.k.
3,3 V	5 V CMOS	High-Pegel reicht nicht aus; besondere Pufferschaltkreise (Level Shifting Transceivers) mit 2 Speisespannungen erforderlich (z. B. SN74ALVC164245)

Tabelle 3.10 Kopplung von Logikfamilien mit verschiedener Speisespannung

3.3.4. Transfer Gates und digitale Schalterbauelemente

Die Source-Drain-Strecke eines Feldeffekttransistors (FETs) wird leitend, wenn man auf das Gate eine Ladung gibt (d. h., an das Gate eine positive Spannung anlegt)[*)]. Ein solcher Feldeffekttransistor verhält sich also analog zu einem Schalter mit Arbeitskontakt (Abbildung 3.21).

*): wir beschränken uns hier auf n-Kanal-FETs vom Anreicherungstyp.

Abbildung 3.21 Schalter und Gatter

Erklärung:

a) der Feldeffekttransistor als Schalter,

b) ein herkömmlicher Schalter (zum Vergleich),

c) Aufbau eines Gatters,

d) Besonderheiten: wir betrachten einen Signalfluß von Seite A zu Seite B. Eine Treiberstufe (z. B. als Ausgangsstufe eines Gatters) entkoppelt beide Seiten elektrisch voneinander (Seite B wird aktiv getrieben und wirkt somit nicht auf Seite A zurück). Ein geschlossener Schalter ist hingegen praktisch dasselbe wie ein Stück Draht:
   - Seite A sieht alle Störungen auf Seite B,
   - Seite A sieht die parasitäre Kapazität C_p der Seite B,
   - wird der Schalter geschlossen, so muß ein Ausgang (Treiberstufe) auf Seite A auch die Lasten von Seite B mit treiben.

Ebenso wie man mit Kontaktbauelementen logische Verknüpfungen realisieren kann (Heft 08) gelingt dies mit Feldeffekttransistoren. Tabelle 3.10 nennt die wesentlichen Unterschiede zwischen Gatter- und Schalterbauelementen.

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 3.11 Gatter- und Schalterbauelemente

FET-Schalter sind überall dort vorteilhaft einsetzbar, wo es tatsächlich um das Durchschalten von Signalwegen geht (Beispiele u. a. in Heft 03). Sie gibt es in zwei Grundformen:

1. als sog. Transfer Gates im Innern hochintegrierter Schaltkreise,
2. als digitale Schalterbauelemente in Form von Busschaltern (Abbildung 3.22), Multiplexern (Heft 03) usw.

Derartige Bauelemente werden unter verschiedenen Handelsnamen angeboten (Crossbar, QuickSwitch usw.). *Sonderanwendungen* betreffen u. a. die Pegelwandlung beim Zusammenschalten verschiedener Logikbaureihen, das teilweise Abschalten der Speisespannung (zwecks Stromsparen ausgeschaltete Funktionseinheiten in einer eingeschalteten Umgebung: Partial Power Down) sowie das Austauschen steckbarer Funktionseinheiten im arbeitenden System (Hot Plugging). Nähere Einzelheiten in Heft 19.

Abbildung 3.22 Ein Ausführungsbeispiel: der Achtfach-Schalter 74CBT3245 (Datenblattauszug; Texas Instruments)

Erklärung:
Der Schaltkreis ist in der Lage, 8 Signalwege zu trennen bzw. bidirektional durchzuschalten. In der Anschlußbelegung entspricht er dem verbreiteten Bus-Transceiver '245. Zu beachten ist, daß das CBT-Bauelement im Gegensatz zum Transceiver nur durchreicht oder sperrt (genauso wie etwa ein Relais-Kontakt), also im Ein-Zustand weder beide Seiten voneinander entkoppelt (das Bauelement ist nicht rückwirkungsfrei) noch die Signale regeneriert (das Bauelement hat keine eigene Treibfähigkeit).

3.3.5. Freie Eingänge und Festwerte

Gelegentlich braucht man in einer Schaltung weniger Eingänge als am Schaltkreis vorhanden sind. Man hat dann freie Eingänge, die irgendwie zu beschalten sind. Diese Beschaltung läuft auf das allgemeinere Problem hinaus, *Festwerte* (Low und High) bereitzustellen. Abbildung 3.23 zeigt einige bewährte Lösungen.

Abbildung 3.23 Beschaltung freier Eingänge und Bereitstellung von Festwerten. *): bei den meisten Logikbaureihen ist als Festwert High auch ein Direktanschluß an V_{CC} zulässig

Hinweise:

1. Freie Eingänge sollten *nie* offengelassen werden! (Offene - unbeschaltete - Eingänge wirken als "Antennen" für Störungen aus der Außenwelt und erhöhen - insbesondere bei CMOS- Schaltungen - die Stromaufnahme, beeinträchtigen also insgesamt die Zuverlässigkeit.)
2. In *TTL-Versuchsaufbauten* dürfen in manchen Fällen Eingänge durchaus offen bleiben (offener Eingang = High-Belegung).
3. *CMOS*-Eingänge *nie* offenlassen! (Offener Eingang = undefinierte Belegung = Stromfluß “quer” durch den Schaltkreis. Viele offene Eingänge = starker Stromfluß → Schaltkreis wird wärmer als üblich und kann kaputtgehen!)
4. Ungenutzte Eingänge sind typischerweise so zu beschalten, daß der Festwert die Funktion nicht blockiert (UND/NAND: High; OR/NOR: Low; ungenutzte Steuereingänge an Flipflops, Zählern, Treibern usw. mit dem jeweils “inaktiven” Logikpegel).
5. Auch die Festbeschaltung kann eine Fehler-Ursache sein (ein inkorrekter oder fehlender Festwert oder auch ein Fehler im Schaltkreis, der verhindert, daß der von außen zugeführte Festwert im Gatter zur Wirkung kommt).

4. 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. 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.).

4.1. Dokumentation von Systemen, Geräten und Bauelementen

Systeme bzw. Geräte werden zumeist in *Handbüchern* (Reference Manuals) und in *Zeichnungssätzen* dokumentiert, Bauelemente in *Datenblättern* bzw. Datenbüchern (Data Sheets, Data Books). Tabelle 4.1 gibt einen Überblick darüber, welche Dokumentation beim Entwickeln elektronischer Geräte auszuwerten bzw. zu erstellen ist.

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 4.1 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).

4.2. 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.

4.3.2. 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 ("Drähte", 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.

4.2.1.1. 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, wie Gattern oder Flipflops, deren elektrisches "Innenleben" vollkommen uninteressant ist, um die Schaltung zu verstehen. Deshalb werden logische Schaltungen vorwiegend durch Schaltsymbole für logische Funktionseinheiten (vom Gatter bis zum Prozessorschaltkreis) dargestellt, wobei die elektrischen Anschlüsse (z. B. für die Versorgungsspannung) oft nicht mitgezeichnet werden. 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- bzw. Bauelemente.

4.2.1.2. Schaltsymbole (Schaltzeichen)

Trotz vieler Bemühungen um Standardisierung gibt es keine weltweit einheitliche Symbolik. In Deutschland ist DIN 40900 der grundlegende Standard. Abbildung 4.1 stellt verschiedene Schaltsymbole "logischer" Funktionselemente gegenüber.

Abbildung 4.1 Schaltsymbole für Gatter

Abbildung 4.2 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*).

*): siehe dazu auch Abschnitt 4.2.1.4. Zu nicht wenigen Entwicklungsumgebungen muß der Symbolvorrat gemäß DIN 40 900 gesondert bestellt (und extra bezahlt) werden! (Manchmal ist er überhaupt nicht vorgesehen.)

Kombinatorische und sequentielle Schaltungen
In der modernen Symbolik gibt es keine Besonderheiten, wodurch sich Schaltplan-darstellungen kombinatorischer und sequentieller Schaltungen auf den ersten Blick unterscheiden (alle Symbole haben Kästchenform). Hingegen erkennt man in den eher

traditionellen Darstellungen sequentielle bzw. Speichermittel enthaltende Funktionselemente an der Kästchenform, während Gatter als halbkreisförmige, bogenförmige oder abgerundete Symbole erkennbar sind.

4.2.1.3. 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 Sternen oder senkrechten und waagerechten Strichen, "zusammenstückeln" zu können.) Im Laufe der Zeit hatte man sich ein bedeutend ehrgeizigeres Ziel gestellt: eine symbolische Darstellung auch komplizierter logischer Funktionseinheiten, aus der die Funktion 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 4.3 bis 4.5 sollen einen ersten Eindruck von dieser Symbolik vermitteln.

Abbildung 4.3 Wichtige Gestaltungsprinzipien der Schaltsymbole nach DIN 40900 bzw. IEEE 91-1984

Abbildung 4.4 Anordnung der Funktionskennzeichnungen (Qualifiers) im Schaltsymbol

Hinweis:
Zu Einzelheiten siehe Anhang 2.

Abbildung 4.5 Beispiele für Schaltsymbole nach DIN 40900 bzw. IEEE 91-1984 (Texas Instruments)

Hinweis:
Die Zahlenangaben, wie 10, 257, 353 usw. kennzeichnen den Schaltkreistyp. Vgl. dazu die Hefte 03, 04 und 19.

4.2.1.4. 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 des Steuerschaltkreises einer PC-Hauptplatine 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 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 4.6 gibt einen Überblick über diesen "privaten" Standard, und zwar anhand von Beispielen (die Sie mit Abbildung 4.5 vergleichen sollten).

Abbildung 4.6 Standard für eigene Schaltpläne (anhand von Beispielen)

Hinweis:
Moderne Entwicklungssysteme verzichten auf die Hilfslinien. Dies ist auch ohne weiters annehmbar, da sich durch Wahl verschiedener Linienstärken, Schriftgrade usw. die Symbole hinreichend übersichtlich gestalten lassen.

4.2.1.5. Verbindungen

Verbindungen 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; vgl. die einschlägigen Abbildungen in Heft 04). 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 4.7).

Abbildung 4.7 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 entsprechen den Variablennamen, die wir aus Kapitel 3 kennen. In der Praxis haben nicht alle Verbindungen Signalbezeichner. Am Rand des Schaltplanes bzw. des einzelnen Blattes sollte jede hinein- bzw. herausführende Leitung einen Bezeichner haben (s. oben); "mittendrin" benennt man hingegen oft nur besonders markante Signale. (Manche

Entwicklungssysteme vergeben automatisch Signalbezeichner, bisweilen auch für alle Signale. Allerdings ist dies kaum eine Hilfe, um sich die Bedeutung bzw. Funktion klarzumachen, denn mnemonisch sinnvoll werden solche Bezeichner in der Regel nicht gebildet.)

Masse und Speisespannungen

In "logischen" Schaltplänen verzichtet man zumeist weitgehend auf die Darstellung der Spannungsversorgung. Wenn überhaupt, sind deren Besonderheiten, z. B. Stützkondensatoren, zusammengefaßt dargestellt (üblicherweise auf einem besonderen Blatt oder in irgendeiner Ecke). Manchmal muß man aber auch solche Verbindungen in der Logik kenntlich machen. Abbildung 4.8 veranschaulicht die entsprechende Symbolik

Abbildung 4.8 Masse und Speisespannungen in Logik-Schaltplänen

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 4.9). 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 Vollinien 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 4.9 rechts), in manchen gibt man sich hingegen mit einer mehr schematischen Darstellung zufrieden (Abbildung 4.9 links). Die Signale sind entweder durchnumeriert oder einzeln mit Signalnamen (Identifiers) bezeichnet. Um den Signalfluß im Detail zu erkennen, müssen Sie also den Kabelbaum selbst sowie die jeweiligen Leitungsnummern verfolgen.

Abbildung 4.9 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 4.10).

Abbildung 4.10 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 mit den Funktionselementen 8, 10, 11 und 13.

4.2.1.6. 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.

4.2.1.7. 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, Übernahmeerlaubnis, Aufschaltverhinderung, Funktionsauswahl usw. Beim Leiterplattenentwurf und 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.

Hinweise:

1. Beim Messen sind die technischen Anschlußbezeichnungen (Pin-Nummern) wichtiger als die logischen Funktionskennzeichen, beim Entwickeln bzw. Einarbeiten verhält es sich zumeist umgekehrt.

2. Haben wir die Pin-Nummern, können wir die Funktionskennzeichnung aus dem Datenblatt entnehmen. Andernfalls wissen wir aber nicht immer, wo wir messen sollen, auch wenn wir im Datenblatt nachschlagen können (die Zuordnung ist nicht immer eindeutig).

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 etwa eine einfachere Leiterzugführung auf der Leiterplatte zu erreichen. Das Vertauschen ist auch in weniger einsichtigen Fällen möglich, so an Multiplexern, Decodern (Abbildung 4.11) 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. Wer Leiterplatten "von Hand" entwirft, wird sich wohl nur selten an solche Tricks wagen. Es gibt aber Entwurfsautomatisierungsprogramme, für die das Durchprobieren solch komplizierter Vertauschungen keine Schwierigkeit darstellt.

Abbildung 4.11 Decoder mit vertauscht angeschlossenen Adreßsignalen

Erklärung:

Naheliegenderweise soll der Decoder 8 Ausgangssignale DEC7...0 im 1-aus-n-Code erregen, und zwar DEC0 bei AD2...0 = 0H, DEC1 bei AD2...0 = 1H usw. Die Adreßsignale sind aber nicht so angeschlossen, wie dies die logische Struktur der Adresse erwarten läßt. Trotzdem funktioniert die Anordnung. Hierzu muß man auch die Ausgangsleitungen DEC7...0 entsprechend vertauscht an den Decoder anschließen.

Vorgehensweise:

- eine erste Wahrheitstabelle gemäß Beschaltung aufstellen (in der Abbildung links).
- eine zweite Wahrheitstabelle gemäß der gewünschte logischen Funktionsweise aufstellen (in der Abbildung rechts),
- zu jeder AD-Belegung der zweiten Tabelle die gleiche Belegung in der ersten suchen und die Ausgangsbelegung übernehmen.

4.2.2. Blockschaltbilder

Solche Schaltbilder sollen einen Überblick über Aufbau und Funktionsweise vermitteln. Die Symbolik ist praktisch 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 einzelne Signale die jeweilige Verbindung umfaßt). Soll das Blockschaltbild bereits nähere Einzelheiten veranschaulichen, so deutet man oft die Anzahl der Signale in einer Verbindung durch die Dicke der verbindenden Linien an (vereinfachte Kabelbaumdarstellung). Auch unterscheidet man häufig zwischen rein kombinatorisch und sequentiell wirkenden Funktionsblöcken (Trapezform; Kästchenform). Insbesondere in Funktionsbeschreibungen bzw. 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 4.12 zeigt verschiedene Darstellungselemente von Blockschaltbildern.

Abbildung 4.12 Darstellungselemente von Blockschaltbildern

Hinweis:
Blockschaltbilder vorwiegend *analoger* Schaltungen haben oft eine etwas detailliertere Symbolik. Dies ist in Abbildung 4.13 anhand eines Beispiels dargestellt.

Abbildung 4.13 Blockschaltbild eines "analogen" Gerätes (Rohde & Schwarz)

Praxisbeispiele von Blockschaltbildern und Schaltplänen
Die Abbildungen 4.14 bis 4.19 sollen beispielhaft veranschaulichen, was in der Praxis zu erwarten ist. (Es geht hier nur um einen ersten Eindruck von den praxisüblichen Gepflogenheiten, nicht aber darum, die Schaltungen im einzelnen zu verstehen.)

Abbildung 4.14 Blockschaltbild eines DMA-Controllerschaltkreises (Intel)

Abbildung 4.15 Blockschaltbild eines einfachen Mikroprozessorsystems (Hewlett-Packard)

Abbildung 4.16 Ausschnitt aus dem Schaltplan eines Festplattencontrollers (Intel)

Abbildung 4.17 Schaltplan eines Testers für das 3270-Interface (Hewlett-Packard)

Abbildung 4.18 Innenschaltung eines Addiererschaltkreises (National Semiconductor)

Abbildung 4.19 Schaltplan eines Logikprüfstifts (Hewlett-Packard)

4.2.3. 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. Der Bestückungsplan ist eine zeichnerische (oder auch photographische) Darstellung der Leiterplatte, aus der die Lage jedes einzelnen Bauelementes erkennbar sein muß. Die Abbildungen 4.20 bis 4.22 zeigen einige Beispiele.

Abbildung 4.20 Bestückungsplan für den Logikprüfstift gemäß Abbildung 4.19 (Hewlett-Packard)

Abbildung 4.21 Stückliste für den Logikprüfstift gemäß Abbildung 4.19 (Hewlett-Packard)

Erklärung:
1 - Bezugszeichen (Reference Designator); 2 - Bezeichnung; 3 - Teilnummer (hier: firmenintern); 4 - Anzahl (Quantity).

Abbildung 4.22 Bestückungsplan (Hewlett-Packard)

Abbildung 4.23 Ausschnitt aus einer Stückliste (Keithley)

Erklärung:

1) Bezugszeichen,
2) Bezeichnung,
3) Position der Teile auf Schaltplan (Sch. = Schematic) und Leiterplatte (Pcb. = Printed Circuit Board),
4) Hersteller (Manufacturer; Kurzzeichen),
5) Artikel-Nr,. des Herstellers (Manufacturers Designation),
6) firmeninterneTeilnummer (des Geräteherstellers).

Gelegentlich finden wir in der Dokumentation überblicksmäßige Bestückungspläne, die nur die Lage jener Bauelemente zeigen, auf die es im betreffenden Zusammenhang gerade ankommt (Abbildung 4.24).

Abbildung 4.24 Überblicksmäßiger Bestückungsplan (für Zwecke der Konfigurations-Einstellung) einer PC-Steckkarte (LAN-Adapter)

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 müssen!) 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 Bauelementes. Dann läßt es sich leicht finden, ohne extra einen Bestückungsplan heranziehen oder auf der Leiterplatte herumsuchen zu müssen.

Bezugszeichen der Funktions- bzw. Bauelemente

Bezugszeichen (Component Identifiers, Reference Designators) sind oft herstellerspezifisch (es gibt einfache laufende Nummern oder Buchstaben-Ziffern-Kombinationen, aber gelegentlich auch mnemonische Bezeichnungen).

Die gebräuchlichste Verfahrensweise: man kennzeichnet jedes Element entsprechend seiner Art durch Kennbuchstaben und vergibt für Elemente mit gleichen Kennbuchstaben laufende Nummern. Insbesondere im englischen Sprachraum sind die Kennbuchstaben gemäß Tabelle 4.2 üblich.

Kennbuchstabe	Element
A	Baugruppe (Assembly)
BT	Batterie
C	Kondensator
D, CR	Diode
F	Sicherung (Fuse)
FL	Filter
J	Steckverbinder allgemein (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 4.2 Gebräuchliche Kennbuchstaben für Bau- bzw. Funktionselemente

4.3. 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 wichtige Darstellungsmittel.

4.4.2. Gleichungen und Listen

4.3.1.1. Wahrheitstabellen

Wahrheitstabellen (Truth Tables) werden in Datenblättern und Funktionsbeschreibungen verwendet, um vergleichsweise einfache kombinatorische Schaltungen darzustellen. Der Vorteil: Das Verhalten ist für sämtliche Eingangsbelegungen 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 (es sei denn im Rahmen eines Entwicklungssystems, das riesige Listen verwalten, anzeigen und ausdrucken kann). Neben der üblichen Tabellenform (vgl. Kapitel 3), gibt es auch Wahrheitstabellen in Darstellungsweisen, die an Programmiersprachen angelehnt sind (vorzugsweise zum Beschreiben programmierbarer Logikschaltungen). Abbildung 4.25 zeigt dies am Beispiel des Entwicklungssystems ST-CUPL (SGS-Thomson).

Abbildung 4.25 Wahrheitstabellen-Darstellung im Entwicklungssystem ST-CUPL (SGS-Thomson)

4.3.1.2. Belegungslisten

In Belegungslisten sind die Eingangsbelegungen einer kombinatorischen Schaltung aufgeführt, die eine bestimmte Ausgangsbelegung (0 oder 1) bewirken. Es gibt (vgl. Kapitel 3) binäre und ternäre Belegungslisten. Solche Listen werden in der Dokumentation recht selten verwendet.

Belegungslisten, auch in unvollständiger Form (es sind nicht *alle* Belegungen enthalten, die den jeweiligen Ausgangswert hervorrufen), eignen sich gut, um das Verhalten von Schaltungen zu erfassen. Solche Listen kann man von Hand aufstellen; sie werden aber auch von Logikanalysatoren, Entwicklungssystemen oder Simulationsprogrammen geliefert.

4.3.1.3. Zustands- bzw. Automatentabellen

Zustands- bzw. 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). Zu konkreten Beispielen siehe Heft 04.

4.3.1.4. Boolesche Gleichungen

Boolesche Gleichungen (vgl. Kapitel 3) werden gelegentlich verwendet, um kombinatorische Schaltungen zu beschreiben (das betrifft im besonderen programmierbare Schaltungen).

Hinweise:

1. Boolesche Gleichungen zur Beschreibung programmierbarer Schaltungen (PLDs, GALs usw.) werden oft in einer herstellerspezifischen Symbolik angegeben. (Eine verbreitete Symbolik: "&" = UND, "#" = ODER, "!" = NICHT. Vgl auch Tabelle 3.5. Dort sind jedoch nicht alle Symbole erfaßt, die in der Praxis vorkommen können.)
2. Beim Fehlersuchen ist die Wahrheitstabelle (oder auch die Belegungstabelle) besser als eine Gleichung, denn daraus ist sofort ersehen, ob die Schaltung auf eine meßtechnisch ermittelte Signalbelegung korrekt reagiert oder nicht. In eine Boolesche Gleichung muß man hingegen die gemessenen Werte einsetzen und die Gleichung damit "ausrechnen".

Zustands- bzw. Automatengleichungen
Das sind Boolesche Gleichungen mit zeitabhängigen Variablen. Sie beschreiben, welche Belegung Speicherelemente nach dem jeweils nächsten Takt einnehmen (Folgezustand; siehe auch oben unter "Zustands- bzw. Automatentabellen").

4.3.2. Graphische Darstellungen

4.3.2.1. Zustandsgraphen (State Diagrams)

Abbildung 4.26 veranschaulicht das Prinzip: jeder Zustand wird durch einen *Knoten* (Node) dargestellt, und die Zustandsübergänge werden durch gerichtete Verbindungen zwischen den Knoten (*Kanten;* Edges) angegeben. Dabei ist an jeder Verbindung (Kante) die Bedingung angetragen, die den jeweiligen Zustandswechsel auslöst (solche Bedingungen

werden auch als Ereignisse (Events) bezeichnet). Ist keine Bedingung eingezeichnet, so wird der Übergang in der jeweils nächsten Taktperiode wirksam. Eine Rückführung auf denselben Knoten bedeutet, daß der aktuelle Zustand beibehalten wird, solange kein Ereignis wirksam ist, um ihn zu verlassen.

Abbildung 4.26 Zustandsgraphen (Beispiele)

Hinweis:
Zu Einzelheiten siehe Heft 04 (zum dargestellten Beispiel “Retriggerbare Zeitstufe” auch Heft 07).

Zustandscodierung
Ein Zustandsgraph sagt im allgemeinen nichts darüber aus, wie der jeweilige Zustand codiert ist. Aus "akademischer" Sicht braucht man ceil (ld (n)) Flipflops, um n verschiedene Zustände schaltungstechnisch zu repräsentieren (binäre Zustandscodierung). In der Praxis weicht man aber nicht selten davon ab. So bevorzugt man oft die 1-ausn-Codierung (je Zustand ein Flipflop; One Hot Enable), es gibt aber auch vielfältige Mischformen (Näheres in Heft 04).

Zustandsgraphen werden gern verwendet, um das Verhalten einfacher bis mittelmäßig komplizierter Schaltungen zu dokumentieren. Wirklich exakte Zustandgraphen 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 4.26 gezeigt). Weiterhin bietet es sich gelegentlich an, einen komplexen, unübersichtlichen Zustandsgraphen in mehrere einfachere zu zerlegen (und deren Zusammenwirken anderweitig zu beschreiben).

4.3.2.2. Flußdiagramme (Flowcharts, Programmablaufpläne)

Flußdiagramme werden gelegentlich in Funktionsbeschreibungen verwendet, um Abläufe zu veranschaulichen. Als "exaktes" Beschreibungsmittel sind sie kaum gebräuchlich (sie belegen viel Platz und sind, wenn länger als 2..3 Seiten, kaum noch überschaubar). Abbildung 4.27 gibt einen Überblick über die übliche Symbolik.

Abbildung 4.27 Symbole in Flußdiagrammen (Auswahl)

4.3.3. Aktivitätslisten

Aktivitätslisten dienen dazu, das Zusammenarbeiten mehrerer Einrichtungen zu veranschaulichen, z. B. zwischen einem Prozessor, einer Interfacesteuerung und einem peripheren Gerät. Der grundsätzliche Aufbau (Abbildung 4.28): für jede der beteiligten

Einrichtungen ist eine Spalte vorgesehen. Die Zeitachse verläuft von oben nach unten. (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!) 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. Für das Dokumentieren von Einzelheiten sind solche Listen allerdings weniger geeignet.

Abbildung 4.28 Aktivitätsliste (Beispiel)

4.3.3.2. Impulsdiagramme (Taktdiagramme)

Impulsdiagramme (Pulse Diagrams, Timing Diagrams) geben den zeitlichen Verlauf von Signalen mehr oder weniger symbolisch wieder.

Oszillogrammdarstellung
Das Oszillogramm zeigt den genauen Signalverlauf, wie er in der Schaltung tatsächlich mit einem Oszilloskop gemessen werden kann. Solche Oszillogrammbilder (teils sogar als photographische Wiedergabe tatsächlich gemessener Abläufe) sind seit langem in der Servicedokumentation üblich. So enthalten Serviceschaltpläne von Fernsehgeräten vielfach Oszillogrammbilder besonders markanter Signale. Wenn es darauf ankommt, Einzelheiten auch 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 auch hier 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. Solche Impulsdiagramme kann man auch in der Schaltung meßtechnisch aufnehmen, und zwar mit einem Logikanalysator.

Aktivitätsdarstellung
Die Aktivitätsdarstellung kennzeichnet *aktive* Signale durch eine (üblicherweise dickere) Linie.

Hinweis:
Aus den anderen Darstellungsformen ist der Signalwechsel zwischen High und Low direkt ersichtlich (High ist weiter oben, Low weiter unten). Hingegen ist die Aktivitätsdarstellung oft - wörtlich genommen - so ausgelegt, daß die *aktiven* Belegungen der jeweiligen Signale hervorgehoben werden (ist ein Signal als "aktiv Low" definiert, so erscheint eine entsprechende Linie für die Zeit, in der es mit Low-Pegel belegt ist).

In der beschreibenden Dokumentation finden wir häufig - manchmal auch kombiniert - die linearisierte, die idealisierte oder die Aktivitätsdarstellung. Abbildung 4.29 veranschaulicht die übliche Symbolik in Impulsdiagrammen.

Abbildung 4.29 Symbolische Darstellungen in Impulsdiagrammen

Hinweise:

1. Gute Impulsdiagramme sollten Bezugslinien (z. B. auf Taktflanken) oder ein Raster enthalten, um die Zeitbezüge bequem ablesen zu können. Gelegentlich enthalten solche Diagramme auch Pfeile, die Schaltfolgen und andere Abhängigkeiten kennzeichnen.
2. Oszillogramme, die wir selbst auf dem Bildschirm sehen, können von denen in der Service-Dokumentation abweichen, auch wenn die zu prüfende Hardware funktioniert. Oft sind die Photos mit einem besonders hochwertigen Oszilloskop aufgenommen worden und geben deshalb mehr Einzelheiten wieder als wir mit unserem Gerät beobachten können. Das betrifft sowohl den Frequenzgang als auch die Möglichkeiten des Triggerns. Abweichende Darstellungen sind somit nicht immer Anzeichen für Fehler!
3. 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.
4. Simulationsprogramme: die analoge (auf elektrischer Ebene wirkende) Simulation (die Nachbildung der Schaltung mittels Software im Computer) liefert Impulsdiagramme, die eine gewisse Zwischenstellung zwischen Oszillogramm und linearisierter Darstellung einnehmen (es handelt sich gleichsam um Oszillogrammdarstellungen einer idealisierten, d. h. von den tatsächlichen Störeinflüssen freien Schaltung). Logik-Simulationsprogramme liefern Impulsdiagramme oft in idealisierter Darstellung.
5. Erschließen Sie sich genaue Zeitverhältnisse *nie* allein aus dem Impulsdiagramm (hier sind schematisierte Darstellungen, z. B. in Datenblättern und Service-Unterlagen, gemeint; nicht gemessene Impulsverläufe). Impulsdiagramme sind fast immer (wenn nicht ausdrücklich anders angegeben) *unmaßstäbliche* Darstellungen. Zeitangaben zum Soll-Verhalten man nur aus dem jeweiligen Datenblatt entnehmen!

Anhang 1

Grundbegriffe der Komplexitätstheorie

Die Komplexitätstheorie ist ein Teilgebiet der Mathematik und befaßt sich unter anderem damit, auf einer ganz allgemeinen abstrakten Grundlage zu untersuchen, wieviel Verarbeitungsleistung für bestimmte Algorithmen überhaupt benötigt wird (unabhängig von einem konkreten Programm, vom benutzten Prozessor usw.). Im besonderen ist von Interesse, wie sich die Verarbeitungszeit in Abhängigkeit von der *Problemgröße* entwickelt. Typische Algorithmen betreffen das Lösen von Gleichungssystemen, das Suchen in Datenbanken usw. Es ist klar, daß es hierbei kleinere und größere Probleme gibt (die Problemgröße wird beispielsweise durch die Anzahl der Unbekannten (Gleichungssystem) oder der Datensätze (Datenbank) bestimmt).

Der jeweilige Kennwert der Problemgröße wird üblicherweise mit n bezeichnet. Aussagen zur Komplexität (z. B. zur Rechenzeit oder zum Speicherbedarf) haben dann die allgemeine Form O(f(n)), wobei f(n) den jeweiligen funktionellen Zusammenhang beschreibt. Die Symbolik läßt sich folgendermaßen veranschaulichen: "O" = "Größenordnung von" (Order of).

Je nachdem, wie f(n) aussieht, spricht man von linearer, polynomialer, exponentieller usw. Komplexität:

Lineare Komplexität: O(n)
Beispielsweise bedeutet die Verdoppelung der Problemgröße n, daß sich auch die erforderliche Rechenzeit verdoppelt.

Polynomiale Komplexität: $O(n^p)$
Hierin ist p ein fester Wert. Beispiele: $O(n^2)$ (quadratische Komplexität), $O(n^3)$ usw. Bei quadratischer Abhängigkeit bedeutet beispielsweise das Verdoppeln von n eine 4-fache Rechenzeit ($2^2 = 4$), das Verdreifachen eine 9-fache ($3^2 = 9$). Gilt die Abhängigkeit $O(n^3)$, so steigt der Rechenzeitbedarf mit der dritten Potenz der Problemgröße (Verdoppelung bedeutet 8-fache, Verdreifachung 27-fache Rechenzeit).

Exponentielle Komplexität: $O(p^n)$
Das Verdoppeln von n bedeutet einen Anstieg gemäß p^2, das Verdreifachen einen Anstieg gemäß p^3 usw. (jeweils mit $p > 1$). Beispiel: wir nehmen $p = 10$ an. Ein Verdoppeln der Problemgröße n führt dann auf einen 100-fachen (10^2), ein Verdreifachen von n auf einen 1000-fachen (10^3) Rechenzeitbedarf. Nehmen wir an, für ein bestimmtes Problem benötigen

wir bei gegebener Problemgröße 5 Minuten Rechenzeit. Ein Verdoppeln von n führt zu einem Zeitbedarf von 500 Minuten (ca. 8 h 20'), ein Verdreifachen zu 5000 Minuten (ca. 3½ Tage) usw.

Beachten Sie, daß kein genaues Wachstumsgesetz angegeben wird, sondern eben nur eine Größenordnung bzw. ein allgemeines Prinzip, wie sich die Problemgröße beispielsweise auf die Verarbeitungszeit auswirkt.

NP-Vollständigkeit
Den Ausdruck müssen wir als gegeben hinnehmen. Er bedeutet: der Rechenzeitbedarf steigt im ungünstigsten Fall exponentiell, in günstigeren Fällen polynomial mit der Problemgröße an. Wichtig ist, daß viele Aufgaben (beim Schaltungsentwurf, beim Suchen in Datenbanken, beim Testen von Hardware usw.) grundsätzlich NP-vollständiger Natur sind.

Hinweise:

1. Gegen exponentielle Komplexität ist letzten Endes kein Kraut gewachsen. Natürlich kann man den Zeitbedarf durch Erhöhung der Rechenleistung verringern. Trotzdem wächst der Bedarf schneller als die mögliche Beschleunigung, so daß von einer gewissen Größenordnung an eine rechentechnische Bearbeitung nicht mehr möglich ist (im obigen "exponentiellen" Beispiel: Verdreifachung der Problemgröße würde, damit die Rechenzeit gleich- bleibt (nämlich 5 Minuten), eine 1000-fache Leistungssteigerung erfordern). Die Auswege: (1) Zerlegung in mehrere kleinere Probleme (Parallelisierung - was aber nicht immer gelingt), (2) Verzicht auf exakte Problemlösung (was bedeutet, daß man sich mit einer näherungsweisen Lösung zufriedengibt).

2. Nicht die Begriffe verwechseln: "Komplexität" heißt nicht unbedingt "Kompliziertheit". Es gibt hochkomplexe Probleme, die überhaupt nicht kompliziert aussehen (zum Beispiel das Nachprüfen, ob eine gegebene Variablenbelegung aus Einsen und Nullen eine gegebene Boolesche Gleichung erfüllt oder nicht).

Anhang 2

Funktionskennzeichnungen nach DIN 40900/IEEE 91-1984

Im folgenden werden die Funktionskennzeichnungen anhand der Abbildungen A.1 bis A.3 näher erläutert.

Abbildung A.1 Übersicht über die allgemeinen Funktionskennzeichnungen

Abbildung A.2 Übersicht über die Funktionskennzeichnungen an den Ein- und Ausgängen

Abbildung A.3 Übersicht über die Funktionskennzeichnungen im Innern des Schaltsymbols

Symbolerklärung zu Abbildung A.1:

1) UND-Verknüpfung,
2) ODER-Verknüpfung. Das Zeichen versinnbildlicht, daß wenigstens 1 Eingang aktiv sein muß, damit der Ausgang aktiv wird.
3) Antivalenzverknüpfung (XOR). Es darf nur ein Eingang aktiv sein, damit der Ausgang aktiv wird.
4) logische Gleichheit,
5) gerade Parität. Eine gerade Anzahl von Eingängen muß aktiv sein, damit der Ausgang aktiv wird.
6) ungerade Parität. Eine ungerade Anzahl von Eingängen muß aktiv sein, damit der Ausgang aktiv wird.
7) der eine (einzige) Eingang muß aktiv sein, damit der Ausgang aktiv wird,
8) Puffer/Treiber (ein Funktionselement mit besonders hoher Treibfähigkeit). Die Spitze des Dreiecks zeigt in Signalflußrichtung.
9) Schmitt-Trigger,
10) Codewandler (z. B. Dezimal → BCD, Binär → 7-Segment usw.),
11) Multiplexer (Datenselektor),
12) Demultiplexer,
13) Addierer,
14) Subtrahierer,
15) Übertragsvorausschau (Carry Look Ahead),
16) Multiplizierer,
17) arithmetischer Vergleich (Magnitude Comparator),
18) arithmetisch-logische Einheit (ALU),
19) retriggerbares Monoflop,
20) nicht retriggerbares Monoflop (Single Shot),
21) astabile Schaltung (Impulsgenerator),
22) Impulsgenerator mit synchronem Start,
23) Impulsgenerator mit synchronem Halt (hält nur nach Abgabe eines vollständigen Impulses, schneidet also keine Impulse ab),
24) Schieberegister. m = Bitanzahl,
25) Zähler. m = Bitanzahl, Zählweite = 2^m.
26) Zähler mit Zählweite m,
27) asynchroner Binärzähler mit Zählweite 2^m,
28) Festwertspeicher (ROM),
29) Lese-Schreib-Speicher mit wahlfreiem Zugriff (RAM),
30) Pufferspeicher (Warteschlangenorganisation; FIFO = First-in-First-Out),
31) Funktionselement wird beim Einschalten gelöscht (auf 0 zurückgesetzt),
32) Funktionselement wird beim Einschalten gesetzt (auf 1),
33) hochkomplexes Funktionselement.

Symbolerklärung zu Abbildung A.2:

1) logische Negation am Eingang. Eine logische 0 (1) außen führt innen zu einer logischen 1 (0).
2) logische Negation am Ausgang. Eine logische 0 (1) innen führt außen zu einer logischen 1 (0).
3) Polaritätsanzeige am Eingang. Signal ist LO- aktiv.
4) Polaritätsanzeige am Ausgang. Signal ist LO-aktiv.
5) Polaritätsanzeige am Eingang. Signal ist LO- aktiv; Signalfluß von rechts nach links.
6) Polaritätsanzeige am Ausgang. Signal ist LO-aktiv; Signalfluß von rechts nach links.
7) Anzeige des Signalflusses (eine Richtung),
8) bidirektionaler Signalfluß,
9) dynamisch wirkender Eingang. Schalten auf Flanke "inaktiv → aktiv". Die gezeigten Übergänge bewirken, daß die innere Logik aktiv wird.
10) dynamisch wirkender Eingang. Schalten auf Flanke "aktiv → inaktiv". Die gezeigten Übergänge bewirken, daß die innere Logik aktiv wird.
11) dynamisch wirkender Eingang. Schalten auf HI-LO-Flanke. Der gezeigte Übergang bewirkt, daß die innere Logik aktiv wird.
12) nicht-logische Verbindung,
13) Analogsignal.

Symbolerklärung zu Abbildung A.3:

1) verzögerter Ausgang,
2) Schmitt-Trigger-Eingang.
3) Open-Collector- oder Open-Drain-Ausgang (ohne eingebauten Pull-up-Widerstand),
4) Open-Collector- oder Open-Drain-Ausgang (mit eingebautem Pull-up-Widerstand),
5) Open-Emitter- oder Open-Source-Ausgang (ohne eingebauten Pull-down-Widerstand),
6) Open-Emitter- oder Open-Source-Ausgang (mit eingebautem Pull-down-Widerstand),
7) Tri-state-Ausgang,
8) gepufferter Ausgang (hohe Treibfähigkeit). Die Spitze des Dreiecks zeigt in Signalflußrichtung.
9) Erlaubniseingang (Enable),
10) übliche Eingangsbezeichnungen bei Flipflops,
11) Triggereingang (bewirkt, daß der innere Zustand den jeweils entgegengesetzten Wert annimmt),
12) Dateneingang eines Latches oder Flipflops,
13) Eingänge für Rechts- bzw. Linksverschiebung,
14) Eingänge zum Auf- oder Abwärtszählen,
15) in binärer Zählweise angeordnete Signale (m entspricht der höchsten Zweierpotenz),
16) Laden eines Festwertes (ist der Eingang aktiv, wird das Register mit dem angegebenen Wert geladen),
17) Inhaltsvergleich. Der Ausgang ist aktiv, wenn der Registerinhalt dem angegebenen Wert entspricht.
18) Eingangsgruppe. Kennzeichnet, daß mehrere Anschlüsse für ein und dasselbe logische Eingangssignal genutzt werden.
19) Ausgang, der einen Festwert liefert.