PEARSON Education

PEARSON Studium

Andrew S. Tanenbaum

Computerarchitektur

Strukturen – Konzepte – Grundlagen

5. Auflage

PEARSON Studium

ein Imprint von Pearson Education
München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Die Ebene der digitalen Logik

3

ÜBERBLICK

In der Hierarchie gemäß *Abbildung 1.2* liegt ganz unten die Ebene der digitalen Logik – die reale Computerhardware. In diesem Kapitel untersuchen wir viele Aspekte der digitalen Logik. Das sind die Bausteine, auf denen wir in nachfolgenden Kapiteln mit den höheren Ebenen aufbauen. Auf dieser Ebene befinden wir uns an der Grenze zwischen Informatik und Elektrotechnik. Es handelt sich allerdings um ein eigenständiges Thema, das keine Kenntnisse zu Hardware oder Technik voraussetzt.

Alle Digitalrechner bestehen aus erstaunlich einfachen Grundelementen. Als Erstes sehen wir uns diese Grundelemente an und erläutern eine spezielle zweiwertige (boolesche) Algebra, mit der man diese Elemente analysieren und beschreiben kann. Dann wenden wir uns einigen Grundschaltungen zu, die sich aus Gattern in einfachen Kombinationen aufbauen lassen. Dazu gehören auch arithmetische Schaltungen. Das nächste Thema zeigt, wie man Gatter verknüpfen kann, um Informationen zu speichern, d.h. wie Speicher aufgebaut sind. Die Prozessoren bilden den nächsten Themenkomplex. Insbesondere erfahren Sie, wie Einchipprozessoren mit Speicher- und Peripheriebausteinen kommunizieren. Später in diesem Kapitel werden zahlreiche Beispiele aus der Industrie vorgestellt.

3.1 Gatter und boolesche Algebra

Digitale Schaltungen lassen sich aus einer kleinen Anzahl einfachster Elemente aufbauen, indem man diese in geeigneter Weise miteinander verknüpft. Die folgenden Abschnitte beschreiben diese Grundelemente, zeigen, wie man sie kombiniert, und führen ein leistungsfähiges mathematisches Instrument ein, mit dem sich das Verhalten dieser Schaltungen analysieren lässt.

3.1.1 Gatter

Eine digitale Schaltung kennt nur zwei logische Werte. Normalerweise stellt ein Signal zwischen 0 und 1 Volt den einen Wert (z.B. die binäre 0) und ein Signal zwischen 2 und 5 Volt den anderen Wert (z.B. die binäre 1) dar. Spannungen außerhalb dieser beiden Bereiche sind nicht zulässig. **Gatter** (Gates) sind kleine elektronische Bauelemente, die verschiedene Funktionen dieser zweiwertigen Signale realisieren. Diese Gatter bilden die Hardwarebasis, auf der alle Digitalrechner aufgebaut sind.

Es ginge über den Rahmen dieses Buchs hinaus, die interne Arbeitsweise von Gattern zu beschreiben. Das ist Thema der **Geräteebene** (Device Level), die noch unter der Ebene 0 liegt. Dennoch tauchen wir ganz kurz in diese Sphäre ein und werfen einen Blick auf das Grundkonzept, das nicht weiter kompliziert ist. Jede moderne digitale Logik beruht letztendlich auf der Tatsache, dass sich ein Transistor als sehr schneller, binärer Schalter betreiben lässt. Abbildung 3.1(a) zeigt einen Bipolartransistor in einer einfachen Schaltung. Dieser Transistor hat drei Verbindungen mit der Außenwelt: **Kollektor**, **Basis** und **Emitter**. Liegt die Eingangsspannung U_{ein} unter einem bestimmten kritischen Wert, sperrt der Transistor und verhält sich wie ein unendlich hoher Widerstand. Dadurch nimmt die Ausgangsspannung der Schaltung U_{aus} einen Wert nahe der Betriebsspannung U_{CC} (eine externe geregelte Gleichspannung mit einem für derartige Schaltungen typischen Wert von +5 V) an. Überschreitet U_{ein} den kritischen Wert, leitet der Transistor und wirkt wie ein Draht, wodurch U_{aus} auf Masse (gemäß Konvention 0 V) gezogen wird. Es ist üblich, den niedrigeren Spannungswert als **Low** zu bezeichnen und den höheren Spannungswert als **High**.

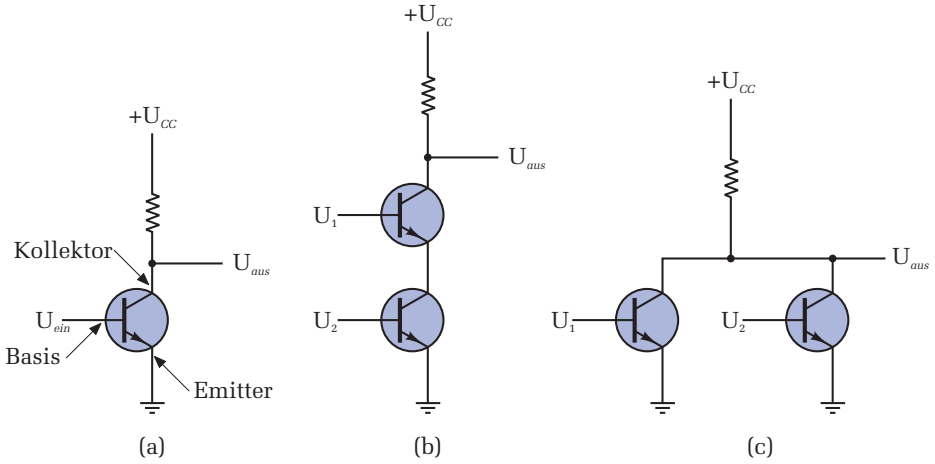


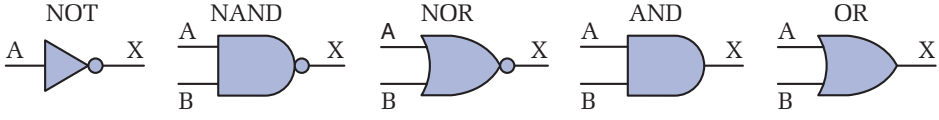
Abbildung 3.1: (a) Transistor als Inverter. (b) Ein NAND-Gatter. (c) Ein NOR-Gatter.

Wichtig hierbei ist: Wenn U_{ein} auf Low liegt, hat U_{aus} den Pegel High und umgekehrt. Folglich stellt diese Schaltung einen Inverter dar, der eine logische 0 in eine logische 1 und eine logische 1 in eine logische 0 umwandelt. Der Widerstand begrenzt den Strom, der durch den Transistor fließt. Der Umschaltvorgang von einem Zustand in den anderen dauert normalerweise nur wenige Nanosekunden.

Abbildung 3.1(b) zeigt zwei in Reihe geschaltete Transistoren. Wenn sowohl U_1 als auch U_2 auf High liegen, leiten beide Transistoren und ziehen U_{aus} auf Low. Liegt einer der beiden Eingänge auf Low, sperrt der entsprechende Transistor und der Ausgang geht auf High. Anders ausgedrückt: U_{aus} ist nur dann Low, wenn sowohl U_1 als auch U_2 auf High liegen.

Die beiden Transistoren in Abbildung 3.1(c) sind nicht in Reihe, sondern parallel geschaltet. Liegt einer der Eingänge in dieser Konfiguration auf High, schaltet der entsprechende Transistor ein und zieht den Ausgang auf Masse. Sind beide Eingänge auf Low, bleibt der Ausgang auf High.

Diese drei Schaltungen (oder äquivalente Schaltungen bei anderen Transistortypen) bilden die drei einfachsten Gatter, die die Funktionen NOT, NAND und NOR realisieren. NOT-Gatter bezeichnet man auch als **Inverter**. Wir benutzen beide Begriffe gleichberechtigt nebeneinander. Wenn wir nun die Konvention anwenden, dass „High“ (nahezu Betriebsspannung, U_{cc}) eine logische 1 und „Low“ (Masse) eine logische 0 darstellt, können wir den Ausgangswert als Funktion der Eingangswerte ausdrücken. In Abbildung 3.2 zeigen die Teile (a) bis (c) die Schaltzeichen, mit denen man diese drei Gatter darstellt, und ihr funktionelles Verhalten. In diesen Abbildungen sind *A* und *B* Eingänge, *X* ist der Ausgang. Jede Zeile gibt das Ausgangssignal für eine bestimmte Kombination von Eingangssignalen an.



A	X
0	1
1	0

(a)

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(b)

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

(c)

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

(d)

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

(e)

Abbildung 3.2: Die Symbole und das funktionelle Verhalten für die fünf Grundgatter

Wenn man das Ausgangssignal des Gatters von Abbildung 3.1(b) in eine Inverterschaltung einspeist, erhält man eine andere Schaltung, die die genaue Umkehrung des NAND-Gatters darstellt – nämlich eine Schaltung, deren Ausgang nur dann 1 ist, wenn beide Eingänge 1 sind. Eine derartige Schaltung nennt man AND-Gatter. Abbildung 3.2(d) zeigt das entsprechende Symbol und die funktionale Beschreibung. Ebenso kann man das NOR-Gatter mit einem Inverter verbinden und erhält eine Schaltung, deren Ausgang 1 ist, wenn mindestens einer der Eingänge 1 ist, die aber 0 liefert, wenn beide Eingänge 0 sind. Abbildung 3.2(e) gibt das Symbol und die funktionale Beschreibung dieses so genannten OR-Gatters an. Die kleinen Kreise in den Symbolen

für den Inverter, das NAND-Gatter und das NOR-Gatter nennt man **Inversionsblasen** (Inversion Bubbles). Sie werden häufig auch in anderen Zusammenhängen benutzt, um ein invertiertes Signal zu bezeichnen.

Die fünf Gatter von Abbildung 3.2 sind die grundlegenden Bausteine der digitalen logischen Ebene. Aus der vorhergehenden Darstellung wird klar, dass NAND- und NOR-Gatter je zwei Transistoren benötigen, während AND- und OR-Gatter jeweils drei brauchen. Aus diesem Grund basieren viele Computer auf NAND- und NOR-Gattern und nicht auf den vertrauteren AND- und OR-Gattern. (Auch wenn die konkrete Schaltungstechnik etwas anders aussieht, sind NAND und NOR dennoch einfacher als AND und OR.) Nebenbei bemerkt, können Gatter auch mehr als zwei – prinzipiell beliebig viele – Eingänge haben. In der Praxis sind aber mehr als acht Eingänge unüblich.

Obwohl die elektronische Realisierung von Gattern zur Geräteebene gehört, gehen wir hier kurz auf die wichtigsten Schaltkreisfamilien oder Herstellungstechnologien ein, da man mit diesen Begriffen häufig zu tun hat. Nach dem als Basis für alle Schaltfunktionen verwendeten Transistor unterscheidet man grundsätzlich zwischen **Bipolartechnik** und **MOS-Technik** (Metal Oxide Semiconductor). Hauptvertreter der bipolaren Technik sind die Familien **TTL** (Transistor-Transistor-Logic) – jahrelang das eigentliche Arbeitspferd der digitalen Elektronik – und **ECL** (Emitter-Coupled Logic) – speziell für hohe Geschwindigkeitsanforderungen. Als Technologie für Computerschaltkreise hat sich größtenteils MOS durchgesetzt.

MOS-Gatter sind langsamer als TTL und ECL, erfordern aber weniger Strom und nehmen viel weniger Platz auf dem Chip ein. Deshalb lassen sie sich in großer Zahl auf engem Raum packen. Die MOS-Technologie gibt es in vielen Varianten, darunter PMOS, NMOS und CMOS. Zwar unterscheiden sich MOS-Transitoren in Bezug auf elektronischen Aufbau und Wirkprinzip von Bipolartransistoren, die prinzipielle Funktionsweise als Schalter ist aber gleich. Die meisten modernen Prozessoren und Speicher basieren auf der CMOS-Technologie mit einer Betriebsspannung von +3,3 V. Damit sind wir schon am Ende des kurzen Ausflugs in die Geräteebene angelangt. Der an dieser Ebene interessierte Leser findet einschlägige Literaturhinweise in *Kapitel 9*.

3.1.2 Boolesche Algebra

Um die Schaltungen zu beschreiben, die sich durch Kombinieren von Gattern aufbauen lassen, ist eine neue Art von Algebra erforderlich – und zwar eine, bei der Variablen und Funktionen nur die Werte 0 und 1 annehmen können. Eine derartige Algebra nennt man **boolesche Algebra** nach ihrem Begründer, dem englischen Mathematiker George Boole (1815–1864). Genau genommen haben wir es hier eigentlich mit einer bestimmten Art von boolescher Algebra zu tun, einer **Schaltalgebra** (Switching Algebra). Da sich aber der Begriff „boolesche Algebra“ in der Bedeutung „Schaltalgebra“ eingebürgert hat, sind wir ebenfalls nicht so streng und verwenden beide Begriffe gleichberechtigt.

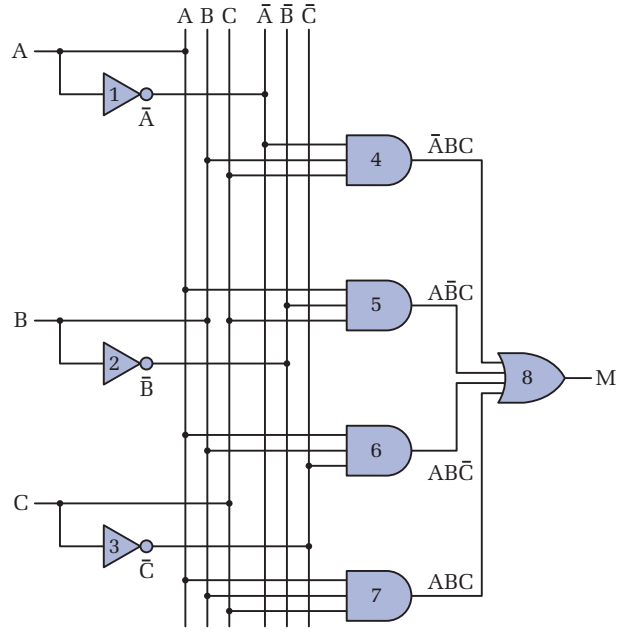
Genauso, wie es Funktionen in der „normalen“ Algebra (d.h. Schulalgebra) gibt, kennt man Funktionen in der booleschen Algebra. Eine boolesche Funktion hat eine oder mehrere Eingabevariablen und führt zu einem Ergebnis, das nur von den Werten dieser Variablen abhängt. Eine einfache Funktion f lässt sich wie folgt definieren: $f(A)$ ist 1, wenn A gleich 0 ist, und $f(A)$ ist 0, wenn A gleich 1 ist. Dies ist die NOT-Funktion aus Abbildung 3.2(a).

Da eine boolesche Funktion mit n Variablen nur 2^n mögliche Kombinationen von Eingabewerten hat, kann man die Funktion vollständig durch eine Tabelle mit 2^n Zeilen beschreiben. Jede Zeile dieser Tabelle enthält den Wert der Funktion für eine andere Kombination von Eingabewerten. Eine solche Tabelle heißt **Wahrheitstabelle** (Truth Table). Die Tabellen in Abbildung 3.2 sind Beispiele für Wahrheitstabellen. Ordnet man die Zeilen einer Wahrheitstabelle in numerischer Reihenfolge (zur Basis 2) an, was bei zwei Variablen die Reihenfolge 00, 01, 10 und 11 ergibt, kann man die Funktion vollständig durch die binäre 2^n-Bit-Zahl beschreiben, die man aus der Ergebnisspalte der Wahrheitstabelle von oben nach unten abliest. Das heißt: NAND ist 1110, NOR ist 1000, AND ist 0001 und OR ist 0111. Selbstverständlich gibt es nur 16 boolesche Funktionen von zwei Variablen, entsprechend den 16 möglichen 4-Bit-Ergebnisketten. Demgegenüber kennt die normale Algebra unendlich viele Funktionen von zwei Variablen und die Ergebnisse lassen sich auch nicht durch eine Tabelle mit den Ausgangswerten für alle möglichen Eingänge beschreiben, weil jede Variable jeden beliebigen Wert einer unendlichen Anzahl möglicher Werte annehmen kann.

Abbildung 3.3(a) zeigt die Wahrheitstabelle für eine boolesche Funktion von drei Variablen: $M = f(A, B, C)$. Diese Funktion ist die logische Mehrheitsfunktion. Das heißt, sie ist 0, wenn eine Mehrheit ihrer Eingänge 0 ist, und 1, wenn eine Mehrheit ihrer Eingänge 1 ist. Obwohl man jede boolesche Funktion vollständig mithilfe einer Wahrheitstabelle festlegen kann, wird diese Notation zunehmend mühsamer, je höher die Anzahl der Variablen ist. Deshalb verwendet man häufig eine andere Notation.

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(a)



(b)

Abbildung 3.3: (a) Die Wahrheitstabelle der Mehrheitsfunktion von drei Variablen; (b) eine Schaltung für (a)

Ausgangspunkt für diese andere Notation ist, dass man jede boolesche Funktion dadurch beschreiben kann, dass man die Kombinationen der Eingangsvariablen angibt, die den Ausgangswert 1 ergeben. Die Funktion von Abbildung 3.3(a) hat vier Kombinationen von Eingangsvariablen, die am Ausgang M zum Wert 1 führen. Der Konvention entsprechend kennzeichnen wir mit einem waagerechten Strich über einer Eingangsvariablen, dass ihr Wert invertiert ist. Fehlt der Überstrich, bedeutet das einen nicht invertierten Wert. Außerdem verwenden wir eine implizite Multiplikation oder einen Punktoperator für die boolesche AND-Funktion und ein Pluszeichen für die boolesche OR-Funktion. Das heißt z.B., dass $A\overline{B}C$ nur dann den Wert 1 annimmt, wenn $A = 1$ und $B = 0$ und $C = 1$ ist. $A\overline{B} + B\overline{C}$ ist nur 1, wenn ($A = 1$ und $B = 0$) oder ($B = 1$ und $C = 0$). Die vier Zeilen in Abbildung 3.3(a), die 1-Bits liefern, lauten: $\overline{A}BC$ $A\overline{B}C$, $AB\overline{C}$ und ABC. Die Funktion M ist wahr (d.h. 1), wenn eine dieser vier Bedingungen wahr ist. Folglich können wir

$$M = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

als kompakte Schreibweise für die Wahrheitstabelle verwenden. Eine Funktion von n Variablen lässt sich somit dadurch beschreiben, dass man eine „Summe“ von höchstens 2^n „Produkttermen“ aus n Variablen angibt. Wir werden gleich sehen, dass diese Form besonders wichtig ist, weil sie direkt zu einer Implementierung der Funktion unter Verwendung von Standardgattern führt.

Wichtig ist die Unterscheidung zwischen einer abstrakten booleschen Funktion und ihrer Implementierung in einer elektronischen Schaltung. Eine boolesche Funktion besteht aus Variablen, z.B. A, B und C, und booleschen Operatoren, z.B. AND, OR und NOT. Man beschreibt eine boolesche Funktion mit einer Wahrheitstabelle oder einer booleschen Funktion, wie

$$F = A\overline{B}C + AB\overline{C}.$$

Eine boolesche Funktion kann man durch eine elektronische Schaltung (oft auf unterschiedliche Weise) implementieren, indem man die Ein- und Ausgangsvariablen durch Signale und die Funktionen wie AND, OR und NOT durch Gatter darstellt. Im Allgemeinen schreiben wir AND, OR und NOT für die booleschen Operatoren und AND, OR und NOT in Bezug auf Gatter, was oftmals aber nicht eindeutig ist.

3.1.3 Implementierung von booleschen Funktionen

Wie bereits erwähnt, führt die Formulierung einer booleschen Funktion als Summe von bis zu 2^n Produkttermen unmittelbar zu einer möglichen Implementierung. Anhand des Beispiels von Abbildung 3.3 wird gezeigt, wie die Funktion implementiert werden kann. In Abbildung 3.3(b) sind die Eingänge A, B und C auf der linken Seite und die Ausgangsfunktion M auf der rechten Seite zu sehen. Da Komplemente (Umkehrfunktionen) der Eingangsvariablen erforderlich sind, führen wir die Eingangssignale auf Inverter, die in der Abbildung mit 1, 2 und 3 bezeichnet sind. Damit die Abbildung übersichtlich bleibt, sind sechs senkrechte Linien eingezeichnet, von denen drei mit den Eingangsvariablen und drei mit deren Komplementen verbunden sind. Diese Linien eignen sich gut als Quellen für die Eingänge zu nachfolgenden Gattern. Beispielsweise benutzen die Gatter 5, 6 und 7 den Wert von A als Eingang. Denken Sie immer daran, dass ein Schaltplan wie in Abbildung 3.3(b) lediglich Symbolcharakter hat. Die konkrete Leiterzugführung sieht in der Regel vollkommen

anders aus und wird von konstruktiven Gegebenheiten bestimmt. In diesem Sinne dienen die sechs senkrechten Hilfslinien nur der Orientierung – in einer praktisch ausgeführten Schaltung sind die Gatter wahrscheinlich ohne diesen Umweg direkt mit A verbunden.

Die Schaltung enthält vier AND-Gatter – je ein Gatter für jeden Term in der Gleichung für M (d.h. ein Gatter für jede Zeile der Wahrheitstabelle, die in der Ergebnisspalte ein 1-Bit hat). Jedes AND-Gatter „berechnet" eine Zeile der Wahrheitstabelle. Schließlich verknüpft ein OR-Gatter alle Produktterme und liefert das Endergebnis.

Die Schaltungsdarstellung von Abbildung 3.3(b) verwendet genormte Schaltzeichen, die Sie auch an anderen Stellen im Buch finden. So sind zwei sich kreuzende Linien nur dann (elektrisch) verbunden, wenn sie im Kreuzungspunkt durch einen Punkt gekennzeichnet sind. Zum Beispiel kreuzt die Ausgangsleitung von Gatter 3 alle sechs vertikalen Linien, sie ist aber nur mit der Linie für das Signal $\overline{C}$ verbunden. Man beachte, dass einige Autoren andere (allerdings nicht genormte) Konventionen verwenden.

Aus dem Beispiel von Abbildung 3.3 wird deutlich, wie Sie eine Schaltung für eine beliebige boolesche Funktion implementieren:

1. Schreiben die Wahrheitstabelle für die Funktion nieder.
2. Sehen Sie Inverter vor, um die komplementären Eingangsignale zu erzeugen.
3. Zeichnen Sie ein AND-Gatter für jeden Term mit einer 1 in der Ergebnisspalte.
4. Verbinden Sie die AND-Gatter mit den entsprechenden Eingängen.
5. Führen Sie die Ausgänge aller AND-Gatter in einem OR-Gatter zusammen.

Obwohl wir hier gezeigt haben, wie Sie eine beliebige boolesche Funktion mit NOT-, AND- und OR-Gattern implementieren, ist es oftmals angebracht oder sogar notwendig, Schaltungen ausschließlich mit einem Gattertyp aufzubauen. Zum Glück ist es nicht kompliziert, die mit dem obigen Algorithmus erzeugten Schaltungen in eine reine NAND- oder eine reine NOR-Form umzuwandeln. Für eine solche Umwandlung benötigen wir nur eine Möglichkeit, NOT, AND und OR mithilfe eines einzigen Gattertyps zu implementieren. Die jeweils obere Zeile in Abbildung 3.4 zeigt, wie sich alle drei Typen ausschließlich mit NAND-Gattern realisieren lassen. In der jeweils unteren Zeile ist das Gleiche für NOR-Gatter dargestellt. (Neben diesen relativ unkomplizierten Schaltungen gibt es selbstverständlich noch andere Möglichkeiten.)

Um eine boolesche Funktion nur mit NAND- oder nur mit NOR-Gattern zu implementieren, kann man die Schaltung zuerst nach dem oben beschriebenen Verfahren mit NOT, AND und OR entwickeln. Dann ersetzt man die Gatter mit mehreren Eingängen durch äquivalente Nachbildungen, die aus Gattern mit zwei Eingängen bestehen. Zum Beispiel lässt sich $A + B + C + D$ mithilfe von drei 2-Eingangs-OR-Gattern als $(A + B) + (C + D)$ darstellen. Schließlich ersetzt man die NOT-, AND- und OR-Gatter durch die in Abbildung 3.4 angegebenen äquivalenten Schaltungen.

Im Sinne einer möglichst geringen Anzahl von Gattern führt dieses Verfahren zwar nicht zu optimalen Schaltungen, zeigt aber, dass es immer eine Lösung gibt. NAND- und NOR-Gatter bezeichnet man als **vollständig**, weil sich jede boolesche Funktion allein mit einem dieser Gattertypen realisieren lässt. Kein anderes Gatter hat diese Eigenschaft. Nicht zuletzt deshalb setzt man sie oftmals als Bausteine für komplexere Schaltungen ein.

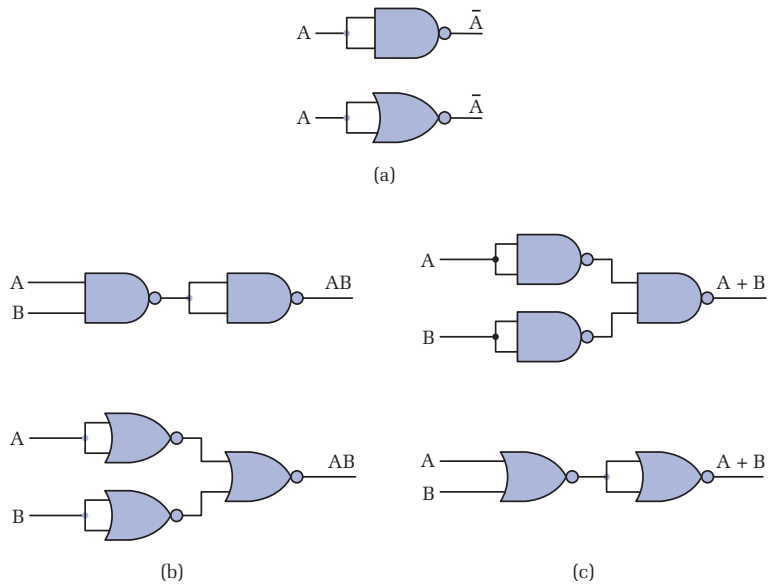


Abbildung 3.4: Nachbildung von (a) NOT-, (b) AND- und (c) OR-Gattern ausschließlich durch NAND- oder NOR-Gatter

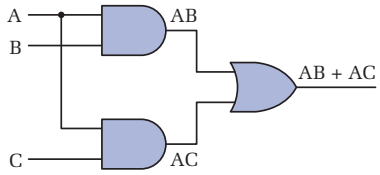
3.1.4 Schaltungsäquivalenz

Entwickler von Schaltungen versuchen in der Regel, die Anzahl der Gatter in ihren Produkten zu minimieren, um Bauteilekosten, Platz auf der Leiterplatte, Stromverbrauch usw. zu reduzieren. Möchte der Entwickler die Komplexität einer Schaltung verringern, muss er eine andere Schaltung finden, die zwar dieselbe Funktion wie das Original berechnet, aber mit weniger Gattern (oder auch einfacheren Gattern, beispielsweise mit zwei statt vier Eingängen) auskommt. Auf der Suche nach äquivalenten Schaltungen erweist sich die boolesche Algebra als nützliches Werkzeug.

Sehen Sie sich als Beispiel dafür die Schaltung und die Wahrheitstabelle für $AB + AC$ in Abbildung 3.5(a) an. Die Regeln der booleschen Algebra haben wir zwar noch nicht behandelt, doch gelten hier auch viele Regeln der normalen Algebra. Insbesondere können wir $AB + AC$ mithilfe des Distributivgesetzes in $A\,(B + C)$ umformen. Abbildung 3.5(b) zeigt die Schaltung und die Wahrheitstabelle für $A\,(B + C)$. Da zwei Funktionen nur dann äquivalent sind, wenn sie denselben Ausgang für alle möglichen Eingänge haben, erkennt man aus den Wahrheitstabellen von Abbildung 3.5, dass $A\,(B + C)$ mit $AB + AC$ äquivalent ist. Trotz dieser Äquivalenz ist die Schaltung von Abbildung 3.5(b) zweifellos besser als die in Abbildung 3.5(a), weil sie weniger Gatter enthält.

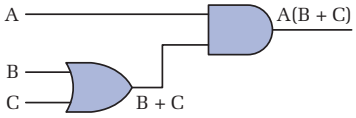
Im Allgemeinen beginnt ein Schaltungsentwickler mit einer booleschen Funktion und wendet dann darauf die Gesetze der booleschen Algebra an, um eine einfachere, aber äquivalente Schaltung zu finden. Aus der endgültigen Funktion lässt sich eine Schaltung aufbauen.

Für diesen Ansatz benötigen wir einige Identitäten aus der booleschen Algebra. Tabelle 3.1 zeigt die wichtigsten Gesetze. Interessant ist, dass jedes Gesetz zwei Formen hat, die **dual** zueinander sind (**Dualitätseigenschaft**). Vertauscht man sowohl AND mit OR als auch 0 mit 1, lässt sich jede Form aus der anderen ableiten.



A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

(a)



A	B	C	A	B + C	A(B + C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

(b)

Abbildung 3.5: Zwei äquivalente Funktionen: (a) $AB + AC$, (b) $A\,(B + C)$

Die Gültigkeit dieser Gesetze kann man mithilfe von Wahrheitstabellen beweisen. Mit Ausnahme der De Morganschen Gleichung, des Absorptionsgesetzes und der AND-Form des Distributivgesetzes sind die Ergebnisse unmittelbar verständlich. Die De Morganschen Gleichungen können auf mehr als zwei Variablen erweitert werden, z.B. $\overline{ABC} = \overline{A} + \overline{B} + \overline{C}$.

Tabelle 3.1

Einige Identitäten der booleschen Algebra

Gesetz	AND-Form	OR-Form
Identität	$1A = A$	$0 + A = A$
Null	$0A = 0$	$1 + A = 1$
Idempotenz	$AA = A$	$A + A = A$
Komplement	$A\overline{A} = 0$	$A + \overline{A} = 1$
Kommutativität	$AB = BA$	$A + B = B + A$
Assoziativität	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributivität	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption	$A(A + B) = A$	$A + AB = A$
De Morgan	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

Die De Morganschen Gleichungen legen eine andere Schreibweise nahe. Abbildung 3.6(a) zeigt die AND-Form mit Negation (durch Inversionsblasen dargestellt) sowohl für Eingang als auch Ausgang. Somit ist ein OR-Gatter mit invertierten Eingängen einem NAND-Gatter äquivalent. Die in Abbildung 3.6(b) dargestellte duale Form der De Morganschen Gleichungen macht deutlich, dass man ein NOR-Gatter als AND-Gatter mit invertierten Eingängen zeichnen kann. Durch Negation beider Formen der De Morganschen Gleichungen gelangen wir zu Abbildung 3.6(c) und (d), den äquivalenten Darstellungen der AND- und OR-Gatter. Entsprechende Symbole gibt es für die Formen der De Morganschen Gleichungen mit mehreren Variablen (z.B. wird aus einem NAND-Gatter mit n Eingängen ein OR-Gatter mit n invertierten Eingängen).

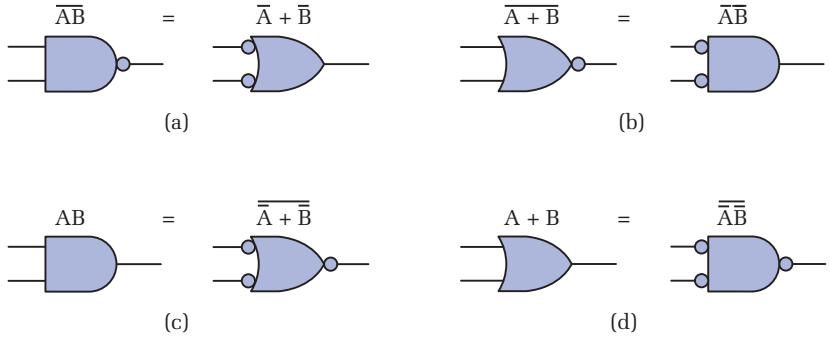


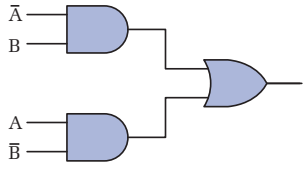
Abbildung 3.6: Alternative Symbole für einige Gatter: (a) NAND, (b) NOR, (c) AND, (d) OR

Mithilfe der Identitäten aus Abbildung 3.6 (die für Gatter mit mehreren Eingängen analog sind) lässt sich die Produktsummendarstellung einer Wahrheitstabelle leicht in die reine NAND- oder die reine NOR-Form umwandeln. Sehen Sie sich als Beispiel die Xor-Funktion von Abbildung 3.7(a) an. Abbildung 3.7(b) zeigt die Standardschaltung gemäß den Produktsummen. Um sie in die NAND-Form umzuwandeln, zeichnen Sie die Linien von den Ausgängen der AND-Gatter zum Eingang des OR-Gatters mit zwei Inversionsblasen neu, wie es Abbildung 3.7(c) zeigt. Schließlich gelangen Sie mithilfe von Abbildung 3.6(a) zu Abbildung 3.7(d). Die Variablen $\overline{A}$ und $\overline{B}$ lassen sich aus A und B mithilfe von NAND- oder NOR-Gattern erzeugen, deren Eingänge jeweils miteinander verbunden werden. Beachten Sie, dass sich Inversionsblasen entlang einer Linie beliebig verschieben lassen, beispielsweise von den Ausgängen der Eingangsgatter in Abbildung 3.7(d) zu den Eingängen des Ausgangsgatters.

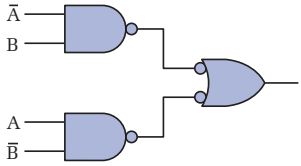
Abschließend demonstrieren wir zum Thema Schaltungsäquivalenz nun das überraschende Ergebnis, dass dasselbe physische Gatter verschiedene Funktionen je nach den benutzten Konventionen berechnen kann. Tabelle 3.2(a) zeigt den Ausgang eines bestimmten Gatters F für verschiedene Eingangskombinationen. Eingänge und Ausgänge sind in Volt dargestellt. Wenn wir 0 V der logischen 0 und 3,3 V bis 5 V der logischen 1 zuordnen, arbeiten wir mit **positiver Logik** und erhalten die Wahrheitstabelle von Tabelle 3.2(b), d.h. die AND-Funktion. Ordnen wir dagegen bei **negativer Logik** 0 V der logischen 1 und 3,3–5 V der logischen 0 zu, ist das Ergebnis die Wahrheitstabelle von Abbildung 3.2(c), die OR-Funktion.

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

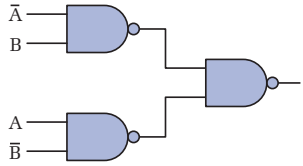
(a)



(b)



(c)



(d)

Abbildung 3.7: (a) Die Wahrheitstabelle für die XOR-Funktion; (b)–(d) drei Schaltungen zu ihrer Realisierung

Tabelle 3.2

(a) Elektrische Eigenschaften eines Bauelements; (b) positive Logik; (c) negative Logik

A	B	F
0 V	0 V	0 V
0 V	5 V	0 V
5 V	0 V	0 V
0 V	0 V	0 V

(a)

A	B	F
0	0	0
0	1	0
1	0	0
0	0	0

(b)

A	B	F
1	1	1
1	0	1
0	1	1
1	1	1

(c)

Dies verdeutlicht, dass die zur Abbildung von Spannungen auf logische Werte gewählte Konvention von großer Bedeutung ist. Soweit nicht anders angegeben, verwenden wir ab jetzt die positive Logik. Somit sind die Begriffe „logische 1“, „wahr“ und „High“ sowie die Begriffe „logische 0“, „falsch“ und „Low“ Synonyme.

3.2 Grundschaltungen der digitalen Logik

Die vorherigen Abschnitte haben gezeigt, wie man Wahrheitstabellen und andere einfache Schaltungen mit einzelnen Gattern implementieren kann. In der Praxis findet man kaum noch Schaltungen, die durchgehend aus einzelnen Gattern aufgebaut sind, während das früher die Regel war. Heutzutage fassen die gängigen Bausteine bereits mehrere Gatterfunktionen zusammen. In den folgenden Abschnitten lernen Sie einige dieser Bausteine näher kennen und erfahren, wie man sie einsetzt und aus Einzelgattern aufbauen kann.

3.2.1 Integrierte Schaltungen

Gatter werden nicht einzeln, sondern als Einheit mit anderen Gattern in Form einer **integrierten Schaltung** (IS) hergestellt und verkauft. Andere gebräuchliche Bezeichnungen sind **Chip** oder **IC** (Integrated Circuit). Auf einem kleinen Siliziumplättchen von ca. 5×5 mm^2 sind mehrere Gatter untergebracht. Kleine integrierte Schaltungen werden gewöhnlich in rechteckige Plastik- oder Keramikgehäuse von 5 bis 15 mm Breite und 20 bis 50 mm Länge montiert. An den Längsseiten befinden sich zwei parallele Reihen von Kontaktstiften oder Pins, die man in Fassungen stecken oder auf Leiterplatten löten kann. Jeder Pin ist mit einem Eingang oder Ausgang eines Gatters auf dem Chip verbunden. Zudem gibt es zwei Anschlüsse für Betriebsspannung und Masse. Schaltkreisgehäuse mit zwei Kontaktreihen bezeichnet man auch als **DIP** (**Dual Inline Package**) oder **DIL** (**Dual In-Line**). Meistens spricht man aber einfach von Chips und verwischt damit den Bedeutungsunterschied zwischen dem eigentlichen Siliziumchip, auf dem die elektrischen Funktionen realisiert sind, und seinem Gehäuse. Gebräuchliche DIL-Gehäuse haben 14, 16, 18, 20, 22, 24, 28, 40, 64 oder 68 Pins. Für größere Chips sind quadratische Gehäuse mit Pins an allen vier Seiten oder an der Unterseite üblich.

Chips lassen sich nach der Anzahl der enthaltenen Gatter klassifizieren. Diese Klassifizierung ist offensichtlich sehr grob, manchmal aber trotzdem nützlich:

- SSI (Small Scale Integration): 1 bis 10 Gatter
- MSI (Medium Scale Integration): 10 bis 100 Gatter
- LSI (Large Scale Integration): 100 bis 100000 Gatter
- VLSI (Very Large Scale Integration): > 100000 Gatter

Diese Klassen weisen unterschiedliche Eigenschaften auf und werden unterschiedlich verwendet.

Ein SSI-Chip enthält normalerweise zwei bis sechs unabhängige Gatter, die sich einzeln wie in den letzten Abschnitten beschrieben verwenden lassen. Abbildung 3.8 stellt schematisch einen gebräuchlichen SSI-Chip mit vier `NAND`-Gattern dar. Jedes Gatter hat zwei Eingänge und einen Ausgang, sodass insgesamt 12 Pins für die vier Gatter erforderlich sind. Außerdem gibt es zwei Anschlüsse für Betriebsspannung (U_{CC}) und Masse (*GND*), die für alle Gatter gemeinsam verwendet werden. Das Gehäuse ist im Allgemeinen mit einer Kerbe oder einer anderen Markierung versehen, um die Seite mit Pin 1 zu kennzeichnen, damit sich die Anschlüsse des Schaltkreises eindeutig zuordnen lassen. Damit Schaltpläne übersichtlich bleiben, verzichtet man in der Regel darauf, die Anschlüsse für Betriebsspannung und Masse sowie ungenutzte Gatter darzustellen.

Solche und ähnliche Chips sind heute bereits für wenige Cent erhältlich. Jeder SSI-Chip umfasst eine Hand voll Gatter, die über bis zu 20 Pins zugänglich sind. In den siebziger Jahren hat man Computer aus großen Mengen solcher Chips gebaut. Mittlerweile ist man in der Lage, einen kompletten Prozessor und einen relativ großen Speicher (Cache) auf einem einzigen Chip unterzubringen.

Für unsere Zwecke betrachten wir alle Gatter in dem Sinne als ideal, dass das Ausgangssignal unmittelbar nach Anlegen der Eingangssignale erscheint. In Wirklichkeit weisen Chips eine endliche **Gatterverzögerung** (Gate Delay) auf, die sowohl die Signalausbreitung (Signal Propagation) durch den Chip als auch die Schaltzeit beinhaltet. Übliche Verzögerungen reichen von 1 bis 10 ns.

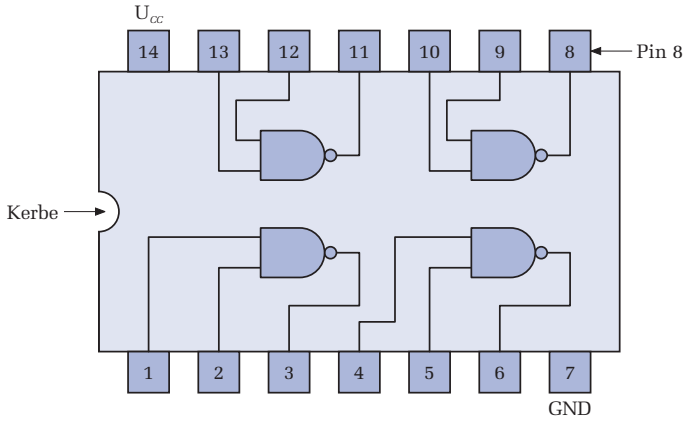


Abbildung 3.8: SSI-Chip mit vier Gattern

Nach dem derzeitigen Stand der Technik könnte man fast 10 Millionen Transistoren auf einen einzigen Chip packen. Da sich jede Schaltung aus NAND-Gattern aufbauen lässt, ist es theoretisch vorstellbar, dass ein Hersteller einen sehr allgemeinen Chip mit 5 Millionen NAND-Gattern fertigt. Leider würde ein solcher Chip 15.000.002 Pins benötigen. Bei einem Standardabstand von 0,1 Zoll (2,54 mm) zwischen den einzelnen Pins wäre der Chip über 19 km lang, was sich auf den Vertrieb des Chips doch sehr nachteilig auswirken würde. Die einzige Möglichkeit, die Technologie zu nutzen, ist also die Entwicklung von Schaltungen mit einem hohen Gatter/Pin-Verhältnis. In den folgenden Abschnitten betrachten wir einfache MSI-Schaltungen, die intern eine Reihe von Gattern zu einer nützlichen Funktion kombinieren und damit nur wenige externe Verbindungen (Pins) verlangen.

3.2.2 Schaltnetze

Viele Anwendungen im Bereich der digitalen Logik erfordern eine Schaltung mit mehreren Ein- und Ausgängen, wobei die Ausgänge eindeutig durch die aktuellen Eingangssignale bestimmt sind. Eine solche Schaltung nennt man **Schaltnetz** (Combinational Circuit). Nicht alle Schaltungen weisen diese Eigenschaft auf. Enthält eine Schaltung beispielsweise Speicherelemente, können die Ausgangssignale sowohl von den gespeicherten Werten als auch den Eingangsvariablen abhängen. Ein typisches Beispiel für ein Schaltnetz ist eine Schaltung, die eine Wahrheitstabelle wie in Abbildung 3.3(a) implementiert. In diesem Abschnitt lernen Sie einige häufig eingesetzte Schaltnetze kennen.

Multiplexer

Auf der Ebene der digitalen Logik ist ein **Multiplexer** eine Schaltung mit 2^n Dateneingängen, einem Datenausgang und n Steuereingängen, die einen der Dateneingänge auswählen. Der gewählte Dateneingang wird zum Ausgang geleitet. Abbildung 3.9 zeigt den Schaltplan eines Multiplexers mit acht Eingängen. Die drei Steuerleitungen A, B und C kodieren eine 3-Bit-Zahl, die festlegt, welche der acht Eingangsleitungen zum OR-Gatter und somit zum Ausgang durchgeschaltet wird. Unabhängig von den Werten

der Steuerleitungen liegen die Ausgänge von sieben AND-Gattern immer auf 0, während das verbleibende achte AND-Gatter je nach dem Signal der ausgewählten Eingangsleitung entweder 0 oder 1 am Ausgang zeigt. Jedes AND-Gatter wird durch eine andere Kombination der Steuereingänge freigegeben. Abbildung 3.9 zeigt die vollständige Multiplexer-Schaltung. Fügt man noch Betriebsspannungs- und Massenanschlüsse hinzu, lässt sich das Ganze in einem 14-poligen DIL-Gehäuse unterbringen.

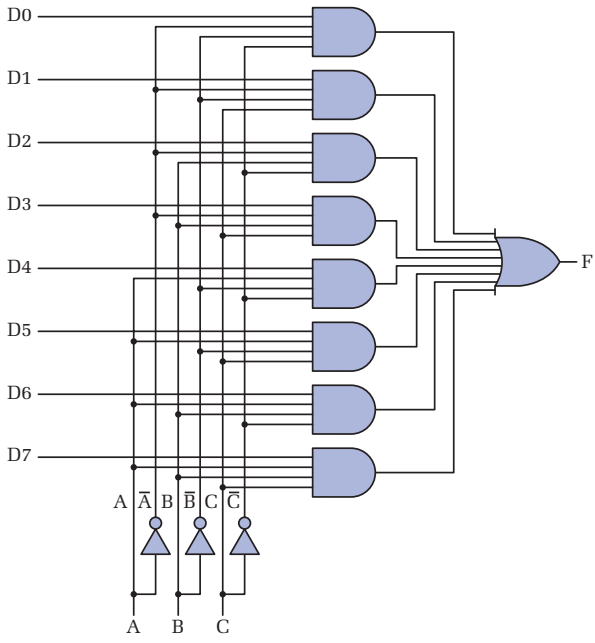


Abbildung 3.9: Multiplexer-Schaltung mit acht Eingängen

Mit einem Multiplexer kann man die Mehrheitsfunktion von Abbildung 3.3(a) implementieren (siehe Abbildung 3.10). Jede Kombination von A, B und C wählt genau eine Dateneingangsleitung aus. Jeder Eingang liegt entweder an U_{CC} (logisch 1) oder Masse (logisch 0). Der Algorithmus für die Belegung der Eingänge ist einfach: Eingang D_i entspricht dem Wert in Zeile i der Wahrheitstabelle. In Abbildung 3.3(a) sind die Ausgangswerte in den Zeilen 0, 1, 2 und 4 gleich 0, sodass die entsprechenden Eingänge auf Masse liegen. Die übrigen Zeilen sind 1 und demnach mit der logischen 1 verbunden. Auf diese Weise kann man eine beliebige Wahrheitstabelle von drei Variablen mit dem Chip von Abbildung 3.10(a) implementieren.

Das Beispiel eben hat gezeigt, wie ein Multiplexer einen von mehreren Eingängen auswählen und eine Wahrheitstabelle implementieren kann. Ein Multiplexer lässt sich auch als Parallel/Serien-Wandler einsetzen. Legt man 8 Datenbits auf die Eingangsleitungen und schaltet die Steuerleitungen nacheinander von 000 bis 111 (binär) weiter, gelangen die Eingangssignale sequenziell auf die Ausgangsleitung. Die Parallel/Serien-Umwandlung findet man beispielsweise in einer Tastatur. Jeder Taste ist eine 7- oder 8-Bit-Zahl zugeordnet, die bei einem Tastendruck seriell über das Tastaturkabel übertragen werden muss.

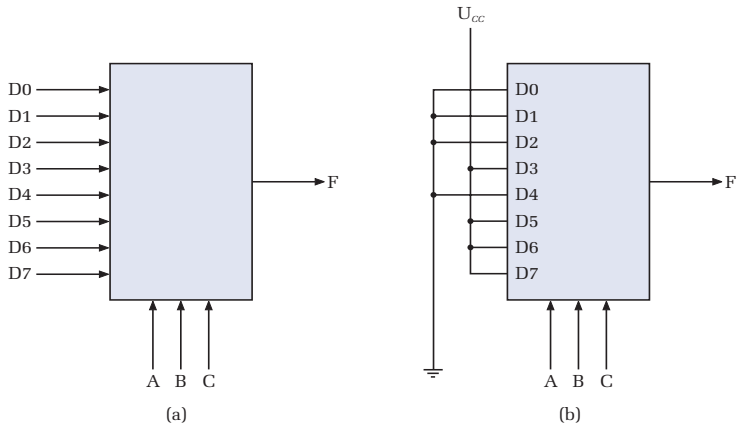


Abbildung 3.10: (a) Ein MSI-Multiplexer. (b) Derselbe Multiplexer wurde für die Berechnung der Mehrheitsfunktion beschaltet.

Das Gegenstück des Multiplexers ist der **Demultiplexer**, der je nach den Werten der n Steuerleitungen ein einzelnes Eingangssignal auf einen von 2^n Ausgängen leitet. Liegt an den Steuerleitungen der Binärwert k an, wird Ausgang k gewählt.

Dekodierer

Als zweites Beispiel sehen wir uns nun eine Schaltung an, die eine n-Bit-Zahl als Eingabe übernimmt und dieser Zahl entsprechend genau eine der 2^n Ausgangsleitungen wählt (d.h. auf 1 setzt). Eine solche Schaltung nennt man **Dekodierer** (Decoder). Abbildung 3.11 zeigt ein Beispiel für $n = 3$.

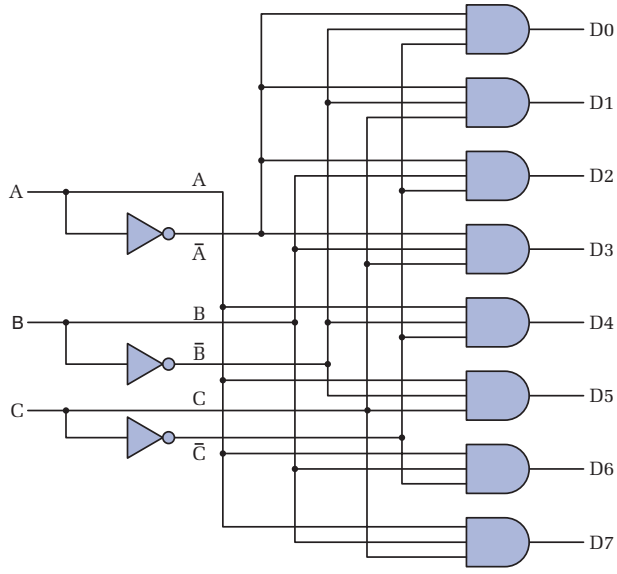


Abbildung 3.11: Schaltung eines 3-zu-8-Dekodierers

Wofür braucht man Dekodierer? Stellen Sie sich als Beispiel einen Speicher vor, der aus acht Chips mit je 1 MB besteht. Chip 0 hat die Adressen 0 bis 1 MB, Chip 1 die Adressen 1 MB bis 2 MB usw. Legt man eine Adresse an den Speicher an, dienen die 3 höherwertigen Bits dazu, einen der acht Chips auszuwählen. Verwendet man die Schaltung nach Abbildung 3.11, entsprechen diese 3 Bits den drei Eingängen *A*, *B* und *C*. Abhängig von den Eingängen hat genau eine der acht Ausgangsleitungen D_0,..., D_7 den Wert 1 und die restlichen liegen auf 0. Jede Ausgangsleitung ist mit den Freigabeeingängen der acht Speicherchips verbunden. Und da jeweils nur eine Ausgangsleitung auf 1 gesetzt wird, ist jeweils nur ein Chip aktiv.

Die Arbeitsweise der Schaltung von Abbildung 3.11 ist leicht nachvollziehbar. Jedes AND-Gatter hat drei Eingänge, von denen der erste entweder A oder $\overline{A}$, der zweite entweder B oder $\overline{B}$ und der dritte entweder C oder $\overline{C}$ ist. Jedes Gatter wird durch eine andere Eingangskombination freigegeben: D_0 durch $\overline{ABC}$, D_1 durch $\overline{AB}C$ usw.

Komparatoren

Eine weitere nützliche Schaltung ist der **Komparator** (Comparator), der zwei Eingangsworte vergleicht. Der einfache Komparator von Abbildung 3.12 übernimmt zwei Eingaben *A* und *B* mit einer Länge von je 4 Bit und liefert das Ergebnis 1, falls die Eingangswerte gleich sind, andernfalls 0. Die Schaltung basiert auf dem XOR-Gatter (Exclusive OR), das bei gleichen Eingangssignalen das Ergebnis 0 und bei ungleichen Eingängen das Ergebnis 1 liefert. Wenn die beiden Eingabeworte gleich sind, müssen alle vier XOR-Gatter 0 ausgeben. Diese vier Signale lassen sich dann mit OR verknüpfen. Ist das Ergebnis 0, sind die Eingabewörter gleich, andernfalls nicht. In unserem Beispiel haben wir ein NOR-Gatter als letzte Stufe vorgesehen, um die Aussage des Tests umzudrehen: 1 bedeutet „gleich“ und 0 bedeutet „ungleich“.

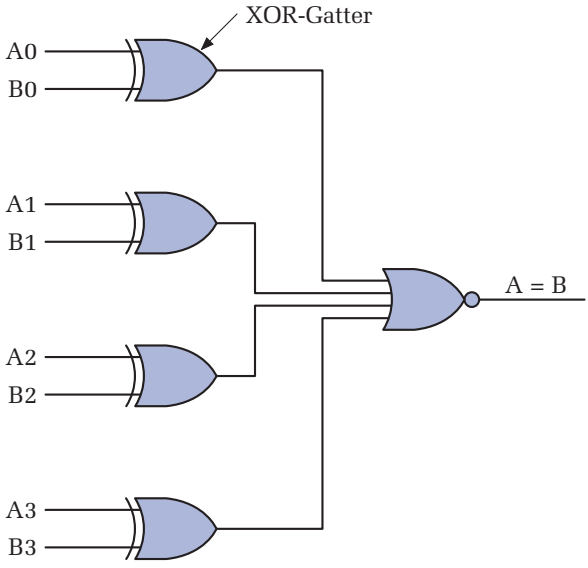


Abbildung 3.12: Ein einfacher 4-Bit-Komparator

Programmierbare logische Anordnung

Wir haben bereits gesehen, dass man beliebige Funktionen (Wahrheitstabellen) dadurch erzeugen kann, dass man Produktterme mit AND-Gattern berechnet und dann die Produkte mit OR verknüpft. Ein sehr allgemeiner Chip zur Summenbildung von Produkten ist die **programmierbare logische Anordnung** (Programmable Logic Array), die man auch als **programmierbares logisches Array** oder einfach mit der Abkürzung **PLA** bezeichnet. Abbildung 3.13 zeigt ein kleines Beispiel. Dieser Chip besitzt Eingangsleitungen für 12 Variablen. Das Komplement jedes Eingangs wird intern erzeugt, sodass sich insgesamt 24 Eingangssignale ergeben. Der Kern der Schaltung ist ein Array von 50 AND-Gattern, für die jeweils eine beliebige Untermenge der 24 Eingangssignale als Eingang möglich ist. Eine vom Benutzer bereitgestellte 24 × 50-Bit-Matrix bestimmt, welches Eingangssignal an welches AND-Gatter gelangt. Jede Eingangsleitung zu den 50 AND-Gattern enthält eine Sicherung. Bei Auslieferung sind alle 1200 Sicherungen intakt. Um die Matrix zu programmieren, brennt der Benutzer ausgewählte Sicherungen durch, indem er am entsprechenden Kreuzungspunkt eine hohe Spannung anlegt.

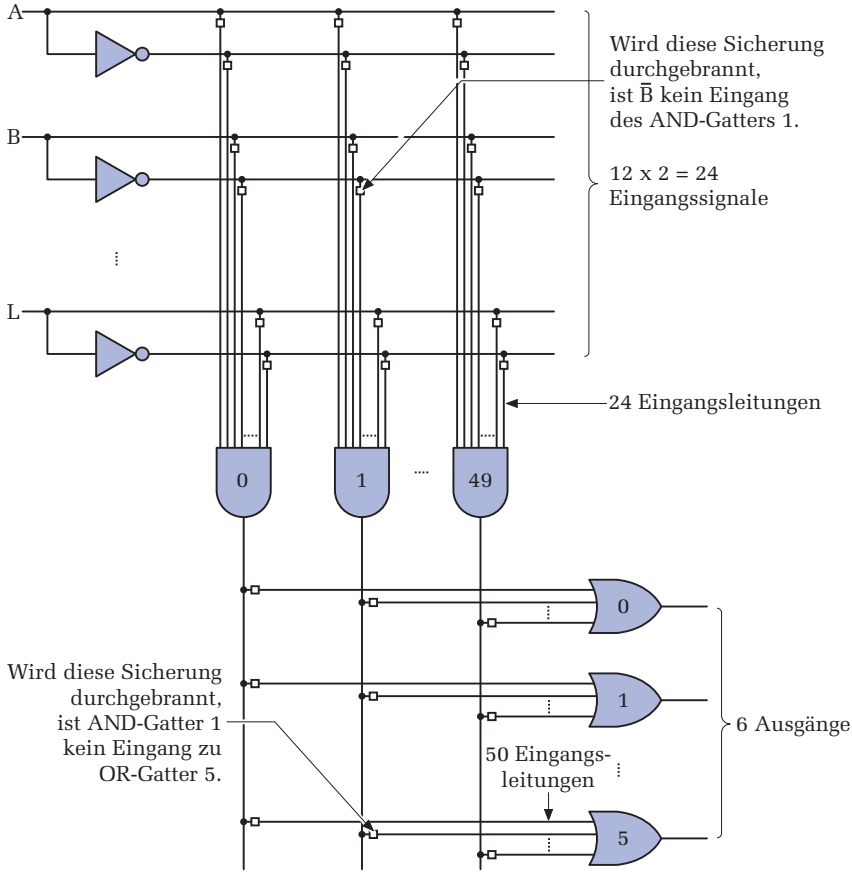


Abbildung 3.13: Programmierbares logisches Array (PLA) mit 12 Ein- und 6 Ausgängen. Die kleinen Quadrate stellen Sicherungen dar, die zur Bestimmung der zu berechnenden Funktion durchgebrannt werden können. Die Sicherungen sind in zwei Matrizen angeordnet: die obere für die AND-Gatter und die untere für die OR-Gatter.

Der Ausgangsteil der Schaltung besteht aus sechs OR-Gattern mit bis zu 50 Eingängen, die den 50 Ausgängen der AND-Gatter entsprechen. Auch hier bestimmt die vom Benutzer bereitgestellte (50 × 6) Matrix, welche der potenziellen Verbindungen tatsächlich existiert. Der Chip hat 12 Eingangspins, 6 Ausgangspins sowie Betriebspannung und Masse, was insgesamt 20 Pins ergibt.

Als Anwendungsbeispiel für ein PLA ziehen wir wieder die Schaltung von Abbildung 3.3(b) heran. Sie hat drei Eingänge, vier AND-Gatter, ein OR-Gatter und drei Inverter. Mit den entsprechenden internen Verbindungen kann unser PLA dieselbe Funktion realisieren. Dazu sind 3 der 12 Eingänge, 4 der 50 AND-Gatter und 1 der 6 OR-Gatter erforderlich. (Die vier AND-Gatter bilden die Verknüpfungen $\overline{A}BC$, $A\overline{B}C$, $AB\overline{C}$ und ABC. Das OR-Gatter übernimmt diese vier Produktterme als Eingabe.) Dasselbe PLA könnte man sogar so verdrahten, dass es gleichzeitig insgesamt vier Funktionen ähnlicher Komplexität berechnet. Für diese einfachen Funktionen ist der Begrenzungsfaktor die Anzahl der Eingabevariablen; für komplexere könnte es die Anzahl der AND- oder OR-Gatter sein.

Obwohl es solche vor Ort programmierbaren PLA noch gibt, zieht man für viele Anwendungen kundenspezifisch hergestellte PLA vor. Diese werden von Großabnehmern entwickelt und vom Hersteller gemäß den Kundenspezifikationen gefertigt. Sie sind billiger als die vor Ort programmierbaren.

Wir können nun die drei unterschiedlichen Implementierungsmöglichkeiten für die Wahrheitstabelle von Abbildung 3.3(a) vergleichen. Mit SSI-Bauelementen brauchen wir vier Chips. Alternativ würde ein MSI-Multiplexer-Chip wie in Abbildung 3.10(b) ausreichen. Schließlich könnten wir ein Viertel eines PLA-Chips verwenden. Natürlich ist die PLA-Lösung effizienter als die beiden anderen, wenn man viele Funktionen benötigt. Für einfache Schaltungen sind die billigeren SSI- und MSI-Chips aber sicher vorzuziehen.

3.2.3 Arithmetische Schaltungen

Jetzt ist es an der Zeit, von den bisher behandelten allgemeinen MSI-Schaltkreisen zu MSI-Schaltnetzen für arithmetische Funktionen überzugehen. Wir beginnen mit einem einfachen 8-Bit-Schieberegister, sehen uns dann einen Addierer an und untersuchen schließlich Rechenwerke, die in jedem Computer eine zentrale Rolle spielen.

Schieberegister

Unsere erste arithmetische MSI-Schaltung ist ein **Schieberegister** (Shifter) mit acht Eingängen und acht Ausgängen (siehe Abbildung 3.14). An den Leitungen D_0, ..., D_7 liegen acht Eingangsbits an. Die Ausgänge stellen die um ein Bit verschobenen Eingänge dar und sind auf den Leitungen S_0, ..., S_7 verfügbar. Die Steuerleitung C bestimmt die Richtung der Verschiebung – 0 für Linksschieben, 1 für Rechtsschieben. Beim Linksschieben läuft in Bit 7 eine 0 ein, beim Rechtsschieben in Bit 0 eine 1.

Als Erstes fallen in dieser Schaltung die Paare der AND-Gatter auf, die für jedes Bit außer dem ersten und letzten vorhanden sind. Bei $C = 1$ ist das rechte AND-Gatter jedes Paars aktiv und das entsprechende Eingangsbit gelangt zum Ausgang. Da das rechte AND-Gatter mit dem OR-Gatter rechts daneben verbunden ist, findet eine Rechtsverschiebung statt. Ist $C = 1$, wird das linke AND-Gatter eines Paars aktiv, wodurch sich eine Linksverschiebung ergibt.

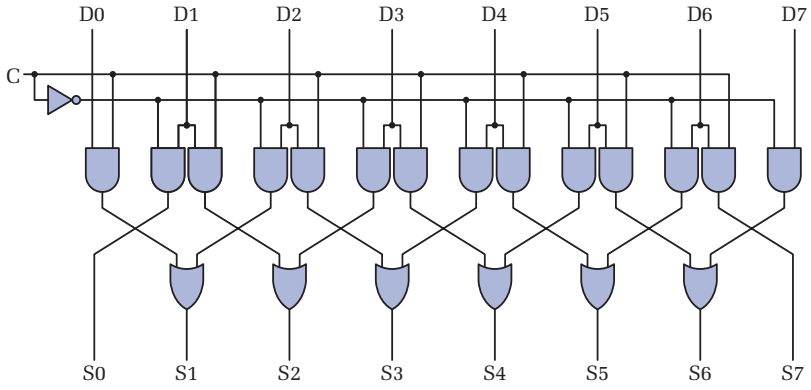


Abbildung 3.14: 1-Bit-Links-/Rechts-Schieberegister

Addierer

Ein Computer, der keine Ganzzahlen addieren kann, ist nahezu undenkbar. Folglich ist eine hardwareseitige Additionsschaltung ein unentbehrlicher Teil eines jeden Prozessors. Die Wahrheitstabelle für eine 1-Bit-Addition ist in Abbildung 3.15(a) dargestellt. Hier gibt es zwei Ausgänge: die Summe (S) der Eingänge A und B sowie den Übertrag ($\ddot{U}$) zur nächsten Stelle (nach links). Abbildung 3.15(b) zeigt eine Schaltung, die sowohl das Summenbit als auch das Übertragsbit berechnet. Diese einfache Schaltung bezeichnet man als **Halbaddierer** (Half Adder).

A	B	S	Ü
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

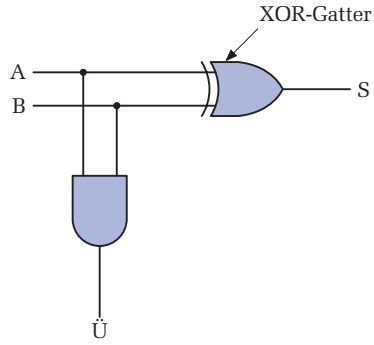
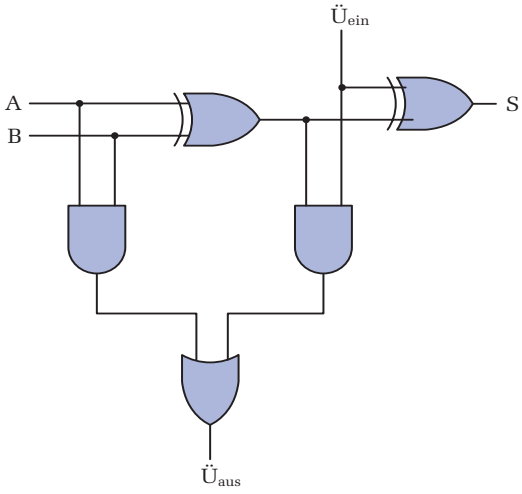


Abbildung 3.15: (a) Wahrheitstabelle für eine 1-Bit-Addition; (b) Schaltung für einen Halbaddierer

Nun ist ein Halbaddierer zwar für die Addition der niederwertigen Bits zweier aus mehreren Bits bestehenden Eingabeworte geeignet, reicht aber nicht für eine Bitstelle in der Wortmitte, weil er den von rechts in die Bitstelle einlaufenden Übertrag nicht behandeln kann. Hier benötigt man den **Volladdierer** (Full Adder) von Abbildung 3.16. Ein Blick auf die Schaltung zeigt, dass ein Volladdierer aus zwei Halbaddierern besteht. Die Ausgangsleitung für die Summe S ist 1, wenn eine ungerade Anzahl der Bits A, B und $\ddot{U}_{ein}$ auf 1 gesetzt ist. Der Übertragsausgang $\ddot{U}_{aus}$ ist 1, wenn entweder A und B gleich 1 sind (linker Eingang zum OR-Gatter) oder genau ein Bit dieser Eingänge 1 und das $\ddot{U}_{ein}$-Bit ebenfalls 1 ist. Zusammen erzeugen die beiden Halbaddierer sowohl die Summen- als auch die Übertragsbits.

A	B	$Ü_{ein}$	S	$Ü_{aus}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)



(b)

Abbildung 3.16: (a) Wahrheitstabelle und (b) Schaltung für einen Volladdierer

Um einen Addierer für beispielsweise zwei 16-Bit-Wörter zu bauen, wiederholt man lediglich die Schaltung von Abbildung 3.16(b) 16-mal. Der Übertragsausgang einer Bitstelle dient als Übertragseintrag der links daneben befindlichen Bitstelle. Den Übertragseingang der ersten (ganz rechts liegenden) Bitstelle legt man fest auf 0. Eine derartige Schaltung bezeichnet man als **Carry-Ripple-Addierer** (von carry = Übertrag und ripple = rieseln, in etwa: Addierer mit Übertragsweiterleitung), weil die Addition von 1 zu 111...111 (binär) im ungünstigsten Fall erst abgeschlossen ist, wenn der Übertrag den gesamten Weg von der äußerst rechten zur äußerst linken Bitstelle zurückgelegt hat. Es gibt auch Addierer, die diese Verzögerung nicht aufweisen und somit schneller sind. Bei größeren Wortlängen sind solche Addierer zu bevorzugen.

Als einfaches Beispiel eines schnelleren Addierers teilen wir einen 32-Bit-Addierer in eine untere und eine obere Hälfte von je 16 Bits auf. Bei Beginn der Addition kann sich der obere Addierer noch nicht an die Arbeit machen, weil er noch nicht weiß, welchen Übertrag er für die ersten 16 Additionen übernehmen muss.

Sehen Sie sich nun folgende Modifikation an: Die Hardware der oberen Hälfte wird nicht nur einmal, sondern zweimal parallel aufgebaut. Somit besteht die Schaltung aus drei 16-Bit-Addierern: einer unteren Hälfte und zwei oberen Hälften *U0* und *U1*, die parallel arbeiten. Als Übertrag für U0 wird eine 0 eingespeist, als Übertrag für U1 eine 1. Jetzt können die beiden oberen Hälften gleichzeitig mit der unteren Hälfte beginnen, wobei aber nur ein Ergebnis der oberen Hälfte korrekt ist. Nach 16 Bitadditionen ist bekannt, wie der Übertrag für die obere Hälfte aussieht, sodass sich nun die entsprechende obere Hälfte aus den beiden verfügbaren Antworten auswählen lässt. Dadurch halbiert sich die Additionszeit. Einen derartigen Addierer nennt man **Carry-Select-Addierer** (in etwa: Übertragsauswahladdierer). Dieser Trick lässt sich wiederholen, um jeden 16-Bit-Addierer aus replizierten 8-Bit-Addierern aufzubauen, usw.

Rechenwerk

Die meisten Computer enthalten nur eine einzige Schaltung, um AND- und OR-Verknüpfungen zu realisieren sowie zwei Maschinenworte zu addieren. Normalerweise besteht eine solche Schaltung für n-Bit-Worte aus n identischen Schaltungen für die einzelnen Bitstellen. Abbildung 3.17 zeigt ein einfaches Beispiel einer solchen Schaltung, die man **Rechenwerk** (Arithmetic Logic Unit) kurz **ALU** nennt. Es kann die vier Funktionen A AND B, A OR B, $\overline{B}$ und $A + B$ berechnen, je nachdem, ob an den Funktionsauswahlleitungen F_0 und F_1 die Werte 00, 01, 10 oder 11 (binär) anliegen. Hier bedeutet $A + B$ die arithmetische Summe von A und B und nicht das boolesche OR.

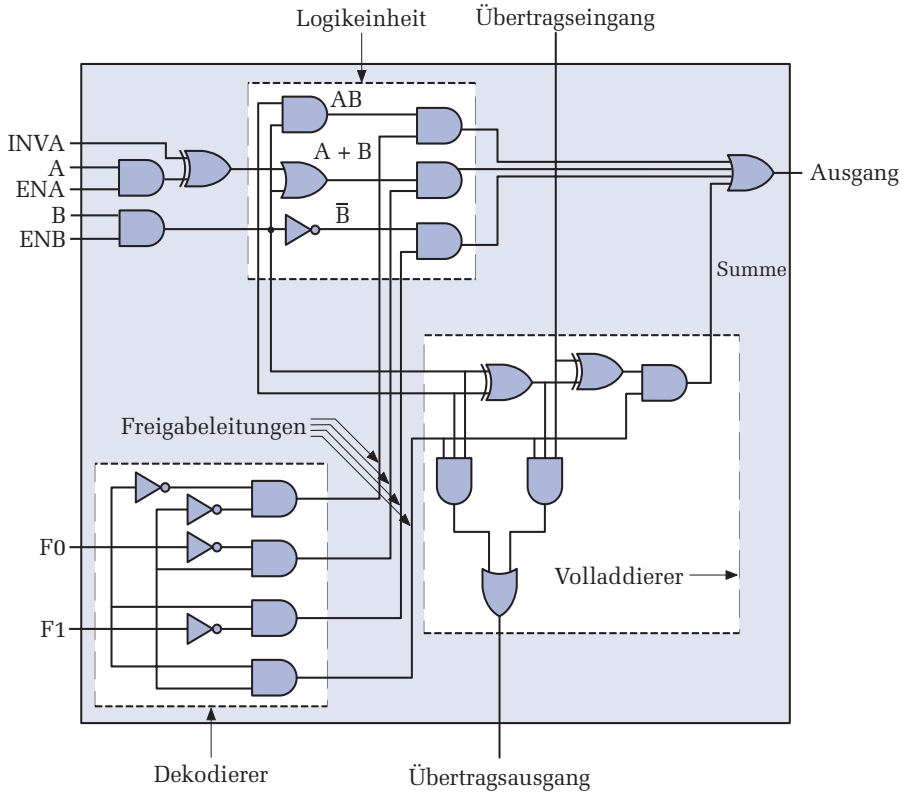


Abbildung 3.17: Eine 1-Bit-ALU

In der unteren linken Ecke unserer ALU befindet sich ein 2-Bit-Dekodierer für die Steuersignale F_0 und F_1, um die Freigabesignale für die vier Operationen zu erzeugen. Je nach den Werten von F_0 und F_1 wird genau eine der vier Freigabeleitungen ausgewählt. Wenn die entsprechende Leitung auf 1 gesetzt ist, gelangt das Ausgangssignal der gewählten Funktion zum abschließenden OR-Gatter und bestimmt damit den Ausgang der Gesamtschaltung.

In der oberen linken Ecke befindet sich die Logik zur Berechnung von A AND B, A OR B und $\overline{B}$; wobei aber nur eines dieser Ergebnisse zum letzten OR-Gatter gelangt. Welcher Ausgang das ist, hängt vom Zustand der Leitungen ab, die vom Dekodierer kommen. Da genau einer der Dekodierer-Ausgänge auf 1 gesetzt ist, wird genau eines der

vier AND-Gatter, deren Ausgänge zum OR-Gatter führen, freigegeben und die Ausgänge der anderen drei AND-Gatter haben unabhängig von *A* und *B* den Wert 0.

Man kann *A* und *B* nicht nur als Eingänge für logische oder arithmetische Operationen verwenden, sondern diese Eingangssignale auch auf 0 ziehen, indem man ENA bzw. ENB deaktiviert (sprich: auf 0 setzt). Außerdem lässt sich $\overline{A}$ erhalten, wenn INVA auf 1 gesetzt wird. *Kapitel 4* erläutert, wie man INVA, ENA und ENB verwendet. Normalerweise setzt man ENA und ENB auf 1, um beide Eingänge freizugeben, und INVA auf 0. In diesem Fall gelangen die Eingangssignale *A* und *B* unverändert zur Logikeinheit.

Der Volladdierer in der rechten unteren Ecke der ALU berechnet die Summe von *A* und *B*. Außerdem behandelt er die Überträge, da man höchstwahrscheinlich mehrere dieser Schaltungen verknüpft, um Operationen mit vollständigen Worten durchführen zu können. Chips wie in Abbildung 3.17 sind als so genannte **Bit-Slices** (Bitscheiben) tatsächlich im Handel erhältlich. Damit ist es möglich, eine ALU mit jeder gewünschten Verarbeitungsbreite aufzubauen. Abbildung 3.18 zeigt eine 8-Bit-ALU, die sich aus acht 1-Bit-ALU-Slices zusammensetzt. Das INC-Signal ist nur für Additionen nützlich. Ist es auf 1 gesetzt, wird das Ergebnis inkrementiert (d.h. 1 zum Ergebnis addiert). Damit lassen sich Summen wie $A + 1$ und $A + B + 1$ berechnen.

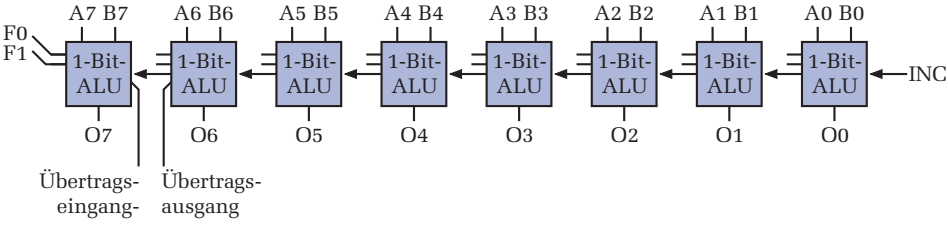


Abbildung 3.18: Acht 1-Bit-ALU-Slices sind verbunden und bilden eine 8-Bit-ALU

3.2.4 Taktgeber

In vielen digitalen Schaltungen ist die Reihenfolge, in der Ereignisse stattfinden, von maßgeblicher Bedeutung. Manchmal muss ein Ereignis einem anderen vorausgehen, in anderen Fällen müssen zwei Ereignisse gleichzeitig eintreten. Damit Entwickler die geforderten Zeitrelationen erreichen können, werden viele digitale Schaltungen mit Taktgebern synchronisiert. Ein **Taktgeber** (Clock) ist in diesem Zusammenhang eine Schaltung, die eine Impulsfolge mit genauer Impulsbreite und genauen Abständen zwischen aufeinander folgenden Impulsen ausgibt. Das Intervall zwischen gleichen Flanken von zwei aufeinander folgenden Impulsen nennt man **Taktzykluszeit** (Clock Cycle Time). Impulsfrequenzen liegen normalerweise zwischen 1 MHz und 500 MHz, was Taktzyklen von 1000 ns bis 2 ns entspricht. Um eine hohe Frequenzstabilität zu erreichen, steuert man Taktgeber normalerweise mit einem Quarzoszillator.

In einem Computer können innerhalb eines einzigen Taktzyklus viele Ereignisse stattfinden. Sollen diese Ereignisse in einer bestimmten Reihenfolge eintreten, muss der Taktzyklus in Teilzyklen aufgeteilt werden. Um eine feinere Auslösung als den Grundtakt bereitzustellen, ist es üblich, den Haupttakt abzugreifen und in eine Schaltung mit einer bekannten Verzögerung einzuspeisen, sodass ein sekundäres Taktsignal entsteht, das gegenüber dem Haupttakt phasenverschoben ist (siehe Abbildung 3.19(a)). Aus dem Taktdiagramm von Abbildung 3.19(b) lassen sich vier Zeitbezugspunkte für diskrete Ereignisse ableiten:

1. Steigende Flanke von C1
2. Fallende Flanke von C1
3. Steigende Flanke von C2
4. Fallende Flanke von C2

Verknüpft man verschiedene Ereignisse mit den verschiedenen Flanken, lässt sich die erforderliche Sequenz umsetzen. Werden innerhalb eines Taktzyklus mehr als vier Zeitbezugspunkte benötigt, kann man weitere sekundäre Leitungen vom Haupttakt abgreifen und mit unterschiedlichen Verzögerungen belegen.

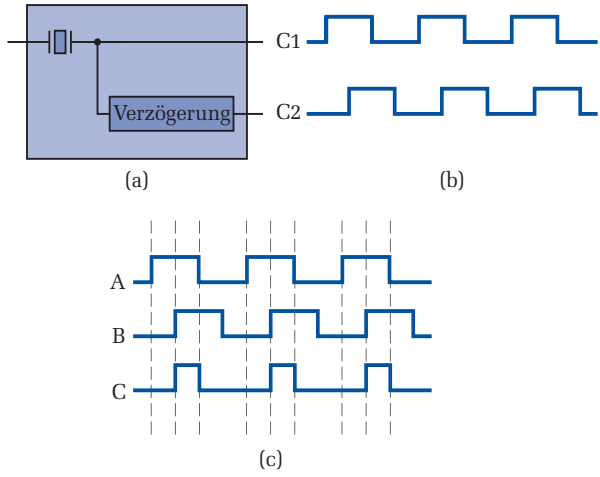


Abbildung 3.19: (a) Taktgeber; (b) Taktdiagramm für den Taktgeber; (c) Erzeugung eines asymmetrischen Takts

Bei manchen Schaltungen ist man mehr an Zeitintervallen als an diskreten Zeitpunkten interessiert. Zum Beispiel darf ein Ereignis jederzeit eintreten, wenn C1 auf High liegt, und nicht nur an der steigenden Flanke von C1. Ein weiteres Ereignis kann nur eintreten, wenn C2 auf High liegt. Benötigt man mehr als zwei Intervalle, kann man mehr Taktleitungen bereitstellen oder die High-Zustände der beiden Taktgeber so einrichten, dass sie sich zeitlich teilweise überlappen. Im zweiten Fall lassen sich vier eindeutige Intervalle unterscheiden: $\overline{C1}$ AND $\overline{C2}$, $\overline{C1}$ AND C2, C1 AND $\overline{C2}$ und C1 AND C2.

Nebenbei bemerkt sind Takte symmetrisch, wobei die High-Zeit gleich der Low-Zeit ist (siehe Abbildung 3.19(b)). Um eine asymmetrische Impulsfolge zu erzeugen, verschiebt man den Basistakt mit einer Verzögerungsschaltung und verknüpft deren Ausgang durch AND mit dem Ursprungssignal. Auf diese Weise erhält man eine Impulsfolge wie C in Abbildung 3.19(c).

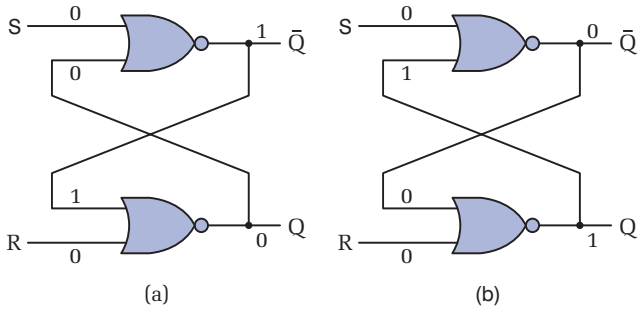
3.3 Speicher

Der Speicher bildet eine wesentliche Komponente jedes Computersystems. Ohne Speicher wären Computer nicht das, was sie sind. Ein Speicher nimmt sowohl die auszuführenden Befehle als auch die Daten auf. Die folgenden Abschnitte untersu-

chen die grundlegenden Komponenten eines Speichersystems. Dabei zeigen wir zunächst auf der Gatterebene, wie Speicherzellen funktionieren und wie sie sich zu großen Speichern kombinieren lassen.

3.3.1 Latches

Um einen 1-Bit-Speicher zu realisieren, brauchen wir eine Schaltung, die sich irgendwie vorherige Eingabewerte „merkt“. Eine solche Schaltung lässt sich aus zwei NOR-Gattern aufbauen, wie es Abbildung 3.20(a) zeigt. Analoge Schaltungen sind auch mit NAND-Gattern möglich. Darauf gehen wir hier nicht weiter ein, weil sie konzeptionell mit den NOR-Versionen identisch sind.



A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Abbildung 3.20: (a) NOR-Latch im Zustand 0; (b) NOR-Latch im Zustand 1; (c) Wahrheitstabelle für NOR

Die Schaltung von Abbildung 3.20(a) nennt man **SR-Latch**. Ein Latch hat zwei Eingänge – S zum Setzen und R zum Zurücksetzen (d.h. Löschen). Außerdem hat es zwei komplementäre Ausgänge Q und $\overline{Q}$ – mehr dazu gleich. Im Gegensatz zu einem Schaltnetz werden die Ausgänge des Latches nicht eindeutig durch die aktuellen Eingangssignale bestimmt.

Um zu sehen, wie das zustande kommt, nehmen wir an, dass S und R gleich 0 sind – das ist der Normal- oder Ruhezustand. Weiter nehmen wir $Q = 0$ an. Da Q in das obere NOR-Gatter zurückgeführt wird, liegen beide Eingänge dieses Gatters auf 0 und der Ausgang $\overline{Q}$ entsprechend auf 1. Die 1 wird in das untere Gatter zurückgeführt, das dann die Eingangsbelegung 1 und 0 hat, woraus sich $Q = 0$ ergibt. Dieser Zustand ist stabil und in Abbildung 3.20(a) dargestellt.

Nun stellen wir uns vor, dass Q nicht 0, sondern 1 ist, während R und S immer noch 0 sind. Die Eingänge des oberen Gatters liegen nun auf 0 und 1. Dementsprechend hat der Ausgang $\overline{Q}$ den Wert 0, der auf das untere Gatter zurückgeführt wird. Dieser Zustand, den Abbildung 3.20(b) zeigt, ist ebenfalls stabil. Dagegen ist ein Zustand, bei dem beide Ausgänge gleich 0 sind, unbestimmt, weil er verlangt, dass an beiden Gattern zwei Nullen als Eingangssignale anliegen. Wäre dies wahr, würde der Ausgang 1 und nicht 0 sein. Ebenso ist es nicht möglich, dass beide Ausgänge gleich 1 sind, weil dies die Eingänge auf 0 und 1 zwingen würde, was 0 und nicht 1 ergibt. Unsere Schlussfolgerung ist einfach: Bei $R = S = 0$ hat das Latch zwei stabile Zustände, die wir nach dem Ausgangswert von Q als 0 und 1 bezeichnen.

Jetzt untersuchen wir die Wirkung der Eingänge auf den Zustand des Latches. Dazu nehmen wir an, dass S auf 1 geht, während $Q = 0$ ist. Die Eingänge des oberen Gatters sind dann 1 und 0, was zum Ausgangszustand $\overline{Q} = 0$ führt. Diese Änderung bewirkt,

dass beide Eingänge des unteren Gatters auf 0 liegen, was zum Ausgangszustand $Q = 1$ führt. Setzt man also S auf 1, wechselt der Zustand von 0 auf 1. Setzt man R auf 1, wenn sich das Latch im Zustand 0 befindet, passiert nichts, weil der Ausgang des unteren NOR-Gatters für die Eingangskombinationen 10 und 11 jeweils 0 ist.

Analog lässt sich leicht erkennen, dass beim Setzen von S auf 1 im Zustand $Q = 1$ nichts passiert, aber das Latch in den Zustand $Q = 0$ übergeht, wenn man R auf 1 setzt. Fassen wir zusammen: Setzt man S kurzzeitig auf 1, nimmt das Latch unabhängig von seinem vorherigen Zustand den Zustand $Q = 1$ an. Und wenn man R kurzzeitig auf 1 setzt, geht das Latch in den Zustand $Q = 0$ über. Die Schaltung „merkt" sich, ob S oder R zuletzt auf 1 gesetzt war. Auf dieser Eigenschaft aufbauend können wir Computerspeicher entwickeln.

Getaktete SR-Latches

Oftmals ist es zweckmäßig, Zustandsänderungen eines Latch nur zu bestimmten festgelegten Zeitpunkten zuzulassen. Dazu modifizieren wir die Grundschaltung etwas, um ein **getaktetes SR-Latch** (Clocked SR Latch) zu erhalten, wie es Abbildung 3.21 zeigt.

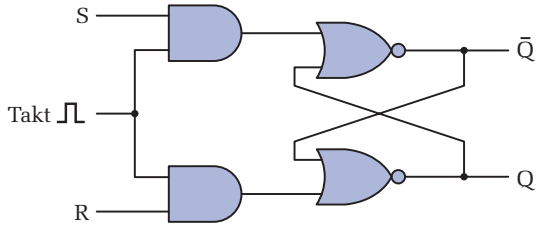


Abbildung 3.21: Getaktetes SR-Latch

Diese Schaltung besitzt einen zusätzlichen Takteingang, der normalerweise auf 0 liegt. In diesem Fall liegen die Ausgänge beider AND-Gatter unabhängig von S und R auf 0 und das Latch behält seinen aktuellen Zustand bei. Geht der Takteingang auf 1, ist die Sperrwirkung der AND-Gatter aufgehoben und das Latch reagiert auf S und R. Die englische Bezeichnung des Takteingangs – „Clock" – hat nichts mit einer Uhr zu tun. Üblich sind auch die Bezeichnungen **Enable** und **Strobe** im Sinne von „aktivieren" oder „freigeben". Das ist lediglich ein anderer Ausdruck dafür, dass der Takteingang den Wert 1 annimmt und die Schaltung auf den Zustand von S und R reagiert.

Bis jetzt haben wir geflissentlich das Problem übergangen, was passiert, wenn sowohl S als auch R auf 1 gesetzt sind. Und das aus gutem Grund: Die Schaltung befindet sich in einem unbestimmten Zustand, wenn R und S schließlich wieder auf 0 zurückgehen. Der einzige konsistente Zustand für $S = R = 1$ ist $Q = \overline{Q} = 0$. Sobald aber beide Eingänge auf 0 zurückkehren, muss das Latch in einen seiner beiden stabilen Zustände springen. Fällt ein Eingang vor dem anderen auf 0 zurück, gewinnt derjenige, der am längsten auf 1 verbleibt. Wenn bei einem NOR-Gatter mindestens ein Eingang auf 1 liegt, wird der Ausgang auf 0 gezogen. Man spricht deshalb bei NOR-Gattern davon, dass der Zustand 1 „durchgreift". Kehren beide Eingänge gleichzeitig auf 0 zurück (was sehr unwahrscheinlich ist), springt das Latch zufällig in einen seiner stabilen Zustände.

Getaktete D-Latches

Das Problem der (durch $S = R = 1$ verursachten) Zweideutigkeit beim SR-Latch lässt sich am besten lösten, wenn man sie von vornherein verhindert. Abbildung 3.22 zeigt eine Latch-Schaltung mit nur einem Eingang D. Da am Eingang des unteren AND-Gatters immer das Komplement des oberen Eingangswertes anliegt, tritt das Problem der gleichzeitig auf 1 gesetzten Eingänge niemals auf. Wenn $D = 1$ ist und der Takteingang auf 1 geht, wird das Latch in den Zustand $Q = 1$ versetzt. Ist $D = 0$ und der Takt geht auf 1, nimmt das Latch den Zustand $Q = 0$ an. Mit anderen Worten: Wenn der Takteingang 1 ist, wird der aktuelle Wert von D abgetastet und im Latch gespeichert. Diese als **getaktetes D-Latch** (Clocked D Latch) bezeichnete Schaltung ist ein echter 1-Bit-Speicher. Der gespeicherte Wert ist stets an Q verfügbar. Um den aktuellen Wert von D in den Speicher zu laden, legt man einen positiven Impuls auf die Taktleitung.

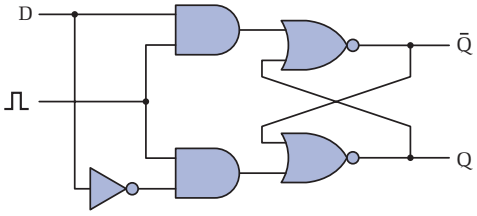


Abbildung 3.22: Getaktetes D-Latch

Diese Schaltung benötigt 11 Transistoren. Speziellere (wenn auch nicht ganz so verständliche) Schaltungen können 1 Bit mit nur 6 Transistoren speichern, sodass man in der Praxis solche Konzepte bevorzugt.

3.3.2 Flipflops

Bei vielen Schaltungen muss man den Wert einer Leitung zu einem bestimmten Zeitpunkt abtasten und speichern. Bei dieser als **Flipflop** bezeichneten Schaltungsvariante findet kein Zustandswechsel statt, wenn das Taktsignal auf 1 liegt, sondern nur beim Übergang des Taktsignals von 0 auf 1 (d.h. mit der steigenden Flanke) oder von 1 auf 0 (mit der fallenden Flanke). Die Länge des Taktimpulses spielt demnach keine Rolle, nur der Potentialübergang muss sich genügend schnell vollziehen.

Um den Unterschied noch einmal deutlich zu machen: Ein Flipflop ist **taktflankengesteuert** (edge triggered), ein Latch **taktpegelgesteuert** (level triggered). In der Literatur werden diese Begriffe häufig durcheinander gebracht. Viele Autoren verwenden „Flipflop“, wenn sie Latch meinen, und umgekehrt.

Ein Flipflop lässt sich nach verschiedenen Konzepten realisieren. Könnte man beispielsweise einen sehr kurzen Impuls auf der steigenden Flanke des Taktsignals erzeugen, ließe sich dieser Impuls als Takt für ein D-Latch verwenden. Diese Möglichkeit gibt es tatsächlich, eine entsprechende Schaltung ist in Abbildung 3.23(a) zu sehen.

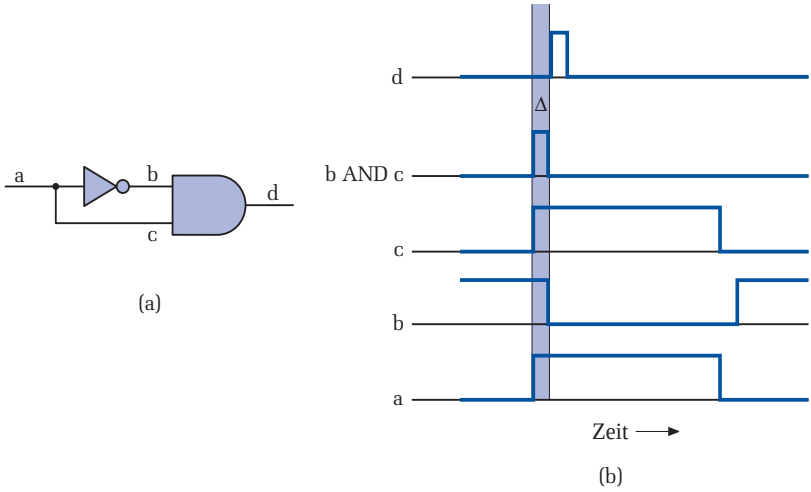


Abbildung 3.23: (a) Impulsgenerator; (b) Zeitlicher Verlauf der Spannungen an vier Punkten der Schaltung

Auf den ersten Blick sieht es so aus, als ob der Ausgang des AND-Gatters immer null wäre, da die AND-Verknüpfung eines Signals mit seinem Komplement immer null ist. Allerdings gibt es hier eine Besonderheit. Der Inverter weist eine kleine, von null verschiedene Signalverzögerung auf. Und genau auf dieser Signalverzögerung beruht die Funktion der Schaltung. Misst man die Spannungen an den vier Punkten, die in Abbildung 3.23(a) eingetragen sind, ergibt sich der in Abbildung 3.23(b) dargestellte zeitliche Verlauf. Das an *a* gemessene Eingangssignal ist ein langer Taktimpuls, wie er in Abbildung 3.23(b) unten dargestellt ist. Darüber ist das Signal am Punkt *b* zu sehen. Beachten Sie, dass es sowohl invertiert als auch leicht verzögert ist. Die Verzögerungszeit liegt im Bereich von wenigen Nanosekunden und ist vom Schaltkreistyp des eingesetzten Inverters abhängig.

Das Signal an *c* ist ebenfalls verzögert, aber nur um die Signallaufzeit, wobei die Ausbreitung der Signale mit Lichtgeschwindigkeit erfolgt. Nimmt man eine physische Entfernung zwischen *a* und *c* von beispielsweise 20 µm an, beträgt die Signalverzögerung rund 0,0001 ns – sicherlich ein tolerierbarer Wert im Vergleich zu der Zeit, die das Signal durch den Inverter benötigt. Das Signal an *c* ist also für alle praktischen Belange so gut wie identisch mit dem Signal an *a*.

Die AND-Verknüpfung der Eingangssignale *b* und *c* durch das AND-Gatter liefert einen kurzen Impuls, wie in Abbildung 3.23(b) dargestellt, wobei die Impulsbreite Δ gleich der Gatterverzögerung des Inverters ist und typischerweise bei 5 ns oder weniger liegt. Am Ausgang des AND-Gatters erscheint genau dieser Impuls verschoben um die Verzögerung des AND-Gatters, wie es der oberste Verlauf in Abbildung 3.23(b) zeigt. Diese Zeitverschiebung bedeutet lediglich, dass das D-Latch erst mit einer festen Verzögerung nach der steigenden Taktflanke aktiviert wird. Auf die Impulsbreite hat die Verzögerung des AND-Gatters keine Auswirkung. Bei einem Speicher mit einer Zykluszeit von 50 ns ist ein Impuls von 5 ns für die Übernahme der Informationen an D noch ausreichend kurz, sodass sich die vollständige Schaltung gemäß Abbildung 3.24 aufbauen lässt. Es sei darauf hingewiesen, dass dieses Flipflop-Design zwar leicht zu verstehen ist, in der Praxis aber wesentlich komplexere Flipflops üblich sind.

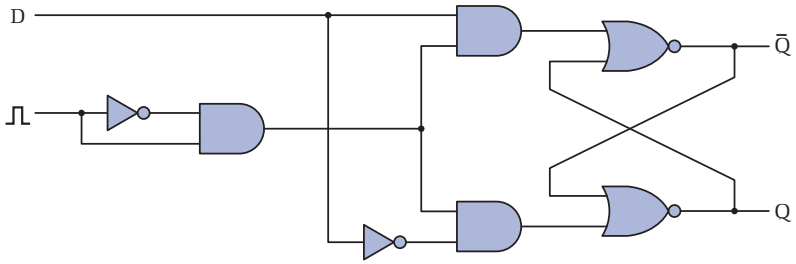


Abbildung 3.24: D-Flipflop-Schaltung

Abbildung 3.25 zeigt die Standardsymbole für Latches und Flipflops. Das Latch in Abbildung 3.25(a) übernimmt den Zustand an *D*, wenn das Taktsignal *CK* gleich 1 ist. Beim Latch in Abbildung 3.25(b) ist der Takt normalerweise 1 und fällt kurzzeitig auf 0, um den Zustand von *D* zu übernehmen. Die Teile (c) und (d) von Abbildung 3.25 zeigen Flipflops statt Latches, was an den Pfeilspitzen an den Takteingängen zu erkennen ist. Das Flipflop nach Abbildung 3.25(c) ändert den Zustand bei der steigenden Flanke des Taktimpulses (Übergang von 0 auf 1), während das Flipflop in Abbildung 3.25(d) den Zustand bei der fallenden Flanke (Übergang von 1 auf 0) ändert. Viele Latches und Flipflops verfügen auch über einen $\overline{Q}$-Ausgang und manche Schaltkreise bieten zwei zusätzliche Stelleingänge mit der Bezeichnung *Set*, *Preset* oder *S* (um den Zustand $Q = 1$ direkt zu setzen) und *Reset*, *Clear* oder *R* (um den Zustand $Q = 0$ direkt zu setzen).

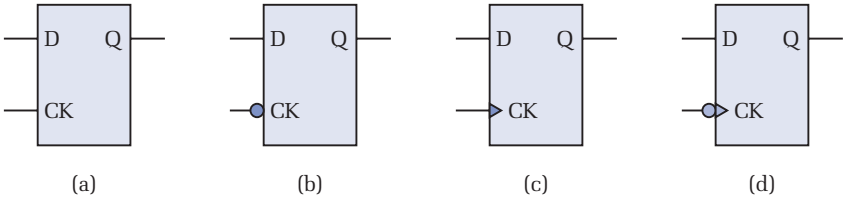
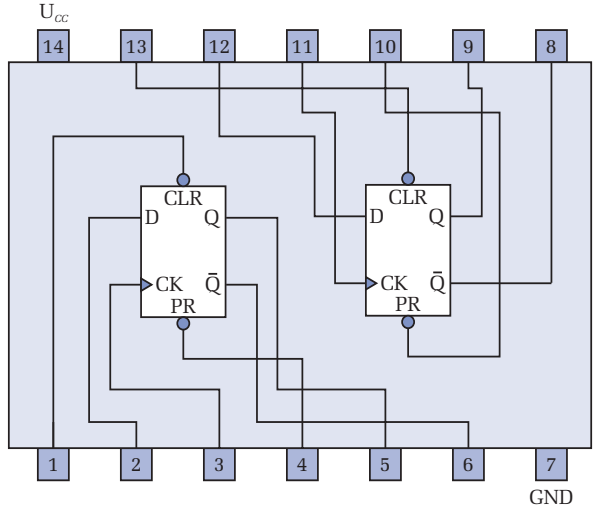


Abbildung 3.25: Standardsymbole für D-Latches und Flipflops

3.3.3 Register

Flipflops gibt es in vielen verschiedenen Konfigurationen. Abbildung 3.26(a) zeigt einen einfachen Schaltkreis, der zwei unabhängige D-Flipflops mit Set- und Reset-Eingängen enthält. Die beiden Flipflops sind zwar im selben 14-poligen Gehäuse untergebracht und nutzen die Anschlüsse für Betriebsspannung und Masse gemeinsam, stehen sonst aber in keinerlei Beziehung zueinander. Dagegen zeigt Abbildung 3.26(b) eine ganz andere Anordnung. Hier fehlen den acht D-Flipflops nicht nur die $\overline{Q}$-Ausgänge und Stelleingänge, es sind auch alle Takteingänge miteinander verbunden und werden von Pin 11 gesteuert. Der Typ der Flipflops selbst entspricht Abbildung 3.25(d), jedoch werden die Inversionsblasen durch den mit Pin 11 verbundenen Inverter kompensiert, sodass die Flipflops die Informationen bei der steigenden Taktflanke übernehmen. Außerdem sind alle acht Löschsignale miteinander verbunden und werden von einem Inverter an Pin 1 gesteuert. Wenn also Pin 1 auf 1 geht, werden alle Flipflops auf den Zustand 0 zurückgesetzt. Wieso verwendet man überhaupt einen Inverter an Pin 11 und negiert den Takt dann noch einmal an jedem *CK*-Eingang? Das hängt mit dem so genannten **Lastfaktor** zusammen: Der Ausgang eines

Schaltkreises kann nur eine bestimmte Anzahl nachgeschalteter Eingänge treiben (mit genügend Strom versorgen). Der Inverter an Pin 11 wird hier eigentlich als Verstärker verwendet. Das Gleiche gilt sinngemäß für Pin 1.



(a)

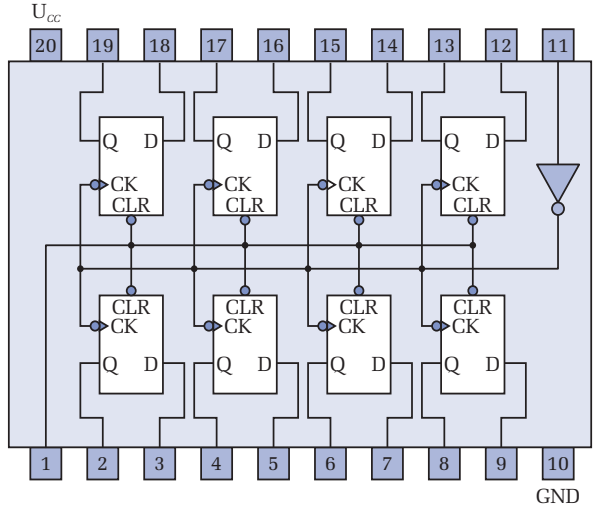


Abbildung 3.26: (a) Schaltkreis mit zwei D-Flipflops (z.B. TTL-Typ 7474); (b) Schaltkreis mit acht D-Flipflops (z.B. TTL-Typ 74LS273)

Die Takt- und Reset-Leitungen des Schaltkreises in Abbildung 3.26(b) schaltet man zusammen, um Pins zu sparen. Allerdings setzt man einen Chip mit dieser Konfiguration auch nicht ein, wenn man acht unabhängige Flipflops braucht, sondern als einzelnes 8-Bit-Register. Alternativ kann man zwei derartige Chips zu einem 16-Bit-Register zusammenschalten, indem man die Pins 1 bzw. 11 der beiden Schaltkreise miteinander verbindet. *Kapitel 4* geht näher auf Register und ihren Einsatz ein.

3.3.4 Speicherorganisation

Vom einfachen 1-Bit-Speicher in Abbildung 3.22 ausgehend sind wir inzwischen beim 8-Bit-Speicher nach Abbildung 3.26(b) angekommen. Um aber große Speicher aufzubauen, ist eine andere Organisation erforderlich, bei der man einzelne Wörter adressieren kann. Eine weit verbreitete Speicherorganisation, die dieses Kriterium erfüllt, ist in Abbildung 3.27 dargestellt. Dieses Beispiel zeigt einen Speicher mit vier 3-Bit-Worten. Jede Operation liest oder schreibt ein volles 3-Bit-Wort. Die gesamte Speicherkapazität von 12 Bits liegt zwar nur wenig über der unseres 8-fach Flipflops, die Schaltung benötigt aber weniger Pins und lässt sich vor allem leicht auf große Speicher erweitern.

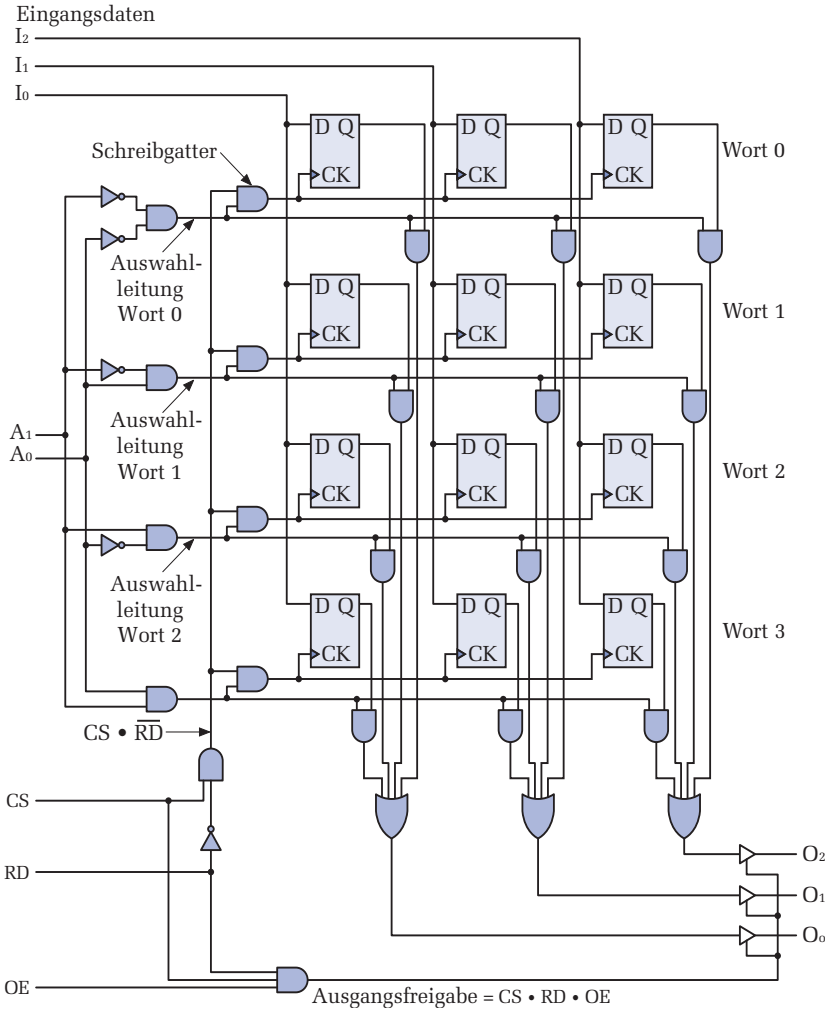


Abbildung 3.27: Logikdiagramm für einen 4 × 3-Speicher. Jede Zeile stellt eines der vier 3-Bit-Worte dar. Eine Lese- oder Schreiboperation bezieht sich immer auf ein ganzes Wort.

Der Speicher in Abbildung 3.27 sieht auf den ersten Blick kompliziert aus, ist es dank seiner regelmäßigen Struktur aber nicht. Er hat acht Eingangs- und drei Ausgangsleitungen. An den drei Eingängen I_0, I_1 und I_2 werden die Daten angelegt, die beiden Eingänge A_0 und A_1 sind für die Adressen vorgesehen und drei weitere Eingänge dienen der Steuerung: CS (Chip Select), um den Chip auszuwählen, RD (Read) zur Umschaltung zwischen Lesen und Schreiben und OE (Output Enable), um die Ausgänge zu aktivieren. Die drei Datenausgänge sind mit O_0, O_1 und O_2 bezeichnet. Prinzipiell lässt sich dieser Speicher einschließlich Betriebsspannung und Masse in einem 14-Pin-Gehäuse unterbringen, während das 8-fach Flipflop 20 Pins benötigt.

Um diesen Speicherchip auszuwählen, setzt die externe Logik CS auf High (logisch 1). An RD muss zum Lesen High und zum Schreiben Low (logisch 0) liegen. Mit den beiden Adressleitungen legt man fest, welches der vier 3-Bit-Worte gelesen bzw. geschrieben werden soll. Eine Leseoperation legt das ausgewählte Wort auf die Datenausgangsleitungen. Die Dateneingangsleitungen werden dabei nicht genutzt. Eine Schreiboperation übernimmt die auf den Dateneingangsleitungen vorhandenen Bitwerte in das ausgewählte Speicherwort. Jetzt werden die Datenausgangsleitungen nicht verwendet.

Betrachten wir nun die Funktionsweise anhand von Abbildung 3.27. Die vier AND-Gatter zur Wortauswahl links vom Speicher bilden einen Dekodierer. Die Eingangsinverter sind so angeordnet, dass jedes Gatter durch eine andere Adresse aktiviert wird (Ausgang liegt auf High). Jedes Gatter steuert eine Wortauswahlleitung und zwar von oben nach unten für die Worte 0, 1, 2 und 3. Ist der Chip für eine Schreiboperation ausgewählt, geht die mit $CS \cdot RD$ beschriftete vertikale Leitung auf High und gibt eines der vier Schreibgatter frei, je nachdem, welche Wortauswahlleitung auf High liegt. Der Ausgang des Schreibgatters steuert alle CK-Signale für das ausgewählte Wort, sodass die Eingangsdaten für dieses Wort in die Flipflops geladen werden. Eine Schreiboperation findet nur statt, wenn CS auf High und RD auf Low liegen. In diesem Fall wird ausschließlich das durch A_0 und A_1 ausgewählte Wort geschrieben – die übrigen Worte bleiben unverändert.

Der Lesevorgang ist dem Schreibvorgang ähnlich. Die Adressdekodierung erfolgt genau wie beim Schreiben. Allerdings liegt jetzt die Leitung $CS \cdot RD$ auf Low, sodass alle Schreibgatter gesperrt sind und keine Flipflops geändert werden. Stattdessen aktiviert die gewählte Wortauswahlleitung die mit den Q-Bits des ausgewählten Wortes verbundenen AND-Gatter. Somit gelangen die Ausgangsdaten des ausgewählten Wortes zu den 4-Eingangs-OR-Gattern im unteren Teil der Abbildung, während die anderen drei Worte jeweils Nullen als Ausgabe liefern. Folglich sind die Ausgänge der OR-Gatter mit dem gespeicherten Wert im ausgewählten Wort identisch. Die drei nicht gewählten Wörter tragen nichts zur Ausgabe bei.

In der Schaltung hätte man die Ausgänge der drei OR-Gatter auch direkt mit den Datenausgangsleitungen verbinden können. Allerdings führt das manchmal zu Problemen. Insbesondere haben wir darauf hingewiesen, dass die Dateneingangs- und Datenausgangsleitungen verschieden sind, während man in realen Speicherchips dieselben Leitungen verwendet. Wenn man dann die OR-Gatter direkt mit den Datenausgangsleitungen verbindet, versucht der Chip, selbst bei Schreibvorgängen Daten auszugeben. Das heißt, die Leitungen werden auf einen bestimmten Wert gezogen, was zu Konflikten mit den Eingabedaten führt. Aus diesem Grund ist es wünschenswert, die OR-Gatter bei Leseoperationen mit den Datenausgangsleitungen zu verbinden, bei Schreiboperationen aber vollständig davon zu trennen. Wir benötigen also einen elektronischen Schalter, der eine Verbindung in wenigen Nanosekunden aufbauen und trennen kann.

Zum Glück gibt es einen solchen Schalter. Abbildung 3.28(a) zeigt einen so genannten **nichtinvertierenden Puffer** (Noninverting Buffer). Er hat einen Dateneingang, einen Datenausgang und einen Steuereingang. Ist der Steuereingang High, agiert der Puffer wie ein Draht (siehe Abbildung 3.28(b)). Liegt der Steuereingang auf Low, verhält sich der Puffer wie ein offener Schalter (siehe Abbildung 3.28(c)) – so als ob jemand den Datenausgang mit einem Seitenschneider vom Rest der Schaltung getrennt hätte. Im Unterschied zur Lösung mit dem Seitenschneider lässt sich die Verbindung in wenigen Nanosekunden wiederherstellen, indem man einfach das Steuersignal auf High setzt.

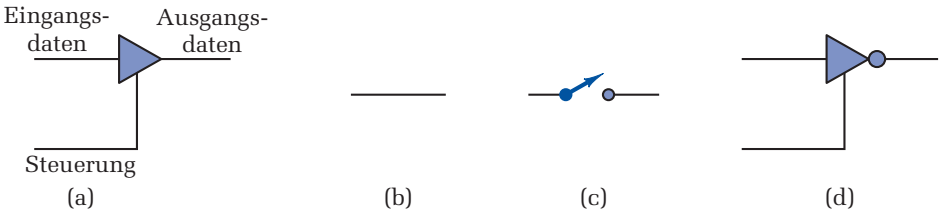


Abbildung 3.28: (a) Nichtinvertierender Puffer; (b) Wirkung von (a), wenn der Steuereingang auf High liegt; (c) Wirkung von (a), wenn der Steuereingang auf Low liegt; (d) invertierender Puffer

Abbildung 3.28(d) zeigt einen **invertierenden Puffer** (Inverting Buffer), der sich wie ein normaler Inverter verhält, wenn der Steuereingang auf High liegt, und den Ausgang von der Schaltung trennt, wenn der Steuereingang Low ist. Beide Puffer sind **Tristate-Bauelemente** (Schaltung mit drei Zuständen), weil an ihren Ausgängen 0, 1 oder nichts davon (offener Schalter) anliegen kann. Puffer verstärken außerdem Signale, sodass sie viele Eingänge gleichzeitig steuern können. Manchmal setzt man sie aus diesem Grund in einer Schaltung ein, selbst wenn man ihre Schaltereigenschaften nicht benötigt.

Wenden wir uns wieder der Speicherschaltung zu. Inzwischen dürfte klar sein, welchen Zweck die drei nichtinvertierenden Puffer in den Datenausgangsleitungen erfüllen. Liegen `CS`, `RD` und `OE` auf High, ist auch das Ausgangsfreigabesignal High, das die Puffer aktiviert und ein Wort auf die Ausgangsleitungen legt. Befindet sich eines der Signale `CS`, `RD` oder `OE` auf Low, werden die Datenausgänge vom Rest der Schaltung getrennt.

3.3.5 Speicherchips

Der Speicher nach Abbildung 3.27 bietet den Vorteil, dass er sich mühelos für größere Kapazitäten erweitern lässt. In der aktuellen Darstellung hat der Speicher eine Kapazität von 4×3 Bits, d.h. vier Wörter zu je 3 Bits. Um ihn auf eine Kapazität von 4×8 zu erweitern, müssen wir lediglich fünf weitere Spalten mit je vier Flipflops und fünf Ausgangsleitungen hinzufügen. Wollen wir von 4×3 auf 8×3 aufrüsten, sind vier zusätzliche Zeilen mit je drei Flipflops sowie eine Adressleitung A_2 erforderlich. Bei dieser Struktur ist es am effizientesten, wenn die Anzahl der Worte im Speicher eine Zweierpotenz ist, während die Anzahl der Bits je Wort beliebig sein kann.

Da die Technologie der integrierten Schaltungen dafür prädestiniert ist, Chips herzustellen, deren interne Struktur ein wiederkehrendes zweidimensionales Muster aufweist, sind Speicherchips ideale Kandidaten. Die sich ständig verbessernde Technik erlaubt es, immer mehr Bits auf einen einzelnen Chip zu packen, wobei sich die Anzahl der Bits entsprechend dem Mooreschen Gesetz etwa alle 18 Monate verdoppelt. Größere Chips machen die kleineren nicht automatisch zum alten Eisen, weil

hinsichtlich Kapazität, Geschwindigkeit, Leistung, Preis und Anschlussgestaltung viele Kompromisse notwendig sind. Bezogen auf ein Bit sind die neuesten Chips im Vergleich zu den älteren und kleineren in der Regel wesentlich teurer.

Für jede bestimmte Speichergröße gibt es verschiedene Möglichkeiten, den Chip zu organisieren. Abbildung 3.29 zeigt zwei Varianten für einen älteren Speicherchip der Größe 4 Mbit: 512 Kbit × 8 und 4096 Kbit × 1. (Übrigens gibt man Speichergrößen normalerweise in Bits und nicht in Bytes an, sodass wir uns hier auch an diese Konvention halten.) In Abbildung 3.29(a) sind 19 Adressleitungen erforderlich, um eines der 2^{19} Byte zu adressieren. Zum Laden oder Speichern des ausgewählten Bytes braucht man acht Datenleitungen.

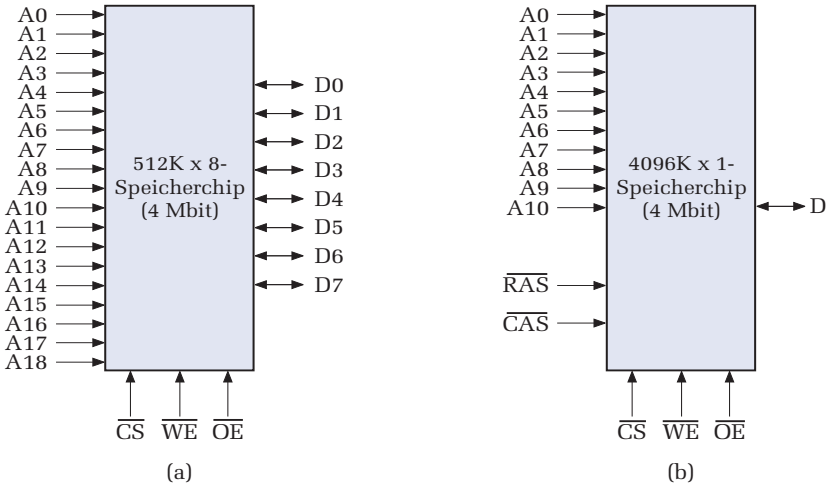


Abbildung 3.29: Zwei Varianten für die Organisation eines 4-Mbit-Speicherchips

An dieser Stelle ist eine Anmerkung zur Terminologie angebracht: An manchen Pins wird eine Aktion durch hohe Spannung bewirkt, an anderen Pins durch niedrige Spannung. Um Verwechslungen zu vermeiden, sprechen wir im weiteren Verlauf davon, dass ein Signal **aktiviert** (statt „auf High oder Low gesetzt wird“). Das bedeutet, dass das Signal gesetzt wird, um eine bestimmte Aktion zu veranlassen. Bei einigen Pins bedeutet Aktivieren, dass der Pin auf High, bei anderen, dass der Pin auf Low gesetzt wird. Pins, die bei Low aktiviert sind, erhalten Signalbezeichnungen mit einem Überstrich. Somit wird ein Signal namens CS bei High aktiviert, während ein Signal namens $\overline{\text{CS}}$ bei Low aktiviert wird. Das Gegenteil von aktiviert ist **deaktiviert**. Wenn nichts Besonderes passiert, werden Pins deaktiviert.

Kehren wir nun wieder zu unserem Speicherchip zurück. Da ein Computer normalerweise viele Speicherchips besitzt, ist ein Signal erforderlich, um den momentan benötigten Chip auszuwählen, damit er und kein anderer reagiert. Für diesen Zweck ist das Signal $\overline{\text{CS}}$ (Chip Select) vorgesehen. Es wird aktiviert, um den Chip freizugeben. Außerdem muss man zwischen Lese- und Schreiboperationen unterscheiden können. Das Signal $\overline{\text{WE}}$ (Write Enable) zeigt an, dass Daten geschrieben und nicht gelesen werden. Schließlich wird das Signal $\overline{\text{OE}}$ (Output Enable) aktiviert, um die Ausgangssignale freizugeben. Ist das Signal deaktiviert, wird der Chip-Ausgang von der Schaltung getrennt.

Der Chip in Abbildung 3.29(b) verwendet ein anderes Adressierungsschema. Intern ist dieser Chip als 2048 × 2048-Matrix von 1-Bit-Zellen organisiert, was 4 Mbit ergibt. Um den Chip zu adressieren, wählt man zuerst eine Zeile aus, indem man ihre 11-Bit-Zahl an die Adresspins legt. Dann wird das Signal $\overline{\text{RAS}}$ (Row Address Strobe) aktiviert. Anschließend wird eine Spaltennummer an die Adresspins gelegt und $\overline{\text{CAS}}$ (Column Address Strobe) aktiviert. Der Chip reagiert, indem er ein Datenbit übernimmt oder ausgibt.

Große Speicherchips werden oft als $n \times n$-Matrizen aufgebaut, die durch Zeile und Spalte adressiert werden. Diese Organisation verringert die erforderliche Anzahl der Pins, verlangsamt aber auch die Adressierung des Chips, da zwei Adressierungszyklen erforderlich sind – einer für die Zeile und einer für die Spalte. Um den Geschwindigkeitsverlust durch dieses Design teilweise wieder wettzumachen, kann man an einige Speicherchips eine Zeilenadresse ausgeben und dann durch eine Sequenz von Spaltenadressen nacheinander auf Bits zugreifen, die in einer Zeile aufeinander folgen.

Vor Jahren hat man die größten Speicherchips oftmals gemäß Abbildung 3.29(b) organisiert. Als die Speicherworte von 8 auf 32 Bits und darüber zugenommen haben, wurden Chips mit einer Breite von 1 Bit unhandlich. Um einen Speicher mit einem 32-Bit-Wort aus 4096 Kbit × 1 großen Chips aufzubauen, sind 32 parallele Chips erforderlich. Diese 32 Chips haben eine Gesamtkapazität von mindestens 16 MB, während bei Verwendung von 512 Kbit × 8-Chips nur vier parallele Chips nötig sind und kleinere Speicher bis herab zu 2 MB möglich sind. Um die hohe Anzahl von 32 Chips für einen Speicher zu vermeiden, sind die meisten Chiphersteller heute zu Chipfamilien mit Breiten von 1, 4, 8 und 16 Bits übergegangen. Natürlich sieht die Lage bei 64-Bit-Worten noch ungünstiger aus.

Abbildung 3.30 gibt zwei Beispiele für moderne 512-Mbit-Chips an. Diese Chips haben vier interne Speicherbänke von jeweils 128 Mbit. Es sind also zwei Bankauswahlleitungen erforderlich, um eine Bank zu wählen. Das Konzept in Abbildung 3.30(a) mit 32 Mbit × 16 verwendet 13 Leitungen für das $\overline{\text{RAS}}$-Signal, 10 Leitungen für das $\overline{\text{CAS}}$-Signal und 2 Leitungen für die Bankauswahl. Mit diesen insgesamt 25 Leitungen lässt sich jede der 2^{25} internen 16-Bit-Zellen adressieren. Im Unterschied dazu zeigt Abbildung 3.30(b) einen 128 Mbit × 4-Entwurf mit 13 Leitungen für das $\overline{\text{RAS}}$-Signal, 12 Leitungen für das $\overline{\text{CAS}}$-Signal und 2 Leitungen für die Bankauswahl. Hier können 27 Signale jede der 2^{27} internen 4-Bit-Zellen adressieren. Die Entscheidung, wie viele Zeilen und Spalten ein Chip erhält, basiert auf technischen Gesichtspunkten. Die Matrix muss nicht unbedingt quadratisch sein.

Diese Beispiele weisen auf zwei getrennte und voneinander unabhängige Fragen beim Entwurf von Speicherchips hin. Die erste betrifft die Ausgabebreite (in Bit): Stellt der Chip 1, 4, 8, 16 oder eine andere Anzahl von Bits gleichzeitig bereit? Zweitens: Werden alle Adressbits an separate Pins gleichzeitig angelegt, oder ist es erforderlich, die Zeilen und Spalten nacheinander wie in den Beispielen von Abbildung 3.30 bereitzustellen? Ein Speicherchipentwickler muss beide Fragen beantworten, bevor er sich mit dem eigentlichen Entwurf beschäftigen kann.

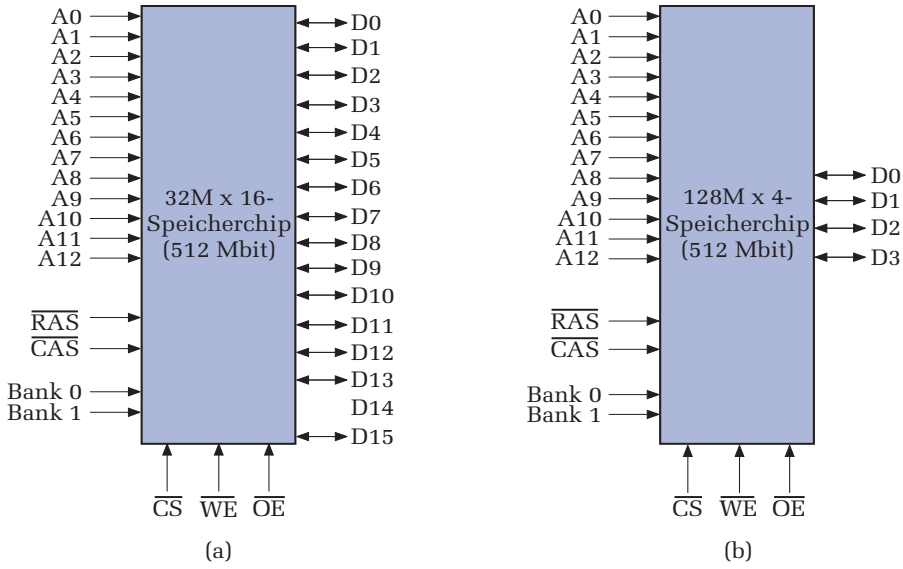


Abbildung 3.30: Zwei Möglichkeiten für die Organisation eines 512-Mbit-Speicherchips

3.3.6 RAM und ROM

Die bislang untersuchten Speicher lassen sich alle lesen und beschreiben. Diese Speicherform nennt man **RAM** (Random Access Memory) – Speicher mit wahlfreiem Zugriff. Diese Bezeichnung ist unglücklich gewählt, weil sämtliche Speicherchips wahlfreien Zugriff bieten. Allerdings hat sich diese Bezeichnung mittlerweile so etabliert, dass man sie jetzt nicht mehr über Bord werfen kann. RAM gibt es in zwei Varianten: statisch und dynamisch. **Statischer RAM** (**SRAM**) wird intern mit Schaltungen realisiert, die unserem grundlegenden D-Flipflop ähneln. Diese Speicher haben die Eigenschaft, ihren Inhalt zu behalten, solange die Betriebsspannung anliegt: Sekunden, Minuten, Stunden oder gar Tage. Statische RAMs sind sehr schnell. Die typischen Zugriffszeiten betragen nur wenige Nanosekunden. Aus diesem Grund realisiert man den Level-2-Cache vorzugsweise mit RAM.

Demgegenüber verwendet **dynamischer RAM** (**DRAM**) keine Flipflops. Stattdessen ist ein dynamischer RAM ein Array von Zellen, die jeweils aus einem Transistor und einem winzigen Kondensator bestehen. Die Kondensatoren lassen sich laden und entladen – ein geladener Kondensator entspricht einer gespeicherten 1, ein entladener Kondensator einer 0. Da sich die elektrische Ladung infolge von Kriechströmen mehr oder weniger schnell abbaut, muss jedes Bit in einem dynamischen RAM alle paar Millisekunden **aufgefrischt** (nachgeladen) werden, um Datenverluste zu vermeiden. Da eine externe Logik dieses Auffrischen übernehmen muss, sieht die Anschlussgestaltung bei dynamischen RAMs komplizierter als bei statischen RAMs aus. Dieser Nachteil gleicht sich aber in vielen Anwendungen durch ihre größeren Speicherkapazitäten aus.

Da dynamische RAMs nur einen Transistor und einen Kondensator pro Bit benötigen (gegenüber sechs Transistoren pro Bit beim besten statischen RAM), weisen dynamische RAMs eine hohe Speicherdichte auf (viele Bits pro Chip). Aus diesem Grund

sind Hauptspeicher nahezu immer aus dynamischen RAMs aufgebaut. Allerdings hat diese große Speicherkapazität ihren Preis: Dynamische RAMs sind relativ langsam (einige zehn Nanosekunden). Somit versucht man, durch eine Kombination aus statischem RAM-Cache und dynamischen RAM-Hauptspeicher die besten Eigenschaften von beiden Typen zu vereinen.

Dynamische RAM-Chips sind in verschiedenen Typen erhältlich. Der älteste Typ (der noch in älteren Computern zu finden sein dürfte) ist der **FPM-DRAM** (FPM = Fast Page Mode). Intern ist dieser DRAM als Bitmatrix organisiert. Zum Adressieren legt man an diesen RAM eine Zeilenadresse an und durchläuft dann schrittweise die Spaltenadressen, wie wir es im Zusammenhang mit Abbildung 3.29 für die Signale $\overline{\text{RAS}}$ und $\overline{\text{CAS}}$ beschrieben haben. Explizite Signale sagen dem Speicher, wann er antworten soll, sodass der Speicher nicht mit dem Hauptsystemtakt synchron läuft.

FPM-DRAMs wurden durch den **EDO-DRAM** (Extended Data Output) abgelöst. Beim EDO-DRAM kann eine zweite Speicherreferenz vor Beendigung der ersten beginnen. Diese einfache Fließbandverarbeitung beschleunigte einen einzelnen Speicherzugriff nicht, verbesserte aber die Speicherbandbreite und lieferte so mehr Worte je Sekunde.

FPM und EDO waren ausreichend, als die Speicherzykluszeiten bei 12 ns und darüber lagen. Mit zunehmenden Prozessorgeschwindigkeiten war schließlich schnellerer Speicher unabdingbar. So wurden FPM und EDO durch **SDRAM** (Synchronous DRAM) – eine Hybridlösung aus statischem und dynamischem RAM – ersetzt. Da SDRAM vom Hauptsystemtakt gesteuert wird, benötigt der Speicherchip keine Steuersignale mehr, die sein Antwortverhalten regeln. Stattdessen teilt der Prozessor dem Speicher mit, wie viele Zyklen er laufen soll, und startet ihn dann. Bei jedem darauf folgenden Zyklus gibt der Speicher 4, 8 oder 16 Bits aus, je nach Anzahl seiner Ausgangsleitungen. Der Wegfall der Steuersignale steigert die Datenübertragungsrate zwischen Prozessor und Speicher.

Die nächste Verbesserung gegenüber SDRAM war DDR-SDRAM (Double Data Rate). Dieser Speichertyp liefert Ausgaben sowohl an der steigenden als auch an der fallenden Flanke des Taktsignals, was die Datenrate verdoppelt. Somit liefert ein 8-Bit breiter DDR-Chip, der mit 200 MHz getaktet wird, 200 Millionen Mal in der Sekunde zwei 8-Bit-Werte (natürlich nur für eine kurze Zeitspanne), was eine theoretische Burst-Rate von 3,2 Gbit/s ergibt.

Nichtflüchtige Speicherchips

RAM ist nicht der einzige Speichertyp. In vielen Anwendungen, z.B. Spielzeugen, Elektrogeräten und Autos, müssen das Programm und ein Teil der Daten auch im stromlosen Zustand gespeichert bleiben. Außerdem ist es hier nach der Installation nicht mehr erforderlich, Programm oder Daten zu ändern. Diese Anforderungen haben zur Entwicklung des **ROM** (Read-Only Memory) geführt, der sich weder absichtlich noch anderweitig ändern oder löschen lässt. Der ROM bekommt seinen Dateninhalt während der Herstellung des Chips. In einem bestimmten Prozessschritt belichtet man dazu eine fotoempfindliche Schicht durch eine Maske, die das gewünschte Bitmuster enthält, und ätzt dann die belichteten (oder unbelichteten) Teile der Oberfläche weg. Das in einem ROM gespeicherte Programm lässt sich nur ersetzen, wenn man den kompletten Chip austauscht.

ROMs sind wesentlich billiger als RAMs, sofern das Auftragsvolumen groß genug ist, damit sich die Herstellungskosten der Maske über die Stückzahl amortisieren. Allerdings sind ROMs unflexibel, weil sich ihr Inhalt nach der Herstellung nicht mehr ändern lässt. Zudem können zwischen der Bestellung und der Auslieferung mehrere Wochen vergehen. Um die Entwicklung von ROM-basierten Produkten zu erleichtern, hat man den **PROM** (Programmable ROM) eingeführt. Dieser Speichertyp ist dem ROM sehr ähnlich, außer dass man ihn (einmalig) vor Ort programmieren kann, wodurch sich der Entwicklungszyklus insbesondere bei Programmänderungen erheblich verkürzt. Viele PROMs enthalten ein Array mit winzigen Sicherungen.

Um eine bestimmte Sicherung durchzubrennen, wählt man ihre Zeile und Spalte aus und legt dann eine hohe Spannung an einen speziellen Pin (den Programmiereingang) des Chips.

Der nächste Entwicklungsschritt auf diesem Weg ist der **EPROM** (Erasable PROM), den man vor Ort nicht nur programmieren sondern auch löschen kann. Setzt man das Quarzglasfenster in einem EPROM rund 15 Minuten einem starken ultravioletten Licht aus, werden alle Bits auf 1 gesetzt. Treten im Verlauf des Designzyklus viele Änderungen auf, sind EPROMs weitaus wirtschaftlicher als PROMs, weil man sie wiederverwenden kann. EPROMs sind normalerweise genau wie statische RAMs organisiert. Beispielsweise verwendet der 4-Mbit-EPROM 27C040 eine Organisation gemäß Abbildung 3.30(a), die für einen statischen RAM typisch ist.

Noch besser als der EPROM ist der **EEPROM** (Electrical Erasable PROM), der sich mit elektrischen Impulsen löschen lässt, sodass keine spezielle Löschkammer mit ultraviolettem Licht erforderlich ist. Zudem ist es möglich, den EEPROM in der Anwendungsschaltung zu programmieren, während man für einen EPROM ein spezielles Programmiergerät benötigt. Nachteilig ist, dass die größten EEPROMs normalerweise nur 1/64 so groß und halb so schnell wie übliche EPROMs sind. EEPROMs können nicht mit DRAMs oder SRAMs konkurrieren, weil sie zehnmal langsamer sind, nur ein hunderstel der Speicherkapazität mitbringen und viel mehr kosten. Man setzt sie nur dort ein, wo ihre Nichtflüchtigkeit entscheidend ist.

Eine neuere Art von EEPROM ist der **Flash-Speicher** (Flash Memory). Im Gegensatz zum EPROM, der durch ultraviolettes Licht gelöscht wird, und zum EEPROM, der byteweise löschbar ist, kann man den Flash-Speicher blockweise löschen und neu beschreiben. Und wie beim EEPROM lässt sich der Flash-Speicher löschen, ohne dass man ihn aus der Schaltung entfernen muss. Verschiedene Hersteller bieten kleine Leiterplatten mit bis zu 1 GB Flash-Speicher an, die man als eine Art „Film“ in digitalen Kameras und ähnlichen Anwendungen nutzen kann, um Bilder zu speichern. Eines Tages lösen Flash-Speicher möglicherweise die Festplatten ab, was angesichts ihrer Zugriffszeiten von 50 ns ein enormer Fortschritt wäre. Das derzeit größte technische Problem ist, dass sie nach rund 100.000 Löschvorgängen regelrecht verschlissen sind, während Festplatten viele Jahre überstehen, ganz egal, wie oft man sie löscht und neu beschreibt. Tabelle 3.3 zeigt eine Übersicht der verschiedenen Speichertypen.

Tabelle 3.3

Vergleich verschiedener Speichertypen

Typ	Kategorie	Löschen	änderbar in Byte-Einheiten	Flüchtig	Typische Anwendung
SRAM	Lesen/ Schreiben	Elektrisch	Ja	Ja	Level-2-Cache
DRAM	Lesen/ Schreiben	Elektrisch	Ja	Ja	Hauptspeicher (alt)
SDRAM	Lesen/ Schreiben	Elektrisch	Ja	Ja	Hauptspeicher (neu)
ROM	Nur Lesen	Nicht möglich	Nein	Nein	Großvolumige Geräte
PROM	Nur Lesen	Nicht möglich	Nein	Nein	Kleinvolumige Geräte
EPROM	Vorwiegend Lesen	UV-Licht	Nein	Nein	Herstellung von Geräteprototypen
EEPROM	Vorwiegend Lesen	Elektrisch	Ja	Nein	Herstellung von Geräteprototypen
Flash-Speicher	Lesen/ Schreiben	Elektrisch	Nein	Nein	„Film“ für digitale Kamera

3.4 CPU-Chips und Busse

Nachdem Sie nun mit Informationen über SSI-, MSI- und Speicherchips ausgerüstet sind, können wir nun die einzelnen Teile zusammensetzen und vollständige Systeme betrachten. Dieser Abschnitt geht zuerst auf einige allgemeine Aspekte von Prozessoren ein, die mit der Ebene der digitalen Logik zu tun haben. Dazu gehört die **Anschlussbelegung** (das **Pinout**), die die Bedeutung der verschiedenen Pins angibt. Da Prozessoren sehr eng mit dem Design der von ihnen genutzten Busse verflochten sind, gibt dieser Abschnitt auch eine Einführung in Buskonzepte. Darauf folgende Abschnitte zeigen ausführliche Beispiele für Prozessoren mit den dazu gehörenden Bussen und Schnittstellen.

3.4.1 Prozessorchips

Fast alle modernen Prozessoren sind auf einem einzigen Chip untergebracht. Dadurch ist ihre Interaktion mit dem restlichen System gut definiert. Jeder Prozessorchip besitzt eine Reihe von Pins, durch die seine gesamte Kommunikation mit der Außenwelt stattfinden muss. Einige Pins geben Signale vom Prozessor aus, andere nehmen

Signale von der Außenwelt entgegen und wieder andere arbeiten in beiden Richtungen. Wenn man die Funktionen aller Pins kennt, wird klar, wie die CPU mit dem Speicher und den E/A-Geräten auf der Ebene der digitalen Logik interagiert.

Die Pins eines Prozessors lassen sich drei Kategorien zuordnen: Adressen, Daten und Steuerung. Diese Pins sind mit funktionell gleichen Pins auf den Speicher- und E/A-Chips über ein Leitungssystem – den so genannten „Bus“ – verbunden. Um einen Befehl aus dem Hauptspeicher abzurufen, legt die CPU zuerst die Speicheradresse dieses Befehls auf ihre Adresspins. Dann aktiviert sie eine oder mehrere Steuerleitungen, um den Speicher darüber zu informieren, dass (zum Beispiel) ein Wort gelesen werden soll. Der Speicher antwortet, indem er das angeforderte Wort auf die Datenpins der CPU legt und ein Bestätigungssignal für die Anforderung aktiviert. Wenn die CPU dieses Signal erkennt, nimmt sie das Wort an und führt den Befehl aus.

Der Befehl kann das Lesen oder Schreiben von Datenworten erfordern. In diesem Fall wiederholt sich der gesamte Vorgang für jedes weitere Wort. Zur Funktionsweise der Lese- und Schreiboperationen kommen wir gleich. Vorläufig interessiert nur, wie der Prozessor mit Speicher und E/A-Geräten kommuniziert: Über seine Pins gibt er Signale aus und nimmt Signale entgegen. Eine andere Kommunikation ist nicht möglich.

Zwei Schlüsselparameter, die die Leistung eines Prozessors bestimmen, sind die Anzahl der Adress- und Datenpins. Ein Chip mit m Adresspins kann bis zu 2^m Speicherstellen adressieren. Übliche Werte von m sind 16, 20, 32 und 64. Analog kann ein Chip mit n Datenpins ein n-Bit-Wort in einer Operation lesen oder schreiben. Übliche Werte von n sind 8, 16, 32 und 64. Eine CPU mit 8 Datenpins benötigt vier Operationen, um ein 32-Bit-Wort zu lesen, während eine CPU mit 32 Datenpins dieselbe Aufgabe in einer einzigen Operation abwickelt. Somit ist ein Chip mit 32 Datenpins wesentlich schneller, aber auch erheblich teurer.

Zusätzlich zu Adress- und Datenpins besitzt jede CPU einige Steuerpins, die das zeitliche Verhalten beim Datenaustausch zwischen CPU und anderen Bausteinen regeln sowie verschiedene andere Aufgaben übernehmen. Alle CPUs besitzen Pins für Betriebsspannung (normalerweise +3,3 V oder +5 V), Masse und ein Taktsignal (eine Rechteckschwingung mit einer bestimmten Frequenz). Hinsichtlich der übrigen Pins bestehen zwischen den einzelnen Prozessoren mehr oder weniger große Unterschiede. Dennoch kann man die Steuerpins grob den folgenden Kategorien zuordnen:

1. Bussteuerung
2. Interrupts
3. Bus-Verwaltung (Konkurrenzbereinigung bei gleichzeitigem Zugriff auf den Bus)
4. Coprozessor-Signalisierung
5. Status
6. Verschiedenes

Auf diese Kategorien gehen wir gleich noch ein. Wenn wir uns später den Prozessoren Pentium 4, UltraSPARC III und 8051 zuwenden, erfahren Sie mehr Einzelheiten. Abbildung 3.31 zeigt einen generischen Prozessorchip mit den entsprechenden Signalgruppen.

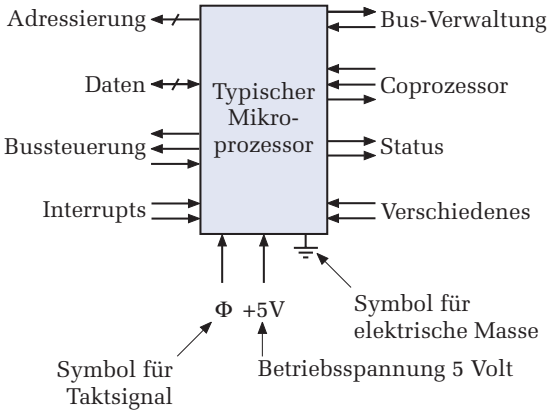


Abbildung 3.31: Logische Anschlussbelegung eines generischen Prozessors. Die Pfeile symbolisieren Ein- und Ausgangssignale. Die kurzen Schrägstriche zeigen an, dass der jeweilige Pfeil für mehrere Leitungen (bzw. Pins) steht. Bei einem konkreten Prozessor steht an diesem Schrägstrich noch eine Zahl, um die tatsächliche Anzahl der Leitungen anzugeben.

Die Bussteuerpins sind vorwiegend Ausgänge von der CPU zum Bus (somit Eingänge in den Speicher und die E/A-Chips) und zeigen an, ob die CPU in den Speicher schreiben, von dort lesen oder etwas anderes tun möchte. Die CPU steuert also über diese Pins das übrige System.

Die Interrupt-Pins sind Eingänge von E/A-Geräten zur CPU. In den meisten Systemen kann der Prozessor eine Operation bei einem E/A-Gerät einleiten und daraufhin mit etwas anderem fortfahren, während das E/A-Gerät seine Arbeit aufnimmt. Sobald die E/A-Operation abgeschlossen ist, aktiviert der E/A-Controllerchip ein Signal auf einem dieser Pins, um die Arbeit des Prozessors zu unterbrechen und ihn das E/A-Gerät bedienen zu lassen – z.B. um zu prüfen, ob E/A-Fehler aufgetreten sind. Einige Prozessoren besitzen ein Ausgangspin, um das Interrupt-Signal zu bestätigen.

Die Bus-Verwaltungs-Pins regeln den Verkehr auf dem Bus, wenn zwei oder mehr Geräte gleichzeitig darauf zuzugreifen versuchen. Im Hinblick auf die Verwaltung zählt die CPU selbst als Gerät und muss den Bus wie jedes andere Gerät anfordern.

Einige Prozessoren sind so konzipiert, dass sie mit Coprozessoren zusammenarbeiten können, beispielsweise mit Gleitkommachips oder Grafikchips. Um die Kommunikation zwischen der CPU und einem Coprozessor zu vereinfachen, gibt es spezielle Pins, um verschiedene Anforderungen zu stellen und zu gewähren.

Neben diesen Signalen kann ein Prozessor mehrere andere Pins besitzen, die zur Kategorie „Verschiedenes“ gehören. Einige davon liefern und übernehmen Statusinformationen, andere können den Computer zurücksetzen und wieder andere dienen dazu, die Kompatibilität mit älteren E/A-Chips zu gewährleisten.

3.4.2 Computer-Busse

Ein **Bus** ist ein gemeinsamer, elektrischer Weg zwischen mehreren Geräten. Busse lassen sich nach ihrer Funktion einteilen. Prozessor-interne Busse realisieren den Datenaustausch mit der ALU, Prozessor-externe Busse verbinden die CPU mit den E/A-Geräten. Jeder Bustyp hat seine eigenen Anforderungen und Eigenschaften. In diesem

und den folgenden Abschnitten konzentrieren wir uns auf Busse, die den Prozessor mit dem Speicher und den E/A-Geräten verbinden. Das nächste Kapitel beschäftigt sich näher mit den Bussen innerhalb des Prozessors.

Die ersten Personalcomputer hatten einen einzigen externen Bus, den **Systembus**. Er bestand aus 50 bis 100 parallelen Leiterzügen auf der Hauptplatine – dem Motherboard – des Computers. Der Anschluss an den Bus erfolgte über Steckverbinder, die in regelmäßigen Abständen angeordnet waren und in die sich Speicher- und E/A-Karten einstecken ließen. Moderne Personalcomputer besitzen im Allgemeinen einen spezialisierten Bus zwischen der CPU und dem Speicher und (mindestens) einen weiteren für die E/A-Geräte. Abbildung 3.32 zeigt ein Minimalsystem mit einem Speicherbus und einem E/A-Bus.

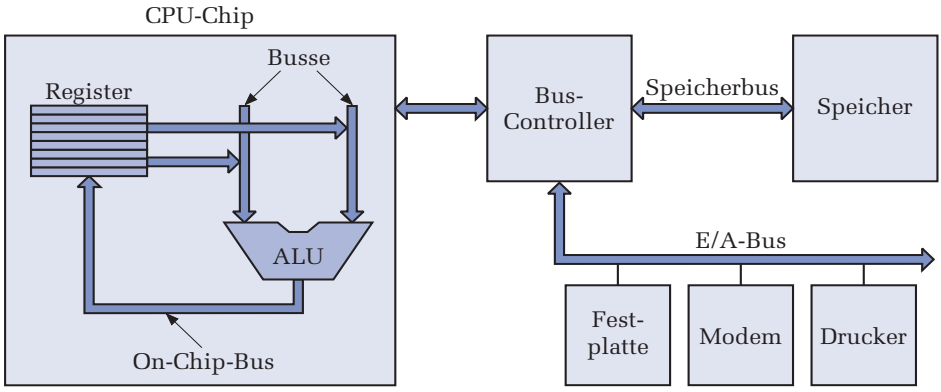


Abbildung 3.32: Ein Computersystem mit mehreren Bussen

In der Literatur stellt man Busse zuweilen als „dicke“ Pfeile dar, wie es auch in Abbildung 3.32 zu sehen ist. Der Unterschied zwischen einem dicken Pfeil und einer einfachen Linie mit Schrägstrich und Angabe der Bits ist eher etwas spitzfindig. Sind alle Bits vom selben Typ, beispielsweise ausschließlich Adressbits oder ausschließlich Datenbits, zieht man die Linie mit kurzem Schrägstrich vor. Sind Adress-, Daten- und Steuerleitungen beteiligt, ist ein dicker Pfeil üblich.

Während es den Entwicklern des Prozessors freisteht, welche Art von Bus sie innerhalb des Chips verwenden, muss es klare Regeln geben, wie der Systembus arbeitet, damit sich Leiterkarten von Drittherstellern anschließen lassen. An diese Regeln müssen sich dann alle anzuschließenden Geräte halten. Bei diesem Regelwerk handelt es sich um das so genannte **Busprotokoll**. Darüber hinaus ist eine mechanische und elektrische Spezifikation erforderlich, damit Platinen verschiedener Hersteller in das vorgesehene Platinenprofil passen, die Steckverbinder mechanisch mit denen auf dem Motherboard übereinstimmen und sämtliche Bedingungen hinsichtlich der Spannungen, der Zeitabläufe usw. eingehalten werden.

In der Computer-Welt sind viele Busse verbreitet. Um einige der bekanntesten – moderne und historische – zu nennen: Omnibus (PDP-8), Unibus (PDP-11), Multibus (8086), VME-Bus (Prozessrechner für wissenschaftliche Einrichtungen), IBM-PC-Bus (PC/XT), ISA-Bus (PC/AT), EISA-Bus (80386), Microchannel (PS/2), Nubus (Macintosh), PCI-Bus (viele PCs), SCSI-Bus (viele PCs und Workstations), Universal Serial Bus (moderne PCs) und FireWire (Konsumgüterelektronik). Die Welt wäre wahrscheinlich schöner, wenn alle Busse außer einem (nun gut, sagen wir außer zweien)

plötzlich von der Erdoberfläche verschwinden würden. Leider scheint eine Standardisierung in diesem Bereich sehr unwahrscheinlich, weil bereits zu viel in diese verschiedenen, untereinander nicht kompatiblen Systeme investiert wurde.

Wie funktionieren Busse? Einige Geräte, die man an einen Bus anschließt, sind aktiv und können Busübertragungen einleiten, während andere passiv sind und auf Anforderungen warten. Die aktiven nennt man **Master**, die passiven **Slaves**. Fordert die CPU einen Laufwerkscontroller auf, einen Block zu lesen oder zu schreiben, tritt die CPU als Master und der Laufwerkscontroller als Slave auf. Allerdings kann der Laufwerkscontroller im weiteren Verlauf auch als Master agieren, wenn er den Speicher anweist, die vom Laufwerk gelesenen Worte zu übernehmen. Tabelle 3.4 listet mehre typische Kombinationen von Master und Slave auf. Speicher kommt unter keinen Umständen als Master infrage.

Tabelle 3.4

Beispiele für Kombinationen von Master und Slave auf einem Bussystem

Master	Slave	Beispiel
CPU	Speicher	Befehle und Daten abrufen
CPU	E/A-Gerät	Datenübertragung einleiten
CPU	Coprozessor	CPU gibt Befehle an den Coprozessor weiter
E/A	Speicher	DMA (Direct Memory Access)
Coprozessor	CPU	Coprozessor ruft Operanden von der CPU ab

Die von Computerbaugruppen abgegebenen binären Signale sind häufig zu schwach, um einen Bus direkt zu speisen. Das gilt vor allem, wenn der Bus relativ lang ist oder viele Geräte an ihn angeschlossen sind. Aus diesem Grund werden die meisten Bus-Master mithilfe eines Chips an den Bus angeschlossen, den man **Bustreiber** (Bus Driver) nennt. Dabei handelt es sich im Wesentlichen um einen digitalen Verstärker. Dementsprechend sind die meisten Slaves über einen **Busempfänger** (Bus Receiver) an den Bus angeschlossen. Für Geräte, die sowohl als Master als auch als Slave auftreten können, verwendet man einen kombinierten Chip namens **Bustransceiver**. Diese Busschnittstellenchips haben meistens Tristate-Ausgänge, die im hochohmigen (inaktiven) Zustand den Bus nicht belasten. Elektrisch ähnlich verhält sich der **offene Kollektorausgang**, der die jeweiligen Busleitungen nur im leitenden Zustand niederohmig mit Masse verbindet. Wenn zwei oder mehr Bauelemente mit offenem Kollektorausgang die Leitung gleichzeitig aktivieren, ist das Ergebnis die boolesche OR-Verknüpfung der betreffenden Signale. Diese logische Funktion bezeichnet man auch als **Wired-OR** (OR durch Verbinden). Auf den meisten Bussen sind einige Leitungen für Tristate-Betrieb ausgelegt und andere, die die Wired-OR-Eigenschaft benötigen, für offene Kollektorausgänge konzipiert.

Wie eine CPU hat auch ein Bus Adress-, Daten- und Steuerleitungen. Zwischen den Pins der CPU und den Bussignalen besteht aber nicht unbedingt eine 1:1-Zuordnung. Beispielsweise kodieren einige Prozessoren mit Signalen an drei Pins, ob ein Speicherlesen, ein Speicherschreiben, ein E/A-Lesen, ein E/A-Schreiben oder eine andere Ope-

ration erfolgen soll. Ein typischer Bus besitzt eine Leitung für Speicherschreiben, eine zweite für Speicherlesen, eine dritte für E/A-Lesen, eine vierte für E/A-Schreiben usw. Zwischen dem Prozessor und einem solchen Bus ist dann ein Dekodierchip erforderlich, um die beiden Seiten aneinander anzupassen. Dieser Chip konvertiert das mit 3 Bit kodierte Signal in getrennte Signale, die die Busleitungen speisen können.

Design und Betrieb von Bussen sind recht komplexe Themen, denen ganze Bücher gewidmet sind, z.B. [Anderson u.a., 2004] und [Solari und Willse, 2004]. Die hauptsächlichen Kriterien beim Busdesign sind Busbreite, Bustaktung, Bus-Verwaltung und Busoperationen. Diese Kriterien haben großen Einfluss auf Geschwindigkeit und Bandbreite des Busses. Die nächsten vier Abschnitte beschäftigen sich ausführlich mit diesen Fragen.

3.4.3 Busbreite

Die Busbreite ist der offensichtlichste Entwurfsparameter. Je mehr Adressleitungen ein Bus hat, um so mehr Speicher kann die CPU direkt adressieren. Verfügt ein Bus über n Adressleitungen, kann eine CPU über diesen Bus 2^n verschiedene Speicherstellen adressieren. Um große Speicher zu ermöglichen, brauchen Busse viele Adressleitungen. Das hört sich einfacher an, als es ist.

Breite Busse erfordern mehr Leitungen als schmale. Sie beanspruchen mehr Platz (auf dem Motherboard) und verlangen größere Steckverbinder. Diese Faktoren machen den Bus teuer. Somit ist ein Kompromiss zwischen der maximalen Speichergröße und den Systemkosten zu finden. Ein System mit einem Adressbus von 64 Leitungen und 2^{32} Bytes Speicher kostet mehr als ein System mit 32 Adressleitungen und derselben Speicherkapazität. Auch die Möglichkeit einer späteren Erweiterung ist nicht umsonst zu haben.

Aus diesem Grund neigen viele Systemdesigner zu kurzsichtigen Lösungen, die später zu verhängnisvollen Konsequenzen führen. Der ursprüngliche IBM-PC enthielt eine 8088-CPU und einen 20-Bit-Adressbus, wie in Abbildung 3.33(a) dargestellt. Mit diesen 20 Bit war der PC in der Lage, 1 MB Speicher zu adressieren.

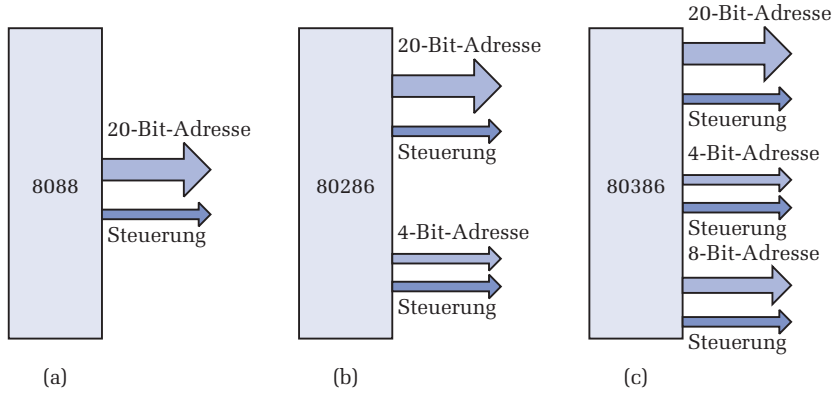


Abbildung 3.33: Wachstum eines Adressbusses im Laufe der Zeit

Als der nächste CPU-Chip (der 80286) herauskam, entschloss sich Intel, den Adressraum auf 16 MB zu vergrößern, sodass vier weitere Busleitungen hinzugefügt werden

mussten (ohne dass Konflikte mit den bereits vorhandenen 20 Adressleitungen auftreten und somit die Abwärtskompatibilität gewahrt bleibt). Dieser Bus ist in Abbildung 3.33(b) zu sehen. Leider waren auch weitere Steuerleitungen erforderlich, um das neue Adressierungsschema zu unterstützen. Mit Einführung des 80386 kamen weitere acht Adressleitungen und mehr Steuerleitungen hinzu, wie es Abbildung 3.33(c) veranschaulicht. Das resultierende Konzept (der EISA-Bus) wäre wesentlich ordentlicher ausgefallen, wenn man den Bus von vornherein mit 32 Leitungen ausgestattet hätte.

Nicht nur die Anzahl der Adressleitungen nimmt mit der Zeit zu. Das Gleiche gilt auch für die Datenleitungen, allerdings aus einem etwas anderen Grund. Es gibt zwei Möglichkeiten, die Datenbandbreite eines Busses zu erhöhen: Verringerung der Buszykluszeit (mehr Übertragungen je Sekunde) oder Erhöhung der Datenbusbreite (mehr Bits pro Übertragung). Eine Beschleunigung des Busses ist zwar möglich, aber schwierig zu realisieren, weil sich die Signale auf verschiedenen Leitungen mit geringfügig unterschiedlichen Geschwindigkeiten ausbreiten. Diesen Effekt bezeichnet man als **Bus Skew** (Signalversatz auf dem Bus). Er tritt um so deutlicher zutage, je schneller der Bus ist.

Ein weiteres Problem bei der Beschleunigung des Busses ist der Verlust der Abwärtskompatibilität. Ältere Platinen, die für den langsameren Bus konzipiert wurden, funktionieren nicht am neuen Bus. Wenn man die alten Platinen als überholt deklariert, kommt weder bei den Besitzern noch den Herstellern der alten Platinen Freude auf. Deshalb verbessert man die Leistung normalerweise dadurch, dass man mehr Datenleitungen vorsieht, wie es aus Abbildung 3.33 hervorgeht. Es liegt auf der Hand, dass dieses allmähliche Wachstum letztlich nicht zu einem sauberen Entwurf führt. Beispielsweise sind der IBM-PC und seine Nachfolger von 8 Datenleitungen zu 16 und dann zu 32 übergangen, wobei der Bus praktisch derselbe geblieben ist.

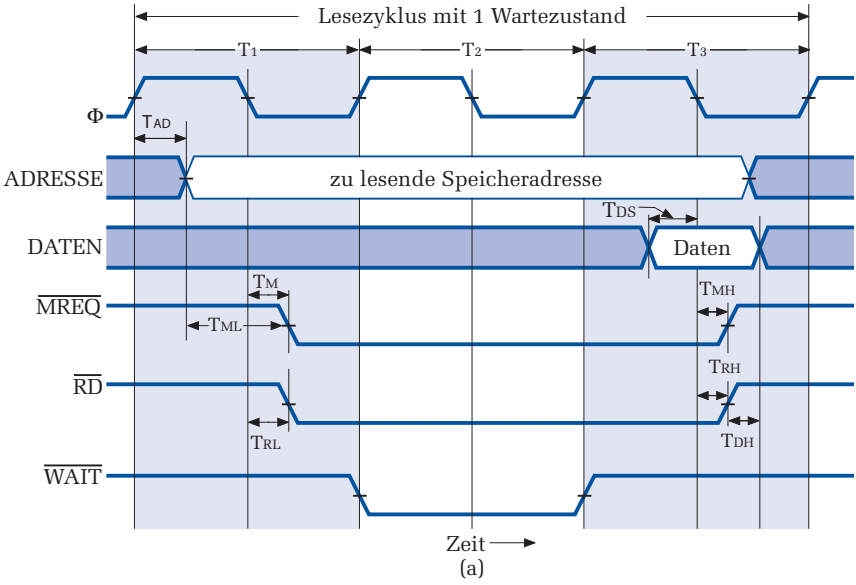
Um das Problem sehr breiter Busse zu umgehen, greifen Entwickler manchmal auf einen **Multiplexbus** zurück. Dieses Konzept führt die Daten- und Adressleitungen nicht separat aus, sondern verwendet zum Beispiel 32 Leitungen für Adressen und Daten gemeinsam. Eine Busoperation belegt diese Leitungen zunächst mit der Adresse und verwendet sie später für die Daten. Das bedeutet beispielsweise bei einem Schreibvorgang auf dem Speicher, dass zuerst die Adresse über die Leitungen zum Speicher übertragen werden muss, bevor man die Daten auf den Bus legen kann. Bei getrennten Leitungen lassen sich die Adresse und die Daten zusammen auf dem Bus übertragen. Das Multiplexen der Leitungen verringert die Bandbreite (und die Kosten), resultiert aber auch in einem langsameren System. Busdesigner müssen diese Optionen sorgfältig gegeneinander abwägen.

3.4.4 Bustaktung

Busse lassen sich nach ihrem Taktschema in zwei Kategorien unterteilen. Bei einem **synchronen Bus** wird eine Leitung von einem Quarzoszillator gesteuert, der eine Rechteckschwingung mit einer Frequenz zwischen 5–100 MHz erzeugt. Alle Busaktivitäten benötigen ein ganzzahliges Vielfaches dieser Zyklen, der so genannten **Buszyklen**. Die zweite Variante ist der **asynchrone Bus** ohne Mastertakt. Die Buszyklen können jede erforderliche Länge haben und müssen nicht unbedingt zwischen allen Gerätepaaren gleich sein. Die folgenden Unterabschnitte beschäftigen sich ausführlich mit diesen beiden Bustypen.

Synchrone Busse

Sehen Sie sich als Beispiel für die Funktionsweise eines synchronen Busses das Taktschema in (a) an. Wir verwenden hier einen 100-MHz-Takt, der einem Buszyklus von 10 ns entspricht. Dies mag im Vergleich zu CPU-Geschwindigkeiten von 3 GHz und mehr langsam erscheinen; es gibt aber kaum schnellere PC-Busse. Zum Beispiel läuft der verbreitete PCI-Bus normalerweise entweder mit 33 MHz oder mit 66 MHz. Die Gründe für die Langsamkeit der heutigen Busse wurden bereits erwähnt: technische Entwurfsprobleme wie Bus-Skew und Abwärtskompatibilität.



(a)

Symbol	Parameter	Min [in ns]	Max [in ns]
T_{AD}	Adressausgabeverzögerung		4
T_{ML}	Adresse ist vor $\overline{\text{MREQ}}$ stabil	2	
T_M	$\overline{\text{MREQ}}$-Verzögerung von der fallenden Flanke von Φ in T_1		3
T_{RL}	RD-Verzögerung von der fallenden Flanke von Φ in T_1		3
T_{DS}	Dateneinrichtezeit vor der fallenden Flanke von Φ	2	
T_{MH}	$\overline{\text{MREQ}}$-Verzögerung von der fallenden Flanke von Φ in T_3		3
T_{RH}	$\overline{\text{RD}}$-Verzögerung von der fallenden Flanke von Φ in T_3		3
T_{DH}	Datenhaltezeit nach der Deaktivierung von $\overline{\text{RD}}$	0	

(b)

Abbildung 3.34: (a) Signalverlauf auf einem synchronen Bus; (b) Spezifikation wichtiger Zeitpunkte

Außerdem gehen wir in unserem Beispiel davon aus, dass das Lesen aus dem Speicher ab dem Zeitpunkt, an dem die Adresse stabil ist, 15 ns dauert. Wir werden gleich sehen, dass es mit diesen Parametern drei Buszyklen dauert, um ein Wort zu lesen. Der erste Zyklus beginnt an der steigenden Flanke von T_1 und der dritte endet an der steigenden Flanke von T_4. Beachten Sie, dass die Flanken schräg verlaufend dargestellt sind, weil jede elektrische Signaländerung eine bestimmte Mindestzeit benötigt. Bei diesem Beispiel nehmen wir an, dass die Änderung eines Signals 1 ns dauert. Alle Leitungen (Takt, Adresse, Daten, $\overline{\text{MREQ}}$, $\overline{\text{RD}}$ und $\overline{\text{WAIT}}$) werden auf derselben Zeitskala dargestellt.

Der Beginn von T_1 ist durch die steigende Flanke des Taktgebers definiert. Irgendwo auf dem Weg durch T_1 legt die CPU die Adresse des gewünschten Wortes auf die Adressleitungen. Da die Adresse im Gegensatz zum Takt kein Einzelwert ist, können wir sie in der Abbildung nicht als einzelnes Signal darstellen. Stattdessen erscheint die Adresse in Form von zwei Linien, die sich bei den Zeitpunkten kreuzen, an denen sich die Adresse ändert. Außerdem bedeutet die Schattierung, dass der Wert im schattierten Bereich nicht von Belang ist. Für die Daten zeigt die Schattierung an, dass der Inhalt der Datenleitungen bis weit in T_3 hinein uninteressant ist.

Nachdem sich die Adressleitungen auf die neuen Werte einpendeln konnten, werden $\overline{\text{MREQ}}$ und $\overline{\text{RD}}$ aktiviert. Das erste Signal zeigt an, dass auf den Speicher (und nicht auf ein E/A-Gerät) zugegriffen wird, während das zweite Signal für Leseoperationen aktiviert und für Schreiboperationen deaktiviert wird. Da der Speicher nach der Stabilisierung der Adresse (mitten im ersten Taktzyklus) 15 ns braucht, kann er die angeforderten Daten innerhalb von T_2 noch nicht bereitstellen. Damit die CPU die Daten nicht zu früh erwartet, aktiviert der Speicher die Leitung $\overline{\text{WAIT}}$ zu Beginn von T_2. Diese Aktion fügt **Wartezustände** (zusätzliche Buszyklen) ein, bis der Speicher fertig ist und $\overline{\text{WAIT}}$ deaktiviert. In unserem Beispiel wurde ein Wartezustand (T_2) eingefügt, weil der Speicher zu langsam ist. Der Speicher deaktiviert $\overline{\text{WAIT}}$ zu Beginn von T_3, wenn sicher ist, dass er die Daten im laufenden Zyklus bereitstellen kann.

Während der ersten Hälfte von T_3 legt der Speicher die Daten auf die Datenleitungen. An der fallenden Flanke von T_3 liest die CPU die Datenleitungen und speichert den Wert in einem internen Register. Nach dem Lesen der Daten deaktiviert die CPU $\overline{\text{MREQ}}$ und $\overline{\text{RD}}$. Bei Bedarf kann an der nächsten steigenden Flanke des Takts ein weiterer Speicherzyklus beginnen. Diese Sequenz lässt sich beliebig oft wiederholen.

Die Taktspezifikation von (b) erläutert acht Symbole, die im Taktdiagramm eingezeichnet sind. Beispielsweise ist T_{AD} das Zeitintervall zwischen der steigenden Flanke von T_1 und der Ausgabe der Adresse. Entsprechend der Taktspezifikation ist $T_{AD} \leq 4$ ns. Damit garantiert der CPU-Hersteller, dass die CPU innerhalb eines beliebigen Lesezyklus die Adresse der zu lesenden Daten innerhalb von 4 ns gerechnet ab dem Mittelpunkt der steigenden Flanke von T_1 ausgibt.

Die Taktspezifikationen verlangen auch, dass die Daten auf den Datenleitungen mindestens T_{DS} (2 ns) vor der fallenden Flanke von T_3 verfügbar sein müssen, damit sich die Signale auf den Busleitungen stabilisieren (einpendeln) können, bevor die CPU die Daten übernimmt. Die Kombination der Einschränkungen für T_{AD} und T_{DS} bedeuten, dass dem Speicher im ungünstigsten Fall ab der Adressausgabe nur $25 - 4 - 2 = 19$ ns bleiben, um die Daten bereitzustellen. Da 10 ns auch im ungünstigsten Fall ausreichen, kann ein 10-ns-Speicher immer innerhalb von T_3 antworten. Ein 20-ns-Speicher müsste allerdings einen zweiten Wartezustand einfügen und innerhalb von T_4 antworten.

Die Taktspezifikation gewährleistet weiterhin, dass die Adresse mindestens 2 ns vor der Aktivierung von $\overline{\text{MREQ}}$ eingerichtet ist. Diese Zeit kann wichtig sein, falls $\overline{\text{MREQ}}$ die Chipauswahl auf dem Speicherchip ansteuert, weil einige Speicher eine gewisse Adresseinrichtungszeit vor der Chipauswahl verlangen. Selbstverständlich sollte der Systemdesigner keinen Speicherchip wählen, der eine Einrichtungszeit von 3 ns benötigt.

Die Einschränkungen auf T_M und T_{RL} bedeuten, dass $\overline{\text{MREQ}}$ und $\overline{\text{RD}}$ innerhalb von 3 ns ab der fallenden Flanke von T_1 aktiviert werden. Im ungünstigsten Fall bleiben dem Speicherchip nur 10 + 10 – 3 – 2 = 15 ns nach der Aktivierung von $\overline{\text{MREQ}}$ und $\overline{\text{RD}}$, um seine Daten auf den Bus zu bringen. Diese Einschränkung gilt zusätzlich zum (und unabhängig vom) 15-ns-Intervall, das nach der Stabilisierung der Adresse nötig ist.

T_{MH} und T_{RH} geben an, wie lange es dauert, $\overline{\text{MREQ}}$ und $\overline{\text{RD}}$ zu deaktivieren, nachdem die CPU die Daten übernommen hat. Schließlich sagt T_{DH} aus, wie lange der Speicher die Daten nach dem Deaktivieren von $\overline{\text{RD}}$ auf dem Bus halten muss. Bei unserem Beispielprozessor kann der Speicher die Daten vom Bus nehmen, sobald $\overline{\text{RD}}$ deaktiviert wurde. Bei einigen neueren CPU-Typen müssen die Daten aber etwas länger stabil gehalten werden.

Wir möchten darauf hinweisen, dass die Abbildung eine stark vereinfachte Version echter Takteinschränkungen ist. In der Praxis hat man es stets mit einer ganzen Reihe weiterer wichtiger Zeiten zu tun. Dennoch vermittelt das Beispiel eine gute Vorstellung von der Funktionsweise eines synchronen Busses.

Abschließend sei erwähnt, dass Steuersignale auf High oder Low aktiviert werden können. Es bleibt dem Busdesigner überlassen, welche Variante er wählt – die Entscheidung kann im Wesentlichen willkürlich erfolgen. Man kann dies als Hardwareentsprechung zur Entscheidung eines Programmierers betrachten, ob er freie Blöcke auf dem Datenträger in einer Zuordnungstabelle durch Nullen oder Einsen kennzeichnet.

Asynchrone Busse

Obwohl sich mit synchronen Bussen aufgrund ihrer diskreten Zeitintervalle relativ einfach arbeiten lässt, weisen sie auch einige Probleme auf. Erstens spielt sich alles in ganzzahligen Vielfachen des Bustakts ab. Selbst wenn CPU und Speicher eine Übertragung in 3,1 Zyklen fertig stellen könnten, müssen sie das Ganze auf 4,0 strecken, weil Teilzyklen verboten sind.

Schwerer wiegt allerdings folgende Tatsache: Nachdem einmal die Entscheidung für einen bestimmten Buszyklus gefallen ist und Speicher sowie E/A-Karten dafür hergestellt wurden. lassen sich künftige technische Verbesserungen kaum nutzen. Nehmen wir beispielsweise an, dass ein paar Jahre nach der Entwicklung des Systems von Abbildung 3.34 neue Speicher mit Zugriffszeiten von 8 ns statt 15 ns auf den Markt kommen. Damit entfallen die Wartezustände für den Speicher und der Rechner läuft schneller. Und da sich die Technik ständig weiterentwickelt, sind etwas später 4-ns-Speicher verfügbar. Jetzt gibt es aber keinen Leistungsgewinn mehr, weil die Mindestzeit für eine Leseoperation bei diesem Buskonzept zwei Zyklen beträgt.

Das Ganze lässt sich auch so formulieren: Ist an einem synchronen Bus mit heterogenen Geräten zu rechnen, von denen einige schnell und einige langsam sind, muss man den Bus für das langsamste Gerät auslegen und die schnellsten Geräte können ihr volles Potenzial nie ausreizen.

Gemischte Technologien lassen sich in den Griff bekommen, wenn man zu einem asynchronen Bus übergeht, d.h. zu einem Bus ohne Mastertakt wie in Abbildung 3.35

gezeigt. Anstatt alles an den Takt zu binden, aktiviert der Busmaster ein spezielles Signal, sobald er die Adresse, $\overline{\text{MREQ}}$, $\overline{\text{RD}}$ und alle sonst noch erforderlichen Signale aktiviert hat. Dieses Signal nennen wir $\overline{\text{MSYN}}$ (Master SYNchronization). Erkennt der Slave dieses Signal, führt er die ihm übertragene Aufgabe schnellstmöglich aus und aktiviert anschließend $\overline{\text{SSYN}}$ (Slave SYNchronization).

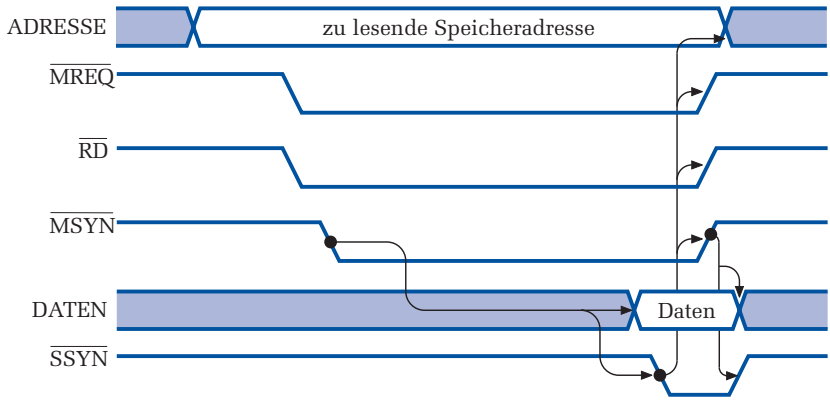


Abbildung 3.35: Arbeitsweise eines asynchronen Busses

Sobald $\overline{\text{SSYN}}$ aktiviert wurde, weiß der Master, dass die Daten verfügbar sind. Er übernimmt sie und deaktiviert die Adressleitungen sowie $\overline{\text{MREQ}}$, $\overline{\text{RD}}$ und $\overline{\text{MSYN}}$. Wenn der Slave die Deaktivierung von $\overline{\text{MSYN}}$ registriert, weiß er, dass der Zyklus beendet ist. Er deaktiviert also $\overline{\text{SSYN}}$ und wir befinden uns wieder in der Ausgangssituation, in der alle Signale deaktiviert sind und auf den nächsten Master warten.

Bei Taktdiagrammen von asynchronen (und manchmal auch von synchronen) Bussen verwendet man Pfeile, um Ursache und Wirkung anzuzeigen, wie es in Abbildung 3.35 zu sehen ist. Die Aktivierung von $\overline{\text{MSYN}}$ löst die Aktivierung der Datenleitungen aus und veranlasst außerdem den Slave, $\overline{\text{SSYN}}$ zu aktivieren. Die Aktivierung von $\overline{\text{SSYN}}$ verursacht wiederum, dass die Adressleitungen sowie $\overline{\text{MREQ}}$, $\overline{\text{RD}}$ und $\overline{\text{MSYN}}$ deaktiviert werden. Schließlich zieht die Deaktivierung von $\overline{\text{MSYN}}$ die Deaktivierung von $\overline{\text{SSYN}}$ nach sich, was den Lesevorgang beendet und das System in seinen ursprünglichen Zustand zurückbringt.

Eine auf diese Weise ineinander greifende Menge von Signalen bezeichnet man als **vollständigen Handshake**. Der wesentliche Teil besteht aus vier Ereignissen:

1. $\overline{\text{MSYN}}$ wird aktiviert.
2. $\overline{\text{SSYN}}$ wird als Reaktion auf $\overline{\text{MSYN}}$ aktiviert.
3. $\overline{\text{MSYN}}$ wird als Reaktion auf $\overline{\text{SSYN}}$ deaktiviert.
4. $\overline{\text{SSYN}}$ wird als Reaktion auf die Deaktivierung von $\overline{\text{MSYN}}$ deaktiviert.

Es liegt auf der Hand, dass vollständige Handshakes zeitunabhängig sind. Jedes Ereignis wird durch ein vorhergehendes Ereignis und nicht durch einen Taktimpuls ausgelöst. Arbeitet ein bestimmtes Master/Slave-Paar langsam, wirkt sich das in keiner Weise auf das nächste Master/Slave-Paar aus, das wesentlich schneller ist.

Der Vorteil eines asynchronen Busses dürfte nun klar sein. Tatsache ist aber, dass die meisten Busse synchron arbeiten. Das ist darauf zurückzuführen, dass sich synchrone Systeme einfacher entwickeln lassen. Die CPU aktiviert einfach nur ihre Signale, und der Speicher reagiert einfach nur. Es gibt keine Rückmeldung (Ursache und Wirkung), aber wenn die Komponenten richtig ausgewählt sind, funktioniert alles ohne Handshake. Außerdem hat man im Laufe der Zeit sehr viel in die synchrone Bustechnik investiert.

3.4.5 Bus-Verwaltung

Bis jetzt haben wir stillschweigend angenommen, dass es nur einen Busmaster – die CPU – gibt. In Wirklichkeit müssen E/A-Chips zum Busmaster werden, um den Speicher zu lesen oder zu beschreiben und um Interrupts auszulösen. Auch Coprozessoren fällt gegebenenfalls die Busmasterfunktion zu. Dabei taucht die Frage auf: „Was passiert, wenn zwei oder mehr Geräte gleichzeitig die Rolle des Busmasters übernehmen wollen?“ Die Antwort ist, dass ein bestimmter Mechanismus, die sogenannte **Bus-Verwaltung** (Bus Arbitration), erforderlich ist, um Chaos zu verhindern.

Verwaltungsmechanismen können zentralisiert oder dezentralisiert arbeiten. Zuerst betrachten wir die zentrale Verwaltung. Abbildung 3.36(a) zeigt eine besonders einfache Form der zentralen Verwaltung. In diesem Schema bestimmt ein einzelner **Bus-Verwalter**, wer als Nächstes an die Reihe kommt. Bei vielen CPUs ist der Verwalter im CPU-Chip integriert, doch manchmal ist ein separater Chip erforderlich. Der Bus enthält eine einzelne Anforderungsleitung vom Typ Wired-OR, die von einem oder mehreren Geräten aktiviert werden kann. Der Verwalter hat keine Möglichkeit festzustellen, wie viele Geräte den Bus angefordert haben. Er kann lediglich zwischen den Kategorien „einige Anforderungen“ und „keine Anforderungen“ unterscheiden.

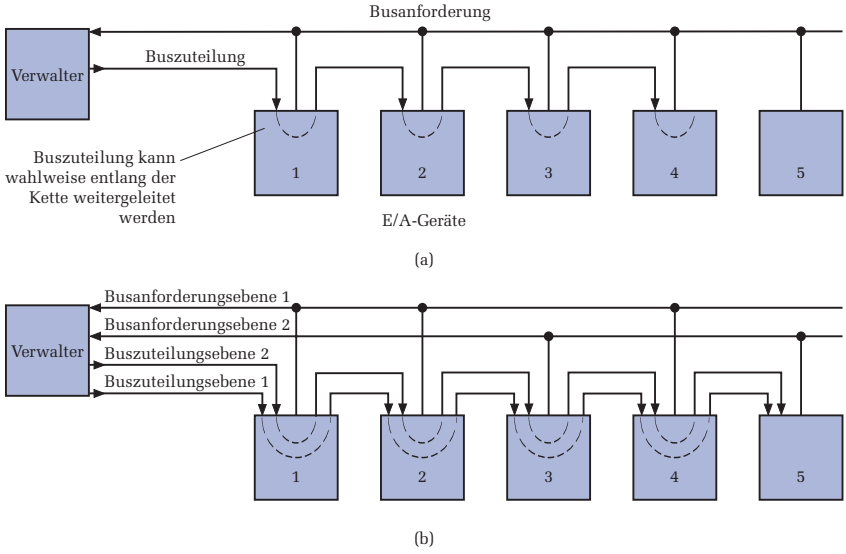


Abbildung 3.36: (a) Zentraler Bus-Verwalter mit einer Ebene in Daisy-Chain-Formation; (b) Der gleiche Verwalter, jedoch mit zwei Ebenen

Wenn der Verwalter eine Busanforderung empfängt, gibt er eine Zuteilung aus, indem er die Buszuteilungsleitung aktiviert. Diese Leitung zieht sich wie eine Lichterkette für den Weihnachtsbaum durch alle in Reihe liegenden E/A-Geräte. Erkennt das dem Verwalter physisch am nächsten liegende Gerät die Zuteilung, prüft es, ob es eine Anforderung gestellt hat. In diesem Fall übernimmt es den Bus und leitet die Zuteilung nicht mehr an den danach liegenden Teil der Kette weiter. Hat das Gerät keine Anforderung gestellt, reicht es die Zuteilung an das nächste Gerät in der Reihe weiter, das sich genauso verhält, und so weiter, bis ein Gerät die Zuteilung akzeptiert und den Bus übernimmt. Dieses Schema bezeichnet man als **Daisy-Chain** („Kette aus Gänseblümchen"). Praktisch weist es jedem Gerät eine Priorität zu, die um so größer ist, je näher das Gerät am Verwalter liegt. Das nächstliegende Gerät gewinnt.

Um die impliziten Prioritäten zu umgehen, die sich aus der Entfernung zum Verwalter ergeben, besitzen viele Busse mehrere Prioritätsebenen. Für jede Prioritätsebene gibt es eine Busanforderungsleitung und eine Buszuteilungsleitung. Der Bus in Abbildung 3.36(b) hat zwei Ebenen, die mit 1 und 2 nummeriert sind (reale Busse weisen oftmals 4, 8 oder 16 Ebenen auf). Jedes Gerät wird an eine der Busanforderungsebenen angeschlossen, wobei man zeitkritischere Geräte den Leitungen mit den höheren Prioritäten zuordnet. In Abbildung 3.36(b) verwenden die Geräte 1, 2 und 4 die Priorität 1, die Geräte 3 und 5 die Priorität 2.

Wenn mehrere Prioritätsebenen gleichzeitig angefordert werden, gibt der Verwalter eine Zuteilung nur auf der Ebene mit der höchsten Priorität aus. Unter den Geräten derselben Priorität kommt das Daisy-Chain-Verfahren zum Einsatz. Wenn bei der Variante von Abbildung 3.36(b) Konflikte auftreten, gewinnt Gerät 2 gegenüber Gerät 4, das seinerseits Gerät 3 schlägt. Gerät 5 hat die niedrigste Priorität, weil es am Ende der Kette mit der niedrigsten Priorität liegt.

Übrigens ist es technisch nicht notwendig, die Buszuteilungsleitung der Ebene 2 seriell durch die Geräte 1 und 2 zu schleifen, da sie auf dieser Ebene ohnehin keine Anforderungen vornehmen können. Allerdings ist es konstruktiv wesentlich einfacher, alle Zuteilungsleitungen durch alle Geräte zu führen, als spezielle Verdrahtungen vorzusehen, die von den Prioritäten der jeweiligen Geräte abhängig sind.

Einige Verwalter verfügen über eine dritte Leitung, die von einem Gerät aktiviert wird, wenn es eine Zuteilung erhalten und den Bus belegt hat. Sobald das Gerät diese Rückmeldeleitung aktiviert hat, können die Anforderungs- und Zuteilungsleitungen deaktiviert werden. Somit ist es möglich, dass andere Geräte den Bus anfordern, während das erste Gerät den Bus noch verwendet. Wenn dann die aktuelle Busübertragung abgeschlossen ist, wird der nächste Busmaster bereits ausgewählt sein. Er kann seine Arbeit sofort aufnehmen, sobald die Rückmeldeleitung deaktiviert wurde. Zu diesem Zeitpunkt beginnt die nächste Verwaltungsrunde. Dieses Verfahren erfordert eine zusätzliche Busleitung und mehr Logik in jedem Gerät, nutzt aber die Buszyklen besser aus.

In Systemen, bei denen sich der Speicher auf dem Hauptbus befindet, muss auch die CPU mit allen E/A-Geräten in fast jedem Zyklus um den Bus kämpfen. In dieser Situation ist es üblich, der CPU die niedrigste Priorität zu geben, sodass sie den Bus nur erhält, wenn niemand sonst ihn haben möchte. Diesem Vorgehen liegt der Gedanke zugrunde, dass die CPU immer warten kann, während E/A-Geräte oftmals unverzüglich auf den Bus zugreifen müssen, da sie andernfalls eintreffende Daten verlieren. Festplatten mit hoher Drehzahl können nicht warten. In vielen modernen Computersystemen vermeidet man dieses Problem dadurch, dass man den Speicher auf einem von den E/A-Geräten getrennten Bus unterbringt, sodass nicht alle Geräte um den Bus konkurrieren müssen.

Dezentrale Bus-Verwaltung ist ebenfalls möglich. Beispielsweise ist ein Computer mit 16 priorisierten Busanforderungsleitungen denkbar. Möchte ein Gerät den Bus verwenden, aktiviert es seine Anforderungsleitung. Alle Geräte überwachen sämtliche Anforderungsleitungen, sodass am Ende eines jeden Buszyklus jedes Gerät weiß, ob es die Anforderung mit der höchsten Priorität gestellt hat und damit die Erlaubnis erhält, den Bus im nächsten Zyklus zu verwenden. Im Vergleich zur zentralen Verwaltung erfordert diese Methode mehr Busleitungen, vermeidet aber die potenziellen Kosten des Verwalters. Außerdem begrenzt dieses Verfahren die Anzahl der Geräte auf die Anzahl der Anforderungsleitungen.

Abbildung 3.37 zeigt eine weitere Variante der dezentralen Bus-Verwaltung. Sie verwendet nur drei Leitungen, unabhängig davon, wie viele Geräte vorhanden sind. Die erste Busleitung ist eine Wired-OR-Leitung für die Anforderung des Busses. Die zweite Busleitung heißt `BUSY` (beschäftigt) und wird vom momentanen Busmaster aktiviert. Die dritte Leitung läuft als Daisy Chain durch alle Geräte und dient dazu, über die Busnutzung zu entscheiden. Der Kopf dieser Kette wird fest auf +5 V gelegt und bleibt damit ständig aktiviert.

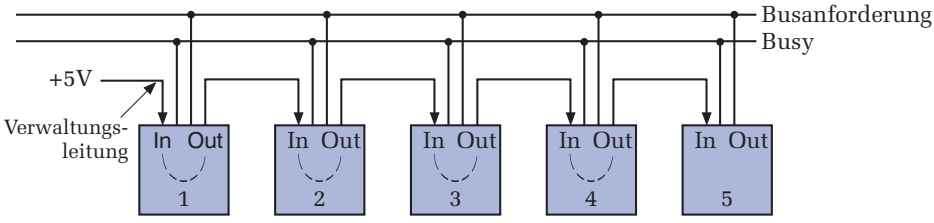


Abbildung 3.37: Dezentrale Bus-Verwaltung

Wenn kein Gerät den Bus beansprucht, wird die aktivierte Verwaltungsleitung an alle Geräte weitergegeben. Um den Bus zu erlangen, prüft ein Gerät zuerst, ob sich der Bus im Leerlauf befindet und ob das empfangene Verwaltungssignal `IN` aktiviert ist. Ist `IN` deaktiviert, kann das Gerät nicht zum Busmaster werden und deaktiviert `OUT`. Wenn `IN` jedoch aktiviert ist, deaktiviert das Gerät das Signal `OUT`. Der nächste Nachbar in der Kette erkennt damit das Signal `IN` und deaktiviert seinerseits `OUT`. Folglich erhalten alle darauf folgenden Geräte in der Kette das deaktivierte `IN` und negieren dementsprechend `OUT`. Am Ende dieser Prozedur hat nur ein Gerät `IN` aktiviert und `OUT` deaktiviert. Dieses Gerät wird zum Busmaster, aktiviert `BUSY` und `OUT` und beginnt mit der Übertragung.

Es zeigt sich, dass das Gerät ganz links immer den Bus bekommt, wenn es ihn anfordert. Dieses Schema ist also mit der ursprünglichen Daisy-Chain-Verwaltung vergleichbar, außer dass es keinen Verwalter gibt. Daher ist es billiger, schneller und für Verwalter-Ausfälle nicht anfällig.

3.4.6 Busoperationen

Bisher haben wir nur gewöhnliche Buszyklen betrachtet, bei denen ein Master (in der Regel die CPU) von einem Slave (in der Regel der Speicher) liest oder dorthin schreibt. Daneben gibt es noch mehrere andere Arten von Buszyklen. Einige davon sehen wir uns jetzt an.

Normalerweise wird je ein Wort übertragen. Setzt man aber einen Cache ein, ist es wünschenswert, eine ganze Cachezeile (d.h. 16 aufeinander folgende 32-Bit-Worte) auf einmal einzulesen. Vielfach lassen sich Blockübertragungen effizienter als aufeinander folgende einzelne Übertragungen gestalten. Beginnt eine Blockleseoperation, teilt der Busmaster dem Slave die Anzahl der zu übertragenden Worte mit. Dazu kann er beispielsweise die Wortanzahl im Zyklus T_1 auf die Datenleitungen legen. Anstatt nun lediglich ein Wort zurückzugeben, liefert der Slave in jedem Zyklus ein Wort, bis die angegebene Anzahl erreicht ist. Abbildung 3.38 zeigt eine modifizierte Version von Abbildung 3.34, jetzt aber mit dem zusätzlichen Signal $\overline{\text{BLOCK}}$. Dieses Signal wird aktiviert, um eine Blockübertragung anzufordern. Bei diesem Beispiel dauert eine Blockleseoperation von vier Worten 6 statt 12 Zyklen.

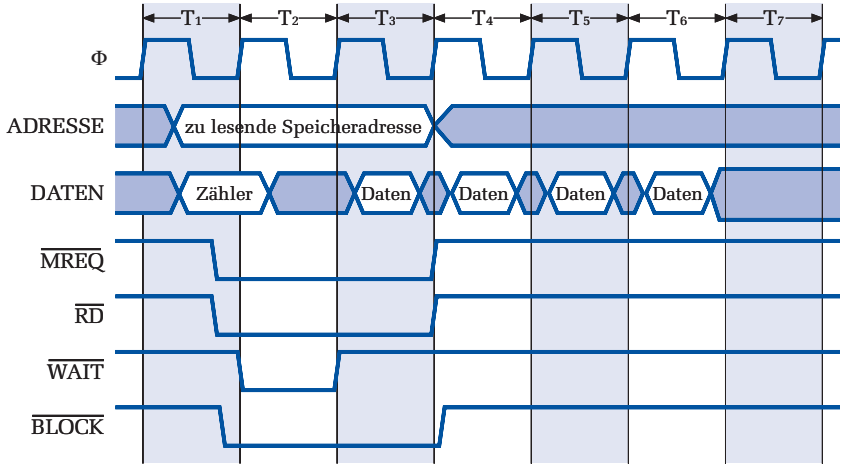


Abbildung 3.38: Eine Blockübertragung

Daneben gibt es noch weitere Arten von Buszyklen. Beispielsweise muss man bei einem Mehrprozessorsystem mit zwei oder mehr Prozessoren am selben Bus oftmals sicherstellen, dass jeweils nur eine CPU auf eine bestimmte kritische Datenstruktur im Speicher zugreift. Normalerweise verwendet man dazu eine Variable im Speicher, die 0 ist, wenn keine CPU auf die Datenstruktur zugreift, und 1, wenn sie verwendet wird. Möchte eine CPU auf die Datenstruktur zugreifen, muss sie die Variable lesen und auf 1 setzen, falls sie 0 ist. Im ungünstigsten Fall könnten zwei Prozessoren die Variable in aufeinander folgenden Buszyklen lesen. Finden beide den Wert 0 vor, setzt jede CPU die Variable auf 1 und nimmt an, dass sie die einzige CPU ist, die die Datenstruktur verwendet. Diese Sequenz von Ereignissen führt zum Chaos.

Um eine solche Situation zu verhindern, sind Mehrprozessorsysteme oft mit einem speziellen Buszyklus (Read-Modify-Write) ausgestattet, der jeder CPU erlaubt, ein Wort aus dem Speicher zu lesen, es zu inspizieren und zu modifizieren und es dann wieder in den Speicher zurückzuschreiben, ohne in dieser Zeit den Bus freizugeben. Diese Zyklusart verhindert, dass konkurrierende Prozessoren den Bus gleichzeitig nutzen und sich gegenseitig stören.

Ein anderer wichtiger Buszyklus betrifft die Interrupt-Behandlung. Wenn die CPU einem E/A-Gerät eine Aufgabe erteilt, erwartet sie gewöhnlich einen Interrupt, wenn die Aufgabe abgeschlossen ist. Die Signalisierung des Interrupts erfolgt über den Bus.

Nun kann es vorkommen, dass mehrere Geräte gleichzeitig einen Interrupt auslösen. Dabei stellen sich die gleichen Verwaltungsprobleme wie bei gewöhnlichen Buszyklen. Als übliche Lösung weist man den Geräten Prioritäten zu und verwendet einen zentralen Verwalter, um den zeitkritischsten Geräten den Vorrang zu gewähren. Hierfür sind Standard-Interrupt-Controller-Chips weit verbreitet. Der IBM-PC und alle seine Nachfolger benutzen den Intel-Chip 8259A (siehe Abbildung 3.39).

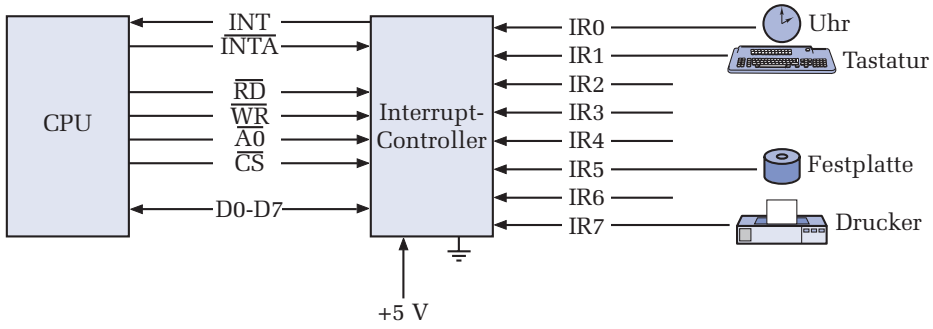


Abbildung 3.39: Einsatz des Interrupt-Controllers 8259A

An die 8 IRx-Eingänge (Interrupt Request) des 8259A lassen sich bis zu 8 E/A-Controllerchips direkt anschließen. Möchte eines dieser Geräte einen Interrupt auslösen, aktiviert es seine Eingangsleitung. Wenn einer oder mehrere Eingänge aktiviert sind, aktiviert der 8259A die Leitung `INT` (INTerrupt), die direkt das Interrupt-Pin der CPU steuert. Kann die CPU den Interrupt behandeln, sendet sie auf der Leitung `INTA` (INTerrupt Acknowledge) einen Impuls zurück an den 8259A. Jetzt muss der 8259A spezifizieren, welcher Eingang den Interrupt ausgelöst hat. Dazu legt er die Nummer dieses Eingangs auf den Datenbus. Diese Operation erfordert einen gesonderten Buszyklus. Die CPU-Hardware verwendet dann diese Nummer als Index auf eine Zeigertabelle (Pointer Table). Hier stehen die so genannten **Interrupt-Vektoren** – d.h. die Adressen der auszuführenden Prozeduren, die den jeweiligen Interrupt bedienen.

Der 8259A besitzt mehrere Register, die der Prozessor über gewöhnliche Buszyklen und die Signale $\overline{\text{RD}}$ (ReaD), $\overline{\text{WR}}$ (WRite), $\overline{\text{CS}}$ (Chip Select) und $\overline{\text{A0}}$ lesen und schreiben kann. Wenn das Programm den Interrupt abgearbeitet hat und bereit ist, den nächsten entgegenzunehmen, schreibt es einen speziellen Code in eines der 8259A-Register. Daraufhin deaktiviert der 8259A die `INT`-Leitung, sofern kein anderer Interrupt anhängig ist. Indem man bestimmte Werte in die Register schreibt, kann man die Interrupt-Modi des 8259A festlegen, eine bestimmte Gruppe von Interrupts ausmaskieren und andere Merkmale aktivieren.

Falls mehr als acht E/A-Geräte zu bedienen sind, kann man den 8259A kaskadieren. Im Extremfall lassen sich alle acht Eingänge mit den Ausgängen von weiteren acht 8259A-Chips verbinden, sodass bis zu 64 E/A-Geräte in einem zweistufigen Interrupt-Netz möglich sind. Verschiedene Pins des 8259A – die wir hier der Einfachheit halber weggelassen haben – sind dafür vorgesehen, die Kaskadierung zu steuern.

Damit ist das Thema „Busdesign“ keinesfalls erschöpft. Die obigen Erläuterungen bieten aber genügend Hintergrundinformationen, um zu verstehen, wie Busse funktionieren und wie Prozessoren und Busse zusammenarbeiten. Wir kommen nun vom Allgemeinen zum Speziellen und sehen uns einige Beispiele marktüblicher Prozessoren und ihrer Busse an.

3.5 Beispiele für CPU-Chips

Dieser Abschnitt beschäftigt sich recht ausführlich mit den Prozessoren Pentium 4, UltraSPARC III und 8051 auf der Hardwareebene.

3.5.1 Pentium 4

Der Pentium 4 ist ein direkter Nachfolger des 8088-Prozessors, der im ursprünglichen IBM-PC eingesetzt wurde. Intel hat den ersten Pentium 4 im November 2000 als Prozessor mit 42 Millionen Transistoren, einer Taktfrequenz von 1,5 GHz und einer Stegbreite von 0,18 µm eingeführt. Die Stegbreite gibt an, wie breit die Leiterzüge zwischen den Transistoren sind (und dient gleichzeitig als Maß für die Größe der Transistoren selbst). Je geringer die Stegbreite, desto mehr Transistoren lassen sich auf dem Chip unterbringen. Das Mooresche Gesetz liefert eine grundsätzliche Aussage über die Fähigkeit der Prozesstechniker, die Stegbreiten ständig zu verringern. Kleinere Stegbreiten lassen auch höhere Taktfrequenzen zu. Zum Vergleich: Die Dicke eines menschlichen Haares reicht von 20 µm bis 100 µm, wobei blondes Haar feiner als dunkles Haar ist.

Als Intel in den drei folgenden Jahren Erfahrungen mit dem Herstellungsprozess gesammelt hatte, kam ein verbesserter Typ mit 55 Millionen Transistoren, Geschwindigkeiten bis zu 3,2 GHz und Stegbreiten von 0,09 µm heraus. Obwohl sich der Pentium 4 weit von den 29.000 Transistoren des 8088 entfernt hat, ist er mit seinem Urahn voll abwärtskompatibel und kann 8088-Binärprogramme (ganz zu schweigen von Programmen, die für die dazwischenliegenden Modelle geschrieben wurden) unverändert ausführen.

Aus programmtechnischer Sicht ist der Pentium 4 ein echter 32-Bit-Prozessor. Dem Anwender bietet er die gleichen ISA-Merkmale wie die Prozessoren 80386, 80486, Pentium, Pentium II, Pentium Pro und Pentium III, einschließlich der gleichen Register, der gleichen Befehle und einer auf demselben Chip implementierten Gleitkommaeinheit nach dem Standard IEEE 754. Daneben gibt es einige neue Befehle, die hauptsächlich für Multimediaanwendungen vorgesehen sind.

Von der Hardware her gesehen ist der Pentium 4 teilweise ein 64-Bit-Prozessor, da er Daten in Einheiten von 64 Bit mit dem Speicher austauschen kann. Obwohl der Programmierer diese 64-Bit-Übertragungen nicht beobachten kann, machen sie den Rechner tatsächlich schneller als es bei einem echten 32-Bit-Prozessor der Fall wäre.

Auf der Mikroarchitekturebene unterscheidet sich der Pentium 4 radikal von allen seinen Vorgängern. Seine unmittelbaren Vorgänger – der Pentium II, der Pentium Pro und der Pentium III – bauen alle auf derselben internen Mikroarchitektur (namens P6) auf und unterscheiden sich nur in der Geschwindigkeit und ein paar unwesentlichen Details. Dagegen verwendet der Pentium 4 eine neue Mikroarchitektur (NetBurst genannt), die sich beträchtlich von der P6-Architektur unterschiedet. Sie hat eine tiefere Pipeline und zwei ALUs, die beide mit der doppelten Taktfrequenz arbeiten, um zwei Operationen je Taktzyklus zu ermöglichen. Zudem unterstützt der Pentium 4 mit zwei Registersätzen und einigen anderen internen Ressourcen das so genannte Hyperthreading, wodurch der Prozessor sehr schnell zwischen Programmen umschalten kann, als würde der Computer zwei physische CPUs enthalten. Auf die Mikroarchitektur geht *Kapitel 4* näher ein. Wie seine Vorgänger kann der Pentium 4 allerdings auch mehrere Befehle gleichzeitig ausführen, was ihn zu einem superskalaren Prozessor macht.

Einige Modelle des Pentium 4 besitzen einen zweistufigen und einige einen dreistufigen Cache. Bei allen Modellen ist ein Level-1-Cache (L1-Cache) mit einem 8-KB-SRAM auf dem Chip realisiert. Im Unterschied zum L1-Cache des Pentium III, der lediglich Rohbytes aus dem Speicher aufnimmt, geht der Pentium 4 weiter. Wenn der Prozessor Befehle aus dem Speicher abruft, konvertiert er sie in Mikrooperationen für die eigentliche Ausführung im RISC-Kern des Pentium 4. Der L1-Cache im Pentium 4 nimmt bis zu 12.000 dekodierte Mikrooperationen auf, sodass diese nicht wiederholt dekodiert werden müssen. Der Cache der zweiten Ebene hat eine Speicherkapazität von 256 KB in den älteren Modellen und bis zu 1 MB in den neueren. Der L2-Cache übernimmt die reinen Bytes aus dem Hauptspeicher, nichts wird dekodiert. Der Cache kann eine Mischung von Code und Daten aufnehmen. Außerdem besitzt die Pentium 4 Extreme Edition einen L3-Cache von 2 MB, um die Leistung noch weiter anzuheben.

Da alle Pentium 4-Chips mindestens zwei Cacheebenen besitzen, entsteht ein Problem bei Mehrprozessorsystemen, wenn eine CPU ein Wort in ihrem Cache geändert hat. Versucht eine andere CPU, dieses Wort aus dem Hauptspeicher zu lesen, erhält sie einen veralteten Wert, da modifizierte Cacheworte nicht unverzüglich in den Hauptspeicher zurückgeschrieben werden. Um die Speicherkonsistenz aufrechtzuerhalten, „**schnüffelt**“ jeder Prozessor in einem Multiprozessorsystem auf dem Speicherbus und sucht nach Verweisen auf Worte, die er in seinem lokalen Cache gespeichert hat (Snooping). Wenn er einen derartigen Verweis findet, greift er ein und stellt die angeforderten Daten bereit, bevor der Speicher eine Möglichkeit dazu erhält. Auf Snooping geht *Kapitel 8* ausführlich ein.

Pentium 4-Systeme verwenden zwei primäre externe Busse, die beide synchron arbeiten. Der Speicherbus wird benutzt, um auf den Haupt-(S)DRAM zuzugreifen, und der PCI-Bus, um mit den E/A-Geräten zu kommunizieren. Manchmal hängt man an den PCI-Bus noch einen Legacy-Bus (d.h. einen veralteten Bus) an, damit auch ältere Peripheriegeräte noch eingesetzt werden können.

Ein wesentlicher Unterschied zwischen dem Pentium 4 und allen seinen Vorgängern betrifft die Gehäusegestaltung. Ein Problem bei allen modernen Chips ist ihre Leistungsaufnahme und die dadurch produzierte Wärme. Der Pentium 4 verbraucht zwischen 63 und 82 W je nach Taktfrequenz. Folglich sucht Intel beständig nach Wegen, um die von den CPU-Chips erzeugte Wärme in den Griff zu bekommen. Das Gehäuse des Pentium 4 ist quadratisch mit einer Kantenlänge von 35 mm. An seiner Unterseite befinden sich 478 Pins, von denen 85 für die Betriebsspannung vorgesehen sind und 180 auf Masse liegen, um Störungen zu unterbinden. Die Pins sind in einem Raster von 26 × 26 angeordnet, wobei die mittleren 14 × 14 Pins ausgespart sind. Zudem fehlen zwei Pins in einer Ecke, damit der Chip nicht versehentlich mit falscher Orientierung in den Sockel gesteckt werden kann. Abbildung 3.40 zeigt die physische Pinanordnung.

Zum Lieferumfang des Chips gehört eine Montageklammer für einen Kühlkörper, um die Wärme zu verteilen, und ein Lüfter, der den Chip kühlt. Um sich ein Bild davon zu machen, wo das Problem liegt, schalten Sie am besten eine 60-W-Glühbirne ein und umschließen sie mit Ihren Händen (aber nicht berühren!). So viel Wärme muss ständig abgeführt werden. Und wenn ein Pentium 4 als CPU ausgedient hat, kann er immer noch als Campingkocher fungieren.

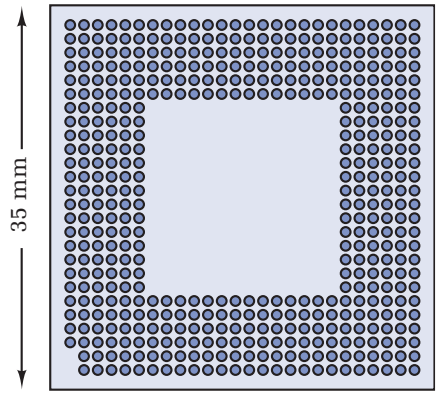


Abbildung 3.40: Physische Pinanordnung beim Pentium 4

Nach den Gesetzen der Physik muss alles, was eine Menge Hitze abgibt, eine Menge Energie aufnehmen. In einem tragbaren Computer mit einer begrenzten Batterieladung ist ein hoher Energieverbrauch erst recht nicht erwünscht. Um dieses Problem zu lösen, hat Intel die CPU so entworfen, dass sie schlummert, wenn sie im Leerlauf arbeitet, und in einen Tiefschlaf fällt, wenn der Leerlaufzustand höchstwahrscheinlich noch eine ganze Zeit anhält. Es gibt fünf Zustände, die von „vollständig aktiv“ bis zum Tiefschlaf reichen. In den Zwischenzuständen werden bestimmte Funktionen aktiviert (beispielsweise das Cache-Schnüffeln und die Interrupt-Behandlung), während andere Funktionen deaktiviert werden. Im Tiefschlaf behält der Prozessor alle Cache- und Registerwerte bei, schaltet aber den Takt und alle internen Einheiten ab. Im Tiefschlafzustand ist ein Hardwaresignal erforderlich, um den Prozessor aufzuwecken. Es ist nicht bekannt, ob ein Pentium 4 träumen kann, wenn er sich im Tiefschlaf befindet.

Die logische Anschlussbelegung des Pentium 4

Die 478 Pins des Pentium 4 sind mit 198 Signalen, 85 Stromversorgungsanschlüssen (mit mehreren verschiedenen Spannungen), 180 Masseverbindungen und 15 Reserveanschlüssen für künftige Verwendung belegt. Einige logische Signale verwenden zwei oder mehr Pins (beispielsweise die Speicheranforderungsadresse), sodass es nur 56 unterschiedliche Signale gibt. Abbildung 3.41 gibt eine vereinfachte logische Anschlussbelegung an. Auf der linken Seite der Abbildung befinden sich fünf Hauptgruppen der Speicherbussignale. Die durchgängig in Großbuchstaben angegebenen Bezeichnungen entsprechen den Intel-Signalnamen. Die Bezeichnungen in gemischter Schreibweise sind Sammelnamen für mehrere zusammengehörende Signale.

Intel verwendet eine Namenskonvention, die man kennen sollte. Da alle Chips heutzutage mit dem Computer entworfen werden, muss man die Signalnamen als ASCII-Text darstellen können. Da es zu schwierig ist, die Low-aktiven Signale mit Überstrichen zu versehen, schreibt Intel stattdessen das Nummernzeichen (#) nach dem Signalnamen. Somit wird $\overline{\text{BPRI}}$ als BPRI# dargestellt. Wie die Abbildung zeigt, sind die meisten Pentium 4-Signale Low-aktiv.

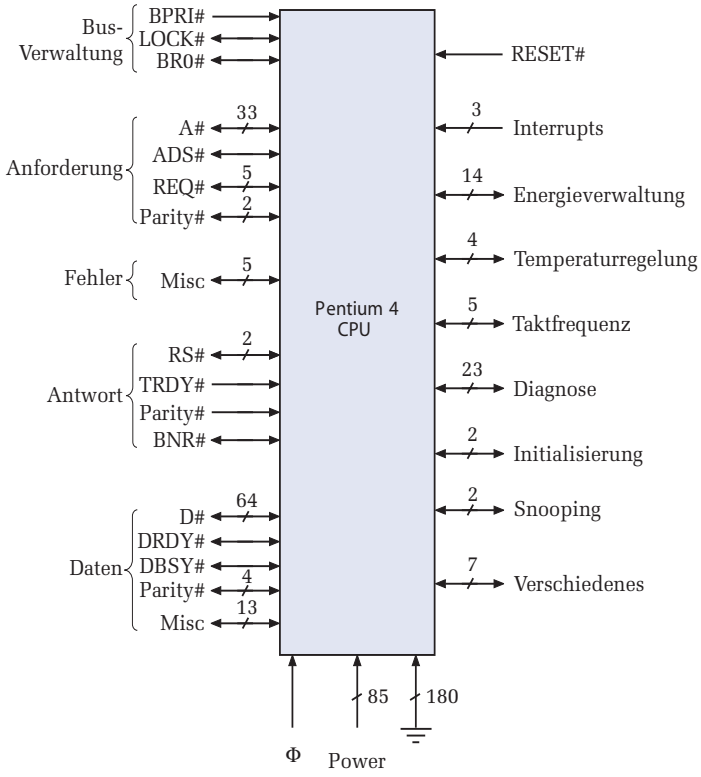


Abbildung 3.41: Logische Anschlussbelegung des Pentium 4. Die großgeschriebenen Namen sind die offiziellen Intel-Bezeichnungen für die verschiedenen Signale. Namen in gemischter Schreibweise sind Gruppen zusammengehörender Signale oder Signalbeschreibungen.

Wir sehen uns nun die Signale im Einzelnen an und beginnen mit den Bussignalen. Die erste Signalgruppe dient der Busanforderung (d.h. für die Bus-Verwaltung). BR0# fordert den Bus an. BPRI# erlaubt einem Gerät, eine Anforderung mit hoher Priorität auszulösen, die den Vorrang vor einer normalen Priorität hat. Mit LOCK# kann eine CPU den Bus sperren und damit verhindern, dass andere Geräte den Bus erhalten, bis die CPU fertig ist.

Hat der Prozessor oder ein anderer Busmaster den Bus erhalten, kann er mithilfe der nächsten Signalgruppe eine Anforderung über den Bus auslösen. Adressen sind 36 Bit breit. Allerdings müssen die niederwertigen 3 Bits immer 0 sein, sodass für sie keine Pins vorgesehen sind – A# umfasst also nur 33 Pins. Alle Übertragungen erfolgen mit 8 Byte, die an einer 8-Byte-Grenze ausgerichtet sind. Die 36 Adressbits erlauben einen maximal adressierbaren Speicher von 2^{36} Byte oder 64 GB.

Wird eine Adresse auf einen Bus gelegt, wird das Signal ADS# aktiviert, um dem Ziel (z.B. dem Speicher) mitzuteilen, dass die Adressleitungen gültig sind. Der Typ des Buszyklus (z.B. lies ein Wort oder schreibe einen Block) wird auf den REQ#-Leitungen übermittelt. Die beiden Paritätsleitungen schützen A# und REQ#.

Die fünf Fehlerleitungen dienen der Meldung von Gleitkommafehlern, internen Fehlern, Fehlern bei der Prüfung des Computers (d.h. der Hardware) und bestimmten anderen Fehlern.

Die Antwortgruppe enthält Signale, mit denen der Slave Rückmeldungen an den Master schickt. RS# enthält den Statuscode. TRDY# zeigt an, dass der Slave (das Ziel) bereit ist, Daten vom Master entgegenzunehmen. Auch diese Signale unterliegen einer Paritätsprüfung. BNR aktiviert einen Wartezustand, wenn das adressierte Ziel nicht rechtzeitig reagieren kann.

Die letzte Busgruppe dient der eigentlichen Datenübertragung. Mit D# werden 8 Datenbytes auf den Bus gelegt. Wenn sie auf dem Bus liegen, wird DRDY# aktiviert, um ihre Anwesenheit anzukündigen. DBSY# teilt der Außenwelt mit, dass der Bus momentan beschäftigt ist. Die Daten werden ebenfalls einer Paritätsprüfung unterzogen. Die Gruppe der verschiedenen Signale hat mit der Übernahme von Werten und Ähnlichem zu tun.

Über RESET# lässt sich der Prozessor bei einem Rechnerabsturz zurücksetzen, beispielsweise durch die RESET-Taste am Computer.

Der Pentium 4 lässt sich so konfigurieren, dass er die Interrupts genau wie der 8088 (zum Zwecke der Abwärtskompatibilität) behandelt. Er kann aber auch ein neues Interrupt-System mit einem so genannten **APIC** (**Advanced Programmable Interrupt Controller**) verwenden.

Der Pentium 4 lässt sich mit einer beliebigen der vordefinierten Spannungen betreiben, er muss aber wissen, welche Spannung anliegt. Über die Signale für die Betriebsspannungsverwaltung ist es möglich, die Betriebsspannung automatisch auszuwählen, der CPU mitzuteilen, dass die Betriebsspannung stabil anliegt, und andere mit der Stromversorgung im Zusammenhang stehende Aufgaben zu realisieren. Auch die verschiedenen Schlafzustände werden über diese Signale gesteuert, da das zum Bereich der Energieverwaltung gehört.

Trotz ausgeklügelter Energieverwaltung kann der Pentium 4 sehr heiß werden. Deshalb sind Signale für die Temperaturregelung vorgesehen. Die CPU kann damit ihrer Umgebung mitteilen, dass eine Gefahr der Überhitzung besteht. Eines der Pins wird von der CPU aktiviert, wenn ihre interne Temperatur 130 °C erreicht. Sollte eine CPU jemals bei dieser Temperatur ankommen, denkt sie wahrscheinlich ohnehin an ihre Pensionierung und an ein Leben als Campingkocher.

Über die Taktfrequenzgruppe wird die Frequenz des Systembusses bestimmt. Die Diagnosegruppe umfasst Signale für das Testen und Debuggen von Systemen entsprechend dem JTAG-Standard nach IEEE 1149.1. Die Initialisierungsgruppe ist für das Booten (Starten) des Systems vorgesehen.

Schließlich bildet die Gruppe „Verschiedenes“ ein Sammelsurium von Signalen, die spezielle Aufgaben haben, beispielsweise anzuzeigen, dass der CPU-Sockel belegt ist, oder dass ein 8088 emuliert wird.

Fließbandverarbeitung auf dem Speicherbus des Pentium 4

Moderne CPUs wie der Pentium 4 sind viel schneller als moderne DRAM-Speicher. Um die CPU mangels Daten vor dem Hungertod zu retten, muss man aus dem Speicher den höchstmöglichen Durchsatz herausholen. Aus diesem Grund ist der Speicherbus des Pentium 4 in hohem Maße auf Fließbandverarbeitung (Pipelining) ausgerichtet, wobei bis zu acht Bustransaktionen gleichzeitig stattfinden. Das Konzept der Fließbandverarbeitung hat Kapitel 2 im Zusammenhang mit einer Pipeline-CPU (siehe *Abbildung 2.3*) demonstriert, es lässt sich aber auch auf den Speicher anwenden.

Um die Fließbandverarbeitung zu ermöglichen, laufen Speicheranforderungen beim Pentium 4 – so genannte **Transaktionen** – in sechs Phasen ab (in Klammern sind die Bezeichnungen für Abbildung 3.42 angegeben):

1. Bus-Verwaltung
2. Anforderung (`Req`)
3. Fehlermeldung (`Error`)
4. Snooping (`Snoop`)
5. Antwort (`Resp`)
6. Daten (`Data`)

Nicht alle Phasen sind bei allen Transaktionen erforderlich. Die Bus-Verwaltungsphase bestimmt, welcher der potenziellen Busmaster als Nächster an die Reihe kommt. In der Anforderungsphase kann die Adresse auf den Bus gelegt und die Anforderung ausgeführt werden. In der Fehlermeldungsphase kann der Slave ankündigen, dass die Adresse einen Paritätsfehler aufweist oder etwas anderes schief gegangen ist. Die Snooping-Phase erlaubt einer CPU, bei einer anderen CPU zu „schnüffeln" – eine Eigenschaft, die nur in einem Multiprozessorsystem benötigt wird. In der Antwortphase erfährt der Master, ob er die angeforderten Daten erhält. Schließlich lassen sich in der Datenphase die Daten an die CPU senden, die sie angefordert hat.

Das Geheimnis des Pipeline-Speicherbusses des Pentium 4 ist, dass jede Phase unterschiedliche Bussignale nutzt, sodass alle Phasen untereinander vollkommen unabhängig sind. Die sechs benötigten Signalgruppen sind in Abbildung 3.41 links dargestellt. Beispielsweise kann eine CPU mithilfe der Verwaltungssignale versuchen, den Bus zu erhalten. Nachdem ihr der Bus zugeteilt wurde, gibt sie diese Busleitungen frei und geht zu den Leitungen der Anfragegruppe über. Unterdessen kann die andere CPU oder ein E/A-Gerät in die Bus-Verwaltungsphase eintreten usw. Abbildung 3.42 zeigt, wie mehrere Bustransaktionen gleichzeitig ausgeführt werden können.

In Abbildung 3.42 ist die Bus-Verwaltungsphase nicht dargestellt, weil sie nicht immer erforderlich ist. Wenn zum Beispiel der aktuelle Busbesitzer (oftmals der Prozessor) eine andere Transaktion ausführen möchte, muss er den Bus nicht noch einmal übernehmen. Er muss nur dann den Bus wieder anfordern, wenn er den Busbesitz an ein anderes anforderndes Gerät abgegeben hat. Die Transaktionen 1 und 2 laufen unkompliziert ab: fünf Phasen in fünf Buszyklen. Transaktion 3 führt eine längere Datenphase ein, beispielsweise, weil es sich um einen Blocktransfer handelt oder der adressierte Speicher einen Wartezustand eingefügt hat. Transaktion 4 kann deshalb ihre Datenphase nicht beginnen, wann sie möchte. Sie beobachtet, dass das Signal `DBSY#` immer noch aktiviert ist, und wartet, bis es deaktiviert wird. In Transaktion 5 ist zu sehen, dass auch die Antwortphase mehrere Buszyklen umfassen kann, wodurch sich Transaktion 6 verzögert. Transaktion 7 macht deutlich, dass eine „Blase", nachdem sie einmal in einer Transaktion entstanden ist, an dieser Position immer mitgeschleppt wird, wenn neue Transaktionen fortlaufend starten. In realen Programmen ist es allerdings unwahrscheinlich, dass die CPU in jedem einzelnen Buszyklus eine neue Transaktion startet, sodass die Blasen auch keinen langen Bestand haben.

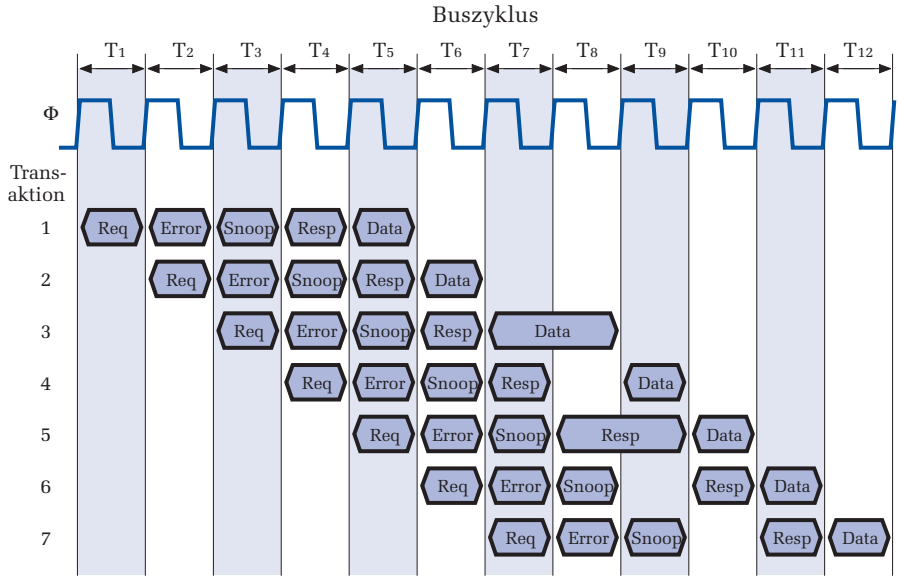


Abbildung 3.42: Fließbandverarbeitung von Anfragen auf dem Speicherbus des Pentium 4

3.5.2 UltraSPARC III

Als zweites Beispiel eines CPU-Chips untersuchen wir nun die UltraSPARC-Familie von Sun. Dabei handelt es sich um Suns Linie der 64-Bit-SPARC-CPUs. Diese Prozessoren sind vollständig konform mit der SPARC-Architektur Version 9, die auch für 64-Bit-CPUs ausgelegt ist. Man findet sie in Workstations und Servern von Sun sowie in vielen anderen Anwendungen. Zu dieser Familie gehören UltraSPARC I, UltraSPARC II und UltraSPARC III, die sich von der Architektur her sehr ähneln und hauptsächlich durch Einführungsdatum, Taktgeschwindigkeit und einige mit jeder Version eingeführte Zusatzbefehle unterscheiden. Wir beziehen uns in diesem Buch hauptsächlich auf UltraSPARC III, wobei aber die architektonischen (d.h. die von der Technologie unabhängigen) Teile auch auf andere UltraSPARC-Modelle zutreffen.

UltraSPARC III ist ein herkömmlicher RISC-Prozessor und vollständig binärkompatibel mit der 32-Bit-SPARC-Architektur V8. Er kann die für 32-Bit-SPARC V8 geschriebenen Binärprogramme ohne Modifikation ausführen, weil die SPARC-Architektur V9 zur SPARC-Architektur V8 abwärtskompatibel ist. Der einzige Punkt, an dem der UltraSPARC III von der SPARC-V9-Architektur abweicht, betrifft den zusätzlichen VIS 2.0-Befehlssatz, der für 3D-Grafikanwendungen, MPEG-Dekodierung in Echtzeit, Datenkomprimierungen, Signalverarbeitung, Ausführung von Java-Programmen und Netzwerk-Aufgaben vorgesehen ist.

Obwohl der UltraSPARC III auch in Workstations zu finden ist, wurde der Prozessor ursprünglich für das Kerngeschäft von Sun konzipiert, nämlich große Mehrprozessorserver mit gemeinsam genutztem Speicher, die im Internet und firmeneigenen Intranets eingesetzt werden. Insbesondere ist ein großer Teil des für den Aufbau eines Mehrprozessorsystems erforderlichen „Klebstoffs“ auf jedem UltraSPARC III-Chip vorhanden, womit sich viele Prozessoren relativ einfach miteinander verbinden lassen.

Der erste UltraSPARC III-Prozessor wurde 2000 mit einer Taktfrequenz von 600 MHz eingeführt. Er basiert auf einer 0,18 µm-Technologie mit Leiterbahnen aus Aluminium. Die Chips enthalten 29 Millionen Transistoren. Da die Anzahl der von Sun benötigten Prozessoren zu gering ist, um den Bau einer eigenen modernen Chipfabrik zu rechtfertigen, konzentriert sich das Unternehmen vorzugsweise auf Chipentwurf und Software. Die Chipproduktion wurde per Vertrag zu großen Chipherstellern ausgelagert. Im Fall von UltraSPARC III stellt Texas Instruments (TI) die Chips her. Im Jahre 2001 hat TI die Technologie verbessert und begonnen, 900-MHz-Chips mit einer Stegbreite von 0,15 µm und Kupfer anstelle von Aluminium für die Leiterbahnen zu produzieren. Die Stegbreite konnte 2002 auf 0,13 µm gesenkt und die Taktfrequenz auf 1,2 GHz angehoben werden. Der Leistungsverbrauch von 50 W bringt aber die gleichen Probleme der Wärmeableitung wie beim Pentium 4 mit sich.

Es ist schwierig, einen CISC-Chip (wie den Pentium 4) mit einem RISC-Chip (wie dem UltraSPARC III) allein anhand der Taktfrequenz zu vergleichen. Zum Beispiel kann der UltraSPARC III fortwährend vier Befehle je Taktzyklus initiieren, was fast die gleiche Ausführungsrate bringt wie eine mit 4,8 GHz laufende CPU, die jeweils nur einen Befehl ausgibt. Außerdem verfügt der UltraSPARC III über sechs interne Pipelines – zwei 14-stufige Pipelines für Ganzzahloperationen, zwei für Gleitkommaoperationen, eine für Lade-/Speicheroperationen und eine für Verzweigungen. Weitere Leistungsverbesserungen ergeben sich durch ein anderes Cachekonzept, breitere Busse und andere Faktoren. Allerdings kann auch der Pentium 4 auf besondere Stärken verweisen. Wenn man zwei so grundsätzlich verschiedene Chips lediglich nach der Taktfrequenz vergleicht, lässt sich letztlich kaum etwas über die relative Leistung bei einer bestimmten Aufgabe sagen.

Der UltraSPARC III wird mit einem 1368-Pin-LGA (Land Grid Array) geliefert, das Abbildung 3.43 zeigt. Dieses Gehäuse besteht aus einem quadratischen Feld von 37 × 37 = 1369 Pins an der Unterseite des Chips, wobei das Pin in der linken unteren Ecke fehlt. Der Sockel ist entsprechend gestaltet und verhindert, dass man den Chip versehentlich falsch einsetzt.

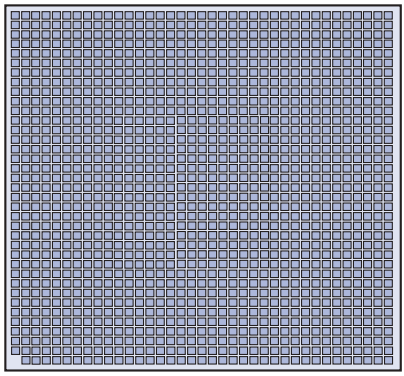


Abbildung 3.43: Der CPU-Chip UltraSPARC III

Der UltraSPARC III besitzt zwei interne L1-Caches: 32 KB für Befehle und 64 KB für Daten. Außerdem gibt es einen 2-KB-Prefetch-Cache und einen 2-KB-Schreib-Cache, der Schreiboperationen in den L2-Cache sammelt. Dadurch lassen sich Schreiboperationen in größeren Blöcken ausführen, was eine bessere Bandbreitennutzung bedeutet. Wie

beim Pentium 4 gibt es auch einen vom Chip unabhängigen L2-Cache, der aber im Unterschied zum Pentium 4 nicht auf demselben Chip unterbracht ist. Der Cachecontroller und die Logik für die Lokalisierung der Cacheblöcke sind zwar auf dem Chip realisiert, der eigentliche SRAM-Speicher jedoch nicht. Stattdessen können die Systemdesigner für den L2-Cache beliebige kommerziell verfügbare Cachechips einsetzen.

Die Entscheidung, den Level-2-Cache beim Pentium 4 zu integrieren und bei der UltraSPARC abzutrennen, beruht teilweise auf technischen Gründen und teilweise auf den unterschiedlichen Geschäftsmodellen von Intel und Sun. Vom technischen Standpunkt aus ist ein externer Cache größer und flexibler (der L2-Cache des UltraSPARC III kann von 1 MB bis 8 MB reichen, während der L2-Cache des Pentium 4 auf 512 KB festgelegt ist). Allerdings kann er auch langsamer sein, weil er weiter von der CPU entfernt ist. Zudem sind mehr Signale von der CPU nach außen zu führen, um den Cache zu adressieren. Vor allem ist die Verbindung zwischen dem UltraSPARC III und seinem L2-Cache 256 Bit breit, wodurch sich ein kompletter 32-Byte-Cacheblock in einem Zyklus übertragen lässt.

Aus kommerzieller Sicht ist Intel ein Halbleiterhersteller, der seinen eigenen Level-2-Cache-Chip entwickeln, herstellen und über eine herstellerspezifische Hochleistungsschnittstelle mit der CPU verbinden kann. Sun ist demgegenüber ein Computeranbieter und kein Chiphersteller. Das Unternehmen entwickelt einen Teil seiner eigenen Chips (z.B. den UltraSPARC), lässt sie aber von Halbleiterherstellern produzieren. Sun greift nach Möglichkeit auf kommerziell erhältliche Chips zurück, die sich bereits auf dem Markt behauptet haben. Die für den Level-2-Cache eingesetzten SRAMs sind von zahlreichen Chipanbietern beziehbar, sodass für Sun keine Notwendigkeit einer Eigenentwicklung bestand. Diese Entscheidung impliziert, dass man den Level-2-Cache vom CPU-Chip unabhängig macht.

Der UltraSPARC III verwendet einen Adressbus mit 43 Bit Breite, der bis zu 8 TB Hauptspeicher adressieren kann. Der Datenbus ist 128 Bit breit und kann somit 16 Byte auf einmal zwischen CPU und Speicher übertragen. Der Bustakt von 150 MHz erlaubt eine Speicherbandbreite von 2,4 GB/s, die wesentlich höher liegt als die 528 MB/s des PCI-Busses.

Um (mehrere) UltraSPARC-CPUs miteinander zu verbinden und deren Kommunikation zu ermöglichen, hat Sun die **UPA** (**Ultra Port Architecture**) entwickelt. Die UPA lässt sich als Bus, als Switch oder als Kombination dieser beiden Formen implementieren. Unterschiedliche Workstation- und Servermodelle arbeiten mit verschiedenen UPA-Implementierungen. Auf die CPU wirkt sich die UPA-Implementierung nicht aus, weil die Schnittstelle zur UPA genau definiert ist und der CPU-Chip genau diese Schnittstelle unterstützen muss (und natürlich unterstützt).

Abbildung 3.44 zeigt den Kern eines UltraSPARC III-Systems mit CPU-Chip, UPA-Schnittstelle und L2-Cache (zwei handelsüblichen SRAMs). Außerdem ist in der Abbildung ein UDB II-Chip (UltraSPARC Data Buffer II) zu sehen, dessen Funktion in Kürze erläutert wird. Benötigt die CPU ein Speicherwort, sucht sie zuerst in einem ihrer (internen) L1-Caches danach. Kann sie das Wort finden, fährt sie in voller Geschwindigkeit mit der Ausführung fort. Steht das Wort nicht im L1-Cache, sucht die CPU im L2-Cache.

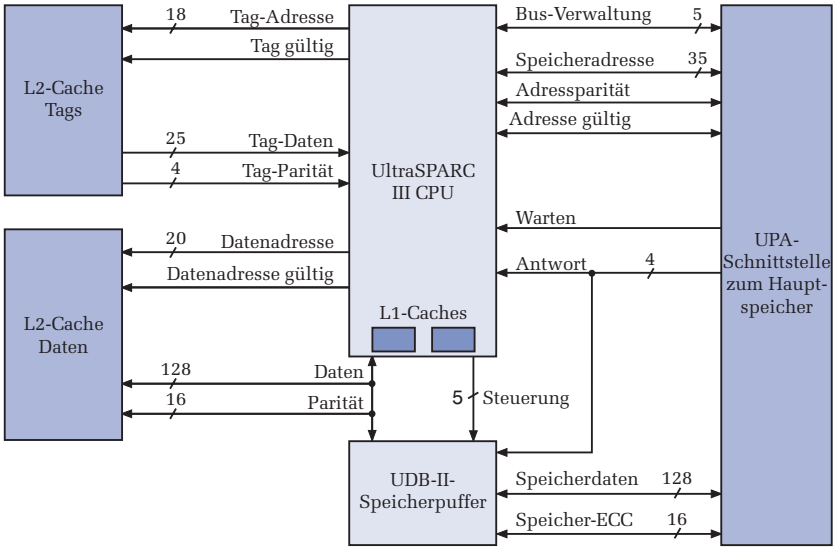


Abbildung 3.44: Die Hauptmerkmale des Kerns eines UltraSPARC III-Systems

Auch wenn Kapitel 4 ausführlich auf das Caching eingeht, sind hier einige Worte angebracht. Der gesamte Hauptspeicher ist in Cachezeilen (Blöcke) zu je 64 Byte unterteilt. Die 256 am häufigsten benutzten Befehlszeilen und die 256 am häufigsten benutzten Datenzeilen befinden sich im Level-1-Cache. Häufig benutzte Cachezeilen, die nicht in den Level-1-Cache passen, kommen in den Level-2-Cache. Dieser Cache enthält Daten- und Befehlszeilen zufällig gemischt. In Abbildung 3.44 sind sie im Kästchen mit der Beschriftung „L2-Cache Daten“ zu finden. Das System muss verfolgen, welche Zeilen im L2-Cache enthalten sind. Diese Informationen stehen in einem zweiten SRAM mit der Bezeichnung „L2-Cache Tags“.

Schlägt eine Suche im Level-1-Cache fehl, schickt die CPU den Bezeichner (Identifier) der gesuchten Zeile (Tag-Adresse) an den Level-2-Cache. Aus der Antwort (den Tag-Daten) kann die CPU erkennen, ob die Zeile im Level-2-Cache vorhanden ist und erfährt in diesem Fall auch den Zustand der Zeile. Ist die Zeile im Cache vorhanden, holt sich die CPU diese Zeile. Die Datenübertragungen erfolgen mit einer Breite von 16 Byte, sodass vier Zyklen erforderlich sind, um eine komplette Zeile in den L1-Cache zu übernehmen.

Befindet sich die Cachezeile nicht im Level-2-Cache, muss sie über die UPA-Schnittstelle aus dem Hauptspeicher abgerufen werden. Die UPA des UltraSPARC III ist mit einem zentralen Controller implementiert. An ihn gehen die Adress- und Steuersignale von der CPU (bzw. aller CPUs, falls mehrere vorhanden sind). Für den Speicherzugriff muss sich die CPU zuerst die Berechtigung über die Bus-Verwaltung einholen. Nachdem die Berechtigung erteilt wurde, gibt die CPU die Speicheradresse aus, legt den Anforderungstyp fest und aktiviert die Leitung „Adresse gültig“. (Diese Leitungen arbeiten bidirektional, da andere CPUs in einem UltraSPARC III-Multiprozessorsystem auf entfernte Caches zugreifen müssen, um alle Caches konsistent zu halten.) Die Adresse und der Buszyklustyp werden in zwei Zyklen über die Adresspins ausgegeben – im ersten Zyklus die Zeile und im zweiten Zyklus die Spalte, wie in Abbildung 3.29 gezeigt.

Während die CPU auf die Ergebnisse wartet, kann sie durchaus mit anderen Arbeiten fortfahren. Beispielsweise sperrt ein Cachefehler beim Prefetching eines Befehls nicht die Ausführung eines oder mehrerer bereits geladener Befehle. Somit können mehrere Transaktionen zur UPA gleichzeitig in Arbeit sein. Die UPA kann zwei unabhängige Transaktionsströme (normalerweise Lese- und Schreibvorgänge) behandeln, bei denen jeweils mehrere Transaktionen ausstehen können. Es liegt beim zentralisierten Controller, diese Abläufe zu verfolgen und die tatsächlichen Speicheranforderungen in der effizientesten Reihenfolge vorzunehmen.

Wenn die Daten schließlich aus dem Speicher eintreffen, können sie gleichzeitig mit 8 Byte ankommen und einen 16-Bit-Fehlerkorrekturcode enthalten, um die Zuverlässigkeit zu verbessern. Eine Transaktion kann einen gesamten Cacheblock, ein Quadwort (8 Byte) oder sogar weniger Bytes anfordern. Alle ankommenden Daten gelangen in den UDB, der sie puffert. Der UDB soll die CPU noch weiter vom Speichersystem abkoppeln, sodass sie asynchron arbeiten können. Wenn zum Beispiel die CPU ein Wort oder eine Cachezeile in den Speicher schreiben muss, braucht sie nicht zu warten, bis sie auf die UPA zugreifen kann, sondern schreibt die Daten unverzüglich in den UDB und überlässt es diesem, die Daten später in den Speicher zu übertragen. Der UDB erzeugt außerdem den Fehlerkorrekturcode und prüft ihn. Es sei noch einmal darauf hingewiesen, dass die obige Beschreibung des UltraSPARC III genau wie vorher beim Pentium 4 sehr vereinfacht ist, aber die wesentlichen Aspekte der Arbeitsweise enthält.

3.5.3 Der 8051

Pentium 4 und UltraSPARC III sind Beispiele für Hochleistungsprozessoren, die für sehr schnelle PCs und Server konzipiert sind. An derartige Systeme denkt man in der Regel zuerst, wenn von Computern die Rede ist. Daneben gibt es aber noch eine ganze Welt von Computern, die tatsächlich viel größer ist: eingebettete Systeme. Dieser Abschnitt vermittelt einen kurzen Einblick in diese Welt.

Wahrscheinlich ist es nur leicht übertrieben zu sagen, dass jedes elektrische Gerät, das mehr als 100 Dollar kostet, einen Computer enthält. Sicherlich sind Fernseher, Mobiltelefone, PDAs, Mikrowellenherde, Camcorder, Videorecorder, Laserdrucker, Einbruchsicherungen, Hörgeräte, elektronische Spiele und andere Geräte, die man gar nicht alle aufzählen kann, heutzutage computergesteuert. Die in diese Dinge integrierten Computer sind eher auf geringe Kosten als auf hohe Leistung optimiert, was zu anderen Kompromissen als bei den bisher untersuchten CPUs des oberen Leistungsbereichs führt.

Wie bereits in *Kapitel 1* erwähnt, ist der 8051 der wahrscheinlich bekannteste Mikrocontroller, der derzeit eingesetzt wird, was vor allem auf seine geringen Kosten zurückzuführen ist. Zudem ist es ein einfacher Chip, der einfache und preiswerte Schnittstellen erlaubt. Deshalb untersuchen wir nun den 8051-Chip, dessen physische Anschlussbelegung in Abbildung 3.45 zu sehen ist.

Neben dem in Abbildung 3.45 gezeigten 40-Pin-Standardgehäuse gibt es noch andere Gehäuseformen. Der 8051 hat 16 Adressleitungen und kann damit bis zu 64 KB Speicher adressieren. Der Datenbus ist 8 Bit breit, sodass Datenübertragungen zwischen der CPU und dem Speicher mit jeweils einem Byte erfolgen (gegenüber 8 Byte gleichzeitig beim Pentium 4 und 16 Byte gleichzeitig beim UltraSPARC III). Der Prozessor hat eine ganze Reihe von Steuerleitungen, auf die wir gleich eingehen. Den größten Kontrast zu den „reinrassigen“ CPUs Pentium 4 und UltraSPARC bilden aber die 32 E/A-Leitungen, die in vier Gruppen zu je 8 Bit angeordnet sind.

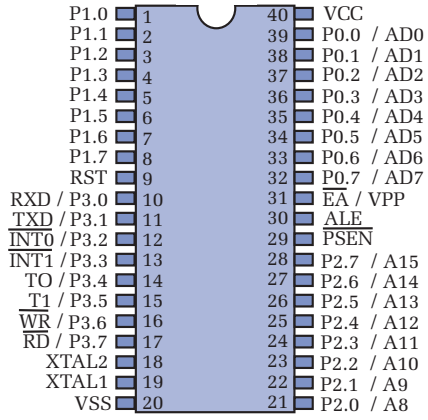


Abbildung 3.45: Physische Anschlussbelegung des 8051

An diese E/A-Leitungen lassen sich Tasten, Schalter, Leuchtdioden (Light Emitting Diode – LED) oder andere Bauelemente anschließen, um dem 8051 Eingaben anzubieten oder Ausgaben vom 8051 weiterzuverarbeiten. Auf diese Weise lassen sich die meisten – wenn nicht sogar alle – Funktionen eines Uhrenradios per Software steuern, wodurch man auf kostspielige diskrete Logik verzichten kann.

Abbildung 3.46 zeigt die logische Anschlussbelegung des 8051. Der Standardtyp 8051 enthält einen internen ROM von 4 KB (beim 8052 sind es 8 KB). Sollte das für eine Anwendung nicht genügen, kann man bis zu 64 KB externen Speicher über einen Bus an den 8051 anschließen. Die ersten sieben Signale auf der linken Seite von Abbildung 3.46 stellen die Schnittstelle zum externen Speicher her. Die mit A bezeichneten 16 Adressleitungen adressieren das Byte im externen Speicher, das der Mikrocontroller liest oder schreibt. Die acht D-Leitungen realisieren den Datentransport. Um die Anzahl der Pins gering zu halten, werden die niederwertigen acht Adressleitungen im Multiplexbetrieb über dieselben Pins wie die Datenleitungen ausgegeben. Bei einer Bustransaktion liefern diese Pins die Adresse im ersten Taktzyklus und übertragen die Daten in darauf folgenden Zyklen.

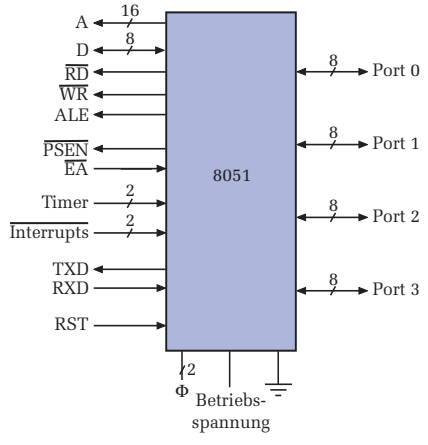


Abbildung 3.46: Logische Anschlussbelegung des 8051

Beim Einsatz von externem Speicher zeigt der 8051 über die Leitungen $\overline{RD}$ und $\overline{WR}$ an, ob er den Speicher liest bzw. schreibt. Die CPU zeigt mit dem Signal ALE (Address Latch Enable) an, dass die Adresse gültig ist. Externe Speicherchips übernehmen auf dieses Signal hin die auf den Datenleitungen ausgegebene Adresse, da die CPU die Adresse im nächsten Zyklus wieder abschaltet, um dieselben Pins für die Daten zu nutzen.

Die Signale $\overline{PSEN}$ und $\overline{EA}$ beziehen sich ebenfalls auf den externen Speicher. Das Signal PSEN (Program Store ENable) wird aktiviert, um anzuzeigen, dass der 8051 aus dem Programmspeicher lesen möchte. Normalerweise verbindet man diese Leitung mit dem OE-Eingang des Speichers, wie es Abbildung 3.27 gezeigt hat.

Das Signal $\overline{EA}$ (External Access) wird normalerweise auf ein festes Potential gelegt. Wenn es mit High verbunden ist, wird der interne 4-KB-Speicher (8 KB beim 8052) für die Adressen im Bereich des internen Speichers und der externe Speicher für Adressen oberhalb von 4 KB (bzw. 8 KB beim 8052) verwendet. Liegt das Signal auf Low, wird der externe Speicher für alle Adressen verwendet und der Onchip-Speicher wird praktisch umgangen. Bei den Chips 8031 und 8032 aus dieser Mikrocontrollerfamilie muss das Signal $\overline{EA}$ fest mit Low verbunden sein, da diese Chips keinen Onchip-Speicher besitzen.

Die beiden Timer-Leitungen erlauben den Anschluss eines externen Timers an die CPU. Über die beiden Interrupt-Leitungen können zwei verschiedene externe Geräte das Programm der CPU unterbrechen. Die Leitungen TXT und RXD sind für eine serielle E/A-Verbindung zu einem Terminal oder Modem vorgesehen. Schließlich kann der Benutzer oder externe Hardware über das Signal RST den 8051 zurücksetzen. Diese Leitung wird normalerweise aktiviert, wenn etwas schief gegangen ist und das System neu gestartet werden muss.

Im Wesentlichen ist der 8051 den meisten anderen 8-Bit-Prozessoren ähnlich, abgesehen von den seriellen E/A-Leitungen. Was den 8051 auszeichnet, sind die 32 E/A-Leitungen, die als vier Ports organisiert und auf der rechten Seite von Abbildung 3.46 dargestellt sind. Diese Leitungen sind bidirektional ausgelegt und lassen sich per Programm lesen oder schreiben. Der 8051 kommuniziert hauptsächlich über diese Ports mit der Außenwelt. Und das macht ihn auch so wertvoll: Ein einziger Chip vereint CPU, Speicher und E/A-Möglichkeiten.

3.6 Beispielbusse

Busse sind der „Klebstoff“, der Computersysteme zusammenhält. Dieser Abschnitt beschäftigt sich mit einigen verbreiteten Bussen im Detail: ISA, PCI und USB (Universal Serial Bus). Der ISA-Bus ist eine leicht erweiterte Ausführung des ursprünglichen IBM-PC-Busses. Aus Gründen der Abwärtskompatibilität war er bis vor einigen Jahren immer noch in allen Intel-basierten PCs vorhanden, bis Intel und Microsoft übereingekommen sind, diesen Bus aufzugeben. Alle diese Rechner haben aber stets noch einen zweiten, schnelleren Bus – den PCI-Bus. Der PCI-Bus ist breiter als der ISA-Bus und läuft mit einer höheren Taktrate. Folglich kann er mehr Daten je Sekunde übertragen als der ISA-Bus. Er ist immer noch das Arbeitspferd in den meisten aktuellen PCs, obwohl ein Nachfolger bereits in Sicht ist. Der Universal Serial Bus setzt sich immer stärker als E/A-Bus für langsamere Peripheriegeräte, wie Mäuse und Tastaturen, durch. Eine zweite Version des USB läuft mit wesentlich höheren Geschwindigkeiten. In den folgenden Abschnitten sehen wir uns diese Busse und ihre Arbeitsweise nacheinander an.

3.6.1 ISA-Bus

Der IBM-PC-Bus war der De-facto-Standard in 8088-Systemen, weil ihn fast alle Hersteller von PC-Klonen kopiert haben, um die große Anzahl an E/A-Karten von Drittanbietern in ihren Systemen nutzen zu können. Er hatte 62 Signalleitungen, darunter 20 für eine Speicheradresse, 8 für Daten und je eine für Speicherlesen, Speicherschreiben, E/A-Lesen und E/A-Schreiben. Außerdem gab es Signale für die Anforderung und Gewährung von Interrupts und die DMA-Verwendung. Mehr nicht – es war ein sehr einfacher Bus.

Physisch war der Bus in Form von Leiterzügen auf dem PC-Motherboard realisiert. Der Bus war über etwa ein halbes Dutzend Steckverbinder im Abstand von 2 cm zugänglich, in die sich Leiterkarten einstecken ließen. Am Rand jeder Leiterkarte befand sich eine Kontaktreihe mit 31 vergoldeten Streifen, die den elektrischen Kontakt mit dem Steckverbinder herstellten.

Als IBM den 80286-PC/AT einführte, sah sich das Unternehmen einem Problem gegenüber. Bei einem völlig neu entwickelten 16-Bit-Bus hätten viele potenzielle Kunden mit dem Kauf gezögert, weil die zahlreichen PC-Steckkarten von Drittanbietern nicht mit dem neuen Computer funktioniert hätten. Wäre man andererseits bei dem PC-Bus mit seinen 20 Adress- und 8 Datenleitungen geblieben, hätte man die Fähigkeit des 80286, 16 MB Speicher zu adressieren und 16-Bit-Wörter zu übertragen, nicht nutzen können.

Man entschied sich dafür, den PC-Bus zu erweitern. PC-Steckkarten besitzen am Rand eine Kontaktreihe mit 62 Kontakten, die sich aber nicht über die gesamte Länge der Karte erstreckt. Die PC/AT-Lösung verwendete einfach einen zweiten Steckverbinder neben dem bisherigen und legte die AT-Schaltungstechnik so aus, dass sie mit beiden Kartentypen arbeitet. Abbildung 3.47 veranschaulicht dieses Konzept.

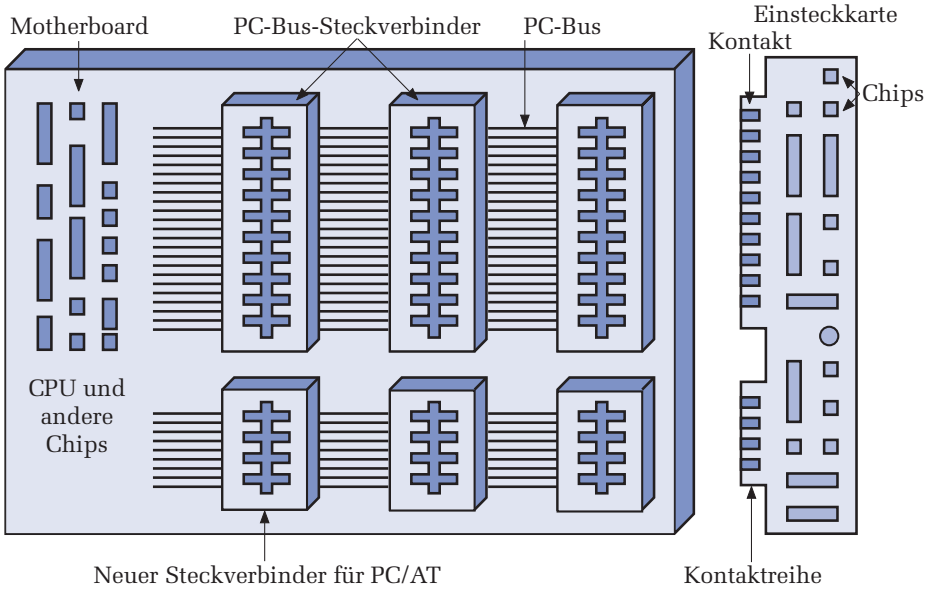


Abbildung 3.47: Der PC/AT-Bus besteht aus zwei Komponenten, dem ursprünglichen PC-Teil und dem neuen Teil

Der zweite Steckverbinder am PC/AT-Bus enthält 36 Leitungen. Davon sind 31 für zusätzliche Adress-, Daten- und Interrupt-Leitungen, für weitere DMA-Kanäle sowie Betriebsspannung und Masse vorgesehen. Der Rest hat mit den Unterschieden zwischen 8-Bit- und 16-Bit-Übertragungen zu tun.

Als IBM die PS/2-Serie als Nachfolger des PC und PC/AT herausbrachte, entschloss sich das Unternehmen zu einem Neubeginn. Diese Entscheidung basierte teilweise auf technischen Überlegungen (der PC-Bus war zu dieser Zeit wirklich überholt), teilweise aber auch auf dem Wunsch, den Herstellern von PC-Klonen ein Bein zu stellen. Diese hatten nämlich inzwischen einen beängstigend großen Marktanteil erobert. Folglich wurden die PS/2-Rechner im mittleren und oberen Leistungsbereich mit einem Bus, dem Microchannel, ausgestattet, der völlig neu und durch eine Mauer von Patenten und eine Armee von Anwälten geschützt war.

Die übrige PC-Industrie reagierte auf diese Entwicklung mit einem eigenen Standard, dem **ISA**-Bus (**Industry Standard Architecture**), der grundsätzlich der PC/AT-Bus mit einer Taktfrequenz von 8,33 MHz war. Dies bot den Vorteil, dass die Kompatibilität mit allen existierenden Rechnern und Karten gewahrt wurde. Er basierte zudem auf einem Bus, den IBM freizügig an viele Unternehmen lizensiert hatte, damit Drittanbieter möglichst viele Karten für den Original-PC produzierten, was sich im Nachhinein als Verhängnis für IBM herausstellte und das Unternehmen aus dem PC-Geschäft drängte. Bis vor einigen Jahren war dieser Bus noch in den meisten Intel-basierten PCs vorhanden, wenn auch mit einem oder mehreren anderen Bussen zusammen.

Später wurde der ISA-Bus auf 32 Bit erweitert und um ein paar neue Merkmale (z.B. für Mehrprozessorsysteme) ergänzt. Diesen neuen Bus nannte man **EISA** (**Extended ISA**).

3.6.2 PCI-Bus

Im ursprünglichen IBM-PC waren die meisten Anwendungen textbasiert. Mit der Einführung von Windows setzten sich nach und nach grafische Benutzeroberflächen durch. Keine dieser Anwendungen hatte dem ISA-Bus viel abgefordert. Als aber mit der Zeit viele neue Anwendungen – insbesondere Multimediaspiele – den Computer nutzten, um Video in Vollbilddarstellung anzuzeigen, änderte sich die Situation drastisch.

Ein einfaches Rechenbeispiel soll dies verdeutlichen. Wir nehmen eine Bildschirmauflösung von 1024×768 Bildpunkten für Video mit hoher Farbqualität (TrueColor = 3 Byte je Pixel) an. Ein Bildschirm enthält 2,25 MB Daten. Damit die Bewegungen flüssig ablaufen, sind mindestens 30 Bilder/s erforderlich, was eine Datenrate von 67,5 MB/s ergibt. Möchte man Videos von der Festplatte, einer CD-ROM oder von DVD anzeigen, kommt erschwerend hinzu, dass die Daten vom Laufwerk über den Bus zum Speicher fließen müssen. Und für die eigentliche Anzeige ist es erforderlich, die Daten erneut über den Bus zum Grafikadapter zu leiten. Somit brauchen wir allein für das Video 135 MB/s Bandbreite, ganz zu schweigen von der Bandbreite, die die CPU und andere Geräte benötigen.

Der ISA-Bus lief mit einem maximalen Takt von 8,33 MHz und konnte pro Zyklus 2 Bytes übertragen, was eine maximale Bandbreite von 16,7 MB/s bedeutet. Der EISA-Bus konnte 4 Bytes in jedem Zyklus übertragen und erreicht 33,3 MB/s. Es ist klar, dass beide Busse nicht einmal annähernd an das herankamen, was für Vollbildvideo notwendig ist.

Intel erkannte 1990 diese Entwicklung und konzipierte einen neuen Bus mit einer wesentlich höheren Bandbreite als der EISA-Bus. Dieser Bus erhielt die Bezeichnung

PCI-Bus (**Peripheral Component Interconnect Bus**). Um seinen Einsatz voranzutreiben, patentierte Intel den PCI-Bus und stellte dann alle Patente zur freien Verfügung, damit jedes Unternehmen Peripheriegeräte dafür bauen konnte, ohne Lizenzgebühren zahlen zu müssen. Außerdem gründete Intel ein Industriekonsortium, die PCI Special Interest Group, um die Zukunft des PCI-Busses zu verwalten. Diese Schritte führten dazu, dass der PCI-Bus sehr populär wurde. Praktisch jeder Intel-basierte Computer seit dem Pentium besitzt einen PCI-Bus und viele andere Computer haben ihn ebenfalls übernommen. Sun bietet sogar eine Version des UltraSPARC, den UltraSPARC IIIi, mit dem PCI-Bus an. Der PCI-Bus wird ausführlich in [Shanley und Anderson, 1999] und [Solari und Willse, 2004] beschrieben.

Der ursprüngliche PCI-Bus überträgt 32 Bit pro Zyklus und läuft mit 33 MHz (d.h. einer Zykluszeit von 30 ns), was eine Gesamtbandbreite von 133 MB/s ergibt. Im Jahre 1993 wurde PCI 2.0 eingeführt und 1995 kam die Version PCI 2.1 heraus. Die Spezifikation PCI 2.2 ist auf mobile Computer ausgerichtet (hauptsächlich, um die Batterie zu schonen). Der PCI-Bus läuft mit bis zu 66 MHz und kann 64 Bit übertragen, was eine Bandbreite von 528 MB/s bedeutet. Mit diesen Eigenschaften ist Vollbildvideo möglich (unter der Voraussetzung, dass die Festplatte und der Rest des Systems diesen Anforderungen genügen). Auf jeden Fall ist der PCI-Bus nicht mehr der Flaschenhals.

Selbst wenn sich die Rate von 528 MB/s gut anhört, bringt sie zwei Probleme mit sich. Erstens genügt sie nicht für einen Speicherbus. Zweitens ist sie nicht mit den vielen noch vorhandenen alten ISA-Karten kompatibel. Bei der von Intel entwickelten Lösung werden Computer mit drei oder mehr Bussen ausgestattet. Abbildung 3.48 zeigt, dass die CPU mit dem Hauptspeicher über einen speziellen Speicherbus kommuniziert und dass sich ein ISA-Bus an den PCI-Bus anschließen lässt. Da diese Anordnung alle Anforderungen erfüllte, wurde sie in den 90ern weithin eingesetzt.

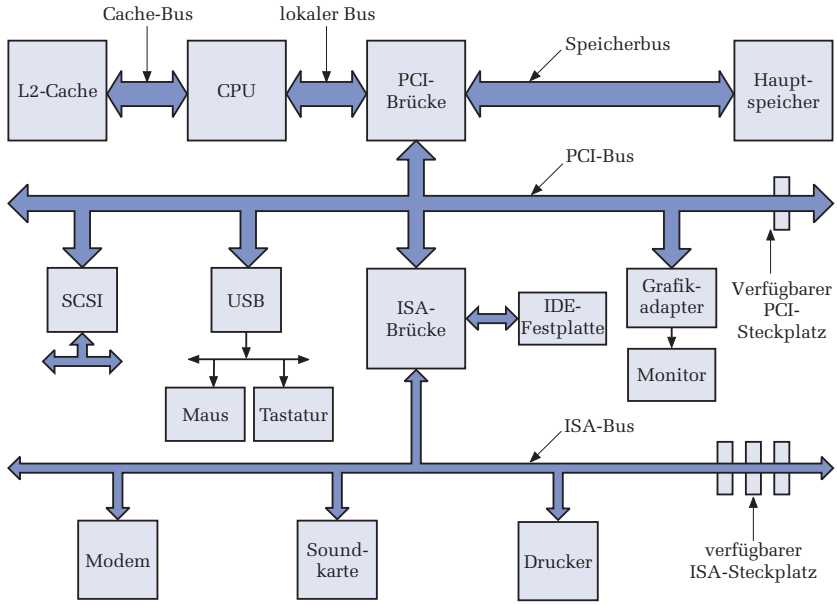


Abbildung 3.48: Architektur eines frühen Pentium-Systems. Die dicker gezeichneten Busse haben mehr Bandbreite als die dünneren, auch wenn die Darstellung nicht maßstabsgerecht ist.

Die beiden Brückenchips (Bridges) sind die Schlüsselkomponenten in diesem System. (Da Intel diese Chips herstellt, erklärt das wohl auch das Interesse an diesem System.) Die PCI-Brücke verbindet die CPU, den Speicher und den PCI-Bus. Die ISA-Brücke schließt den ISA-Bus an den PCI-Bus an und unterstützt eine oder zwei IDE-Festplatten. Nahezu alle Pentium 4-Systeme bieten einen oder mehrere freie PCI-Steckplätze für neue Hochgeschwindigkeitsperipheriegeräte und einen oder mehrere ISA-Steckplätze für langsamere Peripheriegeräte.

Das System in Abbildung 3.48 hat den großen Vorteil, dass der CPU über einen proprietären Speicherbus eine außerordentlich hohe Bandbreite zum Speicher zur Verfügung steht, der PCI-Bus eine hohe Bandbreite für schnelle Peripheriegeräte, wie zum Beispiel SCSI-Platten, Grafikadapter usw., bietet und ältere ISA-Karten weiterhin genutzt werden können. In der Abbildung bezieht sich das Kästchen mit der Bezeichnung USB auf den Universal Serial Bus, der später in diesem Kapitel behandelt wird.

Es wäre schön gewesen, wenn es nur bei einer Sorte PCI-Karten geblieben wäre. Leider ist das nicht der Fall. Somit existieren Optionen für Spannung, Verarbeitungsbreite und zeitliche Abläufe. Ältere Computer nutzen meist 5 V, während neuere zu 3,3 V übergehen, sodass der PCI-Bus beides unterstützt. Die Steckverbinder sind gleich, mit Ausnahme von zwei Kunststoffnasen, die verhindern sollen, dass man eine 5-V-Karte in einen 3,3-V-Bus steckt oder umgekehrt. Zum Glück gibt es auch universell einsetzbare Karten, die beide Spannungen unterstützen und für jeden Steckplatz geeignet sind. Abgesehen von den beiden Spannungen gibt es Karten in 32- und 64-Bit-Versionen. Die 32-Bit-Karten sind mit 120 Pins versehen, die 64-Bit-Karten besitzen die gleichen 120 Pins und zusätzlich 64 Pins. Vergleichbar ist das mit der Erweiterung des IBM-PC-Busses auf 16 Bit (siehe Abbildung 3.47). Ein PCI-Bussystem, das 64-Bit-Karten unterstützt, kann auch 32-Bit-Karten aufnehmen. Schließlich können PCI-Busse mit 33 oder 66 MHz laufen. Die Auswahl erfolgt über ein Pin, das entweder mit Betriebsspannung oder mit Masse verbunden wird. Die Steckverbinder sind für beide Geschwindigkeiten identisch.

In den späten 90ern war man sich ziemlich einig, dass der ISA-Bus überholt ist, sodass er in neuen Konzepten nicht mehr auftauchte. Allerdings hat sich inzwischen die Bildschirmauflösung in manchen Fällen auf 1600×1200 erhöht und der Bedarf nach Vollbildvideo ist ebenfalls größer geworden. Deshalb hat Intel einen weiteren Bus eingeführt, der lediglich die Grafikkarten bedient. Es handelt sich dabei um den **AGP-Bus** (**Accelerated Graphics Port Bus**). Die ursprüngliche Version AGP 1.0 lief mit 264 MB/s und wurde als 1x definiert. Der Bus ist zwar langsamer als der PCI-Bus, bleibt aber der Grafikkarte vorbehalten. Im Laufe der Jahre sind neue Versionen herausgekommen und inzwischen läuft die Version AGP 3.0 mit 2,1 GB/s (8x). Abbildung 3.49 zeigt ein modernes Pentium 4-System.

In diesem Design nimmt nun der Brückenchip eine zentrale Stellung ein und verbindet die fünf Hauptkomponenten des Systems: CPU, Speicher, Grafikkarte, ATAPI-Controller und PCI-Bus. In verschiedenen Varianten unterstützt er auch Ethernet und andere Hochgeschwindigkeitsgeräte. Die langsameren Geräte werden an den PCI-Bus angeschlossen.

Intern besteht der Brückenchip aus zwei Teilen: der Speicherbrücke und der E/A-Brücke. Die Speicherbrücke verbindet die CPU mit dem Speicher und dem Grafikadapter. Die E/A-Brücke hat die Aufgabe, den ATAPI-Controller, den PCI-Bus und (optional) andere schnelle E/A-Geräte mit einer direkten Brückenverbindung untereinander zu verbinden. Zwischen beiden Brücken besteht eine äußerst schnelle Zwischenverbindung.

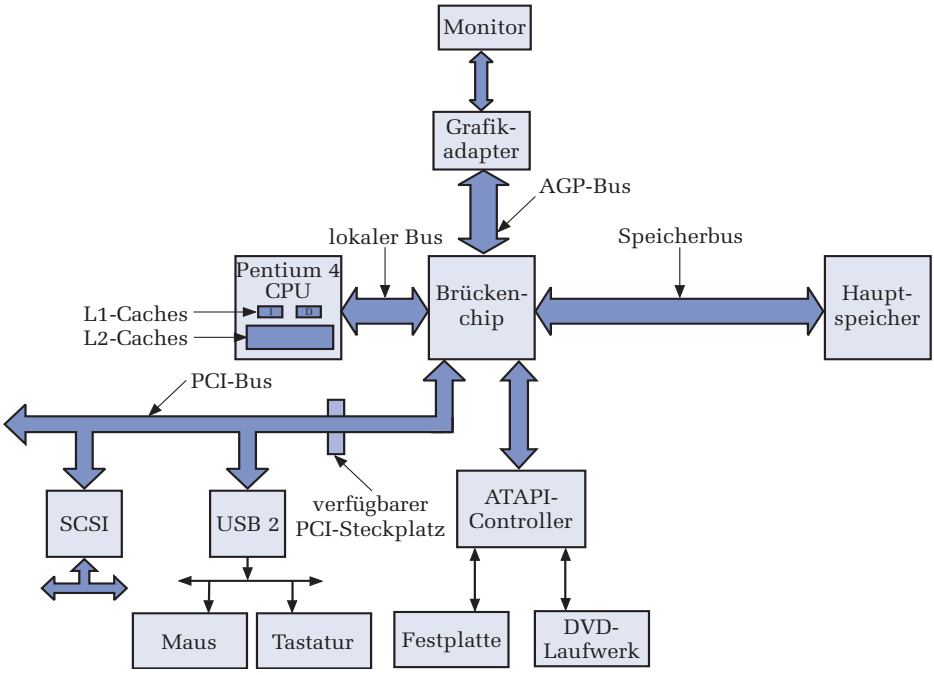


Abbildung 3.49: Die Busstruktur eines modernen Pentium 4-Systems

Wie alle PC-Busse, die auf den ursprünglichen IBM-PC zurückgehen, ist der PCI-Bus synchron. Alle Transaktionen erfolgen zwischen einem Master, der offiziell **Initiator** heißt, und einem Slave, den man **Target** nennt. Um die PCI-Pinanzahl gering zu halten, werden die Adress- und Datenleitungen gemultiplext. Auf diese Weise sind nur 64 Pins auf PCI-Karten für die Adresse und die Datensignale erforderlich, selbst wenn der PCI-Bus 64-Bit-Adressen und 64-Bit-Daten unterstützt.

Die gemultiplexten Adress- und Datenpins funktionieren wie folgt: Bei einer Leseoperation im Zyklus 1 legt der Master die Adresse auf den Bus. In Zyklus 2 entfernt der Master die Adresse und der Bus wird umgekehrt, sodass ihn der Slave benutzen kann. In Zyklus 3 gibt der Slave die angeforderten Daten aus. Bei Schreiboperationen muss der Bus nicht umgekehrt werden, weil der Master sowohl Adresse als auch Daten ausgibt. Die minimale Transaktion erfordert trotzdem immer drei Zyklen. Ist der Slave nicht in der Lage, in drei Zyklen zu reagieren, kann er Wartezustände einfügen. Blockübertragungen in unbegrenzter Größe und verschiedene andere Arten von Buszyklen sind ebenfalls zulässig.

PCI-Bus-Verwaltung

Damit ein Gerät den PCI-Bus benutzen kann, muss dieser ihm zuerst zugeteilt werden. Die PCI-Bus-Verwaltung setzt dafür einen zentralen Bus-Verwalter ein (siehe Abbildung 3.50). Bei den meisten Designs ist der Bus-Verwalter in einen der Brückenchips integriert. Von jedem PCI-Gerät gehen zwei dedizierte Leitungen zum Verwalter. Über die Leitung `REQ#` sendet das Gerät die Busanforderung (Request) und empfängt auf der Leitung `GNT#` die Buszuteilung (Grant).

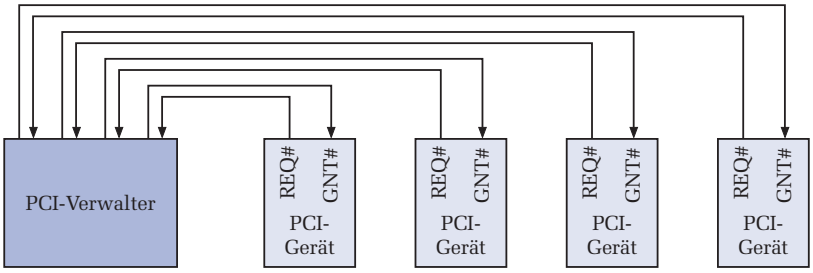


Abbildung 3.50: Der PCI-Bus verwendet einen zentralen Bus-Verwalter.

Um den Bus anzufordern, aktiviert ein PCI-Gerät (einschließlich der CPU) das Signal REQ# und wartet, bis seine GNT#-Leitung vom Verwalter aktiviert wird. Dann kann das Gerät den Bus im nächsten Zyklus benutzen. Der vom Verwalter benutzte Algorithmus ist in der PCI-Spezifikation jedoch nicht definiert. Round-Robin-Verwaltung, Prioritäten-Verwaltung und andere Verfahren sind zulässig. Selbstverständlich ist ein guter Verwalter fair und lässt kein Gerät ewig warten.

Eine Busüberlassung gilt nur für eine Transaktion, obwohl die Länge dieser Transaktion theoretisch unbegrenzt ist. Wenn ein Gerät eine zweite Transaktion durchführen möchte und kein anderes Gerät den Bus angefordert hat, kann es sofort fortfahren, auch wenn häufig ein Leerlaufzyklus zwischen zwei Transaktionen eingeschoben werden muss. Allerdings kann ein Gerät unter besonderen Umständen – wenn es gerade keinen Wettbewerb um den Bus gibt – mehrere Transaktionen hintereinander ausführen, ohne einen Leerlaufzyklus einschieben zu müssen. Führt ein Busmaster einen sehr langen Transfer durch und hat ein anderes Gerät währenddessen den Bus angefordert, kann der Verwalter die GNT#-Leitung negieren. Der aktuelle Busmaster muss die GNT#-Leitung überwachen, damit er den Bus beim Erkennen der Deaktivierung im nächsten Zyklus freigeben kann. Dieses Schema erlaubt sehr lange Transfers (die effizient sind), wenn es nur einen Anwärter auf den Busmaster gibt, aber dennoch kurze Reaktionszeiten für konkurrierende Geräte.

PCI-Bussignale

Der PCI-Bus hat mehrere obligatorische Signale, die in Tabelle 3.5 zusammengefasst sind, und eine Reihe von optionalen Signalen, die Tabelle 3.6 zeigt. Der Rest der 120 bzw. 184 Pins wird für Betriebsspannung, Masse und verschiedene Funktionen, die damit im Zusammenhang stehen, verwendet. (Diese Signale sind hier nicht aufgelistet.) In den Spalten *Master* und *Slave* ist angegeben, wer das Signal in einer normalen Transaktion aktiviert. Wird das Signal von einem anderen Gerät aktiviert (z.B. CLK), bleiben beide Spalten leer.

Tabelle 3.5

Obligatorische PCI-Bussignale

Signal	Leitungen	Master	Slave	Beschreibung
CLK	1			Takt (33 oder 66 MHz)
AD	32	×	×	Gemultiplexte Adress- und Datenleitungen
PAR	1	×		Adress- oder Datenparitätsbit
C/BE	4	×		Busbefehl/Bitmap für Byte Enable (zeigt gültige Datenbytes an)
FRAME#	1	×		Kennzeichnet, dass AD und C/BE aktiviert sind
IRDY#	1	×		Lesen: Master wird akzeptieren; Schreiben: Daten liegen an
IDSEL	1	×		Wählt Konfigurationsraum statt Speicher
DEVSEL#	1		×	Slave hat seine Adresse dekodiert und ist in Bereitschaft
TRDY#	1		×	Lesen: Daten liegen an; Schreiben: Slave wird akzeptieren
STOP#	1		×	Slave möchte Transaktion sofort abbrechen
PERR#	1			Empfänger hat Datenparitätsfehler erkannt
SERR#	1			Adressparitätsfehler oder Systemfehler erkannt
REQ#	1			Bus-Verwaltung: Anforderung des Busses
GNT#	1			Bus-Verwaltung: Zuteilung des Busses
RST#	1			Setzt das System und alle Geräte zurück

Wir sehen uns nun die einzelnen PCI-Bussignale kurz an und beginnen mit den obligatorischen (32-Bit-)Signalen. Anschließend wenden wir uns den optionalen (64-Bit-) Signalen zu. Das CLK-Signal steuert den Bus. Die meisten anderen Signale sind mit diesem Signal synchron. Im Gegensatz zum ISA-Bus beginnt eine PCI-Bustransaktion an der fallenden Flanke von CLK. Das ist nicht der Beginn, sondern die Mitte des Zyklus.

Die 32 AD-Signale sind für die Adresse und die Daten (bei 32-Bit-Transaktionen) bestimmt. Im Allgemeinen wird die Adresse in Zyklus 1 übertragen und die Daten in Zyklus 3. Das PAR-Signal ist ein Paritätsbit für AD. Das Signal C/BE# erfüllt zwei Aufgaben: In Zyklus 1 enthält es den Busbefehl (1 Wort lesen, Block lesen usw.). In Zyklus 2 enthält es ein Bitmap mit 4 Bits, die besagen, welche Bytes des 32-Bit-Wortes gültig sind. Mit C/BE# können 1, 2 oder 3 beliebige Bytes oder das ganze Wort gelesen oder geschrieben werden.

Tabelle 3.6

Optionale PCI-Bussignale

Signal	Leitungen	Master	Slave	Beschreibung
REQ64#	1	×		Anforderung zur Ausführung einer 64-Bit-Transaktion
ACK64#	1		×	Berechtigung wird für eine 64-Bit-Transaktion gewährt
AD	32	×		Zusätzliche 32 Bit für Adresse oder Daten
PAR64	1	×		Parität für die 32 zusätzlichen Adress-/Datenbits
C/BE#	4	×		Weitere 4 Bit für Byte Enable (zeigt gültige Datenbytes an)
LOCK	1	×		Bus sperren, um mehrere Transaktionen zu erlauben
SBO#	1			Treffer in einem entfernten Cache (bei einem Mehrprozessorsystem)
SDONE	1			Snooping ausgeführt (bei einem Mehrprozessorsystem)
INTx	4			Anforderung eines Interrupts
JTAG	5			JTAG-Testsignale nach IEEE 1149.1
M66EN	1			An Betriebsspannung oder Masse gelegt (wählt 66 oder 33 MHz)

Das Signal FRAME# wird vom Busmaster für den Start einer Bustransaktion aktiviert, um eine Bustransaktion einzuleiten. Es informiert den Slave darüber, dass die Adress- und Busbefehle jetzt gültig sind. In einer Leseoperation wird IRDY# normalerweise gleichzeitig mit FRAME# aktiviert. Dies besagt, dass der Master bereit ist, die eintreffenden Daten zu akzeptieren. Bei einer Schreiboperation wird IRDY# später aktiviert, wenn sich die Daten auf dem Bus befinden.

Das Signal IDSEL bezieht sich auf die Tatsache, dass jedes PCI-Gerät über einen Konfigurationsraum von 256 Byte verfügen muss, den andere Geräte (durch Aktivierung von IDSEL) lesen können. Dieser Konfigurationsraum enthält die Eigenschaften des Geräts. Die Plug-and-Play-Funktion mancher Betriebssysteme bestimmt anhand des Konfigurationsraums, welche Geräte auf dem Bus vorhanden sind.

Nun kommen wir zu den Signalen, die der Slave aktiviert. DEVSEL# kündigt an, dass der Slave seine Adresse auf den AD-Leitungen erkannt hat und bereit ist, in die Transaktion einzutreten. Wird DEVSEL# nicht innerhalb eines bestimmten Zeitraums aktiviert, läuft der Timer des Masters ab und der Master nimmt dann an, dass das adressierte Gerät nicht vorhanden oder ausgefallen ist.

Mit dem Signal TRDY# kündigt der Slave bei Leseoperationen an, dass sich die Daten auf den AD-Leitungen befinden. Bei Schreiboperationen kündigt der Slave damit an, dass er bereit ist, Daten zu akzeptieren.

Die nächsten drei Signale sind für Fehlermeldungen reserviert. STOP# wird vom Slave aktiviert, wenn etwas Unvorhergesehenes passiert und er die aktuelle Transaktion abbrechen möchte. PERR# wird benutzt, um einen Datenparitätsfehler aus dem vorherigen Zyklus zu melden. Bei einer Leseoperation wird es vom Master und bei einer Schreiboperation vom Slave aktiviert. Schließlich dient SERR# dazu, Adress- und Systemfehlern zu melden.

Über die Signale REQ# und GNT# erfolgt die Bus-Verwaltung. Diese Signale aktiviert nicht der aktuelle Busmaster, sondern ein Gerät, das zum Busmaster werden möchte. Das letzte obligatorische Signal ist RST#. Mit ihm setzt man das System zurück, wenn der Benutzer die RESET-Taste gedrückt oder ein Systemgerät einen nicht reparierbaren Fehler festgestellt hat. Die Aktivierung dieses Signals setzt alle Geräte zurück und startet den Computer neu.

Die optionalen Signale beziehen sich vor allem auf die Erweiterung von 32 auf 64 Bit. Die Signale REQ64# und ACK64# erlauben dem Master, eine Berechtigung für die Durchführung einer 64-Bit-Transaktion einzuholen, bzw. dem Slave, eine Transaktion mit 64 Bit Breite zu akzeptieren. Die Signale AD, PAR64 und C/BE# sind lediglich Erweiterungen der entsprechenden 32-Bit-Signale.

Die nächsten drei Signale haben nichts mit der Erweiterung von 32 auf 64 Bit zu tun, sondern mit Mehrprozessorsystemen – etwas, was PCI-Platinen nicht unbedingt unterstützen müssen. Mit dem Signal LOCK lässt sich der Bus für mehrere Transaktionen sperren. Die nächsten beiden Signale beziehen sich auf das Bus-Snooping, um die Kohärenz der Caches aufrechtzuerhalten.

Die INTx-Signale sind für die Anforderung von Interrupts vorgesehen. Auf einer PCI-Karte lassen sich bis zu vier separate logische Geräte realisieren, wobei jedes Gerät seine eigene Interrupt-Anforderungsleitung besitzen kann. Die JTAG-Signale sind für das JTAG-Testverfahren gemäß IEEE-Standard 1149.1 reserviert. Das Signal M66EN wird fest auf High oder Low gelegt, um die Taktgeschwindigkeit auszuwählen. Während des Systembetriebs darf es sich nicht ändern.

PCI-Bustransaktionen

Der PCI-Bus ist eigentlich sehr einfach. Das Taktdiagramm in Abbildung 3.51 soll Ihnen eine bessere Vorstellung davon vermitteln. Die Abbildung zeigt eine Lesetransaktion gefolgt von einem Leerlaufzyklus, an den sich eine Schreibtransaktion durch denselben Busmaster anschließt.

Mit der fallenden Flanke des Takts innerhalb von T_1 legt der Master die Speicheradresse auf AD und den Busbefehl auf C/BE#. Dann aktiviert er FRAME#, um die Bustransaktion zu starten.

In T_2 schaltet der Master den Adressbus hochohmig (Tristate), um eine Busumschaltung (Turnaround) vorzubereiten, damit der Slave in T_3 den Bus steuern kann. Außerdem zeigt der Master mit C/BE# an, welche Bytes im adressierten Wort gültig sein sollen (d.h. eingelesen werden).

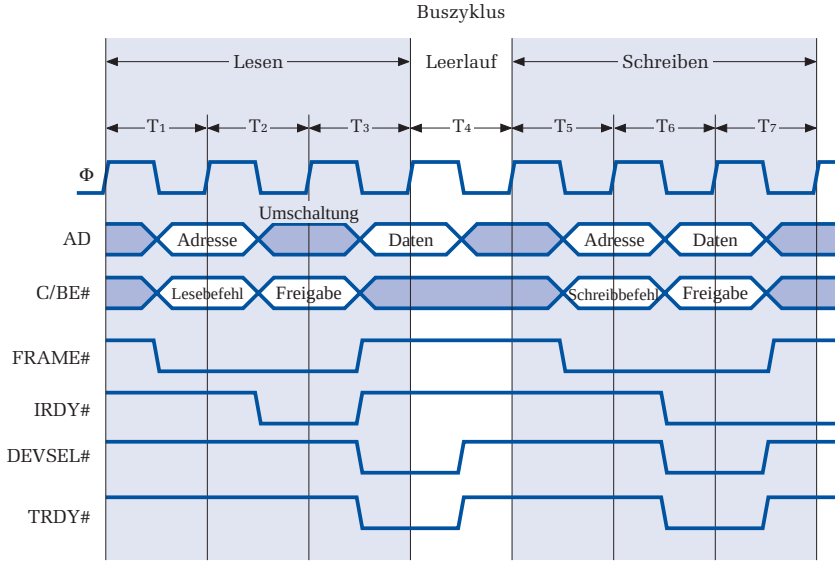


Abbildung 3.51: Beispiele für PCI-Bustransaktionen (32 Bit). In den ersten drei Zyklen findet eine Leseoperation statt, daran schließt sich ein Leerlaufzyklus an und in den folgenden drei Zyklen läuft eine Schreiboperation ab.

In T_3 aktiviert der Slave DEVSEL#, damit der Master weiß, dass der Slave die Adresse erhalten hat und antworten möchte. Außerdem legt er die Daten auf die AD-Leitungen und aktiviert TRDY#, um dies dem Master mitzuteilen. Auch wenn der Slave nicht in der Lage war, so schnell zu antworten, aktiviert er trotzdem DEVSEL#, um seine Anwesenheit anzukündigen, hält aber TRDY# deaktiviert, bis er die Daten absenden kann. Diese Prozedur kann einen oder mehrere Wartezustände einschieben.

In diesem Beispiel (und oftmals in der Praxis) ist der nächste Zyklus ein Leerlaufzyklus. Beginnend in T_5 leitet derselbe Master eine Schreiboperation ein. Zunächst setzt er wie üblich die Adresse und den Befehl auf den Bus. Jetzt aber aktiviert er im zweiten Zyklus die Daten. Da dasselbe Gerät die AD-Leitungen steuert, ist kein Busumschaltungszyklus notwendig. In T_7 akzeptiert der Speicher die Daten.

3.6.3 PCI Express

Obwohl der PCI-Bus für die meisten derzeitigen Anwendungen angemessen funktioniert, führt die Nachfrage nach größerer E/A-Bandbreite zu einem Durcheinander in der bisher so aufgeräumten PC-Architektur. Aus Abbildung 3.49 geht klar hervor, dass der PCI-Bus nicht mehr das zentrale Element ist, das die Komponenten des PCs zusammenhält. Die Brückenchips haben einen Teil dieser Rolle übernommen.

Das Problem besteht darin, dass es immer mehr E/A-Geräte gibt, die für den PCI-Bus zu schnell sind. Es ist keine gute Lösung, die Taktfrequenz auf dem Bus hochzuschrauben, weil sich dann Probleme durch unterschiedliche Signalgeschwindigkeiten (Bus Skew), Übersprechen zwischen den Leitungen und Effekte infolge parasitärer Kapazitäten verstärken. Jedes Mal wenn ein E/A-Gerät zu schnell für den PCI-Bus wird (wie zum Beispiel Grafikkarte, Festplatte, Netzwerk usw.), fügt Intel einen neuen speziellen Port in den Brückchip ein, damit dieses Gerät den PCI-Bus umgehen kann. Zweifellos ist dies auch keine längerfristige Lösung.

Dem PCI-Bus haftet weiter das Problem an, dass die Karten ziemlich groß sind. Sie passen nicht in Laptops oder Palmtops und die Hersteller wollen ja sogar noch kleinere Geräte produzieren. Und einige Hersteller möchten den PC ganz neu definieren, wobei CPU und Speicher in einer kleinen versiegelten Box und die Festplatte im Monitor untergebracht sind. Mit PCI-Karten ist dies unmöglich.

Von den verschiedenen vorgeschlagenen Lösungen hat eine Aussicht, das Rennen zu machen (nicht zuletzt deshalb, weil Intel dahintersteht): **PCI Express**. Das Ganze hat wenig mit dem PCI-Bus zu tun und ist eigentlich überhaupt kein Bus. Doch die Marketing-Strategen möchten den gut eingeführten Namen PCI nicht fallen lassen. PCs, die damit ausgerüstet sind, gibt es schon seit einiger Zeit auf dem Markt. Sehen wir uns nun an, wie PCI Express funktioniert.

Die PCI Express-Architektur

Zentrales Anliegen der PCI Express-Lösung ist es, vom parallelen Bus mit seinen vielen Mastern und Slaves abzukommen und zu einem Design überzugehen, das auf seriellen Punkt-zu-Punkt-Hochgeschwindigkeitsverbindungen basiert. Diese Lösung stellt einen radikalen Bruch mit der ISA/EISA/PCI-Bus-Tradition dar und übernimmt viele Ideen aus der Welt der lokalen Netze, insbesondere vom Switched Ethernet. Das Grundkonzept lässt sich wie folgt auf einen Nenner bringen: Im Innersten stellt ein PC eine Ansammlung von CPU, Speicher und E/A-Controllerchips dar, die untereinander verbunden werden müssen. PCI Express stellt einen universellen Switch bereit, um Chips mithilfe von seriellen Verbindungen zusammenzuschalten. Abbildung 3.52 zeigt eine typische Konfiguration.

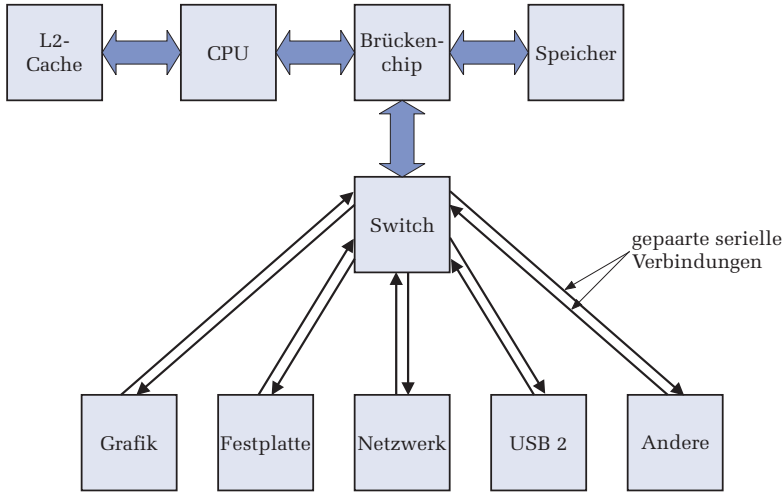


Abbildung 3.52: Ein typisches PCI Express-System

Wie Abbildung 3.52 zeigt, sind die CPU, der Speicher und der Cache in der herkömmlichen Form an den Brückenchip angeschlossen. Das Neue hier ist ein Switch, der mit der Brücke verbunden (und möglicherweise Teil der Brücke selbst) ist. Von jedem E/A-Chip führt eine dedizierte Punkt-zu-Punkt-Verbindung zum Switch. Jede Verbindung umfasst ein Paar unidirektionaler Kanäle – einer zum Switch hin und einer von

ihm weg. Ein Kanal besteht aus zwei Leitungen, einer Signalleitung und einer Masseleitung, um hohe Störsicherheit während der Hochgeschwindigkeitsübertragung zu gewährleisten. Diese Architektur ersetzt die derzeitige durch ein wesentlich einheitlicheres Modell, das alle Geräte gleich behandelt.

Die PCI Express-Architektur unterscheidet sich von der alten PCI-Bus-Architektur in drei wesentlichen Punkten. Zwei haben wir bereits kennen gelernt: ein zentraler Switch gegenüber einem Mehrpunktbus und die Verwendung von schmalen Punkt-zu-Punkt-Verbindungen gegenüber einem breiten Parallelbus. Der dritte Punkt ist etwas subtiler. Konzeptionell löst beim PCI-Bus ein Busmaster einen Befehl an einen Slave aus, um ein Wort oder einen Block von Worten zu lesen. Beim PCI Express-Modell sendet ein Gerät ein Datenpaket an ein anderes Gerät. Das Konzept des **Pakets**, das aus Header und Nutzdaten besteht, stammt aus der Netzwerkwelt. Der **Header** enthält Steuerinformationen, die die vielen Steuerleitungen des PCI-Busses überflüssig machen. Die Nutzdaten enthalten die zu übertragenden Daten. In der Tat ist ein PC mit PCI Express ein Paketvermittlungsnetz (Packet Switching Network) en miniature.

Der Bruch mit der Vergangenheit dokumentiert sich neben diesen drei Hauptpunkten auch noch in mehreren kleineren Unterschieden. So sind viertens die Pakete mit einem Fehlerkorrekturcode versehen, was eine höhere Zuverlässigkeit als auf dem PCI-Bus bringt. Fünftens ist die Verbindung zwischen einem Chip und dem Switch mit 50 cm länger als bisher, um eine Systemaufteilung zu ermöglichen. Sechstens ist das System erweiterbar, weil ein Gerät durchaus auch ein anderer Switch sein kann und sich somit ein Baum von Switches bilden lässt. Siebentens sind Geräte Hot-Plugging-fähig, d.h. sie lassen sich im laufenden Betrieb des Systems ein- und ausbauen. Und da schließlich die Größe der seriellen Steckverbinder nur einen Bruchteil der alten PCI-Steckverbinder ausmacht, kann man Geräte und Computer wesentlich kleiner gestalten. Insgesamt gesehen ist PCI Express eine radikale Abkehr vom PCI-Bus.

Der PCI Express-Protokollstapel

Entsprechend dem Modell eines paketvermittelten Netzwerks baut das PCI Express-System auf einem geschichteten Protokollstapel auf. Ein **Protokoll** ist eine Menge von Regeln, die die Konversation zwischen zwei Teilnehmern bestimmen. Ein Protokollstapel umfasst eine Hierarchie von Protokollen, die sich mit unterschiedlichen Fragen auf unterschiedlichen Ebenen befassen. Nehmen Sie als Beispiel einen Geschäftsbrief. Er hält sich an bestimmte Konventionen hinsichtlich Platzierung und Inhalt von Briefkopf, Empfängeradresse, Datum, Anrede, eigentlichem Text, Unterschrift usw. Das kann man sich als Briefprotokoll vorstellen. Daneben gibt es eine Menge von Konventionen zum Briefumschlag – zum Beispiel Größe, Platz und Format des Absenders, Platz und Format der Anschrift, Anordnung der Briefmarke usw. Diese beiden Ebenen und ihre Protokolle sind unabhängig. Zum Beispiel ist es möglich, den Brief vollständig neu zu formatieren, aber denselben Umschlag zu verwenden und umgekehrt. Geschichtete Protokolle fördern ein modulares flexibles Design und haben sich seit Jahrzehnten in der Welt der Netzwerke weithin etabliert. Neu ist hier, dass man sie in die „Bus-“ Hardware integriert.

Abbildung 3.53(a) zeigt den Protokollstapel des PCI Express-Systems.

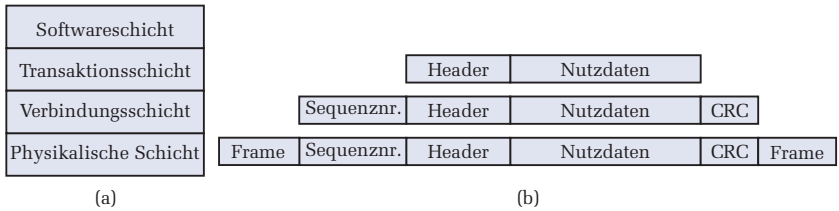


Abbildung 3.53: (a) Der PCI Express-Protokollstapel, (b) Format eines Pakets

Wir untersuchen nun die Schichten von unten nach oben. Die unterste Schicht ist die **physikalische Schicht** (Physical Layer). Hier findet die Übertragung der Bits von einem Sender zu einem Empfänger über eine Punkt-zu-Punkt-Verbindung statt. Jede Punkt-zu-Punkt-Verbindung besteht aus einem oder mehreren Paaren von Simplexverbindungen (d.h. unidirektionalen Verbindungen). Im einfachsten Fall gibt es in jeder Richtung ein Paar, jedoch sind auch 2, 4, 8, 16 oder 32 Paare zulässig. Eine derartige Verbindung wird als **Lane** (engl. Spur, Strang) bezeichnet. Die Anzahl der Lanes muss in jeder Richtung gleich sein. Produkte der ersten Generation müssen auf jedem Weg eine Datenrate von mindestens 2,5 Gbit/s unterstützen. Es ist aber recht bald damit zu rechnen, dass man auf jedem Weg zu einer Geschwindigkeit von 10 Gbit/s übergeht.

Im Unterschied zu den ISA/EISA/PCI-Bussen verfügt PCI Express nicht über einen Mastertakt. Geräte können aufs Geratewohl mit einer Übertragung beginnen, sobald sie zu sendende Daten haben. Diese Freiheit vereinfacht das System, führt aber auch zu Problemen. Nehmen wir an, dass ein 1-Bit mit +3 V und ein 0-Bit mit 0 V kodiert ist. Wenn die ersten Bytes sämtlich 0 sind, wie erkennt dann der Empfänger, dass Daten übertragen werden? Immerhin sieht eine Folge von 0-Bits genau wie eine Verbindung im Leerlauf aus. Als Lösung hat man die so genannte **8b/10b-Kodierung** gewählt. Dabei werden 10 Bits verwendet, um 1 Byte der eigentlichen Daten in einem 10-Bit-Symbol zu kodieren. Von den 1024 möglichen 10-Bit-Symbolen hat man solche mit genügend Taktübergängen ausgewählt, um Sender und Empfänger auf den Bitgrenzen selbst ohne Mastertakt synchronisieren zu können. Aufgrund der 8b/10b-Kodierung kann eine Verbindung mit einer Bruttokapazität von 2,5 Gbit/s nur 2 Gbit/s Benutzerdaten (Nettodaten) transportieren.

Während die physikalische Schicht mit der Bitübertragung zu tun hat, beschäftigt sich die **Verbindungsschicht** (Link Layer) mit der Paketübertragung. Sie übernimmt den Header und die Nutzdaten von der Transaktionsschicht, fügt diesen Daten eine Sequenznummer hinzu, berechnet einen Fehlerkorrekturcode und hängt diesen an das Paket an. Der als **CRC** (**Cyclic Redundancy Code**) bezeichnete Fehlerkorrekturcode wird nach einem bestimmten Algorithmus aus den Header- und Nutzdaten berechnet. Erhält der Empfänger ein Paket, führt er die gleiche Berechnung auf den Header- und Nutzdaten durch. Stimmt das Ergebnis mit dem im Paket angefügten CRC-Wert überein, sendet der Empfänger ein kurzes **Bestätigungspaket** (Acknowledgment Packet) zurück, das den korrekten Empfang quittiert. Bei abweichenden Ergebnissen fordert der Empfänger eine Neuübertragung an. Dieses Verfahren gewährleistet eine deutlich bessere Datenintegrität gegenüber PCI-Bussystemen, die keinerlei Vorkehrungen für Verifizierung und Neuübertragung der über den Bus gesendeten Daten kennen.

Um zu verhindern, dass ein schneller Sender einen langsamen Empfänger in Paketen erstickt, die dieser nicht behandeln kann, wird eine **Flusssteuerung** verwendet. Der verwendete Mechanismus stützt sich darauf, dass der Empfänger dem Sender eine

bestimmte Anzahl von Credits gibt, die grundsätzlich der Größe des Puffers entsprechen, die er für eintreffende Pakete verfügbar hat. Wenn die Credits aufgebraucht sind, muss der Sender das Senden stoppen, bis er neue Credits erhält. Dieses in allen Netzwerken übliche Schema verhindert, dass Pakete aufgrund nicht übereinstimmender Geschwindigkeiten von Sender und Empfänger verloren gehen.

Die **Transaktionsschicht** behandelt Busaktionen. Um ein Wort aus dem Speicher zu lesen, sind zwei Transaktionen erforderlich: eine von der CPU oder einem DMA-Kanal eingeleitete, um bestimmte Daten anzufordern, und eine von dem Target eingeleitete, das die Daten bereitstellt. Allerdings hat die Transaktionsschicht noch mehr zu tun, als reine Lese- und Schreibvorgänge abzuwickeln. Sie bereitet die von der Verbindungsschicht angebotene Rohpaketübertragung auf. Dazu kann sie zunächst jede Lane in bis zu acht **virtuelle Verbindungen** unterteilen, die jeweils eine andere Klasse von Verkehr behandeln. Die Transaktionsschicht markiert Pakete entsprechend ihrer Verkehrsklasse, wozu Attribute wie hohe Priorität, niedrige Priorität, „schnüffelt“ nicht, kann unabhängig von der Reihenfolge bereitgestellt werden usw. gehören. Anhand dieser Tags kann der Switch entscheiden, welches Paket als Nächstes an die Reihe kommt.

Jede Transaktion verwendet einen von vier Adressräumen:

1. Speicherraum (für normale Lese- und Schreiboperationen)
2. E/A-Raum (für die Adressierung von Geräteregistern)
3. Konfigurationsraum (für Systeminitialisierung etc.)
4. Nachrichtenraum (für Signalisierung, Interrupts etc.)

Die Speicher- und E/A-Räume sind mit aktuellen Systemen vergleichbar. Mit dem Konfigurationsraum lassen sich Funktionen wie Plug&Play implementieren. Der Nachrichtenraum übernimmt die Rolle vieler vorhandener Steuersignale. Etwas in der Art wie dieser Raum ist unbedingt notwendig, da in PCI Express keine Steuerleitungen des PCI-Busses existieren.

Die **Softwareschicht** stellt die Schnittstelle des PCI Express-Systems zum Betriebssystem her. Sie kann den PCI-Bus emulieren, sodass vorhandene Betriebssysteme unverändert auf PCI Express-Systemen laufen können. Natürlich nutzt ein derartiger Betrieb nicht die volle Leistung von PCI Express, aber die Abwärtskompatibilität ist ein notwendiges Übel, auf das man erst verzichten kann, wenn die Betriebssysteme vollständig auf PCI Express ausgerichtet sind. Die Erfahrung zeigt, dass das noch eine ganze Weile dauern kann.

Abbildung 3.53(b) zeigt den Informationsfluss. Wird ein Befehl an die Softwareschicht erteilt, übergibt sie ihn an die Transaktionsschicht, die ihn als Header und Nutzdaten formuliert. Diese beiden Teile werden an die Verbindungsschicht übergeben, die eine Sequenznummer vorn und eine CRC-Prüfsumme hinten anfügt. Dieses vergrößerte Paket geht dann zur physikalischen Schicht, die Rahmeninformationen an jedes Paket anfügt, um das physische Paket zu bilden, das dann tatsächlich gesendet wird. Am empfangenden Ende findet der umgekehrte Prozess statt, wobei die voran- und nachgestellten Verbindungsinformationen entfernt werden und das Ergebnis an die Transaktionsschicht übergeben wird.

Das Konzept, dass jede Schicht zusätzliche Informationen an die Daten anfügt, wenn sie den Protokollstapel von oben nach unten durchlaufen, wird seit Jahrzehnten

mit Erfolg in der Netzwerkwelt angewandt. Während aber der Code in der Netzwerkwelt nahezu immer durch Software des Betriebssystems realisiert ist, gehört er bei PCI Express zur Gerätehardware.

PCI Express ist ein komplexes Thema. Weitere Informationen finden Sie in [Mayhew und Krishnan, 2003] und [Solari und Congdon, 2005].

3.6.4 USB (Universal Serial Bus)

Der PCI-Bus und PCI Express eignen sich gut zum Anschließen von Hochgeschwindigkeitsperipherie an einen Computer, sind aber für langsame E/A-Geräte wie Tastaturen und Mäuse bei weitem zu teuer. Im Verlauf der Geschichte wurde jedes E/A-Standardgerät auf besondere Weise an den Computer angeschlossen, wofür freie ISA- und PCI-Steckplätze genutzt wurden. Leider war dieses Verfahren von Anfang an mit Problemen behaftet.

Beispielsweise wird ein neues E/A-Gerät meistens mit einer eigenen ISA- oder PCI-Karte geliefert. Der Benutzer muss manchmal die Schalter und Steckbrücken (Jumper) auf der Karte selbst einstellen und gewährleisten, dass die Einstellungen nicht zu Konflikten mit anderen Karten führen. Dann muss der Benutzer das Gehäuse öffnen, die Karte vorsichtig einstecken, das Gehäuse schließen und den Rechner neu starten. Für viele Benutzer ist dieser Vorgang schwierig und birgt viele Fehlerquellen. Außerdem ist die Anzahl der ISA- und PCI-Steckplätze recht begrenzt (normalerweise auf zwei oder drei). Plug-and-Play-Karten ersparen dem Benutzer zwar die Einstellung der Jumper, zur Installation muss er aber trotzdem den Computer öffnen und die Karte einstecken. An der Anzahl der Bussteckplätze hat sich nichts geändert.

Um dieses Problem anzugehen, haben sich 1993 die Vertreter von sieben Unternehmen (Compaq, DEC, IBM, Intel, Microsoft, NEC und Northern Telecom) an einen Tisch gesetzt, um ein besseres Designkonzept für den Anschluss langsamer E/A-Geräte an einen Computer auf den Weg zu bringen. Seither sind Hunderte weiterer Unternehmen der Gruppe beigetreten. Der daraus hervorgegangene Standard heißt **USB** (**Universal Serial Bus**) und wird inzwischen in vielen Computern implementiert. Detaillierte Informationen finden Sie in [Anderson, 1997] und [Tan, 1997].

Mit dem USB-Projekt verfolgten die Gründungsmitglieder der USB-Arbeitsgruppe unter anderem folgende Ziele:

1. Benutzer sollen keine Schalter und Brücken auf Platinen oder Geräten einstellen müssen.
2. Benutzer sollen das Gehäuse nicht zur Installation neuer E/A-Geräte öffnen müssen.
3. Es soll nur eine Kabelart für den Anschluss aller Geräte verwendet werden.
4. Die Stromversorgung der E/A-Geräte soll über das Kabel erfolgen.
5. An einen einzigen Computer sollen sich bis zu 127 Geräte anschließen lassen.
6. Das System soll Echtzeitgeräte (z.B. Sound, Telefon) unterstützen.
7. Geräte sollen bei laufendem Computer installierbar sein.
8. Nach der Installation eines neuen Geräts soll kein Neustart des Computers erforderlich sein.
9. Der neue Bus und seine E/A-Geräte sollen preisgünstig gefertigt werden können.

Der USB erfüllt alle diese Ziele. Er wurde für langsame Geräte, z.B. Tastaturen, Mäuse, Standbildkameras, Büroscanner, digitale Telefone usw., ausgelegt. Version 1.0 hatte eine Bandbreite von 1,5 Mbit/s, die für Tastaturen und Mäuse ausreicht. Version 1.1 läuft bei 12 Mbit/s, was für Drucker, Digitalkameras und viele andere Geräte genügt. Diese Untergrenzen wurden gewählt, um die Kosten niedrig zu halten.

Ein USB-System besteht aus einem **Root-Hub**, der in den Hauptbus (siehe Abbildung 3.48) eingesteckt wird. Dieser Hub hat Buchsen für Kabel, über die E/A-Geräte oder Erweiterungshubs angeschlossen werden können, sodass weitere Buchsen bereitstehen. Die Topologie eines USB-Systems ist also ein Baum mit der Wurzel beim Root-Hub im Innern des Computers. Die Steckverbinder der Kabel unterscheiden sich zwischen Hub-Seite und Geräteseite, um zu verhindern, dass man versehentlich zwei Hubs zusammensteckt.

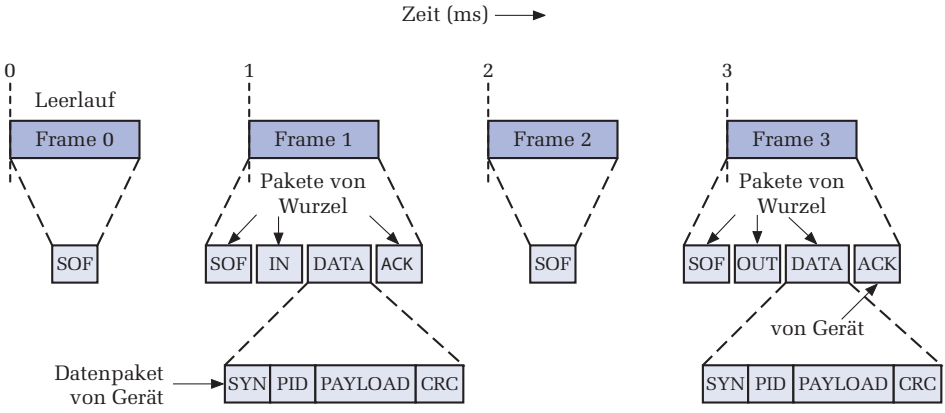


Abbildung 3.54: Der USB-Root-Hub sendet alle 1,00 ms Frames aus

Das Kabel besteht aus vier Drähten – zwei für Daten, einem für die Betriebsspannung (+5 V) und einem für Masse. Das Signalisierungssystem überträgt eine 0 als Spannungsübergang und eine 1 als Fehlen eines Spannungsübergangs, sodass lange Folgen von Nullen einen regelmäßigen Impulsstrom erzeugen.

Der Root-Hub erkennt, wenn ein neues E/A-Gerät eingesteckt wird, und unterbricht das Betriebssystem. Das Betriebssystem fragt das Gerät danach ab, um welches es sich handelt, und wie viel USB-Bandbreite es benötigt. Stellt das Betriebssystem fest, dass ausreichend Bandbreite für das Gerät vorhanden ist, weist es ihm eine eindeutige Adresse (1–127) zu. Dann lädt es diese Adresse und andere Informationen in Konfigurationsregister innerhalb des Geräts. Auf diese Weise können neue Geräte problemlos hinzugefügt werden, ohne dass der Benutzer etwas konfigurieren oder installieren muss. Nicht initialisierte Karten erhalten zunächst die Adresse 0, damit sie sich ansprechen lassen. Zur Vereinfachung der Verkabelung sind in vielen USB-Geräten Hubs zur Aufnahme zusätzlicher USB-Geräte eingebaut. Ein Monitor kann beispielsweise zwei Hub-Buchsen bekommen, um den linken und rechten Lautsprecher unterzubringen.

Logisch kann man das USB-System als Menge von Bit-Pipes vom Root-Hub zu den E/A-Geräten betrachten. Jedes Gerät kann seine Bit-Pipe in maximal 16 Teil-Pipes für unterschiedliche Datenarten (z.B. Audio und Video) aufteilen. Innerhalb jeder Pipe oder jeder Teil-Pipe fließen Daten vom Root-Hub zum Gerät oder umgekehrt. Zwischen zwei E/A-Geräten gibt es keinen Verkehr.

Genau alle 1,00 ± 0,05 ms sendet der Root-Hub für die Synchronisation einen neuen Frame an alle Geräte. Ein Frame ist mit einer Bit-Pipe verbunden und besteht aus Paketen. Das erste Paket kommt vom Root-Hub des Geräts. Die folgenden Pakete im Frame können ebenfalls in diese Richtung oder vom Gerät zurück zum Root-Hub laufen. Abbildung 3.54 zeigt eine Folge von vier Frames.

In Abbildung 3.54 haben die Frames 0 und 2 nichts zu tun, sodass nur ein SOF-Paket (Start Of Frame) erforderlich ist. Dieses Paket wird immer als Rundruf an alle Geräte gesendet. Frame 1 ist eine Abfrage, z.B. eine Anforderung an einen Scanner, die im abgetasteten Bild gefundenen Bits zurückzugeben. Frame 3 besteht aus den Bereitstellungsdaten für ein bestimmtes Gerät, wie zum Beispiel einen Drucker.

Der USB unterstützt vier Arten von Frames: Control, Isochronous, Bulk und Interrupt. Control-Frames werden benutzt, um Geräte zu konfigurieren, Befehle an sie zu erteilen und ihren Status abzufragen. Isochronous-Frames sind für Echtzeitgeräte, wie Mikrofone, Lautsprecher und Telefone, die Daten in genauen Zeitintervallen senden oder annehmen müssen, vorgesehen. Sie haben eine genau vorhersagbare Verzögerung, bieten im Fall von Fehlern aber keine Möglichkeit der erneuten Übertragung. Bulk-Frames werden für umfangreiche Übertragungen zu oder von Geräten ohne Echtzeitanforderungen, z.B. Drucker, benutzt. Schließlich sind Interrupt-Frames nötig, weil der USB keine Interrupts unterstützt. Anstatt die Tastatur bei jedem Tastenanschlag einen Interrupt auslösen zu lassen, kann das Betriebssystem die Tastatur alle 50 ms abfragen, um eventuell ausstehende Tastenanschläge zu sammeln.

Ein Frame besteht aus einem oder mehreren Paketen, die möglicherweise in beide Richtungen gesendet werden. Es gibt vier Paketarten: Token, Data, Handshake und Special. Token-Pakete fließen vom Root-Hub zu einem Gerät und dienen der Systemsteuerung. Die Pakete SOF, IN und OUT in Abbildung 3.54 sind Token-Pakete. Das Paket SOF (Start Of Frame) ist das erste in jedem Frame und kennzeichnet den Frame-Beginn. Wenn es nichts zu tun gibt, ist SOF das einzige Paket im Frame. Das Token-Paket IN ist eine Abfrage, die das Gerät auffordert, bestimmte Daten zurückzugeben. Die Felder im IN-Paket geben Auskunft darüber, welche Bit-Pipe abgefragt wird, sodass das Gerät weiß, welche Daten zurückzugeben sind (wenn es mehrere Ströme hat). Das Token-Paket OUT kündigt an, dass Daten für das Gerät folgen. Ein vierter Token-Pakettyp SETUP (in der Abbildung nicht gezeigt) wird für die Konfiguration verwendet.

Neben dem Token-Paket gibt es noch drei weitere: DATA (für die Übertragung von 64 Byte Informationen in beide Richtungen, Handshake und Special. Das Format eines Datenpakets ist aus Abbildung 3.54 ersichtlich. Es besteht aus einem 8-Bit-Synchronisierungsfeld, einem 8-Bit-Pakettyp (PID), den Nutzdaten (Payload) und einem 16-Bit-**CRC** (**Cyclic Redundancy Code**) zur Fehlererkennung. Es sind drei Arten von Handshake-Paketen definiert: ACK (das vorherige Datenpaket wurde korrekt empfangen), NAK (ein CRC-Fehler wurde erkannt) und STALL (bitte warten, bin momentan beschäftigt).

Sehen Sie sich Abbildung 3.54 noch einmal an. Alle 1,00 ms muss ein Frame vom Root-Hub gesendet werden, auch wenn nichts zu tun ist. Die Frames 0 und 2 bestehen nur aus einem SOF-Paket, das diesen Leerlauf anzeigt. Frame 1 ist eine Abfrage und beginnt deshalb mit SOF- und IN-Paketen vom Computer zum E/A-Gerät, gefolgt von einem DATA-Paket vom Gerät zum Computer. Das ACK-Paket teilt dem Gerät mit, welche Daten korrekt empfangen wurden. Im Fall eines Fehlers würde ein NAK an das Gerät zurückgeschickt und das Paket bei Bulk-Daten (nicht aber bei isochronen Daten) erneut übertragen werden. Der Aufbau von Frame 3 entspricht dem von Frame 1, außer dass die Daten vom Computer zum Gerät fließen.

Gerade als der USB-Standard im Jahre 1998 verabschiedet wurde, hatten die Designer von USB nichts besseres zu tun, als den Beginn der Arbeiten an einer neuen Hochgeschwindigkeitsversion von USB namens **USB 2.0** zu verkünden. Dieser Standard ist dem älteren USB 1.1 ähnlich und dazu abwärtskompatibel, außer dass er den bisher vorhandenen zwei Geschwindigkeiten mit 480 Mbit/s noch eine dritte Geschwindigkeit hinzufügt. Außerdem gibt es einige kleinere Unterschiede, wie zum Beispiel bei der Schnittstelle zwischen dem Root-Hub und dem Controller. Mit USB 1.1 waren zwei Schnittstellen verfügbar. Die erste namens **UHCI** (**Universal Host Controller Interface**) stammt von Intel und verlagert einen Großteil der Arbeitslast auf die Softwareentwickler (sprich: Microsoft). Die zweite Schnittstelle mit der Bezeichnung **OHCI** (**Open Host Controller Interface**) kommt aus dem Hause Microsoft und bürdet die meiste Arbeit den Hardwareentwicklern (sprich: Intel) auf. In USB 2.0 haben sich nun alle Partner auf eine einzige neue Schnittstelle namens **EHCI** (**Enhanced Host Controller Interface**) geeinigt.

USB ist nun mit einer Arbeitsgeschwindigkeit von 480 Mbit/s ein klarer Konkurrent zum seriellen IEEE 1394-Bus, der unter dem Namen FireWire bekannter ist und bei 400 Mbit/s läuft. Obwohl praktisch jedes neue Pentium-System mit USB 2.0 ausgeliefert wird, lässt sich 1394 nicht unterkriegen, weil diese Schnittstelle das Rückgrat der Konsumelektronikindustrie bildet. Camcorder, DVD-Player und ähnliche Geräte werden in absehbarer Zukunft weiterhin mit 1394-Schnittstellen ausgestattet, weil die Hersteller dieser Geräte den Preis für einen anderen Standard scheuen, der kaum besser sein dürfte, als was sie momentan haben. Auch die Verbraucher sind auf Standardwechsel nicht erpicht.

3.7 Schnittstellen

Ein typisches kleines bis mittleres Computersystem besteht aus einem CPU-Chip, Speicherchips und einigen E/A-Controllern, die alle durch einen Bus miteinander verbunden sind. Speicher, CPUs und Busse haben wir bereits mehr oder weniger ausführlich behandelt. Nun ist es an der Zeit, den letzten Teil des Puzzles – die E/A-Chips – zu betrachten. Über diese Chips kann ein Computer mit der Außenwelt kommunizieren.

3.7.1 E/A-Chips

E/A-Chips sind in Hülle und Fülle erhältlich und laufend kommen neue hinzu. Zu den Standardchips gehören UART, USART, CRT-Controller, Festplattencontroller und PIO. Ein **UART** (**Universal Asynchronous Receiver Transmitter**) ist ein Chip, der ein Byte vom Datenbus lesen und es bitweise über eine serielle Leitung an ein Terminal ausgeben oder Daten von einem Terminal entgegennehmen kann. UARTs unterstützen verschiedene Geschwindigkeiten von 50 bis 19.200 Bit/s, Zeichenbreiten von 5 bis 8 Bits, 1, 1,5 oder 2 Stoppbits und gerade, ungerade oder keine Parität. Alle diese Parameter lassen sich per Programm einstellen. Ein **USART** (**Universal Synchronous Asynchronous Receiver Transmitter**) beherrscht synchrone Übertragungen nach verschiedenen Protokollen sowie alle UART-Funktionen. Da Kapitel 2 bereits auf UART-Chips eingegangen ist, betrachten wir nun die parallele Schnittstelle als Beispiel für einen E/A-Chip.

PIO-Chips

Abbildung 3.55 zeigt den Schaltkreis 8255A von Intel als typischen Vertreter eines **PIO** (**Parallel Input/Output**)-Chips. Er hat 24 E/A-Leitungen, die für TTL-Pegel ausgelegt sind und z.B. Tastaturen, Schalter, Lampen oder Drucker ansteuern können. Kurz gesagt kann das CPU-Programm auf jede Leitung eine 0 oder eine 1 schreiben oder den Eingangsstatus einer Leitung lesen, sodass sich dieser Schaltkreis sehr flexibel einsetzen lässt. Ein kleines, CPU-basiertes System mit einem PIO-Schaltkreis kann oft eine ganze, mit SSI- oder MSI-Chips voll bestückte Platine ersetzen.

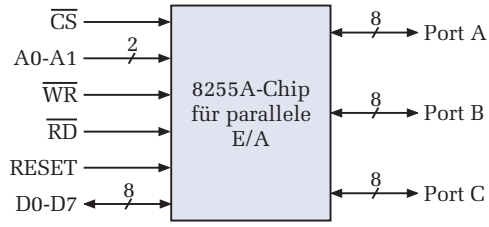


Abbildung 3.55: Ein PIO-Chip vom Typ 8255A

Obwohl die CPU den 8255A auf unterschiedlichste Weise über Statusregister innerhalb des Chips konfigurieren kann, konzentrieren wir uns hier auf einige der einfacheren Betriebsarten. Am einfachsten lässt sich der 8255A in Form von drei unabhängigen 8-Bit-Ports A, B und C nutzen. Jedem Port ist ein 8-Bit-Zwischenregister (Latch) zugeordnet. Um die Leitungen auf einem Port zu setzen, schreibt die CPU eine 8-Bit-Zahl in das entsprechende Register. Dann erscheint die 8-Bit-Zahl auf den Ausgangsleitungen und bleibt dort, bis das Register erneut beschrieben wird. Um einen Port für die Eingabe zu verwenden, liest die CPU einfach das entsprechende Register.

Andere Betriebsarten bieten Handshaking mit externen Geräten. Um beispielsweise eine Ausgabe an ein nicht ständig zur Datenaufnahme bereites Gerät zu schicken, kann der 8255A an einen Ausgangsport Daten legen und warten, bis das Gerät mit einem Rückmeldeimpuls anzeigt, dass es die Daten akzeptiert hat und auf weitere Daten wartet. Die erforderliche Logik, um derartige Impulse zwischenzuspeichern und sie für die CPU bereitzustellen, ist in der 8255A-Hardware enthalten.

Aus dem Funktionsdiagramm des 8255A ist ersichtlich, dass er zusätzlich zu den 24 Pins für die drei Ports noch über acht Leitungen verfügt, die direkt mit dem Datenbus, einer Chipauswahlleitung, Lese- und Schreibleitungen, zwei Adressleitungen und einer Leitung zum Zurücksetzen des Chips verbunden sind. Die beiden Adressleitungen wählen eines der vier internen Register aus, die den Ports A, B, C und dem Statusregister entsprechen. Die Bits des Statusregisters legen die Ports für Eingabe und Ausgabe sowie andere Funktionen fest. Normalerweise sind die beiden Adressleitungen mit den niederwertigen Bits des Adressbus verbunden.

3.7.2 Dekodierung von Adressen

Bis jetzt haben wir absichtlich ungenaue Aussagen darüber gemacht, wie die Chipauswahl auf den bisher beschriebenen Speicher- und E/A-Chips aktiviert wird. Nun ist es an der Zeit, sich mit diesem Thema zu befassen. Betrachten wir einen einfachen eingebetteten 16-Bit-Computer, der aus einer CPU, einem 2KB × 8-EPROM für das Pro-

gramm, einem 2KB × 8-RAM für die Daten und einem PIO-Schaltkreis besteht. Dieses kleine System lässt sich als Prototyp für die Steuerung eines billigen Spielzeugs oder eines einfachen Haushaltsgeräts einsetzen. Nach der Erprobung kann der EPROM durch einen ROM ersetzt werden.

Den PIO-Schaltkreis kann man auf zweierlei Weise auswählen: als echtes E/A-Gerät oder als Teil des Speichers. Soll der PIO-Schaltkreis als E/A-Gerät arbeiten, ist eine besondere Busleitung erforderlich, die anzeigt, dass ein E/A-Gerät und nicht der Speicher angesprochen wird. Der zweite Ansatz ist die so genannte **Memory Mapped I/O**. Dabei werden den drei Ports und dem Steuerregister 4 Bytes des Speicheradressraums zugeordnet. Die Wahl einer dieser Varianten kann mehr oder weniger willkürlich geschehen. Wir entscheiden uns für Memory Mapped I/O, weil dieser Ansatz interessante Aspekte der E/A-Schnittstellen veranschaulicht.

Der EPROM und der RAM benötigen jeweils einen Adressraum von 2 KB. Der PIO-Schaltkreis belegt 4 Bytes. Da der Adressraum in unserem Beispiel 64 KB groß ist, müssen wir entscheiden, wo wir die drei Geräte einordnen wollen. Abbildung 3.56 zeigt eine Möglichkeit. Der EPROM belegt die Adressen bis 2K, der RAM von 32K bis 34K und der PIO-Schaltkreis die höchsten 4 Bytes des Adressraums, 65532 bis 65535. Aus Sicht des Programmierers macht es keinen Unterschied, welche Adressen wir benutzen. Für die Schnittstellen ist das aber von Bedeutung. Hätten wir uns entschieden, den PIO-Schaltkreis über den E/A-Adressraum anzusprechen, würde er keine Speicheradressen belegen (dafür aber vier E/A-Adressen).

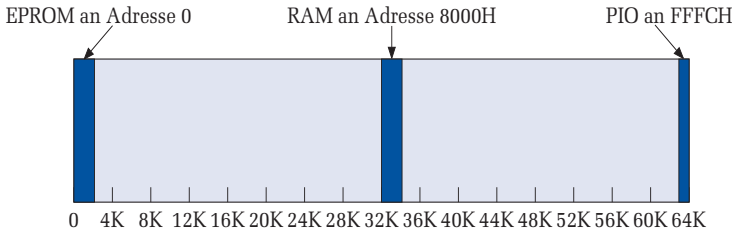


Abbildung 3.56: Speicherzuordnung für EPROM, RAM und PIO in einem 64KB-Adressraum

Mit den Adresszuordnungen nach Abbildung 3.56 soll der EPROM durch eine 16-Bit-Speicheradresse im Format 0000 0xxx xxxx xxxx (binär) ausgewählt werden. Anders ausgedrückt: Jede Adresse, deren fünf höherwertige Bits Nullen sind, fällt in die unteren 2K des Speichers, also in den EPROM-Bereich. Somit lässt sich die EPROM-Chipauswahl mit einem 5-Bit-Komparator realisieren, dessen Vergleichseingänge fest auf 00000 gelegt sind.

Die gleiche Wirkung lässt sich besser durch ein `OR`-Gatter mit fünf Eingängen realisieren, an dem die fünf Adressleitungen `A11` bis `A15` anliegen. Nur dann, wenn alle fünf Leitungen 0 sind, liegt am Ausgang des OR-Gatters 0-Potenzial, das das Signal $\overline{\text{CS}}$ aktiviert (das Low-aktiv ist). Leider ist ein `OR`-Gatter mit 5 Eingängen nicht als SSI-Standardschaltkreis verfügbar. Als nächstliegende Wahl kommt ein `NOR`-Gatter mit 8 Eingängen infrage. Um das gewünschte Signal zu erhalten, verbinden wir drei Eingänge fest mit Masse und invertieren den Ausgang, wie es Abbildung 3.57(a) zeigt. Da SSI-Chips sehr preiswert sind, ist es (abgesehen von außergewöhnlichen Umständen) kein großes Problem, wenn man einen Schaltkreis nicht vollständig ausnutzt. Der schon erwähnten Konvention entsprechend stellt man unbenutzte Eingänge in einer Schaltung nicht dar.

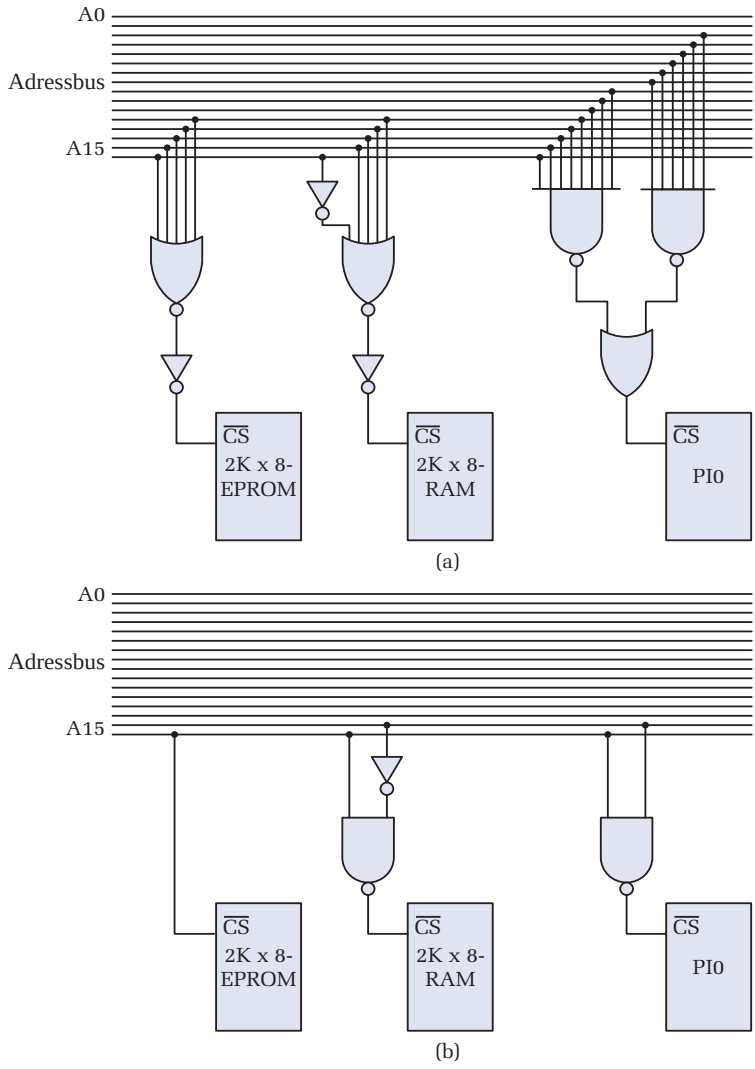


Abbildung 3.57: (a) Volle Adressdekodierung; (b) partielle Adressdekodierung

Das gleiche Prinzip lässt sich auf den RAM anwenden. Allerdings soll der RAM auf Binäradressen der Form 1000 0xxx xxxx xxxx reagieren, sodass ein zusätzlicher Inverter erforderlich ist, wie es aus der Abbildung hervorgeht. Die PIO-Adressdekodierung ist komplizierter, weil der Schaltkreis durch die vier Adressen der Form 1111 1111 1111 11xx ausgewählt wird. Die Abbildung zeigt eine mögliche Schaltung, die $\overline{\text{CS}}$ nur dann aktiviert, wenn die richtige Adresse auf dem Adressbus liegt. Die Schaltung verwendet zwei NAND-Gatter mit je 8 Eingängen, deren Ausgänge in einem OR-Gatter verknüpft werden. Die Adressdekodierung nach Abbildung 3.57(a) benötigt sechs SSI-Chips – vier Schaltkreise mit je 8 Eingängen, ein OR-Gatter und einen Schaltkreis mit drei Invertern.

Falls aber der Computer wirklich nur aus der CPU, den beiden Speicherchips und dem PIO-Schaltkreis besteht, können wir einen Trick anwenden, um die Adressdekodierung enorm zu vereinfachen. Alle EPROM-Adressen – und nur die EPROM-Adressen – enthalten im höchstwertigen Bit A15 eine 0. Somit können wir $\overline{CS}$ direkt mit A15 verbinden, wie es in Abbildung 3.57(b) zu sehen ist.

Unter diesem Aspekt sieht die Entscheidung, den RAM bei 8000H anzusiedeln, gar nicht mehr so willkürlich aus. Gültige Adressen für den RAM haben nämlich die Form 10xx xxxx xxxx xxxx. Es genügt also, die höchstwertigen 2 Bits zu dekodieren. Analog muss jede mit 11 beginnende Adresse eine PIO-Adresse sein. Die vollständige Dekodierlogik kommt nun mit zwei NAND-Gattern und einem Inverter aus. Da sich ein Inverter auch mit einem NAND-Gatter realisieren lässt, wenn man die beiden Eingänge verbindet, ist für die gesamte Schaltung ein Standardchip mit 4 NAND-Gattern (z.B. 7400) mehr als ausreichend.

Die in Abbildung 3.57(b) dargestellte Logik zur Adressdekodierung heißt **partielle Adressdekodierung** (Partial Address Decoding), weil zur Dekodierung nicht die vollständigen Adressen herangezogen werden. Eine derartige Schaltung hat die Eigenschaft, dass eine Leseoperation von den Adressen 0001 0000 0000 0000, 0001 1000 0000 0000 oder 0010 0000 0000 0000 zum selben Ergebnis führt. Jede Adresse in der unteren Hälfte des Adressraums wählt den EPROM aus. Da die zusätzlichen Adressen anderweitig nicht genutzt werden, kann nichts passieren. Will man aber einen Computer auf künftige Erweiterung auslegen (was bei einem Spielzeug kaum der Fall sein dürfte), sollte man die partielle Dekodierung vermeiden, weil sie zuviel Adressraum belegt.

Es ist auch üblich, die Adressdekodierung mit einem Dekodierer zu realisieren, wie er beispielsweise in Abbildung 3.11 zu sehen ist. Wenn man die drei Eingänge mit den drei höchstwertigen Adressleitungen verbindet, erhält man auch Ausgänge, die den Adressen in den ersten 8K, den zweiten 8K usw. entsprechen. Für einen Computer mit acht RAMs zu je 8K × 8 lässt sich mit einem derartigen Chip die gesamte Dekodierung erledigen. Auch für einen Computer mit acht 2K × 8-Speicherchips genügt ein Dekodierer, vorausgesetzt, dass sich die Speicherchips jeweils an getrennten 8K-Blöcken des Adressraums befinden. (Wie weiter oben angemerkt, spielt die Lage der Speicher- und E/A-Chips innerhalb des Adressraums eine Rolle.)

ZUSAMMENFASSUNG

Computer werden aus integrierten Schaltkreisen aufgebaut, die winzige Schaltelemente – die so genannten Gatter – enthalten. Gebräuchliche Gatterfunktionen sind AND, OR, NAND, NOR und NOT. Einfache Schaltungen lassen sich direkt durch Verknüpfen einzelner Gatter realisieren.

Zu den komplexeren Schaltkreisen gehören Multiplexer, Demultiplexer, Kodierer, Dekodierer, Schieberegister und ALUs. Mit einem PLA kann man beliebige boolesche Funktionen programmieren. Benötigt man viele boolesche Funktionen, sind PLAs oftmals effizienter. Mithilfe der Gesetze der booleschen Algebra lassen sich Schaltungen von einer Form in eine andere überführen. In vielen Fällen kann man auf diese Weise Schaltungen erstellen, die sich wirtschaftlicher (d.h. mit einfachen Standardschaltkreisen) realisieren lassen.

Der Grundbaustein für Computerarithmetik ist der Addierer. Ein Einzelbit-Volladdierer lässt sich aus zwei Halbaddierern aufbauen. Um einen Addierer für Mehrbitworte zu erhalten, schaltet man mehrere Volladdierer so zusammen, dass der Übertrag jeder einzelnen Bitstelle in ihre linke Nachbarstelle eingespeist wird.

Die Komponenten von (statischen) Speichern sind Latches und Flipflops, die jeweils ein Bit speichern können. Diese Schaltungen lassen sich linear zu 8-fach-Latches und -Flipflops oder logarithmisch zu ausgewachsenen wortorientierten Speichern kombinieren. Als Speichertypen sind RAM, ROM, PROM, EPROM, EEPROM und Flash-Speicher erhältlich. Statische RAMs müssen nicht aufgefrischt werden; sie behalten ihre Werte, solange die Betriebsspannung anliegt. Dagegen ist es bei dynamischen RAMs erforderlich, die Inhalte periodisch aufzufrischen, um die (durch Kriechströme verursachten) Ladungsverluste der winzigen Kondensatoren auf dem Chip zu kompensieren.

Die Komponenten eines Computersystems werden durch Busse miteinander verbunden. Die Pins einer typischen CPU sind zum Teil direkt mit den Busleitungen verbunden. Die Busleitungen lassen sich in Adress-, Daten- und Steuerleitungen unterteilen. Synchrone Busse werden von einem Mastertakt gesteuert. Asynchrone Busse arbeiten mit vollständigem Handshaking, um den Slave mit dem Master zu synchronisieren.

Der Pentium 4 ist ein Beispiel für einen modernen Prozessor. Moderne Systeme, die mit diesem Prozessor ausgestattet sind, verfügen über einen Speicherbus, einen PCI-Bus, einen ISA-Bus und einen USB-Bus. Der PCI-Bus kann 64 Bits gleichzeitig bei einer Taktfrequenz von 66 MHz übertragen. Damit ist er ausreichend schnell für fast alle Peripheriegeräte, aber nicht schnell genug für den Speicher.

Schalter, Lampen, Drucker und viele andere E/A-Geräte kann man über parallele Schnittstellen an den Computer anschließen. Dafür sind spezielle E/A-Chips wie der 8255A verfügbar. Diese Chips lassen sich konfigurieren und können je nach Bedarf über den E/A-Adressraum oder den Speicheradressraum angesprochen werden. Um die Chips auszuwählen, kann man die Adressen abhängig von der konkreten Anwendung vollständig oder partiell dekodieren.

ZUSAMMENFASSUNG

Aufgaben

Die Lösungen zu den Aufgaben finden Sie auf der Companion Website (CWS) unter *www.pearson-studium.de*.

1. Ein Logiker fährt in ein Drive-in-Restaurant und sagt: „Ich möchte einen Hamburger oder ein Hot-Dog und Pommes Frites." Der Koch hat die Schule zu häufig geschwänzt und weiß nicht (oder kümmert sich nicht darum), ob „und" Vorrang vor „oder" hat. Für ihn ist das eine so gut wie das andere. Welche der folgenden Fälle sind gültige Interpretationen der Bestellung? (Beachten Sie, dass hier das Wort „oder" im Sinne von „entweder oder" verwendet wird.)

a. Nur ein Hamburger.
b. Nur ein Hot-Dog.
c. Nur Pommes Frites.
d. Ein Hot-Dog und Pommes Frites.
e. Ein Hamburger und Pommes Frites.
f. Ein Hot-Dog und ein Hamburger.
g. Alle drei.
h. Nichts – der Logiker fährt hungrig von dannen, weil er zu schlau war.

2. Ein Missionar hat sich in Südkalifornien verirrt und stoppt an einer Kreuzung. Er weiß, dass zwei Motorradbanden die Gegend unsicher machen. Eine davon sagt immer die Wahrheit, und die andere lügt immer. Er möchte wissen, welche Straße nach Disneyland führt. Welche Frage sollte er stellen?

3. Zeigen Sie mithilfe einer Wahrheitstabelle, dass X = (X AND Y) OR (X AND NOT Y) gilt.

4. Es gibt vier boolesche Funktionen für eine einzelne Variable und 16 Funktionen für zwei Variablen. Wie viele Funktionen von drei Variablen gibt es? Verallgemeinern Sie die Aussage für n Variablen.

5. Zeigen Sie, wie sich die AND-Funktion aus zwei NAND-Gatter konstruieren lässt.

6. Verwenden Sie den Multiplexerchip für drei Variablen gemäß Abbildung 3.10 und implementieren Sie eine Funktion, die die Parität für die Eingabesignale ausgibt, d.h. den Ausgang nur dann auf 1 setzt, wenn eine gerade Anzahl von Eingängen auf 1 gesetzt ist.

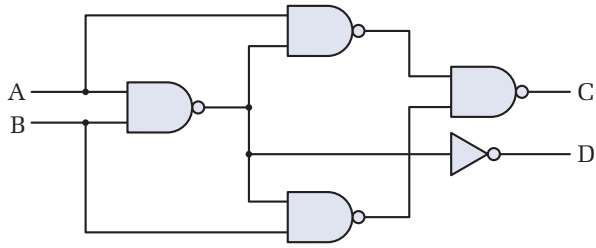
7. Denken Sie mal scharf nach: Der Multiplexerchip für drei Variablen von Abbildung 3.10 ist eigentlich in der Lage, eine beliebige Funktion von *vier* booleschen Variablen zu berechnen. Geben Sie eine Beschreibung und als Beispiel das Logikdiagramm für die Funktion an, die 0 ist, falls das englische Wort für die Zeilennummer in der Wahrheitstabelle eine gerade Zahl von Buchstaben hat, und 1 ist, falls sie eine ungerade Zahl von Buchstaben hat (z.B. 0000 = zero = vier Buchstaben → 0; 0111 = seven = fünf Buchstaben → 1; 1101 = thirteen = acht Buchstaben → 0). *Hinweis:* Wenn wir die vierte Eingabevariable D nennen, können die acht Eingangsleitungen mit U_{CC}, Masse, D oder $\overline{D}$ verdrahtet werden.

8. Zeichnen Sie das Logikdiagramm für einen 2-Bit-Kodierer, d.h. eine Schaltung mit vier Eingangsleitungen, von denen immer nur genau eine auf High liegt, und zwei Ausgangsleitungen, deren 2-Bit-Binärwert angibt, welcher Eingang auf High liegt.

9. Zeichnen Sie das Logikdiagramm für einen 2-Bit-Demultiplexer, d.h. eine Schaltung, deren einzige Eingangsleitung zu einer der vier Ausgangsleitungen je nach Zustand der beiden Steuerleitungen durchgeschaltet wird.

10. Zeichnen Sie das PLA von Abbildung 3.13 neu, um detailliert zu zeigen, wie die logische Mehrheitsfunktion von Abbildung 3.3 implementiert werden kann. Stellen Sie vor allem dar, welche Verbindungen in beiden Matrizen vorhanden sind.

11. Was bewirkt diese Schaltung?



12. Ein 4-Bit-Addierer ist als MSI-Standardchip erhältlich. Vier dieser Chips können zu einem 16-Bit-Addierer verbunden werden. Wie viele Pins erwarten Sie für den 4-Bit-Addiererchip? Warum?

13. Um einen n-Bit-Addierer aufzubauen, kann man n Volladdierer in Reihe kaskadieren, wobei der Übertrag C_i in Stufe i vom Ausgang der Stufe $i-1$ kommt. Der Übertrag C_0 in Stufe 0 ist 0. Wenn jede Stufe T ns benötigt, um Summe und Übertrag zu ermitteln, ist der Übertrag zu Stufe i erst nach iT ns ab dem Beginn der Addition gültig. Bei großen n ist die Zeit, die der Übertrag zum Durchlaufen der gesamten Kette bis in die höchste Stufe benötigt, gegebenenfalls nicht akzeptabel. Entwerfen Sie einen Addierer, der schneller arbeitet. *Hinweis:* Jeder Übertrag C_i lässt sich durch die Operandenbits A_{i-1} und B_{i-1} sowie den Übertrag C_{i-1} ausdrücken. Mit dieser Beziehung ist es möglich, C_i als Funktion der Eingaben in die Stufen 0 bis $i-1$ darzustellen, sodass alle Überträge gleichzeitig erzeugt werden können.

14. Wann kann eine Schaltung nach Abbildung 3.17 frühestens ein gültiges Ausgabebit gewährleisten, wenn man für alle Gatter eine Laufzeit von 1 ns annimmt und alle anderen Verzögerungen ignoriert werden können?

15. Die ALU von Abbildung 3.18 kann Zweierkomplementadditionen mit 8 Bit ausführen. Ist sie auch in der Lage, Zweierkomplementsubtraktionen auszuführen? Falls ja, erklären Sie, wie. Ändern Sie andernfalls die ALU entsprechend ab, sodass sie auch für Subtraktionen geeignet ist.

16. Eine 16-Bit-ALU besteht aus 16 1-Bit-ALUs, die jeweils eine Additionszeit von 10 ns haben. Angenommen, es gibt eine zusätzliche Verzögerung von 1 ns für die Verbreitung von einer ALU zur nächsten. Wie lange dauert es, bis das Ergebnis einer 16-Bit-Addition erscheint?

17. Manchmal ist es nützlich, dass eine 8-Bit-ALU wie in Abbildung 3.18 die Konstante –1 als Ausgabe erzeugt. Zeigen Sie zwei unterschiedliche Wege, wie sich dies realisieren lässt. Geben Sie für jede Lösung die Werte der sechs Steuersignale an.

18. Wie lautet der Ruhezustand der Eingänge S und R in einem SR-Latch, das aus zwei NAND-Gattern aufgebaut ist?

19. Die Schaltung von Abbildung 3.24 ist ein Flipflop, das an der steigenden Flanke des Takts ausgelöst wird. Ändern Sie die Schaltung so ab, dass das Flipflop an der fallenden Flanke des Takts ausgelöst wird.

20. Der 4 × 3-Speicher von Abbildung 3.27 ist mit 22 AND-Gattern und drei OR-Gattern aufgebaut. Wie viele Gatter sind von jedem Typ erforderlich, wenn man die Schaltung auf 256 × 8 erweitert?

21. Um die Raten für Ihren neuen Personalcomputer schneller tilgen zu können, werden Sie als Berater für Firmenneugründungen tätig, die SSI-Chips herstellen. Einer Ihrer Kunden überlegt, ob er für einen potenziell wichtigen Abnehmer einen Chip mit vier D-Flipflops entwerfen soll, die jeweils Q und $\overline{Q}$ enthalten. Gemäß Pflichtenheft sind im vorgeschlagenen Entwurf alle vier Taktsignale miteinander verbunden. Es gibt weder Preset noch Clear. Ihre Aufgabe ist es, eine professionelle Einschätzung des Designs zu geben.

22. Je mehr Speicher auf einen einzigen Chip gequetscht wird, um so höher ist auch die Anzahl der für die Adressierung benötigten Pins. Oftmals ist eine große Zahl von Adresspins an einem Chip unpraktisch. Geben Sie eine Möglichkeit an, um 2^n Speicherworte mit weniger als n Pins zu adressieren.

23. Ein Computer mit einem 32 Bit breiten Datenbus benutzt dynamische 1M × 1-RAM-Speicherchips. Geben Sie den kleinsten Speicher (in Byte) an, den dieser Computer haben kann.

24. Verwenden Sie das Taktdiagramm von Abbildung 3.34 und nehmen Sie an, dass Sie den Takt auf eine Impulsdauer von 20 ns statt der angegebenen 10 ns herabgesetzt haben, die Zeitbeschränkungen aber unverändert geblieben sind. Wie viel Zeit steht dem Speicher im ungünstigsten Fall zur Verfügung, um die Daten während T_3 nach der Aktivierung von $\overline{\text{MREQ}}$ auf den Bus zu legen?

25. Verwenden Sie wieder das Taktdiagramm von Abbildung 3.34 und nehmen Sie an, dass der Takt unverändert bei 100 MHz bleibt, aber T_{DS} auf 4 ns angehoben wurde. Lassen sich jetzt noch 10-ns-Speicherchips einsetzen?

26. In Abbildung 3.34(b) ist T_{ML} mit mindestens 2 ns angegeben. Können Sie sich einen Chip vorstellen, bei dem diese Zeit negativ ist? Anders ausgedrückt: Könnte die CPU $\overline{\text{MREQ}}$ aktivieren, bevor die Adresse stabil ist? Warum bzw. warum nicht?

27. Nehmen Sie an, dass der Blocktransfer von Abbildung 3.38 auf dem Bus von Abbildung 3.34 erfolgt. Wie viel mehr Bandbreite erhält man, wenn man statt einzelner Übertragungen einen Blocktransfer mit langen Blöcken verwendet? Nehmen Sie jetzt an, dass der Bus nicht 8 Bit, sondern 32 Bit breit ist. Beantworten Sie die Frage erneut.

28. Beschriften Sie die Übergangszeiten der Adressleitungen von Abbildung 3.35 mit T_{A1} und T_{A2}, die Übergangszeiten von $\overline{\text{MREQ}}$ mit T_{MREQ1} und T_{MREQ2} usw. Schreiben Sie alle durch das volle Handshake implizierten Ungleichheiten auf.

29. Die meisten 32-Bit-Busse erlauben 16-Bit-Lese- und Schreiboperationen. Steht es eindeutig fest, wo die Daten zu platzieren sind?

30. Viele CPUs verfügen über einen speziellen Buszyklustyp, um Interrupts zu bestätigen. Warum?

31. Ein 64-Bit-Computer mit einem 200-MHz-Bus erfordert vier Zyklen, um ein 64-Bit-Wort zu lesen. Wie viel Busbandbreite verbraucht die CPU im ungünstigsten Fall?

32. Eine 32-Bit-CPU mit den Adressleitungen A2–A31 verlangt, dass alle Speicherreferenzen ausgerichtet werden. Das heißt, Worte müssen an Vielfachen von 4 Bytes und Halbworte an geraden Adressen (Vielfachen von 2 Bytes) liegen. Bytes dürfen sich an beliebigen Positionen befinden. Wie viele zulässige Kombinationen gibt es für Speicherleseoperationen und wie viele Pins sind erforderlich, um sie auszudrücken? Geben Sie zwei Antworten mit je einem Fallbeispiel an.

33. Warum kann der Pentium 4 nicht an einem 32-Bit-PCI-Bus arbeiten, ohne an Funktionalität einzubüßen? Immerhin beherrschen andere Computer mit einem 64-Bit-Datenbus auch Transfers mit Breiten von 32 Bit, 16 Bit und sogar 8 Bit.

34. Eine CPU habe einen Level-1-Cache und einen Level-2-Cache mit Zugriffszeiten von 1 bzw. 2 ns. Die Hauptspeicherzugriffszeit beträgt 50 ns. Wie groß ist die durchschnittliche Zugriffszeit, wenn 20% der Zugriffe L1-Cache-Treffer und 60% der Zugriffe L2-Cache-Treffer sind?

35. Ist es wahrscheinlich, dass ein kleines, eingebettetes 8051-System einen 8255A-Chip beinhaltet?

36. Berechnen Sie die erforderliche Busbreite, um einen Videofilm mit 30 Frames/s (VGA, 640 × 480, TrueColor) anzuzeigen. Gehen Sie davon aus, dass die Daten zweimal den Bus passieren müssen, einmal von der CD-ROM zum Speicher und einmal vom Speicher zum Bildschirm.

37. Welches Pentium 4-Signal steuert Ihrer Meinung nach die Leitung FRAME# des PCI-Busses?

38. Welches der Signale in Abbildung 3.51 ist nicht unbedingt notwendig, damit das Busprotokoll funktioniert?

39. Ein PCI-Express-System verfügt über 5 Mbit/s-Verbindungen (Bruttokapazität). Wie viele Signalleitungen sind in jeder Richtung für eine 8x-Betriebsart erforderlich? Wie groß ist jeweils die Bruttokapazität? Wie groß ist jeweils die Nettokapazität?

40. Ein Computer hat Befehle, die je zwei Buszyklen erfordern – einen zum Holen des Befehls und einen zum Holen der Daten. Jeder Buszyklus dauert 10 ns und jeder Befehl 20 ns (d.h., die interne Verarbeitungszeit ist zu vernachlässigen). Außerdem verfügt der Computer über eine Festplatte mit 2048 Sektoren (zu je 512 Byte) pro Spur. Die Zeit für eine Umdrehung der Platte beträgt 5 ms. Auf wie viel Prozent der ursprünglichen Geschwindigkeit wird der Computer während eines DMA-Transfers gebremst, wenn jeder 32-Bit-DMA-Transfer einen Buszyklus dauert?

41. Auf dem USB-Bus lassen sich in einem isochronen Datenpaket maximal 1023 Byte Nutzdaten transportieren. Wie groß ist die maximale Bandbreite für ein einzelnes isochrones Gerät, wenn man annimmt, dass ein Gerät nur ein Datenpaket pro Frame senden kann?

42. Was passiert, wenn man in der Schaltung von Abbildung 3.57(b) für die Auswahl des PIO-Schaltkreises ein NAND-Gatter mit 3 statt 2 Eingängen verwendet und den dritten Eingang mit A13 verbindet?

43. Schreiben Sie ein Programm, um das Verhalten eines $m \times n$-Arrays aus NAND-Gattern mit je zwei Eingängen zu simulieren. Diese auf einem Chip untergebrachte Schaltung hat j Eingangs- und k Ausgangspins. Die Werte von j, k, m und n werden als Parameter zur Kompilierzeit des Simulationsprogramms eingerichtet. Das Programm soll zunächst eine „Verdrahtungsliste“ einlesen, in der jede Leitung einen Eingang und einen Ausgang spezifiziert. Ein Eingang ist entweder einer der j Eingangspins oder der Ausgang eines NAND-Gatters. Ein Ausgang ist entweder einer der k Ausgangspins oder ein Eingang eines NAND-Gatters. Unbenutzte Eingänge sind logisch 1. Nach dem Lesen der Verdrahtungsliste soll das Programm die Ausgangswerte für alle möglichen 2^j Eingangsbelegungen ausgeben. Derartige Gatearray-Chips sind für kundenspezifische Schaltungen auf einem Chip weit verbreitet, weil der größte Teil des Herstellungsprozesses (Realisieren des Gatearrays auf dem Chip) unabhängig von der zu implementierenden Schaltung ist. Nur die Verdrahtung ist designspezifisch.

44. Schreiben Sie ein Programm, das zwei beliebige boolesche Ausdrücke einliest und bestimmt, ob sie die gleiche Funktion darstellen. Für die Eingabesprache sind einzelne Buchstaben als boolesche Variablen, die Operanden `AND`, `OR` und `NOT` sowie Klammern vorzusehen. Jeder Ausdruck muss auf eine Eingabezeile passen. Das Programm soll die Wahrheitstabellen für beide Funktionen berechnen und vergleichen.

45. Schreiben Sie ein Programm, das eine Liste von booleschen Ausdrücken einliest und die zu ihrer Implementierung erforderlichen beiden Matrizen (24×50 und 50×6) für das PLA in Abbildung 3.13 berechnet. Verwenden Sie die gleiche Eingabesprache wie in Aufgabe 44. Geben Sie die Matrizen auf einem Zeilendrucker aus.