3

Die digitale logische Ebene

Unten in der Hierarchie von Abb. 1.2 liegt die digitale logische Ebene – die echte Computer-Hardware. In diesem Kapitel untersuchen wir viele Aspekte der digitalen Logik als Baustein für die Studie der höheren Ebenen, die in späteren Kapiteln behandelt werden. Dieses Thema liegt an der Grenze zwischen Informatik und Elektrotechnik. Der Stoff wird aber in sich geschlossen und selbsterklärend vorgestellt, so daß keine Kenntnisse über Hardware oder Technik erforderlich sind.

Die Grundelemente, aus denen alle digitalen Computer gebaut werden, sind erstaunlich einfach. Wir beginnen unsere Untersuchung mit einer Betrachtung dieser Grundelemente und der speziellen, zweiwertigen Algebra (boolesche Algebra), die zur Analyse dieser Elemente benutzt wird. Anschließend untersuchen wir einige grundlegenden Schaltungen, die mit Hilfe von Gates in einfachen Kombinationen gebaut werden können, darunter Schaltungen für die Durchführung von Arithmetik. Das nächste Thema ist die Kombination von Gates, um Informationen zu speichern, d.h. Speicherorganisation. Danach befassen wir uns mit dem Thema »CPU«, insbesondere damit, wie Einzelchip-CPUs Daten mit Speichern und Peripheriegeräten austauschen. Zahlreiche Beispiele aus der Industrie werden in späteren Abschnitten dieses Kapitels betrachtet.

3.1 Gates und boolesche Algebra

Digitale Schaltungen lassen sich aus einer kleinen Zahl von primitiven Elementen bauen, indem man diese auf vielfache Art und Weise kombiniert. In den folgenden Abschnitten werden diese primitiven Elemente beschrieben. Außerdem wird erklärt, wie sie sich kombinieren lassen. Es wird weiter eine mächtige mathematische Methode vorgestellt, mit der man das Verhalten dieser Elemente analysieren kann.

3.1.1 Gates

Bei einer digitalen Schaltung gibt es nur zwei logische Werte. Normalerweise stellt ein Signal zwischen 0 und 1 Volt einen Wert (z.B. eine binäre 0) und ein Signal zwischen 2 und 5 Volt einen anderen Wert (z.B. eine binäre 1) dar. Spannungen außerhalb dieser beiden Bereiche sind nicht zulässig. Winzige elektronische Geräte namens **Gates** (Gatter) können mit diesen zweiwertigen Signalen verschiedene Funktionen berechnen. Diese Gates bilden die Hardware-Grundlage, auf der alle digitalen Computer basieren.

Die Einzelheiten darüber, wie diese Gates im Innern funktionieren, sind nicht Gegenstand dieses Buchs, sondern gehören zur **Geräteebene** (Device Level),

die unterhalb unserer Ebene 0 liegt. Wir schweifen jetzt aber dennoch kurz ab und werfen einen kurzen Blick auf das – einfache – Grundkonzept. Jede moderne digitale Logik beruht letztendlich auf der Tatsache, daß ein Transistor so gebaut werden kann, daß er als sehr schneller, binärer Schalter arbeitet. In Abb. 3.1(a) ist ein einzelner bipolarer Transistor (der Kreis) in eine einfache Schaltung eingebettet. Dieser Transistor hat drei Verbindungen mit der Außenwelt: den **Kollektor**, die **Basis** und den **Emitter**. Liegt die Eingangsspannung V_{in} unter einem bestimmten kritischen Wert, schaltet sich der Transistor aus und verhält sich wie ein unendlicher Widerstand. Dies veranlaßt den Ausgang der Schaltung V_{out}, einen Wert nahe V_{cc} – eine extern geregelte Spannung – anzunehmen, die normalerweise bei diesem Transistortyp bei +5 Volt liegt. Überschreitet V_{in} den kritischen Wert, schaltet sich der Transistor ein und funktioniert wie ein Draht, wodurch V_{out} nach unten zur Masse (gemäß Konvention 0 Volt) gezogen wird.

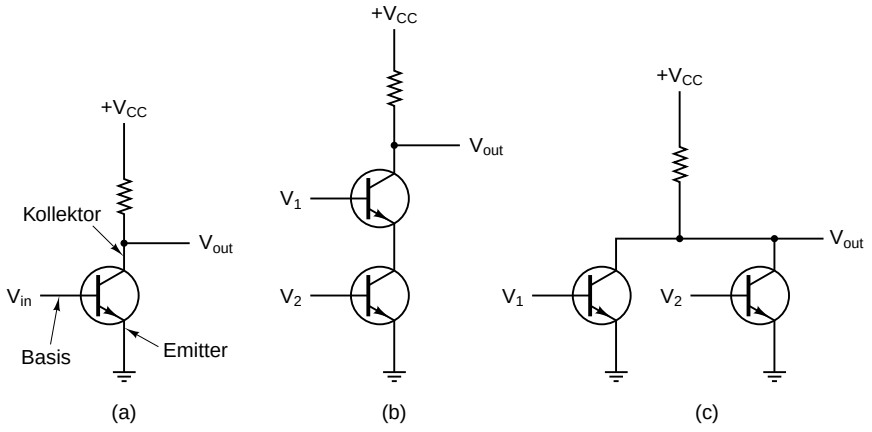


Abb. 3.1: (A) Transistor-Inverter; (b) NAND-Gate; (c) NOR-Gate

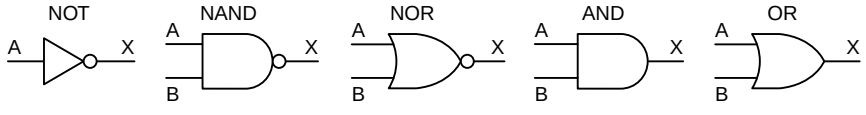
Wichtig dabei ist folgendes: Wenn V_{in} auf Low liegt, liegt V_{out} auf High, und umgekehrt. Folglich ist diese Schaltung ein Inverter, der eine logische 0 in eine logische 1 und eine logische 1 in eine logische 0 konvertiert. Der Widerstand (Resistor) – die gezackte Linie – ist nötig, um den vom Transistor gezogenen Strom zu begrenzen. Die Zeit, die benötigt wird, um von einem Zustand in den anderen umzuschalten, beträgt normalerweise ein paar Nanosekunden.

Abb. 3.1(b) zeigt zwei in Reihe geschaltete Transistoren. Liegen V_1 und V_2 auf High, leiten beide Transistoren, und V_{out} wird nach unten gezogen. Liegt einer der beiden Eingänge auf Low, schaltet der entsprechende Transistor aus, und der Ausgang geht auf High. Anders ausgedrückt: V_{out} ist auf Low, aber nur dann, wenn sowohl V_1 als auch V_2 auf High liegen.

Die beiden Transistoren in Abb. 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 nach unten zur Masse. Sind beide Eingänge auf Low, bleibt der Ausgang auf High.

Diese drei Schaltungen bzw. ihre Gegenstücke bilden die drei einfachsten Gates. Sie heißen NOT, NAND und NOR. NOT-Gates nennt man oft **Inverter**. Wir benutzen die beiden Begriffe als Synonyme. Wenden wir nun die Konvention an, daß »High« (V_{cc} Volt) eine logische 1 und »Low« (Masse) eine logische 0 ist, können wir den Ausgangswert als Funktion der Eingangswerte ausdrükken. Abb. 3.2(a) bis (c) zeigt die üblicherweise für die Darstellung dieser drei Gates benutzten Symbole und die Funktion der einzelnen Schaltungen. In diesen Abbildungen sind *A* und *B* Eingänge, und *X* ist der Ausgang. Jede Reihe legt den Ausgang für eine unterschiedliche Kombination von Eingängen fest.



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)

Abb. 3.2: Symbole und funktionales Verhalten der fünf grundlegenden Gates

Wird das Ausgangssignal von Abb. 3.1(b) in eine Inverter-Schaltung eingespeist, erhalten wir eine weitere Schaltung mit genau dem Gegenteil des NAND-Gates, d.h. eine Schaltung, deren Ausgang 1 ist, wenn beide Eingänge 1 sind. Eine solche Schaltung nennt man AND-Gate. Abb. 3.2(d) zeigt das entsprechende Symbol und die funktionale Beschreibung. Ebenso kann das NOR-Gate mit einem Inverter verbunden werden, um eine Schaltung zu erzielen, deren Ausgang 1 ist, wenn entweder einer oder beide Eingänge 1 sind, aber 0 ist, wenn beide Eingänge 0 sind. Das Symbol und die funktionale Beschreibung dieser Schaltung, die man »OR-Gate« nennt, sind in Abb. 3.2(e) dargestellt. Die kleinen Kreise in den Symbolen des Inverters, des NAND-Gates und des NOR-Gates nennt man **Inversionsblasen** (Inversion Bubbles). Sie werden häufig auch in anderen Zusammenhängen benutzt, um ein invertiertes Signal zu bezeichnen.

Die fünf Gates von Abb. 3.2 sind die grundlegenden Bausteine der digitalen logischen Ebene. Aus der vorhergehenden Darstellung wird klar, daß das NAND- und NOR-Gate je zwei Transistoren benötigen, während das AND- und OR-Gate jeweils drei braucht. Aus diesem Grund basieren viele Computer auf NAND- und NOR-Gates, und nicht auf den bekannteren AND- und OR-Gates. (In der Praxis werden alle Gates etwas anders implementiert, NAND und NOR sind aber dennoch einfacher als AND und OR.) Nebenbei bemerkt, können

Gates mehr als zwei Eingänge haben. Im Prinzip kann beispielsweise ein NAND-Gate beliebig viele Eingänge haben. In der Praxis sind aber mehr als acht Eingänge ungewöhnlich.

Obwohl die Konstruktion von Gates zur Geräteebene gehört, werden hier die wichtigen Herstellungstechnologien erwähnt, auf die häufig Bezug genommen wird. Die beiden wichtigsten Technologien sind **bipolar** und **MOS** (Metal Oxide Semiconductor). Die wichtigen bipolaren Typen sind **TTL** (Transistor-Transistor Logic) – seit Jahren das Arbeitspferd der digitalen Elektronik – und **ECL** (Emitter-Coupled Logic), die benutzt wird, wenn sehr hohe Geschwindigkeiten gefordert werden.

MOS-Gates sind langsamer als TTL und ECL, erfordern aber weniger Strom und nehmen viel weniger Platz ein. Deshalb können sie in großer Zahl auf engem Raum gepackt werden. MOS gibt es in vielen Varianten, darunter PMOS, NMOS und CMOS. Während MOS-Transistoren anders als bipolare Transistoren gebaut werden, ist ihre Fähigkeit, als elektronische Schalter zu funktionieren, gleich. Die meisten modernen CPUs und Speicher basieren auf der CMOS-Technologie, die mit +3,3 Volt läuft. Das ist alles, was wir über die Geräteebene festhalten wollen. Leser, die an dieser Ebene interessiert sind, finden einschlägige Literaturhinweise in Kapitel 9.

3.1.2 Boolesche Algebra

Um die Schaltungen zu beschreiben, die durch Kombination von Gates gebaut werden können, ist eine neue Art von Algebra erforderlich – eine, bei der Variablen und Funktionen nur die Werte 0 und 1 annehmen können. Eine solche 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). Der Begriff »boolesche Algebra« hat sich aber stark eingebürgert, so daß wir hier ebenfalls nicht unterscheiden.

Genauso, wie es Funktionen in der »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* 0 ist und *f(A)* ist 0, wenn *A* 1 ist. Dies ist die NOT-Funktion von Abb. 3.2(a).

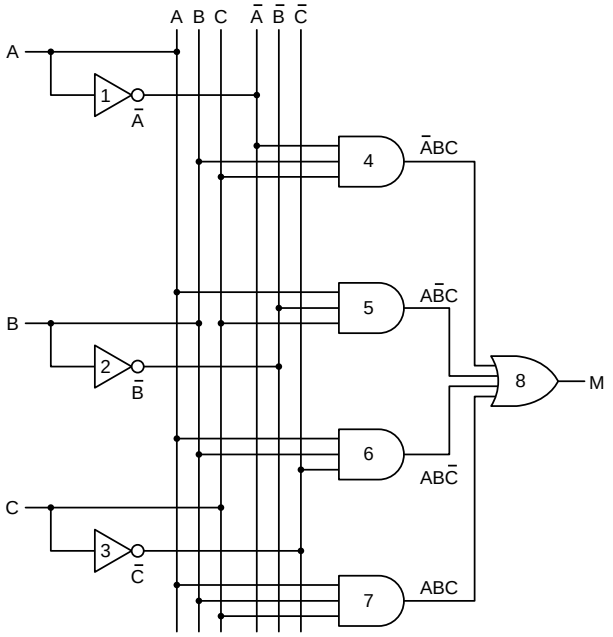
Da eine boolesche Funktion mit n Variablen nur 2^n mögliche Kombinationen von Eingabewerten hat, kann man die Funktion vollständig beschreiben, indem man eine Tabelle mit 2^n Reihen angibt, wobei jede Reihe den Wert der Funktion für eine andere Kombination von Eingangswerten angibt. Eine solche Tabelle heißt **Wahrheitstabelle** (Truth Table). Die Tabellen in Abb. 3.2 geben Beispiele. Wollen wir uns darauf festlegen, die Reihen einer Wahrheitstabelle immer in numerischer Reihenfolge (Basis 2) anzuordnen. Für zwei Variablen in der Reihenfolge 00, 01, 10 und 11 kann dann also die Funktion

vollständig mit der 2^n-Bit-Binärzahl beschrieben werden, die wir erhalten, wenn wir die Ergebnisspalte der Wahrheitstabelle vertikal lesen. 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 verfügt die gewöhnliche Algebra über eine unendliche Zahl an Funktionen von zwei Variablen, von denen sich keine dadurch beschreiben läßt, daß man eine Tabelle mit Ausgängen für alle möglichen Eingänge schreibt, weil jede Variable jeden beliebigen einer unendlichen Zahl möglicher Werte annehmen kann.

Abb. 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 mit Hilfe einer Wahrheitstabelle festlegen kann, wird diese Notation zunehmend mühsamer, je höher die Anzahl der Variablen ist. Deshalb wird häufig eine andere Notation benutzt.

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)

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

Um zu sehen, wie diese andere Notation entstanden ist, beachte man, daß jede boolesche Funktion dadurch festgelegt werden kann, daß man sagt, welche Kombinationen von Eingangsvariablen einen Ausgangswert von 1 ergeben.

Für die Funktion von Abb. 3.3(a) gibt es vier Kombinationen von Eingangsvariablen, die M den Wert 1 annehmen lassen. Der Konvention entsprechend schreiben wir die Eingangsvariablen mit einem Überstrich, um anzudeuten, daß ihr Wert invertiert ist. Fehlt der Überstrich, bedeutet das einen nicht invertierten Wert. Außerdem verwenden wir eine implizierte Multiplikation bzw. einen Punkt, um ein gemitteltes AND anzudeuten, und ein Pluszeichen (+) für ein gemitteltes OR. Das heißt z.B., daß $A\overline{B}C$ nur dann den Wert 1 annimmt, wenn $A = 1$ und $B = 0$ und $C = 1$. $A\overline{B} + B\overline{C}$ ist nur 1, wenn ($A = 1$ und $B = 0$) oder ($B = 1$ und $C = 0$). Die vier Zeilen in Abb. 3.3(a), die Einserbits produzieren, sind: $\overline{A}BC$ $A\overline{B}C$ $AB\overline{C}$ 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 Wahrheitstabelle schreiben. Eine Funktion von n Variablen kann somit dadurch beschrieben werden, daß man einer »Summe« von höchstens 2^n n-Variablen »Produktterme« gibt. Wir werden gleich sehen, daß diese Form besonders wichtig ist, weil sie direkt zu einer Implementierung der Funktion unter Verwendung von Standard-Gates 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. Sie wird mit Hilfe einer Wahrheitstabelle oder einer booleschen Funktion, wie

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

beschrieben. Eine boolesche Funktion kann in einer elektronischen Schaltung (oft auf unterschiedliche Weise) mit Hilfe von Signalen implementiert werden, die die Ein- und Ausgangsvariablen und Gates, wie AND, OR und NOT, darstellen. Wir verwenden allgemein die Notation AND, OR und NOT in bezug auf boolesche Operatoren, und AND, OR und NOT in bezug auf Gates, was oft aber zweideutig ist.

3.1.3 Implementierung von booleschen Funktionen

Wie erwähnt, führt die Formulierung einer booleschen Funktion als Summe von bis zu 2^n Produkttermen unmittelbar zu einer möglichen Implementierung. Anhand des Beispiels von Abb. 3.3 wird gezeigt, wie diese Implementierung erreicht wird. In Abb. 3.3(b) sind die Eingänge A, B und C an der linken Kante und die Ausgangsfunktion M an der rechten Kante dargestellt. Da Komplemente (Umkehrfunktionen) der Eingangsvariablen erforderlich sind, werden sie durch Abgreifen der Eingänge (Tapping) erzeugt und durch die mit 1, 2 und 3 beschrifteten Inverter weitergegeben. Damit das Bild nicht unübersichtlich wird, haben wir sechs vertikale Linien gezeichnet, von denen drei mit den Eingangsvariablen und drei mit deren Komplementen verbunden sind. Diese Linien bieten eine bequeme Quelle für die Eingänge zu nachfolgenden Gates. Beispielsweise benutzen die Gates 5, 6 und 7 alle A als Eingang.

Bei einer echten Schaltung würde man diese Gates wahrscheinlich direkt, ohne dazwischenliegende »vertikale« Drähte, mit A verbinden.

Die Schaltung enthält vier AND-Gates, je eines für jeden Term in der Gleichung für M (d.h. eines für jede Zeile der Wahrheitstabelle, die in der Ergebnisspalte ein 1-Bit hat). Jedes AND-Gate berechnet eine Zeile der Wahrheitstabelle. Schließlich werden alle Produktterme mit OR verknüpft, um das Endergebnis zu ermitteln.

Die Schaltung von Abb. 3.3(b) basiert auf einer Konvention, die wir wiederholt in diesem Buch verwenden: Wenn sich zwei Linien kreuzen, ist dies keine Verbindung, es sei denn, auf dem Schnittpunkt befindet sich ein dicker Punkt. Der Ausgang von Gate 3 kreuzt z.B. alle sechs vertikalen Linien, ist aber nur mit $\overline{C}$ verbunden. Man beachte, daß einige Autoren andere Konventionen verwenden.

Aus dem Beispiel von Abb. 3.3 wird deutlich, wie eine Schaltung für eine beliebige boolesche Funktion implementiert wird:

1. Schreibe die Wahrheitstabelle für die Funktion.
2. Erstelle Inverter, um das Komplement für jeden Eingang zu erzeugen.
3. Zeichne ein AND-Gate für jeden Term, mit einer 1 in der Ergebnisspalte.
4. Verdrahte die AND-Gates mit den entsprechenden Eingängen.
5. Speise den Ausgang aller AND-Gates in ein OR-Gate.

Damit haben wir aufgezeigt, wie eine boolesche Funktion mit Hilfe von NOT-, AND- und OR-Gates implementiert werden kann. Meist ist es aber praktischer, Schaltungen mit nur einer Sorte Gates zu implementieren. 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 mit Hilfe einer einzigen Gate-Art zu implementieren. Die oberste Zeile in Abb. 3.4 zeigt, wie alle drei Typen nur mit NAND-Gates implementiert werden können. Die unterste Zeile zeigt, wie dies nur mit NOR-Gates gelingt. (Das sind die einfachen, aber nicht die einzigen Möglichkeiten.)

Eine Möglichkeit, eine boolesche Funktion mit nur NAND- oder nur NOR-Gates zu implementieren, ist die Anwendung zuerst des oben beschriebenen Verfahrens, um sie mit NOT, AND und OR zu bilden. Dann ersetzt man die Gates mit mehreren Eingängen mit Hilfe von Gates mit zwei Eingängen durch gleichwertige Schaltungen. Benutzt man drei OR-Gates mit zwei Eingängen, kann $A + B + C + D$ beispielsweise als $(A + B) + (C + D)$ berechnet werden. Schließlich werden die NOT-, AND- und OR-Gates durch die Schaltungen von Abb. 3.4 ersetzt.

Im Sinne einer möglichst geringen Anzahl von Gates führt dieses Verfahren nicht zu optimalen Schaltungen, zeigt aber, daß es immer eine Lösung gibt. NAND- und NOR-Gates bezeichnet man als **komplett**, weil mit einem von bei-

den jede boolesche Funktion berechnet werden kann. Kein anderes Gate weist dieses Merkmal auf. Das ist ein weiterer Grund dafür, daß sie oft als Bausteine für Schaltungen benutzt werden.

3.1.4 Schaltungsäquivalenz

Entwickler von Schaltungen versuchen in der Regel, die Anzahl von Gates in ihren Produkten zu minimieren, um Komponentenkosten, PCB-Platz, Stromverbrauch usw. zu reduzieren. Zur Verringerung der Komplexität einer Schaltung muß der Entwickler eine andere Schaltung finden, die zwar die gleiche Funktion wie das Original berechnet, dies aber mit weniger Gates (oder auch einfacheren Gates, z.B. solchen mit zwei statt vier Eingängen) erreicht. Auf der Suche nach entsprechenden Schaltungen kann sich die boolesche Algebra als nützliches Werkzeug erweisen.

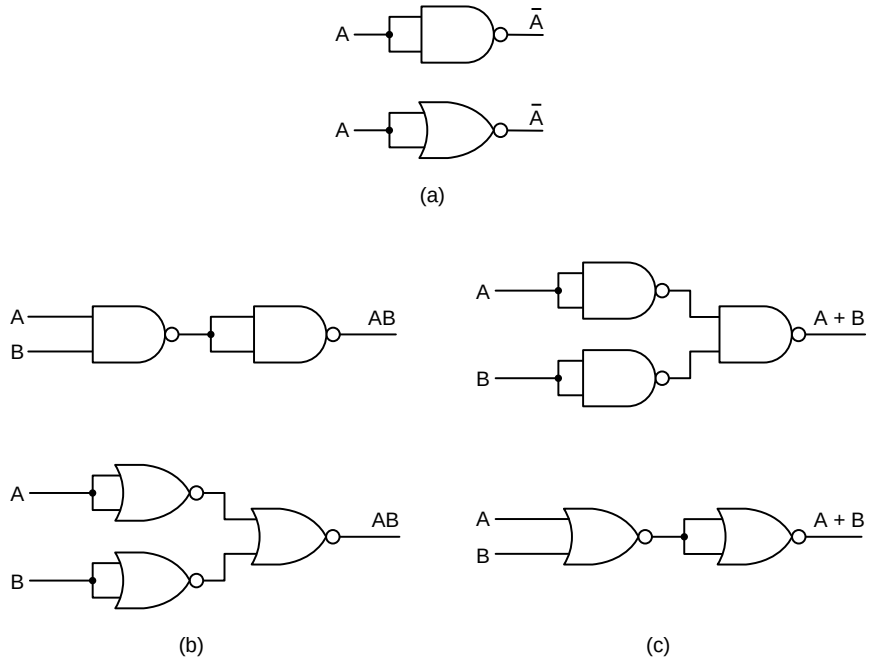
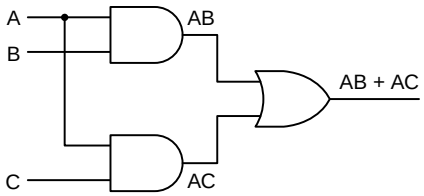


Abb. 3.4: Konstruktion von (a) NOT-, (b) AND- und (c) OR-Gates mit nur NAND- oder nur NOR-Gates

Als Beispiel dafür betrachte man die Schaltung und die Wahrheitstabelle für *AB + AC* in Abb. 3.5(a). Wir haben dies zwar noch nicht behandelt, viele Regeln der herkömmlichen Algebra treffen aber auch auf die boolesche zu. Insbesondere können *AB + AC* mit Hilfe des Distributivgesetzes in *A (B + C)* zusammengefaßt werden. Abb. 3.5(b) zeigt die Schaltung und die Wahrheitstabelle für *A (B + C)*. Da zwei Funktionen äquivalent sind, sofern sie den

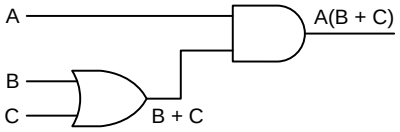
gleichen und nur den gleichen Ausgang für alle möglichen Eingänge haben, wird aus den Wahrheitstabellen von Abb. 3.5 leicht ersichtlich, daß *A (B + C)* das Äquivalent zu *AB + AC* ist. Ungeachtet dessen ist die Schaltung von Abb. 3.5(b) deutlich besser als die in Abb. 3.5(a), weil sie weniger Gates enthält.

Im allgemeinen kann ein Schaltungsentwickler eine Schaltung als boolesche Funktion darstellen und dann die Gesetze der booleschen Algebra auf diese Darstellung anwenden, um eine einfachere und gleichwertige Schaltung zu finden. Aus der endgültigen Darstellung läßt sich eine neue Schaltung konstruieren.



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)

Abb. 3.5: Zwei äquivalente Funktionen, (a) AB + AC, (b) A (B + C)

Für diesen Ansatz benötigen wir einige Identitäten aus der booleschen Algebra. Abb. 3.6 zeigt einige der wichtigen. Interessant ist, daß jedes Gesetz zwei Formen hat, die gegenseitige **Duale** sind. Durch Austausch von AND und OR und auch von 0 und 1 kann eine Form aus der anderen produziert werden. Alle diese Gesetze können leicht durch Aufstellung entsprechender Wahrheitstabellen bewiesen werden. Mit Ausnahme der De Morganschen Gleichung, des Absorptionsgesetzes und der AND-Form des Distributivgesetzes sind die Ergebnisse mehr oder weniger selbsterklärend. Die De Morganschen Gleichungen können auf mehr als zwei Variablen erweitert werden, z.B. $\overline{ABC} = \overline{A} + \overline{B} + \overline{C}$.

Bezeichnung	**AND-Form**	**OR-Form**
Identitätsgesetz	$1A = A$	$0 + A = A$
Nullgesetz	$0A = 0$	$1 + A = 1$
Idempotenzgesetz	$AA = A$	$A + A = A$
Inversionsgesetz	$A\overline{A} = 0$	$A + \overline{A} = 1$
Kommutativgesetz	$AB = BA$	$A + B = B + A$
Assoziativgesetz	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributivgesetz	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorptionsgesetz	$A(A + B) = A$	$A + AB = A$
De Morgansche Gleichungen	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A} + \overline{B} = \overline{AB}$

Abb. 3.6: Einige Identitäten der booleschen Algebra

Die De Morganschen Gleichungen legen eine andere Schreibweise nahe. In Abb. 3.7(a) wird die AND-Form mit Negation, angedeutet durch Inversionsblasen, für Ein- und Ausgang dargestellt. Somit entspricht ein OR-Gate mit invertierten Eingängen einem NAND-Gate. Aus der dualen Form der De Morganschen Gleichungen in Abb. 3.7(b) wird deutlich, daß ein NOR-Gate als AND-Gate mit invertierten Eingängen gezeichnet werden kann. Durch Negation beider Formen der De Morganschen Gleichungen gelangen wir zu Abb. 3.7(c) und (d), den äquivalenten Darstellungen der AND- und OR-Gates. Entsprechende Symbole gibt es für mehrere variable Formen der De Morganschen Gleichungen (z.B. wird aus einem NAND-Gate mit n Eingängen ein OR-Gate mit n invertierten Eingängen).

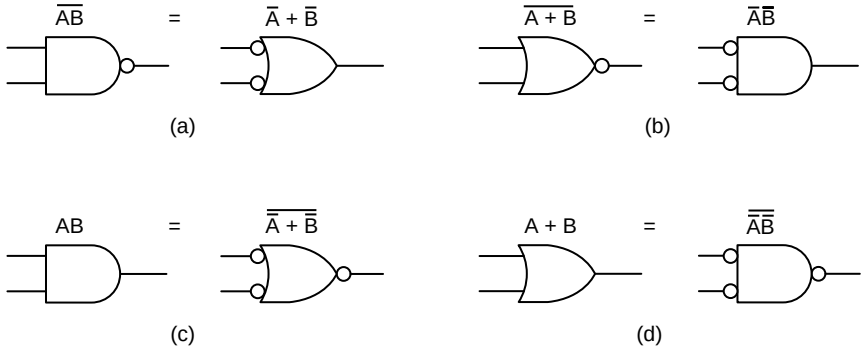


Abb. 3.7: Alternative Symbole für einige Gates: (a) NAND, (b) NOR, (c) AND, (d) OR

Mit Hilfe der Identitäten aus Abb. 3.7 und der entsprechenden für Gates mit mehreren Eingängen läßt sich die Produktsummendarstellung einer Wahrheitstabelle leicht in die reine NAND- oder die reine NOR-Form umwandeln. Als Beispiel betrachte man die EXCLUSIVE-OR-Funktion von Abb. 3.8(a). Abb.

3.8(b) zeigt die Standardproduktsummenschaltung. Um in die NAND-Form umwandeln zu können, müssen die Linien, die den Ausgang der AND-Gates mit dem Eingang des OR-Gates verbinden, mit zwei Inversionsblasen neu gezeichnet werden wie in Abb. 3.8(c). Verwenden wir nun Abb. 3.7(a), gelangen wir zu Abb. 3.8(d). Die Variablen $\overline{A}$ und $\overline{B}$ können aus A und B erzeugt werden, wenn man NAND- oder NOR-Gates mit untereinander verbundenen Eingängen nimmt. Beachten Sie, daß Inversionsblasen entlang einer Linie beliebig verschoben werden können, beispielsweise von den Ausgängen der Eingangs-Gates in Abb. 3.8(d) zu den Eingängen des Ausgangs-Gates.

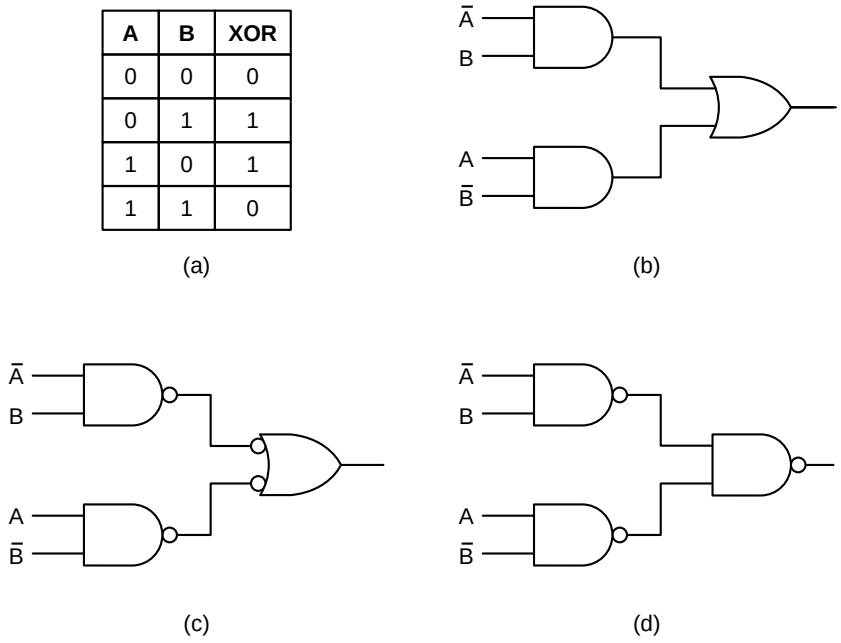


Abb. 3.8: (a) Die Wahrheitstabelle für die EXCLUSIVE-OR-Funktion; (b)–(d) drei Schaltungen zu ihrer Berechnung

Abschließend zum Thema Schaltungsäquivalenz wird nun das überraschende Ergebnis aufgezeigt, daß das gleiche physische Gate verschiedene Funktionen je nach den benutzten Konventionen berechnen kann. In Abb. 3.9(a) zeigen wir den Ausgang eines bestimmten Gates F für verschiedene Eingangskombinationen. Eingänge und Ausgänge sind in Volt dargestellt. Einigen wir uns auf die Konvention, daß 0 Volt die logische 0 und 3,3 Volt oder 5 Volt die logische 1, **positive Logik** genannt, sind, dann erhalten wir die Wahrheitstabelle von Abb. 3.9(b), die AND-Funktion. Wenden wir demgegenüber die **negative Logik** an, bei der 0 Volt die logische 1 und 3,3 Volt oder 5 Volt die logische 0 sind, erhalten wir die Wahrheitstabelle von Abb. 3.9(c) bzw. die OR-Funktion.

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, was bedeutet, daß die Begriffe »logische 1«, »wahr« und »High« sowie die Begriffe »logische 0«, »falsch« und »Low« Synonyme sind.

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

(a)

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

(b)

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

(c)

Abb. 3.9: (a) Elektrische Merkmale eines Geräts; (b) positive Logik; (c) negative Logik

3.2 Grundlegende digitale logische Schaltungen

In den vorherigen Abschnitten haben wir gesehen, wie man Wahrheitstabellen und andere einfache Schaltungen mit einzelnen Gates implementieren kann. In der Praxis werden kaum noch Schaltungen auf einer Gate-zu-Gate-Basis entwickelt, obwohl dies früher die Regel war. Heute sind Module mit einer Reihe von Gates die gängigen Bausteine. In den folgenden Abschnitten betrachten wir diese Bausteine genauer. Es wird erklärt, wie sie benutzt werden und aus einzelnen Gates entwickelt werden können.

3.2.1 Integrierte Schaltungen

Gates werden nicht einzeln, sondern in Einheiten gefertigt und verkauft, die man **integrierte Schaltungen** oder **IC** (Integrated Circuits) oder **Chips** nennt. Ein IC ist ein quadratisches Stück Silikon in der Größe von etwa 5×5 mm, das mit einigen Gates bestückt ist. Kleine ICs sind normalerweise auf einem rechteckigen Kunststoff- oder Keramikstück in der Abmessung 5 bis 15 mm breit und 20 bis 50 mm lang aufgebracht. An den langen Kanten befinden sich zwei parallele Reihen von Pins (Stiften), die etwa 5 mm lang sind, und die man in Stecker (Sockets) einstecken oder auf gedruckte Schaltkarten (Circuit Boards) aufschweißen kann. Jeder Pin paßt zum Ein- oder Ausgang eines Gates auf dem Chip oder zum Strom bzw. zur Masse. Die Einheiten mit zwei Pinreihen außen und IC innen tragen den technischen Fachausdruck **Dual Inline Packages** oder **DIP**. Jeder nennt sie aber Chips, so daß sich die Silikonstücke und die bestückten Einheiten von der Bezeichnung her nicht mehr unterscheiden. Die üblichen Einheiten haben 14, 16, 18, 20, 22, 24, 28, 40, 64 oder 68 Pins. Für größere Chips sind quadratische Einheiten mit Pins an allen vier Seiten oder an der Unterseite üblich.

Chips lassen sich nach der Anzahl von Gates, die sie aufweisen, in Klassen unterteilen. Diese Klassifizierung ist selbstverständlich sehr grob, dennoch aber nützlich:

- SSI-Schaltung (Small Scale Integrated): 1 bis 10 Gates
- MSI-Schaltung (Medium Scale Integrated): 10 bis 100 Gates
- LSI-Schaltung (Large Scale Integrated): 100 bis 100000 Gates
- VLSI-Schaltung (Very Large Scale Integrated): > 100000 Gates

Diese Klassen weisen unterschiedliche Merkmale auf und werden unterschiedlich benutzt.

Ein SSI-Chip enthält normalerweise zwei bis sechs unabhängige Gates, die je einzeln wie in früheren Abschnitten beschrieben benutzt werden können. Abb. 3.10 zeigt einen üblichen SSI-Chip mit vier NAND-Gates. Jedes dieser Gates hat zwei Eingänge und einen Ausgang, so daß insgesamt 12 Pins für die vier Gates nötig sind. Außerdem braucht der Chip Strom (V_{cc}) und Masse (GND), was beides von allen Gates benutzt wird. Die Einheit hat im allgemeinen nahe von Pin 1 eine Kerbe, die die Ausrichtung kennzeichnet. Um Schaltdiagramme nicht zu überladen, werden Strom, Masse und unbenutzte Gates normalerweise nicht dargestellt.

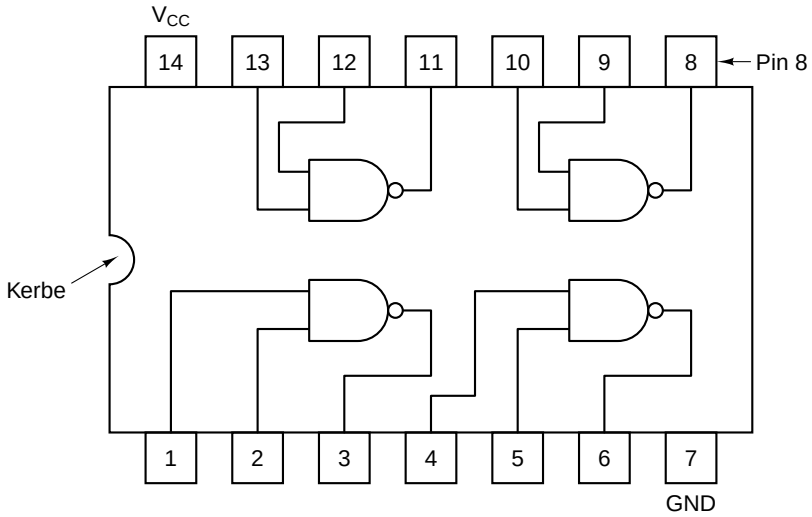


Abb. 3.10: SSI-Chip mit vier Gates

Für wenig Geld sind viele andere solcher Chips erhältlich. Jeder hat eine Handvoll Gates und bis zu 20 Pins. In den siebziger Jahren wurden Computer aus großen Mengen solcher Chips gebaut. Heute brennt man eine ganze CPU und einen großen Teil des (Cache) Speichers auf einen einzigen Chip.

Für unsere Zwecke sind alle Gates dahingehend ideal, daß der Ausgang erscheint, sobald der Eingang angewandt wird. In Wirklichkeit haben Chips eine endliche **Gate-Verzögerung** (Gate Delay), die sowohl die Signalausbreitung (Signal Propagation) durch den Chip als auch die Schaltzeit beinhaltet. Übliche Verzögerungen reichen von 1 bis 10 ns.

Nach dem derzeitigen Stand der Technik könnten fast 10 Millionen Transistoren auf einen einzigen Chip gepackt werden. Da jede Schaltung aus NAND-Gates gebaut werden kann, möchte man meinen, daß ein Hersteller einen sehr allgemeinen Chip mit 5 Millionen NAND-Gates fertigen kann. Leider würde ein solcher Chip 15.000.002 Pins benötigen. Bei einem Standardabstand von 0,1 Zoll zwischen den einzelnen Pins wäre der Chip über 18 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 Gate-Pin-Verhältnis. In den folgenden Abschnitten betrachten wir einfache MSI-Schaltungen, die intern eine Reihe von Gates auf sich vereinen, um eine nützliche Funktion zu bieten, wobei nur eine begrenzte Anzahl von externen Verbindungen (Pins) nötig ist.

3.2.2 Kombinationsschaltungen

Viele Anwendungen im Bereich der digitalen Logik erfordern eine Schaltung mit mehreren Ein- und Ausgängen, wobei die Ausgänge eindeutig durch die Stromeingänge bestimmt werden. Eine solche Schaltung nennt man **Kombinationsschaltung** (Combinational Circuit). Nicht alle Schaltungen weisen dieses Merkmal auf. Beispielsweise kann eine Schaltung, die Speicherelemente beinhaltet, Ausgänge erzeugen, die von den gespeicherten Werten und den Eingangsvariablen abhängen. Eine Schaltung, die eine Wahrheitstabelle implementiert, z.B. die von Abb. 3.3(a), ist ein typisches Beispiel einer Kombinationsschaltung. In diesem Abschnitt betrachten wir einige häufig benutzte Kombinationsschaltungen.

Multiplexer

Auf der digitalen logischen Ebene ist ein **Multiplexer** eine Schaltung mit 2^n Dateneingängen, einem Datenausgang und n Steuereingängen, die einen der Dateneingänge wählen. Der gewählte Dateneingang wird zum Ausgang geleitet. Abb. 3.11 zeigt schematisch einen Multiplexer mit acht Eingängen. Die drei Steuerleitungen A, B und C kodieren eine 3-Bit-Zahl, die definiert, welche der acht Eingangsleitungen zum OR-Gate und somit zum Ausgang geleitet wird. Ungeachtet des Werts der Steuerleitungen sind sieben der AND-Gates immer Ausgang 0; das übrige kann je nach dem Wert der gewählten Eingangsleitung Ausgang 0 oder 1 sein. Jedes AND-Gate wird durch unterschiedliche Kombinationen der Steuereingänge aktiviert. Die Multiplexer-Schaltung ist aus Abb. 3.11 ersichtlich. Fügt man Strom und Masse hinzu, läßt sich das Teil auf eine 14-Pins-Einheit packen.

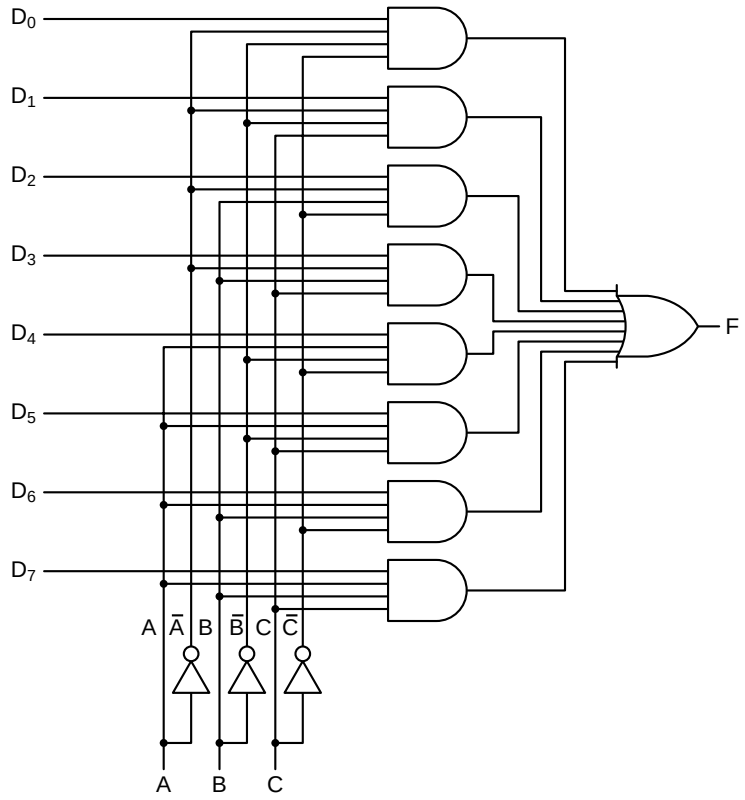


Abb. 3.11: Multiplexer-Schaltung mit acht Eingängen

Mit Hilfe des Multiplexers können wir die Mehrheitsfunktion von Abb. 3.3(a) wie in Abb. 3.12(b) implementieren. Für jede Kombination von *A*, *B* und *C* wird eine der Dateneingangsleitungen gewählt. Jeder Eingang wird entweder mit V_{cc} (logische 1) oder mit Masse (logische 0) verdrahtet. Der Algorithmus für die Verdrahtung der Eingänge ist einfach: Eingang D_i entspricht dem Wert in Zeile *i* der Wahrheitstabelle. In Abb. 3.3(a) sind die Zeilen 0, 1, 2 und 4 gleich 0, so daß die entsprechenden Eingänge geerdet sind. Die übrigen Zeilen sind 1, so daß sie mit der logischen 1 verbunden sind. Auf diese Weise kann eine beliebige Wahrheitstabelle von drei Variablen mit dem Chip von Abb. 3.12(a) implementiert werden.

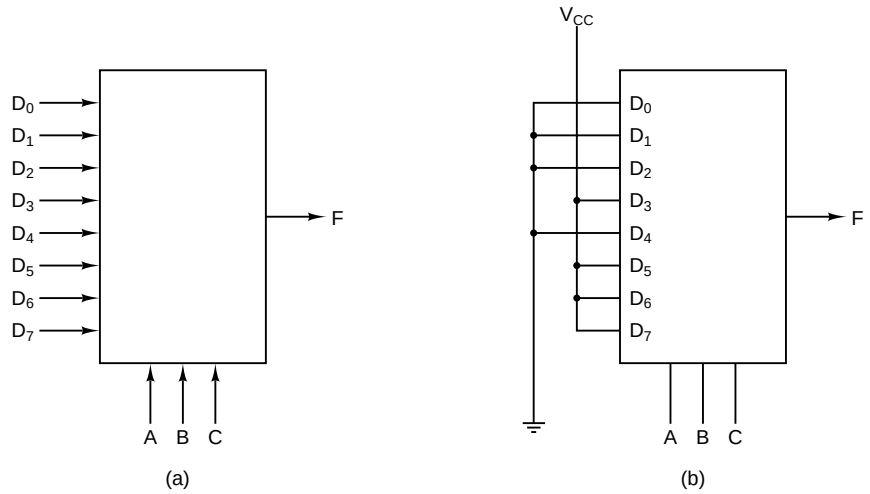


Abb. 3.12: (a) MSI-Multiplexer-Chip; (b) der gleiche Chip zur Berechnung der Mehrheitsfunktion verschaltet

Wir haben bereits gesehen, wie ein Multiplexer-Chip benutzt werden kann, um einen von mehreren Eingängen zu wählen, und wie er eine Wahrheitstabelle implementieren kann. Als weiterer Anwendungsbereich ist ein Parallel/Seriell-Datenwandler machbar. Legt man 8 Datenbits auf die Eingangsleitungen und schaltet man die Steuerleitungen sequentiell von 000 bis 111 (binär) weiter, werden diese in Reihe auf die Ausgangsleitung gelegt. Eine typische Nutzung für die Parallel/Seriell-Umwandlung ist eine Tastatur. Dabei definiert jeder Anschlag implizit eine 7- oder 8-Bit-Zahl, die seriell über eine Telefonleitung ausgegeben werden muß.

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

Dekodierer

Als zweites Beispiel eines MSI-Chips betrachten wir eine Schaltung, die eine n-Bit-Zahl als Eingabe annimmt und sie benutzt, um genau eine der 2^n Ausgangsleitungen zu wählen (d.h. auf 1 zu setzen). Eine solche Schaltung nennt man **Dekodierer** (Decoder). Abb. 3.13 zeigt ein Beispiel mit $n = 3$.

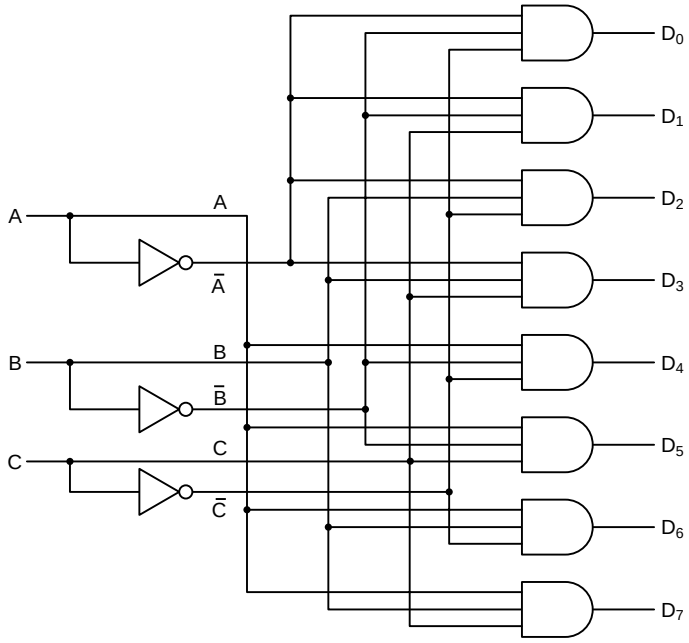


Abb. 3.13: 3-zu-8-Dekodierer-Schaltung

Um den Nutzen eines Dekodierers zu sehen, stelle man sich einen Speicher vor, der aus acht Chips mit je 1 Mbyte besteht. Chip 0 hat die Adressen 0 bis 1 Mbyte, Chip 1 die Adressen 1 Mbyte bis 2 Mbyte usw. Wird eine Adresse im Speicher dargestellt, werden die 3 höherwertigen Bits benutzt, um einen der acht Chips zu wählen. Unter Verwendung der Schaltung von Abb. 3.13 sind diese 3 Bits die drei Eingänge *A*, *B* und *C*. Je nach den Eingängen ist genau eine der acht Ausgangsleitungen D_0, ..., D_7 gleich 1, und die restlichen sind 0. Jede Ausgangsleitung aktiviert einen der acht Speicherchips. Da nur eine Ausgangsleitung auf 1 gesetzt wurde, ist nur ein Chip eingeschaltet.

Die Betriebsweise der Schaltung von Abb. 3.13 ist leicht nachvollziehbar. Jedes AND-Gate 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 Gate wird durch eine andere Eingangskombination eingeschaltet: D_0 durch ABC, D_1 durch $\overline{A}\overline{B}C$ usw.

Komparatoren

Eine weitere nützliche Schaltung ist der **Komparator** (Comparator), der zwei Eingangswörter vergleicht. Der einfache Komparator von Abb. 3.14 nimmt zwei Eingaben *A* und *B* mit einer Länge von je 4 Bits und produziert 1, falls sie gleich sind, und 0, falls sie ungleich sind. Die Schaltung basiert auf dem EXCLUSIVE-OR-Gate (XOR), das eine 0 ausgibt, wenn seine Eingänge gleich sind, und eine 1 ausgibt, wenn sie ungleich sind. Sind die beiden Eingabewör-

ter gleich, müssen alle vier XOR-Gates 0 ausgeben. Diese vier Signale können dann mit OR verknüpft werden. Ist das Ergebnis 0, sind die Eingabewörter gleich, andernfalls nicht. In unserem Beispiel haben wir ein NOR-Gate als endgültiges Stadium zur Umkehrung der Prüfrichtung benutzt: 1 bedeutet gleich und 0 bedeutet ungleich.

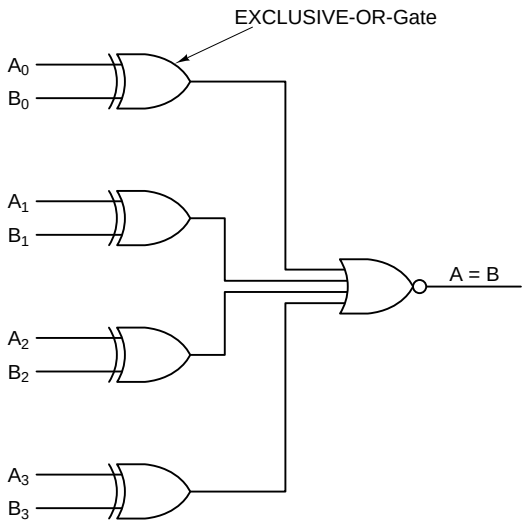


Abb. 3.14: Einfacher 4-Bit-Komparator

Programmierte Logik-Arrays

Wir haben bereits gesehen, daß beliebige Funktionen (Wahrheitstabellen) dadurch erzeugt werden können, daß man Produktterme mit AND-Gates berechnet und dann die Produkte mit OR verknüpft. Ein sehr allgemeiner Chip zur Summenbildung von Produkten ist das **programmierbare Logik-Array** (Programmable Logic Array – PLA). Abb. 3.15 zeigt ein einfaches Beispiel. Dieser Chip hat Eingangsleitungen für 12 Variablen. Das Komplement jedes Eingangs wird intern erzeugt, so daß sich insgesamt 24 Eingangssignale ergeben. Der Kern der Schaltung ist ein Array von 50 AND-Gates, die jeweils potentiell jede beliebige Untermenge von 24 Eingangssignalen als Eingabe haben können. Welches Eingangssignal an welches AND-Gate geht, wird durch eine 24×50-Bit-Matrix bestimmt, die vom Benutzer bereitgestellt wird. Jede Eingangsleitung zu den 50 AND-Gates enthält eine Sicherung. Bei Auslieferung sind alle 1200 Sicherungen intakt. Um die Matrix zu programmieren, brennt der Benutzer durch Anlegen einer hohen Spannung auf den Chip ausgewählte Sicherungen durch.

Der Ausgangsteil der Schaltung besteht aus sechs OR-Gates, die entsprechend den 50 Ausgängen der AND-Gates je bis zu 50 Eingänge haben. Auch hier bestimmt die vom Benutzer bereitgestellte (50×6) Matrix, welche der potentiel-

len Verbindungen tatsächlich existiert. Der Chip hat 12 Eingangspins, 6 Ausgangspins sowie Strom und Masse, d.h. insgesamt 20 Pins.

Als praktisches Beispiel eines PLAs ziehen wir wieder die Schaltung von Abb. 3.3(b) heran. Sie hat drei Eingänge, vier AND-Gates, ein OR-Gate und drei Inverter. Mit den entsprechenden internen Verbindungen kann unser PLA diese Funktion mit Hilfe seiner 12 Eingänge, vier seiner 50 AND-Gates und einem seiner sechs OR-Gates berechnen. (Die vier AND-Gates sollten $\overline{A}BC$, $A\overline{B}C$, $AB\overline{C}$ bzw. ABC berechnen. Das OR-Gate nimmt diese vier Produktterme als Eingabe an.) Wir könnten den gleichen PLA so verdrahten, daß er gleichzeitig insgesamt vier Funktionen ähnlicher Komplexität berechnet. Für diese einfachen Funktionen ist der Begrenzungsfaktor die Anzahl der Eingabevariablen; für komplexere könnten es die AND- oder OR-Gates sein.

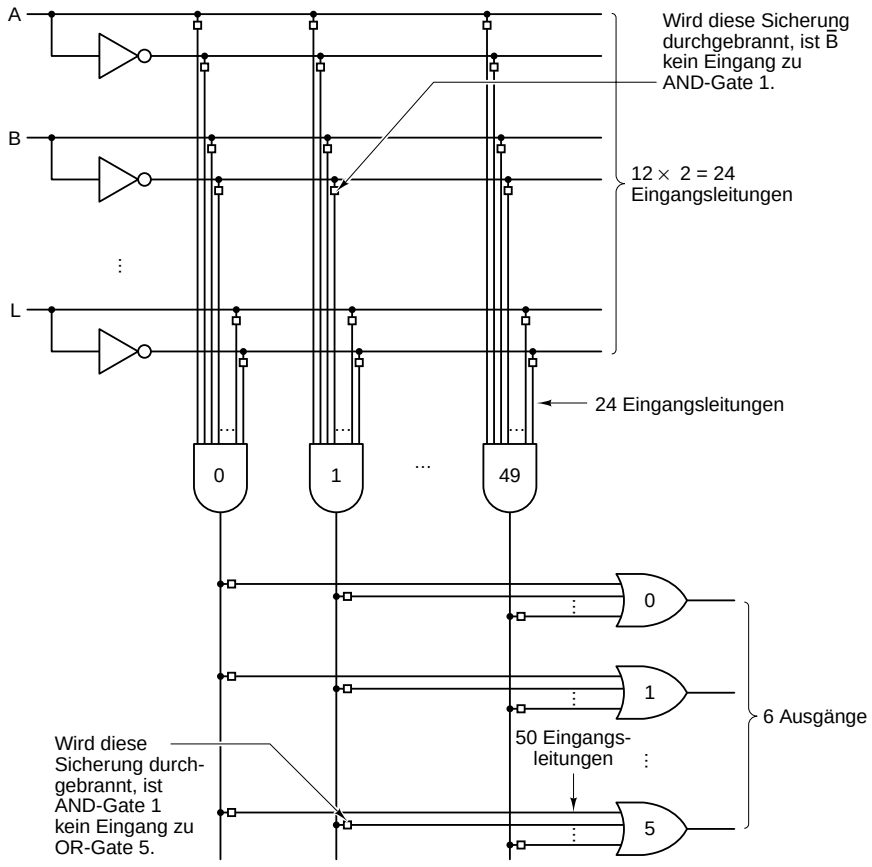


Abb. 3.15: Programmierbares Logik-Array (PLA) mit 12 Ein- und 6 Ausgängen. Die kleinen Quadrate stellen Sicherungen dar, die zur Bestimmung der zu berechnenden Funktion ausgebrannt werden können. Die Sicherungen sind in zwei Matrizen angeordnet: die obere für die AND-Gates und die untere für die OR-Gates.

Obwohl solche im Feld programmierbare PLAs noch in Gebrauch sind, werden individuell angepaßte PLAs für viele Anwendungen bevorzugt. Diese Einheiten werden vom Großabnehmer entwickelt und vom Hersteller gemäß den Kundenspezifikationen gefertigt. Sie sind billiger als die im Feld programmierbaren.

Wir können jetzt die drei verschiedenen Implementierungsmöglichkeiten der Wahrheitstabelle von Abb. 3.3(a) vergleichend gegenüberstellen. Bei SSI-Komponenten brauchen wir vier Chips. Alternativ würde ein MSI-Multiplexer-Chip (Abb. 3.12(b)) ausreichen. Schließlich könnten wir ein Viertel eines PLA-Chips benutzen. Werden viele Funktionen benötigt, ist der PLA natürlich effizienter als die anderen beiden Methoden. Für einfache Schaltungen sind die billigeren SSI- und MSI-Chips sicherlich vorzuziehen.

3.2.3 Arithmetische Schaltungen

Jetzt ist es an der Zeit, uns von den oben behandelten MSI-Allzweckschaltungen auf MSI-Kombinationsschaltungen zu verlagern, die für arithmetische Berechnungen benutzt werden. Wir beginnen mit einem einfachen 8-Bit-Schieber; dann betrachten wir einen Addierer und schließlich arithmetische Logikeinheiten, die in jedem Computer eine zentrale Rolle spielen.

Schieber

Unsere erste arithmetische MSI-Schaltung ist ein **Schieber** (Shifter) mit acht Eingängen und acht Ausgängen (siehe Abb. 3.16). Acht Eingangsbits sind auf den Leitungen D_0, ..., D_7 dargestellt. Der Ausgang, der lediglich der um ein Bit verschobene Eingang ist, ist auf den Leitungen S_0, ..., S_7 verfügbar. Die Steuerleitung *C* bestimmt die Richtung der Verschiebung; sie beträgt 0 = links und 1 = rechts.

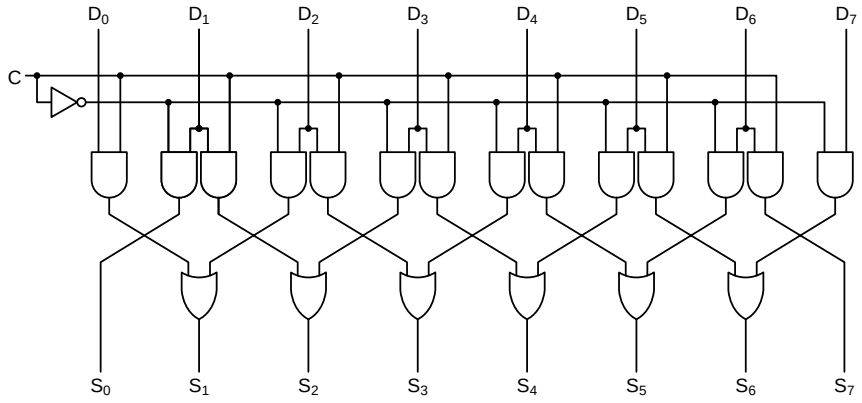


Abb. 3.16: 1-Bit-Schieber links/rechts

Um die Funktionsweise der Schaltung zu sehen, beachte man die Paare von AND-Gates für alle Bits, mit Ausnahme der Gates am Ende. Wenn $C = 1$, wird das rechte Mitglied jedes Paars eingeschaltet, wodurch das entsprechende Eingangsbit zum Ausgang gereicht wird. Da das rechte AND-Gate mit dem Eingang des OR-Gates rechts daneben verdrahtet ist, wird eine Verschiebung nach rechts durchgeführt. Wenn $C = 0$, wird das linke Mitglied des AND-Gates eingeschaltet, wodurch sich eine Verschiebung nach links ergibt.

Addierer

Ein Computer, der keine Ganzzahlen addieren kann, ist undenkbar. Folglich sind Schaltungen zur Durchführung von Additionen ein unentbehrlicher Teil jeder CPU. Die Wahrheitstabelle für eine 1-Bit-Addition ist in Abb. 3.17(a) dargestellt. Hier gibt es zwei Ausgänge, die Summe der Eingänge *A* und *B* und den Übertrag zur nächsten Stelle (nach links). Abb. 3.17(b) zeigt eine Schaltung zur Berechnung von Summe und Übertrag. Diese Schaltung heißt **Halbaddierer** (Half Adder).

A	B	Su	Über
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

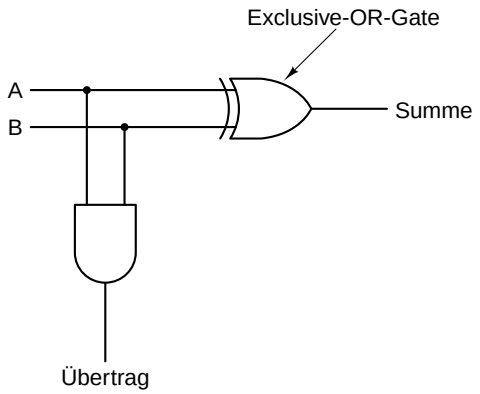
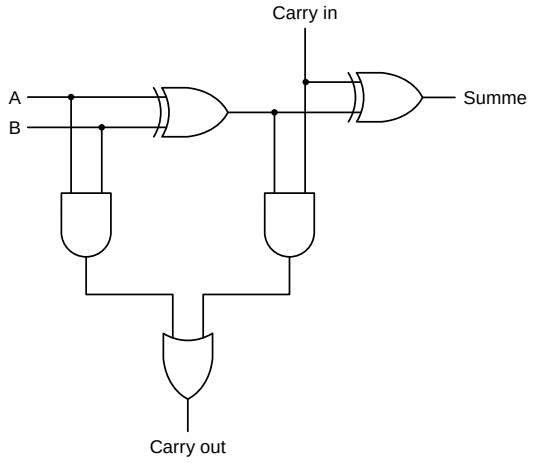


Abb. 3.17: (a) Wahrheitstabelle für eine 1-Bit-Addition; (b) Schaltung für einen Halbaddierer

Nun ist ein Halbaddierer zwar für die Addition der niederwertigen Bits zweier aus mehreren Bits bestehenden Eingabewörter geeignet, reicht aber nicht für eine Bitstelle in der Wortmitte, weil er den Übertrag an die Stelle von rechts nicht durchführt. Hier benötigt man den **Volladdierer** (Full Adder) von Abb. 3.18. Aus Sicht der Schaltung ist klar, daß ein Volladdierer aus zwei Halbaddierern besteht. Die Ausgangsleitung *Sum* ist 1, wenn eine ungerade Zahl von *A*, *B* und *Carry in* 1 sind. *Carry out* ist 1, wenn entweder *A* und *B* beide 1 sind (linker Eingang zum OR-Gate) oder eine davon 1 und das *Carry-in*-Bit ebenfalls 1 ist. Zusammen erzeugen die beiden Halbaddierer sowohl die Summen- als auch die Übertragsbits.

A	B	Carry in	Su	Carry out
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)

Abb. 3.18: (a) Wahrheitstabelle und (b) Schaltung für einen Volladdierer

Um einen Addierer für, sagen wir, zwei 16-Bit-Wörter zu bauen, wiederholt man lediglich die Schaltung von Abb. 3.18(b) 16 Mal. Das *Carry out* eines Bits braucht man als Übertrag in die linke Nachbarstelle. Der Übertrag in das Bit ganz rechts wird auf 0 verdrahtet. Dies nennt man **Ripple Carry Adder** (in etwa: Durchlaufübertragaddierer bzw. Addierer mit Übertragsweiterleitung), weil die Addition von 1 zu 111...111 (binär) im schlechtesten Fall so lange nicht beendet werden kann, bis der Übertrag die ganze Strecke vom ganz linken zum ganz rechten Bit durchlaufen hat. Es gibt auch Addierer, die diese Verzögerung nicht haben und somit schneller sind.

Als einfaches Beispiel eines schnelleren Addierers teilen wir einen 32-Bit-Addierer in eine untere Hälfte 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 16 Additionen übernehmen muß.

Sehen Sie sich aber diese Veränderung an: Statt einer einzelnen oberen Hälfte verleihen wir dem Addierer durch Duplizierung der Hardware der oberen Hälfte parallel zwei obere Hälften. Jetzt besteht die Schaltung aus drei 16-Bit-Addierern: einer unteren Hälfte und zwei oberen Hälften *U0* und *U1*, die parallel laufen. Eine 0 wird als Übertrag in *U0* gespeist; eine 1 wird als Übertrag in *U1* gespeist. Jetzt können diese beiden gleichzeitig mit der unteren Hälfte starten, aber nur einer wird korrekt sein. Nach 16 Bitadditionen wird der in die obere Hälfte zu übernehmende Übertrag bekannt sein, so daß die richtige obere Hälfte nun aus den beiden verfügbaren Optionen ausgewählt werden kann. Auf diese Weise wird die Additionshäufigkeit um einen Faktor von Zwei verringert. Einen solchen Addierer nennt man **Carry Select Adder** (in etwa:

Übertragsauswahladdierer). Dieser Trick läßt sich wiederholen, um jeden 16-Bit-Addierer aus replizierten 8-Bit-Addierern usw. zu konstruieren.

Arithmetische Logikeinheiten

Die meisten Computer enthalten zur Durchführung von AND, OR und Addition zweier Maschinenwörter nur eine Schaltung. Normalerweise wird eine solche Schaltung für n-Bit-Wörter aus n identischen Schaltungen für die einzelnen Bitstellen gebaut. Abb. 3.19 ist ein einfaches Beispiel einer solchen Schaltung, die man **arithmetische Logikeinheit** (Arithmetic Logic Unit – ALU) nennt. Sie kann vier Funktionen, nämlich A AND B, A OR B, $\overline{B}$ oder $A + B$ berechnen, je nachdem, ob F_0 und F_1 die für die Funktionen gewählten Eingangsleitungen 00, 01, 10 oder 11 (binär) enthalten. Hier bedeutet $A + B$ die arithmetische Summe von A und B, und nicht das boolesche AND.

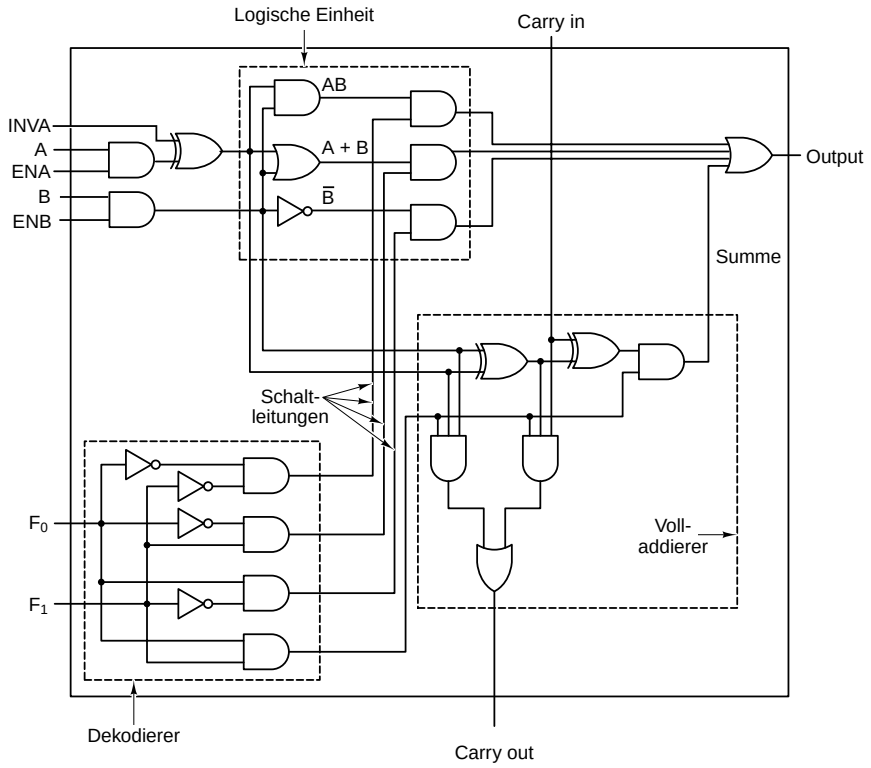


Abb. 3.19: Eine 1-Bit-ALU

In der unteren linken Ecke unserer ALU befindet sich ein 2-Bit-Dekodierer, der die Schaltsignale für die vier Operationen auf der Grundlage von F_0 und F_1 erzeugt. Je nach den Werten von F_0 und F_1 wird genau eine der vier Schaltleitungen gewählt. Die Assertion dieser Leitung erlaubt dem Ausgang der ge-

wählten Funktion, bezüglich des Ausgangs bis zum letzten OR-Gate zu passieren.

In der oberen linken Ecke befindet sich die Logik zur Berechnung von A AND B, A OR B und $\overline{B}$; höchstens eines dieser Ergebnisse wird aber an das letzte OR-Gate weitergereicht, je nach den aus dem Dekodierer austretenden Schaltleitungen. Da genau einer der Dekodierer-Ausgänge 1 ist, wird genau eines der vier das OR-Gate steuernden AND-Gates eingeschaltet; die übrigen drei sind Ausgang 0, unabhängig von A und B.

Abgesehen davon, daß man A und B als Eingänge für logische oder arithmetische Operationen benutzen kann, ist es auch möglich, einen davon auf 0 zu zwingen, wenn man ENA bzw. ENB negiert. Außerdem erhält man $\overline{A}$ durch Assertion von INVA. Anwendungen für INVA, ENA und ENB werden in Kapitel 4 behandelt. Unter normalen Bedingungen werden ENA und ENB asseriert, um beide Eingänge einzuschalten, und INVA wird negiert. In diesem Fall werden A und B einfach unverändert in die Logikeinheit eingespeist.

In der unteren rechten Ecke der ALU befindet sich ein Volladdierer zur Berechnung der Summe von A und B, einschließlich den Überträgen, weil wahrscheinlich mehrere dieser Schaltungen letztendlich parallel verschaltet werden, um Vollwort-Operationen durchzuführen. Chips wie der in Abb. 3.19 sind im Handel erhältlich und werden als **Bit-Slice-Prozessoren** (Bitscheiben-Prozessoren) bezeichnet. Sie ermöglichen es dem Computer-Designer, eine ALU in jeder gewünschten Breite zu entwerfen. Abb. 3.20 zeigt eine 8-Bit-ALU, die sich aus acht 1-Bit-ALU-Slices zusammensetzt. Das INC-Signal ist nur für Additionen nützlich. Bei Assertion erhöht es das Ergebnis um 1, so daß es möglich ist, Summen wie $A + 1$ und $A + B + 1$ zu berechnen.

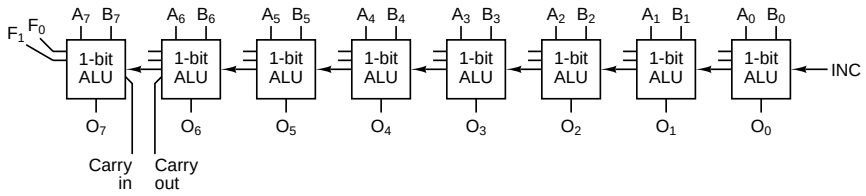


Abb. 3.20: Acht 1-Bit-ALU-Slices sind verbunden und bilden eine 8-Bit-ALU

3.2.4 Taktgeber

In vielen digitalen Schaltungen ist die Reihenfolge, in der Ereignisse passieren, von maßgeblicher Bedeutung. Zuweilen muß ein Ereignis einem anderen vorausgehen, in anderen Fällen müssen zwei Ereignisse gleichzeitig eintreten. Damit Entwickler die erforderlichen Zeitrelationen erreichen können, sind viele digitale Schaltungen mit Taktgebern zum Zwecke der Synchronisation ausgestattet. Ein **Taktgeber** (Clock) ist in diesem Zusammenhang eine Schaltung, die eine Reihe von Impulsen in einer präzisen Impulsbreite und in ei-

nem präzisen Intervall zwischen aufeinanderfolgenden Impulsen ausgibt. Das Intervall zwischen entsprechenden Kanten von zwei aufeinanderfolgenden Impulsen nennt man **Taktzykluszeit** (Clock Cycle Time). Impulsfrequenzen liegen normalerweise zwischen 1 MHz und 500 MHz und entsprechen den Taktzyklen von 1000 ns bis 2 ns. Zur Erzielung einer hohen Genauigkeit wird die Taktfrequenz normalerweise von einem Kristalloszillator gesteuert.

In einem Computer können innerhalb eines einzigen Taktzyklus viele Ereignisse ablaufen. Sollen diese Ereignisse in einer bestimmten Reihenfolge eintreten, muß der Taktzyklus in Teilzyklen aufgeteilt werden. Ein übliches Vorgehen, eine feinere Auflösung als der einfache Taktgeber zu bieten, ist die Anzapfung der primären Taktleitung und das Einfügen einer Schaltung mit einer bekannten Verzögerung, so daß ein sekundäres, zum primären phasenverschobenes Taktsignal erzeugt wird (siehe Abb. 3.21(a)). Das Taktdiagramm von Abb. 3.21(b) bietet vier Taktverweise für bestimmte Ereignisse:

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 Abfolge umsetzen. Werden innerhalb eines Taktzyklus mehr als vier Zeitverweise benötigt, kann man mehr sekundäre Leitungen von der primären aus anzapfen und mit verschiedenen Verzögerungen belegen.

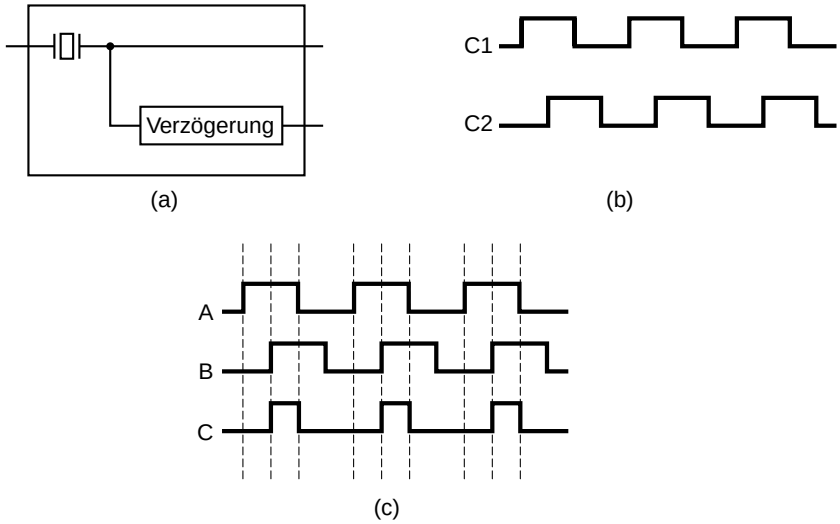


Abb. 3.21: (a) Taktgeber; (b) Taktdiagramm für den Taktgeber; (c) Erzeugung eines asymmetrischen Takts

Bei manchen Schaltungen ist man mehr an Zeitintervallen als an diskreten Zeitinstanzen interessiert. Ein Ereignis kann z.B. jederzeit eintreten, wenn C1 auf High und nicht genau an der steigenden Flanke liegt. Ein weiteres Ereignis kann nur eintreten, wenn C2 auf High liegt. Werden mehr als zwei Intervalle benötigt, stellt man mehrere Taktleitungen bereit oder die High-Zustände der beiden Taktgeber so ein, daß sie zeitlich teilweise überlappen. Im letzteren Fall lassen sich vier Intervalle unterscheiden: $\overline{C1}$ AND $\overline{C2}$, $\overline{C1}$ AND C2, C1 AND $\overline{C2}$ und C1 AND C2.

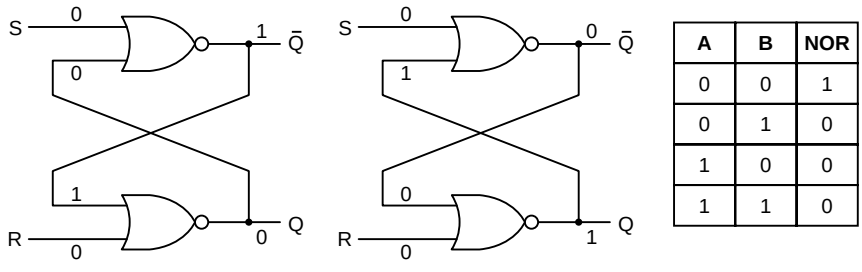
Nebenbei bemerkt sind Taktgeber symmetrisch, wobei die im High-Zustand ablaufende Zeit der Zeit entspricht, die im Low-Zustand abläuft, wie aus Abb. 3.21(b) ersichtlich ist. Um eine asymmetrische Impulsfolge zu erzeugen, wird der Basistakt mit Hilfe einer Verzögerungsschaltung verschoben und mittels AND mit dem Ursprungssignal verknüpft, wie C in Abb. 3.21(c).

3.3 Speicher

Eine wesentliche Komponente eines jeden Computers ist sein Speicher. Ohne Speicher wären Computer nicht das, was sie sind. Ein Speicher dient der Speicherung auszuführender Instruktionen und Daten. In den folgenden Abschnitten werden die grundlegenden Komponenten eines Speichersystems, mit der Gate-Ebene beginnend, behandelt, um aufzuzeigen, wie sie funktionieren und zu großen Speichern zusammengefaßt werden können.

3.3.1 Latches

Um einen 1-Bit-Speicher zu erstellen, brauchen wir eine Schaltung, die sich irgendwie vorherige Eingabewerte »merkt«. Eine solche Schaltung läßt sich aus zwei NOR-Gates bauen (siehe Abb. 3.22(a)). Analoge Schaltungen können aus NAND-Gates gebaut werden. Wir erwähnen diese nicht weiter, weil sie vom Konzept her mit den NOR-Versionen identisch sind.



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

(c)

Abb. 3.22: (a) NOR-Latch im Zustand 0; (b) NOR-Latch im Zustand 1; (c) Wahrheitstabelle für NOR

Die Schaltung von Abb. 3.22(a) nennt man **SR-Latch**. Sie hat zwei Eingänge – S zum Setzen der Latch und R zum Zurücksetzen (d.h. Löschen) derselben. Ferner hat sie zwei komplementäre Ausgänge Q und $\overline{Q}$, wie wir gleich sehen werden. Im Gegensatz zu einer Kombinationsschaltung werden die Ausgänge der Latch nicht eindeutig durch die Stromeingänge bestimmt.

Das hat folgenden Grund: Angenommen, S und R sind 0, was sie die meiste Zeit sind. Weiter nehmen wir an, daß $Q = 0$. Da Q in das obere NOR-Gate zurückgespeist wird, sind seine beiden Eingänge 0; deshalb ist sein Ausgang $\overline{Q}$ 1. Die 1 wird in das untere Gate zurückgespeist, das dann die Eingänge 1 und 0 hat, woraus sich $Q = 0$ ergibt. Dieser Zustand ist konsistent und aus Abb. 3.22(a) ersichtlich.

Nun stellen wir uns vor, daß Q nicht 0, sondern 1 ist, während R und S immer noch 0 sind. Das obere Gate hat die Eingänge 0 und 1 und einen Ausgang $\overline{Q}$ von 0, der zum unteren Gate zurückgespeist wird. Dieser Zustand (siehe Abb. 3.22(b)) ist ebenfalls konsistent. Ein Zustand, bei dem beide Ausgänge gleich 0 sind, ist konsistent, weil er beide Gates zu zwei Nullen als Eingang zwingt, was im Falle von wahr 1 und nicht 0 als Ausgang produzieren würde. Ebenso ist es 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 die Latch zwei stabile Zustände, die wir abhängig von Q 0 und 1 nennen.

Jetzt prüfen wir die Wirkung der Eingänge auf den Zustand der Latch. Angenommen, S wird 1, während $Q = 0$. Die Eingänge zum oberen Gate sind dann 1 und 0, wodurch der Ausgang $\overline{Q}$ auf 0 gezwungen wird. Dadurch werden beide Eingänge zum unteren Gate 0, was den Ausgang in 1 zwingt. Setzt man also S (auf 1), schaltet der Zustand von 0 in 1. Setzt man R auf 1, wenn sich die Latch im Zustand 0 befindet, tritt keine Wirkung ein, weil der Ausgang des unteren NOR-Gates für die Eingänge von 10 und die Eingänge von 11 je 0 ist.

Dadurch läßt sich leicht erkennen, daß das Setzen von S auf 1 im Zustand $Q = 1$ wirkungslos bleibt, während das Setzen von R die Latch in den Zustand $Q = 0$ treibt. Das heißt: Wird S vorübergehend auf 1 gesetzt, endet die Latch im Zustand $Q = 1$, ungeachtet des Zustands, in dem sie sich vorher befunden hat. Setzt man R vorübergehend auf 1, wird die Latch in den Zustand $Q = 0$ gezwungen. Die Schaltung »merkt« sich, ob S oder R zuletzt an war. Anhand dieser Eigenschaft können wir Computer-Speicher entwickeln.

Getaktete SR-Latches

Latches müssen häufig so konstruiert werden, daß sie nur in ganz bestimmten Fällen ihren Zustand ändern können. Um dieses Ziel zu erreichen, ändern wir die Basisschaltung geringfügig, wie in Abb. 3.23 aufgezeigt, um eine sogenannte **Getaktete SR-Latch** (Clocked SR Latch) zu realisieren.

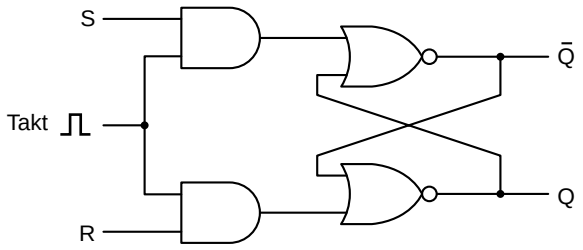


Abb. 3.23: Getaktete SR-Latch

Diese Schaltung verfügt über einen zusätzlichen Eingang, den Taktgeber, der normalerweise 0 ist. Ist der Taktgeber auf 0, geben beide AND-Gates unabhängig von S und R 0 aus, und die Latch ändert ihren Zustand nicht. Ist der Taktgeber 1, verschwindet die Wirkung der AND-Gates, und die Latch reagiert auf S und R. Das Taktsignal muß nicht durch eine Uhr gespeist werden. Die Begriffe **enable** und **strobe** sind weit verbreitet und bedeuten, daß der Takteingang 1 ist. Das bedeutet, daß die Schaltung auf den Zustand von S und R reagiert.

Bis jetzt haben wir das Problem, was passiert, wenn S und R 1 sind, übergangen. Und das aus gutem Grund: Die Schaltung wird nicht deterministisch, wenn R und S letztendlich auf 0 zurückkehren. Der einzige konsistente Zustand für $S = R = 1$ ist $Q = \overline{Q} = 0$. Sobald aber beide Eingänge auf 0 zurückkehren, muß die Latch in einen ihrer beiden stabilen Zustände springen. Fällt ein Eingang vor dem anderen auf 0 zurück, gewinnt derjenige, der am längsten in 1 verbleibt. Wenn nämlich nur ein Eingang 1 ist, erzwingt er den Zustand. Kehren beide Eingänge gleichzeitig (was sehr unwahrscheinlich ist) auf 0 zurück, springt die Latch willkürlich in einen ihrer stabilen Zustände.

Getaktete D-Latches

Das Problem der Zweideutigkeit (wenn $S = R = 1$) der SR-Latch wird am besten gelöst, wenn man seinem Auftreten vorbeugt. Abb. 3.24 zeigt eine Latchschaltung mit nur einem Eingang D. Da der Eingang zum unteren AND-Gate immer das Komplement des Eingangs zum oberen ist, entsteht das Problem, daß beide Eingänge 1 sind, überhaupt nie. Wenn $D = 1$ und der Taktgeber ist 1, wird die Latch in den Zustand $Q = 1$ getrieben. Wenn $D = 0$ und die Uhr ist 1, wird die Latch in den Zustand $Q = 0$ versetzt. Anders ausgedrückt: Ist der Taktgeber 1, wird der aktuelle Wert von D als Muster abgetastet und in der Latch gespeichert. Diese Schaltung nennt man **getaktete D-Latch** (Clocked D Latch); sie 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, wird ein positiver Impuls auf die Taktleitung gelegt.

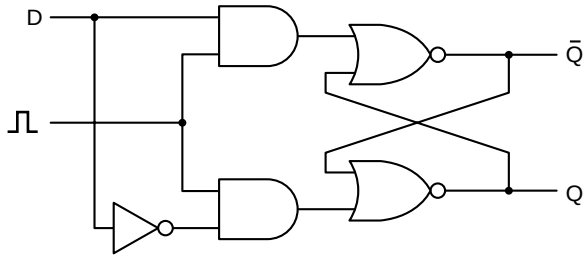


Abb. 3.24: Getaktete D-Latch

3.3.2 Flip-Flop-Schaltungen

Bei vielen Schaltungen muß man den Wert einer Leitung zu einem bestimmten Zeitpunkt abtasten und speichern. Bei dieser Variante – der **Flip-Flop-Schaltung** – tritt der Zustandsübergang nicht ein, wenn der Taktgeber 1 ist, sondern von 0 auf 1 (steigende Flanke) oder von 1 auf 0 (fallende Flanke) übergeht. Die Länge des Taktimpulses ist also unwichtig, solange die Übergänge schnell stattfinden.

Zur Verdeutlichung betrachten wir den Unterschied zwischen einer Flip-Flop-Schaltung und einer Latch genauer. Eine Flip-Flop-Schaltung wird **flankengesteuert** (edge-triggered), während eine Latch **pegelgesteuert** (level-triggered) wird. In der Literatur werden diese Begriffe häufig durcheinander gebracht. Viele Autoren verwenden »Flip-Flop-Schaltung« für eine Latch, und umgekehrt.

Für das Design einer Flip-Flop-Schaltung gibt es mehrere Ansätze. Hätte man beispielsweise die Möglichkeit, einen sehr kurzen Impuls auf der steigenden Flanke des Taktsignals zu erzeugen, könnte man diesen Impuls in eine D-Latch einspeisen. Diese Möglichkeit gibt es, und eine entsprechende Schaltung wird in Abb. 3.25(a) vorgestellt.

Auf den ersten Blick sieht es so aus, als ob der Ausgang des AND-Gates immer Null wäre, da das AND eines Signals zu seinem Gegenstück Null ist. Die Sache ist aber nicht so einfach. Durch den Inverter zieht sich eine kleine (ungleich Null) Ausbreitungsverzögerung, und genau diese Verzögerung sorgt für das Funktionieren der Schaltung. Angenommen, wir messen die Spannung der vier Meßpunkte *a*, *b*, *c* und *d*. Das an *a* gemessene Eingangssignal ist ein langer Taktimpuls (in Abb. 3.25(b) unten). Oberhalb dieses Impulses wird das an *b* gemessene Signal dargestellt. Letzteres ist sowohl invertiert als auch leicht verzögert, normalerweise um ein paar Nanosekunden, je nach dem benutzten Inverter.

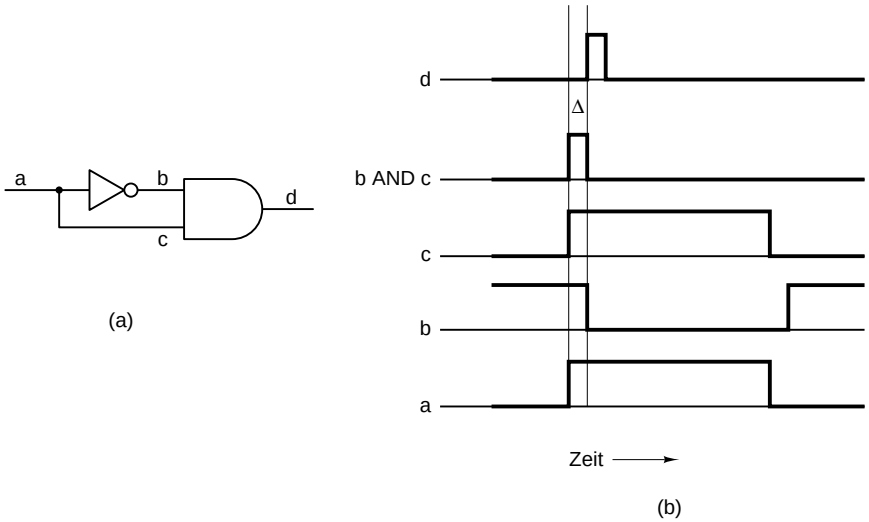


Abb. 3.25: (a) Pulsgenerator; (b) Taktgabe an vier Punkten der Schaltung

Das Signal an *c* ist ebenfalls verzögert, aber nur um die Ausbreitungszeit des Signals (also der Lichtgeschwindigkeit). Ist die physische Entfernung zwischen *a* und *c* beispielsweise 20 Mikron, ergibt sich eine Ausbreitungsverzögerung von 0,0001 ns – sicherlich ein tolerierbarer Wert im Vergleich zu der Zeit, die das Signal für die Ausbreitung durch den Inverter braucht. Das Signal an *c* ist im Grunde also identisch mit dem an *a*.

Werden die Eingänge *b* und *c* zum AND-Gate mit AND verknüpft, ist das Ergebnis ein kurzer Impuls wie in Abb. 3.25(b), wobei die Breite des Impulses Δ der Gate-Verzögerung des Inverters – normalerweise 5 ns oder weniger – entspricht. Der Ausgang zum AND-Gate ist nur dieser Impuls, verschoben um die Verzögerung des AND-Gates, wie in Abb. 3.25(b) dargestellt. Diese Zeitverschiebung bedeutet lediglich, daß die D-Latch in einer feststehenden Verzögerung nach der steigenden Flanke des Taktgebers aktiviert wird. Dies hat aber keine Wirkung auf die Pulsbreite. Bei einem Speicher mit einer Zykluszeit von 50 ns kann ein Impuls von 5 ns, der den Speicher anweist, wann die D-Leitung abzutasten ist, kurz genug sein. In diesem Fall kann die volle Schaltung Abb. 3.26 entsprechen. Dieses Flip-Flop-Schaltungsdesign ist nett; weil es leicht verständlich ist, wäre in der Praxis aber viel zu einfach.

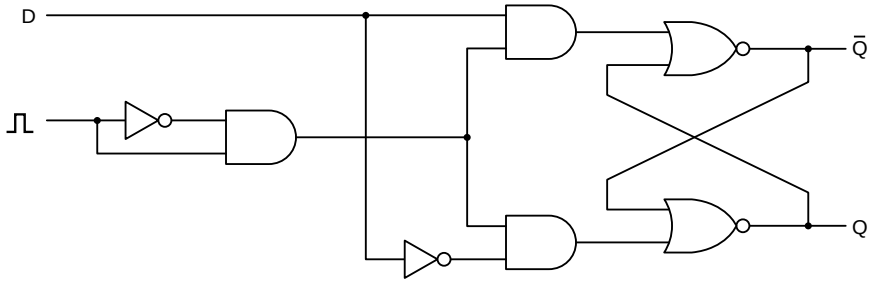


Abb. 3.26: D-Flip-Flop-Schaltung

Abb. 3.27 zeigt die Standardsymbole für Latches und Flip-Flop-Schaltungen. In Abb. 3.27(a) sehen wir eine Latch, die in ihren Zustand eintritt, wenn der Taktgeber *CK* 1 ist. Abb. 3.27(b) zeigt im Gegensatz dazu eine Latch, deren Taktgeber normalerweise 1 ist, plötzlich aber auf 0 abfällt, um den Zustand von *D* zu laden. Die Abbildungen 3.27(c) und (d) stellen keine Latches, sondern Flip-Flop-Schaltungen dar, was durch Pfeilspitzen an den Takteingängen angedeutet ist. Abb. 3.27(c) ändert den Zustand auf der steigenden Flanke des Taktimpulses (Übergang von 0 auf 1), während Abb. 3.27(d) den Zustand auf der fallenden Flanke (Übergang von 1 auf 0) ändert. Viele, aber nicht alle Latches und Flip-Flop-Schaltungen haben auch $\overline{Q}$ als Ausgang. Einige haben zwei zusätzliche Eingänge *Set* oder *Preset* (Zustand in $Q = 1$ erzwingen) und *Reset* oder *Clear* (Zustand in $Q = 0$ erzwingen).

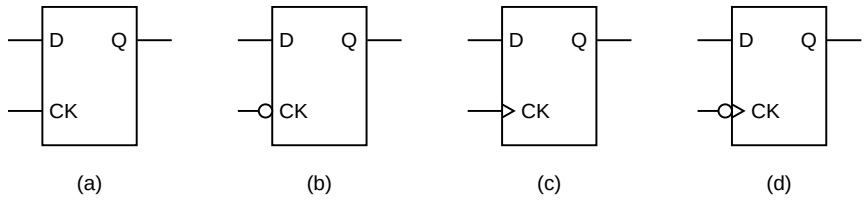


Abb. 3.27: D-Latches und Flip-Flop-Schaltungen

3.3.3 Register

Flip-Flop-Schaltungen gibt es in vielen verschiedenen Konfigurationen. Eine einfache mit zwei unabhängigen D-Flip-Flop-Schaltungen sowie Clear- und Preset-Signalen findet sich in Abb. 3.28(a). Obwohl sie auf einen 14-Pin-Chip gepackt wurden, haben die beiden Flip-Flop-Schaltungen aber keinen Zusammenhang. Eine recht unterschiedliche Anordnung weist die oktale Flip-Flop-Schaltung von Abb. 3.28(b) auf. Hier fehlen den acht (daher der Begriff »oktal«) D-Flip-Flop-Schaltungen nicht nur die $\overline{Q}$- und Preset-Leitungen. Alle Taktleitungen sind außerdem gruppiert und werden von Pin 11 gesteuert. Die Flip-Flop-Schaltungen selbst sind vom gleichen Typ wie Abb. 3.27(d), verfügen aber nicht über Inversionsblasen, weil der Inverter mit Pin 11 verbunden

ist, so daß die Flip-Flop-Schaltungen bei steigendem Übergang geladen werden. Alle acht Clear-Signale sind ebenfalls gruppiert, so daß alle Flip-Flop-Schaltungen in den 0-Zustand gezwungen werden, wenn Pin 1 in 0 wechselt. Falls Sie sich jetzt fragen, warum Pin 11 am Eingang und dann wieder bei jedem CK-Signal invertiert wird, hier die Antwort: Ein Eingangssignal hat möglicherweise nicht genug Strom, um alle acht Flip-Flop-Schaltungen zu betreiben, so daß der Eingangsinverter eigentlich als Verstärker verwendet wird.

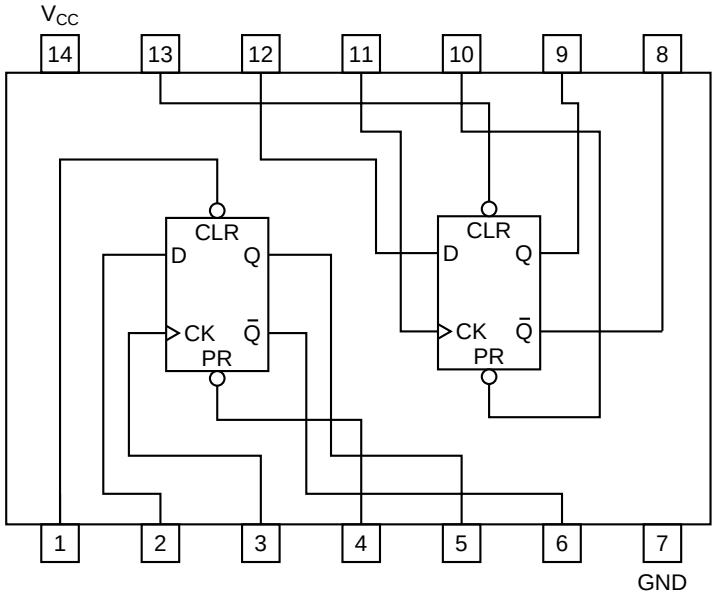
Während die Takt- und Clear-Leitungen von Abb. 3.28(b) einerseits gruppiert wurden, um Pins einzusparen, wird der Chip bei dieser Konfiguration anders als bei acht unzusammenhängenden Flip-Flop-Schaltungen benutzt. Er dient als einziges 8-Bit-Register. Alternativ können zwei solche Chips parallel zur Bildung eines 16-Bit-Registers verwendet werden. Hierfür verbindet man ihre Pins 1 und 11. In Kapitel 4 werden Register und ihre Anwendungsbereiche genauer betrachtet.

3.3.4 Speicherorganisation

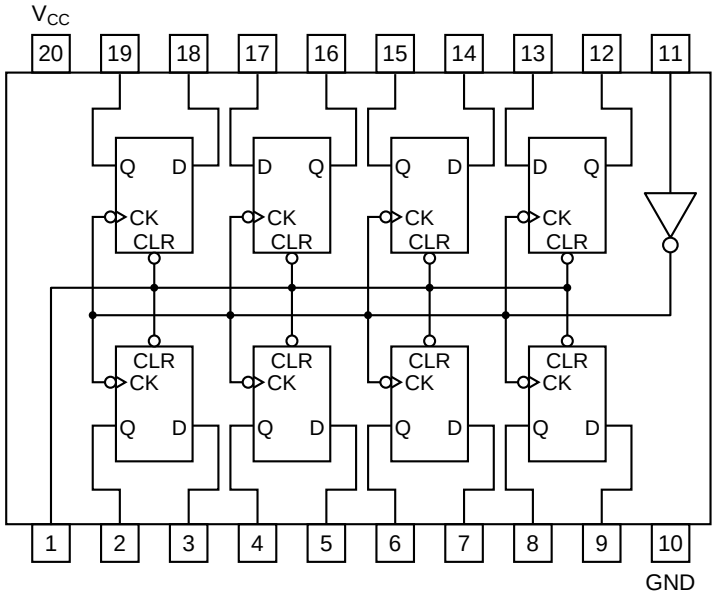
Wir haben die Strecke vom einfachen 1-Bit-Speicher in Abb. 3.24 bis zum 8-Bit-Speicher in Abb. 3.28(b) zurückgelegt. Um große Speicher zu konstruieren, ist aber eine andere Organisation erforderlich, bei der man einzelne Wörter adressieren kann. Eine häufig benutzte Speicherorganisation, die dieses Kriterium erfüllt, ist in Abb. 3.29 dargestellt. Ein Speicher mit vier 3-Bit-Wörtern wird in diesem Beispiel gezeigt. Jede Operation liest oder schreibt ein volles 3-Bit-Wort. Während die gesamte Speicherkapazität von 12 Bits kaum mehr als unsere oktale Flip-Flop-Schaltung beträgt, benötigt sie weniger Pins; vor allem aber läßt sich das Design leicht auf große Speicher erweitern.

Der Speicher in Abb. 3.29 sieht auf den ersten Blick kompliziert aus, ist es dank seiner regelmäßigen Struktur aber nicht. Er hat acht Eingangs- und drei Ausgangsleitungen. Die drei Eingänge sind Daten: I_0, I_1 und I_2; zwei sind für die Adressen A_0 und A_1, und drei dienen der Steuerung: CS für Chip Select, RD zum Unterscheiden zwischen Lesen und Schreiben und OE für Output Enable. Die drei Ausgänge für Daten sind D_0, D_1 und D_2. Im Prinzip kann dieser Speicher in eine 14-Pin-Einheit gepackt werden, einschließlich Strom und Masse. Bei der oktalen Flip-Flop-Schaltung benötigt man hingegen 20 Pins.

Um diesen Speicherchip zu wählen, muß die externe Logik CS und auch RD zum Lesen auf High (logisch 1) und zum Schreiben auf Low (logisch 0) setzen. Die beiden Adreßleitungen müssen gesetzt werden, um anzuzeigen, welches der vier 3-Bit-Wörter gelesen oder geschrieben werden soll. Für eine Leseoperation werden die Dateneingangsleitungen nicht benutzt, das ausgewählte Wort wird aber auf die Datenausgangsleitungen gegeben. Bei einer Schreiboperation werden die auf den Dateneingangsleitungen vorhandenen Bits in das ausgewählte Speicherwort geladen. Die Datenausgangsleitungen werden nicht benutzt.



(a)



(b)

Abb. 3.28: (a) Duale D-Flip-Flop-Schaltung; (b) oktale Flip-Flop-Schaltung

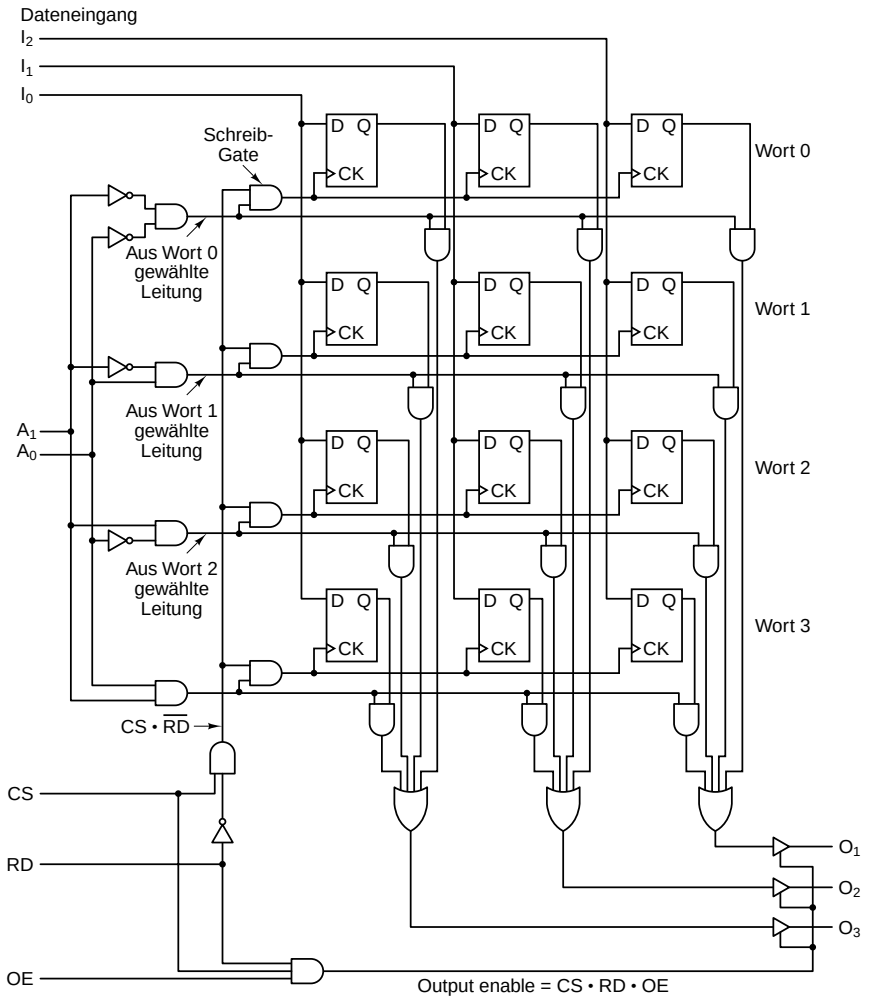


Abb. 3.29: Logikdiagramm für einen 4×3-Speicher. Jede Reihe ist eines der vier 3-Bit-Wörter. Eine Lese- oder Schreiboperation liest bzw. schreibt immer ein ganzes Wort.

Betrachten wir Abb. 3.29 näher, um die Funktionsweise zu sehen. Die vier wortgewählten AND-Gates links vom Speicher bilden einen Dekodierer. Die Eingangsinverter wurden so angeordnet, daß jedes Gate durch eine andere Adresse eingeschaltet wird (Ausgang liegt auf High). Jedes Gate steuert eine wortgewählte Leitung, und zwar von oben nach unten für die Wörter 0, 1, 2 und 3. Wird der Chip für eine Schreiboperation ausgewählt, geht die mit $CS \cdot \overline{RD}$ beschriftete vertikale Leitung auf High, wodurch eines der vier Schreib-Gates eingeschaltet wird, je nachdem, welche wortgewählte Leitung auf High ist. Die Ausgabe des Schreib-Gates steuert alle CK-Signale für das ausgewählte Wort, so daß die Eingangsdaten für das jeweilige Wort in die

Flip-Flop-Schaltungen geladen werden. Es erfolgt eine Schreiboperation nur in dem Fall, daß CS auf High und RD auf Low liegen, und auch dann wird nur das von A_0 und A_1 gewählte Wort geschrieben. Die übrigen Wörter werden nicht verändert.

Der Lesevorgang verläuft ähnlich wie der Schreibvorgang. Die Adresse wird genau so wie beim Schreiben dekodiert. Die Leitung $CS \cdot \overline{RD}$ liegt aber auf Low, so daß alle Schreib-Gates ausgeschaltet sind und keine der Flip-Flop-Schaltungen geändert wird. Statt dessen aktiviert die wortgewählte Leitung die AND-Gates, die mit den *Q*-Bits des gewählten Worts verknüpft sind. Somit gibt das gewählte Wort seine Daten auf die OR-Gates aus, die vier Eingänge haben und unten in der Abbildung erscheinen, während die anderen drei Wörter Nullen ausgeben. Folglich ist die Ausgabe der OR-Gates mit dem im gewählten Wort gespeicherten Wert identisch. Die drei nicht gewählten Wörter tragen nichts zur Ausgabe bei.

Wir hätten auch eine Schaltung auslegen können, bei der die drei OR-Gates lediglich in die drei Ausgabedatenleitungen gespeist werden. Dies führt manchmal aber zu Problemen. Wir haben gezeigt, daß die Dateneingangs- und die Datenausgangsleitungen unterschiedlich sind. In echten Speichern werden aber die gleichen Leitungen benutzt. Hätten wir die OR-Gates mit den Datenausgangsleitungen verknüpft, würde der Chip versuchen, Daten auszugeben. Er würde also jede Leitung in einen bestimmten Wert zwingen, und dies sogar bei Schreiboperationen, was mit den Eingabedaten kollidieren würde. Aus diesem Grund ist eine Möglichkeit wünschenswert, die OR-Gates mit den Datenausgangsleitungen bei Leseoperationen zu verbinden, sie bei Schreiboperationen aber völlig zu trennen. Wir benötigen einen elektronischen Schalter, der eine Verbindung in wenigen Nanosekunden aufbauen und trennen kann.

Zum Glück gibt es einen solchen Schalter. Abb. 3.30(a) zeigt das Symbol eines sogenannten **nicht invertierenden Puffers** (Noninverting Buffer). Er hat einen Dateneingang, einen Datenausgang und einen Steuereingang. Ist der Steuereingang auf High, agiert der Puffer wie ein Draht, was aus Abb. 3.30(b) ersichtlich wird. Liegt der Steuereingang auf Low, verhält sich der Puffer wie eine offene Schaltung, wie in Abb. 3.30(c). Dies hat den Effekt, als ob jemand den Datenausgang vom Rest der Schaltung mit einer Drahtschere abgeschnitten hätte. Im Gegensatz zum Gebrauch einer Drahtschere kann die Verbindung nachträglich aber in wenigen Nanosekunden wiederhergestellt werden, wenn man das Steuersignal wieder auf High setzt.

Abb. 3.30(d) zeigt einen **invertierenden Puffer** (Inverting Buffer), der sich wie ein normaler Inverter verhält, wenn die Steuerung auf High liegt, und den Ausgang von der Schaltung abtrennt, wenn die Steuerung auf Low ist. Beide Pufferarten sind **Tri-State-Geräte**, weil sie 0, 1 oder nichts (offene Schaltung) ausgeben können. Puffer verstärken außerdem Signale, so daß sie mehrere

Eingänge gleichzeitig steuern können. Sie werden manchmal aus diesem Grund in Schaltungen verwendet, obwohl ihre Schaltfähigkeiten nicht benötigt werden.

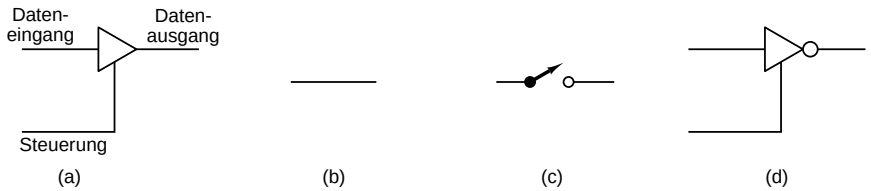


Abb. 3.30: (a) Nicht invertierender Puffer; (b) Wirkung von (a), wenn die Steuerung auf High liegt; (c) Wirkung von (a), wenn die Steuerung auf Low liegt; (d) invertierender Puffer

Wenden wir uns wieder der Speicherschaltung zu. Inzwischen dürfte klar sein, welchen Zweck die drei nicht invertierenden Puffer auf den Datenausgangsleitungen haben. Liegen CS, RD und OE auf High, ist auch das Ausgabeschaltsignal auf High, wodurch die Puffer eingeschaltet werden und ein Wort auf die Ausgangsleitungen gegeben wird. Liegt entweder CS oder RD oder OE auf Low, werden die Datenausgänge vom Rest der Schaltung getrennt.

3.3.5 Speicherchips

Angenehm an dem Speicher von Abb. 3.29 ist, daß er von seiner Größe her mühelos erweitert werden kann. Ursprünglich haben wir ihn mit 4×3, d.h. vier Wörtern von je 3 Bits, gezeichnet. Möchten wir ihn auf 4×8 erweitern, müssen wir lediglich fünf weitere Spalten von je vier Flip-Flop-Schaltungen, fünf weitere Eingangsleitungen und fünf Ausgangsleitungen hinzufügen. Rüsten wir von 4×3 auf 8×3 auf, müssen wir vier weitere Zeilen von je drei Flip-Flop-Schaltungen und eine Adreßzeile A_2 hinzufügen. Bei dieser Struktur sollte die Anzahl der Wörter im Speicher eine Potenz von 2 betragen, um maximale Effizienz bei beliebiger Bitanzahl im Wort zu erreichen.

Da sich die integrierte Schaltkreistechnik gut zur Herstellung von Chips eignet, deren interne Struktur ein wiederkehrendes, zweidimensionales Muster aufweist, sind Speicherchips der ideale Anwendungsbereich dafür. Im Zuge der Verbesserung dieser Technik steigt die Bitanzahl, die man auf einen Chip packen kann, ständig, etwa um einen Faktor von Zwei alle 18 Monate (Mooresches Gesetz). Durch größere Chips werden kleinere nicht immer veraltet, weil hinsichtlich Kapazität, Geschwindigkeit, Leistung, Preis und Schnittstellen viele Kompromisse gemacht werden müssen. In der Regel sind die derzeit erhältlichen größten Chips im Vergleich zu den älteren, kleineren pro Bit wesentlich teurer.

Der Chip kann für jede Speichergröße unterschiedlich organisiert werden. Abb. 3.31 zeigt zwei Aufbaumöglichkeiten eines 4-Mbit-Chips: 512K $\times$ 8 und

4096K × 1. (Speicherchipgrößen werden normalerweise in Bits, und nicht in Bytes angeboten. Deshalb halten wir uns ebenfalls an diese Konvention.) In Abb. 3.31(a) sind 19 Adreßleitungen erforderlich, um eines der 2^{19} Byte zu adressieren. Zum Laden oder Speichern des ausgewählten Bytes braucht man acht Datenleitungen.

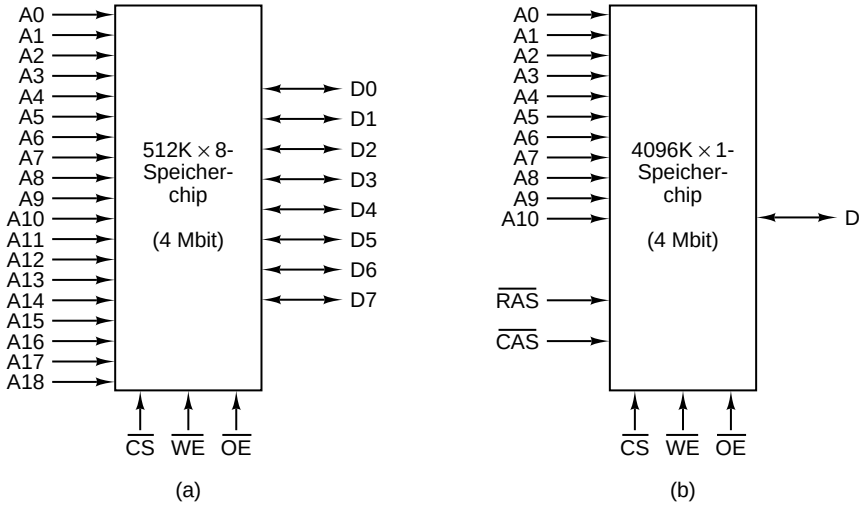


Abb. 3.31: Zwei Arten der Organisation eines 4-Mbit-Speicherchips

An dieser Stelle eine Anmerkung zur Terminologie: An einigen Pins erfolgt durch hohe Spannung eine Aktion. Bei anderen verursacht die niedrige Spannung eine Aktion. Um Verwechslungen zu vermeiden, sprechen wir im weiteren Verlauf davon, daß ein Signal **asseriert** (statt »auf High oder Low gesetzt wird«), um anzudeuten, daß es für das Bewirken einer Aktion gesetzt wird. Bei einigen Pins bedeutet Asserieren, daß sie auf High gelegt werden. Bei anderen bedeutet es, daß der Pin auf Low gesetzt wird. Bei Pins, die auf Low liegen, werden die Signalbezeichnungen mit einem Überstrich angegeben. Ein Signal namens CS wird also auf High und eines mit der Bezeichnung $\overline{CS}$ auf Low gelegt. Das Gegenteil von Asserieren ist **Negieren**. Wenn nichts Besonderes passiert, werden Pins negiert.

Zurück zu unserem Speicherchip. Da ein Computer normalerweise viele Speicherchips hat, ist ein Signal erforderlich, um den aktuell benötigten Chip zu wählen, damit er und kein anderer reagiert. Das Signal $\overline{CS}$ (Chip Select) erfüllt diesen Zweck. Es wird zur Chipaktivierung asseriert. Außerdem muß man zwischen Lese- und Schreiboperationen unterscheiden können. Das Signal $\overline{WE}$ (Write Enable) wird benutzt, um zu bezeichnen, daß Daten geschrieben, und nicht gelesen werden. Schließlich wird das Signal $\overline{OE}$ (Output Enable) zur Steuerung der Ausgangssignale asseriert. Bei Nichtasserierung wird der Chipausgang von der Schaltung getrennt.

In Abb. 3.31(b) wird ein anderes Adressierschema benutzt. Intern ist dieser Chip als Matrix von 2048 × 2048 1-Bit-Zellen organisiert, die 4 Mbit ergeben. Zur Chipadressierung wird zuerst eine Zeile ausgewählt, indem man ihre 11-Bit-Zahl auf die Adreßpins legt. Dann wird $\overline{RAS}$ (Row Address Strobe) asseriert. Anschließend wird eine Spaltennummer auf die Adreßpins gelegt und $\overline{CAS}$ (Column Address Strobe) asseriert. Der Chip reagiert durch Annahme oder Ausgabe eines Datenbits.

Große Speicherchips werden oft als $n \times n$-Matrizen ausgelegt, die durch Zeile und Spalte adressiert werden. Dieser Aufbau reduziert die benötigte Pinanzahl, verlangsamt aber auch die Adressierung des Chips, da zwei Adressierzyklen erforderlich sind – einer je Zeile und Spalte. Um einen Teil der Geschwindigkeit zurückzugewinnen, können Speicherchips mit einer Zeilenadresse, gefolgt von mehreren Spaltenadressen, eingerichtet werden, so daß aufeinanderfolgende Bits in einer Reihe adressiert werden können.

Vor Jahren wurden die größten Speicherchips allgemein wie in Abb. 3.31(b) ausgelegt. Im Zuge der Vergrößerung von Speicherwörtern von 8 auf 32 Bits und mehr wurden 1 Bit große Chips unpraktisch. Um einen Speicher mit einem 32-Bit-Wort aus 4096K × 1 großen Chips zu entwickeln, werden 32 parallele Chips benötigt. Diese 32 Chips haben eine Gesamtkapazität von mindestens 16 Mbyte, während bei Verwendung von 512K × 8-Chips nur vier parallele Chips nötig und kleine Speicher, etwa 2 Mbyte, möglich sind. Um die Verwendung von 32 Chips für einen Speicher zu vermeiden, haben die meisten Chiphersteller heute Chipfamilien mit Breiten von 1, 4, 8 und 16 Bits.

3.3.6 RAM und ROM

Alle Speicher, die wir bisher untersucht haben, können lesen und schreiben. Solche Speicher nennt man **RAMs** (Random Access Memories) – eine falsche Bezeichnung, weil alle Speicherchips zufallsgesteuerten Zugriff haben. RAMs gibt es in zwei Varianten: statisch und dynamisch. **Statische RAMs** (**SRAMs**) werden intern mit Schaltungen realisiert, die unserer grundlegenden D-Flip-Flop-Schaltung ähneln. Diese Speicher haben die Fähigkeit, ihre Inhalte zu behalten, solange der Strom fließt: Sekunden, Minuten, Stunden oder gar Tage. Statische RAMs sind sehr schnell. Die typischen Zugriffszeiten betragen nur wenige Nanosekunden. Aus diesem Grund sind RAMs als Cache-Speicher der Ebene 2 (Cache Level-2) sehr beliebt.

Demgegenüber werden in **dynamischen RAMs** (**DRAMs**) keine Flip-Flop-Schaltungen benutzt. Ein dynamischer RAM besteht aus einer Reihe von Zellen, wobei jede Zelle einen Transistor und einen winzigen Kondensator enthält. Kondensatoren können ge- und entladen werden, so daß sich Nullen und Einsen speichern lassen. Da elektrische Ladung zu Kriechverlusten neigt, muß jedes Bit in einem dynamischen RAM zur Vermeidung von Datenverlust alle paar Millisekunden **aufgefrischt** (nachgeladen) werden. Weil sich externe

Logik um die Auffrischung kümmern muß, erfordern dynamische RAMs komplexere Schnittstellen als statische. Dieser Nachteil gleicht sich in vielen Anwendungen aber durch ihre größeren Kapazitäten aus.

Dynamische RAMs benötigen nur einen Transistor und einen Kondensator pro Bit (gegenüber sechs Transistoren pro Bit beim besten statischen RAM). Deshalb haben dynamische RAMs eine sehr hohe Dichte (viele Bits pro Chip). Aus diesem Grund werden Hauptspeicher fast immer mit dynamischem RAM realisiert. Diese große Kapazität hat aber ihren Preis: Dynamische RAMs sind langsam (zig Nanosekunden). Durch eine Kombination aus statischem RAM-Cache und dynamischem RAM-Hauptspeicher wird versucht, die guten Eigenschaften von beiden zu nutzen.

Dynamische RAM-Chips sind in verschiedenen Typen erhältlich. Der älteste Typ ist der **FPM-DRAM** (FPM = Fast Page Mode); er ist immer noch in Gebrauch. Intern ist dieser DRAM als Bitmatrix organisiert. Er funktioniert mittels Präsentation einer Leitungsadresse durch die Hardware. Dann werden die Spaltenadressen durchlaufen, wie in Zusammenhang mit $\overline{RAS}$ und $\overline{CAS}$ von Abb. 3.31(b) beschrieben wurde.

FPM-DRAMs werden nach und nach durch den **EDO-DRAM** (Extended Data Output) abgelöst. Beim EDO-DRAM kann eine zweite Speicherreferenz vor Beendigung der ersten beginnen. Dieses einfache Pipelining beschleunigt eine einzelne Speicherreferenz nicht, verbessert aber die Speicherbandbreite, d.h., ein Durchschleusen von mehr Wörtern pro Sekunde ist möglich.

FPM- und EDO-Chips sind asynchron, was bedeutet, daß die Adreß- und Datenleitungen nicht durch einen einzigen Taktgeber gesteuert werden. Demgegenüber ist der **SDRAM** (Synchronous DRAM) eine Mischung aus statischem und dynamischem RAM; er wird durch einen einzelnen, synchronen Taktgeber gesteuert. SDRAMs werden häufig in großen Cache-Speichern benutzt und gelten als die bevorzugte Technologie für Hauptspeicher der Zukunft.

RAMs sind nicht die einzige Art von Speicherchips. In vielen Anwendungen, z.B. Spielzeugen, Elektrogeräten und Autos, müssen das Programm und ein Teil der Daten auch bei ausgeschaltetem Strom gespeichert bleiben. Außerdem werden weder das Programm noch die Daten nach der Installation jemals geändert. Diese Anforderungen haben zur Entwicklung von **ROMs** (Read-Only Memories) geführt, die weder absichtlich noch anderweitig geändert oder gelöscht werden können. Beim ROM werden die Daten bei der Herstellung des ROMs auf der Oberfläche eingebrannt. Will man das in einem ROM gespeicherte Programm ändern, muß man den gesamten Chip ersetzen. Es gibt keine andere Änderungsmöglichkeit.

ROMs sind viel billiger als RAMs, wenn man sie in großen Mengen bestellt, weil sich durch die Menge die Kosten für die Herstellung der Maske amortisieren. Sie sind aber unflexibel, weil man sie nach der Herstellung nicht ändern kann. Außerdem können zwischen der Bestellung und der Auslieferung

viele Wochen vergehen. Um es Unternehmen zu vereinfachen, neue ROM-basierte Produkte zu entwickeln, wurde der **PROM** (Programmable ROM) erfunden. Dieser Chip arbeitet wie ein ROM, ausgenommen, daß man ihn (einmal) im Feld programmieren kann, wodurch sich die Vorlaufzeit erheblich verkürzt. Viele PROMs enthalten eine Reihe winziger Sicherungen. Diese Sicherungen können selektiv durchgebrannt werden, wenn man eine Zeile und Spalte wählt und dann auf einen der Pins Hochspannung zur Zerstörung einer bestimmten Sicherung aufbringt.

Die nächste Entwicklung dieser Linie war der **EPROM** (Erasable PROM), den man im Feld nicht nur programmieren sondern auch löschen kann. Wird das Quarzfenster in einem EPROM 15 Minuten lang einem starken ultravioletten Licht ausgesetzt, werden alle Bits auf 1 gesetzt. Gibt es im Verlauf des Designzyklus viele Änderungen, sind EPROMs weitaus wirtschaftlicher als PROMs, weil man sie wiederverwenden kann. EPROMs sind normalerweise als statische RAMs organisiert. Der 4-Mbit-EPROM 27C040 basiert beispielsweise auf einer Struktur wie in Abb. 3.31(a), die für einen statischen RAM typisch ist.

Noch besser als der EPROM ist der **EEPROM**, der gelöscht werden kann, wenn man ihn Impulsen statt in einer speziellen Kammer ultraviolettem Licht aussetzt. Außerdem kann der EEPROM vor Ort neu programmiert werden, während ein EPROM zu diesem Zweck in ein spezielles EPROM-Programmiergerät eingelegt werden muß. Als Nachteil wäre festzustellen, daß die größten EEPROMs normalerweise nur 1/64 so groß und halb so schnell sind wie übliche EPROMs. EEPROMs können nicht mit DRAMs oder SRAMs konkurrieren, weil sie zehnmal langsamer, 100 Mal kleiner und viel teurer sind. Sie werden nur in Fällen benutzt, in denen 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 der Flash-Speicher blockweise gelöscht und wiederbeschrieben werden. Wie der EEPROM kann der Flash-Speicher gelöscht werden, ohne daß man ihn aus der Schaltung herausnimmt. Verschiedene Hersteller produzieren kleine gedruckte Schaltkarten, die mit etlichen Mbyte an Flash-Speicher bestückt sind und als eine Art »Film« zum Speichern von Bildern in digitalen Kameras und ähnlichen Anwendungen benutzt werden. Eines Tages lösen Flash-Speicher möglicherweise Platten ab. Das wäre angesichts ihrer Zugriffszeiten von 100 ns eine enorme Verbesserung. Das größte Konstruktionsproblem ist derzeit, daß sie verschleißen und nach etwa 10000 Löschungen unbrauchbar sind, während Platten Jahre überdauern, ganz egal, wie oft man sie löscht und neu beschreibt. Abb. 3.32 zeigt eine Übersicht der verschiedenen Speicherarten.

Typ	Kategorie	Löschung	Byte änderbar	Flüchtig	Übliche Verwendung
SRAM	Lesen/Schreiben	Elektrisch	Ja	Ja	Level-2-Cache
DRAM	Lesen/Schreiben	Elektrisch	Ja	Ja	Hauptspeicher
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

Abb. 3.32: Gegenüberstellung verschiedener Speicherarten

3.4 CPU-Chips und Busse

Mit Informationen über SSI-Chips, MSI-Chips und Speicher-Chips ausgestattet, können wir nun beginnen, alle Teile zusammenzusetzen, um Komplettsysteme zu betrachten. In diesem Abschnitt werden zuerst einige allgemeine Aspekte von CPUs aus Sicht der digitalen logischen Ebene untersucht, darunter **Pinout** (die Bedeutung der Signale auf den verschiedenen Pins). Da CPUs so eng mit dem Design der von ihnen benutzten Busse verflochten sind, werden in diesem Abschnitt auch Busse vorgestellt. In späteren Abschnitten werden ausführliche Beispiele von CPUs mit den jeweiligen Bussen beschrieben.

3.4.1 CPU-Chips

Fast alle modernen CPUs befinden sich auf einem einzigen Chip. Dadurch ist ihre Interaktion mit dem restlichen System gut definiert. Jeder CPU-Chip hat eine Reihe von Pins, durch die seine gesamte Kommunikation mit der Außenwelt fließen muß. Einige Pins geben Signale von der CPU aus, andere nehmen Signale von der Außenwelt entgegen, und wieder andere können beides. Kennt man die Funktion der Pins, wird verständlich, wie die CPU mit dem Speicher und den E/A-Geräten auf der digitalen logischen Ebene interagiert.

Die Pins auf einem CPU-Chip sind drei verschiedenen Zielrichtungen zugeordnet: Adresse, Daten und Steuerung. Diese Pins sind mit ähnlichen Pins auf den Speicher- und E/A-Chips über eine Sammlung von parallelen Drähten, die man »Bus« nennt, verbunden. Um eine Instruktion zu lesen, setzt die CPU zuerst die Speicheradresse dieser Instruktion auf ihre Adreßpins. Dann asseriert sie eine oder mehrere Steuerleitungen, um den Speicher darüber zu informieren, daß ein Wort gelesen werden soll. Der Speicher reagiert, indem

er das angeforderte Wort auf die Datenpins der CPU legt und ein Signal asseriert, daß die Anforderung erfüllt wurde. Sieht die CPU dieses Signal, nimmt sie das Wort an und führt die Instruktion aus.

Die Instruktion kann das Lesen oder Schreiben von Datenwörtern erfordern. In diesem Fall wird der gesamte Prozeß für jedes weitere Wort wiederholt. Die Funktionsweise von Lesen und Schreiben wird später behandelt. Vorläufig interessiert uns nur, wie die CPU mit Speicher und E/A-Geräten kommuniziert: Sie legt Signale auf ihre Pins und nimmt Signale von Pins entgegen. Eine andere Kommunikation ist nicht möglich.

Zwei der wichtigsten Parameter, mit denen die Leistung einer CPU ermittelt werden kann, sind die Anzahl von 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. Ein Chip mit n Datenpins kann 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 mit 32 Datenpins die gleiche Arbeit in einer Operation erledigt. Der Chip mit 32 Datenpins ist also viel schneller, aber auch erheblich teurer.

Zusätzlich zu Adreß- und Datenpins hat jede CPU einige Steuerpins. Diese Pins steuern den Fluß und den Takt von Daten von und zur CPU; sie dienen verschiedenen Zwecken. Alle CPUs haben Pins für Strom (normalerweise +3,3 oder +5 Volt), Masse und ein Taktsignal (eine Quadratwelle). Die übrigen Pins weichen von einem Chip zum anderen aber erheblich ab. Dennoch kann man die Steuerpins grob in folgende Kategorien gruppieren:

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

Diese Kategorien werden weiter unten allgemein beschrieben. Betrachten wir in späteren Kapiteln Pentium II, UltraSPARC II und picoJava II-Chips, werden weitere Einzelheiten erläutert. Ein allgemeiner CPU-Chip mit diesen Signalgruppen wird in Abb. 3.33 gezeigt.

Die Bussteuerpins sind vorwiegend Ausgänge von der CPU zum Bus (und damit Eingänge in den Speicher und die E/A-Chips). Sie zeigen an, ob die CPU in den Speicher schreiben, von dort lesen oder etwas anderes machen möchte.

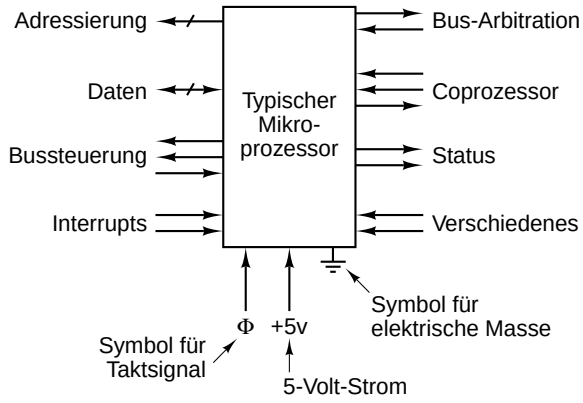


Abb. 3.33: Logisches Pinout einer typischen (hypothetischen) CPU. Die Pfeile bedeuten Ein- und Ausgangssignale. Die kurzen diagonalen Linien zeigen die Benutzung mehrerer Pins an. Für eine spezifische CPU wird die genaue Zahl angegeben.

Die Interrupt-Pins sind Eingänge von E/A-Geräten zur CPU. Bei den meisten Systemen kann die CPU ein E/A-Gerät anweisen, eine Operation zu starten, und dann etwas Nützliches tun, während das langsame E/A-Gerät seine Arbeit verrichtet. Ist der E/A-Vorgang beendet, asseriert der E/A-Controller-Chip ein Signal auf eines dieser Pins, um die CPU zu unterbrechen, damit sie sich um das E/A-Gerät kümmert, z.B. um zu prüfen, ob ein E/A-Fehler aufgetreten ist. Einige CPUs haben einen Ausgangspin zur Bestätigung von Interrupt-Signalen.

Die Bus-Arbitration-Pins werden benötigt, um den Verkehr auf dem Bus zu regeln, damit zwei Geräte an dessen gleichzeitiger Benutzung gehindert werden. In dieser Hinsicht zählt die CPU selbst als Gerät.

Einige CPU-Chips sind so ausgelegt, daß sie mit Coprozessoren, z.B. Gleitkommachips oder Grafikchips, arbeiten. Um die Kommunikation zwischen der CPU und einem Coprozessor zu vereinfachen, gibt es spezielle Pins, die verschiedene Anfragen stellen und erfüllen.

Zusätzlich zu diesen Signalen kann eine CPU verschiedene Pins haben. Einige davon dienen der Bereitstellung oder Annahme von Statusinformationen, andere setzen den Computer zurück, und wieder andere gewährleisten Kompatibilität mit älteren E/A-Chips.

3.4.2 Computer-Busse

Ein **Bus** ist ein gemeinsamer, elektrischer Weg zwischen mehreren Geräten. Busse werden nach ihrer Funktion kategorisiert. Man kann sie intern in der CPU benutzen, um Daten von und zur ALU zu befördern, oder extern, um die CPU mit einem Speicher oder E/A-Geräten zu verbinden. Jede Busart hat ihre eigenen Anforderungen und Merkmale. In diesem und in den nächsten Abschnitten konzentrieren wir uns auf Busse, die eine CPU mit dem Speicher

und E/A-Geräten verbinden. Im nächsten Kapitel werden CPU-interne Busse beschrieben.

Die ersten Personalcomputer hatten einen einzigen externen Bus, den **Systembus**. Er bestand aus 50 bis 100 parallelen Kupferdrähten, die auf das Motherboard gelötet waren. In regelmäßigen Abständen befanden sich Stecker, um Speicher und E/A-Karten einzubauen. Moderne Personalcomputer haben im allgemeinen einen spezialisierten Bus zwischen der CPU und dem Speicher und (mindestens) einen weiteren für die E/A-Geräte. Abb. 3.34 zeigt ein kleines System mit einem Speicherbus und einem E/A-Bus.

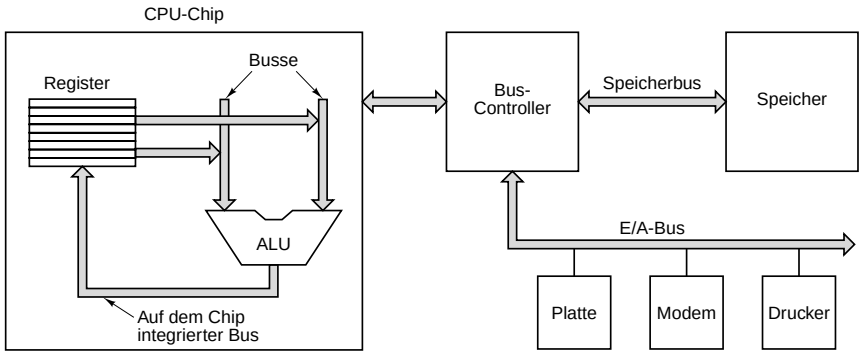


Abb. 3.34: Ein Computersystem mit mehreren Bussen

In der Literatur werden Busse zuweilen als »fette« Pfeile gezeichnet, wie in Abb. 3.34. Der Unterschied zwischen einem fetten Pfeil und einer einfachen Linie, die mit einer diagonalen Linie durchzogen und mit einer Bitzahl beschriftet ist, ist sehr gering, aber gleichwohl genau zu beachten. Sind alle Bits vom gleichen Typ, etwa alle Adreß- oder alle Datenbits, wird die kurze diagonale Linie als Symbol benutzt. Liegen Adreß-, Daten- und Steuerleitungen vor, ist ein fetter Pfeil üblich.

CPU-Entwickler können einen Chip im Grunde genommen mit jedem beliebigen Bus ausstatten. Sollen aber Platinen unterschiedlicher Hersteller an den Systembus angeschlossen werden, müssen gut definierte Regeln bezüglich der Funktionsweise des Busses beachtet werden. Diese Regeln müssen von allen anzuschließenden Geräten befolgt werden. Insgesamt nennt man das Regelwerk **Busprotokoll**. Darüber hinaus muß eine mechanische und elektrische Spezifikation vorliegen, damit Platinen verschiedener Hersteller in den Kartenrahmen passen und mit Steckern ausgestattet sind, die in die Buchsen des Motherboards passen, sowohl physisch als auch bezüglich Spannung.

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), IBM-PC-Bus (PC/XT), ISA-Bus (PC/AT), EISA-Bus (80386), Microchannel (PS/2), PCI-Bus (viele PCs), SCSI-Bus (viele PCs und Workstations), Nubus (Macintosh), Universal Serial Bus (moderne PCs), FireWire

(Verbrauchselektronik), VME-Bus (Physiklaboranlagen) und Camac-Bus (Physik). Die Welt wäre wahrscheinlich ein angenehmerer Platz, wenn alle Busse außer einem plötzlich von der Erdoberfläche verschwinden würden (na ja, wie wäre es mit allen außer zweien). Leider scheint eine Standardisierung in diesem Bereich sehr unwahrscheinlich, weil bereits zuviel in diese verschiedenen, untereinander nicht kompatiblen Systeme investiert wurde.

Wie funktionieren Busse? Einige Geräte, die man an einen Bus anschließt, sind aktiv und können Bustransfers einleiten, während andere passiv sind und auf Anfragen warten. Die aktiven nennt man **Master**, die passiven **Slaves**. Fordert die CPU einen Platten-Controller auf, einen Block zu lesen oder zu schreiben, tritt die CPU als Master und der Platten-Controller als Slave auf. Im weiteren Verlauf werden wir noch sehen, daß der Platten-Controller auch als Master auftreten kann, wenn er dem Speicher befiehlt, die von ihm vom Plattenlaufwerk gelesenen Wörter zu akzeptieren. Abb. 3.35 zeigt mehrere übliche Master/Slave-Kombinationen. Unter keinen Umständen kann ein Speicher jemals ein Master sein.

Master	**Slave**	**Beispiel**
CPU	Speicher	Instruktionen und Daten einlesen
CPU	E/A-Gerät	Datentransfer einleiten
CPU	Coprozessor	CPU übergibt dem Coprozessor Instruktionen
E/A	Speicher	DMA (Direct Memory Access)
Coprozessor	CPU	Coprozessor holt Operanden von der CPU

Abb. 3.35: Beispiele von Bus-Mastern und –Slaves

Die binären Signale, die Computer-Geräte ausgeben, sind häufig nicht stark genug, um einen Bus zu betreiben, insbesondere, wenn der Bus relativ lang ist oder viele Geräte an ihn angeschlossen sind. Aus diesem Grund werden die meisten Bus-Master mit Hilfe eines Chips an den Bus angeschlossen, den man **Bustreiber** (Bus Driver) nennt. Dabei handelt es sich im wesentlichen um einen digitalen Verstärker. Die meisten Slaves werden über einen **Busempfänger** (Bus Receiver) an den Bus angeschlossen. Für Geräte, die sowohl als Master als auch als Slave auftreten können, wird ein Kombichip namens **Bus-Transceiver** benutzt. Diese Busschnittstellen-Chips sind meist Tri-State-Geräte, damit sie gleiten (sich abtrennen) können, wenn man sie nicht braucht. Eine Alternative ist der **Open-Collector**, mit dem man die gleiche Wirkung erzielt. Asserieren zwei oder mehr Geräte an einer Open-Collector-Leitung die Leitung gleichzeitig, ist das Ergebnis das boolesche OR aller Signale. Diese Anordnung nennt man auch **Wired-OR**. Bei den meisten Bussen kommen teilweise Tri-State- und Open-Collector-Leitungen zum Einsatz, wenn die Wired-OR-Eigenschaft benötigt wird.

Wie eine CPU hat auch ein Bus Adreß-, Daten- und Steuerleitungen. Es besteht aber nicht notwendigerweise eine Eins-zu-Eins-Abbildung zwischen der

CPU und den Bussignalen. Beispielsweise haben einige CPUs drei Pins zum Kodieren der verschiedenen Operationen. Ein typischer Bus hat je eine Leitung zum Lesen des Speichers, zum Schreiben des Speichers, zum Lesen von E/A-Geräten, zum Schreiben auf E/A-Geräte usw. Zwischen der CPU und einem solchen Bus ist ein Dekodiererchip erforderlich, um das 3-Bit-kodierte Signal in getrennte Signale zu konvertieren, mit denen die Busleitungen betrieben werden können.

Design und Betrieb von Bussen sind recht komplexe Themen, denen dicke Bücher gewidmet wurden (Shanley und Anderson, 1995b; Solari, 1993; Solari und Willse, 1998). Die prinzipiellen Kriterien beim Busdesign sind Busbreite, Bustaktung, Bus-Arbitration und Busoperationen. Diese Kriterien haben großen Einfluß auf Geschwindigkeit und Bandbreite des Busses. Sie werden in den nächsten vier Abschnitten behandelt.

3.4.3 Busbreite

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

Breite Busse erfordern mehr Kupferdrähte als schmalere. Sie beanspruchen auch mehr Platz (auf dem Motherboard) und größere Connectors. Durch diese Faktoren wird der Bus teurer. Folglich muß zwischen der maximalen Speichergröße und den Systemkosten ein Kompromiß gefunden werden. Ein System mit 64-Leitungen-Adreßbus und 2^{32} Bytes an Speicher kostet mehr als eines mit 32 Adreßleitungen und dem gleichen Speicher. Eine spätere Erweiterung ist ebenfalls nicht kostenlos.

Aus diesem Grund neigen viele Systemdesigner zu Kurzsichtigkeit, und dies mit bedauerlichen Folgen. Der ursprüngliche IBM-PC enthielt eine 8088-CPU und einen 20-Bit-Adreßbus (siehe Abb. 3.36(a)). Damit hatte der PC 20 Bits zur Adressierung eines Speichers von einem Mbyte.

Als der nächste CPU-Chip (80286) herauskam, entschloß sich Intel, den Adreßraum auf 16 Mbyte zu vergrößern, so daß vier weitere Busleitungen hinzugefügt werden mußten (ohne die ursprünglichen 20 aus Gründen der Abwärtskompatibilität zu stören); wir sehen dies in Abb. 3.36(b). Leider mußten weitere Steuerleitungen zur Unterstützung der neuen Adreßleitungen hinzugefügt werden. Als der 80386 herauskam, gab es weitere acht Adreßleitungen und noch mehr Steuerleitungen, wie wir in Abb. 3.36(c) sehen. Das daraus resultierende Design (der EISA-Bus) wäre nicht so schlampig geworden, wenn man den Bus von Anfang an mit 32 Leitungen ausgestattet hätte.

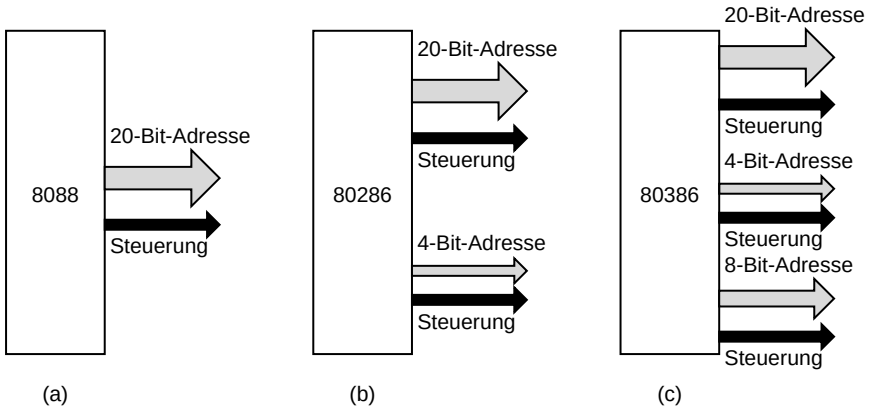


Abb. 3.36: Wachstum eines Adreßbusses im Lauf der Zeit

Nicht nur die Anzahl der Adreßleitungen neigt dazu, im Lauf der Zeit zuzunehmen, sondern auch die der Datenleitungen, allerdings aus einem anderen Grund. Es gibt zwei Möglichkeiten, die Datenbandbreite eines Busses zu erhöhen: Verringerung der Buszykluszeit (mehr Transfers/Sekunde) oder Erhöhung der Datenbusbreite (mehr Bit/Transfer). Auch eine Beschleunigung des Busses ist möglich, aber schwierig, weil die Signale in den verschiedenen Leitungen in unterschiedlichen Geschwindigkeiten fließen, was als **Bus-Skew** (Bus-Schräglauf) bezeichnet wird. Je schneller der Bus, um so höher dieser Faktor.

Ein weiteres Problem bei der Beschleunigung des Busses ist der Verlust der Abwärtskompatibilität. Ältere Platinen, die für den langsameren Bus ausgelegt wurden, funktionieren nicht am neuen Bus. Die Beseitigung der älteren Platinen bereitet Besitzern und Herstellern Kopfschmerzen. Deshalb wird die Leistung normalerweise dadurch verbessert, daß man mehr Datenleitungen hinzufügt, wie aus Abb. 3.36 ersichtlich wird. Selbstverständlich führt dieser Ansatz nicht zu einem sauberen Design. Der IBM-PC und seine Nachfolger veränderten sich beispielsweise von 8 Datenleitungen auf 16 und dann 32 am gleichen Bus.

Zur Vermeidung sehr breiter Busse entscheiden sich Designer manchmal für einen **gemultiplexten Bus**. Bei diesem Design liegen die Adreß- und Datenleitungen nicht separat, sondern z.B. 32 Leitungen für Adressen und Daten zusammen. Zu Beginn einer Busoperation werden die Leitungen für die Adresse benutzt. Später werden sie für die Daten gebraucht. Das heißt z.B. bei einem Schreibvorgang auf den Speicher, daß die Adreßleitungen zum Speicher eingerichtet werden müssen, bevor man Daten auf den Bus schicken kann. Durch Multiplexen der Leitungen reduziert sich die Bandbreite (und der Preis); es führt aber auch zu einer Verlangsamung des Systems. Busdesigner müssen diese Möglichkeiten sorgfältig abwägen.

3.4.4 Bustaktung

Busse lassen sich je nach ihrer Taktung in zwei Kategorien unterteilen. Ein **synchroner Bus** hat eine Leitung, die von einem Kristalloszillator gesteuert wird. Das Signal auf dieser Leitung besteht aus einer Quadratwelle mit einer Frequenz von 5 MHz bis 100 MHz. Alle Busaktivitäten nehmen eine Ganzzahl dieser Zyklen, die sogenannten **Buszyklen**. Die zweite Variante ist der **asynchrone Bus** ohne Mastertakt. Die Buszyklen können die benötigte Länge haben und müssen nicht unbedingt zwischen allen Gerätepaaren gleich sein. Die beiden Bustypen werden im folgenden beschrieben.

Synchrone Busse

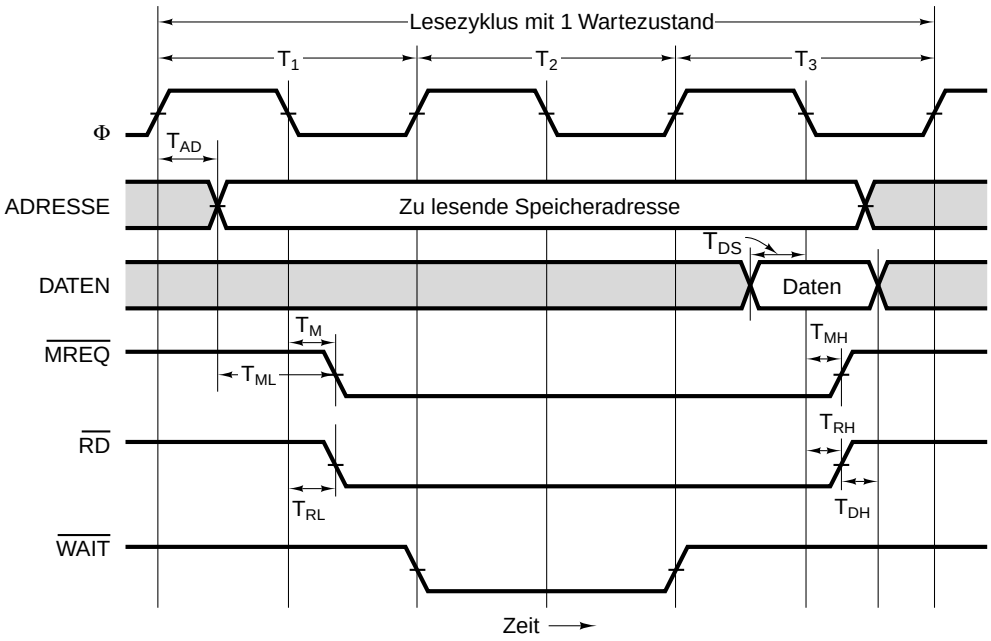
Betrachten wir die Taktkonfiguration in Abb. 3.37(a) als Beispiel der Funktionsweise eines synchronen Busses. Bei diesem Beispiel verwenden wir einen Takt von 40 MHz, was einen Buszyklus von 25 ns ergibt. Dies mag im Vergleich zu CPU-Geschwindigkeiten von 500 MHz und mehr langsam erscheinen; es gibt aber kaum schnellere PC-Busse. Der in allen Intel-PCs vorhandene ISA-Bus läuft beispielsweise mit 8,33 MHz und auch der beliebte PCI-Bus höchstens mit 33 oder 66 MHz. Die Gründe für die Langsamkeit der heutigen Busse wurden bereits erwähnt: technische Designprobleme wie Bus-Skew und Abwärtskompatibilität.

Wir gehen in unserem Beispiel davon aus, daß das Lesen vom Speicher ab dem Zeitpunkt, an dem die Adresse stabil ist, 40 ns dauert. Wir werden gleich sehen, daß es mit diesen Parametern drei Buszyklen zum Lesen eines Worts dauert. Der erste Zyklus beginnt an der steigenden Flanke von T_1, und der dritte endet an der steigenden Flanke von T_4. Keine der steigenden oder fallenden Flanken wurde vertikal gezeichnet, weil kein elektrisches Signal seinen Wert in Nullzeit ändern kann. Bei diesem Beispiel nehmen wir an, daß die Änderung eines Signals 1 ns dauert. Alle Leitungen (Takt, Adresse, Daten, $\overline{MREQ}$, $\overline{RD}$ und $\overline{WAIT}$) werden auf der gleichen Zeitskala dargestellt.

Der Beginn von T_1 wird durch die steigende Flanke des Taktgebers bestimmt. Irgendwo auf dem Weg durch T_1 legt die CPU die Adresse des gewünschten Worts auf die Adreßleitungen. Da die Adresse im Gegensatz zum Takt kein Einzelwert ist, können wir sie in der Abbildung nicht als einzelne Leitung darstellen. Sie erscheint vielmehr in zwei Linien mit einem Kreuzpunkt an der Zeit, an der sich die Adresse ändert. Außerdem bedeutet die Schattierung vor dem Kreuzpunkt, daß der schattierte Wert nicht von Belang ist. Mit Hilfe der gleichen Schattierung kann gezeigt werden, daß der Inhalt der Datenleitungen bis weit in T_3 hinein bedeutungslos ist.

Nachdem sich die Adreßleitungen auf die neuen Werte eingependelt haben, werden $\overline{MREQ}$ und $\overline{RD}$ asseriert. Ersteres bedeutet, daß auf einen Speicher (und nicht auf ein E/A-Gerät) zugegriffen wird, während zweiteres zum Lesen und Negieren von Schreiboperationen erforderlich ist. Da der Speicher nach der Stabilisierung der Adresse (teilweise im ersten Taktzyklus) 40 ns braucht,

kann er die angeforderten Daten nicht innerhalb von T_2 liefern. Damit die CPU die Daten nicht zu früh erwartet, asseriert der Speicher die Leitung *WAIT* zu Beginn von T_2. Diese Aktion fügt **Wartezustände** (zusätzliche Buszyklen) ein, bis der Speicher fertig ist und *WAIT* negiert. In unserem Beispiel wurde ein Wartezustand (T_2) eingefügt, weil der Speicher zu langsam ist. Der Speicher negiert *WAIT* zu Beginn von T_3, wenn er bereits sicher ist, daß die Daten im laufenden Zyklus bereitgestellt werden können.



(a)

Symbol	Parameter	Min.	Max.	Einheit
T_{AD}	Adreßausgabeverzögerung		11	ns
T_{ML}	Adresse ist vor $\overline{MREQ}$ stabil	6		ns
T_M	$\overline{MREQ}$ -Verzögerung von der fallenden Kante von Φ in T_1		8	ns
T_{RL}	RD-Verzögerung von der fallenden Kante von Φ in T_1		8	ns
T_{DS}	Dateneinrichtezeit vor der fallenden Kante von Φ	5		ns
T_{MH}	$\overline{MREQ}$ -Verzögerung von der fallenden Kante von Φ in T_3		8	ns
T_{RH}	$\overline{RD}$-Verzögerung von der fallenden Kante von Φ in T_3		8	ns
T_{DH}	Datenhaltezeit von der Negation von $\overline{RD}$	0		ns

(b)

Abb. 3.37: (a) Lesetaktung bei einem synchronen Bus; (b) Spezifikation wichtiger Zeitgaben

Während der ersten Hälfte von T_3 legt der Speicher 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 negiert die CPU $\overline{MREQ}$ und $\overline{RD}$. Bei Bedarf wird an der nächsten steigenden Flanke des Takts ein weiterer Speicherzyklus gestartet.

In der Taktspezifikation von Abb. 3.37(b) werden acht Symbole, die im Taktdiagramm vorkommen, näher erklärt. T_{AD} ist beispielsweise das Zeitintervall zwischen der steigenden Flanke von T_1 und den zu setzenden Adreßleitungen. Laut Taktspezifikation ist $T_{AD} \leq 11$ ns. Das bedeutet, der CPU-Hersteller gewährleistet, daß die CPU innerhalb eines Lesezyklus die zu lesende Adresse innerhalb von 11 ns des Mittelpunkts an der steigenden Flanke von T_1 ausgibt.

Die Taktspezifikation gibt auch vor, daß die Daten auf den Datenleitungen mindestens T_{DS} (5 ns) vor der fallenden Flanke von T_3 verfügbar sein müssen, um eine ausreichende Stabilisierungszeit bis zum Lesebeginn der CPU sicherzustellen. Die Einschränkungen auf T_{AD} und T_{DS} bedeuten, daß der Speicher im schlechtesten Fall nur 62,5 – 11 – 5 = 46,5 ns ab dem Zeitpunkt zur Verfügung hat, an dem die Adresse erscheint, um die Daten bereitzustellen. Da 40 ns auch im schlechtesten Fall ausreichen, kann ein 40-ns-Speicher immer innerhalb von T_3 reagieren. Bei einem 50-ns-Speicher müßte man allerdings einen zweiten Wartezustand einfügen und innerhalb von T_4 reagieren.

Die Taktspezifikation gewährleistet weiterhin, daß die Adresse mindestens 6 ns vor der Assertion von $\overline{MREQ}$ eingerichtet wird. Diese Zeit kann wichtig sein, falls $\overline{MREQ}$ die Chipauswahl auf dem Speicherchip ansteuert, weil einige Speicher die Einrichtung einer Adresse vor der Chipauswahl verlangen. Selbstverständlich sollte der Systemdesigner keinen Speicherchip wählen, der eine Einrichtzeit von 10 ns benötigt.

Die Einschränkungen auf T_M und T_{RL} bedeuten, daß $\overline{MREQ}$ und $\overline{RD}$ innerhalb von 8 ns am fallenden Takt von T_1 asseriert werden. Im schlechtesten Fall hat der Speicherchip nur 25 + 25 – 8 – 5 = 37 ns nach der Assertion von $\overline{MREQ}$ und $\overline{RD}$, um seine Daten auf den Bus zu bekommen. Diese Einschränkung gilt zusätzlich zum (und unabhängig vom) 40-ns-Intervall, der nach der Stabilisierung der Adresse nötig ist.

T_{MH} und T_{RH} sind die Zeitspanne nach dem Negieren von $\overline{MREQ}$ und $\overline{RD}$ nach Einlesen der Daten. T_{DH} sagt aus, wie lange der Speicher die Daten nach dem Negieren von $\overline{RD}$ auf dem Bus halten muß. Was unsere Beispiel-CPU betrifft, kann der Speicher die Daten vom Bus entfernen, sobald $\overline{RD}$ negiert wurde. Bei einigen neueren CPU-Typen müssen die Daten aber etwas länger stabil gehalten werden.

Wir möchten darauf hinweisen, daß Abb. 3.37 eine stark vereinfachte Darstellung echter Takteinschränkungen ist. In der Praxis wird stets eine große Anzahl viel wichtigerer Zeiten spezifiziert. Dennoch liefert das Beispiel einen guten Überblick über die Funktionsweise eines synchronen Busses.

Steuersignale können auf High oder Low gelegt werden. Die Entscheidung wird dem Busdesigner überlassen. Man kann dies als Hardware-Gegenstück zur Entscheidung eines Programmierers betrachten, ob er freie Plattenblöcke in einer Bitmap als Nullen oder Einsen darstellt.

Asynchrone Busse

Synchrone Busse sind dank ihrer diskreten Zeitintervalle zwar nutzungsfreundlich, bergen aber auch ihre Probleme. Erstens funktioniert alles in integralen Mehrfachen des Bustakts. Sind eine CPU und ein Speicher in der Lage, einen Transfer in 3,1 Zyklen zu bewältigen, müssen sie sich auf 4,0 ausdehnen, weil Teilzyklen verboten sind.

Noch schlimmer: Wurde ein Buszyklus gewählt und Speicher sowie E/A-Karten wurden dafür eingerichtet, lassen sich künftige technische Verbesserungen kaum nutzen. Nehmen wir an, daß ein paar Jahre nach der Entwicklung des Systems von Abb. 3.37 neue Speicher mit Zugriffszeiten von 20 ns statt 40 ns auf den Markt kommen. Für die neuen Speicher bräuchte man keine Wartezustände mehr, und die Maschine wäre wesentlich schneller. Nehmen wir weiter an, daß Speicher mit 10 ns verfügbar werden. In diesem Fall gäbe es keinen Leistungsgewinn, weil die Mindestzeit für eine Leseoperation bei diesem Design zwei Zyklen beträgt.

Formulieren wir diese Tatsache anders: Hat ein Bus eine heterogene Sammlung von Geräten, von denen einige schnell und einige langsam sind, muß er nach dem langsamsten eingerichtet werden, und das schnellste würde natürlich nie voll genutzt werden.

Eine Mischtechnologie läßt sich realisieren, wenn man sich einem asynchronen Bus zuwendet, d.h. einem Bus ohne Mastertakt wie in Abb. 3.38. Hier muß man nicht alles mit dem Taktgeber koppeln. Hat der Busmaster die Adresse, $\overline{MREQ}$, $\overline{RD}$, und ist alles andere asseriert, asseriert er ein spezielles Signal, das wir $\overline{MSYN}$ (Master SYNchronization) nennen. Erkennt der Slave dieses Signal, führt er die Arbeit so schnell wie möglich aus. Anschließend asseriert er $\overline{SSYN}$ (Slave SYNchronization).

Sobald $\overline{SSYN}$ asseriert wurde, weiß der Master, daß die Daten verfügbar sind. Er speichert sie und negiert die Adreßleitungen sowie $\overline{MREQ}$, $\overline{RD}$ und $\overline{MSYN}$. Sieht der Slave die Negation von $\overline{MSYN}$, weiß er, daß der Zyklus beendet ist. Er negiert also $\overline{SSYN}$, und wir sind wieder in der ursprünglichen Lage, in der alle Signale negiert 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 in Abb. 3.38. Die Assertion von $\overline{MSYN}$ verursacht die Assertion der Datenleitungen und veranlaßt den Slave, $\overline{SSYN}$ zu asserieren. Die Assertion von $\overline{SSYN}$ verursacht wiederum, daß die Adreßleitungen sowie $\overline{MREQ}$, $\overline{RD}$ und $\overline{MSYN}$ negiert werden. Schließlich zieht die Negation von $\overline{MSYN}$ die Negation von $\overline{SSYN}$ nach sich, was den Lesevorgang beendet.

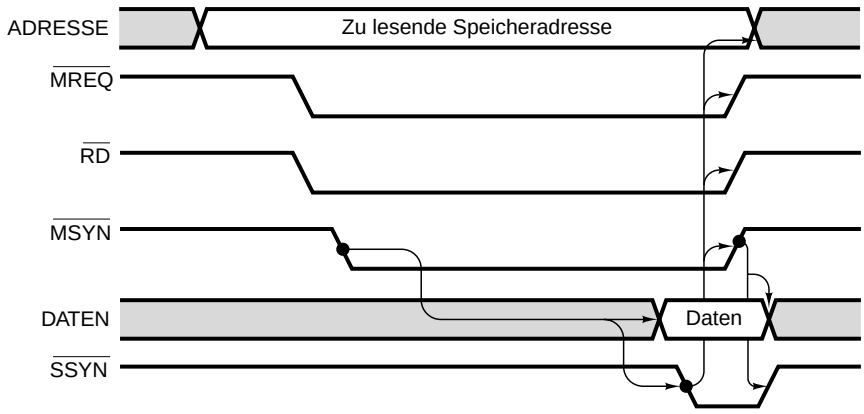


Abb. 3.38: Operation eines synchronen Busses

Eine auf diese Weise verriegelte Signalmenge nennt man ein **volles Handshake**. Der wesentliche Teil besteht aus vier Ereignissen:

1. $\overline{MSYN}$ wird asseriert.
2. $\overline{SSYN}$ wird als Reaktion auf $\overline{MSYN}$ asseriert.
3. $\overline{MSYN}$ wird als Reaktion auf $\overline{SSYN}$ negiert.
4. $\overline{SSYN}$ wird als Reaktion auf die Negation von $\overline{MSYN}$ negiert.

Daraus wird deutlich, daß volle Handshakes zeitunabhängig sind. Jedes Ereignis wird durch ein vorhergehendes Ereignis und nicht durch einen Taktimpuls ausgelöst. Ist ein bestimmtes Master/Slave-Paar langsam, kann sich das nicht auf das nächste, viel schnellere Master/Slave-Paar auswirken.

Der Vorteil eines asynchronen Bus ist nun sicherlich klar. Die meisten Busse sind aber synchron. Das ist darauf zurückzuführen, daß sich synchrone Systeme einfacher entwickeln lassen. Die CPU asseriert einfach nur ihre Signale, und der Speicher reagiert einfach nur. Es gibt keine Rückmeldung (Ursache und Wirkung). Werden die Komponenten richtig ausgewählt, funktioniert alles ohne Handshake. Außerdem wurde im Laufe der Zeit viel Geld in die synchrone Bustechnik investiert.

3.4.5 Bus-Arbitration

Bis jetzt haben wir stillschweigend angenommen, daß es nur einen Busmaster – die CPU – gibt. In Wirklichkeit müssen E/A-Chips als Busmaster auftreten, um vom Speicher zu lesen oder in den Speicher zu schreiben, und um Interrupts auszulösen. Coprozessoren müssen möglicherweise Busmaster sein. Hier stellt sich die Frage: »Was passiert, wenn zwei oder mehr Geräte gleichzeitig die Rolle des Busmasters übernehmen wollen?« Die Antwort ist, daß ein bestimmter Mechanismus, die sogenannte **Bus-Arbitration**, zur Vermeidung von Chaos erforderlich ist.

Arbitrationsmechanismen können zentraler oder dezentraler Natur sein. Betrachten wir zuerst die zentrale Arbitration. Eine bestimmte, einfache Form der zentralen Arbitration ist aus Abb. 3.39(a) ersichtlich. Bei diesem Aufbau bestimmt ein einzelner **Bus-Arbiter**, nach Manier eines Schiedsrichters, wer als nächstes an die Reihe kommt. Bei vielen CPUs ist der Arbiter in den CPU-Chip eingebaut. In anderen Fällen ist aber ein separater Chip erforderlich. Der Bus enthält eine einzelne Anfrageleitung vom Typ Wired-OR, die von einem oder mehreren Geräten asseriert werden kann. Der Arbiter kann nicht erkennen, wie viele Geräte den Bus angefordert haben. Die einzigen Kategorien, die er unterscheiden kann, sind einige oder keine Anfragen.

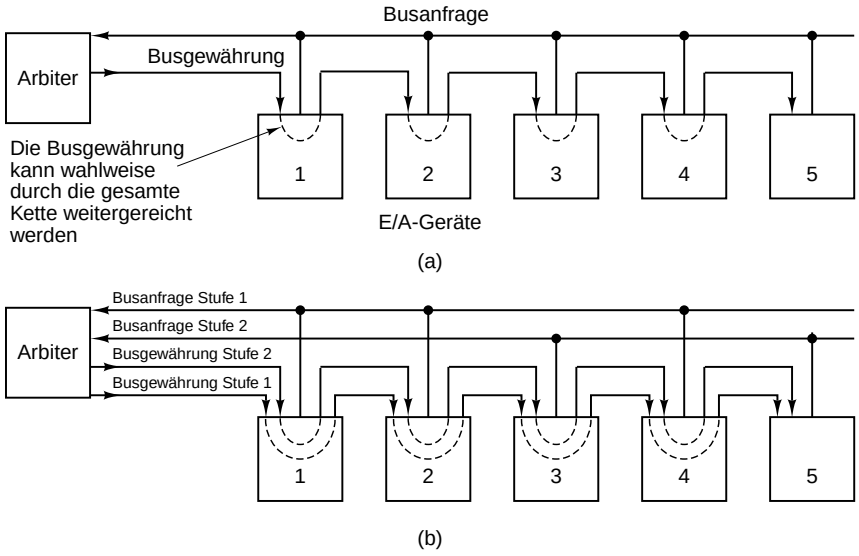


Abb. 3.39: (a) Zentraler Bus-Arbiter mit einer Stufe im Daisy-Chaining; (b) der gleiche Arbiter, jedoch mit zwei Stufen

Erhält der Arbiter eine Busanfrage, asseriert er die Busleitung, um sie bereitzustellen. Diese Leitung ist durch alle E/A-Geräte in Reihe wie ein Satz Elektrokerzen für den Christbaum verbunden. Erkennt das dem Arbiter physisch am nächsten liegende Gerät die Gewährung, prüft es, ob es eine Anfrage gestellt hat. Trifft das zu, übernimmt es den Bus, reicht die Gewährung aber nicht weiter. Hat es keine Anfrage gestellt, reicht es die Gewährung an das nächste Gerät in der Reihe weiter, das sich genauso verhält, und so weiter, bis ein Gerät die Gewährung übernimmt und den Bus beansprucht. Dies nennt man **Daisy-Chaining**. Mit ihm kann man Geräten effektiv Prioritäten entsprechend ihrer Nähe zum Arbiter zuweisen. Das nächstliegende Gerät gewinnt.

Um die impliziten Prioritäten auf der Grundlage der Entfernung vom Arbiter zu umgehen, haben viele Busse mehrere Prioritätsstufen. Für jede Stufe gibt es eine Busanfrage- und eine Busgewährungsleitung. Der Bus in Abb. 3.39(b) hat die beiden Stufen 1 und 2 (echte Busse haben oft 4, 8 oder 16 Stufen).

Jedes Gerät wird an eine der Busanfragestufen angeschlossen. Zeitkritische Geräte werden an die Leitungen mit höherer Priorität angeschlossen. In Abb. 3.39(b) haben die Geräte 1 und 2 die Priorität 1 und die Geräte 3, 4 und 5 die Priorität 2.

Werden mehrere Prioritätsstufen gleichzeitig angefordert, gibt der Arbiter nur der mit der höheren Priorität eine Gewährung aus. Unter den Geräten der gleichen Priorität kommt das Daisy-Chaining zum Einsatz. In Abb. 3.39(b) schlägt Gerät 2 Gerät 4, das seinerseits im Konfliktfall 3 schlägt. Gerät 5 hat die niedrigste Priorität, weil es am Ende der Kette mit der niedrigsten Priorität liegt.

Aus technischer Sicht ist es im übrigen nicht notwendig, die Busgewährungsleitung der Stufe 2 seriell durch die Geräte 1 und 2 zu verbinden, weil sie auf dieser Leitung keine Anfragen stellen können. Die Implementierung gestaltet sich aber einfacher, wenn alle Gewährungsleitungen durch alle Geräte statt Geräte getrennt nach Priorität verbunden werden.

Einige Arbiter verfügen über eine dritte Leitung, die ein Gerät asseriert, wenn es eine Gewährung erhalten und den Bus übernommen hat. Sobald es diese Bestätigungsleitung asseriert hat, können die Anfrage- und Gewährungsleitungen negiert werden. Dies hat zur Folge, daß andere Geräte den Bus anfordern können, während das erste Gerät den Bus noch nutzt. Bis der laufende Transfer abgeschlossen ist, wurde der nächste Busmaster bereits ausgewählt. Er kann unmittelbar im Anschluß an die Negation der Bestätigungsleitung anfangen. Zu diesem Zeitpunkt kann die nächste Arbitrationsrunde beginnen. Dieses Vorgehen erfordert eine zusätzliche Busleitung und mehr Logik in jedem Gerät, verbessert aber die Nutzung der Buszyklen.

Bei Systemen, bei denen sich der Speicher auf dem Hauptbus befindet, muß sich auch die CPU mit allen E/A-Geräten in fast jedem Zyklus um den Bus bemühen. Normalerweise erhält sie aber die niedrigste Priorität, kann also nur auf den Bus zugreifen, wenn ihn kein anderer haben will. Dies basiert auf dem Konzept, daß die CPU immer warten kann, die E/A-Geräte aber häufig schnell auf den Bus zugreifen oder ankommende Daten weitersenden müssen. Platten, die sich in hoher Geschwindigkeit drehen, kann man nicht warten lassen. Dieses Problem wird bei vielen modernen Computer-Systemen dadurch vermieden, daß man den Speicher auf einem von den E/A-Geräten getrennten Bus anordnet, so daß sich nicht alle um den Bus bemühen müssen.

Dezentrale Bus-Arbitration ist aber auch möglich. Ein Computer kann z.B. über 16 mit Prioritäten belegte Busanfrageleitungen verfügen. Möchte ein Gerät den Bus benutzen, asseriert es seine Anfrageleitung. Alle Geräte überwachen alle Anfrageleitungen. Am Ende eines jeden Buszyklus weiß also jedes Gerät, ob es die Anfrage mit der höchsten Priorität gestellt hat und damit die Erlaubnis zur Busnutzung im nächsten Zyklus erhält. Im Vergleich zur zentralen Arbitration erfordert diese Methode mehr Busleitungen, vermeidet aber die potentiellen Kosten des Arbiters. Außerdem wird die Anzahl der Geräte auf die Anzahl der Anfrageleitungen begrenzt.

Eine weitere Variante der dezentralen Bus-Arbitration ist in Abb. 3.40 dargestellt. Hier werden nur drei Leitungen benutzt, ungeachtet dessen, wie viele Geräte vorhanden sind. Die erste Busleitung ist eine Wired-OR-Leitung für die Anfrage um den Bus. Die zweite Busleitung heißt BUSY und wird vom momentanen Busmaster asseriert. Die dritte Leitung wird zur Vergabe des Bus benutzt. Sie ist im Daisy-Chaining angeordnet und führt durch alle Geräte. Der Kopf dieser Kette wird asseriert gehalten, wenn man ihn mit der 5-Volt-Stromspannung verbindet.

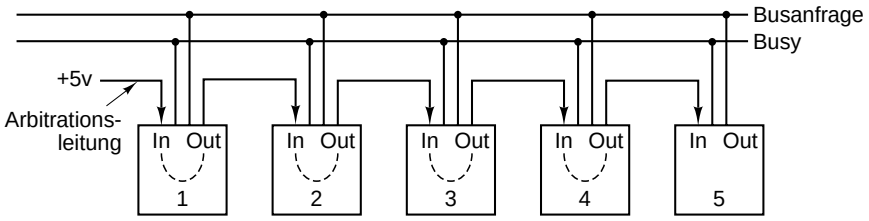


Abb. 3.40: Dezentrale Bus-Arbitration

Wenn kein Gerät den Bus beansprucht, wird die asserierte Arbitrationsleitung an alle Geräte weitergegeben. Zur Busübernahme prüft ein Gerät zuerst, ob der Bus frei ist. Dann wird das Signal IN, das es erhält, asseriert. Wird IN negiert, kann das Gerät nicht als Busmaster auftreten und negiert OUT. Wird IN asseriert, negiert das Gerät OUT, wodurch der nächste Nachbar in der Reihe erkennt, daß IN negiert wurde, so daß er sein OUT negiert. Alle weiteren Geräte in der Kette sehen also, daß IN negiert wurde, und negieren dementsprechend OUT. Hat sich der Staub gelegt, hat nur ein Gerät IN asseriert und OUT negiert. Dieses Gerät nimmt die Rolle des Busmasters ein, asseriert BUSY und OUT und beginnt mit dem Transfer.

Das Gerät ganz links hat nie Schwierigkeiten, den Bus zu bekommen. Dieses Schema ist also mit der ursprünglichen Daisy-Chain-Arbitration vergleichbar, außer daß es keinen Arbiter gibt. Daher ist es billiger, schneller und keinen Arbiter-Ausfällen ausgesetzt.

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 werden in diesem Abschnitt beschrieben.

Normalerweise wird je ein Wort übertragen. Kommt Caching zum Einsatz, ist es aber wünschenswert, eine ganze Cache-Zeile (d.h. 16 aufeinanderfolgende 32-Bit-Wörter) auf einmal einzulesen. Vielfach können Blocktransfers effizienter als aufeinanderfolgende einzelne Transfers ausgelegt werden. Beginnt eine Blockleseoperation, teilt der Busmaster dem Slave die Anzahl der zu übertra-

genden Wörter mit. Das kann er beispielsweise durch Einfügen eines Wortzählers in die Datenleitungen innerhalb von T_1. Statt nur ein Wort zurückzugeben, gibt der Slave in jedem Zyklus ein Wort aus, bis der Zähler abgelaufen ist. Abb. 3.41 zeigt eine modifizierte Version von Abb. 3.37(a), hier aber mit dem zusätzlichen Signal $\overline{BLOCK}$. Dieses Signal wird asseriert, um einen Blocktransfer anzufordern. Bei diesem Beispiel dauert eine Blockleseoperation von vier Wörtern sechs statt zwölf Zyklen.

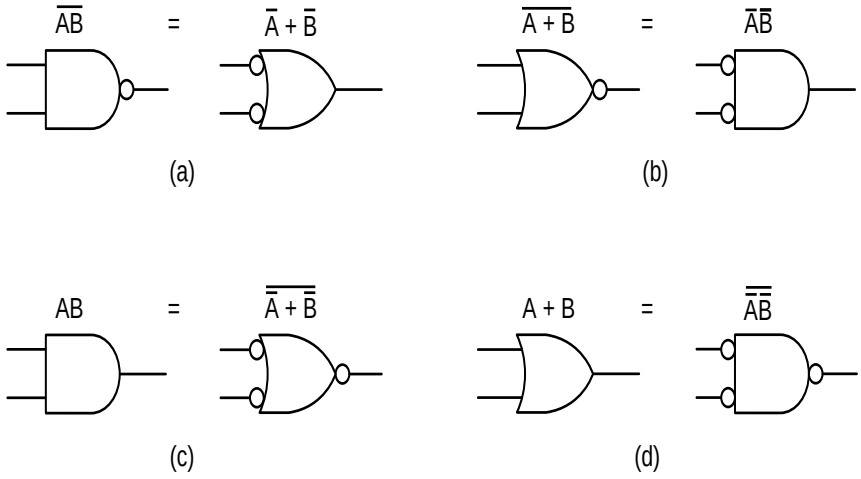


Abb. 3.41: Ein Blocktransfer

Daneben gibt es noch weitere Arten von Buszyklen. Beispielsweise muß man bei einem Mehrprozessorsystem mit zwei oder mehr CPUs am gleichen Bus oft sicherstellen, daß jeweils nur eine CPU auf eine kritische Datenstruktur im Speicher zugreift. Häufig wird diese Anordnung realisiert durch die Verwendung einer Variablen im Speicher, die 0 ist, wenn keine CPU die Datenstruktur benutzt, und andernfalls 1 beträgt. Möchte eine CPU auf die Datenstruktur zugreifen, muß sie die Variable lesen, und sie auf 1 setzen, falls sie 0 ist. Hat man Pech, lesen zwei CPUs die Variable in aufeinanderfolgenden Buszyklen, wenn beide erkennen, daß die Variable auf 0 steht, und beide sie auf 1 in der Annahme setzen, daß sie jeweils die einzige mit diesem Bemühen ist. Chaos ist die Folge dieser Ereignisse.

Um eine solche Situation zu verhindern, sind Mehrprozessorsysteme oft mit einem speziellen Buszyklus (Read-Write-Modify) ausgestattet. Dies gewährleistet, daß nur eine CPU ein Wort aus dem Speicher lesen, es prüfen, ändern und dann wieder zurückschreiben kann, ohne bei diesem Ablauf den Bus freizugeben. Diese Zyklusart hindert zwei konkurrierende CPUs, den Bus zu nutzen und sich gegenseitig zu stören.

Als weitere wichtige Methode gibt es einen Buszyklus für den Umgang mit Interrupts. Befiehlt die CPU einem E/A-Gerät eine Aktion, erwartet sie norma-

lerweise einen Interrupt nach getaner Arbeit. Die Signalisierung des Interrupts erfordert den Bus.

Nun kann es vorkommen, daß mehrere Geräte gleichzeitig einen Interrupt veranlassen. Dabei stellen sich die gleichen Arbitrationsprobleme wie bei gewöhnlichen Buszyklen. Als übliche Lösung werden den Geräten Prioritäten zugewiesen und ein zentraler Arbiter benutzt, um den zeitkritischen 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 Abb. 3.42).

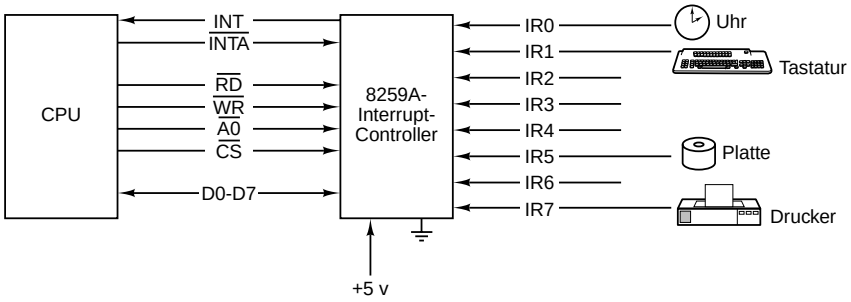


Abb. 3.42: Benutzung des Interrupt-Controllers 8259A

Bis zu acht E/A-Controller-Chips können direkt an die acht IRx-Eingänge (Interrupt Request) des 8259A angeschlossen werden. Möchte eines dieser Geräte einen Interrupt veranlassen, asseriert es seine Eingangsleitung. Im Fall der Assertion eines oder mehrerer Eingänge asseriert der 8259A INT (INTerrupt), wodurch der Interrupt-Pin auf der CPU direkt angesteuert wird. Kann die CPU mit dem Interrupt umgehen, schickt sie einen Impuls auf INTA (INTerrupt Acknowledge) an den 8259A zurück. An diesem Punkt muß der 8259A festlegen, welcher Eingang den Interrupt veranlaßt hat, indem er die Nummer dieses Eingangs auf den Datenbus ausgibt. Diese Operation erfordert einen gesonderten Buszyklus. Die CPU-Hardware benutzt diese Nummer dann als Index in einer Zeigertabelle (Pointer Table) namens **Interrupt Vectors**, um die Adresse der auszuführenden Prozedur herauszufinden, mit der der Interrupt bedient wird.

Im 8259A befinden sich mehrere Register, die von der CPU gelesen und beschrieben werden können. Hierfür benutzt die CPU gewöhnliche Buszyklen und die Pins $\overline{RD}$ (ReaD), $\overline{WR}$ (WRite), $\overline{CS}$ (Chip Select) und $\overline{AO}$. Hat die Software den Interrupt abgearbeitet und ist sie bereit, das nächste entgegenzunehmen, wird ein spezieller Code in eines der Register geschrieben. Im Anschluß daran negiert der 8259A INT, sofern er keine weiteren Interrupts anstehen hat. Diese Register können auch beschrieben werden, um den 8259A in einen von mehreren Modi zu versetzen, eine Reihe von Interrupts zu maskieren und andere Merkmale zu ermöglichen.

Sind mehr als acht E/A-Geräte vorhanden, kann der 8259A kaskadiert werden. Im Extremfall können alle acht Eingänge mit den Ausgängen von weiteren acht 8259A-Chips verbunden werden, so daß bis zu 64 E/A-Geräte in einem zweiphasigen Interrupt-Netz möglich sind. Der 8259A verfügt über ein paar Pins für die Handhabung dieser Kaskadierung, die wir der Einfachheit halber ausgelassen haben.

Damit ist das Thema »Busdesign« keinesfalls erschöpft. Die obigen Erläuterungen bieten aber eine ausreichende Grundlage für das Verständnis der wesentlichen Aspekte der Funktionsweise von Bussen und der Interaktion von CPUs und Bussen. Wir gehen vom Allgemeinen zum Speziellen über. In den nächsten Abschnitten werden einige Beispiele echter CPUs und ihrer Busse beschrieben.

3.5 Beispiele von CPU-Chips

In diesem Abschnitt werden die CPU-Chips Pentium II, UltraSPARC II und picoJava auf Hardware-Ebene beschrieben.

3.5.1 Pentium II

Der Pentium II ist der direkte Nachfolger der ursprünglich im IBM-PC benutzten 8088-CPU. Obwohl der Pentium II mit seinen 7,5 Millionen Transistoren weit von den 29.000 Transistoren des 8088 entfernt ist, 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 Sicht der Software ist der Pentium II eine volle 32-Bit-Maschine. Er hat den gleichen ISA wie 80386, 80486, Pentium und Pentium Pro, sowie die gleichen Register, gleichen Instruktionen und eine volle Implementierung des IEEE-Standards 754 für Gleitkomma auf dem Chip.

Aus Sicht der Hardware ist der Pentium II etwas mehr, weil er einen physischen Speicher von 64 Gbyte adressieren und Daten in Einheiten von 64 Bit mit dem Speicher austauschen kann. Der Programmierer kann diese 64-Bit-Transfers zwar nicht feststellen; sie machen die Maschine aber schneller als eine reine 32-Bit-Maschine.

Intern auf der Mikroarchitekturebene ist der Pentium II im Grunde ein Pentium Pro mit MMX-Instruktionen. Instruktionen der ISA-Ebene werden vom Speicher reichlich im voraus eingelesen und in RISC-ähnliche Mikrooperationen aufgeteilt. Diese Mikrooperationen werden in einem Puffer gespeichert. Sobald eine davon die zur Ausführung erforderlichen Ressourcen zur Verfügung hat, kann sie gestartet werden. Mehrere Mikrooperationen kann man im gleichen Zyklus starten, was den Pentium II zu einer superskalaren Maschine macht.

Der Pentium II hat ein zweistufiges Cache-System. Auf dem Chip ist ein Cache-Paar integriert – 16 Kbyte für Instruktionen und 16 Kbyte für Daten. Ferner gibt es einen einheitlichen Cache von 512 Kbyte auf der zweiten Ebene. Die Cache-Leitung ist 32 Byte groß. Der Cache der zweiten Ebene läuft mit der halben Taktfrequenz der CPU. CPU-Takte sind von 233 MHz aufwärts erhältlich.

Im Pentium II werden zwei primäre externe, synchrone Busse benutzt. Der Speicherbus wird benutzt, um auf den Haupt-DRAM zuzugreifen, und der PCI-Bus, um mit den E/A-Geräten zu kommunizieren. Zuweilen wird ein alter Bus mit dem PCI-Bus verbunden, damit ältere Peripheriegeräte angeschlossen werden können.

Ein Pentium II kann mit einer oder zwei CPUs ausgestattet sein, die sich einen Speicher teilen. Bei einem System mit zwei CPUs besteht folgende Gefahr: Wird ein Wort in einen Cache gelesen und dort modifiziert, dann aber nicht in den Speicher zurückgeschrieben, erhält die andere CPU beim Einlesen des Worts einen falschen Wert. Durch eine spezielle Unterstützung (Snooping) wird dieses Problem umgangen.

Ein deutlicher Unterschied zwischen dem Pentium II und allen seinen Vorgängern zeigt sich in der Zusammensetzung der Baueinheit. Vom 8088 bis zum Pentium Pro waren alle Intel-CPUs normale Chips, seitlich oder unten mit Pins, die man in Buchsen einstecken konnte. Im Gegensatz dazu weist der Pentium II etwas auf, was Intel **SEC** (**Single Edge Cartridge**) nennt. Abb. 3.43 zeigt, daß ein SEC eine relativ große Kunststoffbox ist, in der sich die CPU, der Level-2-Cache und ein Edge-Connector für den Signalexport befinden. Der Pentium II SEC hat 242 Connectors.

Während Intel zweifellos gute Gründe hatte, sich für diese Baueinheit zu entscheiden, birgt das Modell ein Problem in einer Dimension, die Intel nicht vorhersehen konnte. Anscheinend haben viele Kunden die Gewohnheit, ihre Computer aufzuschrauben und sich den CPU-Chip anzusehen. Die ersten Auslieferungen des Pentium II brachten zu Tage, daß die Kunden die CPU nicht finden konnten. Es hagelte Beschwerden. Intel löste dieses Problem durch Aufkleben eines Bilds (eigentlich eines Hologramms) des CPU-Chips auf der Vorderseite aller SEC-Boxen.

Das Power-Management ist beim Pentium II ein zentrales Thema. Die abgegebene Hitze hängt von der Taktfrequenz ab, liegt aber im Bereich von 30 bis 50 Watt. Das ist für einen Computer-Chip eine enorme Menge. Um eine Vorstellung davon zu bekommen, wie sich 50 Watt anfühlen, halte man seine Hand über (aber nicht auf!) eine schon längere Zeit eingeschaltete 50-Watt-Glühbirne. Folglich ist die SEC-Box mit einem Kühlsystem ausgestattet, das die erzeugte Hitze verteilt. Diese Eigenschaft bedeutet, daß ein Pentium II als Campingstövchen benutzt werden kann, wenn er als CPU ausgedient hat.

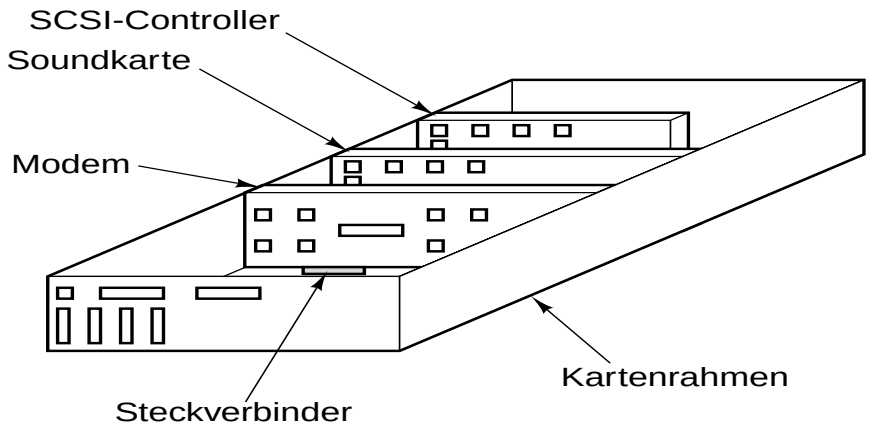


Abb. 3.43: Die SEC-Box des Pentium II

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 unerwünscht. Um dieses Problem zu lösen, hat Intel die CPU so entworfen, daß sie schlummert, wenn sie nichts tut, und in Tiefschlaf verfällt, wenn es eine Zeitlang nichts zu tun gibt. Befindet sie sich im Zustand des Tiefschlafs, werden die Cache- und Registerwerte beibehalten, die Uhr und alle internen Einheiten werden hingegen abgeschaltet. Es ist nicht bekannt, ob ein Pentium II träumt, wenn er sich im Tiefschlaf befindet.

Das logische Pinout des Pentium II

Die 242 Kantenverbinder der SEC werden für 170 Signale, 27 Stromanschlüsse (in unterschiedlichen Spannungen), 35 Masseleitungen und 10 Reserven für künftige Verwendungen benutzt. Ein Teil der logischen Signale belegt zwei oder mehr Pins (z.B. zur Anforderung der Speicheradresse), so daß es nur 53 verschiedene Pins gibt. Abb. 3.44 stellt das logische Pinout vereinfacht dar. Links in der Abbildung befinden sich die sechs wichtigen Gruppen der Speicherbussignale. Auf der rechten Seite sind verschiedene Signale. Die Bezeichnungen (alle in Großbuchstaben) entsprechen den Intel-Signalnamen. Die in Groß- und Kleinschreibung sind Sammelbezeichnungen für mehrere zusammenhängende Signale.

Intel benutzt eine eigene Namenskonvention, die man unbedingt nachvollziehen sollte. Da alle Chips für die Verwendung in Computern unserer Tage entwickelt werden, muß man Signalnamen als ASCII-Text darstellen. Die Verwendung von Überstrichen für Signale, die auf Low gelegt werden, ist zu schwierig. Deshalb hängt Intel dem Namen ein # (Raute oder Schweinegitter) an, $\overline{BPRI}$ wird also BPRI# geschrieben. Wir sehen in der Abbildung, daß die meisten Signale des Pentium II auf Low asseriert sind.

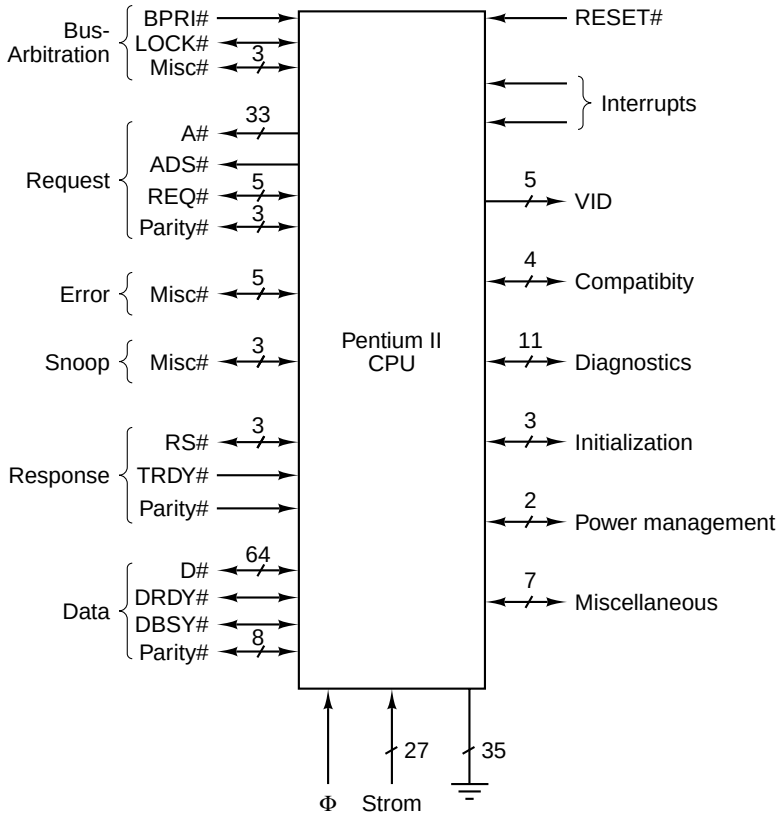


Abb. 3.44: Logisches Pinout des Pentium II. Die großgeschriebenen Namen sind die offiziellen Intel-Bezeichnungen für die verschiedenen Signale. Namen in Groß- und Kleinschreibung sind Gruppen zusammenhängender Signale.

Nun betrachten wir die Signale näher. Wir beginnen mit den Bussignalen. Die erste Signalgruppe wird zur Busanforderung benutzt (d.h. für die Bus-Arbitration). BPRI# erlaubt es einem Gerät, eine Anforderung mit hoher Priorität auszugeben, die Vorrang vor der normalen Priorität hat. Mit LOCK# kann eine CPU den Bus sperren, um andere am Zugriff zu hindern, bis sie fertig ist.

Hat die CPU oder ein anderer Busmaster die Businhaberschaft an sich genommen, kann sie mit Hilfe der nächsten Signalgruppe eine Busanfrage stellen. Die Adressen sind 36 Bit breit, die drei niederwertigen Bits müssen aber immer 0 sein; deshalb sind ihnen keine Pins zugewiesen. A# hat also nur 33 Pins. Alle Transfers sind 8 Bytes und in einer 8-Byte-Grenze angeordnet. Mit 36 Adreßbits beträgt der maximal adressierbare Speicher 2^{36}, was 64 Gbyte entspricht.

Wird eine Adresse auf einen Bus gelegt, wird das Signal ADS# asseriert, um dem Ziel (z.B. dem Speicher) mitzuteilen, daß die Adreßleitungen gültig sind. Die Art des Buszyklus (z.B. lies ein Wort oder schreibe einen Block) geht an

die Leitungen REQ#. Zwei der Paritätssignale schützen A#, und eines schützt ADS# und REQ#. Die fünf Fehlerleitungen werden vom Slave zur Vermeidung von Paritätsfehlern und allen Geräten benutzt, um bestimmte andere Fehler zu melden.

Die `Snooping`-Gruppe wird in Mehrprozessorsystemen gebraucht, damit eine CPU herausfinden kann, ob sich das benötigte Wort im Cache der anderen CPU befindet. Wie der Pentium II das Snooping bewältigt, wird in Kapitel 8 beschrieben.

Die Gruppe `Response` enthält Signale für den Slave, um Rückmeldungen an den Master zu schicken. RS# enthält den Statuscode. TRDY# zeigt an, daß der Slave (das Ziel) bereit ist, Daten vom Master entgegenzunehmen. Diese Signale werden einer Paritätsprüfung unterzogen.

Die letzte Busgruppe dient dem eigentlichen Datentransfer. D# braucht man, um 8 Datenbytes auf den Bus zu legen. Wenn sie sich dort befinden, wird DRDY# asseriert, um ihre Anwesenheit anzukündigen. DBSY# wird benutzt, um der Welt mitzuteilen, daß der Bus momentan beschäftigt ist.

RESET# setzt die CPU in dem Fall zurück, daß etwas Schlimmes passiert ist. Der Pentium II läßt sich so konfigurieren, daß die Interrupts auf die gleiche Weise wie beim 8088 (zum Zwecke der Abwärtskompatibilität) benutzt werden. Er kann aber auch für eine anspruchsvollere Interrupt-Signalisierung konfiguriert werden. Dafür gibt es ein Gerät namens **APIC** (**Advanced Programmable Interrupt Controller**).

Der Pentium II unterstützt mehrere Spannungen. Man muß ihm aber sagen, welche er benutzen soll. Über die VID-Signale kann man die Stromversorgungsspannung automatisch wählen. Die Signale `Compatibility` werden benutzt, um ältere Geräte am Bus glauben zu machen, sie seien an einen 8088 angeschlossen. Die Gruppe `Diagnostics` enthält Signale für Test- und Diagnosesysteme gemäß IEEE-Standard 1149.1 für JTAG-Tests. Die Gruppe `Initialization` befaßt sich mit dem Starten des Systems. Die Gruppe `Power Management` erlaubt es, die CPU schlafen zu legen oder sie in Tiefschlaf zu versetzen. Die Gruppe `Miscellaneous` ist eine Ansammlung aus Signalen, einschließlich eines Signals, das von der CPU asseriert wird, wenn die Temperatur der CPU 130 °C erreicht. Sollte eine CPU jemals diese Temperatur erreichen, denkt sie wahrscheinlich an ihre Pensionierung und träumt von einem Leben als Campingstövchen.

Pipelining im Speicherbus des Pentium II

Moderne CPUs wie der Pentium II sind viel schneller als moderne DRAM-Speicher. Um die CPU mangels Daten vor dem Hungertod zu erretten, muß man aus dem Speicher den höchstmöglichen Durchsatz herausholen. Aus diesem Grund wird auf den Speicherbus des Pentium II ein hohes Maß an Pipelining angewandt, wobei nicht weniger als acht Bustransaktionen gleichzeitig ablaufen. Das Pipelining-Konzept wurde in Kapitel 2 in Zusammenhang mit

einer Pipeline-CPU (siehe Abb. 2.4) vorgestellt, es kann aber auch auf Speicher angewandt werden.

Um Pipelining zu ermöglichen, umfassen Speicheranforderungen, die beim Pentium II **Transaktionen** heißen, sechs Phasen:

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

Nicht alle Phasen sind bei allen Transaktionen nötig. Die Phase `Bus Arbitration` bestimmt, welcher der potentiellen Busmaster als nächstes an der Reihe ist. In der Phase `Req` kann die Adresse auf den Bus gelegt und die Anfrage gestellt werden. Die Phase `Error` erlaubt dem Slave die Ankündigung, daß die Adresse einen Paritätsfehler hatte oder sonst etwas falsch ist. Die Phase `Snoop` gestattet es einer CPU, die andere auszuschnüffeln, was eigentlich nur in einem Mehrprozessorsystem nötig ist. In der Phase `Resp` erfährt der Master, ob er die angeforderten Daten erhält. In der Phase `Data` können schließlich die Daten zurückgesendet werden.

Das Geheimnis des Pipeline-Speicherbusses des Pentium II ist, daß jede Phase unterschiedliche Bussignale nutzt, so daß sie alle voneinander unabhängig sind. Die sechs benötigten Signalgruppen sind in Abb. 3.44 links dargestellt. Eine CPU kann beispielsweise versuchen, mit Hilfe der Arbitrationssignale Zugriff auf den Bus zu erhalten. Hat sie dieses Recht erhalten, gibt sie diese Busleitungen frei und beginnt, sich der Leitungen der Anfragegruppe zu bedienen. Unterdessen kann die andere CPU oder ein E/A-Gerät in die Bus-Arbitrationsphase eintreten usw. Abb. 3.45 zeigt mehrere, gleichzeitig anstehende Bustransaktionen.

Die Bus-Arbitrationsphase ist in Abb. 3.45 nicht dargestellt, weil sie nicht immer gebraucht wird. Möchte der momentane Businhaber (oft die CPU) beispielsweise eine andere Transaktion ausführen, braucht sie den Bus nicht erneut anzufordern. Sie muß lediglich wieder nach dem Bus verlangen, nachdem sie die Businhaberschaft an ein anderes Gerät abgegeben hat. Die Transaktionen 1 und 2 sind klar nachvollziehbar: 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 eingebracht hat. Als Folge kann Transaktion 4 ihre Datenphase nicht starten, wann sie will. Sie beobachtet, daß das DBSY#-Signal immer noch asseriert ist, und wartet, bis es negiert wird. An Transaktion 5 sehen wir, daß die Antwortphase auch mehrere Buszyklen beanspruchen kann, so daß sich Transaktion 6 verzögert. In Transaktion 7 stellen wir fest, daß eine einmal in

die Pipeline eingefügte Blase dort bleibt, falls weitere Transaktionen hintereinander starten. In der Praxis ist es unwahrscheinlich, daß eine neue Transaktion in jedem Buszyklus starten kann; deshalb überdauern Blasen nicht so lange.

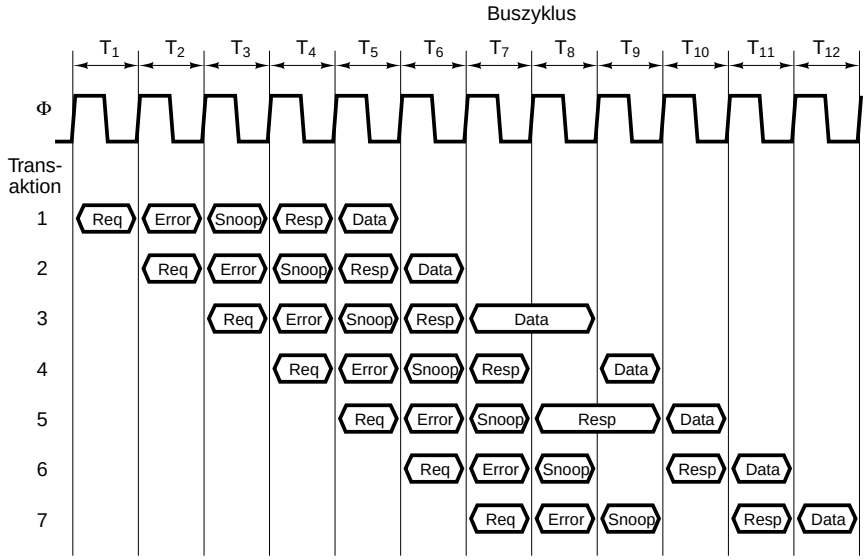


Abb. 3.45: Pipelining-Anfragen auf den Speicherbus des Pentium II

3.5.2 UltraSPARC II

Als zweites Beispiel eines CPU-Chips werden wir nun die UltraSPARC-Familie von Sun untersuchen. Dabei handelt es sich um Suns Linie der 64-Bit-SPARC-CPUs. Diese CPUs entsprechen uneingeschränkt der SPARC-Architektur Version 9, die auch 64-Bit-CPUs aufweisen. Sie werden in Sun-Workstations und -Servern sowie verschiedenen anderen Anwendungen benutzt. Zu dieser Familie zählen UltraSPARC I, UltraSPARC II und UltraSPARC III, die sich von der Architektur her sehr ähneln, aber durch Einführungsdatum und Taktgeschwindigkeit unterscheiden. Wir beziehen uns im weiteren Verlauf auf UltraSPARC II, meinen aber alle drei Modelle.

UltraSPARC II ist eine herkömmliche RISC-Maschine und voll binär kompatibel mit der 32-Bit-SPARC-Architektur V8. Sie kann die für 32-Bit-SPARC V8 geschriebenen Binärprogramme ohne Modifikation ausführen, weil die SPARC-Architektur V9 mit der SPARC-Architektur V9 abwärtskompatibel ist. Der einzige Punkt, in dem sich die beiden Architekturen unterscheiden, ist die Erweiterung von V9 um die Multimedia-Instruktionen VIS für Grafikanwendungen, MPEG-Echtzeit-Dekodierung usw.

UltraSPARC II wurde eigens dafür entwickelt, Vier-Knoten-Mehrprozessoren mit gemeinsamer Speichernutzung ohne Verwendung einer externen Schalttechnik und größeren Mehrprozessoren mit einem Minimum an externer Schalttechnik zu realisieren. Anders ausgedrückt: Ein Großteil des zum Bau eines Mehrprozessors nötigen »Klebstoffs« ist auf dem UltraSPARC-II-Chip enthalten.

Im Gegensatz zur SEC-Box des Pentium II ist die UltraSPARC-II-CPU ein – allerdings großer – Einzelchip mit 5,4 Millionen Transistoren. Er hat 787 Pins an der Unterseite; die Anordnung ist aus Abb. 3.46 ersichtlich. Diese große Pinzahl ist hauptsächlich der Nutzung von 64 Bits für Adressen und 128 Bits für Daten zuzuschreiben, teilweise aber auch auf die Arbeitsweise des Caching zurückzuführen. Viele Pins sind außerdem unbenutzt oder redundant. Die Zahl 787 wurde gewählt, um eine Zusammensetzung nach dem Industriestandard zu ermöglichen. Die Industrie glaubt scheinbar Glück zu haben, wenn sie über eine Primzahl von Pins verfügt.

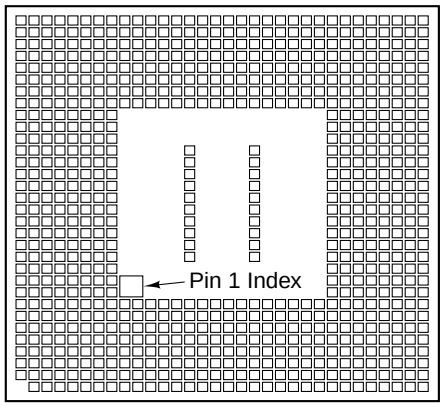


Abb. 3.46: Der CPU-Chip der UltraSPARC II

UltraSPARC II hat zwei interne Caches: 16 Kbyte für Instruktionen und 16 Kbyte für Daten. Wie der Pentium II nutzt sie ebenfalls einen Level-2-Cache außerhalb des Chips. Im Gegensatz zum Pentium II ist bei der UltraSPARC II der Level-2-Cache aber nicht in eine herstellerspezifische Box gepackt. Vielmehr wird es Systemdesignern freigestellt, im Handel erhältliche Cache-Chips als Level-2-Cache zu wählen.

Die Entscheidung, den Level-2-Cache beim Pentium II zu integrieren und bei der UltraSPARC abzutrennen, wurde teilweise auf der Grundlage technischer Optionen und teilweise aufgrund der unterschiedlichen Geschäftsmodelle von Intel und Sun getroffen. In technischer Hinsicht ist ein externer Cache flexibler (ein Cache der UltraSPARC II kann von 512 Kbyte bis 16 Mbyte reichen; der Cache des Pentium II ist auf 512 Kbyte festgesetzt). Er kann aufgrund seiner größeren Entfernung von der CPU aber auch langsamer sein. Außer-

dem sind mehr sichtbare Signale nötig, um den Cache zu adressieren (die 242 Kantenverbinder der SEC-Box des Pentium II beinhalten keine Signale für den Cache, da die CPU/Cache-Interaktion im Innern der Box abläuft). Die 787 Pins der UltraSPARC II beinhalten aber eine Cache-Steuerung.

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 Computer-Anbieter, kein Chiphersteller. Das Unternehmen entwikkelt einen Teil seiner eigenen Chips (z.B. die UltraSPARC), läßt aber von Halbleiterherstellern, wie Texas Instruments und Fujitsu, fertigen. Soweit möglich, bevorzugt Sun die Verwendung kommerziell erhältlicher Chips, die sich im Wettbewerbsumfeld bewährt haben. Die für Level-2-Cache benutzten SRAMs sind von zahlreichen Chipanbietern beziehbar, so daß für Sun keine Notwendigkeit einer Eigenentwicklung bestand. Diese Entscheidung impliziert, daß man den Level-2-Cache vom CPU-Chip unabhängig macht.

Die meisten Sun-Workstations sind mit dem **SBus**, einem synchronen 25-MHz-Bus, ausgestattet. In diesen Bus können E/A-Geräte eingesteckt werden. Für einen Speicher ist der SBus aber zu langsam, so daß Sun einen anderen Mechanismus geschaffen hat, damit (mehrere) UltraSPARC-CPUs mit (mehreren) Speichern kommunizieren können. Dieser Mechanismus heißt **UPA** (**Ultra Port Architecture**) und kann als Bus, Schalter (Switch) oder beides implementiert werden. In den verschiedenen Workstation- und Servermodellen geschieht dies auch. Die UPA-Implementierung spielt für die CPU aber keine Rolle, weil die Schnittstelle zur UPA genau definiert ist. Das ist auch die Schnittstelle, die der CPU-Chip unterstützen muß.

In Abb. 3.47 sehen wir den Kern einer UltraSPARC II mit CPU-Chip, UPA-Schnittstelle und Level-2-Cache (zwei handelsübliche SRAMs). Man sieht außerdem einen **UDB II** (**UltraSPARC Data Buffer II**). Die Funktion dieses Chips wird später erklärt. Benötigt die CPU ein Speicherwort, sucht sie zuerst in einem der (internen) Level-2-Cache. Findet sie das Wort, fährt sie in voller Geschwindigkeit mit der Ausführung fort. Findet sie das Wort nicht im ersten (Level 1) Cache, sucht sie im zweiten (Level 2).

Caching wird ausführlich in Kapitel 4 behandelt. Hier nur ein paar nützliche Worte darüber. Der Hauptspeicher ist in Cache-Leitungen (Blöcke) von 64 Byte unterteilt. Die 256 am häufigsten benutzten Instruktionsleitungen und die 256 am häufigsten benutzten Datenleitungen befinden sich im Level-1-Cache. Cache-Leitungen, die häufig benutzt werden, aber nicht in den Level-1-Cache passen, werden in den Level-2-Cache gestellt. Dieser Cache enthält eine zufällige Mischung aus Daten- und Instruktionsleitungen. Sie werden in dem Rechteck mit der Beschriftung »Daten im Level-2-Cache« gespeichert. Das System muß überblicken, welche Leitungen sich im Level-2-Cache befinden. Diese Information wird in einem zweiten SRAM (beschriftet mit »Tags im Level-2-Cache«) vorgehalten.

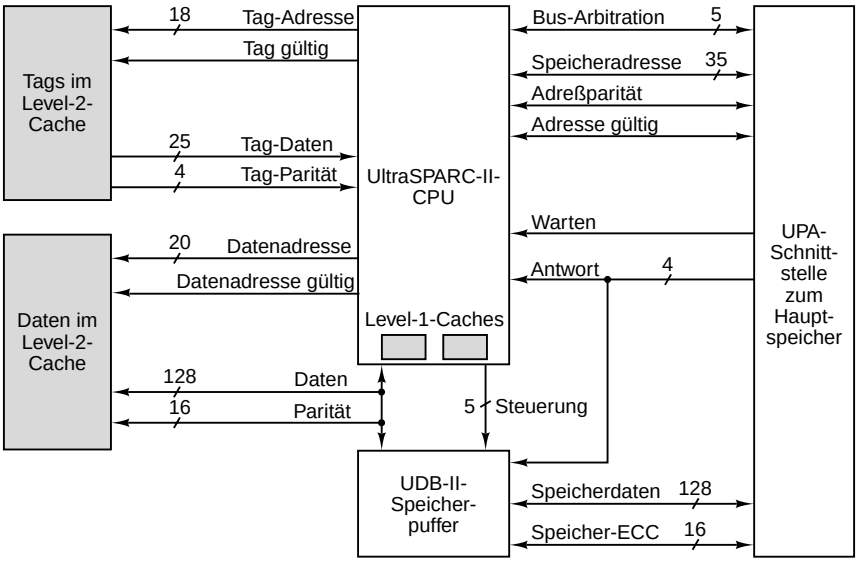


Abb. 3.47: Hauptmerkmale des Kerns einer UltraSPARC II

Schlägt eine Suche im Level-1-Cache fehl, schickt die CPU den Bezeichner (Identifier) der gesuchten Leitung (Tag-Adresse) an den Level-2-Cache. Die Antwort (Tag-Daten) liefert die Information, ob sich die Leitung im Level-2-Cache befindet. Trifft dies zu, gibt die Information auch Auskunft über den Zustand der Leitung. In diesem Fall schnappt sich die CPU die Leitung. Die Datentransfers sind 16 Byte breit, so daß vier Zyklen nötig sind, um eine ganze Leitung in den Level-1-Cache einzulesen.

Befindet sich die Cache-Zeile nicht im Level-2-Cache, muß sie über die UPA-Schnittstelle vom Hauptspeicher geholt werden. Die UPA der UltraSPARC II ist mit einem zentralen Controller implementiert. Die Adreß- und Steuersignale von der CPU (bzw. allen CPUs, falls mehrere vorhanden sind) gelangen dorthin. Für den Speicherzugriff muß die CPU zuerst die Bus-Arbitration-Pins benutzen, um die Erlaubnis zum Fortfahren zu erhalten. Wird die Erlaubnis gewährt, gibt die CPU die Speicheradreßpins aus, legt die Anfrageart fest und asseriert den für die Adresse gültigen Pin. (Diese Pins sind bidirektional, weil andere CPUs in einem UltraSPARC-II-Mehrprozessor auf entfernte Caches zur Konsistenzwahrung für alle Caches zugreifen müssen.) Die Adreß- und Buszyklen werden in zwei Zyklen auf den Adreßpins ausgegeben, wobei die Zeile aus dem ersten Zyklus und die Spalte aus dem zweiten Zyklus herausführt, wie wir aus Abb. 3.31 wissen.

Während die CPU auf die Ergebnisse wartet, kann sie mit anderen Arbeiten fortfahren. Schlägt das Einlesen einer Instruktion vom Cache beispielsweise fehl, wird die Ausführung von bereits eingelesenen Instruktionen nicht beeinträchtigt. Mehrere Transaktionen können also in der UPA gleichzeitig ausstehen. Die UPA kann zwei unabhängige Transaktionsströme (normalerweise

Lesen und Schreiben) handhaben, während mehrere Transaktionen anstehen. Dem zentralen Controller wird es überlassen, dies alles zu verfolgen und die tatsächlichen Speicheranforderungen in der effizientesten Reihenfolge anzuordnen.

Daten können vom Speicher in je 8 Byte ankommen und enthalten zum Zwecke einer höheren Zuverlässigkeit einen 16-Bit-Fehlerkorrekturcode. Eine Transaktion kann einen gesamten Cache-Block, ein Quadwort (8 Byte) oder auch weniger Bytes anfordern. Alle ankommenden Daten fließen für die Pufferung zum UDB. Der UDB koppelt die CPU noch mehr vom Speichersystem ab, so daß beide asynchron arbeiten können. Muß die CPU beispielsweise ein Wort oder eine Cache-Zeile in den Speicher schreiben, braucht sie nicht auf den UPA-Zugriff zu warten, sondern kann die Daten sofort in den UDB schreiben. Vom UDB kann sie die Daten später abrufen. Außerdem erzeugt und prüft der UDB den Fehlerkorrekturcode. Die Beschreibung der UltraSPARC II ist wie die des Pentium II stark vereinfacht und auf die notwendigen Aspekte der jeweiligen Betriebsweise beschränkt.

3.5.3 picoJava II

Pentium II und UltraSPARC II sind Beispiele von Hochleistungs-CPUs für sehr schnelle PCs und Workstations. Wenn man an Computer denkt, neigt man dazu, sich auf diese Systemart zu konzentrieren. Daneben gibt es aber noch eine ganze Welt von Computern, die weit größer ist: eingebettete Systeme. In diesem Abschnitt befassen wir uns kurz mit dieser Welt.

Die Behauptung, daß in jedem elektrischen Gerät, das mehr als 100 Euro kostet, ein Computer eingebaut ist, ist wahrscheinlich kaum übertrieben. Sicherlich sind Fernseher, Handys, elektronische Personal-Organizer, Mikrowellenherde, Camcorder, Videorecorder, Laserdrucker, Einbruchsicherungen, Hörgeräte, elektronische Spiele und andere Geräte, die man gar nicht alle aufzählen kann, heutzutage computergesteuert. Die in diesen Dingen integrierten Computer werden nach Niedrigpreisigkeit und nicht nach Leistung optimiert. Dies führt zu anderen Kompromissen als bei den vorher beschriebenen CPUs des oberen Leistungsbereichs.

Traditionell werden eingebettete Prozessoren in Assembler programmiert. Mit zunehmender Komplexität von Geräten und zunehmend schweren Folgen von Software-Fehlern wendet man sich verstärkt anderen Ansätzen zu. Insbesondere ist die Benutzung von Java als Programmiersprache für eingebettete Systeme attraktiv, weil sie sich relativ leicht schreiben läßt, einen geringen Codeumfang hat und plattformunabhängig ist. Der größte Nachteil bei der Verwendung von Java in eingebetteten Anwendungen ist die Notwendigkeit eines großen Software-Interpreters zum Ausführen des JVM-Codes durch den Java-Compiler. Außerdem dauert der Interpretationsprozeß lang.

Sun und mehrere andere Unternehmen haben sich mit diesem Problem beschäftigt und CPU-Chips entwickelt, die JVM als ihre native Instruktions-

menge benutzen. Dieser Ansatz kombiniert die Vorteile von Java als Programmiersprache, die Portabilität und kleine Größe des vom Java-Compiler produzierten JVM-Binärcodes und die schnelle Ausführung durch spezialisierte Hardware. In diesem Abschnitt wird ein moderner, auf Java basierter CPU-Chip betrachtet, der eigens für den eingebetteten Systemmarkt entwikkelt wurde.

Der CPU-Chip ist Suns picoJava II, der auf der Architektur von Suns microJava-701 basiert, die an andere Unternehmen lizensiert wurde. Es handelt sich um eine Einzelchip-CPU mit zwei Busschnittstellen, eine für den 64 Bit breiten Speicherbus und eine für den 32 Bit breiten PCI-Bus (siehe Abb. 3.48). Wie der Pentium II und die UltraSPARC II ist auf dem Chip ein Level-1-Cache mit 16 Kbyte für Instruktionen und 16 Kbyte für Daten. Im Gegensatz zu diesen beiden CPUs hat der Chip aber keinen Level-2-Cache, da die Kosten bei den meisten eingebetteten Systemen eines der wichtigsten Designkriterien ist. Nachfolgend wird mit microJava-701 die Sun-Implementierung von picoJava II beschrieben. Der Chip ist gemessen an heutigen Maßstäben klein: nur 2 Millionen Transistoren für den Kern sowie weitere 1,5 Millionen für die beiden 16-Kbyte-Caches.

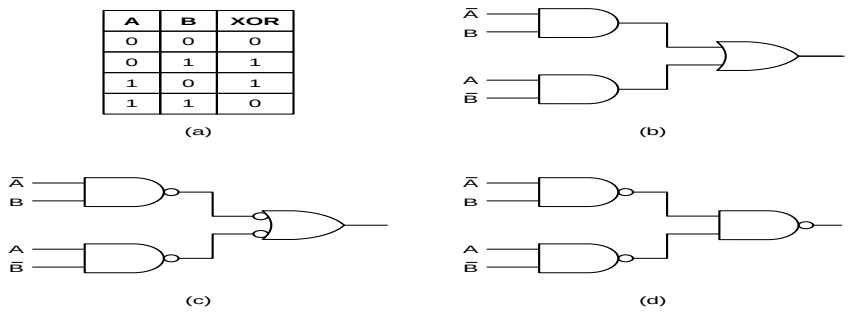


Abb. 3.48: Ein microJava-701-System

Drei Merkmale fallen in Abb. 3.48 sofort auf. Erstens nutzt microJava-701 den PCI-Bus (bei entweder 33 oder 66 MHz). Dieser Bus wurde von Intel für Pentium-Systeme im oberen Leistungsbereich entwickelt, der Prozessor ist aber unabhängig. Die Verwendung des PCI-Bus hat den Vorteil, daß man einen Standard nutzt und keinen neuen Bus entwickeln muß. Eine große Anzahl von Steckkarten ist zudem für ihn erhältlich. Für die Herstellung von Handys ist das Vorhandensein von PCI-Karten kaum von Nutzen, für Web-Fernseher und andere Geräte mit großen Abmessungen ist das aber ein Plus.

Zweitens beinhaltet ein microJava-701-System normalerweise einen Flash-PROM. Interessant daran ist, daß ein Großteil – wenn nicht das ganze Programm – für ein solches Gerät im Gerät integriert sein muß. Ein Flash-PROM ist eine gute Stelle zur Programmspeicherung. Deshalb ist es vorteilhaft, über ihn zu verfügen. Ein weiterer Chip (in der Abbildung nicht dargestellt), der in

ein System integriert werden kann, enthält die für einen PC üblichen seriellen und parallelen E/A-Schnittstellen.

Drittens hat microJava-701 16 programmierbare E/A-Leitungen, an die man Knöpfe, Schalter und Lampen von Geräten anschließen kann. Ein Mikrowellenherd hat z.B. normalerweise ein numerisches Tastenfeld und ein paar Knöpfe, die man direkt an den CPU-Chip anschließen kann. Mit programmierbaren E/A-Leitungen direkt an der CPU braucht man keine programmierbaren E/A-Controller-Chips, was einfacheres Design und geringere Kosten bedeutet. Auf dem Chip befinden sich außerdem drei programmierbare Zeitgeber (Timer), die ebenfalls sehr nützlich sind, weil viele Geräte in Echtzeit arbeiten.

Der microJava-701-Chip wird in der Industriestandard-Ausführung mit 316 Pins als **BGA** (**Ball Grid Array**) ausgeliefert. Von diesen Pins werden 59 an den PCI-Bus angeschlossen. Der PCI-Bus wird später in diesem Kapitel ausführlich behandelt. Weitere 123 Pins dienen dem Speicherbus, darunter 64 unidirektionale Daten- und getrennte Adreßpins. Weitere Pins dienen der Steuerung (7), dem Timer (3), den Interrupts (11), dem Test (10) und der programmierbaren E/A (16). Einige der restlichen Pins werden (redundant) für Strom und Masse benutzt, und einige sind nicht belegt. Anderen Herstellern von picoJava-II-Systemen steht es frei, einen anderen Bus, eine andere Baueinheit usw. zu wählen.

Der Chip weist außerdem viele weitere Hardware-Merkmale auf, z.B. einen Schlafmodus, um Batterie zu sparen, einen auf dem Chip integrierten Interrupt-Controller und volle Unterstützung des IEEE-Standards 1149.1 für JTAG-Tests.

3.6 Beispielbusse

Busse sind der Klebstoff, der Computer-Systeme zusammenhält. In diesem Abschnitt werden einige beliebte Busse näher betrachtet: ISA, PCI und USB (Universal Serial Bus). Der ISA-Bus ist eine leicht erweiterte Ausführung des ursprünglichen IBM-PC-Busses. Aus Gründen der Abwärtskompatibilität ist er immer noch in allen Intel-basierten PCs vorhanden. Alle diese Rechner haben aber noch einen zweiten, schnelleren Bus – den PCI-Bus. Der PCI ist breiter als der ISA und läuft mit einer höheren Taktrate. Der USB setzt sich immer stärker als E/A-Bus für Peripheriegeräte mit niedriger Geschwindigkeit, wie Mäuse und Tastaturen, durch. Diese Busse werden im folgenden beschrieben.

3.6.1 ISA-Bus

Der IBM-PC-Bus war der De-facto-Standard in 8088-Systemen, weil ihn fast alle Hersteller von PC-Clones kopiert haben, um die große Zahl 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

die Assertion von Leseoperationen zum/vom Speicher und Ein-/Ausgaben. Außerdem gab es Signale für die Anfrage und Gewährung von Interrupts und die DMA-Verwendung, und das war's schon. Es war ein sehr einfacher Bus.

Physisch war der Bus auf dem PC-Motherboard aufgelötet, mit etwa einem halben Dutzend Connectors in einem Abstand von 2 cm, in die man Karten einstecken konnte. Jede Karte hatte eine Kerbe, die in einen dieser Connectors paßte. Die Kerbe hatte 31 vergoldete Streifen auf jeder Seite, durch die der elektrische Kontakt mit dem Connector hergestellt wurde.

Als IBM den 80286-PC/AT vorstellte, trat ein Problem auf. Hätte das Unternehmen einen völlig neuen 16-Bit-Bus entwickelt, hätten viele potentielle Kunden mit dem Kauf gezögert, weil die zahlreichen PC-Steckkarten von Drittanbietern nicht mit der neuen Maschine funktioniert hätten. Wäre man andererseits bei dem PC-Bus mit seinen 20 Adreß- und 8 Datenleitungen geblieben, hätte man die Fähigkeit des 80286, 16 Mbyte Speicher zu adressieren und 16-Bit-Wörter zu übertragen, nicht nutzen können.

Als Lösung entschied man sich für die Erweiterung des PC-Bus. PC-Steckkarten haben einen Edge-Connector mit 62 Kontakten, der sich aber nicht über die ganze Länge der Karte erstreckt. Als PC/AT-Lösung versah man die Karte an der Unterseite mit einem zweiten Connector neben dem ersten und änderte die AT-Schalttechnik so, daß sie beide Kartenarten unterstützte. Das allgemeine Konzept ist in Abb. 3.49 dargestellt.

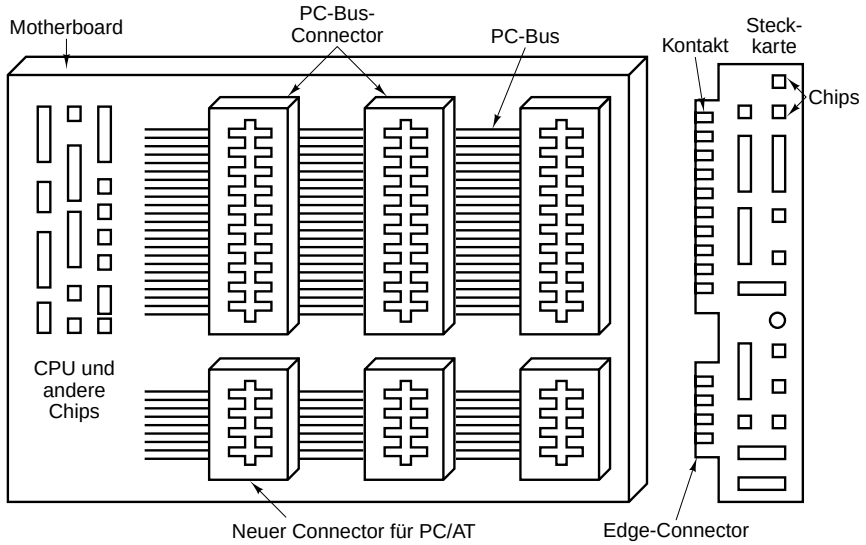


Abb. 3.49: Der PC/AT-Bus hat zwei Komponenten, das alte PC-Teil und ein neues Teil

Der zweite Connector am PC/AT-Bus enthält 36 Leitungen. Davon sind 31 für mehr Adreßleitungen, mehr Datenleitungen, mehr Interrupt-Leitungen und

mehr DMA-Kanäle sowie Strom und Masse vorgesehen. Der Rest beschäftigt sich mit dem Unterschied zwischen 8-Bit- und 16-Bit-Transfers.

Als IBM die PS/2-Serie als Nachfolger des PC und PC/AT herausbrachte, entschloß sich das Unternehmen zu einem Neubeginn. Diese Entscheidung gründete teilweise auf technischen Überlegungen (der PC-Bus war zu dieser Zeit wirklich überholt), teilweise aber auch auf dem Wunsch, den Herstellern von PC-Klons ein Bein zu stellen. Diese hatten nämlich einen beängstigend großen Marktanteil. 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.

Der Rest der PC-Industrie reagierte auf diese Entwicklung mit der Übernahme eines eigenen Standards, den **ISA**-Bus (ISA = **Industry Standard Architecture**), im Grunde genommen der PC/AT-Bus mit 8,33 MHz. Dies bot den Vorteil, daß die Kompatibilität mit allen existierenden Rechnern und Karten gewahrt wurde. Er basiert auch auf einem Bus, den IBM freizügig an viele Unternehmen lizensiert hatte, damit möglichst viele Drittanbieter möglichst viele Karten für den Original-PC produzierten, was sich im nachhinein als Verhängnis für IBM herausstellte. In jedem Intel-basierten PC gibt es diesen Bus immer noch, normalerweise aber zusammen mit einem oder mehreren anderen Bussen. Der Leser findet eine umfangreiche Beschreibung in Shanley und Anderson (1995a).

Später wurde der ISA-Bus auf 32 Bits und um ein paar neue Merkmale (z.B. für Multiprocessing) erweitert. Diesen neuen Bus nannte man **EISA** (**Extended ISA**). Für ihn wurden aber nur wenige Karten produziert.

3.6.2 PCI-Bus

Im ursprünglichen IBM-PC waren die meisten Anwendungen textbasiert. Nach und nach, mit der Einführung von Windows, verbreiteten sich grafische Benutzeroberflächen. Keine dieser Anwendungen hat dem ISA-Bus viel abgefordert. Im Laufe der Zeit und vielen neuen Anwendungen, insbesondere Multimedia-Spielen, änderte sich diese Situation radikal.

Stellen wir am Beispiel von Laufbildern auf einem Bildschirm mit einer Auflösung von 1024 × 768 Pixel und hoher Farbauflösung (TrueColor = 3 Byte/Pixel) eine einfache Rechnung auf. Ein Bild einer Laufbildserie umfaßt 2,25 Mbyte an Daten. Damit die Bewegungen ruckelfrei ablaufen, sind mindestens 30 Bilder/s erforderlich, und dies bei einer Datenrate von 67,5 Mbyte/s. Schlimmer noch: Will man Videodaten von einer Festplatte, CD-ROM oder DVD ablaufen lassen, müssen sie vom Laufwerk über den Bus zum Speicher fließen. Für die Anzeige reisen die Daten dann erneut über den Bus zum Grafikadapter. Wir brauchen also eine Busbandbreite von 135 Mbyte/s allein für das Video, ohne die Bandbreite, die für die CPU und die anderen Geräte benötigt wird.

Der ISA-Bus läuft mit einer Höchstrate von 8,33 MHz und kann pro Zyklus 2 Bytes übertragen. Die maximale Bandbreite ist 16,7 Mbyte/s. Der EISA-Bus kann 4 Bytes pro Zyklus befördern und erreicht 33,3 Mbyte/s. Natürlich kommen beide nicht annähernd an die für ein Vollbildvideo nötige Menge.

In weiser Voraussicht entwickelte Intel 1990 einen neuen Bus mit einer viel höheren Bandbreite als sogar der EISA-Bus und nannte ihn **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 lizensieren konnte. Ferner gründete Intel ein Industriekonsortium, die PCI Special Interest Group, um die Zukunft des PCI-Bus zu verwalten. Diese Schritte führten dazu, daß der PCI-Bus sehr beliebt wurde. Praktisch jeder Intel-basierte Computer seit dem Pentium hat heute einen PCI-Bus, und viele andere Computer auch. Sun hat sogar eine Version der UltraSPARC, die UltraSPARC IIi, mit dem PCI-Bus ausgestattet. Der PCI-Bus wird ausführlich in Shanley und Anderson (1995b) und Solari und Willse (1998) beschrieben.

Der ursprüngliche PCI-Bus übertrug 32 Bit pro Zyklus und lief mit 33 MHz (d.h. einer Zykluszeit von 30 ns). Die Bandbreite betrug insgesamt 133 Mbyte/s. 1993 wurde der PCI 2.0 eingeführt, und 1995 kam PCI 2.1 heraus. PCI 2.2 hat Eigenschaften, die ihn für mobile Computer geeignet machen (vorwiegend zur Einsparung von Batteriestrom). Der PCI-Bus läuft mit bis zu 66 MHz und handhabt 64-Bit-Transfers. Er hat eine Bandbreite von insgesamt 528 Mbyte/s. Mit diesen Fähigkeiten ist Vollbildvideo machbar (unter der Annahme, daß die Platte und der Rest des Systems diesen Anforderungen genügen). Auf jeden Fall ist der PCI-Bus nicht der Flaschenhals bei solchen Anwendungen.

Nun hört sich eine Rate von 528 Mbyte/s zwar schnell an, ist aber nicht ganz unproblematisch. Erstens ist sie nicht gut genug für einen Speicherbus. Zweitens ist sie nicht mit den vielen alten ISA-Karten im Feld kompatibel. Intel dachte sich eine Lösung aus und entwickelte Computer mit drei oder mehr Bussen. Wir sehen in Abb. 3.50, daß die CPU mit dem Hauptspeicher über einen speziellen Speicherbus kommuniziert, und daß ein ISA-Bus an den PCI-Bus angeschlossen werden kann. Diese Anordnung erfüllt alle Anforderungen. Praktisch alle Pentium-II-Rechner weisen diese Architektur auf.

Zwei Schlüsselkomponenten bei dieser Architektur sind die Zwei-Brücken-Chips (die Intel fertigt, daher das Interesse an dem ganzen Projekt). Die PCI-Brücke verbindet die CPU, den Speicher und den PCI-Bus. Die ISA-Brücke verbindet den PCI-Bus mit dem ISA-Bus und unterstützt eine oder zwei IDE-Platten. Fast alle Pentium-II-Systeme haben einen oder mehr freie PCI-Steckplätze für neue schnelle Peripheriegeräte und einen oder mehr ISA-Steckplätze für langsame Peripheriegeräte.

Das System in Abb. 3.50 hat den großen Vorteil, daß die CPU über einen herstellerspezifischen Speicherbus bei einer sehr hohen Bandbreite zum Speicher verfügt. Der PCI-Bus hat eine sehr hohe Bandbreite für schnelle Peripheriege-

räte, z.B. SCSI-Platten, Grafikadapter usw., und ältere ISA-Karten können weiter benutzt werden. Die USB-Box in der Abbildung bezieht sich auf den Universal Serial Bus, der später in diesem Kapitel behandelt wird.

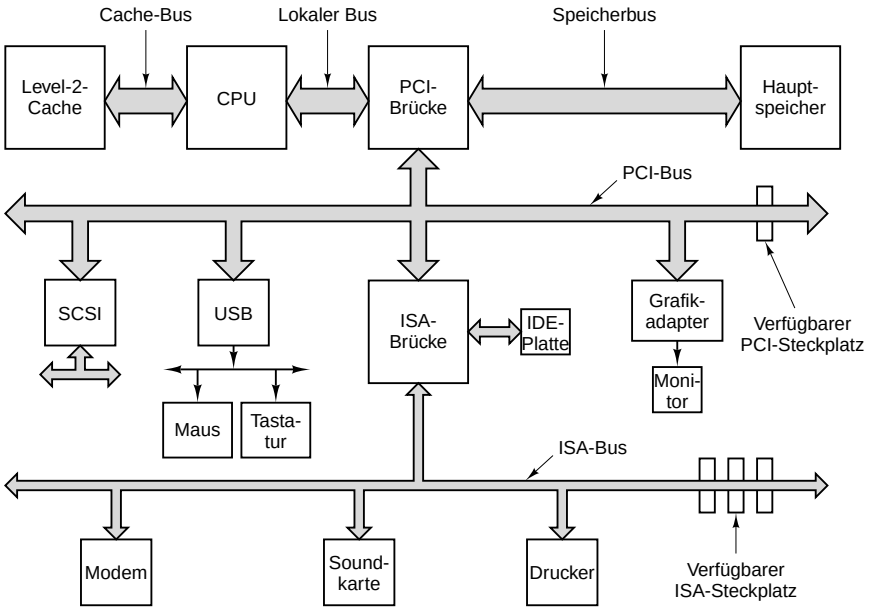


Abb. 3.50: Architektur eines typischen Pentium-II-Systems; die dickeren Busse haben mehr Bandbreite als die dünneren

Wir haben zwar ein System mit einem PCI-Bus und einem ISA-Bus dargestellt. Es sind aber auch mehrere von beiden möglich. Es sind PCI/PCI-Brükkenchips erhältlich, die zwei PCI-Busse verbinden, so daß größere Systeme mit zwei oder mehr unabhängigen PCI-Bussen ausgestattet werden können. Außerdem ist es möglich, mehrere ISA-Busse über zwei oder mehr PCI/ISA-Brückenchips anzuschließen.

Es wäre angenehm, wenn es nur eine Sorte PCI-Karten gäbe. Leider ist das nicht der Fall. Für Spannung, Breite und Takt gibt es unterschiedliche Varianten. Ältere Computer nutzen meist 5 Volt, und neuere neigen zu 3,3 Volt, so daß der PCI-Bus beides unterstützt. Die Connectors sind gleich, mit Ausnahme von zwei Kunststoffplättchen, die verhindern sollen, daß man eine 5-Volt-Karte in einen 3,3-Volt-Bus, oder umgekehrt, einsteckt. Zum Glück gibt es auch universell einsetzbare Karten, die beide Spannungen unterstützen und in jeden Steckplatz eingeschoben werden können. Abgesehen von den beiden Spannungen gibt es Karten in 32- und 64-Bit-Version. Die 32-Bit-Karten haben 120 Pins. Die 64-Bit-Karten haben die gleichen 120 sowie zusätzlich 64 Pins, analog der Erweiterung des IBM-PC-Bus auf 16 Bit (siehe Abb. 3.49). Ein PCI-Bus-System, das 64-Bit-Karten unterstützt, kann auch 32-Bit-

Karten aufnehmen, umgekehrt geht es aber nicht. Ferner können PCI-Busse mit 33 oder 66 MHz laufen. Dafür verbindet man einen Pin entweder mit der Stromversorgung oder mit der Masse. Die Connectors sind bei beiden Geschwindigkeiten die gleichen.

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 plus Datensignale erforderlich, obwohl der PCI 64-Bit-Adressen und 64-Bit-Daten unterstützt.

Die gemultiplexten Adreß- und Datenpins funktionieren wie folgt: Bei einer Leseoperation im Zyklus 1 setzt der Master die Adresse auf den Bus. In Zyklus 2 entfernt der Master die Adresse, und der Bus wird umgeschaltet, so daß ihn der Slave benutzen kann. In Zyklus 3 gibt der Slave die angeforderten Daten aus. Bei Schreiboperationen muß der Bus nicht umgeschaltet werden, weil der Master Adresse und Daten ausgibt. Die minimale Transaktion erfordert aber immer drei Zyklen. Ist der Slave nicht in der Lage, in drei Zyklen zu reagieren, kann er Wartezustände einfügen. Blocktransfers in unbegrenzter Größe und verschiedene andere Arten von Buszyklen sind ebenfalls zulässig.

PCI-Bus-Arbitration

Damit ein Gerät den PCI-Bus benutzen kann, muß es ihn zuerst in Beschlag nehmen. Die PCI-Bus-Arbitration benutzt dafür einen zentralen Bus-Arbiter (siehe Abb. 3.51). Bei den meisten Designs ist der Bus-Arbiter in einen der Brückenchips eingebaut. Jedes PCI-Gerät hat zwei bestimmte Leitungen, die vom Gerät zum Arbiter führen. Mit der Leitung REQ# wird der Bus angefordert. Leitung GNT# nimmt Busgewährungen entgegen.

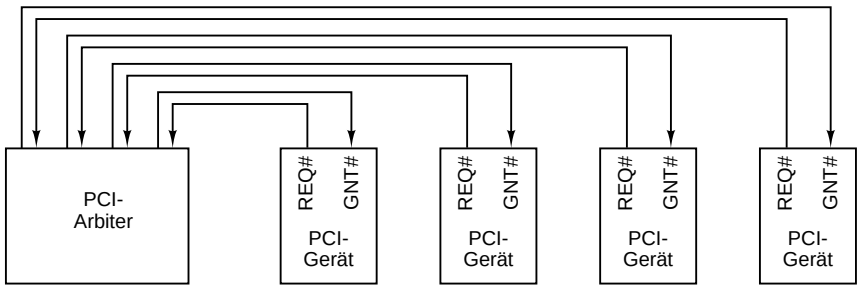


Abb. 3.51: Der PCI-Bus nutzt einen zentralen Bus-Arbiter

Um den Bus anzufordern, asseriert ein PCI-Gerät (auch die CPU zählt dazu) REQ# und wartet, bis seine GNT#-Leitung vom Arbiter asseriert wird. Dann kann das Gerät den Bus im nächsten Zyklus benutzen. Der vom Arbiter benutzte Algorithmus ist in der PCI-Spezifikation nicht definiert. Round-Robin-

Arbitration, Prioritäten-Arbitration und andere Verfahren sind zulässig. Selbstverständlich ist ein guter Arbiter fair und läßt kein Gerät ewig warten.

Die Nutzung des Bus wird für eine Transaktion gewährt, die Länge dieser Transaktion ist theoretisch aber unbegrenzt. Möchte ein Gerät eine zweite Transaktion durchführen, und wird der Bus von keinem anderen Gerät angefordert, kann es erneut starten, obwohl normalerweise zwischen zwei Transaktionen ein leerer Zyklus ablaufen muß. Unter besonderen Umständen – wenn nicht um den Bus geworben wird – kann ein Gerät Transaktionen hintereinander ausführen, ohne einen leeren Zyklus zu respektieren. Führt ein Busmaster einen sehr langen Transfer durch, und hat ein anderes Gerät währenddessen den Bus angefordert, kann der Arbiter die GNT#-Leitung negieren. In diesem Fall muß der momentane Busmaster den Bus im nächsten Zyklus freigeben. Dieses Vorgehen ermöglicht sehr lange Transfers (was effizient ist), wenn sich nur ein Kandidat um den Bus bewirbt, wobei dennoch auf Anfragen von Geräten schnell reagiert wird.

PCI-Bussignale

Der PCI-Bus hat eine Reihe von zwingenden (Abb. 3.52(a)) und eine Reihe von optionalen Signalen (Abb. 3.52(b)). Der Rest der 120 bzw. 184 Pins wird für Strom, Masse und verschiedene, damit zusammenhängende Funktionen benutzt, die hier nicht behandelt werden. In den Spalten *Master* und *Slave* sieht man, wer das Signal in einer normalen Transaktion asseriert. Wird das Signal von einem anderen Gerät asseriert (z.B. CLK [Takt]), bleiben beide Spalten leer.

Wir betrachten nun die einzelnen PCI-Bussignale und beginnen mit den zwingenden (32-Bit-)Signalen. Anschließend werden die optionalen (64-Bit-)-Signale beschrieben. 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 Anfang, sondern die Mitte des Zyklus.

Die 32 AD-Signale sind für die Adresse und die Daten (für 32-Bit-Transaktionen) bestimmt. Im allgemeinen wird die Adresse in Zyklus 1 asseriert und die Daten in Zyklus 3. Das PAR-Signal ist ein Paritätsbit für AD. Das Signal C/BE# wird für zwei verschiedene Aktionen benutzt. In Zyklus 1 enthält es den Busbefehl (1 Wort lesen, Lesen blockieren usw.). In Zyklus 2 enthält es eine Bitmap aus 4 Bits, die besagt, welche Bytes des 32-Bit-Worts gültig sind. Mit C/BE# können 1, 2 oder 3 Bytes oder das ganze Wort beliebig gelesen oder geschrieben werden.

Das Signal FRAME# wird vom Busmaster für den Start einer Bustransaktion asseriert. Es informiert den Slave darüber, daß die Adreß- und Busbefehle jetzt gültig sind. In einer Lesetransaktion wird IRDY# normalerweise gleichzeitig mit FRAME# asseriert. Dies besagt, daß der Master zur Entgegennahme der ankommenden Daten bereit ist. Bei einer Schreiboperation wird IRDY# später asseriert, wenn die Daten auf dem Bus sind.

Signal	Leitungen	Master	Slave	Erläuterung
CLK	1			Takt (33 oder 66 MHz)
AD	32	×	×	Gemultiplexte Adreß- und Datenleitungen
PAR	1	×		Adreß- oder Datenparitätsbit
C/BE	4	×		Busbefehl/Bitmap für eingeschaltete Bytes
FRAME#	1	×		Kennzeichnet, daß AD und C/BE asseriert 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-Arbitration: Anforderung des Bus
GNT#	1			Bus-Arbitration: Gewährung des Bus
RST#	1			Setzt das System und alle Geräte zurück

(a)

Zeichen	Leitungen	Master	Slave	Erläuterung
REQ64#	1	×		Anfrage zur Ausführung einer 64-Bit-Transaktion
ACK64#	1		×	Gewährung für eine 64-Bit-Transaktion
AD	32	×		Weitere 32 Bit für Adresse oder Daten
PAR64	1	×		Parität für die 32 zusätzlichen Adreß-/Datenbits
C/BE#	4	×		Weitere 4 Bit für Byteeinschaltungen
LOCK	1	×		Sperren des Bus, um mehrere Transaktionen zu gewähren
SBO#	1			Treffer auf 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			Verdrahtung mit Strom oder Masse (66 oder 33 MHz)

(b)

Abb. 3.52: (a) Zwingende und (b) optionale PCI-Bussignale

Das Signal IDSEL bezieht sich auf die Tatsache, daß jedes PCI-Gerät über einen Konfigurationsraum von 256 Byte verfügen muß, den die anderen Geräte (durch Asserierung von IDSEL) lesen können. Dieser Konfigurationsraum enthält die Eigenschaften des Geräts. Die Plug-and-Play-Eigenschaft mancher Betriebssysteme nutzt ihn, um herauszufinden, welche Geräte sich am Bus befinden.

Nun kommen wir zu den Signalen, die vom Slave asseriert werden. 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 asseriert, läuft der Timer des Masters ab; der Master nimmt dann an, daß das adressierte Gerät nicht vorhanden oder ausgefallen ist.

Das Signal TRDY# wird vom Slave bei Leseoperationen für die Ankündigung asseriert, daß sich die Daten auf den AD-Leitungen befinden. Bei Schreiboperationen kündigt es an, daß er zur Datenannahme bereit ist.

Die nächsten drei Signale sind für Fehlermeldungen reserviert. STOP# wird vom Slave asseriert, wenn etwas Verheerendes 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 asseriert. SERR# wird zum Melden von Adressier- und Systemfehlern benutzt.

Die Signale REQ# und GNT# dienen der Bus-Arbitration. Sie werden nicht vom momentanen Busmaster, sondern von einem Gerät asseriert, das Busmaster werden möchte. Das letzte zwingende Signal ist RST#. Mit ihm setzt man das System zurück, wenn der Benutzer die RESET-Taste drückt oder ein Systemgerät einen nicht reparierbaren Fehler feststellt. Die Assertion dieses Signals setzt alle Geräte zurück und startet den Computer neu.

Von den optionalen Signalen beziehen sich die meisten auf die Erweiterung von 32 auf 64 Bit. Die Signale REQ64# und ACK64# erlauben es dem Master, die Erlaubnis zur Durchführung einer 64-Bit-Transaktion einzuholen, bzw. dem Slave die Erlaubnis zu gewähren. 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 kann der Bus für mehrere Transaktionen gesperrt werden. Die nächsten beiden Signale beziehen sich auf Bus-Snooping zur Wahrung der Cache-Kohärenz.

Die INTX-Signale werden zur Anforderung von Interrupts benutzt. An eine PCI-Karte können bis zu vier logische Geräte angeschlossen werden. Jedes davon kann eine eigene Interrupt-Anfrageleitung haben. Die JTAG-Signale sind für das JTAG-Testverfahren gemäß IEEE-Standard 1149.1 reserviert. Das Signal M66EN wird entweder auf High oder Low für das Setzen der Takt-

geschwindigkeit geschaltet. Es darf sich während des Systembetriebs nicht ändern.

PCI-Bustransaktionen

Der PCI-Bus ist eigentlich sehr einfach. Um dafür ein besseres Gefühl zu entwickeln, betrachte man das Taktdiagramm der Abb. 3.53. Hier erkennt man eine Lesetransaktion, der ein leerer Zyklus sowie eine Schreibtransaktion durch den gleichen Busmaster folgen.

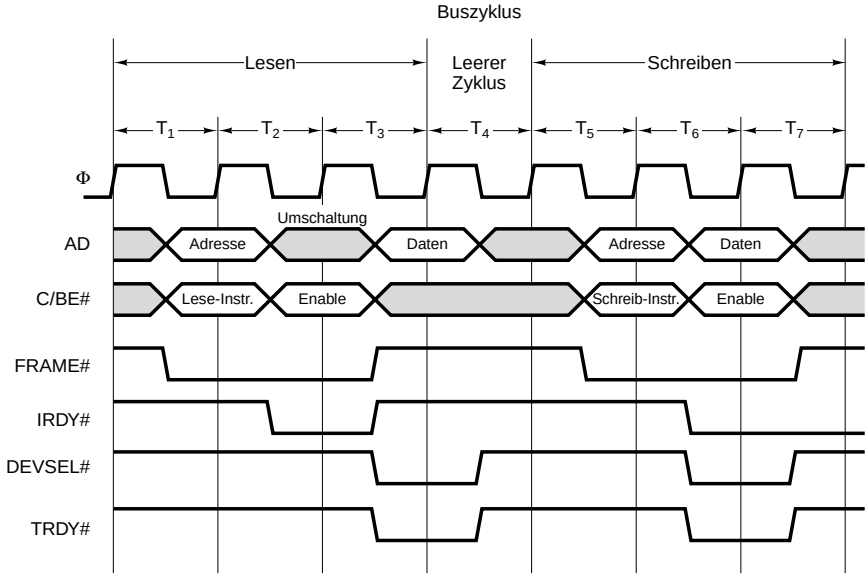


Abb. 3.53: Beispiele von PCI-Bustransaktionen (32 Bit). Die ersten drei Zyklen dienen der Leseoperation, dann folgt ein leerer Zyklus, und anschließend folgen drei Zyklen für eine Schreiboperation.

Wenn die fallende Flanke des Takts innerhalb von T_1 erfolgt, setzt der Master die Speicheradresse auf AD und den Busbefehl auf C/BE#. Dann asseriert er FRAME#, um die Bustransaktion zu starten.

Im Verlauf von T_2 puffert der Master den Adreßbus, damit dieser sich in Vorbereitung auf den Slave in T_3 umschaltet. Außerdem ändert der Master C/BE#, um anzuzeigen, welche Bytes im adressierten Wort eingeschaltet (d.h. eingelesen) werden sollen.

In T_3 asseriert der Slave DEVSEL#, damit der Master weiß, daß er die Adresse hat, und seinerseits die Antwort plant. Außerdem setzt er die Daten auf die AD-Leitungen und asseriert TRDY#, um dies dem Master mitzuteilen. Auch wenn der Slave nicht in der Lage war, so schnell zu antworten, so asseriert er dennoch DEVSEL#, um seine Anwesenheit bekanntzugeben. Er hält TRDY#

aber negiert, bis er die Daten herausnehmen kann. Dieses Verfahren bringt einen oder mehrere Wartezustände mit sich.

Bei diesem Beispiel (und oft auch in der Wirklichkeit) ist der nächste Zyklus leer. Beginnend in T_5 leitet der gleiche Master eine Schreiboperation ein. Zunächst setzt er auf die übliche Weise die Adresse und den Befehl auf den Bus. Im zweiten Zyklus setzt er die Daten. Da das gleiche Gerät die AD-Leitungen steuert, besteht keine Notwendigkeit eines Umschaltzyklus. In T_7 nimmt der Speicher die Daten an.

3.6.3 USB (Universal Serial Bus)

Der PCI-Bus eignet sich gut zum Anschließen von Hochgeschwindigkeitsperipherie an einen Computer, ist aber bei weitem zu teuer, wenn man über eine PCI-Schnittstelle für jedes langsame E/A-Gerät, z.B. eine Tastatur oder eine Maus, verfügt. Historisch wurde jedes E/A-Standardgerät auf besondere Weise an den Computer angeschlossen, mit einigen freien ISA- und PCI-Steckplätzen zum Hinzufügen neuer Geräte. Leider war dieses Verfahren von Anfang an mit Problemen behaftet. Jedes neue E/A-Gerät wird beispielsweise meist mit einer eigenen ISA- oder PCI-Karte ausgeliefert. Die Einstellung der Schalter und Brücken (Jumper) auf der Karte wird dem Benutzer überlassen. Er muß sich auch darum kümmern, daß zwischen den einzelnen Karten kein Konflikt entsteht. Der Benutzer muß das Gehäuse öffnen, die Karte vorsichtig einstekken, das Gehäuse schließen und den Rechner starten. Für viele Benutzer ist dieser Vorgang schwierig und birgt viele Fehlerquellen. Außerdem ist die Anzahl der ISA- und PCI-Steckplätze immer begrenzt (normalerweise auf zwei oder drei). Plug-and-Play-Karten ersparen dem Benutzer zwar die Einstellung der Brücken. Er muß aber trotzdem für die Installation den Computer öffnen, und die Bussteckplätze sind nach wie vor begrenzt.

Zur Lösung dieses Problem haben sich Mitte der neunziger Jahre Vertreter von sieben Unternehmen (Compaq, DEC, IBM, Intel, Microsoft, NEC und Northern Telecom) zusammengefunden, um über ein besseres Design zu beraten, wie langsame E/A-Geräte an einen Computer angeschlossen werden können. Seither sind Hunderte weiterer Unternehmen der Gruppe beigetreten. Der daraus entwickelte Standard heißt **USB** (**Universal Serial Bus**). Er wird inzwischen in vielen Computern implementiert. Der Standard wird ausführlich in Anderson (1997) und Tan (1997) beschrieben.

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 setzen 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. E/A-Geräte sollen über das Kabel mit Strom versorgt werden.
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 installiert werden können.
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. Die USB-Bandbreite ist insgesamt 1,5 Mbyte/s. Das ist für viele Geräte dieser Art ausreichend. Diese niedrige Zahl wurde mit Blick auf Kosteneindämmung gewählt.

Ein USB-System besteht aus einem **Root-Hub**, der in den Hauptbus (siehe Abb. 3.50) eingesteckt wird. Dieser Hub hat Buchsen für Kabel, über die E/A-Geräte oder Erweiterungshubs angeschlossen werden können, so daß weitere Buchsen bereitstehen. Die Topologie eines USB-Systems ist also ein Baum, dessen Wurzel der Root-Hub im Innern des Computers bildet. Die Kabel haben unterschiedliche Connectors am Hub-Ende und am Geräteende, um zu verhindern, daß man versehentlich zwei Hubs zusammensteckt.

Das Kabel besteht aus vier Drähten – zwei für Daten, einen für Strom (+5 Volt) und einen für Masse. Das Signalisierungssystem überträgt eine 0 als Spannungsübergang und eine 1 in Abwesenheit eines Spannungsübergangs, so daß 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 wieviel 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, um Register im Innern des Geräts zu konfigurieren. Auf diese Weise können neue Geräte problemlos hinzugefügt werden, ohne daß der Benutzer etwas konfigurieren oder installieren muß. Nicht initialisierte Karten beginnen mit der Adresse 0, damit auch sie angesprochen werden können. 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 das USB-System als Menge von Bit-Pipes vom Root-Hub zu den E/A-Geräten betrachtet werden. 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 den beiden E/A-Geräten fließt kein Verkehr.

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

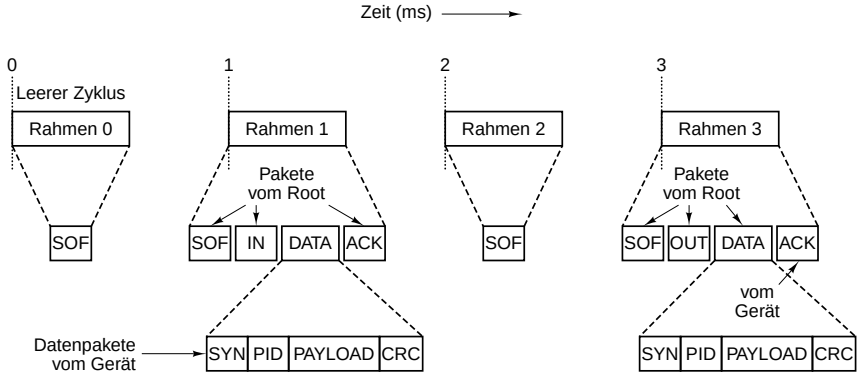


Abb. 3.54: Der USB-Root-Hub sendet alle 1,00 ms Rahmen aus

In Abb. 3.54 muß in den Rahmen 0 und 2 keine Arbeit verrichtet werden, so daß nur ein SOF-Paket (`Start of Frame`) erforderlich ist. Dieses Paket wird immer an alle Geräte gesendet. Rahmen 1 ist eine Abfrage, z.B. eine Anforderung an einen Scanner, die in einem Bild gelesenen Bits zurückzugeben. Rahmen 3 besteht aus den Zustelldaten an ein Gerät, z.B. einen Drucker.

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

Ein Rahmen besteht aus einem oder mehr 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 Abb. 3.54

sind Token-Pakete. Das Paket SOF (Start of Frame) ist das erste in jedem Rahmen. Es kennzeichnet den Rahmenanfang. Ist keine Arbeit zu verrichten, ist SOF das einzige Paket im Rahmen. IN ist ein Token-Paket für die Abfrage von Geräten, damit diese bestimmte Daten zurückgeben. Die Felder im IN-Paket geben Auskunft darüber, welche Bit-Pipe abgefragt wird, so daß das Gerät weiß, welche Daten zurückzugeben sind (wenn es mehrere Ströme hat). OUT ist ein Token-Paket, das ankündigt, daß Daten für das Gerät folgen. SETUP (in der Abbildung nicht vorhanden) ist ein Token-Paket für die Konfiguration.

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

Betrachten wir wieder Abb. 3.54. Alle 1,00 ms muß ein Rahmen vom Root-Hub gesendet werden, auch wenn keine Arbeit ansteht. Die Rahmen 0 und 2 bestehen nur aus einem SOF-Paket, aus dem hervorgeht, daß keine Arbeit anstand. Rahmen 1 ist eine Abfrage; deshalb beginnt er mit einem SOF- und einem IN-Paket 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 Rahmen 3 entspricht dem von Rahmen 1, außer daß die Daten vom Computer zum Gerät fließen.

3.7 Schnittstellen

Ein typisches kleines bis mittleres Computer-System besteht aus einem CPU-Chip, Speicherchips und einigen allesamt über einen Bus angeschlossenen E/A-Controllern. Wir haben Speicher, CPU und Busse mehr oder weniger ausführlich behandelt. Nun ist es an der Zeit, den letzten Teil des Puzzles – die E/A-Chips – zu betrachten. Diese Chips ermöglichen einem Computer die Kommunikation mit der Außenwelt.

3.7.1 E/A-Chips

E/A-Chips sind in Hülle und Fülle erhältlich, und laufend werden neue eingeführt. Zu den üblichen zählen UARTs, USARTs, CRT-Controller, Platten-Controller und PIOs. Ein **UART** (**Universal Asynchronous Receiver Transmitter**) ist ein Chip, der ein Byte vom Datenbus lesen und je ein Bit über eine serielle Leitung an ein Terminal ausgeben oder vom Terminal empfangen kann. UARTs unterstützen verschiedene Geschwindigkeiten und Zeichenbrei-

ten von 5 bis 8 Bits sowie 1, 1,5 oder 2 Stoppbits. Sie bieten gerade (EVEN), ungerade (ODD) oder keine (NONE) Parität; all dies läuft unter Programmsteuerung ab. Ein **USART** (**Universal Synchronous Asynchronous Receiver Transmitter**) kann die synchrone Übertragung mit Hilfe verschiedener Protokolle handhaben und alle UART-Funktionen ausführen. Weil UART-Chips bereits in Kapitel 2 beschrieben wurden, betrachten wir hier als Beispiel eines E/A-Chips deren parallele Schnittstelle.

PIO-Chips

Ein typischer PIO-Chip (**PIO** = **Parallel Input/Output**) ist der Intel 8255A (siehe Abb. 3.55). Er hat 24 E/A-Leitungen, die als Schnittstellen für ein beliebiges, TTL-kompatibles Gerät dienen, z.B. Tastaturen, Schalter, Lampen oder Drucker. Kurz, das CPU-Programm kann auf jede Leitung eine 0 oder eine 1 schreiben oder den Eingangsstatus einer Leitung lesen, so daß große Flexibilität geboten wird. Ein kleines, CPU-basiertes System mit einem PIO kann oft eine ganze, mit SSI- oder MSI-Chips voll bestückte Platine ersetzen.

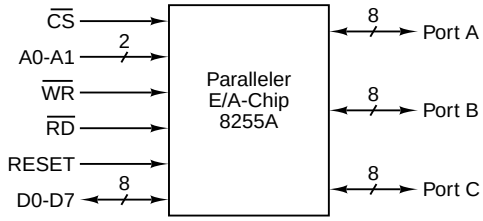


Abb. 3.55: Der PIO-Chip 8255A

Die CPU kann den 8255A auf vielerlei Weise konfigurieren, wenn sie Statusregister im Chip lädt. Wir konzentrieren uns aber auf einfachere Betriebsarten. In der einfachsten Einsatzart fungiert der 8255A als drei unabhängige 8-Bit-Ports A, B und C. Mit jedem Port steht ein 8-Bit-Register in Verbindung. Um die Leitungen eines Ports 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. Zur Nutzung eines Ports für Eingaben liest die CPU das entsprechende Register.

Weitere Betriebsarten bieten Handshaking mit externen Geräten. Um beispielsweise eine Ausgabe an ein nicht ständig zur Datenaufnahme bereites Gerät zu schicken, kann der 8255A an einem Ausgangsport Daten präsentieren und warten, bis das Gerät einen Impuls zurückschickt, durch den es angibt, daß es Daten angenommen hat und auf weitere wartet. Die nötige Logik, um solche Impulse zu speichern und sie für die CPU bereitzustellen, ist in der 8255A-Hardware enthalten.

Aus dem Funktionsdiagramm des 8255A ist ersichtlich, daß er zusätzlich zu 24 Pins für die drei Ports noch über acht Leitungen verfügt, die direkt an den Datenbus, eine Chipauswahlleitung, Lese- und Schreibleitungen, zwei Adreß-

leitungen und eine Leitung zum Zurücksetzen des Chips angeschlossen werden. Die beiden Adreßleitungen wählen einen der vier internen Register, entsprechend den Ports A, B, C und Statusregister. Im Statusregister befinden sich Bits, die ermitteln, welche Ports für Ein- und welche für Ausgaben benutzt werden. Normalerweise werden 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 asseriert wird. Nun ist es an der Zeit, sich mit diesem Thema zu befassen. Betrachten wir einen einfachen Mikrocomputer, der aus einer CPU, einem 2K × 8-EPROM für das Programm, einem 2K × 8-RAM für die Daten und einem PIO besteht. Dieses kleine System kann als Prototyp für das Gehirn eines billigen Spielzeugs oder eines einfachen Geräts benutzt werden. Einmal in der Produktion, kann der EPROM durch einen ROM ausgetauscht werden.

Der PIO kann auf zweierlei Weise ausgewählt werden: als echtes E/A-Gerät oder als Teil des Speichers. Wählen wir ihn als E/A-Gerät, müssen wir ihn mit Hilfe einer expliziten Busleitung, die das zu adressierende E/A-Gerät und nicht den Speicher bezeichnet, auswählen. Entscheiden wir uns für einen anderen Ansatz – **Memory-mapped I/O** –, müssen wir ihm 4 Byte des Adreßraums für die drei Ports und das Steuerregister zuweisen. Die Wahl ist mehr oder weniger willkürlich. Wir entscheiden uns für das Memory-mapped I/O, weil dieser Ansatz interessante Aspekte der E/A-Schnittstellen aufdeckt.

Der EPROM, aber auch der RAM, brauchen einen Adreßraum von 2K. Der PIO benötigt 4 Byte. Unser Beispiel hat einen Adreßraum von 64K. Deshalb müssen wir uns entscheiden, wo wir die drei Geräte ansiedeln wollen. Eine Möglichkeit ist in Abb. 3.56 dargestellt. Der EPROM belegt Adressen von 2K, der RAM belegt Adressen von 32K bis 34K, und der PIO belegt die höchsten 4 Byte 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 über den E/A-Raum zu adressieren, bräuchte er keine Speicheradressen (statt dessen aber vier E/A-Raumadressen).

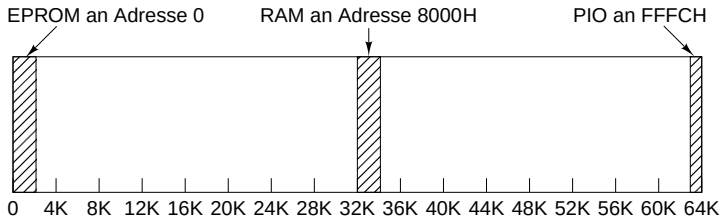


Abb. 3.56: Anordnung des EPROM, RAM und PIO in einem 64K-Adreßraum

Mit den Adreßzuweisungen von Abb. 3.56 sollte der EPROM von einer beliebigen 16-Bit-Speicheradresse im Format 00000xxxxxxxxxx (binär) ausgewählt werden. Anders ausgedrückt: Jede Adresse, deren fünf höherwertige Bits Nullen sind, fällt in die unteren 2K des Speichers, also in den EPROM. Somit könnte die EPROM-Chipauswahl mit einem 5-Bit-Komparator verdrahtet werden, dessen Eingänge fest auf 00000 verdrahtet sind.

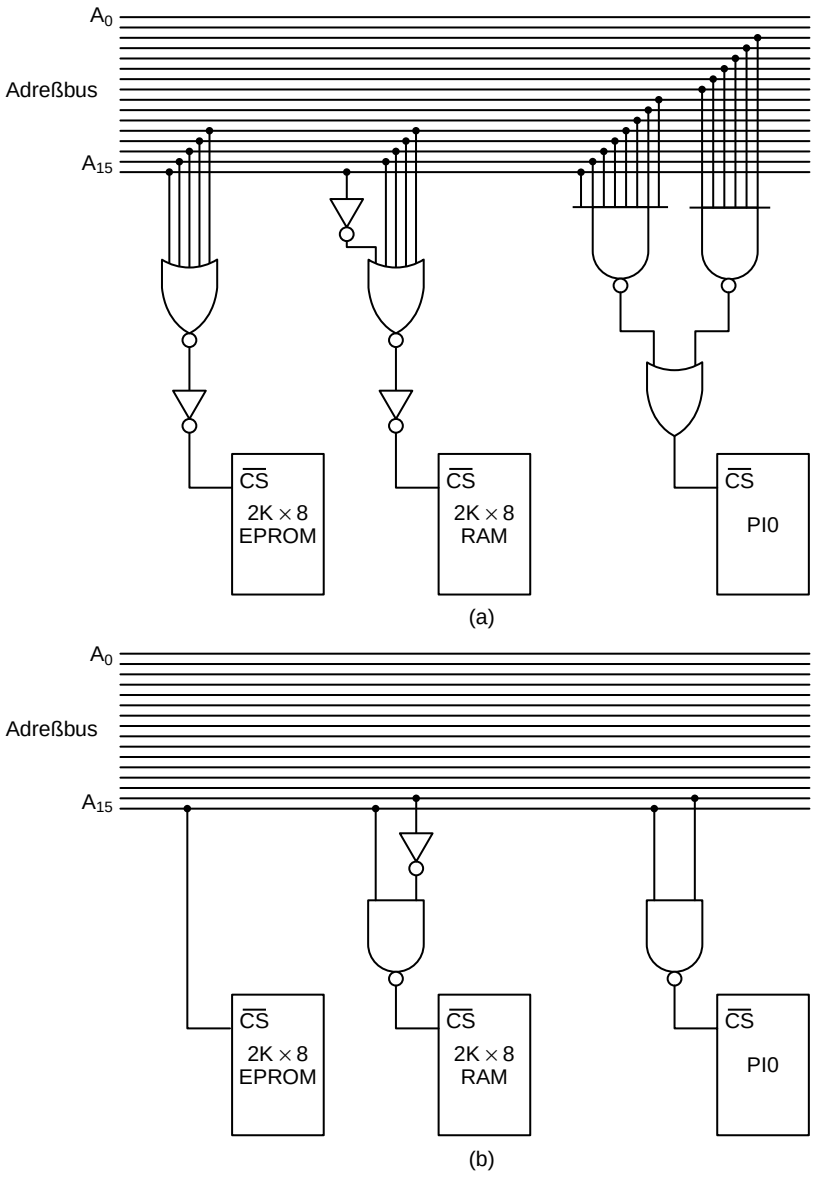


Abb. 3.57: (a) Volle Adreßdekodierung; (b) partielle Adreßdekodierung

Die gleiche Wirkung läßt sich besser erreichen, wenn man ein OR-Gate mit fünf Eingängen benutzt und die fünf Eingänge an die Adreßleitungen A11 bis A15 anschließt. Nur und nur wenn alle fünf Leitungen 0 sind, ist der Ausgang 0, so daß $\overline{CS}$ (auf Low) asseriert wird. Leider gibt es in der SSI-Standardserie kein OR-Gate mit fünf Eingängen. Das nächstliegende, das zur Verfügung steht, ist ein NOR-Gate mit acht Eingängen. Wenn wir drei Eingänge mit Masse verbinden und den Ausgang invertieren, können wir dennoch das richtige Signal herstellen, wie Abb. 3.57(a) zeigt. SSI-Chips sind so preiswert, daß es, von außergewöhnlichen Umständen abgesehen, kein Thema ist, wenn einmal einer ineffizient benutzt wird. Der Konvention zufolge werden unbenutzte Eingänge in Schaltdiagrammen nicht dargestellt.

Das gleiche Prinzip läßt sich auf den RAM anwenden. Er sollte aber auf Binäradressen im Format 10000xxxxxxxxxx reagieren, so daß ein zusätzlicher Inverter nötig ist (siehe die Abbildung). Die PIO-Adreßdekodierung ist komplizierter, weil sie von den vier Adressen im Format 11111111111111xx gewählt wird. Eine mögliche Schaltung, die $\overline{CS}$ nur dann asseriert, wenn die richtige Adresse im Adreßbus erscheint, ist in der Abbildung dargestellt. Sie hat zwei NAND-Gates mit acht Eingängen zur Versorgung eines OR-Gates. Um die in Abb. 3.57(a) dargestellte Logik zur Adreßdekodierung mit SSI zu realisieren, benötigt man sechs Chips – die vier Chips mit acht Eingängen, ein OR-Gate und einen Chip mit drei Invertern.

Falls der Computer wirklich nur aus der CPU, zwei Speicherchips und dem PIO besteht, können wir einen Trick anwenden, um die Adreßdekodierung enorm zu vereinfachen. Alle EPROM-Adressen, und nur EPROM-Adressen, haben im höherwertigen Bit A15 eine 0. Somit genügt es, $\overline{CS}$ direkt mit A15 zu verbinden, wie in Abb. 3.57(b) dargestellt.

An diesem Punkt mag die Entscheidung, den RAM auf 8000H zu setzen, viel weniger willkürlich erscheinen. Die RAM-Dekodierung kann durchgeführt werden, wenn nur gültige Adressen im Format 10xxxxxxxxxxxxx im RAM sind, so daß eine 2-Bit-Dekodierung ausreicht. Jede mit 11 beginnende Adresse muß eine PIO-Adresse sein. Die vollständige Dekodierlogik sind jetzt zwei NAND-Gates und ein Inverter. Da man aus einem NAND-Gate einen Inverter machen kann, wenn man einfach die beiden Eingänge miteinander verbindet, ist ein einzelner Quad-NAND-Chip mehr als ausreichend.

Die in Abb. 3.57(b) dargestellte Logik zur Adreßdekodierung heißt **partielle Adreßdekodierung** (**Partial Address Decoding**), weil die vollen Adressen nicht benutzt werden. Sie hat die Eigenschaft, daß eine Leseoperation von den Adressen 0001000000000000, 0001100000000000 oder 0010000000000000 zum gleichen Ergebnis führt. Jede Adresse in der unteren Hälfte des Adreßraums wählt den EPROM. Da die zusätzlichen Adressen nicht benutzt 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.

Bei einer weiteren üblichen Adreßdekodierungstechnik wird ein Dekodiererchip verwendet, z.B. wie in Abb. 3.13. Durch Verbindung der drei Eingänge mit den drei höherwertigen Adreßleitungen erhalten wir acht Ausgänge, entsprechend den Adressen in den ersten 8K, den zweiten 8K usw. Für einen Computer mit acht RAMs mit Speicherchips von je 8K × 8 bietet ein solcher Chip die gesamte Dekodierung. Für einen Computer mit acht Speicherchips von 2K × 8 reicht auch ein Dekodierer, vorausgesetzt, daß die Speicherchips jeweils an getrennten 8K-Stücken des Adreßraums angeordnet sind. (Man erinnere sich an die frühere Bemerkung, daß die Position der Speicher- und E/A-Chips innerhalb des Adreßraums eine Rolle spielt.)

3.8 Zusammenfassung

Computer werden aus integrierten Schaltchips gebaut, die winzige Schaltelemente enthalten, die man Gates nennt. Die üblichen Gates sind AND, OR, NAND, NOR und NOT. Einfache Schaltungen können direkt durch Kombination einzelner Gates mit Hilfe von SSI-Chips realisiert werden.

Komplexere Schaltungen werden mit Hilfe von MSI-Standardkomponenten gebaut, z.B. Multiplexer, Demultiplexer, Kodierer, Dekodierer, Schieber und ALUs. Mit einem PLA können beliebige boolesche Funktionen programmiert werden. Werden viele boolesche Funktionen benötigt, sind PLAs viel effizienter. Die Gesetze der booleschen Algebra lassen sich anwenden, um Schaltungen von einer Form in eine andere umzuwandeln. In vielen Fällen können auf diese Weise wirtschaftlichere Schaltungen produziert werden.

Computer-Arithmetik wird mittels Addierern durchgeführt. Ein Einzelbit-Volladdierer kann aus zwei Halbaddierern gebildet werden. Ein Addierer für ein Multibit-Wort kann gebildet werden, wenn man mehrere Volladdierer so zusammenschließt, daß der Übertrag jedes einzelnen in den jeweiligen linken Nachbarn eingespeist wird.

Die Komponenten von (statischen) Speichern sind Latches und Flip-Flop-Schaltungen, die jeweils ein Bit an Informationen speichern können. Diese Schaltungen können linear zu oktalen Latches und Flip-Flop-Schaltungen oder logarithmisch zu wortorientierten Speichern kombiniert werden. Speicher sind als RAMs, ROMs, PROMs, EPROMs, EEPROMs und Flashs erhältlich. Statische RAMs brauchen nicht aufgefrischt zu werden. Sie behalten ihre Werte, solange der Strom fließt. Dynamische RAMs müssen demgegenüber periodisch aufgefrischt werden, damit der Kriechverlust der kleinen Kondensatoren auf dem Chip kompensiert wird.

Die Komponenten eines Computer-Systems werden über Busse verbunden. Viele, aber nicht alle Pins auf einem typischen CPU-Chip steuern eine Busleitung direkt. Die Busleitungen können in Adreß-, Daten- und Steuerleitungen unterteilt werden. Synchrone Busse werden von einem Mastertakt gesteuert. Asynchrone Busse wenden zur Synchronisation des Slave mit dem Master volles Handshaking an.

Der Pentium II ist ein Beispiel einer modernen CPU. Die meisten damit ausgestatteten Systeme haben einen Speicherbus, einen PCI-Bus, einen ISA-Bus und einen USB-Bus. Der PCI-Bus kann je 64 Bit in einer Rate von 66 MHz übertragen, so daß er ausreichend schnell für fast alle Peripheriegeräte, aber nicht schnell genug für Speicher ist.

Schalter, Lampen, Drucker und viele andere E/A-Geräte können über parallele E/A-Chips, z.B. den 8255A, an Computer angeschlossen werden. Diese Chips können als Teil des E/A-Raums oder des Speicherraums konfiguriert werden. Man kann sie je nach Anwendung voll oder teilweise kodieren.

3.9 Aufgaben

1. Ein Logiker fährt in ein Drive-in-Restaurant und sagt: »Ich möchte einen Hamburger oder einen 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? (Im Englischen hat das Wort »or« [oder] die Bedeutung von »exclusive or« [ausschließliches oder].)

   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 bleibt zwar klug, fährt aber hungrig von dannen.

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. Es gibt vier boolesche Funktionen einer einzigen Variablen und 16 Funktionen von zwei Variablen. Wie viele Funktionen von drei Variablen gibt es? Wie viele gibt es von n Variablen?

4. Benutzen Sie eine Wahrheitstabelle, um folgendes zu zeigen: P = (P AND Q) OR (P AND NOT Q).

5. Zeigen Sie, wie die AND-Funktion aus zwei NAND-Gates gebildet werden kann.

6. Benutzen Sie die De Morganschen Gleichungen, um das Komplement von $A\overline{B}$ zu ermitteln.

7. Benutzen Sie den auf drei Variablen basierten Multiplexer-Chip von Abb. 3.12, und implementieren Sie eine Funktion, deren Ausgabe die Parität der Eingaben ist, d.h., die Ausgabe ist 1, falls und nur falls eine ungerade Zahl von Eingaben 1 sind.

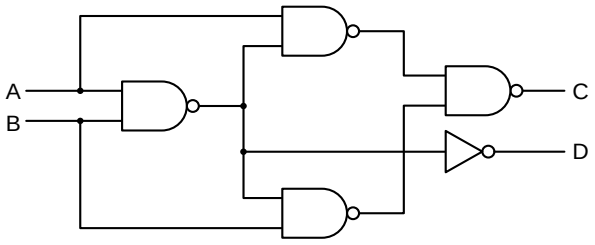
8. Denken Sie mal scharf nach: Der auf drei Variablen basierte Multiplexer-Chip von Abb. 3.12 kann eine beliebige Funktion von *vier* booleschen Variablen berechnen. Beschreiben Sie, wie, und zeichnen Sie als Beispiel das Logikdiagramm für die Funktion, die 0 ist, falls das englische Wort für die Zeile in der Wahrheitstabelle eine gerade Zahl von Buchstaben hat, und 1 ist, falls sie eine ungerade Zahl von Buchstaben hat (z.B. 0000 = Null = 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 V_{CC}, Masse, D oder $\overline{D}$ verdrahtet werden.

9. Zeichnen Sie das Logikdiagramm eines 2-Bit-Demultiplexers, d.h. eine Schaltung, deren einzige Eingangsleitung eine der vier Ausgangsleitungen ansteuert, je nach Zustand der beiden Steuerleitungen.

10. Zeichnen Sie das Logikdiagramm eines 2-Bit-Kodierers, d.h. eine Schaltung mit vier Eingangsleitungen, von denen genau eine zu einem Zeitpunkt auf High liegt, und zwei Ausgangsleitungen, deren 2-Bit-Binärwert aussagt, welcher Eingang auf High liegt.

11. Zeichnen Sie das PLA von Abb. 3.15 ausreichend detailliert, um darzulegen, wie die logische Mehrheitsfunktion von Abb. 3.3 implementiert werden kann. Achten Sie insbesondere darauf, daß Sie darstellen, welche Verbindungen in beiden Matrizen vorhanden sind.

12. Was macht diese Schaltung?



13. Ein üblicher MSI-Chip ist ein 4-Bit-Addierer. Vier dieser Chips können untereinander verbunden werden, um einen 16-Bit-Addierer zu bilden. Wie viele Pins hat der 4-Bit-Addiererchip erwartungsgemäß? Warum?

14. Ein n-Bit-Addierer kann realisiert werden, wenn man n Volladdierer in Reihe kaskadiert, wobei der Übertrag in Phase i, C_i von der Ausgabe von Phase $i-1$ kommt. Der Übertrag in Phase 0, C_0 ist 0. Wenn jede Phase

T ns dauert, um ihre Summe und den Übertrag zu produzieren, ist der Übertrag in Phase i erst iT ns nach dem Beginn der Addition gültig. Ist n groß, kann die Zeit, bis der Übertrag die höherwertige Phase durchlaufen hat, zu lang sein. Entwerfen Sie einen Addierer, der schneller arbeitet. Hinweis: Jeder C_i kann in Operandenbits A_{i-1} und B_{i-1} sowie als Übertrag C_{i-1} ausgedrückt werden. Mit Hilfe dieser Relation ist es möglich, C_i als Funktion der Eingaben in den Phasen 0 bis $i-1$ auszudrücken, so daß alle Überträge gleichzeitig erzeugt werden können.

15. Angenommen, alle Gates von Abb. 3.19 haben eine Verbreitungsverzögerung von 10 ns, und alle anderen Verzögerungen können ignoriert werden. Wann ist der früheste Zeitpunkt, an dem eine auf diesem Design basierte Schaltung sicher sein kann, daß sie ein gültiges Ausgabebit hat?

16. Die ALU von Abb. 3.20 kann 8-Bit-2-Komplement-Additionen ausführen. Ist sie auch in der Lage, 2-Komplement-Subtraktionen auszuführen? Falls ja, erklären Sie, wie. Andernfalls ändern Sie die ALU entsprechend ab, so daß sie auch Subtraktionen bewältigen kann.

17. Zuweilen muß eine 8-Bit-ALU, wie die in Abb. 3.20, die Konstante –1 als Ausgabe erzeugen. Führen Sie zwei Möglichkeiten auf, wie dies bewältigt werden kann. Für jede geben Sie die Werte der sechs Steuersignale an.

18. Eine 16-Bit-ALU wird aus 16 1-Bit-ALUs gebaut, die je 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?

19. Wie lautet der Wartezustand der Eingänge S und R in eine SR-Latch, die aus zwei NAND-Gates gebaut wurde?

20. Die Schaltung von Abb. 3.26 ist eine Flip-Flop-Schaltung, die an der steigenden Flanke des Takts ausgelöst wird. Ändern Sie die Schaltung so ab, daß eine Flip-Flop-Schaltung entsteht, die an der fallenden Flanke des Takts ausgelöst wird.

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 potentiell wichtigen Kunden einen Chip mit vier D-Flip-Flop-Schaltungen entwerfen soll, die jeweils Q und $\overline{Q}$ enthalten. Gemäß Kundenspezifikation sind im fraglichen Design alle vier Taktsignale verbunden. Es gibt weder Preset noch Clear. Ihre Aufgabe ist es, eine professionelle Einschätzung des Designs zu geben.

22. Der 4×3-Speicher von Abb. 3.29 nutzt 22 AND-Gates und drei OR-Gates. Angenommen, die Schaltung wird auf 256×8 erweitert. Wie viele von beiden würde man benötigen?

23. Je mehr Speicher aus einem einzigen Chip herausgequetscht wird, um so höher auch die Anzahl der für die Adressierung benötigten Pins. Oft ist

eine große Zahl von Adreßpins an einem Chip unpraktisch. Denken Sie sich eine Möglichkeit aus, um 2^n Speicherwörter mit weniger als n Pins zu adressieren.

24. Ein Computer mit einem 32 Bit breiten Datenbus benutzt dynamische RAM-Speicherchips von 1M × 1. Welche Größe (in Byte) ist der kleinste Speicher, den dieser Computer haben kann?

25. Unter Bezugnahme auf das Taktdiagramm in Abb. 3.37 nehmen wir an, daß Sie den Taktgeber um 40 ns statt um 25 ns (wie dargestellt) bei gleichen Taktbeschränkungen verlangsamt haben. Wieviel Zeit hätte der Speicher im schlechtesten Fall, um die Daten während T_3 nach der Assertion von $\overline{\text{MREQ}}$ auf den Bus zu bringen?

26. Noch einmal zurück zu Abb. 3.37. Angenommen, der Taktgeber bleibt bei 40 MHz, aber T_{AD} wird auf 16 ns erhöht. Könnten 40 ns-Speicherchips immer noch verwendet werden?

27. In Abb. 3.37(b) ist T_{ML} auf mindestens 6 ns spezifiziert. Können Sie sich einen Chip vorstellen, bei dem sie negativ ist? Anders ausgedrückt: Könnte die CPU $\overline{\text{MREQ}}$ asserieren, bevor die Adresse stabil ist? Warum bzw. warum nicht?

28. Angenommen, der Blocktransfer von Abb. 3.41 wird mit dem Bus von Abb. 3.37 durchgeführt. Wieviel mehr Bandbreite erhält man, wenn man statt einzelner Transfers 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 von neuem.

29. Beschriften Sie die Übergangszeiten der Adreßleitungen von Abb. 3.38 mit T_{A1} und T_{A2} und die Übergangszeiten von $\overline{\text{MREQ}}$ mit T_{MREQ1} und T_{MREQ2}. Schreiben Sie alle durch das volle Handshake implizierten Ungleichheiten auf.

30. Die meisten 32-Bit-Busse erlauben 16-Bit-Lese- und Schreiboperationen. Besteht Zweideutigkeit darüber, wo die Daten abgestellt werden? Erläutern Sie Ihre Antwort.

31. Viele CPUs verfügen über eine spezielle Buszyklusart für Interrupt-Bestätigungen (Acknowledge). Warum?

32. Ein 10-MHz-PC/AT erfordert vier Zyklen, um ein Wort zu lesen. Wieviel Busbandbreite verbraucht die CPU?

33. Eine 32-Bit-CPU mit den Adreßleitungen A2–A31 erfordert alle Speicherreferenzen für die Ausrichtung. Das heißt, Wörter müssen in Mehrfachen von 4 Byte und halbe Wörter in geraden Bytes adressiert werden. Bytes können sich an beliebiger Stelle befinden. Wie viele zulässige Kombinationen gibt es für Speicher-Lese-Operationen, und wie viele Pins sind erforderlich, um sie auszudrücken? Geben Sie zwei Antworten, und begründen Sie beide.

34. Warum kann der Pentium II nicht an einem 32-Bit-PCI-Bus arbeiten, ohne an Funktionsumfang einzubüßen? Schließlich bewältigen andere Computer mit einem 64-Bit-Datenbus 32 Bit, 16 Bit und sogar 8 Bit breite Transfers.

35. Angenommen, eine CPU hat einen Level-1-Cache und einen Level-2-Cache mit Zugriffszeiten von 5 bzw. 10 ns. Die Hauptspeicher-Zugriffszeit ist 50 ns. Angenommen, 20% der Zugriffe erfolgen auf den Level-1-Cache und 60% auf den Level-2-Cache. Welche durchschnittliche Zugriffszeit ergibt sich?

36. Ist es wahrscheinlich, daß ein kleines, eingebettetes System auf der Basis von picoJava II einen 8255A-Chip beinhaltet?

37. Berechnen Sie die nötige Busbreite, um einen Videofilm in 30 Rahmen/Sekunde (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.

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

39. Welches der Signale in Abb. 3.53 ist nicht unbedingt notwendig, damit das Busprotokoll funktioniert?

40. Ein Computer hat Instruktionen, die je zwei Buszyklen erfordern – eine zum Holen der Instruktion und eine zum Holen der Daten. Jeder Buszyklus dauert 250 ns und jede Instruktion 500 ns (d.h., die interne Verarbeitungszeit ist verschwindend gering). Der Computer hat auch eine Platte mit 16 512-Byte-Sektoren pro Spur. Die Umdrehungszeit der Platte beträgt 8,092 ms. Auf welchen Prozentsatz seiner normalen Geschwindigkeit wird der Computer während eines DMA-Transfers reduziert, wenn jeder DMA-Transfer einen Buszyklus dauert? Betrachten Sie die beiden Fälle: 8-Bit- und 16-Bit-Bustransfers.

41. Die maximale Nutzlast (Payload) eines isochronen Datenpakets am USB-Bus ist 1023 Byte. Nehmen wir an, daß ein Gerät nur ein Datenpaket pro Rahmen senden darf. Wie hoch ist die maximale Bandbreite für ein einzelnes isochrones Gerät?

42. Welche Wirkung würde sich bei dem PIO von Abb. 3.57(b) ergeben, wenn man eine dritte Eingangsleitung zum NAND-Gate hinzufügt? Und was passiert, wenn man diese neue Leitung an A13 anschließt?

43. Schreiben Sie ein Programm, welches das Verhalten eines $m \times n$-Arrays mit NAND-Gates mit zwei Eingängen simuliert. Diese auf einem Chip befindliche Schaltung hat j Eingangs- und k Ausgangspins. Die Werte von j, k, m und n sind Kompilierzeit-Parameter der Simulation. Das Programm sollte mit dem Lesen einer »Verdrahtungsliste« beginnen. Jeder darin enthaltene Draht spezifiziert einen Eingang und einen Ausgang. Ein Eingang ist entweder einer der j Eingangspins oder der Ausgang eines

NAND-Gates. Ein Ausgang ist entweder einer der k Ausgangspins oder ein Eingang in ein NAND-Gate. Unbenutzte Eingänge sind logisch 1. Nach dem Lesen der Verdrahtungsliste sollte das Programm den Ausgang für jeden der möglichen 2^j Eingänge drucken. Gate-Arraychips wie dieser sind in auftragsspezifischen Schaltungen auf einem Chip weit verbreitet, weil der Großteil der Arbeit (Aufbringen des Gate-Arrays auf den Chip) unabhängig von der zu implementierenden Schaltung ist. Nur die Verdrahtung ist designspezifisch.

44. Schreiben Sie ein Programm zum Einlesen von zwei willkürlichen booleschen Ausdrücken, und stellen Sie fest, ob sie die gleiche Funktion darstellen. Die Eingabesprache sollte einzelne Buchstaben als boolesche Variablen, die Operanden AND, OR und NOT und Klammern enthalten. Jeder Ausdruck sollte in eine Eingangsleitung passen. Das Programm sollte die Wahrheitstabellen für beide Funktionen berechnen und vergleichen.

45. Schreiben Sie ein Programm, das eine Sammlung von booleschen Ausdrücken einliest und die Matrizen 24×50 und 50×6 berechnet, die zur Implementierung mit dem PLA von Abb. 3.15 erforderlich sind. Die Eingabesprache sollte die gleiche sein wie in Aufgabe 44. Geben Sie die Matrizen auf einem Zeilendrucker aus.