- Kapitel 1 -
Kombinatorische Digitalschaltungen

Ausgabestand 1.1

- Nur zur Information -

1. Kombinatorische Digitalschaltungen

Wenn man von Grund auf entwickelt, so entsteht eine kombinatorische Schaltung typischerweise aus einer Entwurfsaufgabe heraus, die als Wahrheitstabelle, als Belegungsliste, als Boolesche Gleichung oder auch nur in umgangssprachlicher Formulierung vorliegt. Nicht selten ist es heutzutage so, daß der Entwickler gar keinen Schaltplan mehr zeichnet, sondern sich darauf beschränkt, die gewünschte Schaltfunktion mit formalen Mitteln zu beschreiben - den Rest erledigt die Entwurfssoftware (Stichwort: EDA = Electronic Design Automation).

Mit wirklich komplizierten kombinatorischen Schaltungen müssen wir uns nicht beschäftigen. Wir sollten uns aber - zumindest überblicksmäßig - in folgende Problemkreise einarbeiten:

- sehr elementare Schaltungsstrukturen - bis hin zum (gelegentlich trickreich eingesetzten) einzelnen Gatter in der Restlogik auch der ganz neumodischen Hardware,
- Schaltungsstrukturen, die wichtige Grundfunktionen der Computertechnik verwirklichen und die als vorgefertigte Bausteine immer wieder eingesetzt werden. Sie gibt es u. a. in Form von Bauelementen mittleren Integrationsgrades, die den Charakter von Industriestandards haben.
- Verarbeitungsschaltungen, die - beispielsweise im Prozessor - wirklich "Leistung" (im Sinne der beliebten MIPS) erbringen. Hier müssen wir nicht die letzten Einzelheiten kennen. Wir sollten aber wissen, wie solche Schaltungen im Prinzip funktionieren und welche grundsätzlichen Zusammenhänge es gibt.

1.1. Grundschaltungen

1.1.1. Grundgatter

Grundgatter sind logische Funktionselemente, die in Gatterschaltkreisen oder als elementare Zellenstrukturen (in programmierbaren und anwendungsspezifischen Schaltkreisen) verfügbar sind. Kennzeichnende *funktionelle* Merkmale sind die jeweilige logische Verknüpfung und die Anzahl der Eingänge. Um die Anzahl der Eingänge zu bezeichnen, spricht man z. B. von einem Zweifach-NAND, einem Vierfach-NOR usw. Zu den *technischen* Merkmalen gehören u. a. die Verzögerungszeit, die Speisespannung, die Stromaufnahme sowie die Lastkennwerte der Ein- und Ausgänge.

1.1.2. Schmitt-Trigger-Eingänge

Signaländerungen an Digitalschaltkreisen müssen eine vorgeschriebene Mindest-Flankensteilheit aufweisen. Nun haben nicht alle Signale, die eine Digitalschaltung aus ihrer Umgebung bezieht, Flankensteilheiten, die den jeweiligen Anforderungen entsprechen. Um aus beliebig langsamen ("lahmen") Signalverläufen am Eingang exakte, steile Impulsflanken am Ausgang zu machen, braucht man Schaltungen, die ein sog. Schwellwertverhalten zeigen. Unterschreitet das Eingangssignal einen vorgegebenen Schwellwert (Threshold), so führt der Ausgang beispielsweise Low-Signal, überschreitet es ihn, schaltet der Ausgang mit hoher Flankensteilheit auf High um. Die ausgangsseitige Flankensteilheit ist dabei unabhängig von der eingangsseitigen.

Eine einfache, kostengünstige Grundschaltung, die ein solches Verhalten aufweist, ist der sog. *Schmitt-Trigger*. Es gibt entsprechende Digitalschaltkreise in Form von Gattern und Negatoren. Auch manche höher integrierte Schaltkreise haben an bestimmten Eingängen (wo dies anwendungspraktisch sinnvoll ist), Schmitt-Trigger-Stufen (Schmitt Trigger Inputs).

Schmitt-Trigger sind in der Lage, beliebig langsame Eingangssignale in Logiksignale mit angemessen steilen Flanken zu wandeln, es sind aber keine Präzisionsbauelemente (die Schwellwerte haben beachtliche Toleranzen). Sind präzise Schwellwerte einzuhalten, so sind Comparatoren einzusetzen.

1.1.3. Binäre und Tri-State-Ausgänge

Binäre Ausgänge
Die typische Digitalschaltung arbeitet mit zwei Signalwerten. Demgemäß verhalten sich auch die Ausgänge: von Umschaltvorgängen (Signalflanken) abgesehen, führen sie entweder einen Low-Pegel oder einen High-Pegel (binäre bzw. zweiwertige Ausgänge).

Tri-State-Ausgänge
Es bleibt bei den zwei Signalwerten. Solche Ausgänge (sprich: Trieh Steht ...) können aber einen weiteren (den dritten) Zustand einnehmen (Abbildung 1.1). In diesem Zustand liefern sie praktisch nichts - sie schalten überhaupt keinen Signalpegel auf (ein typischer Fachbegriff: der Ausgang ist "hochohmig" - weil er einem extrem hohen elektrischenWiderstand entspricht). Um das Schaltverhalten umzusteuern*), sind solche Stufen mit einem zusätzlichen Erlaubnissignal beschaltet (engl. Enable; sprich: Ennebl).

*): zwischen binärer Arbeitsweise und hochohmigem Ausgang.

Der Zweck: schaltet die eine Stufe keinen Signalpegel auf, so gibt sie anderen Stufen Gelegenheit zum Aufschalten. Auf diese Weise können mehrere Ausgänge auf einen einzigen Signalweg arbeiten. Die wichtigste Anwendung: Bussysteme.

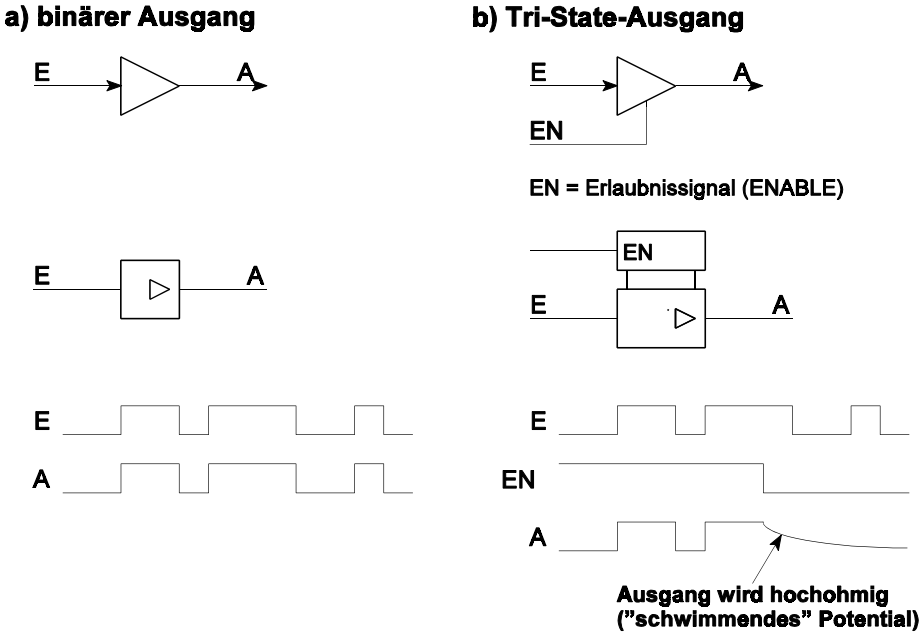


Abbildung 1.1 Binäre und Tri-State-Ausgänge. Erklärung des Schaltverhaltens. Oben: herkömmliche Schaltsymbole, darunter: Schaltsymbole gemäß DIN 40 900

1.1.4. Wechselseitige Wandlungen

Grundsätzlich reicht ein einziger Gattertyp aus, um jede beliebige Schaltfunktion zu verwirklichen, vorausgesetzt, es handelt sich um eine UND- bzw. ODER-Verknüpfung gefolgt von einer Negation (vollständige Realisierungsbasis). Diese Tatsache wird technisch ausgenutzt, um mit wenigen Gattertypen auszukommen, manchmal sogar (namentlich in ASICs) mit einem einzigen (z. B. mit einem Zweifach-NAND). Abbildung 1.2 zeigt, wie man mit NAND- und NOR-Gattern verschiedene Funktionen verwirklichen kann. Darüber hinaus ist die wechselseitige Wandlung von NAND und NOR dargestellt.

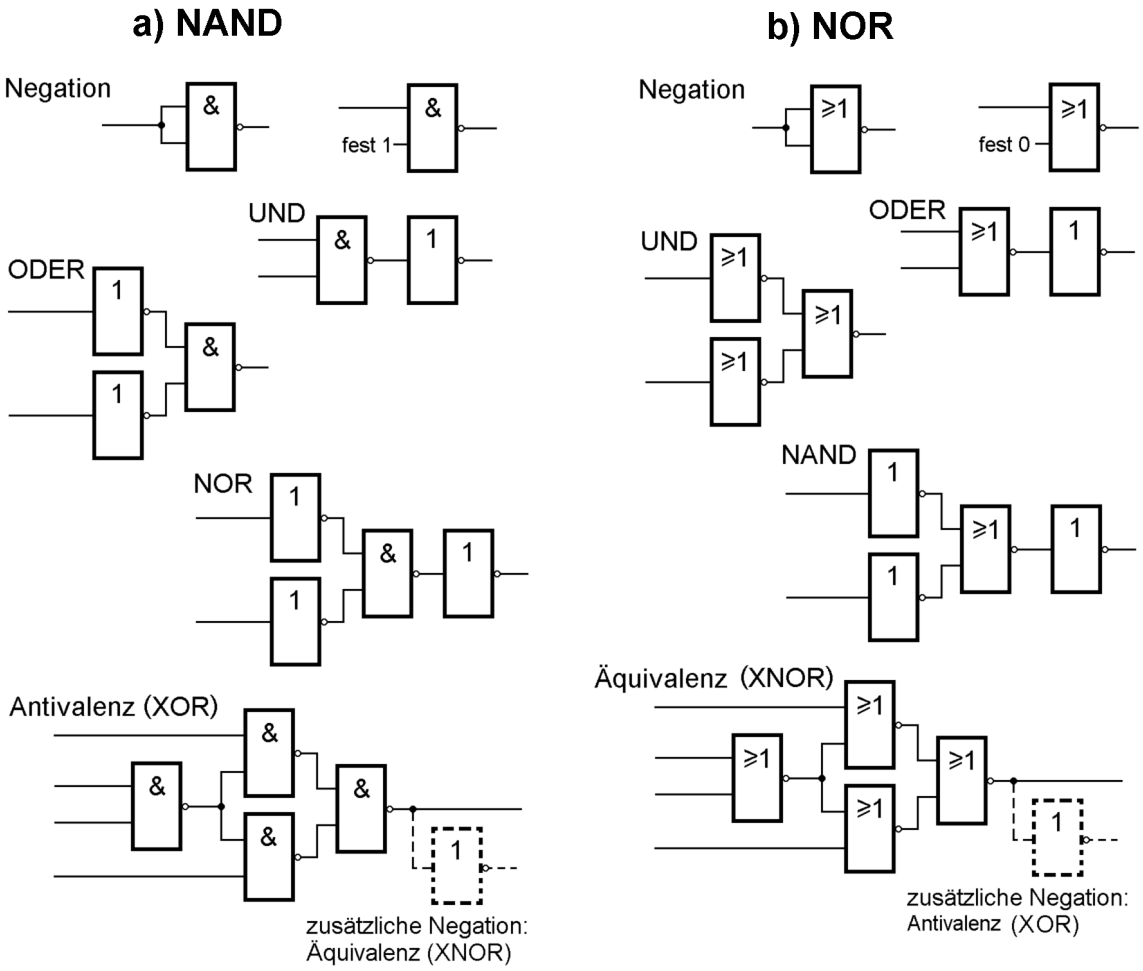


Abbildung 1.2 Funktionen aus Grundgattern (Beispiele) und wechselseitige Wandlungen von Gatterfunktionen. a) NAND, b) NOR als Realisierungsbasis. Darunter jeweils andere Funktionen, die sich damit realisieren lassen

Positive und negative Logik; NAND und NOR
Ein NAND in positiver Logik entspricht einem NOR in negativer Logik und umgekehrt (*Dualität* von NAND und NOR als Konsequenz der DeMorganschen Regeln.

1.1.5. Kaskadierung

Kaskadierung bedeutet hier, aus Schaltungen mit vergleichsweise wenigen Eingängen funktionell gleichartige Schaltungen mit entsprechend mehreren Eingängen aufzubauen.

Kaskadierung von Grundgattern
Oftmals braucht man Verknüpfungen, wie UND, ODER, NAND, NOR usw. mit mehr als zwei Eingängen. So etwas läßt sich durch entsprechende elektrische Auslegung der Gatter erreichen. Es gelingt aber auch, Gatter mit mehr als zwei Eingängen aus Gattern mit zwei Eingängen und (falls erforderlich) aus Negatoren aufzubauen (Abbildungen 1.3, 1.4).

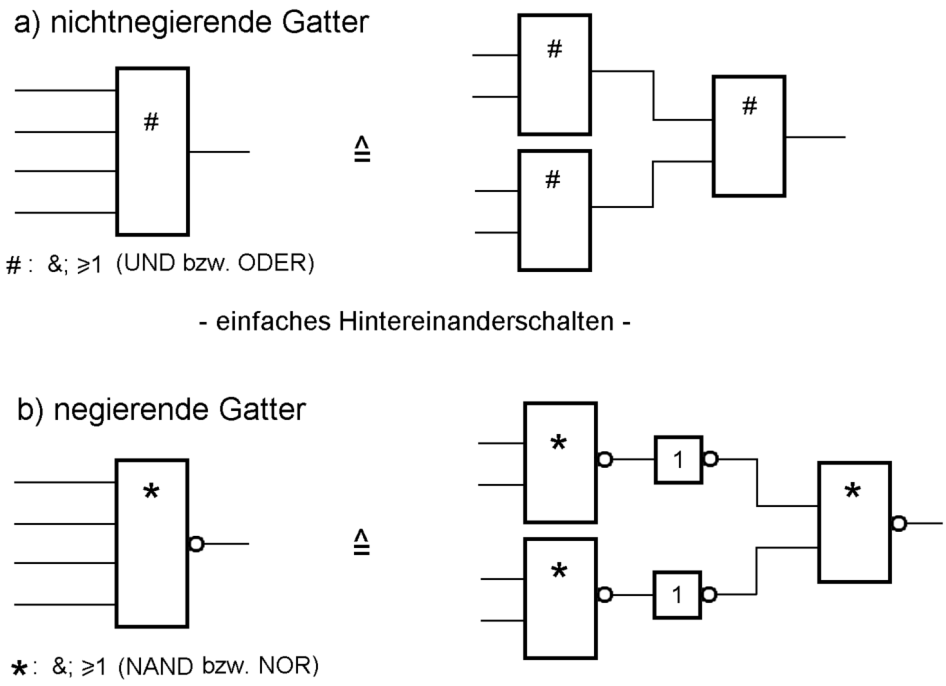


Abbildung 1.3 Kaskadierung

Erklärung:
Die Abbildung zeigt, wie aus Gattern mit 2 Eingängen gleichartige Verknüpfungen mit 4 Eingängen aufgebaut werden können. Nichtnegierende Gatter kann man unmittelbar hintereinanderschalten. Handelt es sich um negierende Gatter, sind jeweils Negatoren einzufügen. Grundsätzlich lassen sich Gatter mit beliebiger Eingangszahl durch fortgesetztes Kaskadieren aufbauen.

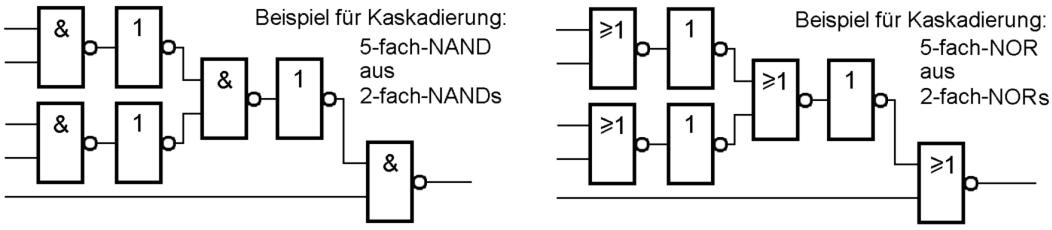


Abbildung 1.4 Kaskadierungsbeispiele: 5-fach-NAND und 5-fach-NOR

Wieviele Gatter braucht man wirklich?
Sehen wir uns Abbildung 1.4 genauer an, so ist erkennbar, daß eine vorgegebene Anzahl Grundgatter mit wenigen Eingängen schnell aufgebraucht sein wird. Dies ist besonders dann von Bedeutung, wenn man FPGA- oder ASIC-Technologien nutzen will. Da mögen beispielsweise mehrere tausend Zweifach-NANDs zunächst viel erscheinen. Das ist aber die einzige Grundlage, aus der alle anderen Verknüpfungen aufgebaut werden müssen!

Mehrfachkaskadierung
Sind kombinatorische Verknüpfungen mit vielen Eingängen durch Kaskadieren zu verwirklichen, so hat man die Wahl zwischen 2 grundsätzlichen Strukturen (Abbildung 1.5).

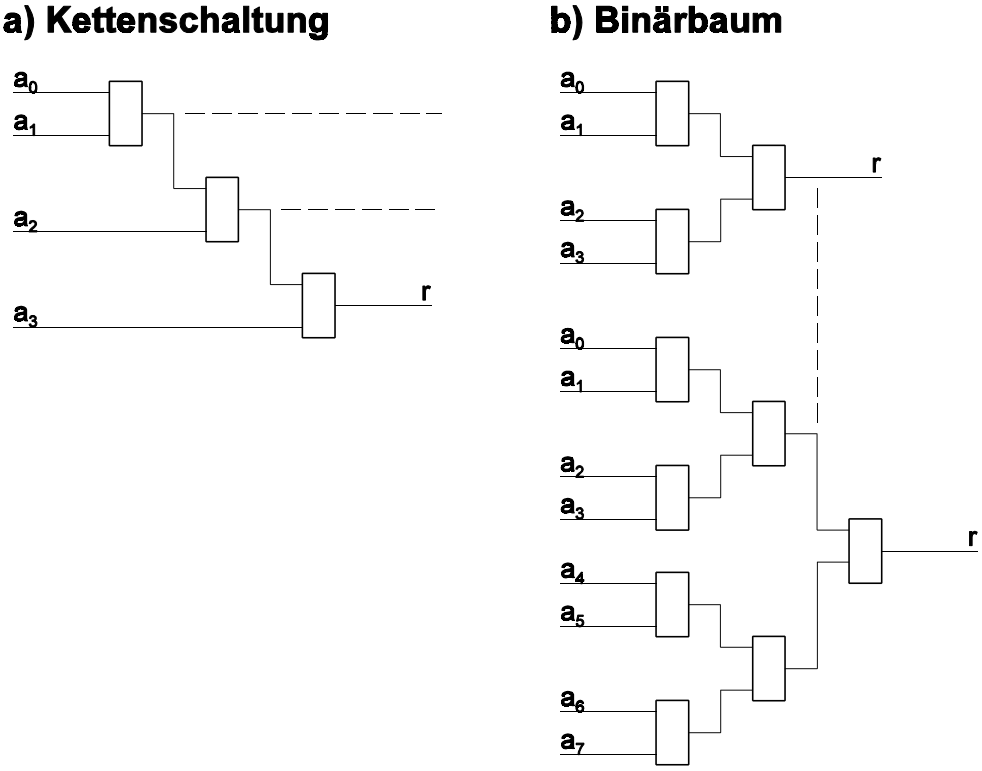


Abbildung 1.5 Mehrfachkaskadierung

Erklärung:

a) Kettenschaltung (Daisy Chain bzw. Ripple Thru),
b) Binärbaum (Binary Tree). 2 Beispiele; oben für 4, unten für 8 Variable.

Wir nehmen hierbei an, daß die kaskadierte Struktur mit gleichartigen Gattern aufgebaut wird. Handelt es sich um negierende Gatter, so nehmen wir an, jedem Gatter sei ein - in der Abbildung nicht dargestellter - Negator nachgeschaltet (den wir einfach zum Gatter hinzurechnen).

Ersichtlicherweise erfordert es stets n - 1 Gatter mit 2 Eingängen, um ein Gatter mit n Eingängen aufzubauen. Worin sich beide Strukturen wesentlich unterscheiden, ist die Schaltungstiefe (und damit die Schaltverzögerungszeit):

- die Schaltungstiefe s der Kettenschaltung wächst linear mit der Eingangszahl, also gemäß O(n). Bei n Eingängen ist $s = n - 1^{*)}$.
- die Schaltungstiefe s der Baumstruktur wächst gemäß dem Zweierlogarithmus der Eingangszahl, also gemäß O(ld n). Bei n Eingängen ist $s = \text{ceil}(\text{ld } n)^{*)}$.

*): ohne Berücksichtigung technischer Nebenbedingungen (zwischenzuschaltende Negatoren, ggf. erforderliche Treiberstufen usw.).

Damit ist - was die Geschwindigkeit angeht - die Baumstruktur der Kettenschaltung grundsätzlich überlegen, und zwar umso mehr, je größer die Eingangszahl ist. Beispiel: Bei 32 Eingängen ist bei Wahl der Kettenschaltung s = 31, bei Wahl der Baumstruktur s = 5.

Die Probleme der Baumstruktur zeigen sich aber dann, wenn Folgen gleichartiger Verknüpfungen über gemeinsame Eingangssignale zu bilden sind. Genauer: es sind mehrere Ausgangssignale zu bilden, und das jeweils nachfolgende Ausgangssignal hängt vom jeweils vorhergehenden ab (typische Beispiele sind Prioritätsnetzwerke, die Übertragsweitergabe beim Addieren, Zählnetzwerke usw.). Derartige funktionelle Abhängigkeiten legen eine Kettenstruktur nahe (in Abbildung 1.5a durch die gestrichelten Linien angedeutet). Mit dem ersten Netzwerk bilden wir das erste Ausgangssignal. Dieses führen wir dem zweiten Netzwerk zu, um das zweite Ausgangssignal zu bilden usw. Der Vorteil: geringer Aufwand, da die jeweils vorhergehenden Gatter für die nachfolgenden Verknüpfungen mitgenutzt werden. Der Nachteil: mit zunehmender Verarbeitungsbreite gemäß O(n) zunehmende Schaltungstiefe.

Der Ausweg liegt nahe: wir bilden alle Ausgangssignale unabhängig voneinander, und setzen dazu Gatter mit entsprechend vielen Eingängen ein - die wir ggf. als Baumstrukturen aus Grundgattern mit beschränkter Eingangszahl aufbauen. Die Schaltungstiefe wächst somit nur gemäß O(ld n). Aber: der Aufwand wächst - wie wir gleich sehen werden - gemäß $O(n^2)$. Und gegen quadratisch wachsende Komplexität hilft letzten Endes auch die neueste Schaltkreistechnologie nicht ... (Selbst wenn man sich die Gatter durchaus leisten könnte: viele Gatter bedeuten auch lange Verbindungswege, eine hohe kapazitive Belastung usw., so daß womöglich der riesige Schaltungsaufwand gar nicht im Sinne der angestrebten Schaltzeitverkürzung wirksam werden kann.)

Beispiel: aus 32 Variablen $a_{31}...a_0$ sind 31 Ergebnisbits zu bilden (Bit 1 durch Verknüpfung von a_0 und a_1, Bit 2 durch Verknüpfung von Bit 1 mit a_2 usw.).

- die Kettenschaltung (gemäß Abbildung 1.5a) kommt mit 31 Gattern aus. Für Bit 31 ergibt sich aber eine Schaltungstiefe s = 31.
- die Baumschaltung (gemäß Abbildung 1.5b) braucht hingegen für Bit 1 ein Gatter (2 Variable), für Bit 2 zwei Gatter (3 Variable) usw. Bit 31 erfordert somit 31 Gatter (32 Variable $a_0...a_{31}$). Insgesamt brauchen wir also 1 + 2 + 3 +... + 31 Gatter, also allgemein für n Variable*):

$$\frac{(n-1)^2 + n - 1}{2} \text{ Gatter.}$$

  Für 32 Variable ergeben sich somit 496 Gatter.

*): gemäß Summenformel der arithmetischen Reihe 1 + 2 + 3 +...+ (n-1).

Dieser Aufwand (Wachstum gemäß $O(n^2)$) setzt der Anwendbarkeit von Baumstrukturen Grenzen, so daß typischerweise Kompromißlösungen zwischen Baum- und Kettenstrukturen gewählt werden. Beispielsweise liegt es nahe, für die Verknüpfung von 8 Variablen in Abbildung 1.5b (unten) die Verknüpfung der ersten 4 Variablen (oben) mitzunutzen (vgl. die gestrichelte Linie; die untere Verknüpfung der Variablen $a^0...a^3$ könne somit entfallen). In der

Schaltungspraxis sind die Grenzen dieser Mitnutzung u. a. durch Leitungslängen, kapazitive Belastung und verfügbare Verdrahtungsressourcen gegeben.

1.1.6. Verknüpfungen in disjunktiver Normalform

UND-ODER
Es liegt nahe, die konjunktiven Verknüpfungen mit UND-Gattern und die disjunktive Verknüpfung mit einem nachgeschalteten ODER-Gatter zu realisieren (Abbildung 1.6).

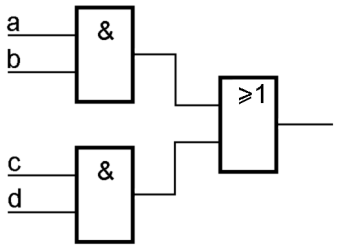


Abbildung 1.6 Verknüpfung in disjunktiver Normalform als UND-ODER-Struktur. Beispiel: ab ∨ cd

NAND-NAND
Sowohl die Konjunktionen als auch die Disjunktion lassen sich mit NAND-Gattern verwirklichen (Abbildung 1.7).

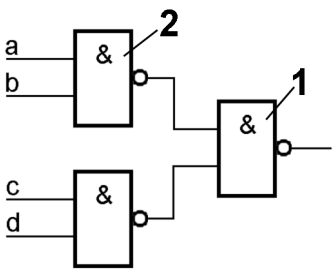


Abbildung 1.7 Verknüpfung in disjunktiver Normalform als NAND-NAND-Struktur. Beispiel: ab ∨ cd

Erklärung:
1 - disjunktive Verknüpfung: NAND wirkt als ODER für invertierte Signale (Ausgang = 1, falls wenigstens ein Eingang = 0); 2 - konjunktive Verknüpfung: NAND wirkt als UND, das das erforderliche negierte Ausgangssignal liefert (Ausgang = 0, falls alle Eingänge = 1, d. h. falls die UND-Bedingung erfüllt ist).

Hinweis:
Wir erkennen hieran, weshalb das NAND als Realisierungsbasis so beliebt ist (NAND - NAND ≙ UND - ODER).

NOR-NOR

Ein NOR wirkt als konjunktive Verknüpfung negierter Signale (Ausgang = 1, falls alle Eingänge = 0). Demgemäß kann man disjunktive Nomalformen auch mit NOR-Gattern aufbauen (Abbildung 1.8).

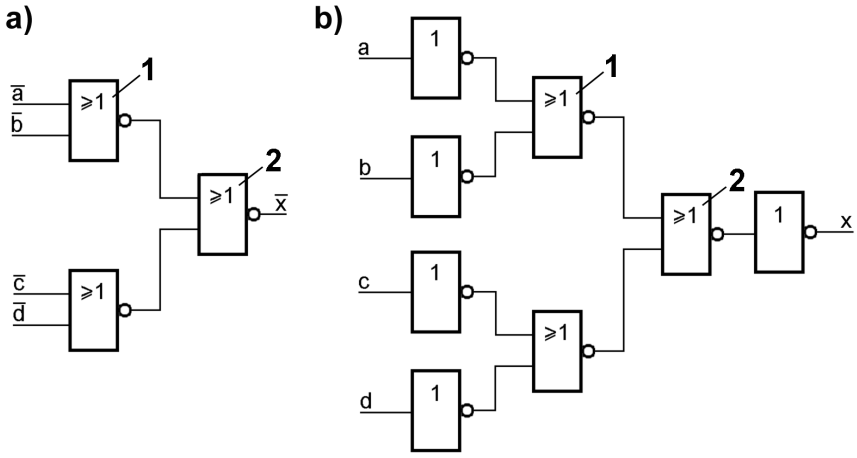


Abbildung 1.8 Verknüpfung in disjunktiver Normalform als NOR-NOR-Struktur. Beispiel: ab ∨ cd

Erklärung:

a) - zweistufiges Netzwerk mit invertierten (negierten) Ein- und Ausgängen*); b) - Netzwerk zur UND-ODER-Verknüpfung nichtnegierter Signale. 1 - konjunktive Verknüpfung der invertierten Eingangssignale (2. DeMorgansche Regel; $\overline{\bar{a} \vee \bar{b}} = a \cdot b$); 2 - disjunktive Verknüpfung, die wiederum ein invertiertes Ausgangssignal liefert.

Wie werden die negierten Signale gebildet?

- es werden - wie in Abbildung 1.8b gezeigt - Negatoren eingefügt,
- wir arbeiten grundsätzlich nur mit invertierten Signalen (Konventionssache - man muß sich eben von Anfang an darauf einstellen)**). Da die NOR-NOR-Anordnung ohnehin ein negiertes Ausgangssignal liefert, sind ggf. nur an den Schnittstellen zur Außenwelt zusätzliche Negatoren erforderlich.
- die Negation ergibt sich gleichsam umsonst durch entsprechendes Anschließen an die invertierten Ausgänge der vorgeschalteten Flipflops.

*): bei der Bestimmung der Schaltungstiefe kombinatorischer Netzwerke zählt die Negation von Eingangssignalen nicht mit (zumindest in der Theorie). In der Praxis ist diese Festlegung oft gerechtfertigt (z. B. beim Anschließen an Flipflopausgänge oder in programmierbaren Logikschaltungen (vgl. Abbildung 4.16)).

**): ein gern verwendeter Trick: man rechnet im Innern der Schaltung mit negativer Logik (Low = 1, High = 0). Somit werden - gemäß den DeMorganschen Regeln - die NORs zu NANDs, mit denen es sich bequemer entwirft (vgl. Abbildung 1.7).

1.1.7. Ausgewählte Schaltungstricks

Mehrstufige Netzwerke

Gelegentlich läßt sich der Schaltungsaufwand verringern, indem man von der disjunktiven Normalform bzw. vom zweistufigen Gatternetzwerk abgeht (Abbildung 1.9).

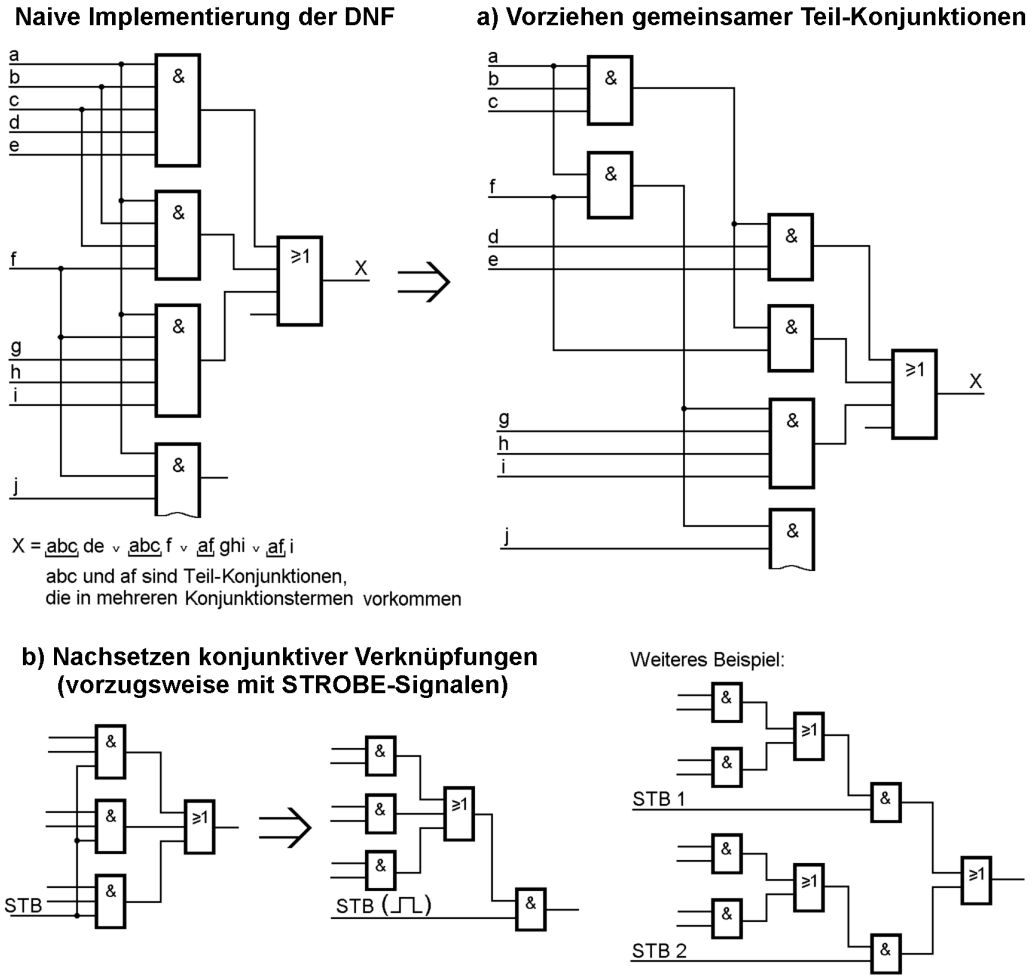


Abbildung 1.9 Optimierungstricks (eine kleine Auswahl)

Erklärung:

a) Vorziehen. Gleichartige Teil-Konjunktionen (Variablenbelegungen, die in mehreren Konjunktionstermen gemeinsam vorkommen), kann man in besonderen Gattern zusammenfassen, die dem zweistufigen Netzwerk vorgeschaltet werden.

b) Nachsetzen. Einzelne Variable, die in allen Konjunktionstermen gleichermaßen vorkommen, werden aus der zweistufigen Verknüpfung herausgenommen. Die erforderlichen konjunktiven Verknüpfungen werden ausgangsseitig nachgeschaltet. Das Nachsetzen ist im besonderen bei Signalen üblich, die als Takt- oder Strobe-Impulse wirken. Man hat dann die jeweils kürzeste Verzögerungszeit vom Impuls zum Ausgang.

Elementare Verknüpfungen invertiert wirkender (aktiv-Low-) Signale

Manche Signale sind als aktiv Low spezifiziert (das ist z. B. typisch für Schaltkreisauswahlsignale (Chip Enables, Chip Selects) von Treibern und Speichern, für Speicher-Schreibimpulse, für Rücksetzsignale usw.). Nicht selten ist es erforderlich, elementare Verknüpfungen derartiger Signale zu realisieren. Hierzu lassen sich die DeMorgan'schen Regeln vorteilhaft ausnutzen (Abbildung 1.10).

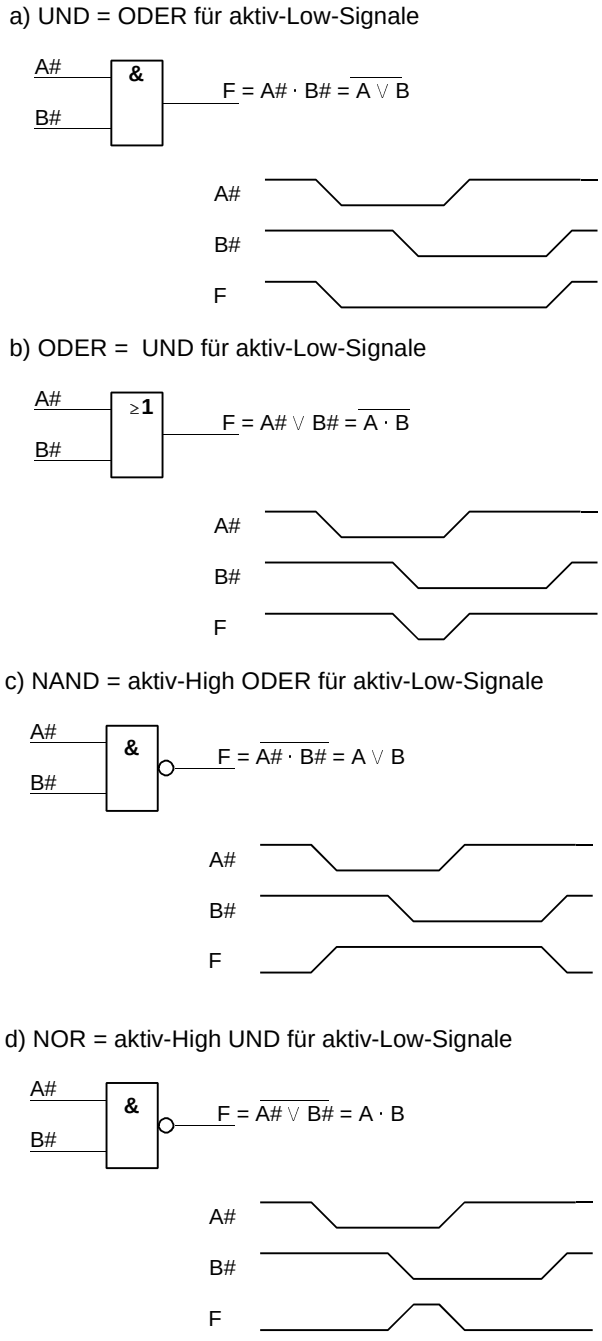


Abbildung 1.10 Elementare Verknüpfungen von Signalen, die aktiv Low wirken

Erklärung zu Abbildung 1.10:

a) ein UND-Gatter wirkt als ODER: der Ausgang wird bereits dann aktiv (Low), sobald auch nur einer der Eingänge aktiv (Low) ist,

b) ein ODER-Gatter wirkt als UND: nur dann, wenn beide Eingänge aktiv (Low) sind, wird der Ausgang aktiv (Low),

c) ein NAND-Gatter wirkt als ODER mit aktiv-High-Ausgang für aktiv-Low-Eingangssignale. Anwendungsbeispiel: ODER-Verknüpfung, wenn nur invertierte Signale vorliegen (billiger als 2 · Negator + ODER-Gatter).

d) ein NOR-Gatter wirkt als UND mit aktiv-High-Ausgang für aktiv-Low-Eingangssignale. Anwendungsbeispiel: UND-Verknüpfung, wenn nur invertierte Signale vorliegen (billiger als 2 · Negator + UND-Gatter).

Das XOR als steuerbarer Inverter/Noninverter
Ein XOR-Gatter kann als steuerbarer Inverter eingesetzt werden (Abbildung 1.11).

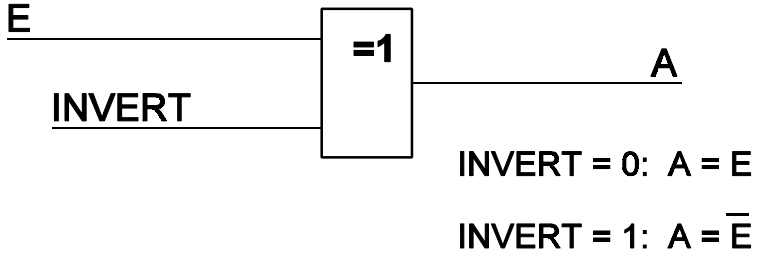


Abbildung 1.11 XOR als steuerbarer Inverter

Erklärung:
In den Signalweg (von E nach A) wird ein XOR geschaltet. INVERT = 0: Signal wird nicht negiert: 0 am Eingang wird zu 0 am Ausgang ($0 \oplus 0 = 0$), 1 am Eingang wird zu 1 am Ausgang ($1 \oplus 0 = 1$). INVERT = 1: Signal wird negiert: 0 am Eingang wird zu 1 am Ausgang ($0 \oplus 1 = 1$), 1 am Eingang wird zu 0 am Ausgang ($1 \oplus 1 = 0$).

1.2. Elementarfunktionen

1.2.1. Auswählen

Der Datenselektor
Die Aufgabe besteht darin, einen von mehreren Eingängen zu einem Ausgang durchzuschalten. Das heißt: am Ausgang muß eine Eins erscheinen, wenn der betreffende Eingang eine Eins führt UND wenn er ausgewählt ist. Man muß also jeden Eingang mit einem Auswahl-Steuersignal konjunktiv verknüpfen und alle diese Verknüpfungen disjunktiv zusammenfassen ("odern"). Abbildung 1.12 zeigt die grundsätzliche Schaltung.

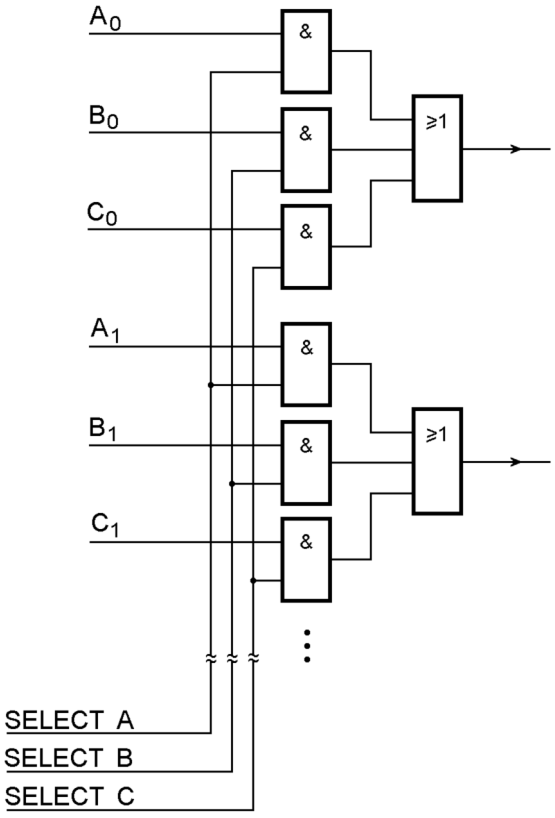


Abbildung 1.12 Auswahlschaltung (Datenselektor)

Erklärung:
Es gibt drei eingangsseitige Signalwege (A, B, C). Für jeden dieser Signalwege gibt es ein Auswahl-Steuersignal (SELECT A, SELECT B usw.). Je Bitposition ist eine UND-ODER-Schaltung vorgesehen. Ist beispielsweise das Steuersignal SELECT A aktiv, so werden die Eingänge A_0, A_1 usw. zu den Ausgängen durchgeschaltet.

Der Multiplexer (MUX)
Oft liegen die Auswahlangaben in binär codierter Form vor. Um einen von n Eingängen auszuwählen, braucht man ld n Auswahlsignale (mit anderen Worten: eine Auswahladresse). Deren Belegung muß decodiert werden, um die einzelnen UND-Gatter gemäß Abbildung 1.12 anzusteuern. Die Kombination aus Decoder und Datenselektor heißt *Multiplexer* (Abbildung 1.13).

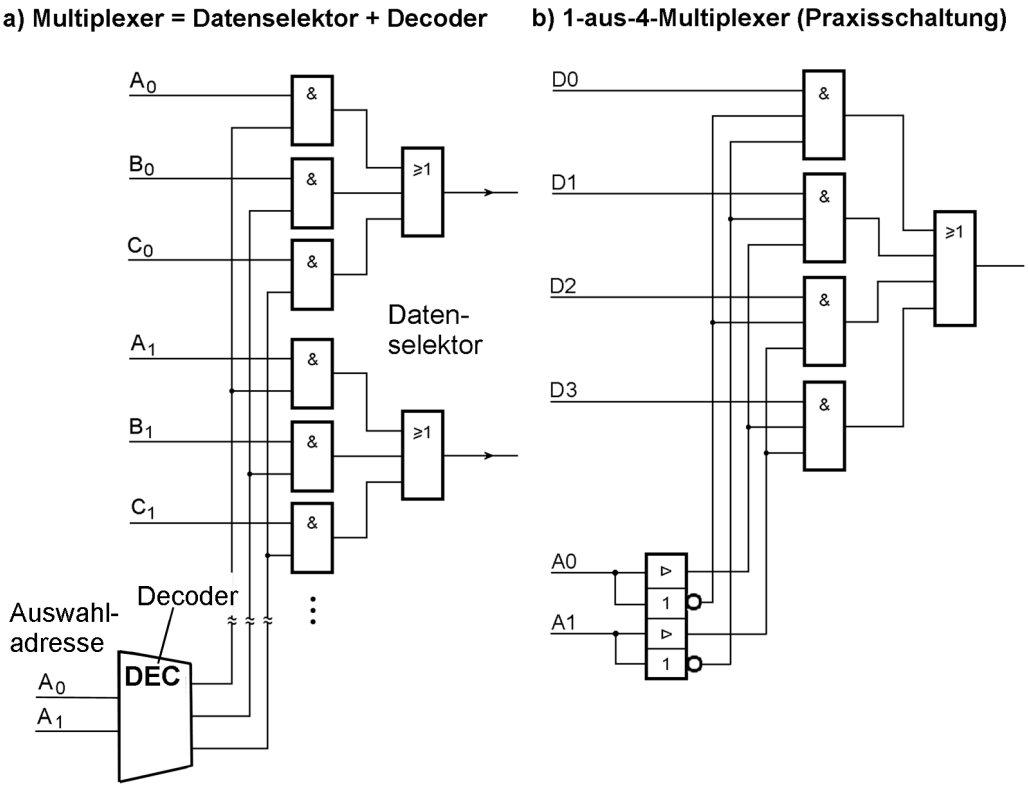


Abbildung 1.13 Multiplexer

Erklärung:

a) zur Veranschaulichung des Prinzips: dem Datenselektor von Abbildung 1.12 wurde ein Decoder vorgeschaltet,

b) aus Geschwindigkeitsgründen ist es sinnvoll, die Decodier- und Auswahlgatter miteinander zu kombinieren (Verringerung der Schaltungstiefe). Typische Multiplexer ermöglichen es, einen von 2, 4, 8 oder 16 Eingängen auszuwählen. Übliche Bezeichnungen:

- 2-fach, 4-fach usw.,
- 2-zu-1, 4-zu-1 (2 to 1, 4 to 1) usw.

Hier ist ein 4-zu-1-Multiplexer dargestellt.

Schaltungen ähnlich Abbildung 1.13b sind als Schaltkreise erhältlich. Manche Schaltkreise enthalten mehrere gleichartige Multiplexer mit gemeinsamen Adreßeingängen. Solche Multiplexerkonfigurationen sind praktisch Industriestandards (Abbildungen 1.14 bis 1.17). Sie werden in verschiedenen Baureihen und mit unterschiedlichen Ausgängen (zweiwertig oder Tri-State, direkt oder invertierend) angeboten.

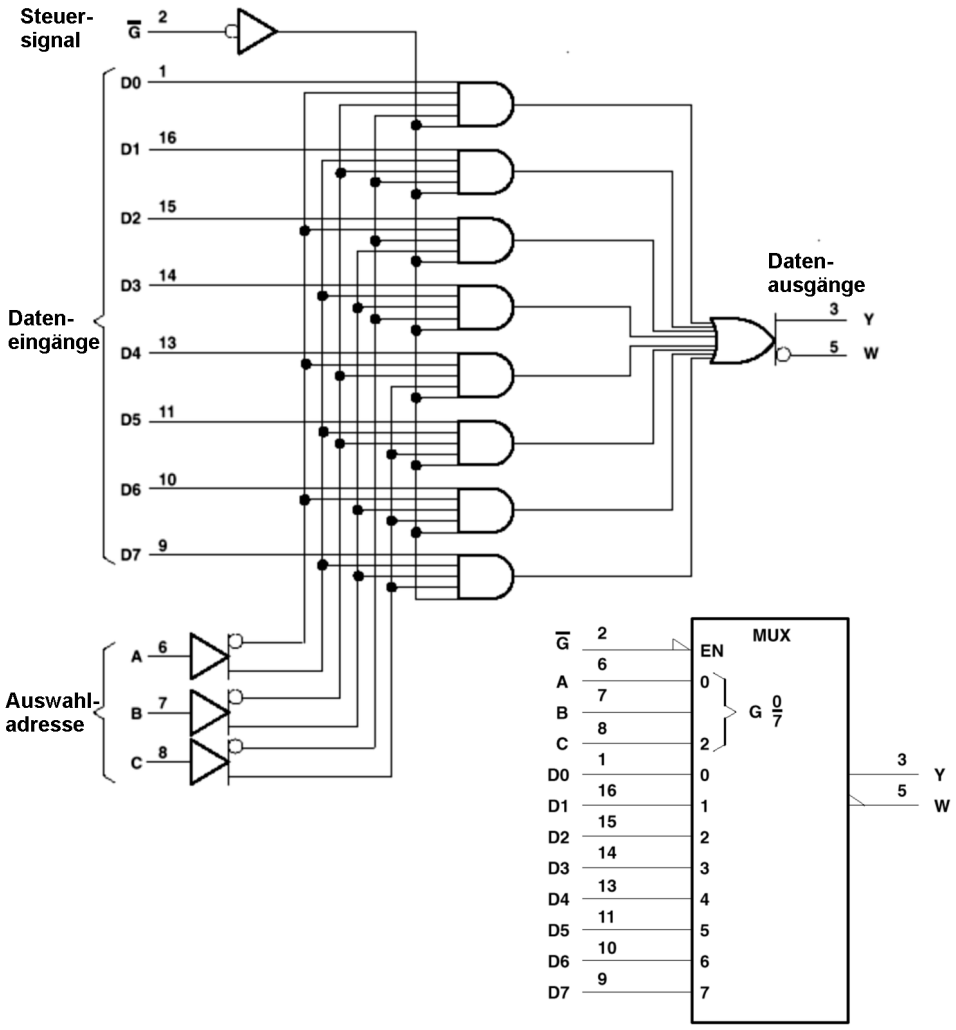


Abbildung 1.14 8-zu-1-Multiplexer. Oben: Innenschaltung, darunter: Schaltsymbol nach DIN 40 900 (Texas Instruments)

Erklärung:
Dieser Multiplexer hat einen direkten und einen invertierten Datenausgang. Die Datenausgänge sind zweiwertig (binär). Die Schaltung entspricht Abbildung 1.13b.

Das Steuersignal
Multiplexerschaltkreise haben typischerweise zusätzliche Steuereingänge, die direkt auf die Datenausgänge einwirken. Typische Bezeichnungen: Strobe, Gate, Enable, Output Control. Nur bei aktivem Steuersignal wird das ausgewählte Eingangssignal zum Ausgang durchgereicht. Ansonsten ist der Ausgang inaktiv. Und hier heißt es aufpassen:

- ein zweiwertiger direkter (nicht-invertierter) Ausgang verharrt auf 0 (Low),
- ein zweiwertiger invertierter Ausgang verharrt auf 1 (High),
- ein Tri-State-Ausgang wird hochohmig (liefert also "nichts").

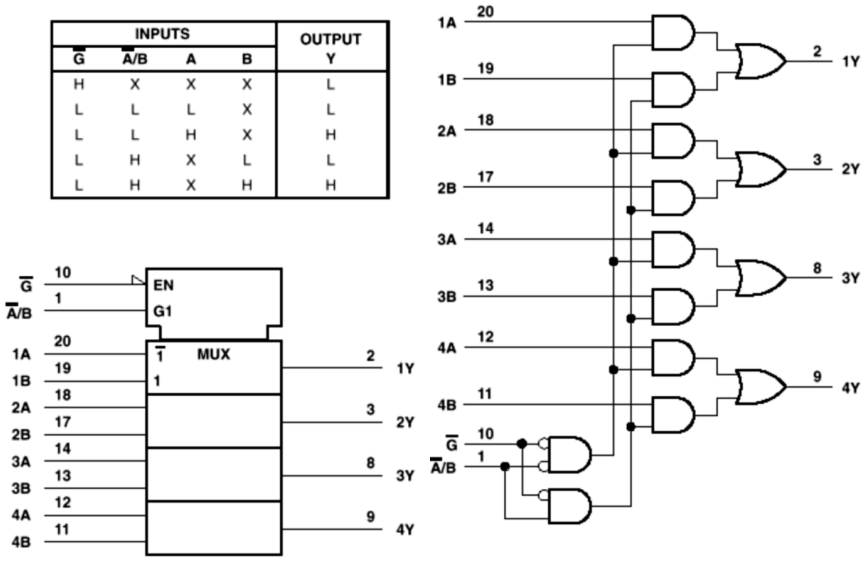


Abbildung 1.15 Multiplexerschaltkreis (Texas Instruments). Vier 2-zu-1-Multiplexer mit binären Ausgängen. Funktionstabelle, Innenschaltung, Schaltsymbol nach DIN 40 900

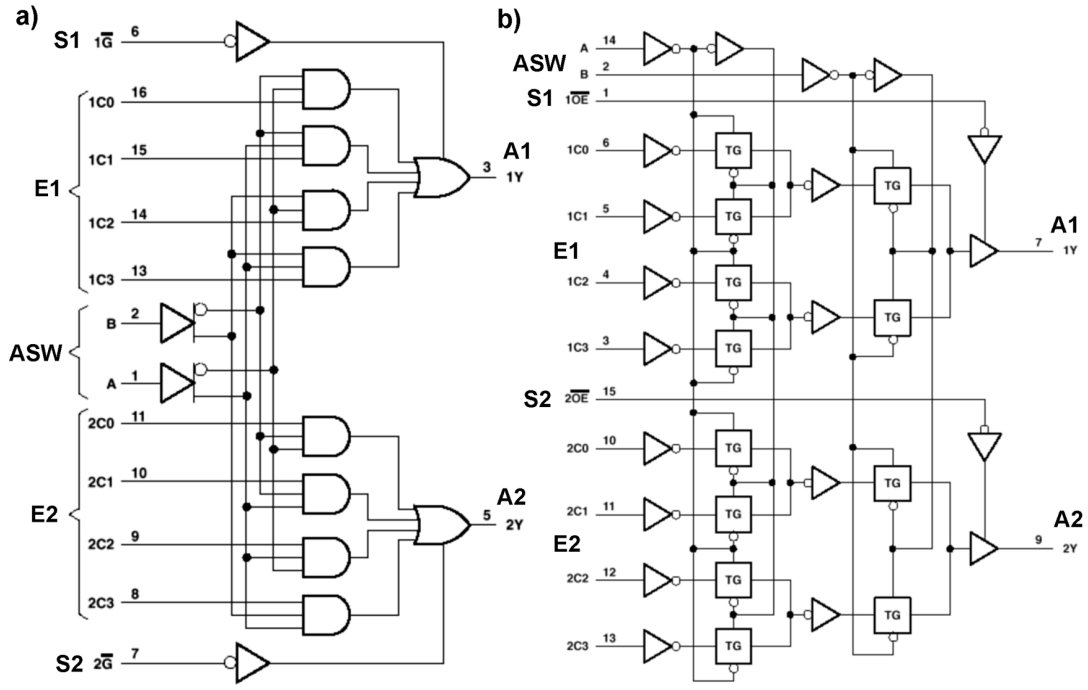


Abbildung 1.16 Eine Organisationsform, zwei Innenschaltungen (Texas Instruments)

Erklärung:
Die Schaltkreise enthalten zwei 4-zu-1-Multiplexer mit Tri-State-Ausgängen. Jeder der beiden Multiplexer hat ein eigenes Steuersignal. E1, A1, E2, A2 - Datenein- und -ausgänge des ersten und des zweiten Multiplexers; S1, S2 - Steuersignale des ersten und des zweiten Multiplexers; ASW - gemeinsame Auswahladresse.

a) Schaltkreis in TTL-Technologie. Die Multiplexer an sich entsprechen Abbildung 1.13a,

b) Schaltkreis in CMOS-Technologie. Die Datenauswahl erfolgt über gesteuerte CMOS-Schalter (TG = Transfer Gate). Da die Anschlüsse über Koppelstufen geführt sind, verhält sich der Schaltkreis wie ein aus Gattern aufgebauter Multiplexer (die Ausgänge werden aktiv getrieben; keine Rückwirkungen von den Ausgängen auf die Eingänge, nur eine Signalflußrichtung).

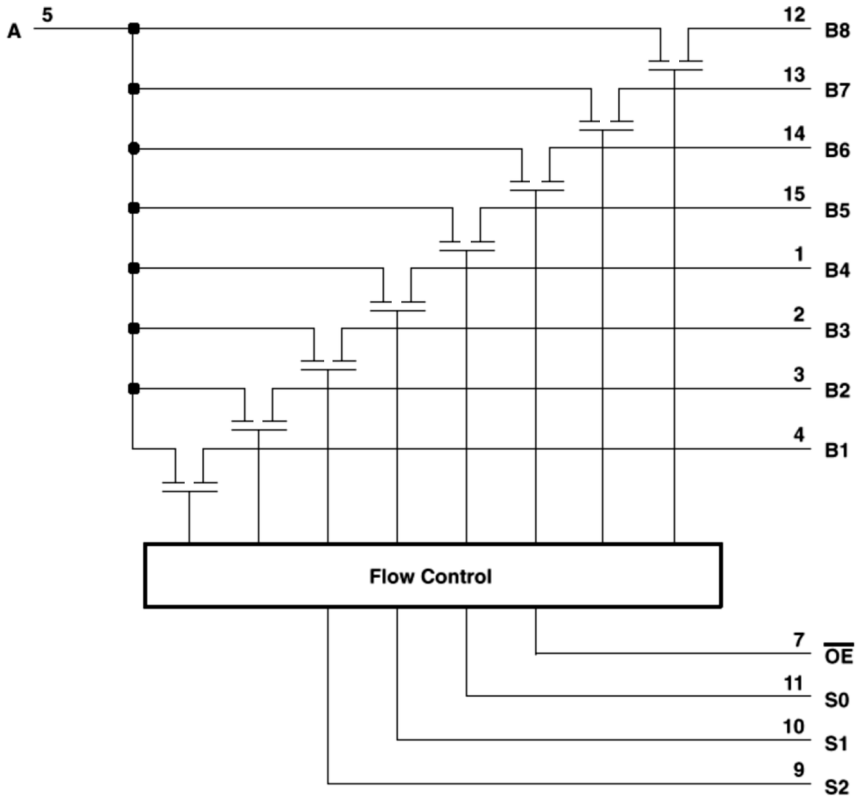


Abbildung 1.17 Multiplexer als Schalterbauelement (Texas Instruments)

Erklärung:
Als Beispiel ist hier ein 8-zu-1-Multiplexer mit FET-Schaltern dagestellt. S0...S2 - Auswahladresse; OE - Steuersignal (sperrt oder erlaubt Datendurchleitung). Besonderheiten:

- die Signallaufzeit zwischen ausgewähltem Eingang und Ausgang ist praktisch vernachlässigbar (z. B. maximal 250 ps),
- die Signale können in beide Richtungen fließen. Damit ist der Schaltkreis auch als Verteiler (Demultiplexer) einsetzbar.
- die Ausgänge werden nicht aktiv getrieben; somit muß die Treibfähigkeit von den jeweils eingangsseitig vorgeordneten Schaltkreisen erbracht werden.

1.2.2. Verteilen/Sperren

Beim Verteilen (Demultiplexing) sind ankommende Signale wahlweise an verschiedene Einrichtungen zu liefern; beim Sperren ist die Weitergabe von Signalen zu verhindern. Beide

Funktionen lassen sich gleichermaßen durch konjunktive Verknüpfung lösen (Abbildung 1.18): an der ausgewählten Einrichtung muß eine Eins ankommen, wenn das betreffende Signal den Wert Eins hat UND wenn die betreffende Einrichtung ausgewählt ist. Ansonsten ist eine feste Nullbelegung zu liefern (das ist gleichbedeutend mit dem Sperren des Signalflusses).

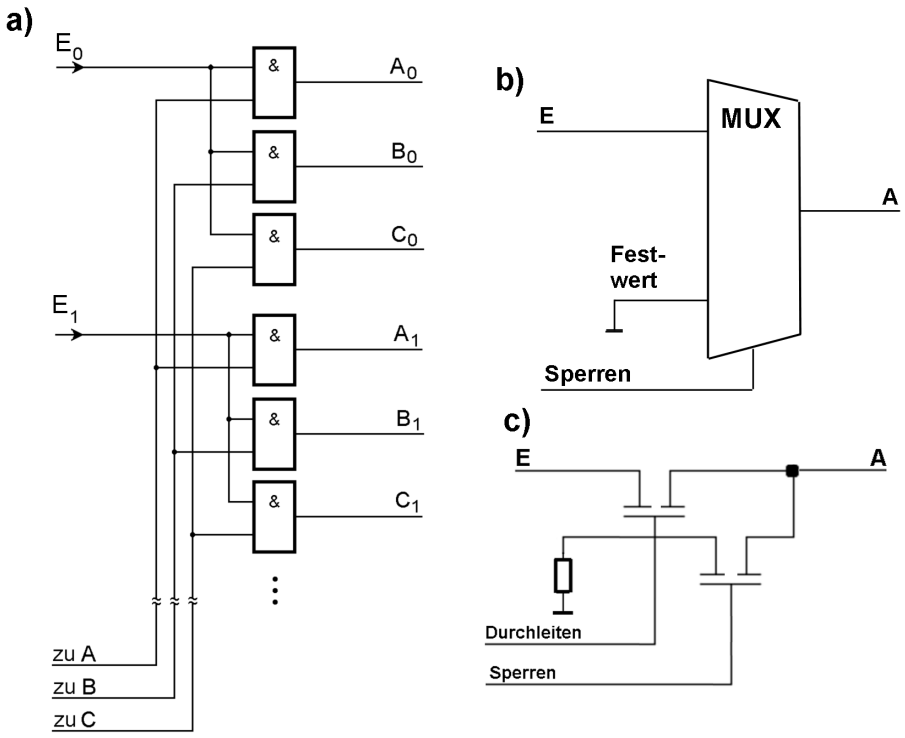


Abbildung 1.18 Verteiler- und Sperrschaltungen

Erklärung:

a) Verteilen bzw. Sperren mit UND-Gattern. Der eingangsseitige Signalweg E wird auf drei ausgangsseitige Signalwege geschaltet (A, B, C). Für jeden dieser Signalwege gibt es ein Auswahl-Steuersignal (zu A, zu B usw.). Je Bitposition ist ein UND-Gatter vorgesehen. Ist beispielsweise das Steuersignal *zu A* aktiv, so werden die Eingänge E_0, E_1 usw. zu den Ausgängen A_0, A_1 usw. durchgeschaltet.

b) Sperren mit Multiplexer. Ein 2-zu-1-Multiplexer schaltet entweder das Eingangssignal zum Ausgang durch oder legt den Ausgang auf einen Festwert (im Beispiel auf 0 V = Low).

c) eine Praxislösung. Sperren mit FET-Schaltern. Der Ausgang A wird entweder zum Eingang E durchgeschaltet oder über einen Pull-down-Widerstand an Masse gelegt (= Festwert Low). Die Vorteile:

- die Durchlaufverzögerung ist praktisch vernachlässigbar,
- die FET-Schalter halten es aus, wenn eine der beiden Seiten (E oder A) ausgeschaltet ist (Partial Power Down - ein wichtiger Betriebsfall beim Stromsparen und beim Wechseln von Funktionseinheiten während des Betriebs).

Hinweis:
Aus rein funktioneller Sicht ist es oftmals nicht erforderlich, Signalwege dann, wenn sie nicht benötigt werden, zu sperren, sie also mit festen Werten zu belegen (da es nicht schadet, wenn sie sozusagen nebenher mitschalten). Es gibt aber neumodische Anforderungen, die es geradezu erzwingen, Sperrvorkehrungen in viele Signalwege einzubauen:

- Verringerung der Störstrahlung (was nicht schaltet, kann auch nichts abstrahlen),
- Verringerung der kapazitiven Belastung (der jeweilige Treiber sieht nur die parasitären Kapazitäten der zugeschalteten Einrichtungen, nicht aber jene der abgeschalteten),
- Voraussetzung zum Stromsparen (was nicht schaltet, verbraucht - in CMOS-Technologie - auch (fast) keinen Strom),
- Voraussetzung zum Tauschen von Funktionseinheiten bei laufendem Betrieb (Hot Swapping).

Der Demultiplexer
Der Demultiplexer ist eine Verteilerschaltung, die - wie der Multiplexer - über binär codierte Auswahlsignale gesteuert wird. Hierzu werden die einzelnen Auswahlsignale (in Abbildung 1.18a: *zu A, zu B* usw.) an einen Decoder angeschlossen (wie beim Multiplexer in Abbildung 1.13a). Die zum Verteilen dienenden UND-Gatter (Abbildung 1.18a) können mit denen des Decoders zusammengefaßt werden (wie beim Multiplexer in Abbildung 1.13b). Siehe weiterhin die Abbildungen 1.20 und 1.21 nebst Erklärung.

1.2.3. Decodieren

Decodieren heißt: es gibt mehrere Eingangsleitungen, und es sind bestimmte Belegungen dieser Leitungen zu erkennen. Ist eine solche Belegung gegeben, so soll die betreffende Ausgangsleitung aktiv werden. Jede einzelne Bedingung läßt sich durch eine UND-Verknüpfung der betreffenden Eingangssignale erfassen. Diese werden gemäß der zu decodierenden Belegung entweder direkt oder negiert der UND-Verknüpfung zugeführt. Handelt es sich um n Eingangsleitungen, so können maximal 2^n verschiedene Belegungen decodiert werden. Man hat dann 2^n Ausgangsleitungen, von denen nur jeweils eine aktiv ist (1-aus-n-Decoder). Abbildung 1.19 zeigt einen Decoder für zweistellige Binärzahlen (1-aus-4), Abbildung 1.20 einen für dreistellige (1-aus-8).

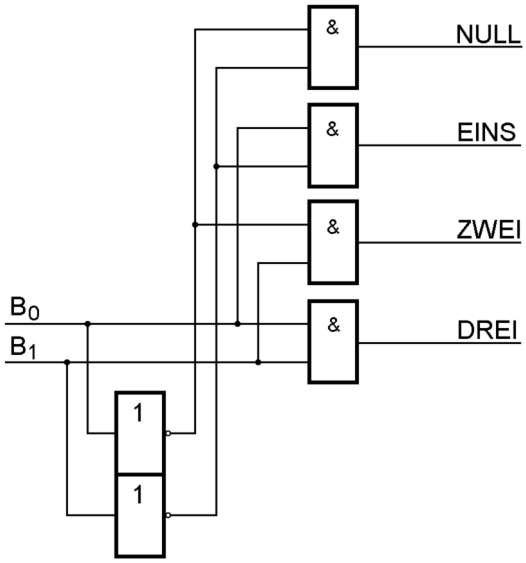


Abbildung 1.19 Decoder für zweistellige Binärzahlen (1-aus-4-Decoder). Funktionserklärung: siehe Tabelle 1.1

B1	B0	Ausgang	Verknüpfung
0	0	NULL	$\overline{B_0} \cdot \overline{B_1}$
0	1	EINS	$\overline{B_0} \cdot B_1$
1	0	ZWEI	$B_0 \cdot \overline{B_1}$
1	1	DREI	$B_0 \cdot B_1$

Tabelle 1.1 Wahrheitstabelle des 1-aus-4-Decoders

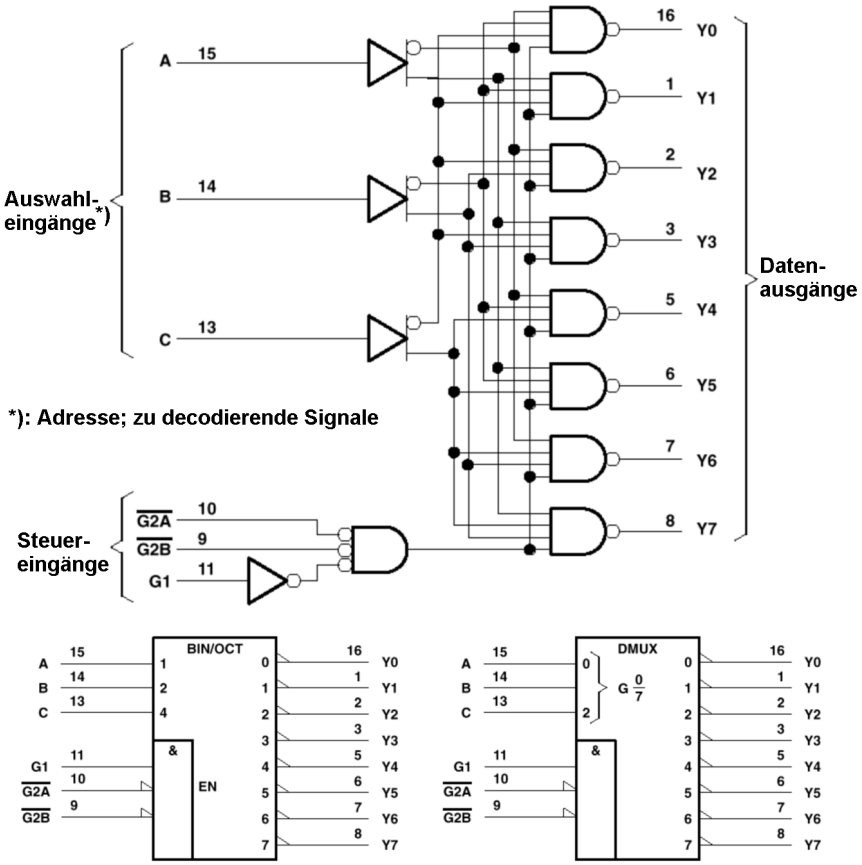


Abbildung 1.20 Ein Industriestandard: der Decoder- und Demultiplexerschaltkreis (74x138). Oben: Innenschaltung, darunter: Schaltsymbole nach DIN 40 900 (Texas Instruments)

Erklärung:
Der 74x138 ist ein 1-aus-8-Decoder/Demultiplexer mit invertierten Ausgängen und 3 zusätzlichen, konjunktiv verknüpften Steuereingängen (Enable Inputs). Ist die Steuerbedingung (G1 = 1, G2A = 0, G2B = 0) nicht erfüllt, so verharren alle 8 Ausgänge auf 1 (High). Die Abbildung zeigt weiterhin zwei Schaltsymbole. Auswahl: je nach Einsatzfall:

1. als Adreßdecoder (Codewandlung binär nach 1-aus-8 (BIN/OCT)),
2. als Verteiler bzw. Demultiplexer (DMUX).

Der 74x138 als Adreßdecoder
Dies ist die Anwendung, für die der 74x138 vor allem entwickelt wurde. Die Ausgänge sind deshalb aktiv Low, weil die Steuereingänge der nachzuschaltenden E-A- und Speicherschaltkreise ebenfalls typischerweise aktiv Low sind. Die Steuereingänge des Decoders können in den meisten Einsatzfällen mit den Steuersignalen der gängigen Mikroprozessor-Bussysteme direkt ("nach Kochbuch") zusammengeschaltet werden.

Der 74x138 als Verteiler/Demultiplexer
Das zu verteilende Signal wird an einen der Steuereingänge angeschlossen (Abbildung 1.21). Der über die Adreßeingänge ausgewählte Ausgang schaltet wie das zu verteilende Signal (je nach Anschluß direkt oder invertiert); die anderen Ausgänge verharren auf High.

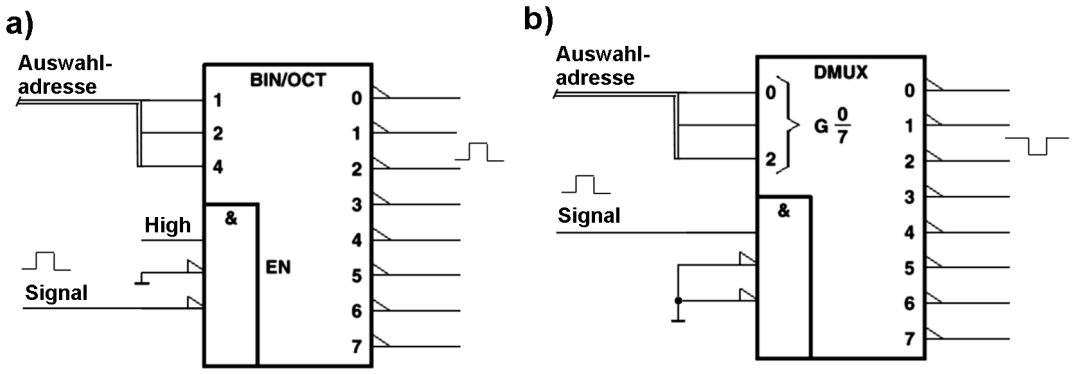


Abbildung 1.21 Der 74138 als Demultiplexer

Erklärung:
a) - da der Schaltkreis invertierende Ausgänge hat, wird das zu verteilende Signal an einen negierten Steuereingang angeschlossen; b) - bei Anschluß an den nichtnegierten Steuereingang erscheint ausgangsseitig ein invertiertes Signal.

Decoder für mehrstellige Binärzahlen
Ein Decoder mit n Eingängen erfordert - in naiver Implementierung gemäß den Abbildungen 1.19 und 1.20 - 2^n UND-Gatter mit n Eingängen. Durch eine mehrstufige, matrixförmige Struktur läßt sich der Aufwand deutlich verringern, wie Abbildung 1.22 am Beispiel eines Decoders für 6-Bit-Binärzahlen zeigt (1-aus-64).

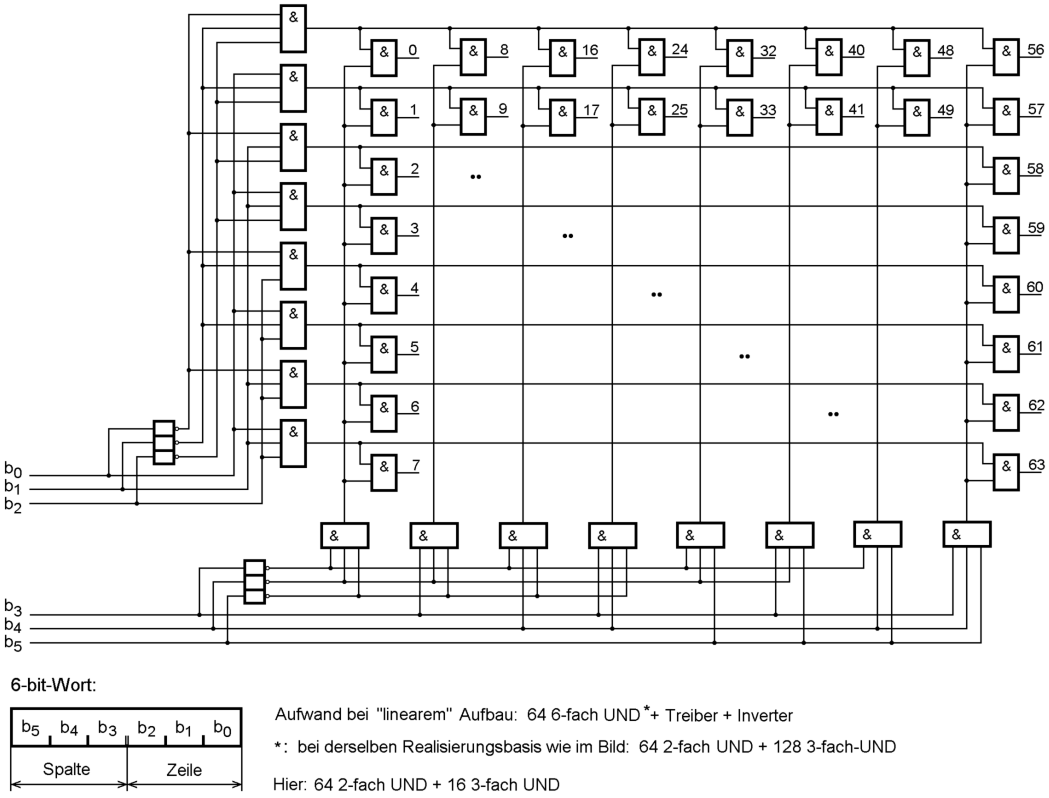
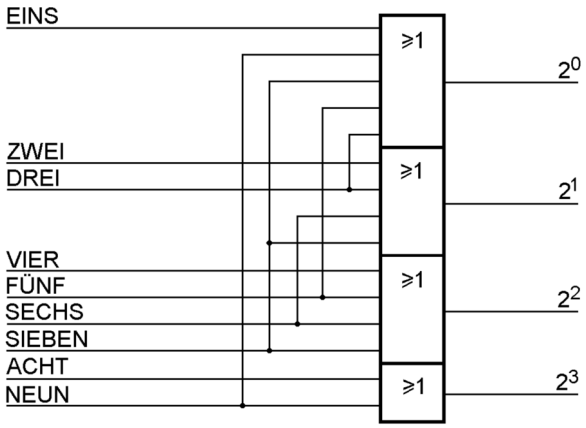


Abbildung 1.22 6-Bit-Decoder (1-aus-64) in Matrixstruktur

Erklärung:
Ein 1-aus-64-Decoder würde 64 6-fach-UNDs erfordern. Hier werden die zu decodierenden 6 Bits in zwei Abschnitte (Spalte und Zeile) zu je 3 Bits zerlegt. Beide Abschnitte werden unabhängig voneinander decodiert (1-aus-8; das erfordert jeweils 8 3-fach-UNDs). Die Zeilen- und Spaltensignale werden mit insgesamt 64 2-fach-UNDs zu den eigentlichen Ausgangssignalen verknüpft.

1.2.4. Codieren

Codieren heißt, Ausgangsleitungen mit einem bestimmten Bitmuster zu belegen, wenn jeweils eine von n Eingangsleitungen aktiv ist (also, einen 1-aus-n-Code in einen beliebigen anderen Code zu wandeln). Dazu wird je Ausgangsleitung eine ODER-Verknüpfung all jener Eingangsleitungen vorgesehen, bei deren Aktivierung die betreffende Ausgangsleitung aktiv werden muß. Abbildung 1.23 zeigt das Prinzip am Beispiel eines Codieres (Encoders) für binär codierte Dezimalzahlen.



Codetabelle:

	2^3	2^2	2^1	2^0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Abbildung 1.23 Codierer (Encoder) für binär codierte Dezimalzahlen

Erklärung:
Aus der Codetabelle ergibt sich die Schaltung auf naheliegende Weise. Beispiel: das Signal 2^1 ist zu aktivieren, wenn eingangsseitig eine der Dezimalzahlen ZWEI oder DREI oder SECHS oder SIEBEN anliegt.

1.2.5. Umcodieren (Codes wandeln)

Beliebige Codewandlungen (von einem Code A in einen Code B) lassen sich durch Hintereinanderschalten eines Decoders und eines Encoders verwirklichen (Abbildung 1.24). Gelegentlich sind in einer solchen Anordnung schaltalgebraische Vereinfachungen möglich. Heutzutage werden für Codewandlungen oft Speicherschaltkreise (z. B. PROMs) eingesetzt (Kapitel 4).

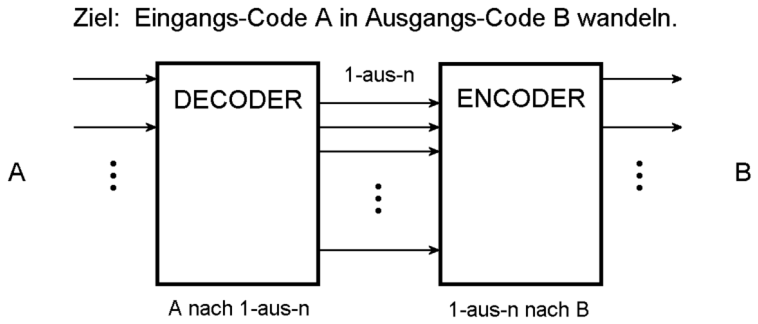


Abbildung 1.24 Codewandlung (Prinzip)

1.3. Verarbeitungs- und Rechenschaltungen

Kombinatorische Operationswerke
Kombinatorische Operationswerke führen in Prozessoren und Spezialschaltungen Transport- und Verknüpfungsoperationen aus (Abbildung 1.25). Die Eingangsbelegungen sind Operanden und Operationscodes, wobei der Operationscode die jeweils auszuführende Operandenverknüpfung bestimmt. Deren Ergebnis erscheint an den Ausgängen. Es wird zumeist durch zusätzliche Bedingungssignale (Flagbits) ergänzt.

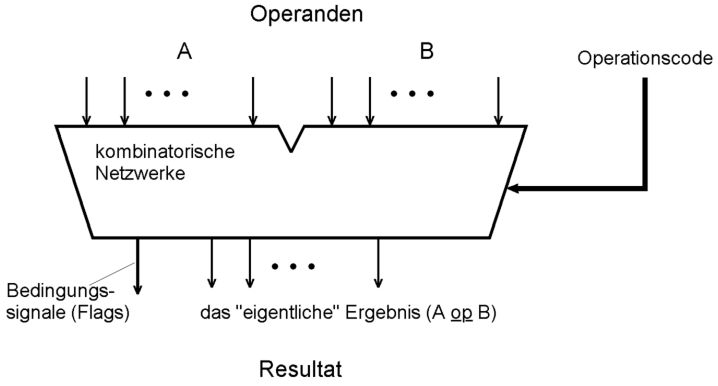


Abbildung 1.25 Das kombinatorische Operationswerk: die allgemeine Struktur

Die ALU
ALU = Arithmetic/Logic Unit (sprich: Arrissmättick Loddschigg Juhnitt). Dies ist die gängige Bezeichnung des Operationswerks typischer Universalrechner. Ein solches Operationswerk kann Rechenoperationen mit Binärzahlen und elementare logische Verknüpfungen ausführen (hinzu kommen Einzelbitoperationen, Verschiebeoperationen usw.). Im folgenden wollen wir die Auslegung solcher Operationswerke näher betrachten.

1.3.1. Elementare bitweise Verknüpfungen

Die Logikeinheit eines Prozessors führt üblicherweise wenigstens folgende Operationen aus:

- unmodifizierte Weiterleitung eines Operanden (MOV*)),
- Negation eines Operanden (NEG, NOT, CPL o. ä.),
- Konjunktion, Disjunktion und Antivalenz als bitweise Verknüpfung zweier Operanden (AND, OR, XOR).

*): wir beziehen uns hier - und in den folgenden Abschnitten - auf typische Maschinenbefehle, die wir (ohne Bindung an einen bestimmten Prozessortyp) mit allgemein üblichen Kürzeln (Assembler-Mnemonics) bezeichnen.

Die Schaltung ist einfach: für die einzelnen Funktionen sieht man die entsprechenden Gatter vor und faßt deren Ausgänge disjunktiv zusammen (Abbildung 1.26).

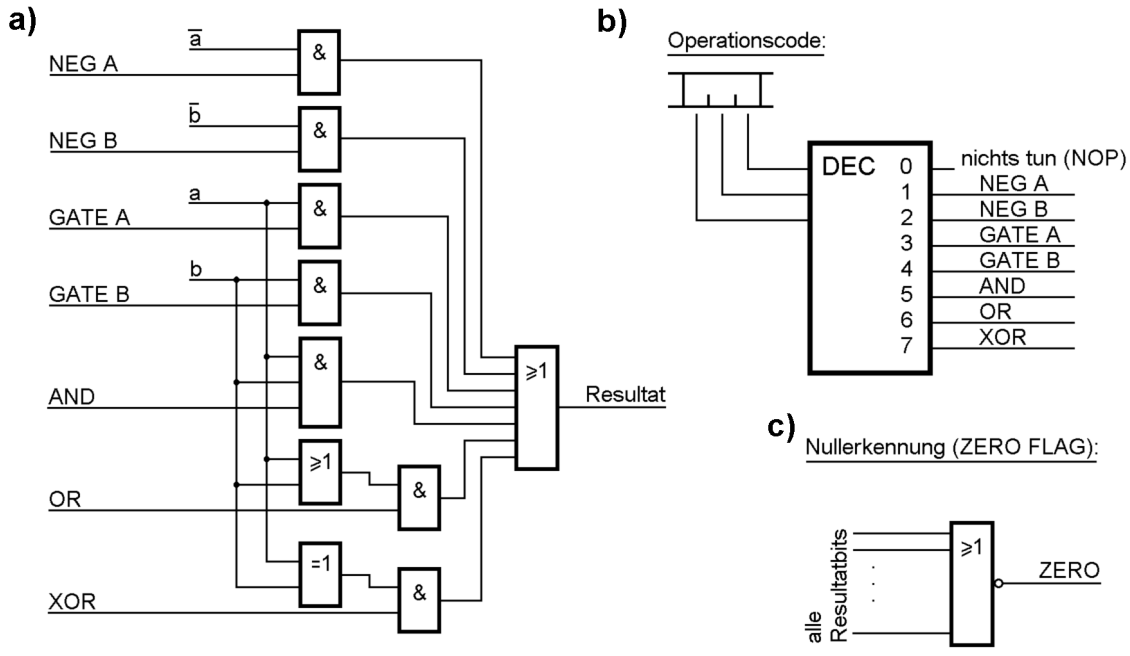


Abbildung 1.26 Universelle Logikeinheit

Erklärung:
a) - eine Bitposition im Schaltbild; b) - Befehlsdecodierung; c) - Nullerkennung.

Befehlsdecodierung:
Der auszuführende Befehl steht typischerweise in einem Befehlsregister. Der Operationscode wird decodiert. In Abbildung 1.26b ist ein Decoder angedeutet, der die Steuersignale der hier in Rede stehenden Befehle bildet (AND, OR, XOR usw.).

Funktionserklärung an Beispielen:

1. Operand A zum Ausgang transportieren (GATE A): Steuersignal GATE A aktiviert das UND-Gatter 1, so daß das Operandenbit a zum Ausgang (Resultat) durchgesteuert wird,
2. Operanden A und B disjunktiv verknüpfen (OR A, B): die Verknüpfung wird mittels ODER-Gatter 2 gebildet. Steuersignal OR aktiviert das UND-Gatter 3, so daß das Verknüpfungsergebnis zum Ausgang (Resultat) durchgesteuert wird.

Nullerkennung (Zero Flag)
Bei logischen Operationen gibt es eine Bedingung, die anwendungspraktisch besonders wichtig ist und deshalb als Flagbit oder Bedingungscode bereitgestellt werden muß: sie besagt, ob das Ergebnis nur Nullen enthält oder nicht (Zero Flag). Das ist durch eine NOR-Verknüpfung aller Ergebnissignale erkennbar.

1.3.2. Nullerweiterung

Hat ein Operand weniger Bits als die Verarbeitungsbreite der Hardware, so müssen die ungenutzten Bitpositionen mit Nullen belegt werden (Nullerweiterung; Zero Extend). Dies läßt sich durch konjunktive Verknüpfungen mit entsprechenden Steuersignalen erreichen (Abbildung 1.27).

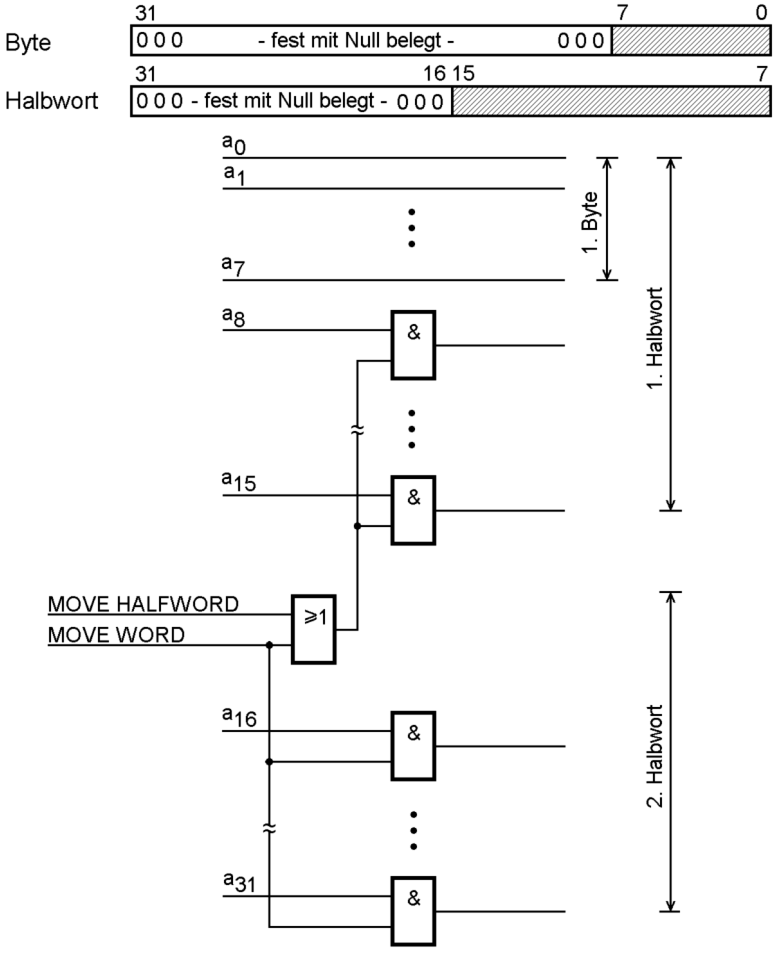


Abbildung 1.27 Nullerweiterung (Zero Extend)

Erklärung:
Die Schaltung wird in den entsprechenden Datenweg eingebaut (in Abbildung 1.25: einem der Eingänge A oder B vorgeschaltet). Die Steuersignale MOVE HALFWORD, MOVE WORD werden von der Befehlsablaufsteuerung erregt (z. B. von entsprechenden Mikrobefehlen). Die Wirkung geht aus Tabelle 1.2 hervor.

zu transportieren	aktive Steuersignale	Wirkung
Byte	keines	alle UND-Gatter gesperrt; an den Ausgangsbitpositionen 31...8 erscheinen Nullen
Halbwort (16 Bits)	MOVE HALFWORD	die UND-Gatter in den Bitpositionen 15...8 leiten die Eingangs-belegung weiter; an den Ausgangsbitpositionen 31...16 er-scheinen Nullen
Wort (32 Bits)	MOVE WORD	alle UND-Gatter leiten die Eingangsbelegung weiter; an den Ausgängen erscheint das eingangsseitige 32-Bit-Wort

Tabelle 1.2 Nullerweiterung: zur Erklärung der Wirkungsweise

1.3.3. Verschieben und Rotieren

In den meisten Prozessoren sind Verschiebe- und Rotationsbefehle vorgesehen. Auch diese Operationen werden typischerweise mit kombinatorischen Schaltungen ausgeführt. Was bedeutet eigentlich "Verschieben"? Beim *Rechtsverschieben* um ein Bit gelangt Bit 1 in Bitposition 0, Bit 2 in Bitposition 1 usw. Entsprechend gelangt beim *Linksverschieben* um ein Bit Bit 6 in Bitposition 7, Bit 5 in Bitposition Bit 6 usw. Sinngemäß werden die Daten bewegt, wenn eine Verschiebung um n Bits auszuführen ist (so kommt beim Linksverschieben um zwei Bits Bit 5 in Bitposition 7, Bit 4 in Bitposition 6 usw.).

Verschieben und Rotieren
Betrachten wir das Innere einer Datenstruktur, so bereitet es uns keine Schwierigkeiten, die Bewegung der Bits beim Verschieben zu verstehen. Was geschieht aber links und rechts, also gewissermaßen an den Rändern? Dies ist der Punkt, in dem sich Verschiebe- und Rotationsabläufe unterscheiden (Abbildung 1.28).

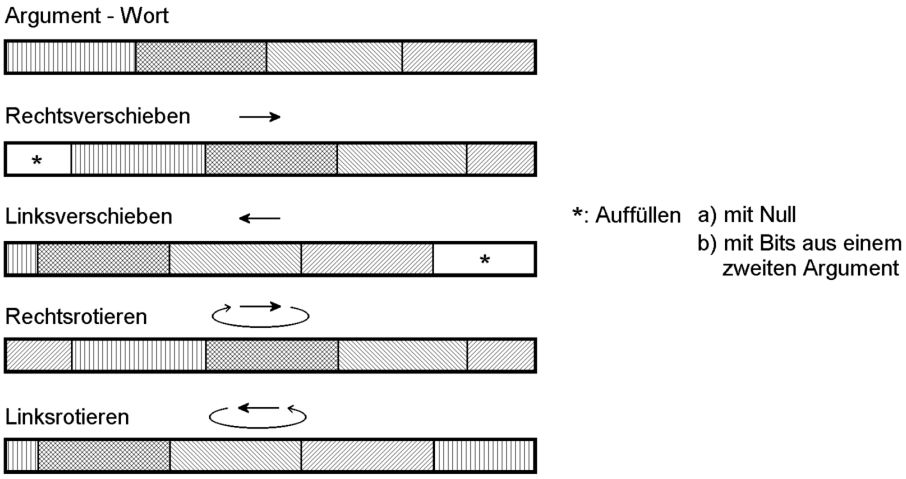


Abbildung 1.28 Verschieben und Rotieren

Beim *Rotieren* werden die hinausgeschobenen Bits am jeweils anderen Ende wieder zurückgeführt (Wrap Around). Rotieren ist also ein zyklisches Verschieben in den Grenzen der jeweiligen Operandenlänge. Wird ein Byte um ein Bit nach links rotiert, gelangt Bit 7 nach Bit 0, bei Rechtsrotation entsprechend Bit 0 nach Bit 7.

Hingegen gehen beim eigentlichen *Verschieben* die hinausgeschobenen Bits verloren. Die Bitpositionen, die am anderen Ende frei werden, werden meist mit Nullen aufgefüllt. In manchen Architekturen (beispielsweise IA-32) gibt es zusätzlich Verschiebebefehle mit Erweiterung. Diese haben einen zweiten Operanden, aus dem die besagten freigewordenen Bitpositionen aufgefüllt werden.

In vielen Architekturen werden bestimmte Flagbits in Verschiebe- bzw. Rotationsabläufe einbezogen. Zumeist ist das jeweils zuletzt hinausgeschobene oder umlaufende Bit in einem Flagbit direkt abfragbar. Ist das Flagbit am Rotieren beteiligt, so laufen die herausgeschobenen Bits nicht unmittelbar am anderen Ende des Operanden wieder ein, sondern gelangen zunächst in das Flagbit und von dort aus in das eigentliche Ergebnis zurück.

Wie lassen sich solche Abläufe schaltungstechnisch verwirklichen? Es ist dazu nur notwendig, für jede Ergebnis-Bitposition eine Auswahlschaltung vorzusehen, an die all jene Operanden-Bits angeschlossen sind, die irgendwie als Ergebnis wirksam werden können. Betrachten wir als Beispiel die Bitposition 5. Soll um 2 Bits nach rechts verschoben werden, so ist die Belegung der Bitposition 7 auszuwählen, soll um 4 Bits nach links verschoben werden, die Belegung der Bitposition 1. Das läßt sich mit Multiplexern realisieren, die in passender Reihenfolge mit den einzelnen Operandenbits beschaltet werden.

Der Barrel Shifter

Das ist die übliche Bezeichnung für eine universelle Verschiebeeinheit, mit der man Daten um mehr als ein Bit verschieben oder rotieren kann. Wir wollen uns die Funktionsweise zunächst an einer 8-Bit-Ausführung klarmachen. Abbildung 1.29 zeigt eine Schaltung, die beliebige Verschiebungen um 0..7 Bitpositionen ermöglicht. Es sind 8 Multiplexer mit 8 Eingängen vorgesehen. Alle Adreßeingänge sind gleichermaßen an einen 3-Bit-Datenweg angeschlossen,

der die Anzahl der Bits liefert, um die verschoben werden soll (SHIFT COUNT). Ein SHIFT COUNT von Null heißt: Operand nicht verschieben, sondern einfach durchreichen.

Abbildung 1.30 verdeutlicht den Anschluß der Operandenbits an die Multiplexer. Die Schaltung löst zunächst die Aufgabe des Rotierens. Ein SHIFT COUNT a bedeutet Rechtsrotieren um a Bits bzw. ein Linksrotieren um 8-a Bits (mit anderen Worten: zum Linksrotieren ist der SHIFT COUNT im Zweierkomplement anzugeben).

Anhand der Abbildung 1.30 können wir uns leicht davon überzeugen, daß wegen des Wrap Around ein Rechtsrotieren um a und ein Linksrotieren um 8-a Bits tatsächlich auf das gleiche Ergebnis führt.

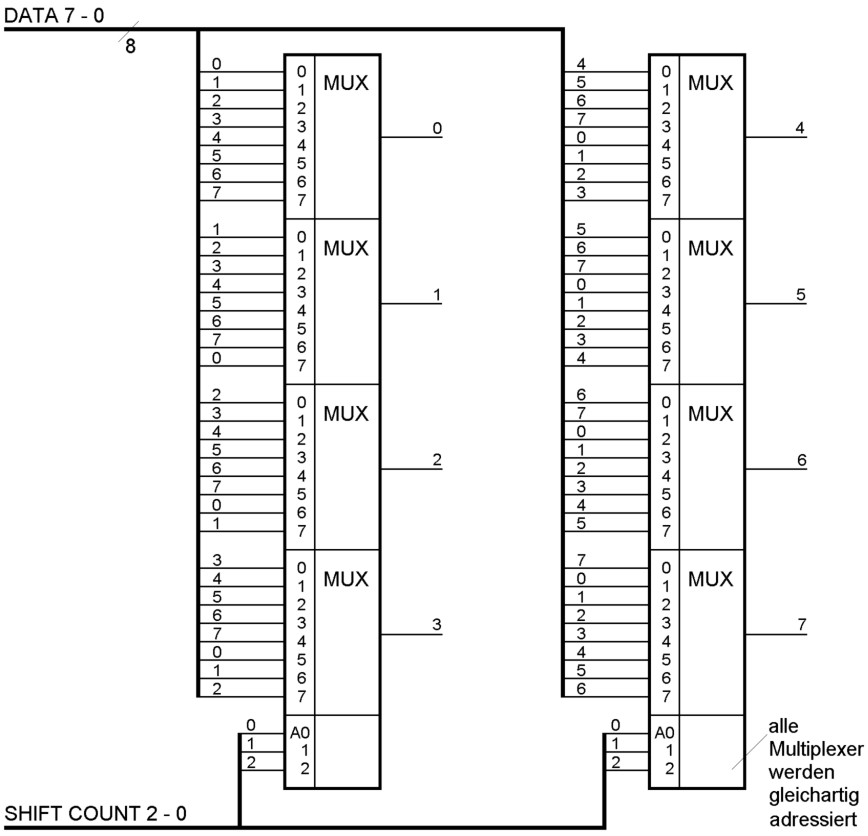


Abbildung 1.29 8-Bit-Verschiebeanordnung (Barrel Shifter)

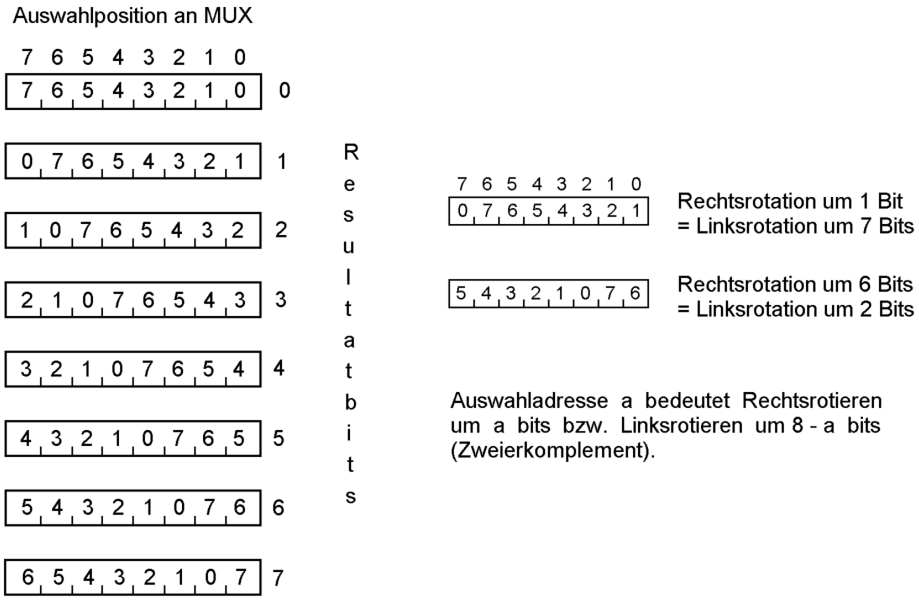


Abbildung 1.30 Anschluß der Operandenbits an die Multiplexer-Eingänge beim 8-Bit-Barrel Shifter

Das eigentliche Verschieben (SHIFT) bereitet nun keine Schwierigkeiten mehr - es genügt, die freiwerdenden Positionen mit Nullen zu belegen. Verschieben ist also ein Rotieren mit Ausblenden der freiwerdenden Bits. Zum Ausblenden können wir den Multiplexern UND-Gatter nachschalten. Diese sind folgendermaßen mit einer sog. Bitmaske anzusteuern: bei Linksverschiebung um ein Bit mit 1111 1110B, um zwei Bits mit 1111 1100B usw.; entsprechend bei Rechtsverschiebung mit 0111 1111B, 0011 1111B usw. Beim Rotieren muß die gesamte Bitmaske Einsen enthalten. Nehmen wir die Linksverschiebung als Beispiel. SHIFT COUNT 0 entspricht 1111 1111B (alles durchlassen), SHIFT COUNT 1 entspricht 1111 1110B (freiwerdende Bitposition 0 ausblenden) usw. Wir müssen somit nichts weiter tun, als den SHIFT COUNT zu decodieren und die im 1-aus-8-Code gelieferte Eins zu allen jeweils höherwertigeren Bitpositionen durchzureichen (Abbildung 1.31).

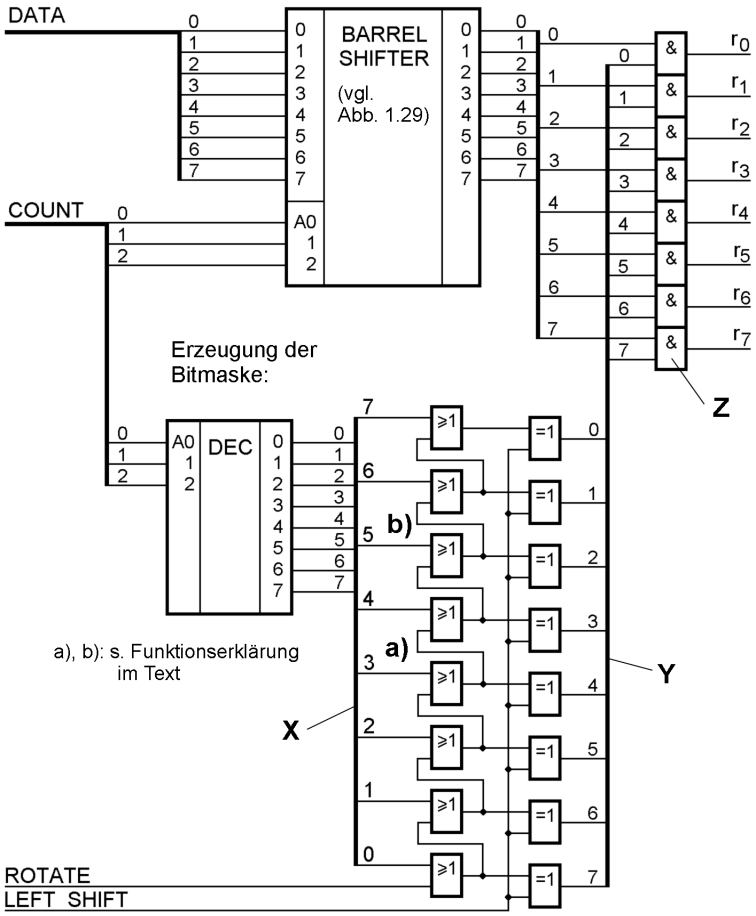


Abbildung 1.31 Universelle 8-Bit-Verschiebe- und Rotationseinrichtung

Funktionserklärung an Beispielen:

a) Rechtsverschieben um 3 Bits. COUNT = 3. Der Decoder liefert eine 1 in Bitposition 3. Diese pflanzt sich über die ODER-Gatter bis in Bitposition 7 (Kabelbaum X) fort. Am Kabelbaum Y entspricht das den Bitpositionen 4...0. Somit wird an den UND-Gattern Z eine Bitmaske 0001 1111B wirksam.

b) Linksverschieben um 3 Bits. COUNT = 5 (Zweierkomplement). Der Decoder liefert eine 1 in Bitposition 5, die sich bis in Bitposition 7 (Kabelbaum X) fortpflanzt. Am Kabelbaum Y entspricht das den Bitpositionen 2...0 bzw. der Bitmaske 0000 0111. Bei Linksverschiebung (LEFT SHIFT) wird aber die Bitmaske über die XOR-Gatter invertiert. Sie wird somit als 1111 1000 wirksam.

Barrel Shifter sind Bestandteil aller Hochleistungsprozessoren. Damit können Verschiebe- und Rotationsabläufe über die volle Verarbeitungsbreite der Hardware in einem einzigen Maschinenzyklus ausgeführt werden. Das ist nicht nur für die eigentlichen Verschiebe- und Rotations*befehle* von Bedeutung, sondern auch für alle anderen Abläufe, bei denen solche Vorgänge benötigt werden (beispielsweise für die Multiplikation und Division).

Das Crossbar-Netzwerk als Barrel Shifter

In der Praxis wird der Barrel-Shifter meist nicht mit Multiplexern aufgebaut. Vielmehr wird die steuerbare Verbindung "Jeder mit Jedem" durch eine Matrixanordnung von Koppelstufen (Transfer Gates) verwirklicht (Abbildung 1.32). Der Name "Crossbar" stammt aus der Fernsprechtechnik, wo solche Strukturen erstmals entwickelt wurden, und zwar seinerzeit noch auf elektromechanischer Grundlage (Kreuzschienenverteiler).

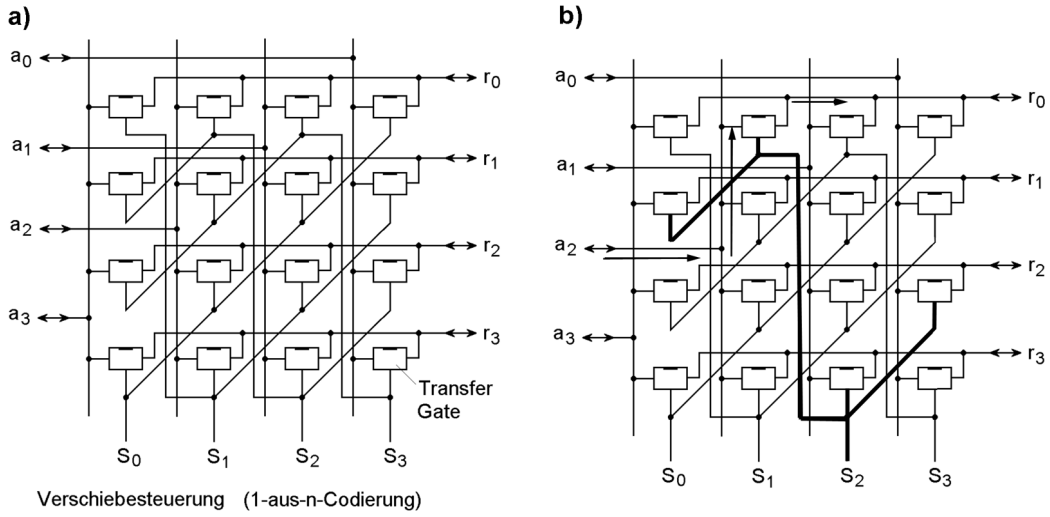


Abbildung 1.32 Crossbar-Netzwerk als technische Verwirklichung eines Barrel Shifters. a) Struktur; b) Beispiel einer Verschiebung

Erklärung:

a_0, a_1 usw. - Eingänge; r_0, r_1 usw. - Ausgänge; S_0...S_3 - Verschiebesteuersignale (S_0 - Verschieben um 0 Bits (= keine Verschiebung); S_1 - Rechtsverschieben um 1 Bit usw.). Als Beispiel aktivieren wir S_2 (Rechtsverschiebung um 2 Bits). Verfolgen wir die Signale im Schaltplan, so erkennen wir, daß a_2 zu r_0 durchgeschaltet wird (Pfeile in Abbildung 1.32b), a_3 zu r_1 usw.

1.3.4. Markierungsvektoren, Prioritätscodierung, Indexwerte

Diese Begriffe wollen wir verwenden, um Schaltmittel zur Lösung folgender Aufgaben zu betrachten:

- aus einem Operanden-Binärvektor ist ein Binärvektor gleicher Länge zu erzeugen, der *nur eine einzige Eins* enthält, und zwar - wählbar - jene, die der niedrigstwertigen oder der höchstwertigen Eins im Operanden entspricht (Markierungsvektor, Prioritätscodierung),
- die Position dieser Eins ist als Binärzahl zu bestimmen (Indexwert oder Bitadresse),
- aus einem Indexwert (einer Bitadresse) ist ein Binärvektor mit einer Eins an der entsprechenden Position zu bilden.

Einige der modernen Prozessorarchitekturen haben entsprechende Maschinenbefehle. Die Bezeichnungsweise ist allerdings nicht einheitlich (wir haben hier eine gewählt, die etwas mehr akademisch (= herstellerneutral) ist).

Markierungsvektor (Prioritätscodierung)
Wir wollen zunächst einen Markierungsvektor für die niedrigstwertige Eins bilden. Dazu müssen wir nichts weiter tun, als in jeder Bitposition ein UND-Gatter vorzusehen, das das "eigene" Operandenbit nur dann durchläßt, wenn alle vorhergehenden (jene mit niedrigerem Bit-Index) mit Null belegt sind. Sinngemäß erfordert ein Markierungsvektor für die höchstwertige Eins UND-Gatter, die das jeweilige "eigene" Operandenbit nur dann weitergeben, wenn alle nachfolgenden Operandenbits (mit höherem Bit-Index) Nullen sind. Dabei können alle jeweils in Frage kommenden Operandenbits parallel verknüpft werden (Abbildung 1.33), es ist aber auch "Weiterschleifen" von Bitposition zu Bitposition (Kaskadierung) möglich. Zudem liegt es nahe, an Kompromißlösungen zu denken (abschnittsweise Parallelverknüpfung (z. B. über jeweils 8 Bits) und Kaskadierung von Abschnitt zu Abschnitt).

Solche Netzwerke heißen auch *Prioritätscodierer (Priority Encoder)*, da sie einen Ergebnisvektor bilden, in dem nur die Bitposition mit der niedrigsten bzw. höchsten "Priorität" aktiv ist (eine offensichtliche Anwendung besteht darin, aus verschiedenen gleichzeitig anhängigen Anforderungen diejenige auszuwählen, welche zuerst - mit höchster Priorität - bearbeitet werden soll*)). Sie sind als Schaltkreise verfügbar.

*): typische Anwendungen: (1) Unterbrechungsannahme, (2) Vermittlung von Master-Anforderungen an einem Bussystem, (3) Vermittlung von Speicherzugriffsanforderungen.

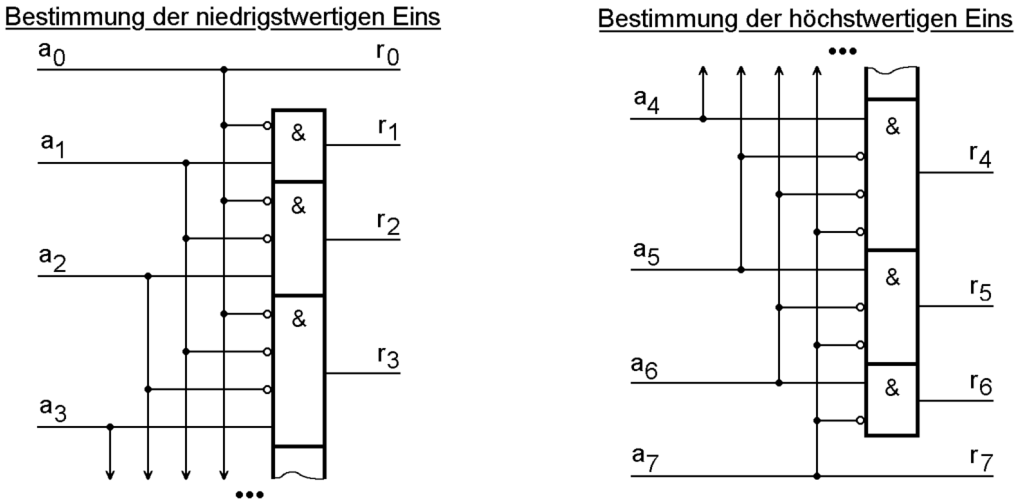


Abbildung 1.33 Bildung von Markierungsvektoren (Prioritätscodierung)

Indexwert aus Markierungsvektor
Hat man erst einmal einen Markierungsvektor, so ist nur eine gewöhnliche Codierschaltung (aus ODER-Gattern) notwendig, um aus dem 1-aus-n-Code die Bitadresse bzw. den Index des jeweils gesetzten Bits zu bestimmen. Es gibt Priority-Encoder-Schaltkreise mit eingebauter Codierfunktion.

Markierungsvektor aus Indexwert
Die Wandlung einer als Binärzahl codierten Bitadresse in einen Markierungsvektor erfordert lediglich einen 1-aus-n-Decoder.

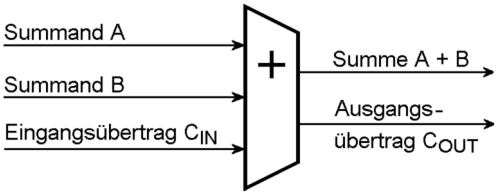
Anwendungsbeispiel: manche Prozessoren haben Einzelbitbefehle (Bits setzen, löschen, testen usw.). Um sie in einer üblichen universellen Logikeinheit auszuführen, werden aus den Bitadressen Binärvektoren gebildet, deren Länge der Verarbeitungsbreite entspricht (z. B. in einer 8-Bit-Maschine aus der Bitadresse 5 ein Binärvektor 0010 0000).

1.3.5. Addieren und Subtrahieren von Binärzahlen

Addieren und Subtrahieren in einer Binärstelle

In einer beliebigen Stelle haben wir zwei Operandenbits und einen einlaufenden Übertrag zu verarbeiten. Wir erhalten ein Summenbit und einen in die nächsthöhere Stelle weiterzugebenden Übertrag. Beim Subtrahieren kennzeichnen die Überträge ein "Borgen" von der jeweils höherwertigen Stelle, genau wie im Dezimalen. Abbildung 1.34 zeigt die vollständigen Wahrheitstabellen für Addition und Subtraktion in einer Binärstelle sowie die Zusammenschaltung für mehrere Binärstellen, wobei die Überträge jeweils zur nächsthöheren Stelle weitergegeben werden (Kaskadierung).

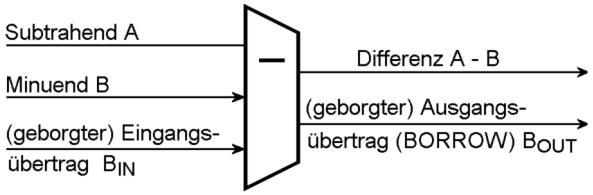
a) Addition (A + B)



A + B

A	B	C_{IN}	S	C_{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

b) Subtraktion (A - B)



A - B

A	B	B_{IN}	D	B_{OUT}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Kaskadierung:

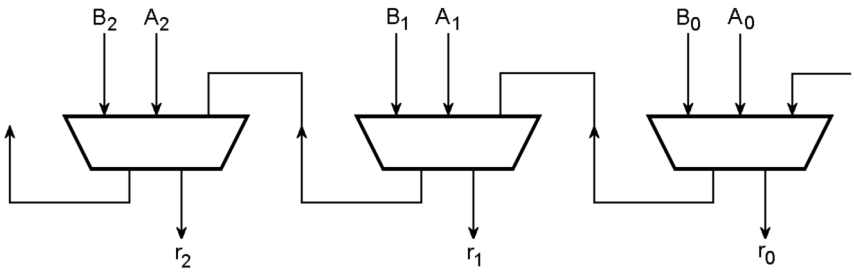


Abbildung 1.34 Rechnen in einer Binärstelle

Halbaddierer

Ein Halbaddierer (Half Adder) ist eine Schaltung, die zwei Binärstellen zueinander addiert, ohne einen einlaufenden Übertrag zu berücksichtigen. Abbildung 1.35 zeigt Wahrheitstabelle und Schaltung.

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

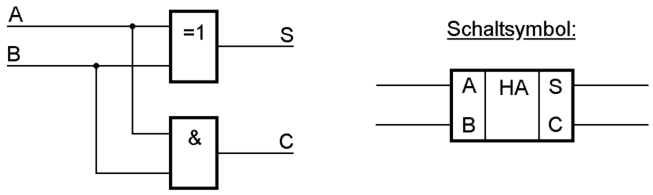


Abbildung 1.35 Halbaddierer

Volladdierer

Ein Volladdierer (Full Adder) addiert zwei Binärstellen unter Berücksichtigung des einlaufenden Übertrages zueinander. Die vollständige Wahrheitstabelle haben wir bereits in Abbildung 1.34 kennengelernt. Ein Volladdierer kann aus zwei Halbaddierern aufgebaut werden, wobei der zweite zur Summe, die der erste gebildet hat, den einlaufenden Übertrag addiert. Der Ausgangsübertrag entsteht durch ODER-Verknüpfung der Ausgangsüberträge beider Halbaddierer. Alternativ dazu kann man die Wahrheitstabelle von Abbildung 1.34 direkt in eine zweistufige UND-ODER- (bzw. NAND-NAND-) Schaltung umsetzen (Abbildung 1.36).

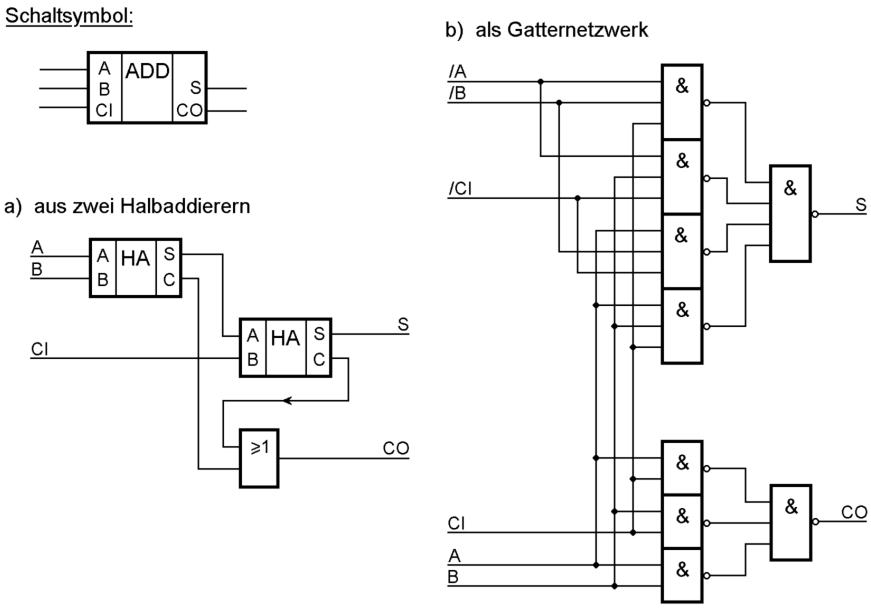


Abbildung 1.36 Volladdierer

Mehrstelliger Addierer

Die Kaskadierung der einzelnen Volladdierer zum mehrstelligen Addierer geht bereits aus Abbildung 1.34 hervor. Abbildung 1.37 enthält einen genaueren Schaltplan eines vierstelligen Addierers. Solche Addierer sind als Schaltkreise erhältlich.

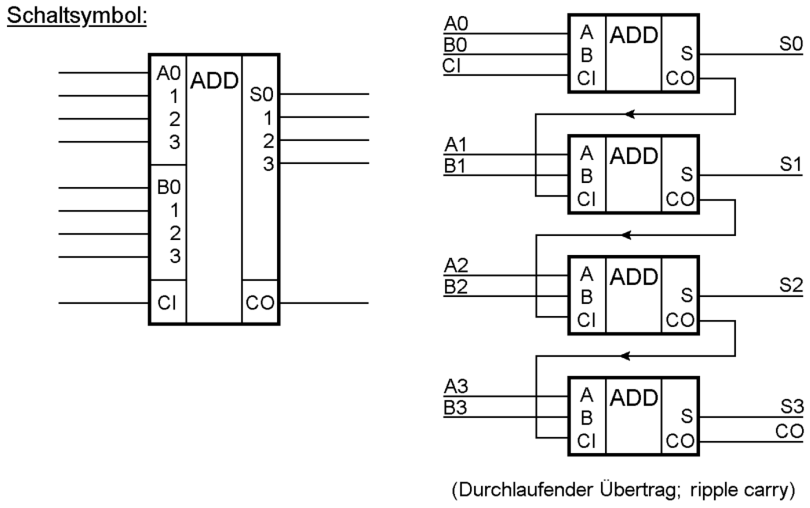


Abbildung 1.37 Addierer für vier Binärstellen

Der Übertrag bestimmt letzten Endes die Dauer des Additionsvorgangs
Eine Addition ist erst dann abgeschlossen, wenn der Ausgangsübertrag gebildet wurde. Betrachten wir beide Extremfälle:

- der günstigste Fall: zwischen den einzelnen Stellen gibt es gar keine Überträge. Rechenbeispiel: EE ... EH + 11 ... 1H.
- der schlimmste Fall: der Ausgangsübertrag hängt von dem in die erste Stelle einlaufenden Übertrag (dem Eingangsübertrag CI) ab; der Übertrag läuft also gleichsam von der niedrigstwertigen bis zur höchstwertigen Binärstelle hindurch. Rechenbeispiel:
  EE...EH + 11 ... 1H + CI .

Wann rechnet man mit einem einlaufenden Übertrag CI? - Vor allem in folgenden Fällen:

- beim Subtrahieren bzw. beim Bilden des Zweierkomplements (um die erforderliche Eins zu addieren, wird üblicherweise ein fester Eingangsübertrag eingespeist),
- beim Rechnen mit Binärzahlen, die länger sind als die Verarbeitungsbreite. (Dabei wird abschnittsweise addiert, wobei vom zweiten Abschnitt an der Ausgangsübertrag der vorhergehenden Addition als Eingangsübertrag einfließt.)

Die Rechenzeit der Addition - im Sinne der Durchlaufzeit durch die kombinatorischen Schaltungen - ist also an sich datenabhängig. Es hat immer wieder Versuche gegeben, diese Tatsache auszunutzen und Addierwerke zu bauen, die von sich aus melden, wenn sie mit dem Rechnen fertig sind. Diese sog. asynchrone Betriebsweise hat aber auch ihre Schwierigkeiten. Deshalb bevorzugt man beim heutigen Stand der Technik die vollsynchrone Arbeitsweise mit festem Taktraster. Ein solches Taktraster muß aber auf den jeweils ungünstigsten Betriebsfall (Worst Case) abgestellt sein. Damit kommt dem Durchlauf der Überträge entscheidende Bedeutung zu.

Die Durchlaufzeit durch ein Addierwerk wird im wesentlichen von der Übertragsweiterschaltung bestimmt. Der kritische Weg: vom eingespeisten Eingangsübertrag CI zum Ausgangsübertrag. Im ungünstigsten Fall wirken hierbei Signale "quer" über alle Binärstellen. Dies hat entscheidenden Einfluß auf die Festlegung des internen Taktrasters. Grob und vereinfacht gesagt: will man einen 100-MHz-Prozessor bauen, so darf es vom Eingangs- bis zum Ausgangsübertrag keinesfalls länger dauern als 10 ns. Je mehr MHz oder GHz die Marketingleute von den Entwicklern verlangen, desto weniger Zeit steht zur Verfügung - also heißt es, sich etwas einfallen zu lassen.

Achtung:
Nehmen wir an, wir haben einen Prozessor mit einem internen Takt von 500 MHz, der in jedem Taktzyklus eine Verarbeitungsoperation ausführen kann. Wie lange darf der Übertragsdurchlauf dauern? - Richtig: keinesfalls länger als 2 ns. Das heißt: 2 ns nach Bereitstellung der Operanden wird das Rechenergebnis übernommen. Und der Hersteller hat es so hingetrickst, daß auch im ungünstigsten Fall das Ergebnis tatsächlich innerhalb von 2 ns gebildet wird (selbstverständlich mit den erforderlichen Sicherheitszuschlägen - es sind schließlich Profis). Nun kommen ganz kluge Zeitschriften- und Bestsellerschreiber und empfehlen uns, die Taktfrequenz einfach hochzudrehen (Übertaktung). Was hat das zur Folge? - Die Zeit zwischen Bereitstellung der Operanden und Übernahme des Ergebnisses wird entsprechend geringer. Die kombinatorischen Schaltungen werden aber nicht schneller. Damit werden zunächst die vom Hersteller vorgesehenen Sicherheitsreserven gleichsam aufgefressen. Bei all den Datenkombinationen, in denen der ungünstigste Betriebsfall nicht eintritt, wird es weiterhin funktionieren. Aber dann, wenn ... (soll heißen: Übertakten ist Glücksspiel - sogar dann, wenn es "offensichtlich" funktioniert (das System also nach der Bastelaktion nicht öfter abstürzt als vorher)).

Durchlaufender Übertrag (Ripple Carry)
Bei der in Abbildung 1.37 gezeigten Schaltung läuft der Übertrag nacheinander durch alle Stellen hindurch. Das kostet nur wenig Aufwand. Durch die hohe Schaltungstiefe (namentlich bei Verarbeitungsbreiten von 32 Bits und mehr) ergeben sich aber sehr lange Additionszeiten (bei einem zweistufigen Netzwerk je Binärstelle kommt man für 32 Bits auf eine Schaltungstiefe von 64).

Übertragsvorausschau (Carry Look Ahead, CLA)
Mit zusätzlichen Schaltmitteln werden Überträge parallel zur Summe gebildet. Betrachten wir zunächst eine einzige Binärstelle i mit den Operandenbits A_i, B_i. Auf welche Weise kann ein Übertrag zustande kommen?

1. er kann unabhängig von einem einlaufenden Übertrag *entstehen*. Diese Bedingung bezeichnet man als CARRY GENERATE G_i. Der Übertrag entsteht, wenn beide Operandenbits mit Eins belegt sind, also gilt: $G_i = A_iB_i$.
2. es kann ein *einlaufender Übertrag* zum Ausgangsübertrag *durchgereicht werden*. Diese Bedingung bezeichnet man als CARRY PROPAGATE P_i. Sie wird wirksam, wenn wenigstens eines der Operandenbits mit Eins belegt ist (ansonsten würde der einlaufende Übertrag in der Stelle i "verschluckt" (absorbiert) werden). Es gilt: $P_i = A_i \vee B_i$.

Wie wird nun der Übertrag in der ersten Stelle gebildet? Er kann dort entstehen, oder ein einlaufender Übertrag (CARRY INJECT C_i) wird weitergereicht.

Es gilt also: $C_0 = G_0 \vee P_0 C_i$.

Das gilt auch für die zweite Stelle sinngemäß; also: $C_1 = G_1 \vee P_1 C_0$. Für C_0 können wir aber die zuvor genannte Formel einsetzen: $C_1 = G_1 \vee P_1 G_0 \vee P_1 P_0 C_i$.

Entsprechend gilt für die dritte Stelle $C_2 = G_2 \vee P_2 C_1$. Hier können wir die beiden bisher genannten Formeln einsetzen: $C_2 = G_2 \vee P_2 G_1 \vee P_2 P_1 G_0$.

Sinngemäß gilt in einer beliebigen Stelle i:

$C_i = P_i G_{i-1} \vee P_i P_{i-1} G_{i-2} \vee P_i P_{i-1} P_{i-2} G_{i-3} \vee ...$

Abbildung 1.38 zeigt das Prinzip der Übertragsweitergabe in einer Stelle sowie eine Schaltung für vier Stellen.

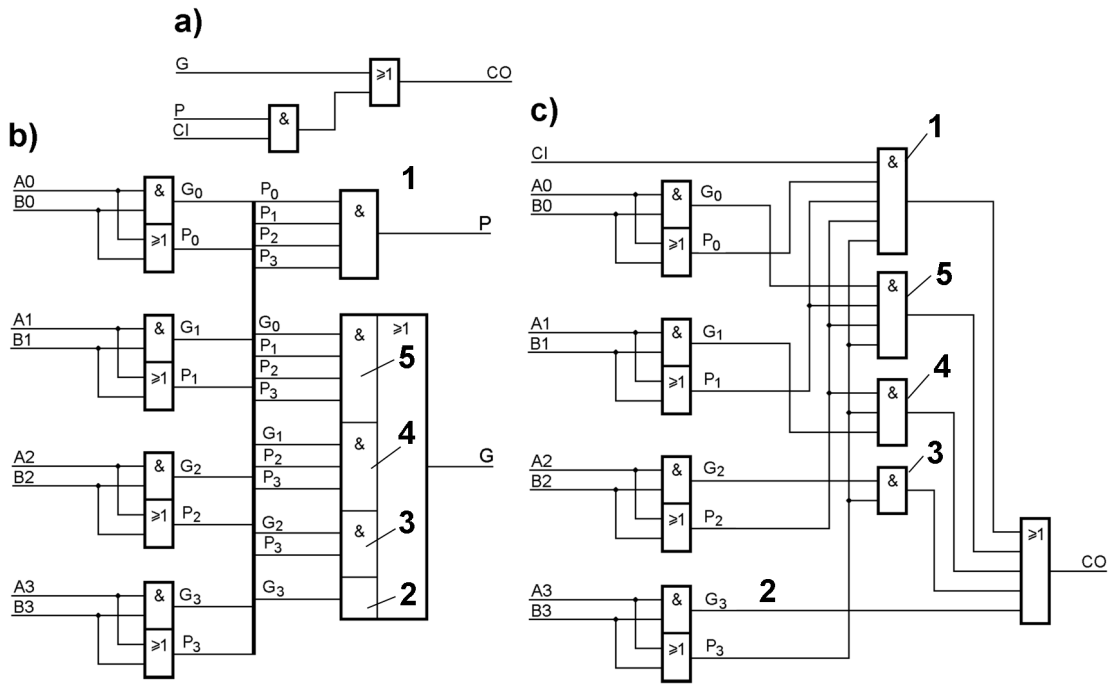


Abbildung 1.38 Übertragsvorausschau (Carry Look Ahead)

Erklärung zu Abbildung 1.38:

a) zum Prinzip. Es ist der Ausgangsübertrag zu bestimmen, der über einen vorgegebenen Bereich von Binärstellen gebildet wird. Dieser Ausgangsübertrag kann zum einen innerhalb dieser Binärstellen entstehen (Carry Generate G). Zum anderen können diese Binärstellen so belegt sein, daß ein einlaufender Übertrag (Carry In CI) durchgereicht wird (Carry Propagate P). Demgemäß ergibt sich der Ausgangsübertrag (Carry Out CO) wie folgt: $CO = G \vee P \cdot CI$.

b) Bildung der Übertragssignale P und G über 4 Binärstellen. Um den eigentlichen Ausgangsübertrag CO zu bilden, ist die Anordung a) nachzuschalten.

c) Bildung des Ausgangsübertrags CO über 4 Binärstellen (Vereinigung von a) und b); Gatter einzeln dargestellt, einige Verknüpfungen des Verbundes a), b) zusammengefaßt).

Näheres zu Abbildung 1.38b, c - Erklärung der einzelnen UND-Verknüpfungen:

1) Übertragsweitergabe. Ein einlaufender Übertrag CI wird nur dann weitergegeben, wenn in allen 4 Stellen jeweils die Weitergabebedingung erfüllt ist ($P_3 \cdot P_2 \cdot P_1 \cdot P_0$ - ansonsten wird der einlaufende Übertrag absorbiert - er verschwindet gleichsam innerhalb der 4 Stellen).
2) ein Ausgangsübertrag wird gebildet, wenn in der 4. (höchstwertigen) Stelle ein Übertrag entsteht (G_3),
3) ein Ausgangsübertrag wird gebildet, wenn in der 3. Stelle ein Übertrag entsteht und dieser über die 4. Stelle weitergegeben wird ($G_2 \cdot P_3$),
4) ein Ausgangsübertrag wird gebildet, wenn in der 2. Stelle ein Übertrag entsteht und dieser über die 3. und die 4. Stelle weitergegeben wird ($G_1 \cdot P_2 \cdot P_3$),
5) ein Ausgangsübertrag wird gebildet, wenn in der 1. Stelle ein Übertrag entsteht und dieser über die 2., 3. und die 4. Stelle weitergegeben wird ($G_0 \cdot P_1 \cdot P_2 \cdot P_3$).

Addierwerke mit großer Verarbeitungsbreite

Bei großen Verarbeitungsbreiten ist es nicht mehr möglich, alle Übertragsbits parallel vorausschauend zu bestimmen: die Netzwerke werden - zu den höherwertigen Stellen hin - immer breiter, so daß man die einzelnen Verknüpfungen ihrerseits aus elementaren Gattern kaskadieren muß (was wiederum längere Durchlaufzeiten zur Folge hat). Deshalb werden solche Addierer gleichsam in Scheiben (Slices) geschnitten, und innerhalb der einzelnen Scheiben bzw. Stufen wird die Übertragsvorausschau verwirklicht. Richtwert zur Breite der einzelnen Stufe: 4...8 Bits sind typisch. Es gibt verschiedene Auslegungen (Abbildungen 1.39 bis 1.41).

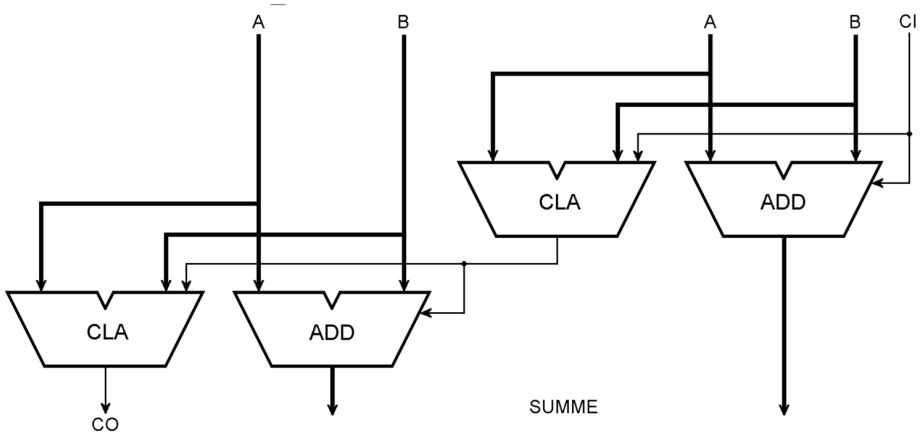


Abbildung 1.39 Kaskadierte Übertragsvorausschau (Carry Skip Adder)

Erklärung:
Jede Stufe des Addierwerks besteht aus einem Ripple-Carry-Addierer ADD (z. B. gemäß Abbildung 1.37) und aus einem Übertragsvorausschaunetzwerk CLA gemäß Abbildung 1.38c. Die jeweils nachfolgende (höherwertige) Stufe erhält einen vorausschauend gebildeten Eingangsübertrag; der von den Addierern selbst gebildete Ausgangsübertrag wird gleichsam übersprungen (deshalb heißen solche Addierer auch Carry Skip Adder). Dieser Verbund liefert den Ausgangsübertrag CO schneller als ein Ripple-Carry-Addierer. Am Beispiel der Abbildungen 1.37 und 1.38c: Schaltungstiefe der vorausschauenden Übertragsbildung: 3, Schaltungstiefe der Bildung von CO als durchlaufender Übertrag (Ripple Carry): wenigstens 8 (wenn die einzelnen Volladdierer zweistufige Netzwerke (Abbildung 1.36b) sind).

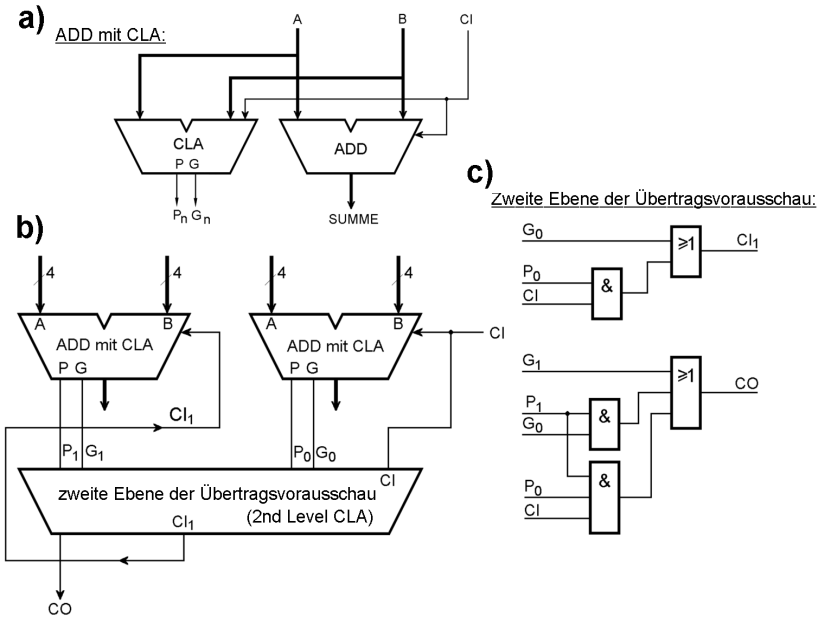


Abbildung 1.40 Zweistufige Übertragsvorausschau (Carry Lookahead Adder)

Erklärung zu Abbildung 1.40:

a) jede Stufe des Addierwerks besteht aus einem Ripple-Carry-Addierer ADD (z. B. Abbildung 1.37) und einem Übertragsvorausschaunetzwerk CLA gemäß Abbildung 1.38b, das die Signale P und G der gesamten Stufe liefert (P = durch die Stufe wird ein ggf. anliegender Eingangsübertrag CI durchgeleitet; G = in dieser Stufe ist ein Ausgangsübertrag entstanden).

b) das Addierwerk besteht aus zwei solchen Stufen (ADD mit CLA), denen ein weiteres Vorausschaunetzwerk nachgeschaltet ist (zweite Ebene der Übertragsvorausschau). Dieses Netzwerk bildet aus den P- und G-Signalen die eigentlichen Übertragssignale, nämlich:
   - den Eingangsübertrag CI_1 der höherwertigen Addierer-Stufe,
   - den Ausgangsübertrag CO des Addierwerks.

c) der Aufbau der 2. Ebene der Übertragsvorausschau:
   - CI_1: der Übertrag in die höherwertige Stufe (CI_1) ergibt sich so, wie in Abbildung 1.38a dargestellt,
   - CO: der Ausgangsübertrag wird gebildet, wenn (1) in der höchstwertigen Addierer-Stufe ein Übertrag entsteht (G_1) oder wenn (2) in der vorgeordneten Stufe ein Übertrag entsteht und durch die nachfolgende Stufe weitergeleitet wird ($G_0 \cdot P_1$) oder wenn (3) ein Eingangsübertrag über beide Stufen weitergereicht wird ($CI \cdot P_0 \cdot P_1$).

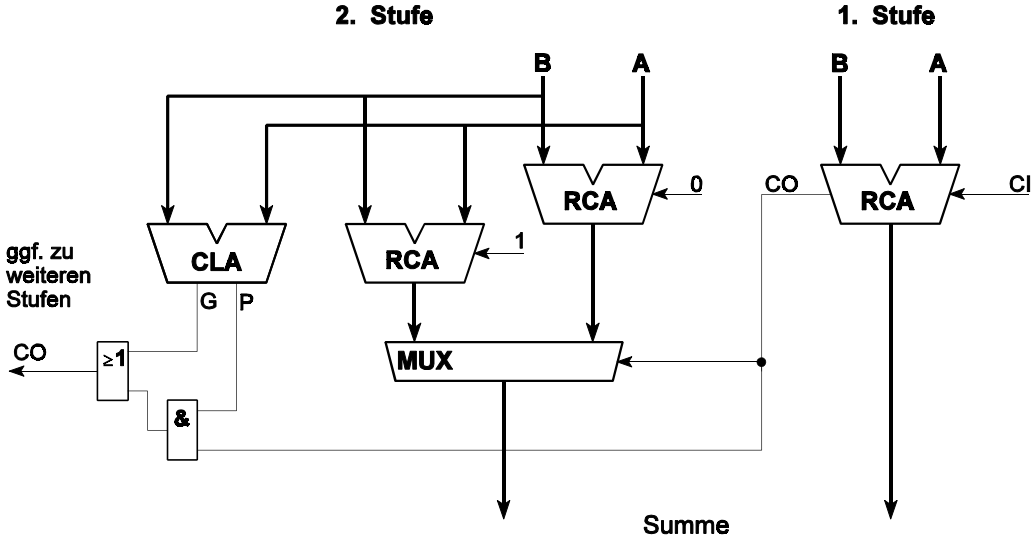


Abbildung 1.41 Übertragsgesteuerte Ergebnisauswahl (Carry Select Adder)

Erklärung:
RCA - Ripple-Carry-Addierer (Abbildung 1.37); CLA - Übertragsvorausschau (Abbildung 1.38b). Die erste Stufe des Addierwerks ist ein gewöhnlicher Addierer mit durchlaufendem Übertrag. Jede weitere Stufe enthält zwei Addierer, deren Summenausgänge an eine Auswahlschaltung (Multiplexer) angeschlossen sind. Der erste dieser Addierer ist mit dem festen Eingangsübertrag Null beschaltet, der zweite mit dem festen Eingangsübertrag Eins. Der Grundgedanke: wir bilden von der zweiten Stufe an zwei Ergebnisse gleichzeitig - einmal ohne und einmal mit Eingangsübertrag. Je nachdem, ob tatsächlich ein Eingangsübertrag eintrifft oder nicht, wählen wir das entsprechende Ergebnis aus (über den Multiplexer). Der Ausgangsübertrag

(CO) dieser Stufen wird typischerweise durch Übertragsvorausschau gebildet (disjunktive Verknüpfung eines in der Stufe entstandenen Übertrags G und eines ggf. weiterzureichenden Eingangsübertrags (Weiterreichbedingung P UND Eingangsübertrag)). Eine dritte Stufe wird dem CO-Ausgang der zweiten ebenso nachgeschaltet wie die zweite dem CO-Ausgang der ersten.

Abschnittsweises Addieren
Kommen die Taktfrequenzen in den GHz-Bereich, so wird es wirklich schwierig, bei größeren Verarbeitungsbreiten die Addition in einem einzigen Taktzyklus zu erledigen. Ein Ausweg: die Addition wird in mehreren Abschnitten nacheinander ausgeführt. Dabei kann die Ausführungszeit der einzelnen Teil-Additionen sogar verkürzt werden. Beispiel: Staggered Add beim Pentium-4-Prozessor (Abbildung 1.42).

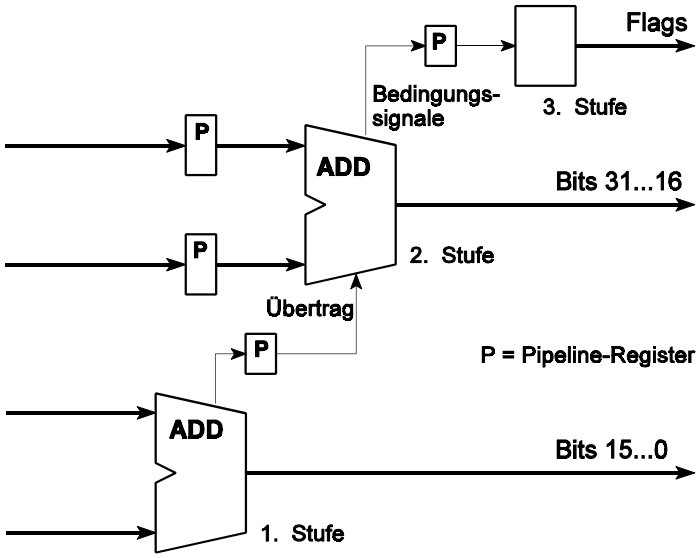


Abbildung 1.42 Abschnittsweises Addieren: Staggered Add (nach: Intel)

Erklärung:
32-Bit-Worte werden nacheinander in zwei 16-Bit-Abschnitten addiert. Das Addierwerk arbeitet dabei mit dem Doppelten der internen Taktfrequenz. In jeder halben Taktperiode wird ein 16-Bit-Teilergebnis erzeugt. Insgesamt dauert die Addition $^3/_2$ Taktperioden: Addieren der niederwertigen 16 Bits → Addieren der höherwertigen 16 Bits → Stellen der Flagbits. Eine weitere Besonderheit: befindet sich ein weiterer, nachfolgender Befehl in der Ausführungsphase, der sich auf das Additionsergebnis bezieht, so kann er bereits nach der Bildung des ersten 16-Bit-Teilergebnisses mit diesem Wert weiterarbeiten (muß also nicht warten, bis der gesamte Additionsablauf beendet wurde).

Addierer ohne Übertragsverrechnung (Carry Save Adder, CSA)

Ein Carry-Save-Addierer für n Stellen ist nichts anderes als die Nebeneinander-Anordnung von n Volladdierern, ohne daß die Überträge von Stelle zu Stelle weitergegeben werden. Eine solche Schaltung (Abbildung 1.43) hat 3 Eingangs-Operanden (1. Summand, 2. Summand, Eingangsübertrag) und liefert 2 Ergebnisse (Summe und Ausgangsübertrag). Der Vorteil: geringste Schaltungstiefe (im Extremfall: 2; vgl. Abbildung 1.36b), und zwar unabhängig von der Verarbeitungsbreite. Das gewährleistet geringste Verzögerungszeiten. Wozu ist eine solche Anordnung brauchbar? - Für die einzelne Addition hat das Nicht-Verrechnen der Überträge keinen Sinn. Der Vorteil kommt vielmehr dann zur Wirkung, wenn eine Vielzahl von Additionen nacheinander auszuführen ist und dazu mehrere Addierer zusammengeschaltet werden müssen (das typische Beispiel ist die Multiplikation; vgl. Abschnitt 1.3.10.). Man bildet dann die Zwischensummen mit Carry-Save-Addierern und sieht nur für die Schlußrechnung einen "richtigen" Addierer (mit Übertragsverrechnung) vor, der Summen und Einzel-Überträge zum Endergebnis aufaddiert.

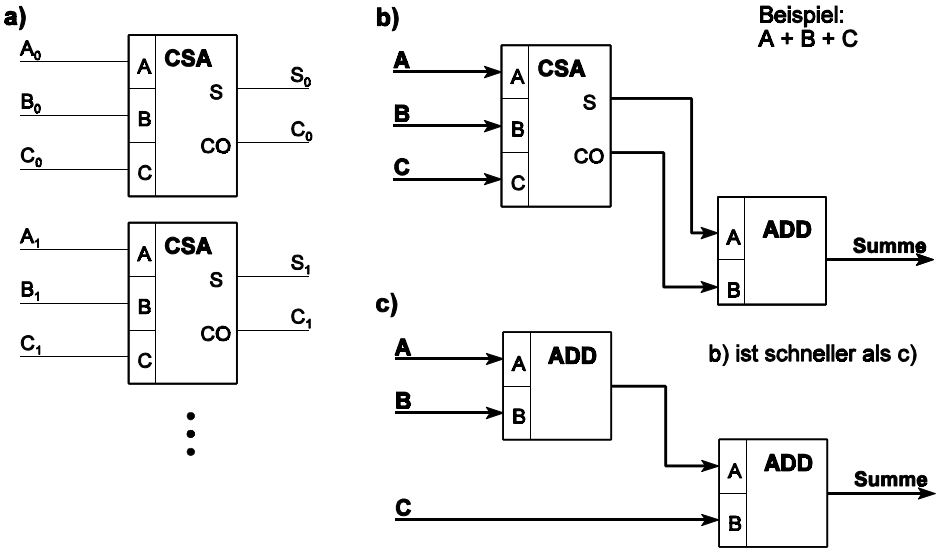


Abbildung 1.43 Carry-Save-Addierer

Erklärung:

a) Addierwerk aus Carry-Save-Addierern (= Volladdierern, die ohne Übertragsweitergabe betrieben werden),

b) Anwendungsbeispiel: die 3-Operanden Addition (A + B + C). Es genügt ein Carry-Save-Addierer, dem ein gewöhnlicher Addierer zwecks Übertragsverrechnung nachgeschaltet ist.

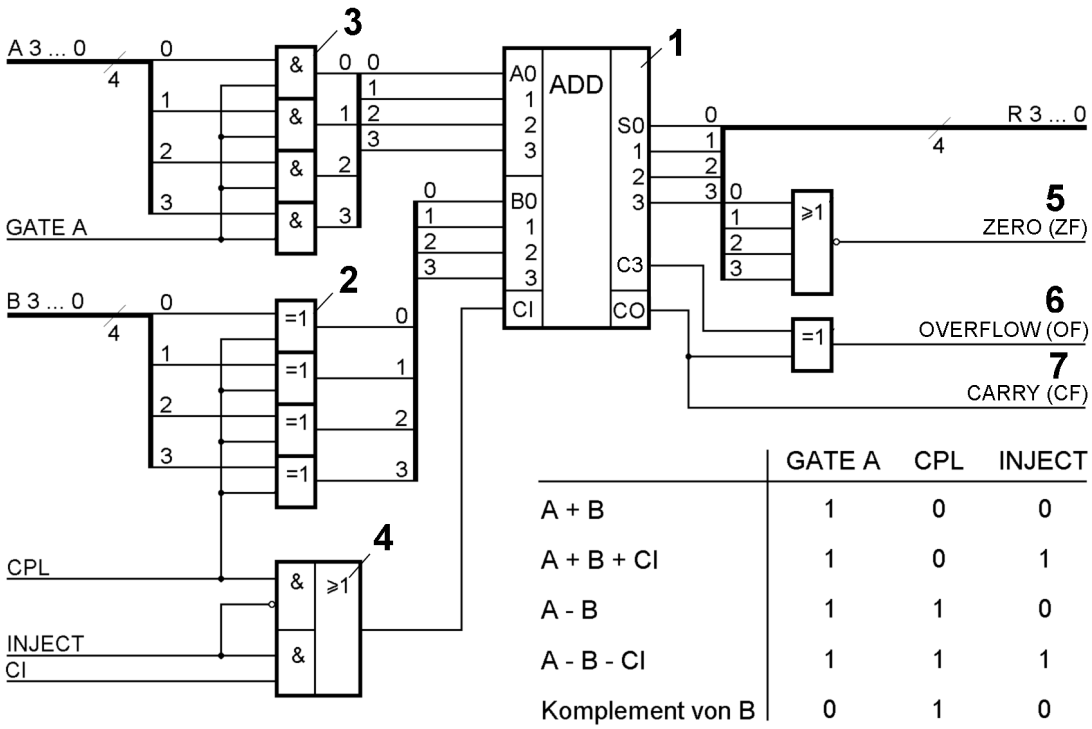
c) zum Vergleich: 3-Operanden Addition (A + B + C) mit zwei hintereinandergeschalteten herkömmlichen Addierern. Rechnung (A + B) + C. Die Lösung b) ist deutlich schneller - und zudem weniger aufwendig.

Subtraktion im Zweierkomplement

Die Bildung des Zweierkomplements erfordert (1) die bitweise Negation des Operanden und (2) das Addieren einer Eins. Letzteres läßt sich durch Einspeisen eines Eingangsübertrags erreichen. Eine Schaltung, die addieren und subtrahieren kann, läßt sich auf Grundlage eines Addierers mit vorgeschalteten Antivalenzgattern zum bedarfsweisen Negieren aufbauen.

1.3.6. Die universelle Arithmetikeinheit

Eine solche Einrichtung soll - typischerweise als Teil einer ALU - Additions- und Subtraktionsoperationen mit natürlichen und ganzen Binärzahlen ausführen können (Abbildungen 1.44, 1.45).



	GATE A	CPL	INJECT
A + B	1	0	0
A + B + CI	1	0	1
A - B	1	1	0
A - B - CI	1	1	1
Komplement von B	0	1	0

Abbildung 1.44 Universelle 4-Bit-Arithmetikeinheit. Erklärung im Anschluß an Abbildung 1.45

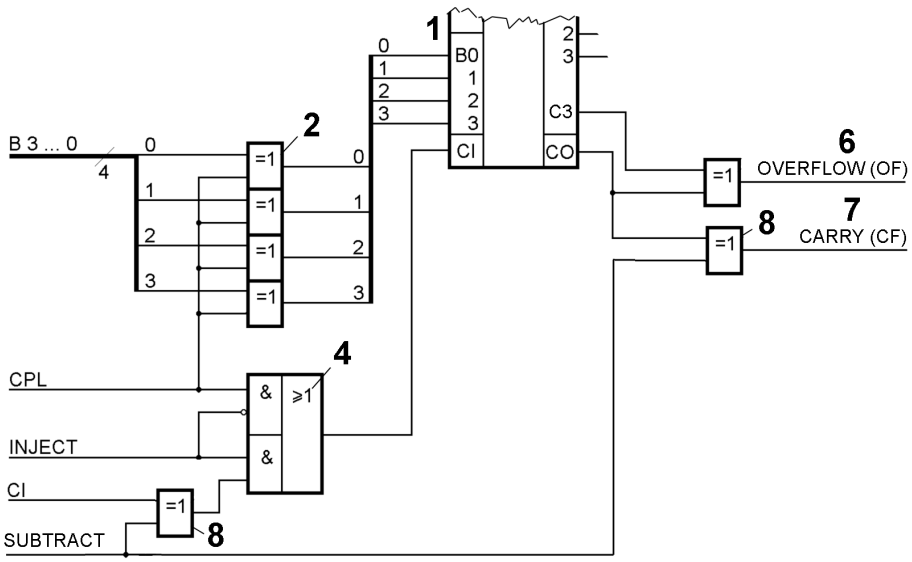


Abbildung 1.45 Eine Spitzfindigkeit ...

Erklärung zu den Abbildungen 1.44 und 1.45:
1 - Addierwerk; 2 - XOR-Gatter zum Bilden des Zweierkomplements; 3 - UND-Gatter zum Ausblenden des 2. Operanden (nur erforderlich für die Operation "Zweierkomplement von B"); 4 - Steuerung Eingangsübertrag; 5 - Ergebnis = Null; 6 - Überlaufbedingung*); 7 - Ausgangsübertrag (Carry Flag); 8 - Komplementierung von Übertragssignalen. Die Signale 5...7 werden typischerweise in ein Flagregister übernommen und sind dort mit Verzweigungsbefehlen abfragbar.

*): zum Rechnen mit vorzeichenbehafteten Binärzahlen. Überlauf = Übertrag in die höchstwertige Binärstelle (C3) ⊕ Ausgangsübertrag).

Der Eingangsübertrag
Man braucht insgesamt drei Möglichkeiten, um den Eingangsübertrag zu bilden:

1. fest Null zum Addieren,
2. fest Eins zum Subtrahieren (Zweierkomplementbildung),
3. Einspeisung eines geretteten Übertrages (Carry Inject) zum Rechnen mit Binärzahlen, die länger sind als die Verarbeitungsbreite*).

*): sind die Zahlen länger als die Verarbeitungsbreite, so kann man sie trotzdem addieren oder subtrahieren, und zwar gewissermaßen stückweise, mit den niederwertigen Stellen beginnend. Der Ausgangsübertrag muß dabei gespeichert und bei der nachfolgenden Addition oder Subtraktion wieder als Eingangsübertrag verwendet werden. Um solche Abläufe programmieren zu können, gibt es Additions- und Subtraktionsbefehle sowohl ohne als auch mit Einspeisen des (im Carry-Flag CF) geretteten Übertrags.

Zu Abbildung 1.45
Beim schulmäßigen Addieren und Subtrahieren *natürlicher* (vorzeichenloser) Binärzahlen zeigt der Ausgangsübertrag das Über- oder Unterschreiten des Wertebereichs an. Viele Architekturen (darunter auch IA-32) verwalten das Übertrags-Flag (CF) in diesem Sinne. Die Schaltung von Abbildung 1.44 subtrahiert aber im Zweierkomplement. Dabei verhält sich der Ausgangsübertrag genau andersherum. Die Einzelheiten sind aus Tabelle 1.3 ersichtlich.

Rechen-operation	**Ausgangsübertrag gemäß Zweierkomplementarithmetik**	**Übertrags-Flag CF in vielen Architekturen**
Addition A + B	▪ 0, wenn Ergebnis im Wertebereich, ▪ 1, wenn Ergebnis Wertebereich überschreitet	▪ 0, wenn Ergebnis im Wertebereich, ▪ 1, wenn Ergebnis außerhalb des Wertebereichs
Subtraktion A - B	▪ 1, wenn Ergebnis im Wertebereich (≥ 0), ▪ 0, wenn Ergebnis Wertebereich unterschreitet (< 0)	

Tabelle 1.3 Ausgangsübertrag und Übertrags-Flag beim Rechnen mit natürlichen Binärzahlen

Wir merken uns:
Bei den meisten der gängigen Architekturen entspricht das Übertrags-Flag (Carry Flag CF) nicht dem Zweierkomplement-Ausgangsübertrag, sondern dient als generelle Überlaufanzeige.

Beim Addieren natürlicher Binärzahlen entsprechen Ausgangsübertrag und Überlauf (Bereichsüberschreitung) einander, beim Subtrahieren nicht (der Übertrag tritt vielmehr dann auf, wenn der Wertebereich *nicht* unterschritten wird). Abbildung 1.45 veranschaulicht die typische Schaltungslösung: beim Subtrahieren (SUBTRACT) wird der Ausgangsübertrag über ein XOR-Gatter invertiert. Da beim abschnittsweisen Rechnen der eingespeiste Eingangsübertrag dem Ausgangsübertrag der vorhergehenden Subtraktion entsprechen muß, wird diese Negation durch ein weiteres XOR im CI-Signalweg wieder rückgängig gemacht.

1.3.7. Laden mit Vorzeichenerweiterung

Ist eine ganze Binärzahl kürzer als die Verarbeitungsbreite, so muß beim Auffüllen das Vorzeichen berücksichtigt werden: es ist in alle höherwertigen Stellen zu übernehmen (Vorzeichenerweiterung; Sign Extend). Abbildung 1.46 zeigt eine entsprechende Schaltung (die beispielsweise der Arithmetikeinheit gemäß Abbildung 1.44 vorgeschaltet werden kann).

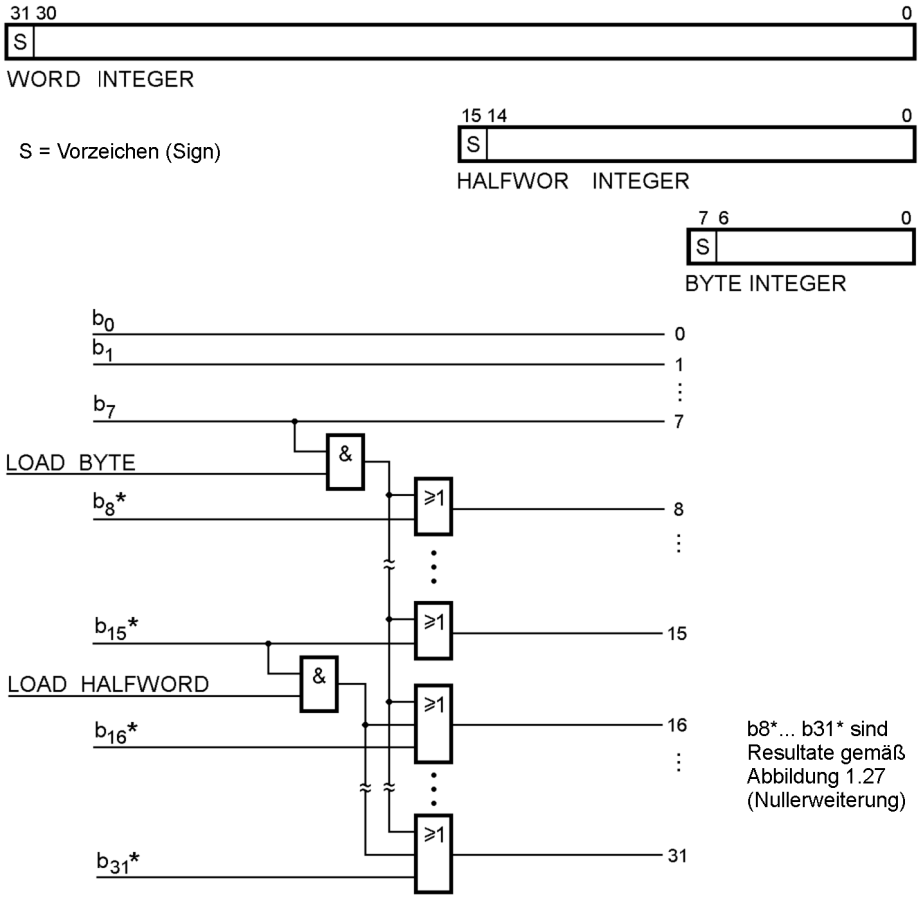


Abbildung 1.46 Vorzeichenerweiterung (Sign Extend)

Erklärung zu Abbildung 1.46:
Die ODER-Gatter dienen dazu, das jeweilige Vorzeichen (Bit 7 beim Laden eines Bytes, Bit 15 beim Laden eines 16-Bit-Halbworts) in die jeweils nachfolgenden höherwertigen Stellen einzufügen. Voraussetzung ist, daß die höherwertigen Stellen Nullen führen (was sich durch Vorschalten der Nullerweiterung gemäß Abbildung 1.27 gewährleisten läßt - vgl. die folgende Abbildung 1.47).

1.3.8. Arithmetische Verschiebungen

Arithmetische Verschiebungen unterscheiden sich nur in einem Punkt von den logischen Verschiebungen, die wir in Abschnitt 1.3.3. behandelt haben: beim *Rechts*verschieben wird nicht die Null oder ein Füllwert, sondern das Vorzeichen (also die Belegung der höchstwertigen Stelle) in alle freiwerdenden Stellen eingetragen (Vorzeichenerweiterung). Beim Linksverschieben gibt es keine Unterschiede.

1.3.9. Die ALU

Die Arithmetik-Logik-Einheit besteht aus einem Verbund der bisher beschriebenen Schaltungen (Abbildung 1.47).

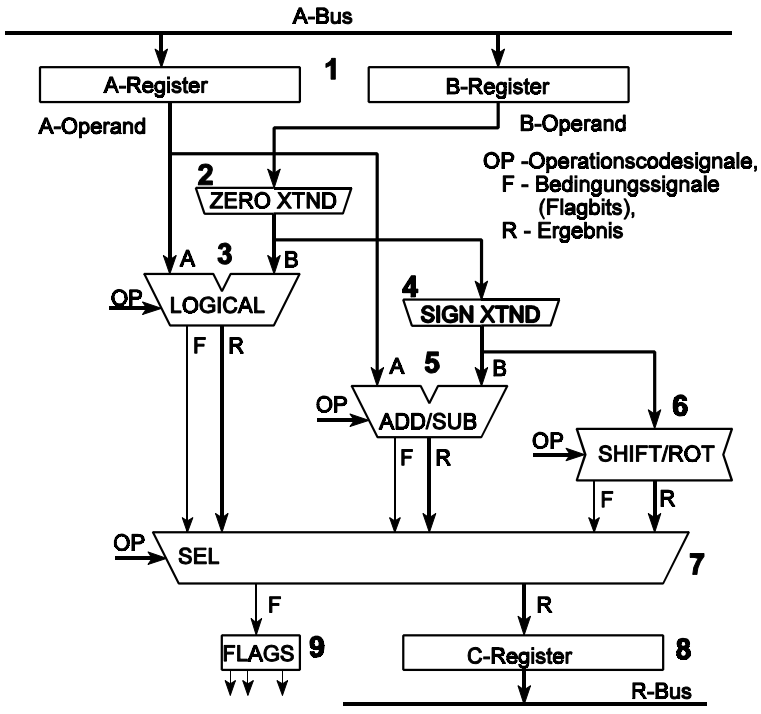


Abbildung 1.47 Arithmetik-Logik-Einheit (ALU) im Blockschaltbild

Erklärung:
Wir zeigen hier eine herkömmliche ALU (kein Pipelining), die in einen herkömmlichen Prozessor eingebaut ist. Die kombinatorischen Netzwerke sind zwischen zwei Registerstufen angeordnet. 1 - Operandenregister; 2 - Nullerweiterung (Abbildung 1.27); 3 - logische

Verknüpfungen und Transporte (Abbildung 1.26); 4 - Vorzeichenerweiterung (Abbildung 1.46); 5 - Addition und Subtraktion (Abbildung 1.44); 6 - Verschieben und Rotieren (Abbildungen 1.29 bis 1.31); 7 - Ergebnisauswahl; 8 - Ergebnisregister; 9 - Flagregister.

1.3.10. Multiplikation und Division

In der Hardware laufen solche Operationen zumeist in mehreren Maschinenzyklen ab. Das gilt für die Multiplikation in vielen Fällen, für die Division praktisch immer (auf Grund des Verhältnisses von Nutzungshäufigkeit und Aufwand lohnt sich für die Division keine besondere Hochleistungs-Hardware). Die Abläufe werden in vielen Prozessoren durch ein internes Mikroprogramm gesteuert.

Hardware-Multiplizierer
Die Multiplikation läuft praktisch darauf hinaus, gegeneinander verschobene Multiplikandenwerte aufzuaddieren. Jedem Multiplikatorbit entspricht dabei ein Multiplikandenwert. Dessen Rechtsverschiebung entspricht dem Index (der Bitadresse) des jeweiligen Multiplikatorbits (Bit 0: keine Verschiebung, Bit 1: Verschiebung um 1 Bit usw.). Ist das Multiplikatorbit gleich Eins, wird der betreffende verschobene Multiplikandenwert aufaddiert, ansonsten nicht.

Dieses Prinzip läßt sich direkt in eine Schaltung umsetzen (Abbildung 1.48). Es ist ein "Baum" von Addierern vorgesehen, wobei den obersten Addierern der Multiplikand über UND-Verknüpfungen vorgeschaltet ist, die jeweils alle mit der entsprechenden Bitposition des Multiplikators verbunden sind (je nach deren Belegung wird also entweder der Multiplikand oder eine Null auf den Addierer-Eingang geschaltet). Die nachfolgenden Addierer dienen dazu, die Teilprodukte zu summieren. Die notwendige Verschiebung wird durch entsprechend versetztes Anschließen der jeweils nachgeordneten Addierer gewährleistet.

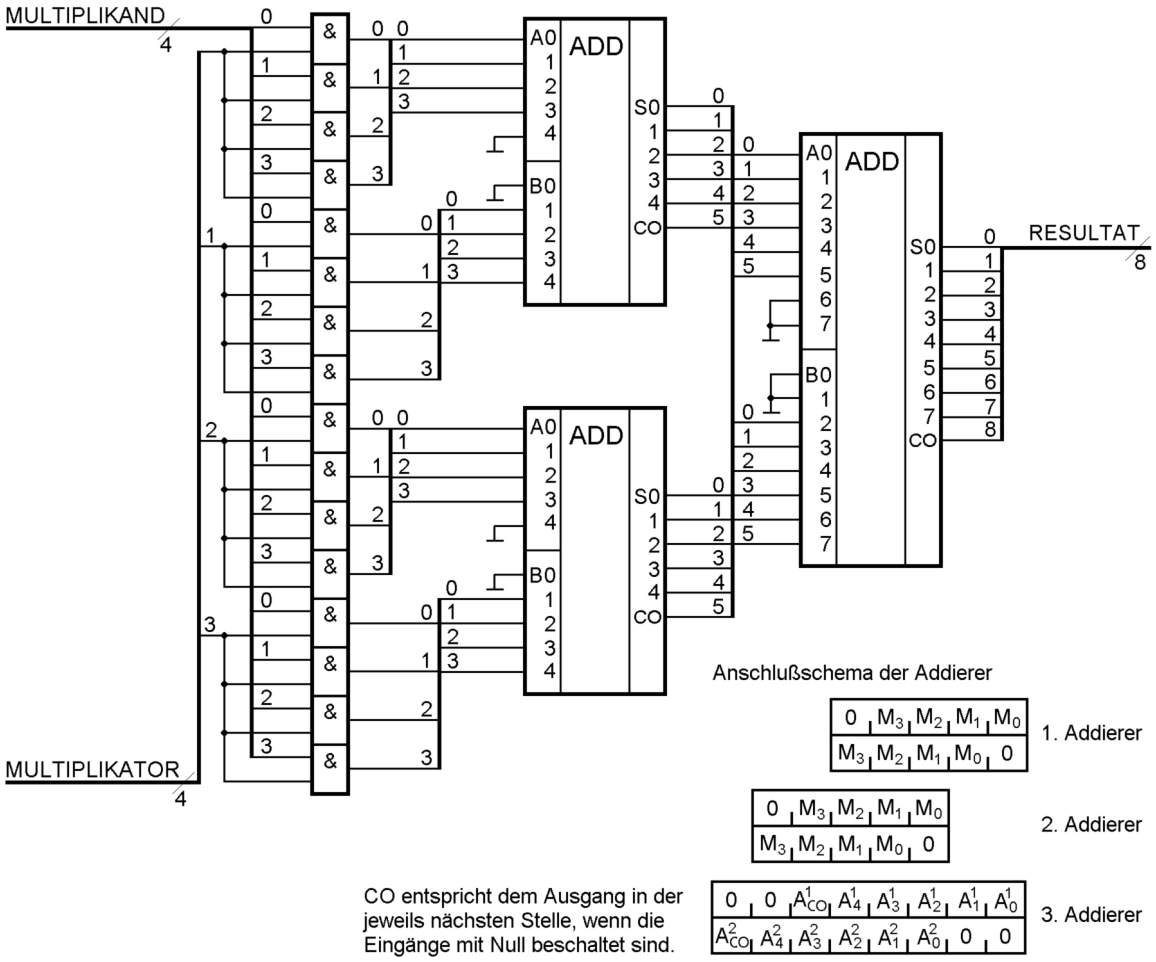


Abbildung 1.48 4-Bit-Binärmultiplizierer

Multiplizieren mit Carry-Save-Addierern

Der als Addiererbaum aufgebaute Binärmultiplizierer nach Abbildung 1.48 ist ein gutes Modell, das wirklich funktioniert (ausprobieren!) und das Wesentliche veranschaulicht. Der Nachteil: er ist - angesichts des ohnehin getriebenen Aufwands - einfach noch zu langsam. Der Grund: die Verzögerungszeit jeder Addierer-Schicht wird durch die Verzögerungszeit der Übertragsrechnung bestimmt. Ein Ausweg: man verwendet in den oberen Schichten Carry-Save-Addierer und setzt nur zur Schlußrechnung einen gewöhnlichen Addierer ein (Abbildung 1.49).

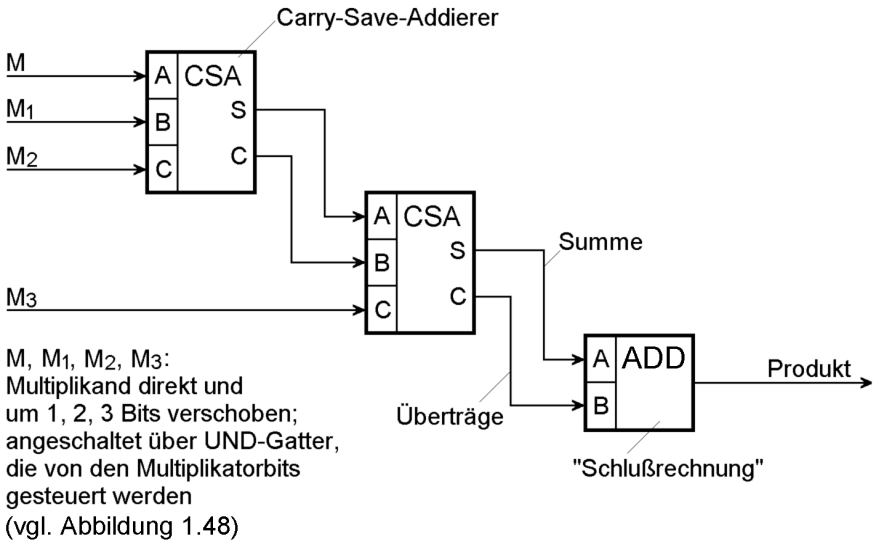


Abbildung 1.49 Multiplizierer mit Carry-Save-Addierern

1.3.11. Gleitkommaoperationen

Eine Gleitkommazahl besteht aus Vorzeichen, Exponent und Signifikand (Mantisse). Vorzeichen und Wert sind voneinander unabhängig (Sign-Magnitude-Darstellung). Der Exponent wird vorzeichenlos dargestellt. Die codierte Angabe ist dabei um einen Versatz (Bias) gegenüber dem tatsächlichen Wert des Exponenten erhöht.

Multiplikation und Division
Bei Multiplikation und Division können die Exponenten und Signifikanden unabhängig voneinander verarbeitet werden. Die Exponenten werden zueinander addiert bzw. voneinander subtrahiert; die Signifikanden werden miteinander multipliziert bzw. durcheinander dividiert. (Potenzen zur gleichen Basis werden multipliziert, indem man ihre Exponenten addiert; sie werden dividiert, indem man ihre Exponenten subtrahiert.)

Addition und Subtraktion
Potenzen mit verschiedenen Exponenten kann man nicht zueinander addieren oder voneinander subtrahieren. Dazu müssen vielmehr die Exponenten beider Operanden gleich sein. Bei Gleitkommazahlen müssen wir also die Mantissen so verändern, daß beide Exponenten gleich sind (*Normalisierung*). Um die Aufgabe $s_1 \cdot 2^b + s_2 \cdot 2^c$ zu lösen, können wir beispielsweise den ersten Operanden unverändert lassen und die Zweierpotenz im zweiten Operanden folgendermaßen umschreiben: $2^c = 2^{(c-b)} \cdot 2^b$, denn es gilt $2^{(c-b)+b} = 2^c$ (Potenzmultiplikation bedeutet Exponentenaddition).

Vor der eigentlichen Addition müssen wir den Ausdruck $2^{(c-b)}$ noch mit dem Signifikanden s_2 multiplizieren. Also in Vorbereitung einer Addition erst eine Multiplikation? - So aufwendig ist das gar nicht: Eine Multiplikation mit einer positiven Zweierpotenz ist nichts anderes als eine Linksverschiebung, eine Multiplikation mit einer negativen Zweierpotenz eine Rechtsverschiebung. Wir müssen somit den Signifikanden s_2 nur gemäß der Exponentendifferenz (also um c-b Bits) in die entsprechende Richtung verschieben.

Wir sehen, daß auch die Gleitkommaverarbeitung auf Grundschaltungen beruht, die wir bereits kennengelernt haben. Abbildung 1.50 gibt einen Überblick über Hardware-Strukturen für die vier Grundrechenarten.

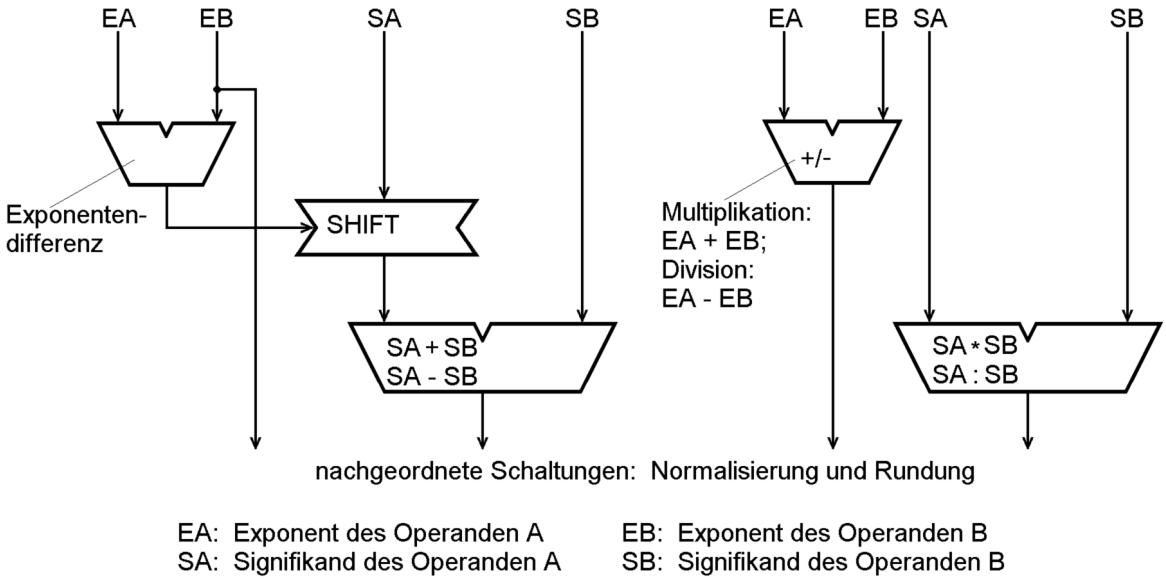


Abbildung 1.50 Hardware zum Rechnen mit Gleitkommazahlen