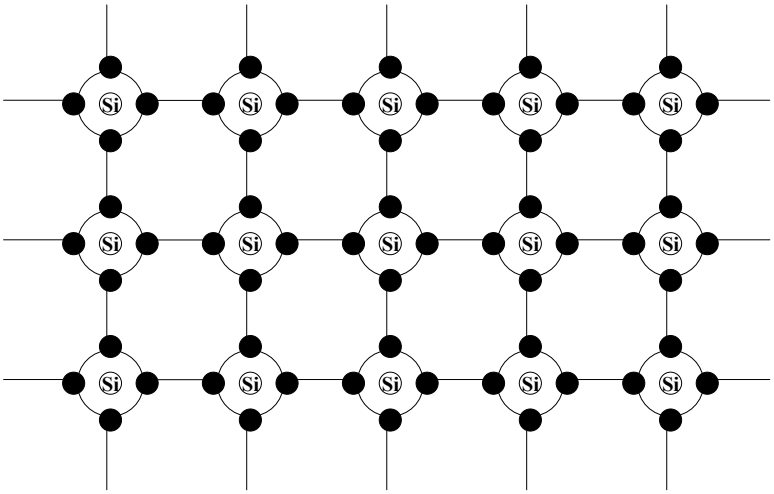
3 Realisierung von Booleschen Funktionen in CMOS-Technologie

3.1 Halbleiterdioden

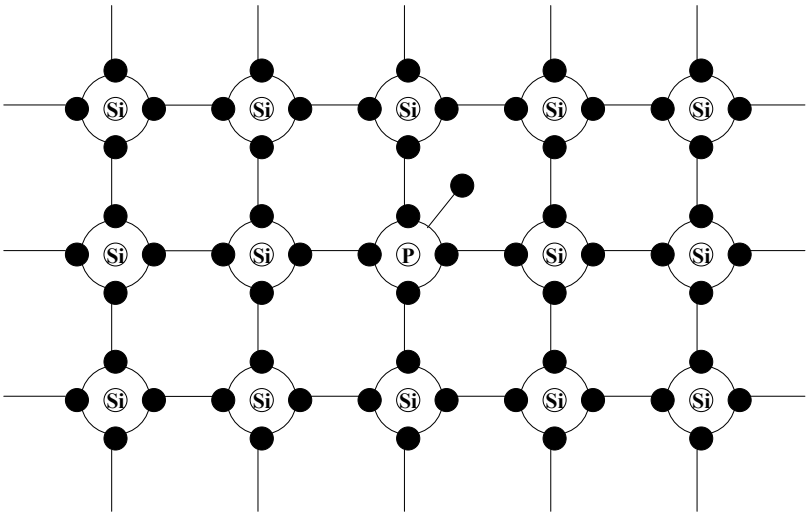
Halbleiterdioden sind Bauelemente, die die Leitfähigkeitseigenschaften eines **pn-Übergangs** nutzen. Sie werden meist aus Silizium hergestellt. Ein pn-Übergang ist der Übergang von positiv dotiertem (p-) Silizium zu negativ dotiertem (n-) Silizium. Dotierung ist das gezielte Einfügen von Fremdatomen in die Kristallstruktur des Siliziums. Silizium ist vierwertig, und bildet in reinster Form eine Kristallstruktur, bei der je zwei Elektronen zweier benachbarter Atome eine Bindung eingehen. Man kann sich das als ein regelmäßiges rechteckiges Gitter vorstellen, wie es auf dem nächsten Bild gezeigt ist. Es enthält so keine freien Elektronen und ist daher auch kaum leitfähig.

Bild: Silizium-Kristallgitter



Wenn man in diesem Gitter nun ein Atom durch ein 5-wertiges Fremdatom ersetzt, z.B. Phosphor, so gibt es zusätzlich zu dem vollständigen Gitter ein freies Elektron, das keine Bindung eingehen kann.

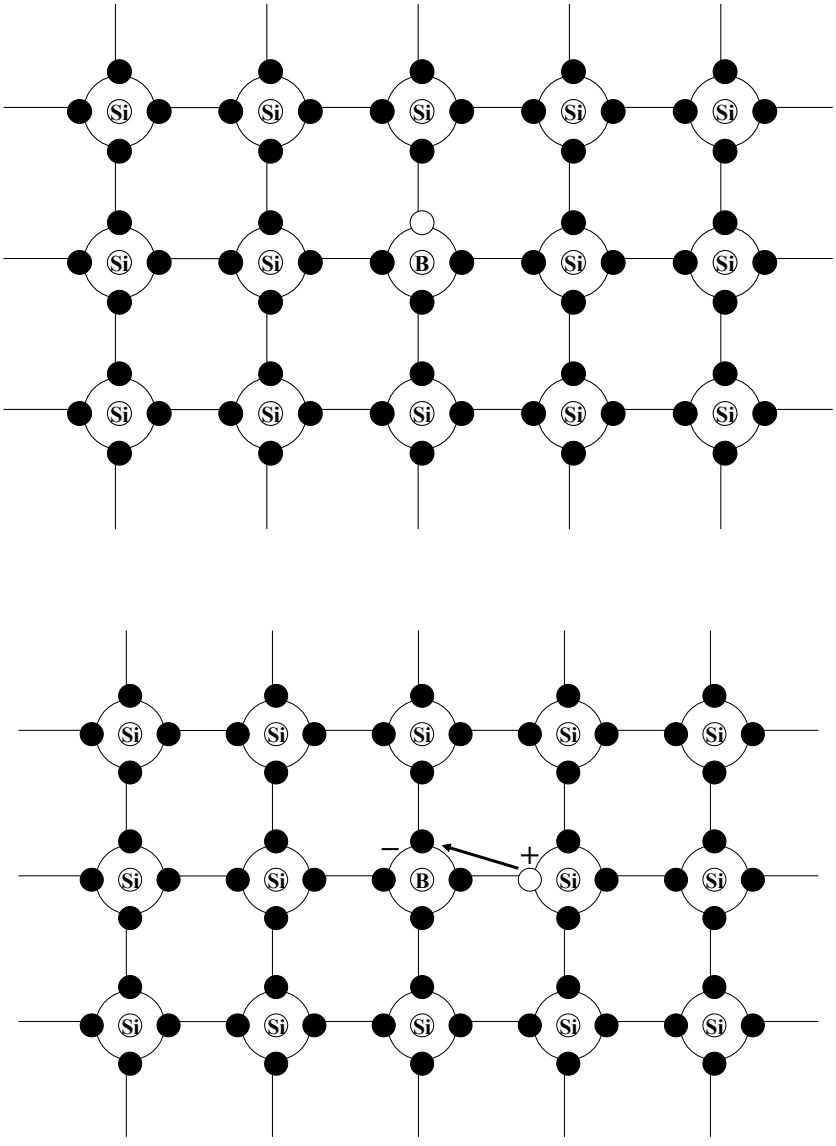
Bild: Negativ dotiertes Silizium



Dieses freie Elektron kann jetzt zur Leitung von elektrischem Strom benutzt werden. Durch **Dotierung von Silizium mit Phosphor** erzeugt man somit aus dem zunächst schlecht leitenden Silizium einen Leiter. Aus diesem Grund bezeichnet man Silizium als **Halbleiter**. Die Dotierung mit Phosphor generiert freie Elektronen im Silizium. Da diese negativ geladen sind, spricht man von einer **n-Dotierung**, gelegentlich auch von einem **n-Halbleiter**.

Verwendet man anstelle von Phosphor ein dreiwertiges Element, z.B. Bor, so kommt man zu einer **p-Dotierung**. Ein entsprechendes Kristallgitter ist im folgenden Bild zu sehen. Es gibt allerdings einen qualitativen Unterschied. Im p-dotierten Material fehlen Elektronen im Kristallgitter. Das fehlende Elektron eines Atoms kann durch ein Elektron eines Nachbaratoms ersetzt werden, wenn dieses aus seiner Paarbindung herausgelöst wird. Es entsteht dort eine **positive Ladung**, ein **Loch** oder **Defektelektron**. Das Fremdatom (Bor), das jetzt ein Elektron aufgenommen hat, wird negativ geladen, bleibt aber ortsfest.

Bild: Positiv dotiertes Silizium, Mobilität der „Löcher“



Durch diesen Vorgang des Ersetzens eines fehlenden Elektrons durch ein Nachbarelektron **wandert das Loch** nun im Kristallgitter. Es kann also ebenfalls genutzt werden, um elektrischen Strom zu transportieren. Allerdings ist die Beweglichkeit der Löcher im p-

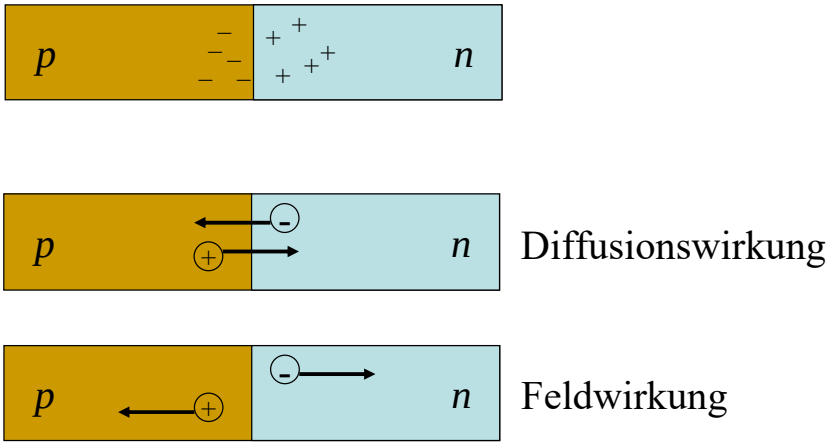
dotierten Halbleiter nicht so groß wie die Beweglichkeit der freien Elektronen im n-dotierten Halbleiter, weil die Elektronen im p-dotierten Material ja zuerst aus ihrer bestehenden Bindung herausgelöst werden müssen. Als Daumenregel kann man sich merken, dass n-dotiertes Silizium etwa eine dreimal so hohe Leitfähigkeit hat wie p-dotiertes.

Wenn man nun eine p-Dotierung direkt an eine n-Dotierung angrenzen lässt, entsteht ein **pn-Übergang**. Wegen der freien Elektronen in der n-Zone und der (frei beweglichen) Löcher in der p-Zone entsteht an der Grenze eine spezielle Reaktion: Freie Elektronen diffundieren in die p-Zone und Löcher in die n-Zone, wo sie rekombinieren. Dadurch verringert sich die Zahl der freien Ladungsträger in der Grenzschicht. Die ladungsträgerfreie Grenzschicht wird zu einer hochohmigen sogenannten Sperrschicht.

Durch die Diffusion der Elektronen in die Sperrschicht bleiben aber ortsfeste, positive Ionen (**sogenannte Raumladungen**) zurück, und durch Rekombination der Löcher mit den Elektronen entsehen in der p-Zone ortsfeste negative Ionen.

Zwischen der positiven Raumladung und der negativen Raumladung entsteht ein elektrisches Feld. Auf freie Ladungsträger innerhalb der Raumladungszone wirkt die Diffusion und in entgegengesetzter Richtung die elektrische Feldkraft.

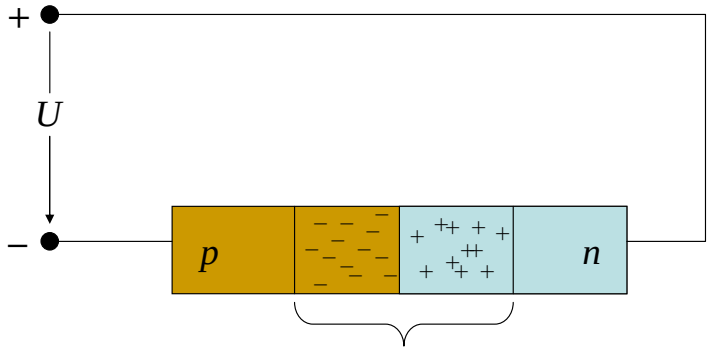
Bild: pn-Übergang (oben) und Kräfte auf Ladungsträger am pn-Übergang (mitte und unten)



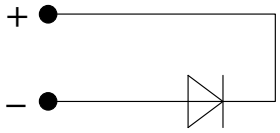
Es stellt sich ein dynamisches Gleichgewicht am pn-Übergang ein, wenn die Feldwirkung und die Diffusionswirkung gleich groß ist. Dann besteht zwischen der positiven Raumladung in der n-Zone und der negativen Raumladung in der p-Zone eine feste Spannung, die Diffusionsspannung U_D. Diese beträgt bei Silizium etwa 0,75 V.

Wenn man an eine Diode eine Gleichspannung anlegt, wird sie – je nach Polung der Gleichspannung – **leitend oder sperrend**. Wenn der Minuspol der Spannungsquelle an die p-Zone und der Pluspol and die n-Zone der Diode gelegt wird, steigt die Spannung in der Raumladungszone auf U_D+U. Die Feldstärke wird größer und die ladungsträgerfreie Raumladungszone wird breiter. Sie wird zu einer hochohmigen Sperrschicht; man sagt, die Diode ist in Sperrrichtung gepolt.

Bild: Diode in Sperrrichtung

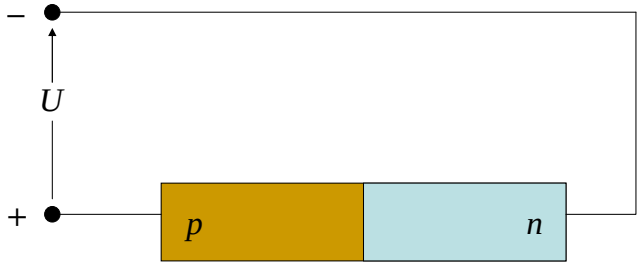


Raumladungszone ohne
freie Elektronen oder Löcher

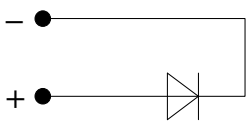


Wenn umgekehrt der Minuspol der Spannungsquelle an die n-Zone und der Pluspol and die p-Zone der Diode gelegt wird, sinkt die Spannung in der Raumladungszone auf U_D-U. Die ladungsträgerfreie Raumladungszone wird schmaler und verschwindet ganz, wenn $U>U_D$ ist. (Der Wert von U, für den das gilt, wird auch **Schwellspannung** genannt). Dadurch ist die Diode leitend, weil jetzt auf dem ganzen Weg von + nach – genügend freie Ladungsträger sind.

Bild: Diode in Durchlassrichtung



Keine Raumladungszone, freie
Elektronen und Löcher vorhanden



Durch Kombination von zwei pn-Übergängen kann man Bipolartransistoren aufbauen. Da diese aber in der digitalen Schaltungstechnik gegenwärtig keine große Bedeutung mehr haben, werden sie hier nicht behandelt. Stattdessen konzentrieren wir uns auf Schaltelemente, die heute und sicher auch noch in Zukunft die bedeutendste Rolle im Aufbau von Digitalschaltungen haben, die MOS-Transistoren.

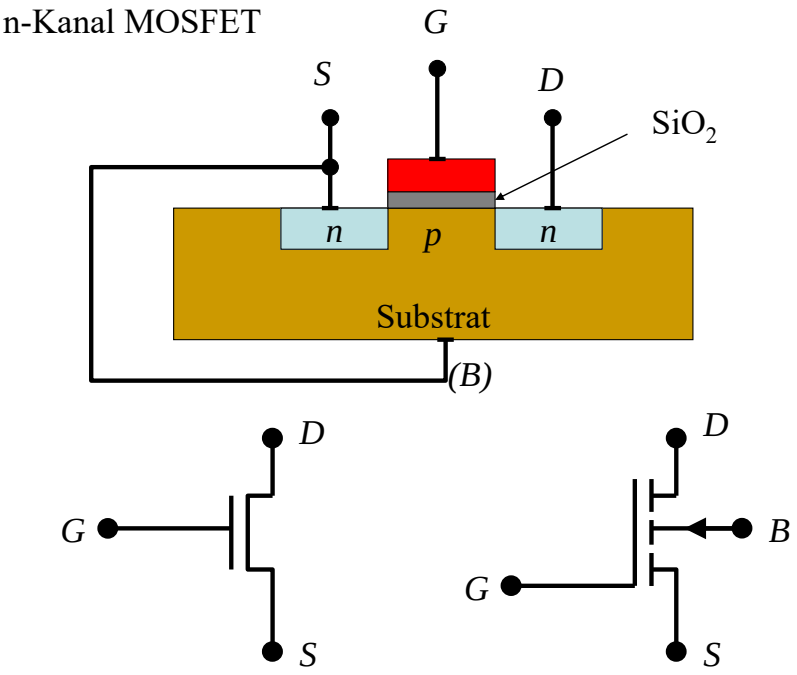
3.2 MOS-Transistoren

Die Abkürzung **MOSFET** steht für **Metal-Oxide-Semiconductor Field Effect Transistor**. Metal-Oxide-Semiconductor bezeichnet die Schichtenfolge, durch die er ursprünglich aufgebaut wurde (heute nimmt man anstelle von Metall in der Regel polykristallines Silizium). Field Effect Transistor bedeutet: Der Transistor-Effekt wird erzielt durch Erzeugen eines elektrischen Feldes durch Anlegen einer Spannung an die Steuerelektrode.

Die drei Anschlüsse eines FETs werden mit **D (Drain), S (Source)** und **G (Gate)** bezeichnet. Das Gate ist die **Steuerelektrode**, auf die man eine Spannung legt, um dadurch eine Verbindung zwischen Drain und Source zu schalten.

Man unterscheidet MOSFETs nach der Art der Dotierung des Halbleitermaterials, in dem (bei geeigneter Beschaltung) der leitende Kanal zwischen Drain und Source entsteht. Wir beginnen mit dem **selbstsperrenden n-Kanal MOSFET**, **(enhancement mode n-channel MOSFET)** dessen Aufbau im folgenden Bild zu sehen ist.

Bild: n-Kanal-MOSFET, Aufbau der Schichten

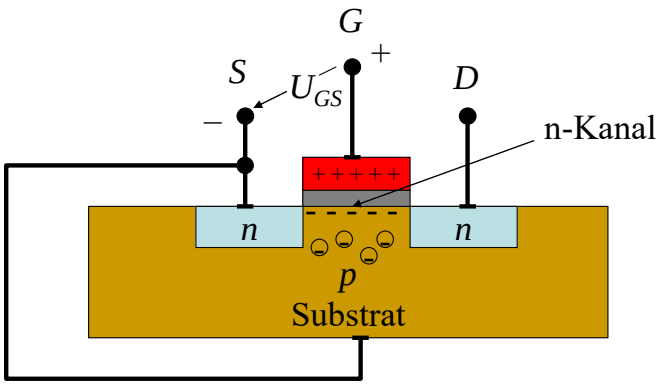


In den als **Substrat** bezeichneten p-Halbleiter sind zwei hochdotierte n-Zonen als Source und Drain eindiffundiert. Sie sind mit dem Source- bzw. Drainanschluss verbunden. Auf das Substrat ist zwischen diesen beiden Zonen eine Isolierschicht aus Siliziumdioxyd aufgebracht. Darüber befindet sich das Gate, das somit isoliert ist gegenüber Source, Drain und Substrat. Da die Oxydschicht allerdings sehr dünn ist, bildet das Gate mit dem Substrat einen Kondensator. Die Zonenfolge Source-Substrat-Drain ist eine npn-Anordnung. Weil der Abstand zwischen den beiden n-Zonen zu groß ist, bildet sich aber kein bipolarer Transistor.

Wird nun an das Gate eine gegenüber dem Substrat positive Spannung angelegt, so werden die Löcher als bewegliche Ladungsträger vom Gate weg in das Substrat abgestoßen. Es entsteht an der Randschicht zum Oxyd hin eine negative Raumladungszone. Wenn das dadurch gebildete

elektrische Feld so groß ist, dass die freien Elektronen nicht mehr in das Substrat hineindiffundieren, so bildet sich am Rand der Raumladungszone eine leitende Schicht aus freien Ladungsträgern (Elektronen). Diese wird **n-Kanal** genannt.

Bild: Kanalbildung im n-Kanal-MOSFET



Die Spannung, ab der sich ein leitender Kanal bildet wird **Schwellspanung** (Threshold) $\boldsymbol{U_{th}}$ genannt (im englischen V_{th}).

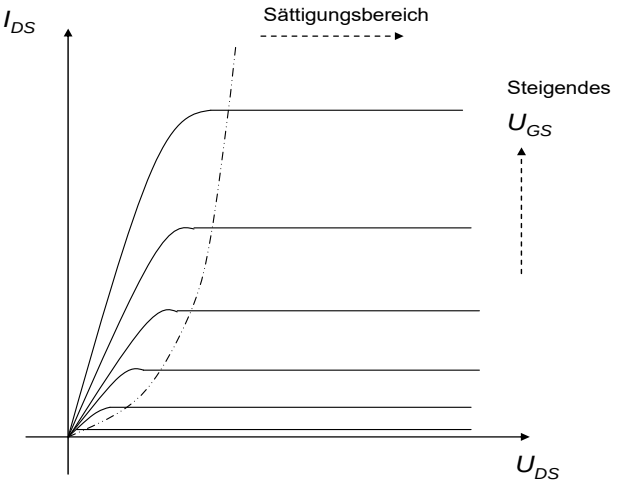
Hat sich ein solch leitender Kanal gebildet, dann kann ein Drainstrom fließen (von Drain nach Source), wenn $U_{DS} > 0V$ ist. Die Abhängigkeit des Drainstroms I_D von U_{DS} und U_{GS} wird als **Kennlinienfeld** dargestellt (nächstes Bild).

Der Substratanschluss wird mit dem Sourceanschluss verbunden und auf das Spannungspotential 0V gelegt . Dies wird als Bezugspotenzial (Bulk, B) genutzt (das im alten deutschen Schaltzeichen auch explizit eingetragen wird).

Im Sperrbereich ist $U_{DS} > 0V$ und $U_{GS} < U_{th}$. Es kann sich kein leitender Kanal aufbauen. Da der Drain-Substrat-Übergang eine in Sperrrichtung beschaltete Diode darstellt, fließt kein Strom I_D.

Im Arbeitsbereich $U_{GS} > U_{th}$ bildet sich ein leitender n-Kanal. Durch diesen fließen die Elektronen aus der n-Zone als Drainstrom aufgrund der Spannung zwischen Drain und Source.

Bild: Kennlinienfeld eines n-Kanal-MOSFET



Solange $U_{DS} < U_{GS} - U_{th}$ ist, steigt der Drainstrom I_D etwa proportional zur Drainspannung U_{DS}. Dies ist der **lineare Bereich der Kennlinie (oder Widerstandsbereich)**. Wird aber $U_{DS} > U_{GS} - U_{th}$, so wird die Raumladungszone am Drain-Substrat-Übergang größer (weil er eine Diode in Sperrrichtung darstellt) und der leitende Kanal wird „abgeschnürt“. Der Drainstrom I_D geht in den so genannten **Sättigungsbereich** (und steigt nicht nennenswert weiter, auch wenn U_{DS} wächst). Trotz der Abschnürung fließt aber weiterhin ein Strom, da der Kanal bis zu einem bestimmten Abstand von der Drain besteht, und die Elektronen von dort aus durch das elektrische Feld der Drain-Source-Spannung zur Drain hingezogen werden.

Der **p-Kanal-MOSFET** arbeitet analog. Hier wird allerdings das Gate negativ gegenüber Source und Substrat angesteuert. Dadurch wird am Rand der Isolationsschicht ein leitender **p-Kanal** aus Löchern gebildet, über den der Drainstrom fließen kann.

Bild: p-Kanal-MOSFET

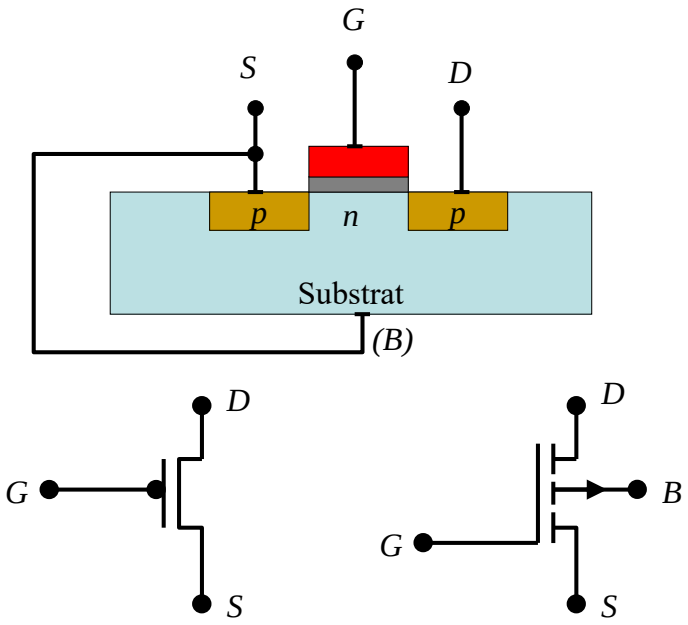


Bild: Leitender Kanal im p-Kanal-MOSFET bei negativer Gate-Substrat-Spannung

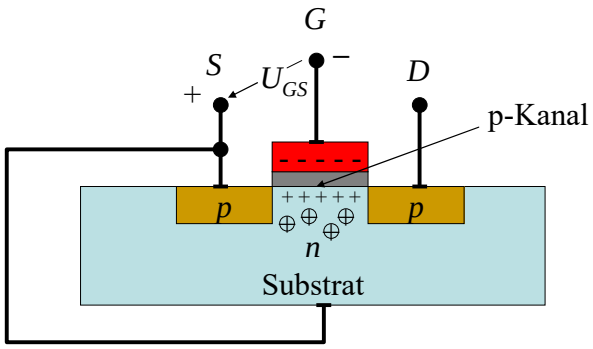
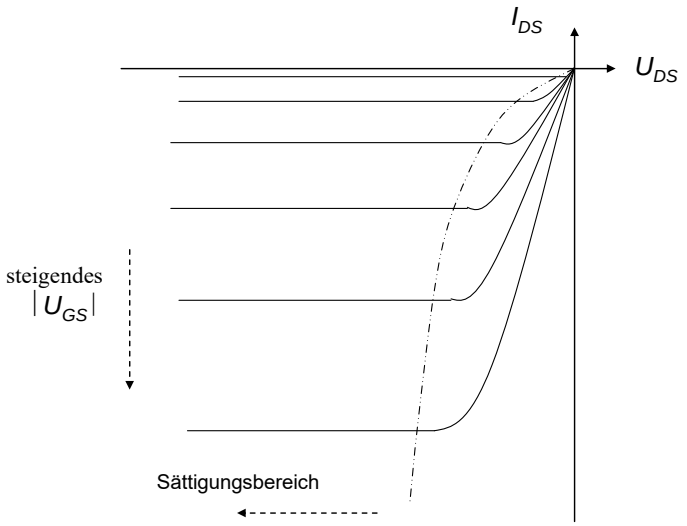


Bild: Kennlinienfeld des p-Kanal-MOSFET



3.3 Aufbau einfacher Gatter aus MOS-Transistoren

MOSFETs können als Schalter benutzt werden. Ein n-Kanal Transistor zum Beispiel verbindet Drain und Source, wenn an seinem Gate eine ausreichend hohe Spannung anliegt ($U_{GS}>U_{th}$). Wenn die Eingangsspannung niedrig ist, sind Drain und Source getrennt ($U_{GS}<U_{th}$). Der p-Kanal Transistor verbindet, wenn seine Eingangsspannung hinreichend klein (negativ) ist im Vergleich zur Source ($U_{GS}<V_{dd}-U_{th}$). und trennt, wenn sie nicht klein genug ist ($U_{GS}>V_{dd}-U_{th}$).

Durch Kombination dieser beiden Typen von Transistoren können wir jetzt logische Schaltungen aufbauen. Da zu jeder Zusammenschaltung von n-Transistoren (die für die logische 0 am Ausgang zuständig ist) immer eine komplementäre Schaltung aus p-Transistoren (für die logische 1 am Ausgang) benutzt wird, nennt man diese Technik **CMOS (complementary MOS) Technik**. Die einfachste solche Schaltung ist ein **Inverter**.

Der Inverter hat einen Eingang und einen Ausgang. Die Spannungen am Ein- und Ausgang identifizieren wir mit logischen Werten, z.B. die volle Versorgungsspannung V_{dd} mit logisch 1 (oder wahr oder TRUE) und das Massepotenzial *GND* (0V) mit logisch 0 (oder falsch oder FALSE). In Spannungen geschrieben sieht diese Tabelle so aus (dabei wird von einer Versorgungsspannung V_{dd} von 2,5 V ausgegangen, die bei heutigen integrierten Schaltkreisen üblich ist:

Bild: Wertetabelle des Inverters, Spannungstabelle des Inverters

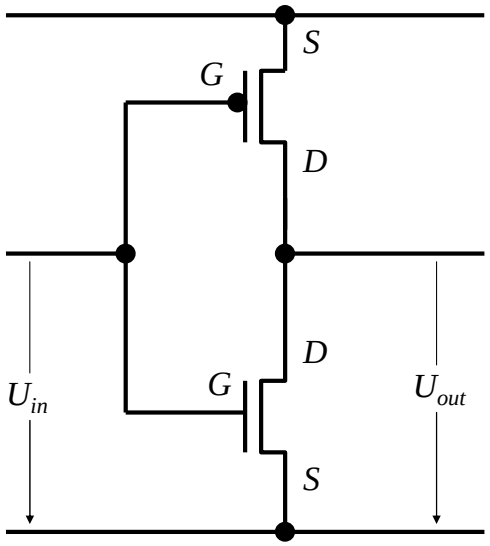
In	out
0	1
1	0

U_{in}	U_{out}
0V	2,5V
2,5V	0V

Wir bauen einen Inverter auf, indem wir einen p-Tansistor und einen n-Transistor in Serie zwischen V_{dd} und *GND* schalten, deren Gate gemeinsam mit dem Eingang beschaltet wird. Da abhängig vom logischen Pegel des Gates immer einer der Transistoren sperrt, kann nie ein Kurzschluss zwischen V_{dd} und *GND* entstehen, bei dem Strom fließt. (Genaugenommen stimmt das nicht ganz: Im Moment des Umschaltens des Eingangs von 0 auf 1 oder umgekehrt von 1

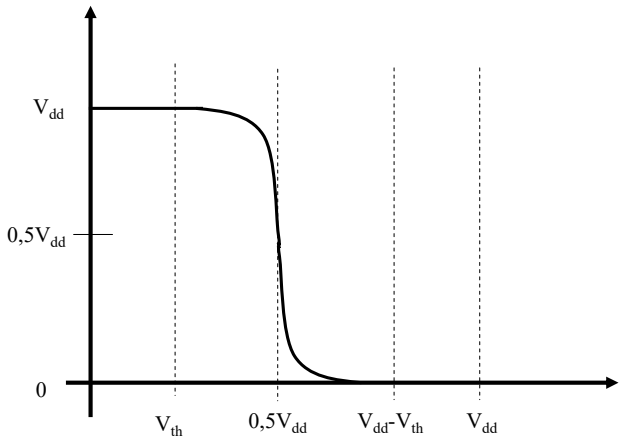
auf 0 werden kurzzeitig beide Transistoren leitfähig. In diesem kurzen Zeitraum fließt ein kleiner Strom von V_{dd} nach *GND*).

Bild: Inverter auf Transistorebene



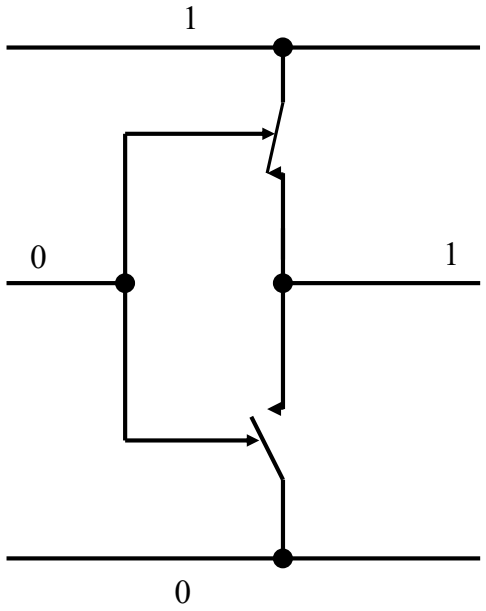
Das folgende Bild zeigt die Ausgangsspannung in Abhängigkeit von der Eingangsspannung am CMOS-Inverter. Wir sehen, dass genau das in der Wertetabelle vorgegebene Verhalten gezeigt wird. Mehr noch: Auch wenn die Eingangsspannung nicht genau auf V_{dd}- oder *GND*-Potenzial liegt, wird der Ausgang ein sauberes Spannungssignal liefern. Ein CMOS-Inverter kann also „schlechte“ (geringfügig verfälschte) Eingangssignale verarbeiten (und durch sein Schaltverhalten „reparieren“).

Bild: Ausgangsspannung in Abhängigkeit von der Eingangsspannung



Eine einfache (für Informatiker, die sich den elektrotechnischen Hintergrund nicht verinnerlichen können) Veranschaulichung der Inverterfunktion liefert das folgende Ersatzschaltbild, in dem die Transistoren als Schalter modelliert werden: Der p-Transistor ist ein Schalter, der bei Eingabe einer 0 geschlossen ist und bei Eingabe einer 1 offen. Der n-Transistor ist bei Eingabe 0 offen und bei Eingabe 1 geschlossen.

Bild: Veranschaulichung der Transsitorfunktion als Schalter



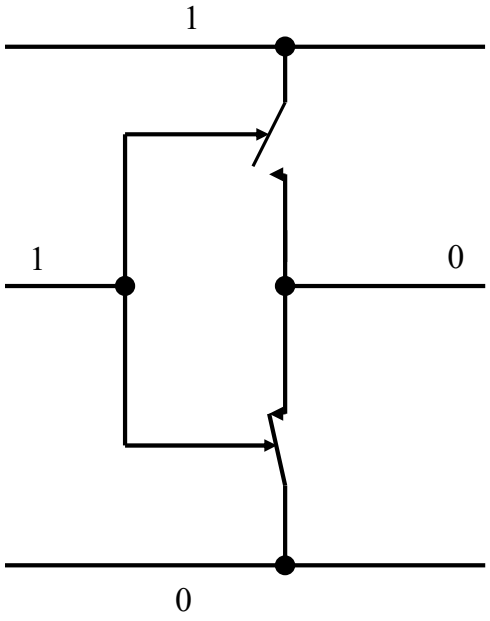
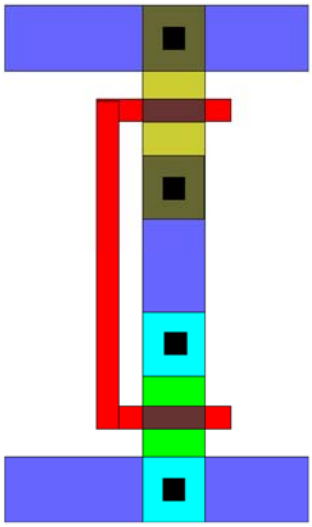


Bild: Ansicht eines Inverters auf dem Chip von oben



Ein Effekt muss hier (auch den Informatikern gegenüber) erwähnt werden. Wenn man n-MOS-Transistoren im Sättigungsbereich betreibt, besteht nur ein Kanal, wenn die Spannung zwischen Gate und Source größer als die Schwellspannung ist ($U_{GS} > U_{th}$). Das bedeutet, wenn die volle Versorgungsspannung an der Drain und am Gate anliegt, und die Source offen ist, kann sich an der Source kein Potenzial einstellen, das höher ist als $U_{GS} - U_{th}$. Das bedeutet, ein n-Transistor ist zwar gut geeignet, um das *GND*-Potenzial weiterzuleiten, bei der Weiterleitung der vollen Versorgungsspannung aber wird diese um eine Schwellspannung vermindert.

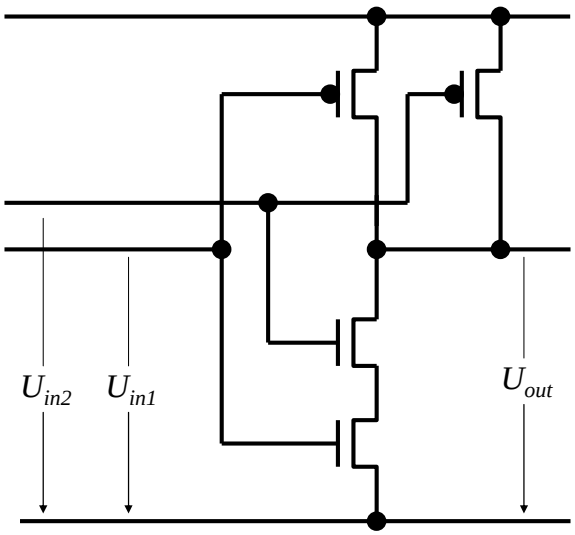
In der Begrifflichkeit der logischen Werte bedeutet das, eine 0 kann von einem n-Transistor gut weitergegeben werden, eine 1 aber nicht. Am Ausgang würde eine „schlechte“ 1 entstehen, also eine Spannung, die um eine Schwellspannung niedriger ist als die Eingangsspannung.

Bei Eingabe einer 0 öffnet der p-Transistor und wird im Widerstandsbereich betrieben, d.h. am Ausgang ensteht eine gute 1. Der n-Transistor sperrt, da keine positive Gate-Spannung gegenüber dem Substrat vorliegt.

Die nächste Funktion, die wir in einem Gatter mit MOS-Transistoren realisieren wollen, ist ein Nand-Gatter. Der Ausgang eines Nand-Gatters ist 1, wenn nicht beide Eingänge auf 1 sind. Wir müssen also erreichen, dass auf den Ausgang das *GND*-Potenzial gelegt wird, wenn beide Eingänge auf 1 sind (*). Ferner müssen wir das Versorgungspotenzial V_{dd} an den Ausgang bringen, falls mindestens einer der Eingänge 0 ist (**).

Wie gelingt dies? Wir schalten zwei n-Transistoren in Serie und verbinden die Source des ersten mit *GND*. Damit erfüllt die Drain des zweiten die erste Bedingung (*) für den Ausgang. Ferner schalten wir zwei p-Transistoren parallel, deren Sourcen wir an V_{dd} anschließen. Wiederum ist die Drain der Ausgang. Damit ist die zweite Bedingung (**) für den Ausgang erfüllt. Verbinden wir nun die Drainanschlüsse dieser beiden Pfade, so ist das Nand-Gatter fertig.

Bild: Nand-Gatter



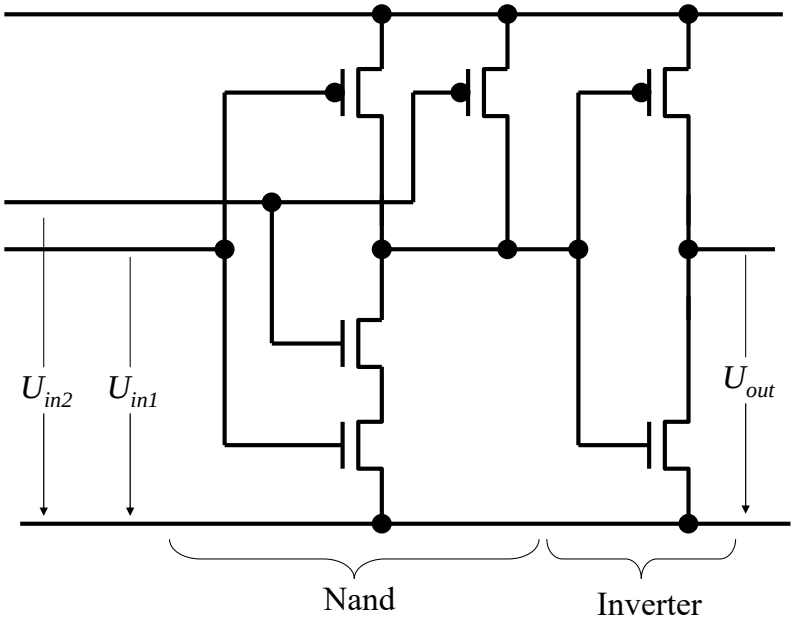
in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0

Man beachte, dass die n-Transistoren wieder lediglich gebraucht werden, um das Potenzial der logischen 0 (*GND*) an den Ausgang zu bringen. Ebenso benutzen wir die p-Transistoren nur zur Weiterleitung des logischen 1-Potentials (V_{dd}). Somit sind die Signale am Ausgang nicht um eine Schwellspannung unterschiedlich zu den beabsichtigten Potenzialen für die entsprechenden logischen Werte. Es ist ein „gute“ 0 und eine „gute“ 1, die am Ausgang zu beobachten ist.

Wie können wir nun ein Und-Gatter aufbauen?

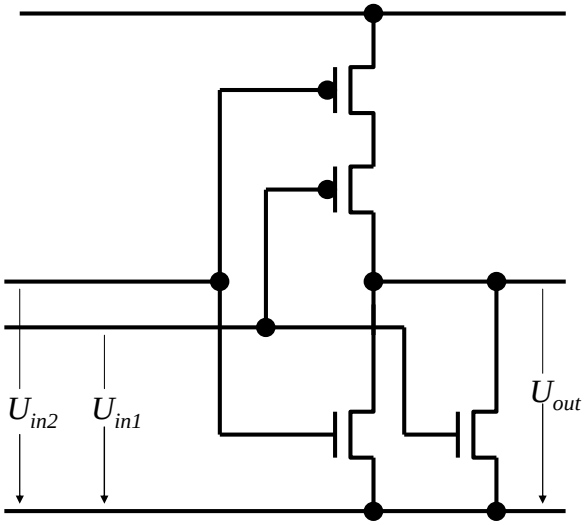
Durch Hintereinanderschalten eines Nand-Gatters mit einem Inverter. Diese Schaltung ist im nächsten Bild zu sehen.

Bild: Und-Gatter



Auch ein Nor-Gatter kann man mit zwei n-Transistoren und zwei p-Transistoren aufbauen. Hier müssen allerdings die p-Transistoren in Serie geschaltet werden und die n-Transistoren parallel.

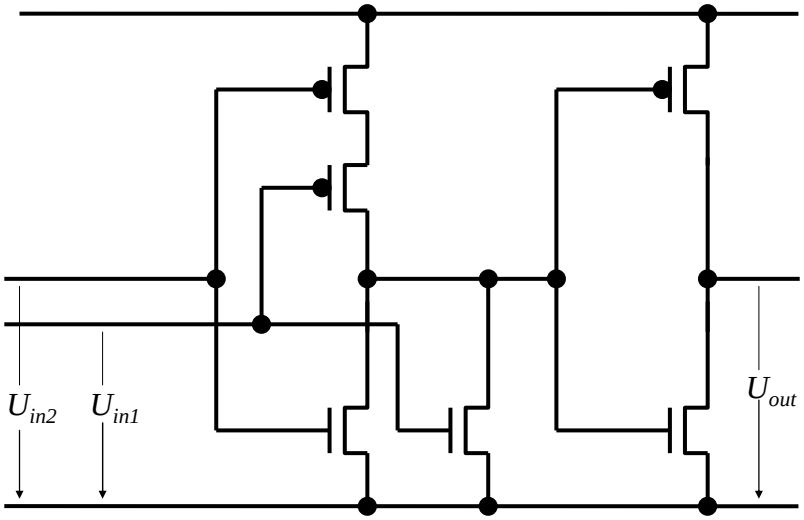
Bild: Nor-Gatter



in1	in2	out
0	0	1
0	1	0
1	0	0
1	1	0

Wiederum können wir ein Oder-Gatter aus einem Nor-Gatter mit einem nachgeschalteten Inverter bauen.

Bild: Oder-Gatter



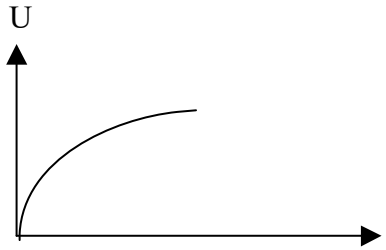
Für die technisch Interessierten Hörer oder Leser sei erwähnt, dass ein Nand-Gatter etwas schneller schaltet als ein Nor-Gatter. Das liegt an der höheren Mobilität der Elektronen im n-Kanal im Vergleich zu den Löchern im p-Kanal. Da beim Nand-Gatter zwei (schnelle) n-Transistoren in Reihe (also hintereinander) zu durchlaufen sind und beim Nor-Gatter zwei (langsame) p-Transisoren ist das Nand-Gatter schneller. Da es im Gegensatz zu Und- und Oder-Gattern einstufing statt zweistufig aufgebaut werden kann, ist Nand in vielen Technologien die gebräuchlichste Gatterform. Aus diesem Grunde kann man sich fragen, ob man die disjunktiven und konjunktiven Normalformen nicht auch mit Nand- und Nor-Gattern aufbauen kann.

3.4 Fan-out und Fan-in

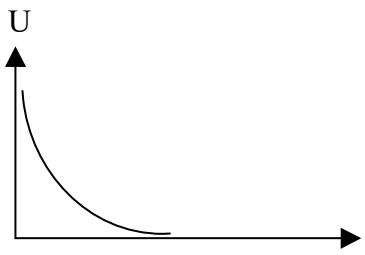
Bisher haben wir die Schaltelemente, mit denen wir gearbeitet haben als „ideale“ Bauteile benutzt, d.h. wir haben vorausgesetzt, dass Änderungen von Werten auf Signalleitungen in der Zeit 0 vollzogen werden. Ein Signal springt von 0 auf 1 und umgekehrt. Dies ist natürlich in der Realität nicht der Fall, sondern der Übergang von logisch 0 auf logisch 1 vollzieht sich als kontinuierlicher Prozess, z. B. indem eine Spannung von 0V auf 3,3V ansteigt. Im Verlaufe eines solchen Übergangs gibt es fünf Phasen:

1. Das Signal ist (nach Beendigung eines vorherigen Schaltvorgangs) im Zustand 0, d.h. die Spannung, die das Signal repräsentiert ist annähernd 0V.
2. Die Spannung beginnt anzusteigen, sie ist größer als 0V, aber sie ist deutlich näher an 0V als an 3,3V, so dass das Signal noch eindeutig als logisch 0 zu erkennen ist.
3. Das Signal ist deutlich größer als 0V aber noch deutlich kleiner als 3,3V, d.h. es kann kein logischer Wert zugeordnet werden.
4. Die Spannung hat einen Wert erreicht, der so dicht an 3,3V liegt, dass das Signal eindeutig als logisch 1 erkannt werden kann.
5. Die Spannung ist annähernd gleich 3,3V, das Signal ist logisch 1, der Schaltvorgang ist abgeschlossen.

Jede dieser Phasen ist für ein positives Zeitintervall aktiv, insbesondere die dritte, in der das Signal weder 0 noch 1 ist. Um die Schaltung insgesamt so schnell (und stabil) wie möglich zu machen, ist es natürlich das Bestreben des Schaltungsentwerfers, die Phase 3 so kurz wie möglich zu machen. Die Strategie dafür hängt einerseits von den Schwellwerten ab, bis zu denen ein Signal eindeutig als 0 bzw. ab denen es eindeutig als 1 angesehen wird, andererseits von der Technologie, die der Realisierung zugrunde liegt. Als Schwellenwerte sind 10% und 90% des Spannungspegels für die 1 (hier 3,3V) sinnvolle Richtwerte. Bei der in Abschnitt 2.6.6 vorgestellten CMOS -Technologie (sowie bei den meisten anderen Schaltkreisfamilien) kann der Schaltvorgang als Umladen einer Kapazität über einen Widerstand modelliert werden. Diese Anordnung stellt den aus der Vorlesung Technische Informatik II bekannten Tiefpass dar, dessen Spannungskurven in der Abbildung skizziert sind.



Wechsel von 0 auf 1



Wechsel von 1 auf 0

Die Zeiten des Wechsels von 0 auf 1 und von 1 auf 0 lassen sich wie folgt berechnen.

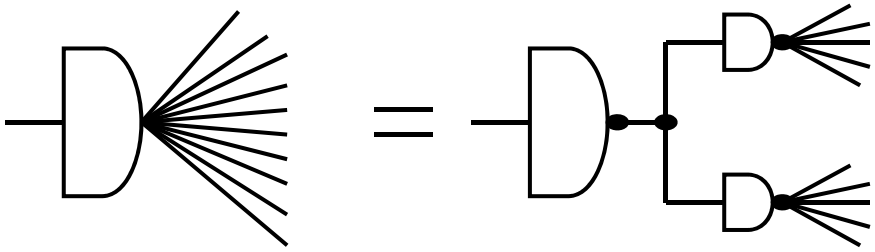
$U_a=U_e e^{-t/RC} \Rightarrow U_a/U_e=e^{-t/RC} \Rightarrow \ln(U_a/U_e)=-t/RC \Rightarrow t=RC(-\ln(U_a/U_e))$ mit $U_a=0{,}9U_e$ erhält man als Nährung $t=0{,}105RC\approx 0{,}1RC$

Man sieht also, dass sowohl der Widerstand als auch die Kapazität als linearer Faktor in die Schaltzeit eingehen. Der Widerstand ist durch den Widerstand des Transistors in den

schaltenden Bauteil und die Kapazität durch die Eingangskapazität des beschalteten Bauteils gegeben.

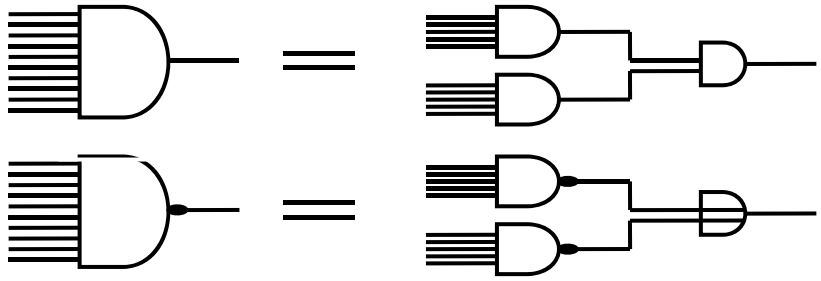
Wenn nun ein Ausgang mit n Eingänge verbunden wird, so muss der Ausgang die n-fache Kapazität eines einzelnen Eingangs auf- bzw. entladen. Die Schaltzeit vergrößert sich also entsprechend. Der Begriff Fan-out (Ausfächerung) skizziert diese Problematik. Der Fan-out eines Gatters ist die Anzahl der Eingänge (der Größe des Gattereingangs selbst), die vom Ausgang des Gatters beschaltet werden. Wenn der Fan-out nun so groß ist, dass die Schaltzeiten zu langsam werden (bei CMOS etwa bei einem Fan-out von 5-10), behilft man sich, indem man zwischen den Ausgang und die Eingänge ein Schaltnetz schaltet, das die Identität realisiert, bei dem aber der Fan-out an jeder Stelle geringer ist.

Beispiel: Fan-Out

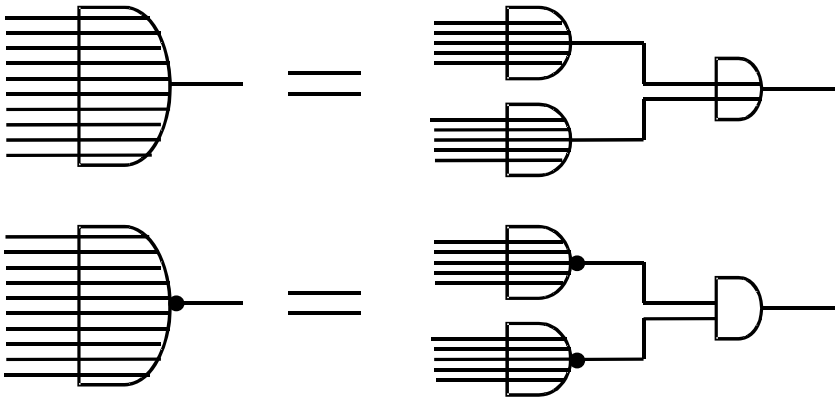


Ein ähnliches Problem tritt auf, wenn die Anzahl der Eingänge in ein Gatter zu groß wird (Fan-in). Hier wird der Widerstand des Gatters zu groß, denn die Widerstände der in Reihe befindlichen Transistoren addieren sich zum Gatterwiderstand (typisch für CMOS 1-10KΩ pro Transistor). Mit der Anzahl der Eingänge sinkt also die Schaltgeschwindigkeit. Auch hier behilft man sich mit der Verwendung von Ersatzschaltungen für Gatter mit zu großem Fan-in.

Beispiel 1: And, Nand



Beispiel 2: Or, Nor



4 Standard-Schaltnetze

Bisher haben wir gelernt, wie man von einer funktionalen Beschreibung einer Booleschen Funktion über eine Wertetabelle zu einer Realisierung als Schaltnetz kommen kann. Im nun folgenden Abschnitt wollen wir einzelne Beispiele von Schaltnetzen studieren, die häufig verwendete Grundbausteine für den Aufbau komplexer Schaltnetze darstellen.

4.1.1 Kodierer

Kodierer wandeln einen „1 aus N Code“ in einen kompakteren Code um, meistens in den Binärcode. Wenn die i-te Eingangsleitung auf 1 ist, wird die Zahl i in binärer Verschlüsselung an die Ausgänge geführt.

Eine Voraussetzung für die einwandfreie Funktion eines Kodierers ist die Tatsache, dass jederzeit genau eine der Eingangsleitungen auf 1 ist und alle anderen auf 0. Wenn diese Bedingung verletzt wird, sind die Werte an den Ausgängen undefiniert.

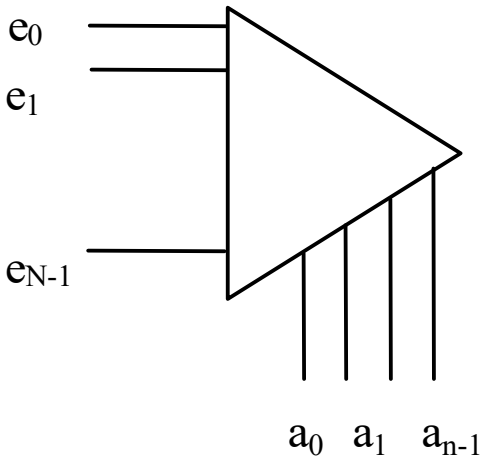
Bei N Eingangsleitungen braucht man [ld(N)] Ausgänge. ld steht für Logarithmus dualis, also Logarithmus zur Basis 2. Die eckigen Klammern bedeuten: die nächstgrößere ganze Zahl. Wenn $N = 2^n$ ist, so ist [ld(N)] = ld(N) = n. Ist N keine Zweierpotenz, so ist die Anzahl der Ausgänge der Logarithmus der nächstgrößeren Zweierpotenz.

Wir betrachten den 1-aus-16 zu 8421-Kodierer (4 Bit Binärdarstellung)

Tabelle Kodierer:

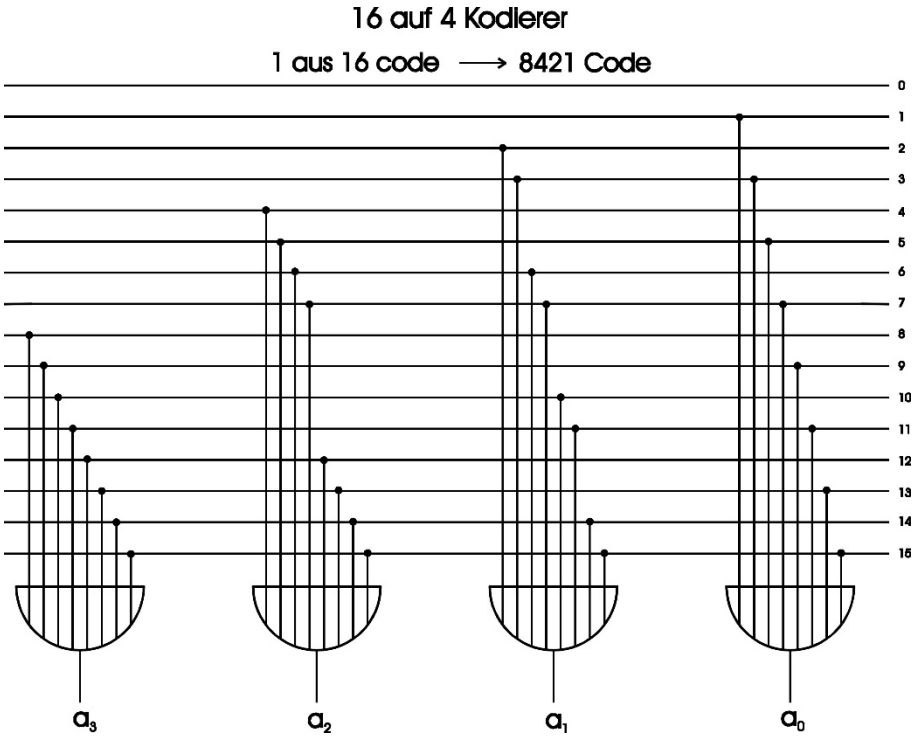
e_{15}	e_{14}	e_{13}	e_{12}	e_{11}	e_{10}	e_9	e_8	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	a_3	a_2	a_1	a_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Bild Kodierer:



Realisierung:

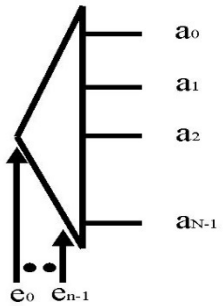
Natürlich werden die großen Oder-Gatter in der dargestellten Realisierung nicht tatsächlich in dieser einstufigen Weise realisiert (vgl. Fan-in-Problematik). Für ein 16-fach Oder bietet sich beispielsweise die Realisierung aus And- und Nor-Gattern an.



4.1.2 Dekodierer

Ein Dekodierer ist das Gegenstück zum Kodierer: Die Eingabevariablen liefern einen codierten Wert, und die Ausgänge wandeln diesen in einen „1 aus N Code“ um. Er „entschlüsselt“ den Eingangscode und weist jeder Eingabekombination genau ein Aktivsignal auf einer einzigen Ausgabeleitung zu. Bei n Eingabeleitungen sind hier $N = 2^n$ Ausgangsleitungen erforderlich.

Bild Dekodierer:

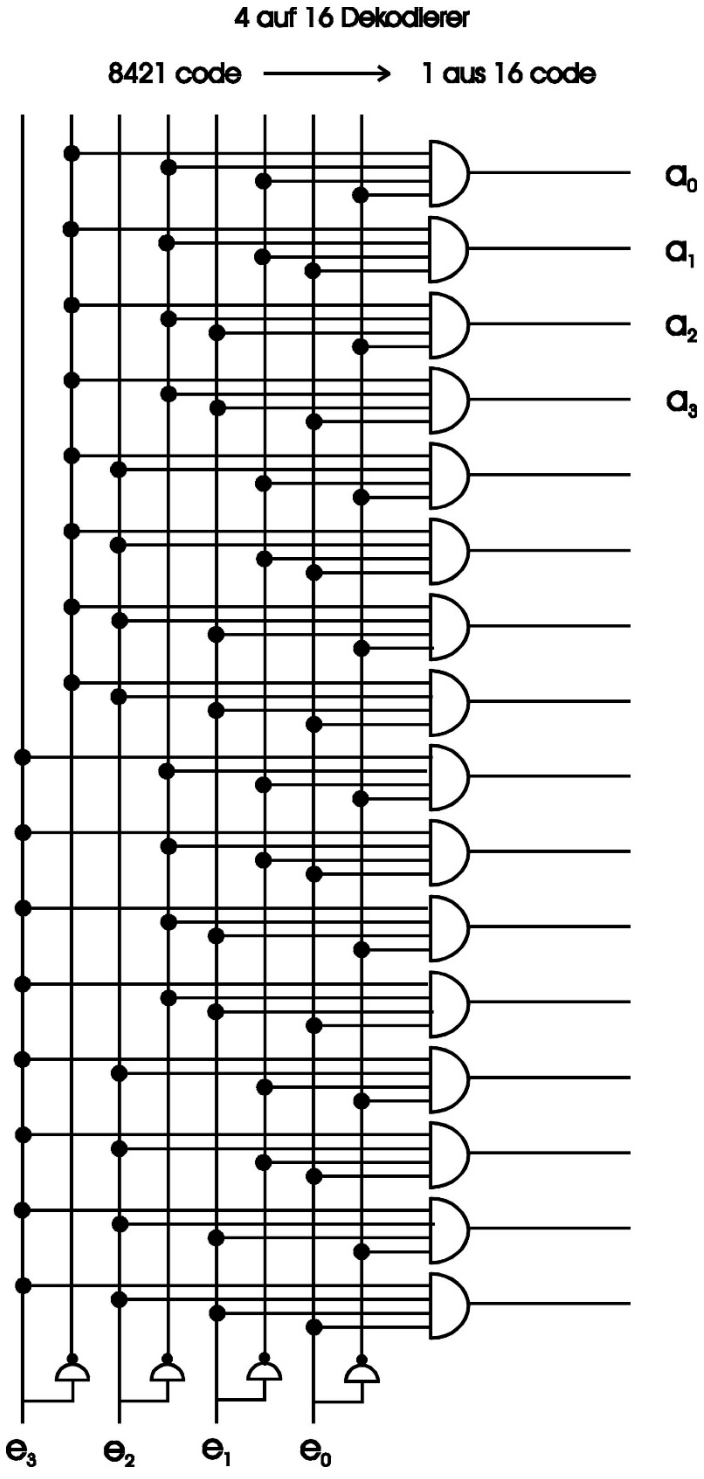


Als Beispiel betrachten wir 8421 zu 1-aus-16 Decodierer

Tabelle:

e_3	e_2	e_1	e_0	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Realisierung:



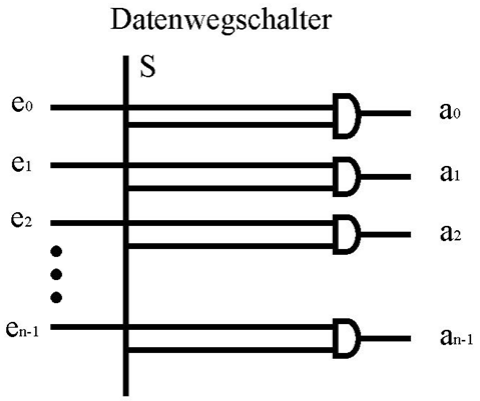
4.1.3 Datenwegschalter

Ein Datenwegschalter schaltet die Eingänge e_i abhängig von einem Steuereingang s auf die Ausgänge a_i. Wenn s = 1 ist, werden die Eingänge unverändert durchgeleitet, wenn s = 0 ist, werden sie abgekoppelt, die Ausgänge werden auf 0 gelegt.

Funktionsbeschreibung: Wir haben n Eingänge (e_0, e1, e_2, ... , e_{n-1}), n Ausgänge (a_0, a1, a_2, ... , a_{n-1}) und „Schalter“ S. Die folgenden Funktionen beschreiben die Funktionsweise eines Datenwegschalters:

$a_0 = e_0S$; $a_1 = e_1S$; $a_2 = e_2S$; ... ; $a_{n-1} = e_{n-1}S$.

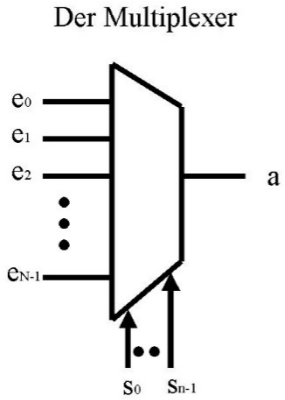
Realisierung:



4.1.4 Multiplexer

Ein Multiplexer (kurz Mux) wählt, abhängig von den Steuereingängen s_i, einen der Eingänge e_i aus und schaltet diesen auf den Ausgang a. Mit n Steuereingängen können so $N = 2^n$ Signaleingänge e_0 , e_1 ,..., e_{N-1} kontrolliert werden.

Funktionsbild Multiplexer:



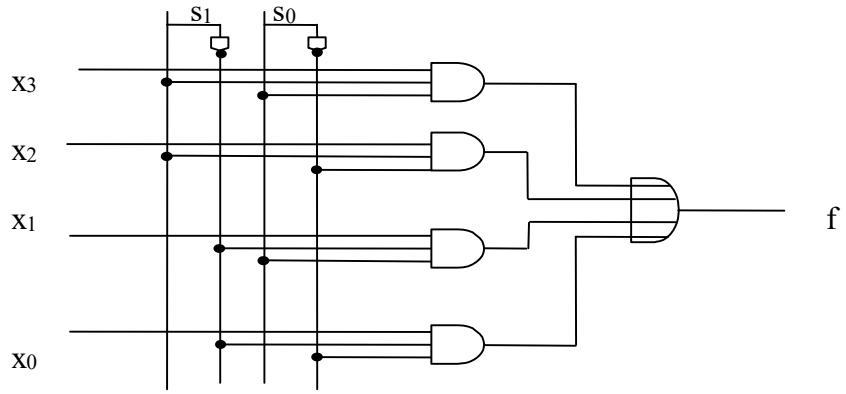
Als Beispiel sehen wir uns ein 16-auf-1 Multiplexer an.

Wertetabelle:

s_3	s_2	s_1	s_0	a
0	0	0	0	e_0
0	0	0	1	e_1
0	0	1	0	e_2
0	0	1	1	e_3
0	1	0	0	e_4
0	1	0	1	e_5
0	1	1	0	e_6
0	1	1	1	e_7
1	0	0	0	e_8
1	0	0	1	e_9
1	0	1	0	e_{10}
1	0	1	1	e_{11}
1	1	0	0	e_{12}
1	1	0	1	e_{13}
1	1	1	0	e_{14}
1	1	1	1	e_{15}

$a = e_i$, wenn $(i)_{10} = (s_3 s_2 s_1 s_0)_2$

Beispiel: Realisierung eines 4-1-Multiplexers in disjunktiver Minimalform



4.1.5 Demultiplexer

Ein Demultiplexer (kurz Demux) bildet das Gegenstück zum Multiplexer. Ein Eingang e wird auf einen der möglichen Ausgänge a_i gelegt in Abhängigkeiten von den Steuereingängen s_i. Mit

n Steuereingängen kann das Eingangssignal an einen von $N=2^n$ Ausgängen a_0 , a_1 , ... , a_{N-1} verbunden werden.

Funktionsbild Demultiplexer:

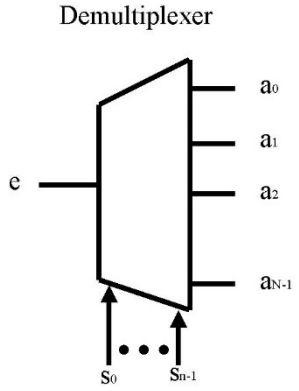
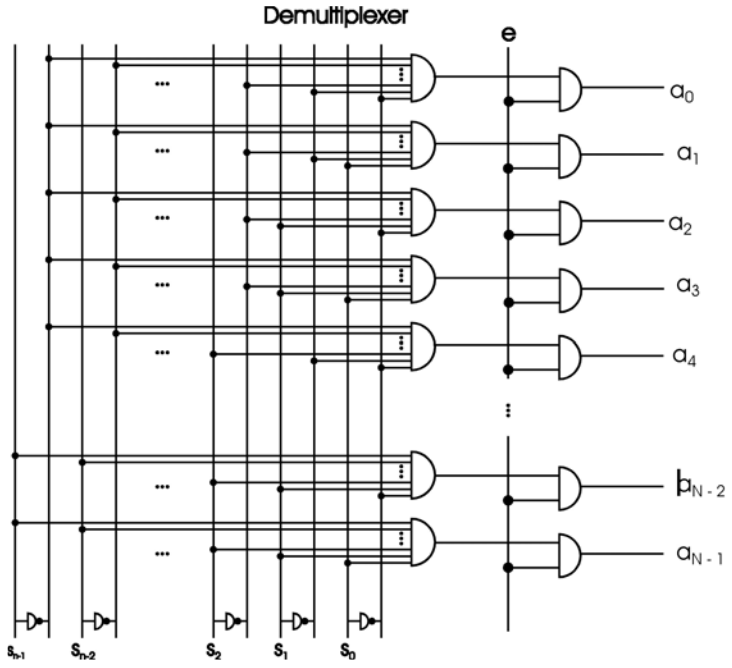


Tabelle:

1 auf 16 Demultiplexer

e	s_3	s_2	s_1	s_0	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
0	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

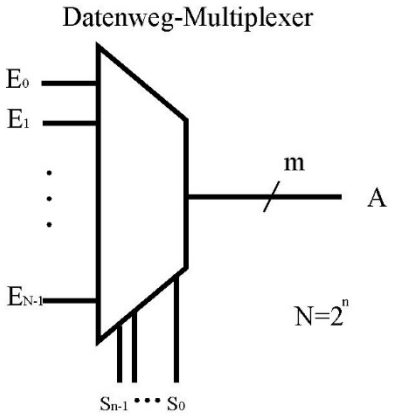
Realisierung:



4.1.6 Datenweg-Multiplexer

Ein Datenweg-Multiplexer ist ein Multiplexer, der gleichzeitig eine Menge von m Eingangssignalen auf m Ausgangsleitungen verschaltet. Aus einer Menge von m*N Eingängen wird also gemäß den Werten auf den Steuereingängen s_i, eine Teilmenge von m Eingängen ausgewählt, die auf die m Ausgänge geschaltet werden.

Bild:



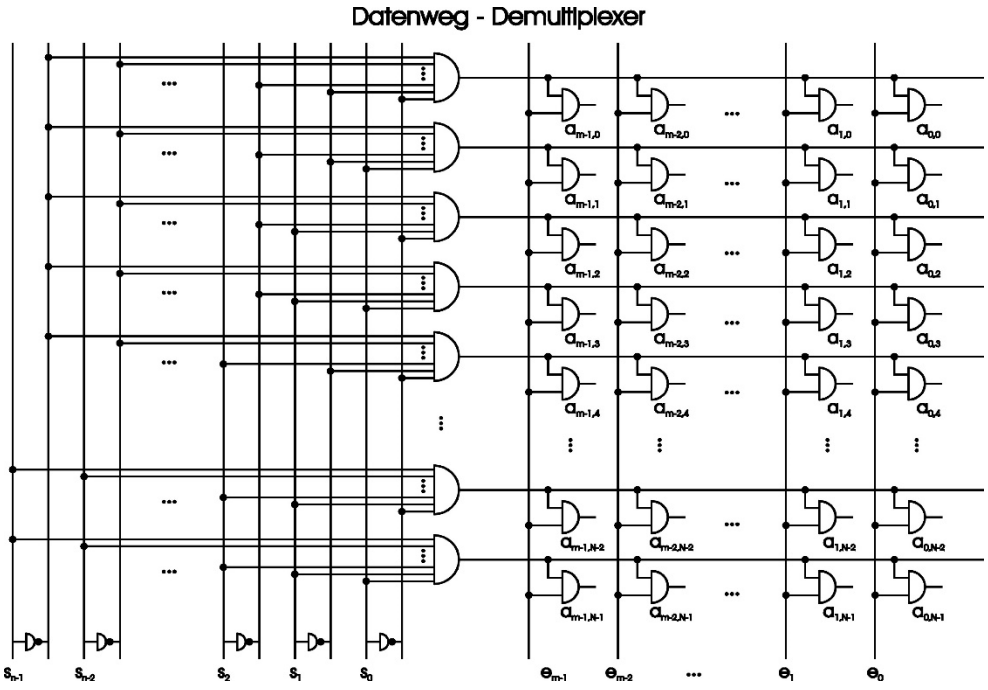
4.1.7 Datenweg-Demultiplexer

Ein Datenweg-Demultiplexer ist ein Demultiplexer, der gleichzeitig eine Menge von m Eingangssignalen auf m Ausgangsleitungen verschaltet. Aus einer Menge von m*N Eingängen wird also gemäß den Werten auf den Steuereingängen s_i, eine Teilmenge von m Eingängen ausgewählt, die auf die m Ausgänge geschaltet werden.

Tabelle:

s_{n-1}	s_{n-1}	...	s_2	s_1	s_0	A
0	0	...	0	0	0	A_0=E, A_i=0 für i≠0
0	0	...	0	0	1	A_1=E, A_i=0 für i≠1
0	0	...	0	1	0	A_2=E, A_i=0 für i≠2
0	0	...	0	1	1	A_3=E, A_i=0 für i≠3
0	0	...	1	0	0	A_4=E, A_i=0 für i≠4
		•				•
		•				•
		•				•
1	1	...	1	1	0	A_{N-2}=E, A_i=0 für i≠N-2
1	1	...	1	1	1	A_{N-1}=E, A_i=0 für i≠N-1

Realisierung:

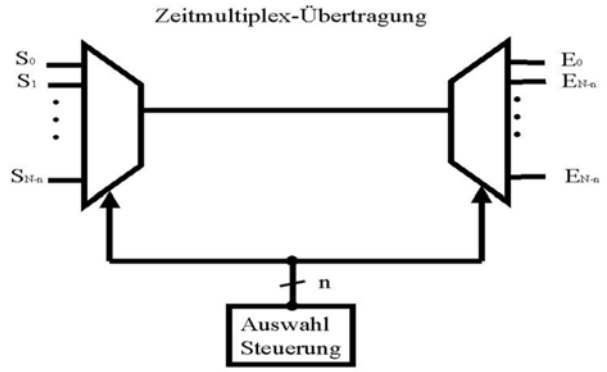


4.1.8 Zeitmultiplex-Übertragung

Wenn eine Menge von Paaren (Sender/Empfänger) Daten austauschen wollen, für diese aber nur ein Kanal zur Verfügung steht, kann man unter Benutzung eines Multiplexers und eines Demultiplexers den Kanal über die Zeit an die Kommunikationspartner aufteilen. Das Bild unten stellt die Anordnung einer Zeitmultiplex-Übertragung dar. Jedes Paar E_i, S_i, bekommt den Kanal für ein Zeitintervall T, und alle Paare werden nacheinander bedient. Dies wird erreicht durch eine Zentrale Steuerung, die den Multiplexer und den Demultiplexer mit jeweils

denselben Steuersignalen ansprechen. Die Steuerung verfügt über einen internen Zähler, der zyklisch von 0 bis N-1 zählt. Der Wert des Zählers stellt das Steuersignal S an den Multiplexer und den Demultiplexer dar.

Bild:



Zeitachse:

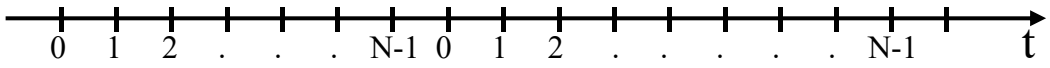


Tabelle:

T	S_{n-1}	S_{n-2}		S_2	S_1	S_0	Verbindung
0-Δt	0	0	...	0	0	0	$E_0 \Rightarrow A_0$
Δt-2Δt	0	0	...	0	0	1	$E_1 \Rightarrow A_1$
2Δt-3Δt	0	0	...	0	1	0	$E_2 \Rightarrow A_2$
3Δt-4Δt	0	0	...	1	0	0	$E_3 \Rightarrow A_3$
. . .			...				. . .
(N-2) Δt - (N-1) Δt	1	1	...	1	1	1	$E_{N-1} \Rightarrow A_{N-1}$

4.1.9 Datenbus

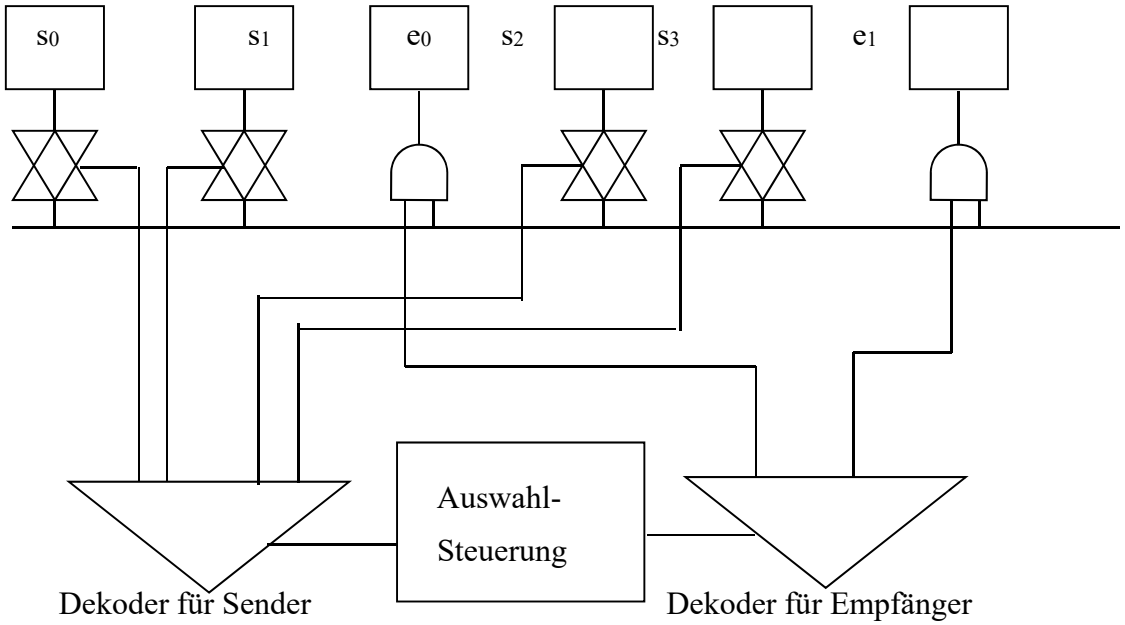
Häufig sind die Paare (Sender/Empfänger) nicht physikalisch benachbart in der Hardware eines Chips, eines Rechners oder eines Rechnersystems angeordnet. In diesem Falle müssen Multiplexer und Demultiplexer an die einzelnen Komponenten verteilt werden. Dies führt zum Konzept des Datenbus.

Ein Bus (wie Omnibus, lat.: für alle) ist ein gemeinsamer Kanal, auf den mehrere Komponenten schreibend und lesend zugreifen können. Natürlich muss geregelt werden, wer zu welchem Zeitpunkt schreibenden oder lesenden Zugriff auf den Bus bekommt. Dies wird wieder von einer zentralen Auswahlsteuerung (Bus-Master) geleistet.

Der Multiplexer aus der Zeitmultiplex-Übertragung wird hier zu einem zentralen Dekodierer, dessen Ausgänge als Steuerleitungen an die Sender geführt werden. Ist eine solche Steuerleitung auf logisch 1, so wird der Schreib-Einheit der Zugriff auf den Bus gewährt. Dies kann durch ein Transmissions-Gatter geschehen. Ist die Steuerleitung auf 0, so sperrt das Transmissionsgatter, d.h. der Ausgang ist hochohmig und es wird von dieser Einheit nicht auf den Bus geschrieben.

Der Demultiplexer aus der Zeitmultiplex-Übertragung wird hier zu einem zentralen Dekodierer, dessen Ausgänge als Steuerleitungen an die Empfänger geführt werden. Ist eine solche Steuerleitung auf logisch 1, so wird über einen Datenwegschalter der lesende Zugriff auf den Bus gewährt. Ist die Steuerleitung auf 0, so sperrt der Datenwegschalter die Übertragung vom Bus.

Bild:



Bus mit zentraler Auswahlsteuerung

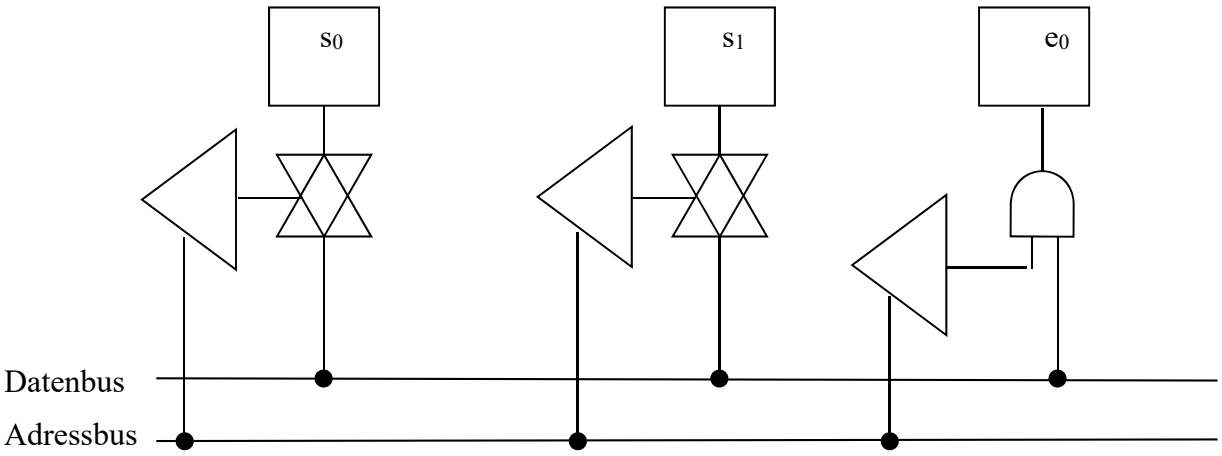
4.1.10 Daten- und Adressbus

Der Nachteil der Anordnung aus dem letzten Abschnitt ist darin zu sehen, dass von den Dekodierern der zentralen Auswahlsteuerung an alle Einheiten direkte Leitungen geführt werden müssen. Dies wird in der Praxis vermieden, indem man dem Datenbus zusätzliche Leitungen zuordnet, auf denen die Adressen der jeweiligen Sender und Empfänger in kodierter Form an alle Einheiten übertragen werden. Durch diese Maßnahme werden auch die Dekodierer an die Einheiten verteilt.

Die zusätzlichen Leitungen werden Adressbus genannt. Adressbusse können für Sender und Empfänger identisch sein. In Bild (siehe unten) sind sie jedoch für das einfachere Verständnis als zwei getrennte Leitungsbündel eingezeichnet. Jeder Sender hat nun lokal einen Teil des Dekodierers (meist realisiert durch eine einfache Konjunktion) zur Verfügung, über die er aus dem Sender-Adressbus ersehen kann, ob er das Schreibrecht auf den Bus besitzt. In diesem Fall schaltet er seinen Ausgang über ein Transmissions-Gatter auf den Bus.

Jeder Empfänger entschlüsselt die Adresse auf dem Empfänger-Adressbus. Findet er seine eigene Adresse vor, so liest er über seinen Datenwegschalter die Nachrichten vom Bus, solange, bis von der zentralen Auswahlsteuerung eine neue Adresse auf den Empfänger-Adressbus gelegt wird.

Bild:



Bus mit verteilter Auswahlsteuerung

4.2 Schaltnetzrealisierungen durch Speicher und PLAs

4.2.1 Schaltnetzrealisierungen durch Speicher (z.B. ROM, PROM, RAM)

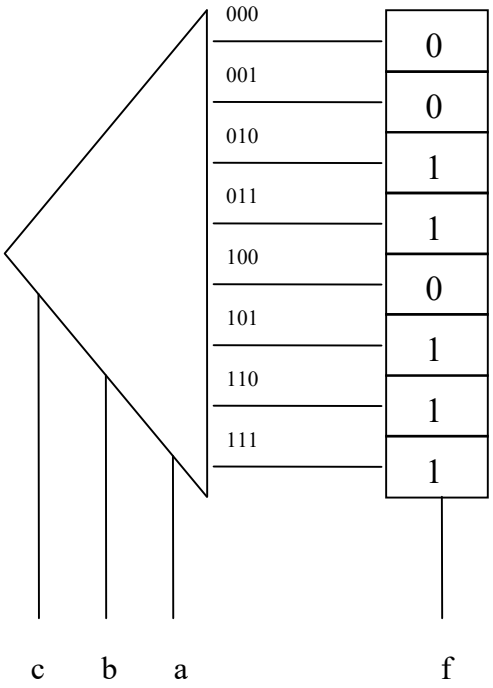
Durch die günstige Verfügbarkeit bestimmter vorgefertigter Logik-Bausteine in unterschiedlichen Technologien können auch andere Strategien als die der Schaltnetzminimierung für die Realisierung einer Schaltung sinnvoll sein. Zum Beispiel kann man eine Schaltung in kanonischer disjunktiver Normalform unter Benutzung eines Speichers realisieren. Ein Speicher ist eine geordnete Menge von Speicherzellen, von denen jede ein Wort fester Breite (z.B. 1 Bit) aufnehmen und ggf. wiedergeben kann. Allen solchen Speichern, die wir später in der Vorlesung noch detaillierter behandeln werden, ist gemeinsam, dass jede einzelne Speicherstelle eine Adresse besitzt. Über einen Dekoder kann man diese Adresse am Speicher anlegen, und danach hat man Zugriff auf die gewählte Speicherstelle.

Wenn wir eine Boolesche Funktion mit einem Speicher (anstatt mit logischen Gattern) realisieren wollen, müssen wir die Wertetabelle in diesen Speicher schreiben. Wenn wir sodann

die Eingänge unserer Booleschen Funktion als Adresse an den Speicher anlegen, erscheint am Ausgang automatisch der Funktionswert.

Beispiel:

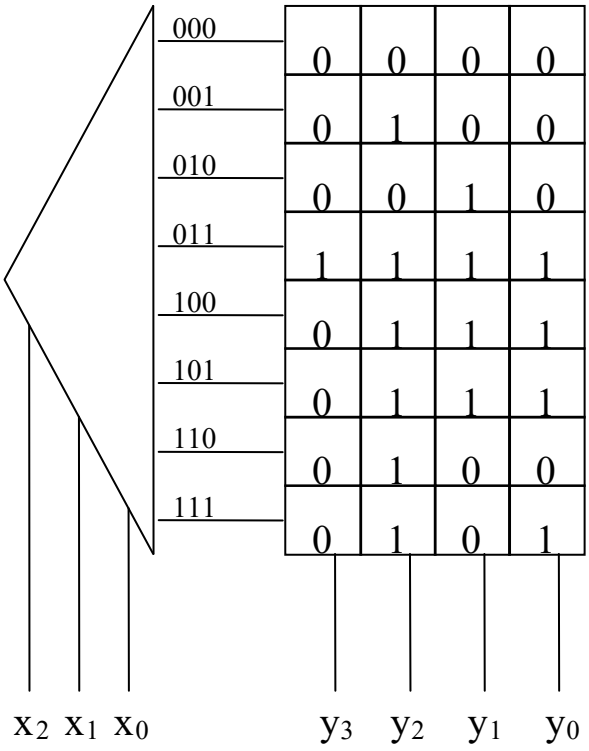
$$f = b + ac$$



Natürlich können auch mehrwertige Funktionen mit einem Speicher anstelle von Gattern realisiert werden. Dann benötigt der Speicher eine Wortbreite, die mit der Anzahl der zu berechnenden Bits der Funktion übereinstimmt.

Beispiel:

$$y_0 = x_0x_1 + x'_1x_2$$
$$y_1 = x'_1x_2 + x_1x'_2$$
$$y_2 = x_0 + x_2$$
$$y_3 = x_0x_1x'_2$$

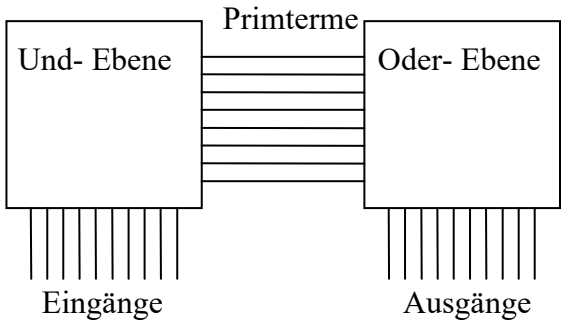


4.2.2 Schaltnetzrealisierungen durch PLAs

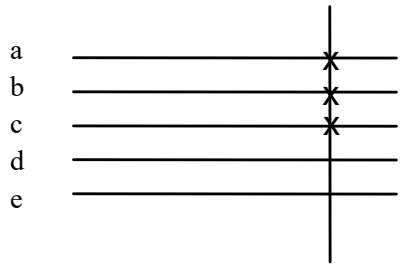
Natürlich ist eine Schaltungsrealisierung in disjunktiver Normalform im allgemeinen unverhältnismäßig aufwendig. Immerhin braucht man für die Realisierung einer Funktion mit k Eingängen 2^k Zeilen in der Oder-Ebene. Man kann aber die Ideen aus dem letzten Abschnitt so modifizieren, dass auch minimierte Schaltungen, z.B. in disjunktiver Minimalform, realisiert werden können. Auf diese Weise kann häufig die Anzahl der Produktterme um Größenordnungen vermindert werden. Dies führt zur Realisierung von Schaltnetzen mit PLAs (Programmable Logic Arrays).

Ein PLA besteht aus einer Und-Ebene und einer Oder-Ebene. In der Und-Ebene werden aus den Eingabewerten beliebige Produktterme (z.B. Minterme) gebildet. Diese werden in die Oder-Ebene geführt, wo sie in disjunktiver Weise verknüpft werden. Die Ausgänge der Oder-Ebene sind die Ausgänge des Schaltnetzes.

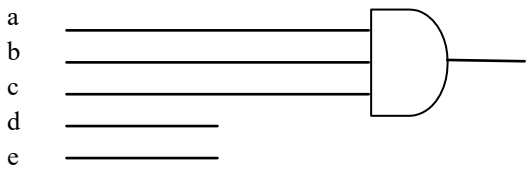
Bild: Und-Ebene, Oder-Ebene, Eingänge, Ausgänge, Primterme



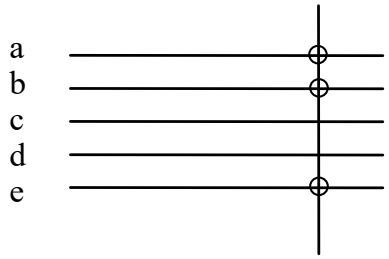
Für die einfache Darstellung benutzt man die sogenannte „verdrahtete Logik“. Bei dieser werden logische Gatterfunktionen durch zeichnen von Kreuzen (UND) und Kreisen (ODER) auf den Eingangsleitungen symbolisiert. Ein UND-Gatter mit drei Eingängen a, b und c könnte z.B. so dargestellt werden:



Die Zeichnung ist gleichbedeutend mit



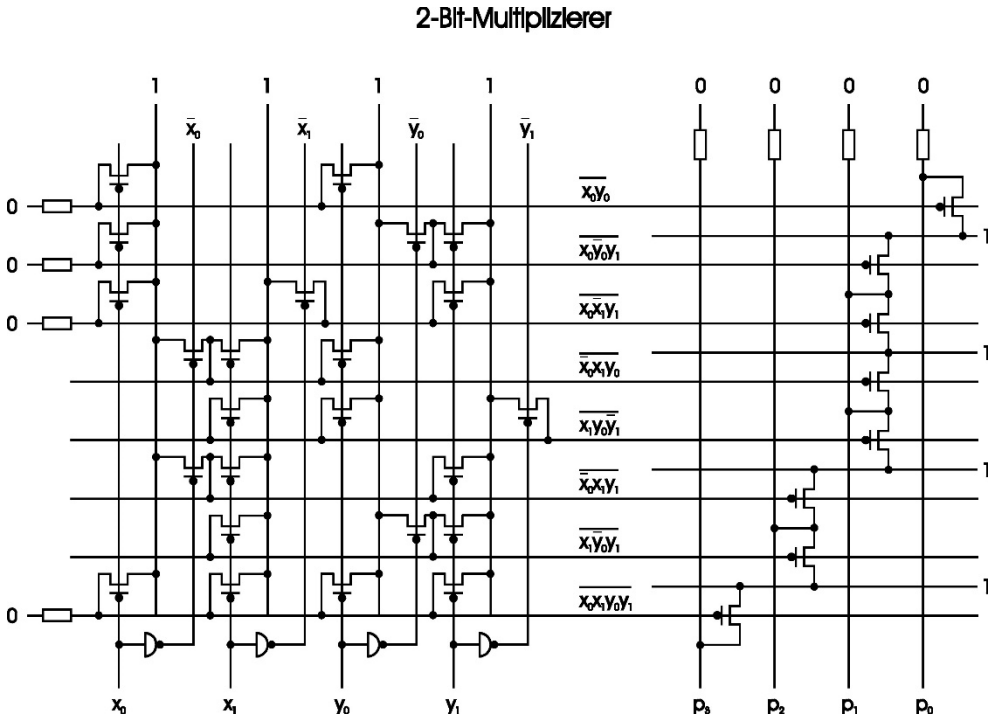
Ein ODER-Gatter mit den Eingängen a, b, e könnte so aussehen:



In der technischen Realisierung wird in der Und-Ebene und in der Oder-Ebene die Nand-Funktion (als wired Nand) gebildet, so daß die Primterme in invertierter Form von der Und-Ebene in die Oder-Ebene überführt werden. Dies Vorgehen basiert auf der in Abschnitt 2.6.7 dargestellten Form der disjunktiven Minimalform in Nand-Logik.

Das folgende Bild zeigt ein PLA mit vier Eingängen und maximal acht Produkttermen als MOS-Realisierung. Dieses PLA ist so gebrannt worden, dass der 2-Bit Multiplizierer aus der Vorlesung in DMF realisiert wird.

Bild:



Beim wired Nand arbeitet man mit pull-down-Widerständen, d. h. eine Ausgangsleitung wird über den Widerstand auf 0 „heruntergezogen“. Wenn mindestens einer der p-Transistoren jedoch eine 1 auf die Leitung schreibt, fällt an diesem Widerstand die gesamte Versorgungsspannung ab, der Ausgang ist auf 1. Die p-Transistoren sind ja leitend, wenn ihr Eingang auf 0 liegt und sie sperren, wenn eine 1 am Eingang liegt.

4.3 Dynamik in Schaltnetzen

Die Diskussion über Fan-out und Fan-in hatte uns bereits mit der Situation konfrontiert, dass Schaltelemente wie Gatter in der Realität ein geringfügig anderes Verhalten an den Tag legen als in der Theorie. Die Flanken auf den Signalleitungen sind nicht ideale Sprünge von 0 auf 1 oder umgekehrt, sondern kontinuierliche (exponential) Kurven. Ein wichtiger Effekt davon ist die Tatsache, dass jeder Schaltvorgang eine Zeit größer als Null benötigt. In einer ersten Näherung kann man unterstellen, dass alle Gatter die gleiche Verzögerungszeit brauchen, genannt eine Schaltzeit, Δt. Die Schaltzeit ist definiert als die Zeitdifferenz zwischen dem Zeitpunkt, an dem die Eingänge des Gatters wechseln und dem Zeitpunkt, an dem das neue Signal am Ausgang eindeutig zu erkennen ist (letzteres ist wichtig, da Signalverläufe in der Praxis so sein können, dass bei einem Signalwechsel mehrfach zwischen einem definierten und einem undefinierten Zustand gewechselt wird. Natürlich hängt das davon ab, wie „definierter Zustand“ erklärt ist. Dies wird uns noch genauer in kommenden Kapiteln beschäftigen).

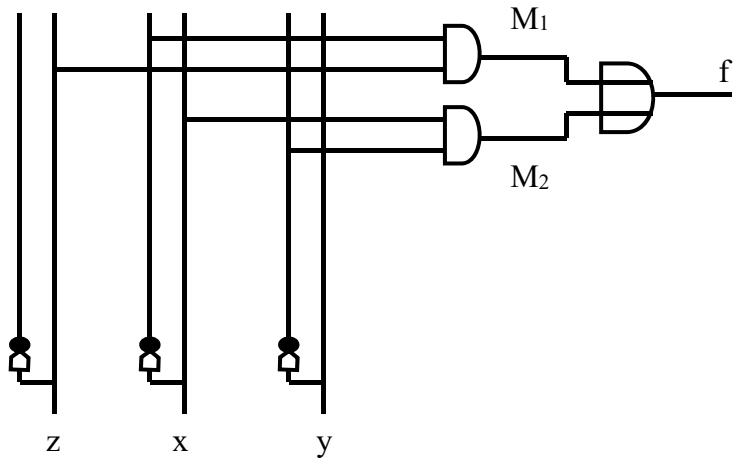
4.3.1 Hazards

Hazards sind kurzzeitige falsche Signale auf Leitungen, die dadurch entstehen, dass Schaltzeiten von Gattern zu Zeitdifferenzen führen zwischen den Zeitpunkten, an denen an Eingängen die Signale wechseln. Dadurch kann ein Ausgang (für kurze Zeit) einen Wert annehmen, der weder der Wert vor dem Schaltvorgang noch der Wert nach dem Schaltvorgang ist.

Beispiel:

$f = z\bar{x} + x\bar{y}$

Realisierung:



Seien z = 1, y = 0 und x schaltet von 1 auf 0. Dabei muss f immer gleich 0 sein. Jetzt betrachten wir wie sich f in Wirklichkeit verhält:

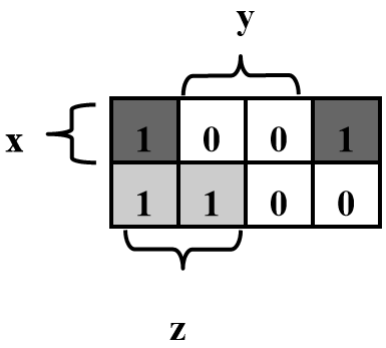
$x=1 \Rightarrow f=M_1+M_2=0+1=1;$

$x=1\rightarrow 0 \Rightarrow$ durch die Verzögerung bei der Invertierung von x bleibt $\bar{x}$ noch ein Moment auf 0 $\Rightarrow M_1=0, M_2=0 \Rightarrow f=M_1+M_2=0+0=0$. Kurz danach ist $M_1=1$ und $f=1+0=1$. Dieses kurzzeitig falsche Signal ist ein Hazard.

4.3.2 Vermeidbare Hazards

Die Hazards können nur dann entstehen, wenn in der betrachteten Funktion (Beispiel siehe oben) in der KV-Diagramm die Blöcke benachbart sind.

Beispiel: KV-Diagramm zur oben angegebenen Funktion:



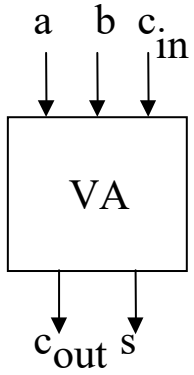
Um solche Hazards zu vermeiden, nimmt man noch einen Block dazu. Dieser Block muss nichtkritische Variable enthalten und die Arbeitsweise der Funktion dabei nicht verändern. In diesem Fall ist das $z\,\overline{y}$. Die neue Funktion lautet $f_{neu} = z\overline{x} + x\overline{y} + z\overline{y}$. Bei dem Übergang von x von 1 auf 0 gewährleistet $z\,\overline{y}$ eine 1 (z=1, y=0), sonst macht $z\,\overline{y}$ gar nichts. Jetzt tritt kein Hazard auf.

5 Computer Arithmetik

In diesem Abschnitt wollen wir einige grundlegende Techniken kennen lernen, mit denen in Computern arithmetische Operationen ausgeführt werden. Das dabei erworben Wissen werden wir später in den Abschnitten über Schaltwerke, ALU-Aufbau und Rechnerarchitektur vertiefen.

5.1 Addition

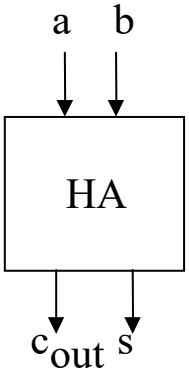
Wir kennen bereits einen Volladdierer. Es ist ein Schaltnetz mit drei Eingängen a, b, c_{in} und zwei Ausgängen s und c_{out}. Der Volladdierer ist in der Lage, drei Bits zu addieren und das Ergebnis als 2-Bit-Zahl auszugeben. Das Ergebnis liegt ja zwischen 0 und 3 und ist daher in zwei Bits zu codieren. Wir sehen hier das Schaltbild eines Volladdierers und seine Wertetabelle:



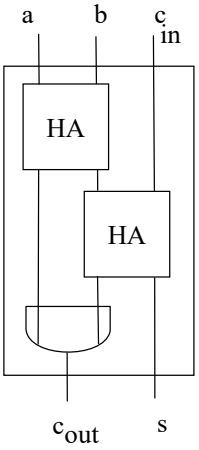
a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Häufig realisiert man einen Volladdierer nicht in DMF sondern in einer mehrstufigen Form, wobei man sogenannte Halbaddierer benutzt. Halbaddierer sind Schaltnetze, die zwei Bits addieren können (und demzufolge ein Ergebnis im Bereich 0 bis 2 produzieren). Durch Zusammenschalten von zwei Halbaddierern und einem Oder-Gatter erhält man die Funktionalität eines Volladdierers. Wie sehen im Folgenden das Schaltsymbol eines Halbaddierers, seine Wertetabelle und den Aufbau eines Volladdierers aus Halbaddierern.

a	B	s	c_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Volladdierer aus zwei Halbaddierern und einem Oder- Gatter



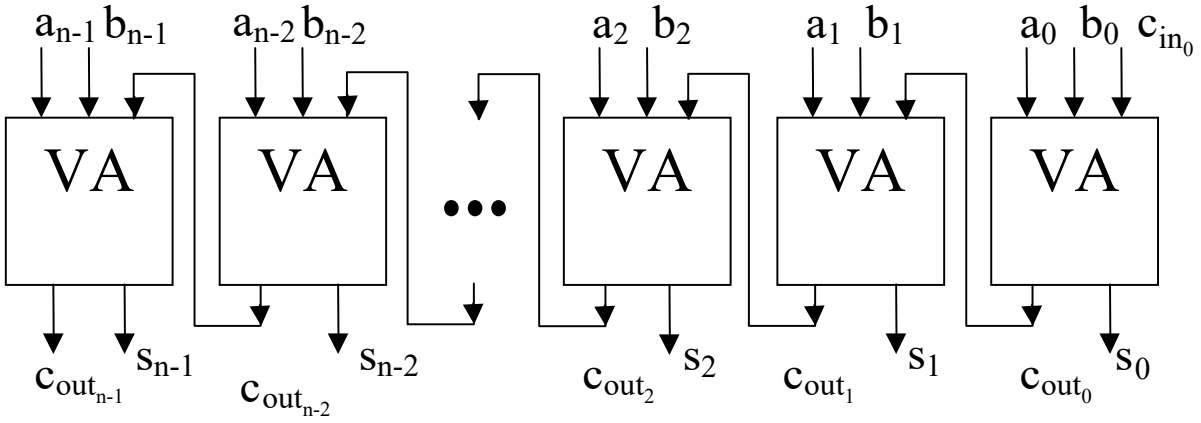
Nun wollen wir aber in der Regel längere Operanden addieren, zum Beispiel die Binärzahlen $a=a_{n-1}a_{n-2}...a_1a_0$ und $b=b_{n-1}b_{n-2}...b_1b_0$. Natürlich könnte man ein dafür erforderliches Addierwerk in KDNF oder DMF aufbauen. Dies bringt aber eine Reihe von Problemen mit sich:

- für jedes n ergibt sich eine völlig andere Realisierung
- das Fan-in und das Fan-out an den Gattern wächst polynomiell mit n

5.1.1 Ripple-Carry-Addierer

Insbesondere wegen dieser zweiten Eigenschaft ist der zweistufige Aufbau z. B. in DMF nicht sinnvoll. Statt dessen verwendet man im einfachsten Fall eine Kette von Volladdierern, die im

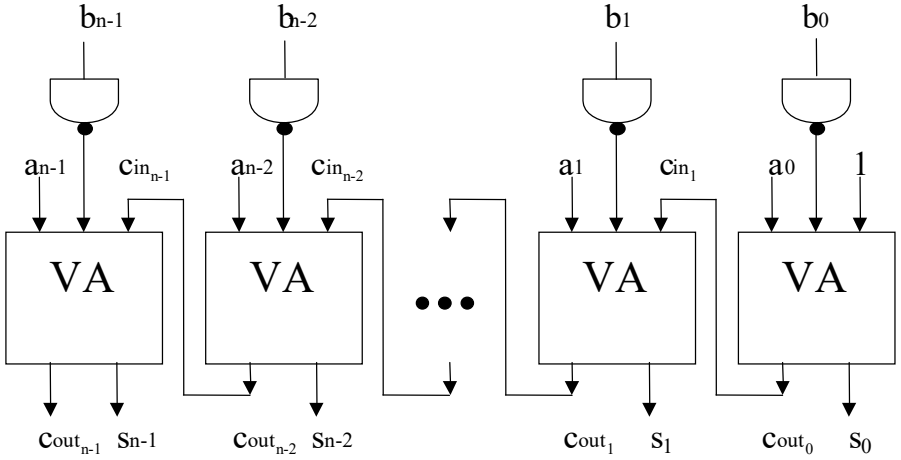
Grunde genau das machen, was wir von der Addition in der „Schulmethode“ kennen. Man beginnt mit den LSBs (least significant bits), addiert diese, erzeugt einen Übertrag, mit dessen Kenntnis man das nächste Bit bearbeiten kann, usw. Ein entsprechendes Schaltnetz sieht dann so aus:



Das Ergebnis der Addition von zwei n-Bit-Zahlen ist eine n+1-Bit Zahl. Diese ist repräsentiert durch die Ausgänge $c_{n-1}S_{n-1}S_{n-2}...S_2S_1S_0$.

Einen solchen Addierer nennt man einen **ripple-carry-adder**. Sein Vorteil ist der einfache und modulare Aufbau. Sein wesentlichster Nachteil wird bereits durch diesen Namen ausgedrückt: Wenn die Operanden gerade eine ungünstige Bit-Kombination aufweisen, muss die Carry- (übertrags-) -Information durch alle Volladdierer hindurch von der Stelle mit der geringsten Wertigkeit bis zur Stelle mit der höchsten Wertigkeit hindurchklappern (rippeln). Damit ergibt sich die Schaltzeit eines ripple-carry-adders als proportional zur Zahl n der Stellen. Dies ist insbesondere dann ein Problem, wenn in unserem Rechner ein Zahlenformat mit vielen Bits (z.B. 64 Bits) verarbeitet werden soll. Sicher wollen wir den Maschinentakt nicht so langsam machen, dass in einem Takt 64 Volladdierer nacheinander schalten können. Wir werden bald sehen, wie man dieses Problem behandeln kann.

Zunächst wollen wir uns aber damit beschäftigen, wie man mit einem Addierer auch subtrahieren kann. Wir wissen bereits: das Zweierkomplement einer Zahl lässt sich berechnen als Einerkomplement plus 1. Ferner ist das Einerkomplement die bitweise Negation der Zahl. Wenn wir nun die Addition der 1 über den Carry-Eingang c_{in_0} erledigen, können wir mit dem Schaltnetz auf dem folgenden Bild a-b berechnen, indem wir zu a das Zweierkomplement von b hinzuaddieren:

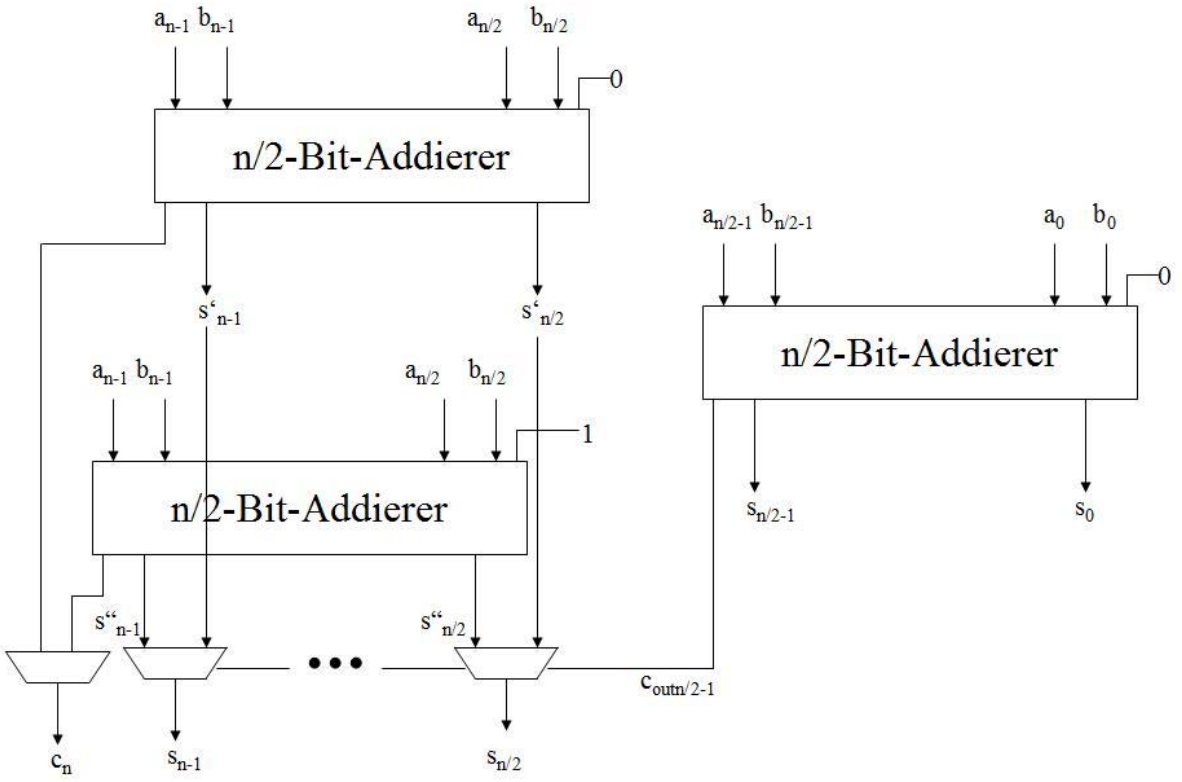


5.1.2 Carry-Select-Addierer

Wodurch ergibt sich die Schaltzeit für eine Addition oder Subtraktion? Wie kann die Addition beschleunigt werden? Durch die Anzahl der Volladdierer, durch die ein Carry nacheinander hindurchklappern muss. Wenn wir z. B. +1 und -1 addieren, liegt an den Eingängen des Addierers a = 00000001 und b = 11111111. Bei der Addition entsteht an der letzten Stelle ein Übertrag, dieser bewirkt an der vorletzten Stelle einen Übertrag usw. bis hin zur ersten Stelle, wo schließlich auch ein Übertrag entsteht. Der Zeitaufwand ist also die Anzahl der Stellen, durch die ein Übertrag hindurchwandern muss. Wenn jeder Volladdierer die Zeit t_{VA} benötigt, ist die Gesamtzeit also n * t_{VA}. Wie kann man diese Zeit nun vermindern? Eine hübsche Lösung, die auch in der Praxis der Rechnerarchitektur häufig Verwendung findet, bietet der **carry-select-adder**.

Die Idee ist folgende: Der Addierer wird in zwei gleich lange Hälften unterteilt. Und für beide Hälften wird gleichzeitig mit der Addition begonnen. Bei der linken (höher signifikanten) Hälfte wissen wir aber nicht, ob am Carry-Eingang des rechtesten Volladdierers eine 1 oder eine 0 ankommt. Deshalb führen wir die Addition der linken Hälfte gleichzeitig zweimal aus, einmal mit einer 0 am Carry-Eingang und einmal mit einer 1. Wenn die rechte Hälfte mit ihrer Addition fertig ist, kennen wir das eingehende Carry der linken Hälfte. Somit wissen wir, welches der Ergebnisse das richtige ist, das wir sodann auswählen (select). Das andere (falsche) Ergebnis wird einfach verworfen.

Die Auswahl geschieht über eine Menge von Multiplexern, die vom eingehenden Carry gesteuert werden. Das Prinzip ist auf dem folgenden Bild dargestellt.



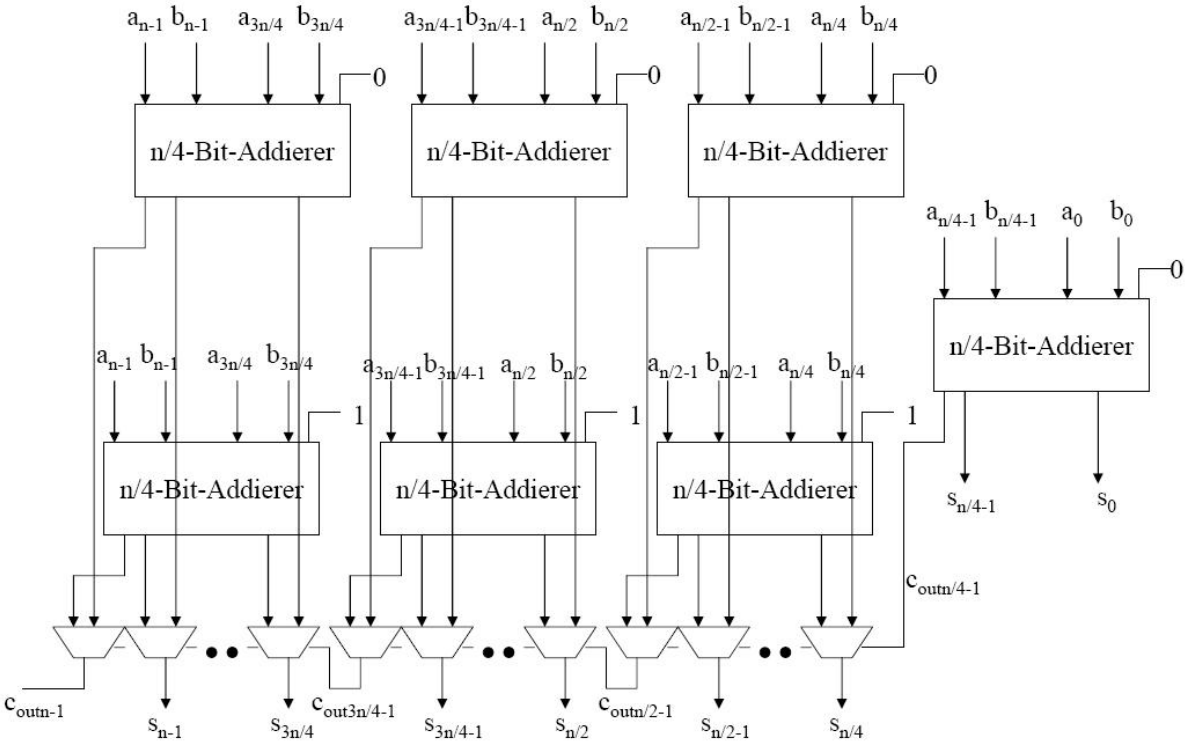
Wir sehen sofort: der Aufwand an Gattern ist etwas mehr als eineinhalb mal soviel wie beim ripple-carry-adder. Wie ist nun der Schaltzeitaufwand für einen solchen Addierer. Da alle n 2-Bit-Addierer gleichzeitig arbeiten benötigen wir nur noch die halbe Zeit, nämlich n/2 * t_{VA} für die Addition. Dazu kommt noch eine kleine konstante Zeit für die Multiplexer also ist die Gesamtzeit gleich n/2 * t_{VA} + t_{MUX}

Wir haben also etwa einen Faktor 2 in der Zeit gewonnen. Nun lässt sich dieses Prinzip natürlich wiederholt anwenden: Anstelle von Addierern der Länge n/2 können wir auch solche der Länge n/4 oder n/8 usw. verwenden. Alle solchen Addierer (außer dem am wenigsten signifikanten) werden doppelt ausgelegt, wovon einer mit einem Carryeingang 0 und der andere mit einem Carryeingang 1 arbeitet. Welches der Ergebnisse schließlich verwendet wird, entscheidet das Carry der nächst niedrigeren Stufe.

Das folgende Bild zeigt das Ergebnis dieser Technik für eine Unterteilung in vier Abschnitte. Die Laufzeit reduziert sich auf n/4 * t_{VA} + 3T_{MUX}. Allgemein gilt für eine Unterteilung in m Abschnitte:

$$t_{Gesamt} = n/m * t_{VA} + (m-1) * T_{MUX}$$

Diese Gesamtzeit nimmt ein Minimum an für $m = n^{1/2}$, also ist die Addition mit einem Carry-select-Addierer in $O(n^{1/2})$, während der Ripple-carry-Addierer in O(n) arbeitete.



5.1.3 Carry-Save-Addierer

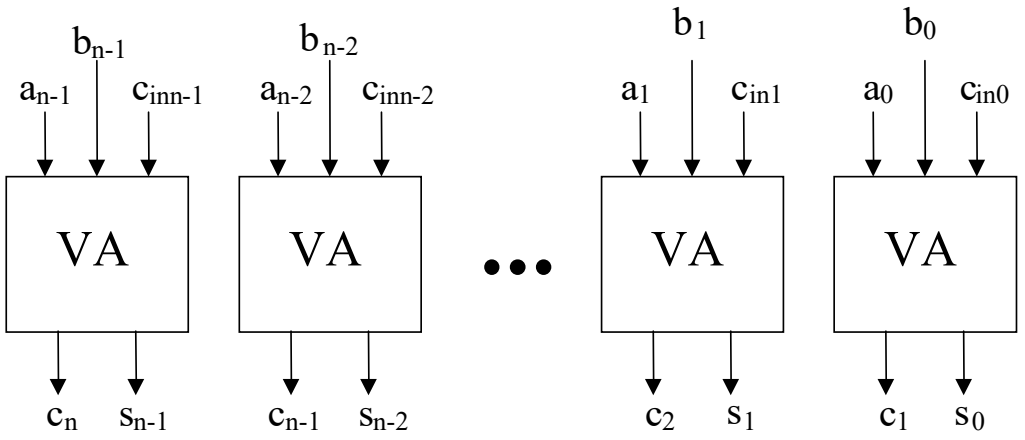
Ein anderer Ansatz ist die Verwendung einer redundanten Zahlendarstellung für die Zwischenergebnisse. Diese Technik stellt sicher, dass ein eventuell auftretendes Carry nur

einen Einfluss in seinem unmittelbaren Bereich hat, aber nicht zu einer „Kettenreaktion“ führen kann, wie beim ripple-carry-adder. Der hier vorgestellte Addierer heißt **carry-save-adder**.

Die Idee besteht darin, nicht zwei Operanden zu einem Ergebnis zu addieren, sondern ***drei*** Operanden zu ***zwei*** Ergebnissen. Dies erscheint zwar für unser Verständnis zunächst unnatürlich, es hat aber den Vorteil einer sehr einfachen und schnellen Realisierung. Die Anwendung eines solchen Carry-save-Addierers ist immer dann sinnvoll, wenn man mehrere Additionen nacheinander ausführen möchte. Und dies wiederum ist in Multiplizierern erforderlich. Wir werden daher sehen, wie ein extrem schneller Multiplizierer aus Carry-save-Addierern aufgebaut werden kann.

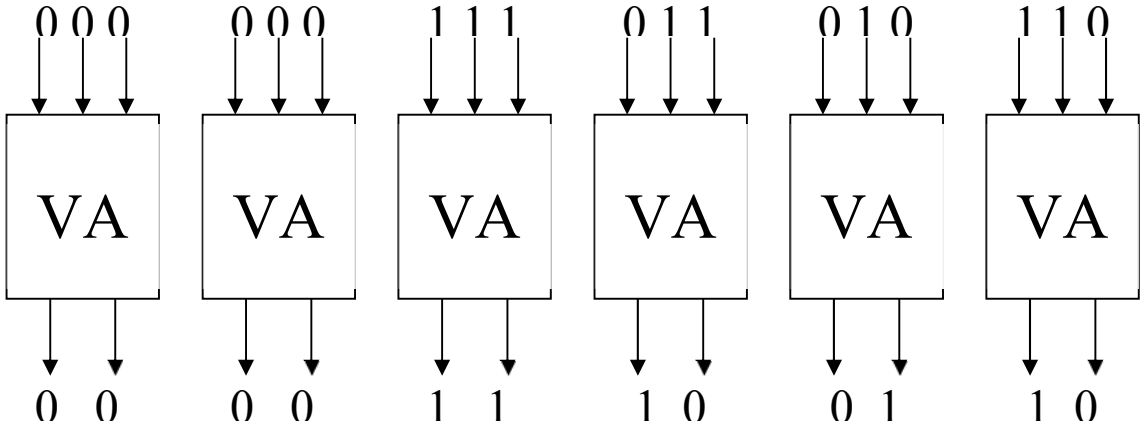
Der Aufbau eines Carry-save-Addierers ist lediglich die Parallelschaltung von n Volladdierern. Diese sind nun nicht verkettet, sondern jeder liefert zwei Ausgabebits, ein s-Bit und ein c-Bit. Aus diesen wird ein s-Wort und ein c-Wort gebildet, die zusammen die beiden Ergebnisworte darstellen. Das c-Wort wird durch eine 0 an der am wenigsten signifikanten Stelle ergänzt und das s-Wort um eine 0 an der höchst signifikanten Stelle.

Somit ist die Summe aus s- und c-Wort gleich der Summe der drei Operandenworte a, b, c_{in}.



Beispiel:

Wir addieren die Worte a = 001001, b = 001111, c_{in} = 001100



Die Ergebnisworte sind also c = 011010 und s = 001010. Wenn wir die drei Operanden im Dezimalsystem addieren, kommt 36 heraus. Dasselbe Ergebnis bekommen wir, wenn wie c und s addieren. Man beachte, dass bei dieser Art der Addition kein Carry „durchklappern“ kann. Die Zeit für eine Schaltung ist also t_{VA} und somit in O(1).

Wo können wir diese Art von Addition sinnvoll einsetzen?

Angenommen, wir müssen eine Kolonne von 64 Zahlen addieren. Dann können wir diese mit 62 Carry-Save-Additionen zusammenzählen, so dass schließlich zwei Ergebnisworte berechnet werden. Diese müssen dann mit einem „richtigen“ Addierer, z.B. einem Carry-Select-Addierer zu einem Endergebnis zusammengezählt werden. Die 62 Additionen benötigen 62 * t_{VA}. Die letzte Addition benötigt 8 * t_{VA} + 7 t_{MUX}. Der Zeitaufwand gesamt ist also 70t_{VA} + 7 t_{MUX}. Hätten wir die gesamte Addition mit einem Carry-Select-Addierer gemacht, würden wir zusammen 63 * (8 t_{VA} + 7 t_{MUX}) = 504 t_{VA} + 441 t_{MUX} .

Man sieht, um wie viel sparsamer die Carry-Save-Addition in diesem Falle ist. Allgemein gilt: Wenn wir m Additionen der Länge n Bit machen wollen, benötigen wir mit einem Ripple-Carry-Addierer Zeit O(n*m), mit einem Carry-Select-Addierer Zeit O($n^{1/2}$*m) und mit einem Carry-Save-Addierer (mit nachgeschaltetem Carry-Select-Addierer für den letzten Schritt) Zeit O($n^{1/2}$ +m).

Die optimale Zeit bei der Addition zweier Zahlen erhält man mit einem sogenannten Carry-Lookahead-Addierer. Dieser benötigt nur die Zeit O(log n) für eine Addition. Aus Zeitgründen wird dieser Addierertyp aber an dieser Stelle nicht behandelt.

5.2 Multiplikation

Bei der Multiplikation nach der Schulmethode wird für jede Stelle eines Operanden das Produkt dieser Stelle mit dem anderen Operanden berechnet. Danach werden alle diese Produkte addiert. An dieser Stelle können wir den soeben erlernten Carry-Save-Addierer einsetzen, denn jetzt haben wir den Fall einer großen Anzahl von Operanden, die addiert werden müssen.

Beispiel:

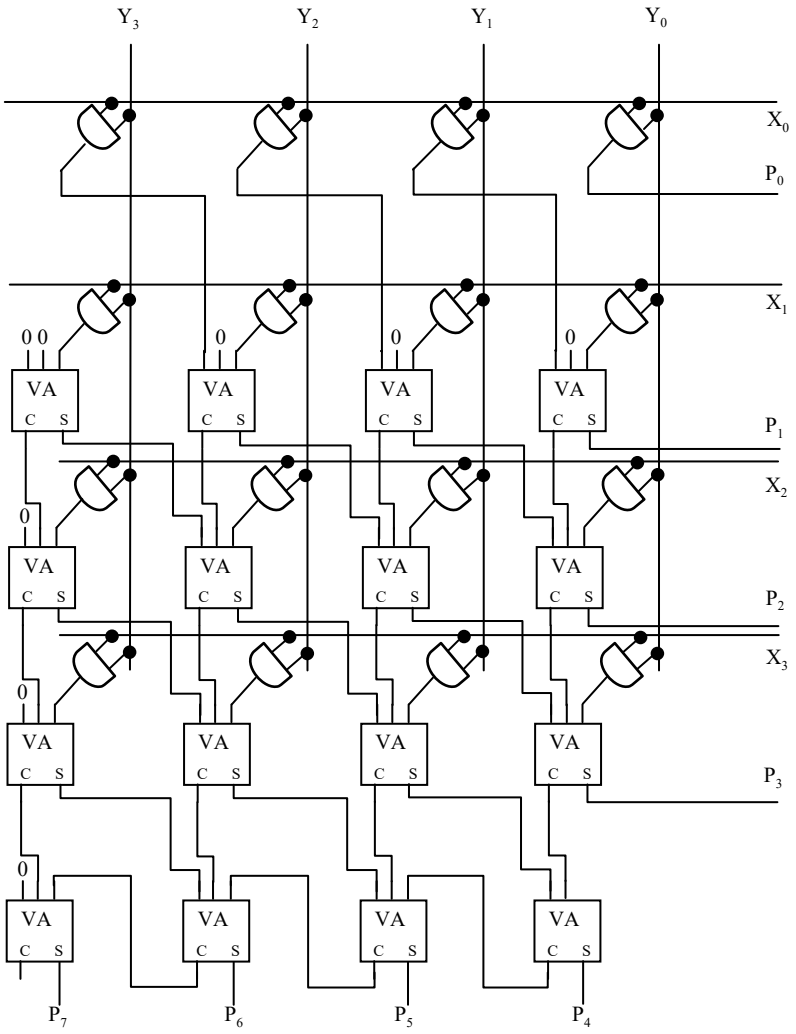
```
10110011 * 10010111

    10110011
     00000000
      00000000
       10110011
        00000000
         10110011
          10110011
           10110011
-------------------
0110100110010101
```

Ein Carry-Save-Multiplizierer für zwei Operanden der Länge n besteht aus n^2 AND-Gattern, die gleichzeitig alle erforderlichen 1-Bit-Multiplikationen ausführen (Die binäre Multiplikation von 1-Bit Zahlen ist gerade das logische „UND“). Danach sind nur noch alle Teilprodukte (wie bei der Schulmethode) zu addieren. Dies geschieht nun in der bekannten 3-auf-2 Operanden Manier, die wir soeben beim Carry-Save-Addierer kennengelernt haben. Am Ende ist für die höchstsignifikanten n Bits noch eine (klassische) 2-auf-1-Operanden-Addition erforderlich. Diese wird mit einem konventionellen Addierer ausgeführt.

Wie lang ist die Verarbeitungszeit für eine solche Multiplikation? Wir müssen in diesem Schaltnetz den „kritischen Pfad“ suchen, also den Pfad, bei dem ein Signal durch die maximale Anzahl von Schaltelementen hindurchwandern muss, bevor das Endergebnis berechnet ist. Dieser Pfad besteht zunächst einmal aus den n-2 Stufen, bei denen jeweils ein Operand neu hinzuaddiert wird plus die n-1 Volladdierer, durch die ein Carry bei der letzten Addition hindurchklappern muss (wir setzen hier einen ripple-carry-adder voraus).

Ein Beispiel für einen solchen 4-Bit Multiplizierer sehen wir auf dem nächsten Bild.

4-Bit-carry-save-Multiplizierer



5.3 Division

Einige Prozessoren haben eigene Divisionseinheiten, die in der Regel ähnlich des Carry-Save-Adders eine interne Darstellung der Zwischenergebnisse durch zwei Worte benutzt.

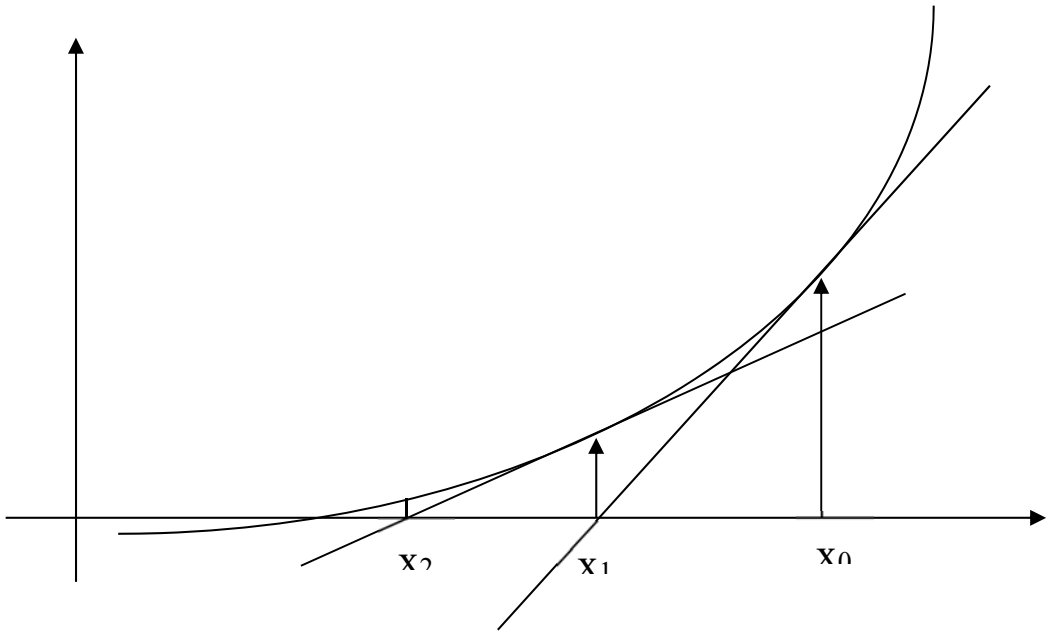
Andererseits ist die Division eine seltene Operation. Daher (make the common case fast) verzichten viele Prozessoren auf eigene Hardware für die Division, sondern implementieren sie in Software. Ein gängiges Verfahren dafür ist die Newton-Raphson-Methode, die wir hier kennen lernen wollen.

Vorweg sei erwähnt, dass frühere Rechner (< 1980) die Division in der Regel (ebenfalls in Software) entsprechend der Schulmethode ausführten, d.h. die Ergebnisbits werden eins nach dem anderen berechnet durch Vergleich des Divisors mit den verbleibenden höchstsignifikanten Stellen des Dividenden. Wenn der Divisor größer ist, ergibt sich ein Bit 0 sonst ein Bit 1. Im letzteren Falle wird sodann der Divisor von den höchstsignifikanten Stellen des Dividenden subtrahiert und eine weitere Stelle des Dividenden wird für die nächste Ergebnisstelle herangezogen.

Dieses Verfahren hat natürlich die Komplexität von $O(n^2)$, wenn der Dividend n Stellen hat. Genauer: Man braucht n Schritte, und in jedem Schritt muss ein Vergleich und eine Subtraktion ausgeführt werden. Das erwies sich zu Zeiten steigender Rechenleistung als zu langsam. Daher suchte man nach Verfahren, die in weniger Schritten zu genauen Ergebnissen führten.

Die Idee des Newton-Verfahrens ist die Approximation der Nullstelle einer Funktion durch Konstruktion einer Folge von Werten, die sehr schnell gegen die Nullstelle konvergiert. Man beginnt damit, dass man die Tangente an die Funktion in einem geschätzten Anfangswert anlegt und deren Schnittpunkt mit der Abszisse als nächsten Folgenwert berechnet. Nun legt man an dessen Funktionswert die Tangente an usw. Wenn die Funktion bestimmte Bedingungen erfüllt und wenn der Anfangswert geeignet gewählt ist, konvergiert diese Folge gegen die Nullstelle.

Beispiel:



Für zwei Werte x_i und x_{i+1} in dieser Folge gilt:

$$f'(x_i) = \frac{f(x_i)}{x_i - x_{i+1}}$$

Aufgelöst nach x_{i+1} bedeutet das

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Betrachten wir nun f(x) = 1/x - B . Diese Funktion hat in 1/B eine Nullstelle. Wenn wir in die obige Formel einsetzen, ergibt sich

$$x_{i+1} = x_i - \frac{\frac{1}{x_i} - B}{-\frac{1}{x_i^2}} = x_i + \left(x_i - Bx_i^2\right) = x_i(2 - Bx_i)$$

Damit haben wir eine sehr einfache Iterationsformel, die mit zwei Multiplikationen und einer Subtraktion für einen Iterationsschritt auskommt.

Wie viele Schritte benötigen wir aber, oder anders gefragt, wie schnell konvergiert die Folge?

Betrachten wir den Fehler ε, also die Differenz des Folgenwertes x_i von der gesuchten Nullstelle 1/B. Es gilt:

$$x_{i+1} = 2\left(\frac{1}{B} - \varepsilon\right) - B\left(\frac{1}{B} - \varepsilon\right)^2 = \frac{1}{B} - B\varepsilon^2$$

Somit ist der Fehler nach der nächsten Iteration nur noch $B\varepsilon^2$. Wenn B nun zwischen 0 und 1 liegt, bedeutet dies, dass wir eine quadratische Konvergenz haben, genauer: wenn das Ergebnis nach der i-ten Iteration bereits auf m Bits genau ist, ist es nach der i+1-ten Iteration auf 2m Bits genau.

Wir müssen also dafür sorgen, dass B zwischen 0 und 1 liegt und dass x_0 auf 1 Bit genau ist. Dann wird x_1 auf zwei Bits genau sein, x_2 auf vier Bits usw., x_i auf 2^i Bits genau.

Angenommen, wir müssen a/b berechnen. Dann können wir dies mit einer Multiplikation als a * 1/b berechnen, wobei wir in der Lage sein müssen, den Kehrwert der Zahl b also 1/b zu ermitteln. Wenn b zwischen 0 und 1 ist und die erste Stelle nach dem Komma eine 1 ist (also normalisiert), dann geht das mit obiger Iteration. Wenn nicht, müssen wir $B = 2^k * b$ nehmen mit einem geeigneten k, so dass B normalisiert ist. Sodann berechnen wir 1/B und multiplizieren dies schließlich mit 2^{-k} (Bitverschiebung).

Fazit: Durch die Iteration wird der Aufwand von 2n Operationen auf 3log n Operationen reduziert, nämlich log n Iterationen und in jeder drei Operationen. Bei einer 64-Bit Division ist das eine Reduktion von 128 auf 18 Operationen.