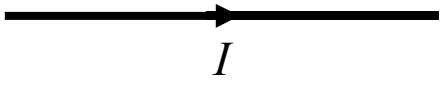
4. Schaltnetze

- **Technische Realisierung von Schaltfunktionen**
  - Halbleitertechnik
    - meist basierend auf Silizium, aber auch Germanium oder GaAs sind Halbleiter
      - Leitfähigkeit liegt zwischen der von Leitern (Metallen) und Isolatoren
    - Transistoren
      - heute dominiert die CMOS-Schaltungstechnik auf Basis von Isolierschicht-Feldeffekttransistoren (MOSFET)

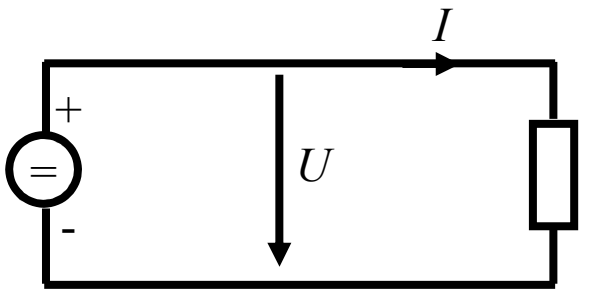
Grundlagen der Elektrotechnik

- **Ladung**
  - Eigenschaft der Materie
  - es gibt positive (Atomkerne) und negative (Elektronen) Ladungen
    - gleichnamige Ladungen stoßen sich ab, entgegen gesetzte Ladungen ziehen sich an
  - Ladungsmenge, meist mit Q bezeichnet, gemessen in Coulomb (C)
- **Strom**
  - bewegte Ladungen
    - in Metallen sind die Elektronen die beweglichen Ladungsträger
  - Stromstärke ist die Ladungsmenge, die pro Zeiteinheit an einer Stelle vorbeifließt, meist mit I bezeichnet
  - gemessen in Ampere (A), 1 A = 1 C/s
  - der Stromzählpfeil zeigt in der Regel in technische Stromrichtung von + nach -
    - muss er aber nicht, dann ist der Strom eben negativ (z.B. I = -3 mA)



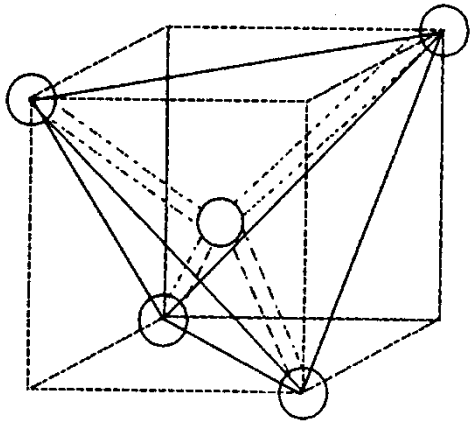
Grundlagen der Elektrotechnik (2)

- **Elektrische Energie**
  - In Ladungsträgern kann Energie stecken, die frei wird, wenn sich die Ladungsträger von einem Ort zu einem anderen bewegen.
    - Energie wird gemessen in Joule (J)
  - Ladungsträger sind bestrebt, sich von Orten mit höherer Energie zu Orten mit niedrigerer Energie zu bewegen.
  - Damit ist die Energiedifferenz die Ursache für den elektrischen Strom.
- **Spannung**
  - Spannung ist definiert als Energie pro Ladungsmenge, meist mit *U* bezeichnet
  - gemessen in Volt (V), 1 V = 1 J/C
  - Spannung wird immer zwischen zwei Punkten gemessen
  - der Zählpfeil zeigt in der Regel von + nach –
    - muss er aber nicht, dann ist die Spannung eben negativ (z.B. $U = -5V$)

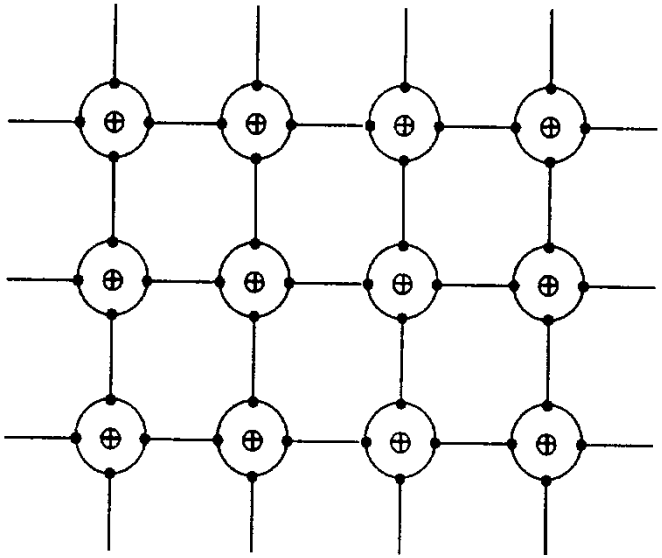


Kristalle

- Ge und Si Atome haben in ihrer äußersten Schale 4 Elektronen (Valenzelektronen), die bestrebt sind, sich mit je einem Elektron von je einem Nachbaratomen zu paaren
- die Atome gehen dadurch chemische Bindungen ein
- für Halbleiterbauelemente werden Einkristalle extrem hoher Reinheit (nur 1 Fremdatom auf 10^{10} Si-Atome) benötigt



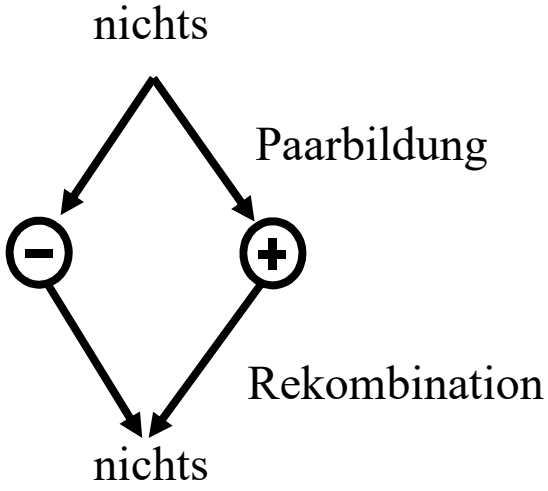
3d-Tetraeder-struktur



meist nur symbolische 2d-Darstellungen

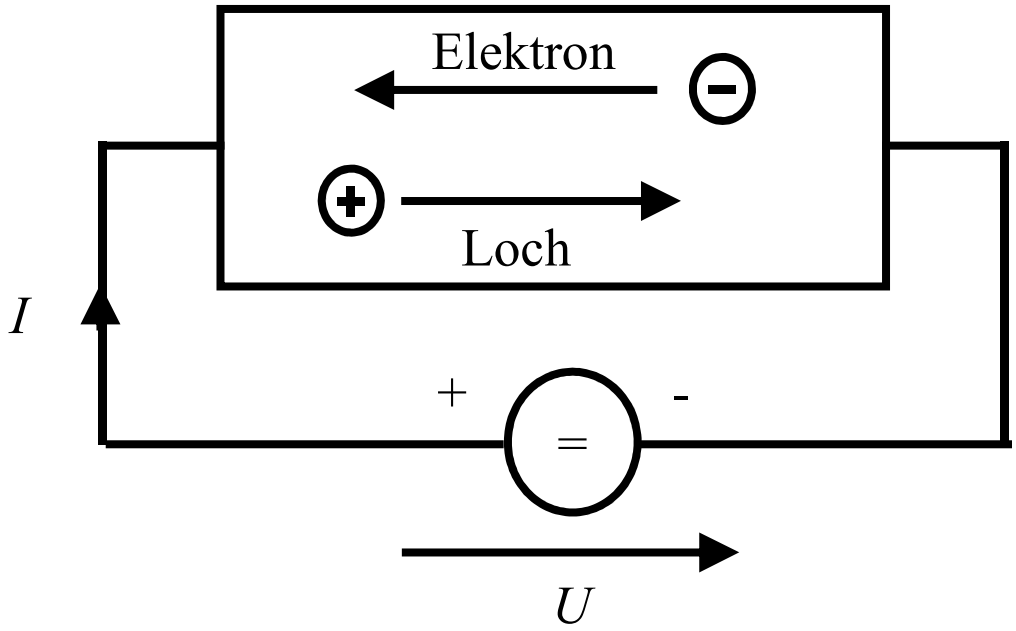
Eigenleitfähigkeit von Halbleitern

- durch Aufnahme von Wärmeenergie werden einige Atombindungen aufgebrochen, d.h. Elektronen lösen sich vom Atomkern und bewegen sich frei im Kristall
- der feststehende Atomrumpf ist dann positiv geladen
- das fehlende Elektron kann von einem Elektron einer benachbarten Bindung aufgefüllt werden
- dann fehlt am Nachbaratom ein Elektron, das Atom ist daher positiv geladen
- die wandernde positive Ladung nennt man Loch (Defektelektron)
- ein Loch verhält sich genau wie ein positiver, frei beweglicher Ladungsträger
- fällt ein frei bewegliches Elektron in ein Loch, so löschen sich beide gegenseitig aus (Rekombination)



Eigenleitfähigkeit von Halbleitern (2)

- legt man eine Spannung an den Halbleiter an, bewegen sich beide Arten von Ladungsträgern und es fließt ein Strom
- die Anzahl der freien Ladungsträger ist aber sehr gering und temperaturabhängig

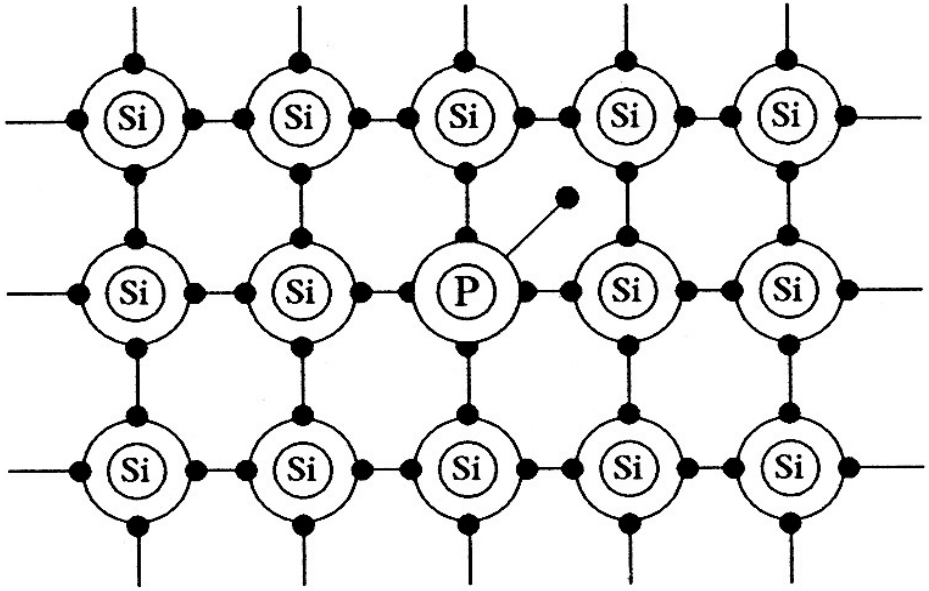


Störstellenleitfähigkeit

- **dotierte Halbleiter**
  - Eigenleitfähigkeit bei Zimmertemperatur ist sehr gering
  - durch gezielte Zusetzung von Fremdatomen kann die Leitfähigkeit um einige Zehnerpotenzen erhöht werden (Dotierung)
  - man verwendet Fremdatome mit 5 oder 3 Valenzelektronen
    - dadurch entstehen n- bzw. p-dotierte Halbleiter (s.u.)

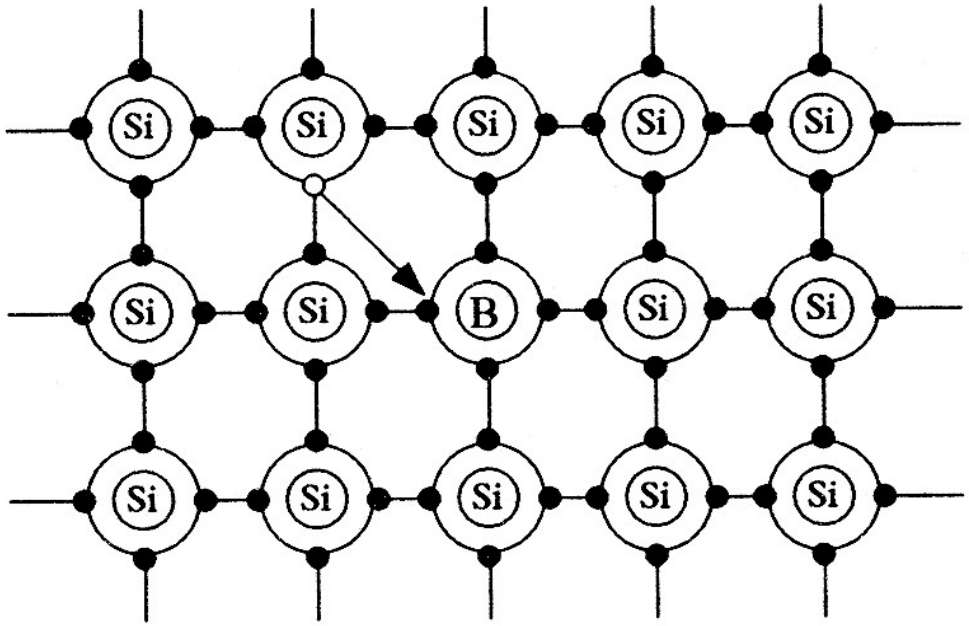
n-dotierte Halbleiter

- Fremdatome mit 5 Elektronen in der äußersten Hülle
  - Arsen As, Antimon Sb, Phosphor P
  - werden in geringer Menge in das Kristallgitter mit eingebaut
  - nur vier Elektronen werden für die Bindungen benötigt
  - das fünfte Elektron ist ganz lose an den Atomkern gebunden, da es keinen Bindungspartner findet
  - schon bei Zimmertemperatur sind diese Elektronen frei beweglich



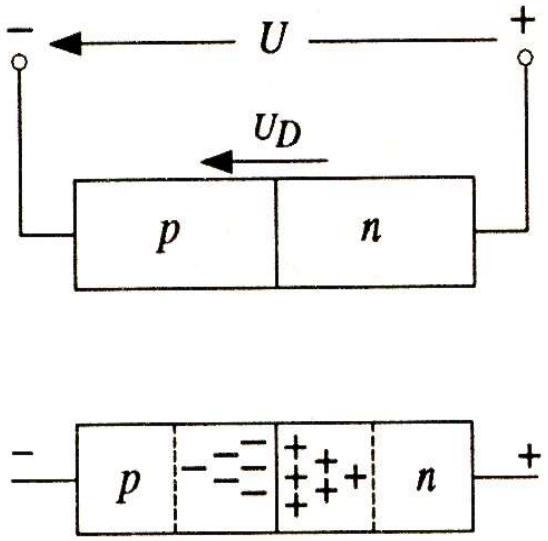
p-dotierte Halbleiter

- Fremdatome mit 3 Elektronen in der äußersten Hülle
  - Aluminium Al, Bor B, Indium In
  - werden in geringer Menge in das Kristallgitter mit eingebaut
  - es stehen nur drei Elektronen für die Bindungen zur Verfügung
  - das fehlende vierte Elektron wird sehr leicht von einem Nachbaratom zur Verfügung gestellt
  - es entsteht ein frei bewegliches Loch

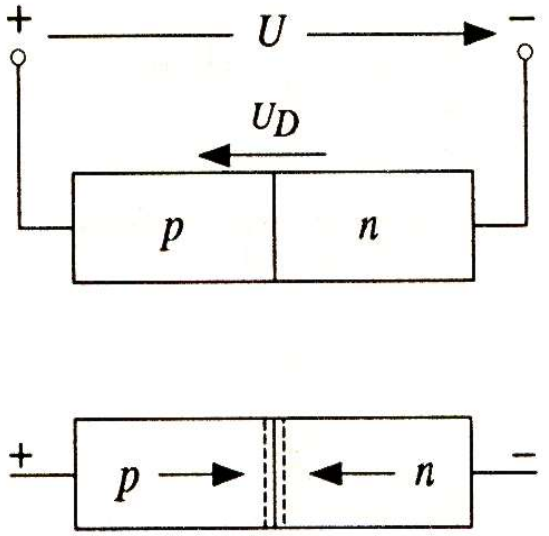


pn-Übergang

- Elektronen wandern von der n-dotierten Zone in die p-dotierte Zone und rekombinieren mit den Löchern, bis die entstehenden Raumladungen den Prozess unterbinden
- keine beweglichen Ladungsträger in der so entstandenen Sperrzone
- eine angelegte Spannung verstärkt oder verhindert den Effekt



Sperrrichtung

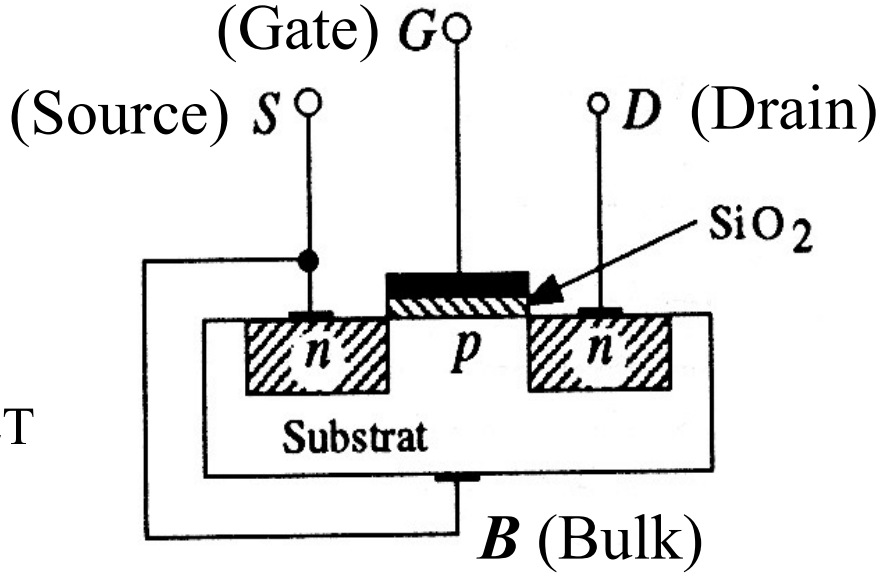


Durchlassrichtung

MOSFET

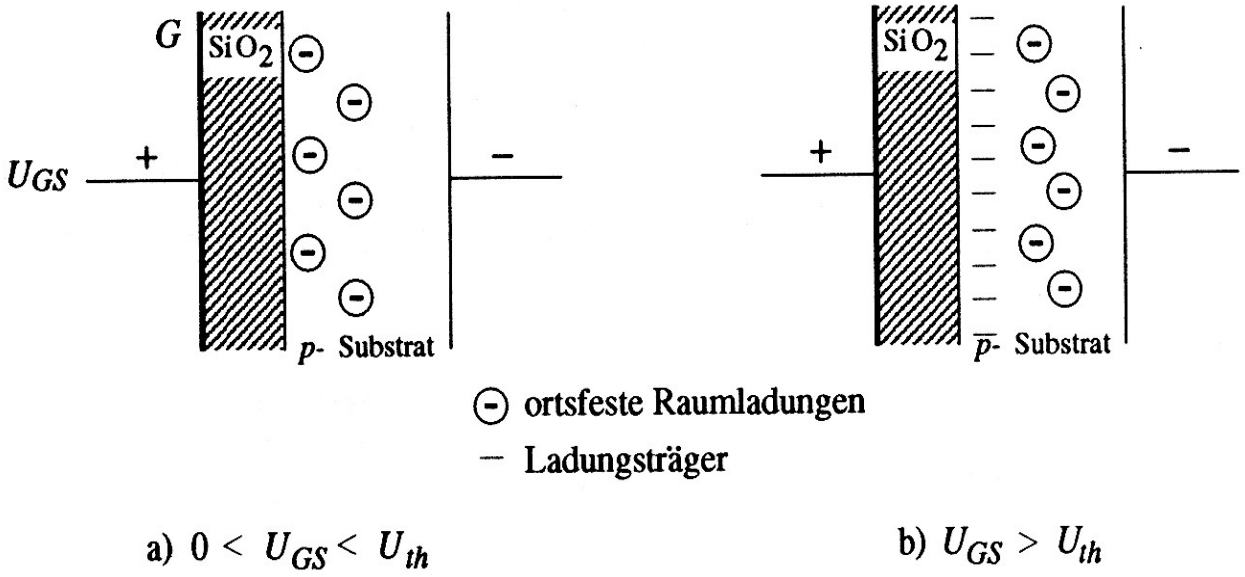
- **Isolierschicht-Feldeffekt-Transistor**
  - die Gateelektrode ist durch eine dünne Siliziumoxidschicht (SiO_2, sehr guter Isolator) vom eigentlichen Halbleiter getrennt
    - Metal-Oxide-Semiconductor (MOSFET)
    - manchmal auch Metal-Insulator-Semiconductor (MISFET) genannt

Aufbau:
n-Kanal-MOS-FET



MOSFET (3)

- **Funktionsweise**
  - exemplarisch für n-Kanal MOSFET (p-Kanal analog)
  - eine positive Spannung am Gate gegenüber dem Substrat zieht die auch in der p-dotierten Schicht immer noch vorhandenen Elektronen an, bzw. erzeugt neue Elektronen durch Paarbildung
  - ab einer Schwellspannung U_{th} (Threshold-Spannung) entsteht eine leitfähige Schicht von Elektronen (n-Kanal)

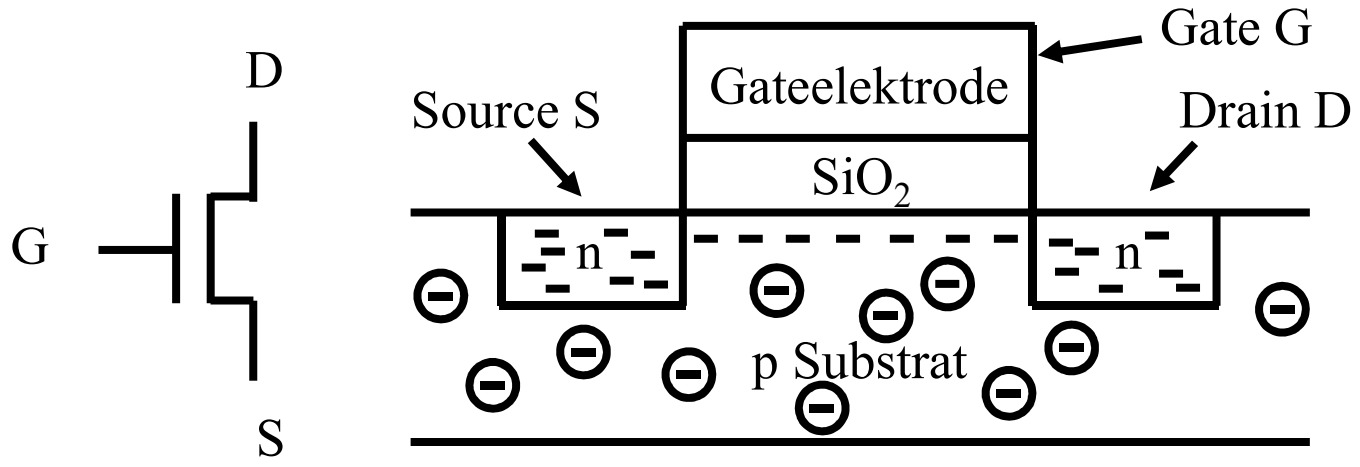


MOSFET (3)

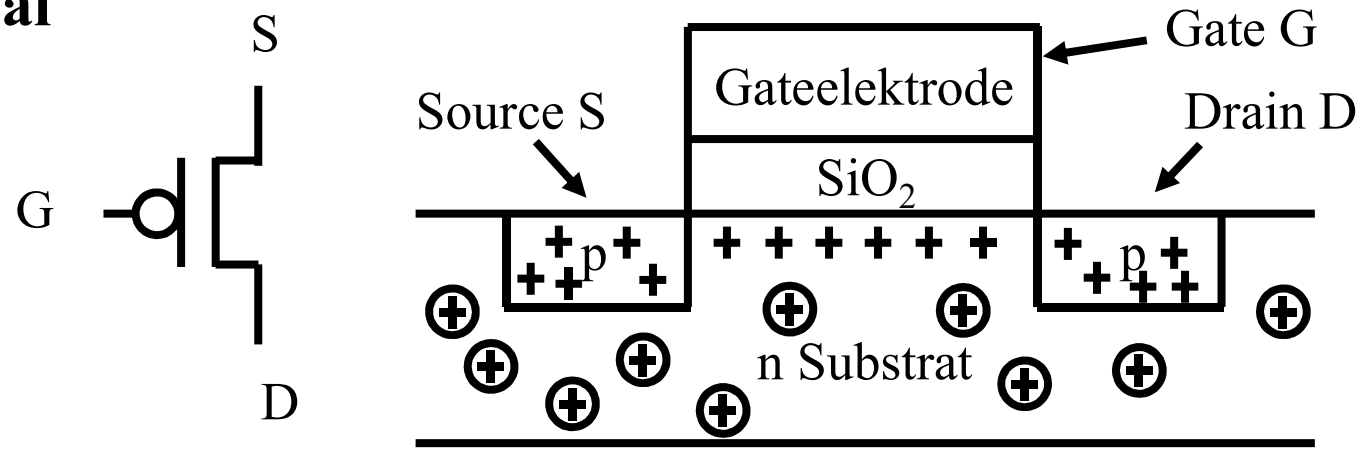
- **beim p-Kanal MOSFET ist alles umgekehrt**
  - dort wird ein n-Substrat verwendet
  - Löcher sind die beweglichen Ladungsträger
  - eine negative Spannung baut den Kanal aus Löchern auf
  - Bulk wird wieder mit Source verbunden

MOSFET: vereinfachte Darstellung

- **n-Kanal**



- **p-Kanal**



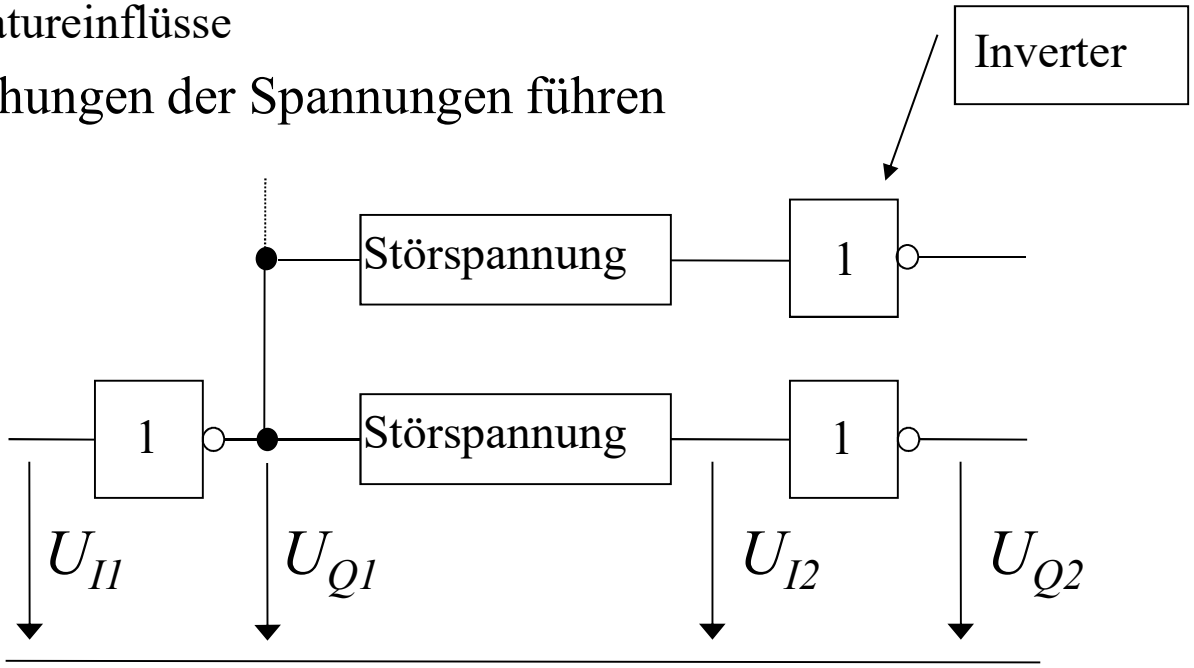
Verknüpfungsglieder

- **Verknüpfungsglieder (Gatter)**
  - in digitalen Schaltungen werden die Gesetze der Schaltalgebra mit Hilfe von elektronischen Verknüpfungsgliedern realisiert
  - die Wahrheitswerte `wahr` und `falsch` werden durch die Zustände an und aus (oder hohe und niedrige Spannung) realisiert
  - damit man Verknüpfungsglieder zu größeren Einheiten zusammenschalten kann, ist es erforderlich, dass die Schaltungen alle die gleichen Signalpegel benutzen und die Signallaufzeiten vergleichbar sind
  - deshalb sind Verknüpfungsglieder *standardisiert*
  - man spricht auch von Schaltkreisfamilien (z.B. TTL, ECL, NMOS, PMOS, ***CMOS***)
  - innerhalb einer Familie sind die Schaltungen nach denselben Konzepten mit derselben Art von Bauelementen realisiert

Signalpegel

- ein Schaltglied steuert normalerweise mehrere nachfolgende Schaltglieder an
- dabei können
  - Fertigungsschwankungen
  - Betriebsspannungsschwankungen
  - Störungen von anderen Leitungen
  - Temperatureinflüsse

  zu Verfälschungen der Spannungen führen



Signalpegel (2)

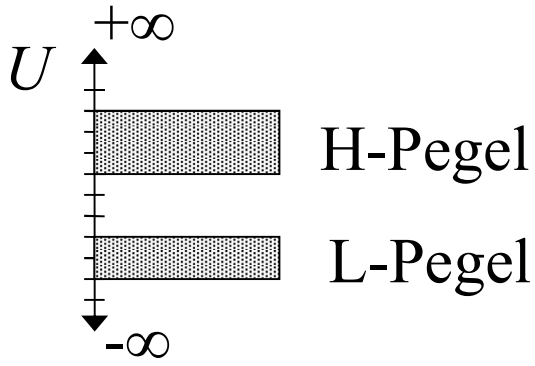
- daher werden für an/aus nicht zwei scharf definierte Spannungen spezifiziert, sondern Spannungs***bereiche***, der H- und der L-Pegel
- zwei mögliche Zuordnungen
  - positive Zuordnung (*active high*)

    H=1

    L=0
  - negative Zuordnung (*active low*)

    H=0

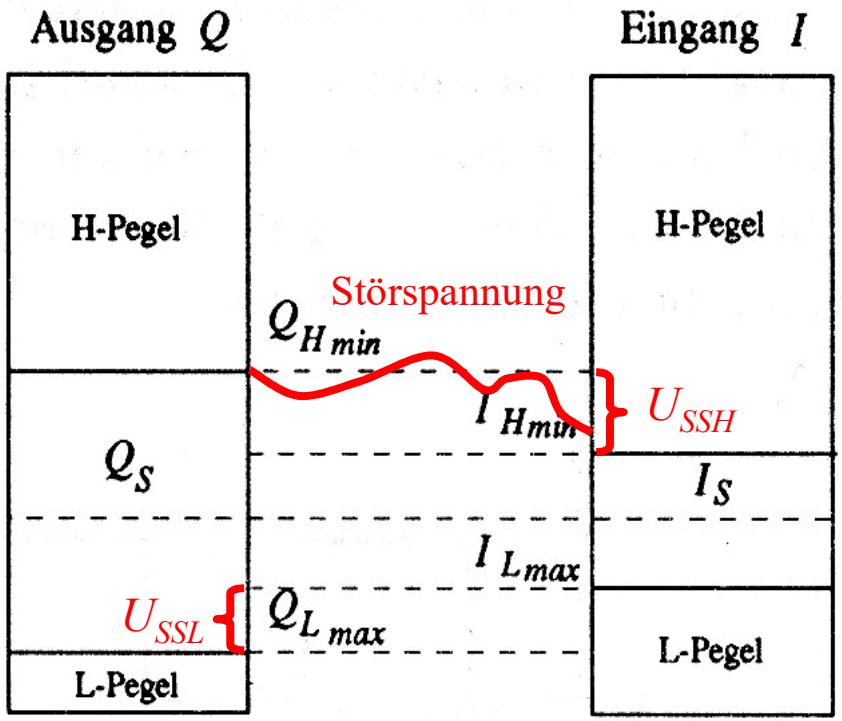
    L=1



- die Zuordnung ist willkürlich, meist wird jedoch die positive (*active high*) verwendet

Statische Störsicherheit

- schaltet man zwei Schaltglieder hintereinander, ist die Ausgangsspannung des ersten gleichzeitig die Eingangsspannung des zweiten Gliedes
- damit die Zuordnung eindeutig bleibt, müssen bestimmte Grenzwerte bei den Pegelbereichen eingehalten werden
- die erlaubten Spannungs-bereiche an Eingängen sind größer als an Ausgängen, um Störungen tolerieren zu können



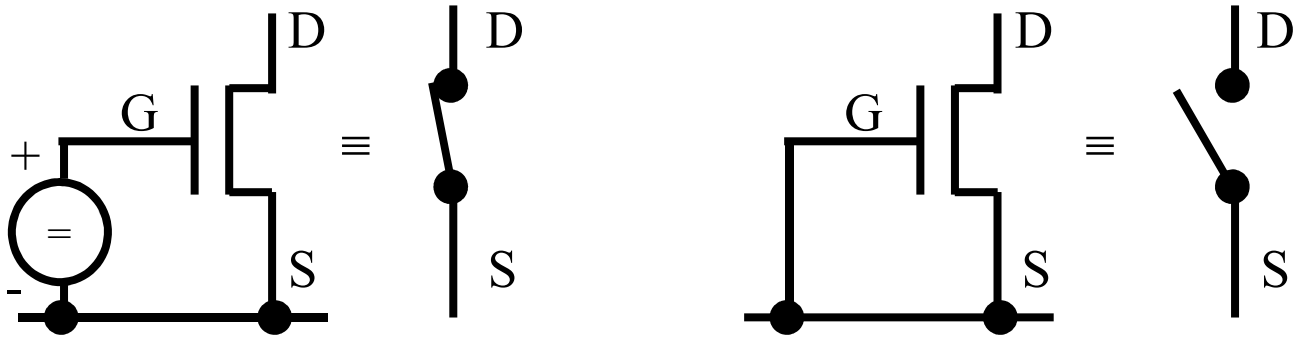
Statische Störspannungsabstände:

$$U_{SSH} = U_{QH\,\min} - U_{IH\,\min}$$

$$U_{SSL} = U_{IL\,\max} - U_{QL\,\max}$$

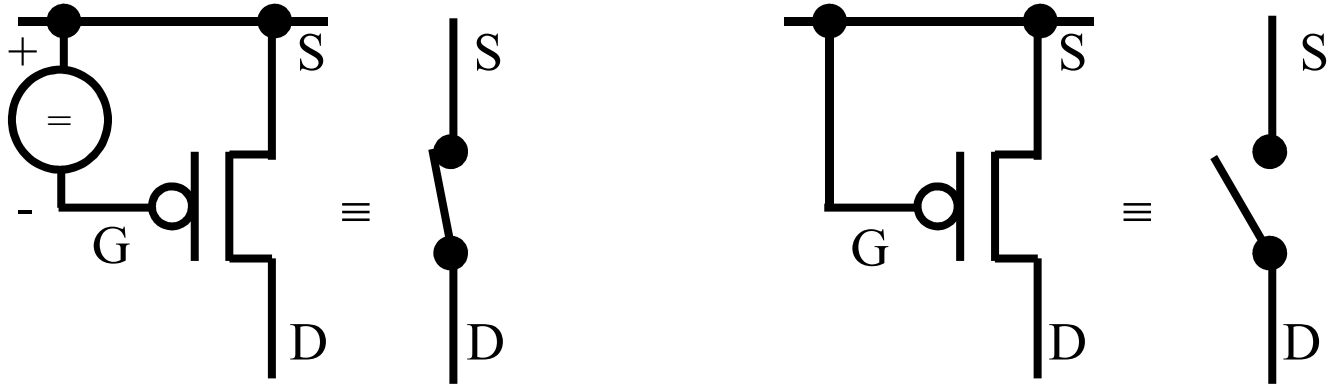
MOSFET

- **MOSFET arbeitet wie ein Schalter**
  - Größe der Spannung an Gate bestimmt, ob Schalter zwischen Source und Drain geöffnet oder geschlossen ist
- **Beispiel n-Kanal MOSFET**
  - positive Spannung an G gegenüber S schließt den Schalter
  - eine geringe Spannung öffnet den Schalter



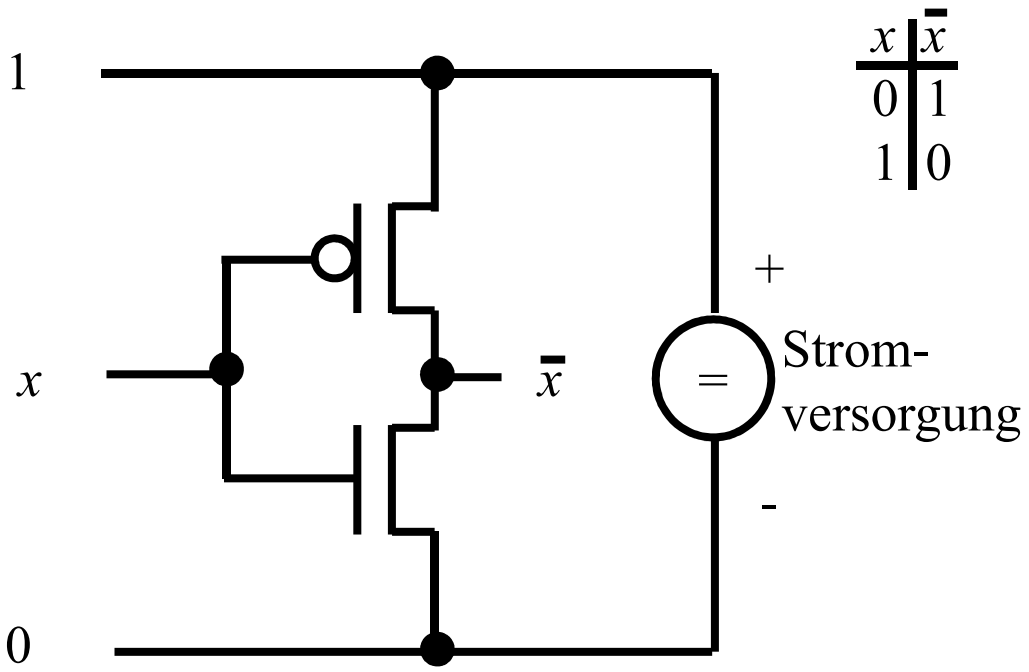
MOSFET (2)

- **p-Kanal MOSFET benötigt negative Spannung, um Schalter zu schließen**
  - alles ist umgekehrt
  - positive Löcher bilden den p-Kanal, daher wird eine negative Spannung zur Bildung des Kanals benötigt



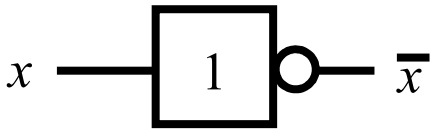
CMOS Inverter

- **CMOS Inverter, NOT-Gatter**
  - C steht für *complementary*, also komplementär
    - Kombination aus n-Kanal und p-Kanal MOSFET
  - hier gilt (*active high*): eine hohe, positive Spannung bedeutet eine 1
  - eine niedrige Spannung (bzw. 0V) bedeutet 0

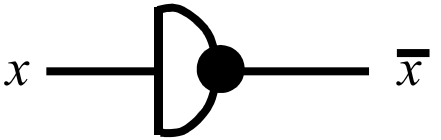


x	$\bar{x}$
0	1
1	0

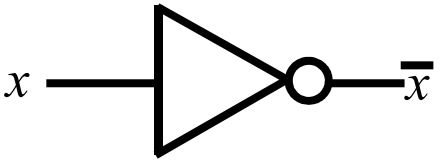
DIN:



früher:

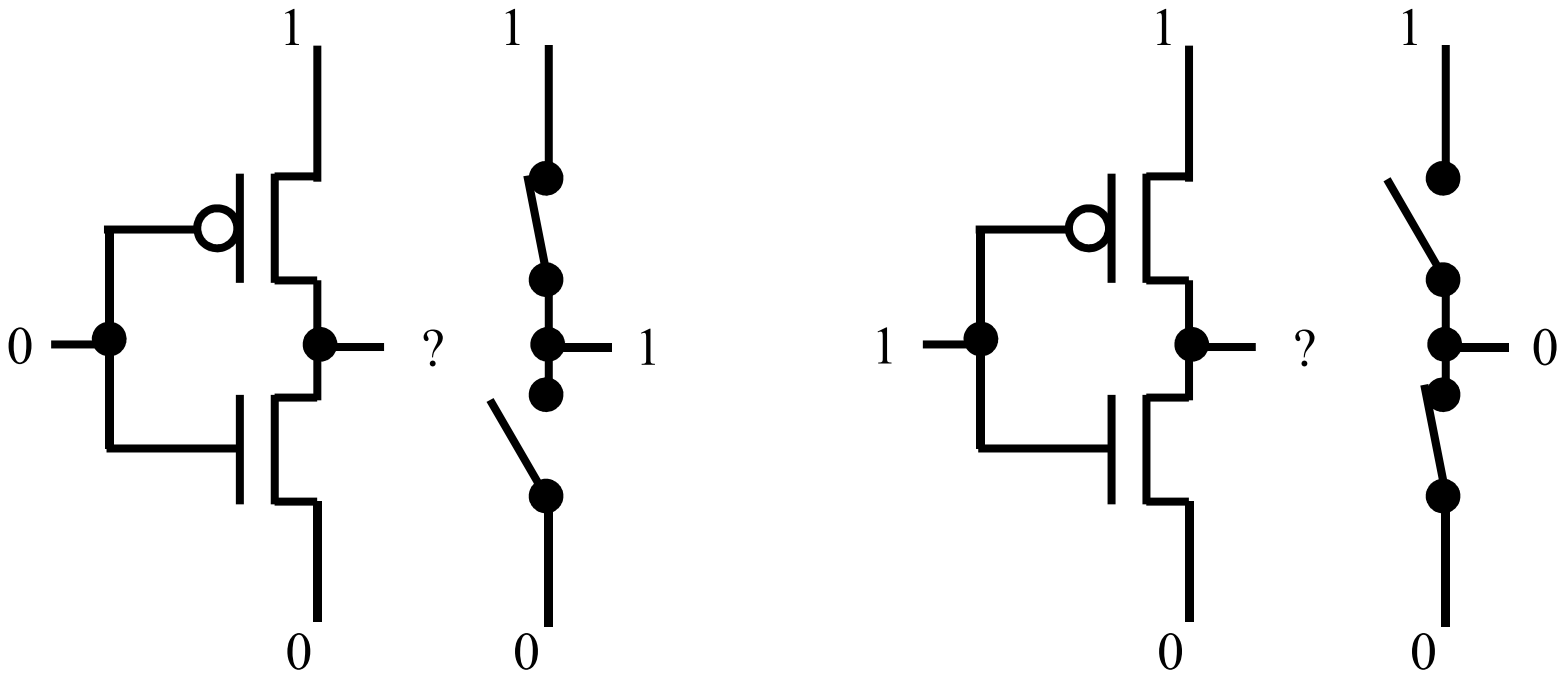


IEEE:



CMOS Inverter (2)

Funktionsweise

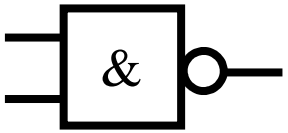


CMOS NAND-Gatter

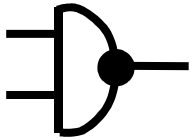
NAND

a	b	$\overline{a \bullet b}$
0	0	1
0	1	1
1	0	1
1	1	0

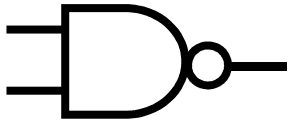
DIN:

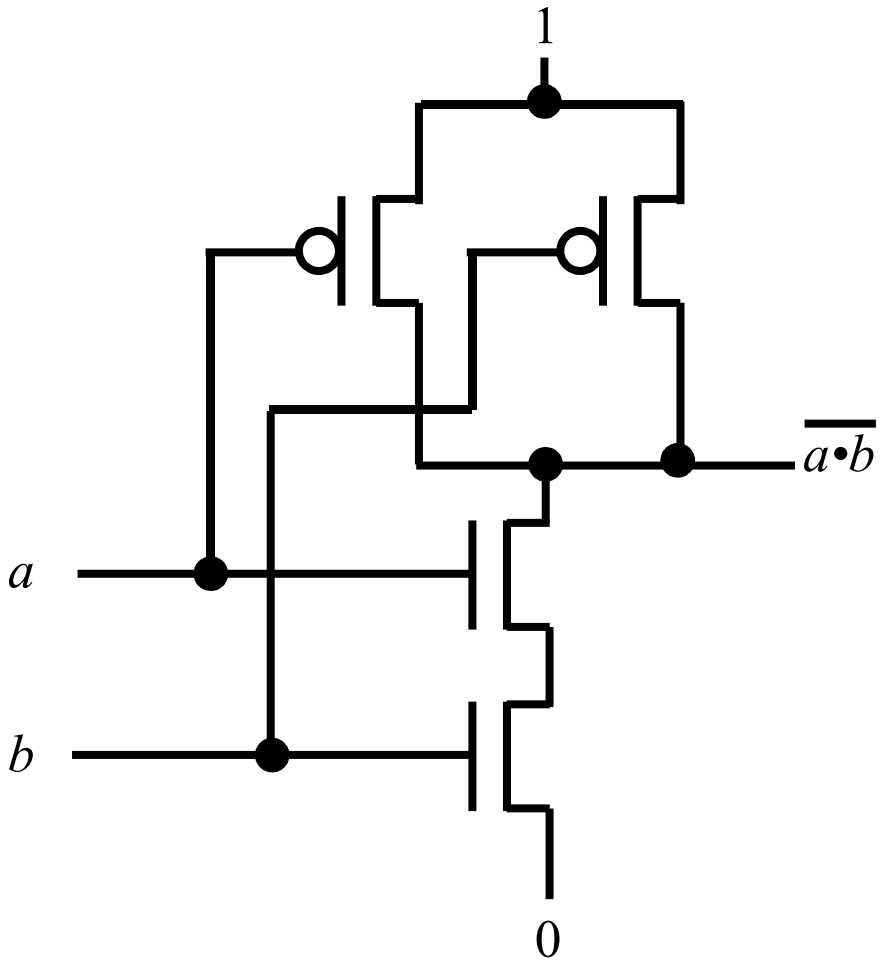


früher:



IEEE:



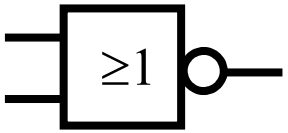


CMOS NOR-Gatter

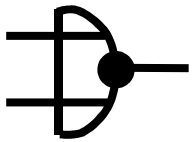
NOR

a	b	$\overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

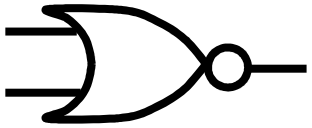
DIN:

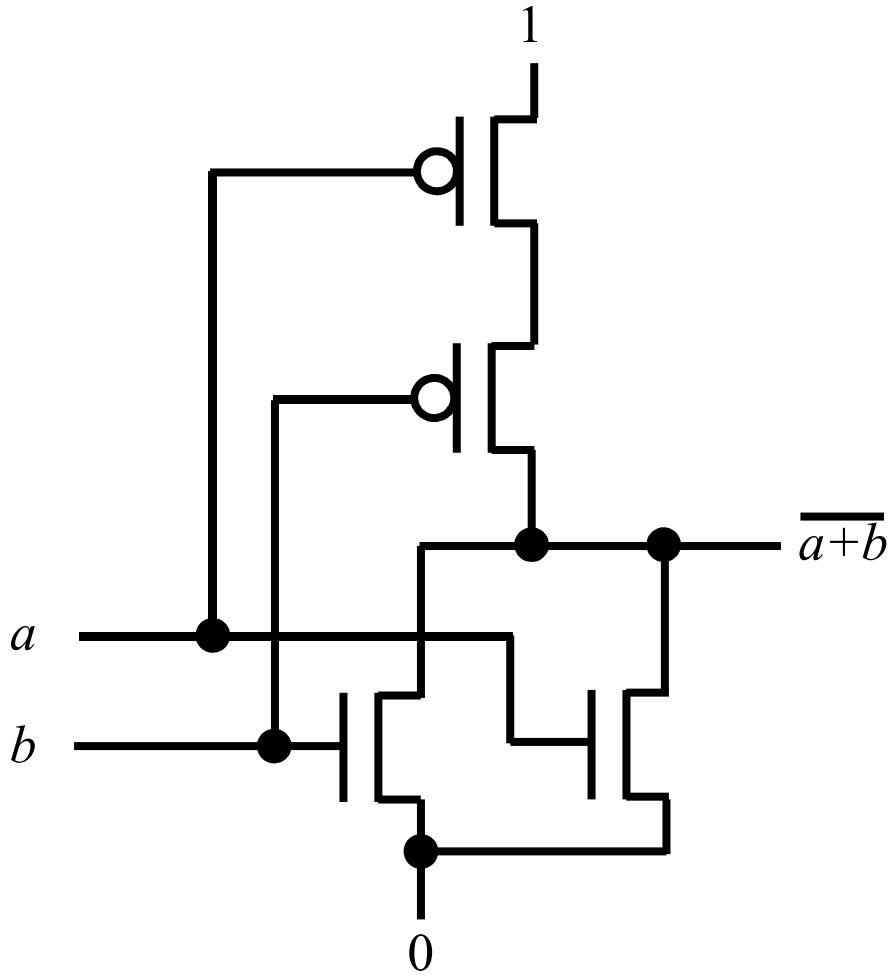


früher:



IEEE:

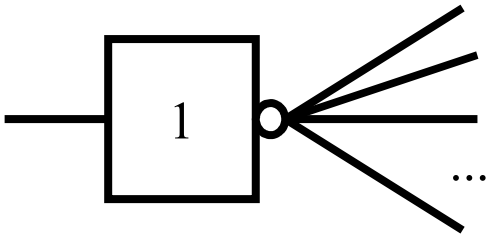




Ausgangslastfaktor (Fan-out)

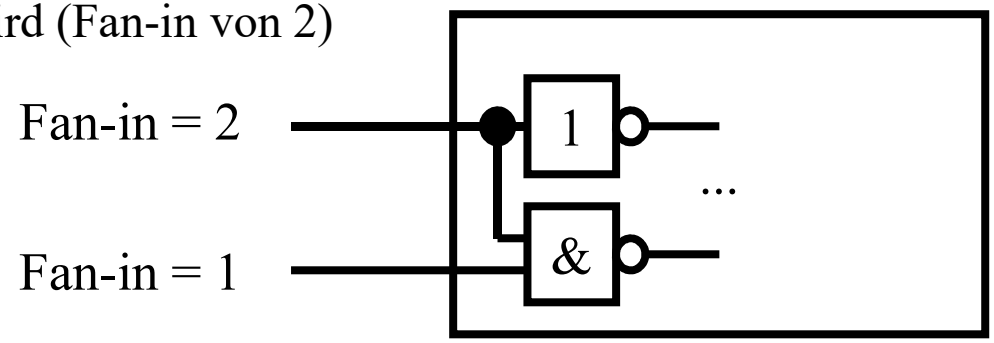
- oft werden von dem Ausgang eines Verknüpfungsgliedes mehrere Eingänge weiterer Verknüpfungsglieder angesteuert
- da die angeschlossenen Eingänge die Pegel verändern (da Strom fließt), können nicht beliebig viele Eingänge angeschlossen werden
- z.B. (über den Daumen)
  - TTL: maximal 10 Eingänge an einen Ausgang (TTL-Schaltungstechnik besprechen wir hier nicht)
  - CMOS: maximal 50 Eingänge an einen Ausgang

- **Fan-out**
  - maximale Anzahl der Standardeingänge, die ein Ausgang treiben kann
    - TTL: Fan-out = 10
    - CMOS: Fan-out = 50



Eingangslastfaktor (Fan-in)

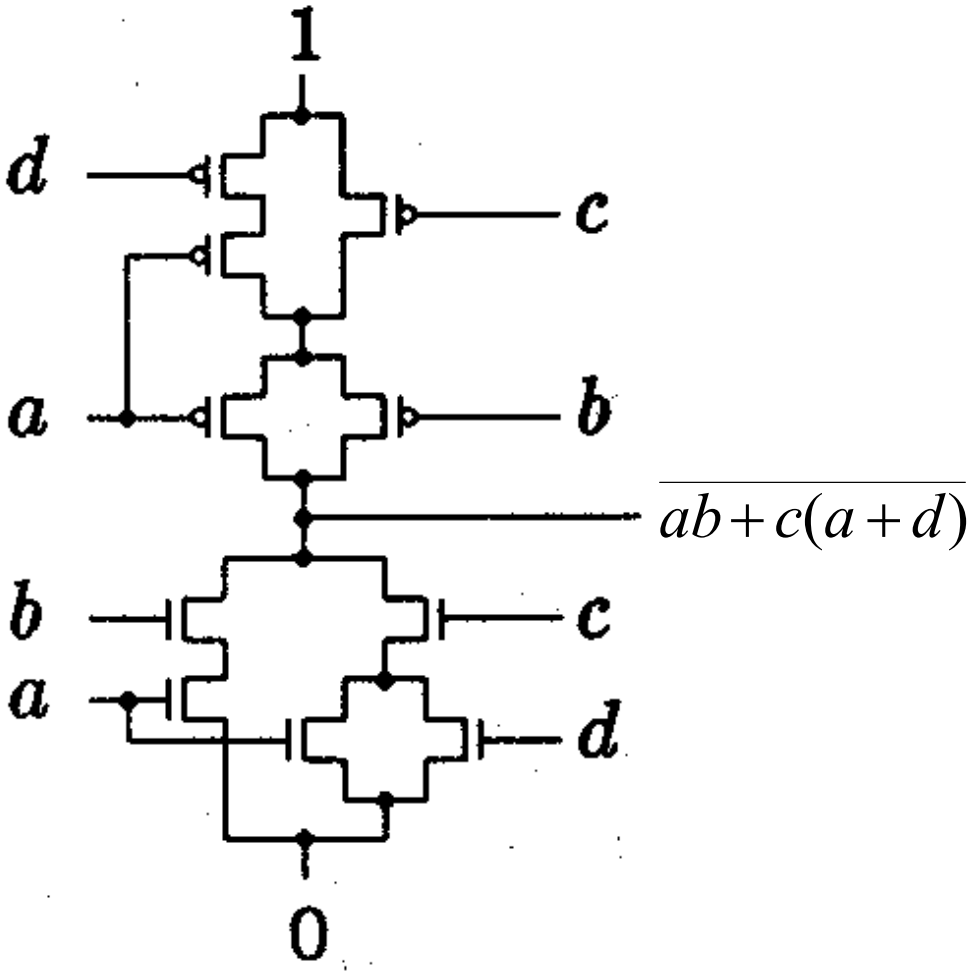
- **Fan-in**
  - einem Standard-Gattereingang, wird der Lastfaktor 1 (Fan-in von 1) zugeordnet
  - bei manchen Schaltungen belastet der Eingang den vorhergehenden Ausgang höher (höheres Fan-in)
    - dies ist z.B. der Fall, wenn ein Eingangssignal intern zur Ansteuerung von zwei Gattern verwendet wird (Fan-in von 2)



  - die Summe der Fan-in-Werte der angeschlossenen Eingänge darf den Fan-out-Wert des treibenden Ausgangs nicht überschreiten
  - dies gilt nur, wenn Verknüpfungsglieder derselben Familie miteinander verschaltet werden

Mischgatter

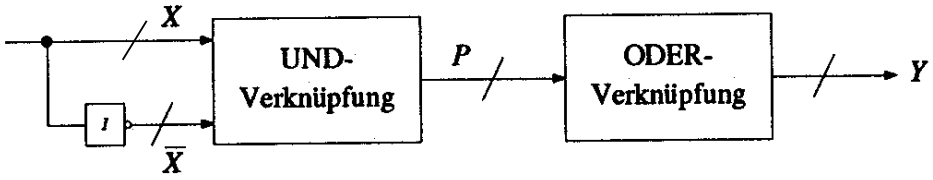
- Boolesche Ausdrücke mit nur einer Negation am Ausgang können auch als Mischgatter dargestellt werden
- Ersparnis von Transistoren
- Addition von Spannungsabfällen
  - die Pegel müssen regeneriert werden
  - daher nur als Teil einer noch größeren Schaltung verwendbar



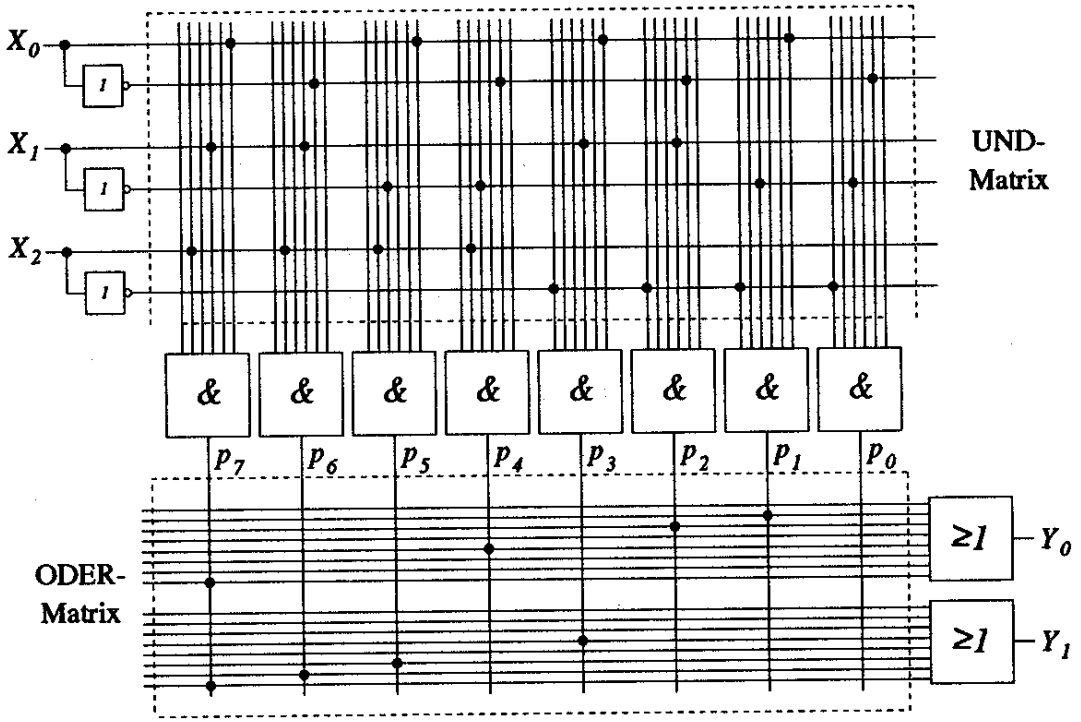
PLA

- **Schaltnetze**
  - nur sehr einfache Schaltnetze werden heute noch aus diskreten Verknüpfungsgliedern zusammengesetzt
  - für komplexe Schaltnetze verwendet man statt dessen flexiblere Lösungen
- **Ausnutzung der DNF oder der minimierten Polynome**
  - Inverter
  - UND-Matrix
  - ODER-Matrix
- **programmierbare Logik-Bausteine**
  - PLA: Programmable Logic Array
  - vorgefertigte Schaltkreise, in denen die UND- und die ODER-Matrix vom Anwender programmiert werden können

PLA (2)



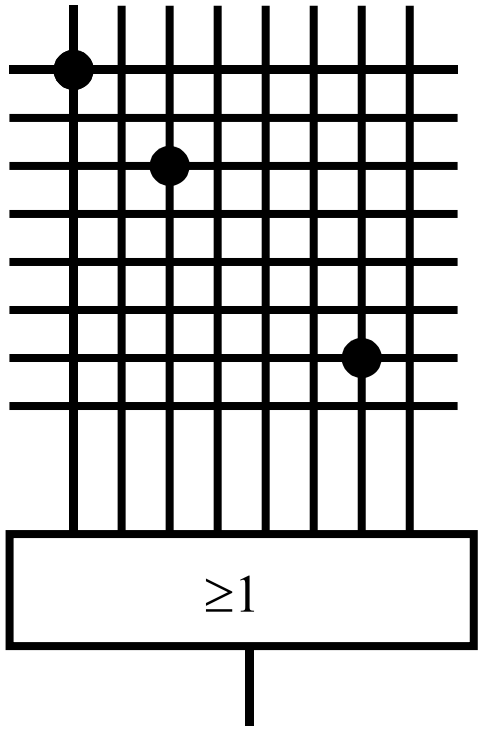
a) Strukturmodell



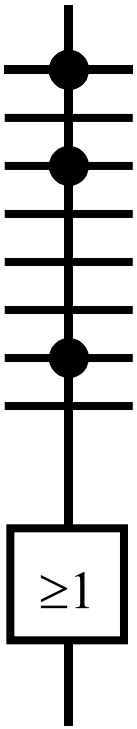
b) Darstellung mit Verknüpfungsgliedern

PLA (3)

Programmierbare ODER-Matrix



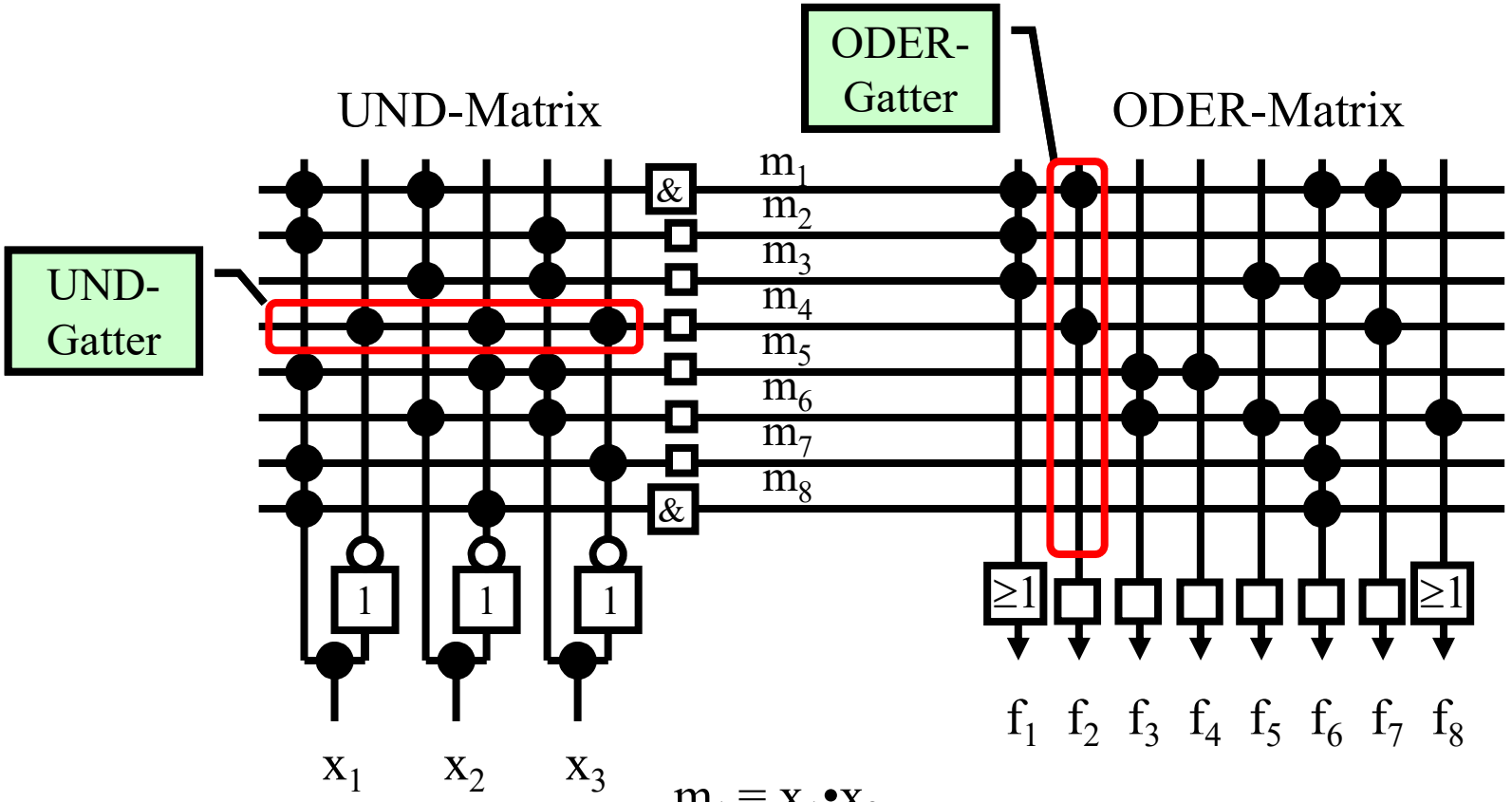
Kurzform



Programmierung:
z.B. Durchbrennen
von Sicherungen

Darstellung der UND-Matrix analog

PLA (4)



$m_1 = x_1 \bullet x_2$
$m_2 = x_1 \bullet x_3$
$m_3 = x_2 \bullet x_3$
$m_4 = x_1' \bullet x_2' \bullet x_3'$
...

$f_1 = m_1 + m_2 + m_3$
$f_2 = m_1 + m_4$
...

PLA (5)

- **Programmierung z.B. durch Schmelzsicherungen**
  - an allen Kreuzungspunkten befinden sich schmale leitende Verbindungen (Schmelzsicherungen)
  - durch gezieltes Anlegen einer hohen Programmierspannung und dem resultierenden Strom kann eine solche Sicherung zerstört werden
  - es verbleiben nur die gewünschten Verbindungen
- **Variante**
  - durch hohe Programmierspannung fließt Strom durch eine normalerweise isolierende Schicht
  - der hohe Strom transportiert Material und stellt eine leitende Verbindung her

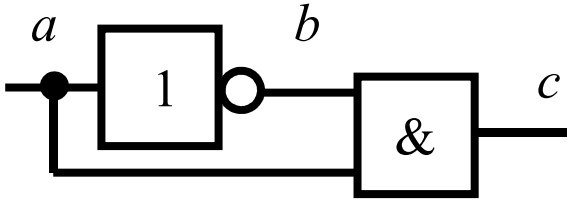
Laufzeiteffekte

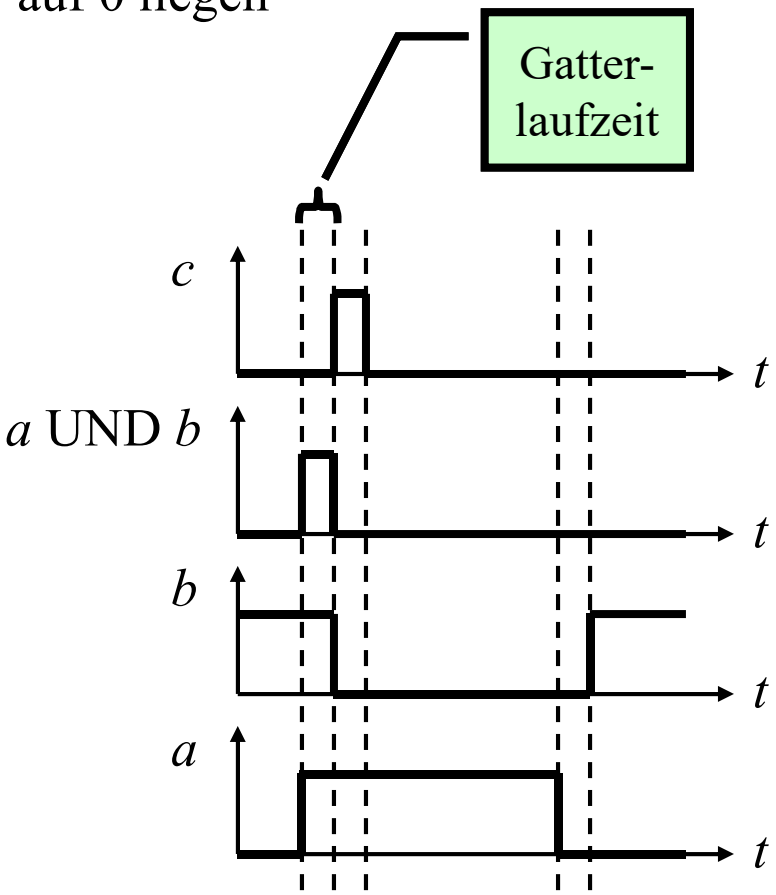
- durch Signallaufzeiten kann der Ausgang eines Schaltnetzes Werte annehmen, die aufgrund der logischen Schaltfunktion eigentlich nicht möglich sind
- solche Falsch-Werte werden Hazards (engl. für Gefahr, Risiko) genannt
- Statischer Hazard
  - Verfälschung von statischen Signalen
- Dynamischer Hazard
  - Verfälschung von Signalflanken (Signalübergängen)

Hazards

- **Beispiel**

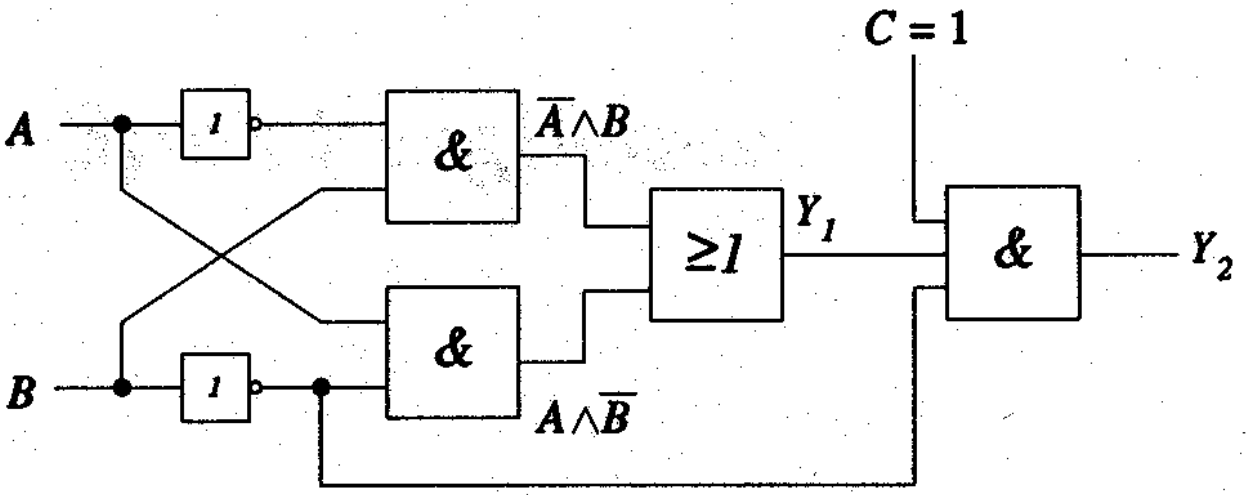
  $c = a \bullet a'$ müsste eigentlich immer auf 0 liegen





Hazards (2)

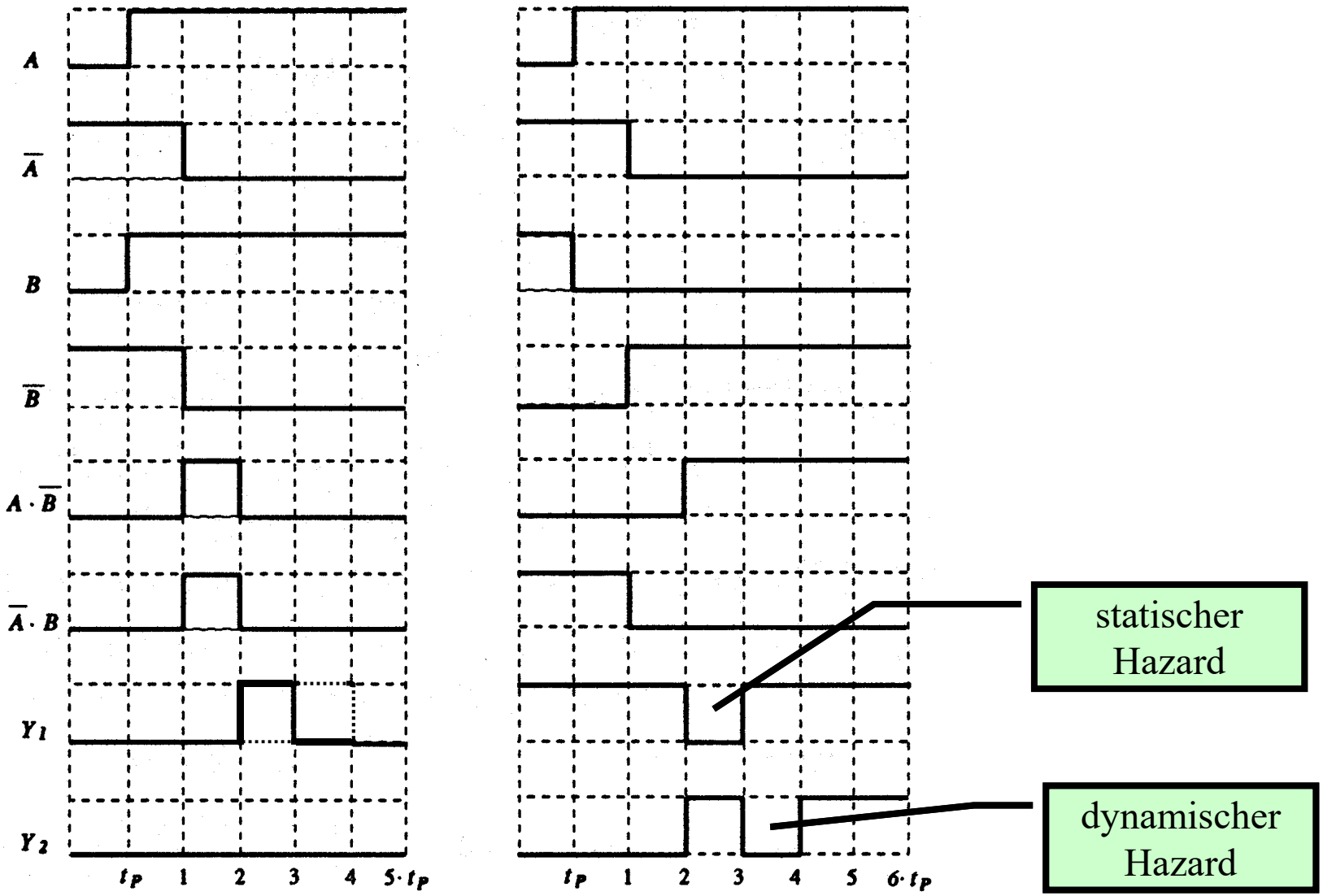
- **Beispiel XOR**



B	A	Y_1	Y_2
0	0	0	0
1	1	0	0
1	0	1	0
0	1	1	1

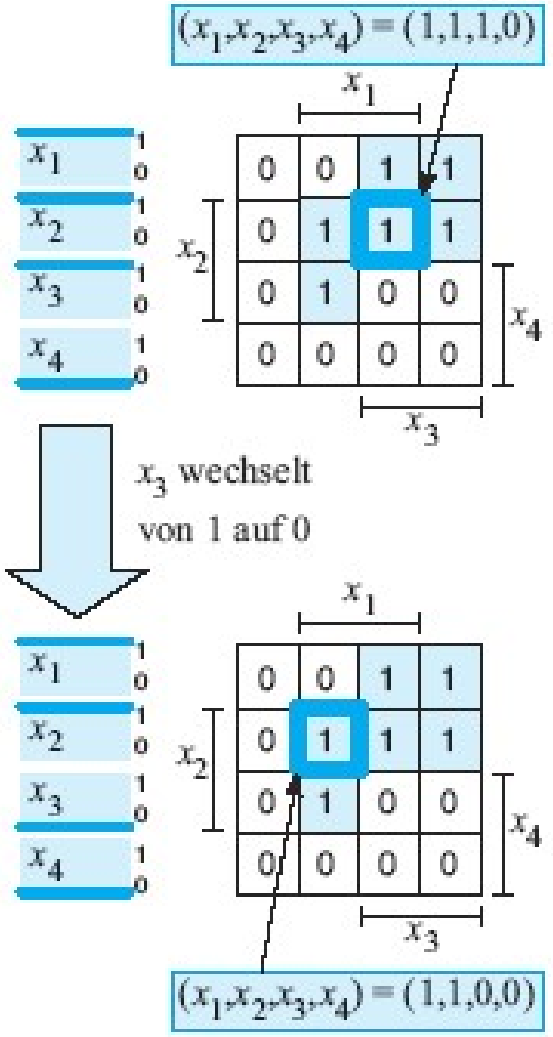
Sehen wir uns diese beiden Übergänge genauer an!

Hazards (3)



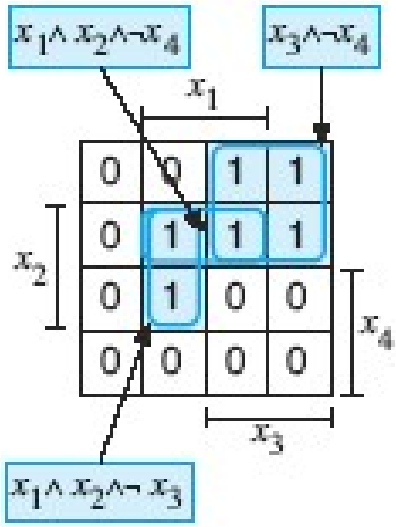
Hazards (4)

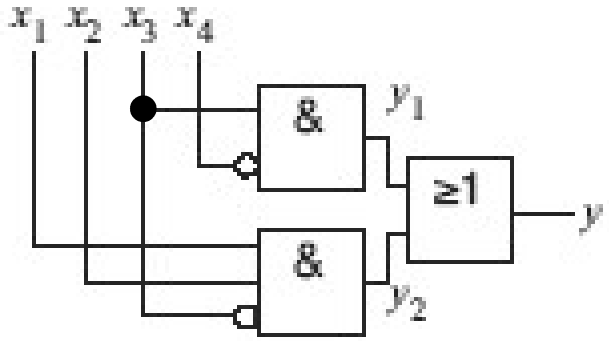
- es gibt Methoden, Schaltungen so auszulegen, dass beim Wechsel ***eines*** Eingangssignals keine statischen Hazards entstehen
  - bleibt man beim Wechsel eines Eingangssignals im selben Resolutionsblock, passiert gar nichts, da das Signal zur Erzeugung der 1 gar nicht benutzt wird
  - ein statischer Hazard kann entstehen, wenn beim Wechsel des Eingangssignals zwei Resolutionsblöcke (Primimplikanten) gewechselt werden, die überlappungsfrei aneinander grenzen
  - denn dann wechselt das UND-Gatter, das zur 1 in der Ausgabe führt
  - durch unterschiedliche Verzögerungen kann zwischenzeitlich eine 0 entstehen



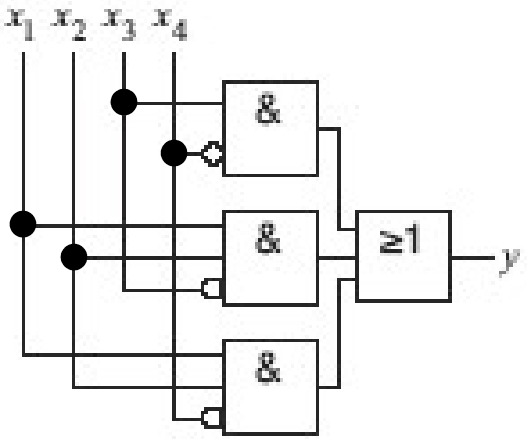
Hazards (5)

- **Hazardfreie Schaltungen**
  - Einfügen eines Resolutionsblocks, der die beiden vorhandenen Blöcke überlappend verbindet
  - Das zusätzliche Monom garantiert eine 1 beim entsprechenden Übergang.
  - Damit können aber nur statische Hazards beseitigt werden, die bei Wechsel ***eines einzigen*** Signals entstehen.





vorher mit Hazard



nachher ohne Hazard

Hazards (6)

- Man muss immer mit dem Auftreten von Hazards in Schaltnetzen rechnen.
- Hazards treten nur für eine gewisse Zeit auf und klingen dann ab.
- Daher werden wir weiter unten meist synchrone Schaltungen entwerfen, die Signale immer nur zu bestimmten aktiven Zeitpunkten auswerten (Taktsignal).
- Hazards sind dann unschädlich, wenn die Zeitpunkte so gewählt werden, dass die Signale unter Garantie stabil geworden sind (die Hazards also abgeklungen sind).

Standardschaltnetze

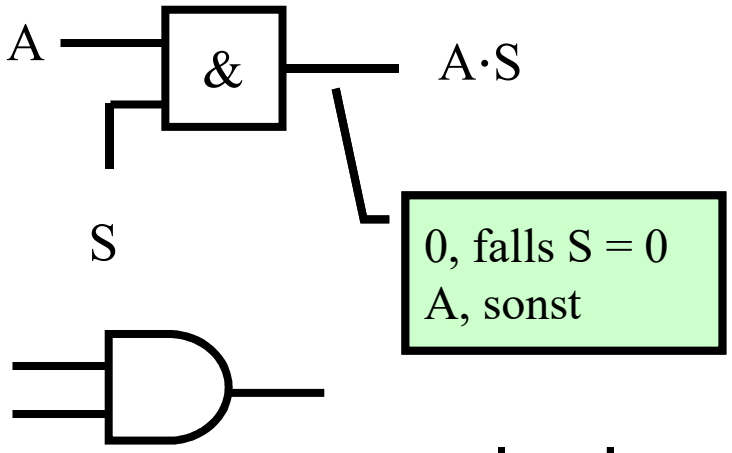
- **Schaltnetze**
  - heißen auch **kombinatorische Schaltungen** (*combinatorial logic*)
  - Ausgänge hängen nur vom momentanen Zustand der Eingänge ab (keine Speicher, im Gegensatz zu den später zu besprechenden Schaltwerken)
  - über Wertetabelle und Minimierung der DNF kann jede beliebige Schaltfunktion als Schaltnetz realisiert werden
  - ein anderer Ansatz ist es, Schaltungen modular aus Standardschaltnetzen aufzubauen
- **Standardschaltnetze sind z.B. (s.u.)**
  - Codeumsetzer (Coder/Decoder)
  - Multiplexer, Demultiplexer, Adressdekodierer
  - Barrelshifter, Addierer, Multiplizierer
  - ALU's, Komparatoren

Grundgatter

Alternative Betrachtungsweise:

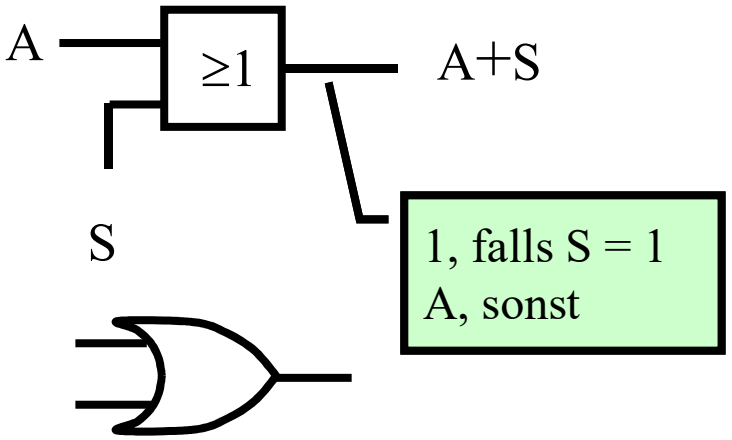
Wie beeinflusst das Steuersignal S das eigentliche Signal A?

UND-Gatter



S	A	A·S
0	0	0
0	1	0
1	0	0
1	1	1

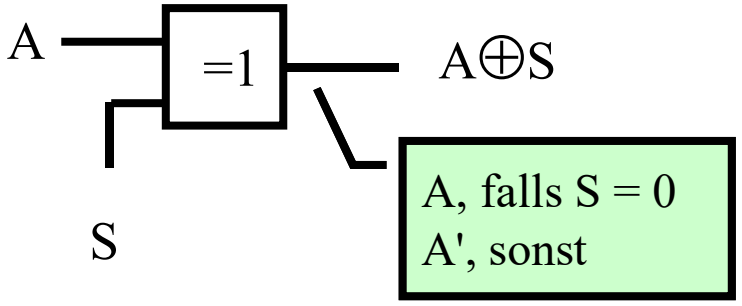
ODER-Gatter

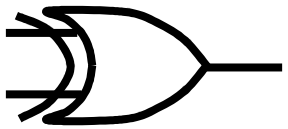


S	A	A+S
0	0	0
0	1	1
1	0	1
1	1	1

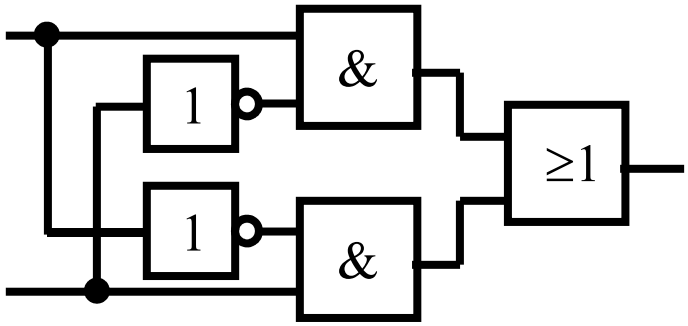
Grundgatter (2)

Exklusiv-ODER-Gatter





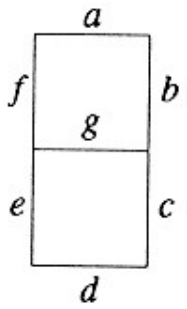
S	A	A⊕S
0	0	0
0	1	1
1	0	1
1	1	0



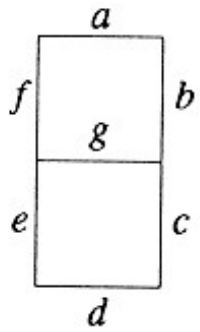
Exklusiv-ODER in DNF

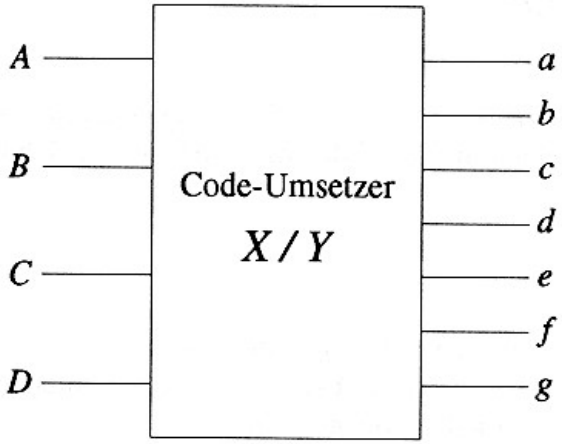
Code-Umsetzer

- **Code-Umsetzer**
  - Schaltfunktionen bilden eine Eingangsbelegung eindeutig auf eine Ausgangsbelegung der Schaltvariablen ab
  - damit kann ein Code in einen anderen umgewandelt werden
- **Beispiel: 7-Segment-Anzeige zur Anzeige der 10 Ziffern**



Beispiel: 7-Segmentanzeige





Dezimal Ziffer	8421–BCD–Code D	C	B	A	7–Segment–Code a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

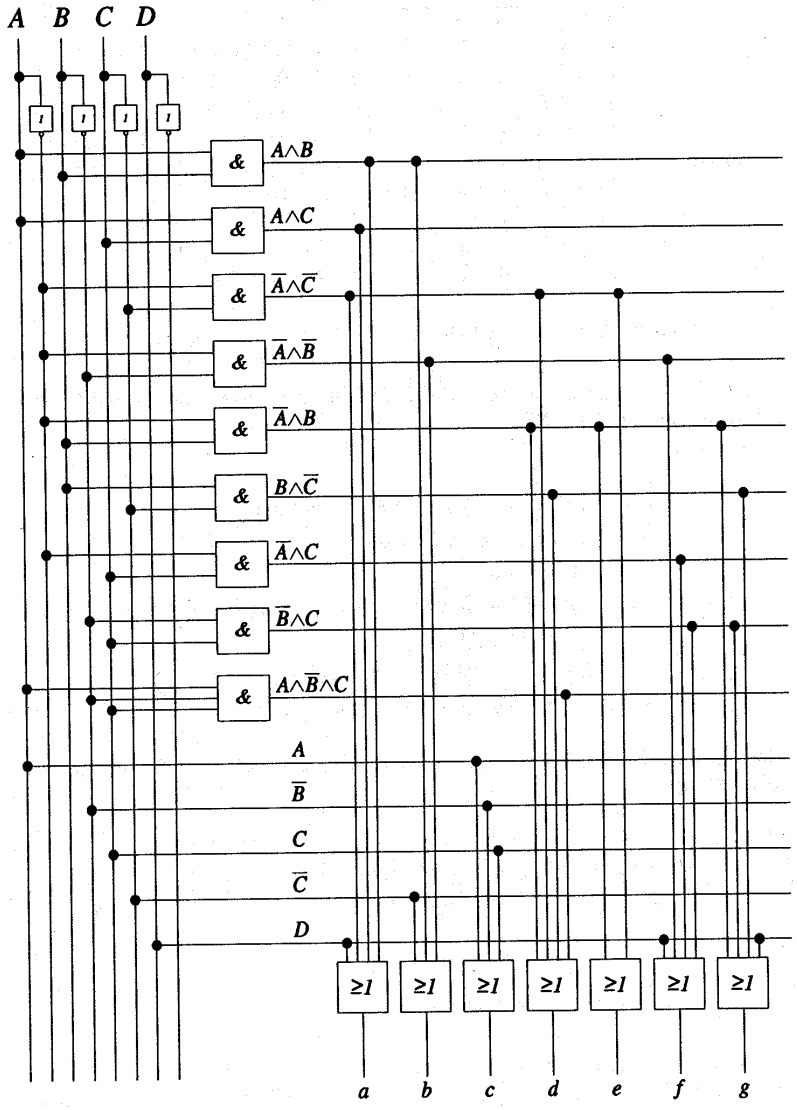
1010 bis 1111 können als don't-cares behandelt werden!

7-Segmentanzeige (2)

– in disjunktiver, minimierter Form

$$
\begin{aligned}
a &= D \vee (\overline{A} \wedge \overline{C}) \vee (A \wedge C) \vee (A \wedge B) \\
b &= \overline{C} \vee (A \wedge B) \vee (\overline{A} \wedge \overline{B}) \\
c &= A \vee \overline{B} \vee C \\
d &= (\overline{A} \wedge B) \vee (\overline{A} \wedge \overline{C}) \vee (B \wedge \overline{C}) \vee (A \wedge \overline{B} \wedge C) \\
e &= (\overline{A} \wedge B) \vee (\overline{A} \wedge \overline{C}) \\
f &= D \vee (\overline{A} \wedge \overline{B}) \vee (\overline{A} \wedge C) \vee (\overline{B} \wedge C) \\
g &= (\overline{A} \wedge B) \vee (\overline{B} \wedge C) \vee (B \wedge \overline{C}) \vee D
\end{aligned}
$$

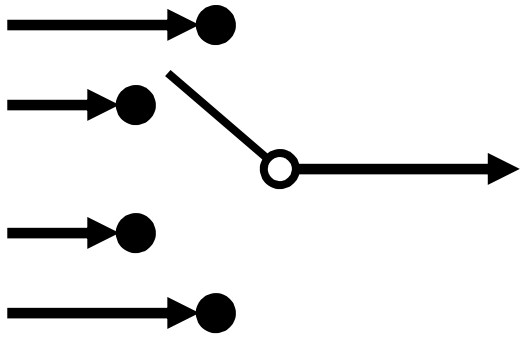
7-Segmentanzeige (3)



- **sieht aus, wie jedes Schaltnetz**
  - drei Schichten
    - Inverter
      - für die Bildung der inversen Schaltvariablen
    - UND-Gatter
      - für die Bildung der Monome
    - ODER-Gatter
      - für die Bildung der Polynome

Multiplexer

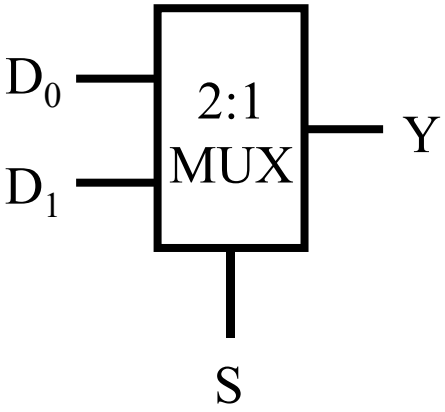
- **Multiplexer (MUX)**
  - n Steuerleitungen, 2^n Eingänge, 1 Ausgang
  - Steuerleitungen legen fest, welcher Eingang auf den Ausgang durchgeschaltet wird
  - wirkt wie ein Drehschalter mit 2^n Stellungen



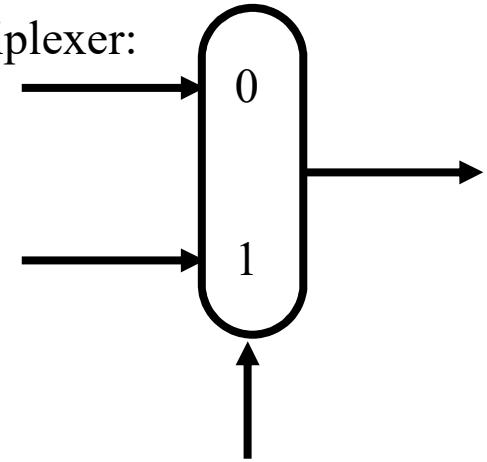
Multiplexer (2)

- **Beispiel 2:1 MUX**

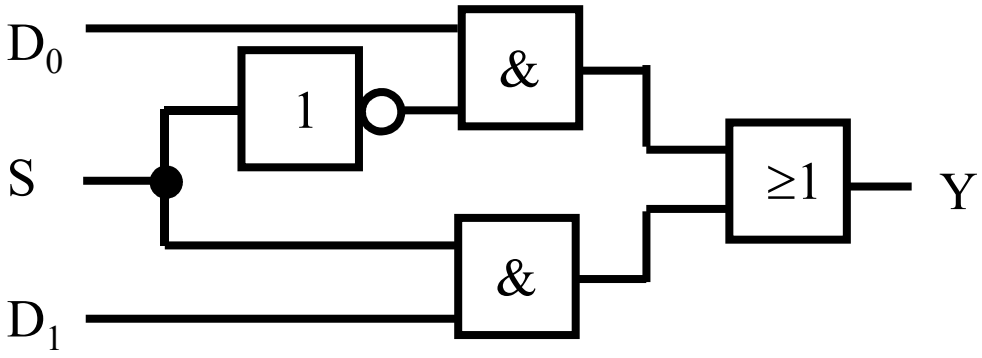
S	Y
0	D_0
1	D_1



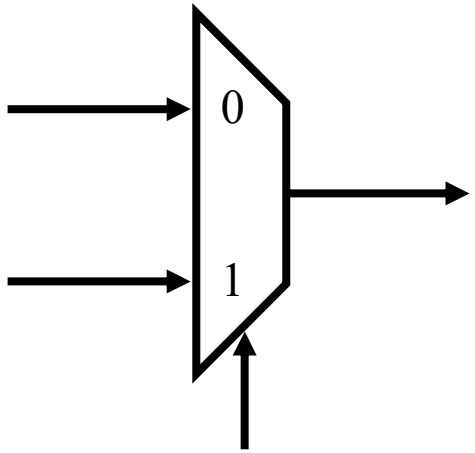
Es gibt viele gebräuchliche Darstellungen für Multiplexer:



Implementierung:



oder so:



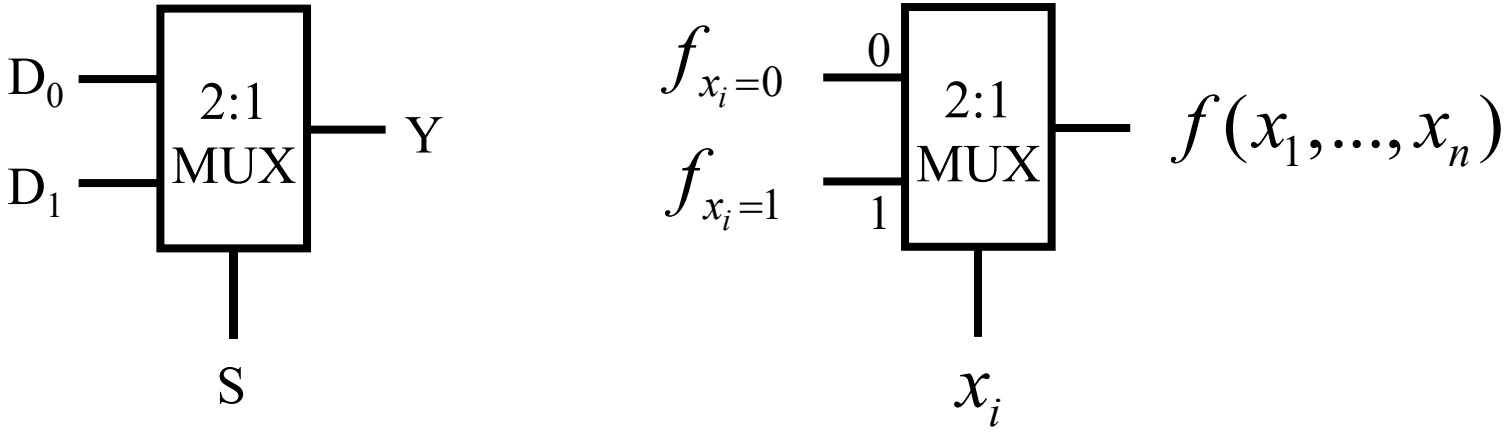
MUX und Entwicklungssatz von Shannon

- **2:1 Multiplexer**

$$Y = SD_1 + \overline{S}D_0$$

- **Entwicklungssatz von Shannon (s.o.)**

$$f(x_1,\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_n) = x_i f_{x_i=1} + \overline{x}_i f_{x_i=0}$$

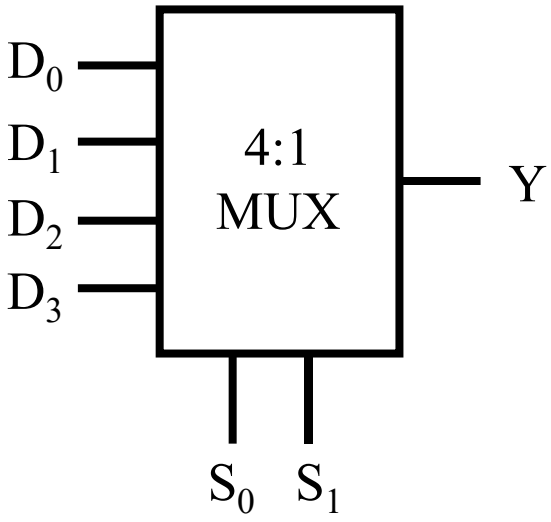


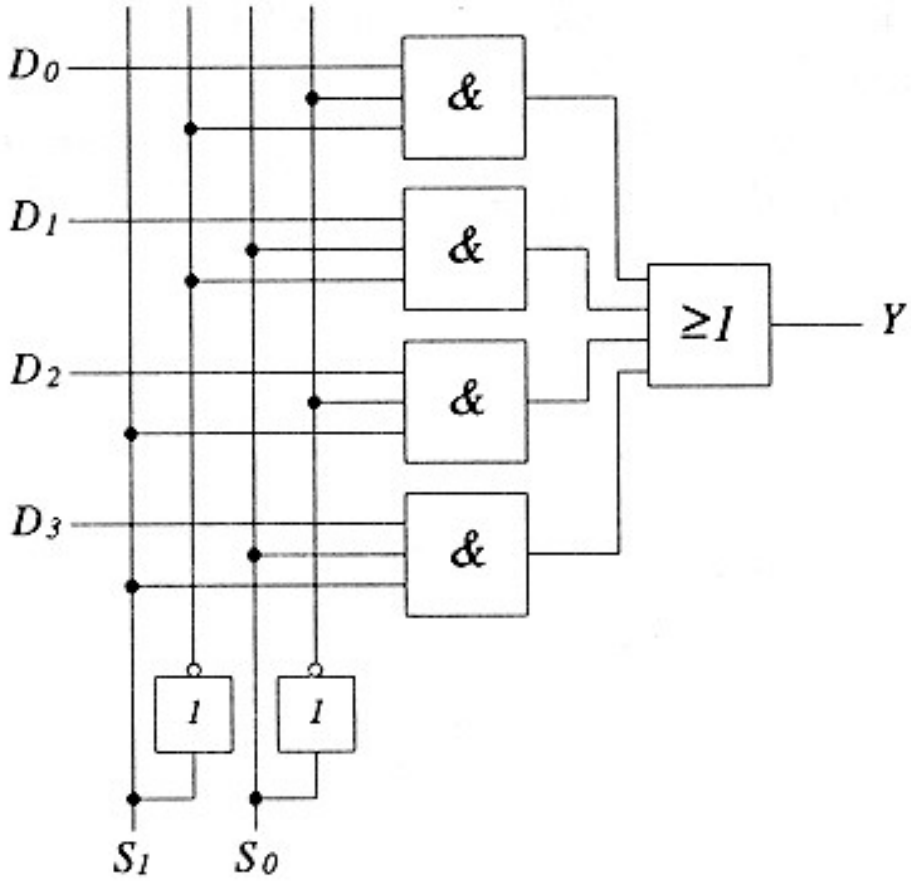
– 2:1 MUX implementiert den Entwicklungssatz (Fallunterscheidung)
– die Kofaktoren müssen an den Eingängen liegen

Multiplexer (3)

- **Beispiel 4:1 Multiplexer**

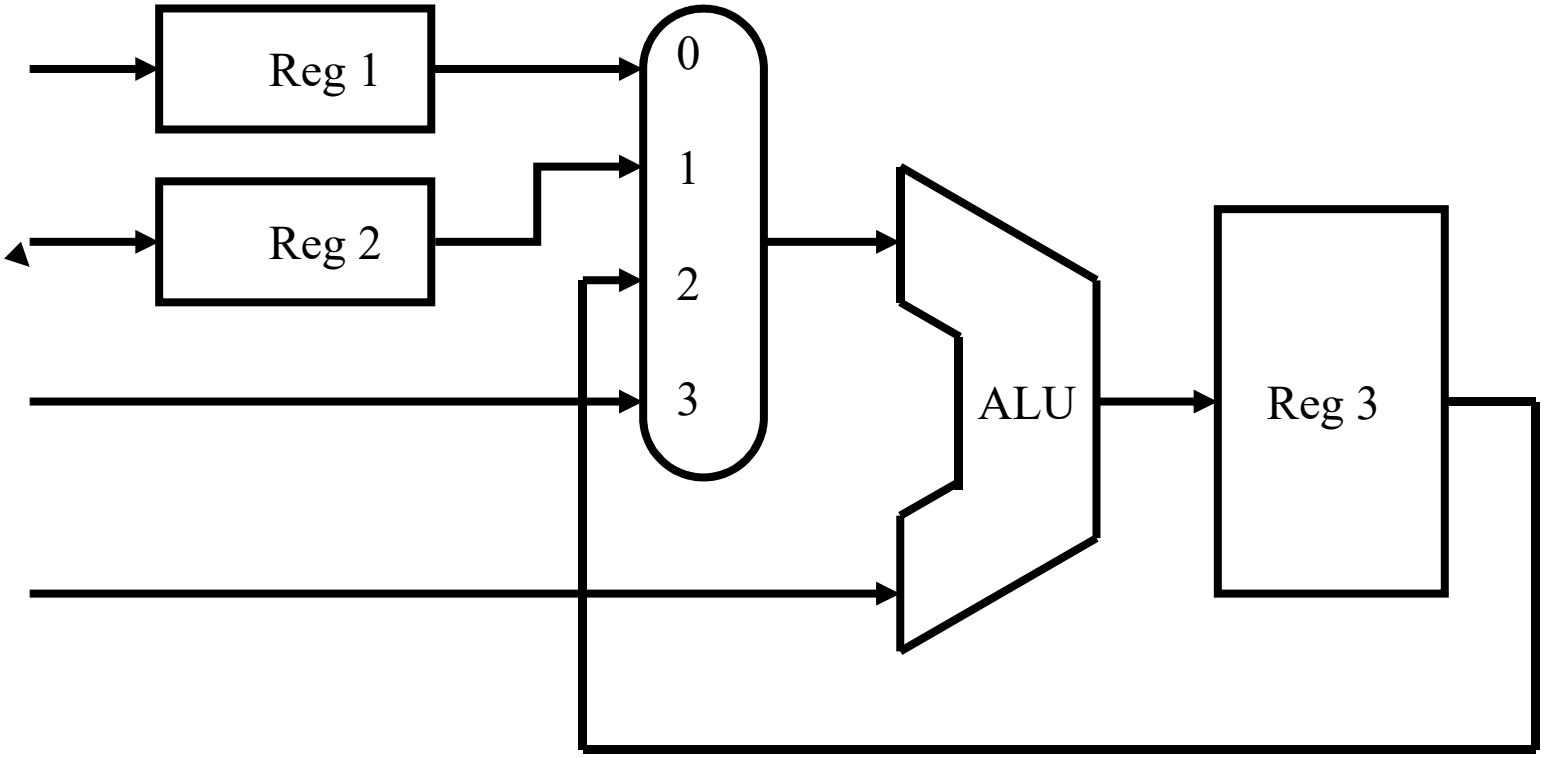
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3





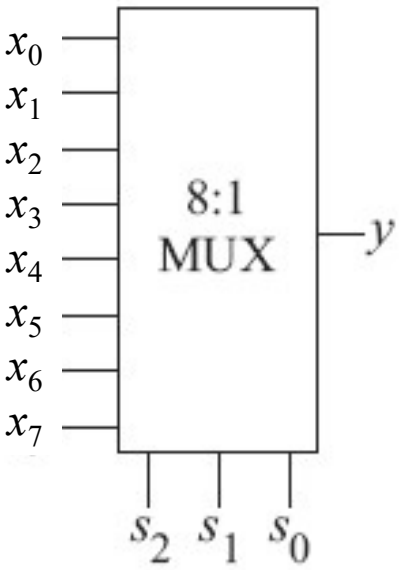
Anwendungen von Multiplexern

- **Daten aus mehreren alternativen Quellen holen**
  - z.B. Steuerung des Datenpfades in Prozessoren

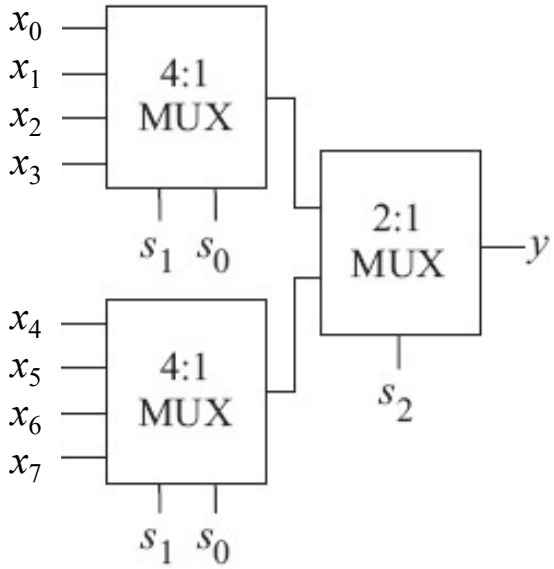


Anwendungen von Multiplexern (2)

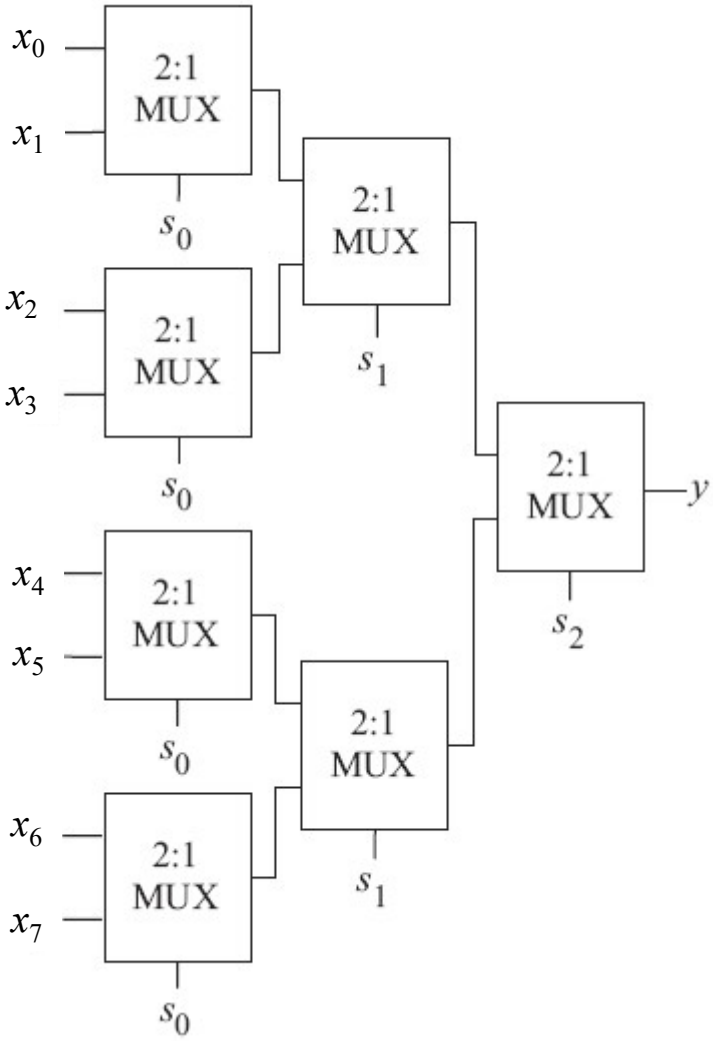
- **Multiplexer kaskadieren**
  - größere Multiplexer können aus kleineren zusammengebaut werden



Verzögerungszeit
3 Gatterlaufzeiten



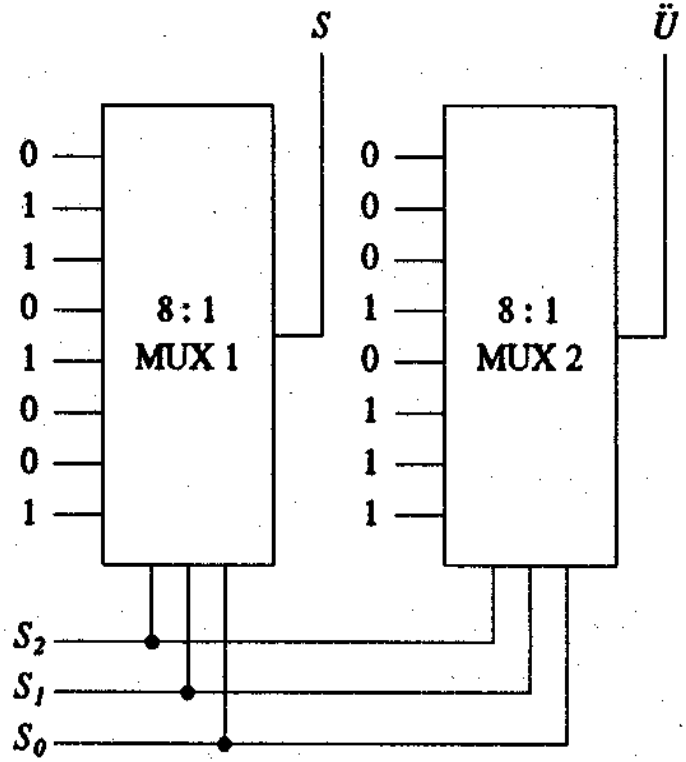
Verzögerungszeit
5 Gatterlaufzeiten



Anwendungen von Multiplexern (3)

- **Realisierung beliebiger Schaltfunktionen**
  - die Steuerleitungen eines Multiplexers adressieren einen Eingang und schalten ihn zum Ausgang durch
  - legt man die Wertetabelle an die Eingänge, erhält man das entsprechende Schaltnetz
    (hier z.B. ein Volladdierer, s.u.)

C_U	B	A	S	$\ddot{U}$
S_2	S_1	S_0	Y_0	Y_1
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



Anwendungen von Multiplexern (4)

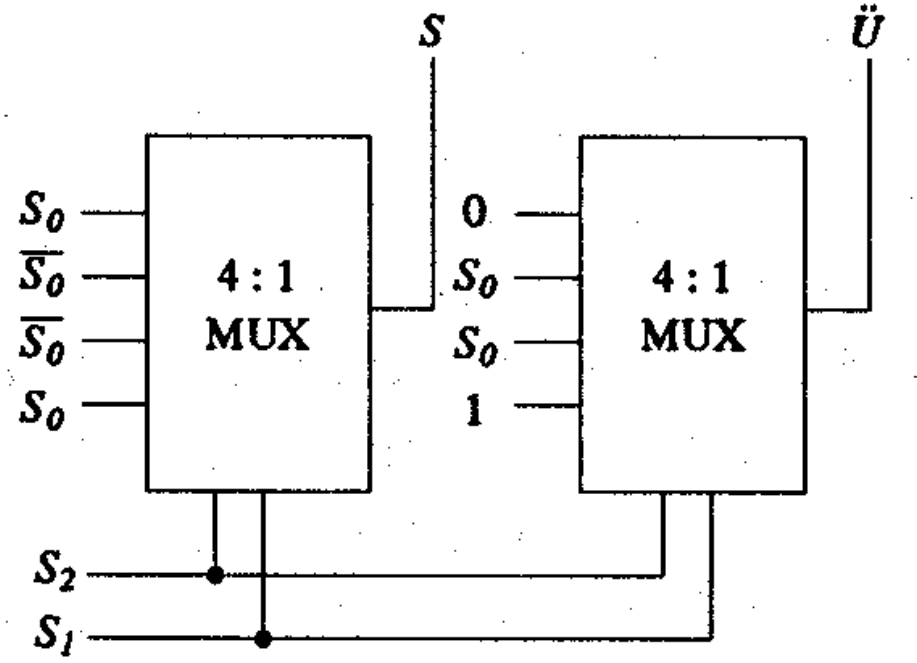
- **man kann immer mit einem Multiplexer auskommen, der eine Steuerleitung weniger hat**
  - eine Schaltvariable, S_0, wird abgespalten
  - jeder Eingang muss damit zwei Zeilen in der Wertetabelle repräsentieren
  - die beiden Zeilen können vier verschiedene Wertekombinationen enthalten (0,0), (0,1), (1,0), (1,1)
  - legt man an den Eingang 0, S_0, S_0' oder 1 an, kann man alle vier Möglichkeiten abdecken

S_0	0	S_0	S_0'	1
0	0	0	1	1
1	0	1	0	1

Anwendungen von Multiplexern (5)

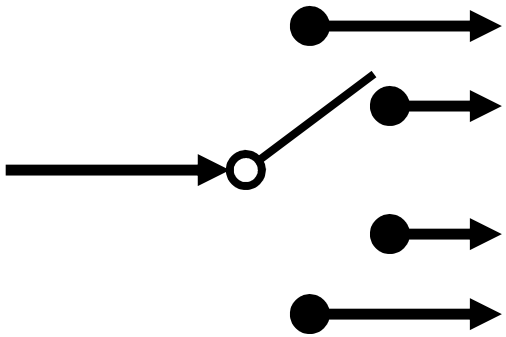
Multiplexer-Belegung für die Dateneingänge

Adresse S_2 S_1	MUX 1 (S)	MUX 2 ($\ddot{U}$)
0 0	S_0	0
0 1	$\overline{S_0}$	S_0
1 0	$\overline{S_0}$	S_0
1 1	S_0	1



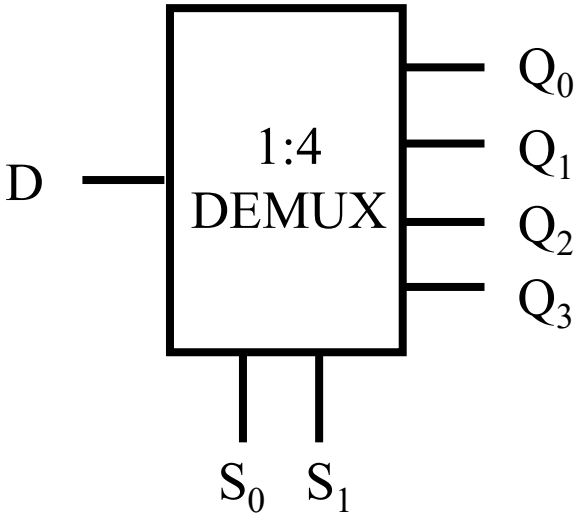
Demultiplexer

- **Demultiplexer (DEMUX)**
  - n Steuerleitungen
  - schaltet den Eingang auf einen von 2^n Ausgängen
  - die anderen Ausgänge führen eine 0
  - wirkt wie ein Verteilerschalter, dessen Stellung durch die n Steuerleitungen festgelegt wird

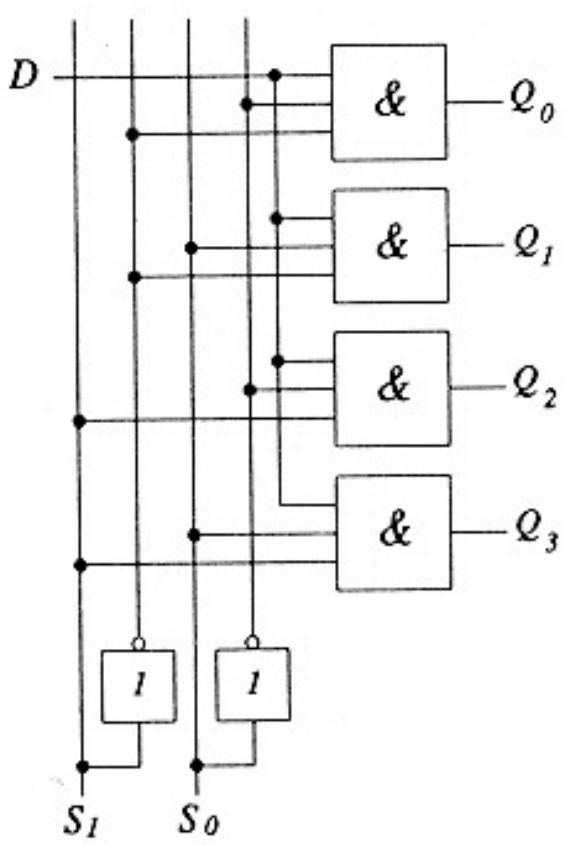


Demultiplexer

- **Beispiel 1:4 Demultiplexer**



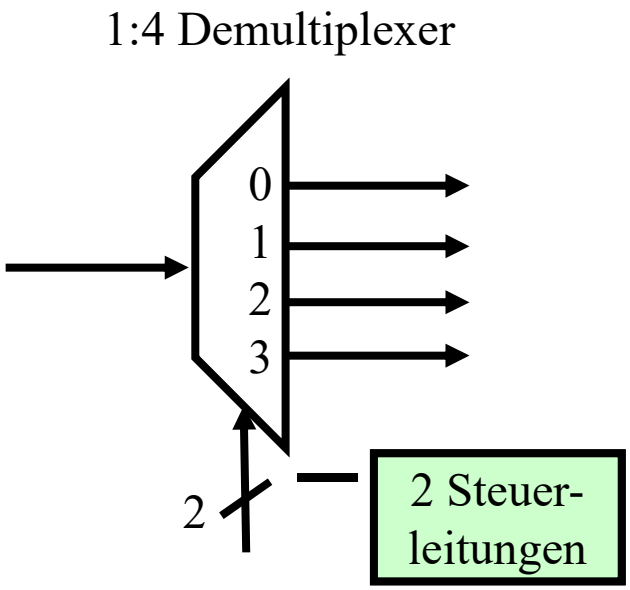
S_1	S_0	Q_0	Q_1	Q_2	Q_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D



Demultiplexer sind wie
Multiplexer ebenfalls kaskadierbar

Demultiplexer (2)

- **Anderes Schaltzeichen**

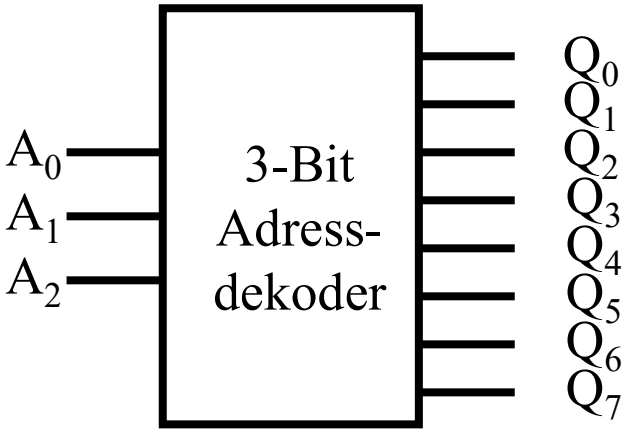


Dekoder

- **Dekoder**
  - auch Adressdekoder genannt
  - n Eingänge
    - eine n-Bit Zahl
  - 2^n Ausgänge
    - eine von 2^n Leitungen liegt auf 1, die anderen auf 0
      - 1 aus n Code (s.o.)

Dekoder (2)

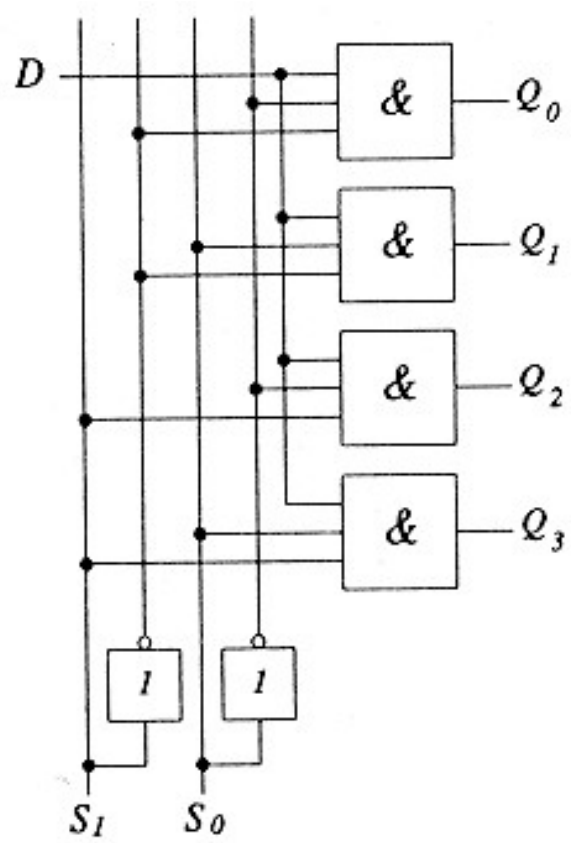
- **Beispiel: 3-Bit Adressdekoder**



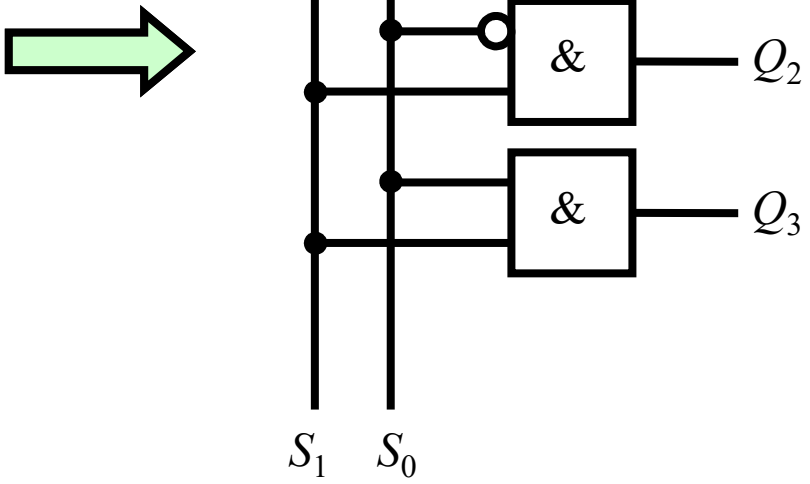
A_2	A_1	A_0	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Dekoder (3)

- **Aufbau**
  - entsteht aus Demultiplexer mit einer 1 am Eingang *D*



Demultiplexer



Adress-Dekoder

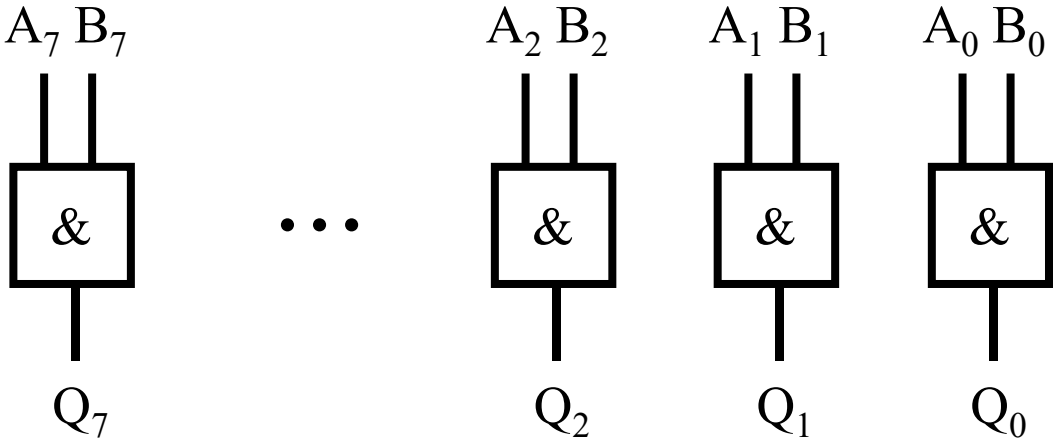
Arithmetische Schaltnetze

- **Arithmetische Schaltungen**
  - Computer führen einfache arithmetische Rechenoperationen mithilfe von Schaltnetzen aus
  - logische Operationen
  - Shifter
  - Addierer
  - Multiplizierer

Logische Operationen

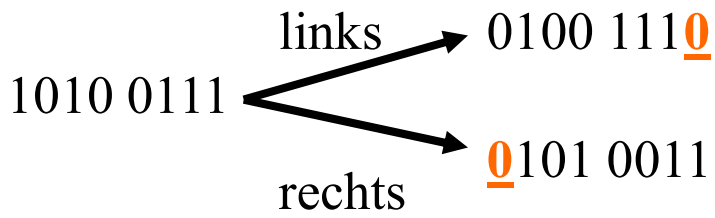
- **Logische Operationen**
  - werden von allen Computern durchgeführt
  - meist gibt es: UND, ODER, Exklusiv-ODER für ganze Worte (bitweise verknüpft)
  - Realisierung mit entsprechenden Grundgattern

z.B. 8-Bit UND-Verknüpfung

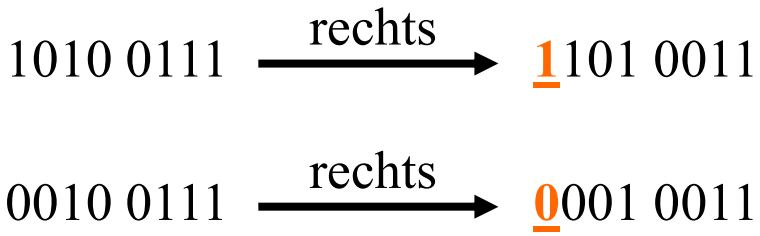


Shifter

- **Shifter**
  - verschieben Bitmuster nach links oder rechts
  - logischer Shift
    - Nullen werden hineingeschoben

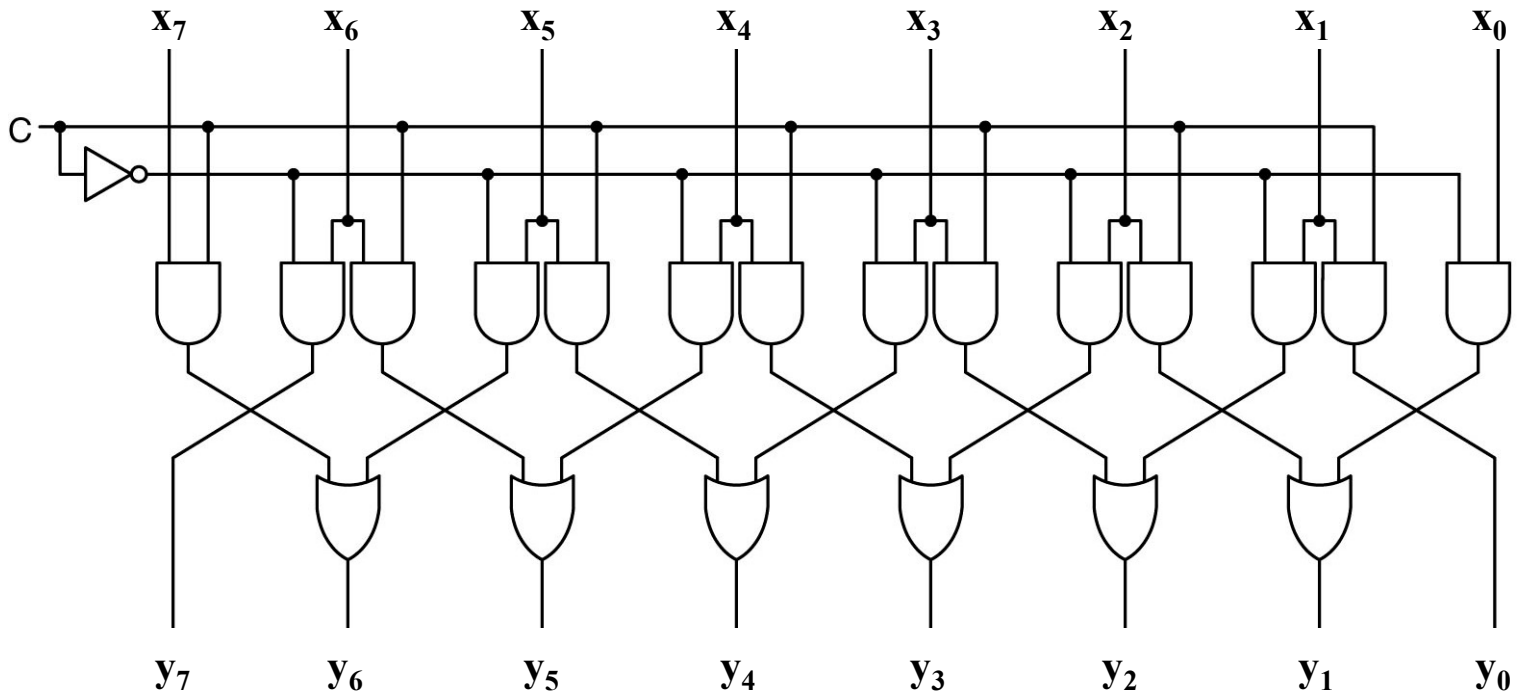


  - arithmetischer Shift
    - beim "Rechtsschieben" wird Vorzeichen beibehalten



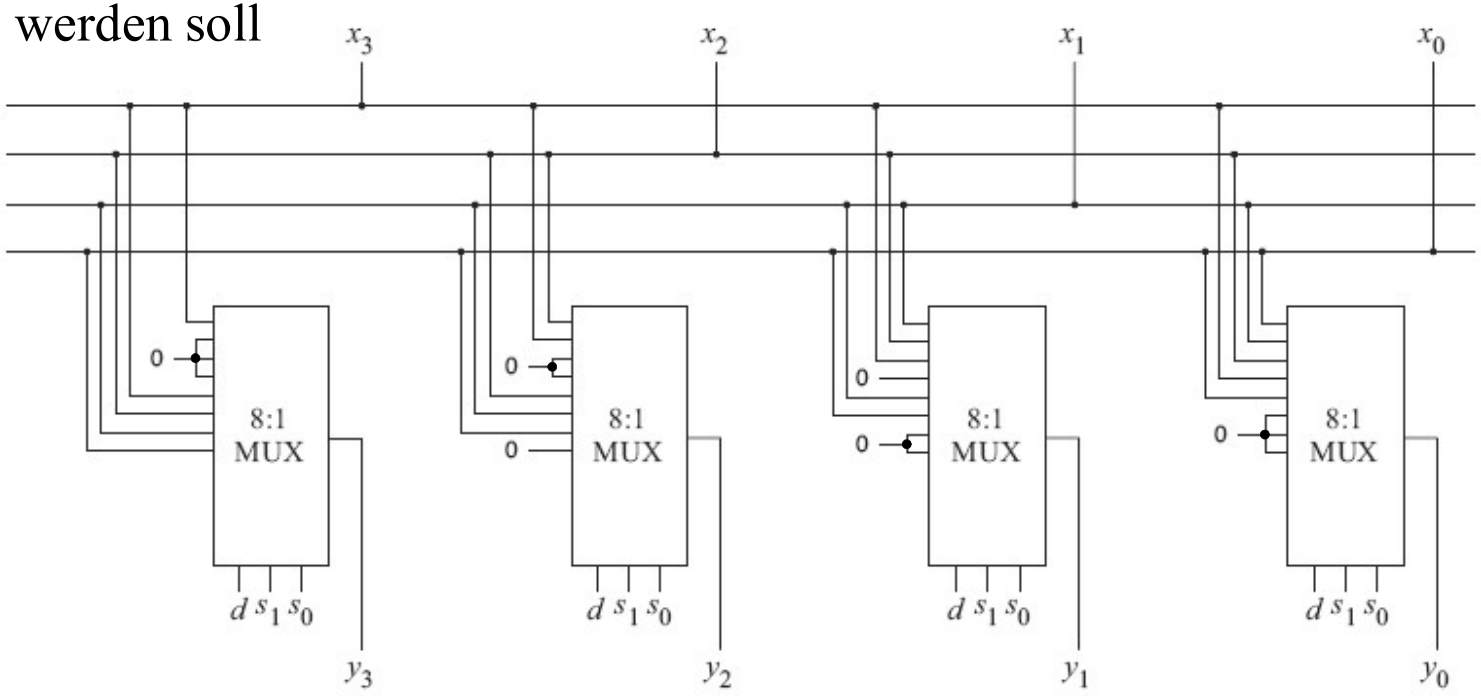
Shifter (2)

- **8-Bit rechts/links Shifter (logischer Shift)**
  - C bestimmt Schieberichtung
    - C=1: rechts, C=0: links
  - zusätzliche Logik für arithmetischen Shift notwendig



Barrel-Shifter

- **Einsatzgebiet**
  - wenn ein Bitvektor effizient um beliebig viele Stellen verschoben werden soll



Rechts-Shift:

d	s_1	s_0	y_3	y_2	y_1	y_0
0	0	0	x_3	x_2	x_1	x_0
0	0	1	0	x_3	x_2	x_1
0	1	0	0	0	x_3	x_2
0	1	1	0	0	0	x_3

Links-Shift:

d	s_1	s_0	y_3	y_2	y_1	y_0
1	0	0	x_3	x_2	x_1	x_0
1	0	1	x_2	x_1	x_0	0
1	1	0	x_1	x_0	0	0
1	1	1	x_0	0	0	0

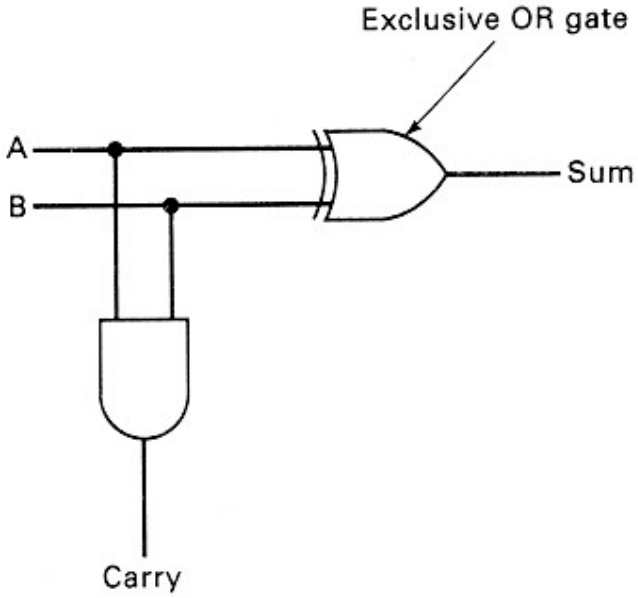
Barrel-Shifter (2)

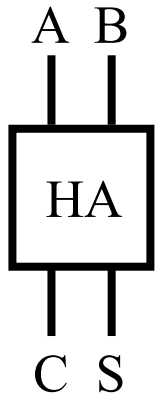
- **Implementierung**
  - sehr elegant mit Multiplexern darstellbar
  - kann um beliebige Funktionalität erweitert werden
    - nur größerer Multiplexer notwendig
    - z.B. arithmetischer Shift: statt 0 wird Vorzeichen reingeschoben
    - z.B. Rotation: ein hinausgeschobenes Bit wird auf der anderen Seite wieder reingeschoben
  - ist aber auch sehr aufwendig
    - Barrel-Shifter sind relativ teuer in der Implementierung
      - 64-Bit Worte erfordern z.B. 64 Multiplexer mit 128 Eingängen, wenn jeder mögliche Shiftwert (rechts und links) realisiert werden soll

Halbaddierer

- Addition zweier 1-Bit Zahlen
- genügt z.B., um das niederwertigste Bit zweier *n*-Bit Zahlen zu addieren

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



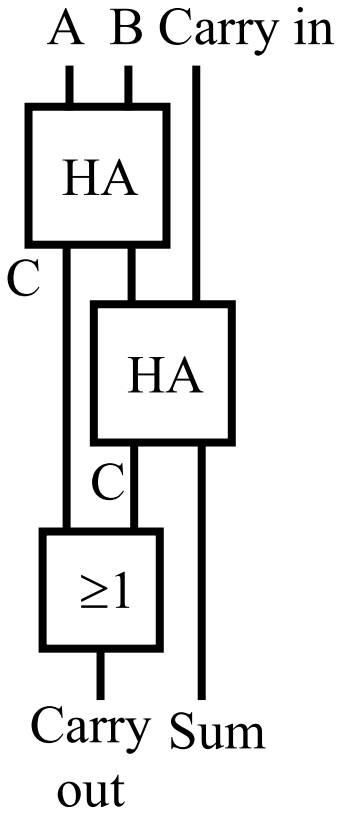


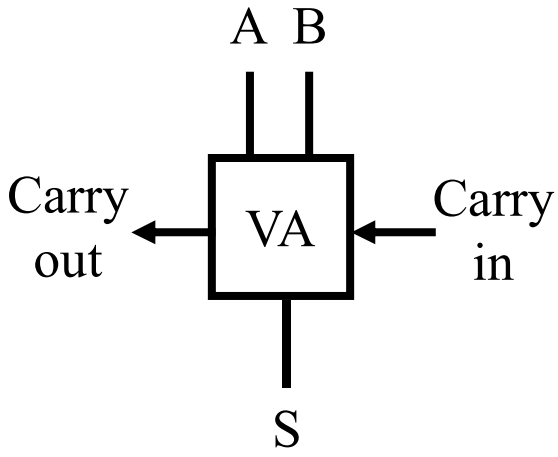
Schaltzeichen

Volladdierer

– um den Übertrag (*carry*) zu verarbeiten, muss man 3 Bit addieren

A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1





Schaltzeichen

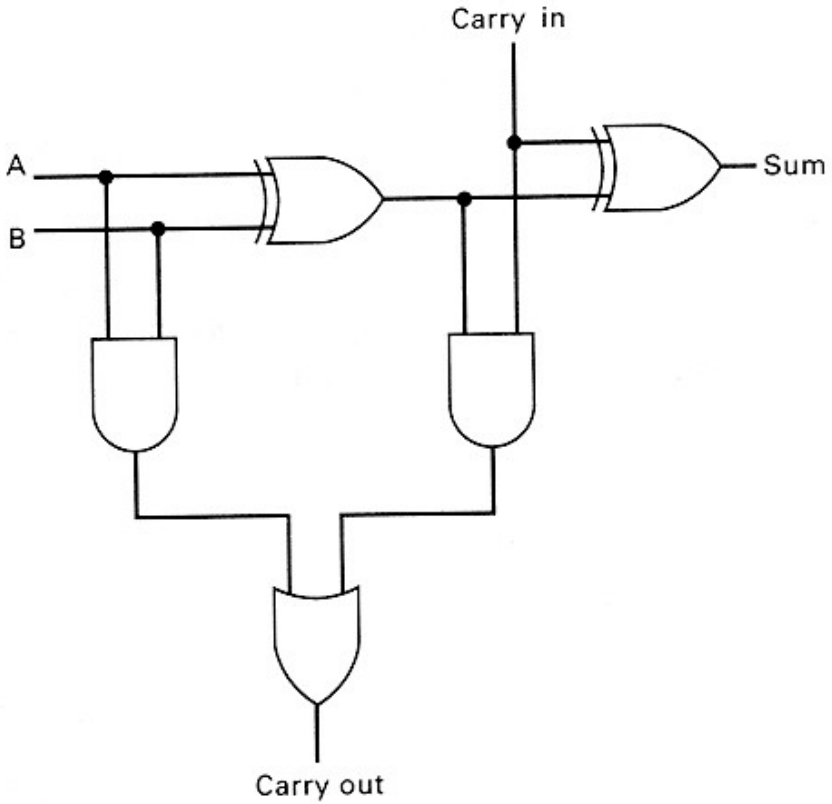
Volladdierer (2)

- **Volladdierer**

A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

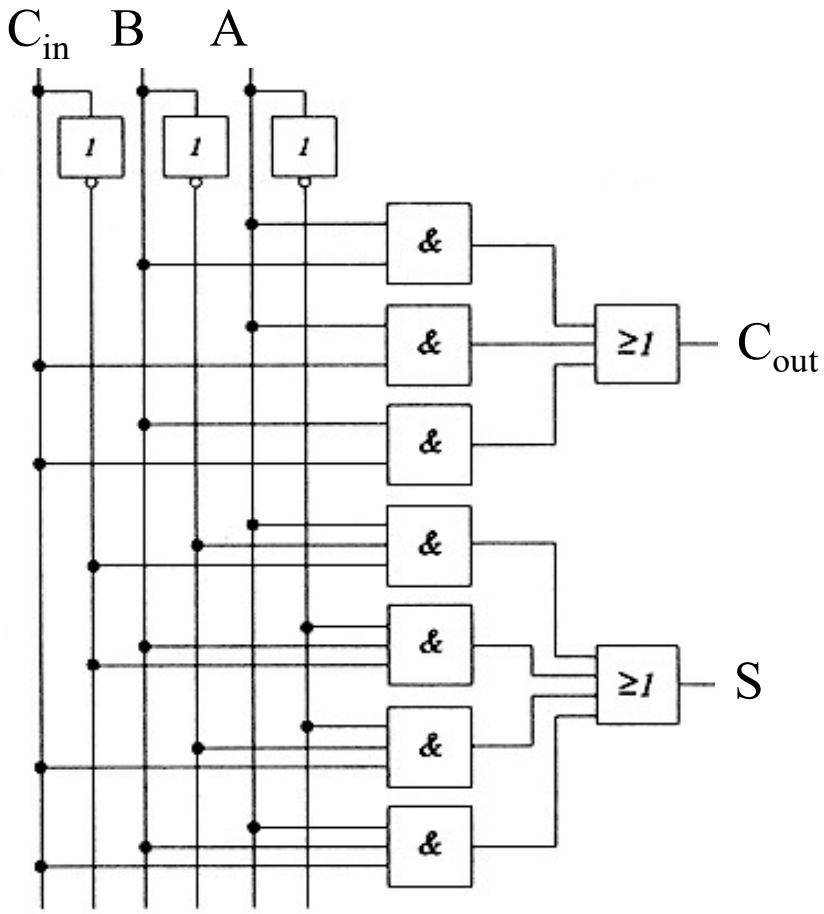
Parität

Majorität



Volladdierer (3)

- **als Disjunktion von Monomen**



Einschub: O-Notation

- beschreibt das asymptotische Verhalten einer Funktion f(n)
- d.h. ihr wesentliches Verhalten für große n

- **genauer**
  - Die Funktion f(n) fällt in die Komplexitätsklasse O(g(n)), wenn es Konstanten c und N gibt, so dass für alle $n \geq N$ gilt:

$$|f(n)| \leq c\,|g(n)|$$

    d.h. f(n) wächst nicht schneller als g(n)

- **Beispiele**

  $f(n) = 5n^2 + 100n + 1000$ fällt in $O(n^2)$

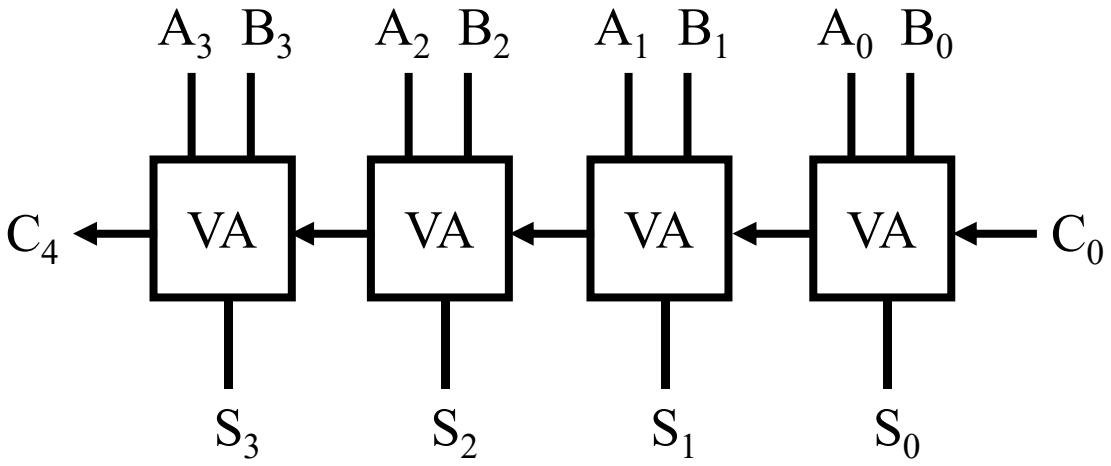
  denn es gilt (z.B.) für $n > 1000000$: $f(n) \leq 1000n^2$

  oder

  $f(n) = 2^n + 1000000\, n^{42}$ fällt in $O(2^n)$

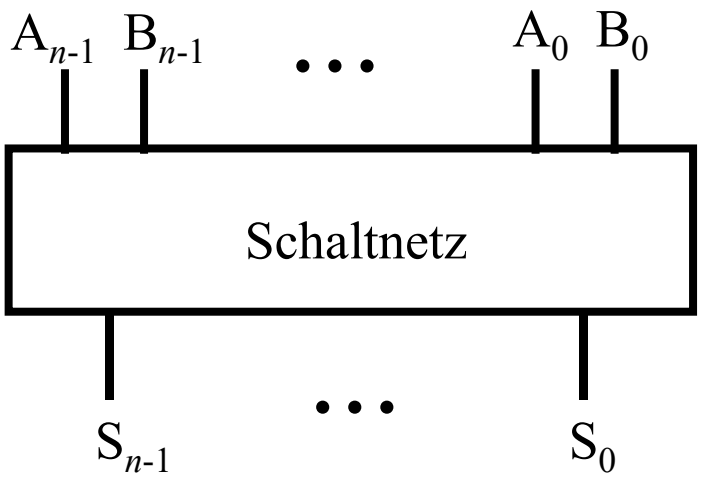
Ripple-Carry Addierer

- **Addition von *n*-Bit Zahlen**
  - Schulmethode: mit dem niederwertigsten Bit anfangen und Überträge zur nächsten Stelle hinzuaddieren
  - Carry-Bit "plätschert" (*ripple*) durch alle Stellen
  - Verzögerung linear in n, also $O(n)$
  - Hardwareaufwand ist ebenfalls $O(n)$



Paralleladdierer

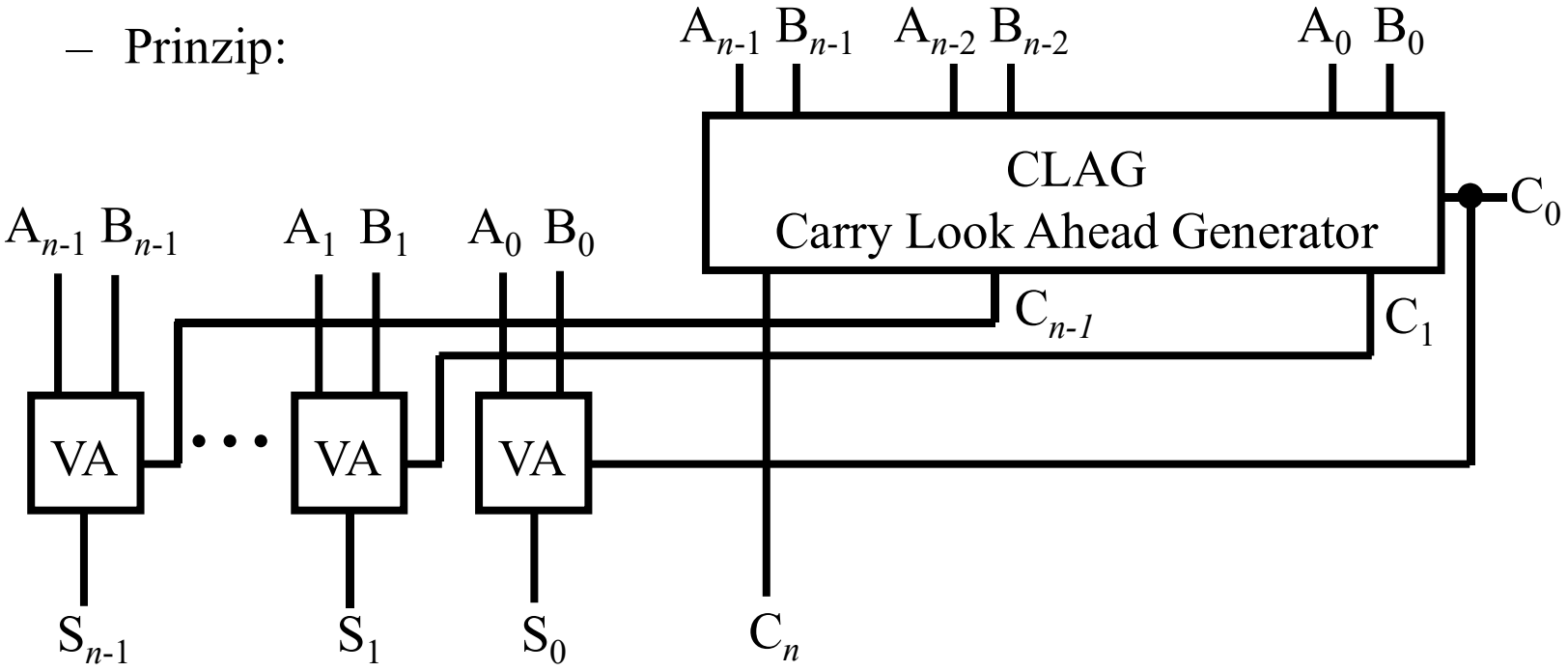
- **Paralleladdierer**
  - Schaltnetz in DNF, das alle n Bits parallel berechnet
  - Verzögerungszeit
    - 3 Gatterlaufzeiten (NICHT, UND, ODER)
    - also konstante Verzögerungszeit oder O(1)
  - viel zu aufwendig für n Bit, da ca. $n \cdot 2^{2n-1}$ Minterme vorhanden sind
    - Hardwareaufwand $O(n \cdot 2^{2n})$



$2n$ Eingänge
2^{2n} Zeilen in Wahrheitstabelle, ca. die Hälfte enthält eine 1
n Ausgänge

Carry-Look-Ahead Addierer

- **Kompromiss**
  - die Carry-Bits werden ***parallel*** berechnet (das geht elegant, s.u.)
    - Schaltfunktion lässt sich stark vereinfachen (s.u.)
  - anschließend wird in jeder Stelle das Carry-Bit, A und B ***parallel*** mit dem Volladdierer addiert
  - Prinzip:



Carry-Look-Ahead Addierer (2)

- **CLAG**
  - Übertrag entspricht Majorität der drei Eingangsvariablen A_n, B_n, C_n
  - Erinnerung: Majoritätsfunktion (siehe vereinfachtes Schaltnetz 3 für Majorität)

$$f(x_1, x_2, x_3) = x_1x_2 + x_1x_3 + x_2x_3$$
$$= x_1(x_2 + x_3) + x_2x_3$$

  - hier

$$C_{n+1} = A_n B_n + (A_n + B_n)C_n$$

  - definiere

$$g_n = A_n B_n \text{ und } p_n = A_n + B_n$$

  - dann gilt

$$C_{n+1} = g_n + p_n C_n$$

Carry-Look-Ahead Addierer (3)

- Bedeutung von g_n und p_n:

  g_n ein Übertrag wird in einer Stelle neu erzeugt, wenn A_n UND B_n gleich 1 sind (*carry generate*)

  p_n ein Übertrag C_n wird in die nächste Stelle weitergeleitet, wenn A_n ODER B_n gleich 1 sind (*carry propagate*)

- es gilt daher

$$C_1 = g_0 + p_0C_0$$
$$C_2 = g_1 + p_1C_1 = g_1 + p_1(g_0 + p_0C_0) = g_1 + p_1g_0 + p_1p_0C_0$$
$$C_3 = g_2 + p_2C_2 = \ldots = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0C_0$$
$$C_4 = g_3 + p_3C_3 = \ldots = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0C_0$$

- Das Ausmultiplizieren (Distributivgesetz) ist hier wichtig! Warum?

Carry-Look-Ahead Addierer (4)

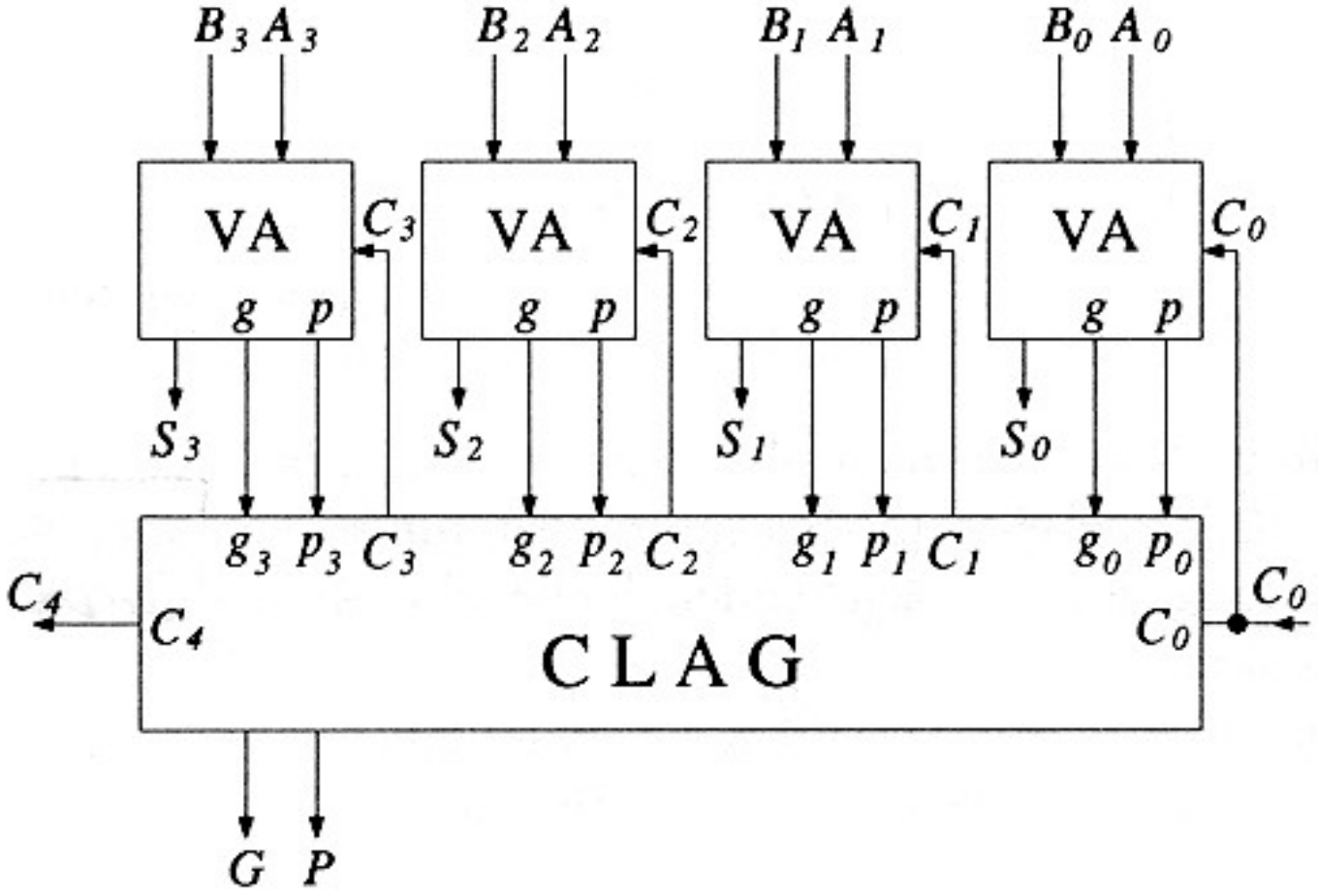
- Schaltfunktionen für CLAG sind also sehr einfach strukturiert, wenn statt A und B, g und p benutzt werden
- ein Volladdierer kann g und p leicht mitberechnen
- Verzögerungszeit ist konstant (unabhängig von n), also O(1)
- Hardwareaufwand
  - Anzahl der Gatter ist $O(n^2)$
    - n Ausgänge
    - im Mittel werden $n/2$ Gatter pro Ausgang benötigt
  - Anzahl der Transistoren in CMOS ist $O(n^3)$
    - der Ausgang i besitzt i UND-Gatter mit 1 bis i Eingängen
    - Anzahl der Eingänge insgesamt und damit die Anzahl der Transistoren (jeder Eingang hat 2 Transistoren in CMOS) ist also

$$\sum_{i=1}^{n}\sum_{j=1}^{i} j = \sum_{i=1}^{n} O(i^2) = O(n^3)$$

    - (wer es nicht glaubt, sollte mal genau nachrechnen!)

Carry-Look-Ahead Addierer (5)

- **Beispiel 4-Bit Addierer mit CLAG**



Carry-Look-Ahead Addierer (6)

- das Prinzip kann hierarchisch auf 4-Bit Blockebene weitergeführt werden, da man für C_4 schreiben kann

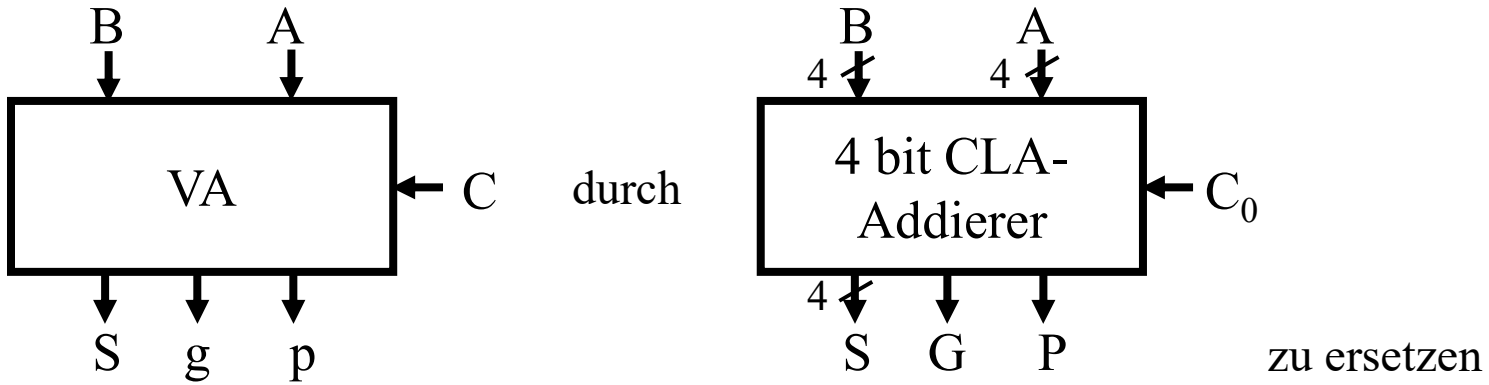
$$C_4 = G + PC_0$$

- mit

$$G = g_3 + p_3g_2 + p_3p_2\,g_1 + p_3p_2\,p_1\,g_0$$

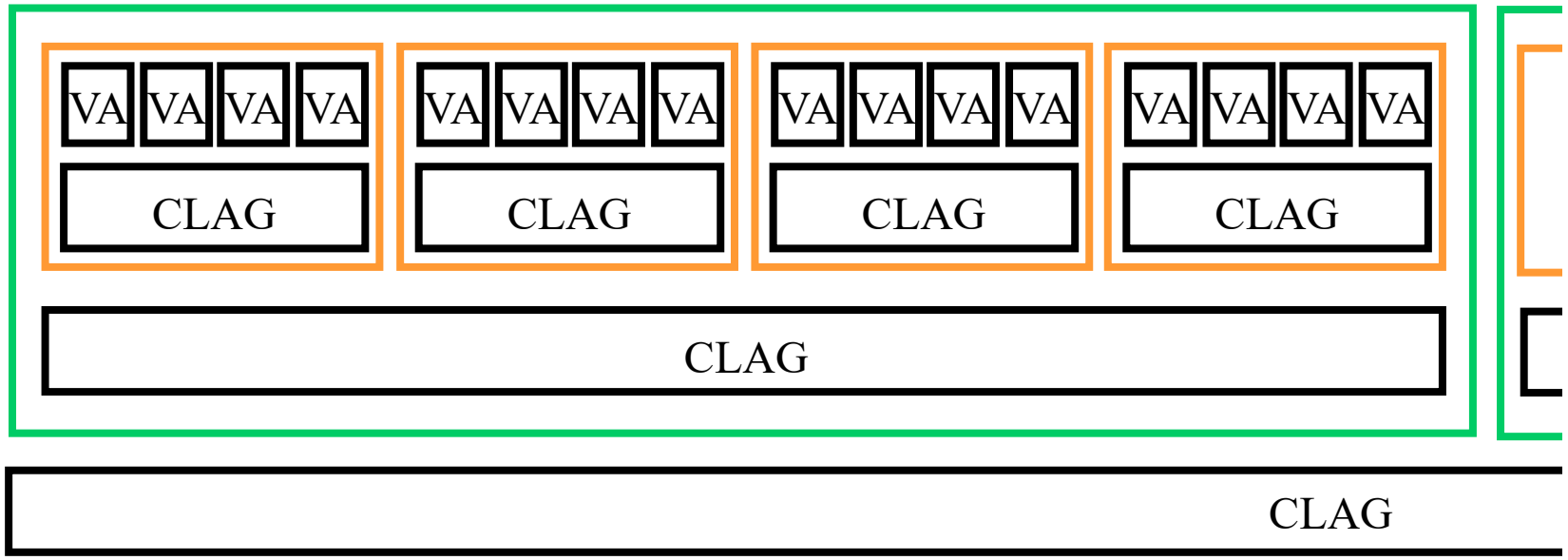
$$P = p_3p_2p_1p_0$$

- der CLAG generiert die Hilfsvariablen Block-Generate G und Block-Propagate P, die man mit einem weiteren CLAG zur Berechnung der Blocküberträge verwendet
- auf der vorherigen Folie ist nur



Carry-Look-Ahead Addierer (7)

- **Struktur**

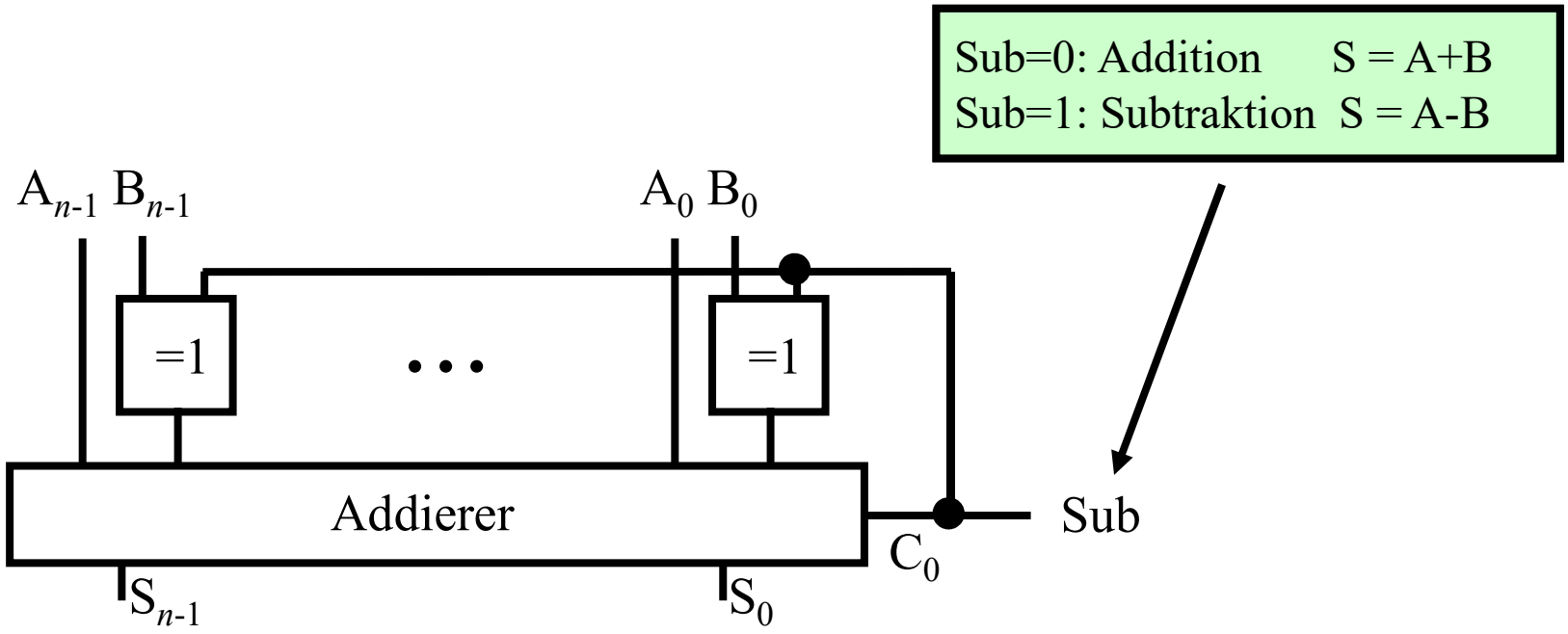


Carry-Look-Ahead Addierer (8)

- **Aufwand**
  - Verzögerungszeit ist O(log n)
    - Anzahl der CLAG-Schichten: k
    - Anzahl der Bits $n = 4^k$, also $k = \log_4 n$
    - da die Verzögerungszeit durch die Volladdierer und Carry-Look-Ahead Generatoren (CLAG's) jeweils konstant sind, also O(1), ist die Gesamtverzögerungszeit O(log n), da das Signal durch alle k Schichten hindurch muss
  - Hardwareaufwand (Anzahl der Gatter) ist O(n)
    - der Aufwand für die Volladdierer ist O(n)
    - ein einzelner CLAG hat Aufwand O(1), da er in diesem Fall nicht mit n wächst, sondern auf 4 Bit festgelegt ist
    - gesamte Anzahl der CLAG = $\sum_{i=0}^{k-1} 4^i = \frac{4^k - 1}{4 - 1} = \frac{n - 1}{3} = \mathrm{O}(n)$
    - also Gesamtaufwand ebenfalls O(n)

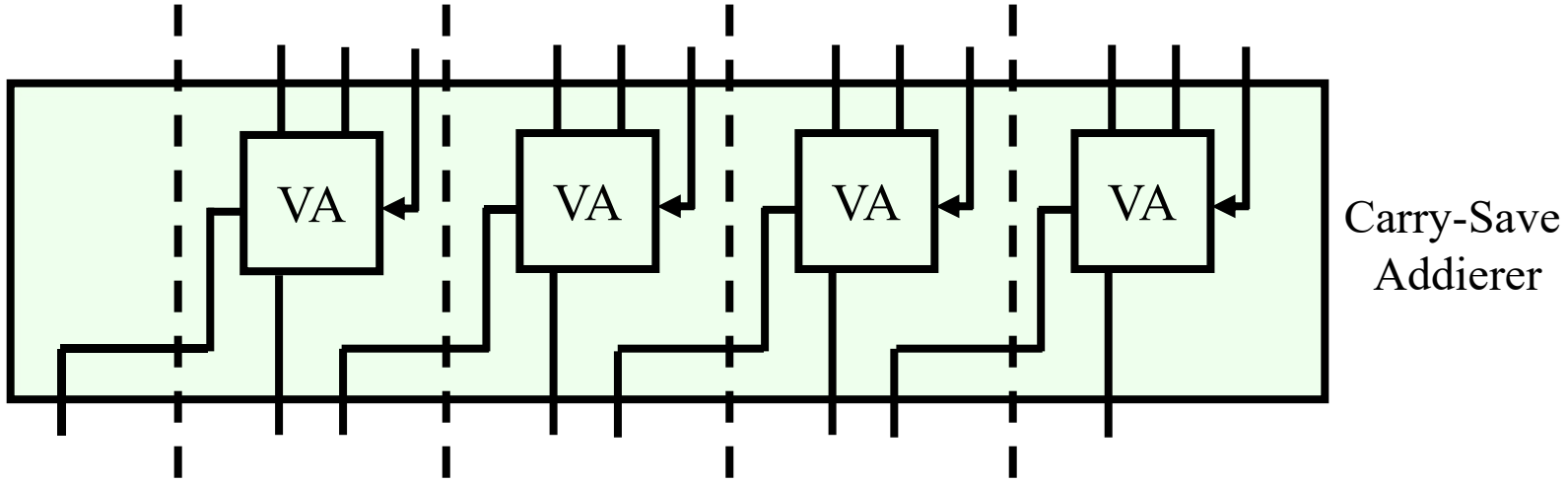
Subtrahierer

- **Subtraktion**
  - wird ersetzt durch Addition des Zweierkomplementes
  - Bildung Zweierkomplement
    - Negation aller Bits
    - Addition einer 1



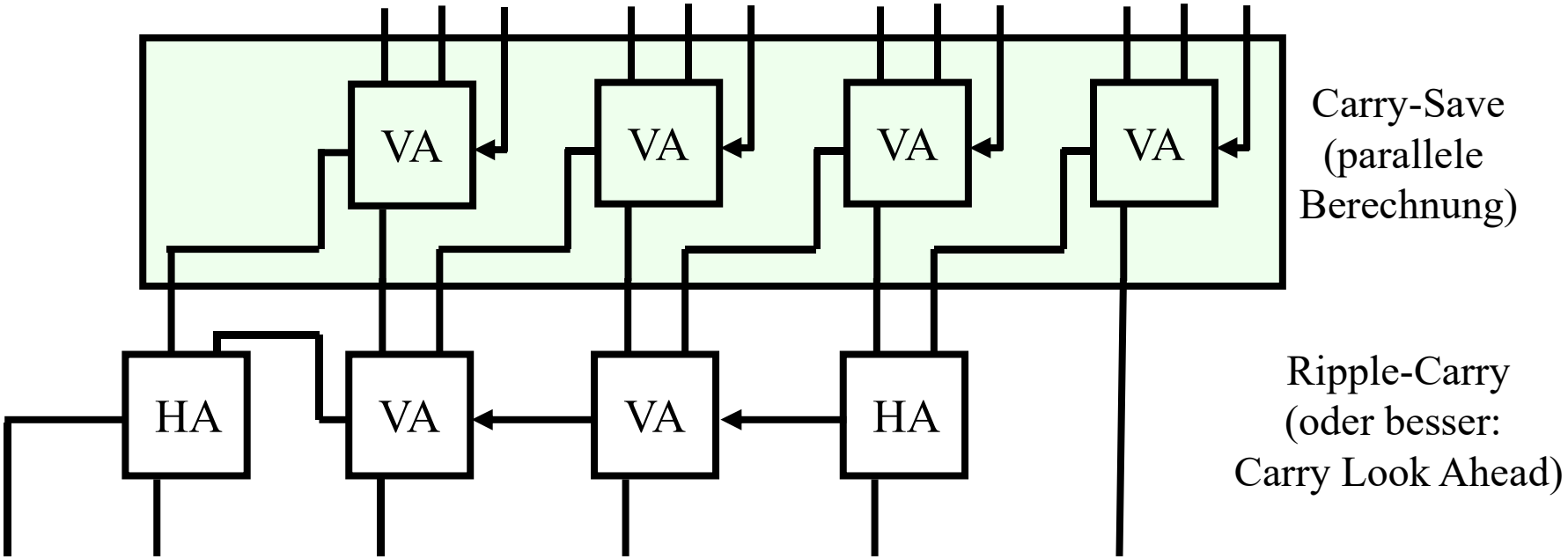
Carry-Save Addierer

- **Addition zweier Zahlen**
  - Carry-Look-Ahead Addierer ist optimiert, um zwei Zahlen in konstanter Zeit (unabhängig von *n*, der Anzahl der Bits) zu addieren
- **Addition von mehr als zwei Zahlen**
  - hier gibt es Lösungen mit weniger Aufwand
  - Idee: drei Zahlen bitweise in Volladdieren addieren und Übertrag als Ergebnis mit ausgeben
    - aus drei *n*-bit Zahlen werden zwei *n*-bit Zahlen mit identischer Summe



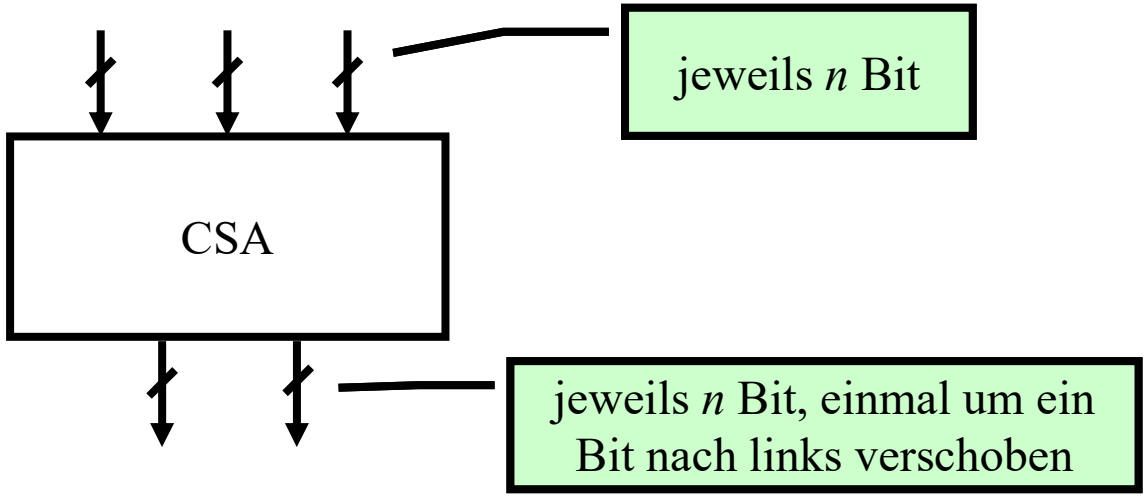
Carry-Save Addierer (2)

- **Anwendung: Addition von drei *n*-bit Zahlen**
  - um die endgültige Summe zu berechnen, müssen die beiden Ergebnisse noch addiert werden
    - z.B mit Ripple-Carry Addierer (wenig Aufwand, aber langsam)
    - oder Carry-Look-Ahead Addierer (mehr Aufwand, aber schneller)



Carry-Save Addierer (3)

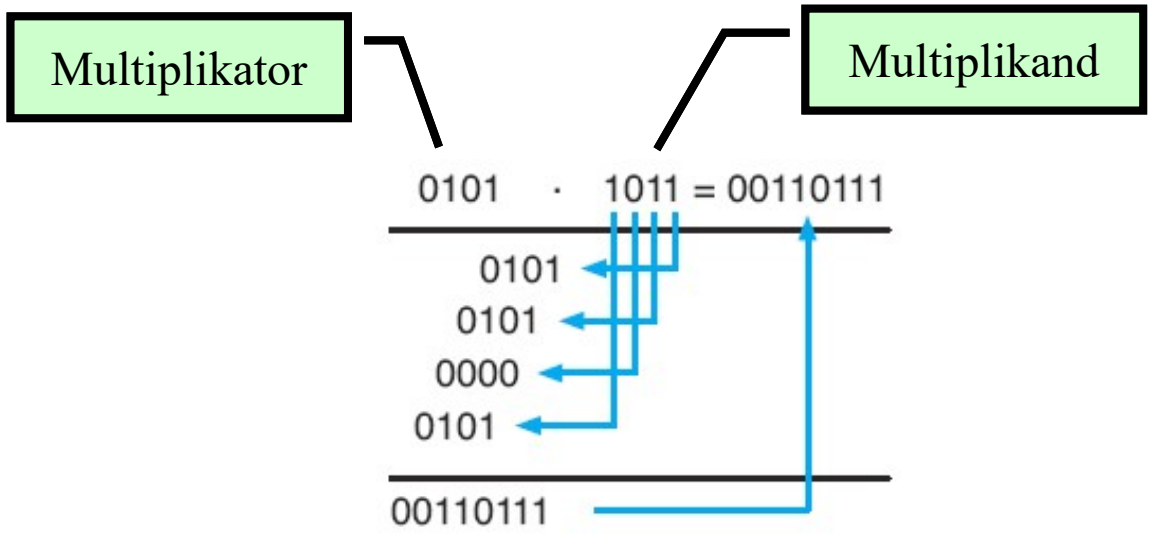
- **Carry-Save Addierer: CSA**
  - reduziert drei *n*-bit Zahlen zu zwei *n*-bit Zahlen mit identischer Summe
  - Berechnung geschieht vollständig parallel (3 Gatterlaufzeiten)



- **Prinzip kaskadierbar für die Addition von mehr als drei Zahlen**
  - zum Beispiel für die Implementierung einer Multiplikation

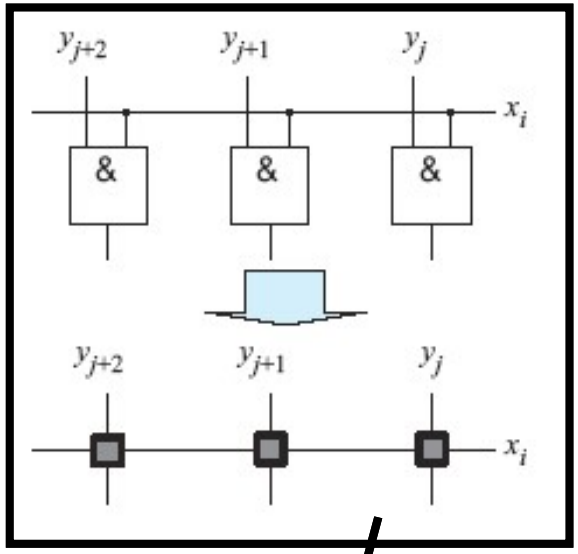
Multiplizierer

- **Einfache Multiplikation**
  - Multiplikation nach Schulmethode



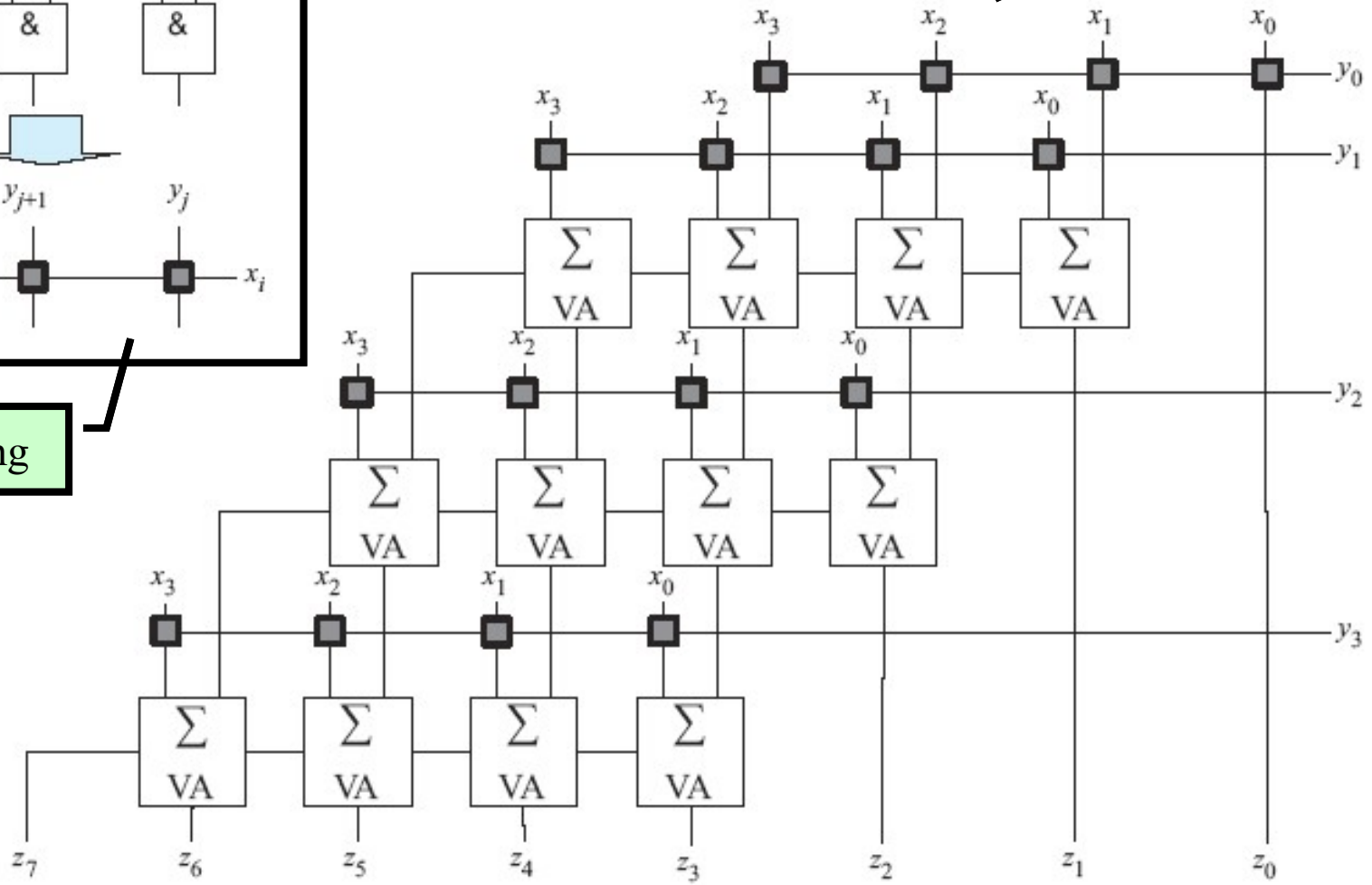
  - bei *n* x *n* Bit müssen *n* Partialprodukte addiert werden
    - Partialprodukte sind entweder 0 oder der Multiplikator selbst, je nach Bit des Multiplikanden
    - mit UND-Gattern realisiert

Multiplizierer (2)



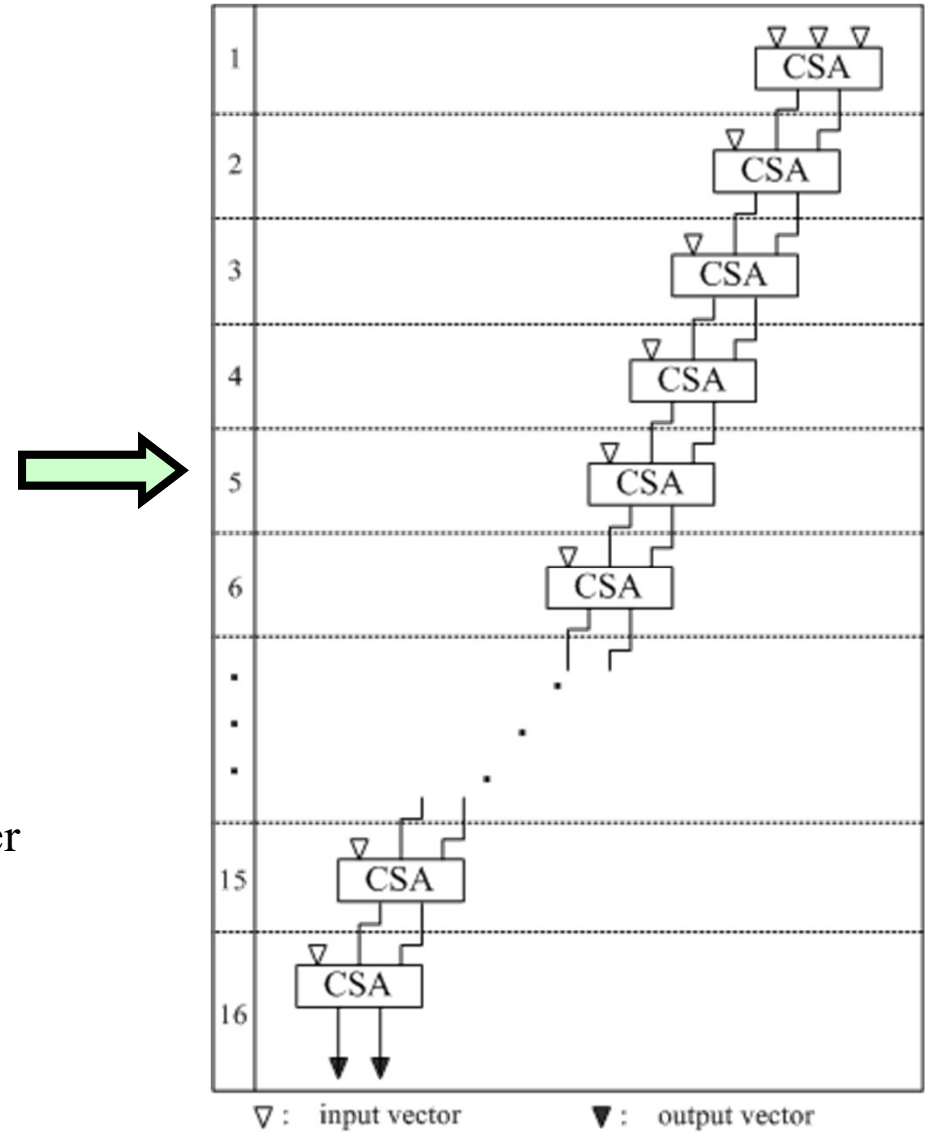
Abkürzung

Vollständiger Multiplizierer



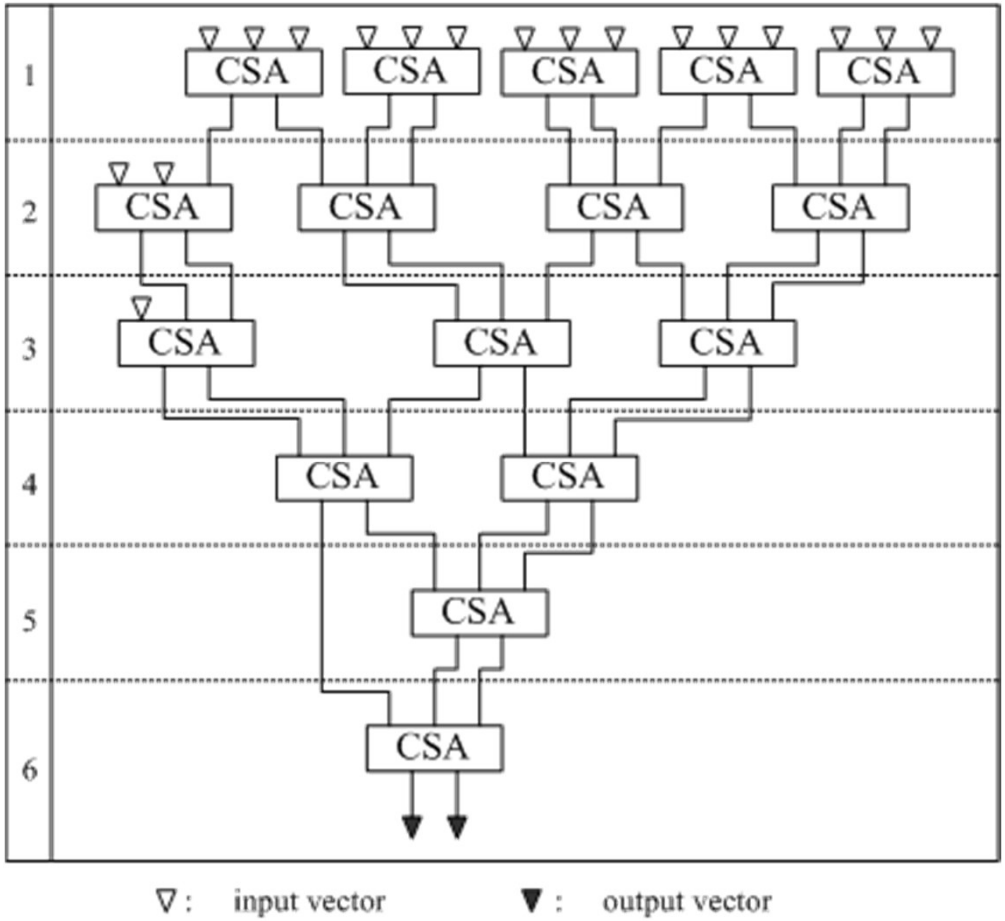
Multiplizierer (3)

- da viele Additionen nacheinander ausgeführt werden müssen, bietet sich die Implementierung mit Carry-Save Addierern an
- z.B. Addition von 18 Partialprodukten bei einer 18x18 Bit Multiplikation
  - es fehlt noch der abschließende Carry-Look-Ahead Addierer
  - für einen Multiplizierer müssen noch die UND-Gatter an den Eingängen ergänzt werden
- Durchlaufverzögerung O(n)



Multiplizierer (4)

- **geschickter: Baumartige Strukturen**
  - Durchlaufverzögerung: O(log *n)*
  - am Ende muss noch eine letzte Addition durchgeführt werden
    - z.B mit Carry-Look-Ahead Addierer



Arithmetisch logische Einheit, ALU

- **ALU**
  - *Arithmetic Logic Unit*
  - berechnet arithmetische und logische Operationen auf *n*-Bit Worten
  - ist häufig aus identischen Teilen zusammengesetzt, die jeweils ein Bit verarbeiten

ALU (2)

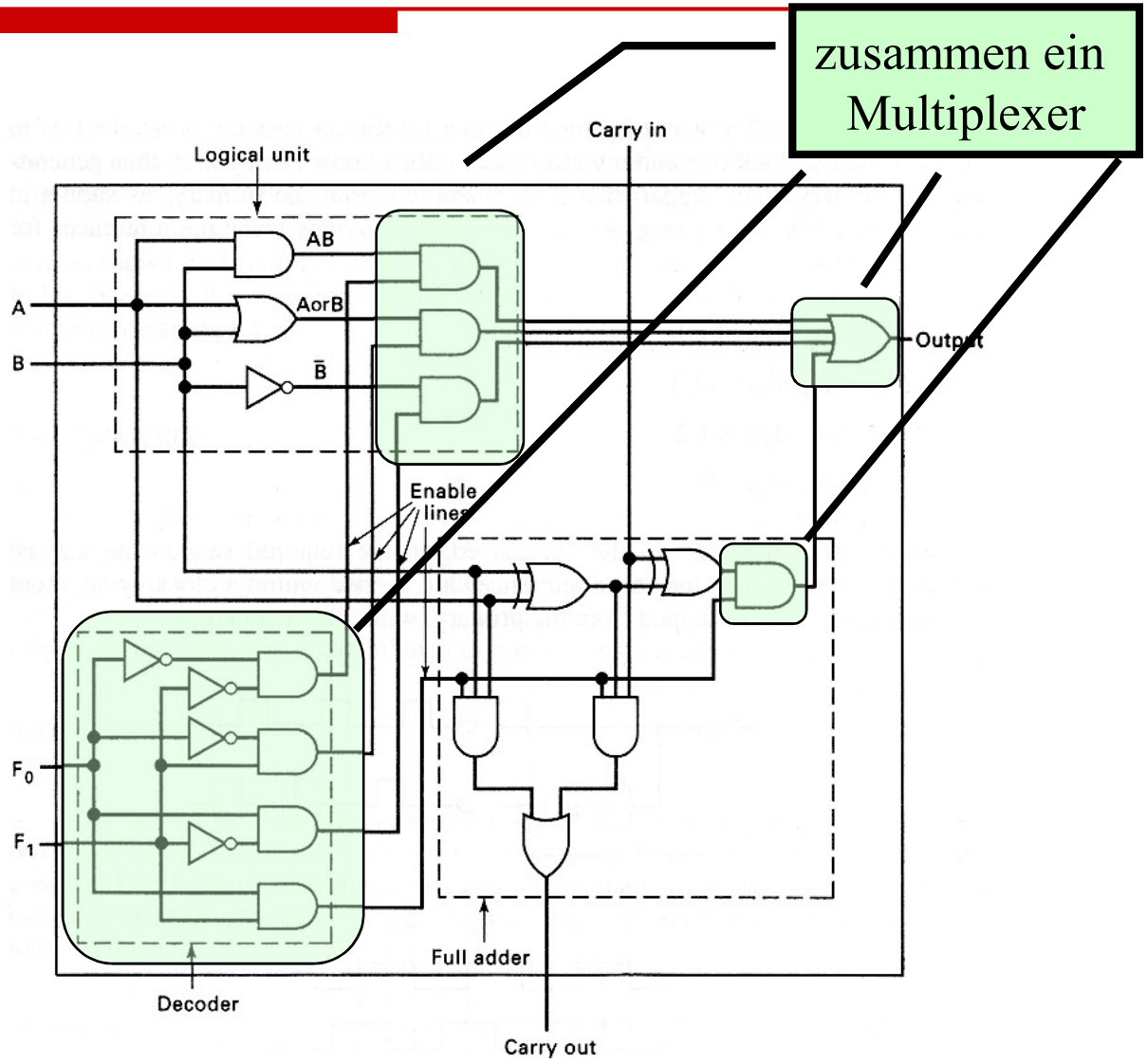
Beispiel:
einfache 1-Bit ALU

Steuerleitungen F_0, F_1

Dekoder wählt eine
von 4 Operationen aus

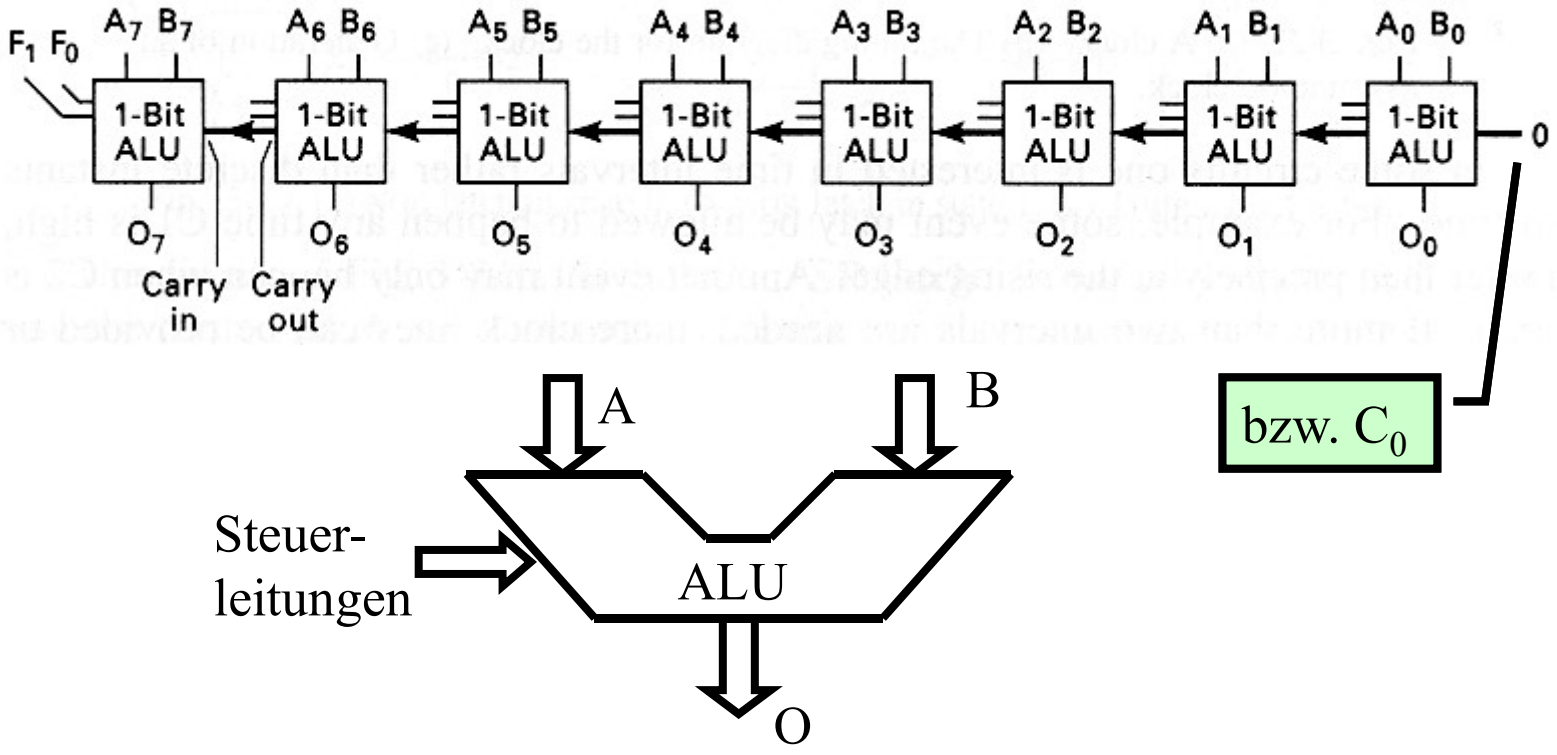
F_1	F_0	Output
0	0	A UND B
0	1	NICHT B
1	0	A ODER B
1	1	A+B

Addition



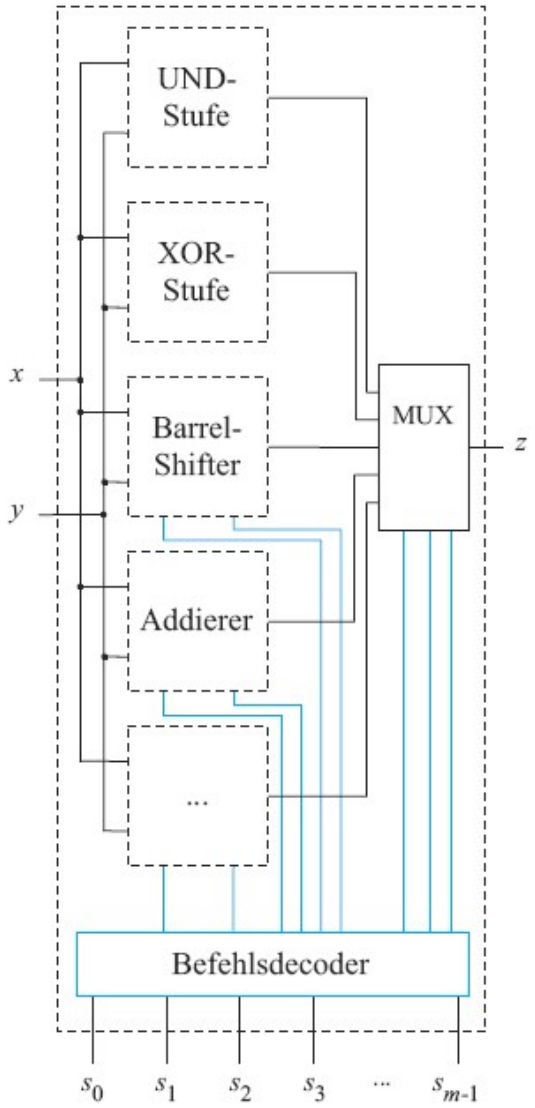
ALU (3)

- **8-Bit ALU aus 8 1-Bit ALU's**



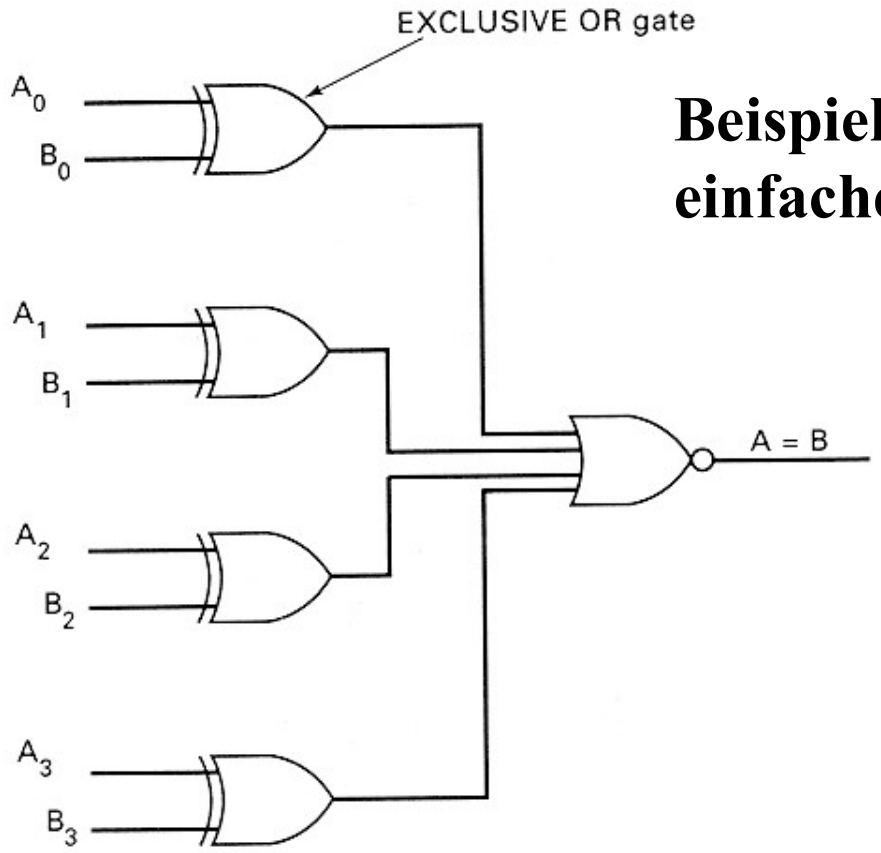
ALU (4)

- **Allgemeiner Aufbau einer ALU**
  - Funktionale Einheiten
    - verantwortlich für die Implementierung einzelner unterstützter Operationen
  - MUX
    - Multiplexer, der das gewünschte Ergebnis auswählt
  - Befehlsdecoder
    - Schaltnetz, das aus den ALU-Steuersignalen interne Steuersignale für die funktionalen Einheiten und den Multiplexer generiert



Komparator

- Vergleich zweier Binärzahlen auf Gleichheit



Beispiel:
einfacher 4-Bit Komparator

Komparator (2)

- **zusätzliche Ausgänge für "<" bzw. ">"**
  - kann immer mit Subtrahierer realisiert werden
  - Annahme: Ergebnisse liegen in Zweierkomplementdarstellung vor