

# Rechnersysteme und -netze

## Kapitel 1

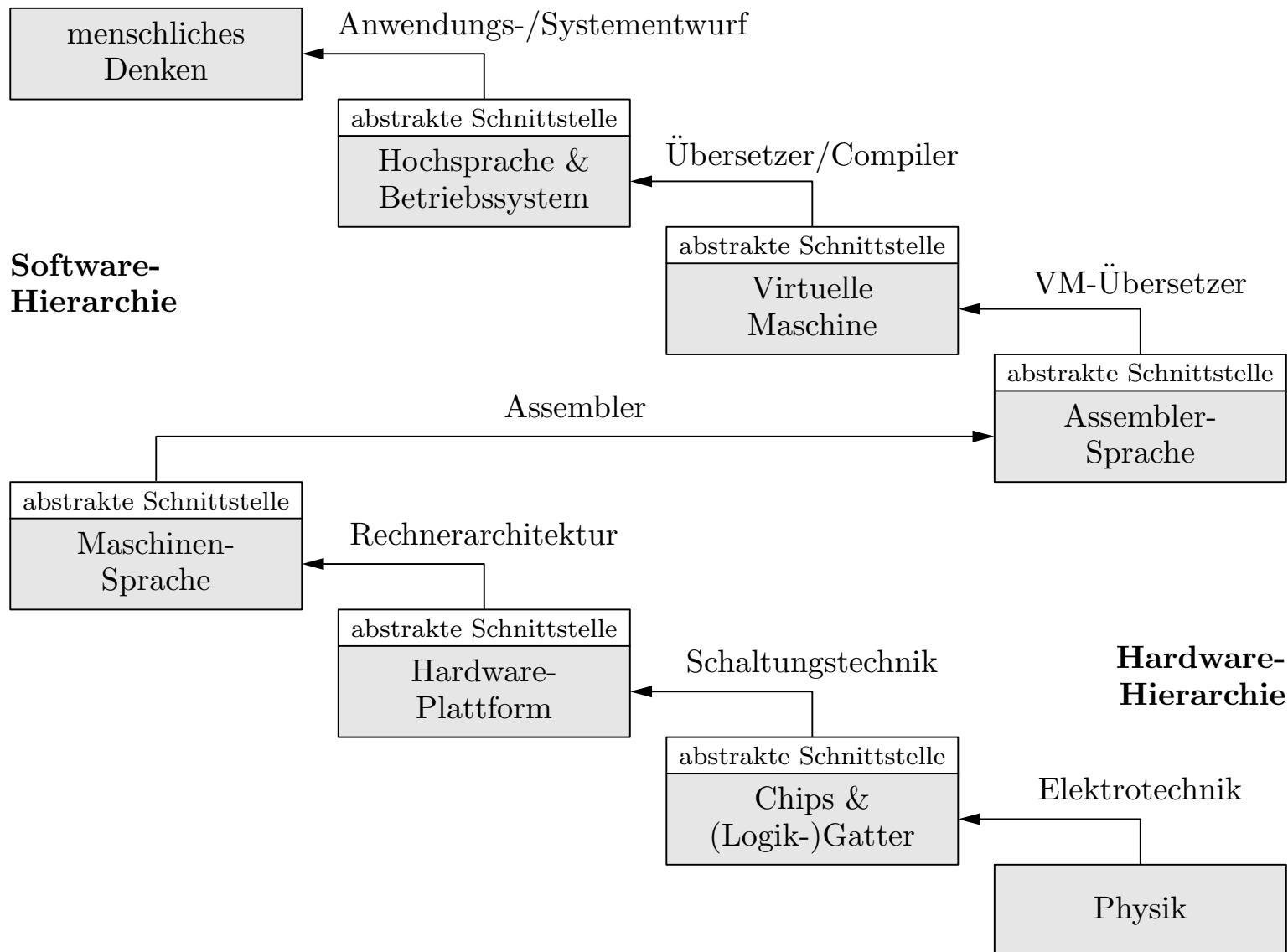
# Schaltungstechnik Teil I

**Bastian Goldlücke**

Universität Konstanz  
WS 2020/21

---

# Rechnersysteme: Plan der Vorlesung



# Inhalt

## **1 Boolesche Algebra / Schaltalgebra**

- 1.1 Definitionen, Operatoren, Schreibweisen
- 1.2 Boolesche Funktionen
- 1.3 Disjunktive und konjunktive Normalform
- 1.4 Vollständige Operationenmengen

## **2 Gatterlogik**

- 2.1 Elementare (Logik-)Gatter, Schaltzeichen für Gatter
- 2.2 Zusammengesetzte Gatter: Schnittstelle und Implementierung
- 2.3 Bitmustererkennung, Datenflußsteuerung, Multiplexer und Dekodierer

## **3 Implementierung in Schaltnetzen**

- 3.1 Elektrotechnische Grundlagen: Schalter und Transistoren
- 3.2 Gatter-Transistorschaltungen
- 3.3 Integrierte Schaltungen und ihre Herstellung
- 3.4 Evolution der Integrierten Schaltungen

# Inhalt

## **1 Boolesche Algebra / Schaltalgebra**

- 1.1 Definitionen, Operatoren, Schreibweisen
- 1.2 Boolesche Funktionen
- 1.3 Disjunktive und konjunktive Normalform
- 1.4 Vollständige Operationenmengen

## **2 Gatterlogik**

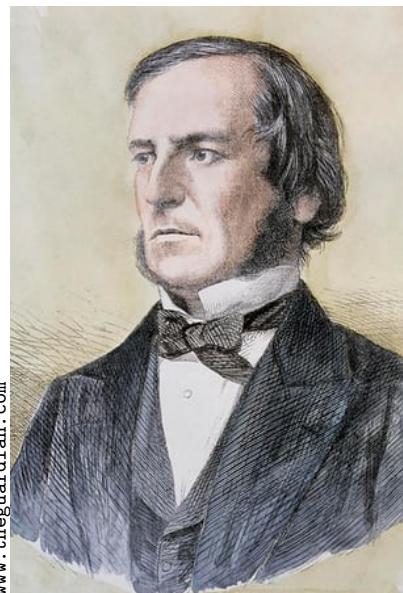
- 2.1 Elementare (Logik-)Gatter, Schaltzeichen für Gatter
- 2.2 Zusammengesetzte Gatter: Schnittstelle und Implementierung
- 2.3 Bitmustererkennung, Datenflußsteuerung, Multiplexer und Dekodierer

## **3 Implementierung in Schaltnetzen**

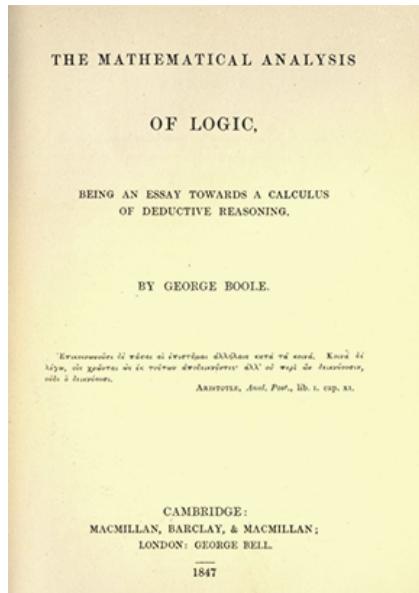
- 3.1 Elektrotechnische Grundlagen: Schalter und Transistoren
- 3.2 Gatter-Transistorschaltungen
- 3.3 Integrierte Schaltungen und ihre Herstellung
- 3.4 Evolution der Integrierten Schaltungen

# Boolesche Algebra

George **Boole** [1815–1864]



[www.theguardian.com](http://www.theguardian.com)



“The Mathematical Analysis of Logic”  
Cambridge/London 1847

“[T]he application of a new and peculiar form of Mathematics to the expression of the operations of the mind in reasoning.”

## Algebra



Das arabische Wort *al-ğabr* bedeutet das Zusammenfügen zerbrochener Teile (reunion of broken parts).

Es ist dem Titel des Rechenbuchs *Al-Kitab al-muh taṣar fī hisāb al-ğabr wa-l-muqābala* („Das kurz gefaßte Buch über die Rechenverfahren durch Ergänzen und Ausgleichen“, um 825) des persischen Mathematikers *al-Chwarizmi* entnommen.

Der Begriff **Algorithmus** leitet sich von dem Namen *al-Chwarizmi* ab.

# Boolesche Algebra: Definition

Algebraische Einkleidung der Aussagenlogik

[George Boole 1847]

Eine **Boolesche Algebra** ist eine Menge  $B$  mit dem Nullelement 0 und dem Einselement 1 (d.h.,  $0, 1 \in B$ ), auf der die zweistelligen Operationen **Konjunktion**  $\wedge$  und **Disjunktion**  $\vee$  sowie die einstellige Operation **Negation**  $\neg$  definiert sind.

(Redundantes) Axiomensystem (es sind  $a, b, c \in B$ ):

[Guiseppe Peano 1888]

Kommutativitat

$$a \wedge b = b \wedge a$$

$$a \vee b = b \vee a$$

Assoziativitat

$$(a \wedge b) \wedge c = a \wedge (b \wedge c)$$

$$(a \vee b) \vee c = a \vee (b \vee c)$$

Idempotenz

$$a \wedge a = a$$

$$a \vee a = a$$

Distributivitat

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

Neutralitat

$$a \wedge 1 = a$$

$$a \vee 0 = a$$

Extremalitat

$$a \wedge 0 = 0$$

$$a \vee 1 = 1$$

Involution

$$\neg\neg a = a$$

[Augustus De Morgan 1860]

De Morgan

$$\neg(a \wedge b) = \neg a \vee \neg b$$

$$\neg(a \vee b) = \neg a \wedge \neg b$$

Komplementaritat

$$a \wedge \neg a = 0$$

$$a \vee \neg a = 1$$

Dualitat

$$\neg 0 = 1$$

$$\neg 1 = 0$$

Absorption

$$a \vee (a \wedge b) = a$$

$$a \wedge (a \vee b) = a$$

# Boolesche Algebra: Schaltalgebra

Eine **Schaltalgebra** ist eine Boolesche Algebra mit der Trägermenge  $B = \{0, 1\}$ .

- Eine Schaltalgebra schränkt die Trägermenge  $B$  gegenüber einer allgemeinen Booleschen Algebra stark ein, denn im Prinzip kann die Trägermenge  $B$  einer Booleschen Algebra außer 0 und 1 noch andere Elemente enthalten.
- Ein Beispiel für eine Boolesche Algebra, die keine Schaltalgebra ist, ist die Potenzmenge  $2^U$  einer Menge  $U$ , mit Durchschnitt, Vereinigung und Komplement (bzgl.  $U$ ). Hier ist 0 die leere Menge  $\emptyset$  und 1 die ganze Menge  $U$ .
- Wir werden in dieser Vorlesung nur Schaltalgebren betrachten.
- Die logischen Operationen können über **Wahrheitstafeln** definiert werden:

$a$	$b$	$a \wedge b$
0	0	0
1	0	0
0	1	0
1	1	1

$a$	$b$	$a \vee b$
0	0	0
1	0	1
0	1	1
1	1	1

$a$	$\neg a$
0	1
1	0

Wahrheitstafel: Auflistung der möglichen Wertekombinationen der Argumente einer Operation zusammen mit den dazu gehörenden Funktions-/Ausgabewerten.

## Boolesche Algebra: Bemerkungen

- George Boole benutzte Rechenzeichen für die logischen Operationen, nämlich das Produkt  $\cdot$  für die Konjunktion  $\wedge$ , die Addition  $+$  für die Disjunktion  $\vee$  (eigentlich  $x \vee y \equiv x + y - x \cdot y$ ) und die Subtraktion  $1 - x$  für die Negation  $\neg$ .
- Giuseppe Peano benutzte die Symbole  $\cap$  und  $\cup$  für Konjunktion bzw. Disjunktion.
- Das aussagenlogische Zeichen  $\vee$  stammt von [Bertrand Russel 1906], die aussagenlogischen Zeichen  $\wedge$  und  $\neg$  von [Arend Heyting 1930].
- Das Negationszeichen erinnert an den Negationsstrich in der Begriffsschrift von [Gottlob Frege 1879].
- Die Negation wird oft auch durch einen Querstrich über einem VariablenSymbol oder einer Teilformel geschrieben:
$$\bar{a} \equiv \neg a \quad \text{und} \quad \overline{a \wedge b} \equiv \neg(a \wedge b).$$
- In abkürzender Schreibweise fehlt manchmal das Konjunktionszeichen  $\wedge$ , wie man auch in der Arithmetik das Produktzeichen  $\cdot$  oft nicht schreibt.

## Boolesche Algebra: Bemerkungen

- Zu jeder Gleichung einer Booleschen Algebra gehört eine **duale Gleichung**, die durch Ersetzen von 0 durch 1 und von  $\wedge$  durch  $\vee$  und umgekehrt entsteht.

- Wenn eine Gleichung einer Booleschen Algebra gültig ist, dann ist auch ihre duale Gleichung gültig.

Beachte: Außer der Involution (zu der es keine duale Gleichung gibt) sind alle Axiome in ihren beiden dualen Formen angegeben.

-  $a \wedge \neg a = 0$  heißt auch **Gesetz des ausgeschlossenen Widerspruchs** (law of non-contradiction).

-  $a \vee \neg a = 1$  heißt auch **Gesetz des ausgeschlossenen Dritten** (law of excluded middle).

- Zusammen mit  $a = a$ , dem **Gesetz der Identität** (law of identity), bilden sie die drei klassischen Denkgesetze der Philosophie.

- Eine Boolesche Algebra kann auch (im System der algebraischen Strukturen) als **distributiver komplementärer Verband** definiert werden.

# Boolesche Funktionen

- Eine **Boolesche Funktion** ist eine Funktion  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ ,  $n \geq 0$ . Die Anzahl  $n$  der Argumente der Funktion  $f$  heißt ihre **Stelligkeit (arity)**.
- Spezielle Ausdrücke für kleine Stelligkeiten (auch anderer Funktionen):  $n = 1$ : **unär (unary)**,  $n = 2$ : **binär (binary)**,  $n = 3$ : **ternär (ternary)**.  
Achtung: "binäre Funktion" wird manchmal synonym zu "Boolesche Funktion" verwendet. Dies kann zu Mißverständnissen führen und sollte vermieden werden.
- Man kann auch verallgemeinerte Boolesche Funktionen  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  betrachten, jedoch ist es einfacher, diese als  $m$  Funktionen  $f_i : \{0, 1\}^n \rightarrow \{0, 1\}$ ,  $i = 1, \dots, m$ , zu behandeln.
- Boolesche Funktionen werden durch **Schaltnetze** implementiert. Kernfragen:
  - Welche Booleschen Funktionen lassen sich implementieren?  
Antwort: Alle! (Dies werden wir im folgenden beweisen.)
  - Welche (Logik-)Gatter braucht man zur Implementierung?  
Antwort: Solche, die eine **vollständige Operationenmenge** bilden.

# Binäre Boolesche Funktionen

- Jedes der beiden Argumente einer binären Booleschen Funktion kann zwei Werte annehmen: 0 und 1.
- Folglich gibt es  $2^2 = 4$  mögliche Argumentwertkombinationen für binäre Boolesche Funktionen.
- Zu jeder Argumentwertkombination gibt es zwei mögliche Funktionswerte: 0 und 1.
- Folglich gibt es  $2^{(2^2)} = 2^4 = 16$  binäre Boolesche Funktionen.
- Diese 16 verschiedenen Funktionen sind in der Tabelle rechts gezeigt.

Funktion	x	0	0	1	1
	y	0	1	0	1
konstant 0	0	0	0	0	0
$x \text{ und } y$	$x \wedge y$	0	0	0	1
$x \text{ und nicht } y$	$x \wedge \neg y$	0	0	1	0
$x$	$x$	0	0	1	1
nicht $x$ und $y$	$\neg x \wedge y$	0	1	0	0
$y$	$y$	0	1	0	1
exklusives Oder	$x \oplus y$	0	1	1	0
$x \text{ oder } y$	$x \vee y$	0	1	1	1
negiertes Oder	$x \downarrow y$	1	0	0	0
Äquivalenz	$x \leftrightarrow y$	1	0	0	1
nicht $y$	$\neg y$	1	0	1	0
wenn $y$ , dann $x$	$y \rightarrow x$	1	0	1	1
nicht $x$	$\neg x$	1	1	0	0
wenn $x$ , dann $y$	$x \rightarrow y$	1	1	0	1
negiertes Und	$x \mid y$	1	1	1	0
konstant 1		1	1	1	1

# Boolesche Funktionen: Darstellung

- Jede Boolesche Funktion kann durch ihre **Wahrheitstafel** dargestellt werden.
- Eine solche Wahrheitstafel enthält:
  - eine Spalte für jedes Funktionsargument,
  - eine Zeile für jede mögliche Wertekombination der Funktionsargumente,
  - eine zusätzliche Spalte, in der zu den verschiedenen Wertekombinationen der zugehörige Funktionswert angegeben wird.
- Die Wahrheitstafel einer  $n$ -stelligen Booleschen Funktion hat  $2^n$  Zeilen (zwei mögliche Werte für jedes der  $n$  Argumente).
- Es gibt  $2^{(2^n)}$   $n$ -stellige Boolesche Funktionen (zwei mögliche Werte für jede der  $2^n$  Zeilen).

## **Beispiel:**

ternäre Boolesche Funktion

$$y = f(x_1, x_2, x_3):$$

$x_1$	$x_2$	$x_3$	$y$
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	0
0	0	1	1
1	0	1	0
0	1	1	1
1	1	1	0

$$2^3 = 8 \text{ Zeilen}$$

$$2^{(2^3)} = 2^8 = 256 \text{ mögliche dreistellige Funktionen}$$

# Boolesche Funktionen: Darstellung

- Jede Boolesche Funktion kann durch **Formeln der Schaltalgebra** dargestellt werden, die aus ihrer Wahrheitstafel konstruiert werden.
- 1. Ansatz: **Disjunktive Normalform**  
(auch: Darstellung durch Minterme, sum of products, SOP)
  - Grundidee: Der Wert der Funktion ist 0, es sei denn, es liegen bestimmte Argumentwertkombinationen vor.
  - Nur die Zeilen der Wahrheitstafel, für die der Funktionswert 1 ist, müssen beschrieben werden (Konvention: leere Disjunktion hat Wert 0).
- 2. Ansatz: **Konjunktive Normalform**  
(auch: Darstellung durch Maxterme, product of sums, POS)
  - Grundidee: Der Wert der Funktion ist 1, es sei denn, es liegen bestimmte Argumentwertkombinationen vor.
  - Nur die Zeilen der Wahrheitstafel, für die der Funktionswert 0 ist, müssen beschrieben werden (Konvention: leere Konjunktion hat Wert 1).

# Boolesche Funktionen: Disjunktive Normalform

- Stelle die Wahrheitstafel der Funktion auf.
- Für jede Zeile, für die der Funktionswert 1 ist, bilde eine **Konjunktion** von Literalen.
- Ein **Literal** ist ein Funktionsargument  $x_k$  (positives Literal) oder seine Negation  $\neg x_k$  bzw.  $\overline{x_k}$  (negatives Literal).
- Die zu bildende Konjunktion enthält ein positives Literal für alle Funktionsargumente, die 1 sind, und ein negatives Literal für alle Funktionsargumente, die 0 sind.
- Bilde die **Disjunktion** der so erhaltenen Konjunktionen.
- Das Ergebnis nennt man **disjuktive Normalform** der Booleschen Funktion.

## Beispiel:

ternäre Boolesche Funktion

$$y = f(x_1, x_2, x_3):$$

$j$	$x_1$	$x_2$	$x_3$	$y$	$C_j$
0	0	0	0	0	
1	1	0	0	1	$x_1 \wedge \overline{x_2} \wedge \overline{x_3}$
2	0	1	0	1	$\overline{x_1} \wedge x_2 \wedge \overline{x_3}$
3	1	1	0	0	
4	0	0	1	1	$\overline{x_1} \wedge \overline{x_2} \wedge x_3$
5	1	0	1	0	
6	0	1	1	1	$\overline{x_1} \wedge x_2 \wedge x_3$
7	1	1	1	0	

$$\begin{aligned}D &= C_1 \vee C_2 \vee C_4 \vee C_6 \\&= (x_1 \wedge \overline{x_2} \wedge \overline{x_3}) \vee (\overline{x_1} \wedge x_2 \wedge \overline{x_3}) \\&\quad \vee (\overline{x_1} \wedge \overline{x_2} \wedge x_3) \vee (\overline{x_1} \wedge x_2 \wedge x_3)\end{aligned}$$

# Boolesche Funktionen: Disjunktive Normalform

- Die Konjunktionen der disjunktiven Normalform sind genau dann wahr ( $= 1$ ), wenn die Wertekombination der Argumente vorliegt, für die sie gebildet wurden.
- Die Disjunktion ist genau dann wahr ( $= 1$ ), wenn mindestens eine der Konjunktionen wahr ( $= 1$ ) ist, d.h., wenn eine der Wertekombinationen vorliegt, für die die Boolesche Funktion 1 liefert; für alle anderen Wertekombinationen ist sie falsch ( $= 0$ ).  
Man beachte, daß auch nur höchstens eine der Konjunktionen wahr ( $= 1$ ) sein kann.
- Folglich stellt die so gebildete Formel die gewünschte Boolesche Funktion dar.  
 $\Rightarrow$  **Jede Boolesche Funktion hat eine disjunktive Normalform.**

**Beispiel:**  
ternäre Boolesche Funktion  
 $y = f(x_1, x_2, x_3)$ :

$j$	$x_1$	$x_2$	$x_3$	$y$	$C_j$
0	0	0	0	0	
1	1	0	0	1	$x_1 \wedge \overline{x_2} \wedge \overline{x_3}$
2	0	1	0	1	$\overline{x_1} \wedge x_2 \wedge \overline{x_3}$
3	1	1	0	0	
4	0	0	1	1	$\overline{x_1} \wedge \overline{x_2} \wedge x_3$
5	1	0	1	0	
6	0	1	1	1	$\overline{x_1} \wedge x_2 \wedge x_3$
7	1	1	1	0	

$$\begin{aligned}D &= C_1 \vee C_2 \vee C_4 \vee C_6 \\&= (x_1 \wedge \overline{x_2} \wedge \overline{x_3}) \vee (\overline{x_1} \wedge x_2 \wedge \overline{x_3}) \\&\quad \vee (\overline{x_1} \wedge \overline{x_2} \wedge x_3) \vee (\overline{x_1} \wedge x_2 \wedge x_3)\end{aligned}$$

# Boolesche Funktionen: Konjunktive Normalform

- Stelle die Wahrheitstafel der Funktion auf.
- Für jede Zeile, für die der Funktionswert 0 ist, bilde eine **Disjunktion** von Literalen.
- Ein **Literal** ist ein Funktionsargument  $x_k$  (positives Literal) oder seine Negation  $\neg x_k$  bzw.  $\overline{x_k}$  (negatives Literal).
- Die zu bildende Disjunktion enthält ein positives Literal für alle Funktionsargumente, die 0 sind, und ein negatives Literal für alle Funktionsargumente, die 1 sind.
- Bilde die **Konjunktion** der so erhaltenen Disjunktionen.
- Das Ergebnis nennt man **konjunktive Normalform** der Booleschen Funktion.

## Beispiel:

ternäre Boolesche Funktion  
 $y = f(x_1, x_2, x_3)$ :

$j$	$x_1$	$x_2$	$x_3$	$y$	$D_j$
0	0	0	0	0	$x_1 \vee x_2 \vee x_3$
1	1	0	0	1	
2	0	1	0	1	
3	1	1	0	0	$\overline{x_1} \vee \overline{x_2} \vee x_3$
4	0	0	1	1	
5	1	0	1	0	$\overline{x_1} \vee x_2 \vee \overline{x_3}$
6	0	1	1	1	
7	1	1	1	0	$\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$

$$\begin{aligned}C &= D_0 \wedge D_3 \wedge D_5 \wedge D_7 \\&= (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \\&\quad \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})\end{aligned}$$

# Boolesche Funktionen: Konjunktive Normalform

- Die Disjunktionen der konjunktiven Normalform sind genau dann falsch ( $= 0$ ), wenn die Wertekombination der Argumente vorliegt, für die sie gebildet wurden.
  - Die Konjunktion ist genau dann falsch ( $= 0$ ), wenn mindestens eine der Disjunktionen falsch ( $= 0$ ) ist, d.h., wenn eine der Wertekombinationen vorliegt, für die die Boolesche Funktion 0 liefert; für alle anderen Wertekombinationen ist sie wahr ( $= 1$ ).  
Man beachte, daß auch nur höchstens eine der Disjunktionen falsch ( $= 0$ ) sein kann.
  - Folglich stellt die so gebildete Formel die gewünschte Boolesche Funktion dar.
- ⇒ **Jede Boolesche Funktion hat eine konjunktive Normalform.**

## Beispiel:

ternäre Boolesche Funktion

$$y = f(x_1, x_2, x_3):$$

$j$	$x_1$	$x_2$	$x_3$	$y$	$D_j$
0	0	0	0	0	$x_1 \vee x_2 \vee x_3$
1	1	0	0	1	
2	0	1	0	1	
3	1	1	0	0	$\overline{x_1} \vee \overline{x_2} \vee x_3$
4	0	0	1	1	
5	1	0	1	0	$\overline{x_1} \vee x_2 \vee \overline{x_3}$
6	0	1	1	1	
7	1	1	1	0	$\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$

$$\begin{aligned}C &= D_0 \wedge D_3 \wedge D_5 \wedge D_7 \\&= (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \\&\quad \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})\end{aligned}$$

# Boolesche Funktionen: Konjunktive Normalform

- Alternative Ableitung der **konjunktiven Normalform**:
- Für jede Zeile, für die der Funktionswert 0 ist, bilde eine **Konjunktion** von Literalen.
- Die zu bildende Konjunktion enthält ein positives Literal für alle Funktionsargumente, die 1 sind, und ein negatives Literal für alle Funktionsargumente, die 0 sind.
- Bilde die **Konjunktion der Negationen** der so erhaltenen Konjunktionen.

Diese Negationen sind genau dann falsch (= 0), wenn die Wertekombination vorliegt, für die sie gebildet wurden; ihre Konjunktion ist genau dann falsch (= 0), wenn mindestens eine der Negationen falsch (= 0) ist.

## Beispiel:

ternäre Boolesche Funktion  
 $y = f(x_1, x_2, x_3)$ :

$j$	$x_1$	$x_2$	$x_3$	$y$	$C_j$
0	0	0	0	0	$\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}$
1	1	0	0	1	
2	0	1	0	1	
3	1	1	0	0	$x_1 \wedge x_2 \wedge \overline{x_3}$
4	0	0	1	1	
5	1	0	1	0	$x_1 \wedge \overline{x_2} \wedge x_3$
6	0	1	1	1	
7	1	1	1	0	$x_1 \wedge x_2 \wedge x_3$

$$\begin{aligned}C &= \overline{C_0} \wedge \overline{C_3} \wedge \overline{C_5} \wedge \overline{C_7} \\&= \overline{(\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3})} \wedge \overline{(x_1 \wedge x_2 \wedge \overline{x_3})} \\&\quad \wedge \overline{(x_1 \wedge \overline{x_2} \wedge x_3)} \wedge \overline{(x_1 \wedge x_2 \wedge x_3)}\end{aligned}$$

## Boolesche Funktionen: Konjunktive Normalform

- Bisher aufgestellte logische Formel:

$$C = \overline{(\bar{x}_1 \wedge \bar{x}_2 \wedge \bar{x}_3)} \wedge \overline{(x_1 \wedge x_2 \wedge \bar{x}_3)} \wedge \overline{(x_1 \wedge \bar{x}_2 \vee x_3)} \wedge \overline{(x_1 \wedge x_2 \vee x_3)}.$$

- Wende nun auf die einzelnen Terme der äußeren Konjunktion (d.h., auf die gebildeten negierten Konjunktionen für die Wertekombinationen, für die  $y = 0$  ist) die **De Morganschen Gesetze** an:

$$\overline{a \wedge b} = \bar{a} \vee \bar{b} \quad \text{und} \quad \overline{a \vee b} = \bar{a} \wedge \bar{b}.$$

Diese gelten auch in erweiterter Form (warum?  $\Rightarrow$  Übungsaufgabe, Blatt 3):

$$\overline{a \wedge b \wedge c} = \bar{a} \vee \bar{b} \vee \bar{c} \quad \text{und} \quad \overline{a \vee b \vee c} = \bar{a} \wedge \bar{b} \wedge \bar{c}.$$

- So erhält man (unter zusätzlicher Ausnutzung der **Involution**  $\bar{\bar{a}} = a$ ):

$$\begin{aligned} C &= \overline{(\bar{x}_1 \wedge \bar{x}_2 \wedge \bar{x}_3)} \wedge \overline{(x_1 \wedge x_2 \wedge \bar{x}_3)} \wedge \overline{(x_1 \wedge \bar{x}_2 \vee x_3)} \wedge \overline{(x_1 \wedge x_2 \vee x_3)} \\ &= (\bar{\bar{x}_1} \vee \bar{\bar{x}_2} \vee \bar{\bar{x}_3}) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{\bar{x}_3}) \wedge (\bar{x_1} \vee \bar{\bar{x}_2} \vee \bar{x_3}) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \\ &= (x_1 \vee x_2 \vee x_3) \wedge (\bar{x_1} \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \end{aligned}$$

# Konjunktive oder Disjunktive Normalform?

- **Disjunktive Normalform** (sum of products, SOP)
    - Besteht aus so vielen Konjunktionen (disjunktiv verknüpft), wie die Wahrheitstafel Zeilen mit Ausgabe 1 enthält.
    - Alle Konjunktion sind „gleich groß“ (enthalten die gleiche Anzahl Literale).
    - Folglich hängt die Größe der Formel, gemessen in der Zahl der Literale, nur von der Zahl der Zeilen mit Ausgabe 1 ab.
  - **Konjunktive Normalform** (product of sums, POS)
    - Besteht aus so vielen Disjunktionen (konjunktiv verknüpft), wie die Wahrheitstafel Zeilen mit Ausgabe 1 enthält.
    - Alle Disjunktion sind „gleich groß“ (enthalten die gleiche Anzahl Literale).
    - Folglich hängt die Größe der Formel, gemessen in der Zahl der Literale, nur von der Zahl der Zeilen mit Ausgabe 0 ab.
- ⇒ Bevorzuge die disjunktive Normalform bei weniger Einsen, die konjunktive Normalform bei weniger Nullen in der Ausgabe.

# Konjunktive oder Disjunktive Normalform?

- **Allgemeine Regel:** (wenn es denn eine der Normalformen sein soll)

Bevorzuge die **disjunktive** Normalform bei **weniger Einsen**,  
bevorzuge die **konjunktive** Normalform bei **weniger Nullen** in der Ausgabe.

**Aber:**

- Es ist i.a. möglich, eine einfachere Formel (und folglich eine einfachere Schaltung) aus der Wahrheitstafel abzuleiten:  
**Minimierung von logischen Ausdrücken** ( $\Rightarrow$  3. Vorlesung).
- Es kann vorteilhaft sein, die Funktion durch andere logische Operationen darzustellen als durch Konjunktion  $\wedge$ , Disjunktion  $\vee$  und Negation  $\neg$  :  
**Verwendung anderer vollständiger Operationenmengen.**
- Entscheidend ist im zweiten Fall die (einfache) Darstellbarkeit der verwendeten logischen Operationen durch Transistorschaltungen.

## Boolesche Funktionen: Vollständige Operationenmengen

- Die Darstellung einer Booleschen Funktion durch ihre disjunktive oder konjunktive Normalform zeigt, daß alle Booleschen Funktionen durch Formeln der Schaltalgebra dargestellt werden können.  
Denn jede Boolesche Funktion kann durch ihre Wahrheitstafel dargestellt werden, und aus der Wahrheitstafel können diese Normalformen konstruiert werden.
- Eine **vollständige Operationenmenge** ist eine Menge von Booleschen Operationen, die ausreicht, um alle Booleschen Funktionen darzustellen. (engl: functionally complete operation set)
- Da sowohl die disjunktive Normalform als auch die konjunktive Normalform nur die Operationen Konjunktion  $\wedge$ , Disjunktion  $\vee$  und Negation  $\neg$  benutzt, ist die Menge  $O = \{\wedge, \vee, \neg\}$  eine vollständige Operationenmenge.
- Von einer anderen Operationenmenge  $O'$  kann man zeigen, daß sie vollständig ist, indem man die drei Operationen Konjunktion  $\wedge$ , Disjunktion  $\vee$  und Negation  $\neg$  nur mit Hilfe von Operationen aus dieser Menge  $O'$  darstellt.

## Boolesche Funktionen: Vollständige Operationenmengen

Die Mengen  $O_1 = \{| \}$  (Sheffer-Strich [Henry Maurice Sheffer 1913], NAND) und  $O_2 = \{\downarrow\}$  (Peirce-Pfeil [Charles Sanders Peirce 1880], NOR) sind vollständig.

$O_1 = \{| \}$  ist vollständig

$(a | b = \neg(a \wedge b) = \text{NAND}(a, b)):$

$$\begin{aligned}\neg a &= \neg(a \wedge a) \\ &= a | a\end{aligned}$$

$$\begin{aligned}a \wedge b &= \neg\neg(a \wedge b) \\ &= \neg(a | b) \\ &= (a | b) | (a | b)\end{aligned}$$

$$\begin{aligned}a \vee b &= \neg\neg(a \vee b) \\ &= \neg(\neg a \wedge \neg b) \\ &= \neg a | \neg b \\ &= (a | a) | (b | b)\end{aligned}$$

$O_2 = \{\downarrow\}$  ist vollständig

$(a \downarrow b = \neg(a \vee b) = \text{NOR}(a, b)):$

Übungsaufgabe

# Inhalt

## 1 Boolesche Algebra / Schaltalgebra

- 1.1 Definitionen, Operatoren, Schreibweisen
- 1.2 Boolesche Funktionen
- 1.3 Disjunktive und konjunktive Normalform
- 1.4 Vollständige Operationenmengen

## 2 Gatterlogik

- 2.1 Elementare (Logik-)Gatter, Schaltzeichen für Gatter
- 2.2 Zusammengesetzte Gatter: Schnittstelle und Implementierung
- 2.3 Bitmustererkennung, Datenflußsteuerung, Multiplexer und Dekodierer

## 3 Implementierung in Schaltnetzen

- 3.1 Elektrotechnische Grundlagen: Schalter und Transistoren
- 3.2 Gatter-Transistorschaltungen
- 3.3 Integrierte Schaltungen und ihre Herstellung
- 3.4 Evolution der Integrierten Schaltungen

# Schaltungstechnik: Gatter

- Boolesche Funktionen werden durch (**Logik-)Gatter** implementiert.
- Sie werden in Schaltplänen mit speziellen Zeichen statt der zugrundeliegenden Transistorschaltungen angegeben, um die Pläne zu vereinfachen.

	AND	OR	NOT	XOR	NAND	NOR	XNOR
IEC 60617-12							
US ANSI 91-1984							
DIN 40700							

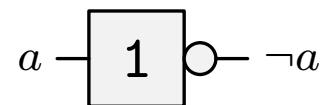
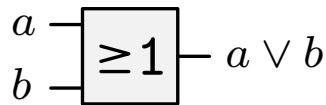
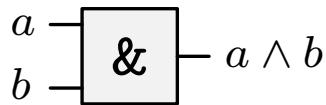
- Es gibt verschiedene Normen für Gatterzeichen.
- Im europäischen Raum wird meist die Norm IEC 60617-12 verwendet.
- In Schaltplänen findet man auch recht häufig die Norm US ANSI 91-1984.
- Die Darstellung nach DIN 40700 ist veraltet.

# Schaltungstechnik: Gatterlogik

**Boolesche Funktionen werden mit Hilfe der Gatterlogik implementiert.**

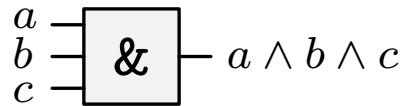
D.h., mehrere elementare Gatter werden zusammengeschaltet, um komplexere Boolesche Funktionen zu berechnen.

- Elementare Gatter, z.B.

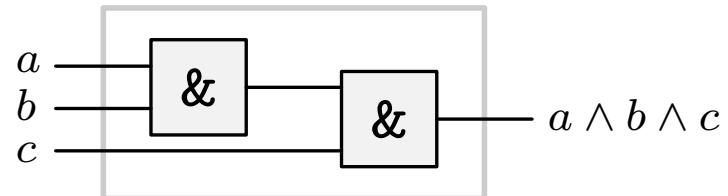


- Zusammengesetzte Gatter, z.B.

**Schnittstelle (interface)**



**Implementierung**

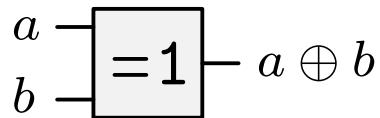


Wenn  $a = b = c = 1$ , dann Ausgabe = 1,  
sonst Ausgabe = 0.

- **Wichtig:** Schnittstelle (was?) im Gegensatz zu Implementierung (wie?).

# Schaltungstechnik: Gatterlogik

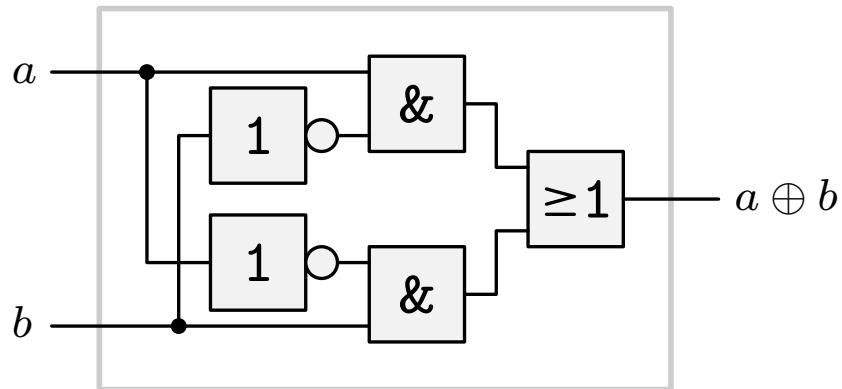
## Schnittstelle (interface)



## Implementierung der Schnittstelle

$$a \oplus b \equiv (a \wedge \neg b) \vee (\neg a \wedge b)$$

$a$	$b$	$a \oplus b$
0	0	0
1	0	1
0	1	1
1	1	0



- Jeder Baustein (Chip) und jedes Gatter hat eine abstrakte Schnittstelle, die angibt, **was** er/es tut, und eine Implementierung, die angibt, **wie** er/es dies tut.
- Hier gezeigt: Schnittstelle eines XOR-Gatters (Wahrheitstafel) und seine Implementierung durch zusammengeschaltete andere Gatter.

# Gatter: Vollständige Operationenmengen

$O_1 = \{| \}$  (NAND) ist vollständig:

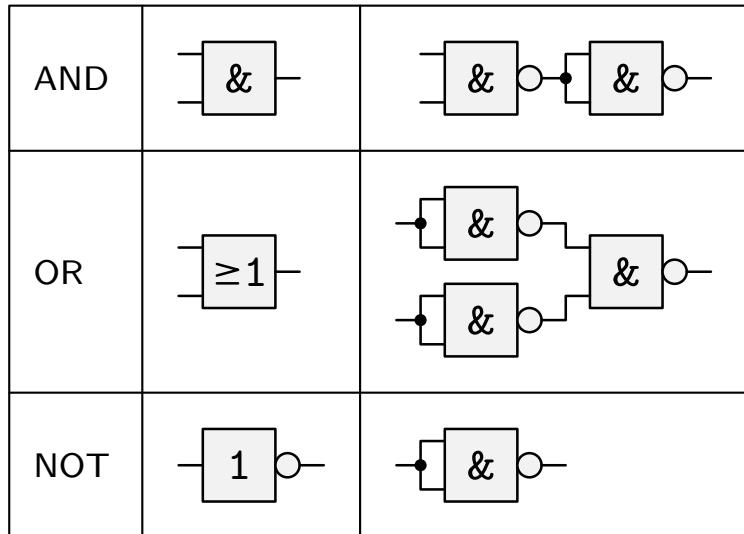
$$a \wedge b = (a | b) | (a | b)$$

$$a \vee b = (a | a) | (b | b)$$

$$\neg a = a | a$$

$O_2 = \{\downarrow\}$  (NOR) ist vollständig:

Übungsaufgabe, Blatt 2



- ⇒ Man kann ausschließlich NAND-Gatter oder ausschließlich NOR-Gatter als **elementare Gatter** verwenden.

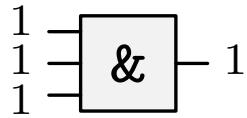
## Gatter: Erkennen von Bitmustern

- Die Funktion des AND-Gatters kann von 2 auf  $n$ ,  $n \geq 2$ , Eingangsgrößen verallgemeinert werden:

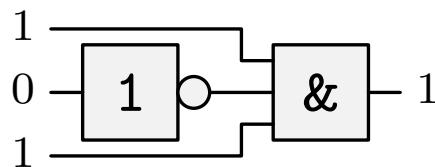
Nur wenn alle Eingangsgrößen 1 sind, ist der Ausgang 1, in allen anderen Fällen ist der Ausgang 0.

(Das Prinzip haben wir beim AND-Gatter mit drei Eingängen gesehen.)

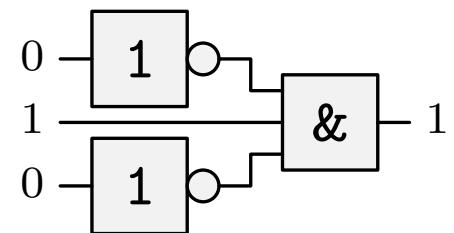
- Durch Hinzunahme von Invertern (NOT-Gatter), durch die Eingangsgrößen invertiert werden können, können AND-Gatter benutzt werden, um beliebige Bitmuster zu erkennen. (bit: *binary digit*)
- **Beispiele:** Erkenne die Bitmuster 111, 101, und 010.



Erkenne Muster 111.



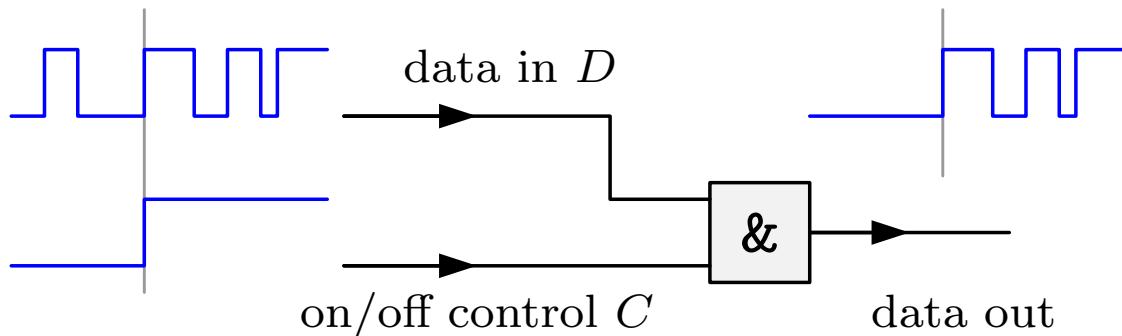
Erkenne Muster 101.



Erkenne Muster 010.

## Gatter: Datenflußsteuerung

- Nehmen wir an, ein **Datensignal**  $d$  läuft auf einer Datenleitung  $D$  ein.
- Nur wenn eine **Steuerleitung**  $C$  eine logische 1 anzeigt, soll das Datensignal  $d$  weitergeleitet werden, anderenfalls soll die Ausgabe auf einer logischen 0 gehalten werden.



- Die Wahrheitstafel des AND-Gatters kann auch so geschrieben werden:

Eingänge		Ausgang
$D$	$C$	$D \wedge C$
$d$	0	0
$d$	1	$d$

Diese Wahrheitstafel entspricht den beiden Axiomen der **Extremalität**  $a \wedge 0 = 0$  und der **Neutralität**  $a \wedge 1 = a$ .

## Gatter: Multiplexer (Auswahlschaltung)

- Um mit (Logik-)Gattern und Booleschen Funktionen vertrauter zu werden, schauen wir uns den Entwurf eines **Multiplexers** an, der als Verallgemeinerung der gerade betrachteten einfachen Datenflußsteuerung gesehen werden kann.
- Ein **n-Wege-Multiplexer** hat  $2 \times n$  Eingänge:  
 $n$  **Datenleitungen** und  $n$  **Steuerleitungen**
- Immer dann, wenn die  $i$ -te Steuerleitung,  $1 \leq i \leq n$ , auf 1 liegt, soll die  $i$ -te Datenleitung an den Ausgang weitergegeben werden.
- Beispiel: Wahrheitstafel eines 4-Wege-Multiplexers

Steuerung				Daten				Ausgang
$C_0$	$C_1$	$C_2$	$C_3$	$D_0$	$D_1$	$D_2$	$D_3$	$O$
1	0	0	0	$d_0$	$d_1$	$d_2$	$d_3$	$d_0$
0	1	0	0	$d_0$	$d_1$	$d_2$	$d_3$	$d_1$
0	0	1	0	$d_0$	$d_1$	$d_2$	$d_3$	$d_2$
0	0	0	1	$d_0$	$d_1$	$d_2$	$d_3$	$d_3$

## Gatter: Multiplexer (Auswahlschaltung)

Steuerung				Daten				Ausgang
$C_0$	$C_1$	$C_2$	$C_3$	$D_0$	$D_1$	$D_2$	$D_3$	$O$
1	0	0	0	$d_0$	$d_1$	$d_2$	$d_3$	$d_0$
0	1	0	0	$d_0$	$d_1$	$d_2$	$d_3$	$d_1$
0	0	1	0	$d_0$	$d_1$	$d_2$	$d_3$	$d_2$
0	0	0	1	$d_0$	$d_1$	$d_2$	$d_3$	$d_3$

- Man erinnere sich wieder an die Beziehungen

$$a \wedge 0 = 0 \quad \text{und} \quad a \vee 1 = 1 \quad (\mathbf{Extremalitat})$$

sowie

$$a \wedge 1 = a \quad \text{und} \quad a \vee 0 = a \quad (\mathbf{Neutralitat}),$$

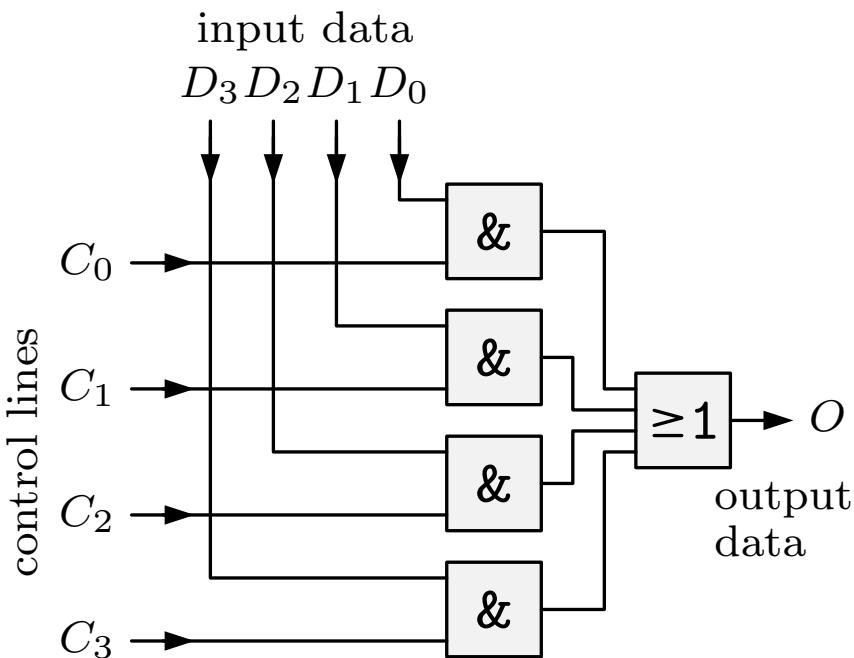
die zu den Axiomen der Booleschen Algebra gehoren. [Guiseppe Peano 1888]

- Eine Boolesche Formel, die einen 4-Wege-Multiplexer implementiert, ist daher:

$$O = (D_0 \wedge C_0) \vee (D_1 \wedge C_1) \vee (D_2 \wedge C_2) \vee (D_3 \wedge C_3).$$

# Gatter: Multiplexer (Auswahlschaltung)

Implementierung  
des 4-Wege-Multiplexers  
durch AND- und OR-Gatter.



**Beispiel:** Die Steuerleitungen seien  $C_0 = 0$ ,  $C_1 = 0$ ,  $C_2 = 1$ ,  $C_3 = 0$ , d.h., die Datenleitung  $D_2$  soll zum Ausgang durchgeschaltet werden:

$$\begin{aligned} O &= (D_0 \wedge C_0) \vee (D_1 \wedge C_1) \\ &\quad \vee (D_2 \wedge C_2) \vee (D_3 \wedge C_3) \\ &= (D_0 \wedge 0) \vee (D_1 \wedge 0) \\ &\quad \vee (D_2 \wedge 1) \vee (D_3 \wedge 0) \\ &= 0 \quad \vee \quad 0 \\ &\quad \vee \quad D_2 \quad \vee \quad 0 \\ &= D_2 \end{aligned}$$

Man beachte, wie Extremalität und Neutralität die gewünschte Funktion erzeugen.

## Gatter: 2-Bit-Dekodierer

- Man beachte, daß der Multiplexer nicht mehr korrekt funktioniert, wenn mehr als eine Steuerleitung eine logische 1 anzeigt.
- Man sollte besser sicherstellen, daß immer nur genau eine Steuerleitung eine logische 1 anzeigt, um stets korrektes Arbeiten zu garantieren.

Dies erreicht man mit Hilfe eines **Dekodierers**:

- Numeriere die  $n$  Steuerleitungen (hier:  $n = 4$ , also: 0, ..., 3).
- Wähle eine Steuerleitung über eine  $k$ -Bit-Eingabe aus ( $k = \lceil \log_2 n \rceil$ , hier:  $k = 2$ )  
 $\Rightarrow 2^k \geq n$  mögliche Steuerleitungen.
- Wenn die  $k$ -Bit-Eingabe den Wert  $i$  anzeigt, gibt die Bitfolge  $00 \dots \underbrace{1} \dots 00$  aus.  
 $i$ -tes Bit
- Gib diese Bitfolge an den Multiplexer weiter.

Die beiden Auswahlleitungen werden als Binärzahl aufgefaßt:  
 $00 \hat{=} 0$ ,  $01 \hat{=} 1$ ,  $10 \hat{=} 2$ ,  $11 \hat{=} 3$ .

Auswahl		Leitungen			
$S_1$	$S_0$	$C_0$	$C_1$	$C_2$	$C_3$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

## Gatter: 2-Bit-Dekodierer

- Um die ausgewählte Steuerleitung  $C_0, C_1, C_2$ , oder  $C_3$  zu bestimmen, müssen wir beide Auswahlleitungen,  $S_0$  und  $S_1$ , auswerten:

$$C_0 = \overline{S_1} \wedge \overline{S_0},$$

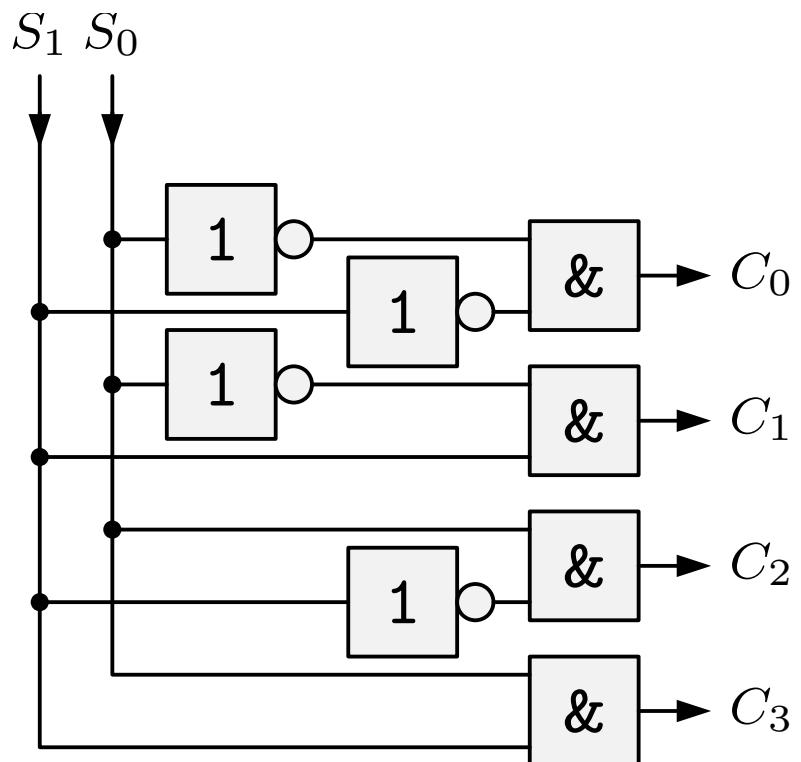
$$C_1 = \overline{S_1} \wedge S_0,$$

$$C_2 = S_1 \wedge \overline{S_0},$$

$$C_3 = S_1 \wedge S_0.$$

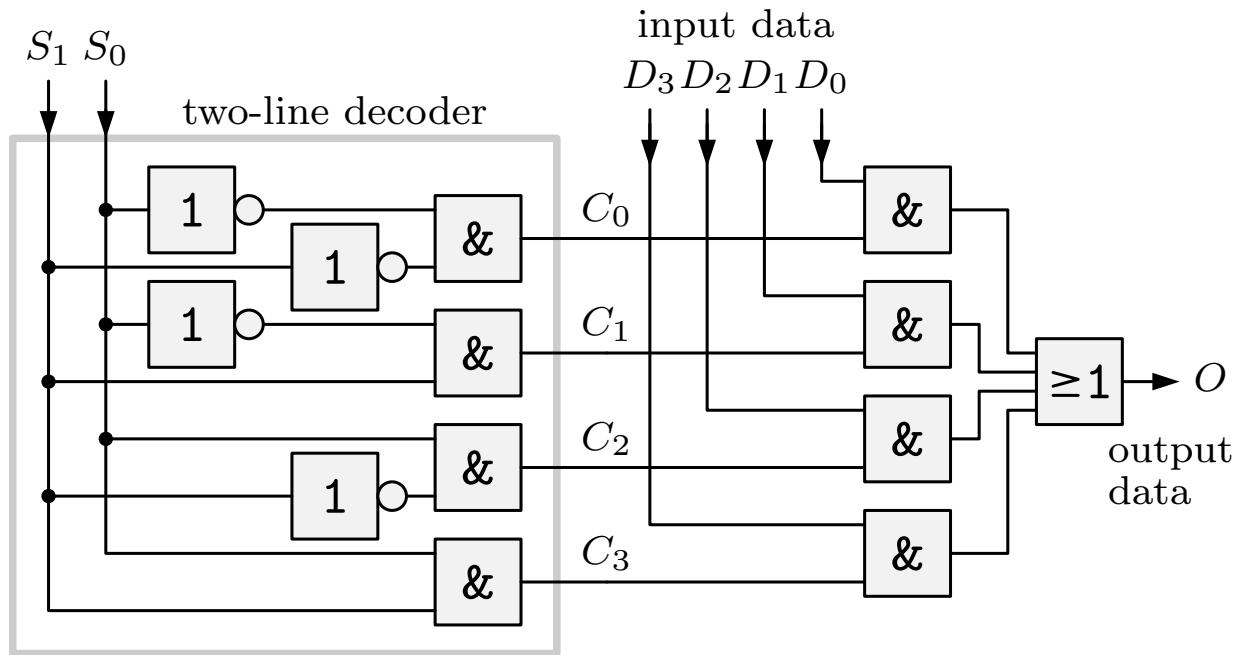
- Da unabhängig vom Zustand der Auswahlleitungen nur genau eine der Konjunktionen wahr sein kann, wird nur genau eine Steuerleitung ausgewählt.

Gatterimplementierung  
des 2-Bit-Dekodierers



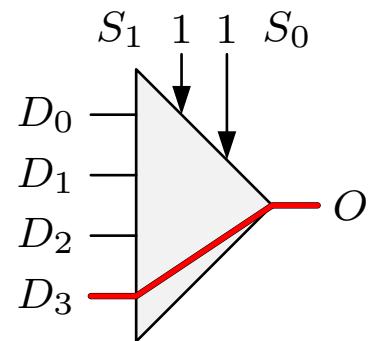
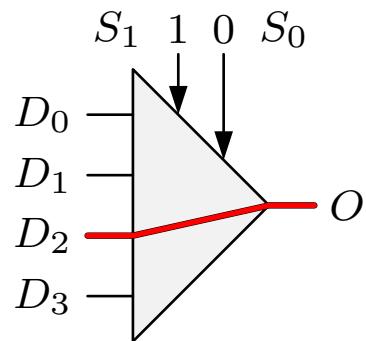
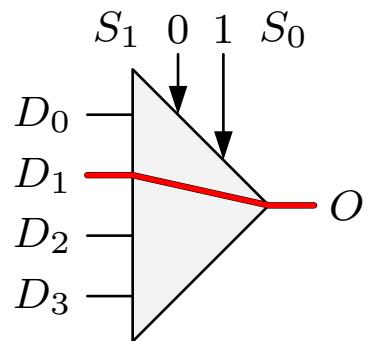
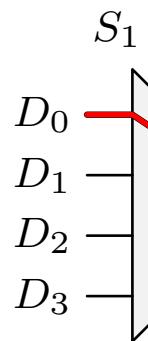
## Gatter: 2-Bit-Dekodierer & Multiplexer

Dekodierer und Multiplexer zu einer Schaltung zusammengesetzt:



Beachte:  
Es werden  
nur Gatter  
AND/OR/NOT  
verwendet.

Zwei Inverter  
können  
eingespart  
werden.



# Inhalt

## 1 Boolesche Algebra / Schaltalgebra

- 1.1 Definitionen, Operatoren, Schreibweisen
- 1.2 Boolesche Funktionen
- 1.3 Disjunktive und konjunktive Normalform
- 1.4 Vollständige Operationenmengen

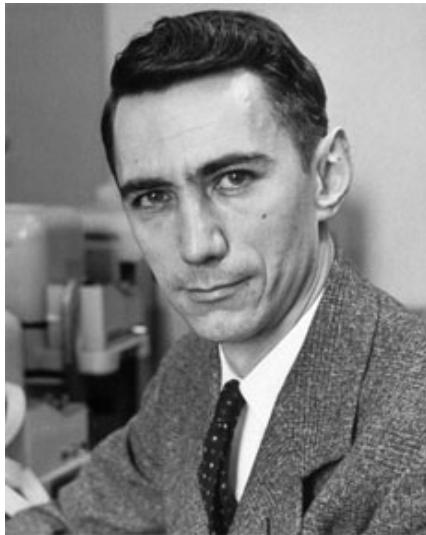
## 2 Gatterlogik

- 2.1 Elementare (Logik-)Gatter, Schaltzeichen für Gatter
- 2.2 Zusammengesetzte Gatter: Schnittstelle und Implementierung
- 2.3 Bitmustererkennung, Datenflußsteuerung, Multiplexer und Dekodierer

## 3 Implementierung in Schaltnetzen

- 3.1 Elektrotechnische Grundlagen: Schalter und Transistoren
- 3.2 Gatter-Transistorschaltungen
- 3.3 Integrierte Schaltungen und ihre Herstellung
- 3.4 Evolution der Integrierten Schaltungen

# Gatterlogik/Schaltalgebra: Implementierung



A Symbolic Analysis of Relay  
and Switching Circuits\*

Claude E. Shannon\*\*

## I. Introduction

In the control and protective circuits of complex electrical systems it is frequently necessary to make intricate interconnections of relay contacts and switches. Examples of these circuits occur in automatic telephone exchange and similar instruments. It is often desired to obtain any circuits designed to perform complex operations automatically.

Given certain characteristics, it is required to find a circuit incorporating these characteristics. The solution of this type of problem is usually a difficult and messy task. In this paper particular attention is given to the problem of finding the least number of relay contacts and switch blades will be studied. Methods will also be described for finding any number of circuits equivalent to a given circuit in all operating characteristics. It will be shown that several well-known theorems of impedance networks have roughly analogous theorems in relay circuits. Notable among these are the delta-wye and star-mesh transformations, and the duality theorem.

The method of attack on these problems may be described briefly as follows: any circuit is represented by a set of equations, the terms of the equations corresponding to the various relays and switches in the circuit. A calculus is developed for manipulating these equations by simple methods, the results of which are similar to those of algebraic algorithms. This calculus is shown to be exactly equivalent to the calculus of propositions, or the symbolic study of logic. For the synthesis problem the desired characteristics are first written as a system of equations, and the equations are then manipulated in the form representing the simplest circuit. The circuit may then be easily drawn from the equations. By this method it is always possible to find the simplest circuit containing no connections in series, and parallel connections, and in some cases the simplest circuit containing any type of connection.

Our notation is taken chiefly from symbolic logic. Of the many systems in common use we have chosen the one which seems simplest and most suggestive for our interpretation. Some of our phraseology, such as node, mesh, delta, wye, etc., is borrowed from ordinary network theory for simple concepts in switching circuits.

\*Transactions American Institute of Electrical Engineers, vol. 57, 1938. (Paper number 34-AM, recommended by the AIEE committee on communications and broadcasting, presented at the summer convention, Washington, D.C., June 20-24, 1938. Manuscript submitted March 1, 1938; made available for publication January 27, 1938.)

\*\*Claude E. Shannon is a research assistant in the department of electrical engineering at Massachusetts Institute of Technology, Cambridge. This paper is an abstract of a thesis presented at MIT for the degree of master of science. The author is indebted to Doctor F. L. Hitchcock, Doctor Vannevar Bush, and Doctor S. H. Caldwell, all of whom gave helpful encouragement and criticism.

471

Claude Elwood Shannon [1916–2001]

“A Symbolic Analysis of Relay  
and Switching Circuits”

Transactions American Institute of  
Electrical Engineers 57:471–495, 1938.  
(war Shannons Master’s Thesis)

- Shannon bewies in dieser Arbeit, daß die Boolesche Algebra benutzt werden kann, um die Anordnung der Relais in Telefonvermittlungsstellen zu vereinfachen und daß es möglich ist, mit Zusammenschaltungen von Relais Fragestellungen der Booleschen Algebra zu beantworten.
- Shannons Arbeit wurde zur Grundlage des Entwurfs von Schaltkreisen, als sie während und nach dem zweiten Weltkrieg weiteren Kreisen bekannt wurde.
- Die binären Eigenschaften elektrischer Schalter zur Berechnung logischer Funktionen zu nutzen, ist das Grundprinzip, das dem Entwurf aller digitalen elektronischen Rechner zugrundeliegt.

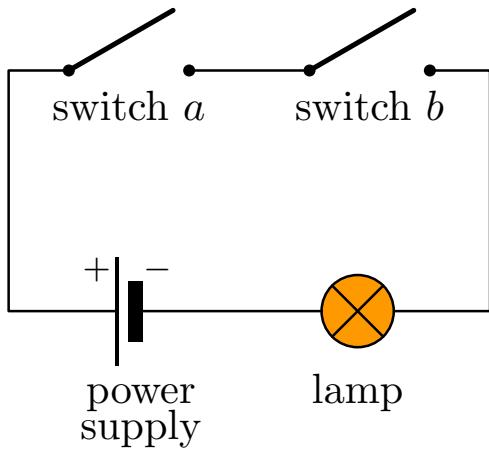
Quelle: Wikipedia

# Gatterlogik/Schaltalgebra: Implementierung

- Durch elektronische Bauteile, speziell (Feldeffekt-)Transistoren, werden (spannungsgesteuerte) **Schalter** dargestellt.
- Aus mehreren Schaltern werden (**Logik-**)**Gatter** zusammengesetzt, die einfache Boolesche Funktionen darstellen.
- Bestimmte (Logik-)Gatter lassen sich sehr leicht durch Transistoren darstellen, speziell NAND (negierte Konjunktion) und NOR (negierte Disjunktion).
- Mit diesen Gattern kann man alle möglichen Booleschen Funktionen darstellen (sie bilden jeweils eine **vollständige Operationenmenge**).
- Es gibt elektronische Bauteile, die diese grundlegenden Gatter implementieren (z.B. die Texas Instruments 74xx-Familie).
- In **integrierten Schaltkreisen** werden eine Vielzahl solcher Gatter zusammengeschaltet, um komplexere Funktionalität zu implementieren (z.B. Speicher, Prozessor etc.)

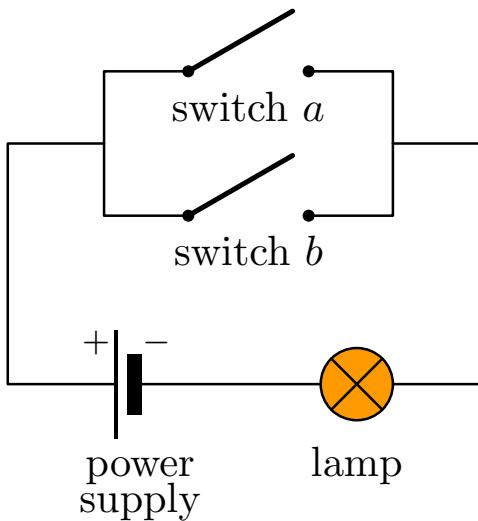
# Gatterlogik/Schaltalgebra: Implementierung

AND-Gatter



a	b	$a \wedge b$
0	0	0
1	0	0
0	1	0
1	1	1

OR-Gatter

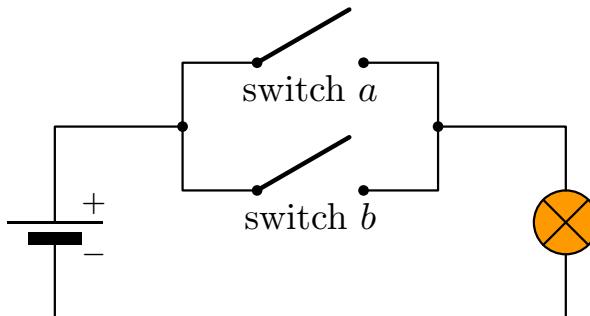


a	b	$a \vee b$
0	0	0
1	0	1
0	1	1
1	1	1

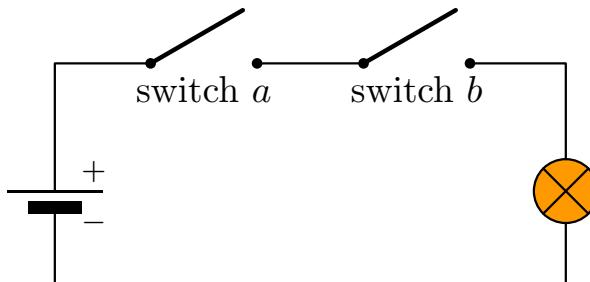
- Ein AND-Gatter (Konjunktion) kann durch zwei **hintereinandergeschaltete** Schalter implementiert werden:  
Strom fließt nur, wenn beide Schalter geschlossen sind.
- Ein OR-Gatter (Disjunktion) kann durch zwei **parallelgeschaltete** Schalter implementiert werden:  
Strom fließt, sobald einer der beiden Schalter geschlossen ist.

## Gatterlogik/Schaltalgebra: Implementierung

Durch Einfügen eines Widerstandes kann eine **Negation** implementiert werden.



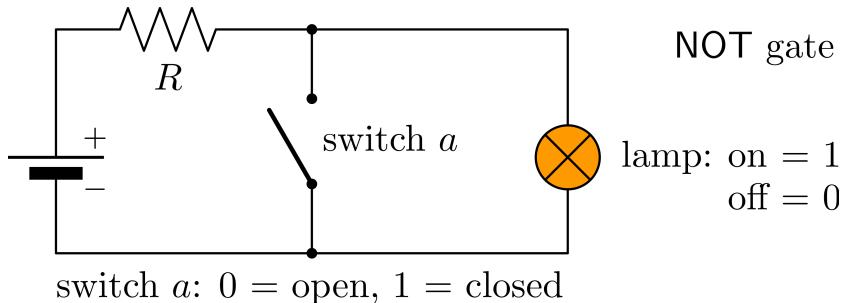
switch a: 0 = open, 1 = closed  
switch b: 0 = open, 1 = closed



switch a: 0 = open, 1 = closed  
switch b: 0 = open, 1 = closed

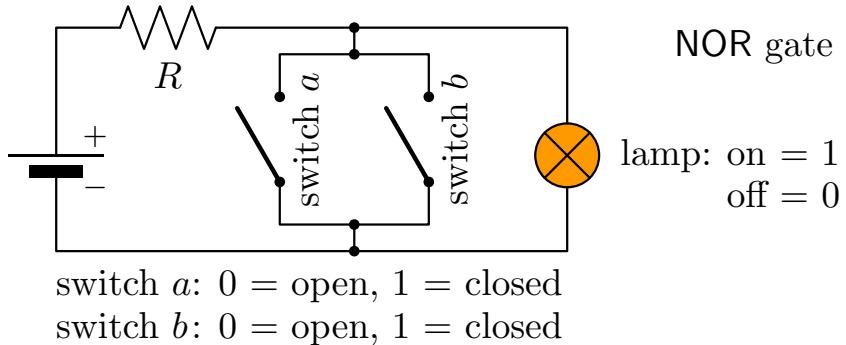
OR gate  
lamp: on = 1  
off = 0

AND gate  
lamp: on = 1  
off = 0



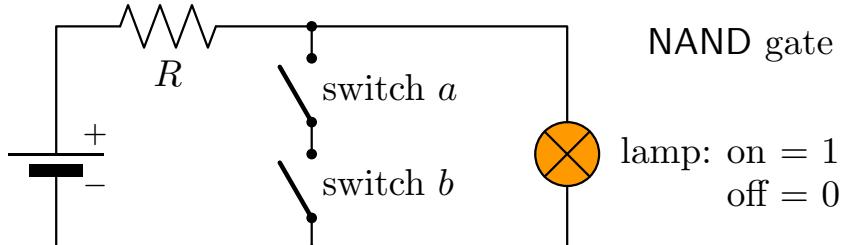
switch a: 0 = open, 1 = closed

NOT gate  
lamp: on = 1  
off = 0



switch a: 0 = open, 1 = closed  
switch b: 0 = open, 1 = closed

NOR gate  
lamp: on = 1  
off = 0



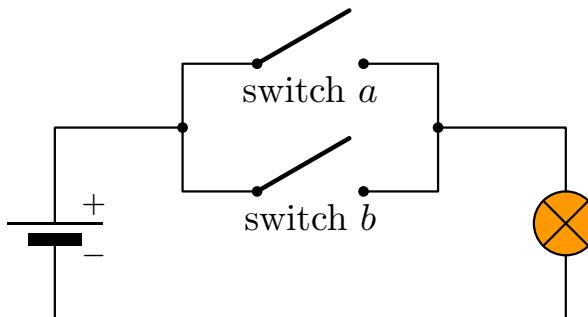
switch a: 0 = open, 1 = closed  
switch b: 0 = open, 1 = closed

NAND gate  
lamp: on = 1  
off = 0

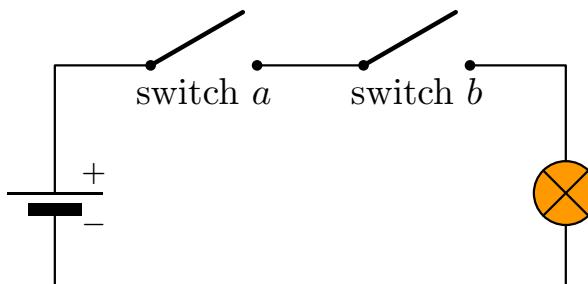
# Gatterlogik/Schaltalgebra: Implementierung

Durch Einfügen eines Widerstandes kann eine **Negation** implementiert werden.

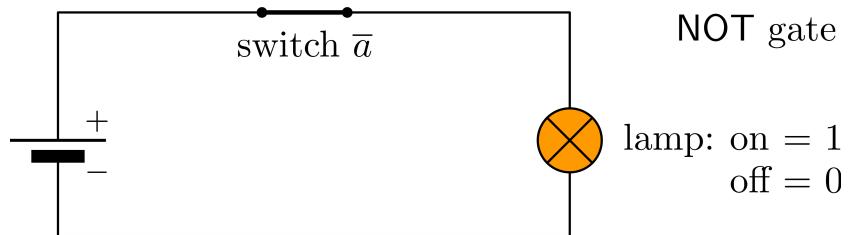
Bessere Lösung jedoch: komplementäre Schalter (Betätigen öffnet Schalter).



switch  $a$ : 0 = open, 1 = closed  
switch  $b$ : 0 = open, 1 = closed



switch  $a$ : 0 = open, 1 = closed  
switch  $b$ : 0 = open, 1 = closed



switch  $\bar{a}$

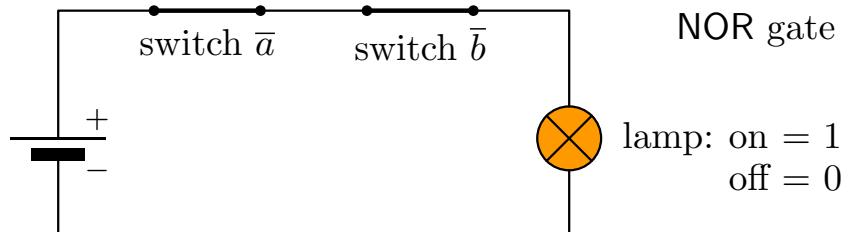
NOT gate

lamp: on = 1  
off = 0

switch  $\bar{a}$ : 0 = open, 1 = closed

OR gate

lamp: on = 1  
off = 0



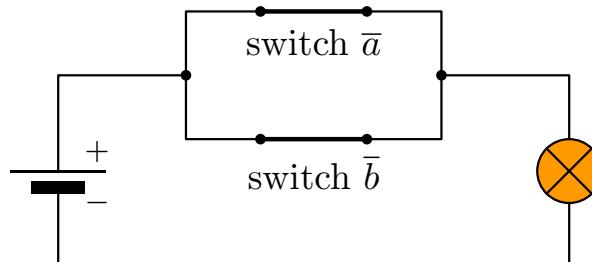
switch  $\bar{a}$ : 0 = closed, 1 = open  
switch  $\bar{b}$ : 0 = closed, 1 = open

NOR gate

lamp: on = 1  
off = 0

AND gate

lamp: on = 1  
off = 0



switch  $\bar{a}$ : 0 = closed, 1 = open  
switch  $\bar{b}$ : 0 = closed, 1 = open

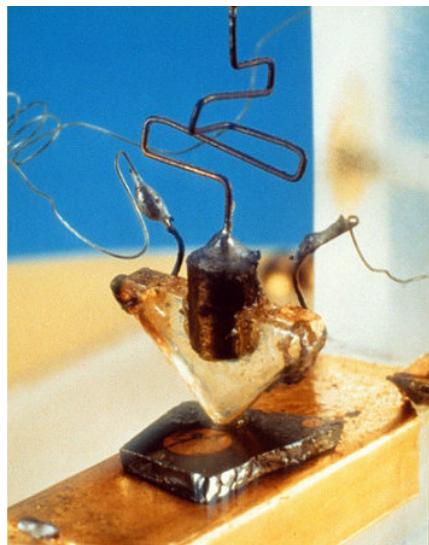
NAND gate

lamp: on = 1  
off = 0

# Elektrotechnische Grundlagen: Schalter

- Benötigt: **Schalter, der automatisch betätigt werden kann.**  
(vorzugsweise elektrisch steuerbar)
- Erste Lösung: **Relais** (elektromechanische/-magnetische Schalter)
  - Mechanisches Schließen eines Schalters durch einen Elektromagneten.
  - Relativ groß, relativ hoher Stromverbrauch.
  - Durch Mechanik störungsanfällig.
- Bessere Lösung: **Vakuumröhren** (Verstärkerröhren)
  - Elektrisches Schließen eines Schalters, zusätzliche Signalverstärkung.
  - Noch relativ groß, aber geringerer Stromverbrauch als Relais.
- Entscheidende Verbesserung: **Transistoren** (Halbleiterschalter/-verstärker)
  - Elektrisches Schließen eines Schalters, zusätzliche Signalverstärkung.
  - Sehr klein, sehr geringer Stromverbrauch (speziell bei Feldeffekttransistoren).

# Elektrotechnische Grundlagen: Transistoren



Sogenannter  
Bipolar-  
transistor:

Durch geringen  
Emitter-Basis-  
Strom wird  
Emitter-Collector-  
Strecke leitend.

Patented Oct. 3, 1950

2,524,035

## UNITED STATES PATENT OFFICE

2,524,035

### THREE-ELECTRODE CIRCUIT ELEMENT UTILIZING SEMICONDUCTIVE MATERIALS

John Bardeen, Summit, and Walter H. Brattain,  
Morristown, N. J., assignors to Bell Telephone  
Laboratories, Incorporated, New York, N. Y., a  
corporation of New York

Application June 17, 1948, Serial No. 33,466

40 Claims. (CL 179—171)

2

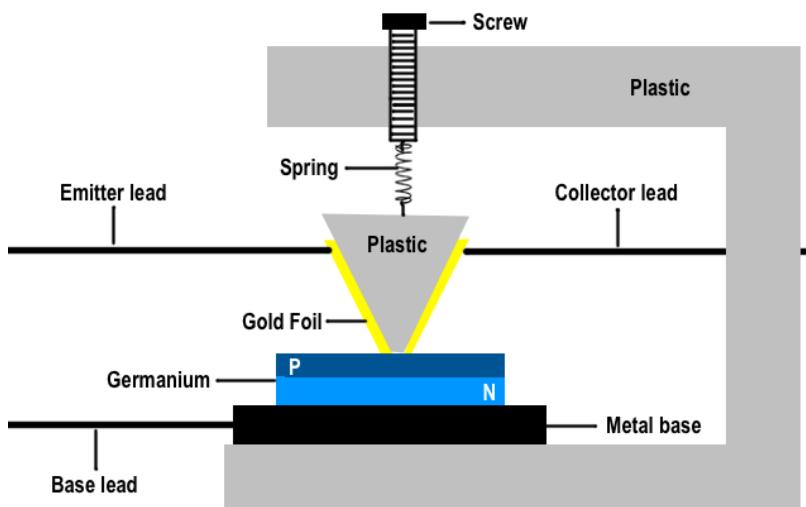
This application is a continuation-in-part of  
application Serial No. 11,165, filed February 26,  
1948, and thereafter abandoned.

This invention relates to a novel method of and  
means for translating electrical variations for  
such purposes as amplification, wave generation,  
and the like.

The principal object of the invention is to am-  
plify or otherwise translate electric signals or  
variations by use of compact, simple, and rugged  
apparatus of novel type.

Another object is to provide a circuit element  
for use as an amplifier or the like which does not  
require a heated thermionic cathode for its op-  
eration, and which therefore is immediately oper-

15 When a metal electrode is placed in contact  
with a semiconductor and a potential difference  
is applied across the junction, the magnitude of  
the current which flows often depends on the  
sign as well as on the magnitude of the poten-  
tial. A junction of this sort is called a rectify-  
ing contact. If the contact is made to an N-  
type semiconductor, the direction of easy cur-  
rent flow is that in which the semiconductor is neg-  
ative with respect to the electrode. With a  
P-type semiconductor, the direction of easy flow  
is that in which the semiconductor is positive.  
A similar rectifying contact exists at the bound-  
ary between two semiconductors of opposite con-  
ductivity types.



www.theregister.co.uk

## Nobel Prize in Physics 1956



William Bradford  
Shockley



John Bardeen



Walter Houser  
Brattain

www.nobelprize.org

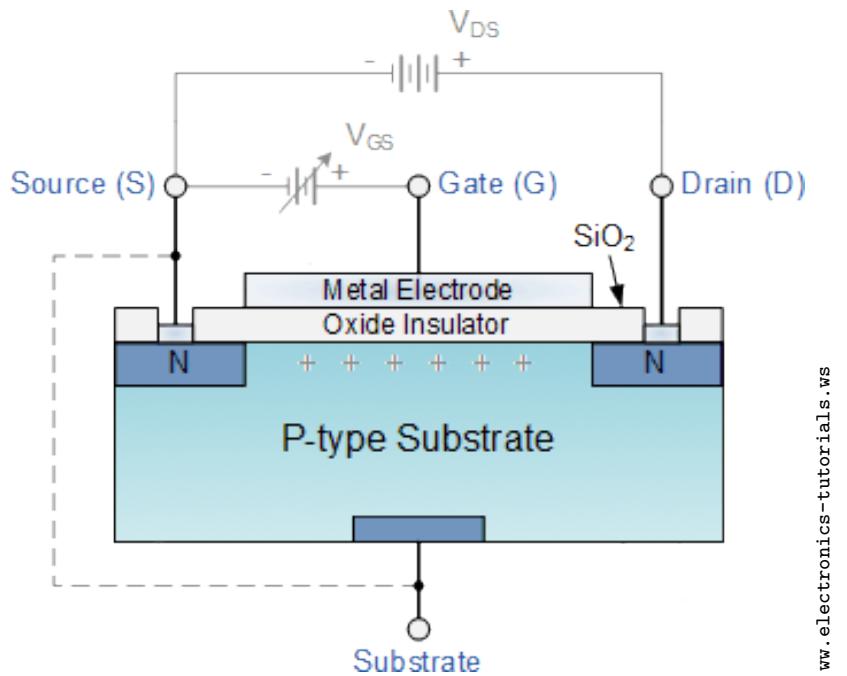
# Elektrotechnische Grundlagen: Transistoren

Man unterscheidet zwei grundsätzliche Arten von Transistoren:

Quelle: Wikipedia

- **Bipolartransistor** (bipolar junction transistor)
  - 3 Anschlüsse: Basis (base), Emitter (emitter), Kollektor (collector).
  - stromgesteuert: Ein kleiner Steuerstrom auf der Basis-Emitter-Strecke führt zu Veränderungen der Raumladung im Innern des Bipolartransistors und kann dadurch einen großen Strom auf der Kollektor-Emitter-Strecke steuern.
- **Feldeffekttransistor** (field effect transistor)
  - 3[4] Anschlüsse: Quelle (source), Senke (drain), Steuerelektrode (gate), [Substrat (substrate, bulk, body)]
  - spannungsgesteuert: Der Widerstand und somit der Strom der Drain-Source-Strecke wird durch die Gate-Source-Spannung und das dadurch entstehende elektrische Feld gesteuert. Die Steuerung ist im statischen Fall fast stromlos.
- Wir werden uns im folgenden auf Feldeffekttransistoren beschränken, da sie für die Rechnertechnik, speziell integrierte Schaltungen, wesentlich wichtiger sind.

# Elektrotechnische Grundlagen: Feldeffekttransistoren

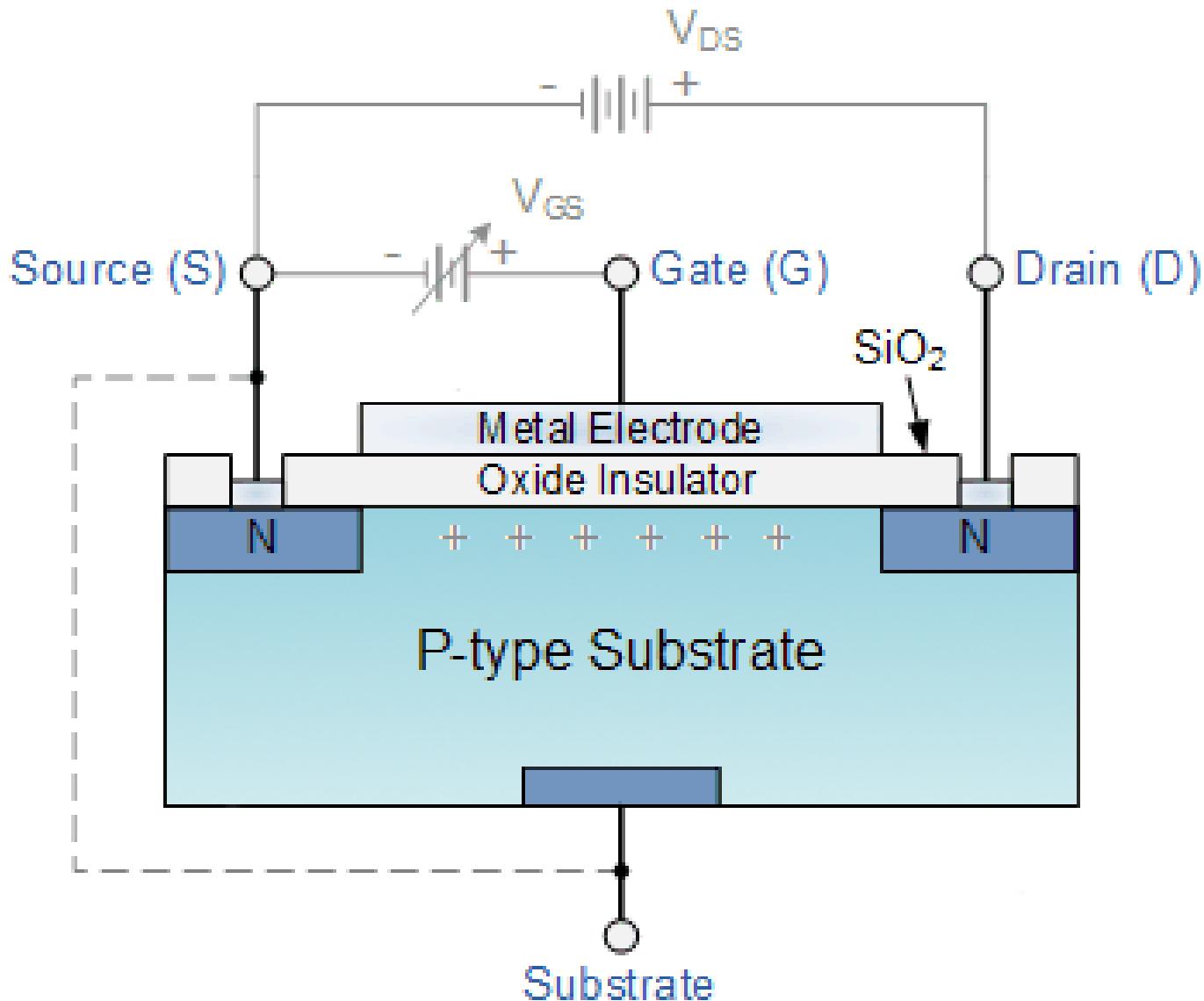


[www.electronics-tutorials.ws](http://www.electronics-tutorials.ws)

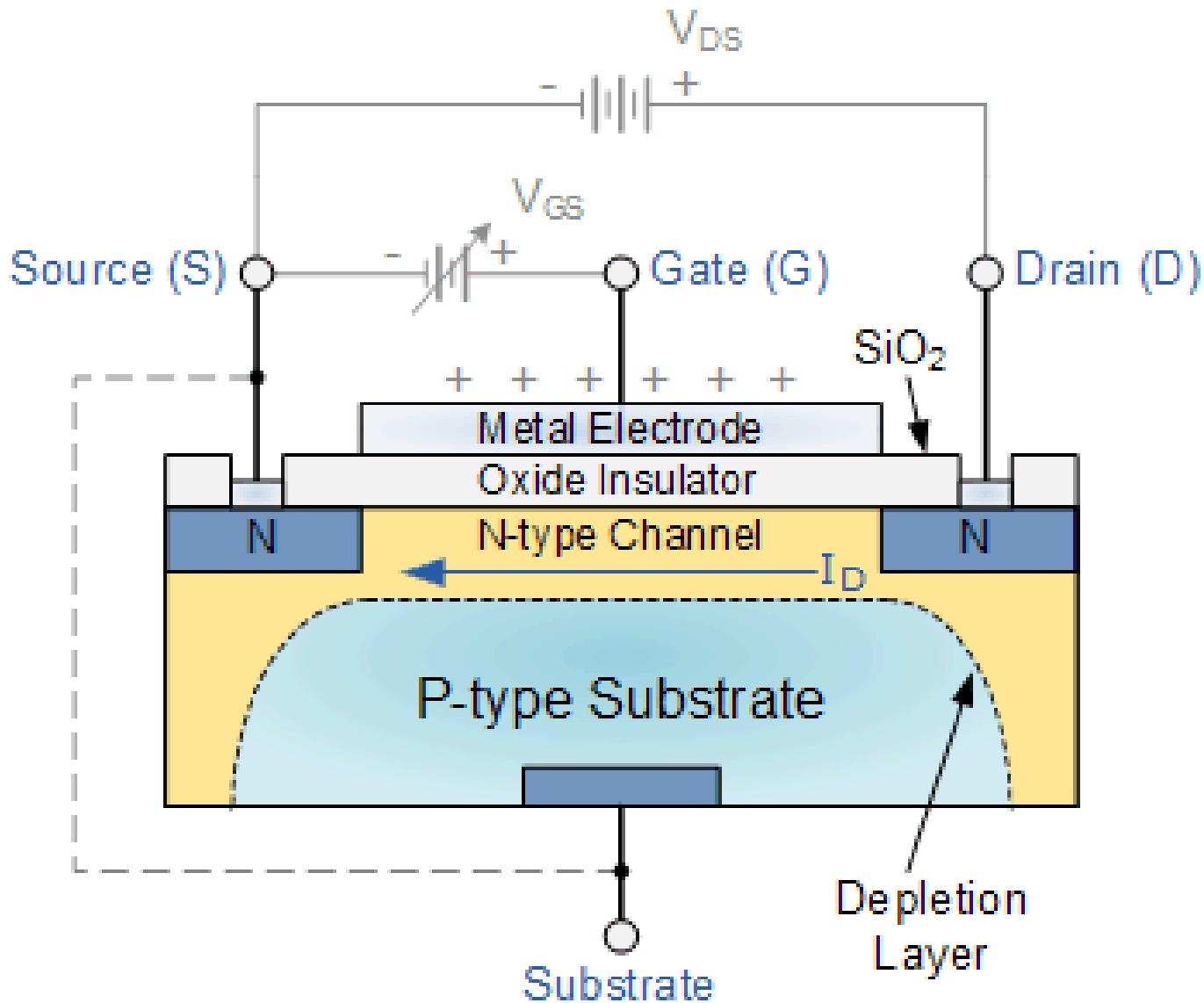
- Metal Oxide Semiconductor Field Effect Transistor (MOSFET)  
“Oxide” ist Siliziumdioxid, das jedoch bei starker Miniaturisierung elektrische Nachteile hat; daher ab ca. 2000 Ersatz durch andere Materialen.
- Metal Insulator Semiconductor Field Effect Transistor (MISFET)  
Insulated Gate  
Field Effect Transistor (IGFET)
- Abbildung links: N-Typ MOSFET

- Ein MOSFET verhält sich i.w. wie ein spannungsgesteuerter Widerstand:  
Durch die Spannung  $V_{GS}$  kann die Leitfähigkeit des Kanals zwischen der Quelle  $S$  (source) und der Senke  $D$  (drain) gesteuert werden.
- In der Digitaltechnik wird ein MOSFET i.w. als **Schalter** eingesetzt.

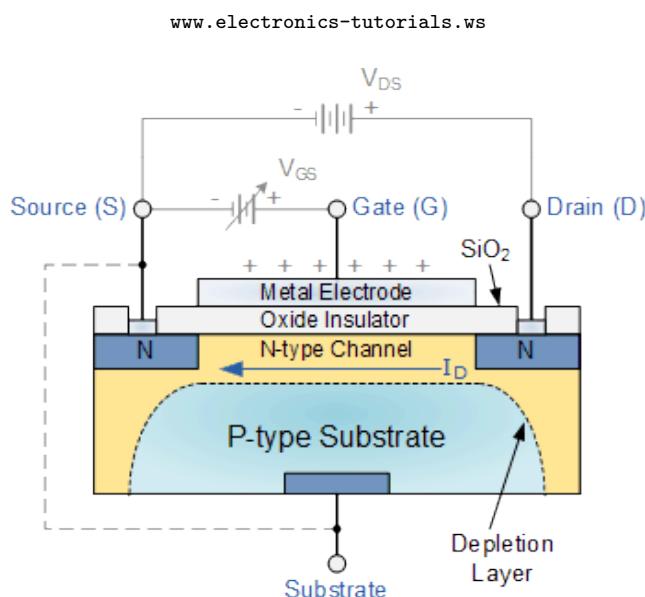
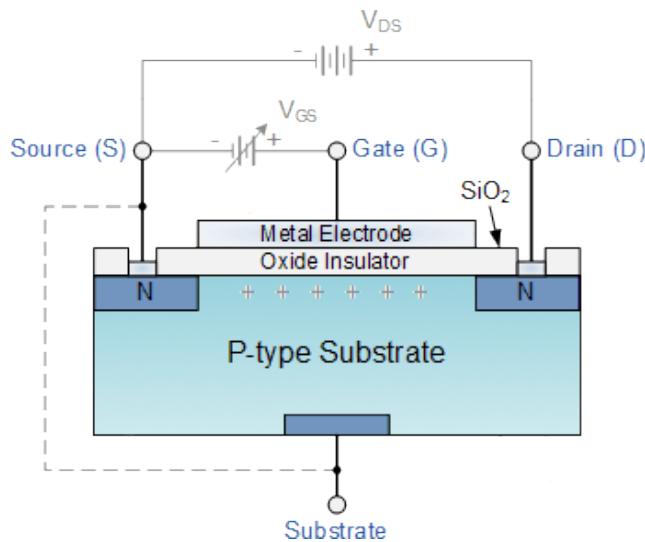
# Elektrotechnische Grundlagen: Feldeffekttransistoren



# Elektrotechnische Grundlagen: Feldeffekttransistoren



# Elektrotechnische Grundlagen: Feldeffekttransistoren



Einsatz eines MOSFET als **Schalter**  
(hier: N-Kanal-Anreicherungstyp):

- Liegt zwischen Steuerelektrode (gate) und Quelle (source) eine Spannung  $U_{GS}$ , die kleiner als die Schwellenspannung  $U_{th}$  ist, so ist der Widerstand  $R_{DS}$  des Kanals zwischen Quelle (source) und Senke (drain) sehr groß, da die positive Dotierung wie eine Barriere wirkt. Es fließt (fast) kein Strom  $I_D$ .
- Liegt zwischen Steuerelektrode (gate) und Quelle (source) eine Spannung  $U_{GS}$ , die mindestens so groß ist wie die Schwellenspannung  $U_{th}$ , so wird die positive Dotierung durch das elektrische Feld "zurueckgedraengt". Die Barriere verschwindet nach und nach, der Widerstand  $R_{DS}$  des Kanals zwischen Quelle (source) und Senke (drain) ist klein. Es fließt ein Strom  $I_D$ .
- Idee: Aus solchen Schaltern kann man **(Logik-)Gatter** zusammensetzen.

# Elektrotechnische Grundlagen: Transistoren

Man unterscheidet MOSFETs nach zwei Eigenschaften:  
der **Kanaleigenschaft** und der Schalterfunktionalität.

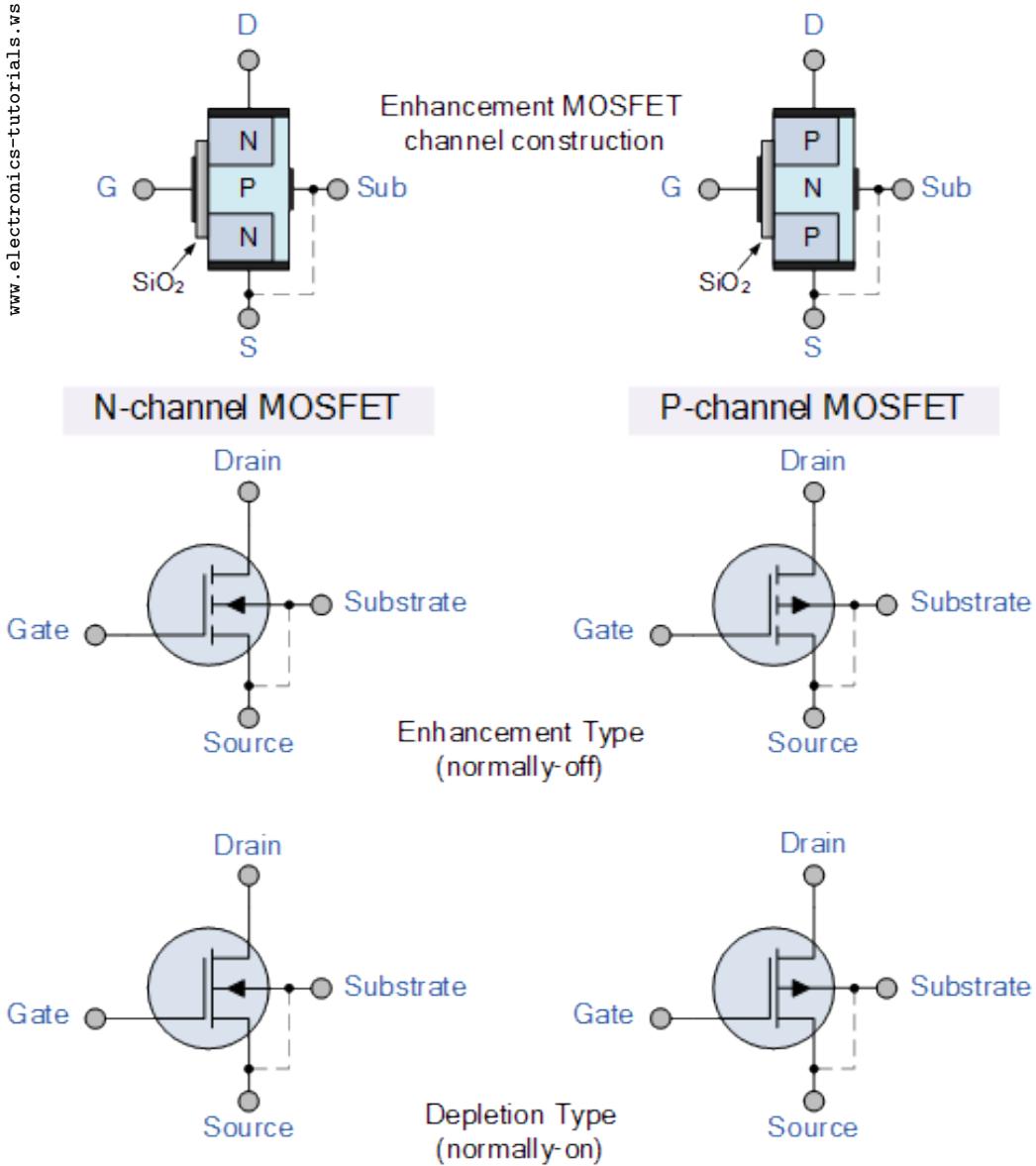
- **N-Typ** (auch N-Kanal, n-leitend oder NMOS):  
Das Substrat ist ein schwach p-dotierter Siliziumeinkristall,  
Quelle (source) und Senke (drain) sind n-dotiert.
- **P-Typ** (auch P-Kanal, p-leitend oder PMOS):  
Das Substrat ist ein schwach n-dotierter Siliziumeinkristall,  
Quelle (source) und Senke (drain) sind p-dotiert.
- Werden beide Typen gemeinsam verwendet, spricht man von CMOS  
(Complementary MOS).
- P-Dotierung: Einbringen von Fremdatomen, die als Elektronen-Akzeptoren dienen  
(z.B. Bor, Indium, Aluminium oder Gallium).
- N-Dotierung: Einbringen von Fremdatomen, die als Elektronen-Donatoren dienen  
(z.B. Phosphor, Arsen oder Antimon).

# Elektrotechnische Grundlagen: Transistoren

Man unterscheidet MOSFETs nach zwei Eigenschaften:  
der Kanaleigenschaft und der **Schalterfunktionalität**.

- **Anreicherungstyp (enhancement type):**
  - Wie ein Schalter, der normalerweise offen ist (**selbstsperrend**).
  - Der Kanal unter der Steuerelektrode (gate) wird durch keine oder nur geringfügige Dotierung fast nichtleitend gemacht.
  - Benötigt eine Spannung  $V_{GS}$  zwischen Gate und Source, um zu leiten.
- **Verarmungstyp (depletion type):**
  - Wie ein Schalter, der normalerweise geschlossen ist (**selbstleitend**).
  - Der Kanal unter der Steuerelektrode (gate) wird durch (hohe) Dotierung leitend gemacht.
  - Benötigt eine Spannung  $V_{GS}$  zwischen Gate und Source, um zu sperren.
- Meist werden nur Anreicherungstypen verwendet.

# Elektrotechnische Grundlagen: Transistoren

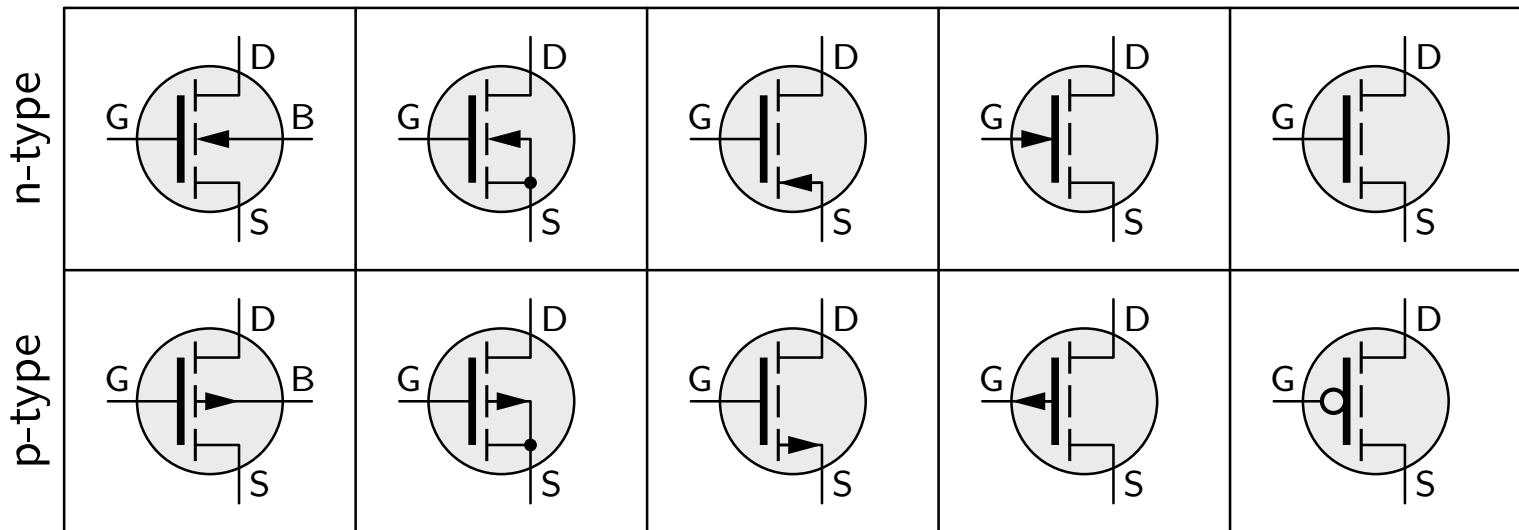


**Schaltzeichen** für die verschiedenen Transistorarten:

- **N-Typ:** Pfeil von Substrat zu Kanal
- **P-Typ:** Pfeil von Kanal zu Substrat
- **Anreicherungstyp:** Durchbrochene Linie für den Kanal
- **Verarmungstyp:** Durchgezogene Linie für den Kanal
- Der umschließende Kreis wird oft weggelassen.

# Elektrotechnische Grundlagen: Transistoren

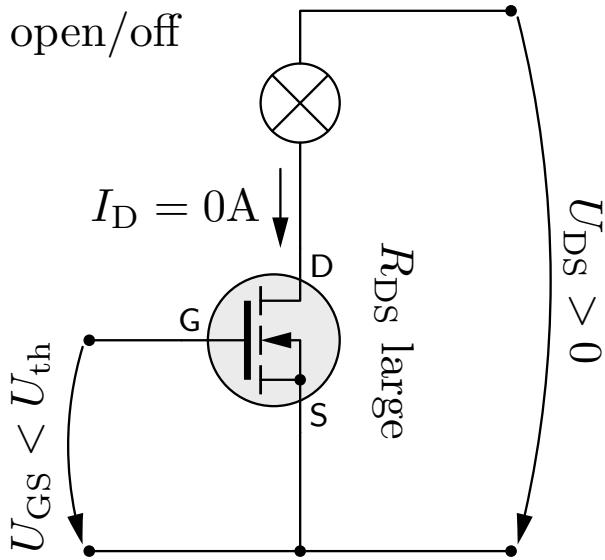
- Es sind auch noch andere Schaltzeichen für MOSFET-Transistoren verbreitet:



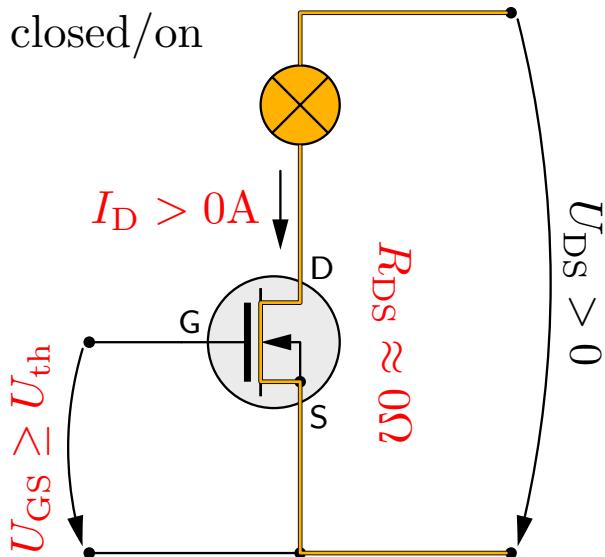
- Es bedeuten: G Steuerelektrode (gate),  
S Quelle (source),  
D Senke (drain),  
B Substrat (substrate, bulk, body).
- Wir werden, soweit möglich, die ganz links dargestellten verwenden,  
wobei meist die Quelle (source) mit dem Substrat verbunden ist (zweite Spalte).

# Elektrotechnische Grundlagen: Transistorschalter

open/off



closed/on

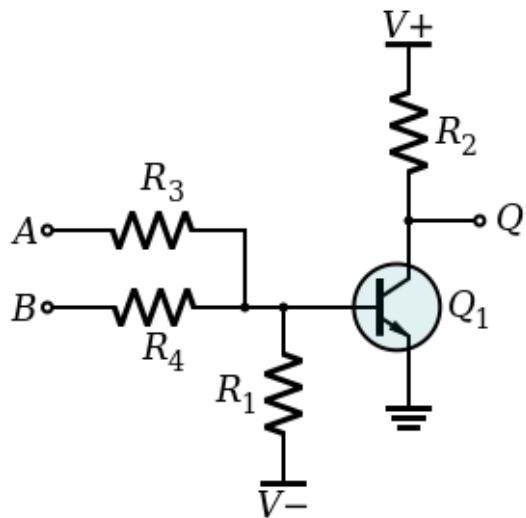


Einsatz eines MOSFET als **Schalter**  
(hier: N-Kanal-Anreicherungstyp):

- Liegt zwischen Steuerelektrode (gate) und Quelle (source) eine Spannung  $U_{GS}$ , die kleiner als die Schwellenspannung  $U_{th}$  ist, so ist der Widerstand  $R_{DS}$  des Kanals zwischen Quelle (source) und Senke (drain) sehr groß. Es fließt (fast) kein Strom  $I_D$ .
- Liegt zwischen Steuerelektrode (gate) und Quelle (source) eine Spannung  $U_{GS}$ , die mindestens so groß ist wie die Schwellenspannung  $U_{th}$ , so ist der Widerstand  $R_{DS}$  des Kanals zwischen Quelle (source) und Senke (drain) klein. Es fließt ein Strom  $I_D$ .
- Idee: Aus solchen Schaltern kann man (**Logik-Gatter**) zusammensetzen.

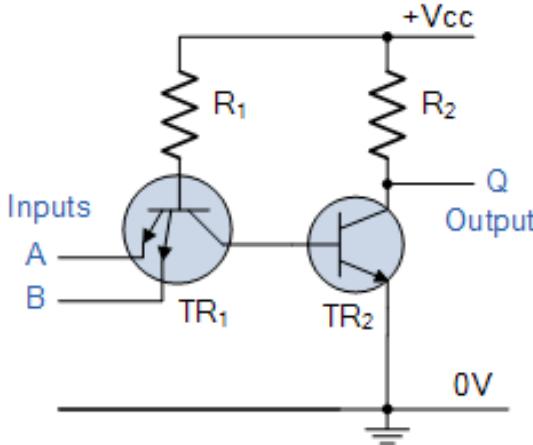
# Schaltungstechnik: Gatter-Transistorschaltungen

Resistor-Transistor Logic  
(RTL)

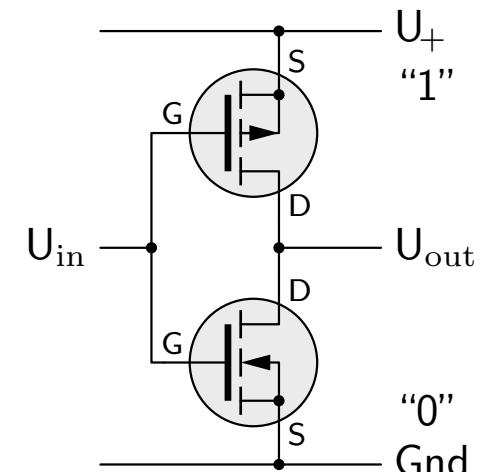


Transistor-Transistor Logic  
(TTL)

[www.electronics-tutorials.ws](http://www.electronics-tutorials.ws)



CMOS Logic  
(CMOS)



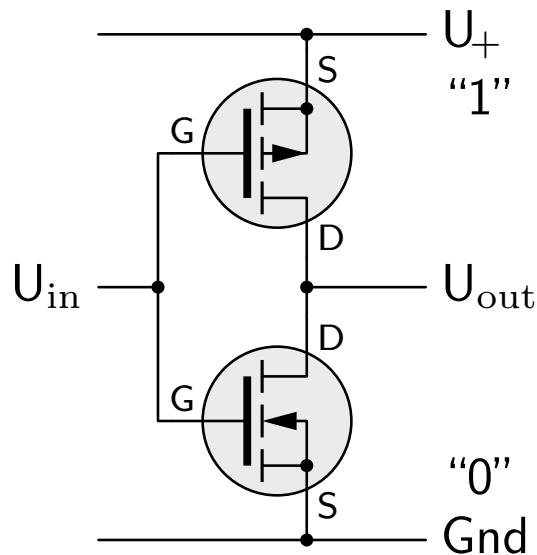
NOR-Gatter

NAND-Gatter

NOT-Gatter

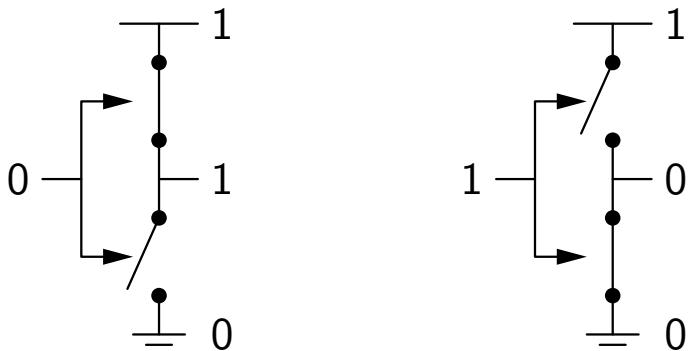
- Sowohl in der Resistor-Transistor als auch in der Transistor-Transistor Logic werden Widerstände eingesetzt, auch (aber nicht nur) um Negationen darzustellen.
- CMOS Logic verwendet rein komplementäre Schalter. Dadurch fließt nur bei Zustandsänderung ein Strom.

# Schaltungstechnik: Gatter-Transistorschaltungen

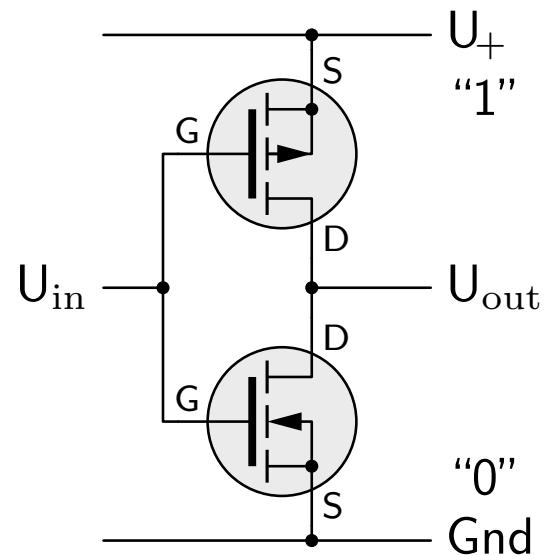


Komplementäre Transistorarten, N-Typ (unten) und P-Typ (oben), wegen des unterschiedlichen Vorzeichen der Potentialdifferenz.

- Wenn das Potential von  $U_{in}$  gering ist (kleine Spannung zwischen  $U_{in}$  und Masse, große Spannung zwischen der Versorgungsspannung  $U_+$  und  $U_{in}$ ), dann leitet der obere Transistor, während der untere Transistor sperrt.  
geringes Potential  $U_{in}$  ("0")  
→ hohes Potential  $U_{out}$  ("1")
- Wenn das Potential von  $U_{in}$  hoch ist (große Spannung zwischen  $U_{in}$  und Masse, kleine Spannung zwischen der Versorgungsspannung  $U_+$  und  $U_{in}$ ), dann sperrt der obere Transistor, während der untere Transistor leitet.  
hohes Potential  $U_{in}$  ("1")  
→ geringes Potential  $U_{out}$  ("0")



# Schaltungstechnik: Gatter-Transistorschaltungen



NOT-Gatter/Inverter  
in CMOS-Technik

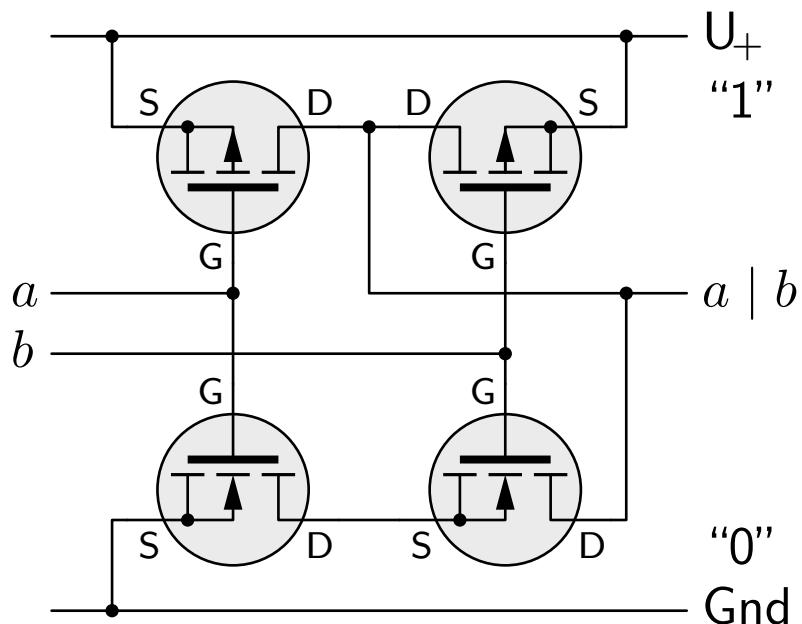
Durch seine analogen Eigenschaften  
ist ein Inverter auch ein **Verstärker**.

Inverter werden daher auch zur  
**Signalaufrischung** und zur  
**Schaltzeitverringerung** benutzt.

Komplementäre Transistorarten, N-Typ (unten) und P-Typ (oben), wegen des unterschiedlichen Vorzeichens der Potentialdifferenz.

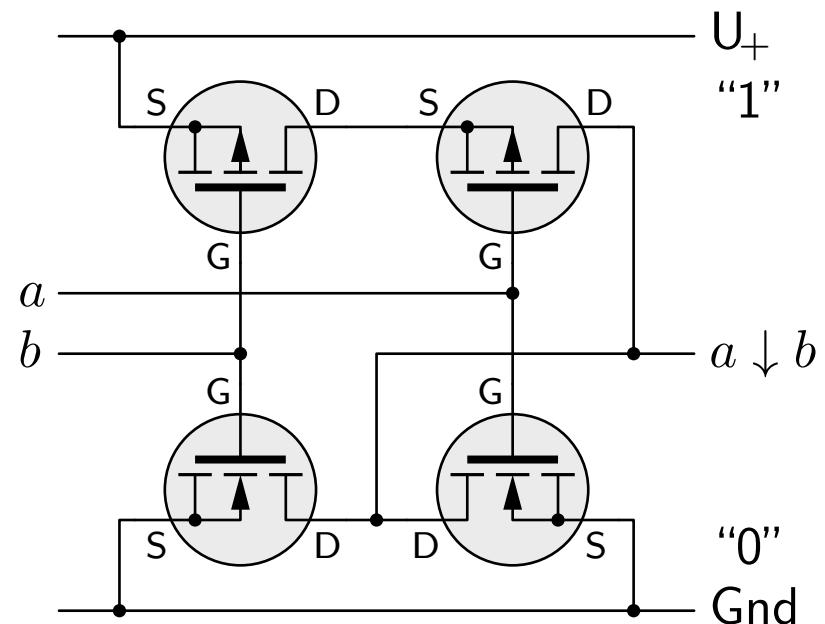
- Wenn das Potential von  $U_{in}$  gering ist (kleine Spannung zwischen  $U_{in}$  und Masse, große Spannung zwischen der Versorgungsspannung  $U_+$  und  $U_{in}$ ), dann leitet der obere Transistor, während der untere Transistor sperrt.  
geringes Potential  $U_{in}$  ("0")  
→ hohes Potential  $U_{out}$  ("1")
- Wenn das Potential von  $U_{in}$  hoch ist (große Spannung zwischen  $U_{in}$  und Masse, kleine Spannung zwischen der Versorgungsspannung  $U_+$  und  $U_{in}$ ), dann sperrt der obere Transistor, während der untere Transistor leitet.  
hohes Potential  $U_{in}$  ("1")  
→ geringes Potential  $U_{out}$  ("0")

# Schaltungstechnik: Gatter-Transistorsschaltungen



NAND-Gatter in CMOS-Technik

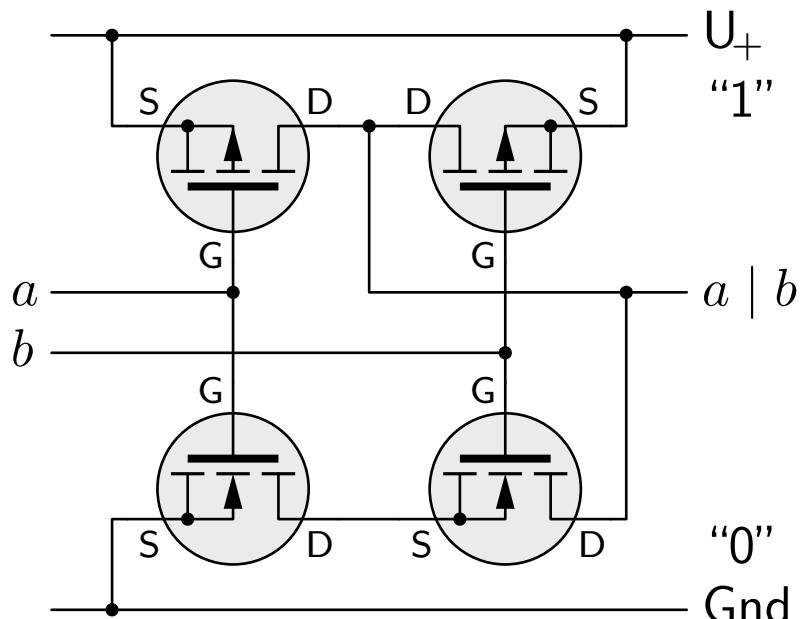
$a$	$b$	$a \downarrow b$
0	0	1
1	0	1
0	1	1
1	1	0



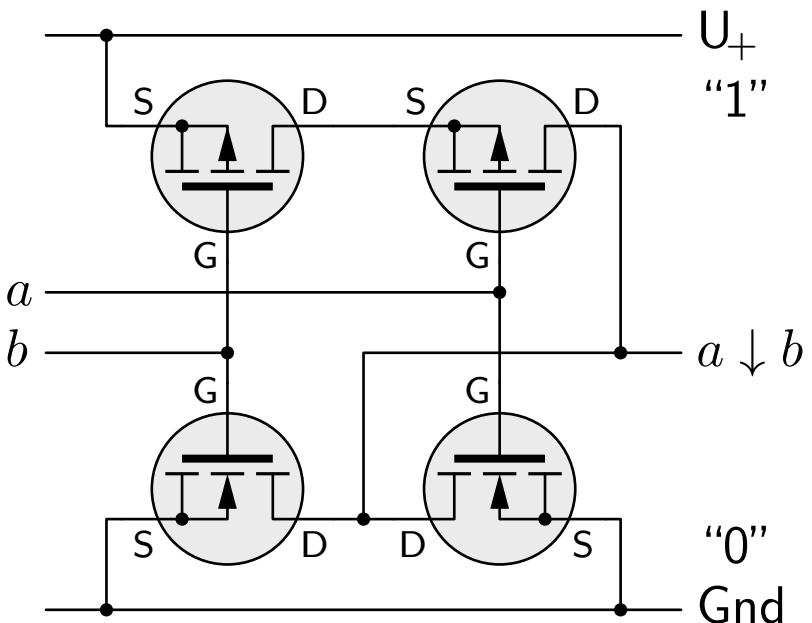
NOR-Gatter in CMOS-Technik

$a$	$b$	$a \downarrow b$
0	0	1
1	0	0
0	1	0
1	1	0

## Schaltungstechnik: Gatter-Transistorschaltungen



NAND-Gatter in CMOS-Technik

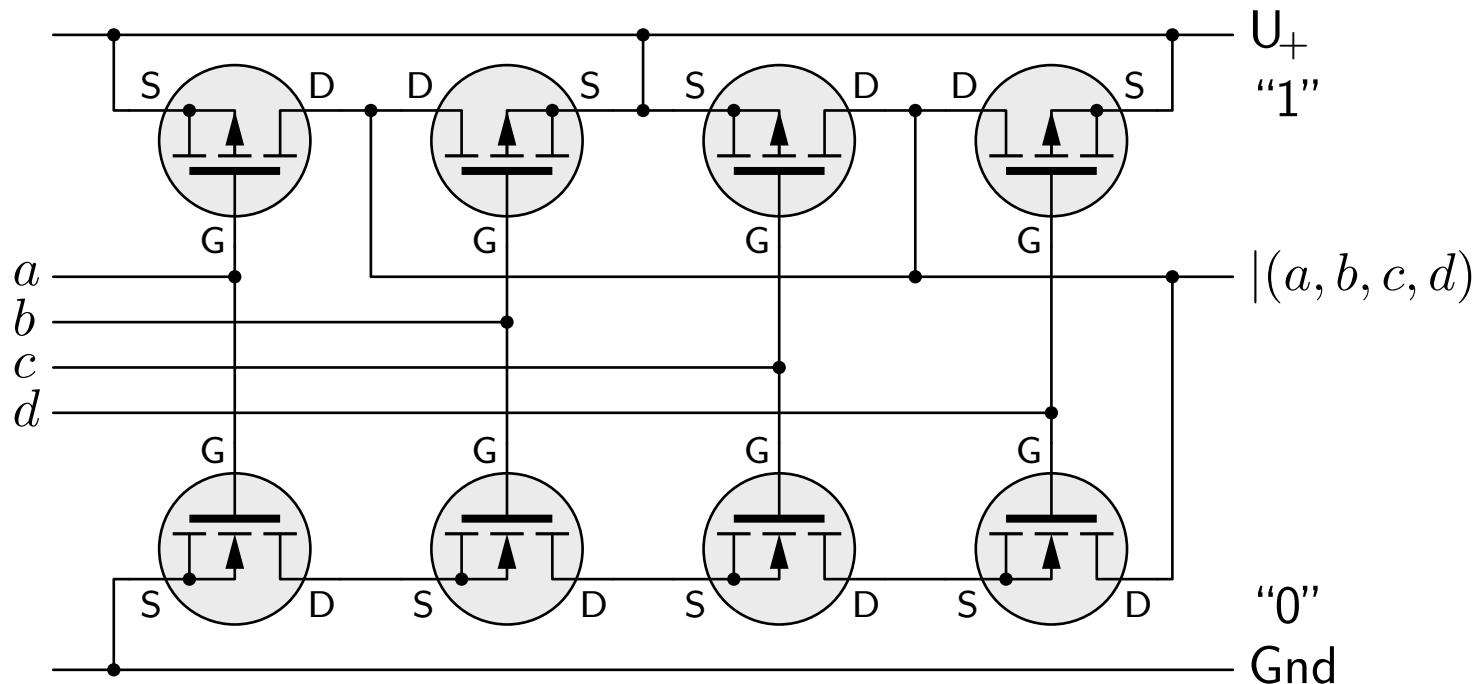


NOR-Gatter in CMOS-Technik

Man beachte die Dualität der Schaltkreise:

Durch Vertauschen der "1" (Versorgungsspannung) und der "0" (Masse) sowie Vertauschen der Transistorarten (N-Typ, unten, gegen P-Typ, oben) und umgekehrt gehen die beiden Schaltungen ineinander über.

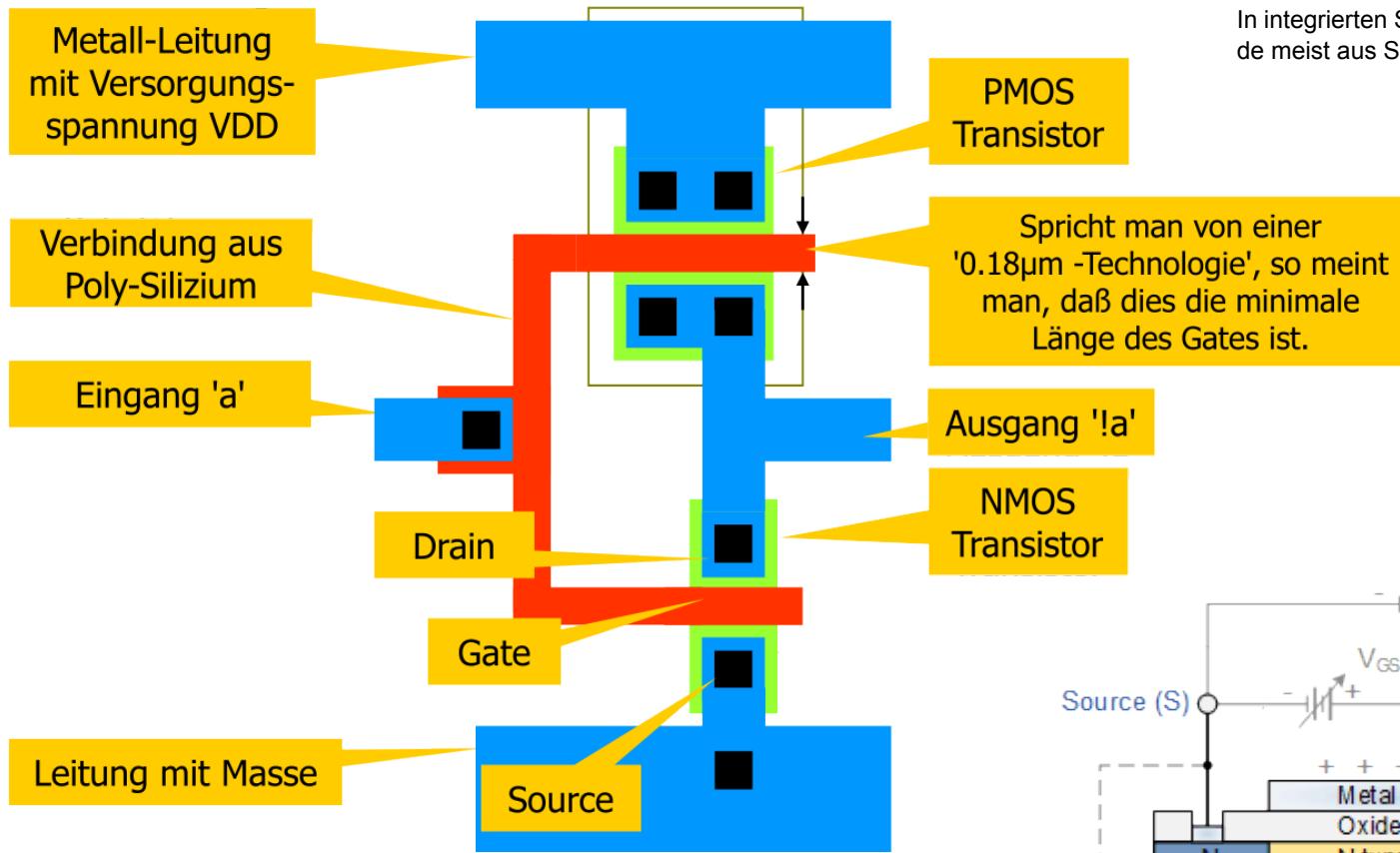
# Schaltungstechnik: Gatter-Transistorschaltungen



NAND-Gatter mit 4 Eingängen in CMOS-Technik

Derartige Transistorschaltungen für NAND- und NOR-Gatter können nicht nur zwei, sondern im Prinzip beliebig viele Eingänge haben. Von dieser Möglichkeit wird z.B. in Logikarrays Gebrauch gemacht (später).

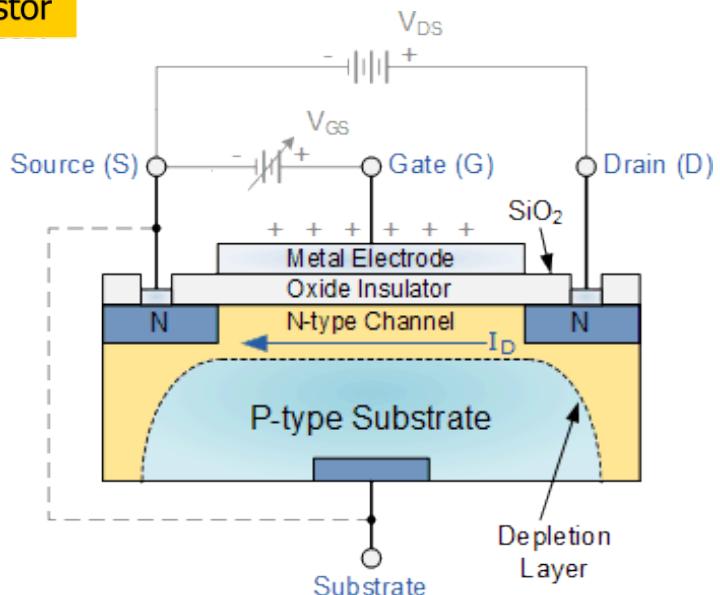
# Integrierte Schaltungen: Layout



In integrierten Schaltungen ist die Steuerelektrode meist aus Silizium statt aus Metall.

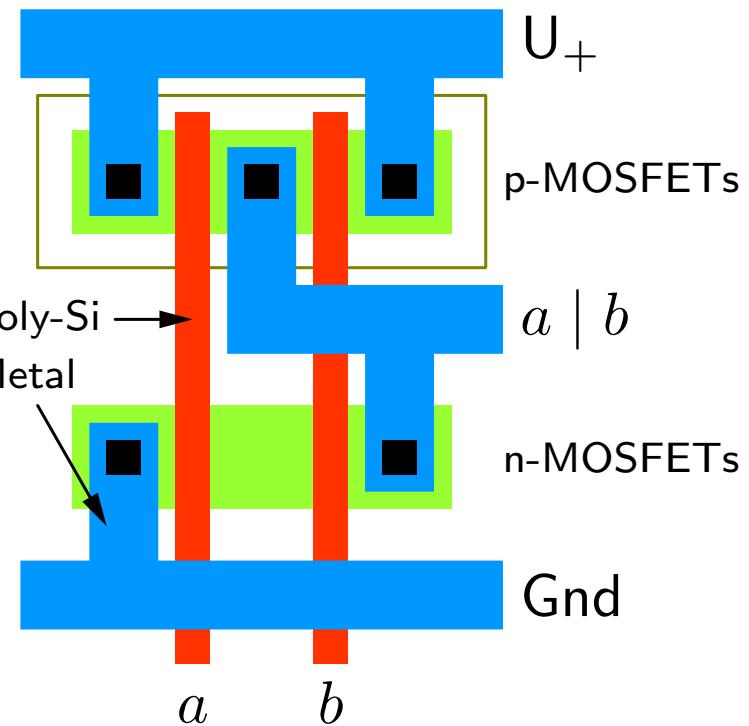
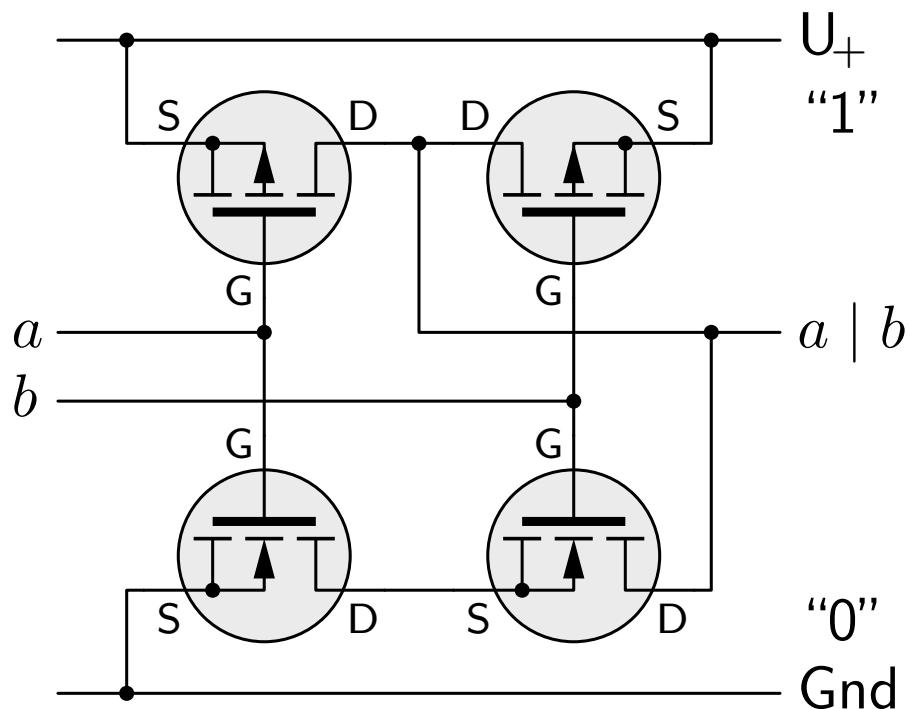
P. Fischer, Universität Heidelberg

[www.electronics-tutorials.ws](http://www.electronics-tutorials.ws)



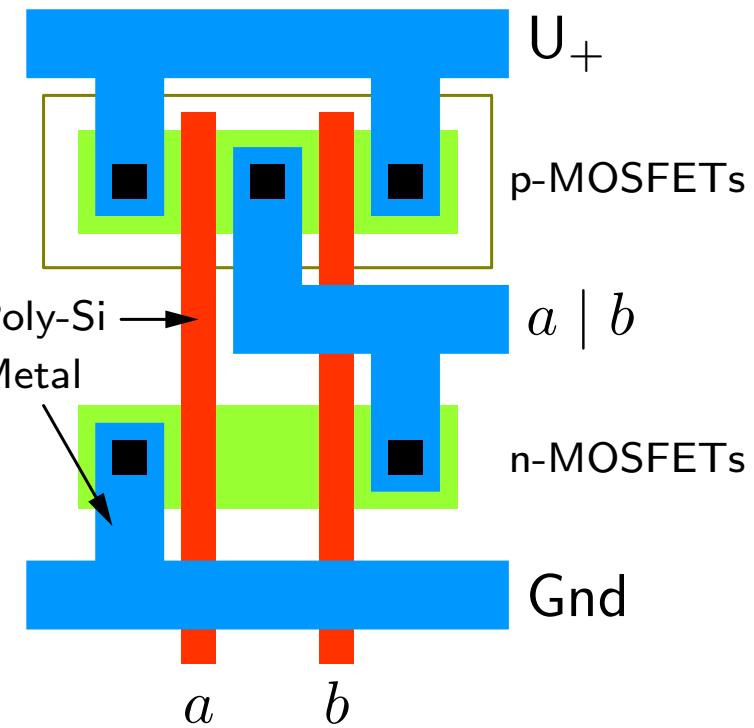
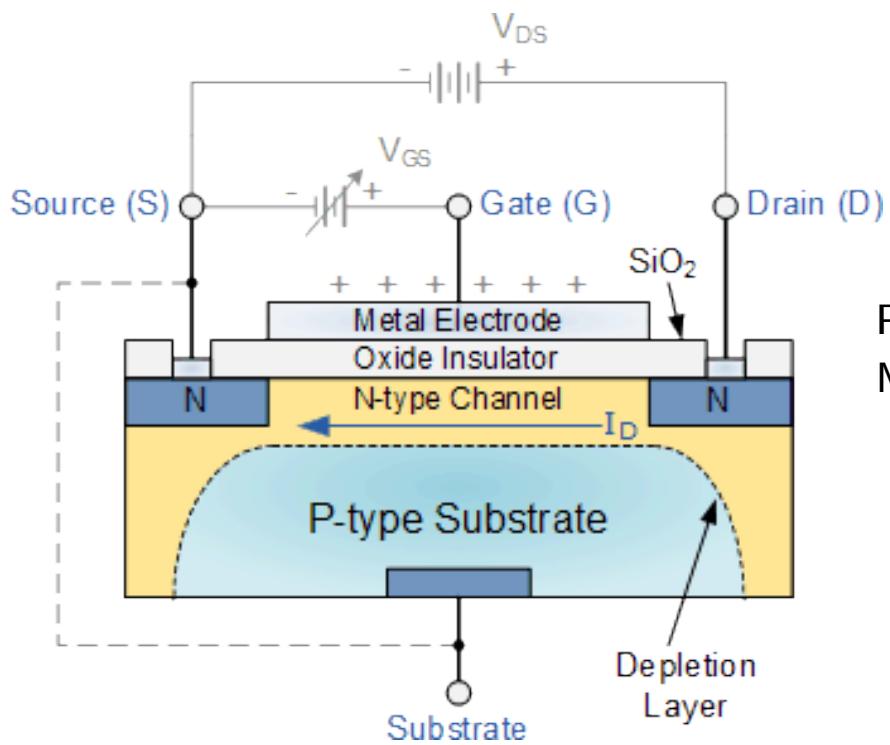
VLSI-Layout (Very Large Scale Integration)  
eines NOT-Gatters/Inverters in CMOS-Technik.

## Integrierte Schaltungen: Layout



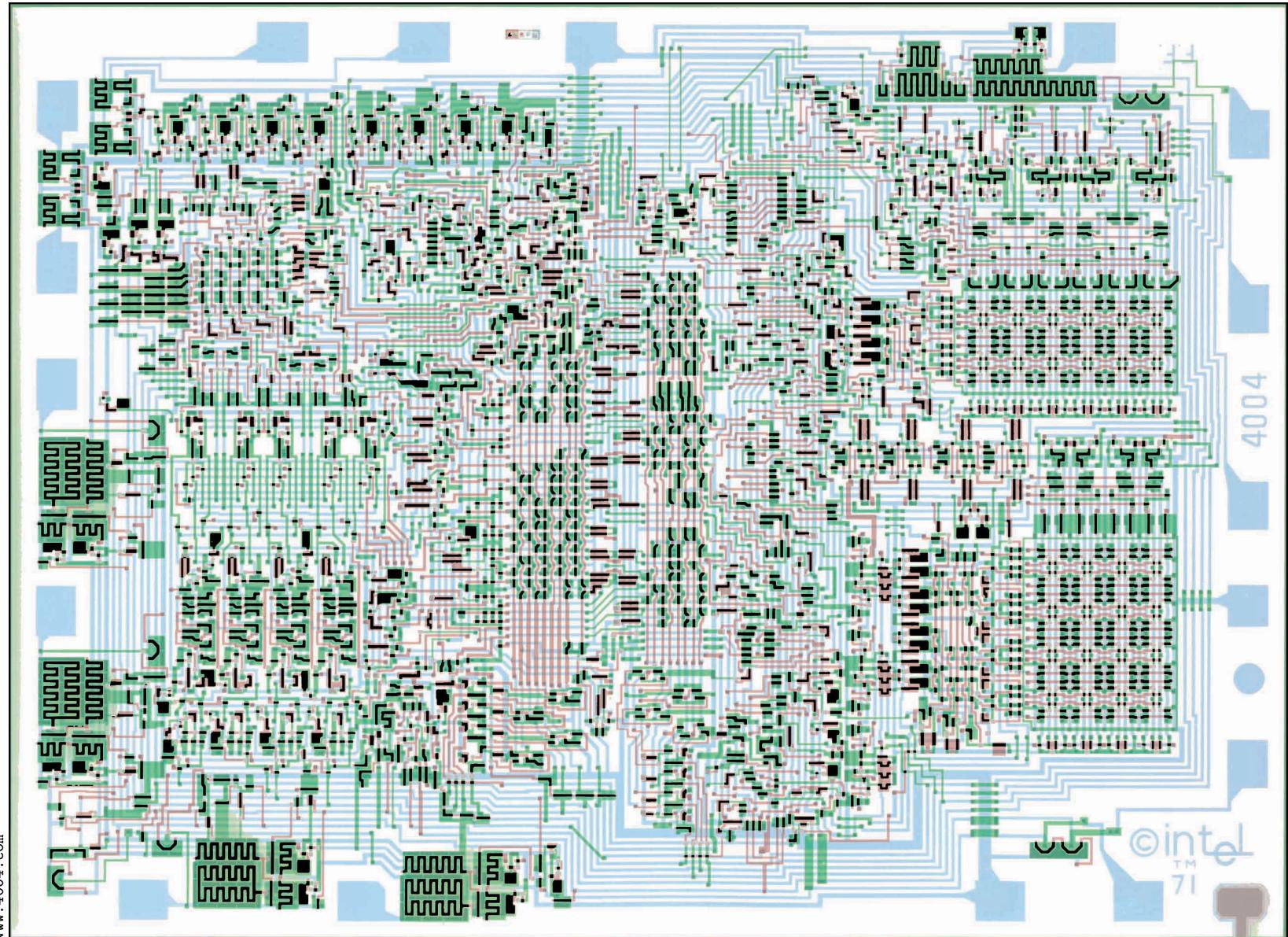
- Layout eines NAND-Gatters in CMOS-Technik als integrierter Schaltkreis.
- Unten: N-Typ-Transistoren, Serienschaltung  
Oben: P-Typ-Transistoren, Parallelenschaltung
- Rot: Steuerelektroden (gates), Blau: Verbindungsleitungen aus Metall
- Schwarz: Verbindungen zwischen Metall und Silizium

# Integrierte Schaltungen: Layout



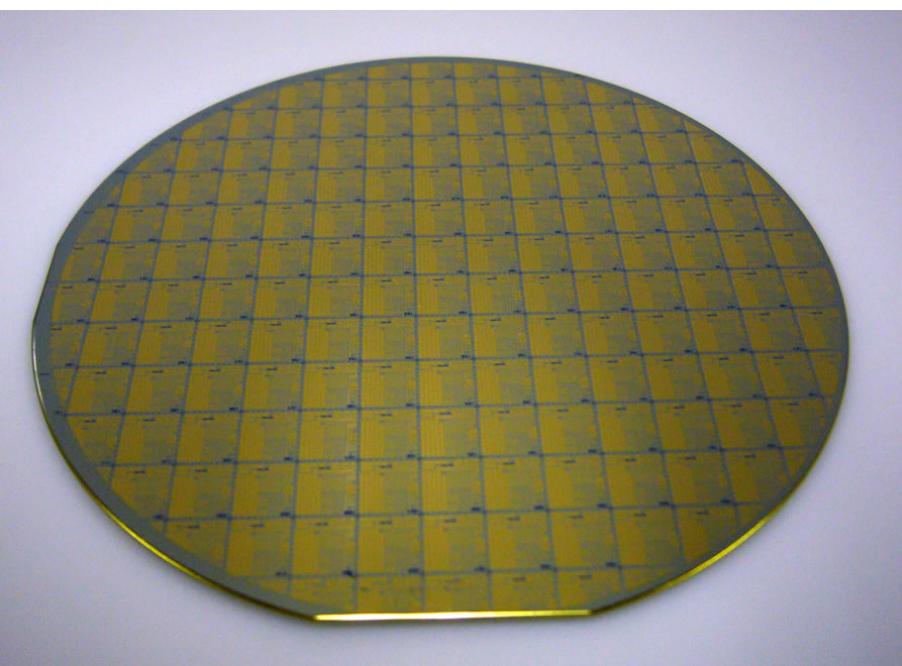
- Layout eines NAND-Gatters in CMOS-Technik als integrierter Schaltkreis.
- Unten: N-Typ-Transistoren, Serienschaltung  
Oben: P-Typ-Transistoren, Parallelschaltung
- Rot: Steuerelektroden (gates), Blau: Verbindungsleitungen aus Metall
- Schwarz: Verbindungen zwischen Metall und Silizium

# Integrierte Schaltungen: Layout



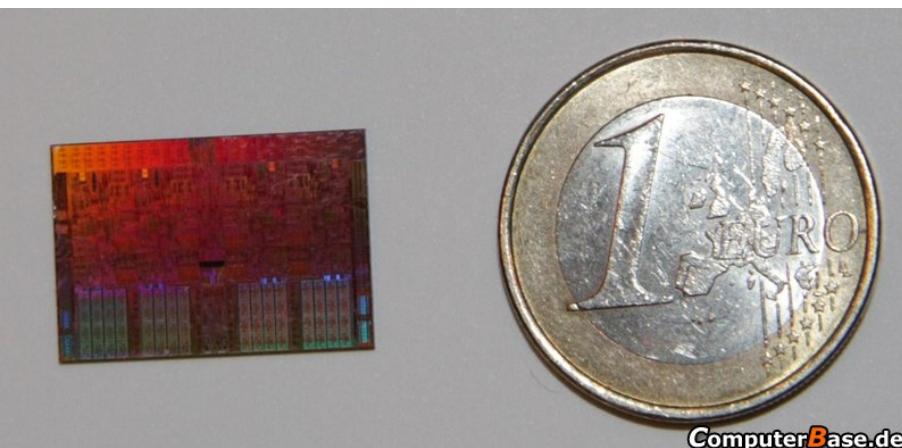
Intel 4004 [1971]  
4/4/12-Bit-Prozessor, 500kHz  
2300 Transistoren, 10 $\mu$ m

# Integrierte Schaltungen: Herstellung



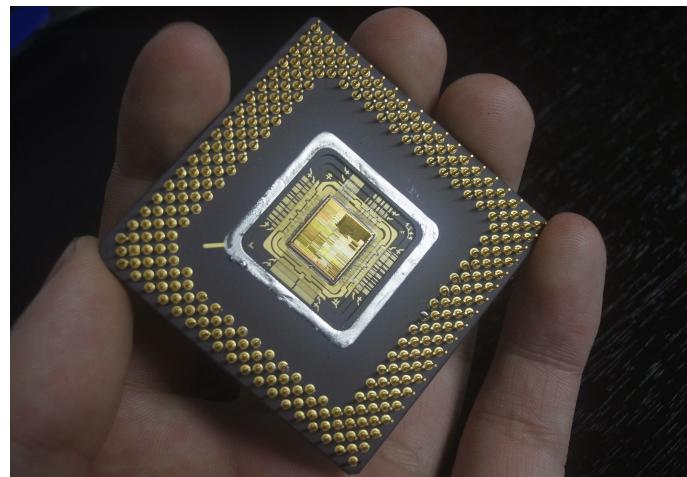
www.phys.org

Intel Nehalem CPU



www.computerbase.de

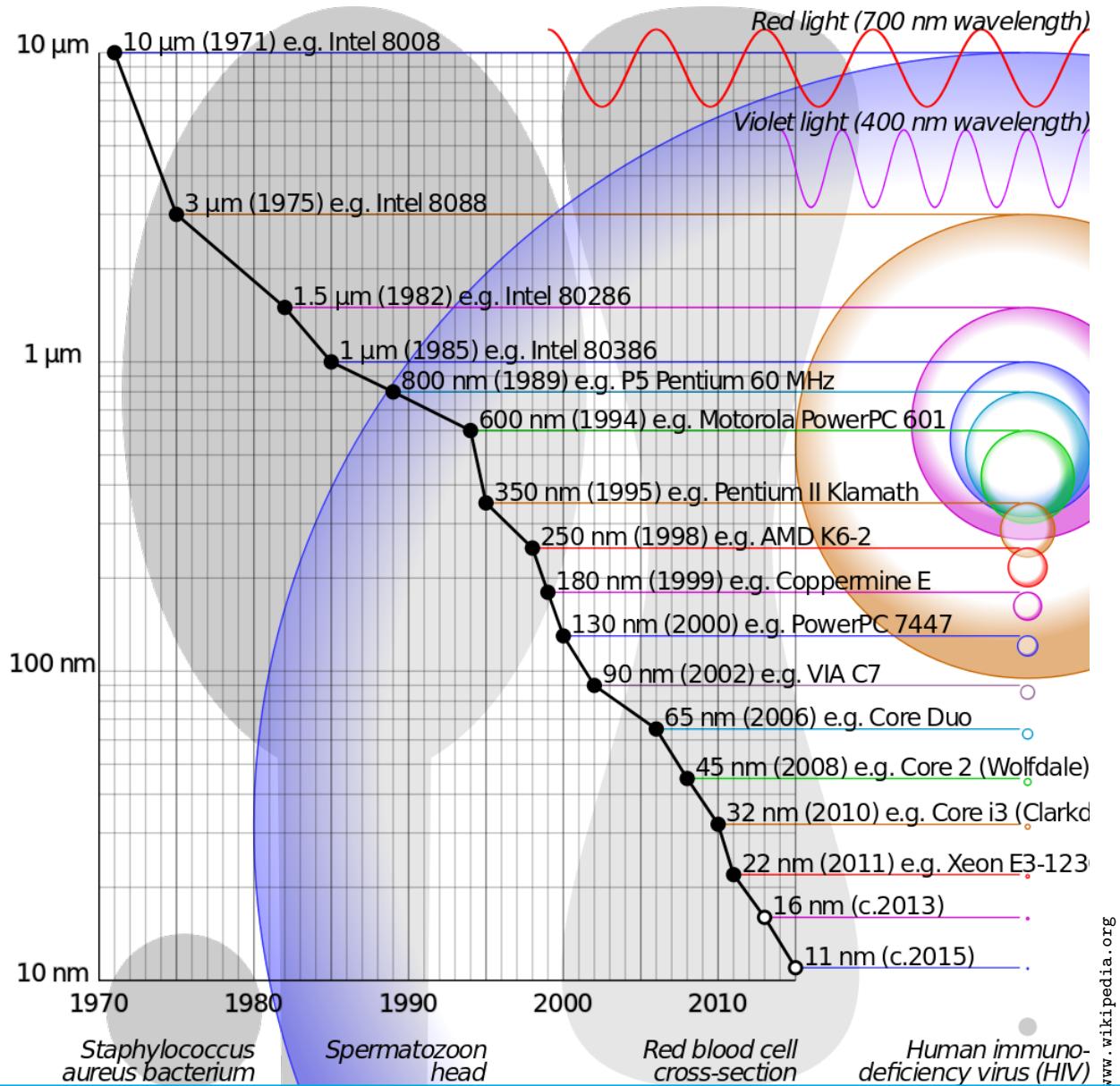
- Es wird ein Silizium-Einkristall hergestellt und in dünne Scheiben (sogenannte wafer) zersägt.
- Durch ein photolithographisches Verfahren wird das Layout der integrierten Schaltung auf diese Scheiben "gedruckt" (mehrere Schichten).
- Die Scheiben werden in kleine Plättchen (chips, dies) zerteilt, die in Gehäuse eingekapselt werden.



Intel Pentium CPU

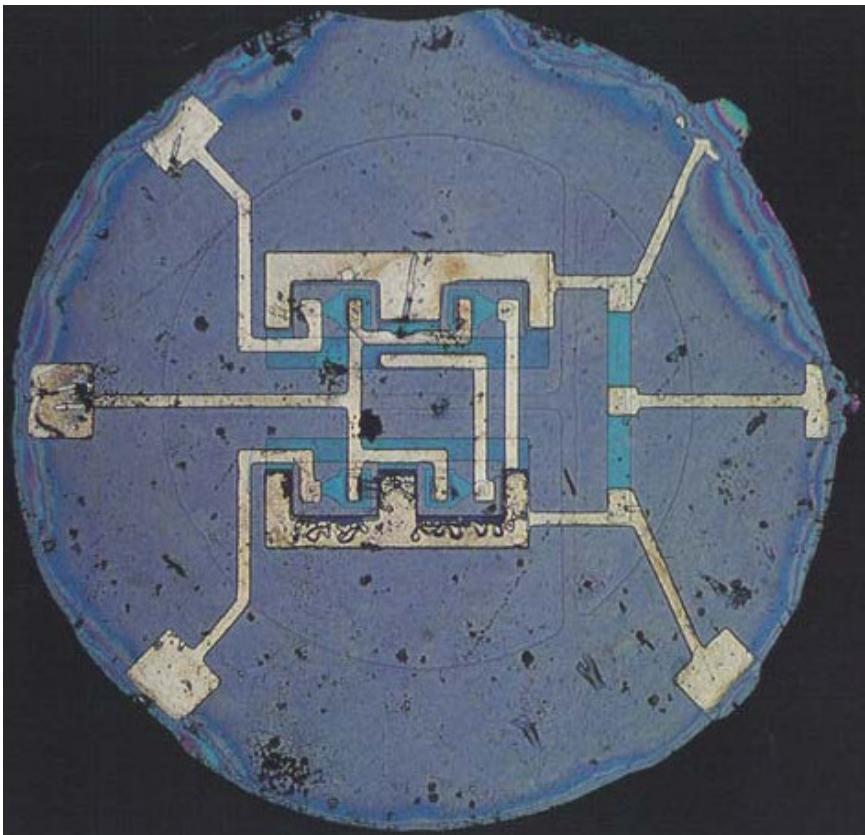
www.wordpress.com

# Integrierte Schaltungen: Herstellung

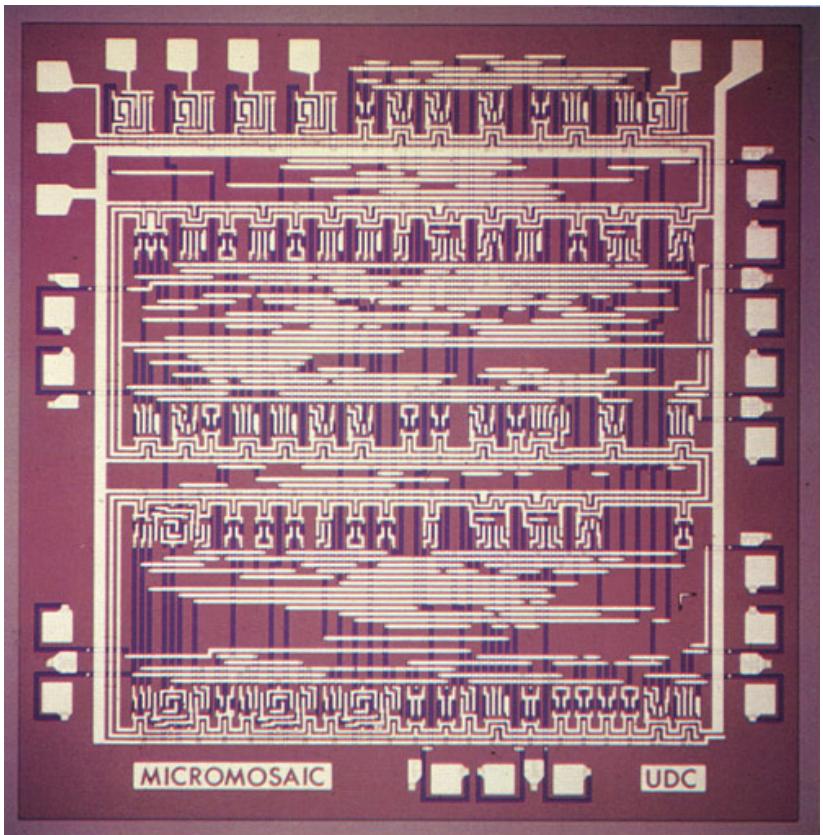


# Integrierte Schaltungen: Speicher

smithsonianchips.si.edu

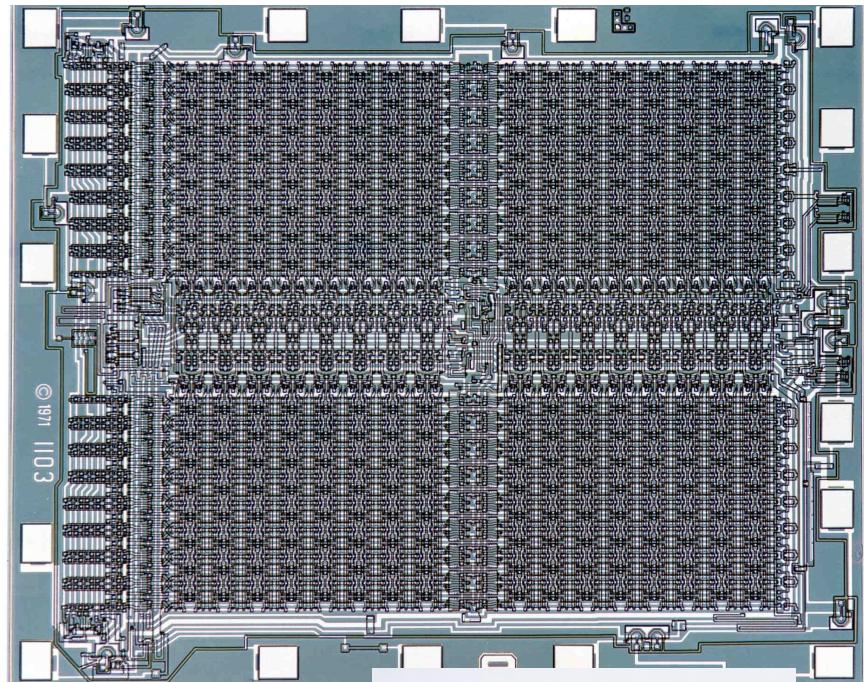
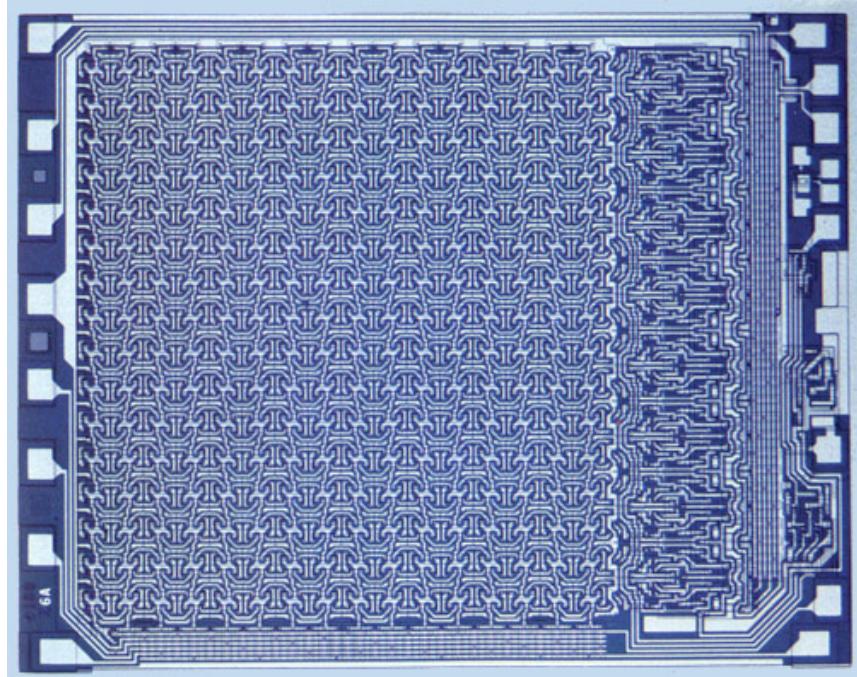


Fairchild Semiconductor  
Flip-Flop (1-Bit-Speicher) [1961]  
4 Transistoren, 5 Widerstände  
“small scale integration (SSI)”



Fairchild Micromosaic [1967]  
Layout erstmals mit CAD entworfen;  
150 AND/OR/NOT-Gatter;  
über die Metallschicht anpassbar.

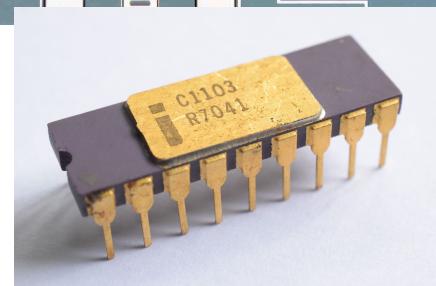
# Integrierte Schaltungen: Speicher



www.themonist.com

Fairchild Semiconductor [1970]  
256 Bit statisches RAM  
(Random Access Memory)

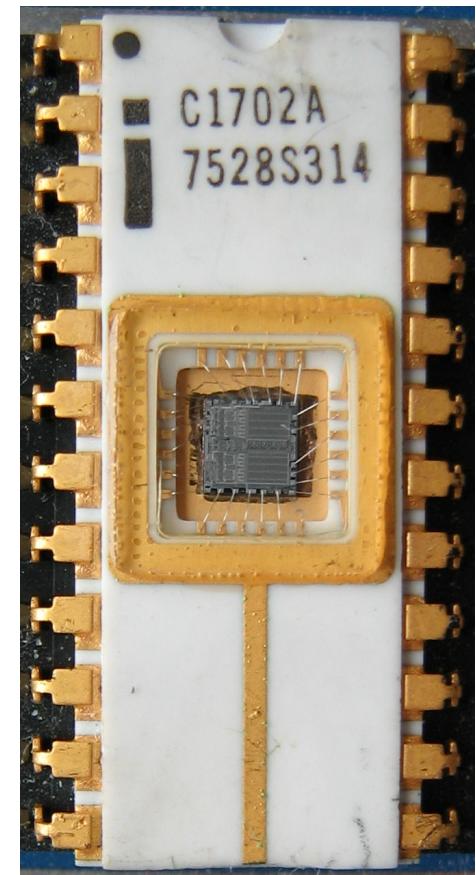
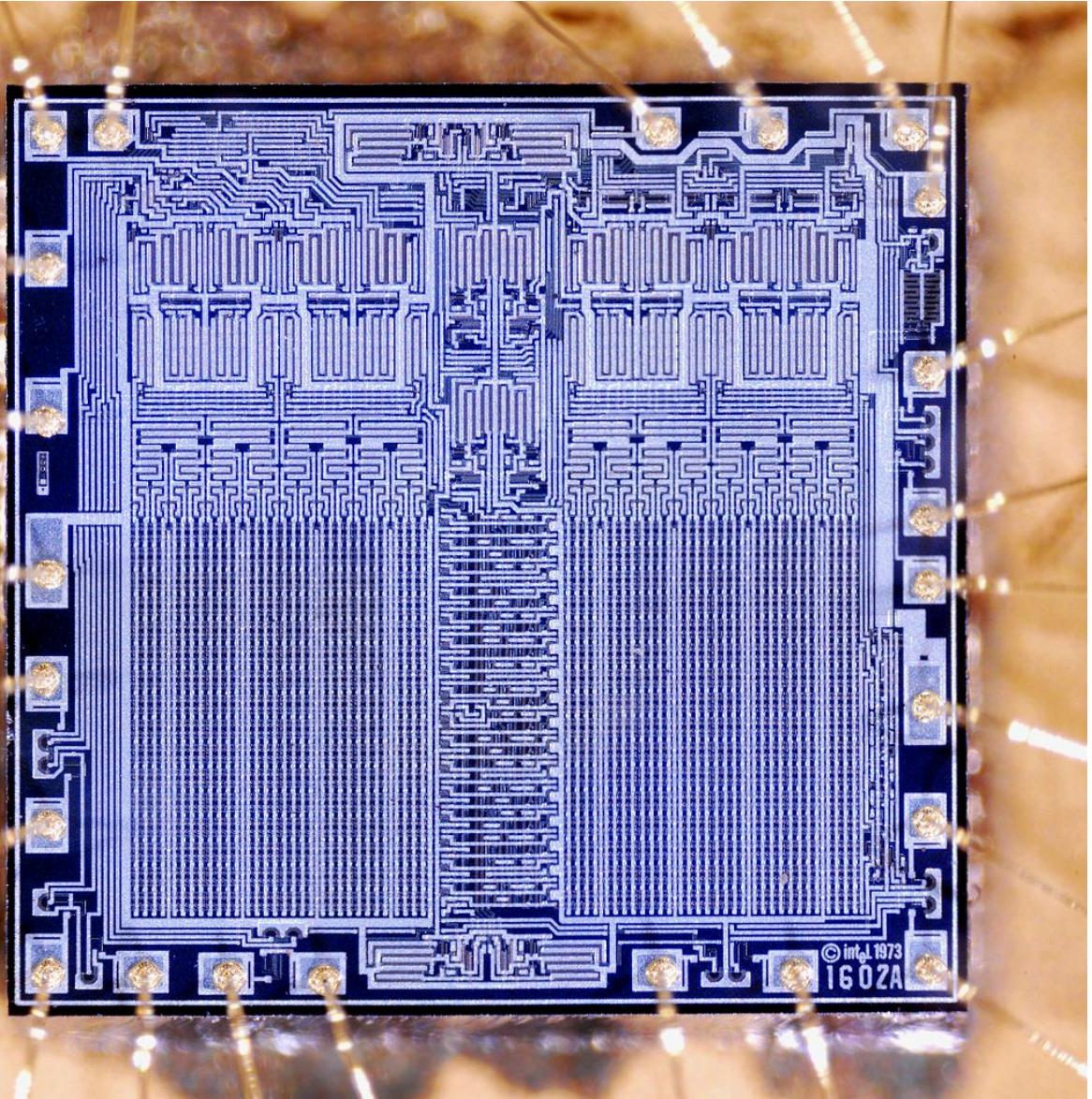
Intel 1103 [1970]  
**(Integrated Electronics)**  
1024 Bit statisches RAM



www.themonist.com

Die Firma Intel wurde 1968 von Gordon Moore und Robert Noyce gegründet  
(frühere Angestellte von Fairchild Semiconductor).

# Integrierte Schaltungen: Speicher

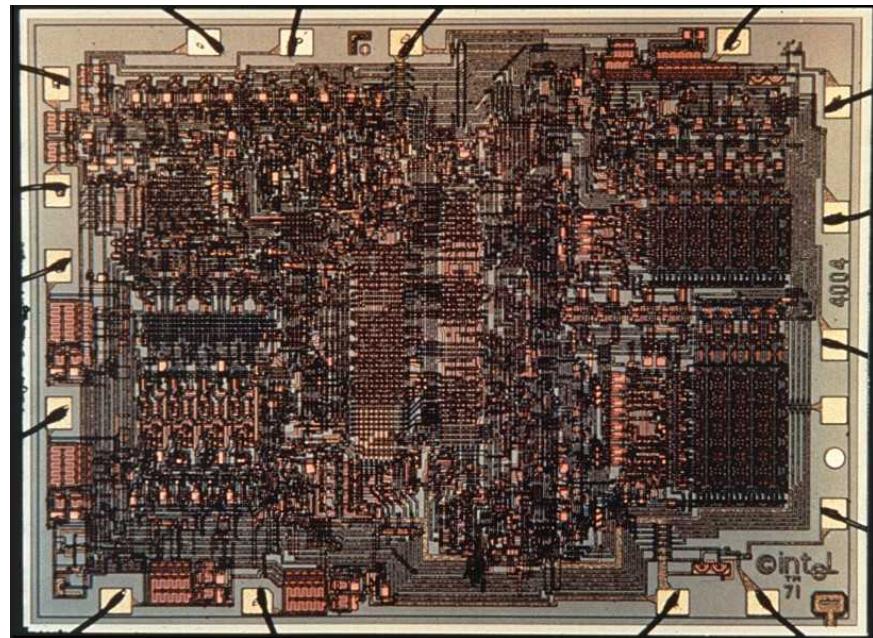


[www.wikipedia.org](http://www.wikipedia.org) (Pöll, 97658)

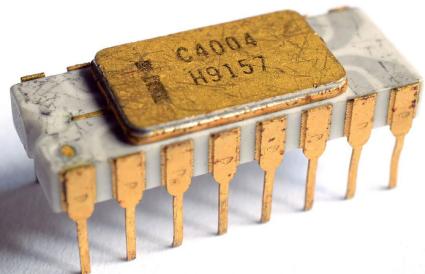
Intel 1702 [1971]  
2048 Bit EPROM  
in  $256 \times 8$  Matrix  
(UV-lösbar)

# Integrierte Schaltungen: Prozessoren

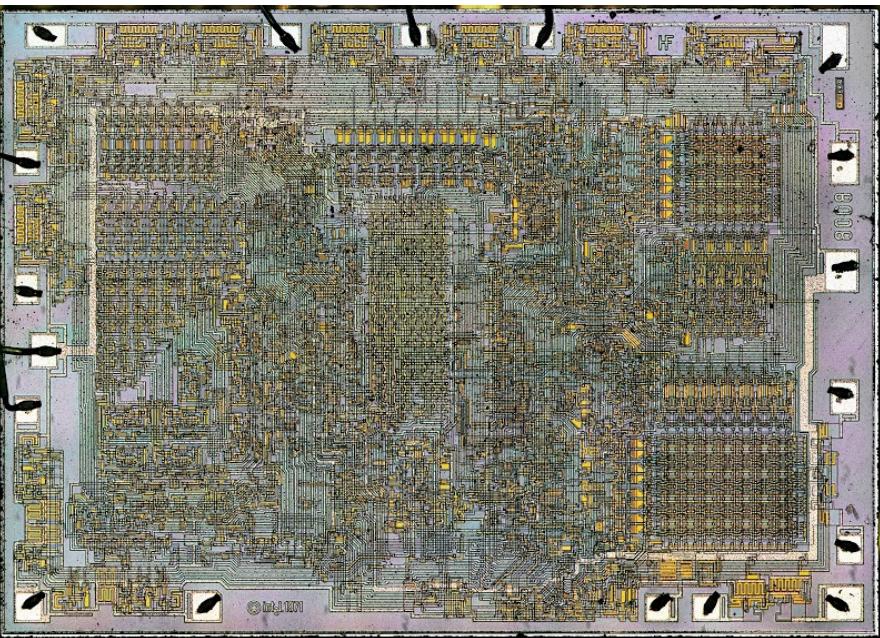
www.4004.com



www.wikipedia.org



Intel 4004 [1971]  
4/12 Bit, 500 kHz  
2300 Transistoren, 10 $\mu$ m



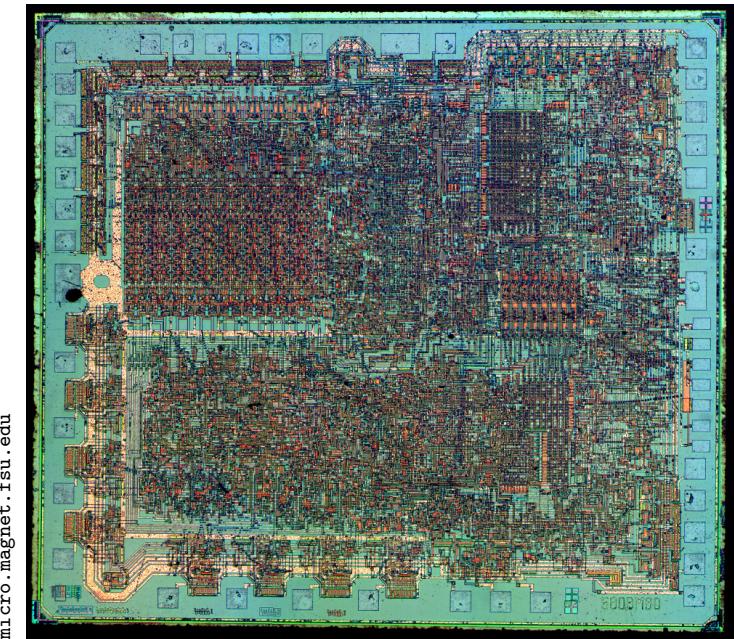
www.righto.com



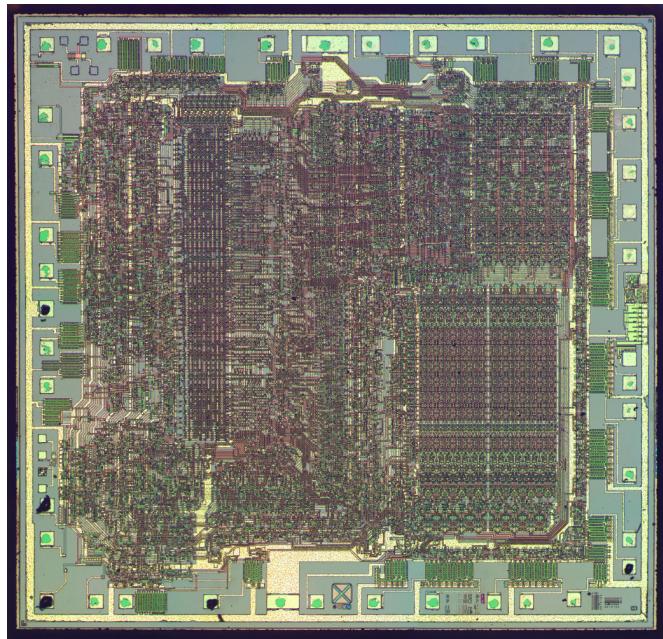
www.wikipedia.org  
(Konstantin Lanzet)

Intel 8008 [1972]  
8/14 Bit, 500 kHz  
3500 Transistoren, 10 $\mu$ m

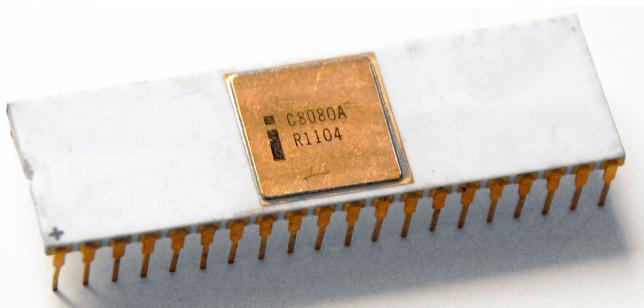
# Integrierte Schaltungen: Prozessoren



[micro.magnet.fsu.edu](http://micro.magnet.fsu.edu)



[zeptobars.com](http://zeptobars.com)



[www.wikipedia.org](http://www.wikipedia.org)  
(Frank Zheng)

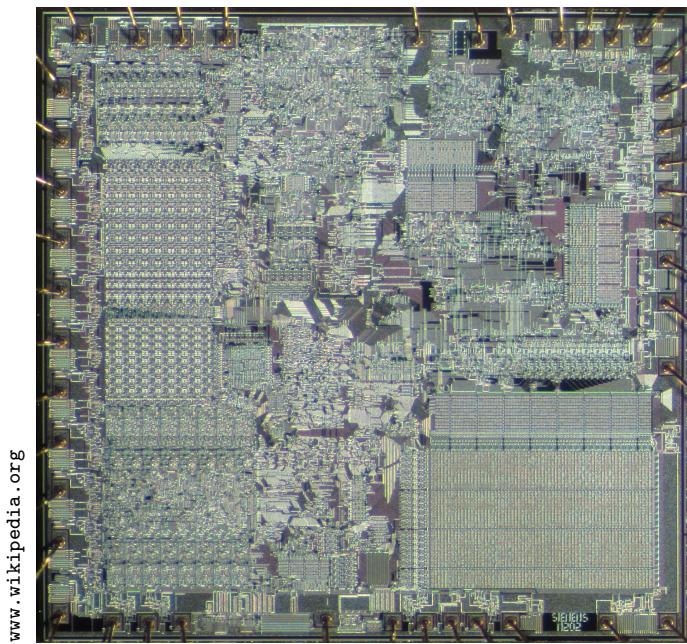
Intel 8080 [1974]  
8–16/8/16 Bit, 2MHz,  
6000 Transistoren,  $6\mu\text{m}$

Zilog Z80 [1976]  
8/8/16 Bit, 2.5MHz  
8500 Transistoren,  $4\mu\text{m}$

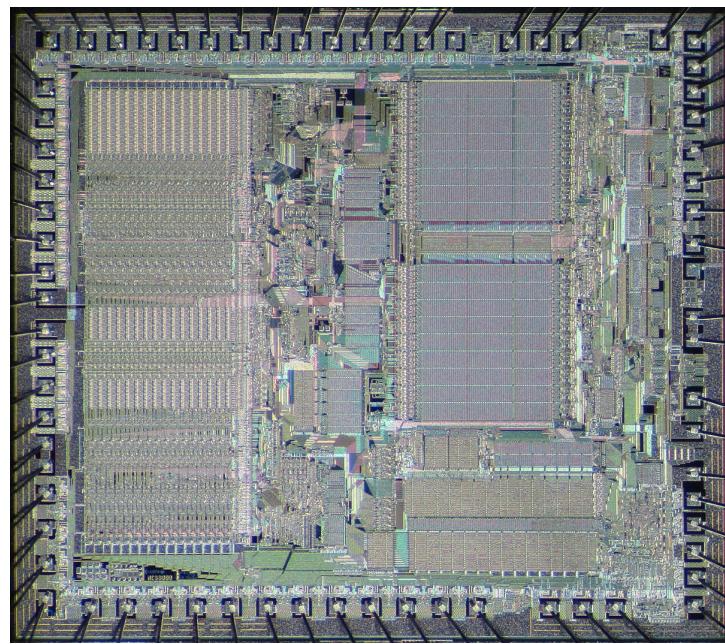


[www.cpu-world.com](http://www.cpu-world.com)

# Integrierte Schaltungen: Prozessoren



[www.wikipedia.org](http://www.wikipedia.org)



[www.wikipedia.org](http://www.wikipedia.org)  
(Pauli Rautakorpi)



[www.wikipedia.org](http://www.wikipedia.org)  
(Thomas Nguyen)

Intel 8086 [1978]  
16/16/20 Bit, 5MHz  
29 000 Transistoren,  $3\mu\text{m}$

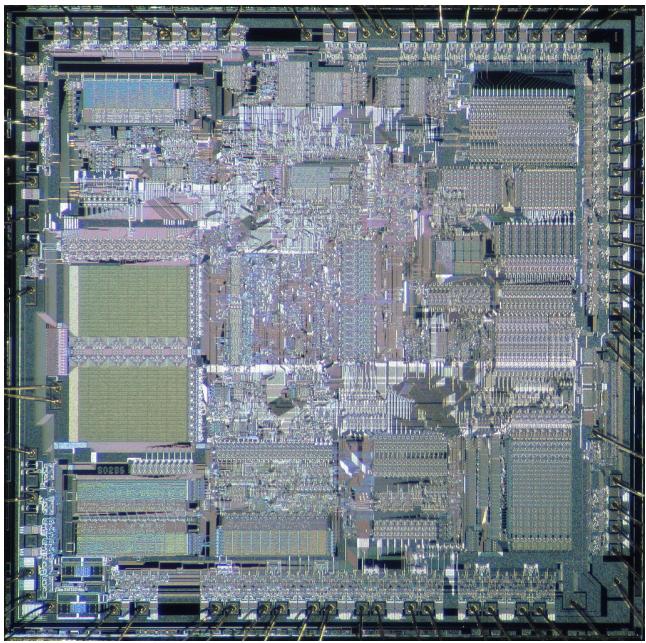


[www.wikipedia.org](http://www.wikipedia.org)  
(Konstantin Lanzet)

Motorola 68000 [1979]  
16–32/16/24 Bit, 4MHz  
68 000 Transistoren,  $3.5\mu\text{m}$

# Integrierte Schaltungen: Prozessoren

[www.wikipedia.org](http://www.wikipedia.org)  
(Pauli Rautakorpi)

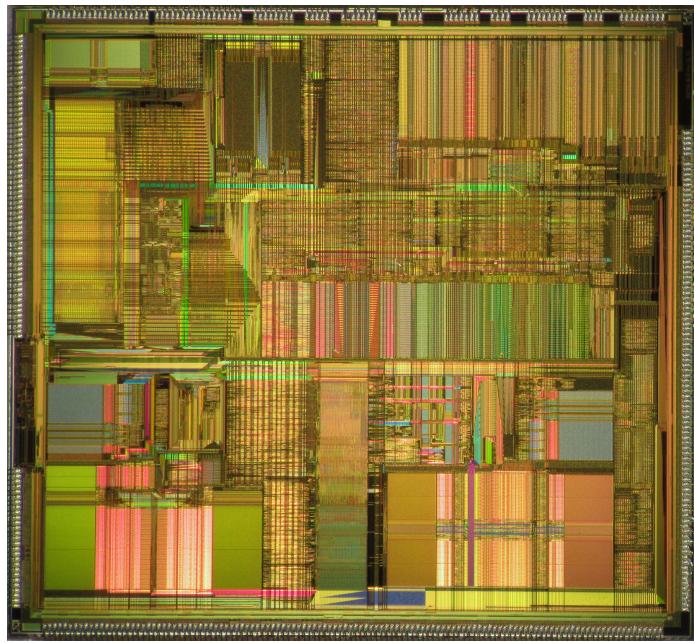


Intel 80286 [1982]  
16/16/24 Bit, 6MHz  
134 000 Transistoren,  $1.5\mu\text{m}$

WS 2020/21

Kapitel 1 - Schaltungstechnik Teil I

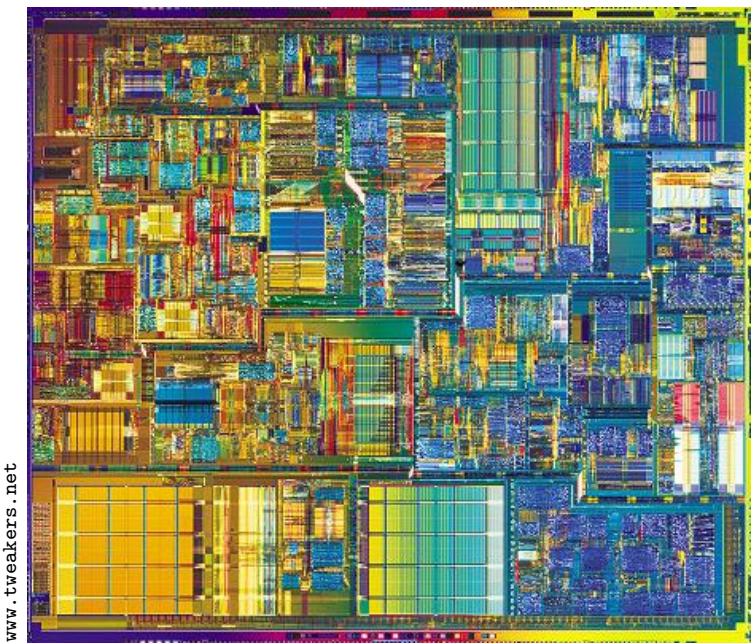
Rechnersysteme und -netze



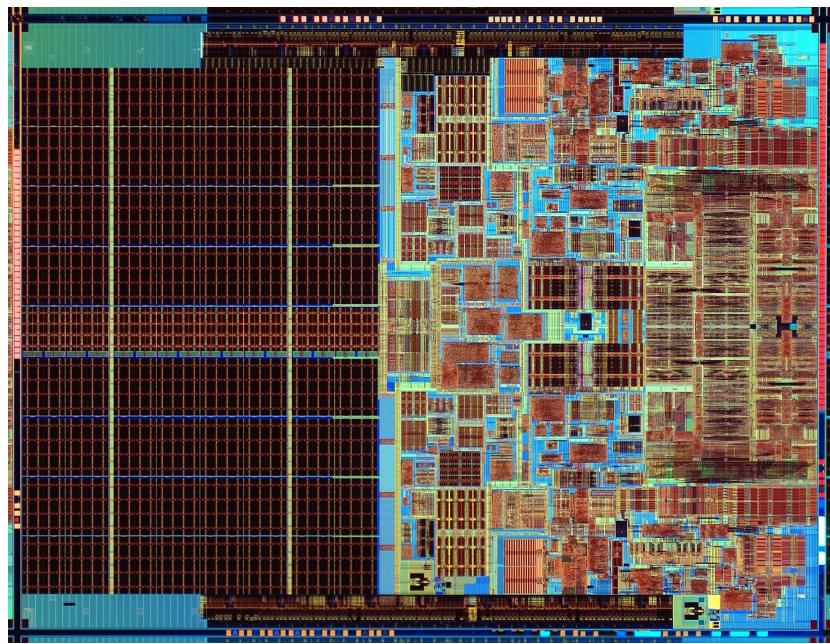
[www.wikipedia.org](http://www.wikipedia.org)  
(Konstantin Lanzet)

Intel Pentium [1993]  
32/64/32 Bit, 60MHz  
 $3.1 \cdot 10^6$  Transistoren, 800nm

# Integrierte Schaltungen: Prozessoren



[www.tweakers.net](http://www.tweakers.net)



[www.infomars.fr](http://www.infomars.fr)



[www.wikipedia.org](http://www.wikipedia.org)  
(Konstantin Lanzet)

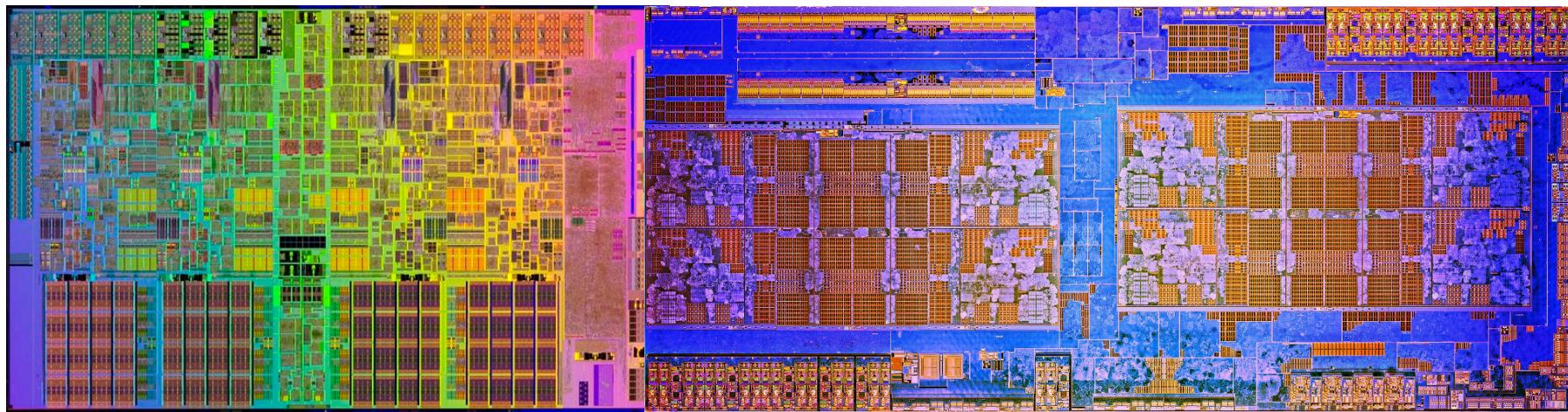
Intel Pentium 4 [2000]  
32/64/32, 1.4GHz  
 $4.2 \cdot 10^7$  Transistoren, 180nm

Intel Core 2 (Conroe) [2006]  
64/64/32 Bit, 1.86GHz  
 $2.9 \cdot 10^8$  Transistoren, 65nm



[www.wikipedia.org](http://www.wikipedia.org)

# Integrierte Schaltungen: Prozessoren

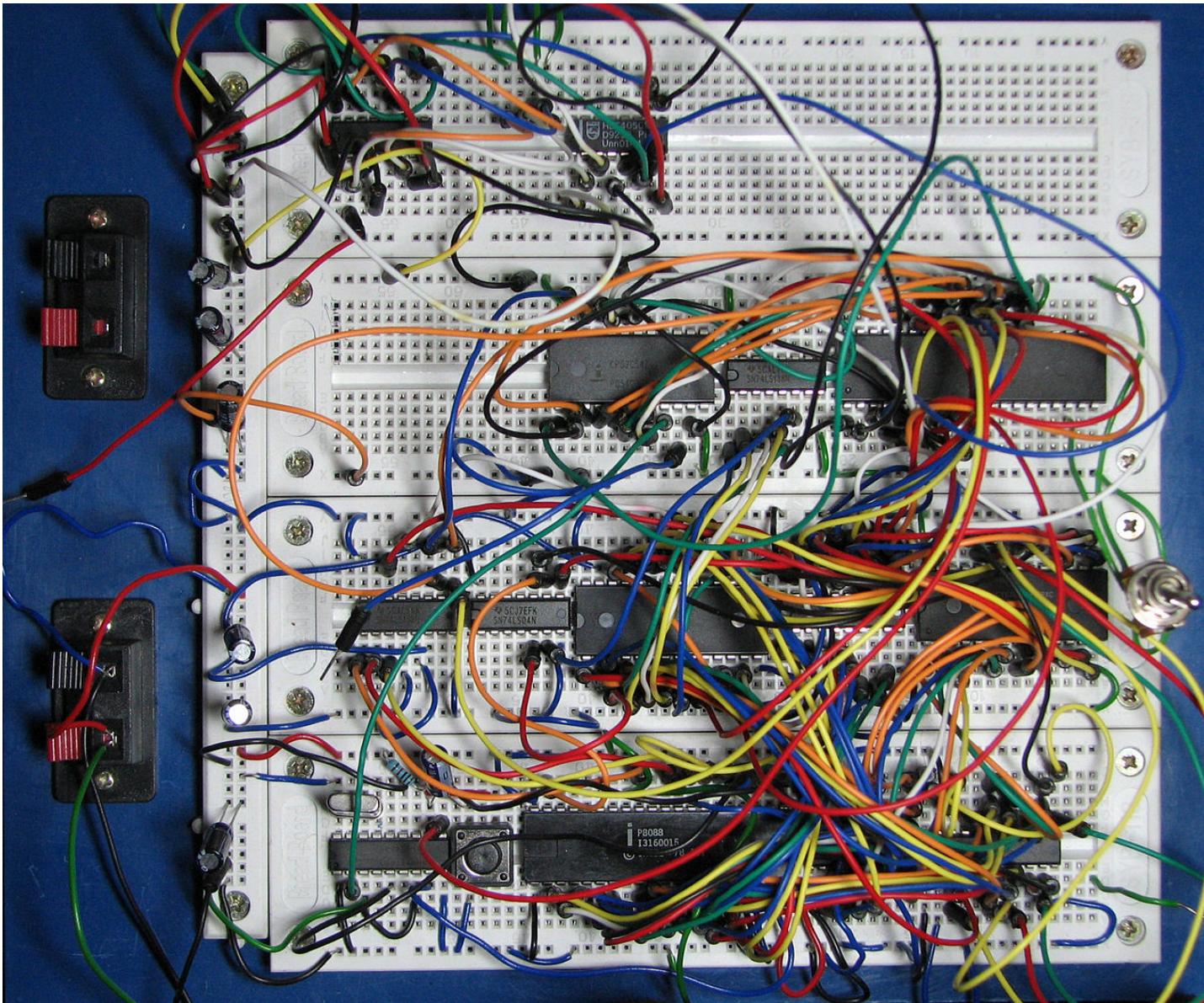


Intel Core i (Nehalem) [2008]  
64/64/36 Bit, 2.67GHz, 4 Kerne  
 $7.31 \cdot 10^8$  Transistoren, 45nm



AMD Ryzen [2017]  
64/64/48 Bit, 3.6GHz, 8 Kerne  
 $4.8 \cdot 10^9$  Transistoren, 14nm

# Steckplatten-Experimente



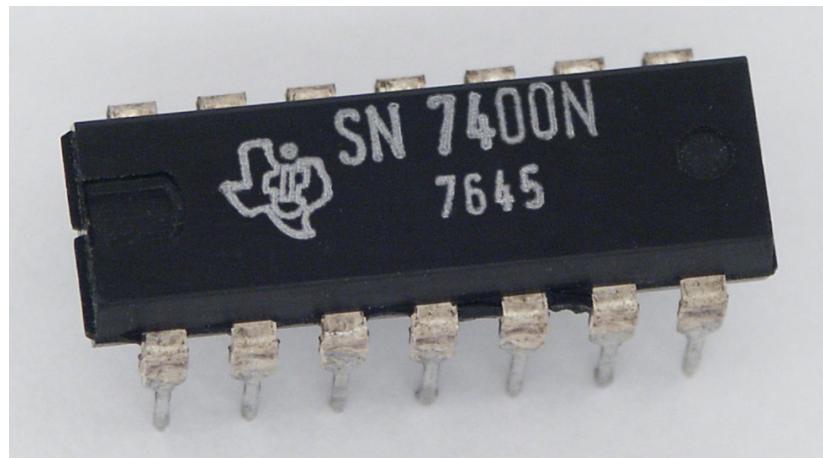
[www.gameover.com.mx](http://www.gameover.com.mx)

# Gatter: Bausteine

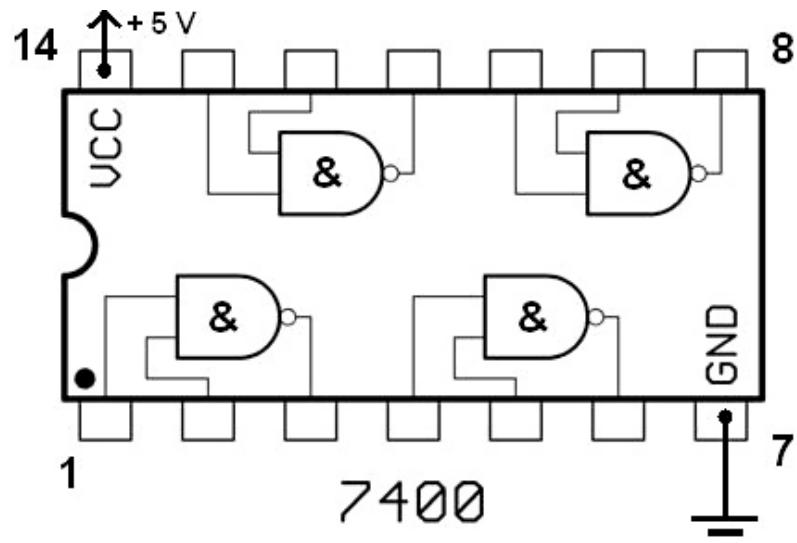
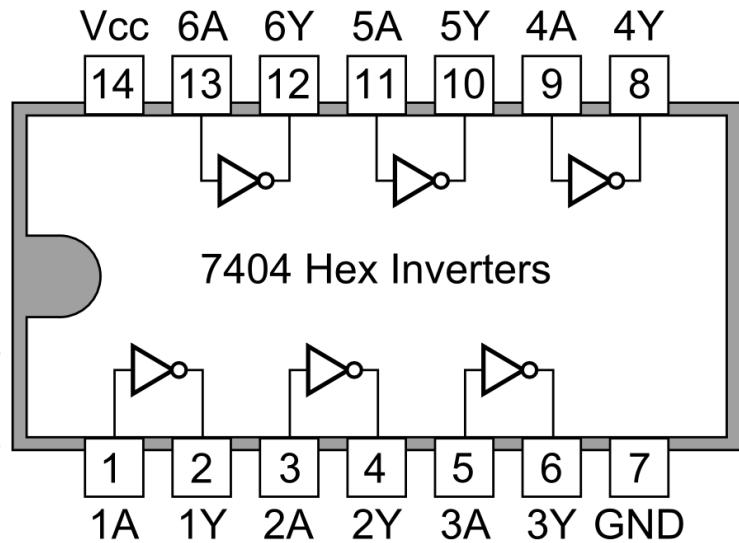
[www.ebay.com](http://www.ebay.com)



[www.wikipedia.org](http://www.wikipedia.org)



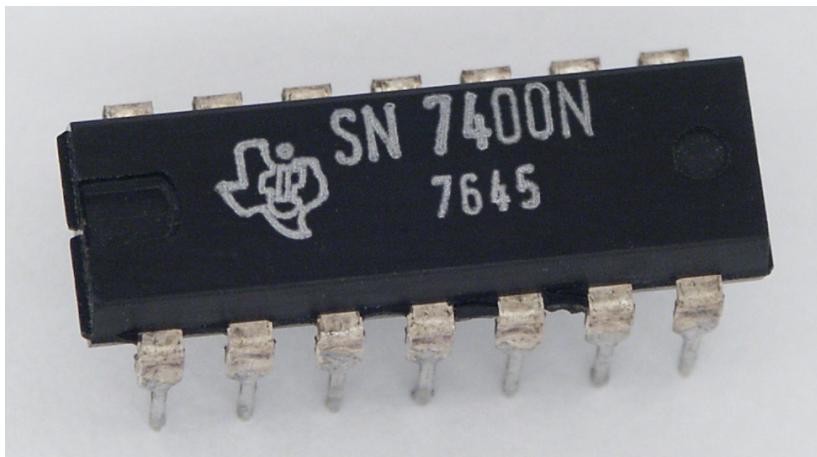
[www.wikipedia.org](http://www.wikipedia.org)



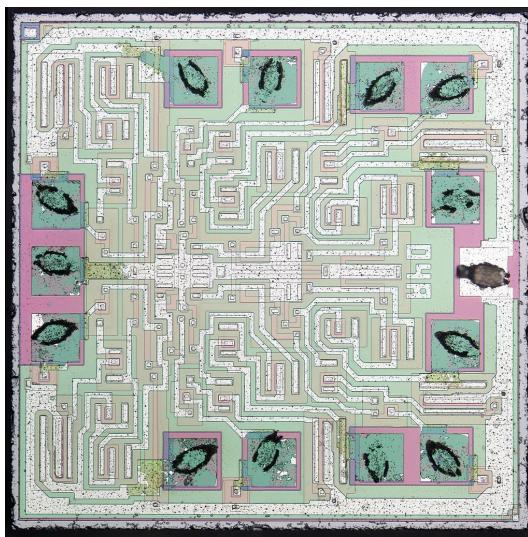
# Gatter: Bausteine



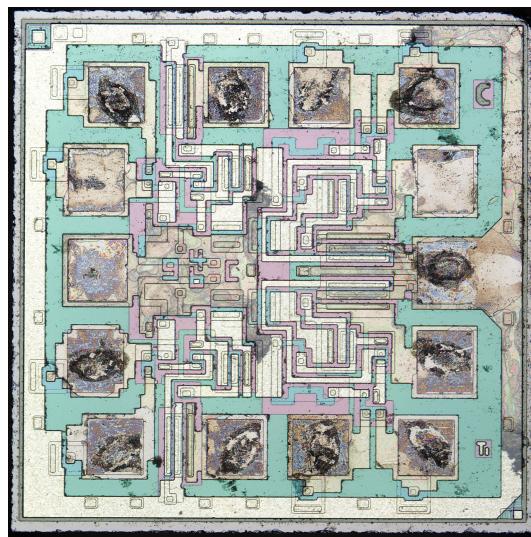
[www.ebay.com](http://www.ebay.com)



[www.wikipedia.org](http://www.wikipedia.org)



[www.wikipedia.org](http://www.wikipedia.org)



[www.wikipedia.org](http://www.wikipedia.org)

# Gatter: Bausteine



Our extensive package range provides maximum flexibility

	Ultra small			Very small			Small		Medium power			
2 Pins	SOD962 (DSN0603-2) 0.6 x 0.3 x 0.3	WL CSP2 0.7 x 0.52 x 0.4	SOD882D (DFN1006D-2) 1.0 x 0.6 x 0.37	SOD882 (DFN1006-2) 1.0 x 0.6 x 0.5	SOD523 1.2 x 0.8 x 0.6	SOD1608 (DFN1608D-2) 1.6 x 0.8 x 0.37	SOD323 1.7 x 1.25 x 0.95	SOD323F 1.7 x 1.25 x 0.7	SOD123F 2.6 x 1.6 x 1.1	SOD123W 2.6 x 1.7 x 1.0	SOD128 3.8 x 2.6 x 1.0	
3 Pins	SOT883B (DFN1006B-3) 1.0 x 0.6 x 0.37	SOT883 (DFN1006-3) 1.0 x 0.6 x 0.5	SOT663 1.6 x 1.2 x 0.55	SOT416 1.6 x 0.8 x 0.77	SOT323 2.0 x 1.25 x 0.95	SOT1061 (DFN2020-3) 2.0 x 2.0 x 0.65	SOT23 2.9 x 1.3 x 1.0	SOT89 4.5 x 2.5 x 1.5				
4/5 Pins	WL CSP4* 0.76 x 0.76 x 0.61	SOT1194 (DFN1010C-4) 1.0 x 1.0 x 0.55	WL CSP5* 1.51 x 1.14 x 0.65	SOT665 1.6 x 1.2 x 0.55	SOT353 2.0 x 1.25 x 0.95	SOT143B 2.9 x 1.3 x 1.0	SOT753 2.9 x 1.5 x 1.0	SOT223 6.5 x 3.5 x 1.65	Power-SO8 4.9 x 3.95 x 1.0			
6/8 Pins	SOT1202 (DFN1010B-6) 1.0 x 1.0 x 0.35	SOT891 (DFN1010-6) 1.0 x 1.0 x 0.5	SOT886 (DFN1410-6) 1.45 x 1.0 x 0.5	SOT666 1.6 x 1.2 x 0.55	WL CSP6* 1.6 x 1.15 x 0.65	SOT363 2.0 x 1.25 x 0.95	SOT1220 (DFN2020MD-6) 2.0 x 2.0 x 0.65	SOT1118 (DFN2020-6) 2.0 x 2.0 x 0.65	SOT457 2.9 x 1.5 x 1.0	SOT505 3.0 x 3.0 x 1.1	SOT873 SOT873-1 (DFN3333-8) 3.3 x 3.3 x 1.0	SOT96 4.9 x 3.9 x 1.75
> 8 Pins	WL CSP9* 1.16 x 1.16 x 0.61	SOT1157 (DFN1712-8) 1.7 x 1.2 x 0.5	SOT983 (DFN1714U-8) 1.7 x 1.35 x 0.5	WL CSP16* 1.9 x 1.97 x 0.65	SOT1178 (DFN2110-9) 2.1 x 1.0 x 0.5	WL CSP24* 2.45 x 2.4 x 0.65	SOT1176 (DFN2510A-10) 2.5 x 1.0 x 0.5	SOT1158 (DFN2512-12) 2.5 x 1.2 x 0.5	SOT1156 (DFN2521-12) 2.5 x 2.1 x 0.5	SOT552 3.0 x 3.0 x 1.1	SOT617 (DFN5050-32) 5.0 x 5.0 x 1.0	SOT510 9.7 x 4.4 x 1.1

\* The exact position of the ball(s) and package dimensions vary.



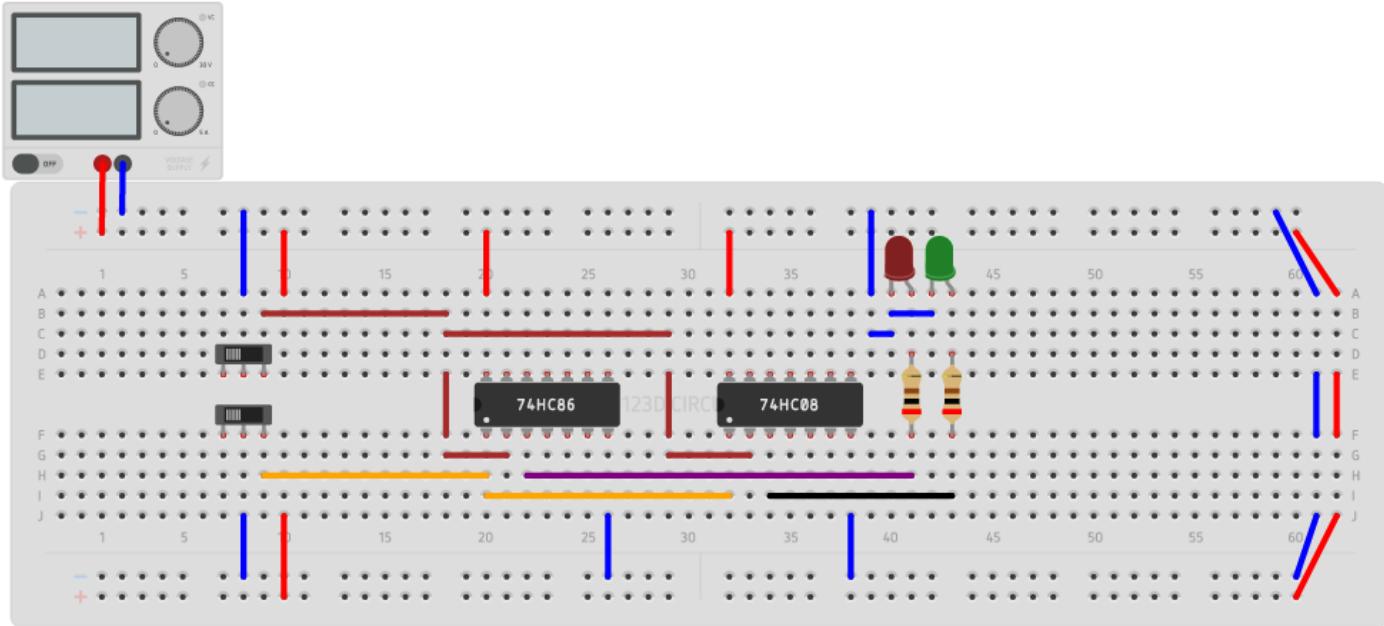
A partner of excellence, experience and innovation for our customers

# Steckplatten-Simulator

AUTODESK 123D CIRCUITS    ≡    🔍 Search for projects, components or people    + New    👤

1 bit half adder | Electronics Lab    Show more

▶ Start Simulation    💻 Code Editor    ⬇️    ⬅️    ➡️    ⬆️



The circuit diagram shows a 1-bit half adder implemented on a breadboard. The inputs are labeled A and B, and the outputs are labeled S (sum) and C (carry). The circuit uses two 74HC08 logic gates. The first gate (74HC08) has its inputs A and B connected to the inputs of the second gate (74HC08). The second gate's output is the sum (S), and its carry output is connected to one input of the first gate. The other input of the first gate is connected to ground. The power supply is a 3V DC source connected to the breadboard. A digital voltmeter is connected across the output lines to measure the signal levels.