# Kapitel 5: Schaltnetze und Schaltwerke


5.1 Einführung und Überblick

5.2 Boolesche Funktionen und Boolesche Algebra

5.3 Schaltnetze

5.4 Schaltwerke

# Quellen


- M. Broy: "Informatik - Eine grundlegende Einführung", Teil II, Springer-Verlag, 1992 (Kap. 2)
- U. Rembold, P. Levi: "Einführung in die Informatik für Naturwissenschaftler und Ingenieure", 3. Auflage, Hanser-Verlag, 1999 (Kap. 2.4)
- D. Werner u.a.: "Taschenbuch der Informatik", Fachbuchverlag Leipzig, 1995 (Kap. 3.2)
- F. Mayer-Lindenberg: "Konstruktion digitaler Systeme", Vieweg-Verlag, 1998 (Kap. 1)
- H.-P. Gumm, M. Sommer: "Einführung in die Informatik", 2. Auflage, Addison-Wesley, 1995 (Kap. IV.2)
- H. Dispert, H.-G. Heuck: "Einführung in die Technische Informatik und Digitaltechnik", Vorlesungsskript FH Kiel (Kap. 1-4),
  http://www.e-technik.fh-kiel.de/~dispert/digital/digital/dig0_00.htm
- F. Flores: "Informatik für Ingenieure", Vorlesungsskript, TU Harburg, (Kap. 2, 4 und 5)
- Th. Schwentick: "Grundzüge der Informatik I", Vorlesungsskript, Uni Mainz, (Kap. 5)

# 5.1 Einführung und Überblick

- **Ziel dieses Kapitels:**
  - **Annäherung an die Realisierung von Rechnern.**
  - **Zunächst "im kleinen": einfache digitale Schaltungen, wie sie im Innern eines Prozessors vorkommen.**
    **Im folgenden Kapitel: "im großen": Architektur von Rechensystemen**
- **Beschreibung digitaler Schaltungen auf der logischen Ebene**
  - **durch mathematische Modelle wie Boolesche Funktionen, Boolesche Terme und Boolesche Algebren (5.2)**
  - **durch Schaltnetze (5.3) aus zyklusfreien Graphen als Modelle der Realisierung Boolescher Funktionen**
  - **durch Schaltwerke (5.4) als Modelle für zustandsbehaftete digitale Systeme (Berücksichtigung von Gedächtnis/Speicher).**
- **Detaillierte Behandlung einschl. der technischen Umsetzung in elektronischen Bauteilen erfolgt in der Veranstaltung Elektrotechnik/Digitaltechnik (2. Semester).**

# 5.2 Boolesche Funktionen und Boolesche Algebra

Def

- Eine *Schaltfunktion* wird definiert durch eine Abbildung

$$f_s : \{0,1\}^n \rightarrow \{0,1\}^m .$$

  Sie bildet die Menge der binären n-Tupel von n *Eingängen* in die Menge der binären m-Tupel von m *Ausgängen* ab.
- Schaltfunktion kann als math. Abstraktion eines elektronischen Bausteins mit n Eingängen und m Ausgängen angesehen werden.
- In 5.2 und 5.3 werden nur solche Schaltfunktionen betrachtet, bei denen die Ausgänge nur von den Eingängen abhängen (Funktionen im math. Sinne, ohne Rückkopplung).
- Eine *n-stellige Boolesche Funktion* ist eine Funktion $f:\{0,1\}^n \rightarrow \{0,1\}$. Sie heißt für n=1 *unär*, für n=2 *binär*, ansonsten auch *n-är*. Zuordnung von Wahrheitswerten: 0 = falsch, 1 = wahr.
- Schaltfunktionen lassen sich als Kombinationen von Booleschen Funktionen auffassen:

$$f_s (x_1, ..., x_n) = ( f^1(x_1,...,x_n), ..., f^m(x_1,...,x_n) )$$

# Wahrheitstafel einer Booleschen Funktion

- Eine n-stellige Boolesche Funktion $f:\{0,1\}^n \rightarrow \{0,1\}$ kann über eine generische Wahrheitstafel (Wertetabelle) mit $2^n$ Zeilen definiert werden:

| $x_1$ | $x_2$ | ... | $x_{n-1}$ | $x_n$ | $f(x_1,...,x_n)$ |
|---|---|---|---|---|---|
| 0 | 0 | ... | 0 | 0 | f(0,0,...,0,0) |
| 0 | 0 | ... | 0 | 1 | f(0,0,...,0,1) |
| ... | ... | ... | ... | ... | ... |
| 1 | 1 | ... | 1 | 0 | f(1,1,...,1,0) |
| 1 | 1 | ... | 1 | 1 | f(1,1,...,1,1) |

- Da in jeder der $2^n$ Zeilen entweder der Funktionswert 0 oder der Funktionswert 1 angenommen wird, existieren genau

$$2^{(2^n)}$$

verschiedene Funktionen $f:\{0,1\}^n \rightarrow \{0,1\}$.

# Beispiel: Einstellige Boolesche Funktionen

- **Für n=1 ergeben sich aus $2^{(2^n)}$ genau 4 Funktionen:**

| $x_1$ | $0(x_1)$ | $1(x_1)$ | $Id(x_1)$ | $NOT(x_1)$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |

- **Dabei bezeichnen**
  - 0() die Null-Funktion $f \equiv 0$,
  - 1() die Eins-Funktion $f \equiv 1$,
  - Id() die Identitätsfunktion $f(x_1)=x_1$ und
  - NOT() die Negation (Verneinung, Inversion) $f(x_1)=\overline{x}_1$ (lies: $x_1$ negiert). Andere Schreibweisen: NICHT(), $f(x_1)=\neg x_1$.

- **Die Negation ist für das Weitere von besonderer Bedeutung.**

# Beispiel: Wichtige 2-stellige Boolesche Funktionen

- Für n=2 ergeben sich aus $2^{(2^n)}$ genau 16 Funktionen.
- Die für die Praxis wichtigen Funktionen sind:

| $x_1$ | $x_2$ | AND $(x_1,x_2)$ | OR $(x_1,x_2)$ | XOR $(x_1,x_2)$ | NAND (x1,x2) | NOR $(x_1,x_2)$ | IMPL $(x_1,x_2)$ | EQUIV $(x_1,x_2)$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |

# Wichtige 2-stellige Boolesche Funktionen (2)

- **Andere Schreibweisen und Bezeichnungen:**

| AND $(x_1,x_2)$ | OR $(x_1,x_2)$ | XOR $(x_1,x_2)$ | NAND $(x_1,x_2)$ | NOR $(x_1,x_2)$ | IMPL $(x_1,x_2)$ | EQUIV $(x_1,x_2)$ |
|---|---|---|---|---|---|---|
| $x_1 \wedge x_2$ | $x_1 \vee x_2$ | $x_1 \oplus x_2$ | $\overline{x_1 \wedge x_2}$ | $\overline{x_1 \vee x_2}$ | $x_1 \Rightarrow x_2$ | $x_1 \Leftrightarrow x_2$ |
| $x_1 {}^{*} x_2$ | $x_1 + x_2$ | $x_1 \neq x_2$ | $\overline{x_1 {}^{*} x_2}$ | $\overline{x_1 + x_2}$ | $\overline{x_1} + x_2$ | $x_1 = x_2$ |
| UND | ODER | Exclusiv Oder | NICHT UND | NICHT ODER | | |
| Kon-junktion | Dis-junktion | Anti-valenz | Sheffer-Funktion | Pierce-Funktion | Impli-kation | Äqui-valenz |

# n-stellige Boolesche Funktionen

- **Satz:**
  **Alle höherstelligen (n≥3) Booleschen Funktionen können durch Verknüpfung 2-stelliger Boolescher Funktionen erzeugt werden.**

- **Technisch sehr einfach realisieren lassen sich**
  - **n-faches NAND:**
    $$\mathrm{NAND}(x_1, \ldots, x_n) = \mathrm{NOT}(\ \mathrm{AND}(x_1, \ldots, x_n)\ ) = \mathrm{NOT}(\ \mathrm{AND}(\ \ldots\ \mathrm{AND}(\ \mathrm{AND}(x_1,x_2),\ x_3),\ \ldots,x_n)\ ) = \overline{x_1 * \ldots * x_n}$$
  - **n-faches NOR:**
    $$\mathrm{NOR}(x_1, \ldots, x_n) = \mathrm{NOT}(\ \mathrm{OR}(x_1, \ldots, x_n)\ ) = \mathrm{NOT}(\ \mathrm{OR}(\ \ldots\ \mathrm{OR}(\ \mathrm{OR}(x_1,x_2),\ x_3),\ \ldots,x_n)\ ) = \overline{x_1 + \ldots + x_n}$$

# Kombination Boolescher Funktionen

- **Durch die Kombination Boolescher Funktionen lassen sich andere Boolesche Funktionen erzeugen.**
- **Von besonderer Bedeutung:**
  **Funktionen oder Funktionenmengen, mit deren Hilfe sich alle Booleschen Funktionen erzeugen lassen (genannt *lässt*; Beschränkung auf wenige verschiedene Bauteile).**
- **Satz: Jede n-stellige Boolesche Funktion lässt sich durch NOT() und das binäre AND() und/oder OR() darstellen.**
- **Satz: Jede n-stellige Boolesche Funktion lässt sich ausschließlich durch binäre NOR-Funktionen darstellen.**
- **Satz: Jede n-stellige Boolesche Funktion lässt sich ausschließlich durch binäre NAND-Funktionen darstellen.**
- **Bemerkung: NOR() und NAND() sind damit von großer praktischer Bedeutung, da damit nur ein Komponententyp für die Realisierung benötigt wird.**

# Boolesche Algebra

- Die Menge {0,1} zusammen mit den binären Operationen OR() und AND() unter Benutzung von NOT() zur Invertierung erfüllt die Eigenschaften einer math. Struktur, die Boolesche Algebra genannt wird:

Def

- Ein Tripel (M,+,*) aus einer Menge M und zwei binären Funktionen +,* : $M \times M \rightarrow M$ heißt ***Boolesche Algebra*** genau dann, wenn für alle $x,y,z \in M$ gilt:
  - Assoziativ-Gesetze: $(x+y)+z = x+(y+z)$ und $(x*y)*z = x*(y*z)$
  - Kommutativ-Gesetze: $(x+y) = (y+x)$ und $(x*y) = (y*x)$
  - Distributiv-Gesetze: $x*(y+z) = (x*y)+(x*z)$ und $\boxed{x+(y*z) = (x+y)*(x+z)}$
  - Absorptions-Gesetze: $\boxed{x*(x+y) = x \text{ und } x+(x*y) = x}$
  - Neutrale Elemente: $\exists\, 0 \in M$ mit $0+x=x$ und $\exists\, 1 \in M$ mit $1*x=x$
  - Inverses Element: $\forall\, x \in M$ existiert $\overline{x} \in M$ mit $\boxed{x*\overline{x}=0 \text{ und } x+\overline{x}=1}$

# Beispiele von Booleschen Algebren

- ({0,1}, OR, AND) ist eine Boolesche Algebra:
  - OR entspricht +
  - AND entspricht *
  - 0 und 1 sind neutrale Elemente
  - Zu $x \in \{0,1\}$ ist NOT(x) das inverse Element $\overline{x}$.
- Gegeben sei eine Menge M, sei $\mathbb{P}(M)$ deren Potenzmenge.
  Dann ist ($\mathbb{P}(M)$, $\cup$, $\cap$) eine Boolesche Algebra:
  - $\cup$ entspricht +
  - $\cap$ entspricht *
  - $\varnothing$ und M sind neutrale Elemente, $\varnothing$ entspricht 0, M entspricht 1.
  - Zu $A \in M$ ist das Mengen-Komplement M\A das inverse Element.
- Sind $B_1, \ldots, B_n$ Boolesche Algebren, dann ist auch das Kreuzprodukt $B_1 \times \ldots \times B_n$ mit komponentenweisen Verknüpfungen eine Boolesche Algebra.

# Rechenregeln für Boolesche Algebren

- **Die folgenden wichtigen Rechenregeln gelten allgemein für jede Boolesche Algebra (M,+,*):**
  - **Idempotenz:** $x+x = x$ und $x*x = x$
  - **Doppelte Negation:** $\overline{\overline{x}} = x$
  - **De Morgansche Regeln:** $\overline{(x+y)} = \overline{x} * \overline{y}$ und $\overline{(x*y)} = \overline{x} + \overline{y}$

    **Praktische Anwendung:**
    **Überführung von Konjunktion in Disjunktion und umgekehrt**

# Dualitätsprinzip

- Zu jeder gültigen Rechenregel einer Booleschen Algebra gehört eine andere gültige (die *duale*) Rechenregel, die aus der ursprünglichen entsteht durch:
  - vertausche die Rollen von * und +
  - vertausche die Rollen von 0 und 1

- Beispiele für duale Regeln:
  - Idempotenzregeln
  - De Morgansche Regeln

# Boolesche Terme

Def

- Wahrheitstafeln sind für vielstellige Funktionen unhandlich, da die Anzahl $2^n$ der Zeilen stark wächst.
- Eine weitere wichtige Repräsentierung Boolescher Funktionen bilden die *Booleschen Terme* (oder *Booleschen Ausdrücke*), implizit definiert durch:
  - Sei VAR=$\{x_1, x_2, ...\}$. Die $x_i \in$ VAR heißen *Boolesche Variablen*.
  - Die Konstanten 0 und 1 sind Boolesche Terme.
  - Für jedes i ist $x_i$ ein Boolescher Term.
  - Sind s und t Boolesche Terme, dann auch $\neg s$, $(s \vee t)$ und $(s \wedge t)$.
- Verwendet wurden hier die logischen Verknüpfungssymbole $\neg$, $\vee$ und $\wedge$. Alternativ werden auch $\overline{\phantom{x}}$, + und * verwendet.
- Festlegungen zur Vereinfachung der Schreibweise:
  - Weglassen v. Klammern: $\neg$ bindet stärker als $\wedge$ bindet stärker als $\vee$.
  - In der ($\overline{\phantom{x}}$, +, *)-Schreibweise "geht Punkt- vor Strichrechnung", und der * kann auch entfallen: $x_1x_2 := x_1 * x_2$

# Boolesche Terme $\leftrightarrow$ Boolesche Funktionen

- **Jedem Booleschen Term entspricht eine Boolesche Funktion:**
  - $\neg$ entspreche der Negation $\overline{\phantom{x}}$ bzw. NOT(),
  - $\wedge$ entspreche der Konjunktion * bzw. AND()
  - $\vee$ entspreche der Disjunktion + bzw. OR()
  - Mittels Wahrheitstafeln kann damit jedem Booleschen Term t über den Booleschen Variablen $\{x_1, ..., x_n\}$ unmittelbar eine Boolesche Funktion

    $$f_t:\{0,1\}^n \rightarrow \{0,1\}$$

    zugeordnet werden.

- **<u>Satz</u>:**
  **Jede Boolesche Funktion $f:\{0,1\}^n \rightarrow \{0,1\}$ lässt sich durch einen Booleschen Term über $\{\neg, \wedge, \vee\}$ beschreiben.**

# Boolesche Terme $\leftrightarrow$ Boolesche Funktionen (2)

**Idee für einen Induktionsbeweis:**

- Den einstelligen Booleschen Funktionen 0(), 1(), Id($x_i$) und NOT($x_i$) werden die Terme 0, 1, $x_i$ und $\neg x_i$ zugeordnet.
- Sei f eine n+1-stellige Boolesche Funktion. Dann sind $f_0$ und $f_1$ mit $f_0(x_1, ..., x_n) := f(x_1, ..., x_n, 0)$ und $f_1(x_1, ..., x_n) := f(x_1, ..., x_n, 1)$ n-stellig und besitzen daher Terme $t_0$ und $t_1$.
- Dann wird f definiert durch den Term $(x_{n+1} \wedge t_1) \vee (\neg x_{n+1} \wedge t_0)$.

- **Daher heißt $\{\neg, \wedge, \vee\}$ auch eine *vollständige Basis*.**

# Beispiel (Funktion$\rightarrow$Term)

| $x_1$ | $x_2$ | XOR $(x_1,x_2)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$(\neg x_1 \wedge x_2) \Rightarrow \mathrm{XOR}(x_1,x_2) = 1$

$(x_1 \wedge \neg x_2) \Rightarrow \mathrm{XOR}(x_1,x_2) = 1$

$$\mathrm{XOR}(x_1,x_2) = 1 \Leftrightarrow \underbrace{(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)}_{\text{zugehöriger Term}}$$

# Normalformen von Booleschen Termen

- **Ein Boolescher Term t über den Variablen $\{x_1, ..., x_n\}$ heißt in *disjunktiver Normalform (DNF)* genau dann, wenn t die Form**

$$t = (a_{11} \wedge ... \wedge a_{1n}) \vee ... \vee (a_{k1} \wedge ... \wedge a_{kn})$$

**besitzt, wobei jedes $a_{ij}$ entweder $x_j$ oder $\neg x_j$ entspricht. Ein Teilausdruck der Form $a_{i1} \wedge ... \wedge a_{in}$, heißt auch *Minterm*.**

- **Ein Boolescher Term t über den Variablen $\{x_1, ..., x_n\}$ heißt in *konjunktiver Normalform (KNF)* genau dann, wenn t die Form**

$$t = (a_{11} \vee ... \vee a_{1n}) \wedge ... \wedge (a_{k1} \vee ... \vee a_{kn})$$

**besitzt, wobei jedes $a_{ij}$ entweder $x_j$ oder $\neg x_j$ entspricht. Ein Teilausdruck der Form $a_{i1} \vee ... \vee a_{in}$ heißt auch *Maxterm*.**

- **Anmerkung:**
  **Jeder Minterm bzw. Maxterm enthält alle Booleschen Variablen $\{x_1, ..., x_n\}$ genau einmal, entweder in der Form $x_j$ oder in der negierten Form $\neg x_j$.**

# Konstruktion der disjunktiven Normalform (DNF)

- Sei eine Boolesche Funktion $f:\{0,1\}^n \to \{0,1\}$ in Form einer Wertetafel gegeben.

- Jeder Zeile $(b_1 \dots b_n)$, $b_i \in \{0,1\}$, in der f den Funktionswert 1 hat ($f(b_1, \dots, b_n)=1$), wird ein Minterm $a_1 \wedge \dots \wedge a_n$ zugeordnet, mit

$$a_j = x_j \text{, falls } b_j=1 \quad \text{und } a_j = \neg x_j \text{, falls } b_j=0,$$

  d.h. im Falle einer 1 wird die zugehörige Variable $x_j$ andernfalls deren Komplement $\neg x_j$ eingesetzt.

- Der gesuchte Term t ist die Disjunktion ($\vee$) aller dieser Minterme.

# Beispiel

| $x_1$ | $x_2$ | $x_3$ | S $(x_1,x_2,x_3)$ | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 1 | $\neg x_1 \wedge \neg x_2 \wedge x_3$ |
| 0 | 1 | 0 | 1 | $\neg x_1 \wedge x_2 \wedge \neg x_3$ |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | $x_1 \wedge \neg x_2 \wedge \neg x_3$ |
| 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 1 | $x_1 \wedge x_2 \wedge x_3$ |

**Minterme**

**resultierender Term**

**$S(x_1,x_2,x_3)$:**

$$(\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2 \wedge x_3)$$

$$\overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2\overline{x}_3 + x_1\overline{x}_2\overline{x}_3 + x_1x_2x_3$$ (kompaktere Notation)

# Anmerkungen

- <u>Komplexitätsmaße</u> zur Beurteilung von Termen, z.B.:
  - die *Größe* als die Anzahl der Operatoren
    - Kosten: Chipfläche, Ausschussquote, el. Leistungsaufnahme, …
  - die *Tiefe* als Maß für die Auswertungszeit.
    - Leistung: Max. Prozessortakt, max. Durchsatz in einem Router, …
- Anmerkungen
  - Die durch DNF oder KNF beschriebenen Normalformen-Terme sind zwar prinzipiell für einen Schaltungsentwurf nutzbar, jedoch i.d.R. nicht minimal in Hinblick auf den Aufwand zur Realisierung.
  - Gesucht werden daher für die praktische Realisierung äquivalente Minimalformen von Termen, die aus der Normalform hergeleitet werden können.
  - Hierzu existieren Algorithmen (z.B. Karnaugh/Veitch, Quine/McCluskey, heuristische Verfahren), auf die hier nicht näher eingegangen wird. → KV-Diagramme

# 5.3 Schaltnetze

- **Motivation:**
  **Boolesche Terme können einen wesentlichen Aspekt der technischen Realisierung nicht angemessen modellieren, nämlich die Mehrfachverwendung bereits ermittelter "Zwischenergebnisse".**
- **Diesen Mangel beheben *Schaltnetze*, auch *kombinatorische Schaltwerke* oder *lineare Schaltungen* genannt.**
- **Schaltnetze sind sehr anschauliche Graphen.**
  - **Die verwendeten graphischen Symbole für Boolesche Funktionen sind durch DIN 40900/12 genormt.**
  - **Daneben existiert eine ebenfalls weit verbreitete Notation als US ANSI-Norm.**

# Definition: Schaltnetz

Def

- Ein ***Schaltnetz*** **ist ein gerichteter, zyklenfreier Graph, dessen Knoten von einem der Typen (a)-(e) sind. Die Knoten werden so angeordnet, dass die verbindenden Kanten "von links nach rechts" verlaufen und daher keine Pfeilspitzen benötigen.**

  **(a) Eingangs-Knoten:**
  - mit Markierung aus $\{0, 1, x_1, ..., x_n\}$,
    d.h. Konstante oder Boolesche Eingangsvariable,
  - nur ausgehende Kanten

  0 •—

  1 •—

  $x_i$ •—

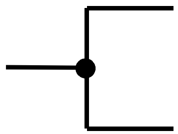
  **(b) Ausgangs-Knoten:**
  - mit Markierung aus $\{y_1, ..., y_m\}$,
    jede Ausgangsvariable muss genau einmal vorkommen,
  - nur einmündende Kanten

  —• $y_j$

# Definition: Schaltnetz (2)

(c) Verzweigungsknoten:

- eine eingehende Kante, zwei oder mehr ausgehende Kanten dienen der Verteilung eines Signals, z.B.



(d) unäre Gatter:

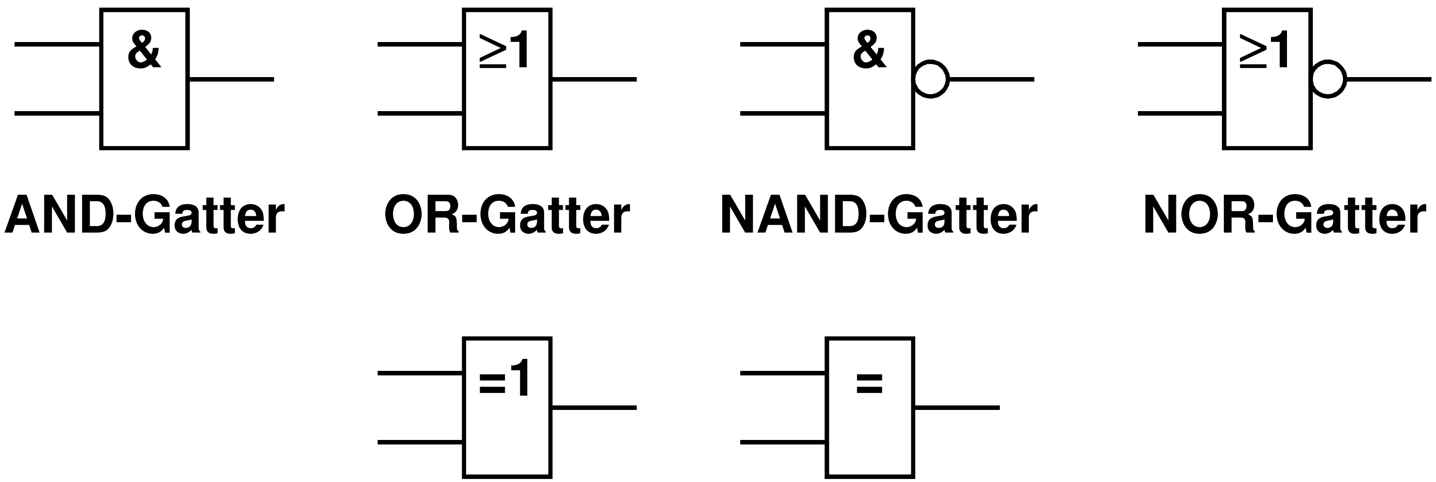
- eine eingehende Kante, eine ausgehende Kante
- NOT-Gatter mit Markierung 1 und O am Ausgang
- Id-Gatter mit Markierung 1 (Identität) (kaum Bedeutung)


NOT-Gatter Id-Gatter

# Definition: Schaltnetz (3)

(e) 2-stellige logische Gatter:

- zwei eingehende Kanten, eine ausgehende Kante
- Markierungen vgl. Symbole



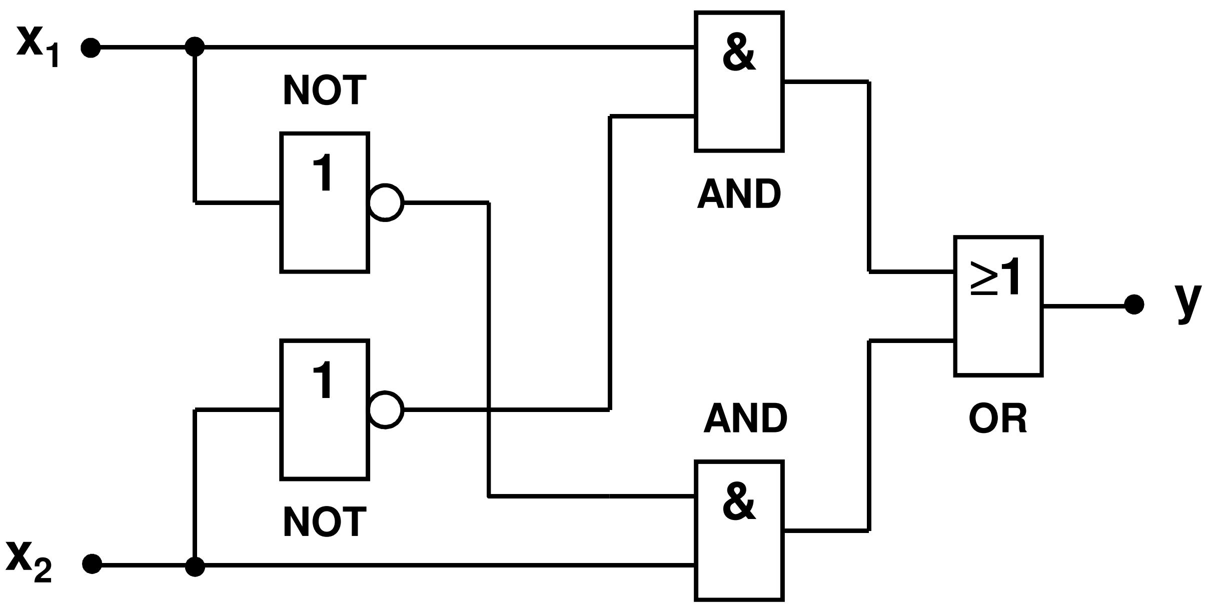

XOR-Gatter EQUIV-Gatter

# Beispiel 1: XOR

**XOR als Schaltnetz basierend auf NOT, AND und OR-Gattern:**

| $x_1$ | $x_2$ | XOR $(x_1,x_2)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$$y = \mathrm{XOR}(x_1,x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

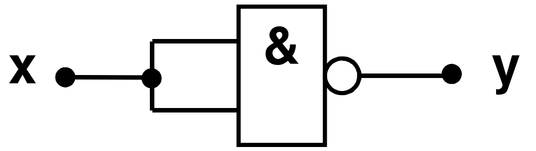



**Hier: Direkte Umsetzung der DNF für XOR**

# Beispiel 2: Universalität des NAND-Gatters

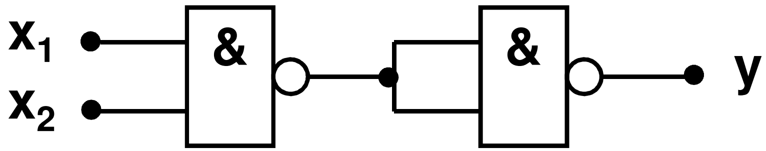
**Schaltnetze für NOT, AND und OR basierend auf dem NAND-Gatter:**

$$y = \mathrm{NOT}(x) = \overline{x \wedge x}$$



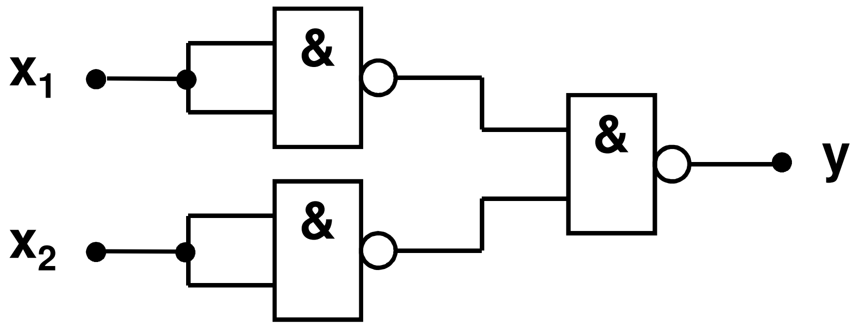

$$y = \mathrm{AND}(x_1,x_2) = \overline{\overline{x_1 \wedge x_2}}$$

$$= \overline{\overline{(x_1 \wedge x_2)} \wedge \overline{(x_1 \wedge x_2)}}$$




$$y = \mathrm{OR}(x_1,x_2) = \overline{\overline{x_1 \vee x_2}} = \overline{\overline{x_1} \wedge \overline{x_2}}$$

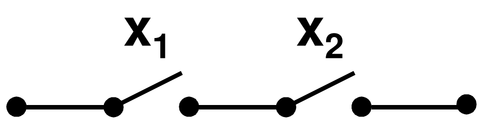
$$= \overline{\overline{(x_1 \wedge x_1)} \wedge \overline{(x_2 \wedge x_2)}}$$



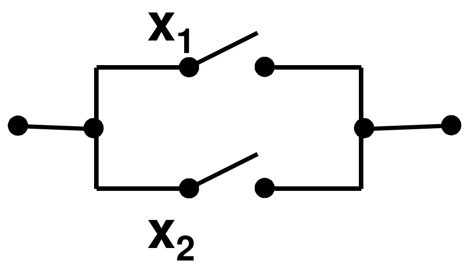
# Technische Realisierung von Gattern

- **Schalter:**

**AND-Verknüpfung als Serienschaltung**




**OR-Verknüpfung als Parallelschaltung**

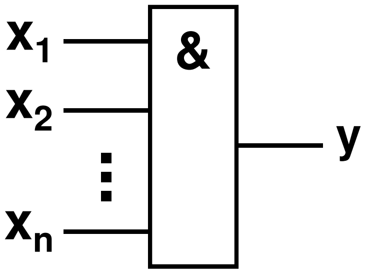


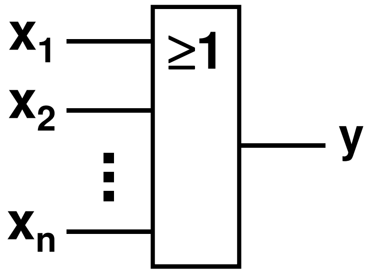

- **In elektronischen Schaltungen werden Gatter i.d.R. durch Transistoren realisiert.**
- **In integrierten Schaltkreisen (ICs) befinden sich heute viele Millionen von Transistoren.**

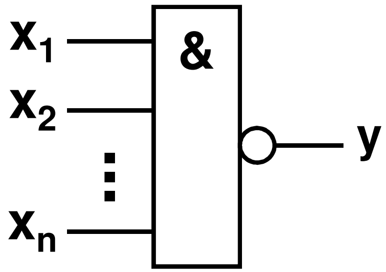
# Erweiterungen der Notation:

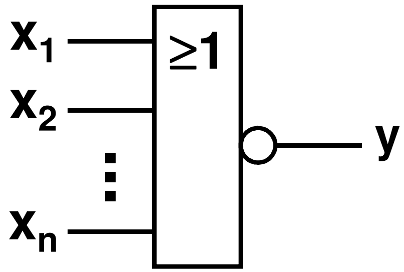
- **Gatter mit n Eingängen (vgl. 5.2):**

| n-faches AND-Gatter | n-faches OR-Gatter | n-faches NAND-Gatter | n-faches NOR-Gatter |
|---|---|---|---|



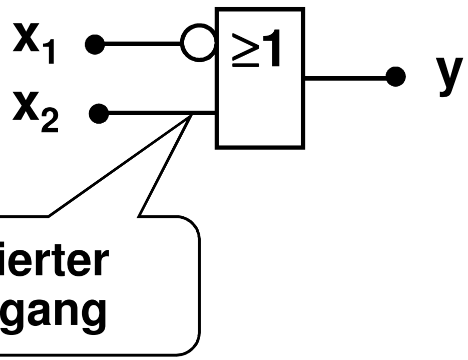




$y = x_1 \wedge x_2 \wedge \ldots \wedge x_n$ $\quad y = x_1 \vee x_2 \vee \ldots \vee x_n$ $\quad y = \overline{x_1 \wedge x_2 \wedge \ldots \wedge x_n}$ $\quad y = \overline{x_1 \vee x_2 \vee \ldots \vee x_n}$

- **Negation an Eingängen:**

  **Beispiel (Implikation):**

  $y = (x_1 \Rightarrow x_2) = \neg x_1 \vee x_2$



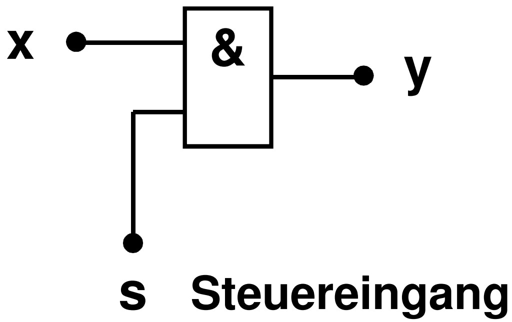
# Wichtige Schaltnetze

- **Im Folgenden:**
  - Beispiele praktisch relevanter Schaltnetze
  - teilweise noch als separate Bausteine (Chips) gefertigt oder Teil eines hochintegrierten Bausteins
  - Insbesondere im Inneren eines Prozessors verwendet
- **Übersicht**
  - Tore
  - Encoder
  - Decoder
  - Multiplexer
  - Demultiplexer
  - Halbaddierer
  - Volladdierer
  - Arithmetisch-logische Einheit (ALU)

# Tore

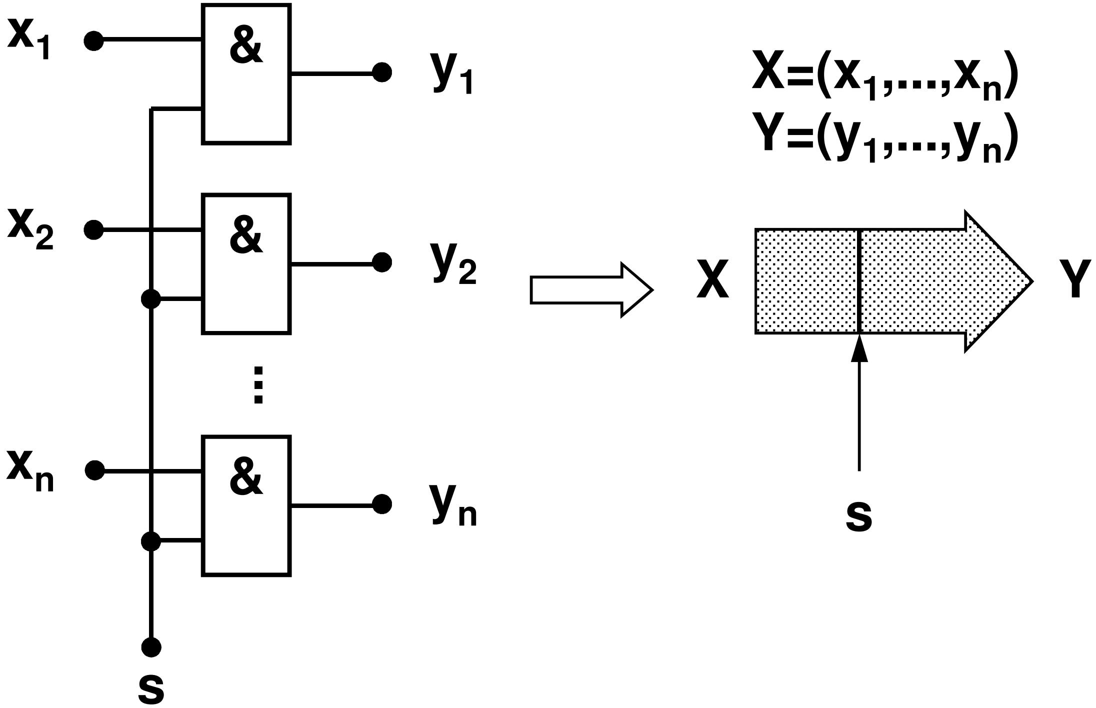
- **kontrollierte Durchleitung (Tor) eines Eingangs oder einer Menge von Eingängen**
  - **Nutzung eines AND-Gatters**
  - **Unterscheidung von Daten- und Steuereingängen**

**Tor für Einzelsignal**




$$y = \begin{cases} x & \text{falls } s=1 \\ 0 & \text{sonst} \end{cases}$$

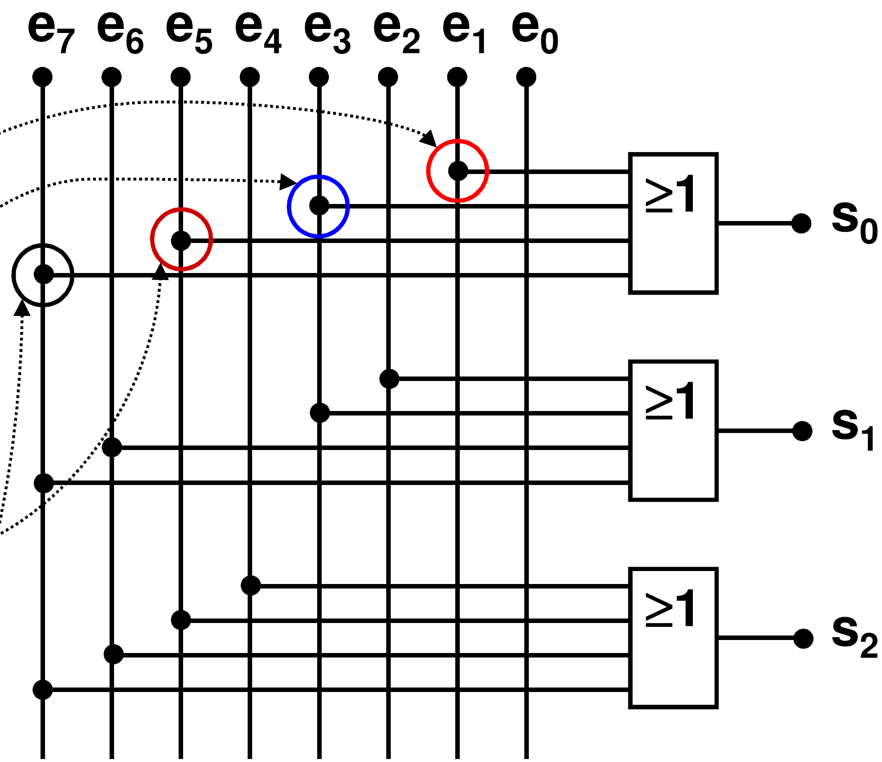
**Tor für Signalgruppe (z.B. Bus)**



# Encoder

- 1-aus-n Code am Eingang wird in einen dichten Code am Ausgang codiert
- Beispiel: n=8  (Ansatz auf beliebiges n übertragbar)

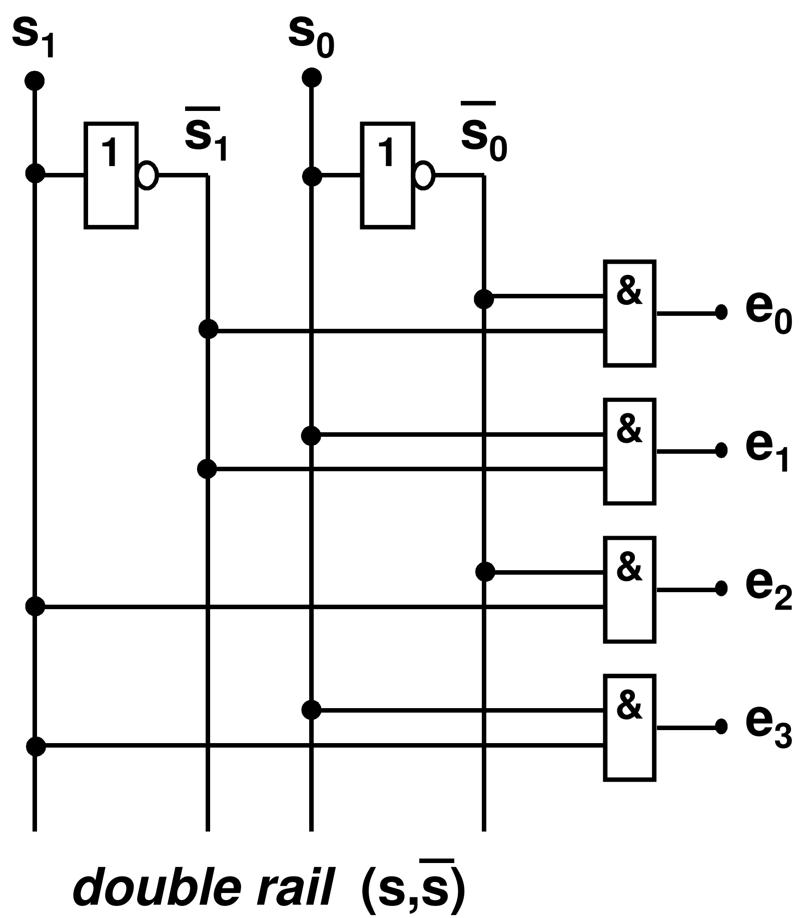
| $e_7$ | ... | $e_2$ | $e_1$ | $e_0$ | $s_2$ | $s_1$ | $s_0$ |
|---|---|---|---|---|---|---|---|
| 0 | ... | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | ... | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | ... | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | ... | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | ... | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | ... | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | ... | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | ... | 0 | 0 | 0 | 1 | 1 | 1 |



# Decoder

- Auswahl eines Ausgangs, Gegenstück zum Encoder
- n-Bit Dualzahl am Eingang wird decodiert in 1-aus-$2^n$ am Ausgang
- Beispiel: n=2

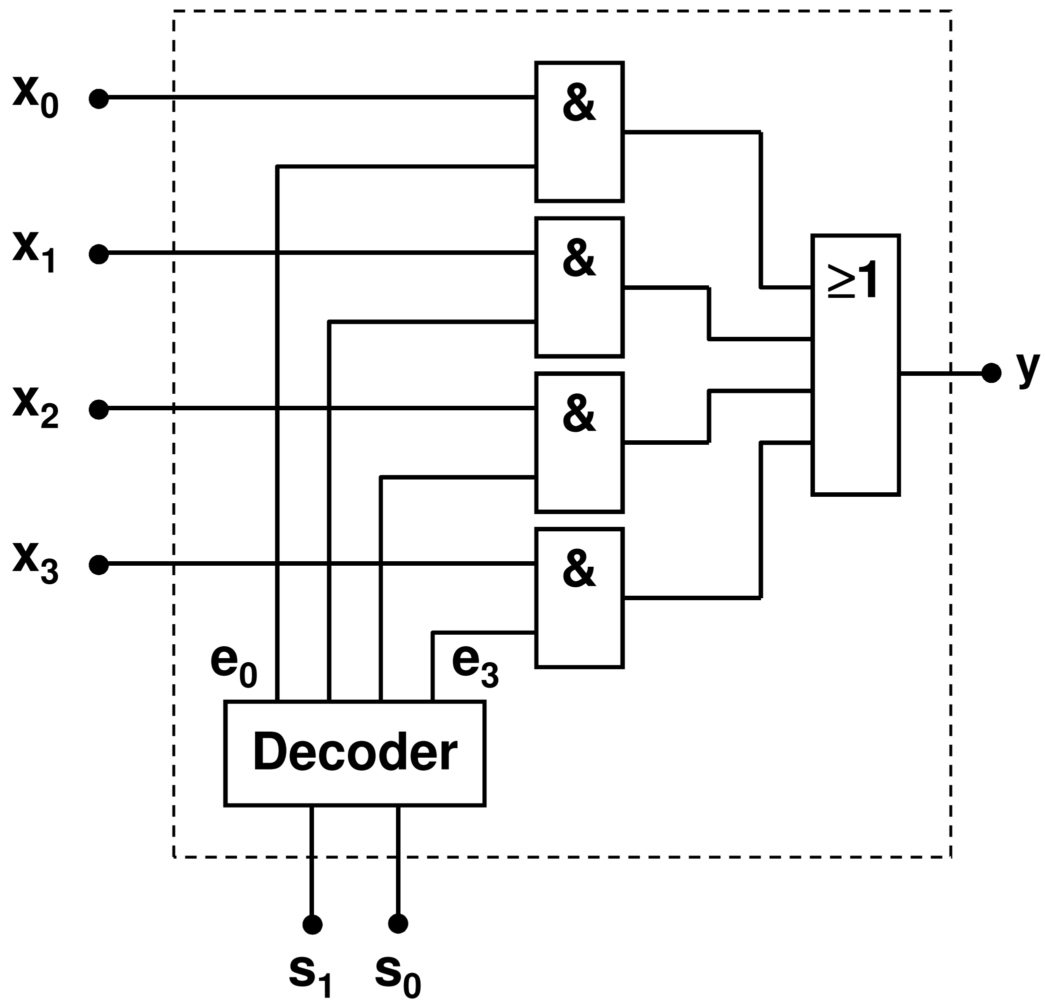
| $s_1$ | $s_0$ | $e_0$ | $e_1$ | $e_2$ | $e_3$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |




Ansatz auf
n Stellen erweiterbar

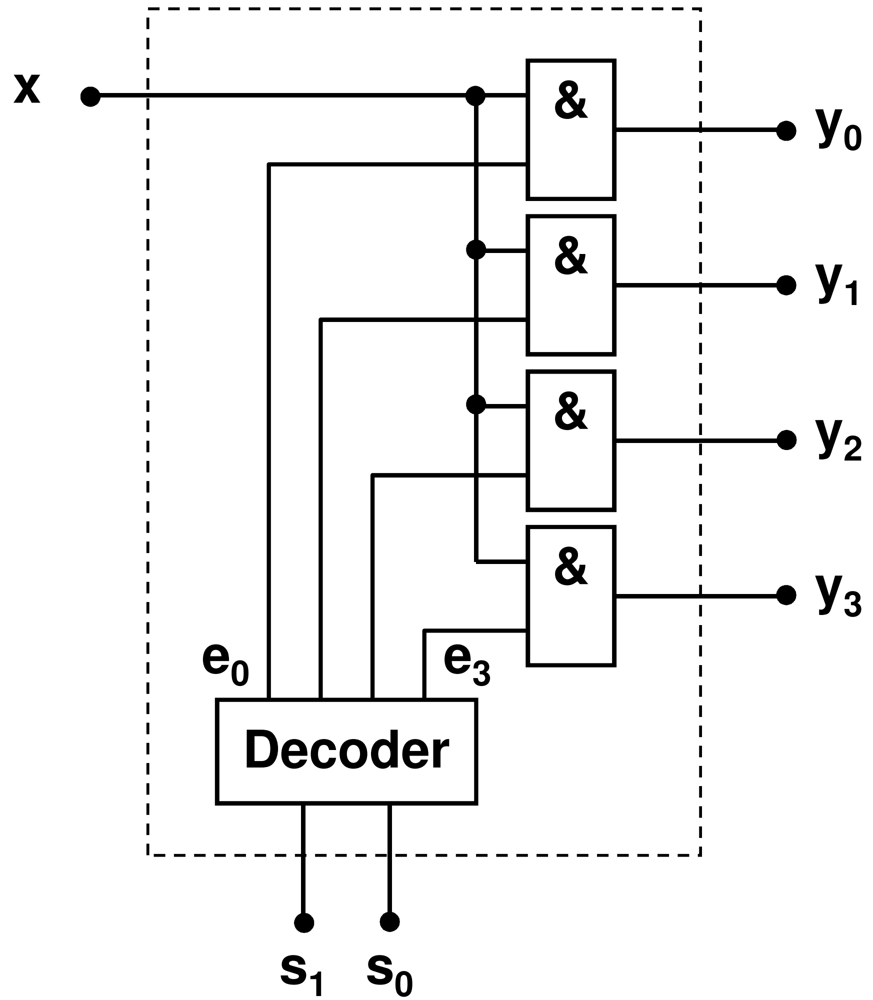
# Multiplexer

- Durchschalten eines von n Eingängen auf den (einzigen) Ausgang
- Auswahl des Eingangs über Steuereingänge, z.B. dual codiert
- Nutzung von Tor und Decoder
- Beispiel: n=4



# Demultiplexer

- Gegenstück zum Multiplexer
- Durchschalten (Verteilen) des (einen) Eingangs auf einen von n Ausgängen
- Auswahl des Ausgangs über Steuereingänge, z.B. dual codiert
- Nutzung von Tor und Decoder
- Beispiel: n=4



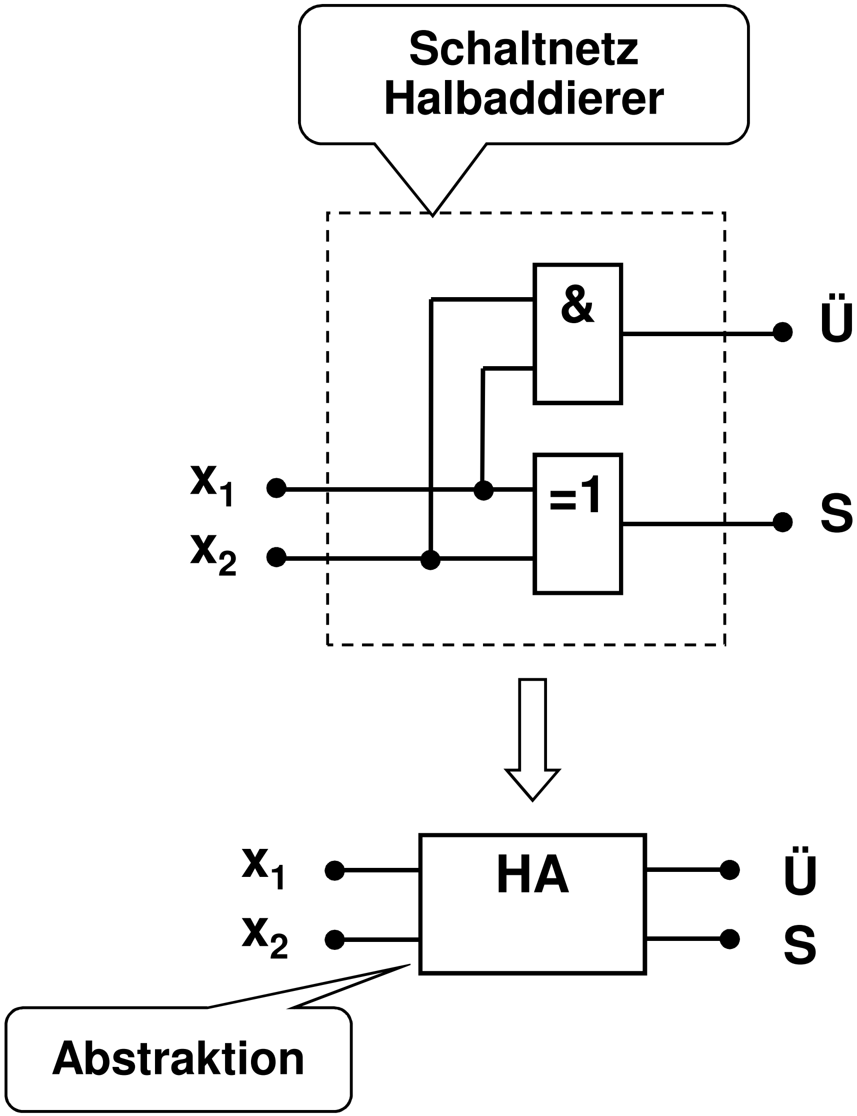
# Halbaddierer

- **Addition zweier Bits:**

| $x_1$ | $x_2$ | S Sum | Ü Carry |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

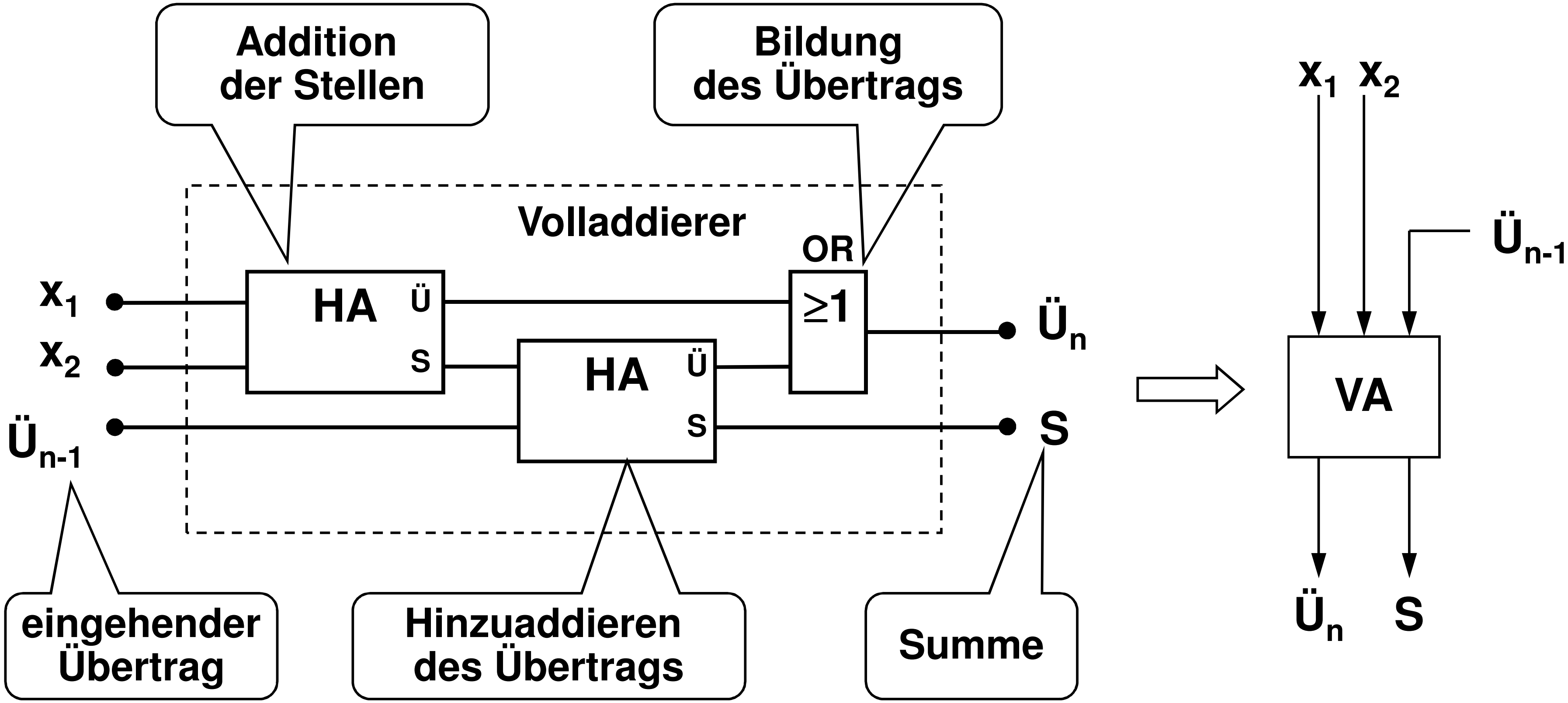
**$S = XOR(x_1,x_2)$ (Summe)**

**$Ü = AND(x_1,x_2)$ (Übertrag, Carry)**



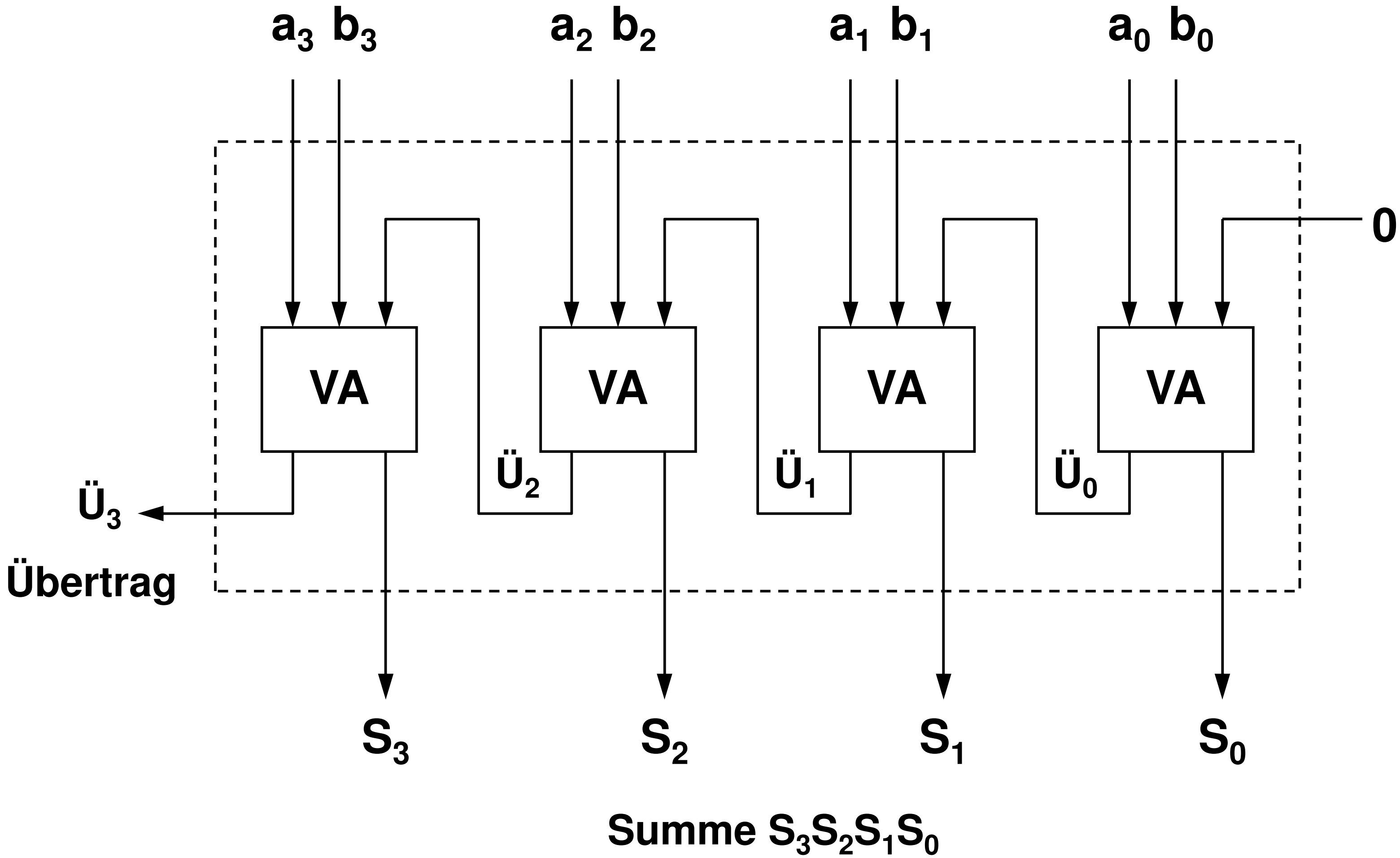
# Volladdierer

- **Addition zweier Bits mit Berücksichtigung des Übertrags der niederwertigeren Stelle:**



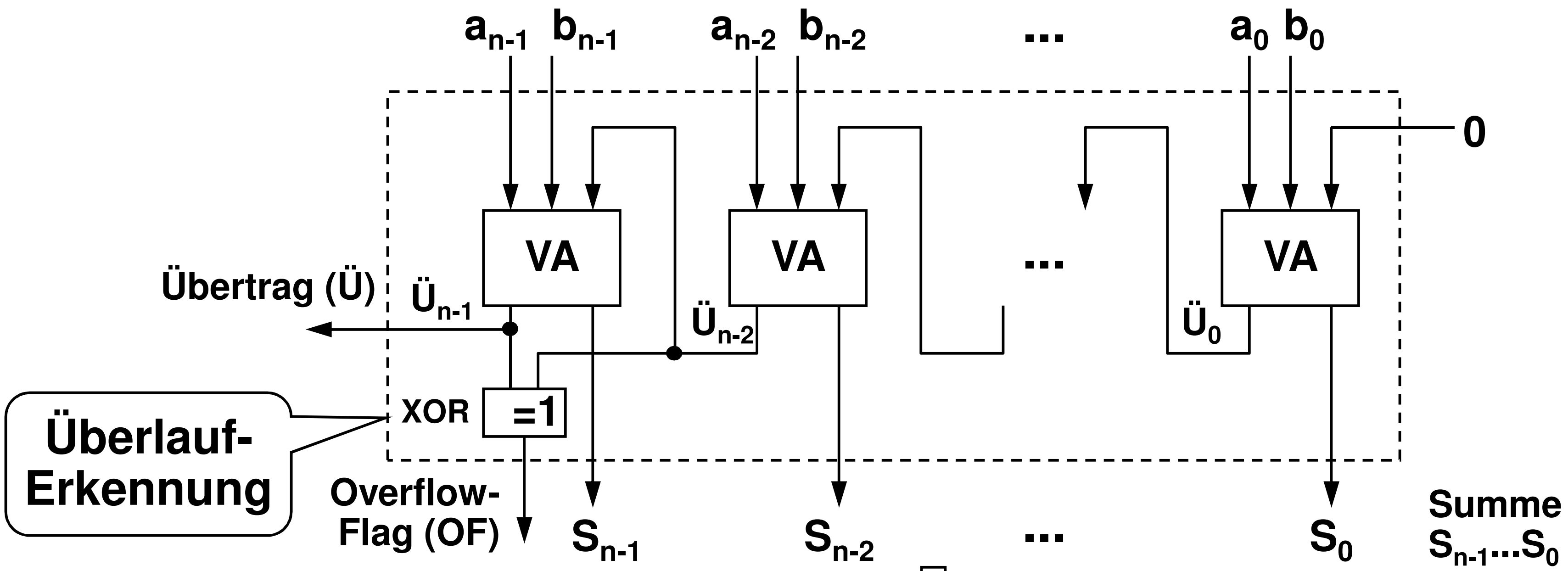
# Paralleladdierer

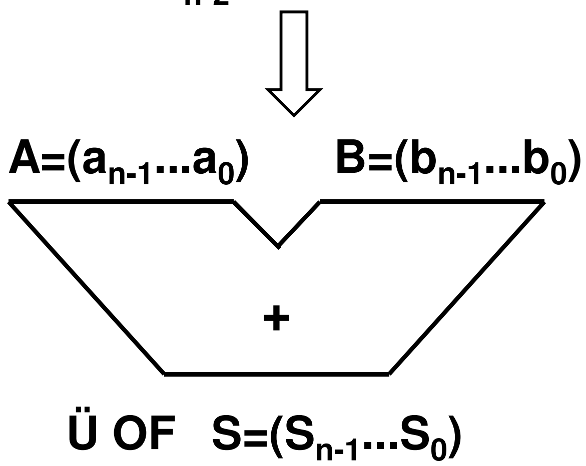
- **Schaltnetz zur Addition von 4-Bit-Dualzahlen $a_3a_2a_1a_0$ und $b_3b_2b_1b_0$ aus 4 Volladdierern:**



# Paralleladdierer für n-Bit-Maschinenwörter

- **Prinzipiell: Erweiterung auf eine beliebige Maschinenwortlänge n.**



# Ripple Carry / Carry-Look-Ahead

- **Problem des n-Bit-Paralleladdierers:**

  **Die Addition dauert mit jedem zusätzlichen Volladdierer länger, da das endgültige Ergebnis erst dann vorliegt, wenn die Überträge von rechts nach links vollständig verarbeitet sind (hohe Tiefe durch *Ripple Carry*).**

  **Beispiel:**

  **Ein Addierer benötige 1 CPU-Takt, 64-bit-CPU, dann:**
  **Addition zweier *unsigned int* dauert 64 Takte bis Carry gültig!**

- **Beschleunigung der Addition ist durch ein *Carry-Look-Ahead*-Schaltnetz möglich. Dabei werden alle Überträge gleichzeitig und unmittelbar aus den Eingangsgrößen errechnet (Preis: Größerer Hardware-Aufwand in Anzahl Gattern).**

# Carry-Look-Ahead

- **Idee: Berechne Übertrag vor dem Addieren**
- **Nutze aus, dass jedes Bitpaar entweder**
  - **einen Übertrag erzeugt (1, 1) – Funktion „generate“:** $g_i = a_i \wedge b_i$
  - **einen Übertrag weiterleitet (1, 0)/(0, 1) – „propagate“:** $p_i = a_i \vee b_i$
  - **ein Übertrag** $c_i$ **ergibt sich, wenn** $g_i \vee (p_i \wedge c_{i-1})$

**für i = 0:** $c_0 = a_0 \wedge b_0 = g_0$

**für i > 0:**

$$\begin{aligned} c_i &= g_i \vee (p_i \wedge c_{i-1}) \\ &= (a_i \wedge b_i) \vee ((a_i \vee b_i) \wedge c_{i-1}) \\ &= (a_i \wedge b_i) \vee (a_i \wedge c_{i-1}) \vee (b_i \wedge c_{i-1}) \end{aligned}$$

# Carry-Look-Ahead

- **Für den 4-Bit-Addierer:**

  $c_0 = a_0 \wedge b_0 = g_0$

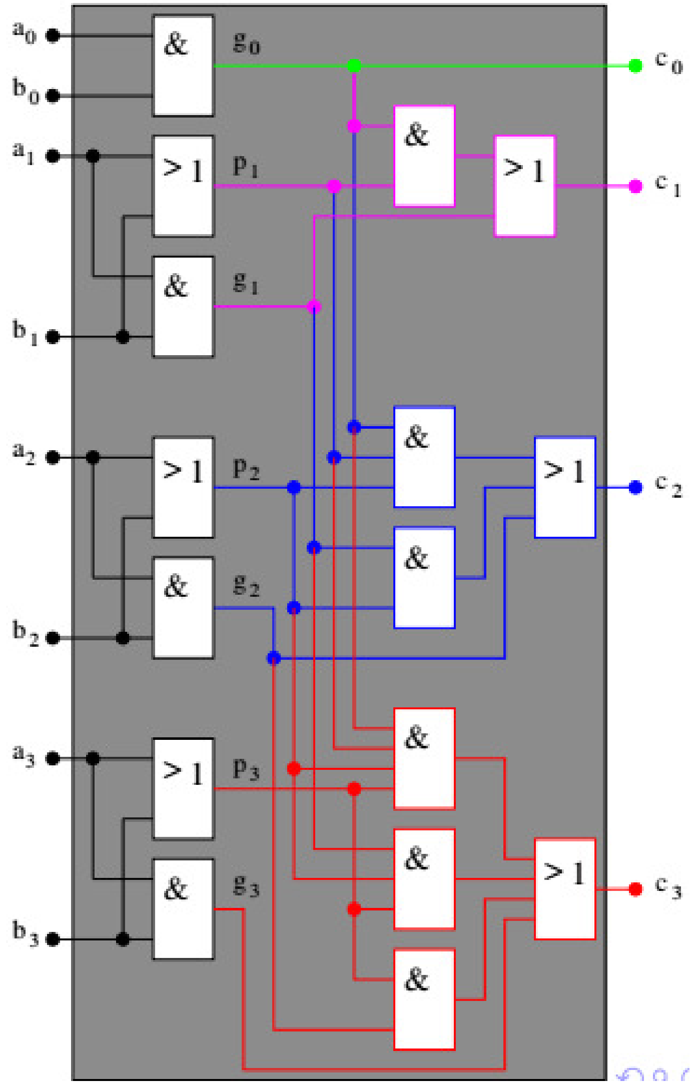
  $c_1 = g_1 \vee (p_1 \wedge c_0) = g_1 \vee (p_1 \wedge g_0)$

  $c_2 = g_2 \vee (p_2 \wedge c_1) = g_2 \vee (p_2 \wedge g_1 \vee (p_1 \wedge g_0))$
  $= g_2 \vee (p_2 \wedge g_1) \vee (p_2 \wedge p_1 \wedge g_0)$

  $c_3 = g_3 \vee (p_3 \wedge c_2)$
  $= g_3 \vee (p_3 \wedge g_2 \vee (p_2 \wedge g_1) \vee (p_2 \wedge p_1 \wedge g_0))$
  $= g_3 \vee (p_3 \wedge g_2) \vee (p_3 \wedge p_2 \wedge g_1)$
  $\vee (p_3 \wedge p_2 \wedge p_1 \wedge g_0)$

  – **Maximale Tiefe: 3**
  – **Maximaler Grad (Gatterinputs): 4**

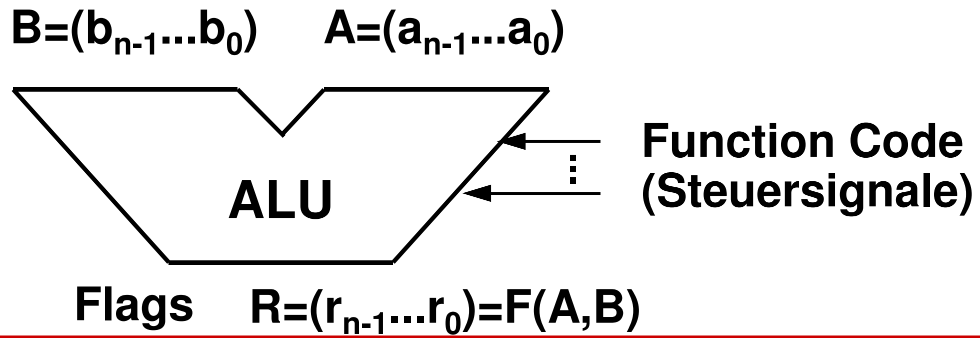
- **16- und 64-Bit-Addierer:**
  – **Durch Kaskadenbildung erreichbar**




Quelle: R. Kaiser, M. Gergeleit. TGI WS 2014/15

# Arithmetisch-logische Einheit (ALU)

- **Eine *arithmetisch-logische Einheit* (*Arithmetical Logical Unit*, *ALU*) ist ein Schaltnetz, das**
  - **die wesentliche Komponente eines jeden Prozessors ist**
  - **als Kern einen Paralleladdierer enthält**
  - **andere Operationsarten durch zusätzliche Gatter realisiert, wie:**
    - **logische Operationen wie AND, OR, XOR, usw.**
    - **Subtraktion, Shift-Operationen, …**
  - **Auswahl der Operation F erfolgt über Steuersignale (*Function Code*)**
  - **außer Ergebnis R (Result) werden *Flags* erzeugt, die Fehlersituationen (z.B. Überlauf) und Aussagen über das Ergebnis (z.B. =0, <0, >0, Übertrag) angeben.**

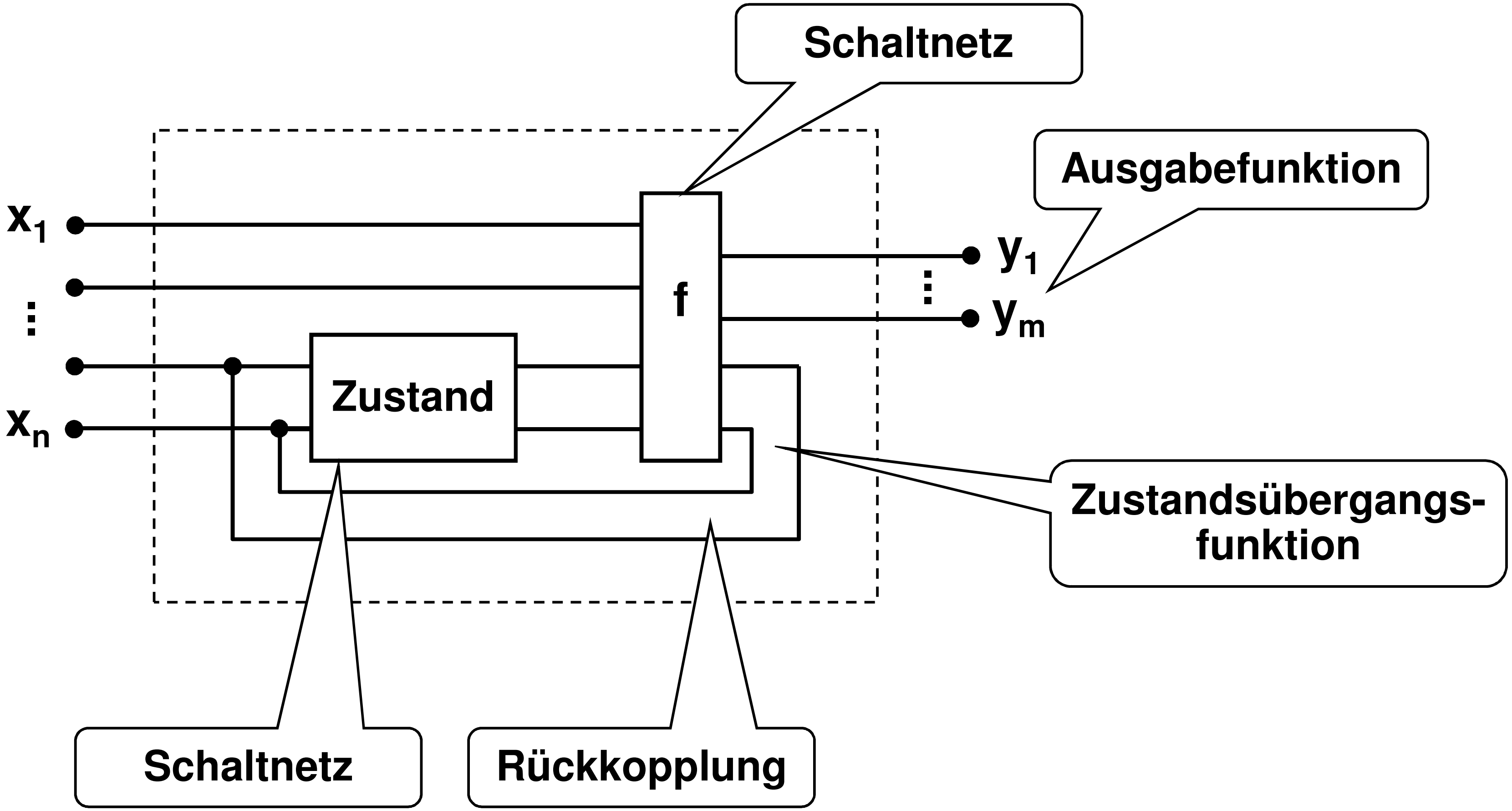


# 5.4 Schaltwerke

- **Motivation:**
  **Die bisher betrachteten Schaltnetze können zwar beliebige Boolesche Funktionen berechnen, können aber keine Werte *speichern*.**

- **Diese Möglichkeit entsteht, wenn die Zyklusfreiheit der Schaltnetze beschreibenden Graphen fallengelassen wird. Derartige Graphen, also Schaltnetze mit *Rückkopplungen*, die Ausgänge wieder auf Eingänge führen, werden *Schaltwerke* oder *sequentielle Schaltwerke* genannt.**

# Prinzip eines Schaltwerks



# Anmerkungen

- Typisch für Schaltwerke ist, dass durch Gatter-Signallaufzeiten *zeitlich verzögerte* Ausgänge als Eingangswerte erscheinen.

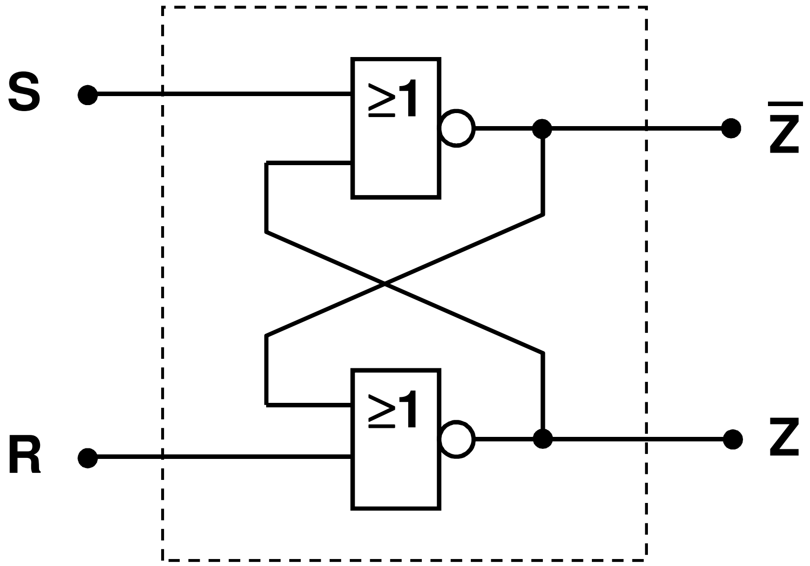
  ⇒ Eingänge und Ausgänge zu diskreten *Zeitpunkten* betrachtet.
- Rückgekoppelte Signale können eine Wirkungsfolge (Sequenz) im Schaltnetz auslösen. Dabei können letztlich entstehen:
  - stabile Zustände: Rückkopplungsausgänge ändern sich nicht weiter
  - instabile Zustände: Rückkopplungsausgänge führen zu fortwährenden Änderungen an den Eingängen.

  ⇒ Die mit instabilen Zuständen verbundenen komplexen Vorgänge interessieren hier nicht.

- Der Zustandsbegriff ist von zentraler Bedeutung.
- Die Zustandübergangsfunktion entspricht einem deterministi-schen endlichen Automaten (vgl. Vorlesung Informatik 2).

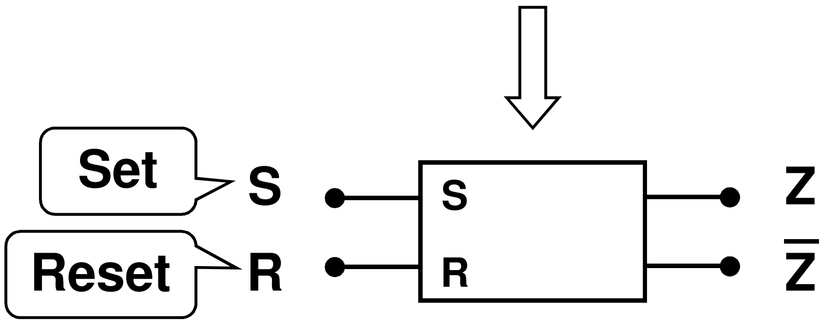
# Asynchrones RS-Flip-Flop (Latch)

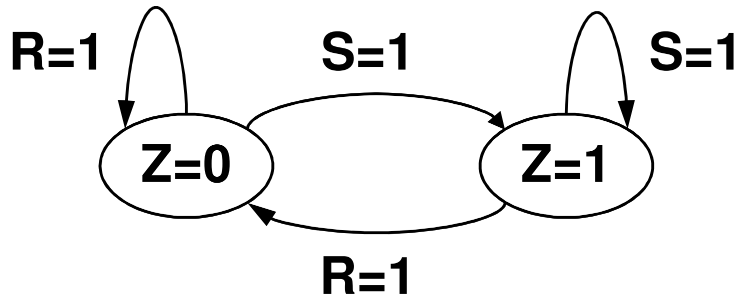
- **RS-Flip-Flop als einfache *bistabile Kippstufe* aus zwei rückgekoppelten NOR-Gattern:**




| R | S | $Z^{t+1}$ | $\overline{Z}^{t+1}$ | Funktion |
|---|---|---|---|---|
| 0 | 0 | $Z^t$ | $\overline{Z}^t$ | Speichern |
| 0 | 1 | 1 | 0 | Setzen |
| 1 | 0 | 0 | 1 | Löschen |
| (1) | (1) | - | - | - |

**R=1 S=1 ist unzulässig**



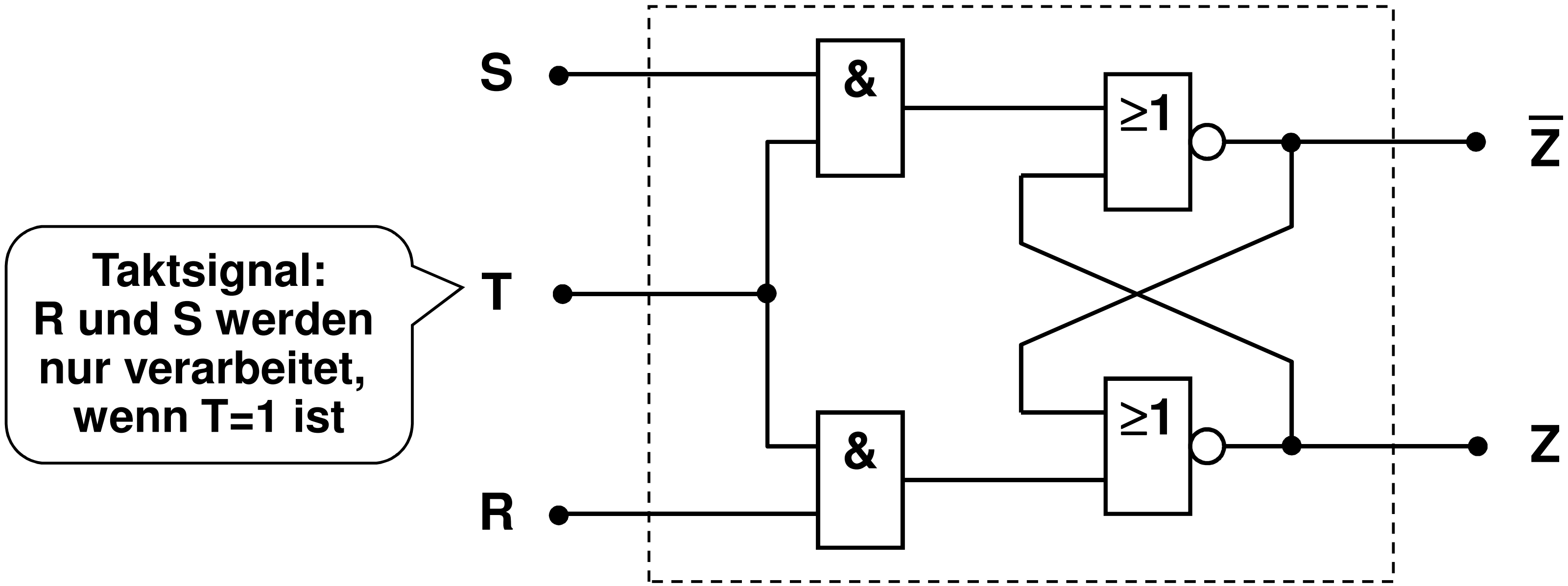



# Anmerkungen

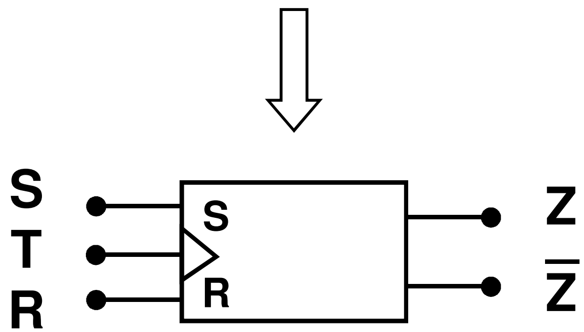
- Es ist oft wünschenswert, dass eine an einem Eingang anliegende Information nur zu einem bestimmten Zeitpunkt verarbeitet werden soll. Ein solcher Zeitpunkt wird ***Takt (Clock)*** genannt.
- Ein Takt vereinfacht das Denken:
  - Abstraktion von komplexen zeitbezogenen Einschwingvorgängen, die abhängig von äußeren Bedingungen, Fertigungstoleranzen, usw. sein können.
  - Verzögerungszeiten (Gatterlaufzeiten) werden irrelevant, d.h. *Wettläufe* zwischen Signalen (*Race Conditions*) werden vermieden.
  - Verhalten wird zu diskreten Zeitpunkten betrachtet.
- ***Synchrone Schaltwerke*** sind solche, die auf einem Takt zur Verarbeitung basieren (weit verbreitet).
- ***Asynchrone Schaltwerke*** besitzen keinen Takt.
- Beispiel: Das bisher behandelte RS-Flip-Flop
  ist ein asynchrones Schaltwerk.

# Synchrones RS-Flip-Flop (Gated Latch)

- **asynchrones RS-Flip-Flop mit zusätzlichem Takteingang T**

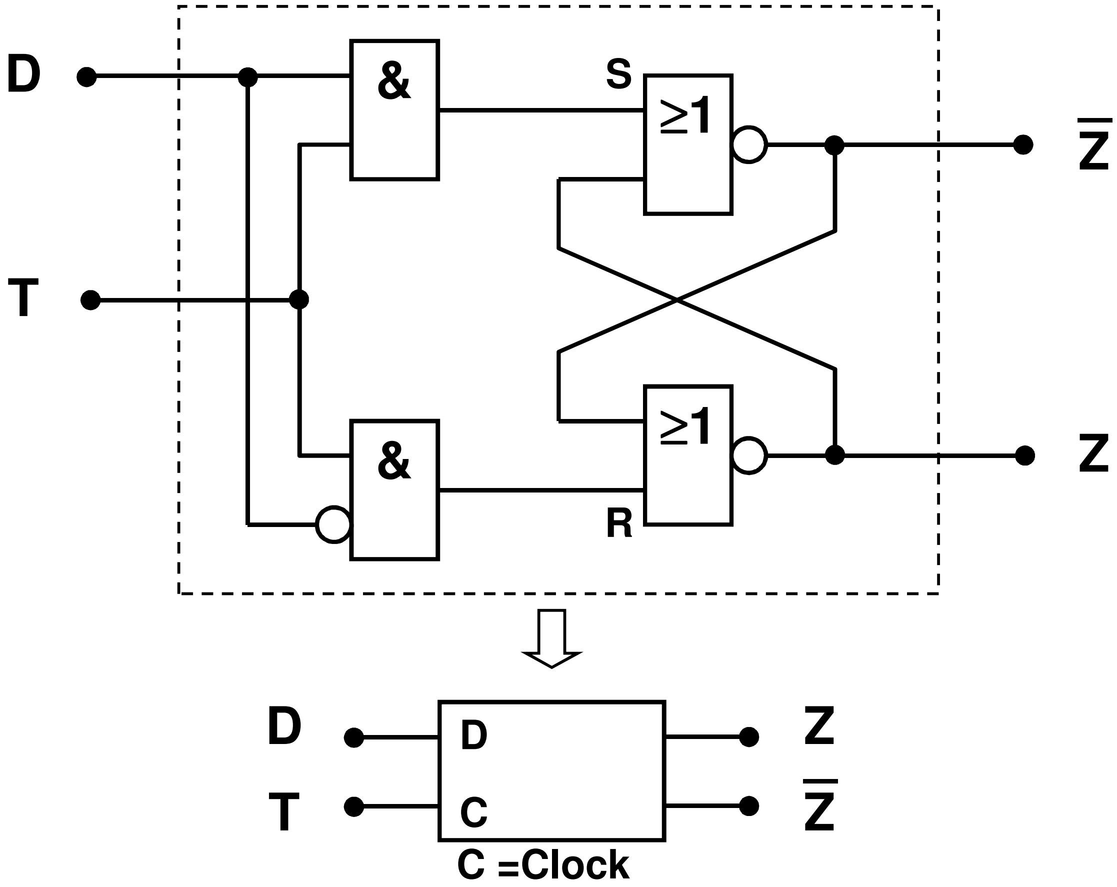



- **Bemerkung:**
  **Reale Flip-Flops verwenden Signal<u>flanken</u> zur präzisen Definition des Verarbeitungszeitpunkts**



# D-Flip-Flop (Data Latch)

- Variation des synchronen RS-Flip-Flop: Nur ein Eingang (Data)
- Vermeidung der verbotenen Eingabe R=S=1
- Der zum Taktsignal vorliegende Eingabewert D wird gespeichert.




| T | D | $Z^{t+1}$ | $\overline{Z}^{t+1}$ |
|---|---|---|---|
| 0 | x | $Z^t$ | $\overline{Z}^t$ |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

x: don't care (egal)

# Wichtige Schaltwerke

- In einem Flip-Flop kann ein einzelnes Bit gespeichert werden.
- Wichtige Schaltwerke sind Zusammenfassungen von mehreren Flip-Flops und Schaltnetzen unter funktionalen Gesichtspunkten.
- Sie werden teilweise als separate Bausteine (Chips) gefertigt oder sind Teil eines hochintegrierten Bausteins. Insbesondere werden sie z.B. im Inneren eines Prozessors verwendet.
- Übersicht
  - Register
  - Schieberegister
  - Zähler

# Register

- Zusammenfassung von Flip-Flops mit gemeinsamem Takt und Toren
- Verwendung z.B. als Prozessorregister mit Wortbreite n