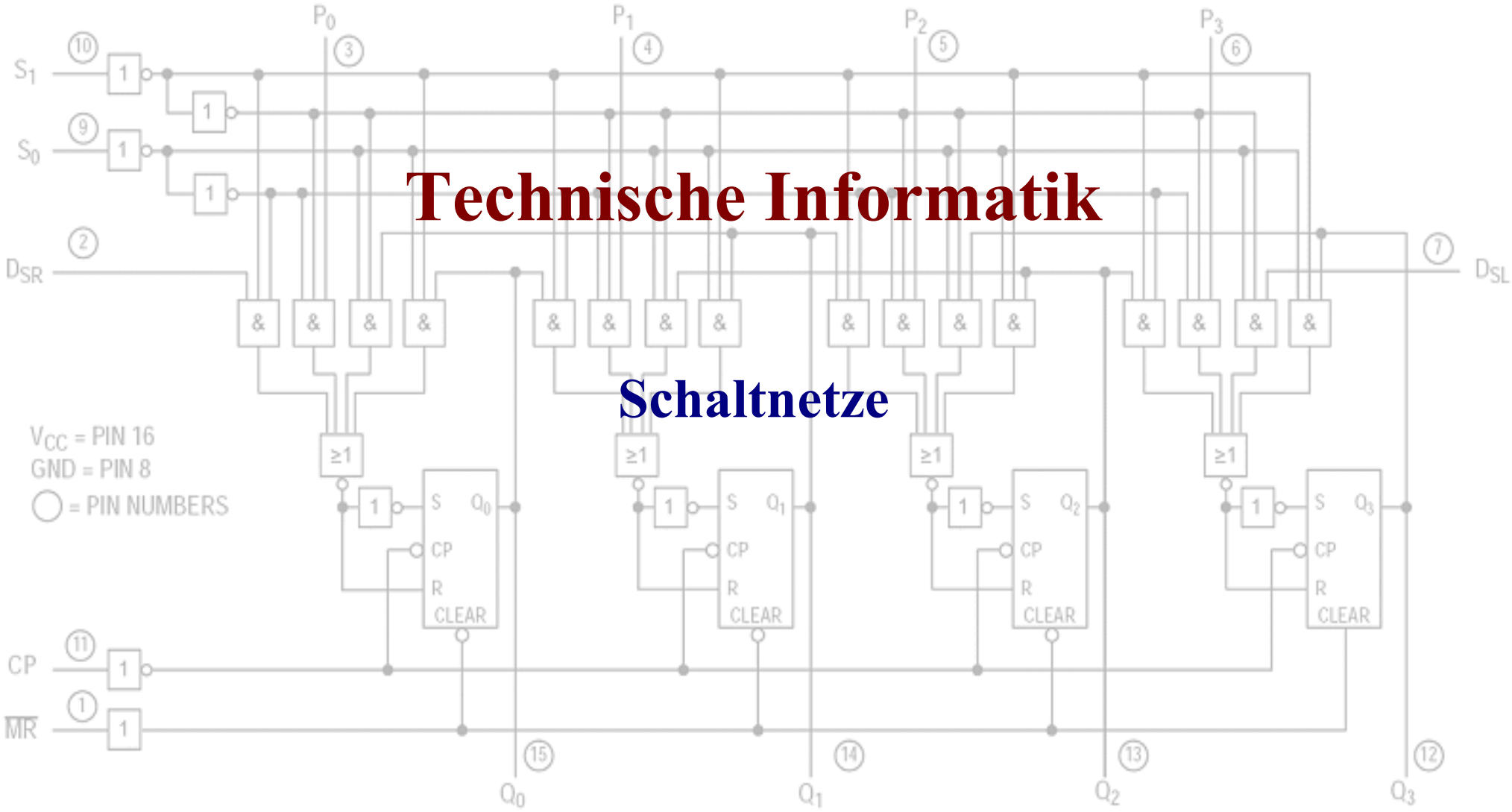


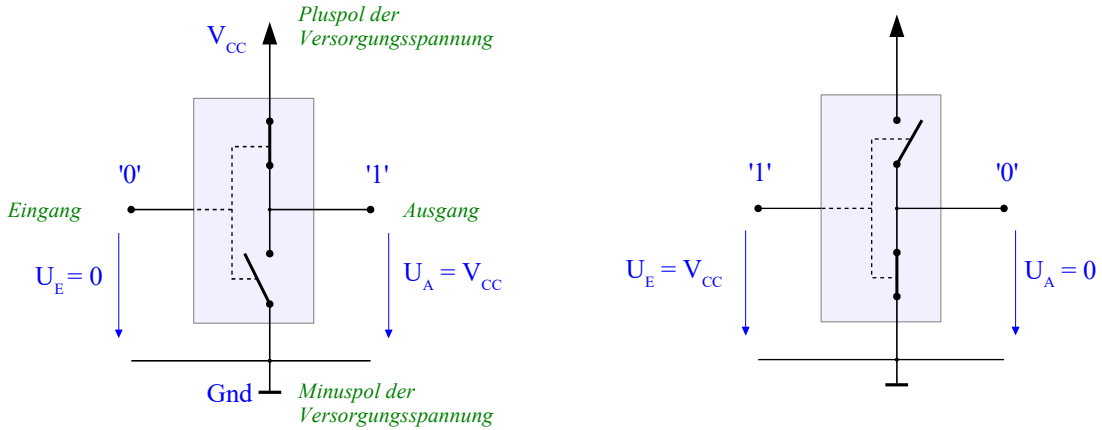
Technische Informatik

Schaltnetze

Logikgatter

- Schaltnetz = Technische Realisierung einer Boolscher Funktionen
- Logikgatter = Komponente eines Schaltnetzes, die eine Boolsche Operation realisiert
  - Elemente '0' und '1' werden durch physikalische Zustände repräsentiert
  - Ein- und Ausgänge eines Logik-Gatters werden gemäß einer Boolschen Operationen verknüpft
  - Der Ausgang eines Logik-Gatters kann den Eingang eines weiteren Logik-Gatters definieren
- Technologie
  - Beliebige Domäne möglich, auch domänenübergreifend
  - z.B. elektronisch (Halbleiter, Röhre), elektrisch (Relais), pneumatisch (Ventil), mechanisch ...

- CMOS-Technologie
  - Zwei Transistoren als Schaltelemente mit komplemetären Schaltzuständen
  - Zunächst:
    - Spannung U = 0 <=> logisch '0'
    - Spannung U = V_{CC} <=> logisch '1'



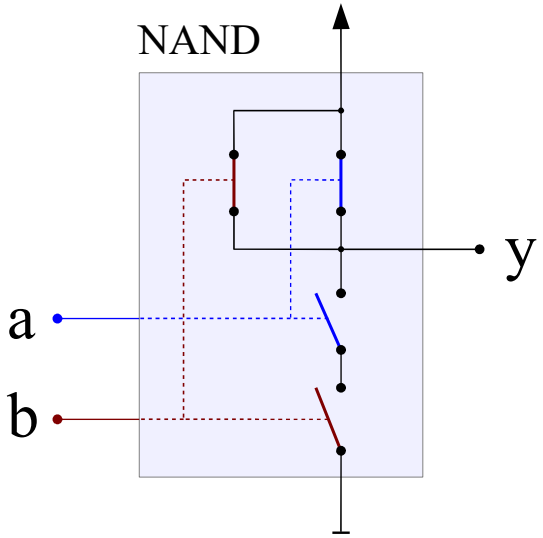
- Funktionsweise:

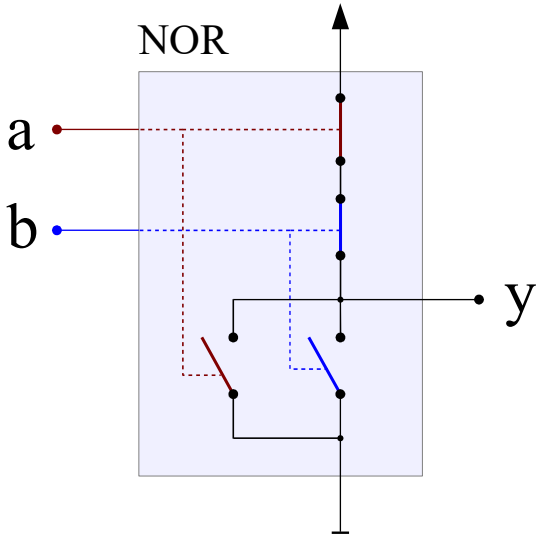
$U_{Eingang}$	Transistor		$U_{Ausgang}$
	unten	oben	
0	offen	geschl	V_{CC}
V_{CC}	geschl	offen	0

  - Diese Schaltung implementiert also die NOT-Operation ("Inverter")

Logikgatter

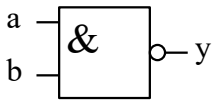
- Durch Kombination komplementärer Transistoren können beliebige Boolsche Operationen realisiert werden:
  - Schaltungsprinzip:

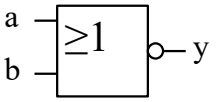




  - Boolsche Operation: $y = \overline{a \cdot b}$ $y = \overline{a + b}$

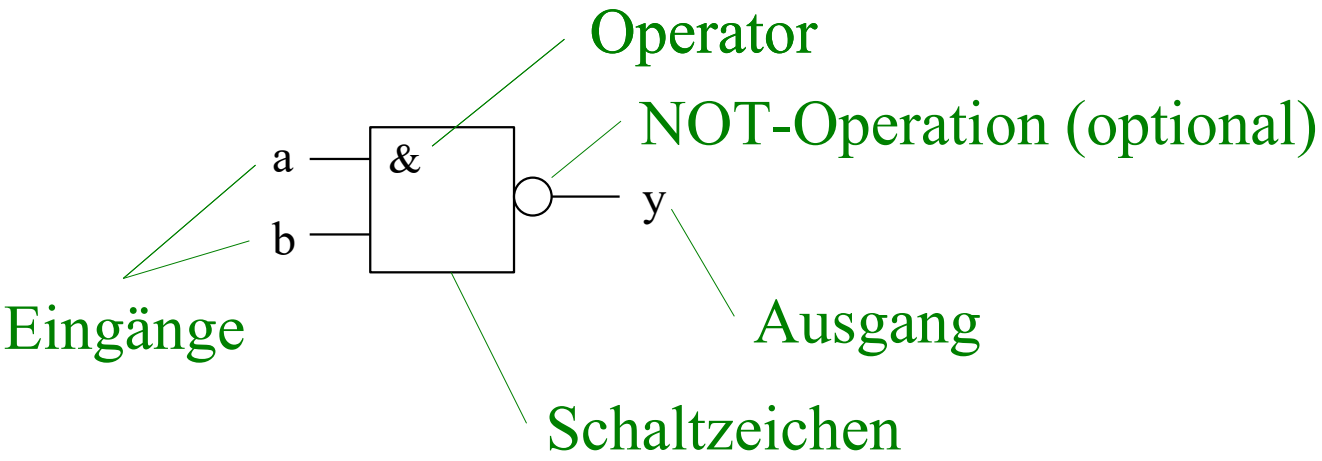
  - In digitaltechnischen Schaltbildern werden abstrakte, implementierungsunabhängige Schaltzeichen verwendet.
  - Hier beispielsweise:





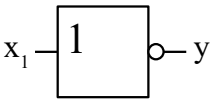
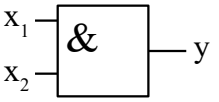
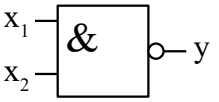
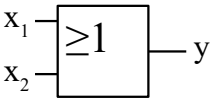
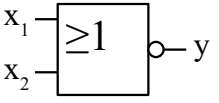
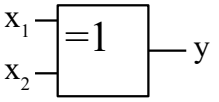
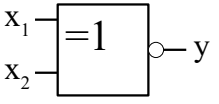
Logiksymbole / Schaltzeichen

- Schaltgatter (Logikgatter) = Realisierung einer schaltalgebraischen Operation
- Abstrakte Darstellung
  - Unabhängig von der Implementierung / Technologie
  - Vereinfachung
    - Keine Spannungs/Energieversorgung
    - Keine Implementierungsdetails (Black Box)
  - Konvention
    - Eingänge links, Ausgänge rechts
    - Ein-/Ausgänge können optional mit NOT-Operation versehen werden



Logiksymbole / Schaltzeichen

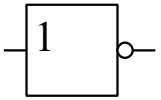
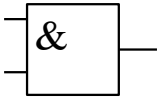
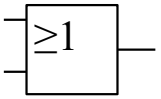
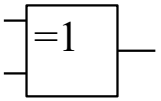
- Schaltzeichen:
  (DIN 60617)

Bezeichnung	*Logikgleichung*	*Wahrheitstabelle* x_1 = 0 0 1 1, x_2 = 0 1 0 1	*Schaltzeichen*
NOT	$y = \overline{x_1}$	y = 1 1 0 0	
AND	$y = x_1 \cdot x_2$	y = 0 0 0 1	
NAND	$y = \overline{x_1 \cdot x_2}$	y = 1 1 1 0	
OR	$y = x_1 + x_2$	y = 0 1 1 1	
NOR	$y = \overline{x_1 + x_2}$	y = 1 0 0 0	
XOR (Antivalenz) *(exclusive-or)*	$y = x_1 \oplus x_2$	y = 0 1 1 0	
XNOR (Äquivalenz) *(exclusive-not-or)*	$y = \overline{x_1 \oplus x_2}$	y = 1 0 0 1	

Hier nur alle kommutativen und nicht-trivialen Operationen sowie deren Negation

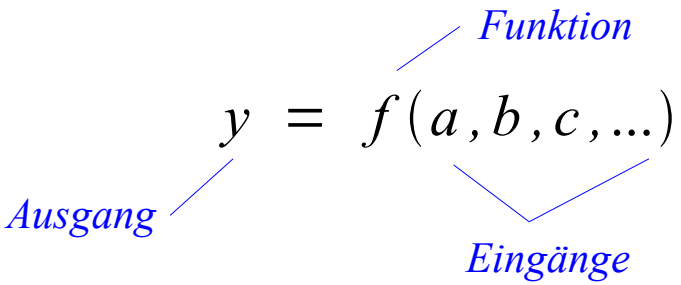
Logiksymbole / Schaltzeichen

- Logiksymbole in verschiedenen Domänen

Operator	*Boolscher Term*		*Schaltzeichen*	*Software*	
	technisch	*mathematisch*	*IEC 60617*	*boolscher Ausdruck*	*bitweise*
NOT	$\overline{x}$	$\neg x$		!	~
AND	$\cdot$	$\wedge$		&&	&
OR	$+$	$\vee$		||	|
XOR	$\oplus$	$\veebar$			^

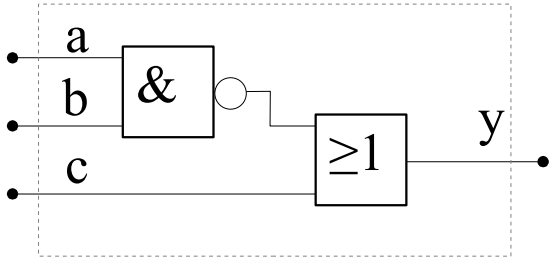
Schaltnetze

- Logische Funktionen werden durch *Schaltnetze* (***kombinatorische Logik***) implementiert
- *Schaltnetze* sind Funktionsblöcke, die aus kombinatorischen Schaltungen bestehen. Sie verarbeiten Schaltvariablen, deren Werte am Ausgang nur von den Eingangswerten zum "gleichen" Zeitpunkt abhängen. (DIN 44300)



"Schnittstelle"

- Beispiel
  - Funktion $y = \overline{a \cdot b} + c$ *"Implementierung"*
  - Schaltnetz

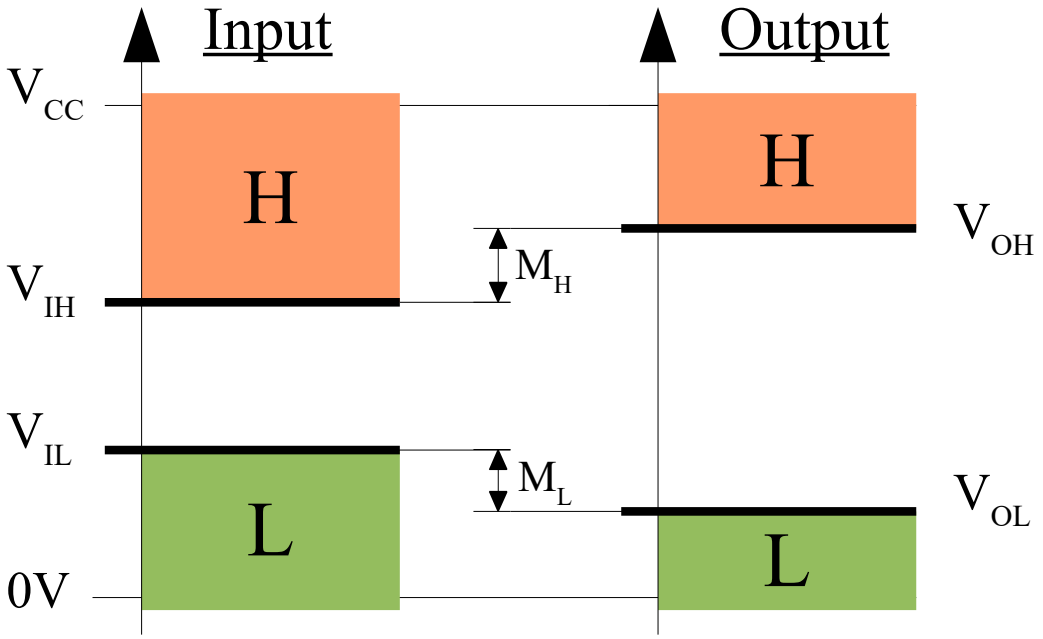


Schaltnetze

- Schaltnetze werden durch elektrische Schaltungen realisiert.
  Dazu gilt folgende Vereinbarung:
  - „positive“ Logik
    - dem logischen 1-Zustand („wahr“) ist der Signalpegel „High“ zugeordnet
    - dem logischen 0-Zustand („falsch“) ist der Signalpegel „Low“ zugeordnet
    - d.h. Spannungen des 1-Zustands sind positiver als die Spannungen des 0-Zustands
  - „negative“ Logik
    - komplementär zur positiven Logik
    - d.h. Spannungen des 1-Zustandes sind negativer als die Spannungen des 0-Zustandes
    - In Datenblättern der Digitaltechnik wird daher meist die Bezeichnung "H" und "L" verwendet.

Signalpegel

- Elektrische Pegel in Digitalschaltungen
  - Tatsächliche Spannungen weichen vom Ideal ab. Ursache:
    - Parasitäre Widerstände und Kapazitäten
    - Bauteiltoleranzen und Temperatureinflüsse
    - Störsignale und Nebensprechen
- Besser: Pegelbereiche mit Störabstand zwischen Eingang und Ausgangspegeln



V_{CC}	Supply Voltage	= 3,3V
V_{IH}	Input High Voltage	= 2,0V
V_{IL}	Input Low Voltage	= 0,8V
V_{OH}	Output High Voltage	= 2,4V
V_{OL}	Output Low Voltage	= 0,4V

(Zahlenwerte 3,3V-CMOS)

Störabstand:

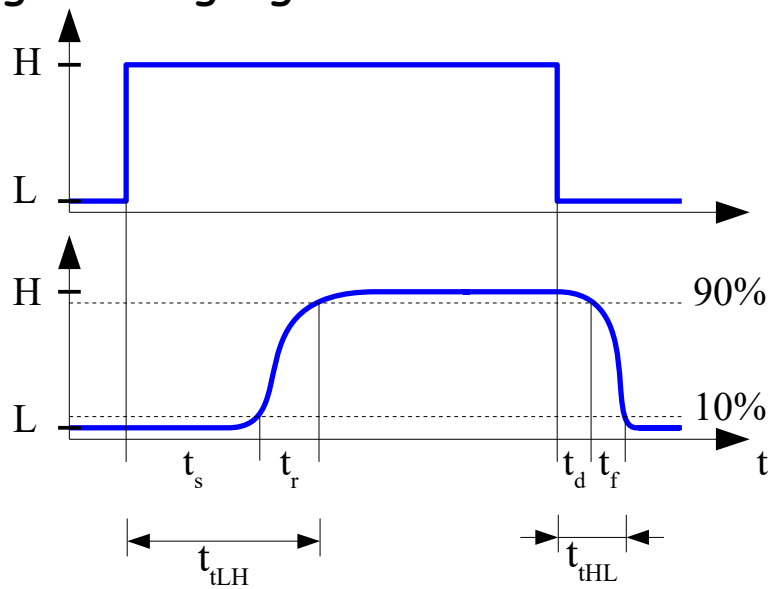
$M_H = V_{OH} - V_{IH} = 0{,}4V$

$M_L = V_{IL} - V_{OL} = 0{,}4V$

Schaltzeiten

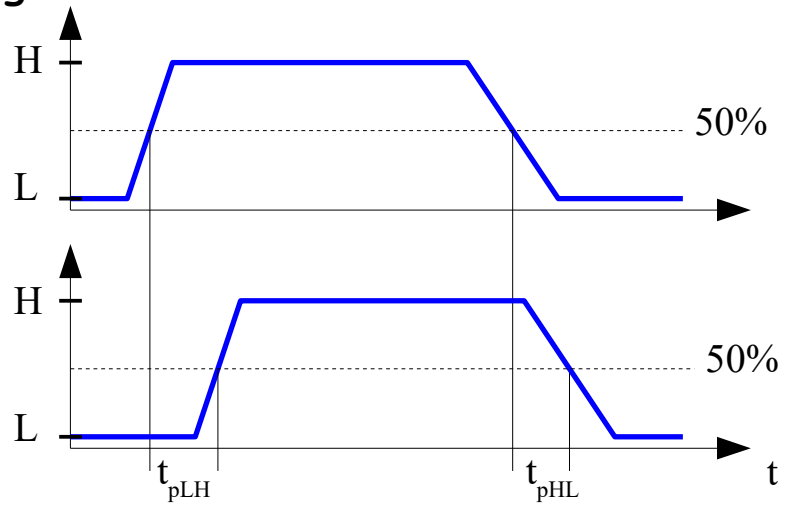
- Bei realen digitalen Bauteile kann das Ausgangssignal dem Eingangssignal nicht in unendlich kurzer Zeit folgen, es müssen *Schaltzeiten* berücksichtigt werden. Die Schaltzeiten bestimmen die maximale Taktfrequenz eines digitalen Systems

Signalübergangszeit:



t_s - storage time
t_r - rise time
t_d - delay time
t_f - fall time
t_{tLH} - transition time low to high
t_{tHL} - transition time high to low

Signallaufzeit:

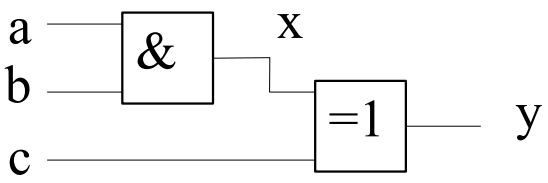


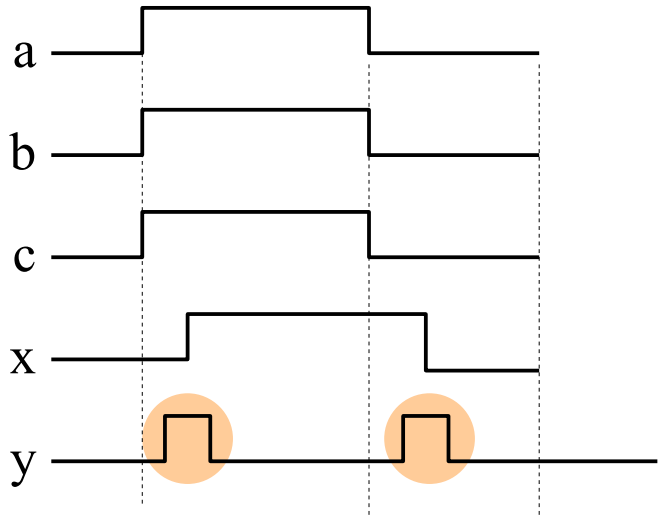
t_{pLH} - low to high
t_{pHL} - high to low
t_p - propagation time $= \frac{t_{pLH} + t_{pHL}}{2}$

Laufzeiteffekte

- Schaltnetze verknüpfen die Eingangssignale kontinuierlich, dadurch:
  - Gewünschte und parasitäre Änderungen der Eingangssignale bzw. interner Signale führen gleichermaßen zu Änderungen der Ausgänge
  - Laufzeiteffekte können während eines Schaltvorgangs Falschwerte (parasitäre Übergangszustände) verursachen
  - *Glitch, Spike*
    - Störimpuls, kurzzeitiger Falschwert
  - *Hazard*
    - Konfiguration, bei der ein Glitch auftreten kann

- Beispiel:





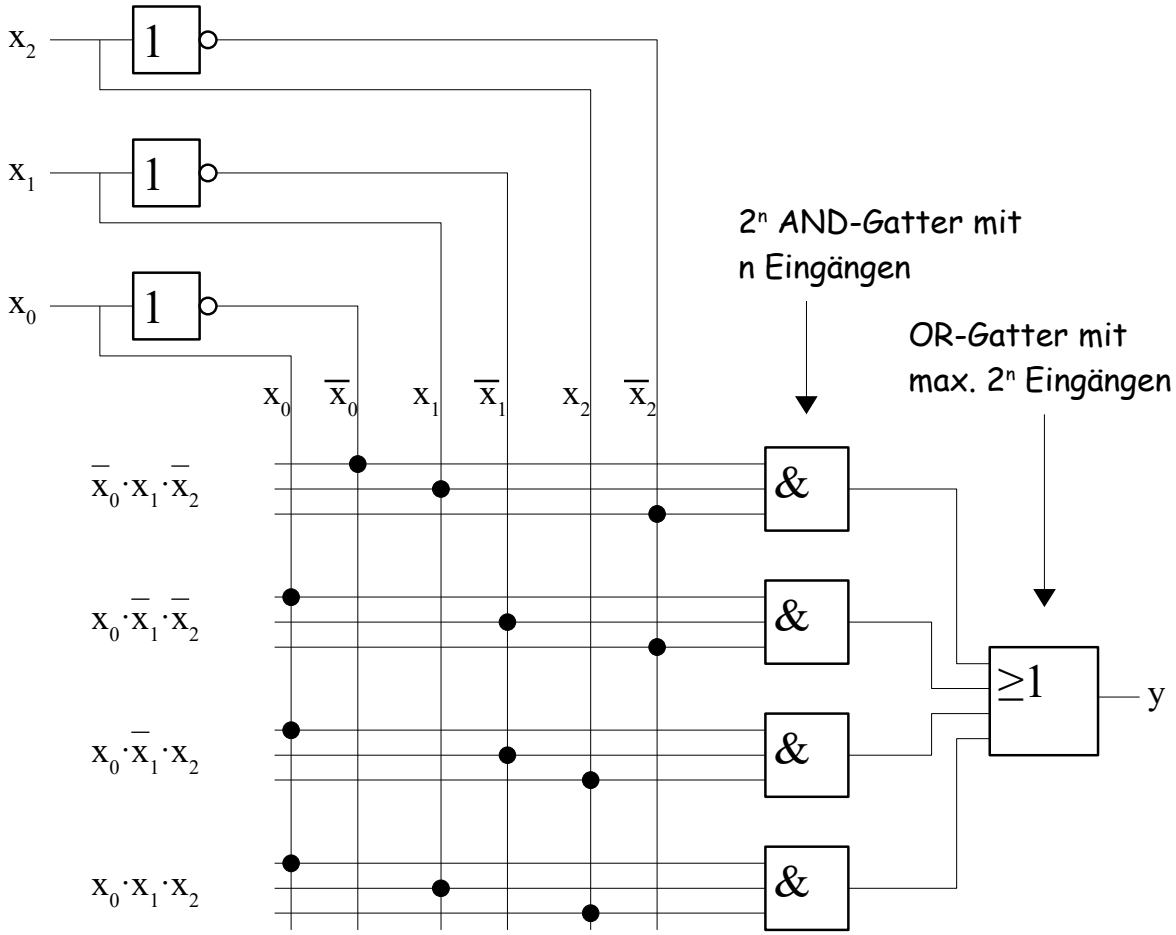
  - alle Eingangssignale ändern sich gleichzeitig, Ausgang y müsste $y = 0$ bleiben
  - Durch die Laufzeit des UND-Gatters wird kurzzeitig $x \neq y$, der Ausgang wird zweimal parasitär zu 1

Schaltnetze

- Realisierung einer (voll-)disjunktiven Schaltfunktionen durch zweistufige Logik
- Beispiel:

$$y = (\overline{x_0} \cdot x_1 \cdot \overline{x_2}) + (x_0 \cdot \overline{x_1} \cdot \overline{x_2}) + (x_0 \cdot \overline{x_1} \cdot x_2) + (x_0 \cdot x_1 \cdot x_2)$$

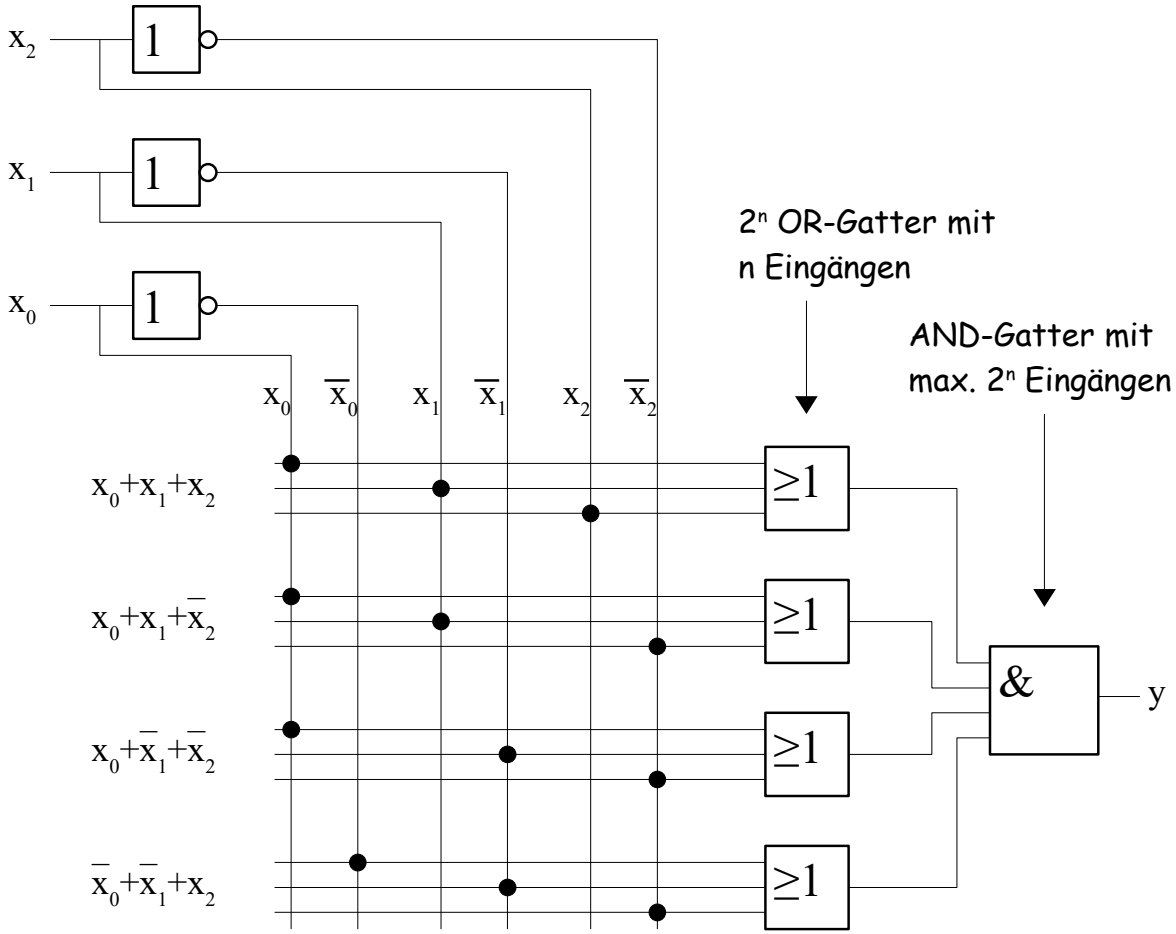
x_0	x_1	x_2	y	*Minterme*
0	0	0	0	
0	0	1	0	
0	1	0	1	$\overline{x_0} \cdot x_1 \cdot \overline{x_2}$
0	1	1	0	
1	0	0	1	$x_0 \cdot \overline{x_1} \cdot \overline{x_2}$
1	0	1	1	$x_0 \cdot \overline{x_1} \cdot x_2$
1	1	0	0	
1	1	1	1	$x_0 \cdot x_1 \cdot x_2$



- Realisierung einer (voll-)konjunktiven Schaltfunktionen durch zweistufige Logik
- Beispiel:

$$y = (x_0 + x_1 + x_2) \cdot (x_0 + x_1 + \overline{x_2}) \cdot (x_0 + \overline{x_1} + \overline{x_2}) \cdot (\overline{x_0} + \overline{x_1} + x_2)$$

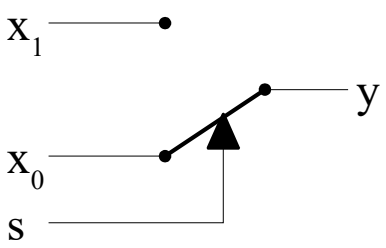
x_0	x_1	x_2	y	*Maxterme*
0	0	0	0	$x_0 + x_1 + x_2$
0	0	1	0	$x_0 + x_1 + \overline{x_2}$
0	1	0	1	
0	1	1	0	$x_0 + \overline{x_1} + \overline{x_2}$
1	0	0	1	
1	0	1	1	
1	1	0	0	$\overline{x_0} + \overline{x_1} + x_2$
1	1	1	1	



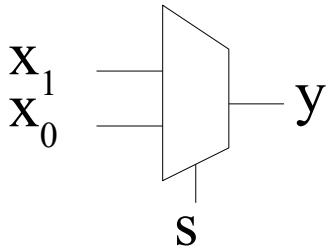
Beispiel: Multiplexer

- Multiplexer:
  - „Weiche“ für logische Signale. Das Steuersignal s entscheidet, welches der Eingangssignale x_i an den Ausgang geschaltet y wird

Schaltungsprinzip:



Schaltzeichen:

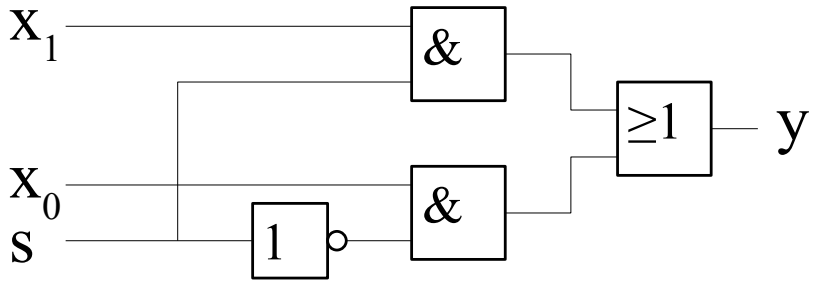


Wahrheitstabelle:

Eingang			*Ausgang*
x_1	x_0	s	y
*	0	0	0
*	1	0	1
0	*	1	0
1	*	1	1

*: don't care

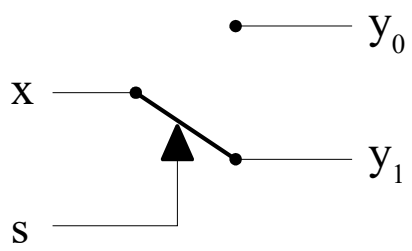
Schaltungsrealisierung:



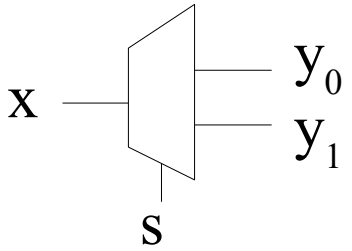
Beispiel: Demultiplexer

- Demultiplexer:
  - Das Steuersignal s entscheidet, an welchem Ausgang das Eingangssignal weitergegeben wird.

Schaltungsprinzip:



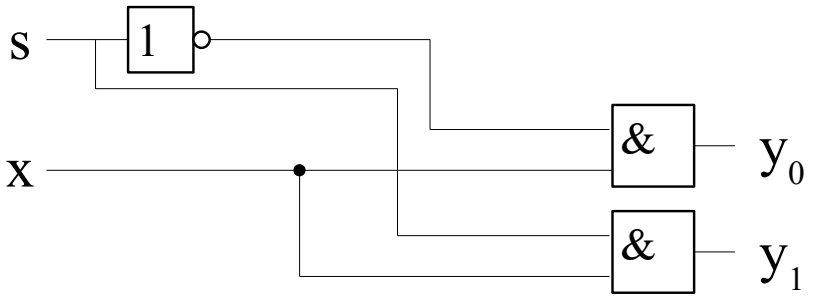
Schaltzeichen:



Wahrheitstabelle:

Eingang		*Ausgang*	
s	***x***	***y_1***	***y_0***
0	0	0	**0**
0	1	0	**1**
1	0	**0**	0
1	1	**1**	0

Schaltungsrealisierung:



Beispiel: Prioritäts-Encoder

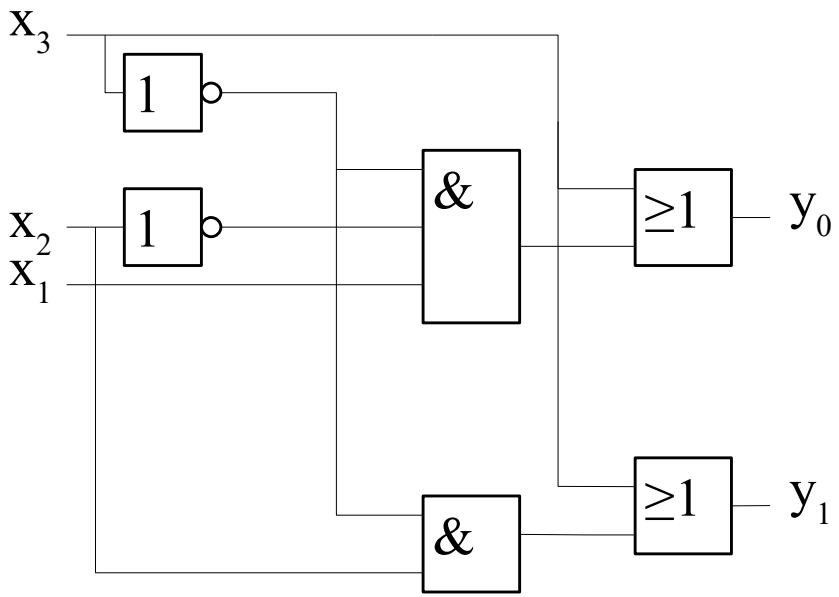
- M-to-N Prioritäts-Encoder:
  - Prüft, welcher höchstwertige Eingang den Wert „1“ hat.
  - Der Ausgang codiert die Nummer dieses Eingangs.
    - M: Anzahl Eingänge, N: Anzahl Ausgänge ($2^N \geq M$)

Wahrheitstabelle:

Eingang			*Ausgang*	
x_3	x_2	x_1	y_1	y_0
0	0	0	0	0
0	0	1	0	1
0	1	*	1	0
1	*	*	1	1

*: don't care

Schaltungsrealisierung:



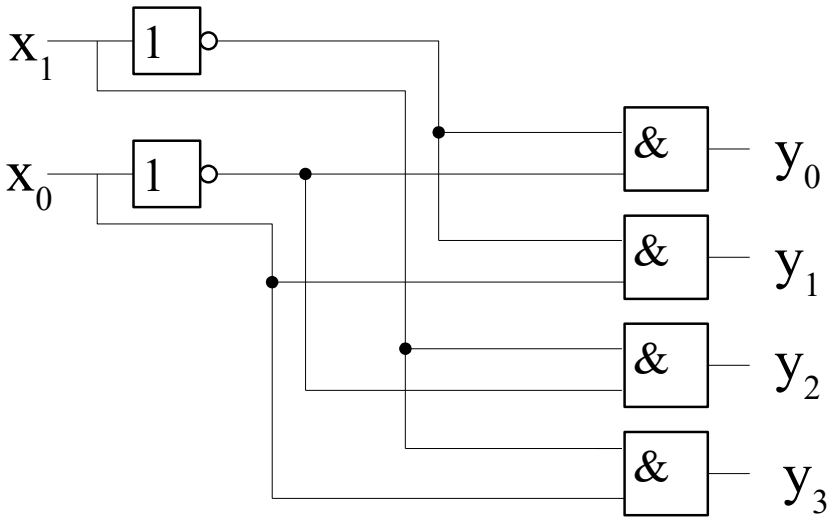
Beispiel: Line Decoder

- M-to-N Line Decoder:
  - Mit den M Eingangsvariablen x_i wird genau einer der N Ausgangsvariablen y_i ausgewählt

Wahrheitstabelle:

Eingang		*Ausgang*			
x_1	x_0	y_3	y_2	y_1	y_0
0	0	0	0	0	**1**
0	1	0	0	**1**	0
1	0	0	**1**	0	0
1	1	**1**	0	0	0

Schaltungsrealisierung:



Programmierbare Logik

- Flexible Realisierung von Schaltnetzen durch „programmierbare Logik“:
  - Grundidee: jede Funktion kann durch eine disjunktive Normalform beschrieben werden
  - Erste programmierbare Logik: **PAL** (= Programmable Array Logic)
    - Mit programmierbarer UND und fester ODER „Matrix“. Nur 1x beschreibbar, denn die Verknüpfungen auf dem Chip werden durch nicht reversibles „Durchbrennen" von Verbindungen beim Programmieren hergestellt.
    - Realisiert die disjunktive Normalform
  - Nachfolger: ***GAL*** (= Generic Array Logic)
    - GALs auch mit programmierbarem UND- und festem ODER Array
    - Können „gelöscht“ und dann neu beschrieben werden
  - Mittlerweile werden PALs und GALs nur noch selten eingesetzt