Digitaltechnik

2 Technologie

Revision 1.05

Abstrakte Schalter

Schalter in Hardware

Integrierte Schaltkreise

Physikalische Aspekte

Latches, Flipflops und Clocks

Field-Programmable Gate Arrays (FPGAs)

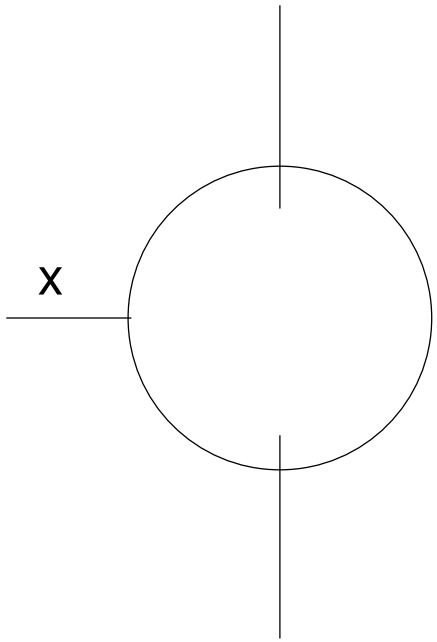
Logikgatter – Logic Gates

- Grundbaustein von digitalen Schaltungen
- Implementieren Boole'sche Funktionen
- Gatter hat mehrere Eingänge, meist einen Ausgang

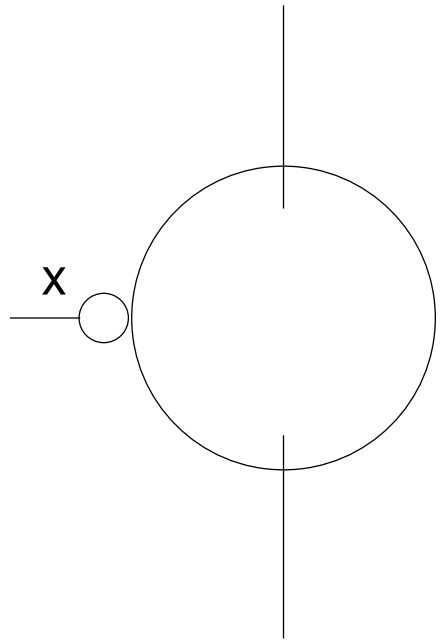
- Logische Belegung pro Verbindung (Eingang/Ausgang):
  0 oder **1**

- Physikalische Repräsentation:
  0 $\approx$ Spannung 0V
  1 $\approx$ Spannung 3.3V

Abstrakte Schalter



Schaltet für x = 1



Schaltet für x = 0

Idee: Schalter geschlossen $\Rightarrow$ oberer und unterer Anschluss verbunden

Inverter und NOR als Beispiel

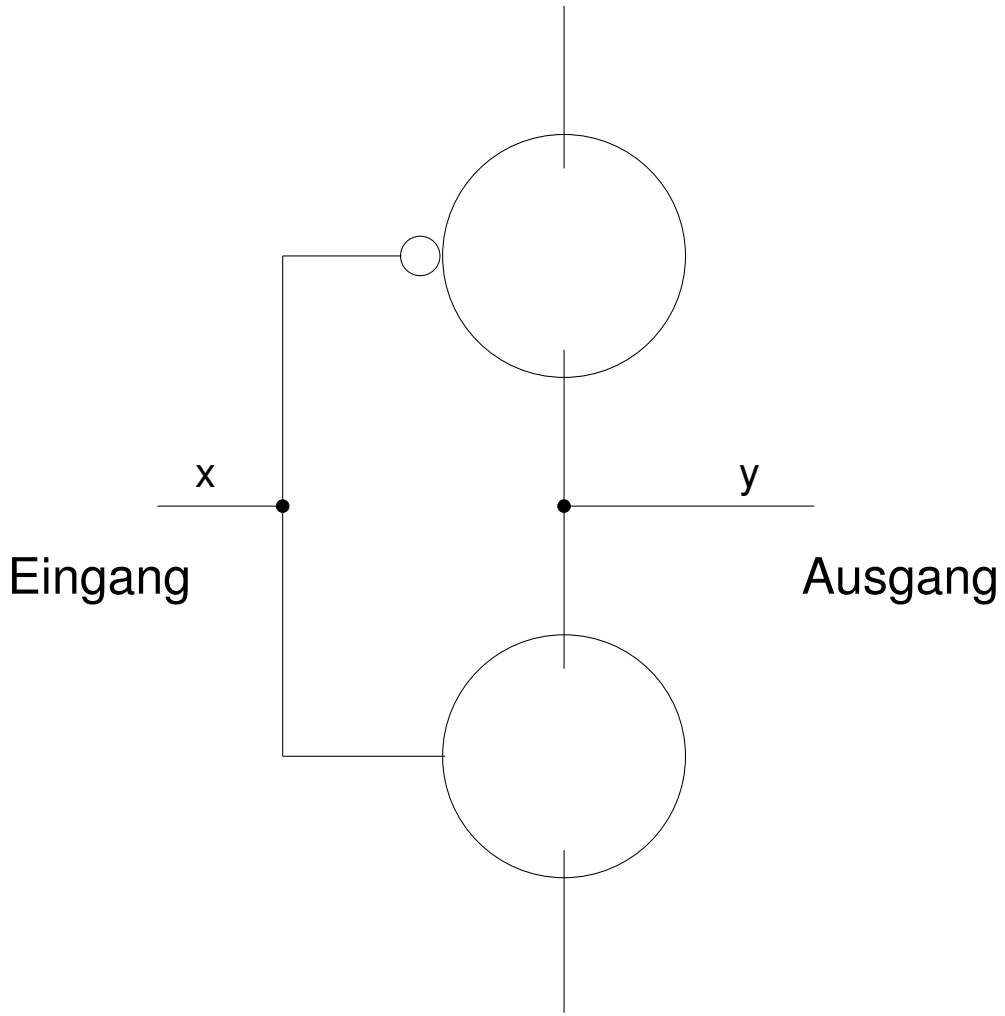
Inverter logische Negation
invertiert den Eingangswert

x	$\overline{x}$
0	1
1	0

NOR Disjunktion mit anschließender Negation
Ausgang 1 gdw. beide Eingänge 0

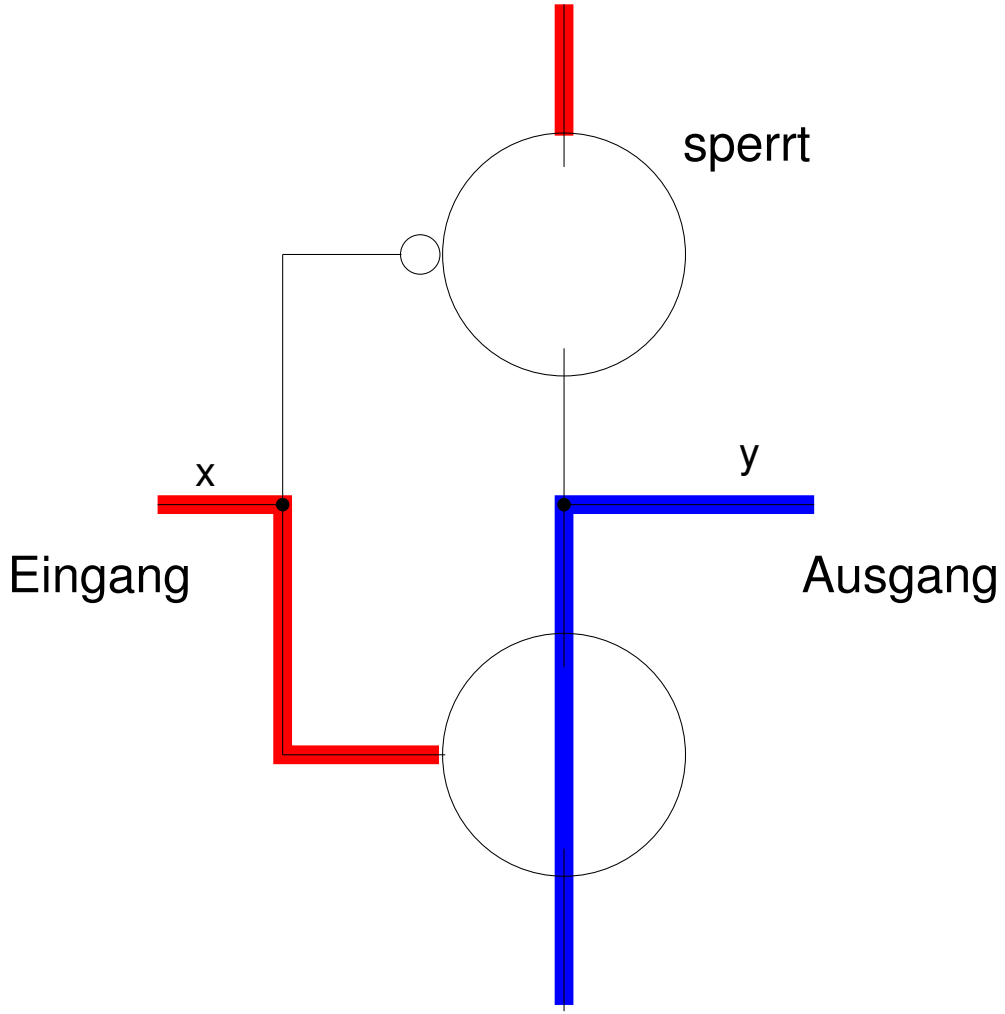
x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

Inverter mit abstrakten Schaltern



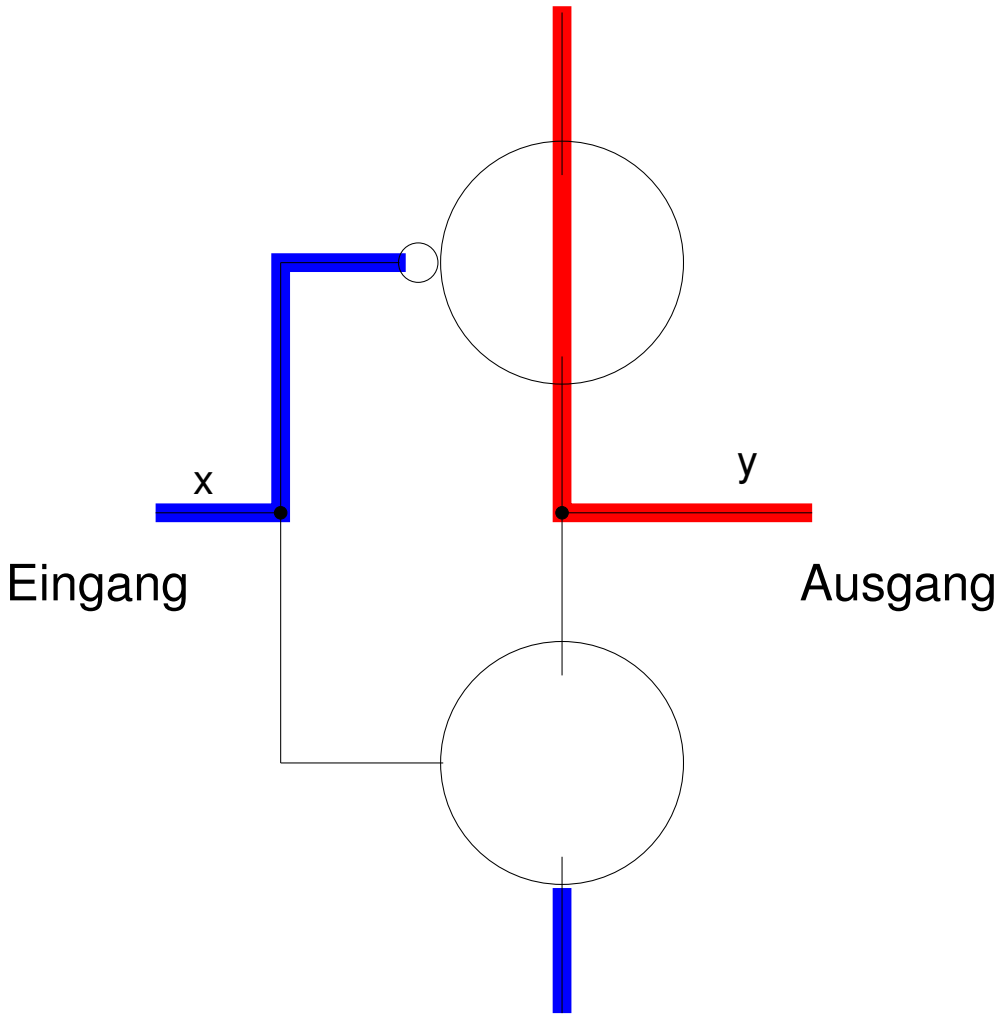
Wie ein kaputter Mischer in der Dusche für warmes und kaltes Wasser!
(entweder kaltes Wasser von unten, oder warmes von oben)

Inverter mit komplementären Schaltern



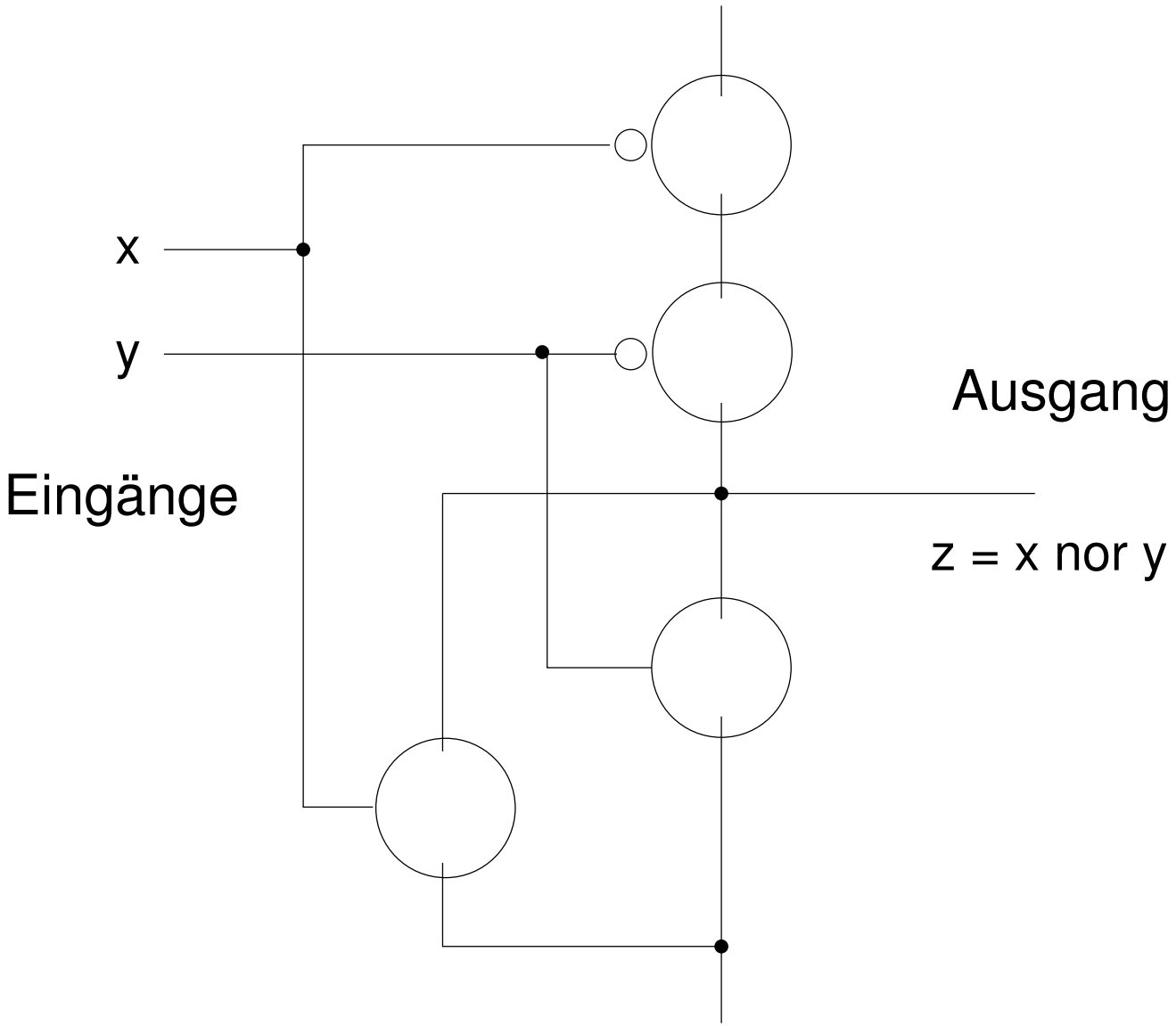
Warmer Eingang schaltet nur den unteren Schalter!

Inverter mit komplementären Schaltern

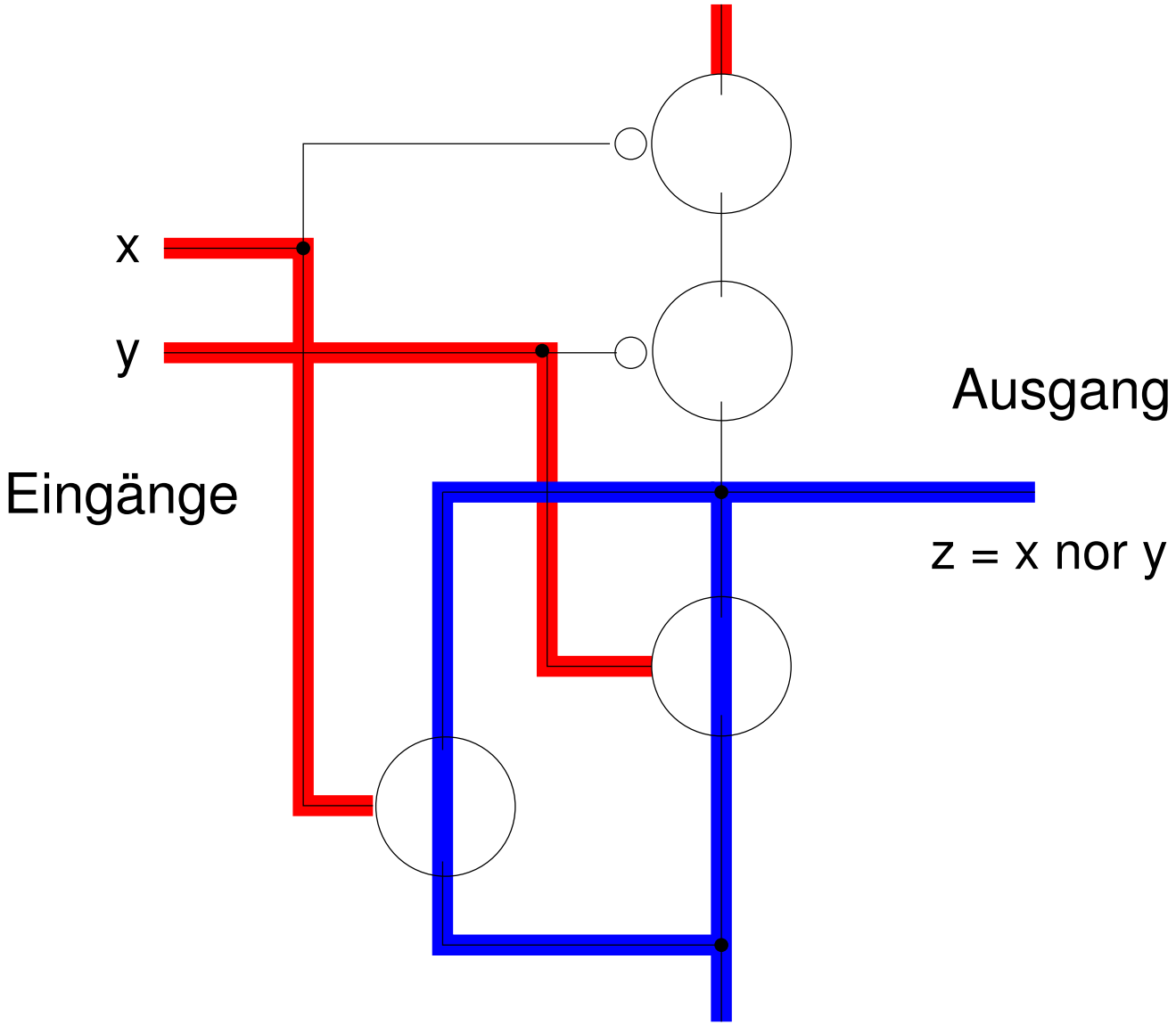


Kalter Eingang schaltet nur den oberen Schalter!

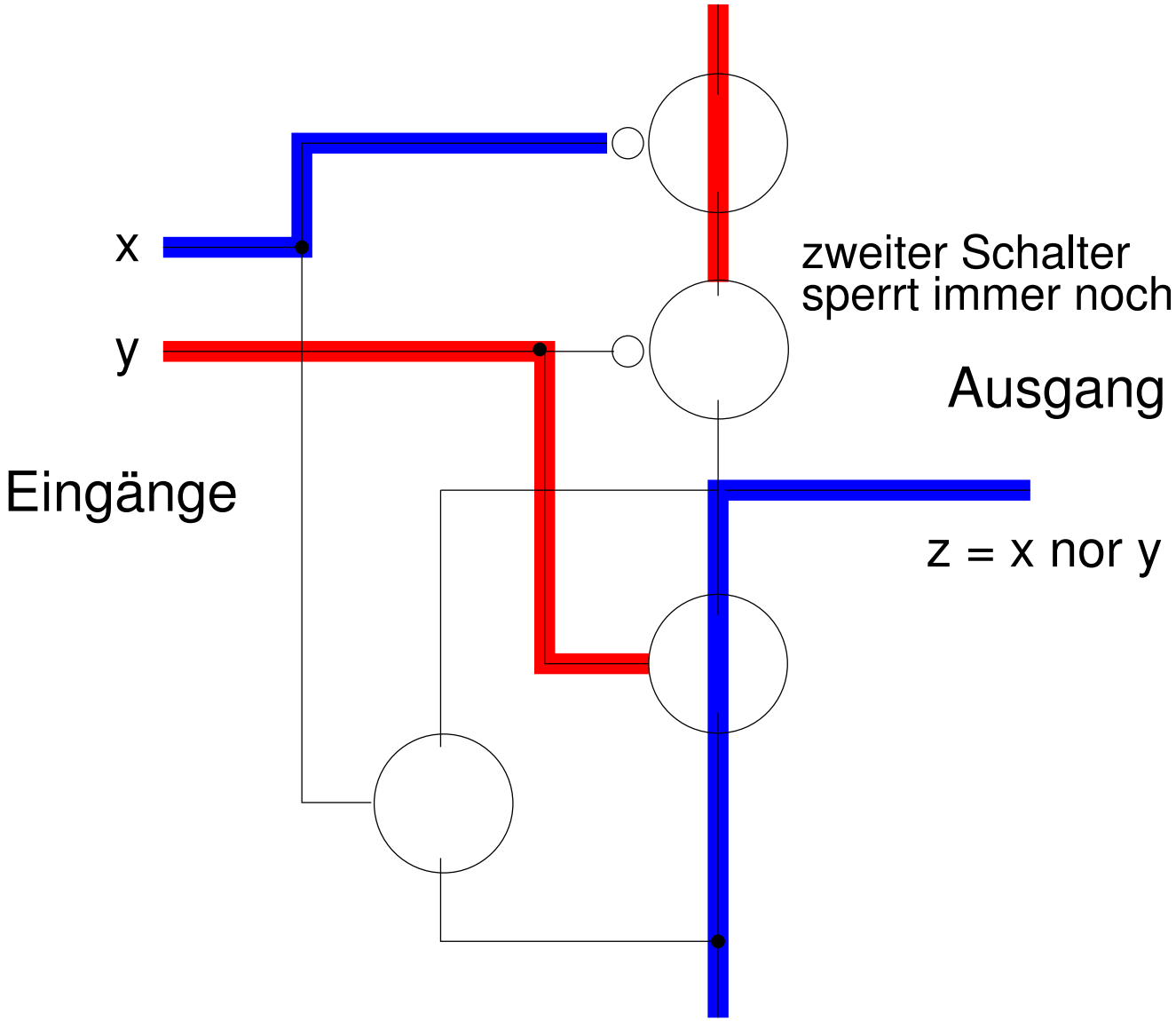
NOR mit abstrakten Schaltern



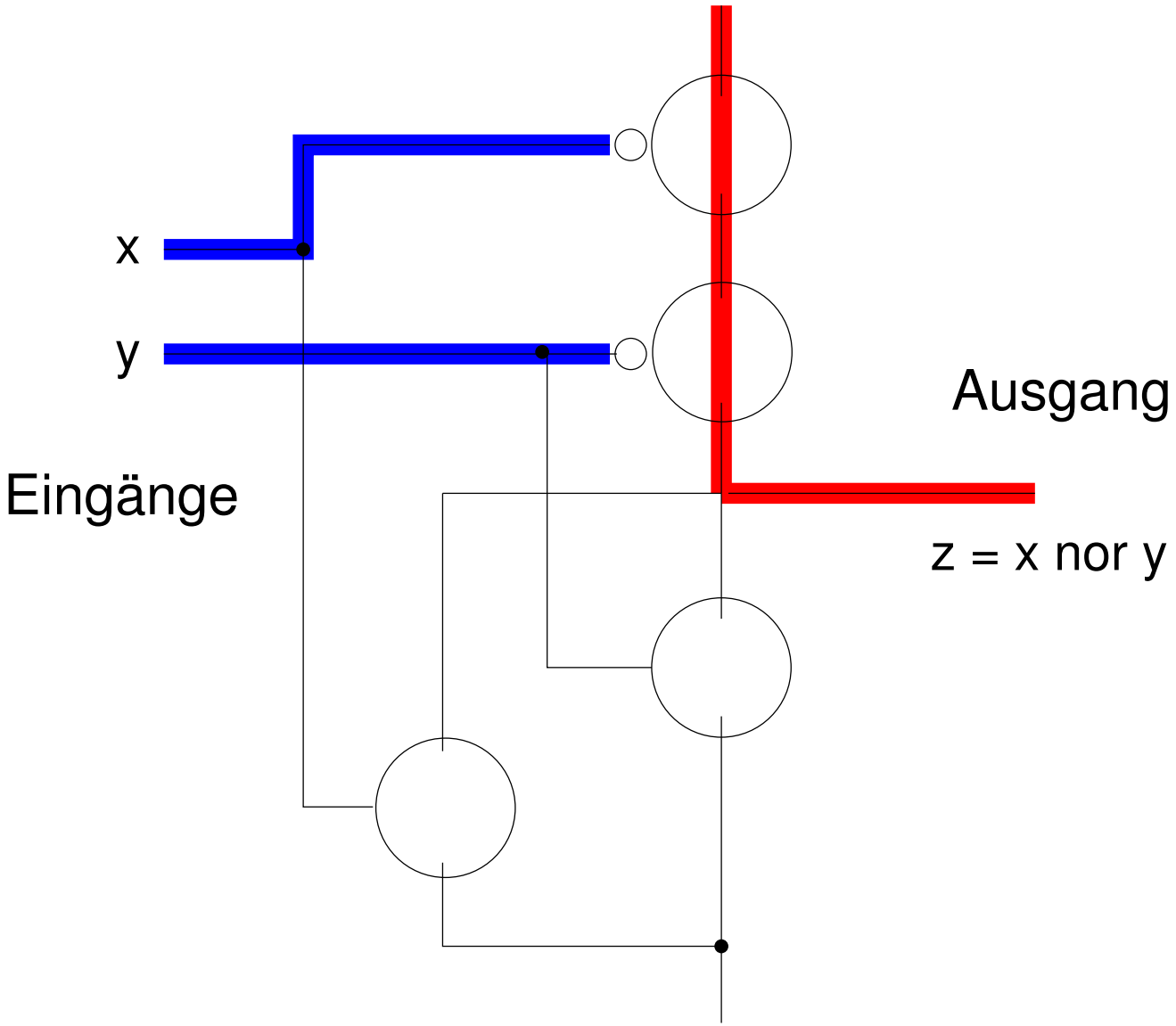
NOR mit abstrakten Schaltern



NOR mit abstrakten Schaltern



NOR mit abstrakten Schaltern

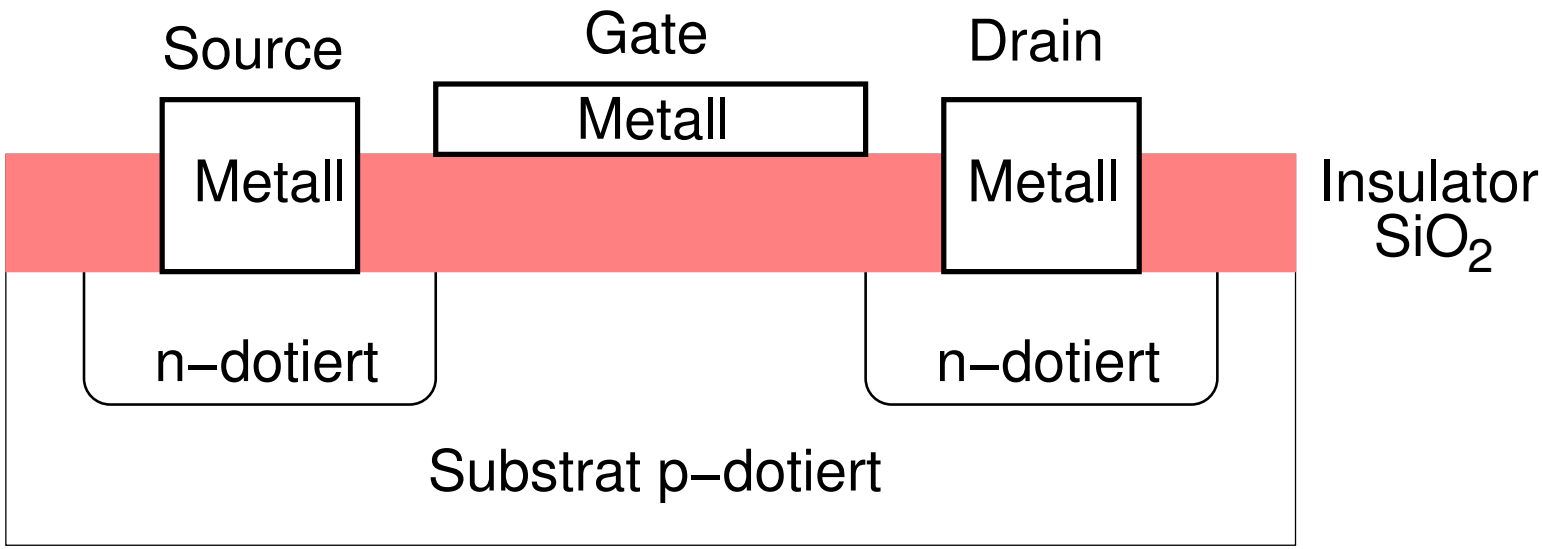


Geläufige Schalter-Implementierungen

Technologie: Physikalische Implementierung eines Schalters

Beispiele:

- Vakuum-Röhre
- BJT: Bipolar Junction Transistor, verwendet in TTL
- TTL: Transistor-Transistor Logik
- FET: Feld Effekt Transistor, verwendet in MOS
- MOS von MOSFET = Metal Oxide Semiconducter
- CMOS: Complementary MOS (unser Hauptinteresse)

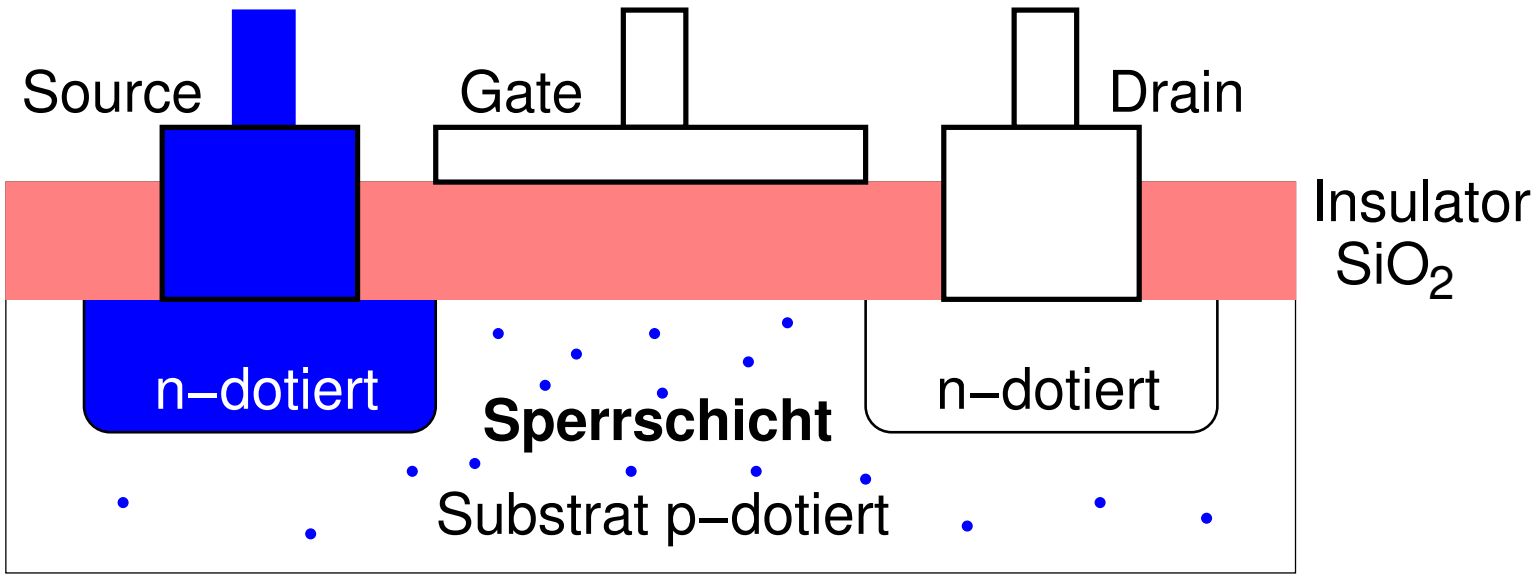


Implementierung eines Schalters mit dotiertem Silizium

n/p-Dotierung: Überschuss **n**egativer/**p**ositiver Ladungsträger

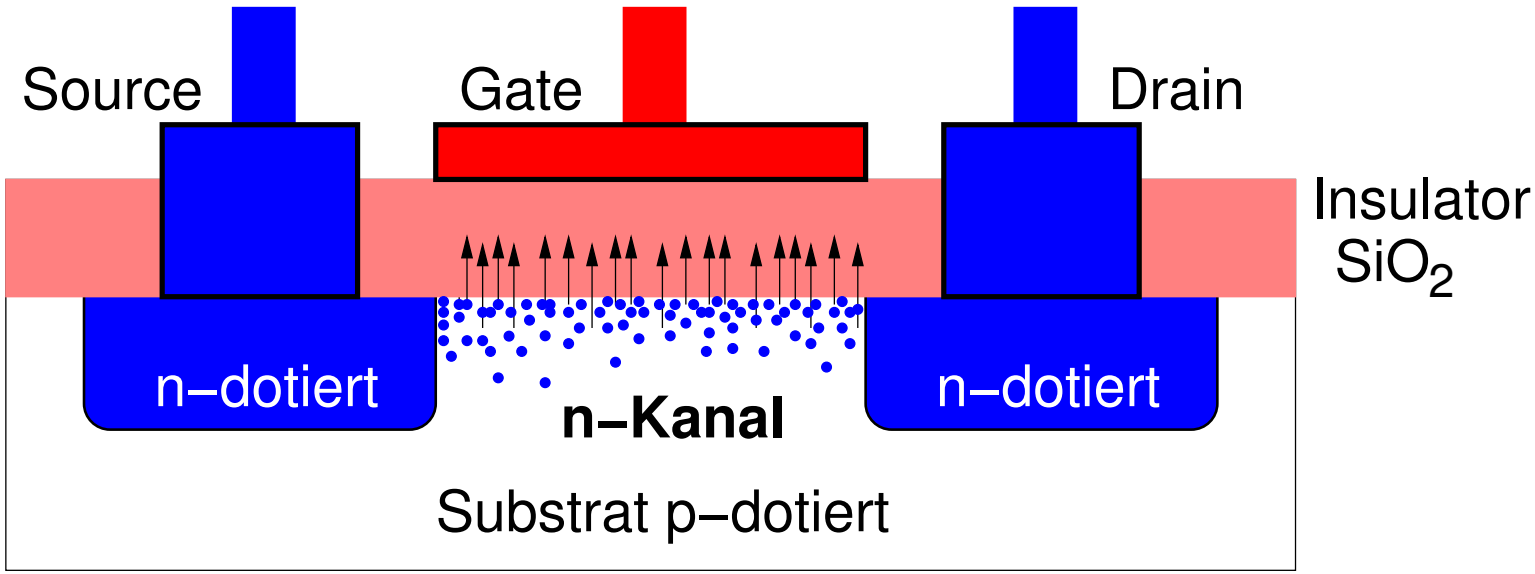
Dual dazu PMOS: Spannungen und logische Zustände jeweils umgekehrt!

NMOS Offen



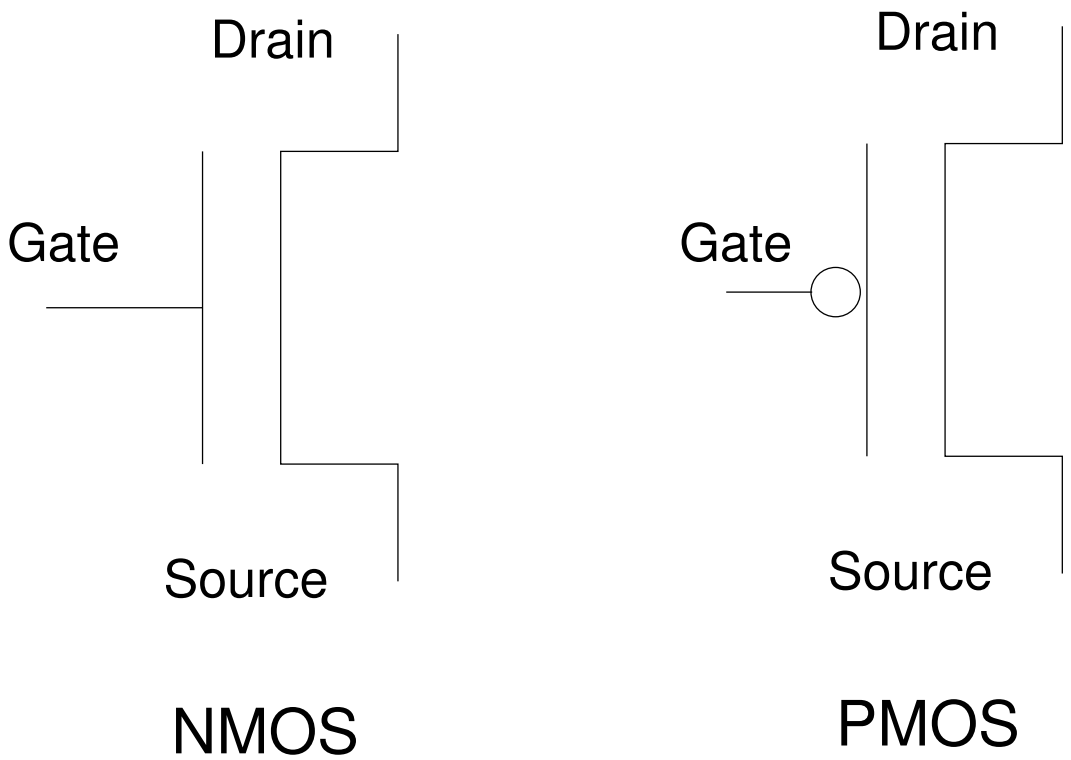
Spannung Gate/Source $\leq$ 0 V: Schalter offen (kein Strom)

NMOS Geschlossen

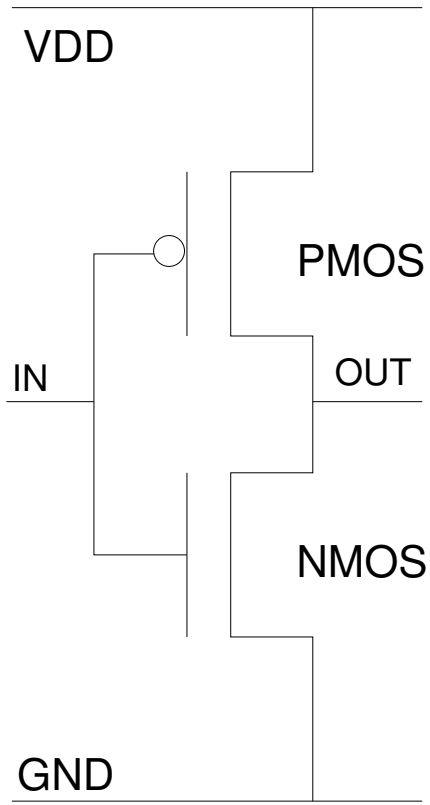


Spannung Gate/Source $>$ 0 V: Schalter geschlossen
(pos. Drain/Source Strom)

Schaltbilder



CMOS Inverter



IN auf 0:

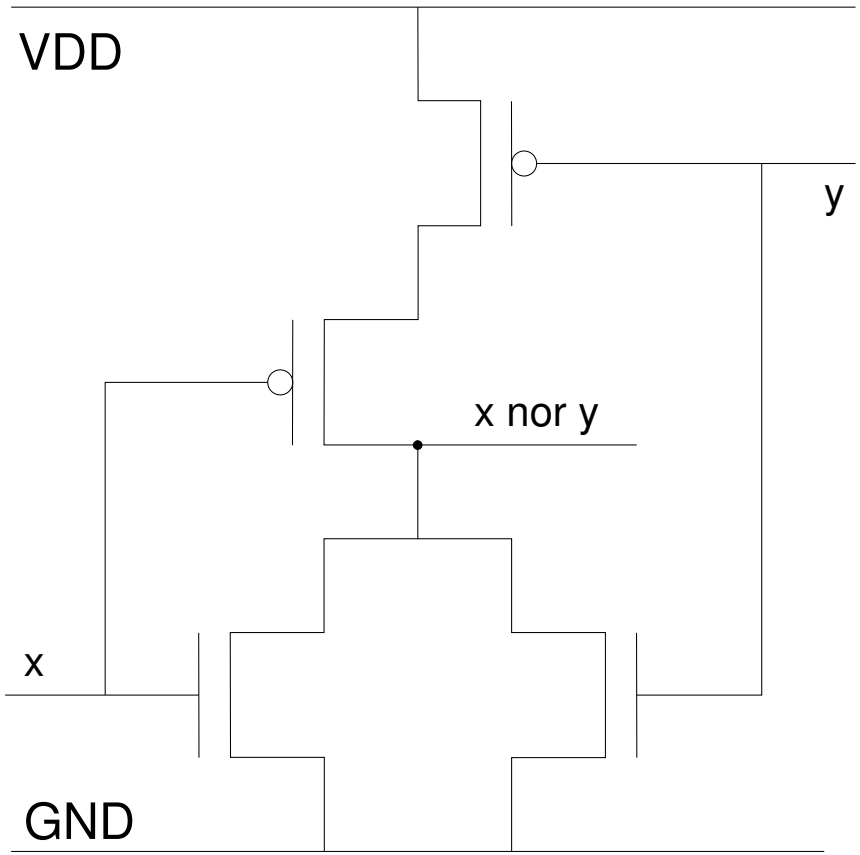
PMOS schaltet, NMOS sperrt
$\Rightarrow$ OUT auf 1

IN auf 1:

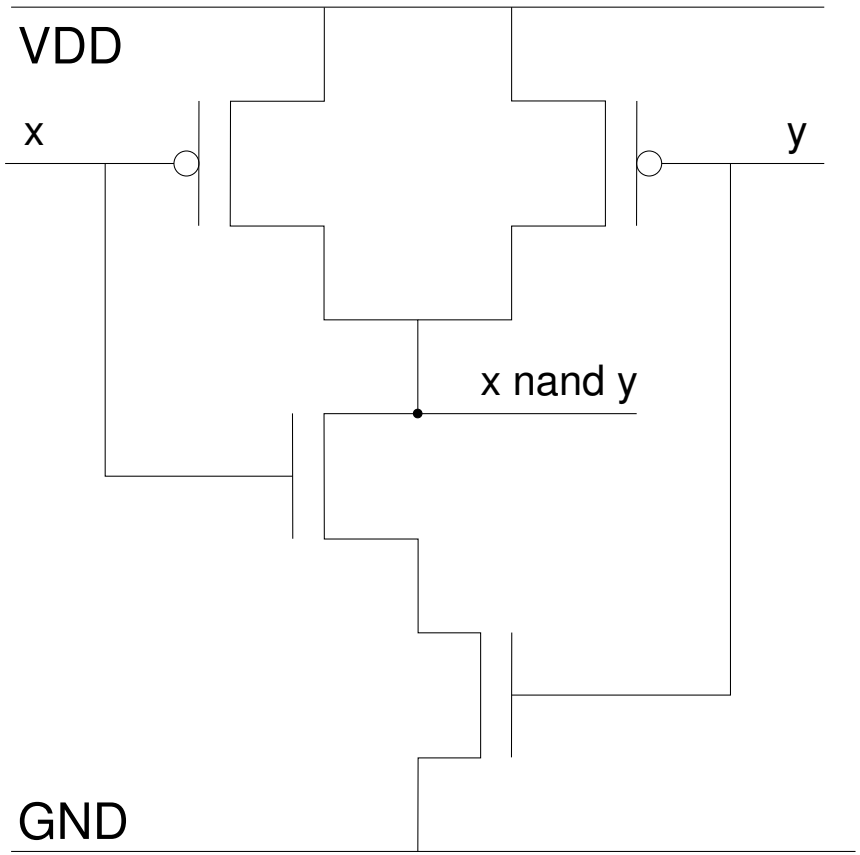
PMOS sperrt, NMOS schaltet
$\Rightarrow$ OUT auf 0

Tipp: `http:`
`//tams-www.informatik.uni-hamburg.de/applets/cmos/`
Java Animationen der meisten Gatter

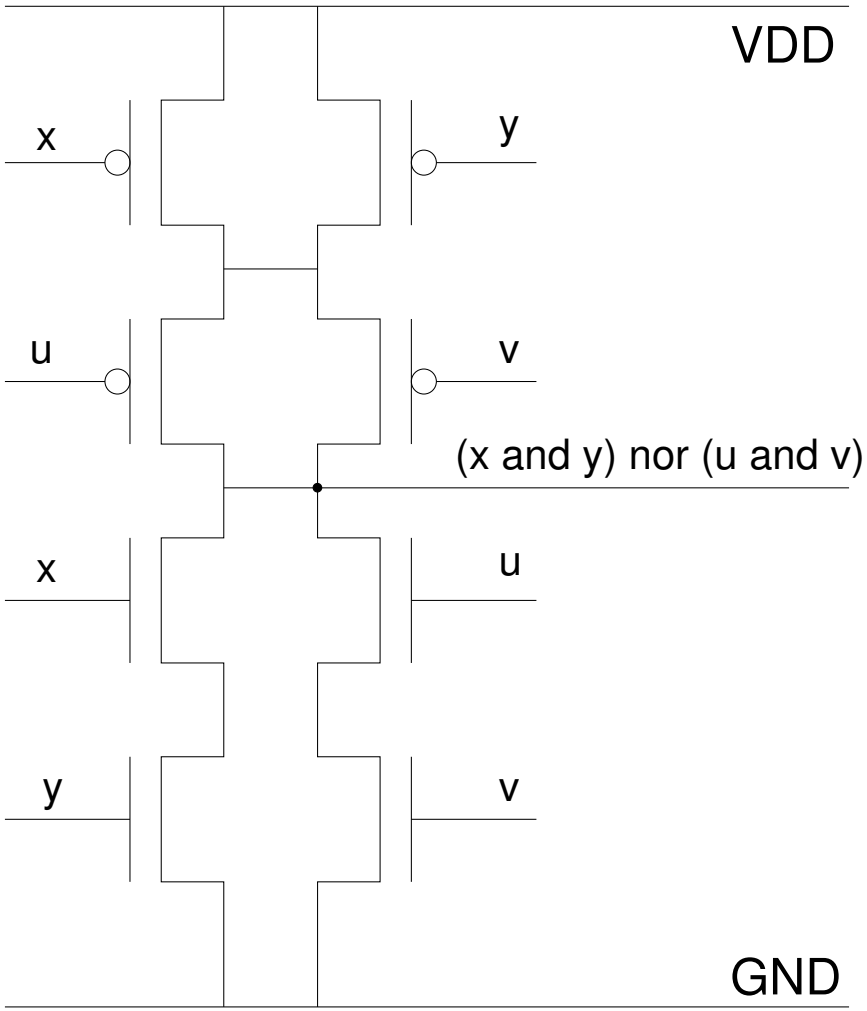
CMOS NOR



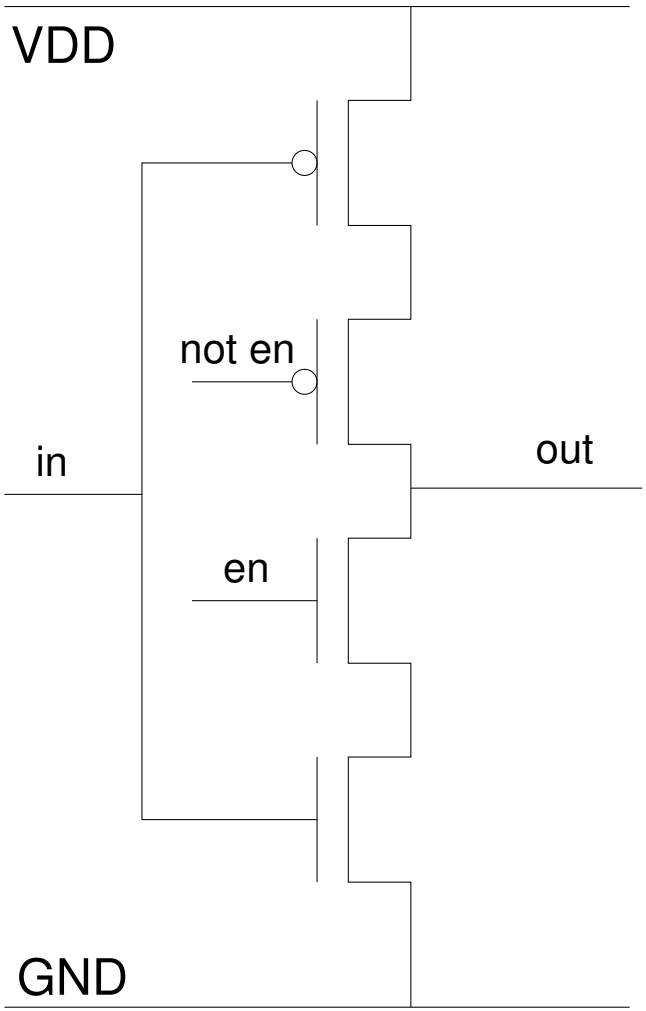
CMOS NAND



CMOS AND-NOR



Threestate-Buffer



in wird negiert

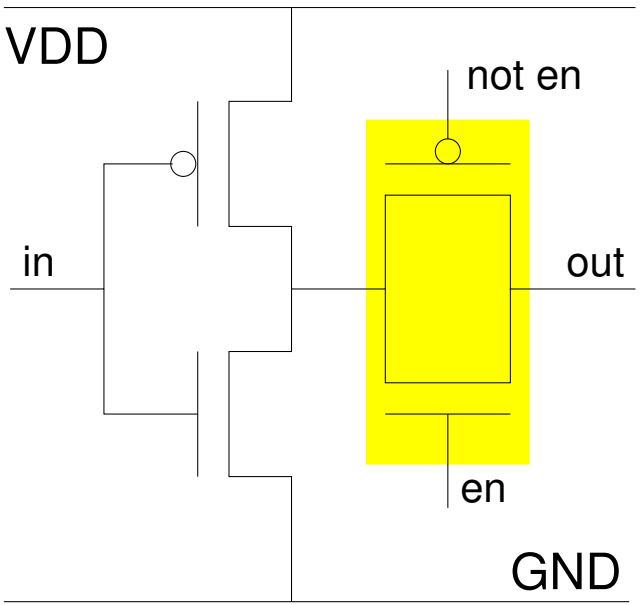
durchgeleitet zu out gdw.

en auf 1 liegt, sonst ist

out hochohmig

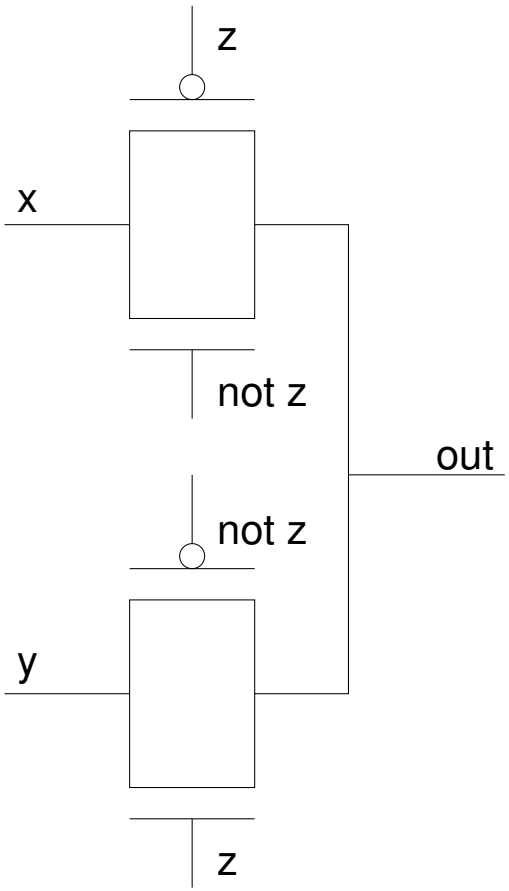
(unverbunden)

Alternative Implementierung eines Threestate-Buffers mit Transmission-Gate



Drei logische Ausgangs-Zustände: 0, 1, Z (hochohmig)

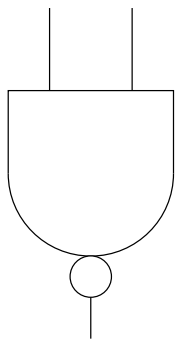
Multiplexer



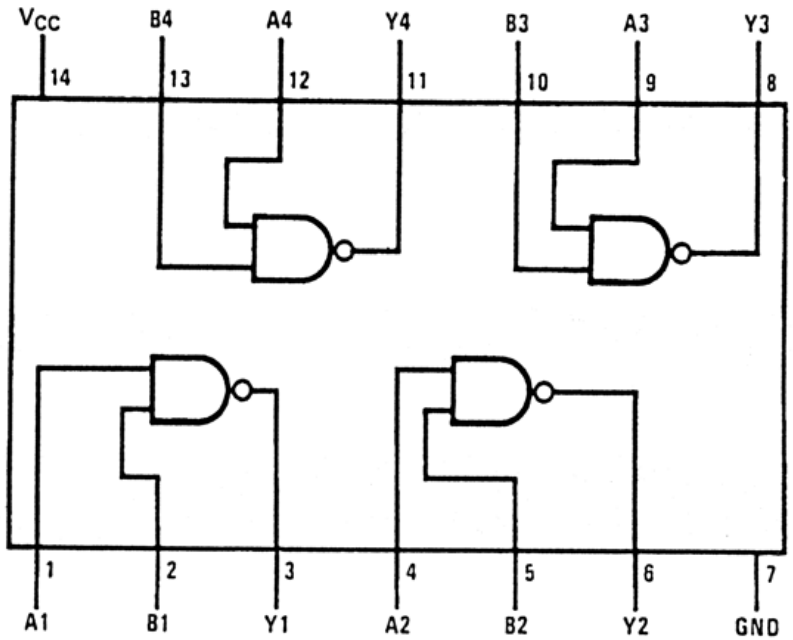
- out = if z then y else x
- ✔ braucht weniger Transistoren (Übung!)
- Switch-level Modellierung in Verilog

NAND Gatter

Zwei Eingänge



Ein Ausgang



Integrierte Schaltkreise – IC = Integrated Circuits

Leiterplatten-Entwurf – PCB = Printed Circuit Boards

- Zusammenlöten oder Verdrahten von kleinen ICs

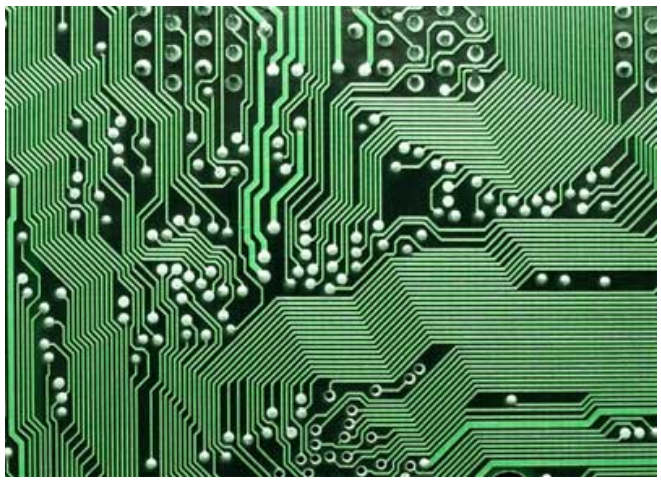
✔ Fixkosten: Prototyp für Platinen (Leiterplatten) recht billig

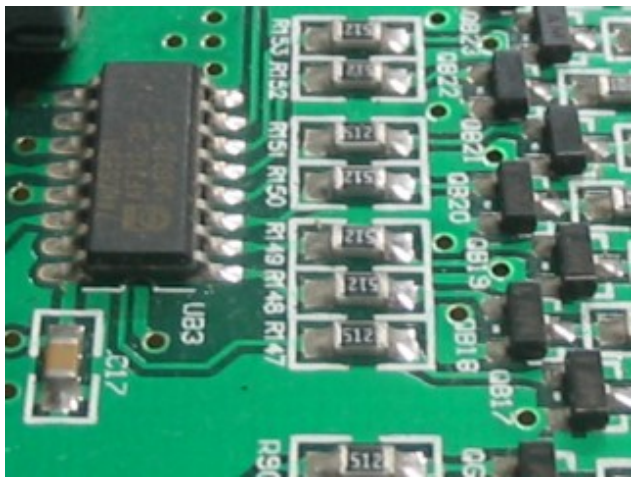
✘ Aufwendige Produktion

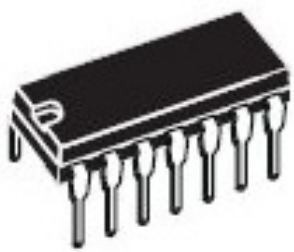
✘ Langsam und nur mäßig flexibel

✘ Teuer in großen Stückzahlen

Leiterplatten-Entwurf – PCB = Printed Circuit Boards







DIP

SOP

TSSOP

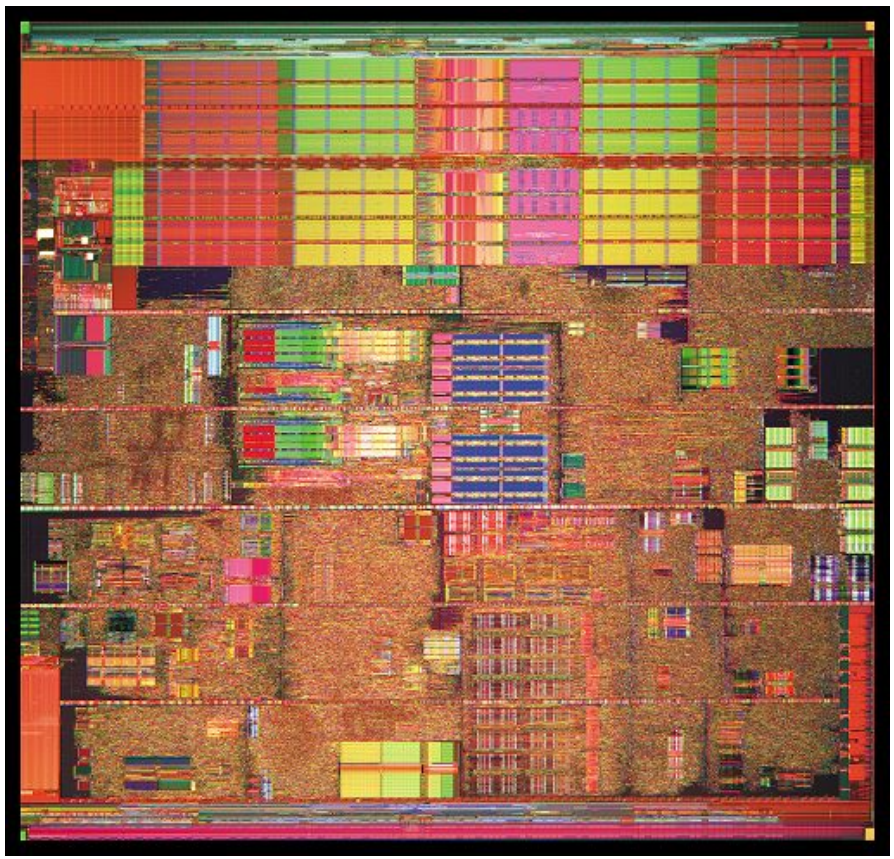
VLSI

- Very Large Scale Integration (sehr hohe Integrationsdichte), auch LSI

✔ Stückpreis gering

✘ Masken sehr teuer (bis zu 1 Million Dollar)

- Lithographische Produktion von sehr vielen Gattern auf einem Chip
- Gemessen in Transistoren (Ungefähr 4-10 Transistoren pro Gatter)
- Millionen Gatter auf einem Chip (halbe Milliarde Transistoren)
- Strukturgröße: kleinste Struktur Chip (100 nm = 0.10 μm)
- Erlaubt “System on a Chip”



Intel Prescott, 90nm Prozess, 125 Mio. Transistoren auf 122mm^2

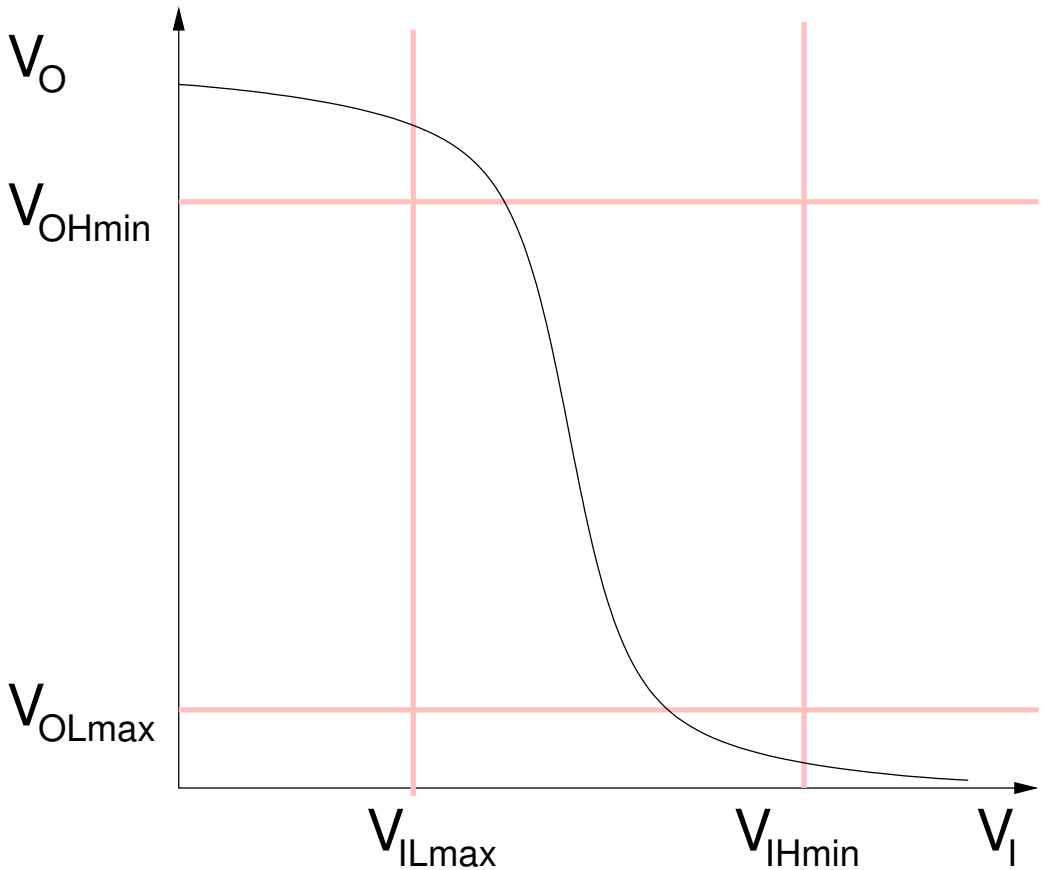
Einzelpreis und Fixkosten

- Je kleiner der Einzelpreis desto höher die Fixkosten (z.B. Masken)
- ASICs (Application-Specific IC) für hohe Stückzahlen
- ASICs werden zumeist vom Chip-Hersteller speziell gefertigt
- Programmierbare Schaltkreise können vom Designer angepasst werden

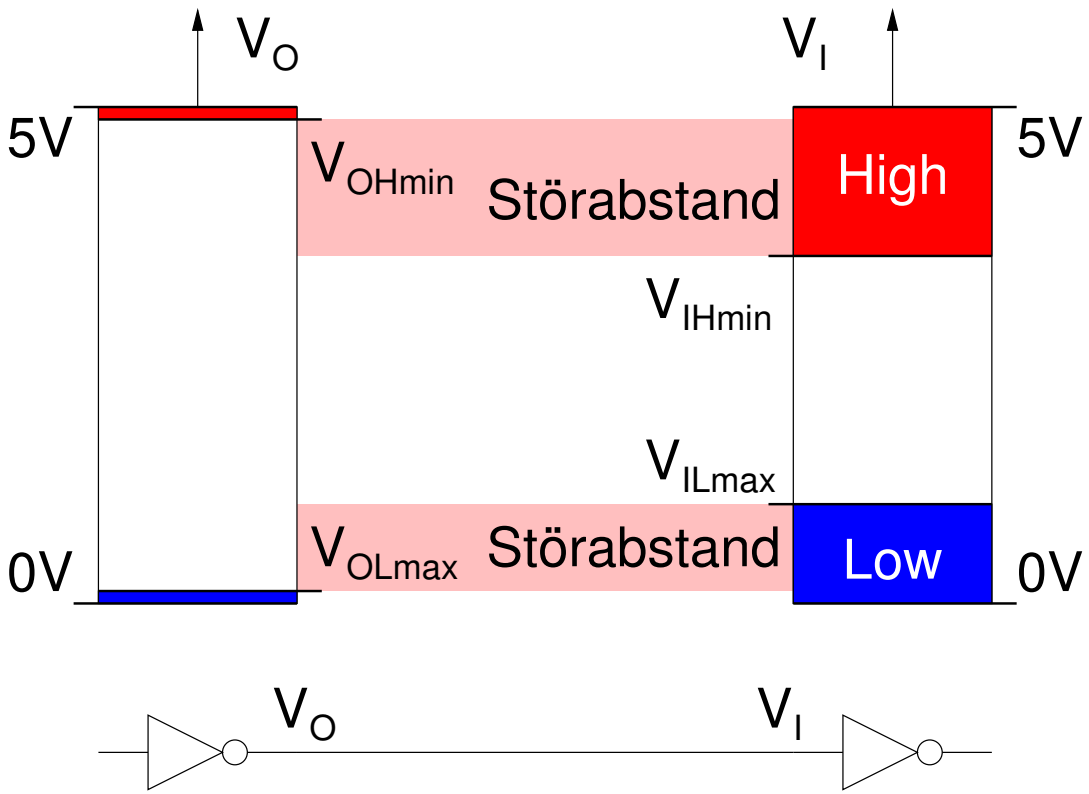
Spannungs-Level für CMOS 5V

Name	Beschreibung	Typischer Wert
V_{IHmax}	max. Spannung erkannt als logisch 1	5.0 V
V_{IHmin}	min. Spannung erkannt als logisch 1	3.5 V
V_{ILmax}	max. Spannung erkannt als logisch 0	1.0 V
V_{ILmin}	min. Spannung erkannt als logisch 0	0.0 V
V_{OHmax}	max. Spannung erzeugt als logisch 1	5.0 V
V_{OHmin}	min. Spannung erzeugt als logisch 1	4.9 V
V_{OLmax}	max. Spannung erzeugt als logisch 0	0.1 V
V_{OLmin}	min. Spannung erzeugt als logisch 0	0.0 V

Inverter Kennlinie



Störabstand erlaubt erst zuverlässige digitale Schaltungen!



Zwei Inverter in Serie

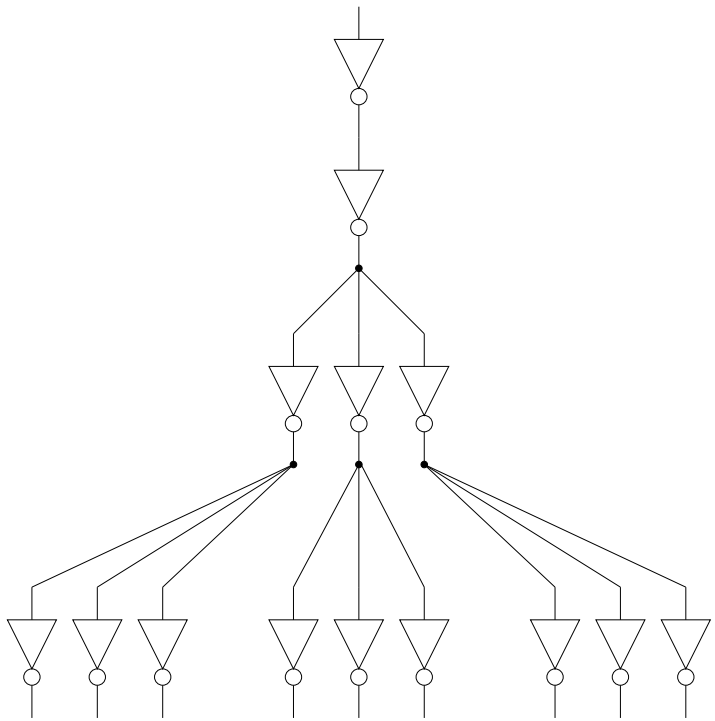
Fanout-Begrenzung

Fanout = Anzahl Gatter verbunden mit dem Ausgang eines Gatters

- Begrenzung Max-Fanout in TTL:
  Verteilung des Stromes verringert erreichte Eingangsspannung
  Typischer Wert 20 Gatter

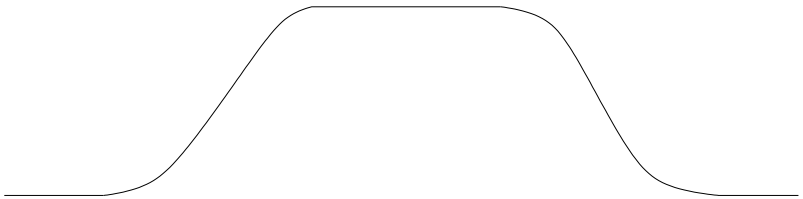
- Begrenzung Max-Fanout in CMOS:
  geringer Stromfluss
  aber kapizitäre Effekte der Verbindungen (interconnect)
  hohe Kapazität am Ausgang führt zu langen Schaltzeiten
  lässt sich erst nach dem Layout berechnen

Abhilfe bei Überschreiten der Fanoutbegrenzung:

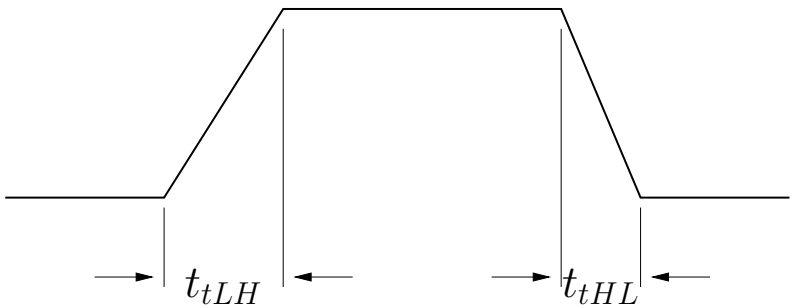


Fanout-Baum (engl. Fanout-Tree)

Bisherige Idealisierung: Umschalten ist unendlich schnell

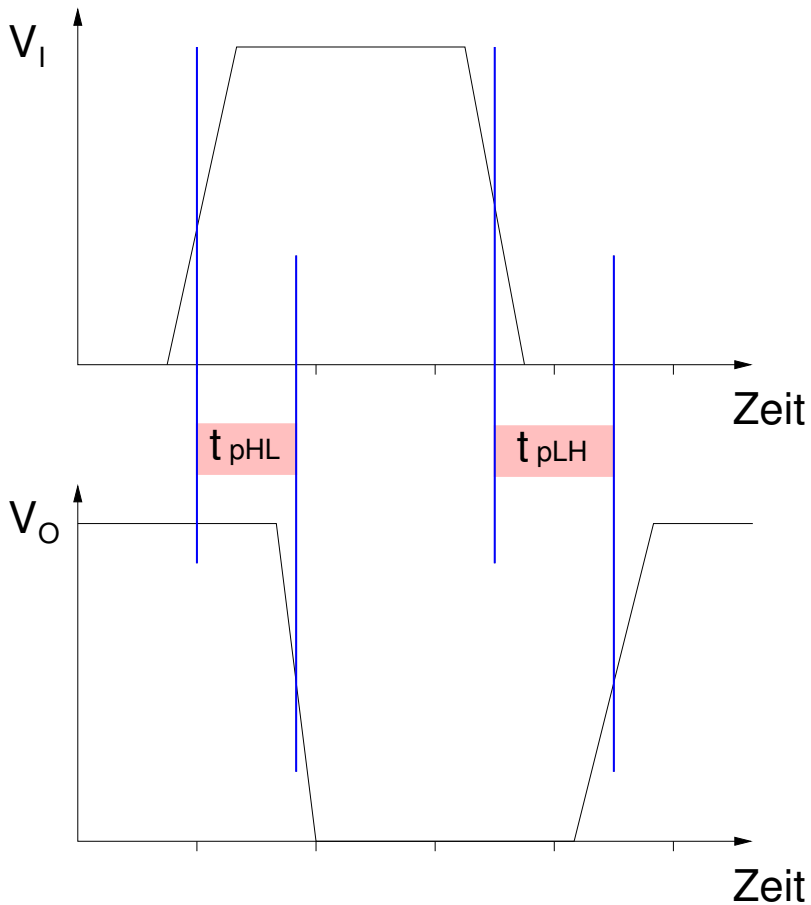


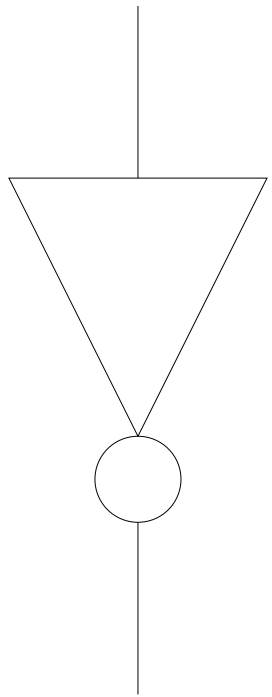
Tatsächlicher Signalüberang



Kompromiss

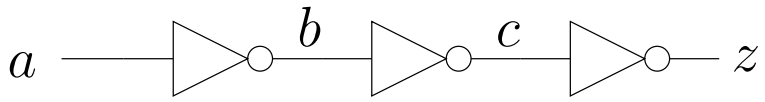
Schaltzeiten

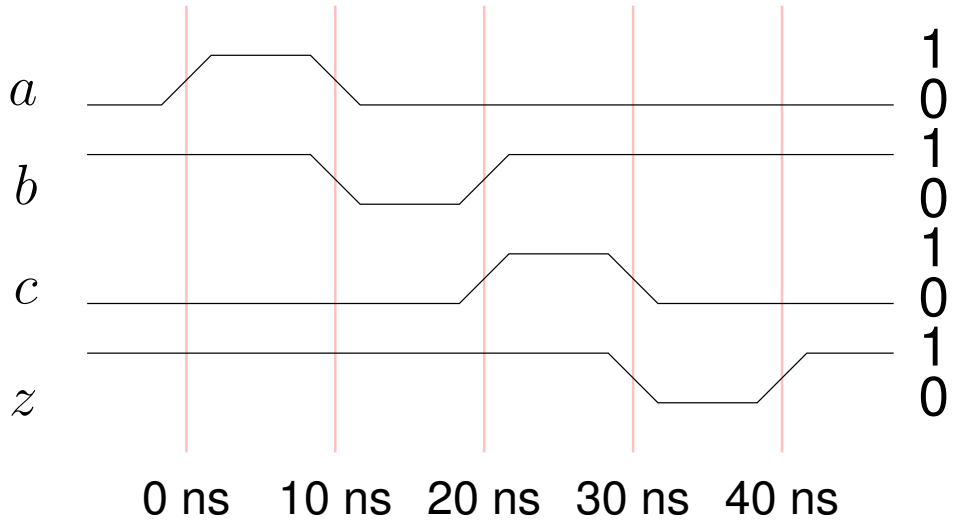




t_{pHL} Schaltzeit von High nach Low
t_{pLH} Schaltzeit von Low nach High

Timing-Diagramm





Das Timing-Diagramm zeigt
zeitliche Abhängigkeiten zwischen den Signalen

Schaltzeiten in Verilog

- Kombinatorische Logik mit Verzögerung:

  assign #*delay* x= a|b;

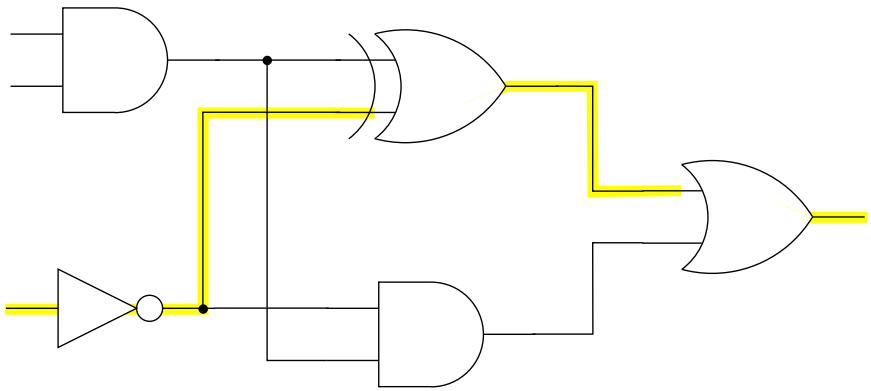
- Nur zur Modellierung und Simulation – das Synthesetool kann die tatsächlichen Schaltzeiten nicht beeinflussen

- Generierung von Testvektoren mit einem **initial** **begin** ... **end** Block

Schaltzeiten in Verilog

```
‘timescale 1 ns / 1 ns

module gate_delay(input a, output b, c, z);
  assign #10 b = !a;
  assign #10 c = !b;
  assign #10 z = !c;
endmodule

module gate_delay_tb;
  wire b, c, z;
  reg a;

  gate_delay M(a, b, c, z);

  initial begin
    a=0; #50; a=1; #10; a=0;
  end

endmodule
```

Kritischer Pfad



Gatter	Laufzeit
AND	3 ns
OR	4 ns
XOR	7 ns
NOT	5 ns

Gesamtverzögerung: NOT+XOR+OR = 5+7+4 = 16 ns

Längster Pfad

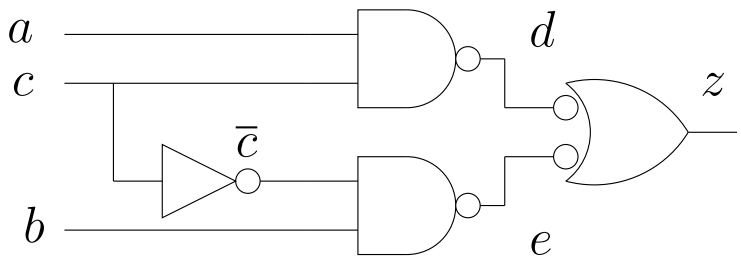
① Die Eingänge der Schaltung mit 0 beschriften.

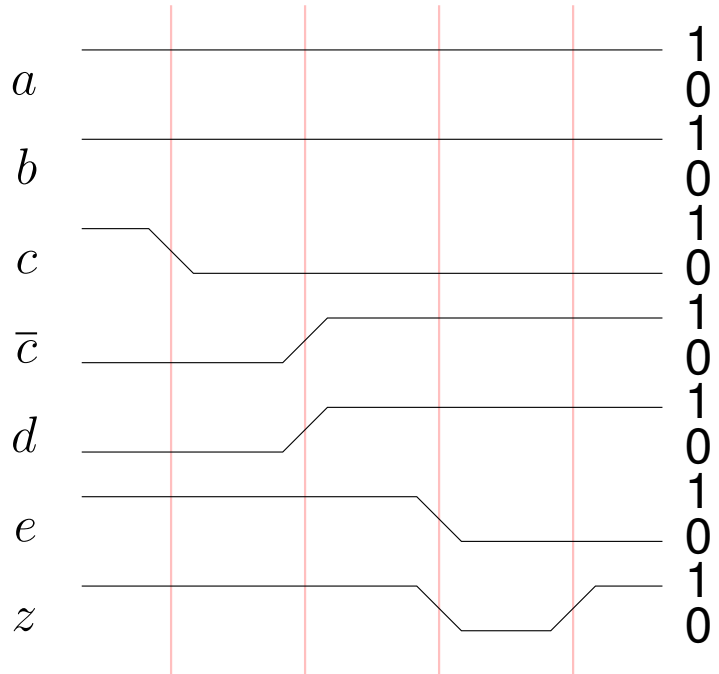
② Für alle Gatter, deren Eingänge beschriftet sind: den Ausgang des Gatters mit $\max\{\textit{Eingänge}\} + t_p$ beschriften.

③ Sind noch unbeschriftete Ausgangsleitungen vorhanden, dann gehe zu ②.

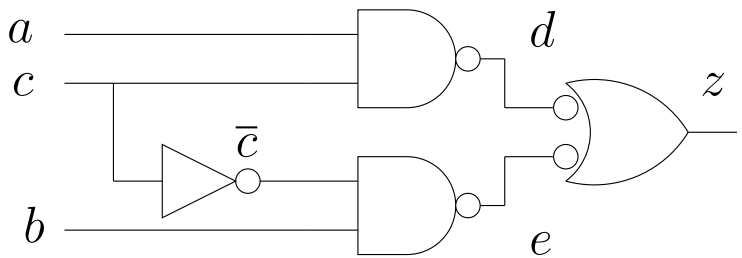
④ Die Länge des längsten Pfades ist $\max\{\textit{Ausgänge}\}$.

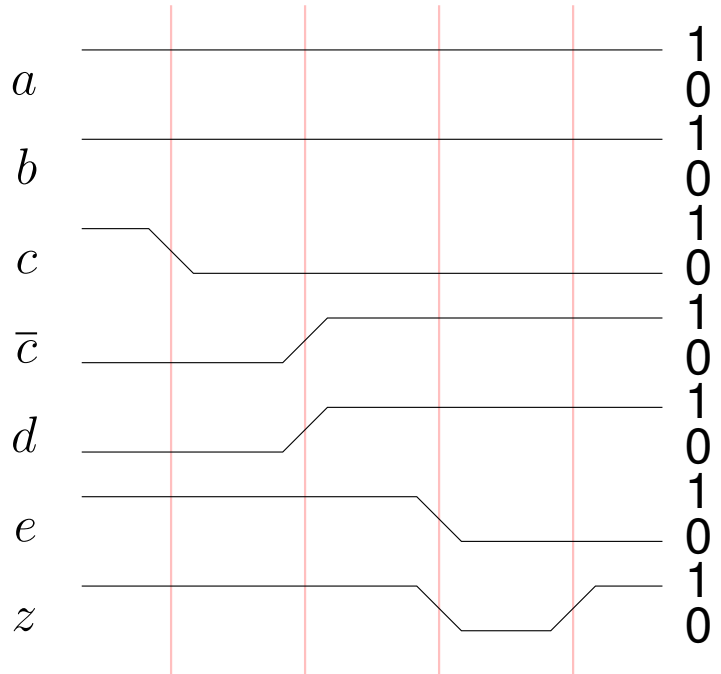
Hazards





Hazards





✘ Ausgangssignal z wechselt zweimal bevor es stabil wird!

Latches und Flipflops

- Bisher: alles kombinatorisch
- Ausgabewerte sind Funktion der Eingangswerte
- Wir würden aber gerne Daten speichern!

Latches und Flipflops

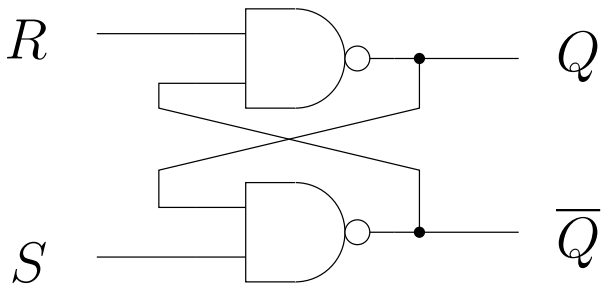
- Bisher: alles kombinatorisch
- Ausgabewerte sind Funktion der Eingangswerte
- Wir würden aber gerne Daten speichern!

- Latches sind pegelgesteuerte Speicher-Elemente
- Flipflops sind flankengesteuerte Speicher-Elemente

- engl. *edge triggered* vs. *level sensitive*

RS-Latch mit NANDs

S = Set $\qquad$ R = Reset

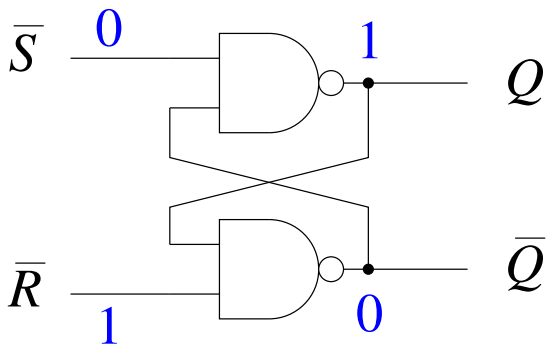


S	R	Q'	$\overline{Q}'$
0	0	1	1
0	1	0	1
1	0	1	0
1	1	Q	$\overline{Q}$

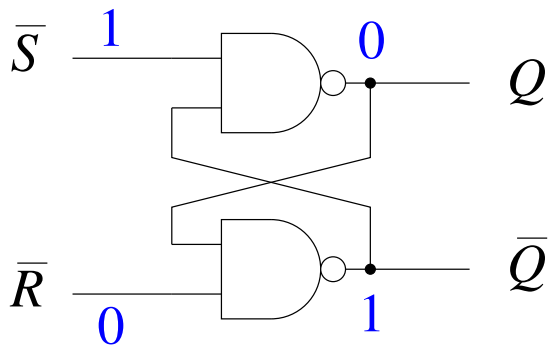
Strich in Q' bedeutet Wert im *nächsten Zustand* (keine Negation)

Erste Zeile ist unerwünscht, da Q' und $\overline{Q}'$ nicht komplementär!

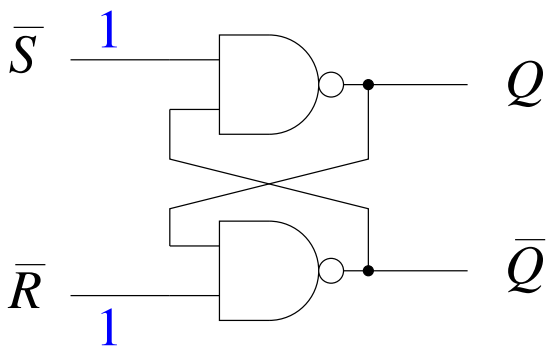
Einfacher RS-Latch



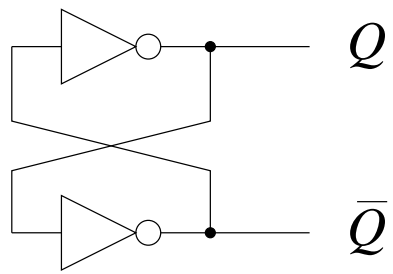
Set



Reset

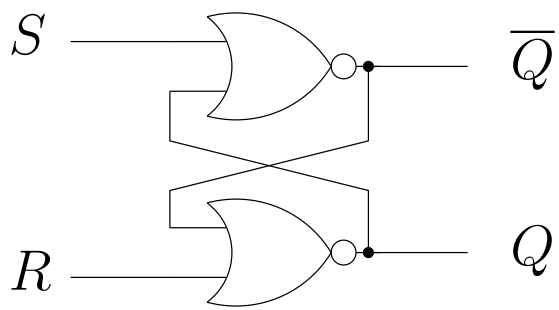


Halten



Halten mit Invertern

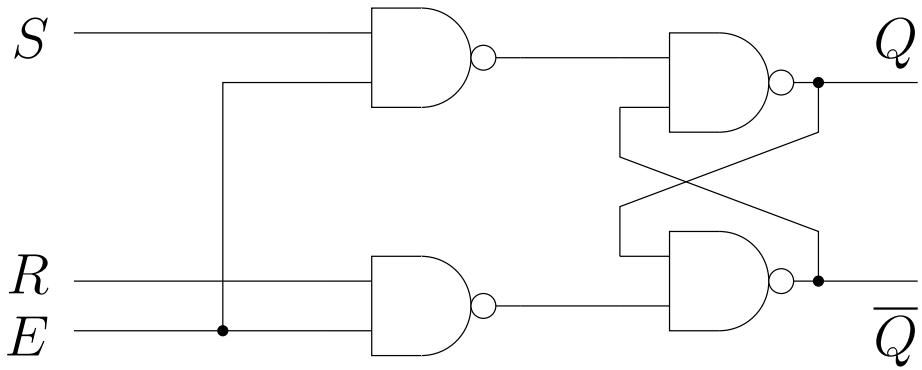
RS-Latch mit NORs



- man kann auch statt NANDs zwei NORs nehmen
  (Q liegt dann am Ausgang vom R gespeisten NOR)

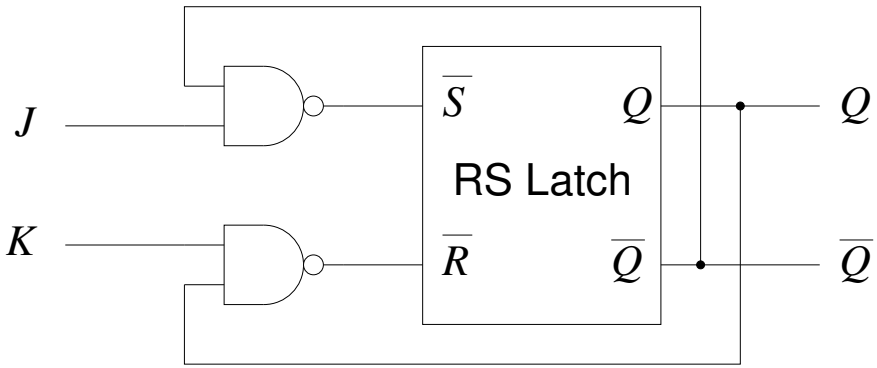
- NAND Variante hat *active-low* Set & Reset
- NOR Variante hat *active-high* Set & Reset

Level-Sensitiver RS-Latch mit Enable-Signal



- R und S werden nach außen geführt maskiert durch E
  (R = Reset, S = Set, E = Enable)
- zusätzliches Enable-Signal und Maskierungslogik vermeidet Hazards
- Annahme: R und S sind nie gleichzeitig 1 wenn $E = 1$
  (sonst hat man eben $Q = \overline{Q} = 1$)

Level-Sensitiver JK-Latch

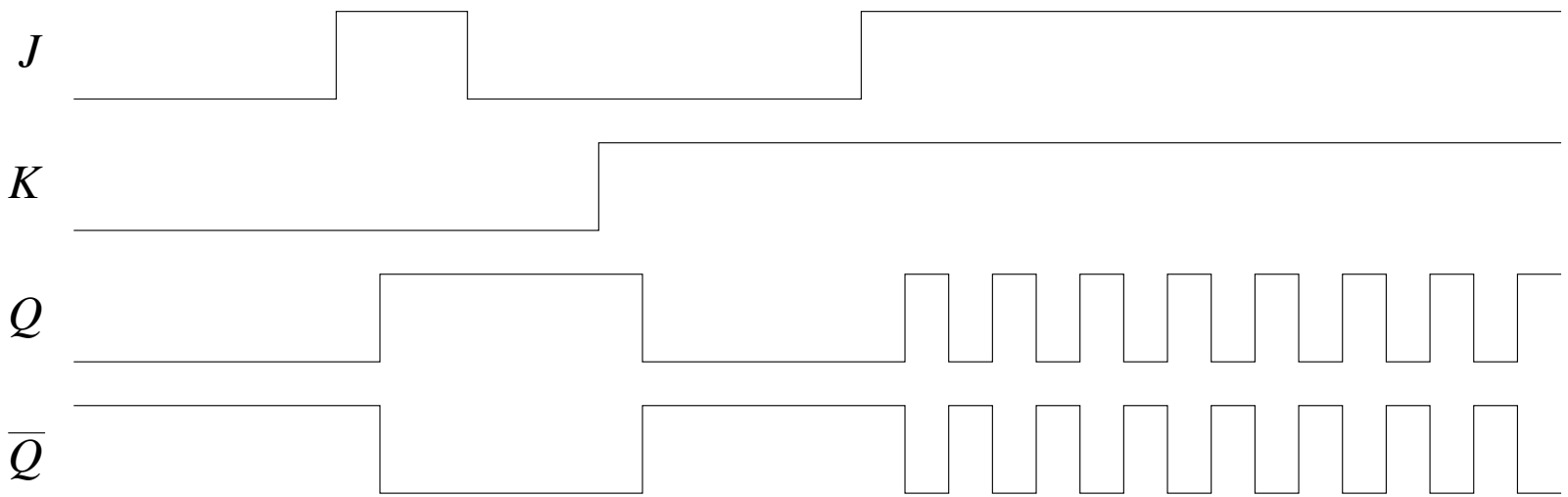


$$Q' \equiv Q \cdot \overline{K} \vee \overline{Q} \cdot J$$

J	K	Q'	$\overline{Q}'$
0	0	Q	$\overline{Q}$
0	1	0	1
1	0	1	0
1	1	$\overline{Q}$	Q

✔ Vorteil: Q ist immer $\neg\overline{Q}$!

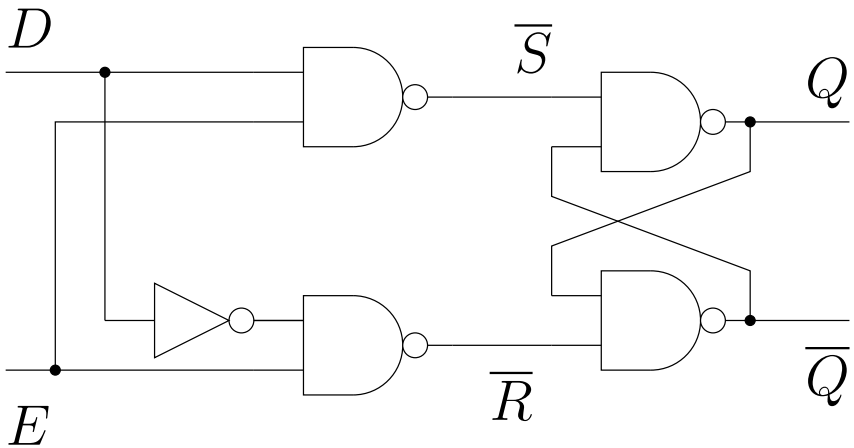
Nachteil des JK-Latchs



- ✘ Oszillation tritt ein bei $(J, K) = (1, 1)$
- Oszillations-Frequenz abhängig vom Delay

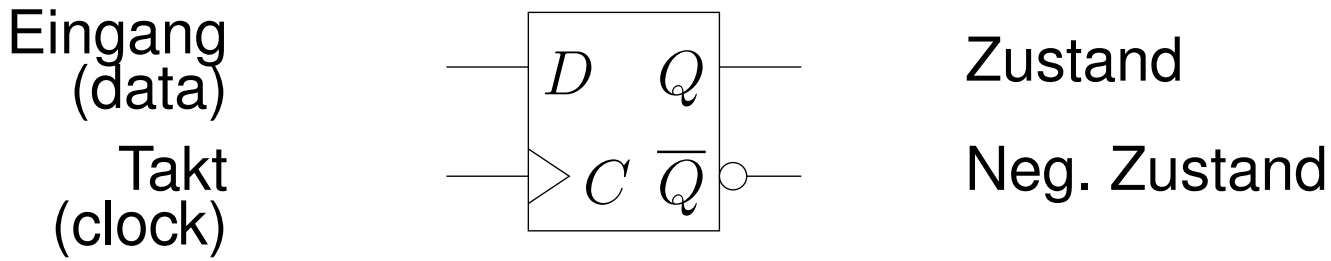
- $(J, K) = (1, 1)$ darf also nicht zu lange anliegen
- ✘ dies lässt sich in der Praxis nicht erreichen

D-Latch



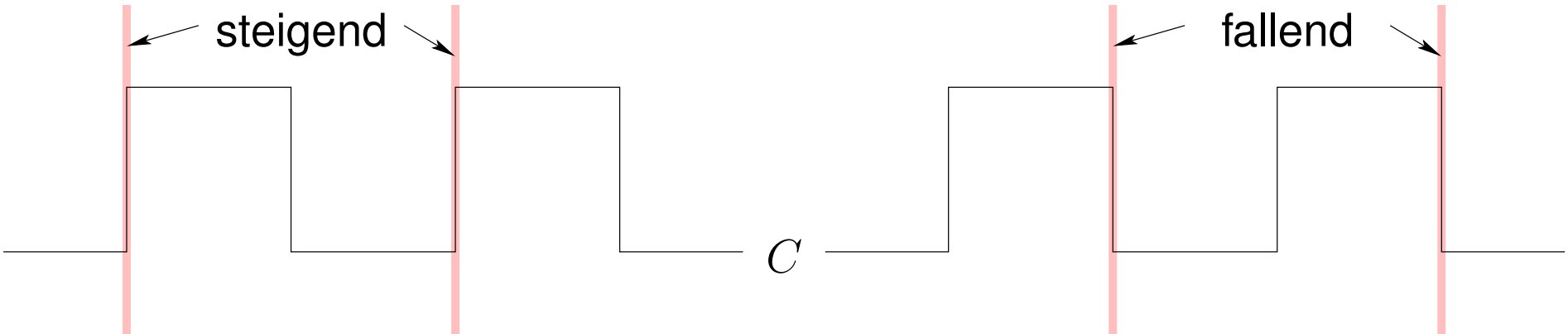
- zwei-stufiger level-sensitiver Schaltkreis
- mit Enable-Signal E
- zweite Stufe: bekannter RS-Latch (speichert den Zustand)

D-Flipflop



- meistgebrauchtes und einfachstes sequentielles Bauteil:
  speichert Eingangswert für einen Taktzyklus
- Dreieck bedeutet *flankengesteuert* (engl. edge triggered)
  entweder steigende oder fallende Flanke
- oft mit zusätzlichem asynchronem *reset* oder *set* Eingang

Positive und negative Flankensteuerung



engl. „falling and rising clock edge“

üblicherweise steigende Flanke (positive Flanke)

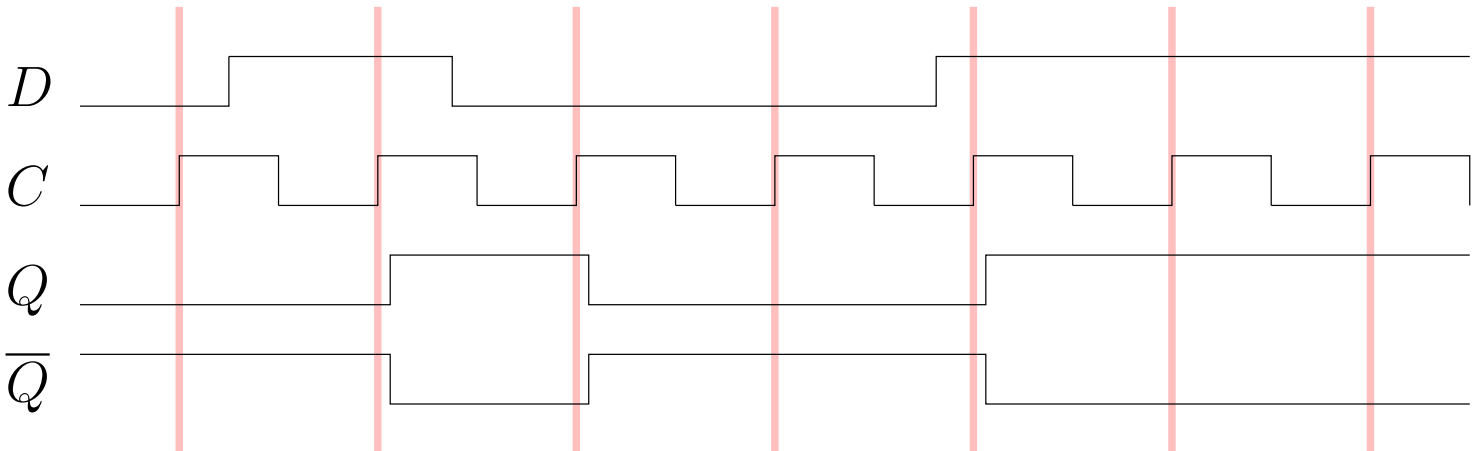
Übergangstabelle

D	C	Q'	$\overline{Q}'$
0	$\uparrow$	0	1
1	$\uparrow$	1	0
–	$\downarrow$	Q	$\overline{Q}$
–	0	Q	$\overline{Q}$
–	1	Q	$\overline{Q}$

Erinnerung: Strich in Q' bedeutet Wert im *nächsten Zustand*

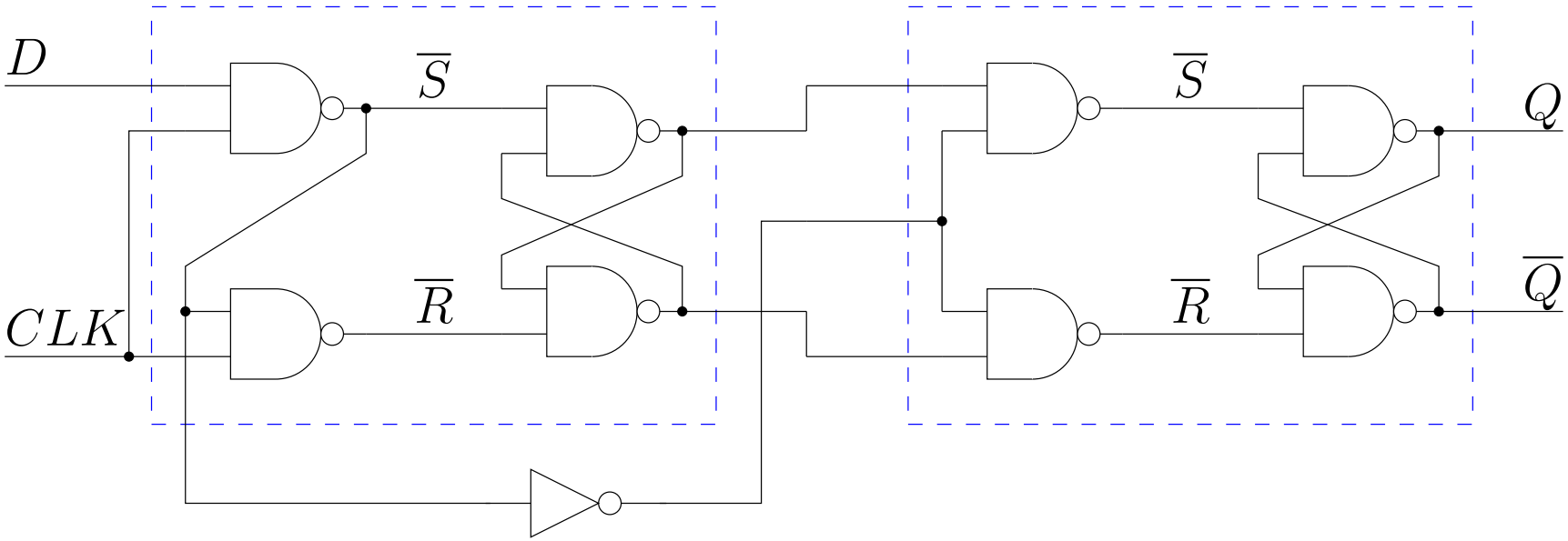
$\uparrow$ positive Flanke, $\downarrow$ negative Flanke

D-Flipflop mit positiver Flankensteuerung



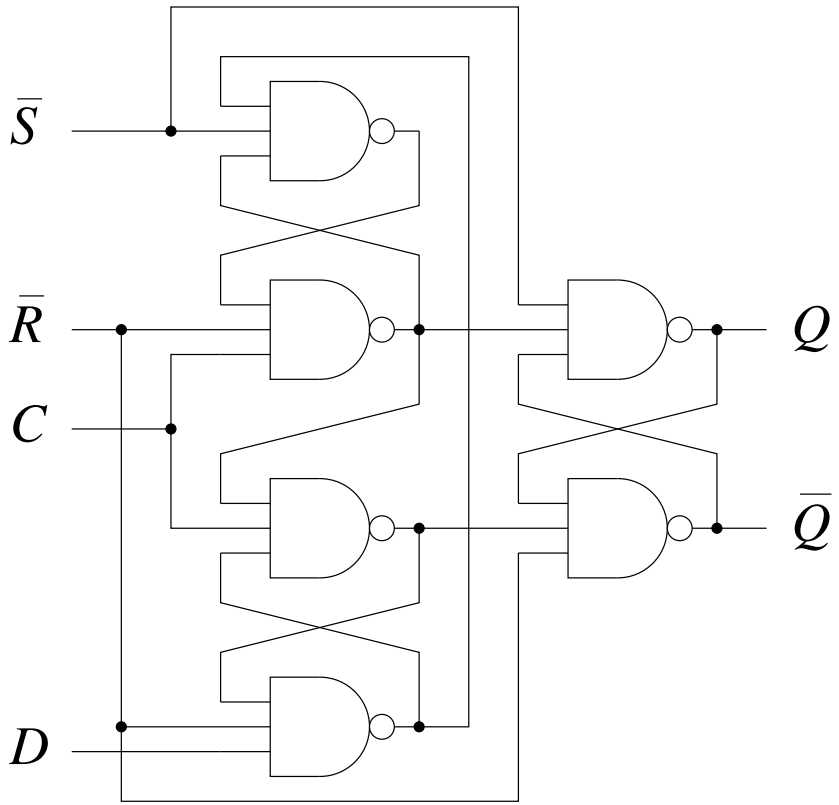
Änderung des Eingangswerts wirkt sich erst bei der nächsten Flanke aus

Master-Slave D-Flipflop



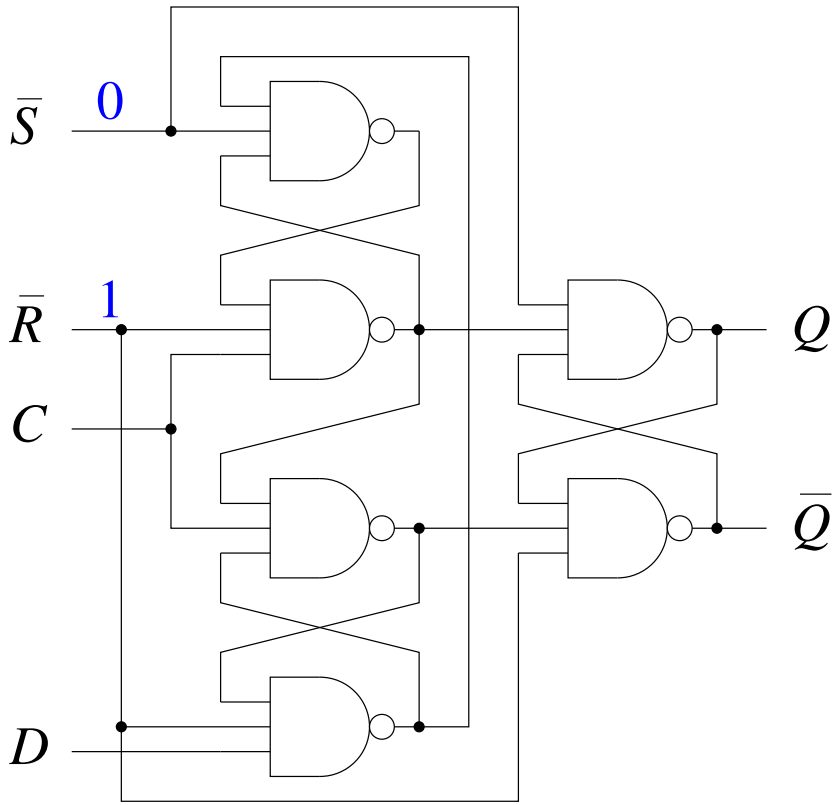
- Aufgebaut aus zwei D-Latches („Master“ und „Slave“)
- Master arbeitet an der steigenden Flanke
- Slave arbeitet an der fallenden Flanke (erzeugt Q und $\overline{Q}$)

D-Flipflop mit asynchronem Reset & Set



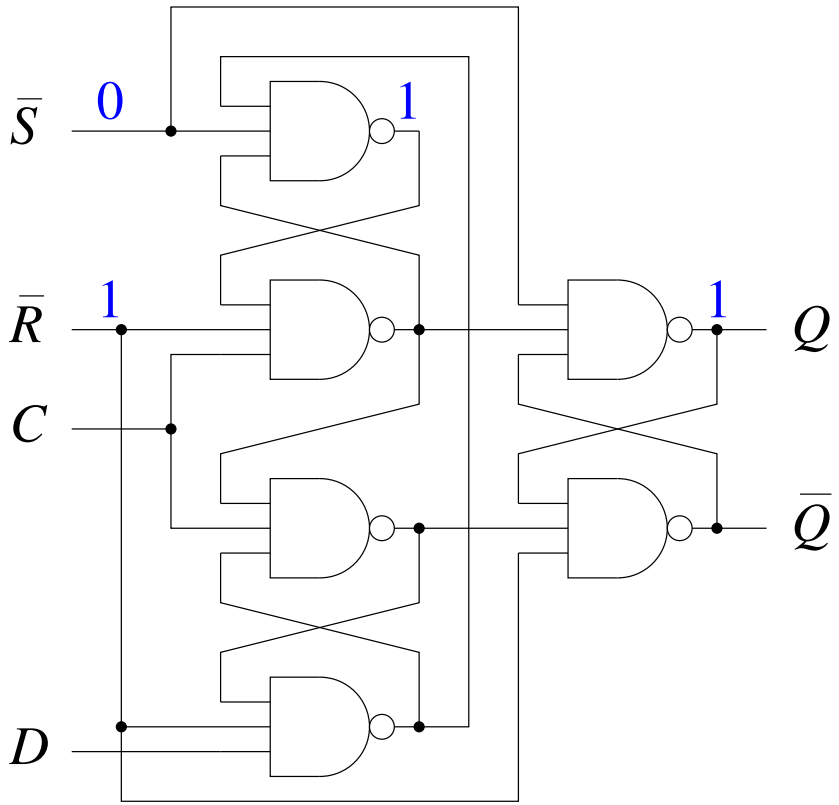
Set: $(\overline{R}, \overline{S}) = (1, 0)$

D-Flipflop mit asynchronem Reset & Set



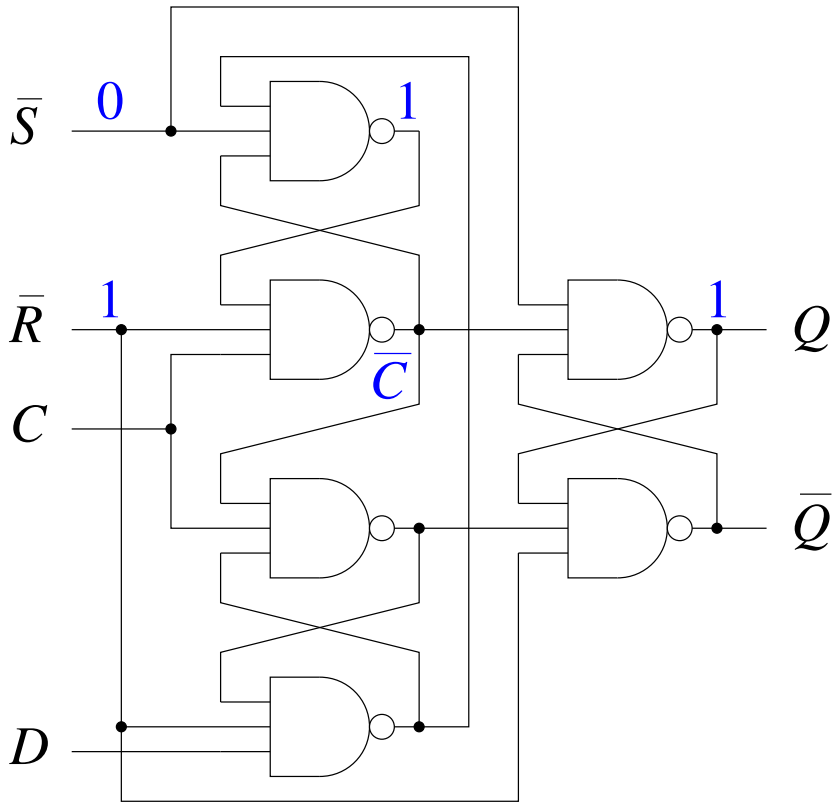
Set: $(\overline{R}, \overline{S}) = (1, 0)$

D-Flipflop mit asynchronem Reset & Set



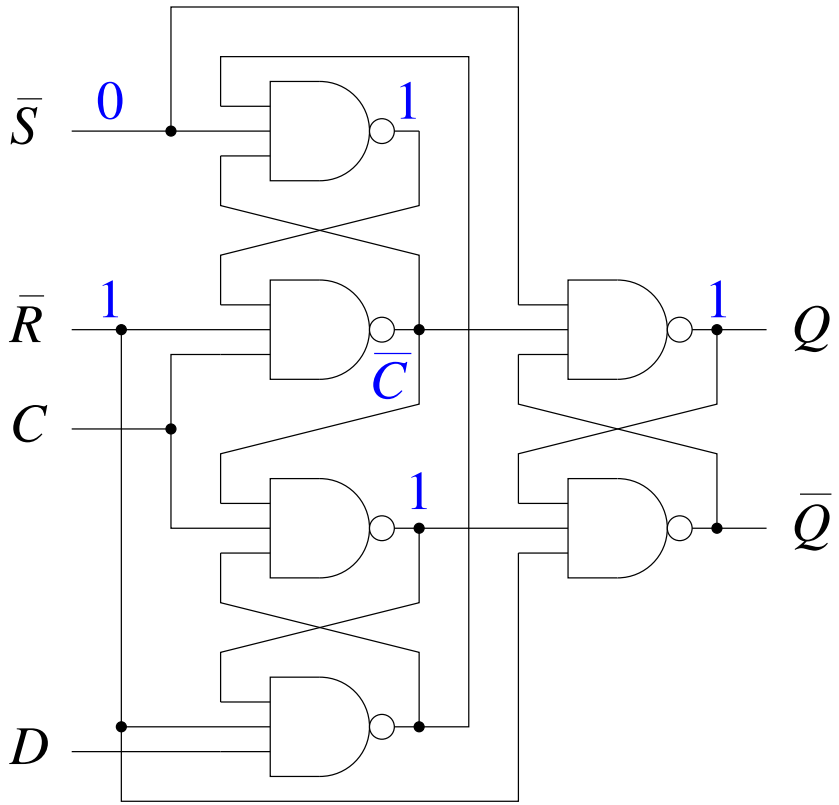
Set: $(\overline{R}, \overline{S}) = (1, 0)$

D-Flipflop mit asynchronem Reset & Set



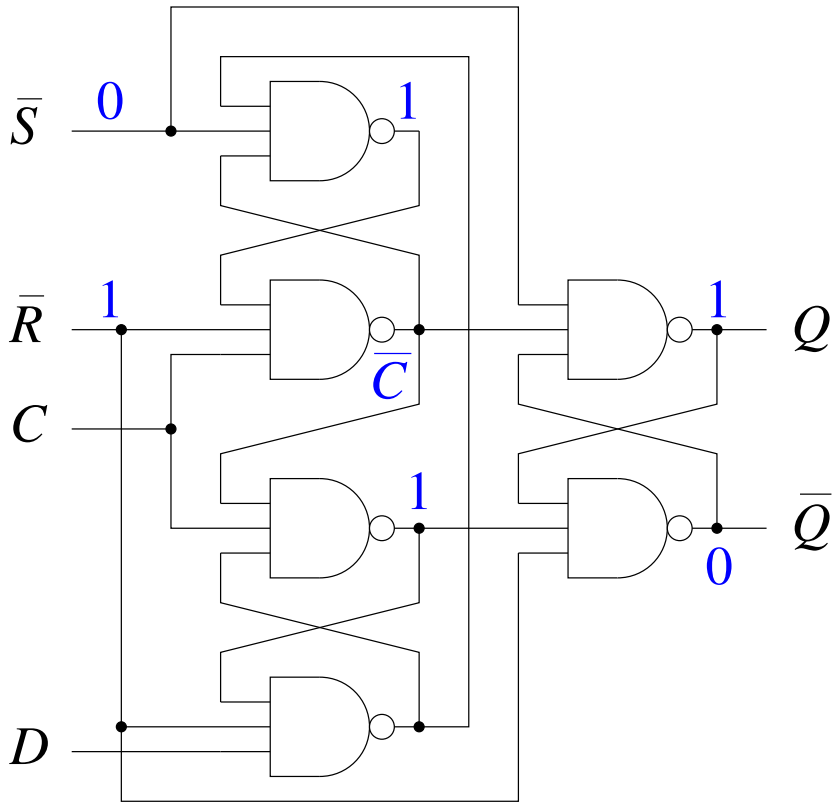
Set: $(\overline{R}, \overline{S}) = (1, 0)$

D-Flipflop mit asynchronem Reset & Set



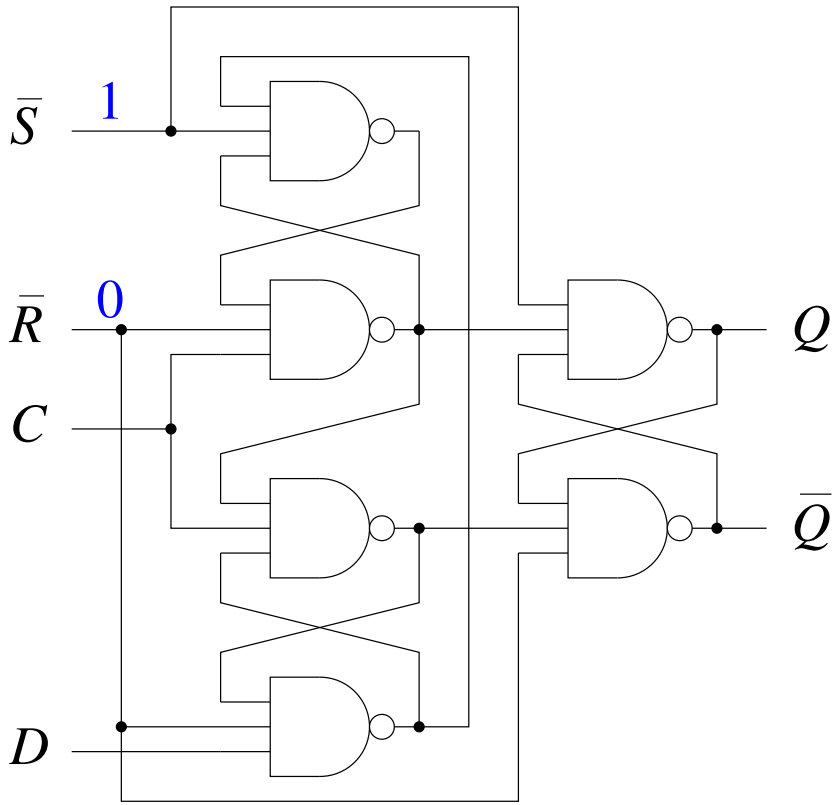
Set: $(\overline{R}, \overline{S}) = (1, 0)$

D-Flipflop mit asynchronem Reset & Set



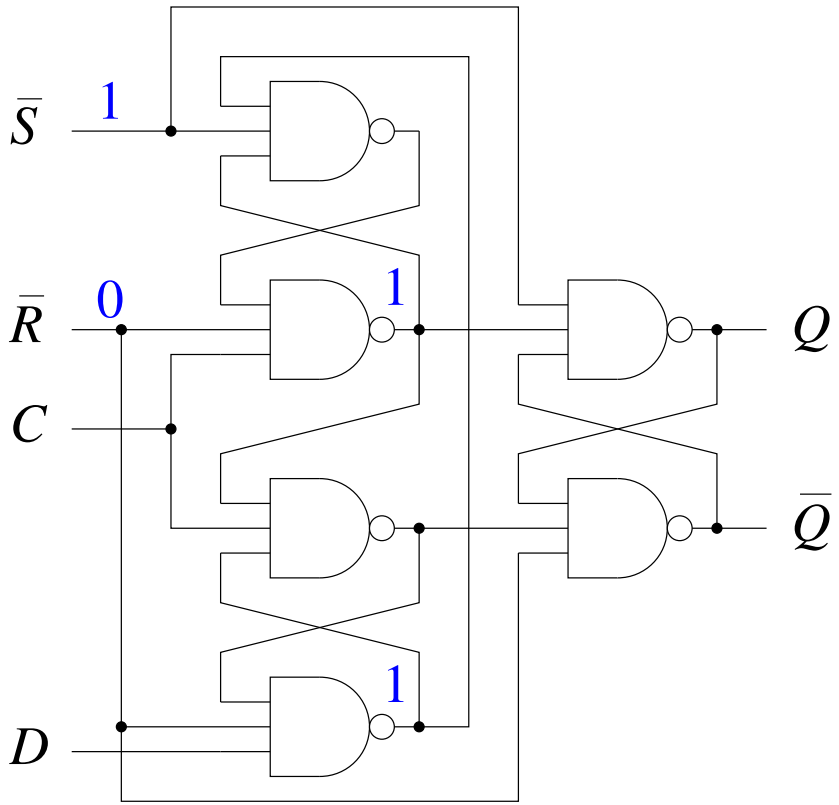
Set: $(\overline{R}, \overline{S}) = (1, 0)$

D-Flipflop mit asynchronem Reset & Set



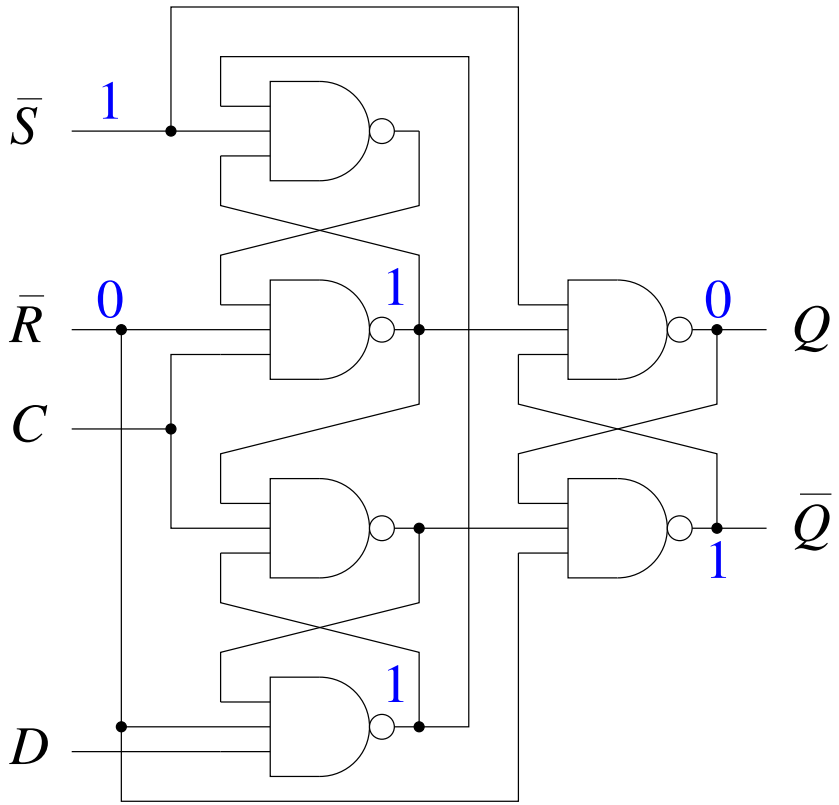
Reset: $(\overline{R}, \overline{S}) = (0, 1)$

D-Flipflop mit asynchronem Reset & Set



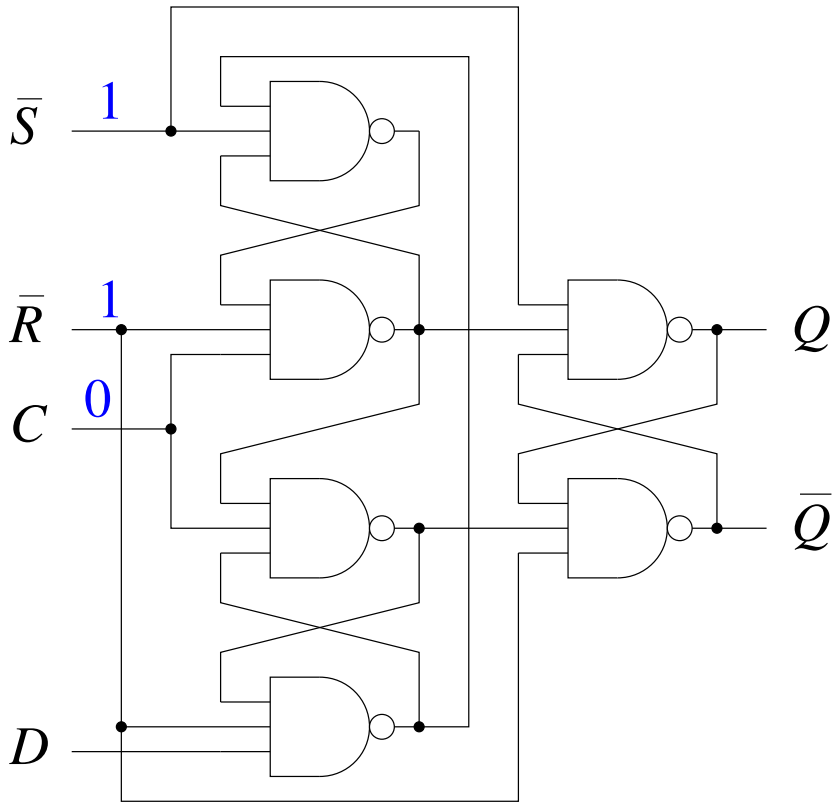
Reset: $(\overline{R}, \overline{S}) = (0, 1)$

D-Flipflop mit asynchronem Reset & Set



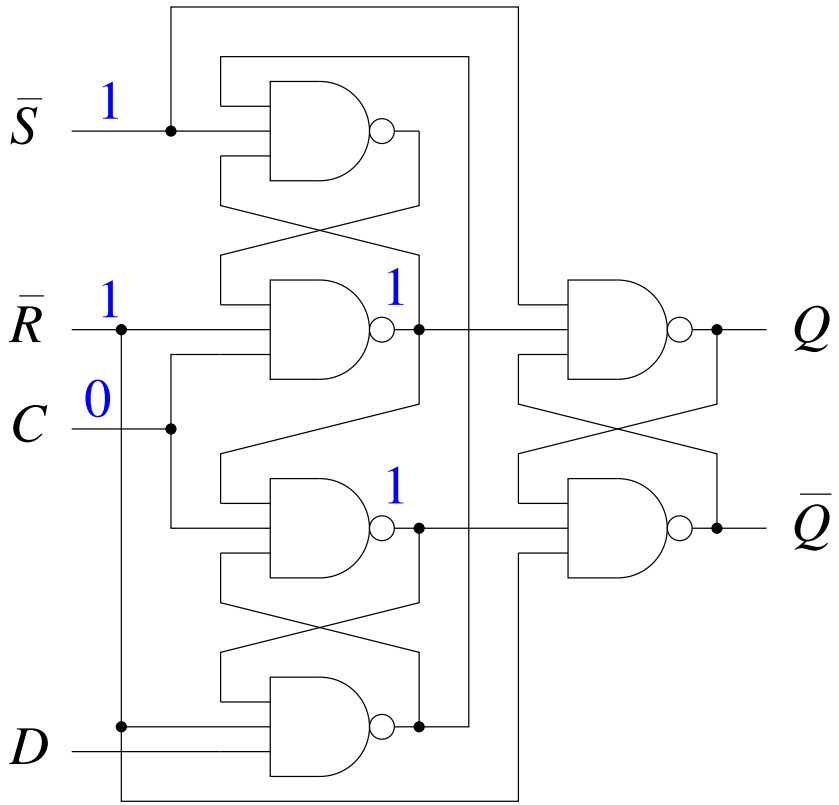
Reset: $(\overline{R}, \overline{S}) = (0, 1)$

D-Flipflop mit asynchronem Reset & Set



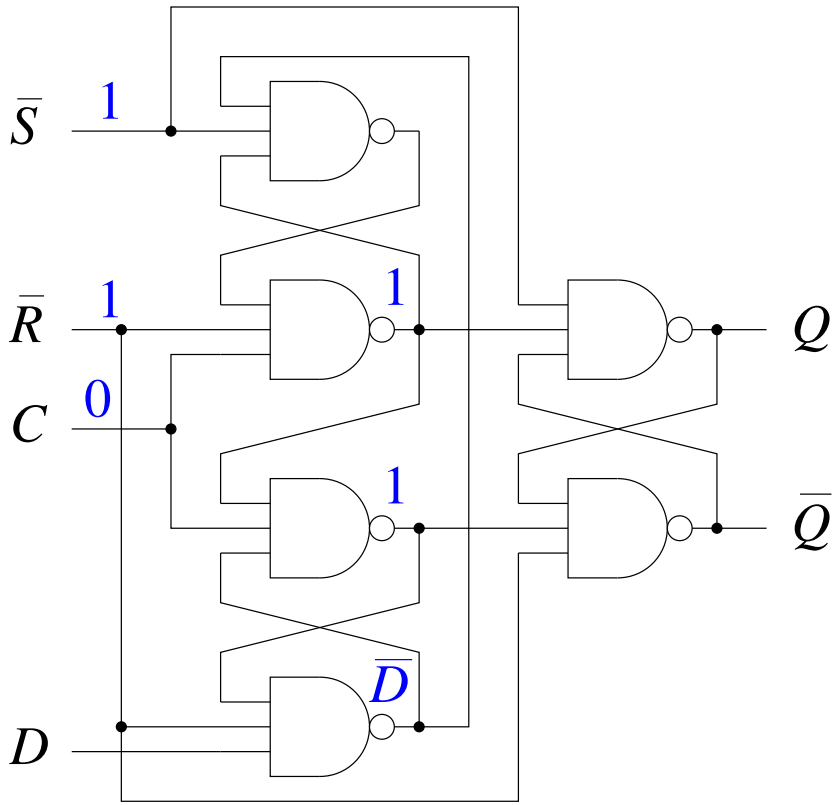
Clock Low: $(\overline{R}, \overline{S}, C) = (1, 1, 0)$

D-Flipflop mit asynchronem Reset & Set



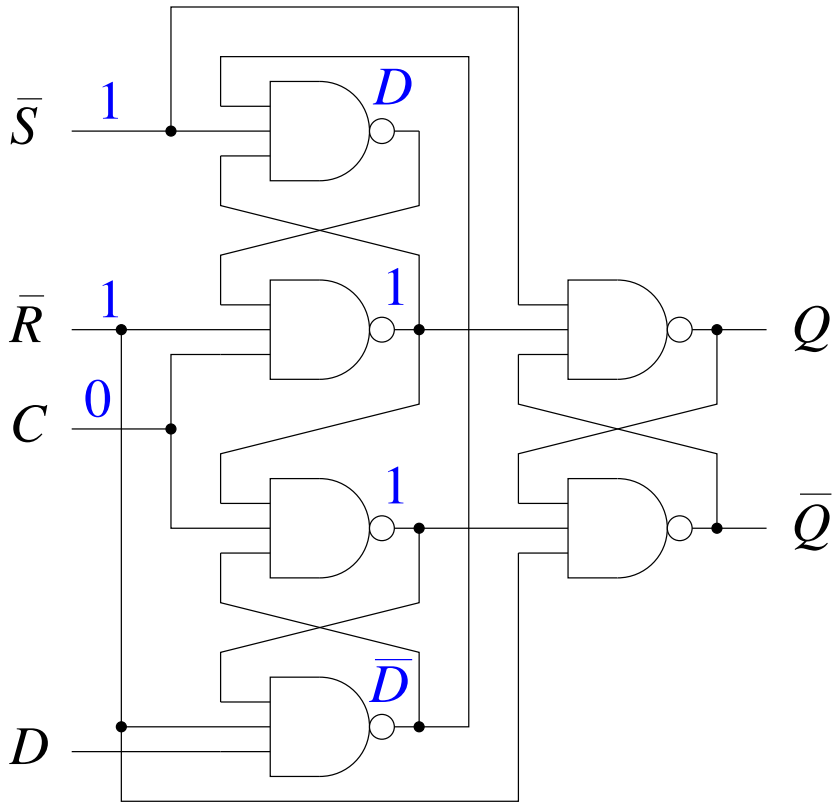
Clock Low: $(\overline{R}, \overline{S}, C) = (1, 1, 0)$

D-Flipflop mit asynchronem Reset & Set



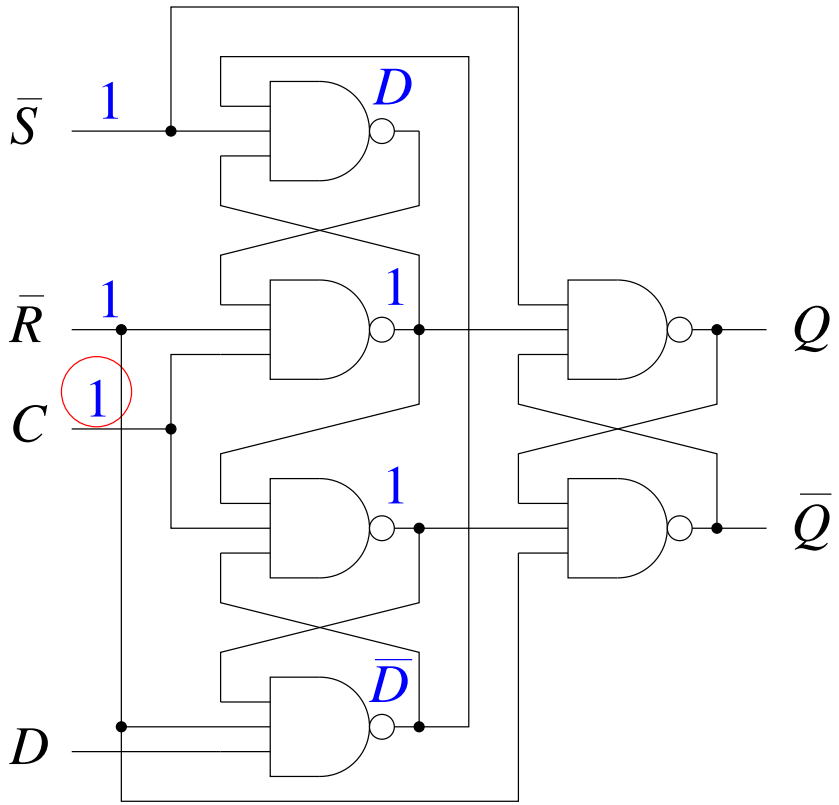
Clock Low: $(\overline{R}, \overline{S}, C) = (1, 1, 0)$

D-Flipflop mit asynchronem Reset & Set



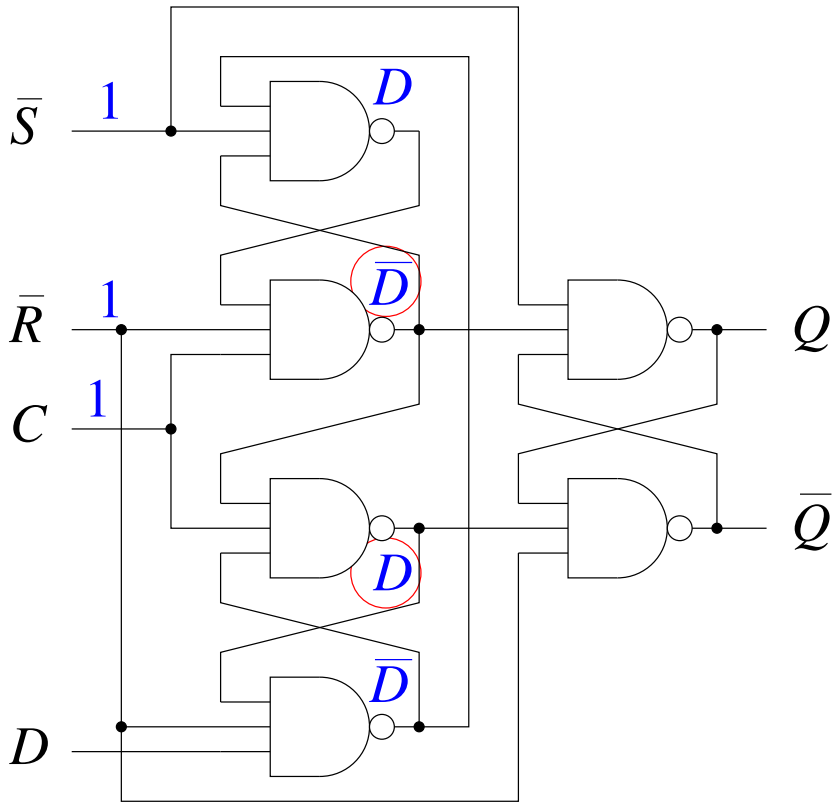
Clock Low: $(\overline{R}, \overline{S}, C) = (1, 1, 0)$

D-Flipflop mit asynchronem Reset & Set



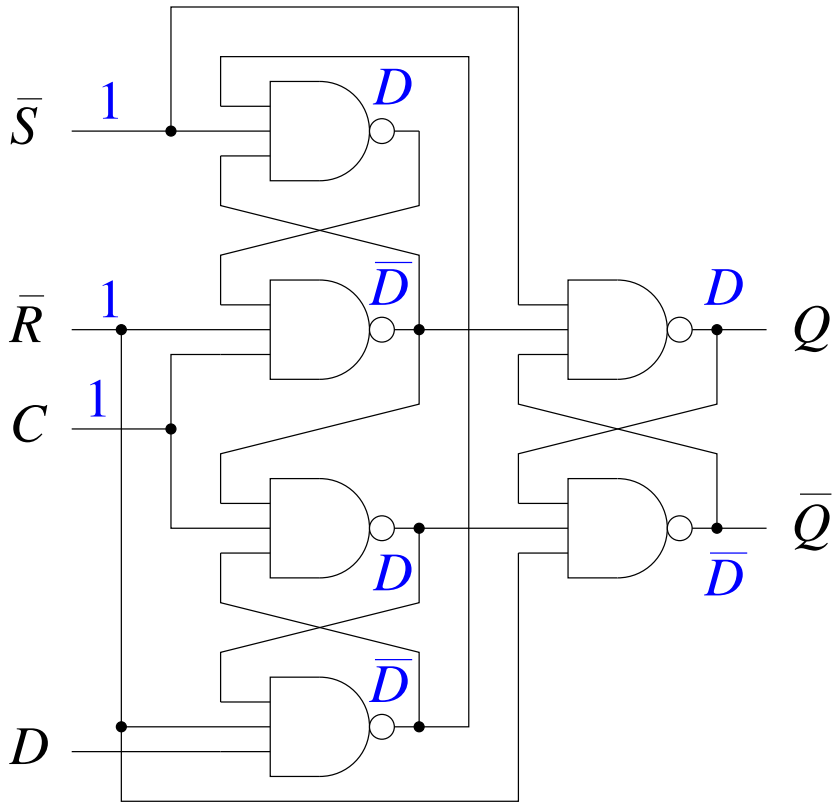
Clock Positive Flanke: $(\overline{R}, \overline{S}, C) = (1, 1, \uparrow)$

D-Flipflop mit asynchronem Reset & Set



Clock Positive Flanke: $(\overline{R}, \overline{S}, C) = (1, 1, \uparrow)$

D-Flipflop mit asynchronem Reset & Set



Clock Positive Flanke: $(\overline{R}, \overline{S}, C) = (1, 1, \uparrow)$

D-Flipflop mit asynchronem Reset & Set

- nur an Flanke wird Zustand der ersten Stufe in die zweite übernommen
- Analyse muss explizit alle Transitionen betrachten
- ✘ Analyse solcher asynchroner Schaltkreise ist sehr mühsam (wir haben nur wenige Szenarien durchgespielt)
- Motivation für Setup- und Hold-Delay erkennbar:

  FFs der 1. Stufe müssen sich erst auf D bzw. $\overline{D}$ einschwingen und unterstes NAND muss noch beim Wechsel von Q weiterhin D lesen

Latch in Verilog

```
reg q;

always @(d, c)
   if (c)  q <= d;
```

standardkonform

```
assign q <= c ? d : q;

assign q <= (d && c) ||
            (q && !c);
```

nicht standardkonform

- Synthese bildet konforme Beschreibung auf Latch aus Bibliothek ab
- nicht konforme Beschreibung evtl. als asynchrone Logik implementiert:
  nicht zu empfehlen, da Gefahr von Hazards

D-Flipflop in Verilog

Flipflops werden standardkonform wie folgt implementiert:

```
reg q;

always @(posedge c)
  q <= d;
```

Achtung:

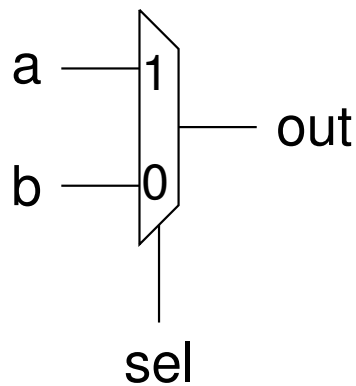
- Genauer Inhalt der Sensitivitätsliste ist wichtig:

  @(**posedge** *clock*)

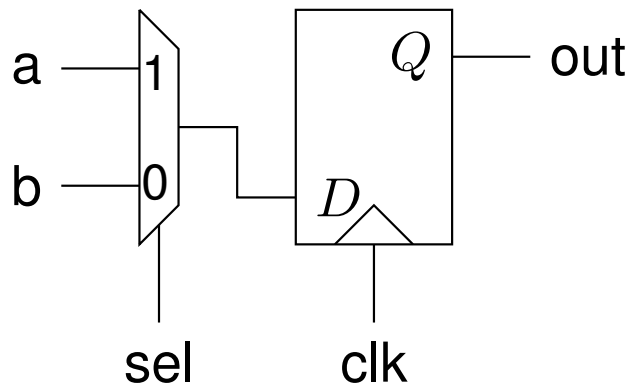
- Zuweisung muss nicht vollständig sein

Verilog: Kombinatorisch vs. sequentiell

```
module combinational(
  input a, b, sel,
  output reg out);

  always @ (a or b or sel)
  begin
    if (sel) out = a;
    else out = b;
  end
endmodule
```

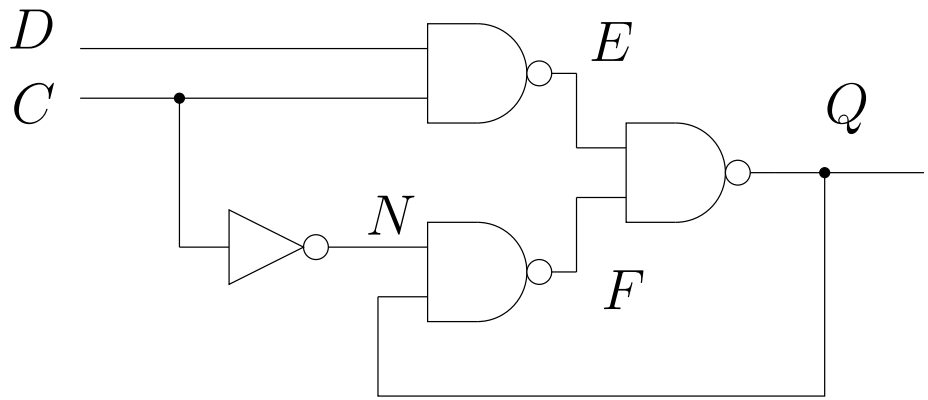



```
module sequential(
  input a, b, sel, clk,
  output reg out);

  always @ (posedge clk)
  begin
    if (sel) out <= a;
    else out <= b;
  end
endmodule
```



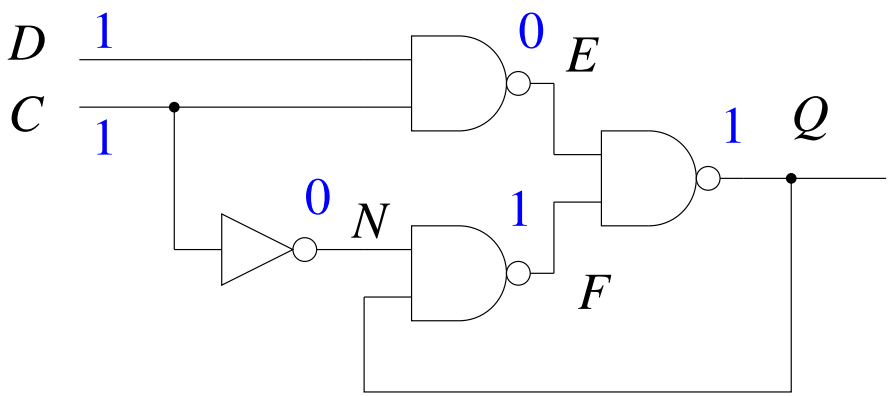
„**reg**“ kann auch rein kombinatorisch sein!

Hazard in selbst-gestricktem D-Latch

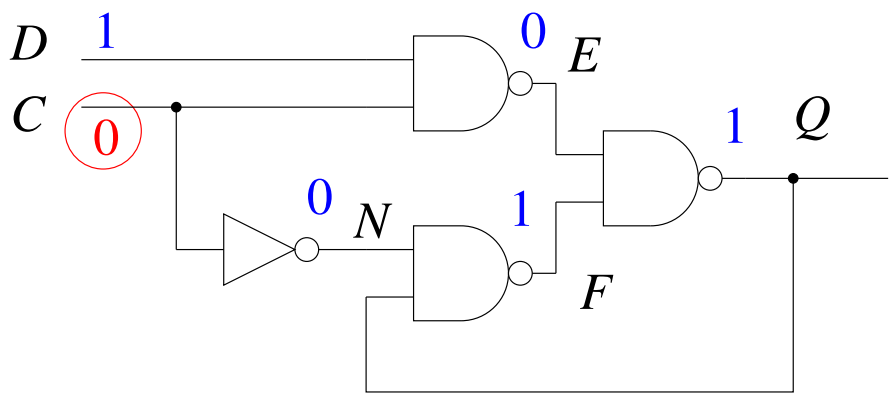


- Annahme: gleiches Unit-Delay für alle Gatter (z.B. 1 ns)
- Wir betrachten den folgenden Übergang:
  $(D, C) = (1, 1)$ nach $(D, C) = (1, 0)$

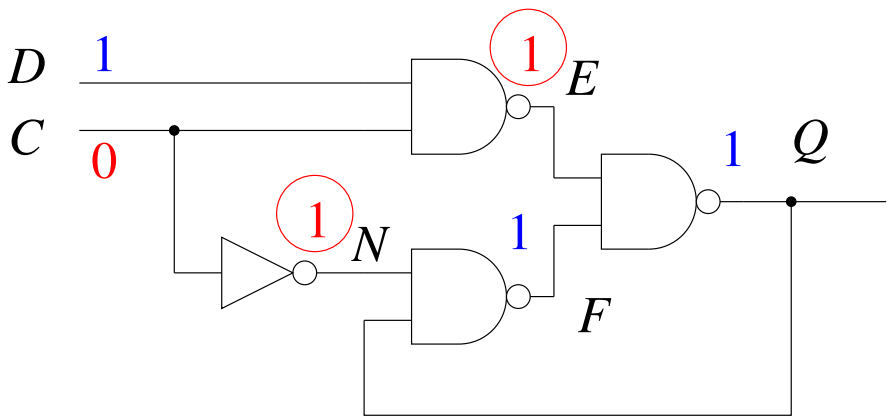
Hazard in selbst-gestricktem D-Latch



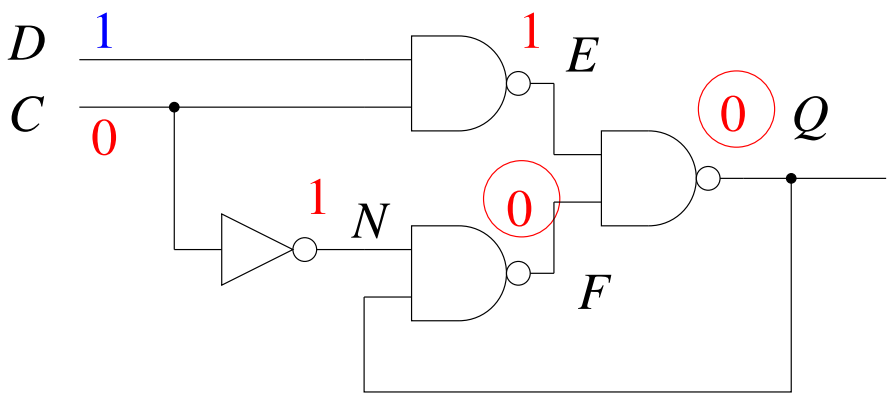
Hazard in selbst-gestricktem D-Latch



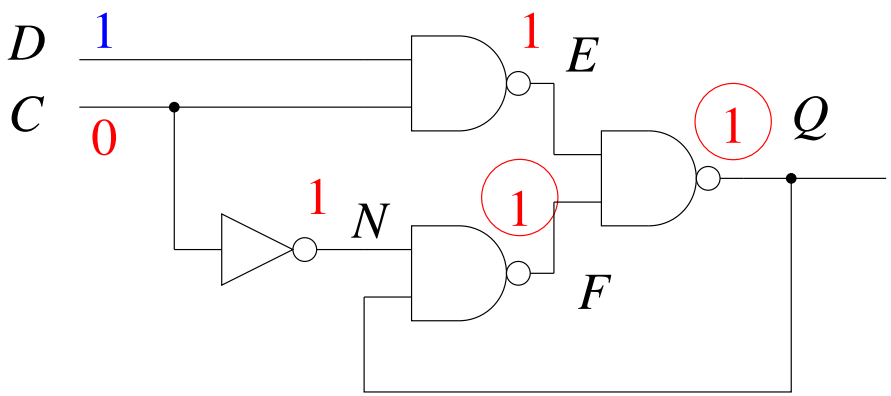
Hazard in selbst-gestricktem D-Latch



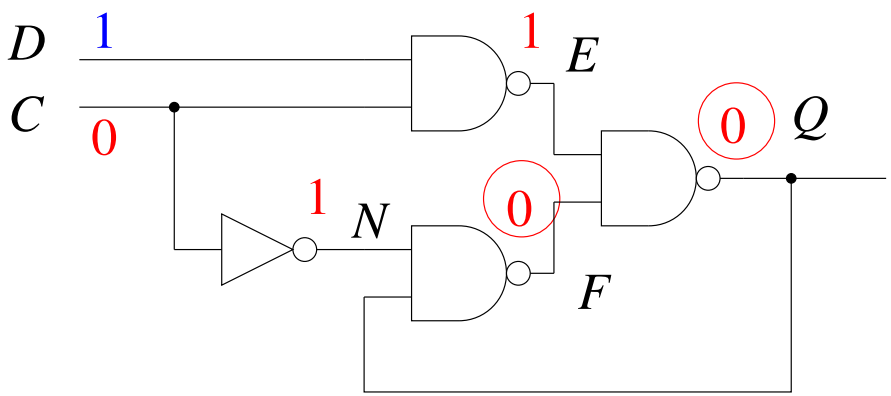
Hazard in selbst-gestricktem D-Latch



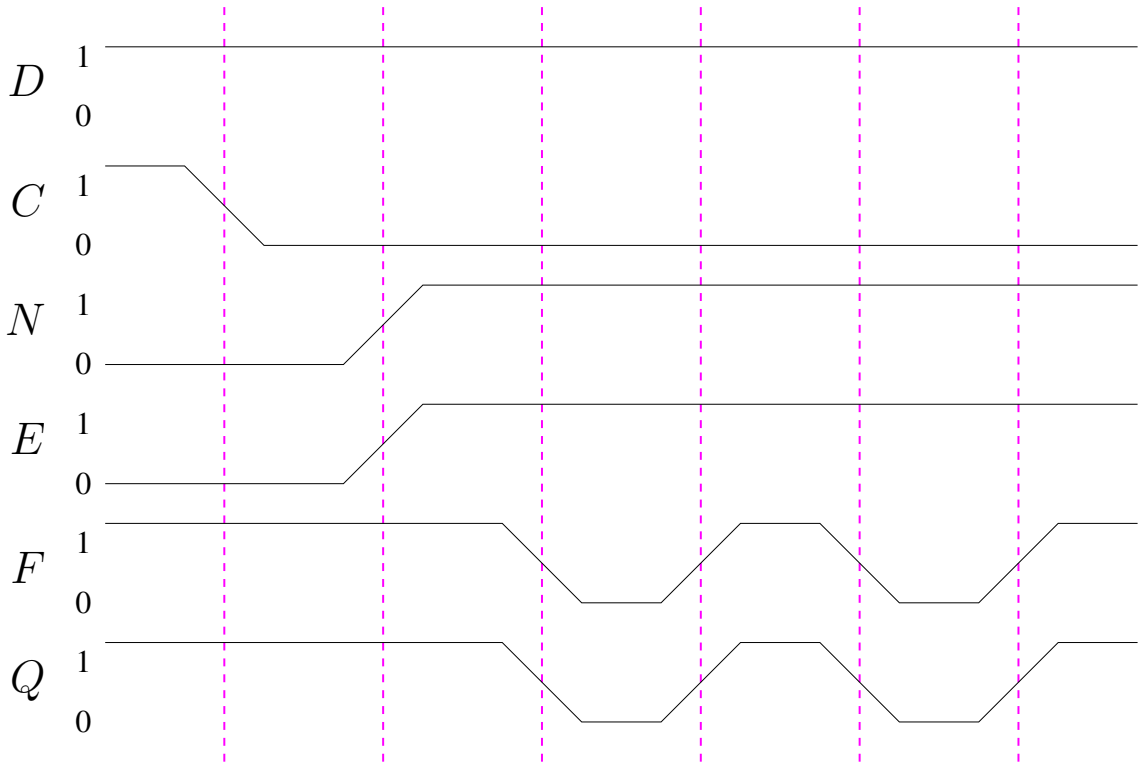
Hazard in selbst-gestricktem D-Latch



Hazard in selbst-gestricktem D-Latch



Hazard als Wave-Form



(idealisierte WF, da Delay von realistischen Gatter unterschiedlich)

Hazard in selbst-gestricktem D-Latch

- Achtung: Analyse zeigt nur, dass ein Problem bestehen *könnte*

- Genauere Analyse mit *SPICE* möglich (Transistor-Level Simulation)

Vermeidung von Hazards durch redundante Logik

- Hazard tritt beim Übergang von $C = 1$ nach $C = 0$ auf,
  wobei $Q = D = 1$

- man beachte: Q sollte sich eigentlich nicht ändern

  wenn immer $D = 1$ und $Q = 1$ ändert sich nichts, unabhängig von C

  (also bleibt Q auf 1 erhalten)

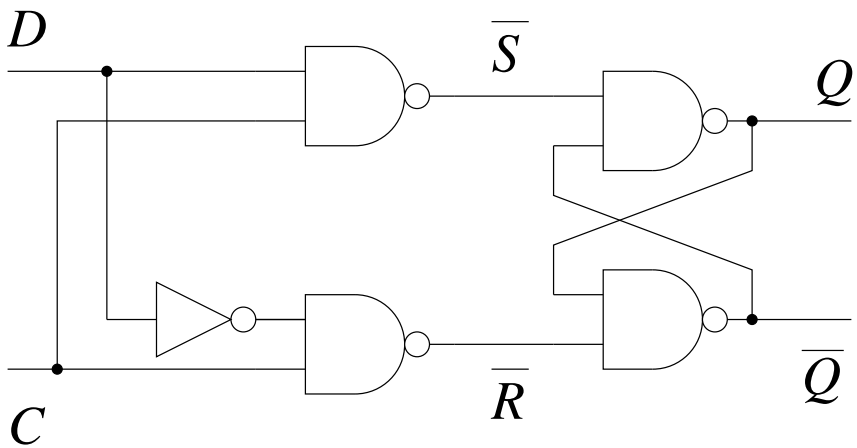
- Idee: füge die Bedingung $D \cdot Q$ als redundanten Min-Term hinzu
  (logische Äquivalenz ergibt sich aus Konsensus-Regel)

$$Q' \equiv \overline{\overline{CD} \cdot \overline{\overline{C}Q}} \equiv CD \vee \overline{C}Q \equiv CD \vee \overline{C}Q \vee DQ$$

$$\equiv CD \vee (\overline{C} \vee D) \cdot Q \equiv \overline{\overline{CD} \cdot \overline{\overline{C}\overline{D}Q}}$$

D-Latch ohne Hazard

Wir haben das normale D-Latch konstruiert:



D und C sollten sich nie gleichzeitig ändern!

Fundamental Mode

Annahme 1: Immer nur ein Eingang ändert sich

Annahme 2: Änderung eines Eingangs erst nach Stabilisierung

Diese Annahmen sind natürlich debattierbar.

Analyse im Fundamental Mode

- Idee: füge bei Feedback-Loops einen *virtuellen* Buffer ein

  weitere Annahme: gesamte Delays nur noch im Buffer

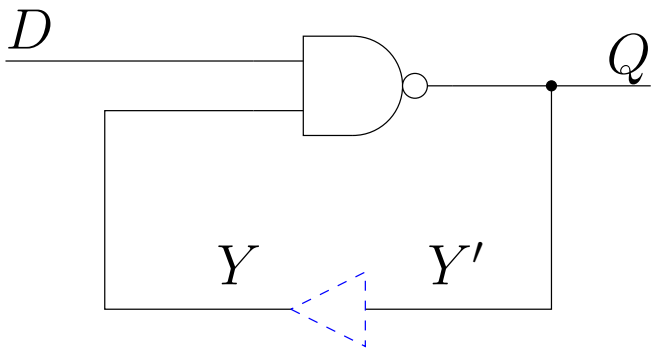
  andere Gates haben also Zero-Delay

  (nur realistisch bei Schaltkreisen ohne Hazard)

- Analysiere resultierende kombinatorische Übergangsfunktion:

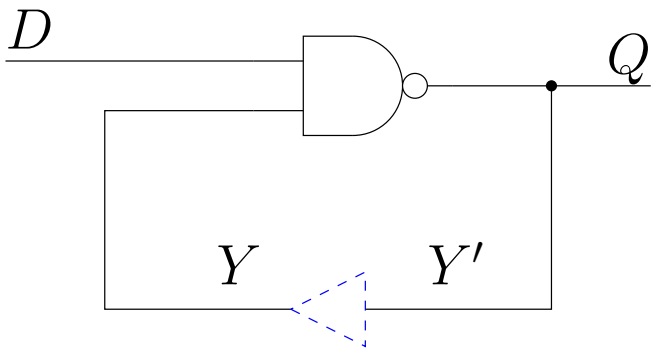
  Suche nach *metastabilen* Zuständen

Metastabile Zustände



$$Y' \equiv \overline{D \cdot Y}$$

Metastabile Zustände

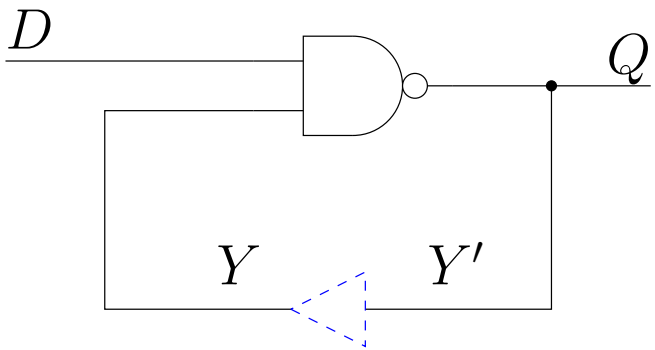


$$Y' \equiv \overline{D \cdot Y}$$

Zustands-Tabelle

Y \ D	0	1
0	1	1
1	**1**	0
	Y'	

Metastabile Zustände



$$Y' \equiv \overline{D \cdot Y}$$

Zustands-Tabelle

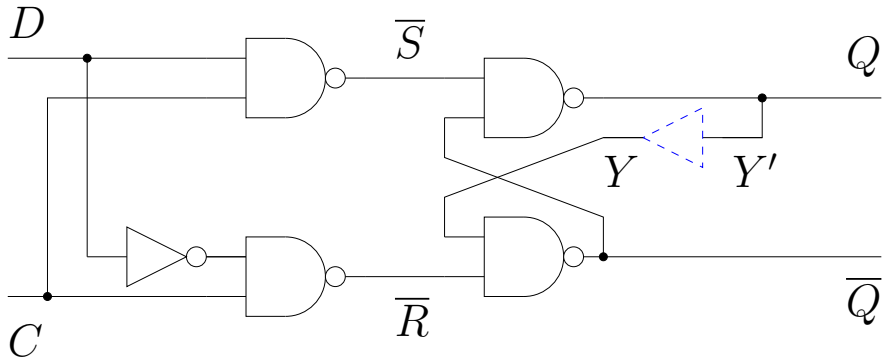
Y \ D	0	1
0	1	1
1	**1**	0

Y'

1. Bei $D = 0$ geht der Schaltkreis in einen stabilen Zustand ($Y' = Y$) über.

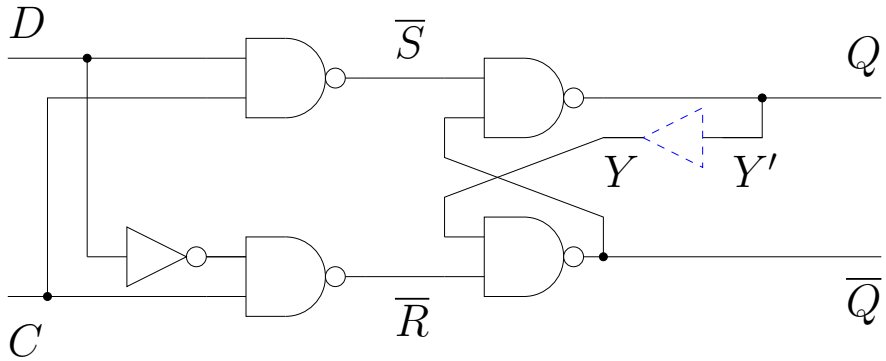
2. Bei $D = 1$ gibt es keinen stabilen Zustand, denn solange $D = 1$ ist, toggelt Y.

Formale(re) Analyse am Beispiel



$$
\begin{aligned}
Y' &\equiv D \cdot C \vee Y \cdot \overline{C} \vee D \cdot Y \\
Q &\equiv D \cdot C \vee Y \cdot \overline{C} \vee D \cdot Y \\
\overline{Q} &\equiv \overline{D} \cdot C \vee \overline{Y}
\end{aligned}
$$

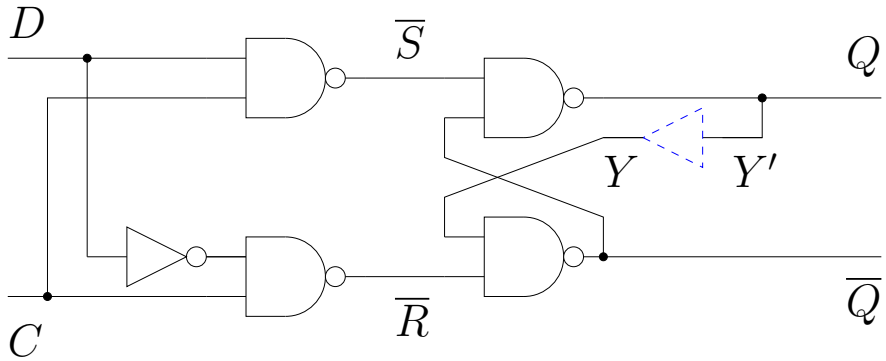
Formale(re) Analyse am Beispiel



$$
\begin{aligned}
Y' &\equiv D \cdot C \vee Y \cdot \overline{C} \vee D \cdot Y \\
Q &\equiv D \cdot C \vee Y \cdot \overline{C} \vee D \cdot Y \\
\overline{Q} &\equiv \overline{D} \cdot C \vee \overline{Y}
\end{aligned}
$$

Y	DC 00	01	11	10
0	**0** 1	**0** 1	1 1	**0** 1
1	**1** 0	0 1	**1** 0	**1** 0
		Y' $\overline{Q}$		

Formale(re) Analyse am Beispiel



$$
\begin{aligned}
Y' &\equiv D \cdot C \vee Y \cdot \overline{C} \vee D \cdot Y \\
Q &\equiv D \cdot C \vee Y \cdot \overline{C} \vee D \cdot Y \\
\overline{Q} &\equiv \overline{D} \cdot C \vee \overline{Y}
\end{aligned}
$$

Y \ DC	00	01	11	10
0	**0** 1	**0** 1	1 1	**0** 1
1	**1** 0	0 1	**1** 0	**1** 0
		Y' $\overline{Q}$		

Die Zustände **0** und **1** sind die stabilen Zustände.

Konstruktion von asynchronen Schaltkreisen

- Spezifikation spricht üblicherweise auch über Flanken
- Änderungsreihenfolge der Zustandsvariablen muss vernachlässigbar sein

  ✘ ansonsten gibt es *races*
- dann muss man Hazards vermeiden (durch Redundanz)

Field-Programmable Gate Arrays (FPGAs)

- Erinnerung: hohe Stückzahl $\longrightarrow$ niedriger Preis

Field-Programmable Gate Arrays (FPGAs)

- Erinnerung: hohe Stückzahl $\longrightarrow$ niedriger Preis
- Ziel: Funktionalität wie herkömmlich hergestellter Chip, aber beliebig oft programmierbar

Field-Programmable Gate Arrays (FPGAs)

- Erinnerung: hohe Stückzahl $\longrightarrow$ niedriger Preis
- Ziel: Funktionalität wie herkömmlich hergestellter Chip, aber beliebig oft programmierbar
- Grundlegende Idee: ein 1-bit RAM mit a Adressbits kann jede Boolesche Funktion über a Argumente implementieren
- RAM (und damit Funktion) kann beliebig oft geändert werden
- Problem: bereits Funktion mit nur 32 Argumenten braucht $2^{32}/8$ Bytes = 500 MB RAM
- Fehlt noch: Latch/Flipflop

Aufteilung in Zellen

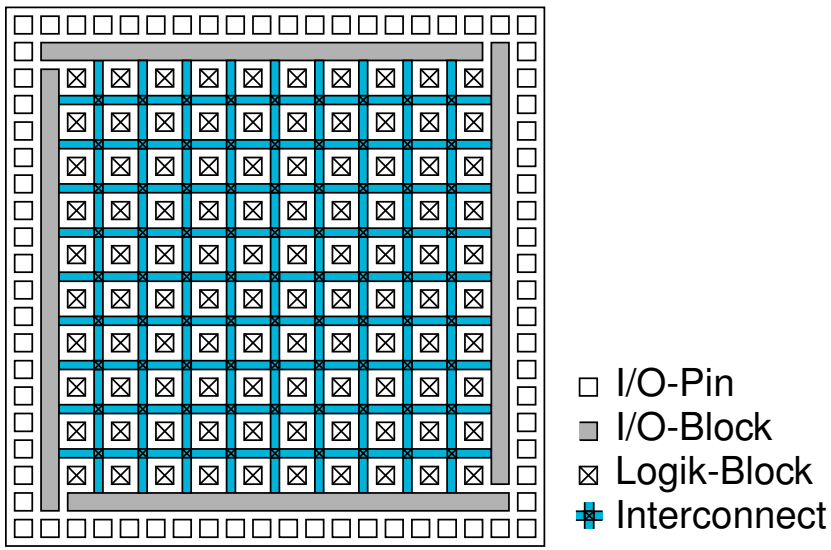
- Abhilfe: Schaltung aus Zellen aufbauen

✔ Zellen implementieren beliebige Funktion

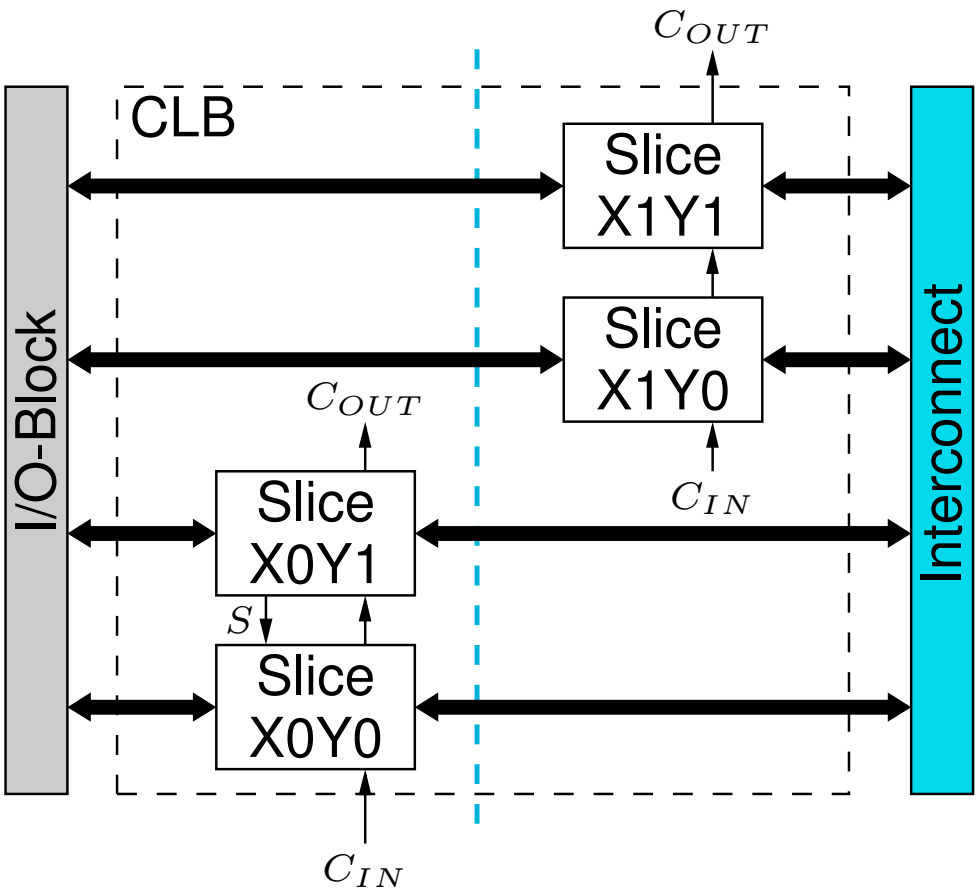
✘ Lassen sich aber nicht beliebig kombinieren

- Kombination aus RAM (look-up table) und 2 Registern: Configurable Logic Block (CLB)
- Verbindung zwischen Zellen: Interconnect
- Noch dazu:
  - RAM
  - Multiplizierer

Aufteilung in Zellen

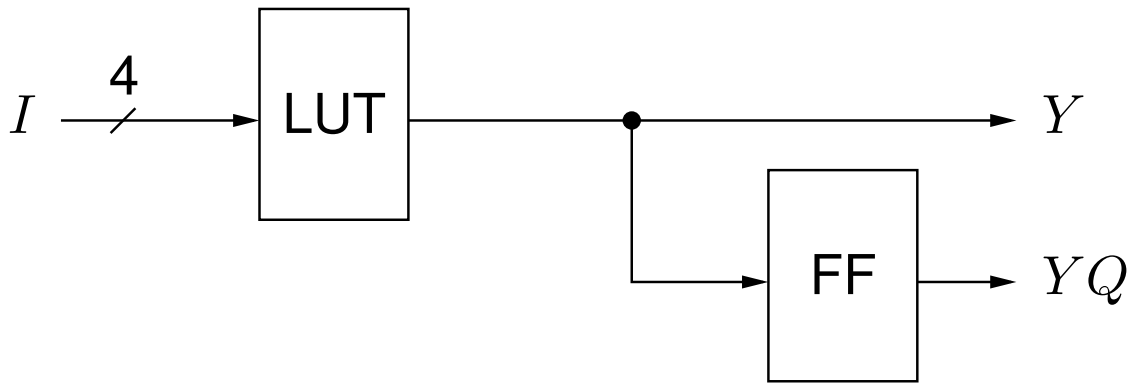


Überblick FPGA

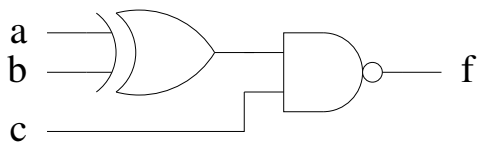


XC3S200: 24x20=480 CLBs (ca. 200.000 Gatter)

Slice



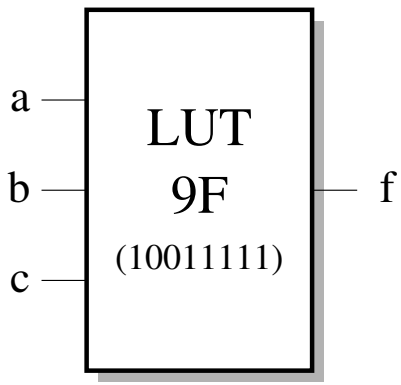
- Konfiguration als kombinatorische Logik oder FF
- Konfiguration wird ebenfalls über RAM gesteuert



$\Longrightarrow$

a	b	c	$a \oplus b$	f
0	0	0	0	1
1	0	0	1	1
0	1	0	1	1
1	1	0	0	1
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	0	1

$\Longrightarrow$



Belegung des RAMs im LUT kann einfach
aus der Funktionstabelle abgelesen werden!

LUT Mini-Quiz

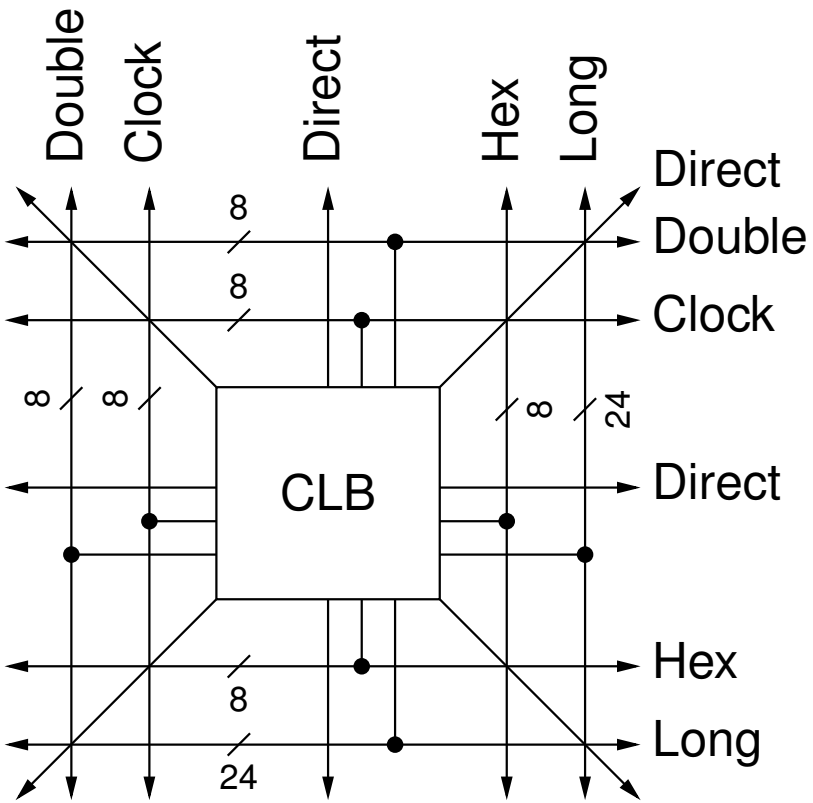
Gegeben seien die Funktionen

$$
\begin{aligned}
o_1 &= (b \wedge d) \vee (\neg c \wedge d) \vee (a \wedge d) \\
o_2 &= (\neg c \wedge e) \vee (a \wedge e) \vee (b \wedge e) \vee \neg d \vee f
\end{aligned}
$$

wobei alle Variablen ein Bit breit sind.

Die Funktionen sollen nun mit der Hilfe von 4-Eingang Lookup Tables (LUT) implementiert werden.

(Klausur Herbst 2008)


Double
Clock
Direct
Hex
Long
Direct
Double
Clock
Direct
Hex
Long
8
8
8
8
8
24
8
24
CLB

RAM auf dem FPGA

- Nicht verwechseln mit RAM auf dem Board!

- Falls RAM aus CLBs nicht reicht oder zu langsam ist

- XC3S200: 12 RAMs, insgesamt 221184 Bits
- Mit zwei Lese/Schreibports
- Breite ist konfigurierbar
- Kann initialisiert werden (z.B. mit einem Programm)

Multiplizierer

- ✔ Schneller als Multiplizierer aus CLBs

- Eingabe: zwei 18-Bit Zahlen im Zweierkomplement
- Ausgabe: 36-Bit Ergebnis

- XC3S200: 12 Stück