

Design digitaler Schaltkreise

- Komplexere CMOS Digitalzellen
- NAND, NOR...
- Multiplexer, Dekoder...
- Latches, Flip-Flops
- Kombinatorische und Sequenzielle Logik
- Kombinatorische - Schaltfunktionen (logische Gatter)
- Sequenzielle Logik – Speicherzellen (Flip-Flips bzw. Register)

- Kombinatorisch gibt es $2^4 = 16$ Booleschen Funktionen von zwei Variablen
- Die Länge der Ergebnistabelle ist 4 und für jede Zeile haben wir zwei Möglichkeiten.
- Die wichtigsten Booleschen Funktionen mit zwei Variablen sind NAND, NOR, EXNOR (Äquivalenz).
- Da es Inverter gibt, können wir aus NAND, NOR und EXNOR AND, OR und die EXOR bauen

NAND

a	b	y
0	0	1
0	1	1
1	0	1
1	1	0

NOR

a	b	y
0	0	1
0	1	0
1	0	0
1	1	0

EXNOR

a	b	y
0	0	1
0	1	0
1	0	0
1	1	1

- Warum sind nur drei (bzw. sechs) Funktionen genug?
- 8 Booleschen Funktionen kann man aus 16 durch Negation bekommen - wie AND aus NAND.
- Zwei Funktionen (von 8 – wir betrachten z.B. nur diese die mit 1 anfangen) sind eigentlich keine Funktionen von zwei sondern nur einer Variable.
- Eine Funktion von 8 ist Konstante.

!a

a	b	y
0	0	1
0	1	1
1	0	0
1	1	0

!b

a	b	y
0	0	1
0	1	0
1	0	0
1	1	0

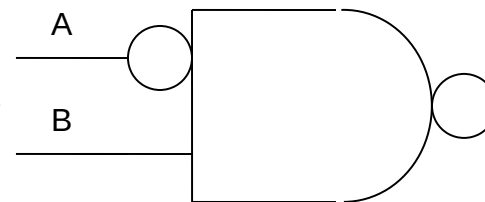
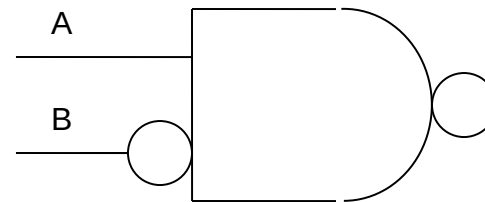
1

a	b	y
0	0	1
0	1	1
1	0	1
1	1	1

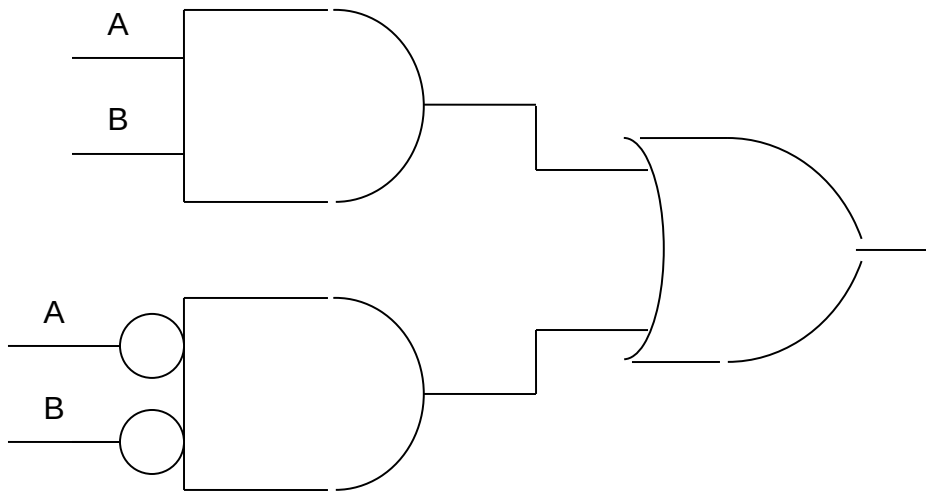
- Zwei Funktionen können mit NAND mit Invertierten Eingängen realisiert werden.
- -> NAND, NOR, EXNOR und Inverter sind ausreichend
- EXNOR kann man mit (N)AND, (N)OR und Inverter realisieren
- NOR kann man in NAND umzuwandeln.
- Streng genommen wäre z.B. NAND genug.

a	b	y
0	0	1
0	1	1
1	0	0
1	1	1

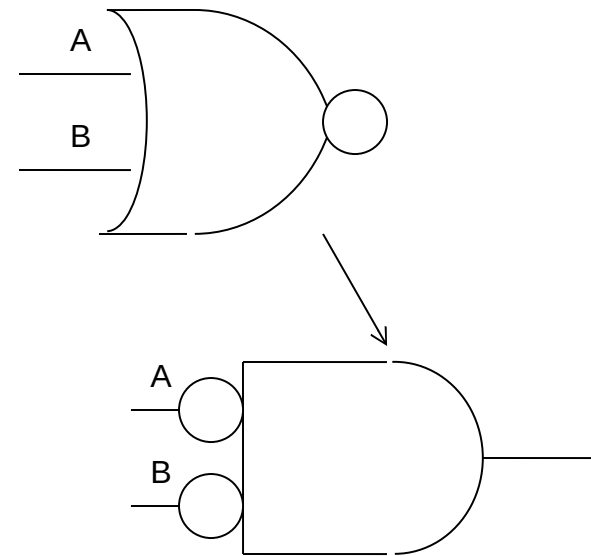
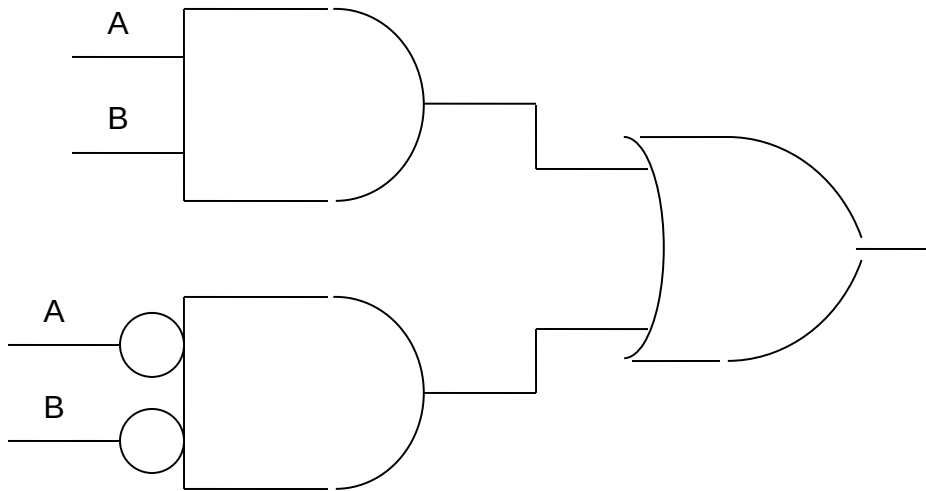
a	b	y
0	0	1
0	1	0
1	0	1
1	1	1



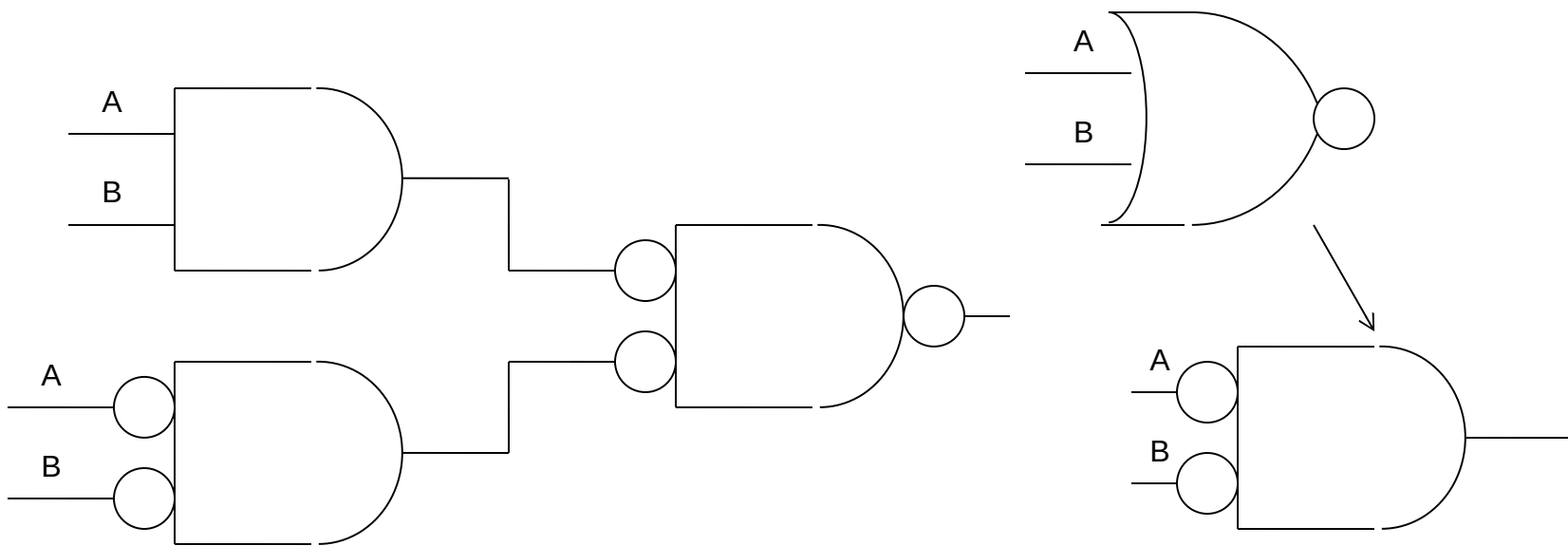
- Zwei Funktionen können mit NAND mit Invertierten Eingängen realisiert werden.
- -> NAND, NOR, EXNOR und Inverter sind ausreichend
- **EXNOR kann man mit (N)AND, (N)OR und Inverter realisieren**
- NOR kann man in NAND umzuwandeln.
- Streng genommen wäre z.B. NAND genug.



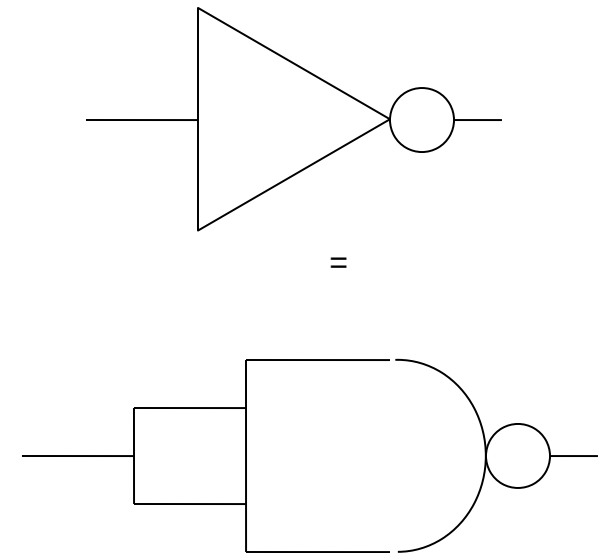
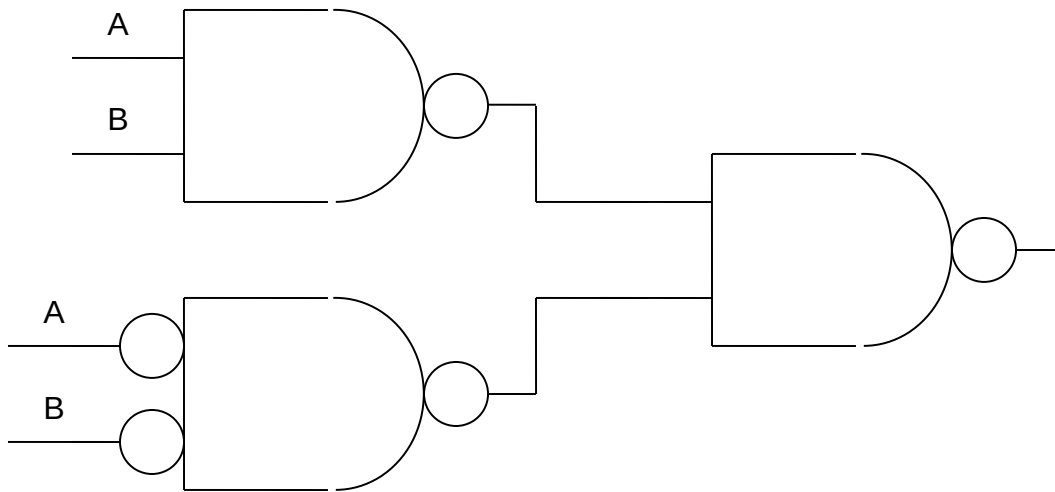
- Zwei Funktionen können mit NAND mit Invertierten Eingängen realisiert werden.
- -> NAND, NOR, EXNOR und Inverter sind ausreichend
- EXNOR kann man mit (N)AND, (N)OR und Inverter realisieren
- **NOR kann man in NAND umzuwandeln.**
- Streng genommen wäre z.B. NAND genug.



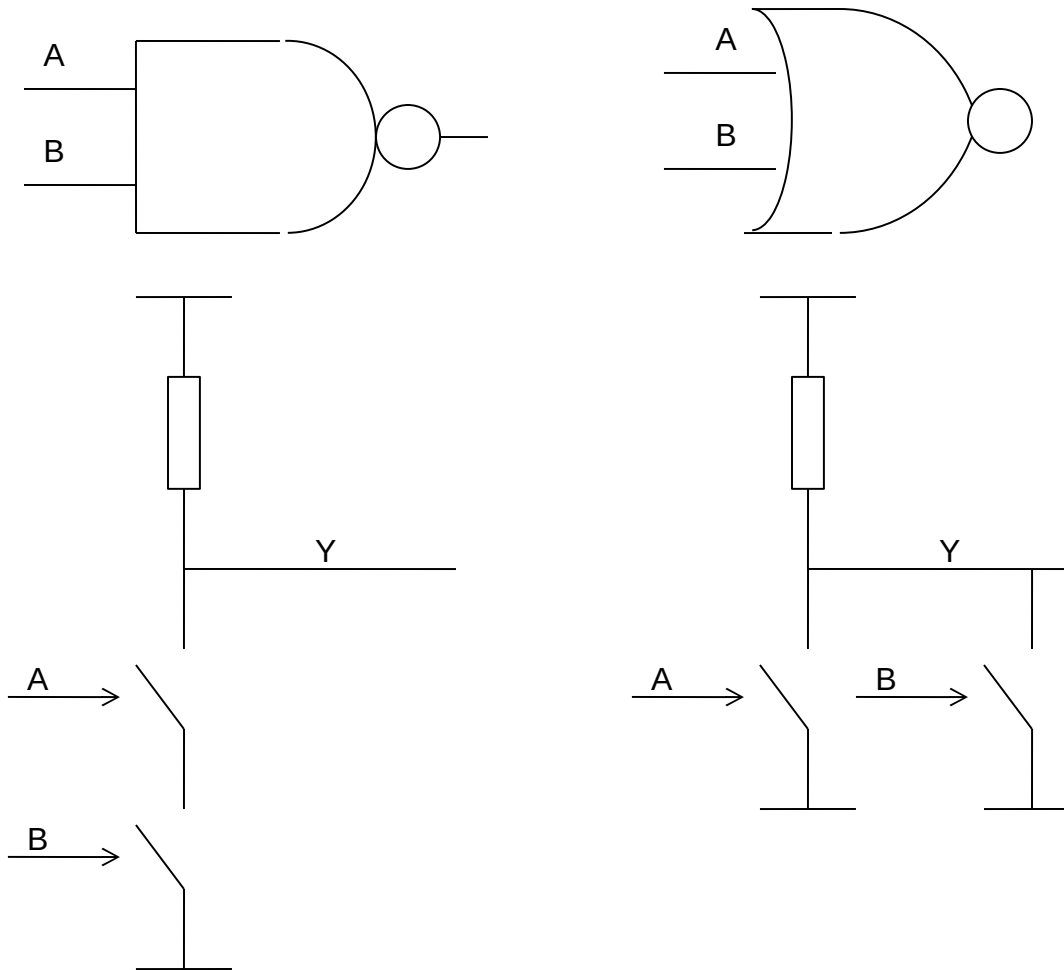
- Zwei Funktionen können mit NAND mit Invertierten Eingängen realisiert werden.
- -> NAND, NOR, EXNOR und Inverter sind ausreichend
- EXNOR kann man mit (N)AND, (N)OR und Inverter realisieren
- NOR kann man in NAND umzuwandeln.
- **Streng genommen wäre z.B. NAND genug.**



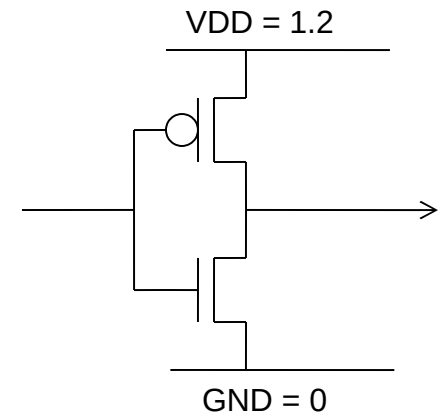
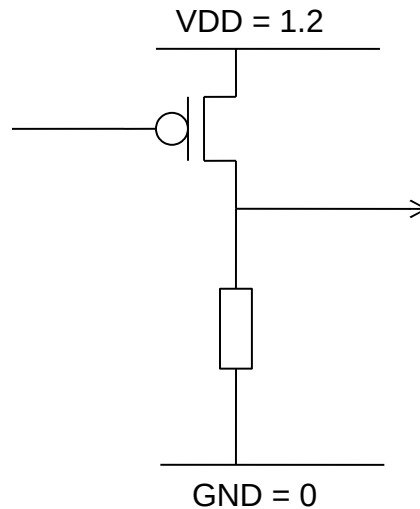
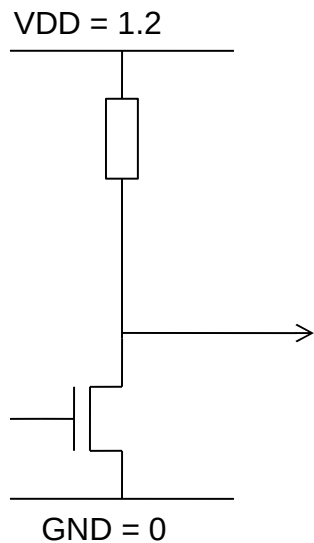
- Zwei Funktionen können mit NAND mit Invertierten Eingängen realisiert werden.
- -> NAND, NOR, EXNOR und Inverter sind ausreichend
- EXNOR kann man mit (N)AND, (N)OR und Inverter realisieren
- NOR kann man in NAND umzuwandeln.
- **Streng genommen wäre z.B. NAND genug.**



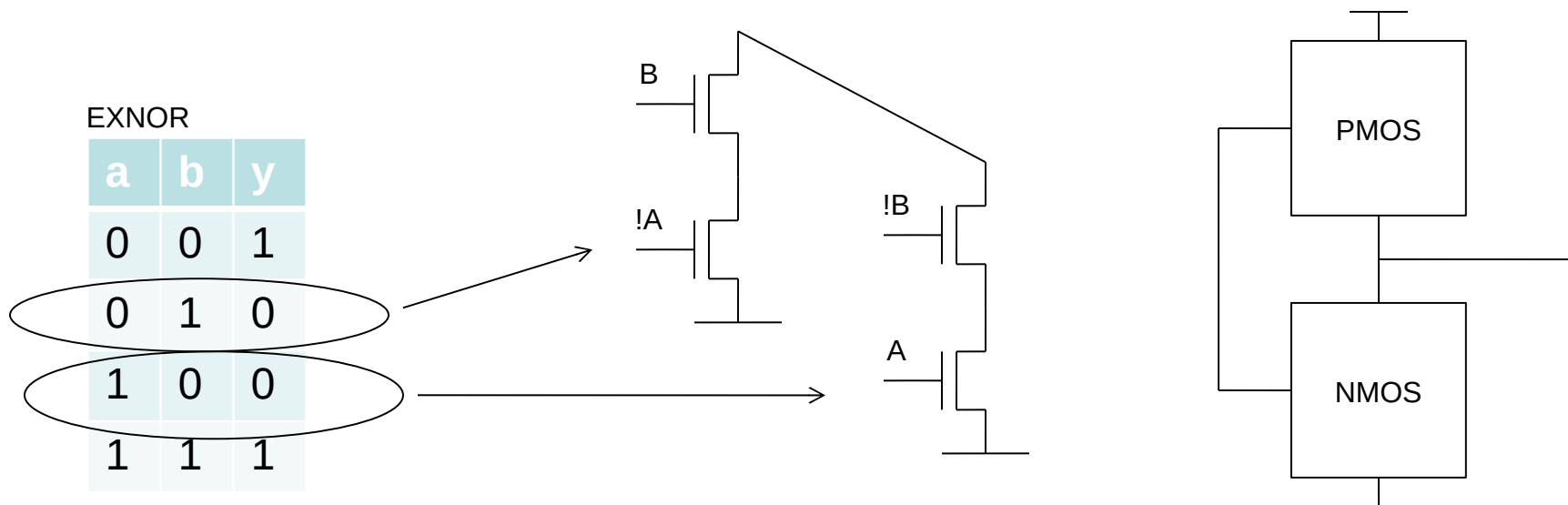
- NAND und NOR als Schalter-Widerstand Logik



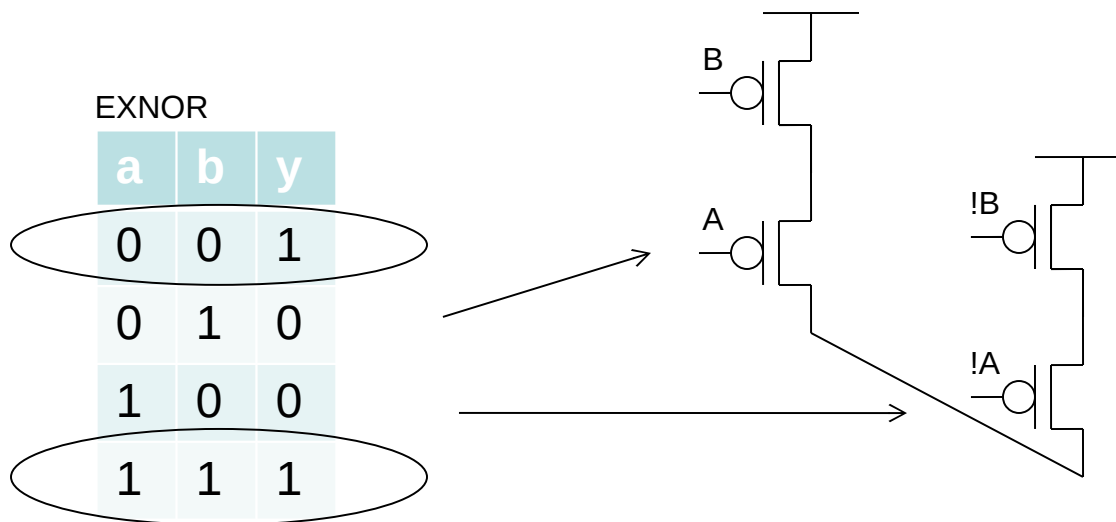
- Inverter als RTL Logik
- NMOS und Pullup oder mit PMOS und Pulldown Widerstand
- -> CMOS Inverter.
- Vorteile sind kein DC Strom und ein kleines Layout.



- NAND, NOR und co. als CMOS
- Wie wird ein CMOS Gate gemacht?
- Wenn NMOS Teil leitet, soll PMOS Teil nicht leiten, und umgekehrt
- Kein Kurzschluss VDD-GND, oder floating-Ausgang
- Jede Zeile mit dem Ergebnis 0 -> Serienschaltung von zwei (oder mehreren) NMOS Transistoren die nur für die Eingangswerte dieser Zeile leiten
- Man muss alle Eingänge = null invertieren.
- Das ganze NMOS Netzwerk ist die Parallelschaltung aller Reihenschaltungen, die Zeilen = 0 entsprechen.



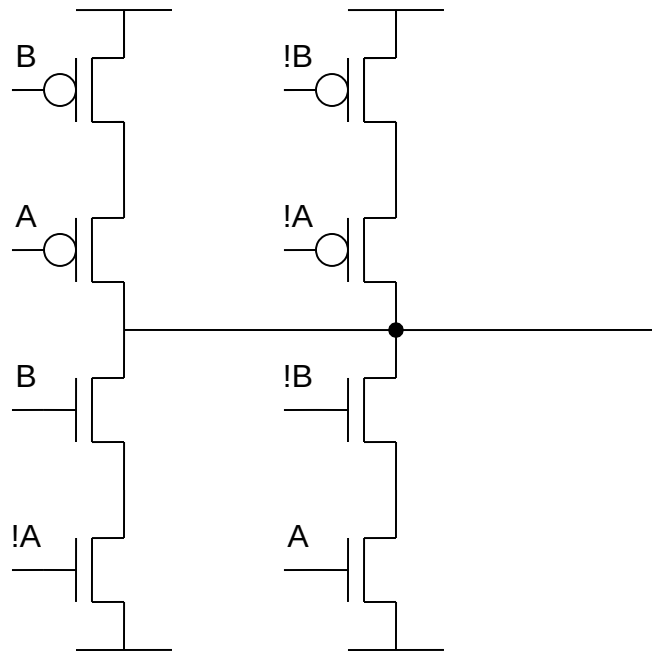
- PMOS Teil macht man dual
- Beachten wir, dass PMOS für niedriges Gate-Potential leitet
- Man muss alle Eingänge = eins invertieren.



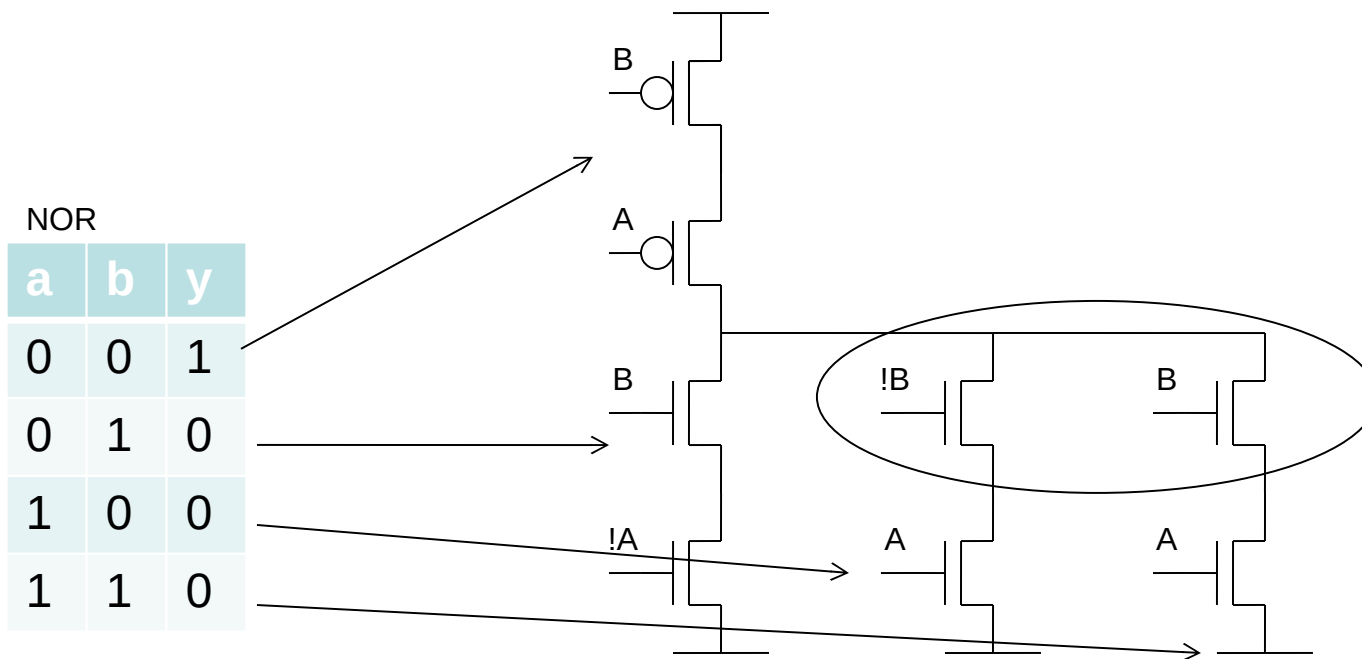
- EXNOR

EXNOR

a	b	y
0	0	1
0	1	0
1	0	0
1	1	1



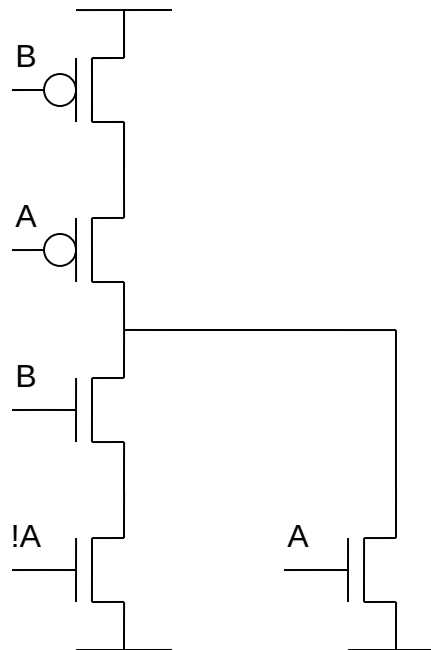
- Oft kann man die logische Funktion vereinfachen



- Oft kann man die logische Funktion vereinfachen

NOR

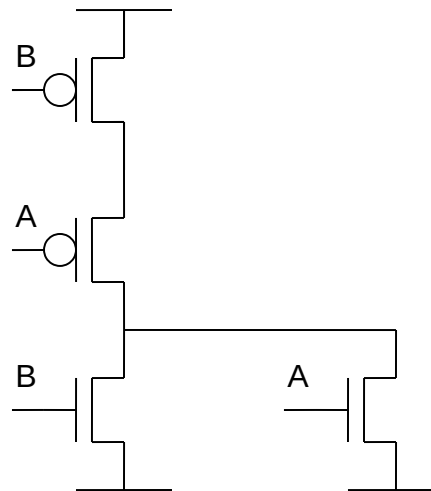
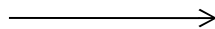
a	b	y
0	0	1
0	1	0
1	0	0
1	1	0



- Oft kann man die logische Funktion vereinfachen
- CMOS NOR
- PMOS Netzwerk leitet für die Eingangskombination 00 – Reihenschaltung
- NMOS Netzwerk leitet immer außer für 00 – Parallelschaltung.

NOR

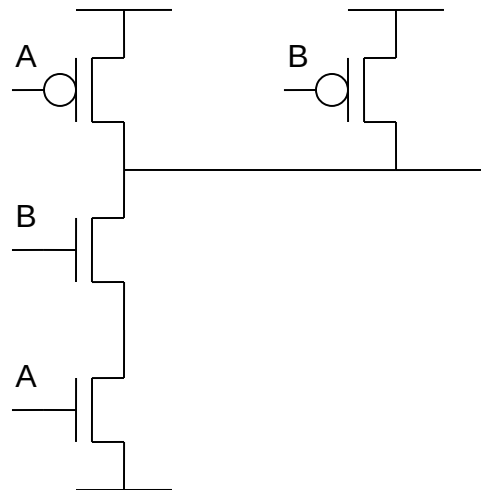
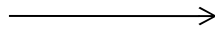
a	b	y
0	0	1
0	1	0
1	0	0
1	1	0



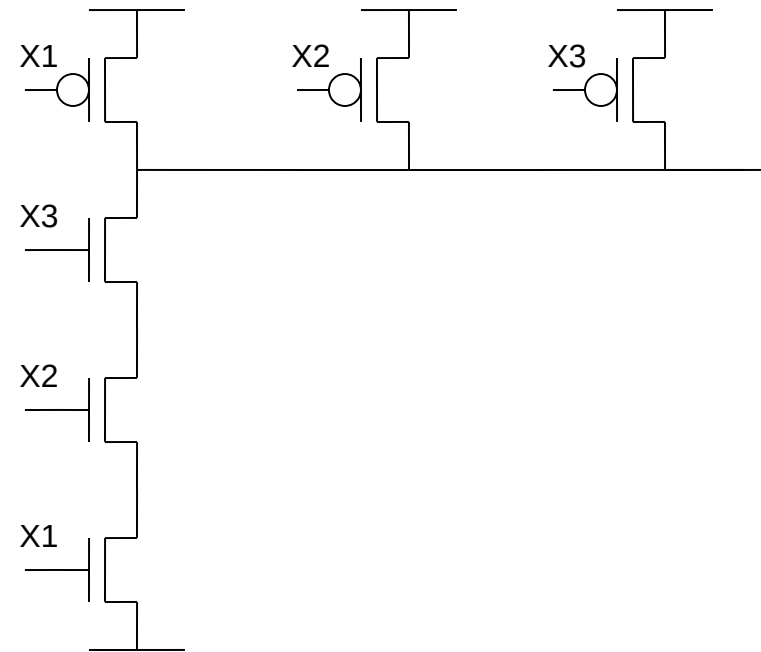
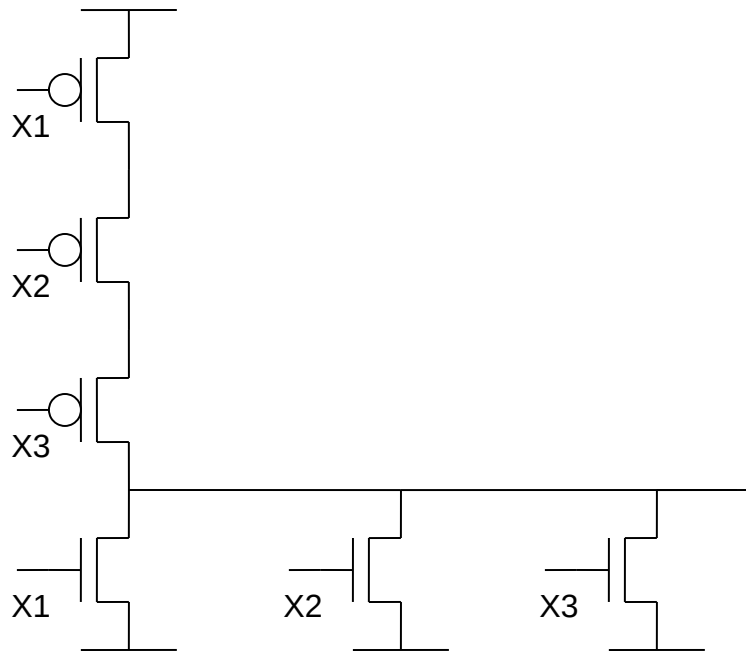
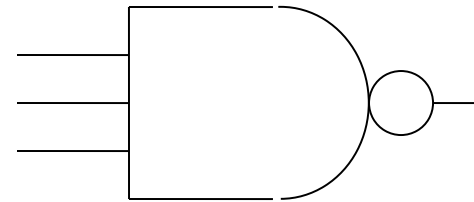
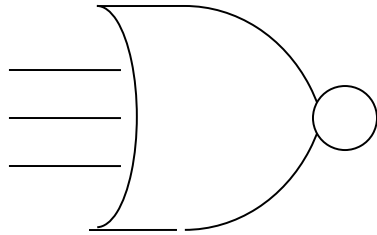
- CMOS NAND

NAND

a	b	y
0	0	1
0	1	1
1	0	1
1	1	0



- Es ist leicht die NAND und NOR auf mehr als 2 Eingänge zu erweitern

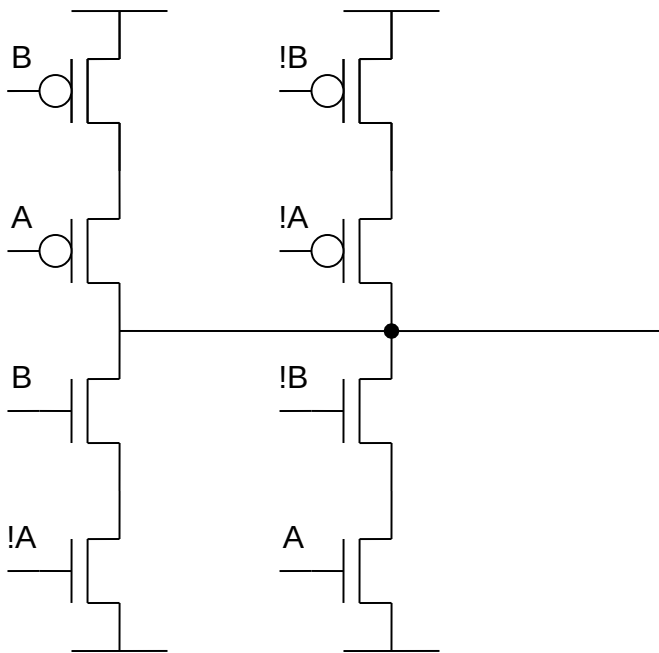


- Einige Regeln
- NMOS Teil leitet für die Zeilen mit null-Ergebnis
- PMOS Teil leitet für die Zeilen mit eins-Ergebnis
- PMOS und NMOS Teile dürfen nie gleichzeitig leiten
- Sonst hätten wir einen großen Querstrom und der Ausgang wäre undefiniert
- PMOS und NMOS Teil sollen auch nie gleichzeitig offene Verbindungen sein. In dem Fall wäre der Ausgang von den Versorgungslinien getrennt. Der logische Wert wäre undefiniert
- Gate mit offenem Ausgang befindet sich im Hochohmigen Zustand

- EXNOR

Variante 1

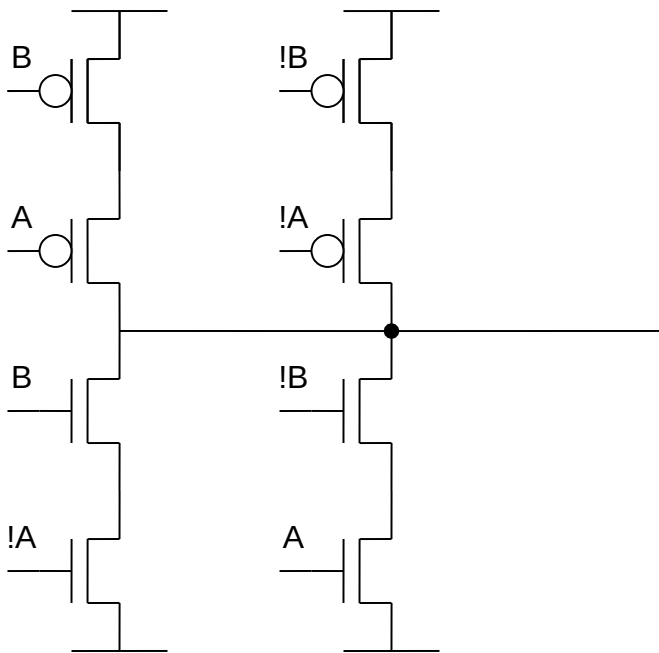
8 Transistoren



- EXNOR
- Disjunktive Normalform
- $\text{EXOR} = (!A \& !B) \mid (A \& B)$

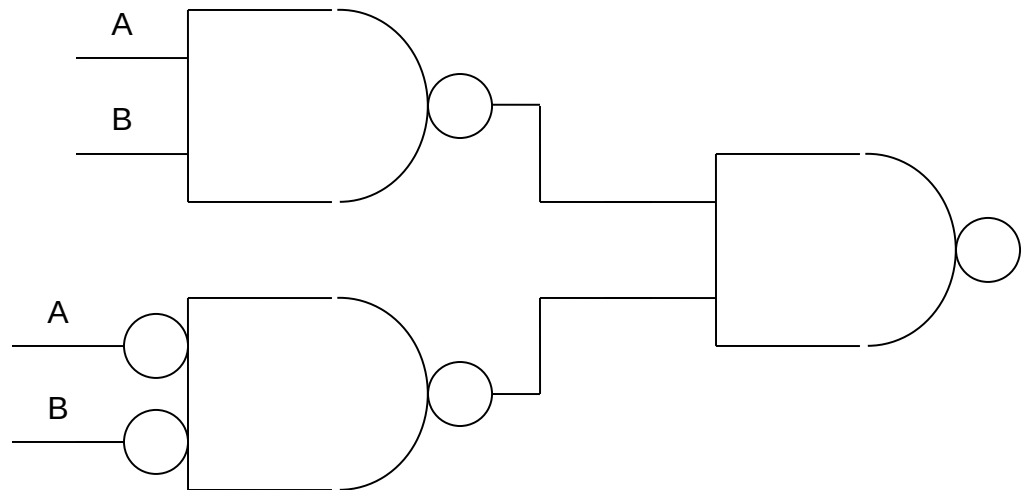
Variante 1

8 Transistoren

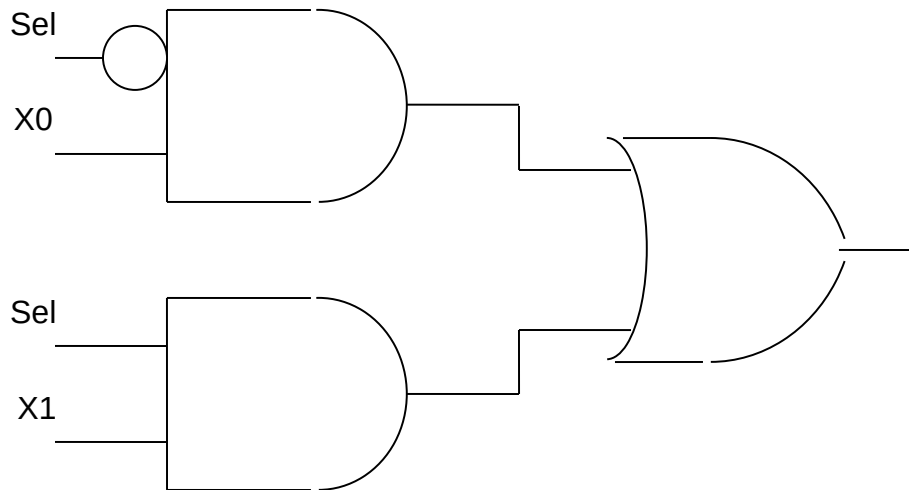


Variante 2

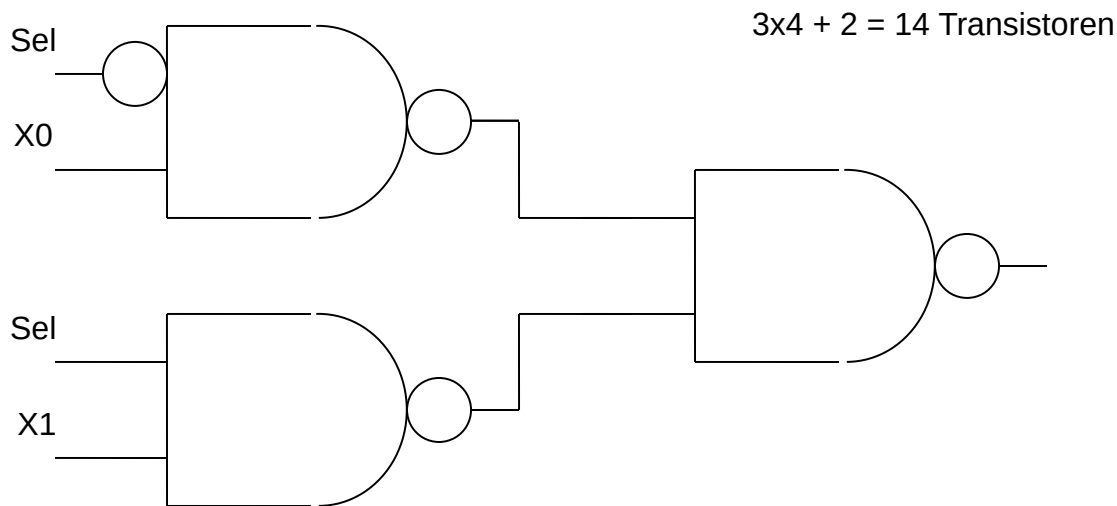
4 (INVs) + 12 (NANDs) Transistoren



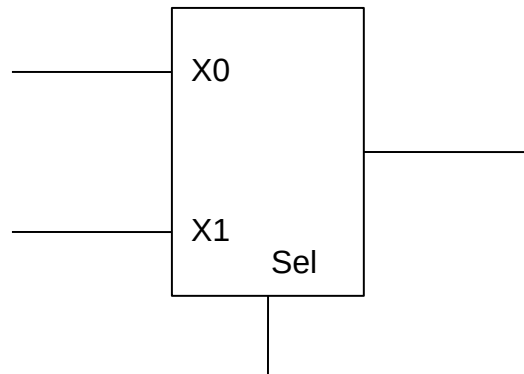
- Noch komplexere Gates
- Gatter mit 3 Eingängen
- Der wichtigste und vielseitigste Bauteil in Digitaltechnik ist der Multiplexer
- Je nachdem ob der Select-Eingang null oder eins ist, ist der Ausgang gleich X0 oder X1
- Im Verilog Code: $Y = \text{sel} ? X1 : X0$
- Disjunktive Normalform: $Y = !\text{sel} \& X0 \mid \text{sel} \& X1$



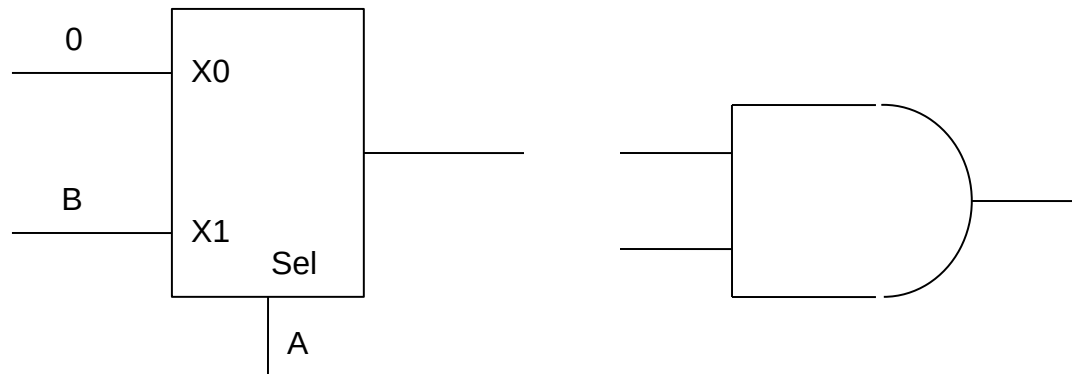
- Noch komplexere Gates
- Gatter mit 3 Eingängen
- Der wichtigste und vielseitigste Bauteil in Digitaltechnik ist der Multiplexer
- Je nachdem ob der Select-Eingang null oder eins ist, ist der Ausgang gleich X0 oder X1
- Im Verilog Code: $Y = \text{sel} ? X1 : X0$
- Disjunktive Normalform: $Y = !\text{sel} \& X0 \mid \text{sel} \& X1$



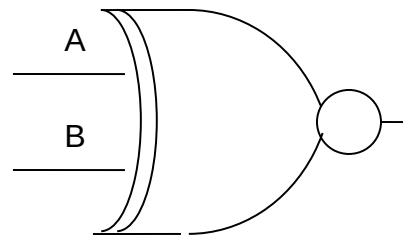
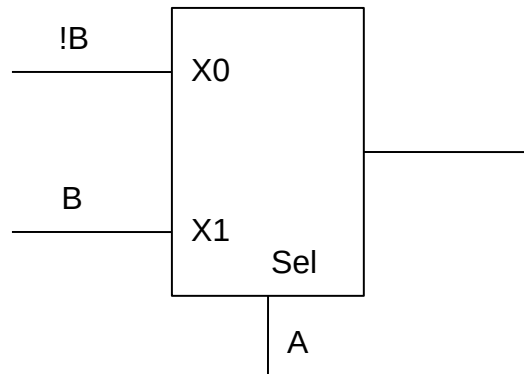
- Warum ist ein Multiplexer so wichtig?
- Jede logische Funktion kann mit Multiplexern, Invertern und logischen Konstanten realisiert werden



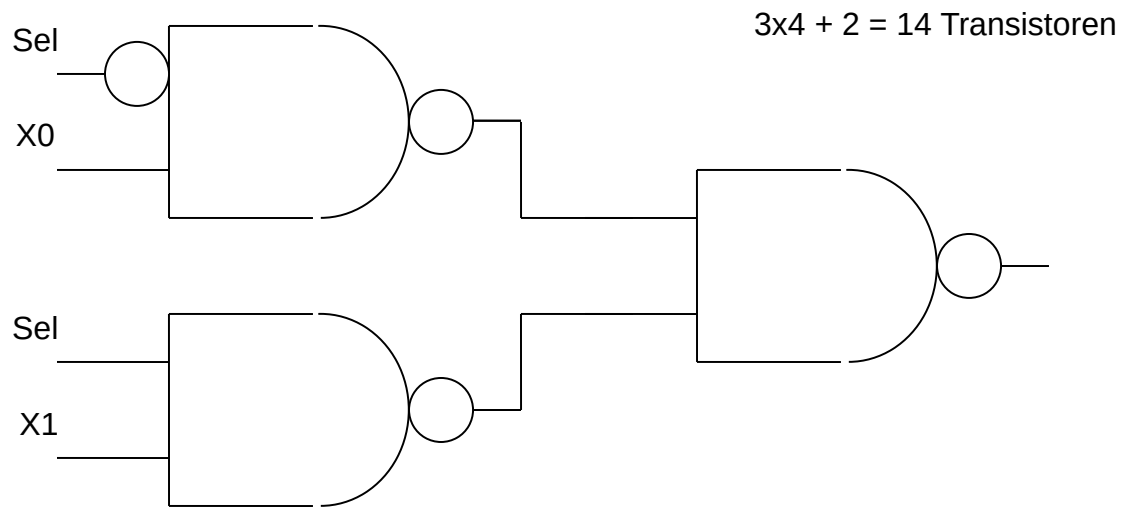
- Beispiel: AND
- AND ist null wenn die Variable A null ist, unabhängig von B
- -> A an Select anschließen, an Eingang X0 schließen wir die logische 0
- Wenn A eins ist (Select = 1), hängt das Ergebnis von Variable B ab
- -> Variable B wird an Eingang X1 angeschlossen.
- $AND = A ? B : 0$



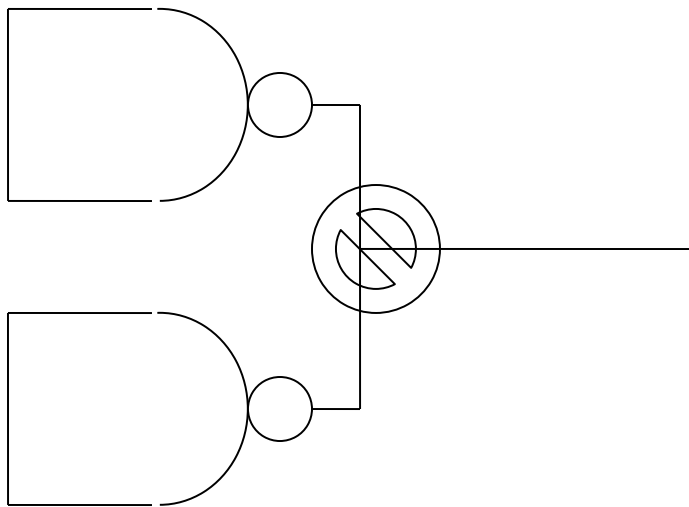
- Beispiel EXNOR
- $\text{EXNOR} = A ? B : !B$



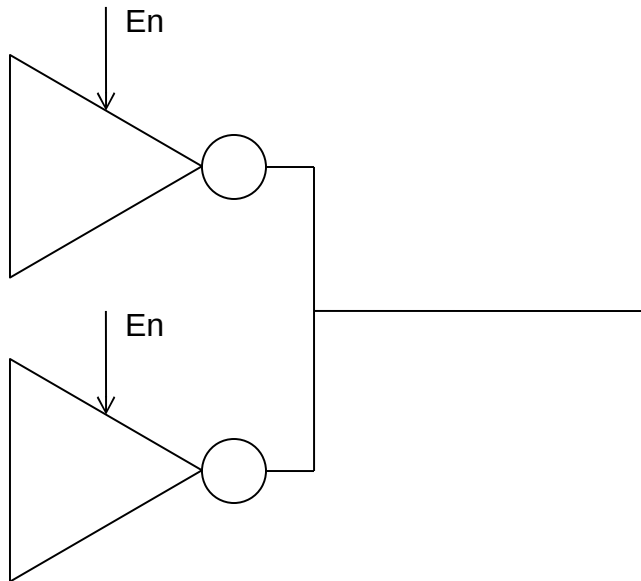
- Multiplexer kann auch einfacher realisiert werden.



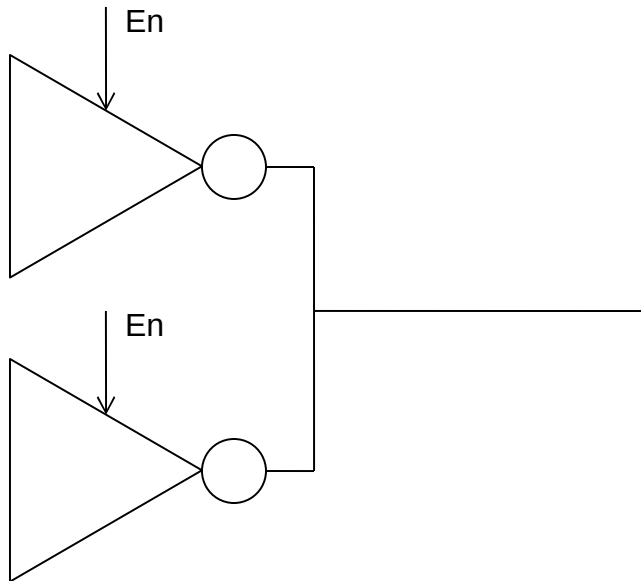
- Multiplexer kann auch einfacher realisiert werden.



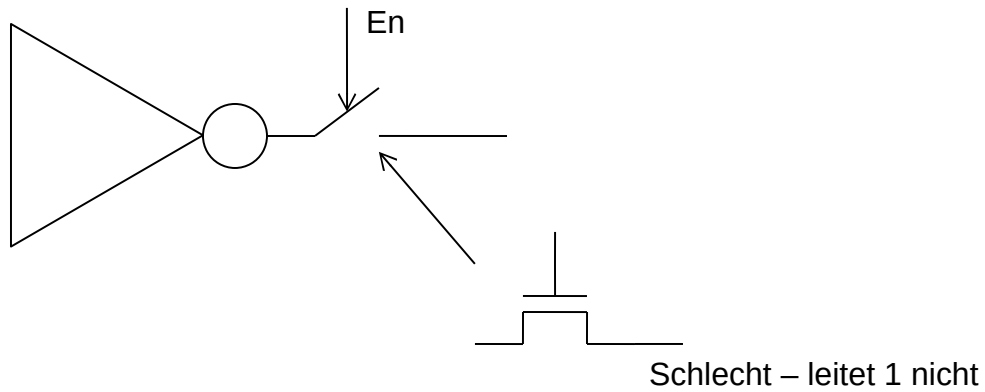
- Wir können die Gatter so erweitern, dass sie sich im hochohmigen Zustand befinden können
- -> Gated Inverter
- Wenn der Enable Eingang eins ist, funktioniert der Inverter wie ein gewöhnlicher Inverter



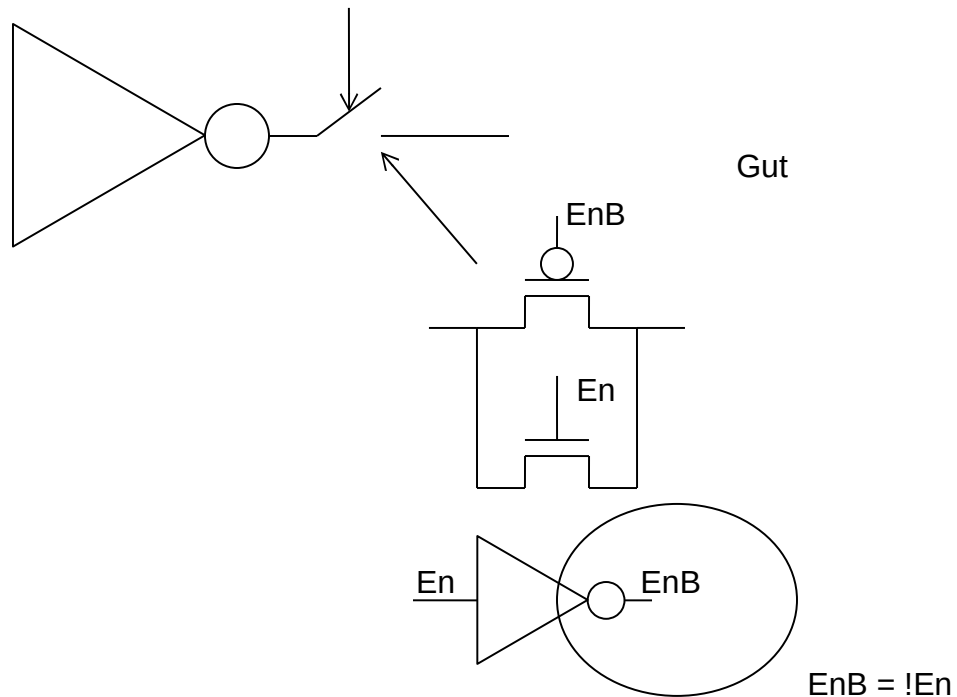
- Mit Enable = null, ist der Ausgang von VDD und GND getrennt, der Ausgang „schwebt“ (float) im hochohmigen (high impedance) Zustand



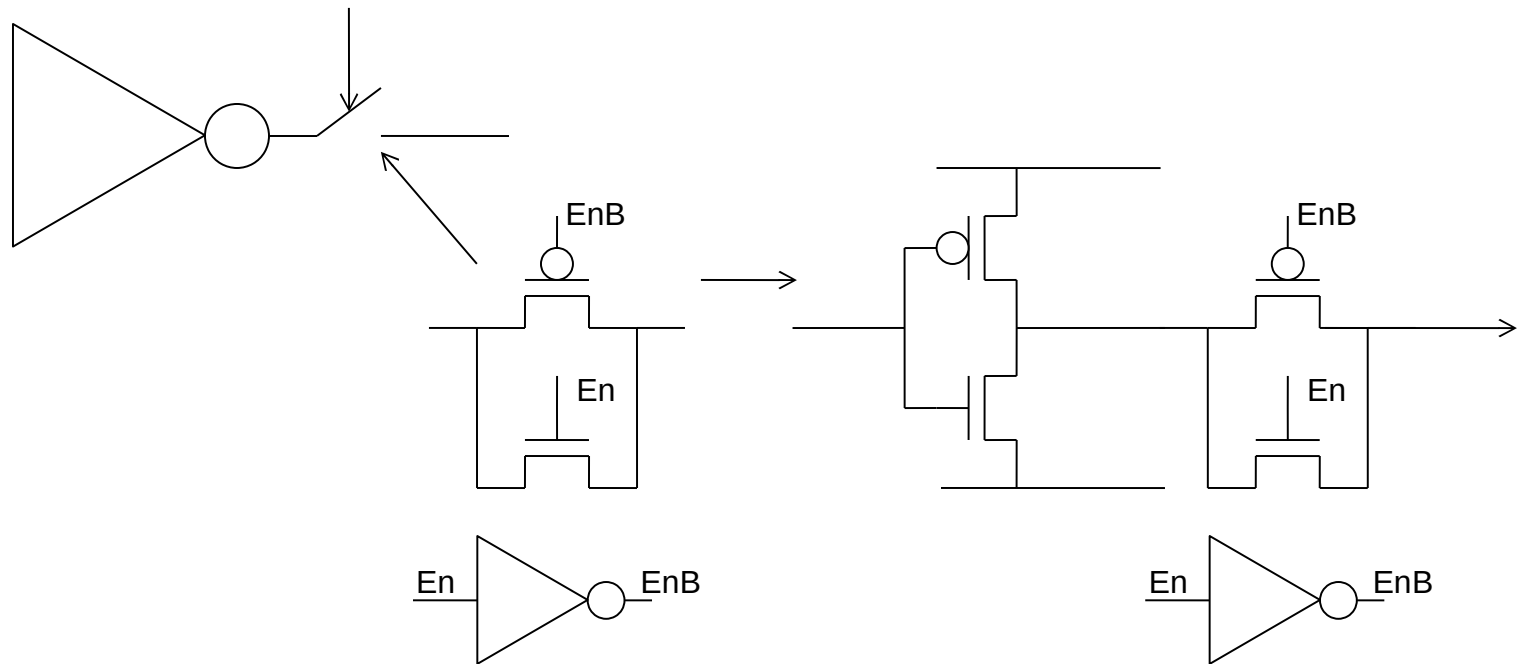
- Schaltung



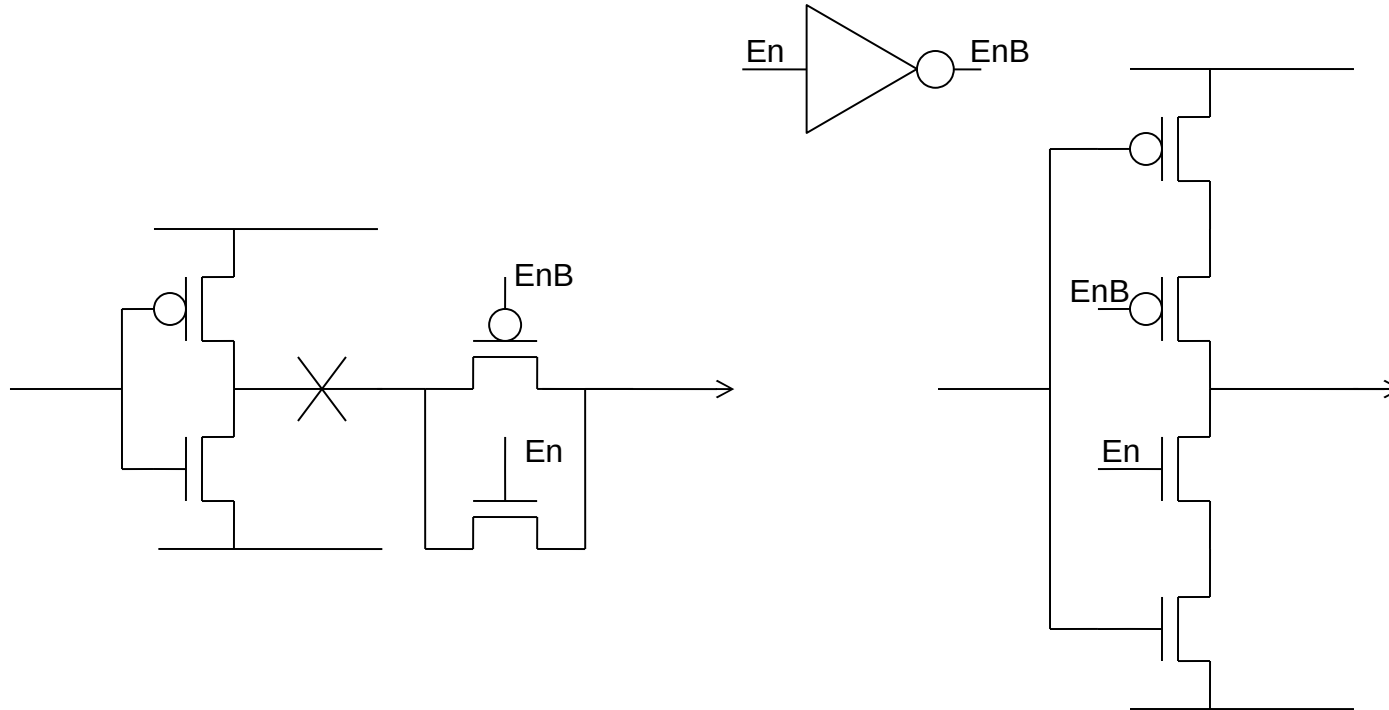
- Schaltung



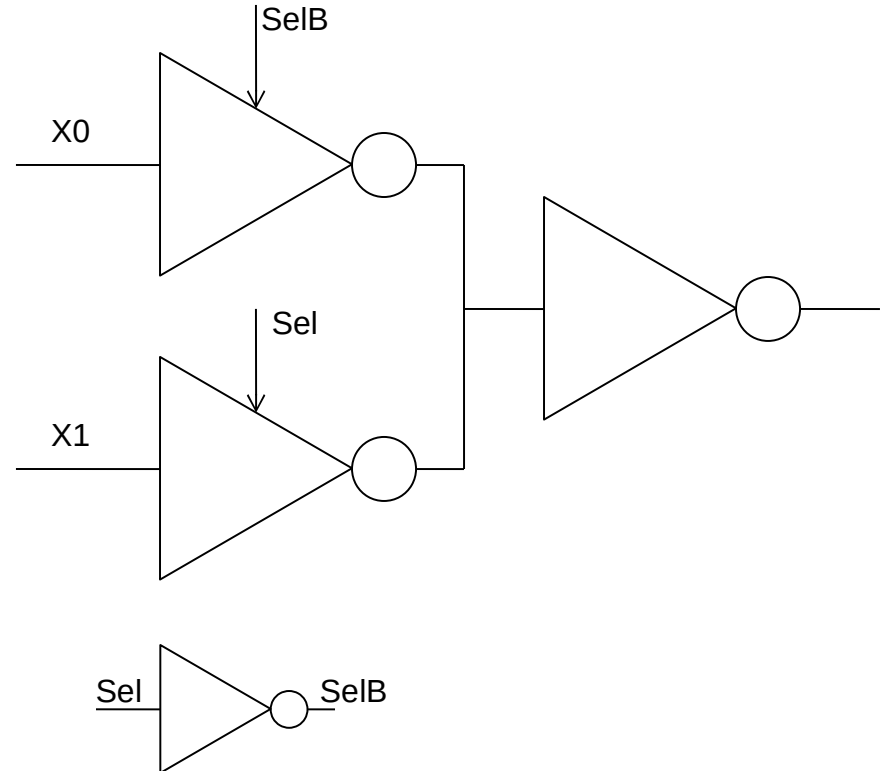
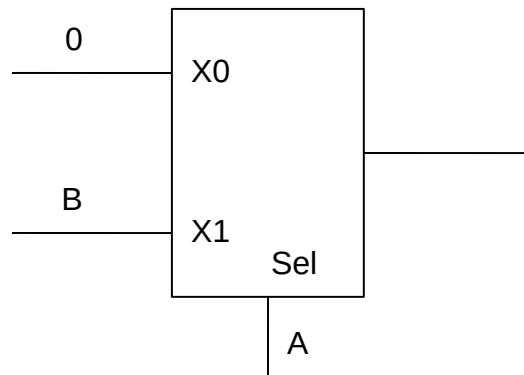
- Schaltung



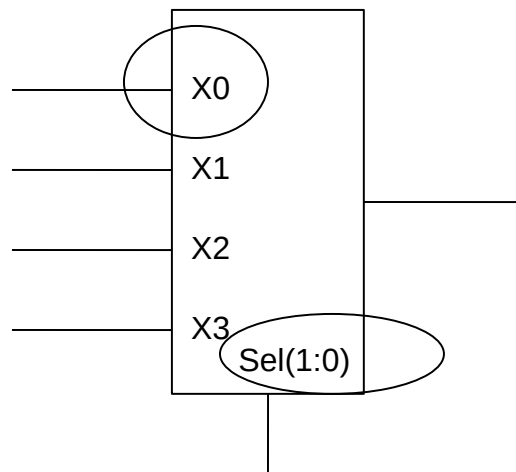
- Schaltung



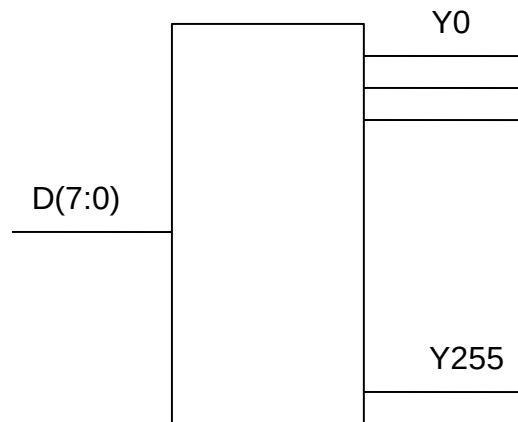
- Multiplexer mit Gated Invertern
- Wir brauchen zwei normale- und zwei Gated Invertern – es sind insgesamt $2 \times 2 + 2 \times 4 = 12$ Transistoren - besser



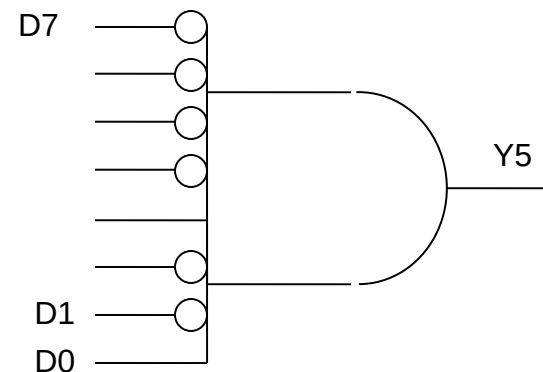
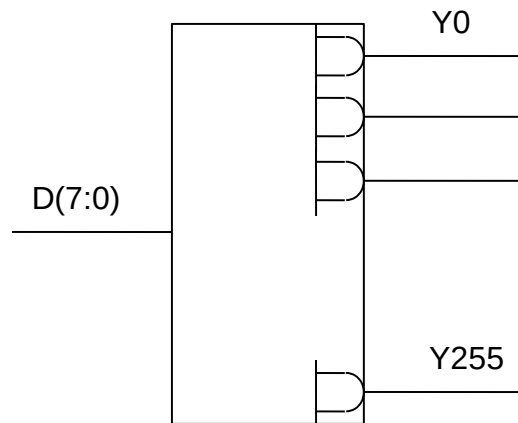
- Oft werden auch mehrfache Multiplexer verwendet.
- ZB wenn man die digitalen Signale von mehreren Quellen über eine Leitung übertragen möchte.
- 2->1 Multiplexer
- 4->1 Multiplexer...



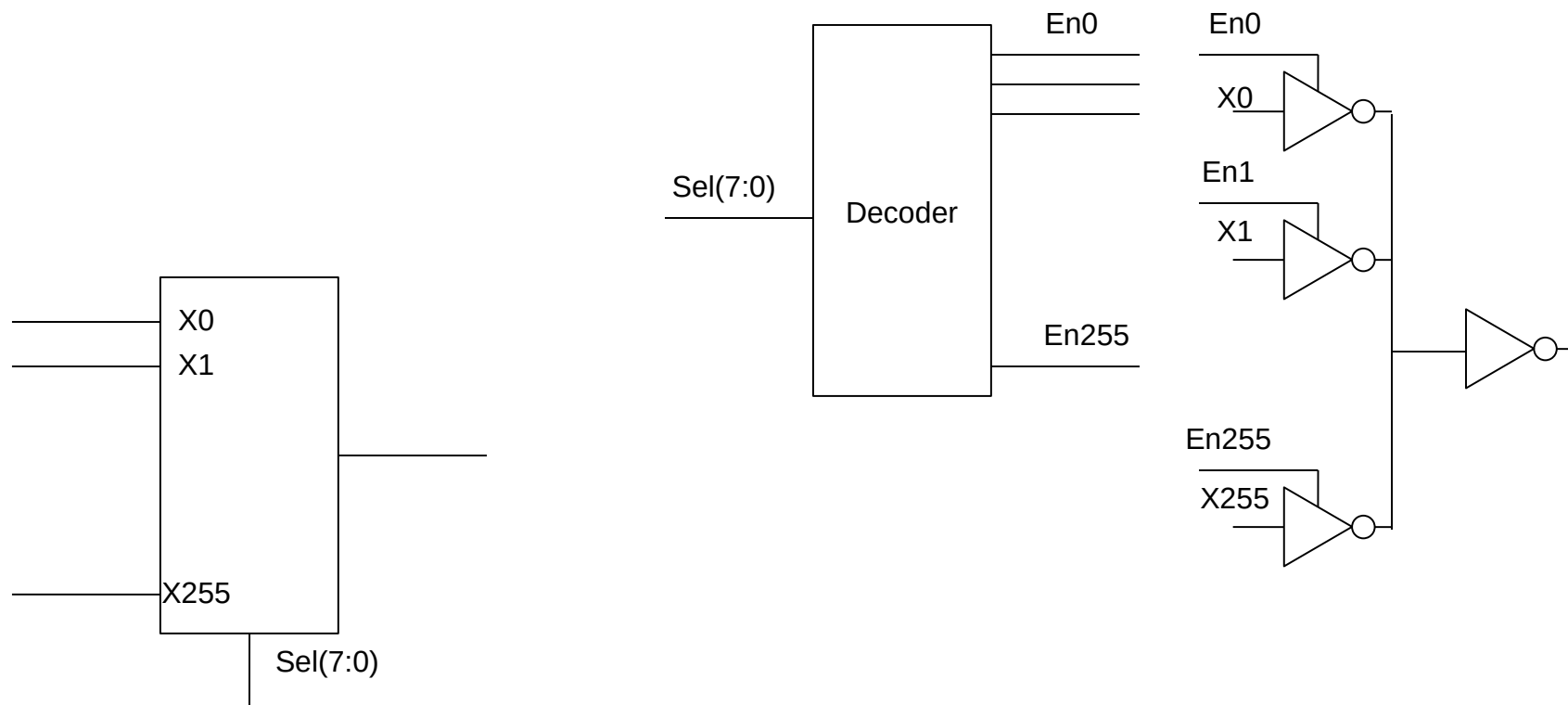
- Decoder.
- Beispiel: 8-bit Eingang D(7:0) (binäre Zahl) und 2^8 Ausgänge.
- Falls Eingang = m (binär Kodiert) ist der m-te Ausgang 1. Alle anderen Ausgänge sind null.



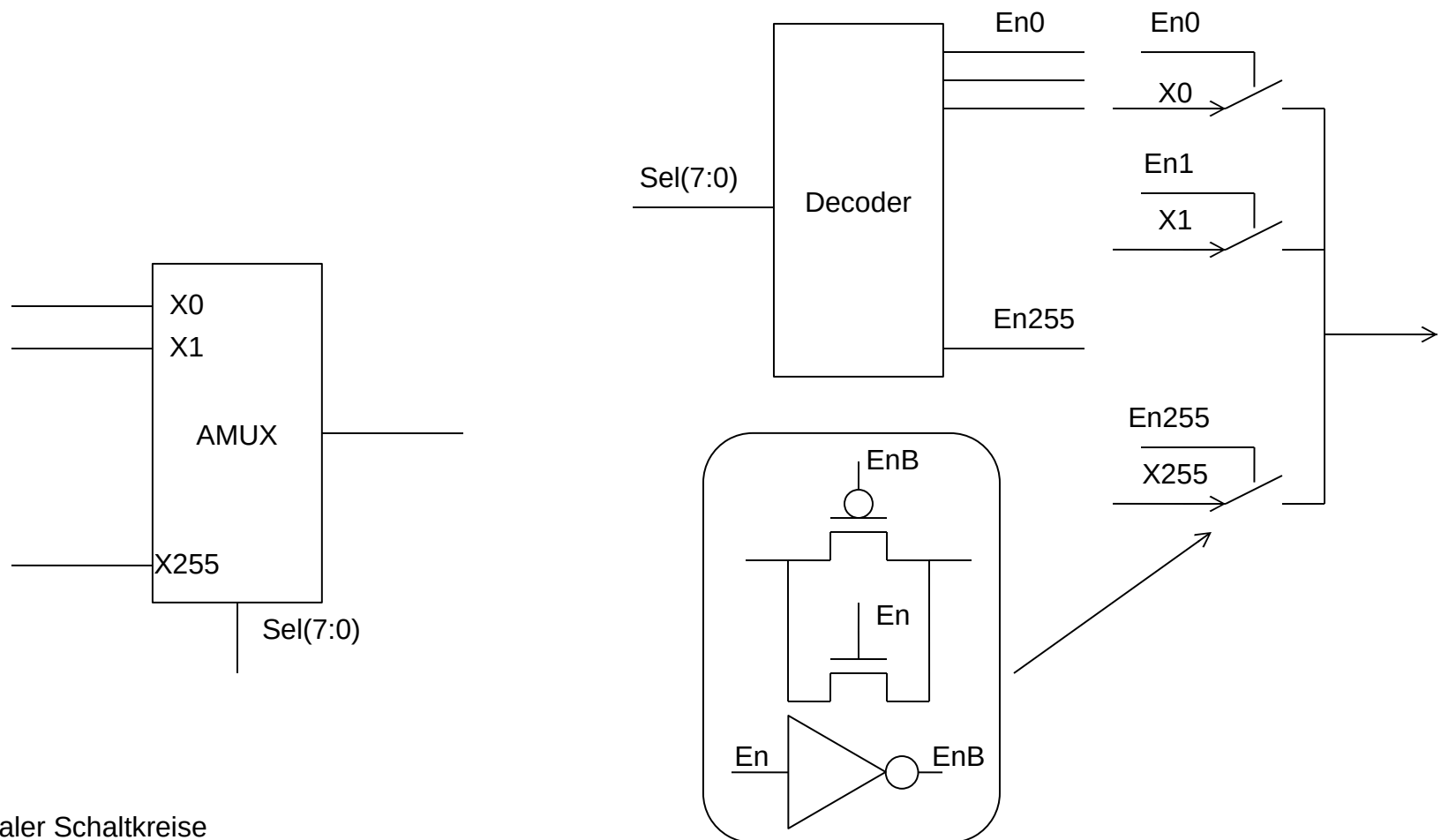
- Realisierung
- 2^8 AND Gattern mit n Eingängen
- Wenn z.B. das AND Gate dem Ausgang 5 gehört, sollte es „1“ für die binäre Zahl $D(7:0) = 0000_1001$ erzeugen
- $Y5 = !D7 \& !D6 \& !D5 \& !D4 \& D3 \& !D2 \& !D1 \& D0$
- Alle Variablen, die null sind, werden negiert
- In solcher Realisierung brauchen wir 256 ANDs mit 8 Eingängen und 8 Invertern. Das sind insgesamt $256 \times 16 + 8 \times 2 \sim 4000$ Transistoren.



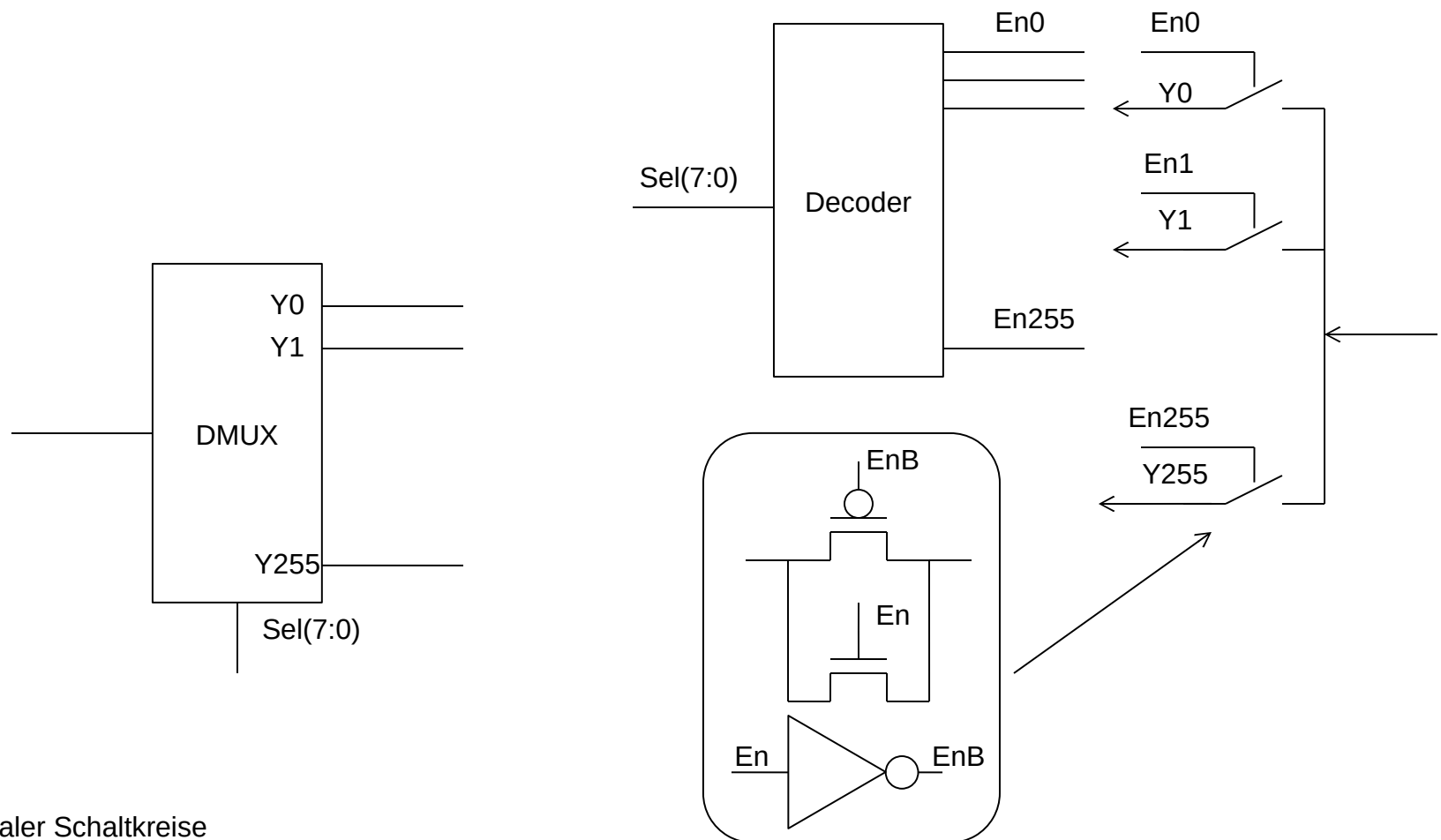
- 256->1 Multiplexer – Realisierung
- $1 \times 4000T + 256 \times 6T + 2T = 5500$ Transistoren



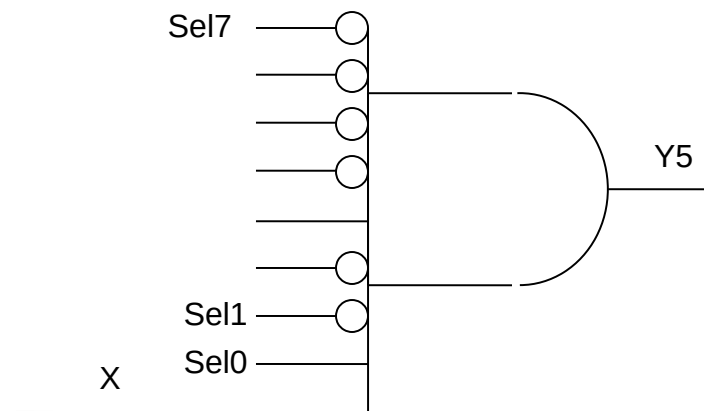
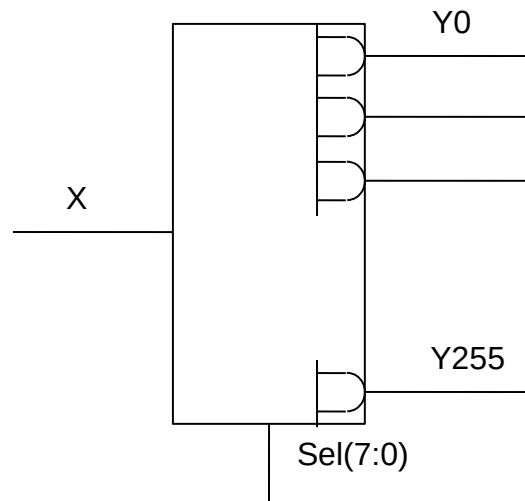
- 256->1 Analog Multiplexer – Realisierung



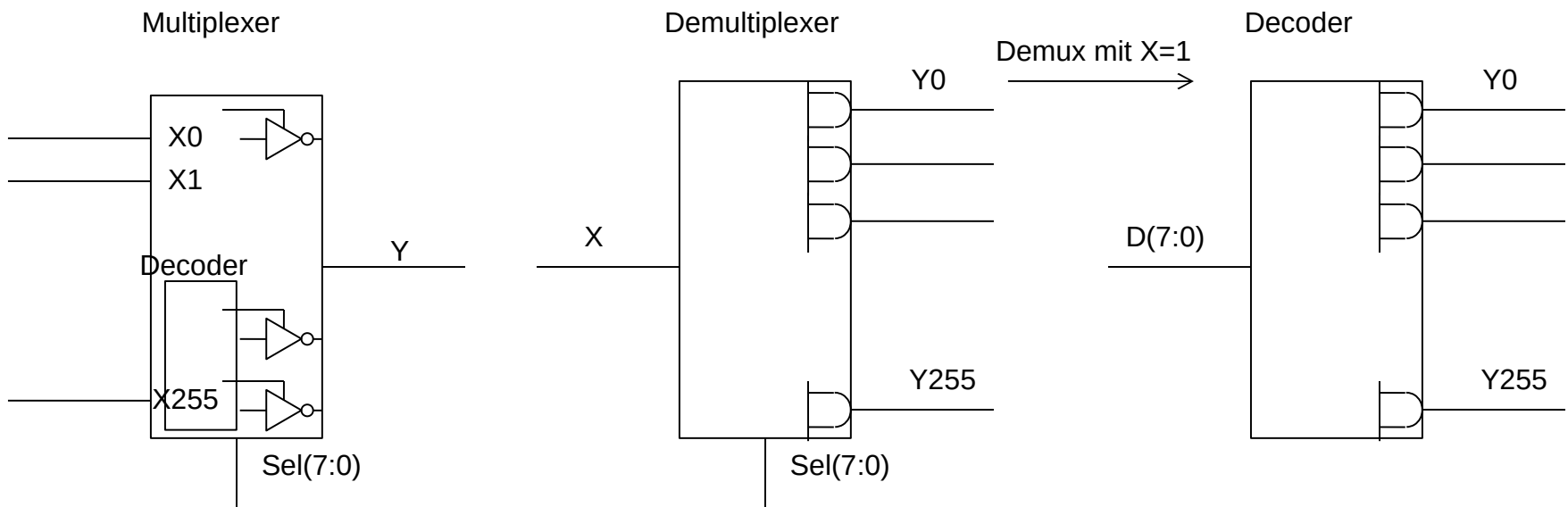
- Analog Multiplexer leitet in beide Richtungen
- Demultiplexer
- Wenn man in einem Analogmultiplexer die Eingänge und Ausgänge „vertauscht“ bekommt man einen Demultiplexer.



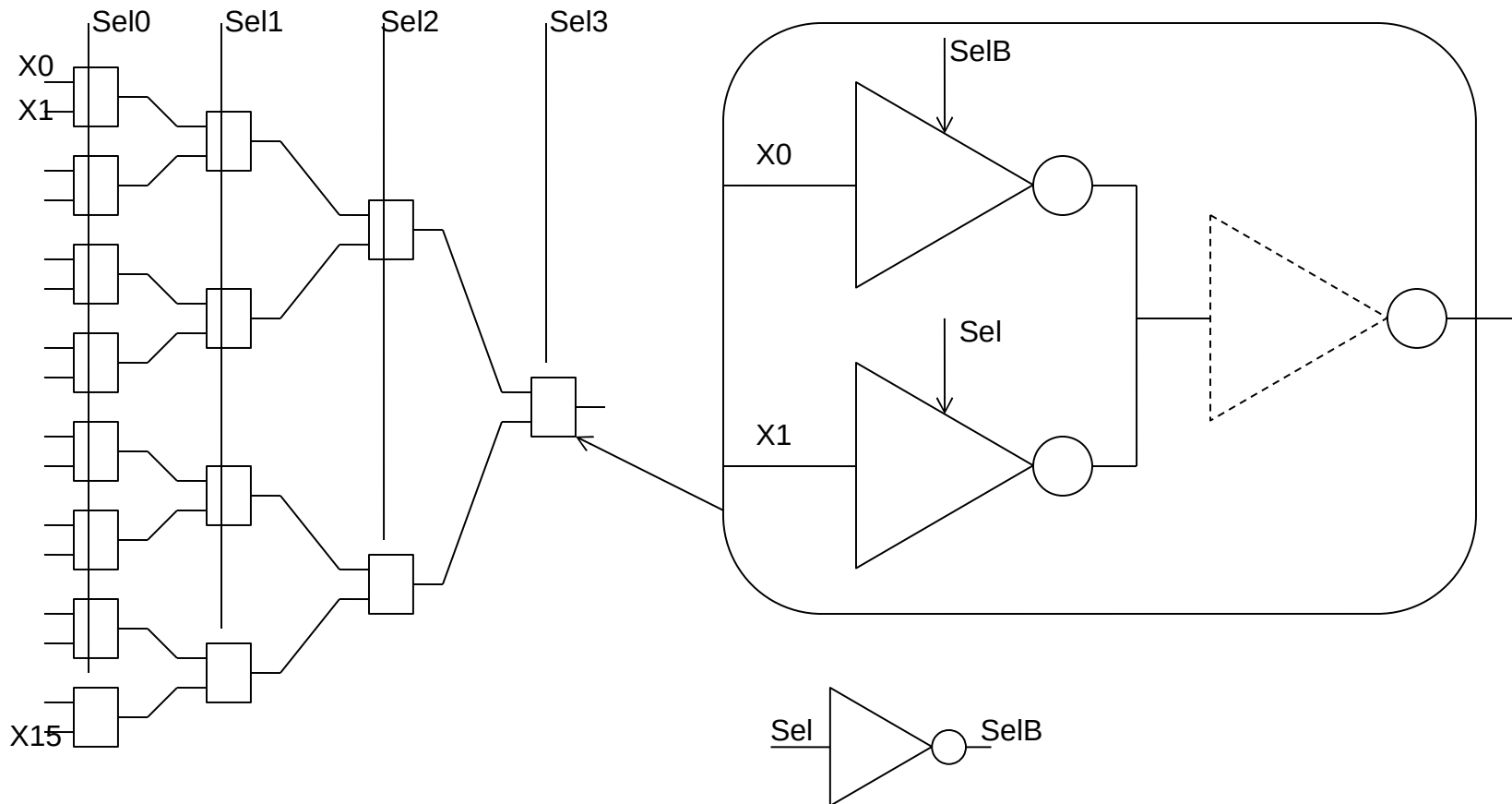
- Digital Demultiplexer
- Demultiplexer mit Eingang 1 -> Dekoder
- $256 \times 20T + 8 \times 2T \sim 5000T$



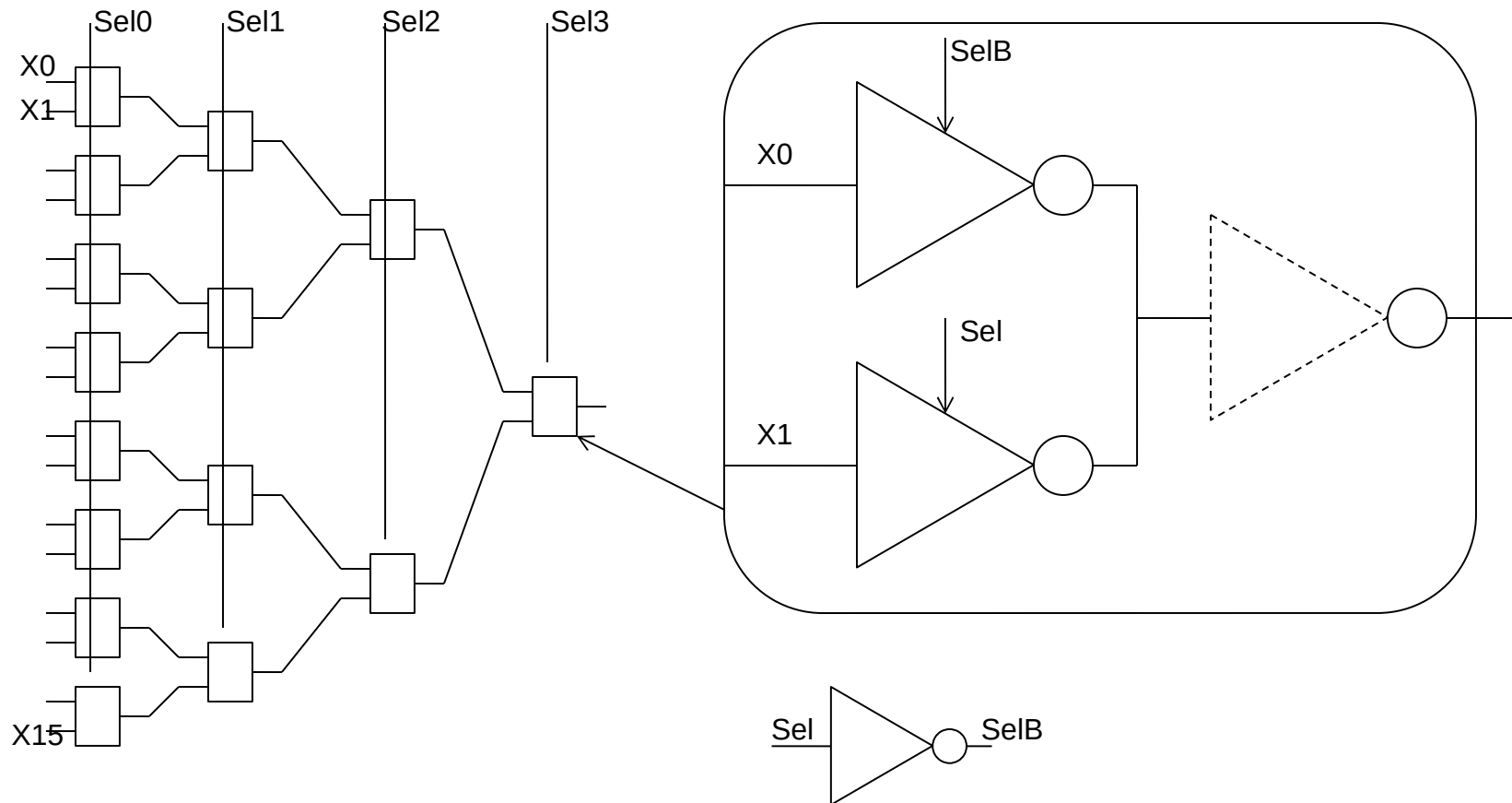
- Multiplexer
- Demultiplexer
- Dekoder
- Coder (nächstes mal)



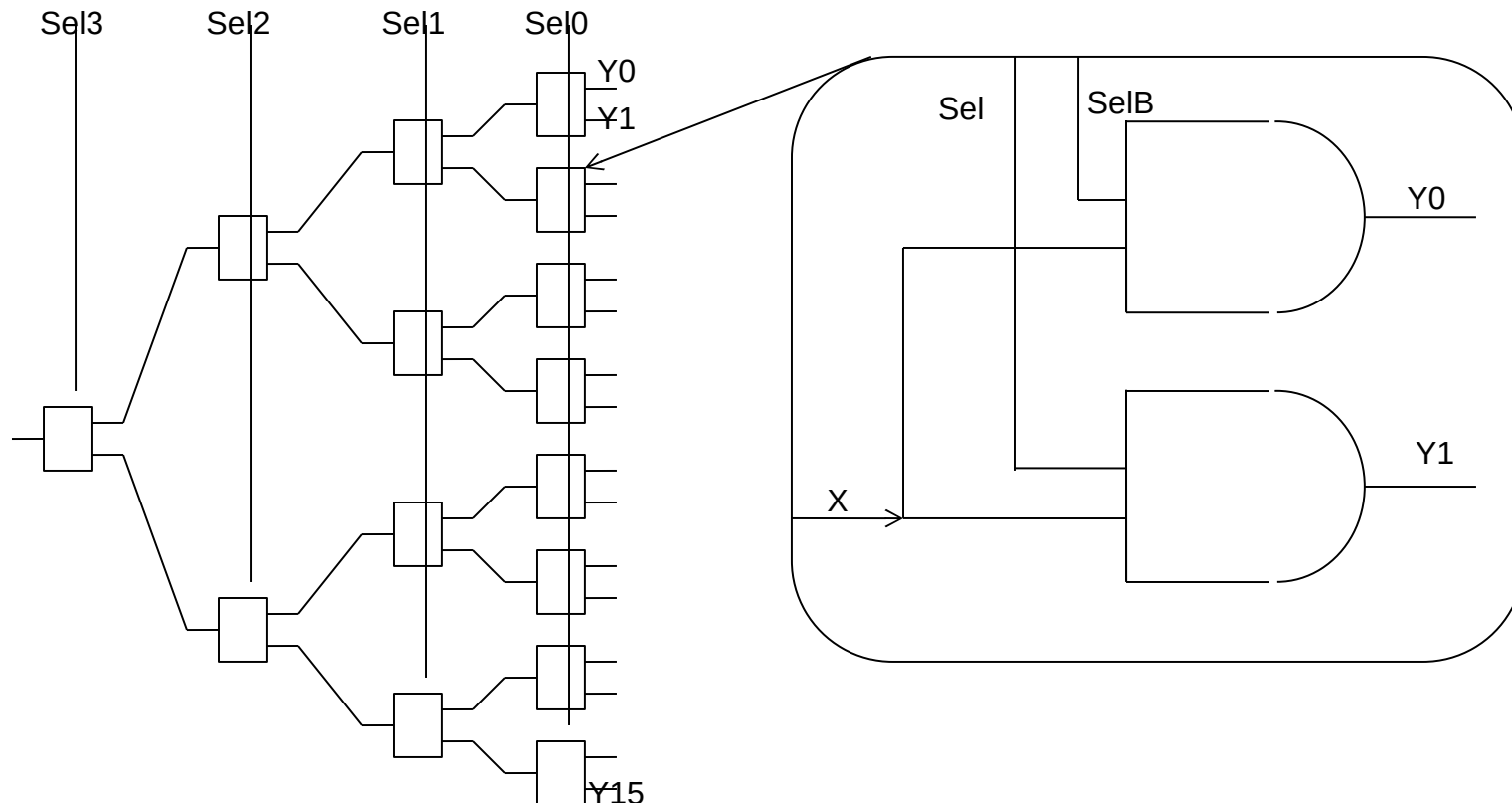
- Multiplexer kann mit weniger Transistoren realisiert werden
- Baumstruktur
- In jedem Knoten verwenden wir jeweils einen (2->1) Multiplexer. Der Select Eingang vom Multiplexer der ersten Stufe wird an Sel0 angeschlossen, usw.



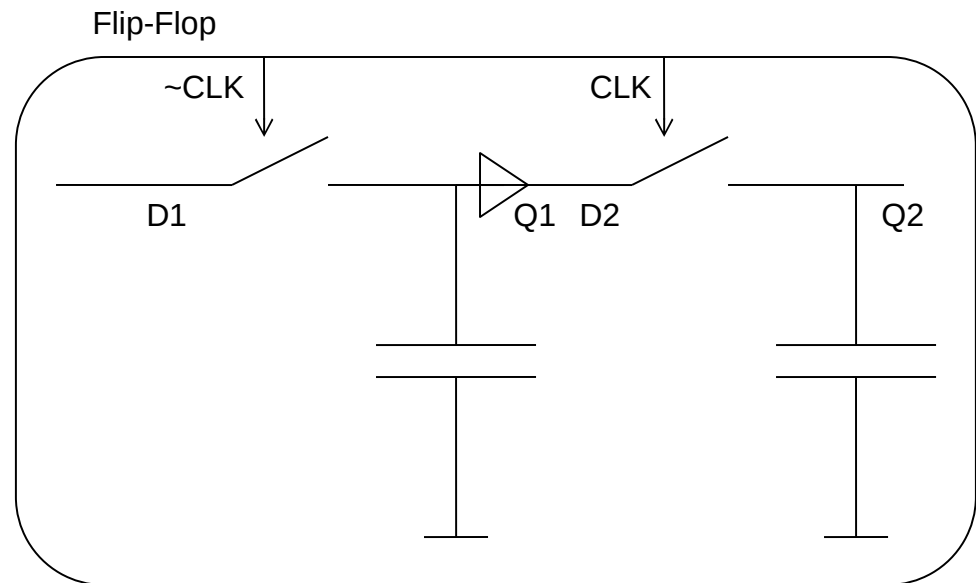
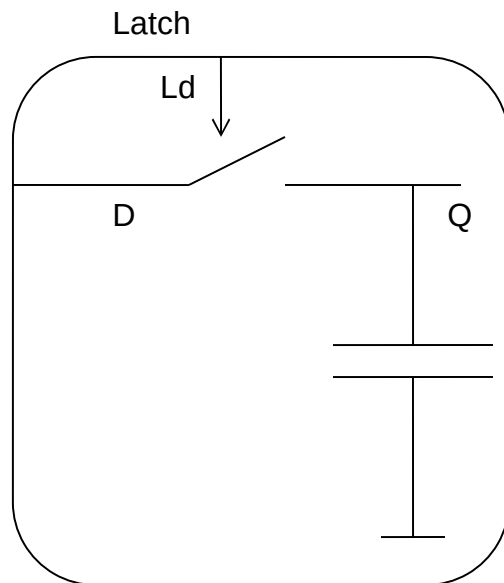
- 256->1 Multiplexer
- $255 (2 \rightarrow 1) \times 4T + 8 \times 2T \sim 1000$ Transistoren (5.5x kleiner)
- Die Schaltung ist langsamer – mehrere Stufen



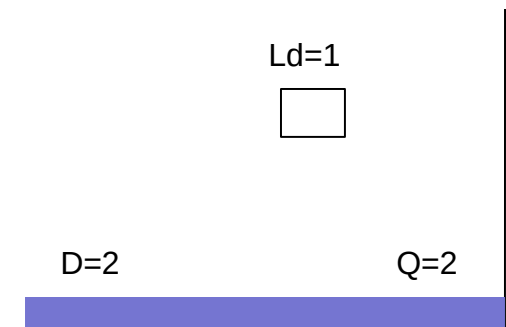
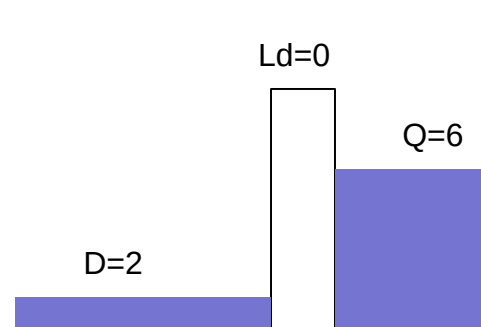
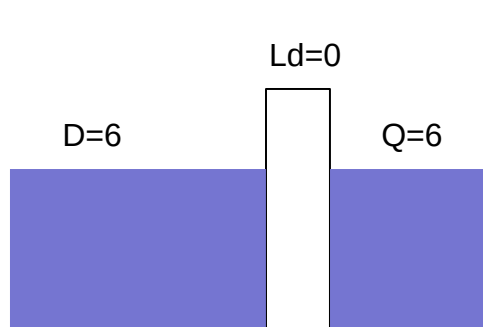
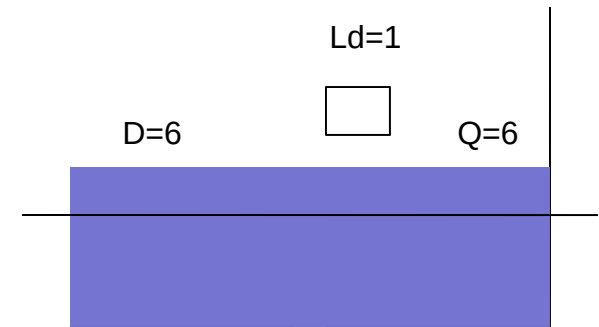
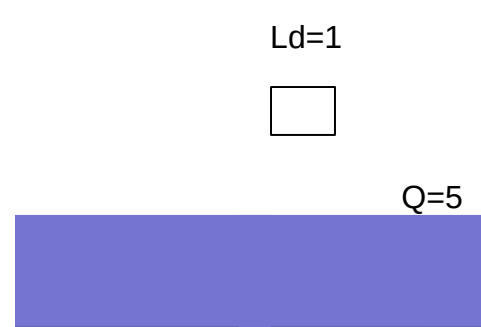
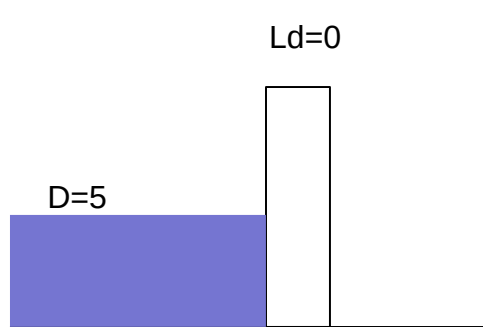
- Auch ein Demultiplexer (und Dekoder) kann als Baumstruktur realisiert werden
- In jedem Knoten verwenden wir jeweils einen (1->2) Demultiplexer
- $255 (1 \rightarrow 2) \times 12T + 8 \times 2T \sim 3000$ Transistoren (2000 mit Tricks)



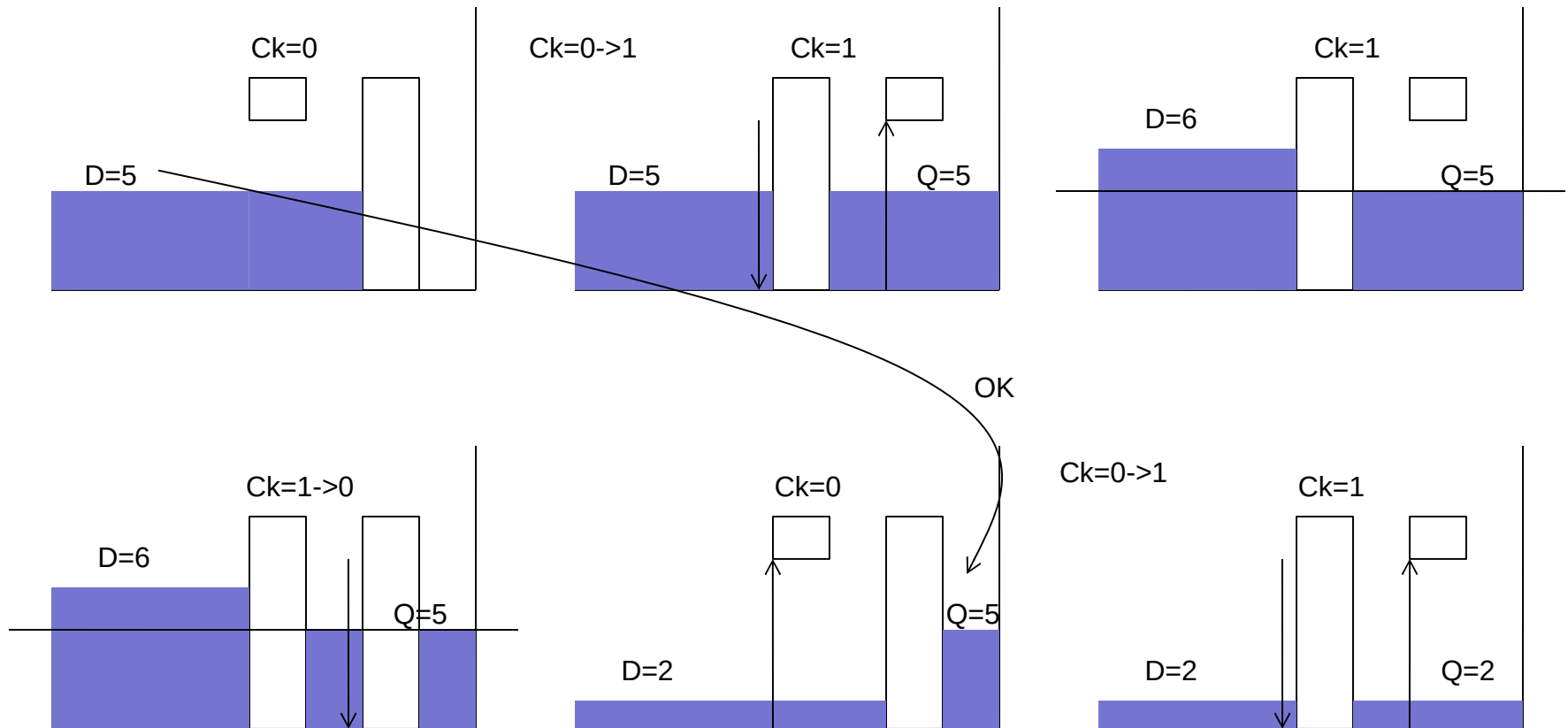
- Sequenzschaltungen
- S. Vorlesung 1 – Latch und Flip-Flop
- Latch – speichert ein Eingangsniveau (auf einem Kondensator) wenn Load Signal = 1. Wenn Load = 0, der Zustand bleibt erhalten
- Flip-Flop – 2 Latch-es in Reihe
- Der Eingangswert D wird im Moment der steigenden Talkflanke gespeichert
- Spätere Änderungen am D-Eingang haben keine Wirkung auf den Ausgang bis zur nächsten Taktflanke



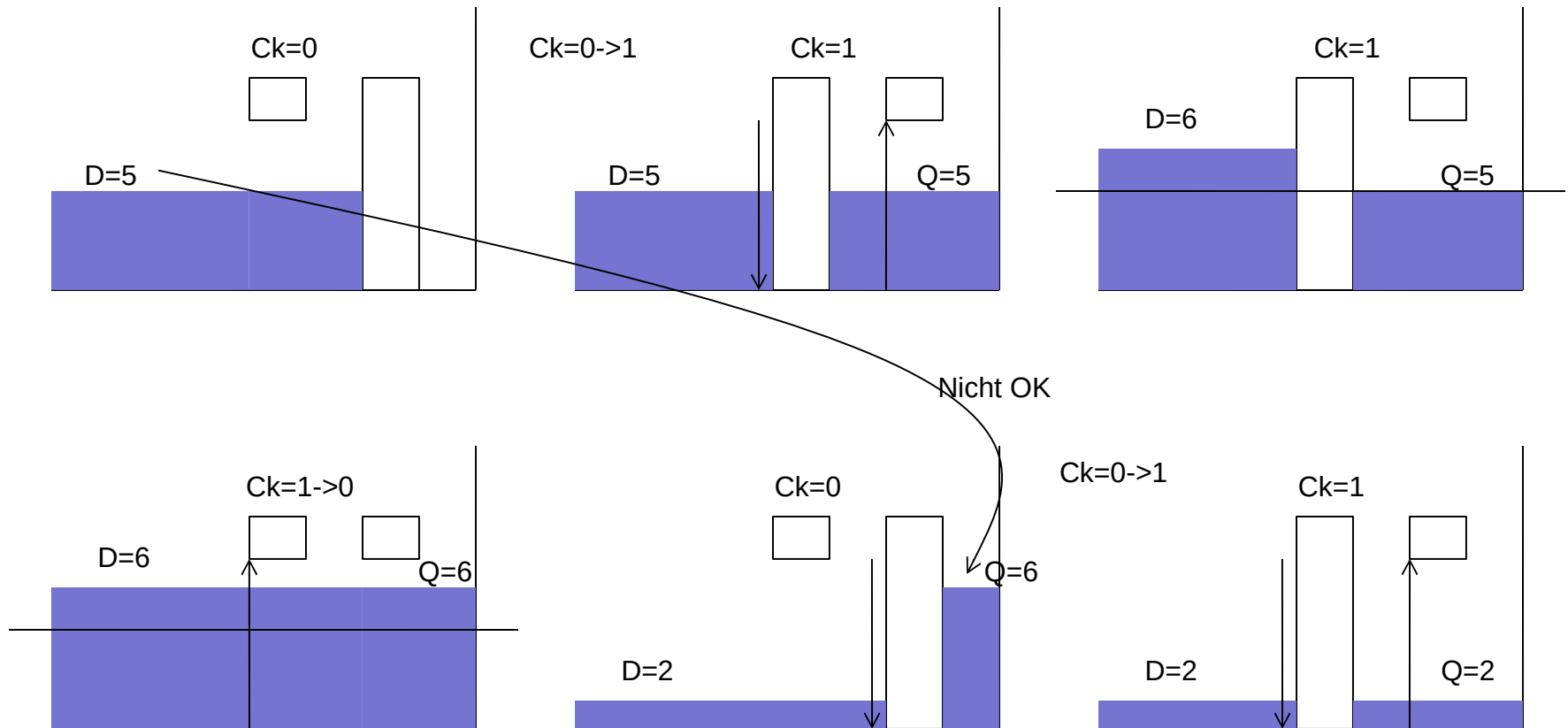
- Es erinnert an ein System mit Schleusen.



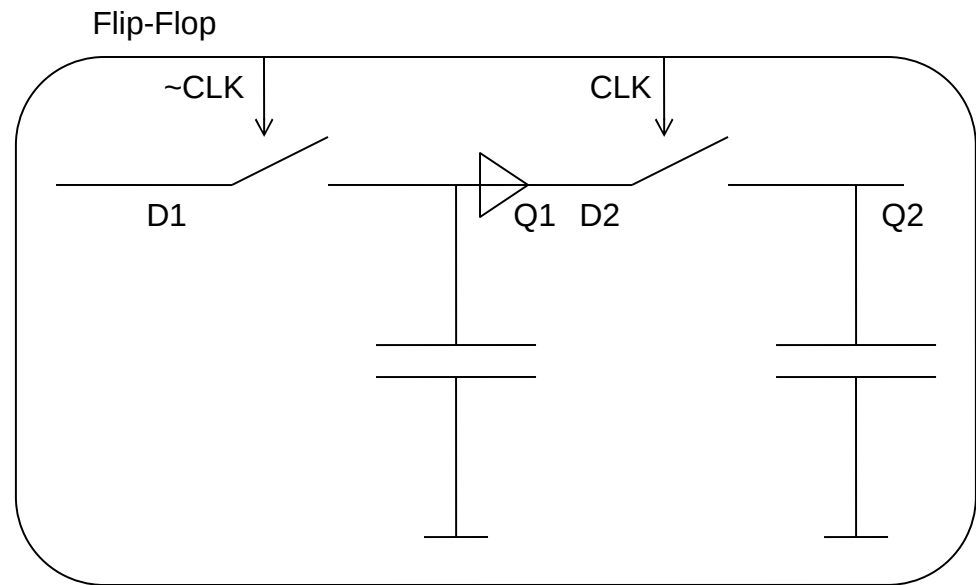
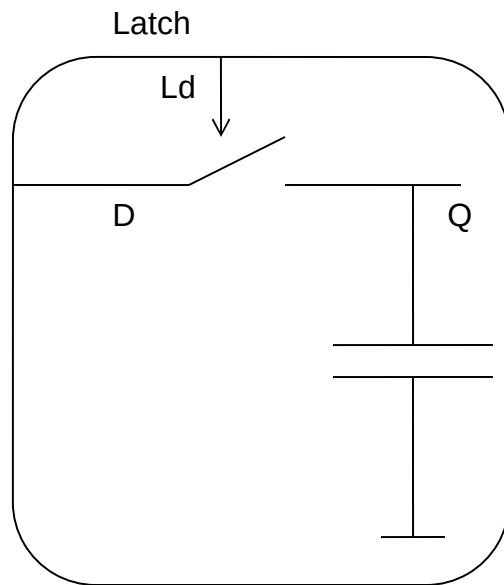
- Es erinnert an ein System mit Schleusen.



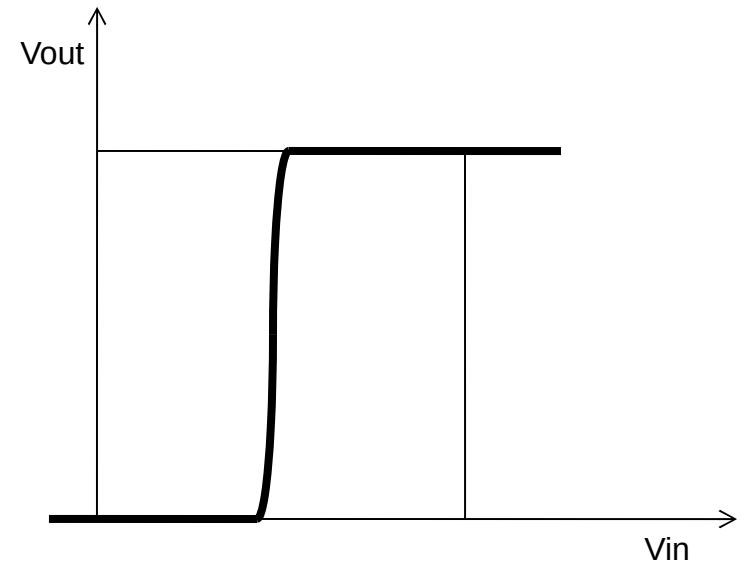
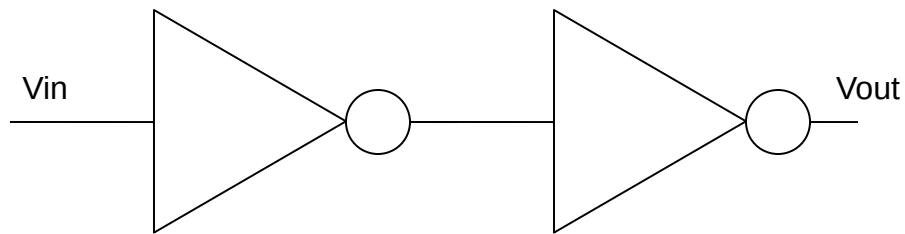
- Es erinnert an ein System mit Schleusen.



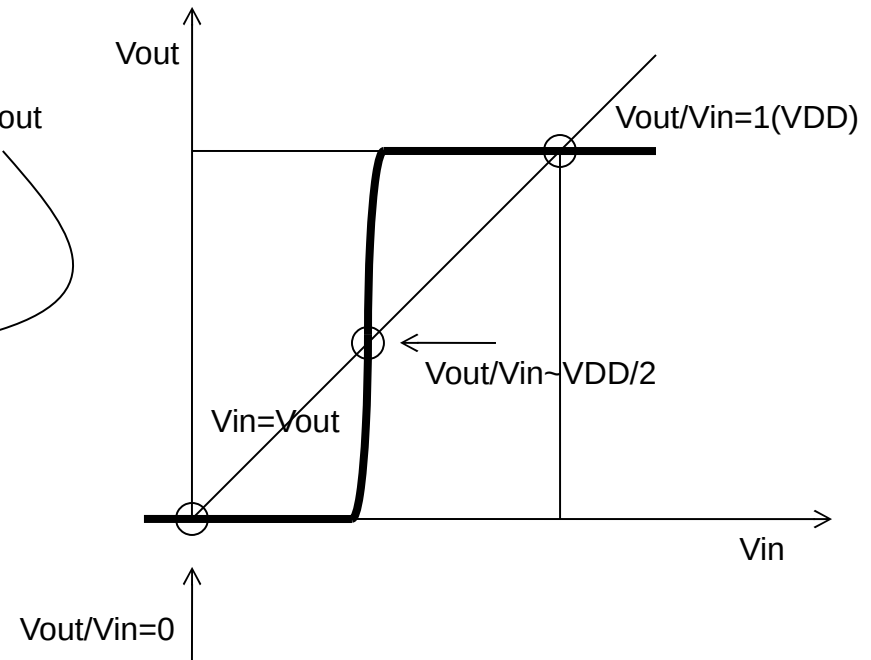
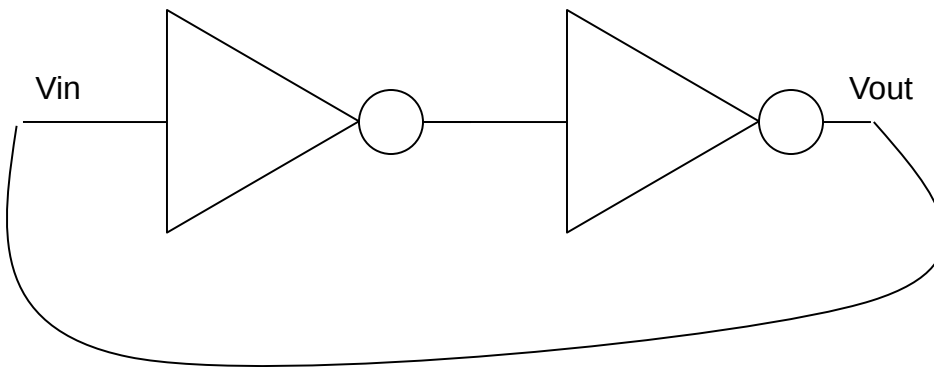
- Nachteil einer Latch Schaltung mit Kondensator – sie kann den Zustand nicht beliebig lange halten.
- Der Kondensator wird langsam entladen.
- Dynamische Logik.



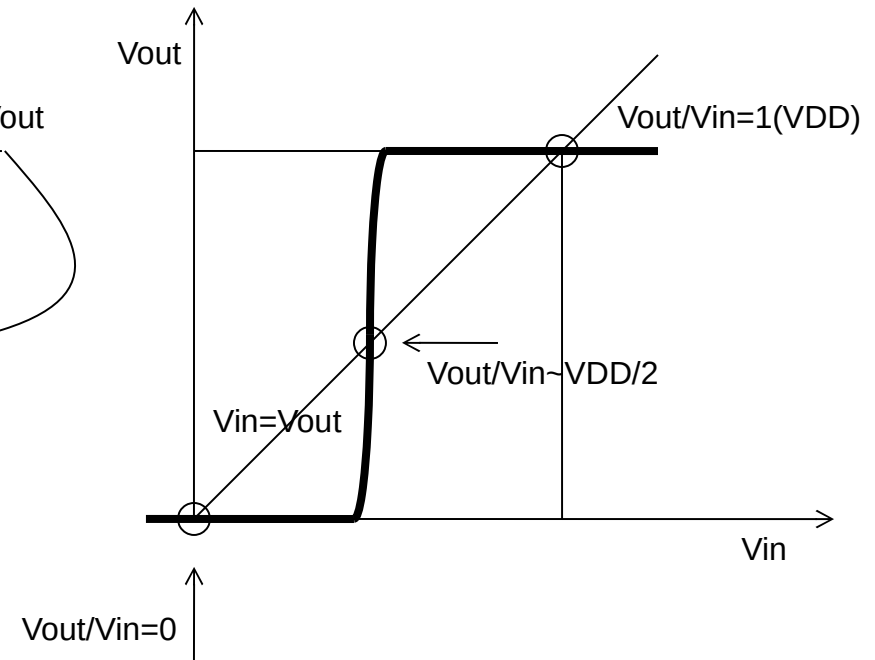
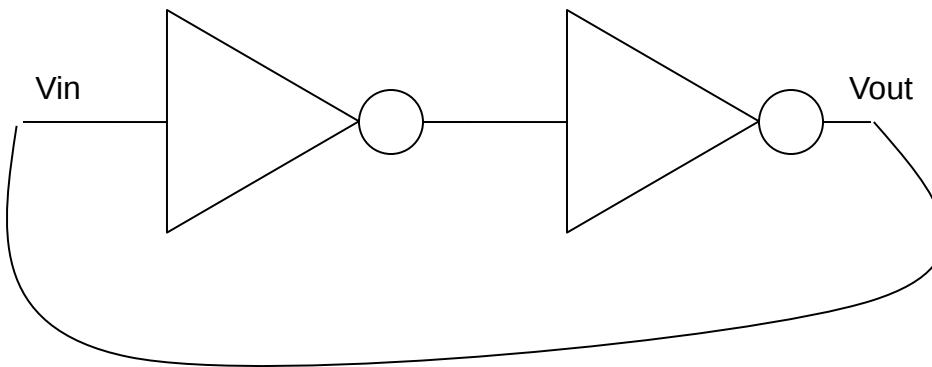
- Statische Speicherzellen



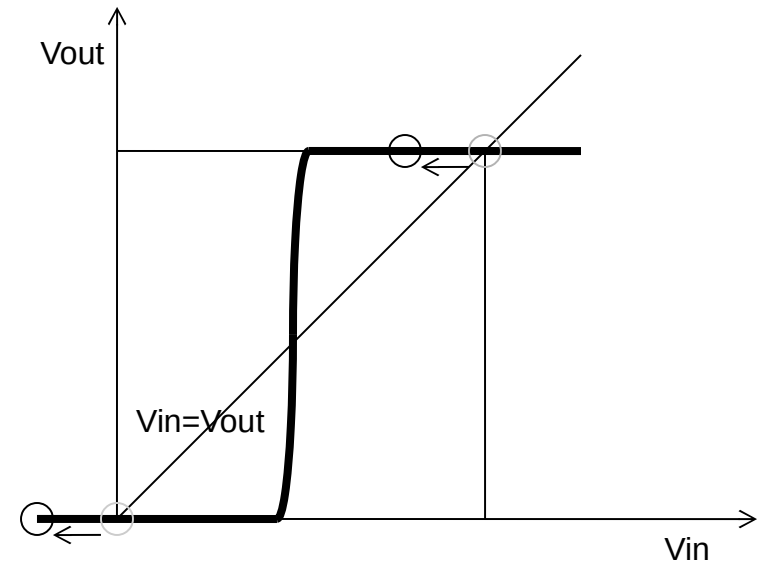
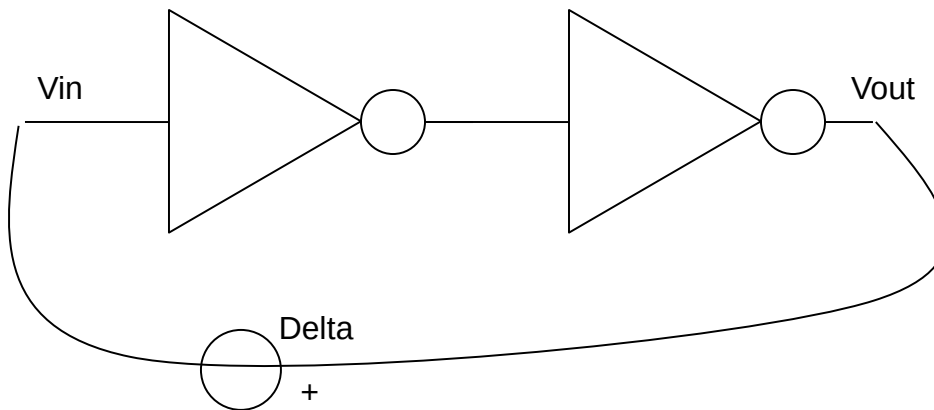
- Statische Speicherzellen
- Wenn wir den Ausgang des zweiten Inverters mit dem Eingang des ersten verbinden, haben wir $V_{in} = V_{out}$.
- Der Zustand der Schaltung liegt im Schnittpunkt der Kennlinie $V_{out} = f(V_{in})$ und der Geraden $V_{out} = V_{in}$.



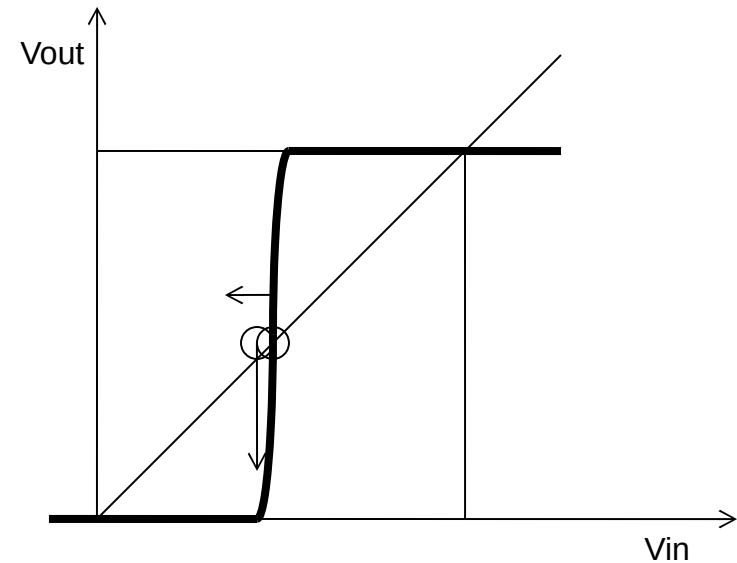
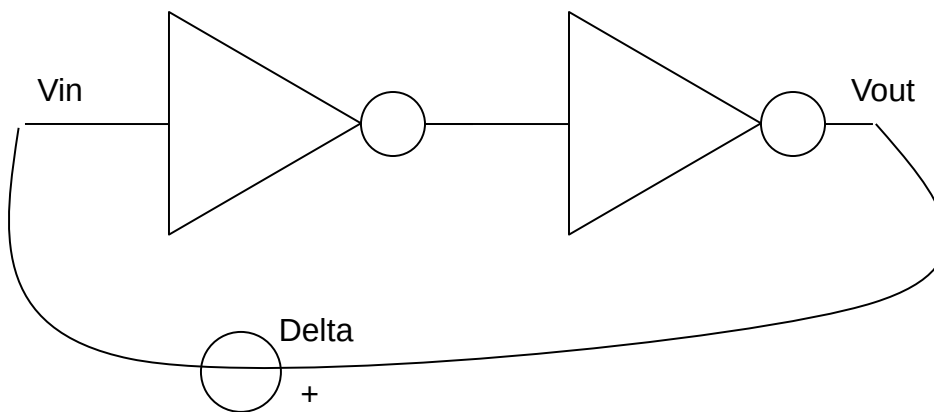
- Drei Schnittpunkte
- 1. $V_{out}/V_{in} = 0$ (logische 0)
- 2. $V_{out}/V_{in} = VDD$ (logische 1)
- 3. $V_{out}=V_{in} \sim VDD/2$ (undefiniert).



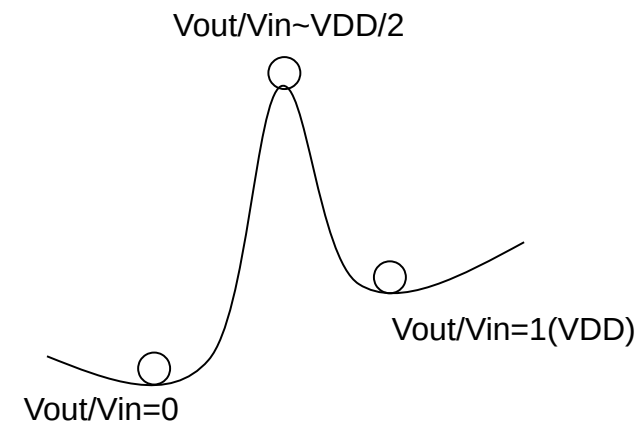
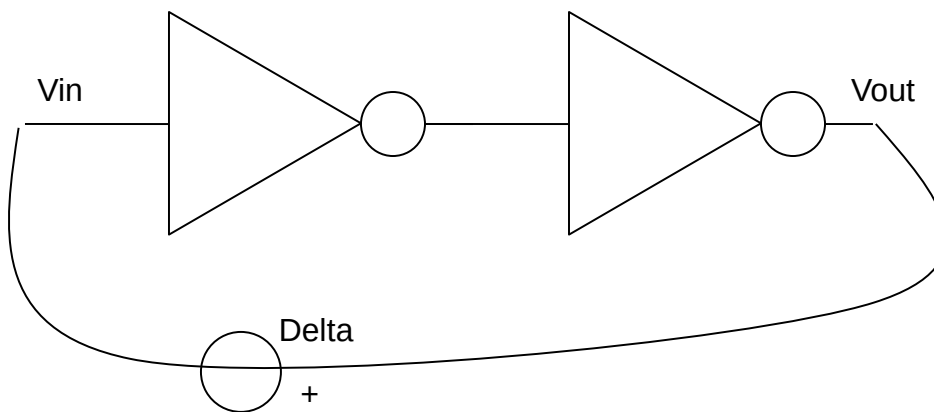
- Die ersten zwei Arbeitspunkte sind stabil
- kleine Störung Delta
- $V_{in} = V_{out} - \text{Delta}$
- V_{out} wird nicht beeinflusst



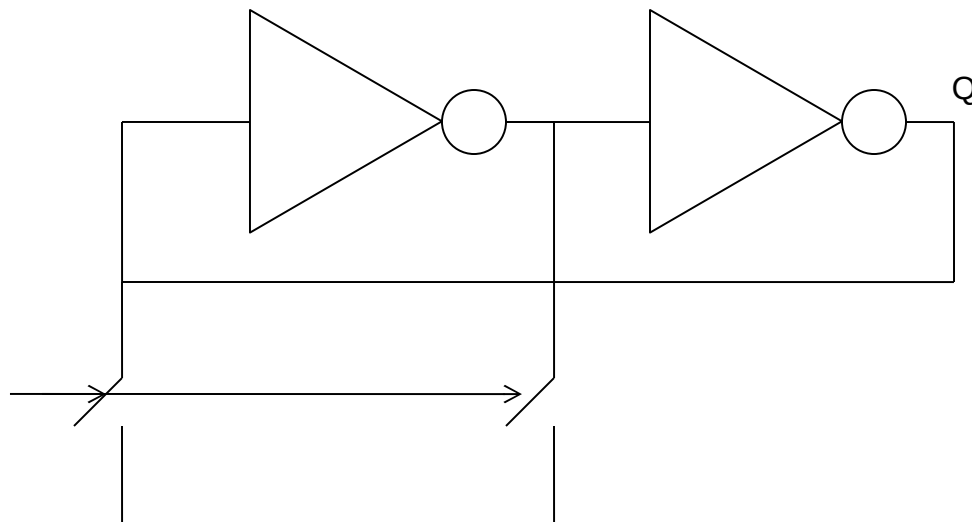
- Der dritte Arbeitspunkt ist instabil
- $V_{in} = V_{out} - \Delta$
- Verringerung von V_{in} führt zu noch größerer Verringerung von V_{out}
- Die Schaltung kommt aus dem instabilen Arbeitspunkt immer in den Arbeitspunkt $V_{out}/V_{in} = 0$ oder in den Arbeitspunkt $V_{out}/V_{in} = V_{DD}$



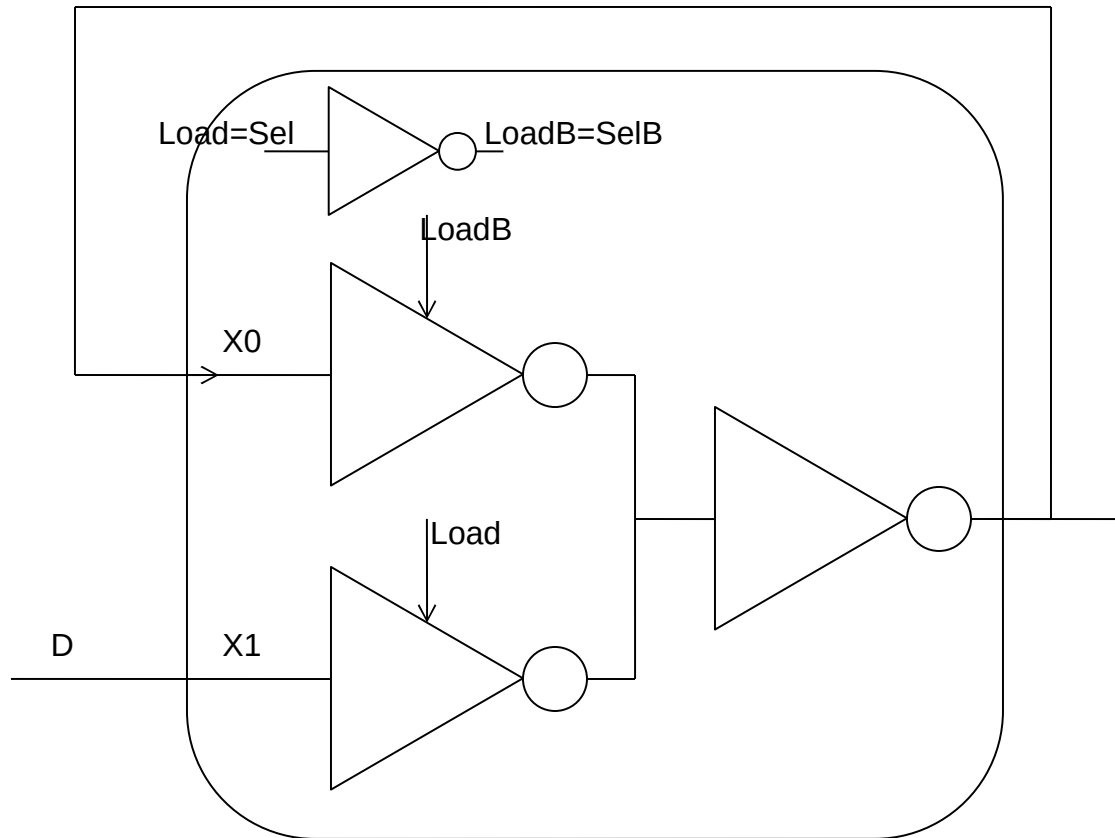
- Der dritte Arbeitspunkt ist instabil
- $V_{in} = V_{out} - \Delta$
- Verringerung von V_{in} führt zu noch größerer Verringerung von V_{out}
- Die Schaltung kommt aus dem instabilen Arbeitspunkt immer in den Arbeitspunkt $V_{out}/V_{in} = 0$ oder in den Arbeitspunkt $V_{out}/V_{in} = VDD$



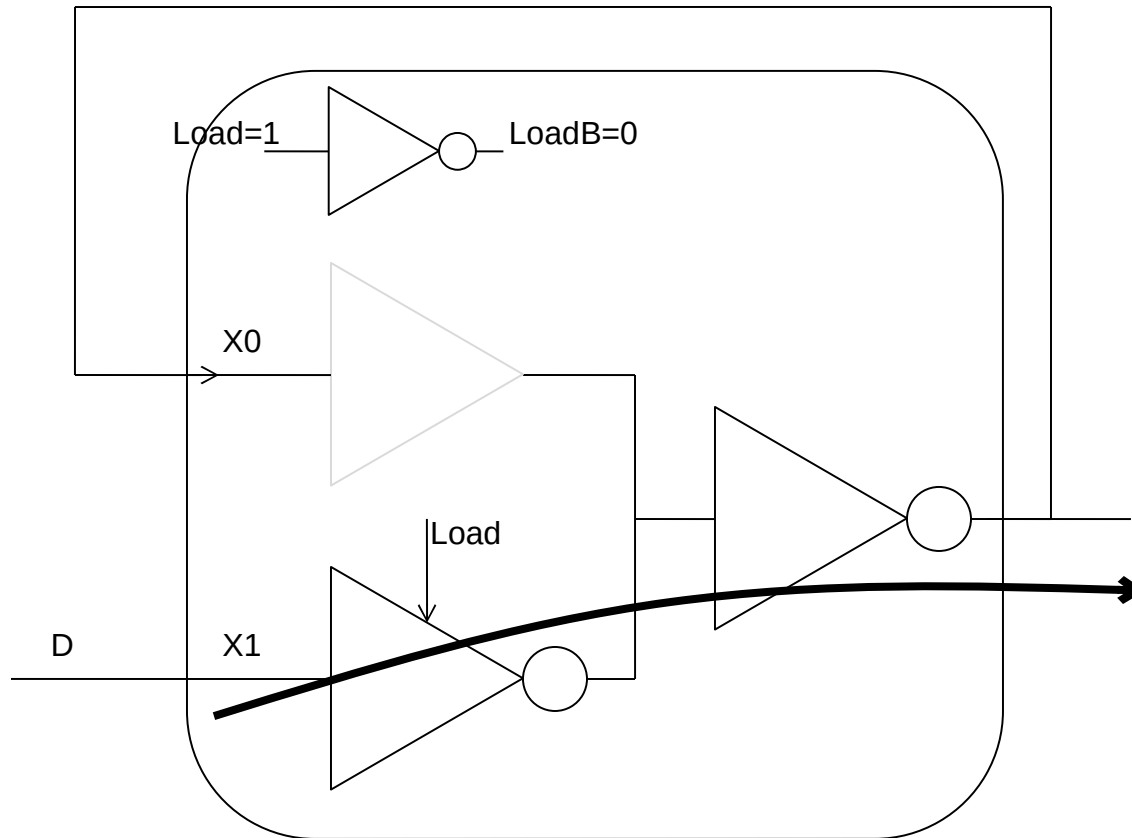
- Die Schaltung ist die Basis einer SRAM Zelle



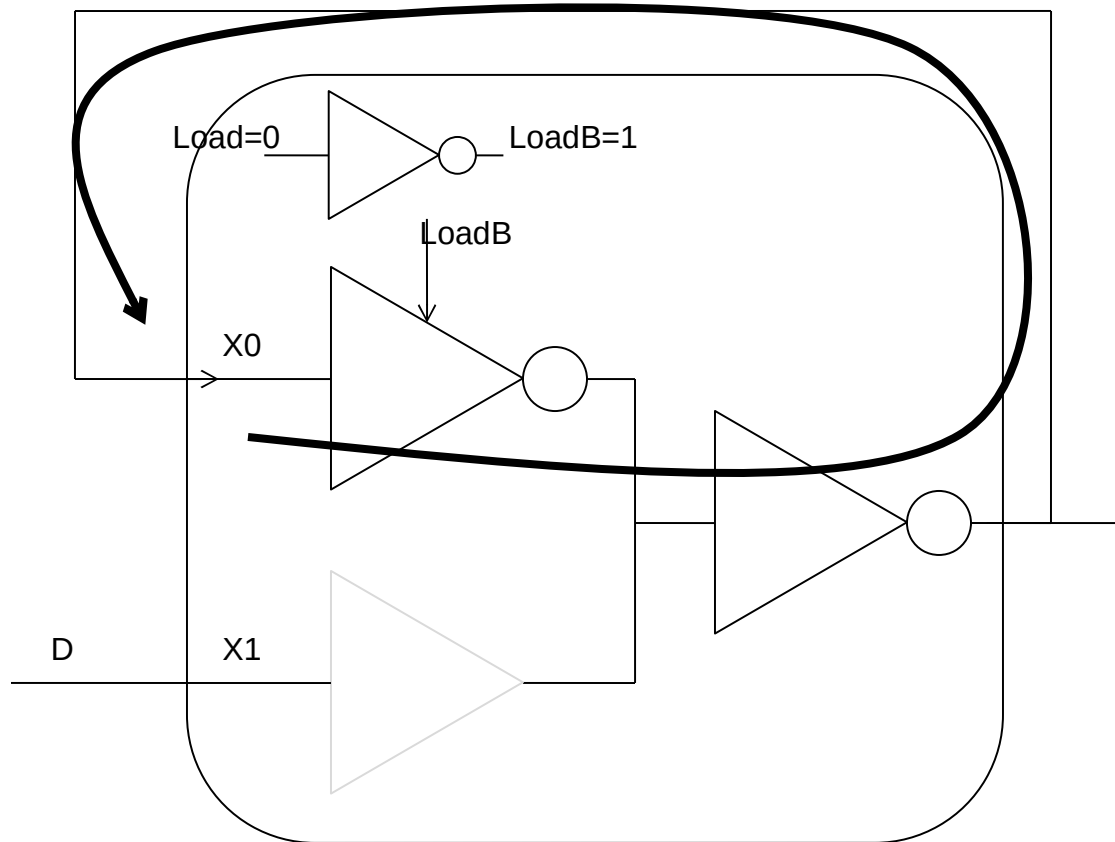
- Ein Latch basiert normalerweise auf einer modifizierten Version der Speicherzelle.
- Multiplexer wird benutzt, Select Eingang ist an Load Signal angeschlossen



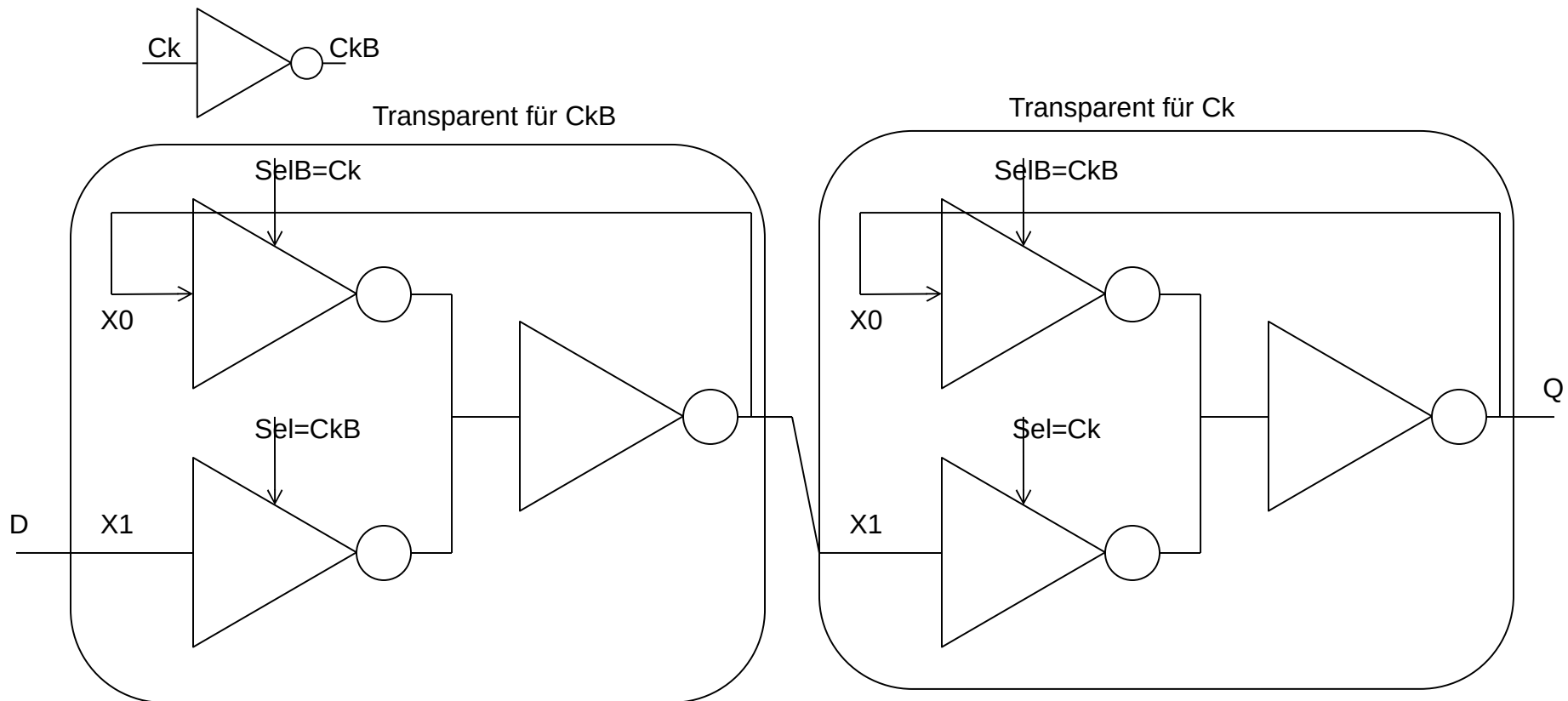
- Wenn Load = 1, das Latch ist „transparent“ – der Eingang ist direkt am Ausgang sichtbar.



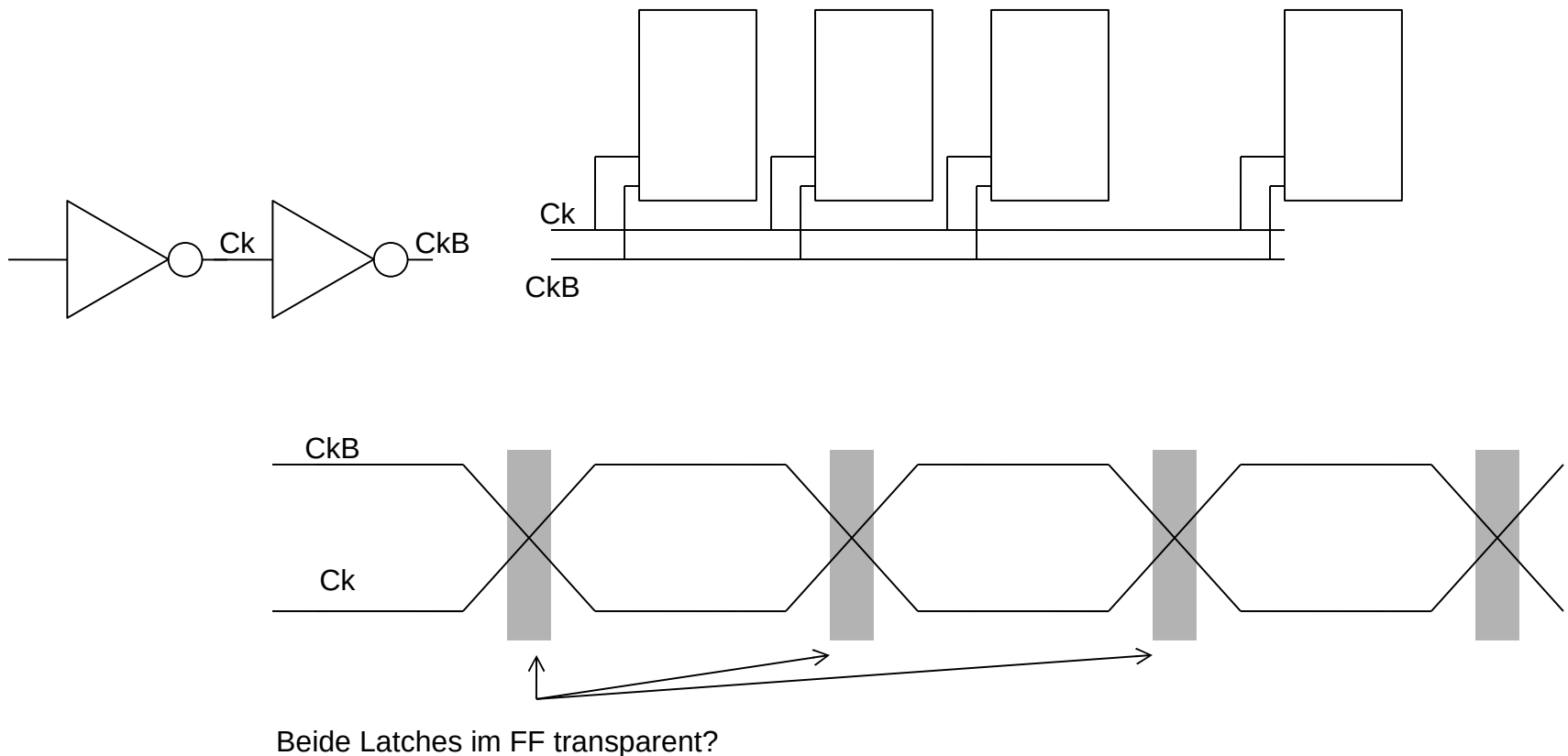
- Wenn Load = 0, haben wir dieselbe Schaltung wie in einer RAM Zelle. Der Multiplexer behält den Zustand



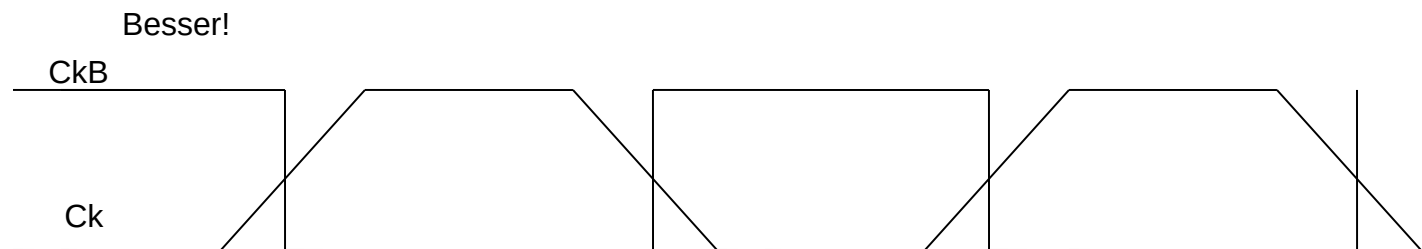
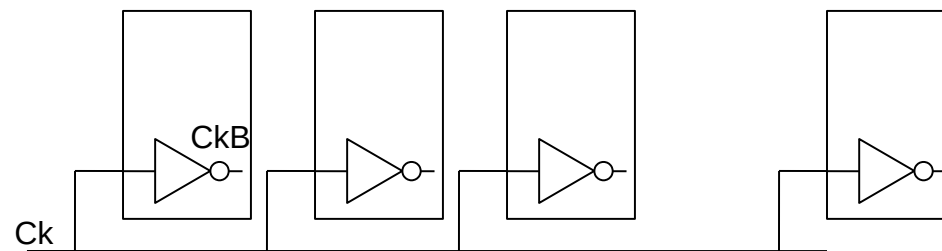
- Einen Flip-Flop bilden wir aus zwei Latches
- Es soll dabei vermieden werden, dass beide Latches gleichzeitig transparent werden, vor allem wenn sich Ck von 1 auf 0 ändert (inaktive Flanke)



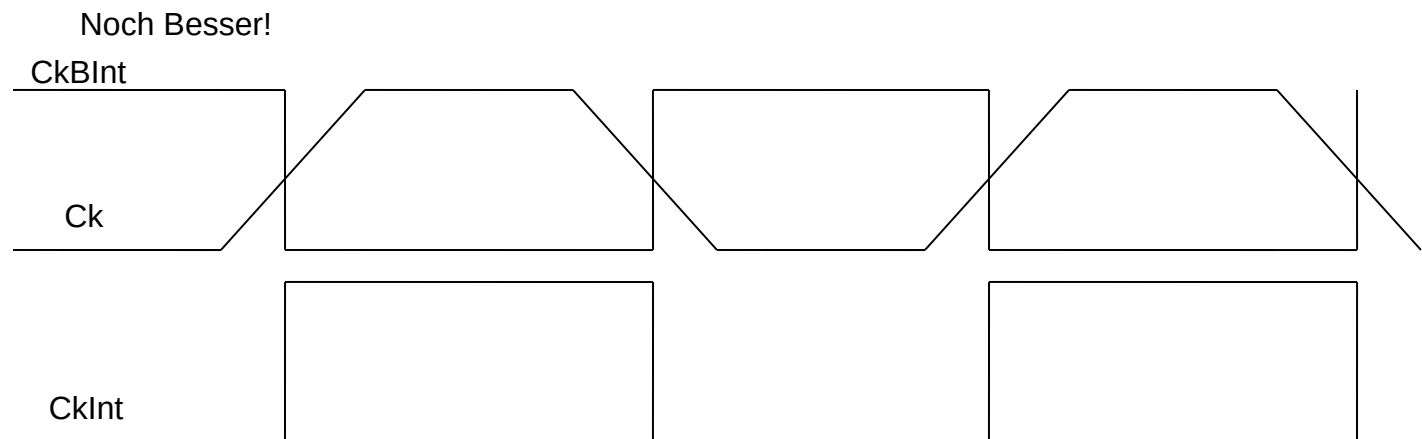
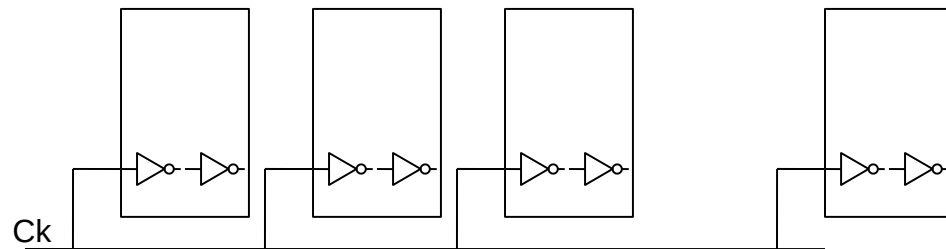
- Kapazitive Last verlangsamt die CMOS Schaltungen
- Schlechte Idee – viele Flip-Flops teilen zwei Taktinvertern
- Layout kleiner aber funktioniert nicht



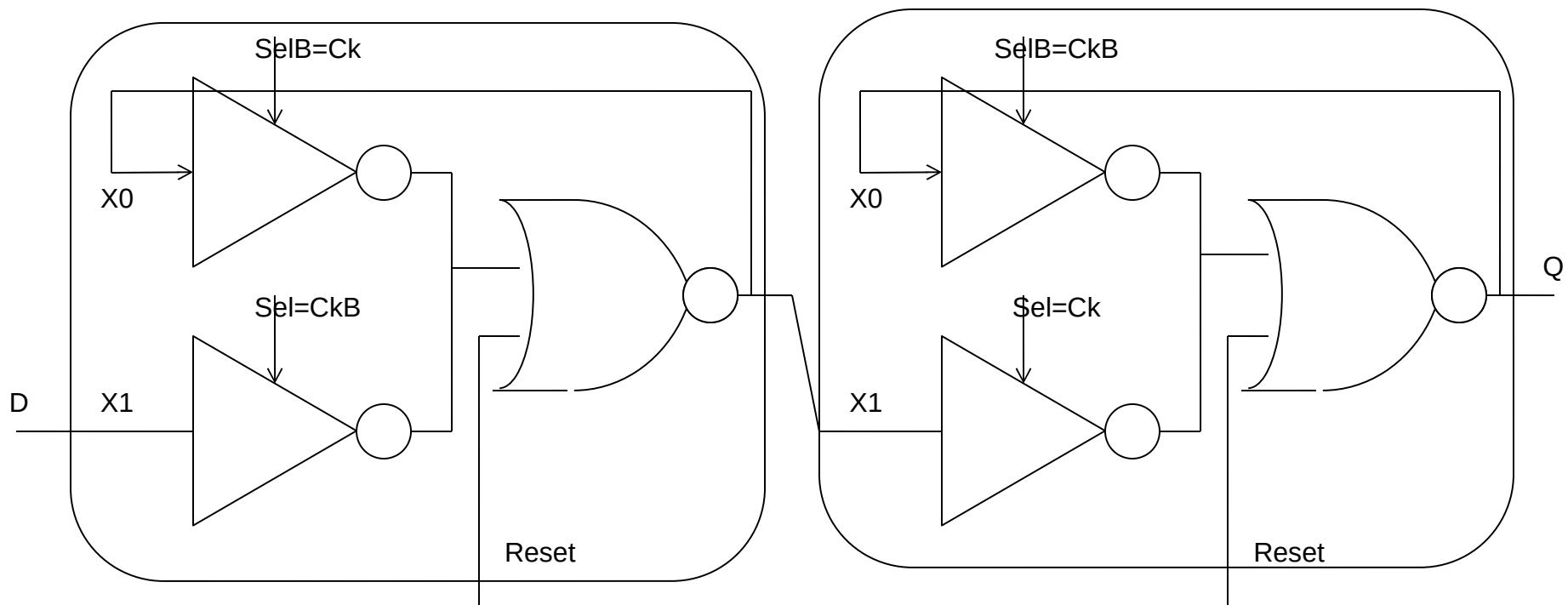
- Um solche Probleme zu vermeiden, werden die Takt-Inverter in der Regel im Flip-Flop eingebaut.



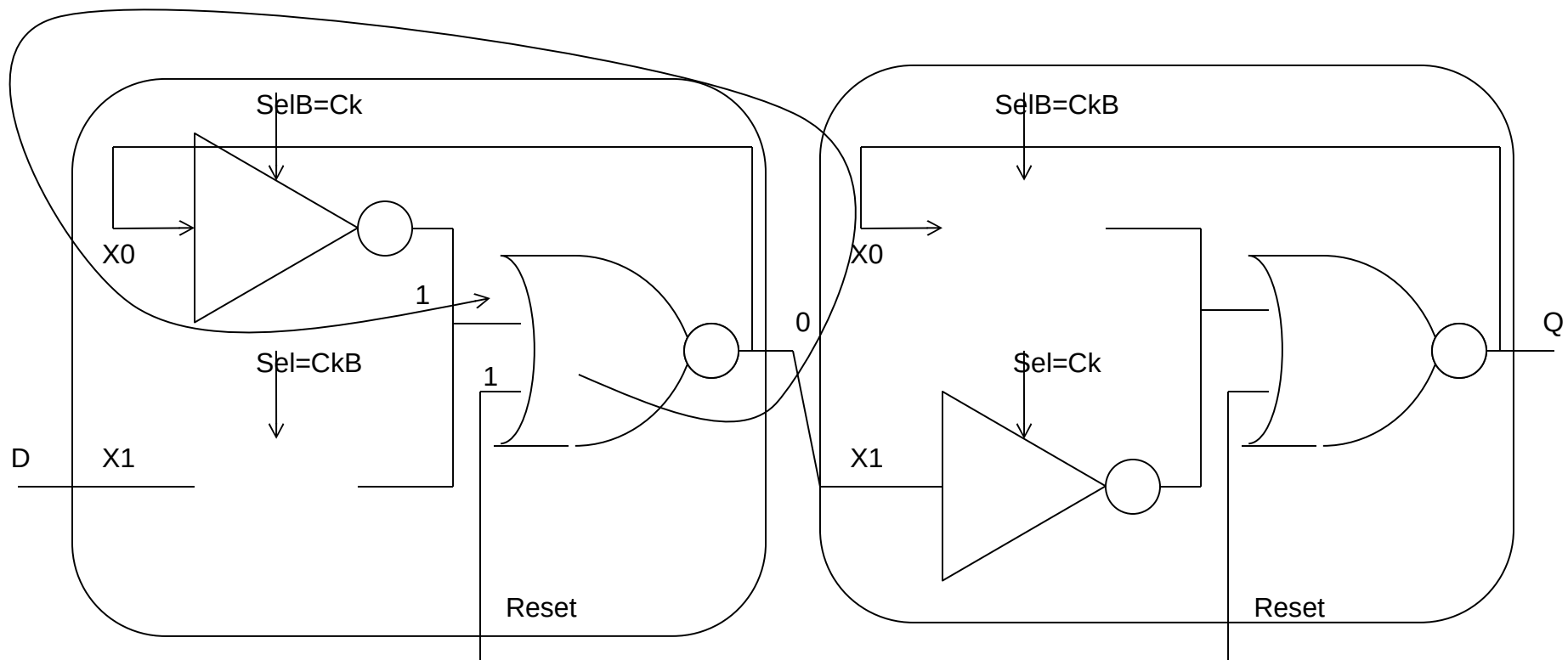
- Um solche Probleme zu vermeiden, werden die Takt-Invertern in der Regel im Flip-Flop eingebaut.



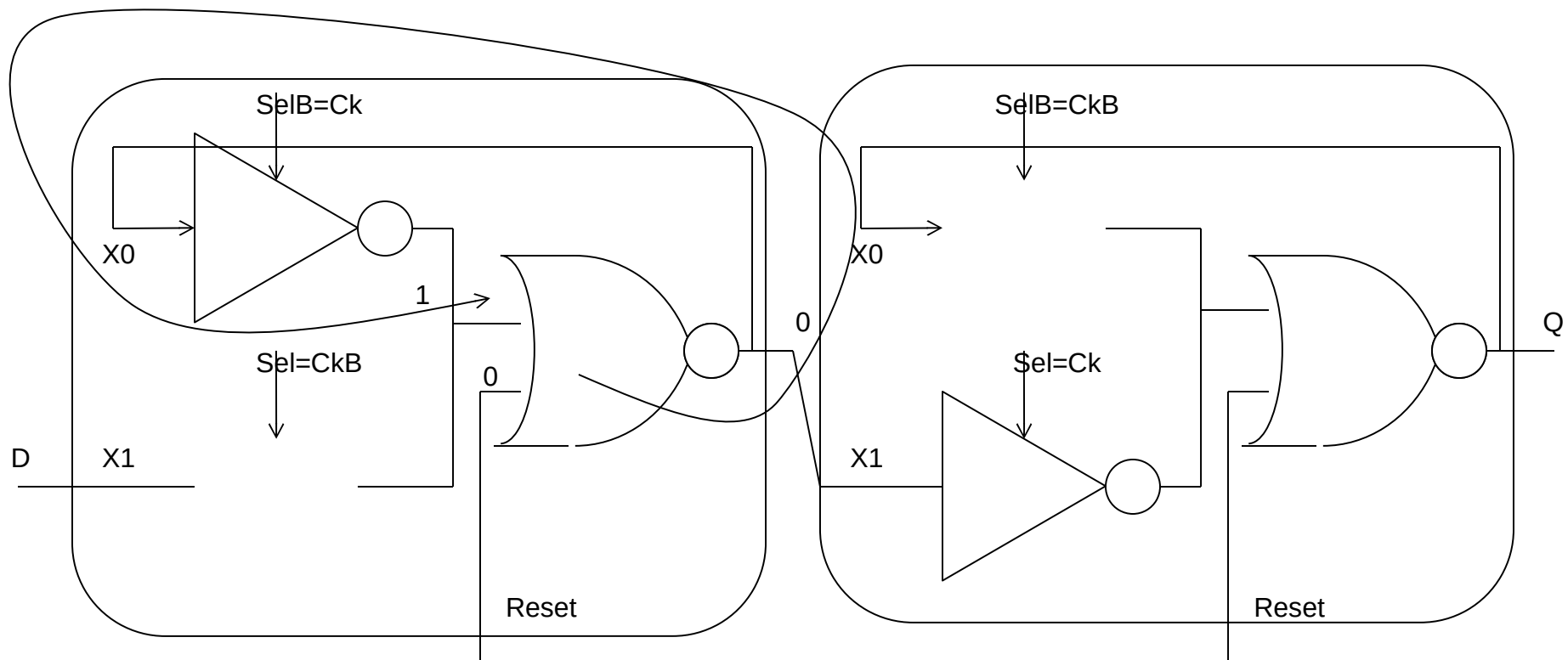
- Nach dem Einschalten der Spannungsversorgung befindet sich ein Flip-Flop, genauso wie eine RAM Zelle, in einem unbekannten logischen Zustand.
- Wir können uns vorstellen, dass sich zuerst alle Flip-Flops in den astabilen Zustand befinden und dann in logisch Eins oder Null Zustand kommen.
- Um einen unbekannten Anfangszustand zu vermeiden, werden die Flip-Flops oft so erweitert, dass sie ein asynchrones Reset Signal haben.



- Betrachten wir einen solch erweiterten FF im $Ck=0$ Zustand
- In dem fall ist das erste Latch im Speichermodus
- $Reset = 1$ (aktiv high) erzwingt logische Null am Ausgang, sie wird rückgekoppelt. (Eins kommt an den zweiten NOR Eingang)
- Auf diese Weise bleibt Null gespeichert auch wenn Reset wieder inaktiv (null) wird.



- Betrachten wir einen solch erweiterten FF im $Ck=0$ Zustand
- In dem fall ist das erste Latch im Speichermodus
- $Reset = 1$ (aktiv high) erzwingt logische Null am Ausgang, sie wird rückgekoppelt. (Eins kommt an den zweiten NOR Eingang)
- Auf diese Weise bleibt Null gespeichert auch wenn Reset wieder inaktiv (null) wird.



- Im Flip-Flop ist immer wenigstens ein Latch im Speicherzustand, so dass ein Reset immer möglich ist wenn beide Latches die Reset Logik enthalten.
- Asynchron Reset ist *stärker* als der Takteingang. Sobald Reset = 1 wird, wird der Flip-Flop Ausgang null, unabhängig von D und Ck Eingängen

