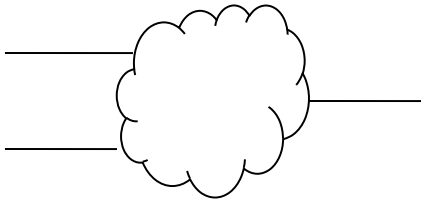
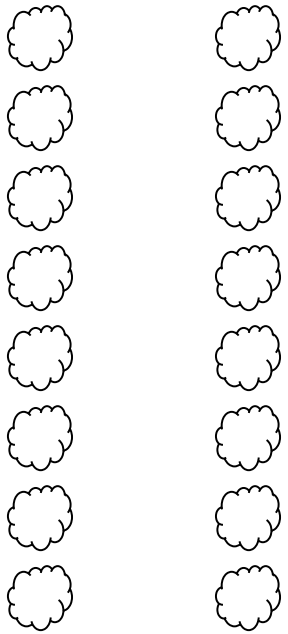
Design digitaler Schaltkreise
Vorlesung 3

- Komplexere CMOS Digitalzellen
- Kombinatorische Logik
- NAND, NOR…
- Multiplexer, Dekoder…
- Sequentielle Logik
- Latches, Flip-Flops

- 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

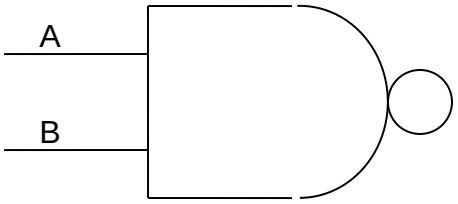


a	b	y
0	0	0/1
0	1	0/1
1	0	0/1
1	1	0/1

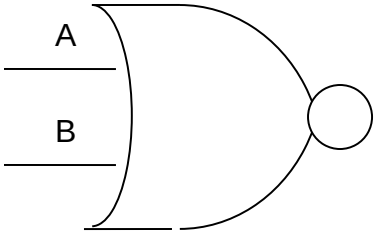


- Die wichtigsten Booleschen Funktionen mit zwei Variablen sind NAND, NOR, EXNOR (Gleichwertigkeit, Äquivalenz)

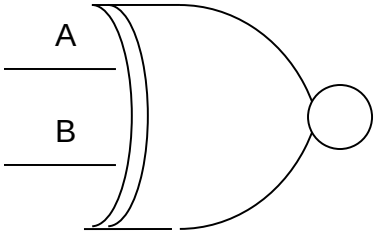
Y = !(A & B)



Y = !(A | B)



Y = !(A ^ B)



NAND

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

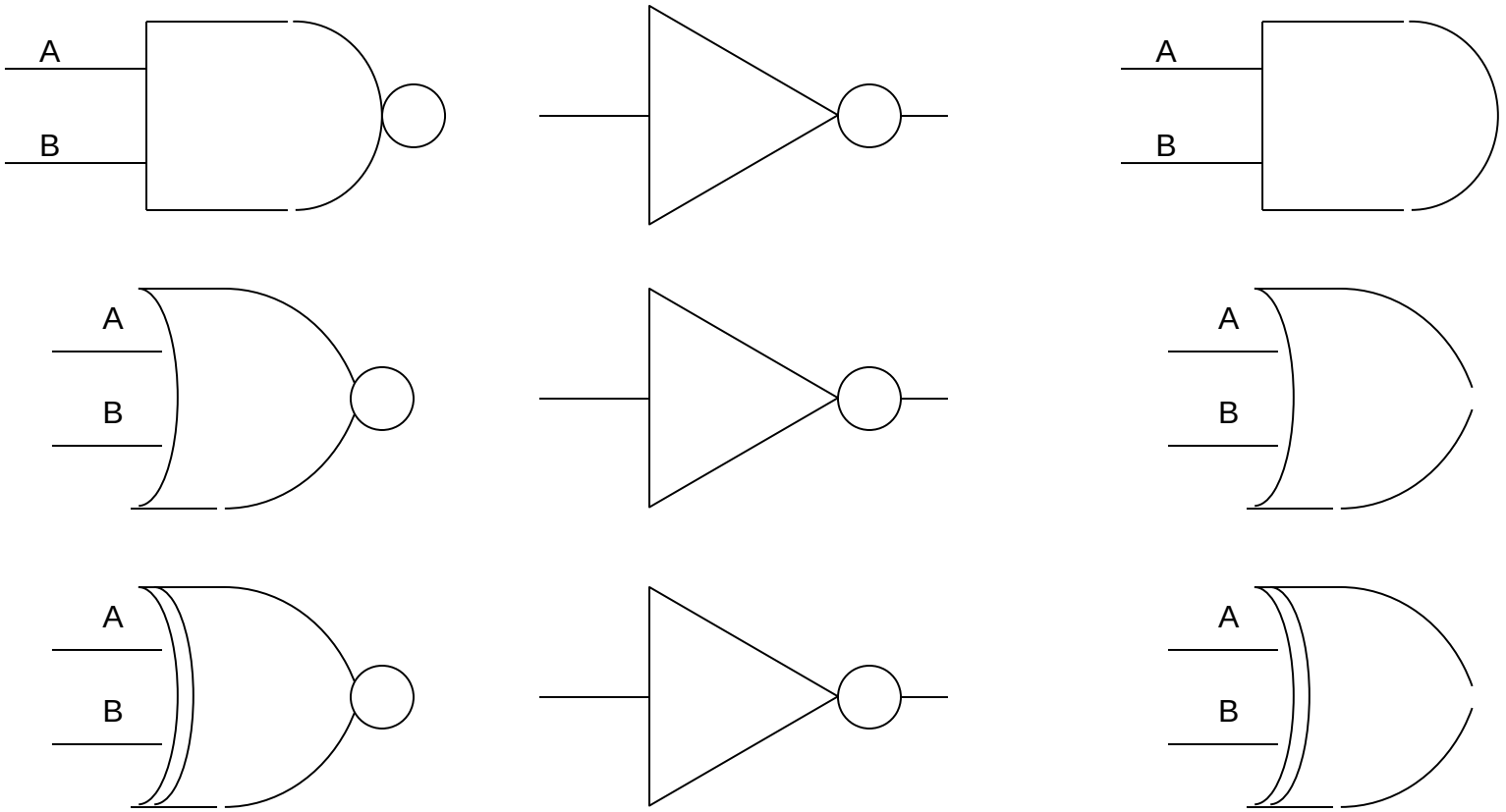
NOR

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

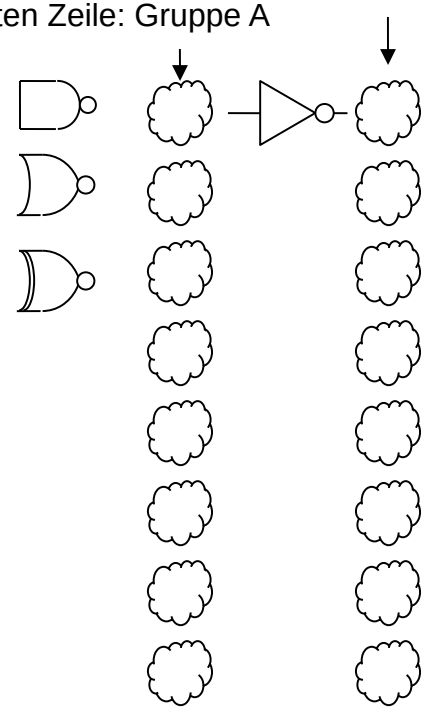
- Mithilfe des Inverters, können wir aus NAND, NOR und EXNOR…
- AND, OR und die EXOR machen



- Sind NAND, NOR, EXNOR und Inverter ausreichend um alle Funktionen 2 Variablen darzustellen?
- 8 Booleschen Funktionen kann man durch Negation aus anderen 8 bekommen - wie AND aus NAND

Haben Y=0 in der ersten Zeile: Gruppe B

Haben Y=1 in der ersten Zeile: Gruppe A



- Zwei Funktionen (von der Gruppe A) 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

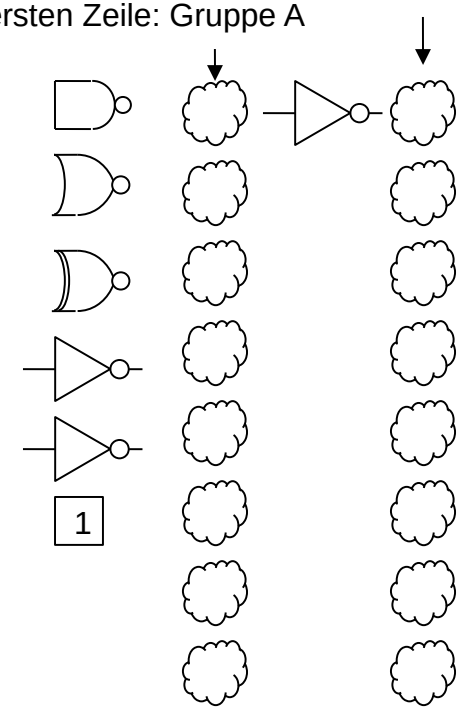
a	b	y
0	0	1
0	1	0
1	0	1
1	1	0

1

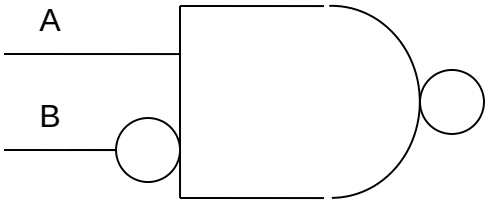
a	b	y
0	0	1
0	1	1
1	0	1
1	1	1

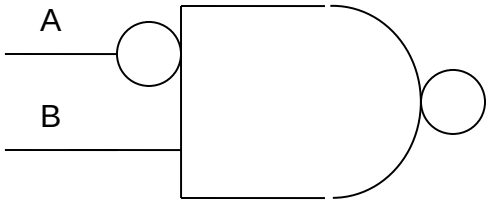
Haben Y=0 in der ersten Zeile: Gruppe B

Haben Y=1 in der ersten Zeile: Gruppe A



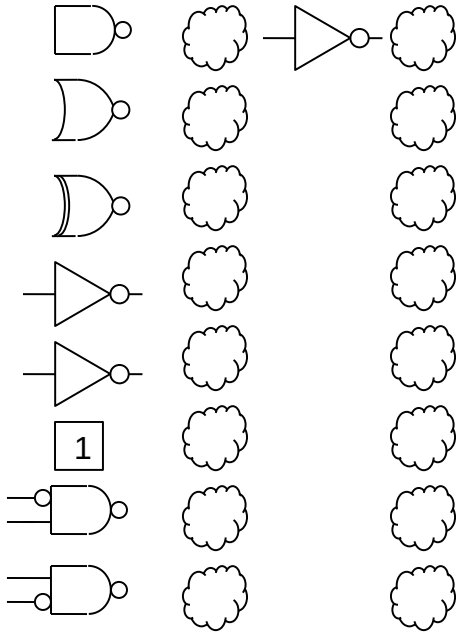
- Zwei Funktionen können mit NAND mit invertierten Eingängen realisiert werden
- => NAND, NOR, EXNOR und Inverter sind ausreichend um alle Funktionen von 2 Variablen darzustellen…



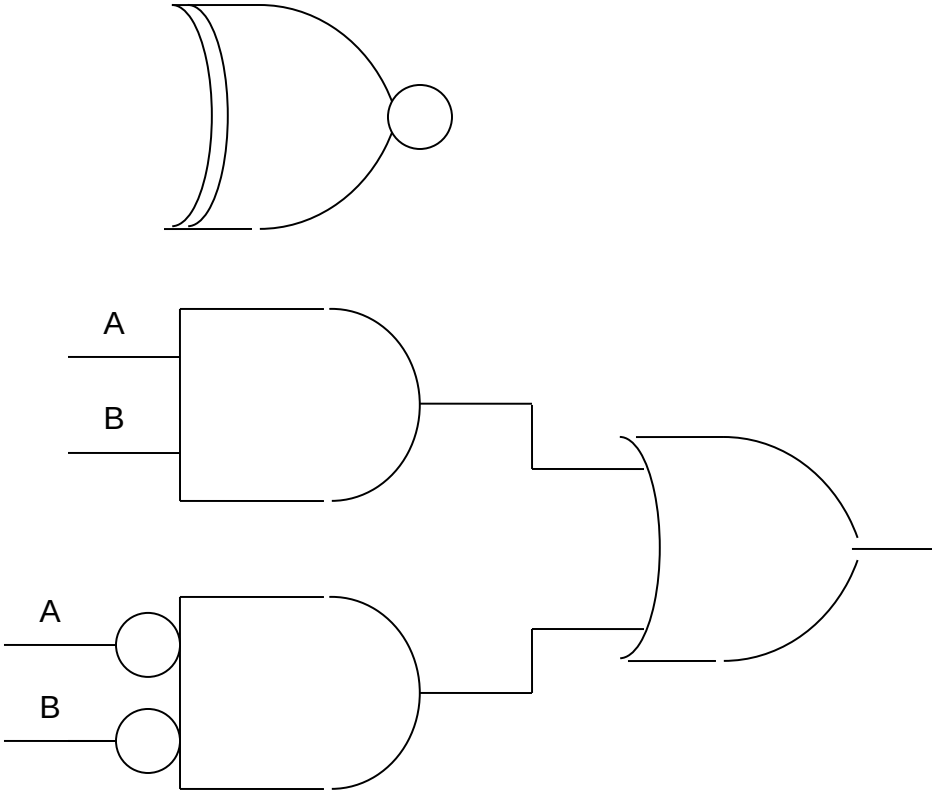


a	b	y
0	0	1
0	1	1
1	0	0
1	1	1

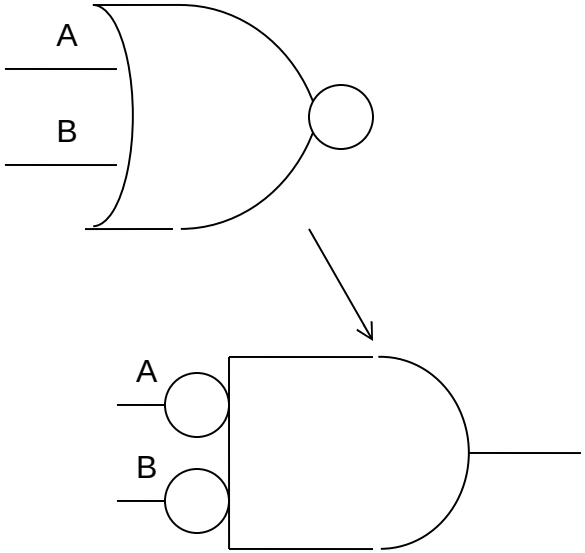
a	b	y
0	0	1
0	1	0
1	0	1
1	1	1



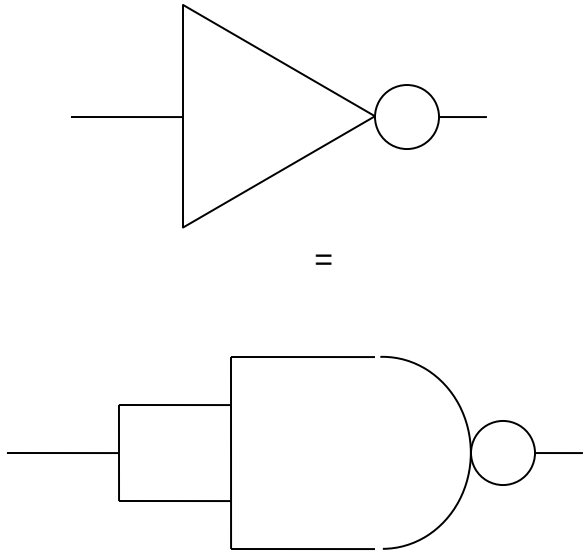
- Es geht auch einfacher…
- EXNOR kann mit AND, OR und Inverter realisiert werden



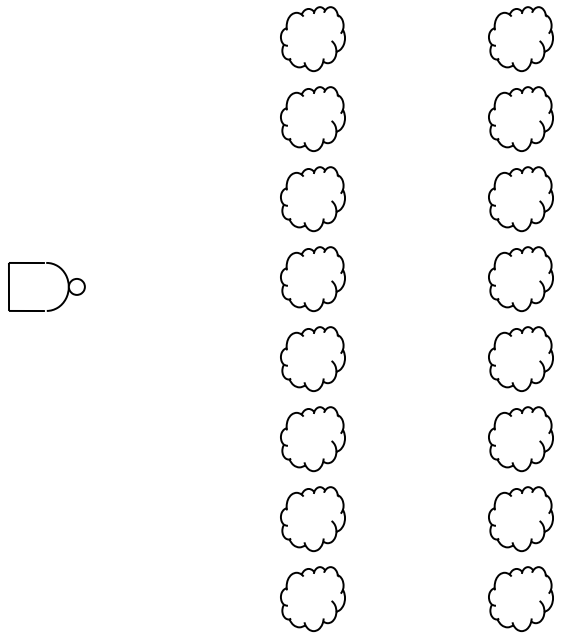
- NOR kann man in NAND umwandeln



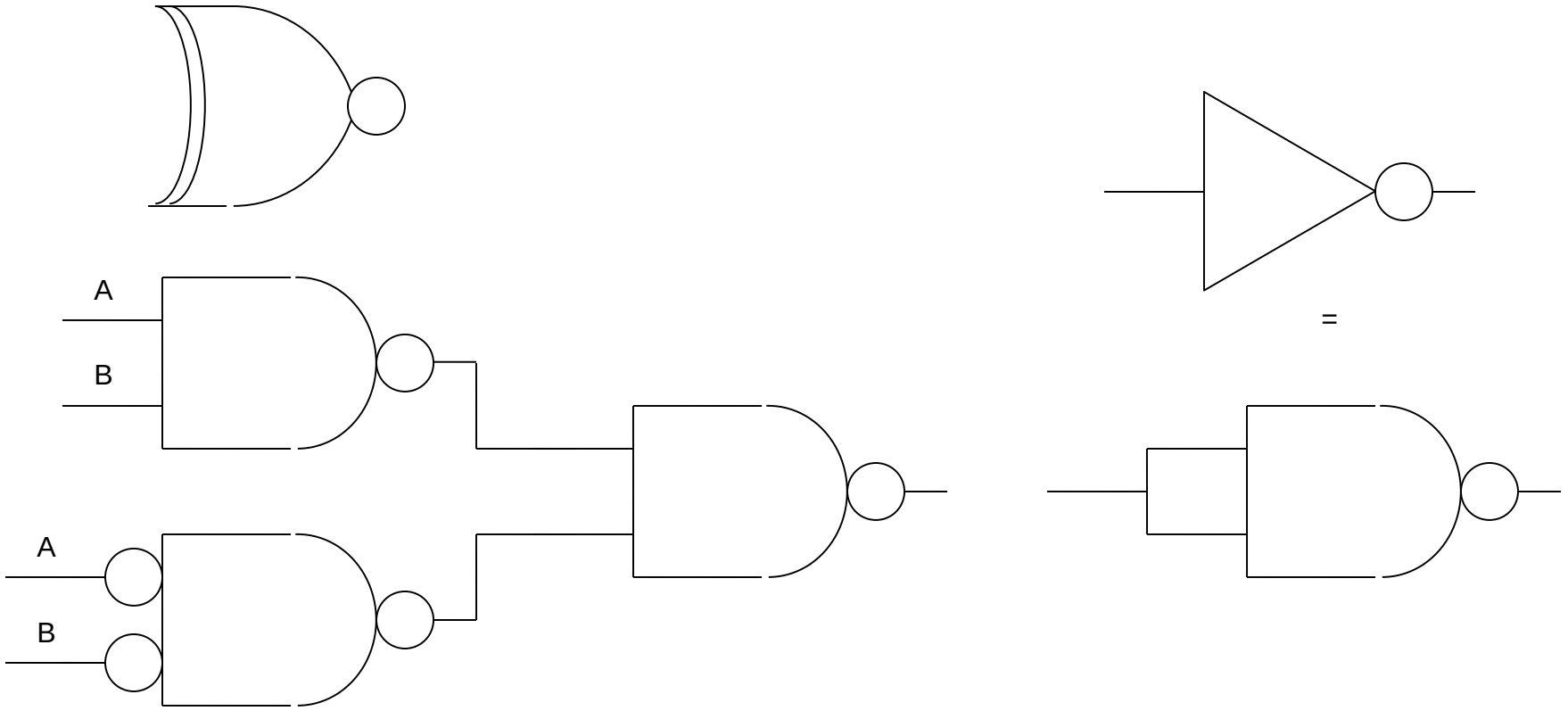
- Inverter kann man mit NAND realisieren



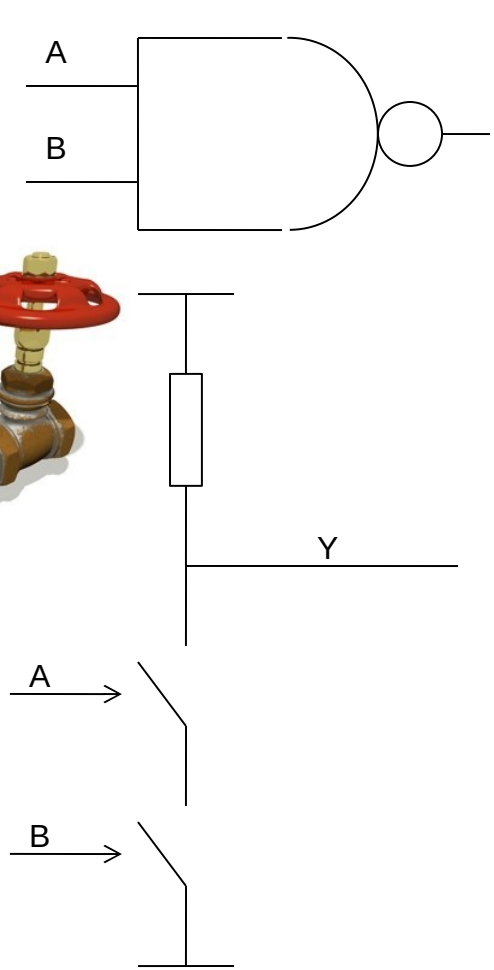
- Alle Funktionen können basierend auf NAND realisiert werden

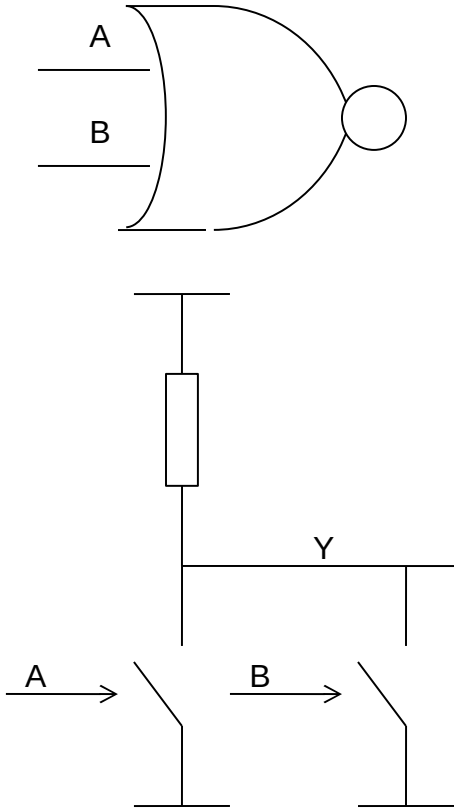


- Alle Funktionen können basierend auf NAND realisiert werden

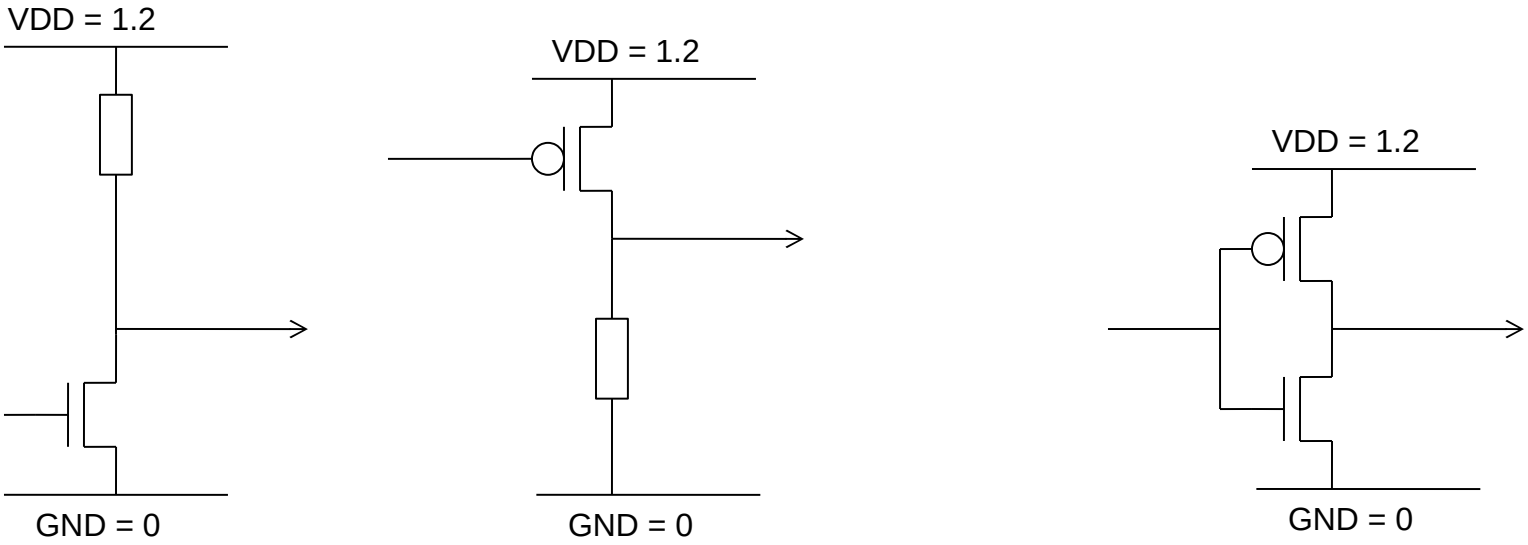


- NAND und NOR als Schalter-Widerstand Logik

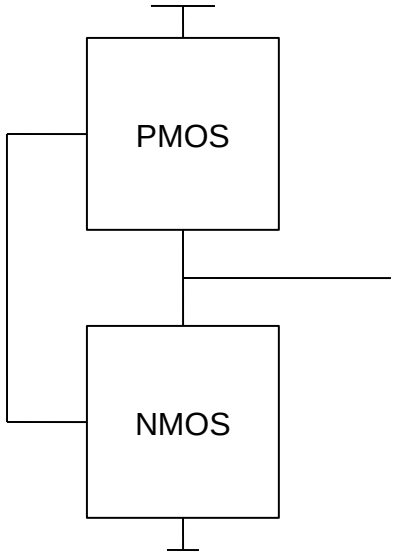




- Inverter als RTL Logik
- NMOS und Pullup oder mit PMOS und Pulldown Widerstand
- -> CMOS Inverter
- Vorteile sind kein DC Strom und ein kleines Layout.



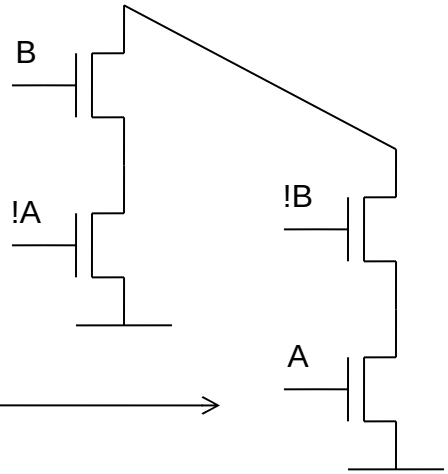
- NAND, NOR und co. als CMOS
- Wie wird ein CMOS Gate gemacht?
- Es gibt zwei Blöcke: NMOS und PMOS
- Wenn NMOS Teil leitet, soll PMOS Teil nicht leiten, und umgekehrt
- Kein Kurzschluss VDD-GND, oder floating-Ausgang

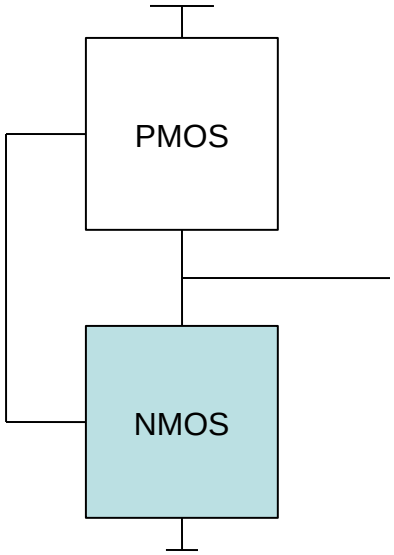


- Wie machen wir einen NMOS Block?
- Jede Zeile mit dem Ergebnis 0 -> Serienschaltung von zwei (oder mehreren) NMOS Transistoren die nur für die Eingangswerte dieser Zeile leiten
- Man muss alle Eingänge = null invertieren.
- Das ganze NMOS Netzwerk ist die Parallelschaltung aller Reihenschaltungen, die Zeilen = 0 entsprechen.

EXNOR

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

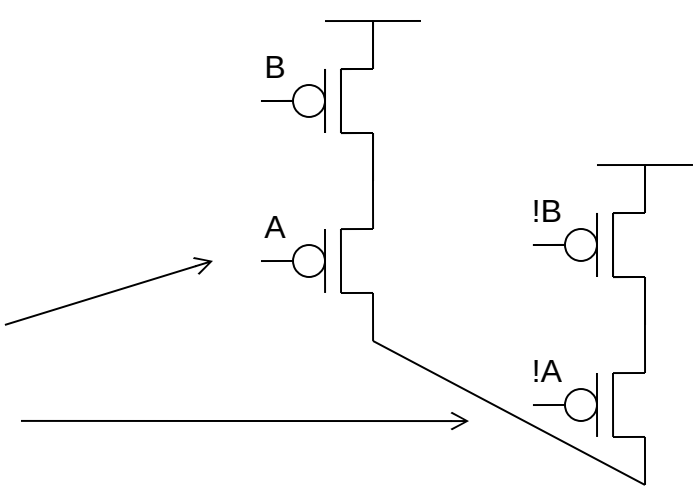


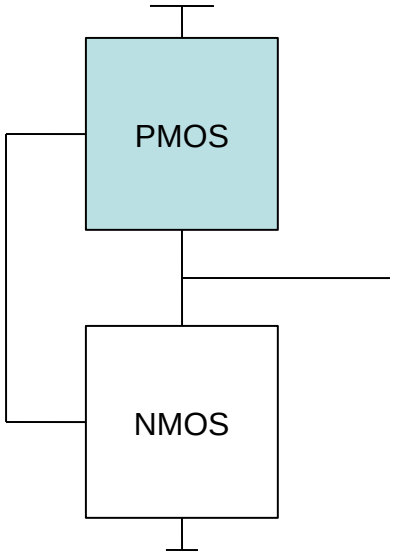


- PMOS Teil macht man dual
- Beachten wir, dass PMOS für niedriges Gate-Potential leitet
- Man muss alle Eingänge = eins invertieren.

EXNOR

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

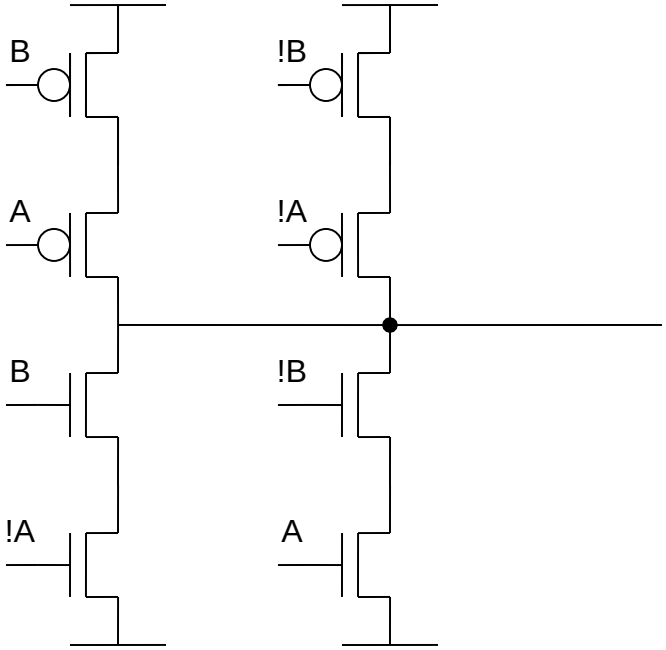




- Beispiel: EXNOR

EXNOR

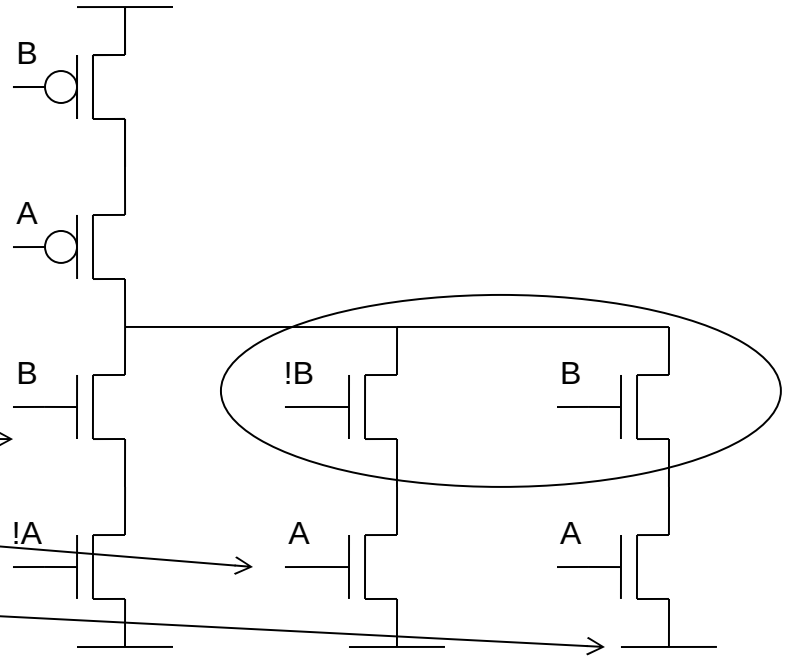
a	b	y
0	0	1
0	1	0
1	0	0
1	1	1



- Oft kann man die logische Funktion vereinfachen
- Beispiel: NOR

NOR

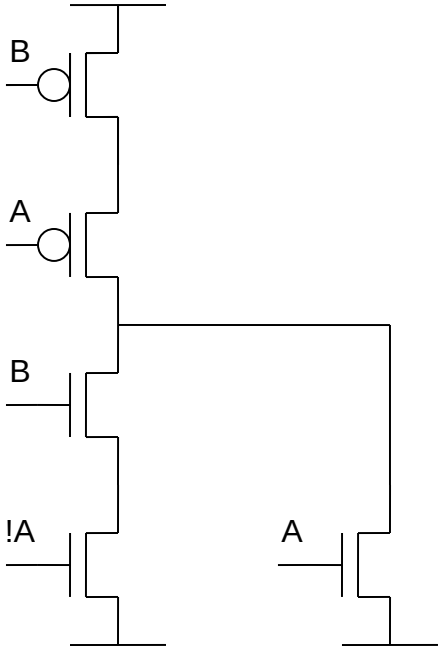
a	b	y
0	0	1
0	1	0
1	0	0
1	1	0



- Oft kann man die logische Funktion vereinfachen
- Beispiel: NOR

NOR

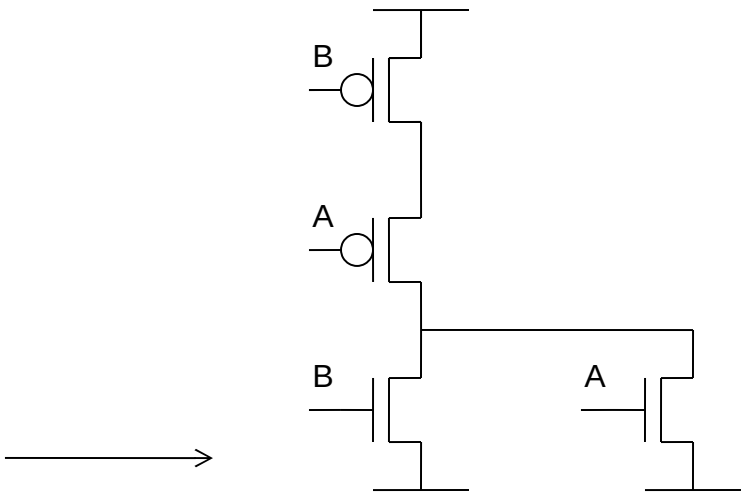
a	b	y
0	0	1
0	1	0
1	0	0
1	1	0



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

NOR

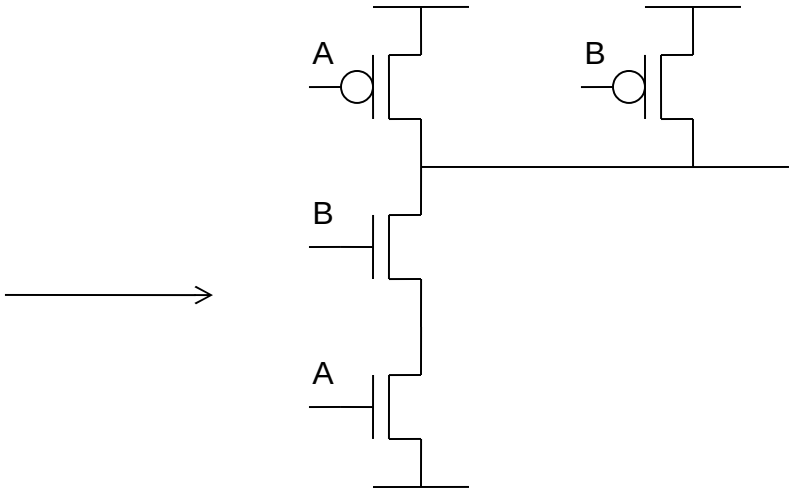
a	b	y
0	0	1
0	1	0
1	0	0
1	1	0



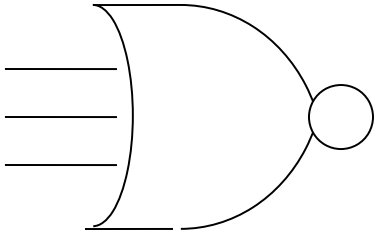
- CMOS NAND

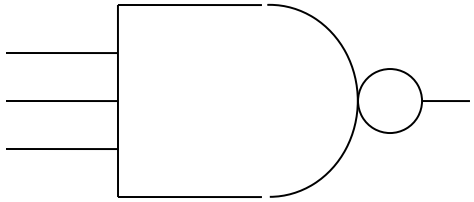
NAND

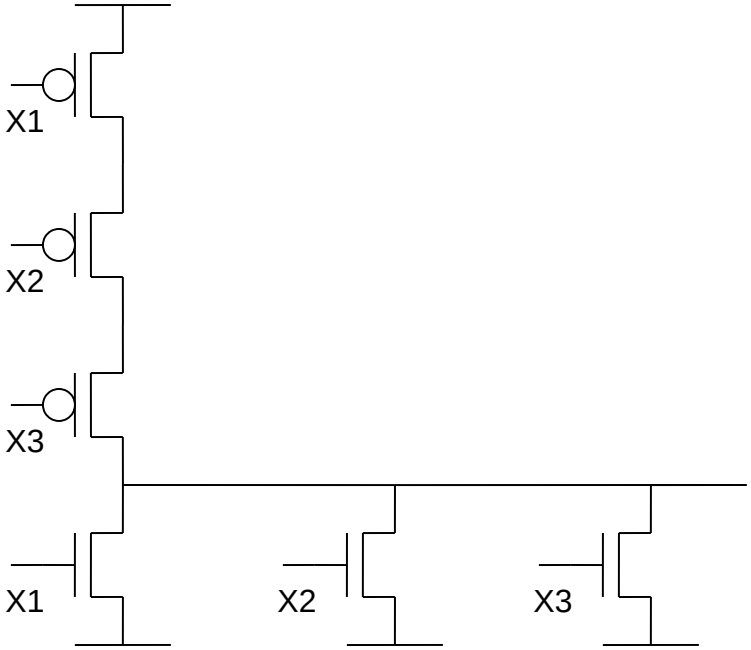
a	b	y
0	0	1
0	1	1
1	0	1
1	1	0

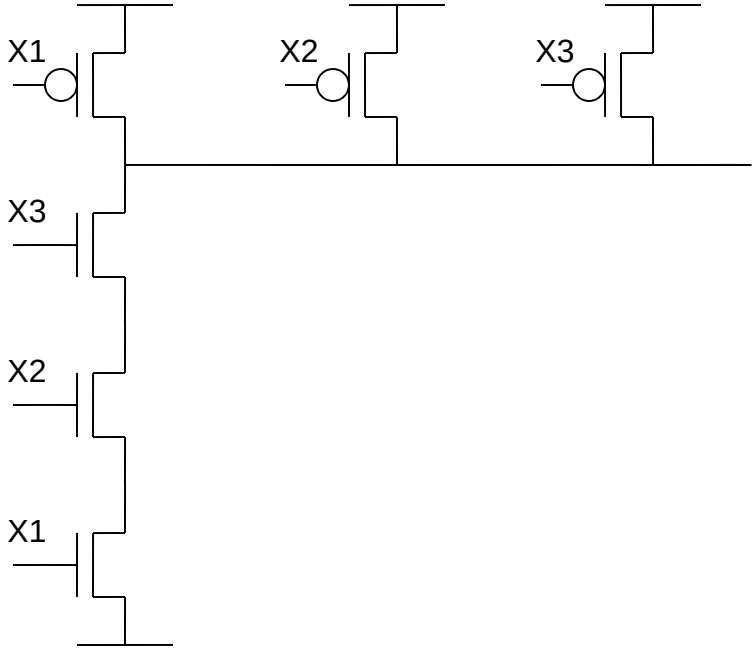


- NAND und NOR mit mehreren Eingängen



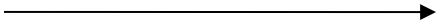


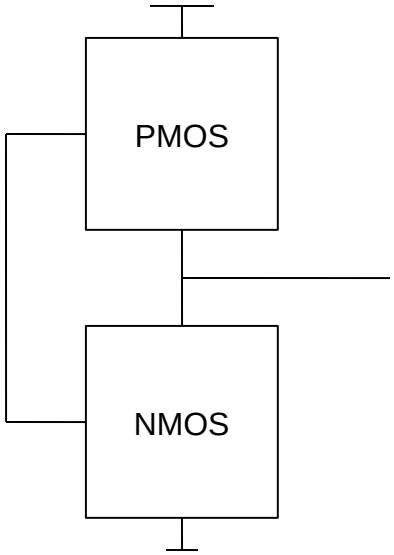




- Einige Regeln für die Konstruktion von CMOS Gattern aus Ergebnistabelle
- NMOS Teil leitet für die Zeilen mit null-Ergebnis
- PMOS Teil leitet für die Zeilen mit eins-Ergebnis
- PMOS und NMOS Teile dürfen nie gleichzeitig leiten, sonst hätten wir einen großen Querstrom und der Ausgang wäre undefiniert
- 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

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

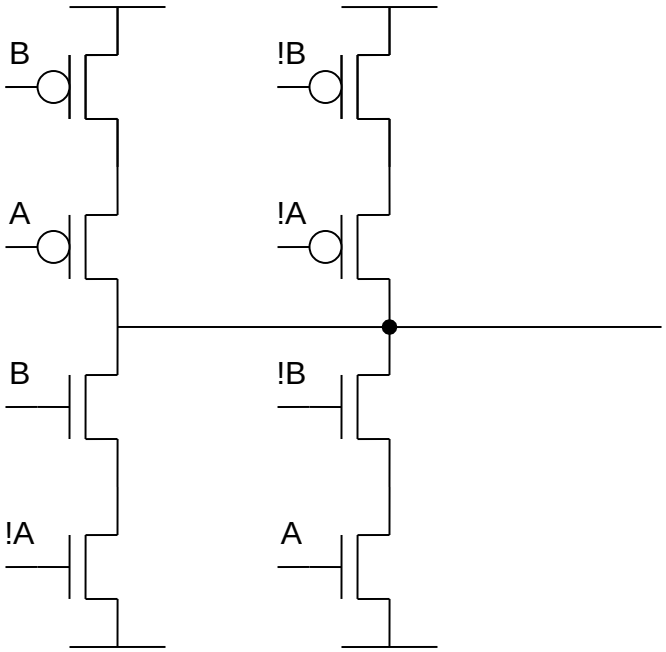




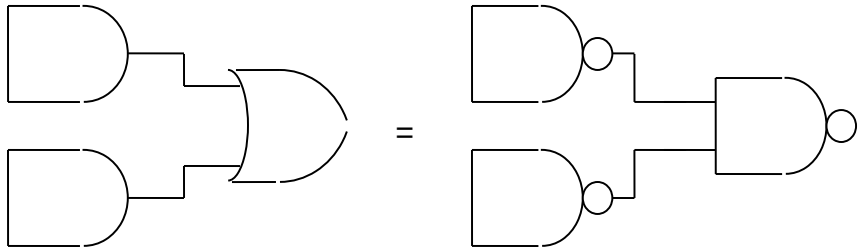
- Es gibt oft mehrere Möglichkeiten eine Funktion mit Transistoren zu realisieren…
- Beispiel EXNOR

Variante 1

8 Transistoren

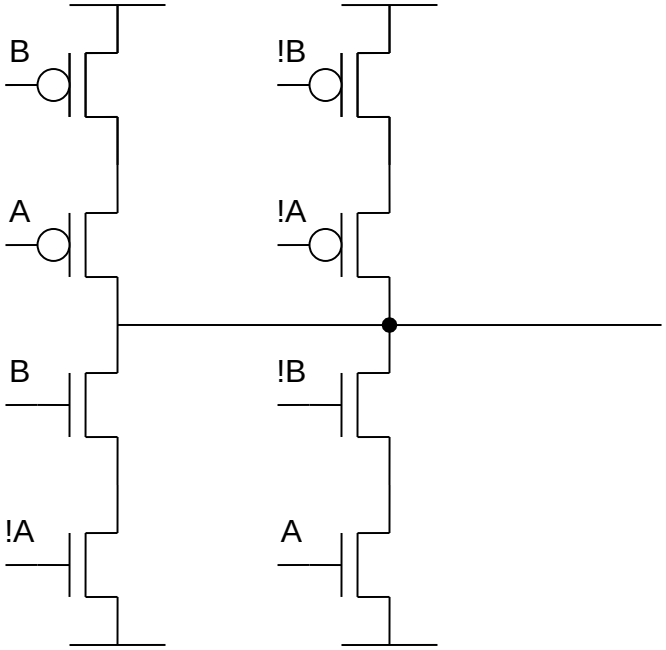


- Variante2: Disjunktive Normalform
- EXOR = (!A & !B) | (A & B)
- V.1 braucht weniger Transistoren



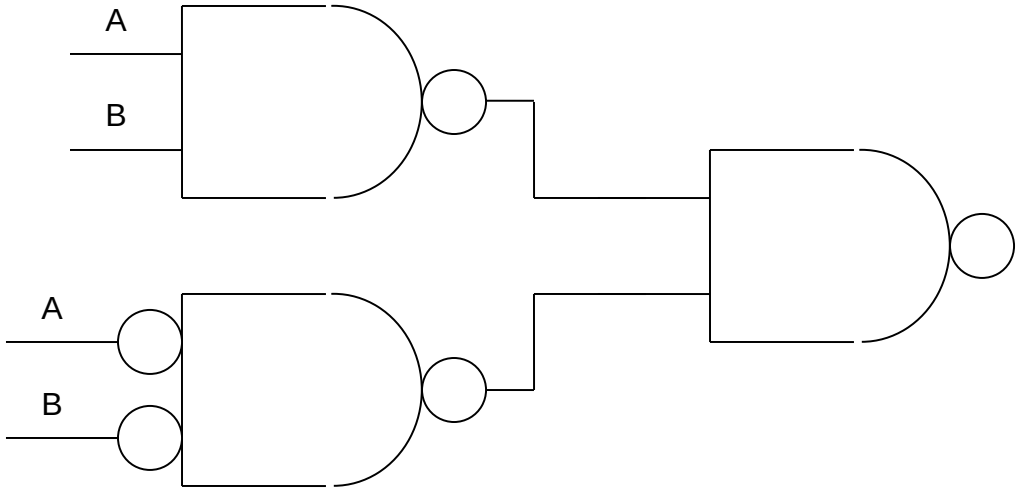
Variante 1

8 Transistoren



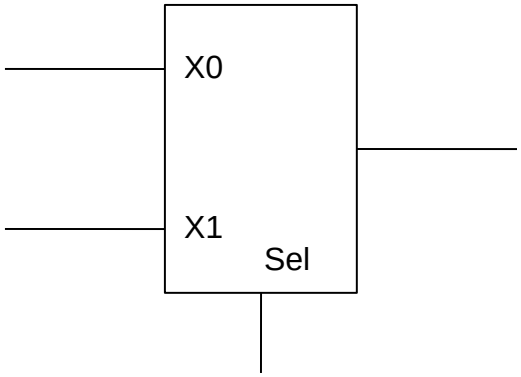
Variante 2

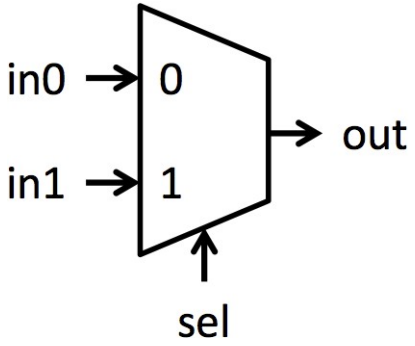
4 (INVs) + 12 (NANDs) Transistoren

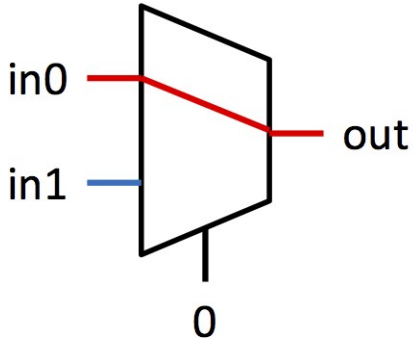


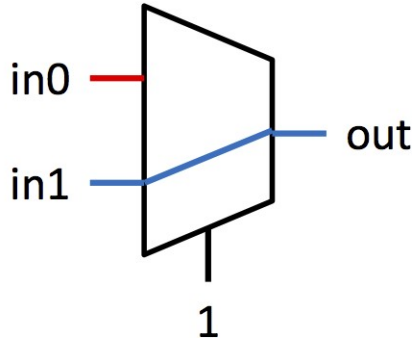
Multiplexer

- Der wichtigste und vielseitigste Bauteil in Digitaltechnik ist der Multiplexer
- Je nachdem ob der Select-Eingang null oder eins ist, ist der Ausgang X0 oder X1
- Im Verilog Code: Y = sel ? X1 : X0



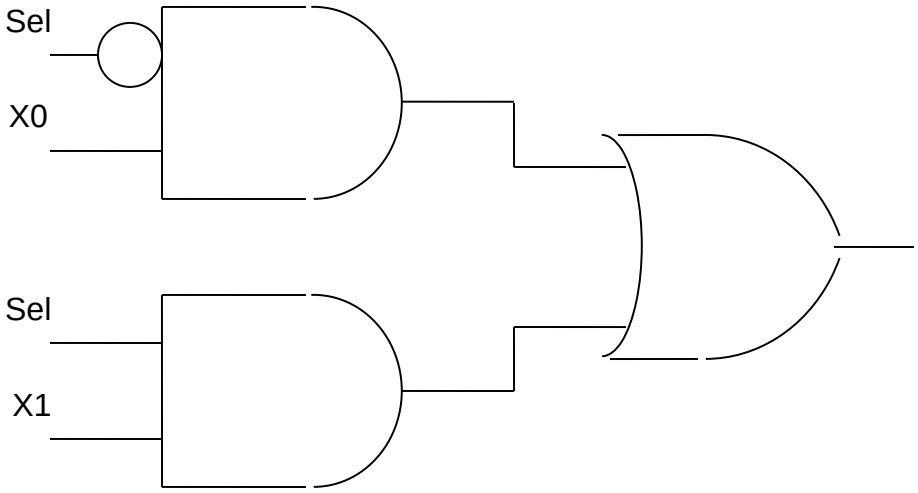






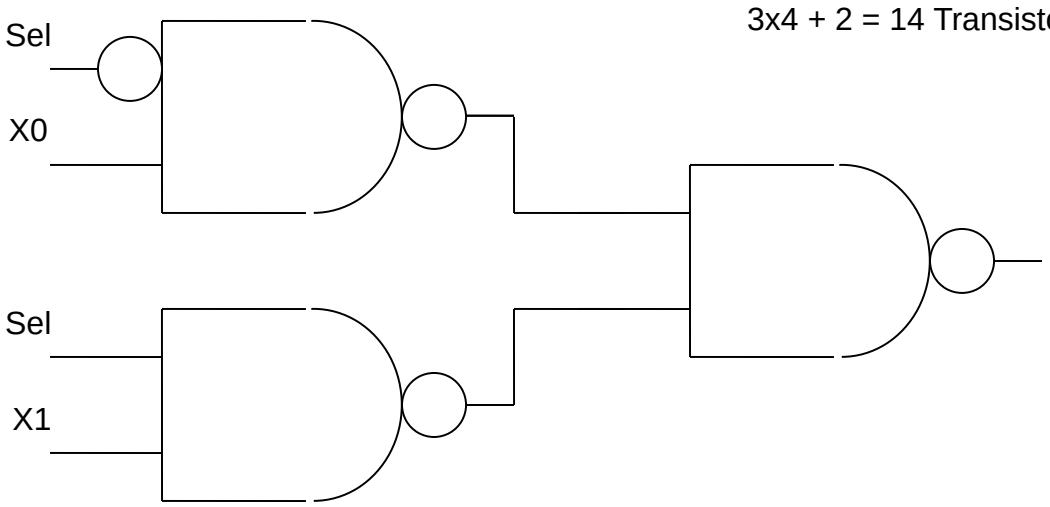
Multiplexer

- Der wichtigste und vielseitigste Bauteil in Digitaltechnik ist der Multiplexer
- Je nachdem ob der Select-Eingang null oder eins ist, ist der Ausgang X0 oder X1
- Im Verilog Code: Y = sel ? X1 : X0
- Disjunktive Normalform: Y = (!sel & X0) | (sel & X1)



Multiplexer

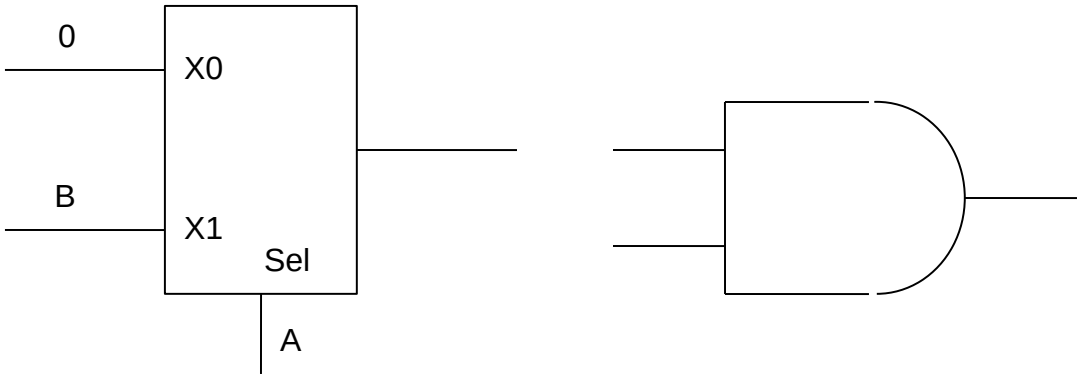
- Der wichtigste und vielseitigste Bauteil in Digitaltechnik ist der Multiplexer
- Je nachdem ob der Select-Eingang null oder eins ist, ist der Ausgang X0 oder X1
- Im Verilog Code: Y = sel ? X1 : X0
- Disjunktive Normalform: Y = (!sel & X0) | (sel & X1)



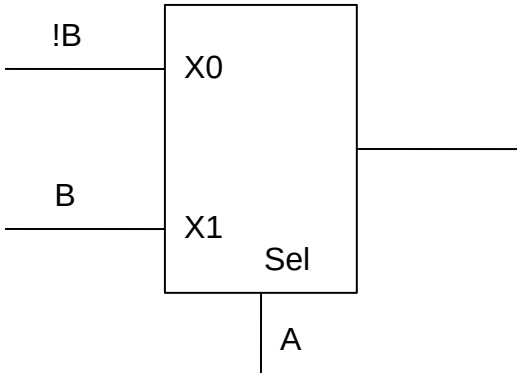
3x4 + 2 = 14 Transistoren

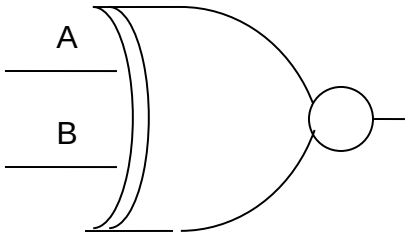
- Warum ist ein Multiplexer so wichtig?
- Jede logische Funktion kann mit Multiplexern, Invertern und logischen Konstanten realisiert werden

- Beispiel: AND
- AND ist null wenn die Variable A null ist, unabhängig von B
- AND = A ? B : 0

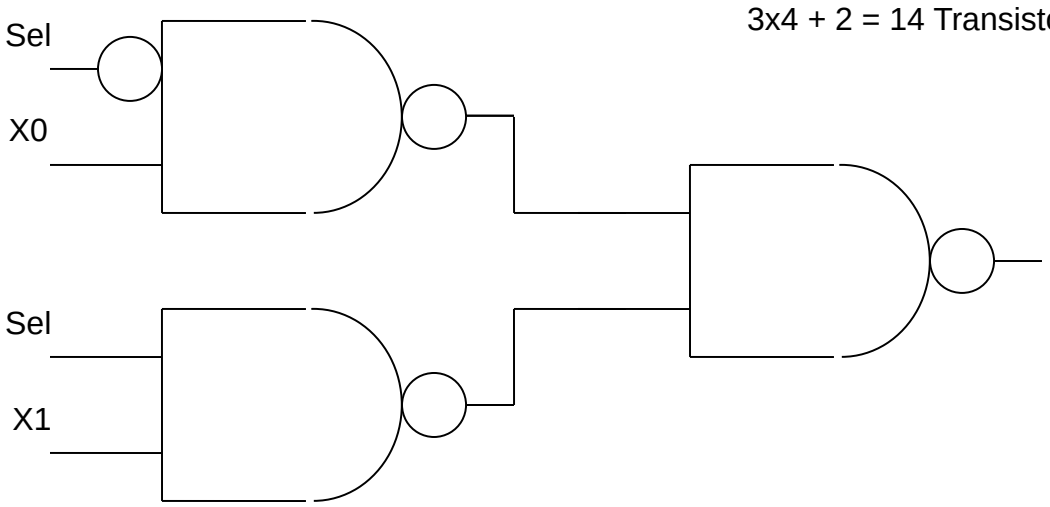


- Beispiel EXNOR
- EXNOR = A ? B : !B



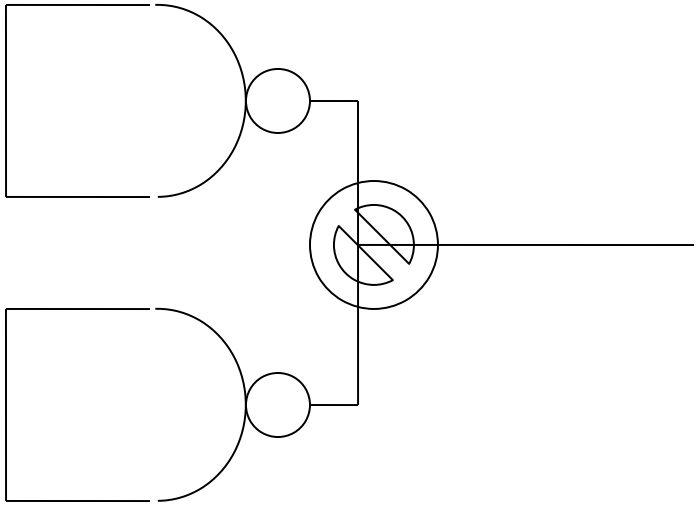


- Multiplexer kann auch einfacher (als unten) realisiert werden.

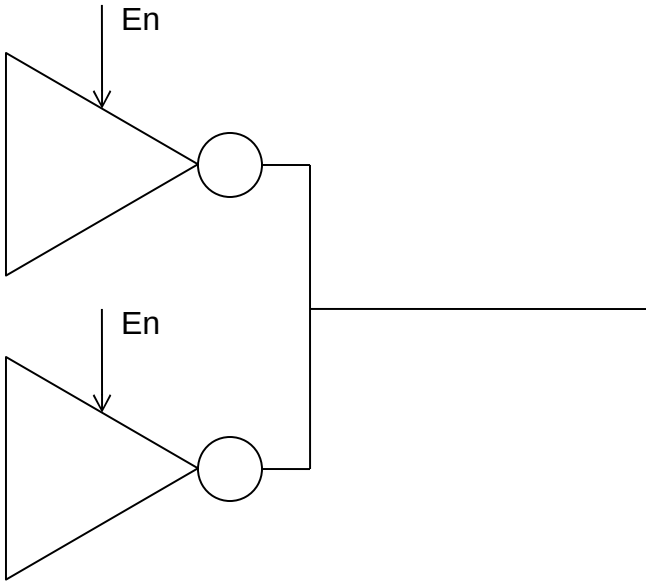


3x4 + 2 = 14 Transistoren

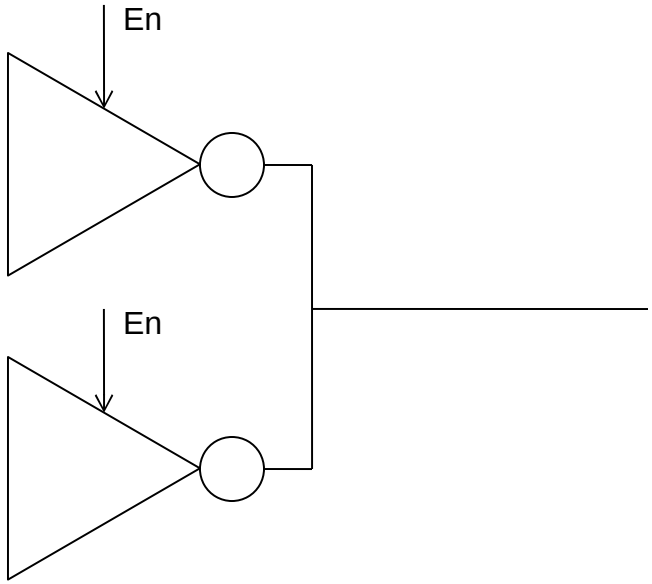
- Normalerweise kann man die Gates nicht kurzschließen…



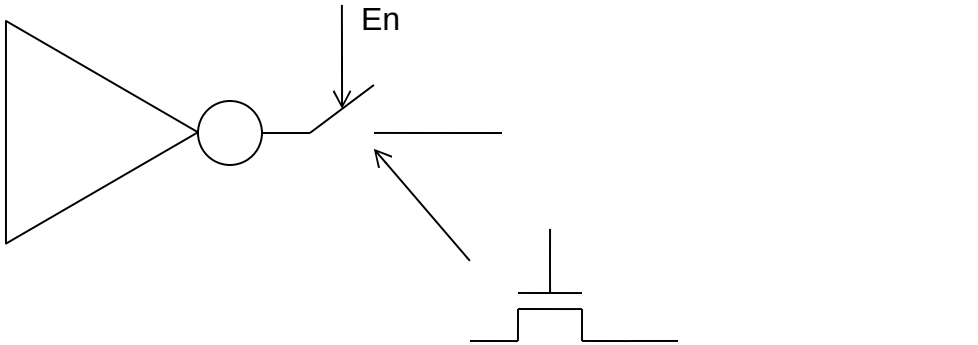
- Wir können die Gatter so erweitern, dass sie sich im hochohmigen Zustand befinden können
- -> Gated Inverter
- Wenn der Enable Eingang eins ist, funktioniert der Inverter wie ein gewöhnlicher Inverter



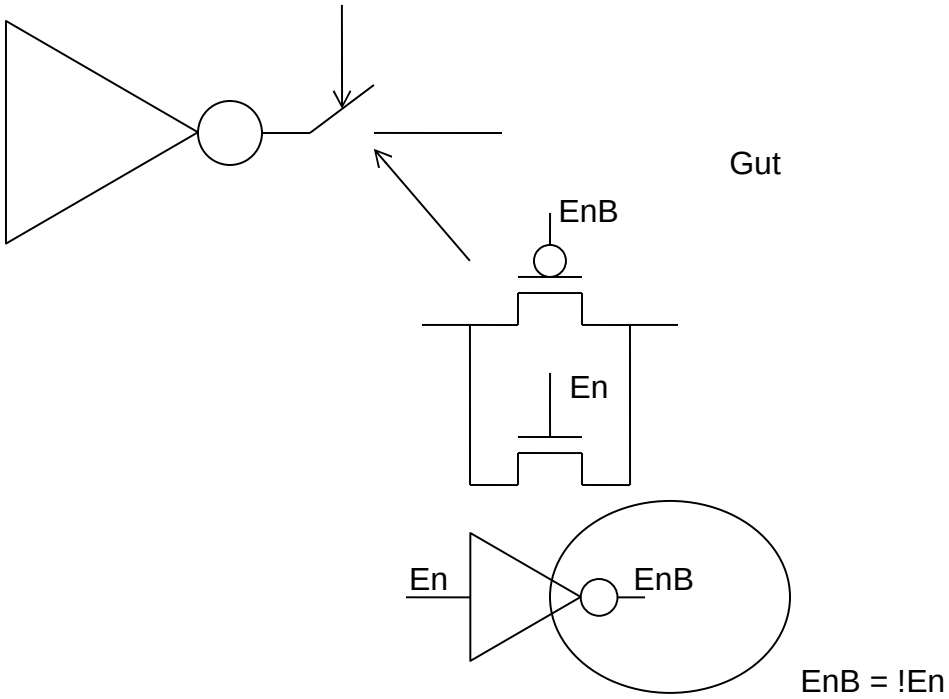
- Mit Enable = null, ist der Ausgang von VDD und GND getrennt, der Ausgang „schwebt“ (float) im hochohmigen (high impedance) Zustand



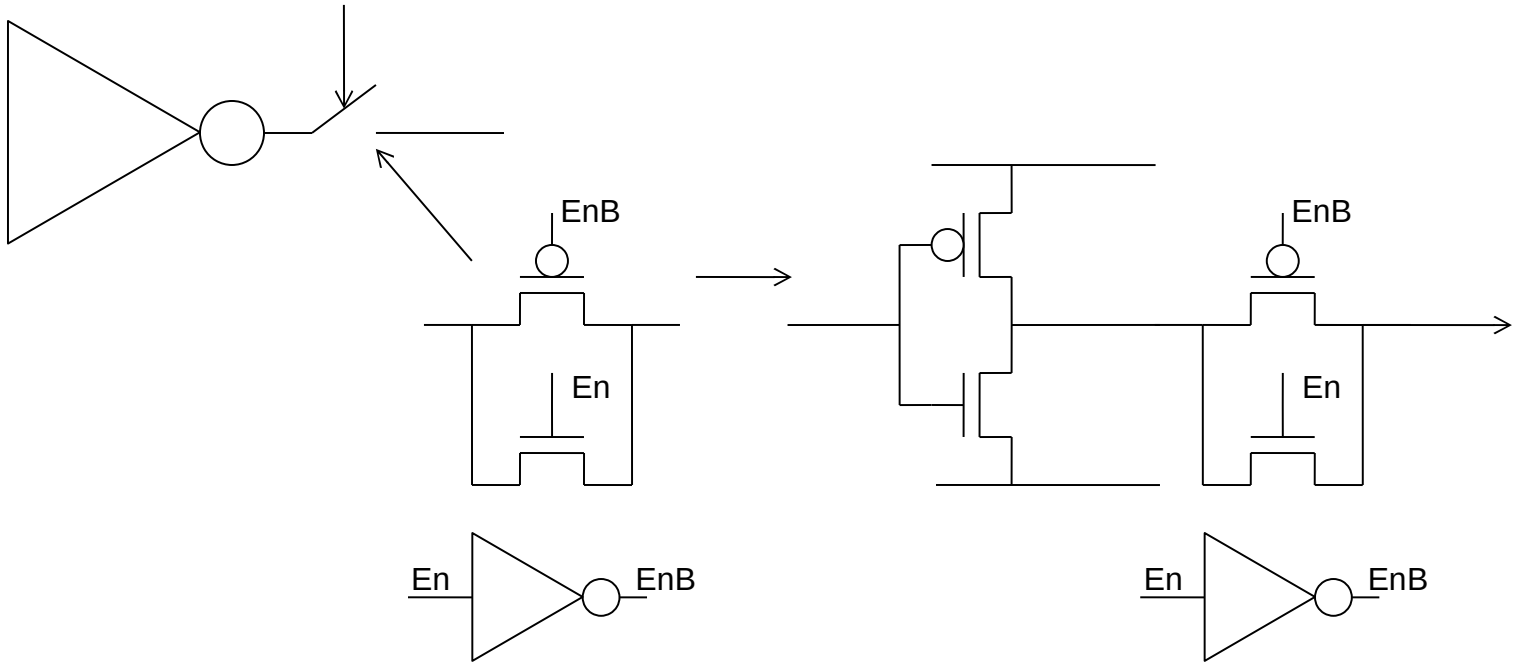
- Schaltung



- Schaltung

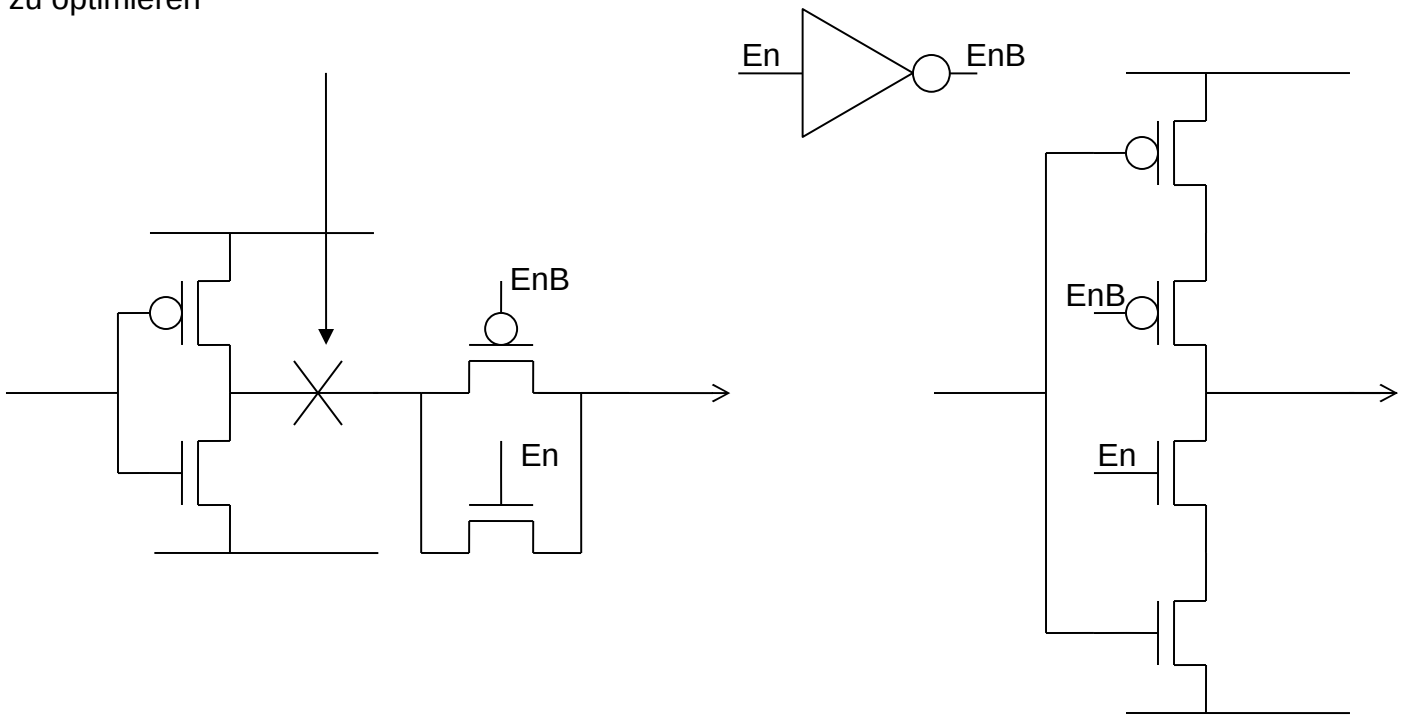


- Schaltung

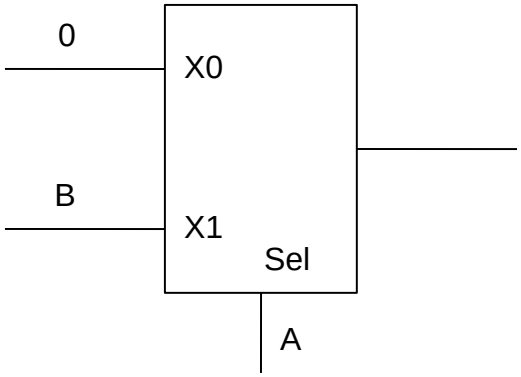


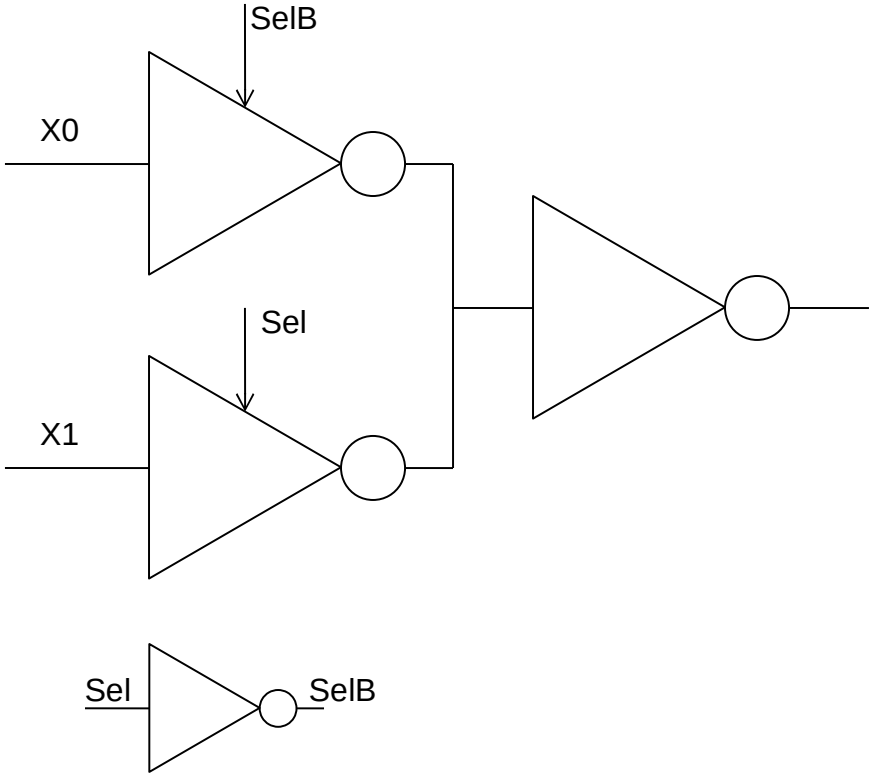
- Schaltung

Leitung wird aufgespalten um Geschwindigkeit zu optimieren



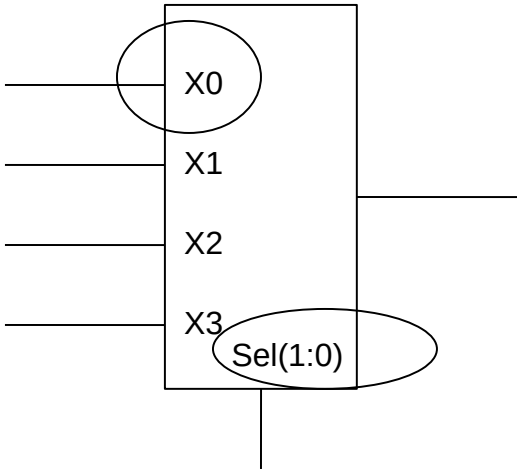
- Multiplexer mit Gated Invertern
- Wir brauchen zwei normale- und zwei Gated Inverter
- Zahl von Transistoren: 2 x 2(Inv) + 2(Inv) x 4(Gated Inv) = 12 Transistoren
- Weniger T. als im Fall von DNF-Implementierung (14 Transistoren)





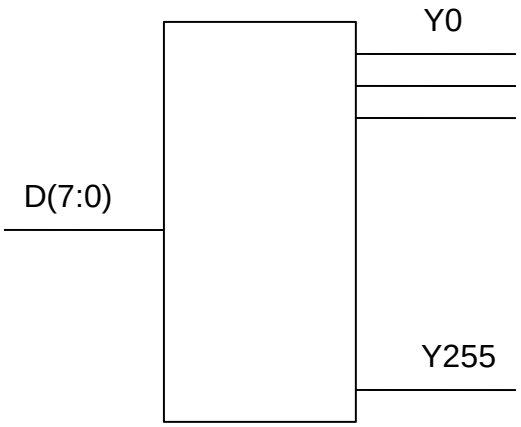
- Oft werden auch mehrfache Multiplexer verwendet.
- ZB wenn man die digitalen Signale von mehreren Quellen über eine Leitung übertragen möchte.
- 2->1 Multiplexer
- 4->1 Multiplexer…

https://stackoverflow.com/questions/50766295/using-verilog-case-statement-with-continuous-assignment



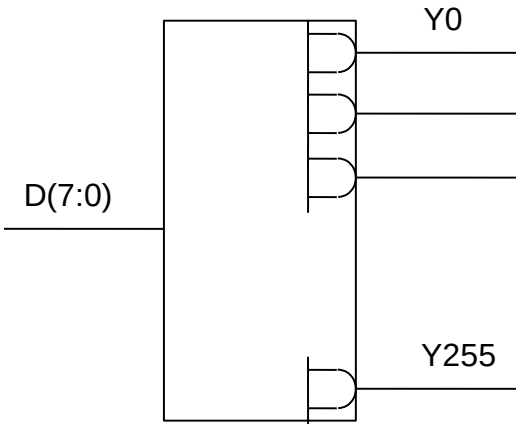

```
reg Y;
wire X0, X1, X2, X3;
wire [1:0] sel;

…begin
        case (sel)
           2'b00 : Y = X0;
           2'b01 : Y = X1;
           2'b10 : Y = X2;
           3'b11 : Y = X3;
        endcase
        end


case (sel)
   2'b00   : Y = in[0];
   2'b01   : Y = in[1];
   2'b10   : Y = in[2];
   3'b11   : Y = in[3];
   default : Y = in[0];
endcase
```

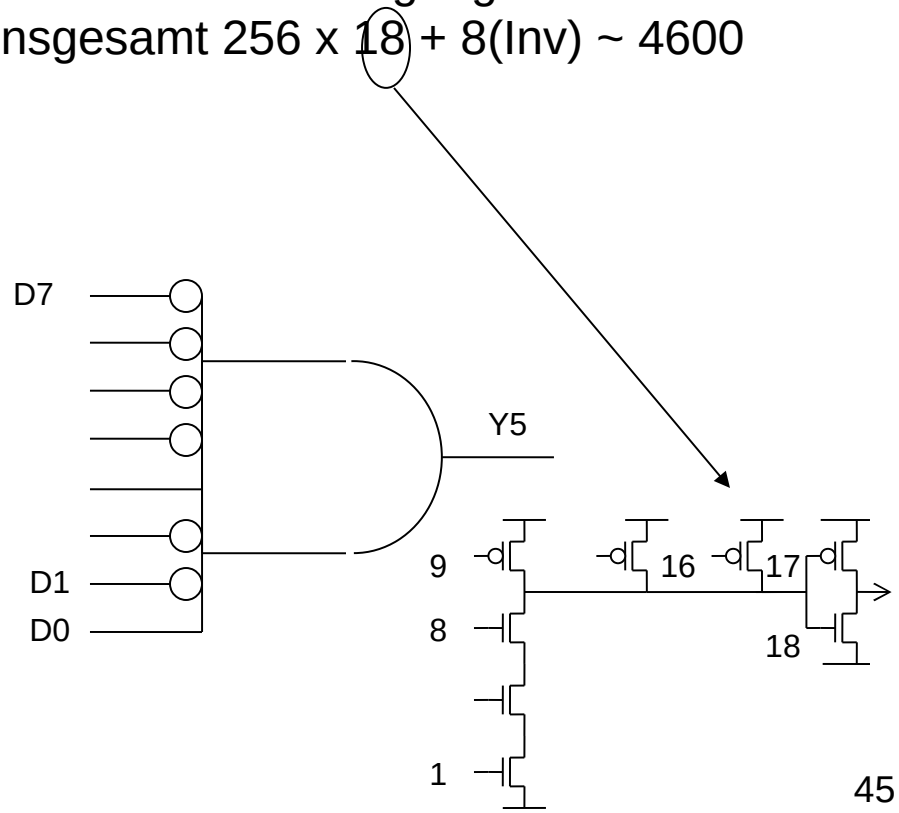
- Um solche Multiplexer zu realisieren brauchen wir einen Decoder.
- Beispiel: 8-bit Eingang D(7:0) (binäre Zahl) und 2^8 Ausgänge.
- Falls Eingang = m (binär Kodiert) ist der m-te Ausgang 1. Alle anderen Ausgänge sind null.



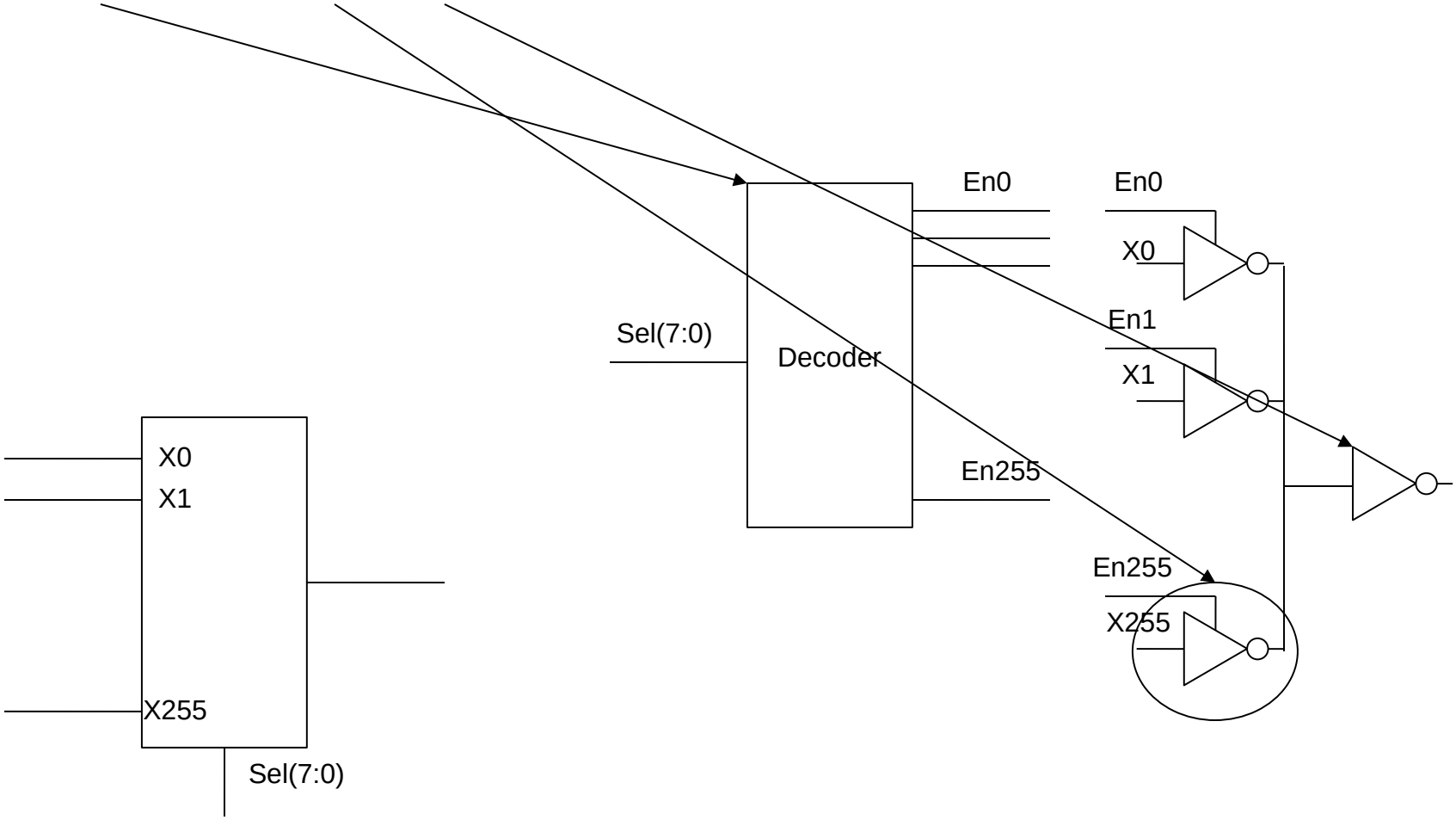

```
Y0 = (D == 8‘d0);
Y1 = (D == 8‘d1);
Y2 = (D == 8’d2);
Y3 = (D == 8’d3);
…
```

- 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 für Di Negationen. Das sind insgesamt 256 x 18 + 8(Inv) ~ 4600 Transistoren.

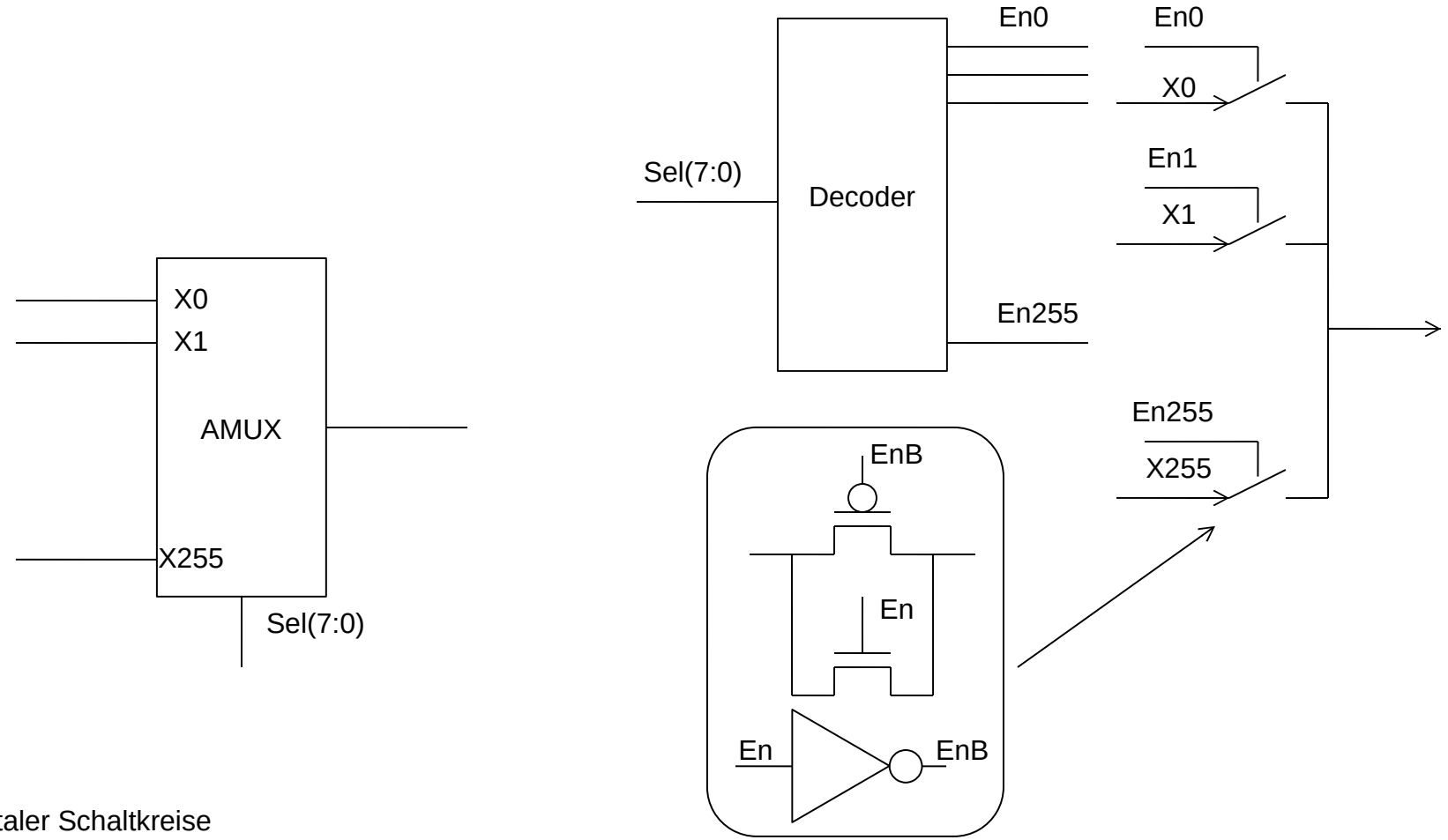




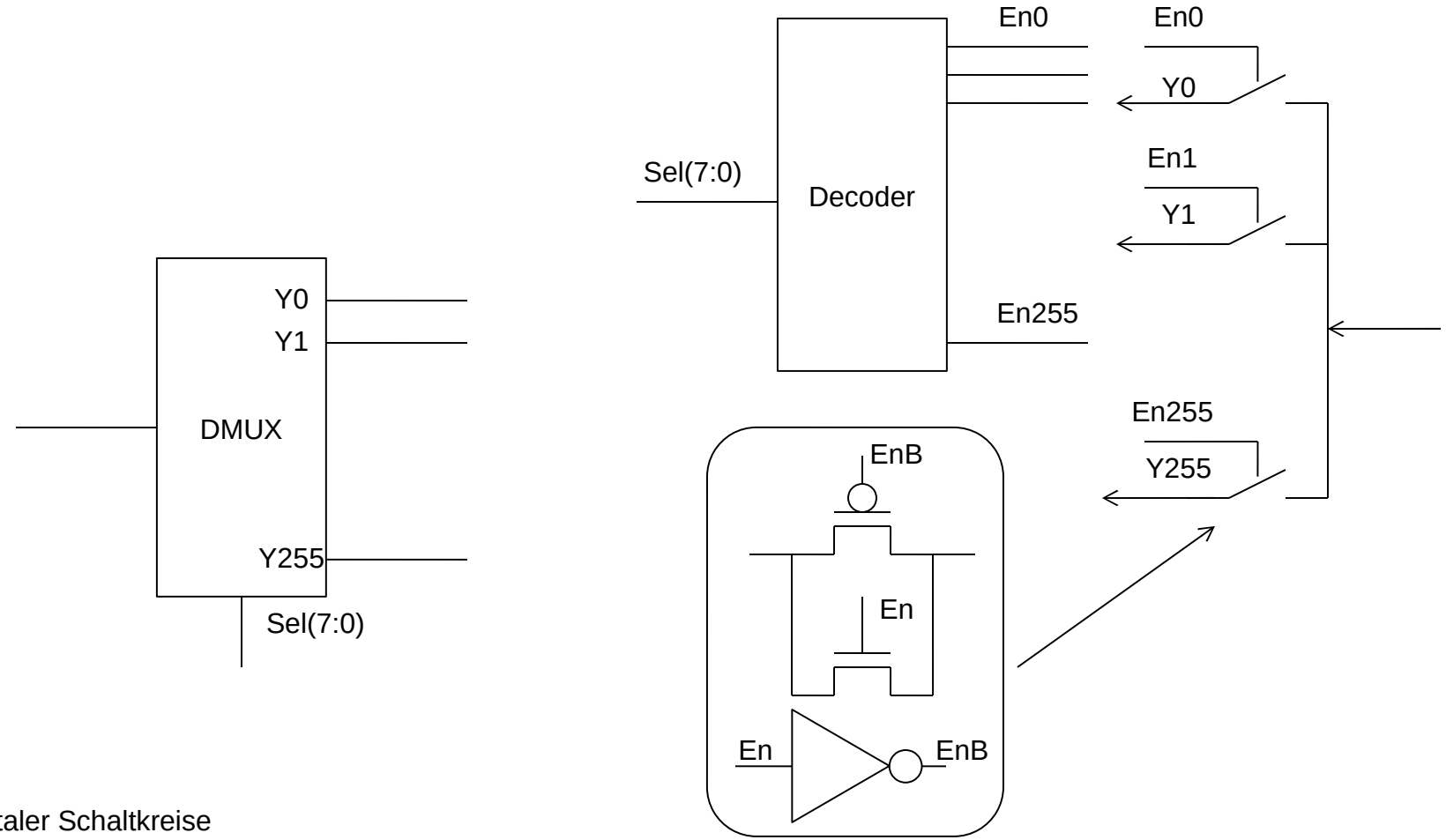
- 256->1 Multiplexer – Implementierung mit Decoder
- 1x4600T + 256 x 6T + 2T = 6100 Transistoren



- 256->1 Analog Multiplexer – Realisierung

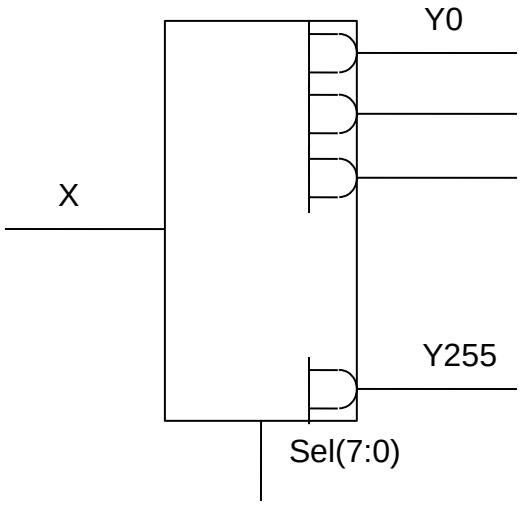


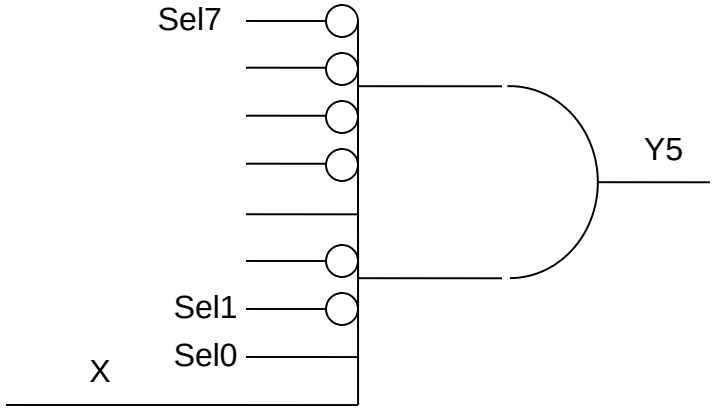
- Analog Multiplexer leitet in beide Richtungen
- Wenn man in einem Analogmultiplexer die Eingänge und Ausgänge „vertauscht“ bekommt man einen Demultiplexer.



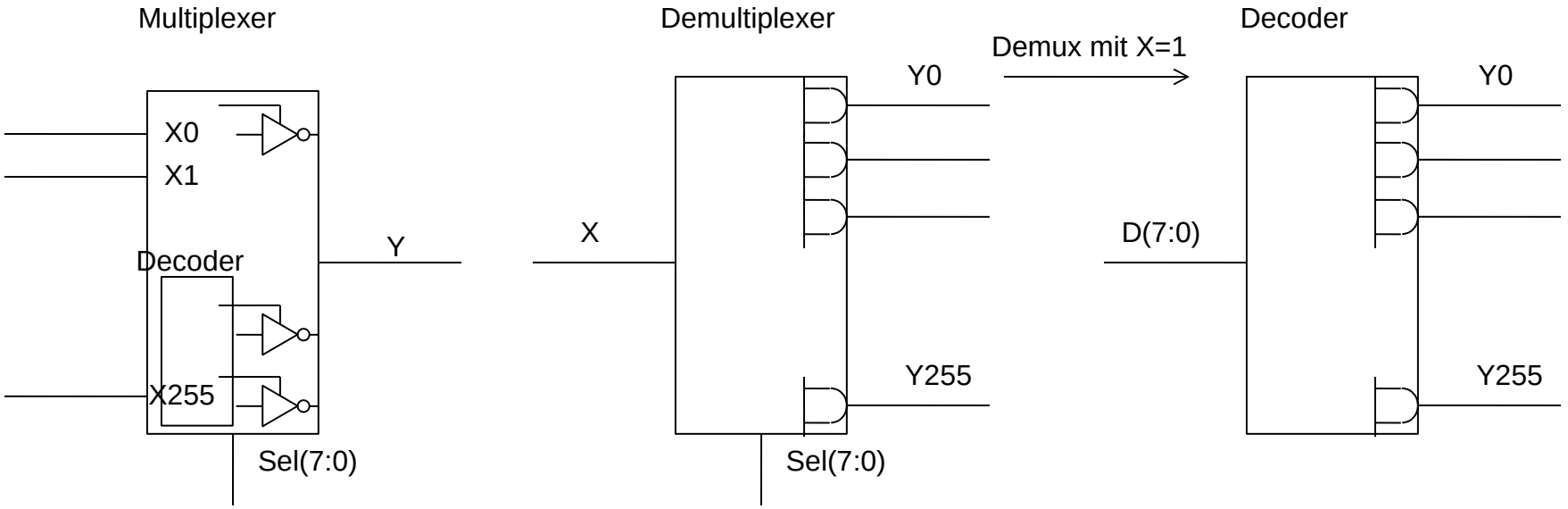
- Digital Demultiplexer
- Demultiplexer mit Eingang 1 -> Dekoder
- 256 x 22T(And) + 8 x 2T(Inv) ~ 5600T

```
Y0 = X & (D == 8‘d0);
Y1 = X & (D == 8‘d1);
Y2 = X & (D == 8‘d2);
Y3 = X & (D == 8’d3);
…
```

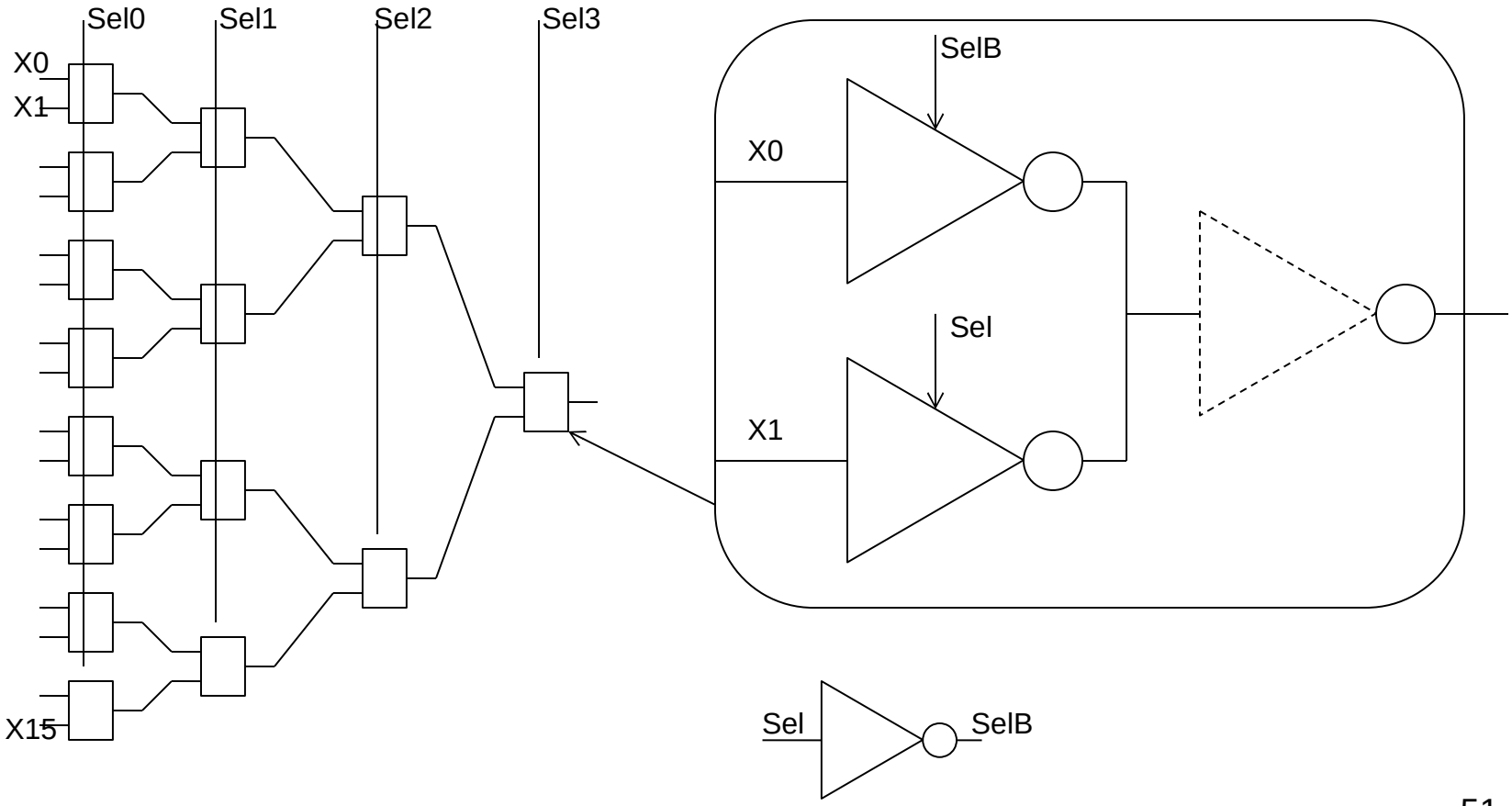




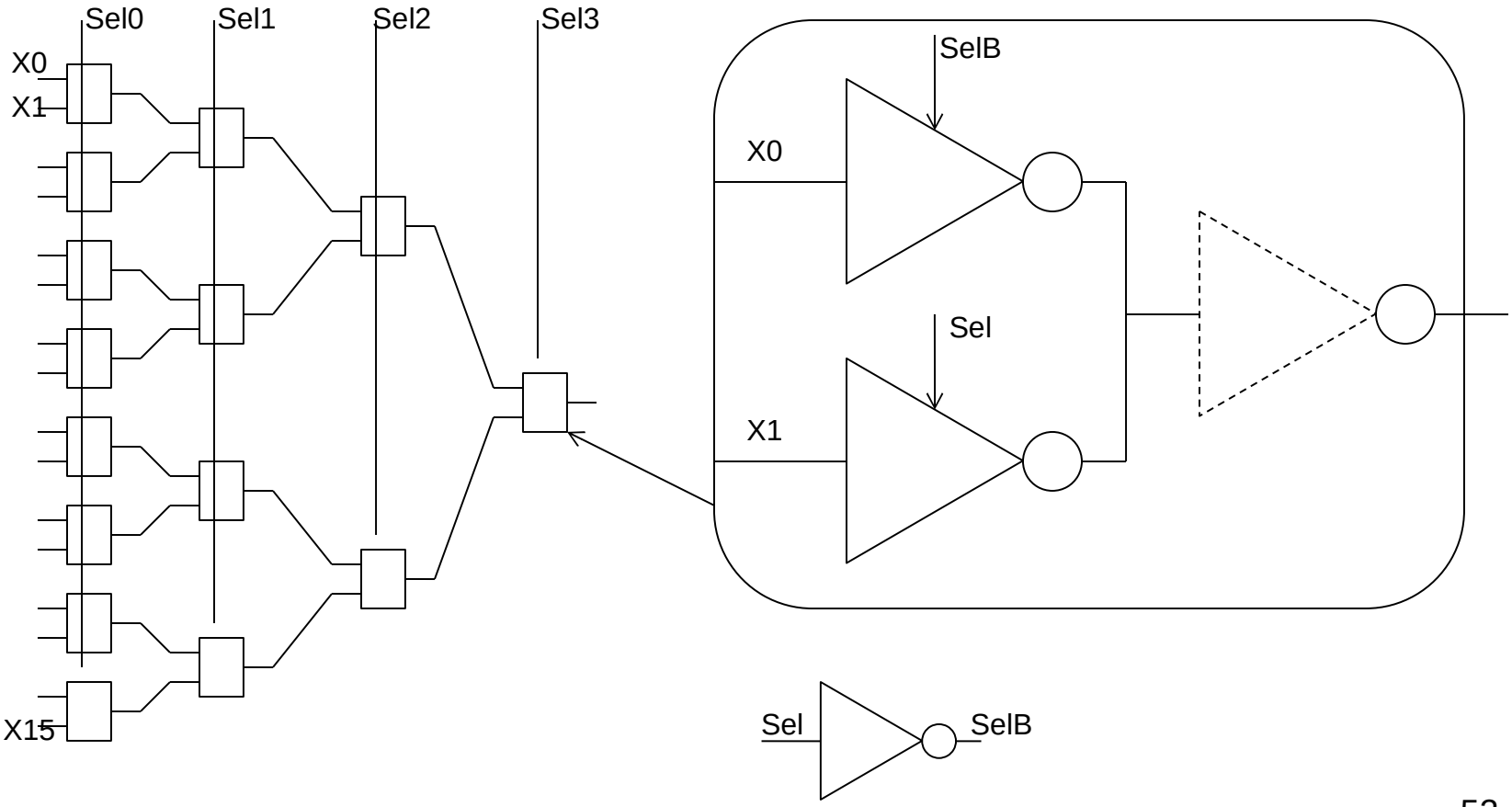
- Zusammenfassung:
- Multiplexer
- Demultiplexer
- Dekoder
- Coder (nächstes mall)



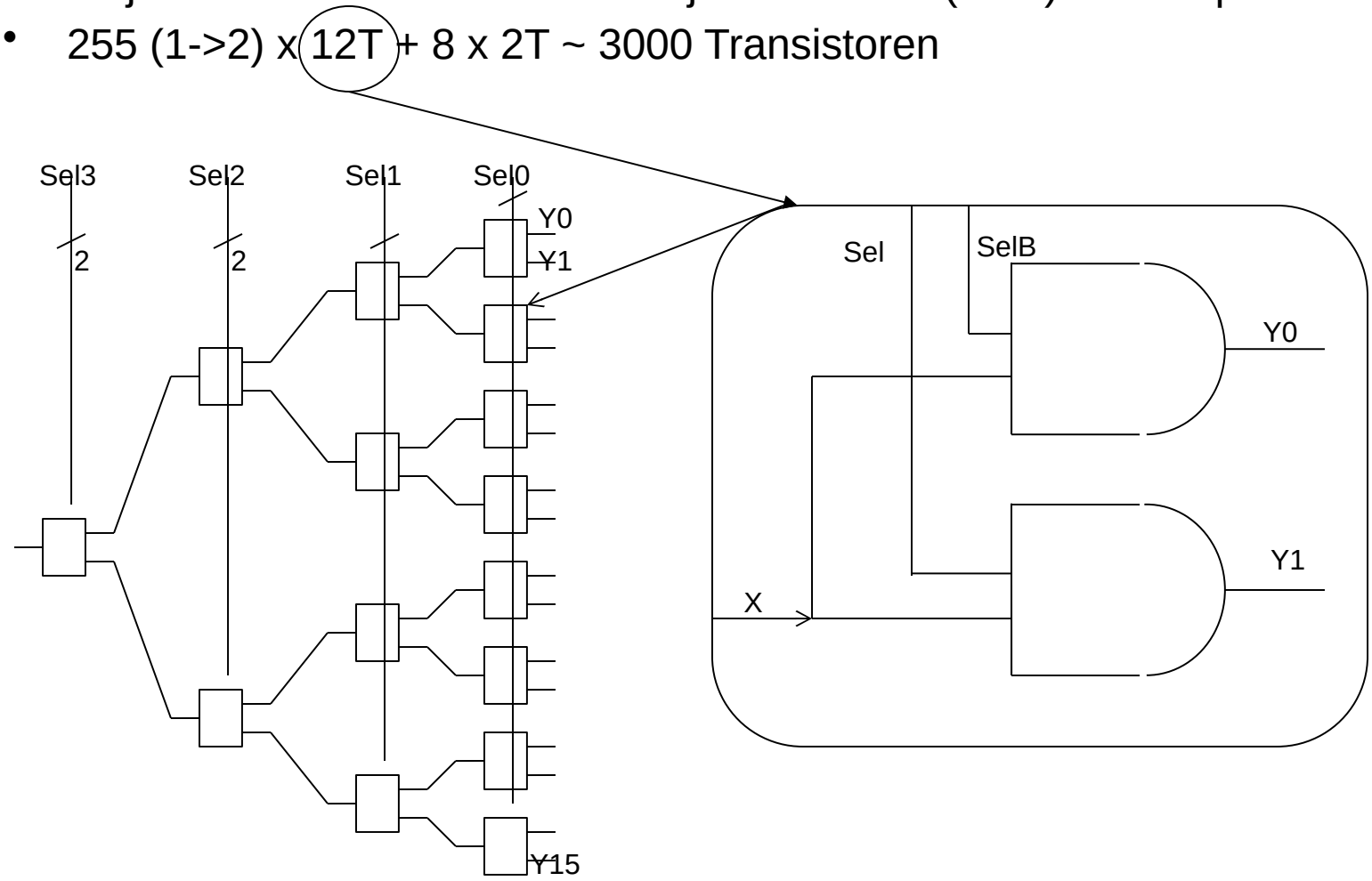
- Multiplexer kann mit weniger Transistoren realisiert werden
- Baumstruktur
- In jedem Knoten verwenden wir jeweils einen (2->1) Multiplexer. Der Select Eingang vom Multiplexer der ersten Stufe wird an Sel0 angeschlossen, usw.



- 256->1 Multiplexer
- 255 x 8T (2->1 Mux) + 8 x 2T(Inv) ~ 2040 Transistoren
- Zu Vergleichen mit 6100 T. (Folie 46) (3x kleiner)
- Die Schaltung ist langsamer – mehrere Stufen

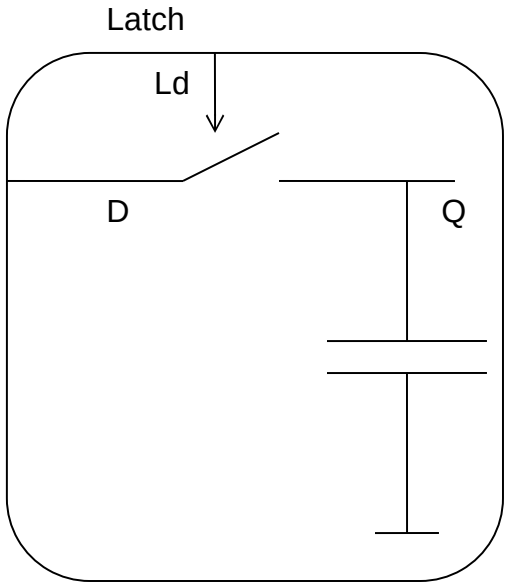


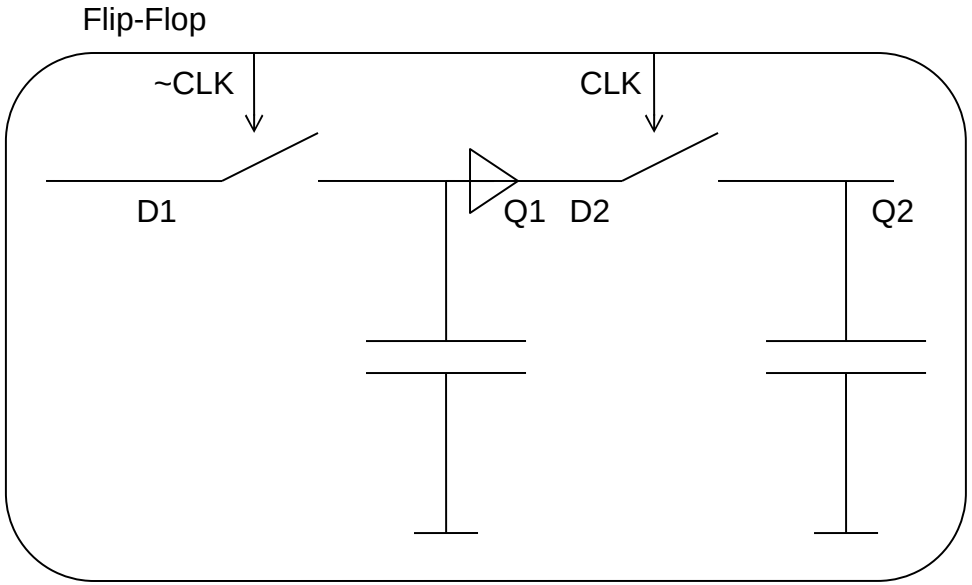
- Auch ein Demultiplexer (und Dekoder) kann als Baumstruktur realisiert werden
- In jedem Knoten verwenden wir jeweils einen (1->2) Demultiplexer
- 255 (1->2) x 12T + 8 x 2T ~ 3000 Transistoren



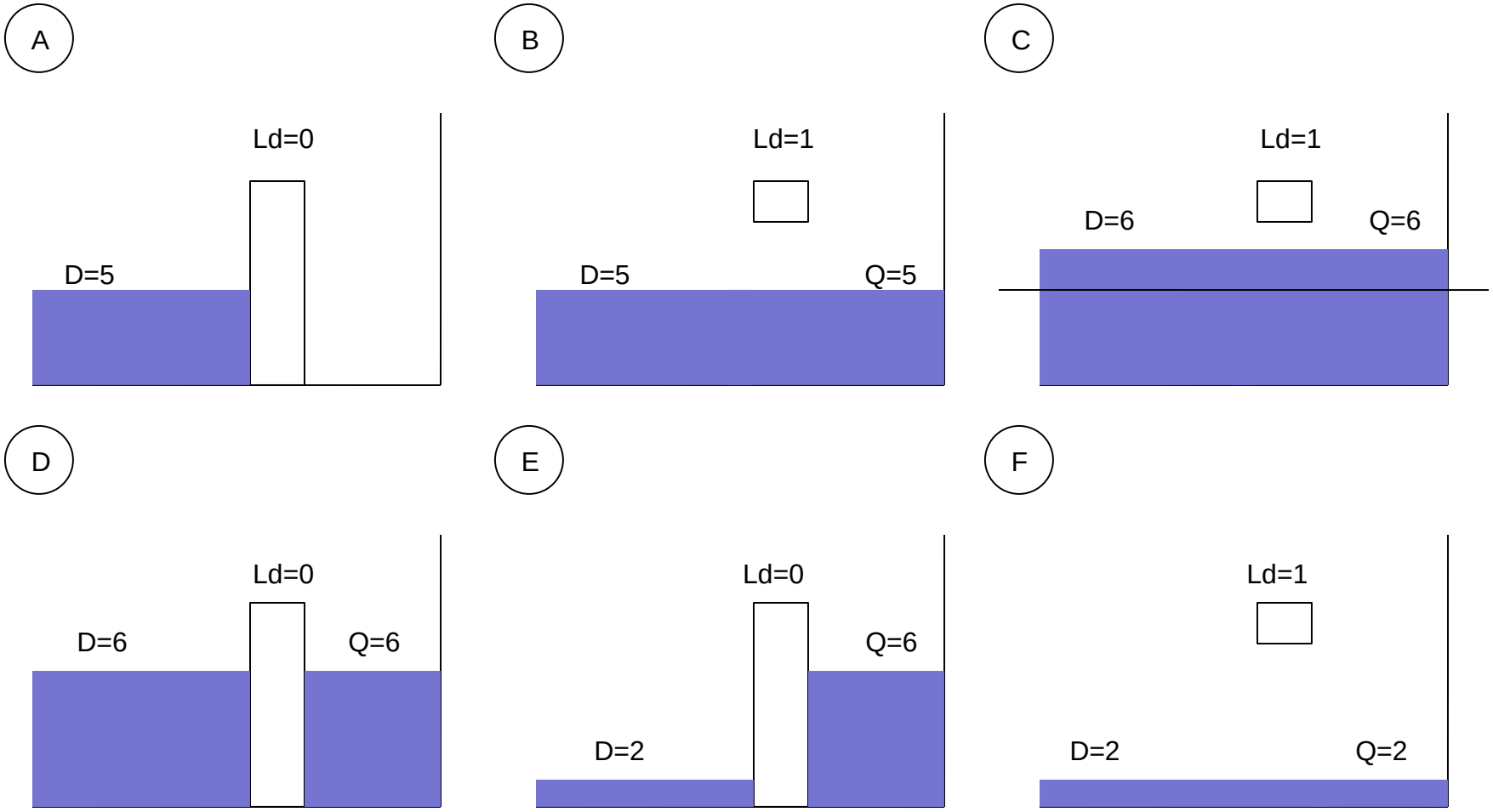
Latch, Flipflop

- 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

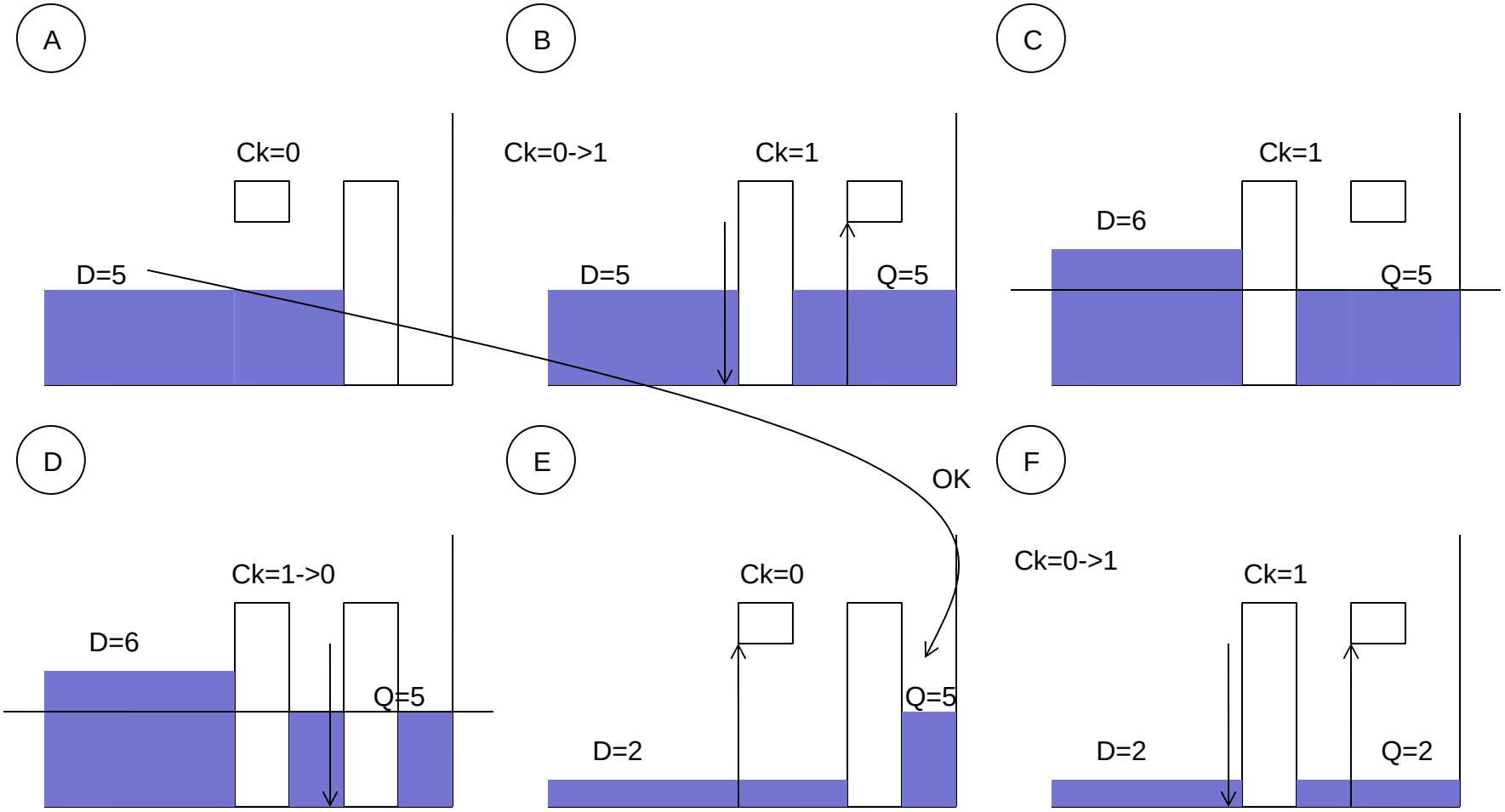




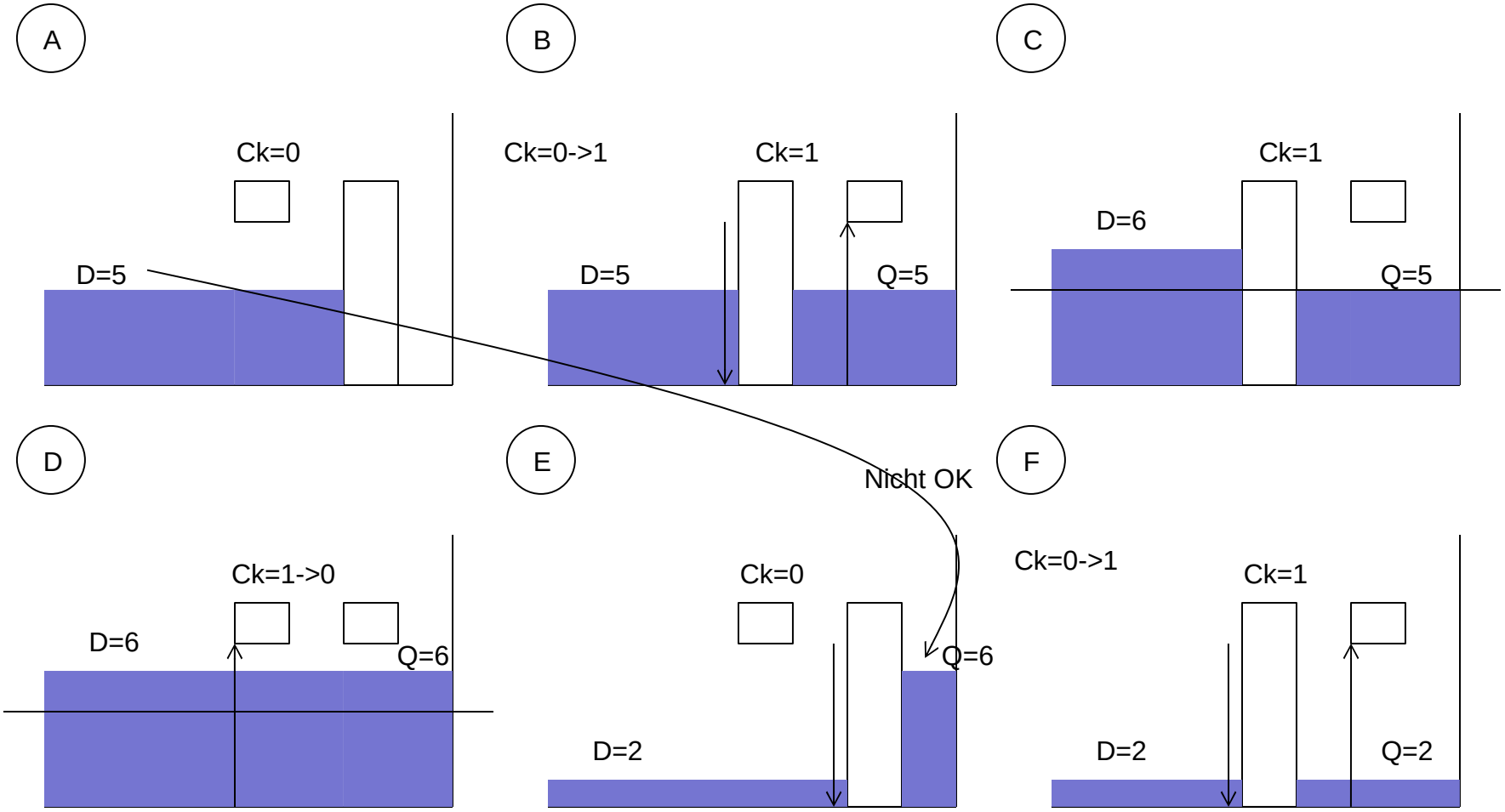
- Latch erinnert an ein System mit 1 Schleuse.



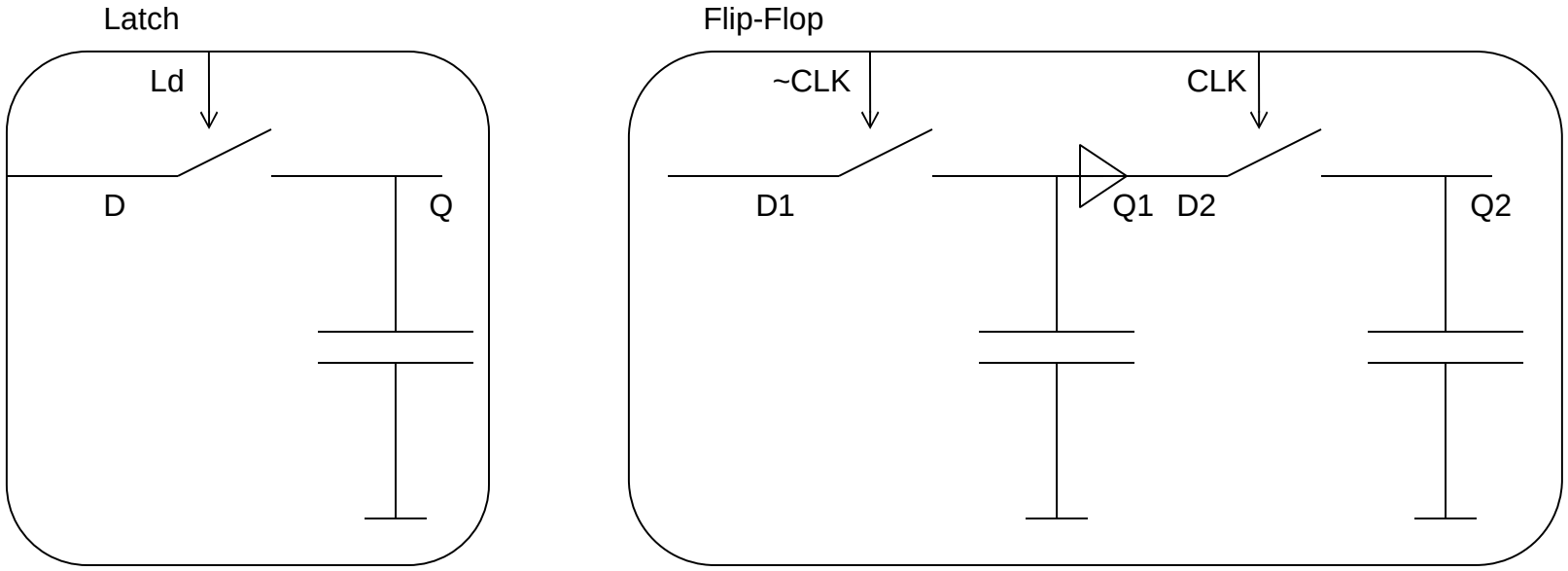
- FF erinnert an ein System mit 2 Schleusen.



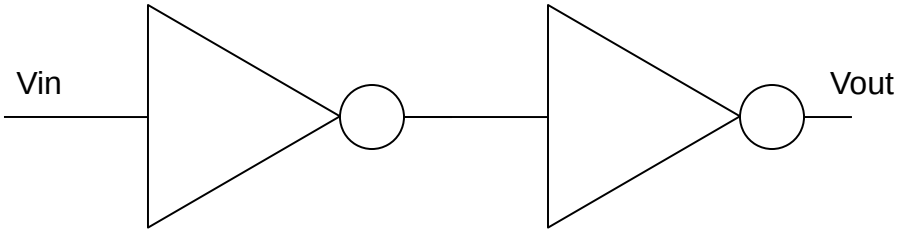
- FF erinnert an ein System mit 2 Schleusen, beide Tore dürfen nicht gleichzeitig geöffnet werden

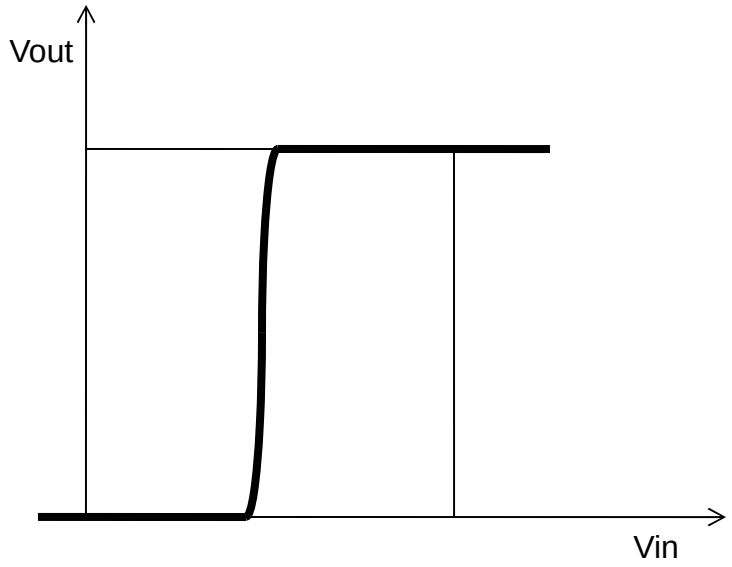


- Nachteil einer Latch-Schaltung mit Kondensatoren – sie kann den Zustand nicht beliebig lange halten
- Der Kondensator wird langsam entladen

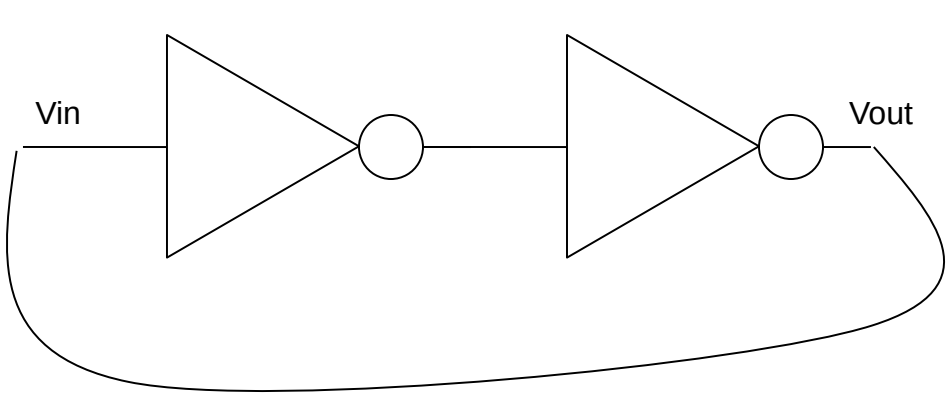


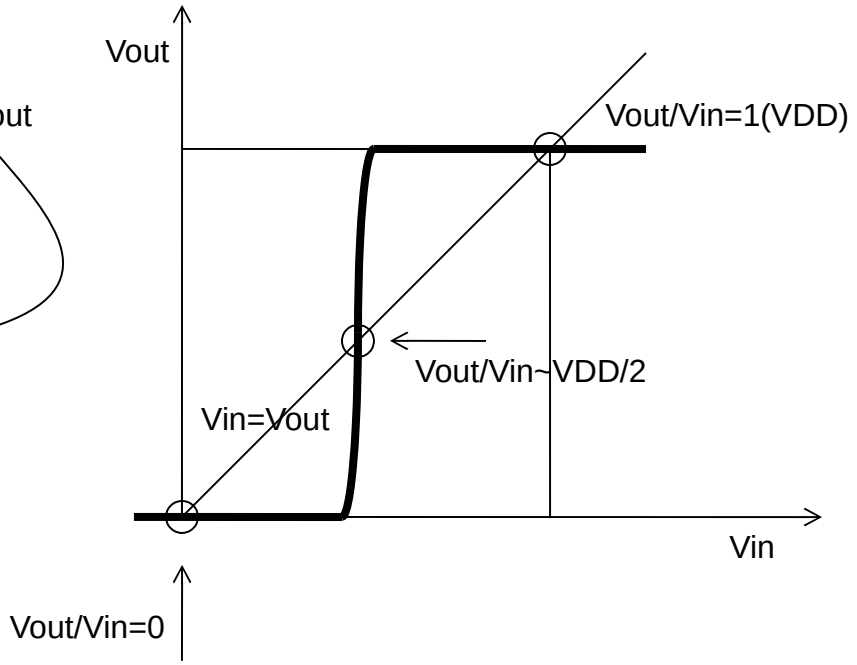
- Statische Speicherzellen



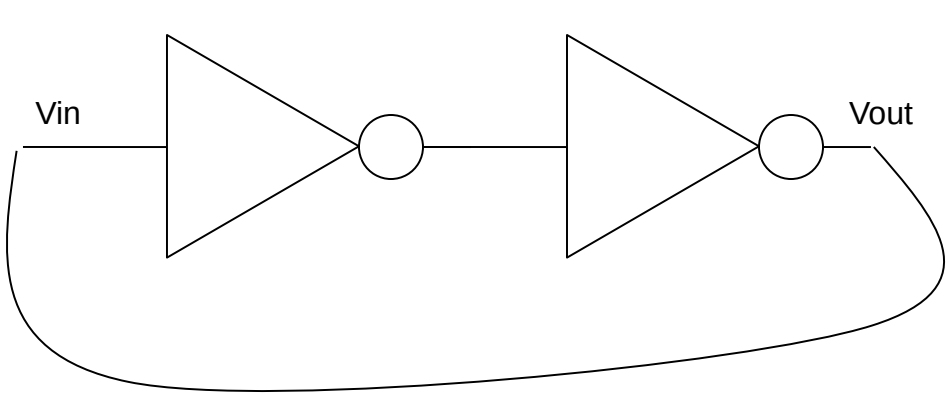


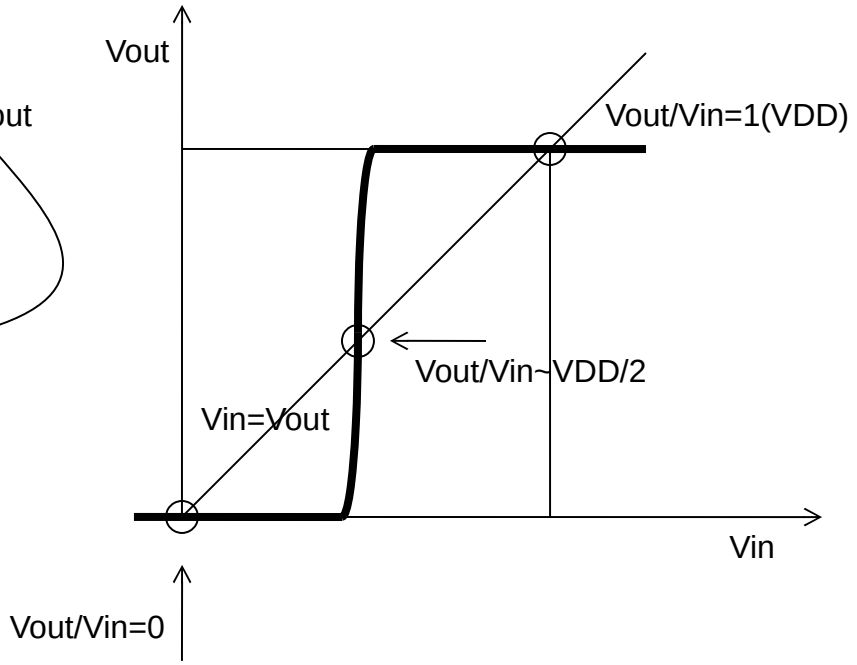
- Statische Speicherzellen
- Wenn wir den Ausgang des zweiten Inverters mit dem Eingang des ersten verbinden, haben wir Vin = Vout.
- Der Zustand der Schaltung liegt im Schnittpunkt der Kennlinie Vout = f(Vin) und der Gerade Vout = Vin.



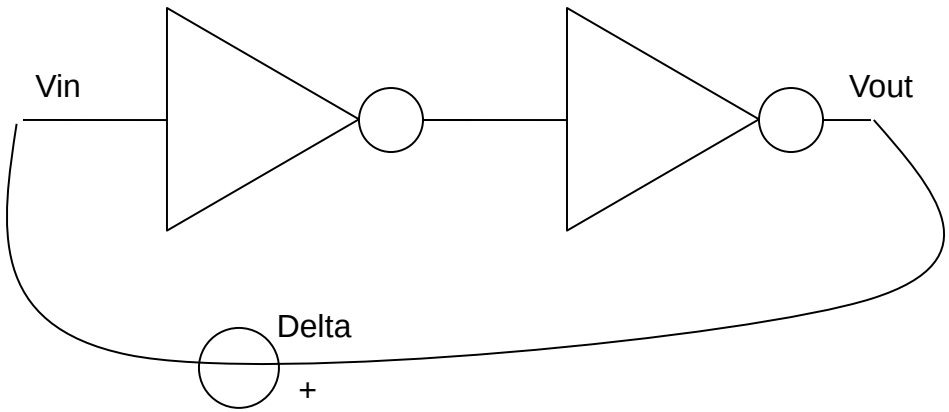


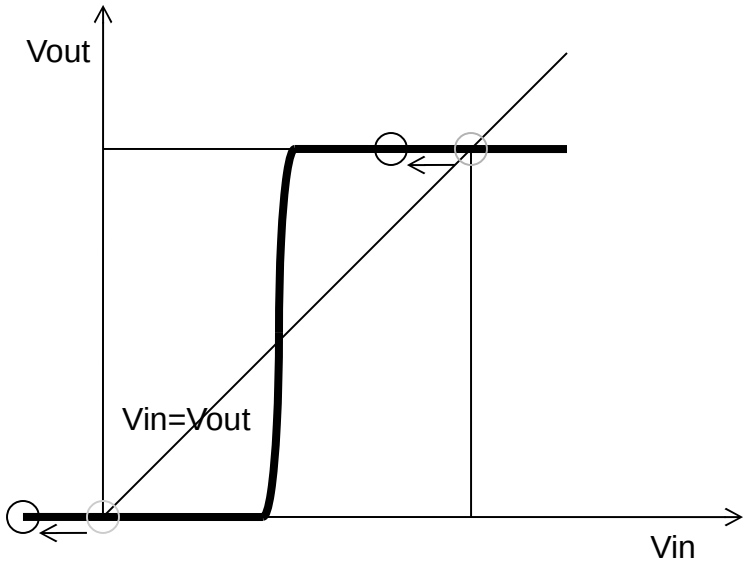
- Drei Schnittpunkte
- 1. Vout/Vin = 0 (logische 0)
- 2. Vout/Voin = VDD (logische 1)
- 3. Vout=Vin ~ VDD/2 (undefiniert).



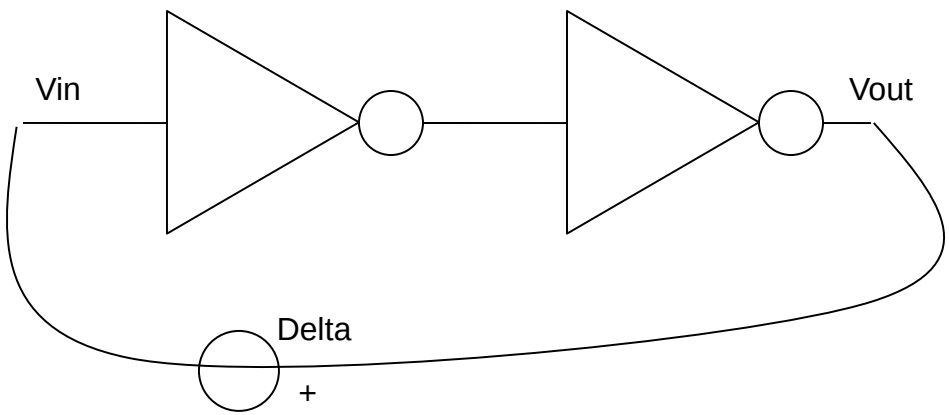


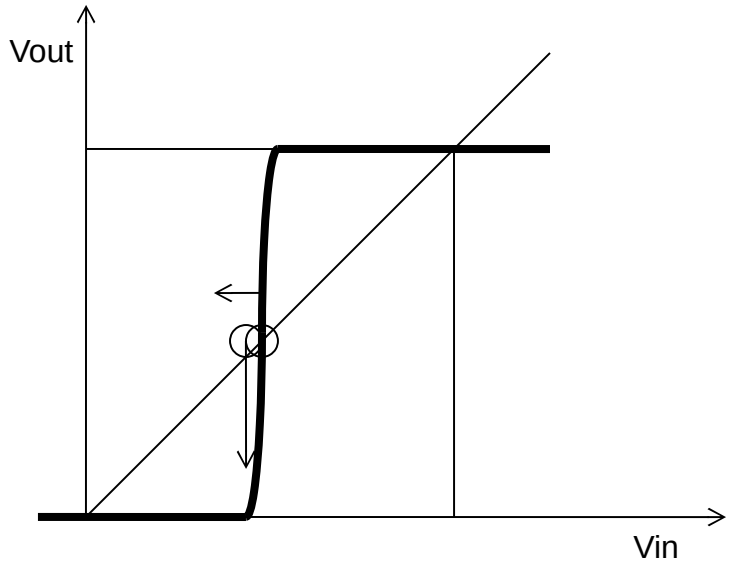
- Die ersten zwei Arbeitspunkte sind stabil
- kleine Störung Delta
- Vin = Vout – Delta
- Vout wird nicht beeinflusst



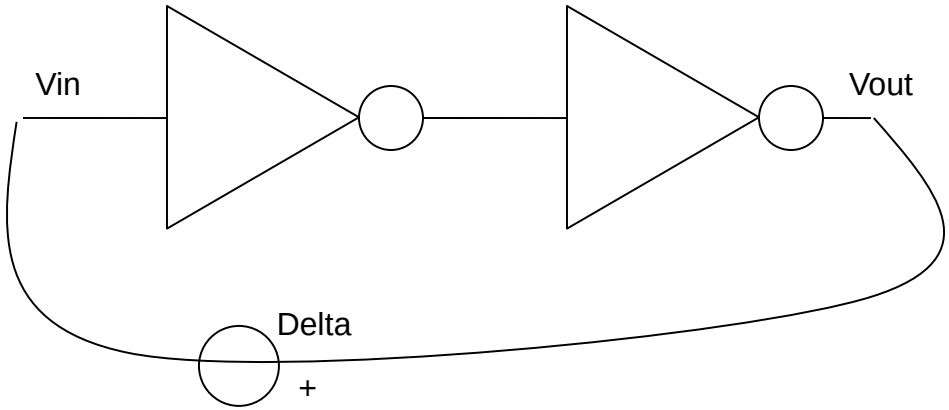


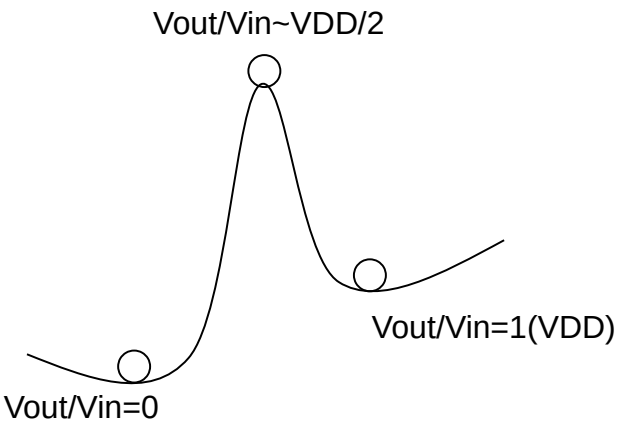
- Der dritte Arbeitspunk ist instabil
- Vin = Vout – Delta
- Verringerung von Vin führt zu noch größerer Verringerung von Vout
- Die Schaltung kommt aus dem instabilen Arbeitspunkt immer in den Arbeitspunkt Vout/Vin = 0 oder in den Arbeitspunkt Vout/Voin = VDD



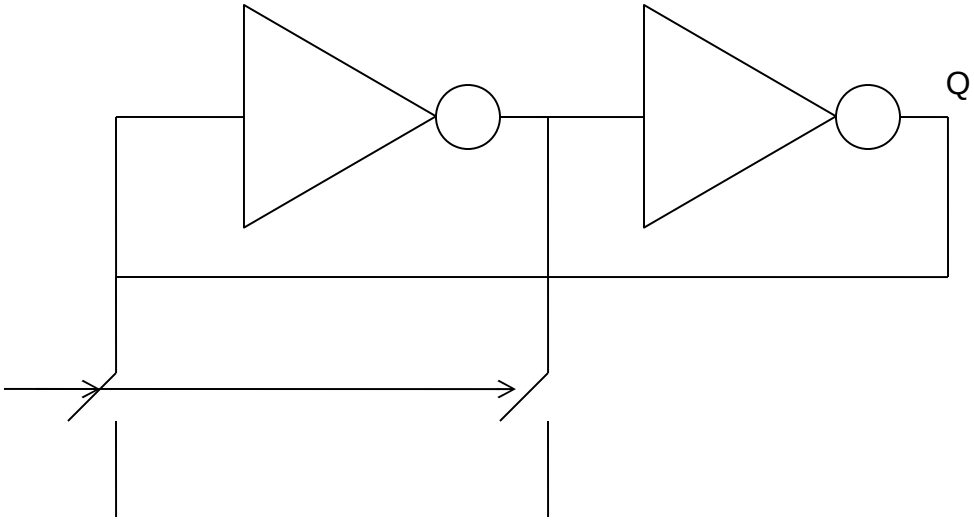


- Der dritte Arbeitspunk ist instabil
- Vin = Vout – Delta
- Verringerung von Vin führt zu noch größerer Verringerung von Vout
- Die Schaltung kommt aus dem instabilen Arbeitspunkt immer in den Arbeitspunkt Vout/Vin = 0 oder in den Arbeitspunkt Vout/Voin = VDD

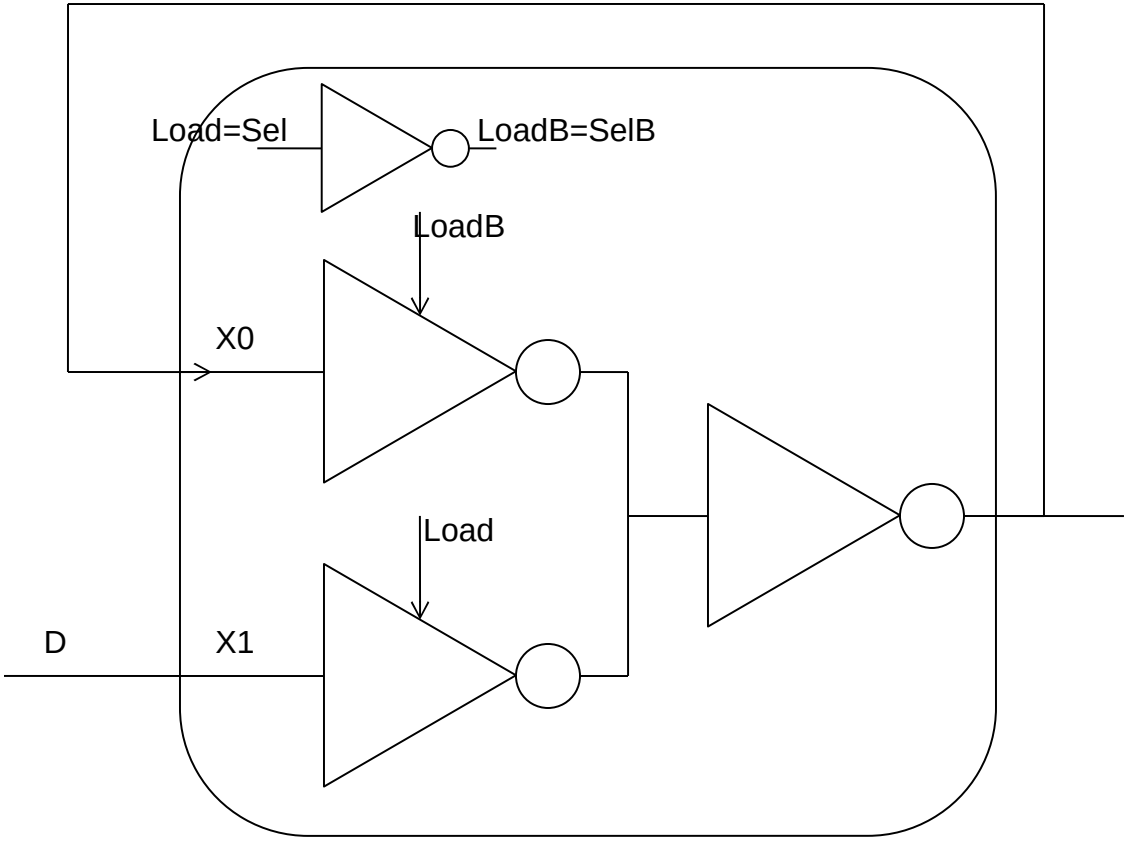




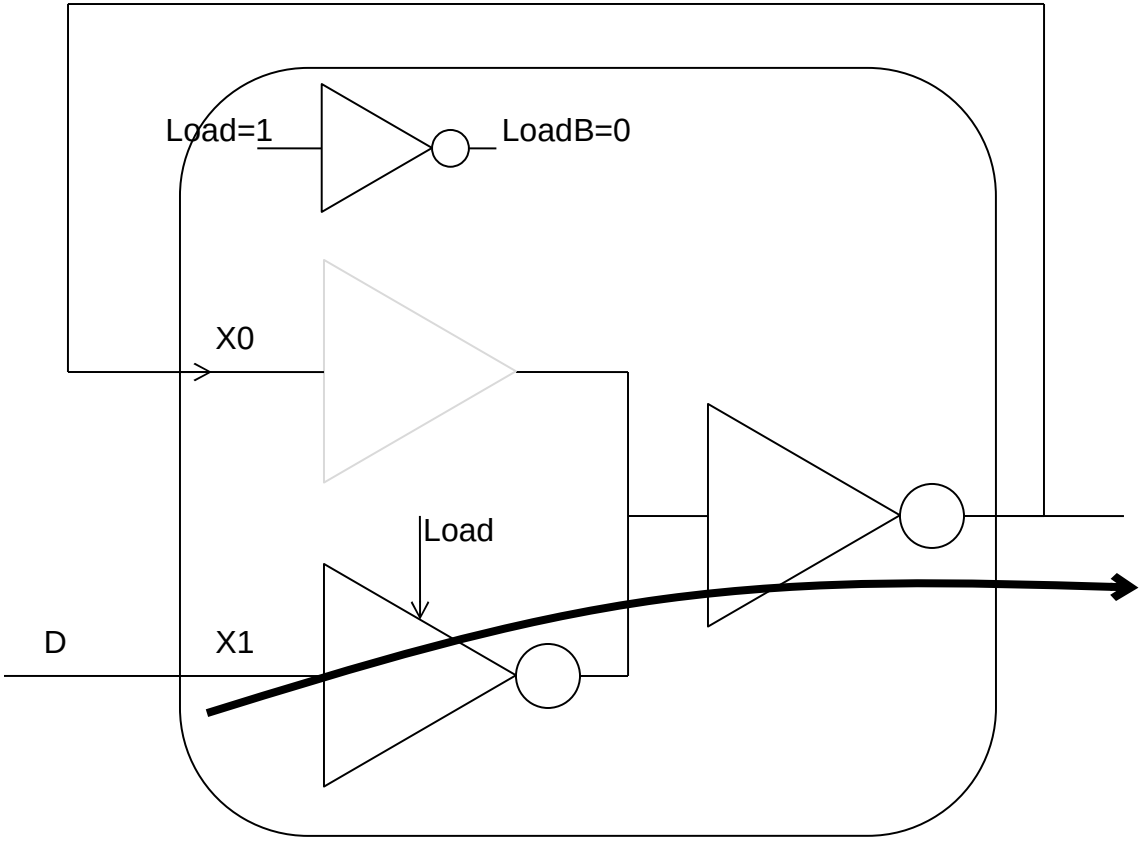
- Die Schaltung ist die Basis einer SRAM Zelle



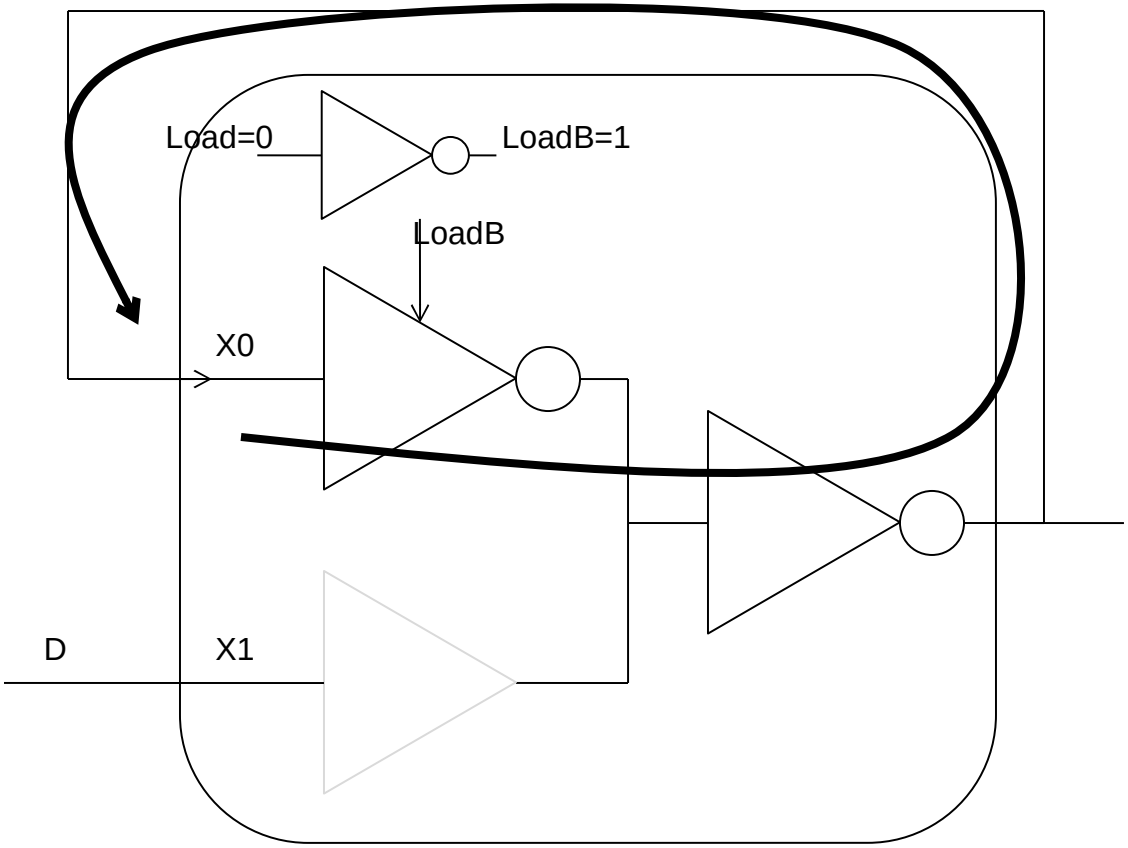
- Ein Latch basiert normalerweise auf einer modifizierten Version der Speicherzelle.
- Multiplexer wird benutzt, Select Eingang ist an Load Signal angeschlossen



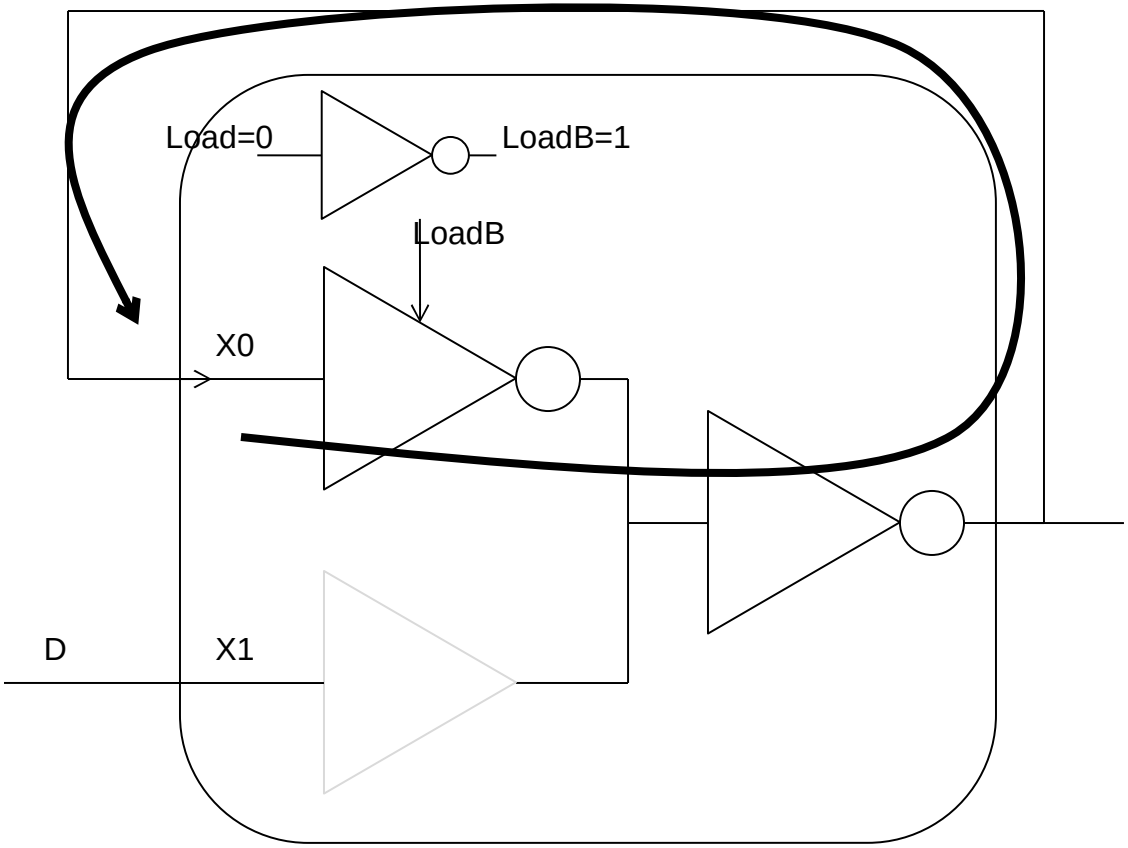
- Wenn Load = 1, das Latch ist „transparent“ – der Eingang ist direkt am Ausgang sichtbar.



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



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



Richtig:

```
reg q;
wire D, Load;

always @* begin

  if (Load) q = D;
end
```

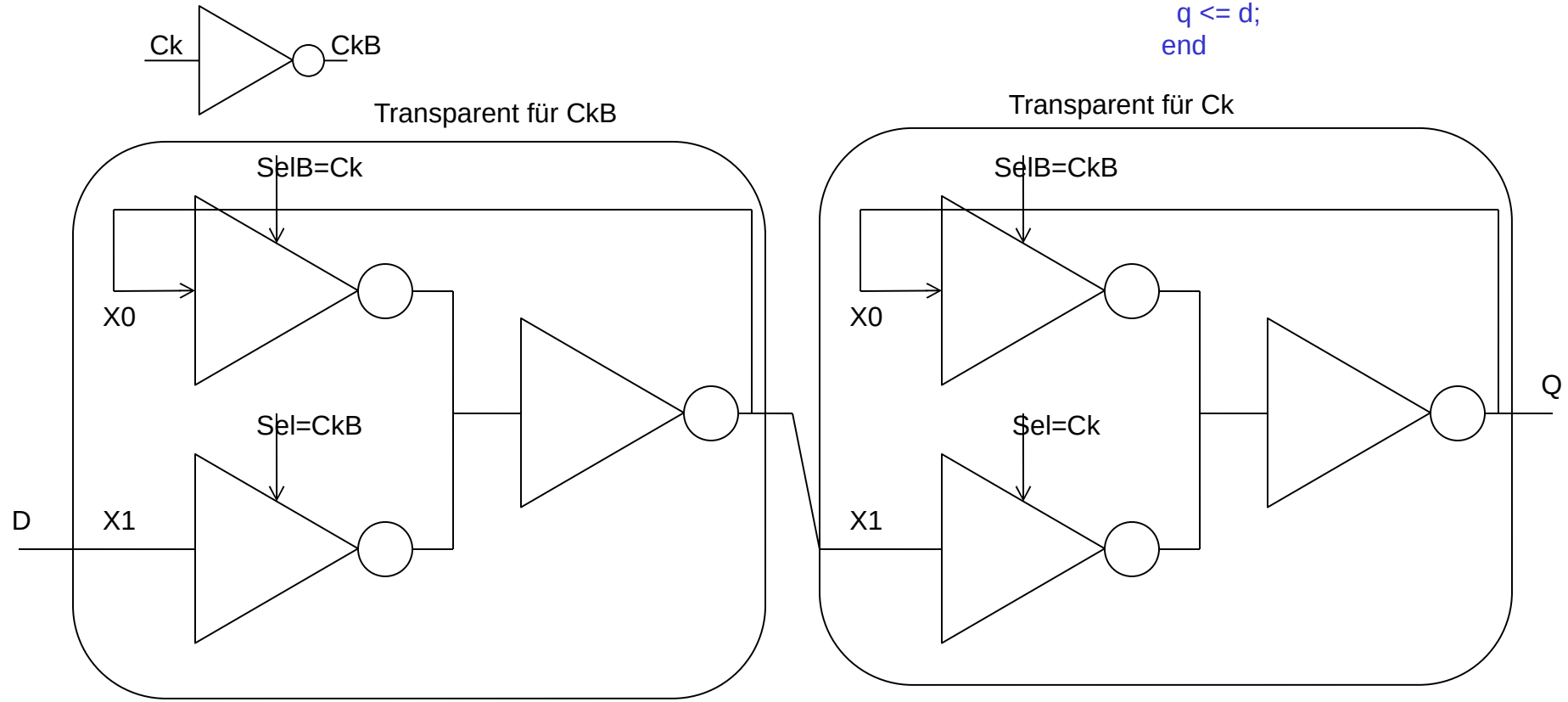
```
reg q;
wire D, Load;

always @(Load or D)  begin

  if (en) q = D;
end
```

Nicht zu empfehlen

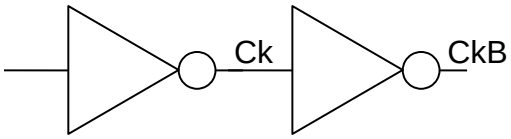
```
wire q;
wire D, Load;

assign q = Load ? D : q;
```

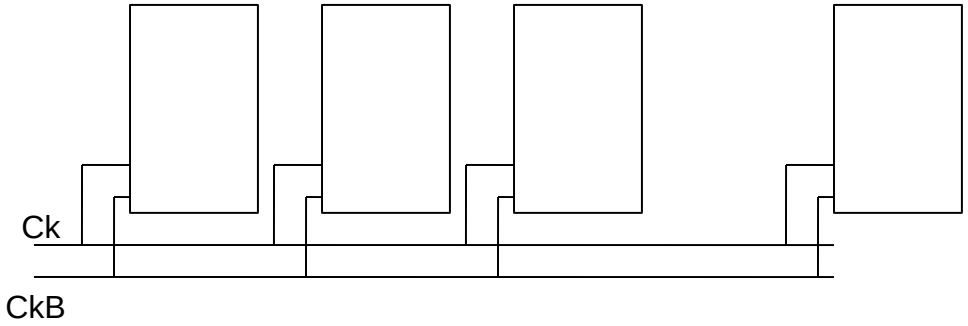
- Einen Flip-Flop bilden wir aus zwei Latches
- Es soll dabei vermieden werden, dass beide Latches gleichzeitig transparent werden, vor allem wenn wich Ck von 1 auf 0 ändert (inaktive Flanke)

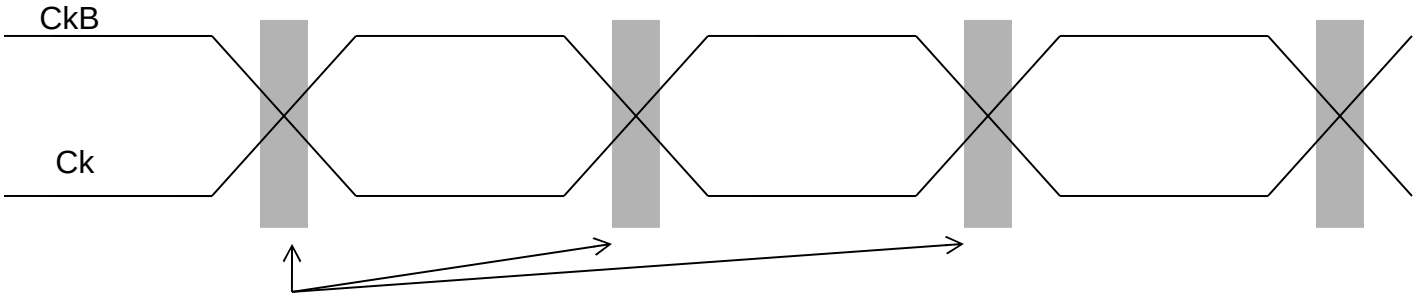
```
reg q;
wire D;
always @(posedge Ck) begin

  q <= d;
end
```



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

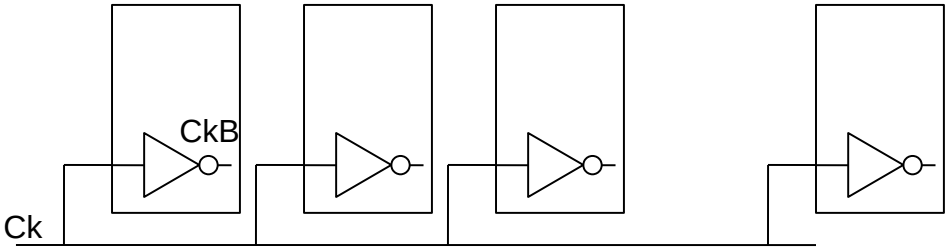


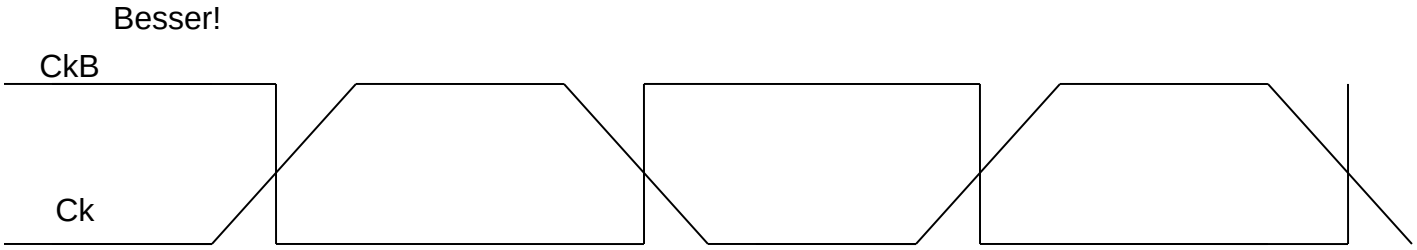




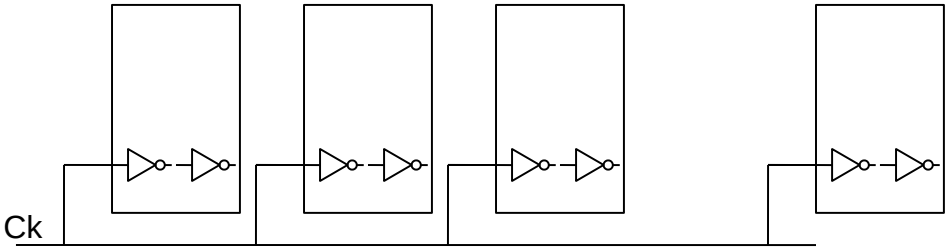
Beide Latches im FF transparent?

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

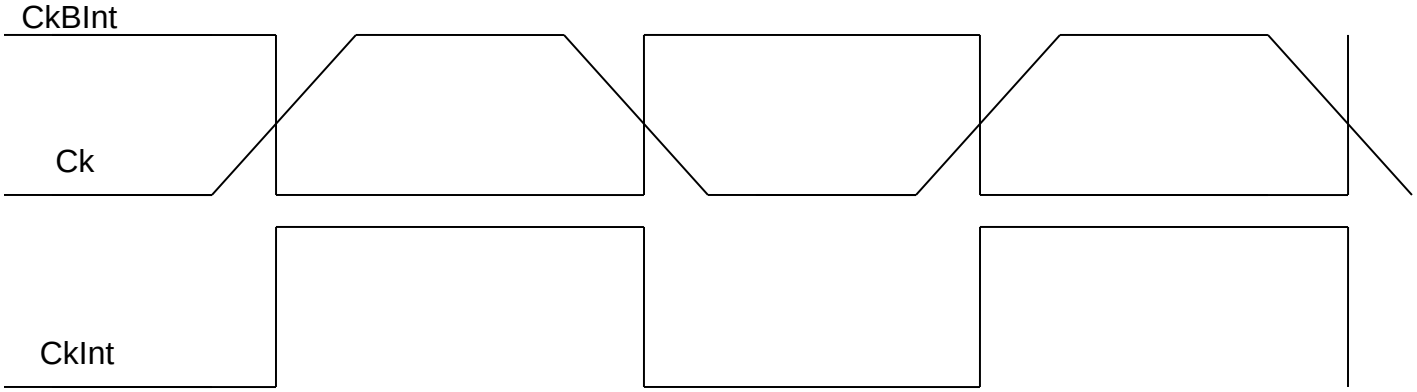




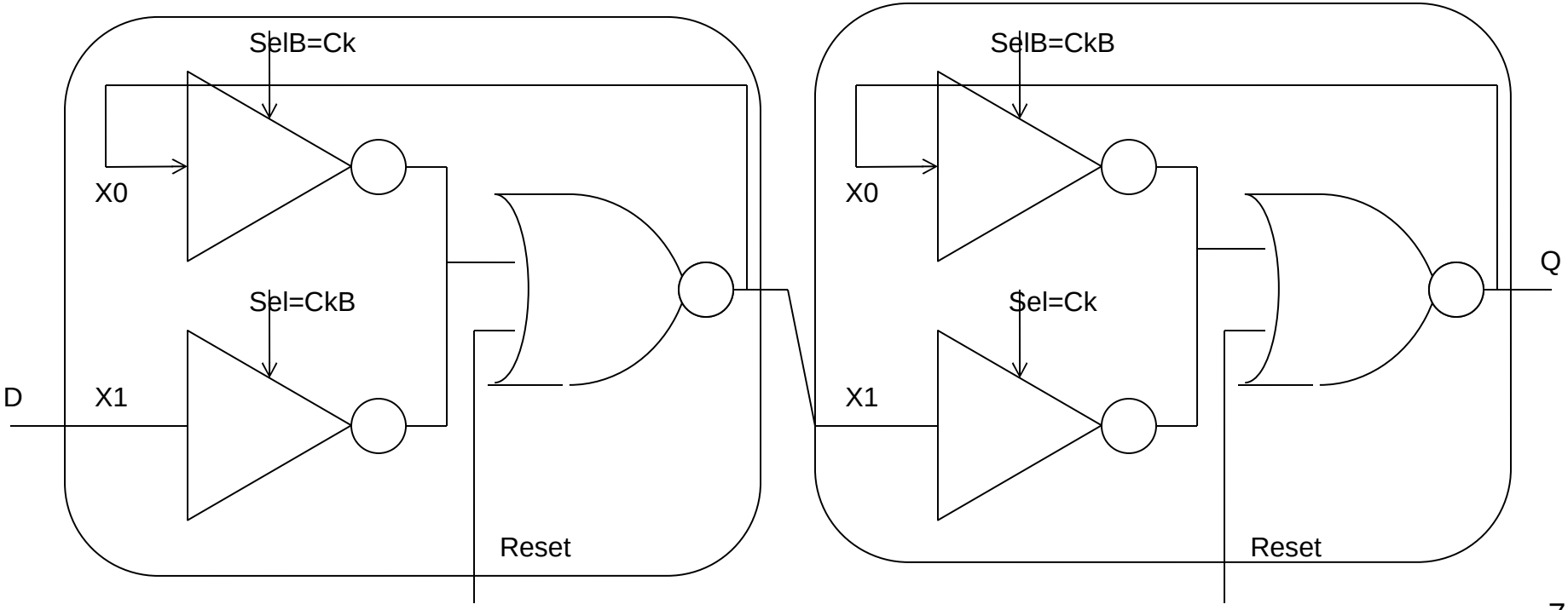
- Um solche Probleme zu vermeiden, werden die Takt-Invertern in der Regel im Flip-Flop eingebaut.



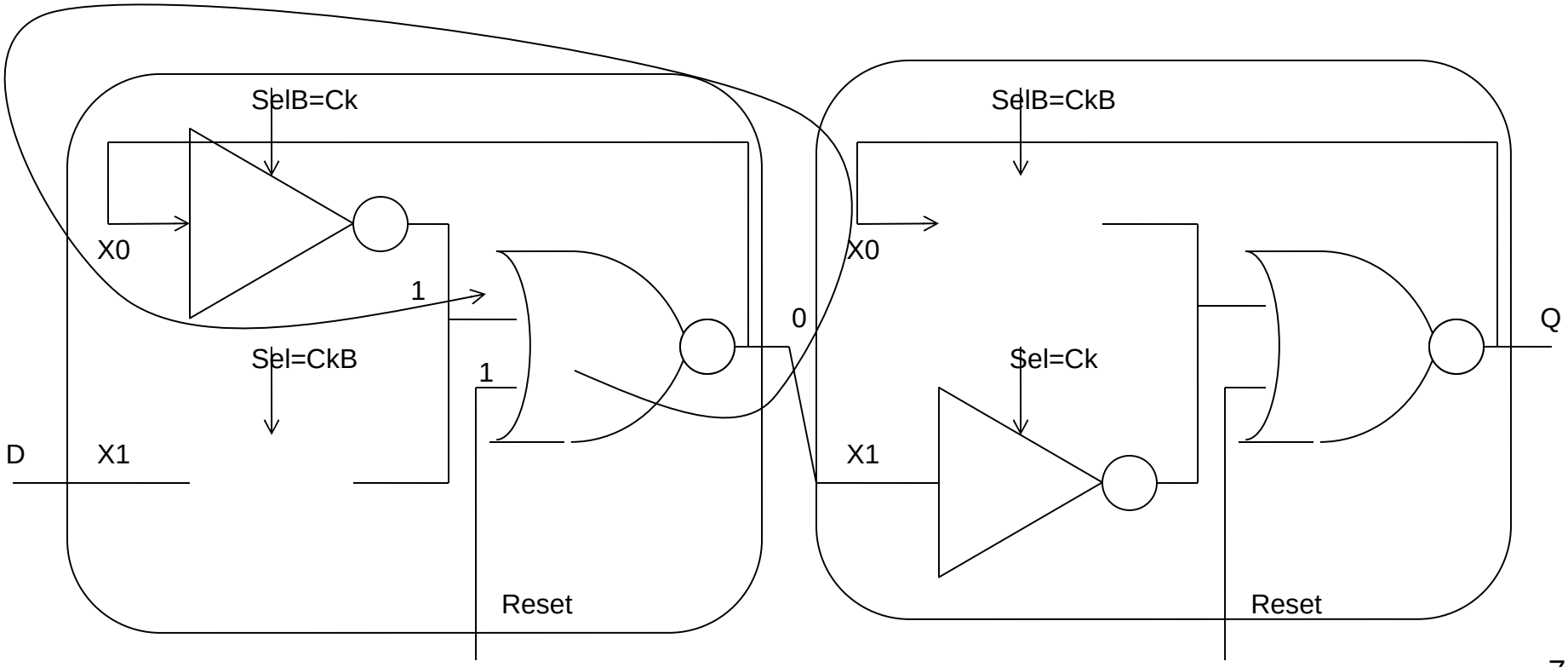
Noch Besser!



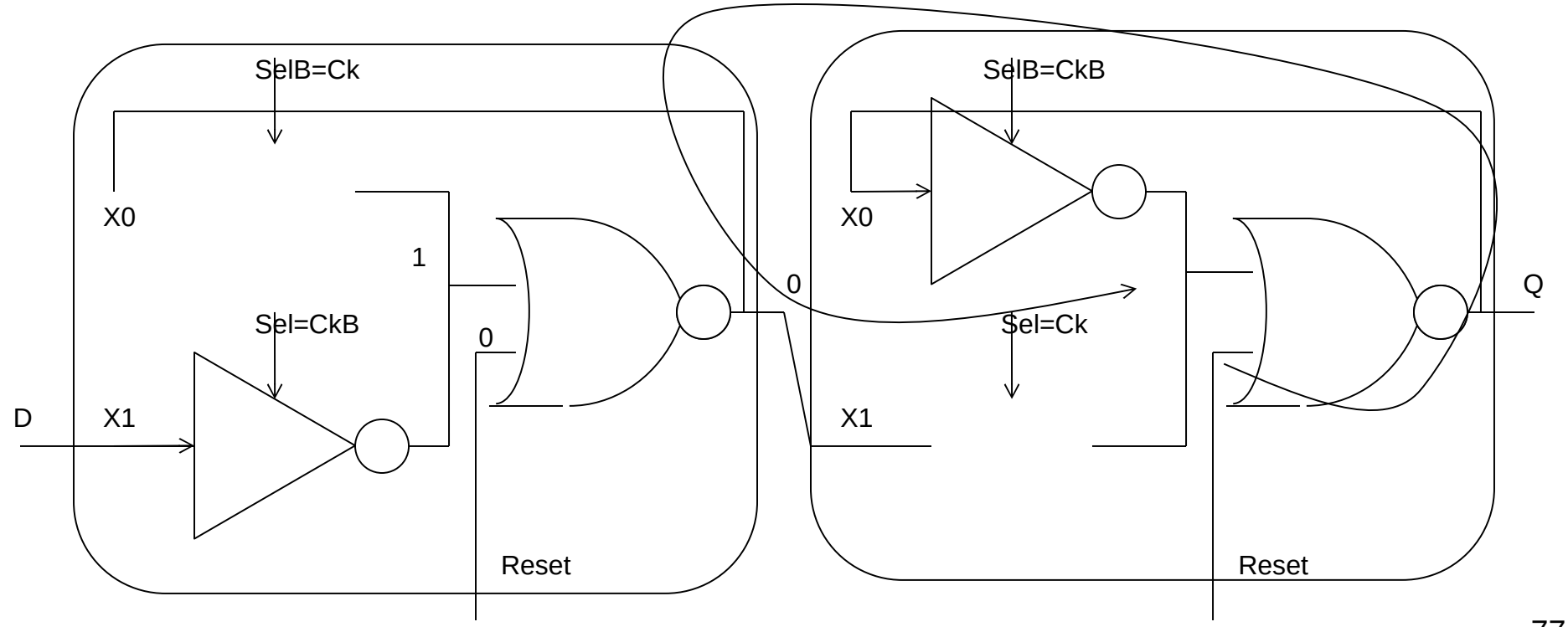
- Nach dem Einschalten der Spannungsversorgung befindet sich ein Flip-Flop, genauso wie eine RAM Zelle, in einem unbekannten logischen Zustand.
- Wir können uns vorstellen, dass sich zuerst alle Flip-Flops in den astabilen Zustand befinden und dann in logisch Eins oder Null Zustand kommen.
- Um einen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



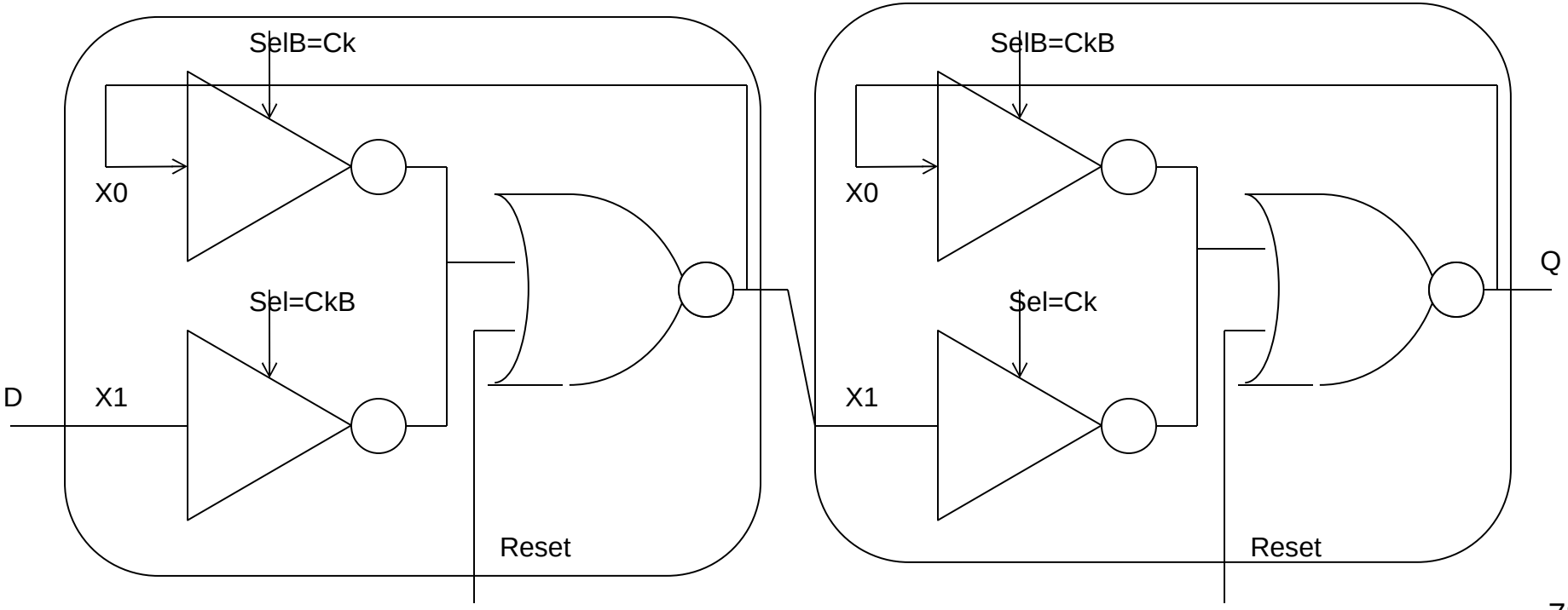
- Betrachten wir einen solch erweiterten FF im Ck=1 Zustand
- In dem fall ist das erste Latch im Speichermodus, das zweite transparent
- Reset = 1 (aktiv high) erzwingt logische Null am Ausgang (1. Latch), sie wird rückgekoppelt. Auf diese Weise bleibt Null gespeichert auch wenn Reset wieder inaktiv (null) wird.



- Betrachten wir einen solch erweiterten FF im Ck=0 Zustand
- In dem fall ist das zweite Latch im Speichermodus, das erste transparent

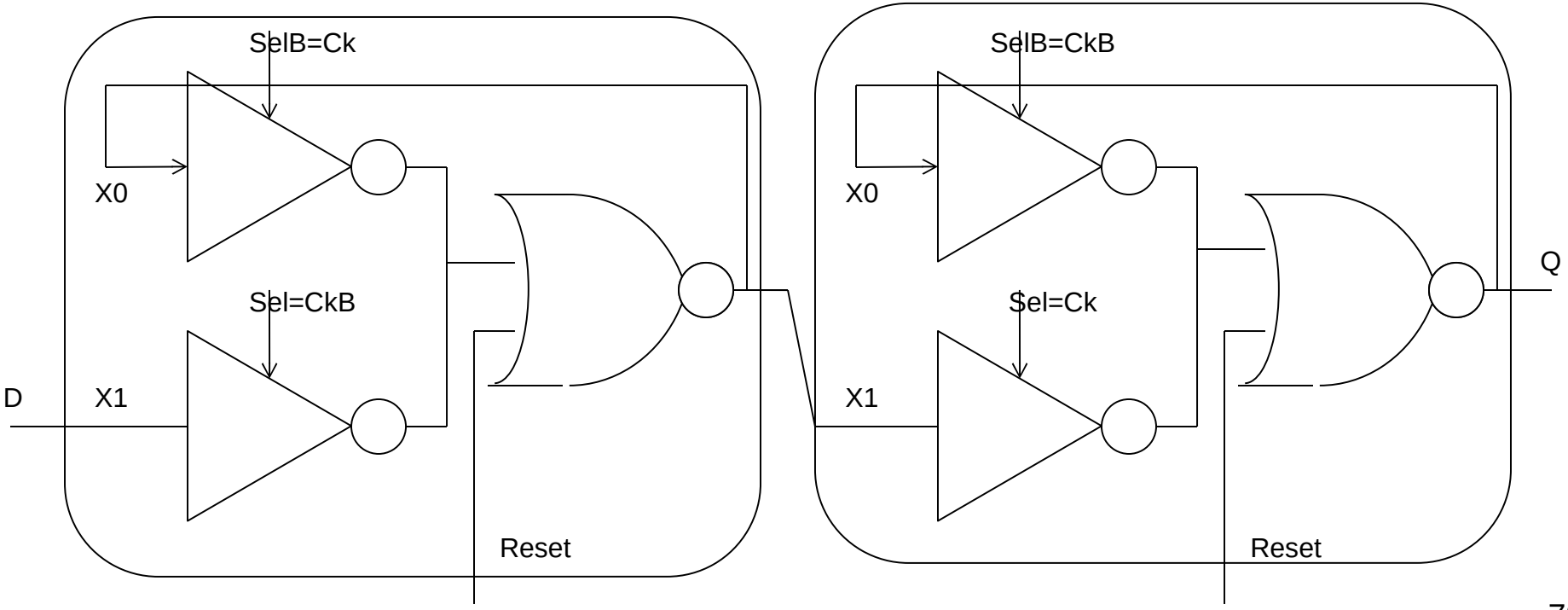


- Im Flip-Flop ist immer wenigstens ein Latch im Speicherzustand, so dass ein Reset immer möglich ist wenn beide Latches die Reset Logik enthalten.
- Asynchrones Reset ist *stärker* als der Takteingang. Sobald Reset = 1 wird, wird der Flip-Flop Ausgang null, unabhängig von D und Ck Eingängen



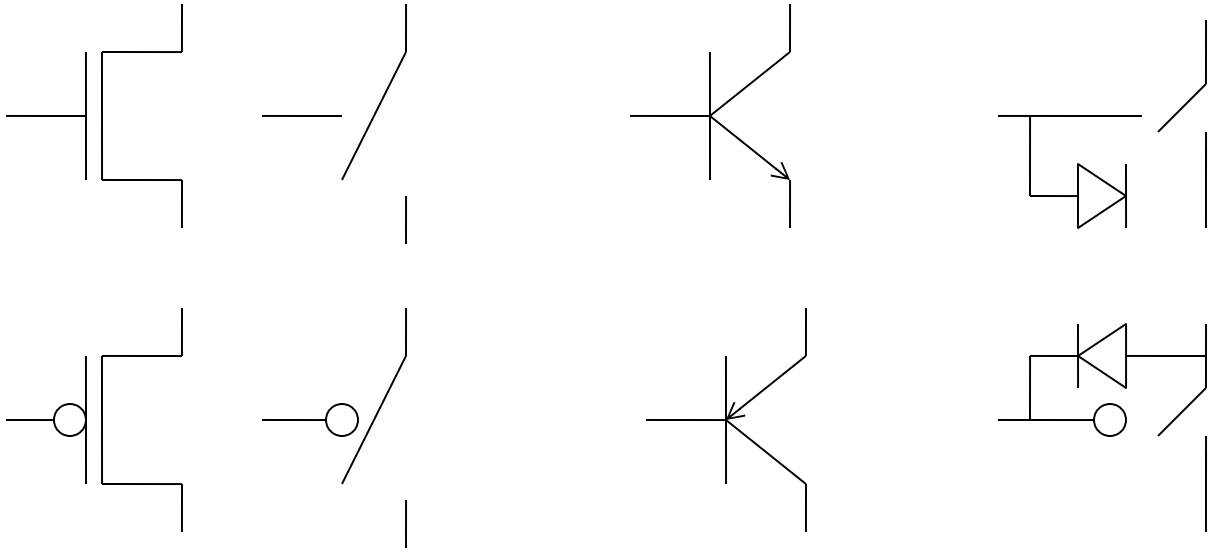
- …

```
reg q;
wire D, Reset;
always @(posedge Ck or posedge Reset) begin
        if(Reset) q <= 0;
        else q <= d;
        end//always
```

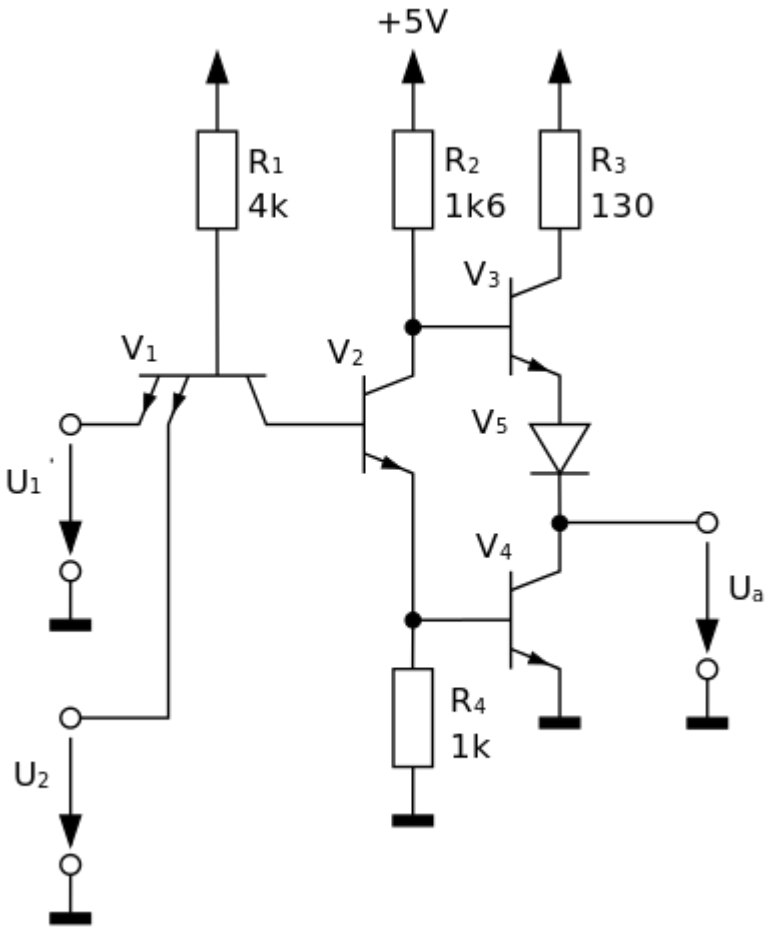


TTL, ECL…

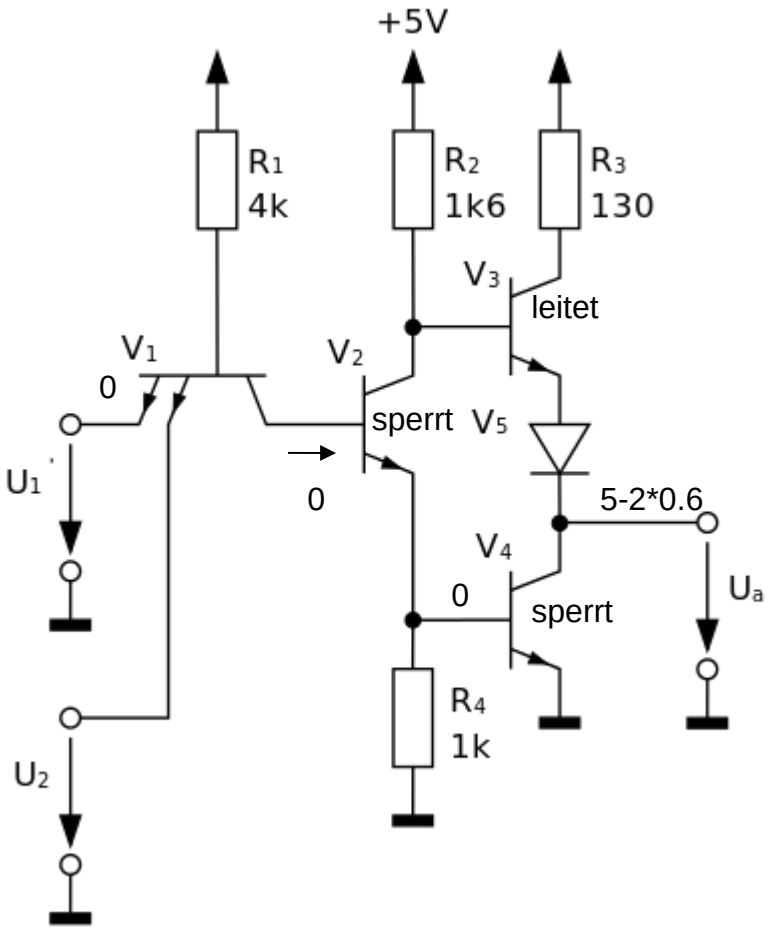
- Weitere Beispiele der Logikelementen



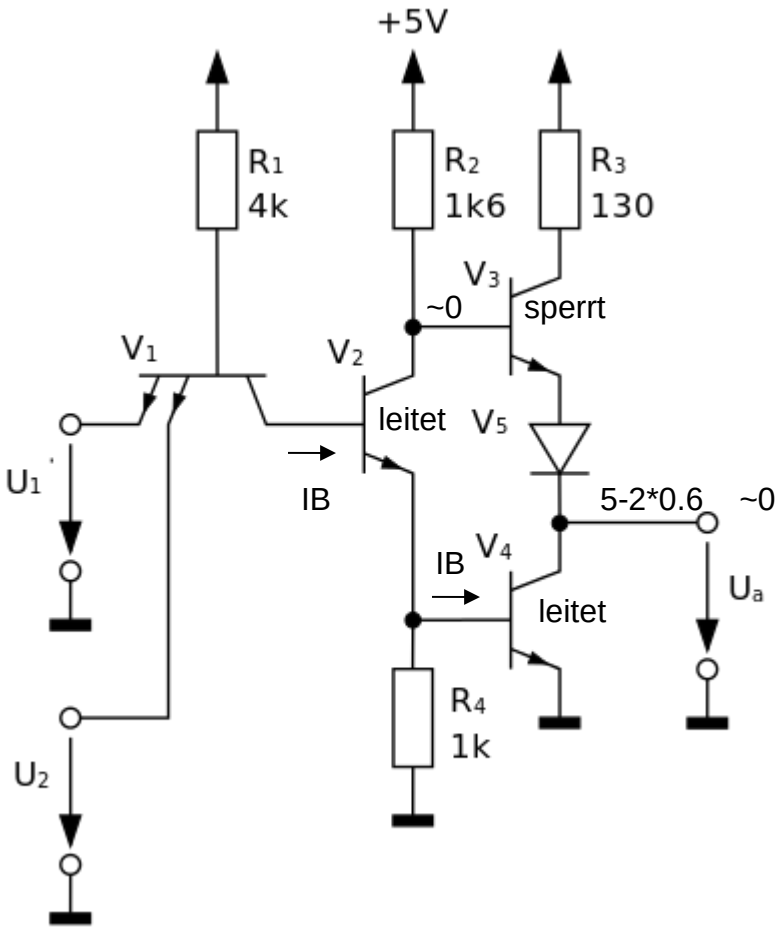
- TTL NAND



- TTL NAND
- U1 = 0

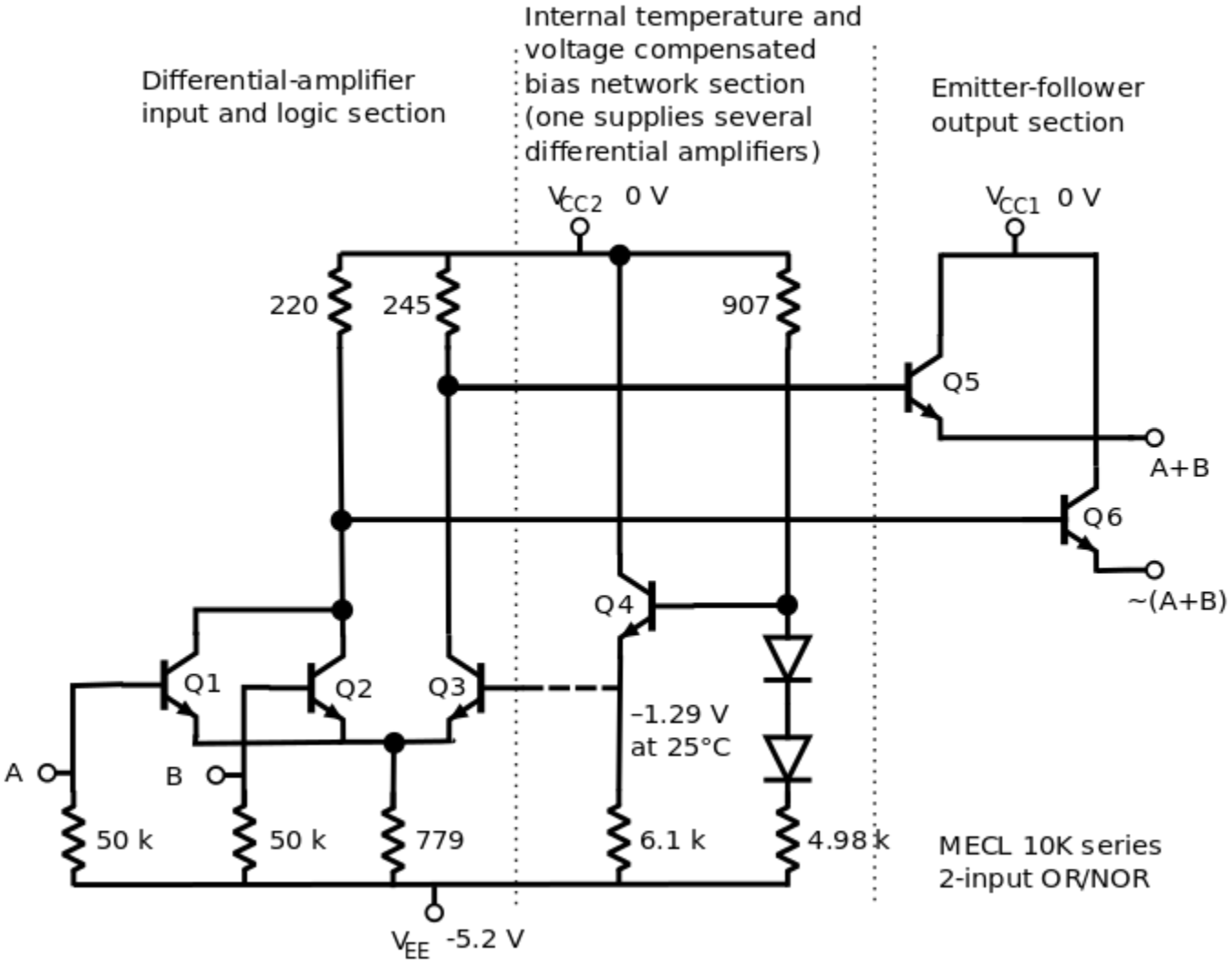

+5V
R1
4k
R2
1k6
R3
130
V3
leitet
V1
V2
0
sperrt
V5
U1
0
5-2*0.6
V4
0
sperrt
Ua
R4
1k
U2

- TTL NAND
- U1 = 1; U2 = 1

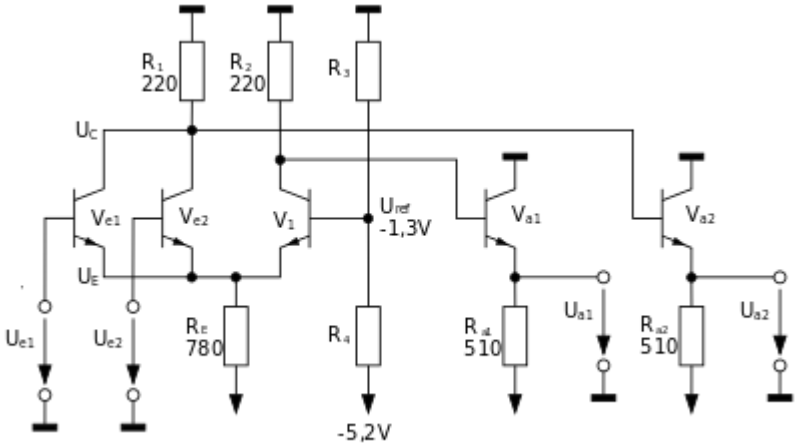


- Logische Bausteine in TTL-Technik haben gegenüber CMOS-Bausteinen den Vorteil, **dass sie unempfindlicher gegenüber elektrostatischen Entladungen sind.** Der Nachteil liegt wegen der stromgesteuerten Transistoren in einer im Vergleich zu CMOS deutlich höheren **Leistungsaufnahme** (Stromverbrauch) bei statischem Betrieb.
- Eine Besonderheit von TTL-Schaltungen besteht darin, dass an Eingängen jedes Potential zwischen 0 V und 5 V liegen darf und sie daher auch **unbeschaltet bleiben dürfen**, ohne dass untolerierbar große Querströme entstehen. Eine Besonderheit einer diskret aufgebauten TTL-Schaltung besteht darin, dass unbeschaltete Eingänge wirken, als lägen sie auf High-Pegel.

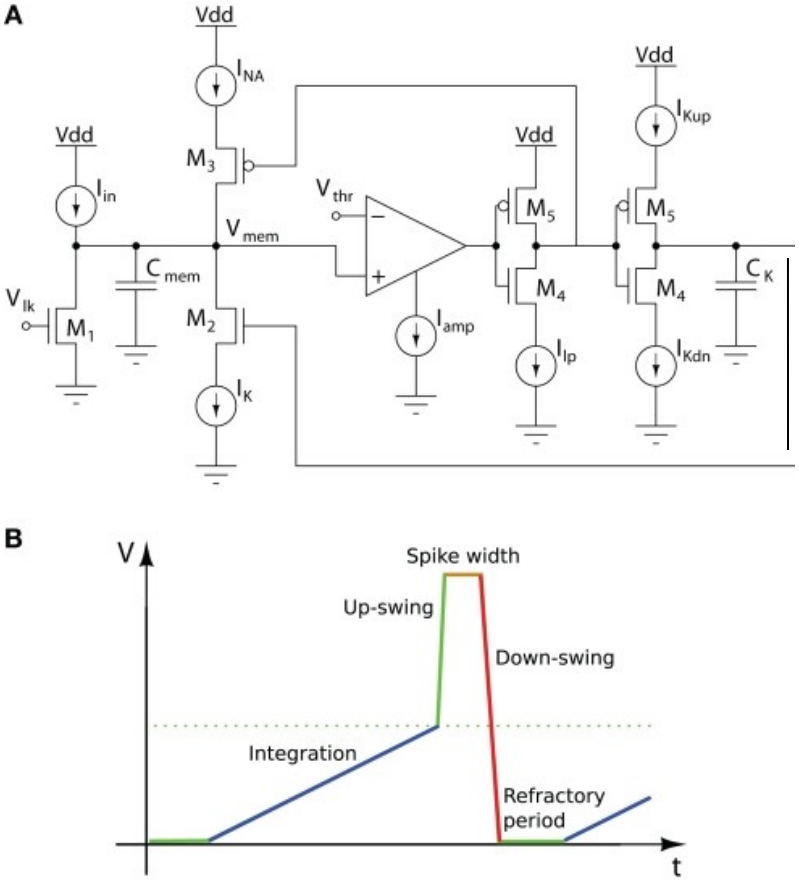
- ECL


Differential-amplifier input and logic section
Internal temperature and voltage compensated bias network section (one supplies several differential amplifiers)
Emitter-follower output section
V_{CC2} 0 V
V_{CC1} 0 V
220
245
907
Q5
A+B
Q6
~(A+B)
Q4
Q1
Q2
Q3
–1.29 V at 25°C
A
B
50 k
50 k
779
6.1 k
4.98 k
MECL 10K series 2-input OR/NOR
V_{EE} -5.2 V

- Die ECL-Familie gehört zu den schnellsten erhältlichen Logikfamilien. Dies wird erreicht, da (anders als zum Beispiel bei der Transistor-Transistor-Logik) im normalen Betriebszustand kein Transistor in Sättigung geht.



- Neuromorphic silicon neuron circuits



- https://www.youtube.com/watch?v=ZoT82NDpcvQ
- https://www.youtube.com/watch?v=S-4Yu3pB_dk