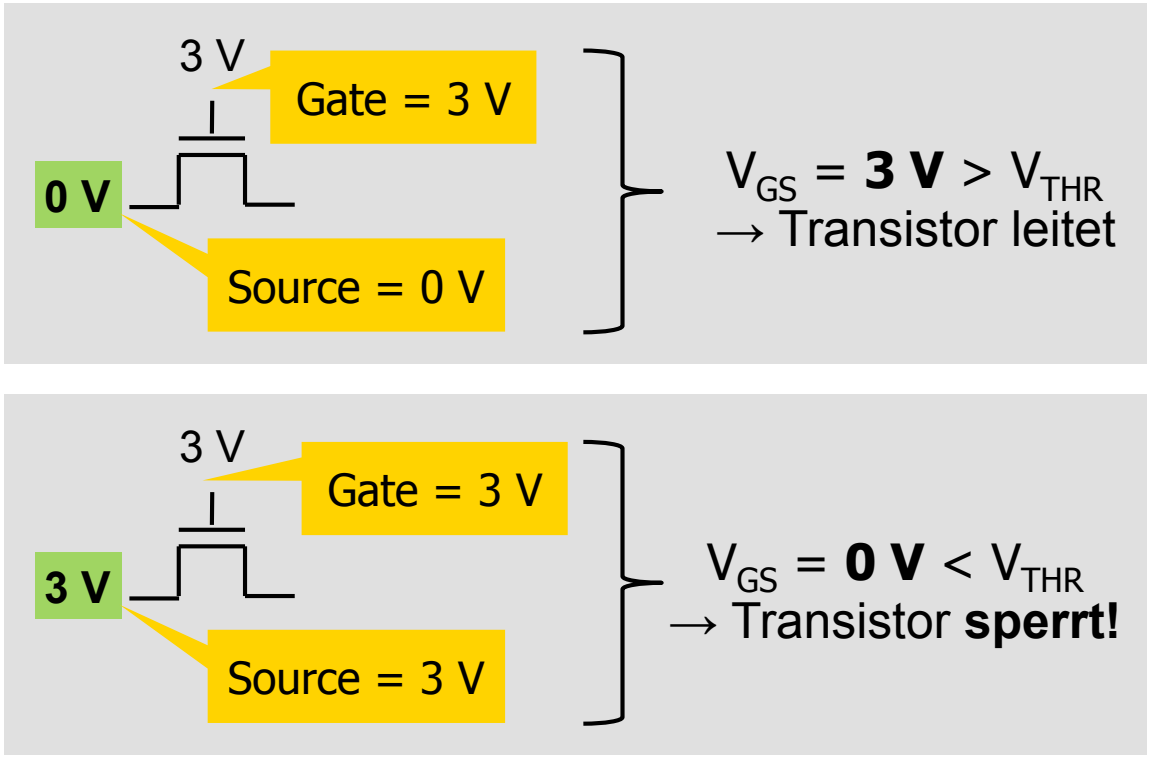
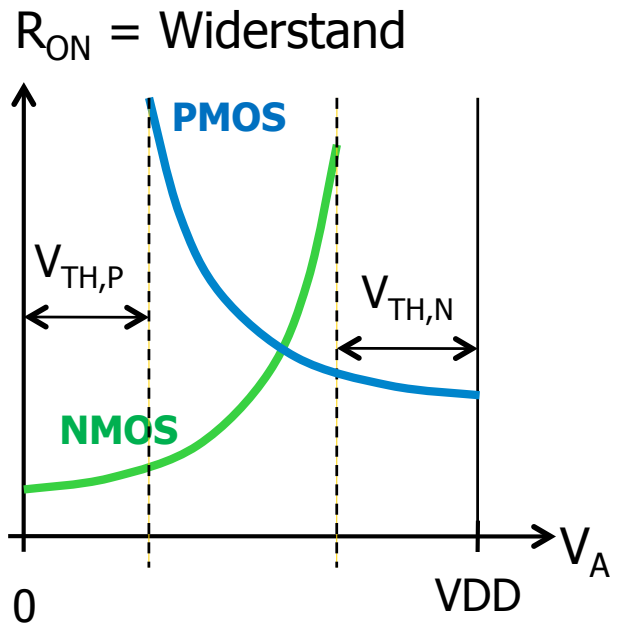
Einfache Schaltungsblöcke

Transmission Gates, Gesteuerte Inverter, Multiplexer, Decoder, Addierer, Latches und Flipflops

Der Transistor als Schalter

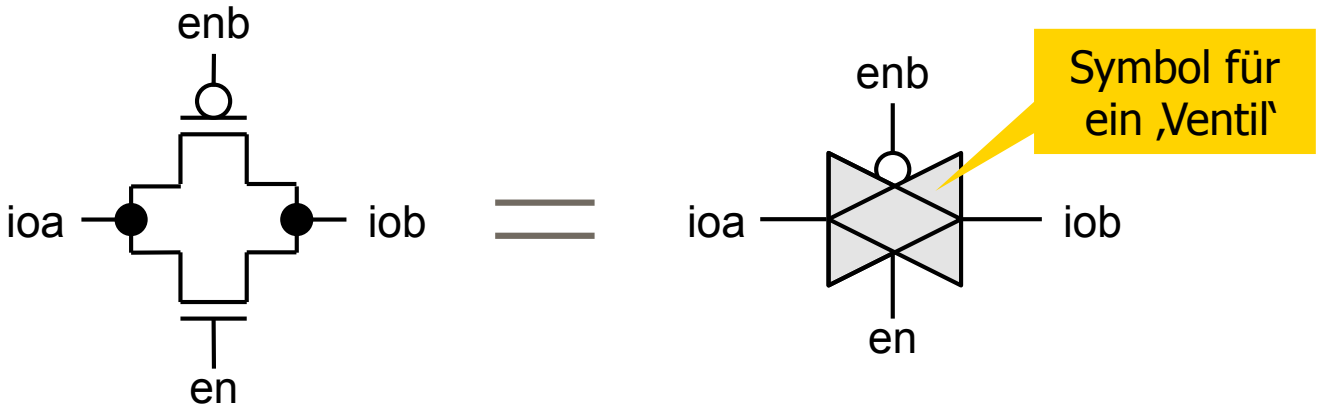
- NEIN: Ein NMOS ist nur für ‚niedrige' Spannungen ein ‚guter' Schalter:





Transmission Gate

- Die **Parallelschaltung** aus einem NMOS und einem PMOS bildet ein **Transmission Gate** (TGATE)



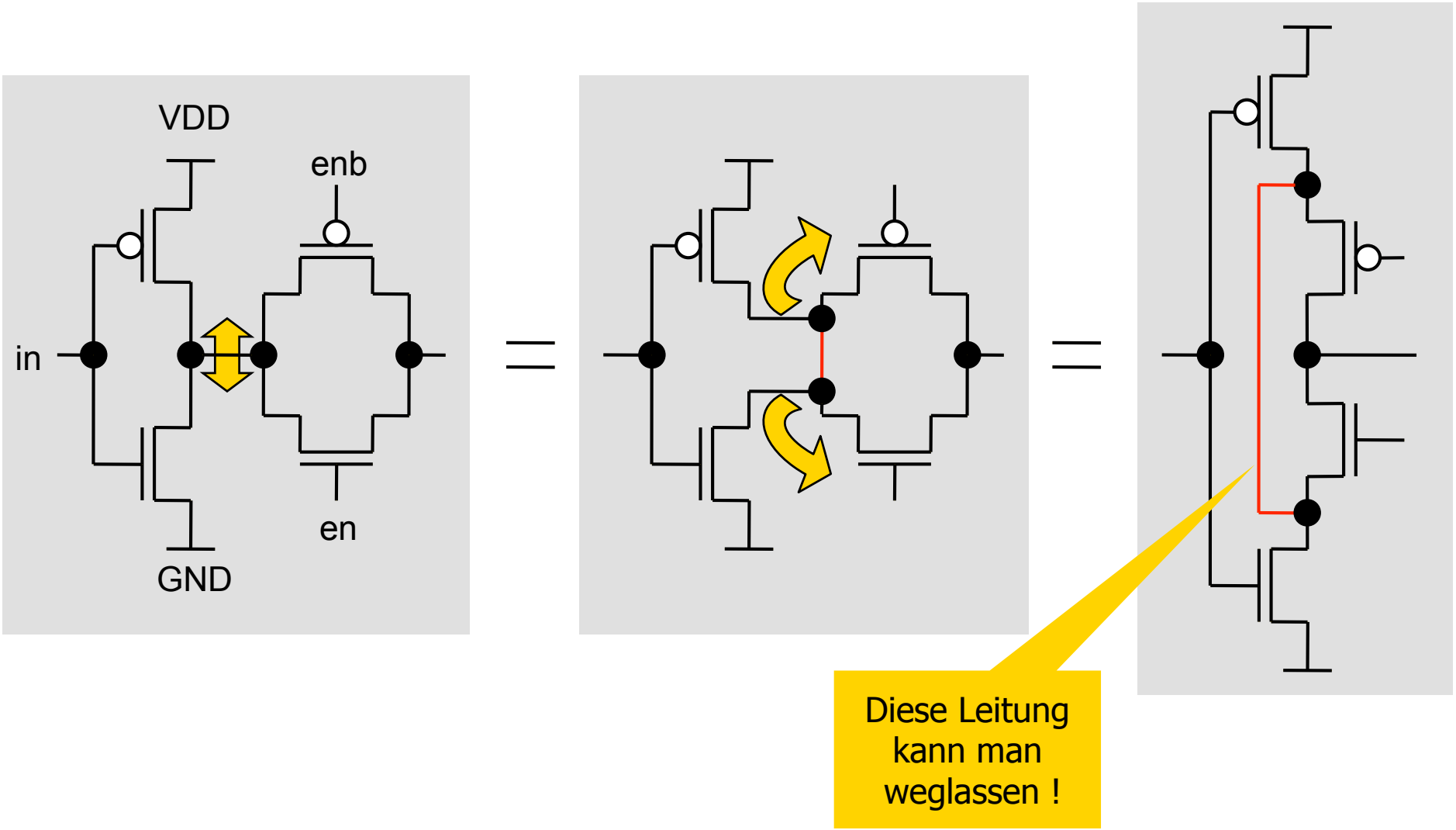
- Es 'lässt Einsen und Nullen gleichermaßen durch', d.h. der Durchgangswiderstand ist für alle Spannungen *relativ* gleich (das stimmt eigentlich nicht… mehr später).
- Nochmal: ein einzelner NMOS kann wegen der Schwellenspannung keine Spannungen in der Nähe von VDD durchlassen, ein einzelner PMOS kann Spannungen in der Nähe von Masse nicht durchlassen!

- Vergleich mit dem Gated Inverter (s. nächste Folie):
  Vorteil: weniger Transistoren
  Nachteil: keine Regeneration des Signals

- NB: Trotz der halben Zahl von Transistoren ist das Layout oft nicht viel kleiner!

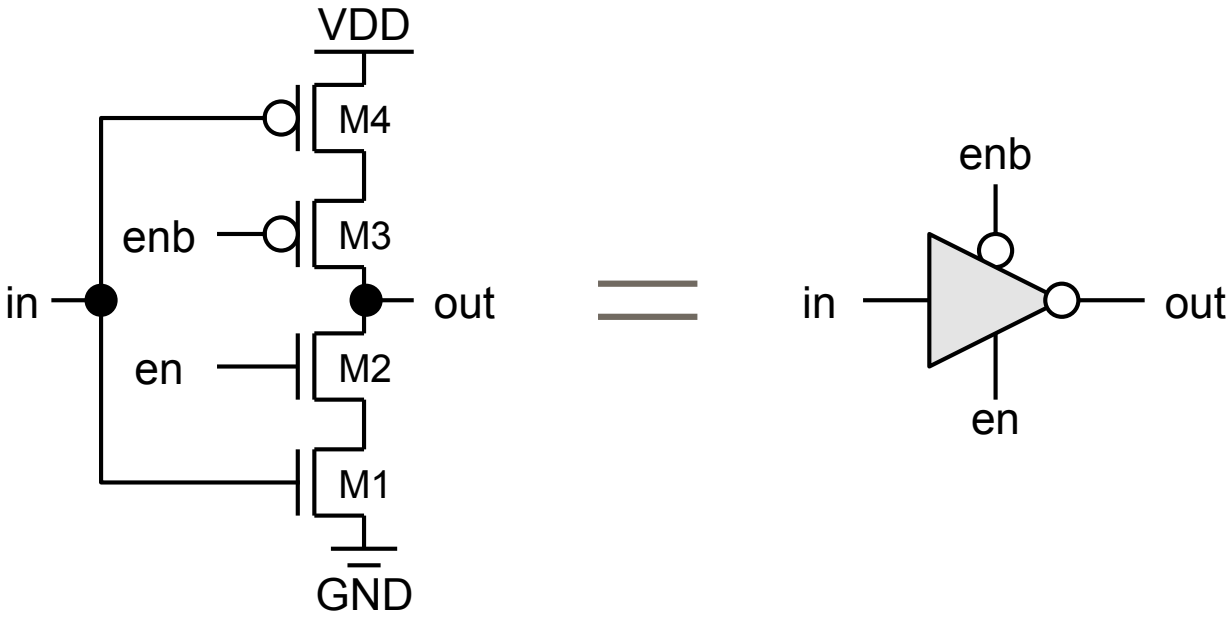
Inverter + Transmission Gate

- Topologische Umformung:



Gesteuerter Inverter

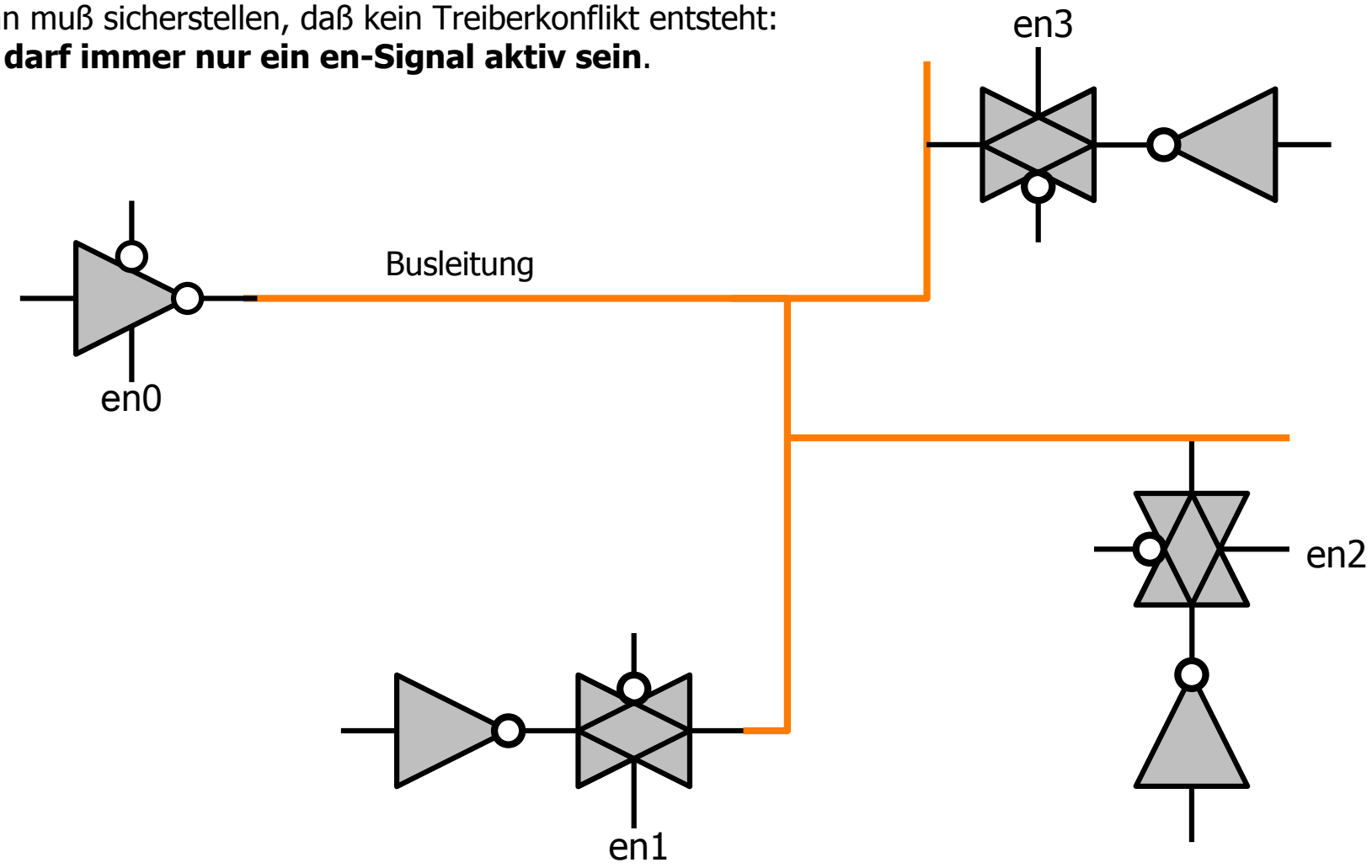
- Durch Weglassen der überflüssigen Leitung erhält man den gesteuerten Inverter (‚Gated Inverter'):



- Das Steuersignal wird oft mit **en** für 'enable' (aktivieren) bezeichnet
- en=1 und enb=0: Die Schaltung verhält sich wie ein **Inverter** (mit etwas längerer Durchlaufzeit...)
- en=0 und enb=1: PMOS M3 und NMOS M2 sperren ⇒ der Ausgang ist abgetrennt ('Tri State').
  Durch seine Kapazität behält der Ausgang seinen Zustand bei.
  Der Zustand kann von anderen Signalquellen verändert werden.
- Anwendung:
  - Dynamische Schaltungen
  - Busse (mehrere Verbraucher bedienen abwechselnd eine Leitung)
  - Multiplexer

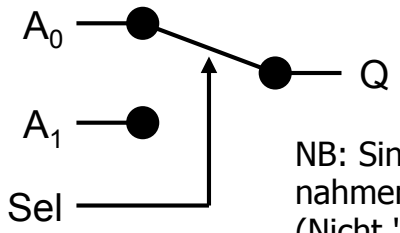
Tristate Ausgänge

- Soll auf einem **Bus** eine von mehreren verteilten Signalquellen ausgewählt werden, so ist es bei verteilten Quellen vorteilhaft den Ausgang abschalten zu können ('high impedance', 'tristate', 'Z')
- Mehrere Quellen können dann abwechselnd auf ein Netz treiben.
- Man muß sicherstellen, daß kein Treiberkonflikt entsteht:
  Es darf immer nur ein en-Signal aktiv sein.



Multiplexer (MUX)

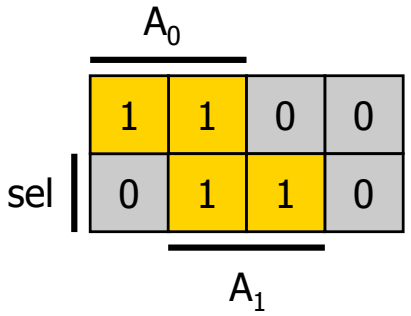
- Wichtige Grundschaltung zur Auswahl eines Signals aus 2 (oder mehreren) Eingängen:



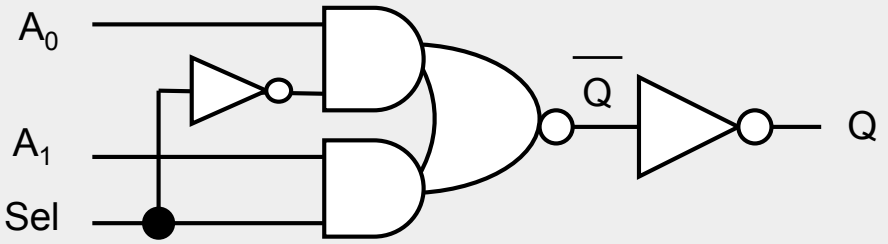
NB: Sinnvolle Signalnahmen benutzen! (Nicht 'a' und 'b' für die Eingänge!)

Sel	Q
0	A_0
1	A_1

Sel	A_0	A_1	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

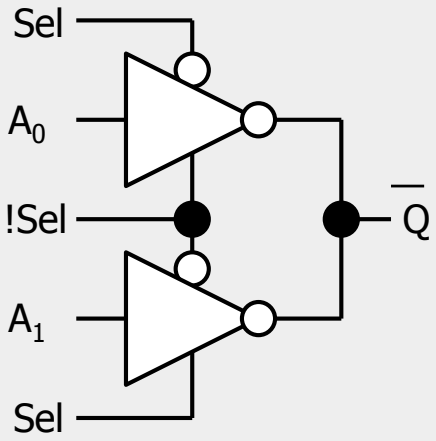


Funktion: $Q = \overline{Sel} \cdot A_0 + Sel \cdot A_1$

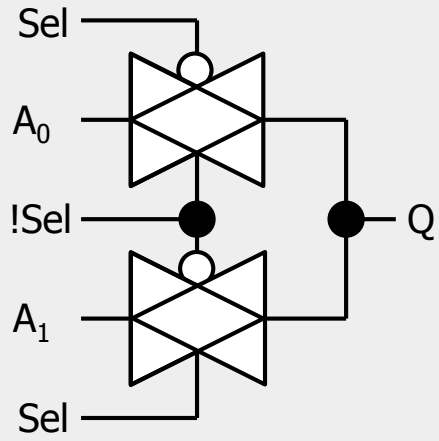


Mit gemischtem Gatter für !Q: 8(+2 für !Sel) Transistoren
2(1) Durchlaufzeiten

MUX mit Gated Invertern und Transmission Gates



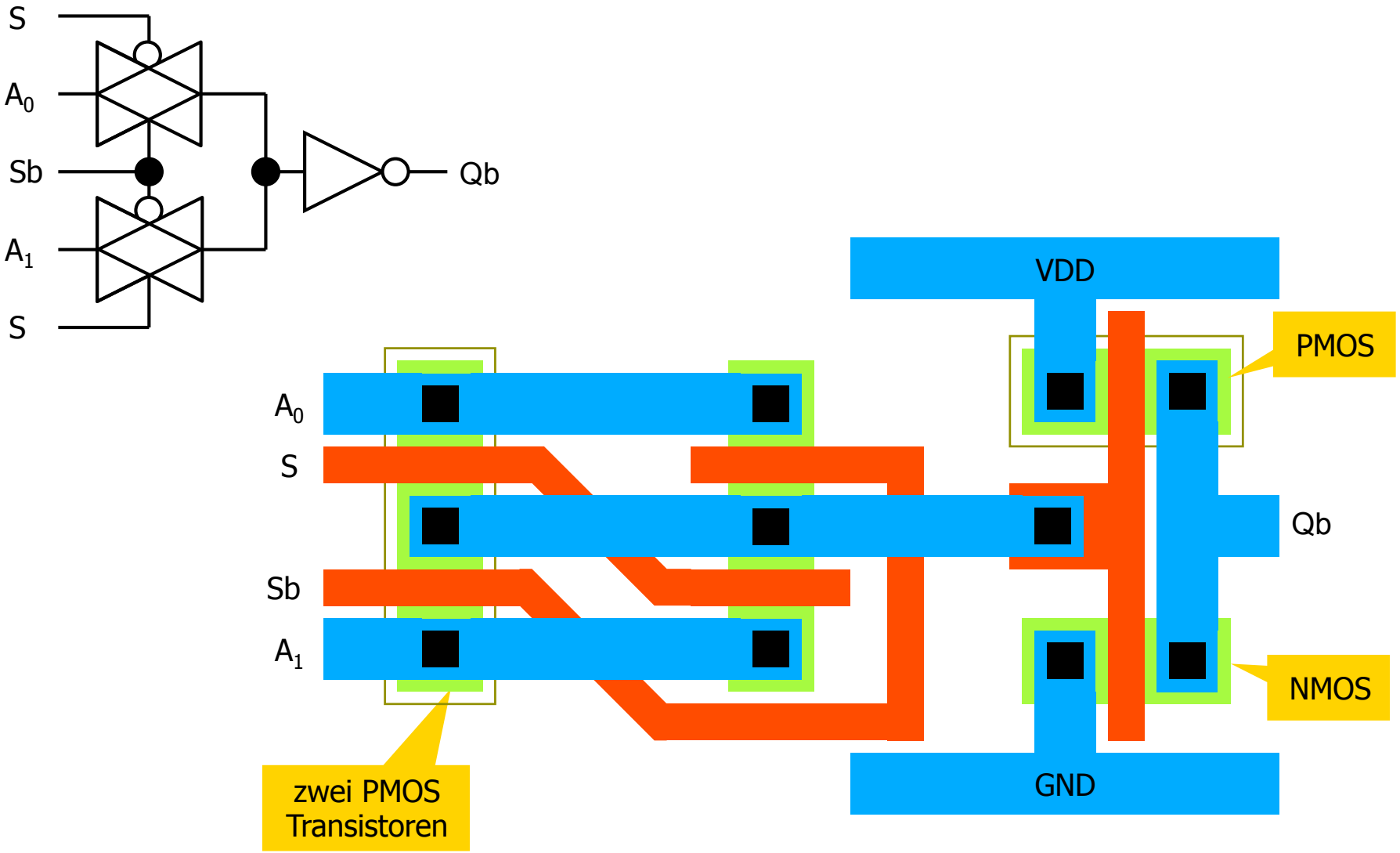
Mit Gated Inverter:
8(+2) Transistoren
1 Durchlaufzeit



Mit TGate: 4(+2) Trans.,
1 Durchlaufzeit
keine Regeneration!

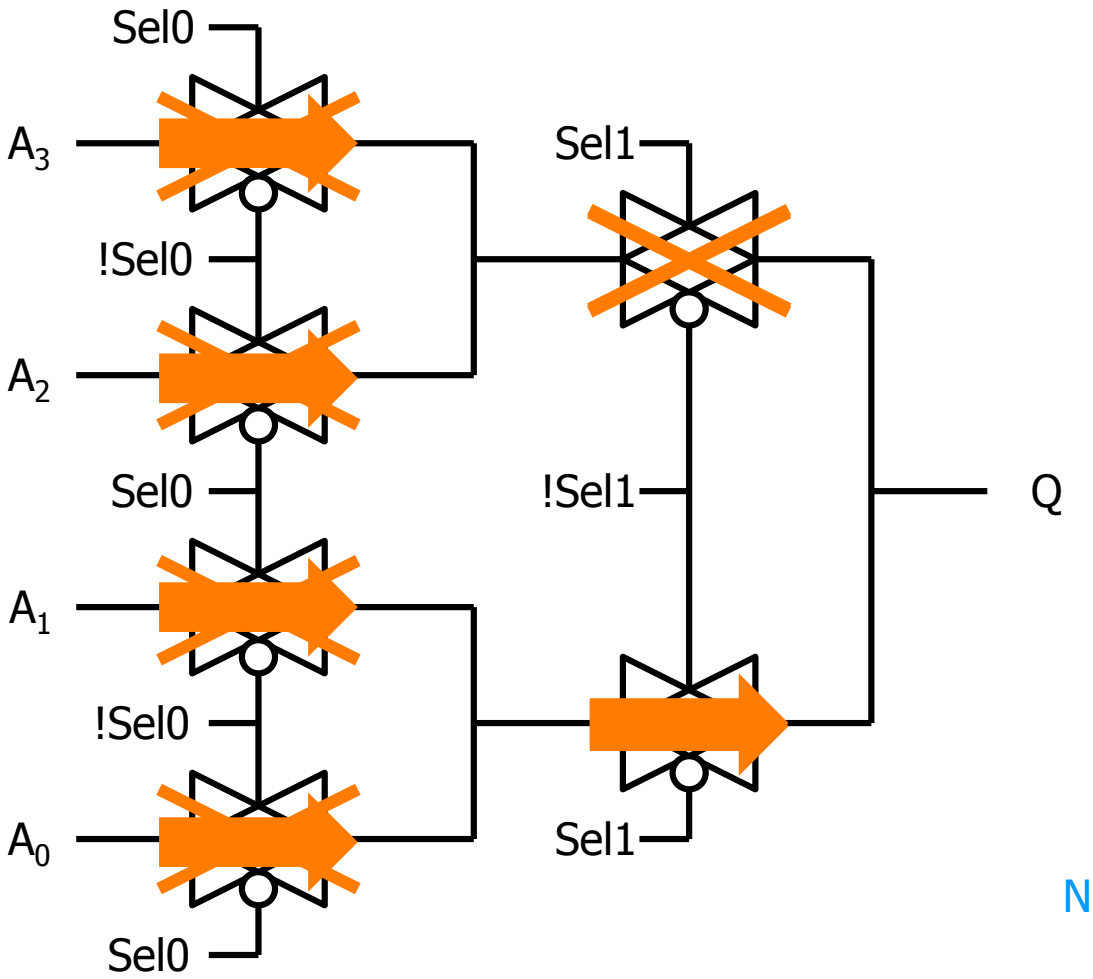
- Weniger Transistoren, meist kürzere Durchlaufzeit.
- Häufig benutzt !

Layout TGATE-MUX mit Inverter

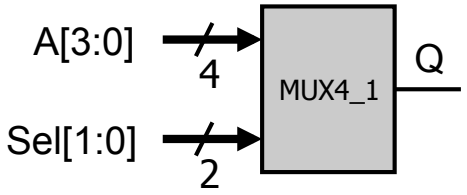


Breite Multiplexer

- Implementierung 1: Einfache Topologie durch kaskadierte 2:1 Multiplexer mit TGATEs



Sel1	Sel0	Q
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3

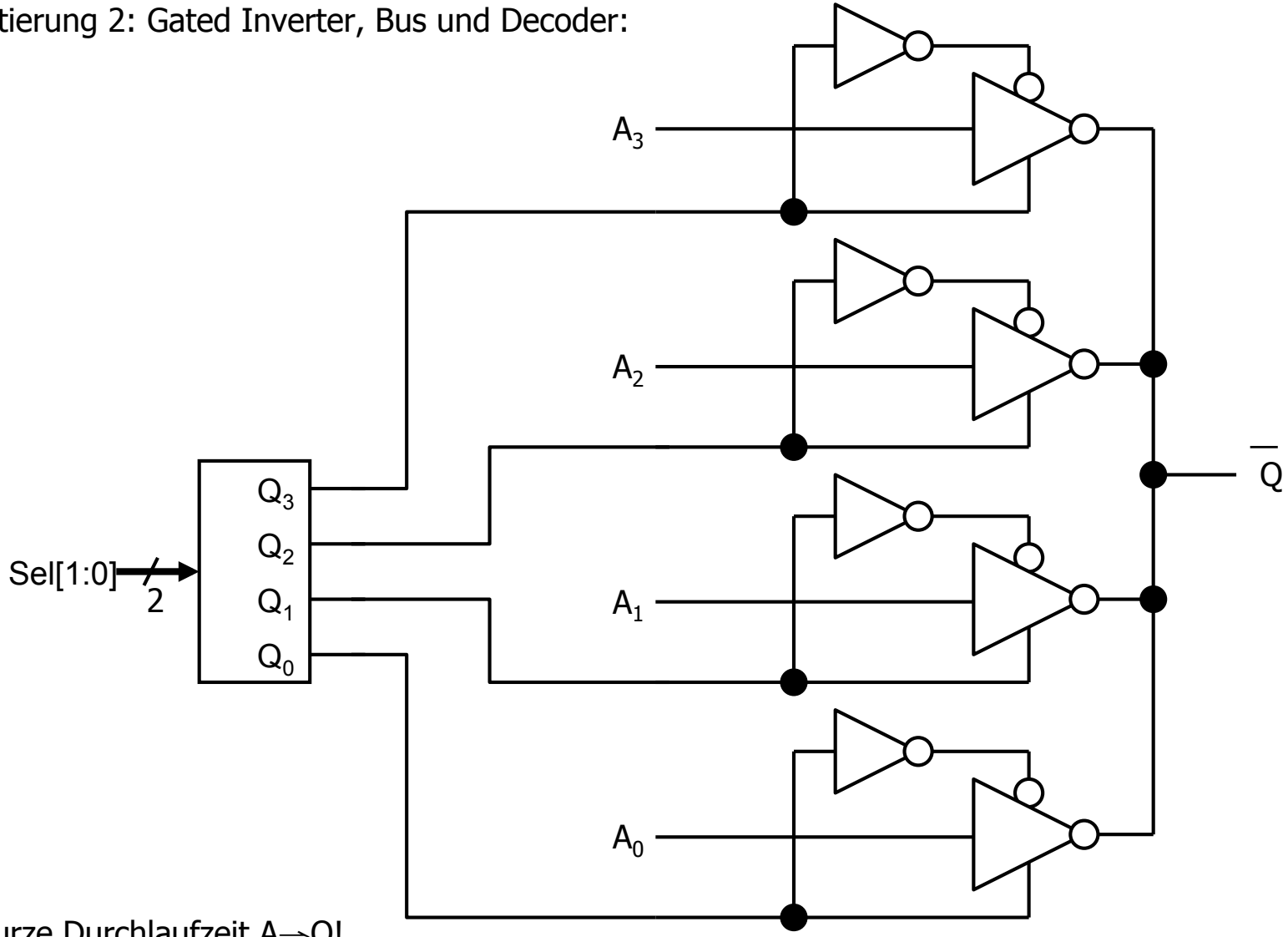


Nur 12 Transistoren für 4 Bit !
(+4 für 2 Inverter)

- Achtung: Zu viele (>4) in Serie geschaltete Transmission Gates führen zu hohen RC-Verzögerungen!

Breite Multiplexer

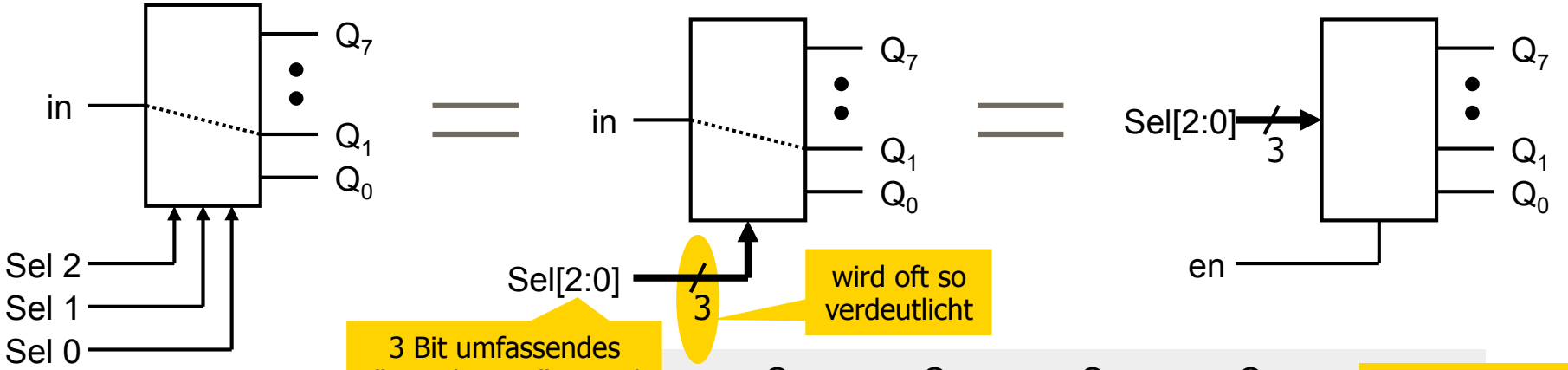
- Implementierung 2: Gated Inverter, Bus und Decoder:



- Vorteil: Kurze Durchlaufzeit A⇒Q!
  (Achtung: Bei breitem MUX wird Buskapazität an Q zu groß!)

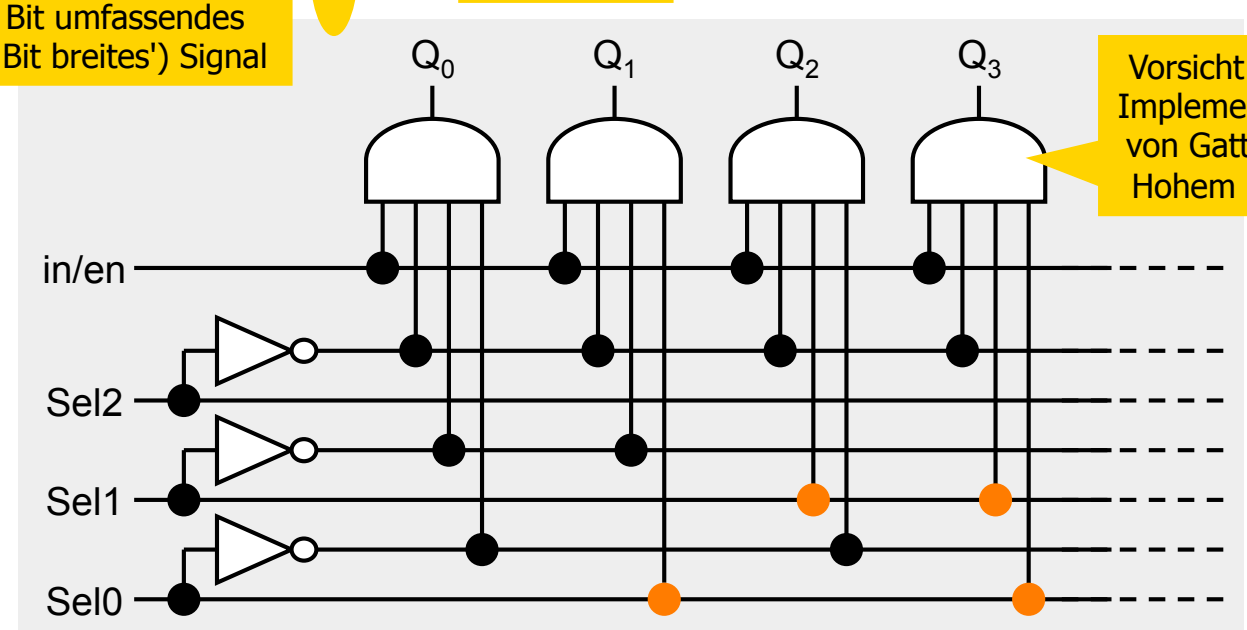
Demultiplexer, Decoder

- Ein Demultiplexer verteilt **ein** Signal 'in' auf einen von **N Ausgängen** $Q_0..Q_{N-1}$.
- Für in = 1 stellt er einen **Binär ⇒ 1 von N Decoder** dar. 'in' wird dann sinnvollerweise mit 'en' bezeichnet



3 Bit umfassendes ('3 Bit breites') Signal

wird oft so verdeutlicht



Vorsicht bei der Implementierung von Gattern mit Hohem Fan-In!

Funktionen :

$$Q_0 = in \cdot \overline{Sel0} \cdot \overline{Sel1} \cdot \overline{Sel2}$$

$$Q_1 = in \cdot Sel0 \cdot \overline{Sel1} \cdot \overline{Sel2}$$

$$Q_2 = in \cdot \overline{Sel0} \cdot Sel1 \cdot \overline{Sel2}$$

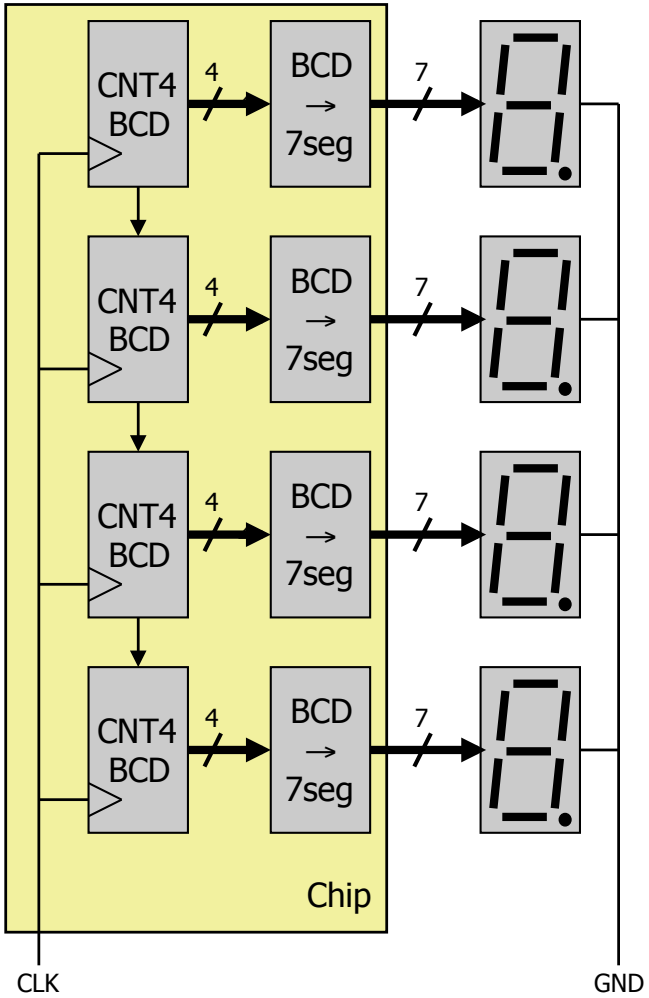
…

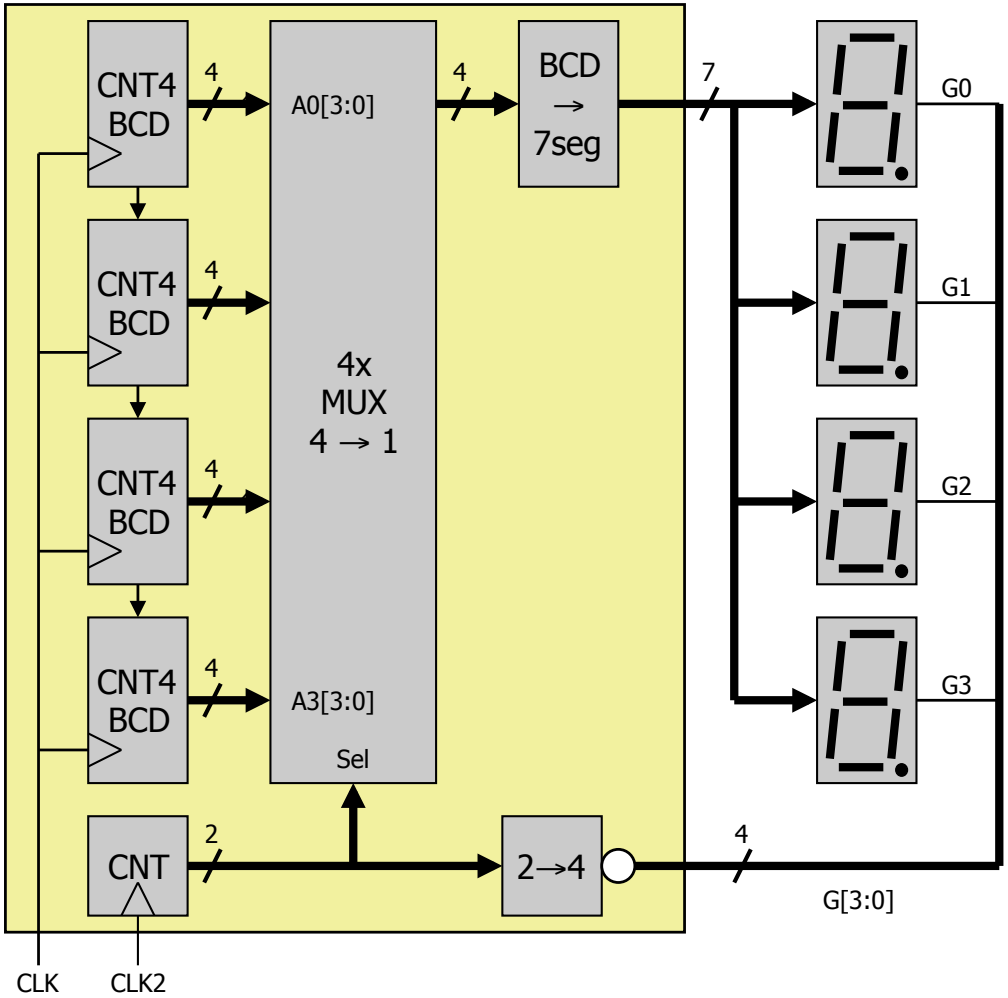
Decoder / Encoder

- Wandeln verschiedene Codes ineinander um.
- Beispiele:
  - Demultiplexer = ‚One-hot'
  - BCD ⇒ 7-Segment
  - Gray ⇒ Binär
  - Binär ⇒ Gray
  - Thermometer ⇒ Binär
  - Tastatur ⇒ ASCII
  - ...
- Implementierung als allgemeine kombinatorische Logik

Anwendung: mehrstellige 7-Segment Anzeige

- Aufgabe: 4-stellige BCD Zahl auf Siebensegmentanzeigen darstellen



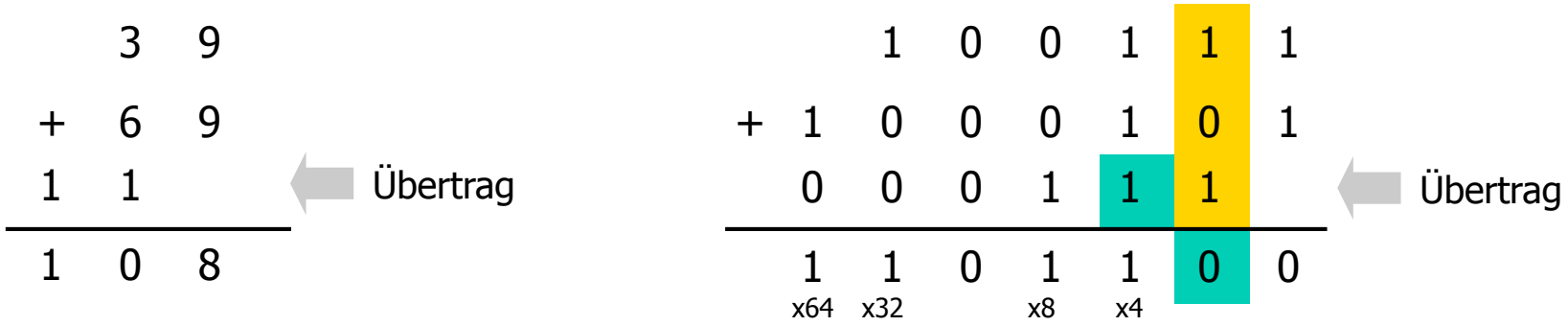


- Direkte Ansteuerung der Anzeige
- Nachteil: Viele Leitungen Chip ⇒Anzeige

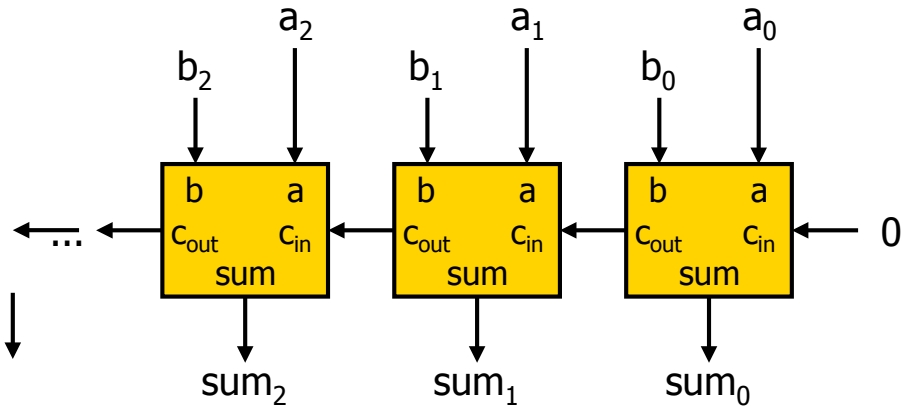
- Zeitliches Multiplexen. Nur 1 Decoder u. 11 Leitungen nötig.
- Nachteil: Anzeige nur ¼ so hell, flimmert u.U.

Addition von Binärzahlen

- Die Addition erfolgt stellenweise wie bei Dezimalzahlen mit einem **Übertrag (carry)**:



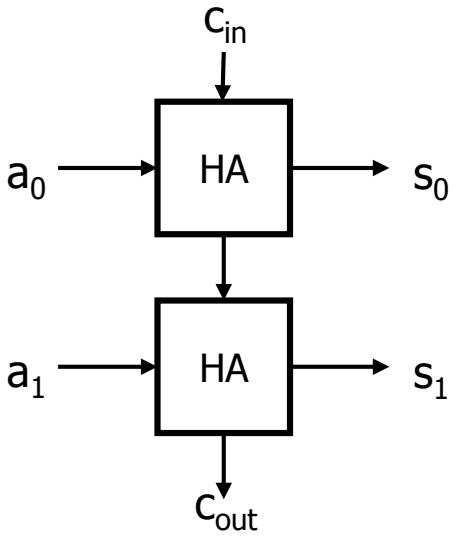
- In jeder Stufe werden also aus den **3 Eingängen** a,b,c_{in} die **Ausgänge** sum und c_{out} erzeugt.
- Man nennt diesen wichtigen Schaltungsblock den **Volladdierer** (full adder, FA):



c_{in}	b	a	c_{out}	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

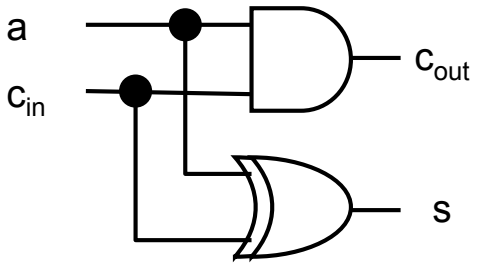
Halbaddierer

- Manchmal (z.B. in Zählern) muss NUR der Übertrag addiert werden.
- Der Addierer hat daher nur **einen** Dateneingang und einen Carry Eingang.
- Man nennt diesen Block einen Halbaddierer (Half-Adder, HA)



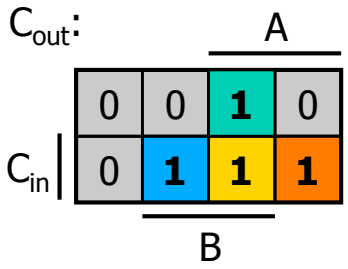
c_{in}	a	s	c_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$c_{out} = a \cdot c_{in}$$
$$s = a \oplus c_{in}$$

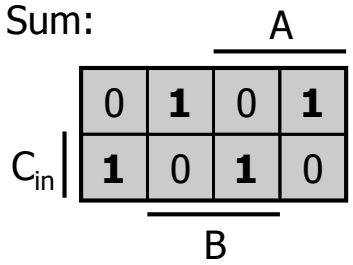


Volladdierer

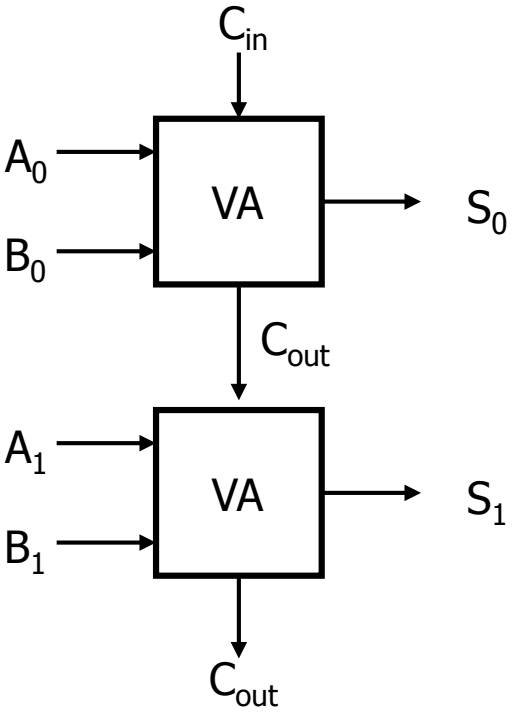
C_{in}	A	B	C_{out}	S	$!C_{out}$
0	0	0	0	0	1
0	0	1	0	**1**	1
0	1	0	0	**1**	1
0	1	1	**1**	0	0
1	0	0	0	**1**	1
1	0	1	**1**	0	0
1	1	0	**1**	0	0
1	1	1	**1**	**1**	0



$$C_{out} = AB + BC_{in} + AC_{in}$$
$$= AB + (A+B)\,C_{in}$$



$$S = A \oplus B \oplus C_{in}$$
$$= ABC_{in} + (A + B + C_{in}) \cdot !C_{out}$$

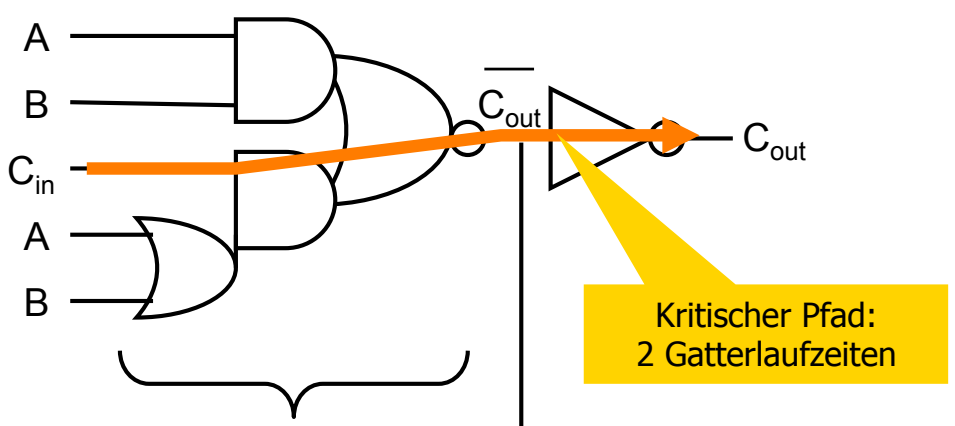


Der Carry-Pfad muß optimiert werden, da das Carry durch alle N Bit 'rippeln' muß

Trick: Carry-Ergebnis wird mitverwendet:
Mehrere Ebenen logische Tiefe:
'Multiple Output Minimization' (MOM).

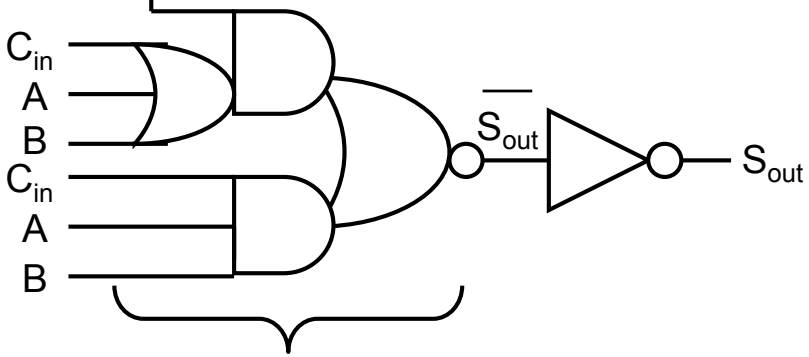
Volladdierer: Implementierung

- $C_{out} = AB + (A+B)\,C_{in}$:



Gemischtes Gatter 10+2 Transistoren

- $S = ABC_{in} + (A + B + C_{in}) \cdot !C_{out}$:



Gemischtes Gatter 14+2 Transistoren

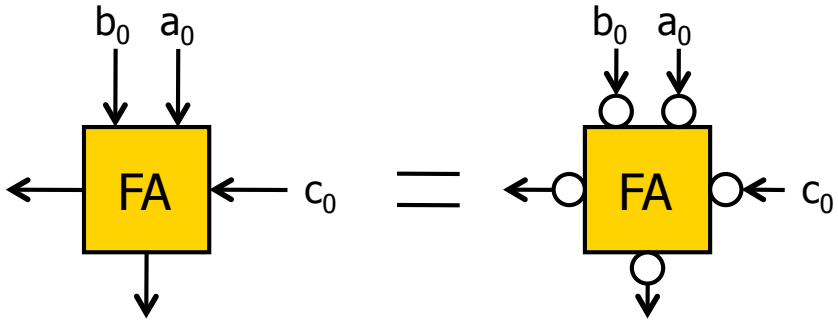
- Vorgriff: Eine offensichtliche Geschwindigkeitsverbesserung bekommt man, indem man $!C_{out}$ weitergibt und in jeder zweiten Stufe Duale Logik benutzt

Optimierung: Polaritätswechsel in jeder 2ten Stufe

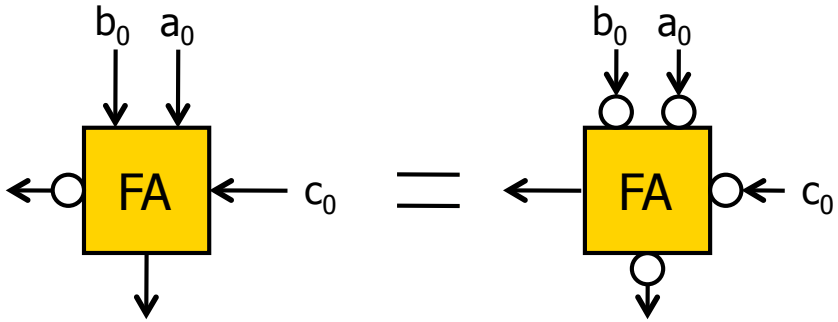
- Beobachtung: Bei Inversion ALLER Eingänge invertieren sich auch die Ausgänge:

C_{in}	A	B	C_{out}	S
0	**0**	**0**	**0**	**0**
0	**0**	**1**	**0**	**1**
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	**1**	**0**	**1**	**0**
1	**1**	**1**	**1**	**1**

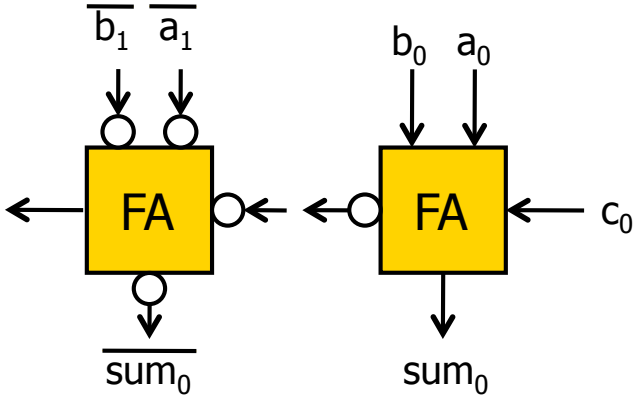
- Also:



- Oder:



- Daher kann man bei mehreren Stufen direkt FA mit invertierten Carry nutzen:

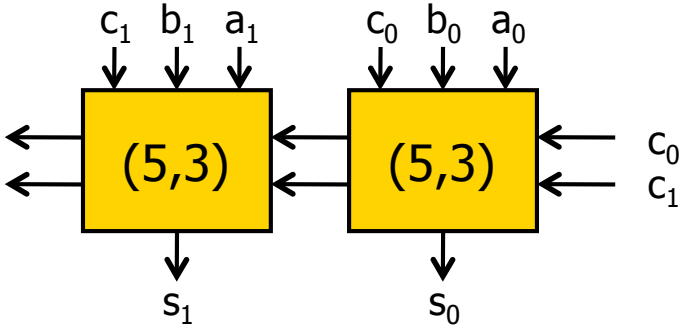


Bemerkung: Addierer als ‚Counter‘

- Die Aufgabe des Addierers ist es ja, die Anzahl EINSEN auf den Eingängen zu zählen und als Binärzahl auf Summe und Carry (höherwertiges Bit) auszugeben:

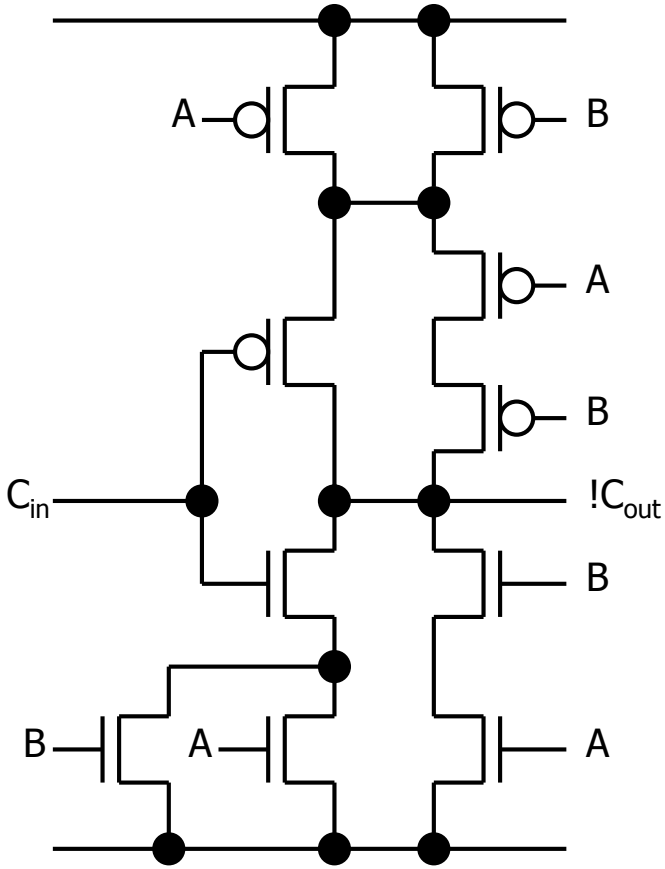
Anzahl Einsen	0	1	2	3
[CarryOut, Sum]	[0,0] = 0	[0,1] = 1	[1,0] = 2	[1,1] = 3

- Allgemein spricht man daher auch von einem ‚Counter‘ or ‚Compressor‘ mit N Eingängen und M Ausgängen
- (3,2) Counter: Bei N=3 Eingängen können maximal 3 Einsen vorkommen, es reichen M=2 Bit Das ist der Volladdierer.
- (4,3) Counter: Bei N=4 Eingängen benötigt man am Ausgang M=3 Bit.
  Wenn die **2** Carry Signale in eine weitere Stufe gegeben werden sollen, ‚verbraucht‘ man in der nächsten Stufe schon 2 der 4 Eingänge, das bringt also nichts (im Vergleich zum Volladdierer)
- (5,3) Counter: Hiermit kann man dann 3 Zahlen gleichzeitig addieren:
  (Solche Schaltungen werden in Multiplizierern benötigt)

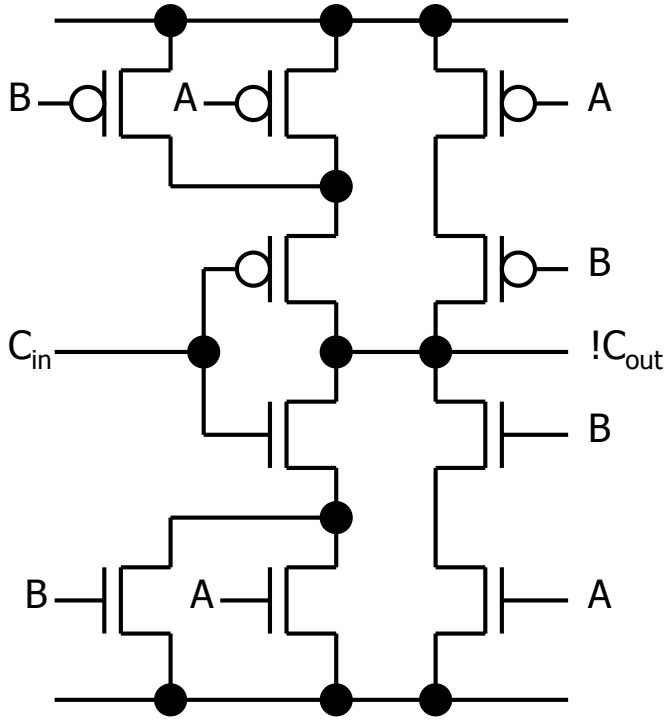


Transistorlevel: Carry-Erzeugung

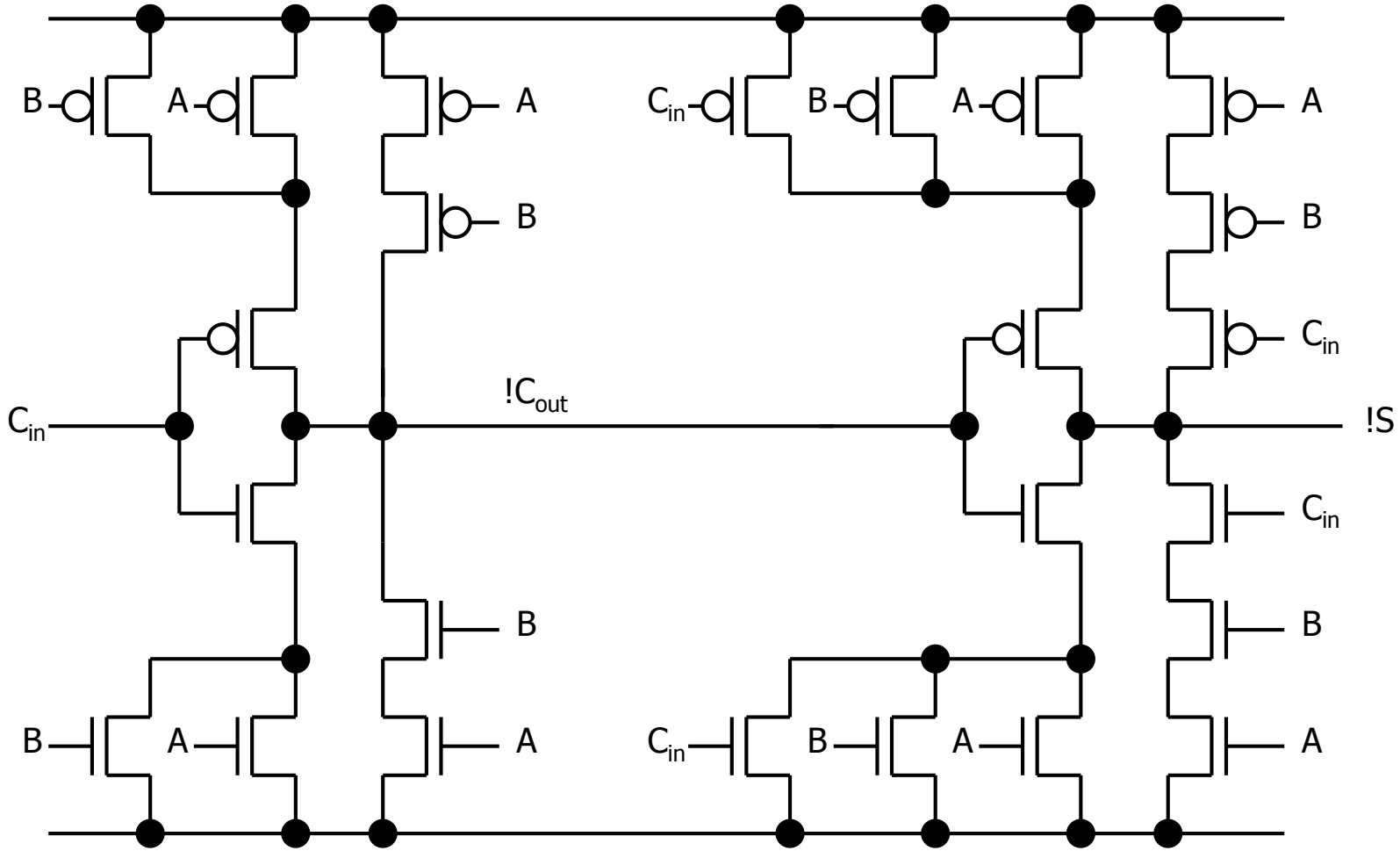
- Das Carry wird durch $C_{out} = AB + (A+B)C_{in}$ gegeben.
- Diese Funktion kann mit dem gemischten Gatter $Y = !(AB+(A+B)C_{in})$ implementiert werden
- Problem: 3 PMOS übereinander ('Stack height' = 3)
- Da je zwei der 5 Eingänge des Y-Gatters identisch sind, kommen nicht alle Eingangskombinationen vor!



- Der PMOS Zweig kann umgeformt werden:
  (!A+!B)(!A!B+!C) = !A(!A!B+!C) + !B(!A!B+!C)
  = !A!B+!A!C+!A!B+!B!C = !A!B + (!A+!B)!C.
- Das führt zu einem **NICHT-CMOS**-Gatter:



Optimierter Volladdierer (invertierte Ausgänge)



$!C_{out} = !(AB + (B + A)C_{in})$
$!S = !(ABC + !C_{out} \bullet (A + B + C_{in}))$

kein CMOS !

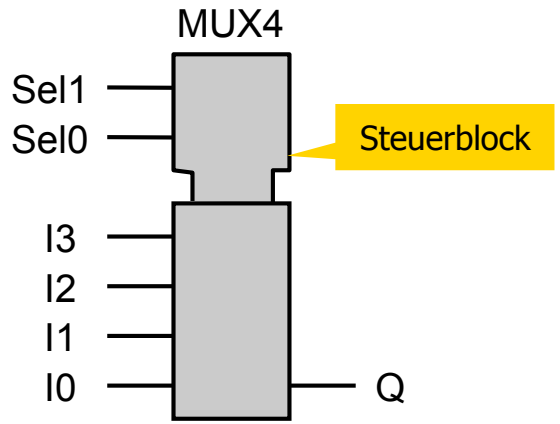
24 Transistoren

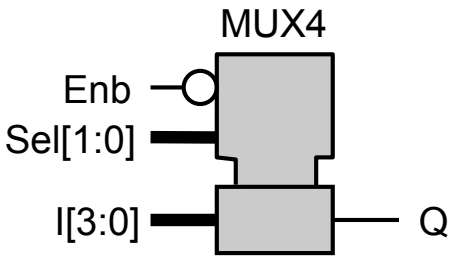
Schaltsymbole

- Schaltsymbole sind nach DIN und nach ANSI/IEEE normiert.
- Diese Normen werden aber oft nicht eingehalten....

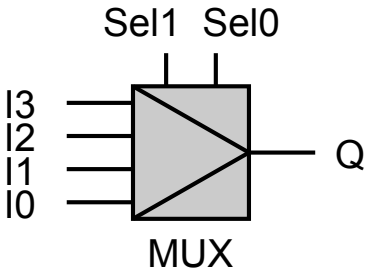
- Generell sollte gelten:
  - Eingänge links, Steuerung links, oben und unten, **Ausgänge rechts**
  - **Sinnvolle Namen** (C, CK, CLK, IN, I, D, Q, EN, SEL, ...)
  - **Bullets** (Kreise) an negierte Ausgänge, Dreieck an Takteingänge
  - Evtl. Busse mit breiter Leitung sichtbar machen, evtl. Breite angeben

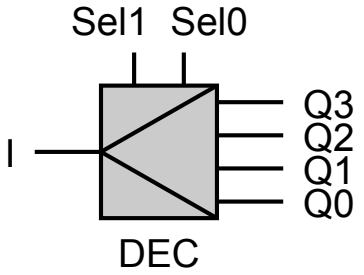
- Beispiele:





-> Rechnerarchitektur...



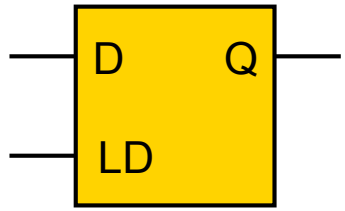


FlipFlops und Latches

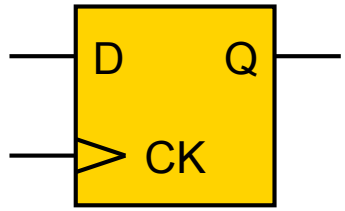
Speicherelemente: Latches und Flipflops

Wir unterscheiden (nicht immer konsistent…):

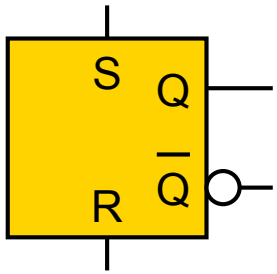
- **LATCH ('transparent latch'):**
  - Ist 'transparent' (d.H. Q=D) solange LD= 1
  - Ausgang behält letzten Zustand bei bei LD= 0



- **FLIP-FLOP ('edge triggered FF'):**
  - Eingang wird zum Ausgang bei der **Flanke** eines Taktsignals übertragen

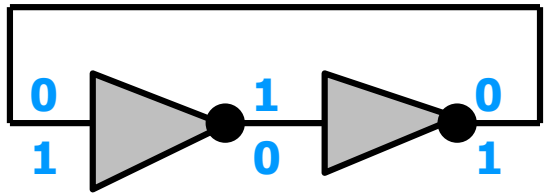


- **Set-Reset FLIP-FLOP ist eher ein LATCH:**
  - Zustand kann über zwei Eingänge auf 0 oder 1 gesetzt werden (es hat keinen Dateneingang)

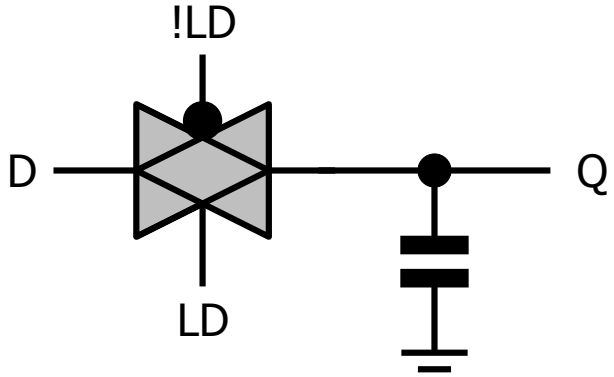


Speicherprinzipien

1. Positive Rückkopplung (statisch)
   – 2 Zustände
   + Durch die Verstärkung der Logik regeneriert sich das Signal, so daß kleine Störungen verschwinden

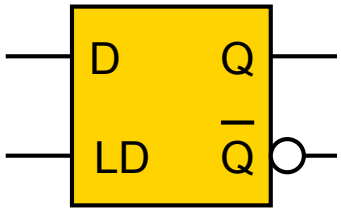


2. Ladungsspeicherung auf Kapazität (dynamisch)
   + Kontinuum an Zuständen (auch analoge Werte!)
   – Keine Regeneration
   – Signal bleibt nur begrenzte Zeit erhalten (Leckströme!)
   + sehr kompakte Schaltungen
   + speichert auch ohne Betriebsspannung!
   - benötigt Refresh



Statisches D – Latch

- Steuereingang LD ('load' = Laden), Eingang D, Ausgang Q (und !Q)
- LD=1: Ausgang = Eingang, LD=0: Zustand beibehalten, unabhängig von D



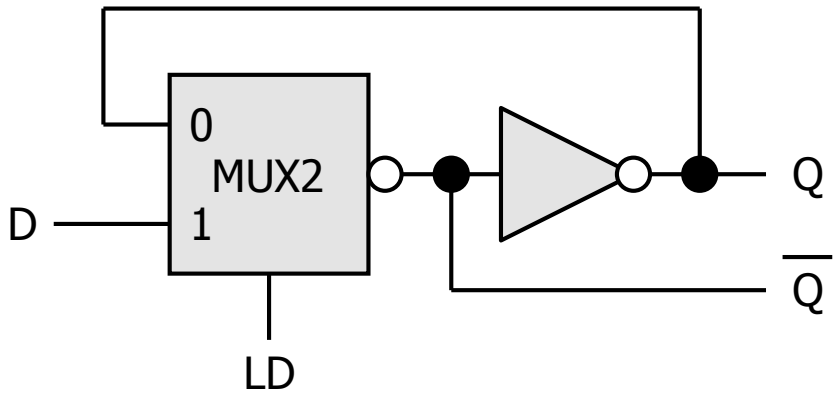
LD	D	Q	!Q
1	0	0	
1	1		0
		?	!?
0	1	?	!?

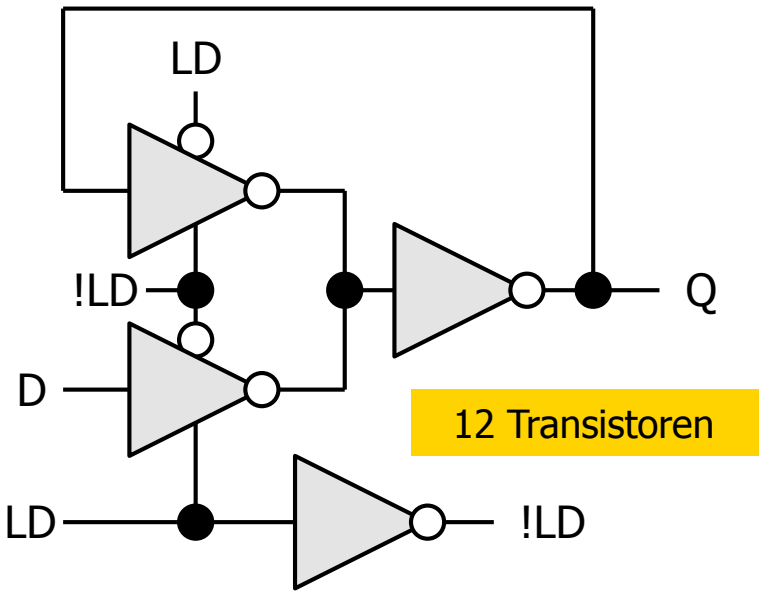
Nicht eindeutig!

D	Q_{n+1}
0	0
1	1

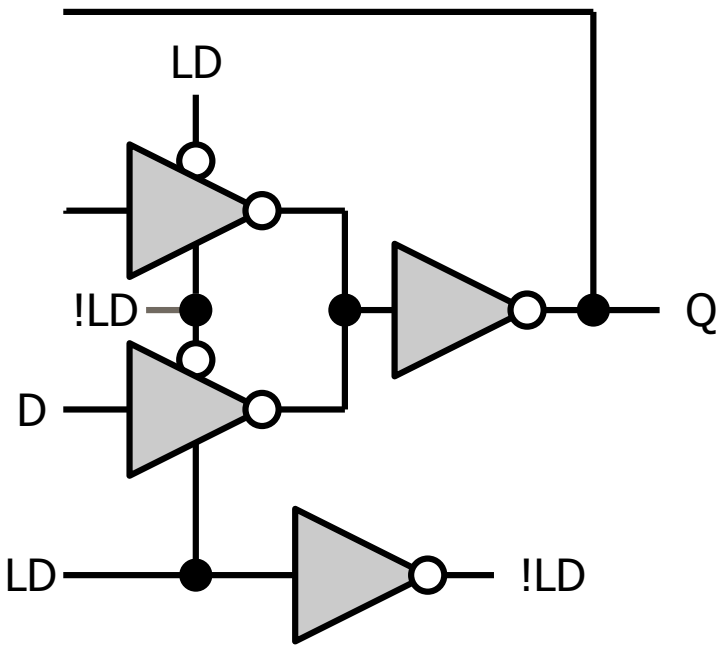
LD=⎍ $Q_n \rightarrow Q_{n+1}$

- Merke: Bei Speicherelementen reicht die 'normale' Funktionstabelle (Eingänge ⇒ Ausgänge) nicht aus!
- Implementierung z.B.:

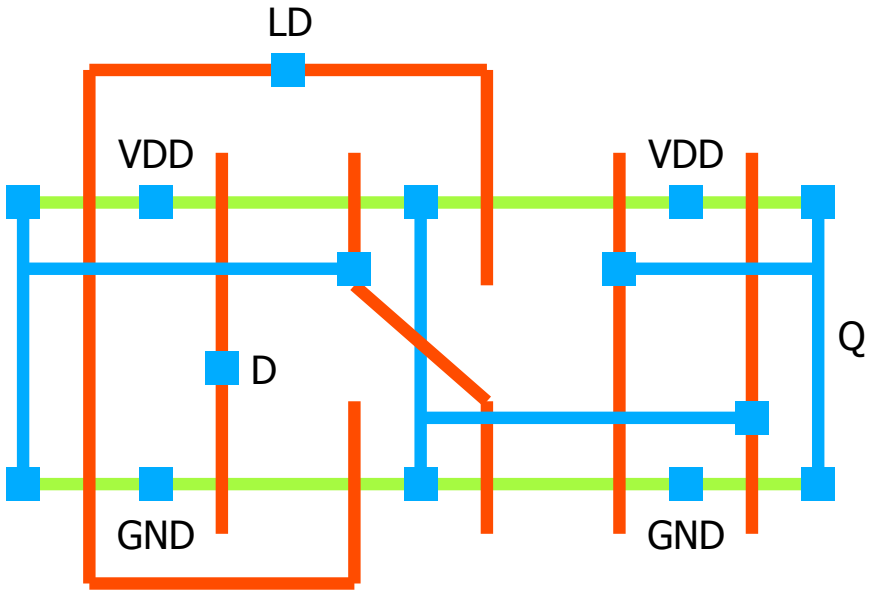




Vorgriff: Stick Diagramm D – Latch



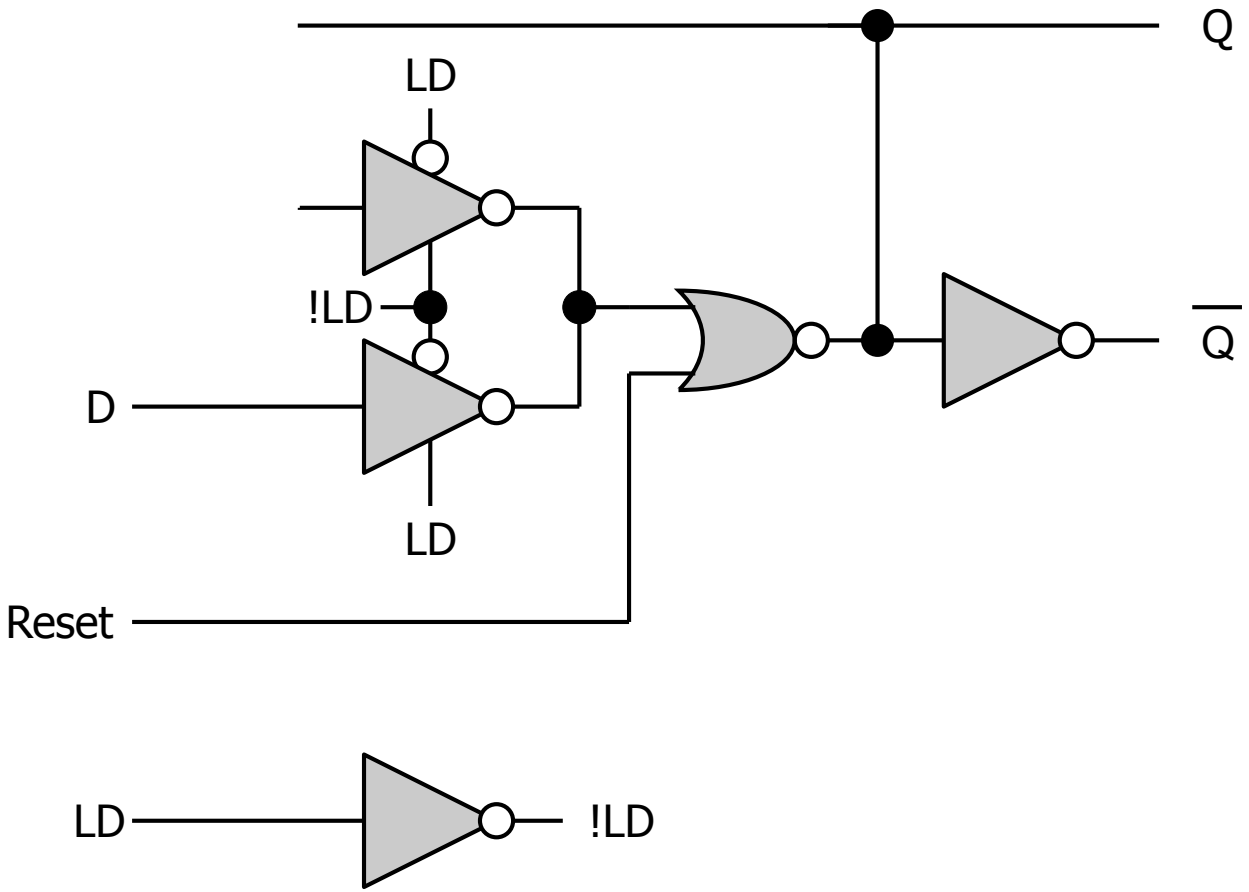
- Stick Diagramm:
  Zeichen nur **Diffusion**, **Gate**, **Metall**
- Transistor = Kreuzung von Diffusion und Gate
- Quadrat ist Kontakt



- Latch: Sehr kompakt
- Transistoren werden 'verschmolzen'
- Die 'extra' Transistoren des Gated-Inverters (im Vergleich zu einem Transmission Gate) fallen kaum ins Gewicht
- Problematisch im Layout sind immer die Überkreuzungen von LD/!LD bzw. EN,!EN

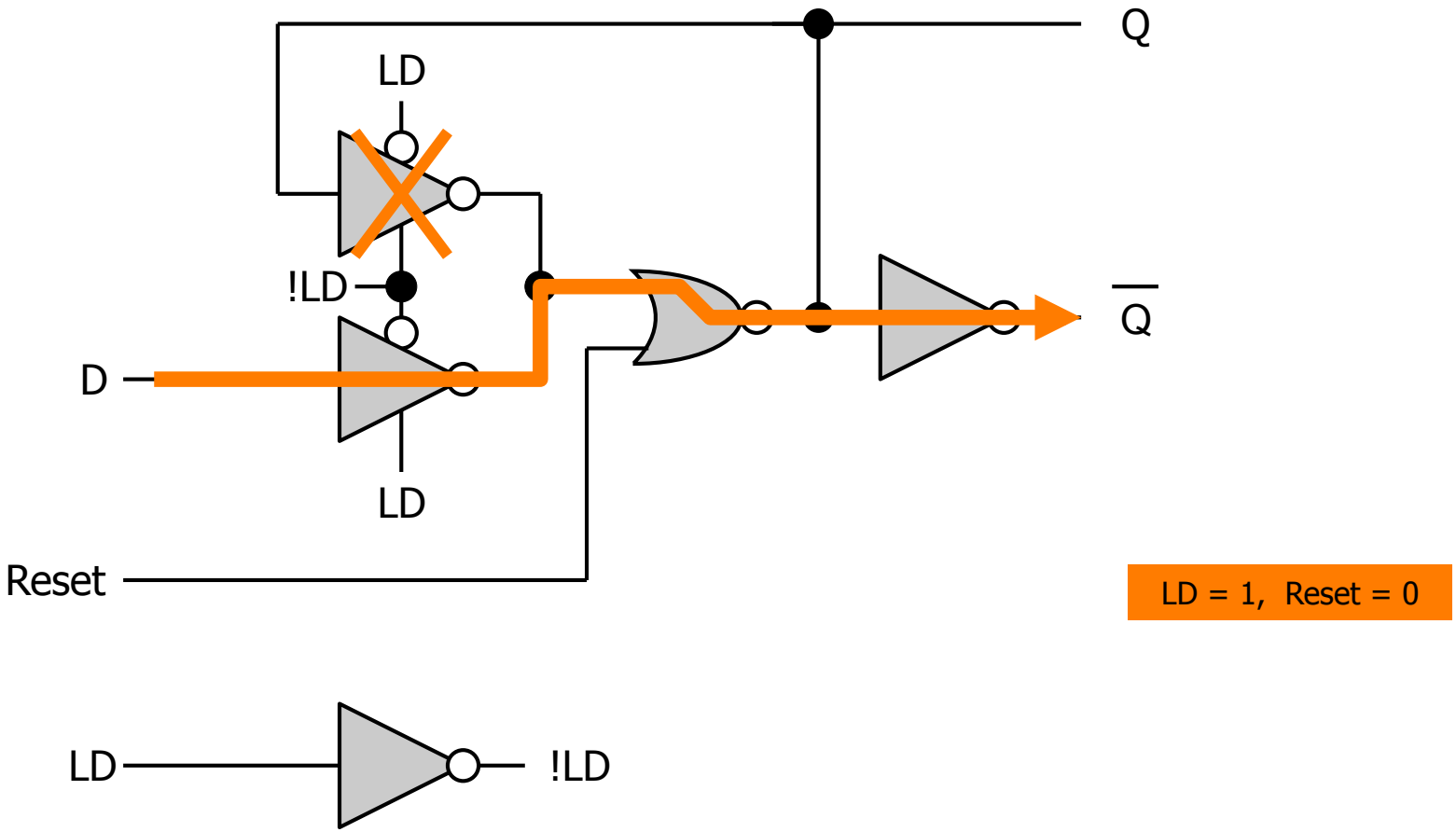
Set/Reset von Latches

- Latch mit Reset



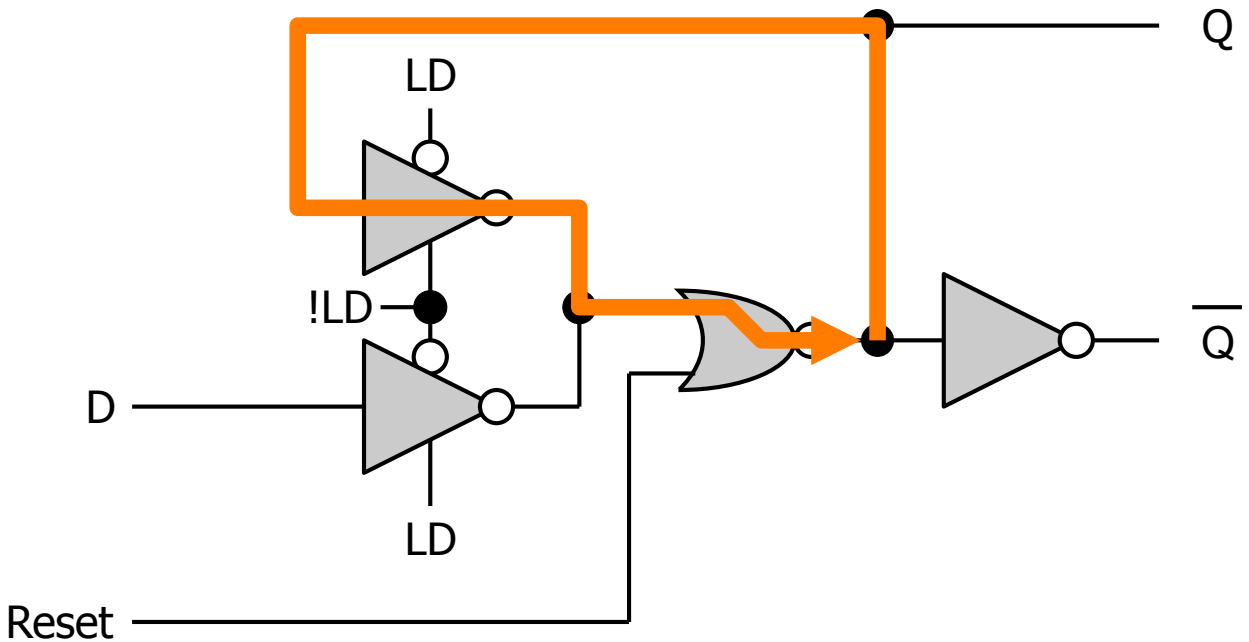
Set/Reset von Latches

- Latch mit Reset

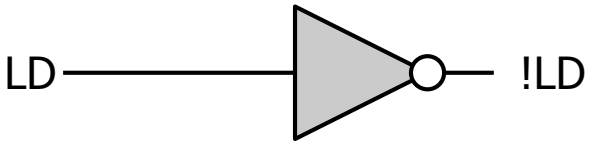


Set/Reset von Latches

- Latch mit Reset

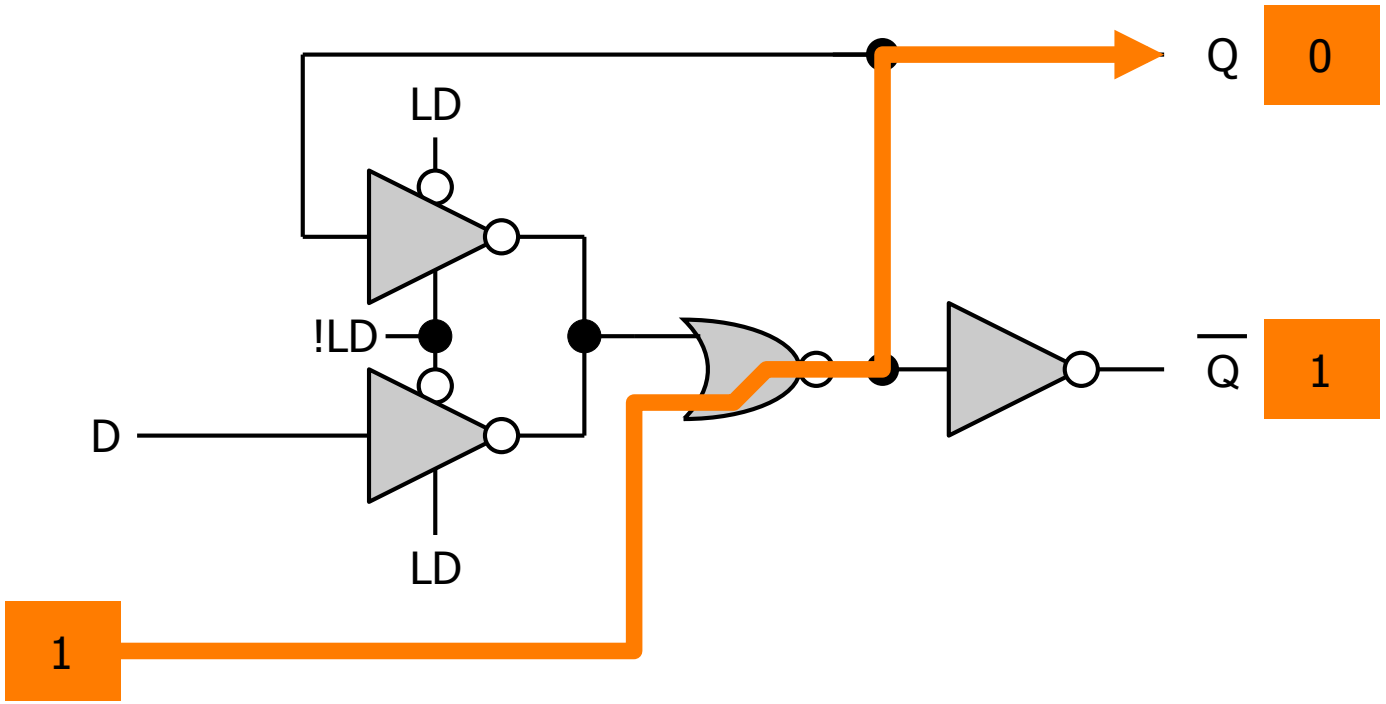


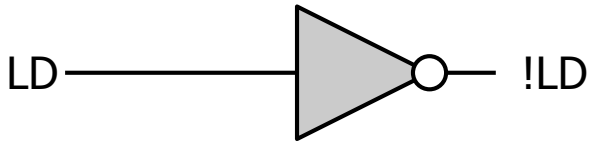
LD = 0, Reset = 0



Set/Reset von Latches

- Latch mit Reset

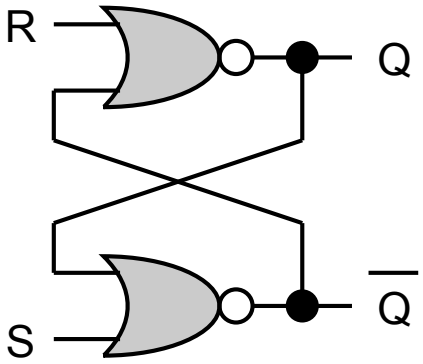




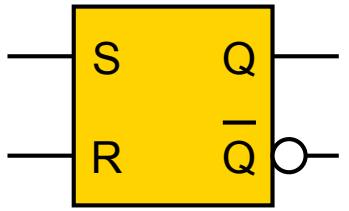
LD = X, Reset = 1

- Entsprechend für !Set (NAND2 statt NOR2)

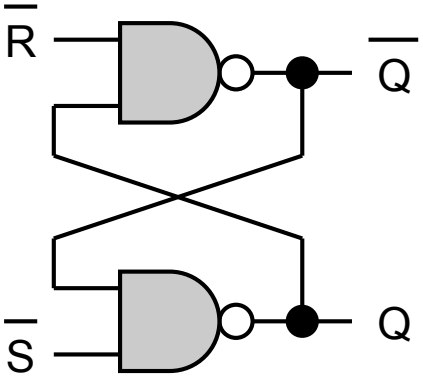
SR – FlipFlop



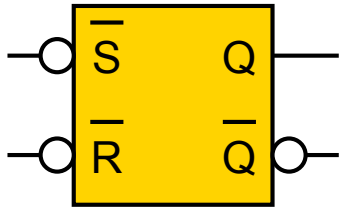
S	R	Q	!Q
0	0	Q	!Q
0	1	0	1
1	0	1	0
1	1	0	0



Diese Zustände muß man vermeiden!

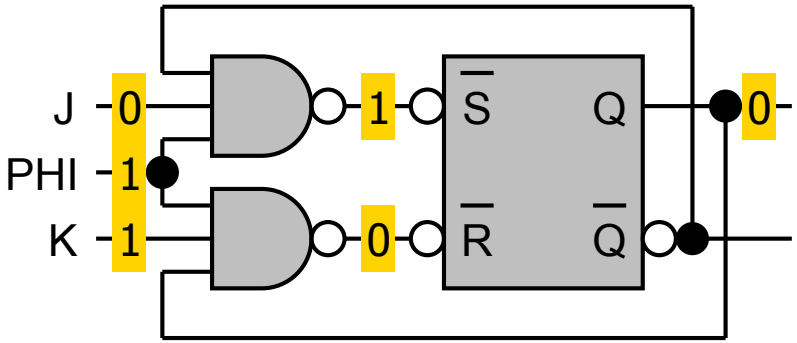


!S	!R	Q	!Q
0	0	1	1
0	1	1	0
1	0	0	1
1	1	Q	!Q

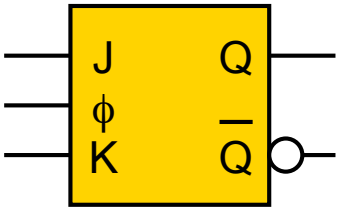


JK – 'Flip Flop' (... eigentlich Latch...)

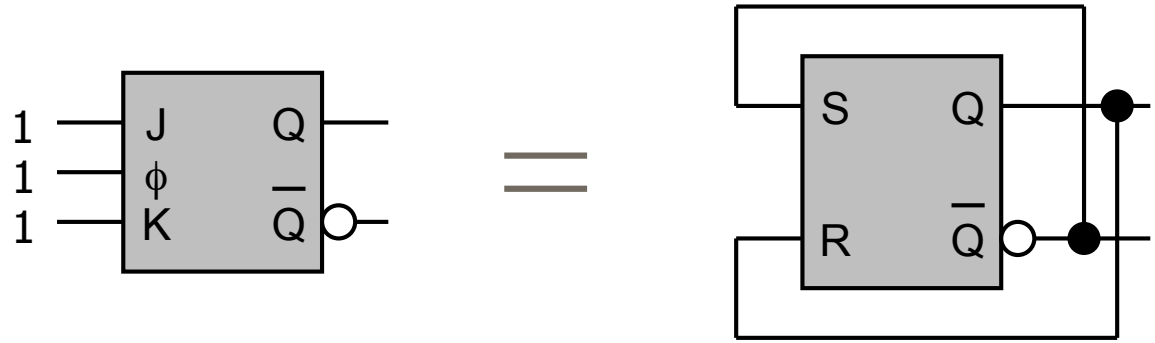
- Hat einen 'Takteingang' PHI, der den Zustand Q_n in den Zustand Q_{n+1} überführt
- Solange PHI=0 ist, wirken sich Änderungen an J und K nicht auf Q aus.
- Verhindert den 'unerlaubten' Zustand des SR-FlipFlops ...



J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$!Q_n$

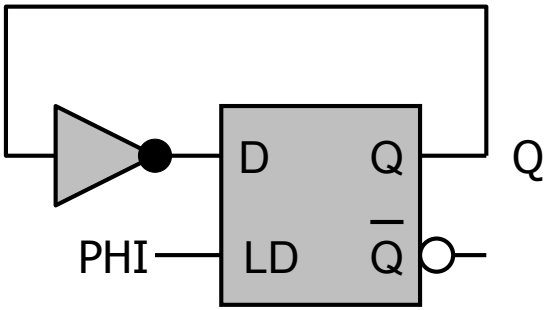


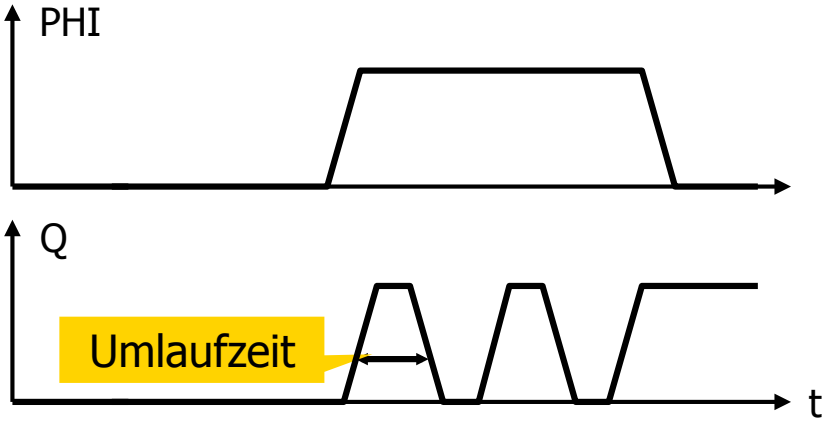
- Problem:
  Für J=K=1 muß der Taktimpuls **PHI kürzer als die Durchlaufzeit** sein, sonst **oszilliert** die Schaltung!



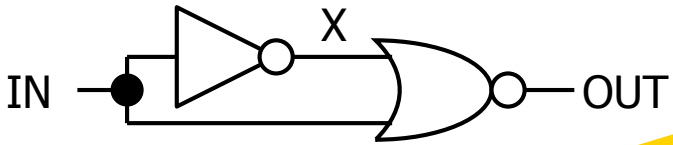
Das 'Race' Problem

- Ein rückgekoppeltes Signal kann bei Latches zu Oszillation führen, wenn das LD Signal länger als die Umlaufzeit ist:

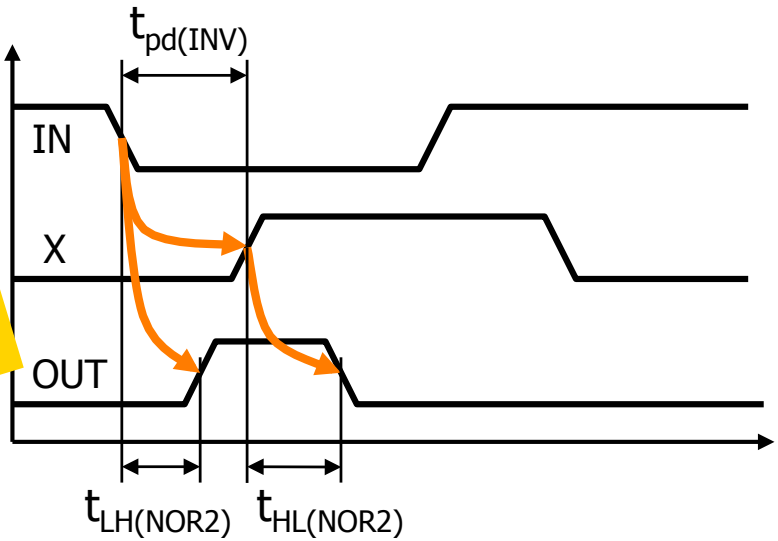




- Lösungsmöglichkeit: Kurzes LD-Signal erzeugen (‚Monoflop', ‚Impulsglied').
  Ist in manchen kommerziellen JK-FFs enthalten. Datenblatt beachten !

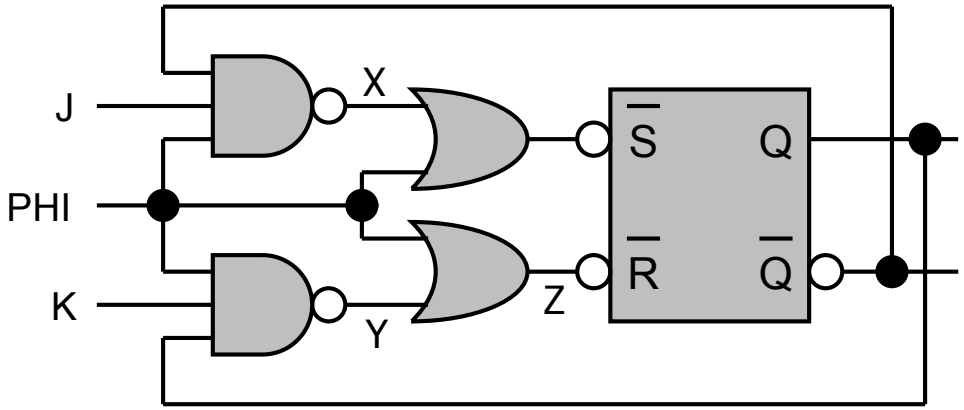


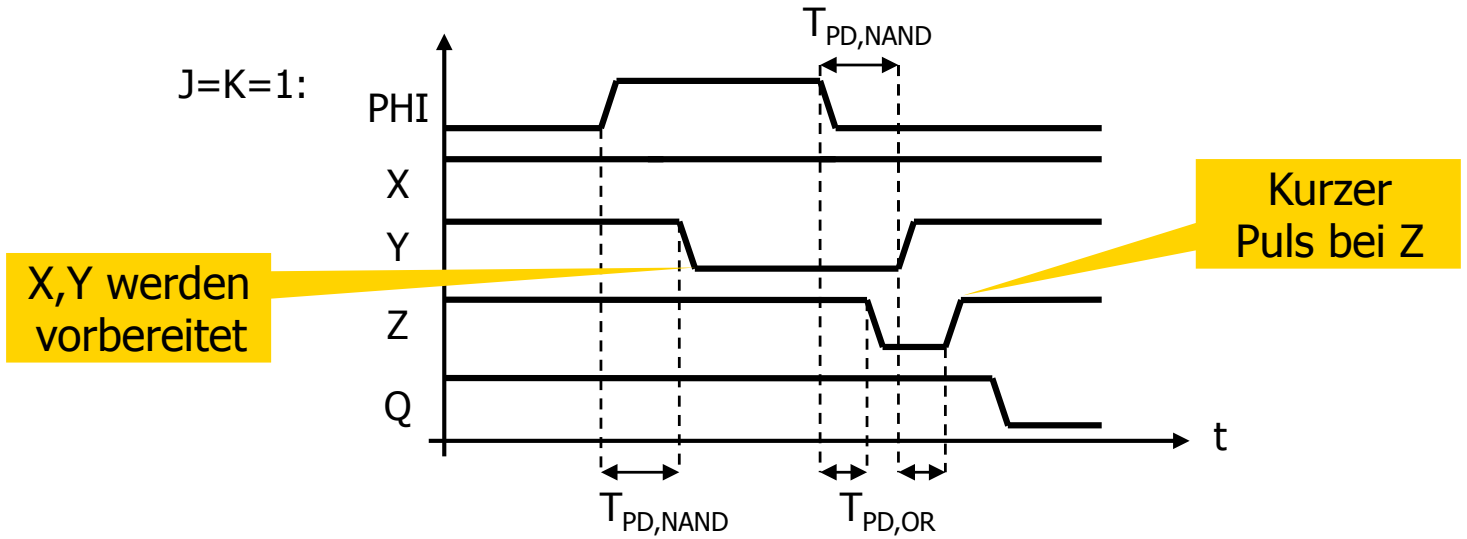
Diese Lösung hängt von Durchlaufzeiten ab. RISKANT !!!



Flankengetriggertes JK - Flip-Flop

- Die **fallende Flanke von PHI** schaltet den Ausgang

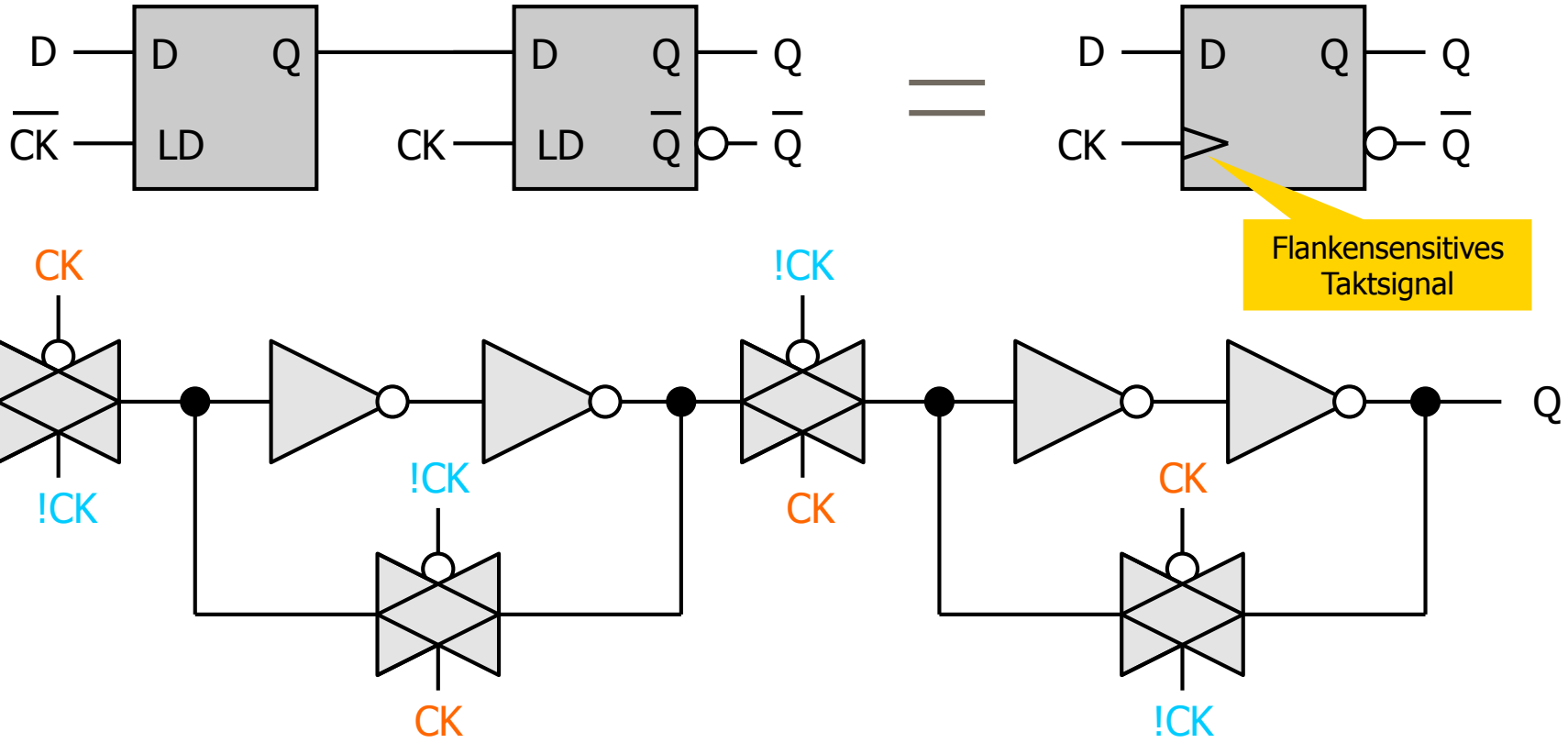




- Gefährliches Design: Basiert auf Durchlaufzeiten. Das SR-FF muss bei kurzen Pulsen schalten!
- Der Eingang muß **VOR** der fallenden Taktflanke anliegen (hier Durchlaufzeit NAND3): **'Setup-Zeit'**

Flankengetriggertes Master-Slave FlipFlop

- Entsteht durch Zusammensetzen von zwei Latches. Hier getriggert auf **positive Flanke**

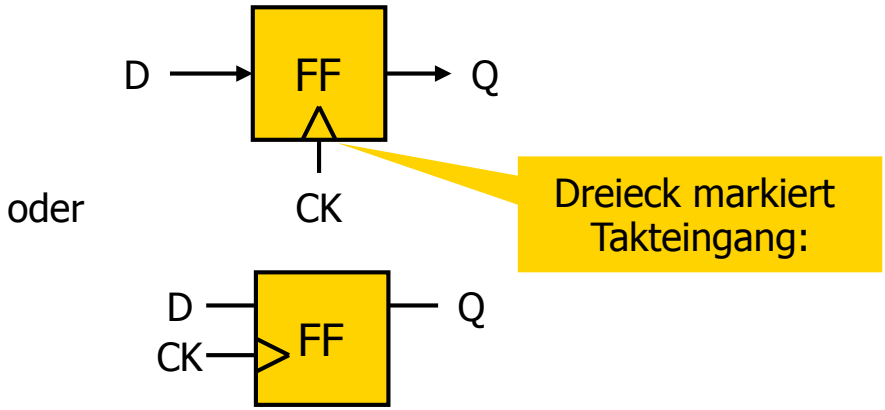


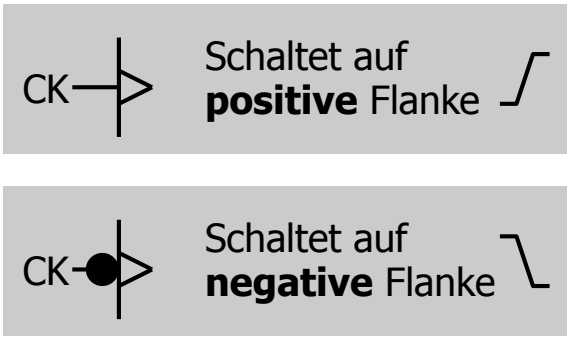
- Achtung: CK und !CK dürfen sich **NICHT ÜBERLAPPEN**
- Optimierung durch Abgriff zwischen den Invertern
- Implementierung auch mit Gated Invertern
- Hinzufügen von Set bzw. Reset Eingängen durch teilweises Ersetzen von Invertern durch Gatter

Nochmal: Timing flankengetriggerter Flip-Flops

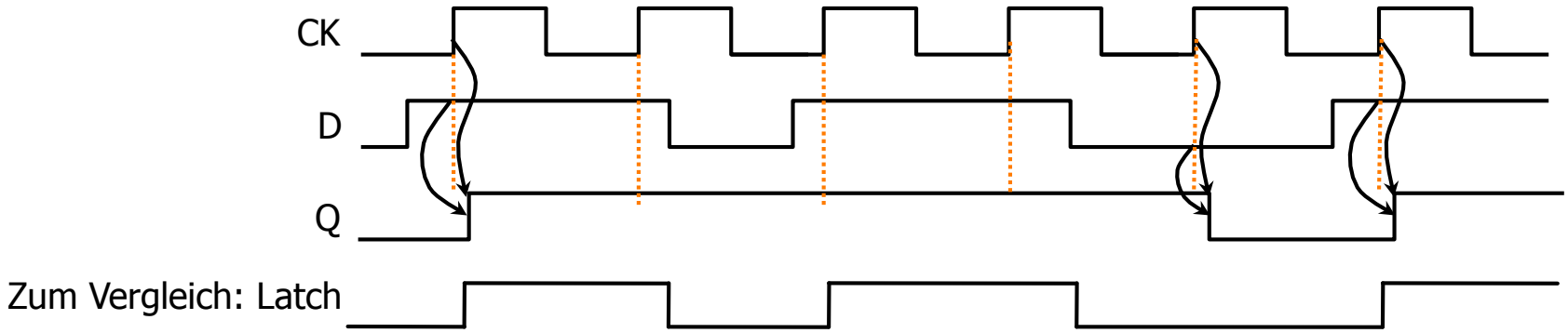
- Ein D-Flipflop überträgt den Wert seines
  - **D-Eingangs** an den
  - **Q-Ausgang** bei der (steigenden) **Flanke** (d.h. beim 0⇒1 Übergang) des Taktes
  - **Takt CK**

- Schaltsymbol

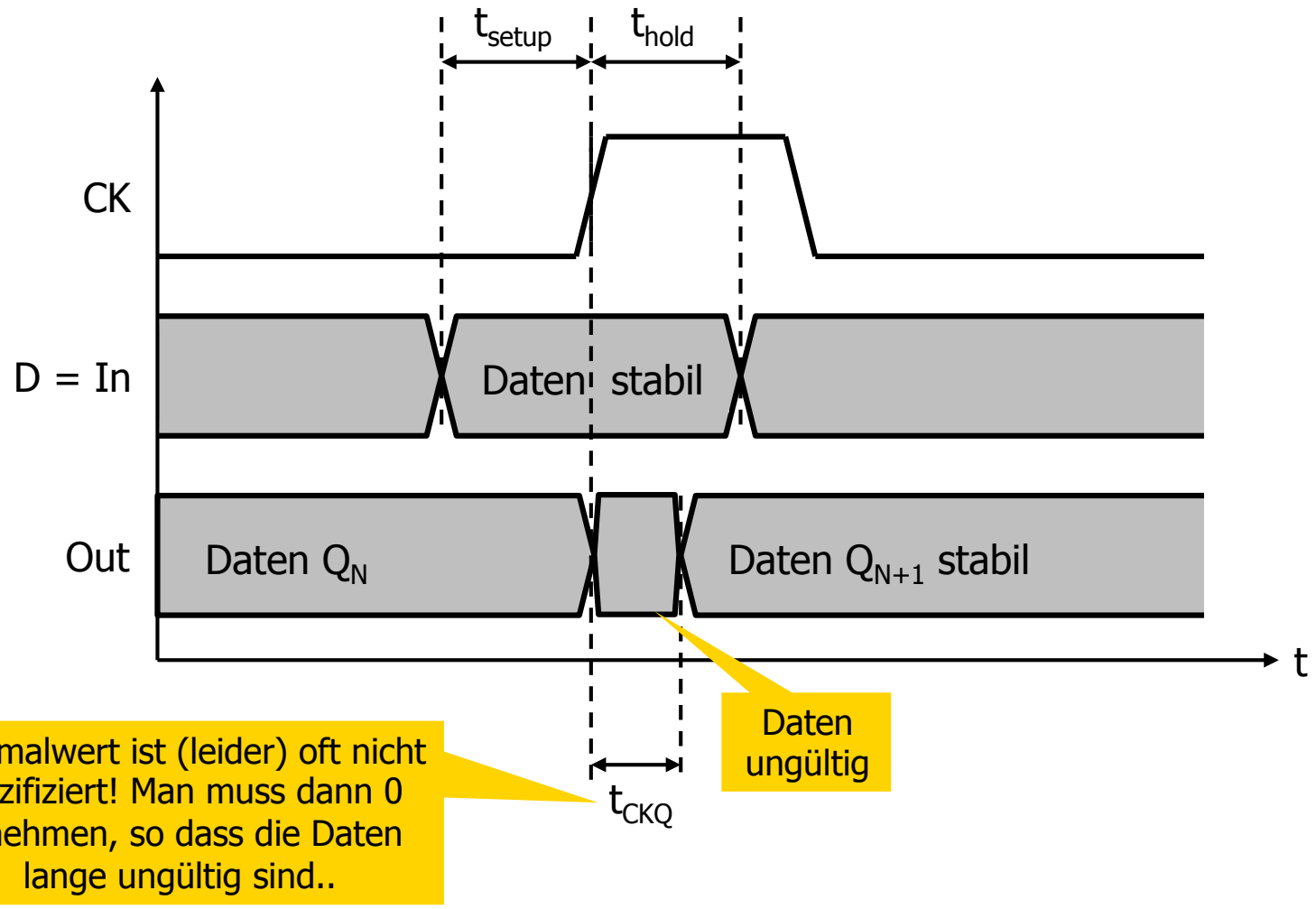




- Zeitliches Verhalten (Triggerung auf positive Taktflanke):



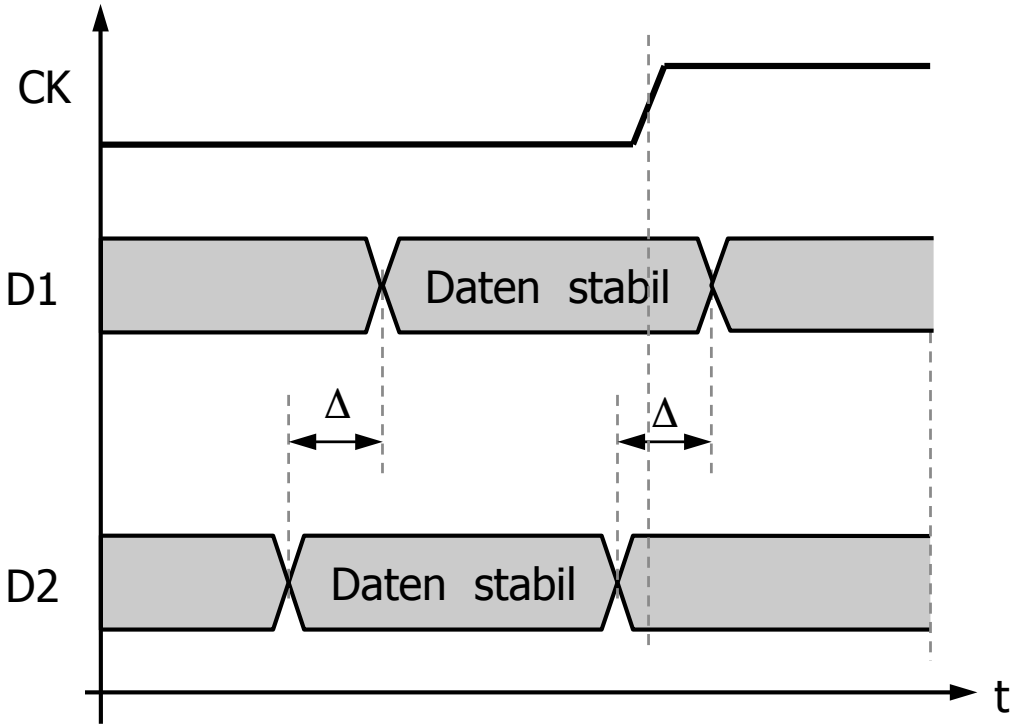
Flipflop: Timing Definitionen

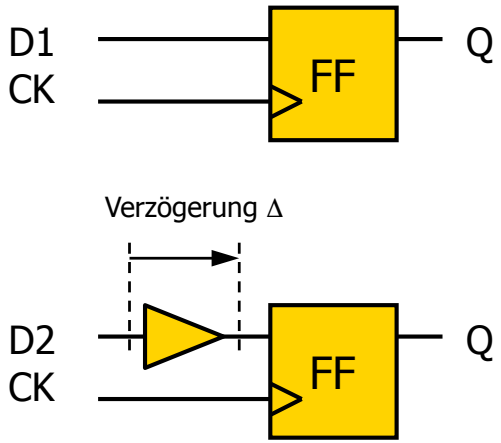


- Die Eingangsdaten müssen VOR der Taktflanke schon mindestens t_{setup} lang gültig gewesen sein
- Sie müssen NACH der Taktflanke noch mindestens während t_{hold} anliegen

Verschiebung von Setup und Hold Zeiten

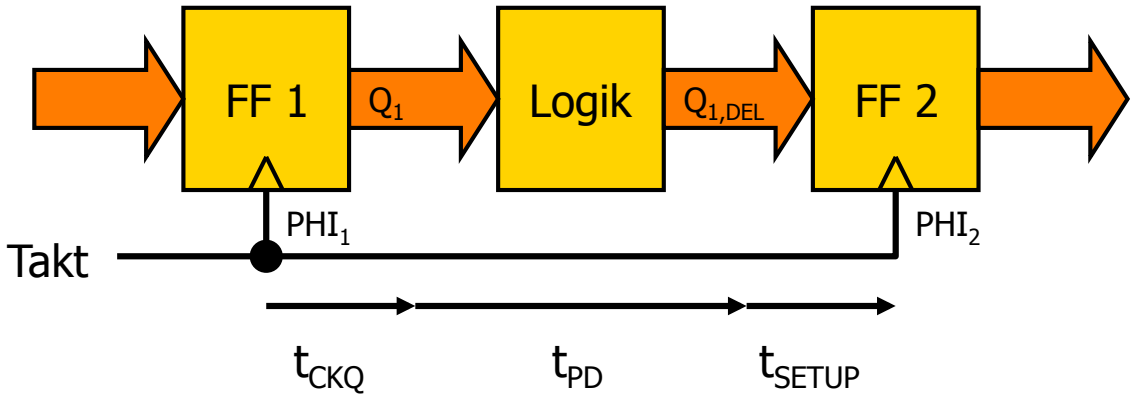
- Das Akzeptanzfenster kann durch Einfügen einer Verzögerung am *Dateneingang* nach vorne geschoben werden.
- Die Hold Zeit kann so zu Null oder negativ gemacht werden, die Setup Zeit erhöht sich.

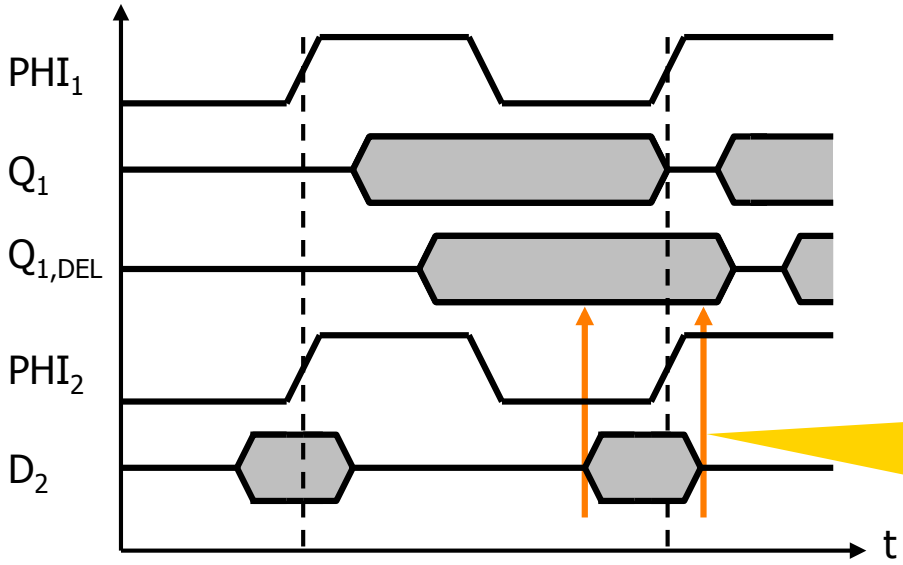




- Durch lokale Verzögerung des *Taktes* wird das Fenster nach hinten verschoben. Hierdurch wird aber auch t_{CKQ} verschoben! Vorsicht!

Erlaubte Taktfrequenzen



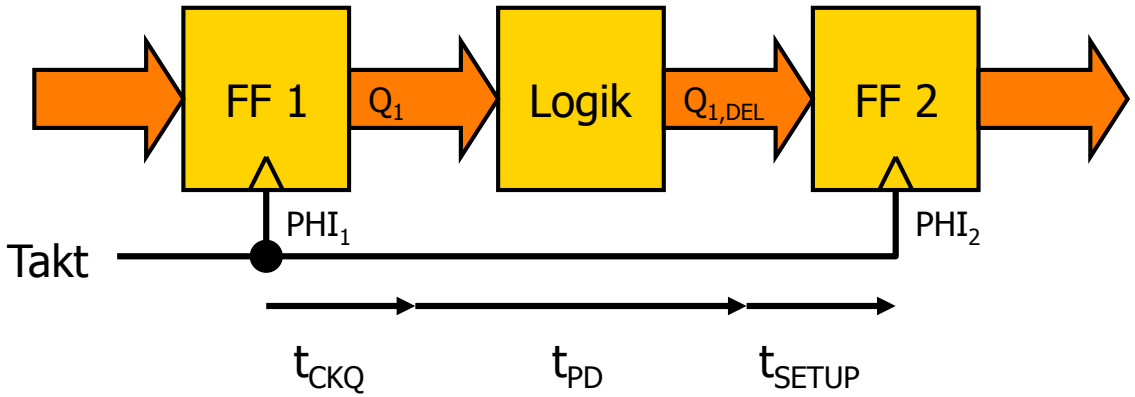


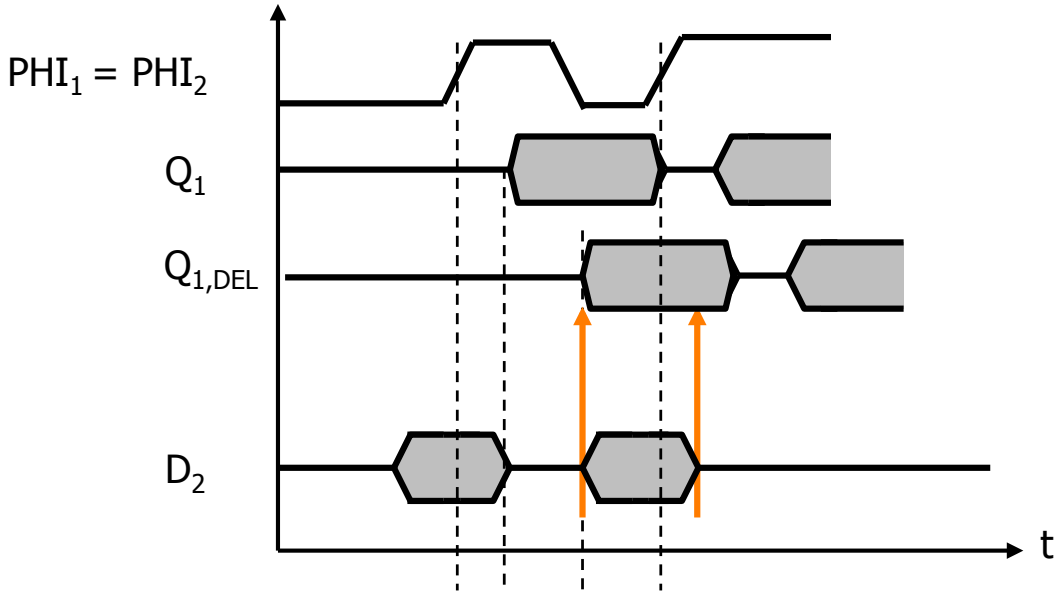
Hold-Erfüllung ist besonders kritisch, wenn $t_{PD} = 0$ (Schieberegister!)

Diese Bedingungen müssen erfüllt sein !

- Taktperiode $\mathbf{t_{CLOCK} > t_{CKQ,max} + t_{PD} + t_{SETUP}}$ und $\mathbf{t_{CKQ,min} + t_{PD} > t_{hold}}$ (kleines t_{CKQ} gefährlich!)
- falls PHI1 und PHI2 nicht zeitgleich sind (RC auf Leitungen, Buffer..) muß dies berücksichtig werden!

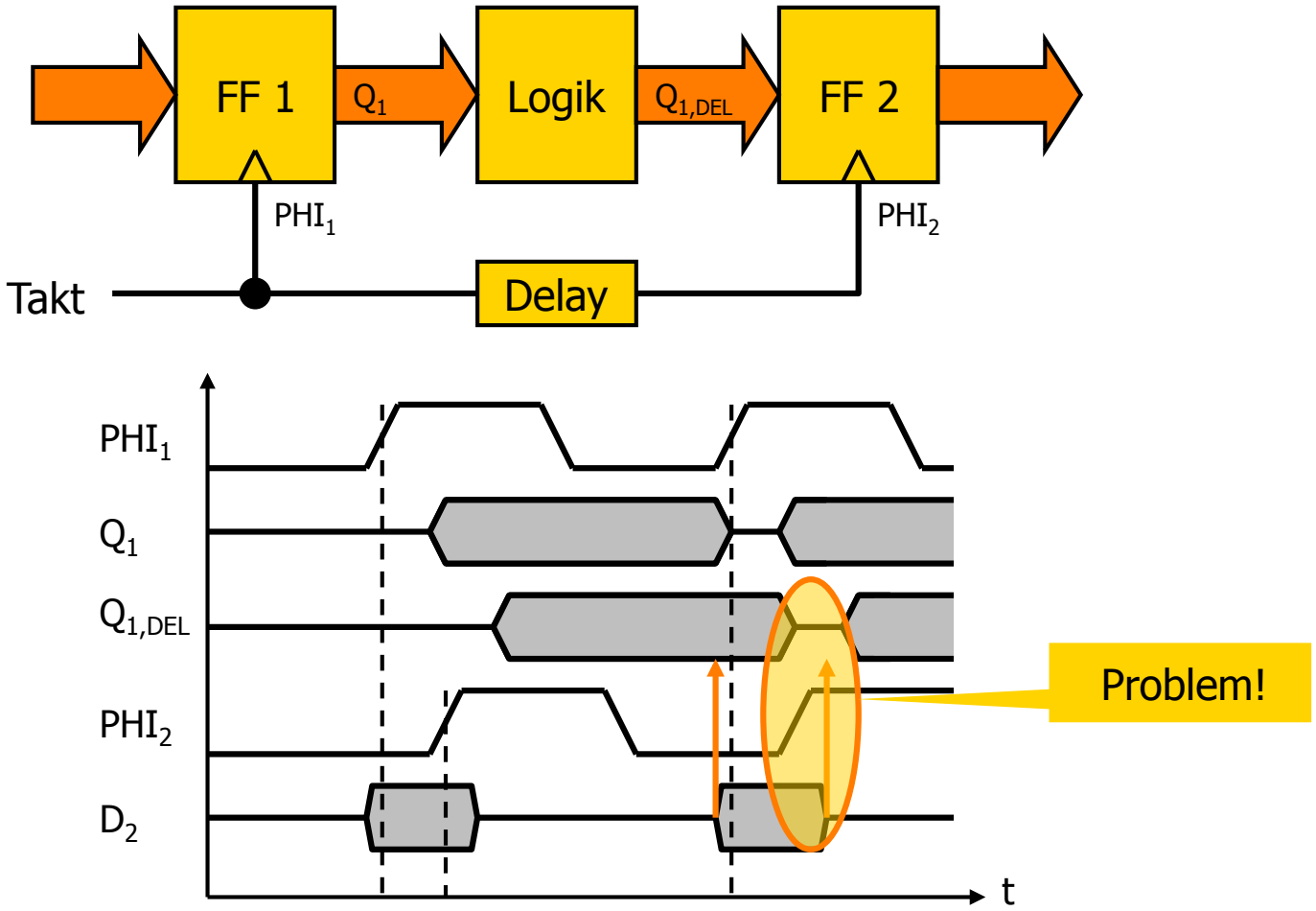
Maximale Taktfrequenz





$$t_{CLOCK} = t_{CKQ,max} + t_{PD} + t_{SETUP}$$

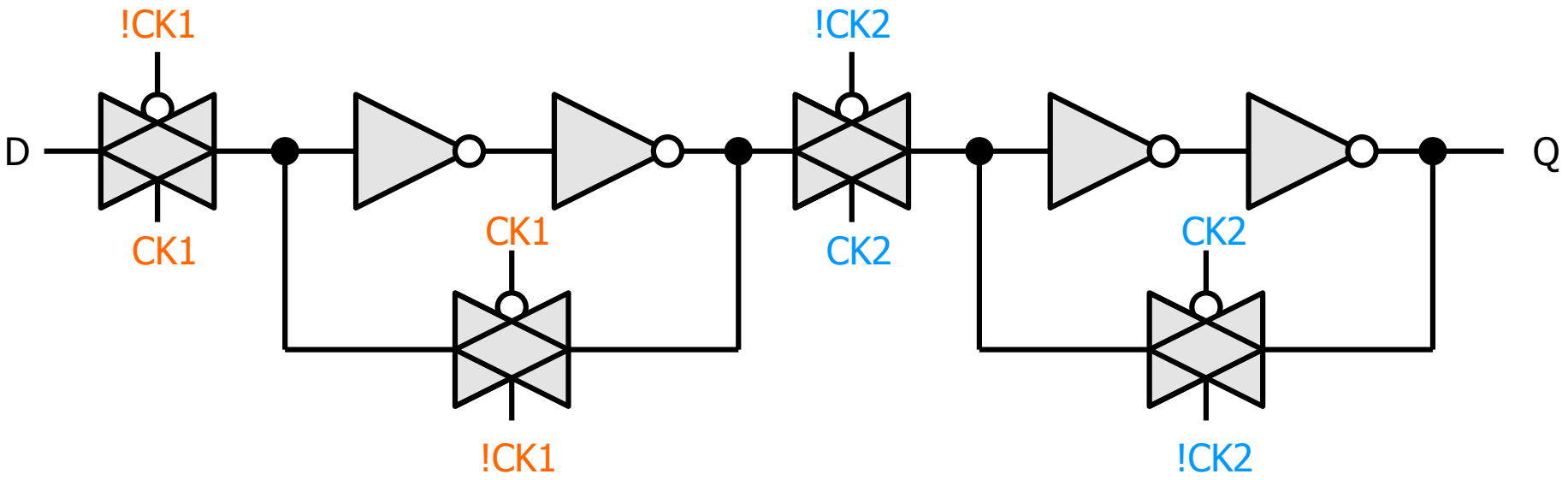
Verzögerung des Takts – 'skew'

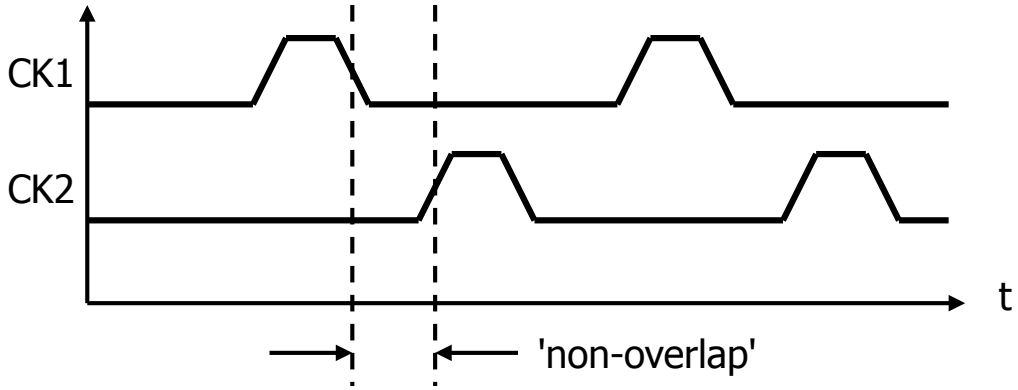


- Daten werden nicht richtig übertragen, wenn sie innerhalb der Setup/Hold-Zeit nicht stabil sind
- **Dies kann bei Verzögerung des Taktes (unabhängig von der Taktfrequenz!) passieren!**

Konservatives Statisches Zwei-Phasen-Flipflop

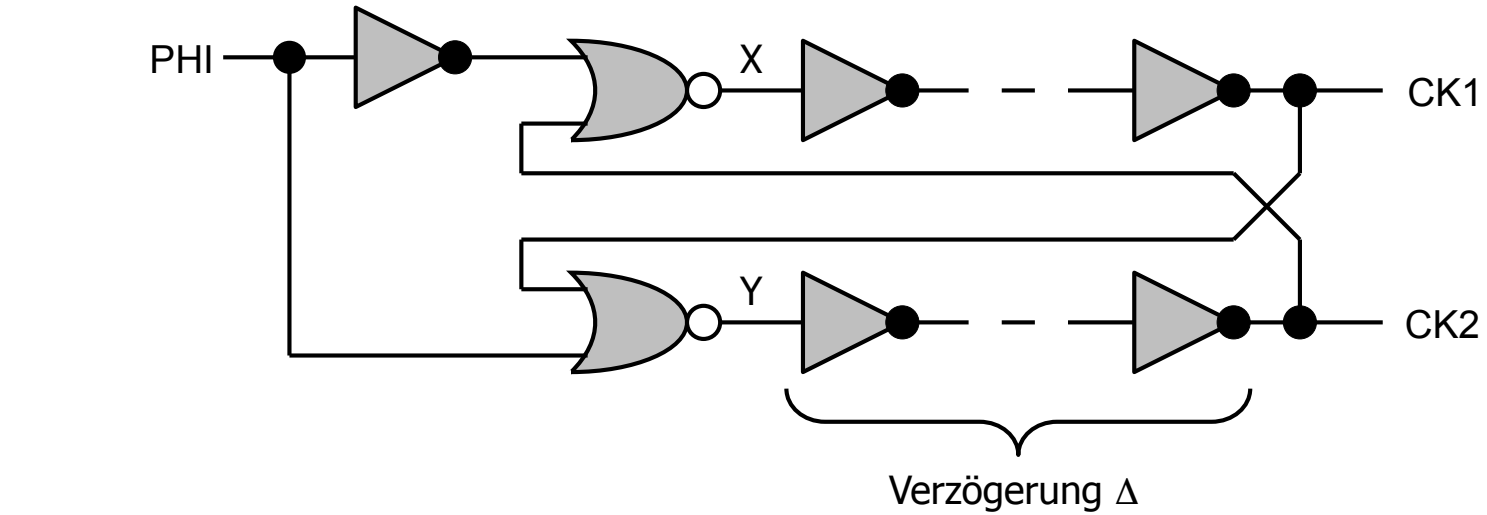
- Vermeide das Risiko der Überlappung von CK und !CK durch Einführung von **zwei** Taktsignalen CK1/CK2
- Ein Latch pro Taktsignal:

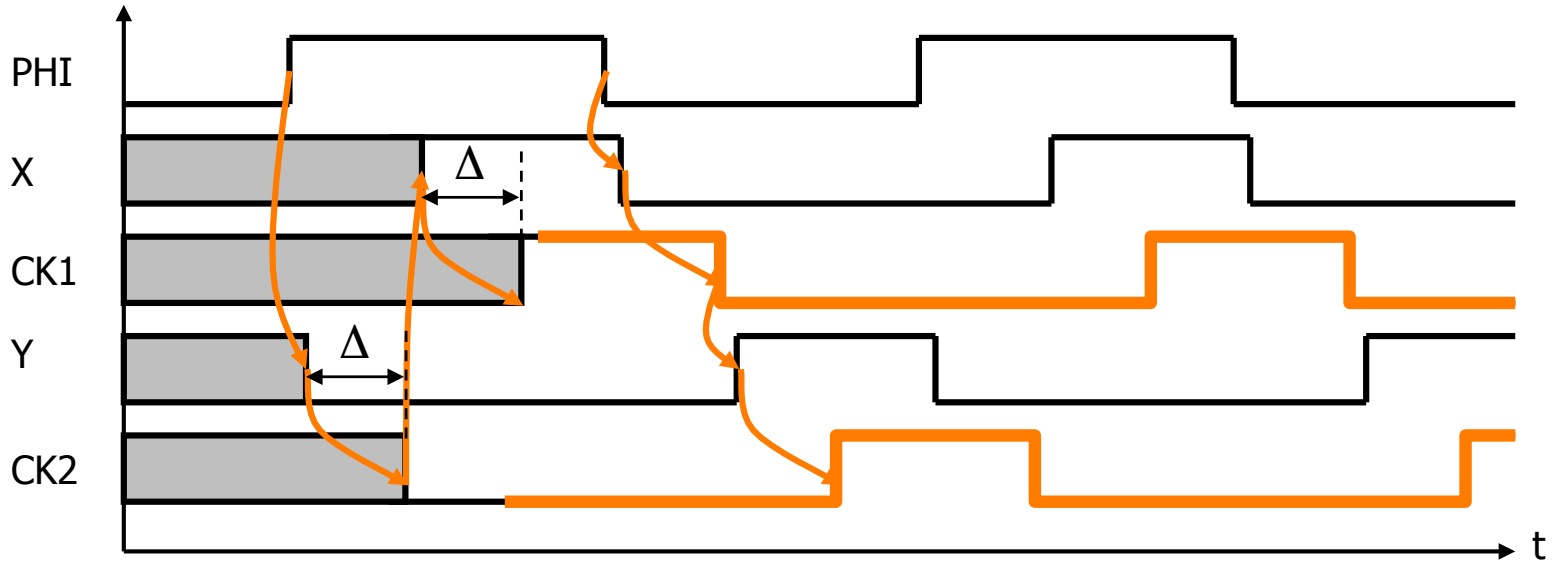




Erzeugung von Nicht-Überlappenden Takten

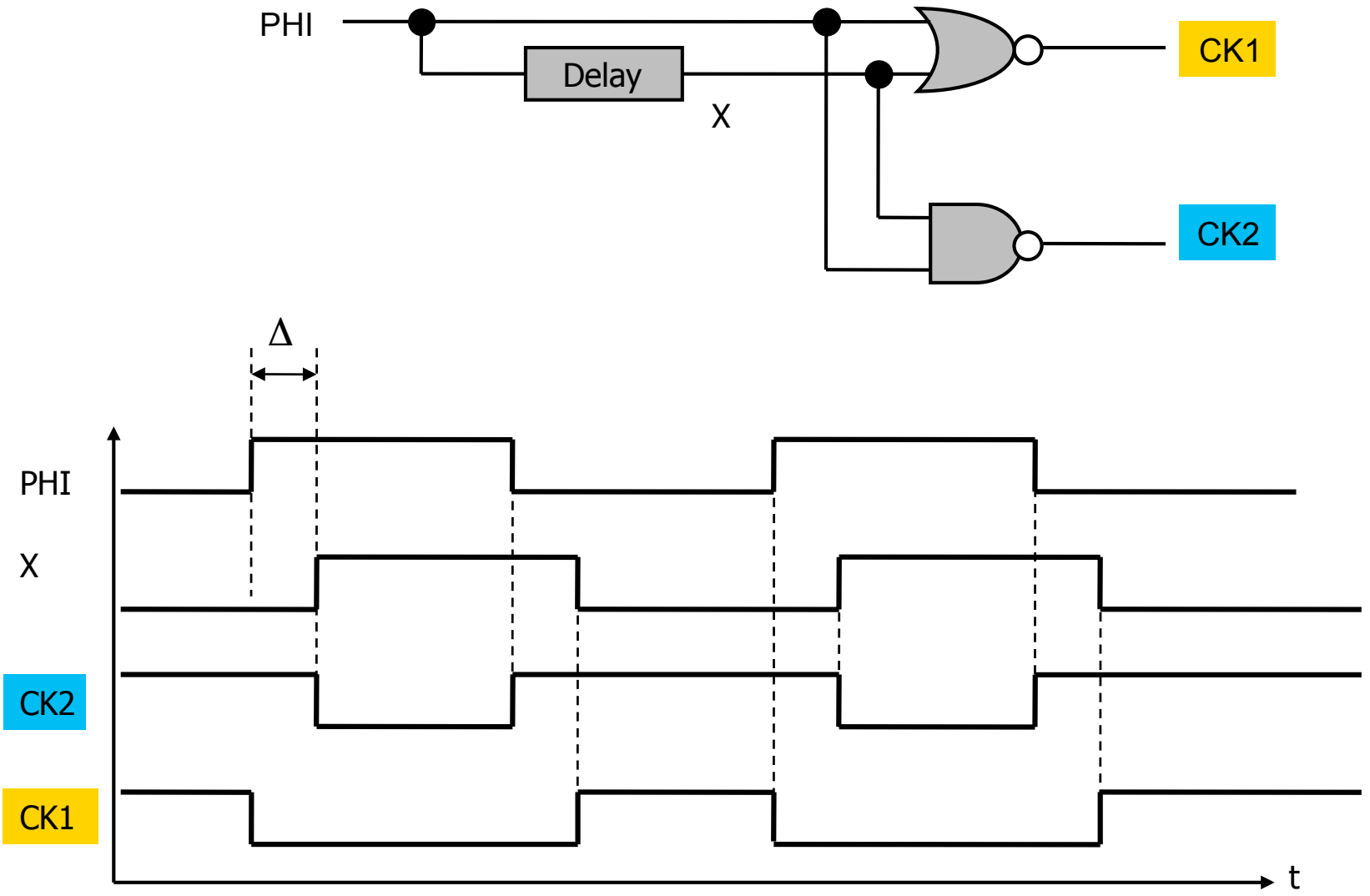
- Klassische ‚Textbook' Schaltung: SR-FF mit interner Verzögerung:





Alternative Schaltung

- Benötigt nur eine Verzögerung → leichter einstellbar!



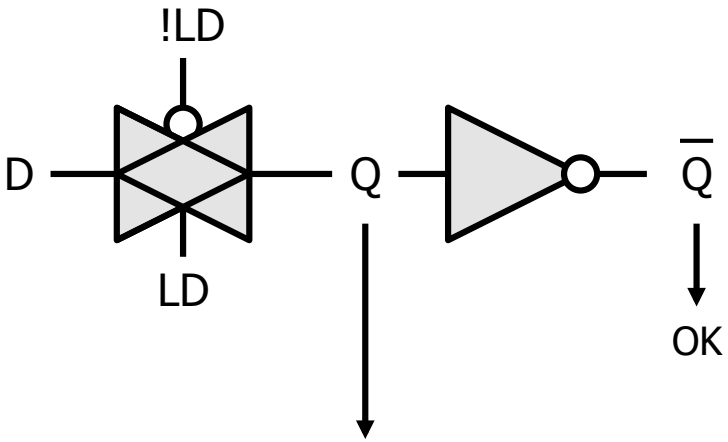
Takt Strategien

- Latches und kurze LD Pulse – GEFÄHRLICH
- Eine Phase – OK, aber Vorsicht vor Clock skew! Standard
- Zwei-Phasen Takt – OK, sehr konservativ und sicher. Gut für langsame Designs!
- Exotisch: 'Self-Timed' – Asynchronous Design:
  - Schaltung meldet, wenn Logische Funktion implementiert ist
  - Wird immer wieder diskutiert
  - Schwierig, starker Extra-Bedarf an Logik – lohnt sich nur selten
  - Aktivität nur wenn sich was ändert -> low power

Vorgriff: Dynamische D – Latches

- Daten werden als Spannung auf einer (parasitären) Kapazität gespeichert (wie in dynamischen RAMs)

Mit Transmission Gate

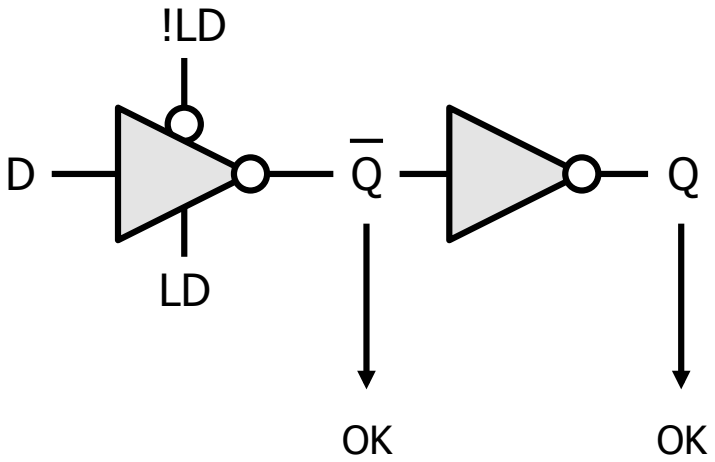


Nicht getrieben ('buffered')

⇓

darf wegen Ladungsteilung nicht direkt an weitere Transmission Gates angeschlossen werden

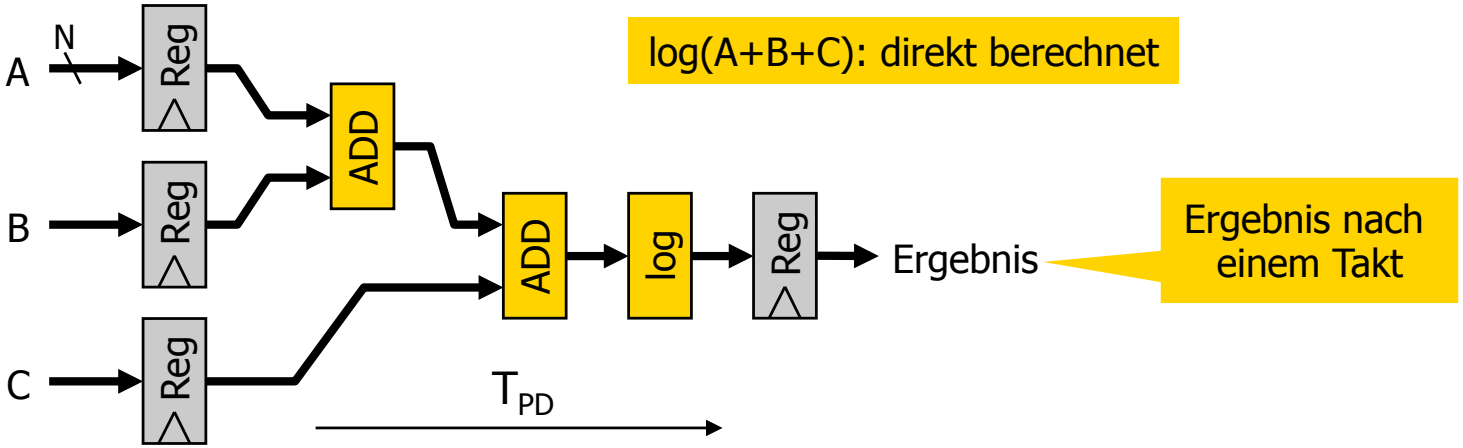
Mit Gated Inverter

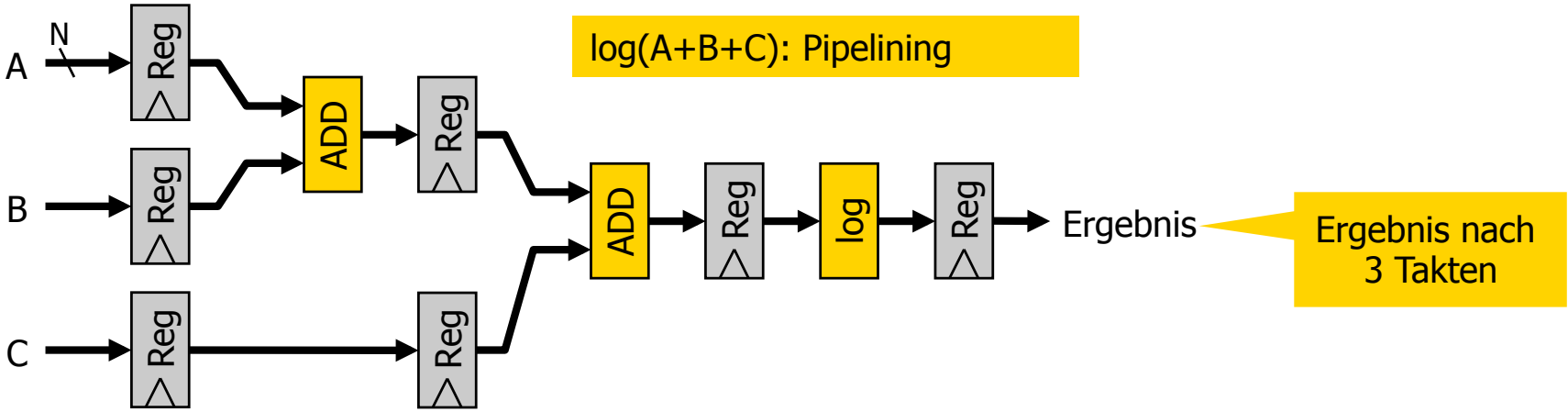


- Sehr einfache Schaltungen
- Layout des Gated Inverters trotz 4 Transistoren oft nicht ungünstiger als Layout des Transmission Gates

Pipelining

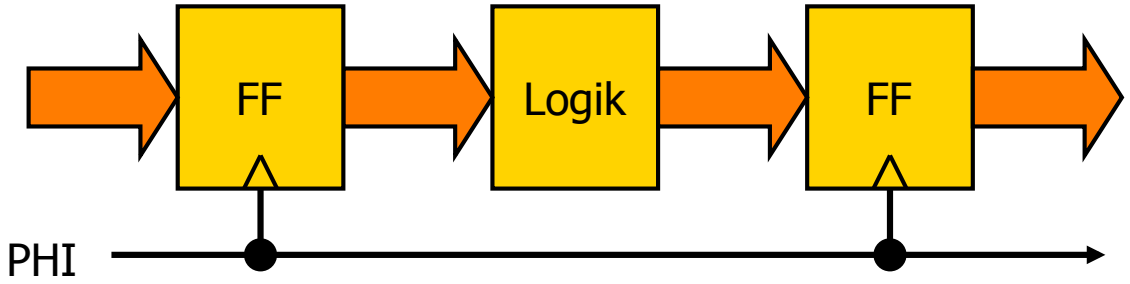
- Pipelining = Konsekutives Abarbeiten einer Aufgabe. Einzelschritte werden je in einem Takt erledigt
- Man benötigt also mehrere Stufen FFs mit Logik dazwischen
- Das Ergebnis ist erst nach einer längeren **Latenz** verfügbar



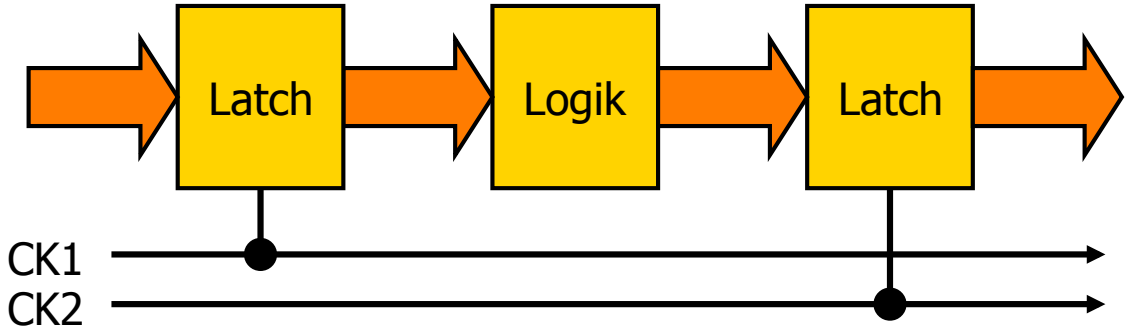


Pipelining - Implementierung

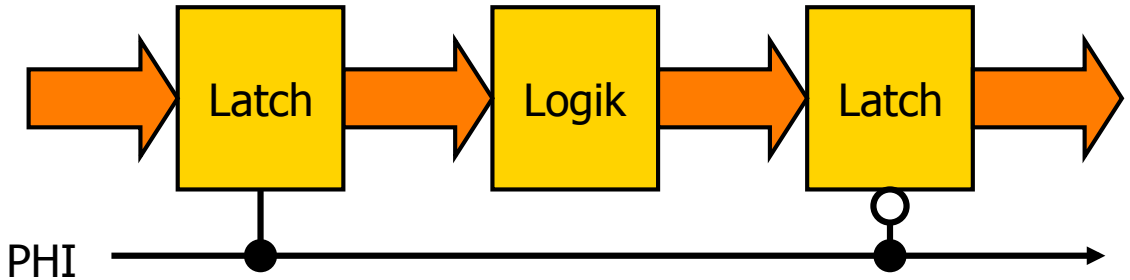
- FFs, Eine Phase
- Standard



- Latches, Zwei Phasen (nichtüberlappend)
- Entspricht dem Einbau von Logik ins FF bei two-phase Clocking

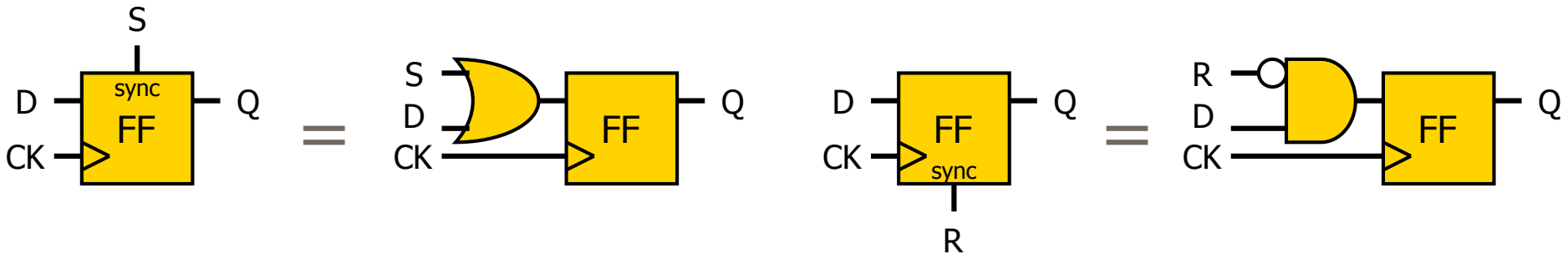


- Latches, Eine Phase
- Entspricht Einbau von Logik in FFs bei einer Taktphase

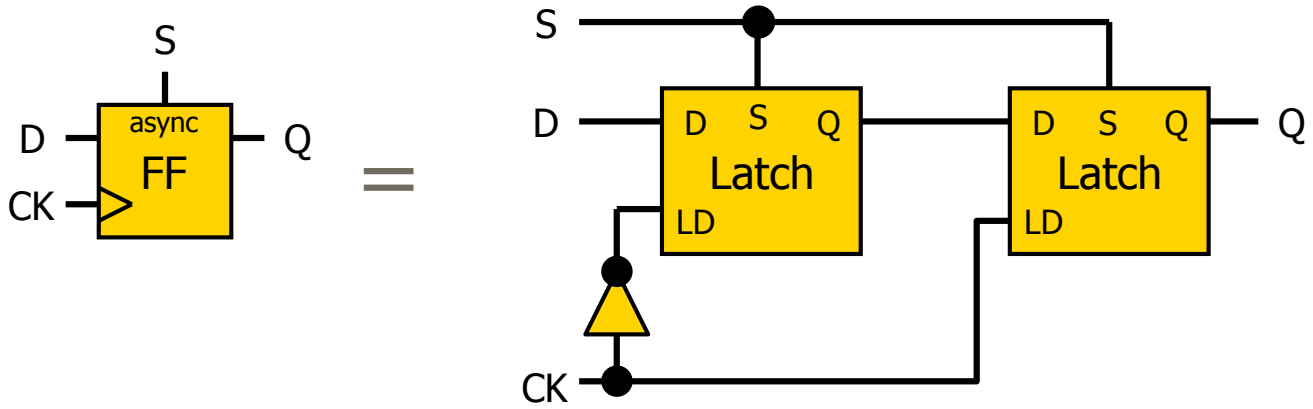


Setzen und Rücksetzen von FFs

- Implementierung **synchrones** Setzen/Rücksetzen:



- Implementierung **asynchrones** Setzen:



- Ob ein Set/Reset Eingang synchron oder asynchron ist, geht aus dem Schaltsymbol oft nicht klar hervor !!

Setzen und Rücksetzen von FFs

- Beim Rücksetzten (Löschen, **'Reset'**) und Setzen (**'Set'**) unterscheidet man grundsätzlich zwischen:
  - **Synchroner** Operation: Set/Reset wird erst **mit dem Takt** ausgeführt
    Der Ausgang wird dann 1/0, unabhängig vom Eingang
    Es gibt Setup und Hold Zeiten für Set/Reset
  - **Asynchrone** Operation: Set/Reset wird **sofort** ausgeführt
- Set und Reset sollten nie gleichzeitig anliegen!