

5. Schaltwerke

- **Bisher: Schaltnetze**
 - Ausgabe nur abhängig von momentaner Eingabe (bis auf Hazards)
- **Jetzt: Schaltwerke**
 - Ausgabe hängt auch von einem gespeicherten Zustand ab
- **Dazu brauchen wir: Speicherglieder**
 - Schaltvariablen aufnehmen, speichern, auslesen
 - Speicherglieder brauchen nur zwei verschiedene Werte aufzunehmen (2 stabile Zustände)
 - 0: Rücksetzzustand (*reset*)
 - 1: Setzzustand (*set*)
 - derartige Speicherglieder heißen
 - bistabile Kippglieder
 - speziell (s.u.) auch Latches oder Flip-Flops

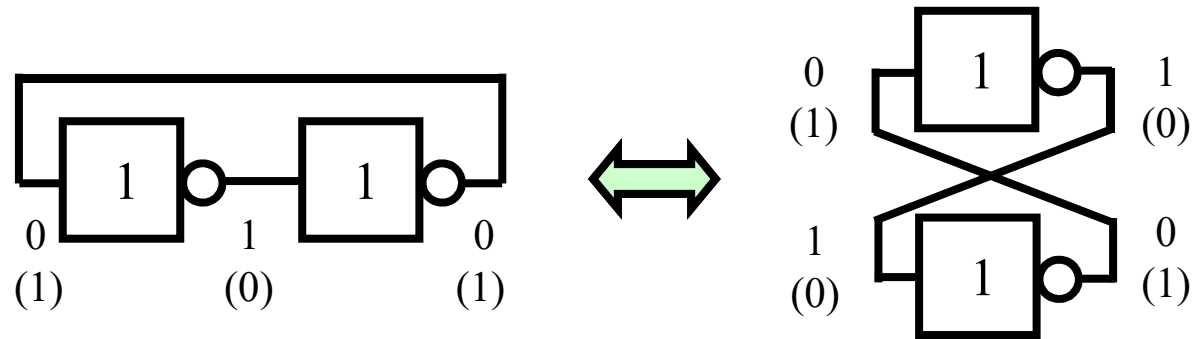
Bistabile Kippglieder

- durch geeignete Ansteuerungen kann ein bistabiles Kippglied von einem Zustand in den anderen übergehen
- **Mögliche Arten der Ansteuerung**
 - taktunabhängig
 - taktabhängig
 - taktzustandsgesteuert
 - taktflankengesteuert
 - je nach Ansteuerung erhält man unterschiedliche Arten von Kippgliedern, die im folgenden beschrieben werden

Grundschtaltung

- **Statisch rückgekoppelte Inverter**

- Grundelement aller bistabilen Kippschaltungen
- zwei statisch rückgekoppelte Inverter
- zwei stabile Zustände



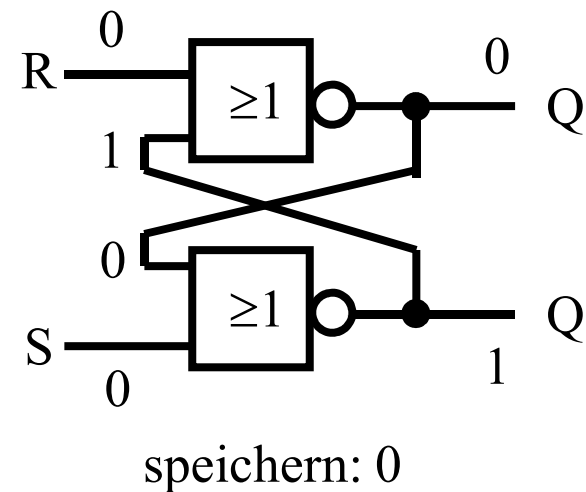
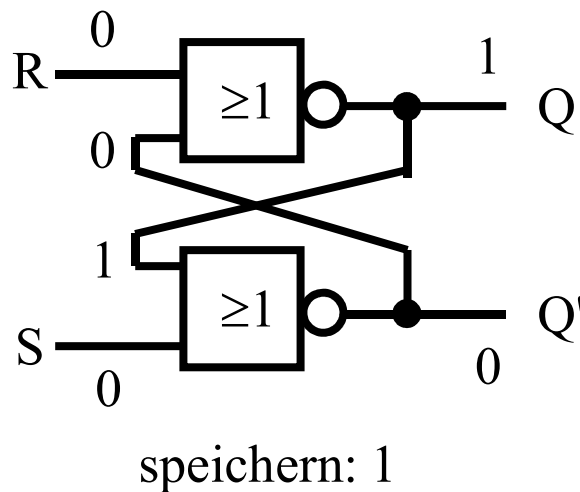
- **Zustandsübergänge**

- um von dem einen in den anderen Zustand zu wechseln, sind weitere Steuereingänge notwendig, die den Ausgang des Gatters (das den Inverter ersetzt) auf einen bestimmten Wert zwingen

Kippglied mit NOR-Gattern

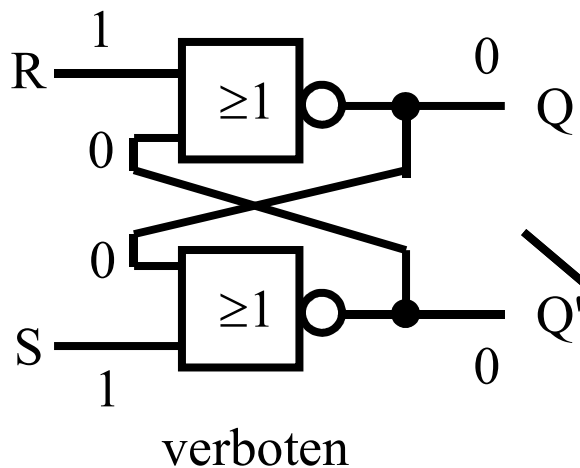
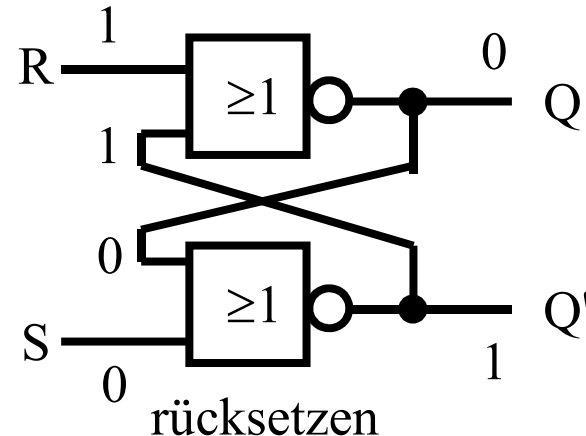
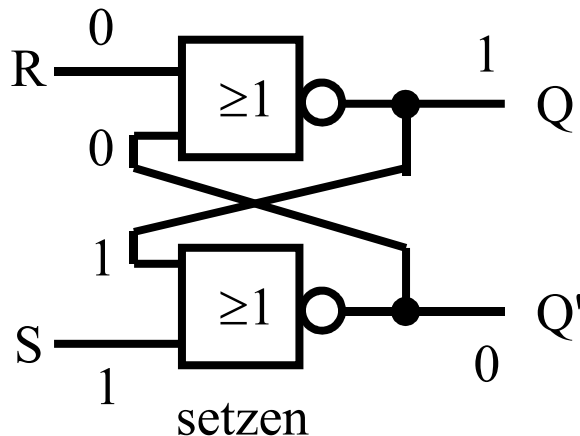
- **RS-Latch**

- einfachstes Speicherglied
- Speicherung gelingt durch Rückkopplung
- dadurch 2 stabile Zustände, falls Eingänge auf 0 liegen (dann verhalten sich NOR-Gatter wie Inverter)



RS-Kippglied (2)

Eine 1 an einem Eingang erzwingt eine 0 am Ausgang eines NOR-Gatters:

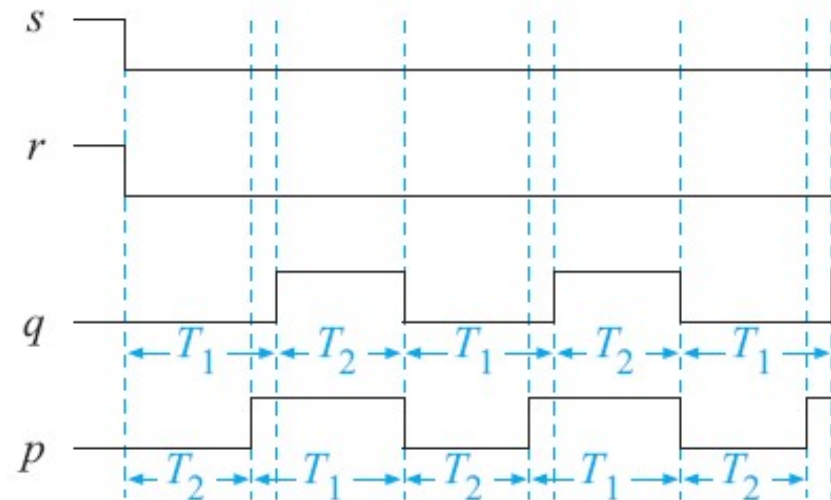
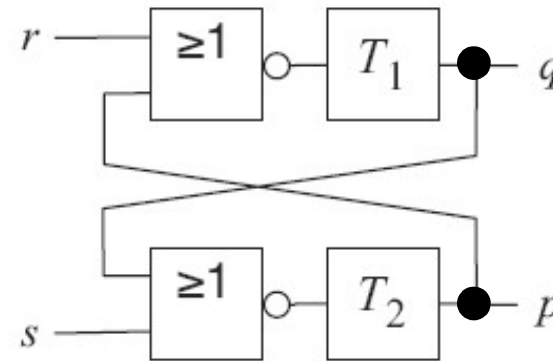


R	S	Q	Q'	
0	0	Q	Q'	speichern
0	1	1	0	<u>s</u> etzen
1	0	0	1	<u>r</u> ücksetzen
1	1	(0)	(0)	verboten

undefinierter Zustand beim Übergang von S=R=1 nach S=R=0

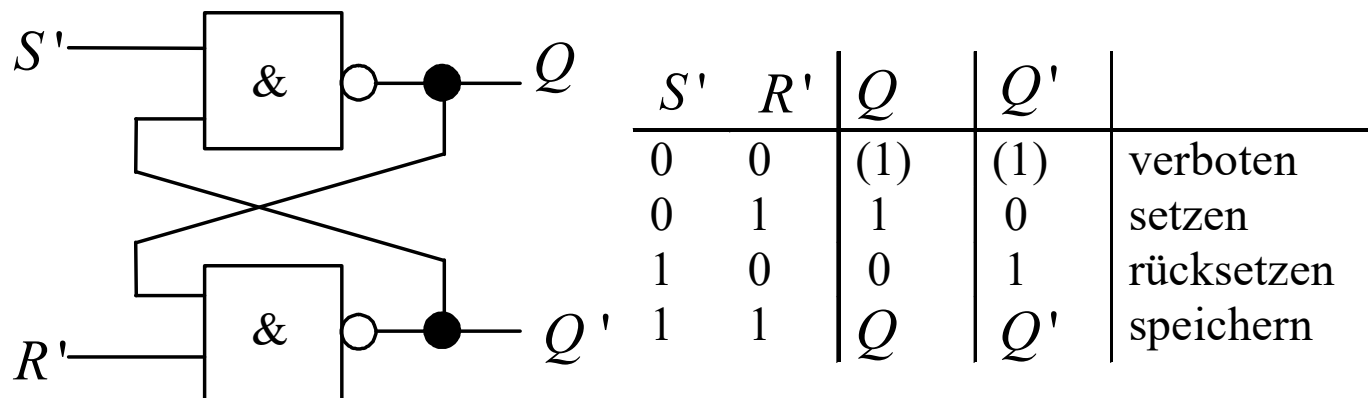
RS-Kippglied (3)

- **Verbotener Übergang von $R=S=1$ nach $R=S=0$**
 - Gatter reagieren nicht sofort, sondern erst nach einer Gatterdurchlaufzeit, hier durch Verzögerungsglieder T_1, T_2 symbolisiert
 - es kann passieren, dass das Latch in Schwingungen gerät
 - Darstellung ist idealisiert
 - T_1 und T_2 sind für die Übergänge $0 \rightarrow 1$ und $1 \rightarrow 0$ nicht völlig identisch
 - daher verschieben sich die Impulsfolgen bis ein stabiler, zufälliger Zustand erreicht wird



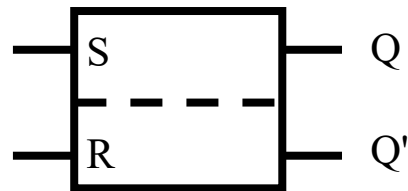
RS-Kippglied mit NAND-Gattern

- eine 0 an einem Eingang erzwingt eine 1 am Ausgang eines NAND-Gatters
- der Zustand mit beiden Eingängen auf 0 ist der nicht erlaubte Zustand
- gespeichert wird, wenn beide Eingänge auf 1 liegen

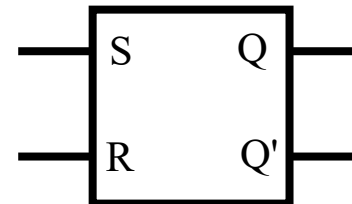


RS-Latch

- Kippglieder mit NOR- und NAND-Gattern zeigen das gleiche Verhalten
- man fasst sie unter dem Oberbegriff RS-Kippglied (RS-Latch) zusammen, der das Verhalten unabhängig von der schaltungstechnischen Realisierung beschreibt
- **Schaltzeichen**

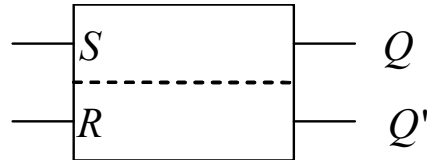


DIN



IEEE

RS-Latch (2)



S	R	Q_{n+1}	Q'_{n+1}	
0	0	Q_n	Q'_n	speichern
0	1	0	1	rücksetzen
1	0	1	0	setzen
1	1	(x)	(x)	verboten

Zustandsfolgetabelle

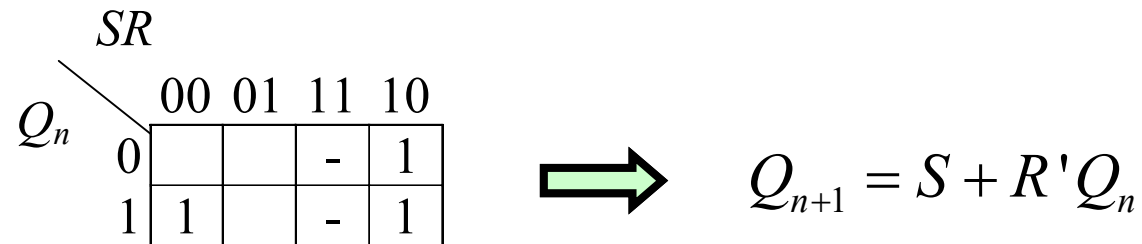
S	R	Q_n	Q_{n+1}	
0	0	0	0	speichern
0	0	1	1	speichern
0	1	0	0	rücksetzen
0	1	1	0	rücksetzen
1	0	0	1	setzen
1	0	1	1	setzen
1	1	0	(x)	verboten
1	1	1	(x)	verboten

Erweiterte Zustandsfolgetabelle

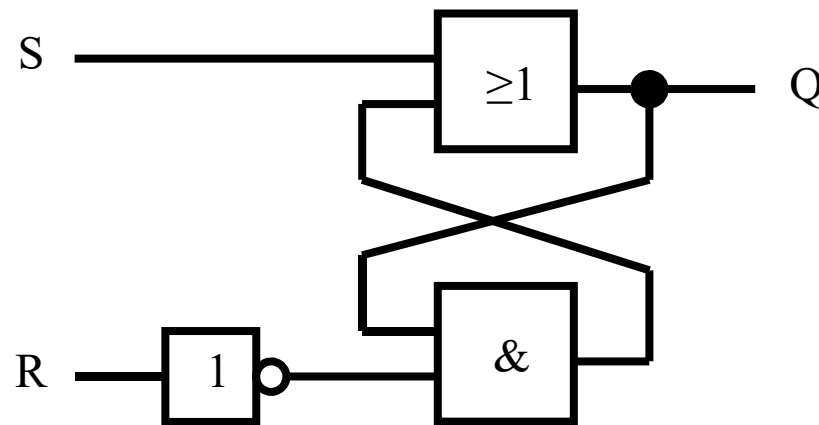
- Q_n bedeutet den Zustand vor Anlegen der Signale R und S, Q_{n+1} den Zustand unmittelbar danach (eine Signallaufzeit durch das Latch)
- die erweiterte Zustandsfolgetabelle kann als Wertetabelle einer Schaltfunktion angesehen werden

RS-Latch (3)

- das KV-Diagramm und die minimierte Schaltfunktion ergeben sich zu

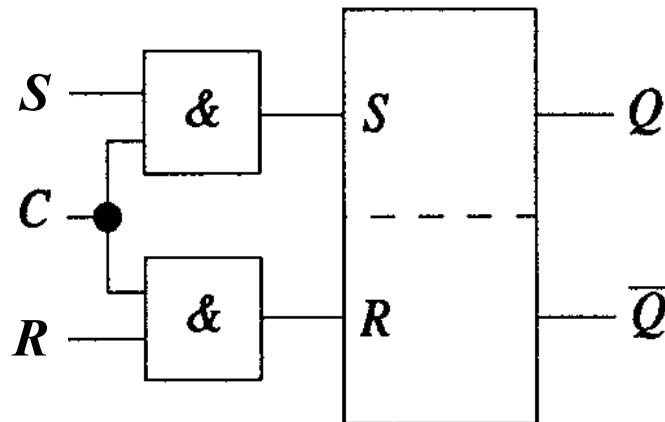


- mit der Nebenbedingung $R \cdot S = 0$, d.h. $R=S=1$ ist verboten
- die Funktionsgleichung wird Übergangsfunktion des RS-Latches genannt



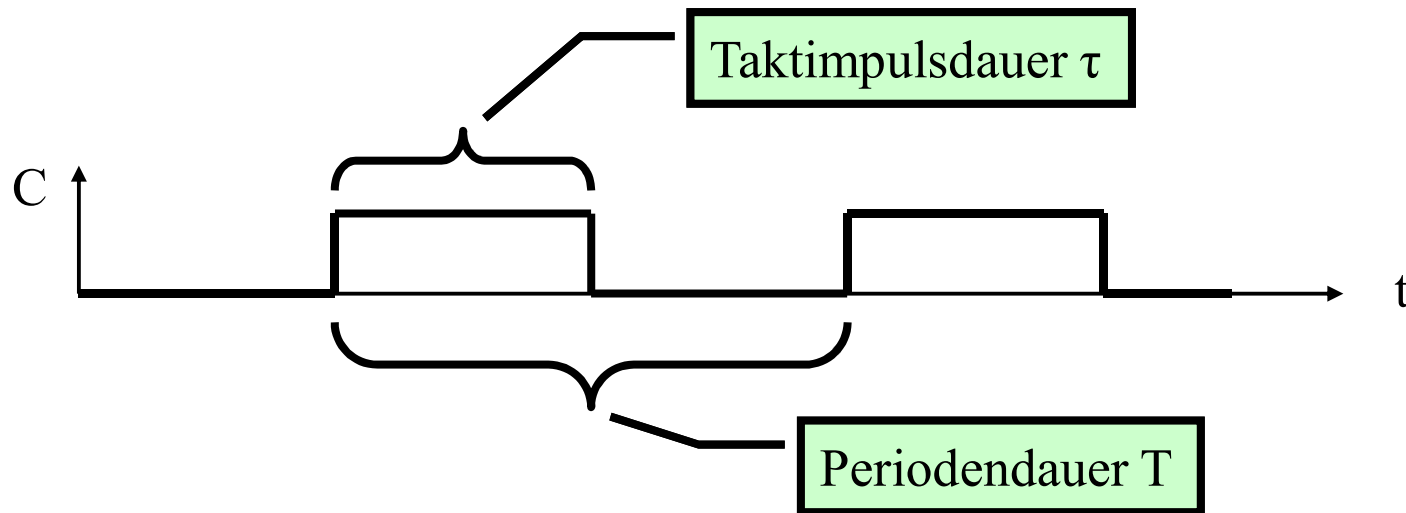
Taktabhängige Kippglieder

- beim RS-Latch wird ein anliegendes Eingangssignal sofort wirksam, d.h. auch kurze Störungen oder Hazards können zu einem Kippen der Schaltung führen
- deshalb ist es sinnvoll, das Wirksamwerden des Eingangszustandes über eine Taktleitung zu steuern,
- man erhält ein RS-Kippglied mit Taktzustandssteuerung



Taktsignale

- Taktsignale
 - dienen zur Synchronisation von Verarbeitungsschritten

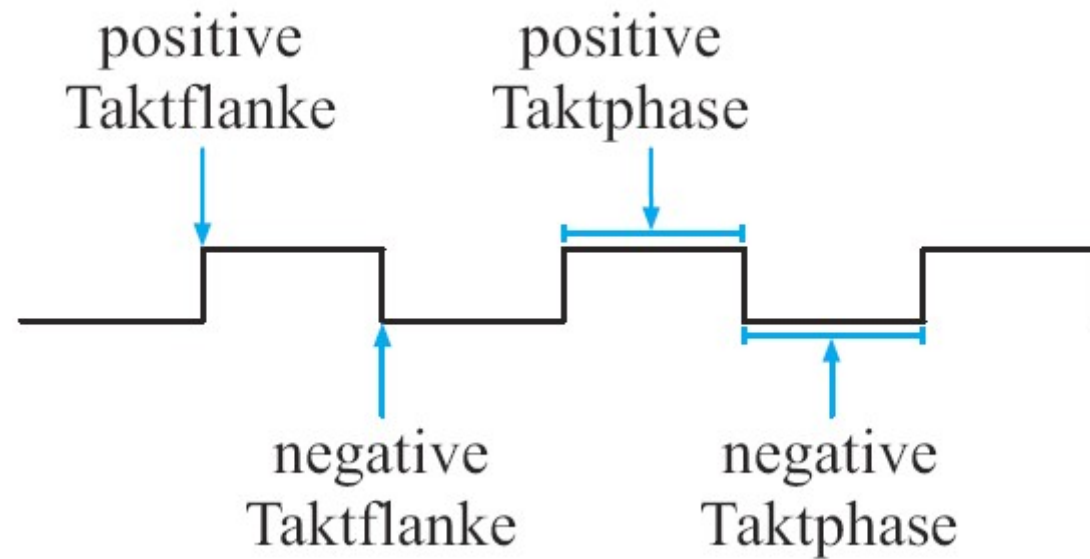


Taktfrequenz = $1/T$ (in Hertz gemessen)
1 GHz entspricht $T = 10^{-9}\text{s} = 1 \text{ ns}$

Tastverhältnis τ/T liegt zwischen 0 und 1 (in der Grafik ist es 0,5)

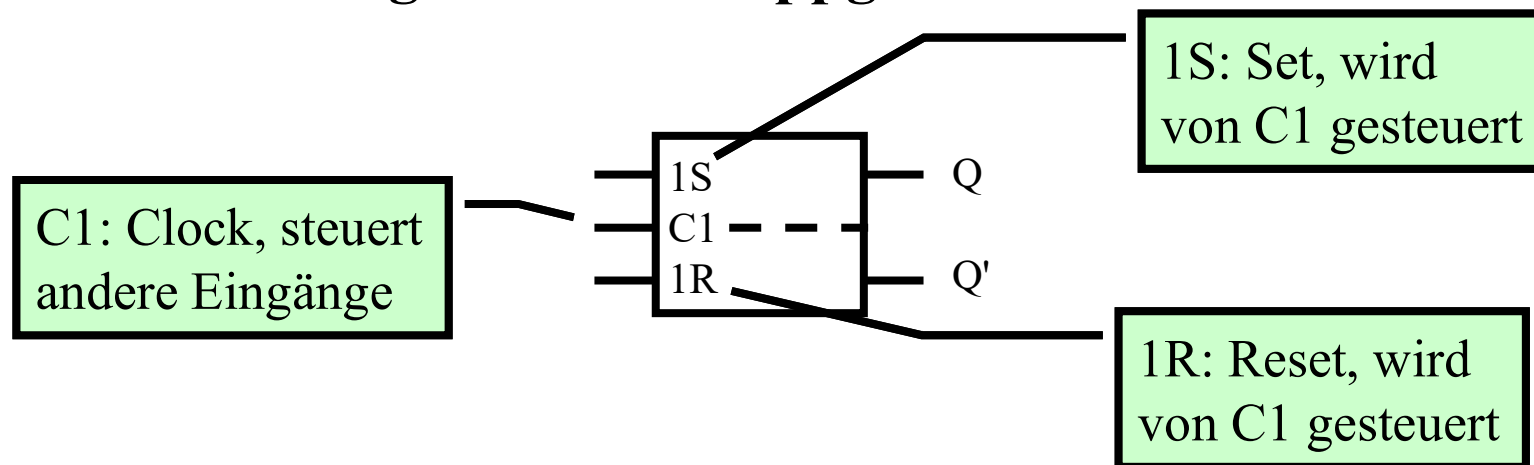
Taktsignale

- Weitere Begriffe im Zusammenhang mit Taktsignalen



Abhängigkeitsnotation

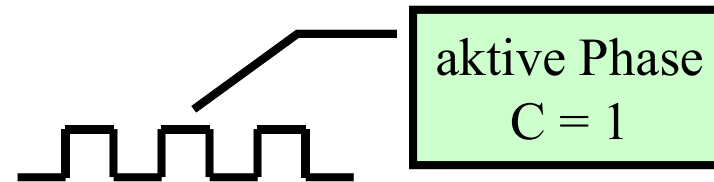
- nach DIN 40900, Teil 12
 - genau festgelegte Beschriftungsregeln, die angeben, wie bestimmte Anschlüsse andere beeinflussen
 - Unterscheidung
 - steuernde Anschlüsse
 - gesteuerte Anschlüsse (können auch wieder andere Anschlüsse steuern)
-
- **Taktzustandsgesteuertes Kippglied**



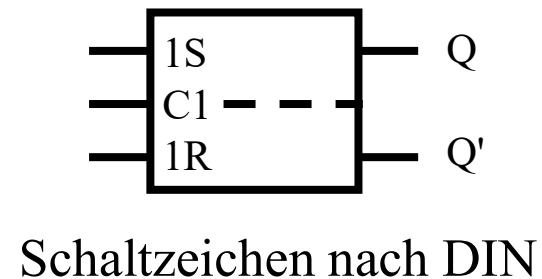
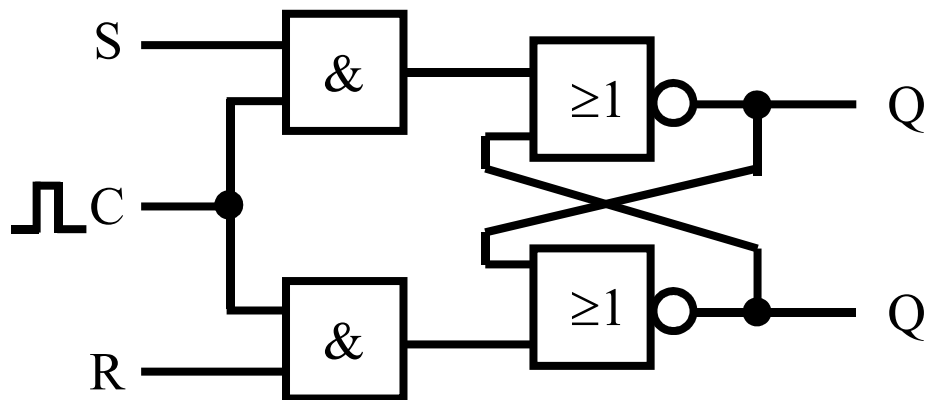
Taktabhängige Kippglieder

- **Taktzustandssteuerung**

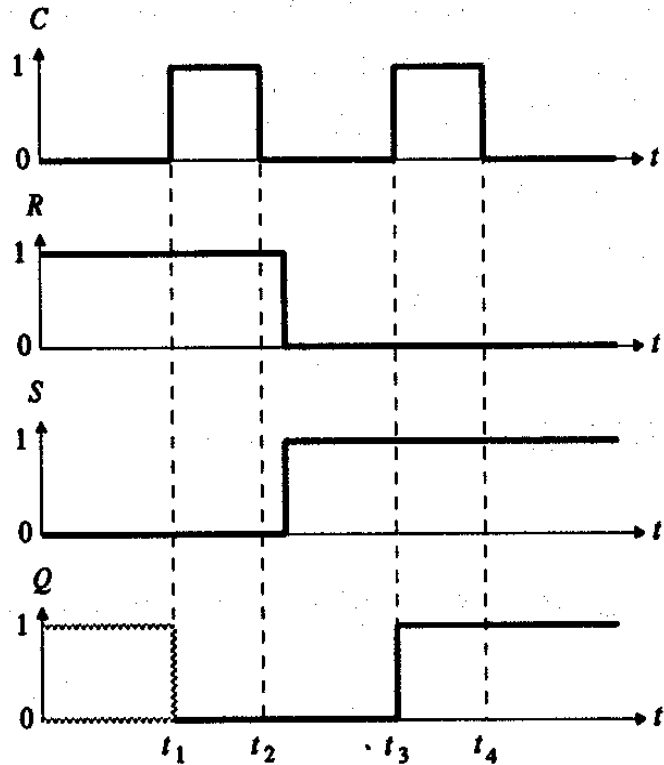
- Taktleitung C (Clock)



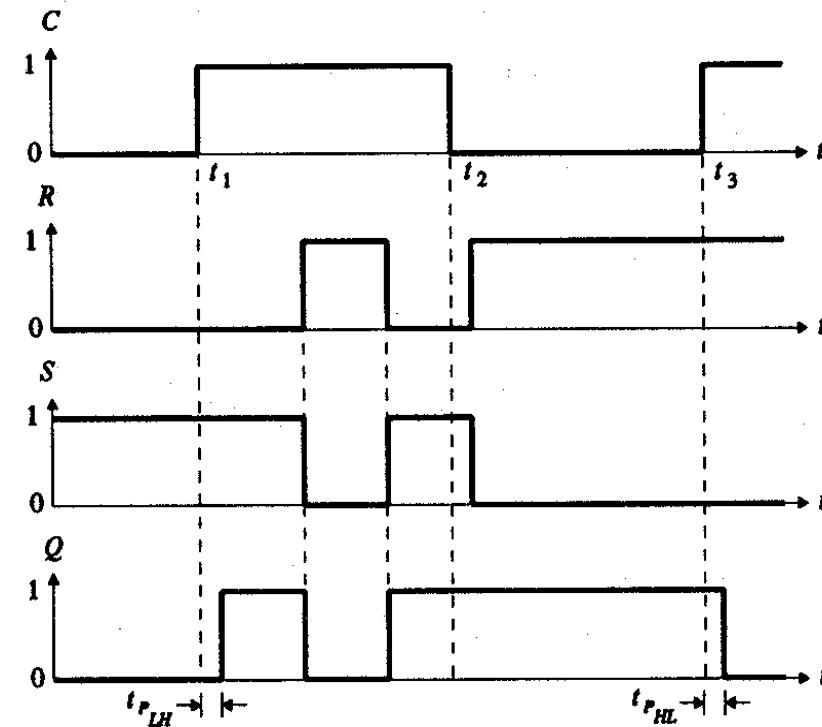
- nur bei $C=1$ (positive Taktphase) reagiert Latch auf Eingänge



Impulsdiagramm



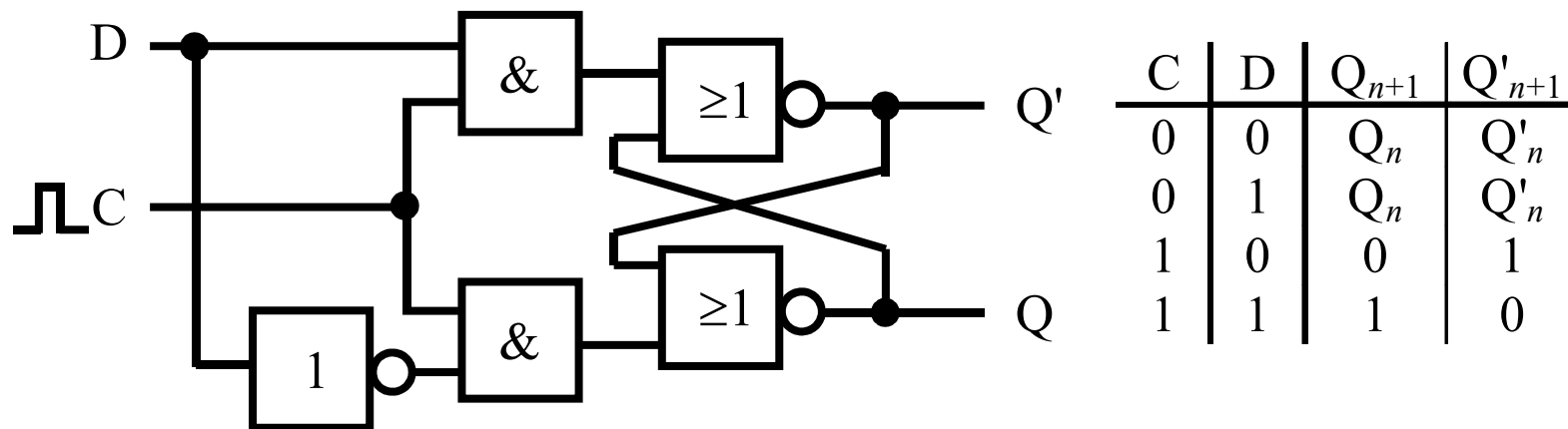
im inaktiven Zustand ($C=0$,
negative Taktphase) ist
Kippglied unanfällig gegen
Störungen (Hazards)



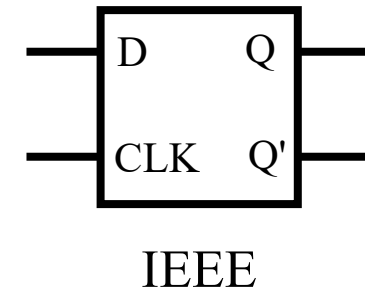
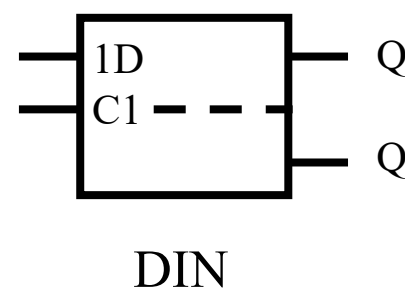
im aktiven Zustand ($C=1$,
positive Taktphase) können
Hazards immer noch zu
ungewünschten Zuständen
führen

D-Latch

- Kippglied ohne verbotene Eingangskombination
- reagiert auf den Zustand am Eingang D, der während $C=1$ anliegt
- auch **D**elay Latch genannt
- Eingangssignal sollte während $C=1$ stabil sein

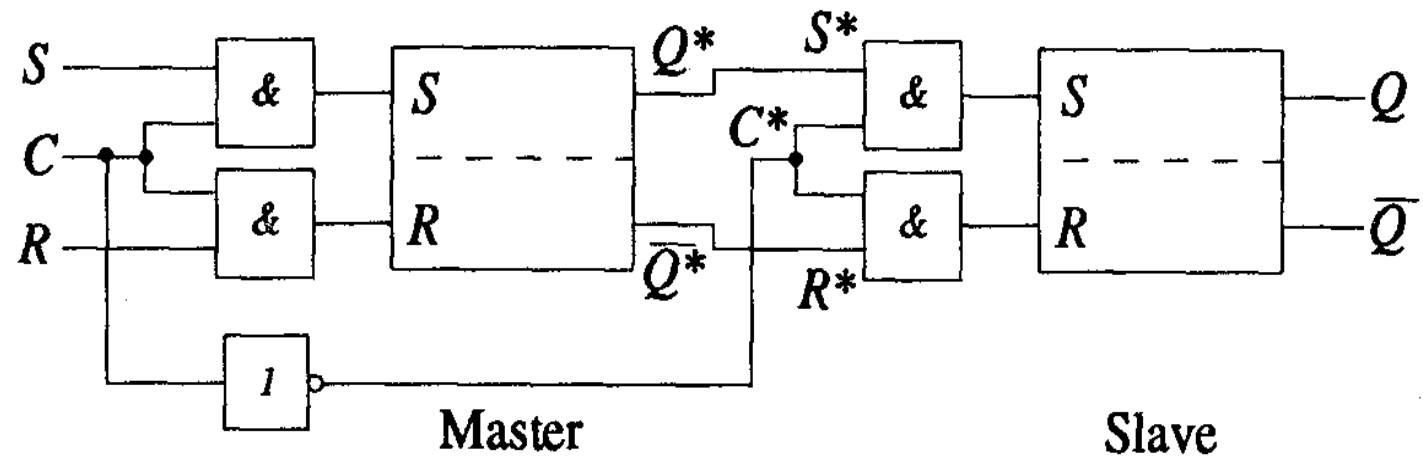


Schaltzeichen:

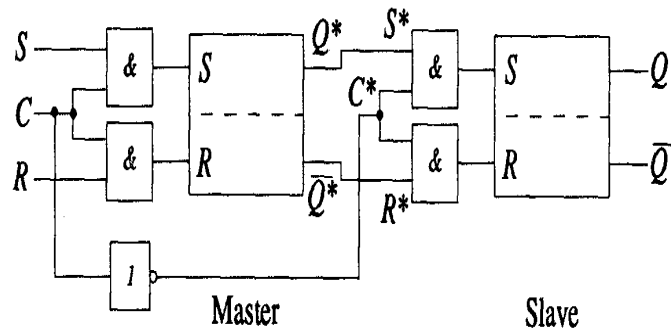


RS-Master-Slave-Kippglied

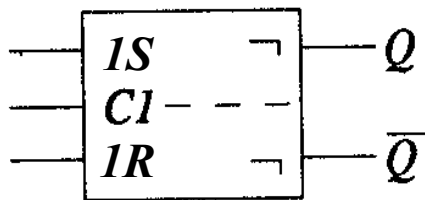
- **Kippglied mit Zweizustandssteuerung**
 - bei den bisher beschriebenen Kippgliedern ist in der aktiven Phase ($C=1$) der Ausgang unmittelbar abhängig vom Eingang
 - würde man mehrere Kippglieder hintereinander schalten, würde eine Änderung am Eingang durch alle Kippglieder hindurchrutschen
 - steuert man das zweite Kippglied mit dem inversen Taktsignal, wird das verhindert
 - Master-Slave-Kippglied: Flip-Flop



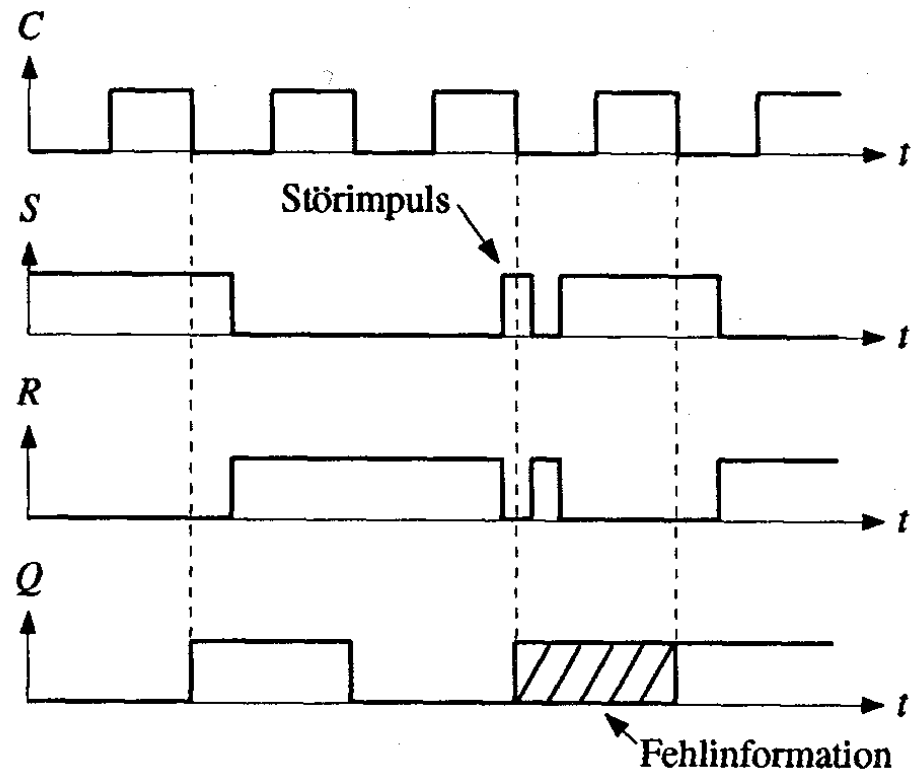
RS-Master-Slave-Kippglied (2)



a) Schaltung mit Master-Slave Prinzip



b) Schaltzeichen



c) Impulsdiagramm

RS-Master-Slave-Kippglied (3)

- zu jedem Zeitpunkt ist immer nur ein Kippglied im aktiven Zustand
- der Eingang ist nie direkt (kombinatorisch, über ein Schaltnetz) mit dem Ausgang verbunden
- das Eingangssignal wird bei $C=1$ übernommen, erscheint am Ausgang aber erst mit $C=0$
- der Ausgang ist *retardiert* (verzögert) und erhält deshalb das Symbol (\neg)
- auch hier kann es immer noch zu Fehlinformationen kommen, denn bei $C=1$ werden Fehlinformationen an den Eingängen wirksam

JK-Master-Slave-Kippglied (2)

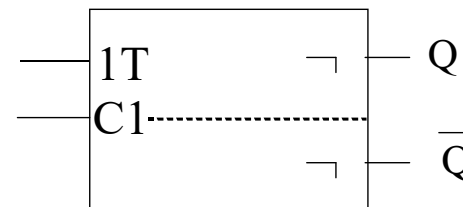
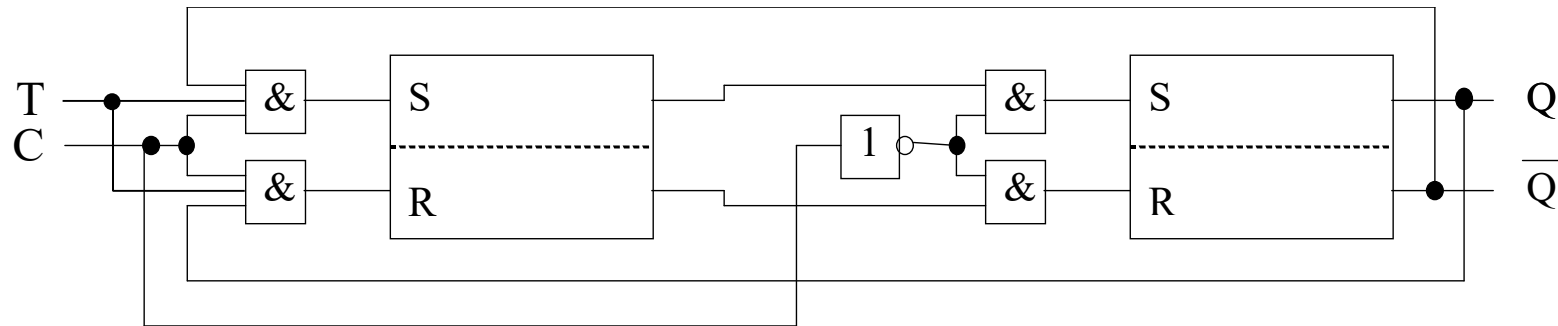
- beim Entwurf von Schaltwerken (s.u.) benötigt man die Eingangskombinationen in Abhängigkeit vom gewünschten Ausgangsverhalten

J	K	Q_{n+1}		Q_n	Q_{n+1}	J	K
0	0	Q_n	Speichern	0	0	0	-
0	1	0	Rücksetzen	0	1	1	-
1	0	1	Setzen	1	0	-	1
1	1	$\overline{Q_n}$	Kippen	1	1	-	0

- "-" steht für don't care, d.h. der Wert des Eingangssignals ist irrelevant
- da alle Eingangszustände zulässig und sinnvoll sind, ist das JK-Master-Slave-Kippglied vielseitig einsetzbar

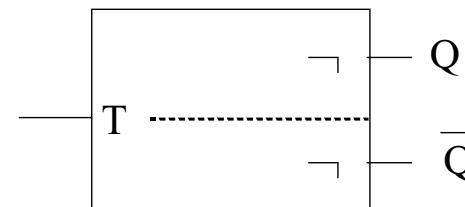
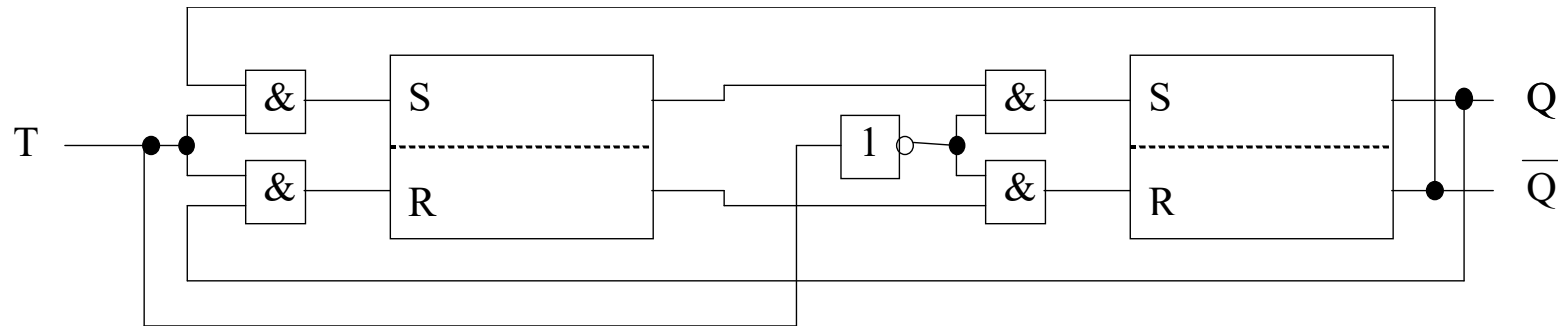
T-Master-Slave-Kippglied

- Eingänge J und K sind verbunden und werden T genannt
- Der Zustand kippt bei jedem Takt (Toggle-Kippglied), falls $T=1$
- sonst bleibt der alte Zustand erhalten



T-Master-Slave-Kippglied (2)

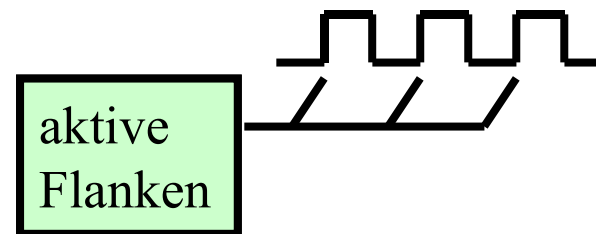
- manchmal wird auch ganz auf das Eingangssignal verzichtet
- das Kippglied toggelt dann mit jedem Takt
- das Taktsignal wird dann auch schon mal T genannt



Taktflankensteuerung

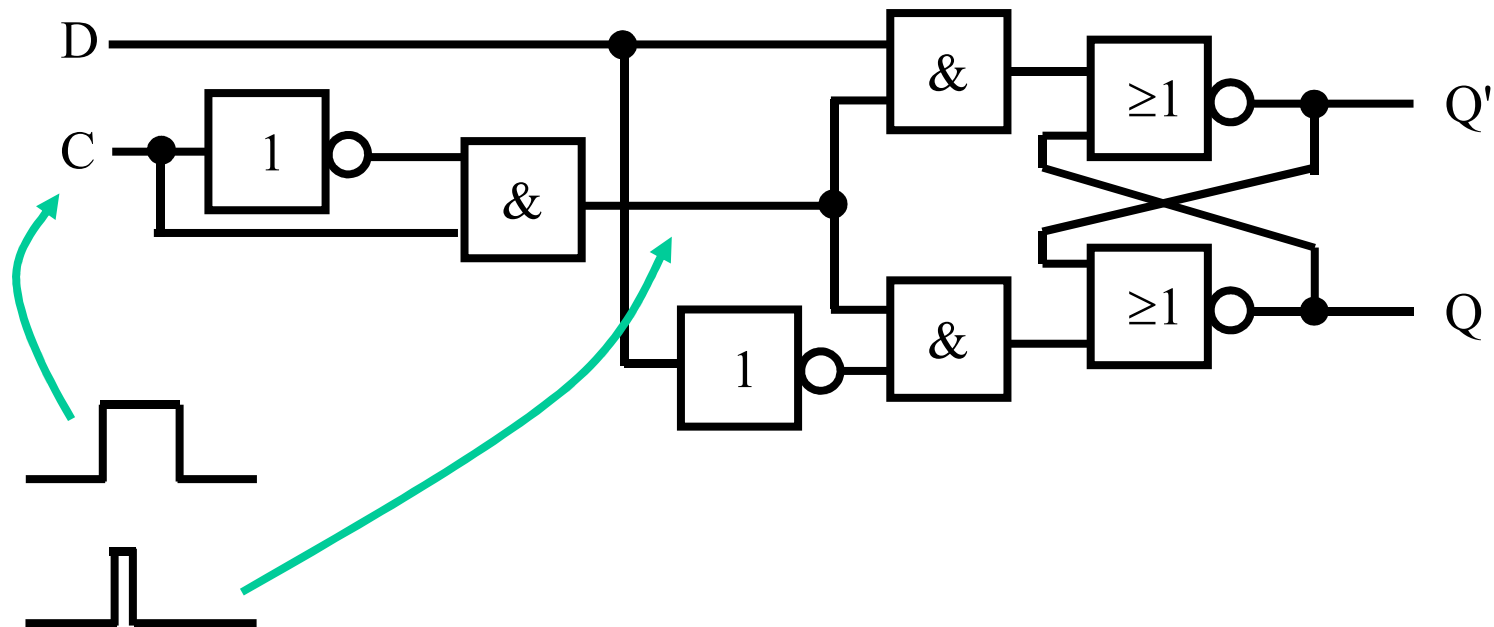
- **Taktflankensteuerung**

- häufig notwendig, dass ein Wert nur in einem bestimmten Augenblick übernommen wird
- prinzipiell mit sehr kurzen Taktimpulsen möglich
- technisch schwierig, extrem kurze Impulse zu erzeugen, die aber noch lang genug sind, damit die Schaltungen sicher darauf reagieren
- besser neue Art von Schaltung, die nur während des Übergangs von 0 nach 1 (oder von 1 nach 0) auf das Eingangssignal reagiert



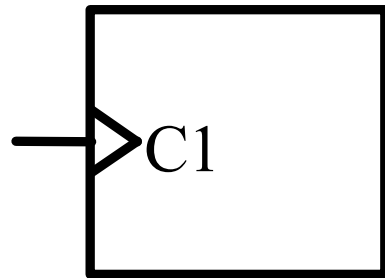
Taktflankensteuerung (2)

- Beispiel: D-Flip-Flop aus Grundgattern
- einfache (aber schlechte) Realisierung
 - Ausnutzen eines Hazards für die Erzeugung eines kurzen Impulses
 - Impuls-Timing ist kritisch
 - zu kurz: Flip-Flop reagiert nicht
 - zu lang: Signal am D-Eingang kann sich schon wieder geändert haben

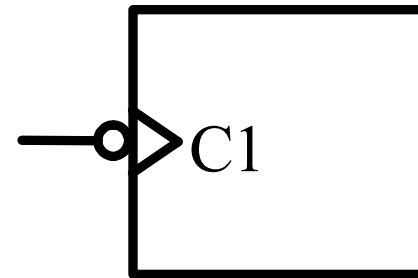


Taktflankensteuerung (3)

- **Schaltzeichen**



Eingänge wirksam
beim 0-1-Übergang

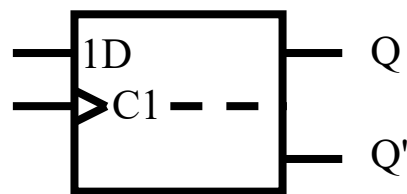


Eingänge wirksam
beim 1-0-Übergang

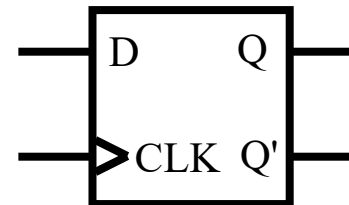
- alle Kippglieder mit Taktzustandssteuerung können auch mit Taktflankensteuerung realisiert werden
- Übergangsfunktionen und Zustandsfolgetabellen bleiben unverändert

D-Flip-flop

- **D-Kippglied mit Taktflankensteuerung**
 - Kippglieder mit Taktflankensteuerung heißen ebenfalls Flip-Flop
 - Flankensteuerung begrenzt die Übernahme von Daten auf einen Zeitpunkt
 - entweder: ansteigende Flanke (positive Flanke)
 - oder: abfallende Flanke (negative Flanke)
 - Schaltzeichen D-Flip-Flop (Speicherung bei positiver Flanke)



DIN



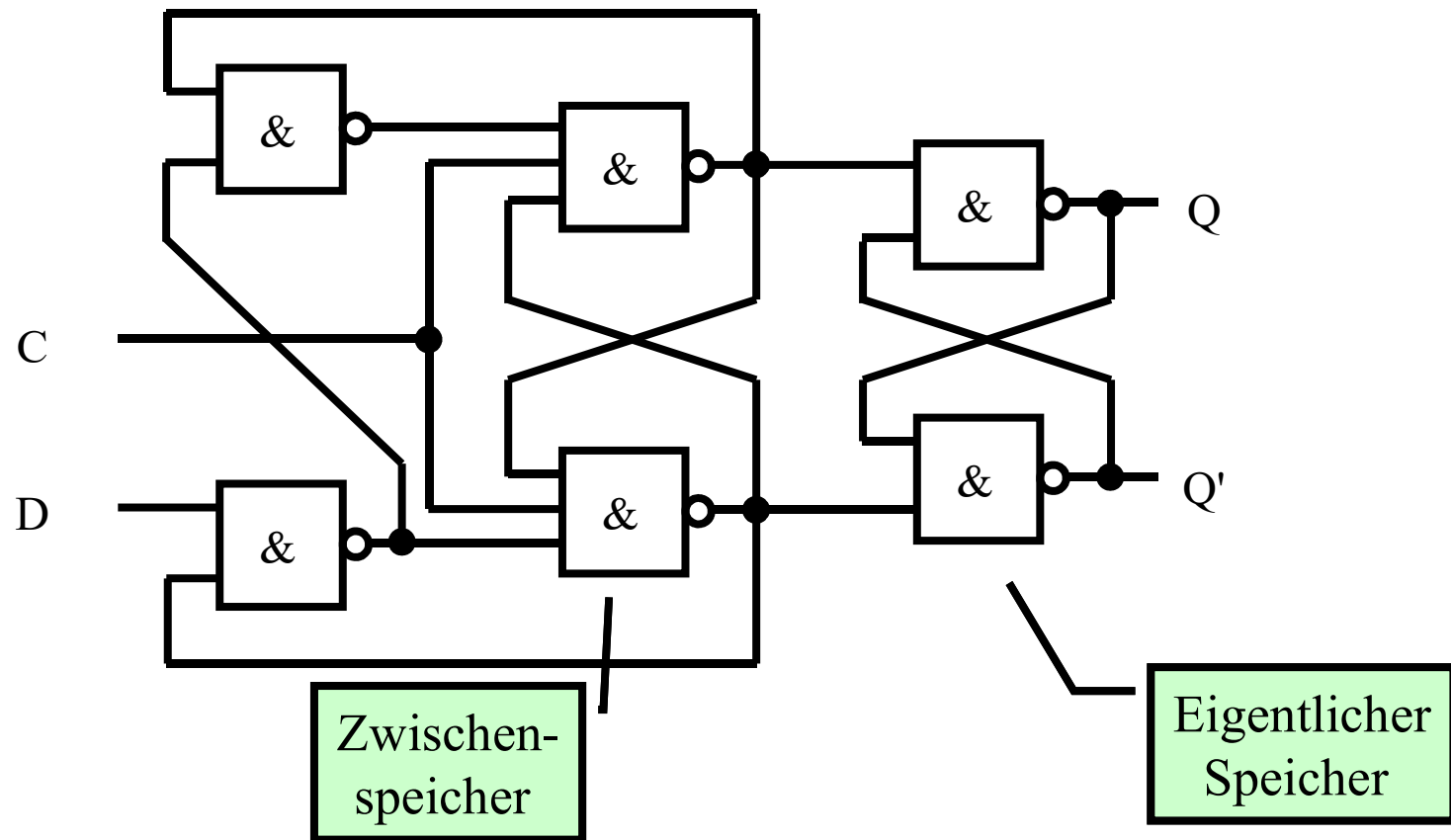
IEEE

Echte Taktflankensteuerung

- **Beispiel: D-Flip-Flop**

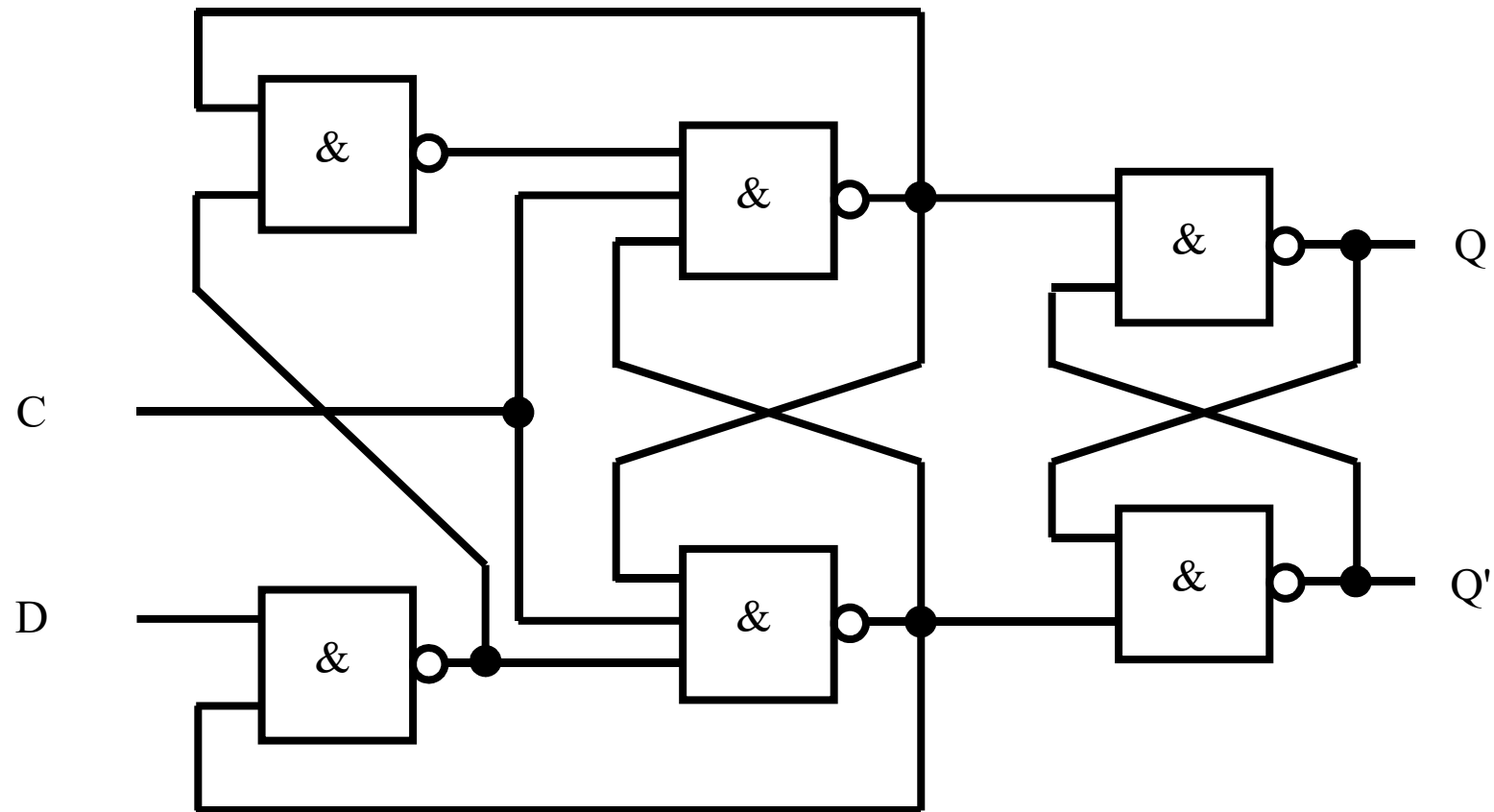
- wesentliches Element:

- Zwischenspeicher, der über eine Rückkopplung seine Eingänge kontrolliert



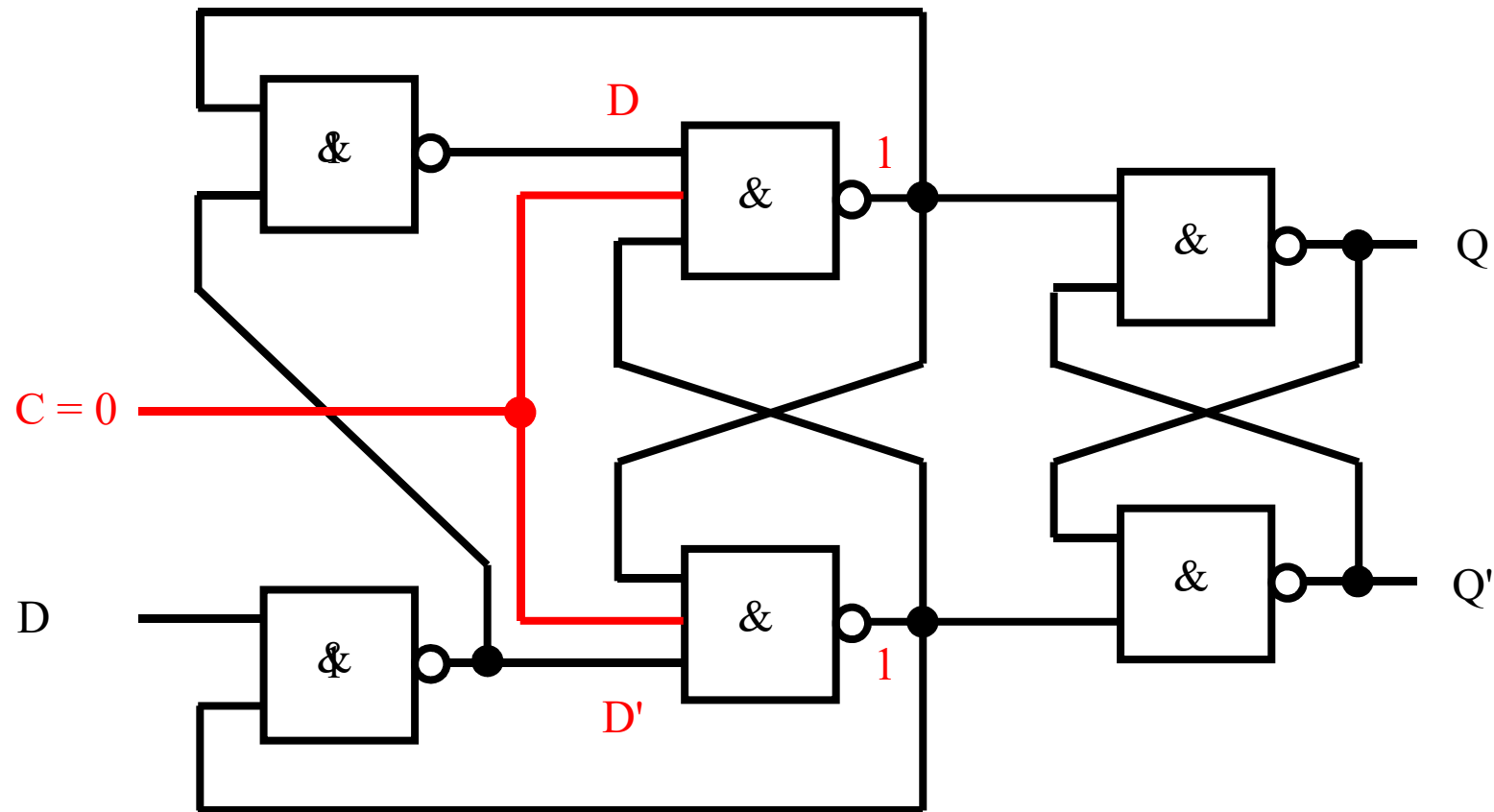
[illegible]

- **D-Flip-Flop**



Echte Taktflankensteuerung (3)

- **D-Flip-Flop: $C = 0$**

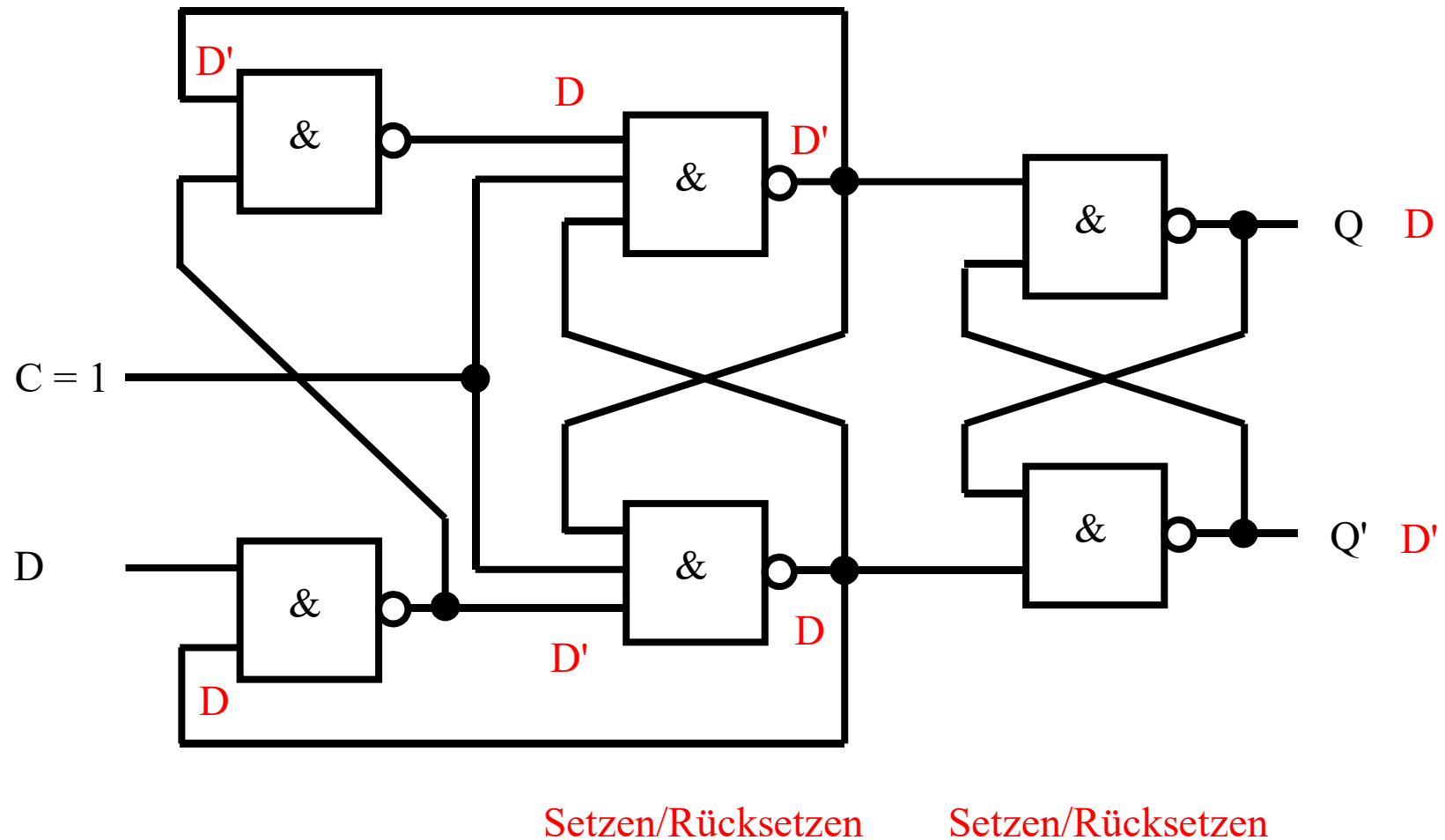


"verbotener" Zustand

Speichern

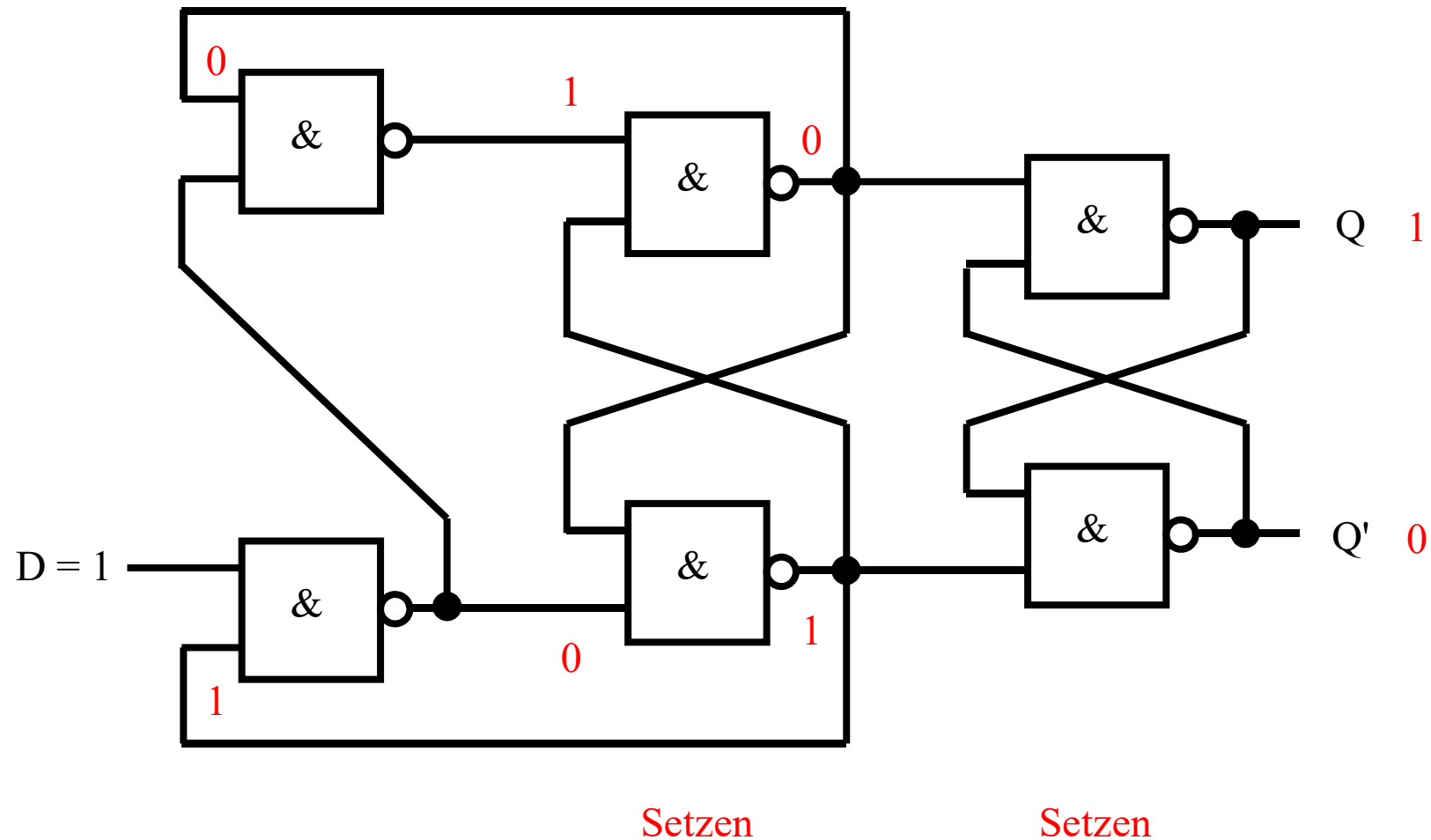
Echte Taktflankensteuerung (4)

- D-Flip-Flop, $C = 0 \rightarrow 1$



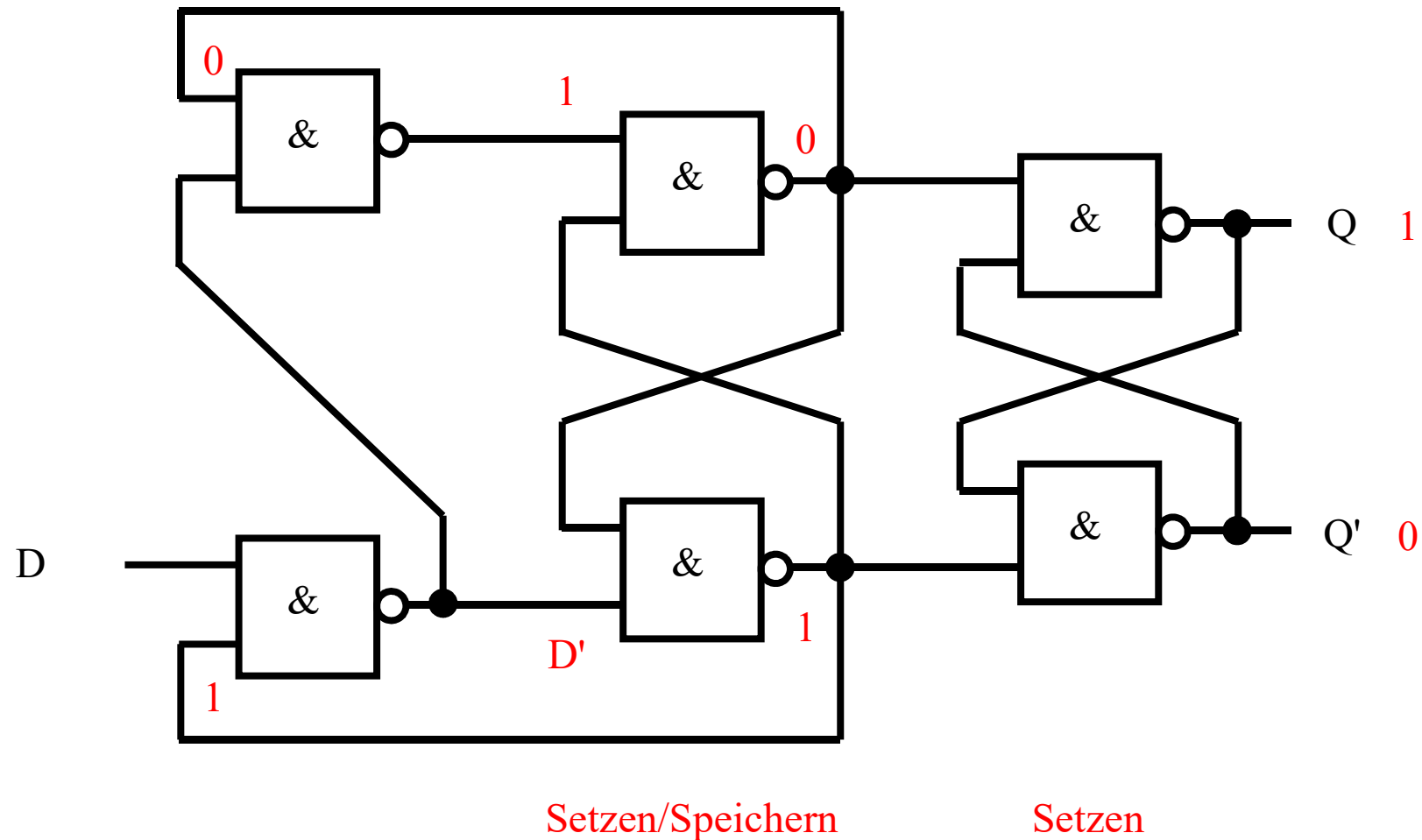
Echte Taktflankensteuerung (5)

- **D-Flip-Flop, $C = 0 \rightarrow 1$, o.B.d.A. $D = 1$**



Echte Taktflankensteuerung (6)

- D-Flip-Flop, $C = 1$, D jetzt wieder beliebig



Echte Taktflankensteuerung (7)

- **Noch einmal in Worten:**
- **$C = 0$**
 - Zwischenspeicher ist im eigentlich “verbotenen” Zustand (wird hier aber sinnvoll genutzt)
 - beide Ausgänge auf 1
 - dadurch Eingänge freigeschaltet
 - Eingangssignal liegt normal und invertiert am Zwischenspeicher an
 - der eigentliche Speicher speichert noch den alten Wert
 - Änderungen am Eingang D wirken sich nicht aus
- **$C = 0 \rightarrow 1$**
 - Zwischenspeicher übernimmt die Eingabe D
 - gleichzeitig übernimmt der eigentliche Speicher auch die Eingabe D

Echte Taktflankensteuerung (8)

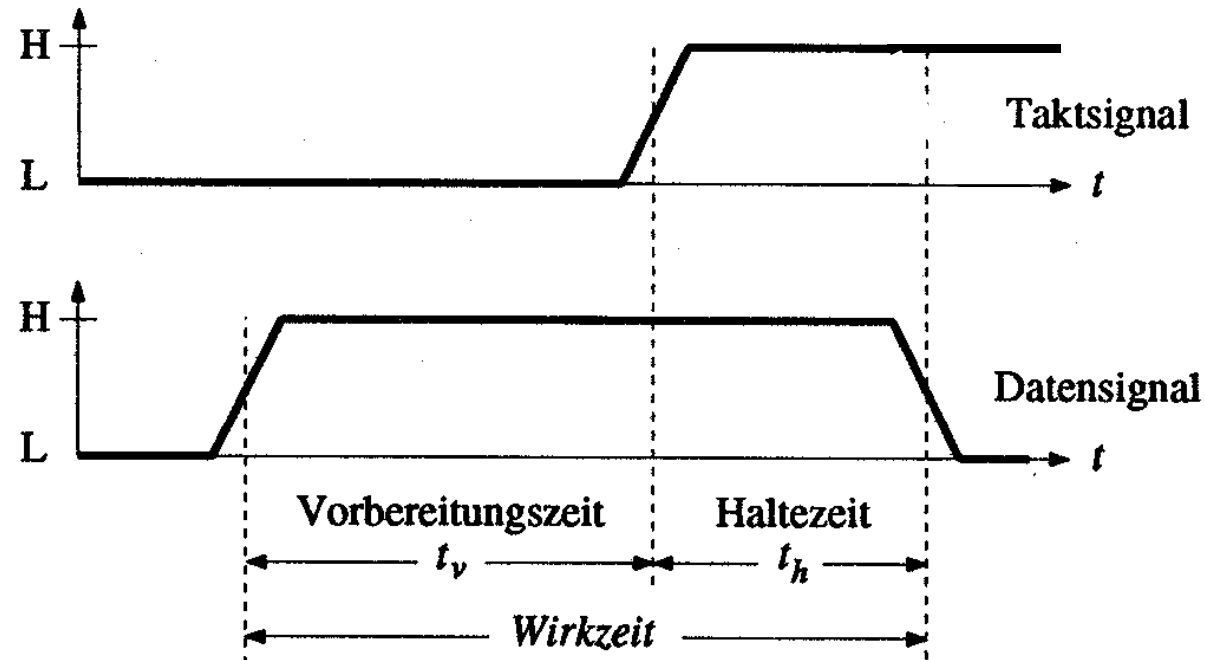
- $C = 1$
 - Zwischenspeicherausgang mit 0 sperrt seinen eigenen Eingang (geht auf 1)
 - dadurch wird verhindert, dass durch Rücksetzen seines Eingangs der zwischengespeicherte Wert verändert werden kann
 - der andere Eingang ist beliebig (1: speichern, 0: Erzwingen des ohnehin gespeicherten Wertes)
 - damit kann der Zwischenspeicher den beim Übergang $C = 0 \rightarrow 1$ übernommenen Wert nicht mehr verändern
 - der eigentliche Speicher kann seinen Wert daher auch nicht mehr verändern

Echte Taktflankensteuerung (9)

- **Zusammenfassung**

- Taktflankensteuerung basiert auf dem Effekt, dass der Zwischenspeicher, sobald er die Eingangsdaten übernommen hat, den kritischen Eingang (auf den der Zwischenspeicher reagieren würde) über eine Rückkopplung sperrt
- dadurch reagiert der Zwischenspeicher nur beim Übergang $C = 0 \rightarrow 1$, sowohl bei $C=0$ als auch bei $C=1$ wirken sich Änderungen von D nicht aus
- dies funktioniert nur, weil Signallaufzeiten in den Verknüpfungsgliedern ausgenutzt werden
- dazu müssen mehrere Bedingungen erfüllt sein:

Setup- und Hold-Zeiten



- das Eingangssignal D muss mindestens zwei Gatterverzögerungszeiten (NAND-Gatter) vor der aktiven Flanke des Taktsignals ($0 \rightarrow 1$) anliegen
 - diese Zeit heißt Vorbereitungszeit t_v (*setup time*)
- das Eingangssignal D muss mindestens so lange stabil bleiben, bis die Rückkopplung die Eingänge sperrt
 - diese Zeit heißt Haltezeit t_h (*hold time*)

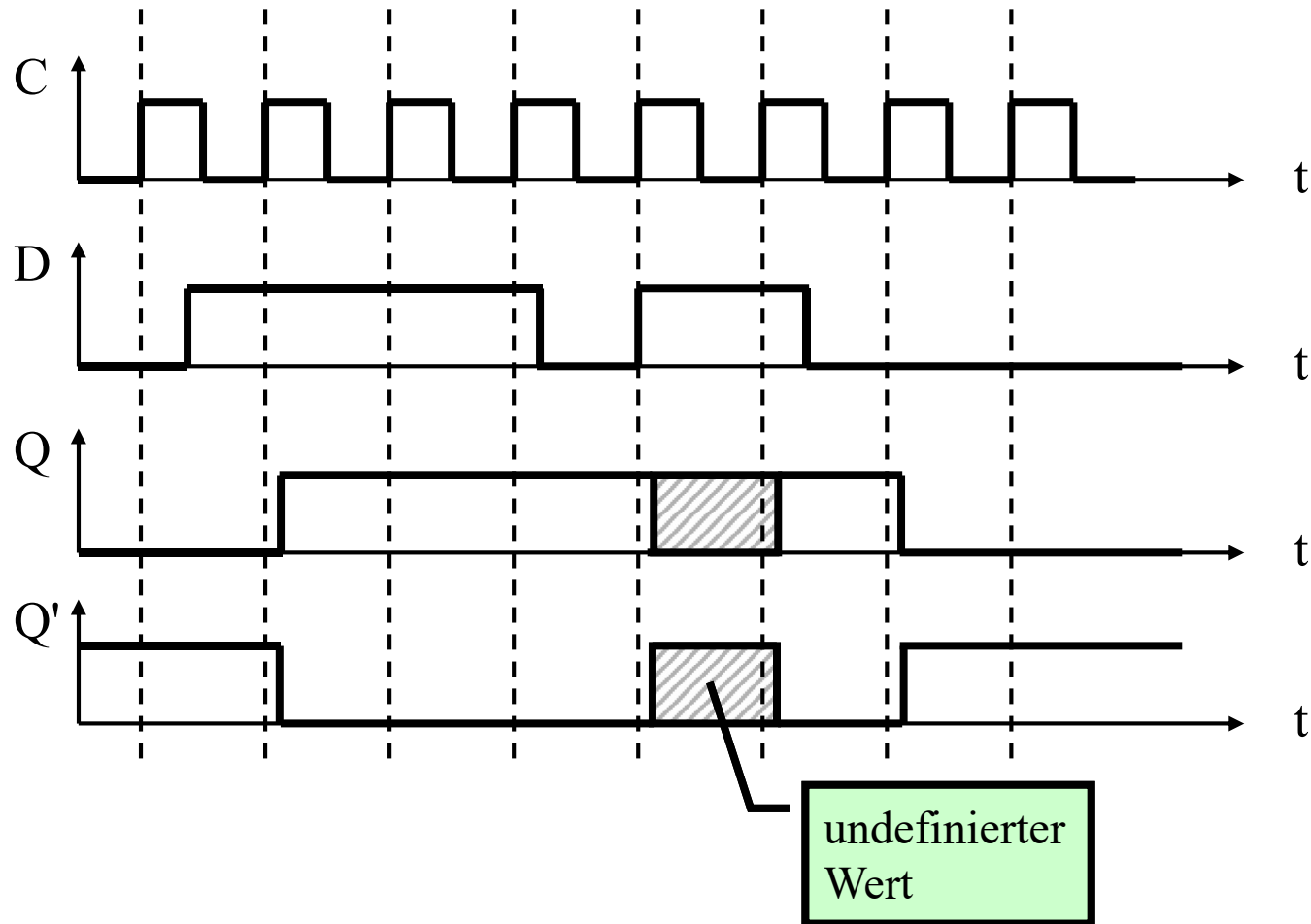
Setup- und Hold-Zeiten (2)

- die Pulsdauer des Taktsignals (Dauer von $C=1$) muss mindestens so groß sein, wie die Signallaufzeit durch die NAND-Glieder des Zwischenspeichers, damit der eigentliche Speicher sicher gesetzt werden kann
- die Pulsdauer t_p muss mindestens so groß sein wie die Haltezeit t_h
- daraus ergibt sich, dass die Eingangssignale mindestens für die Zeit $t_v + t_h$ konstant sein müssen
 - diese Zeit heißt im deutschen Sprachraum auch **Wirkzeit**, weil für diese Zeit die Eingangsvariable wirksam sein muss
 - eine gebräuchliche englische Übersetzung scheint es nicht zu geben

Setup- und Hold-Zeiten (3)

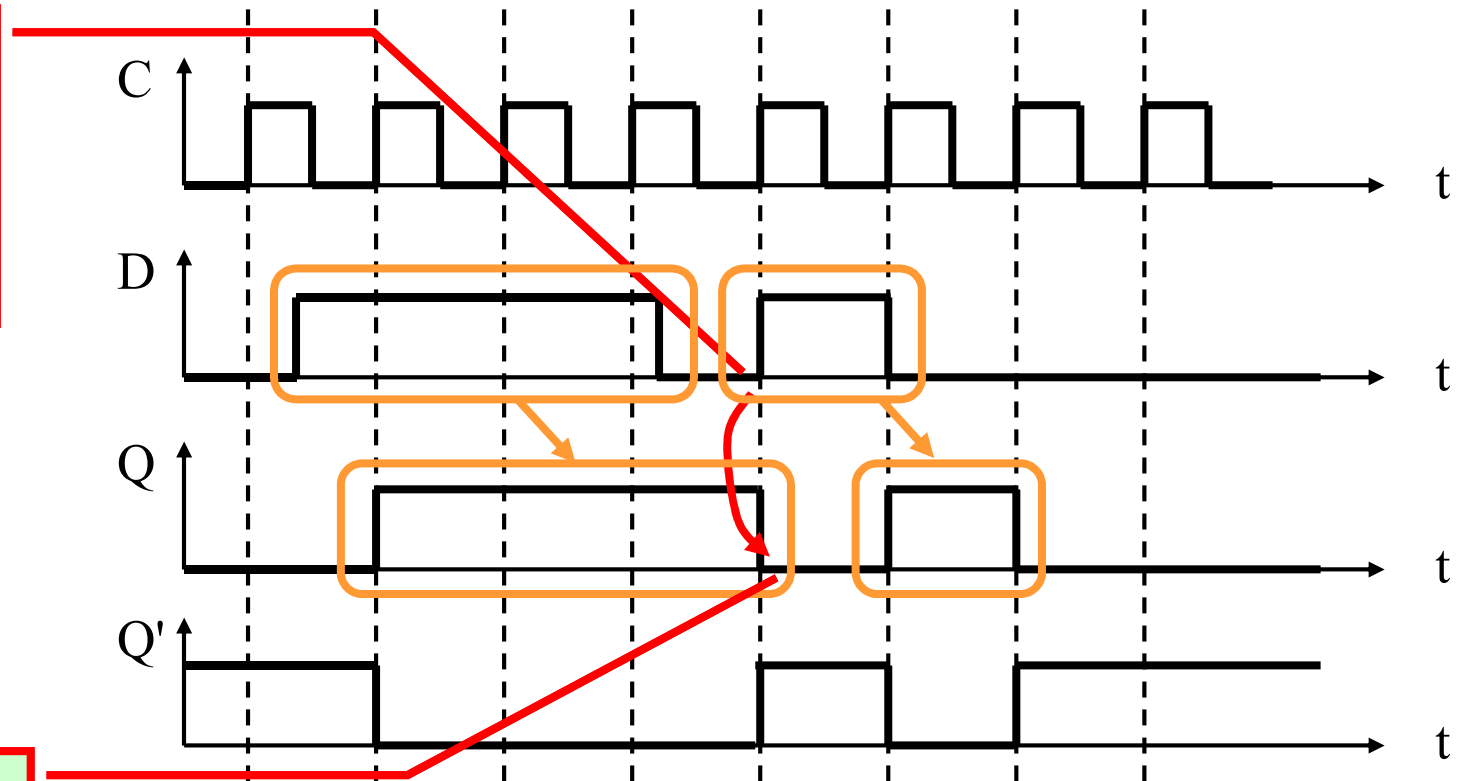
- in der Praxis sind die Schaltungen häufig so ausgelegt, dass die Haltezeit den Wert Null hat
 - das bedeutet, dass das Eingangssignal (bzw. die Eingangssignale, z.B. beim JK-Flip-Flop) einige Zeit vor der aktiven Taktflanke stabil sein muss, nach der Flanke können sich die Eingangssignale sofort wieder ändern
 - kann man durch interne Verzögerung des Datensignals erreichen, z.B. durch eine gerade Anzahl von in Reihe geschalteten Invertern

Impulsdiagramm D-Flip-Flop (real)

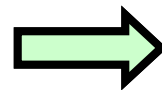


Impulsdiagramm D-Flip-Flop (idealisiert)

Eingang:
Übernahme
des Wertes
vor der
Takt-Flanke



Ausgang:
Wert erscheint
erst *nach* der
Takt-Flanke



Signale werden durch Flip-Flops immer
auf die nächste Taktflanke synchronisiert

Master-Slave und Taktflankensteuerung

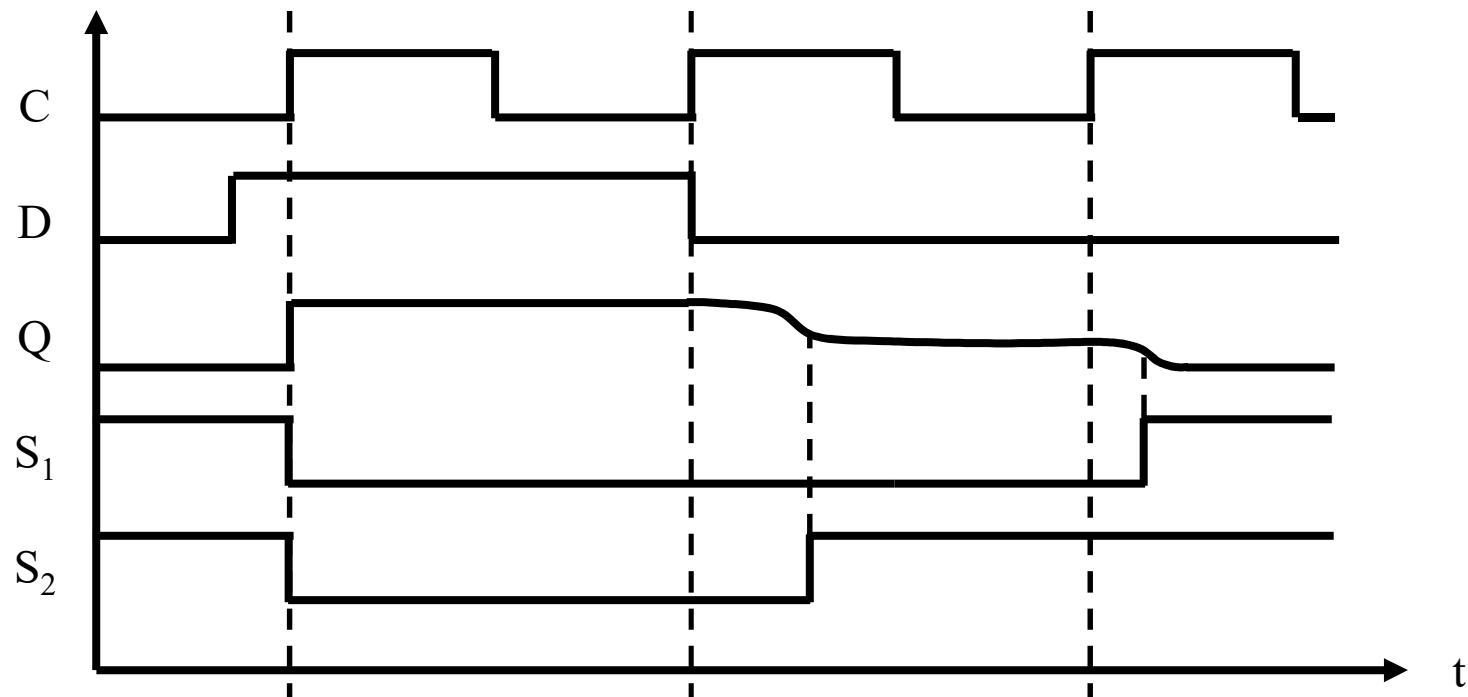
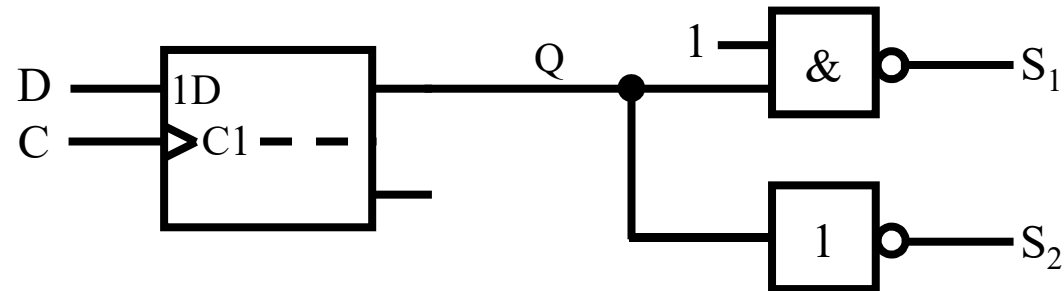
- **Master-Slave-Flip-Flops**
 - können auch mit Taktflankensteuerung realisiert werden
 - Master-FF wird durch positive (ansteigende) Flanke getriggert
 - Slave-FF durch negative (abfallende) Flanke getriggert

Metastabilität

- **Problem**

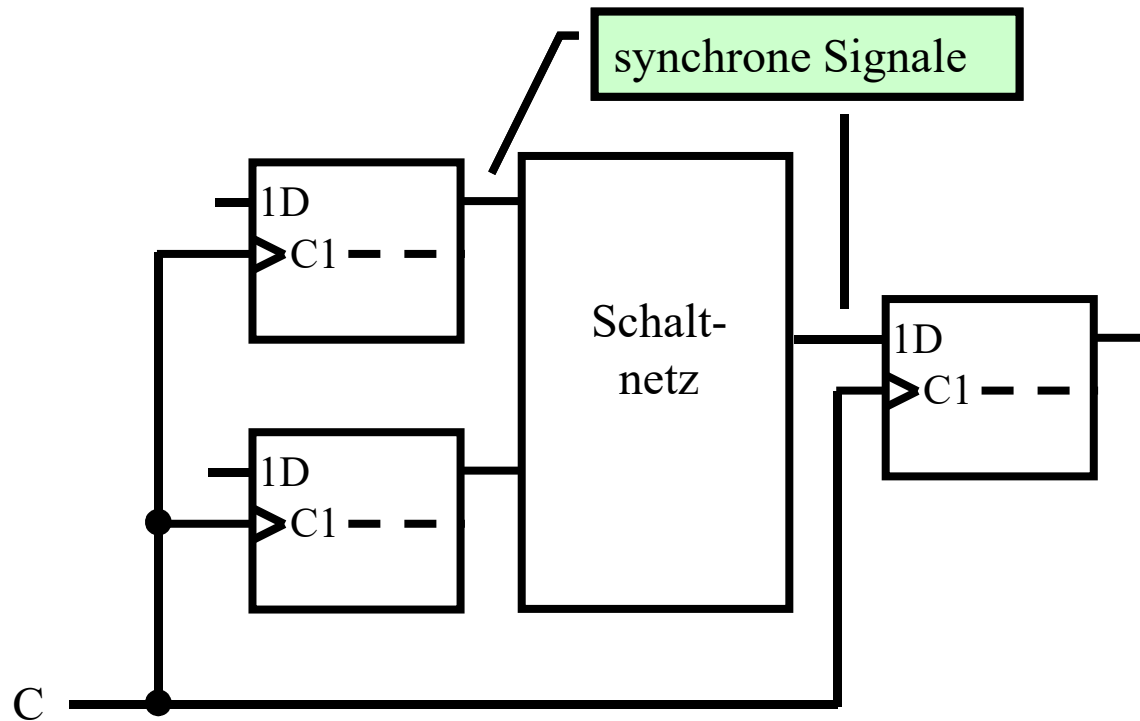
- Setup- und Holdzeiten können zwar klein sein, sind aber immer vorhanden
- wenn das Eingangssignal während der Setup- oder Holdzeit den Wert ändert, muss sich das Flip-Flop entscheiden, ob die Signalfanke oder die Taktflanke zuerst da war
- das kann dazu führen, dass das Ausgangssignal (sehr) lange braucht, um einen stabilen Wert anzunehmen
 - es kann sogar eine Zeit lang oszillieren
 - kann mehrere Takte dauern
- muss unbedingt verhindert werden, da verschiedene nachfolgende Gatter dasselbe Ausgangssignal unterschiedlich interpretieren können

Metastabilität (2)



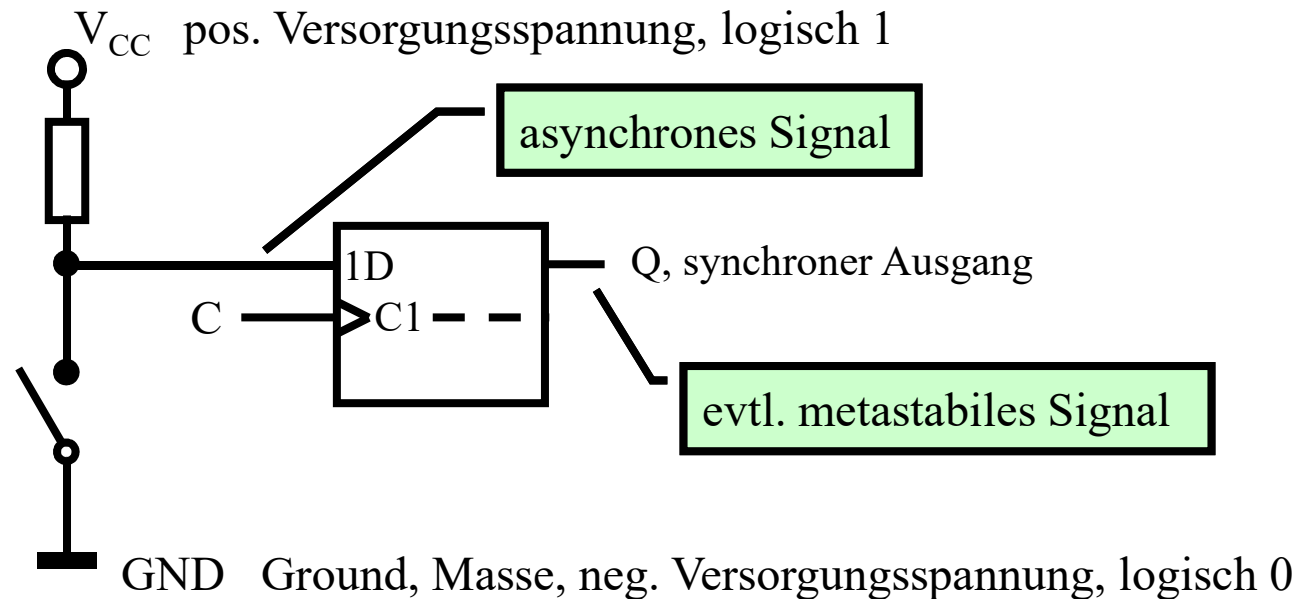
Metastabilität (3)

- **kein Problem bei normalen, getakteten Systemen (s.u.)**
 - Signale ändern sich immer kurz nach einer Taktflanke
 - treffen daher erst nach Ablauf der Holdzeit (meist gleich 0s) beim nächsten Kippglied ein
 - zusätzliche Verzögerungen durch dazwischen liegende Schaltnetze



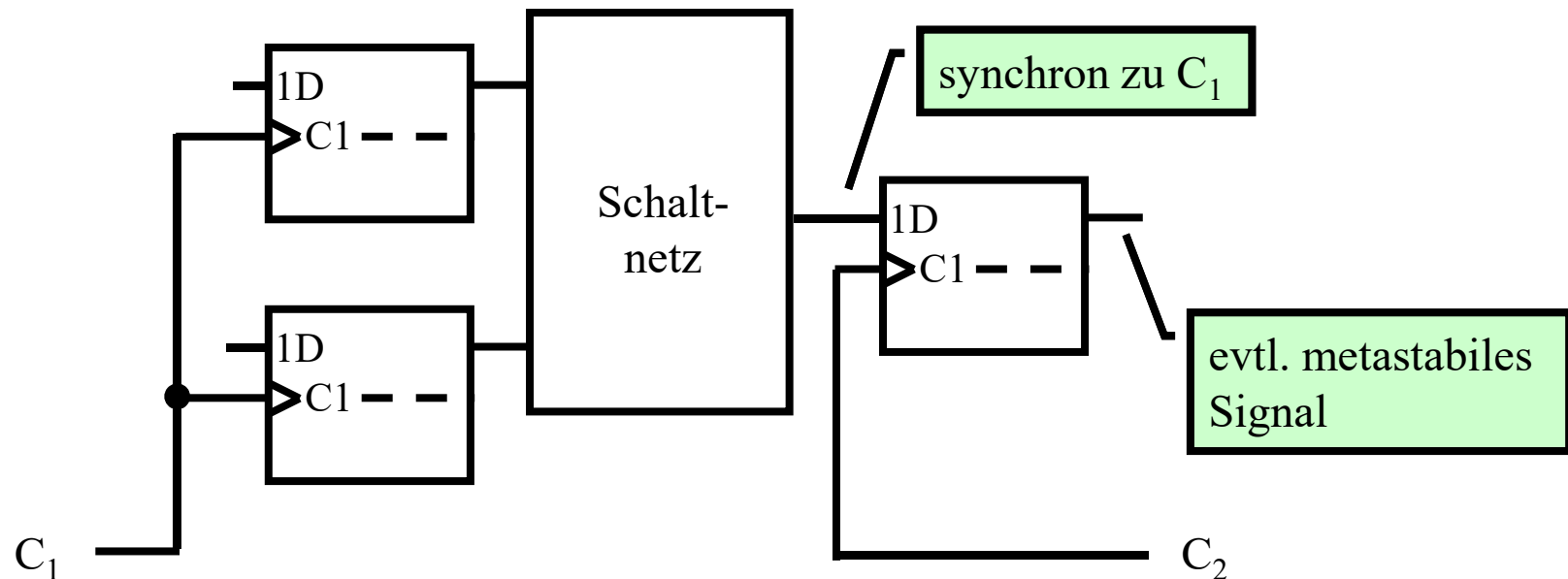
Metastabilität (4)

- **prinzipielles Problem bei**
 - asynchronen Signalen
 - externe Signale, die sich zu jeder Zeit ändern können, z.B.
 - handbediente Schalter
 - Sensoren, etc.



Metastabilität (5)

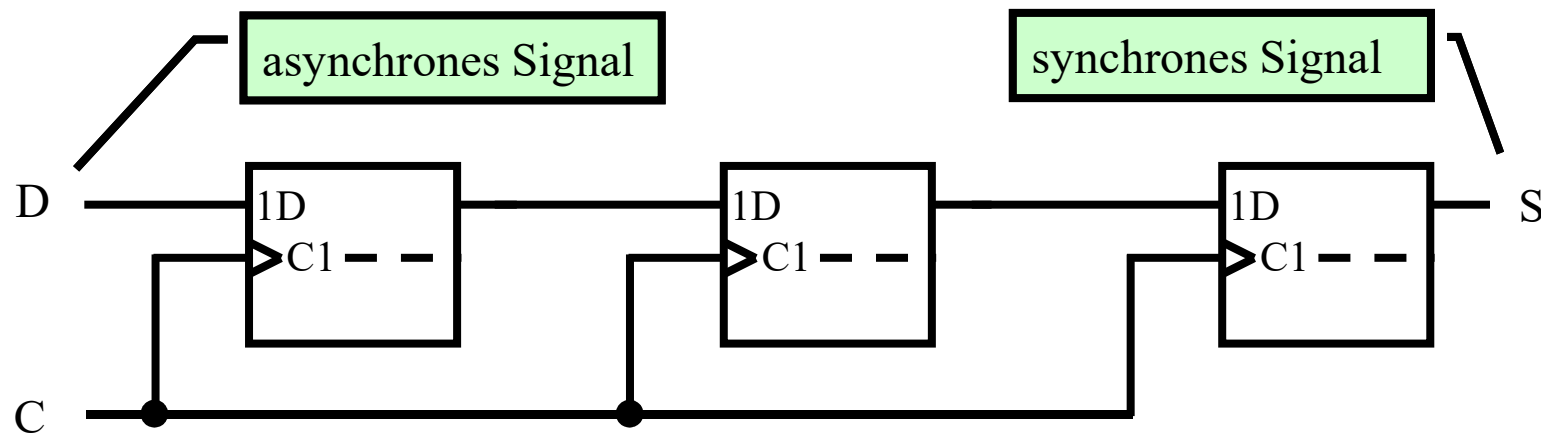
- mehrere Clock-Domänen
 - d.h., wenn verschiedene Teile der Schaltung unterschiedliche Taktfrequenzen haben
 - dann verschieben sich die Taktflanken gegeneinander und es kann passieren, dass Signaländerungen bewirkt durch den Takt in Domäne C_1 die durch in die Setup- oder Holdzeiten der Kippglieder in Domäne C_2 hineinfallen



Metastabilität (6)

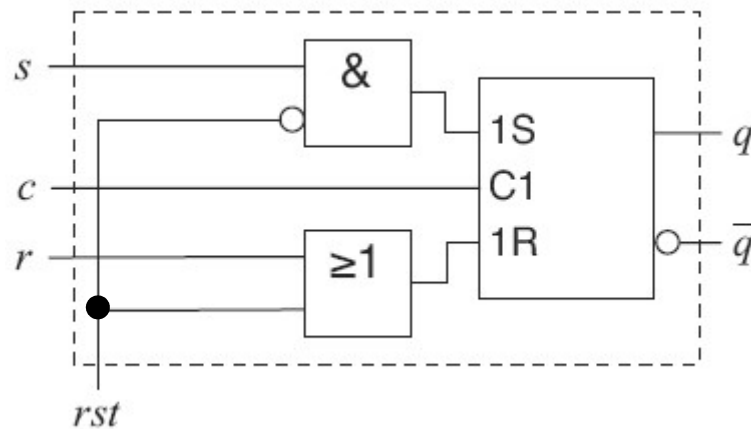
- **Abhilfe**

- man kann Metastabilität nicht komplett verhindern
 - man kann nur versuchen, Metastabilität sehr unwahrscheinlich zu machen
- mehrere (z.B. 3 oder 4) synchronisierende Flip-Flops hintereinander
 - die Wahrscheinlichkeit für Metastabilität ist das Produkt der Wahrscheinlichkeiten für die einzelnen Flip-Flops
 - kann daher beliebig klein gemacht werden (aber eben nicht auf 0 gedrückt werden)
 - synchronisiertes Signal ist um ein paar Takte verzögert

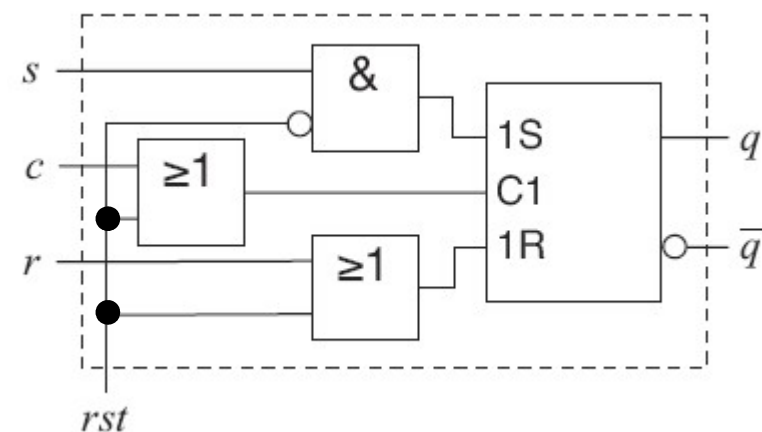


Bevorrechtigte Eingänge

- häufig haben Kippglieder weitere, bevorrechtigte Eingänge, mit denen man, insbesondere nach dem Einschalten, einen bestimmten Zustand erzwingen kann
- Beispiele für RS-Latches:



synchroner Reset-Eingang:
setzt Kippglied bei
positiver Taktphase auf 0

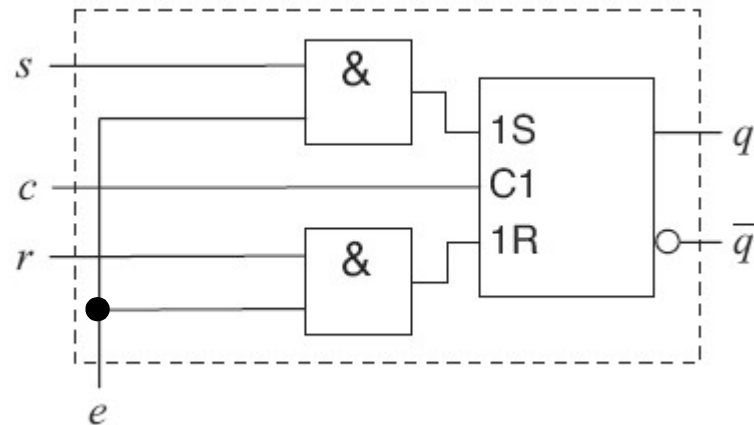


asynchroner Reset-Eingang:
setzt Kippglied unmittelbar auf 0

Bevorrechtigte Eingänge (2)

- **Enable-Eingänge**

- engl. *enable* (deutsch: aktivieren, anschalten, ermöglichen)
- aktivieren die Schaltung
- durch ein gesetztes Enable-Signal ($e=1$) wirkt sich das Taktsignal erst aus



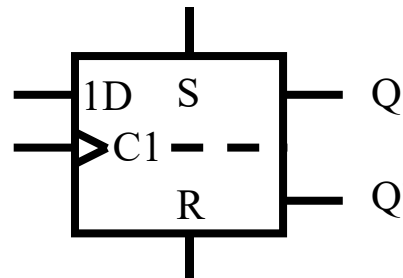
Clock-Enable e :
nur bei $e=1$ wirkt sich das Taktsignal aus

Asynchrone Eingänge

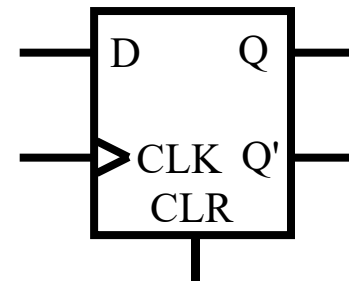
- **Flip-Flops**

- auch flankengetriggerte Speicherglieder haben häufig zusätzliche asynchrone Eingänge
- z.B. Reset oder Set (manchmal auch Clear und Preset genannt), auf die die Flip-Flops unabhängig vom Takt reagieren
- dienen zum Initialisieren z.B. nach dem Einschalten der Spannungsversorgung

Schaltbilder:



DIN



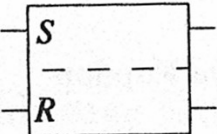
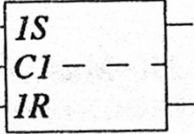
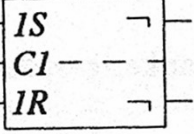
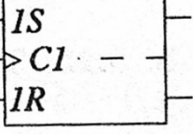
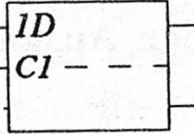
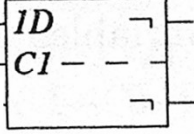
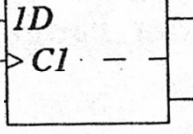
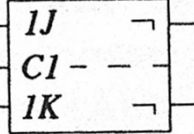
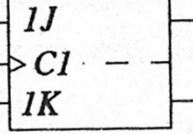
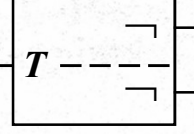
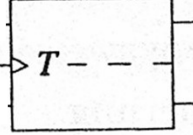
IEEE

Namensgebung von Kippgliedern

- **Bemerkung**

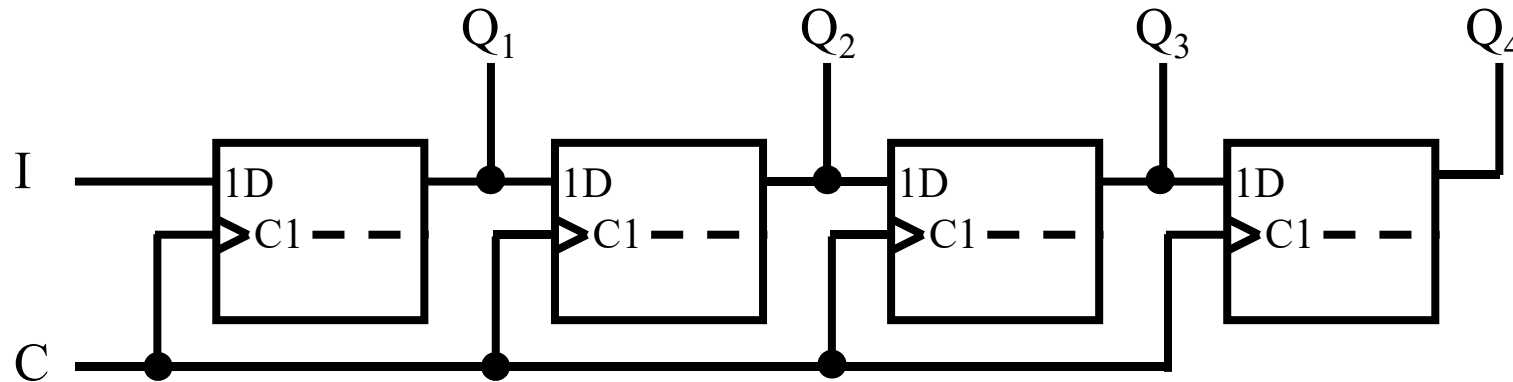
- Latches
 - Ausgangssignale können direkt von den Eingangssignalen abhängen
 - Kippglieder ohne Taktsteuerung
 - Kippglieder mit Taktzustandssteuerung (kein Master-Slave)
- Flip-Flops
 - Ausgangssignale ändern sich nur mit der Taktflanke
 - Master-Slave-Kippglieder
 - Kippglieder mit Taktflankensteuerung
- in der Literatur geht das allerdings häufig durcheinander

Zusammenfassung Kippglieder

	ohne Takt- steuerung	Zustands- steuerung	Master Slave	Taktflanken- steuerung
RS				
D				
JK				
T				

Anwendungen von Flip-Flops

- Schieberegister



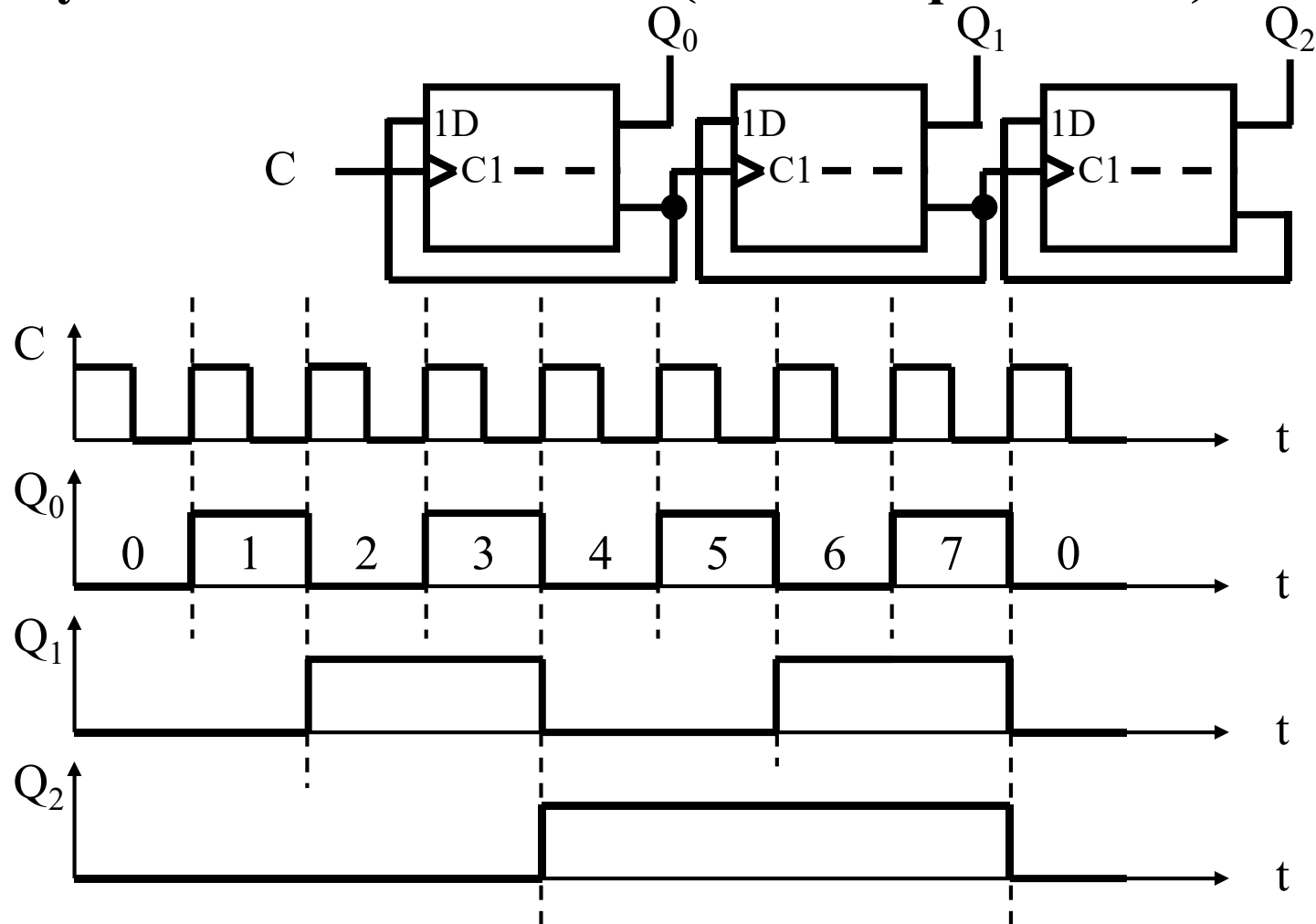
- Eingabe I wird bei jedem Takt um eine Stelle weiter geschoben
- Seriell-parallel Wandlung

$I = 1011$ \Rightarrow

t	I	Q ₁	Q ₂	Q ₃	Q ₄
0	1	?	?	?	?
1	1	1	?	?	?
2	0	1	1	?	?
3	1	0	1	1	?
4	?	1	0	1	1

Anwendungen von Flip-Flops (2)

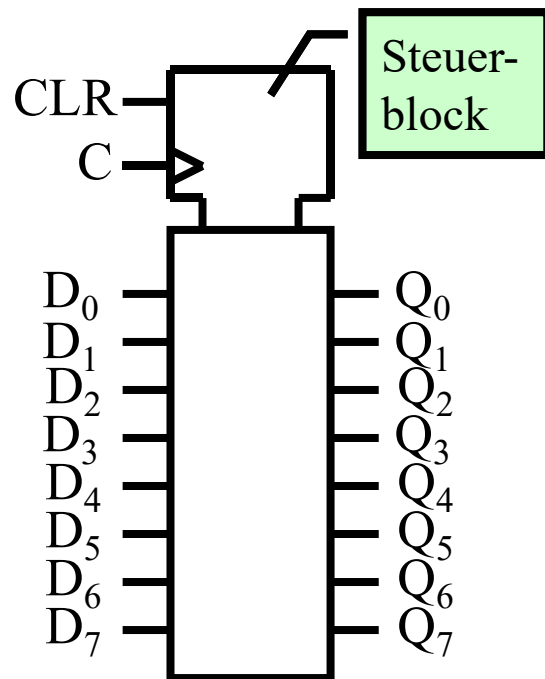
- asynchroner Binär-Zähler (oder Frequenzteiler)



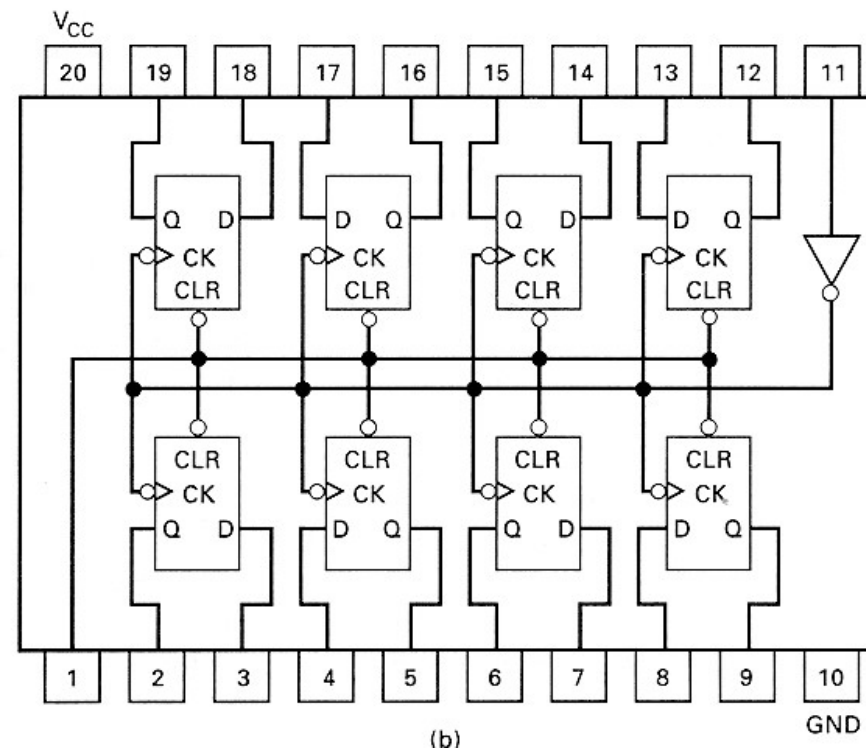
Anwendungen von Flip-Flops (3)

- **Register**

- Zusammenfassung von mehreren (Wortbreite) Flip-Flops mit gemeinsamen Steuerleitungen

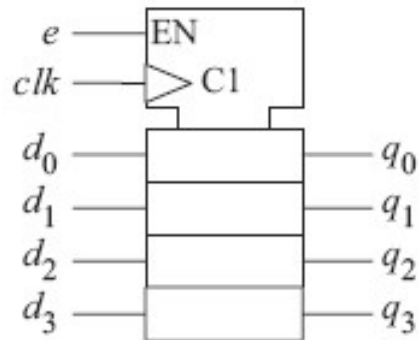


Ansicht realer Chip mit 20 Anschlüssen:
An V_{CC} wird der Pluspol und an GND der Minuspol der Spannungsversorgung angeschlossen



Auffangregister

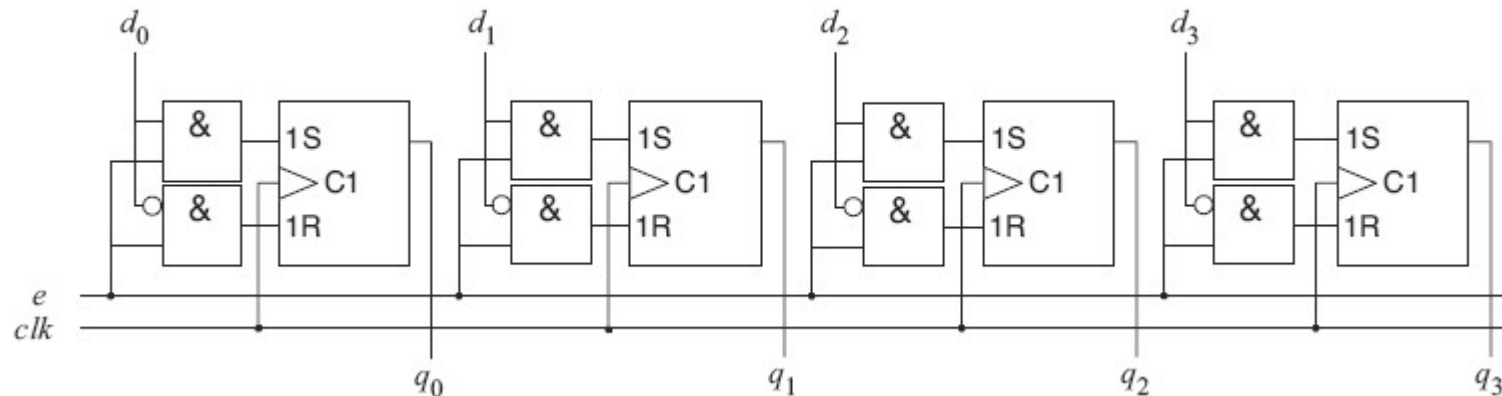
- **wesentliche Komponente eines jeden Prozessors**
 - Daten werden mit der positiven Taktflanke übernommen, falls das Enable Signal e den Wert 1 hat.



Schaltbild

clk	e	Funktion
0/1/↓	–	Speichern
–	0	Speichern
↑	1	Übernehmen

Schaltverhalten



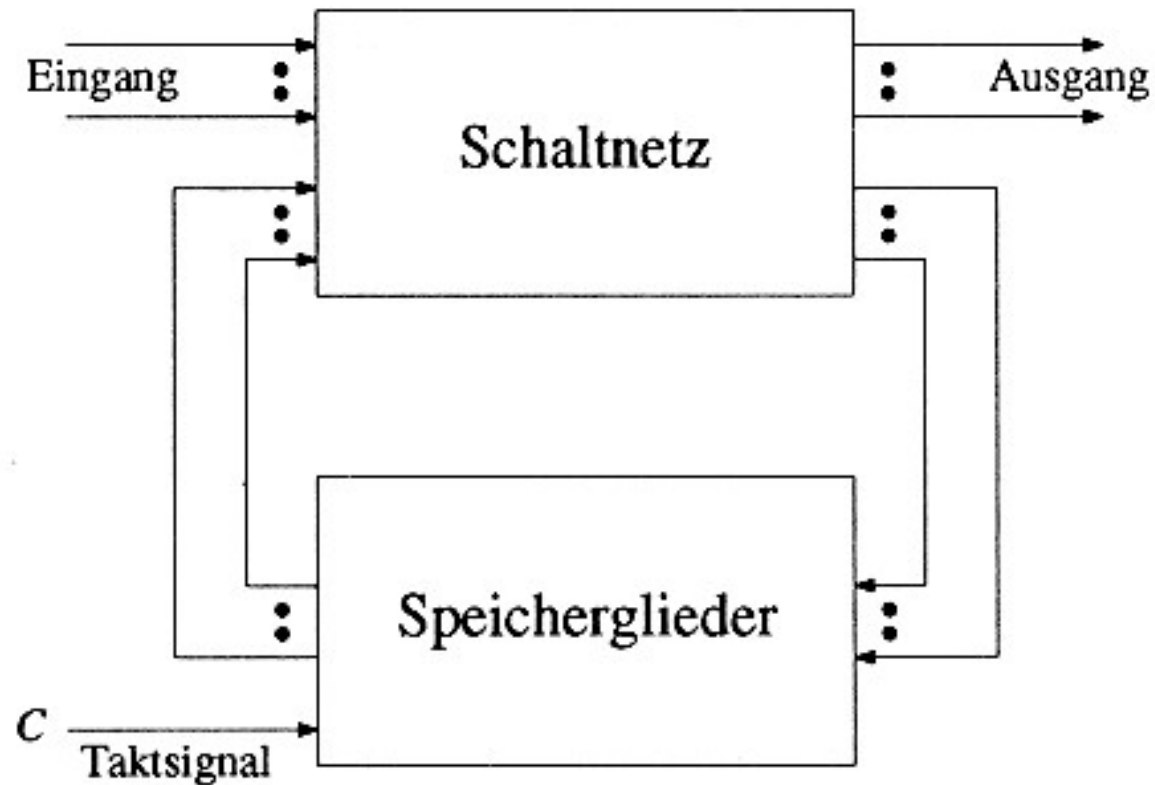
mögliche Implementierung

Schaltwerke

- technische Realisierung von endlichen Automaten (siehe: Theoretische Informatik)
- aufgebaut aus Schaltnetzen und Speichergliedern
 - Speicherglieder
 - speichern inneren Zustand des Schaltwerkes
 - Schaltnetze
 - berechnen den nächsten Zustand und die Ausgabe des Schaltwerkes

Schaltwerke (2)

Schematischer Aufbau

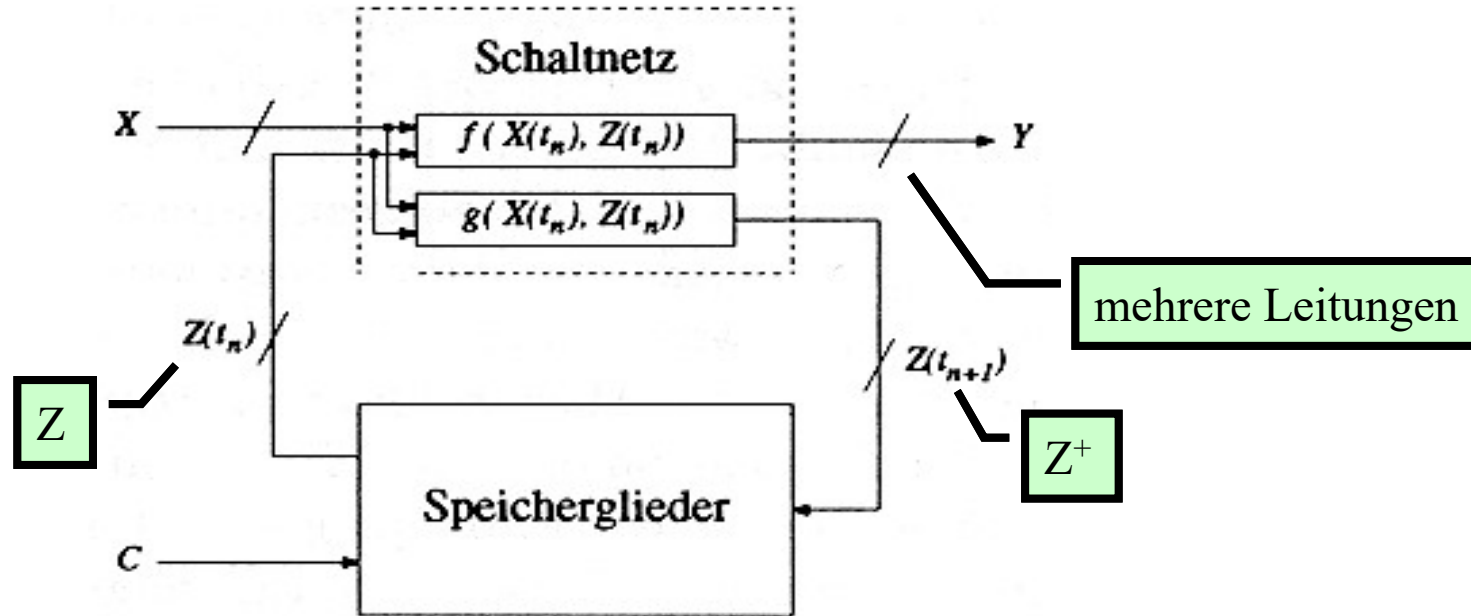


Mealy-Automat

- **Mealy-Automat**

- allgemeiner endlicher Automat
- die Werte an den Ausgängen hängen vom inneren Zustand **und** von den Werten an den Eingängen ab
- Bitvektoren
 - X : Eingabe des Automaten (l Bits)
 - Y : Ausgabe des Automaten (m Bits)
 - Z : Zustand des Automaten (k Bits)
- Z^+ : neuer Zustand (mit nächstem Takt gilt $Z := Z^+$)
 - oder: $Z(t_n) = Z$ t_n : Zeitpunkt der n^{ten} Taktflanke
 - und: $Z(t_{n+1}) = Z^+$
- $g(X, Z)$: Übergangsfunktion: $Z^+ = g(X, Z)$
- $f(X, Z)$: Ausgabefunktion: $Y = f(X, Z)$
- ferner muss ein Startzustand $Z(t_0)$ festgelegt werden

Mealy-Automat (2)



- bei jedem Takt wird der Wert der Übergangsfunktion als neuer Zustand gespeichert ($Z := Z^+$)
- die Taktperiode muss größer sein als die Summe aus
 - Zeit von Taktflanke bis stabile Ausgabe der Speicherglieder
 - Signallaufzeit durch das Schaltnetz g
 - Setup-Zeit der Speichergliederdamit Hazards abgeklungen sind und der Folgezustand stabil ist

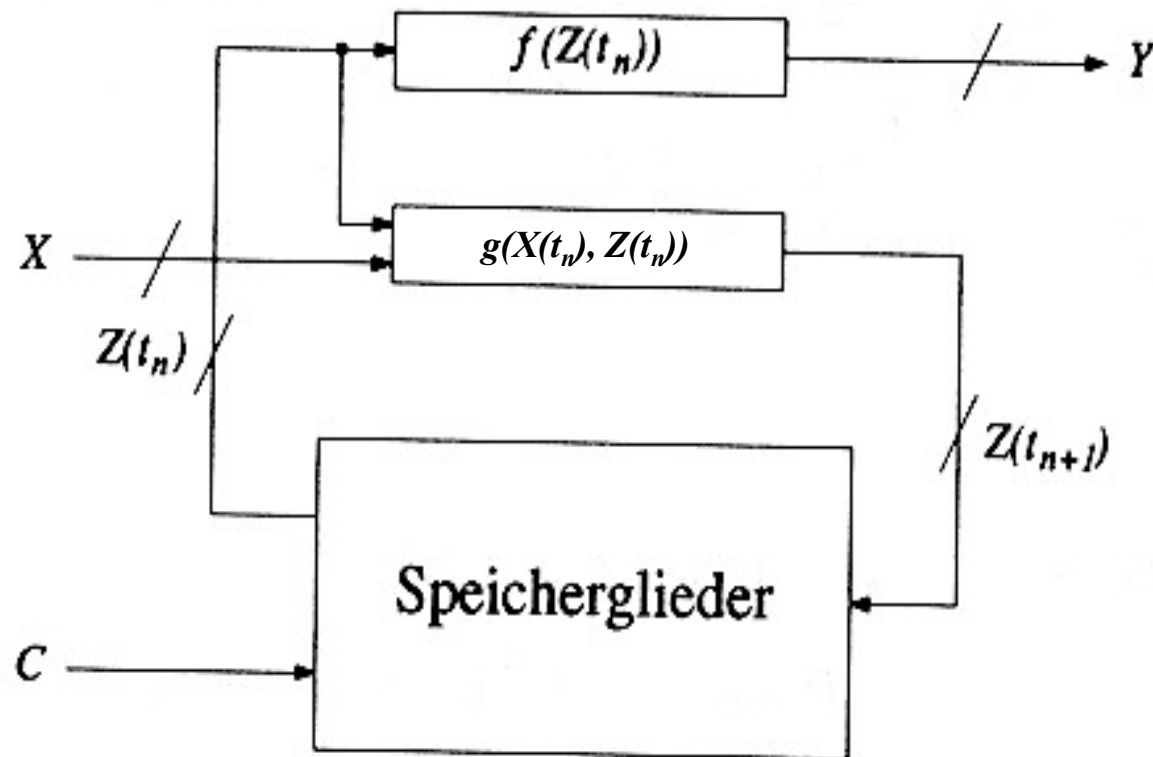
Moore-Automat

- **Moore-Automat**

- Sonderfall des Mealy-Automaten
- die Werte an den Ausgängen hängen **nur** vom inneren Zustand ab

$$Z^+ = g(X, Z)$$

$$Y = f(Z)$$



Beschreibung endlicher Automat

- **endlicher Automat**
 - im wesentlichen durch die beiden Schaltfunktionen f und g charakterisiert
 - zur funktionellen Beschreibung können alle Methoden dienen, die für Schaltfunktionen bekannt sind (Wertetabelle, KV-Diagramme, Boolesche Formeln)
 - kombiniert man beide Schaltfunktionen zu einer gemeinsamen Wertetabelle, erhält man die Zustandsfolgetabelle

Zustandsfolgetabelle

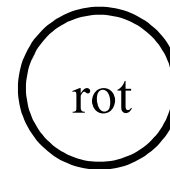
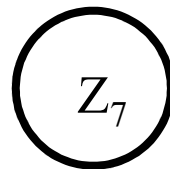
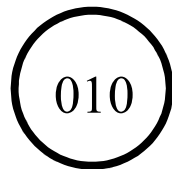
- Zustandsfolgetabelle

$z_1 \dots z_k$	$x_1 \dots x_l$	$z^+_1 \dots z^+_k$	$y_1 \dots y_m$

- hieraus können wie üblich die Schaltfunktionen z.B. mit KV-Diagrammen entwickelt werden
- übersichtlicher ist aber das Zustandsdiagramm

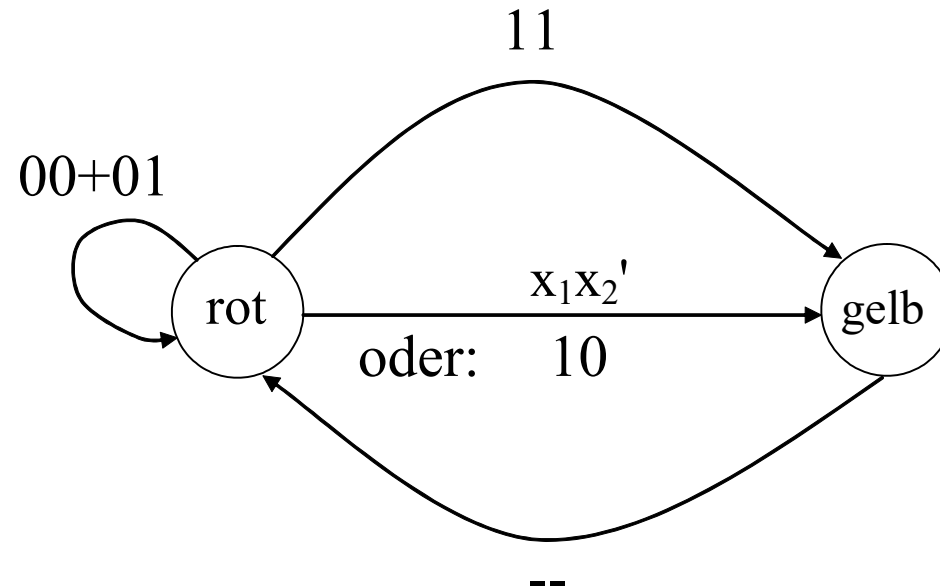
Zustandsdiagramm

- **Zustandsdiagramm**
 - grafische Darstellung eines endlichen Automaten
 - gerichteter Graph
 - Knoten
 - Zustände
 - Darstellung als Kreis mit Werten der Zustandsvariablen oder besser zunächst mit symbolischem Namen



Zustandsdiagramm (2)

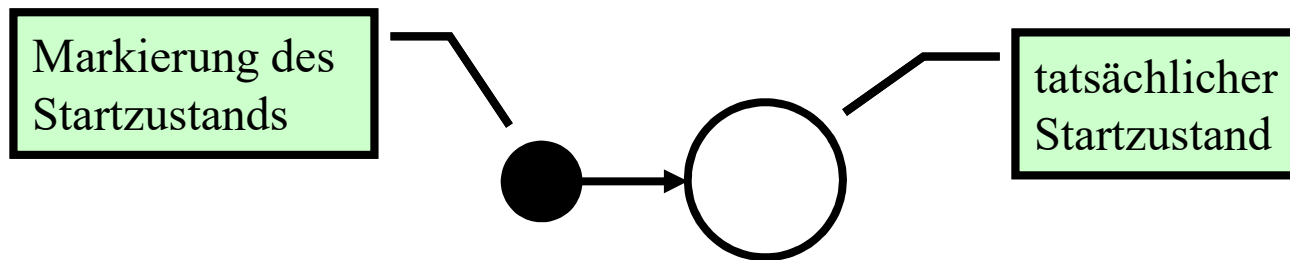
- gerichtete Kanten
 - Übergänge zwischen Zuständen
 - beschriftet mit den Werte der Eingangsvariablen, die den Zustand in den nächsten überführt
 - eine auf den eigenen Knoten zurückführende Kante gibt an, dass der Zustand nicht verändert wird
 - Mehrfachkanten sind erlaubt (ODER-Verknüpfung)



Zustandsdiagramm (3)

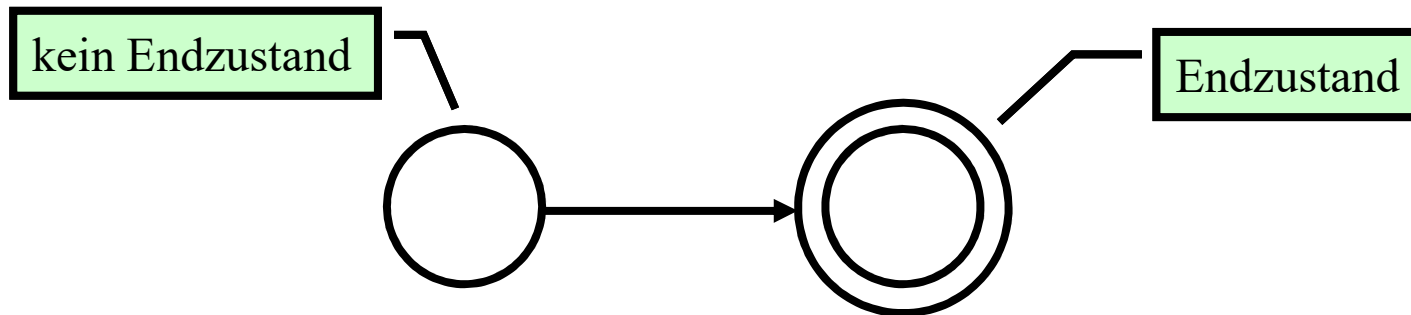
- **Startzustand**

- es gibt genau einen Startzustand



- **Endzustand**

- es kann mehrere geben, muss aber nicht existieren
- in der technischen Informatik kommt ein Endzustand eher selten vor

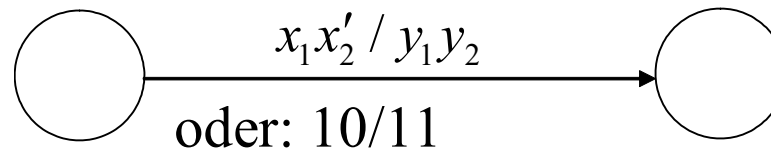


Zustandsdiagramm (4)

- **Spezifikation der Ausgabevariablen**

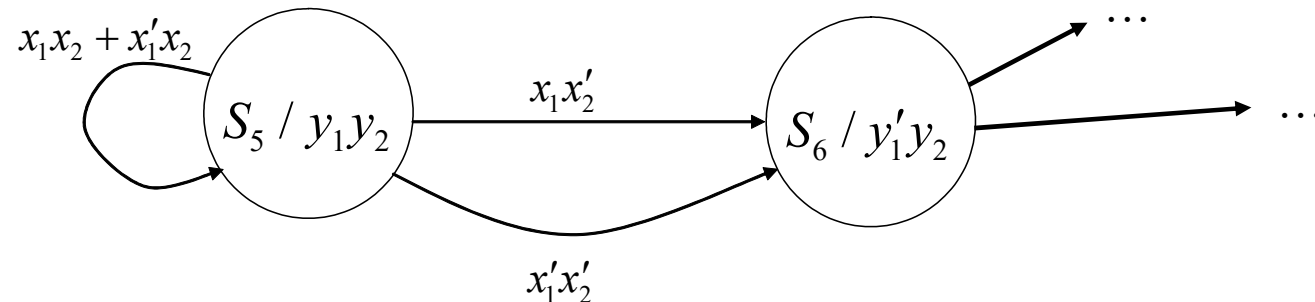
- Mealy-Automat

- jede Kante wird zusätzlich mit den Werten der Ausgabevariablen beschriftet



- Moore-Automat

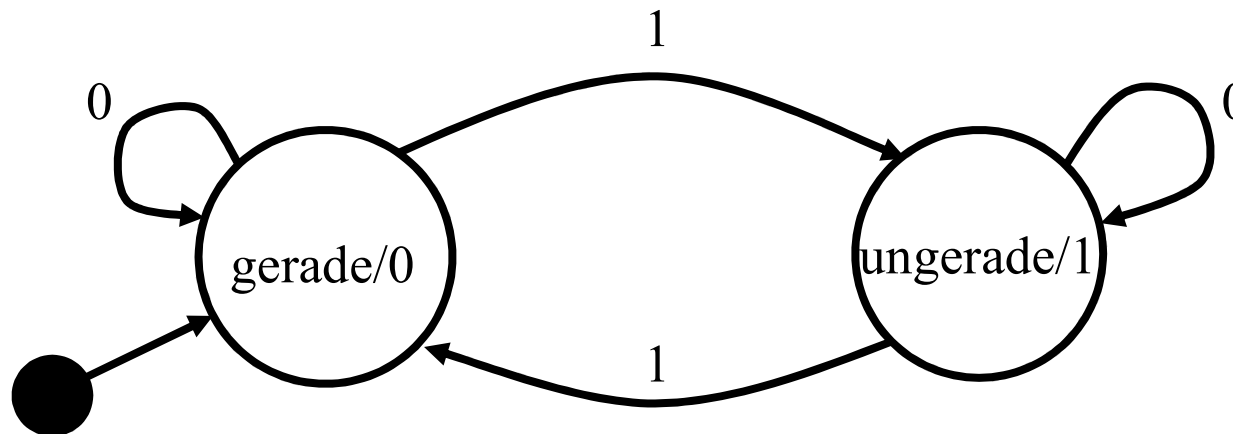
- da der Wert der Ausgangsvariablen nur vom Zustand abhängt, schreibt man die Wertekombination in den Knoten



Beispiel: Parität

- **Beispiel**

- Parität (gerade/ungerade) für eine beliebige Anzahl von Bits
- Eingabe: bei jedem Takt das nächste Bit
- zwei Zustände: gerade/ungerade
- Ausgabe: 0 für gerade, 1 für ungerade Parität



Beispiel: Parität (2)

- Kodierung der Zustände
 - zwei Zustände benötigen nur ein Bit
 - hier ist es sinnvoll, die Zustände so zu kodieren, dass die Ausgabefunktion trivial wird

gerade = 0

ungerade = 1

- Zustandsübergangstabelle

z	x	z^+	y
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

- Übergangsfunktion

$$z^+ = g(z, x) = z \oplus x$$

- Ausgabefunktion (Moore!)

$$y = f(z) = z$$

Beispiel: Parität (3)

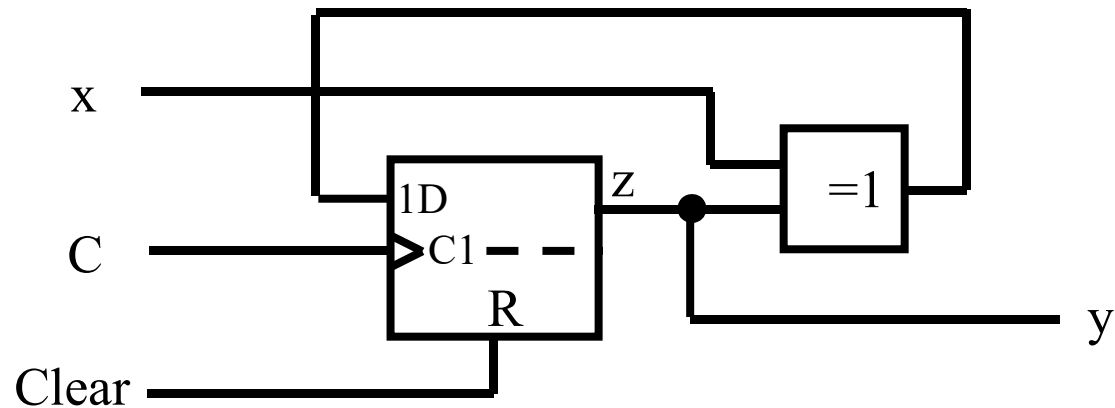
- Übergangsfunktion

$$z^+ = g(z, x) = z \oplus x$$

- Ausgabefunktion (Moore!)

$$y = f(z) = z$$

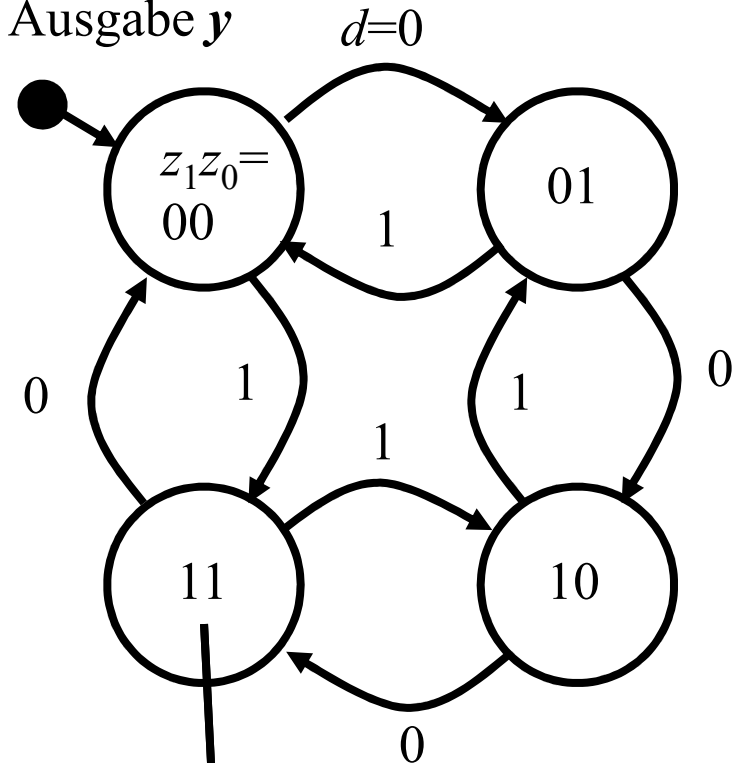
- **Schaltwerk**



Beispiel: Aufwärts-/Abwärtszähler

- **Synchroner Zähler**

- innerer Zustand z ist gleichzeitig die Ausgabe y
 - keine Angabe der Ausgabewerte
- Eingabe: Zählrichtung d
 - aufwärts: $d = 0$
 - abwärts: $d = 1$



- **Siehe Tafel für die Details!**

- Resultat:

$$z_0^+ = \bar{z}_0$$

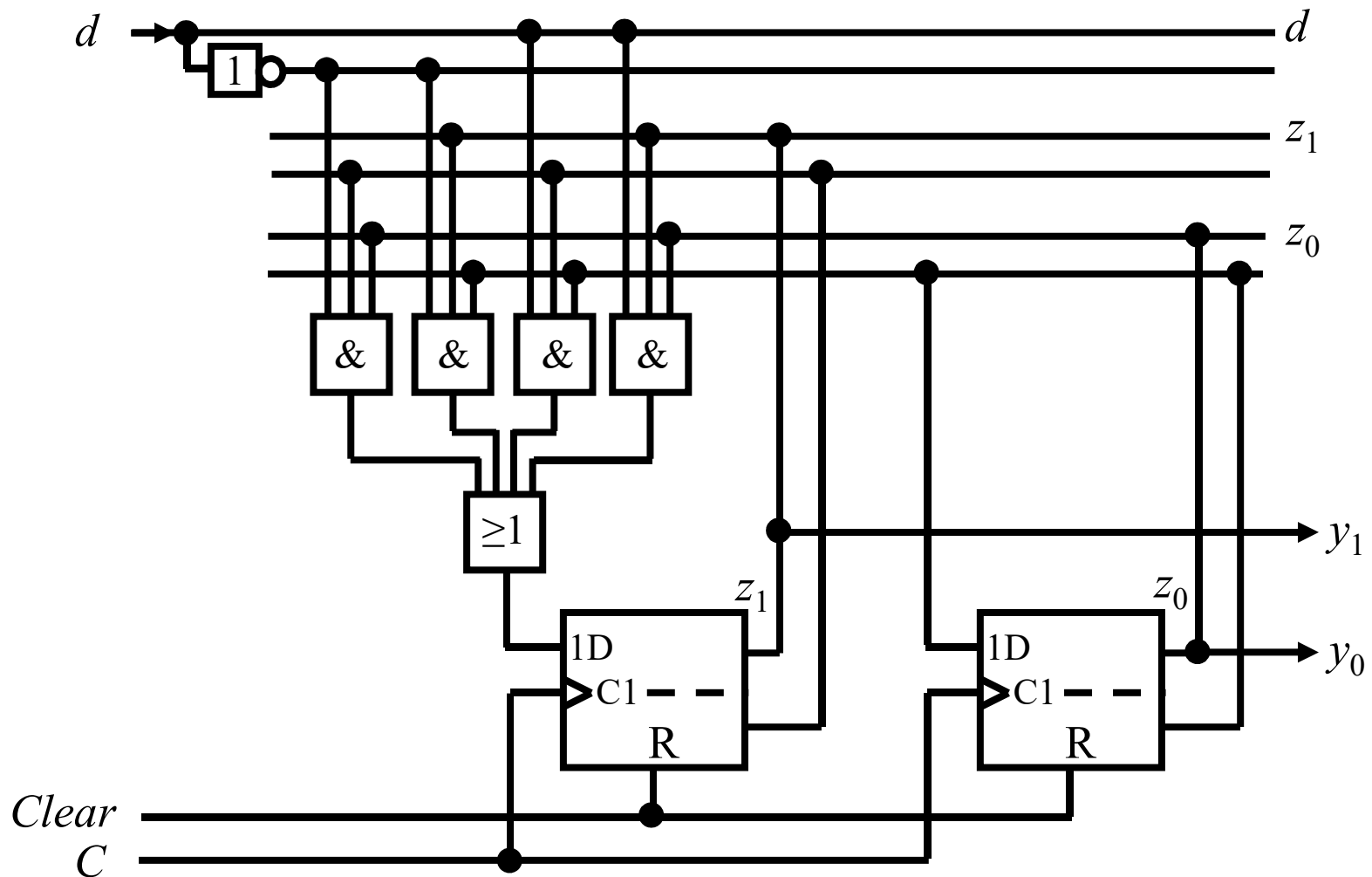
$$z_1^+ = \bar{d}\bar{z}_1z_0 + \bar{d}z_1\bar{z}_0 + d\bar{z}_1\bar{z}_0 + dz_1z_0$$

Zustand und Ausgabe
sind hier identisch

Beispiel: Aufwärts-/Abwärtszähler (2)

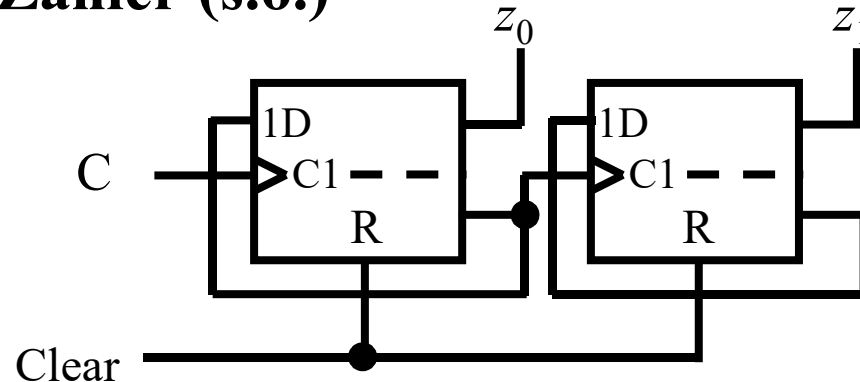
- Schaltplan

$$z_0^+ = \bar{z}_0 \quad z_1^+ = \bar{d}\bar{z}_1z_0 + \bar{d}z_1\bar{z}_0 + d\bar{z}_1\bar{z}_0 + dz_1z_0$$



Beispiel: Aufwärts-/Abwärtszähler (3)

- **Asynchroner Zähler (s.o.)**



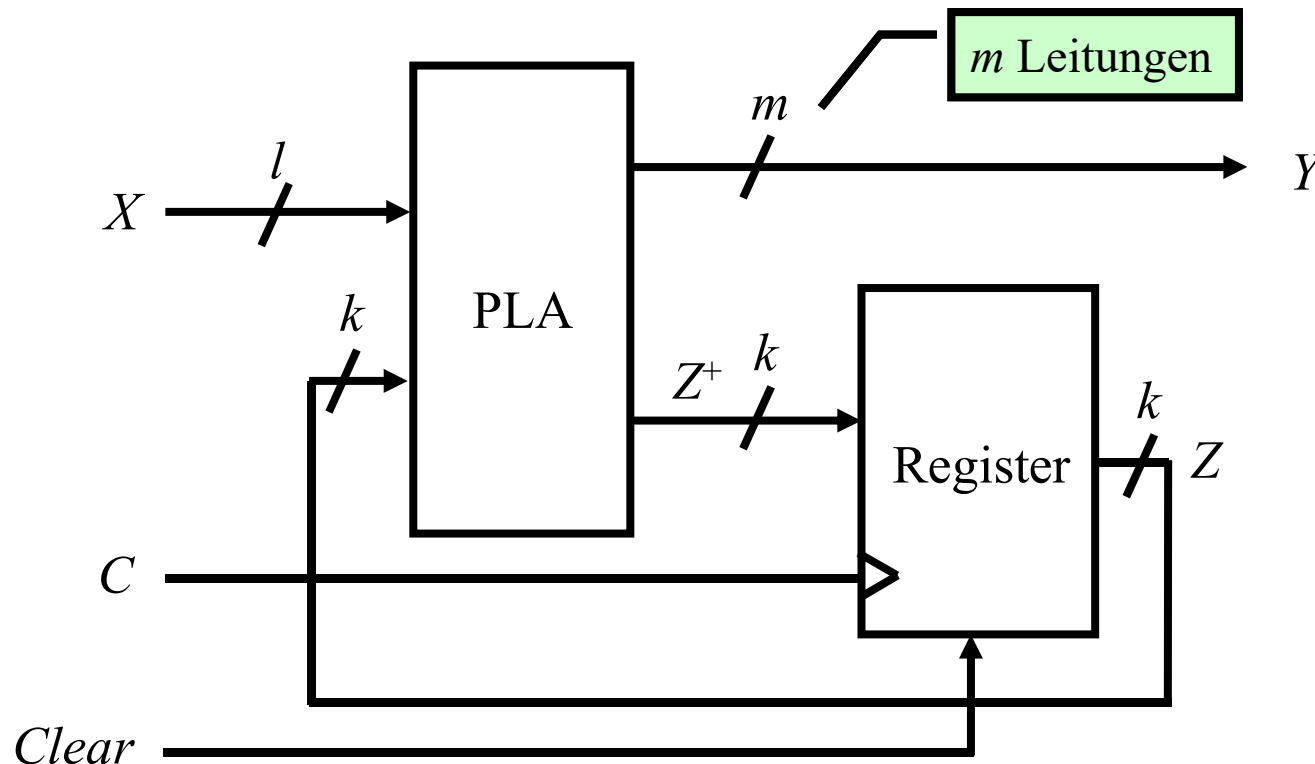
- **Unterschiede zw. synchronem und asynchronem Zähler**
 - synchron
 - höherer HW-Aufwand
 - alle Flip-Flops bekommen dasselbe Taktsignal, schalten also gleichzeitig
 - asynchron
 - niedriger HW-Aufwand
 - Flip-Flops schalten nacheinander
 - kann bei großen Bitbreiten zu Problemen führen (Verzögerungszeit $O(n)$ wie beim Ripple Carry Addierer)

Autonome Schaltwerke

- **Schaltnetze**
 - starre Abbildung von Eingang auf Ausgang
- **Schaltwerke**
 - haben eine Eigendynamik, d.h. sie können zeitlich beliebig verlaufende Ausgangssignale erzeugen
 - Ausgangssignale können zusätzlich von Eingangssignalen abhängen
- **autonome Schaltwerke**
 - haben keine Eingänge
 - produzieren festgelegte Ausgangssignalverläufe
 - Beispiel: verkehrsunabhängige Ampelsteuerung

Realisierung von Schaltwerken

- **PLA + Register**
 - PLA's haben häufig die Register schon eingebaut



Standardschaltwerke

- **im Grunde kann man jede Schaltung einer von zwei Klassen zuordnen**
 - Schaltnetze
 - direkte Umsetzung von Eingaben zu Ausgaben
 - kein Speicher
 - kein Taktsignal
 - nur kombinatorische Logik
 - Schaltwerke
 - besitzen einen Zustand (also Speicherelemente)
 - haben Taktsignal
 - Eingaben und Zustand bestimmen zusammen die Ausgaben
- **in diesem Sinne ist ein Flip-Flop schon ein Schaltwerk**
 - hier fehlt nur das Schaltnetz zur Berechnung der Ausgabe (Schaltfunktion = Identität)
 - nächster Zustand hängt je nach Flip-Flop-Typ von den Eingaben ab

Standardschaltwerke (2)

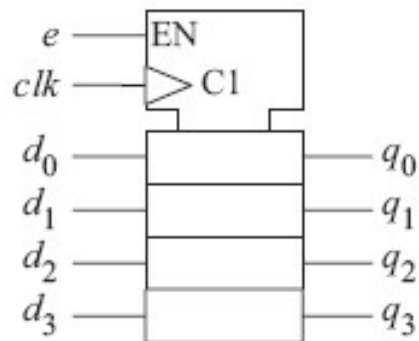
- **weitere wichtige Standardschaltwerke sind**
 - Register
 - Akkumulator
 - Programmzähler
 - Zähler
 - Schieberegister
 - Hauptspeicher

Register

- parallele Anordnung von n Speicherelementen
- synchrone Ansteuerung über eine gemeinsame Taktleitung
- dienen der Speicherung und Manipulation von ganzen Datenworten
- verschiedene Typen

Auffangregister (hatten wir schon!)

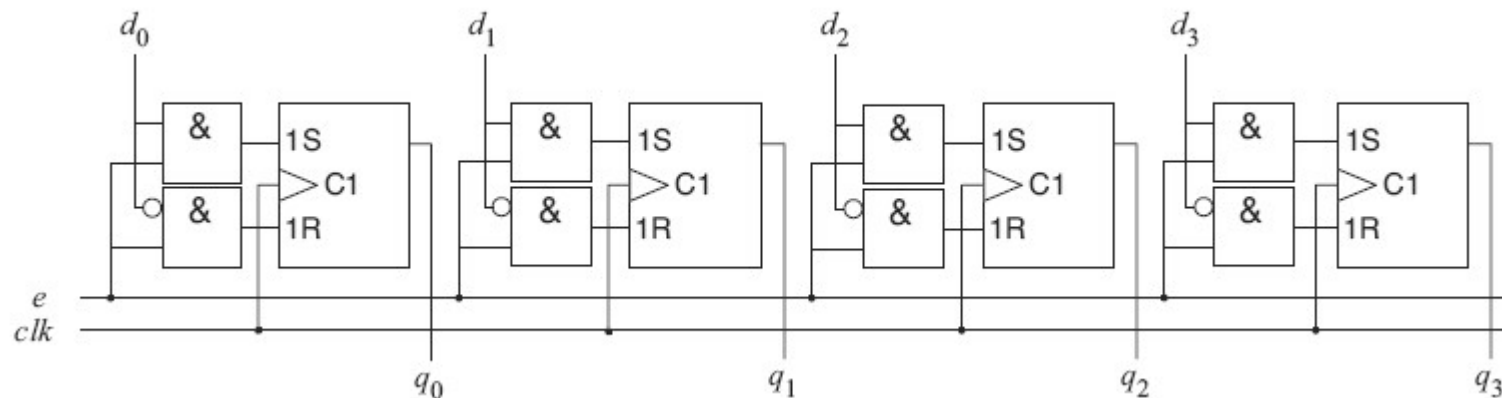
- **wesentliche Komponente eines jeden Prozessors**
 - Daten werden mit der positiven Taktflanke übernommen, falls das Enable Signal e den Wert 1 hat.



Schaltbild

clk	e	Funktion
0/1/↓	–	Speichern
–	0	Speichern
↑	1	Übernehmen

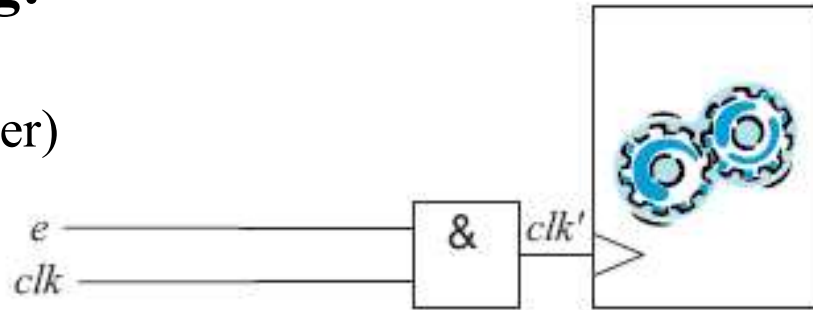
Schaltverhalten



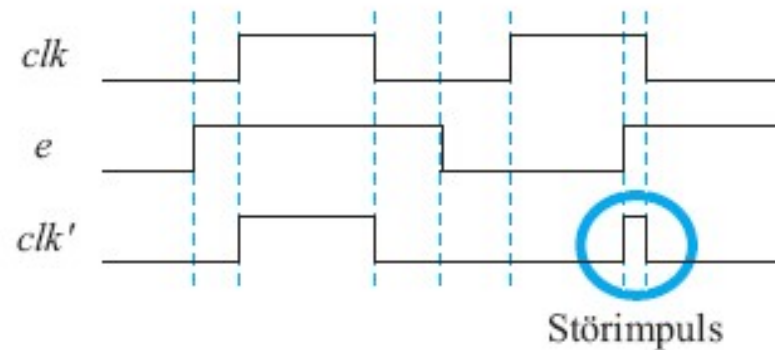
mögliche Implementierung

Auffangregister (2)

- **Enable-Signal verursacht erheblichen Aufwand**
 - zwei UND-Gatter für jedes Bit
- **scheinbar einfachere Lösung:**
 - Maskieren des Taktsignales
(also Abschalten mit UND-Gatter)

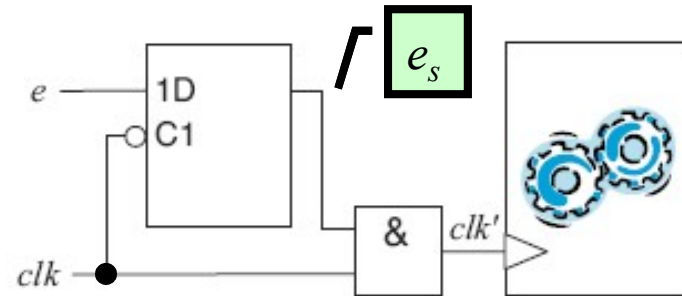


- Risiko durch Hazards
 - während der positiven Taktphase muss das Enable-Signal stabil bleiben
 - Niemals Schaltnetze in den Pfad des Taktsignals legen!

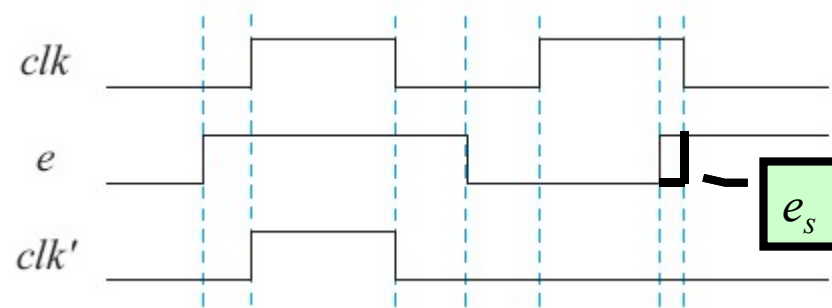


Auffangregister (3)

- **Trick zur Vermeidung des Problems**
 - Vorschalten eines invers angesteuerten D-Latches

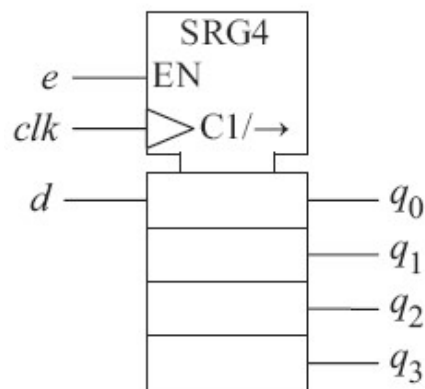


- während der positiven Taktphase wird das Enable-Signal festgehalten
- Änderungen kann es nur in der unkritischen negativen Taktphase geben



Schieberegister

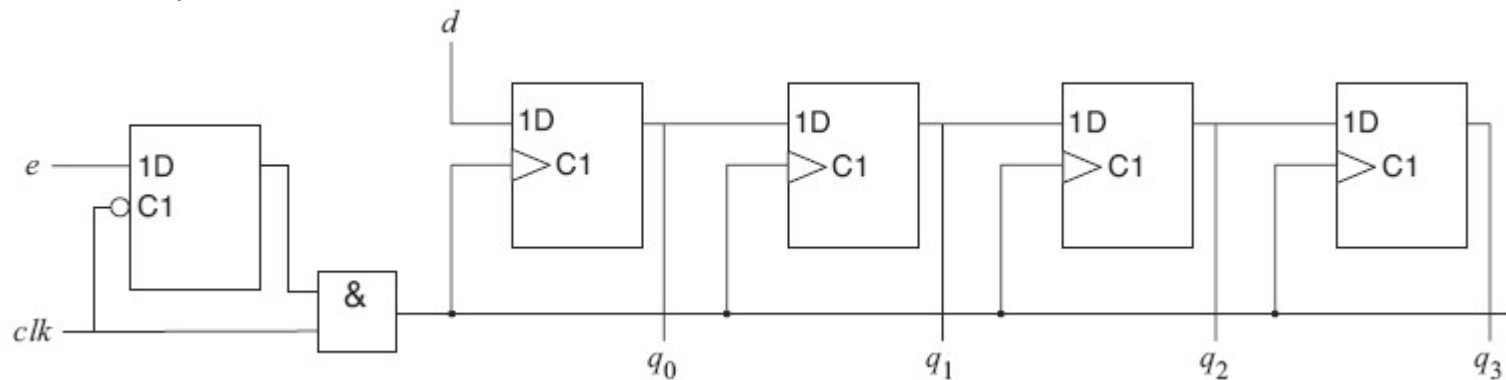
- Daten werden nicht parallel übernommen sondern seriell
- nur ein Eingang, aber n Ausgänge



clk	e	Funktion
0/1/↓	–	Speichern
–	0	Speichern
↑	1	Rechts schieben

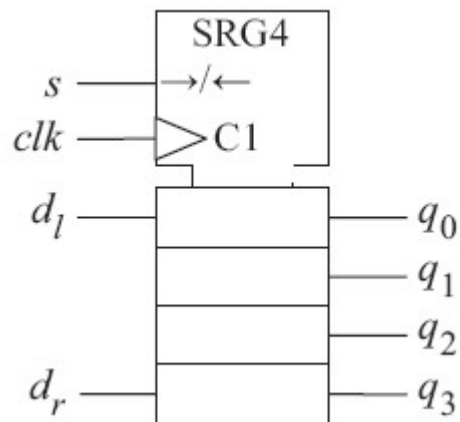
Schaltverhalten

Schaltsymbol

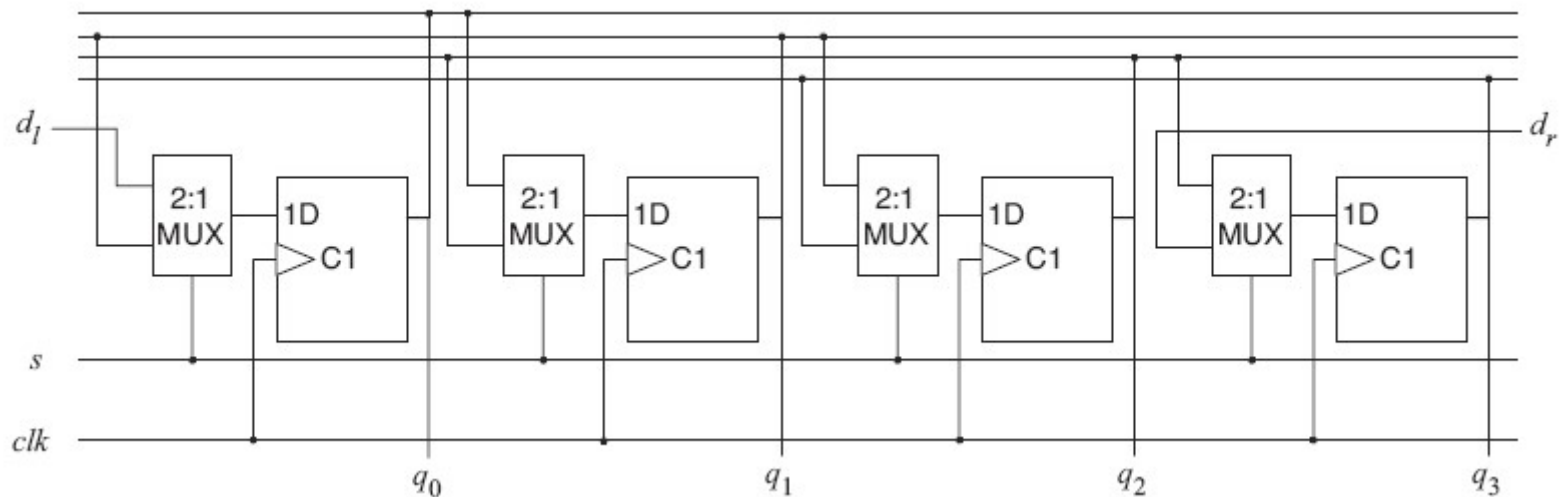


Bidirektionales Schieberegister

- kann in beide Richtungen schieben

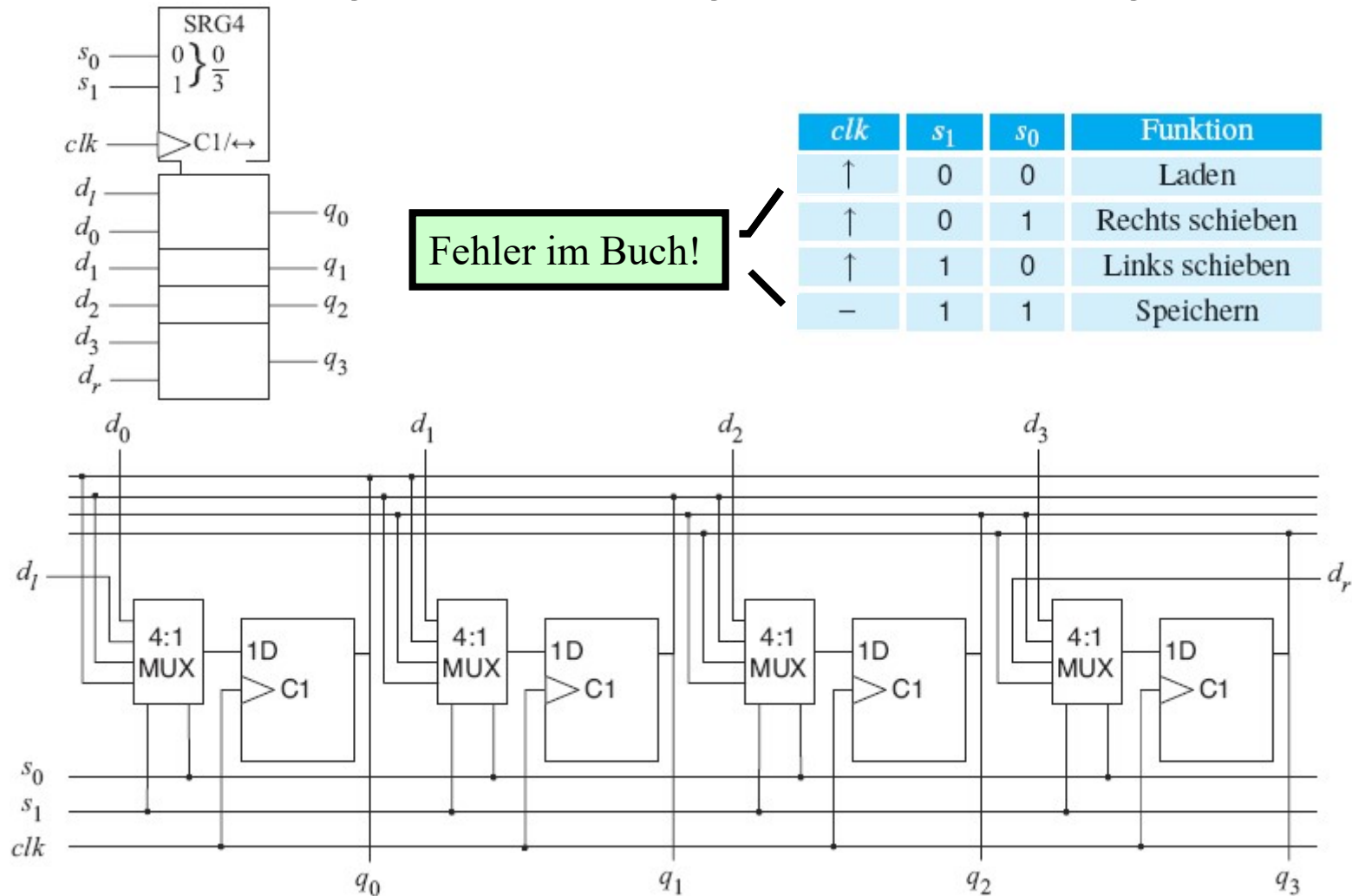


clk	s	Funktion
0/1/↓	–	Speichern
↑	0	Rechts schieben
↑	1	Links schieben



Universalregister

- erfüllt die Aufgaben eines Auffang- und eines Schieberegisters



Universalregister (2)

- **erfüllt mehrere Aufgaben**
 - Speichern
 - Daten werden parallel geschrieben und gelesen
 - Schieben
 - Multiplikation mit 2 bzw. Division durch 2
 - seriell/parallel-Wandlung
 - Daten werden bitweise seriell in das Register geschoben und parallel ausgelesen
 - parallel/seriell-Wandlung
 - Daten werden parallel geladen und seriell durch Schieben ausgelesen

- _____



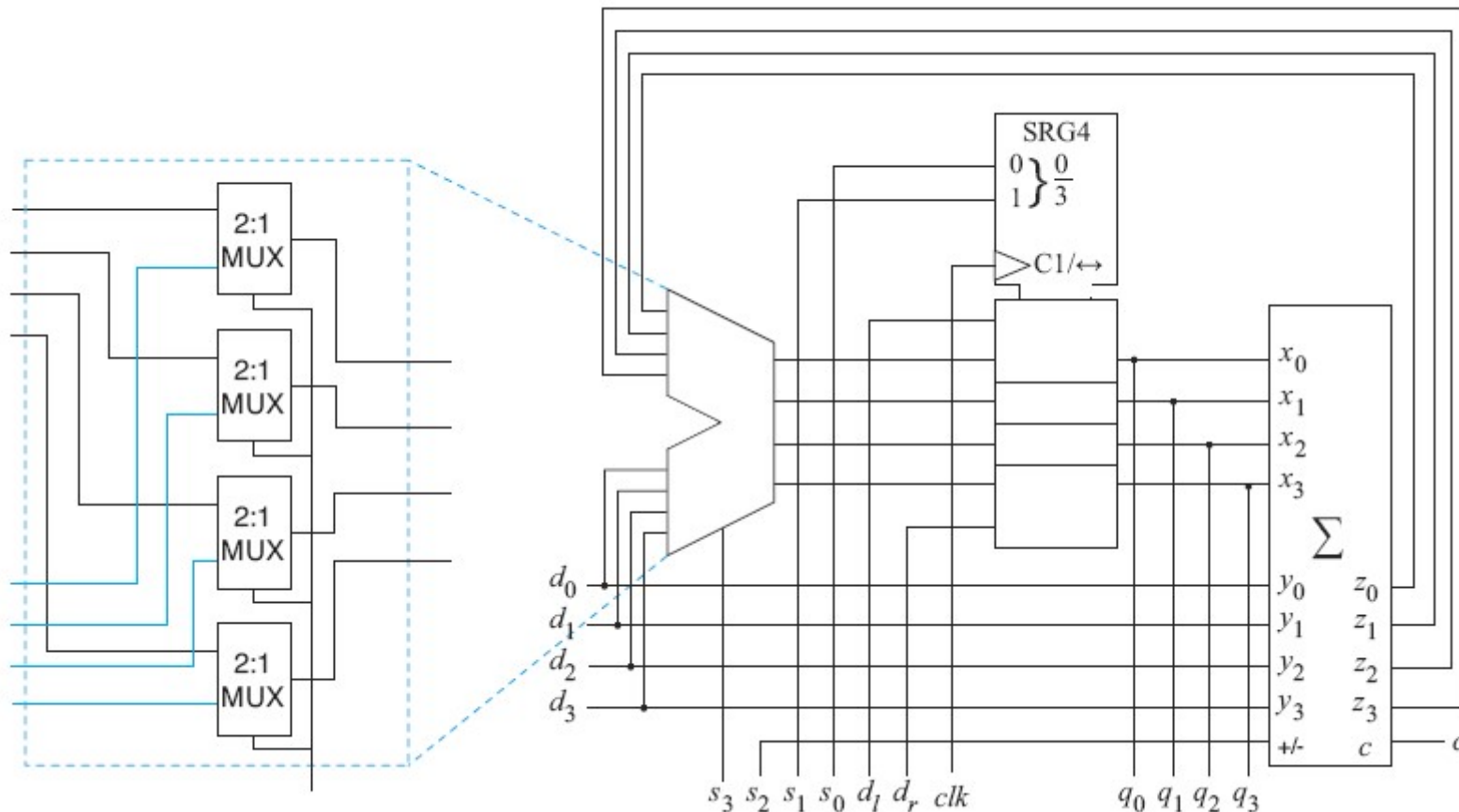
Akkumulator (2)

- **Register, Steuerleitung e**
 - Speichern oder Laden
- **ALU, Steuerleitung s_0**
 - ALU kann addieren oder subtrahieren
 - ein Operand ist das Register (der eigentliche Akkumulator)
 - der andere Operand ist ein externes Datenwort
- **Multiplexer, Steuerleitung s_1**
 - Register kann mit neuen, externen Werten geladen werden oder das Ergebnis der ALU übernehmen
 - im zweiten Fall wird "akkumuliert", d.h. externe Werte werden zum Inhalt des Registers hinzuaddiert (oder subtrahiert)

e	s_1	s_0	Funktion
0	0	0	Speichern
0	0	1	Speichern
0	1	0	Speichern
0	1	1	Speichern
1	0	0	Addieren
1	0	1	Subtrahieren
1	1	0	Laden
1	1	1	Laden

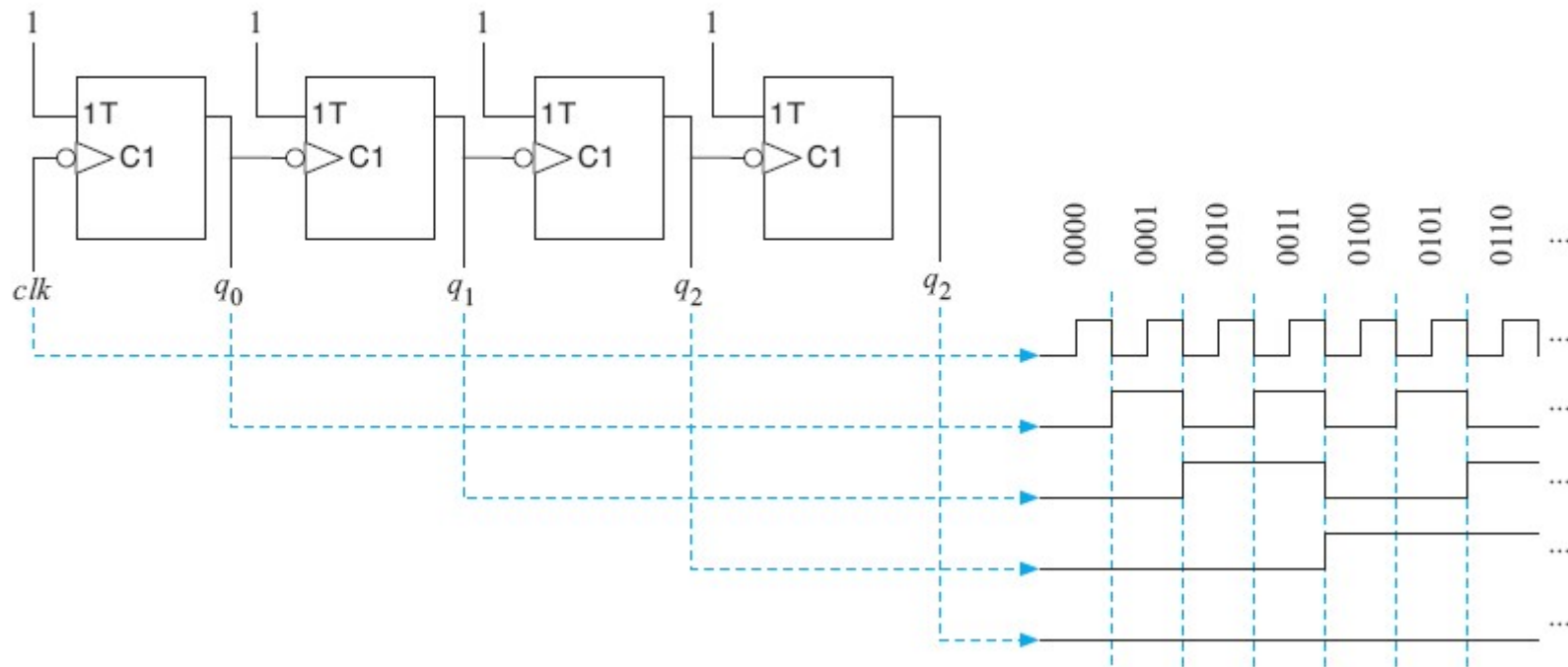
Akkumulator (3)

- **Erweiterter Akkumulator**
 - Verwendung eines Universalregisters
 - zusätzlich könnte die ALU weitere Operationen implementieren



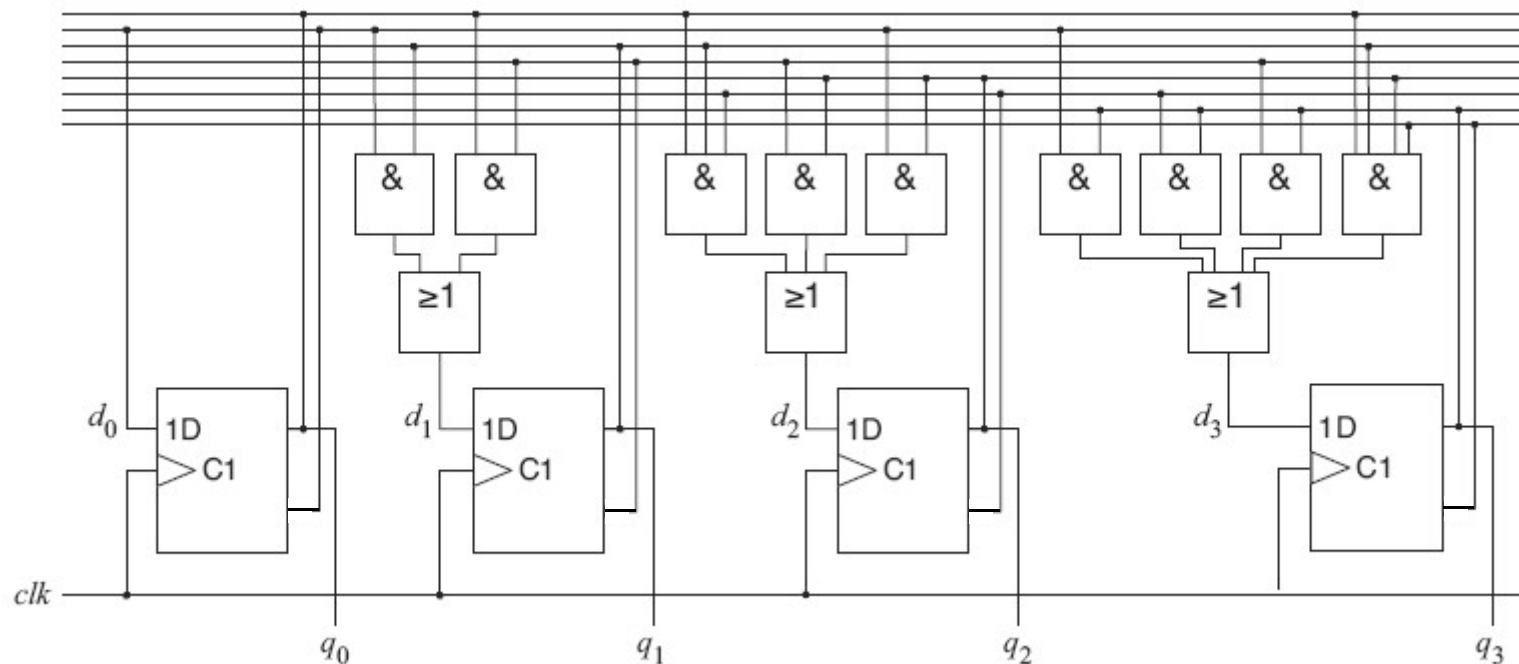
Asynchrone Zähler

- keine strikte Trennung zwischen Daten- und Taktleitungen
 - potentiell gefährlich, begünstigt das Entstehen von Hazards
 - Schaltungstiefe (Verzögerungszeit): $O(n)$
 - maximale Taktfrequenz wird mit wachsendem n immer kleiner
- preiswerte Form
 - Flächenbedarf (Anzahl der Gatter): $O(n)$



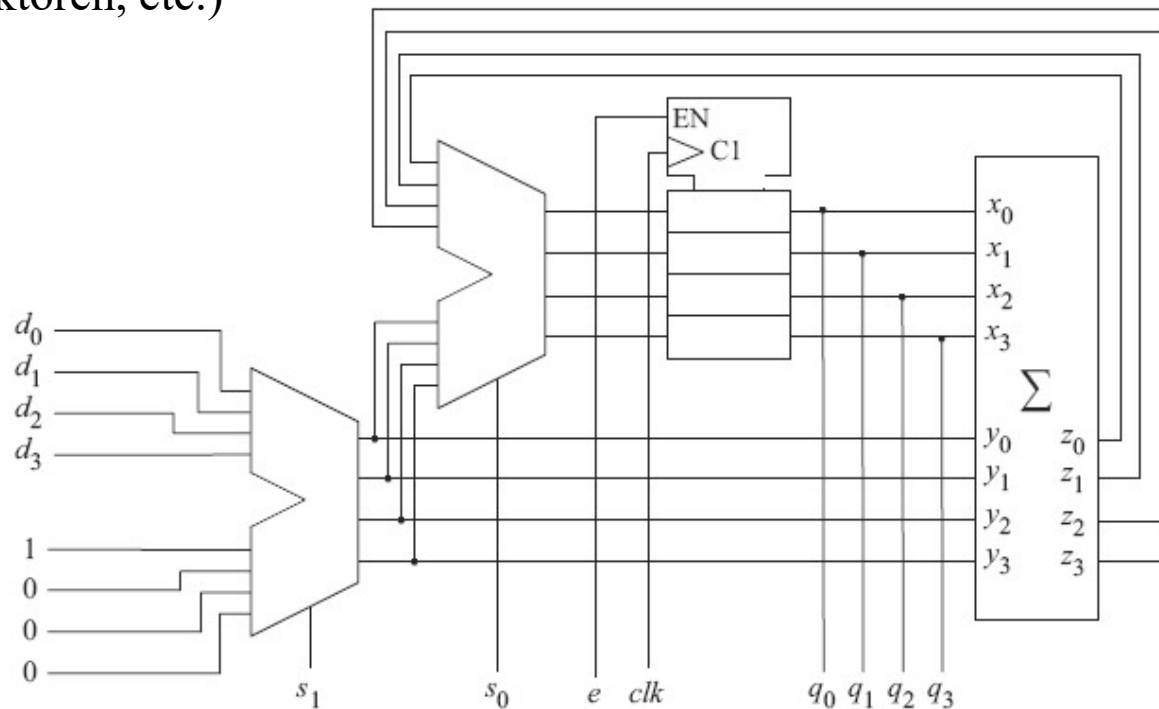
Synchrone Zähler

- alle Speicherglieder werden von demselben Taktsignal gesteuert
- disjunktive Minimalform zur Berechnung des nächsten Zählerstandes
 - Schaltungstiefe (Verzögerungszeit): $O(1)$
 - maximale Taktfrequenz ist unabhängig von n
- aufwendigere Form
 - Flächenbedarf (Anzahl der Gatter): $O(n^2)$



Instruktionszähler

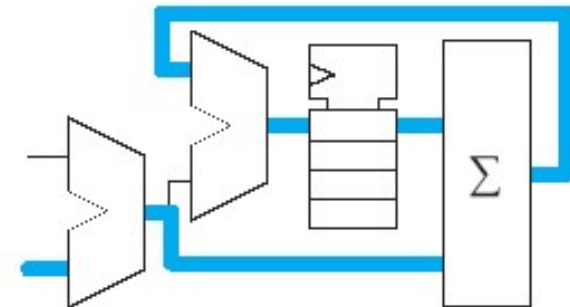
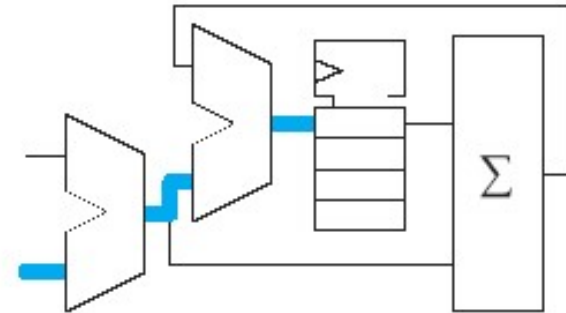
- auch Programmzähler (PC: *Program Counter*) genannt
- weitere Schlüsselkomponente eines jeden Prozessors
- dient der Adressierung des Hauptspeichers
 - enthält meist die Adresse des nächsten zu lesenden Befehls
 - kann auch zur Adressierung von Daten dienen (Abarbeitung von Arrays, Vektoren, etc.)



Instruktionszähler (2)

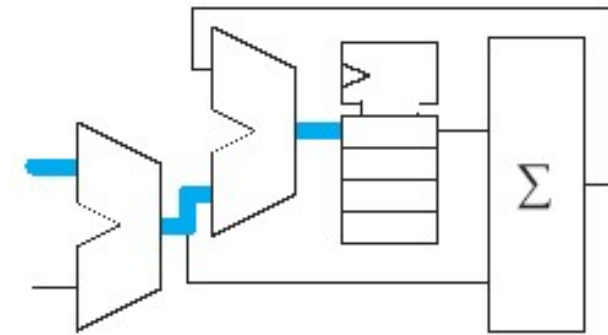
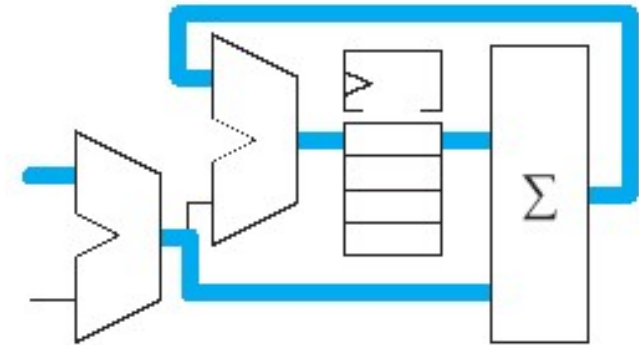
- **Anforderungen an Instruktionszähler**

- Reset ($s_1 = 1, s_0 = 1$)
 - definierter Startzustand
 - hier wird Startadresse der Einfachheit halber auf 1 gesetzt
 - kann natürlich mit größeren Multiplexern auch auf 0 gesetzt werden
 - oder man benutzt den absoluten Sprung (s.u.)
- Schrittzählung ($s_1 = 1, s_0 = 0$)
 - Befehle und Daten sind praktisch immer sequentiell im Hauptspeicher abgelegt (an aufeinander folgenden Adressen)
 - um nacheinander darauf zugreifen zu können, muss der Zählerinhalt inkrementiert (um 1 erhöht) werden können
 - am häufigsten verwendete Betriebsart



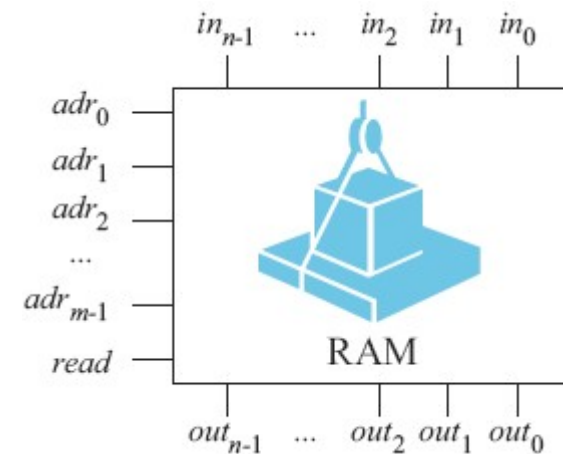
Instruktionszähler (3)

- Relative Sprünge ($s_1 = 0, s_0 = 0$)
 - alle modernen Prozessoren verwenden relative Sprungbefehle, die es erlauben, um eine feste Zahl von Adressen nach vorne oder nach hinten zu springen
 - damit können Schleifen programmiert werden oder eine feste Zahl von Befehlen übersprungen werden (IF-Zweige)
- Absolute Sprünge ($s_1 = 0, s_0 = 1$)
 - Instruktionszähler wird auf eine von außen vorgegebene Adresse gesetzt
 - wird z.B. benutzt, um in ein Unterprogramm (dessen Startadresse bekannt ist) zu springen



Hauptspeicher

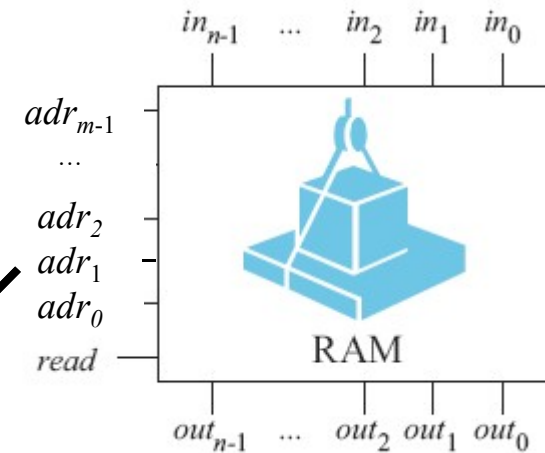
- Hauptkomponente eines jeden Computers
- dient dem Zweck, viele Datenworte zu speichern
 - Ansammlung von Registern zu aufwendig,
 - die Anzahl der Anschlüsse wird zu groß (linear in Anzahl der Worte)
- besser
 - Angabe der Adresse der gesuchten Speicherstelle
 - Anzahl Anschlüsse wächst nur logarithmisch mit der Anzahl der gespeicherten Worte (da für n Adressen nur $\lg n$ Bits benötigt werden)



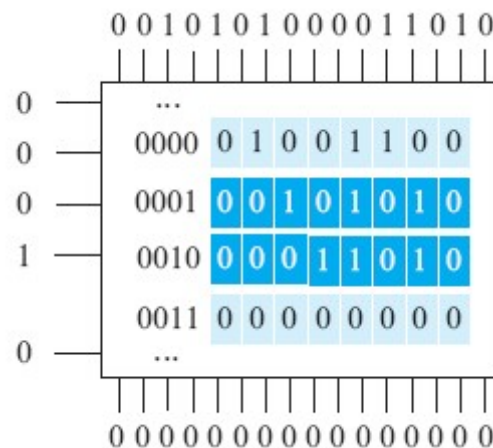
Hauptspeicher (2)

- obwohl ganze Worte gespeichert werden, bieten fast alle Speichersysteme die Möglichkeit, auch einzelne Bytes zu adressieren
- Adressen sind dann Byte-Adressen

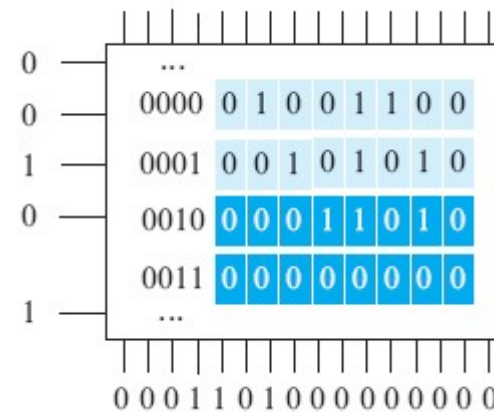
Fehler im Buch



allgemeines Schema



schreibender Zugriff



lesender Zugriff

Speicherarten

- **RAM**
 - **R**andom **A**ccess **M**emory
 - Name falsch gewählt
 - heißt eigentlich: Speicher mit wahlfreiem Zugriff (Zugriff in beliebiger Reihenfolge, z.B. im Gegensatz zu einem Magnetband)
 - ist aber ein: Schreib-/Lesespeicher (mit wahlfreiem Zugriff)
 - zwei Arten
 - statische RAMs (SRAM)
 - zusammengesetzt aus D-Flip-Flop ähnlichen Schaltungen
 - können Inhalte so lange halten, wie Spannungsversorgung vorhanden ist
 - arbeiten sehr schnell, Zugriffszeit wenige Nanosekunden
 - relativ teuer, da eine Speicherzelle aus 4 bis 6 Transistoren besteht
 - werden z.B. als schnelle Zwischenspeicher (Caches) verwendet (s.u.)

Speicherarten (2)

- dynamische RAMs (DRAM)
 - Information wird als Ladung in einem Kondensator abgelegt (Speicherzelle besteht aus einem einzigen MOSFET, daher preiswert)
 - Ladung bleibt nur kurze Zeit erhalten
 - Speicherzelle muss alle paar Millisekunden aufgefrischt werden
 - erhöhter Aufwand zum Auslesen und Auffrischen des Speichers
 - langsamer aber viel billiger als SRAM
 - Einsatz als Hauptspeicher

Speicherarten (3)

- **ROM**

- **Read Only Memory**
- nur-Lese-Speicher
 - Speicherinhalt wird schon bei der Herstellung festgelegt
 - in großen Stückzahlen billiger als RAM
- Nicht-flüchtiger Speicher (unabhängig von Stromversorgung)

- **PROM**

- **Programmable ROM**
- einmal programmierbar
 - Durchbrennen von Sicherungen (siehe auch: PLAs)
- zum Prototyping von ROMs geeignet

Speicherarten (4)

- **EPROM**

- Erasable **PROM**
- Inhalt kann durch Bestrahlung mit ultravioletten Licht gelöscht werden (dauert ca. 15 Minuten)
- danach wieder elektrisch programmierbar
- bei häufigen Änderungen wirtschaftlicher als PROM

- **EEPROM**

- Electrically Erasable **PROM**
- Nicht-flüchtiger Speicher
- gesamter Inhalt kann elektrisch wieder gelöscht werden
- langsamer als EPROM
- viel langsamer und teurer als DRAM oder SRAM

Speicherarten (5)

- **Flash Memory**

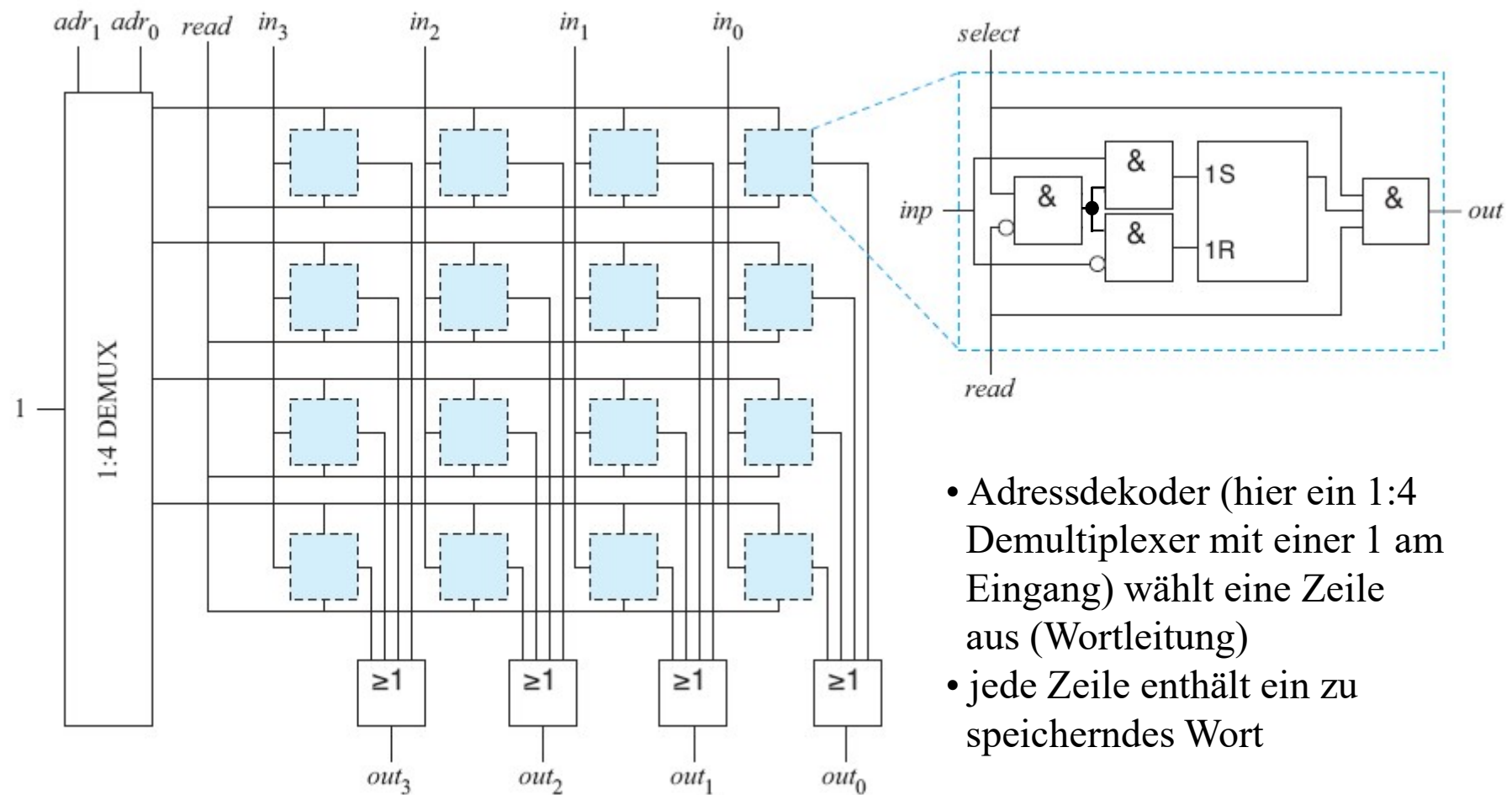
- neuere Art von EEPROM
- wird blockweise elektrisch gelöscht (nicht gesamter Speicher)
- hohe Speicherkapazität
- Speicher für digitale Kameras, Memory-Sticks, Handys, MP3-Player, etc.
- beginnen bereits Festplatten zu ersetzen
 - SSD: Solid State Drive ("Festkörperlaufwerk", d.h. ein Laufwerk ohne bewegliche Teile, nur Halbleiterspeicher)

- **Weitere Formen (zukünftige main stream Technologien?)**

- MRAM: magneto-resistive RAM
 - P-RAM: phase change memory
 - F-RAM: ferroelectric RAM
- } alles nicht-flüchtige Speicher

SRAM

- Schematischer Aufbau



- Adressdekoder (hier ein 1:4 Demultiplexer mit einer 1 am Eingang) wählt eine Zeile aus (Wortleitung)
- jede Zeile enthält ein zu speicherndes Wort

Speicherorganisation

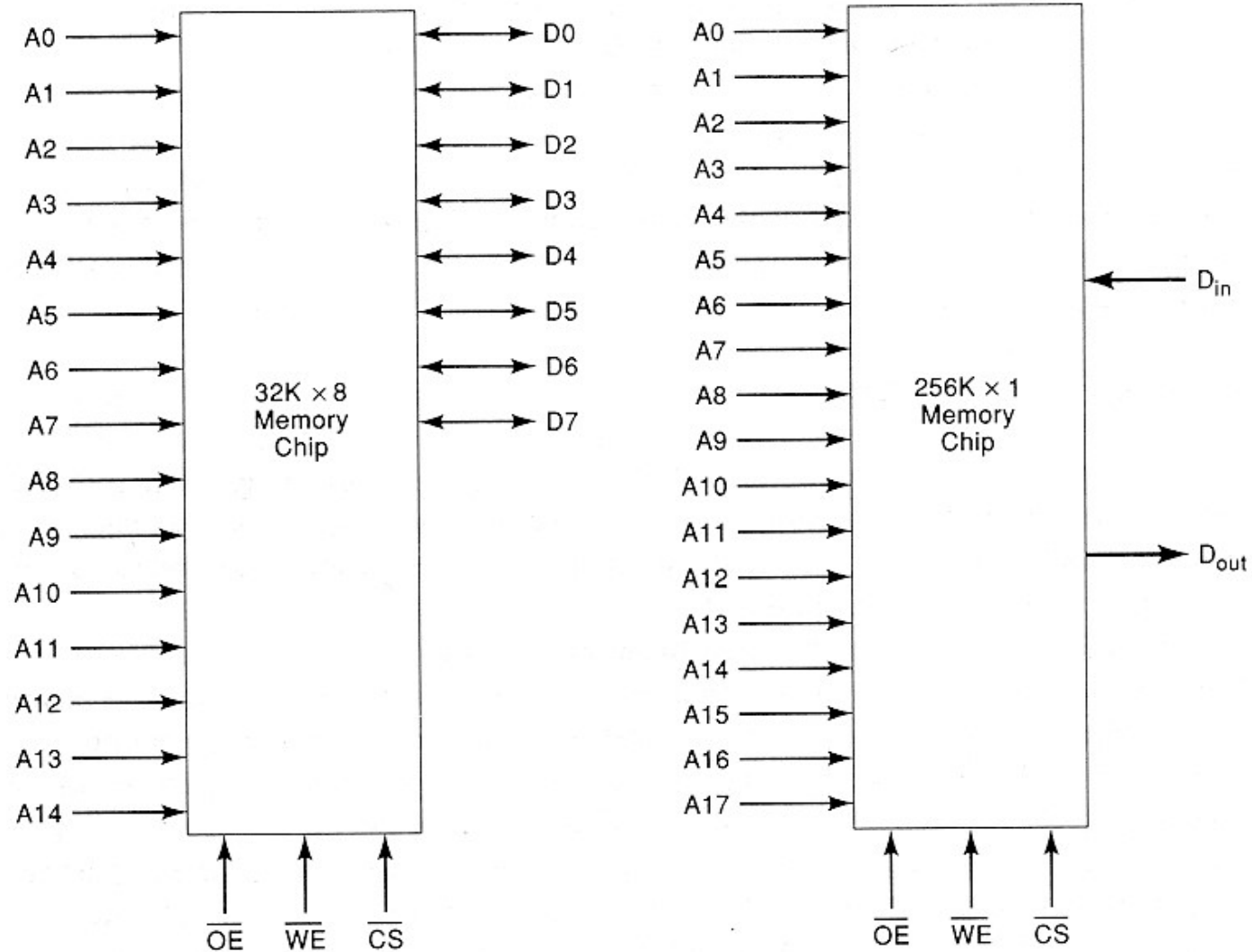
- **weitere oft verwendete Steuerleitungen**
 - OE: Output Enable
 - 1 aktiviert Ausgänge
 - 0 schaltet die Ausgänge hochohmig (in CMOS: beide Transistoren gesperrt)
 - dadurch können ausnahmsweise doch mehrere Ausgänge zusammengeschaltet werden, nur einer darf aktiviert sein
 - CS: Chip Select
 - wie ein Enable Eingang
 - 1 aktiviert den ganzen Chip (die ganze Schaltung)
 - RD: Read
 - 1 schaltet Speicher auf Lesemodus
 - WE: Write Enable
 - Alternative zu RD (häufiger so realisiert)
 - 1 schaltet Speicher auf Schreibmodus

Speicherorganisation (2)

- **Organisationsschema**

- ist leicht auf größere Speicher zu erweitern
- statt mehrere Bits parallel anzusprechen kann man die Leitungen auch nutzen, um eine größere Anzahl von Bits zu adressieren
 - dadurch kann man mit gleich vielen Leitungen mehr Bits speichern
 - oder gleich viele Bits mit weniger Leitungen

Beispiel: 256Kbit Speicher

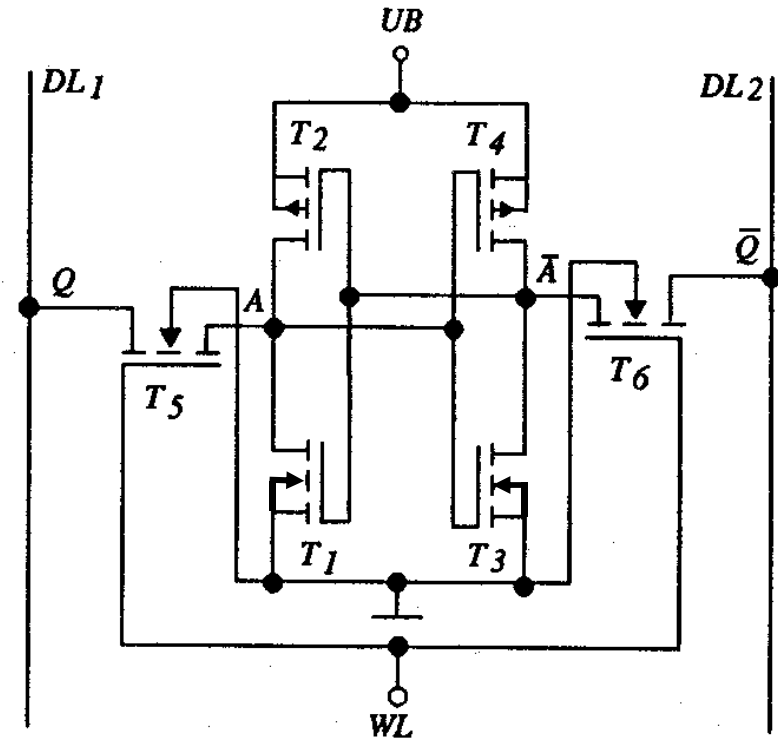


Speicherzellen

- **Speicherzellen sind auf Größe optimiert**
 - in Wahrheit keine komplexen Flip-Flops aus vielen Gattern
 - möglichst einfache Grundschialtung
 - dafür nimmt man einen erhöhten Aufwand zum Ansteuern (Speichern/Lesen) der Zellen in Kauf

CMOS-SRAM-Zelle

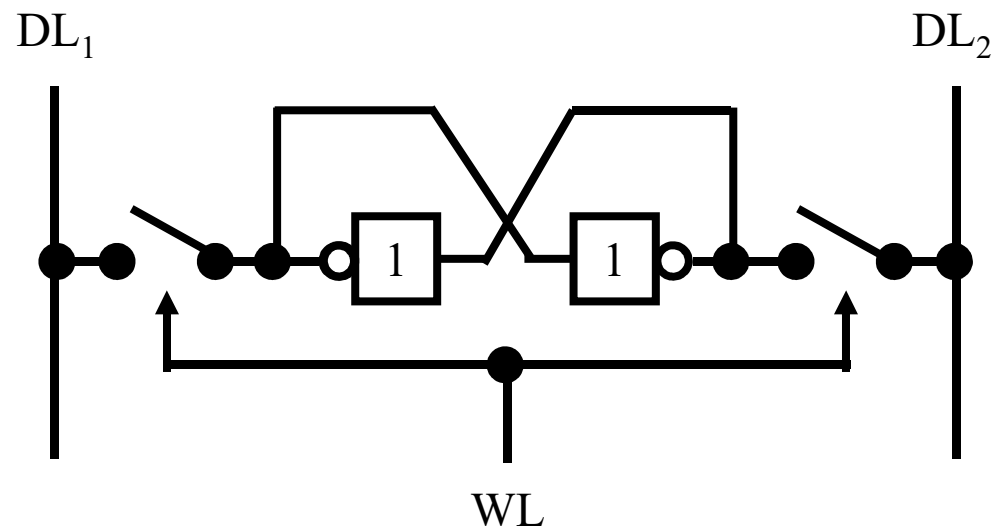
- rückgekoppelte CMOS-Inverter
- Aktivierung durch Wortleitung WL
 - Verbindung der Ausgänge mit zwei Datenleitungen, die für alle Zellen gemeinsamen sind
 - Lesen
 - Messen der Spannung auf den Datenleitungen
 - Schreiben
 - erzwungene L bzw. H Pegel auf den Datenleitungen



T_1, T_3, T_5 und T_6 : n-Kanal MOSFET
 T_2 und T_4 : p-Kanal MOSFET

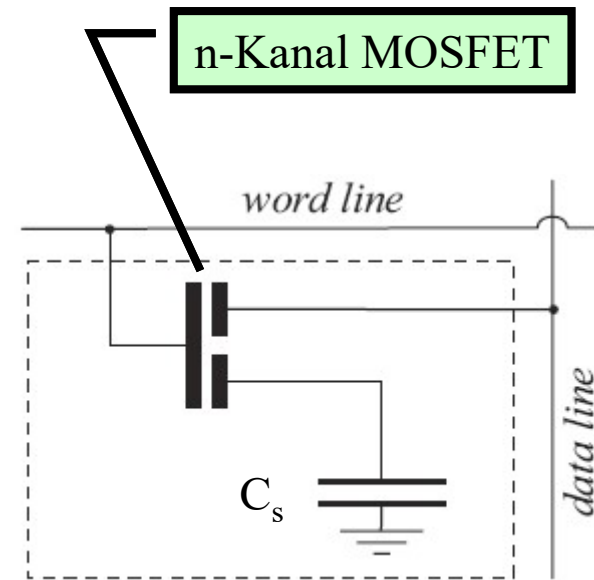
CMOS-SRAM-Zelle, vereinfacht

- Bistabiles Kippglied
- Aktivierung durch Wortleitung WL
 - Verbindung der Ausgänge mit zwei Datenleitungen, die für alle Zellen gemeinsamen sind
 - Lesen
 - Messen der Spannung auf den Datenleitungen
 - Schreiben
 - erzwungene L bzw. H Pegel auf den Datenleitungen
 - "schwache" Transistoren in den Invertern



DRAM-Zelle

- minimaler Platzbedarf, da Ein-Transistor Zelle
- Information als Spannung im Kondensator C_s
 - aus Platzgründen extrem kleine Kapazität
 - Spannung fällt durch Leckströme schnell ab
 - Refresh notwendig
 - zyklisches Lesen und Schreiben
- Adressierung über eine Wortleitung
- Schreiben/Lesen über eine Datenleitung
 - Schreiben
 - Spannung auf der Datenleitung lädt (logische 1) oder entlädt (logische 0) den Kondensator
 - Lesen
 - Ladung des Kondensators fließt über die Datenleitung ab und wird mit aufwändiger Schaltung gemessen (da die Ladungsmengen extrem klein sind)



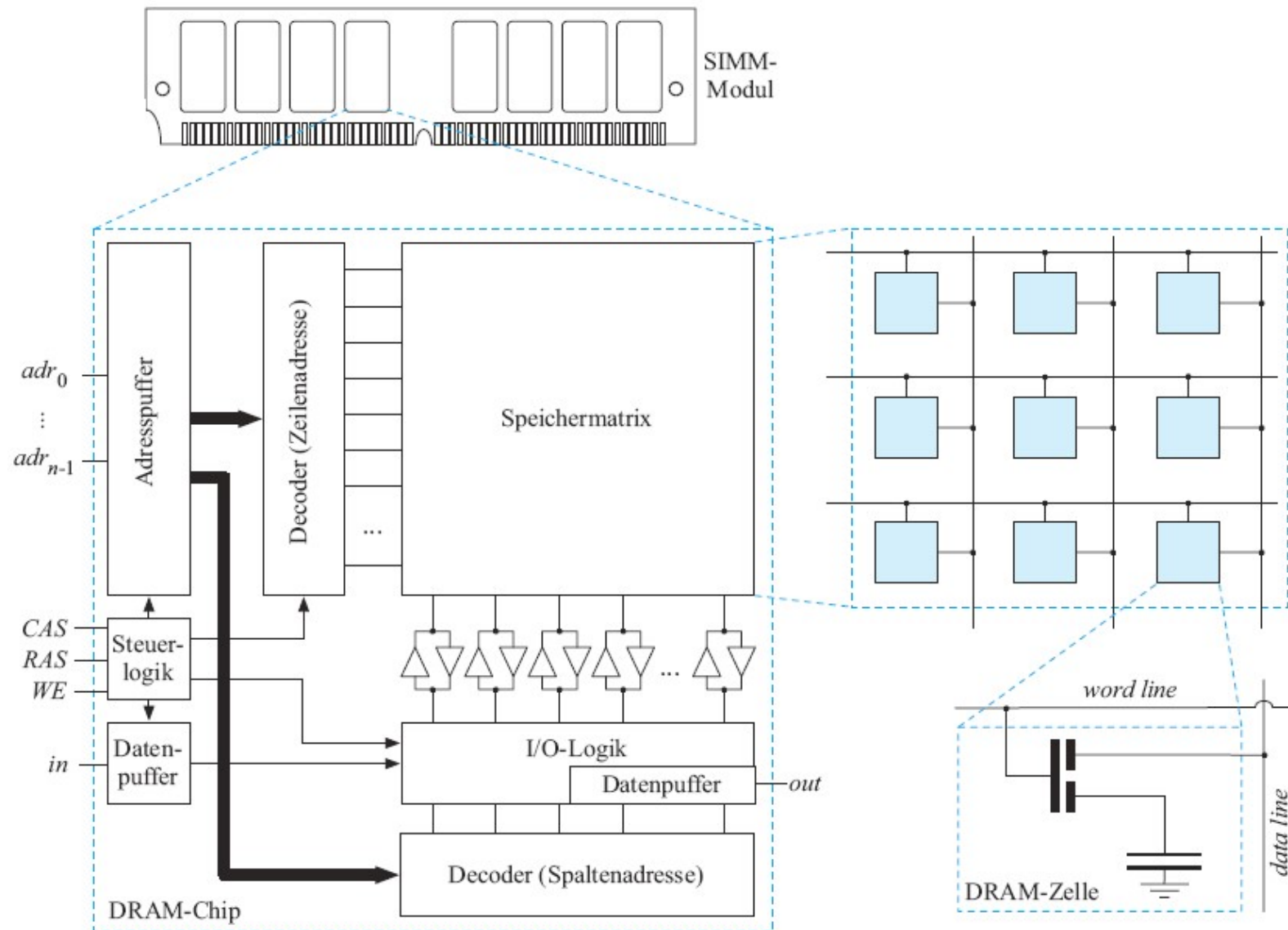
schematischer Aufbau



Selbstentladung

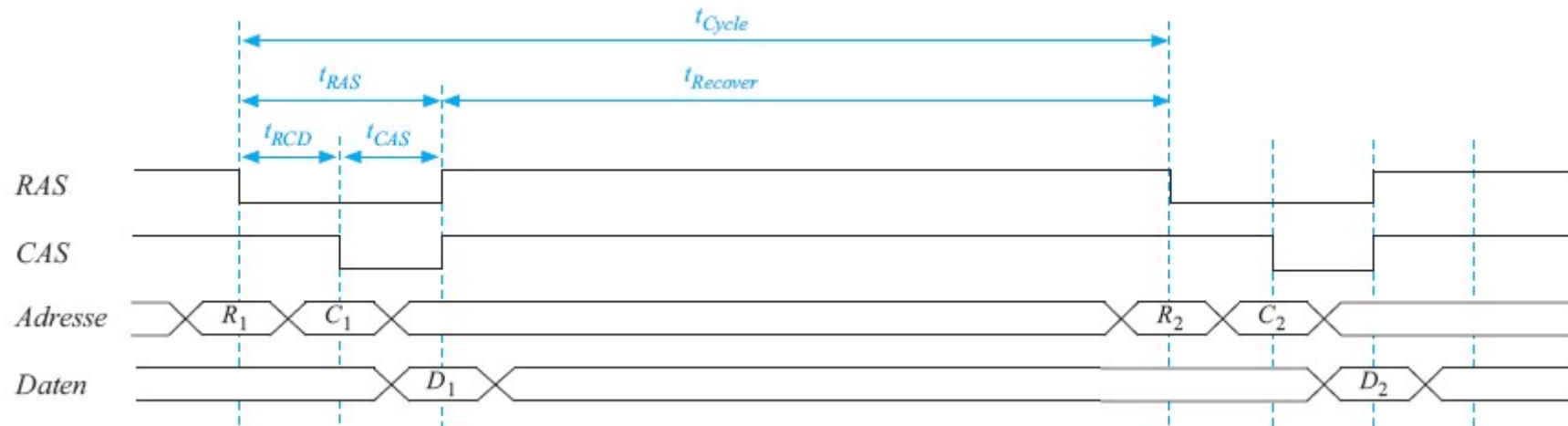


DRAM-Chip



DRAM-Speicherzugriff

- Zeilen großer DRAM Speicher enthalten eine ganze Seite (*page*)
- Adresse
 - obere Hälfte R_1 der Adress-Bits adressiert die Seite (Zeilenadresse, Row)
 - untere Hälfte C_1 der Adressbits adressieren das Bit innerhalb der Seite (Spaltenadresse, Column)
 - Übertragung nacheinander (RAS, *Row Address Strobe*, und CAS, *Column Address Strobe*, signalisieren, welcher Teil der Adresse übertragen wird)
 - spart Anschlüsse (*pins*), kostet aber keine Zeit, da die Seite erst in einen internen Puffer (Zwischenspeicher) übertragen werden muss



DRAM-Speicherzugriff (2)

- **Zugriffsarten (vereinfacht)**

- Normal Mode

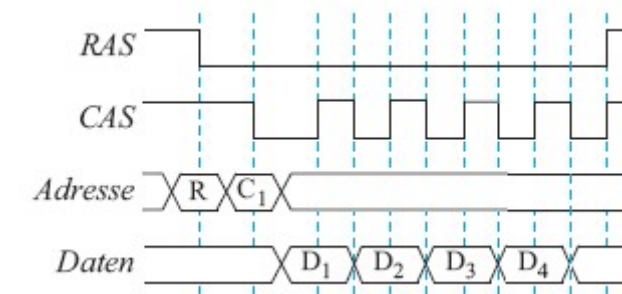
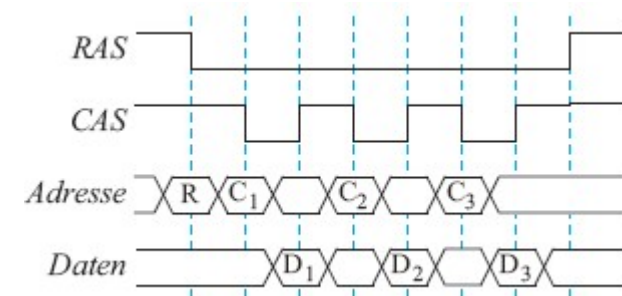
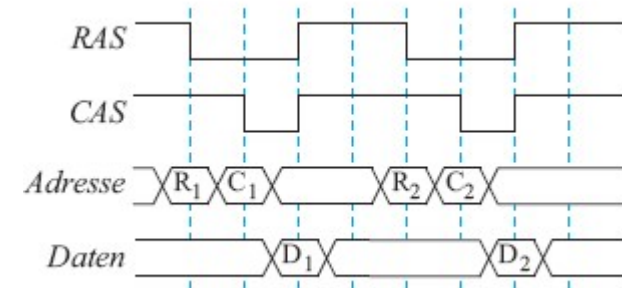
- es wird jeweils die vollständige Adresse übertragen
- langsam!

- Page Mode

- mehrere Bits werden von derselben Seite ausgelesen, ohne jeweils die Seitenadresse neu zu übertragen
- die Seite befindet sich ja noch im Puffer
- schneller

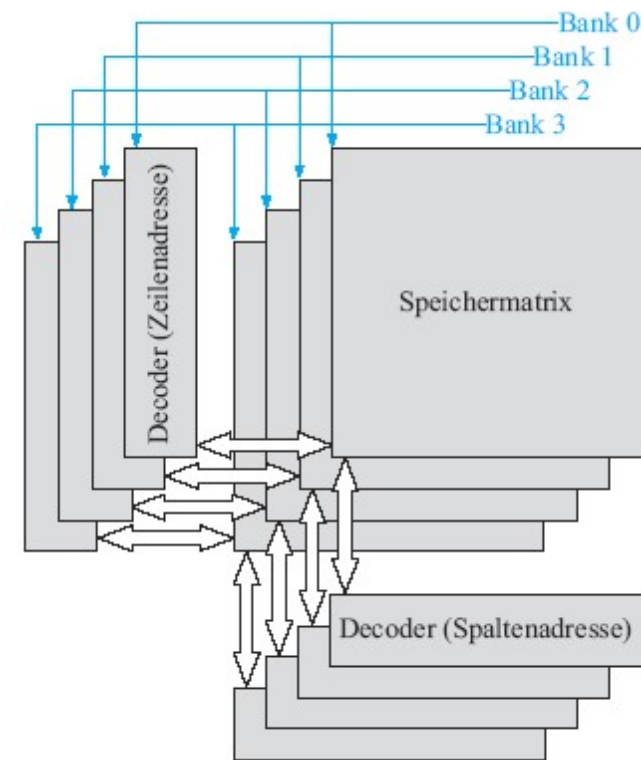
- Nibble Mode

- aufeinander folgende Bits werden ohne erneute Übertragung der Adressen ausgelesen
- noch schneller



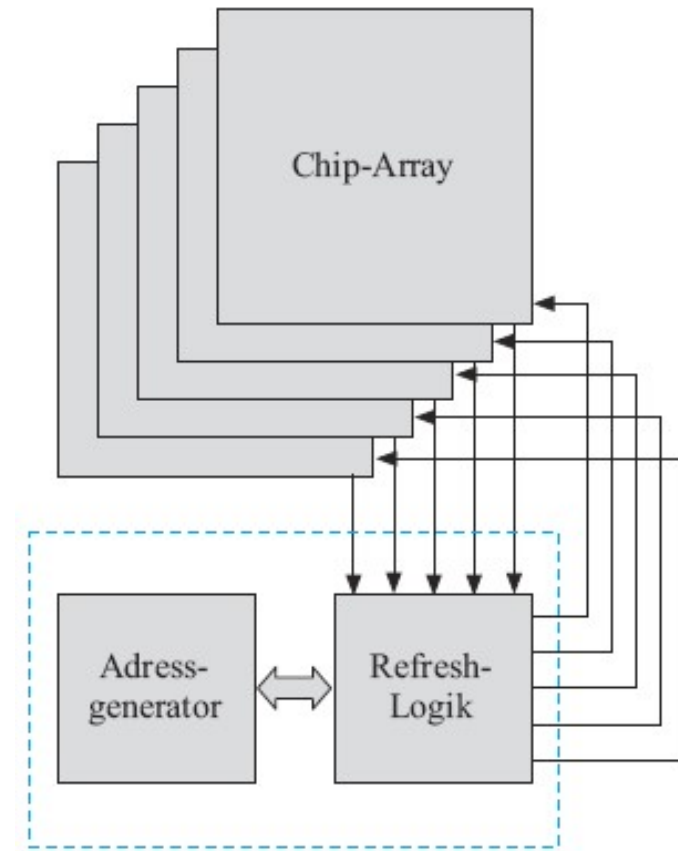
Speicherbänke

- nach jedem Seitenzugriff muss eine gewisse Zeit vergehen, bis der nächste Seitenzugriff möglich ist (s.o. $t_{Recover}$)
- **Abhilfe: Interleaving**
 - Speicher wird in mehrere Bänke aufgeteilt
 - jede Bank hat eigene CAS und RAS Leitungen und ist wie ein kompletter Speicherchip aufgebaut
 - aufeinander folgende Seiten werden über die verschiedenen Speicherbänke verteilt
 - greift man auf aufeinander folgende Seiten zu, muss $t_{Recover}$ nicht abgewartet werden
 - dadurch wird der Datendurchsatz (transferierte Datenmenge pro Zeit) enorm gesteigert



Refresh

- Daten können nur wenige Millisekunden in den Kondensatoren gespeichert werden
- daher müssen alle Seiten regelmäßig ausgelesen und zurückgeschrieben werden
- dazu dient die Refresh-Logik, die von einem Adressgenerator (Seitenzähler) die Seitenadressen bekommt



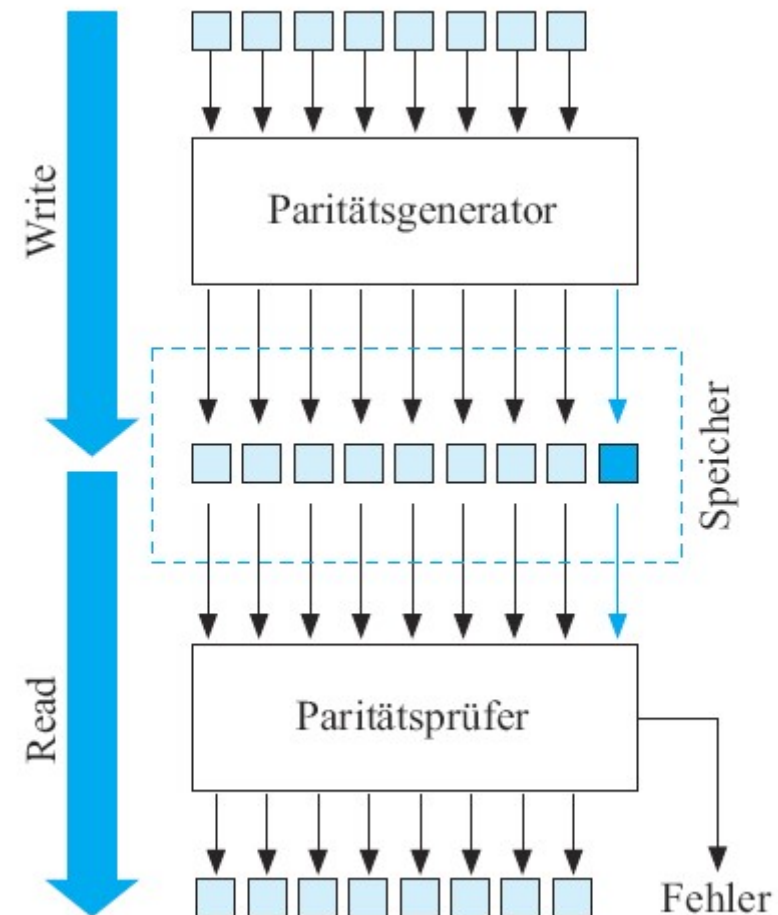
Fehlererkennung

- **DRAMs sind fehleranfälliger als SRAMs**
 - Speicherinhalt wird durch extrem kleine Ladungsmengen definiert
 - durch Leckströme (nicht perfekte Isolierung) fällt die Ladung weiter ab
 - natürliche Radioaktivität in den Gehäusematerialien führt zu Veränderungen der Ladung ("ionisierende Strahlung")
 - insbesondere α -Strahlung (Heliumkerne), die in das Silizium eindringt, kann Ladungen so stark verändern, dass aus einer 1 eine 0 wird oder umgekehrt
 - ist ein ***Soft-Error***, da der Fehler nur zufällig hier oder da auftritt, die Speicherstelle selbst arbeitet danach korrekt weiter
 - wird hohe Zuverlässigkeit verlangt (z.B. in Servern), müssen DRAM Speicher abgesichert werden, um das Risiko eines Soft-Errors zu reduzieren

Fehlererkennung (2)

- **Fehlererkennung mit Parity-Bit**

- für jedes gespeicherte Byte wird ein 9. Bit hinzugefügt
 - z.B. 9 Speicher-Chips statt 8
 - z.B. ungerade Parität: das 9. Bit wird so gesetzt, dass die Anzahl der 1 Bits ungerade ist
- beim Lesen wird die Parität geprüft
 - ist ein einzelnes Bit gekippt, wird der Fehler erkannt
 - Interrupt wird ausgelöst
 - Betriebssystem zeigt Fehlermeldung an und stoppt Programm



Fehlerkorrektur

- **Hamming-Code**

- 32 Bit Datenwort wird *geschickt* um 6 weitere Prüfbits ergänzt (Details siehe Codierungstheorie)
- der resultierende Code hat eine Hamming-Distanz von 3
 - d.h. es müssen mindestens 3 Bits gekippt werden um von einem gültigen Code zu einem anderen gültigen Code zu gelangen
 - Ein-Bit-Fehler können erkannt und korrigiert werden
 - fehlerhafte Codes werden dem nächstliegenden Code zugeordnet
 - Zwei-Bit-Fehler werden noch als Fehler erkannt, aber nicht korrigiert, Abbruch
- Speicher-Refresh wird damit zu einem Korrektur-Refresh erweitert

