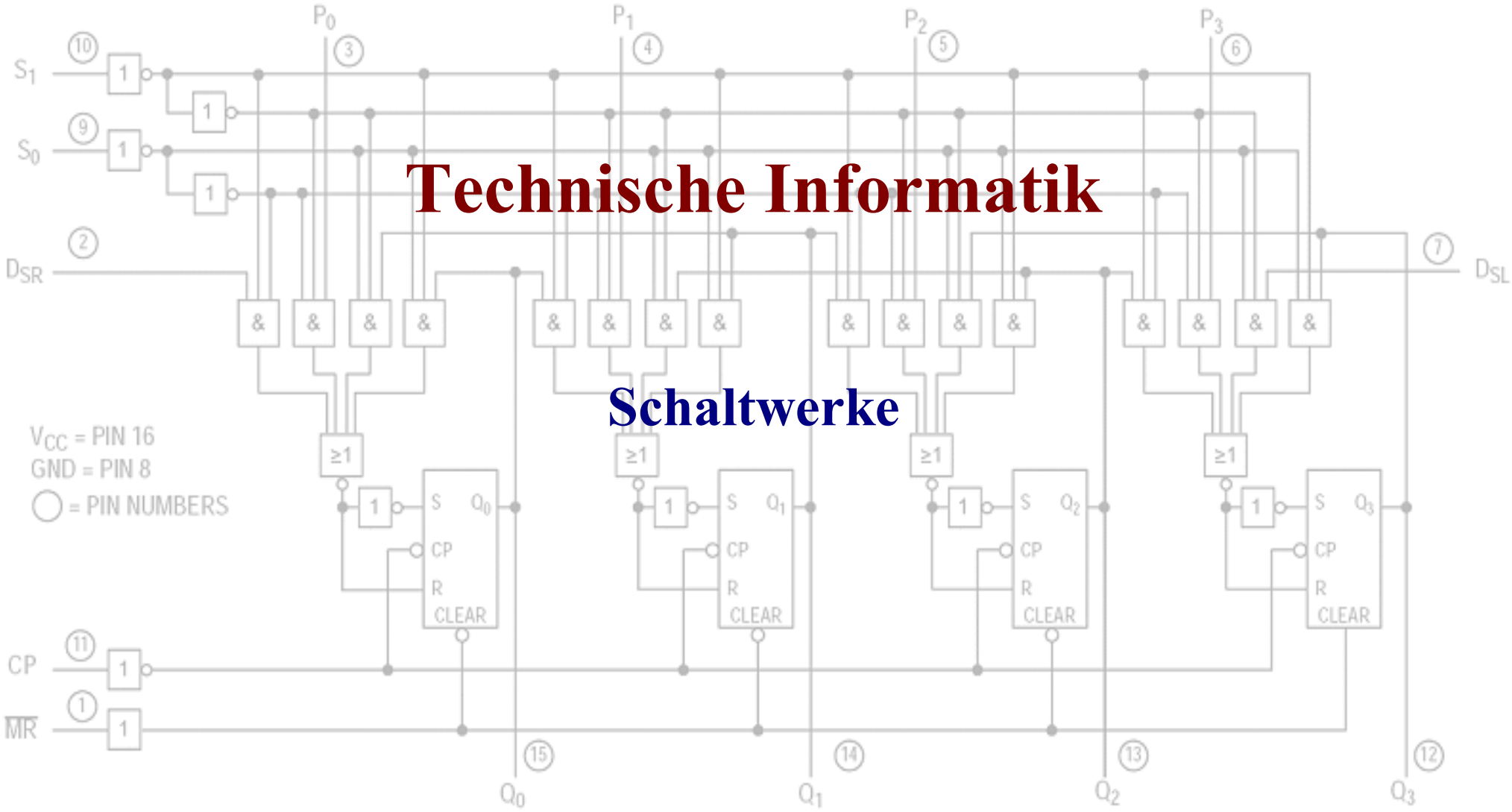
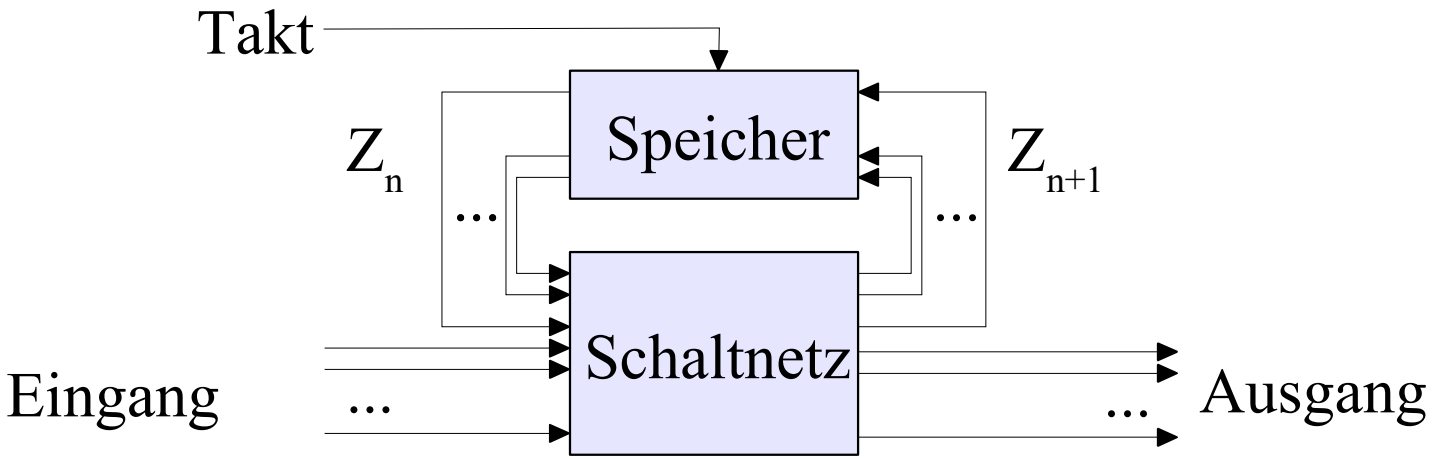
Technische Informatik

Schaltwerke



Grundlagen

- Basiselemente von *Schaltwerken* sind
  - Schaltnetze
  - Speicherelemente bzw. Flip-Flop-Stufen
- Zustände werden innerhalb des Schaltwerkes gespeichert
  - Das Ausgangssignal eines Schaltwerkes ist damit von den aktuellen und vorangegangenen Ereignissen abhängig („Gedächtniswirkung“)
  - Weitere Bezeichnungen: *sequenzielle Logik, endlicher Automat*
  - Schaltwerke lassen sich durch *Zustandsfolgetabellen* beschreiben.
    Innere bzw. vorherige Zustände werden mit $\mathbf{Z_n}$, Folgezustände mit $\mathbf{Z_{n+1}}$ bezeichnet



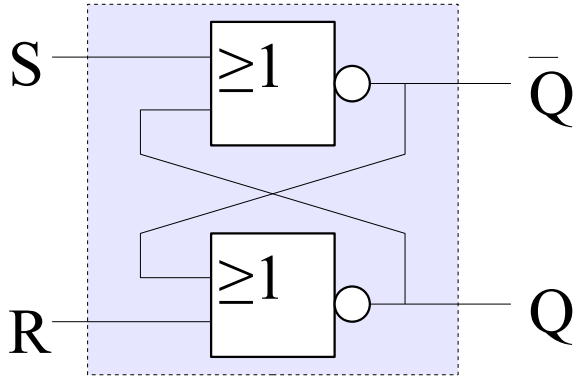
Flip-Flop

- Zustandsspeicherung mit Flip-Flop (Bistabile Kippstufe)
  - Digitale Grundschaltung mit
    - zwei inneren Zuständen
    - einem oder mehreren Eingängen
    - einem oder mehreren Ausgängen (optional: invertierter Ausgang)
    - ggf. Takt und Steuereingängen
- Der Speicherzustand ändert sich abhängig von den Signaleingängen bzw. vom Takt
- Grundaufgaben
  - Verknüpfung der Signaleingängen untereinander und/oder mit dem Takt
  - Zustandsspeicherung (Schreiben, Lesen, Speichern)
- Aufbau
  - Rückgekoppelte Gatter
  - Als Halbleiterelemente erhältlich

Grundprinzip eines Flip-Flops

- Reset/Set-Flip-Flop (RS-Flip-Flop, RS-Latch)
  - Funktionsprinzip:
    - Zunächst sei $S = R = 0$, dann erzeugt eine „1“ an einem der ODER-Gatter folglich eine „0“ am anderen ODER-Gatter. Der aktuelle Zustand bleibt erhalten (= Speichern)
    - Wird nun $S = 1$ und $R = 0$ gesetzt, liefert das obere ODER-Gatter eine „0“ und das untere Gatter eine „1“. Folge: Der Zustand $Q_{n+1} = 1$ wird gesetzt (= Setzen)
    - Mit $S = 0$ und $R = 1$ wird analog der Zustand $Q_{n+1} = 0$ gesetzt (= Rücksetzen)
    - Der Eingangszustand $S = R = 1$ ist nicht erlaubt, da $Q_{n+1} = \neg Q_{n+1} = 0$ wird und eine gleichzeitige Änderung der Eingangszustände zu einem unvorhersehbaren Zustand führt

Schaltungsprinzip:



Zustandstabelle:

Eingang		*Ausgang*	
S	***R***	Q_{n+1}	
0	0	Q_n	speichern
1	0	1	setzen
0	1	0	rücksetzen
1	1	-	nicht erlaubt

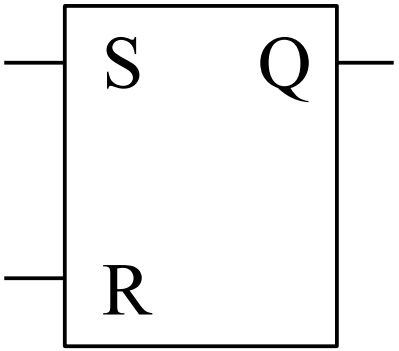
Flip-Flop-Typen

- Einteilung der Flip-Flops nach Taktverfahren:
  - ungetaktet (kein Takteingang)
    - RS-Flip-Flop
  - taktzustandsgesteuert (transparente Flip-Flops)
    - Eingang wirkt sich auf den Ausgang aus, solange Takteingang gesetzt ist.
    - Speicherung erfolgt sobald Takteingang zurückgesetzt wird
    - RS-Flip-Flop
    - D-Flip-Flop
  - taktflankengesteuert (Master-Slave-Flip-Flops)
    - Zustandsänderung erfolgt mit positiver bzw. negativer Taktflanke
    - Realisierung durch Hintereinanderschalten zweier Flip-Flops (Zwischenspeicherung)
    - D-Flip-Flop
    - JK-Flip-Flop
    - T-Flip-Flop

RS-Flip-Flop, ungetaktet

- Eigenschaften:
  - Alternative Bezeichnung: RS-Latch
  - Eingänge: S (= Set) und R (= Reset)
  - kein Takteingang
  - Funktionen: Setzen, Rücksetzen, Speichern
  - Eingangszustand S = R = 1 nicht definiert

Schaltsymbol:



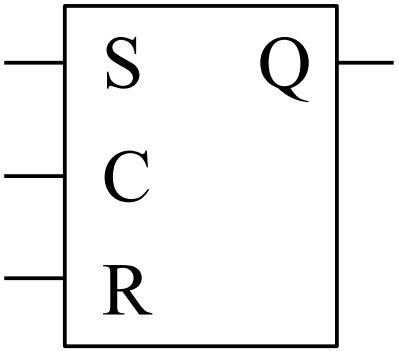
Zustandstabelle:

Eingang		*Ausgang*
S	***R***	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	undef.

RS-Flip-Flop, zustandsgesteuert

- Eigenschaften:
  - Eingänge: S (= Set) und R (= Reset)
  - „Takt“eingang C
  - Eingänge R und S nur bei $C = 1$ aktiv (transparent)
  - Funktionen: Setzen, Rücksetzen, Speichern
  - Eingangszustand $S = R = 1$ nicht definiert

Schaltsymbol:



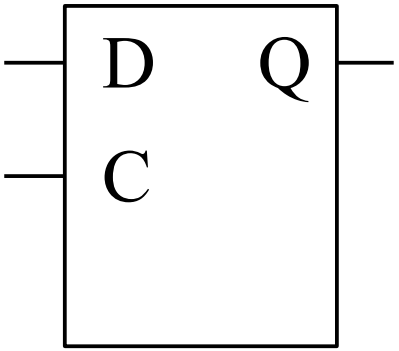
Zustandstabelle:

Eingang			*Ausgang*
S	***R***	***C***	Q_{n+1}
x	x	0	Q_n
0	0	1	Q_n
0	1	1	0
1	0	1	1
1	1	1	undef.

D-Flip-Flop, zustandsgesteuert

- Eigenschaften:
  - Eingang: D (= Data)
  - „Takt“eingang C
  - Eingang D nur bei C = 1 aktiv(transparent)
  - Funktionen: Speicherwert setzen, Speicherwert behalten
  - „Latch“

Schaltsymbol:



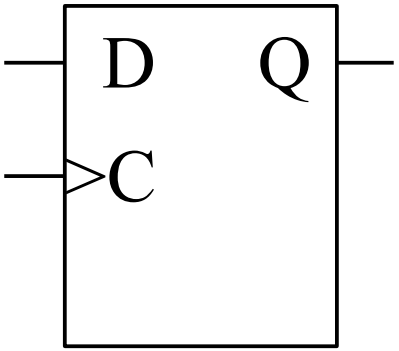
Zustandstabelle:

Eingang		*Ausgang*
D	***C***	Q_{n+1}
x	0	Q_n
0	1	0
1	1	1

D-Flip-Flop, flankengesteuert

- Eigenschaften:
  - Eingang: D (= Data)
  - Takteingang C
  - Eingang D nur bei positiver Taktflanke $C = \uparrow$ aktiv
  - Funktionen: Speicherwert setzen, Speicherwert behalten
  - Master-Slave-Flip-Flop

Schaltsymbol:



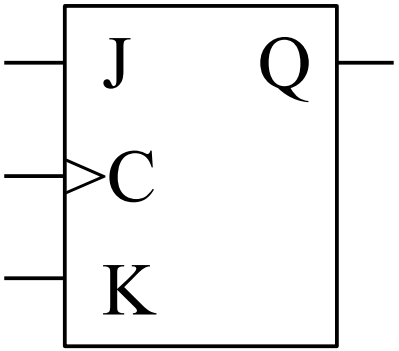
Zustandstabelle:

Eingang		*Ausgang*
D	***C***	Q_{n+1}
x	0	Q_n
x	1	Q_n
x	↓	Q_n
0	↑	0
1	↑	1

JK-Flip-Flop, flankengesteuert

- Eigenschaften:
  - Eingänge: J (= Set) und K (= Reset)
  - Takteingang C
  - Eingänge J und K nur bei positiver Taktflanke $C = \uparrow$ aktiv
  - Funktionen: Setzen, Rücksetzen, Invertieren, Speicherwert behalten
  - Master-Slave-Flip-Flop

Schaltsymbol:



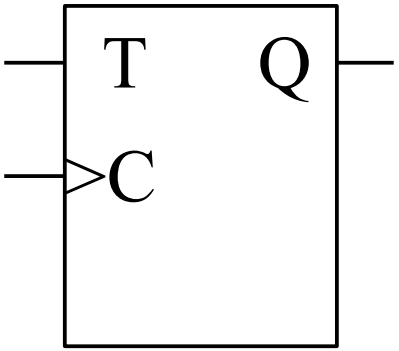
Zustandstabelle:

Eingang			*Ausgang*
J	***K***	***C***	Q_{n+1}
x	x	0	Q_n
x	x	1	Q_n
x	x	↓	Q_n
0	0	↑	Q_n
0	1	↑	0
1	0	↑	1
1	1	↑	$\neg Q_n$

T-Flip-Flop, flankengesteuert

- Eigenschaften:
  - Eingang: T (= Toggle)
  - Takteingang C
  - Eingang T nur bei positiver Taktflanke $C = \uparrow$ aktiv
  - Funktionen: Invertieren, Speicherwert behalten (Toggle-Flip-Flop)
  - Master-Slave-Flip-Flop

Schaltsymbol:

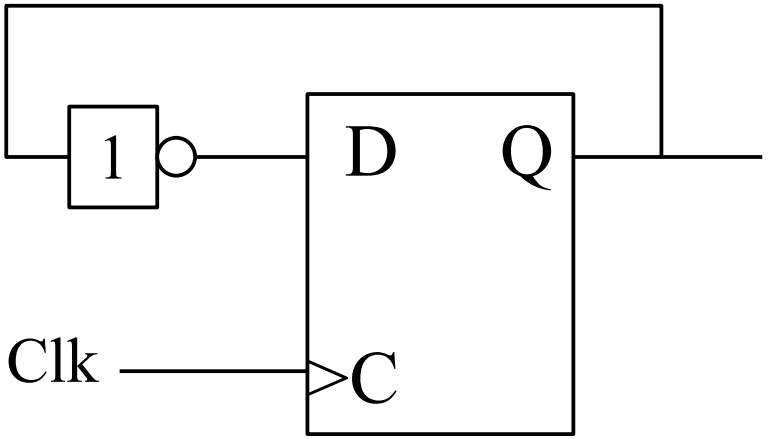


Zustandstabelle:

Eingang		*Ausgang*
T	***C***	Q_{n+1}
x	0	Q_n
x	1	Q_n
x	↓	Q_n
0	↑	Q_n
1	↑	$\neg Q_n$

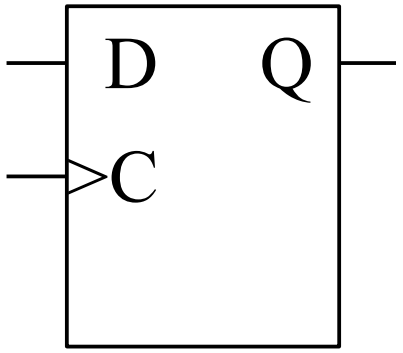
Rückkopplung

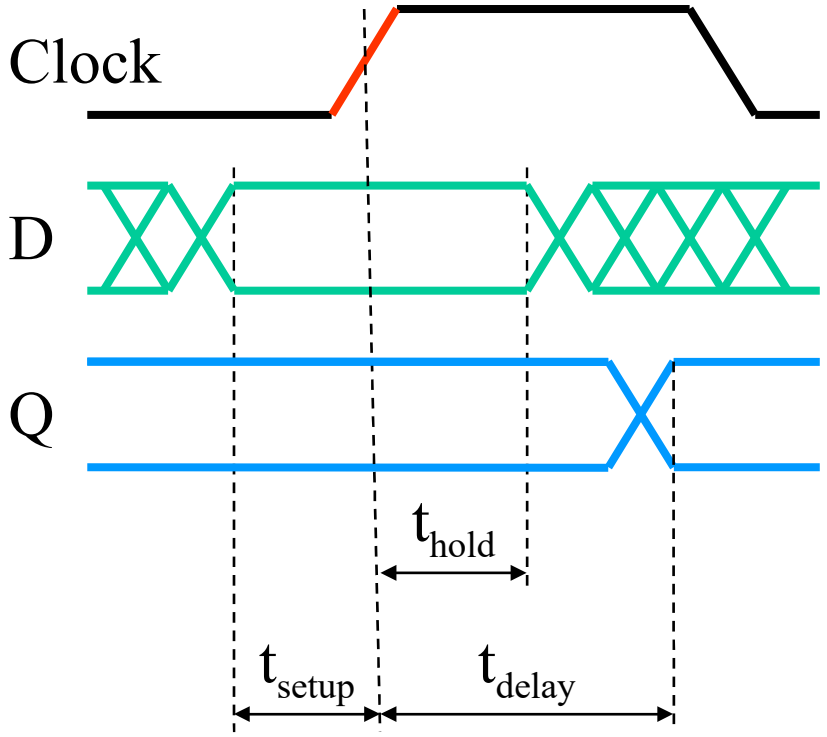
- Im Gegensatz zu taktzustandsgesteuerten Flip-Flops können die Ausgänge taktflankengesteuerte Flip-Flops auf die Eingänge zurückgekoppelt werden
  - Diese Rückkopplung ist essentiell für Zustandsautomaten (Sequentielle Logik), da Folgezustände auch von vorherigen Zuständen abhängig sein können
  - Taktflankengesteuerte Flip-Flops aktualisieren den Ausgang erst nachdem die Taktflanke beendet und der Eingang verriegelt ist



Schaltzeiten

- Eingangssignal ***D*** wird nur zum Zeitpunkt der positiven Takt**flanke** gelesen
- Der gelesene Zustand wird mit einer Verzögerung an den Ausgang ***Q*** weitergegeben
- Das Eingangssignal D darf sich um den Zeitpunkt der positiven Taktflanke herum nicht ändern
  - t_{setup} : Zeit vor der positiven Taktflanke, zu der das Eingangssignal (spätestens) stabil anliegen muss
  - t_{hold} : Zeit nach der positiven Taktflanke, in der das Eingangssignal (noch) stabil anliegen muss
  - t_{delay} : Zeit nach der positiven Taktflanke, bis der neue Zustand stabil ausgegeben wird



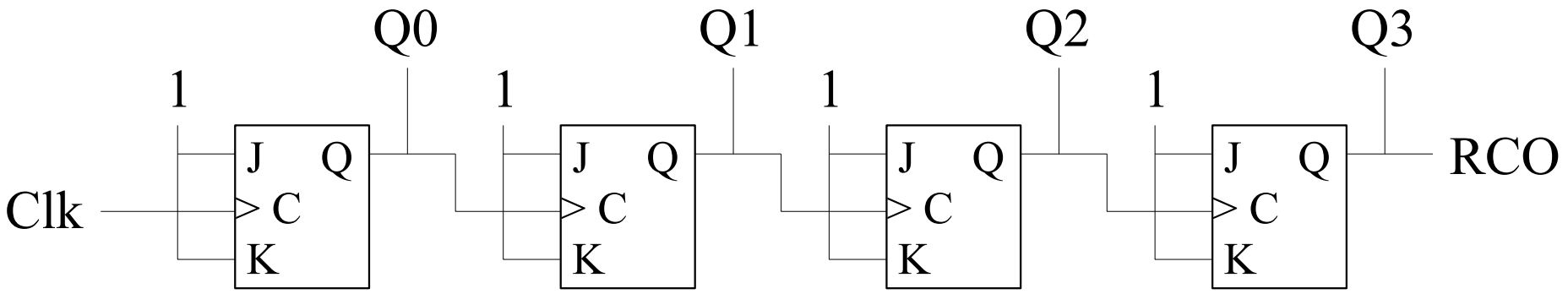


Asynchrone Schaltwerke

- Schaltungsprinzip
  - Ein System, dass zwar auch getaktet sein kann, aber in dem auch Signale ohne direkten Taktbezug Einfluss haben können und in dem keine festen Zeitbezüge zwischen dem Takt und den Signalen obligatorisch sind.
- Vorteile
  - Für dieselbe Anwendung erfordern asynchrone Schaltungen i.d.R. weniger Schaltelemente als synchrone Schaltungen
  - Die maximale Schaltfrequenz kann lokal variieren
  - Zeitliche Verteilung der Schaltvorgänge vermindert i.d.R. EMV-Probleme
- Nachteile
  - Entwicklungsaufwand für ein stabiles System ist i.d.R. wesentlich größer
  - Komplexe Systeme werden oft unbeherrschbar: Resynchronisation

Beispiel: Asynchroner Dualzähler

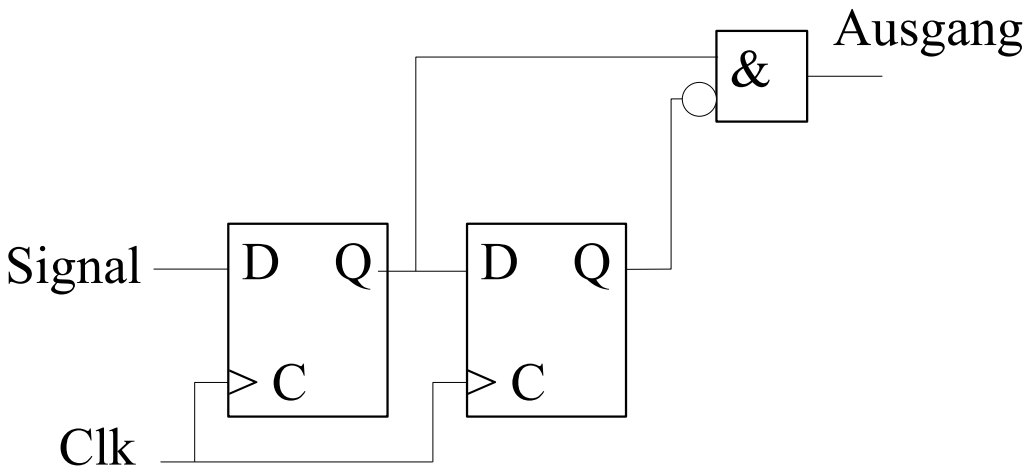
- Reihenschaltung aus *n* flankengesteuerten JK-Flip-Flops
  - Der Takteingang C_n wird mit dem Ausgang Q_{n-1} des vorherigen Flip-Flops verbunden
  - Kein gemeinsamer Takt
  - Der Zähler lässt sich beliebig erweitern (RCO: Ripply Carry Overflow)
- Verwendung als *Frequenzteiler* möglich
  - Der Ausgang Q_n liefert die Frequenz $f_n = \frac{f_{Clk}}{2^{n+1}}$



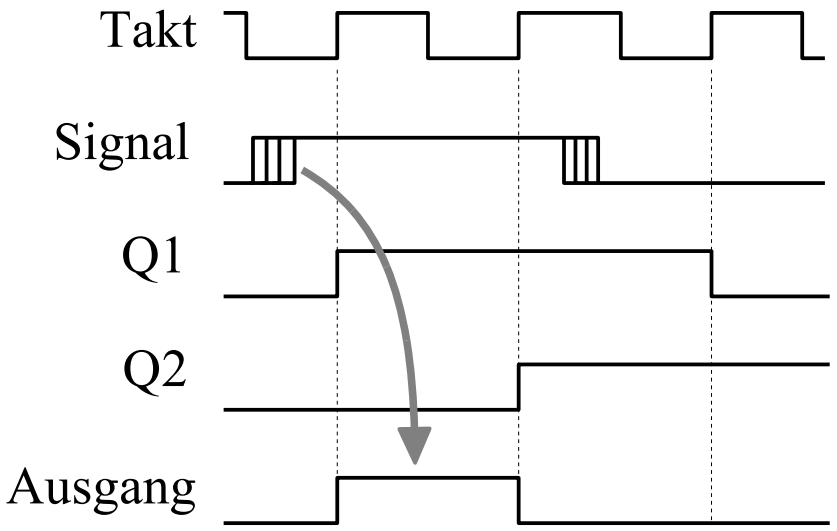
Synchronisation asynchroner Signale

- Asynchrone Signale lassen sich durch Flip-Flop-Schaltungen synchronisieren
- Der Ausgang liefert die synchronisierte positive Flanke des Eingangssignals
- Varianten: Synchronisation negativer oder beider Flanke durch entsprechende Verknüpfung der Zustände Q1 und Q2

Schaltung:

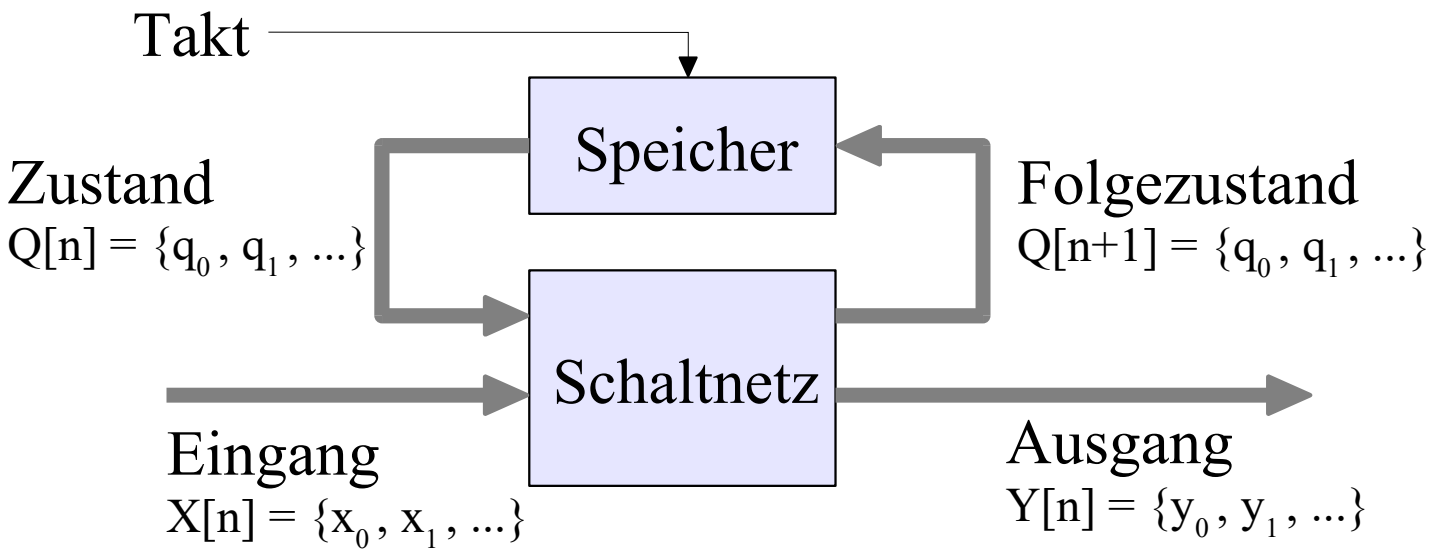


Zustandsdiagramm:



Synchrone Schaltwerke

- Schaltungsprinzip
  - Verknüpfung von Flip-Flop-Stufen mit Schaltnetzen (Kombinatorik)
  - Gemeinsamer Takt für alle Flip-Flops
  - Ausgang des Schaltnetzes wird (teilweise) gespeichert und wird nach dem Takt an den Eingang des Schaltnetzes gelegt
  - Der Ausgang des Schaltnetzes ist so vom aktuellen **und** von früheren Eingängen abhängig



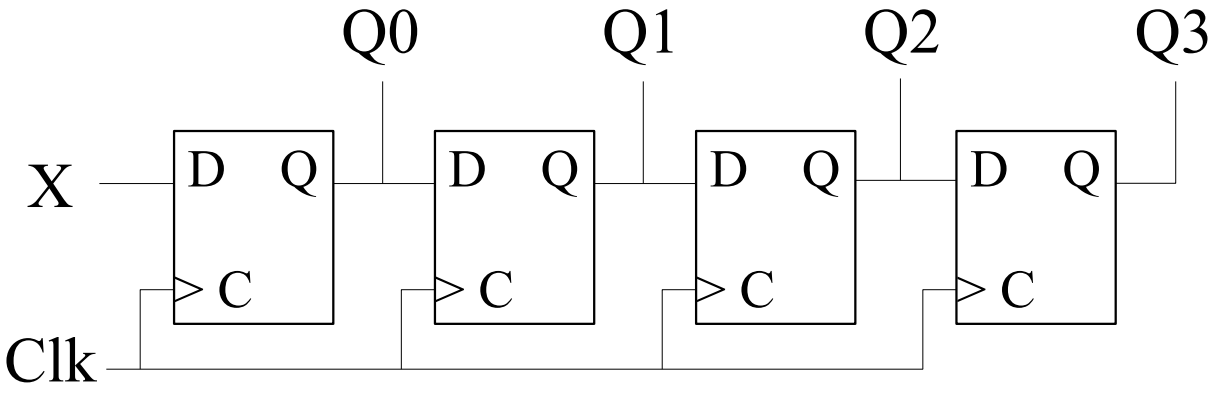
Synchrone Schaltwerke

- Vorteile
  - Übersichtliche Gestaltung der Funktion und der Dimensionierung auch bei komplexen Systemen
  - Bei geeigneter Auslegung unempfindlich gegen Bauteilparameterschwankungen
  - System kann i.d.R. auch beliebig langsam betrieben werden - d.h. erleichterte Testbarkeit
- Nachteile
  - Alle Komponenten schalten gleichzeitig: EMV-Problem
  - Die langsamste Komponente bestimmt die Geschwindigkeit
  - Oft erhöhter Aufwand an Schaltelementen
- Beispiele
  - Schieberegister
  - Synchronzähler
  - SDRAM-Speicher
  - Prozessoren

Beispiel: Synchrones Schieberegister

- Reihenschaltung aus n flankengesteuerten D-Flip-Flops
  - Der Ausgang Q_x eines Flip-Flops ist auf den Eingang des nachfolgenden Flip-Flops geschaltet
  - Der Eingangswert X wird mit jeder Taktflanke in das nachfolgende Flip-Flop geschrieben
  - Nach n Taktflanken liegen die n vorangegangenen Eingangswerte an den parallelen Ausgängen Q_x

Schaltung:



Zustandstabelle:

Takt n	*Ausgang n+1*			
	$Q0_{n+1}$	$Q1_{n+1}$	$Q2_{n+1}$	$Q3_{n+1}$
-1	-	-	-	-
0	X_0	-	-	-
1	X_1	X_0	-	-
2	X_2	X_1	X_0	-
3	X_3	X_2	X_1	X_0
4	X_4	X_3	X_2	X_1

Beispiel: Synchroner Dualzähler

- Reihenschaltung aus *n* T-Flip-Flops
  - Ein Flip-Flop Q_n muss seinen Zustand genau dann invertieren, wenn für alle niederwertigeren Flip-Flops $Q_{n-1} = Q_{n-2} = ... = Q_0 = 1$ gilt.

    Daher:

    $T_0 = 1$

    $T_1 = Q_0$

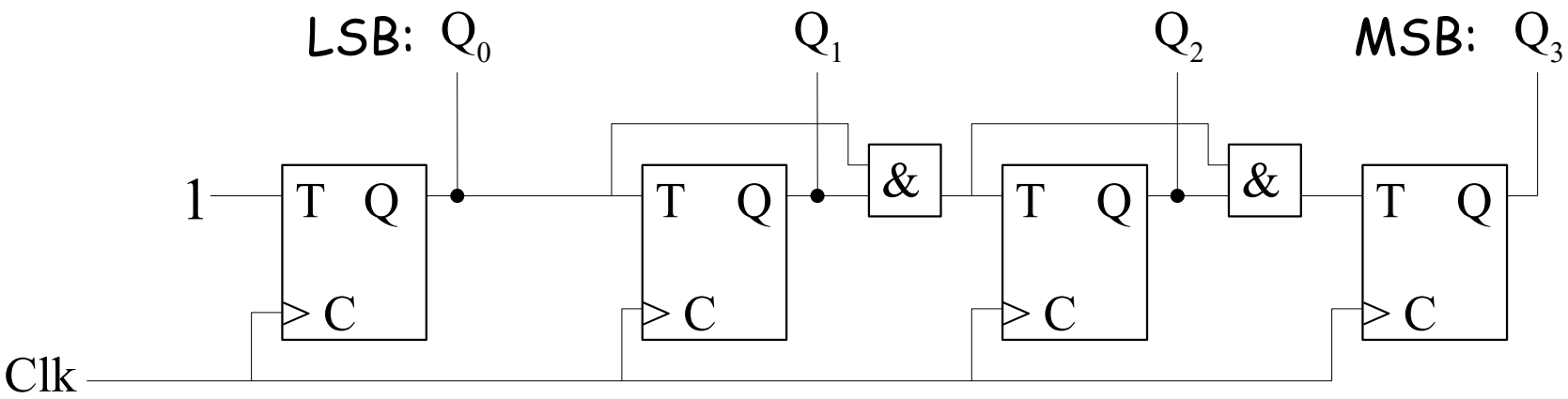
    $T_2 = Q_0 \cdot Q_1$

    ...

    $T_{n-1} = Q_0 \cdot Q_1 \cdot ... \cdot Q_{n-2}$

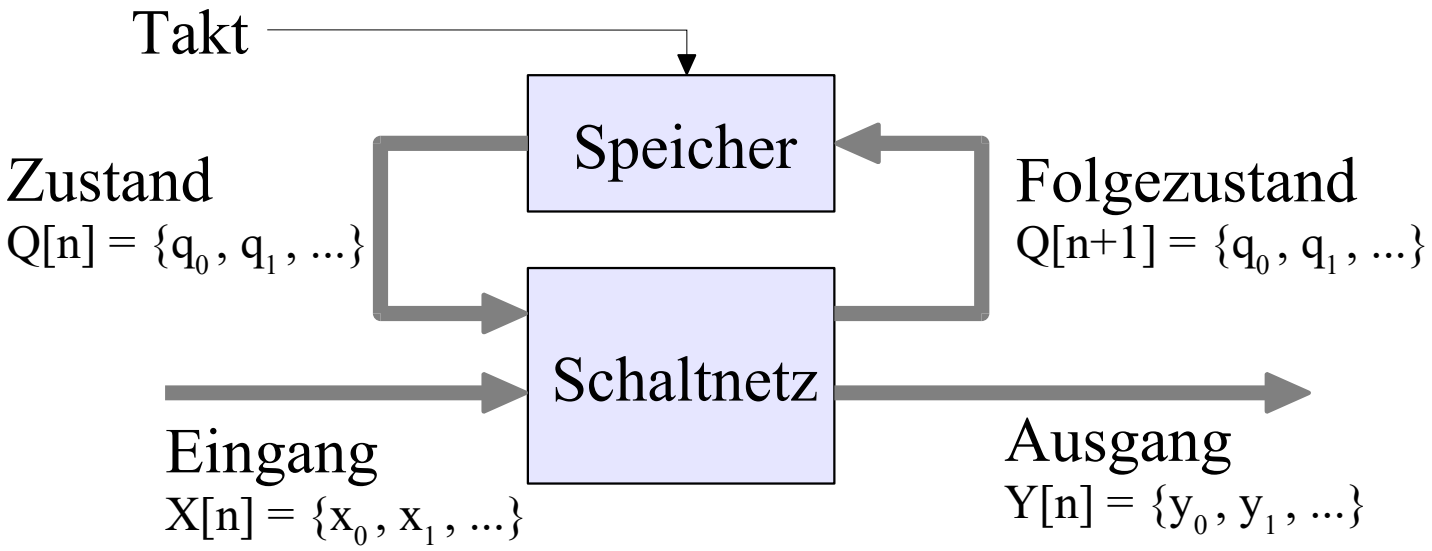
Zahl	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
usw.				

- Realisierung:



Systematische Beschreibung von Schaltwerken

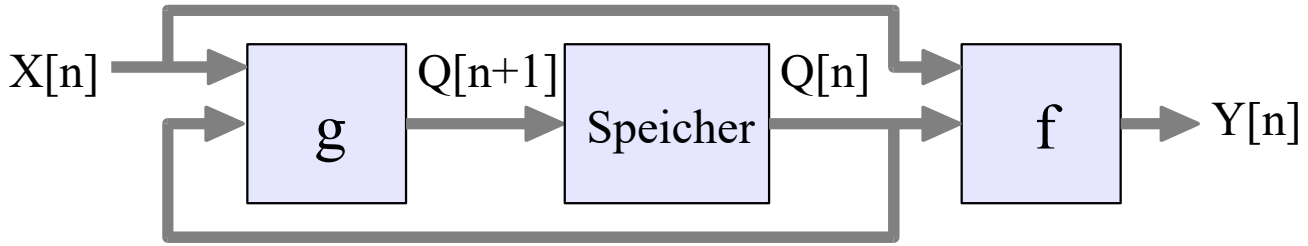
- Schaltwerke werden durch die *Automatentheorie* beschrieben
  - Ein Schaltwerk setzt sich aus Schaltnetz und Zustandsspeicher zusammen
  - Eingänge, Ausgänge und innere Zustände werden als Vektoren betrachtet



Schaltfunktionen

- *Schaltfunktionen* beschreiben die Abhängigkeit des Ausgabe- und Folgezustandsvektors vom Eingabe- und Zustandsvektor
  - Übergangsfunktion g
  - Ausgabefunktion f
- Grundstrukturen:

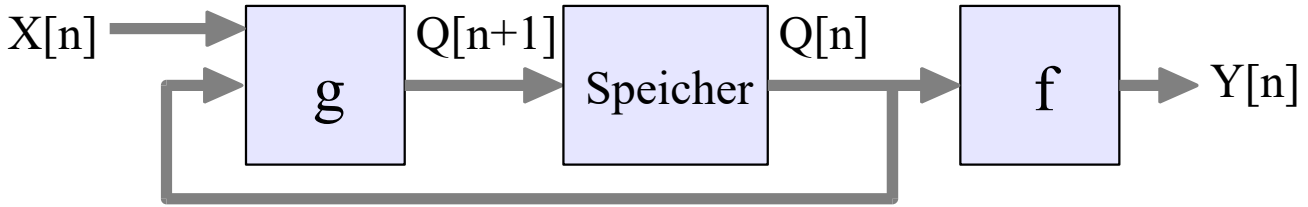
Mealy-Automat (synchron & asynchron):



Schaltfunktionen:

$$Y[n] = \boldsymbol{f}(X[n], Q[n])$$
$$Q[n+1] = \boldsymbol{g}(X[n], Q[n])$$

Moore-Automat (synchron):



$$Y[n] = \boldsymbol{f}(Q[n])$$
$$Q[n+1] = \boldsymbol{g}(X[n], Q[n])$$

Zustandfolgestabelle

- *Zustandsfolgetabelle*
  - Ähnlich aufgebaut wie Wahrheitstabellen
  - Eingangsvariablen: Eingangsvektor *X[n]* und Zustandsvektor Q*[n]*
  - Ausgangsvariablen: Folgezustandsvektor Q*[n+1]* und Ausgangsvektor *Y[n]*

- Aufbau:

Ist-Zustand vor der Taktflanke

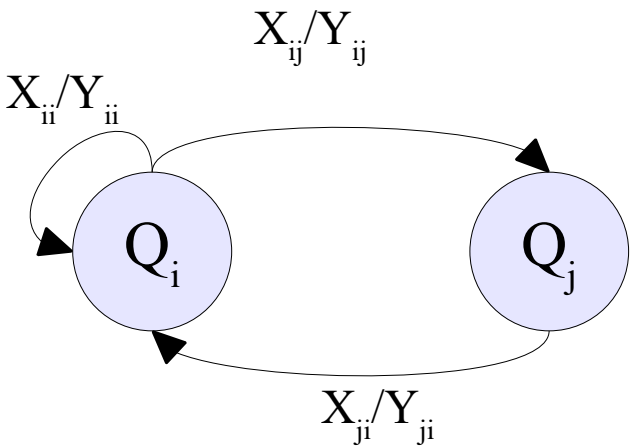
Neuer-Zustand nach der Taktflanke

Eingang			***Zustand***			***Folgezustand***			***Ausgang***		
x_0	x_1	...	$q_0[n]$	$q_1[n]$	...	$q_0[n+1]$	$q_1[n+1]$	...	y_0	y_1	...
0	0	0	0	0	0						
0	0	0	0	0	1						
			…				…			...	
1	1	1	1	1	1						

  - Diese Tabelle gibt an, welcher Folgezustand und Ausgang sich nach einer Taktflanke bei gegebenem Eingang und Ist-Zustand ergibt

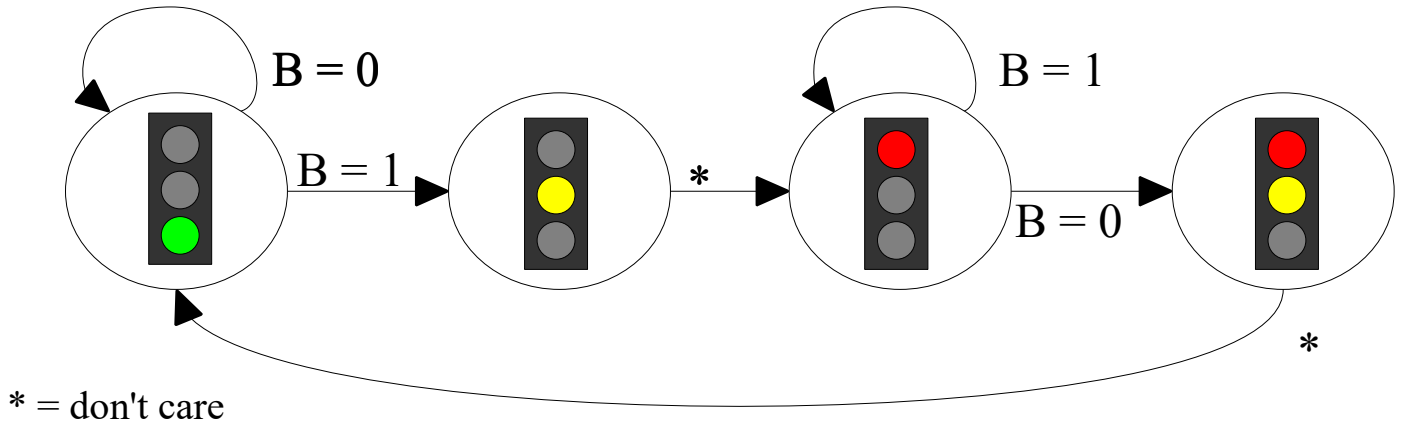
Zustandsgraph

- *Zustandsgraph* (*Zustandsdiagramm*)
- Grafische Darstellung der Zustände (*Knoten*) und Zustandsübergänge (*gerichtete Kanten*)
- Bezeichnung der Knoten:
  - Zustand Q_i
- Bezeichnung der Kanten:
  - Übergangsbedingung = Eingangswerte X_{ij}, bei der ein Übergang von *i* nach *j* stattfindet.
  - Optional auch der Ausgangswert Y_{ij}
- Ein Zustandsgraf ist äquivalent zur Zustandsfolgetabelle

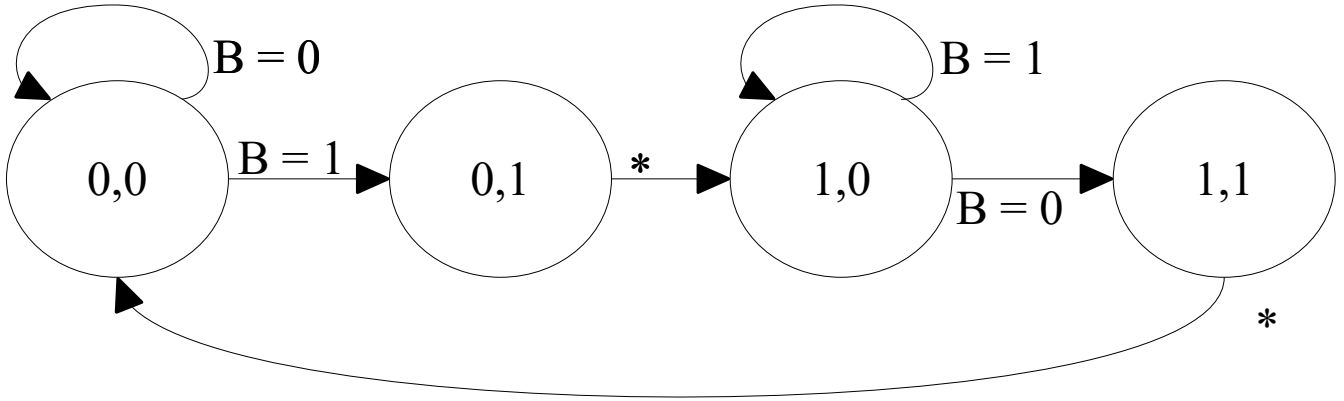


Entwurf

- *Beispiel Ampelsteuerung*
- Eine Verkehrsampel soll so gesteuert werden, dass
  - Im Ruhezustand (B=0) permanent „Grün“ angezeigt wird
  - Im Bedarfsfalle (B=1) sollen nacheinander die Phasen „Gelb“, „Rot“, „Rot-Gelb“ „Grün“ durchlaufen werden
  - Die Phase „Rot“ bleibt solange erhalten, wie der Bedarf (B=1) besteht
- Entwurf
  - 1. Schritt: *Spezifikation* in *Zustandsgrafen* überführen
    - Eindeutige, widerspruchsfreie formale Spezifikation



- *Beispiel Ampelsteuerung*
  - 2. Schritt: Die 4 Zustände lassen sich durch 2 Flip-Flops darstellen
    - Den Ampelzuständen werden Speicherzustände (Q_1, Q_0) zugeordnet



* = don't care

  - 3. Schritt: Aus dem *Zustandsgrafen* kann nun die *Zustandsfolgetabelle* ermittelt werden

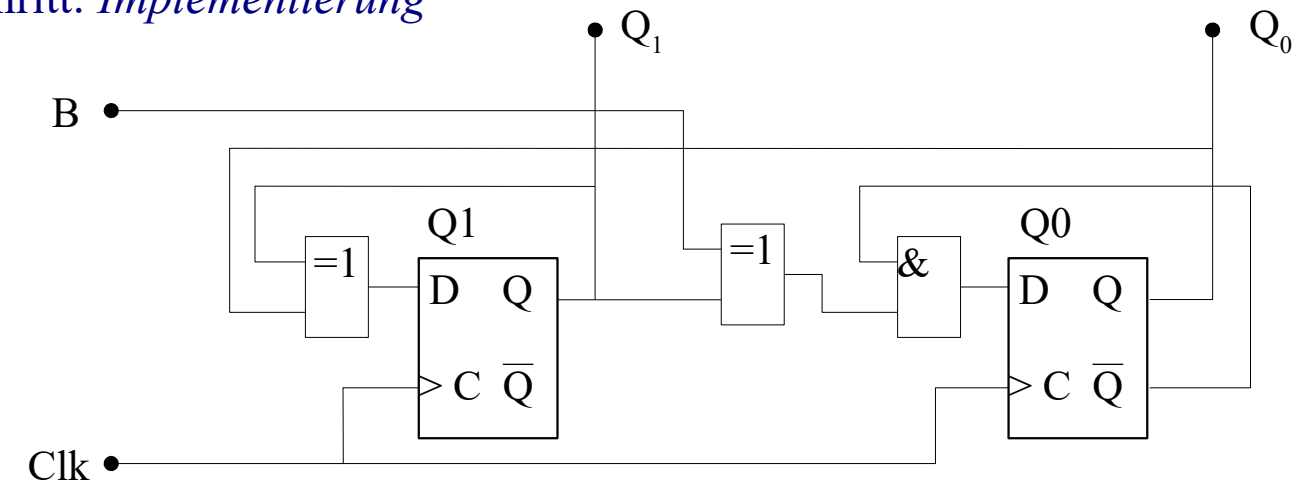
B	$Q_1[n]$	$Q_0[n]$		$Q_1[n+1]$	$Q_0[n+1]$	
0	0	0	*Grün*	0	0	*Grün*
0	0	1	*Gelb*	1	0	*Rot*
0	1	0	*Rot*	1	1	*Rot-Gelb*
0	1	1	*Rot-Gelb*	0	0	*Grün*
1	0	0	*Grün*	0	1	*Gelb*
1	0	1	*Gelb*	1	0	*Rot*
1	1	0	*Rot*	1	0	*Rot*
1	1	1	*Rot-Gelb*	0	0	*Grün*

- *Beispiel Ampelsteuerung*
  - 4. Schritt: Bestimmung der *Zustandsübergangsfunktionen*

B	$Q_1[n]$	$Q_0[n]$		$Q_1[n+1]$	$Q_0[n+1]$	
0	0	0	*Grün*	0	0	*Grün*
0	0	1	*Gelb*	1	0	*Rot*
0	1	0	*Rot*	1	1	*Rot-Gelb*
0	1	1	*Rot-Gelb*	0	0	*Grün*
1	0	0	*Grün*	0	1	*Gelb*
1	0	1	*Gelb*	1	0	*Rot*
1	1	0	*Rot*	1	0	*Rot*
1	1	1	*Rot-Gelb*	0	0	*Grün*

$$Q_1[n+1] = Q_0[n] \oplus Q_1[n]$$
$$Q_0[n+1] = \overline{Q_0} \cdot (B \oplus Q_1)$$

  - 5. Schritt: *Implementierung*



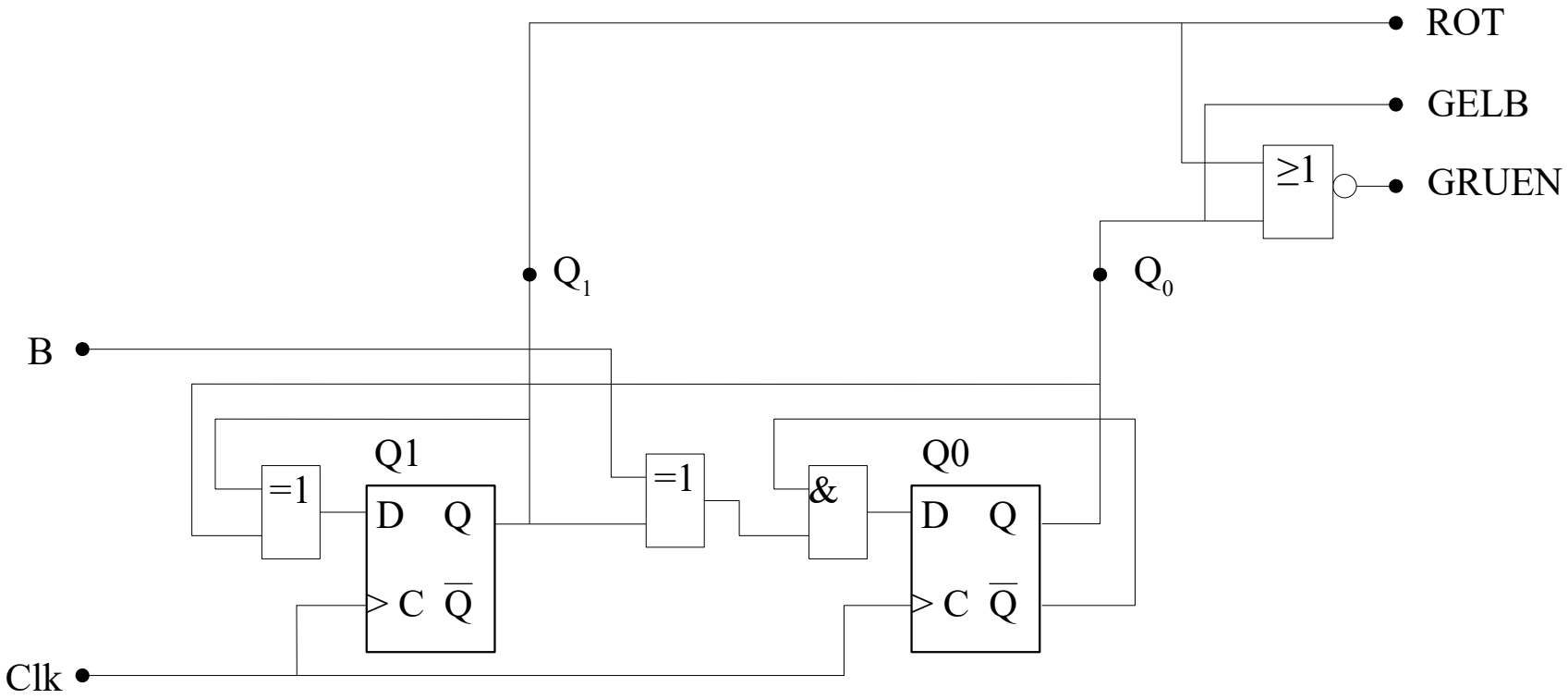
- *Beispiel Ampelsteuerung*
  - 6. Schritt: Implementierung der Ausgabefunktion

$Q_1[n]$	$Q_0[n]$		ROT	GELB	GRUEN
0	0	*Grün*	0	0	1
0	1	*Gelb*	0	1	0
1	0	*Rot*	1	0	0
1	1	*Rot-Gelb*	1	1	0

$$ROT = Q_1$$

$$GELB = Q_0$$

$$GRUEN = \overline{Q_1 + Q_0}$$



Entwurfs- und Simulationssprache: VHDL

- Beschreibungssprache für digitale Hardware
  - Ähnlich C oder Pascal
  - VHDL = **V**HSIC **H**ardware **D**escription **L**anguage
  - VHSIC = **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit
  - Hintergründe:
    - Zwischen 1970 und 1980 vom US Department of Defense entwickelt
    - Seit 1987 als IEEE-Standard 1067 genormt, Überarbeitete Version 1993
    - Als universelle Beschreibungssprache weitgehend etabliert
- Eigenschaften:
  - Modellierung räumlicher Strukturen und des zeitlichen Verhaltens von Schaltkreisen
  - VHDL ermöglicht Design, Simulation und Implementation digitaler Systeme
    - Beispiel: Prozessoren, ASIC, FPGA und deren Umgebung
    - System kann in allen Design-Stufen simuliert werden
    - Implementation der VHDL-Modell in Hardware durch Code-Synthese möglich