HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Technische Grundlagen der Informatik

Schaltwerke

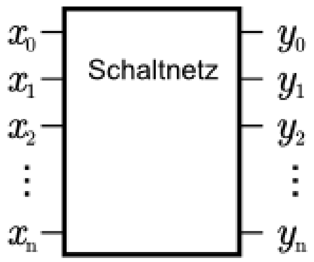
Prof. Dr.-Ing. Benjamin Weinert

Gliederung

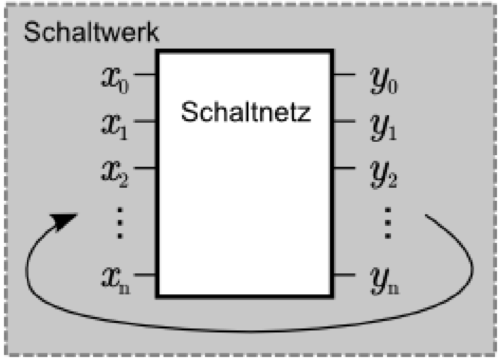
- Elementare Schaltwerke
  - Flip-Flops
  - Register
  - Schieberegister
  - Zähler
- Endliche Automaten

Sequentielle Schaltungen

- Bisher haben wir Schaltungen ohne Rückkopplungen betrachtet
  → **Schaltnetze**
  - Werte an den Ausgängen sind nur abhängig von den Eingängen
  - Solche Schaltungen verhalten sich immer gleich (sind zustandslos) und sind durch ihre Schaltfunktion eindeutig beschrieben
  - Aber: Es können keine Werte gespeichert werden



- Wir betrachten nun Schaltungen mit Rückkopplungen
  → **Schaltwerke**
  - Werte an den Ausgängen sind abhängig von den Eingängen und den vorherigen Ausgangswerten
  - Das Zeitverhalten muss genau betrachtet werden
  - Die vorherigen Ausgangswerte können als Zustand der Gatter interpretiert werden
    - Abhängig vom Zustand verhalten sich die Gatter anders (zustandsabhängige Schaltfunktion)
  - **Es wird möglich, Zustände zu speichern**

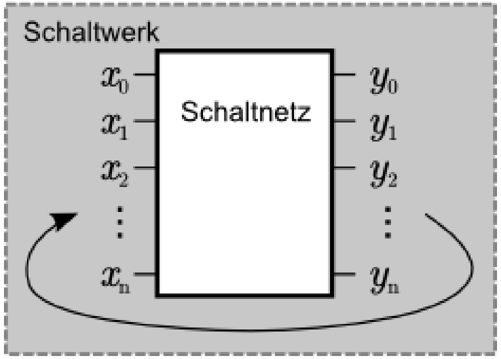


Zustandsspeicher

- Zur Speicherung vergangener Information ist ein Zustandsspeicher erforderlich

- Einfachste Form dieses Zustandsspeicher:

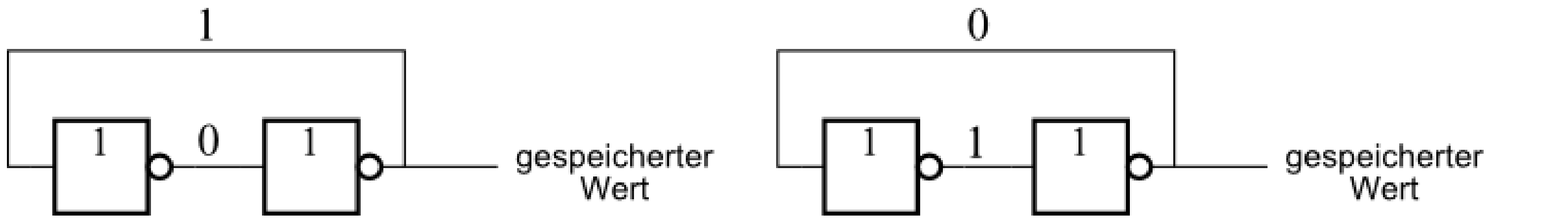
Rückkopplung



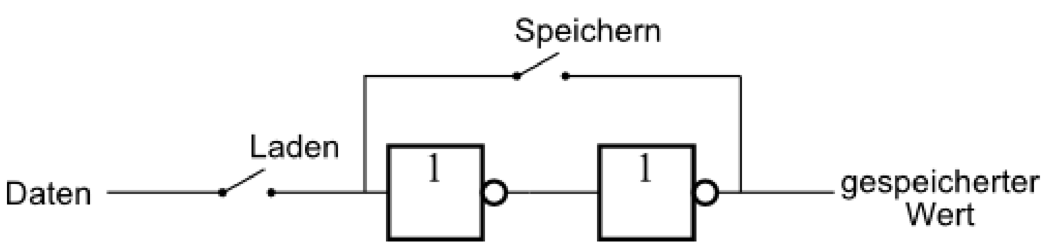
- Durch die Rückkopplung lassen sich die in den Eingangsvariablen nicht mehr repräsentierten Informationen wieder am Eingang zur Verfügung stellen
  - Die aus den bekannten Logikgattern konstruierten Speicherelemente erlauben uns eine Information persistent zu speichern!

Speicher mit rückgekoppelten Gattern

- Zwei Inverter bilden eine statische Speicherzelle („Speicherschleife“)
- Der Wert bleibt erhalten bis die Versorgungsspannung abgeschaltet wird

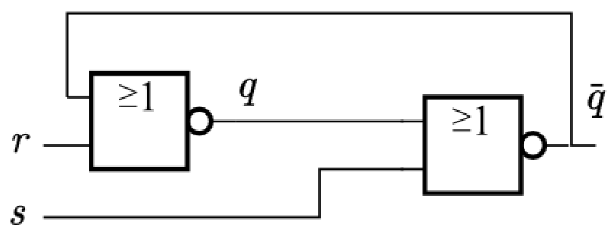


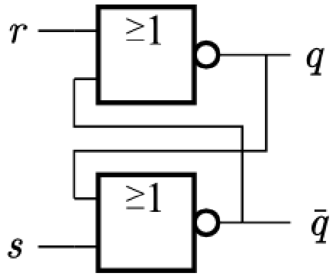
- Wie kann ein neuer Wert gespeichert werden?
  - Öffnen des Rückkopplungspfads
  - Laden des neuen Datenwertes



Speicher mit rückgekoppelten Gattern

- Rückgekoppelte NOR-Gatter mit den beiden Eingangsleitungen r = Reset und s = Set und den beiden Ausgängen q und ¬q
  - rs = 00, rs = 01, rs = 10 erlaubt
    - rs = 11 nicht erlaubt
  - Speicherfähigkeit: Bei r = 0 und s = 0 bleibt der aktuelle Wert q gespeichert (entspricht der Inverter-Schaltung auf der vorherigen Folie)
  - Der gespeicherte Wert q kann
    - mit Reset r = 1 auf q= 0 und
    - mit Set s = 1 auf q= 1 gesetzt werden

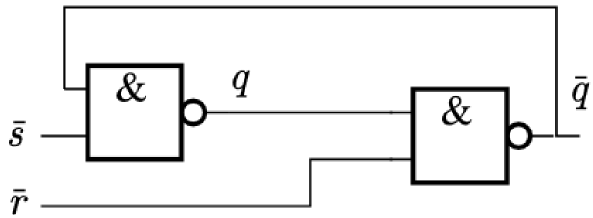


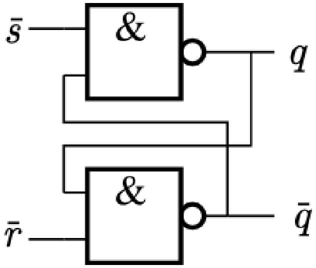


Speicher mit rückgekoppelten Gattern

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

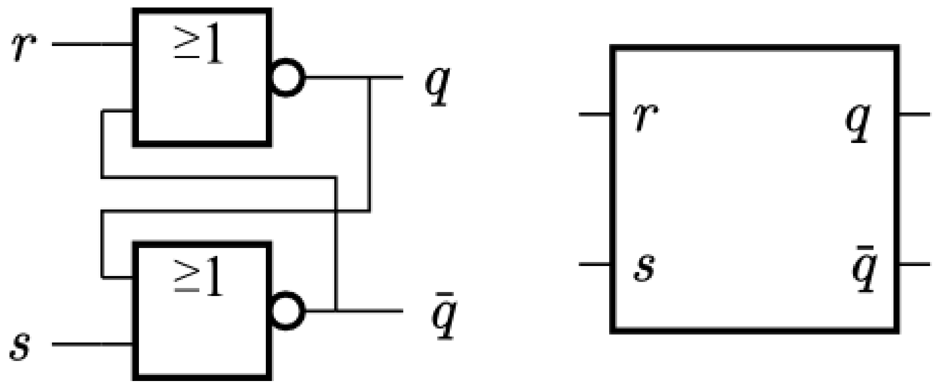
- Rückgekoppelte NAND-Gatter
  - Bei ¬r = 1 und ¬s = 1 bleibt der aktuelle Wert gespeichert
- Der gespeicherte Wert q kann
  - mit Reset ¬r = 0 auf q = 0 und
  - mit Set ¬s = 0 auf q = 1 gesetzt werden





Asynchrones RS-Flip-Flop

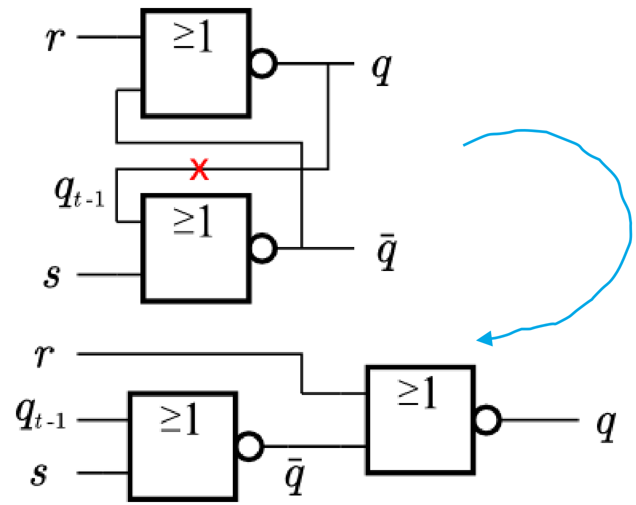
- Diese Schaltung wird asynchrones RS-Flipflop genannt
- Das RS-Flipflop kann 1 Bit speichern
- Für ein **asynchrones** RS-Flipflop wird auch folgendes Ersatzschaltbild verwendet



Asynchrones RS-Flip-Flop

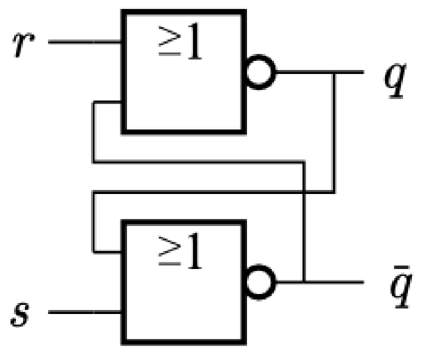
- Wird der **vorherige Ausgangswert** von q mit q_{t-1} bezeichnet, kann folgende Wahrheitstafel aufgestellt werden

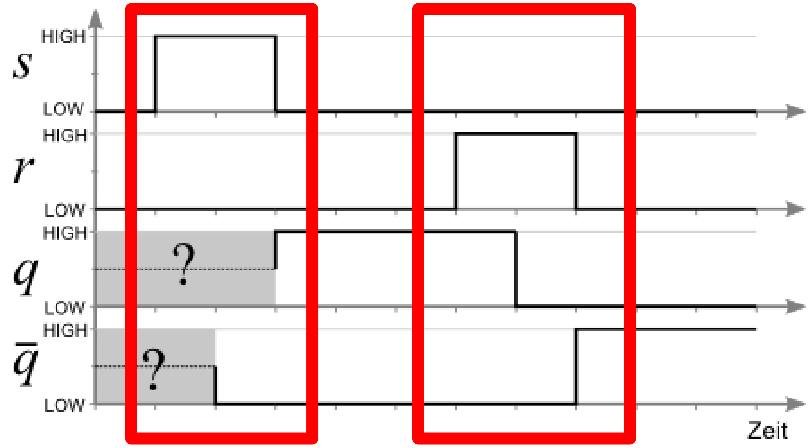
r	s	q_{t-1}	q	Funktion
0	0	0	0	halten
0	0	1	1	halten
0	1	0	1	setzen
0	1	1	1	setzen
1	0	0	0	rücksetzen
1	0	1	0	rücksetzen
1	1	0	x	illegal
1	1	1	x	illegal



- D.h. der Zustand des **vorherigen** Ausgangswertes q_{t-1} hat bei der Schaltung keinen Einfluss auf die **Funktion**
  - Daher kann die Wahrheitstabelle auch reduziert dargestellt werden (**ohne** q_{t-1})

Zeitverhalten eines asynchronen RS-Flip-Flops



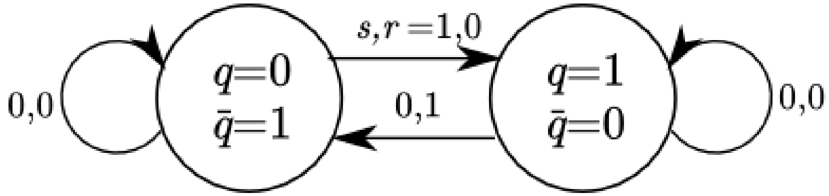


Setzen Rücksetzen

- Die Eingangsbelegung r = 1 und s = 1 sollte vermieden werden
- Beim gleichzeitigen Wechsel auf 0 fängt das System sonst an zu schwingen
  - Vgl: [Simulation in Amilosim (Uni Marburg)]

Zustandsdiagramm eines asynchronen RS-Flip-Flops

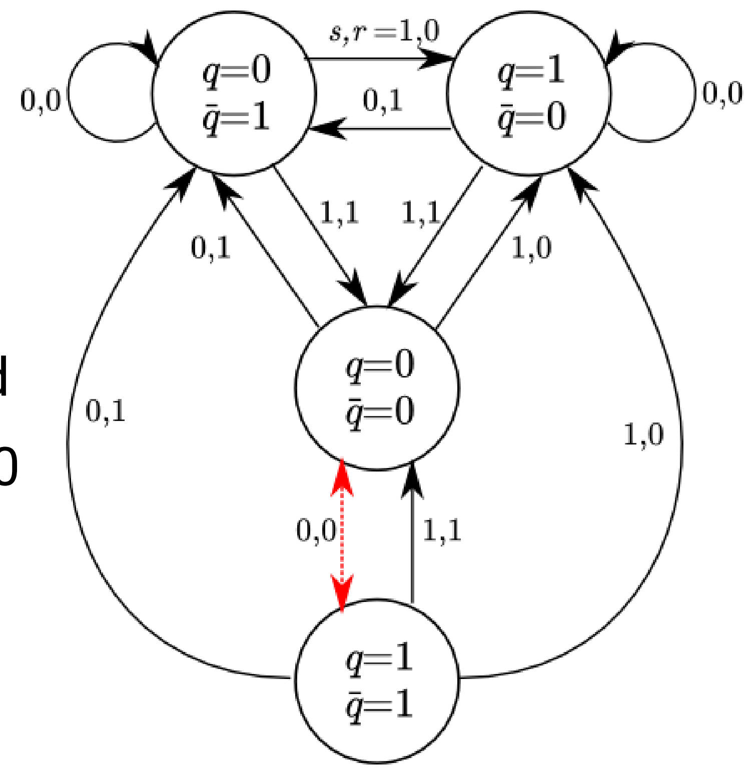
- Wird ein RS-Flipflop **ohne** die illegale Eingangsbelegung r = 1 und s = 1 betrieben, gibt es zwei Zustände, in denen sich das System befinden kann:
  - Zustand 1: q = 0 und ¬q = 1
  - Zustand 2: q = 1 und ¬q = 0
- Im Zustandsdiagramm sind die Zustände mit Kreisen markiert
  - Abhängig von den Eingangsvariablen r und s können Zustandswechsel auftreten (Pfeile im Zustandsdiagramm)



- Das Verhalten des Flipflops an den Ausgängen ist **nicht abhängig** vom Zustand, sondern von den Eingangsvariablen!

Zustandsdiagramm eines asynchronen RS-Flip-Flops

- Wird ein RS-Flipflop mit der illegalen Eingangsbelegung r = 1 und s = 1 betrieben, gibt es, unter Berücksichtigung des Zeitverhaltens, vier Zustände, in denen sich das System befinden kann:
  - Zustand 1: q = 0 und ¬q = 1
  - Zustand 2: q = 1 und ¬q = 0
  - **Zustand 3: q = 0 und ¬q = 0**
  - **Zustand 4: q = 1 und ¬q = 1**
- Das Verhalten des Flipflops an den Ausgängen ist jetzt abhängig vom Zustand
- Beim gleichzeitigen Wechsel auf r = 0, s = 0 im Zustand q = 0, ¬q = 0 fängt das System an zu schwingen



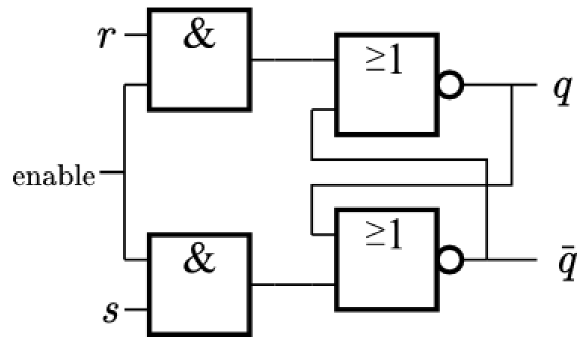
Probleme asynchroner Schaltwerke

- Asynchrone Schaltwerke arbeiten **ohne** einen zentralen Takt
  - Sie reagieren sofort auf jede Änderung der Eingangs- und Zustandsvariablen
  - Sie sind sehr störempfindlich

- Hasardfehler in den Übergangsschaltnetzen
  - Asynchrone Schaltwerke reagieren darauf sehr empfindlich
  - Hasardfehler können ebenfalls falsche Zustandsübergange verursachen
    - „Schwingung" entsteht durch eine Signalverzögerung die im Beispiel durch die beiden NOR-Gatter als auch durch die Verbindungsleitungen zwischen den Gattern verursacht wird

Gesteuertes RS-Flip-Flop

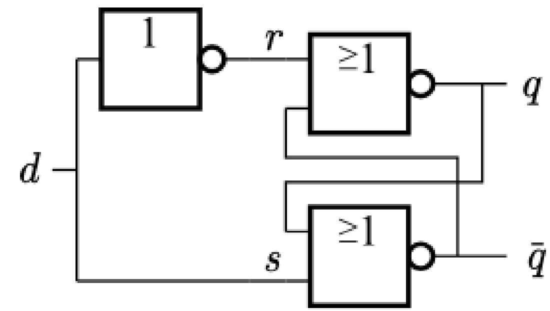
- Der "Enable"-Eingang legt fest, ob r und s eine Rolle spielen
- Änderungen am Eingang werden nur noch für enable = 1 übernommen

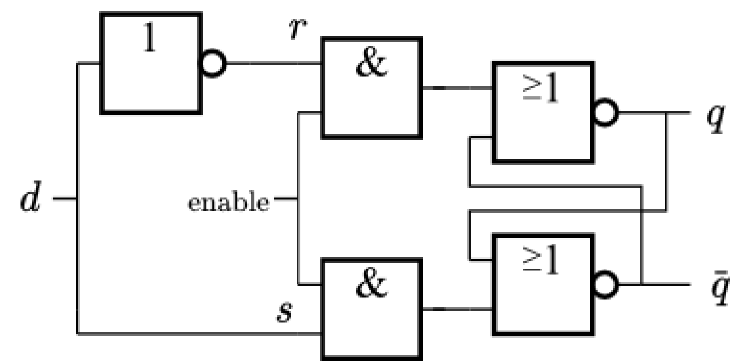


- Diese Maßnahme schützt das Flipflop vor eventuellen Störungen am Eingang so lange enable = 0
- Das Problem, dass die Schaltung in Schwingung geraten kann, löst die Maßnahme jedoch nicht
  - Wenn RS = 11 und enable = 1, gerät die Schaltung potentiell in Schwingung

Gesteuertes D-Flip-Flop

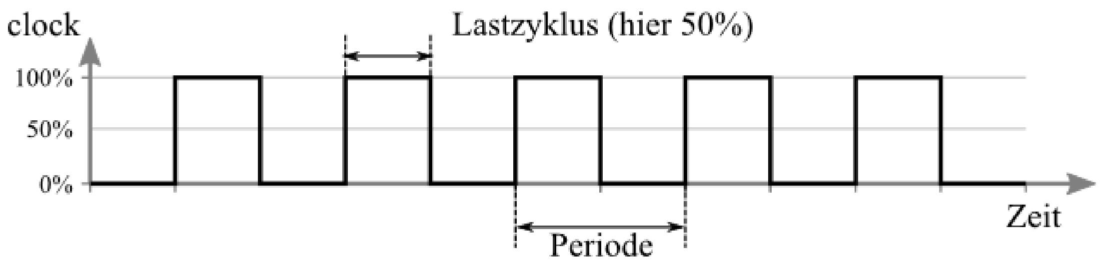
- Einfachste Lösung, die illegale Eingangsbelegung r = 1 und s = 1 zu verhindern
  - nur **ein** Eingangssignal **d** verwenden
- **Aber**: die Eingangsbelegung s = 0 und r = 0, die das Speichern des Wertes bewirkt, kann nicht mehr erzeugt werden
- Wird die Schaltung mit der "Enable"-Schaltung kombiniert, ist Speichern wieder möglich (wenn enable = 0).
- Diese Schaltung wird auch D-Flip-Flop genannt





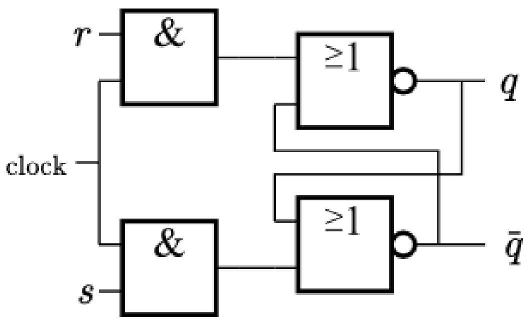
Synchrone Flip-Flops

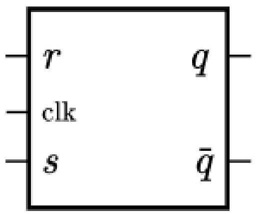
- Durch ein Taktsignal (engl. clock) kann eine synchrone Schaltung entworfen werden
  - Der Takt gibt dem System eine gemeinsame Zeitbasis
- Zweck bei Flipflops:
  - Solange warten, bis die Eingänge r und s stabil sind
  - Dann die beabsichtigten Änderungen zulassen (enable = 1)
  - Dies kann erreicht werden, indem das Taktsignal clock auf den enable-Eingang gelegt wird
- Taktsignale sind periodische Signale
  - Periodendauer: Zeit zwischen zwei gleichen Flanken
  - Lastzyklus: hier 50%, da Zeit für 1 gleich lang, wie für 0



Taktpegelgesteuerte Flip-Flops

- Erweiterung des **asynchronen** RS-Flip-Flops zu einem **synchronen** & **taktpegelgesteuerten** Speicherelement durch Ergänzung zweier AND-Gatter

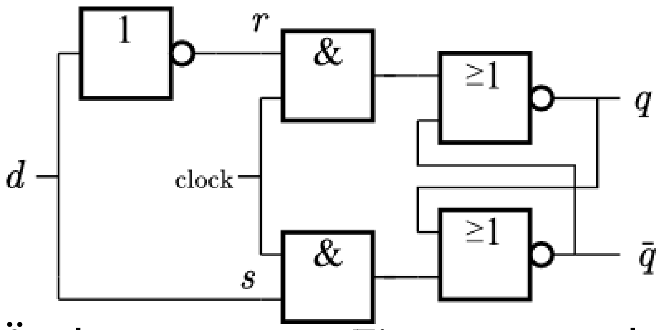


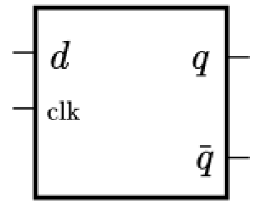


- Änderungen am Eingang werden nur während des Lastzyklus übernommen (d.h. wenn clock = 1)
  - „clock" wechselt im Takt zwischen 0 und 1
    - Die Zustandswechsel sind somit auf bestimmte Zeitintervalle festgelegt
    - Hinweis: Die meisten Schaltwerke arbeiten mit einem einzigen Taktsignal, dass alle synchronen Speicherelemente gemeinsam versorgt

Taktpegelgesteuerte Flip-Flops

- Synchrones D-Flip-Flop

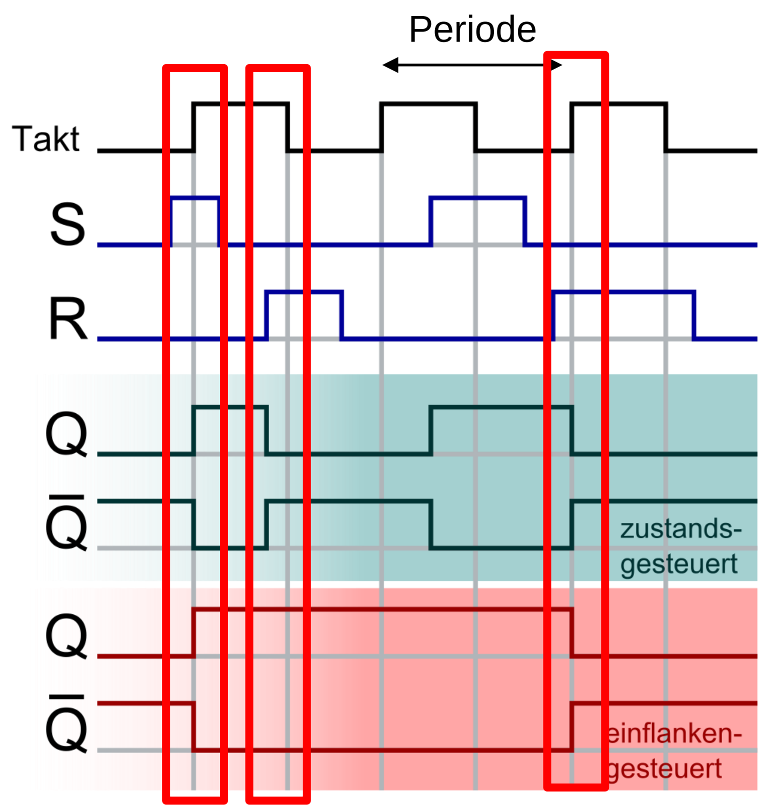




- Änderungen am Eingang werden nur während des Lastzyklus übernommen (d.h. wenn clock = 1)
- Ist während des Lastzyklus d = 1, wird q = 1 gesetzt und für den Rest der Periode gehalten
- Erst beim nächsten Lastzyklus kann sich q wieder ändern
  - sofern d dann = 0
- Das D-Flipflop realisiert damit eine Verzögerung (engl. "Delay") in einem Daten-Stream, daher der Name "D"-Flipflop
  - D: 1 0 1 0
  - CLK: ↑ ↑ ↑ ↑
  - Q: 1 0 1 0

Taktflankengesteuerte Flip-Flops

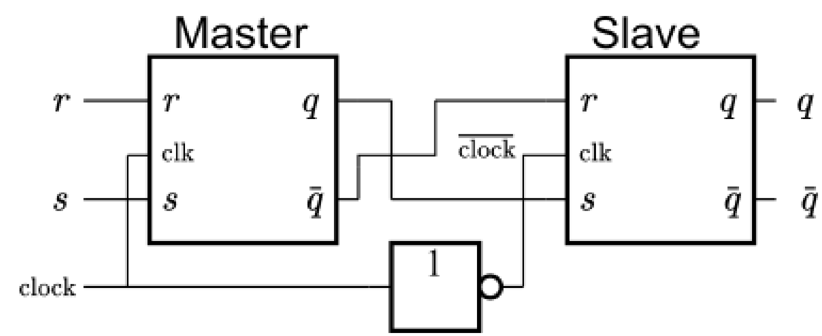
- Bisher waren die Flipflops takt**pegel**gesteuert (oder auch: taktzustandsgesteuert)
  - Es wird ein gewisser Grad der Synchronisation erreicht
  - Oftmals jedoch erforderlich, potentielle Zustandswechsel weiter einzuschränken und auf bestimmte Zeitpunkte zu begrenzen
- Häufig sinnvoller: takt**flanken**gesteuerte Flipflops
- Positiv taktflankengesteuert
  - Eingänge werden bei der **steigenden** Flanke abgetastet
  - Ausgänge ändern sich nach der steigenden Flanke
- Negativ taktflankengesteuert
  - Eingänge werden bei der **fallenden** Flanke abgetastet
  - Ausgänge ändern sich nach der fallenden Flanke



Taktflankengesteuertes Flip-Flop

Master-Slave-Flip-Flop

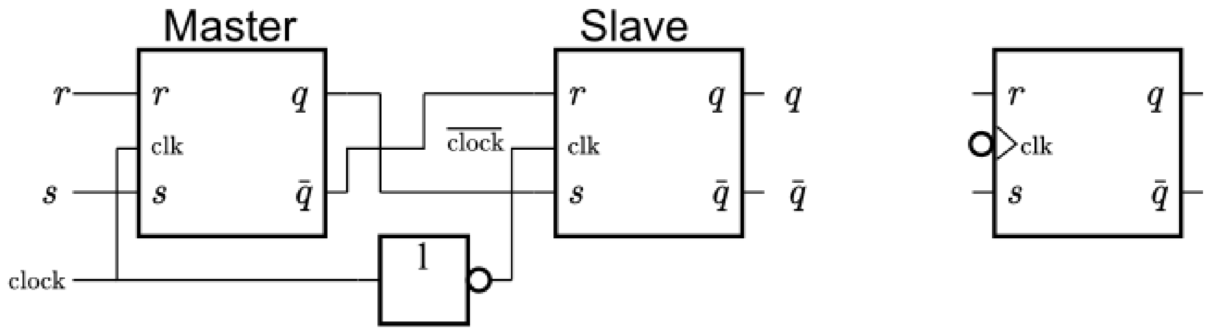
- Durch hintereinander Schalten von zwei takt**pegel**gesteuerten RS-Flipflops
  - entsteht ein takt**flanken**gesteuertes Flipflop
  - auch "Master-Slave Flipflop" genannt:
  - r und s werden bei steigender Flanke des clock-Signals vom **Master** eingelesen

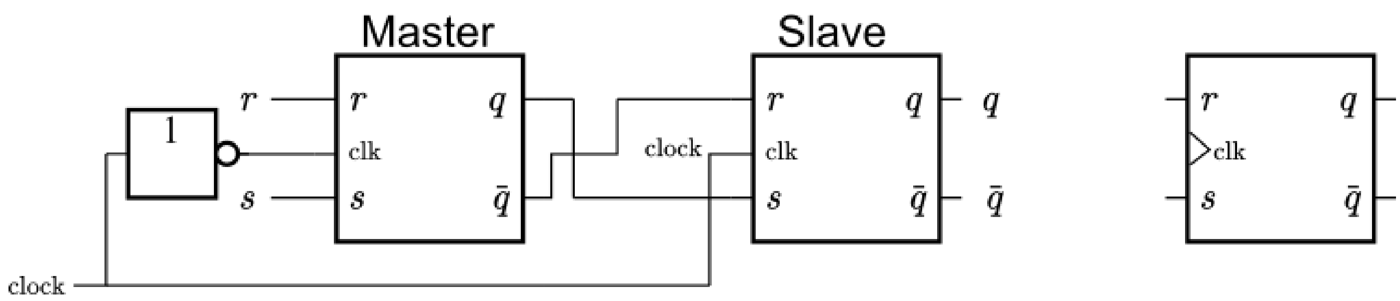


- Während sich die Ausgänge des Masters ändern, passiert beim Slave nichts, da dort das Taktsignal invertiert wurde (Eingang ¬clock = 0) und das Slave Flipflop damit deaktiviert ist
- Bei fallender Flanke des Taktsignals werden die Ausgänge des Masters vom Slave eingelesen, die Ausgänge des Slaves ändern sich
  - Eingang ¬clock = 1
- Eine Änderung der Ausgänge, und damit des Speicherwerts, ist bei diesem Master-Slave Flipflop damit **nur** bei einer **fallenden** Flanke möglich (egal zu welchem Zeitpunkt sich die Eingangssignale ändern)

Taktflankengesteuertes RS-Flip-Flop

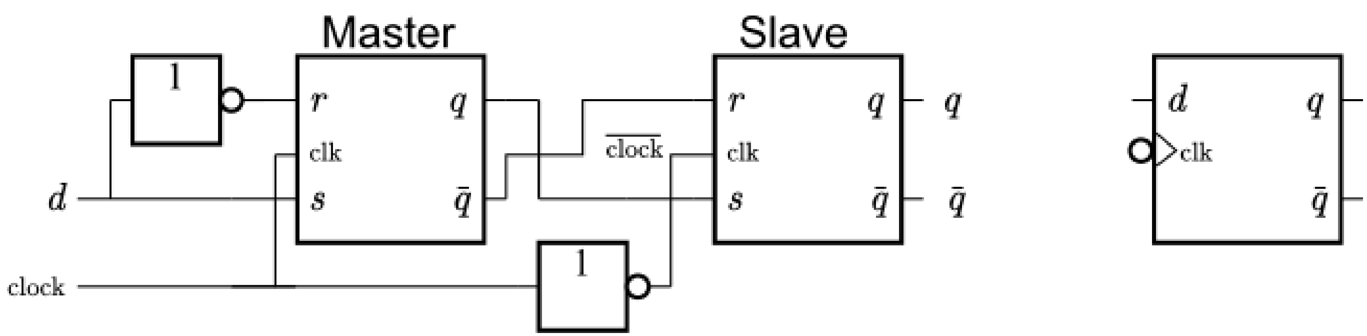
- Es werden die folgenden zwei Ersatzschaltbilder für das taktflankengesteuerte RS-Flipflop verwendet, je nachdem, ob sich die Ausgänge
  - bei **fallender** (Abb. **oben**)
  - oder **steigender** (Abb. **unten**) Flanke ändern

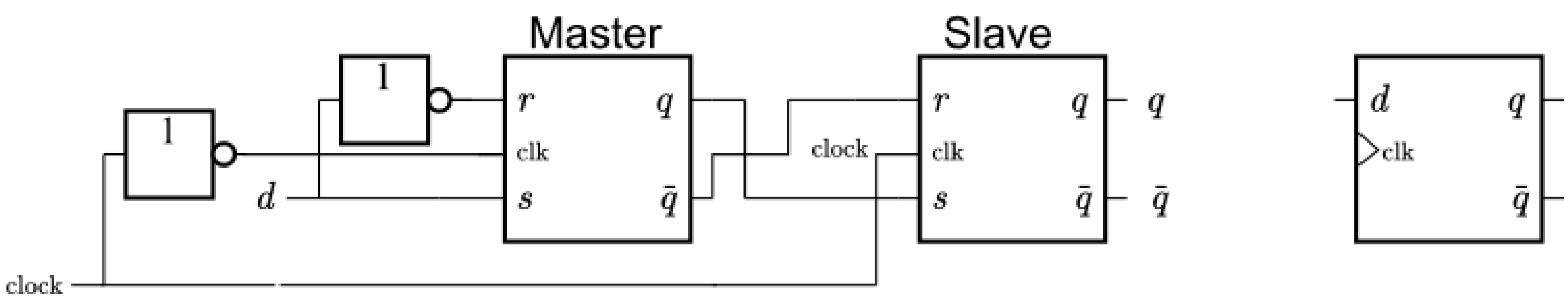




Taktflankengesteuertes D-Flip-Flop

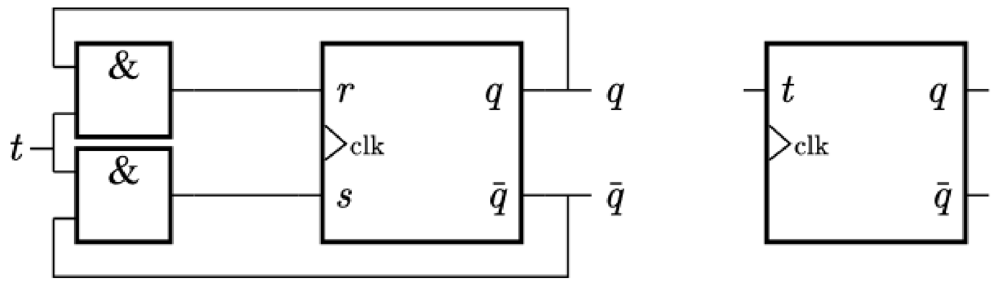
- Ein taktflankengesteuertes **RS**-Flipflop kann leicht in ein taktflankengesteuertes **D**-Flipflop umgewandelt werden
- Auch für das taktflankengesteuerte **D**-Flipflop werden Ersatzschaltbilder eingeführt





T-Flip-Flop

- Neben **RS-** und **D**-Flipflops gibt es auch **T**-Flipflops
- Diese können aus einem taktflankengesteuerten RS-Flipflop durch das Vorschalten zweier UND-Gatter erzeugt werden

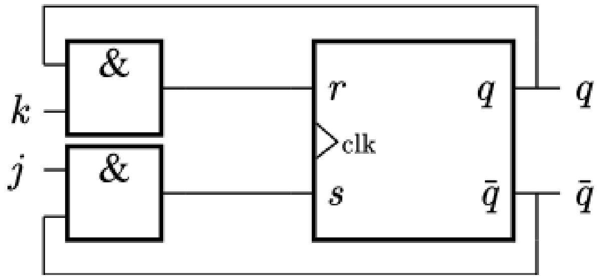


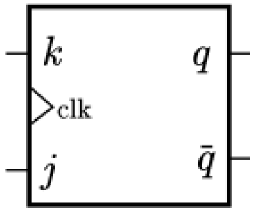
- Wie das D-Flipflop hat das T-Flipflop nur einen Eingang t
  - Ist t=1 ändert, sich der Ausgang q von 0 nach 1 bzw. von 1 nach 0
  - Der Ausgang wird also jeweils bei t = 1 umgeschaltet (engl. "toggle"), daher der Name T-Flipflop
    - t = 0 führt nicht zu einer Zustandsänderung am Ausgang
  - Ist das T-Flipflop einmal in einem stabilen Zustand, bleibt es stabil, da die beiden Ausgänge q und ¬q nicht gleichzeitig 1 sein können

JK-Flip-Flop

- Eine weitere Variante, die **universell** einsetzbar ist, ist das JK-Flipflop
- Dieses kann aus einem taktflankengesteuerten RS-Flipflop durch das Vorschalten zweier UND-Gatter erzeugt werden
  - Ist j = 1 und k = 1, entspricht das Verhalten dem eines T-Flipflop
  - Ansonsten dem Verhalten eines RS-Flipflops
  - D.h. die beim RS-Flipflop illegale Eingangsbelegung j = 1 und k = 1 bekommt eine eigene Funktionalität zugewiesen

k	j	q_{t-1}	q	Funktion
0	0	0	0	halten
0	0	1	1	halten
0	1	0	1	setzen
0	1	1	1	setzen
1	0	0	0	rücksetzen
1	0	1	0	rücksetzen
1	1	0	1	umschalten
1	1	1	0	umschalten





Zusammenfassung

- Flip-Flops zum Halten eines 1-Bit-Werts

- Asynchron vs. Synchron
- Taktflankengesteuert vs. Taktpegelgesteuert

- RS-Flip-Flops speichern Zustände
- D-Flip-Flops speichert Datenfolgen
- T-Flip-Flops flippen kontinuierlich den Wert an Ausgang q => für einen Zustandswechsel
- JK-Flip-Flops sind universelle Flip-Flops für alle vorgenannten Anwendungsfälle

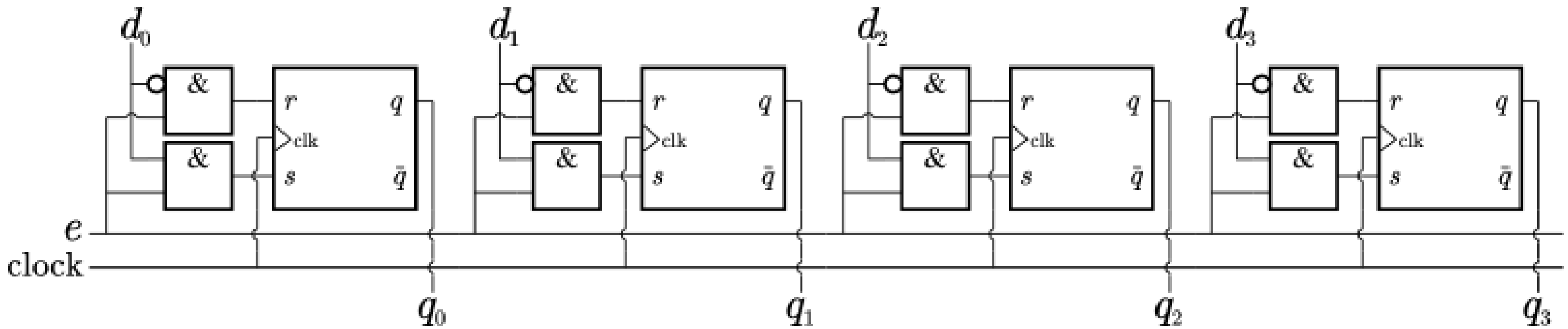
Register

- Ein Register besteht aus parallel angeordneten n Speicherelementen
- Register dienen der Speicherung und Manipulation von vollständigen Datenwörtern
- Eine wichtige Kenngröße ist die Anzahl n der Speicherelemente,
  - z.B. 8-, 16-, 32-, 64-, 128-Bit Register
- Im Folgenden werden drei verschiedene Registertypen vorgestellt
  - Auffangregister
  - Schieberegister
  - Universalregister

Auffangregister

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

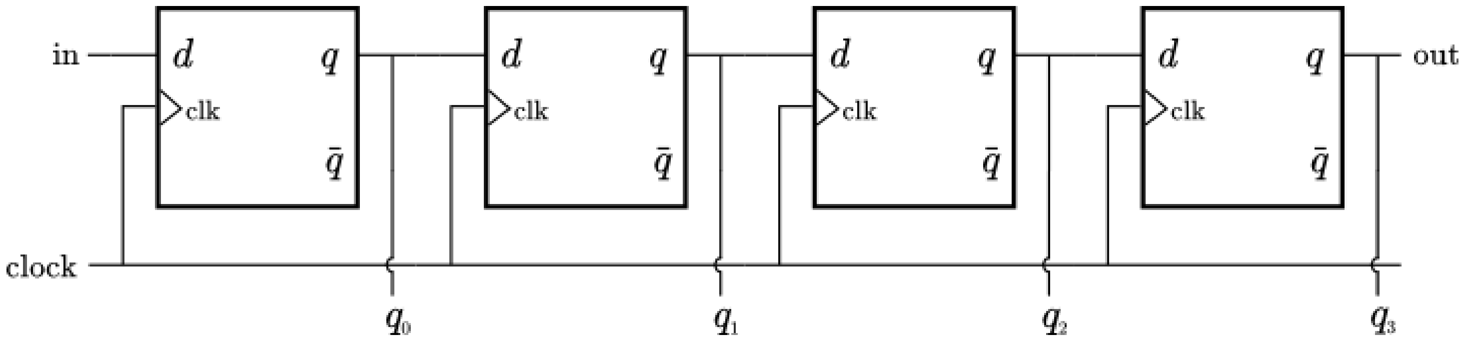
- Das Auffangregister dient zur Zwischenspeicherung von Datenworten
- Ein n-Bit Auffangregister kann aus einer parallelen Anordnung von n Flipflops bestehen, die jeweils 1-Bit speichern
- Beispiel 4-Bit Register mit taktflankengesteuerten RS-Flipflops
  - mit einer gemeinsamen Taktclock und
  - gemeinsamen Enable-Signal e



  - Wenn **e = 1** ist, werden die **Datenbits d_i** parallel mit steigender Flanke in das Register **übernommen**
  - An den Ausgängen **q_i** kann das Datenwort **ausgelesen** werden
  - Wenn **e = 0** ist, werden die Ausgänge konstant gehalten
    - d.h. das Datenwort ist gespeichert

Schieberegister

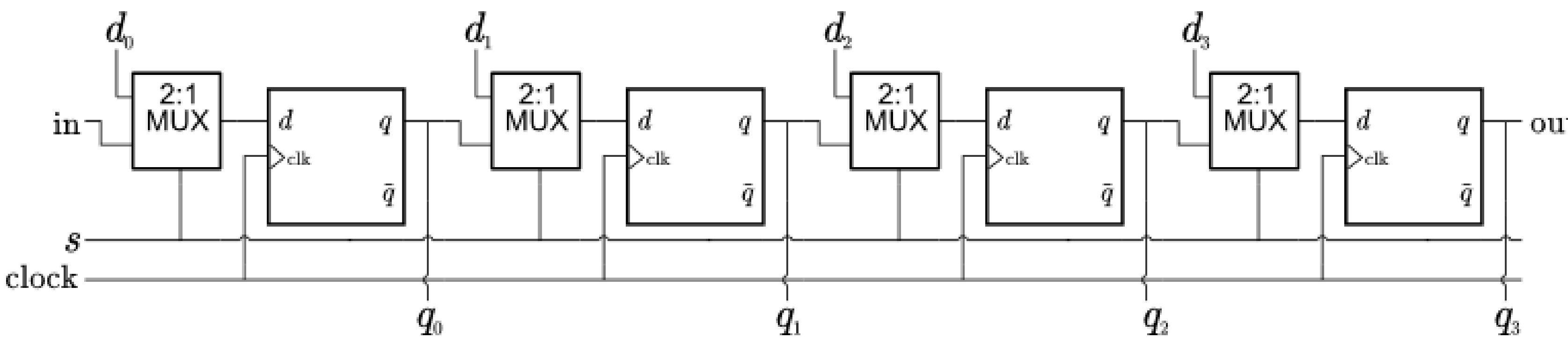
- Ein n-bit Schieberegister hat nur einen Eingang *in* aber *n* parallele Ausgänge
  - Mit jedem Takt wird der Eingang *in* abgefragt und der Wert im ersten Flipflop gespeichert
  - Der Ausgang eines Flipflops ist jeweils mit dem Eingang des nächsten Flipflops verbunden
  - Mit jedem Takt wird die Information ein Register weitergeschoben (daher der Name "Schieberegister")
- Beispiel 4-Bit Registers mit taktflankengesteuerten D-Flipflops



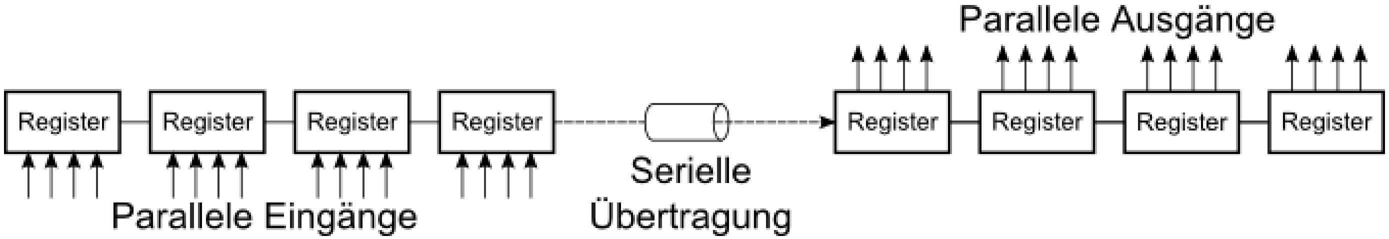
- Eine wichtige Anwendung ist das Umsetzten eines **seriellen** Datenstroms in Datenwörter der Länge n

Schieberegister

- Durch Vorschalten eines 2-zu-1 Multiplexers kann das Schieberegister so erweitert werden, dass
  - entweder eine Schiebeoperation ausgeführt wird (Steuerleitung s=1)
  - oder Daten parallel geladen werden (Steuerleitung s=0)



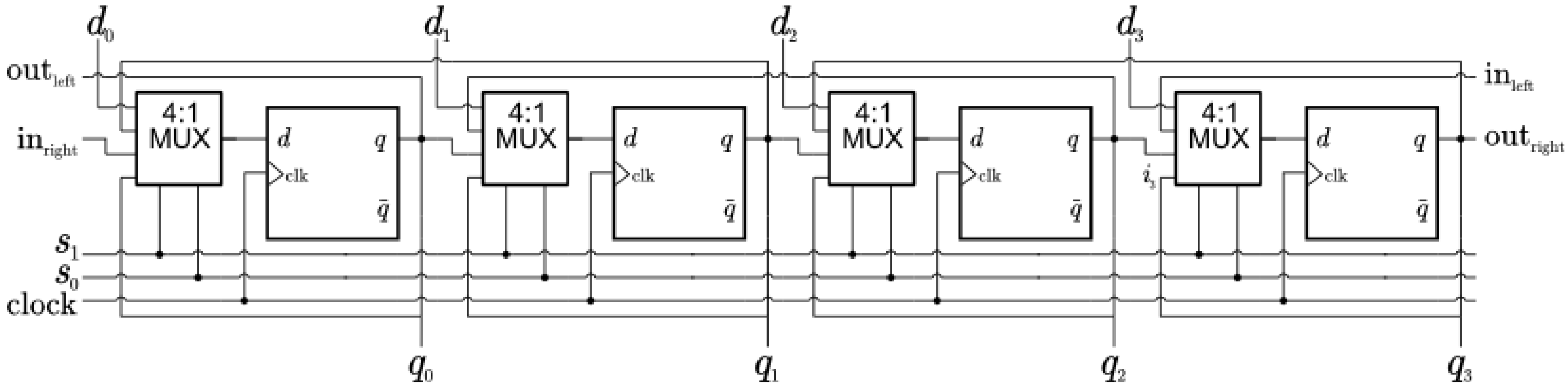
- Anwendung: Serielle Übertragung von Datenströmen können parallel ausgelesen werden



Universalregister

- Das Universalregister vereint die Funktionalität des Auffang- und des Schieberegisters
- Beispiel 4-Bit Universalregister
  - besteht aus 4 taktflankengesteuerten D-Flipflops
  - mit jeweils einem vorgeschalteten 4-zu-1 Multiplexer
  - Mit den gemeinsamen Steuerleitungen s_0 und s_1 der Multiplexer kann die gewünscht Funktion ausgewählt werden (siehe Tabelle)

s_1	s_0	Funktion
0	0	laden
0	1	links schieben
1	0	rechts schieben
1	1	halten



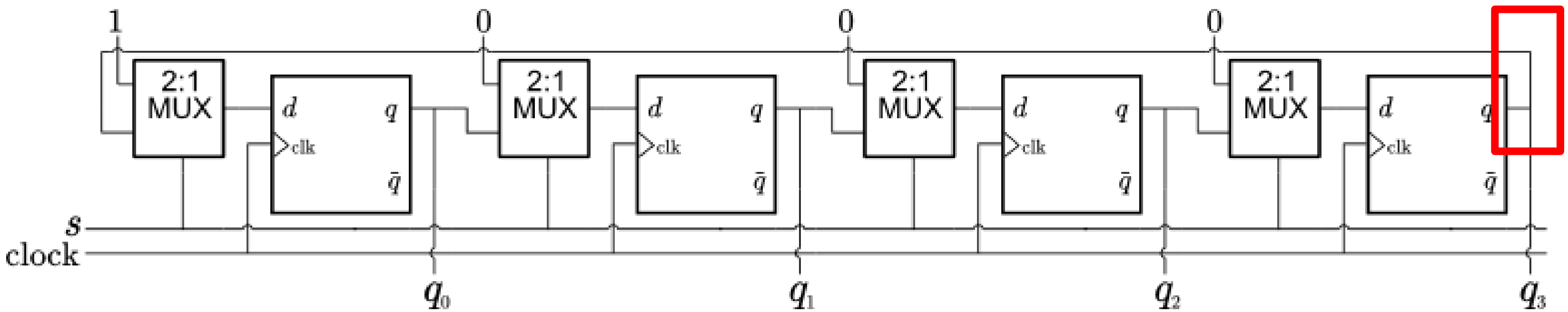
Zähler

- Mit Registern lassen sich leicht Zähler aufbauen
- Zähler unterscheiden sich in den folgenden Eigenschaften
  - Schrittlänge: Erhöhung des Zählerstands um eins oder um einen variablen Wert
  - Implementierung: Aufbau des Zählers / Asynchrone oder Synchrone Implementierung
  - Zählrichtung: Unidirektionale Zähler (Vorwärts- oder Rückwärtszähler), Bidirektionale Zähler
  - Zahlenformat: In welchem Format wird die Zahl repräsentiert (z.B. **B**inary **C**oded **D**ecimal)

- Im Folgenden werden drei Beispiele gezeigt
  - Ringzähler
  - Johnson-Zähler
  - Binärzahler

Ringzähler

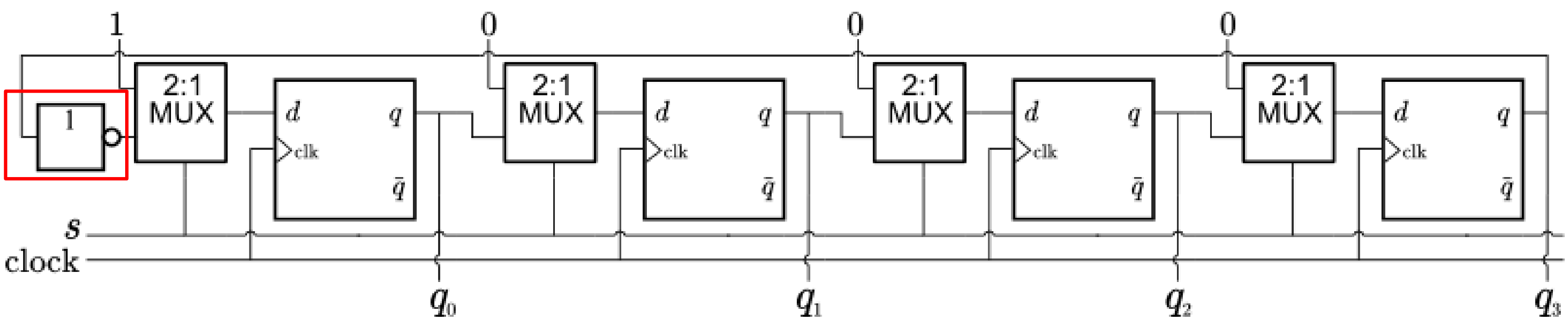
- Zur Realisierung eines Ringzählers kann ein **rückgekoppeltes Schieberegister** verwendet werden



- Ein 4-Bit Ringzähler erzeugt z.B. als Ausgabe die Folge: 1000, 0100, 0010, 0001, 1000, 0100, ...
  - Es ist also jeweils nur ein Ausgang 1, die anderen 0
  - Das Schieberegister im Beispiel wird mit 1000 initialisiert (Steuerleitung s = 0)
  - Anschließend wird die 1 nach rechts durch die Flipflops geschoben (Steuerleitung s = 1)
- Ein Ringzähler kann u.a. eingesetzt werden, um Aktionen nacheinander auszuführen. Dazu kann jeder Ausgang q_i mit einer Aktion verknüpft werden, die ausgeführt wird, sobald der Ausgang 1 ist

Johnson-Zähler

- Ebenfalls ein Ringzähler
- Beim Johnson-Zähler (auch Möbius-Zähler genannt) wird der rückgekoppelte Wert **invertiert**
- Dadurch ergibt sich bei Initialisierung mit 1000 die Folge: 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000, 1000, ...



Binärzähler

- Bei einem Binärzähler werden aufsteigende Binärzahlen erzeugt (**hier niederwertigstes Bit links**)
- Es soll also die Folge entstehen:
  0000, **1**000, **0**100, **1**100, **0**010, **1**010, **0**110, **1**110
  0001, **1**001, **0**101, **1**101, **0**011, **1**011, **0**111, **1**111
  0000, ...
- Ein Binärzahler kann ebenfalls mit Flipflops realisiert werden
  - allerdings werden einige weitere Logikbausteine benötig
  - Das niederwertigste Bit q_0 wechselt mit jedem Takt, dies kann mit einem Inverter realisiert werden

$$q_0^{t+1} = \neg q_0^t$$

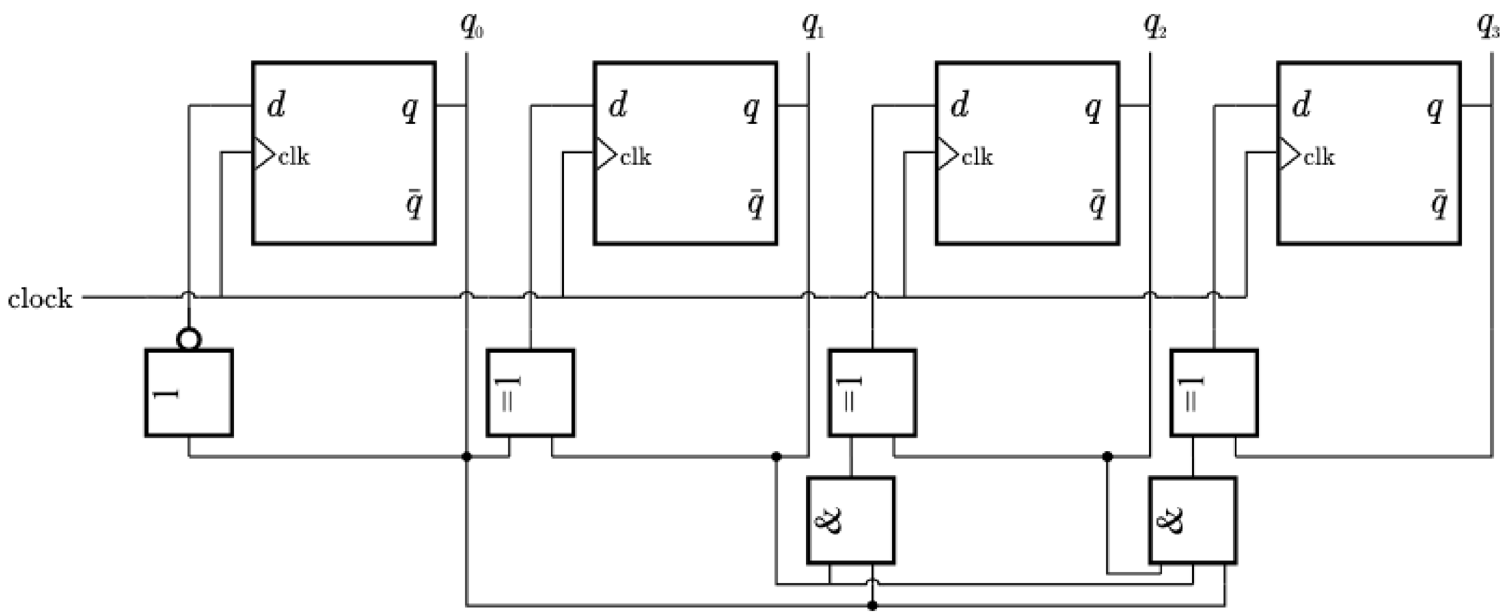
Für q_1 gilt: $q_1^{t+1} = q_1^t \nLeftrightarrow q_0^t$

Für q_2 gilt: $q_2^{t+1} = q_2^t \nLeftrightarrow (q_1^t \wedge q_0^t)$

Für q_3 gilt: $q_3^{t+1} = q_3^t \nLeftrightarrow (q_2^t \wedge q_1^t \wedge q_0^t)$

Binärzähler

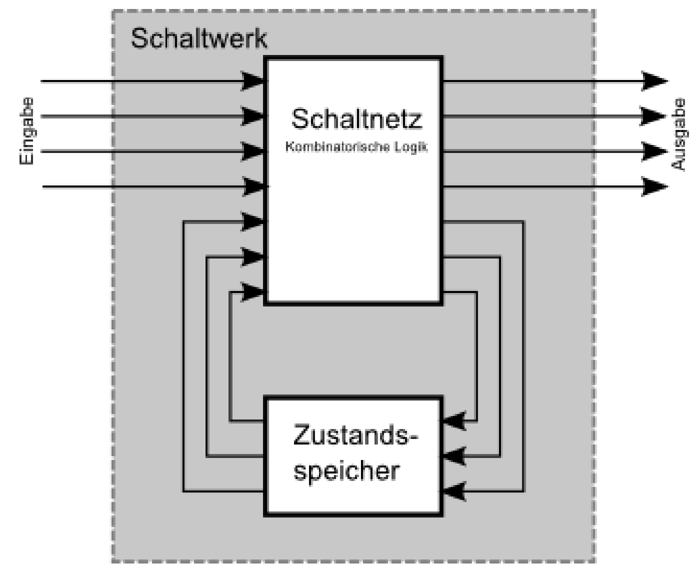
$$q_0^{t+1} = \neg q_0^t$$
$$q_1^{t+1} = q_1^t \nleftrightarrow q_0^t$$
$$q_2^{t+1} = q_2^t \nleftrightarrow (q_1^t \wedge q_0^t)$$
$$q_3^{t+1} = q_3^t \nleftrightarrow (q_2^t \wedge q_1^t \wedge q_0^t)$$



Vom FlipFlop zum Schaltwerk

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Wir haben gesehen, dass durch Rückkopplung Speicherelemente (Flip-Flops, Register) realisiert werden können
  - Speicherung eines Zustands
- Folglich kann ein **Schaltwerk** als Kombination aus einem **Schaltnetz und** einem **Zustandsspeicher** dargestellt werden
- Die Ausgabe des Schaltwerks kann abhängig vom aktuellen Zustand sein
- Der Zustand kann sich in Abhängigkeit von den Eingabewerten ändern
- Zur Modellierung der zustandsabhängigen Schaltfunktion können *endliche Automaten* verwendet werden
  - Endliche Automaten bieten den passenden Beschreibungsformalismus zur Modellierung von Schaltwerken!
  - Davon kann ein konkreter Schaltungsentwurf abgeleitet werden



Endliche Automaten

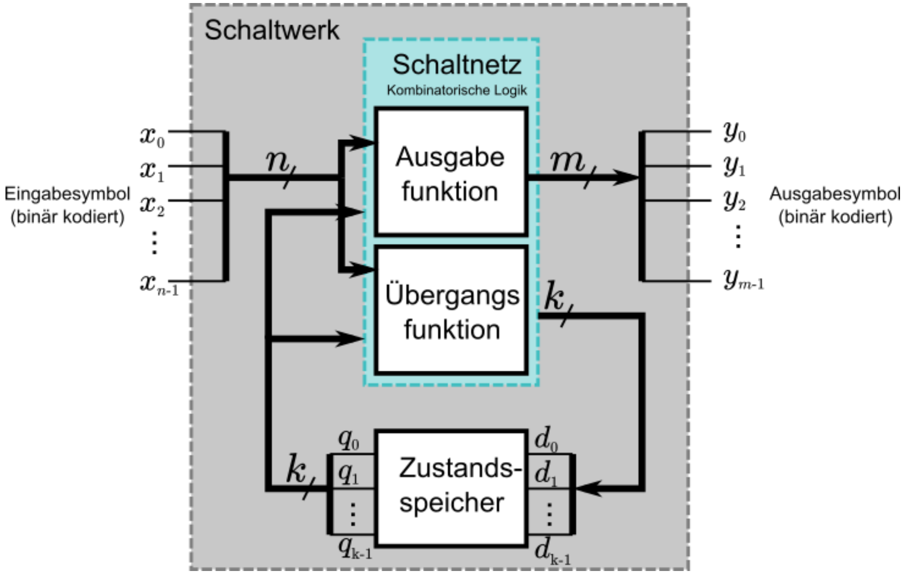
- Ein endlicher Automat „übersetzt" Eingabezeichen in eine Folge von Ausgabezeichen. Dazu verfügt ein solcher Automat intern über eine endliche Menge von Zuständen und Zustandsübergangsregeln
  - Bei Verarbeitung eines Eingabezeichen wird unter Berücksichtigung des aktuellen Zustands das Ausgabezeichen als auch den Folgezustand berechnet.
- Ein endlicher Automat (engl. Finite State Machine, FSM) ist definiert durch
  - eine endliche Menge A von Eingabesymbolen $a_i \in A$ (Alphabet)
  - eine endliche Menge S von Zuständen $s_i \in S$
  - einen Anfangszustand $s0 \in S$
  - eine Zustandsübergangsfunktion $\delta : S \times A \rightarrow S$
- Des weiteren kann er umfassen
  - eine endliche Menge B von Ausgabesymbolen $b_i \in B$
  - eine Ausgabefunktion $\lambda : S \times A \rightarrow B$
- Bei deterministischen Automaten erfolgen die Zustandsübergänge deterministisch (=nicht zufällig)
- Endliche Automaten können durch Zustandsübergangsgraphen dargestellt werden

Automatentypen (Huffmann-Normalform)

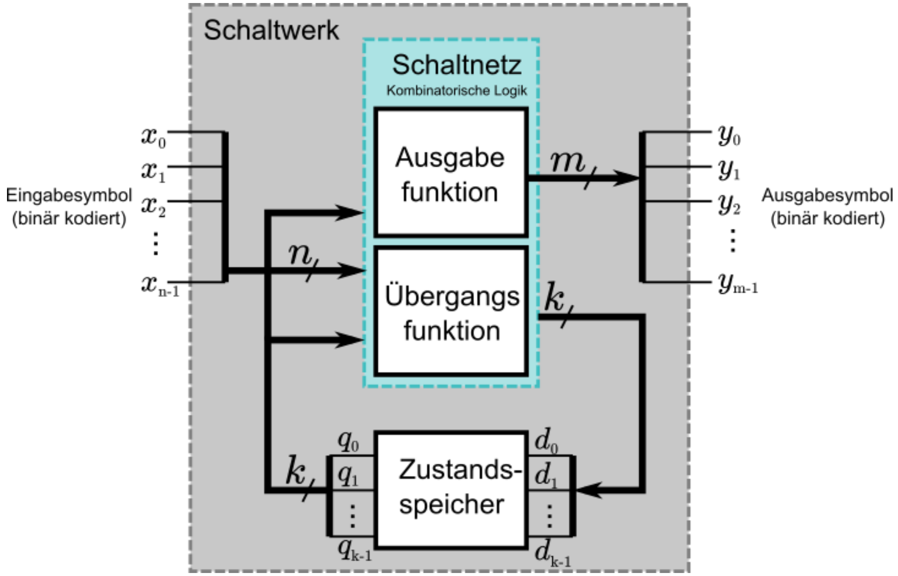
HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Mealy-Automaten
  - Ausgabefunktion λ: S × A → B ist vom aktuellen Zustand **und** der Eingabe abhängig
- Moore-Automaten
  - Ausgabefunktion λ: S → B ist nur vom aktuellen Zustand abhängig

Mealy-Automat

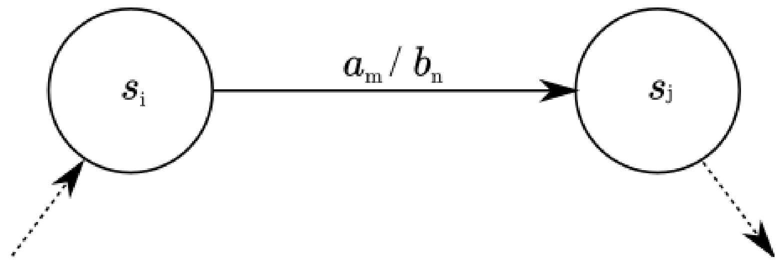


Moore-Automat



Zustandsübergangsgraphen

- In einem Zustandsübergangsgraphen wird jeder Zustand $s_i \in S$ als ein Kreis dargestellt
- Der Anfangszustand wird mit einem auf den Zustand zeigendem Dreieck gekennzeichnet
  - Ist kein Startzustand definiert, wird angenommen, dass sich alle Speicherelemente im Zustand 0 befinden
- Die möglichen Zustandsübergänge werden mit Pfeilen gekennzeichnet
- Jeder Pfeil wird mit dem zugehörigen Eingabesymbol $a_m \in A$ beschriftet, für das dieser Zustandsübergang auftritt
- Außerdem (abgetrennt durch einen "/") kann jeweils das Ausgabesymbol $b_n \in B$ angegeben werden

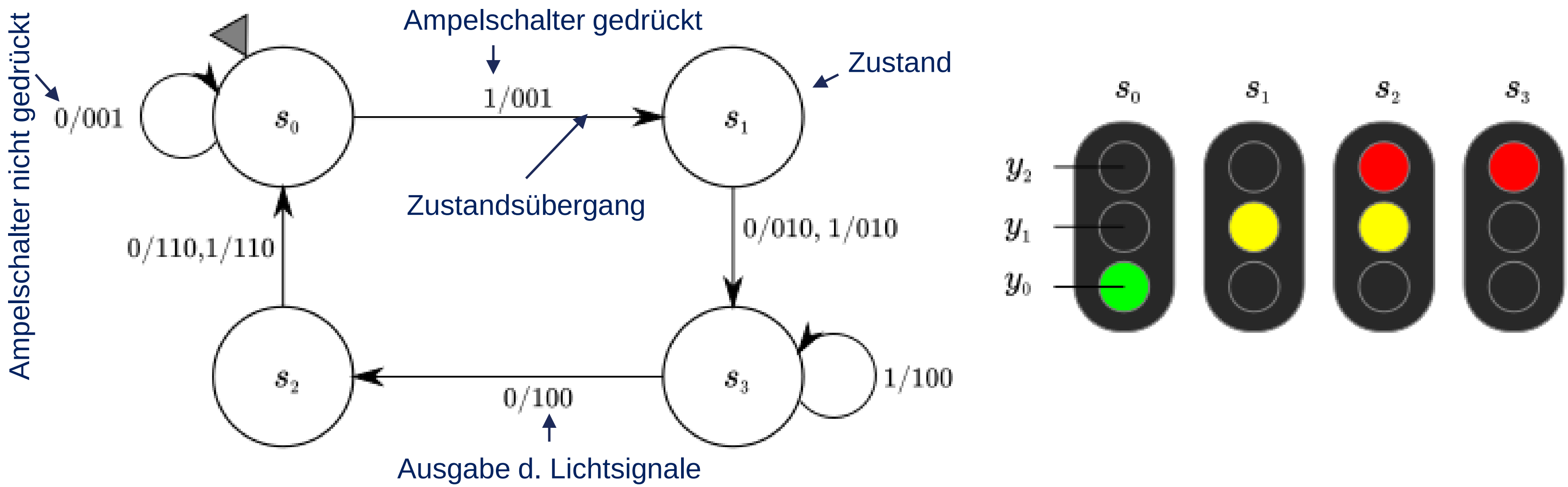


Zustandsübergangsgraphen

- Beispiel: Ampelschaltung (simplifiziert)
  - Es soll eine Schaltung für eine Fußgängerampelanlage erstellt werden
  - Es wird dabei die Ampel, die den Autoverkehr regelt, betrachtet (**nicht** die Fußgängerampel)
  - Die Ampel reagiert auf das Drücken eines Ampelknopfs durch einen Fußgänger:
    - $a_0 = 0$ bedeutet der Ampelknopf wurde nicht gedrückt
    - $a_1 = 1$ bedeutet der Ampelknopf wurde gedrückt
  - Im Anfangszustand $s_0 \in S$ ist die Ampel grün
  - Wurde der Ampelknopf gedrückt, soll die Ampel zunächst auf gelb und dann auf rot schalten
  - Es wird weiterhin davon ausgegangen, dass das durch den Ampelknopf gesteuerte Eingabesymbol automatisch, nachdem der Fußgänger genug Zeit hatte die Straße zu überqueren, von a_1 nach a_0 wechselt
  - Die Ampel soll dann zunächst gelb-rot zeigen und schließlich wieder grün

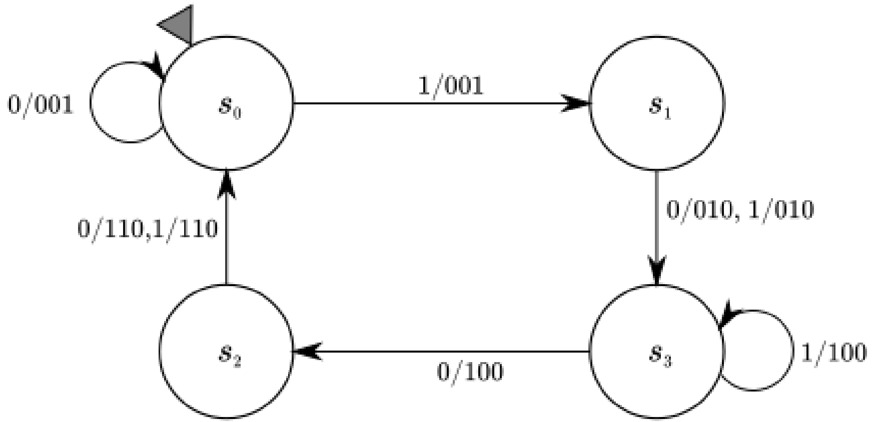
Zustandsübergangsgraphen

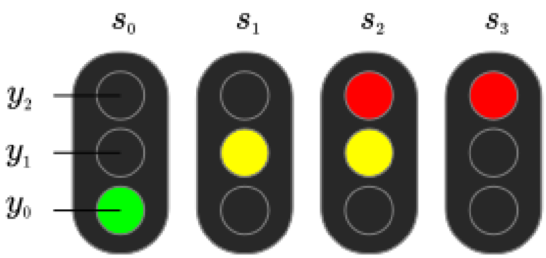
- Beispiel: Ampelschaltung



- Menge der Eingabesymbole A = {a_0, a_1} binär kodiert mit {0,1}
- Menge der Zustände S = {s_0, s_1, s_2, s_3}
- Menge der Ausgabesymbole B = {b_0, b_1, b_2, b_3} binär kodiert mit $y_2y_1y_0$ zu {001, 010, 110, 100}
- Es gibt |S| · |A| = 4 · 2 = 8 mögliche Zustandsübergänge

Beispiel

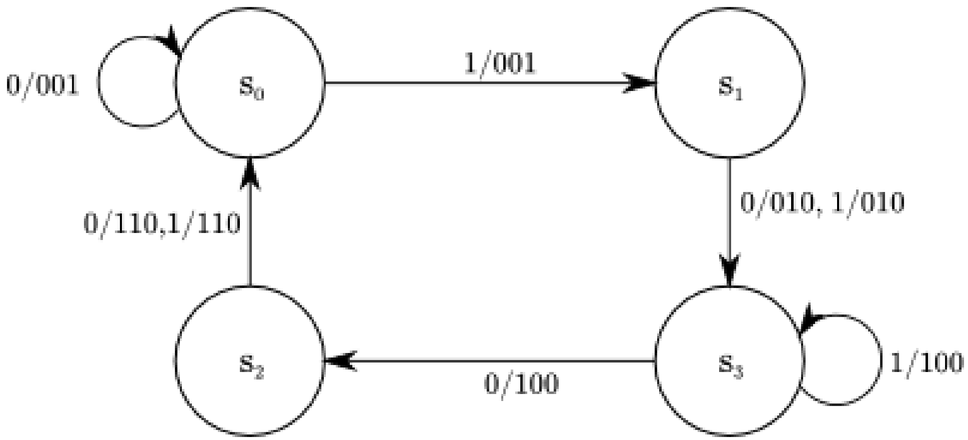




- Ausgabefunktion:
  - $y_2 = q_1$
  - $y_1 = q_0 \nleftrightarrow q_1$
  - $y_0 = \neg q_0 \wedge \neg q_1 = \neg(q_0 \vee q_1)$

Zustand	q_1	q_0	y_2	y_1	y_0
s_0	0	0	0	0	1
s_1	0	1	0	1	0
s_2	1	0	1	1	0
s_3	1	1	1	0	0

Beispiel



Zustand	q_1	q_0	x_0	d_1	d_0
s_0	0	0	0	0	0
s_0	0	0	1	0	1
s_1	0	1	0	1	1
s_1	0	1	1	1	1
s_2	1	0	0	0	0
s_2	1	0	1	0	0
s_3	1	1	0	1	0
s_3	1	1	1	1	1

- Übergangsfunktion
  - $d_1 = q_0$
  - $d_0 = (\neg q_1 x_0) \vee (\neg q_1 q_0) \vee (q_0 x_0)$

- q_1, q_0 = aktueller Zustand
- d_1, d_0 = Eingänge der Zustandsspeicher (FlipFlops)

Beispiel

Ausgabefunktion

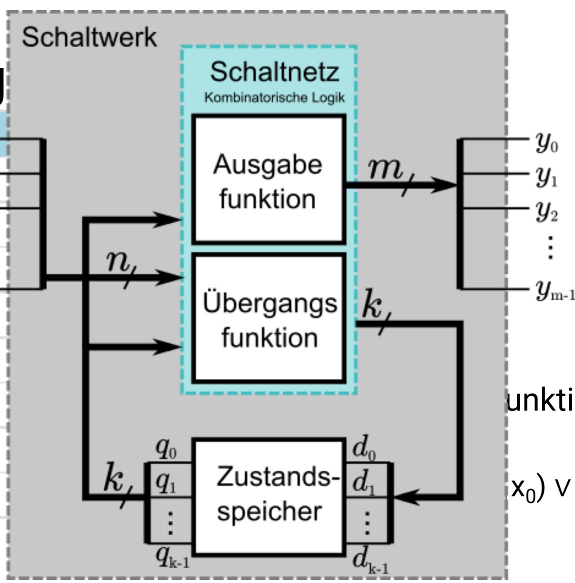
Zustand	q_1	q_0	y_2	y_1	y_0
s_0	0	0	0	0	1
s_1	0	1	0	1	0
s_2	1	0	1	1	0
s_3	1	1	1	0	0

Ausgabefunktion:

- $y_2 = q_1$
- $y_1 = q_0 \nleftrightarrow q_1$
- $y_0 = \neg q_0 \wedge \neg q_1 = \neg(q_0 \vee q_1)$

Überg

Zustand
s_0
s_0
s_1
s_1
s_2
s_2
s_3
s_3



unktion

$x_0) \vee (\neg q_1 q_0) \vee (q_0 x_0)$

Schaltungsentwurf