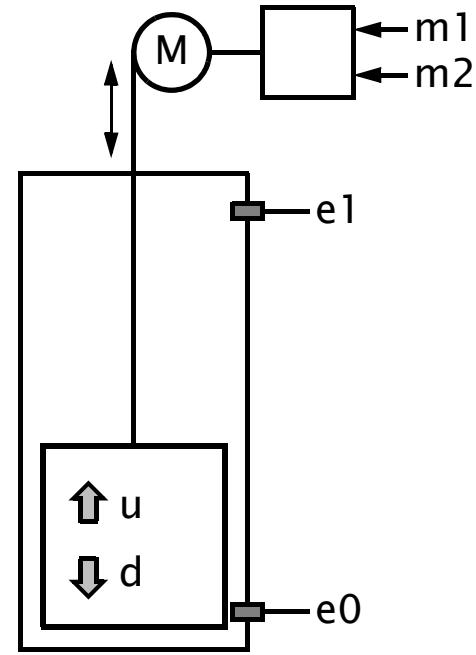
- Einführendes Beispiel – Aufzugsteuerung

Synthese einer Aufzugsteuerung für ein vereinfachtes Modell einer Aufzugsanlage in einem Gebäude mit zwei Etagen



- Aufzugsanlage besteht aus
  - einem Maschinenraum mit dem Antrieb und mit der Aufzugsteuerung
  - einem Aufzugsschacht mit Tragseilen, Gegengewichten und Positionssensoren
  - einer Fahrkabine mit einer einfachen Schalttafel

Im Aufzugsschacht sind zwei digitale (Positions-)Sensoren e0 und e1 zur Erfassung der Position der Fahrkabine montiert. Die Position der Fahrkabine wird laufend an die Aufzugsteuerung mitgeteilt. Mit ei=1 wird ein aktiver Sensor gekennzeichnet.

e1	e0	Position der Fahrkabine
0	0	zwischen den Etagen
1	0	in der oberen Etage
0	1	in der unteren Etage

In der Fahrkabine befindet sich eine einfache Schalttafel mit zwei Tasten u (up) und d (down), mit denen der Fahrgast die gewünschte Fahrtrichtung auswählen kann.

u	d	Fahrtrichtung
0	0	keine
1	0	aufwärts
0	1	abwärts

Der Antrieb besteht aus einem Elektromotor M, der über eine Motorsteuerung mit zwei Signalen m1 und m2 gesteuert wird. Der Elektromotor kann entweder stehen oder sich wahlweise entweder nach links oder rechts drehen. Dabei wird die Drehbewegung des Elektromotors über eine Umlenkrolle in die vertikale Bewegung der Fahrkabine umgesetzt.

m1	m2	Elektromotor
0	0	bleibt stehen
0	1	dreht sich nach rechts
1	0	dreht sich nach links

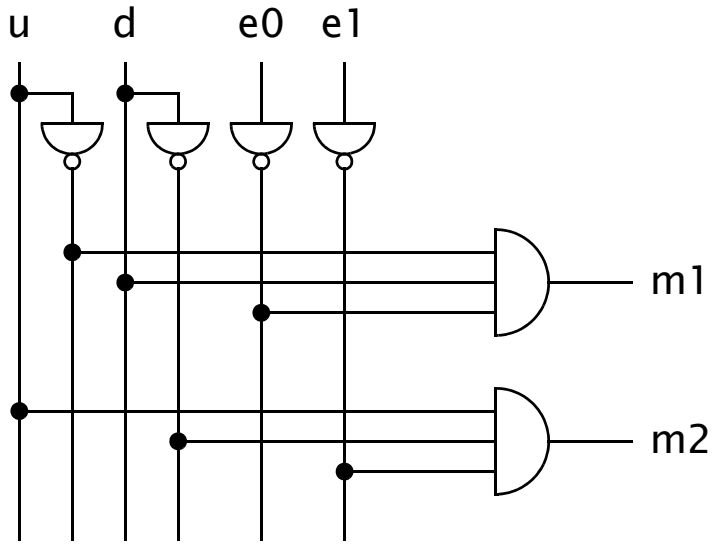
Die Aufzugsteuerung soll folgendes Verhalten aufweisen:

1. Die Fahrkabine wartet auf der aktuellen Etage so lange, bis ein Fahrgast über die Schalttafel die gewünschte Fahrtrichtung gewählt hat.
2. Die Fahrkabine soll sich in die gewünschte Fahrtrichtung bewegen (in die obere bzw. in die untere Etage), sofern sie sich nicht in der entsprechenden Etage befindet.

- Aufzugsteuerung als Schaltnetz
  - Funktionstabelle

e1	e0	u	d	m1	m2
-	-	0	0	0	0
-	-	1	1	0	0
0	0	1	0	0	1
0	1	1	0	0	1
1	0	1	0	0	0
0	0	0	1	1	0
1	0	0	1	1	0
0	1	0	1	0	0

  - Schaltnetz



  - Funktionsgleichungen

    $m1 = f1(u, d, e1, e0) = u' \cdot d \cdot e0'$

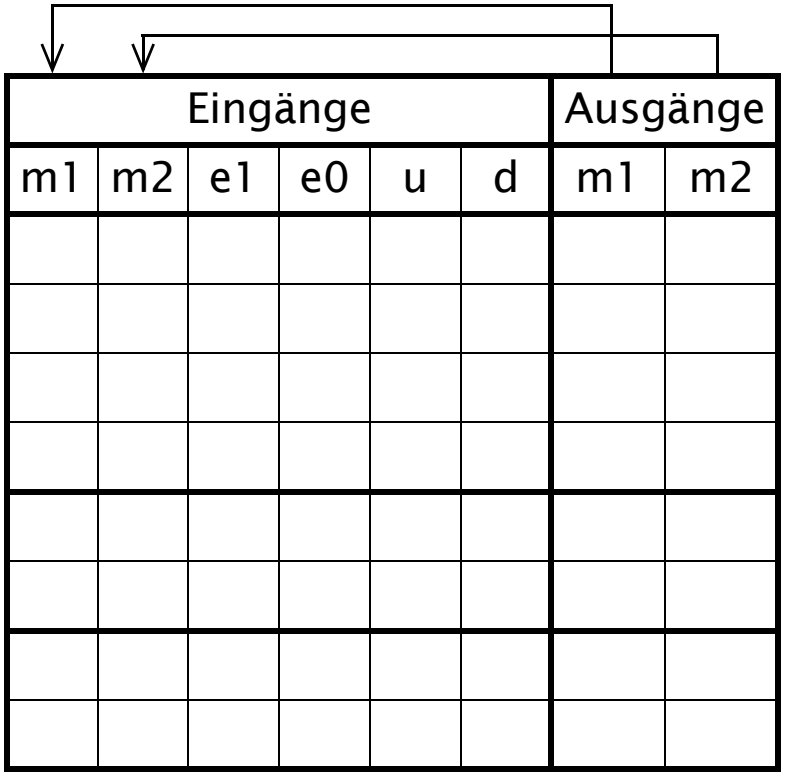
    $m2 = f2(u, d, e1, e0) = u \cdot d' \cdot e1'$

- Analyse der bisherigen Lösung
  - Die Fahrkabine bewegt sich nur, wenn ein Fahrgast die passende Taste u oder d gedrückt hält. Läßt er die Taste los, dann bleibt die Fahrkabine stehen (auch zwischen den Etagen). Der Fahrgast kann die Fahrtrichtung zu jedem Zeitpunkt ändern.

  - In der Realität funktioniert eine Aufzugsteuerung ein wenig anders.
  - Man braucht nicht dauernd eine Taste für die Fahrtrichtung gedrückt zu halten, wenn man mit dem Aufzug fahren will.
  - Man drückt nur einmal auf eine Taste und die Fahrkabine setzt sich in Bewegung.
  - Die Aufzugsteuerung „merkt“ sich also irgendwie die gewählte Fahrtrichtung, so daß die Fahrkabine automatisch weiterfährt, auch nachdem der Fahrgast die Taste losgelassen hat.
  - Wir erweitern deshalb die Aufzugsteuerung um eine neue Funktionalität, mit der dieses Verhalten realisiert wird.

- Analyse der bisherigen Lösung (Fortsetzung)
  - Dieses „Merken“ der gewählten Fahrrichtung kann z.B. so erfolgen,
    - daß die Steuerung sich an die gedrückte Taste erinnert, oder
    - daß die Steuerung die Information aus der Drehrichtung des Elektromotors rekonstruiert.
  - Wir verfolgen den zweiten Ansatz:
    Dazu müssen in der Funktionstabelle eingangsseitig zwei neue Spalten eingefügt werden, die mit den Signalen m1 und m2 zur Steuerung des Elektromotors übereinstimmen.
  - Auf diese Weise entsteht eine Rückkopplung in der Funktion. Die Ausgangsfunktionen m1 und m2 sind nicht nur von der Eingangsvariablen/-signalen e1, e0, u und d abhängig, sondern auch von sich selbst (also von m1 und von m2).
  - Somit ergeben sich folgende, rückgekoppelte Funktionen:
    m1(m1, m2, e1, e0, u, d) und m2(m1, m2, e1, e0, u, d)

- Erweiterte Aufzugsteuerung
  - Funktionstabelle



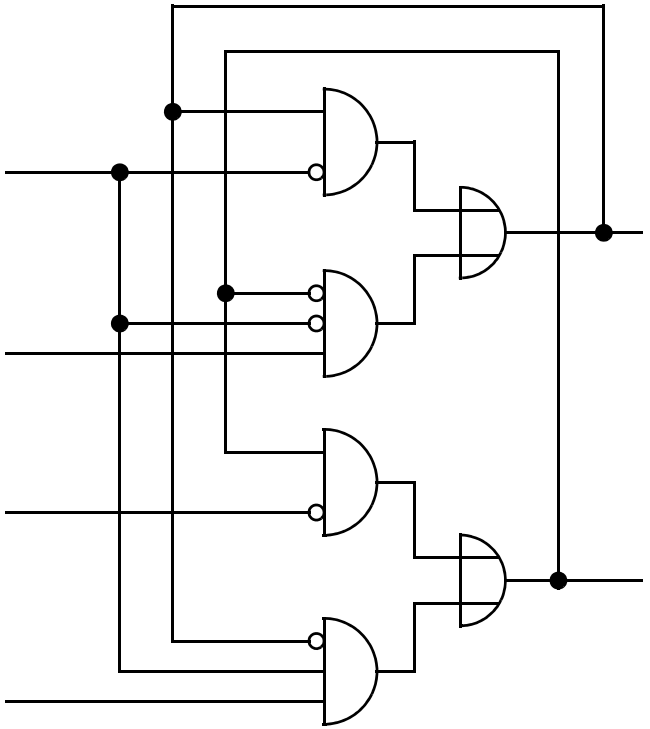
Eingänge						Ausgänge	
m1	m2	e1	e0	u	d	m1	m2

  - Funktionsgleichungen

    m1 = g1(m1, m2, u, d, e1, e0) =

    m2 = g2(m1, m2, u, d, e1, e0) =

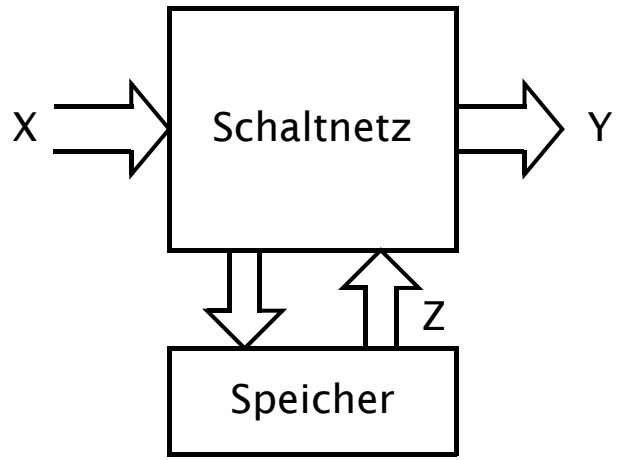
Schaltnetz mit Rückkopplung



- Funktionsweise der erweiterten Aufzugsteuerung
  - die Fahrkabine steht:
    - der Elektromotor dreht sich nicht (m1=0, m2=0)
    - sie steht entweder in der oberen Etage (e1=1, e0=0) oder in der unteren Etage (e1=0, e0=1)
    - den Fall, daß sie zwischen den beiden Etagen stehen bleibt, schließen wir der Einfachheit halber aus.
    - der Fahrgast kann die Fahrtrichtung nicht zu jedem (beliebigen) Zeitpunkt auswählen, sondern nur wenn die Fahrkabine entweder in der unteren oder in der oberen Etage steht.
  - die Fahrkabine fährt aufwärts:
    - der Elektromotor dreht sich nach rechts (m1=0, m2=1), und zwar nur so lange, bis die Fahrkabine die obere Etage erreicht hat (e1=1), dann stoppt die Aufzugsteuerung den Elektromotor.
  - die Fahrkabine fährt abwärts:
    - der Elektromotor dreht sich nach links (m1=1, m2=0), und zwar nur so lange, bis die Fahrkabine die untere Etage erreicht hat (e0=1), dann stoppt die Aufzugsteuerung den Elektromotor.

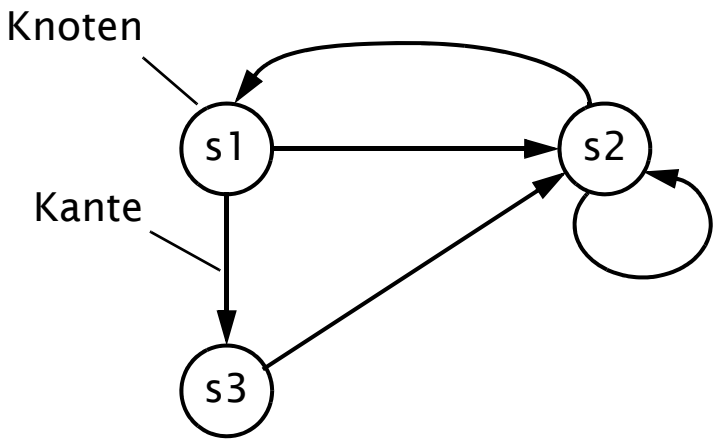
- Ein Schaltnetz ist
  - eine zustandsfreie, kombinatorische Schaltung ohne „Gedächtnis“.
  - Ausgangswerte Y sind ausschließlich und unmittelbar (abgesehen vor einer kurzen Verzögerung) von den aktuellen Eingangswerten X abhängig.

- Ein Schaltwerk ist
  - ein Schaltnetz mit „Gedächtnis“, eine zustandsbehaftete, sequentielle Schaltung.
  - Ausgangswerte Y sind zu einem bestimmten Zeitpunkt sowohl vom vergangenen Verhalten der Schaltung Z als auch von den aktuellen Eingangswerten X abhängig.



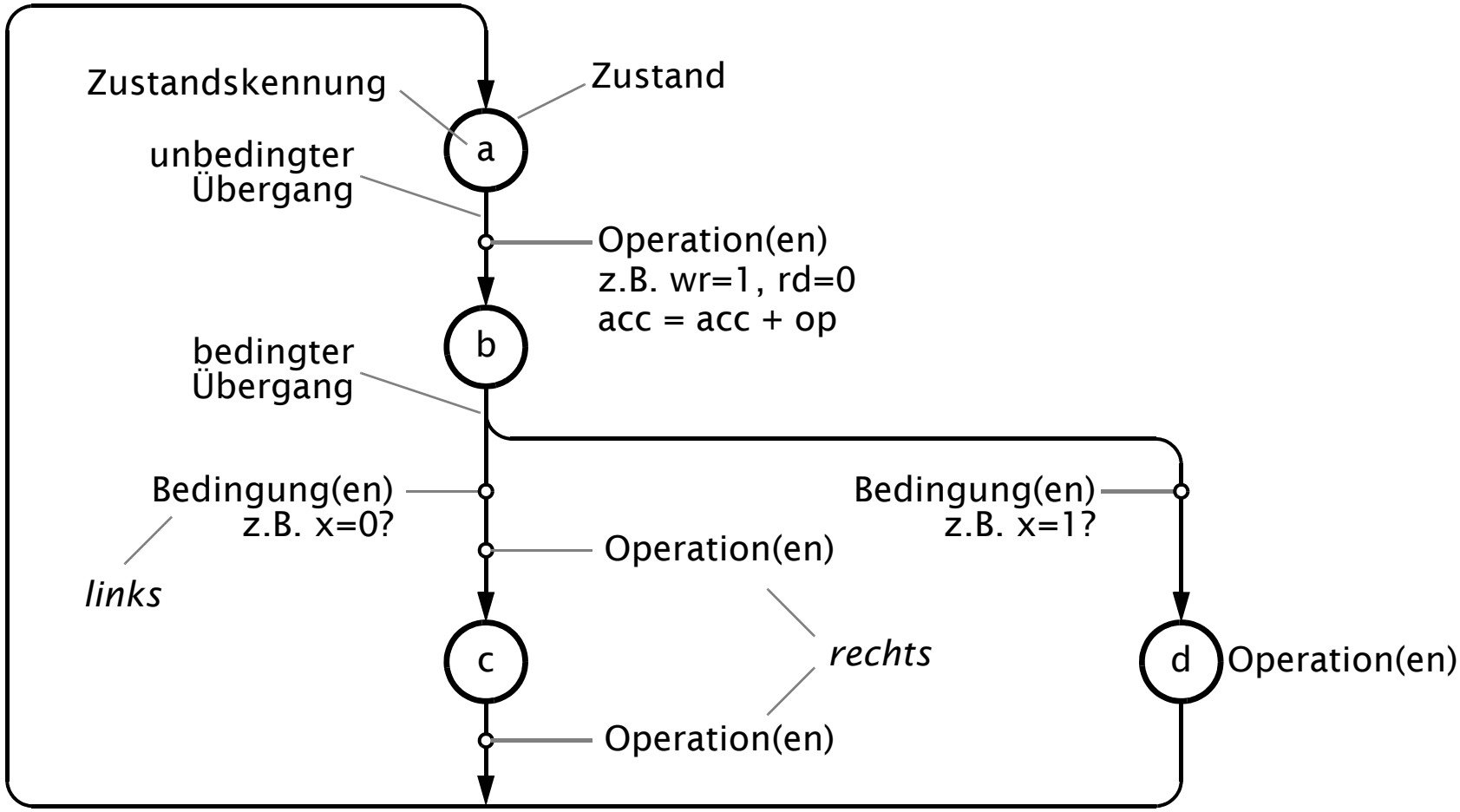
- Ein gerichteter Graph G = (V, E) besteht aus zwei endlichen Mengen:
  - einer nicht leeren Knotenmenge V und
  - einer (möglicherweise leeren) Kantenmenge E,

  so daß jeder Kante e aus E ein geordnetes Knotenpaar (*u*, *v*) zugeordnet ist. Man sagt, daß die Kante *e* den Knoten *u* mit dem Knoten *v* verbindet. Dabei wird *u* als Anfangsknoten von *e* und *v* als Endknoten von *e* bezeichnet.



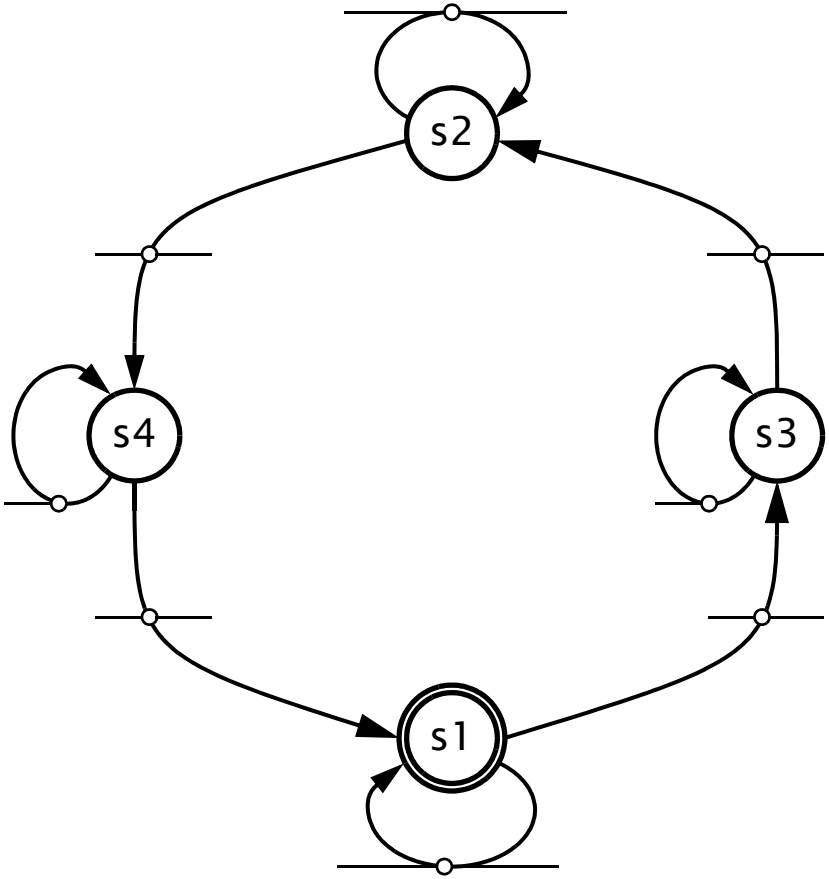
- Ein Zustandsgraph $G_z = (G, V_0, V_N, B, A)$ ist ein gerichteter Graph G, dessen Knoten als *Zustände* und Kanten als *Zustandsübergänge* interpretiert werden, erweitert um folgende endliche Mengen:
  - eine nicht leere Knotenmenge V_0 für Anfangszustände,
  - eine (möglicherweise leere) Knotenmenge V_N für Endzustände,
  - eine (möglicherweise leere) Menge von Bedingungen B, die den Zustandsübergängen zugeordnet sind,
  - eine (möglicherweise leere) Menge von Aktionen A (Operationen), die entweder den Zuständen oder den Zustandsübergängen zugeordnet sein können.
- Semantikregel: Wenn ein Zustand aktiv ist, und eine Bedingung wahr ist, dann finden der Zustandsübergang und die Ausführung der Aktion(en) statt.

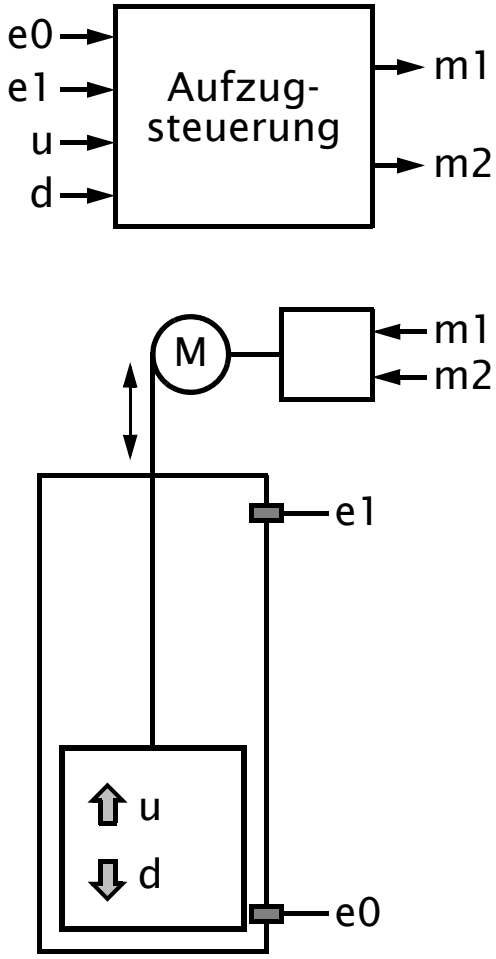
- Zustandsgraph mit einer praktischen Erklärung



- Zustandsorientierte Modellierung der Aufzugsteuerung
  - Zustände:
    - s1: Fahrkabine steht in der unteren Etage
    - s2: Fahrkabine steht in der oberen Etage
    - s3: Fahrkabine fährt aufwärts
    - s4: Fahrkabine fährt abwärts
  - Zustandsübergänge:
    - e1: Fahrkabine setzt sich in Aufwärtsbewegung
    - e2: Fahrkabine erreicht die obere Etage
    - e3: Fahrkabine setzt sich in Abwärtsbewegung
    - e4: Fahrkabine erreicht die untere Etage
  - Bedingungen:
    - b1: die Taste u (up) wird betätigt
    - b2: die Taste d (down) wird betätigt
    - b3: der Positionssensor in der unteren Etage wird aktiviert
    - b4: der Positionssensor in der oberen Etage wird aktiviert
  - Aktionen:
    - a1: der E-Motor für die Abwärtsbewegung wird eingeschaltet
    - a2: der E-Motor für die Aufwärtsbewegung wird eingeschaltet

- Zustandsgraph für die Aufzugsteuerung





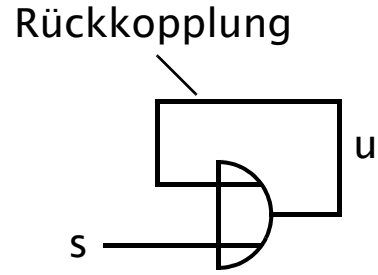
- Digitale Systeme lassen sich grundsätzlich in *Schaltnetze* und *Schaltwerke* aufteilen.
- Schaltwerke können als *synchron* oder *asynchron* arbeitende Schaltwerke realisiert sein.

- Synchrone Schaltwerke
  - Änderungen interner Zustände erfolgen nur zu vorgegebenen Zeitpunkten.
  - Zwischen den Änderungszeitpunkten bleiben alle Zustandswerte stabil.
  - Es gibt ausgewählte (externe) Signale (sog. Taktimpulse), die normalerweise auf alle Zustandsgrößen wirken, und Änderungen des Systems bewirken.
  - Da alle Zustandsgrößen sich gleichzeitig ändern, erhält man so ein synchrones Verhalten des gesamten Systems.

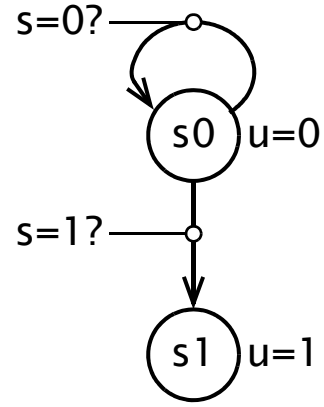
- Asynchrone Schaltwerke
  - Änderungszeitpunkte sind nicht extern vorgegeben (keine expliziten Taktimpulse).
  - Änderungszeitpunkte können jeder Zeit auftreten und sind normalerweise von der konkreten technischen Realisierung anhängig.
  - Aufgrund von Laufzeitunterscheiden bei den einzelnen Zustandsgrößen kann es zu Problemen wie Hazards oder Wettläufen (Races) kommen. Dadurch können sich Signale auf verschiedenen Pfaden überholen.
  - Sind mehrere Zustandsgrößen im System vorhanden, können sich diese asynchron (nicht gleichzeitig) ändern.
- Synchrone Schaltwerke zeigen diese Probleme nicht und sind daher bei der Realisierung zu bevorzugen.

- In Schaltnetzen mit Rückkopplung(en) können Informationen dauerhaft gespeichert werden (vgl. Aufzugsteuerung)
  $\Rightarrow$ Automaten, Schaltwerke
- Aufgrund der Komplexität heutiger Aufgabenstellungen werden Schaltwerke selten als Schaltnetze mit Rückkopplungen gebaut (aufwändige Timing-Analyse, Maßnahmen gegen Hazards).
- Komplexe Aufgabenstellungen werden
  - systematisch mit Hilfe der Automatentheorie modelliert, und
  - modular als Schaltwerke, bestehend aus Speicherelementen sowie aus Übergangs- und Ausgangsschaltnetzen realisiert.
- Die Grundlage solcher Speicherelemente bilden relativ einfache Schaltnetze mit Rückkopplungen.

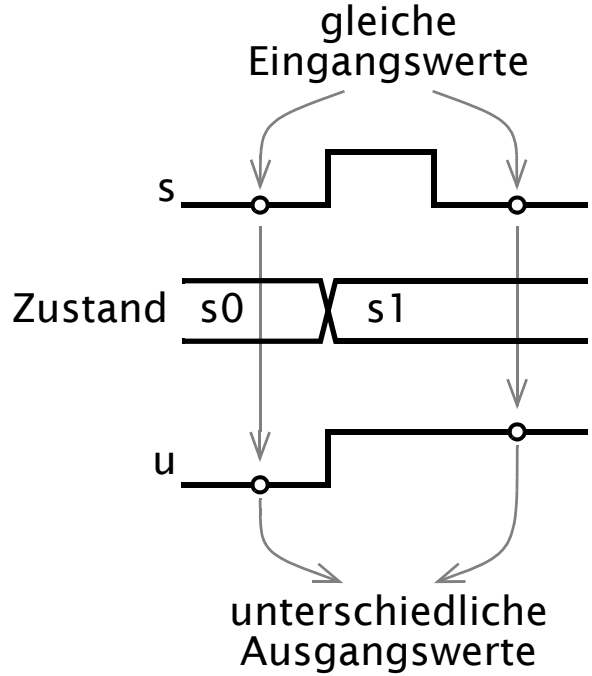
- Speicherelement zur Speicherung eines Impulses
  - Das einfachste Schaltnetze mit einer Rückkopplung besteht aus einem OR-Gatter mit zwei Eingängen und einem Ausgang, in dem der Ausgang mit einem der beiden Eingänge über eine Rückkopplung verbunden ist.



  - Das so aufgebaute minimale Schaltwerk kann an seinem Eingang einen Impuls s=0→1 „aufspüren“ und diese Änderung/Information durch die Rückkopplung dauerhaft als Zustand u=1 „speichern“.
  - Funktionsweise:
    - Solange sich das Speicherelement im Zustand s0 befindet (mit u=0), kann ein Impuls (also eine Signaländerung 0→1) auf der Eingangsleitung s erfaßt werden.
    - Ist ein Impuls (s=1) erfaßt, so geht das Speicherelement in den Zustand s1 (mit u=1) über.
    - Alle nachfolgenden Impulse können den Zustand s1 des Speicherelements nicht mehr ändern.

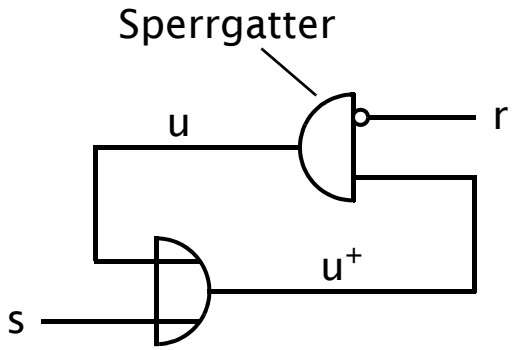


- Speicherelement zur Speicherung eines Impulses
  - Am Impulsplan sieht man, wie sich die Änderung eines Eingangswertes auf den Zustand der Schaltung auswirkt.
  - **Der (innere) Zustand** ist derjenige Teil der Information über eine Schaltung, der neben den aktuellen Eingangswerten die Ausgangswerte mitbestimmt.
  - Das einfachste Schaltwerk zur Speicherung eines Impulses kann nicht in seinen Ausgangszustand s0 (mit u=0) wieder versetzt werden, z.B. um erneut einen Impuls erfassen zu können.

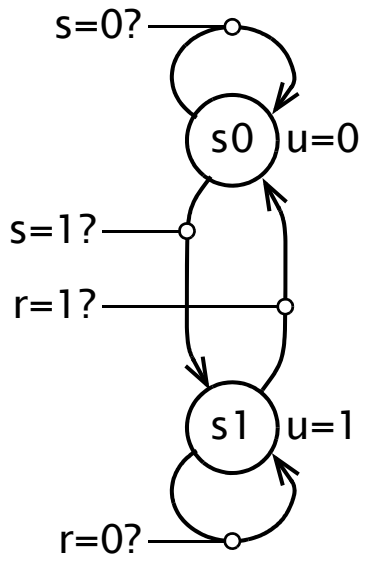


  - Damit das Zurücksetzen (sog. Normieren) des Speicherelementes möglich ist, wird das Speicherelemente um ein sog. Sperrgatter in der Rückkopplungsschleife erweitert.

- Speicherelement zur Speicherung von Werten einer Variable
  - Das Sperrgatter ist ein 2er-AND-Gatter mit einem invertierten Eingang, an dem ein Rücksetzsignal r angelegt ist.
  - Bei r=0 wird die Information durch die Rückkopplung aufrechterhalten.
  - Bei r=1 wirkt das AND-Gatter wie eine Sperre und unterbricht den Informationsfluß in der Rückkopplung.

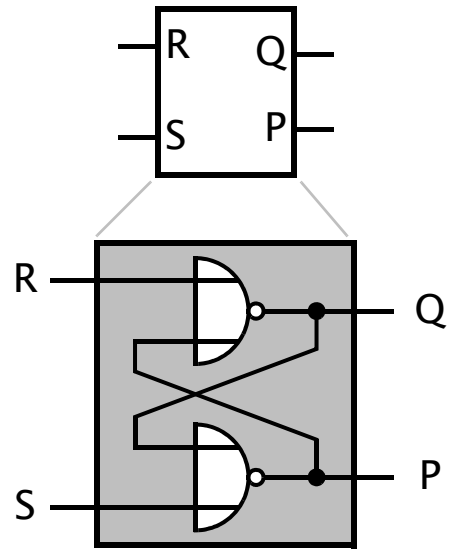


  - Durch Übergang auf NOR-Gatter und Herausführen beider Ausgänge entsteht eine elementare Schaltung, mit der die Werte einer booleschen Variable gespeichert werden können.Diese Schaltung wird als RS-Flipflop bezeichnet.



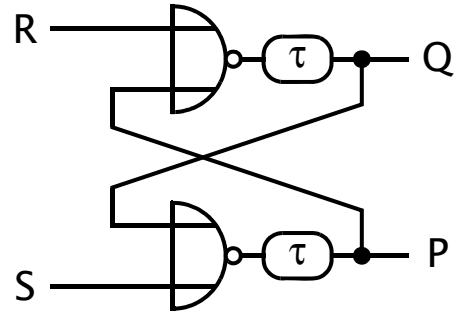
- RS-Flipflop
  - zwei Eingänge: R (Reset) und S (Set)
  - zwei Ausgänge: Q und P (mit P := Q')
  - Funktionstabelle

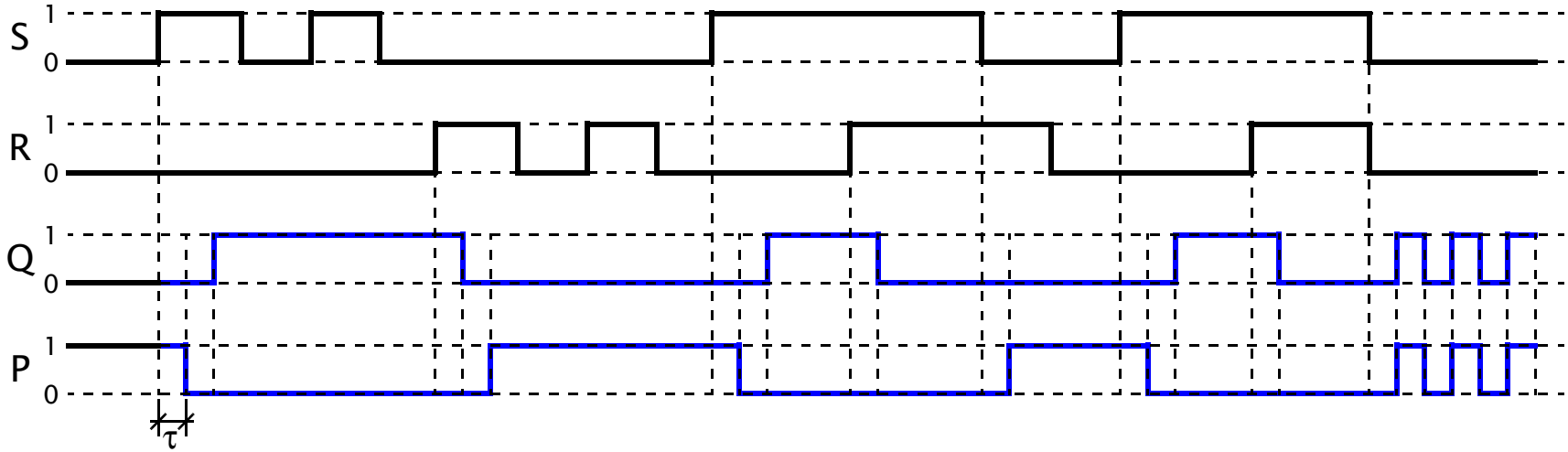
S	R	Q^+	P^+	Funktion
1	0	1	0	setzen
0	1	0	1	löschen
0	0	Q	P	halten
1	1	x	x	irregulär



  - zwei stabile Zustände (P=0, Q=1) und (P=1, Q=0)
  - einen transienten Zustand (P=Q=0)
  - S=R=1 ist irregulär, weil:
    - beide Ausgänge dauerhaft P = Q = 0
    - undefiniertes (metastabiles) Verhalten beim Wechsel von S=R=1 nach S=R=0

- Timing-Analyse eines RS-Flipflop
  - Charakteristische Funktionsgleichung:
    $Q(t + 1) = S(t) + R(t)' \cdot Q(t)$
    $Q^+ = S + R' \cdot Q$ mit $P = Q'$





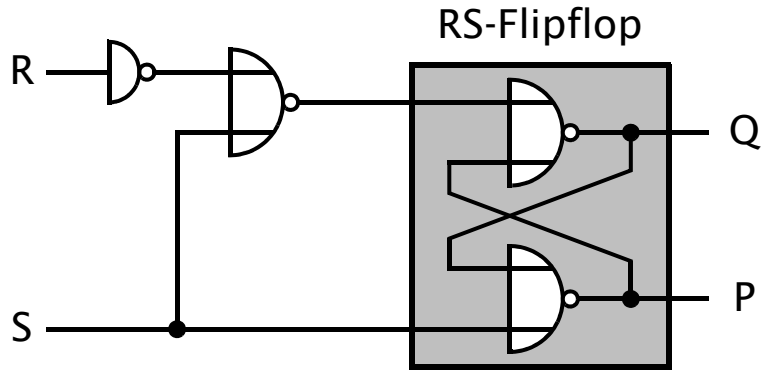
- Zusammenfassung der Timing-Analyse eines RS-Flipflop
  - Im Fall der irregulären Eingangsbedingung (S=R=1) sind beide Ausgangssignale 0 (Q=P=0).
  - Wenn unmittelbar nach der irregulären Eingangsbedingung die Funktion „halten“ (S=R=0) folgt, dann zeigt das RS-Flipflop ein instabiles (oszillierendes) Verhalten.
  - Durch die Verzögerungen der beiden NOR-Gatter ist der übernommene Wert dann nicht eindeutig vorhersagbar.

  - Kann in einem Entwurf aber ausgeschlossen werden, daß unmittelbar nach der irregulären Eingangsbedingung die Funktion „halten“ folgt, dann läßt sich auch eine solche Eingangsbedingung in einer Schaltung sinnvoll einsetzen.
  - Folgen die Funktionen „setzen“ oder „löschen“ unmittelbar nach der irregulären Eingangsbedingung, so nimmt das RS-Flipflop einen definierten und stabilen Zustand an.

- RS-Flipflops mit besonderem Verhalten
  - Die irreguläre Eingangsbedingung S=R=1 läßt sich mit einem zusätzlichen Schaltnetz vor den beiden Eingängen eines RS-Flipflops verhindern.
  - Es gibt drei Alternativen für die Behandlung der Eingangs-bedingung S=R=1, die drei modifizierte RS-Flipflops ergeben:
    - RS-Flipflop mit Speichervorrang
    - RS-Flipflop mit Löschvorrang
    - RS-Flipflop mit Setzvorrang

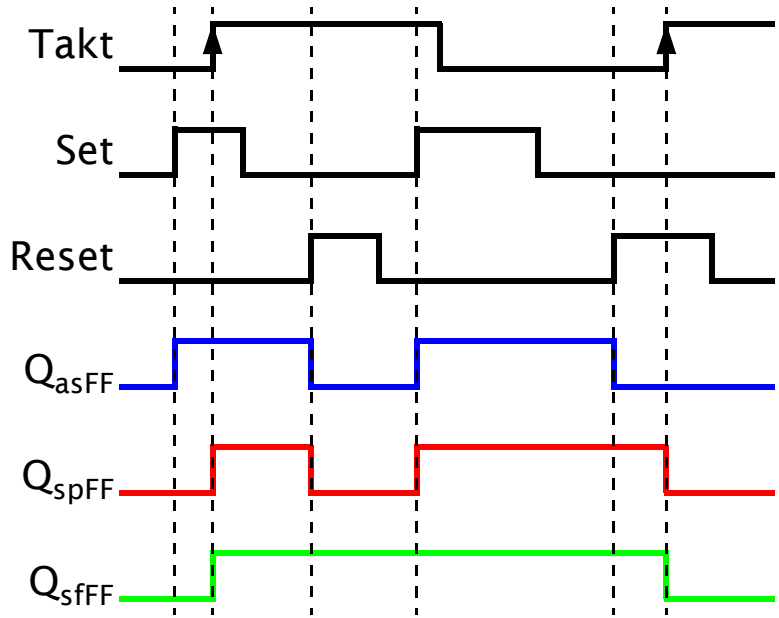
  - RS-Flipflop mit Setzvorrang

S	R	Q^+	P^+	Funktion
1	0	1	0	setzen
0	1	0	1	löschen
0	0	Q	P	halten
1	1	1	0	setzen



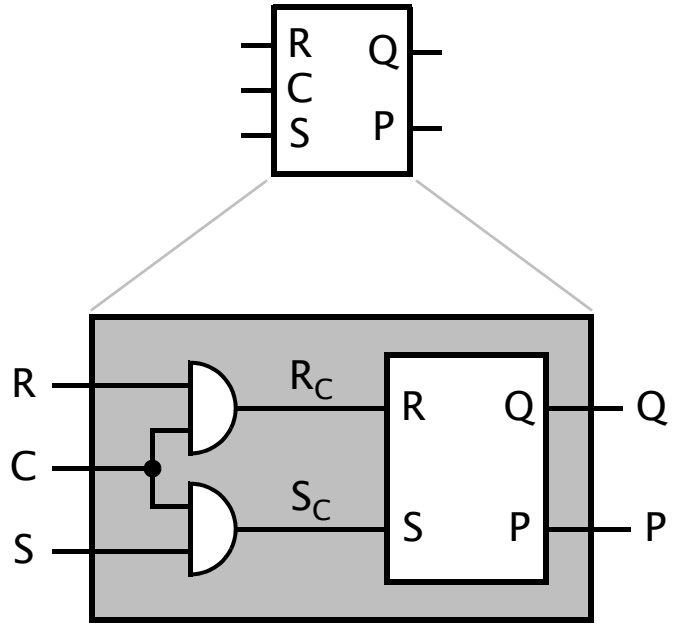
- Verschiedene Arten der Steuerung von Flipflops
  - asynchrone (ungetaktete) Flipflops: ihr Zustand wird nur durch die Setzen-/Rücksetzen-Eingänge (Set-/Reset) gesteuert.
  - synchrone (getaktete, getriggerte) Flipflops: der Zeitpunkt der Informationsübernahme wird durch ein Takt-/Steuersignal vorgegeben.

    - pegelgesteuerte (zustandsgesteuerte) Flipflops: die Übernahme der Information wird durch einen Pegel (0/1) des Steuersignals veranlaßt.
    - einflankengesteuerte Flipflops: die Übernahme der Information wird durch einen Zustandswechsel (0→1 oder 1→0) des Steuer-signals veranlaßt.
    - zweiflankengesteuerte Flipflops (Master-Slave-Flipflops)

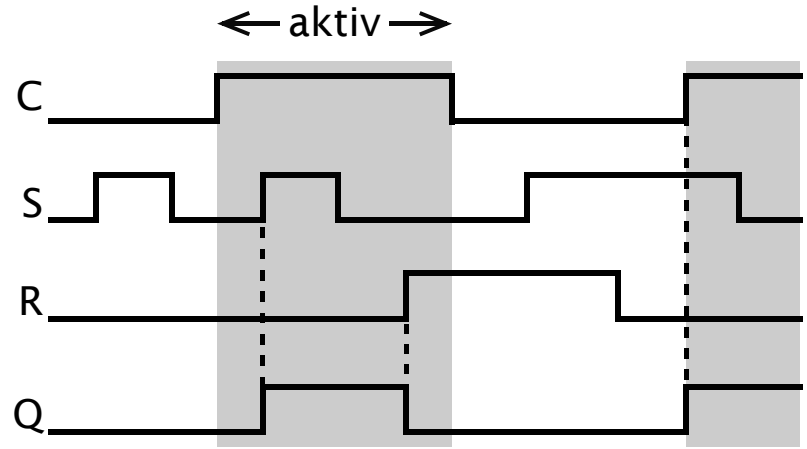


- synchrones, pegelgesteuertes RS-Flipflop
  - Ein asynchrones RS-Flipflop wird um ein zusätzliches Schaltnetz, bestehend aus zwei AND-Gattern, vor den Eingängen R und S zu einem synchronen, pegelgesteuerten RS-Flipflop erweitert.
  - Eingangsignale an R und S werden nur dann übernommen, wenn das Taktsignal C aktiv ist, d.h. C=1.
  - Funktionstabelle

C	S	R	Q^+	P^+	Funktion
0	-	-	Q	P	halten
1	0	0	Q	P	halten
1	0	1	0	1	löschen
1	1	0	1	0	setzen
1	1	1	x	x	irregulär



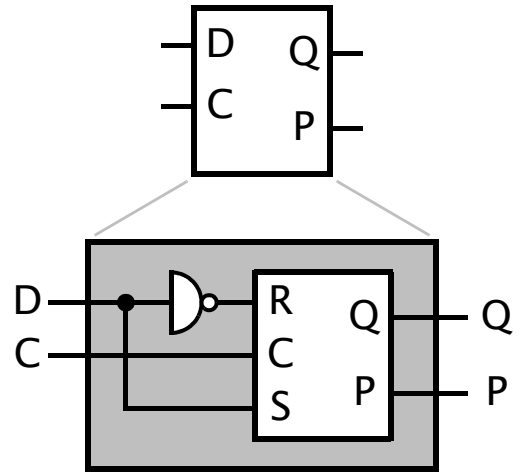
- Timing-Analyse
  - aktive Taktphase (C=1): die R- und S-Eingänge sind „frei geschaltet“, mehrere Zustands-änderungen möglich.
  - nicht aktiven Taktphase (C=0): die R- und S-Eingänge bleiben „gesperrt“; Impulse auf diesen Eingangsleitungen haben keine Wirkung. Der Zustand des RS-Flipflops ist von den R- und S-Eingängen unabhängig und bleibt somit gespeichert.



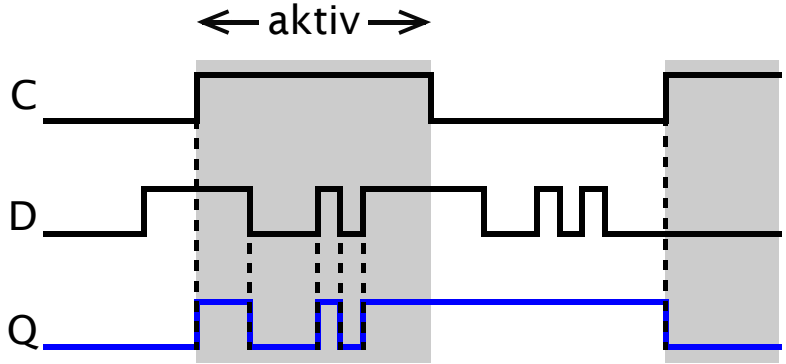
  - Bei einem Übergang des Taktes in die nicht aktive Phase (C: 1→0) wird der letzte Zustand entsprechend den R- und S-Eingängen übernommen.
  - Signale an den R- und S-Eingängen sollten stets länger anliegen, als die aktive Taktphase dauert.

- synchrones, pegelgesteuertes D-Flipflop
  - ein (Daten-)Eingang D
  - zwei Ausgänge: Q und P (mit P := Q')
  - Funktionstabelle

C	D	Q^+	P^+	Funktion
0	–	Q	P	halten
1	0	0	1	transparent
1	1	1	0	transparent



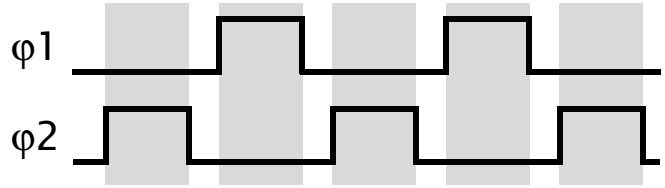
  - Durch geeignete Beschaltung (S≠R) wird die ungültige Eingangskombination vermieden.
  - Solange das Taktsignal aktiv ist (C=1), ist das D-Flipflop für das Datensignal D *transparent*, d.h. der Ausgang folgt dem Eingang.



- Generelles Problem bei pegelgesteuerten Flipflops
  - In der aktiven Taktphase (C=1) sind alle synchronen, pegelgesteuerten Speicherelemente transparent, d.h. eine Änderung am Eingang kann sich direkt auf den Ausgang auswirken.
  - Problematische Anwendung in Schaltwerken mit Rückkopplungen bei langen aktiven Taktphasen und kurzen Signallaufzeiten können asynchrone Signalschleifen entstehen, die zum nicht deterministischen Verhalten eines Schaltwerks führen können.

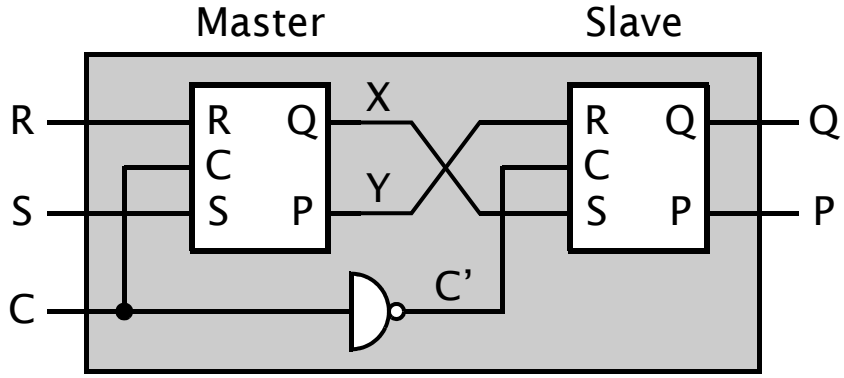
- Maßnahmen zur Beseitigung der Transparenz
  - Verkürzung der aktiven Taktphase
    die aktive Taktphase (C=1) wird so kurz wie möglich gehalten (kürzer als die kürzeste Signallaufzeit), $\Rightarrow$ asymmetrischer Takt mit kurzen aktiven Phasen und langen nicht aktiven Phasen.

- Maßnahmen zur Beseitigung der Transparenz
  - Zusicherung von Stabilität der Eingangssignale während der aktiven Taktphase
  - Einführung eines Zweiphasentaktes mit nicht überlappenden Taktphasen φ1 und φ2
    - möglich im Vollkunden-/ASIC-Design
    - schwierig in Systemen mit diskreten Bauteilen oder FPGA/CPLD-Bausteinen, weil solche Systeme i.d.R mit einen zentralen Taktgeber arbeiten



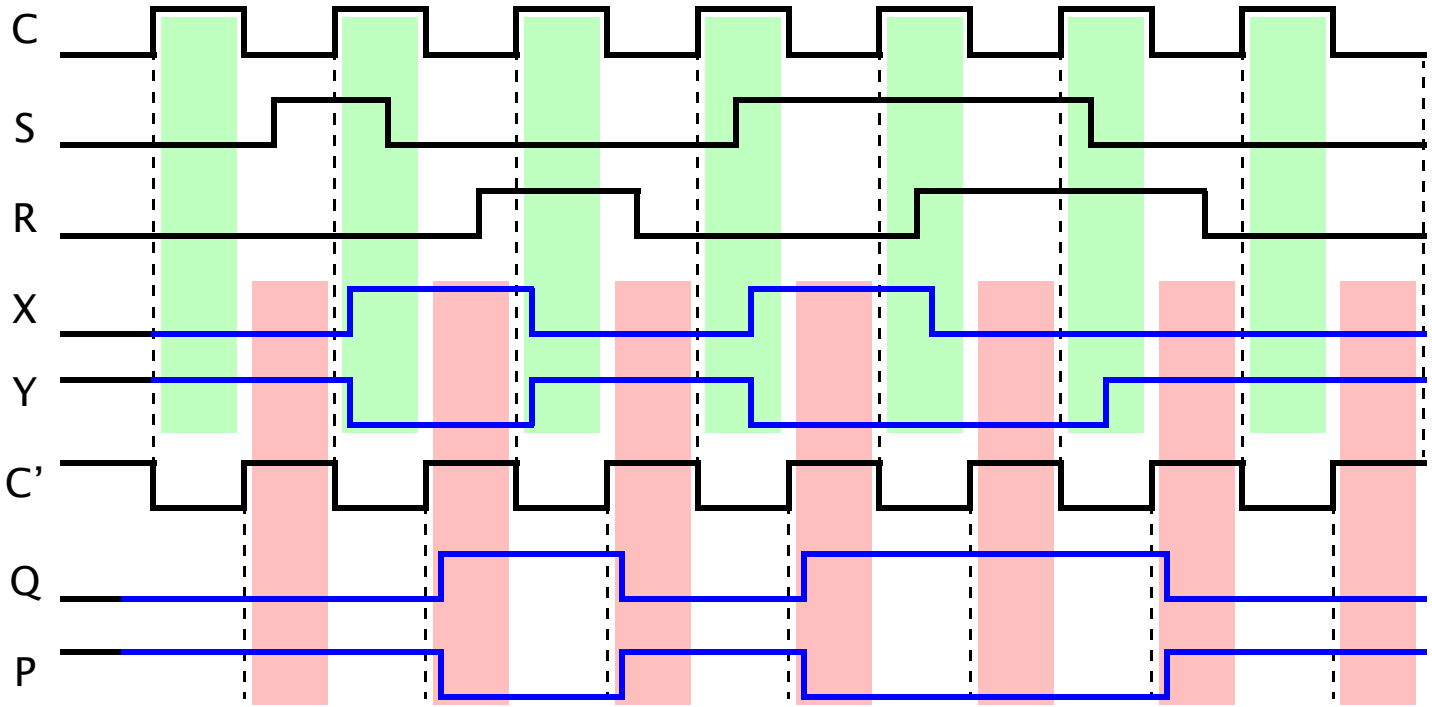
  - Entkopplung der Eingänge von den Ausgängen über
    - zweiflankengesteuerte Flipflops
    - einflankengesteuerte Flipflops

- zweiflankengesteuertes RS-Flipflop
  - Aufbau: zwei synchrone, pegelgesteuerte RS-Flipflops hintereinander geschaltet, angesteuert mit einem komplementären Taktsignal



  - Funktionsweise:
    Während der aktiven Taktphase (C=1) ist das erste Flipflop (Master) transparent, und das zweite Flipflop (Slave) gesperrt. In der nicht aktiven Taktphase (C=0) ist das genau umgekehrt.
  - Auf diese Weise sind beide Flipflops durch ein komplementäres Takt- signal wechselseitig verriegelt, und nie gleichzeitig durchgeschaltet.
  - Ein nach dem Master-Slave-Prinzip arbeitendes Flipflop kann aus synchronen, pegelgesteuerten RS-, D- und JK-Flipflops aufgebaut werden.

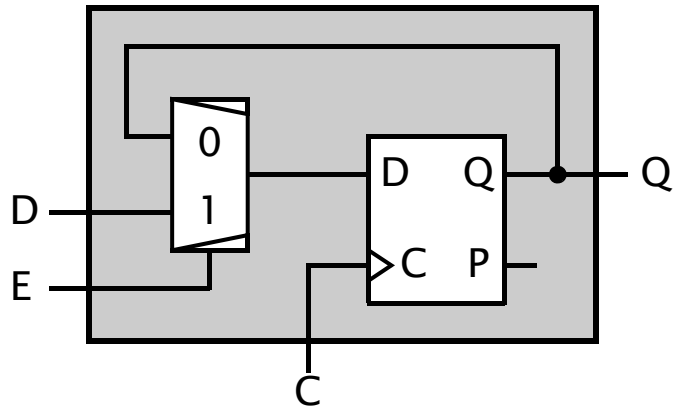
- Timing-Analyse des RS-Master-Slave-Flipflops

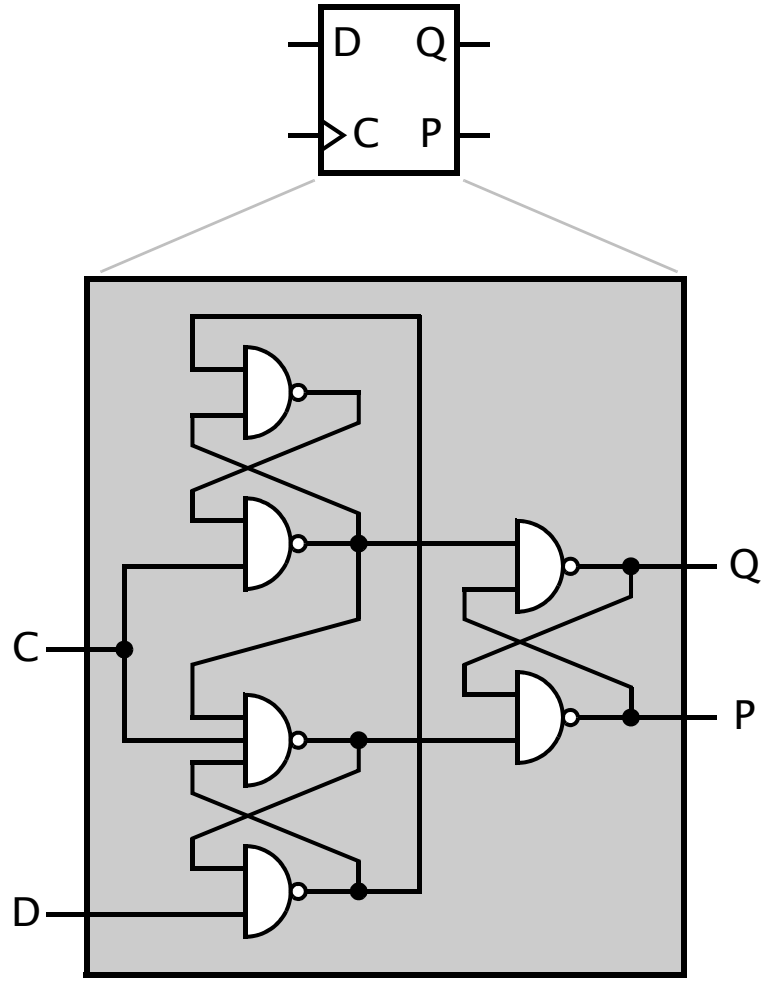


- einflankengesteuertes D-Flipflop
  - Funktionstabelle und Aufbau

C	E	D	Q^+	P^+
0	–	–	Q	P
↑	0	–	Q	P
↑	1	0	0	1
↑	1	1	1	0

  - D-Flipflop mit Enable-Signal

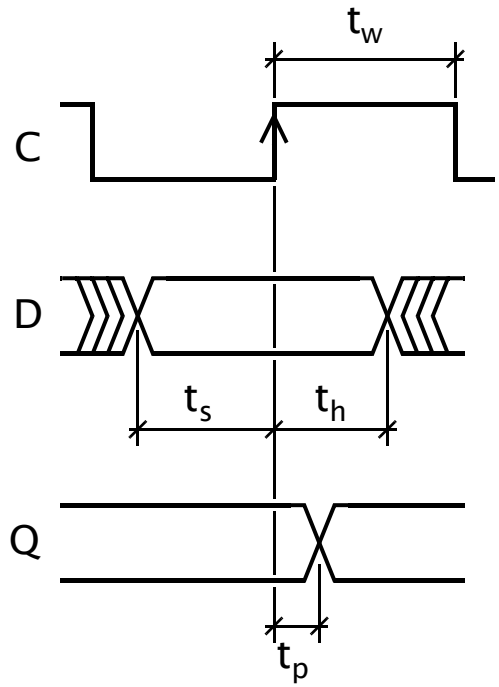




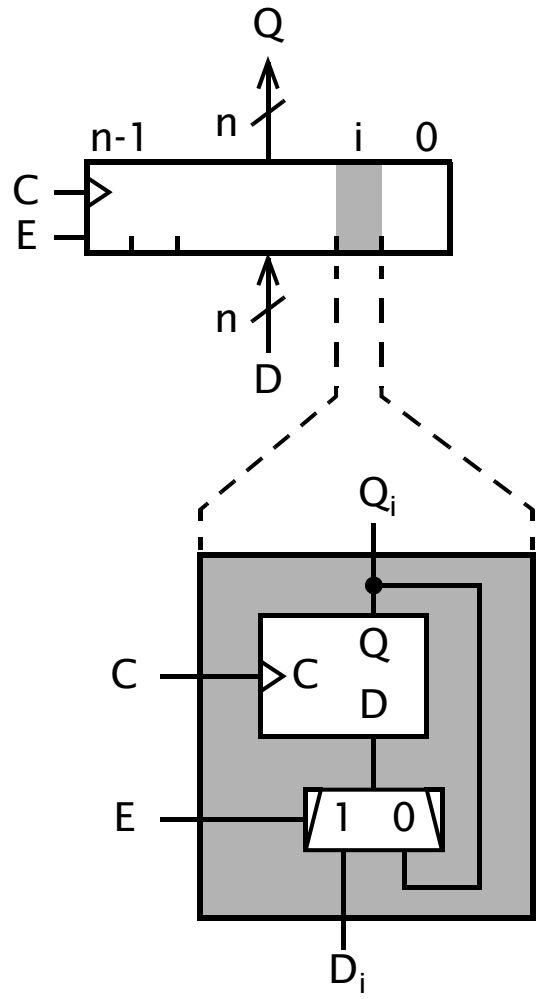
- Komplexe digitale Systeme werden vorwiegend als synchrone Systeme entwickelt.
- Typischerweise werden solche Systeme von einem globalen Taktsignal versorgt, d.h. es existiert nur ein Taktsignal, das von allen Speicherelementen gleichermaßen benutzt wird.
- Flankengesteuerte Flipflops machen den Entwurf von Schaltwerken besonders übersichtlich und werden deshalb bevorzugt in programmierbaren Logikbausteinen (CPLDs, FPGAs) eingesetzt.

- Damit Eingangssignale in Speicherelemente sicher und synchron, d.h. gleichzeitig übernehmen werden, müssen diese Signale bestimmte zeitliche Kriterien erfüllen.

- Zeitliche Kenndaten von Flipflops
  - t_w – pulse width (Impulsbreite): minimale Zeitspanne der aktiven Taktphase
  - t_s – setup time (Vorbereitungszeit): minimale Zeitspanne, in der Daten am Eingang eines Flipflops vor dem Erscheinen der Taktflanke stabil bleiben müssen.
  - t_h – hold time (Haltezeit): minimale Zeitspanne, in der Daten am Eingang eines Flipflops nach der Übernahme mit einer Taktflanke noch stabil bleiben müssen.
  - t_p – propagation time clock to output (Verzögerung): maximale Zeitspanne von der Taktflanke, mit der Daten ins Flipflop übernommen wurden, bis zum Zeitpunkt, an dem die Daten am Ausgang des Flipflops erscheinen.

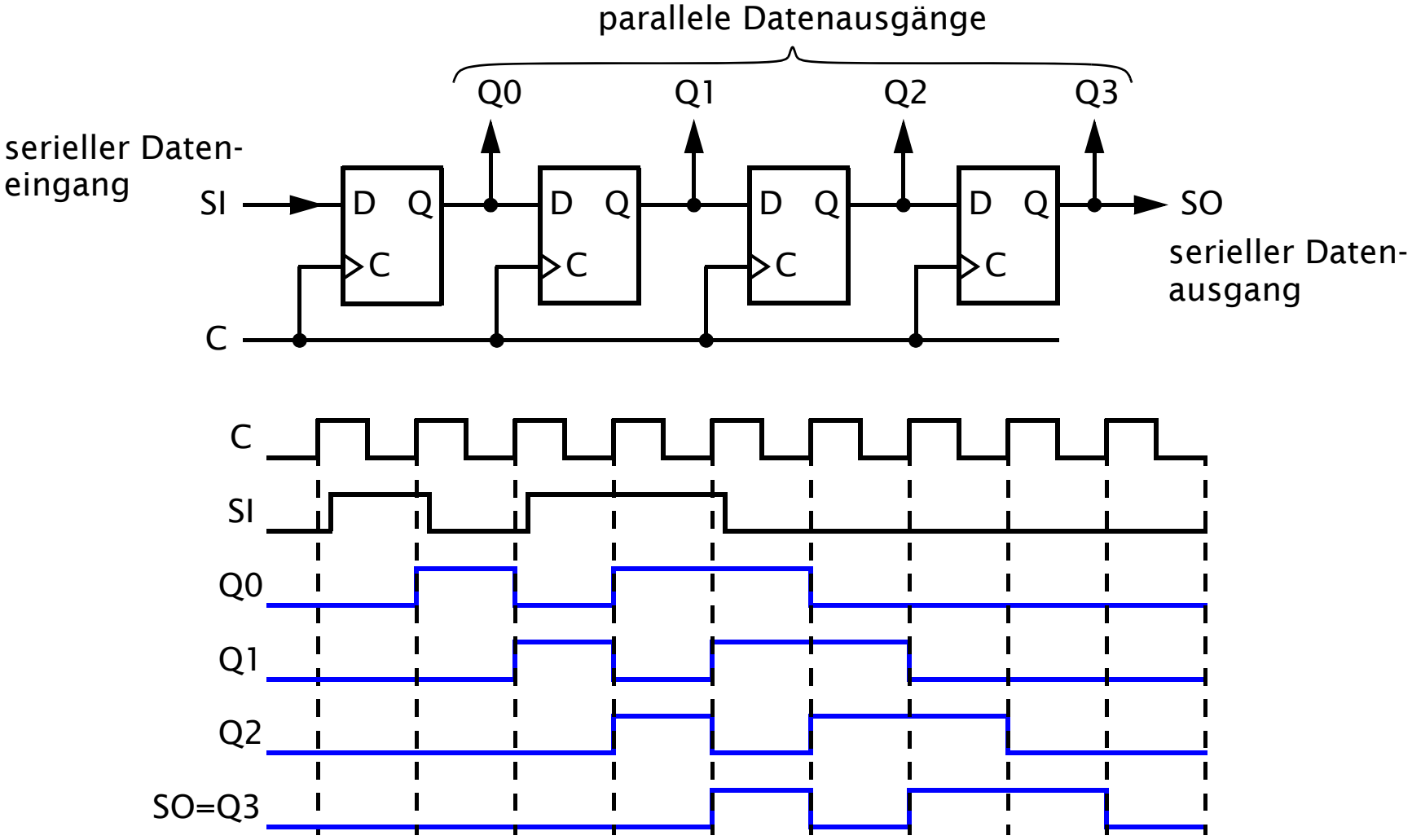


- n-Bit-Register
  - Eine parallele Anordnung von Flipflops, i.d.R zur Speicherung von binären Werten, die für eine bestimmte Zeitspanne zur Weiterverarbeitung bereit gehalten werden.
  - Im allgemeinen werden Register aus flankengesteuerten D-Flipflops mit einem Enable-Signal (E) aufgebaut.
  - Einzelne Flipflops sind über ein gemeinsames Taktsignal (C) miteinander verbunden und arbeiten völlig synchron.
  - In einem n-Bit-Register kann ein n-stelliger Binärwert gespeichert werden. Der Inhalt wird parallel ein- und ausgegeben.



- n-Bit-Schieberegister (Shift-Register)
  - Eine serielle Anordnung von Flipflops in der Form einer Kettenschaltung, d.h. der Ausgang eines Flipflops ist mit dem Dateneingang des folgenden Flipflops verbunden.
  - Der Dateneingang des erste Flipflops in der Kettenschaltung ist der serielle Dateneingang SI des Schieberegisters, und der Ausgang des letzten Flipflops ist der Datenausgang SO des Schieberegisters.
  - Alle Flipflops sind über ein gemeinsames Taktsignal miteinander verbunden. In jedem Takt werden Binärwerte um eine Position nach rechts (links) geschoben.
  - Anwendungen:
    - Seriell-/Parallelwandlung
    - Erzeugung von Pseudozufallszahlen (LFSR – linear feedback shift register, linear rückgekoppeltes Schieberegister)
    - zyklische Redundanzprüfung (CRC – Cyclic Redundancy Check)

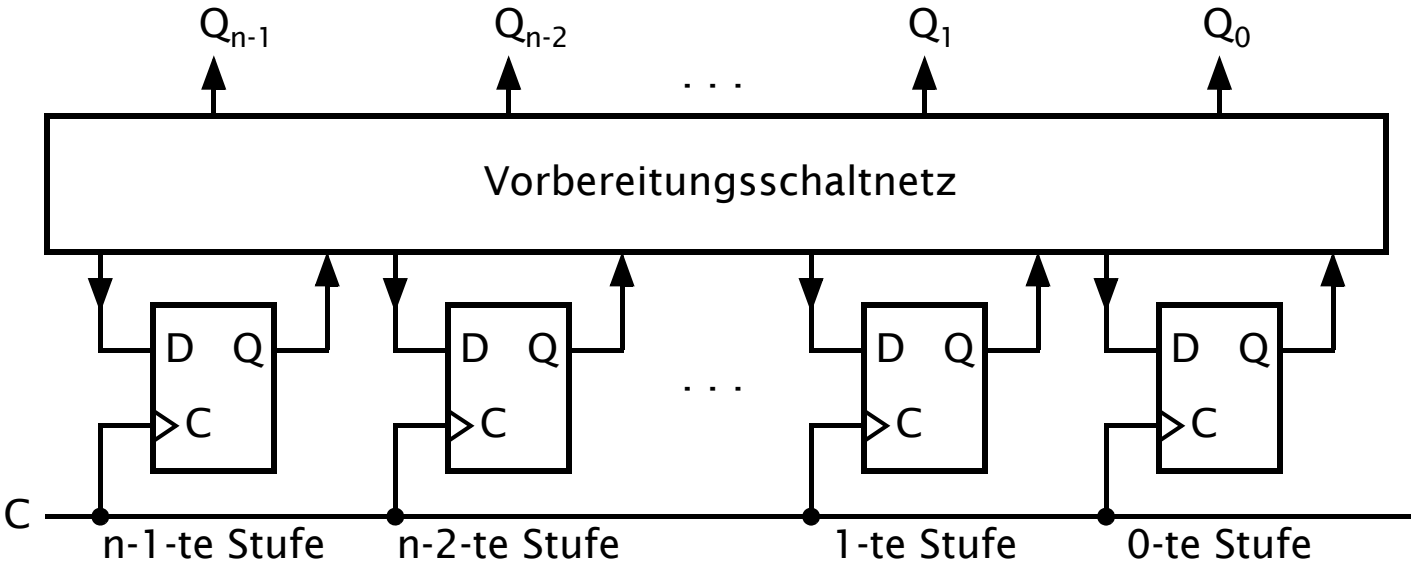
- 4-Bit-Seriell-/Parallel-Schieberegister



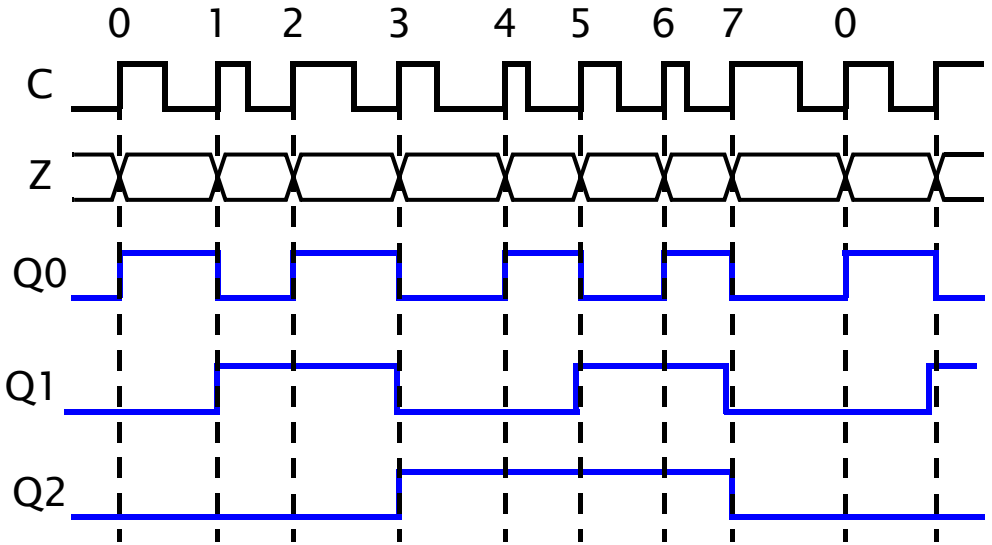
- Zähler
  - Schaltwerke, die Takt-/Steuerimpulse zählen können.
  - Schaltwerke, die eine eindeutige Zuordnung von Zählimpulsen am Eingang zu internen Flipflop-Zuständen ermöglichen.
  - Die internen Zustände müssen dabei nicht unbedingt einer gängigen, bekannten Zahlendarstellung entsprechen.
  - Zählfunktionen sind in vielen Steuerungsvorgänge als Teilaufgaben enthalten. Typische Aufgaben dieser Art sind:
    - die n-malige Wiederholung eines Vorganges
    - die Auslösung einer Reaktion nach n-maligem Auftreten eines bestimmten Ereignisses
    - präzise Zeit-/Positionsmessung
- Klassifikation von Zählern
  - Nach der Art der Darstellung des Zählstandes
    - Dualzähler: der Zählerstand wird als Dualzahl dargestellt,
    - BCD-Zähler: der Zählerstand wird pro Dezimalstelle separat dargestellt,
    - Darstellungen nach anwendungsspezifischen Codierungen.

- Klassifikation von Zählern
  - Nach der Art der Ansteuerung
    - synchrone Zähler: alle FFs werden gleichzeitig mit Zählimpulsen versorgt,
    - asynchrone Zähler: mindestens ein FF enthält ein Taktsignal, das innerhalb des Schaltwerks (Zähler) generiert wurde,
    - semisynchrone Zähler: asynchrone Zähler, die abschnittsweise synchron arbeiten, z.B. bei der Kaskadierung von Zählern,
    - programmierbare Zähler: der Zählerstand läßt sich parallel laden und so von einem neu definierten Zählerstand weiter zählen.
  - Nach der Zählrichtung
    - Vorwärtszähler mit Q := (Q + 1) mod m
    - Rückwärtszähler mit Q := Q – 1
    - umschaltbare Vor-/Rückwärtszähler.
  - Nach der Art der Anordnung der Flipflops
    - Ringzähler: Schieberegister, in dem der Inhalt zyklisch verschoben wird, z.B. 4-Bit-Ringzähler: 1000, 0100, 0010, 0001, 1000, ...,
    - Johnson-Zähler: besondere Form des Ringzählers, z.B. 3-Bit-Johnson-Zähler: 000, 100, 110, 111, 011, 001, 000, ...

- Grundstruktur eines synchronen Zählers
  - Im allgemeinen bestehen synchrone Zähler aus mehreren Stufen.
  - Jede Stufe wird durch ein flankengesteuertes D-Flipflop realisiert.
  - Ein zusätzliches (Vorbereitungs-)Schaltnetz übernimmt logische Verknüpfungen zur Ansteuerung der Flipflop-Eingänge und sofern notwendig der Ausgangsvariablen Q_{n-1}, Q_{n-2}, ..., Q_1, Q_0.

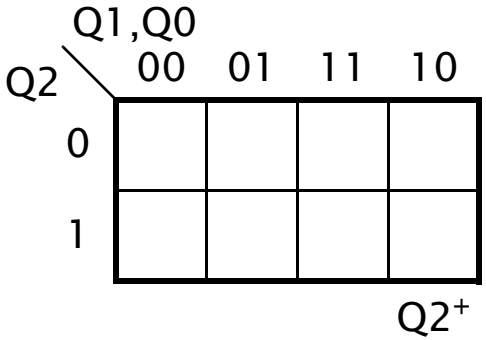


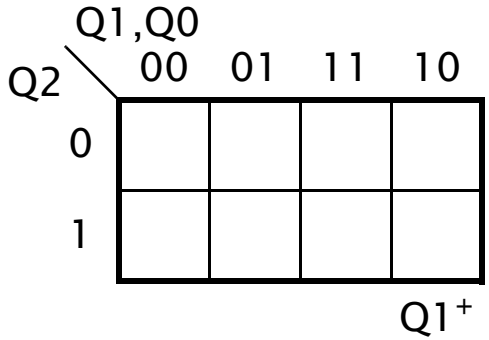
- Entwurf eines 3-Bit-Vorwärtszählers (als Dualzähler)
  - Ein Zähler soll steigende (positive) Flanken eines Eingangssignals C erfassen. An den Ausgängen Q2, Q1 und Q0 des Zählers Z soll der momentane Wert angezeigt werden.
  - Impulsplan und Funktionstabelle

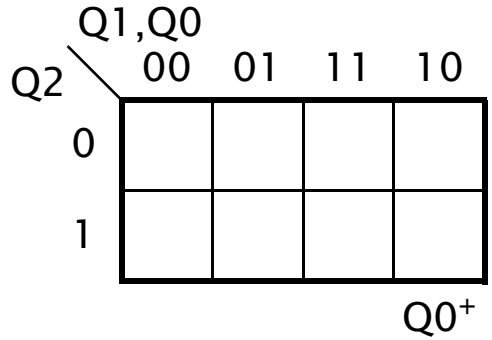


Nr	Q2	Q1	Q0	$Q2^+$	$Q1^+$	$Q0^+$

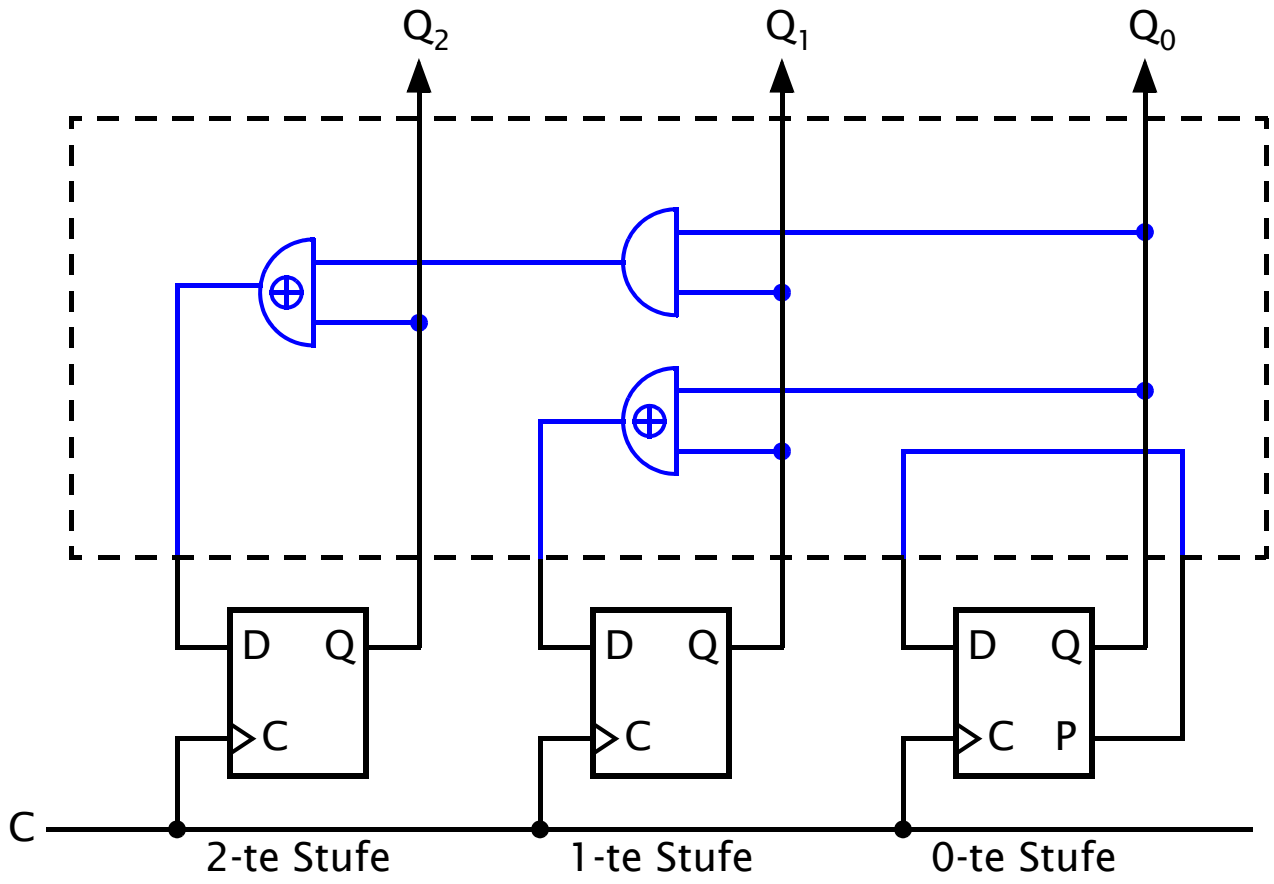
- Entwurf eines 3-Bit-Vorwärtszählers







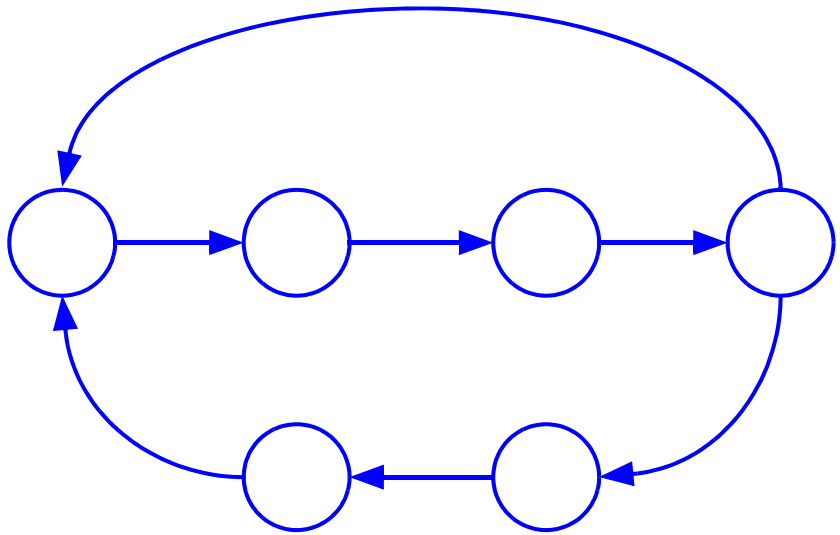
- Entwurf eines 3-Bit-Vorwärtszählers



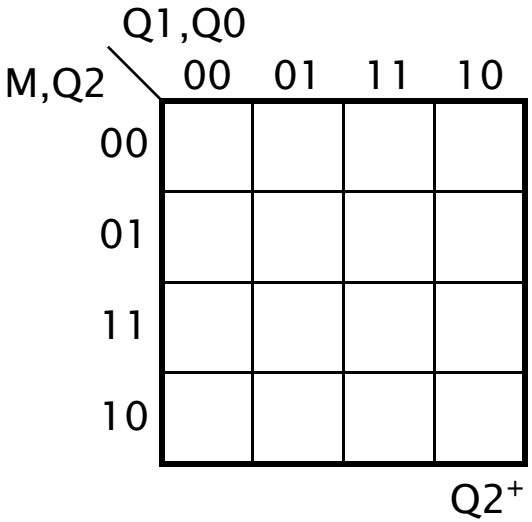
- Modulo-m-Zähler
  - Die aus der Mathematik bekannte Modulo-Funktion (mod) liefert den Rest aus der Divison zweier ganzer Zahlen,
  - Ein Modulo-m-Zähler ist ein Zähler, der i.d.R. zyklisch von 0 bis m-1 vorwärts zählt, d.h. nachdem der Zählerstand den Wert m-1 erreicht hat, wird der Zählvorgang wieder bei 0 fortgesetzt.
    Der Zählerstand bleibt immer kleiner als m.
    z.B. ein Modulo-5-Zähler liefert nur die Werte: 0, 1, 2, 3, 4
  - Ein Modulo-m-Zähler läßt sich aus einem n-Bit-Dualzähler mit der Bedingung $n < \log_2(m)$ aufbauen.
    z.B. aus einem 4-Bit-Dualzähler kann man folgende Modulo-m-Zähler aufbauen: mod 2, mod 3, mod 4, ..., mod 14 und mod 15.
  - Die Bezeichnung „Modulo-m-Zähler“ wird manchmal auch in Verbindung mit Zählern verwendet, die nicht als Dualzähler arbeiten.
  - Die Struktur eines Modulo-m-Zählers basiert auf der Grundstruktur eines synchronen Zählers.

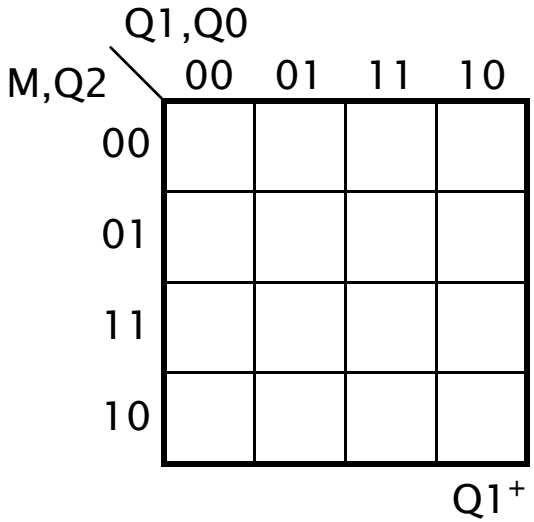
- Entwurf eines einfachen Modulo-m-Zählers
  - Ein Zähler soll in der Abhängigkeit von einem Steuersignal M entweder modulo 6 (M=0) oder modulo 4 (M=1) zählen.
  - Funktionstabelle und Zustandsgraph

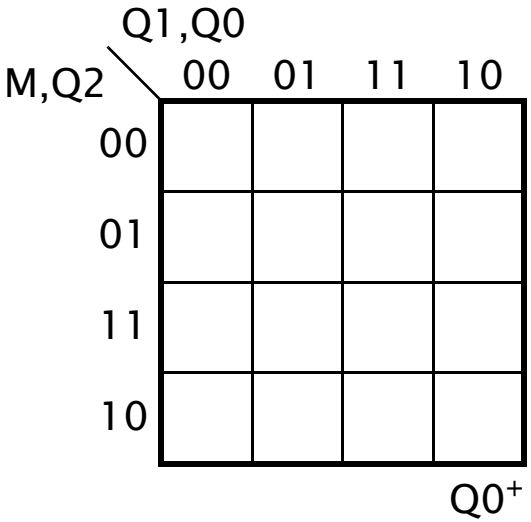
Nr	M	Q2	Q1	Q0	$Q2^+$	$Q1^+$	$Q0^+$



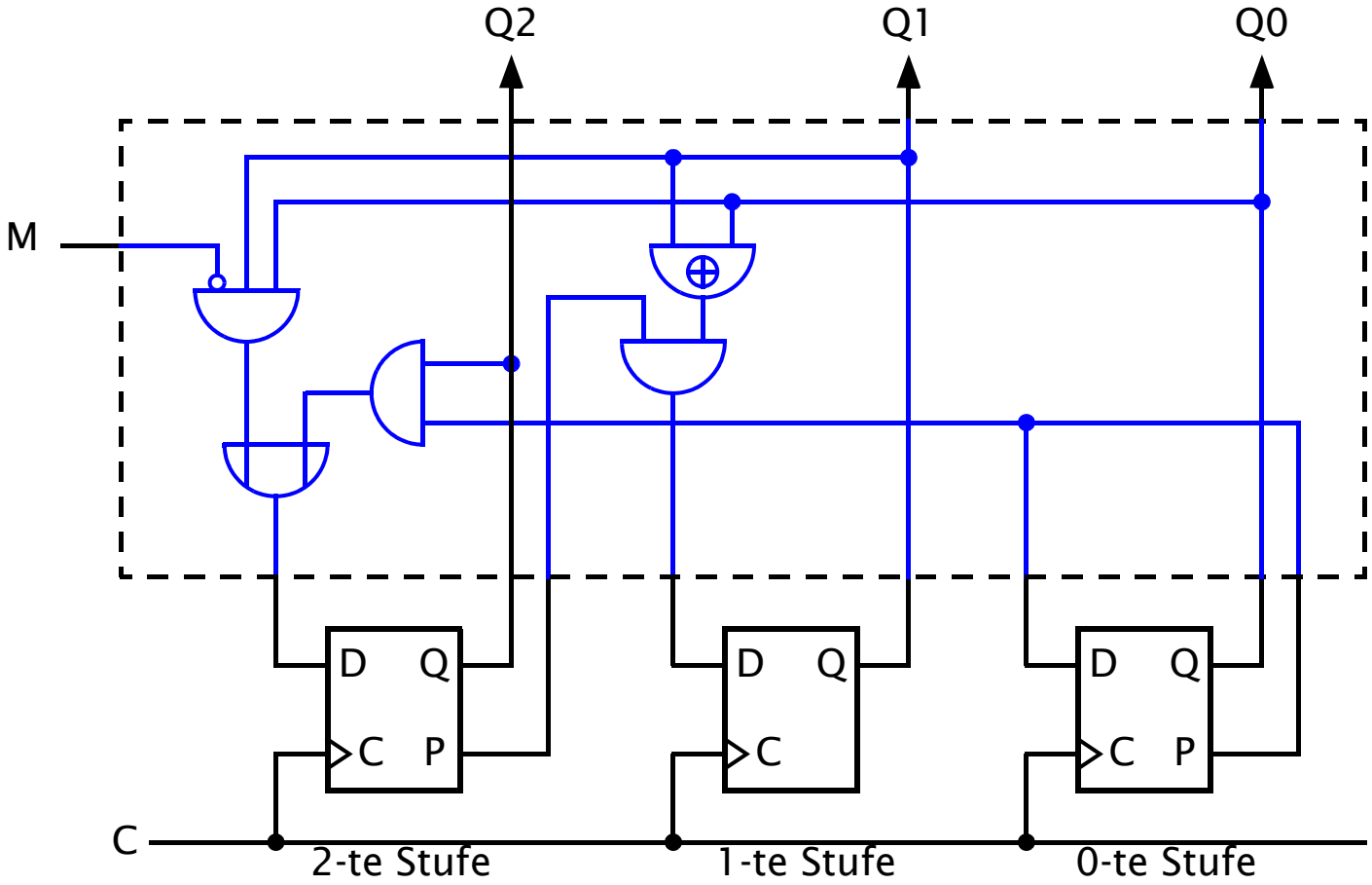
- Entwurf eines einfachen Modulo-m-Zählers





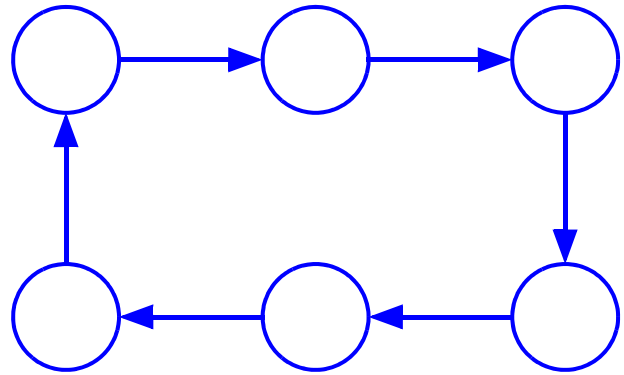


- Entwurf eines einfachen Modulo-m-Zählers



- Entwurf eines selbstkorrigierenden Zählers
  - Ein zyklischer, selbstkorrigierender 3-Bit-Zähler mit einer einschrittig codierten Zählsequenz {000_2, 001_2, 011_2, 010_2, 110_2, 100_2} soll auf der Basis von D-Flipflops entworfen werden.
  - Funktionstabelle und Zustandsgraph

Nr	Q2	Q1	Q0	$Q2^+$	$Q1^+$	$Q0^+$

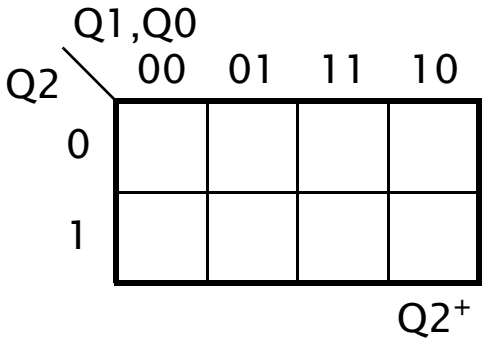


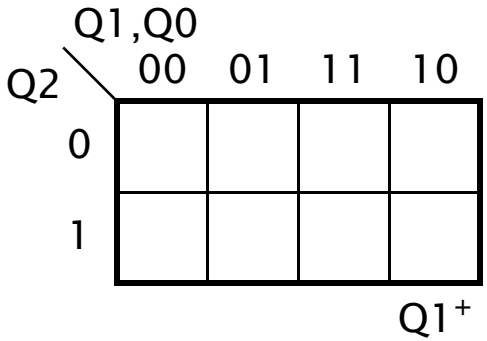
- Entwurf eines selbstkorrigierenden Zählers
  - Ein selbstkorrigierender Zähler zeichnet sich dadurch aus, daß er in der Lage ist, ungültige, fehlerhafte Zählzustände automatisch und selbständig zu korrigieren.
  - Als fehlerhafte Zählzustände bezeichnet man solche Zählzustände, die der Zähler unter normalen Umständen nicht annehmen darf, also solche, die in der Sequenz der „gültigen“ Zählzustände nicht aufgelistet sind.
  - Fehlerhafte Zählzustände können sich aufgrund von elektrischen Störungen in digitalen Systemen oder nach einer Einschaltphase (Power-On) einstellen.

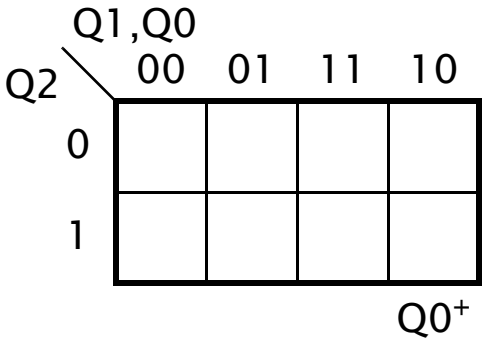
  - Beim Entwurf selbstkorrigierender Zähler ist zu beachten, daß fehlerhafte Zählzustände nach Möglichkeiten immer unter dem Aspekt des geringsten Realisierungsaufwands korrigiert werden sollen.

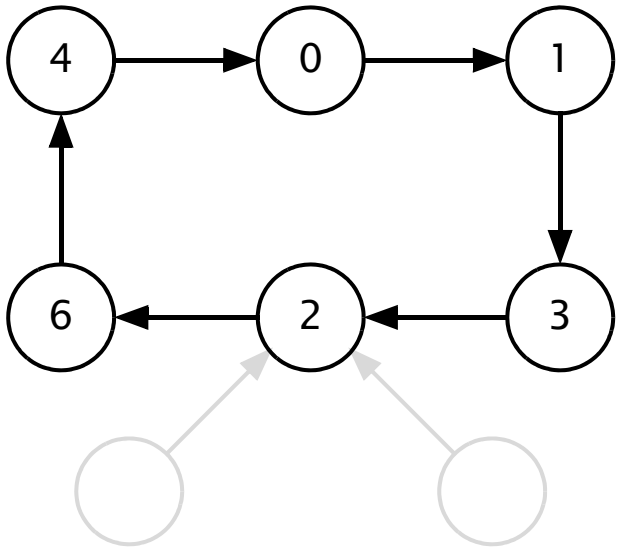
- Regelwerk zum Entwurf eines selbstkorrigierenden Zählers
  1. Für die „gültige“ Zählsequenz wird eine Funktionstabelle aufgestellt und diese dann in entsprechende KV-Diagramme überführt. Fehlerhafte Zählzustände bleiben vorläufig unberücksichtigt.
  2. Der geringste Realisierungsaufwand bei der Korrektur fehlerhafter Zählzustände läßt sich mit Hilfe von KV-Diagrammen ermitteln. Dort werden noch unbestimmte Felder so mit Nullen und Einsen belegt, daß dadurch bei der Minimierung möglichst die größten Gruppen gebildet werden können.
  3. Anschließend wird mit Hilfe des Zustandsgraphen geprüft, ob die dadurch erzeugte Belegung der fehlerhaften Zählzustände dazu führt, daß diese einen Übergang in die Sequenz der „gültigen“ Zählzustände haben.
     Wenn das nicht der Fall ist, ist die Belegung zu korrigieren und mit dem Schritt 2 fortzuführen.

- Entwurf eines selbstkorrigierenden Zählers



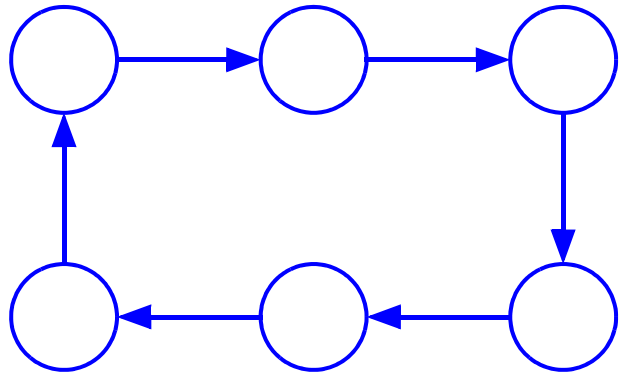




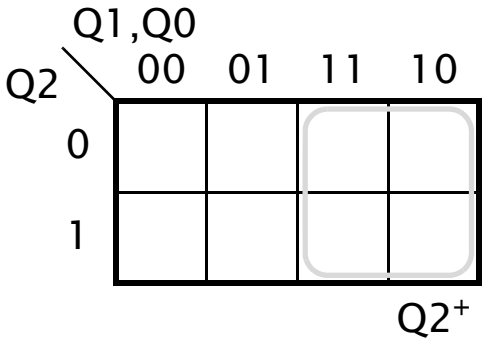


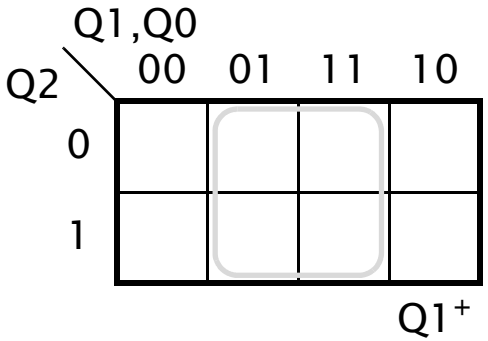
- Entwurf eines selbstkorrigierenden Zählers
  - Auf der Basis von D-Flipflops ist ein zyklischer, selbstkorrigierender 3-Bit-Johnson-Zähler mit folgender dezimalcodierten Zählsequenz {0, 1, 3, 7, 6, 4} zu entwerfen.
  - Funktionstabelle und Zustandsgraph

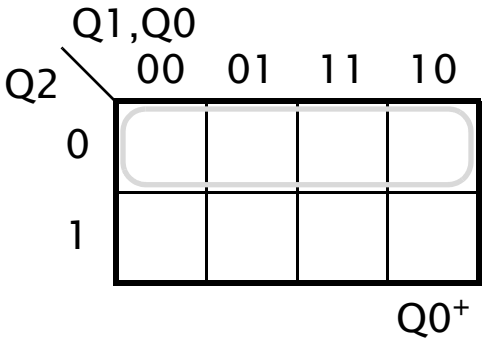
Nr	Q2	Q1	Q0	$Q2^+$	$Q1^+$	$Q0^+$

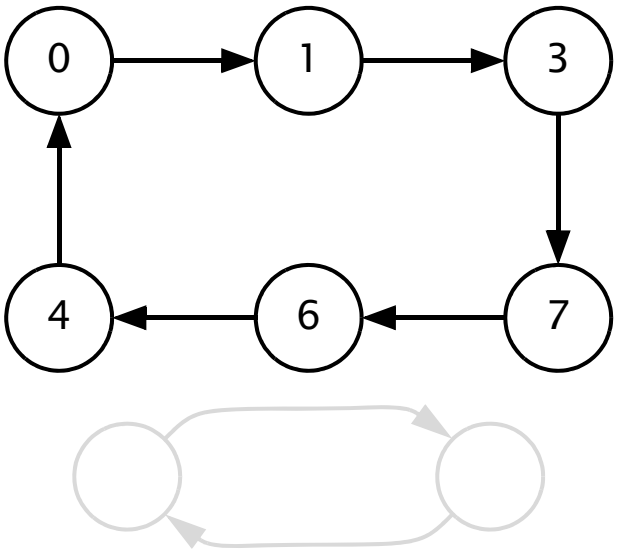


- Entwurf eines selbstkorrigierenden Zählers

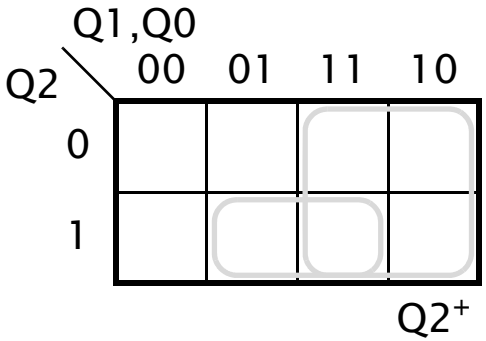


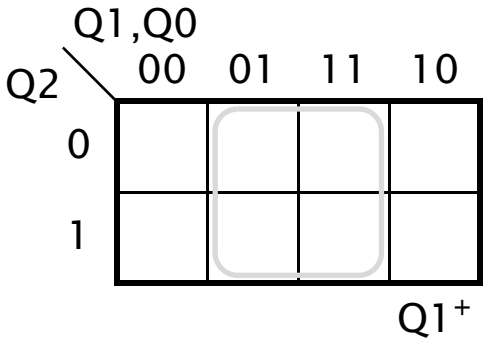


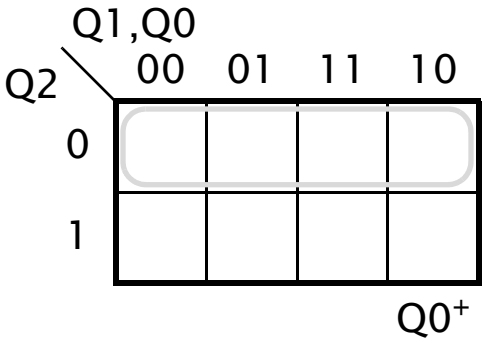


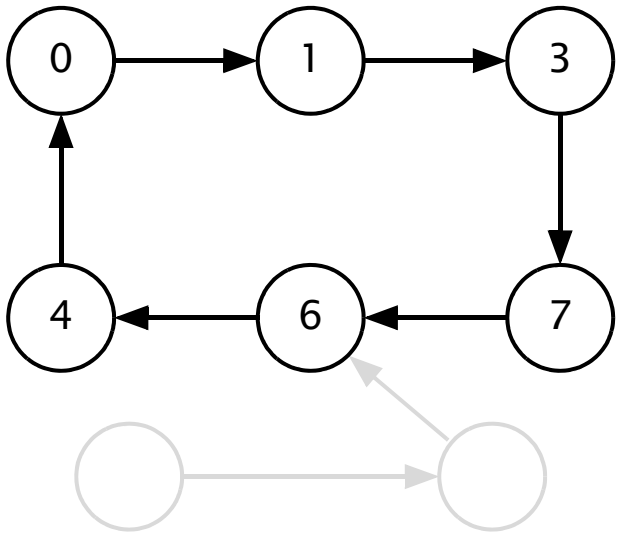


- Entwurf eines selbstkorrigierenden Zählers









- Ein endlicher (boolescher) Automat (Finite State Machine, FSM) ist ein mathematisches Modell für eine Vorrichtung mit einer endlichen Anzahl interner Zustände, die eine Folge von booleschen Eingangswerten einliest, verarbeitet und eine Folge von booleschen Ausgangswerten erzeugt.
- Ein endlicher Automat ist durch 6-Tupel
  $EA = (E, A, Z, Z_0, f, g)$ definiert, wo
  - E endliche Eingabemenge (Eingangsalphabet, Menge aller möglichen Werte, die Eingangsvariablen annehmen können)
  - A Ausgabemenge (Ausgangsalphabet, Menge aller möglichen Werte, die Ausgangsvariablen annehmen können)
  - Z endliche Zustandsmenge
  - Z_0 Anfangszustand
  - f (Zustands-)Übergangsfunktion für den Folgezustand
  - g Ausgabefunktion (abhängig vom Automatentyp)

- Aufzugsteuerung modelliert als Automat
  - Eingabemenge E
    - vier Eingangsvariablen u, d, e0 und e1
    - jede Eingangsvariable kann entweder 0 oder 1 annehmen
    - zusammen ergeben sie also 16 mögliche Eingangskombinationen

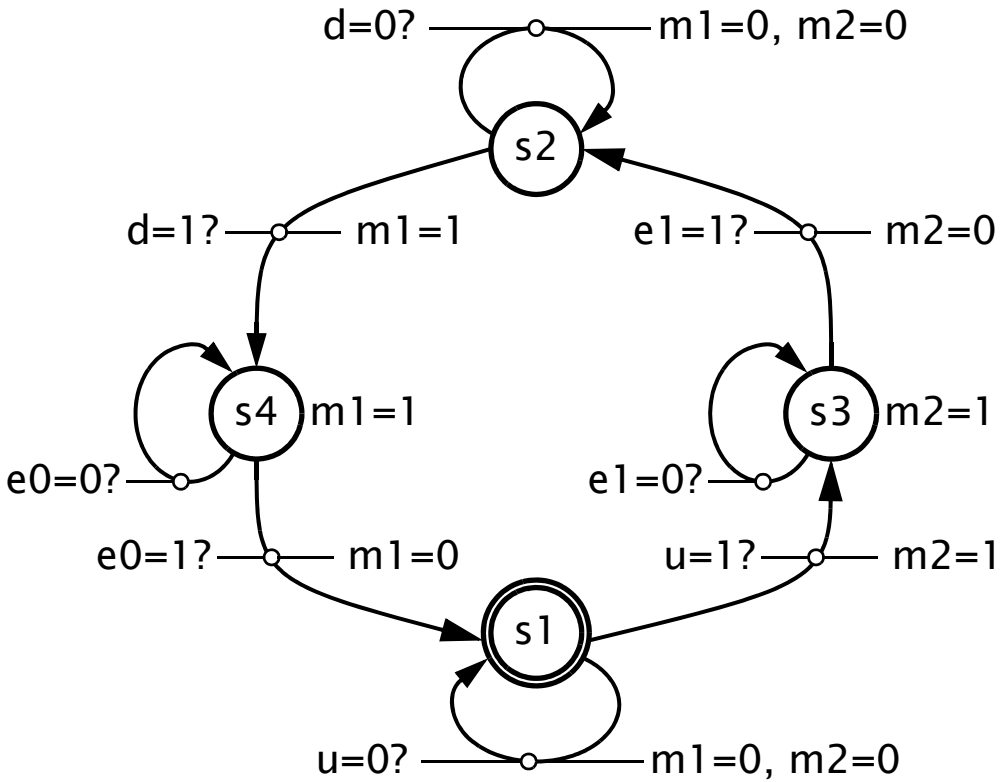
    $E(u, d, e0, e1) = \{(0000)_2, (0001)_2, ..., (1110)_2, (1111)_2\}$
  - Ausgabemenge A
    - zwei Ausgangsvariablen m1 und m2
    - jede Ausgangsvariable kann entweder 0 oder 1 annehmen
    - daraus ergeben sich 4 mögliche Ausgangskombinationen

    $A(m1, m2) = \{(00)_2, (01)_2, (10)_2, (11)_2\}$
  - Zustandsmenge
    - eine Zustandsvariable, die 4 symbolisch codierte Zustände annehmen kann

    $Z = \{s1, s2, s3, s4\}$
  - Anfangszustand

    $Z_0 = \{s1\}$

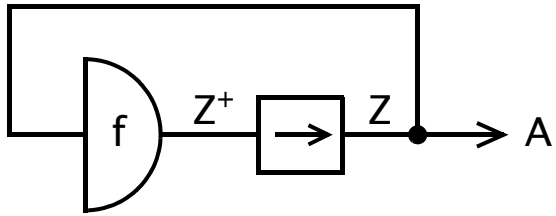
- Aufzugsteuerung modelliert als Automat
  - Tabellarische Notation der Übergangs- und Ausgabefunktionen
  - Übergangsfunktion $Z^+ = f(Z, E)$
  - Ausgabefunktionen $m1 = g1(Z, E)$ und $m2 = g2(Z, E)$



	E(u, d, e0, e1)						
Z	u	d	e0	e1	Z^+	m1	m2

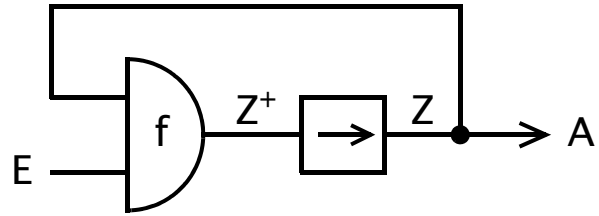
- Automatenmodelle

Autonomer Automat



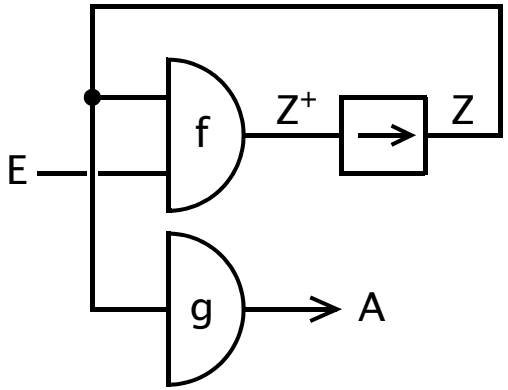
$Z^+ = f(Z), \quad A = Z$

Medwedjew-Automat



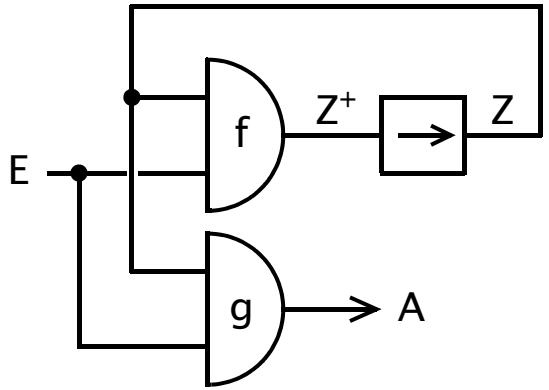
$Z^+ = f(Z, E), \quad A = Z$

Moore-Automat



$Z^+ = f(Z, E), \quad A = g(Z)$

Mealy-Automat



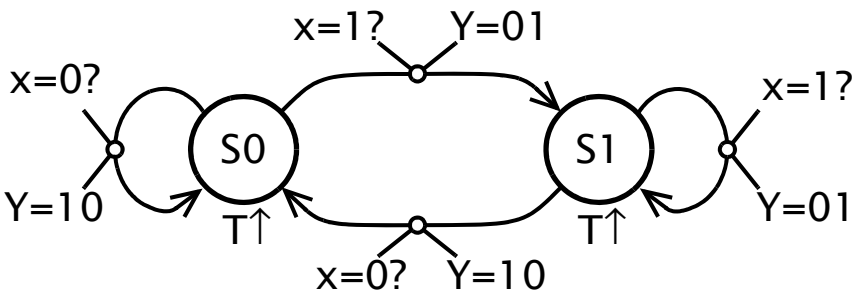
$Z+ = f(Z, E), \quad A = g(Z, E)$

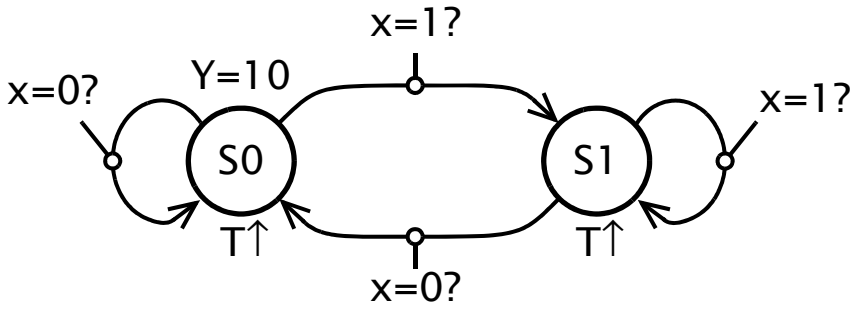
- Das Verhalten eines Automaten wird im allgemein durch Übergangs- und Ausgabefunktionen beschrieben.

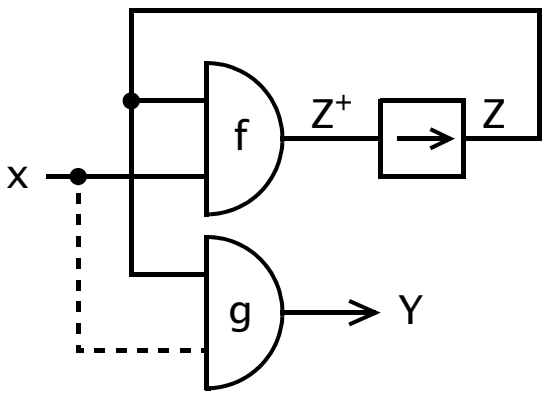
- Die Übergangsfunktion $Z^+ = f(E, Z)$ gibt (abgesehen von autonomen Automaten) in Abhängigkeit von den momentanen Eingangswerten E und den aktuellen Werten der inneren Zustandsvariablen Z an, welche Werte die inneren Zustandsvariablen zum Folgezeitpunkt annehmen.

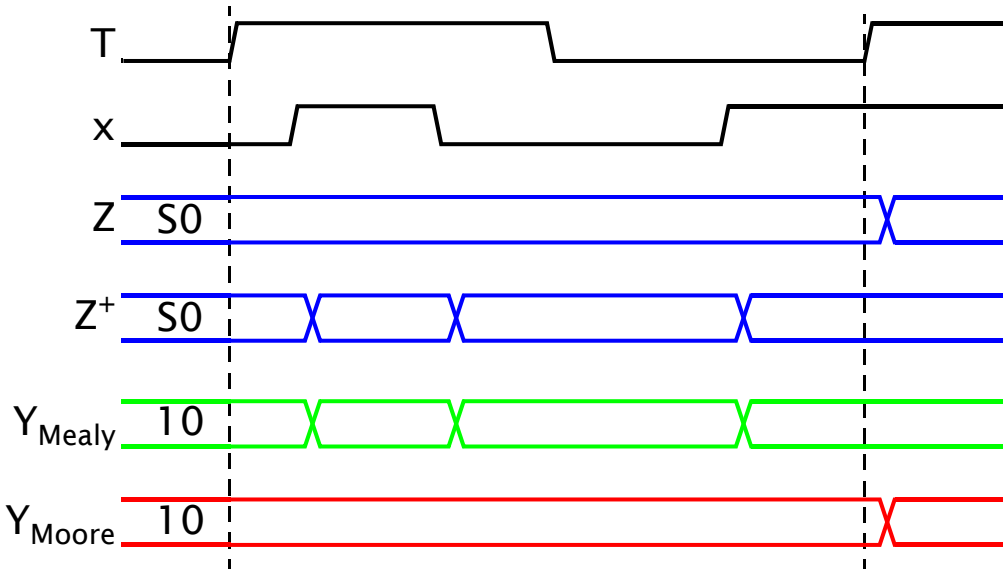
- Die Ausgabefunktion $A = g(E, Z)$ bestimmt die Werte der Ausgangsvariablen in Abhängigkeit von den aktuellen Werten der inneren Zustandsvariablen Z und eventuell von den momentanen Eingangswerten E.

- Timing Analyse von Mealy- und Moore-Automaten



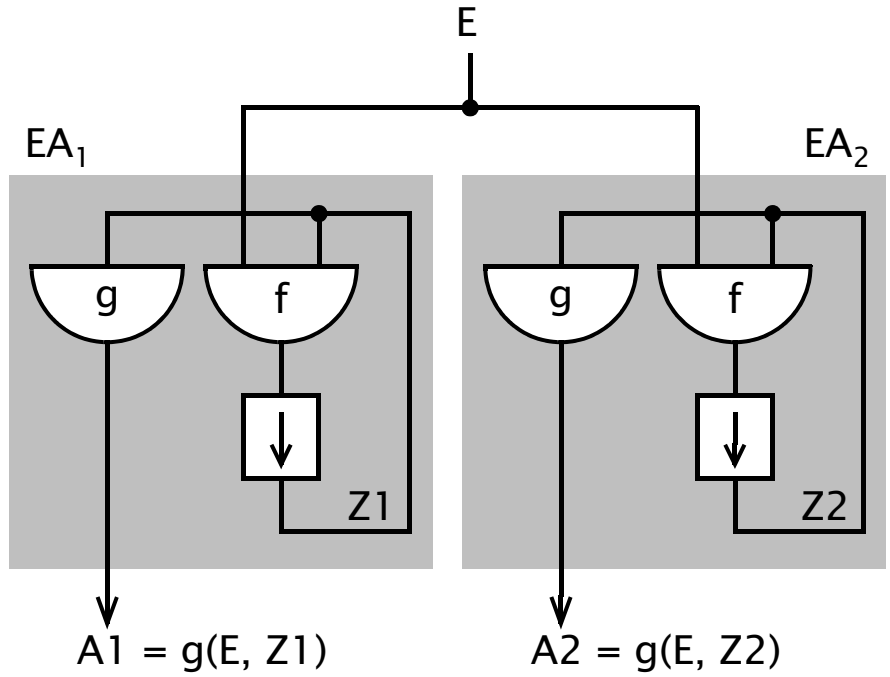




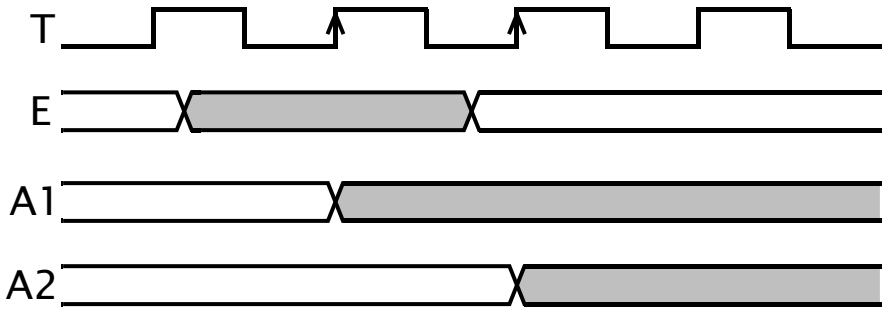


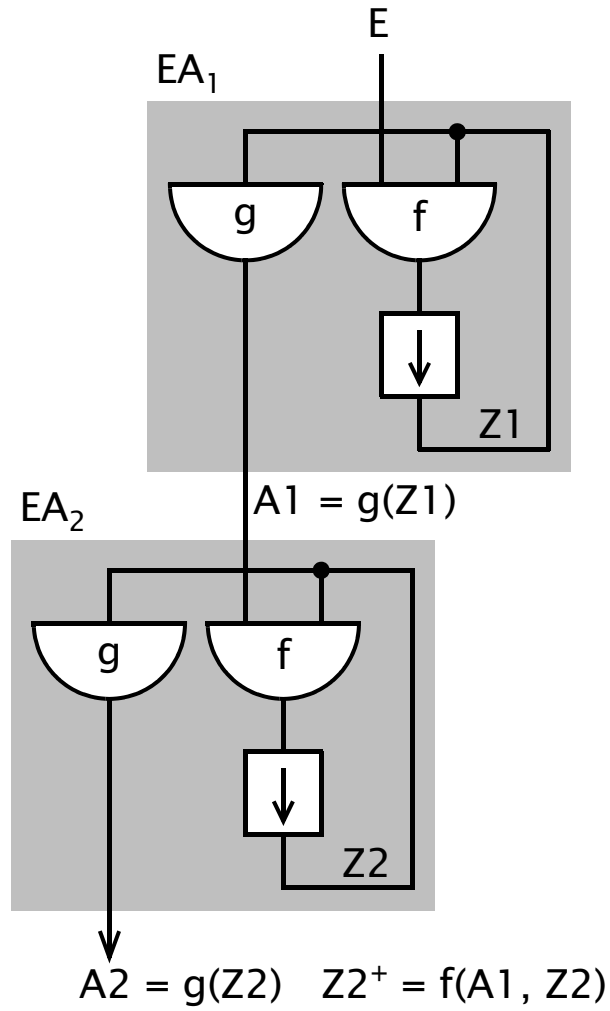
- Komposition von Automaten (Schaltwerken)
  - zwei oder mehrere endliche Automaten EA_i können auf verschiedene Arten miteinander kombiniert werden.

- Parallele Komposition:
  - zwei verschiedene Automaten EA_1 und EA_2 generieren dann verschiedene Ausgaben A1 und A2, werden aber von denselben Eingaben E gespeist.
- Sequentielle Komposition:
  - Hintereinanderschaltung: die Ausgabe des ersten Automaten wird als Eingabe für den zweiten Automaten verwendet.

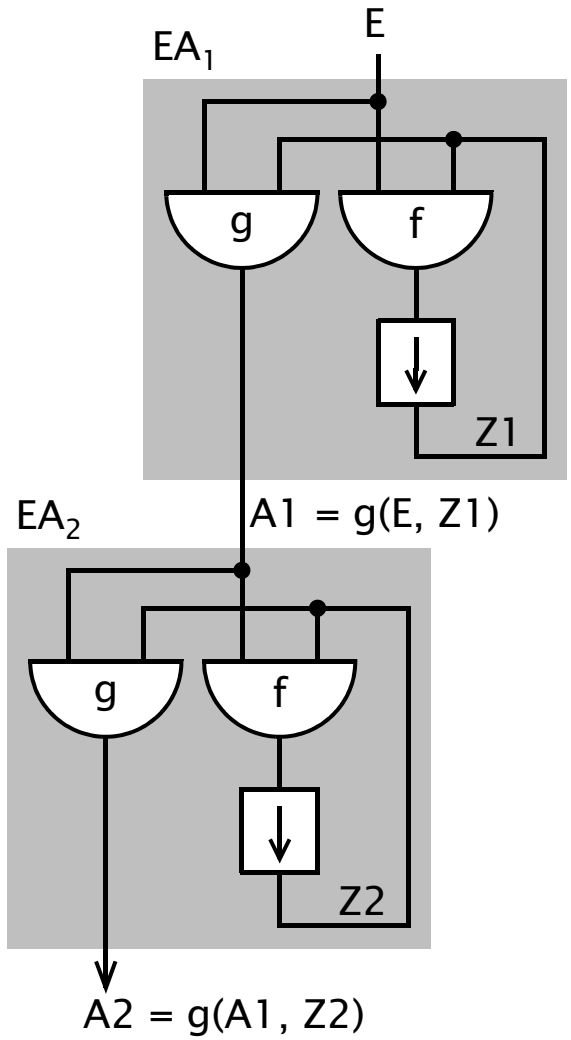


- sequentielle Komposition zweier Moore-Automaten
  - Die Ausgabe A1 aus dem ersten Moore-Automaten EA_1 ist erst nach einem Taktzyklus verfügbar.
  - Somit steht die Eingabe für den zweiten Moore-Automaten EA_2 erst für den zweiten Taktzyklus zur Verfügung.
  - Die Ausgabe A2 aus dem zweiten Moore-Automaten EA_2 ist gegenüber der Eingabe E um zwei Taktzyklen verzögert.

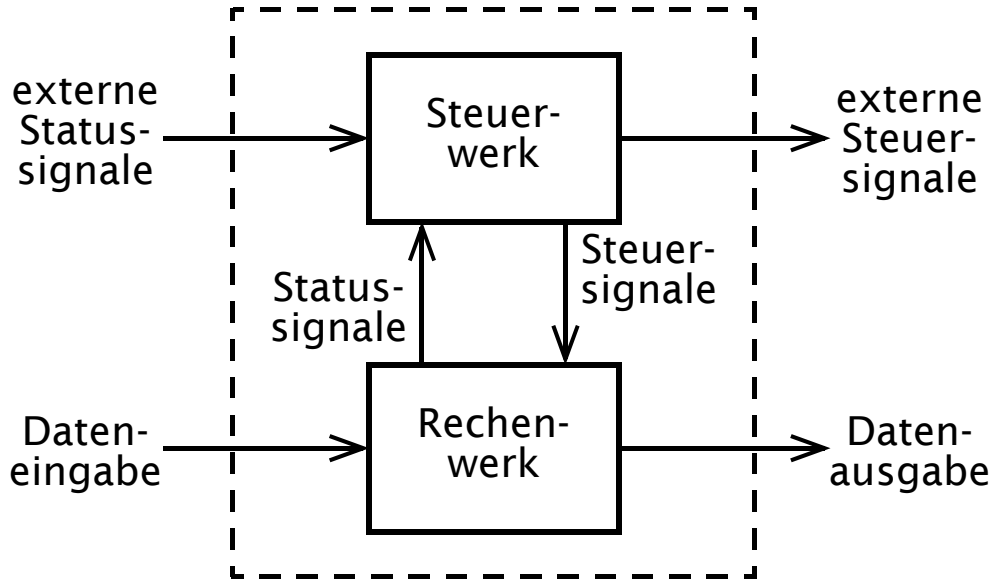




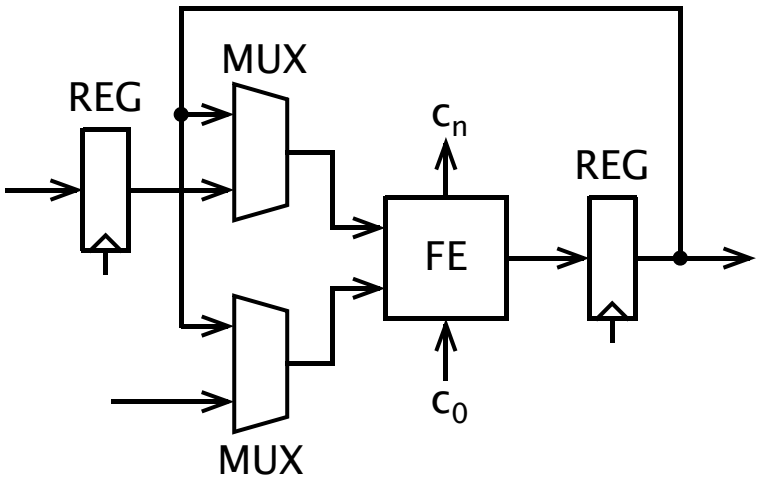
- sequentielle Komposition zweier Mealy-Automaten
  - Die Eingabe E des ersten Mealy-Automaten EA_1 wird durch das Ausgabeschaltnetz g geführt und dient als Eingabe (A1) für den zweiten Mealy-Automaten EA_2.
  - Durch die unmittelbare Abhängigkeit der Ausgabe von den Eingabewerten kommt im allgemeinen zu einer Verlängerung der kombinatorischen Pfade (E→A1→A2).
  - Eine sequentielle Komposition mehrerer Mealy-Automaten muß wegen der Gatterlaufzeiten in den Ausgabeschaltnetzen mit einem um so langsameren Takt t_{cyc} betrieben werden, je mehr solche Automaten hintereinander geschaltet sind ($t_{cyc} \geq \Sigma t_{pd}(g_i)$).



- Modularisierung bedeutet die logische Zerlegung komplexer Systeme in einfachere, leichter zu überschauende und somit auch leichter zu verstehende Subsysteme.
- Aufteilung eines digitalen Systems in
  - eine steuernde Einheit (Steuerwerk, Steuerpfad)
  - eine gesteuerte Einheit (Rechenwerk, Operationswerk, Datenpfad)
  - oft kein Steuerwerk in datenflußdominanten Anwendungen
  - oft kein Rechenwerk in Steuerungsaufgaben



- Das Rechenwerk besteht aus
  - Schaltnetzen/Funktions-einheiten (FE) zur Realisierung arithmetischer und logischer Operationen
  - Registern (REG) zur Aufnahme von Operanden und Ergebnissen
  - Multiplexern (MUX) als steuer-bare Verbindungen zwischen Registern und Schaltnetzen.

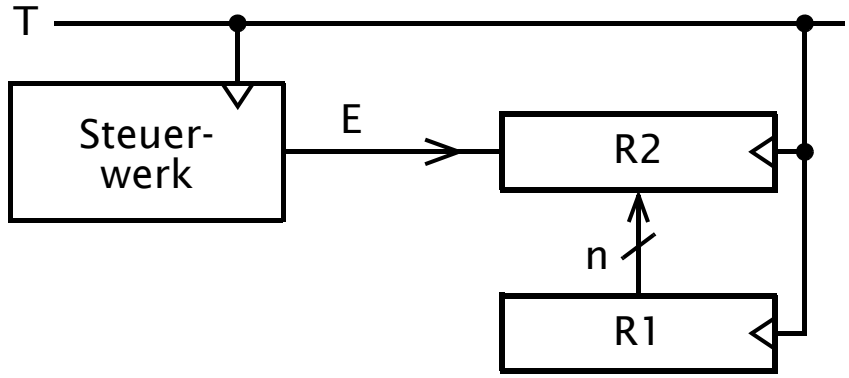


- Das Steuerwerk hat die Aufgabe,
  - Statussignale (z.B. Übertragsbit aus einer Addition) aus dem Rechenwerk auszuwerten, und anhand eines Steuerprogramms daraus Steuersignale für das Rechenwerk zu generieren:
    - Enable-Signale für Register
    - Select-Signale für Multiplexer
    - Function-Signale für universelle Funktionseinheiten

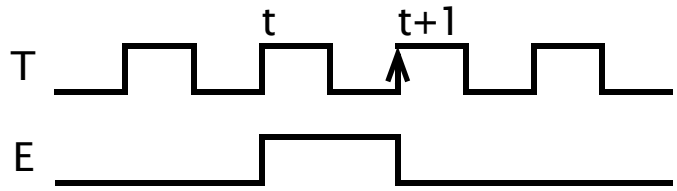
- Register-Transfer-Operationen
  - Ein digitales System verarbeitet Informationen, die in Registern gespeichert sind.
  - Diese Informationen werden oft durch arithemtisch-logische Operationen (Mikrooperationen) verändert und wieder in Registern zurückgeschrieben.
  - Daher kann man ein digitales System beschreiben durch:
    - Menge der Register und ihre Funktionen (laden, löschen, schieben),
    - Menge der möglichen Mikrooperationen (addieren, subtrahieren),
    - Steuerung, die die Ausführung der Mikrooperationen veranlaßt.

  - Der Datentransfer von einem Register zu einem anderen Register wird symbolisch mit einem Zuweisungsoperator := oder ← notiert.
  - Auf der linken Seite des Zuweisungsoperators steht normalerweise ein n-stelliges Zielregister, rechts davon ein boolescher Ausdruck oder eine arithemtische Funktion.
    z.B. ACC := A + B oder R1 ← R2

- Register-Transfer-Operationen
  - Die Semantik der Notation R2 ← R1: der Inhalt von R1 wird nach R2 transportiert (kopiert), dabei wird R1 nicht verändert.
  - Dazu ist ein Datenpfad zwischen den Ausgängen von R1 und den Eingängen von R2 notwendig, und R2 muß parallel geladen werden können.
  - Der Transfer findet oft nicht mit jedem Takt statt, sondern unter gewissen Bedingungen:
    if (E=1) then R2 ← R1 end
    wo E ein Steuersignal



  - kompakte RTL-Notation:
    E: R2 ← R1
    R2 := (E=1) ? R1 : R2



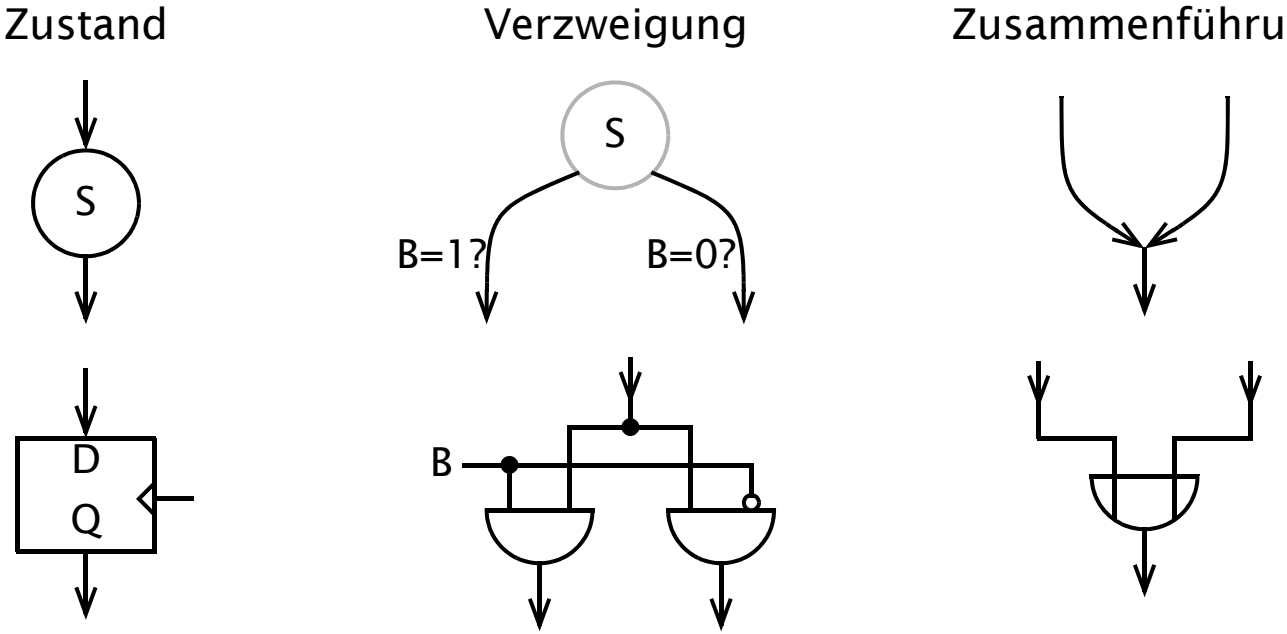
- Register-Transfer-Operationen
  - Alle Speicherelemente werden einheitlich mit dasselben Taktsignal gesteuert (=> Taktsignal erscheint nicht in der RTL-Notation),
  - Register sind entweder aus flankengesteuerten Flipflops oder Master-Slave-Flipflops aufgebaut.
    z.B. n-Bit-Register IR[n-1:0] oder XYZ[1:n]
  - Mehrere Register-Transfer-Operationen können parallel ausgeführt werden, dann werden diese mit Komma voneinander getrennt.

  z.B. der Datentransfer zwischen zwei Registern unter der Kontrolle der Steuervariablen P lautet

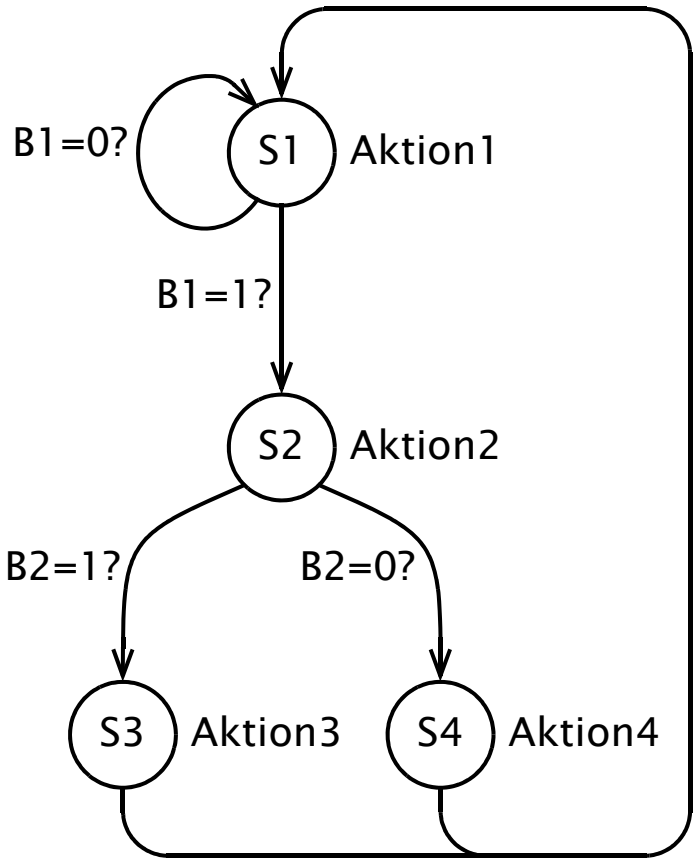
  P: R2 ← R5, PC ← PC+1, XYZ ← 0

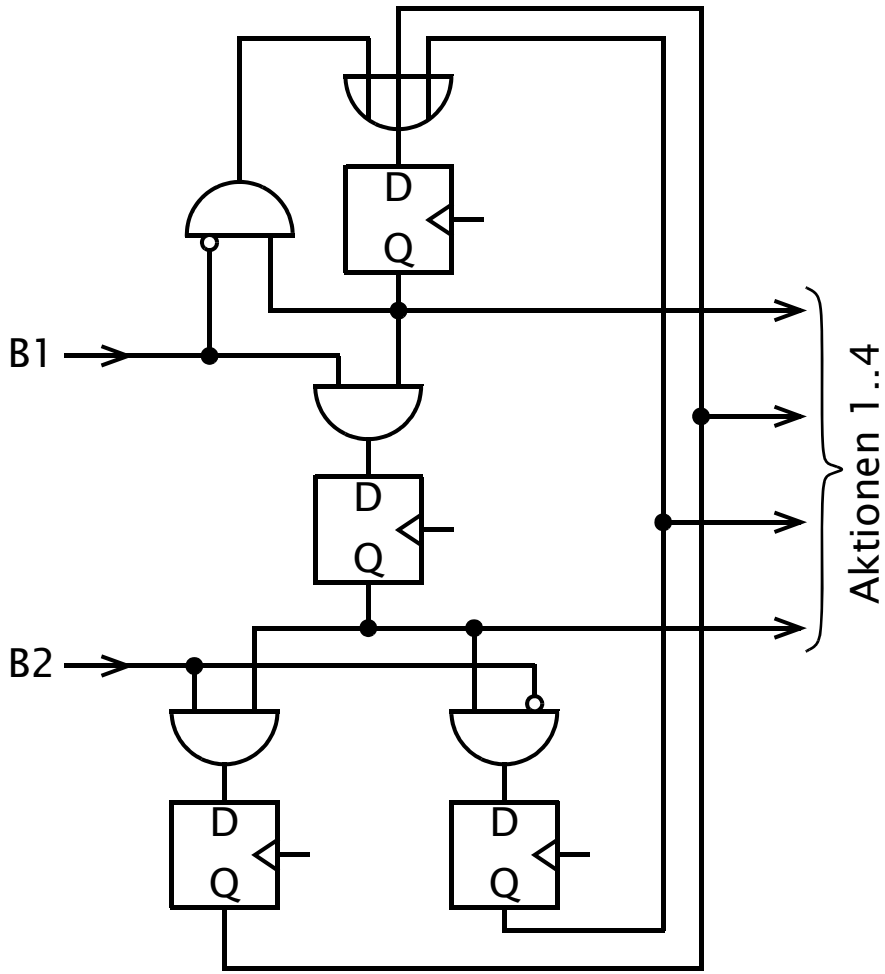
  Interpretation: wenn die Bedingung erfüllt ist (P=1), dann wird der Inhalt aus dem Register R5 ins Register R2 kopiert, und gleichzeitig (d.h. in demselben Takt), werden der Inhalt des Registers PC um 1 erhöht und der Inhalt des Registers XYZ auf Null gesetzt.

- Realisierung von Steuerwerken
  - festverdrahtete Steuerwerke
    - Schrittsteuerwerk (Einphasentakt)
    - Multiphasen-Generator
    - PLA und Zustandsregister
  - (mikro-)programmierbare Steuerwerke
- Schrittsteuerwerk mit One-Hot-Encoding der Zustände

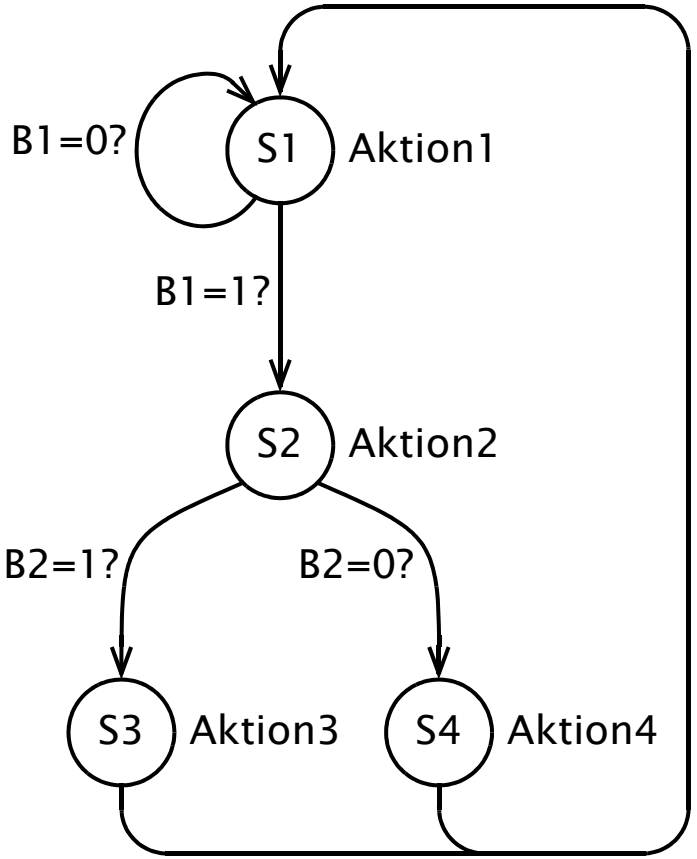


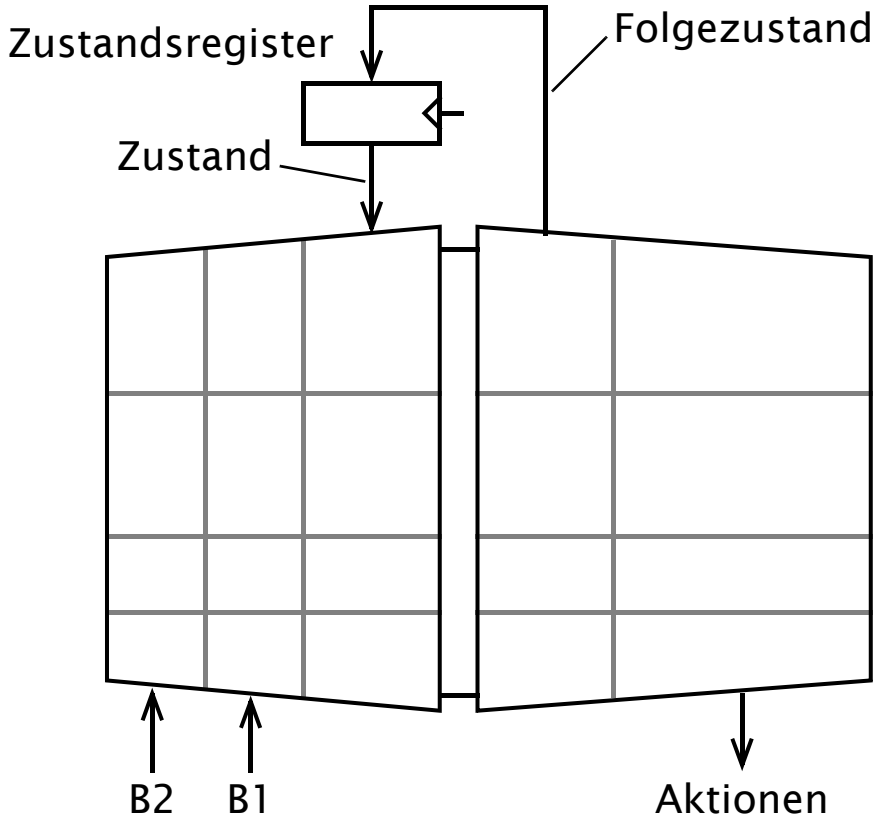
- Schrittsteuerwerk mit One-Hot-Encoding





- PLA-Steuerwerk mit binär codierten Zuständen
  - Codierung der Zustände: S1:=$(00)_2$, S2:=$(01)_2$, S3:=$(10)_2$, S4:=$(11)_2$

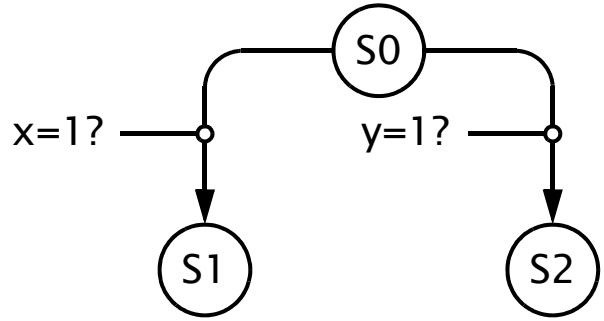


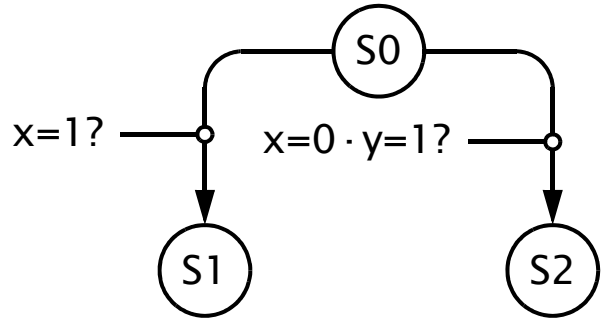


- Synthese synchroner Schaltwerke
  - Das Ziel bei der Synthese von Schaltwerken ist wie bei Schaltnetzen eine Realisierung mit einer möglichst geringen Schaltungskomplexität und möglichst kürzeren Laufzeiten (=> größte Taktfrequenz)
  - Zwei Größen haben erheblichen Einfluß auf die Komplexität von Übergangs- und Ausgangsschaltnetzen in Schaltwerken:
    - die Zahl der Zustandsvariablen
    - die Codierung der Zustände
  - Diese Zusammenhänge sind so komplex, daß heute noch kein Verfahren zur Minimierung des Realisierungsaufwandes existiert, welches alle Entwurfsfreiheiten benutzt.

- Äquivalente Automaten
  - Zwei Automaten EA_1 und EA_2 sind äquivalent, wenn sie aufgrund ihres äußeren Verhaltens nicht unterschieden werden können, d.h. wenn sie auf beliebige und beliebig lange Folgen von Eingangswerten stets identische Folgen von Ausgangswerten erzeugen.
- Äquivalente Zustände
  - Zwei Zustände Z_i und Z_j sind äquivalent, wenn der Automat auf eine beliebige Folge von Eingangswerten immer mit derselben Folge von Ausgangswerten reagiert, gleichgültig ob im Zustand Z_i oder im Zustand Z_j begonnen wird.
  - Äquivalente Zustände können vereinigt werden, was im allgemeinen zur Vereinfachung des Automaten und dessen technischen Realisierung als Steuerwerk führt.
  - Für eine Menge äquivalenter Zustände (sog. Äquivalenzklasse) wird dann stellvertretend nur ein Zustand (sog. Repräsentant) berücksichtigt.

- Deterministische Automaten
  - Zwei verschiedene Eingaben, die ausgehend von einem Zustand den Übergang zu zwei verschiedenen Zuständen auslösen, dürfen nicht gleichzeitig auftreten.
  - In diesem Fall wäre es sonst nicht entscheidbar, welcher Zustands-übergang erfolgen soll. Es entstünde ein nicht-deterministisches Ver-halten, das mit klassischen Verfah-ren nicht mehr untersucht werden kann.
  - Die meisten Analyse-/Synthesever-fahren gehen deshalb davon aus, daß zwei Ereignisse/Eingaben nicht zeitgleich auftreten können.





- Synthese synchroner Schaltwerke

  erfolgt meistens in sechs iterativ ablaufenden Phasen:

  1. Umsetzung einer textuellen (verbalen, nicht formalen) Beschreibung in eine graphische (formale) Spezifikation (wie Zustandsgraph, Impulsplan) mit der Festlegung der Ein-/Ausgangsvariablen
  2. Aufstellung einer sog. primitiven Zustandstabelle
  3. Zustandsminimierung und Zustandsverschmelzung
  4. Wahl der Zustandscodierung
  5. Minimierung der Übergangs- und Ausgangsschaltnetze
  6. Überprüfung des realisierten Schaltwerks

  - Bei einer Unzufriedenheit mit dem Resultat der Synthese kann der Ablauf wiederholt werden, und zwar in einer inneren Schleife mit den Schritten 4, 5 und 6, oder in einer äußeren Schleife, welche die Schritte 3 bis 6 umfaßt.

- Umsetzung einer verbalen, nicht formalen Beschreibung in eine formale Spezifikation

  - verbale Beschreibungen (Pflichtenheft) sind nicht formale Spezifikationen, die oft unvollständig, mehrdeutig, manchmal widersprüchlich sind,

  - die Umsetzung einer verbalen Beschreibung in eine formale Spezifikation (UML-Diagramme, Zustandsgraph, Impulsplan, Hardwarebeschreibungssprache, Programmablaufplan, usw.) ist der schwierigste Schritt in der Schaltwerkssynthese:
    z.B. eine verbale Beschreibung kann durch viele unterschiedliche (semantisch äquivalente) Zustandsgraphen beschrieben werden,

  - leider keine universell anwendbare Methode verfügbar, die von einer verbalen Beschreibung zu einer formalen Spezifikation führt.

- Umsetzung einer verbalen, nicht formalen Beschreibung in eine formale Spezifikation

  - die Vorgehensweise und das Ergebnis der Umsetzung sind im hohen Maße von der Erfahrung und der Geschicklichkeit des Entwicklers abhängig,

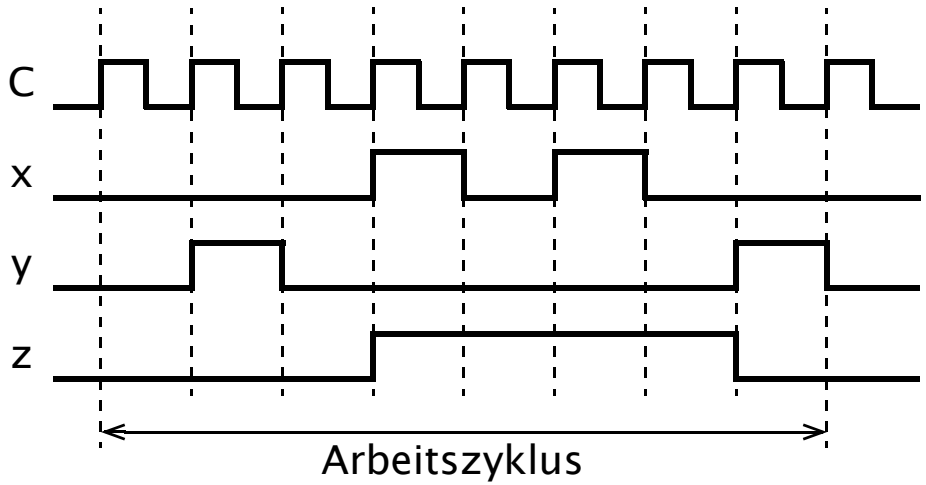
  - Unübersichtlichkeit der Zustandsgraphen mit steigender Anzahl an Zuständen => Partitionierung in mehrere modulare, miteinander kooperierende Schaltwerke erforderlich => Handshaking- und Synchronisationssignale zwischen Schaltwerken notwendig

  - ist eine Aufgabe mit formalen Methoden spezifiziert, so sind restliche Schritte in der Schaltwerkssynthese mit systematischen Verfahren durchführbar.

- Zustandstabelle
  - eine Tabelle mit Zeilen für sämtliche Zustände und mit Spalten für sämtliche Kombinationen der Eingangswerte und einer zusätzlichen Spalte für die Ausgangswerte.
  - Zustände werden an den Zeilen notiert und Folgezustände in die Felder der Tabelle eingetragen.
  - Sind für bestimmte Felder keine Folgezustände definiert, so bleiben diese leer.
  - Die Werte der Ausgangsvariablen sind Zuständen zugeordneten und werden in der zusätzlichen Spalte zeilenweise eingetragen.

	00	01	11	10	x	y	z

- Aufstellung einer primitiven Zustandstabelle aus einem Impulsplan
  1. Arbeitszyklus im Impulsplan identifizieren
  2. Arbeitszyklus in einzelne Taktphasen aufteilen, in den Taktphasen bleibt der Zustand des Schaltwerks stabil
  3. Taktphasen im Arbeitszyklus fortlaufend durchnumerieren
  4. Taktphasen in die primitive Zustandstabelle eintragen



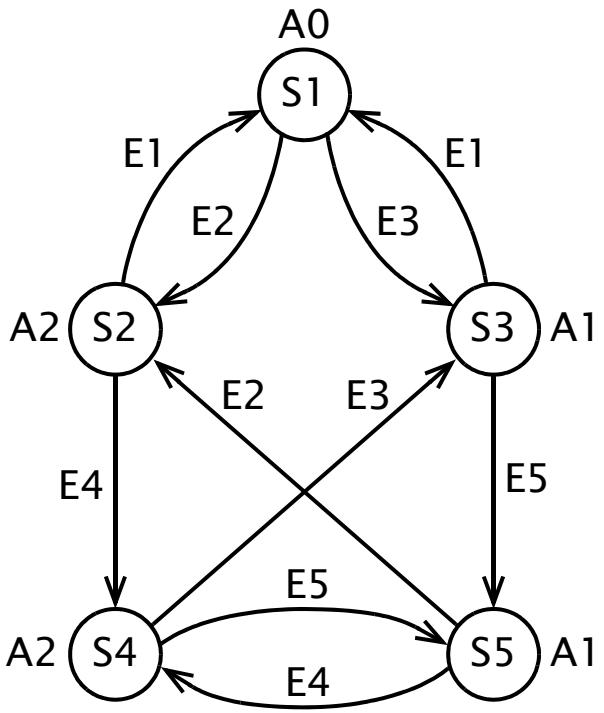
	xy 00	01	11	10	z
1					
2					
3					
4					
5					
6					
7					
8					

- Aufstellung einer primitiven Zustandstabelle aus einem Zustandsgraphen

  Zustände: S1, S2, S3, S4, S5

  Eingangskombinationen: E1, E2, E3, E4, E5

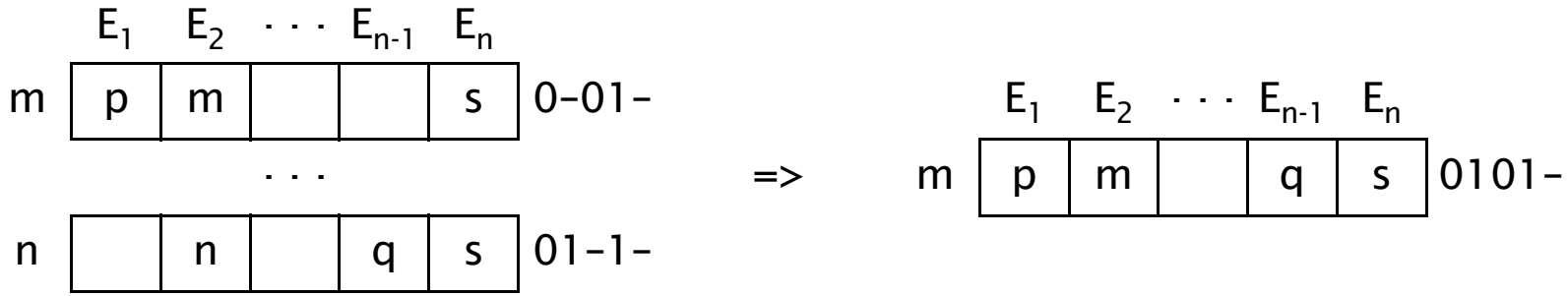
  Ausgangskombinationen: A0, A1,A2



	E1	E2	E3	E4	E5
S1		S2	S3		
S2	S1			S4	
S3	S1				S5
S4			S3		S5
S5		S2		S4	

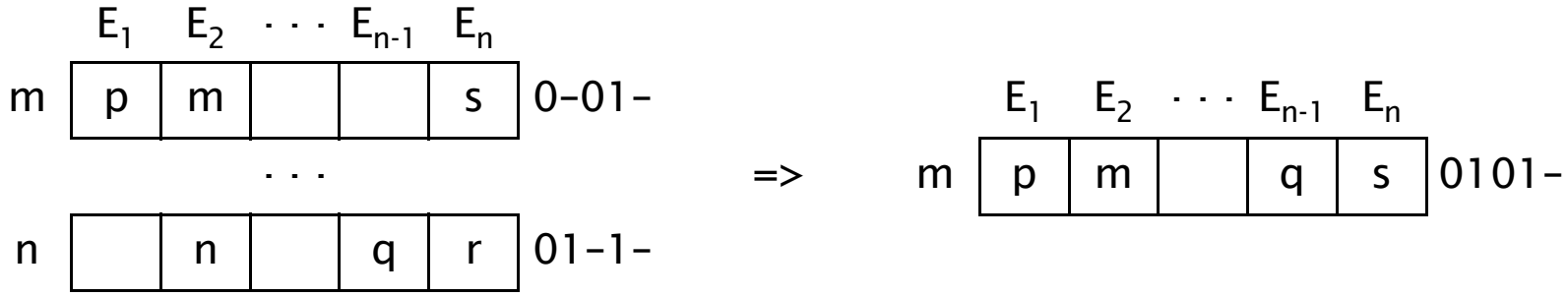
- Zustandsminimierung und Zustandsverschmelzung
  - Das Ziel der Zustandsminimierung und der anschließenden Zustandsverschmelzung ist es, die Anzahl der Zustände in einer primitiven Zustandstabelle zu reduzieren, ohne dadurch das Ein-/Ausgangsverhalten eines Systems zu ändern.
  - Die Reduktion der Anzahl der Zustände führt im allgemeinen zu Vereinfachung in Übergangs-/Ausgangsschaltnetzen.
  - Bei der Zustandsminimierung werden mit Hilfe einer Implikationsmatrix äquivalente und pseudo- äquivalente Zustände ermittelt und zur Reduzierung der Anzahl der Zustände benutzt.
  - Bei der Zustandsverschmelzung werden mit Hilfe eines Verschmelzungsgraphen sog. verschmelzbare Zustände mit dem gleichen Ausgangsverhalten ermittelt.

- Zustandsminimierung (1)
  - zwei Zustände m und n sind äquivalent, und dürfen zu einem Zustand zusammengefaßt werden, wenn



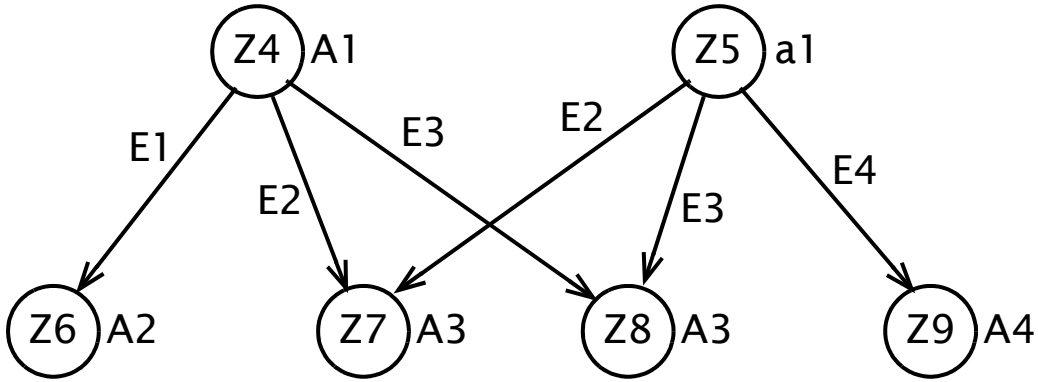
1. die Ausgangswerte der Zustände n und m gleich oder unbestimmt (don't care) sind, und
2. die den Zuständen n und m zugeordneten Folgezustände, also diejenigen, die durch dieselbe Eingangskombination E_i bestimmt sind (in derselben Spalte stehen), gleich oder unbestimmt sind.

- Zustandsminimierung (2)
  - zwei Zustände m und n sind *pseudo*-äquivalent, und können *bedingt* zu einem Zustand zusammengefaßt werden, wenn



1. die Ausgangswerte der Zustände n und m gleich oder unbestimmt sind, und
2. die den Zuständen n und m zugeordneten Folgezustände äquivalent oder pseudo-äquivalent sind.

- Zustandsminimierung



E1	E2	E3	E4
Z6	Z7	Z8	

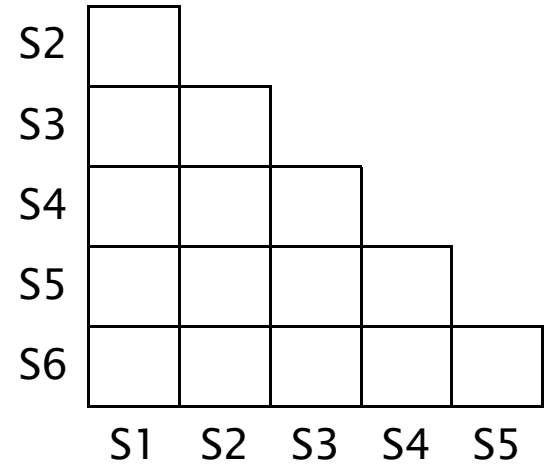
	Z7	Z8	Z9

=>

E1	E2	E3	E4
Z6	Z7	Z8	Z9

- Implikationsmatrix
  - dient zur Auffindung äquivalenter und pseudo-äquivalenter Zustände
  - ist in Form einer Halbmatrix aufgebaut
  - enthält alle Kombinationen der vorhandenen Zustände

  - Beispiel: Automat mit 6 Zuständen: S1, S2, S3, S4, S5, S6
  - in der Horizontalen (von links nach rechts) werden Zustände Z_1 bis Z_{n-1} aufgetragen
    hier: S1, S2, S3, S4 und S5
  - in der Vertikalen (von oben nach unten) werden Zustände Z_2 bis Z_n notiert
    hier: S2, S3, S4, S5 und S6



- Einträge in der Implikationsmatrix:

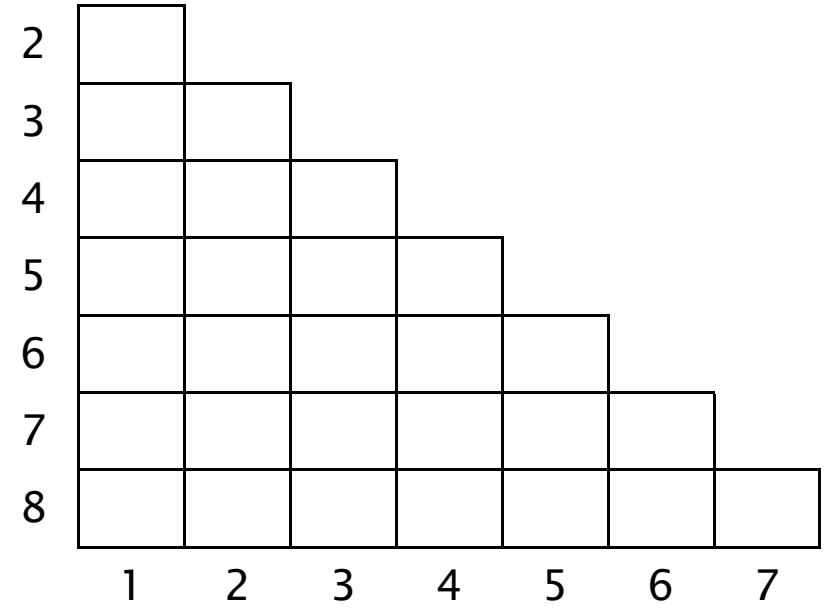
  das (i, j)–Feld in der i-ten Zeile und j-ten Spalte kann einen der folgenden Einträge enthalten:

  - = Zustände i und j sind äquivalent und können unbedingt zusammengefaßt werden
  - x Zustände i und j sind nicht äquivalent und können nicht zusammengefaßt werden
  - r,s Zustände i und j sind pseudo-äquivalent und können bedingt zusammengefaßt werden, und zwar in der Abhängigkeit davon, ob sich die Zustände r und s zusammenfassen lassen.

- Bei der Bestimmung der Einträge für die Implikationsmatrix werden sämtliche Zustände aus der primitiven Zustandstabelle paarweise miteinander verglichen.

- Aufstellung der Implikationsmatrix aus einer primitiven Zustandstabelle

	xy 00	01	11	10	z
1	1	2			0
2	3	2			0
3	3			4	0
4	5			4	1
5	5			6	1
6	7			6	1
7	7	8			1
8	1	8			0



- Analyse der pseudo-äquivalenten Zustände aus der Implikationsmatrix

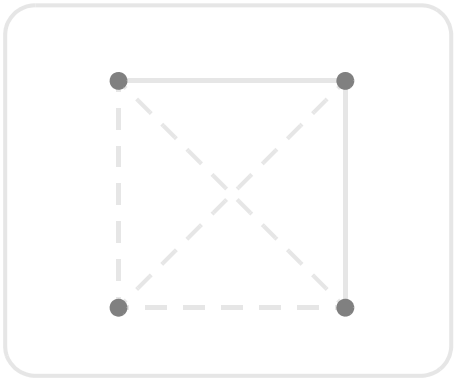
  für jedes Paar der pseudo-äquivalenten Zustände wird geprüft, ob es sich zusammenfassen läßt. Während der Analyse entsteht eine Implikationskette, die in einem Feld mit Zuständen terminiert, die:

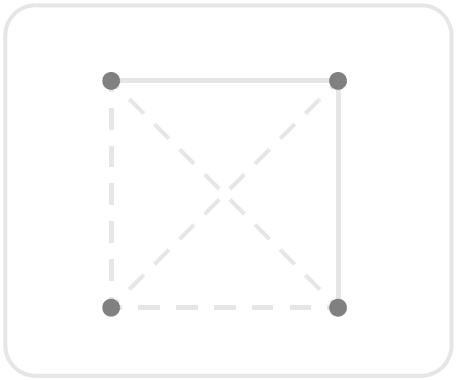
  1. nicht äquivalent sind (x): in diesem Fall werden sämtliche pseudo-äquivalenten Zustände in der entstanden Implikationskette als nicht äquivalent markiert.
  2. äquivalent sind (=): in diesem Fall werden sämtliche pseudo-äquivalenten Zustände in der entstanden Implikationskette als äquivalent markiert.
  3. pseudo-äquivalent sind (m,n), und bereits durch die Implikationskette analysiert worden sind. Somit entsteht eine endlose Rückkopplung in der Implikationskette: in diesem Fall können sämtliche pseudo-äquivalenten Zustände in der entstanden Implikationskette als äquivalent markiert werden.

- Beispielsweise gilt für die Zusammenfassung der Zustände 1 und 8 folgende Implikationskette:
  - Die Zusammenfassung der Zustände 1 und 8 ist davon anhängig, ob die Zustände 2 und 8 zusammenfaßbar sind.
  - Die Zustände 2 und 8 können aber nur in der Abhängigkeit von der Zusammenfassung der Zustände 1 und 3 zusammengefaßt werden.
  - Die Zustände 1 und 3 sind äquivalent, somit können auch die Zustände 2 und 8 zusammengefaßt werden.
  - Und schließlich kann auch der Zustand 1 mit dem Zustand 8 zusammengefaßt werden.
  - Aufgrund der entstandenen Implikationskette bei der Analyse darf man auf keinen Fall nur die Zustände 1 und 8 zusammenfassen, ohne die Zusammenfassung der Zustände 2 und 8.

- Durch die Aufstellung eines Verschmelzungsgraphen wird die Analyse der pseudo-äquivalenten Zustände aus der Implikationsmatrix anschaulicher.
- Verschmelzungsgraph
  - ein ungerichteter Graph mit Knoten und Kanten
  - Knoten repräsentieren Zustände aus der Zustandstabelle
  - Kanten stellen die Möglichkeit der Zusammenfassung von äquivalenten und pseudo-äquivalenten Zuständen dar.
  - Eine Kante, die äquivalente Zustände miteinander verbindet, wird mit einer durchgezogen Linie markiert.
  - Eine Kante, die pseudo-äquivalente Zustände verbindet, wird mit einer gestrichelten Linie markiert.

- Regelwerk für die Zustandsverschmelzung
  - zwei oder mehrere Zustände (Konten) dürfen zu einem Zustand verschmolzen werden, wenn sie alle unmittelbar mit Kanten untereinander verbunden sind (Gruppenbildung).
  - ein und derselbe Zustand darf nicht mehrfach an der Verschmelzung beteiligt sein.





- Aufstellung einer reduzierten Zustandstabelle
  - Einträge in der primitiven Zustandstabelle umbenennen:
    $(1, 3, 2, 3) \rightarrow 1$, $(4, 5, 6, 7) \rightarrow 5$
  - mehrfach vorkommende Zeilen entfernen
  - Zeilen mit der gleichen Zustandsbezeichnung zusammenfassen

	xy 00	01	11	10	z
1	1	2			0
2	3	2			0
3	3			4	0
4	5			4	1
5	5			6	1
6	7			6	1
7	7	8			1
8	1	8			0

	xy 00	01	11	10	z

xy 00	01	11	10	z

- Zustandscodierung

  die Codierung der Zustände bei synchronen Schaltwerken kann theoretisch beliebig erfolgen, praktisch aber wird sie nach Gesichtspunkten der reinen Zweckmäßigkeit vorgenommen:

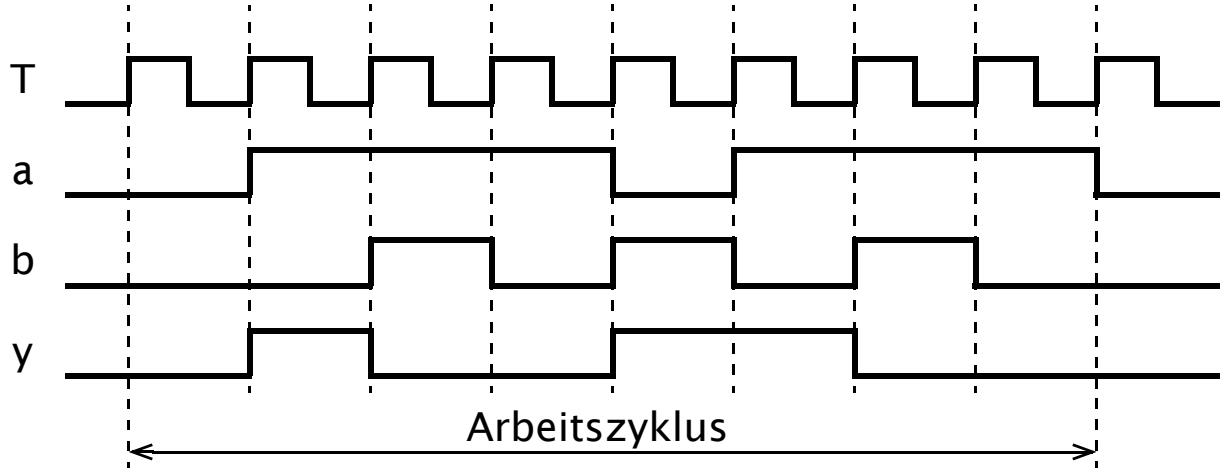
  1. Zustandscodierung maximaler Länge („one hot encoding“):
     jede Zustandsvariable bekommt ein separates Flipflop zugeordnet,
     heute bevorzugt in schnellen Schaltwerken eingesetzt
  2. Zustandscodierung mit minimalem Hardware-Aufwand:
     bspw. kann die Ausgangsfunktion (und somit auch das Ausgangsschaltnetz) bei Moore-Schaltwerken vollständig entfallen, wenn so eine Zustandscodierung gewählt wird, daß alle Werte der Ausgangsvariablen direkt mit der Zustandscodierung übereinstimmen.
  3. Zustandscodierung mit Sicherung gegen externe Fehlerquellen:
     fehlerkorrigierende Codierung wie bei Zählern
     einschrittige Codierung „benachbarter“ Zustände

- Zustandscodierung und Realisierung als Schaltwerk mit festverdrahteter Logik
  - Zustandscodierung $(1) \rightarrow (0)_2$, $(5) \rightarrow (1)_2$
  - Einträge in der reduzierten Zustandstabelle umbenennen
  - Minimierung der Übergangs- und Ausgangsfunktionen mit KV-Diagrammen

	xy 00	01	11	10	z
1	1	1		5	0
5	5	1		5	1

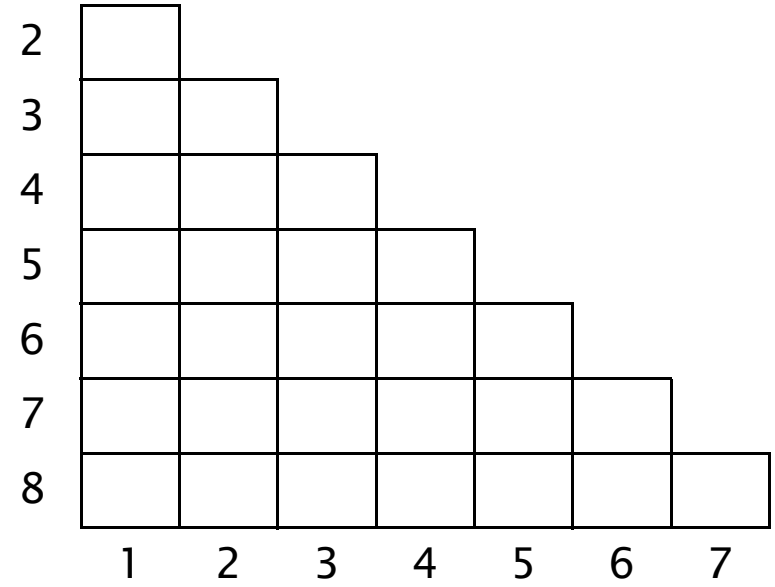
Q	xy 00	01	11	10	z

- Aufgabenstellung
  - Es ist ein synchrones Schaltwerk mit zwei Eingängen a und b und einem Ausgang y mit möglichst minimaler Anzahl von Zuständen zu entwerfen. Das Schaltwerk arbeitet nach dem unten dargestellten Impulsplan.

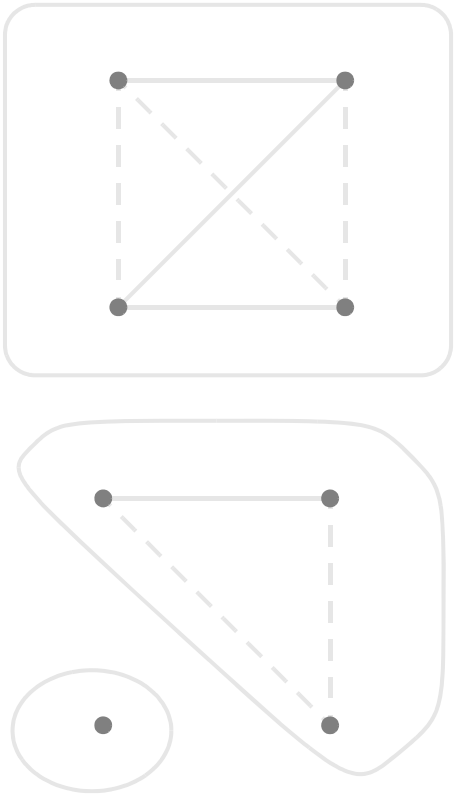


- primitive Zustandstabelle und Implikationsmatrix

ab	00	01	11	10	y



- Analyse der Einträge aus der Implikationsmatrix und Aufbau des Verschmelzungsgraphen
  - äquivalente Zustände:
  - pseudo-äquivalente Zustände:



- reduzierte Zustandstabelle, Codierung der Zustände und Realisierung mit einem PLA-Steuerwerk

alter Zustand	neuer Zustand	bin. Code
1	A	00
2, 5, 6	B	01
3, 4, 7, 8	C	10

ab

	00	01	11	10	y

ab

	00	01	11	10	y