Digitaltechnik

Andreas König
Professur Technische Informatik
Fakultät Informatik
Technische Universität Chemnitz

Wintersemester 2001/2002

Rekapitulierung zu Kapitel 6

- **Prinzipielle Betrachtung zu mehrstufiger Logik**
- **Vorstellung von Verzögerungsmodellen in Gattern und Verzögerungsbestimmung in Schaltnetzen**
- **Betrachtung zeitlichen Fehlverhaltens: Hazards/Glitches**
  - **Statischer 1/0-Hazard**
  - **Dynamischer 1/0-Hazard**
- **Überlegungen zum sicheren Entwurf mit hazardbehafteten Netzwerken**
- **Überlegungen und Herangehensweise zum Entwurf hazardfreier zweistufiger Gatternetze**

Vorlesungsgliederung:

1. **Einführung**
2. **Kodierung und Arithmetik**
3. **Grundlagen der Booleschen Algebra**
4. **Entwurf zweistufiger kombinatorischer Logik**
5. **Zieltechnologien und Technologieanpassung**
6. **Zeitliches Verhalten kombinatorischer Schaltnetze**
7. **Entwurf sequentieller Schaltwerke**
8. **Funktionsblöcke digitaler Rechner und Systeme**
9. **Entwurf von Systemen der Digitaltechnik**
10. **Ausblick**

Kapitelgliederung:

7. Entwurf sequentieller Schaltwerke

- **7.1 Motivation**
- **7.2 Automaten**
- **7.3 Digitale Speicherelemente**
- **7.4 Zähler und Frequenzteiler**
- **7.5 Entwurf von Automaten**
- **7.6 Zustandsminimierung**
- **7.7 Zustandskodierung**
- **7.8 FSM-Partitionierung**
- **7.9 FSM-Zusammenschaltung**

Motivation

- Bislang wurde vom Entwurf rein kombinatorischer Logik ausgegangen
- Die Ausgänge eines Gatternetzwerks sind nur von den Eingängen zum Zeitpunkt einer anliegenden Belegung abhängig
- Zeitliche Effekte werden nur im Hinblick auf die Signalpropagation vom Eingang zum Ausgang berücksichtigt (Verzögerung, Glitches)
- Vorhergehende Eingangsbelegungen haben keine Wirkung auf Schaltwerks-ausgang ! (Keine Speicherung; kein Gedächtnis)

- **Vorteil:** Einfache Struktur, leicht zu verstehen und zu entwerfen
- **Nachteil:** Alle Eingangsinformation muss gleichzeitig anliegen !

Motivation

- Die Verfügbarkeit eines Gedächtnisses oder Kontexts kann ein Schaltwerk durch Vorgangssequenzierung entscheidend vereinfachen
- **Beispiel:** Eingabe eines vierstelligen Pincodes
- Gleichzeitige Eingabe mit vier Fingern auf einer/vier Tastaturen ?
- Wie kann ein Gedächtnis realisiert werden ?
- Puffern einer (endlichen) Reihe von k Eingangsbelegungen, so dass Y=F(X) ersetzt wird durch Y=F(X(t), X(t-1), ..., X(t-k))

- Mit jedem Zeitschritt muss ein Element verworfen werden
- Diese Art der Realisierung kann sehr aufwendig und redundant werden

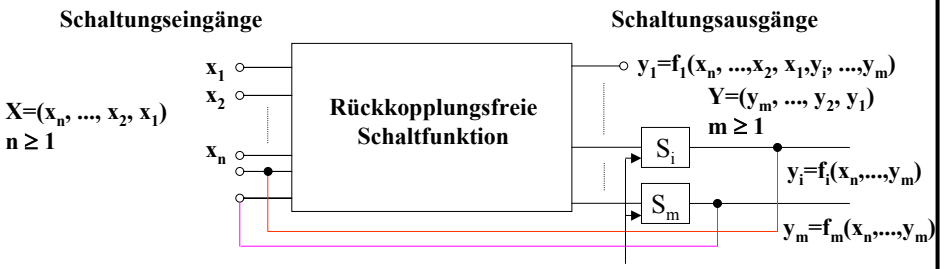
Motivation

- Andere Form der Realisierung eines Gedächtnisses: Rückführung eines oder mehrerer Ausgänge auf Eingänge des Schaltwerks

- Erkennbar hängen Ausgangsänderungen von Eingangsänderungen, dem Kontext und den Laufzeiten in den Pfaden des Schaltwerks ab !
- Sogenanntes asynchron sequentielles Schaltwerk
- **Vorteil:** Schnell (Arbeitsgeschwindigkeit) und aufwandsgünstig zu realisieren
- **Nachteil:** Kritisch zu entwerfen, oft unberechenbares Verhalten (Verzögerungszeiten, Läufe)

Motivation

- Sicherer und berechenbarerer Entwurf einer rückgekoppelten Struktur durch Verwendung von Speicherelementen im Rückkopplungspfad als Gedächtnis
- Steuerung der Speicherelemente durch externen Systemtakt (vgl. Metronom)
- Synchrone Wechsel: Synchron sequentielles Schaltwerk



- Synchronisierung verhindert Läufe und Oszillationen
- Der zielgerichtete, optimierte Entwurf synchron sequentieller Schaltwerke erfordert eine Reihe von Begriffsdefinitionen und Festlegungen

Automaten

- Ein synchron sequentielles Schaltwerk implementiert einen endlichen deterministischen Automaten (Finite-State-Machine, FSM)
- Ein endlicher Automat AT kann durch ein Quintupel

$$AT = (I, O, S, \delta, \lambda)$$

beschrieben werden (ggf. Erweiterung um Startzustand S_1)

- Dabei sind I, O, und S drei endliche Mengen:
  - Eingangsalphabet $I=\{I_1, I_2, ..., I_n\}$
  - Ausgangsalphabet $O=\{O_1, O_2, ..., O_m\}$
  - Zustandsmenge $S=\{S_1, S_2, ..., S_z\}$
- δ und λ sind zwei Abbildungen, die durch rückkopplungsfreie kombinatorische Netze realisiert werden:
  - δ ist die Überführungs- bzw. Zustandsübergangs-funktion von AT mit:

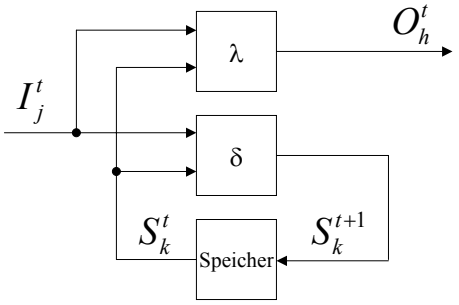
$$S_k^{t+1} = \delta\left(I_j^t, S_k^t\right)$$

  - λ ist die Ausgabefunktion von AT mit:

$$O_h^t = \lambda\left(I_j^t, S_k^t\right)$$

Automaten

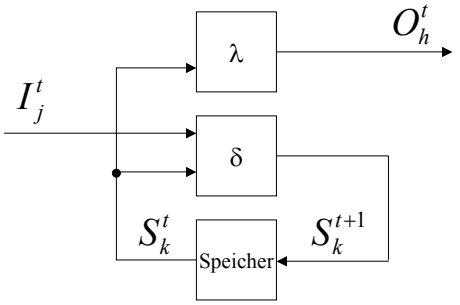
- In der Literatur exisitieren eine Reihe von Typklassen für Automaten, von denen im Kontext der Digitaltechnik die Merkmale endlich, deterministisch und diskret relevant sind
- Für diesen Fall und im Hinblick auf die Ausgabefunktion in Abhängigkeit von I^t und S^t gibt zwei relevante Automatentypen
- Der Mealy-Automat [Katz 94] [Lipp 99] bzw. [Mealy 55]:



$$O_h^t = \lambda\left(I_j^t, S_k^t\right)$$

$$S_k^{t+1} = \delta\left(I_j^t, S_k^t\right)$$

- Der Moore-Automat [Katz 94] [Lipp 99] bzw. [Moore 56]:

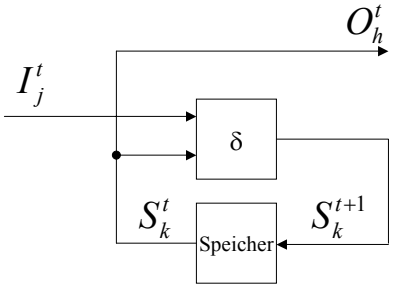


$$O_h^t = \lambda\left(S_k^t\right)$$

$$S_k^{t+1} = \delta\left(I_j^t, S_k^t\right)$$

- Erkennbar hängt bei diesem Automatentyp der Ausgang des Automaten nur vom aktuellen Zustand ab
- Änderungen der Eingangsbelegung werden erst im folgenden Taktzyklus nach Übergang in einen neuen Zustand wirksam

- Der Medwedew-Automat [Lipp 99]:



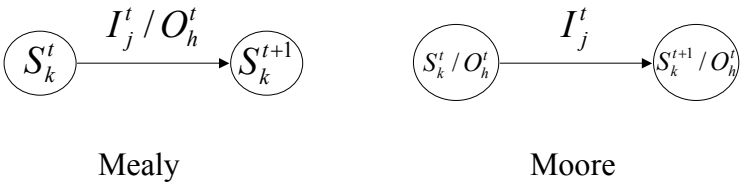
$$O_h^t = \lambda\left(S_k^t\right)$$

$$S_k^{t+1} = \delta\left(I_j^t, S_k^t\right)$$

- Sonderfall des Moore-Automaten, bei dem die Ausgabefunktion eine feste Durchschaltung ist, d.h. Ausgabe ist der Zustand selbst
- Änderungen der Eingangsbelegung werden auch hier erst im folgenden Taktzyklus nach Übergang in einen neuen Zustand wirksam

Automaten

- Beschreibungsformen für Automaten:
  - Automatendiagramm
  - Automatentafel



Mealy Moore

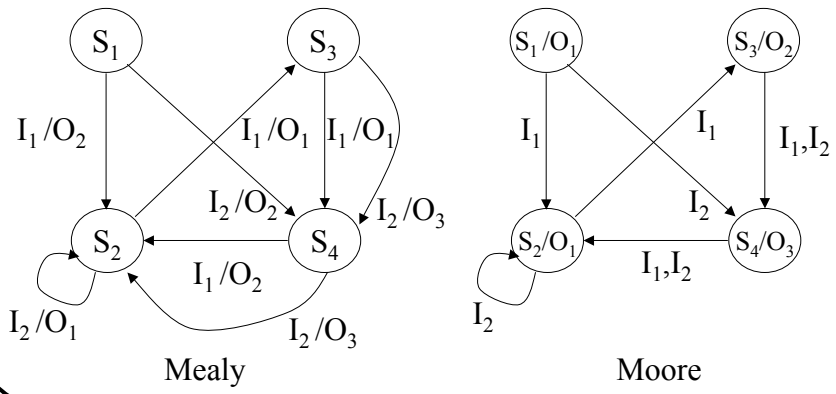
- Graph, dessen Knoten den Zuständen des Automaten entsprechen
- Zustandsübergänge entsprechen den verbindenden, gerichteten Kanten
- Beim Mealy-Automaten wird jede Kante als Attribut mit dem zugehörigen Eingangs- und Ausgangswert zu dem Zustandsübergang versehen
- Beim Moore-Automaten geht der Ausgangswert als Knotenattribut ein

Automaten

- Beispiel eines einfachen Mealy- bzw. Moore- Automaten und der zugehörigen Automatengraphen

$$I = \{I_1, I_2\}, \quad O = \{O_1, O_2, O_3\} \quad S = \{S_1, S_2, S_3, S_4\}$$

δ und λ gehen aus den Automatengraphen hervor



Mealy Moore

- Beschreibungsformen für Automaten:
  - Automatendiagramm
  - Automatentafel

	S^{t+1}/O^t		
S^t	I_1^t	....	I_n^t
...		...	
S_k^t	...	S_k^{t+1}/O_h^t	...
...		...	

Mealy

	S^{t+1}			
S^t	I_1^t	....	I_n^t	O^t
...		...		
S_k^t	...	S_k^{t+1}	...	O_h^t
...		...		

Moore

- Kartesisches Produkt aus Eingabe- und Zustandsmenge
- Beim Mealy-Automaten wird Folgezustand und Ausgangswert
- beim Moore-Automaten nur der Folgezustand eingetragen

- Fortsetzung des Beispiel eines einfachen Mealy- bzw. Moore- Automaten mit den zugehörigen Automatentafeln

	S^{t+1}/O^t	
S^t	I_1^t	I_2^t
S_1	S_2/O_2	S_4/O_2
S_2	S_3/O_1	S_2/O_1
S_3	S_4/O_1	S_4/O_3
S_4	S_2/O_2	S_2/O_3

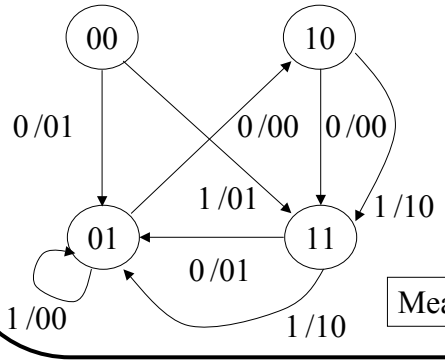
Mealy

	S^{t+1}		
S^t	I_1^t	I_2^t	O^t
S_1	S_2	S_4	O_1
S_2	S_3	S_2	O_1
S_3	S_4	S_4	O_2
S_4	S_2	S_2	O_3

Moore

- Die Umsetzung eines Automaten in ein konkretes Schaltwerk erfordert die Festlegung von Zuordnungen von Automatenelementen zu binären Größen
- Ein- und Ausgabezuordnung: wie Schaltnetze
- Umsetzung der Zustandsmenge: Zustandskodierung

$$I = \{I_1, I_2\}, \quad O = \{O_1, O_2, O_3\} \quad S = \{S_1, S_2, S_3, S_4\}$$
$$0 \quad 1 \qquad 00 \quad 01 \quad 10 \qquad 00 \quad 01 \quad 10 \quad 11$$



Mealy

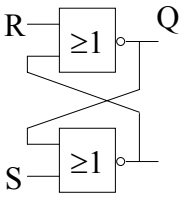
	S^{t+1}/O^t	
S^t	I_1^t	I_2^t
00	01 /01	11 /01
01	10 /00	01 /00
10	11/00	11/10
11	01 /01	01 /10

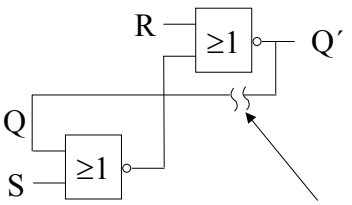
- Nach Umsetzung der Automatenelemente durch ihre Binärrepräsentation wird vom Schaltwerksgraph und Schaltwerkstafel gesprochen [Lipp 99]
- Üblich sind auch die Bezeichnungen Zustandsdiagramm (State-Diagram) und Zustands(übergangs)tabelle (State-Transition-Table, [Katz 94]):

S^t	I^t	S^{t+1}	O^t
S_1	0	S_2	O_1
S_2	0	S_3	O_1
S_3	0	S_4	O_2
S_4	0	S_2	O_3
S_1	1	S_4	O_1
S_2	1	S_2	O_1
S_3	1	S_4	O_2
S_4	1	S_2	O_3

- Symbolische/kodierte Zustandstabelle
- Zusätzlich: Zustandsgraph/-tafel
- Zustandsübergangs- oder Transitions-funktion aus Tabelle verfügbar
- Einzelne Transitionsgleichungen wie Funktionsbündel zu entwerfen
- Kein allg. algorithmischer Weg zur Transformation Spec/Zustandsdiagramm
- Vor Betrachtung des Entwurfs Behandlung der Speicherelemente

- Die für synchron sequentielle Schaltwerke erforderlichen Speicherelemente werden durch Latches oder Flipflops realisiert
- Mehrheitlich werden diese durch Taktsignale (Zustand/Flanke) angesteuert
- Taktsignale (Ein- oder Mehrphasentakt) stellen die synchrone Zustandänderung mehrerer Speicherelemente sicher
- **Zunächst:** Betrachtung eines RS-Flipflops (Latches):

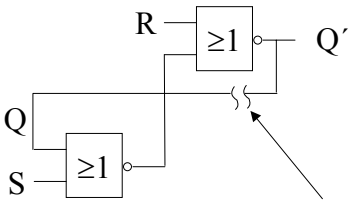




$$Q' = \overline{R}Q \vee \overline{R}S$$

Auftrennung der Rückkopplung zur Analyse des asynchronen Netzwerks

- Darstellung der Verknüpfungsfunktion des aufgetrennten Netzwerks:



Auftrennung der Rückkopplung

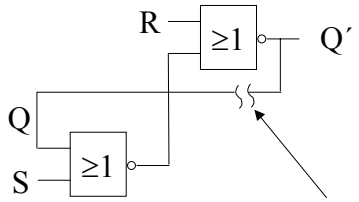
Q \ R,S	00	01	11	10
0	0	0	0	1
1	1	0	0	1

Q´

$$Q' = \overline{R}Q \vee \overline{R}S$$

Digitale Speicherelemente

- Bei rekurrenten Systemen unterscheidet man zwischen stabilen und instabilen Zuständen
- Merkmal eines stabilen Zustands: Q=Q´



Auftrennung der Rückkopplung

Q \ R	0	0	0	1
1	1	0	0	1
S				

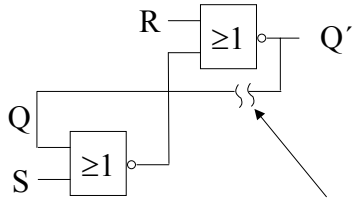
Q´

Beispiel eines stabilen Zustands

$$Q' = \overline{R}Q \vee \overline{R}S$$

Digitale Speicherelemente

- In einem instabilen Zustands gilt: Q≠Q´



Auftrennung der Rückkopplung

Q \ R	0	0	0	1
1	1	0	0	1
S				

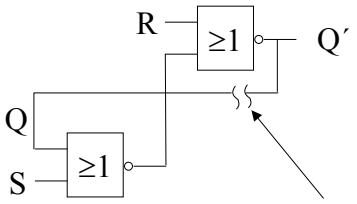
Q´

Beispiel eines instabilen Zustands

$$Q' = \overline{R}Q \vee \overline{R}S$$

- Für alle (im KV-Diagramm gegebenen) stabilen Zustände gilt alter Zustand = neuer Zustand, d.h. die Funktion:

$$F = Q \equiv \overline{R}Q \vee \overline{R}S$$



Auftrennung der Rückkopplung

Q \ R,S	00	01	11	10
0	0	0	0	1
1	1	0	0	1

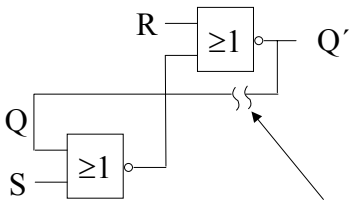
Q´

Q \ R,S	00	01	11	10
0	1	1	1	0
1	1	0	0	1

F

$$F = Q \equiv \overline{R}Q \vee \overline{R}S$$
$$= Q(\overline{R}Q \vee \overline{R}S) \vee \overline{Q}((R \vee \overline{Q})(R \vee \overline{S}))$$
$$= \overline{R}Q \vee R\overline{Q} \vee \overline{Q}\overline{S}$$

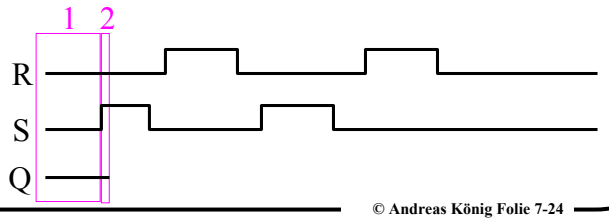
- Wirkung der zeitlichen Änderung des S-Eingangs von 0 nach 1 für Q´=0:
- Q´ wechselt auf 1, während Q für einen Verzögerungszeitraum auf 0 bleibt



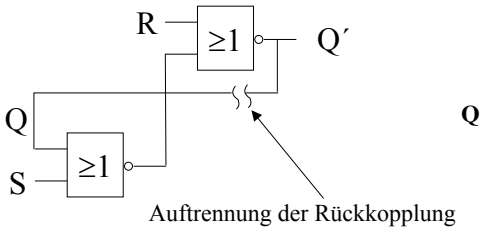
Auftrennung der Rückkopplung

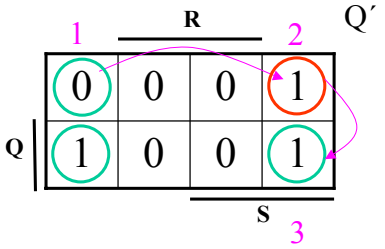
Q \ R,S	00	01	11	10
0	0	0	0	1
1	1	0	0	1

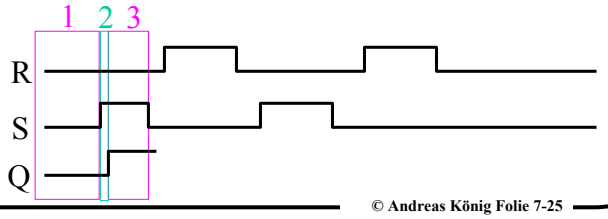
Q´



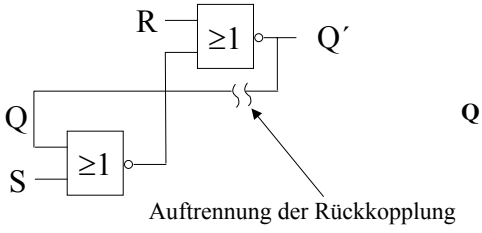
- Der erreichte Zustand ist instabil
- Es erfolgt eine weiterer Übergang in einen stabilen Zustand (selbe Spalte)

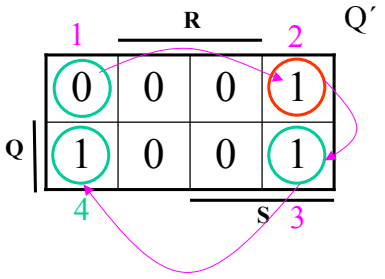


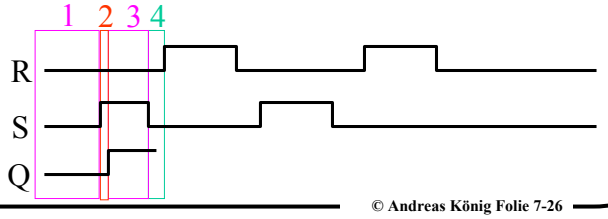




- Der erreichte Zustand ist instabil
- Es erfolgt eine weiterer Übergang in einen stabilen Zustand

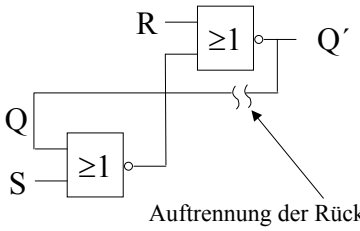






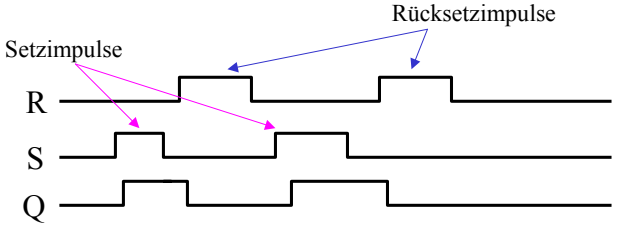
Digitale Speicherelemente

- Eine Zustandsänderung wurde für die Änderung von nur einer Eingangsvariablen betrachtet
- Erkennbar wirken Impulse auf S und R als Setz- und Löschimpulse



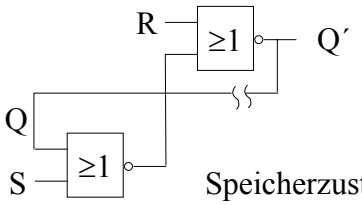
Q \ R	0	0	1	1
0	0	0	0	1
1	1	0	0	1

S



Digitale Speicherelemente

- Für die Situation R=0 und S=0 besitzt die Anordnung zwei stabile Zustände
- Damit wird der vorliegende Wert (dauerhaft) gespeichert

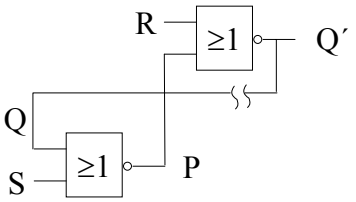


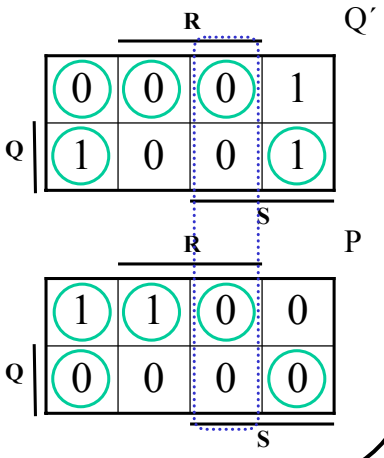
Q \ R	0	0	1	1
0	0	0	0	1
1	1	0	0	1

S

- Erkennbar zeigt die Anordnung sogenanntes bistabiles Verhalten
- Mit S=1 (R=0) wird das RS-FF (Latch) gesetzt, mit R=1 (S=0) gelöscht
- Für R=0 und S=0 wird der vorhergehende Zustand gespeichert
- Die Zustandsänderung erfolgt asynchron in Abhängigkeit von R und S

- Die vierte mögliche Eingangsbelegung R=1 und S=1 ist problematisch
- Für diesen Fall gilt nicht $P=\overline{Q'}$

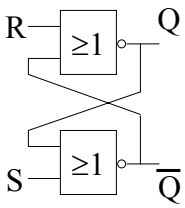




- Verbotener Zustand
- Auf Eingangsänderung erfolgen Zustandswechsel, bis stabiler Zustand erreicht wird
- Sollte durch Ansteuerung oder Erweiterung des FF verhindert werden

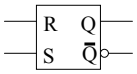
- Erkennbar ist die Analyse (und damit der Entwurf) asynchroner Schaltungen nicht unproblematisch
- Vereinfachende Herangehensweise:
  - Höchstens ein Eingang darf sich zu einem Zeitpunkt ändern
  - Ein weiterer Eingang darf sich erst dann ändern, wenn alle aus der vorhergehenden Änderung bedingten Zustandswechsel beendet sind
- Nur beschränkte Gültigkeit/Anwendbarkeit dieser Modellbetrachtung
- Bei mehreren vorliegenden Rückkopplungen:
  - Annahme gleichzeitiger Wertänderung aller Speicherelemente (paralleles Einheitsverzögerungsmodell)
  - Berücksichtigung unterschiedlicher Verzögerungszeiten auf den vorliegenden Pfaden durch nicht gleichzeitige Wertänderung der Speicherelemente (nebenläufiges Modell)
- Laufzeitbedingtes (instanzabhängiges !) unterschiedliches Endverhalten (Läufe, Races)
- Daher wird in der Praxis dominant synchroner Entwurfstil verfolgt !

- RS-NOR Flipflops (Latch) mit Wahrheitstabelle:

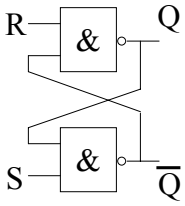


S	R	Q´
0	0	Speichern
0	1	0
1	0	1
1	1	Verboten

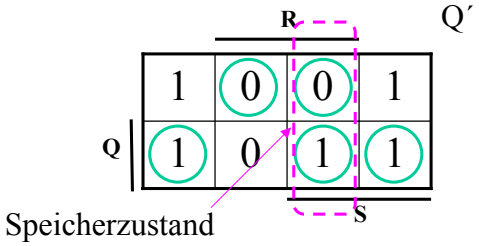
$$Q' = \overline{R}Q \vee \overline{R}S$$



- RS-NAND Flipflop:

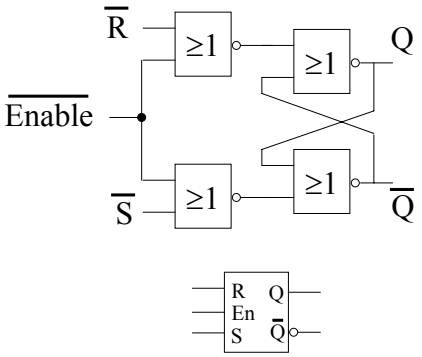


$$Q' = \overline{R} \vee QS$$



S	R	Q´
0	0	Verboten
0	1	0
1	0	1
1	1	Speichern

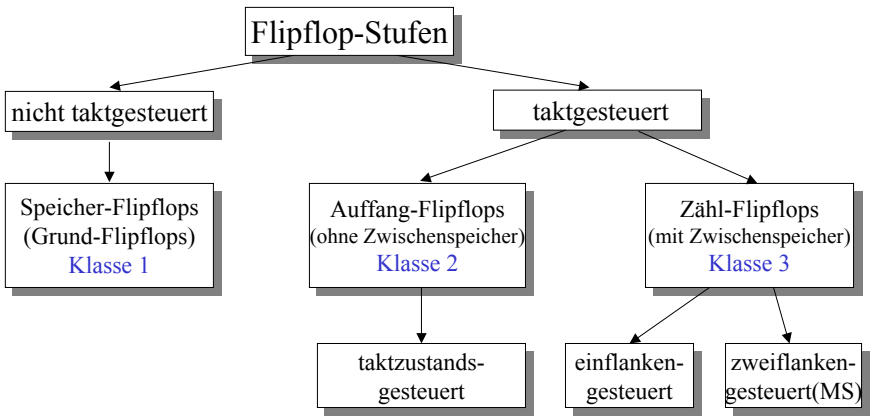
- Erweiterung des RS-NOR Flipflops um Zustandssteuerung:



$\overline{Enable}$	$\overline{S}$	$\overline{R}$	Q´
0	0	0	Verboten
0	0	1	1
0	1	0	0
0	1	1	Speichern
1	0	0	Speichern
1	0	1	Speichern
1	1	0	Speichern
1	1	1	Speichern

- Änderungen werden nur für die Dauer der Aktivierung eines Enable oder Taktsignals durchgestellt (Taktpegel- oder Taktzustandssteuerung)

- Vor der weiteren Betrachtung verschiedener Flipflop (Latch)-Typen sollen im folgenden einige allgemeine Punkte zusammengestellt werden
- Prinzipielle Einteilung in Abhängigkeit der Taktsteuerung [Seifart 98]:

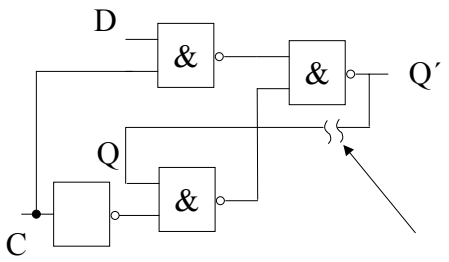


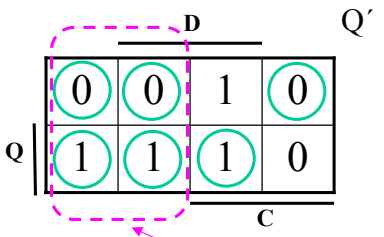
Digitale Speicherelemente

- **FF-Taktsteuerung:**
  - Keine (Grund-FF) Zustandswechsel asynchron durch R- und S-Eingänge
  - statisch (Zustands- oder pegelgesteuert, Enable) Übernahme werden Pulsdauer
  - dynamisch (Flankengesteuert, Vorder/Rückflanke, Clock) Übernahme nur während einer Taktflanke. Zweiflankensteuerung typisch bei zweistufigen sogenannten Master-Slave MS-FF
- **FF-Eingänge:**
  - **Takteingänge** zur Steuerung der Informationsübernahme (s.o)
  - **Vorbereitungseingänge** an denen unter Wahrung von Zeitbedingungen die zu speichernden Binärinformationen bereitgestellt werden (R, S, J, K, D)
  - **Asynchrone Eingänge:** Zusätzlich zu den taktgesteuerten Eingängen besitzen FFs typisch asynchrone Eingänge, die zur Initialisierung (PRESET, CLEAR) der FFs dienen (z.B. Power-Up, System Reset)
- Unter zugelassenen Betriebsbedingungen (s. verbotener Zustand bei RS-FF) liegen an den beiden FF-Ausgängen jeweils negierte Binärwerte vor
- Die sogenannte charakteristische Gleichung beschreibt das Verhalten des FFs in schaltalgebraischer Form; Beispiel RS-NOR-FF: $Q' = \overline{R}Q \vee \overline{R}S$

Digitale Speicherelemente

- Erkennbar gehören RS-NOR- und RS-NAND-FF zur Klasse 1, RS-FF mit Enable-Eingang zur Klasse 2
- Weitere Typen: Das D-Latch (Delay-Latch)

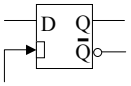




Auftrennung der Rückkopplung

Speicherzustände

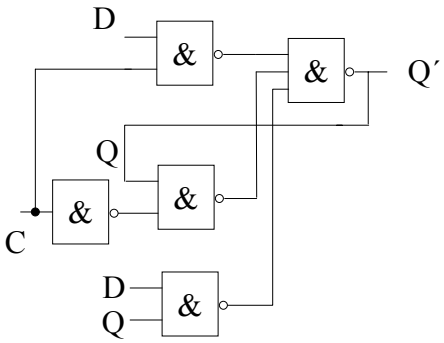
$Q' = CD \vee \overline{C}Q$

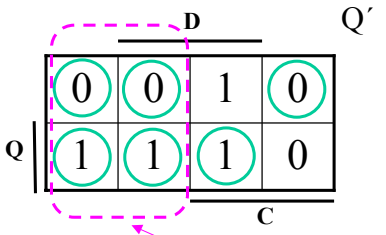


- Für C=1 folgt Q dem Eingangswert von D, für C=0 wird letzter D-Wert gespeichert (Taktpegelsteuerung; *Gated*-Latch)

Digitale Speicherelemente

- Das vorliegende Gatternetz des D-Latches hat einen Hazard
- Sensitivität bezüglich Hazardimpulsen spricht ebenfalls gegen asynchronen Entwurf
- **Bereinigte Variante**:



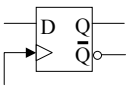


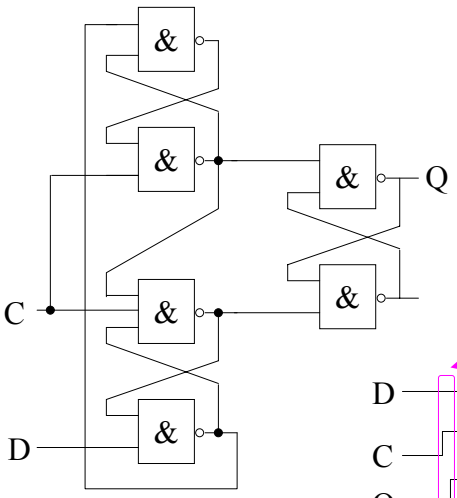
Speicherzustände

$$Q' = CD \vee \overline{C}Q \vee DQ$$

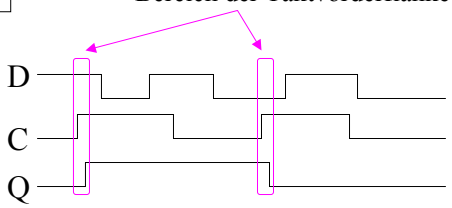
Digitale Speicherelemente

- Vorderflankengesteuertes D-FF (Klasse 3)



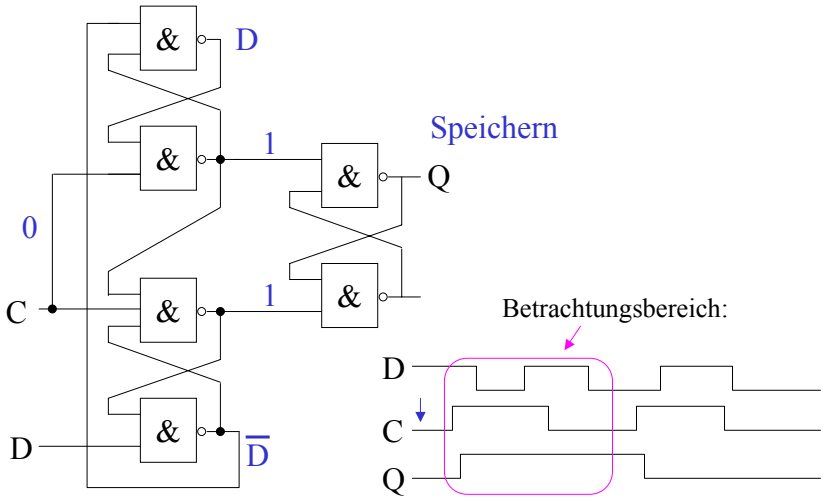


Wirkung des D-Eingangs nur im Bereich der Taktvorderflanke



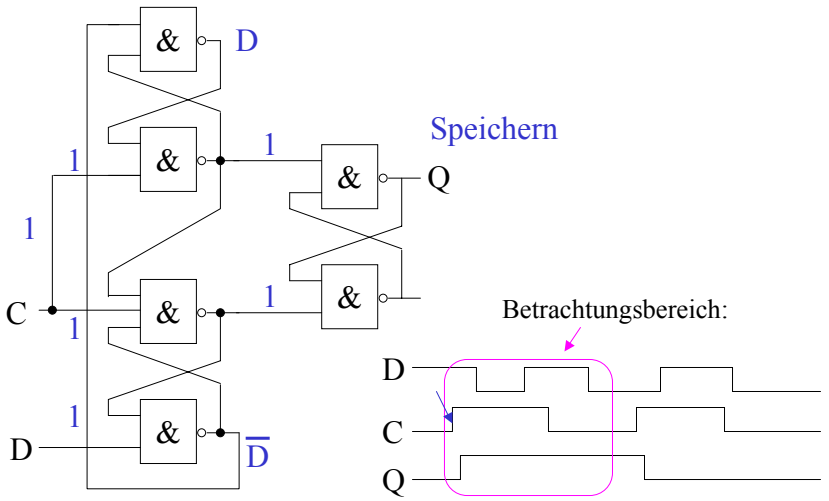
Digitale Speicherelemente

- Ablauf der Übernahme im vorderflankengesteuerten D-FF:
- C=0: Speichern

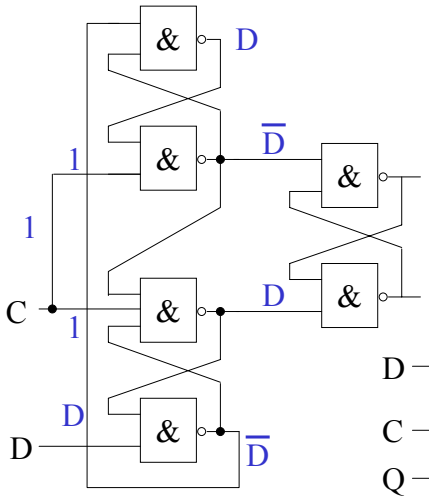


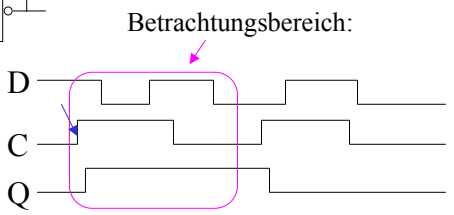
Digitale Speicherelemente

- C wechselt auf 1 (Schritt 1):

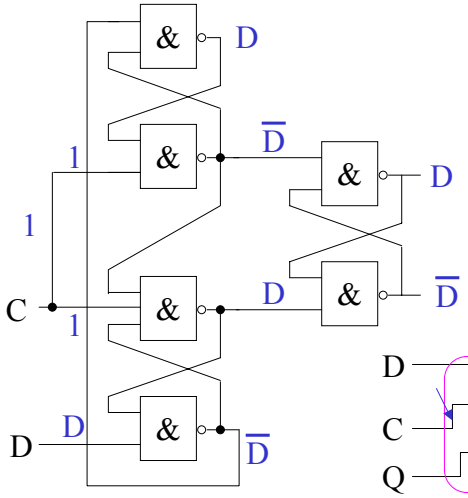


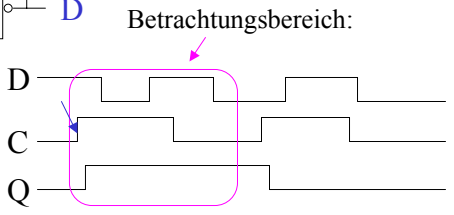
- C wechselt auf 1 (Schritt 2):





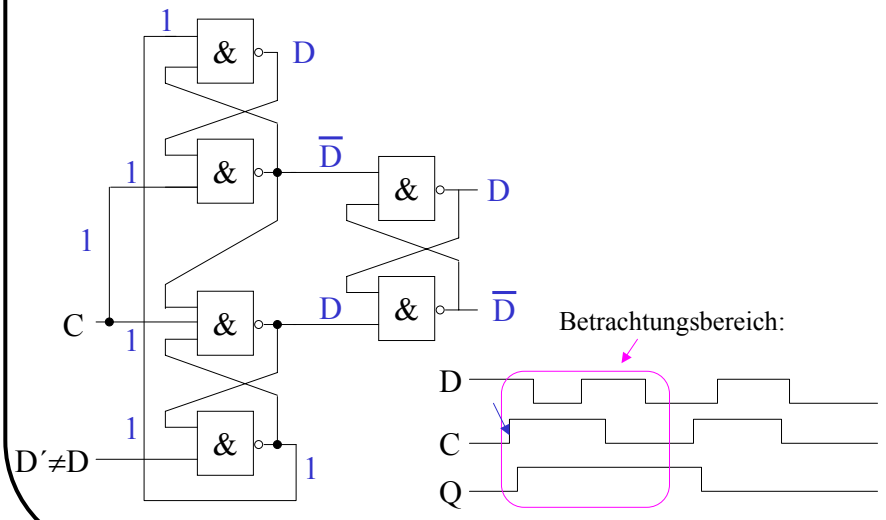
- C wechselt auf 1 (Schritt 3):





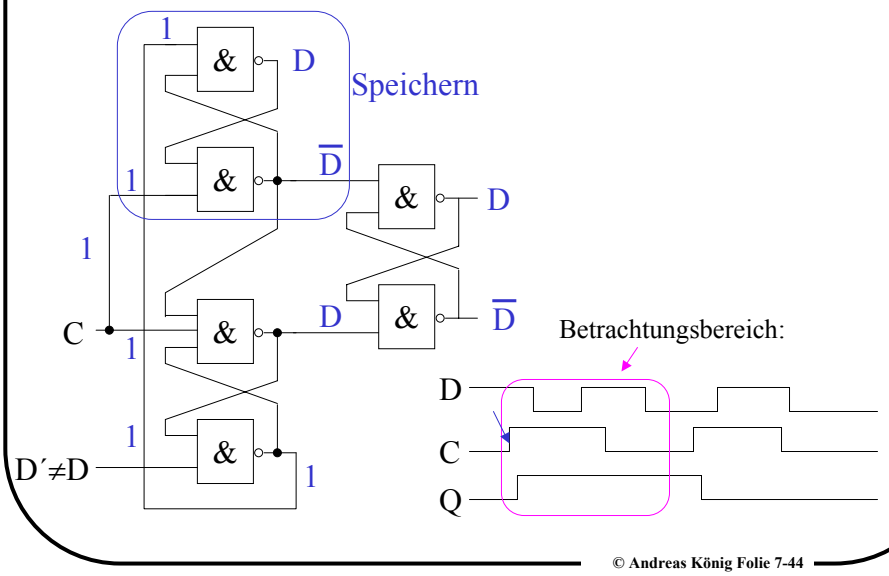
Digitale Speicherelemente

- C=1 und D ändert sich:



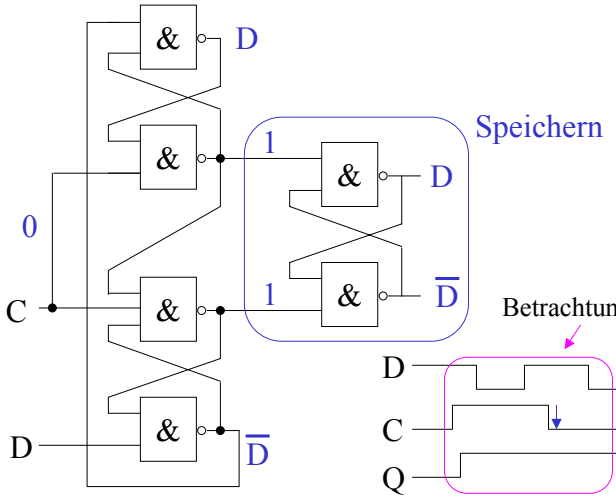
Digitale Speicherelemente

- Oberes Teil-FF speichert D, sonst keine Wirkung:



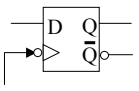
Digitale Speicherelemente

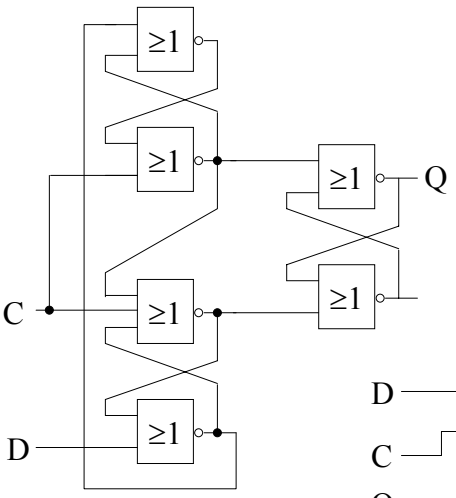
- C geht auf 0 zurück: Speichern



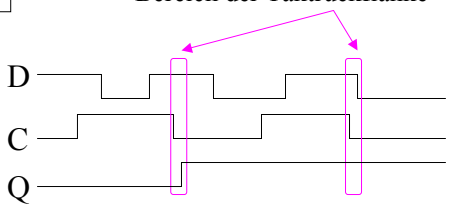
Digitale Speicherelemente

- Rückflankengesteuertes D-FF (Klasse 3)



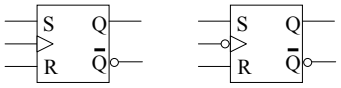


Wirkung des D-Eingangs nur im Bereich der Taktrückflanke



Digitale Speicherelemente

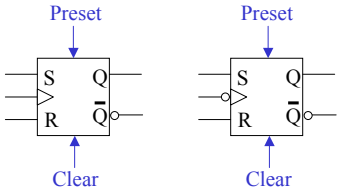
- Weitere FF-Typen: Das SR-FF (Setz-Rücksetz-Flipflop)
- Ein- bzw. zweiflankengesteuert



Einflankensteuerung
steigende fallende

S	R	Q´
0	0	Speichern
0	1	0
1	0	1
1	1	Verboten

Für C geht von 0 nach 1
bzw. von 1 nach 0
Sonst speichern

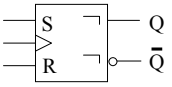


Asynchrone Voreinstellung

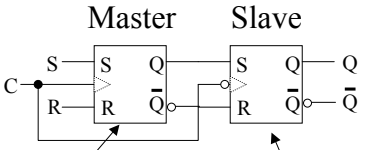
$$Q' = \overline{R} \vee QS$$

Digitale Speicherelemente

- Das SR-FF (Setz-Rücksetz-Flipflop)
- Ein- bzw. zweiflankengesteuert



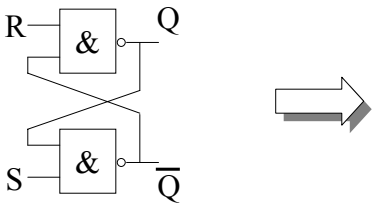
Zweiflankensteuerung
gepuffert (zweistufig)
ggf. mit Data-Lockout

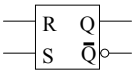


Übernahme für C=1 Übernahme für C=0

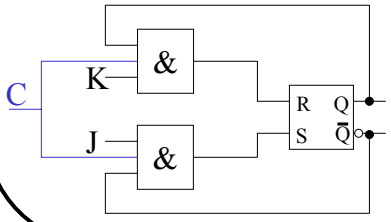
- Die für taktflankengesteuerte FF bestimmten charakteristischen Gleichungen gelten ebenfalls für taktzustandsgesteuerte FFs und umgekehrt

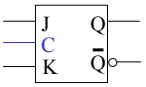
- Weitere FF-Typen: Das JK-Latch (ungepuffert) bzw. JK-FF
- Motivation: Beseitigung des verbotenen Zustands von RS-FF





$$Q' = \overline{R} \vee QS$$

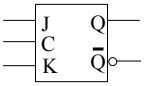




$$Q' = Q\overline{K} \vee \overline{Q}J$$

- Wahrheitstabelle und KV-Diagramm zum JK-Latch

Clock	J	K	Q´
0	0	0	Speichern
0	0	1	Speichern
0	1	0	Speichern
0	1	1	Speichern
1	0	0	Speichern (Q)
1	0	1	0
1	1	0	1
1	1	1	Invertieren ($\overline{Q}$) (Toggle)

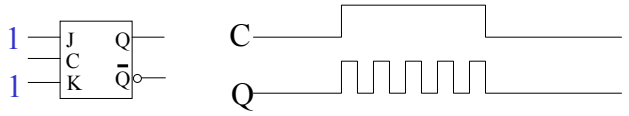


$$Q' = Q\overline{K} \vee \overline{Q}J$$

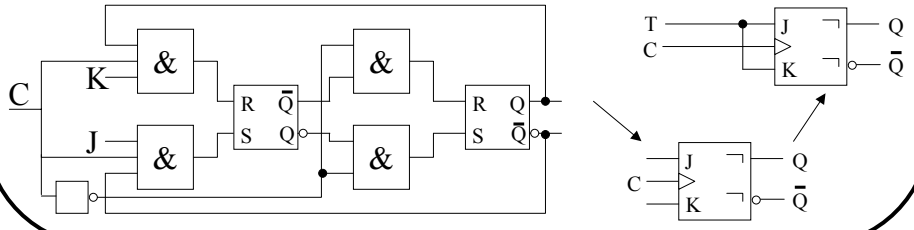
Q \ J,K	00	01	11	10
0	0	1	1	0
1	1	1	0	0

Q´

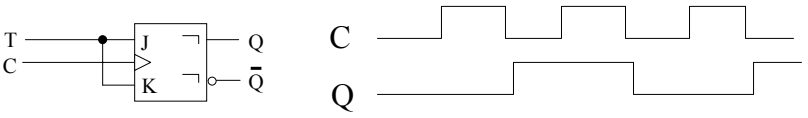
- JK-MS-FF und das T-FF (Toggle-FF)
- Erkennbar ist der verbotene Zustand des RS-FF beseitigt
- Eine Belegung J=K=1 führt zum Invertieren (Umkippen, *Toggle*) des Ausgangs



- Im ungepufferten FF erfolgt die Toggle Funktion asynchron
- JK-MS-FF und seine Beschaltung als Toggle-FF (T-FF):

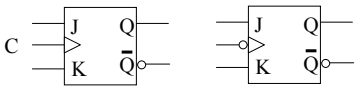


- Es existiert kein eigentliches T-FF, da es durch geringfügige Beschaltung aus anderen Flipflops gewonnen werden kann



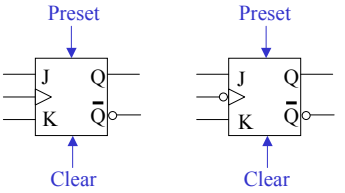
- **Resultat:** Takthalbierung am Ausgang Q

- Taktgesteuertes JK-FF: Ein- bzw. zweiflankengesteuert



Einflankensteuerung
steigende fallende

J	K	Q´
0	0	Speichern (Q)
0	1	0
1	0	1
1	1	Invertieren (Q)

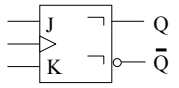


Asynchrone Voreinstellung

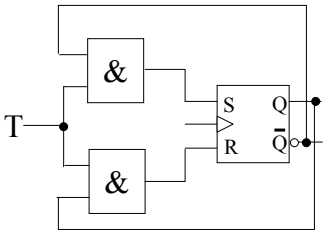
Für C geht von 0 nach 1
bzw. von 1 nach 0
Sonst speichern

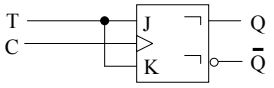
$$Q' = Q\overline{K} \vee \overline{Q}J$$

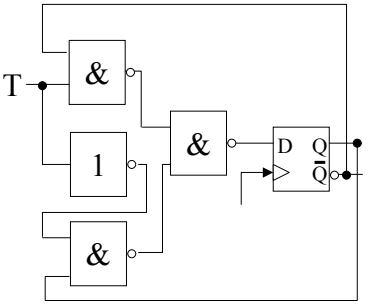
JK-MS-FF
(zweiflankengesteuert)



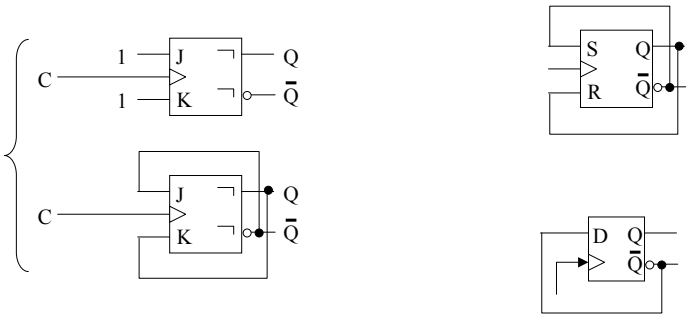
- Durch externe Beschaltung können vorliegende Flipflop-Typen in andere mögliche Flipflop-Typen umgewandelt werden
- Beispiel T-FF:



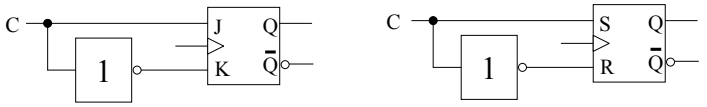




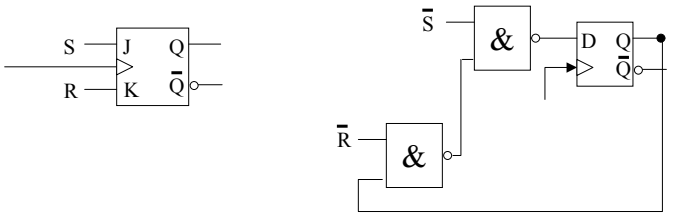
- Beispiel 2:1 Frequenzteiler (*Binäruntersetzer*):



- Beispiel D-FF:

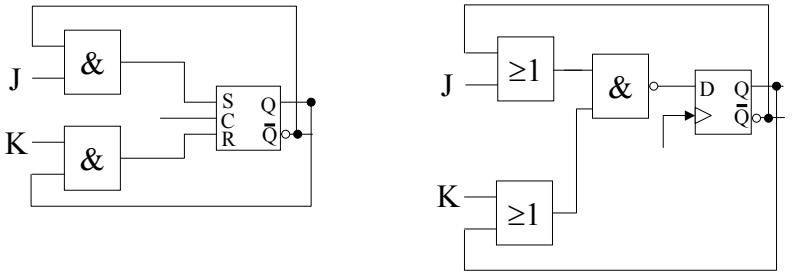


- Beispiel RS-FF:



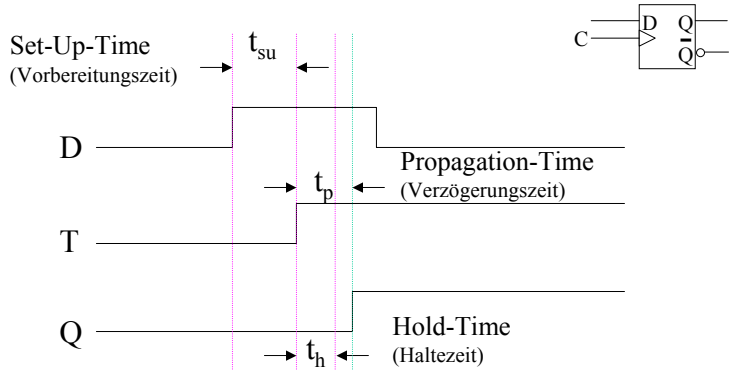
Digitale Speicherelemente

- Beispiel JK-FF

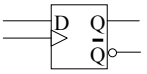


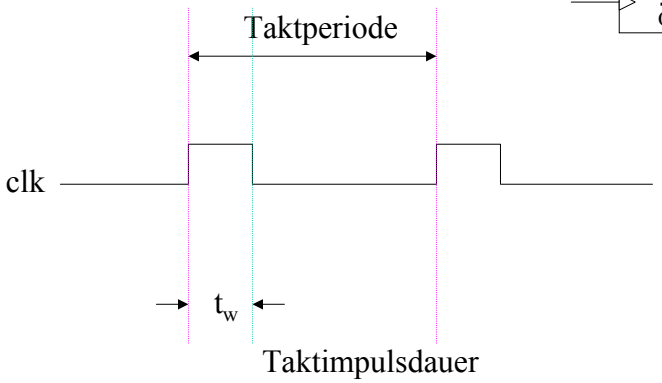
Digitale Speicherelemente

- Fragen der zeitlichen Ansteuerung von Flipflops zum sicheren Betrieb synchoner Schaltwerke
- Die Darstellung insbesondere flankengesteuerter FFs hat gezeigt, dass an Takt und Daten an den Vorbereitungseingängen im Hinblick auf die internen Vorgänge des FF besondere zeitliche Anforderungen gestellt sind
- Beispiel: vorderflankengesteuertes D-FF

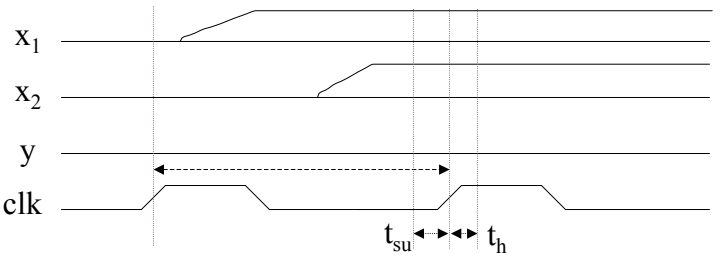


- Die vorgestellten Zeitbedingungen müssen eingehalten werden, um eine korrekte Informationsübernahme sicher zustellen
- Daraus resultieren Anforderungen an Taktfrequenz und Taktimpulsdauer bzw. Taktpausenverhältnis



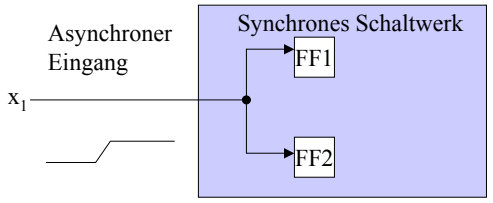


- In Verbindung mit Schaltnetzen muss der gewählte Taktabstand sicherstellen, dass zum nächsten Takt alle Ausgänge des Schaltnetzes stabil sind
- **Probleme:** Laufzeiten in Gattern und FFs, Hazards



Digitale Speicherelemente

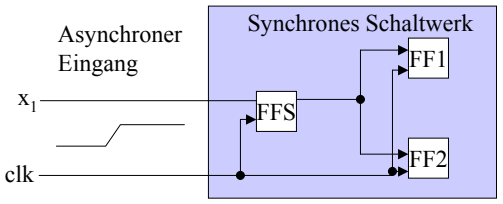
- Ein generelles Problem ist durch asynchrone Eingänge in synchronen Schaltwerken gegeben
- Hier kann die Einhaltung von Zeitbedingungen nicht garantiert werden



- Ein Übergang am asynchronen Eingang kann von einem FF (Gatter) noch als 0 und von einem anderen schon als 1 interpretiert werden
- Damit hat eine Variable nicht mehr überall zu jedem Zeitpunkt einen eindeutigen Wert
- **Folge:** Mögliche fehlerhafte Übernahme beim nächsten Takt

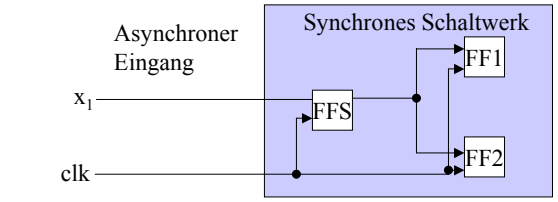
Digitale Speicherelemente

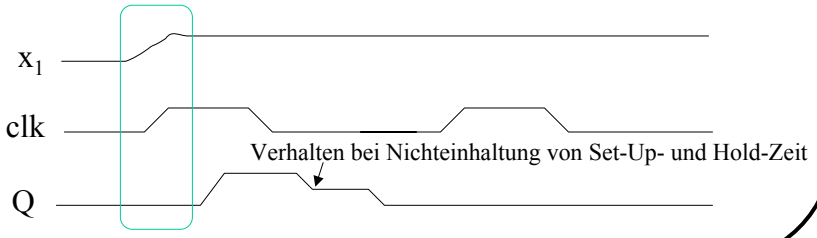
- **Abhilfe:** Einsynchronisierung von asynchronen Signalen durch zusätzliches Synchronisierungs-FF (s. z.B. [Katz 94])



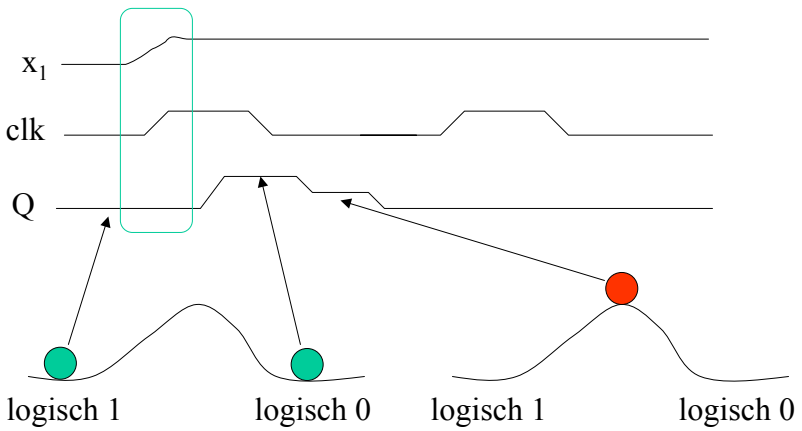
- Das Synchronisierungs-FF FFS übernimmt mit dem Takt am Eingang des Schaltwerks den Wert des asynchronen Eingangs
- Der Ausgangswert von FFS kann bei gegebenem hinreichenden Taktabstand im Schaltwerk propagieren
- Die folgenden Speicherglieder übernehmen alle im nächsten Takt korrekt

- **Bleibendes Problem:** Die Set-Up- und Hold-Zeiten können bei den asynchronen Eingängen anschaulich nicht garantiert werden

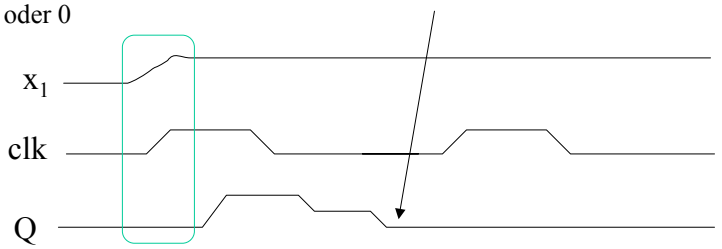




- Verhalten bei Nichteinhaltung von Set-Up- und Hold-Zeit kann undefiniert werden
- Problem des Auftretens metastabilen Verhaltens

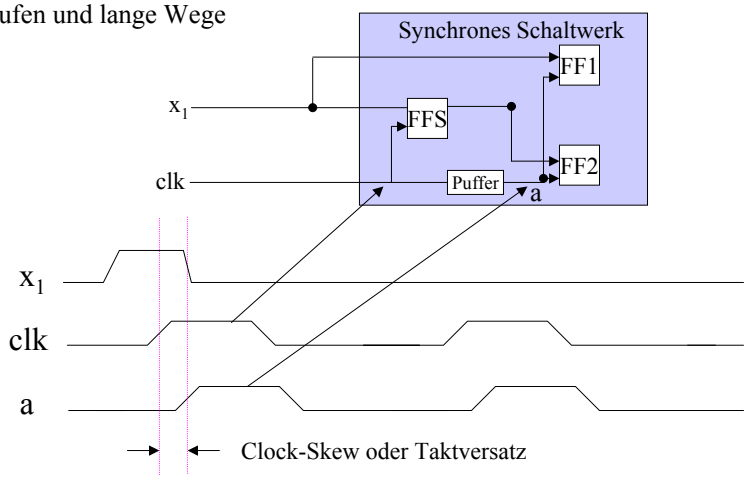


- Metastabile FFS-Ausgänge können folgende Fehlinterpretationen bewirken
- Theoretisch könnte Flipflop in diesem metastabilen Zustand verharren
- Vorliegende Asymmetrien der Realisierung und thermische Anregung führen nach endlicher Zeit zu einem Verlassen des Zustands und Übergang zu 1 oder 0



- **Gegenmassnahmen:**
  - Schnellstmögliches Synchronisierungs-FF (t_{su} und t_h klein)
  - Serienschaltung mehrerer FFS
  - Hinreichender Taktabstand, so dass stabiler Zustand erreicht werden kann
  - *Hoffen auf statistische Beherrschbarkeit (unwahrscheinliches Ereignis)*

- **Weiteres Problem:** Die Taktverteilung in digitalen System ist ein ernstes Problem wachsender Bedeutung
- Fan-Out Problem durch viele Takteingänge, Laufzeitproblem durch Gatterstufen und lange Wege



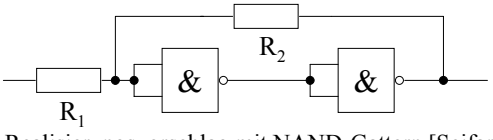
Digitale Speicherelemente

- Die Taktverteilung in digitalen Systemen ist ein vor allem für die immer größer werdenden integrierten Systeme zunehmend problematisch
- Mit größer werdender Die- und genutzter Chipfläche wachsen die Verdrahtungswege und damit sowohl die Zahl angeschlossener Eingänge als auch die kapazitive Belastung durch die Verdrahtungsfläche selbst
- **Folge:** Taktflanken werden *schwammig,* d.h. das Übergangsverhalten verliert an Steilheit
- Treiberstufen zur Taktverteilung erforderlich (Fan-Out)
- Verzögerungszeiten der Treibergatter und Laufzeiten der Leitungen verursachen erhebliche Probleme durch Taktversatz
- Möglicher Systemtaktrate werden Grenzen gesetzt
- **Abhilfemöglichkeiten:** Lokal getaktete Systeme oder Self-Timed Circuits
  - Systeme mit lokalen (schnell)getakteten Komponenten, die über spezielle Schnittstellen und Protokolle, z.B. *Handshake*, kommunizieren
  - Systeme ohne eigenen Takt, die Kommunikationssignale zum Datenaustausch anhand der Kenntnis ihrer eigenen Bearbeitungslatenz generieren

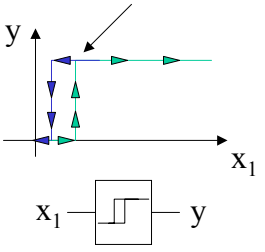
Vertiefungsthema Hauptstudium

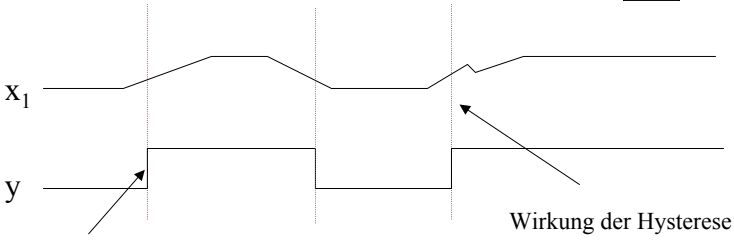
Digitale Speicherelemente

- Weitere wichtige Bausteine der Digitaltechnik
- **Schmitt-Trigger:** Bistabiles Schwellwertelement typisch mit Hysterese-Verhalten



Realisierungsvorschlag mit NAND-Gattern [Seifart 98]



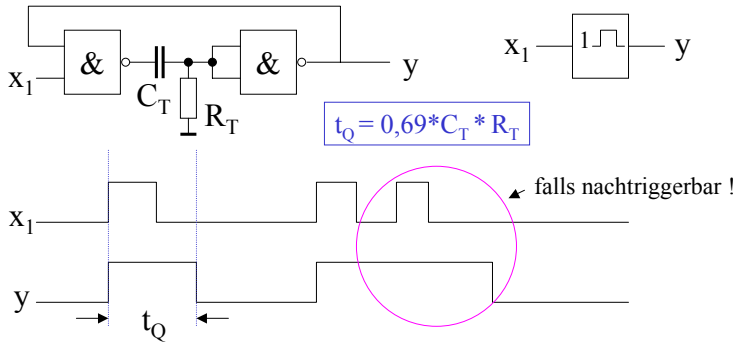


Flankenregeneration von Signalen

Wirkung der Hysterese

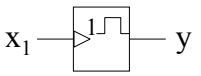
Digitale Speicherelemente

- **Univibrator oder Monoflop:** dynamisches Speicherelement das nach Anlegen eines Triggerimpulses aus dem stabilen Ruhezustand in einen zeitlich begrenzten (metastabilen) Verweilzustand kippt
- Rückkehr in den Verweilzustand nach Ablauf der Verweilzeit t_Q
- Unterscheidung in nachtriggerbare und nichtnachtriggerbare Monoflops, d.h. Ausdehnung der Verweilzeit durch neue Triggerimpulse im Verweilzustand möglich bzw. nicht möglich

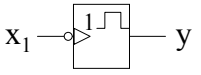


Digitale Speicherelemente

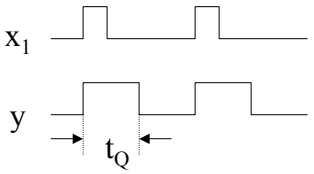
- Flankengesteuerte monostabile Kippstufe

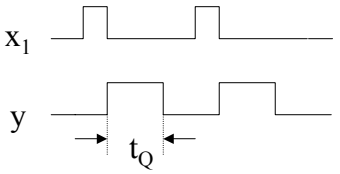


positiv flankengetriggert



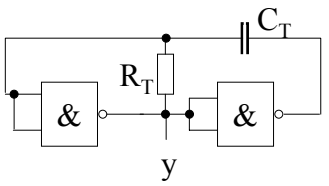
negativ flankengetriggert





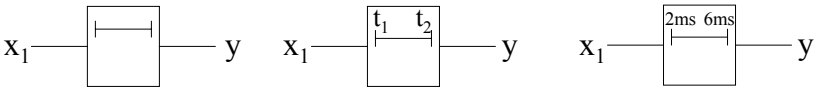
- Erzeugung von Rechteckimpulsen vorgebbarer Dauer, z.B. Schrittmotor-ansteuerung o.ä. Interface-Aufgaben
- Realisierung von Zeitverzögerungen (Entprellen von Schaltern)
- Dynamische Speicherung von Binärsignalen
- Typischer diskreter Baustein: Timer 555 oder 74121 + ext. R,C
- Immer häufiger ersetzt durch präzisere Zähler/Frequenzteiler-Anordnung

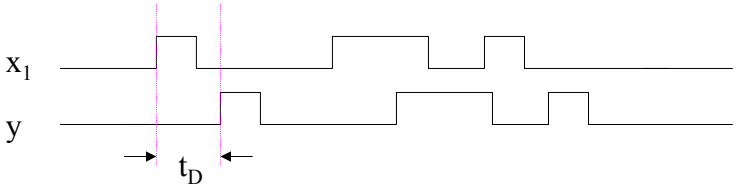
- **Multivibratoren:** Astabile Elemente, die abhängig von zeitbestimmenden Größen zwischen zwei quasistabilen Zuständen hin- und herwechseln
- Schwingungs- oder Takterzeugung
- Aufbau mit einfachen Gattern [Seifart 98]:



Prinzipielles Verhalten der Impulserzeugung

- **Verzögerungsglied:** Element mit einer verzögerten Weitergabe des Binärsignals
- Steigende bzw. fallende Flanken können unterschiedliche Verzögerungszeiten t_1 bzw. t_2 erfahren

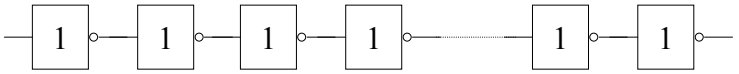




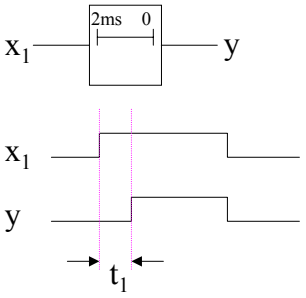
Prinzipielles Verzögerungsverhalten für $t_1 = t_2 = t_D$

Digitale Speicherelemente

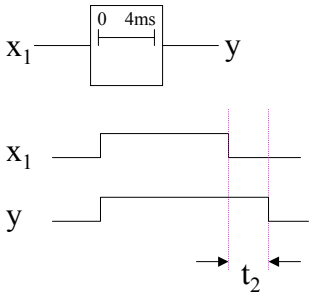
- Mögliche Realisierungsform aus n Inverterstufen:



Einschaltverzögerungsglied

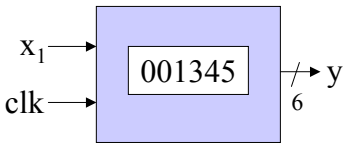


Ausschaltverzögerungsglied

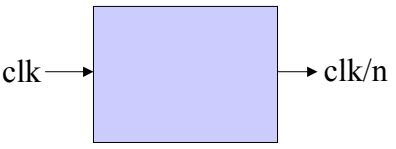


Zähler und Frequenzteiler

- Die Zählung von Taktimpulsen oder Ereignissen ist eine wichtige Funktion bei der Realisierung von Anwendungssystemen
- Simple Beispiele:
  - Verbrauchserfassung bei Strom, Wasser und Gas
  - Telefoneinheiten
  - Zugriffsanzahl auf Internet-Seiten



Zähler mit Ausgang

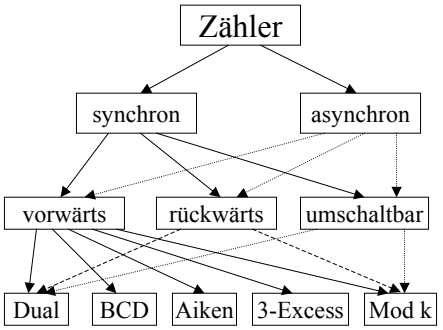


Frequenzteiler n:1

- Zähler und Frequenzteiler sind essentielle Komponenten für die Realisierung digitaler Systeme, z.B. für Echtzeitsteuerung
- Implementierung durch Flipflops und zusätzliche Logik

Zähler und Frequenzteiler

- Unterteilung der Zähler in wichtige Untergruppen nach Funktion und Implementierungstruktur:
  - Zählrichtung
    - Vorwärts
    - Rückwärts
    - Programmierbare Richtung
  - Implementierung:
    - Synchron
    - Asynchron
  - Kodierung
    - Dual-Code
    - BCD-Code
    - Aiken-Code
    - 3-Excess-Code
    - Mod-k (programmierbar)

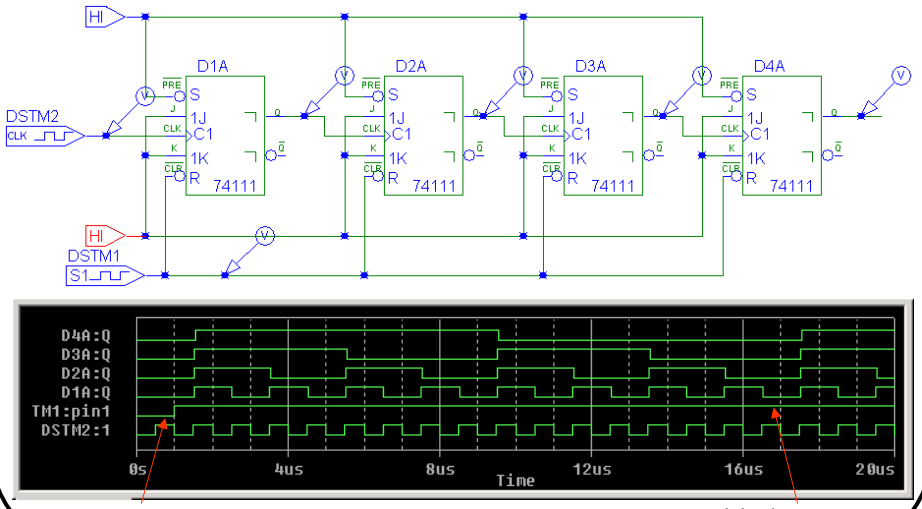


Zähler und Frequenzteiler

- Bei asynchronen Zählern werden die einzelnen Stufen nicht durch einen zentralen Takt gesteuert
- Zustandswechsel erfolgen asynchron in den einzelnen Stufen
- Ggf. propagiert eine Serie von Zustandswechseln über alle Stufen
- Erst nach dieser (maximalen) Zeit darf das Zählerergebnis von Folgeeinheiten ´´ernst´´ genommen werden
- Vorteil ist jedoch geringerer Verknüpfungsaufwand
- Rein asynchrone Schaltungen nur für Dualzähler
- Bei synchronen Zählern wechseln alle Flipflops der Zählerstufen gleichzeitig taktsynchronisiert den Zustand
- Höherer Verknüpfungsaufwand aber freie funktionale Gestaltung durch Verknüpfung
- Häufig höhere Zählgeschwindigkeiten; höhere Robustheit
- Unterschied Zähler/Frequenzteiler: Gesamter Zählerstand bzw. nur Ausgang letzter Stufe herausgeführt

Zähler und Frequenzteiler

- Asynchroner Dualzähler (rückwärts):

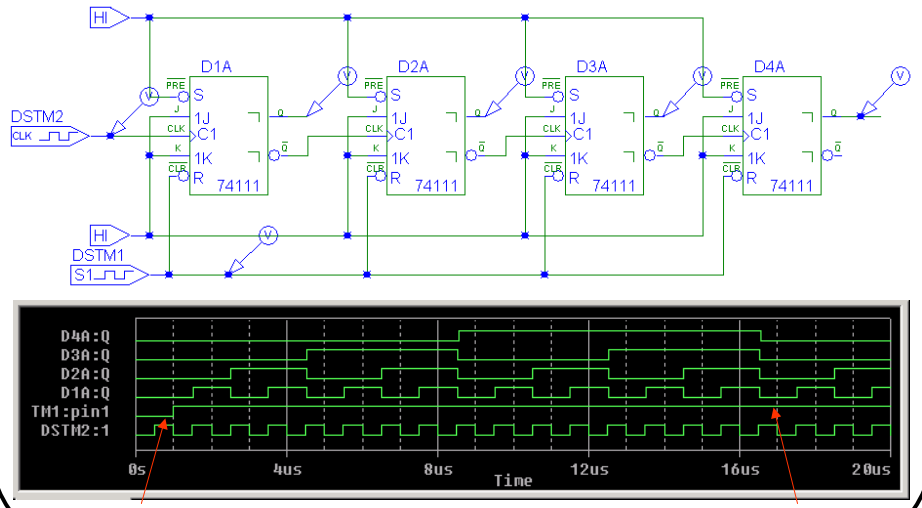


Reset

Rückkehr zu 0

Zähler und Frequenzteiler

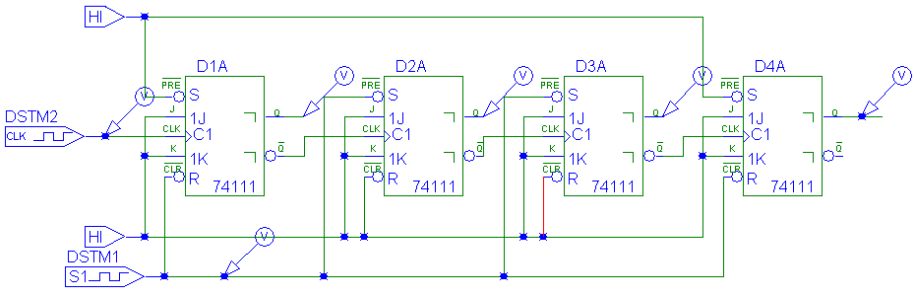
- Asynchroner Dualzähler (vorwärts):

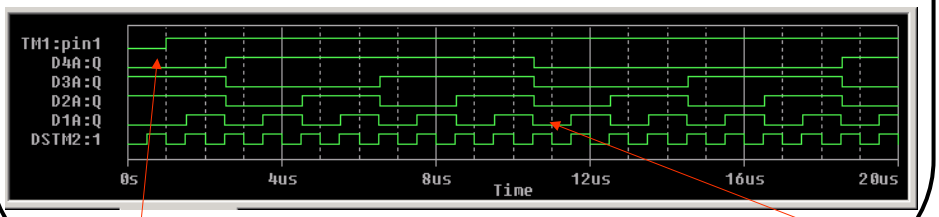


Reset

Rückkehr zu 0

- Asynchroner Dualzähler (vorwärts, Anfangswert 0110):

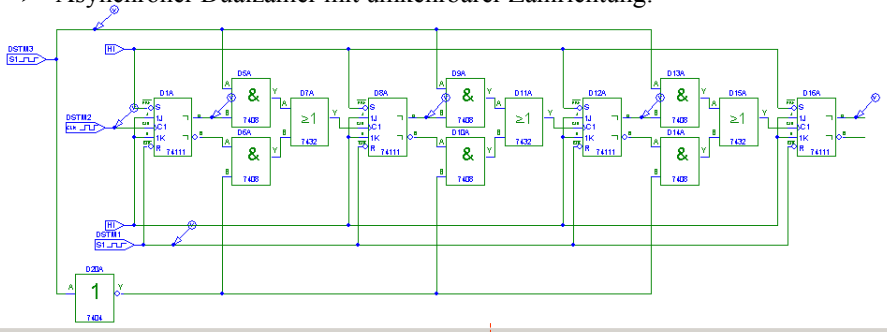


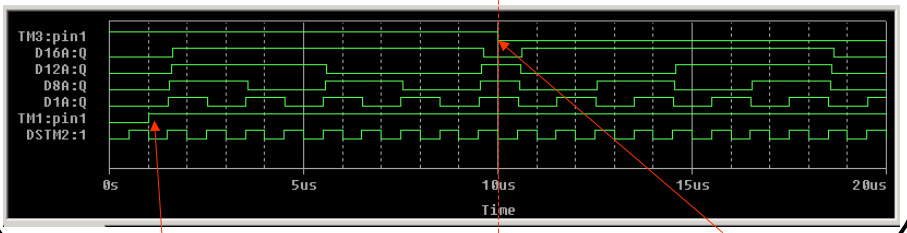


Vorsetzen auf 0110

Rückkehr zu 0

- Asynchroner Dualzähler mit umkehrbarer Zählrichtung:



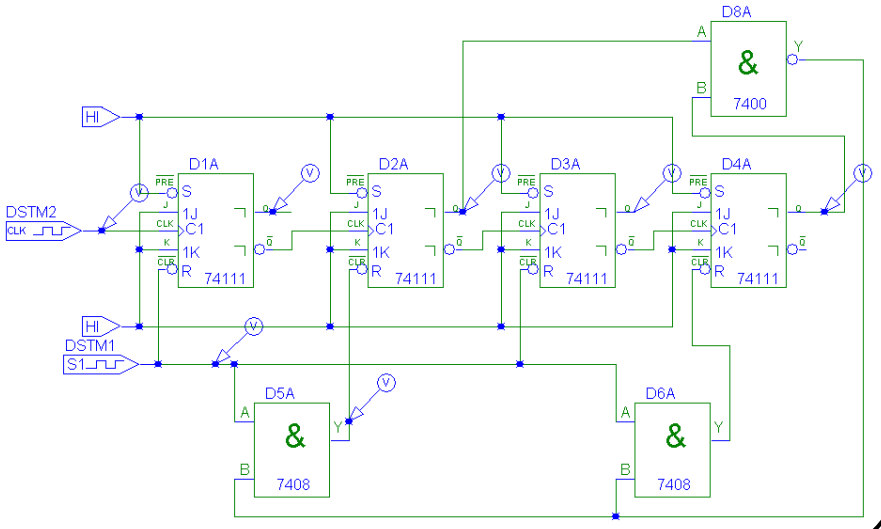


Vorsetzen auf 0000

Zählrichtungsumkehr

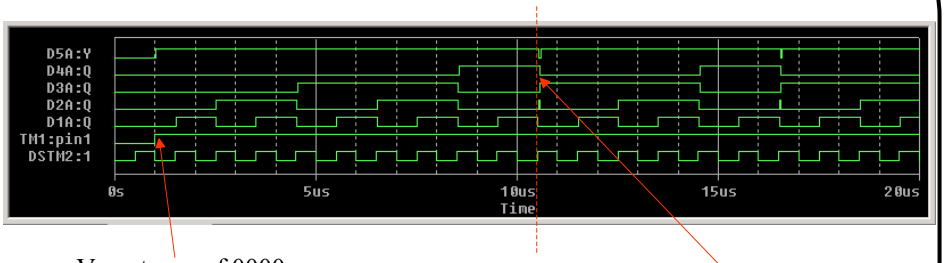
Zähler und Frequenzteiler

- Asynchroner BCD-Vorwärtszähler:



Zähler und Frequenzteiler

- Asynchroner BCD-Vorwärtszähler:



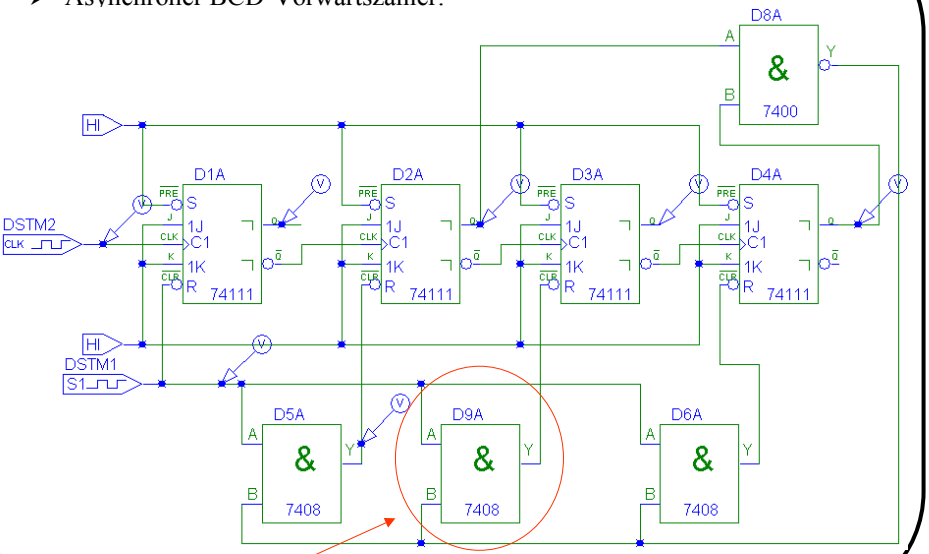
Vorsetzen auf 0000

Fehlerhafte Zählrücksetzung bei 1010 auf 0100

- Eigentlich ist nur eine Rücksetzung der Stufen 2 und 4 des Zählers erforderlich, da der Zustand 1010 nach 0000 überführt werden soll
- Die konkrete Realisierung weist jedoch ein zeitliches Fehlverhalten auf, so dass 1010 auf 0100 übergeht
- **Abhilfe:** Gezielte zusätzliche Rückstellung der Stufe 3

Zähler und Frequenzteiler

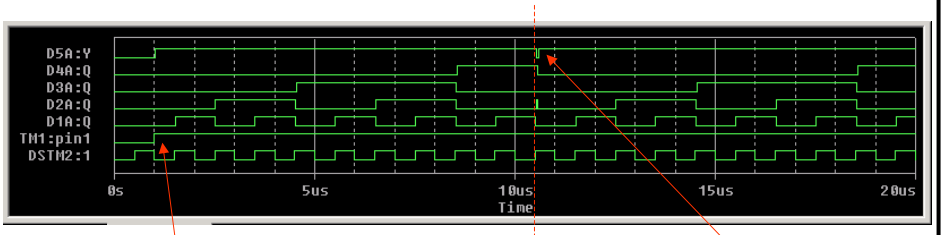
- Asynchroner BCD-Vorwärtszähler:



Zusätzliche Rücksetzung von FF3 (D3A)

Zähler und Frequenzteiler

- Asynchroner BCD-Vorwärtszähler:



Vorsetzen auf 0000

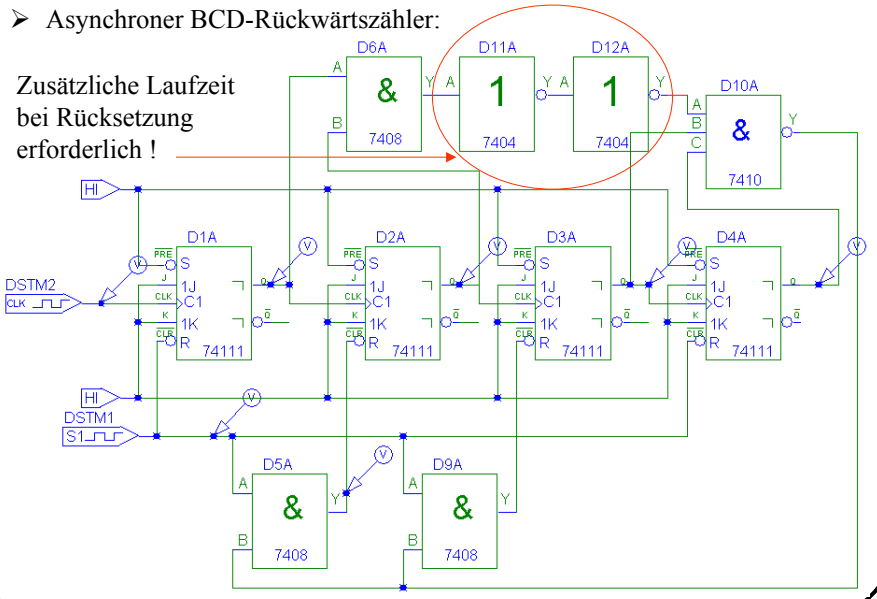
Zählerrücksetzung bei 1010

- Asynchroner BCD-Vorwärtszähler wird nun zyklisch korrekt zurückgesetzt
- Die Gatter zur Anfangsinitialisierung können ggf. eingespart werden, wenn der Zähler freilaufend arbeiten soll
- Generell muss prinzipiell für alle betrachteten Zähler ein Anfangszustand zu einem bestimmten zeitlichen Moment synchron zum Systemtiming sichergestellt werden können

Zähler und Frequenzteiler

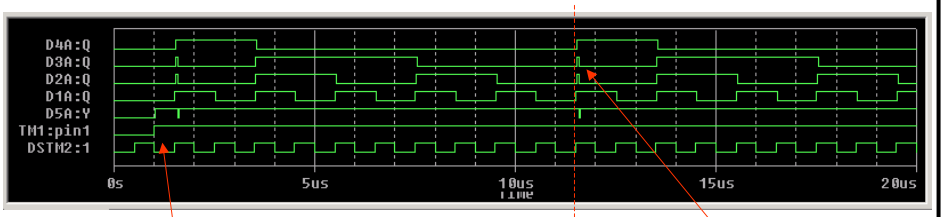
- Asynchroner BCD-Rückwärtszähler:

Zusätzliche Laufzeit bei Rücksetzung erforderlich !



Zähler und Frequenzteiler

- Asynchroner BCD-Rückwärtszähler:



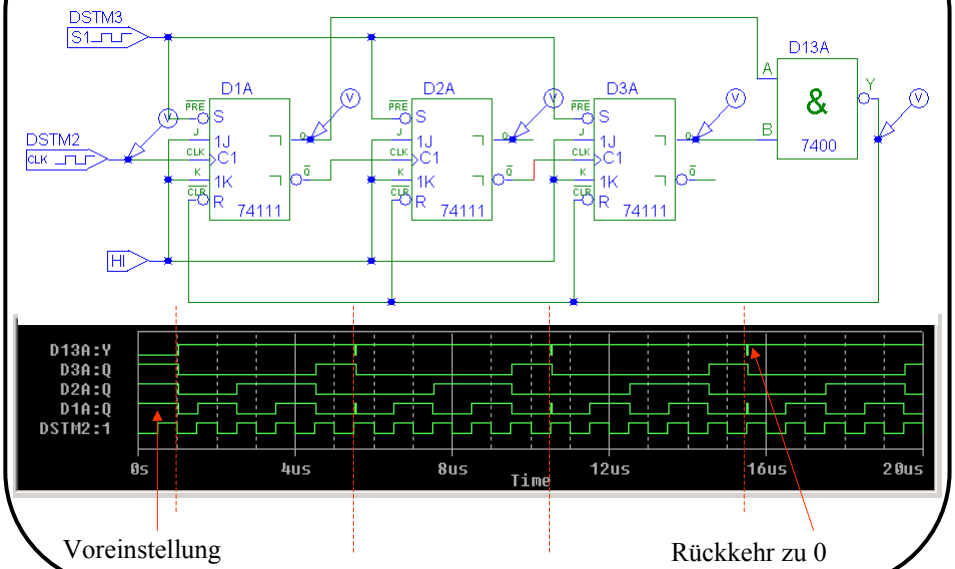
Vorsetzen auf 0000

Zählerrücksetzung bei 1111

- Asynchroner BCD-Rückwärtszähler zählt von 1001 bis 0000 rückwärts
- Bei Erreichen von 1111 durch Abzählen von 0000 muss auf 1001 umgesetzt werden
- Auch bei dieser konkreten Realisierung sind Laufzeiteffekte kritisch
- Inverterkette dient zur Verzögerung um korrektes Verhalten zu erreichen !

Zähler und Frequenzteiler

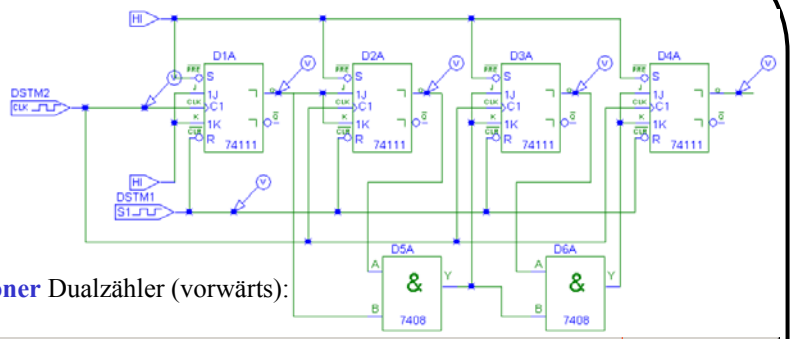
- Asynchroner Mod-5-Zähler (vorwärts):



Voreinstellung

Rückkehr zu 0

Zähler und Frequenzteiler

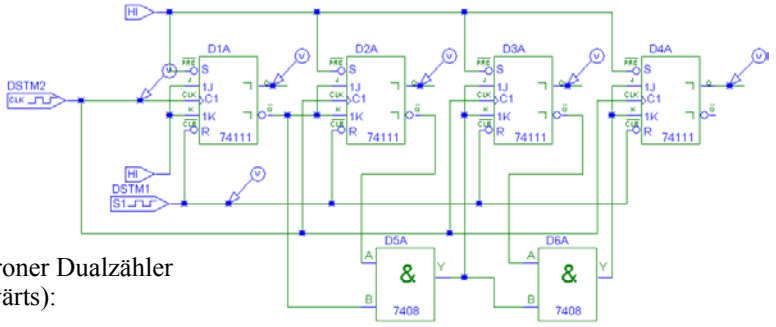


- **Synchroner** Dualzähler (vorwärts):

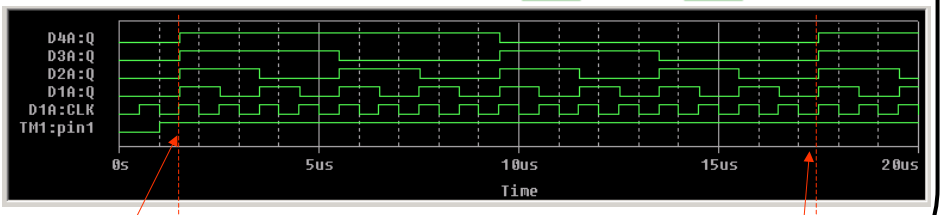
Initialisierung auf 0000

Rückkehr zu 0000

Zähler und Frequenzteiler



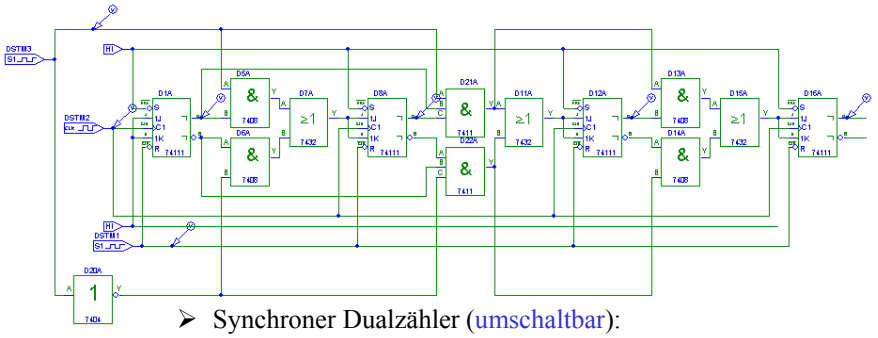
- Synchroner Dualzähler (rückwärts):



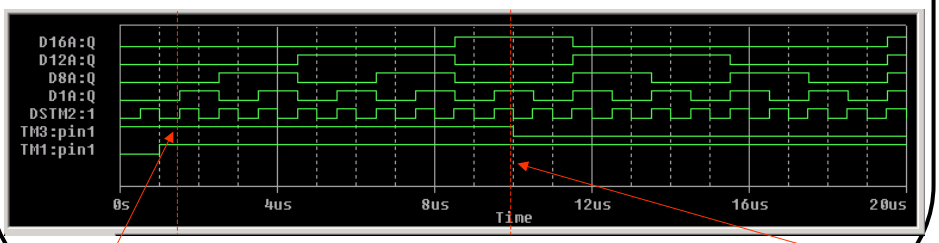
Initialisierung auf 0000

Rückkehr zu 1111

Zähler und Frequenzteiler



- Synchroner Dualzähler (umschaltbar):



Initialisierung auf 0000

Zählrichtungsumkehr

- Entwurf eines gewünschten Zählertyps !
  - Auswahl des Flipflop-Typs, z.B. JK oder D gepuffert
  - Anzahl der FFs anhand des Zählbereichs
  - Verknüpfungsnetzwerk anhand vorgegebener Zustandsübergangstabelle
  - Erkennbar: Es liegt ein Medwedew-Automat vor, bei dem das Eingangsalphabet z.B. die leere Menge ist, d.h.:

$$O_h^t = \lambda\left(S_k^t\right) = S_k^t$$

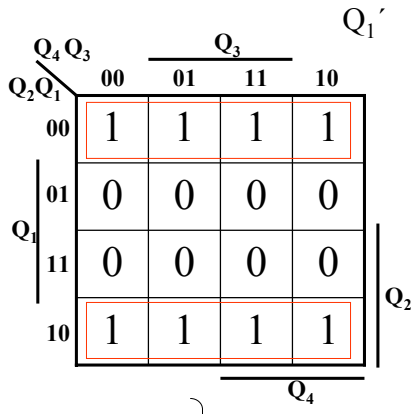
$$S_k^{t+1} = \delta\left(I_j^t, S_k^t\right) = \delta\left(S_k^t\right)$$

Ausnahme, z.B. Richtungsumschaltung

- Einfache Vorgehensweise beim Entwurf:
  - Wahrheitstabelle für die Verknüpfungsfunktion jedes FFs
  - Logikminimierung
  - Koeffizientenvergleich der DMF mit charakteristischer Gleichung des FF-Typs
  - Startzustand durch Belegung der Preset/Clear-Eingänge festlegen (*Zusatzgatter*)
  - Gesamtschaltplan aus Gattern und FFs aufbauen; Simulieren

- Entwurfsbeispiel: Synchroner 4 bit BCD-Vorwärtszähler:
- **Einfache Lösung:** Synchroner Dualzähler mit asynchronem Rücksetzen

$Q_4Q_3Q_2Q_1$	$Q_4'Q_3'Q_2'Q_1'$
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	1010
1010	1011
1011	1100
1100	1101
1101	1110
1110	1111
1111	0000



$$Q_1' = \overline{Q_1}$$

$$Q' = Q\overline{K} \vee \overline{Q}J$$

J=1, K=1

➢ Entwurfsbeispiel: Synchroner 4 bit BCD-Vorwärtszähler:

$Q_4Q_3Q_2Q_1$	$Q_4'Q_3'Q_2'Q_1'$
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	1010
1010	1011
1011	1100
1100	1101
1101	1110
1110	1111
1111	0000

Q_2'

Q_2Q_1 \ Q_4Q_3	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

Q_1, Q_2, Q_3, Q_4

$$Q_2' = Q_2\overline{Q_1} \vee \overline{Q_2}Q_1$$

$$Q' = Q\overline{K} \vee \overline{Q}J$$

J=Q_1, K= Q_1

➢ Entwurfsbeispiel: Synchroner 4 bit BCD-Vorwärtszähler:

$Q_4Q_3Q_2Q_1$	$Q_4'Q_3'Q_2'Q_1'$
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	1010
1010	1011
1011	1100
1100	1101
1101	1110
1110	1111
1111	0000

Q_3'

Q_2Q_1 \ Q_4Q_3	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	0	1
10	0	1	1	0

Q_1, Q_2, Q_3, Q_4

$$Q_3' = Q_3\left(\overline{Q_2} \vee \overline{Q_1}\right) \vee \overline{Q_3}Q_2Q_1$$

$$Q' = Q\overline{K} \vee \overline{Q}J$$

J= Q_2Q_1, K= Q_2Q_1

➢ Entwurfsbeispiel: Synchroner 4 bit BCD-Vorwärtszähler:

$Q_4Q_3Q_2Q_1$	$Q_4'Q_3'Q_2'Q_1'$
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	1010
1010	1011
1011	1100
1100	1101
1101	1110
1110	1111
1111	0000

Q_4'

Q_2Q_1 \ Q_4Q_3	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	1	0	1
10	0	0	1	1

Q_1, Q_2, Q_3, Q_4

$$Q_4' = Q_4\overline{Q_2} \vee Q_4\overline{Q_1} \vee Q_4\overline{Q_3} \vee \overline{Q_4}Q_3Q_2Q_1$$
$$= Q_4\left(\overline{Q_2} \vee \overline{Q_1} \vee \overline{Q_3}\right) \vee \overline{Q_4}Q_3Q_2Q_1$$

$$Q' = Q\overline{K} \vee \overline{Q}J$$

J= $Q_3Q_2Q_1$, K= $Q_3Q_2Q_1$

➢ Entwurfsbeispiel: Synchroner 4 bit BCD-Vorwärtszähler:

$Q_4Q_3Q_2Q_1$	$Z_{Rück}$
0000	0
0001	0
0010	0
0011	0
0100	0
0101	0
0110	0
0111	0
1000	0
1001	0
1010	1
1011	1
1100	1
1101	1
1110	1
1111	1

$Z_{Rück}$

Q_2Q_1 \ Q_4Q_3	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	1
10	0	0	1	1

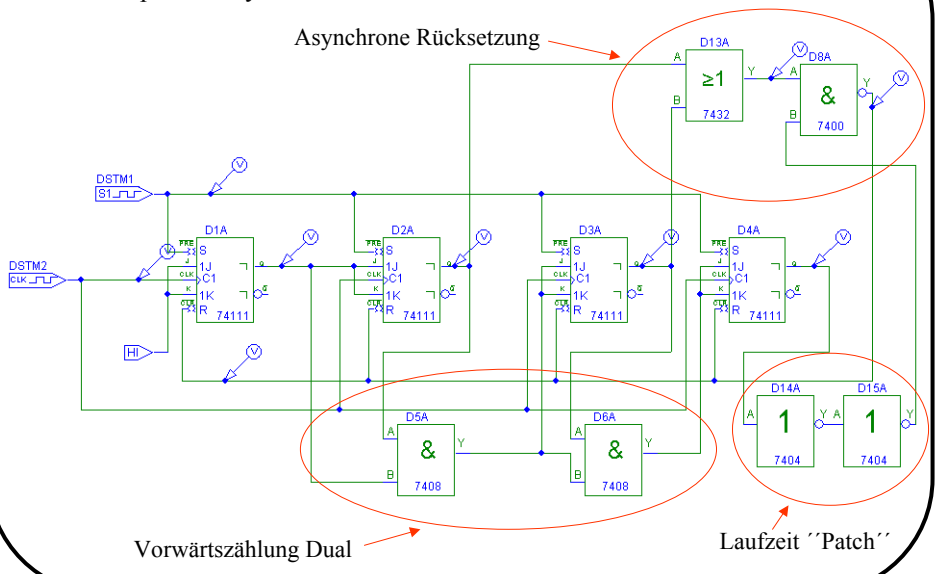
Q_1, Q_2, Q_3, Q_4

$$Z_{Rück} = Q_4Q_2 \vee Q_4Q_3$$

(Ggf. noch invertieren bei low-aktiven Eingängen)

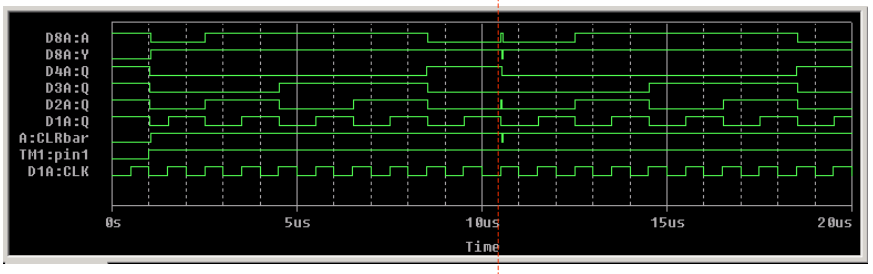
Zähler und Frequenzteiler

- Schaltplan des synchroner 4 bit BCD-Vorwärtszählers:

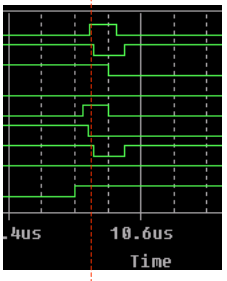


Zähler und Frequenzteiler

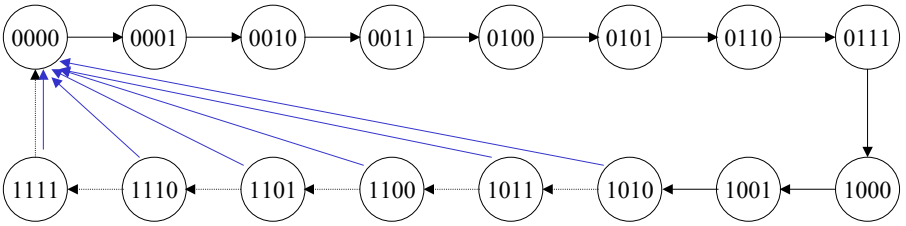
- Schaltplan des synchronen 4 bit BCD-Vorwärtszählers:



- BCD-Zähler erreicht von 1001 Zustand 1010 aus dem asynchron der Übergang nach Zustand 0000 erfolgt
- Dieser ist der Zustand, der aus allen anderen erreicht wird



- Zustandsdiagramm des synchronen 4 bit BCD-Vorwärtszählers unter Einbeziehung der über die asynchronen Eingänge bewirkten Zustandswechsel



- Entwurfsbeispiel: Rein synchroner 4 bit BCD-Vorwärtszähler:

$Q_4Q_3Q_2Q_1$	$Q_4'Q_3'Q_2'Q_1'$
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	0000
1010	0000
1011	0000
1100	0000
1101	0000
1110	0000
1111	0000

Q_4'

Q_4Q_3 \ Q_2Q_1	00	01	11	10
00	0	0	0	1
01	0	0	0	0
11	0	1	0	0
10	0	0	0	0

Q_3, Q_1, Q_2, Q_4

$$Q_4' = \overline{Q_4}Q_3Q_2Q_1 \vee Q_4\overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}$$

$$Q' = Q\overline{K} \vee \overline{Q}J$$

➢ Entwurfsbeispiel: Rein synchroner 4 bit BCD-Vorwärtszähler:

$Q_4Q_3Q_2Q_1$	$Q_4'Q_3'Q_2'Q_1'$
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	0000
1010	0000
1011	0000
1100	0000
1101	0000
1110	0000
1111	0000

Q_3'

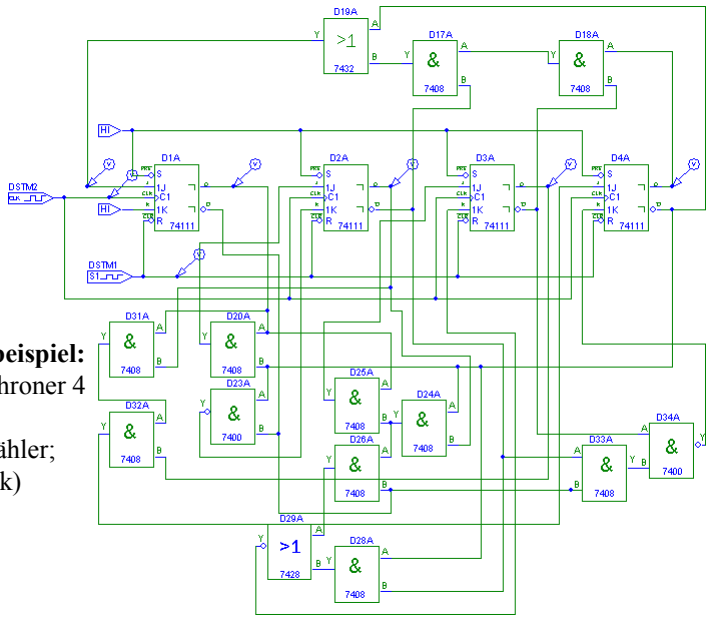
Q_1 \ Q_4Q_3	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	1	0	0	0
10	0	1	0	0

Q_2 Q_4

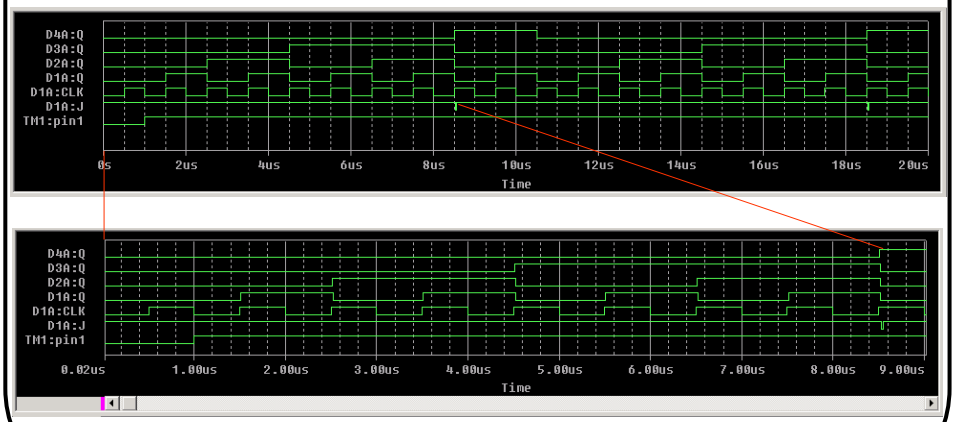
$$Q_3' = Q_3\left(\overline{Q_4}\,\overline{Q_2} \vee \overline{Q_4}Q_2\overline{Q_1}\right) \vee \overline{Q_3}\left(\overline{Q_4}Q_2Q_1\right)$$

$$Q' = Q\overline{K} \vee \overline{Q}J$$

➢ **Entwurfsbeispiel:** Rein synchroner 4 bit BCD-Vorwärtszähler; (Schaltwerk)



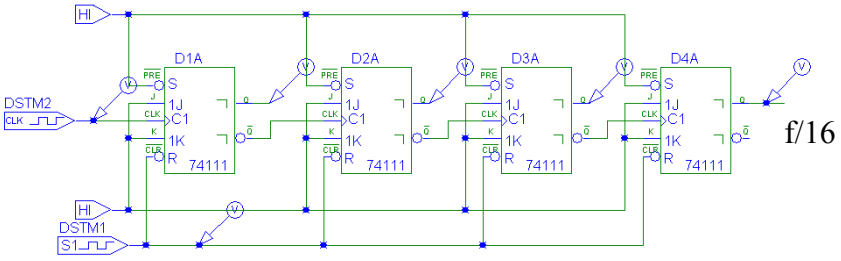
- **Entwurfsbeispiel:** Rein synchroner 4 bit BCD-Vorwärtszähler; (Simulation des Schaltwerks)

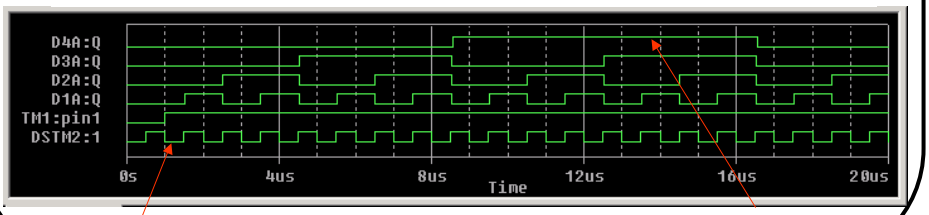


- Anmerkungen zum Zählerentwurf:
- Die Verknüpfungen können stufenübergreifend mit sehr breiten Gattern für jede Stufe ausgeführt werden
- **Nachteil:** Der Verdrahtungsaufwand ist hoch
- **Vorteil:** Diese parallele Lösung ist schneller
- Alternativ können bereits gebildete Terme in Ausdrücken folgender Stufen verwendet werden
- Dadurch kommt es zu einem Propagationseffekt (Ripple), der die mögliche Taktrate für den Zähler reduziert
- Dafür bzgl. Gatteraufwand und Verdrahtung günstiger
- Zähler werden typisch als Bausteine (diskret oder Zellen) einer bestimmten Bitbreite zur Verfügung gestellt (z.B. 74-Familie)
- Größere Zähler können bei Vorliegen entsprechender Signalaus- und eingänge durch Kaskadierung der Bausteine aufgebaut werden

Zähler und Frequenzteiler

- Frequenzteiler entsprechen weitgehend den betrachteten Zählerstrukturen
- Asynchroner 16:1 Frequenzteiler:

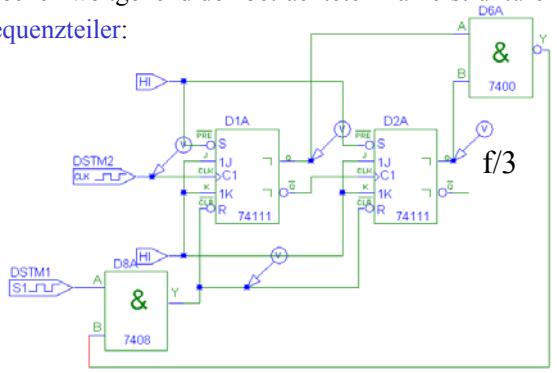


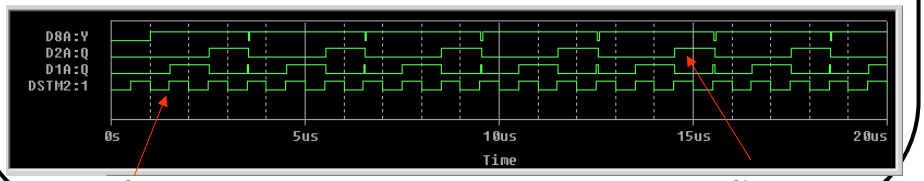


f f/16

Zähler und Frequenzteiler

- Frequenzteiler entsprechen weitgehend den betrachteten Zählerstrukturen
- Asynchroner 3:1 Frequenzteiler:

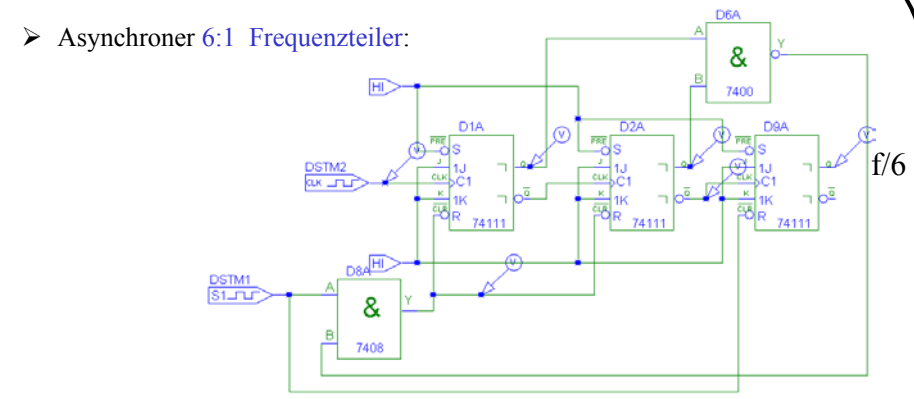


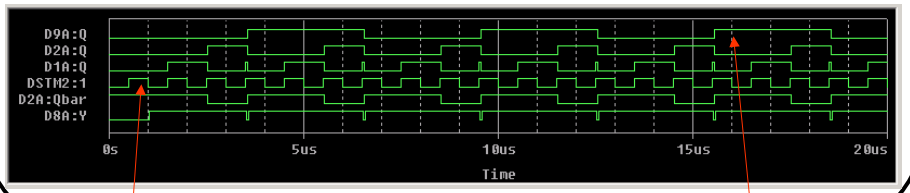


f f/3

Zähler und Frequenzteiler

- Asynchroner 6:1 Frequenzteiler:

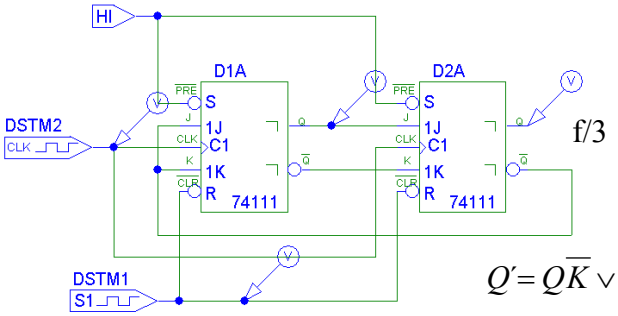


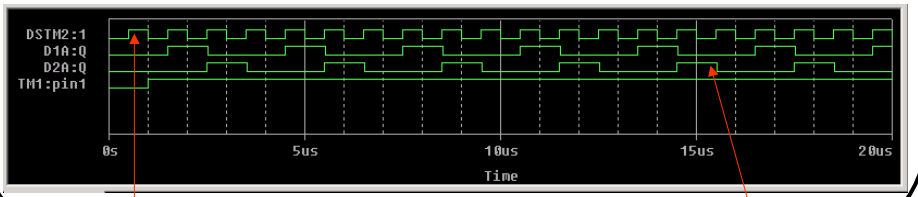


f f/6

Zähler und Frequenzteiler

- Synchroner 3:1 Frequenzteiler:

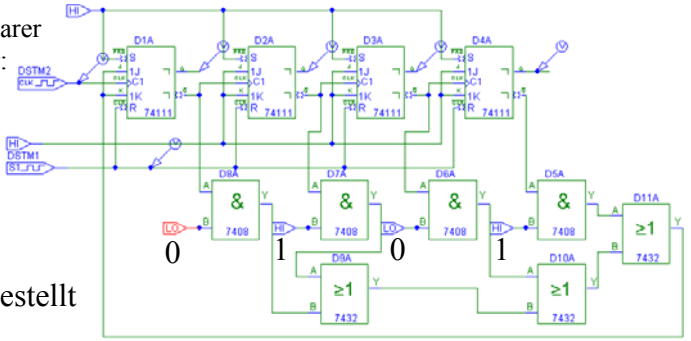




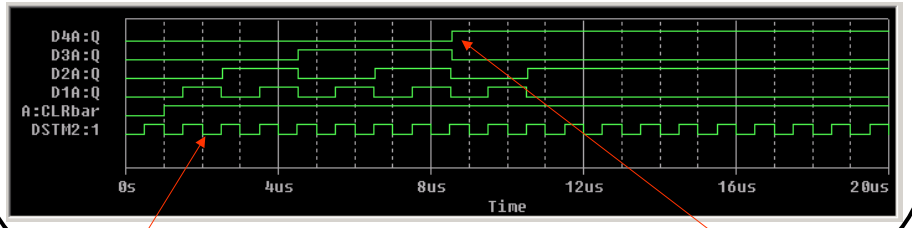
f f/3

Zähler und Frequenzteiler

- Programmierbarer Frequenzteiler:



1010 eingestellt



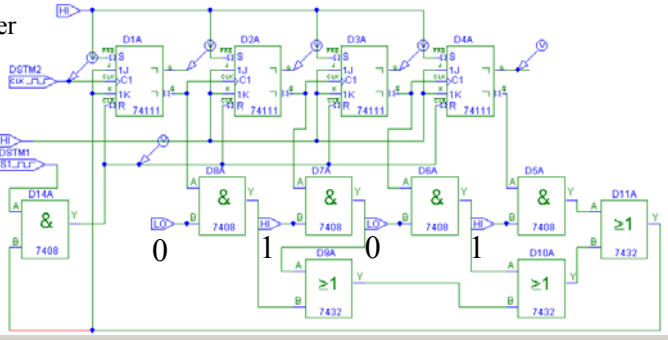
f

f/10

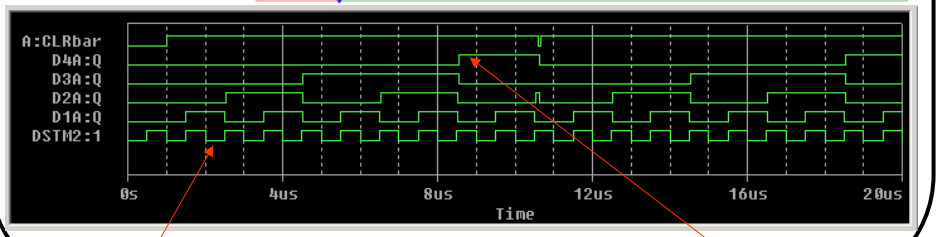
Digitaltechnik
Sequentielle Schaltwerke

Zähler und Frequenzteiler

- Programmierbarer Frequenzteiler:



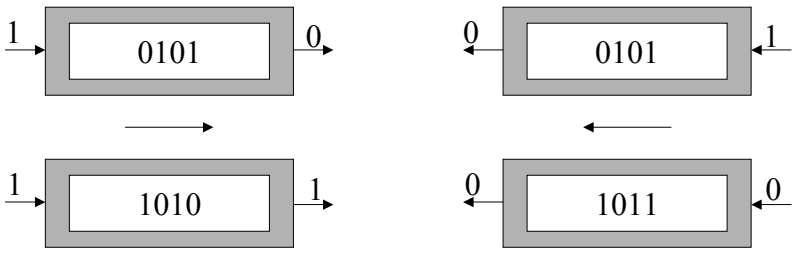
1010 eingestellt



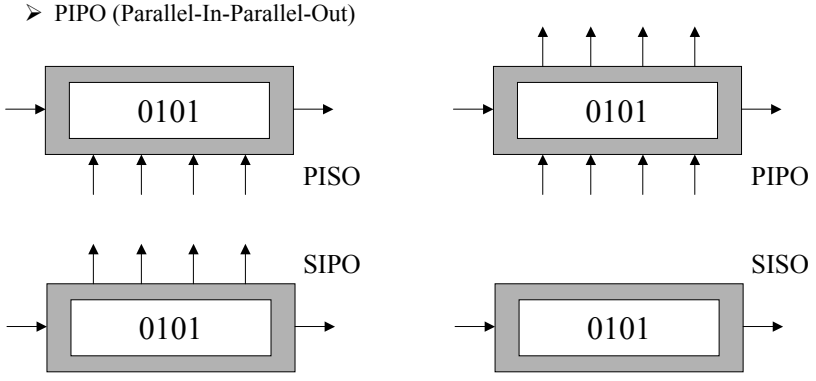
f

f/10

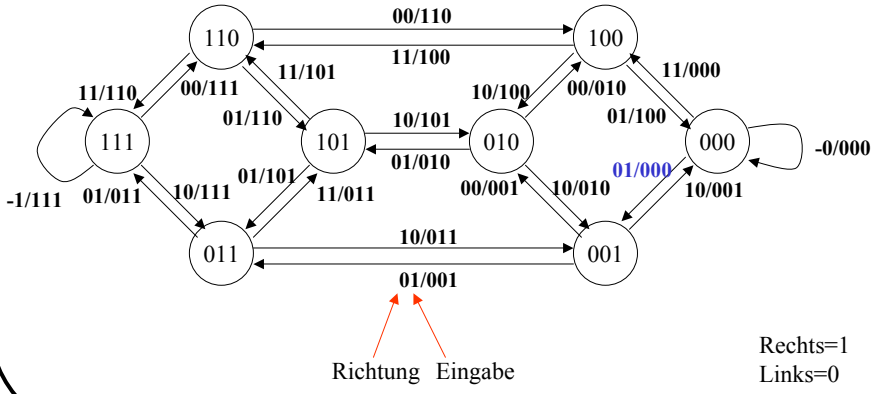
- Schieberegister: Inhalt einer Kette von Flipflops wird nach links bzw. rechts geschoben
- Entspricht der Multiplikation mit 2 bzw. der Division durch 2
- Typisch existiert ein Eingang, aus dem in die freiwerdende Bitstelle von links bzw. rechts eine 0/1 nachgezogen werden
- Zyklische Verbindung von ausgehender und eingehender Bitstelle: Rotation



- Inhalt eines Schieberegisters kann seriell oder parallel geladen werden
- Inhalt eines Schieberegisters kann seriell oder parallel ausgelesen werden
- Entsprechend vier Varianten und Bezeichnungen:
  - PISO (Parallel-In-Serial-Out)
  - SISO (Serial-In-Serial-Out)
  - SIPO (Serial-In-Parallel-Out)
  - PIPO (Parallel-In-Parallel-Out)



- Für ein sequentielles Schieberegister kann nur um eine Position pro Taktzyklus verschoben werden
- Entsprechendes Zustandsdiagramm für ein bidirektionales sequentielles Schieberegister von 3 bit:



- Zustandsübergangstabelle:

	links	rechts
$Q_3Q_2Q_1x$	$Q_3'Q_2'Q_1'$	$Q_3'Q_2'Q_1'$
0 0 0 0	0 0 0	0 0 0
0 0 0 1	0 0 1	1 0 0
0 0 1 0	0 1 0	0 0 0
0 0 1 1	0 1 1	1 0 0
0 1 0 0	1 0 0	0 0 1
0 1 0 1	1 0 1	1 0 1
0 1 1 0	1 1 0	0 0 1
0 1 1 1	1 1 1	1 0 1
1 0 0 0	0 0 0	0 1 0
1 0 0 1	0 0 1	1 1 0
1 0 1 0	0 1 0	0 1 0
1 0 1 1	0 1 1	1 1 0
1 1 0 0	1 0 0	0 1 1
1 1 0 1	1 0 1	1 1 1
1 1 1 0	1 1 0	0 1 1
1 1 1 1	1 1 1	1 1 1

- Folgeschritte:
  - Ansteuertabelle für gewählten FF-Typ
  - Schaltfunktionen bilden und minimieren
  - Umsetzung in Gatternetzwerk + FFs

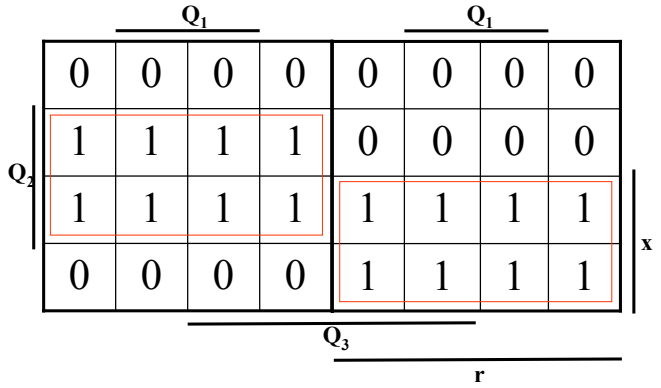
Anmerkung: Links schieben bedeutet verschieben in Richtung des MSB, d.h. *2, rechts schieben hingegen in Richtung des LSB, d.h. /2

	links		rechts	
$Q_3Q_2Q_1x$	$Q_3'Q_2'Q_1'$	$J_3K_3J_2K_2J_1K_1$	$Q_3'Q_2'Q_1'$	$J_3K_3J_2K_2J_1K_1$
0 0 0 0	0 0 0	0 - 0 - 0 -	0 0 0	0 - 0 - 0 -
0 0 0 1	0 0 1	0 - 0 - 1 -	1 0 0	1 – 0 – 0 -
0 0 1 0	0 1 0	0 - 1 - - 1	0 0 0	0 – 0 - - 1
0 0 1 1	0 1 1	0 - 1 - - 0	1 0 0	1 – 0 - - 1
0 1 0 0	1 0 0	1 - - 1 0 -	0 0 1	0 - - 1 1 -
0 1 0 1	1 0 1	1 - - 1 1 -	1 0 1	1 - - 1 1 -
0 1 1 0	1 1 0	1 - - 0 – 1	0 0 1	0 - - 1 – 0
0 1 1 1	1 1 1	1 - - 0 – 0	1 0 1	1 - - 1 – 0
1 0 0 0	0 0 0	- 1 – 0 – 0	0 1 0	- 1 1 – 0
1 0 0 1	0 0 1	- 1 0 - 1 -	1 1 0	- 0 1 – 0 -
1 0 1 0	0 1 0	- 1 1 - - 1	0 1 0	- 1 1 - - 1
1 0 1 1	0 1 1	- 1 1 - - 0	1 1 0	- 0 1 - - 1
1 1 0 0	1 0 0	- 0 – 1 0 -	0 1 1	- 1 – 0 1 -
1 1 0 1	1 0 1	- 0 – 1 1 -	1 1 1	- 0 – 0 1 -
1 1 1 0	1 1 0	- 0 – 0 – 1	0 1 1	- 1 – 0 – 0
1 1 1 1	1 1 1	- 0 – 0 - 0	1 1 1	- 0 – 0 - 0

	links (R=0)		rechts (R=1)	
$Q_3Q_2Q_1x$	$Q_3'Q_2'Q_1'$	$D_3D_2D_1$	$Q_3'Q_2'Q_1'$	$D_3D_2D_1$
0 0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 0 0 1	0 0 1	0 0 1	1 0 0	1 0 0
0 0 1 0	0 1 0	0 1 0	0 0 0	0 0 0
0 0 1 1	0 1 1	0 1 1	1 0 0	1 0 0
0 1 0 0	1 0 0	1 0 0	0 0 1	0 0 1
0 1 0 1	1 0 1	1 0 1	1 0 1	1 0 1
0 1 1 0	1 1 0	1 1 0	0 0 1	0 0 1
0 1 1 1	1 1 1	1 1 1	1 0 1	1 0 1
1 0 0 0	0 0 0	0 0 0	0 1 0	0 1 0
1 0 0 1	0 0 1	0 0 1	1 1 0	1 1 0
1 0 1 0	0 1 0	0 1 0	0 1 0	0 1 0
1 0 1 1	0 1 1	0 1 1	1 1 0	1 1 0
1 1 0 0	1 0 0	1 0 0	0 1 1	0 1 1
1 1 0 1	1 0 1	1 0 1	1 1 1	1 1 1
1 1 1 0	1 1 0	1 1 0	0 1 1	0 1 1
1 1 1 1	1 1 1	1 1 1	1 1 1	1 1 1

- Schritt Logikminimierung, hier für D_3:

D_3



$$D_3 = \bar{r}Q_2 \vee rx$$

- Beschaltung der D-FFs zum bidirektionalen 3 bit Schieberegister:

$$D_3 = \bar{r}Q_2 \vee rx$$

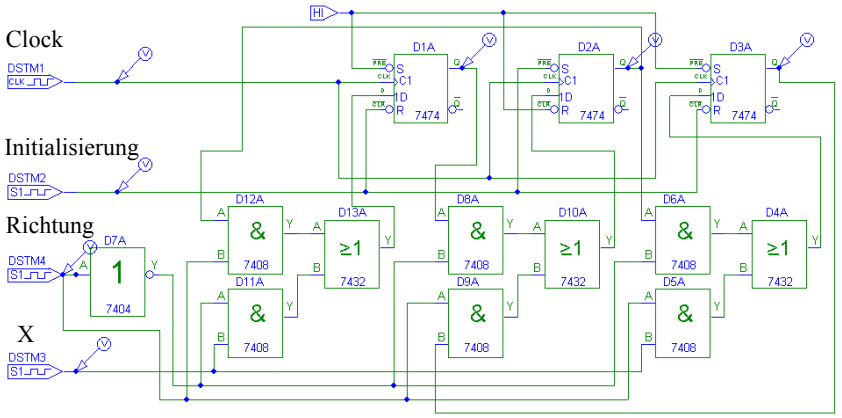
$$D_2 = \bar{r}Q_1 \vee rQ_3$$

$$D_1 = \bar{r}x \vee rQ_2$$

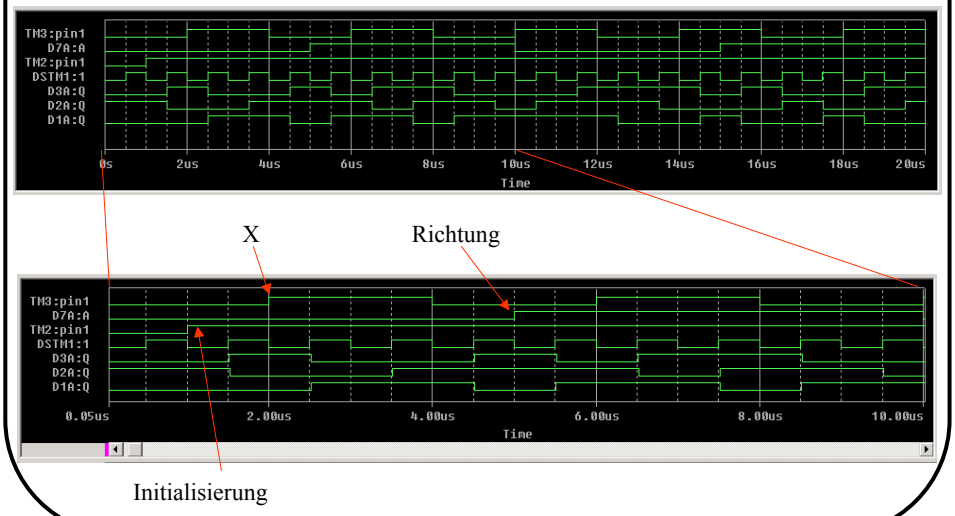
- Erkennbar: Für die internen Stufen kann eine Kaskadierung auf ein n bit SR durch die Verallgemeinerung der Gleichung für D_2 erfolgen:

$$D_i = \bar{r}Q_{i-1} \vee rQ_{i+1}$$

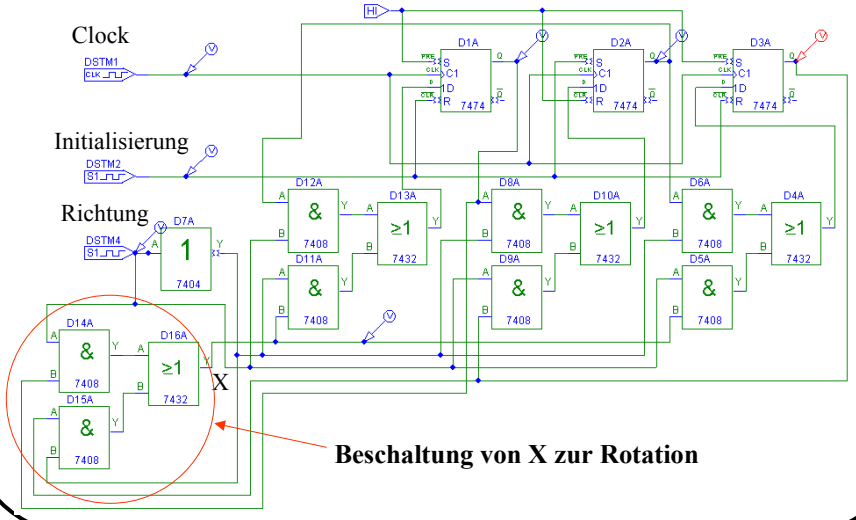
- Schaltwerk zum bidirektionalen 3 bit Schieberegister mit D-FFs:



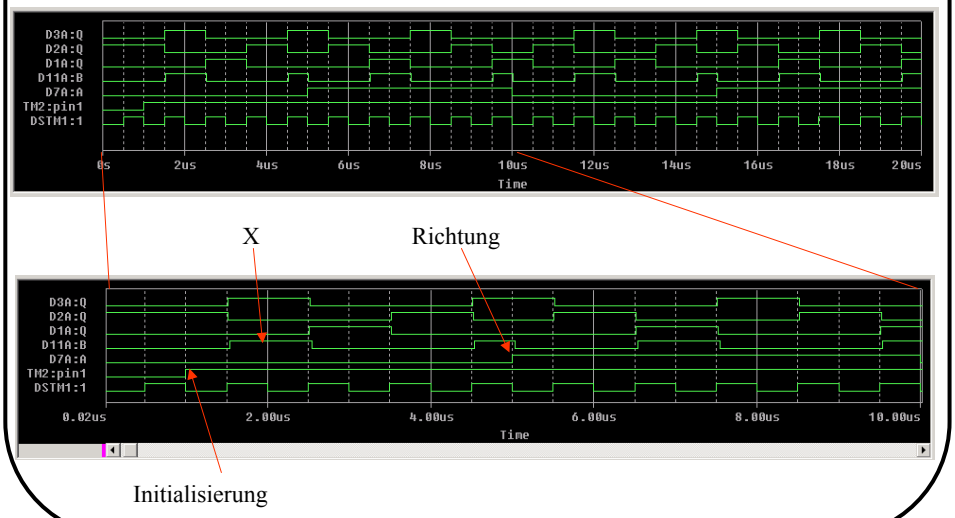
- Simulation des bidirektionalen 3 bit Schieberegister mit D-FFs:



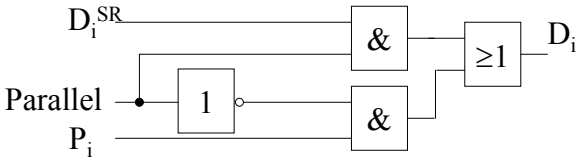
- Schaltwerk zum bidirektionalen 3 bit Schieberegister mit D-FFs:



- Simulation des bidirektionalen 3 bit Schieberegister mit D-FFs:

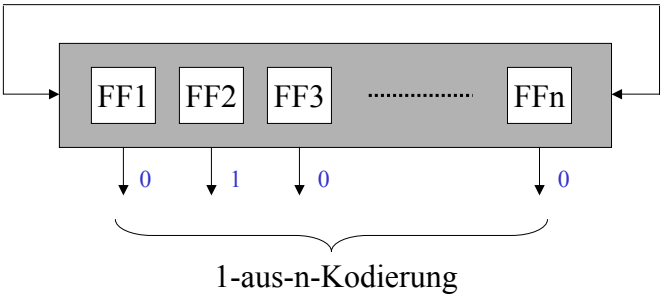


- Erkennbar wird das bidirektionale 3 bit Schieberegister mit D-FFs bislang nur asynchron auf einen Anfangswert eingestellt
- Prinzipiell kann jeder gewünschte Anfangswert so eingestellt werden
- Seriell kann das SR synchron über den X-Eingang und n Schiebeoperationen (n entspricht der SR-Breite) gesetzt werden
- Für Nutzung des SR in synchronem System der Digitaltechnik ist häufig auch ein paralleles Setzen erforderlich (Parallel-Seriell-Konversion für Ifc.)
- Erweiterung der Beschaltung der D-Eingänge erlaubt synchrones paralleles Setzen oder Beschreiben des SR:



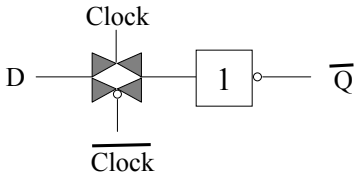
- Ergänzende Beschaltung einer SR-Stufe zum parallelen Laden

- **Spezialfall:** Wie im Entwurfsbeispiel wird nur eine 1 in das SR geladen
- Diese wird taktsynchron rotiert und kombiniert Funktion eines Zählers und Decoders:

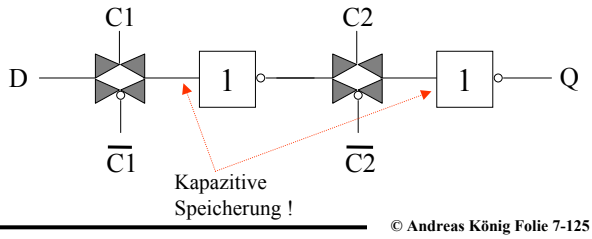


- **Anwendungsbeispiel:** Zyklische Adressgenerierung zur Auslesung einer Bildsensorzeile oder –matrix
- Sonderfall n=10, Dekadischer Zähler; allerdings ungünstiger FF-Verbrauch
- Johnson-Zähler (oder ´´Twisted´´-Ringzähler) kommt mit 5 FF aus [Seiffart 98]

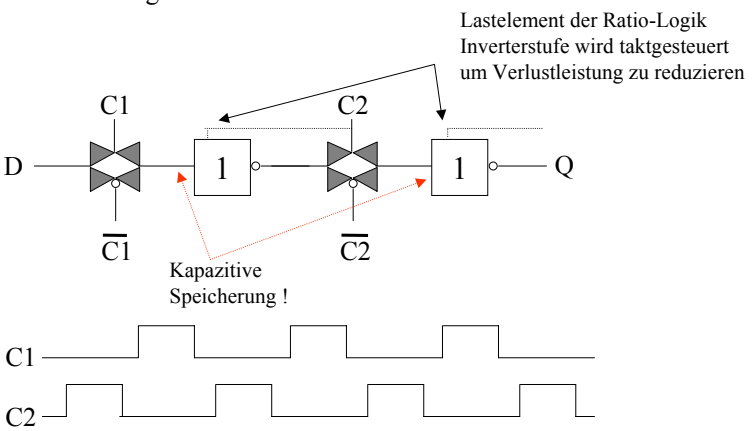
- Dynamische Strukturen zur Realisierung von Zählern, Frequenzteilern und Schieberegistern



- **Erinnerung:** Speicherzeit begrenzt durch Entladeeffekte (Leckströme)
- Dynamische Elemente sind platzsparend, stellen aber höhere Anforderungen an den Entwerfer und sind ggf. langsamer



- Gepufferte dynamische Speicherstufe mit nichtüberlappendem Zweiphasentakt zur Realisierung von Zählern etc.



- **Erkennbar:** Nur noch acht Transistoren für gepufferte FF-Stufe !

Zähler und Frequenzteiler

- Gepufferte dynamische Speicherstufe bringt Vorteile hinsichtlich der Schaltungskompaktheit und der Verlustleistung
- Jedoch kommt zur oberen Grenze der Taktrate (Verzögerungszeiten) eine untere Grenze der Taktrate oder Zugriffsrate durch die Auffrischbedingung bzw. begrenzte Speicherzeit der dynamischen Zelle hinzu
- Generierung und Verbreitung, insbesondere Verdrahtung, des nicht-überlappenden Zweiphasentakts nicht unkritisch
- Dynamische Lösungen (Logik und Speicherung) vor allem für höchst-integrierte Schaltkreis- und Systemlösungen interessant
- Weitere Verlustleistungsersparnis durch nichtüberlappenden Vierphasentakt
- Es wird durch die Taktsteuerung vermieden, dass im statische Fall Querstrom fließt
- Verlustleistung nur beim Umladen der Kapazitäten
- Verwendung von Minimaltransistoren nun möglich (nicht bei Ratio-Logik) !

Entwurf von Automaten

- Ein Elektrounternehmen im Dresdner Süden dekoriert alljährlich ein Fenster seines Werkzeugschuppens weihnachtlich:

- Ein Außentaster erlaubt Besuchern das weihnachtliche Bild zum Leben zu erwecken
- Bei Tastendruck wird die Beleuchtung eingeschaltet, elektrische Kerzen eines Adventskranzes leuchten passend zur Adventswoche auf und ein lebensgroßer Weihnachtsmann wedelt ca. 4 s mit seiner Rute
- Dann wird eine Spieluhr eingeschaltet und eine Modelleisenbahn fängt für ca. 20s an Runden zu fahren während ein Sägewerk einer Dampf-maschine seine Arbeit aufnimmt
- Nach dem Anhalten der Eisenbahn beginnt sich ein altes Blechspielzeugkarussell zu drehen
- Nach insgesamt ca. 1 Minute kehrt wieder Frieden und Dunkelheit ein

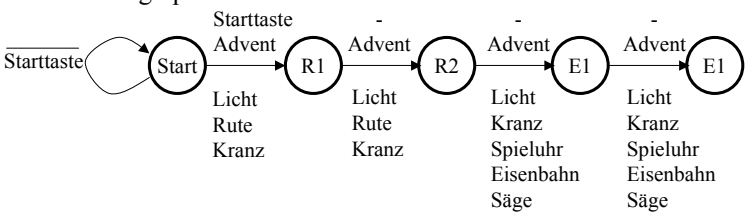
Entwurf von Automaten

- Wie kann man diese umgangssprachliche Beschreibung der Szene in einen Automaten umsetzen, der unter einigen vereinfachenden *Randbedingungen* die Steuerung übernimmt ?
- Es liegt ein Systemtakt von 2s Periodendauer vor, der als Zeitbasis und zum Zustandswechsel dienen kann
- Zur Ansteuerung der elektrischen Motoren und Lampen genüge ein einfaches Digitalsignal
- Die aktuelle Angabe zum Advent liege binärcodiert vor
- Der Taster sei entprellt und taktsynchronisiert

- Zunächst muss das erforderliche Eingangs, Ausgangs und Zustandsalphabet aus Überlegung bestimmt werden und daraus der Automatengraph gewonnen werden

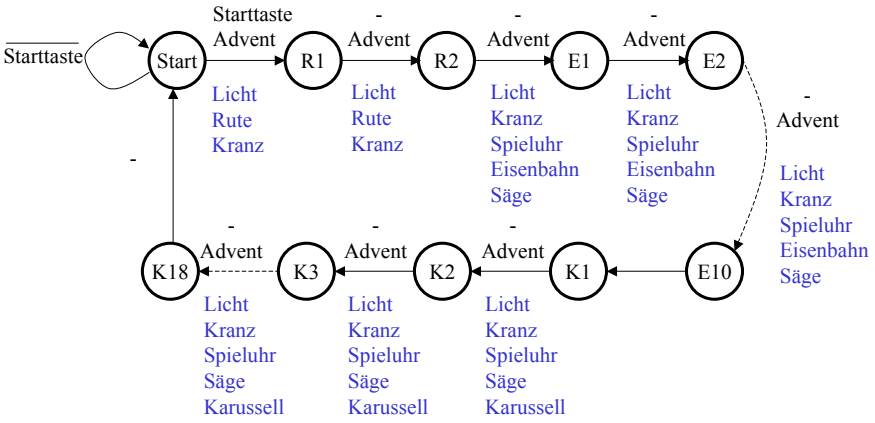
Entwurf von Automaten

- Eingänge: Starttaste; Advent
- Ausgänge: Licht, Rute, Kranz, Spieluhr, Eisenbahn, Säge, Karussell
- Zustände: Start, ... ?
  - Der Weihnachtsmann soll 4s mit der Rute wedeln; Zeitbasis 2s, daher 2 Zustände
  - Eisenbahn soll dann 20s fahren; also 10 Zustände erforderlich
  - Weitere Aktivitäten für 60s-24s bis zur Abschaltung; also 18 Zustände
- Automatengraph:



Fortsetzung nächste Folie

- Automatengraph:



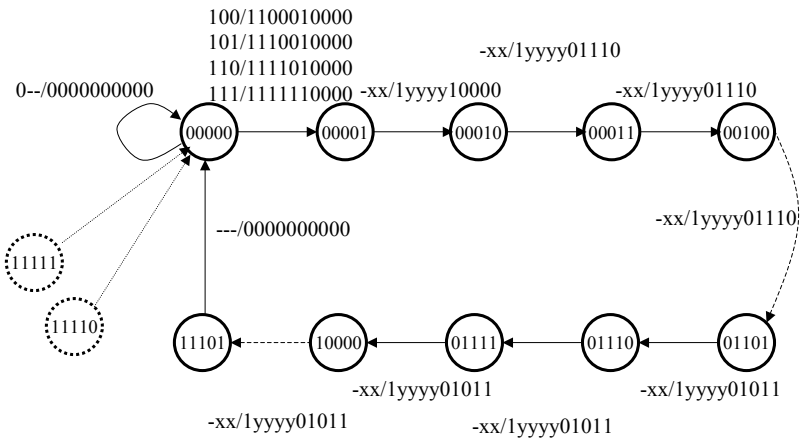
- Kodierungsüberlegungen:
  - Es treten 30 Zustände auf; daher werden 5 bit zur Zustandsdarstellung benötigt
  - Der Taster erfordert ein Eingangsbit; Die Kodierung des Advents zwei weitere
  - Die sieben anzusteuernden Ausgänge erfordern 10 bit
- Welcher Automatentyp wird durch den Automatengraphen impliziert ?
- **Nächster Schritt:** Kodierung und Darstellung als Zustandsdiagramm

Starttaste
Advent1
Advent2

Licht
Kranz1
Kranz2
Kranz3
Kranz4
Rute
Spieluhr
Eisenbahn
Säge
Karussell

Entwurf von Automaten

- Zustandsdiagramm:



Entwurf von Automaten

- Aufstellung der Zustandsübergangstabelle
- FF-Auswahl
- Logikminimierung
- Schaltnetz- bzw. Schaltwerksentwurf, z.B. 74-Familie
- Simulation

- Minimierungsmöglichkeiten für die Zahl der verwendeten Zustände ?

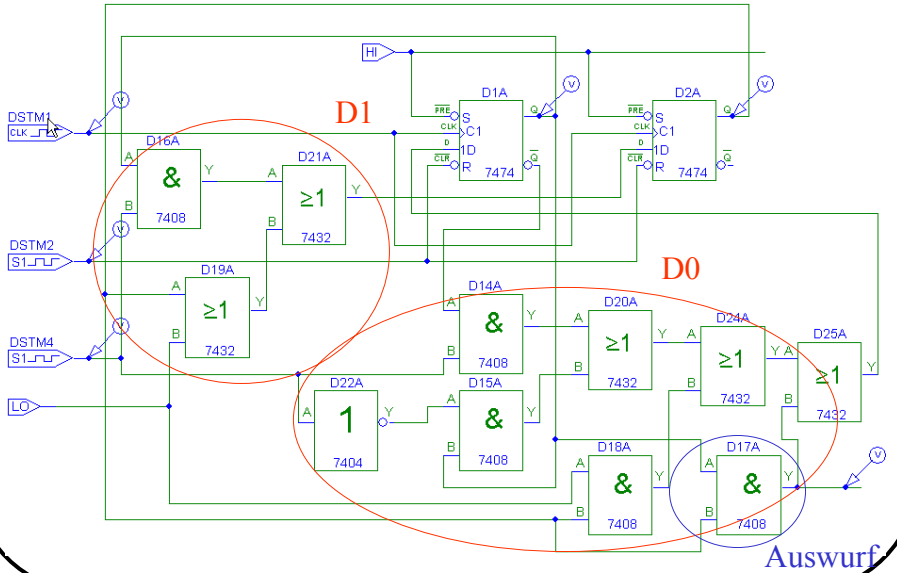
Fröhliche Weihnachten und ein gutes neues Jahr !

Nächste Vorlesung: 03.01.2002

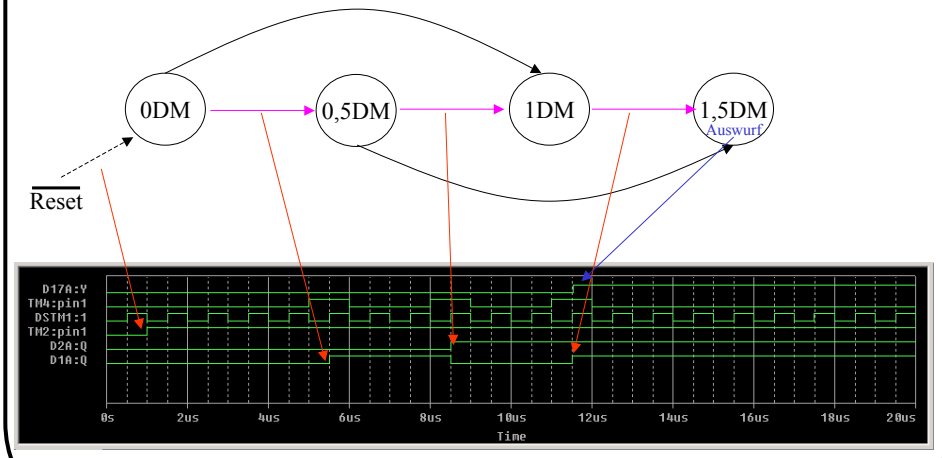
- Detaillierterer Entwurfsablauf für endliche Automaten (FSM):
  - Erarbeitung des Verständnisses der Aufgabe aus gegebener umgangs-sprachlicher Beschreibung; Auflösung aller Doppeldeutigkeiten
  - Erstellung einer abstrakten Repräsentation der FSM, die eine effiziente Implementierung mit den verfügbaren Methoden erlaubt (z.B. Zustandsdiagramm)
  - *Durchführung einer Zustandsminimierung (soweit möglich)*
  - Zustandskodierung und Aufstellung der Zustandsübergangstabelle
  - FF-Auswahl (JK-FF;D-FF;....) wie bei Beispiel Schieberegister und Aufstellen der Ansteuerfunktionen
  - FSM-Implementierung:
    - Implementierungsauswahl (Random- oder Array-Logik, zwei- oder mehrstufig)
    - Logikminimierung für Überführungs- und Ausgabefunktion (soweit sinnvoll) anhand der Ansteuerfunktionen
    - Schaltnetz- bzw. Schaltwerksentwurf, z.B. Gatternetze und FFs aus 74-Familie
    - *Simulation und Validierung*

- Entwurfsbeispiel Verkaufsautomat:
  - Für einen Automaten zum Verkauf von Aspirintabletten soll die zugehörige FSM entworfen werden
  - Der Automat, der noch nicht auf EURO umgestellt wurde, liefert nach Einwurf von DM 1,50 eine Tablette im Ausgabeschacht
  - Angenommen werden nur Geldstücke von einer Mark bzw. fünfzig Pfennigen (bestehender Detektionsblock zur Münzerkennung !)
  - Es wird kein Wechselgeld gegeben, d.h. beim Einwurf von zwei Markstücken verliert der Kunde fünfzig Pfennig
  - Ein Abbruch oder eine Geldrückgabe ist zunächst auch nicht vorgesehen
  - Nach Abgabe der Tablette kann der Automat über die *Reset*-Taste wieder in Grundzustand gebracht werden

- Schaltnetz zum Entwurfsbeispiel Verkaufsautomat:



- Simulation des Schaltnetzes zur FSM Verkaufsautomat:



- Bislang wurde die Beschreibung von FSMs auf der Basis von Beschreibungsformen wie z.B. Zustandsdiagrammen oder –tabellen vorgenommen
- Die Beschreibung komplexer FSMs wird mit diesen Hilfsmitteln zunehmend schwieriger und unübersichtlicher
- So werden z.B. nicht alle Eingangsvariablen in jedem Zustand zur Darstellung der bezweckten Zustandsübergänge und Ausgaben benötigt
- Zustandsdiagramme stellen algorithmische Zusammenhänge ungenügend dar
- Alternative Beschreibungsformen für FSMs:
  - *Algorithmic State Machines* [Katz 94] bzw. Ablaufdiagramme und –tabellen [Lipp 99]
  - Hardware-Description Languages:
    - Abel
    - Verilog HDL
    - VHDL (VHSIC HDL, IEEE Standard)
    - SystemC
  - *Behandlung in Kapitel 9 der Vorlesung*

Entwurf von Automaten

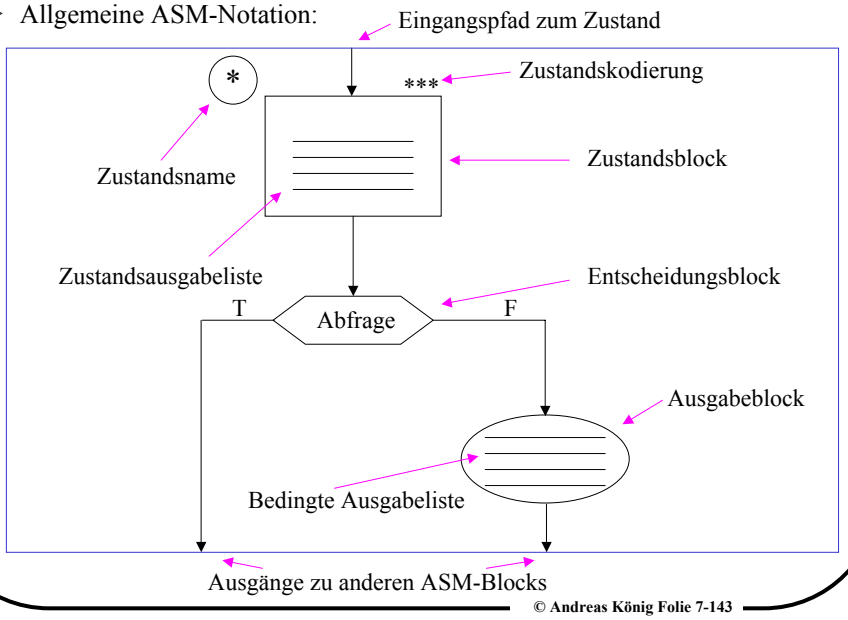
- *Algorithmic State Machines* (ASM) stellen das FSM-Verhalten in einer den Flußdiagrammen vergleichbaren Weise dar
- Jedoch ist mit der ASM-Darstellung ein striktes Timing verbunden
- ASMs sind strukturiert in ASM-Blöcke die wiederum drei Grundbausteine besitzen:
  - Zustandsblock (State-Box)
  - Entscheidungsblock (Condition-Box)
  - Ausgabeblock (Output-Box)
- Pro ASM-Block existiert genau ein Zustandsblock, der über genau einen eingehenden Pfad von anderen ASM-Blocks erreicht werden kann.
- Der ASM-Block wird für die relevanten Kombinationen der Eingangsvariablen über einen eindeutig zugeordneten Pfad verlassen
- Der Zustandsblock ist durch den symbolischen Namen und die gewählte Kodierung gekennzeichnet
- Jeder Zustandsblock enthält eine Ausgabeliste mit den Signalen, die im betrachteten Zustand aktiv sind

Entwurf von Automaten

- Da die digitalen Signale sowohl high-aktiv als auch low-aktiv sein können (pos. Und neg. Logik !) erfolgt üblicherweise bei ASMs eine Präfixkennzeichnung durch H. bzw. L. vor den aktiven Ausgangssignalen
- Mittels eines oder mehrerer Entscheidungsblöcke können Eingangsvariablen zur Bestimmung des Folgezustands abgefragt werden
- Aus diesen Blöcken bestimmen sich die Pfade zu anderen ASM-Blöcken bzw. auf den gegenwärtigen Block selbst bei Verharrung im aktuellen Zustand
- Die bislang eingeführte Ausgabeliste im Zustandsblock erlaubt die Realisierung eines Moore-Automaten
- Ist eine Mealy-Automaten-Realisierung erwünscht, so erlaubt der Ausgabeblock die Beschreibung von Ausgaben die direkt von den Eingangsvariablen abhängig sind
- Die einzelnen Ausgabeblöcke werden mit ihren bedingten Ausgabelisten hinter die Ausgänge der Entscheidungsblöcke gesetzt
- Für die Notation in den Ausgabelisten gilt geg. Konvention (s.o.)

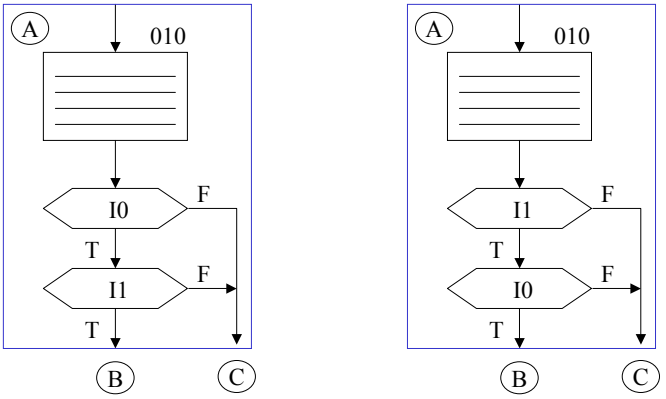
Entwurf von Automaten

- Allgemeine ASM-Notation:



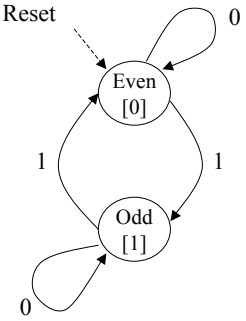
Entwurf von Automaten

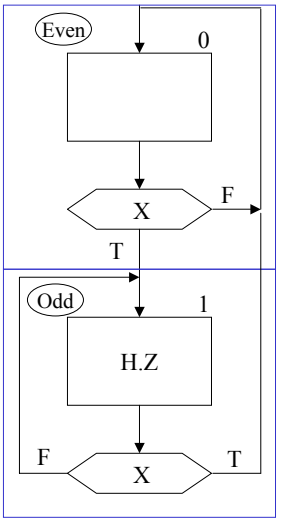
- Die Reihenfolge der Entscheidungsblöcke hat keine Effekt auf den entsprechenden Zustandsübergang bzw. die Ausgabe:



- Die beiden ASM-Blöcke habe äquivalentes Verhalten

- ASM Anwendungsbeispiel: Einfacher *Odd/Even-Parity Checker*
- Aufgabe: Schritthaltende Überprüfung einer bit-Sequenz auf gerade (0) bzw. ungerade (1) Parität

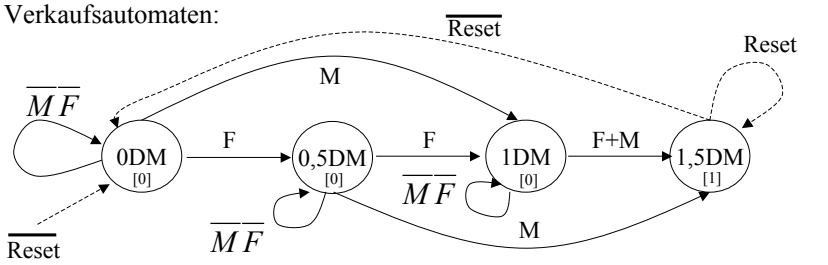


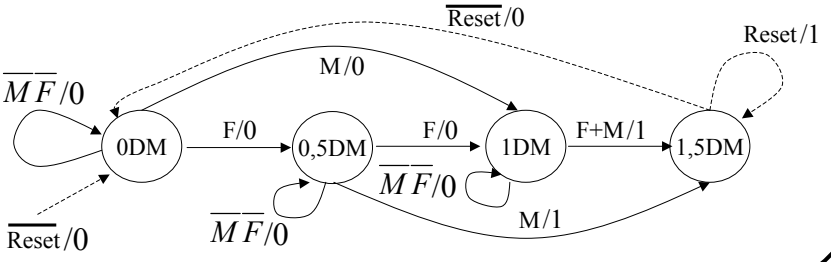


- Anmerkungen zum Entwurf mit Mealy- und Moore-Automaten:
  - Moore-Realisierungen von FSM weisen Ausgänge auf, die sich nur synchron mit dem Takt ändern
  - Sie sind daher sicherer im Hinblick auf Glitches im Schaltnetz der Ausgangs-funktion sowie der Eingangsvariablen
  - Daher wurde für den Verkaufsautomaten (Bsp. 7.2) eine Moore-Realisierung gewählt
  - Mealy-Realisierungen ändern Ihre Ausgänge asynchron mit den Eingangs-variablen
  - Hierdurch kann ein Fehlverhalten angeschlossener Komponenten (Auswurfschacht des Verkaufsautomaten auftreten)
  - Jedoch: Die Mealy-Realisierung einer FSM kann aufwandsgünstiger sein als die entsprechende Moore-Realisierung
  - Kompromiss beider Varianten: Synchroner Mealy-Automat

Entwurf von Automaten

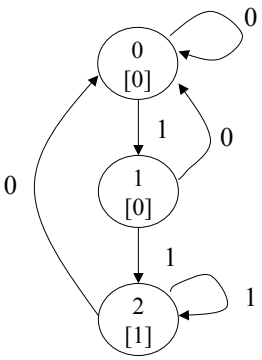
- Betrachtung der Zustandsdiagramme zu Moore- und Mealy-Realisierung des Verkaufsautomaten:



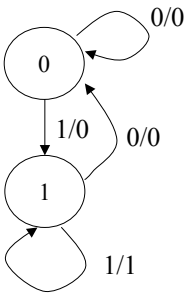


Entwurf von Automaten

- Aufwandsvergleich anhand einer einfachen FSM, die ihren Ausgang aktiviert, wenn in der beobachteten Eingangssequenz zwei Einsen direkt aufeinander folgen



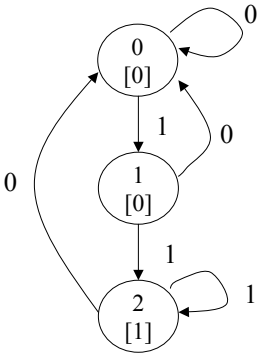
Moore-Automat



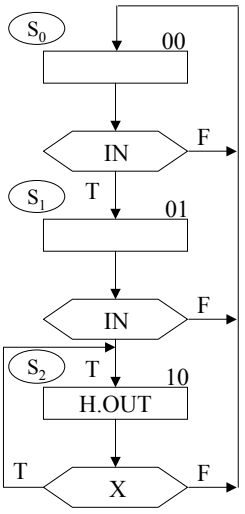
Mealy-Automat

Entwurf von Automaten

- Zustandsdiagramm und alternative ASM-Beschreibung:

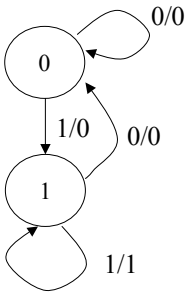


Moore-Automat

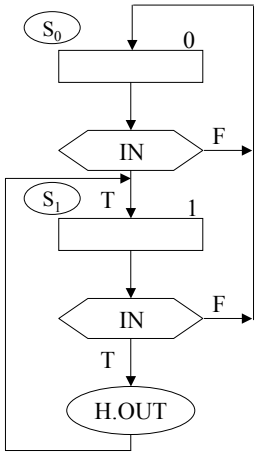


Entwurf von Automaten

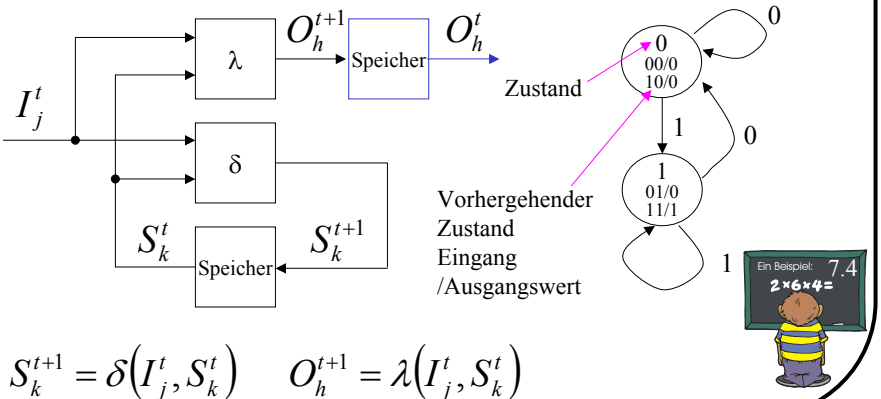
- Zustandsdiagramm und alternative ASM-Beschreibung:



Mealy-Automat



- Erkennbar lässt sich die Aufgabe durch Verwendung eines Mealy-Automaten mit weniger Zuständen und damit geringerem Aufwand lösen
- Als Nachteil bleibt jedoch die asynchrone Wirkung auf folgende (Aktor)stufen
- **Abhilfe:** Synchroner Mealy-Automat durch FF-Pufferung der Ausgänge

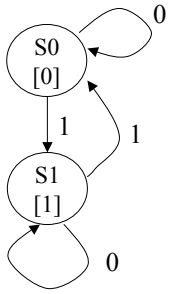


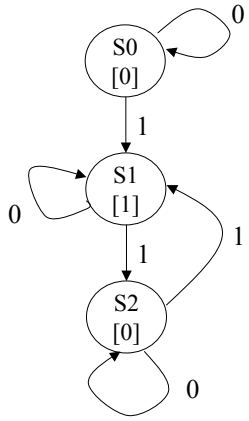
$$S_k^{t+1} = \delta\left(I_j^t, S_k^t\right) \qquad O_h^{t+1} = \lambda\left(I_j^t, S_k^t\right)$$

- Die bisherige Herangehensweise führt zu einer funktionsfähigen FSM-Implementierung
- Einzige Optimierung im bisherigen Ansatz: Minimierung der Schaltnetze für Ausgangs- und Überführungsfunktion
- Weiteres, bislang unausgeschöpftes Optimierungspotenzial durch:
  - Zustandsminimierung
  - Zustandskodierung
  - FSM-Partitionierung
- Im folgenden prinzipielle Betrachtung dieser Aspekte als Erweiterung des bisherigen Entwurfsablaufs
- Manuelle *Papier-und-Bleistift* Methoden
- **Ausblick:** Rechnergestützte Optimierung und entsprechende Werkzeuge

Zustandsminimierung

- Eine Automaten-Aufgabe kann mit sehr unterschiedlichen FSMs und damit verbundenem Aufwand gelöst werden
- Beispiel eines Parity-Checkers mit zwei äquivalenten FSMs:





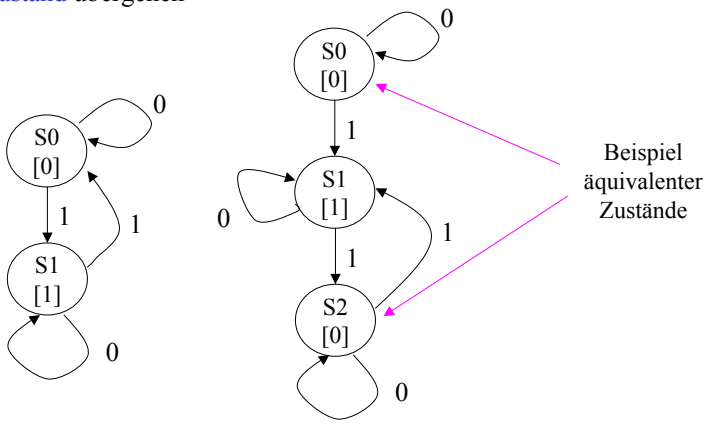
- Die rechte Lösung benötigt mehr FFs und komplexere Schaltnetze

Zustandsminimierung

- **Motivation der Minimierung** der Zustände in abstrakten FSMs
  - Eine Reduktion der Zustandszahl führt potenziell zu einer Verringerung der Anzahl benötigter FF in der Automaten-Implementierung
  - Selbst wenn die Zustandsreduktion gerade noch keine FF-Einsparung bewirkt, werden die Schaltnetze der Überführungs- und Ausgangsfunktion einfacher und schneller
  - Eine Minimierung hinsichtlich Gatteräquivalenten und FFs kann zur Zuschneidung der FSM für einen gegebenen Baustein dienen (Minimierung erforderlicher Bausteinzahl einer Implementierung)
  - Systematische Reduktionsmethoden für abstrakte FSMs erleichtern die Vorgehensweise bei der initialen Erstellung
  - Hier kann das Augenmerk auf eine konsistente, vollständige und widerspruchsfreie Umsetzung gelegt werden
  - Die Optimierung erfolgt nach Aufstellung der abstrakten FSM

Zustandsminimierung

- Eine Zustandsminimierung basiert auf der Auffindung und Verschmelzung äquivalenter Zustände
- Zwei Zustände sind äquivalent, wenn Sie für alle Kombinationen der Eingangsvariablen identische Ausgangswerte aufweisen und in den gleichen Folgezustand übergehen



Zustandsminimierung

- Prinzipielle Vorgehensweise von Algorithmen zur Zustandsreduktion:
  - Zusammenstellung von Zuständen, die für Moore-Automaten identische Zustandsausgaben bzw. für Mealy-Automaten identische Übergangs-ausgaben aufweisen
  - Nur derartige Zustände können potenzielle äquivalent sein
  - Im folgenden werden für alle Eingangskombination die Folgezustände betrachtet
  - Sind auch diese identisch, so sind die Zustände äquivalent und können zu einem Zustand neuen Namens zusammen gefasst werden
  - Die zusammengefassten Zustände werden damit entfernt und alle Übergänge in die entfernten Zustände werden auf den neu eingeführten Zustand gesetzt
  - Der Vorgang wird iterativ wiederholt, bis keine weitere Zusammenfassung mehr möglich ist
- Zwei mögliche Methoden werden im folgenden betrachtet:
  - Row-Matching-Methode (Manuelle, suboptimale Methode)
  - Implication-Charts (Komplexe Methode, optimale Lösungen)

Zustandsminimierung

- Die **Row-Matching-Methode**
- Erläuterung anhand eines einfachen Sequenzdetektors
- Der Detektor wertet aus einem bit-Strom 4 aufeinanderfolgende bits aus und setzt seinen Ausgang aktiv, wenn mit dem letzten bit entweder die Sequenz 0110 oder die Sequenz 1010 detektiert wurden
- Der Detektor kehrt nach der 4-bit-Sequenz wieder in den Anfangszustand zurück
- Verhaltensbeispiel des Detektors:

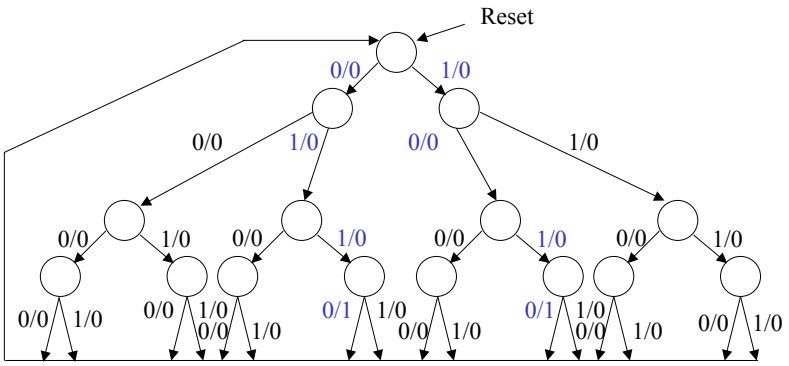
  X = 0010 0110 1100 1010 0011 ....

  Z = 0000 0001 0000 0001 0000 ....

- Die einzelnen Muster überlappen sich nicht
- Die Zahl erforderlicher Zustände ist durch die feste Sequenzlänge nach oben begrenzt
- Aufstellung eines Zustandsdiagramms für das Beispiel

Zustandsminimierung

- Zustandsdiagramm des 4-bit-Sequenzdetektors:



- **15** Zustände und 30 Übergänge, nur zwei Zustände haben aktiven Ausgang
- Intuitiv können viele Zustände zusammengefasst werden
- Systematische Herangehensweise:

Zustandsminimierung

- Aufstellung modifizierter Zustandsübergangstabelle:

		Folgezustand		Ausgang	
Eingangssequenz	**Zustand**	**X=0**	**X=1**	**X=0**	**X=1**
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S_{10}	0	0
10	S_5	S_{11}	S_{12}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011	S_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
101	S_{12}	S_0	S_0	1	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

Zustandsminimierung

- Zusammenfassung von S_{10} und S_{12} zu Zustand S_{10}'

		Folgezustand		Ausgang	
Eingangssequenz	**Zustand**	**X=0**	**X=1**	**X=0**	**X=1**
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S_{10}'	0	0
10	S_5	S_{11}	S_{10}'	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011 oder 101	S_{10}'	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

Zustandsminimierung

- Zusammenfassung von S_7 bis S_9 und S_{11} bis S_{14} zu Zustand S_7'

		Folgezustand		Ausgang	
Eingangssequenz	**Zustand**	**X=0**	**X=1**	**X=0**	**X=1**
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7'	S_7'	0	0
01	S_4	S_7'	S_{10}'	0	0
10	S_5	S_7'	S_{10}'	0	0
11	S_6	S_7'	S_7'	0	0
nicht (011 oder 101)	S_7'	S_0	S_0	0	0
011 oder 101	S_{10}'	S_0	S_0	1	0

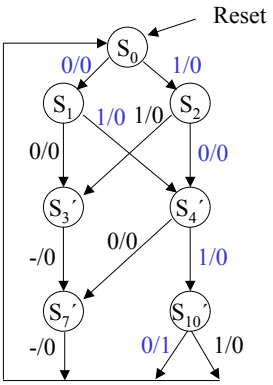
Zustandsminimierung

- Zusammenfassung von S_3 und S_6 zu S_3' sowie S_4 und S_5 zu Zustand S_4'

		Folgezustand		Ausgang	
Eingangssequenz	**Zustand**	**X=0**	**X=1**	**X=0**	**X=1**
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3'	S_4'	0	0
1	S_2	S_4'	S_3'	0	0
00 oder 11	S_3'	S_7'	S_7'	0	0
01 oder 10	S_4'	S_7'	S_{10}'	0	0
nicht (011 oder 101)	S_7'	S_0	S_0	0	0
011 oder 101	S_{10}'	S_0	S_0	1	0

Zustandsminimierung

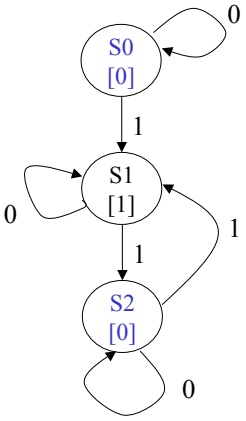
- Aufstellung des minimierten Zustandsdiagramms:



- Reduktion von 15 auf 7 Zustände (3 statt 4 FF)

Zustandsminimierung

- **Row-Matching-Methode** findet nicht alle Optimierungsmöglichkeiten:



	Folgezustand		
Zustand	**X=0**	**X=1**	**Ausgang**
S_0	S_0	S_1	0
S_1	S_1	S_2	1
S_2	S_2	S_1	0

- Die Verzweigung auf sich selbst verhindert die Anwendung der Row-Matching-Methode

Zustandsminimierung

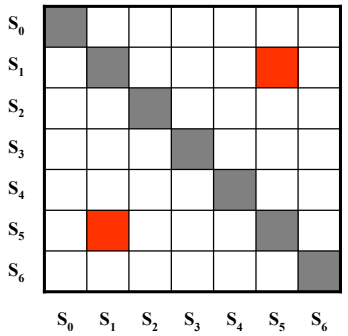
- Die Implication-Chart-Methode
- Diese Methode ist leistungsfähiger aber auch komplexer als Row-Matching
- Daher wird zunächst ein vereinfachter 3-bit-Sequenzdetektor betrachtet:

		Folgezustand		Ausgang	
Eingangssequenz	**Zustand**	**X=0**	**X=1**	**X=0**	**X=1**
Reset	S_0	S_1	S_2	**0**	**0**
0	S_1	S_3	S_4	**0**	**0**
1	S_2	S_5	S_6	**0**	**0**
00	S_3	S_0	S_0	**0**	**0**
01	S_4	S_0	S_0	**1**	**0**
10	S_5	S_0	S_0	**0**	**0**
11	S_6	S_0	S_0	**1**	**0**

- Der Detektor setzt den Ausgang für 010 und 110 aktiv

Zustandsminimierung

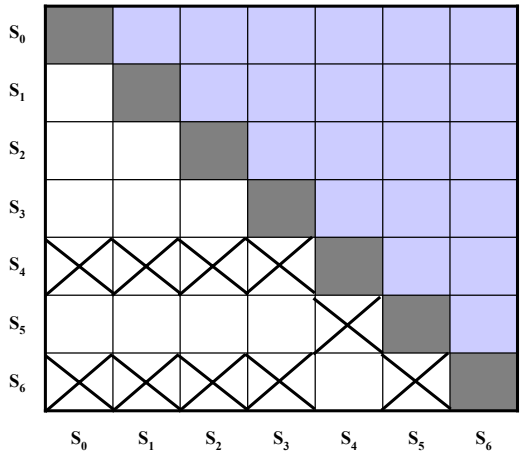
- Zunächst kann eine Matrix aller Zustandskombination als Ausgangsbasis für die Implication-Chart oder Implikationstafel aufgestellt werden



- Obere und untere Dreiecksmatrix trägt selbe Information
- Diagonaleinträge werden für die Betrachtung nicht benötigt
- Im folgenden nur Verwendung der unteren Dreiecksmatrix

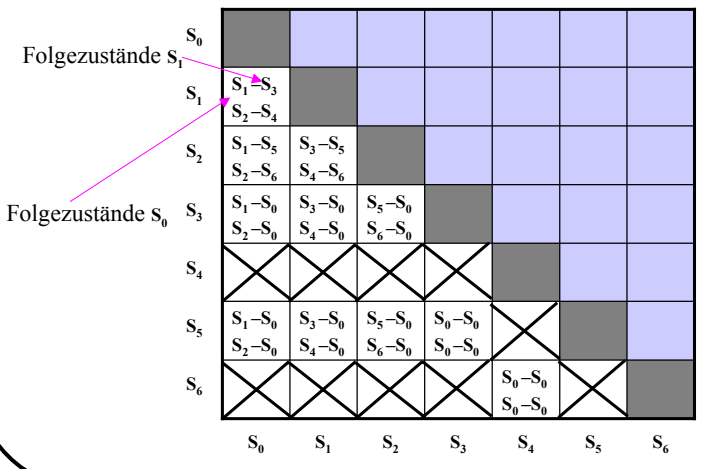
Zustandsminimierung

- Die Implikationstafel wird nun mit Einträgen wie folgt versehen:
  - Zunächst werden alle Felder von Zustandspaaren S_i und S_j mit unterschiedlichen Ausgangswerten mit X als nicht äquivalent gekennzeichnet



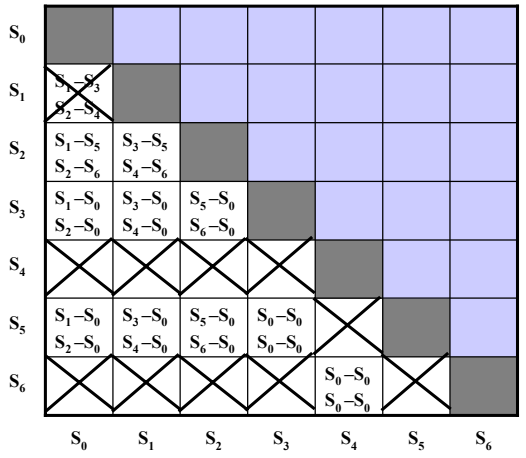
Zustandsminimierung

- Die Implikationstafel wird nun mit Einträgen vervollständigt:
  - Für gleiche Ausgangswerte werden in die Felder von Zustandspaaren S_i und S_j die Folgezustände eingetragen, z.B. $\mathbf{S_1 – S_3}$ heißt impliziertes Zustandspaar



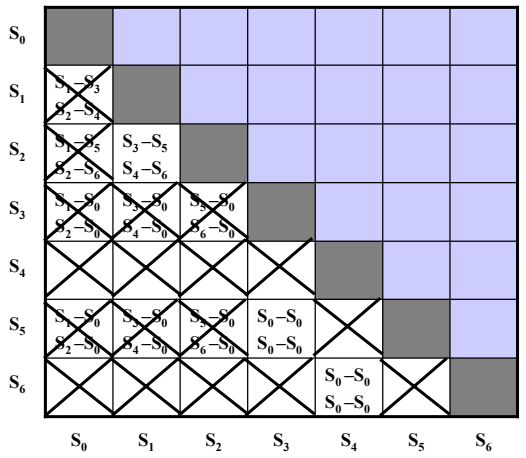
Zustandsminimierung

- Die Implikationstafel wird nun iterativ zur Markierung durchlaufen:
  - Für implizierte Zustandspaare, die nicht äquivalent sind, wird in die Felder von Zustandspaaren S_i und S_j ein X eingetragen, z.B. in S_1 und S_0 wegen $\mathbf{S_2 - S_4}$



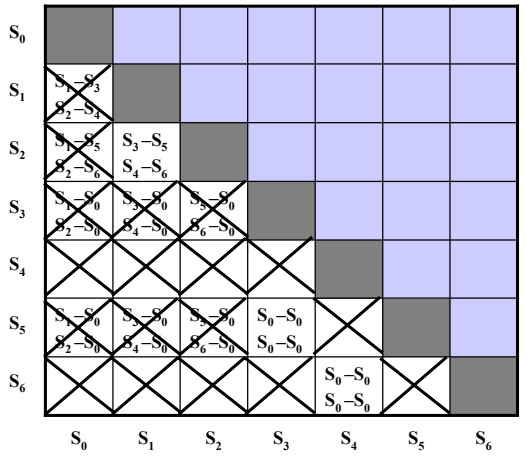
Zustandsminimierung

- Der erste Markierungsdurchlauf durch die Implikationstafel lässt nur drei Einträge unmarkiert



Zustandsminimierung

- Der zweite Markierungsdurchlauf durch die Implikationstafel lässt die drei bestehenden Einträge unmarkiert und das Verfahren wird abgebrochen
- S_3 und S_5 sowie S_4 und S_6 sind äquivalent und werden zusammen gefasst
- Entsprechendes gilt daher auch für S_1 und S_2



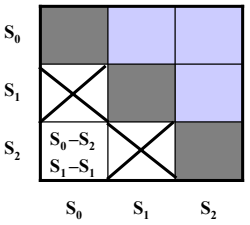
Zustandsminimierung

- Mittels der Implikationstafel reduzierte Zustandsübergangstabelle des 3-bit-Sequenzdetektors:

		Folgezustand		Ausgang	
Eingangssequenz	**Zustand**	**X=0**	**X=1**	**X=0**	**X=1**
Reset	S_0	S_1'	S_1'	**0**	**0**
0 oder 1	S_1'	S_3'	S_4'	**0**	**0**
00 oder 10	S_3'	S_0	S_0	**0**	**0**
01 oder 11	S_4'	S_0	S_0	**1**	**0**

- Die Minimierung reduzierte von 7 auf 4 Zustände (3 auf 2 FFs)

- Anwendung der Implikationstafelmethode auf Parity-Checker:



	Folgezustand		
Zustand	X=0	X=1	Ausgang
S_0	S_0	S_1	0
S_1	S_1	S_2	1
S_2	S_2	S_1	0

- Die Felder S_0 und S_1 sowie S_1 und S_2 scheiden durch unterschiedliche Ausgangswerte als nichtäquivalent aus
- Dagegen S_2 und S_0 können als äquivalent zusammengefasst werden
- Damit ist die korrekte minimale Zustandszahl gefunden worden

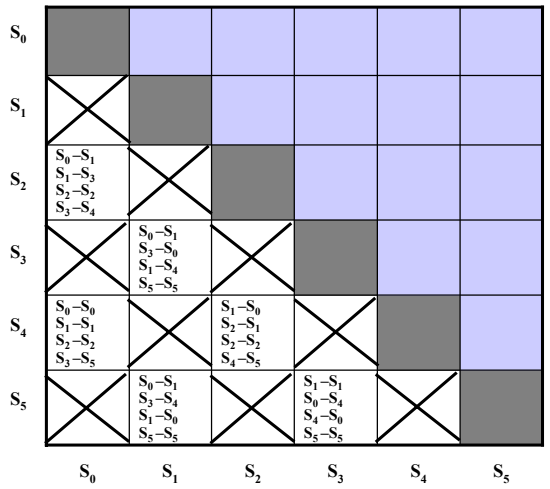
- Anwendung der Implikationstafel bei mehrstelligem Eingabevektor
- Willkürliches Beispiel für zweistelligen Eingabevektor:

	Folgezustand				Ausgang
Zustand	00	01	10	11	
S_0	S_0	S_1	S_2	S_3	1
S_1	S_0	S_3	S_1	S_5	0
S_2	S_1	S_3	S_2	S_4	1
S_3	S_1	S_0	S_4	S_5	0
S_4	S_0	S_1	S_2	S_5	1
S_5	S_1	S_4	S_0	S_5	0

- Die Implikationstafeleinträge erweitern sich auf vier Zeilen, da jeder Zustand nun vier Nachfolger hat

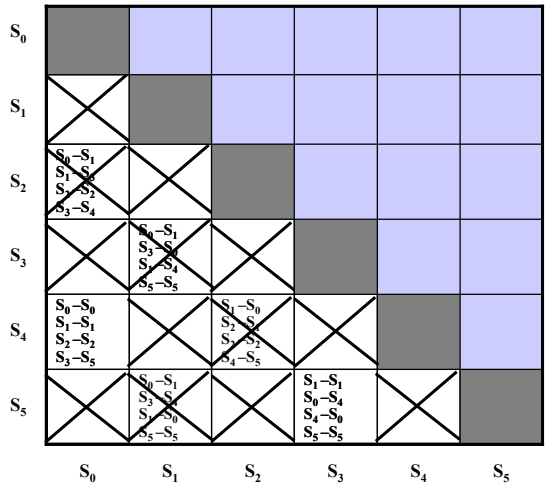
Zustandsminimierung

- Zunächst werden alle unvereinbaren Ausgaben in die Implikationstafel eingetragen

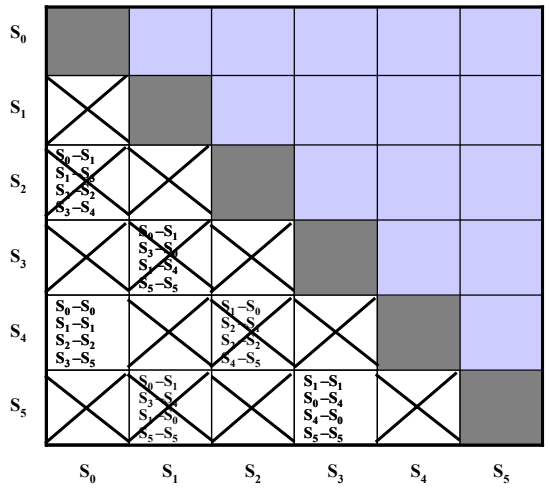


Zustandsminimierung

- Der erste Markierungsdurchlauf durch die Implikationstafel lässt nur zwei Einträge unmarkiert



- Ein weiterer Markierungsdurchlauf bringt keine weiteren Markierungen, das Verfahren wird abgebrochen



- Minimierte Zustandsübergangstabelle durch Zusammenfassung von S_3 und S_5 sowie S_0 und S_4

	Folgezustand				Ausgang
Zustand	**00**	**01**	**10**	**11**	
S_0'	S_0'	S_1	S_2	S_3'	**1**
S_1	S_0'	S_3'	S_1	S_3'	**0**
S_2	S_1	S_3'	S_2	S_0'	**1**
S_3'	S_1	S_0'	S_0'	S_3'	**0**

- Statt 6 werden nur noch 4 Zustände benötigt (Einsparung eines FFs)

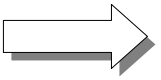
Zustandskodierung

- Nachdem nun eine Minimierung der vorliegenden Anzahl symbolischer Zustände erreicht wurde, kann die Umsetzung in eine konkrete Implementierung vorgenommen werden
- Dazu ist, wie bereits aufgeführt, die Kodierung der symbolischen Zustände durch Binärvektoren erforderlich.
- Gatter- und Verdrahtungsaufwand der Schaltnetze von Übergangs- und Ausgangsfunktion hängt von der gewählten Kodierung stark ab
- Das Auffinden der optimalen Kodierung erfordert ein vollständiges Durchsuchen und Bewerten aller Möglichkeiten
- Eine untere Aufwandsgrenze ist für dichte Kodierungen bei n Zuständen mit **n!** möglicher verschiedener Kodierungen gegeben [Katz 94]
- Die Verwendung sogenannter spärlicher Kodierungen kann zu deutlich höheren Kodierungsmöglichkeiten führen
- Daher ist auch hier der Einsatz von Heuristiken erforderlich
- Im folgenden liegt der Schwerpunkt auf manuellen Methoden gelegt, auf rechnergestützte Methoden und Werkzeuge wird hingewiesen [Katz 94]

Zustandskodierung

- Betrachtung des Beispiels Verkaufsautomat unter Nutzung von ESPRESSO zur Minimierung der Schaltnetze bei gegebener Kodierung und damit zur vergleichenden Bewertung !

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 S0 S0 0
01 S0 S1 0
10 S0 S2 0
11 S0 - -
00 S1 S1 0
01 S1 S2 0
10 S1 S3 0
11 S1 - -
00 S2 S2 0
01 S2 S3 0
10 S2 S3 0
11 S2 - -
00 S3 S3 1
01 S3 S3 1
10 S3 S3 1
11 S3 - -
.e
```



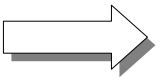
Kodierung:
S0 00
S1 01
S2 10
S3 11

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 00 00 0
01 00 01 0
10 00 10 0
11 00 -- -
00 01 01 0
01 01 10 0
10 01 11 0
11 01 -- -
00 10 10 0
01 10 11 0
10 10 11 0
11 10 -- -
00 11 11 1
01 11 11 1
10 11 11 1
11 11 -- -
.e
```

Zustandskodierung

- Ergebnis der ESPRESSO Minimierung:

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 00 00 0
01 00 01 0
10 00 10 0
11 00 -- -
00 01 01 0
01 01 10 0
10 01 11 0
11 01 -- -
00 10 10 0
01 10 11 0
10 10 11 0
11 10 -- -
00 11 11 1
01 11 11 1
10 11 11 1
11 11 -- -
.e
```



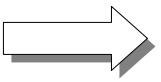
```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 7
-1-0 010
-1-1 100
-0-1 010
1-1- 010
--11 011
1--- 100
--1- 100
.e
```

7 UND, 3 ODER
12 + 8 = 20 Literale

Zustandskodierung

- Zweite von 4! Möglichkeiten:

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 S0 S0 0
01 S0 S1 0
10 S0 S2 0
11 S0 - -
00 S1 S1 0
01 S1 S2 0
10 S1 S3 0
11 S1 - -
00 S2 S2 0
01 S2 S3 0
10 S2 S3 0
11 S2 - -
00 S3 S3 1
01 S3 S3 1
10 S3 S3 1
11 S3 - -
.e
```



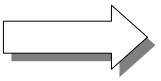
Kodierung:
S0 01
S1 10
S2 11
S3 00

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 01 01 0
01 01 10 0
10 01 11 0
11 01 -- -
00 10 10 0
01 10 11 0
10 10 00 0
11 10 -- -
00 11 11 0
01 11 00 0
10 11 00 0
11 11 -- -
00 00 00 1
01 00 00 1
10 00 00 1
11 00 -- -
.e
```

Zustandskodierung

- Ergebnis der ESPRESSO Minimierung:

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 01 01 0
01 01 10 0
10 01 11 0
11 01 -- -
00 10 10 0
01 10 11 0
10 10 00 0
11 10 -- -
00 11 11 0
01 11 00 0
10 11 00 0
11 11 -- -
00 00 00 1
01 00 00 1
10 00 00 1
11 00 -- -
.e
```



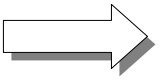
```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 7
-110 010
0011 110
0-10 100
1-01 100
-001 010
-101 100
--00 001
.e
```

7 UND, 3 ODER
21 + 8 = 29 Literale

Zustandskodierung

- Dritte von 4! Möglichkeiten:

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 S0 S0 0
01 S0 S1 0
10 S0 S2 0
11 S0 - -
00 S1 S1 0
01 S1 S2 0
10 S1 S3 0
11 S1 - -
00 S2 S2 0
01 S2 S3 0
10 S2 S3 0
11 S2 - -
00 S3 S3 1
01 S3 S3 1
10 S3 S3 1
11 S3 - -
.e
```



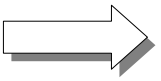
Kodierung:
S0 00
S1 10
S2 11
S3 01

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 00 00 0
01 00 10 0
10 00 11 0
11 00 -- -
00 10 10 0
01 10 11 0
10 10 01 0
11 10 -- -
00 11 11 0
01 11 01 0
10 11 01 0
11 11 -- -
00 01 01 1
01 01 01 1
10 01 01 1
11 01 -- -
.e
```

Zustandskodierung

- Ergebnis der ESPRESSO Minimierung:

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 00 00 0
01 00 10 0
10 00 11 0
11 00 -- -
00 10 10 0
01 10 11 0
10 10 01 0
11 10 -- -
00 11 11 0
01 11 01 0
10 11 01 0
11 11 -- -
00 01 01 1
01 01 01 1
10 01 01 1
11 01 -- -
.e
```

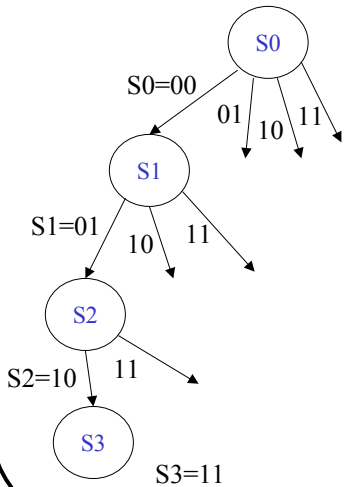


```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 7
001- 100
1-00 100
--01 001
-1-0 100
-11- 010
---1 010
1--- 010
.e
```

7 UND, 3 ODER
14 + 7 = 21 Literale

Zustandskodierung

- Eine FSM mit n Zuständen hat n! Kodierungsmöglichkeiten
- Für n=4 folgen damit 1*2*3*4=**24** Möglichkeiten

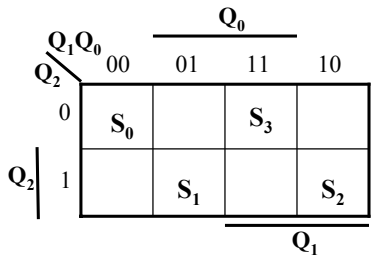


Hilfsdarstellung:

- Vier Möglichkeiten für S0
- Drei Möglichkeiten für S1
- Zwei Möglichkeiten für S2
- Festlegung für S3

Zustandskodierung

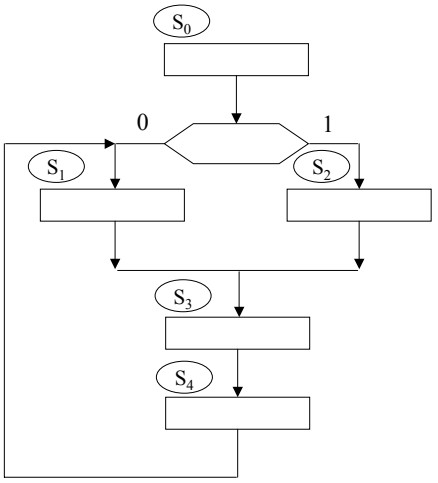
- Die kombinatorische Explosion ist in selbst bei kleineren Beispielen kaum durch manuelle Papier-und-Bleistift-Methoden zu fassen
- Typisch werden Werkzeuge vergleichbar zu ESPRESSO eingesetzt
- Für die manuelle Herangehensweise existieren graphische Hilfsmittel und heuristische Ansätze die z.T. regelgestützt sind (Entwurfsrichtlinien)
- Graphisches Hilfsmittel Zustandstafel (State-Map):



- Vergleichbar KV-Diagramm
- Binäre Kodierung der Zustände durch die $\mathbf{Q_i}$
- Nachbarschaften der in die Felder eingetragenen Zustände beobachtbar bzw. beeinflussbar
- Limitierung: ≤ 6 Variablen

Zustandskodierung

- ASM-Darstellung eines Automaten mit fünf Zuständen:



	Zustandskodierung		
Zustand	$\mathbf{Q_2}$	$\mathbf{Q_1}$	$\mathbf{Q_0}$
$\mathbf{S_0}$	0	0	0
$\mathbf{S_1}$	1	0	1
$\mathbf{S_2}$	1	1	1
$\mathbf{S_3}$	0	1	0
$\mathbf{S_4}$	0	1	1

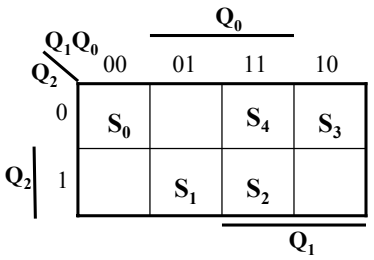
Die Zuordnung S0=0 bietet sich im Hinblick auf asynchrone Reset-Eingänge der FFs an

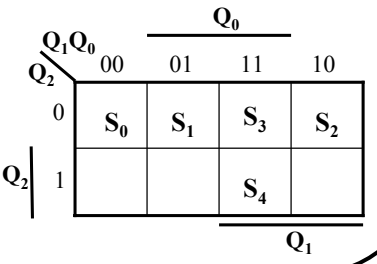
Zustandskodierung

- Zwei mögliche Kodierungen und die zugehörigen Zustandstafeln:

	Zustandskodierung		
Zustand	Q_2	Q_1	Q_0
S_0	0	0	0
S_1	1	0	1
S_2	1	1	1
S_3	0	1	0
S_4	0	1	1

	Zustandskodierung		
Zustand	Q_2	Q_1	Q_0
S_0	0	0	0
S_1	0	0	1
S_2	0	1	0
S_3	0	1	1
S_4	1	1	1

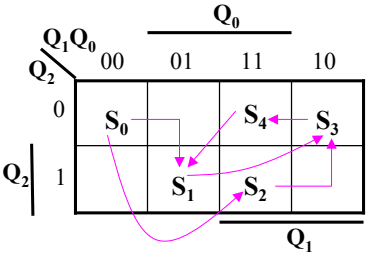


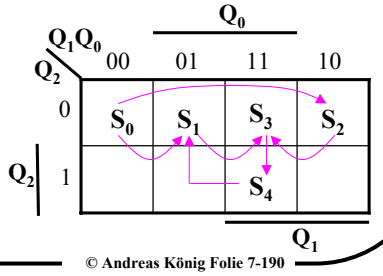


Zustandskodierung

- Eine einfache heuristische Strategie minimiert die Summe der Zahl der sich durch einen Zustandswechsel ändernden bits (Σ Hamming-Distanzen)

Übergang	Variante 1	Variante 2
S_0 nach S_1	2	1
S_0 nach S_2	3	1
S_1 nach S_3	3	1
S_2 nach S_3	2	1
S_3 nach S_4	1	1
S_4 nach S_1	2	2
Summe:	13	7





Zustandskodierung

- Die Heuristik zur Minimierung der bit-Wechsel zielt auf die Vereinfachung der Übergangsfunktionen ab
- Die Vorgehensweise ist einfach, findet aber nicht die günstigsten Zustands-kodierungen
- Bislang bleiben Ein- und Ausgangswerte bei der Betrachtung benachbarter oder folgender Zustände unberücksichtigt
- **Abhilfe:** Erweiterte Richtlinien (Regeln) basierend auf Folgezustand und Eingängen/Ausgängen:

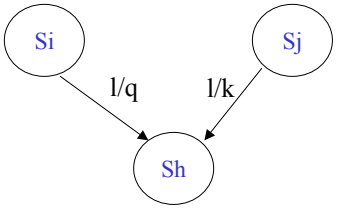
  Höchste Priorität: Zustände mit selben Folgezustand für eine Eingangsbelegung sollten benachbart in der Zustandstafel angeordnet werden

  Mittlere Priorität: Folgezustände eines vorhergehenden Zustands sollten benachbart in der Zustandstafel angeordnet werden

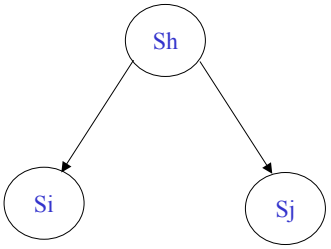
  Niedrigste Priorität: Zustände mit gleichem Ausgang für gegebene Eingangsbelegung sollten benachbart in der Zustandstafel angeordnet werden

Zustandskodierung

- Veranschaulichung der Regeln:



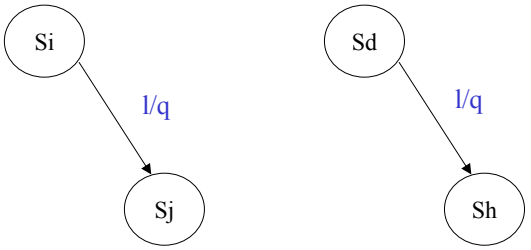
Höchste Priorität



Mittlere Priorität

Zustandskodierung

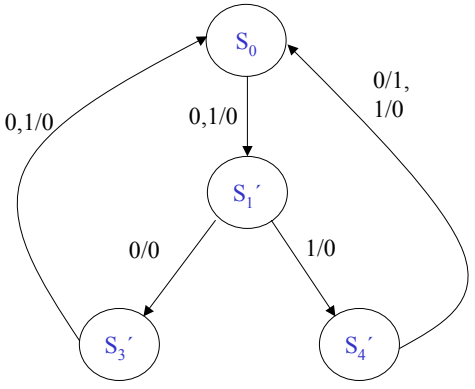
- Veranschaulichung der Regeln (Fortsetzung):



Niedrigste Priorität

Zustandskodierung

- Anwendungsbeispiel 3-bit Sequenzdetektor
- Vereinfachtes Zustandsdiagramm nach Zustandsminimierung (Folie 172)



Höchste Priorität: ($S_3´$, $S_4´$)

Mittlere Priorität: ($S_3´$, $S_4´$)

Niedrigste Priorität:

0/0: (S_0 , $S_1´$, $S_3´$)

1/0: (S_0 , $S_1´$, $S_3´$, $S_4´$)

Zustandskodierung

- Nach Aufstellung der Prioritäten für die Nachbarschaften können nun Kodierungen unter diesen einschränkenden Randbedingungen vorgenommen werden ($S_0 = 0$)

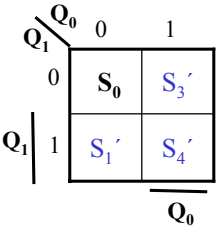
Höchste Priorität: (S_3', S_4')

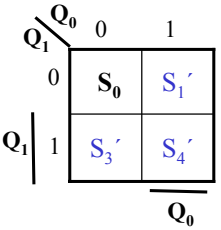
Mittlere Priorität: (S_3', S_4')

Niedrigste Priorität:

0/0: (S_0 , S_1', S_3')

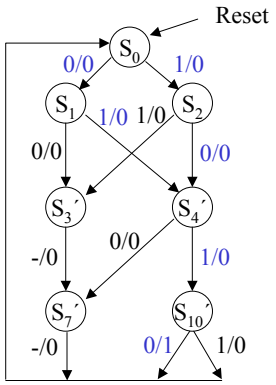
1/0: (S_0 , S_1', S_3', S_4')





Zustandskodierung

- Komplexeres Beispiel des 4-bit Sequenzdetektors:



Höchste Priorität:

(S_3', S_4'), (S_7', S_{10}')

Mittlere Priorität:

(S_1, S_2), 2*(S_3', S_4'), (S_7', S_{10}')

Niedrigste Priorität:

0/0: (S_0, S_1, S_2', S_3', S_4', S_7')

1/0: (S_0, S_1, S_2', S_3', S_4', S_7', S_{10}')

Zustandskodierung

- Kodierung des 4-bit Sequenzdetektors:

Höchste Priorität:

(S_3', S_4'), (S_7', S_{10}')

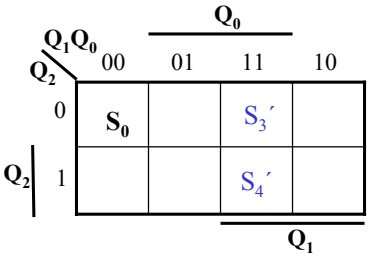
Mittlere Priorität:

(S_1, S_2), 2*(S_3', S_4'), (S_7', S_{10}')

Niedrigste Priorität:

0/0: (S_0, S_1, S_2', S_3', S_4', S_7')

1/0: (S_0, S_1, S_2', S_3', S_4', S_7', S_{10}')



Zustandskodierung

- Kodierung des 4-bit Sequenzdetektors:

Höchste Priorität:

(S_3', S_4'), (S_7', S_{10}')

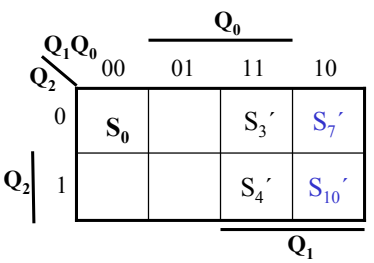
Mittlere Priorität:

(S_1, S_2), 2*(S_3', S_4'), (S_7', S_{10}')

Niedrigste Priorität:

0/0: (S_0, S_1, S_2', S_3', S_4', S_7')

1/0: (S_0, S_1, S_2', S_3', S_4', S_7', S_{10}')



Zustandskodierung

- Kodierung des 4-bit Sequenzdetektors:

Höchste Priorität:

(S_3', S_4'), (S_7', S_{10}')

Mittlere Priorität:

(S_1, S_2), 2*(S_3', S_4'), (S_7', S_{10}')

Niedrigste Priorität:

0/0: (S_0, S_1, S_2, S_3', S_4', S_7')

1/0: (S_0, S_1, S_2, S_3', S_4', S_7', S_{10}')

Q_2 \ Q_1Q_0	00	01	11	10
0	S_0	S_1	S_3'	S_7'
1		S_2	S_4'	S_{10}'

Zustandskodierung

- Alternative Lösung:

Q_2 \ Q_1Q_0	00	01	11	10
0	S_0			
1				

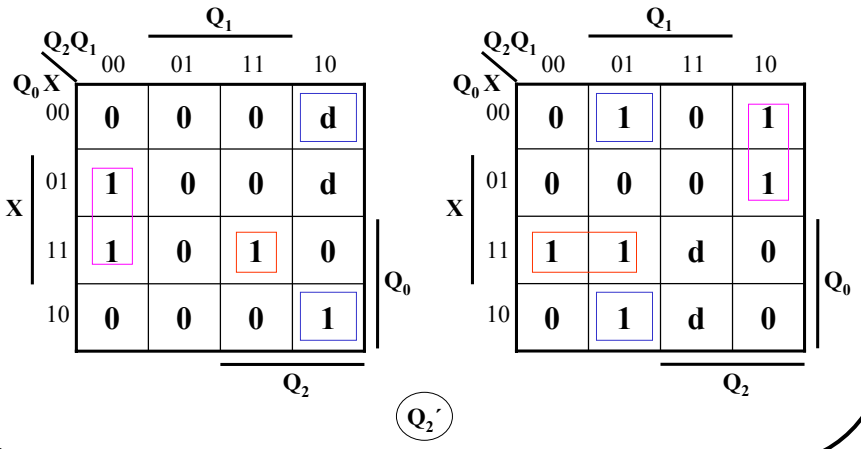
Q_2 \ Q_1Q_0	00	01	11	10
0	S_0			
1	S_7'			S_{10}'

Q_2 \ Q_1Q_0	00	01	11	10
0	S_0		S_3'	
1	S_7'		S_4'	S_{10}'

Q_2 \ Q_1Q_0	00	01	11	10
0	S_0	S_1	S_3'	
1	S_7'	S_2	S_4'	S_{10}'

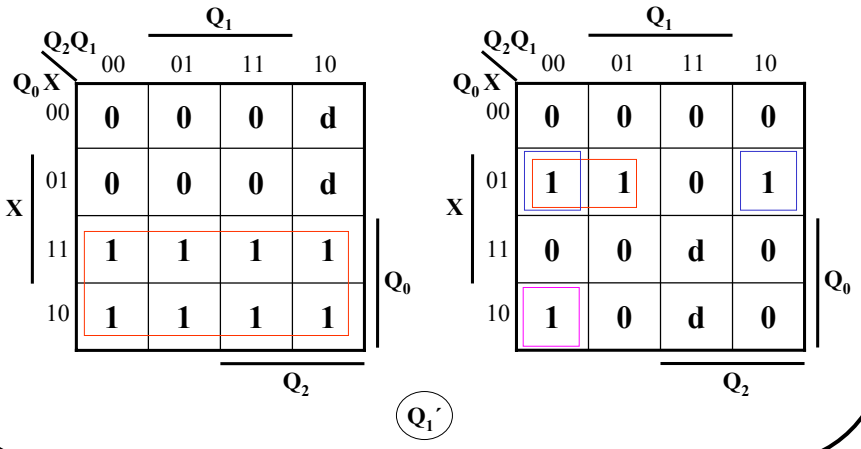
Zustandskodierung

- Veranschaulichung des Effekts der Richtlinien auf die Ballung von Einselementen
- Vergleich mit *naiver* Kodierung: S_0=000, S_1=001, S_2=010, S_3'=011, S_4'=100, S_7'=101, S_{10}'=111



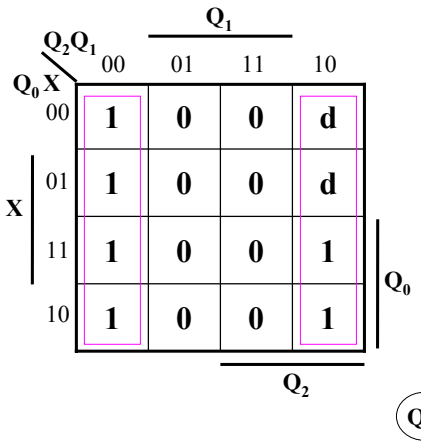
Zustandskodierung

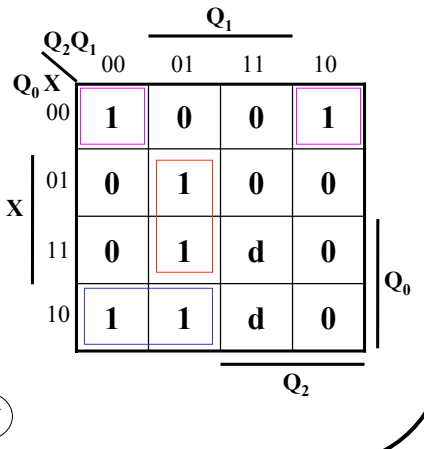
- Veranschaulichung des Effekts der Richtlinien auf die Ballung von Einselemente (Fortsetzung)
- Vergleich mit *naiver* Kodierung: S_0=000, S_1=001, S_2=010, S_3'=011, S_4'=100, S_7'=101, S_{10}'=111



Zustandskodierung

- Veranschaulichung des Effekts der Richtlinien auf die Ballung von Einselemente
- Vergleich mit *naiver* Kodierung: S_0=000, S_1=001, S_2=010, S_3'=011, S_4'=100, S_7'=101, S_{10}'=111





Zustandskodierung

- Abweichungen von dichten Kodierungen, die versuchen mit so wenig wie möglich an bits (FFs) auszukommen, führen zu spärlichen Kodierungen
- In Erwartung der Logikreduktion von Übergangs- und Ausgangsfunktion werden weitere FFs eingeführt
- Eine mögliche Kodierung: 1-aus-n Kodierung (*One-Hot-Encoding*):

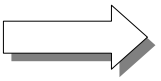
Kodierung Verkaufsautomat

0DM:	0001
0,5DM:	0010
1DM:	0100
1,5DM:	1000

Zustandskodierung

- Eine 1-aus-N-Kodierung:

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 S0 S0 0
01 S0 S1 0
10 S0 S2 0
11 S0 - -
00 S1 S1 0
01 S1 S2 0
10 S1 S3 0
11 S1 - -
00 S2 S2 0
01 S2 S3 0
10 S2 S3 0
11 S2 - -
00 S3 S3 1
01 S3 S3 1
10 S3 S3 1
11 S3 - -
.e
```



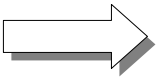
Kodierung:
S0 0001
S1 0010
S2 0100
S3 1000

```
.i 6
.o 5
.ilb m f q3 q2 q1 q0
.ob p3 p2 p1 p0 Out
.p 16
00 0001 0001 0
01 0001 0010 0
10 0001 0100 0
11 0001 ---- -
00 0010 0010 0
01 0010 0100 0
10 0010 1000 0
11 0010 ---- -
00 0100 0100 0
01 0100 1000 0
10 0100 1000 0
11 0100 ---- -
00 1000 1000 1
01 1000 1000 1
10 1000 1000 1
11 1000 ---- -
.e
```


Zustandskodierung

- Ergebnis der ESPRESSO Minimierung:

```
.i 6
.o 5
.ilb m f q3 q2 q1 q0
.ob p3 p2 p1 p0 Out
.p 16
00 0001 0001 0
01 0001 0010 0
10 0001 0100 0
11 0001 ---- -
00 0010 0010 0
01 0010 0100 0
10 0010 1000 0
11 0010 ---- -
00 0100 0100 0
01 0100 1000 0
10 0100 1000 0
11 0100 ---- -
00 1000 1000 1
01 1000 1000 1
10 1000 1000 1
11 1000 ---- -
.e
```



10 UND, 4 ODER
52 + 11 = 63 Literale

```
.i 6
.o 5
.ilb m f q3 q2 q1 q0
.ob p3 p2 p1 p0 Out
.p 10
-10001 00100
-10010 01000
1-0001 01000
1-0100 10000
-10100 10000
1-0010 10000
000001 00010
000010 00100
000100 01000
--1000 10001
.e
```

Zustandskodierung

- Ergebnis der hier gewählten Kodierung deutlich schlechter als Dichte Kodierung

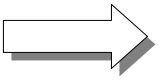
- Bei größerer Komplexität der Aufgabenstellung werden die manuellen Ansätze unbeherrschbar
- Entsprechend existieren eine Reihe von Werkzeugen vergleichbar zu ESPRESSO für die zweistufige Logikminimierung und misII für die mehrstufige Logikminimierung
- Nova für zweistufige Implementierungen [Katz 94]
- Mustang und jedi für mehrstufige Logik [Katz 94]

- Diese Werkzeuge der UCB sind jedoch leider nicht frei verfügbar

Zustandskodierung

- Bewertung der ersten Lösung des Tafelbeispiels 7.7:

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 S0 S0 0
01 S0 S1 0
10 S0 S2 0
11 S0 - -
00 S1 S1 0
01 S1 S2 0
10 S1 S3 0
11 S1 - -
00 S2 S2 0
01 S2 S3 0
10 S2 S3 0
11 S2 - -
00 S3 S3 1
01 S3 S3 1
10 S3 S3 1
11 S3 - -
.e
```



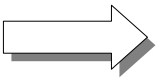
Kodierung:
S0 00
S1 01
S2 11
S3 10

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 00 00 0
01 00 01 0
10 00 11 0
11 00 -- -
00 01 01 0
01 01 11 0
10 01 10 0
11 01 -- -
00 11 11 0
01 11 10 0
10 11 10 0
11 11 -- -
00 10 10 1
01 10 10 1
10 10 10 1
11 10 -- -
.e
```

Zustandskodierung

- Ergebnis der ESPRESSO Minimierung:

```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 16
00 00 00 0
01 00 01 0
10 00 11 0
11 00 -- -
00 01 01 0
01 01 11 0
10 01 10 0
11 01 -- -
00 11 11 0
01 11 10 0
10 11 10 0
11 11 -- -
00 10 10 1
01 10 10 1
10 10 10 1
11 10 -- -
.e
```



```
.i 4
.o 3
.ilb m f q1 q0
.ob p1 p0 Out
.p 7
00-1 010
1-00 010
--10 001
-1-1 100
0-1- 100
-10- 010
1--- 100
.e
```

7 UND, 3 ODER
15 + 7 = 22 Literale

Zustandskodierung

- Zusammenstellung der untersuchten Zustandskodierungen:

Kodierung	S0	S1	S2	S3	Kosten
1	00	01	10	11	20
2	01	10	11	00	29
3	00	10	11	01	21
4	00	01	11	10	22

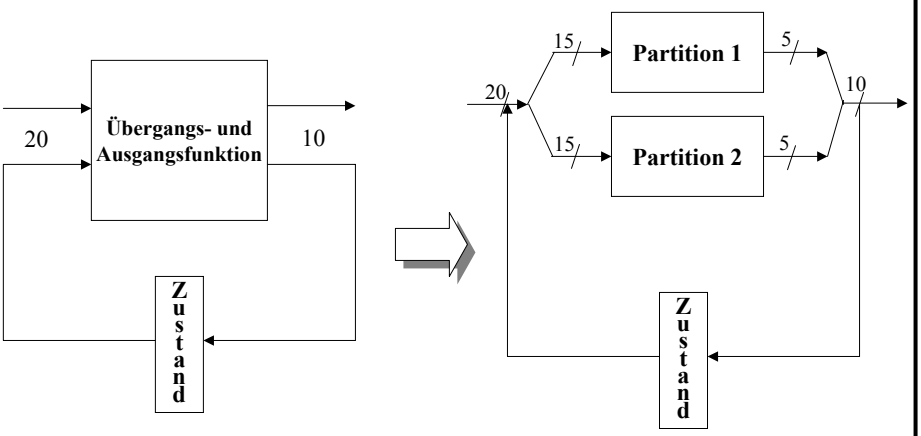
- Augenscheinlich ist die naive Lösung von den untersuchten vier aus 24 diejenige mit den niedrigsten Kosten
- Sie wird jedoch nicht von der Heuristik gefunden
- Jedoch wird die schlechteste Lösung vermieden
- Die beiden alternativen, regelgerechten Lösungen unterscheiden sich geringfügig in den Kosten
- Bei geringen Einbußen konnte eine vollständige Suche vermieden werden

FSM-Partitionierung

- Bislang wurde der Entwurf einer FSM als zusammenhängende Struktur betrachtet
- Jedoch können spezifische Anforderungen des gewählten Entwurfstils und der verfügbaren Bausteine bzw. Zellen weitere Anforderungen stellen
- Ist eine FSM nicht mit einem verfügbaren Baustein bzw. einer Zelle realisierbar, so muss eine geeignete Aufteilung erfolgen
- Man spricht in diesem Zusammenhang von einer FSM-Partitionierung
- Der Bedarf für eine solche Partitionierung hängt stark von der gewählten Entwurfsweise ab:
  - Geringe Bedeutung für z.B. VLSI/ULSI-Entwurf
  - Interessant für Verwendung begrenzter programmierbarer Logikbausteine oder auch entsprechender Zellen in komplexeren programmierbaren Bausteinen
- **Erster Ansatz:** Partitionierung der Übergangs- und Ausgangsfunktion zur Abbildung auf mehrere programmierbare Logikbausteine

FSM-Partitionierung

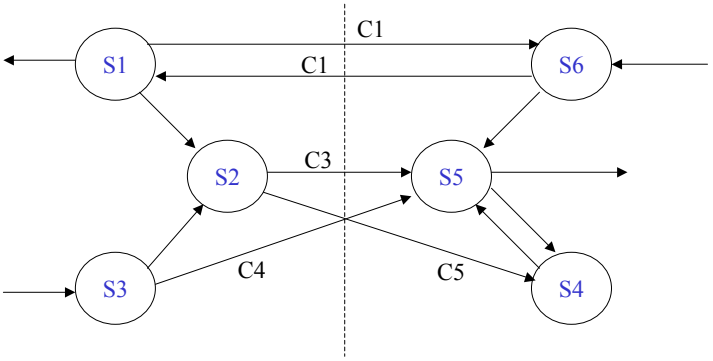
- Partitionierung von Übergangs- und Ausgangsfunktion:



- FSM mit 20 Eingängen und 10 Ausgängen der Übergangs- und Ausgangsfkt.
- Verfügbare Bausteine: Nur 15 Eingänge und 5 Ausgänge
- **Günstiger Sonderfall:** Schaltnetze aufteilbar in zwei 15/5 Partitionen

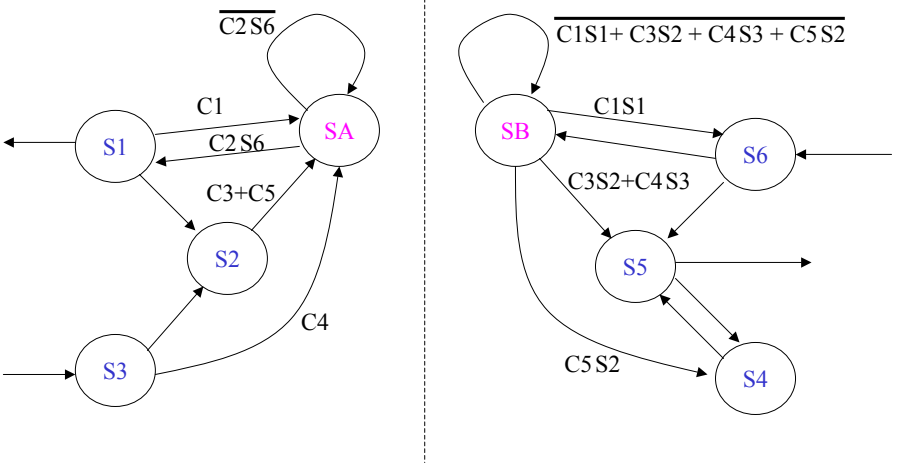
FSM-Partitionierung

- **Anderer Ansatz:** Partitionierung der FSM durch Einführung zusätzlicher Wartezustände (*Idle-States*)
- Die ursprüngliche FSM wird in zwei kommunizierende FSMs aufgeteilt
- Abwägung zwischen dem Aufwand neueingeführter Zustände und dafür erforderlicher FFs und der Reduktion der Schaltnetze für δ und λ
- Beispiel eines FSM-Teils mit sechs Zuständen der aufgeteilt werden soll:



FSM-Partitionierung

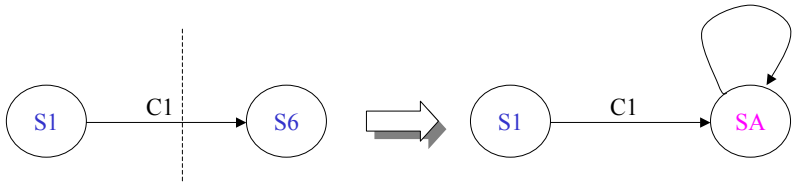
- Aufgeteiltes FSM-Segment:



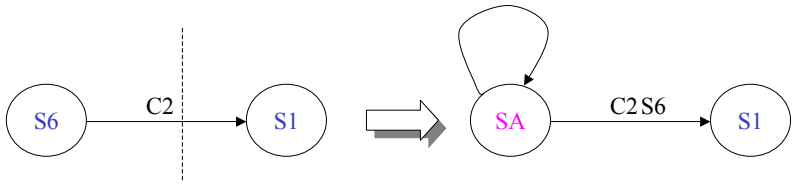
- Information aller Zustandsvariablen und der Eingänge werden in beiden Partitionen benötigt (Schnittstelle/Verdrahtungsminimierung)

FSM-Partitionierung

- **Regeln für die systematische Aufteilung**
- Einfacher Übergang über Partitionsgrenze:



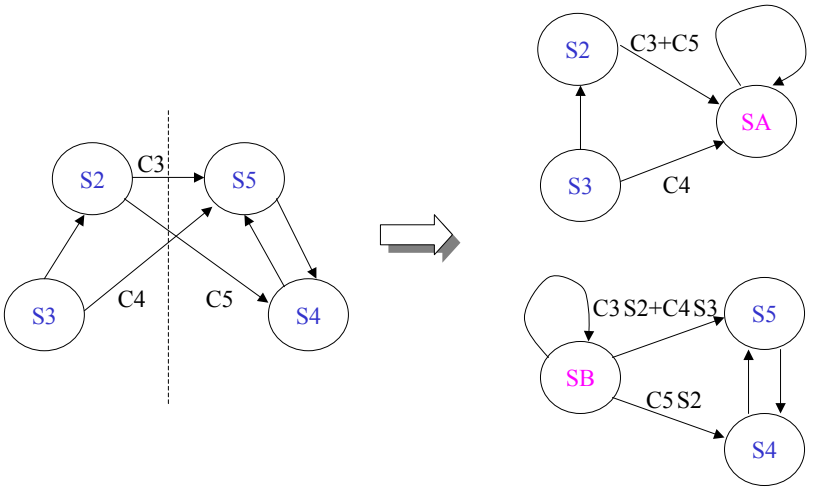
Transformation für Quellenzustand



Transformation für Zielzustand

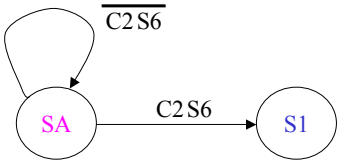
FSM-Partitionierung

- **Regeln für die systematische Aufteilung**
- Mehrfacher Übergang über Partitionsgrenze:



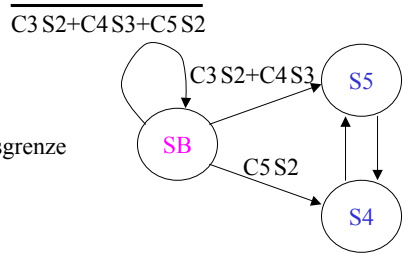
FSM-Partitionierung

- **Regeln für die systematische Aufteilung**
- Verhalten im Wartezustand:



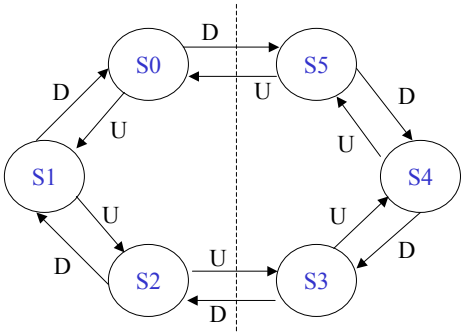
Einfacher Übergang über Partitionsgrenze

Mehrfacher Übergang über Partitionsgrenze



FSM-Partitionierung

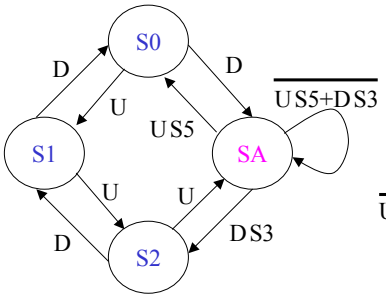
- **Anwendung auf einen simplen Up/Down-Counter mit sechs Zuständen:**



- Für aktives Eingangssignal U wird aufwärts, für aktives D abwärts gezählt
- Sonst wird im aktuellen Zustand verblieben
- Die Symmetrie der Anordnung lässt eine (Bi)-Partitionierung an jeder Stelle gleichwertig (teuer) zu

FSM-Partitionierung

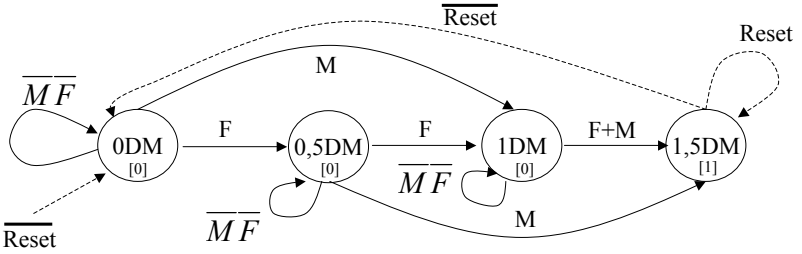
- Aufteilung in zwei Teile, z.B. angepasst an Xilinx CLB mit 2 FFs



- Jede der beiden FSM-Teile benötigt vier Zustände und damit 2 FFs
- Vier Signale müssen zwischen beide Teilen *verdrahtet* werden
- Partitionierung beinhaltet Fragen der Zusammenschaltung von FSMs

FSM-Zusammenschaltung

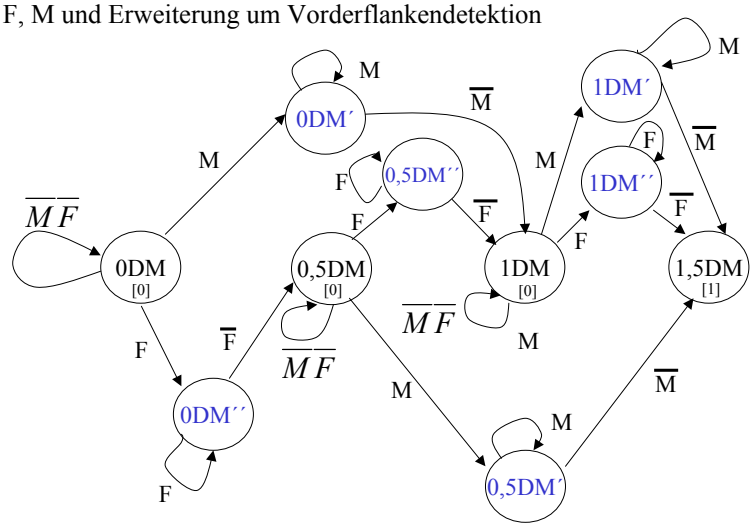
- Was kann die Zusammenschaltung von FSMs für einen Sinn/Nutzen haben ?



- Beispiel FSM Verkaufsautomat; Bislang galt Annahme, dass der Münzsensor für einen Münzeinwurf genau einen Takt ein aktives Signal liefert
- Dies setzt bereits eine Vorverarbeitung voraus
- **Schwächere Annahme:** Der Münzsensor liefert nur noch ein entprelltes (Monoflop) und einsynchronisiertes Signal variabler Dauer !

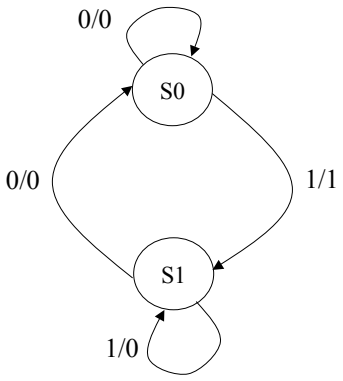
FSM-Zusammenschaltung

- **Konsequenz:** Eine (Vorder)Flankendetektion wird erforderlich
- Vereinfachung des Zustandsdiagramms auf Übergänge bedingt durch aktives F, M und Erweiterung um Vorderflankendetektion



FSM-Zusammenschaltung

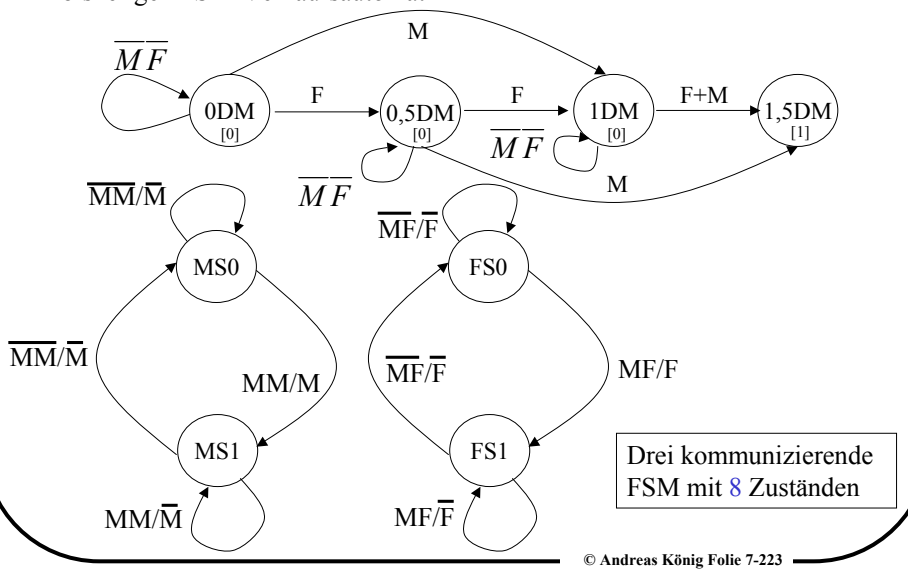
- Das Zustandsdiagramm wächst von 4 auf 10 Zustände für den vereinfachten Münzsensor an
- Kann die Flankendetektion und die folgende FSM modularisiert und zusammengeschaltet werden ?
- Wie könnte ein einfacher Vorderflankendetektor aussehen ?



- Mealy-Typ mit zwei Zuständen
- Abfrage des Sensors über beliebigen Zeitraum
- Für Münzeinwurf Aktivierung des Signals für genau einen Takt (*bzw. vom Moment der Aktivierung des Münzsensor-signals bis zum nächsten Takt*)

FSM-Zusammenschaltung

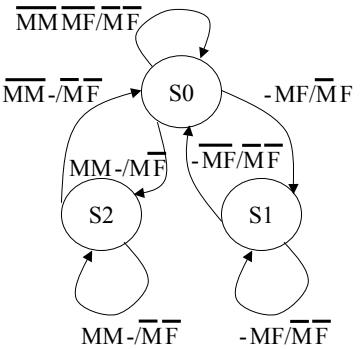
- Eine mögliche Zusammenschaltung von FSMs zur Flankendetektion mit der bisherigen FSM-Verkaufsautomat



Drei kommunizierende FSM mit 8 Zuständen

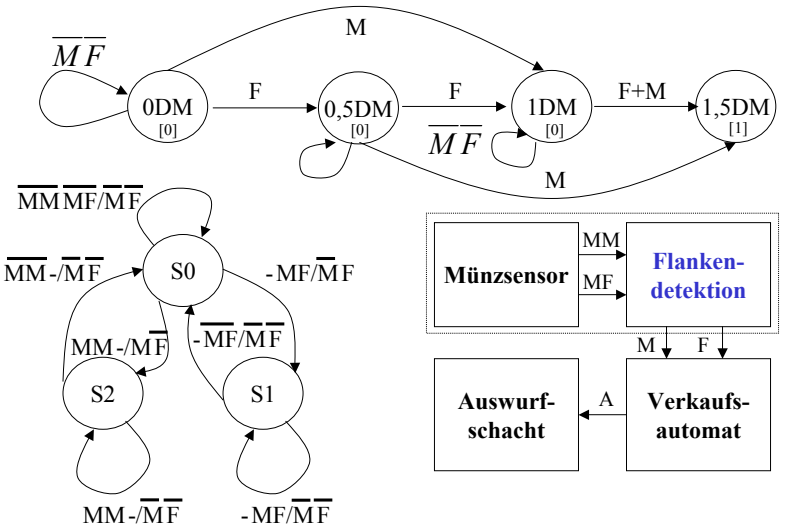
FSM-Zusammenschaltung

- Die Annahme, dass der Münzsensor beide Münztypen nicht gleichzeitig detektieren kann, gilt weiter
- Entsprechend können die FSM zur Flankendetektion zusammengefasst werden



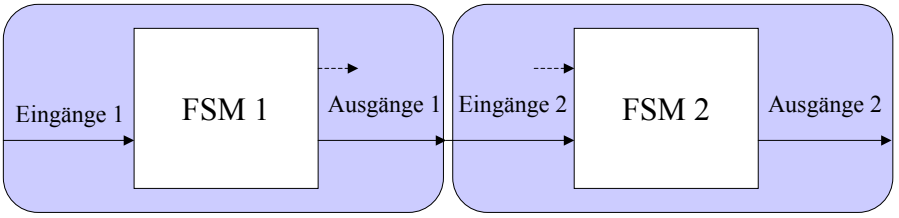
FSM-Zusammenschaltung

- Vereinfachte Zusammenschaltung von zwei FSMs mit 7 Zuständen



FSM-Zusammenschaltung

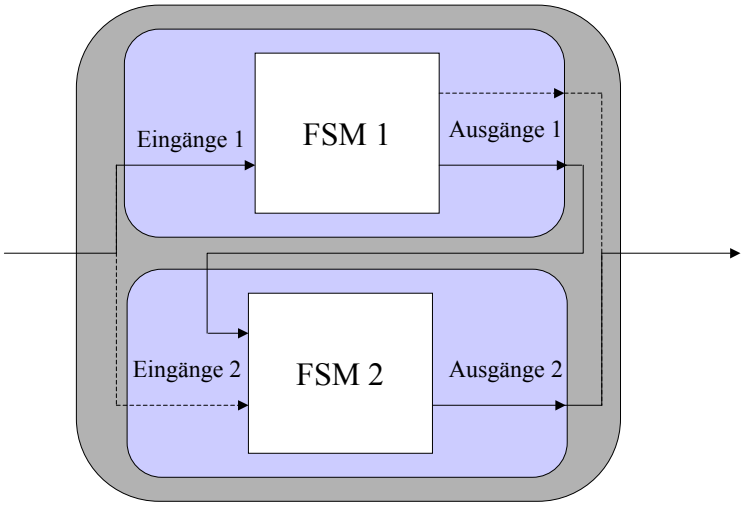
- Generelle Zusammenschaltung von Automaten:



- Die Zusammenschaltung der FSMs lässt sich wieder in ein sequentielles System überführen
- Für die FSMs können beliebige Automatentypen eingesetzt werden, bis auf die Ausnahme der Zusammenschaltung von Mealy-Maschinen
- Hier können sonst durch Ausgangsverknüpfungsnetzwerke direkte (asynchrone) Rückkopplungen gebildet werden !

FSM-Zusammenschaltung

- Überführung der Zusammenschaltung (hier: Serienschaltung) in ein System



- Ggf. lassen sich Schaltnetze zusammenfassen und vereinfachen

FSM-Zusammenschaltung

- Diese Erweiterung der Betrachtung wird z.B. bei Prozessoren/Rechnern angetroffen (Steuerwerk/Datenpfad)
- Allgemein erfordert die Beherrschung der Komplexität für die Zusammenschaltung sequentieller Systeme eine Erweiterung der Beschreibungsmöglichkeiten
- Beispielsweise sogenannte State-Charts [Harel 1987] dienen als graphisches Beschreibungsmittel
- Sie stellen Parallelität und Hierarchie dar
- Entwurfswerkzeuge, z.B. Statemate, SpeedChart

- Vertiefungsthema für Fachveranstaltung im Hauptstudium