

# Einführung in die Technische Informatik

## VLSI-Systementwurf

### Automaten

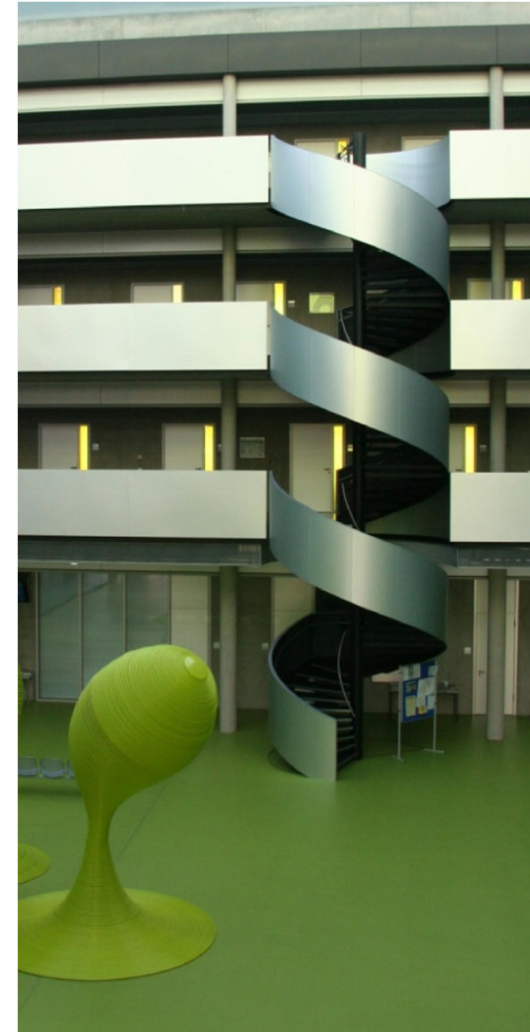
*Rainer G. Spallek  
Martin Zabel*

TU Dresden, 23.10.2013



## Gliederung

- 1 Automatendarstellung
- 2 Automatenkopplung
- 3 Synchrone Kopplung
- 4 Asynchrone Kopplung
- 5 Initialisierung
- 6 Zusammenfassung



# 1 Automatendarstellung

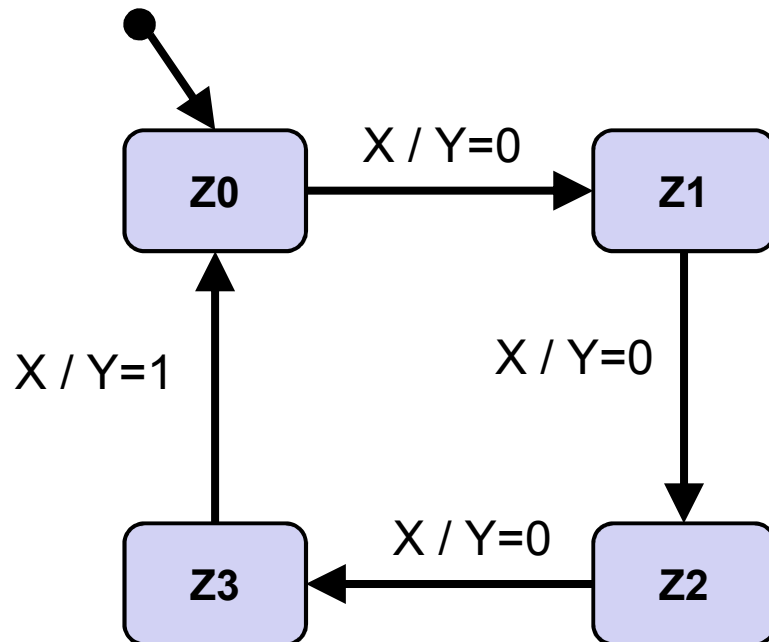
## 1.1 Betrachtungsweisen

### Verschiedene Sichten / Semantik:

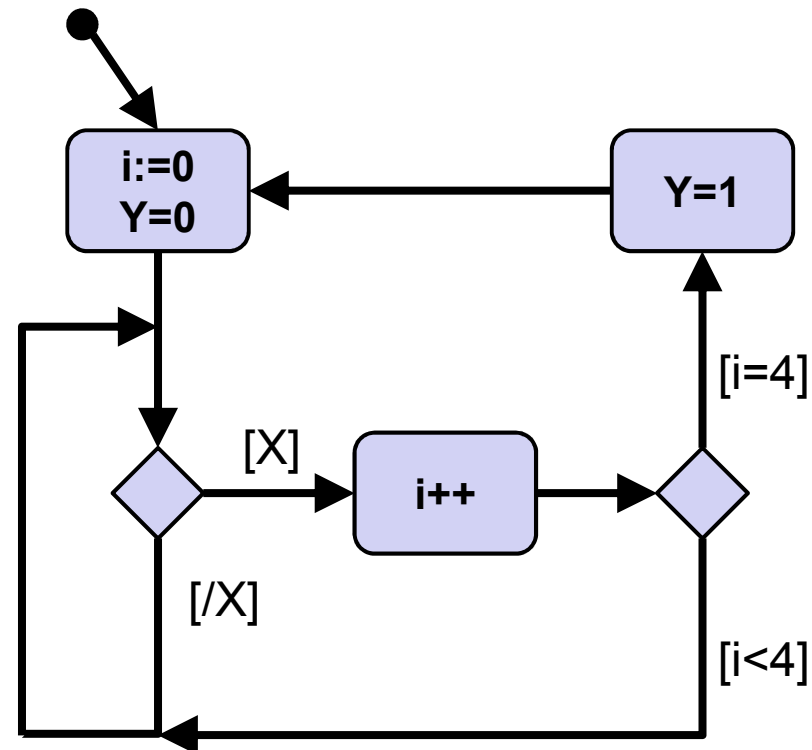
- Zustandsübergangsdiagramm (engl.: state diagram):
  - Vernetzung von Zuständen
  - Beispiele: UML-Zustandsdiagramm, Automatengraphen, SM Charts  
GRAFCET, Sequential Function Charts (SFC)
- Ablaufdiagramm (engl.: flow chart):
  - Vernetzung von Prozessen
  - Zustand ergibt sich aus der Verkettung aller Variablenzustände
  - Beispiele: UML-Aktivitätsdiagramm, Programmablaufplan (PAP)

## Binärzähler mod 4 mit Trigger X und Übertrag Y

**Zustandsübergangsdiagramm**  
(UML-Zustandsdiagramm)

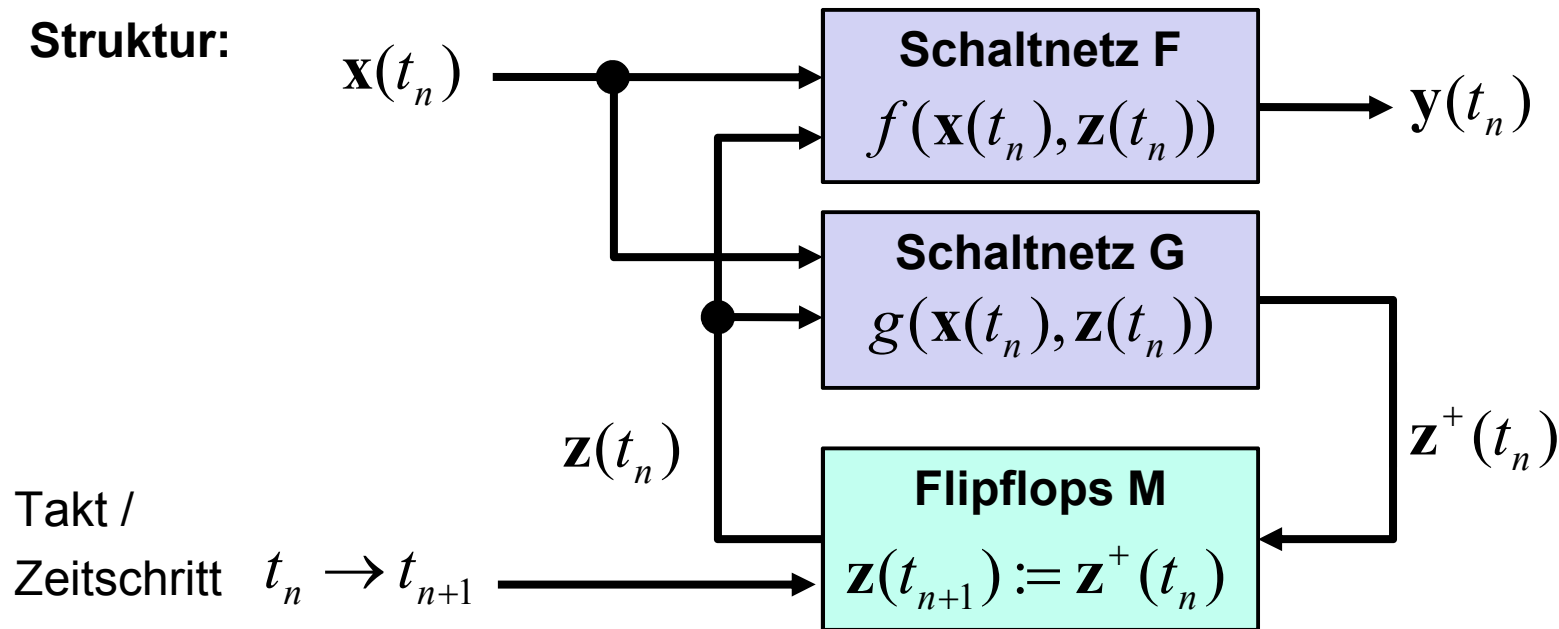


**Ablaufdiagramm**  
(UML-Aktivitätsdiagramm)



## 1.2 Automatengraphen

Struktur:



**Endlicher Automat:** Menge der möglichen Eingabezeichen, Ausgabezeichen und inneren Zustände ist endlich.

**Synchron getakteter Automat:** Zustandsübergänge aller Speicherglieder erfolgen gleichzeitig, synchron zu einem Taktsignal.

## Notation

### Beschreibung endlicher synchroner Automaten:

Eingabealphabet:  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_l\}$

Ausgabealphabet:  $\mathbf{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_m\}$

Zustandsmenge:  $\mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_k\}$

Anfangszustand:  $\mathbf{z}(t_0) \in \mathbf{Z}$

Menge der Endzust.:  $\mathbf{E} \subseteq \mathbf{Z}$

Übergangsfunktion:  $g : (\mathbf{x}_\lambda, \mathbf{z}_\kappa) \rightarrow \mathbf{z}_r$

Ausgabefunktion:  $f : (\mathbf{x}_\lambda, \mathbf{z}_\kappa) \rightarrow \mathbf{y}_\mu$

Automat:  $\mathbf{A} = (\mathbf{X}, \mathbf{Z}, \mathbf{Y}, \mathbf{z}(t_0), \mathbf{E}, f, g)$

### Weitere Vereinbarungen:

Eingabezeichen:  $\mathbf{x}_\lambda = (x_1, \dots, x_l) \in \mathbf{X}$

Ausgabezeichen:  $\mathbf{y}_\mu = (y_1, \dots, y_m) \in \mathbf{Y}$

Zustand:  $\mathbf{z}_\kappa = (z_1, \dots, z_k) \in \mathbf{Z}$

Eingangsvariable:  $x_\lambda \in \mathbf{U}$

Ausgangsvariable:  $y_\mu \in \mathbf{V}$

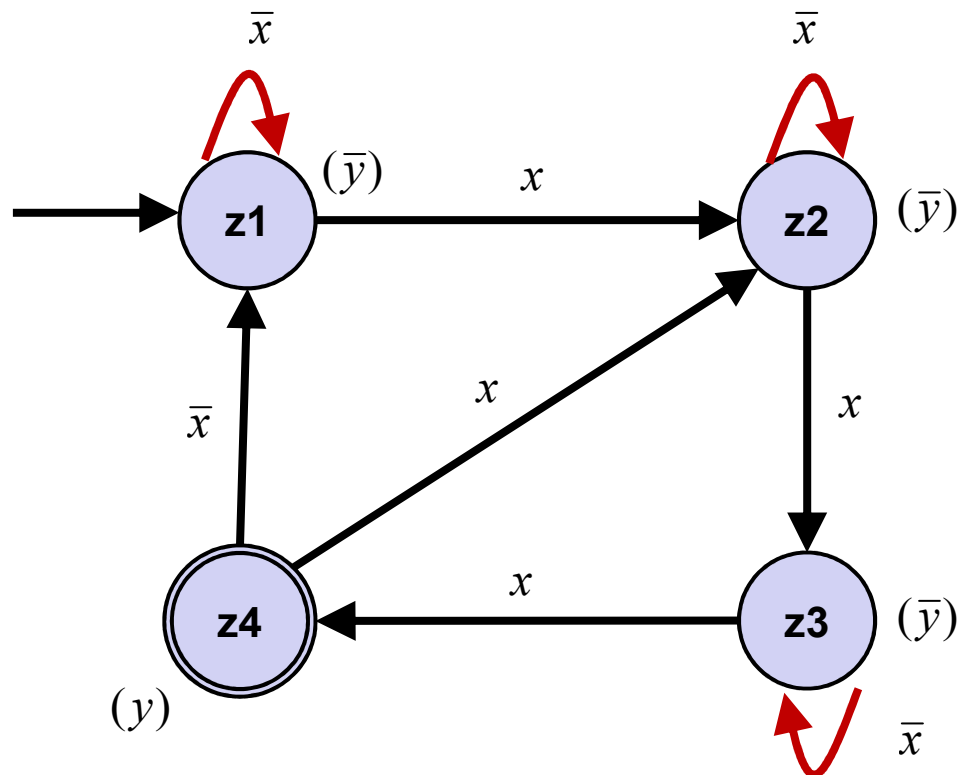
Zustandsvariable:  $z_\kappa \in \mathbf{W}$

Menge der Eing.-var.:  $\mathbf{U} = \{x_1, \dots, x_l\}$

Menge der Ausg.-var.:  $\mathbf{V} = \{y_1, \dots, y_m\}$

Menge der Zust.-var.:  $\mathbf{W} = \{z_1, \dots, z_k\}$

## Grafische Darstellung



$$\mathbf{X} = \{(x), (\bar{x})\}$$

$$\mathbf{Y} = \{(y), (\bar{y})\}$$

$$\mathbf{Z} = \{z_1, z_2, z_3, z_4\}$$

$$z(t_0) = z_1$$

$$\mathbf{E} = \{z_4\}$$

Was fehlt?

**Prüfung auf:** Vollständigkeit und Widerspruchsfreiheit

## Getaktete Automaten (1)

### Theoretische Informatik:

- Verarbeitung von Eingabezeichen sofern vorhanden.
- Jedes Zeichen in der Eingabe wird einmalig verarbeitet.

### Technische Informatik:

Taktung des Automaten mit einem Taktsignal → Abtastung der Eingabe.

Konsequenzen:

- Zeichen für „keine Eingabe“ erforderlich.
- Abtastung „derselben Eingabe“ in aufeinanderfolgenden Takten möglich.

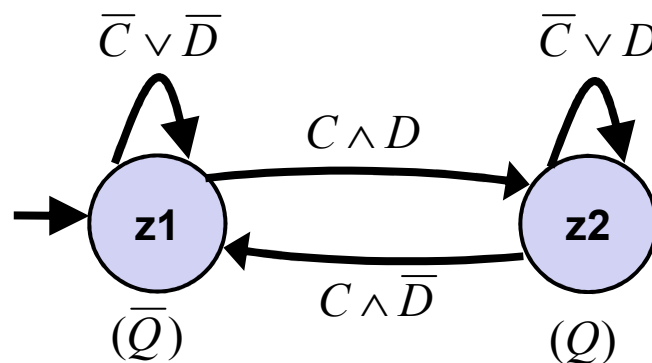
→ Korrekte Modellierung des Taktsignals erforderlich.



## Getaktete Automaten (2)

**Beispiel:** Taktzustandsgesteuertes D-Flip-Flop (Latch)

**Variante:** Taktsignal C als Eingangsvariable



$$\mathbf{X} = \{(\bar{C}, \bar{D}), (\bar{C}, D), (C, \bar{D}), (C, D)\}$$

$$\mathbf{Y} = \{(\bar{Q}), (Q)\}$$

$$\mathbf{Z} = \{z_1, z_2\}$$

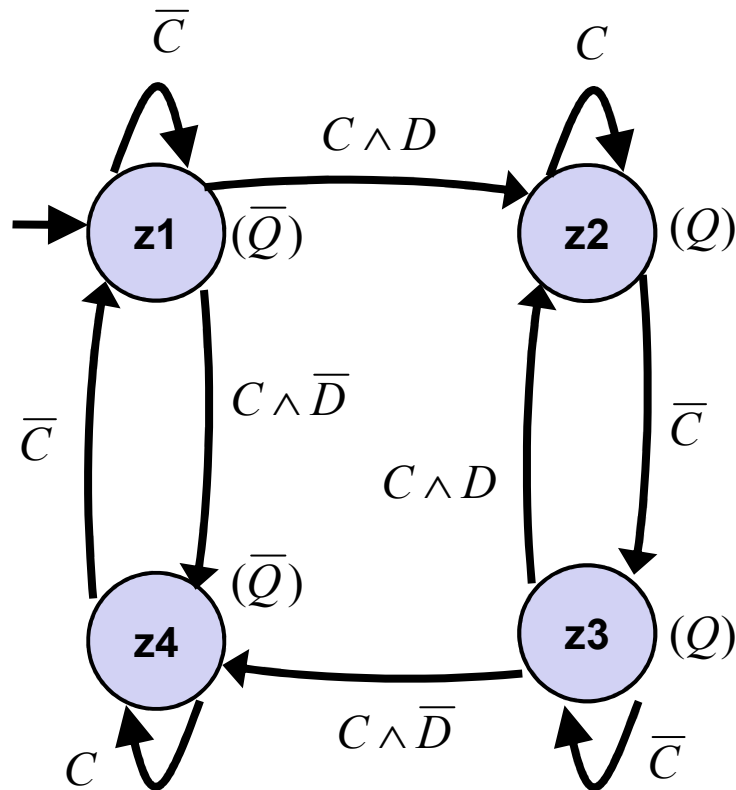
$$z(t_0) = z_1$$

$$\mathbf{E} = \{\}$$

## Getaktete Automaten (3)

**Beispiel:** Taktflanken gesteuertes D-Flip-Flop

**Variante:** Taktsignal C als Eingangsvariable



$$\mathbf{X} = \{(\bar{C}, \bar{D}), (\bar{C}, D), (C, \bar{D}), (C, D)\}$$

$$\mathbf{Y} = \{(\bar{Q}), (Q)\}$$

$$\mathbf{Z} = \{z_1, z_2, z_3, z_4\}$$

$$z(t_0) = z_1$$

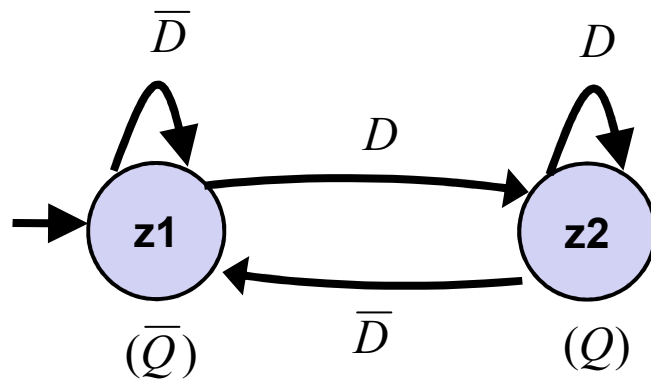
$$\mathbf{E} = \{\}$$

→ Zu kompliziert und  
auch nicht notwendig!

## Getaktete Automaten (4)

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

**Variante:** Definition: Zustandsübergang nur bei Taktflanke



$$\mathbf{X} = \{(\bar{D}), (D)\}$$

$$\mathbf{Y} = \{(\bar{Q}), (Q)\}$$

$$\mathbf{Z} = \{z_1, z_2\}$$

$$z(t_0) = z_1$$

$$\mathbf{E} = \{\}$$

## 1.3 SM Charts

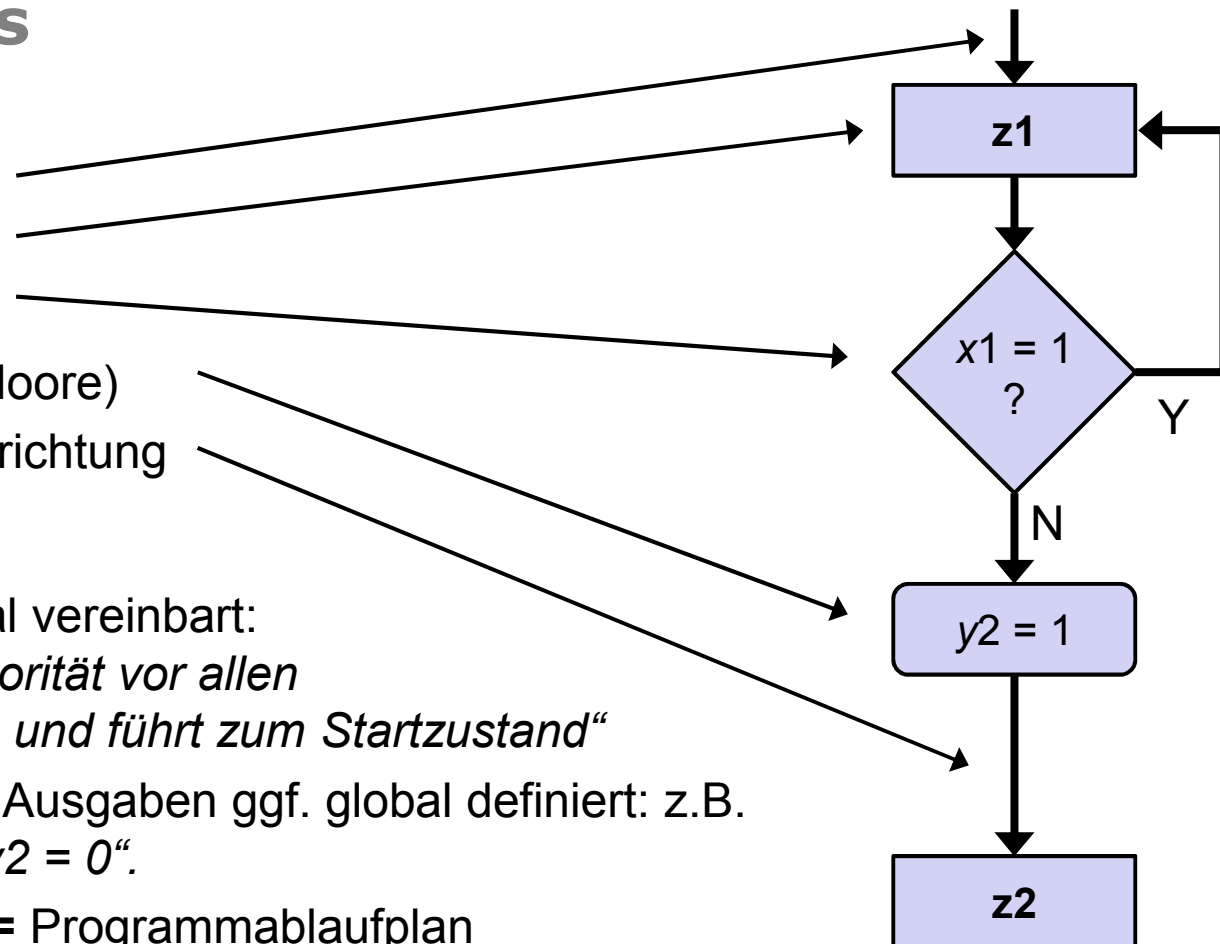
### Basiselemente:

- Start
- Zustand
- Verzweigung
- Ausgabe (Mealy/Moore)
- Verbindung / Leserichtung

### Weiteres:

- Reset i. Allg. global vereinbart:  
z.B. „Reset hat Priorität vor allen  
anderen Eingaben und führt zum Startzustand“
- Standardwerte für Ausgaben ggf. global definiert: z.B.  
„Standardmäßig:  $y2 = 0$ “.

**Achtung:** SM Chart != Programmablaufplan

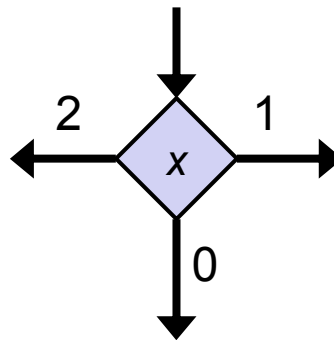
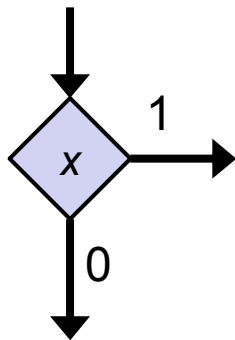


## Verzweigungen

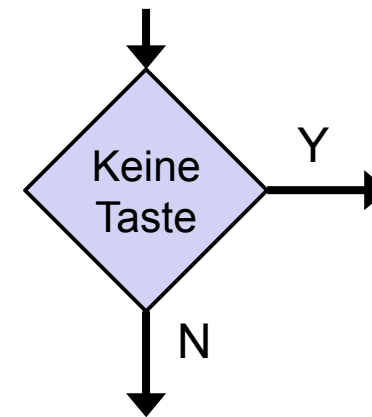
### Weitere Merkmale:

- Selbstdefinierte Abkürzungen für komplexe Ausdrücke üblich
- I. Allg. nur Prüfung einer Eingangsvariablen (nicht Eingabezeichen) pro Verzweigung

### Weitere Beispiele:



Aufzählung aller  
Möglichkeiten!



Definition der  
Abkürzung im Text.

## Ausgaben

### Ebenso:

Selbstdefinierte Abkürzungen für komplexe Ausdrücke üblich

### Weitere Beispiele:

$y_2$

$$y_2 = 1$$

$y = (0, 1, 0)$

$$y_0 = 0$$

$$y_1 = 1$$

$$y_2 = 0$$

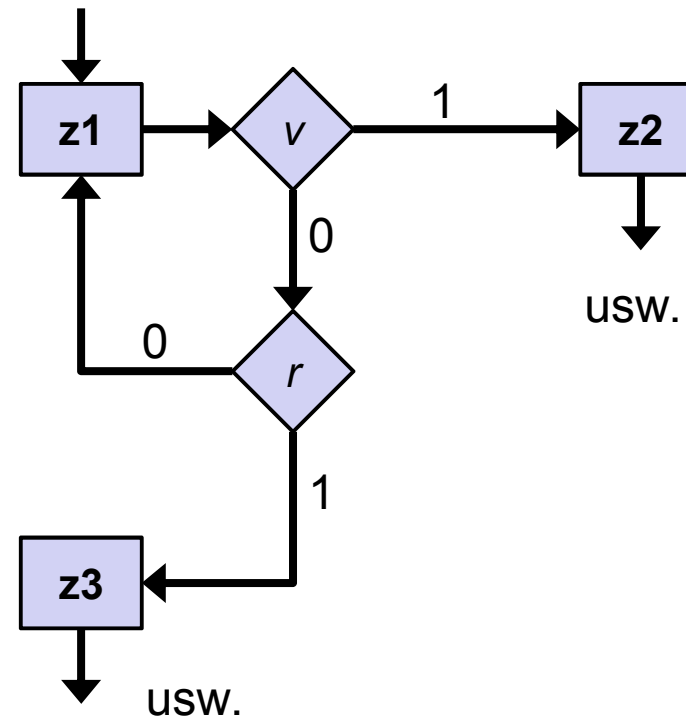
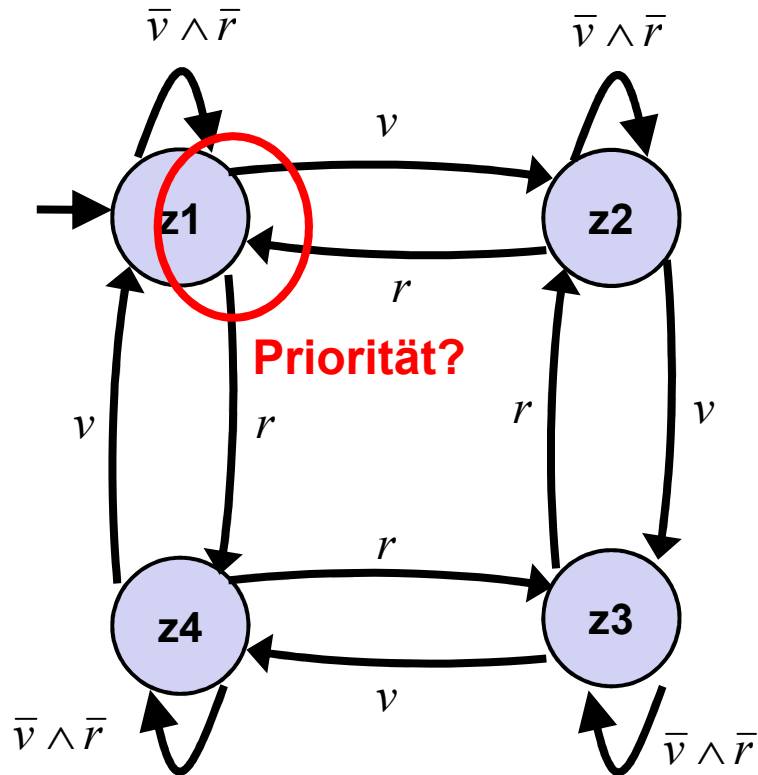
Definition des  
Vektors im Text!

Öffnen

Definition der  
Abkürzung im Text.

## Vorteile gegenüber Automatengraphen

- Prinzipiell vollständig
- Präzise Modellierung hierarchischer Verzweigungen



## 1.4 GRAFCET & SFC

### Allgemeines:

- Struktur entspricht einem Petri-Netz
- Darstellung von schrittweise ausgeführten Ablaufbeschreibungen
- Anwendung in Automatisierungstechnik und Verfahrenstechnik
- Verwendbar zur Programmierung von Speicherprogrammierbaren Steuerungen (SPS)
- Automatenkopplung nicht vorgesehen; muss durch Eingangs- und Ausgangsvariablen realisiert werden

GRAFCET - GRAPhe Functionnel de Commande Etapes/Transitions  
DIN EN 60848

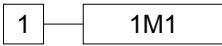

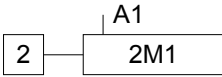
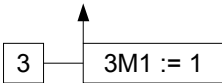
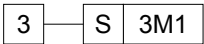
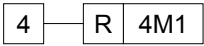
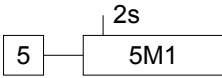
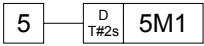
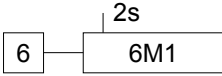


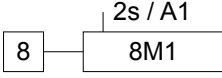
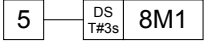
SFC - sequential function chart



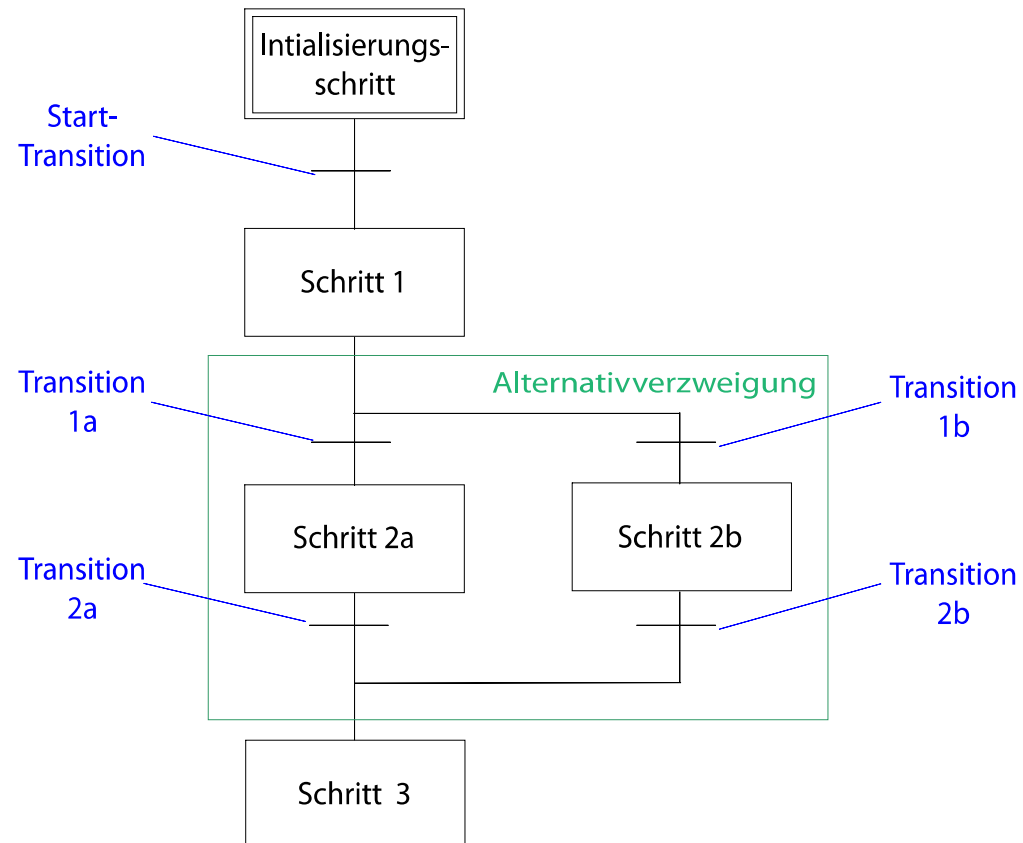
## Grundelemente

- Schritt:
  - entspricht einem Zustand
  - mindestens ein Initialisierungsschritt erforderlich
- Transition
  - Schaltbedingung für Übergang zwischen zwei Schritten (boolesche Gleichung) = zeitliche Ereignisse
  - Schritte und Transitionen folgen immer aufeinander.
  - Leserichtung: Transitionen können nur von oben nach unten durchlaufen werden.
- Aktion
  - einem Schritt zugeordnet
  - realisiert die Ausgabe
  - verschiedene Aktionsarten möglich

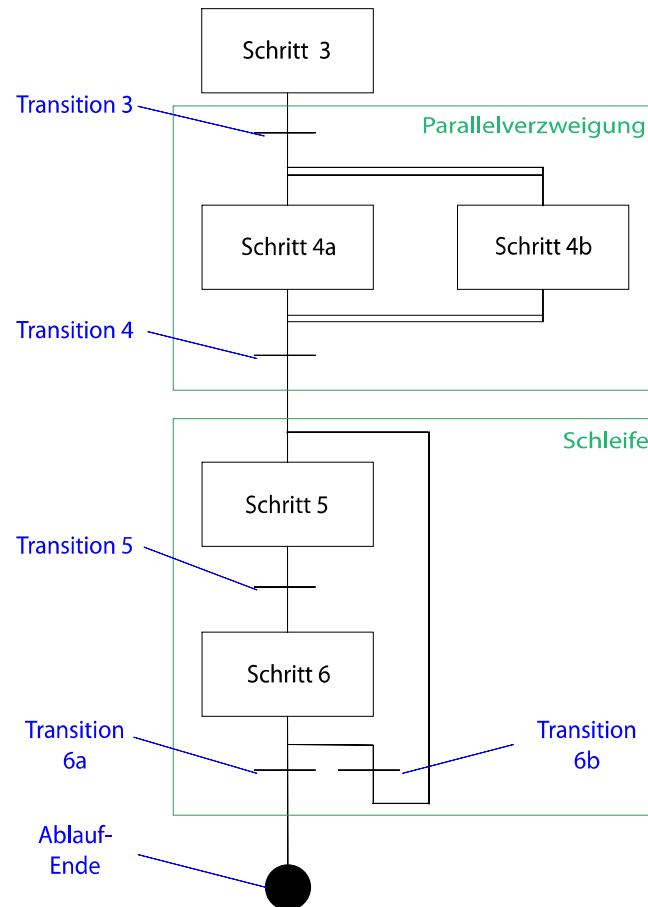
## Aktionen

Aktion	GRAFCET	Sequential Function Chart
kontinuierliche Aktion (solange Schritt aktiv ist)		
Aktion mit Zuweisungsbedingung		
Speichernde Aktion		
Rücksetzende Aktion		
verzögerte Aktion		
zeitbegrenzte Aktion		
Aktion für einen Taktzyklus		
kombinierte Aktion		

## Kontrollfluss: Verzweigung



## Kontrollfluss: Schleife



### Parallelverzweigung:

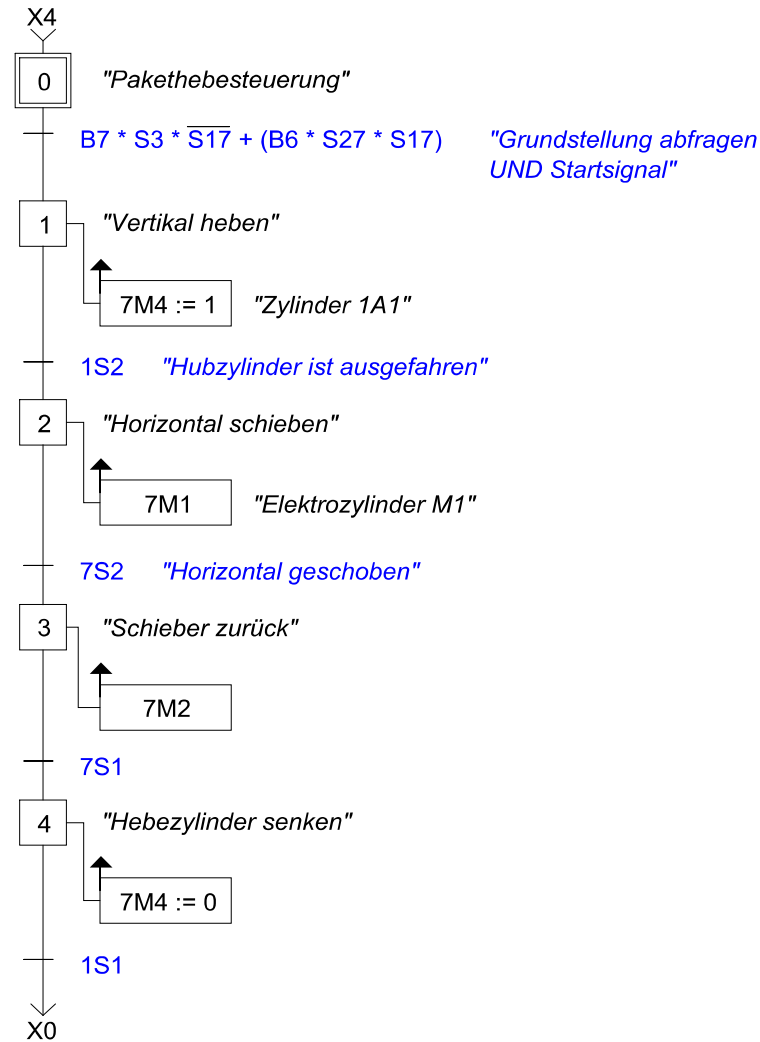
Gemeinsame Transition zum Start bzw. Ende aller Pfade

## Vergleich von SFC und GRAFCET

- identische Grundstruktur, Verzweigungen und Initialisierung
- Ablauf-Ende in GRAFCET nicht notwendig, dann Sprung zu Initialisierungsschritt
- Schleifen in SFC geschlossen, in GRAFCET offen
  - Vorwärtssprung = Alternativverzweigung
  - Rückwärtssprung = Schleife
- Aktionen: siehe oben

GRAFCET ist für den Entwurf konzipiert, SFC für die Implementierung

## Beispiel zu GRAFCET



---

## Weiterführende Literatur

- *Dokumentation in der Elektrotechnik, Darstellungsregeln*. DIN-VDE-Taschenbuch 530, Beuth Verlag, Berlin 2004, ISBN 3-410-15932-0. Darin DIN EN 60848 GRAFCET
- Gerhard Schmidt: *GRAFCET*. Festo Didactic, Esslingen 2007.
- Bernhard Plagemann: *Crashkurs GRAFCET*. Dr.-Ing. Paul Christiani GmbH, Konstanz 2008. ISBN 978-3-86522-441-5
- David Harel: *Statecharts: A Visual Approach to Complex Systems*, CS84-05, Department of Applied Mathematics, The Weizmann Institute of Science, 1984

## 2 Automatenkopplung

### Zerlegung der Gesamtaufgabe in Teilschaltungen zwecks:

- Nebenläufigkeit  
(statt sequentieller Abarbeitung)
- Energieeinsparung durch Abschaltung ungenutzter Komponenten  
(statt Dauerbetrieb der Gesamtschaltung)
- komponentenbasierter, testfreundlicher Entwurf  
(statt monolithischem Design)
- Integration von Komponenten von Drittanbietern  
(statt eigenem Entwurf aller Funktionen)

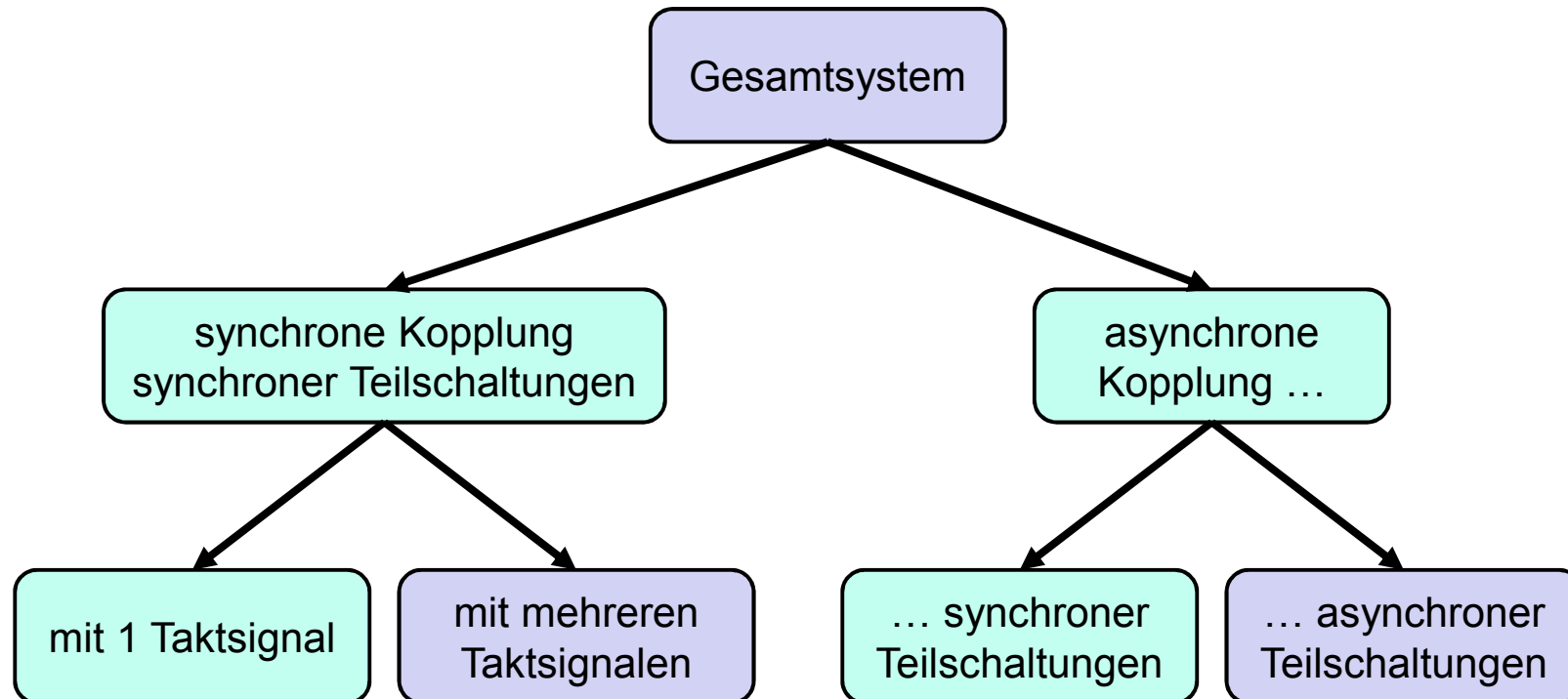
### Außerdem:

Kommunikation mit der Außenwelt erfordert i.Allg. nebenläufig arbeitende I/O-Controller aufgrund asynchron eintreffender Ereignisse (Nachrichten).

Bsp: Netzwerkcontroller in Mikroprozessorsystemen.



## Zeitliche Kopplung



## Synchrones Kopplung mit 1 Taktsignal

**Merkmal:** Synchrones Gesamtsystem mit 1 Taktsignal.

**Vorteile:**

- Einfaches Design
- Timing-Analyse für Gesamtsystem

**Nachteile:**

- Verlustleistung des Taktbaums.
- Kommunikation mit Außenwelt:
  - „High-Speed“-Verbindungen geben Taktfrequenz vor
  - Abtastung von Signalen anderer Frequenz nur für „Low-Speed“-Verbindungen praktisch möglich.
- Taktfrequenz bestimmt durch „langsamstes“ Modul,

## Asynchr. Kopplung synchr. Teilschaltungen

**Merkmal:** Mehrere (lokale) Taktsignale / Taktdomänen (engl.: clock domain)

**Vorteile:**

- Timing-Analyse sichert Zeitverhalten innerhalb der Domäne.
- Passende Taktfrequenz je Domäne:
  - Kombination High-Speed und Low-Power.
  - Taktfrequenz dynamisch anpassbar.
  - Abschalten einer Taktdomäne für Standby.

**Nachteil:**

Datenaustausch zwischen Taktdomains erfordern:

- Module wie Single-Bit-Synchronizer, Cross-Clock-FIFOs,
- Spezielle Timing-Constraints.

## Synchr. Kopplung mit mehreren Taktsignalen

**Merkmal:** Mehrere (lokale) Taktsignale / Taktdomänen, zwischen denen aber spezielle Abhängigkeiten bestehen (engl.: dependent clocks).

Tafelbild.

### Vorteile:

- Timing-Analyse sichert Zeitverhalten innerhalb der Domäne und auch zwischen den Domäne.
- Passende feste Taktfrequenz je Domäne.
- Keine speziellen Synchronisationselemente notwendig.

**Nachteil:** Taktfrequenzen sind statisch.

### 3 Synchrone Kopplung

Kommunikation über:

- Zustände oder
- Ausgaben

Anordnung:

- parallel oder
- seriell

Flusskontrolle

## Kommunikation über Zustände

**Automat 1:**

$$\mathbf{Z}'_1 = \mathbf{Z}_1 \times \mathbf{Z}_2$$

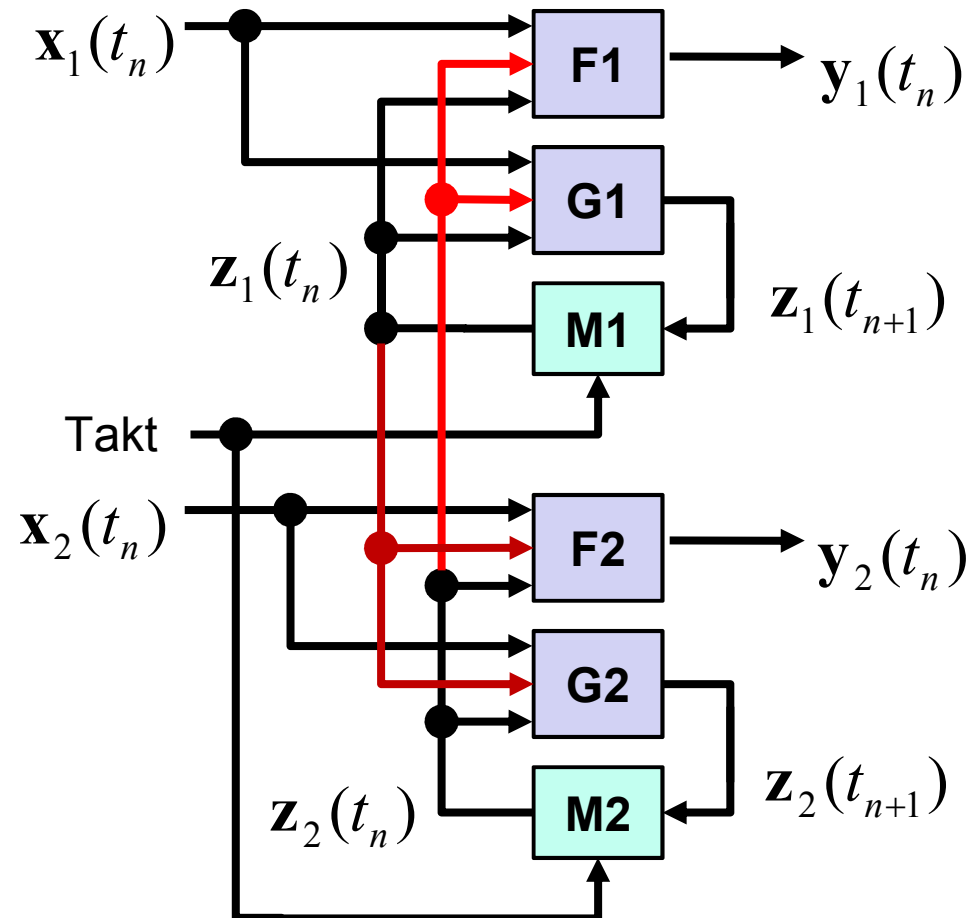
**Automat 2:**

$$\mathbf{Z}'_2 = \mathbf{Z}_2 \times \mathbf{Z}_1$$

$$\rightarrow \mathbf{Z} = \mathbf{Z}_2 \times \mathbf{Z}_1$$

**Anwendung:**

Theoretische Informatik,  
z.B. Produktautomat



## Kommunikation über Ausgaben (1)

**Automat 1:**

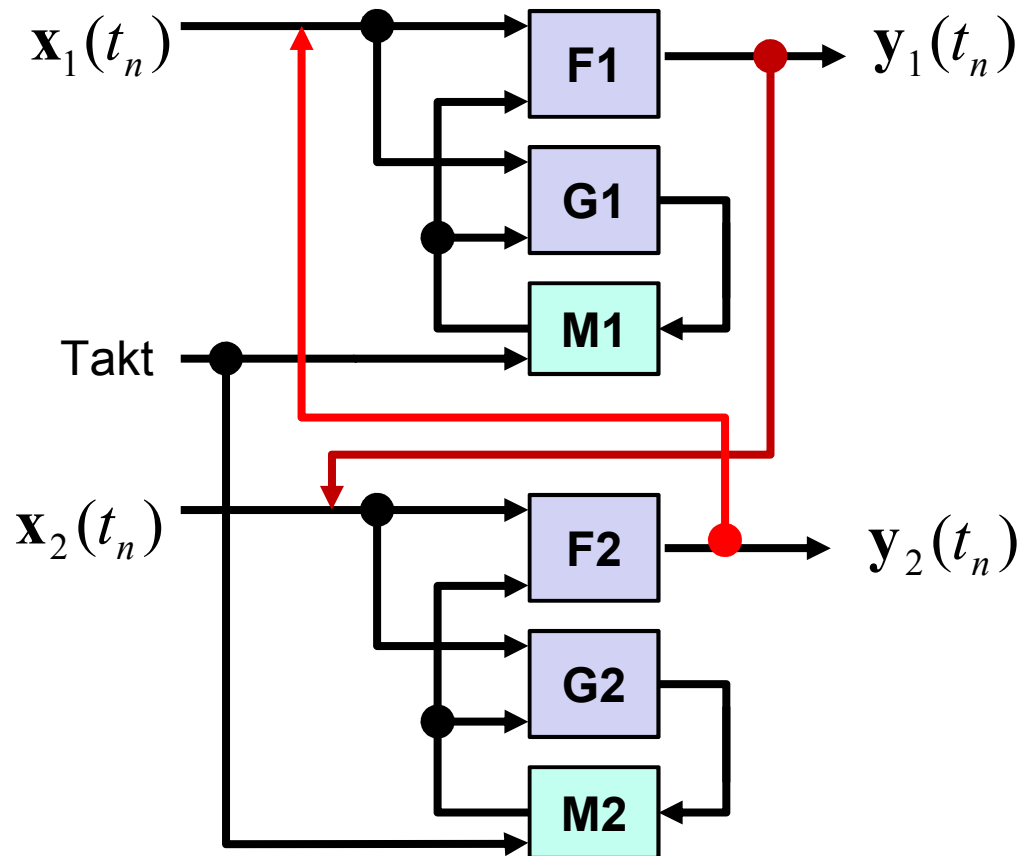
$$\mathbf{X}'_1 = \mathbf{X}_1 \times \mathbf{Y}_2$$

**Automat 2:**

$$\mathbf{X}'_2 = \mathbf{X}_2 \times \mathbf{Y}_1$$

**Anwendung:**

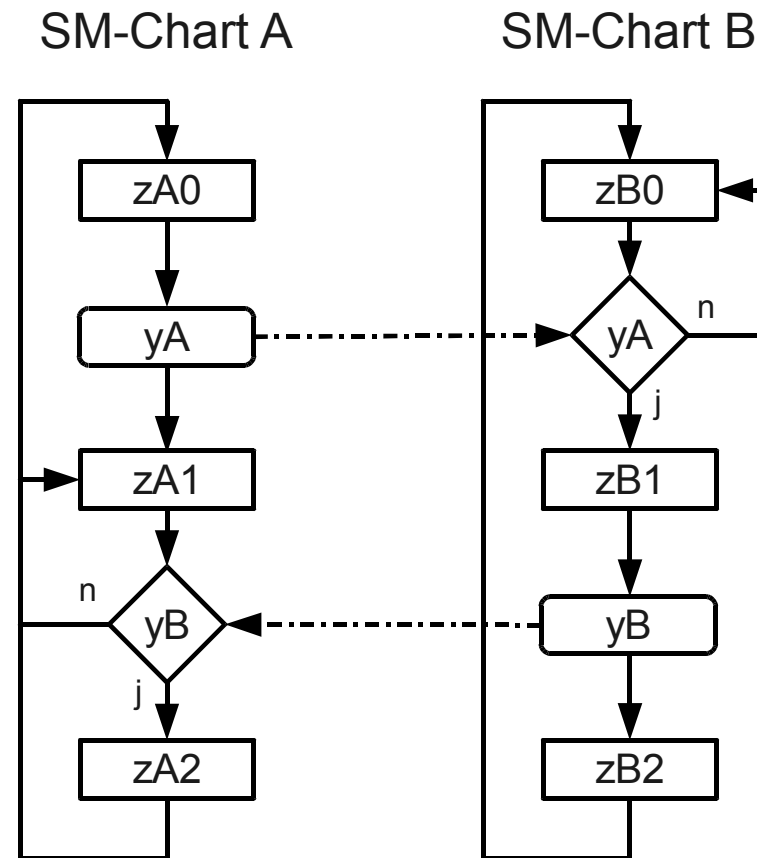
Technische Informatik,  
da aufgrund von Modularisierung  
kein Zugriff auf interne Zustände  
vorgesehen



## Kommunikation über Ausgaben (2)

### Darstellung im SM-Chart:

Bsp.: Kommunikation über die Ausgaben yA und yB





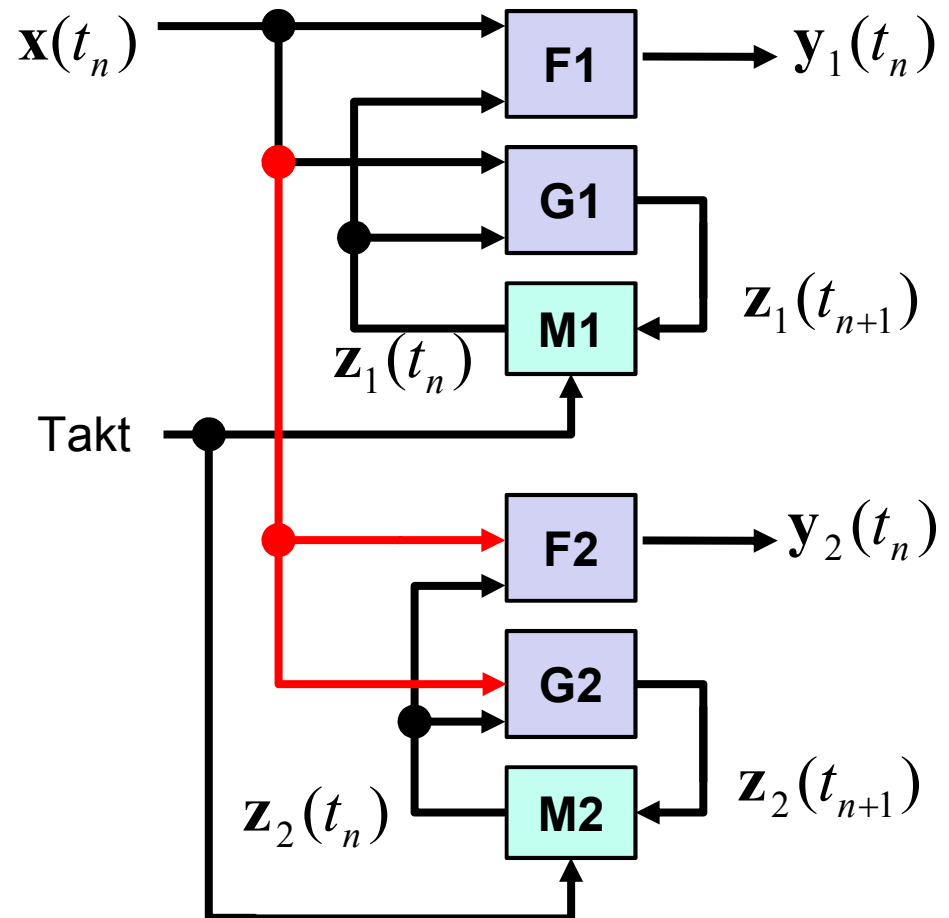
## Parallele Anordnung

**Automat 1 und 2:**

$$\mathbf{X} = \mathbf{X}_1 = \mathbf{X}_2$$

**Anwendung:**

z.B. Erkennung verschiedener  
Eingabefolgen



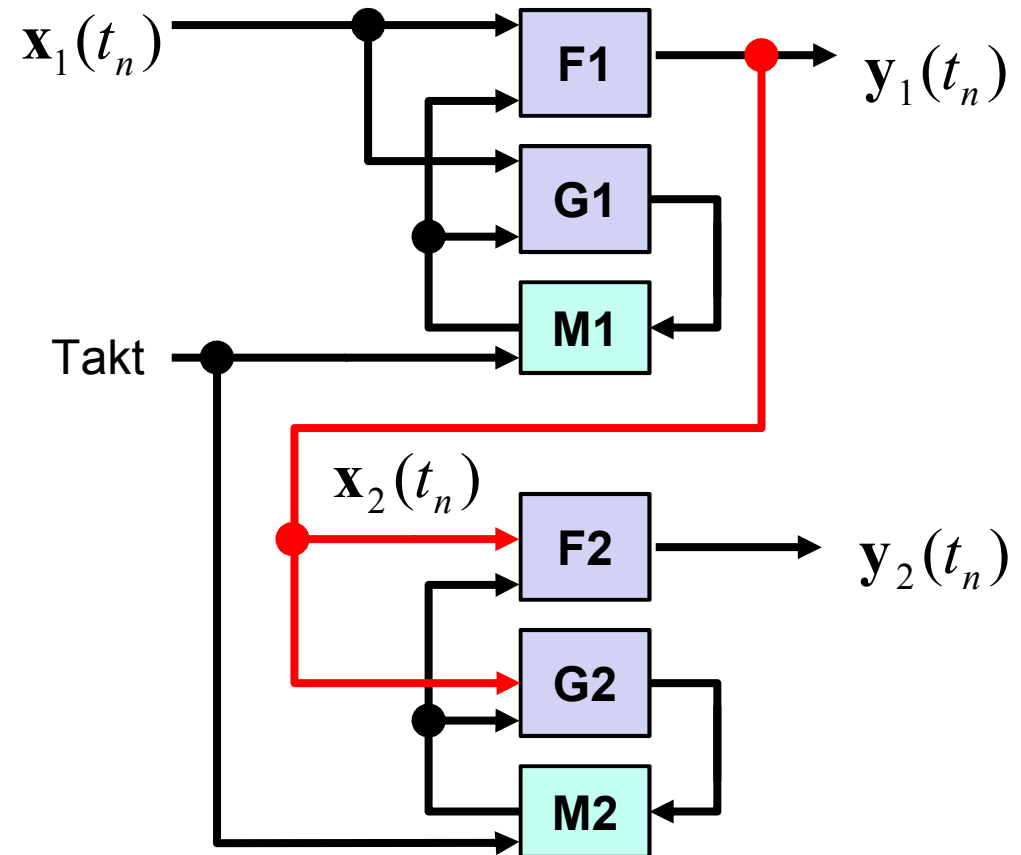
## Serielle Anordnung

### Automat 2:

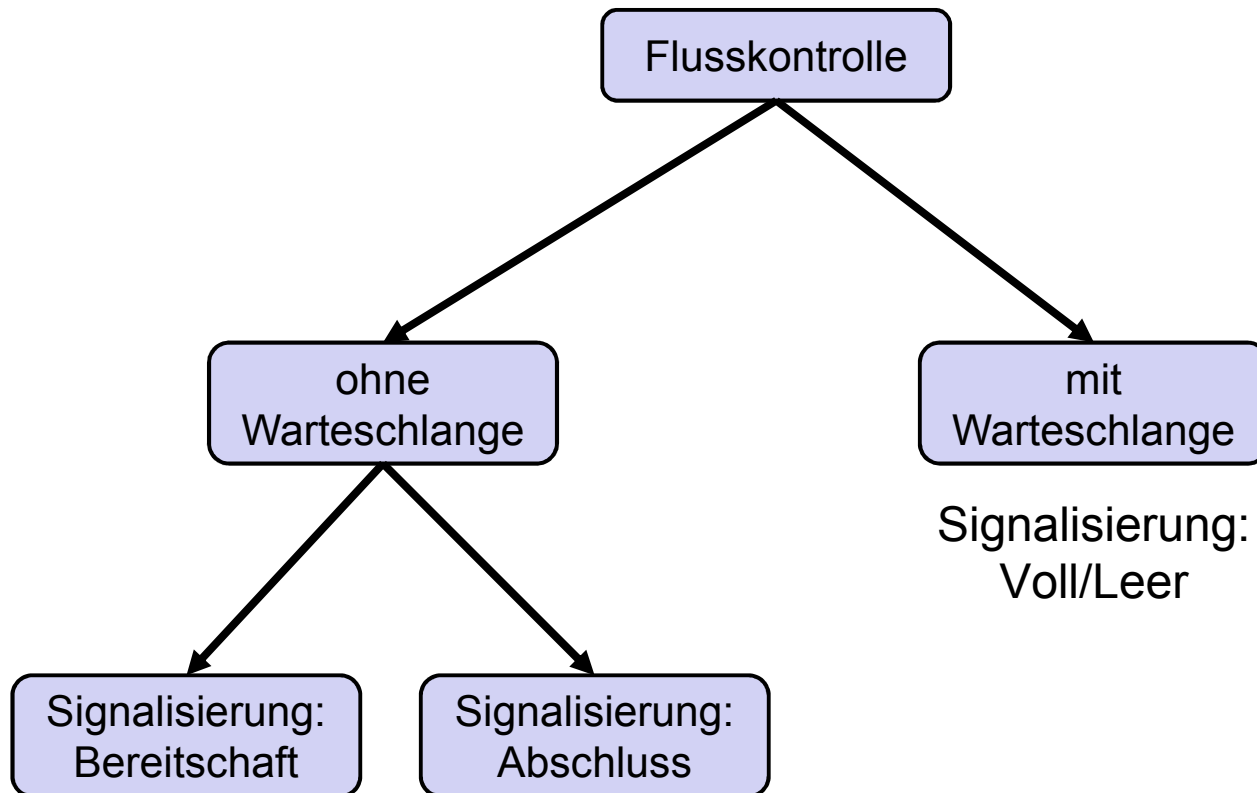
$$X_2 = Y_1$$

### Anwendung:

- Verarbeitung in mehreren Teilschritten, Pipeline
- häufig mehrere seriell angeordnete Kopplungen zu verschiedenen Komponenten



## Flusskontrolle



## Beispiel: Produktautomat

**Definition:**

$$\mathbf{A} = \mathbf{A}_1 \times \mathbf{A}_2 = (\mathbf{X}, \mathbf{Z}_1 \times \mathbf{Z}_2, \mathbf{Y}_1 \times \mathbf{Y}_2, (\mathbf{z}_1(t_0), \mathbf{z}_2(t_0)), \mathbf{E}, f, g)$$

Mit  $\mathbf{X} = \mathbf{X}_1 = \mathbf{X}_2$  ,  $\mathbf{Z} = \mathbf{Z}_2 \times \mathbf{Z}_1$

$$f((\mathbf{z}_1, \mathbf{z}_2), \mathbf{x}) = (f_1(\mathbf{z}_1, \mathbf{x}), f_2(\mathbf{z}_2, \mathbf{x}))$$

$$g((\mathbf{z}_1, \mathbf{z}_2), \mathbf{x}) = (g_1(\mathbf{z}_1, \mathbf{x}), g_2(\mathbf{z}_2, \mathbf{x}))$$

➔ Kopplung über Zustände, parallele Anordnung

Anwendungsbeispiele:

$\mathbf{E} = \mathbf{E}_1 \times \mathbf{E}_2$  : Schnitt zweier Sprachen

$\mathbf{E} = \mathbf{Z}_1 \times \mathbf{E}_2 \cup \mathbf{E}_1 \times \mathbf{Z}_2$  : Vereinigung zweier Sprachen

## Beispiel: Datenverarbeitung (1)

Automat 2 besitze 2 Schnittstellen

- separate Ein- und Ausgabealphabetete je Schnittstelle
- Zusammenfassung der Teilalphabetete:

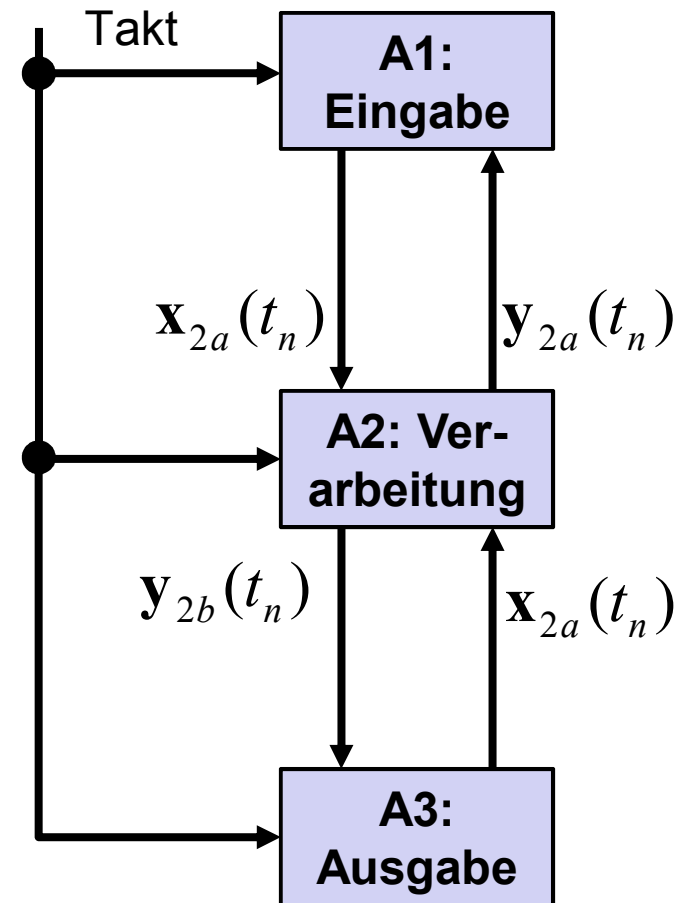
$$\mathbf{X}_2 = \mathbf{X}_{2a} \times \mathbf{X}_{2b}$$

$$\mathbf{Y}_2 = \mathbf{Y}_{2a} \times \mathbf{Y}_{2b}$$

- Serielle Kopplung über Ausgaben:

$$\mathbf{X}_{2a} = \mathbf{Y}_1 \quad \mathbf{X}_1 = \mathbf{Y}_{2a}$$

$$\mathbf{X}_{2b} = \mathbf{Y}_3 \quad \mathbf{X}_3 = \mathbf{Y}_{2b}$$

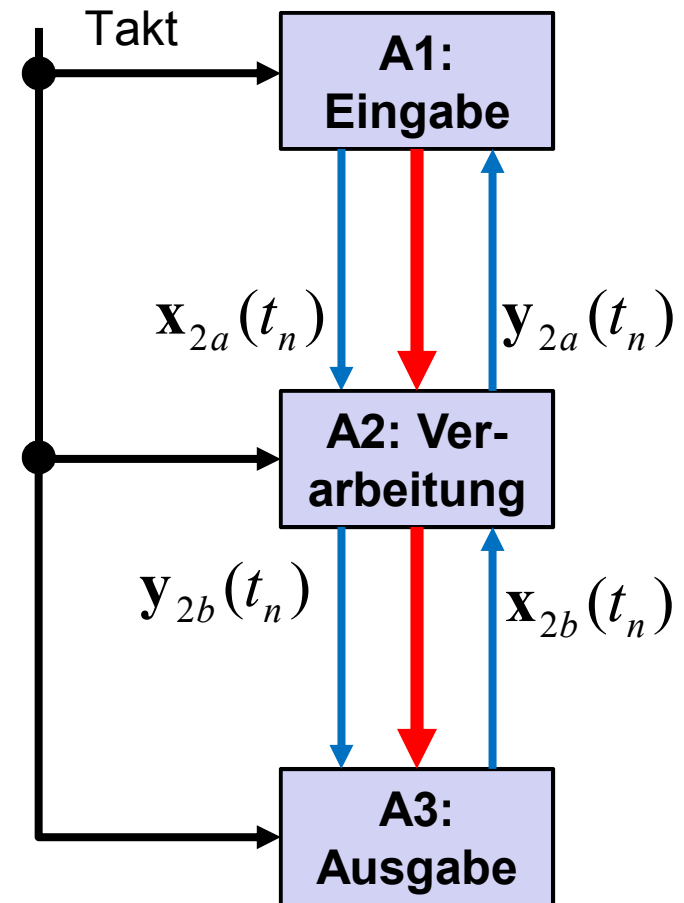


## Beispiel: Datenverarbeitung (2)

Automatenkopplung erfolgt primär über  
**Kontrollpfade** → Flusskontrolle

**Datenfluss** wird indirekt über Kontrollpfade  
gesteuert:

- Datenfluss nicht Bestandteil der Flusskontrolle.
- Ein- und Ausgabealphabete entsprechen den zu verarbeitenden Daten.

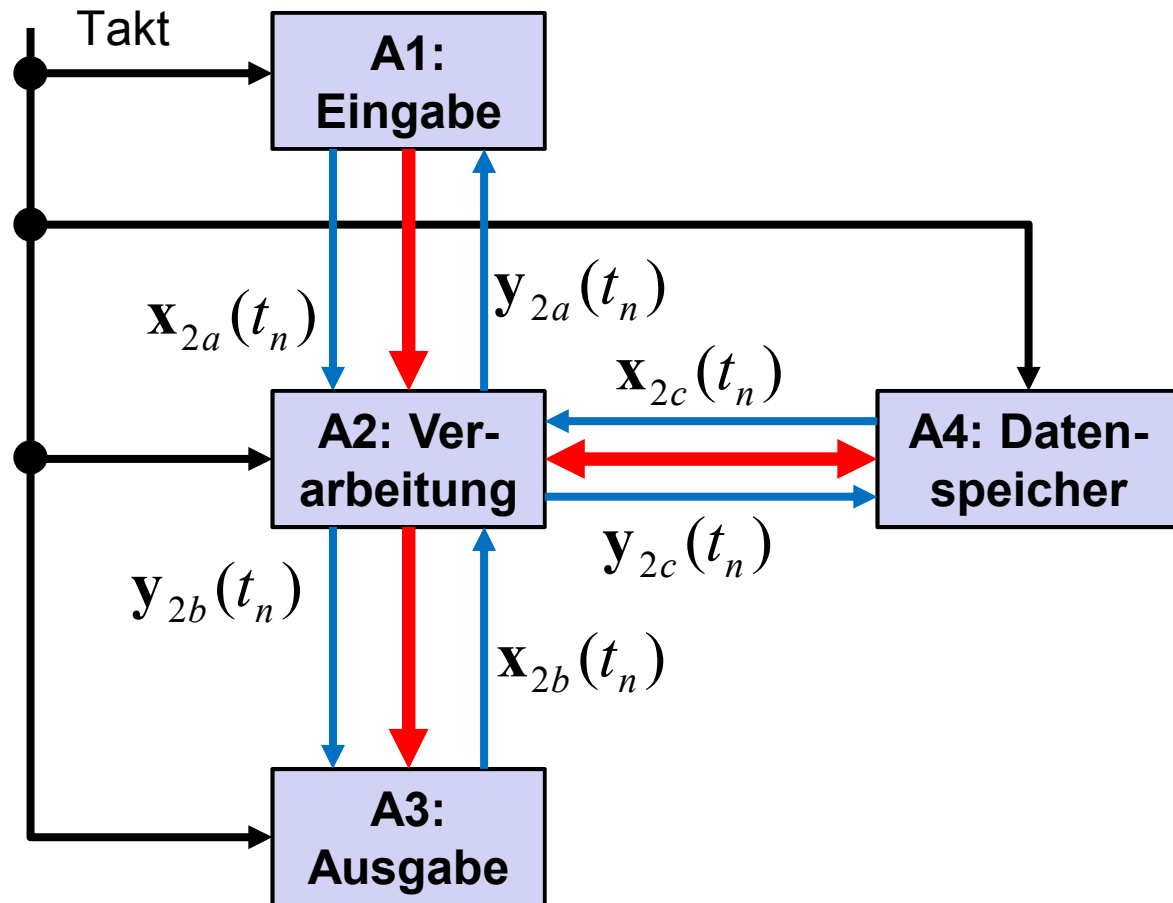


## Beispiel: Datenverarbeitung (3)

Verschiedene  
serielle  
Anordnungen  
denkbar:  
z.B. sternförmig

Kontrollpfade für  
Flusskontrolle

Datenfluss



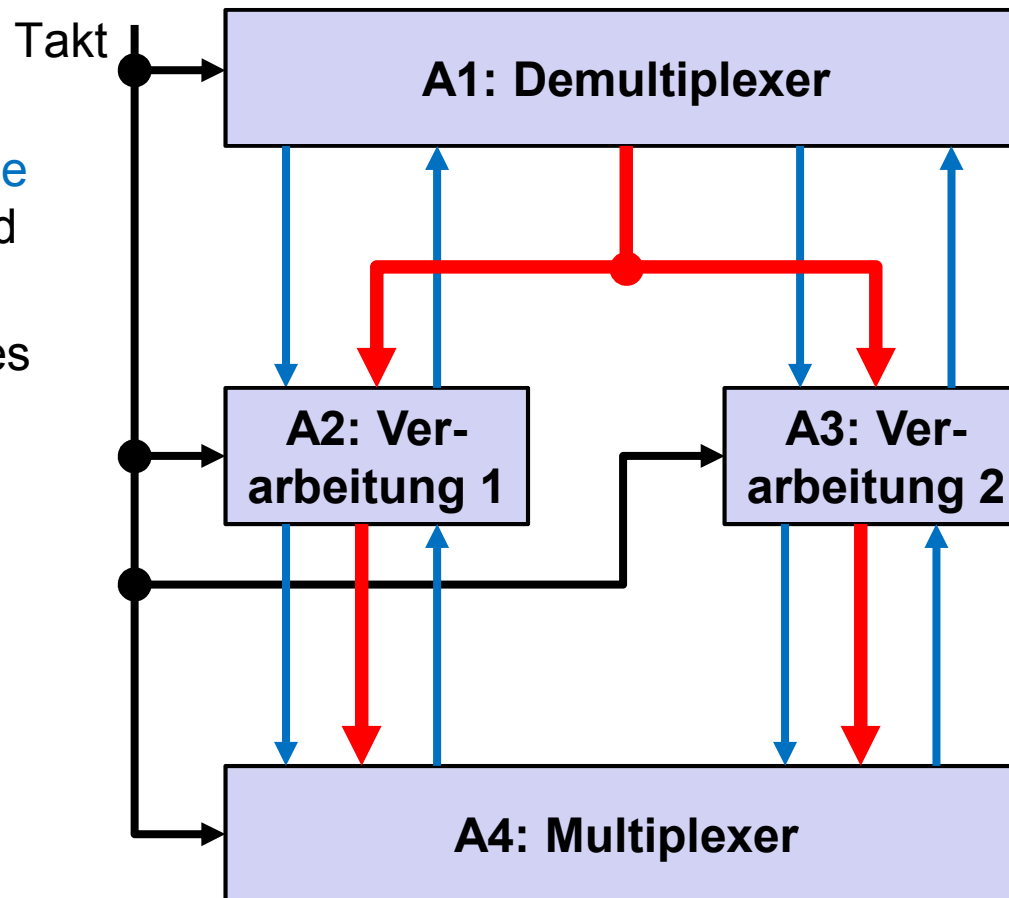
## Beispiel: De-/Multiplexer

### Demultiplexer:

- getrennte **Kontrollpfade** je Empfänger aufgrund Flusskontrolle
- parallele Verteilung des **Datenflusses**

### Multiplexer:

separate Schnittstelle für jeden Sender



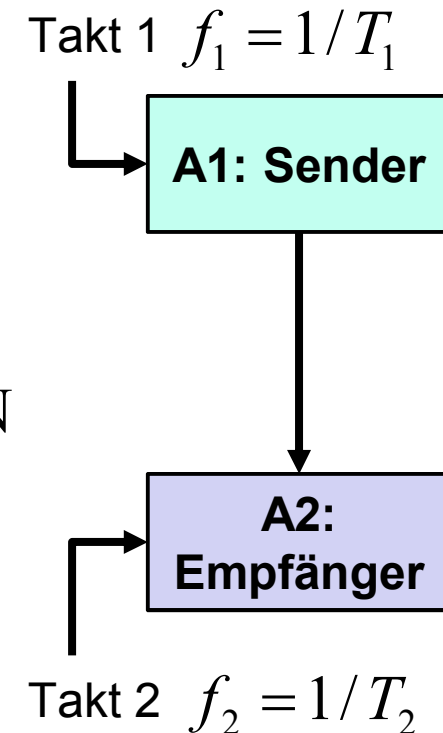


## 4 Asynchrone Kopplung

### Abtastproblem

Kommunikation zwischen verschiedenen Taktdomänen aufgrund verschiedener Taktfrequenzen und –phasen komplexer:

- keine feste Zuordnung zwischen zwei Zeitpunkten,  
allg. Annahme:  $t_n = n \cdot T_1 \neq m \cdot T_2 = t_m$  mit  $n, m \in \mathbb{N}$
- Abtastung von Signalvektoren kann zu Fehlern aufgrund verschiedener Signallaufzeiten führen  
Tafelbild

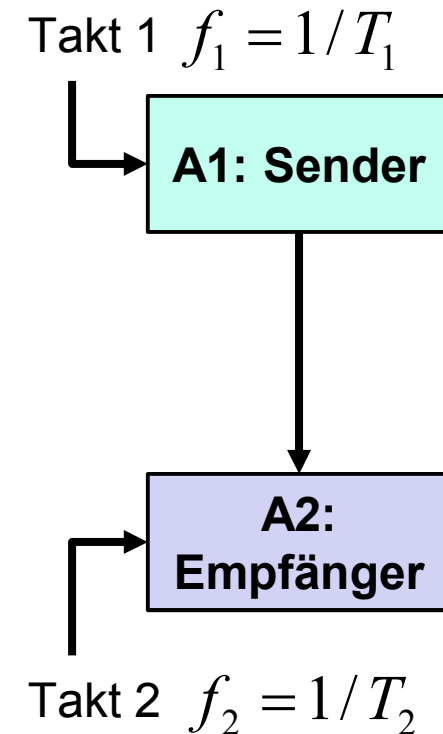


## Flusskontrolle

Berücksichtigung verschiedener Taktfrequenzen im Protokoll notwendig:

- Mehrfaches „Lesen“ desselben Ausgabezeichens, wenn  $f_2 > f_1$
- „Verpassen“ von Ausgabezeichen, wenn  $f_2 < f_1$

→ Explizite Übertragung von Datenwörtern notwendig. (Nachrichten, Flusskontrolle)



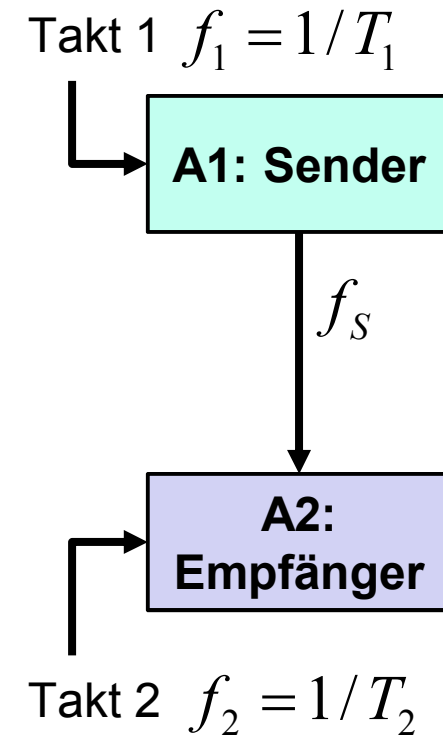
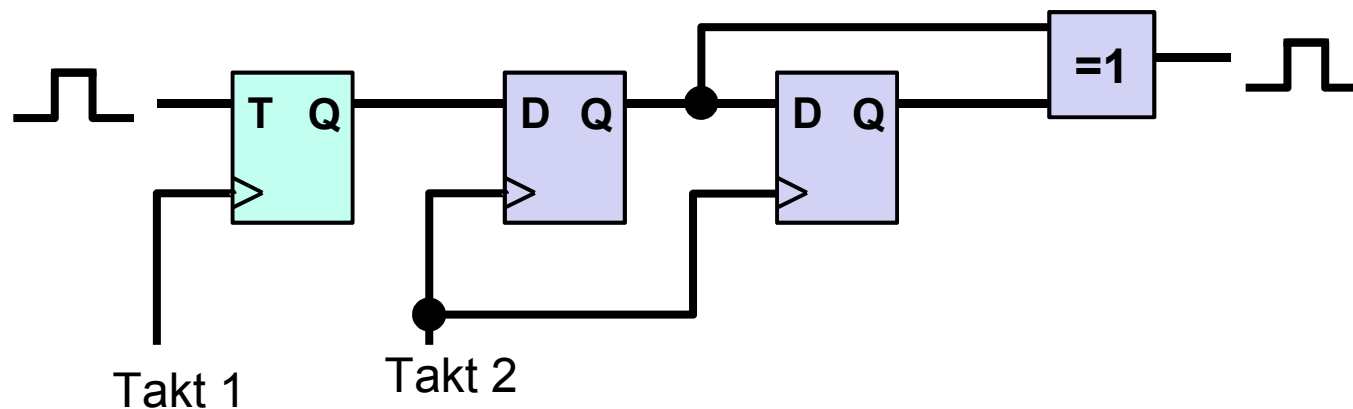
## Übertragung eines einzelnen Bits

**Abtastung** des gesendeten Signals gemäß  
Abtasttheorem mit  $f_2 > 2f_s$

$f_s$  = max. Änderungsfrequenz des Signals

→ minimale Impulsbreite  $t_{p,s} > 2 \cdot T_2$

**Lösung:** Übertragung von einzelnen (mindestens  $t_{p,s}$   
auseinanderliegenden) Impulsen mittels Signalwechseln



## Übertragung eines Signalvektors

Mehrere Varianten möglich:

1. Gray-Code
2. Steuerung mittels einzelner, separater Bits
3. Warteschlange mittels Cross-Clock-FIFO

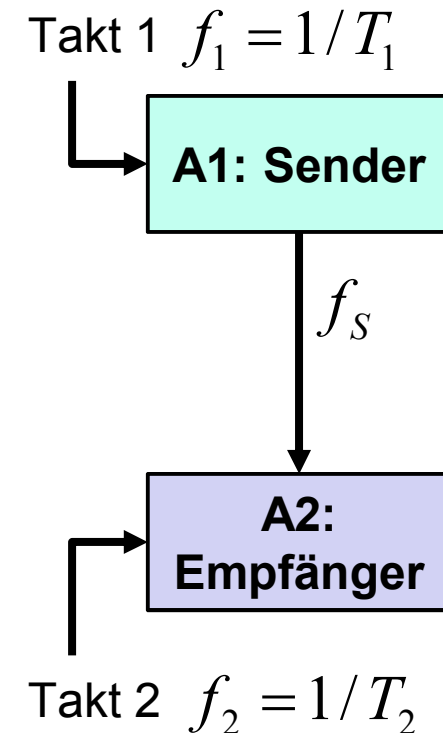
## Variante 1: Gray-Code (1)

### Bedingungen:

- Hamming-Abstand zwischen zwei aufeinanderfolgenden Ausgabezeichen ist kleiner gleich 1.
- Variation der Signallaufzeit  $t_{skew} < T_2$

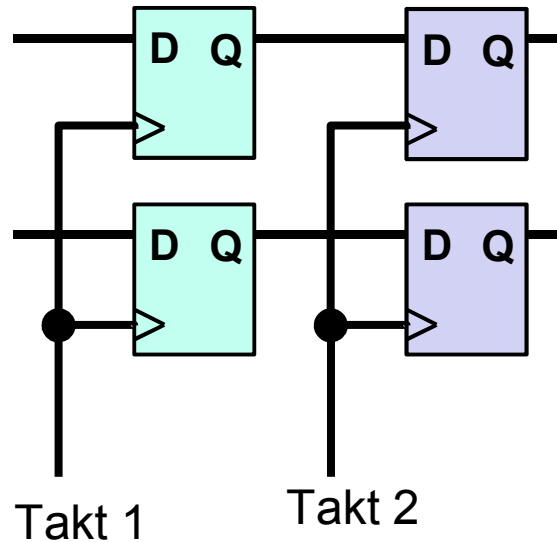
### Erfüllt durch:

- Aufeinanderfolgende Zeichen des Gray-Codes
  - Ausgabe aus Registern damit  $t_{skew}$  klein
- Entweder Abtastung des alten oder des neuen Wertes.

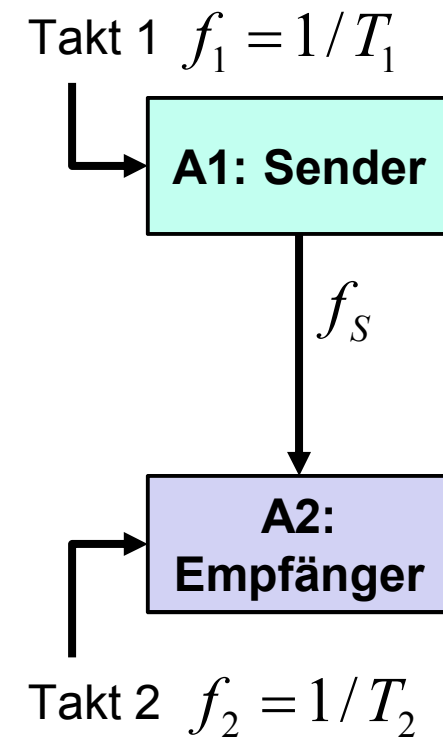


## Variante 1: Gray-Code (2)

Schaltung:



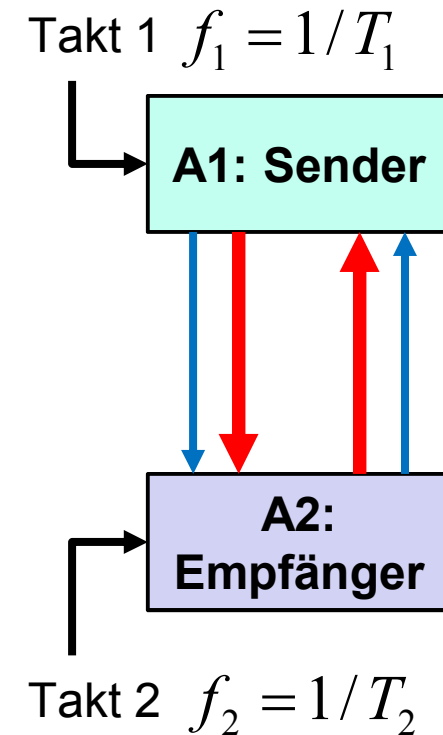
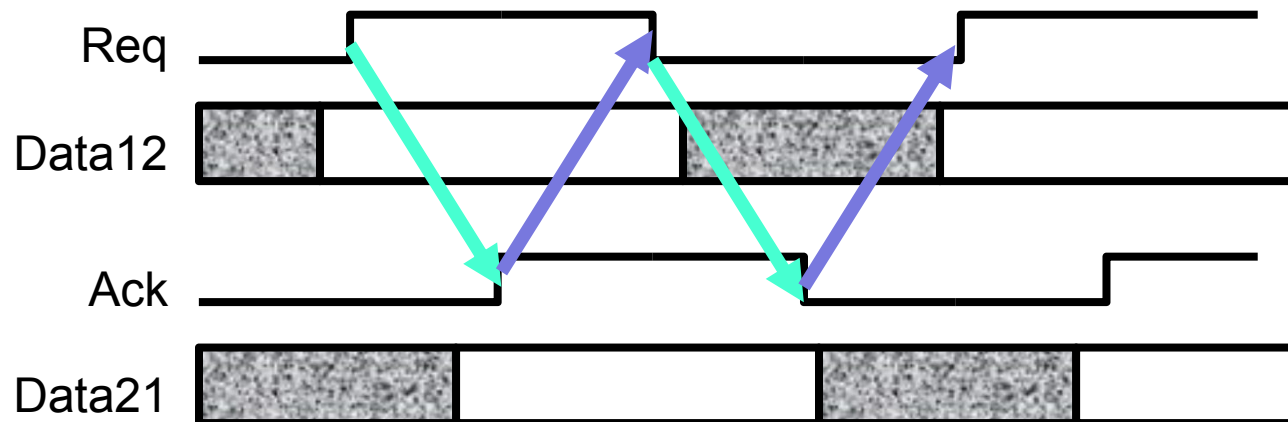
Signalverlauf:



## Variante 2: Steuerung mittels einzelner Bits

**Prinzip:** synchrone Kommunikation mittels

- **Kontrollfluss** bestehend aus jeweils 1 Bit:
  - Sender → Empfänger: Request (Req)
  - Empfänger → Sender: Acknowledge (Ack)
- **Datenbus** bestehend aus mehreren Bits (je Richtung)
- Datenworte bleiben während Übertragung konstant.

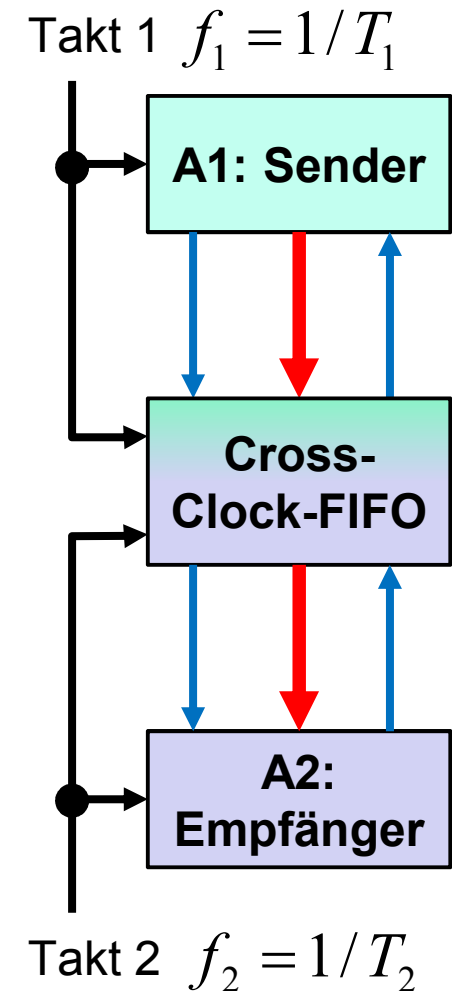


## Variante 3: FIFO (1)

**Nachteil Variante 2:** Niedrige Datenrate

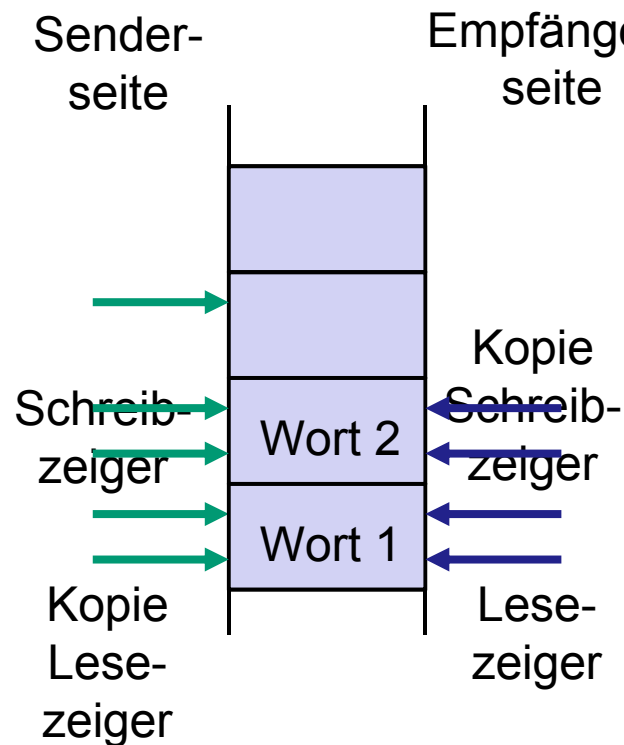
**Lösung:** Warteschlange mit Ringspeicher

- Schreiben von 1 Datenwort pro Takt ( $T_1$ ), solange FIFO nicht voll.
- Lesen von 1 Datenwort pro Takt ( $T_2$ ), solange FIFO nicht leer.
- Lesefreigabe von geschriebenen Datenwörtern erst nachdem diese vollständig in den Ringspeicher geschrieben wurden.
- Kontinuierliche Datenübertragung bei  $f_1 = f_2$  möglich (unabhängig von Phasenlage).





## Variante 3: FIFO (2)



### Eigenschaften:

- Vergleich von Zeigern jeweils innerhalb einer Taktdomäne → Zeigerkopien.
- Übertragung der Zeiger (Signalvektoren) mittels Gray-Code.
- Durch Kopie der Zeiger und anschließendem Vergleich → Latenz  
→ Aktualisierung der Speicherzelle vor dem Lesen abgeschlossen.
- Gleichzeitiges Schreiben und Lesen (verschiedener) Wörter möglich.

## 5 Initialisierung

### Reset vs. Power-Up

#### **Nicht programmierbare Schaltkreise:**

- Reset notwendig für Initialisierung der Zustandsregister.
- Datenregister können von Automaten initialisiert werden.

#### **Programmierbare Schaltkreise:**

- Initiale Registerbelegung wird durch Programmierung festgelegt.
- Reset-Eingang ist daher optional.

#### **→ Wiederverwendungsgerechter Entwurf:**

- Reset-Eingang vorsehen.
- (Zustands-)Register bei Power-Up und Reset gleichermaßen belegen.

## Synchrones Reset (1)

### Vorteile:

- Synthese immer möglich: Set/Reset als zusätzliche Variable in FF-Ansteuergleichungen, sofern nicht explizit vorhanden.
- Logik-Zusammenfassung möglich.
- Überprüfung in Timing-Analyse.

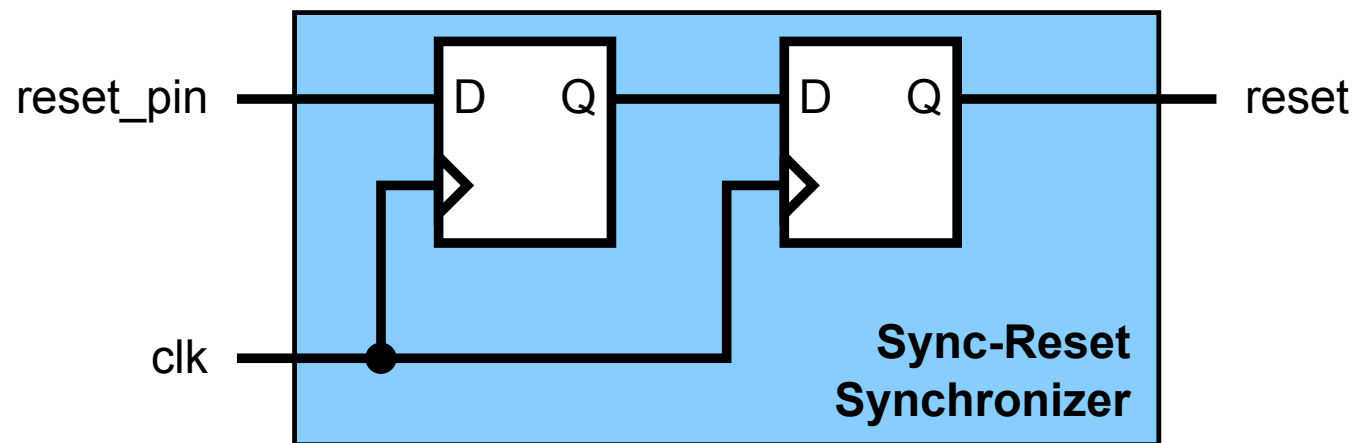
### Nachteil:

Free-Running Clock benötigt, damit Reset auch ausgelöst wird.

## Synchrones Reset (2)

### Reset-Synchronizer:

- Reset-Pin muss synchronisiert werden, damit alle Register in der gleichen Taktperiode zurückgesetzt werden.
- Synchronisation mit (min.) 2 FFs, um Metastabilitäten zu vermeiden.
- Reset-Pin ist ggf. zu negieren
- Schaltung allg. verwendbar für Synchronisation asynchroner Signale.



## Asynchrones Reset (1)

### Vorteile:

- Keine Free-Running Clock benötigt um Reset auszulösen, u.U. notwendig für Register, die Schaltungsausgänge treiben.
- Separater FF-Eingang, damit kein Einfluss auf Timing.

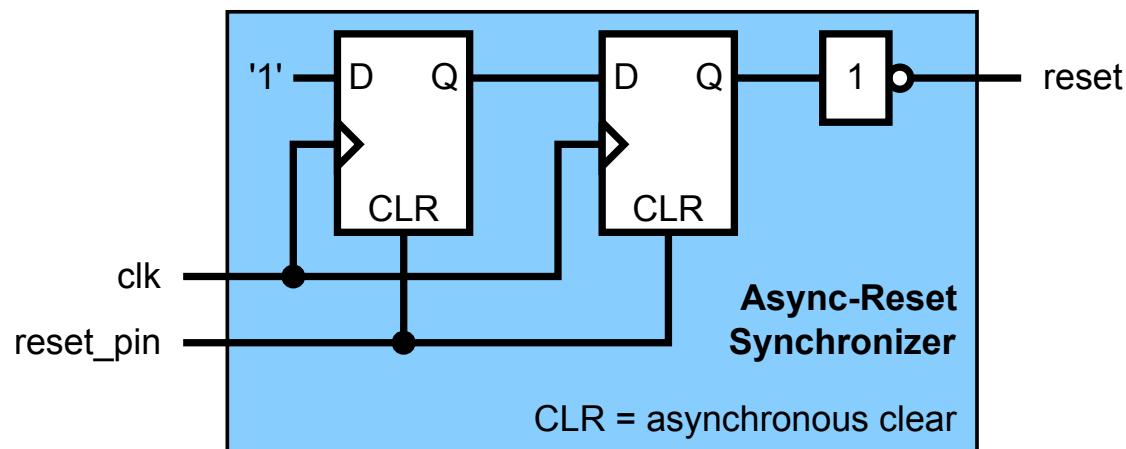
### Nachteile:

- Implementierung erfordert globale Verdrahtungsressourcen wie Taktsignale.
- Timing-Analyse problematisch und häufig standardmäßig ignoriert.
- Reset am Register darf nicht in zeitlicher Nähe zur Taktflanke losgelassen werden → sonst Metastabilitäten.

## Asynchrones Reset (2)

### Reset-Synchronizer:

- Reset muss synchron losgelassen werden, damit alle Register in der gleichen Taktperiode wieder in „Betrieb“ gehen.
- Reset-Pin ist ggf. zu negieren
- Reset-Tree sorgt für zusätzliche, notwendige Verzögerung.



---

## Coding Guidelines für Reset

### Empfehlungen:

- Synchrones Reset.
- Reset-Eingang setzt nur Zustandsregister zurück  
→ Fan-Out von Reset-Eingang klein.
- Power-Up-Wert automatisch ermitteln lassen, damit identisch zu Reset-Wert.

## 6 Zusammenfassung

### Automatendarstellung:

- Zustandsgraphen → häufig unvollständig.
- SM-Charts → korrekte Modellierung hierarchischer Entscheidungen.
- SFC und GRAFCET → abgeleitet von Petri-Netzen.

### Automatenkopplung:

- Synchron vs. asynchron gekoppelte Automaten.
- Parallel vs. serielle Anordnung.
- Kommunikation über Zustände oder Ausgaben.

### Initialisierung:

- Synchrones Reset verwenden.
- Reset-Synchronizer notwendig.