

# Technische Grundlagen der Informatik – Kapitel 3



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Prof. Dr. Andreas Koch  
Fachbereich Informatik  
TU Darmstadt



# Kapitel 3: Themen

---



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Einleitung
- Latches und Flip-Flops
- Entwurf synchroner Logik
- Endliche Zustandsautomaten
- Zeitverhalten sequentieller Logik
- Parallelismus

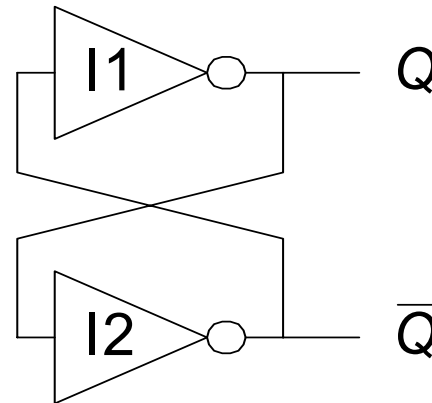
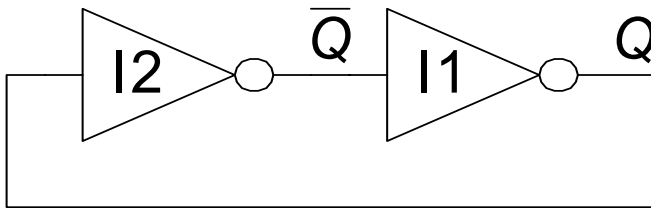
- Ausgänge **sequentieller** Logik hängen ab von
  - **aktuellen** Eingabewerten
  - **vorherigen** Eingabewerten
- Schaltung **speichert** einen internen **Zustand**
- Definitionen
  - **Zustand**: interne Informationen, aus denen weiteres Schaltungsverhalten hergeleitet werden kann
  - **Latches und Flip-Flops**: Speicherelemente für jeweils 1 Bit Zustand
  - **Synchrone sequentielle Schaltung**: Kombinatorische Logik gefolgt von Flip-Flops

- Können **Folgen** von Ereignissen bearbeiten
- Haben “**Gedächtnis**” (in der Regel nur Kurzzeit-)
- Benutzen **Rückkopplungen** von Logikausgängen zu Logikeingänge, um Informationen zu speichern
  - Rückkopplungen: **Keine** kombinatorischen Schaltungen mehr!

- Der **Zustand** einer Schaltung beeinflusst das **zukünftige** Verhalten
- **Speicherelemente** speichern Zustand
  - Bistabile Schaltungen
  - SR Latch
  - D Latch
  - D Flip-Flop
  - Manchmal auch **Zustandselemente** genannt

# Bistabile Grundschaltung

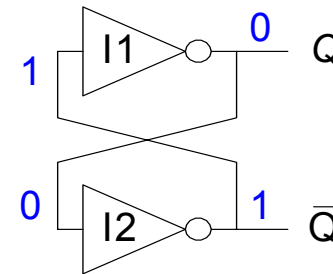
- Fundamentaler Baustein der anderen Speicherelemente
- **Zwei** Ausgänge:  $Q$ ,  $\overline{Q}$
- **Keine** Eingänge



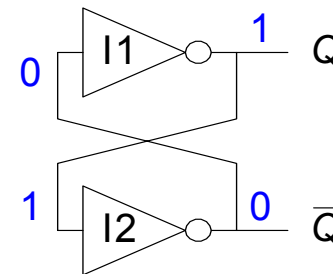
# Analyse der bistabilen Grundschaltung

- Betrachte zwei Möglichkeiten:

- $Q = 0$ : dann  $\bar{Q} = 1$  und  $Q = 0$
- Konsistent und stabil



- $Q = 1$ : dann  $\bar{Q} = 0$  und  $Q = 1$
- Konsistent und stabil

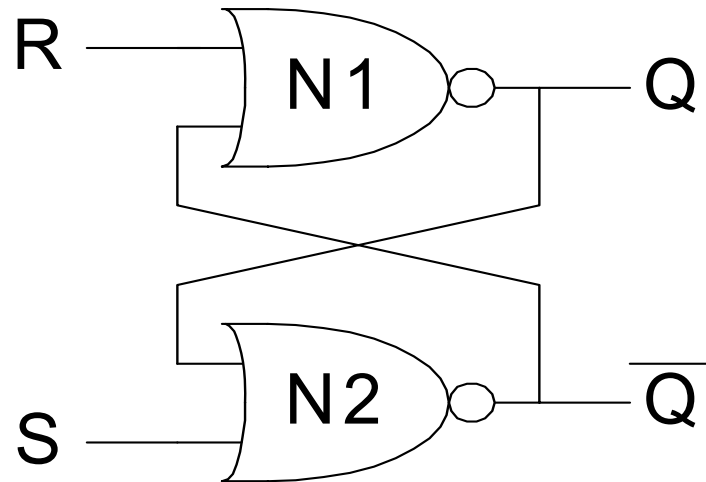


- Bistabile Schaltung speichert 1 Zustandsbit in Zustandsvariable Q (oder  $\bar{Q}$ )
- Es gibt aber bisher **keine Eingänge, um diesen Zustand zu beeinflussen**

# SR (Setzen/Rücksetzen) Latch



- SR Latch



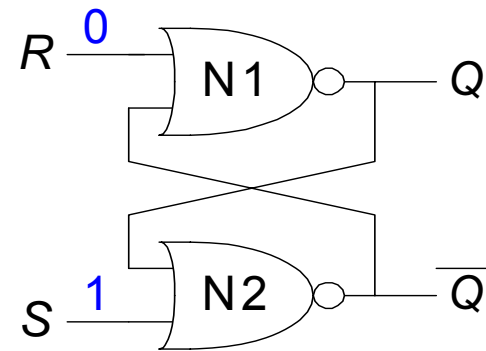
- Betrachte Fälle:

- $S = 1, R = 0$
- $S = 0, R = 1$
- $S = 0, R = 0$
- $S = 1, R = 1$

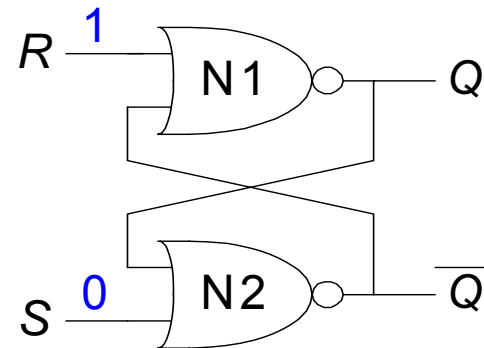


# Analyse des SR Latches

- $S = 1, R = 0$ : dann  $Q = 1$  und  $\overline{Q} = 0$

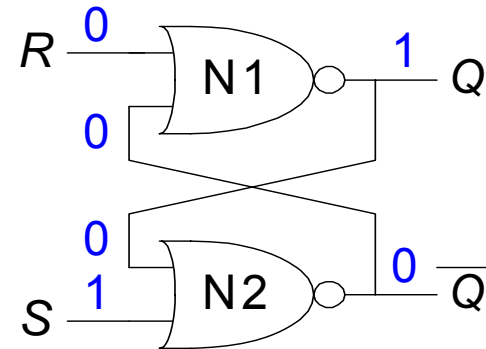


- $S = 0, R = 1$ : dann  $Q = 0$  und  $\overline{Q} = 1$

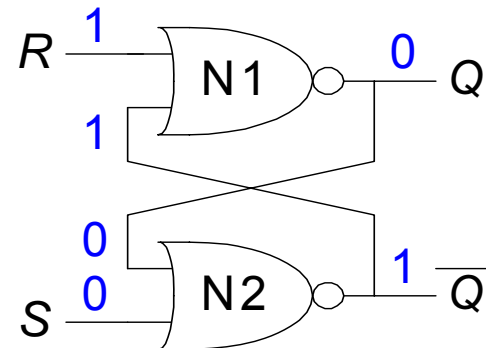


# Analyse des SR Latches

- $S = 1, R = 0$ : dann  $Q = 1$  und  $\overline{Q} = 0$



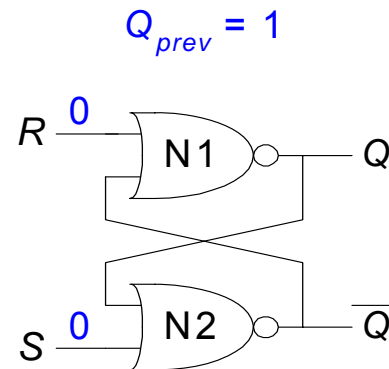
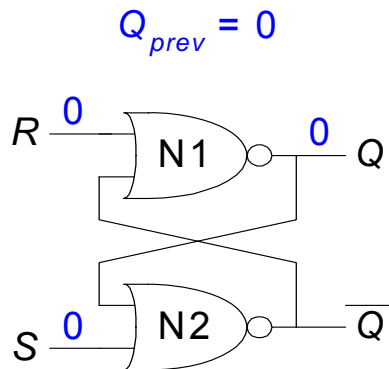
- $S = 0, R = 1$ : dann  $Q = 0$  und  $\overline{Q} = 1$



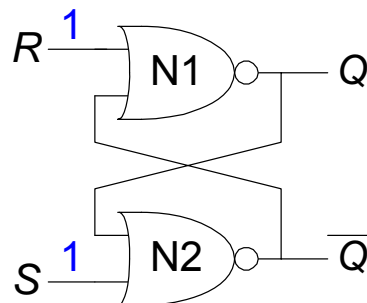
# Analyse des SR Latches



- $S = 0, R = 0$ : dann  $Q = Q_{prev}$



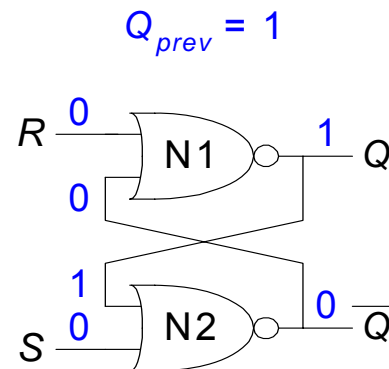
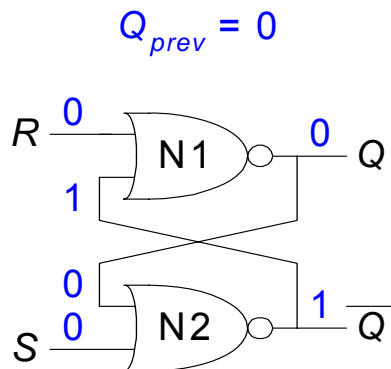
- $S = 1, R = 1$ : dann  $Q = 0$  und  $\overline{Q} = 0$



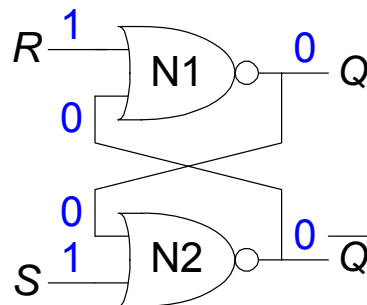
# Analyse des SR Latches



- $S = 0, R = 0$ : dann  $Q = Q_{prev}$  und  $\bar{Q} = \bar{Q}_{prev}$  (**gespeichert!**)



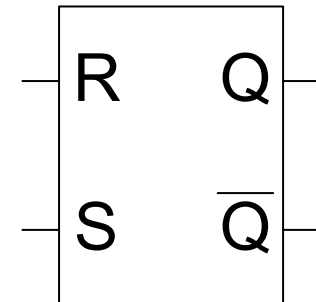
- $S = 1, R = 1$ : dann  $Q = 0$  und  $\bar{Q} = 0$  (**ungültiger Zustand**:  $Q \neq \text{NOT } \bar{Q}$ )



# Schaltplansymbol für SR Latch

- SR steht für Setzen/Rücksetzen Latch (*set/reset*)
  - Speichert ein Bit Zustand ( $Q$ )
- Festlegen des gespeicherten Wertes mit den  $S$ ,  $R$  Eingängen
  - **Set:** Setze Ausgang auf 1 ( $S = 1$ ,  $R = 0$ ,  $Q = 1$ )
  - **Reset:** Zurücksetzen des Ausgangs auf 0 ( $S = 0$ ,  $R = 1$ ,  $Q = 0$ )
- **Illegalen Zustand vermeiden**
  - Es darf niemals  $S = R = 1$  sein

## SR Latch Symbol

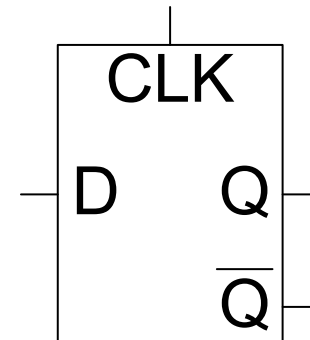


# D Latch

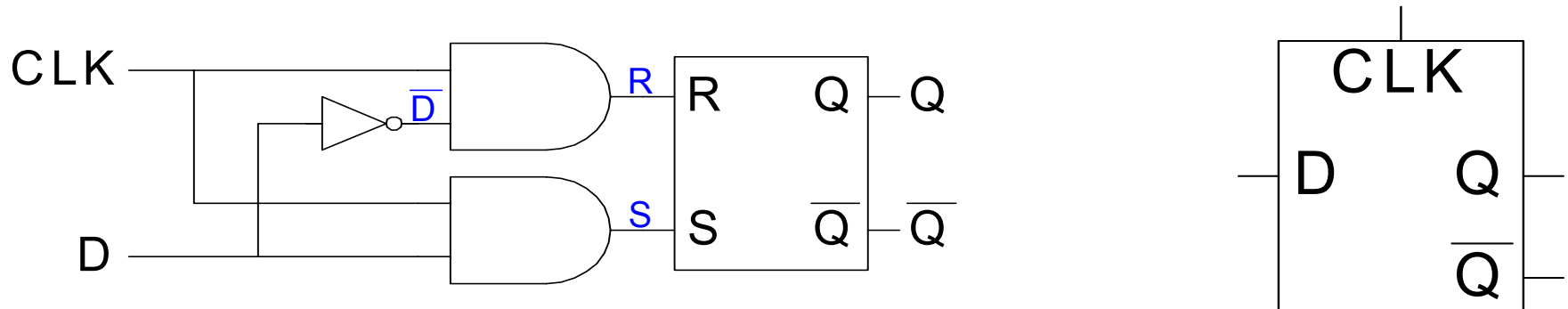


- Zwei Eingänge:  $CLK$ ,  $D$ 
  - $CLK$ : steuert, *wann* sich der Ausgang ändert (*clock*, Taktsignal)
  - $D$  (der Dateneingang): steuert, auf *was* sich der Eingang ändert
- Funktion
  - Wenn  $CLK = 1$  wird  $D$  *weitergereicht* an  $Q$  (das Latch ist transparent)
  - Wenn  $CLK = 0$  *behält*  $Q$  seinen vorigen Wert (das Latch ist *opak*)
- Illegal Fall  $Q \neq \text{NOT } \overline{Q}$  kann *nicht* mehr auftreten

## D Latch Symbol

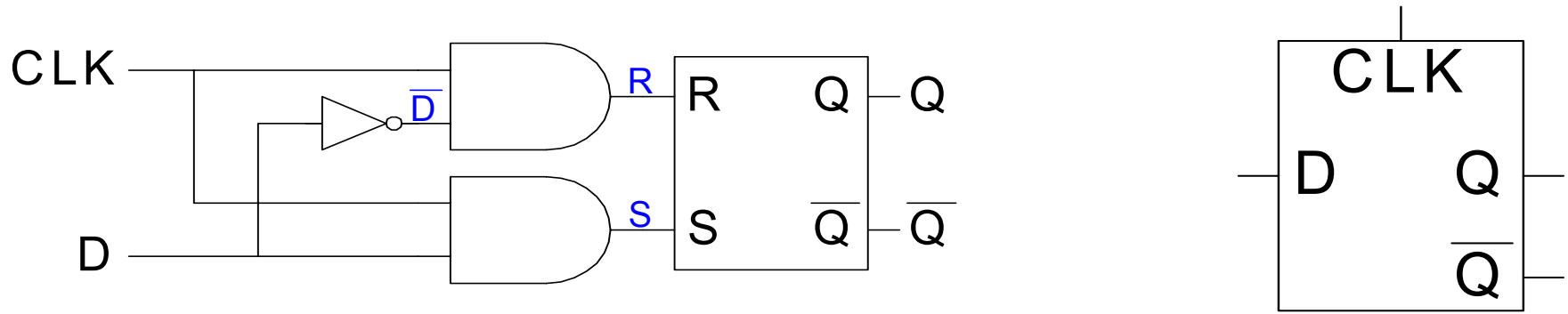


# Interner Aufbau eines D Latches



<i>CLK</i>	<i>D</i>	$\overline{D}$	<i>S</i>	<i>R</i>	<i>Q</i>	$\overline{Q}$
0	X					
1	0					
1	1					

# Interner Aufbau eines D Latches



$CLK$	$D$	$\overline{D}$	$S$	$R$	$Q$	$\overline{Q}$
0	X	$\overline{X}$	0	0	$Q_{prev}$	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

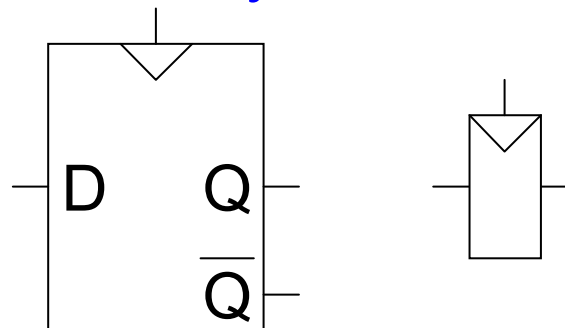


# D Flip-Flop



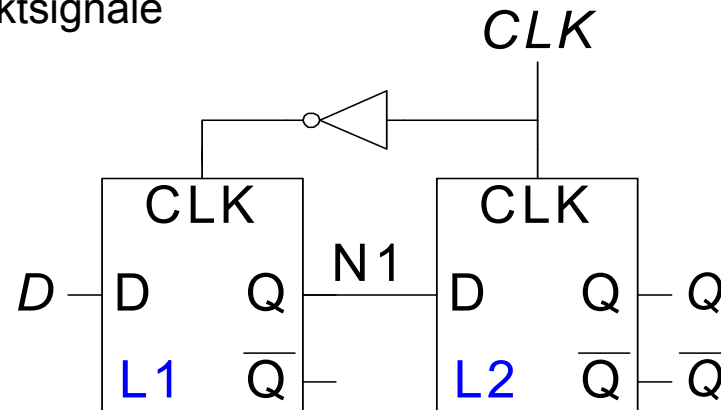
- Zwei Eingänge: *CLK*, *D*
- *Funktion*
  - Das Flip-Flop liest den **aktuellen** Wert von *D* bei einer **steigenden** Flanke von *CLK*
    - Wenn *CLK* von 0 nach 1 steigt, wird *D* **weitergegeben** zu *Q*
    - Sonst **behält** *Q* seinen **vorigen** Wert
  - *Q* ändert sich also nur bei einer **steigenden** Flanke von *CLK*
- Flip-Flop ist **flankengesteuert** (*edge-triggered*)
  - Wird bei Flanke des Taktsignals aktiviert

## D Flip-Flop Symbole

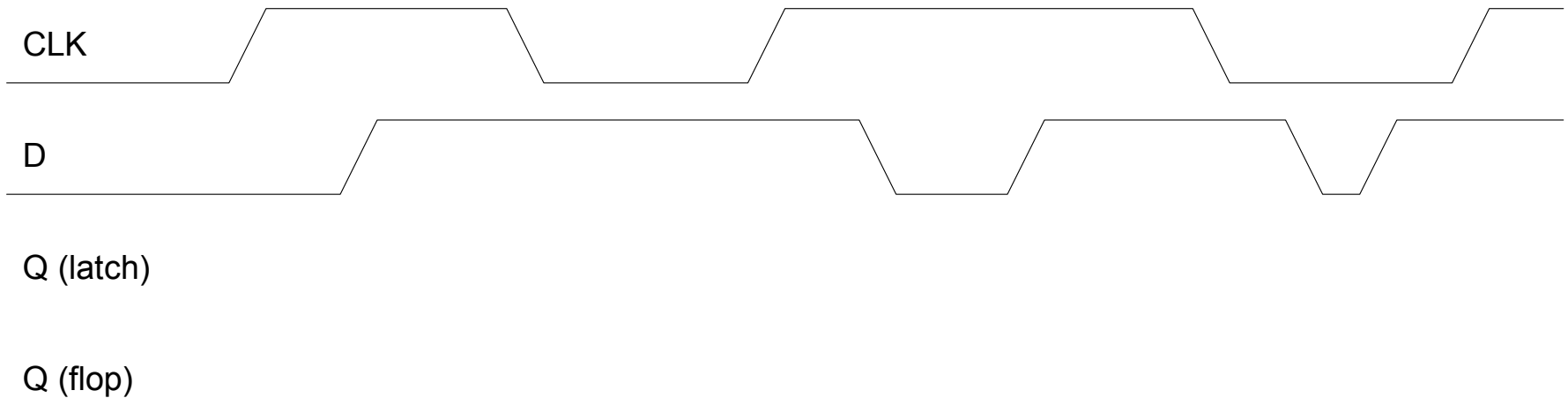
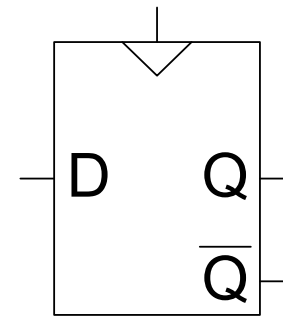
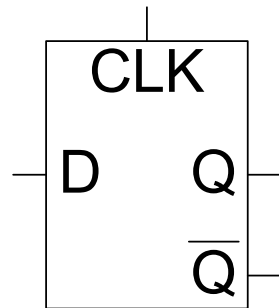


# Interner Aufbau eines D Flip-Flops

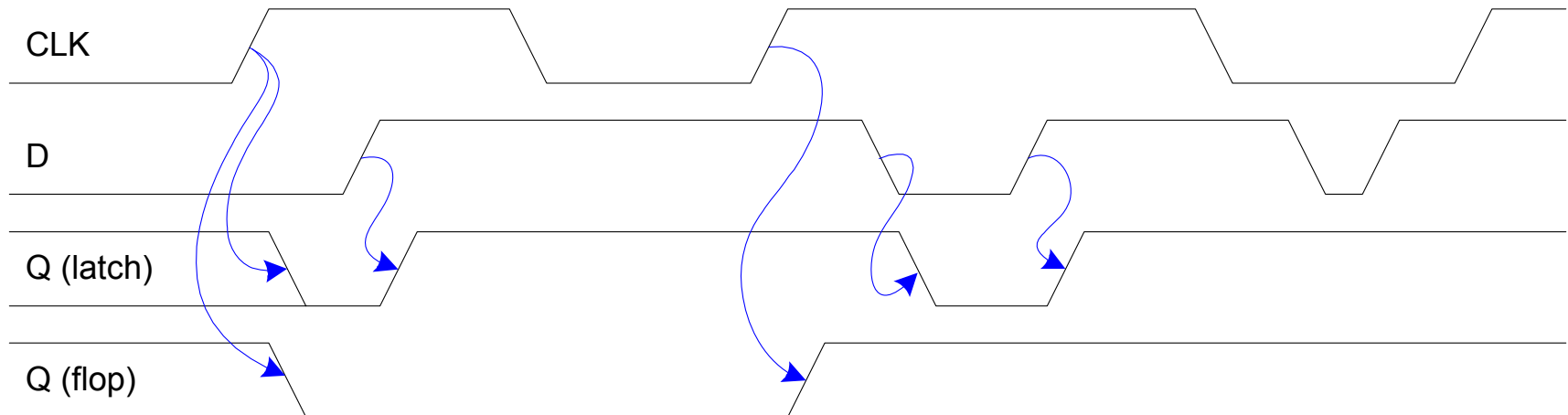
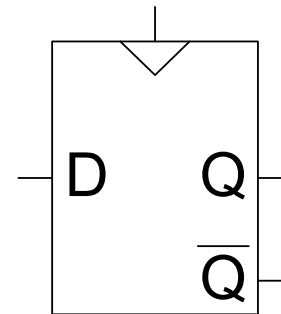
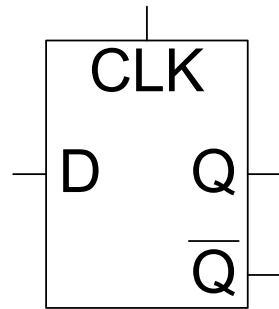
- Zwei **Latches** in Serie (L1 und L2)
  - ... gesteuert durch **komplementäre** Taktsignale
- Wenn **CLK = 0**
  - ... ist L1 transparent
  - ... ist L2 opak
  - *D* wird bis N1 weitergegeben
- Wenn **CLK = 1**
  - ... ist L2 transparent
  - ... ist L1 opak
  - N1 wird an Q weitergegeben
- Bei **steigender** Flanke von CLK (Wechsel von 0 → 1)
  - *D* wird an Q weitergegeben



# Vergleich D Latch mit D Flip-Flop



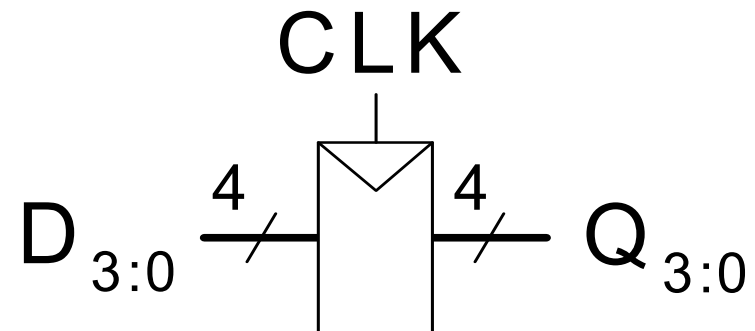
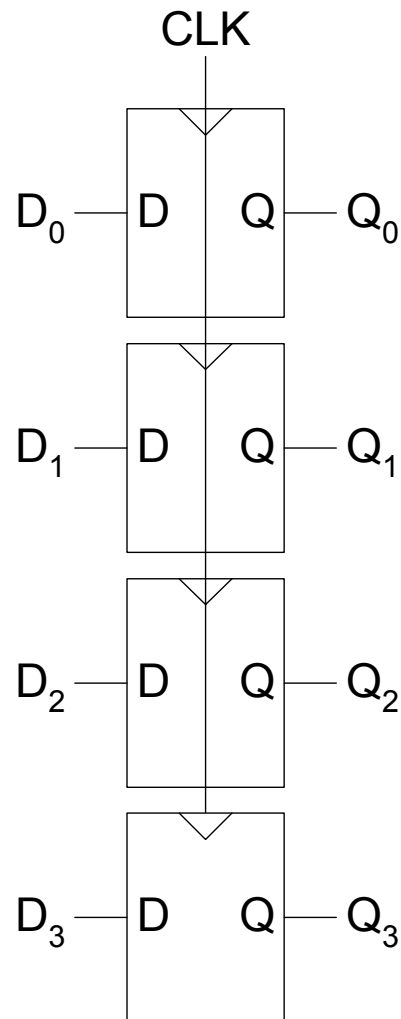
# Vergleich D Latch mit D Flip-Flop



# Register



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



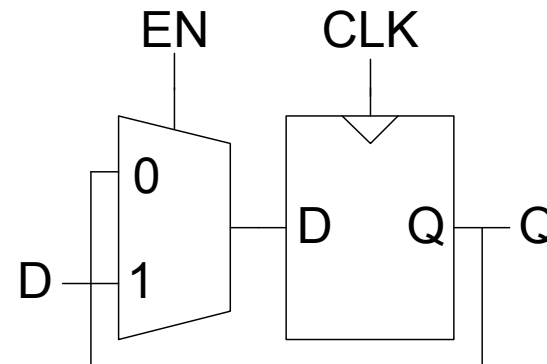
# Flip-Flops mit Taktfreigabesignal (*clock enable*)



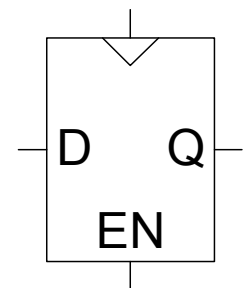
TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Eingänge: *CLK*, *D*, *EN*
  - Freigabeeingang (*EN*, enable) steuert, **wann** neue Daten (*D*) gespeichert werden
- Funktion
  - ***EN* = 1**
    - *D* wird weitergegeben an *Q* bei **steigender** Taktflanke
  - ***EN* = 0**
    - *Q* behält **alten** (gespeicherten) Wert

Interner  
Aufbau



Symbol

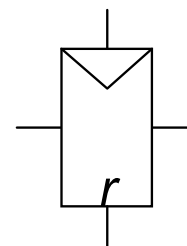
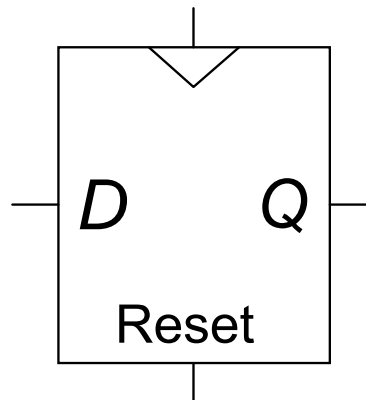


# Zurücksetzbare Flip-Flops



- Eingänge: *CLK*, *D*, *Reset*
- Funktion:
  - *Reset* = 1
    - *Q* wird auf 0 gesetzt
  - *Reset* = 0
    - Verhält sich wie **normales** D Flip-Flop

## Symbole



# Zurücksetzbare Flip-Flops



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Zwei Arten:
  - **Synchron**: Rücksetzen geschieht zu **steigender** Taktflanke
  - **Asynchron**: Rücksetzen geschieht **sofort** bei *Reset* = 1
- Interner Aufbau
  - Asynchron: Übung 3.10 im Buch
  - Synchron?

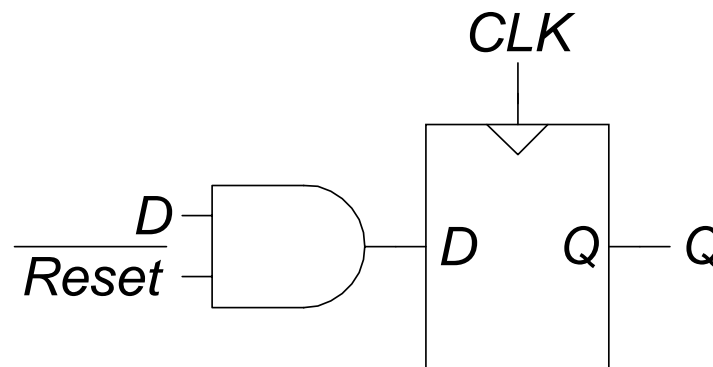


# Zurücksetzbare Flip-Flops



- Zwei Arten:
  - Synchron: Rücksetzen geschieht zu steigender Taktflanke
  - Asynchron: Rücksetzen geschieht sofort bei  $Reset = 1$
- Interner Aufbau
  - Asynchron: Übung 3.10 im Buch
  - Synchron?

Interner  
Aufbau

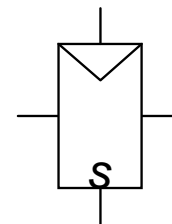
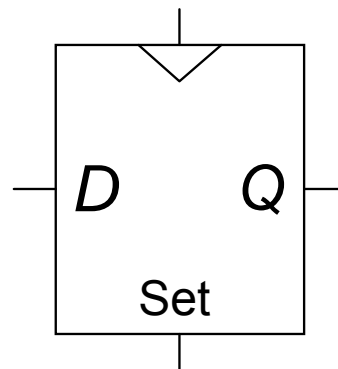


# Setzbare Flip-Flops

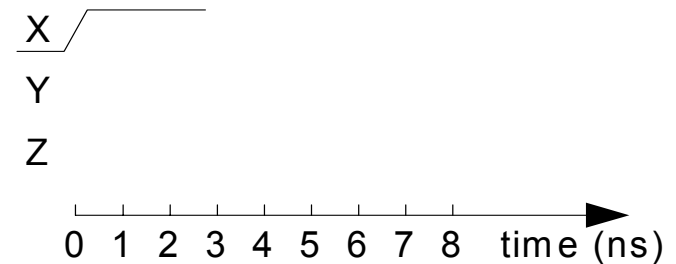
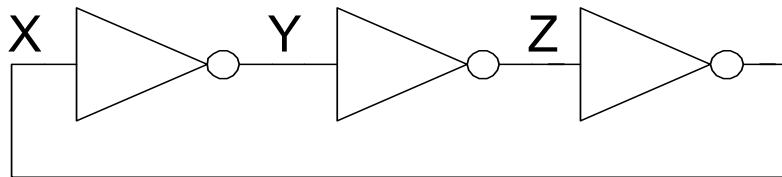


- Eingänge: *CLK*, *D*, *Set*
- Funktion:
  - *Set* = 1
    - *Q* wird auf 1 gesetzt
  - *Set* = 0
    - Verhält sich wie **normales** D Flip-Flop

## Symbole

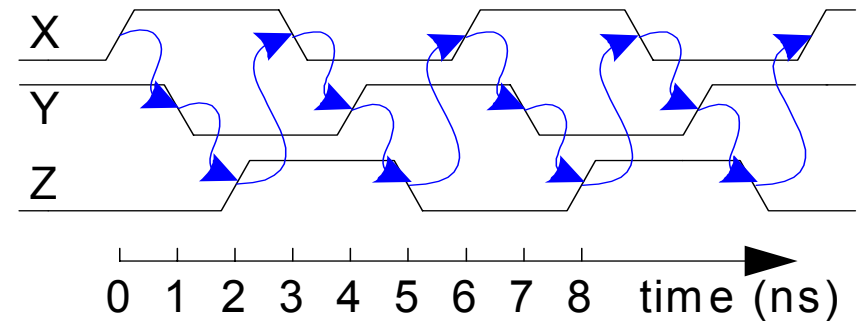
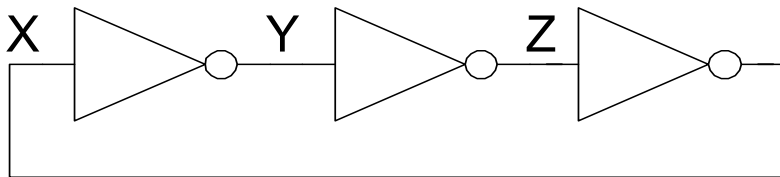


- Sequentielle Schaltungen: Alle **nicht-kombinatorischen** Schaltungen
- Merkwürdige Schaltung:



- Keine Eingänge
- 1...3 Ausgänge (Knoten X, Y, Z)

- Sequentielle Schaltungen: Alle nicht-kombinatorischen Schaltungen
- Merkwürdige Schaltung:



- Keine Eingänge
- 1...3 Ausgänge (Knoten X, Y, Z)
- Instabile Schaltung, **oszilliert**
- Periode hängt von **Inverterverzögerung** ab
  - Variiert mit Herstellungsprozess, Temperatur, ...
- Schaltung hat einen **Zyklus**: Ausgang **rückgekoppelt** auf Eingang

# Entwurf synchroner sequentieller Logik

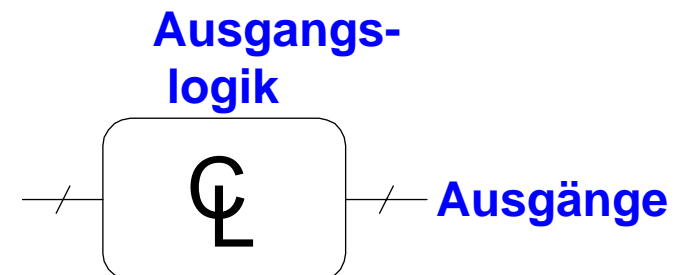
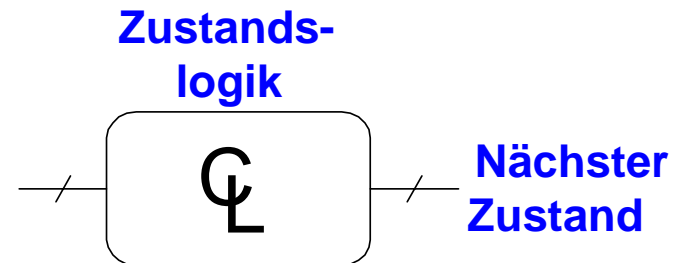
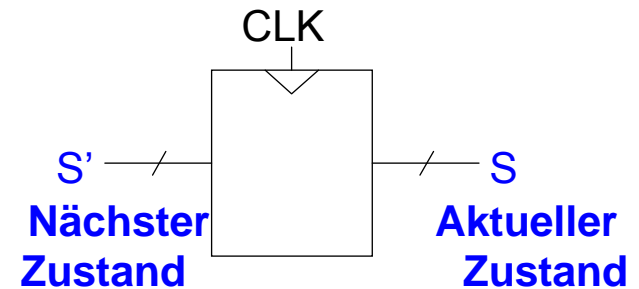


TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Rückkopplungen durch Einfügen von Registern **aufbrechen**
- Diese Register halten den **Zustand** der Schaltung
- Register ändern Zustand nur zur **Taktflanke**
  - Schaltung wird synchronisiert mit der Taktflanke
- Regeln für den **Aufbau** von synchronen sequentiellen Schaltungen
  - Jedes Schaltungselement ist **entweder** ein Register oder eine kombinatorische Schaltung
  - **Mindestens** ein Schaltungselement ist ein Register
  - Alle Register werden durch das **gleiche** Taktsignal gesteuert
  - Jeder Zyklus enthält **mindestens** ein Register
- Zwei weit verbreitete synchrone sequentielle Schaltungen
  - **Endliche Zustandsautomaten** (*Finite State Machines*, FSMs)
  - **Pipelines** (manchmal Fließbandverarbeitung genannt)

# Endliche Zustandsautomaten (FSM)

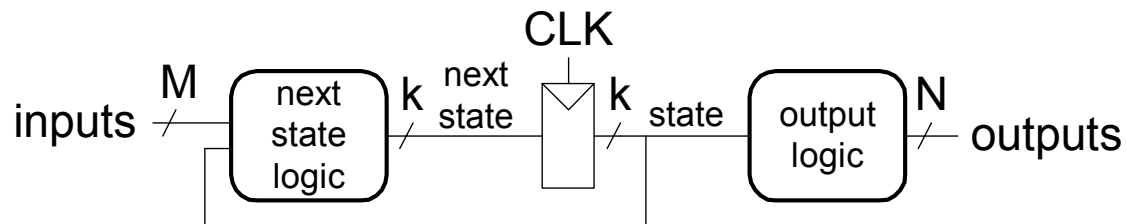
- Bestehen aus:
  - Zustandsregister**
    - Speichert **aktuellen** Zustand
    - Übernimmt **nächsten** Zustand bei Taktflanke
  - Kombinatorische Logik**
    - Berechnet **nächsten** Zustand
    - Berechnet **Ausgänge**



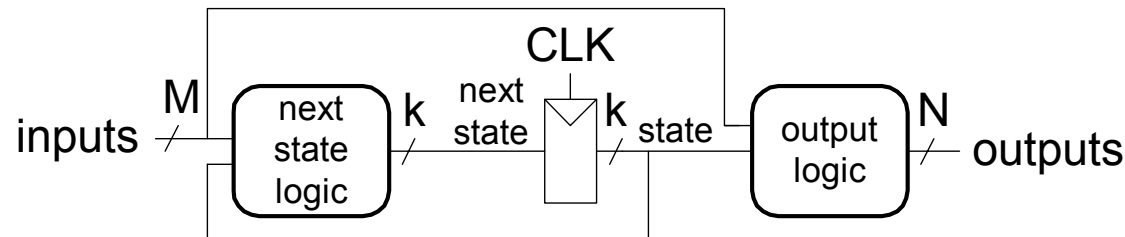
# Endliche Zustandsautomaten (FSM)

- **Nächster** Zustand hängt ab von **aktuellem** Zustand und **Eingangswerten**
- Ausgangswerte werden üblicherweise auf eine von zwei Arten bestimmt:
  - **Moore FSM**: Ausgänge hängen **nur** vom aktuellen Zustand ab
  - **Mealy FSM**: Ausgänge hängen vom aktuellen Zustand **und** den Eingangswerten ab

## Moore FSM

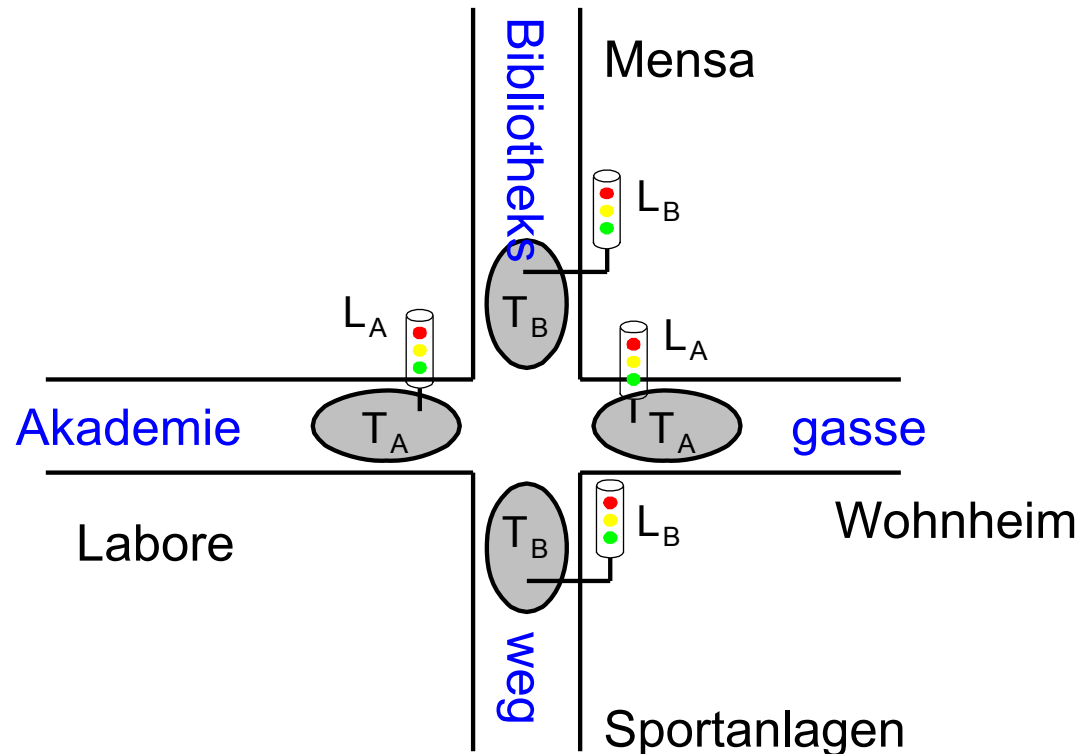


## Mealy FSM



# Beispiel für endlichen Zustandsautomaten

- Ampelsteuerung
  - Induktionsschleifen:  $T_A$ ,  $T_B$  (TRUE wenn Autos detektiert werden)
  - Ampeln:  $L_A$ ,  $L_B$



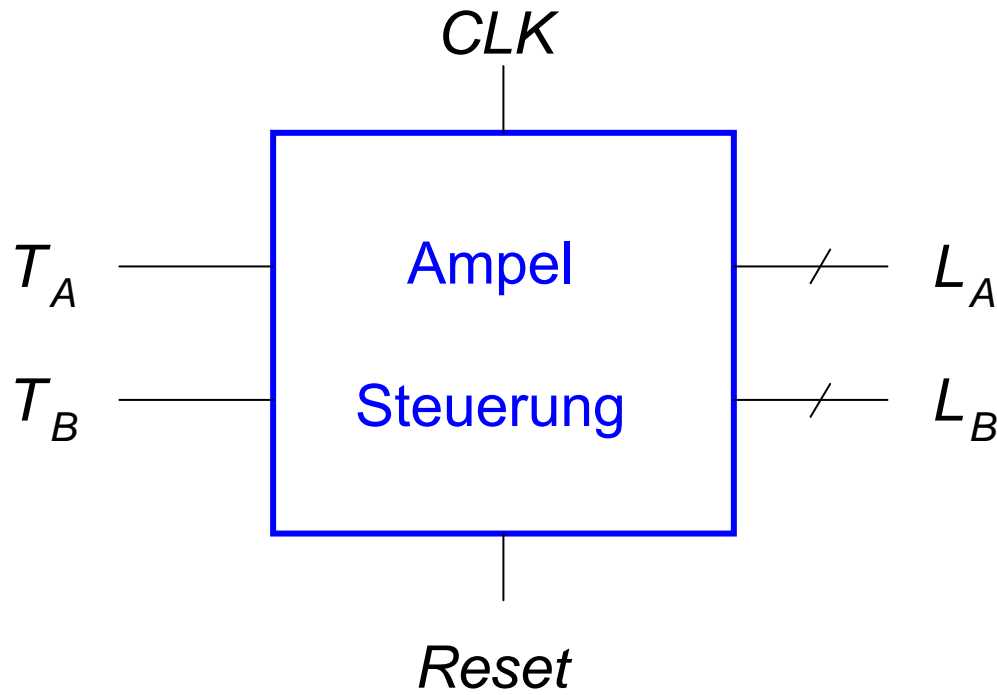


# Endlicher Automat: Außenansicht (*black box*)



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

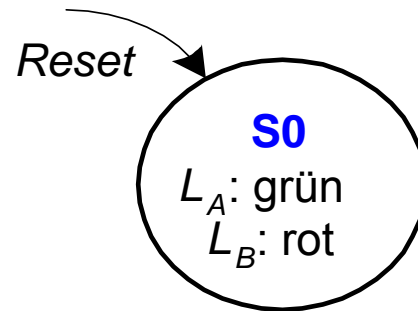
- Eingänge:  $CLK$ ,  $Reset$ ,  $T_A$ ,  $T_B$
- Ausgänge:  $L_A$ ,  $L_B$



# Zustandsübergangsdiagramm der FSM



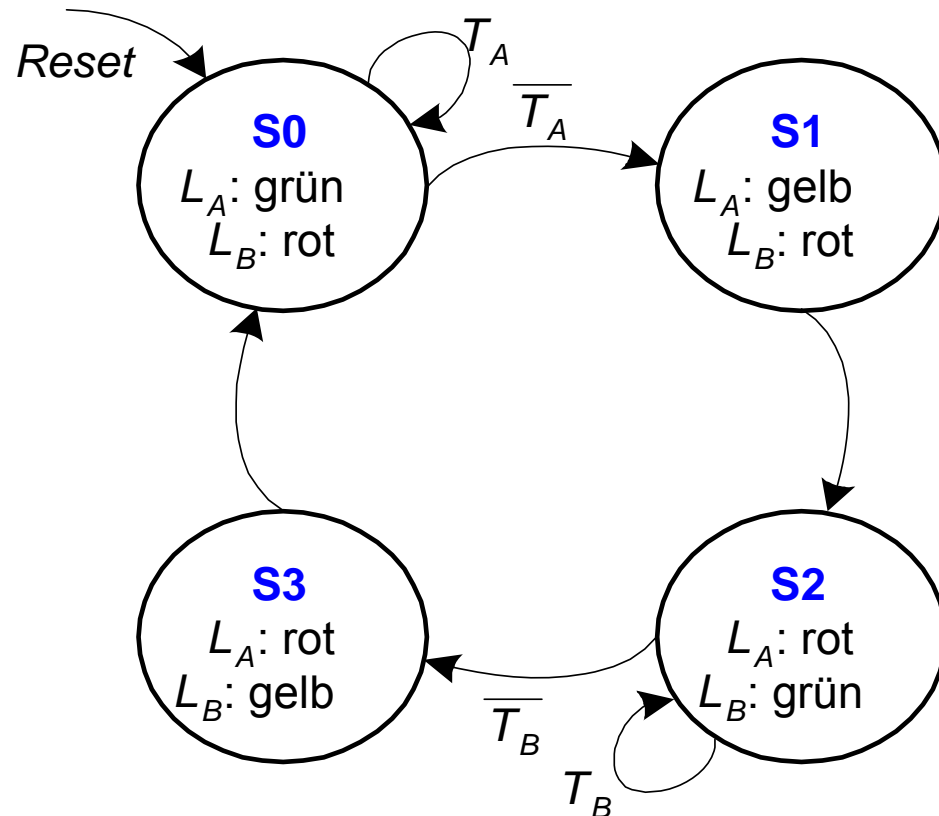
- Moore FSM: Ausgangswerte den Zuständen zuordnen
- Zustände: Kreise
- Übergänge: Pfeile



# Zustandsübergangsdiagramm der FSM



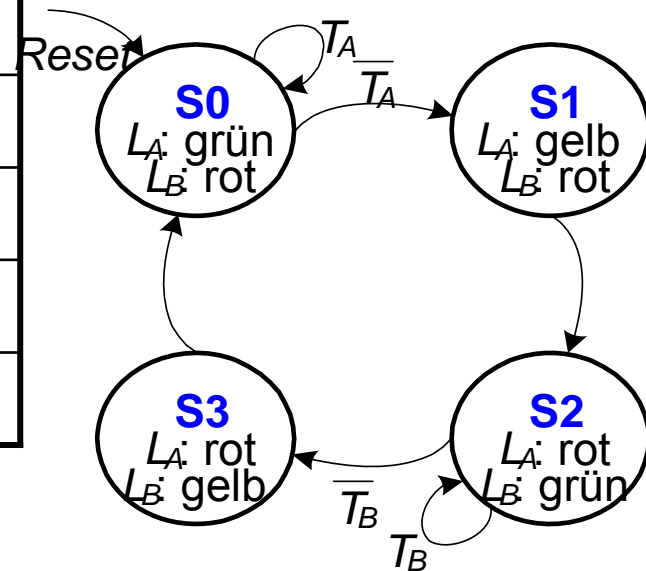
- Moore FSM: Ausgangswerte den Zuständen zuordnen
- Zustände: Kreise
- Übergänge: Pfeile



# Zustandsübergangstabelle



Aktueller Zustand	Eingänge		Nächster Zustand
	$T_A$	$T_B$	
S			$S'$
S0	0	X	
S0	1	X	
S1	X	X	
S2	X	0	
S2	X	1	
S3	X	X	



# Zustandsübergangstabelle



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Aktueller Zustand	Eingänge		Nächster Zustand
	$T_A$	$T_B$	
S			$S'$
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

# Zustandsübergangstabelle mit binärkodierten Zuständen

Zustand	Kodierung
S0	00
S1	01
S2	10
S3	11

Aktueller Zustand		Eingänge		Nächster Zustand	
$S_1$	$S_0$	$T_A$	$T_B$	$S'_1$	$S'_0$
0	0	0	X		
0	0	1	X		
0	1	X	X		
1	0	X	0		
1	0	X	1		
1	1	X	X		

# Zustandsübergangstabelle mit binärkodierten Zuständen

Zustand	Kodierung
S0	00
S1	01
S2	10
S3	11

Aktueller Zustand		Eingänge		Nächster Zustand	
$S_1$	$S_0$	$T_A$	$T_B$	$S'_1$	$S'_0$
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

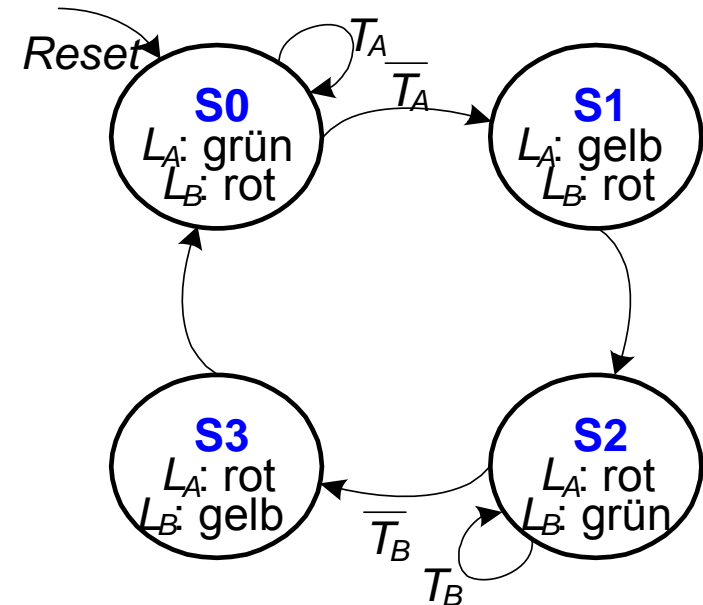
$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S_1} \overline{S_0} \overline{T_A} + S_1 \overline{S_0} \overline{T_B}$$

# FSM Ausgangstabelle

Aktueller Zustand		Ausgänge			
$S_1$	$S_0$	$L_{A1}$	$L_{A0}$	$L_{B1}$	$L_{B0}$
0	0				
0	1				
1	0				
1	1				

Ausgangswert	Kodierung
grün	00
gelb	01
rot	10





# FSM Ausgangstabelle

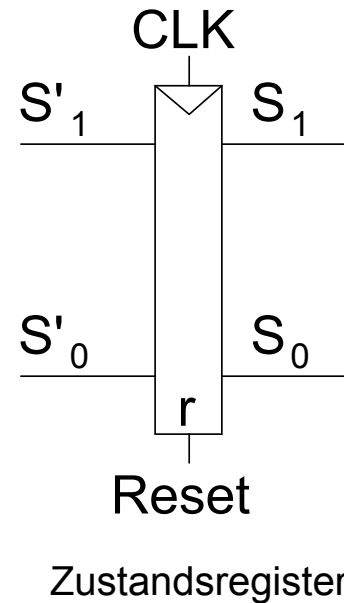


Aktueller Zustand		Ausgänge			
$S_1$	$S_0$	$L_{A1}$	$L_{A0}$	$L_{B1}$	$L_{B0}$
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Ausgangs wert	Kodierung
grün	00
gelb	01
rot	10

$$\begin{aligned}L_{A1} &= S_1 \\L_{A0} &= \overline{S_1} S_0 \\L_{B1} &= \overline{S_1} \\L_{B0} &= S_1 S_0\end{aligned}$$

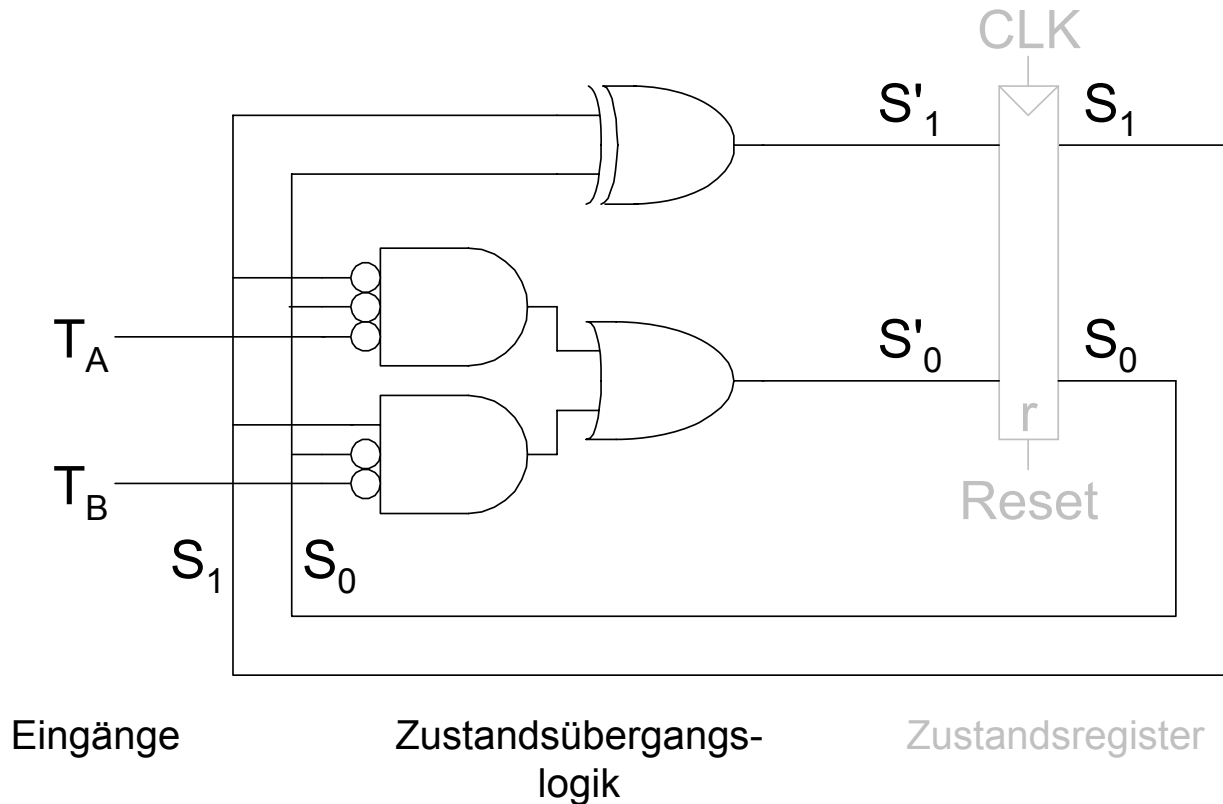
# FSM Schaltplan: Zustandsregister



# FSM Schaltplan: Zustandsübergangslogik



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

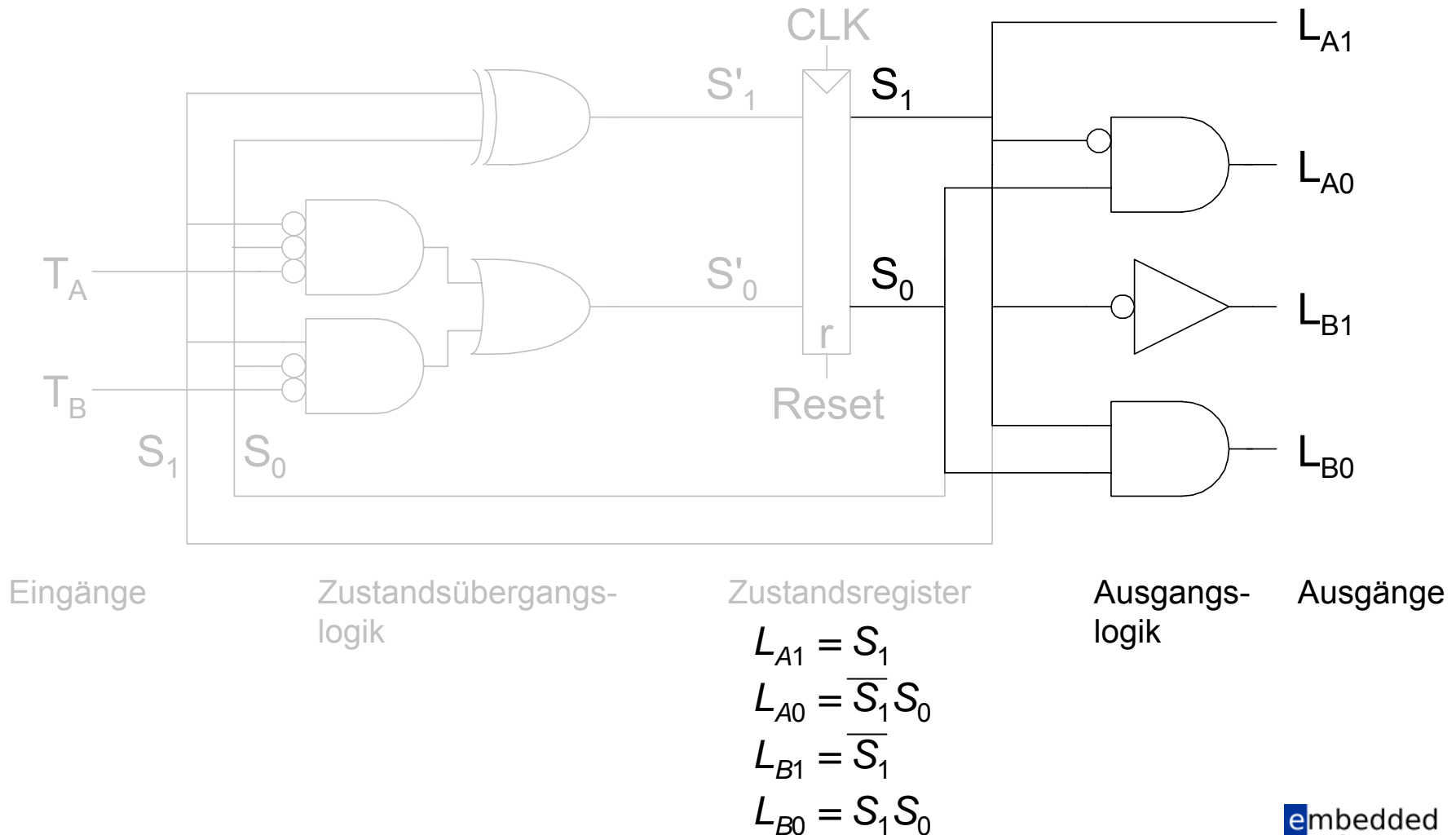


$$S'_1 = S_1 \oplus S_0$$
$$S'_0 = \overline{S_1} \overline{S_0} T_A + S_1 \overline{S_0} T_B$$

# FSM Schaltplan: Ausgangslogik



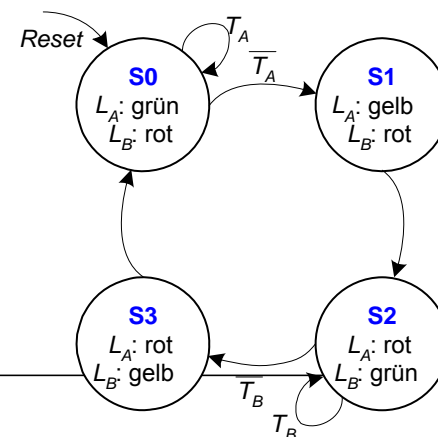
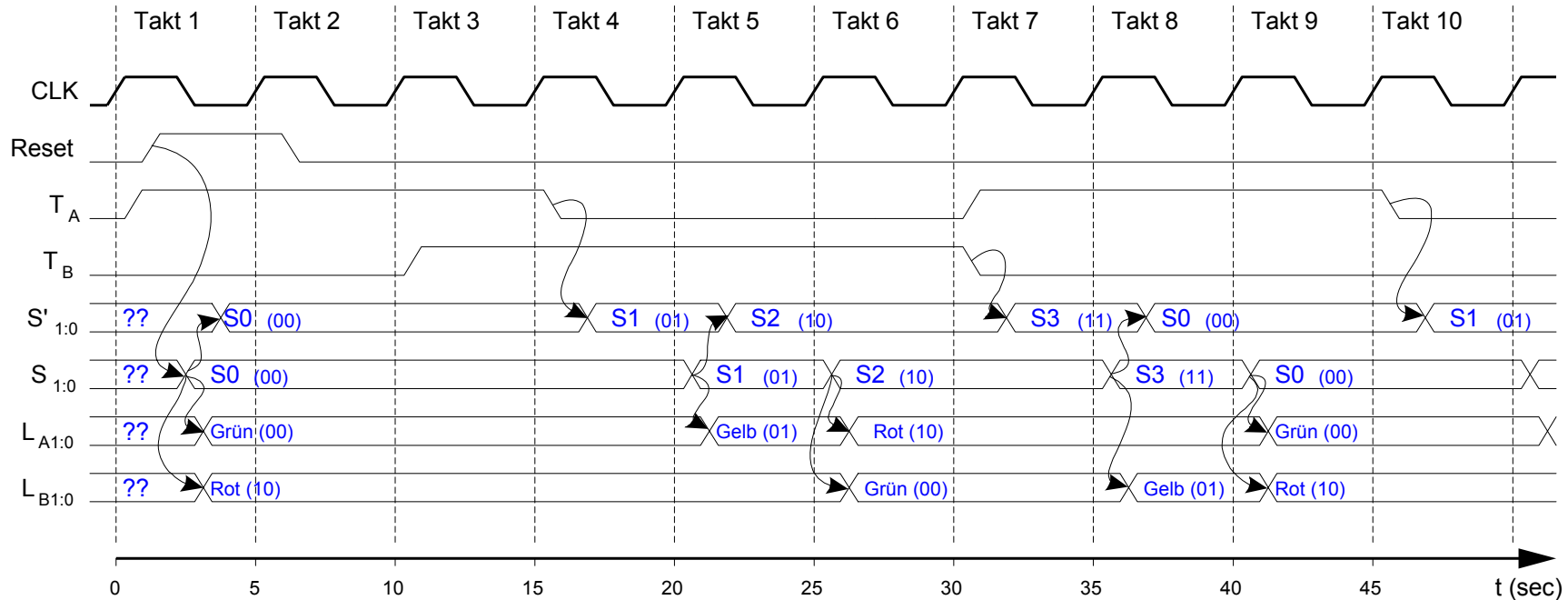
TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



# FSM Zeitverhalten: Timing-Diagramm



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

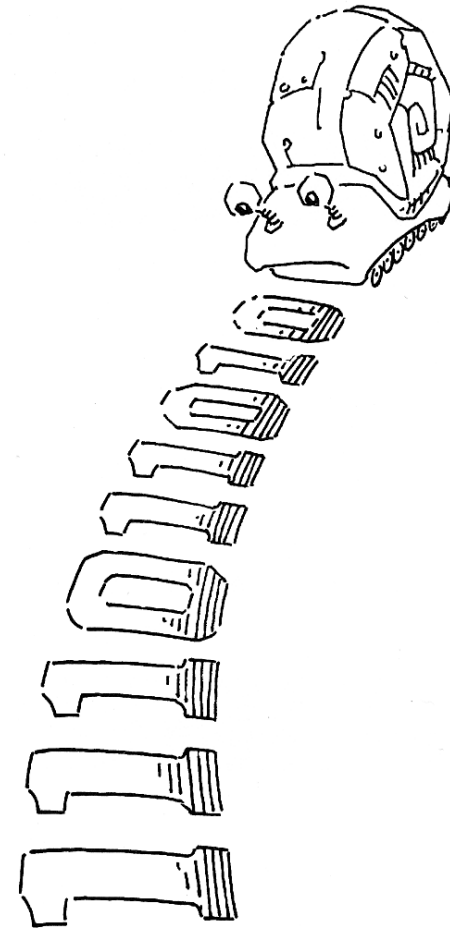




- Binär
  - z.B. für vier Zustände 00, 01, 10, 11
  
- 1-aus-N Code (*One-hot encoding*)
  - Ein Zustandsbit **pro** Zustand
  - Zu jedem Zeitpunkt ist **genau** ein Zustandsbit gesetzt
  - z.B. für vier Zustände 0001, 0010, 0100, 1000
  - Benötigt zwar **mehr** Flip-Flops
  - ... aber Zustandsübergangs- und Ausgangslogiken sind häufig **kleiner**
    - ... und **schneller**

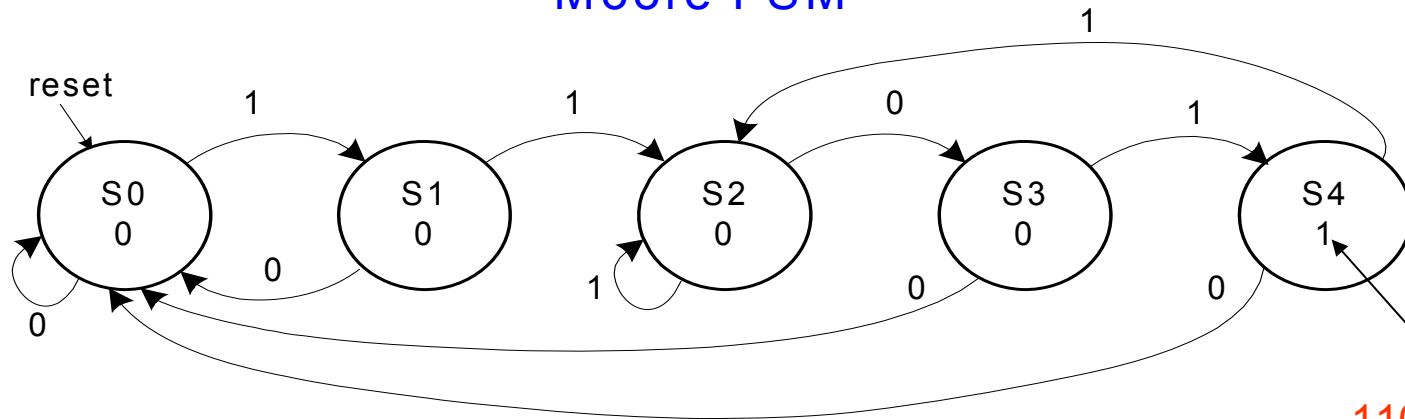
# Vergleich Moore- und Mealy-Automaten

- Erkenne Bitfolge 1101 auf Lochstreifen



# Zustandsübergangsdiagramme

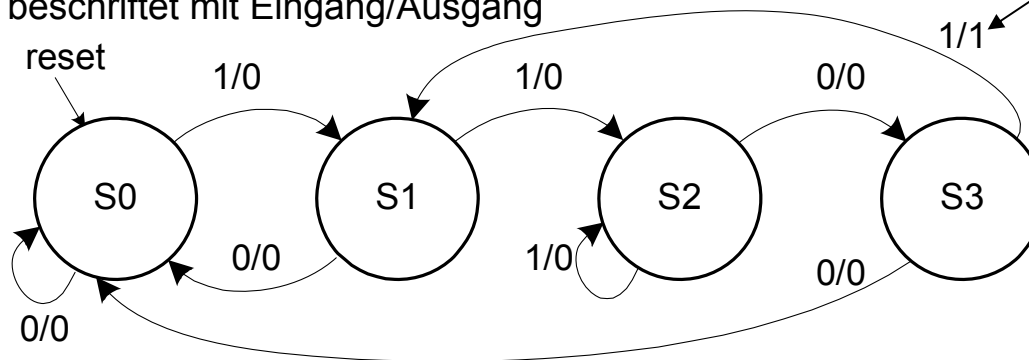
## Moore FSM



1101 erkannt

## Mealy FSM

Mealy FSM: Pfeile beschriftet mit Eingang/Ausgang





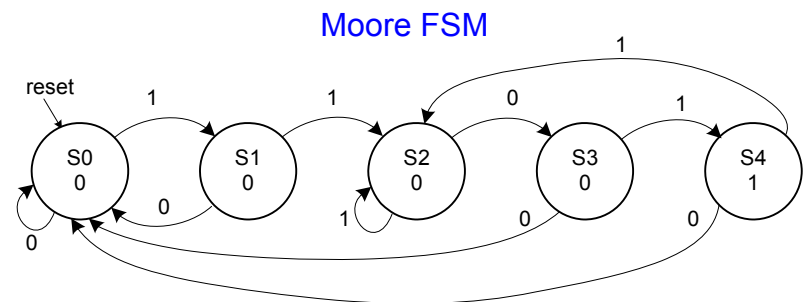
# Moore-Automat: Zustandsübergangstabelle



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Aktueller Zustand			Eingänge e A	Nächster Zustand		
$S_2$	$S_1$	$S_0$		$S'_2$	$S'_1$	$S'_0$
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
1	0	0	0			
1	0	0	1			

Zustand	Kodierung
S0	000
S1	001
S2	010
S3	011
S4	100



# Moore-Automat: Zustandsübergangstabelle



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Aktueller Zustand			Eingang	Nächster Zustand		
$S_2$	$S_1$	$S_0$		$S'_2$	$S'_1$	$S'_0$
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	0
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	0	1	0
0	1	1	0	0	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	0
1	0	0	1	0	1	0

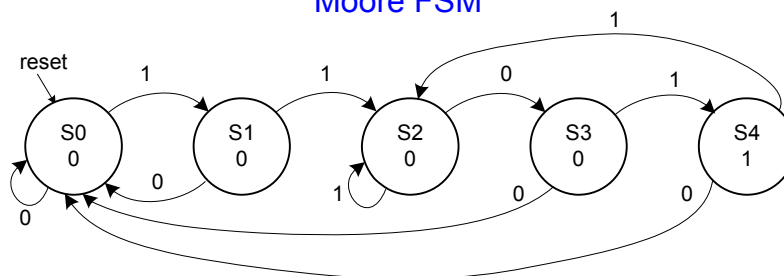
Zustand	Kodierung
S0	000
S1	001
S2	010
S3	011
S4	100

# Moore-Automat: Ausgangstabelle



Aktueller Zustand			Ausgang
$S_2$	$S_1$	$S_0$	$Y$
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	

Moore FSM



# Moore-Automat: Ausgangstabelle



Aktueller Zustand			Ausgang
$S_2$	$S_1$	$S_0$	$Y$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1

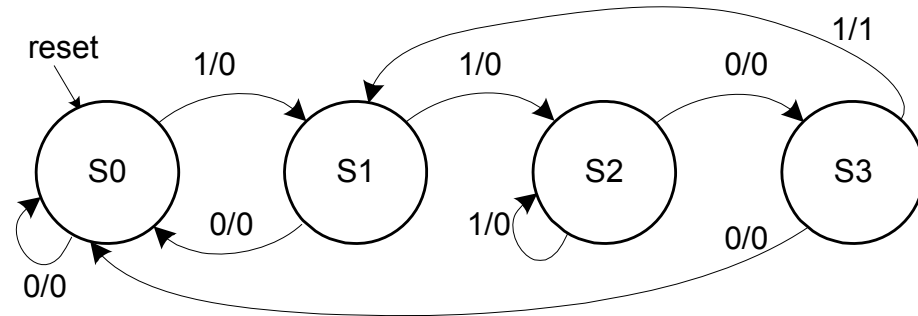
$$Y = S_2$$

# Mealy-Automat: Zustandsübergangs- und Ausgangstabelle

Aktueller Zustand		Eingang	Nächster Zustand		Ausgang
$S_1$	$S_0$	$A$	$S'_1$	$S'_0$	$Y$
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Zustand	Kodierung
S0	00
S1	01
S2	10
S3	11

Mealy FSM



# Mealy-Automat: Zustandsübergangs- und Ausgangstabelle

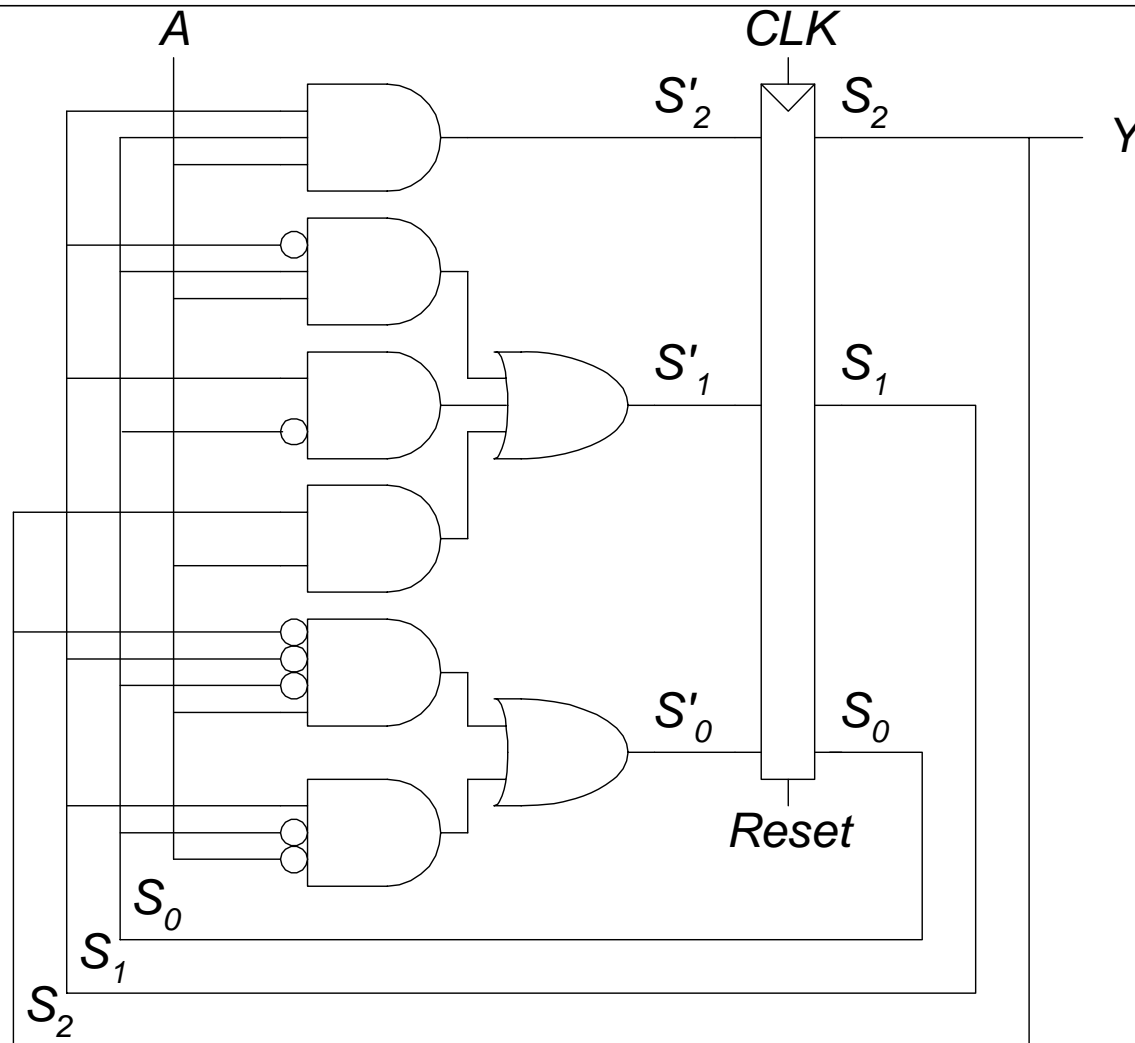
Aktueller Zustand		Eingang	Nächster Zustand		Ausgang
$S_1$	$S_0$	$A$	$S'_1$	$S'_0$	$Y$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	1

Zustand	Kodierung
S0	00
S1	01
S2	10
S3	11

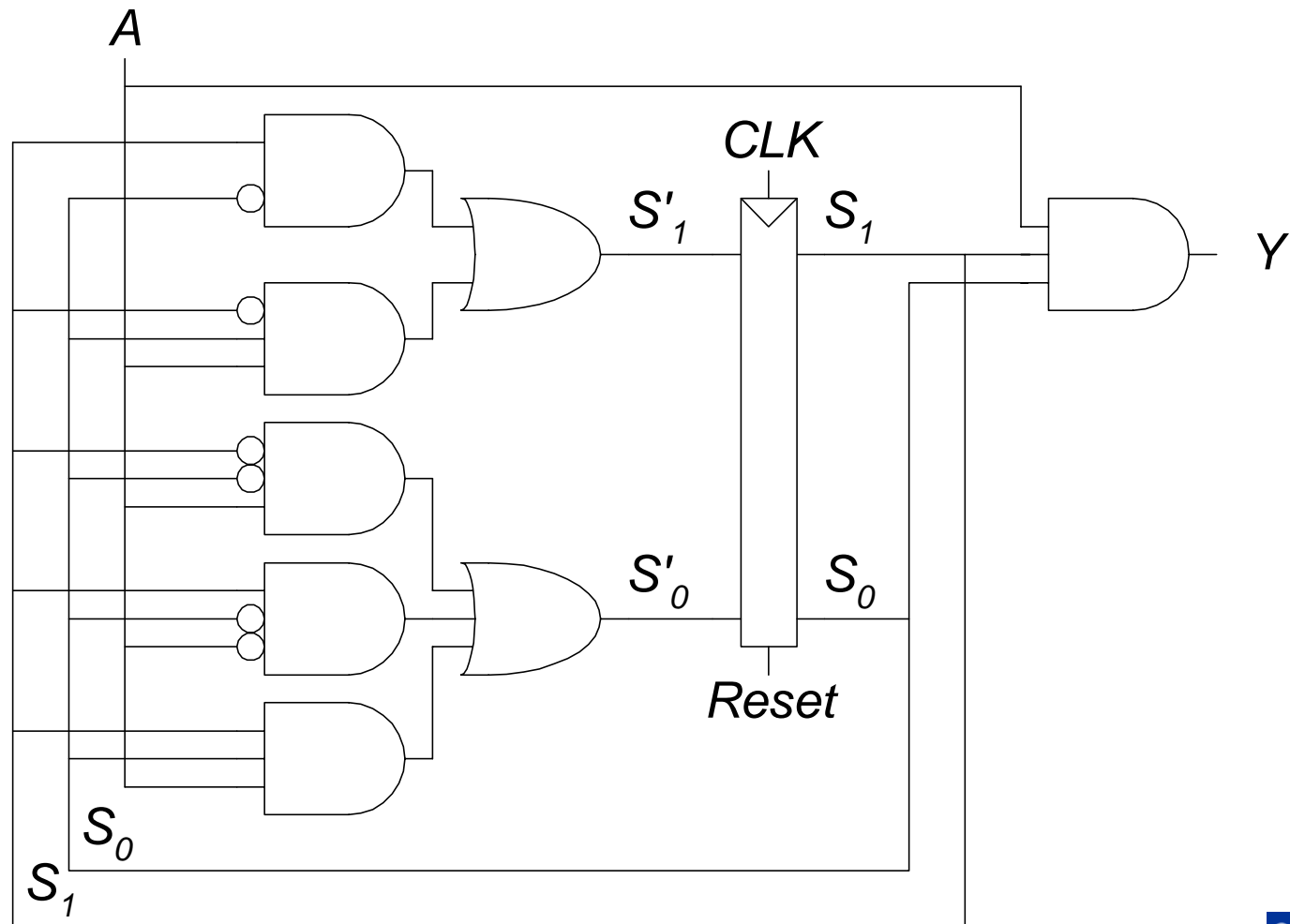
# Moore-Automat: Schaltplan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



# Mealy-Automat: Schaltplan

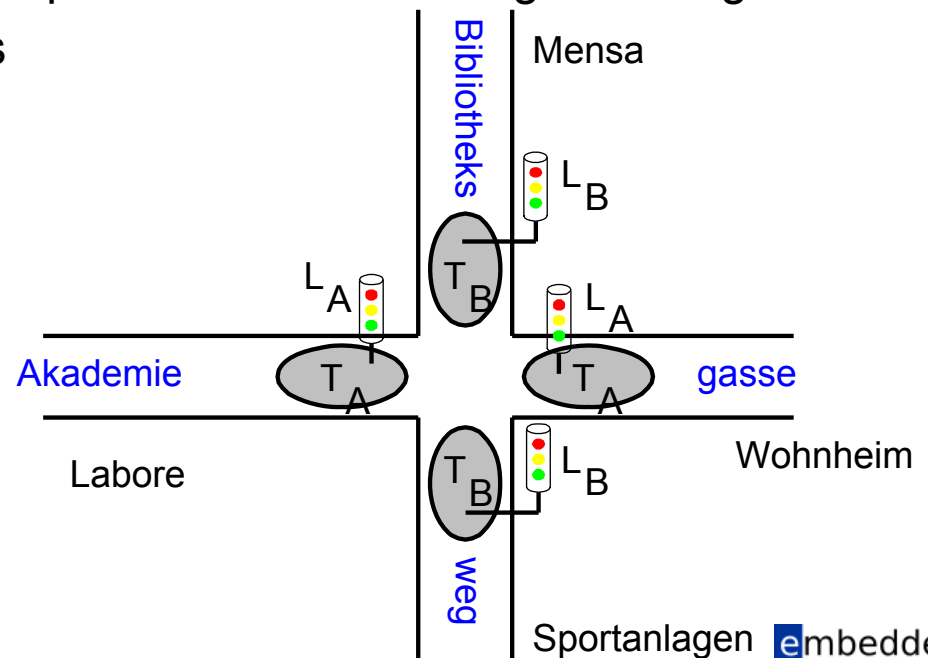






# Zerlegen von Zustandsautomaten

- Aufteilen **komplexer** FSMs in **einfachere interagierende** FSMs
  - Manchmal auch Dekomposition genannt
- Beispiel: Erweiterte Ampelsteuerung um Modus für **Festumzüge**
  - FSM bekommt zwei weitere Eingänge:  $F$ ,  $R$
  - $F = 1$  **aktiviert** Festumzugsmodus: Ampeln für Bibliotheksweg bleiben grün
  - $R = 1$  **deaktiviert** Festumzugsmodus



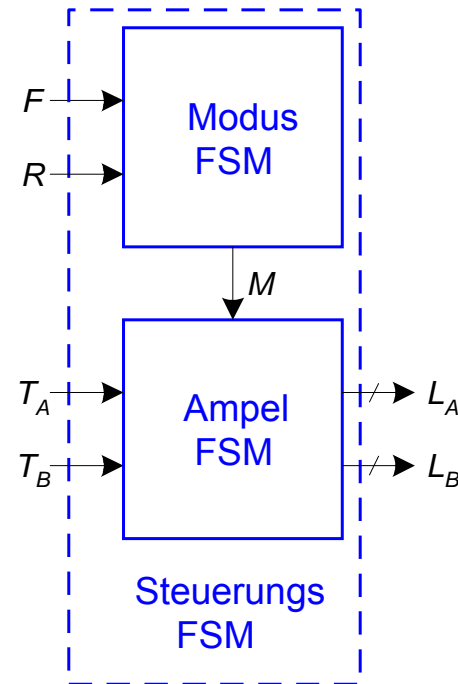
# FSM mit Festumzugsmodus



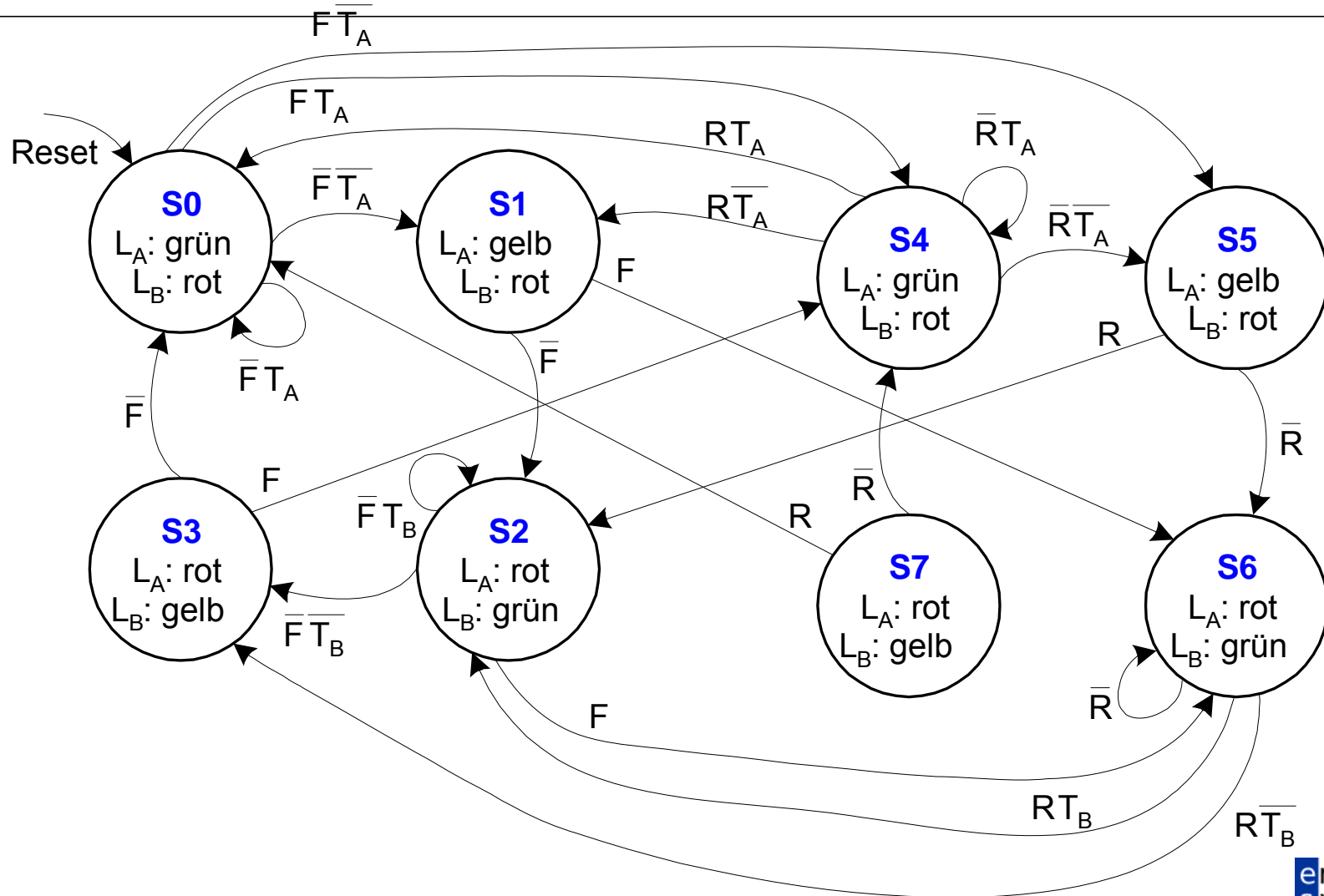
## Unzerlegte FSM



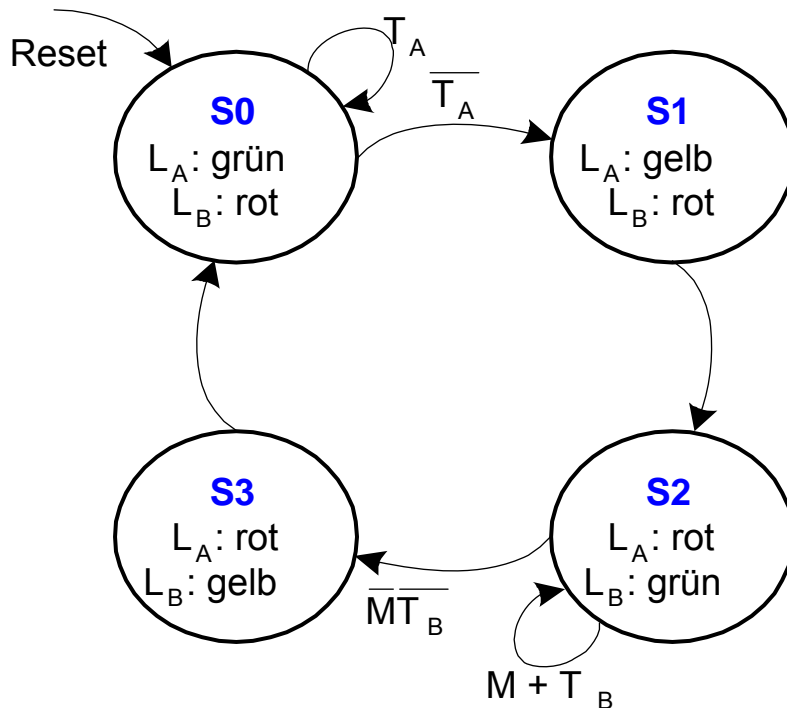
## Zerlegte FSM



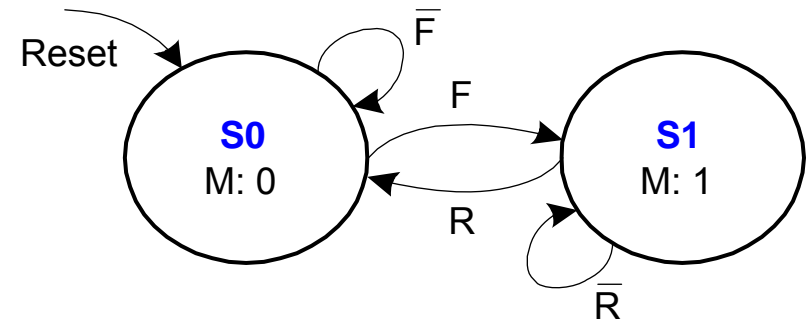
# Zustandsübergangsdiagramm für unzerlegte FSM



# Zustandsübergangsdiagramm für zerlegte FSM



Ampelsteuerungs FSM



Modus FSM



- Definiere Ein- und Ausgänge
- Zeichne Zustandsdiagramm
- Stelle Zustandsübergangstabelle auf
- Kodiere Zustände (binär, one-hot, ...)
- Für Moore-Automat:
  - Verwende kodierte Zustände in Zustandsübergangstabelle
  - Stelle Ausgangstabelle auf
- Für Mealy-Automat
  - Erweitere Zustandsübergangstabelle um Ausgänge und verwende kodierte Zustände
- Stelle Boole'sche Gleichungen für Zustandsübergangs- und Ausgangslogiken auf
- Entwerfe Schaltplan
  - Gatter, Register

# Zeitverhalten von sequentiellen Schaltungen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Flip-Flop übernimmt Daten von  $D$  zur **Taktflanke**
- $D$  darf sich nicht ändern, wenn es **übernommen** wird (*sampled*)
  - Muss stabil sein
- Ähnlich zu Fotografie: Keine Bewegung zum Auslösezeitpunkt
  - Sonst **unscharf**
- Also:  $D$  darf sich nicht zur Taktflanke ändern
  - Sonst möglicherweise **metastabil**
- Genauer:
  - $D$  darf sich nicht in **Zeitfenster** um Taktflanke **herum** ändern

# Zeitanforderungen an Eingangssignale

- Setup-Zeit

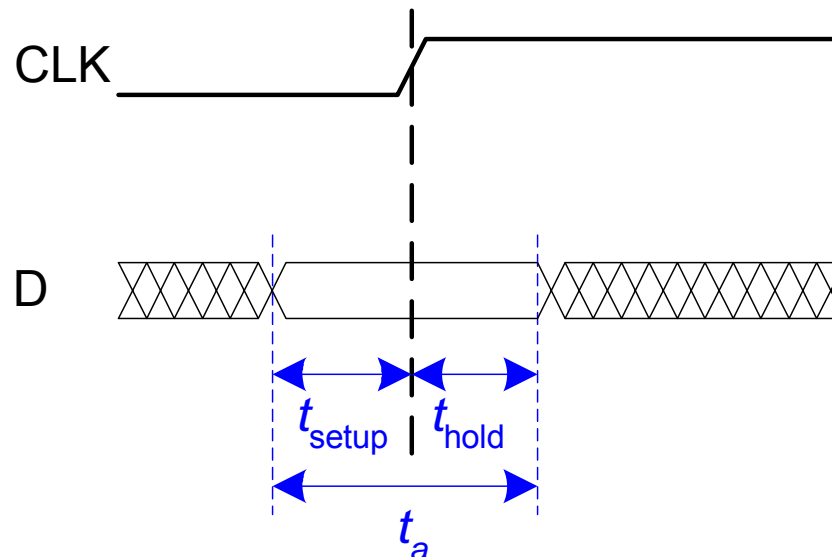
- $t_{\text{setup}}$  = Zeitintervall *vor Taktflanke*, in dem  $D$  sich nicht ändern darf (=stabil sein muss)

- Hold-Zeit

- $t_{\text{hold}}$  = Zeitintervall *nach Taktflanke* in dem  $D$  stabil sein muss

- Abtastzeit:  $t_a$  = Zeitintervall um Taktflanke herum in dem  $D$  stabil sein muss

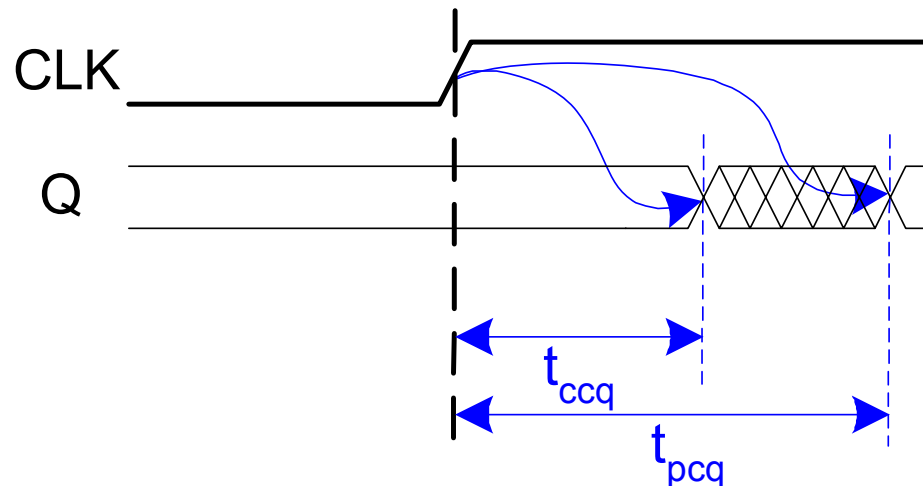
- $t_a = t_{\text{setup}} + t_{\text{hold}}$





# Zeitanforderungen an Ausgangssignale

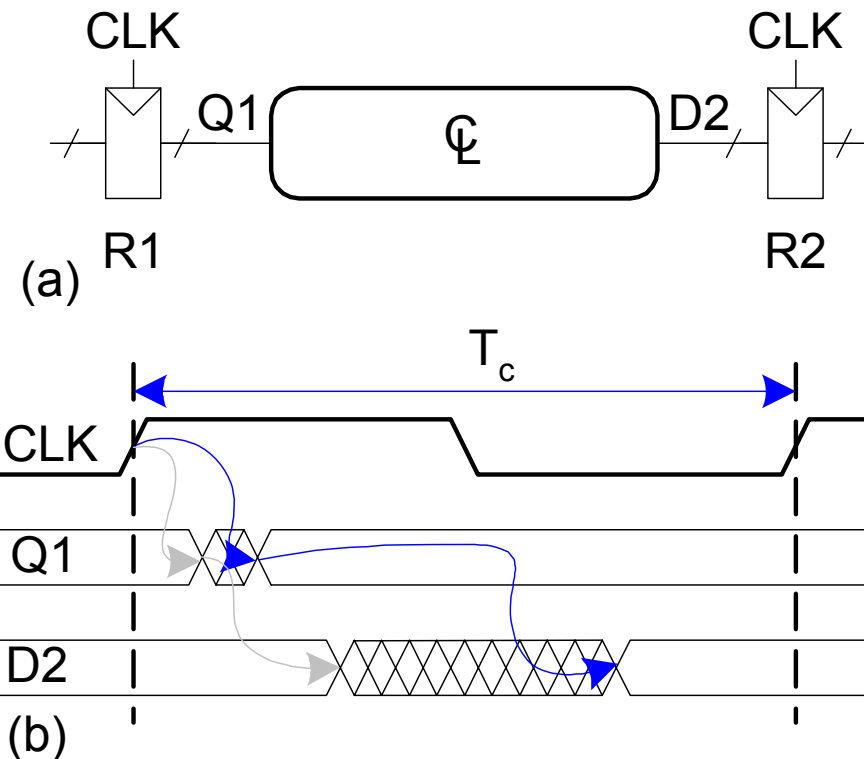
- **Laufzeitverzögerung** (*propagation delay*)
  - $t_{pcq}$  = Zeitintervall nach Taktflanke, nach dem Q garantiert **stabil** ist
    - sich also nicht mehr ändert!
- **Kontaminationsverzögerung** (*contamination delay*)
  - $t_{ccq}$  = Zeitintervall nach Taktflanke, nach dem Q beginnen könnte, sich zu **ändern**





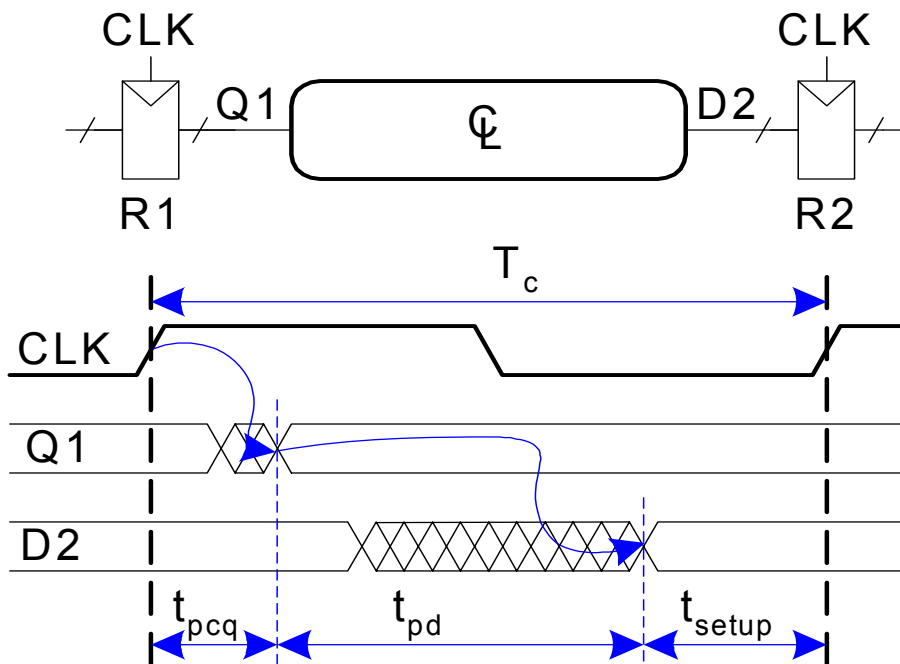
- Die Eingänge in eine synchrone sequentielle Schaltung müssen in der ganzen Abtastzeit stabil sein
- Genauer: Stabil mindestens
  - ... ab  $t_{\text{setup}}$  vor der Taktflanke
  - ... bis  $t_{\text{hold}}$  nach der Taktflanke

- Verzögerung zwischen Registern hat **Maximal-** und **Minimalwert**
  - Abhängig von den Verzögerungen der **kombinatorischen** Schaltelemente



# Anforderungen an Setup-Zeit

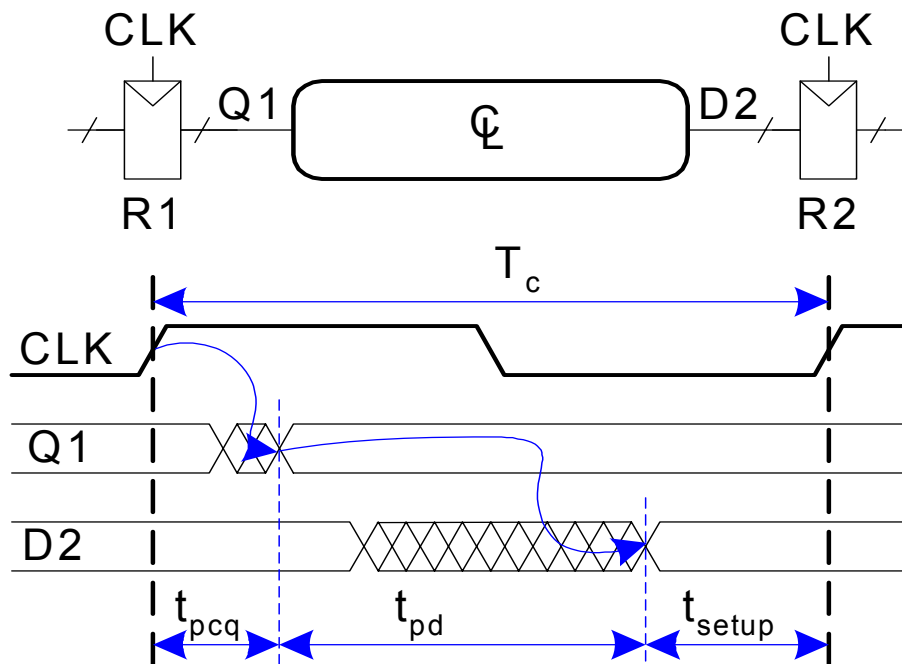
- Einhalten der Setup-Zeit hängt von der **Maximal-Verzögerung** von Register R1 durch kombinatorische Logik ab
- Eingang zu Register muss **mindestens** ab  $t_{\text{setup}}$  **vor** Taktflanke stabil sein



$$T_c \geq$$

# Anforderungen an Setup-Zeit

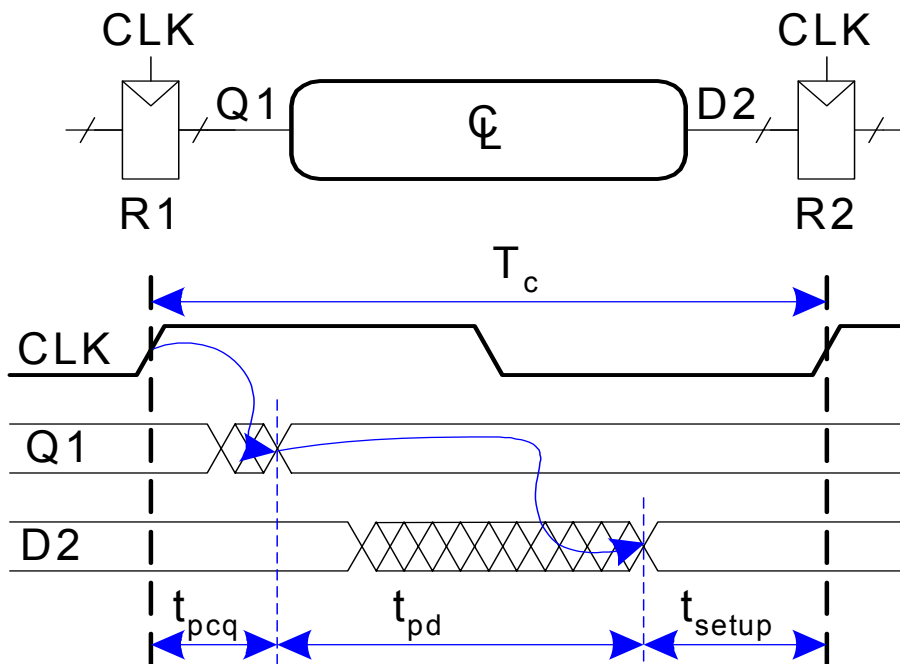
- Einhalten der Setup-Zeit hängt von der **Maximal-Verzögerung** von Register R1 durch kombinatorische Logik ab
- Eingang zu Register muss **mindestens** ab  $t_{\text{setup}}$  **vor** Taktflanke stabil sein



$$T_c \geq t_{pcq} + t_{pd} + t_{\text{setup}}$$
$$t_{pd} \leq$$

# Anforderungen an Setup-Zeit

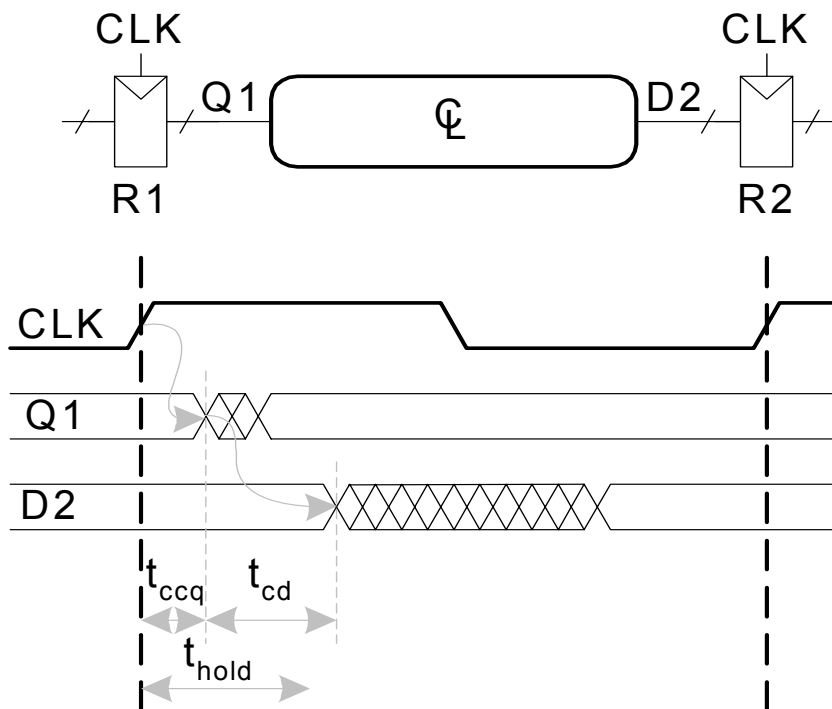
- Einhalten der Setup-Zeit hängt von der **Maximal-Verzögerung** von Register R1 durch kombinatorische Logik ab
- Eingang zu Register muss **mindestens** ab  $t_{\text{setup}}$  **vor** Taktflanke stabil sein



$$T_c \geq t_{pcq} + t_{pd} + t_{\text{setup}}$$
$$t_{pd} \leq T_c - (t_{pcq} + t_{\text{setup}})$$

# Anforderungen an Hold-Zeit

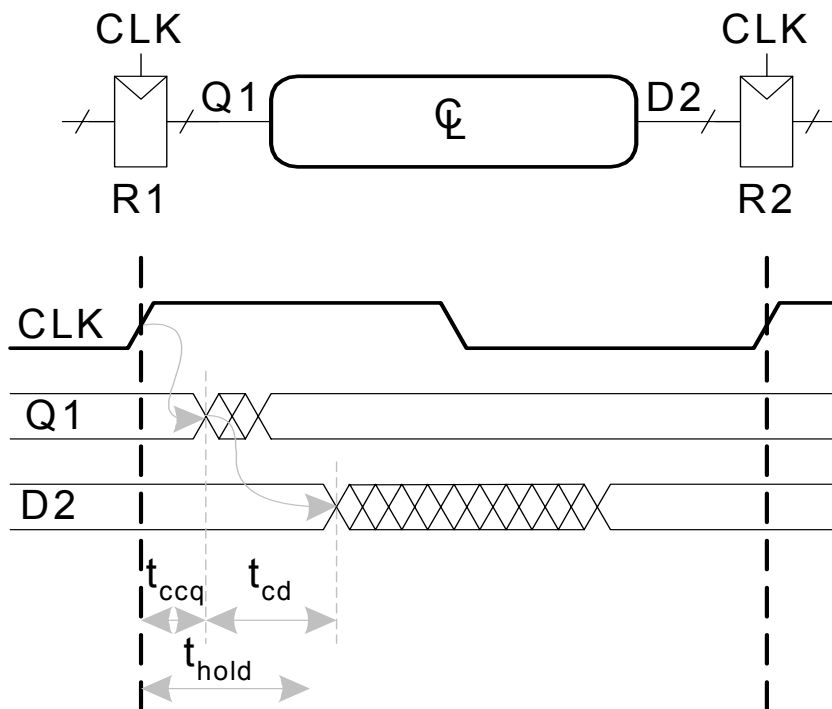
- Einhalten der Hold-Zeit hängt von der **minimalen** Verzögerung von Register R1 durch die kombinatorische Logik ab
- Der Eingang an Register R2 muss **mindestens** bis  $t_{\text{hold}}$  **nach** der Taktflanke stabil sein



$$t_{\text{hold}} <$$

# Anforderungen an Hold-Zeit

- Einhalten der Hold-Zeit hängt von der **minimalen** Verzögerung von Register R1 durch die kombinatorische Logik ab
- Der Eingang an Register R2 muss **mindestens** bis  $t_{\text{hold}}$  **nach** der Taktflanke stabil sein

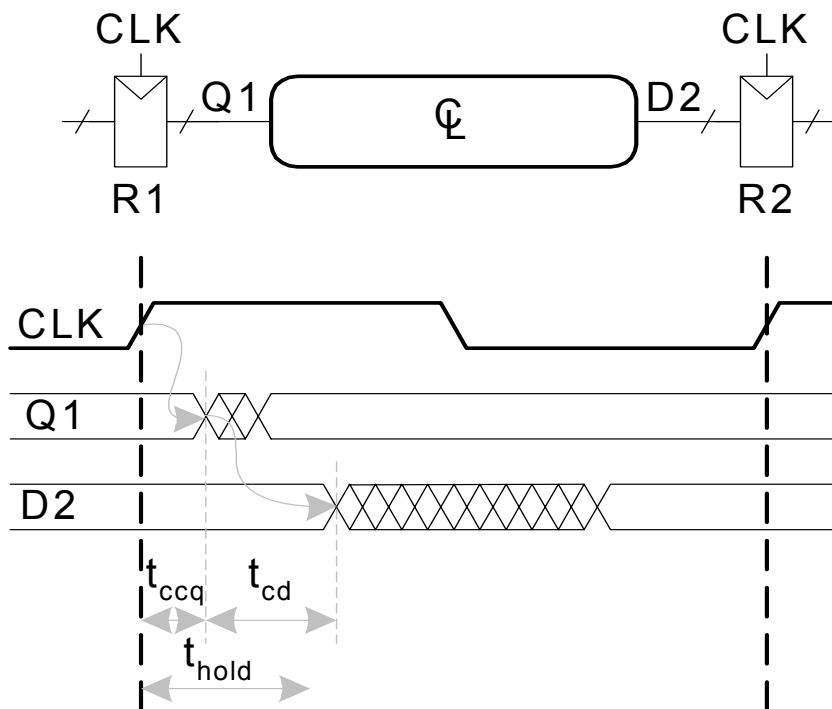


$$t_{\text{hold}} < t_{ccq} + t_{cd}$$
$$t_{cd} >$$



# Anforderungen an Hold-Zeit

- Einhalten der Hold-Zeit hängt von der **minimalen** Verzögerung von Register R1 durch die kombinatorische Logik ab
- Der Eingang an Register R2 muss **mindestens** bis  $t_{\text{hold}}$  **nach** der Taktflanke stabil sein



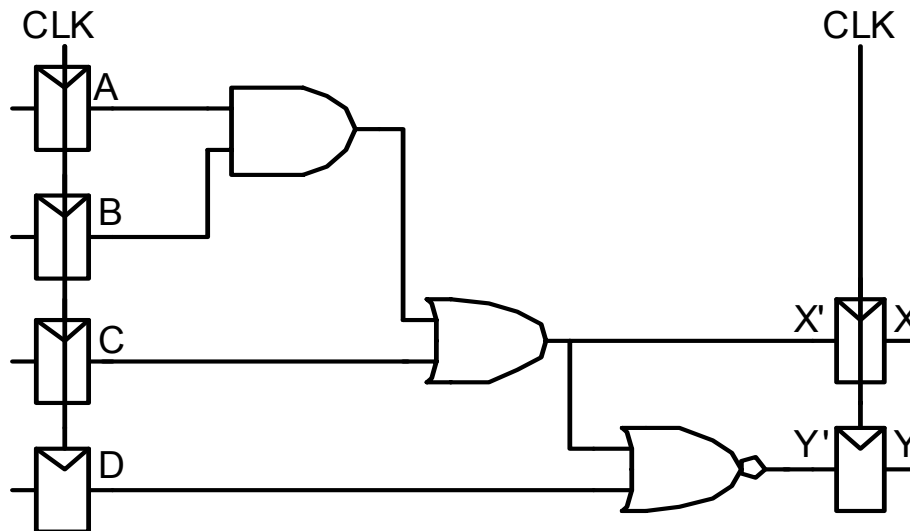
$$t_{\text{hold}} < t_{\text{ccq}} + t_{\text{cd}}$$

$$t_{\text{cd}} > t_{\text{hold}} - t_{\text{ccq}}$$

# Analyse des Zeitverhaltens



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



## Verzögerungsangaben

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{\text{setup}} = 60 \text{ ps}$$

$$t_{\text{hold}} = 70 \text{ ps}$$

Pro Gatter

$$t_{pd} = 35 \text{ ps}$$

$$t_{cd} = 25 \text{ ps}$$

$$t_{pd} =$$

$$t_{cd} =$$

Einhalten von Setup-Zeitanforderung:    Einhalten von Hold-Zeitanforderung:

$$T_c \geq$$

$$f_c = 1/T_c =$$

$$t_{ccq} + t_{cd} > t_{\text{hold}} \quad ?$$

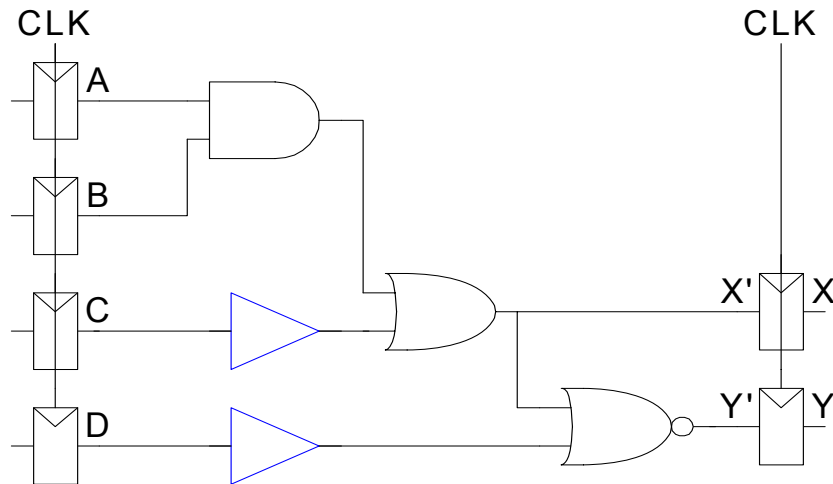


# Beheben der verletzten Hold-Zeit Anforderung



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Füge Puffer in zu kurze Pfade ein!



$$t_{pd} =$$

$$t_{cd} =$$

Einhalten der Setup-Zeit Anforderung:    Einhalten der Hold-Zeit Anforderung:

$$T_c \geq$$

$$f_c =$$

$$t_{ccq} + t_{cd} > t_{hold} \quad ?$$

Verzögerungsangaben

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{setup} = 60 \text{ ps}$$

$$t_{hold} = 70 \text{ ps}$$

Pro Gatter

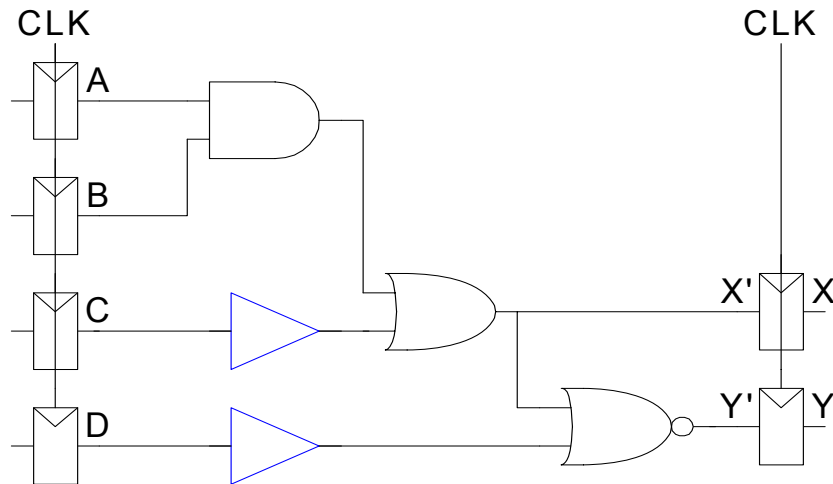
$$\left[ \begin{array}{l} t_{pd} = 35 \text{ ps} \\ t_{cd} = 25 \text{ ps} \end{array} \right.$$

# Beheben der verletzten Hold-Zeit Anforderung



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Füge Puffer in zu kurze Pfade ein!



$$t_{pd} = 3 \times 35 \text{ ps} = 105 \text{ ps}$$

$$t_{cd} = 2 \times 25 \text{ ps} = 50 \text{ ps}$$

Einhalten der Setup-Zeit Anforderung:    Einhalten der Hold-Zeit Anforderung:

$$T_c \geq (50 + 105 + 60) \text{ ps} = 215 \text{ ps}$$

$$f_c = 1/T_c = 4.65 \text{ GHz}$$

$$t_{ccq} + t_{cd} > t_{hold} \text{ ?}$$

(30 + 50) ps > 70 ps ? **Ja, eingehalten!**

Verzögerungsangaben

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{setup} = 60 \text{ ps}$$

$$t_{hold} = 70 \text{ ps}$$

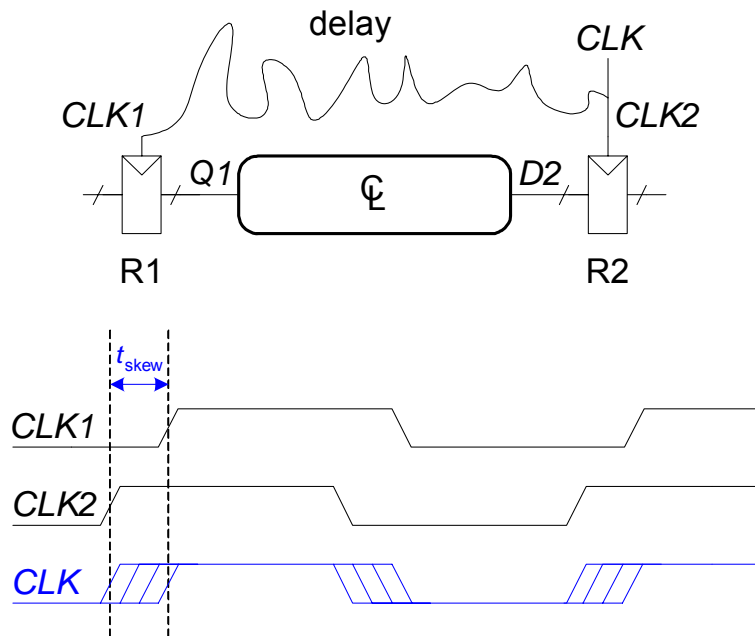
Pro Gatter

$$t_{pd} = 35 \text{ ps}$$

$$t_{cd} = 25 \text{ ps}$$

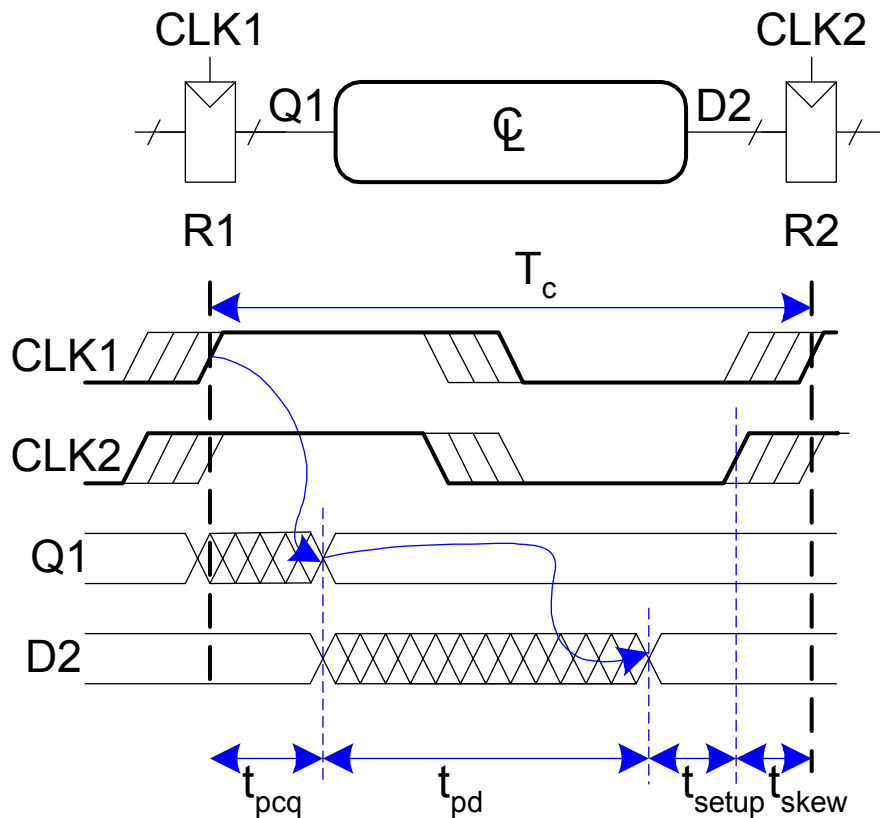
# Taktverschiebung (*clock skew*)

- Der Takt kommt nicht bei allen Registern zur **gleichen** Zeit an
  - Unterschiedliche Verdrahtungswege auf dem Chip, Logik in Taktsignal (gated clock, vermeiden!)
- Verschiebung (oder Versatz, *skew*) ist die **Differenz** der Ankunftszeit zwischen zwei Registern
- Überprüfe, ob auch bei **maximalem** Versatz die dynamische Entwurfsdisziplin noch **eingehalten** ist



# Setup-Zeitorderungen mit Taktverschiebung

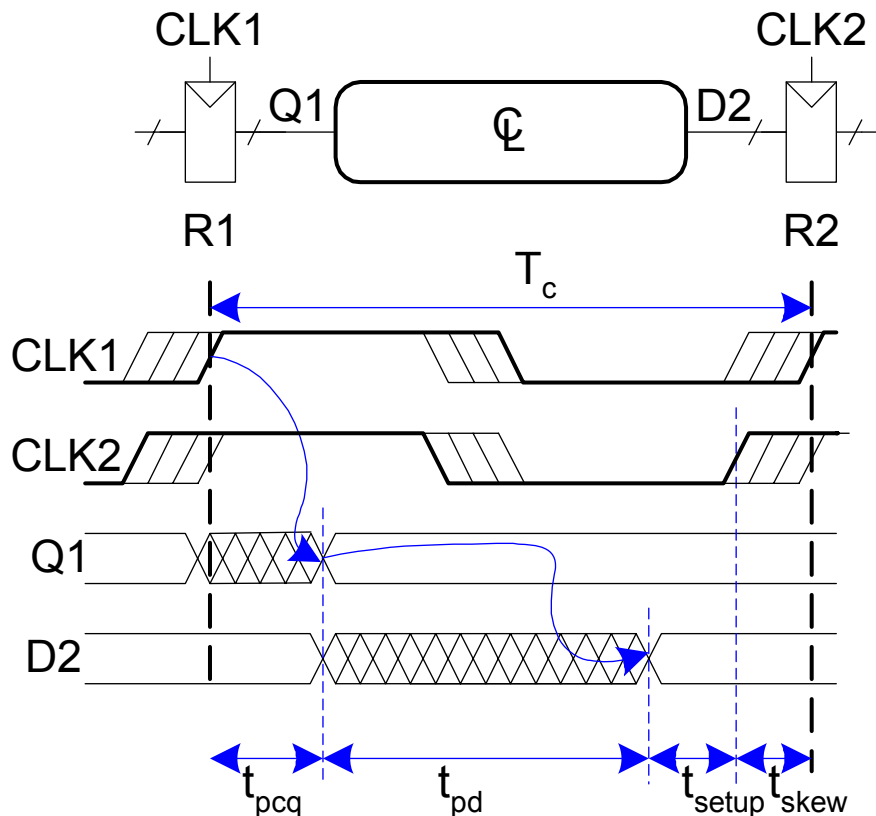
- In diesem Beispiel: CLK2 ist **früher** aktiv als CLK1



$$T_c \geq$$

# Setup-Zeitorderungen mit Taktverschiebung

- In diesem Beispiel: CLK2 ist **früher** aktiv als CLK1

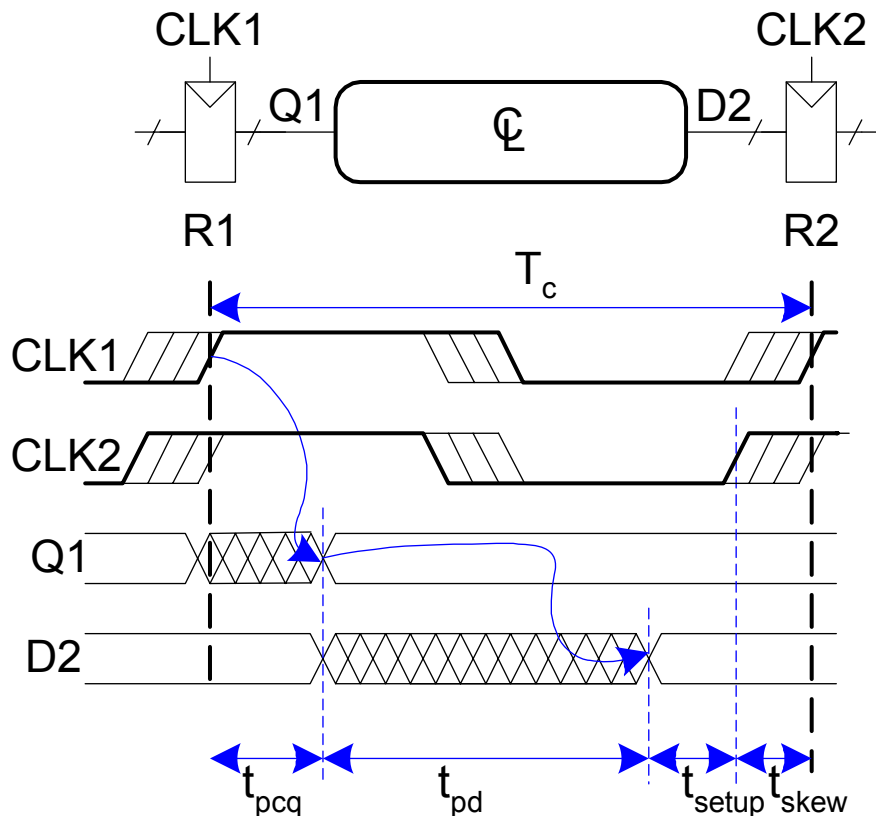


$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$
$$t_{pd} \leq$$



# Setup-Zeitorderungen mit Taktverschiebung

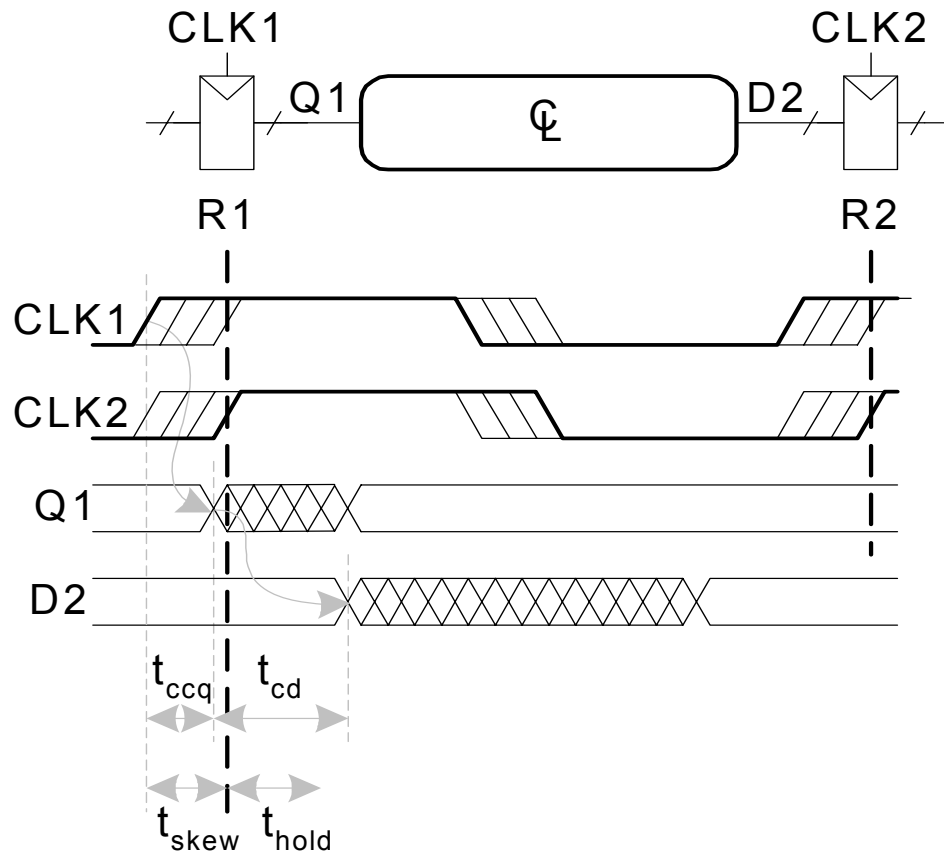
- In diesem Beispiel: CLK2 ist **früher** aktiv als CLK1



$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$
$$t_{pd} \leq T_c - (t_{pcq} + t_{setup} + t_{skew})$$

# Hold-Zeitorderungen mit Taktverschiebung

- In anderem Fall: CLK2 könnte **später** als CLK1 aktiviert werden

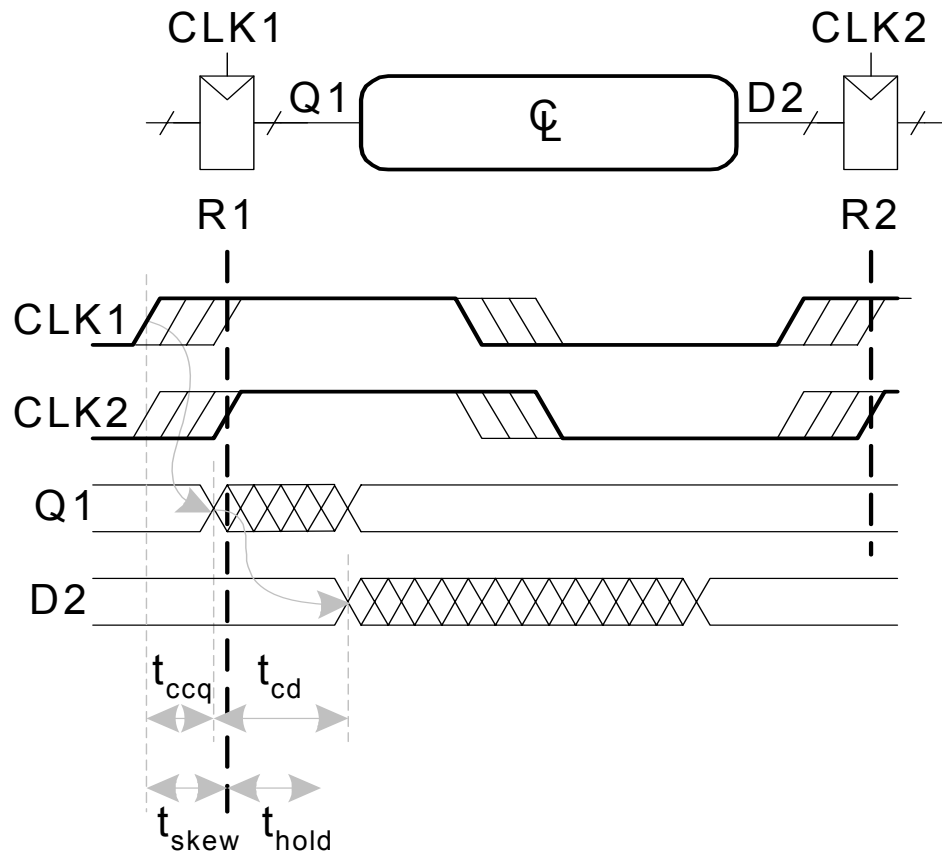


$$t_{\text{ccq}} + t_{\text{cd}} >$$

$$t_{\text{cd}} >$$

# Hold-Zeitanforderungen mit Taktverschiebung

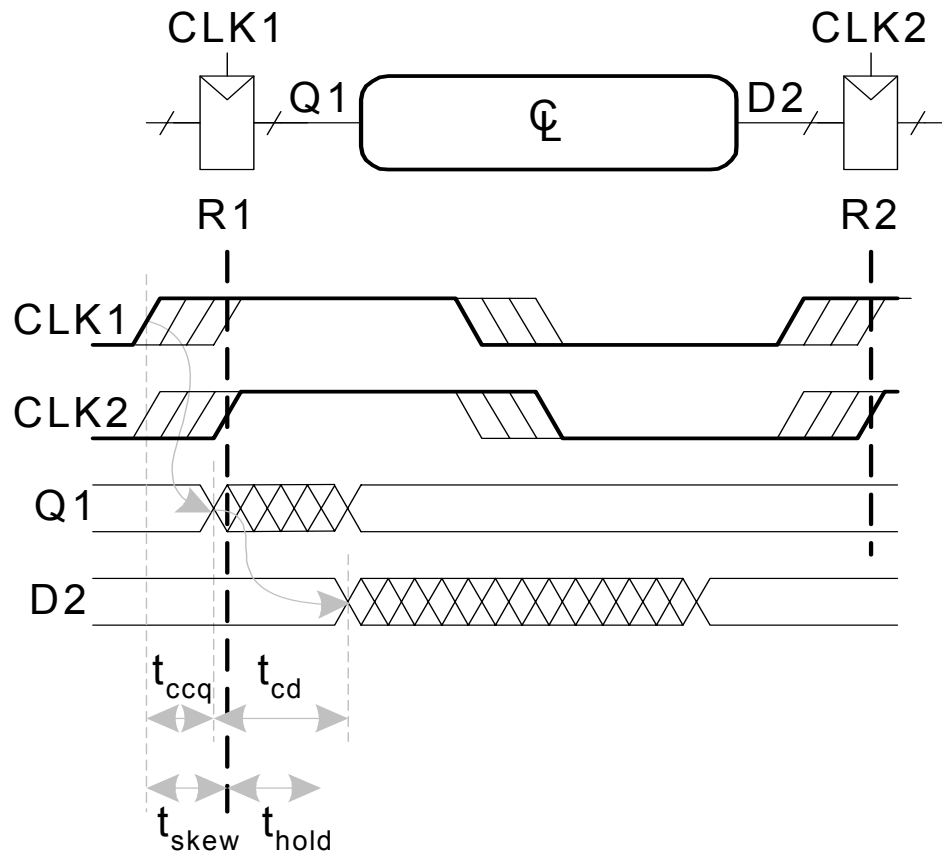
- In anderem Fall: CLK2 könnte **später** als CLK1 aktiviert werden



$$t_{ccq} + t_{cd} > t_{hold} + t_{skew}$$
$$t_{cd} >$$

# Hold-Zeit Anforderungen mit Taktverschiebung

- In anderem Fall: CLK2 könnte **später** als CLK1 aktiviert werden



$$t_{ccq} + t_{cd} > t_{hold} + t_{skew}$$

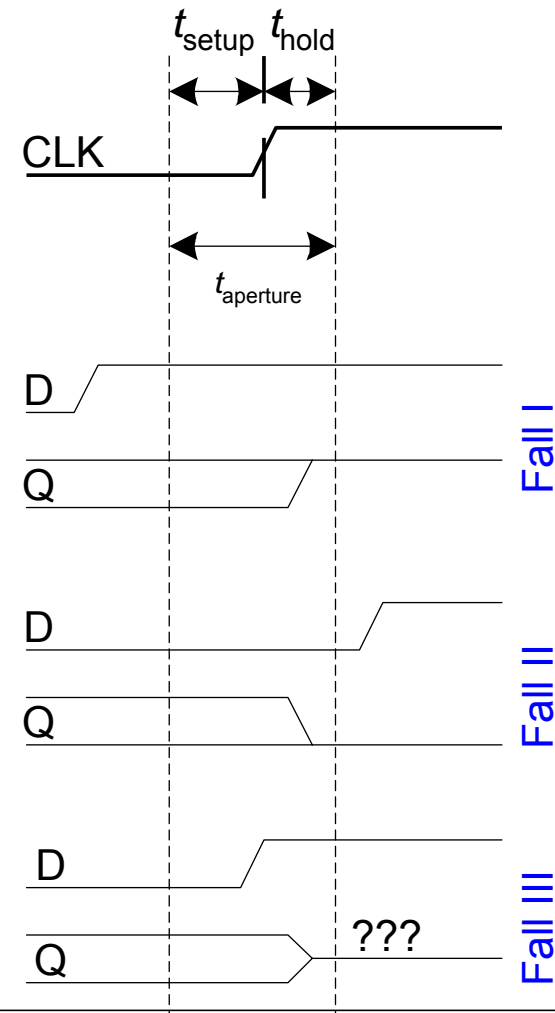
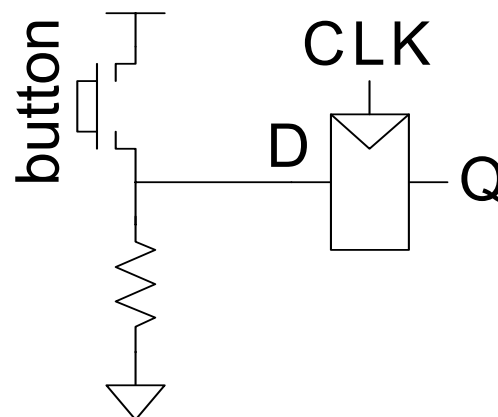
$$t_{cd} > t_{hold} + t_{skew} - t_{ccq}$$

# Verletzung der dynamischen Entwurfsdisziplin

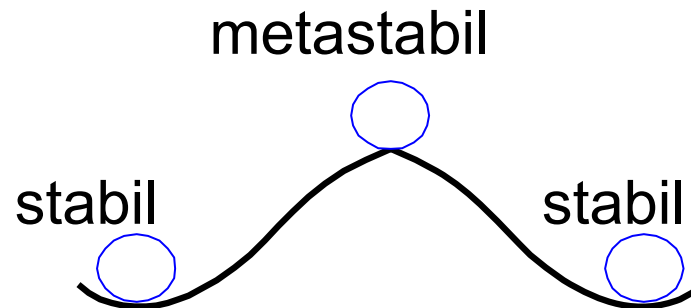


TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Asynchrone Eingänge können dynamische Disziplin verletzen
- Beispiel: Benutzereingaben



- Jedes bistabile Element hat zwei **stabile** Zustände und einen **metastabilen** dazwischen
- Ein **Flip-Flop-Ausgang** hat zwei stabile Zustände (0 und 1) und einen metastabilen Zustand
- Falls das Flip-Flop den **metastabilen** Zustand annimmt, kann es dort für **unbestimmte** Zeit verbleiben

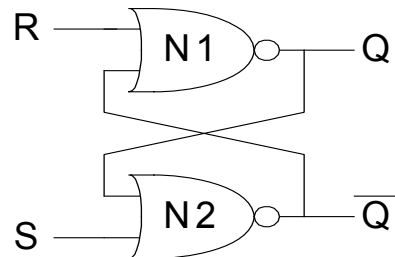


# Interner Aufbau eines Flip-Flops



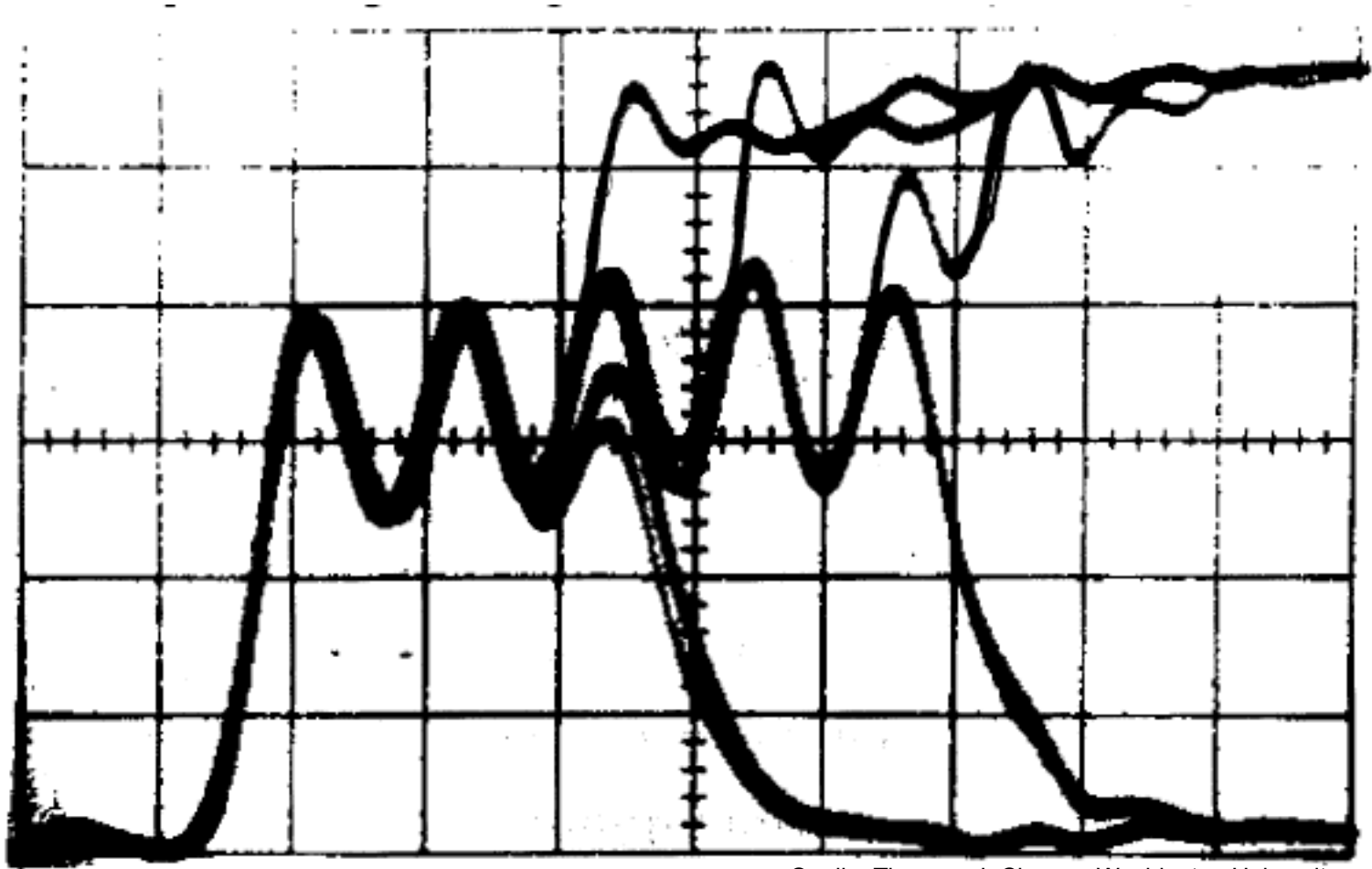
TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Ein Flip-Flop hat intern **Rückkopplungen**
- Falls Q zwischen 1 und 0 liegt:
  - ... wird es von den **kreuzgekoppelten** Gattern **irgendwann** auf 1 oder 0 getrieben
  - Je nachdem, an welchem Spannungspegel es **näher** lag



- Ein Signal wird als **metastabil** bezeichnet, wenn es noch nicht zu 1 oder 0 **aufgelöst** wurde

# Metastabilität: Beispiel



Quelle: Thomas. J. Chaney, Washington University



- Wenn Flip-Flop Eingang  $D$  zu einem zufälligen Zeitpunkt innerhalb der Abtastzeit wechselt
- ... wird Ausgang  $Q$  nach einer zufälligen Zeit  $t_{\text{res}}$  zu 0 oder 1 aufgelöst (*resolved*)
- **Wahrscheinlichkeit**, dass Ausgang  $Q$  nach einer Wartezeit  $t$  noch metastabil ist:

$$P(t_{\text{res}} > t) = (T_0/T_c) e^{-t/\tau}$$

$t_{\text{res}}$  : Zeit um Ausgang sicher nach 1 or 0 auszulösen

$T_0, \tau$  : Eigenschaften der Schaltung

$T_c$  : Taktperiode

- Intuitiv
  - $T_0/T_c$  ist **Wahrscheinlichkeit**, dass Eingang zu einem **ungünstigen** Zeitpunkt schaltet
    - Innerhalb der Abtastzeit, sinkt mit wachsender Taktperiode  $T_c$

$$P(t_{\text{res}} > t) = (T_0/T_c) e^{-t/\tau}$$

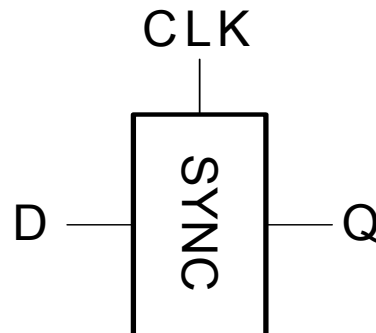
- Die Zeitkonstante  $\tau$  gibt an, wie schnell Flip-Flop sich aus dem metastabilen Zustand **wegbewegen** kann
  - Hängt von der Verzögerung durch die kreuzgekoppelten Gatter ab

$$P(t_{\text{res}} > t) = (T_0/T_c) e^{-t/\tau}$$

- Kurzfassung: Wenn man nur **lange** genug wartet, wird der Ausgang sicher zu 0 oder 1 aufgelöst

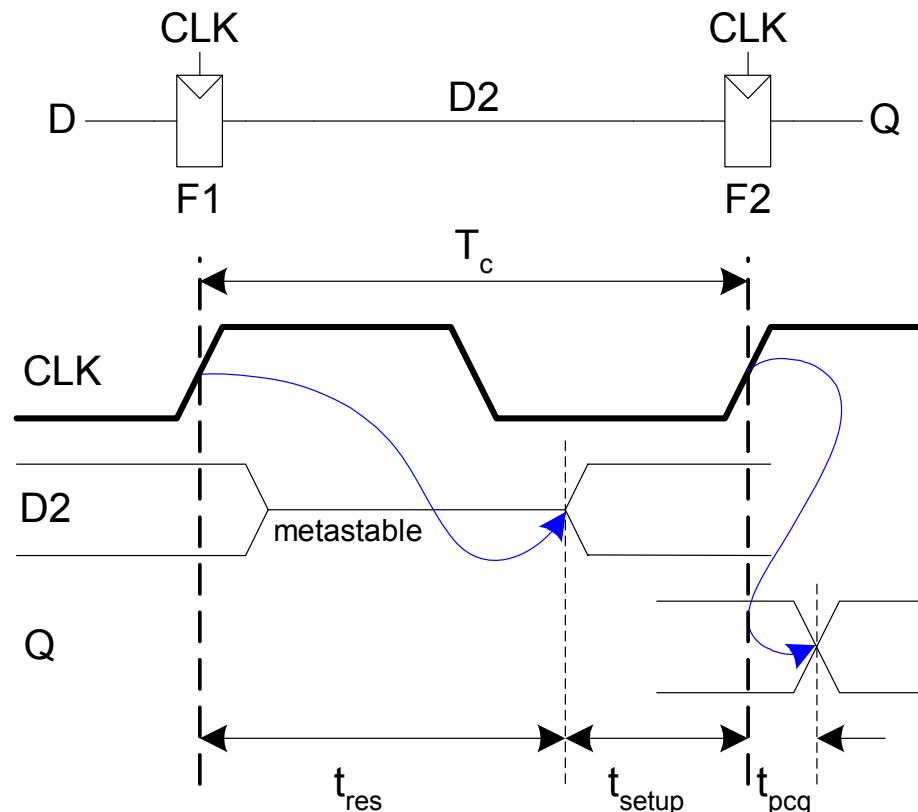
# Synchronisierer (*synchronizer*)

- Asynchrone Eingänge (*D*) lassen sich praktisch **nicht** ganz vermeiden
  - Benutzerschnittstellen
  - Systeme mit mehreren interagierenden Taktsignalen
- Ziel eines **Synchronisierers**
  - **Reduziere** die Wahrscheinlichkeit für metastabilen Zustand
  - Liefere an Q mit **hoher** Wahrscheinlichkeit gültige Werte
- Metastabilität kann aber **nie** völlig ausgeschlossen werden



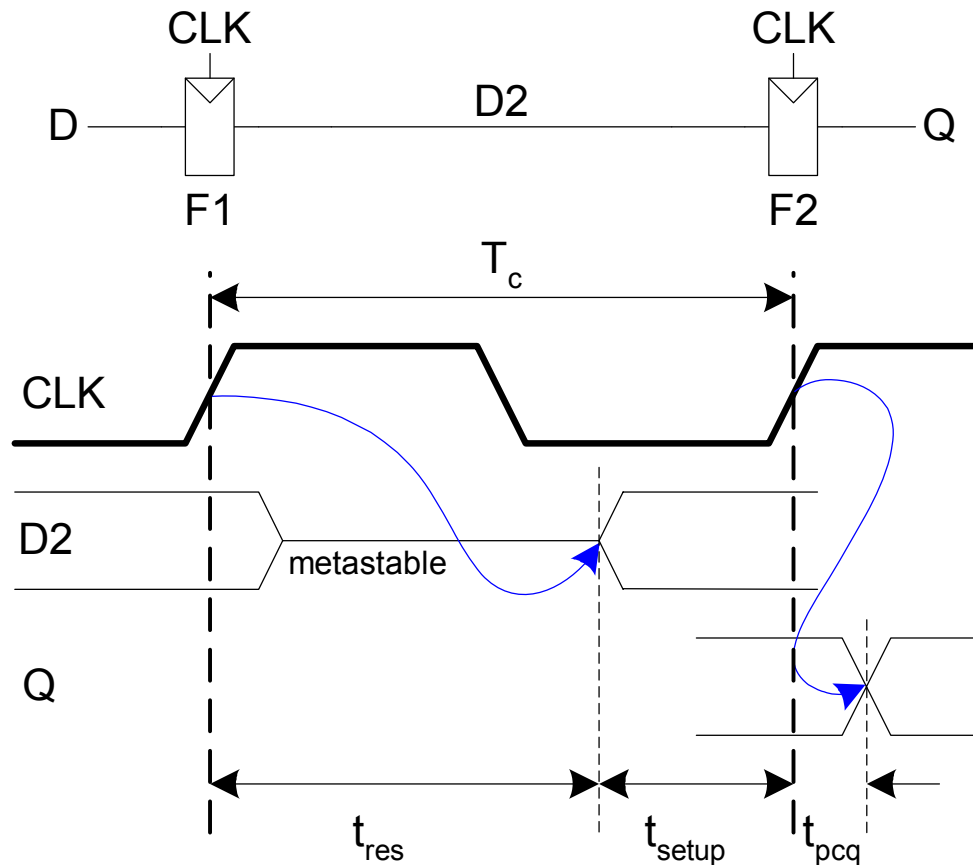
# Interner Aufbau eines Synchronisierers

- Aufgebaut aus **Reihenschaltung** von Flip-Flops
- Annahme: Eingang der wechselt während der Abtastzeit von Flip-Flop F1
- Synchronisation **gelingt**, wenn Signal D2 **innerhalb** von  $(T_c - t_{\text{setup}})$  zu 0 oder 1 aufgelöst wird



## Wahrscheinlichkeit für Scheitern der Synchronisation

Für jede Änderung des Eingangs D:  **$P(\text{Scheitern}) = (T_0/T_c) e^{-(T_c - t_{\text{setup}})/\tau}$**



# Mittlere Betriebsdauer zwischen Ausfällen (*mean time between failures, MTBF*)

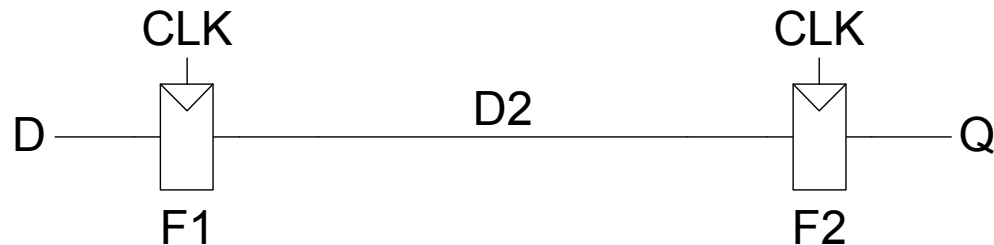
- Bei Änderung des Eingangssignals **einmal** pro Sekunde
  - Ausfallwahrscheinlichkeit des Synchronisierers pro Sekunde ist  $P(\text{Scheitern})$
- Bei Änderung des Eingangssignals  **$N$ -mal** pro Sekunde
  - Ausfallwahrscheinlichkeit des Synchronisierers pro Sekunde ist

$$P(\text{Scheitern})/s = (NT_0/T_c) e^{-(T_c - t_{\text{setup}})/\tau}$$

- Im **Durchschnitt** scheitert die Synchronisation also alle  
 $1/[P(\text{Scheitern})/s]$  Sekunden
- Wird auch genannt “**Mittlere Betriebsdauer zwischen Ausfällen**” (MTBF)

$$\text{MTBF} = 1/[P(\text{Scheitern})/s] = (T_c/NT_0) e^{(T_c - t_{\text{setup}})/\tau}$$

# Beispiel: Synchronisierer



- Annahmen:  
 $T_c = 1/500 \text{ MHz} = 2 \text{ ns}$   
 $T_0 = 150 \text{ ps}$   
 $N = 10 \text{ Änderungen pro Sekunde}$   
 $\tau = 200 \text{ ps}$   
 $t_{\text{setup}} = 100 \text{ ps}$

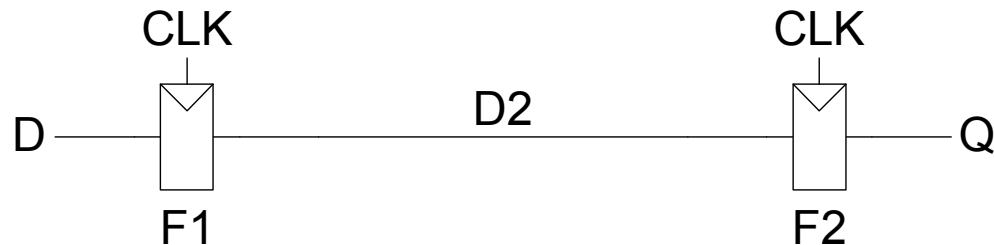
- Ausfallwahrscheinlichkeit? MTBF?

$$P(\text{Scheitern}) =$$

$$P(\text{Scheitern})/\text{s} =$$

$$\text{MTBF} =$$

# Beispiel: Synchronisierer



- Annahmen:  
 $T_c = 1/500 \text{ MHz} = 2 \text{ ns}$   
 $T_0 = 150 \text{ ps}$   
 $N = 10 \text{ Änderungen pro Sekunde}$   
 $\tau = 200 \text{ ps}$   
 $t_{\text{setup}} = 100 \text{ ps}$

- Ausfallwahrscheinlichkeit? MTBF?

$$P(\text{Scheitern}) = (150 \text{ ps} / 2 \text{ ns}) e^{-(1.9 \text{ ns}) / 200 \text{ ps}} = 5.6 \times 10^{-6}$$

$$P(\text{Scheitern}) / \text{s} = 10 \times (5.6 \times 10^{-6}) = 5.6 \times 10^{-5} / \text{s}$$

$$\text{MTBF} = 1 / [P(\text{Scheitern}) / \text{s}] \approx 5 \text{ Stunden}$$



## ▪ Zwei Arten von Parallelität

### ▪ Räumliche Parallelität

- Vervielfachte Hardware bearbeitet mehrere Aufgaben gleichzeitig

### ▪ Zeitliche Parallelität

- Aufgabe wird in mehrere Unteraufgaben aufgeteilt
- Unteraufgaben werden parallel ausgeführt
- Beispiel: Fließbandprinzip bei Autofertigung
  - Nur eine Station für einen Arbeitsschritt
  - Aber alle unterschiedlichen Arbeitsschritte für mehrere Autos werden parallel ausgeführt
- Auch genannt: Pipelining

# Parallelität: Grundlegende Begriffe

- Einige Definitionen:
  - **Datensatz:** Vektor aus Eingabewerten, die zu einem Vektor aus Ausgabewerten bearbeitet werden
  - **Latenz:** Zeit von der Eingabe eines Datensatzes bis zur Ausgabe der zugehörigen Ergebnisse
  - **Durchsatz:** Die Anzahl von Datensätzen die pro Zeiteinheit bearbeitet werden können
- Parallelität erhöht Durchsatz

# Beispiel Parallelität: Plätzchen backen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Weihnachtszeit steht vor der Tür, also rechtzeitig anfangen!
- **Annahmen**
  - Genug Teig ist fertig
  - 5 Minuten um ein Blech mit Teig zu bestücken
  - 15 Minuten Backzeit
- **Vorgehensweise**
  - Ein Blech nach dem anderen vorbereiten und backen

Latenz =

Durchsatz =

# Beispiel Parallelität: Plätzchen backen (seriell)



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Weihnachtszeit steht vor der Tür, also rechtzeitig anfangen!
- **Annahmen**
  - Genug Teig ist fertig
  - 5 Minuten um ein Blech mit Teig zu bestücken
  - 15 Minuten Backzeit
- **Vorgehensweise**
  - Ein Blech nach dem anderen vorbereiten und backen

Latenz =  $5 + 15 = 20$  Minuten =  $1/3$  h

Durchsatz = 1 Blech alle 20 Minuten =  $3$  Bleche/h

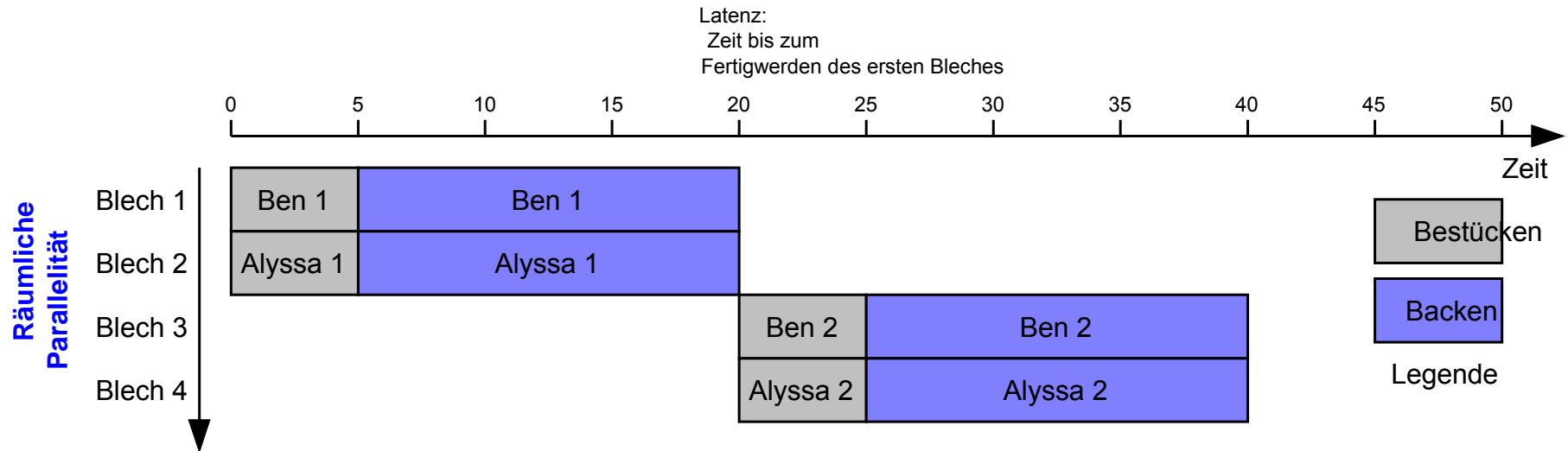
# Beispiel Parallelität: Plätzchen backen (parallel)



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- **Gleiche** Annahmen wie eben
  - 5 Minuten Blech bestücken, 15 Minuten Backen
- **Alternative** Vorgehensweisen
  - **Räumliche Parallelität:** **Zwei** Bäcker (Ben & Alyssa), jeder mit einem **eigenen** Ofen
  - **Zeitliche Parallelität:** Aufteilen der Keksherstellung in **Unteraufgaben**
    - Blech bestücken
    - Backen
    - Nächstes Blech bestücken, **während** erstes noch im Ofen gebacken wird
- Latenz und Durchsatz?

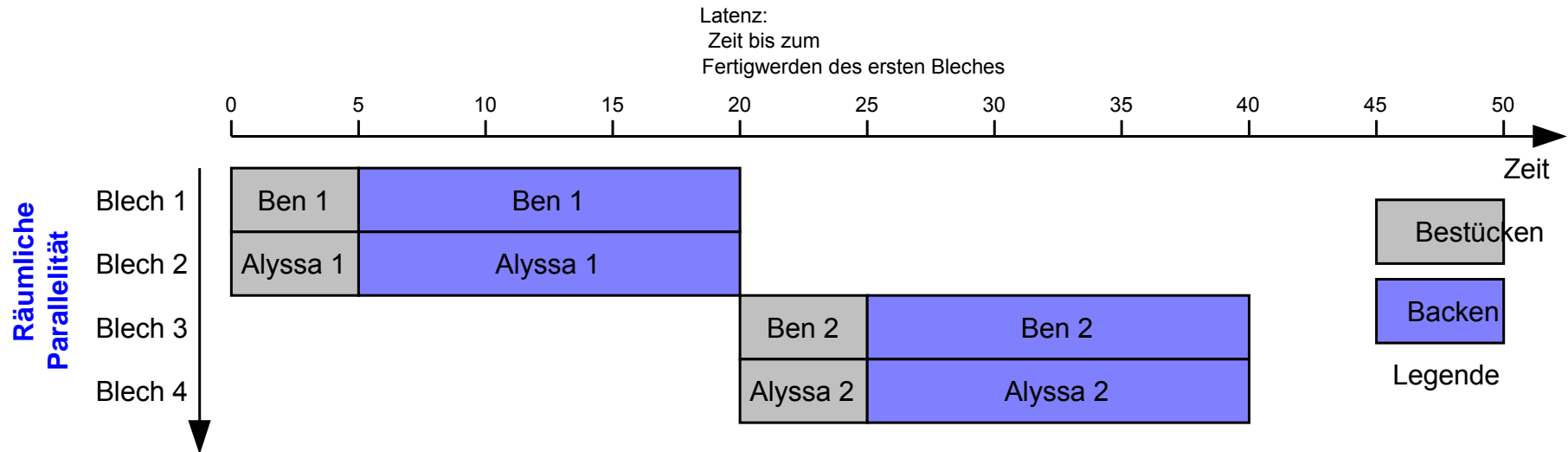
# Räumliche Parallelität



Latenz =

Durchsatz =

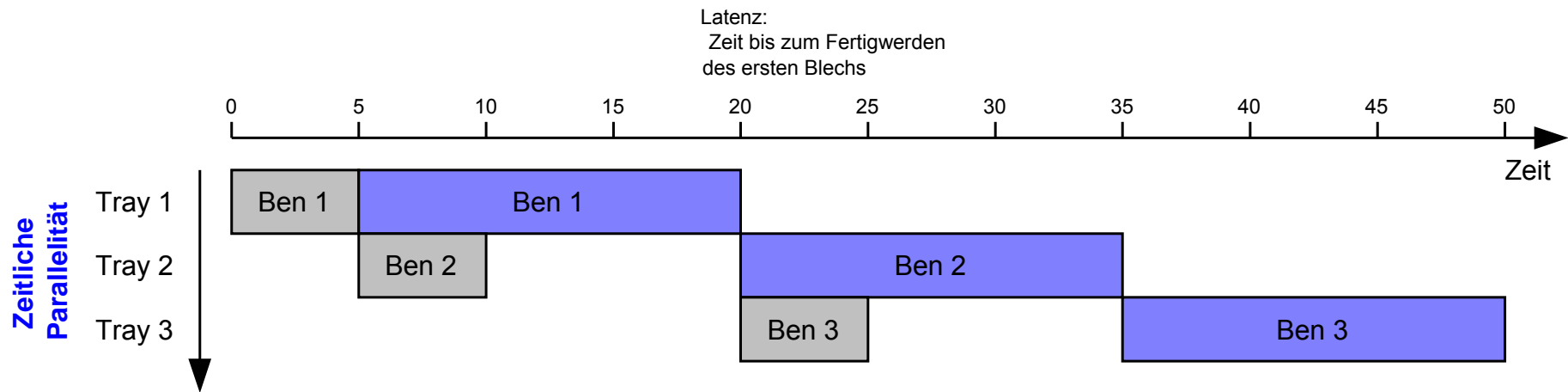
# Räumliche Parallelität



$$\text{Latenz} = 5 + 15 = 20 \text{ Minuten} = \frac{1}{3} \text{ h}$$

$$\text{Durchsatz} = 2 \text{ Bleche alle 20 Minuten} = 6 \text{ Bleche/h}$$

# Zeitliche Parallelität

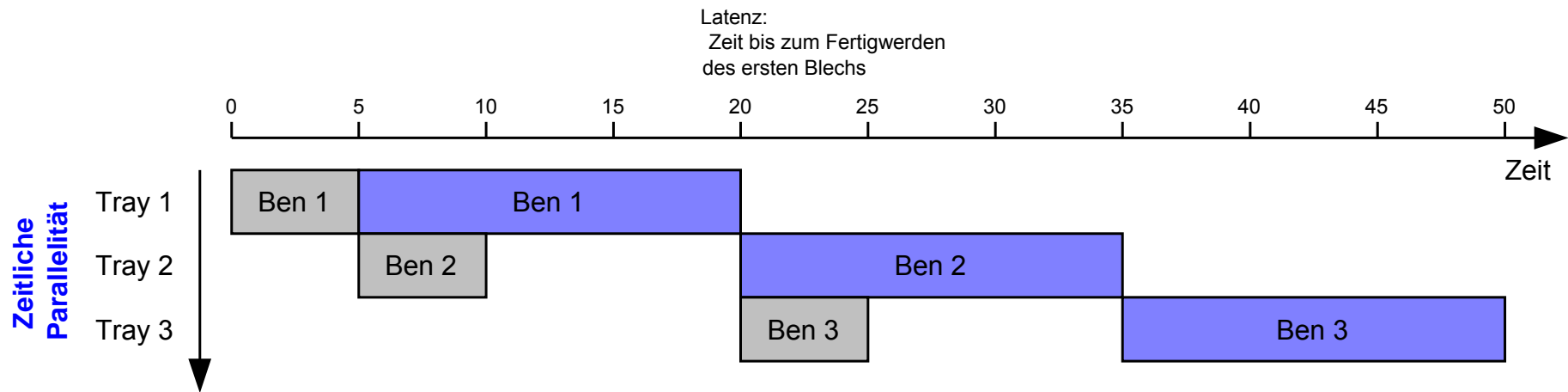


Latenz =

Durchsatz =



# Zeitliche Parallelität

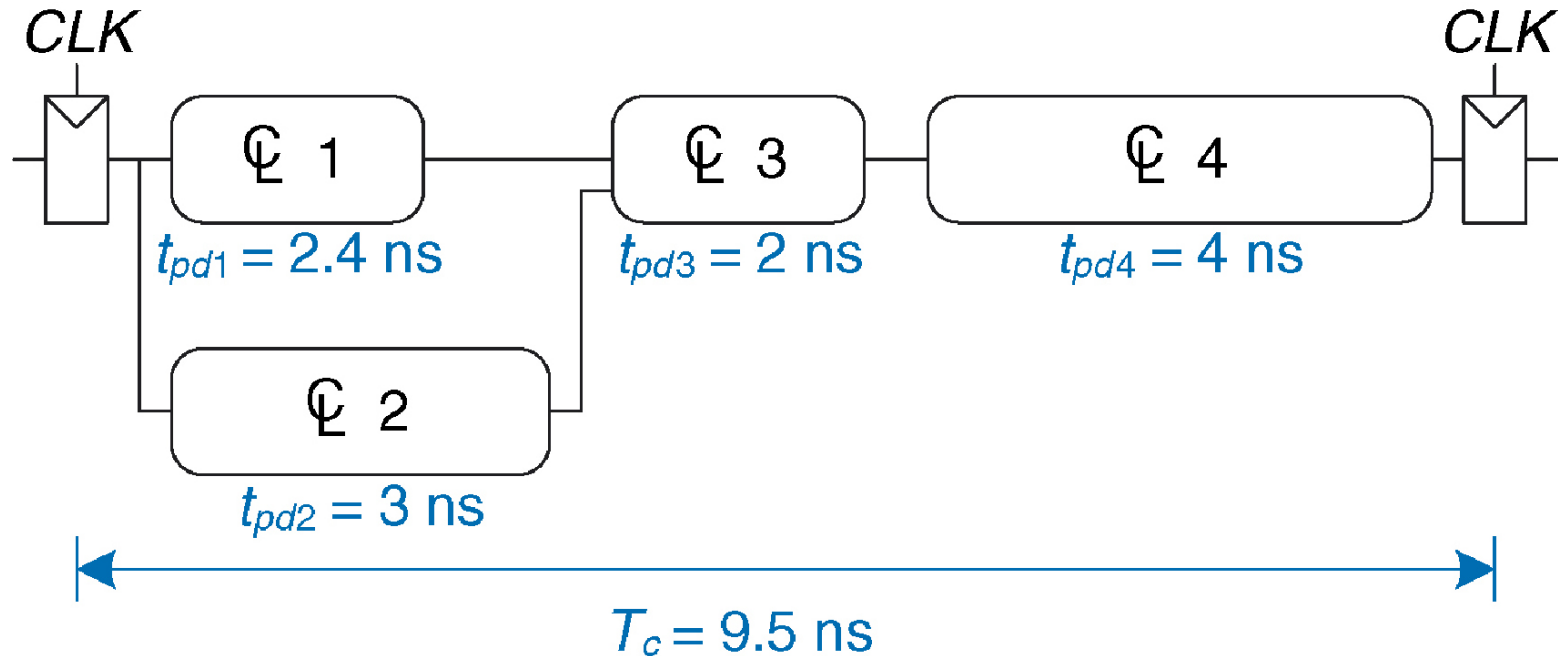


$$\text{Latenz} = 5 + 15 = 20 \text{ Minuten} = \frac{1}{3} \text{ h}$$

$$\text{Durchsatz} = 1 \text{ Blech alle } 15 \text{ Minuten} = 4 \text{ Bleche/h}$$

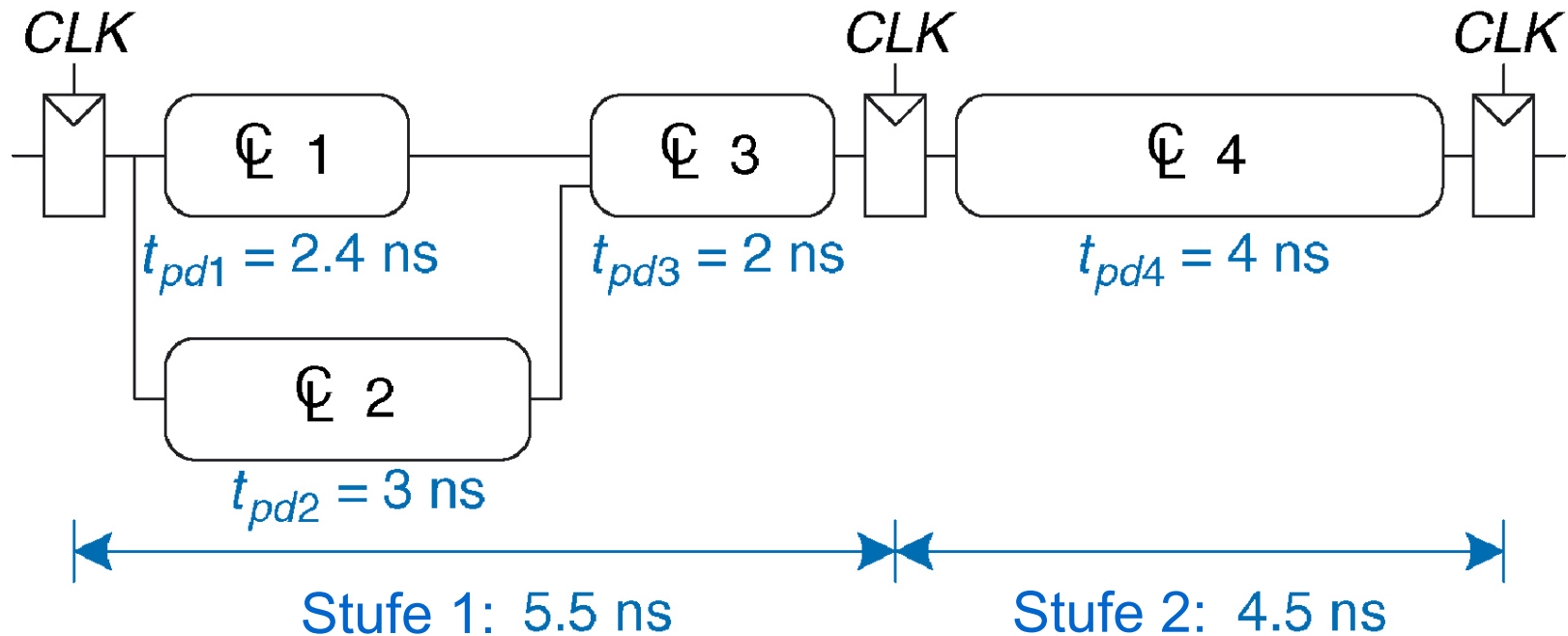
- Zeitliche und räumliche Parallelität können miteinander **kombiniert** werden
- Hier:
  - **Zwei** Bäcker und Öfen
  - Nächstes Blech bestücken **während** altes gebacken wird
- Latenz = 20 Minuten
- Durchsatz = 8 Bleche/h

# Schaltung ohne Pipelining



- Kritischer Pfad durch Elemente 2, 3, 4: 9 ns
- $t_{\text{setup}} = 0,2 \text{ ns}$  und  $t_{\text{pcq}} = 0,3 \text{ ns} \rightarrow T_c = 9 + 0,2 + 0,3 = 9,5 \text{ ns}$
- Latenz = 9,5ns ; Durchsatz =  $1 / 9,5\text{ns} = 105 \text{ MHz}$

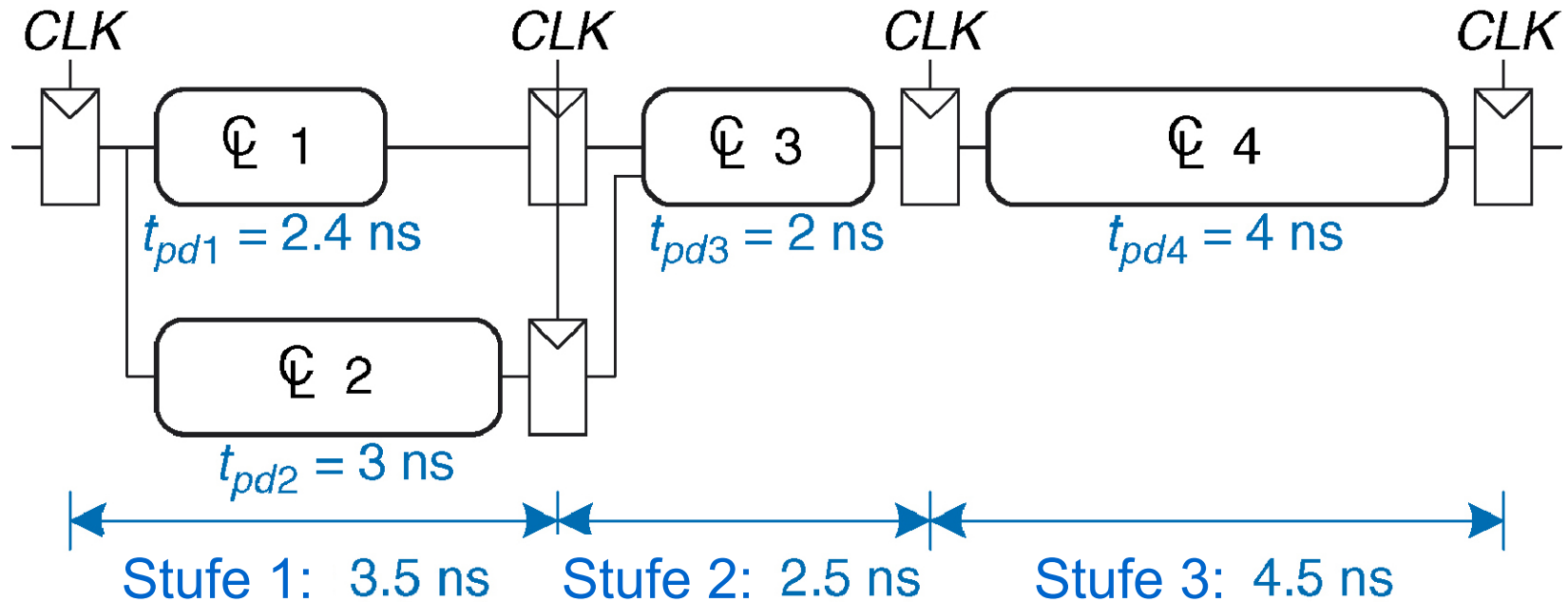
# Schaltung mit zweistufiger Pipeline



- Stufe 1:  $3 + 2 + 0,2 + 0,3 = 5,5 \text{ ns}$
- $\rightarrow T_c = 5,5 \text{ ns}$
- Latenz = 2 Takte = 11 ns
- Durchsatz =  $1 / 5,5 \text{ ns} = 182 \text{ MHz}$

Stufe 2:  $4 + 0,2 + 0,3 = 4,5 \text{ ns}$

# Schaltung mit dreistufiger Pipeline



- $T_c = 4,5 \text{ ns}$
- Latenz = 3 Takte = 13,5 ns
- Durchsatz =  $1 / 4,5 \text{ ns} = 222 \text{ MHz}$

# Diskussion Pipelining

- Mehr Pipelinestufen
  - **Höherer** Durchsatz (mehr Ergebnisse pro Zeiteinheit)
  - aber auch **höhere** Latenz (länger warten auf das erste Ergebnis)
  - → Lohnt sich nur, wenn **viele** Datensätze bearbeitet werden müssen
- Klappt aber **nicht** immer
- Problem: **Abhängigkeiten**
- Beispiel Kekse: **Erstmal** schauen wie ein Blech geworden ist, **bevor** das nächste bestückt wird
- Wird noch intensiv im 7. Kapitel behandelt (Prozessorarchitektur)