

Digitaltechnik

3 Sequenzielle Schaltungen



Revision 1.1

Transitionssysteme

Synchroner sequenzieller Entwurf

Timing-Analyse

Pipelining

Mealy und Moore Maschinen

Zustandsmaschinen in Verilog

- ▶ **Kombinatorisch** (engl. „combinational“)

Zustandsfrei

Flyweight Pattern in OOP: Objekte ohne Zustand

Funktionale Programmierung (Lisp, ML): keine Seiteneffekte

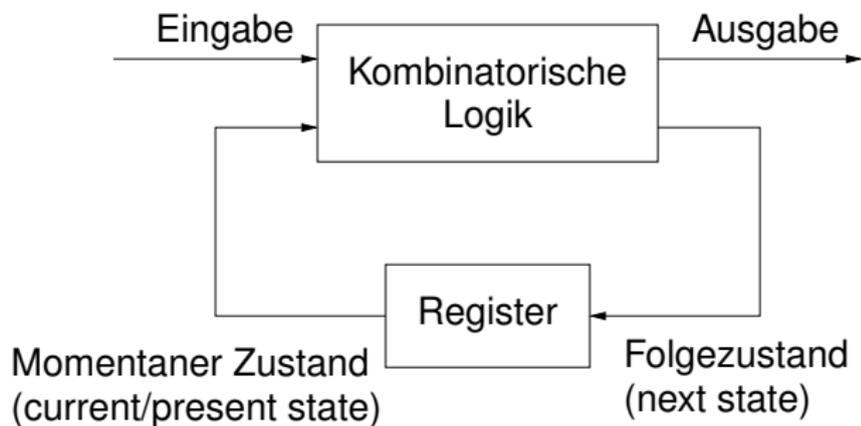
Ausgaben sind funktional von den Eingaben abhängig

- ▶ **Sequentiell** (engl. „sequential“)

Zustandsbehaftet

vgl. Objektorientierte Programmierung (OOP): Objekte mit Zustand

vergangenes Verhalten und **Eingaben** bestimmen den nächsten Zustand



(hier nur Hardwaresicht, kaum Analogie zu OOP)

- ▶ **Synchron** (engl. synchronous)
globales Taktsignal (engl. *clock*)
Zustandsänderungen zu fest spezifizierten Zeiten
✓ Vereinfachter Entwurf
- ▶ **Asynchron** (engl. asynchronous)
Zustandsänderung zu beliebigen Zeitpunkten
Auch Übergang zwischen unterschiedlichen *clock domains*
✗ Behandlung von Hazards ist sehr schwierig

Hier: Nur synchrone Schaltungen

Definition (Transitionssystem)

Ein Transitionssystem ist ein Tripel $\langle S, I, T \rangle$ mit

- ▶ S : Menge der Zustände
- ▶ $I \subseteq S$: Initialzustände
- ▶ $T : S \times S$: Übergangsrelation

Definition (Transitionssystem)

Ein Transitionssystem ist ein Tripel $\langle S, I, T \rangle$ mit

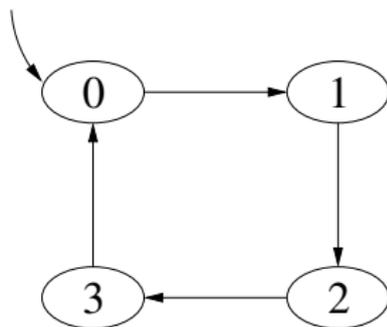
- ▶ S : Menge der Zustände
- ▶ $I \subseteq S$: Initialzustände
- ▶ $T : S \times S$: Übergangsrelation

Definition (Berechnung)

Eine Berechnung ist eine Folge s_0, \dots, s_n mit $s_i \in S$ und

- ▶ $s_0 \in I$
- ▶ $\forall i : (s_i, s_{i+1}) \in T$

Vgl. *endliche Automaten* aus der theoretischen Informatik



- ▶ Menge der Zustände S : die Zahlen 0 bis 3
- ▶ ein Initialzustand: $I = \{0\}$
- ▶ Transitionsrelation: der Zähler wird um eins erhöht modulo 4

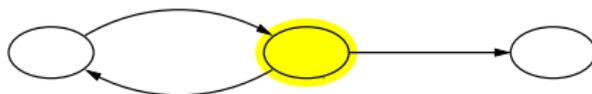
$$T = \{(0, 1), (1, 2), (2, 3), (3, 0)\}$$

- ▶ Solche Automaten beschreiben i. Allg. **nicht-deterministische Systeme**:

1. Deadends: Zustände ohne Nachfolgezustand



2. Zustände mit mehreren möglichen Nachfolgern

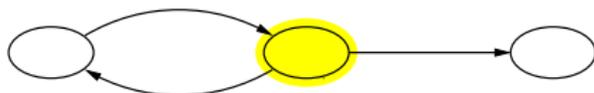


- ▶ Solche Automaten beschreiben i. Allg. **nicht-deterministische Systeme**:

1. Deadends: Zustände ohne Nachfolgezustand



2. Zustände mit mehreren möglichen Nachfolgern



- ▶ Für die Hardware-Generierung brauchen wir Determinismus:

Jeder Zustand hat genau einen Nachfolger

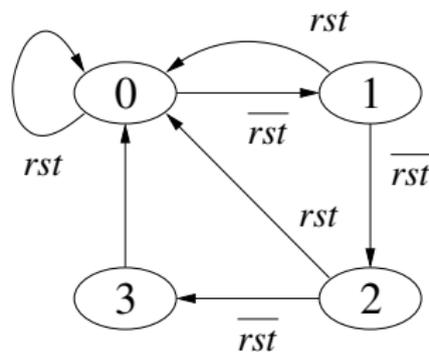
- ▶ Was machen wir mit den Eingängen unserer Schaltung?
- ▶ Wir führen ein **Eingabealphabet** E ein
- ▶ Transitionsfunktion $\delta : (S \times E) \rightarrow S$
Initialzustandsprädikat: $I : (S \times E)$
- ▶ Spezialfall einer Transitionsrelation:

$$(x, y) \in T \iff \exists e \in E. \delta(x, e) = y$$

- ▶ Was machen wir mit den Eingängen unserer Schaltung?
- ▶ Wir führen ein **Eingabealphabet** E ein
- ▶ Transitionsfunktion $\delta : (S \times E) \rightarrow S$
Initialzustandsprädikat: $I : (S \times E)$
- ▶ Spezialfall einer Transitionsrelation:

$$(x, y) \in T \iff \exists e \in E. \delta(x, e) = y$$

- ✓ Vorteil: für gegebenes e offensichtlich deterministisch!
Keine Deadends!

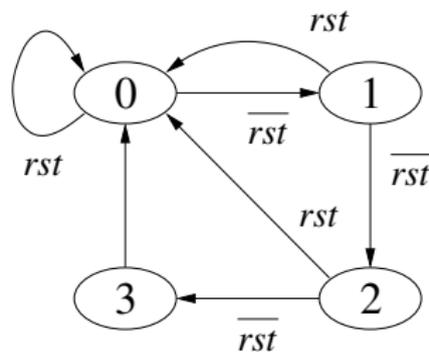


$$S = \{0, \dots, 3\}$$

$$E = \{rst, \overline{rst}\}$$

Zähler wird auf 0 zurückgesetzt wenn rst gilt,
ansonsten wird Zähler um eins erhöht modulo 4:

$$\delta(s, rst) = \begin{cases} 0 & : rst \\ (s + 1) \bmod 4 & : \text{sonst} \end{cases}$$

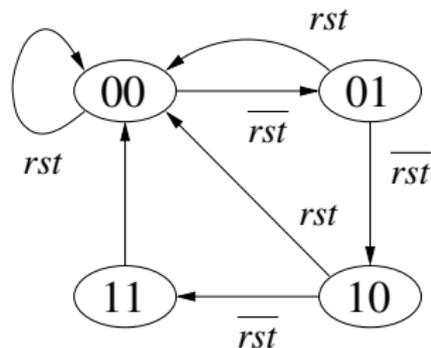


$$S = \{0, \dots, 3\}$$
$$E = \{rst, \overline{rst}\}$$

Zähler wird auf 0 zurückgesetzt wenn rst gilt,
ansonsten wird Zähler um eins erhöht modulo 4:

$$\delta(s, rst) = \begin{cases} 0 & : rst \\ (s + 1) \bmod 4 & : \text{sonst} \end{cases}$$

Fehlt noch: **Bits!**



- ▶ Zustände (ab) sind Belegung von D-Flipflops mit $a, b \in \{0, 1\}$
 Binärkodierung: $0 \mapsto 00$, $1 \mapsto 01$, $2 \mapsto 10$, $3 \mapsto 11$
- ▶ Transition $(ab) \xrightarrow{c} (a'b')$ besteht aus
 momentaner Zustandsbelegung ab ,
 Nachfolgezustandsbelegung $a'b'$ und
 Übergangsbedingung c

$$\delta = \{ \begin{array}{l} (00, 0, 01), (00, 1, 00), \\ (01, 0, 10), (01, 1, 00), \\ (10, 0, 11), (10, 1, 00), \\ (11, 0, 00), (11, 1, 00) \end{array} \}$$

Beispiele:

- ▶ $\delta(00, 0) = 01$,
- ▶ $\delta(10, 1) = 00$

Zustand	a	b	rst	a'	b'
0	0	0	0	0	1
1	0	1	0	1	0
2	1	0	0	1	1
3	1	1	0	0	0
–	–	–	1	0	0

damit ergeben sich die Zustandsgleichungen:

$$a' \equiv \overline{rst} \cdot (a \oplus b)$$

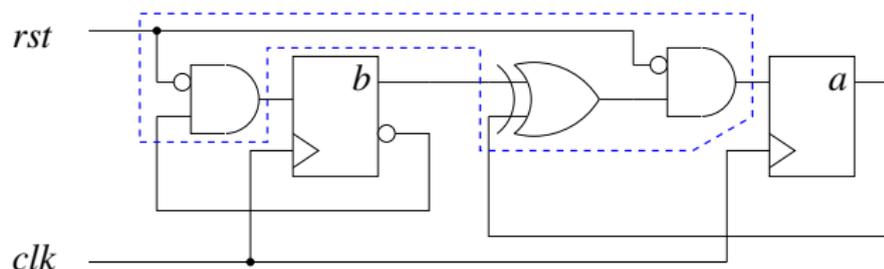
$$b' \equiv \overline{rst} \cdot \bar{b}$$

Zeit	0	1	2	3	4	5	6	...
<i>rst</i>	1	0	0	0	0	0	0	
<i>a</i>	1	0	0	1	1	0	0	
<i>b</i>	0	0	1	0	1	0	1	

Zeit	0	1	2	3	4	5	6	...
<i>rst</i>	1	0	0	0	0	0	0	
<i>a</i>	1	0	0	1	1	0	0	
<i>b</i>	0	0	1	0	1	0	1	

Zeit	0	1	2	3	4	5	6	...
<i>rst</i>	1	0	0	1	0	0	0	
<i>a</i>	1	0	0	1	0	0	1	
<i>b</i>	1	0	1	0	0	1	0	

1. Baue die Transitionsfunktion als kombinatorische Schaltung
2. Der Zustand wird in Flipflops gespeichert
3. Eine globale Clock gibt vor, wann eine Transition ausgeführt wird
4. Der Initialzustand wird bei einem Reset berechnet



- ▶ Globaler Takt: *clk* (wird oft weggelassen)
- ▶ Nur eine Eingabe: *rst*
- ▶ Next-State Logik: Gestrichelt umrandete Gatter
- ▶ Hier keine Ausgabelogik

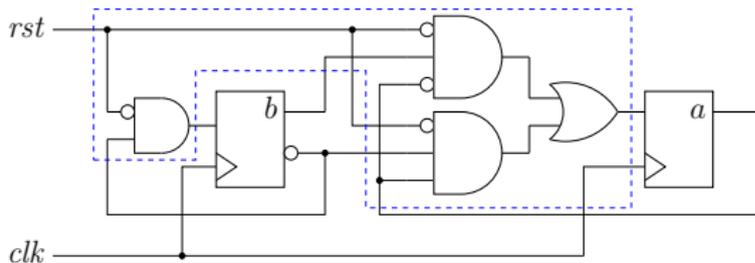
Karnaugh Diagramme für Zustandsgleichungen

	b			
	1	0	0	0
a	1	0	0	0
	rst			

	b			
	0	1	0	0
a	1	0	0	0
	rst			

$$b' \equiv \overline{rst} \cdot \bar{b}$$

$$a' \equiv \overline{rst} \cdot a \cdot \bar{b} + \overline{rst} \cdot \bar{a} \cdot b$$



Wie schnell dürfen wir die Schaltung takten?

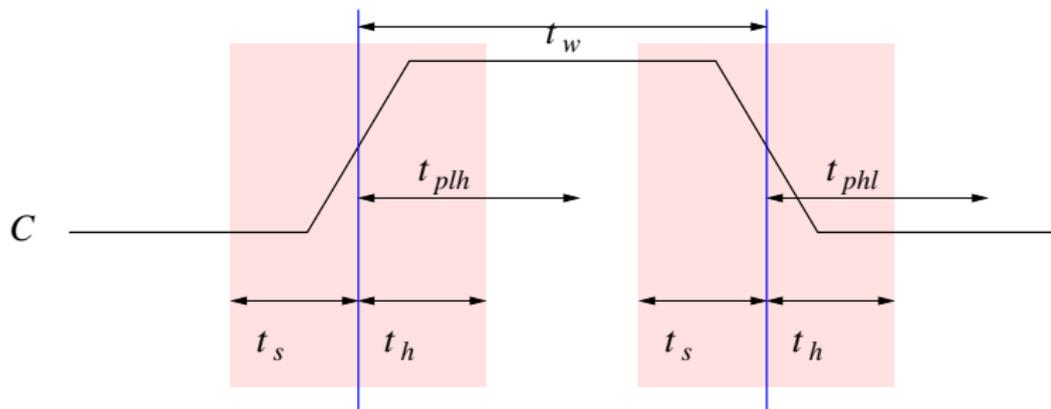
Wie schnell dürfen wir die Schaltung takten?

Was passiert, wenn wir zu schnell takten?

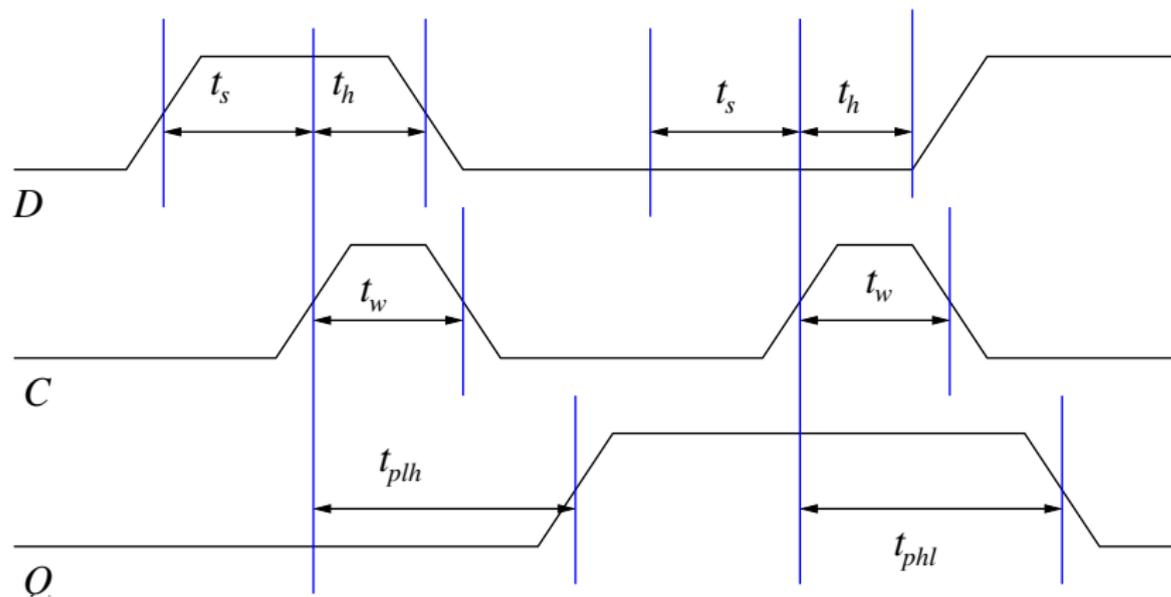
Wie schnell dürfen wir die Schaltung takten?

Was passiert, wenn wir zu schnell takten?

Erinnerung: **metastabile Zustände** in den D-Flipflops



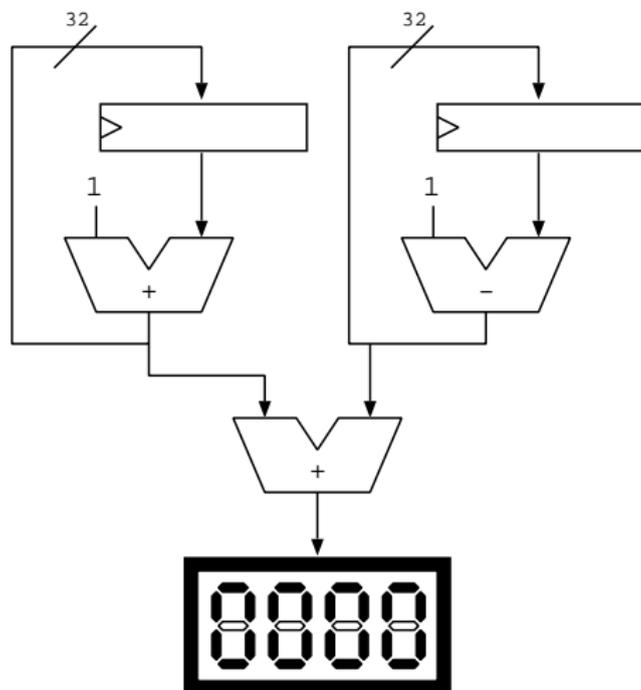
- ▶ Eingang stabil während *setup* Phase t_s vor der Flanke
- ▶ Eingang stabil während *hold* Phase t_h nach der Flanke
- ▶ Ausgang stabil erst nach *propagation* Phase t_{plh} bzw. t_{phl} nach der Flanke
- ▶ Minimale Clock-Signallänge (*width*) t_w



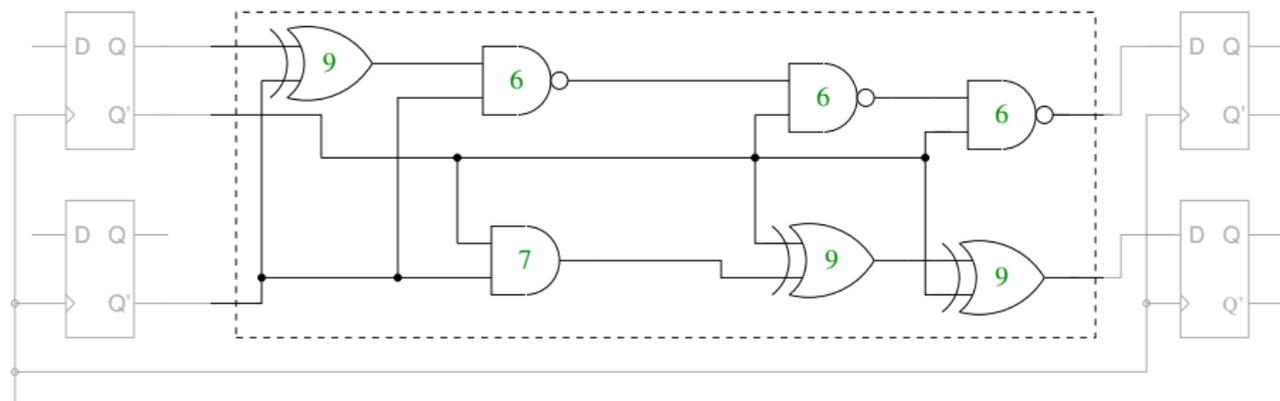
$$\begin{aligned} & \text{Setup+Holdzeiten} \\ + & \text{ Propagation Delay der Flipflops} \\ + & \text{ Verzögerung der Transitionsfunktion} \\ & \text{(längster Pfad!)} \\ \hline = & \text{ Zykluszeit} \end{aligned}$$

Die max. Betriebsfrequenz ist der **Kehrwert der Zykluszeit**

Overclocker Demo



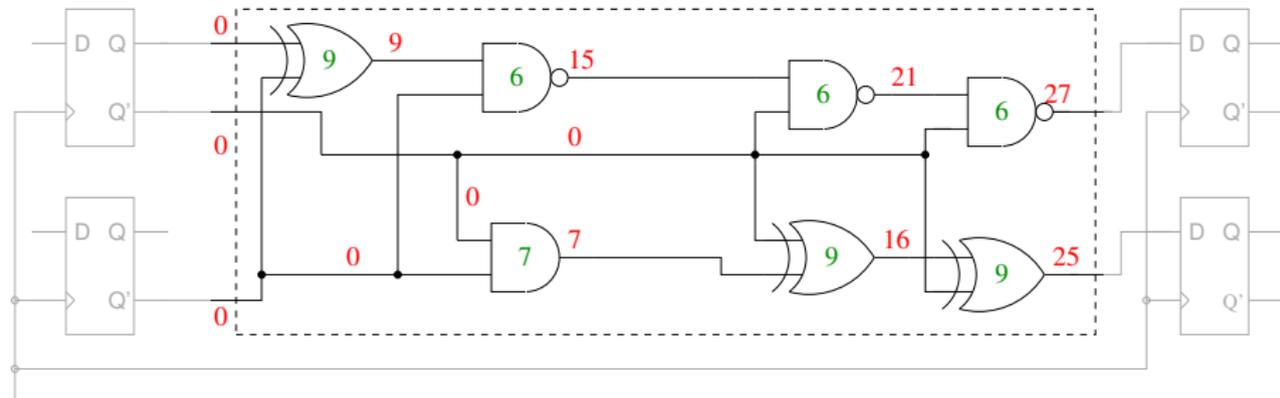
Beispiel Frequenzberechnung



Bauteil	t_p	t_s
AND	7 ns	-
NAND	6 ns	-
OR	6 ns	-

Bauteil	t_p	t_s
NOR	5 ns	-
XOR	9 ns	-
D-Flipflop	11 ns	3 ns

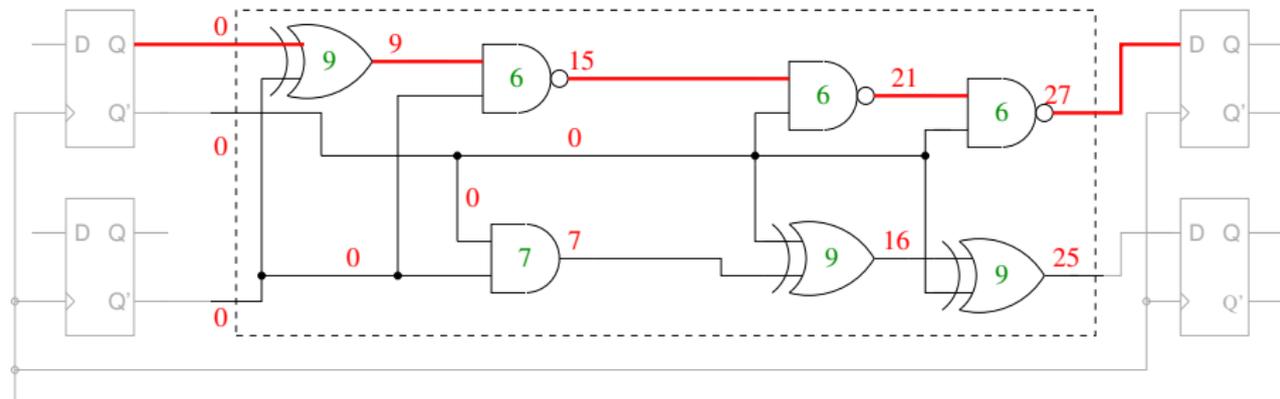
Beispiel Frequenzberechnung



Bauteil	t_p	t_s
AND	7 ns	-
NAND	6 ns	-
OR	6 ns	-

Bauteil	t_p	t_s
NOR	5 ns	-
XOR	9 ns	-
D-Flipflop	11 ns	3 ns

Beispiel Frequenzberechnung



Bauteil	t_p	t_s
AND	7 ns	-
NAND	6 ns	-
OR	6 ns	-

Bauteil	t_p	t_s
NOR	5 ns	-
XOR	9 ns	-
D-Flipflop	11 ns	3 ns

	3 ns	Setup+Holdzeiten
+	11 ns	Propagation Delay der Flipflops
+	27 ns	längster Pfad
<hr/>		
=	41 ns	Zykluszeit

Max. Betriebsfrequenz:

$$\frac{1}{41\text{ns}} \approx 24390244 \text{ Hz} \approx 24.4 \text{ MHz}$$

Q: Wie können wir **die Taktfrequenz erhöhen**?
(Erfreut die Marketing-Abteilung)

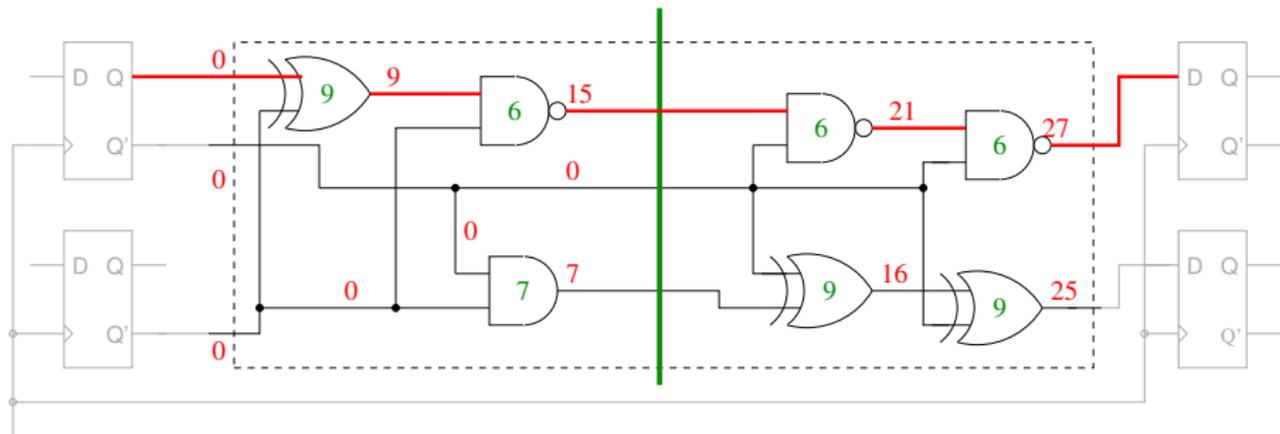
Erinnerung: Die Taktfrequenz wird durch den längsten Pfad zwischen zwei D-Flipflops bestimmt.

Q: Wie können wir **die Taktfrequenz erhöhen**?
(Erfreut die Marketing-Abteilung)

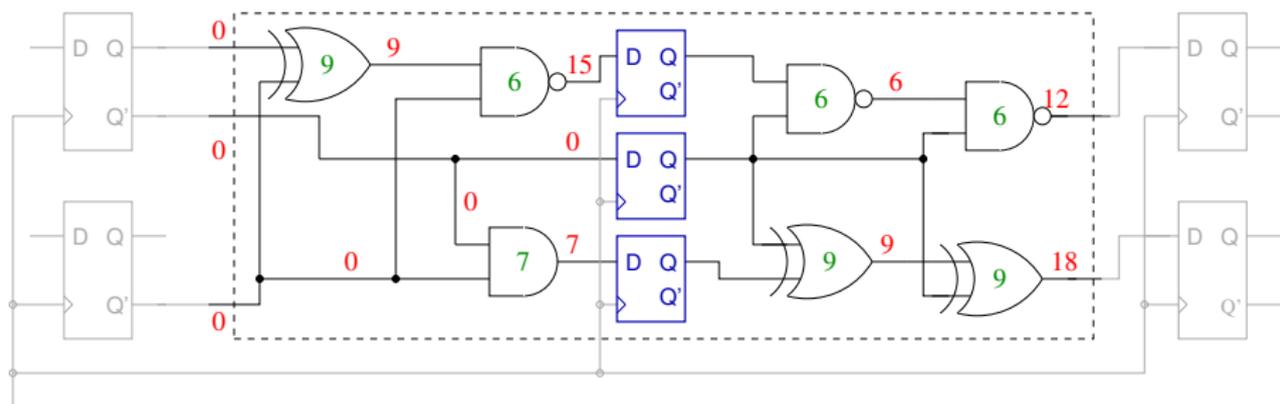
Erinnerung: Die Taktfrequenz wird durch den längsten Pfad zwischen zwei D-Flipflops bestimmt.

Idee: **Einfügen eines D-Flipflops in diesen Pfad!**

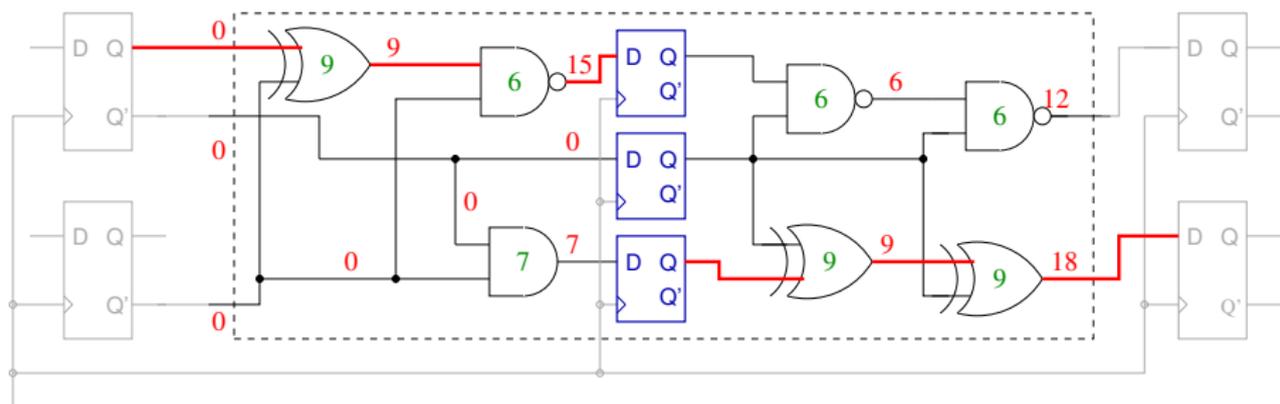
Beispiel Pipelining



Beispiel Pipelining



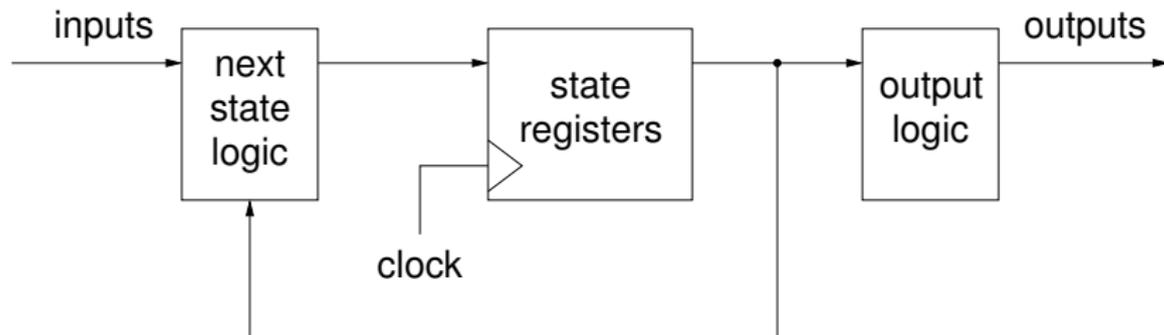
Beispiel Pipelining



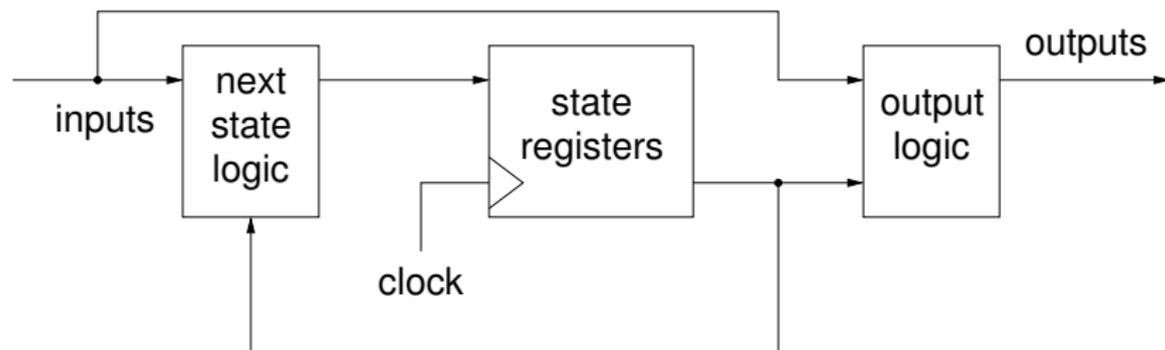
	3 ns	Setup+Holdzeiten
+	11 ns	Propagation Delay der Flipflops
+	18 ns	längster Pfad
<hr/>		
=	32 ns	Zykluszeit

Max. Betriebsfrequenz:

$$\frac{1}{32\text{ns}} \approx 31250000 \text{ Hz} \approx 31.3 \text{ MHz}$$



- ▶ Allgemeiner Prototyp eines synchronen sequentiellen Entwurfes
- ▶ Ausgaben hängen nur vom momentanen Zustand ab und ändern sich mit der Clock-Flanke
- ▶ kombinatorische Logik bildet den aktuellen Zustand in den nächsten ab
- ▶ (Eselsbrücke: nicht abhängig von den Eingaben, also kein **E**)



- ▶ Entwurf mit asynchronem Output
- ▶ Die Ausgaben hängen vom momentanen Zustand und den aktuellen Eingaben ab
- ▶ Die Ausgabe kann sich unabhängig von der Clock mit dem Wechseln des Inputs ändern
- ▶ (Eselbrücke: das **e** in Mealy steht für **E**ingabe)

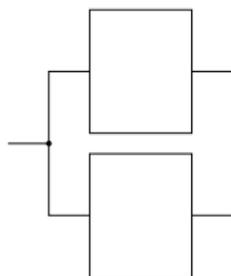
Menge von Zuständen S ,
 Menge von Eingaben E ,
 Menge von Ausgaben O

Moore Maschine: $(S, I, E, O, \delta, \lambda)$ mit $\delta: S \times E \rightarrow S$ und $\lambda: S \rightarrow O$

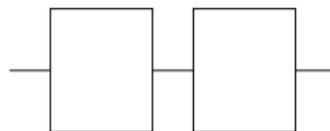
Mealy Maschine: $(S, I, E, O, \delta, \lambda)$ mit $\delta: S \times E \rightarrow S$ und $\lambda: S \times E \rightarrow O$

Boolesche Kodierung: $S = 2^n, E = 2^m, O = 2^k$

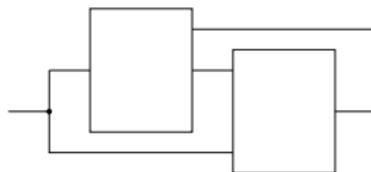
$\delta = (\delta_i)$	mit	$\delta_i: 2^{n+m} \rightarrow \{0, 1\}$	für $i = 0 \dots n$	
$\lambda = (\lambda_j)$	mit	$\lambda_j: 2^n \rightarrow \{0, 1\}$	für $j = 0 \dots k$	(Moore)
$\lambda = (\lambda_j)$	mit	$\lambda_j: 2^{n+m} \rightarrow \{0, 1\}$	für $j = 0 \dots k$	(Mealy)



parallel

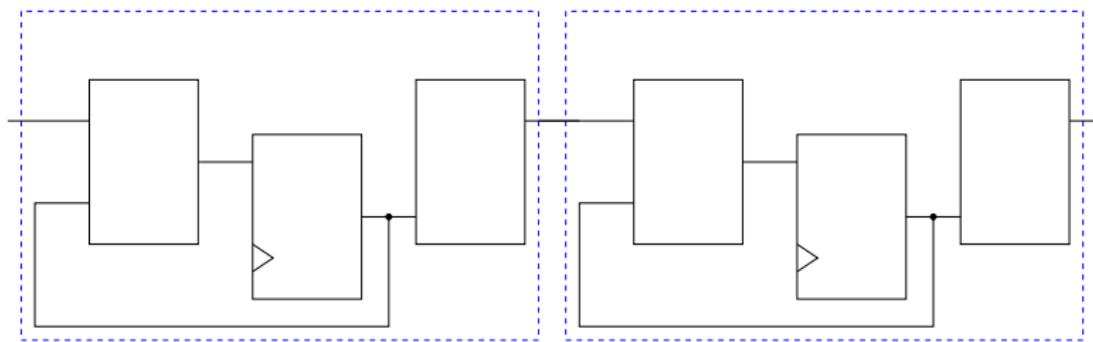


sequentiell

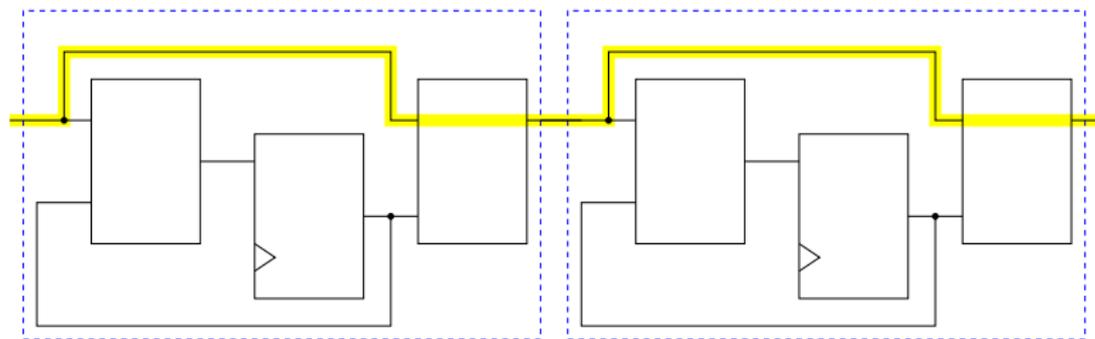


hybrid

(Datenfluss von links nach rechts)

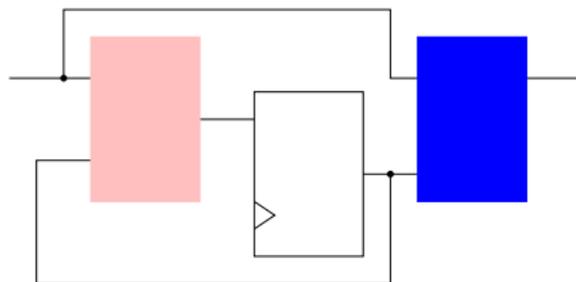


Sequentielle Komposition von Moore-Maschinen erzeugt Verzögerung
(Das berechnete Ergebnis wird erst im nächsten Takt weitergegeben)

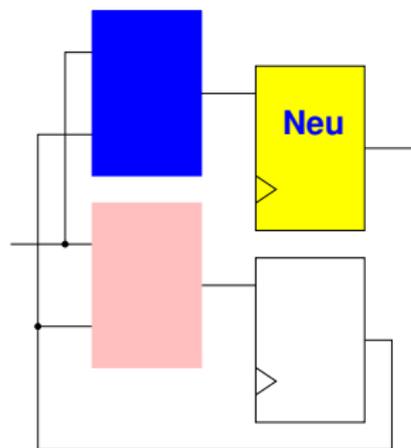


Sequentielle Komposition von Mealy-Maschinen erzeugt lange Signalpfade!

(Erinnerung: längste Pfade
antiproportional zu erzielbaren Taktfrequenz)

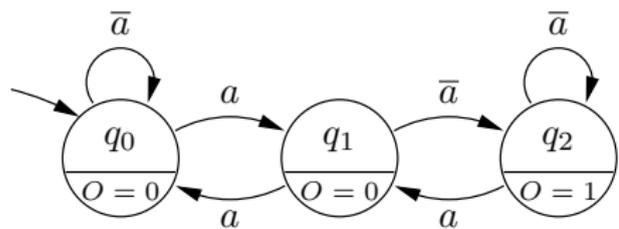


Mealy-Maschine

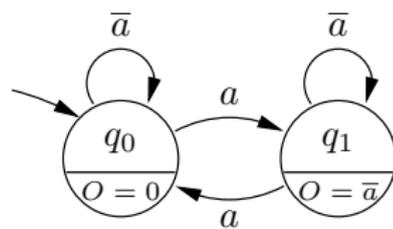


Äquivalente Moore-Maschine

Moore-Maschine ist um einen Takt zeitverzögert

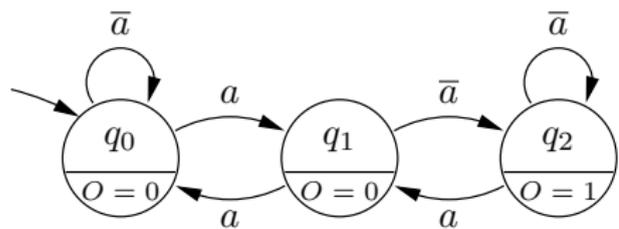


(a)

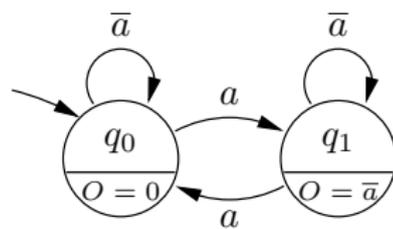


(b)

Unterer Teil: spezifiziert die Ausgabefunktion λ



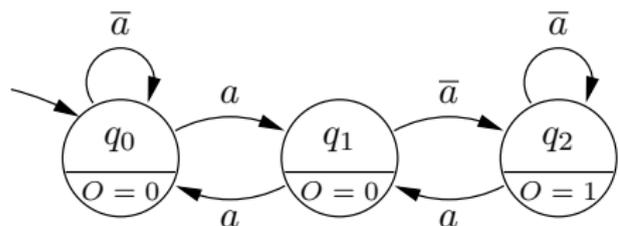
(a)



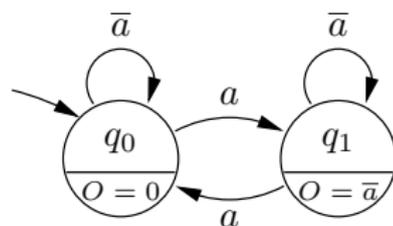
(b)

Unterer Teil: spezifiziert die Ausgabefunktion λ

1) Mealy? Moore?



(a)



(b)

Unterer Teil: spezifiziert die Ausgabefunktion λ

- 1) Mealy? Moore?
- 2) Ist die Ausgabe beider Maschinen bei gleicher Eingabesequenz immer gleich?

(Buch, Aufgabe 3.2, Seite 103)

1. Verilog Modul erstellen
2. Zustandsbits definieren
3. Next-State Funktion definieren
4. Output-Funktion definieren

```
module statemachine(input clk, rst, ... // Inputs  
                   output reg out, ...);
```

```
...
```

5

```
endmodule
```

- ▶ Annahme: Eine Clock für alles
- ▶ Inputs und Outputs hängen von der Anwendung ab

```
localparam [2:0]  
    AUS = 'b000,  
    ROT = 'b001,  
    GELB = 'b010,  
5    GRUEN = 'b011;
```

```
reg [2:0] state;
```

- ▶ `localparam` definiert Modul-lokale Konstanten
- ▶ `reg` bewirkt Generierung von Flipflops
- ▶ `state` ggf. in mehrere Variablen aufteilen

```
always @(posedge clk) begin  
    if ( rst )  
        state=INIT; // Synchroner Reset  
    else case(state)  
5      INIT:  
        if (input1)  
            state=Q2;  
        else  
            state=Q1;  
10  
        Q1: ...  
        Q2: ...  
15      default: next_state=INIT;  
    endcase  
end
```

```
// Mealy haengt auch von Inputs ab!
```

```
always @(state or rst or ...) begin
```

```
    out1 = state[1] | input1;
```

```
    out2 = state[1] & state[2];
```

```
5 end
```

Darf auch ein weiteres `case (state)` zur Fallunterscheidung enthalten