

# 64-040 Modul InfB-RS: Rechnerstrukturen

[https://tams.informatik.uni-hamburg.de/  
lectures/2015ws/vorlesung/rs](https://tams.informatik.uni-hamburg.de/lectures/2015ws/vorlesung/rs)

– Kapitel 10 –

Norman Hendrich



Universität Hamburg  
Fakultät für Mathematik, Informatik und Naturwissenschaften  
Fachbereich Informatik

**Technische Aspekte Multimodaler Systeme**

Wintersemester 2015/2016



# Kapitel 10

## Schaltwerke

Definition und Modelle

Asynchrone (ungetaktete) Schaltungen

Synchrone (getaktete) Schaltungen

Flipflops

RS-Flipflop

D-Latch

D-Flipflop

JK-Flipflop

Hades

Zeitbedingungen

Taktschemata

Beschreibung von Schaltwerken



# Kapitel 10 (cont.)

Entwurf von Schaltwerken

Beispiele

- Ampelsteuerung

- Zählschaltungen

- verschiedene Beispiele

Asynchrone Schaltungen

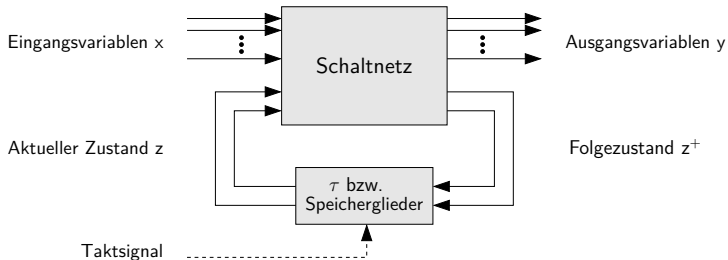
Literatur



# Schaltwerke

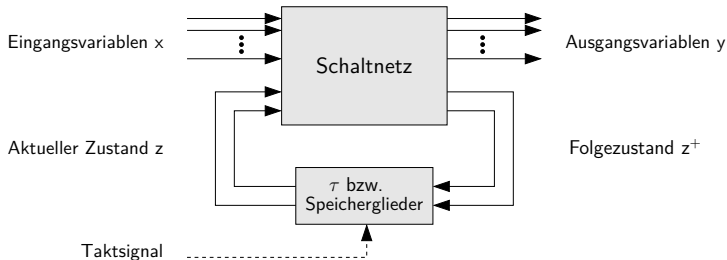
- ▶ **Schaltwerk:** Schaltung mit Rückkopplungen und Verzögerungen
- ▶ fundamental andere Eigenschaften als Schaltnetze
- ▶ Ausgangswerte nicht nur von Eingangswerten abhängig sondern auch von der Vorgeschichte
- ⇒ interner Zustand repräsentiert „Vorgeschichte“
- ▶ ggf. stabile Zustände ⇒ Speicherung von Information
- ▶ bei unvorsichtigem Entwurf: chaotisches Verhalten

# Schaltwerke: Blockschaltbild



- ▶ Eingangsvariablen  $x$  und Ausgangsvariablen  $y$
- ▶ Aktueller Zustand  $z$
- ▶ Folgezustand  $z^+$
- ▶ Rückkopplung läuft über Verzögerungen  $\tau$  / Speicherglieder

## Schaltwerke: Blockschaltbild (cont.)



zwei prinzipielle Varianten für die Zeitglieder

1. nur (Gatter-) Verzögerungen: **asynchrone** oder **nicht getaktete Schaltwerke**
2. getaktete Zeitglieder: **synchrone** oder **getaktete Schaltwerke**



## Synchrone und Asynchrone Schaltwerke

- ▶ **synchrone Schaltwerke:** die Zeitpunkte, an denen das Schaltwerk von einem stabilen Zustand in einen stabilen Folgezustand übergeht, werden explizit durch ein Taktsignal (*clock*) vorgegeben
- ▶ **asynchrone Schaltwerke:** hier fehlt ein Taktgeber, Änderungen der Eingangssignale wirken sich unmittelbar aus (entsprechend der Gatterverzögerungen  $\tau$ )
- ▶ potentiell höhere Arbeitsgeschwindigkeit
- ▶ aber sehr aufwändiger Entwurf
- ▶ fehleranfälliger (z.B. leicht veränderte Gatterverzögerungen durch Bauteil-Toleranzen, Spannungsschwankungen, usw.)

# Theorie: Endliche Automaten

## FSM – Finite State Machine

- ▶ Deterministischer Endlicher Automat mit Ausgabe
- ▶ 2 äquivalente Modelle
  - ▶ Mealy: Ausgabe hängt *von Zustand und Eingabe* ab
  - ▶ Moore: –"– *nur vom Zustand* ab
- ▶ 6-Tupel  $(Z, \Sigma, \Delta, \delta, \lambda, z_0)$ 
  - ▶  $Z$  Menge von Zuständen
  - ▶  $\Sigma$  Eingabealphabet
  - ▶  $\Delta$  Ausgabealphabet
  - ▶  $\delta$  Übergangsfunktion  $\delta : Z \times \Sigma \rightarrow Z$
  - ▶  $\lambda$  Ausgabefunktion  $\lambda : Z \times \Sigma \rightarrow \Delta$       Mealy-Modell
  - $\lambda : Z \rightarrow \Delta$                       Moore- –"–
  - ▶  $z_0$  Startzustand





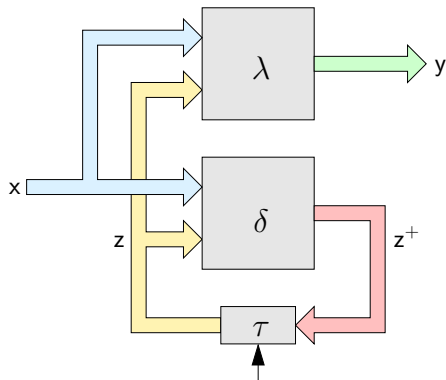
## Mealy-Modell und Moore-Modell

- ▶ **Mealy-Modell:** die Ausgabe hängt vom Zustand  $z$  und vom momentanen Input  $x$  ab
- ▶ **Moore-Modell:** die Ausgabe des Schaltwerks hängt nur vom aktuellen Zustand  $z$  ab
  
- ▶ **Ausgabefunktion:**

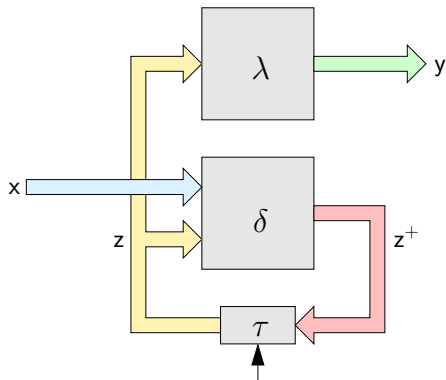
$y = \lambda(z, x)$	Mealy
$y = \lambda(z)$	Moore
- ▶ **Überföhrungsfunktion:**  $z^+ = \delta(z, x)$       Moore und Mealy
  
- ▶ **Speicherglieder** oder Verzögerung  $\tau$  im Rückkopplungspfad

# Mealy-Modell und Moore-Modell (cont.)

## ▶ Mealy-Automat



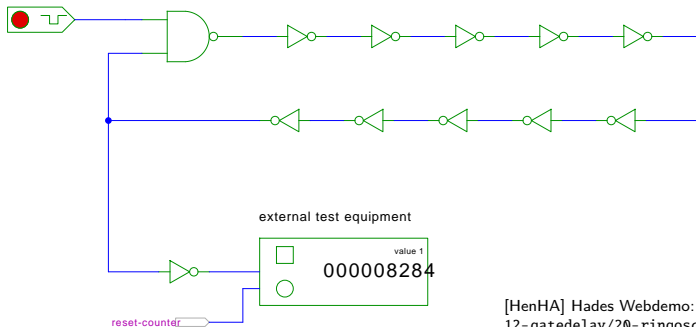
## Moore-Automat



# Asynchrone Schaltungen: Beispiel Ringoszillator

click to start/stop

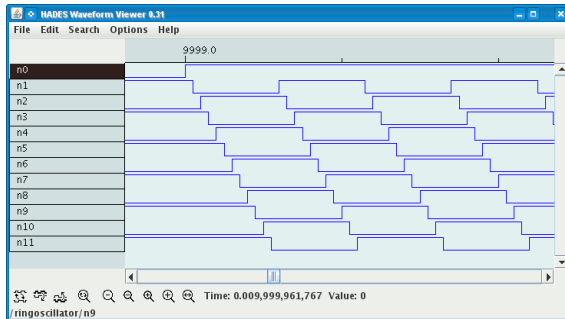
odd number of inverting gates



[HenHA] Hades Webdemo:  
 12-gatedelay/20-ringoscillator/ringoscillator

- ▶ stabiler Zustand, solange der Eingang auf 0 liegt
- ▶ instabil sobald der Eingang auf 1 wechselt (Oszillation)

## Asynchrone Schaltungen: Beispiel Ringoszillator (cont.)



- ▶ Rückkopplung: ungerade Anzahl  $n$  invertierender Gatter ( $n \geq 3$ )
- ▶ Start/Stop über steuerndes NAND-Gatter
- ▶ Oszillation mit maximaler Schaltfrequenz  
z.B.: als Testschaltung für neue (Halbleiter-) Technologien



## Asynchrone Schaltungen: Probleme

- ▶ das Schaltwerk kann stabile und nicht-stabile Zustände enthalten
  - ▶ die Verzögerungen der Bauelemente sind nicht genau bekannt und können sich im Betrieb ändern
  - ▶ Variation durch Umweltparameter  
z.B. Temperatur, Versorgungsspannung, Alterung
- ⇒ sehr schwierig, die korrekte Funktion zu garantieren  
z.B. mehrstufige Handshake-Protokolle
- ▶ in der Praxis überwiegen synchrone Schaltwerke
  - ▶ Realisierung mit **Flipflops** als Zeitgliedern



## Synchrone Schaltungen

- ▶ alle Rückkopplungen der Schaltung laufen über spezielle Zeitglieder: „Flipflops“
  - ▶ diese definieren / speichern einen stabilen Zustand, unabhängig von den Eingabewerten und Vorgängen im  $\delta$ -Schaltnetz
  - ▶ Hinzufügen eines zusätzlichen Eingangssignals: „Takt“
  - ▶ die Zeitglieder werden über das Taktsignal gesteuert  
 verschiedene Möglichkeiten: Pegel- und Flankensteuerung, Mehrphasentakte (s.u.)
- ⇒ synchrone Schaltwerke sind wesentlich einfacher zu entwerfen und zu analysieren als asynchrone Schaltungen

## Zeitglieder / Flipflops

- ▶ **Zeitglieder:** Bezeichnung für die Bauelemente, die den Zustand des Schaltwerks speichern können
- ▶ **bistabile Bauelemente** (Kippglieder) oder **Flipflops**
- ▶ zwei stabile Zustände  $\Rightarrow$  speichert 1 Bit
  - 1 – Setzzustand
  - 0 – Rücksetzzustand
- ▶ Übergang zwischen Zuständen durch geeignete Ansteuerung



# Flipflops

- ▶ Name für die **elementaren** Schaltwerke
- ▶ mit genau zwei Zuständen  $Z_0$  und  $Z_1$
- ▶ Zustandsdiagramm hat zwei Knoten und vier Übergänge (s.u.)
  
- ▶ Ausgang als  $Q$  bezeichnet und dem Zustand gleichgesetzt
- ▶ meistens auch invertierter Ausgang  $\bar{Q}$  verfügbar
  
- ▶ Flipflops sind selbst nicht getaktet
- ▶ sondern „sauber entworfene“ asynchrone Schaltwerke
- ▶ Anwendung als Verzögerungs-/Speicherelemente in getakteten Schaltwerken

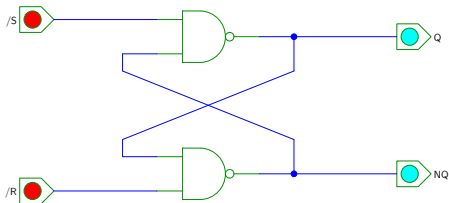




## Flipflops: Typen

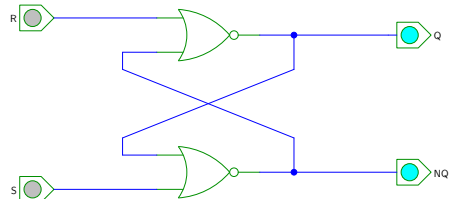
- ▶ Basis-Flipflop „Reset-Set-Flipflop“
- ▶ getaktetes RS-Flipflop
  
- ▶ pegelgesteuertes D-Flipflop „D-Latch“
- ▶ flankengesteuertes D-Flipflop „D-Flipflop“
  
- ▶ JK-Flipflop
- ▶ weitere. . .

# RS-Flipflop: NAND- und NOR-Realisierung



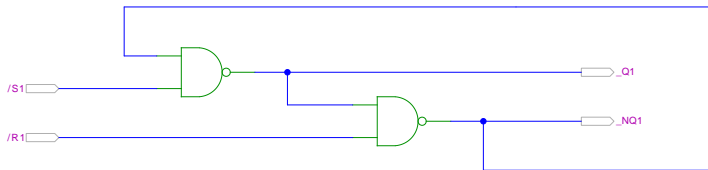
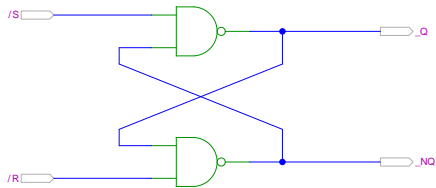
$/S$	$/R$	Q	NQ	NAND
0	0	1	1	forbidden
0	1	1	0	
1	0	0	1	
1	1	$Q^*$	$NQ^*$	store

[HenHA] Hades Demo: 16- flipflops/10- srff/srff



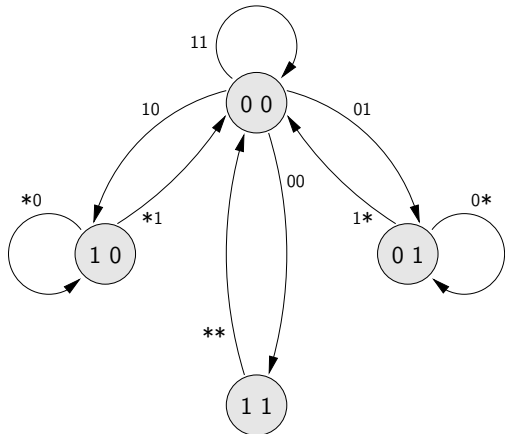
S	R	Q	NQ	NOR
0	0	$Q^*$	$NQ^*$	store
0	1	0	1	
1	0	1	0	
1	1	0	0	forbidden

# RS-Flipflop: Varianten des Schaltbilds



[HenHA] Hades Demo: 16- flipflops/10- srff/srff2

# NOR RS-Flipflop: Zustandsdiagramm und Flusstafel

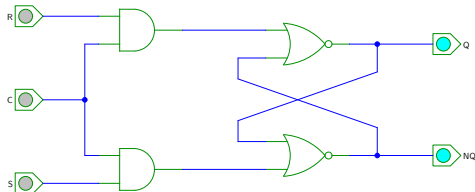


Zustand	Eingabe [S R]			
	00	01	11	10
00	11	01	00	10
01	01	01	00	00
11	00	00	00	00
10	10	00	00	10

stabiler Zustand

## RS-Flipflop mit Takt

- ▶ RS-Basisflipflop mit zusätzlichem Takteingang  $C$
- ▶ Änderungen nur wirksam, während  $C$  aktiv ist
- ▶ Struktur



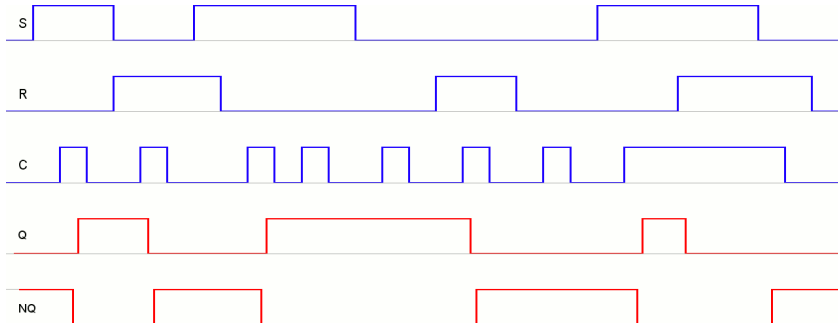
C	S	R	Q	NQ	NOR
0	X	X	Q*	NQ*	store
1	0	0	Q*	NQ*	store
1	0	1	0	1	
1	1	0	1	0	
1	1	1	0	0	forbidden

[HenHA] Hades Demo: 16-flipflops/10-srff/clocked-srff

## RS-Flipflop mit Takt (cont.)

$$\begin{aligned} \blacktriangleright Q &= \overline{(NQ \vee (R \wedge C))} \\ NQ &= \overline{(Q \vee (S \wedge C))} \end{aligned}$$

### ▶ Impulsdiagramm





## Pegelgesteuertes D-Flipflop (D-Latch)

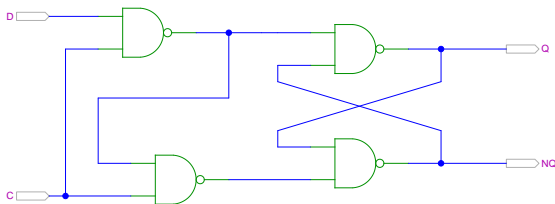
- ▶ Takteingang  $C$
- ▶ Dateneingang  $D$
- ▶ aktueller Zustand  $Q$ , Folgezustand  $Q^+$

$C$	$D$	$Q^+$
0	0	$Q$
0	1	$Q$
1	0	0
1	1	1

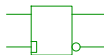
- ▶ Wert am Dateneingang wird durchgeleitet, wenn das Taktsignal 1 ist  $\Rightarrow$  *high*-aktiv  
 0 ist  $\Rightarrow$  *low*-aktiv

## Pegelgesteuertes D-Flipflop (D-Latch) (cont.)

- ▶ Realisierung mit getaktetem RS-Flipflop und einem Inverter  
 $S = D, R = \bar{D}$
- ▶ minimierte NAND-Struktur

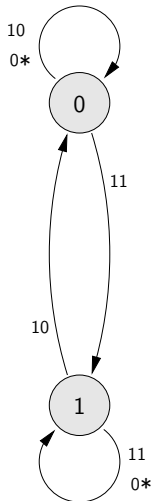


- ▶ Symbol





# D-Latch: Zustandsdiagramm und Flusstafel



Zustand [Q]	Eingabe [C D]			
	00	01	11	10
Folgezustand [Q <sup>+</sup> ]	0	0	1	0
	1	1	1	0

stabiler Zustand

## Flankengesteuertes D-Flipflop

- ▶ Takteingang  $C$
- ▶ Dateneingang  $D$
- ▶ aktueller Zustand  $Q$ , Folgezustand  $Q^+$

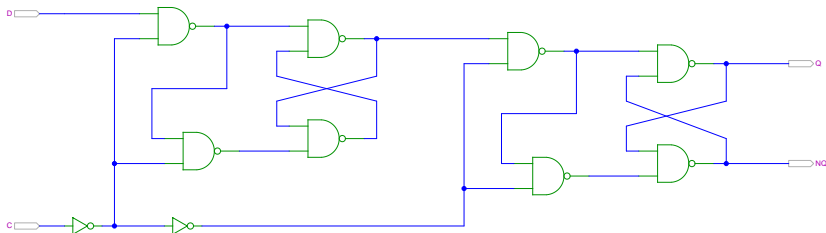
$C$	$D$	$Q^+$
0	*	$Q$
1	*	$Q$
↑	0	0
↑	1	1

- ▶ Wert am Dateneingang wird gespeichert, wenn das Taktsignal sich von 0 auf 1 ändert  $\Rightarrow$  Vorderflankensteuerung  
 –"– 1 auf 0 ändert  $\Rightarrow$  Rückflankensteuerung
- ▶ Realisierung als Master-Slave Flipflop oder direkt

## Master-Slave D-Flipflop

- ▶ zwei kaskadierte D-Latches
  - ▶ hinteres Latch erhält invertierten Takt
  - ▶ vorderes „Master“-Latch: low-aktiv (transparent bei  $C = 0$ )  
 hinteres „Slave“-Latch: high-aktiv (transparent bei  $C = 1$ )
  - ▶ vorderes Latch speichert bei Wechsel auf  $C = 1$
  - ▶ wenig später (Gatterverzögerung im Inverter der Taktleitung)  
 übernimmt das hintere „Slave“-Latch diesen Wert
  - ▶ anschließend Input für das Slave-Latch stabil
  - ▶ Slave-Latch speichert, sobald Takt auf  $C = 0$  wechselt
- ⇒ dies entspricht effektiv einer **Flankensteuerung**:  
 Wert an  $D$  nur relevant, kurz bevor Takt auf  $C = 1$  wechselt

## Master-Slave D-Flipflop (cont.)



[HenHA] Hades Demo: 16-flipflops/20-dlatch/dff

► zwei kaskadierte pegel-gesteuerte D-Latches

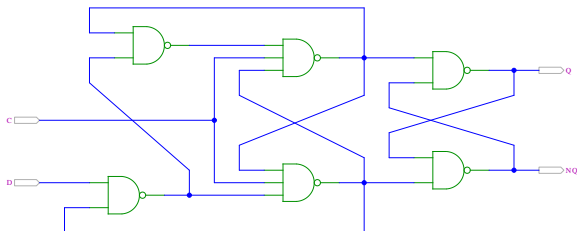
**C=0** Master aktiv (transparent)

Slave hat (vorherigen) Wert gespeichert

**C=1** Master speichert Wert

Slave transparent, leitet Wert von Master weiter

## Vorderflanken-gesteuertes D-Flipflop



- ▶ Dateneingang  $D$  wird nur durch Takt-Vorderflanke ausgewertet
- ▶ Gatterlaufzeiten für Funktion essentiell
- ▶ Einhalten der Vorlauf- und Haltezeiten vor/nach der Taktflanke (s.u. *Zeitbedingungen*)

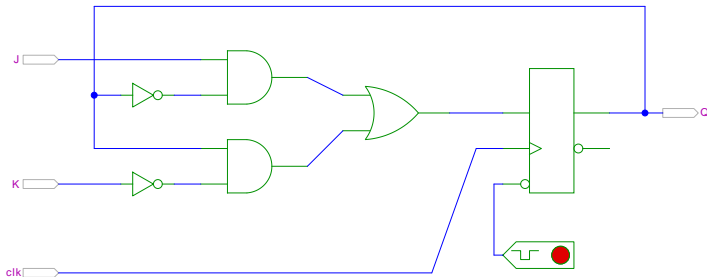
## JK-Flipflop

- ▶ Takteingang  $C$
- ▶ Steuereingänge  $J$  („jump“) und  $K$  („kill“)
- ▶ aktueller Zustand  $Q$ , Folgezustand  $Q^+$

$C$	$J$	$K$	$Q^+$	Funktion
*	*	*	$Q$	Wert gespeichert
↑	0	0	$Q$	Wert gespeichert
↑	0	1	0	Rücksetzen
↑	1	0	1	Setzen
↑	1	1	$\overline{Q}$	Invertieren

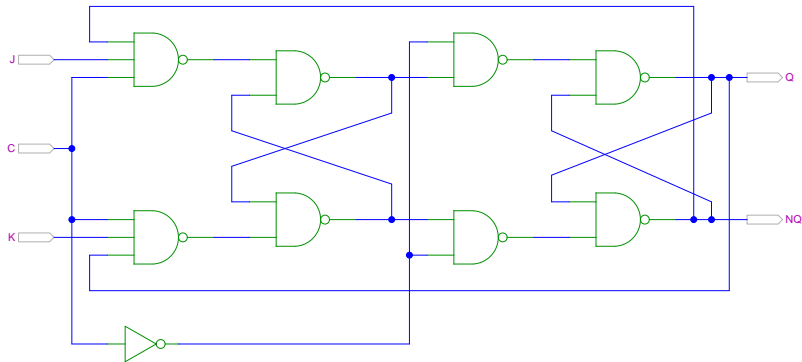
- ▶ universelles Flipflop, sehr flexibel einsetzbar
- ▶ in integrierten Schaltungen nur noch selten verwendet

# JK-Flipflop: Realisierung mit D-Flipflop



[HenHA] Hades Demo: 16-flipflops/40- jkff/jkff-prinzip

## JK-Flipflop: Realisierung als Master-Slave Schaltung

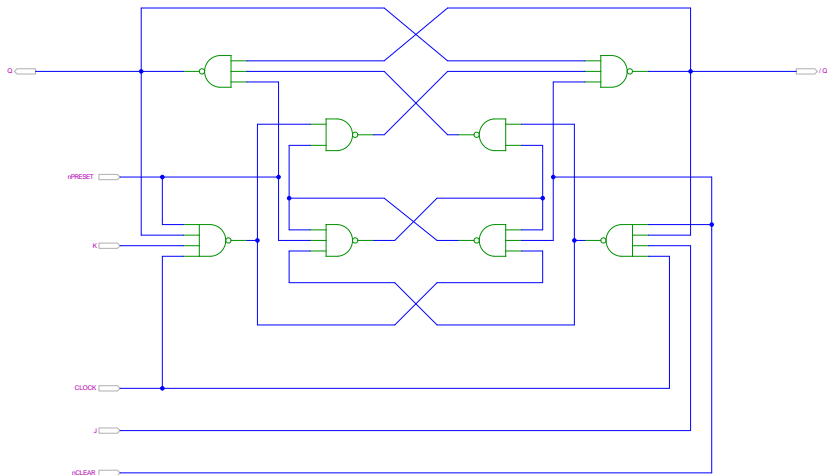


[HenHA] Hades Demo: 16-flipflops/40-jkff/jkff

Achtung: Schaltung wegen Rückkopplungen schwer zu initialisieren



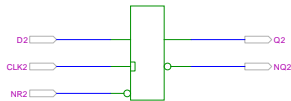
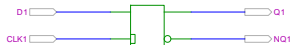
# JK-Flipflop: tatsächliche Schaltung im IC 7476



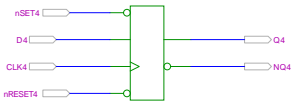
[HenHA] Hades Demo: 16- flipflops/40- jkff/SN7476- single

# Flipflop-Typen: Komponenten/Symbole in Hades

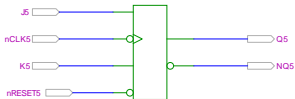
D-type latches



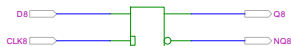
D-type flipflops



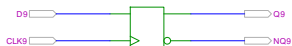
JK flipflop



metastable D-Latch (don't use!)

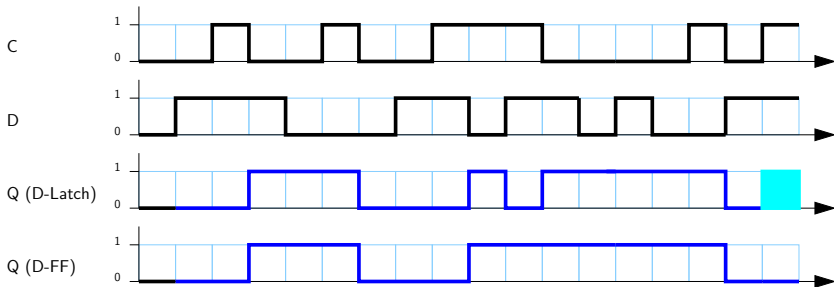


metastable D-flipflop (don't use!)



[HenHA] Hades Webdemo:  
 16-flipflops/50-ffdemo/  
 flipflopdemo

## Flipflop-Typen: Impulsdiagramme



- ▶ pegel- und vorderflankengesteuertes Flipflop
- ▶ beide Flipflops hier mit jeweils einer Zeiteinheit Verzögerung
- ▶ am Ende undefinierte Werte wegen gleichzeitigem Wechsel von  $C$  und  $D$  (Verletzung der Zeitbedingungen)



## Flipflops: Zeitbedingungen

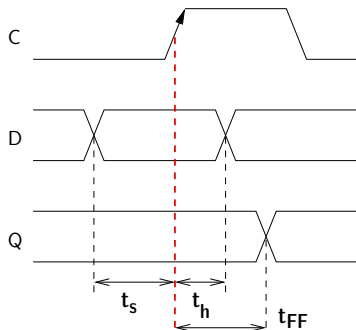
- ▶ Flipflops werden entwickelt, um Schaltwerke einfacher entwerfen und betreiben zu können
  - ▶ Umschalten des Zustandes durch das Taktsignal gesteuert
  - ▶ aber: jedes Flipflop selbst ist ein asynchrones Schaltwerk mit kompliziertem internem Zeitverhalten
  - ▶ Funktion kann nur garantiert werden, wenn (typ-spezifische) Zeitbedingungen eingehalten werden
- ⇒ „Vorlauf- und Haltezeiten“ (*setup- / hold-time*)
- ⇒ Daten- und Takteingänge dürfen sich nie gleichzeitig ändern

## Flipflops: Vorlauf- und Haltezeit

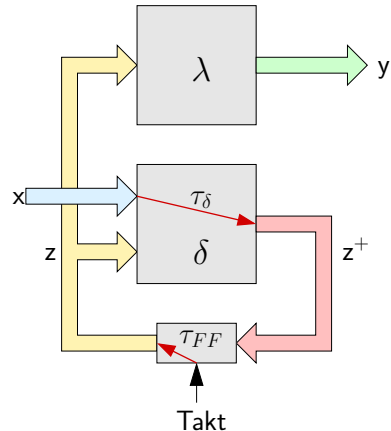
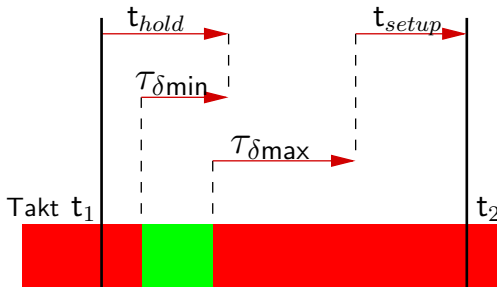
- ▶  $t_s$  Vorlaufzeit (engl. *setup-time*): Zeitintervall, innerhalb dessen das Datensignal *vor* dem nächsten Takt stabil anliegen muss
- ▶  $t_h$  Haltezeit (engl. *hold-time*): Zeitintervall, innerhalb dessen das Datensignal *nach* einem Takt noch stabil anliegen muss

⇒ Verletzung der Zeitbedingungen  
 „falscher“ Wert an Q

- ▶  $t_{FF}$  Ausgangsverzögerung



# Zeitbedingungen: Eingangsvektor

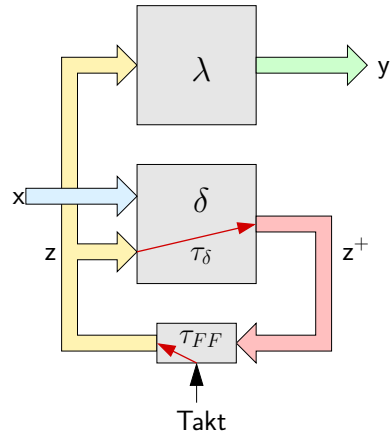
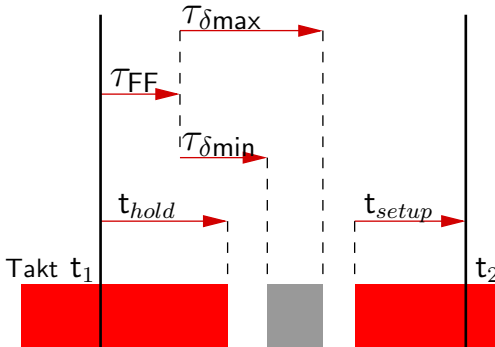




## Zeitbedingungen: Eingangsvektor (cont.)

- ▶ Änderungen der Eingangswerte  $x$  werden beim Durchlaufen von  $\delta$  mindestens um  $\tau_{\delta_{\min}}$ , bzw. maximal um  $\tau_{\delta_{\max}}$  verzögert
  - ▶ um die Haltezeit der Zeitglieder einzuhalten, darf  $x$  sich nach einem Taktimpuls frühestens zum Zeitpunkt  $(t_1 + t_{hold} - \tau_{\delta_{\min}})$  wieder ändern
  - ▶ um die Vorlaufzeit vor dem nächsten Takt einzuhalten, muss  $x$  spätestens zum Zeitpunkt  $(t_2 - t_{setup} - \tau_{\delta_{\max}})$  wieder stabil sein
- ⇒ Änderungen dürfen nur im grün markierten Zeitintervall erfolgen

# Zeitbedingungen: interner Zustand







## Zeitbedingungen: interner Zustand (cont.)

- ▶ zum Zeitpunkt  $t_1$  wird ein Taktimpuls ausgelöst
  - ▶ nach dem Taktimpuls vergeht die Zeit  $\tau_{FF}$ , bis die Zeitglieder (Flipflops) ihren aktuellen Eingangswert  $z^+$  übernommen haben und als neuen Zustand  $z$  am Ausgang bereitstellen
  - ▶ die neuen Werte von  $z$  laufen durch das  $\delta$ -Schaltnetz, der schnellste Pfad ist dabei  $\tau_{\delta_{\min}}$  und der langsamste ist  $\tau_{\delta_{\max}}$
- $\Rightarrow$  innerhalb der Zeitintervalls  $\tau_{FF} + \tau_{\delta_{\min}}$  bis  $\tau_{FF} + \tau_{\delta_{\max}}$  ändern sich die Werte des Folgezustands  $z^+$ 
grauer Bereich



## Zeitbedingungen: interner Zustand (cont.)

- ▶ die Änderungen dürfen frühestens zum Zeitpunkt ( $t_1 + t_{hold}$ ) beginnen, ansonsten würde Haltezeit verletzt  
 ggf. muss  $\tau_{\delta_{min}}$  vergrößert werden, um diese Bedingung einhalten zu können (zusätzliche Gatterverzögerungen)
- ▶ die Änderungen müssen sich spätestens bis zum Zeitpunkt ( $t_2 - t_{setup}$ ) stabilisiert haben (der Vorlaufzeit der Flipflops vor dem nächsten Takt)



## Maximale Taktfrequenz einer Schaltung

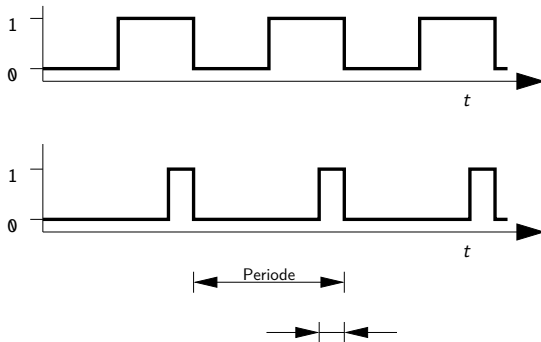
- ▶ aus obigen Bedingungen ergibt sich sofort die maximal zulässige Taktfrequenz einer Schaltung
- ▶ Umformen und Auflösen nach dem Zeitpunkt des nächsten Takts ergibt zwei Bedingungen

$$\Delta t \geq (\tau_{FF} + \tau_{\delta_{\max}} + \tau_{setup}) \quad \text{und}$$

$$\Delta t \geq (\tau_{hold} + \tau_{setup})$$

- ▶ falls diese Bedingung verletzt wird („Übertakten“), kann es (datenabhängig) zu Fehlfunktionen kommen

## Taktsignal: Prinzip



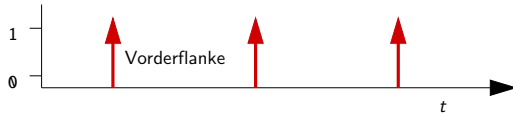
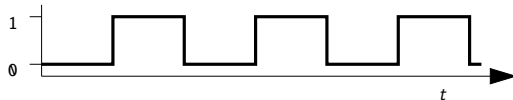
- ▶ periodisches digitales Signal, Frequenz  $f$  bzw. Periode  $\tau$
- ▶ oft symmetrisch
- ▶ asymmetrisch für Zweiphasentakt (s.u.)



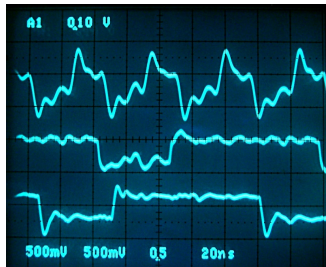
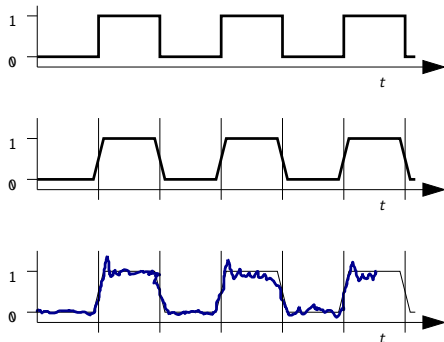
## Taktsignal: Varianten

- ▶ **Pegelsteuerung:** Schaltung reagiert, während das Taktsignal den Wert 1 (bzw. 0) aufweist
  
- ▶ **Flankensteuerung:** Schaltung reagiert nur, während das Taktsignal seinen Wert wechselt
  - ▶ Vorderflankensteuerung: Wechsel von 0 nach 1
  - ▶ Rückflankensteuerung: —"– von 1 nach 0
  
- ▶ Zwei- und Mehrphasentakte

## Taktsignal: Varianten (cont.)



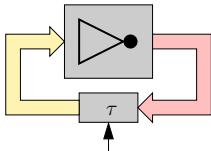
## Taktsignal: Prinzip und Realität



- ▶ Werteverläufe in realen Schaltungen stark gestört
- ▶ Überschwingen/Übersprechen benachbarter Signale
- ▶ Flankensteilheit nicht garantiert (bei starker Belastung)  
ggf. besondere Gatter („Schmitt-Trigger“)

## Problem mit Pegelsteuerung

- ▶ während des aktiven Taktpegels werden Eingangswerte direkt übernommen
- ▶ falls invertierende Rückkopplungspfade in  $\delta$  vorliegen, kommt es dann zu instabilen Zuständen (Oszillationen)



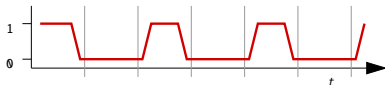
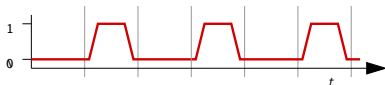
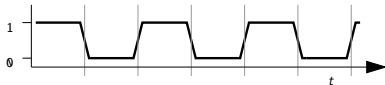
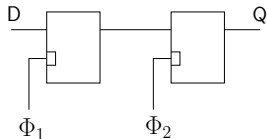
- ▶ einzelne pegelgesteuerte Zeitglieder (D-Latches) garantieren keine stabilen Zustände
- ⇒ Verwendung von je zwei pegelgesteuerten Zeitgliedern und Einsatz von Zweiphasentakt oder
- ⇒ Verwendung flankengesteuerter D-Flipflops



## Zweiphasentakt

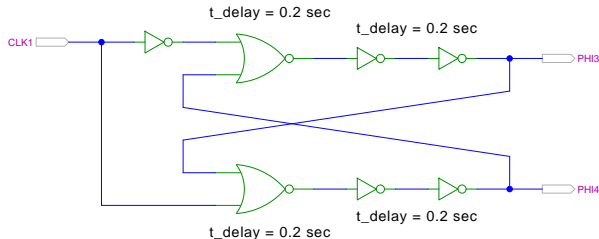
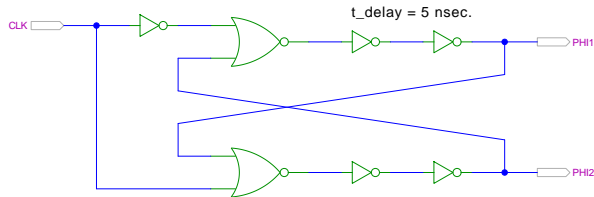
- ▶ pegelgesteuertes D-Latch ist bei aktivem Takt *transparent*
- ▶ rück-gekoppelte Werte werden sofort wieder durchgelassen
- ▶ Oszillation bei invertierten Rückkopplungen
  
- ▶ Reihenschaltung aus jeweils zwei D-Latches
- ▶ zwei separate Takte  $\Phi_1$  und  $\Phi_2$ 
  - ▶ bei Takt  $\Phi_1$  übernimmt vorderes Flipflop den Wert
  - erst bei Takt  $\Phi_2$  übernimmt hinteres Flipflop
  - ▶ vergleichbar Master-Slave Prinzip bei D-FF aus Latches

## Zweiphasentakt (cont.)


 $\Phi_1$ 
 $\Phi_2$ 


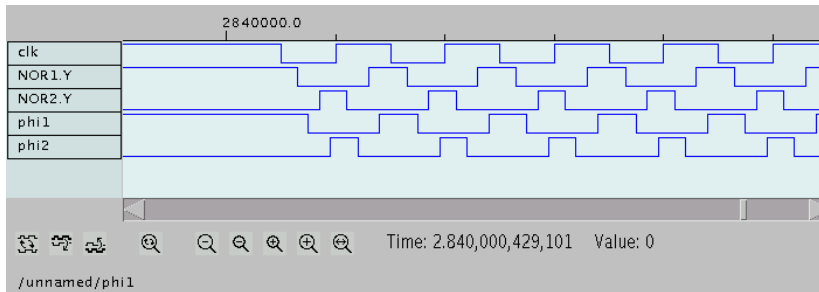
- ▶ nicht überlappender Takt mit Phasen  $\Phi_1$  und  $\Phi_2$
- ▶ vorderes D-Latch übernimmt Eingangswert  $D$  während  $\Phi_1$   
 bei  $\Phi_2$  übernimmt das hintere D-Latch und liefert  $Q$

# Zweiphasentakt: Erzeugung



[HenHA] Hades Webdemo:  
 12-gatedelay/40-tpcg/  
 two-phase-clock-gen

## Zweiphasentakt: Erzeugung (cont.)



- ▶ Verzögerungen geeignet wählen
  - ▶ Eins-Phasen der beiden Takte  $c_1$  und  $c_2$  sauber getrennt
- ⇒ nicht-überlappende Taktimpulse zur Ansteuerung von Schaltungen mit 2-Phasen-Taktung



## Beschreibung von Schaltwerken

- ▶ viele verschiedene Möglichkeiten
- ▶ graphisch oder textuell
  
- ▶ algebraische Formeln/Gleichungen
- ▶ Flusstafel und Ausgangstafel
  
- ▶ Zustandsdiagramm
- ▶ State-Charts (hierarchische Zustandsdiagramme)
  
- ▶ Programme (Hardwarebeschreibungssprachen)



## Flusstafel und Ausgangstafel

- ▶ entspricht der Funktionstabelle von Schaltnetzen
  
- ▶ **Flusstafel:** Tabelle für die Folgezustände als Funktion des aktuellen Zustands und der Eingabewerte  
 = beschreibt das  $\delta$ -Schaltnetz
  
- ▶ **Ausgangstafel:** Tabelle für die Ausgabewerte als Funktion des aktuellen Zustands (und der Eingabewerte [Mealy-Modell])  
 = beschreibt das  $\lambda$ -Schaltnetz

## Beispiel: Ampel

- ▶ vier Zustände: {rot, rot-gelb, grün, gelb}
- ▶ Codierung beispielsweise als 2-bit Vektor  $(z_1, z_0)$
- ▶ Flusstafel

Zustand	Codierung		Folgezustand	
	$z_1$	$z_0$	$z_1^+$	$z_0^+$
rot	0	0	0	1
rot-gelb	0	1	1	0
grün	1	0	1	1
gelb	1	1	0	0

## Beispiel: Ampel (cont.)

### ▶ Ausgangstafel

Zustand	Codierung		Ausgänge		
	$z_1$	$z_0$	$rt$	$ge$	$gr$
rot	0	0	1	0	0
rot-gelb	0	1	1	1	0
grün	1	0	0	0	1
gelb	1	1	0	1	0

- ▶ Funktionstabelle für drei Schaltfunktionen
- ▶ Minimierung z.B. mit KV-Diagrammen

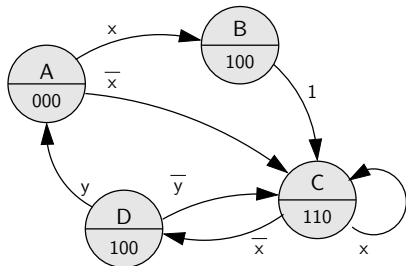




# Zustandsdiagramm

- ▶ **Zustandsdiagramm:** Grafische Darstellung eines Schaltwerks
- ▶ je ein Knoten für jeden Zustand
- ▶ je eine Kante für jeden möglichen Übergang
  
- ▶ Knoten werden passend benannt
- ▶ Kanten werden mit den Eingabemustern gekennzeichnet, bei denen der betreffende Übergang auftritt
  
- ▶ Moore-Schaltwerke: Ausgabe wird zusammen mit dem Namen im Knoten notiert
- ▶ Mealy-Schaltwerke: Ausgabe hängt vom Input ab und wird an den Kanten notiert

## Zustandsdiagramm: Moore-Automat



Zustand

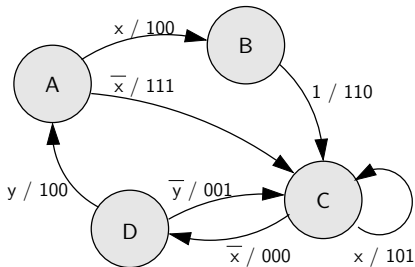


Übergang



- ▶ Ausgangswerte hängen nur vom Zustand ab
- ▶ können also im jeweiligen Knoten notiert werden
- ▶ Übergänge werden als Pfeile mit der Eingangsbelegung notiert, die den Übergang aktiviert
- ▶ ggf. Startzustand markieren (z.B. Segment, doppelter Kreis)

## Zustandsdiagramm: Mealy-Automat



Zustand



Übergang



- ▶ Ausgangswerte hängen nicht nur vom Zustand sondern auch von den Eingabewerten ab
- ▶ Ausgangswerte an den zugehörigen Kanten notieren
- ▶ übliche Notation: *Eingangsbelegung / Ausgangswerte*

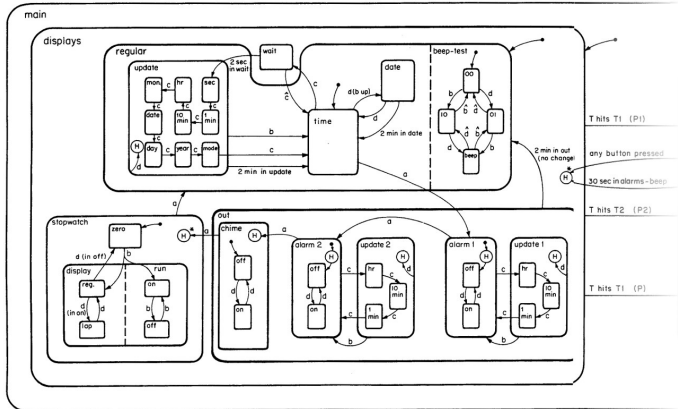
# „State-Charts“

- ▶ erweiterte Zustandsdiagramme
- 1. Hierarchien, erlauben Abstraktion
  - ▶ Knoten repräsentieren entweder einen Zustand
  - ▶ oder einen eigenen (Unter-) Automaten
  - ▶ *History-, Default-Mechanismen*
- 2. Nebenläufigkeit, parallel arbeitende FSMs
  - ▶ beliebte Spezifikation für komplexe Automaten, eingebettete Systeme, Kommunikationssysteme, Protokolle etc.
  - ▶ David Harel, *Statecharts – A visual formalism for complex systems*, CS84-05, Department of Applied Mathematics, The Weizmann Institute of Science, 1984 [Har87]

# „State-Charts“ (cont.)

## ► Beispiel Digitaluhr

Citizen quartz multi-alarm





# Endliche Automaten

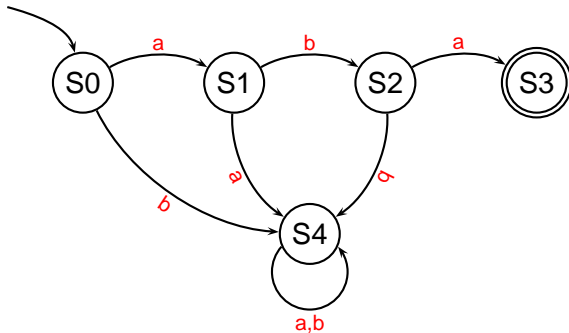
- ▶ eines der **gängigen Konzepte der Informatik**
- ▶ Modellierung, Entwurf und Simulation
  - ▶ zeitliche Abfolgen interner Systemzustände
  - ▶ bedingte Zustandswechsel
  - ▶ Reaktionen des Systems auf „Ereignisse“
  - ▶ Folgen von Aktionen
  - ▶ ...
- ▶ weitere „*spezielle*“ Anwendungsszenarien
  - ▶ verteilte Systeme (Client Server, etc.)
  - ▶ Echtzeitsysteme, ggf. mit Erweiterungen
  - ▶ eingebettete Systeme
  - ▶ ...

zahlreiche Beispiele

## Endliche Automaten (cont.)

- ▶ in der Programmierung. . .

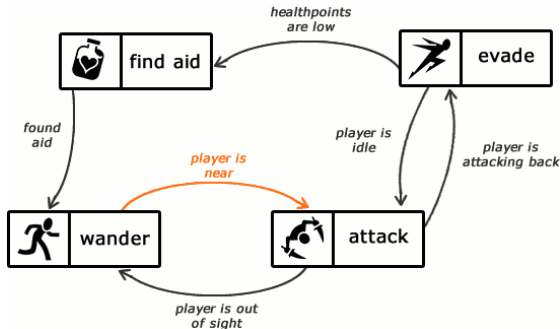
Erkennung des Wortes: „a b a“



# Endliche Automaten (cont.)

## Game-Design: Verhalten eines Bots

[gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867](http://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867)



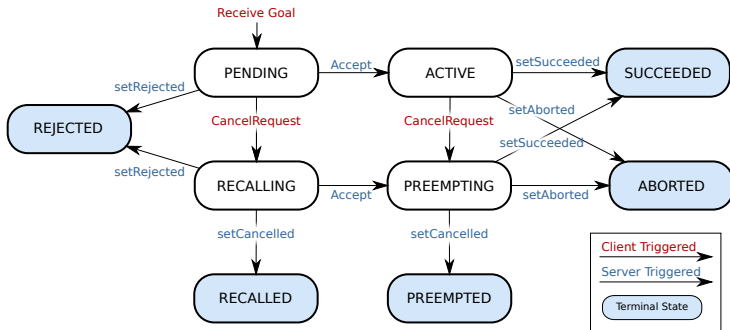


## Endliche Automaten (cont.)

- ▶ Beschreibung von Protokollen
- ▶ Verhalten verteilter Systeme: Client-Server Architektur

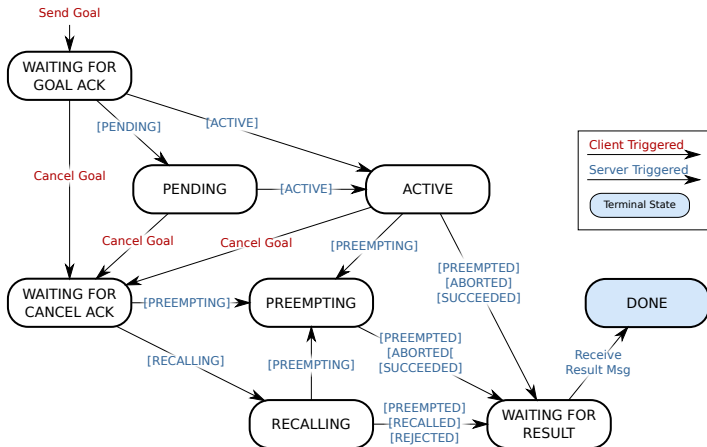
[wiki.ros.org/actionlib/DetailedDescription](http://wiki.ros.org/actionlib/DetailedDescription)

### Server State Transitions



# Endliche Automaten (cont.)

## Client State Transitions

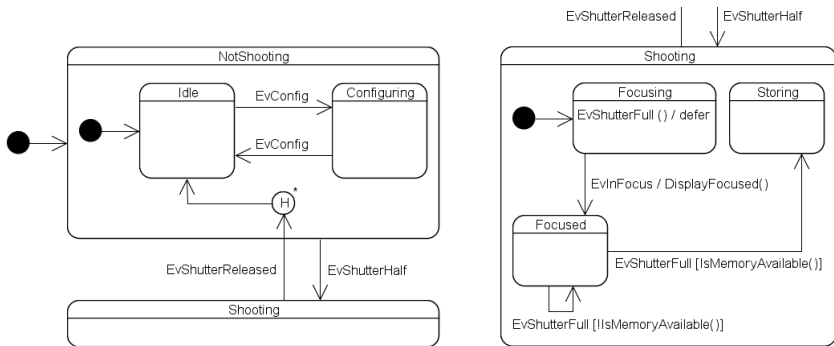


## Endliche Automaten (cont.)

- ▶ Unterstützung durch Bibliotheken und Werkzeuge

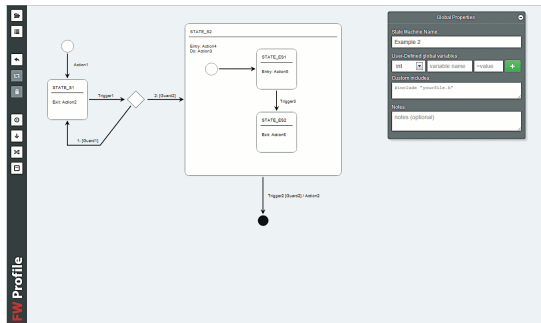
### State-Chart Bibliothek: Beispiel Digitalkamera

[www.boost.org/doc/libs/1\\_57\\_0/libs/statechart/doc](http://www.boost.org/doc/libs/1_57_0/libs/statechart/doc)



# Endliche Automaten (cont.)

## FSM Editor / Code-Generator



[code.google.com/p/fwprofile](http://code.google.com/p/fwprofile)

- ⇒ beliebig viele weitere Beispiele. . .  
 „Endliche Automaten“ werden in RS eher hardwarenah genutzt

## Hardwarebeschreibungssprachen

- ▶ Beschreibung eines Schaltwerks als Programm:
  - ▶ normale Hochsprachen C, Java
  - ▶ spezielle Bibliotheken für normale Sprachen SystemC, Hades
  - ▶ spezielle Hardwarebeschreibungssprachen Verilog, VHDL
  
- ▶ Hardwarebeschreibungssprachen unterstützen Modellierung paralleler Abläufe und des Zeitverhaltens einer Schaltung
- ▶ wird hier nicht vertieft
- ▶ lediglich zwei Beispiele: D-Flipflop in Verilog und VHDL



## D-Flipflop in Verilog

```

module dff (clock, reset, din, dout);
input clock, reset, din;
output dout;

reg dout;

  always @(posedge clock or reset)
  begin
    if (reset)
      dout = 1'b0;
    else
      dout = din;
    end
  endmodule

```

- ▶ Deklaration eines Moduls mit seinen Ein- und Ausgängen
- ▶ Deklaration der speichernden Elemente („reg“)
- ▶ Aktivierung des Codes bei Signalwechseln („posedge clock“)

# D-Flipflop in VHDL

## Very High Speed Integrated Circuit Hardware Description Language

```

library ieee;
use ieee.std_logic_1164.all;

entity dff is
port ( clock      : in  std_logic;
       reset      : in  std_logic;
       din        : in  std_logic;
       dout       : out std_logic);
end entity dff;

architecture behav of dff is
begin
  dff_p: process (reset, clock) is
  begin
    if reset = '1' then
      dout <= '0';
    elsif rising_edge(clock) then
      dout <= din;
    end if;
  end process dff_p;
end architecture behav;

```

*-- Black-Box Beschreibung*  
*-- Ein- und Ausgaenge*  
*--*  
*--*  
*--*  
*--*  
*--*  
*-- Verhaltensmodell*  
*--*  
*--*  
*--*  
*-- async. Reset*  
*--*  
*-- Taktvorderflanke*  
*--*  
*--*  
*--*



## Entwurf von Schaltwerken: sechs Schritte

1. Spezifikation (textuell oder graphisch, z.B. Zustandsdiagramm)
2. Aufstellen einer formalen Übergangstabelle
3. Reduktion der Zahl der Zustände
4. Wahl der Zustandscodierung und Aufstellen der Übergangstabelle
5. Minimierung der Schaltnetze
6. Überprüfung des realisierten Schaltwerks

ggf. mehrere Iterationen





## Entwurf von Schaltwerken: Zustandskodierung

### Vielfalt möglicher Codierungen

- ▶ binäre Codierung: minimale Anzahl der Zustände
- ▶ einschrittige Codes
- ▶ one-hot Codierung: ein aktives Flipflop pro Zustand
- ▶ applikationsspezifische Zwischenformen
  
- ▶ es gibt Entwurfsprogramme zur Automatisierung
- ▶ gemeinsame Minimierung des Realisierungsaufwands von Ausgangsfunktion, Übergangsfunktion und Speichergliedern



## Entwurf von Schaltwerken: Probleme

Entwurf ausgehend von Funktionstabellen problemlos

- ▶ alle Eingangsbelegungen und Zustände werden berücksichtigt
- ▶ don't-care Terme können berücksichtigt werden

zwei typische Fehler bei Entwurf ausgehend vom Zustandsdiagramm

- ▶ mehrere aktive Übergänge bei bestimmten Eingangsbelegungen  
⇒ Widerspruch
- ▶ keine Übergänge bei bestimmten Eingangsbelegungen  
⇒ Vollständigkeit

## Überprüfung der Vollständigkeit

$p$  Zustände, Zustandsdiagramm mit Kanten  $h_{ij}(x)$ :  
 Übergang von Zustand  $i$  nach Zustand  $j$  unter Belegung  $x$

- ▶ für jeden Zustand überprüfen:  
 kommen alle (spezifizierten) Eingangsbelegungen auch tatsächlich in Kanten vor?

$$\forall i : \bigvee_{j=0}^{2^p-1} h_{ij}(x) = 1$$

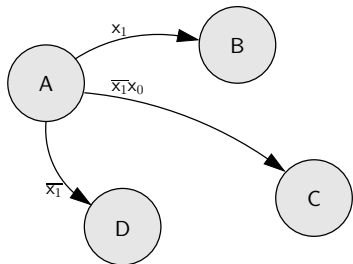
## Überprüfung der Widerspruchsfreiheit

$p$  Zustände, Zustandsdiagramm mit Kanten  $h_{ij}(x)$ :  
 Übergang von Zustand  $i$  nach Zustand  $j$  unter Belegung  $x$

- ▶ für jeden Zustand überprüfen:  
 kommen alle (spezifizierten) Eingangsbelegungen nur einmal vor?

$$\forall i : \bigvee_{j,k=0, j \neq k}^{2^p-1} (h_{ij}(x) \wedge h_{ik}(x)) = 0$$

## Vollständigkeit und Widerspruchsfreiheit: Beispiel



- ▶ Zustand A, Vollständigkeit:  $x_1 \vee \bar{x}_1 x_0 \vee \bar{x}_1 = 1$       vollständig
- ▶ Zustand A, Widerspruchsfreiheit: alle Paare testen

$$x_1 \wedge \bar{x}_1 x_0 = 0 \quad \text{ok}$$

$$x_1 \wedge \bar{x}_1 = 0 \quad \text{ok}$$

$$\bar{x}_1 x_0 \wedge \bar{x}_1 \neq 0 \quad \text{für } x_1 = 0 \text{ und } x_0 = 1 \text{ beide Übergänge aktiv}$$



## Schaltwerke: Beispiele

- ▶ Verkehrsampel
  - ▶ drei Varianten mit unterschiedlicher Zustandskodierung
  
- ▶ Zählschaltungen
  - ▶ einfacher Zähler, Zähler mit Enable (bzw. Stop),
  - ▶ Vorwärts-Rückwärts-Zähler, Realisierung mit JK-Flipflops und D-Flipflops
  
- ▶ Digitaluhr
  - ▶ BCD-Zähler
  
- ▶ ...

## Schaltwerksentwurf: Ampel

Beispiel Verkehrsampel:

- ▶ drei Ausgänge: {rot, gelb, grün}
- ▶ vier Zustände: {rot, rot-gelb, grün, gelb}
- ▶ zunächst kein Eingang, feste Zustandsfolge wie oben
  
- ▶ Aufstellen des Zustandsdiagramms
- ▶ Wahl der Zustandskodierung
- ▶ Aufstellen der Tafeln für  $\delta$ - und  $\lambda$ -Schaltnetz
- ▶ anschließend Minimierung der Schaltnetze
- ▶ Realisierung (je 1 D-Flipflop pro Zustandsbit) und Test

## Schaltwerksentwurf: Ampel – Variante 1

- ▶ vier Zustände, Codierung als 2-bit Vektor  $(z_1, z_0)$
- ▶ Fluss- und Ausgangstafel für binäre Zustandskodierung

Zustand	Codierung		Folgezustand		Ausgänge		
	$z_1$	$z_0$	$z_1^+$	$z_0^+$	$rt$	$ge$	$gr$
rot	0	0	0	1	1	0	0
rot-gelb	0	1	1	0	1	1	0
grün	1	0	1	1	0	0	1
gelb	1	1	0	0	0	1	0

- ▶ resultierende Schaltnetze

$$z_1^+ = (z_1 \wedge \overline{z_0}) \vee (\overline{z_1} \wedge z_0) = z_1 \oplus z_0$$

$$z_0^+ = \overline{z_0}$$

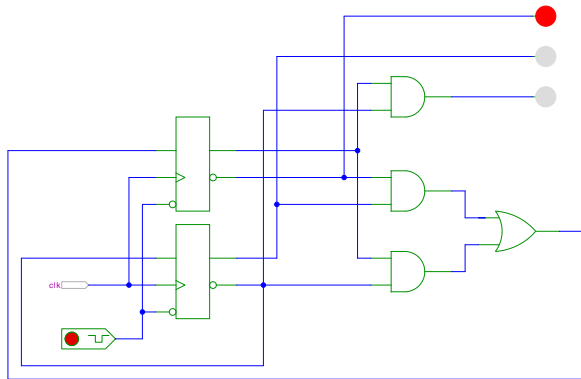
$$rt = \overline{z_1}$$

$$ge = z_0$$

$$gr = (z_1 \wedge \overline{z_0})$$



# Schaltwerksentwurf: Ampel – Variante 1 (cont.)



[HenHA] Hades Demo: 18- fsm/10- trafficlight/ampel\_41

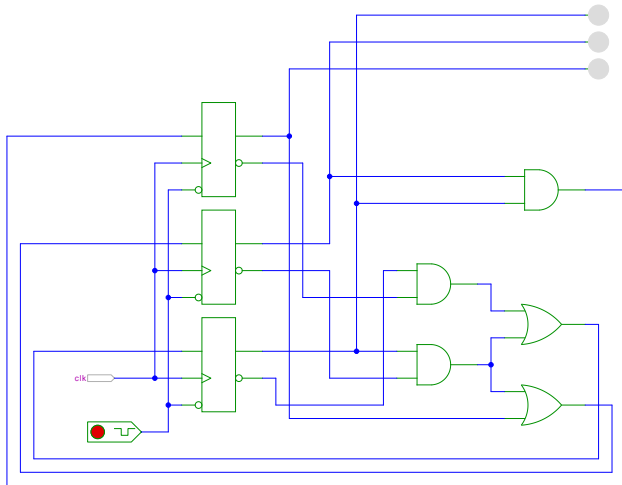
## Schaltwerksentwurf: Ampel – Variante 2

- ▶ vier Zustände, Codierung als 3-bit Vektor ( $z_2, z_1, z_0$ )
- ▶ Zustandsbits korrespondieren mit den aktiven Lampen:  
 $gr = z_2$ ,  $ge = z_1$  und  $rt = z_0$

Zustand	Codierung			Folgezustand		
	$z_2$	$z_1$	$z_0$	$z_2^+$	$z_1^+$	$z_0^+$
reset	0	0	0	0	0	1
rot	0	0	1	0	1	1
rot-gelb	0	1	1	1	0	0
grün	1	0	0	0	1	0
gelb	0	1	0	0	0	1

- ▶ benutzt 1-bit zusätzlich für die Zustände
- ▶ dafür wird die Ausgangsfunktion  $\lambda$  minimal (leer)

## Schaltwerksentwurf: Ampel – Variante 2 (cont.)



[HenHA] Hades Webdemo:  
18-fsm/10-trafficlight/  
ampel\_42



## Schaltwerksentwurf: Ampel – Variante 3

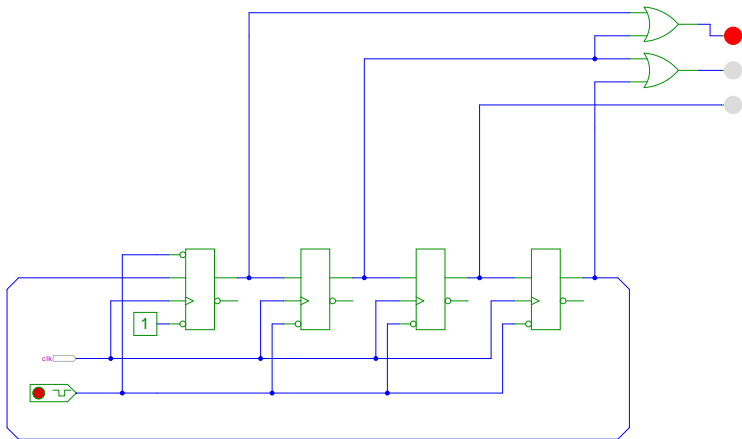
- ▶ vier Zustände, Codierung als 4-bit *one-hot* Vektor  $(z_3, z_2, z_1, z_0)$
- ▶ Beispiel für die Zustandskodierung

Zustand	Codierung				Folgezustand			
	$z_3$	$z_2$	$z_1$	$z_0$	$z_3^+$	$z_2^+$	$z_1^+$	$z_0^+$
rot	0	0	0	1	0	0	1	0
rot-gelb	0	0	1	0	0	1	0	0
grün	0	1	0	0	1	0	0	0
gelb	1	0	0	0	0	0	0	1

- ▶ 4-bit statt minimal 2-bit für die Zustände
- ▶ Übergangsfunktion  $\delta$  minimal (Automat sehr schnell)
- ▶ Ausgangsfunktion  $\lambda$  sehr einfach:

$$gr = z_2, ge = z_1 \vee z_3 \text{ und } rt = z_0 \vee z_1$$

## Schaltwerksentwurf: Ampel – Variante 3 (cont.)



[HenHA] Hades Demo: 18- fsm/10- trafficlight/ampel\_44



## Schaltwerksentwurf: Ampel – Zusammenfassung

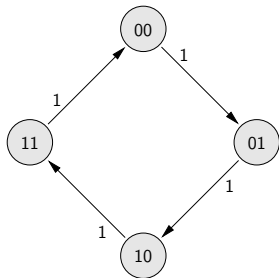
- ▶ viele Möglichkeiten der Zustandskodierung
- ▶ Dualcode: minimale Anzahl der Zustände
- ▶ applikations-spezifische Codierungen
- ▶ One-Hot Encoding: viele Zustände, einfache Schaltnetze
- ▶ ...
- ▶ Kosten/Performance des Schaltwerks abhängig von Codierung
- ▶ Heuristiken zur Suche nach (relativem) Optimum



# Zählschaltungen

- ▶ diverse Beispiele für Zählschaltungen
- ▶ Zustandsdiagramme und Flusstafeln
- ▶ Schaltbilder
- ▶  $n$ -bit Vorwärtszähler
- ▶  $n$ -bit Zähler mit Stop und/oder Reset
- ▶ Vorwärts-/Rückwärtszähler
- ▶ synchrone und asynchrone Zähler
- ▶ Beispiel: Digitaluhr (BCD-Zähler)

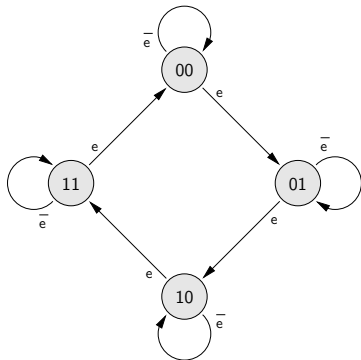
## 2-bit Zähler: Zustandsdiagramm



- ▶ Zähler als „trivialer“ endlicher Automat

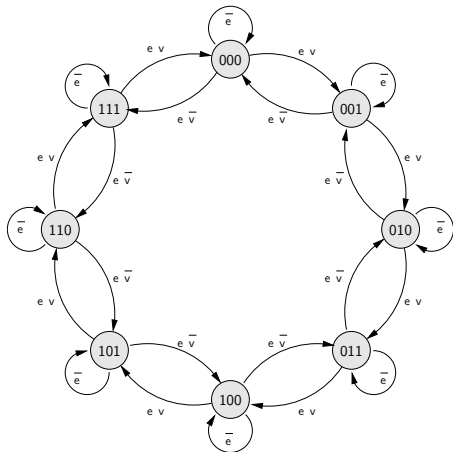


## 2-bit Zähler mit Enable: Zustandsdiagramm und Flusstafel



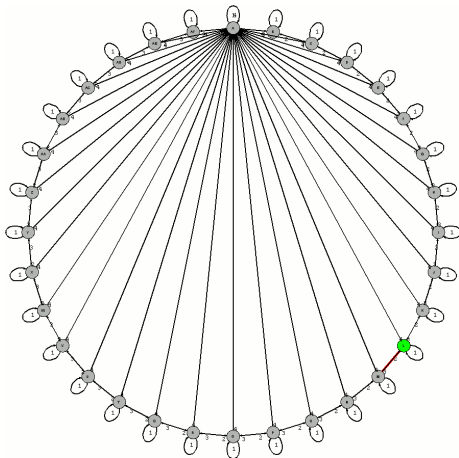
Eingabe	$e$	$\bar{e}$
Zustand	Folgezustand	
00	01	00
01	10	01
10	11	10
11	00	11

## 3-bit Zähler mit Enable, Vor-/Rückwärts



Eingabe	$e v$	$e \bar{v}$	$\bar{e} *$
Zustand	Folgezustand		
000	001	111	000
001	010	000	001
010	011	001	010
011	100	010	011
100	101	011	100
101	110	100	101
110	111	101	110
111	000	110	111

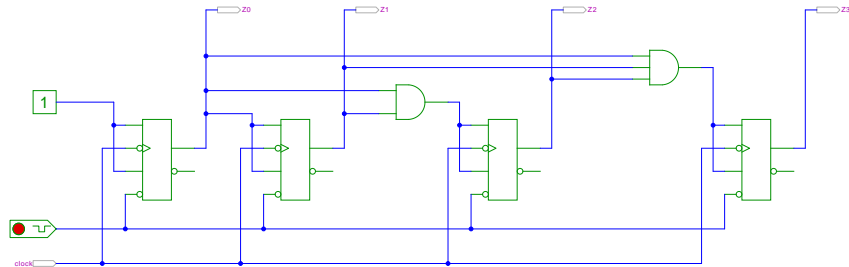
# 5-bit Zähler mit Reset: Zustandsdiagramm und Flusstafel



Zustand	Index der Eingabe			
	1	2	3	4
A	A	B	AF	A
B	B	C	A	A
C	C	D	B	A
D	D	E	C	A
E	E	F	D	A
F	F	G	E	A
G	G	H	F	A
H	H	I	G	A
I	I	J	H	A
J	J	K	I	A
K	K	L	J	A
L	L	M	K	A
M	M	N	L	A
N	N	O	M	A
O	O	P	N	A
P	P	Q	O	A
Q	Q	R	P	A
R	R	S	Q	A
S	S	T	R	A
T	T	U	S	A
U	U	V	T	A
V	V	W	U	A
W	W	X	V	A
X	X	Y	W	A
Y	Y	Z	X	A
Z	Z	AA	Y	A
AA	AA	AB	Z	A
AB	AB	AC	AA	A
AC	AC	AD	AB	A
AD	AD	AE	AC	A
AE	AE	AF	AD	A
AF	AF	A	AE	A

Eingabe 1: stop, 2: zählen, 3: rückwärts zählen, 4: Reset nach A

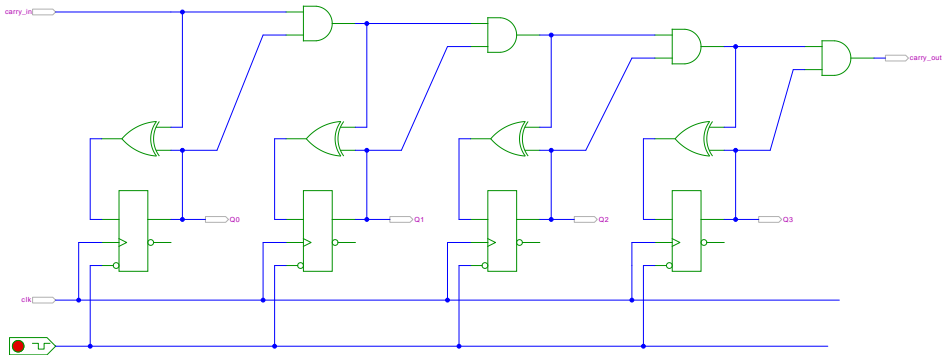
## 4-bit Binärzähler mit JK-Flipflops



[HenHA] Hades Demo: 30-counters/30-sync/sync

- ▶  $J_0 = K_0 = 1$ : Ausgang  $z_0$  wechselt bei jedem Takt
- ▶  $J_i = K_i = (z_0 z_1 \dots z_{i-1})$ : Ausgang  $z_i$  wechselt, wenn alle niedrigeren Stufen 1 sind

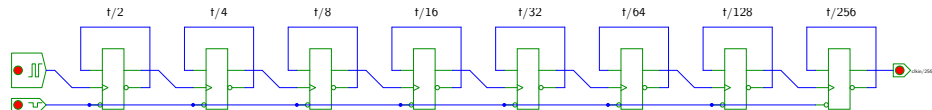
## 4-bit Binärzähler mit D-Flipflops (kaskadierbar)



[HenHA] Hades Demo: 30-counters/30-sync/sync-dff

- ▶  $D_0 = Q_0 \oplus c_{in}$  wechselt bei Takt, wenn  $c_{in}$  aktiv ist
- ▶  $D_i = Q_i \oplus (c_{in} Q_0 Q_1 \dots Q_{i-1})$  wechselt, wenn alle niedrigeren Stufen und Carry-in  $c_{in}$  1 sind

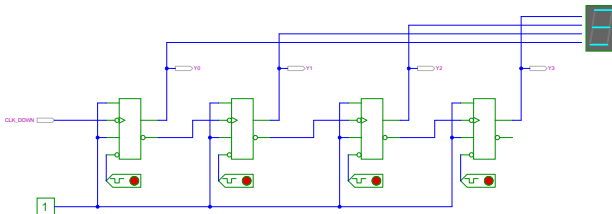
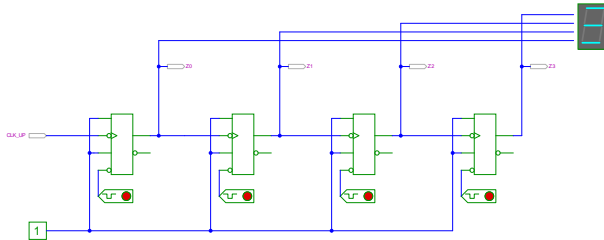
## Asynchroner $n$ -bit Zähler/Teiler mit D-Flipflops



[HenHA] Hades Demo: 30-counters/20-async/counter-dff

- ▶  $D_i = \overline{Q_i}$ : jedes Flipflop wechselt bei seinem Taktimpuls
- ▶ Takteingang  $C_0$  treibt nur das vorderste Flipflop
- ▶  $C_i = Q_{i-1}$ : Ausgang der Vorgängerstufe als Takt von Stufe  $i$
  
- ▶ erstes Flipflop wechselt bei jedem Takt  $\Rightarrow$  Zählrate  $C_0/2$
- ▶ zweites Flipflop bei jedem zweiten Takt  $\Rightarrow$  Zählrate  $C_0/4$
- ▶  $n$ -tes Flipflop bei jedem  $n$ -ten Takt  $\Rightarrow$  Zählrate  $C_0/2^n$
- ▶ sehr hohe maximale Taktrate
- **Achtung**: Flipflops schalten nacheinander, nicht gleichzeitig

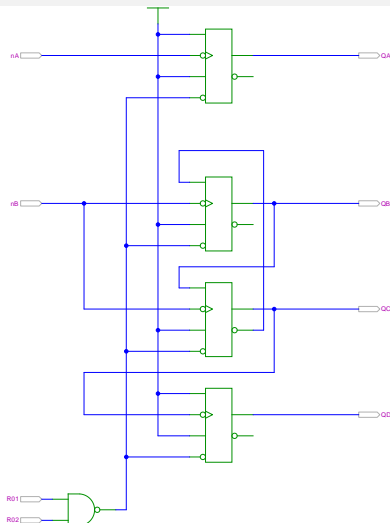
# Asynchrone 4-bit Vorwärts- und Rückwärtszähler



[HenHA] Hades Webdemo:  
30-counters/20-async/counter

## 4-bit 1:2, 1:6, 1:12-Teiler mit JK-Flipflops: IC 7492

- ▶ vier JK-Flipflops
- ▶ zwei Reseteingänge
- ▶ zwei Takteingänge
  
- ▶ Stufe 0 separat (1:2)
  
- ▶ Stufen 1...3 kaskadiert (1:6)
- ▶ Zustandsfolge  
 $\{000, 001, 010, 100, 101, 110\}$

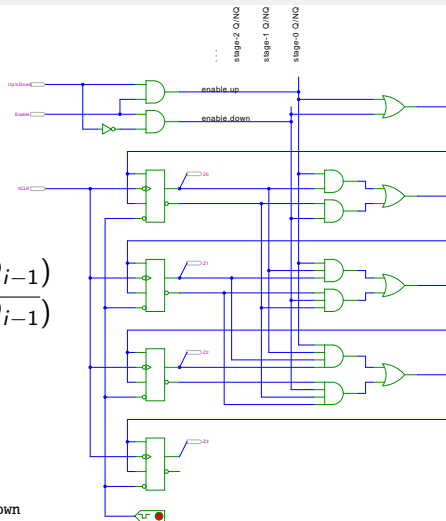


[HenHA] Hades Demo: 30-counters/60-ttl/SN7492



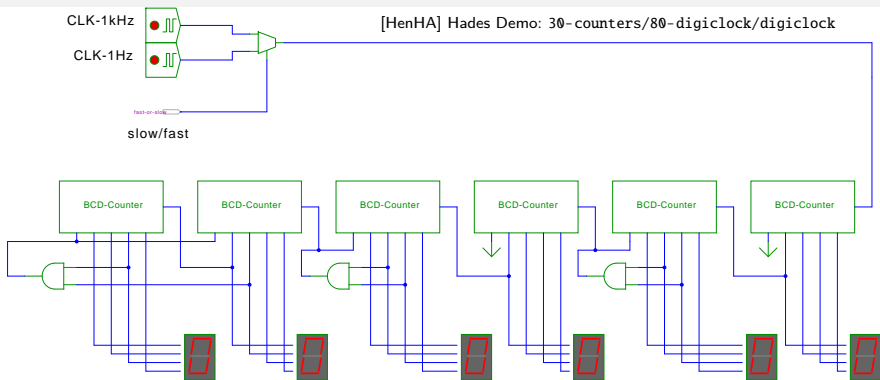
# 4-bit Vorwärts-Rückwärtszähler mit JK-Flipflops

- ▶ Eingänge: nClk  
 Enable  
 Up/nDown
- ▶ Umschaltung der *Carry-Chain*  
 up:  $J_i = K_i = (E Q_0 Q_1 \dots Q_{i-1})$   
 down:  $J_i = K_i = (E \overline{Q_0} \overline{Q_1} \dots \overline{Q_{i-1}})$



[HenHA] Hades Demo: 30-counters/40-updown/updown

# Digitaluhr mit BCD-Zählern



- ▶ Stunden Minuten Sekunden (hh:mm:ss)
- ▶ async. BCD-Zähler mit Takt (rechts) und Reset (links unten)
- ▶ Übertrag 1er- auf 10er-Stelle jeweils beim Übergang 9 → 0
- ▶ Übertrag und Reset der Zehner beim Auftreten des Wertes 6



## Funkgesteuerte DCF 77 Uhr

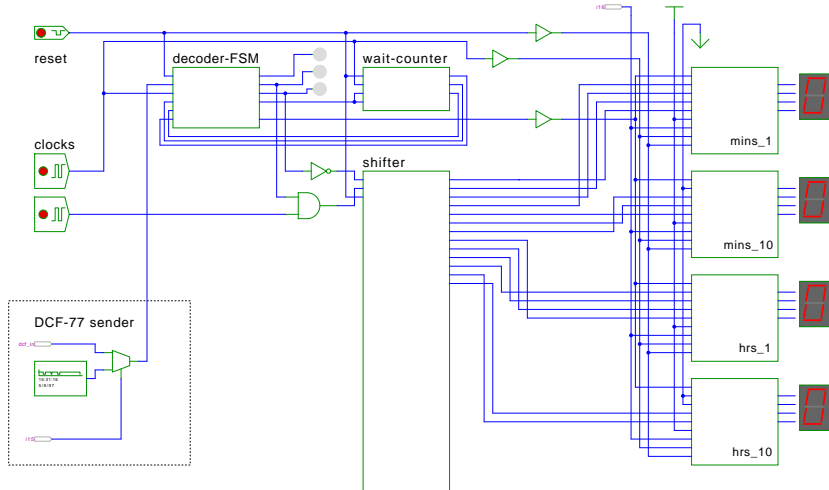
- ▶ Beispiel für eine komplexe Schaltung aus mehreren einfachen Komponenten
- ▶ mehrere gekoppelte Automaten, bzw. Zähler
  
- ▶ DCF 77 Zeitsignal
  - ▶ Langwelle 77,5 KHz
  - ▶ Sender nahe Frankfurt
  - ▶ ganz Deutschland abgedeckt
- ▶ pro Sekunde wird ein Bit übertragen
  - ▶ Puls mit abgesenktem Signalpegel: „Amplitudenmodulation“
  - ▶ Pulslänge: 100 ms entspricht Null, 200 ms entspricht Eins
  - ▶ Pulsbeginn ist Sekundenbeginn



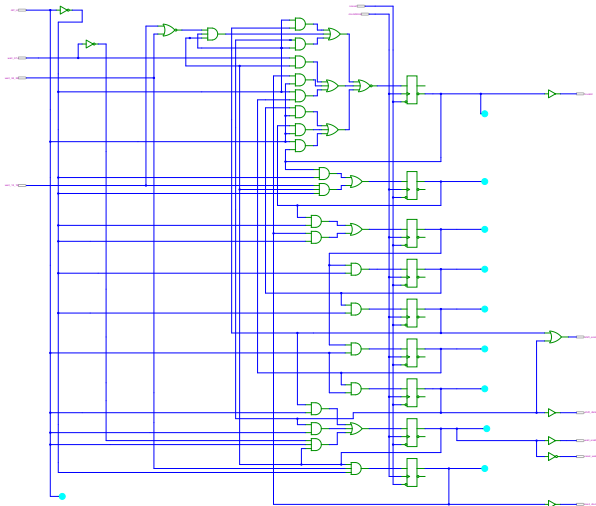
## Funkgesteuerte DCF 77 Uhr (cont.)

- ▶ pro Minute werden 59 Bits übertragen
  - ▶ Uhrzeit hh:mm (implizit Sekunden), MEZ/MESZ
  - ▶ Datum dd:mm:yy, Wochentag
  - ▶ Parität
  - ▶ fehlender 60ster Puls markiert Ende einer Minute
- ▶ Decodierung der Bits nach DCF 77 Protokoll mit entsprechend entworfenem Schaltwerk
- ▶ Beschreibung z.B.: [de.wikipedia.org/wiki/DCF77](http://de.wikipedia.org/wiki/DCF77)

# Funkgesteuerte DCF 77 Uhr: Gesamtsystem



# Funkgesteuerte DCF 77 Uhr: Decoder-Schaltwerk



[HenHA] Hades Webdemo:  
45-misc/80-dcf77/DecoderFSM

## Multiplex-Siebensegment-Anzeige

Ansteuerung mehrstelliger Siebensegment-Anzeigen?

- ▶ direkte Ansteuerung erfordert  $7 \cdot n$  Leitungen für  $n$  Ziffern
- ▶ und je einen Siebensegment-Decoder pro Ziffer

Zeit-Multiplex-Verfahren benötigt nur  $7 + n$  Leitungen

- ▶ die Anzeigen werden nacheinander nur ganz kurz eingeschaltet
- ▶ ein gemeinsamer Siebensegment-Decoder  
Eingabe wird entsprechend der aktiven Ziffer umgeschaltet
- ▶ das Auge sieht die leuchtenden Segmente und „mittelt“
- ▶ ab ca. 100 Hz Frequenz erscheint die Anzeige ruhig

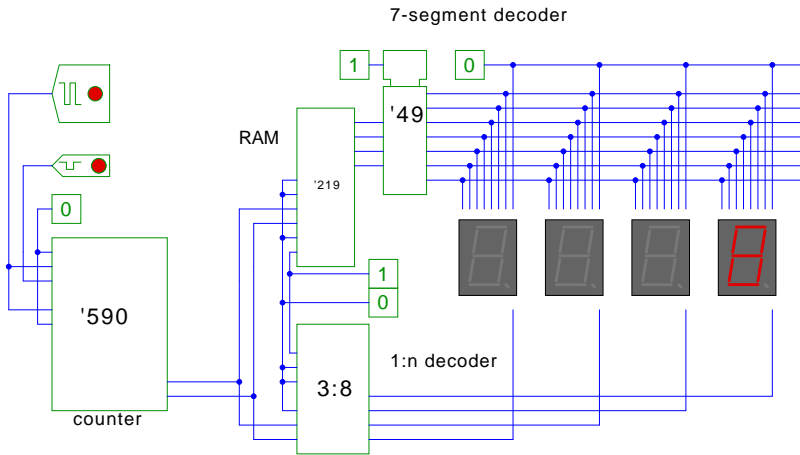
## Multiplex-Siebensegment-Anzeige (cont.)

Hades-Beispiel: Kombination mehrerer bekannter einzelner Schaltungen zu einem komplexen Gesamtsystem

- ▶ vierstellige Anzeige
- ▶ darzustellende Werte sind im RAM (74219) gespeichert
- ▶ Zähler-IC (74590) erzeugt 2-bit Folge {00, 01, 10, 11}
- ▶ 3:8-Decoder-IC (74138) erzeugt daraus die Folge {1110, 1101, 1011, 0111} um nacheinander je eine Anzeige zu aktivieren (low-active)
- ▶ Siebensegment-Decoder-IC (7449) treibt die sieben Segmentleitungen



# Multiplex-Siebensegment-Anzeige (cont.)



[HenHA] Hades Demo: 45-misc/50-displays/multiplexed-display

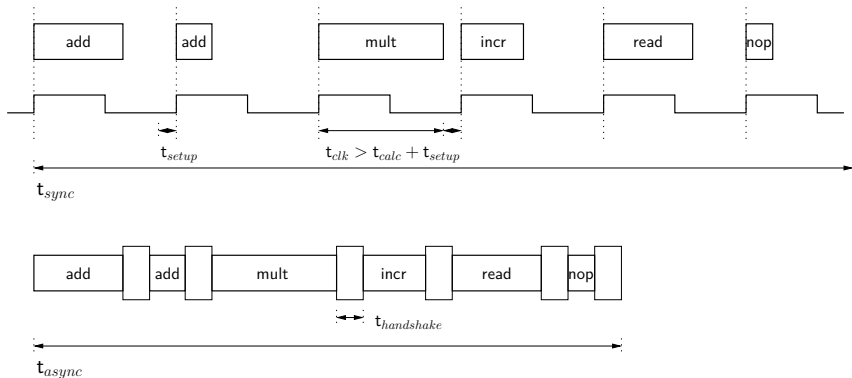
## Ausblick: Asynchrone Schaltungen

- ▶ Kosten und Verzögerung pro Gatter fallen
- ▶ zentraler Takt zunehmend problematisch: Performance, Energieverbrauch, usw.
- ▶ alle Rechenwerke warten auf langsamste Komponente

Umstieg auf nicht-getaktete Schaltwerke?!

- ▶ *Handshake*-Protokolle zwischen Teilschaltungen
  - ▶ Berechnung startet, sobald benötigte Operanden verfügbar
  - ▶ Rechenwerke signalisieren, dass Ergebnisse bereitstehen
- + kein zentraler Takt notwendig  $\Rightarrow$  so schnell wie möglich
- Probleme mit Deadlocks und Initialisierung

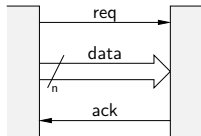
## Asynchrone Schaltungen: Performance



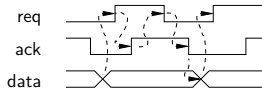
- ▶ synchron: Pipelining/Path-Balancing können Verschnitt verringern
- ▶ asynchron: Operationen langsamer wegen „completion detection“

# Zwei-Phasen und Vier-Phasen Handshake

bundled data

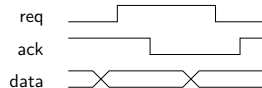


four-phase



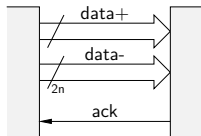
"level"

two-phase

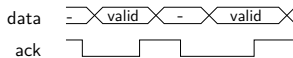


"edge"

dual rail



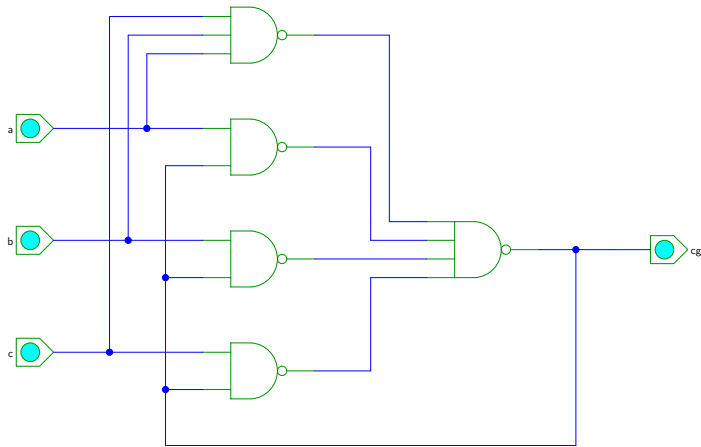
four-phase



	d+	d-
empty	0	0
valid "0"	0	1
valid "1"	1	0
unused	1	1



# Muller C-Gate: 3-Eingänge

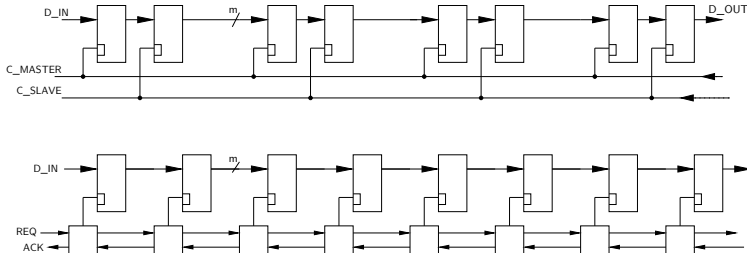




## Asynchrone Schaltungen: Micropipeline

- ▶ einfaches Modell einer generischen nicht-getakteten Schaltung
- ▶ Beispiel zum Entwurf und zur Kaskadierung
- ▶ Muller C-Gate als Speicherglieder
- ▶ beliebige Anzahl Stufen
  
- ▶ neue Datenwerte von links in die Pipeline einfüllen
- ▶ Werte laufen soweit nach rechts wie möglich
- ▶ solange bis Pipeline gefüllt ist
  
- ▶ Datenwerte werden nach rechts entnommen
- ▶ Pipeline signalisiert automatisch, ob Daten eingefüllt oder entnommen werden können

# Micropipeline: Konzept

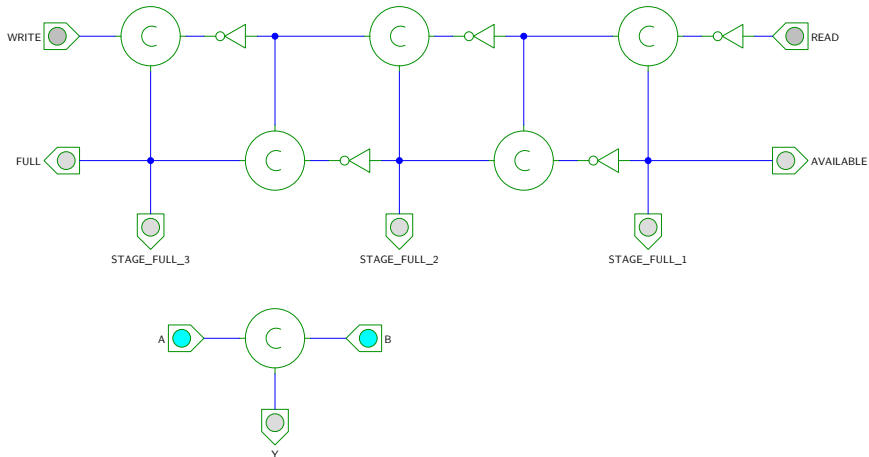


$n$ -stufige Micropipeline vs. getaktetes Schieberegister

- ▶ lokales Handshake statt globalem Taktsignal
- ▶ Datenkapazität entspricht  $2n$ -stufigem Schieberegister
- ▶ leere Latches transparent: schnelles Einfüllen
- ▶ „elastisch“: enthält  $0 \dots 2n$  Datenworte



# Micropipeline: Demo mit C-Gates



[HenHA] Hades Demo: 16- flipflops/80-micropipeline



## Literatur

- [SS04] W. Schiffmann, R. Schmitz: *Technische Informatik 1 – Grundlagen der digitalen Elektronik*.  
 5. Auflage, Springer-Verlag, 2004. ISBN 978-3-540-40418-7
- [Rei98] N. Reifschneider: *CAE-gestützte IC-Entwurfsmethoden*.  
 Prentice Hall, 1998. ISBN 3-8272-9550-5
- [WE94] N.H.E. Weste, K. Eshraghian:  
*Principles of CMOS VLSI design – A systems perspective*.  
 2nd edition, Addison-Wesley, 1994. ISBN 0-201-53376-6
- [Har87] D. Harel: *Statecharts: A visual formalism for complex systems*. in: *Sci. Comput. Program.* 8 (1987), Juni, Nr. 3,  
 S. 231-274. ISSN 0167-6423

## Interaktives Lehrmaterial

[HenHA] N. Hendrich: *HADES — HAMBURG DEsign System*.  
 Universität Hamburg, FB Informatik, Lehrmaterial.  
[tams.informatik.uni-hamburg.de/applets/hades](http://tams.informatik.uni-hamburg.de/applets/hades)

[Hei05] K. von der Heide: *Vorlesung: Technische Informatik 1 — interaktives Skript*. Universität Hamburg, FB Informatik, 2005  
[tams.informatik.uni-hamburg.de/lectures/2004ws/vorlesung/t1](http://tams.informatik.uni-hamburg.de/lectures/2004ws/vorlesung/t1)