Rechnerstrukturen und Betriebssysteme

64-040 Vorlesung RSB
Kapitel 10: Schaltwerke

Norman Hendrich

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Technische Aspekte Multimodaler Systeme

Wintersemester 2025/2026

Gliederung

Schaltwerke
- Definition und Modelle
- Asynchrone (ungetaktete) Schaltungen
- Synchrone (getaktete) Schaltungen
- Flipflops
- Zeitbedingungen
- Taktschemata
- Beschreibung von Schaltwerken
- Entwurf von Schaltwerken
- Beispiele
- Literatur

Schaltwerke

- **Schaltwerk**: Schaltung mit Rückkopplungen und Verzögerungen

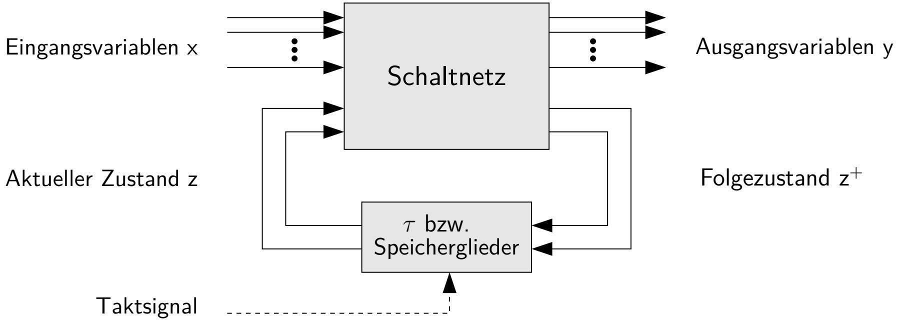
- fundamental andere Eigenschaften als Schaltnetze
- Ausgangswerte nicht nur von Eingangswerten abhängig, sondern auch von der zeitlichen Abfolge der Eingangswerte

$\Rightarrow$ interner Zustand z, repräsentiert „Vorgeschichte“

- stabile Zustände $\Rightarrow$ Speicherung von Information
- bei schlechtem Entwurf: chaotisches Verhalten und Instabilitäten

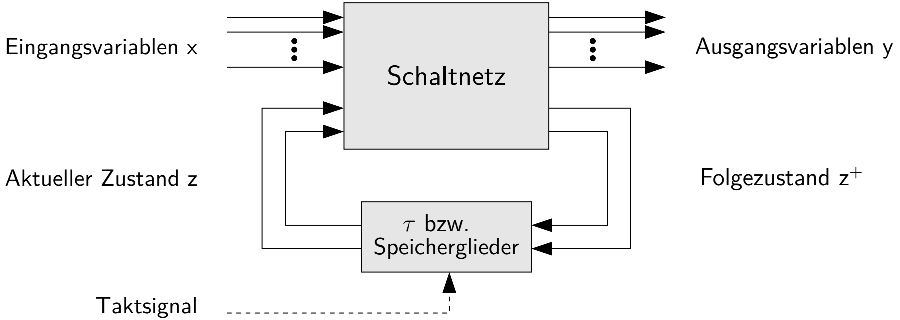
- Definition von Schaltwerken enthält Rückkopplungen
  - explizit, verzögert
    $z = \text{delay}(\overline{z})$
  - indirekt, mehrere Variablen beteiligt
    $z_1 = \overline{(a \wedge z_2)} \quad z_2 = \overline{(b \wedge z_1)}$

Schaltwerke: Blockschaltbild



- Eingangsvariablen x und Ausgangsvariablen y
- Aktueller Zustand z
- Folgezustand z^+
- Rückkopplung läuft über Verzögerungen τ / 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 und Änderungen der Eingangssignale wirken sich unmittelbar aus (nach Gatterverzögerungen τ)
- potenziell höhere Arbeitsgeschwindigkeit
- aber sehr aufwändiger Entwurf
- deutlich fehleranfälliger (z.B. leicht veränderte Gatterverzögerungen durch Bauteil-Toleranzen, Spannungsschwankungen usw.)

Theorie: Endliche Automaten

FSM – **F**inite **S**tate **M**achine

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

Mealy-Modell und Moore-Modell

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

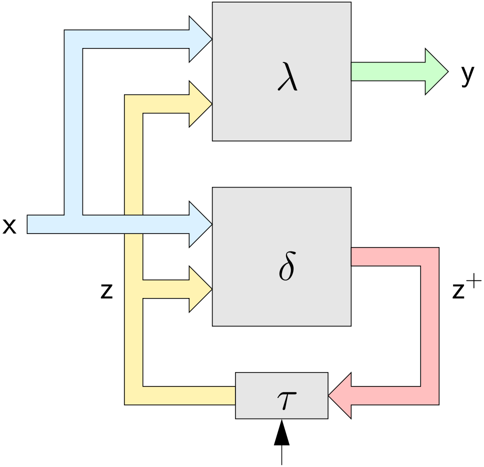
- **Ausgabefunktion**: $y = \lambda(z,x)$ Mealy

  $y = \lambda(z)$ Moore
- **Überführungsfunktion**: $z^+ = \delta(z,x)$ Moore und Mealy

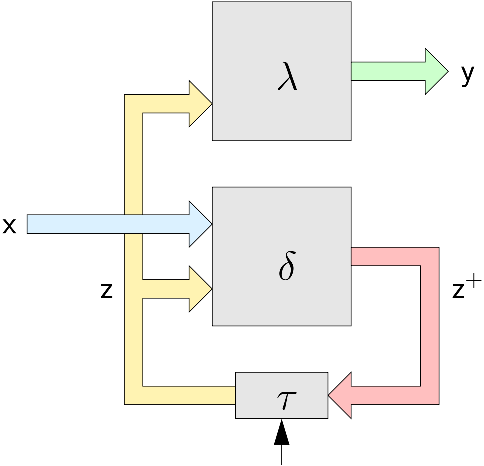
- **Speicherglieder** oder Verzögerung τ im Rückkopplungspfad

Mealy-Modell und Moore-Modell (cont.)

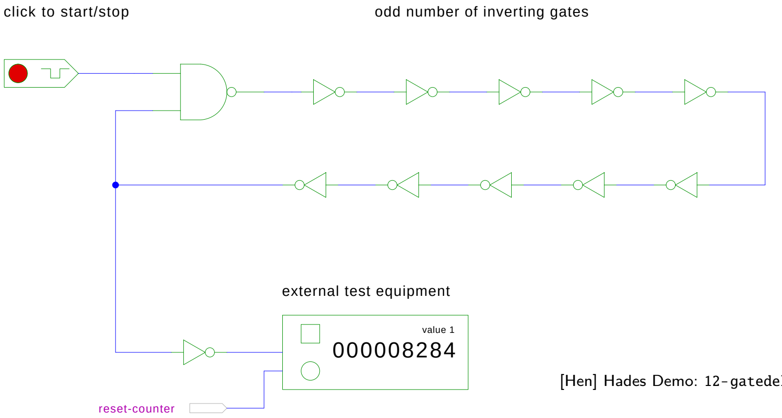
- Mealy-Automat



Moore-Automat



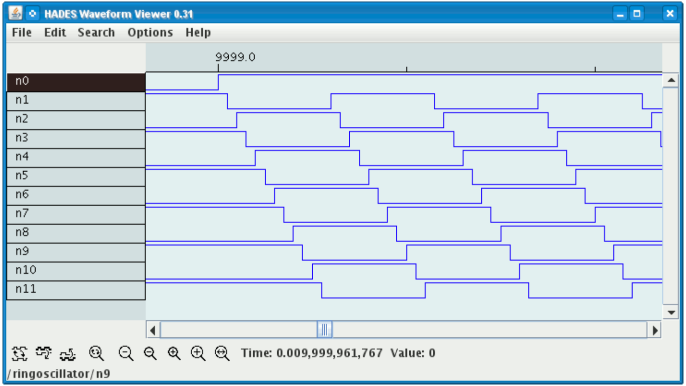
Asynchrone Schaltungen: Beispiel Ringoszillator



[Hen] Hades Demo: `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
- Verzögerungen der elektrischen Bauelemente sind wegen Fertigungsstreuung nicht genau bekannt und können sich im Betrieb ändern
- Variation durch Umweltparameter
  z.B. Temperatur, Versorgungsspannung, Alterung

$\Rightarrow$ sehr schwierig, die korrekte Funktion zu garantieren

$\Rightarrow$ fertig entworfene 1-bit Schaltwerke, **Flipflops**

- in der Praxis deshalb **synchrone Schaltwerke**
  - normale Schaltnetze für λ und δ (ohne Rückkopplungen)
  - **Flipflops** als Zeitglieder (asynchron)

Synchrone Schaltungen

- alle Rückkopplungen der Schaltung laufen über spezielle Zeitglieder: „Flipflops“
- diese definieren/speichern einen stabilen Zustand des Schaltnetzes, unabhängig von den Eingabewerten und Vorgängen im δ-Schaltnetz
- Hinzufügen eines zusätzlichen Eingangssignals: „Takt“
- die Zeitglieder werden über das Taktsignal gesteuert
  verschiedene Möglichkeiten: Pegel- und Flankensteuerung, Mehrphasentakte ...

$\Rightarrow$ synchrone Schaltwerke sind wesentlich einfacher zu entwerfen und zu analysieren als asynchrone Schaltungen

- **Flipflops**, bzw. **bistabile Bauelemente** (Kippglieder)
  - je Flipflop: zwei stabile Zustände $\Rightarrow$ speichert 1 Bit
  - Übergang zwischen diesen Zuständen durch geeignete Ansteuerung

Flipflops

- Bezeichnung für **elementare** Schaltwerke
- mit genau zwei Zuständen Z_0 und Z_1

- Ausgang als Q bezeichnet und dem Zustand gleichgesetzt
- meistens auch invertierter Ausgang $\overline{Q}$ verfügbar

- Flipflops sind selbst nicht getaktet
- sondern „sauber entworfene“ asynchrone Schaltwerke
- Anwendung als Verzögerungs-/Speicherelemente in getakteten Schaltwerken

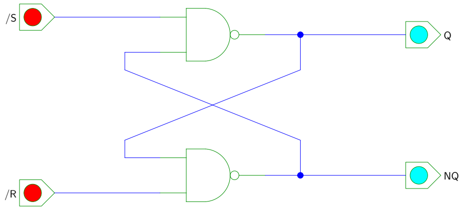
Flipflops: Typen

- Basis-Flipflop „Reset-Set-Flipflop“
- getaktetes RS-Flipflop

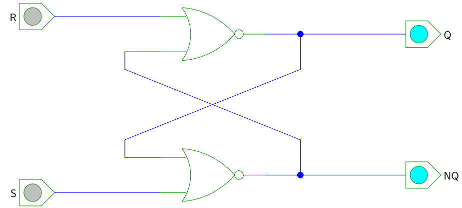
- pegelgesteuertes D-Flipflop „D-Latch“
- flankengesteuertes D-Flipflop „D-Flipflop“

- JK-Flipflop
- weitere. . .

RS-Flipflop: NAND- und NOR-Realisierung



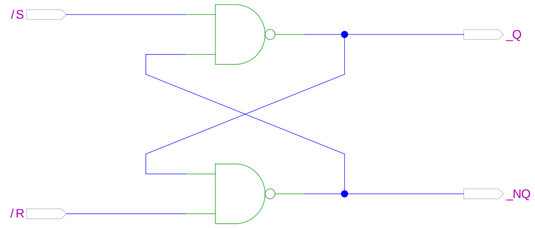
/S	/R	Q	NQ	NAND
0	0	1	1	forbidden
0	1	1	0	
1	0	0	1	
1	1	Q*	NQ*	store

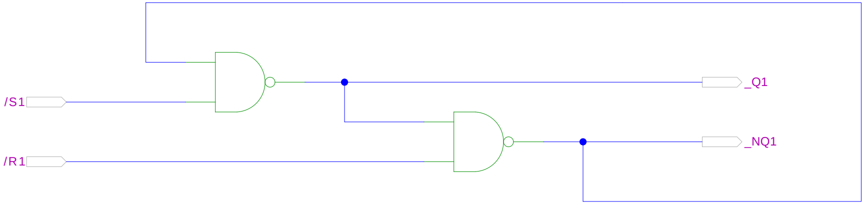


S	R	Q	NQ	NOR
0	0	Q*	NQ*	store
0	1	0	1	
1	0	1	0	
1	1	0	0	forbidden

[Hen] Hades Demo: `16-flipflops/10-srff/srff`

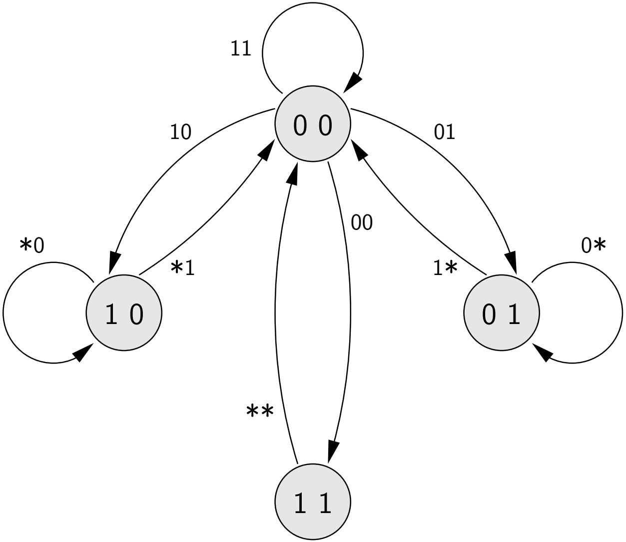
RS-Flipflop: Varianten des Schaltbilds





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

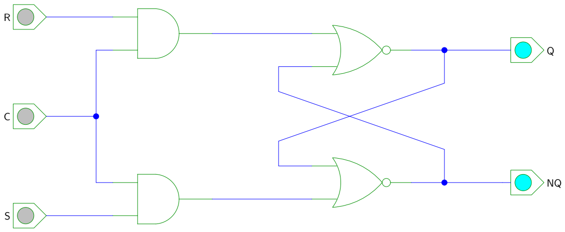
NOR RS-Flipflop: Zustandsdiagramm und Flusstafel



	Eingabe [S R]			
	0 0	0 1	1 1	1 0
Zustand	**Folgezustand** [$Q\,\overline{Q}$]			
0 0	1 1	0 1	0 0	1 0
0 1	0 1	0 1	0 0	0 0
1 1	0 0	0 0	0 0	0 0
1 0	1 0	0 0	0 0	1 0

stabiler Zustand

RS-Flipflop mit Takt



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

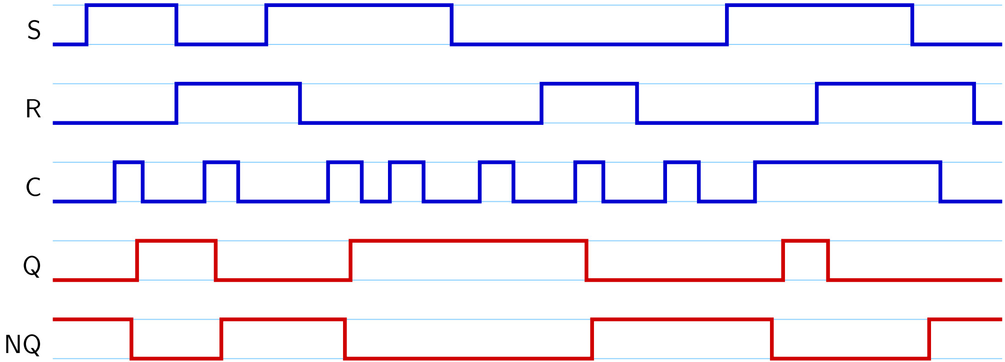
[Hen] Hades Demo: `16-flipflops/10-srff/clocked-srff`

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

- $$Q = \overline{(NQ \vee (R \wedge C))}$$
  $$NQ = \overline{(Q \vee (S \wedge C))}$$

RS-Flipflop mit Takt (cont.)

- Impulsdiagramm



- $$Q = \overline{(NQ \vee (R \wedge C))}$$
  $$NQ = \overline{(Q \vee (S \wedge C))}$$

Pegelgesteuertes D-Flipflop (D-Latch)

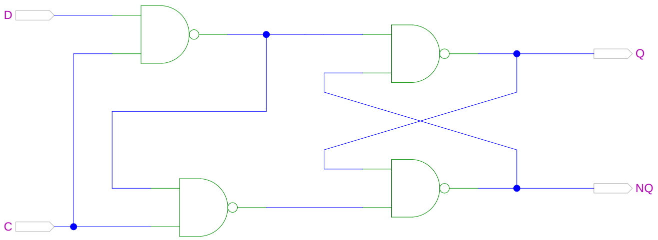
- Takteingang C
- Dateneingang D
- aktueller Zustand Q, Folgezustand Q^+

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

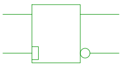
- Wert am Dateneingang wird durchgeleitet, wenn das Taktsignal
  1 ist $\Rightarrow$ *high*-aktiv
  0 ist $\Rightarrow$ *low*-aktiv

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

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

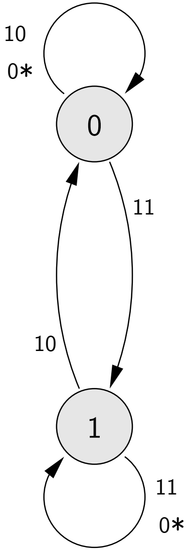


- Symbol



[Hen] Hades Demo: `16-flipflops/20-dlatch/dlatch`

D-Latch: Zustandsdiagramm und Flusstafel



	Eingabe [C D]			
	0 0	0 1	1 1	1 0
Zustand [Q]	Folgezustand [Q^+]			
0	0	0	1	0
1	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
$\uparrow$	0	0
$\uparrow$	1	1

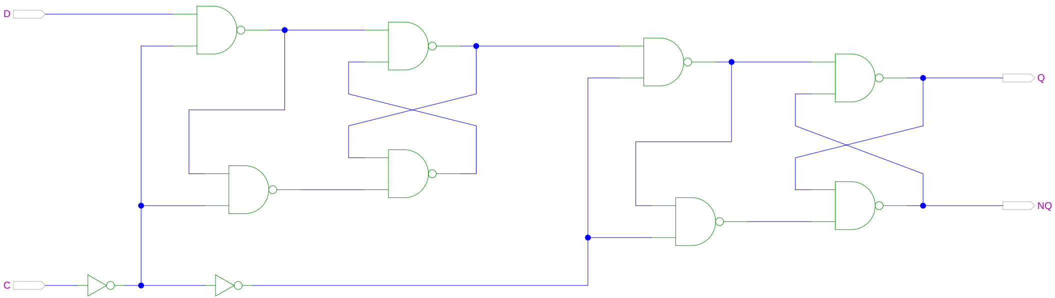
- Wert am Dateneingang wird gespeichert, wenn das Taktsignal sich von 0 auf 1 ändert $\Rightarrow$ Vorderflankensteuerung
  –"– 1 auf 0 ändert $\Rightarrow$ Rückflankensteuerung
- Realisierung als Master-Slave Flipflop oder direkt

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

$\Rightarrow$ dies entspricht effektiv einer **Flankensteuerung**:
Wert an D nur relevant, kurz bevor Takt auf $C = 1$ wechselt

Master-Slave D-Flipflop (cont.)



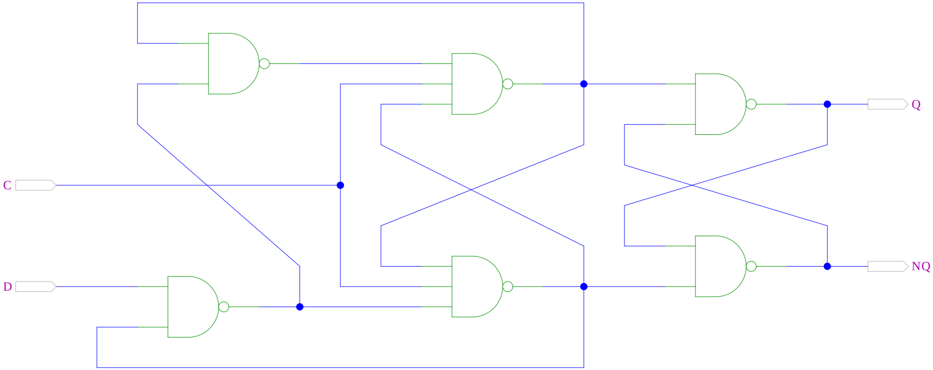
[Hen] Hades Demo: `16-flipflops/20-dlatch/dff`

- 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 essenziell
- Einhalten der Vorlauf- und Haltezeiten vor/nach der Taktflanke (s.u. *Zeitbedingungen*)

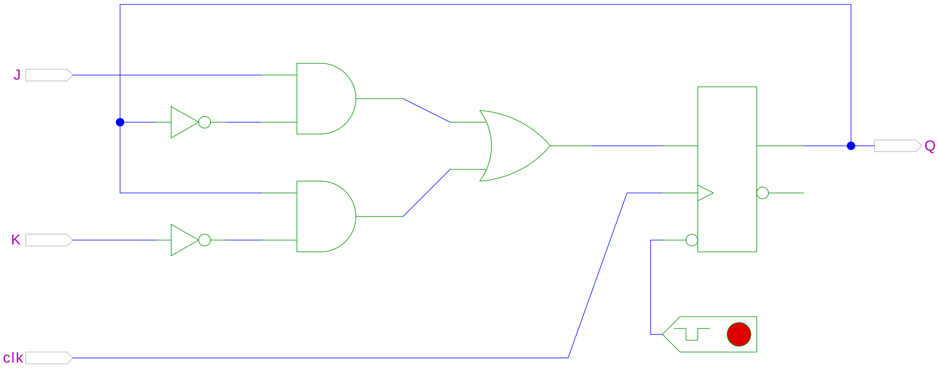
JK-Flipflop

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

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

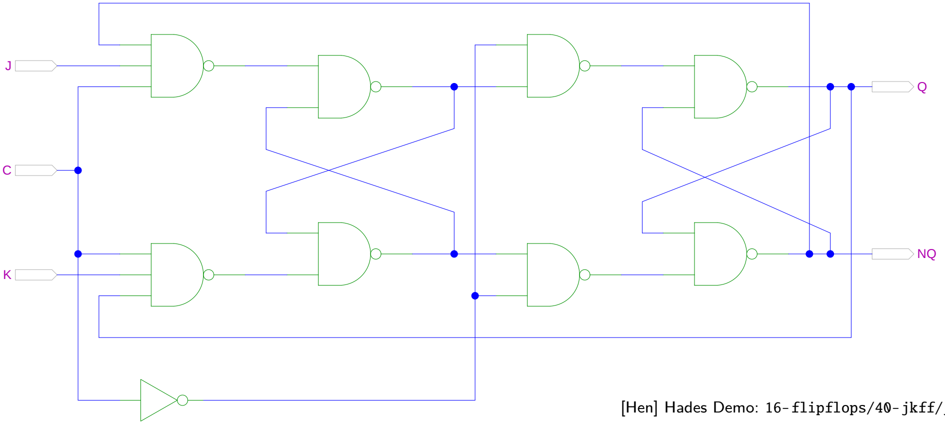
- universelles Flipflop, sehr flexibel einsetzbar
- in integrierten Schaltungen nur noch selten verwendet (höherer Hardware-Aufwand als Latch/D-Flipflop)

JK-Flipflop: Realisierung mit D-Flipflop



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

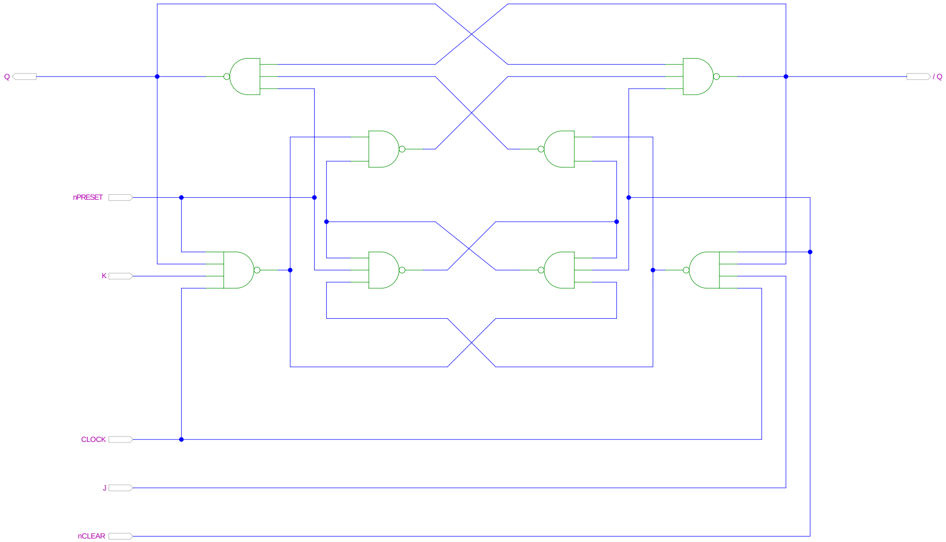
JK-Flipflop: Realisierung als Master-Slave Schaltung



[Hen] Hades Demo: `16-flipflops/40-jkff/jkff`

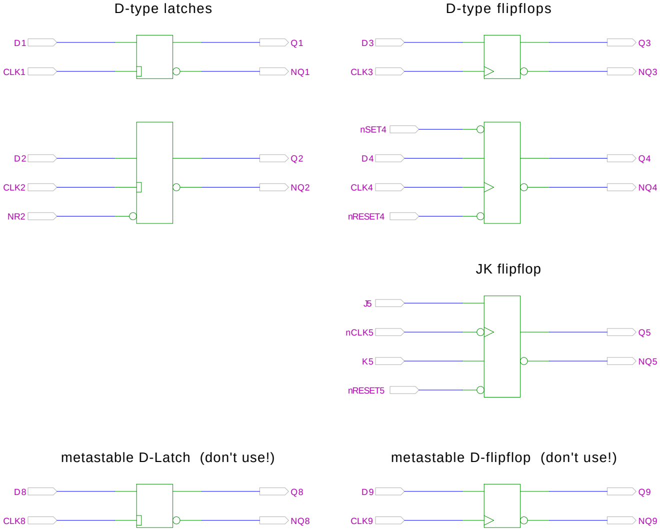
- Achtung: Schaltung wegen Rückkopplungen schwer zu initialisieren

JK-Flipflop: tatsächliche Schaltung im IC 7476



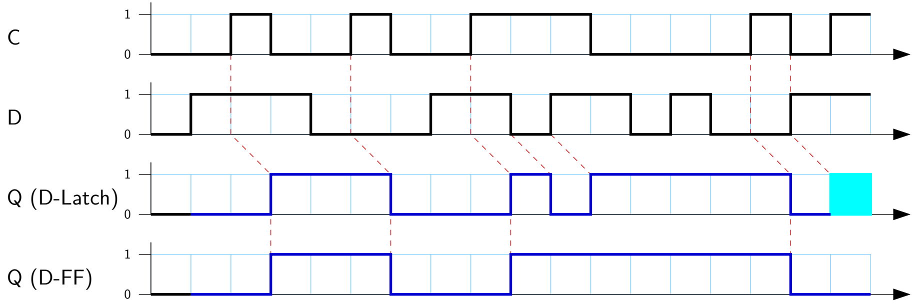
[Hen] Hades Demo: `16-flipflops/40-jkff/SN7476-single`

Flipflop-Typen: Komponenten/Symbole in Hades



[Hen] Hades Demo: `16-flipflops/50-ffdemo/flipflopdemo`

Flipflop-Typen: Impulsdiagramme



- pegel- und vorderflankengesteuertes Flipflop im Vergleich
- beide Flipflops hier mit jeweils einer Zeiteinheit Verzögerung
- undefinierte Werte im Latch (cyan dargestellt)
  - Verletzung der Zeitbedingungen!
  - gleichzeitiger Wechsel von C und D
  - in der Realität wird natürlich ein Wert 0 oder 1 gespeichert, er ist aber von externen Parametern abhängig: Temperatur, Versorgungsspannung etc.

Flipflops: Zeitbedingungen

- Flipflops werden entwickelt, um Schaltwerke einfacher zu entwerfen und betreiben
- Umschalten des Zustandes wird synchron 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

⇒ Daten- und Takteingänge dürfen sich nicht gleichzeitig ändern

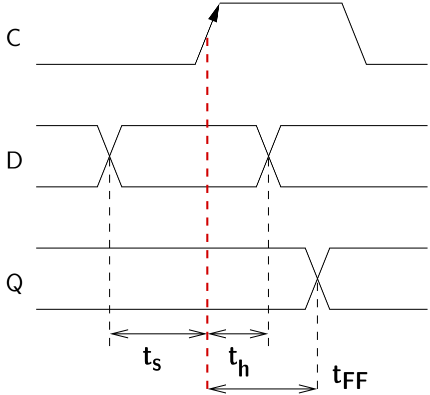
Welcher Wert wird gespeichert?

⇒ „Vorlauf- und Haltezeiten“ (*setup-* / *hold-time*)

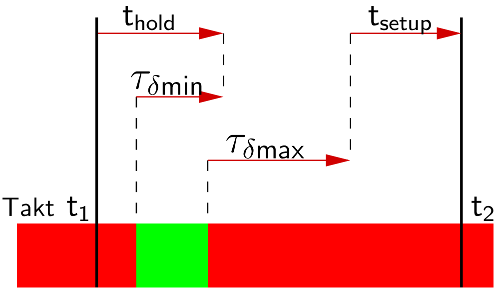
Flipflops: Vorlauf- und Haltezeit

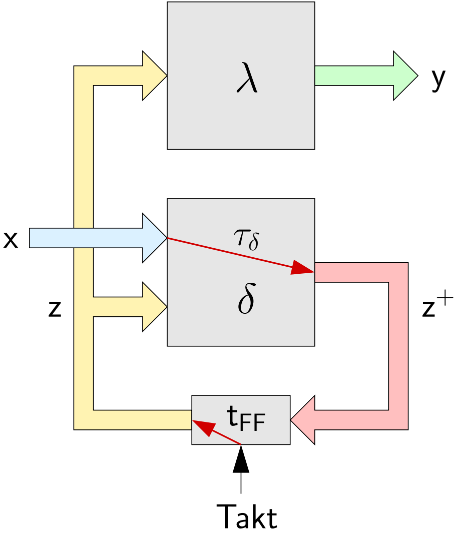
- t_s Vorlaufzeit (engl. *setup-time*): Zeitintervall, innerhalb dessen das Datensignal *vor dem nächsten Takt* stabil anliegen muss
- t_h Haltezeit (engl. *hold-time*): Zeitintervall, innerhalb dessen das Datensignal *nach einem Takt* noch stabil anliegen muss
- t_{FF} Ausgangsverzögerung

⇒ Verletzung der Zeitbedingungen „undefinierter“ Wert an Q



Zeitbedingungen: Eingangsvektor

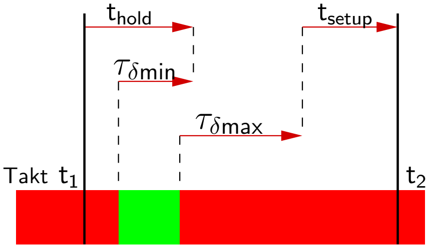




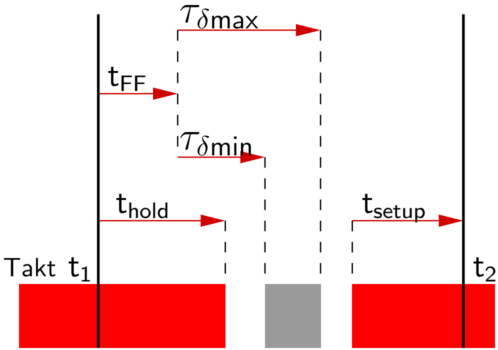
Zeitbedingungen: Eingangsvektor (cont.)

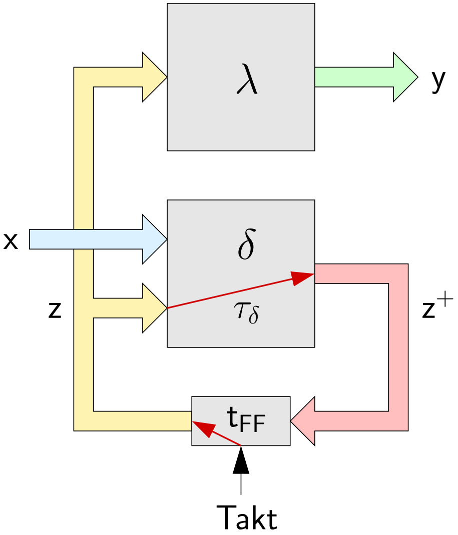
- Änderungen der Eingangswerte x werden beim Durchlaufen von δ mindestens um $\tau_{\delta_{\mathrm{min}}}$, bzw. maximal um $\tau_{\delta_{\mathrm{max}}}$ verzögert
- um die Haltezeit der Zeitglieder einzuhalten, darf x sich nach einem Taktimpuls frühestens zum Zeitpunkt $(t_1 + t_{\mathrm{hold}} -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

$\Rightarrow$ Änderungen dürfen nur innerhalb des grün markierten Zeitintervall erfolgen



Zeitbedingungen: interner Zustand

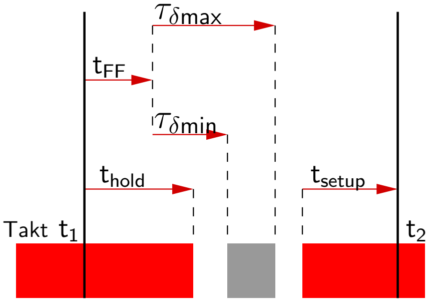




Zeitbedingungen: interner Zustand (cont.)

- zum Zeitpunkt t_1 wird ein Taktimpuls ausgelöst
- nach t_{FF} haben die Zeitglieder (Flipflops) den aktuellen Eingangswert z^+ übernommen und geben ihn am Ausgang als neuen Zustand z aus
- diese neuen Werte von z laufen durch das δ-Schaltnetz, dabei ist der schnellste Pfad $\tau_{\delta_{min}}$ und der langsamste $\tau_{\delta_{max}}$

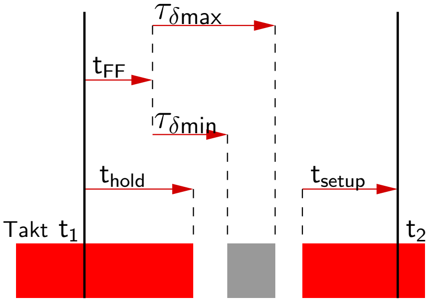
$\Rightarrow$ der Folgezustand ändert sich innerhalb des grau markierten Zeitintervalls:
$[(t_{FF} + \tau_{\delta_{min}}) \dots (t_{FF} + \tau_{\delta_{max}})]$



Zeitbedingungen: interner Zustand (cont.)

- Änderungen am FF-Eingang z^+ dürfen frühestens zum Zeitpunkt $(\mathrm{t}_1 + \mathrm{t}_{\mathrm{hold}})$ beginnen, sonst wird die Haltezeit t_{hold} verletzt

  dazu muss ggf. $\tau_{\delta_{\mathrm{min}}}$ vergrößert werden (zusätzliche Gatterverzögerungen)
- Änderungen am FF-Eingang z^+ müssen sich spätestens bis zum Zeitpunkt $(\mathrm{t}_2 - \mathrm{t}_{\mathrm{setup}})$ stabilisiert haben, wegen der Vorlaufzeit t_{setup} vor dem nächsten Takt



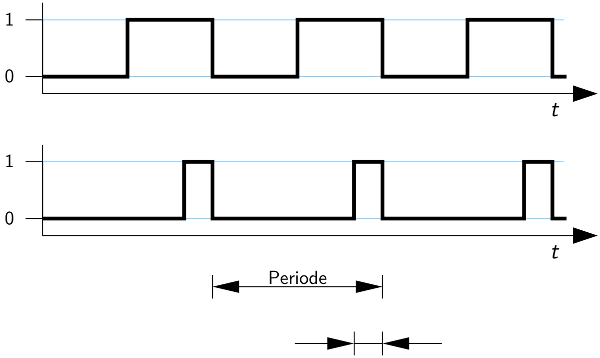
Maximale Taktfrequenz einer Schaltung

- aus den beiden vorigen Bedingungen ergibt sich sofort die maximal zulässige Taktfrequenz einer Schaltung
- Umformen und Auflösen nach dem Zeitpunkt des nächsten Takts ergibt zwei notwendige Zeitbedingungen

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

- falls dieses Timing verletzt wird (z.B. durch „Übertakten“), kann es (datenabhängig) zu Fehlfunktionen kommen

Taktsignal: Prinzip



- periodisches digitales Signal, Frequenz f bzw. Periode τ
- oft symmetrisch
- asymmetrisch für Zweiphasentakt (s.u.)

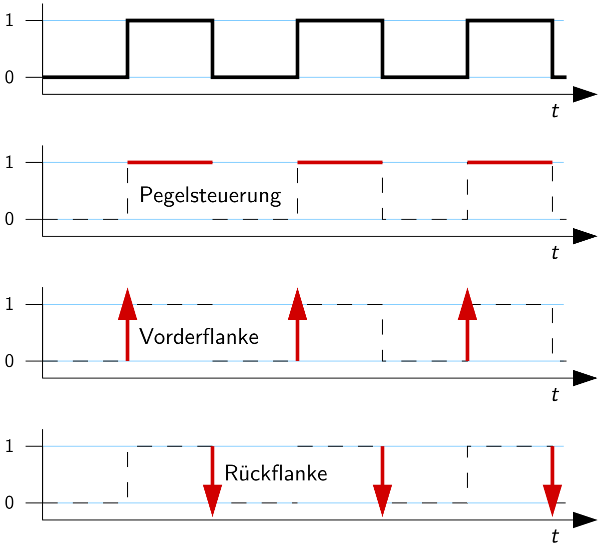
Taktsignal: Varianten

- **Pegelsteuerung**
  Schaltung reagiert auf Eingang, solange das Taktsignal den Wert 1 (bzw. 0) hat
  - high-aktiv: Takt $= 1$
  - low-aktiv: Takt $= 0$

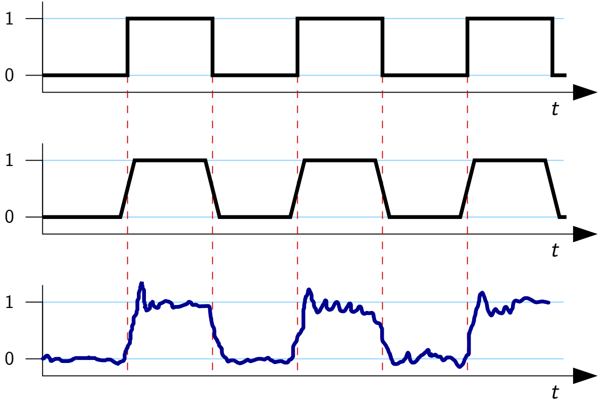
- **Flankensteuerung**
  Schaltung reagiert nur, wenn das Taktsignal den Wert wechselt
  - Vorderflankensteuerung: Wechsel von 0 nach 1
  - Rückflankensteuerung: –"– von 1 nach 0

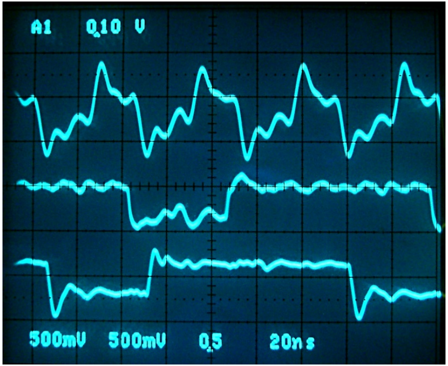
- Zwei- und Mehrphasentakte

Taktsignal: Varianten (cont.)



Taktsignal: Prinzip und Realität

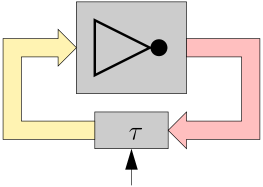




- Werteverläufe in realen Schaltungen stark gestört
- Überschwingen/Übersprechen benachbarter Signale
- Flankensteilheit nicht garantiert (lastabhängig)
  ggf. besondere Gatter („Schmitt-Trigger“)

Problem bei Pegelsteuerung

- während des aktiven Taktpegels werden Eingangswerte minimal verzögert an den Ausgang weiter gegeben
- durch Invertierungen in den Rückkopplungspfaden von δ, kommt es zu instabilen Zuständen (Oszillationen: $0 \rightarrow 1 \rightarrow 0 \rightarrow \ldots$)



- einzelne pegelgesteuerte Zeitglieder (D-Latches) funktionieren nicht in Rückkopplungspfaden

$\Rightarrow$ Verwendung von je zwei pegelgesteuerten Zeitgliedern mit Zweiphasentakt oder

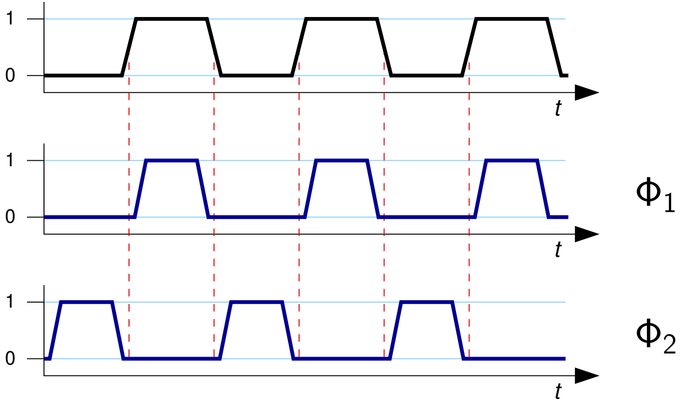
$\Rightarrow$ Verwendung flankengesteuerter D-Flipflops

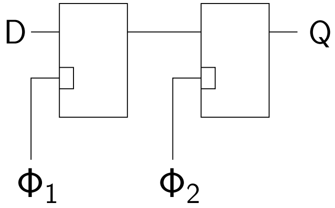
Zweiphasentakt

- 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, disjunkte Takte Φ_1 und Φ_2
  - bei Takt Φ_1 übernimmt vorderes Flipflop den Wert
    erst bei Takt Φ_2 übernimmt hinteres Flipflop
  - vergleichbar Master-Slave Prinzip bei D-FF aus Latches

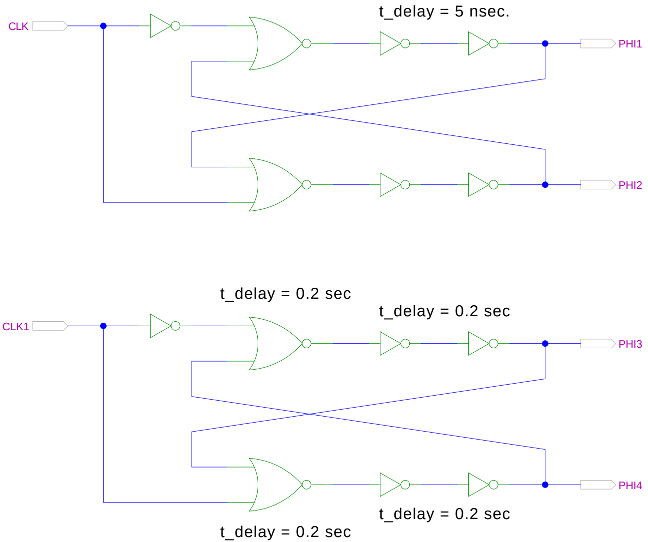
Zweiphasentakt (cont.)





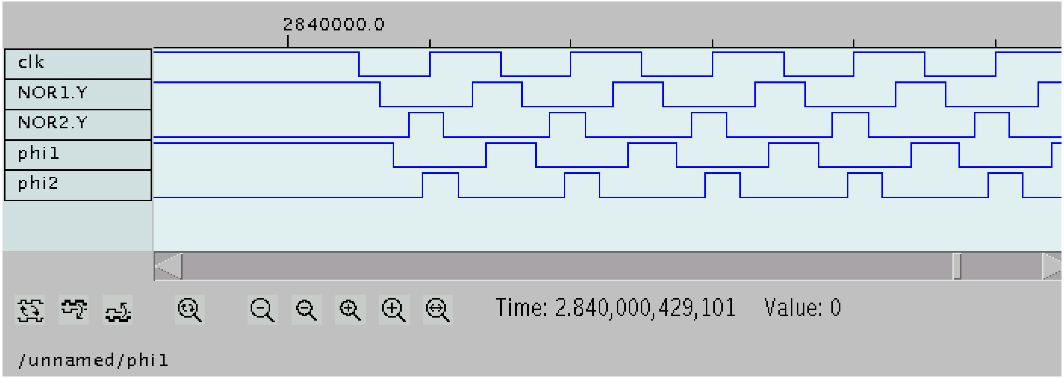
- nicht überlappender Takt mit Phasen Φ_1 und Φ_2
- vorderes D-Latch übernimmt Eingangswert *D* während Φ_1
  bei Φ_2 übernimmt das hintere D-Latch und liefert *Q*

Zweiphasentakt: Erzeugung



[Hen] Hades Demo: `12-gatedelay/40-tpcg/two-phase-clock-gen`

Zweiphasentakt: Erzeugung (cont.)



- Verzögerungen geeignet wählen
- Eins-Phasen der beiden Takte Φ_1 und Φ_2 sauber getrennt

$\Rightarrow$ 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

- **Flusstafel**
  Tabelle für die Folgezustände als Funktion des aktuellen Zustands und der Eingabe

= beschreibt das δ-Schaltnetz

- **Ausgangstafel**
  Tabelle für die Ausgabewerte als Funktion des aktuellen Zustands
  (und der Eingabe [Mealy-Modell])

= beschreibt das λ-Schaltnetz

- entsprechen Funktionstabellen von Schaltnetzen
- meistens in einer gemeinsamen Tabelle zusammengefasst

Beispiel: Ampel

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

- Flusstafel

Zustand	Codierung z_1	z_0	Folgezustand 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

- Ausgangstafel

Zustand	Codierung z_1	z_0	Ausgänge 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

- Funktionstabellen für 2+3 Schaltfunktionen
- Minimierung z.B. mit KV-Diagrammen

Zustandsdiagramm

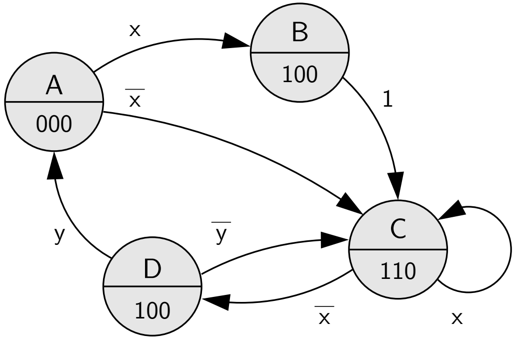
- **Zustandsdiagramm**: Grafische Darstellung eines Schaltwerks
- je ein Knoten für jeden Zustand
- je eine Kante für jeden möglichen Übergang

- 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 von Zustand (Knoten) und Input ab, sie wird deshalb an den Kanten notiert

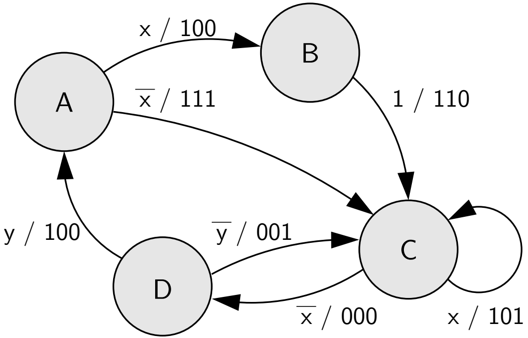
siehe auch en.wikipedia.org/wiki/State_diagram

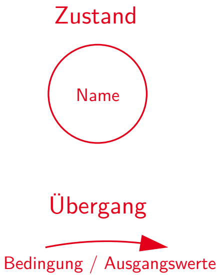
Zustandsdiagramm: Moore-Automat



- Ausgangswerte hängen ausschließlich vom Zustand ab
- können deshalb 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





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

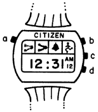
„State-Charts“

- erweiterte Zustandsdiagramme

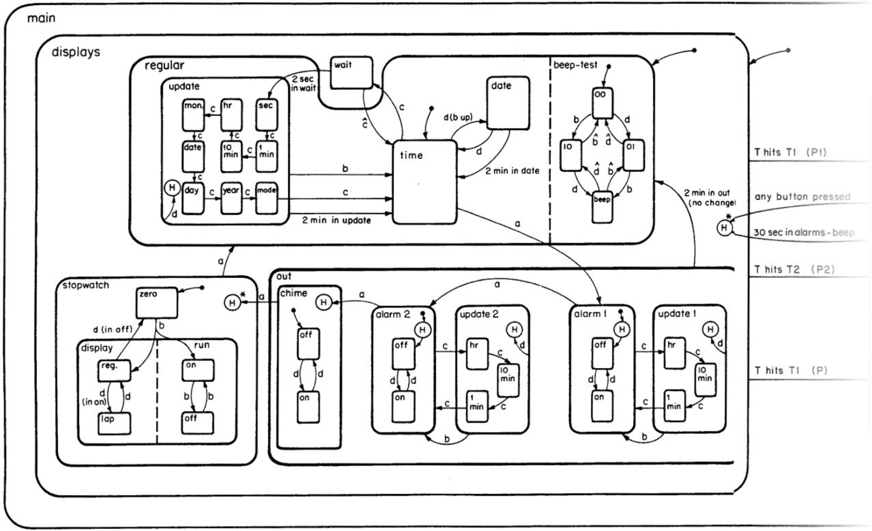
1. Hierarchien von Zuständen/Automaten erlauben Abstraktion
   - Knoten repräsentieren entweder einen Zustand
   - oder einen eigenen (Unter-) Automaten
   - *History*-, *Default*-Mechanismen
2. Nebenläufigkeit durch mehrere parallel arbeitende FSMs
3. Timer: Zustände können nach max. Zeit verlassen werden

- 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] www.weizmann.ac.il/math/harel/sites/math.harel/files/users/user50/Statecharts.pdf

- Beispiel



Citizen quartz multi-alarm



Endliche Automaten

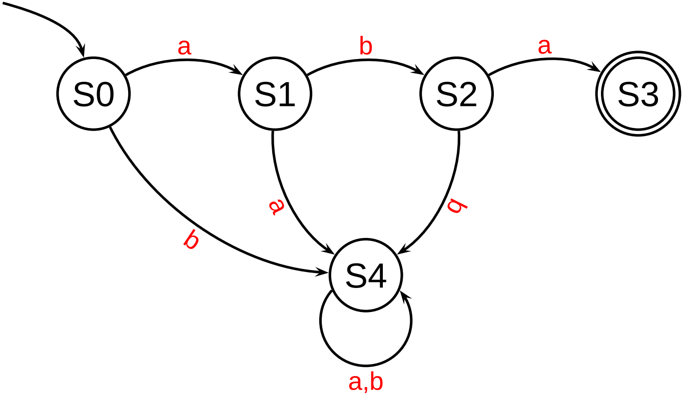
- eines der grundlegenden Konzepte der Informatik
- Modellierung, Entwurf und Simulation
  - zeitliche Abfolgen interner Systemzustände
  - bedingte Zustandswechsel
  - Reaktionen des Systems auf „Ereignisse“
  - Folgen von Aktionen
  - ...
- weitere „*spezielle*“ Anwendungsszenarien
  - verteilte Systeme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

zahlreiche Beispiele

Endliche Automaten (cont.)

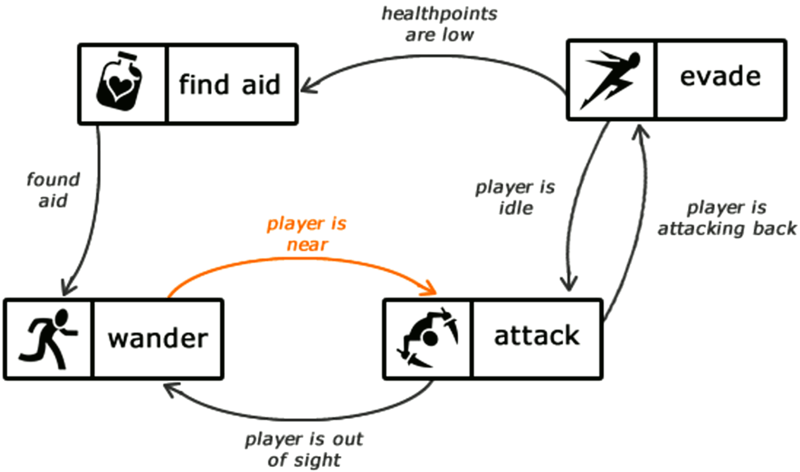
- in der Programmierung ...

  Erkennung des Worts: „a b a“



Endliche Automaten (cont.)

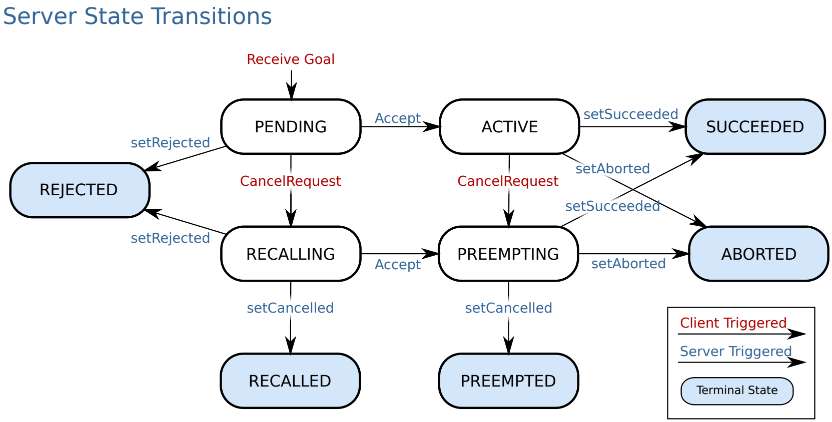
Game-Design: Verhalten eines Bots



code.tutsplus.com/finite-state-machines-theory-and-implementation--gamedev-11867t

Endliche Automaten (cont.)

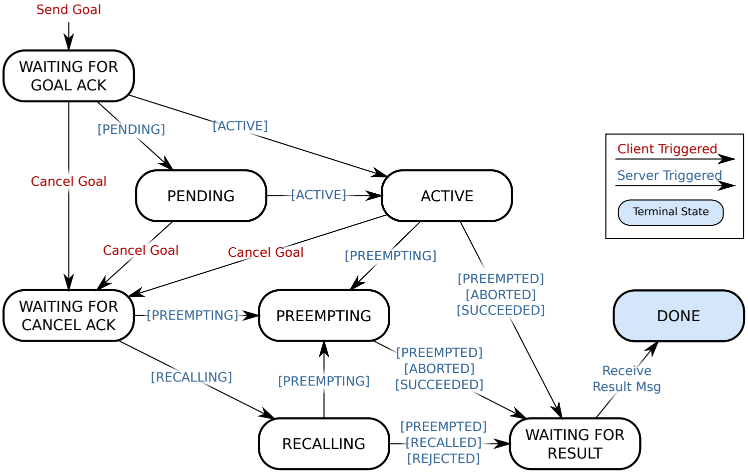
- Beschreibung von Protokollen
- Verhalten verteilter Systeme: Client-Server Architektur



wiki.ros.org/actionlib/DetailedDescription

Endliche Automaten (cont.)

Client State Transitions

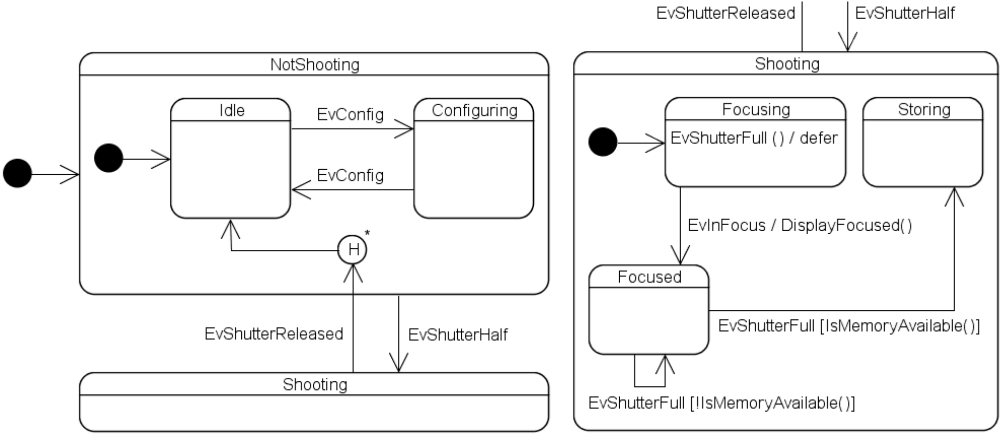


wiki.ros.org/actionlib/DetailedDescription

Endliche Automaten (cont.)

- Unterstützung durch Bibliotheken und Werkzeuge

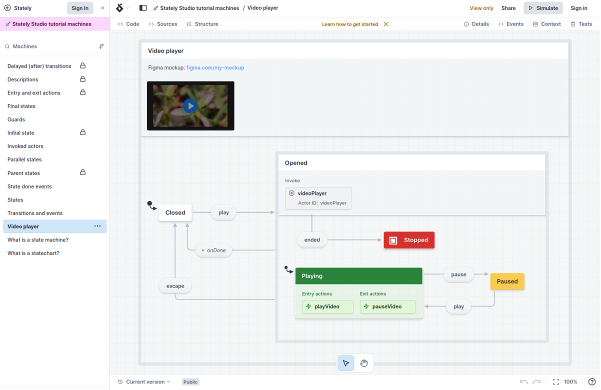
  State-Chart Bibliothek: Beispiel Digitalkamera



`www.boost.org/doc/libs/1_86_0/libs/statechart/doc`

Endliche Automaten (cont.)

Java Script und TypeScript



`xstate.js.org`, `stately.ai`, `github.com/statelyai/xstate`

⇒ beliebig viele weitere Beispiele . . .

„Endliche Automaten“ werden in RSB nur hardwarenah genutzt

Hardwarebeschreibungssprachen

- Beschreibung eines Schaltwerks als Programm
  - normale Hochsprachen C, Java
  - spezielle Bibliotheken für normale Sprachen SystemC, Hades
  - Hardwarebeschreibungssprachen Verilog, VHDL

- Eigenschaften von Hardwarebeschreibungssprachen
  - Abstraktion und Hierarchie
  - Modellierung paralleler Abläufe
  - detailliertes Zeitverhalten von Schaltungen / Leitungen
  - Systembeschreibung von Hardware und Software

- Vergleichbar mit parallelen Programmiersprachen, z.B.: Ada

- hier nicht weiter vertieft. . .
  zwei Beispiele: D-Flipflop in Verilog und VHDL

D-Flipflop in Verilog


```
module dff (clock, reset, din, dout);  // Black-Box Beschreibung
input clock, reset, din;               // Ein- und Ausgänge
output dout;                           //

reg dout;                              // speicherndes Verhalten

  always @(posedge clock or reset)     // Trigger für Code
  begin                                //
    if (reset)                         // async. Reset
      dout = 1'b0;                     //
    else                               // implizite Taktvorderflanke
      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                              -- Black-Box Beschreibung
port (  clock    : in  std_logic;          -- Ein- und Ausgänge
        reset    : in  std_logic;          --
        din      : in  std_logic;          --
        dout     : out std_logic);         --
end entity dff;

architecture behavior of dff is            -- Verhaltensmodell
begin                                      --
  dff_p: process (reset, clock) is         -- Trigger für Prozess
  begin                                    --
    if reset = '1' then                    -- async. Reset
        dout <= '0';                       --
    elsif rising_edge(clock) then          -- Taktvorderflanke
        dout <= din;                       --
    end if;                                --
  end process dff_p;                       --
end architecture behavior;
```

Entwurf von Schaltwerken

Direkte Realisierung durch Schaltnetze

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

Mikroprogrammierung siehe Abschnitt: **??** *Mikroprogrammierung*, ab Folie **??**

- bei großen Schaltwerken
- Befehlsabarbeitung in Prozessoren

Entwurf von Schaltwerken: Zustandscodierung

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
  $\Rightarrow$ Widerspruch
- keine Übergänge bei bestimmten Eingangsbelegungen
  $\Rightarrow$ 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?

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

Überprüfung der Widerspruchsfreiheit

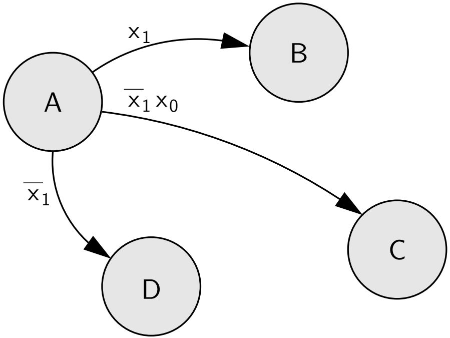
p Zustände, Zustandsdiagramm mit Kanten $h_{ij}(x)$:
Übergang von Zustand i nach Zustand j unter Belegung x

- für jeden Zustand überprüfen:

  kommen alle (spezifizierten) Eingangsbelegungen nur einmal vor?

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

Vollständigkeit und Widerspruchsfreiheit: Beispiel



- für Zustand A

Vollständigkeit	$x_1 \vee \overline{x_1}\,x_0 \vee \overline{x_1} = 1$	ok
Widerspruchsfreiheit	$x_1 \wedge \overline{x_1}\,x_0 = 0$	ok
alle Paare testen	$x_1 \wedge \overline{x_1} = 0$	ok
	$\overline{x_1}\,x_0 \wedge \overline{x_1} \neq 0$	zwei Übergänge aktiv! $x_1 = 0,\ x_0 = 1$

Schaltwerke: Beispiele

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

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 Zustandscodierung
- Aufstellen der Tafeln für δ- und λ-Schaltnetz
- anschließend Minimierung der Schaltnetze
- Realisierung (je 1 D-Flipflop pro Zustandsbit) und Test

Schaltwerksentwurf: Ampel – Variante 1

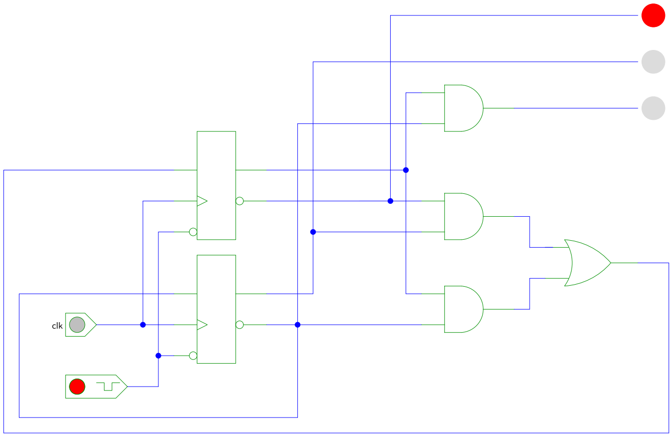
- vier Zustände, Codierung als 2-bit Vektor (z_1, z_0)
- Fluss- und Ausgangstafel für binäre Zustandscodierung

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

$$
\begin{aligned}
z_1^+ &= (z_1 \wedge \overline{z_0}) \vee (\overline{z_1} \wedge z_0) = z_1 \oplus z_0 \\
z_0^+ &= \overline{z_0} \\
rt &= \overline{z_1} \\
ge &= z_0 \\
gr &= (z_1 \wedge \overline{z_0})
\end{aligned}
$$

Schaltwerksentwurf: Ampel – Variante 1 (cont.)



[Hen] Hades Demo: `18-fsm/10-trafficlight/ampel_41`

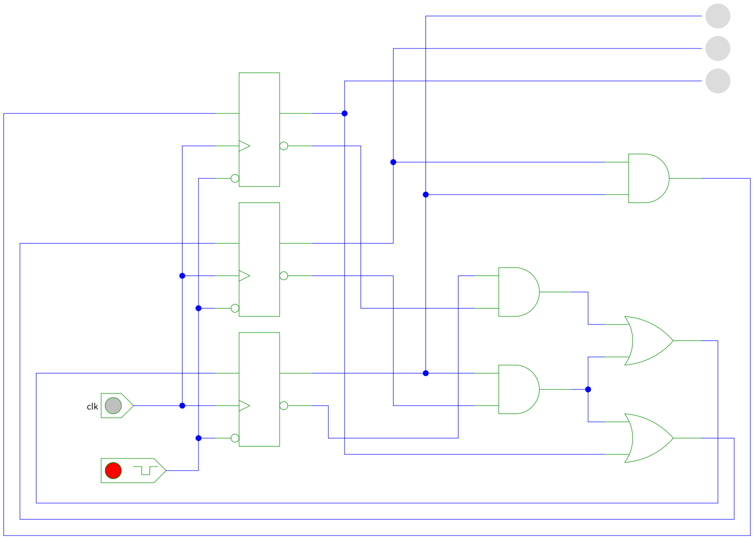
Schaltwerksentwurf: Ampel – Variante 2

- 4+1 Zustände, Codierung als 3-bit Vektor (z_2, z_1, z_0)
  Reset-Zustand: alle Lampen aus
- Zustandsbits korrespondieren mit 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
- Ausgangsfunktion λ minimal: entfällt
- Übergangsfunktion δ: $z_2^+ = (z_1 \wedge z_0)$ $\quad z_1^+ = z_2 \vee (\overline{z_1} \wedge z_0)$
  $z_0^+ = (\overline{z_2} \wedge \overline{z_0}) \vee (\overline{z_1} \wedge z_0)$

Schaltwerksentwurf: Ampel – Variante 2 (cont.)



[Hen] Hades Demo: `18-fsm/10-trafficlight/ampel_42`

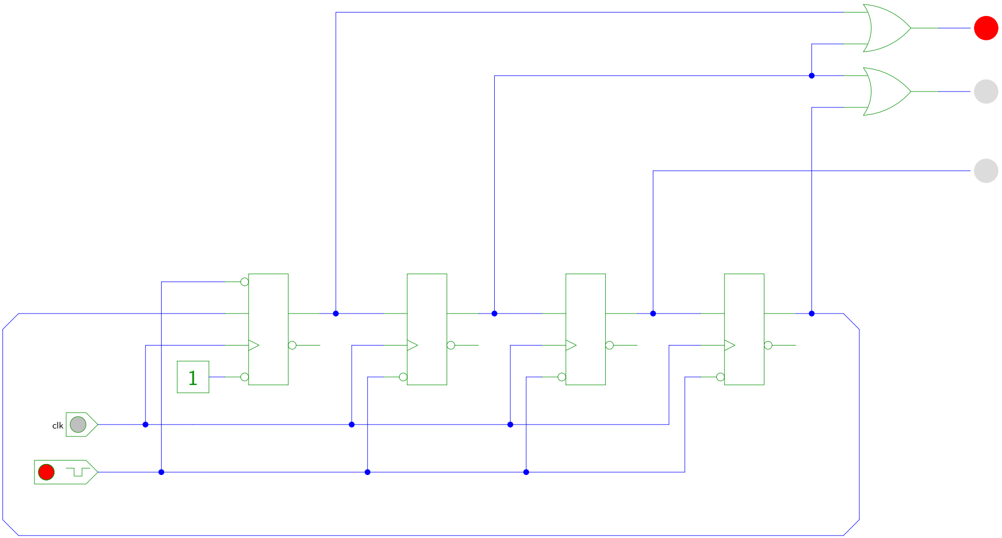
Schaltwerksentwurf: Ampel – Variante 3

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

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 δ minimal: Rotate-Left um 1
  $\Rightarrow$ Automat sehr schnell, hohe Taktrate möglich
- Ausgangsfunktion λ sehr einfach:
  $gr = z_2 \qquad ge = z_3 \vee z_1 \qquad rt = z_1 \vee z_0$

Schaltwerksentwurf: Ampel – Variante 3 (cont.)



[Hen] Hades Demo: `18-fsm/10-trafficlight/ampel_44`

Schaltwerksentwurf: Ampel – Zusammenfassung

- viele Möglichkeiten der Zustandscodierung

- Dualcode: minimale Anzahl der Zustände
- applikations-spezifische Codierungen
- One-Hot Encoding: viele Zustände, einfache Schaltnetze
- ...

- Kosten/Performanz des Schaltwerks abhängig von Codierung
- Heuristiken zur Suche nach (relativem) Optimum

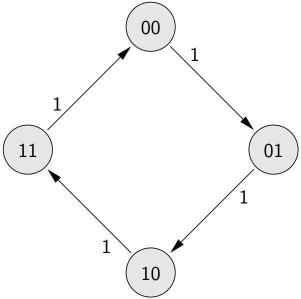
Zählschaltungen

- diverse Beispiele für Zählschaltungen

- Zustandsdiagramme und Flusstafeln
- Schaltbilder

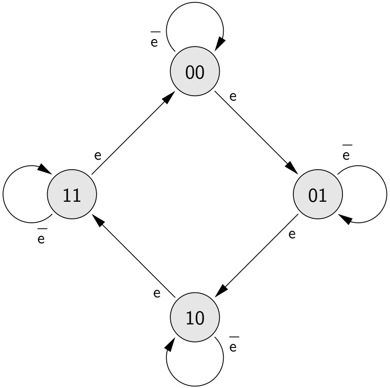
- *n*-bit Vorwärtszähler
- *n*-bit Zähler mit Stop und/oder Reset
- Vorwärts-/Rückwärtszähler
- synchrone und asynchrone Zähler
- Beispiel: Digitaluhr (BCD Zähler)

2-bit Zähler: Zustandsdiagramm



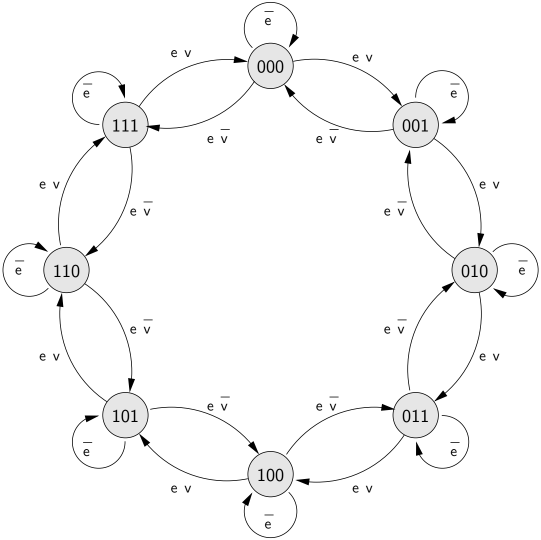
- Zähler als „trivialer“ endlicher Automat

2-bit Zähler mit Enable: Zustandsdiagramm, Flusstafel



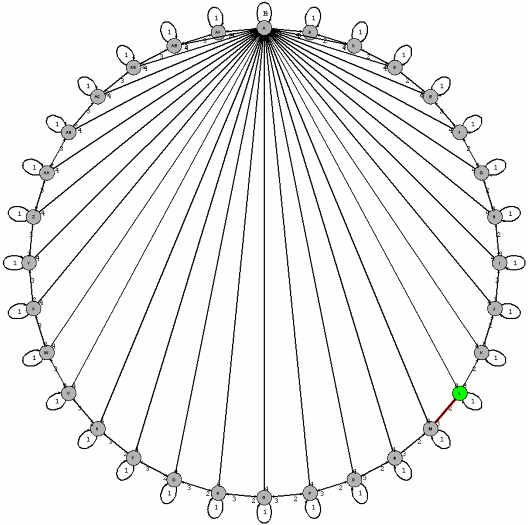
Eingabe	e	$\overline{e}$
Zustand	Folgezustand	
00	01	00
01	10	01
10	11	10
11	00	11

3-bit Zähler mit Enable, Vor-/Rückwärts



Eingabe	$e\ v$	$e\ \overline{v}$	$\overline{e}\ *$
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

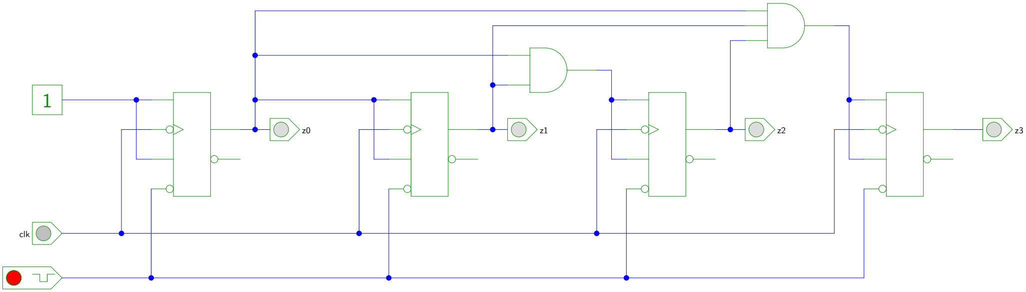
5-bit Zähler mit Reset: Zustandsdiagramm und Flusstafel



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

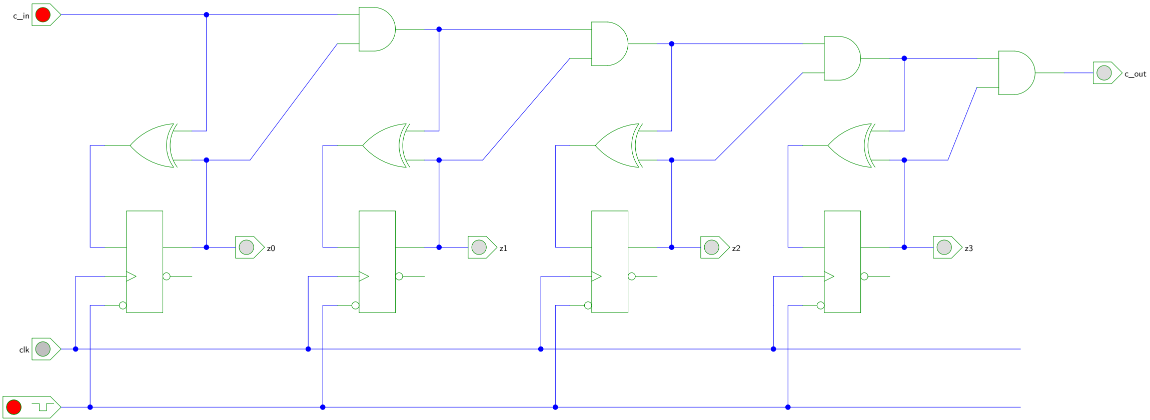
4-bit Binärzähler mit JK-Flipflops



[Hen] Hades Demo: `30-counters/30-sync/sync`

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

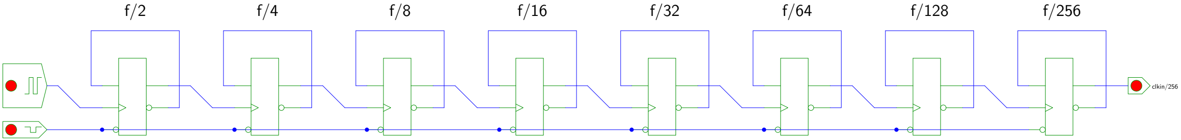
4-bit Binärzähler mit D-Flipflops (kaskadierbar)



[Hen] Hades Demo: `30-counters/30-sync/sync-dff`

- $D_0 = z_0 \oplus c_{in}$ wechselt bei Takt, wenn Carry-in $c_{in} = 1$
- $D_i = z_i \oplus (c_{in} z_0 z_1 \ldots z_{i-1})$ wechselt, wenn alle niedrigeren Stufen und $c_{in} = 1$

Asynchroner n-bit Zähler/Teiler mit D-Flipflops



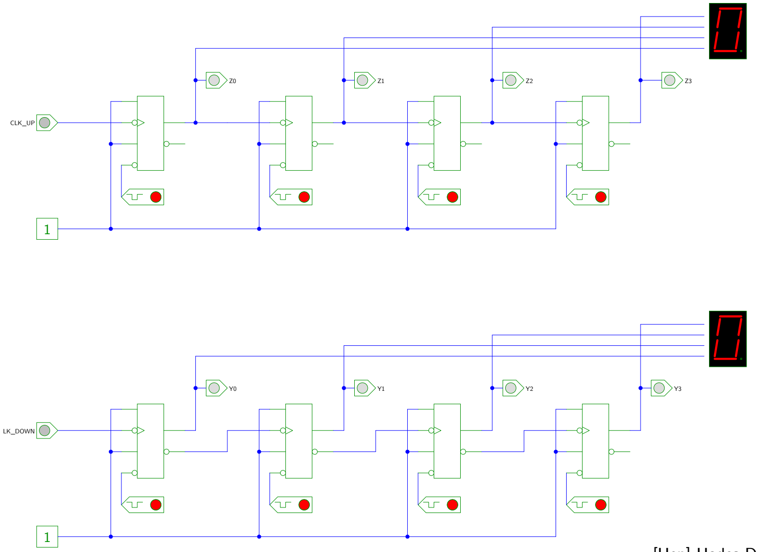
[Hen] Hades Demo: `30-counters/20-async/counter-dff`

- $D_i = \overline{z_i}$: jedes Flipflop wechselt bei seinem Taktimpuls
- Takteingang C_0 treibt nur das vorderste Flipflop
- $C_i = z_{i-1}$: 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



[Hen] Hades Demo: `30-counters/20-async/counter`

Literatur

[Har87] David Harel.
Statecharts: A visual formalism for complex systems.
Sci. Comput. Program., 8(3):231–274, June 1987.

[Hen] Norman Hendrich.
Hades — hamburg design system.
Lehrmaterial, Universität Hamburg, Fachbereich Informatik, AB TAMS.

[Rei98] Norbert Reifschneider.
CAE-gestützte IC-Entwurfsmethoden.
Prentice Hall, München, 1998.

[SS04] Wolfram Schiffmann and Robert Schmitz.
Technische Informatik 1 – Grundlagen der digitalen Elektronik.
Springer-Verlag GmbH, Berlin, fifth edition, 2004.

[vdH05] Klaus von der Heide.
Vorlesung: Technische informatik 1 — interaktives skript.