# 64-040 Modul IP7: Rechnerstrukturen

## 8. Zeitverhalten und Schaltwerke

Norman Hendrich

Universität Hamburg
MIN Fakultät, Department Informatik
Vogt-Kölln-Str. 30, D-22527 Hamburg
hendrich@informatik.uni-hamburg.de

WS 2013/2014

# Inhalt

# Inhalt (cont.)

Literatur

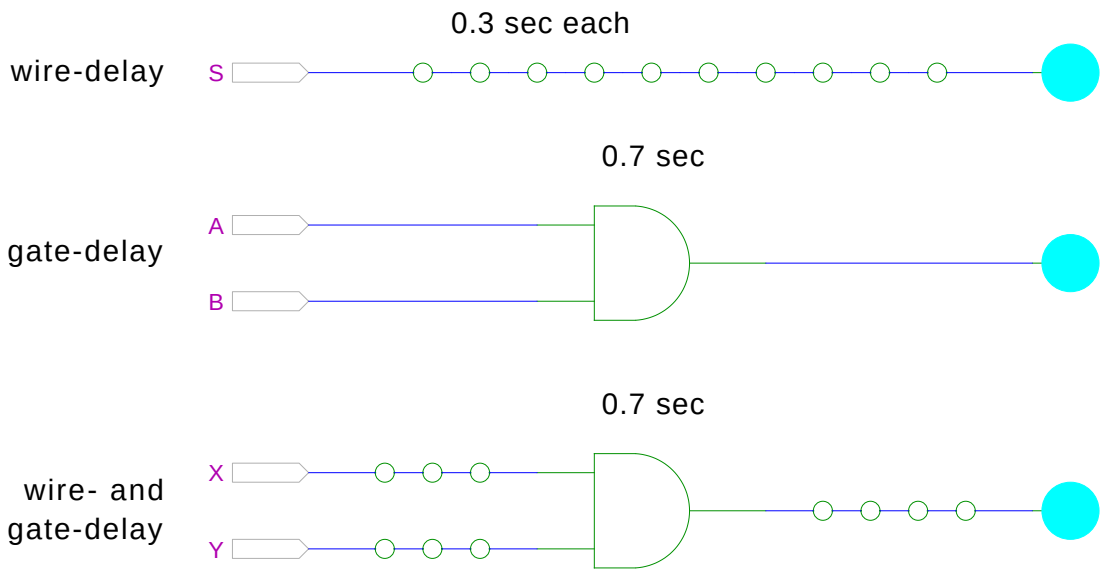
# Zeitverhalten einer Schaltung: Modellierung

- Modellierung des Zeitverhaltens eines Schaltnetzes?
- diverse gängige Abstraktionsebenen:

1. algebraische Ausdrücke: keine zeitliche Abhängigkeit
2. „fundamentales Modell“: Verzögerung des algebraischen Ausdrucks um eine Zeit $\tau$
3. individuelle Gatterverzögerungen
4. individuelle Leitungslaufzeiten
5. Differentialgleichungen für Spannungen und Ströme

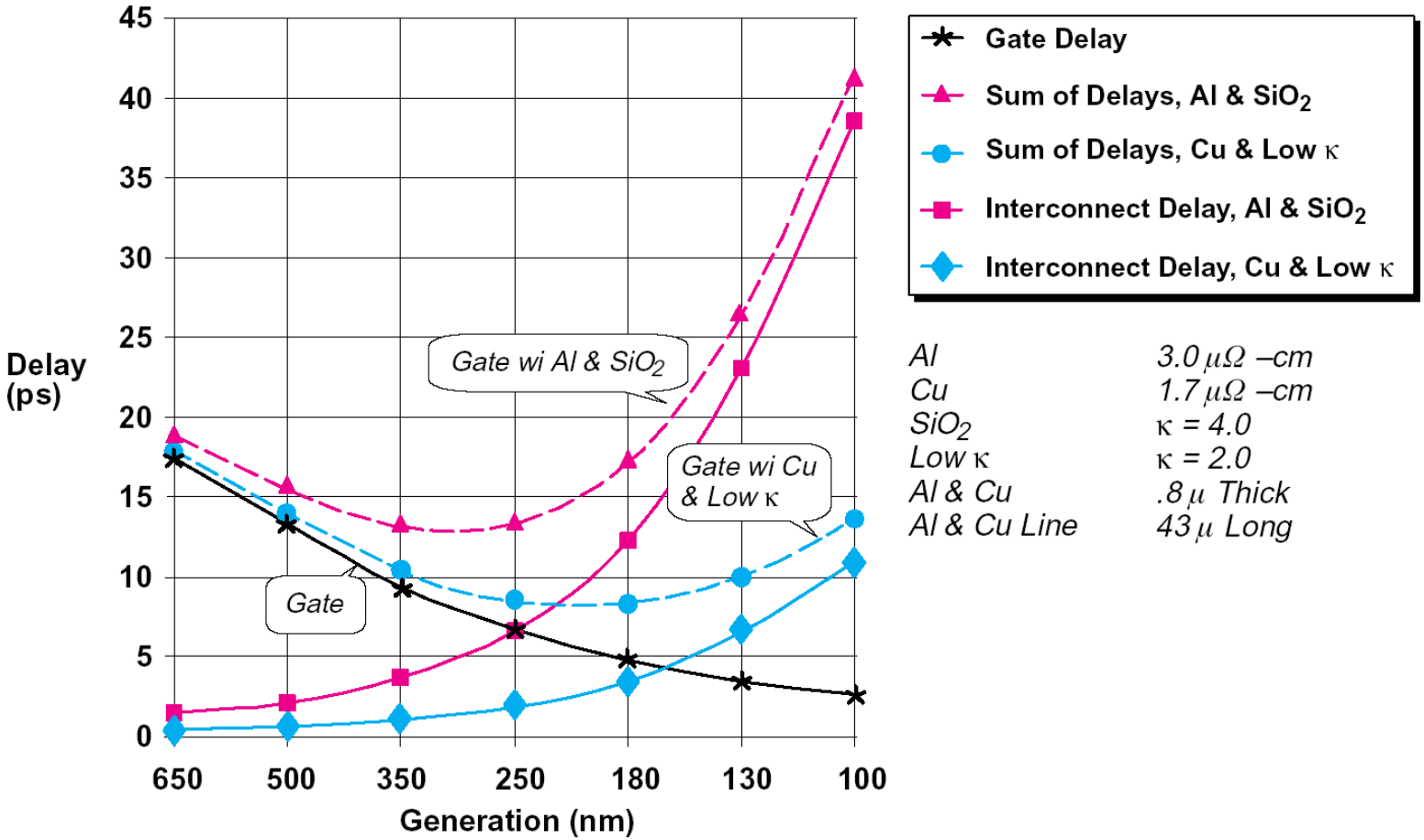
...

# Gatterverzögerung vs. Leitungslaufzeiten




- früher: Gatterverzögerungen $\gg$ Leitungslaufzeiten
- Schaltungen modelliert durch Gatterlaufzeiten
- aktuelle „Submicron“-Halbleitertechnologie: Leitungslaufzeiten $\gg$ Gatterverzögerungen

# Gatterverzögerung vs. Leitungslaufzeiten




| | |
|---|---|
| *Al* | *3.0 μΩ –cm* |
| *Cu* | *1.7 μΩ –cm* |
| *$SiO_2$* | *κ = 4.0* |
| *Low κ* | *κ = 2.0* |
| *Al & Cu* | *.8 μ Thick* |
| *Al & Cu Line* | *43 μ Long* |

SIA'97 http://www.sia-online.org/

# Zeitverhalten

- alle folgenden Schaltungsbeispiele werden mit Gatterverzögerungen modelliert
- Gatterlaufzeiten als Vielfache einer Grundverzögerung ($\tau$)
- aber Leitungslaufzeiten ignoriert

- mögliche Verfeinerungen:
  - gatterabhängige Schaltzeiten für INV, NAND, NOR, XOR, usw.
  - unterschiedliche Schaltzeiten für Wechsel $0 \rightarrow 1$ und $1 \rightarrow 0$
  - unterschiedliche Schaltzeiten für 2-, 3-, 4-Input Gatter
  - Schaltzeiten abhängig von der Belastung eines Gatters, d.h. der vom Gatter angesteuerten Eingänge („fanout“)

# Exkurs: Lichtgeschwindigkeit und Taktraten

- Lichtgeschwindigkeit im Vakuum: $c \approx 300.000\,\text{km/sec}$
- im Vakuum: $c \approx 30\,\text{cm/ns}$
- in Metallen und Halbleitern langsamer: $c \approx 20\,\text{cm/ns}$
- bei 1 Gigahertz Takt: Ausbreitung um ca. 20 Zentimeter

Abschätzungen:

- Prozessor: ca. 2 cm Diagonale $\approx$ 10 GHz Taktrate
- Platine: ca. 20 cm Kantenlänge $\approx$ 1 GHz Takt
- UKW-Radio: 100 MHz, 2 Meter Wellenlänge

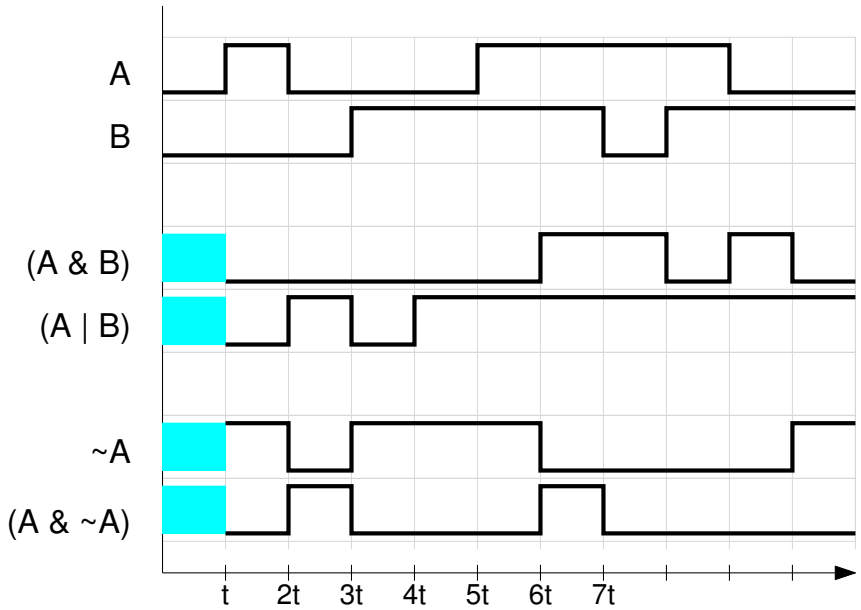
# Darstellung: Impulsdiagramme

- **Impulsdiagramm** (engl.: *waveform*): Darstellung der logischen Werte einer Schaltfunktion als Funktion der Zeit

- als Abstraktion des tatsächlichen Verlaufs
- Zeit läuft von links nach rechts
- Schaltfunktion(en): von oben nach unten

- vergleiche Messwerte am Oszilloskop (analoge Werte)
- bzw. Messwerte am Logic-State-Analyzer (digitale Werte)
- ggf. Darstellung mehrerer logischer Werte (z.B. 0,1,Z,U,X)

# Impulsdiagramm: Beispiel




- im Beispiel jeweils eine „Zeiteinheit“ $t$ Verzögerung für jede einzelne logische Operation (d.h., fundamentales Modell)
- Ergebnis einer Operation nur, wenn die Eingaben definiert sind
- in den ersten Zeitschritten ggf. noch undefinierte Werte

# Hazards

- **Hazard**: die Eigenschaft einer Schaltfunktion, bei bestimmten Kombinationen der individuellen Verzögerungen ihrer Verknüpfungsglieder ein Fehlverhalten zu zeigen

- **Hazardfehler**: das aktuelle Fehlverhalten einer realisierten Schaltfunktion aufgrund eines Hazards

# Hazards: Klassifikation

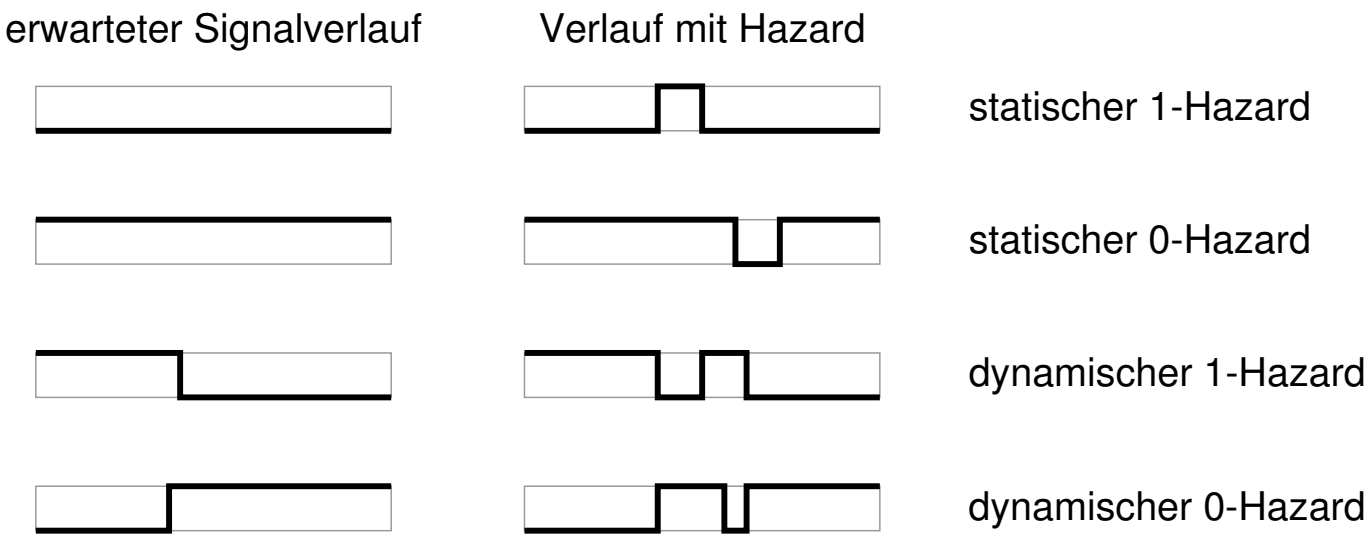
nach der Erscheinungsform am Ausgang:

- **statisch**: der Ausgangswert soll stabil sein, es tritt aber ein Wechsel auf
- **dynamisch**: der Ausgangswert soll (einmal) wechseln, es tritt aber ein mehrfacher Wechsel auf

nach den Eingangsbedingungen, unter denen der Hazard auftritt

- **Strukturhazard**: bedingt durch die Struktur der Schaltung, auch bei Umschalten eines einzigen Eingangswertes
- **Funktionshazard**: bedingt durch die Funktion der Schaltung

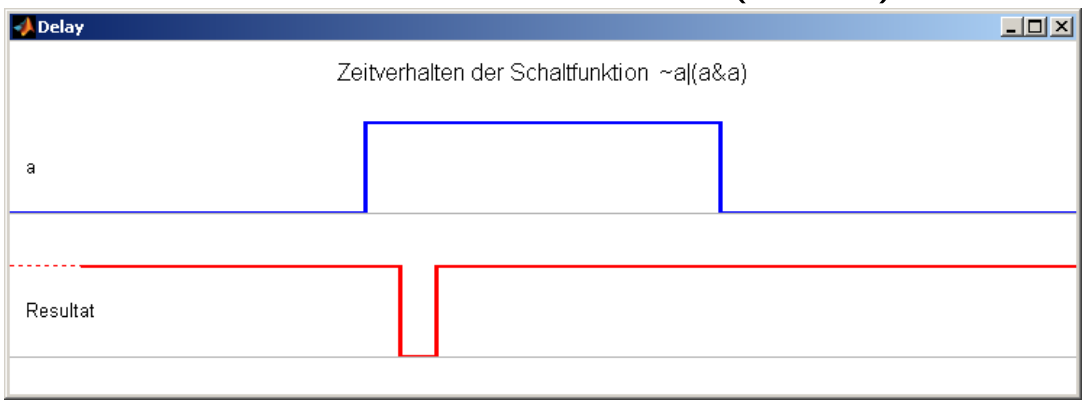
# Hazards: statisch vs. dynamisch




- 1-Hazard wenn fehlerhaft der Wert 1 auftritt, und umgekehrt
- es können natürlich auch mehrfache Hazards auftreten
- Hinweis: Begriffsbildung in der Literatur nicht einheitlich

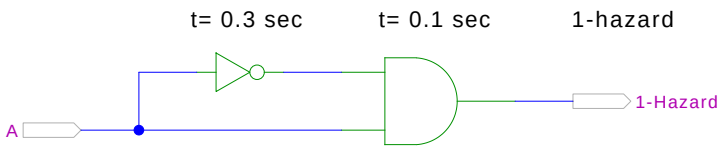
# Hazards: Strukturhazard

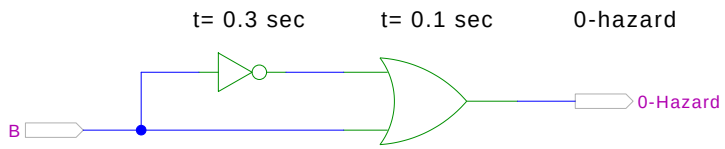
- **Strukturhazard**: ein Hazard, der durch die gewählte Struktur der Schaltung verursacht wird
- auch, wenn sich nur eine Variable ändert
- Beispiel: $f(a) = \neg a \vee (a \wedge a)$
  $\neg a$ schaltet schneller ab, als $(a \wedge a)$ einschaltet

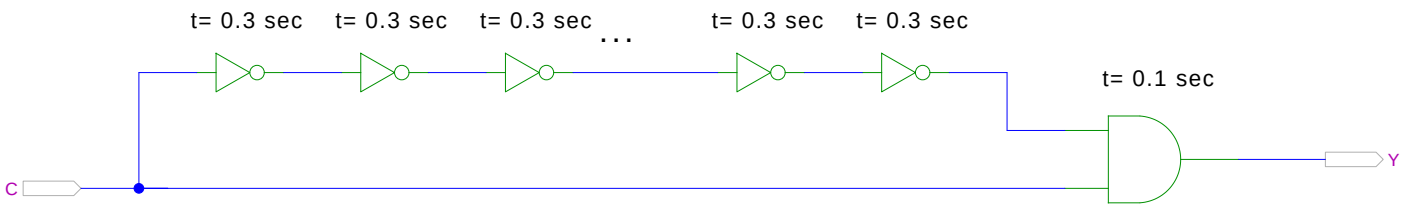



- kann durch Modifikation der Schaltung beseitigt werden
  im Beispiel mit: $f(a) = 1$

# Strukturhazards: Beispiele



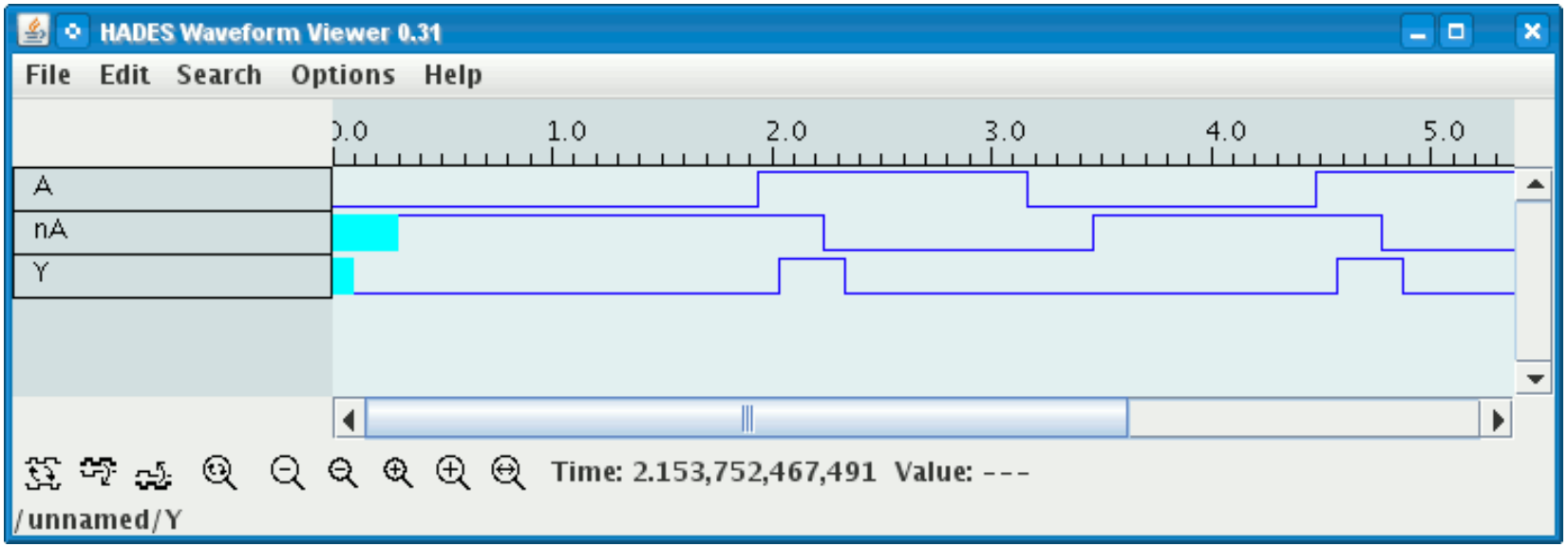




- logische Funktion ist $(a \wedge \overline{a}) = 0$ bzw. $(a \vee \overline{a}) = 1$
- aber ein Eingang jeweils durch Inverter verzögert

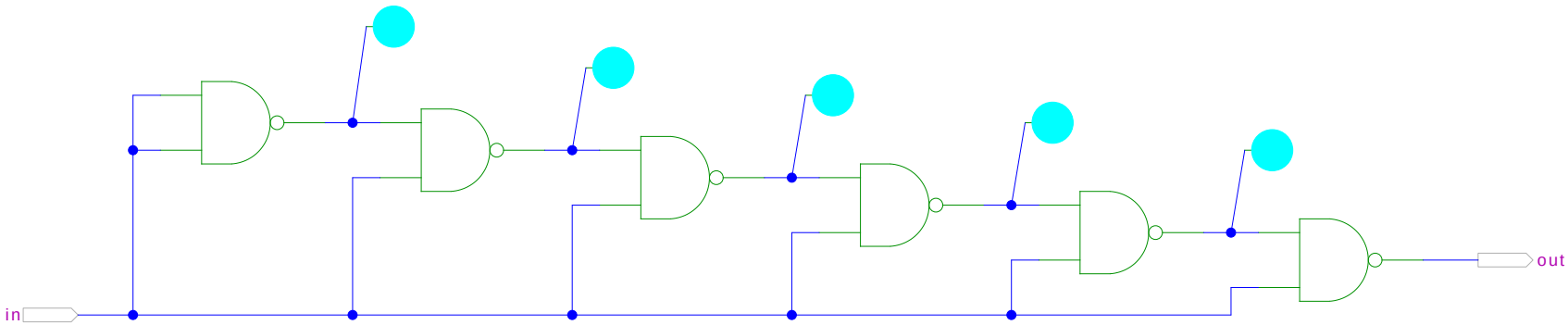
$\Rightarrow$ kurzer Impuls beim Umschalten von $0 \rightarrow 1$ bzw. $1 \rightarrow 0$

# Strukturhazards: Impulsdiagramm für das Beispiel



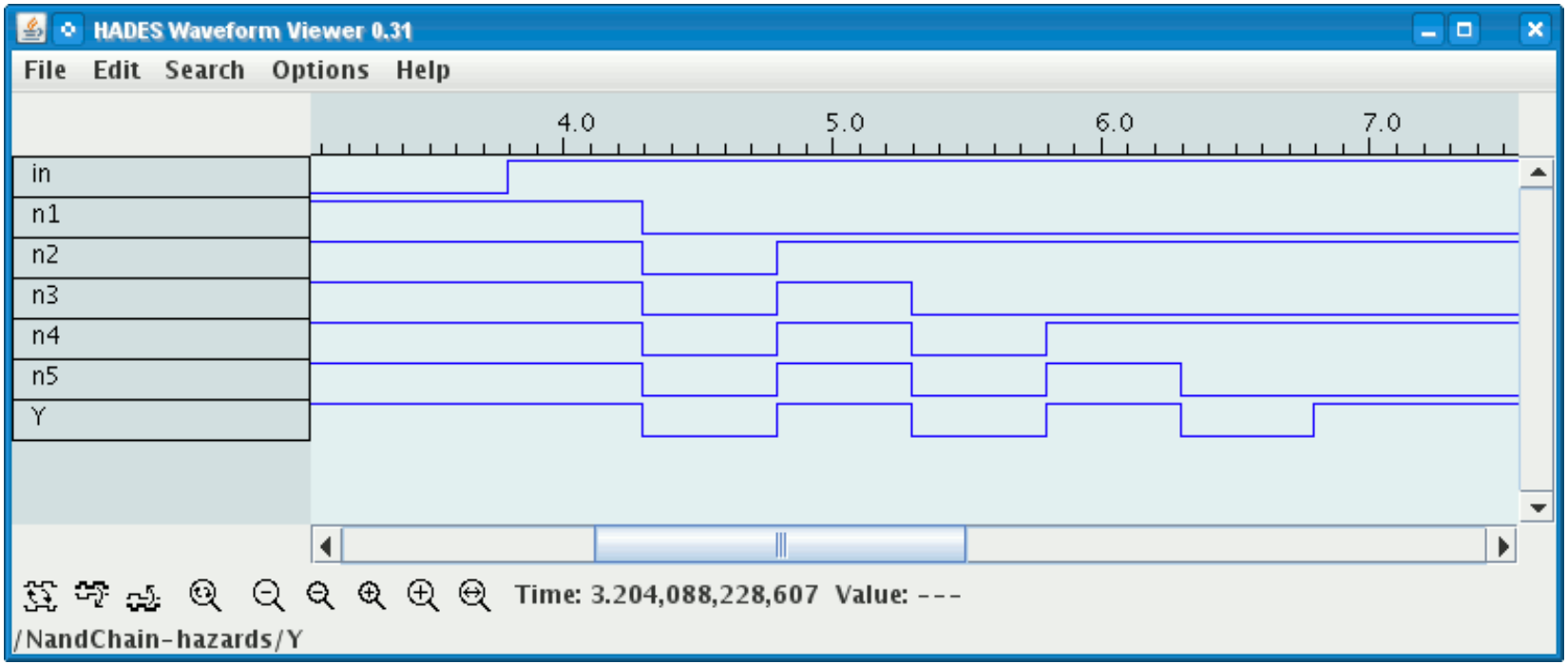

- Schaltung $(a \wedge \bar{a}) = 0$ erzeugt (statischen-1) Hazard
- Länge des Impulses abhängig von Verzögerung im Inverter
- Kette von Invertern erlaubt Einstellung der Pulslänge

# Strukturhazards extrem: NAND-Kette



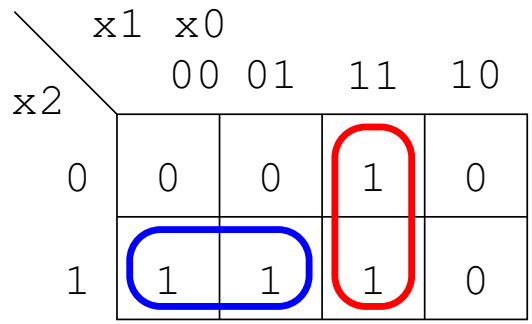

- alle NAND-Gatter an Eingang *in* angeschlossen
- $in = 0$ erzwingt $y_i = 1$
- Übergang *in* von 0 auf 1 startet Folge von Hazards...

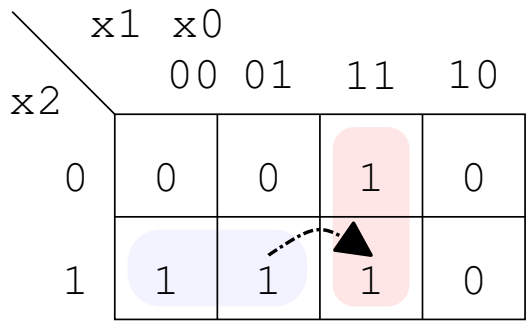
# NAND-Kette: Impulsdiagramm




- Schaltung erzeugt Folge von (dynamischen-0) Hazards
- Anzahl der Impulse abhängig von Anzahl der Gatter

# Strukturhazards: im KV-Diagramm (1)



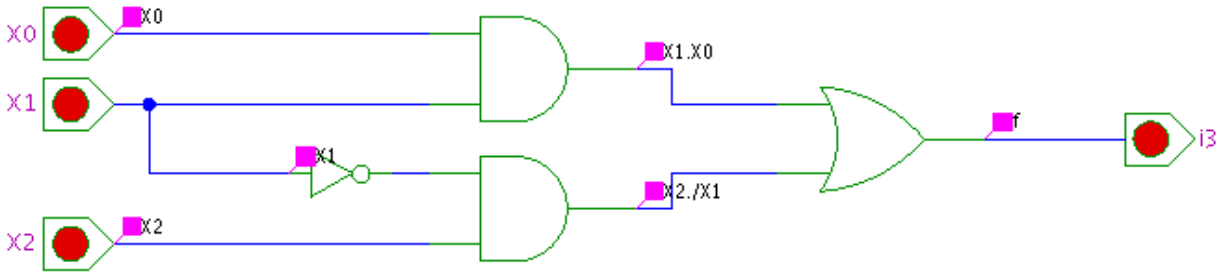



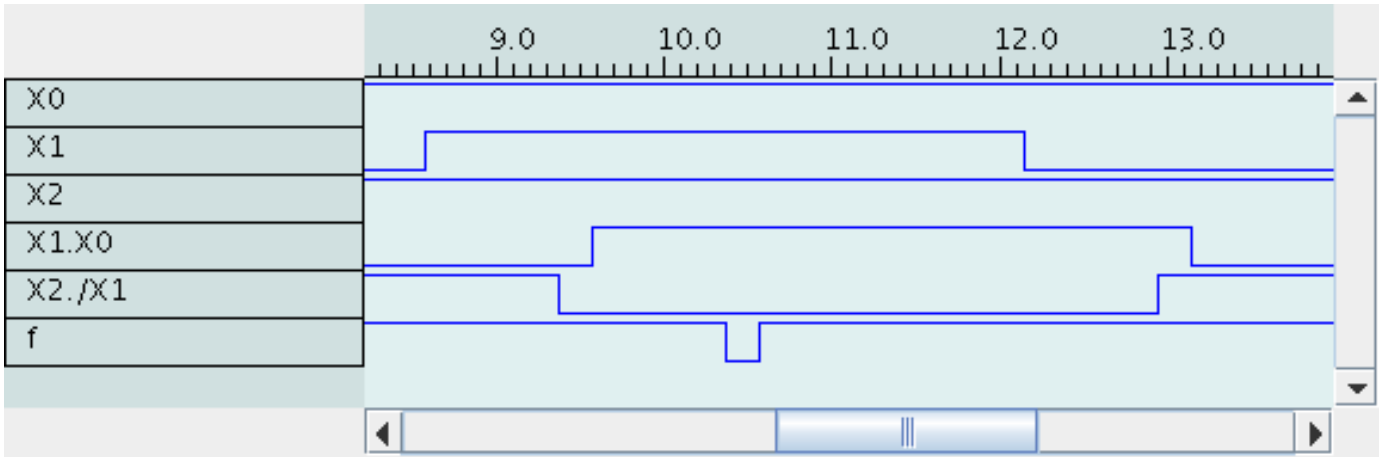

- Funktion $f = (x_2\overline{x}_1) \vee (x_1x_0)$
- links: realisiert in disjunktiver Form mit 2 Schleifen

Strukturhazard beim Übergang von $(x_2\overline{x}_1x_0)$ nach $(x_2x_1x_0)$:

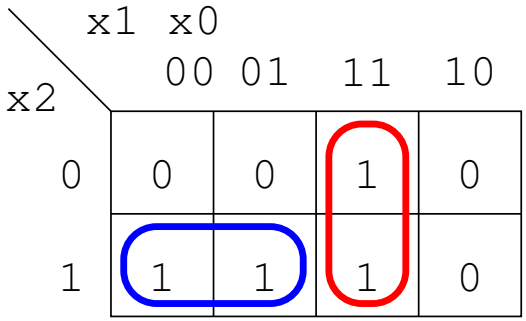
- Gatter $(x_2\overline{x}_1)$ schaltet ab, Gatter $(x_1x_0)$ schaltet ein
- Ausgang evtl. kurz 0, abhängig von Verzögerungen

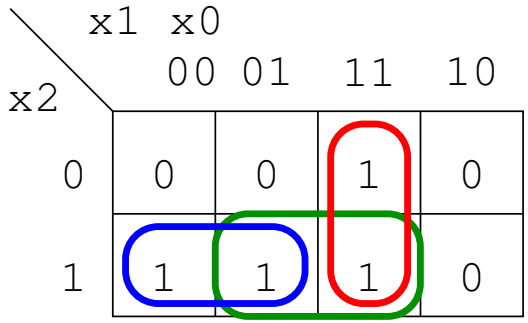
# Strukturhazards: Impulsdiagramm (1)



# Strukturhazards beseitigen

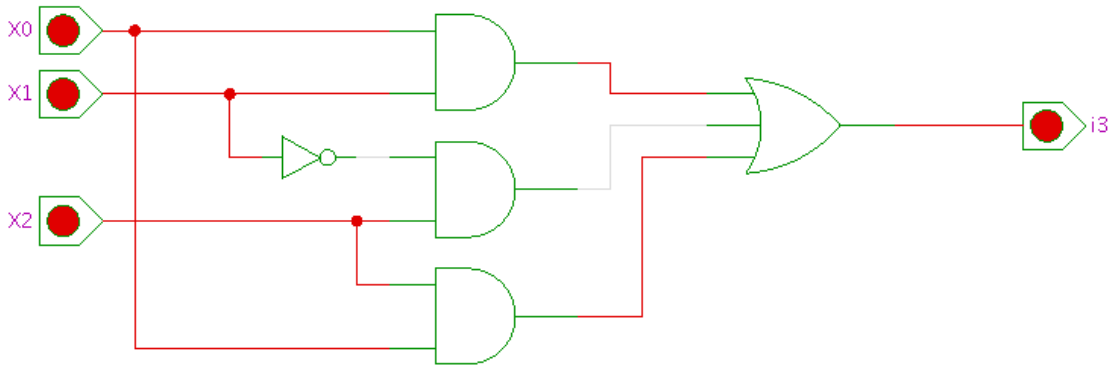


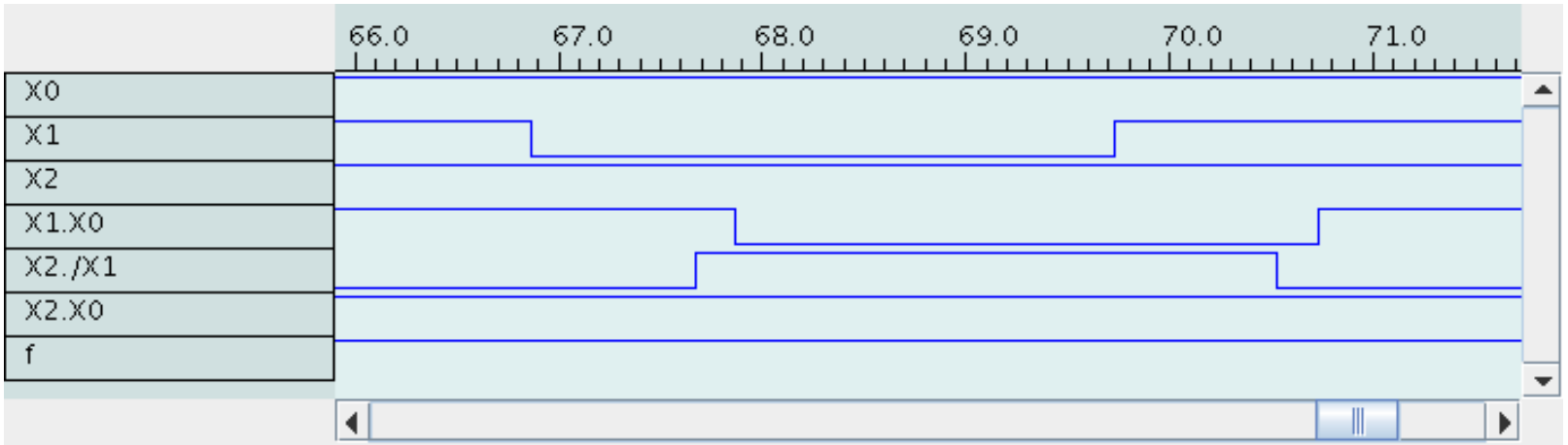




- Funktion $f = (x_2\overline{x}_1) \vee (x_1x_0)$
- rechts: realisiert in disjunktiver Form mit 3 Schleifen, also
  $f = (x_2\overline{x}_1) \vee (x_1x_0) \vee (x_2x_0)$

- Strukturhazard durch zusätzliches Gatter beseitigt
- aber höhere Hardwarekosten als bei minimierter Realisierung

# Strukturhazards: Impulsdiagramm (2)

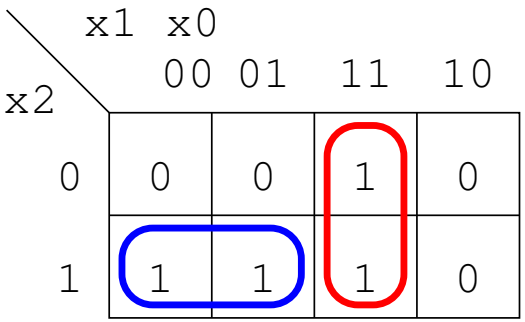


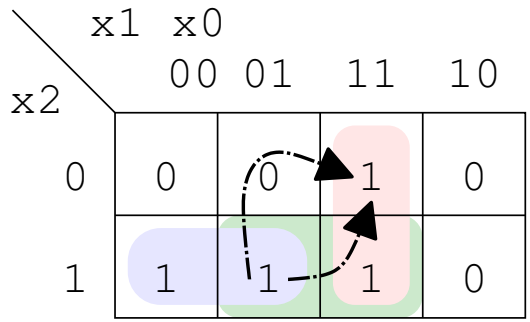



# Hazards: Funktionshazard

- Hazard, der bei gleichzeitigem Wechsel mehrerer Eingangswerte entsteht
- eine Eigenschaft der jeweiligen Schaltfunktion selbst

$\Rightarrow$ Funktionshazard kann nicht durch strukturelle Massnahmen verhindert werden




Beispiel: Übergang von $(x_2\overline{x}_1x_0)$ nach $(\overline{x}_2x_1x_0)$

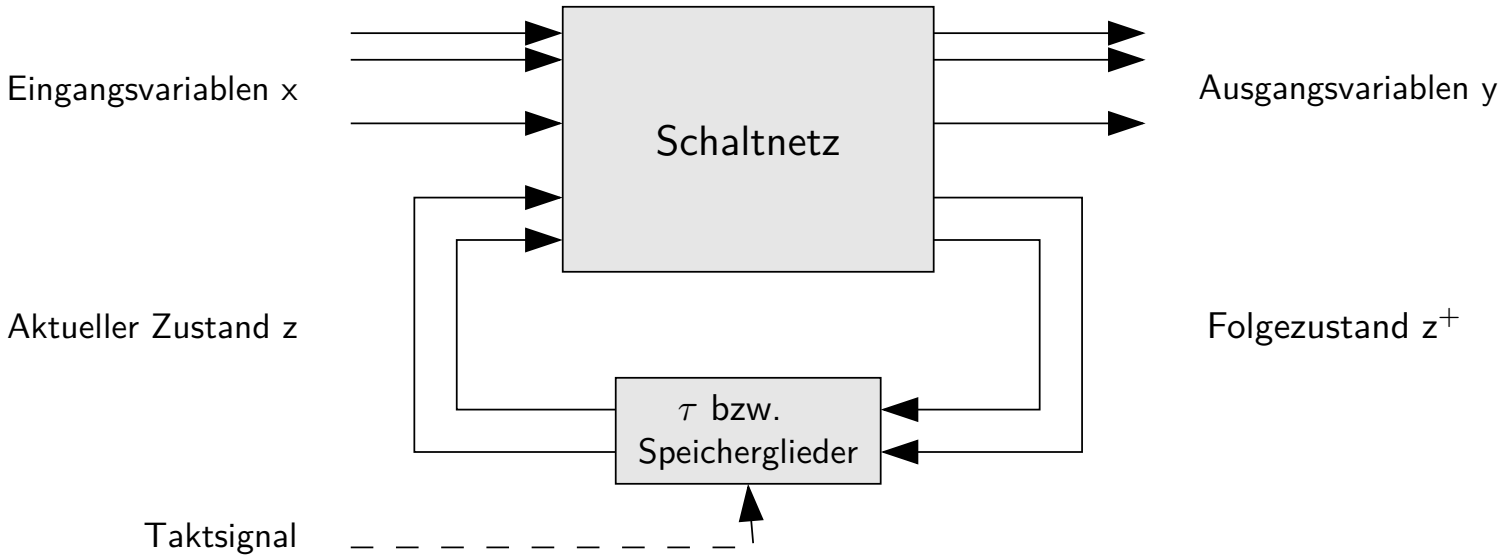
# Schaltwerke

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

- fundamental andere Eigenschaften als Schaltnetze
- Ausgangswerte nicht nur von Eingangswerten abhängig sondern auch von der Vorgeschichte

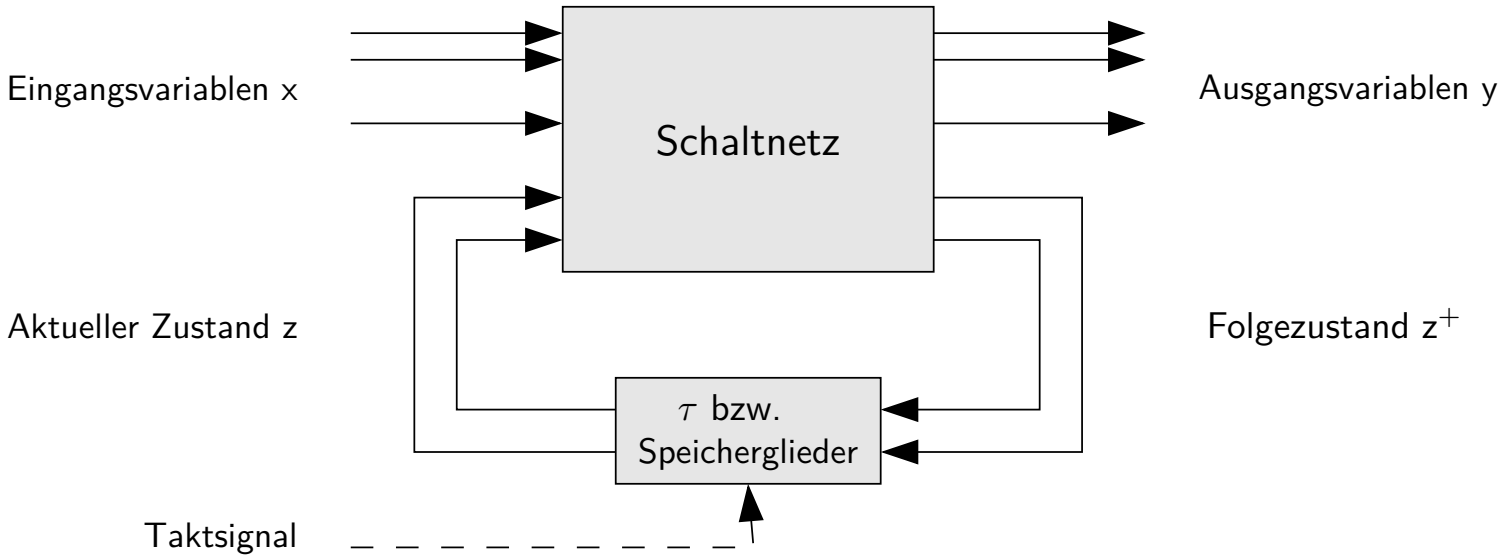
- ggf. stabile Zustände $\Rightarrow$ Speicherung von Information
- bei unvorsichtigem Entwurf: chaotisches Verhalten

# Schaltwerke: Blockschaltbild




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

# Schaltwerke: Blockschaltbild




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 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 aufwendiger Entwurf
  - 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 $(Z, \Sigma, \Delta, \delta, \lambda, z_0)$
  - $Z$ Menge von Zuständen
  - $\Sigma$ Eingabealphabet
  - $\Delta$ Ausgabealphabet
  - $\delta$ Übergangsfunktion $\delta : Z \times \Sigma \to Z$
  - $\lambda$ Ausgabefunktion $\lambda : Z \times \Sigma \to \Delta$ Mealy-Modell
    $\lambda : Z \to \Delta$ Moore-Modell
  - $z_0$ Startzustand

# Moore-Modell und Mealy-Modell

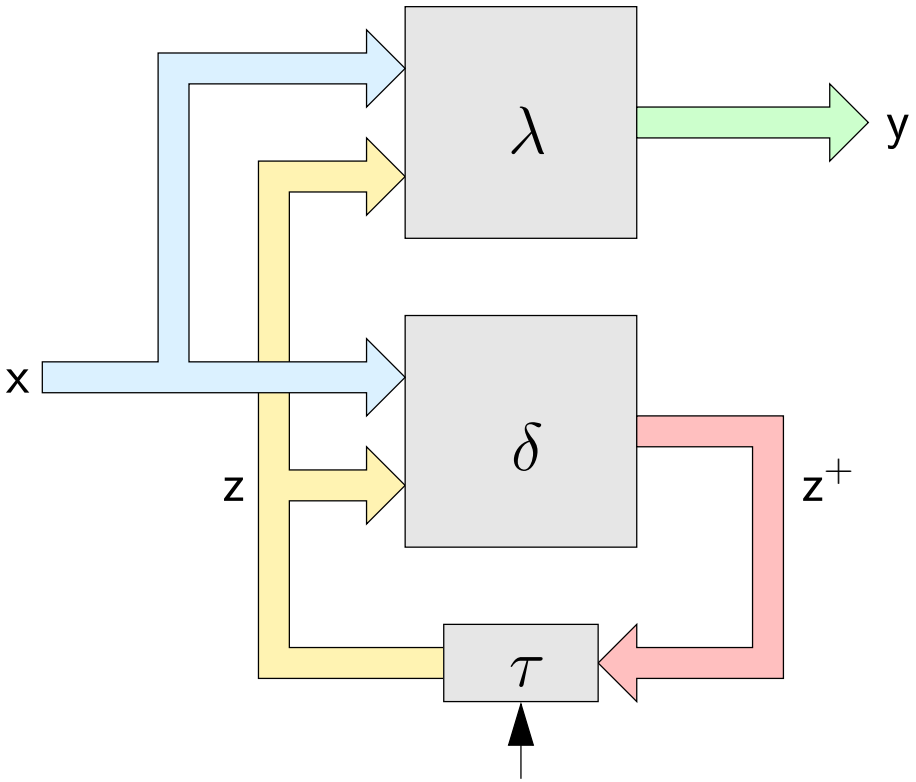
- **Moore-Modell**: die Ausgabe des Schaltwerks hängt nur vom aktuellen Zustand $z$ ab
- **Mealy-Modell**: die Ausgabe hängt vom Zustand $z$ und vom momentanen Input $x$ ab

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

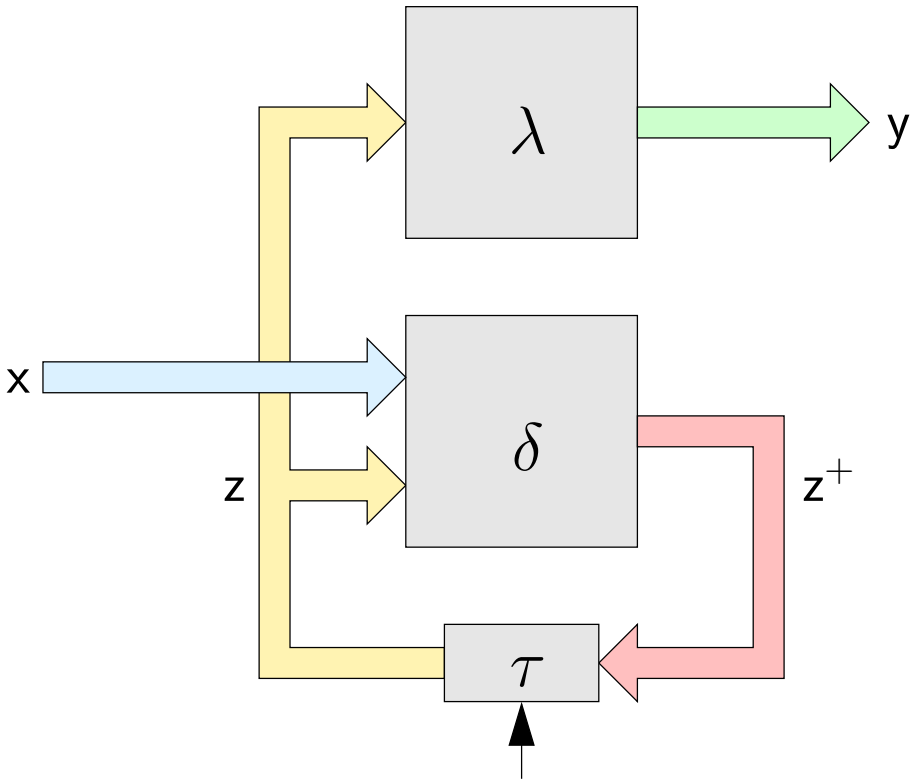
- **Speicherglieder** oder Verzögerung $\tau$ im Rückkopplungspfad

# Mealy- und Moore-Modell

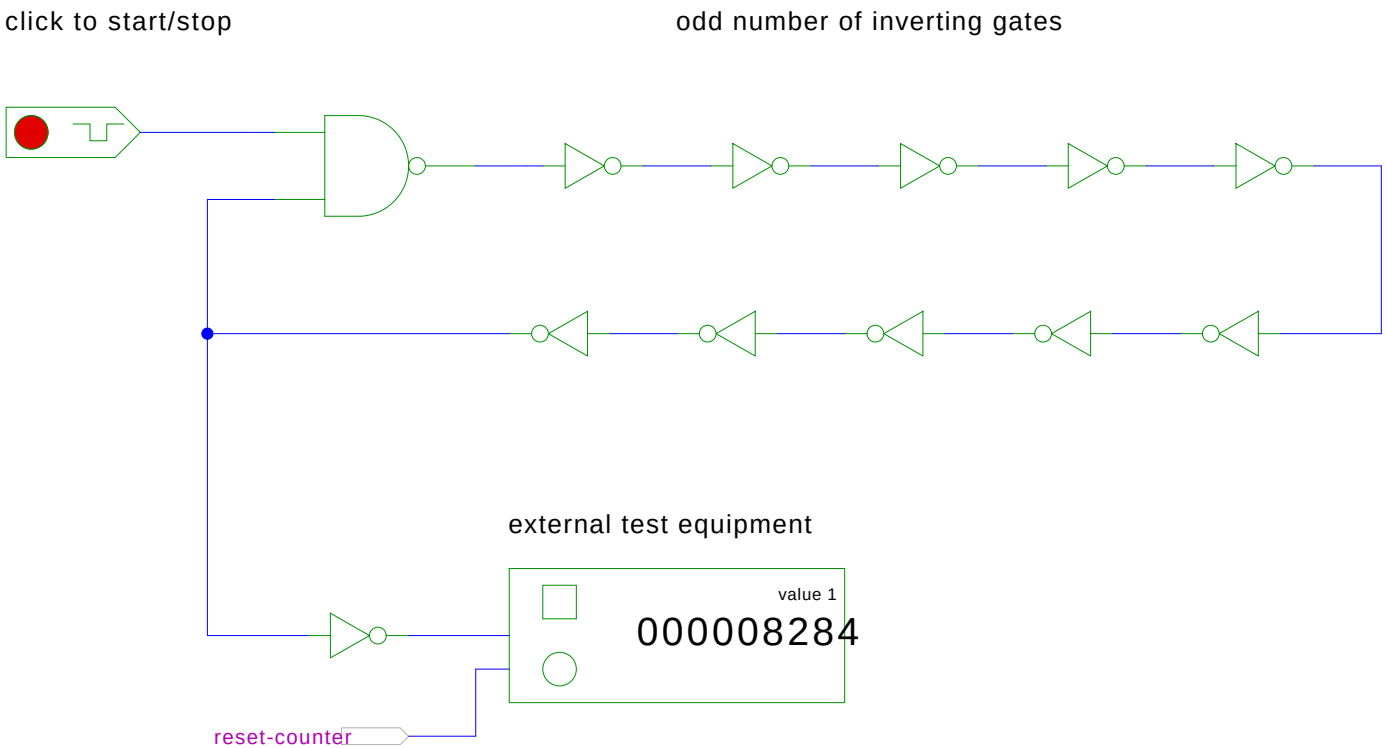
## Mealy-Automat




## Moore-Automat

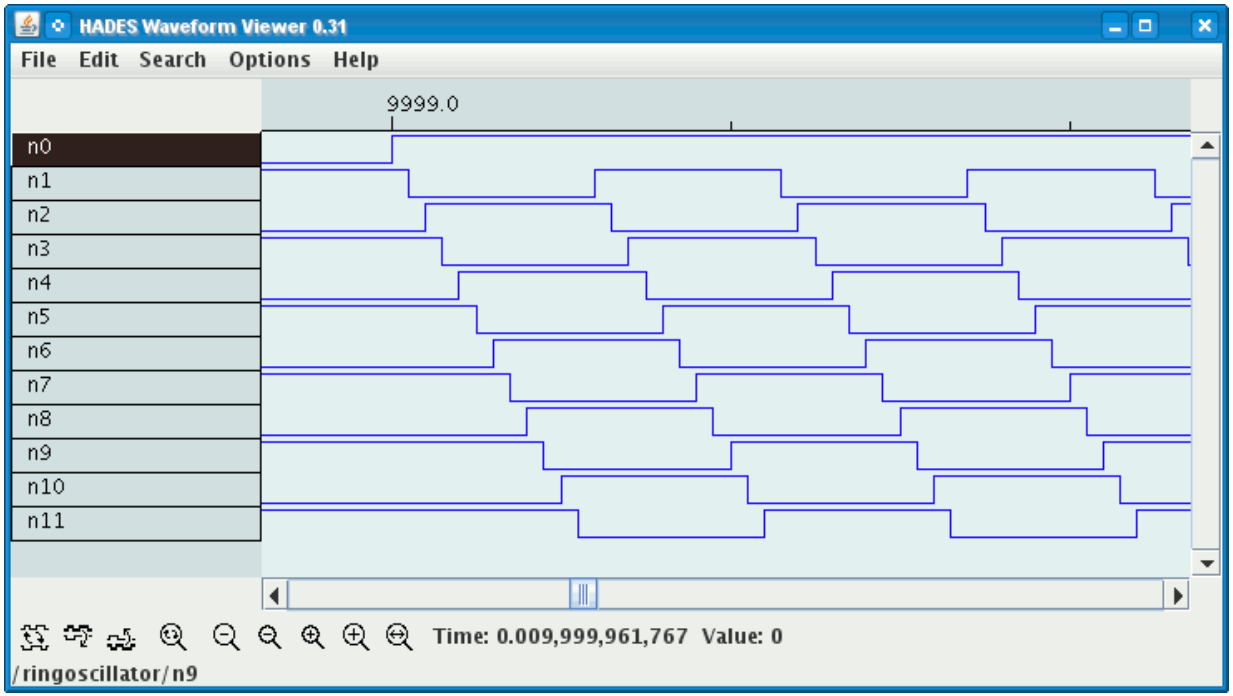


# Asynchrone Schaltungen: Beispiel Ringoszillator




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

# Ringoszillator: Impulsdiagramm




- 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
- außerdem Variation durch Umweltparameter (z.B. Temperatur)

$\Rightarrow$ sehr schwierig, die korrekte Funktion zu garantieren
z.B. über 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 einen stabilen Zustand, unabhängig von den Eingabewerten und Vorgängen im $\delta$-Schaltnetz

- Hinzufügen eines zusätzlichen Eingangssignals: „Takt“
- die Zeitglieder werden über das Taktsignal gesteuert
  - dabei verschiedene Möglichkeiten: Pegel- und Flankensteuerung, Mehrphasentakte (s.u.)

$\Rightarrow$ 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 $\overline{Q}$ verfügbar

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

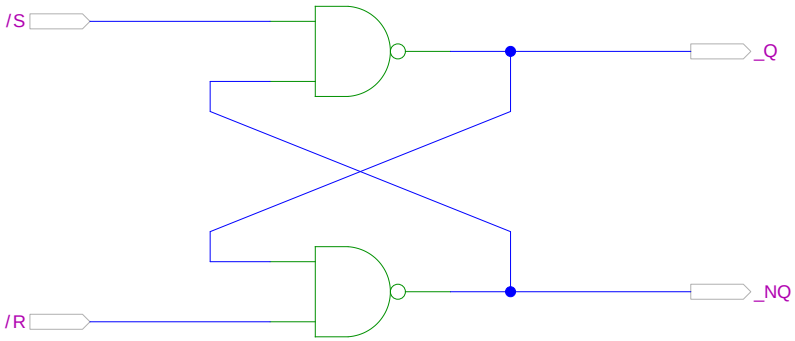
# Flipflops: Typen

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

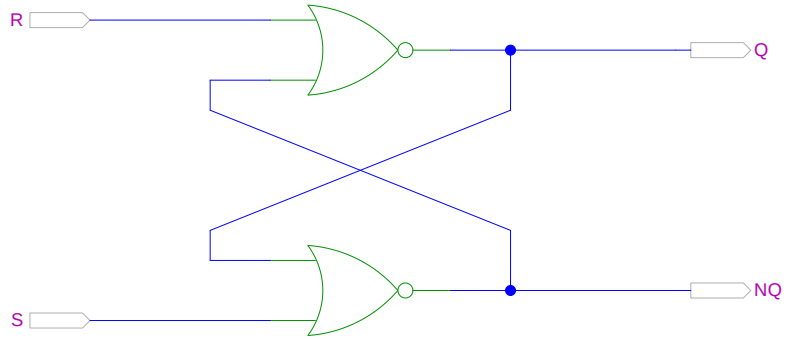
- pegelgesteuertes D-Flipflop („D-Latch“)
- flankengesteuertes D-Flipflop („D-Flipflop“)

- JK-Flipflop
- weitere. . .

# RS-Flipflop: NOR und NAND-Realisierung

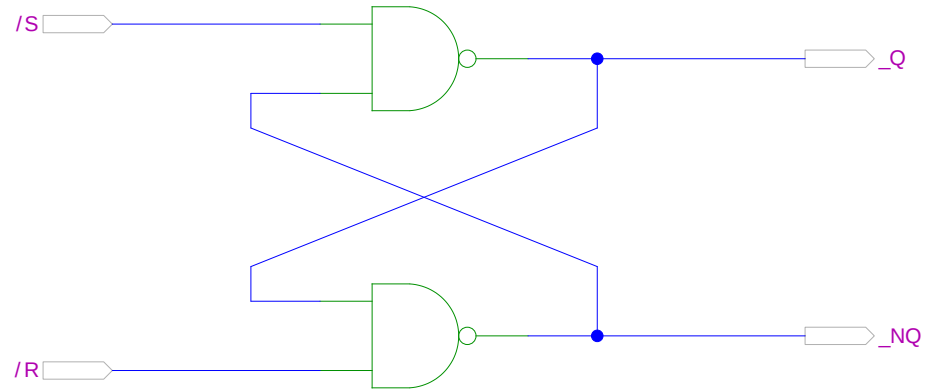


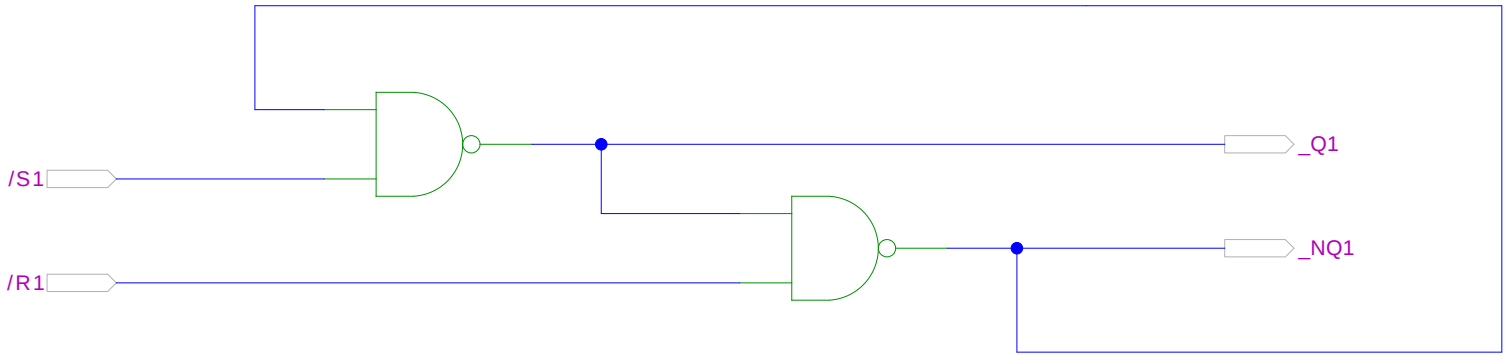

| /S | /R | Q | NQ | NAND |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | (forbidden) |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | |
| 1 | 1 | Q* | NQ* | (store) |



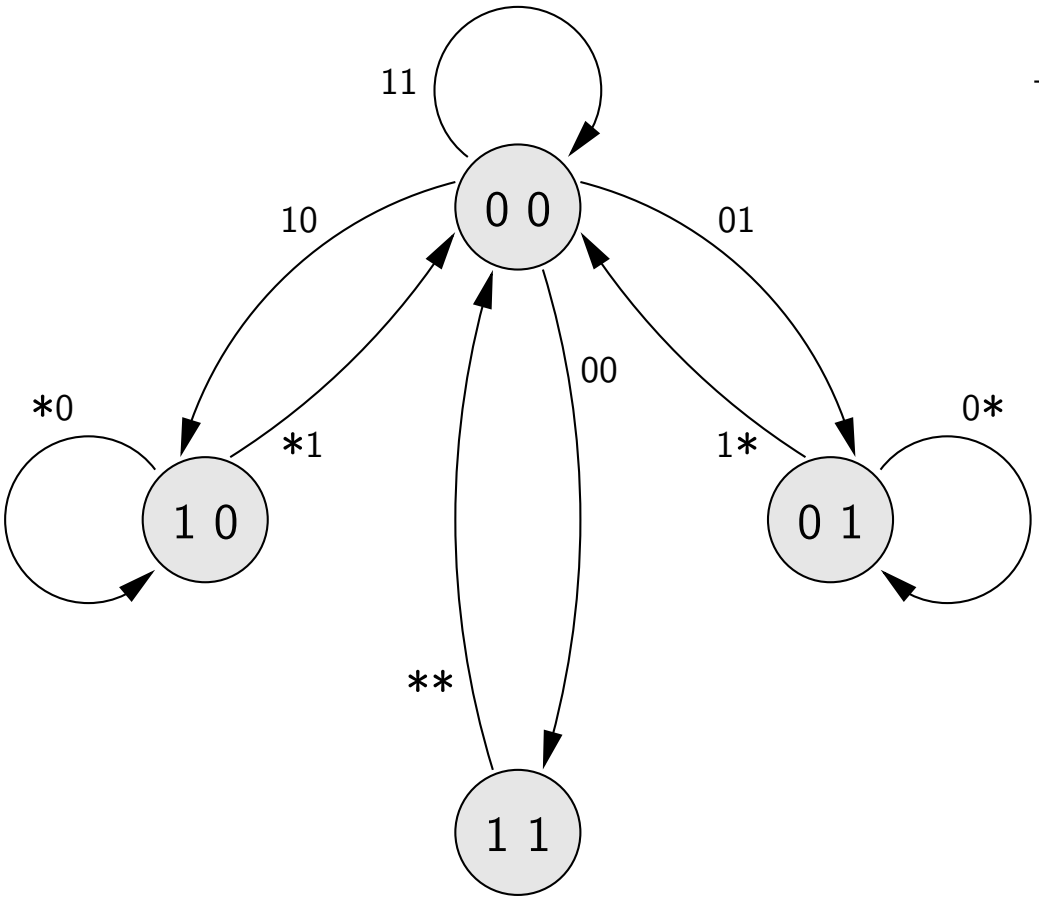

| S | R | Q | NQ | NOR |
|---|---|---|---|---|
| 0 | 0 | Q* | NQ* | (store) |
| 0 | 1 | 0 | 1 | |
| 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 0 | (forbidden) |

# RS-Flipflop: Varianten des Schaltbilds



# NOR RS-Flipflop: Zustandsdiagramm und Flusstafel

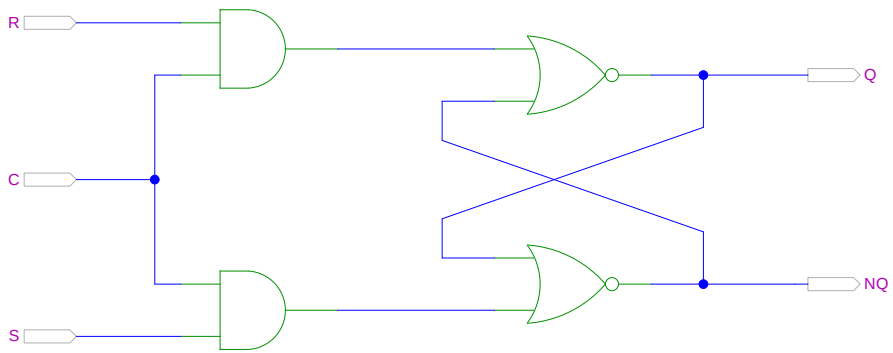



| | Eingabe $[S\,R]$ | | | |
|---|---|---|---|---|
| | 0 0 | 0 1 | 1 1 | 1 0 |
| Zustand | Folgezustand | | | $[Q\,\overline{Q}]$ |
| 0 0 | 1 1 | 0 1 | 0 0 | 1 0 |
| 0 1 | 0 1 | 0 1 | 0 0 | 0 0 |
| 1 1 | 0 0 | 0 0 | 0 0 | 0 0 |
| 1 0 | 1 0 | 0 0 | 0 0 | 1 0 |

stabiler Zustand

# RS-Flipflop mit Takt: Struktur

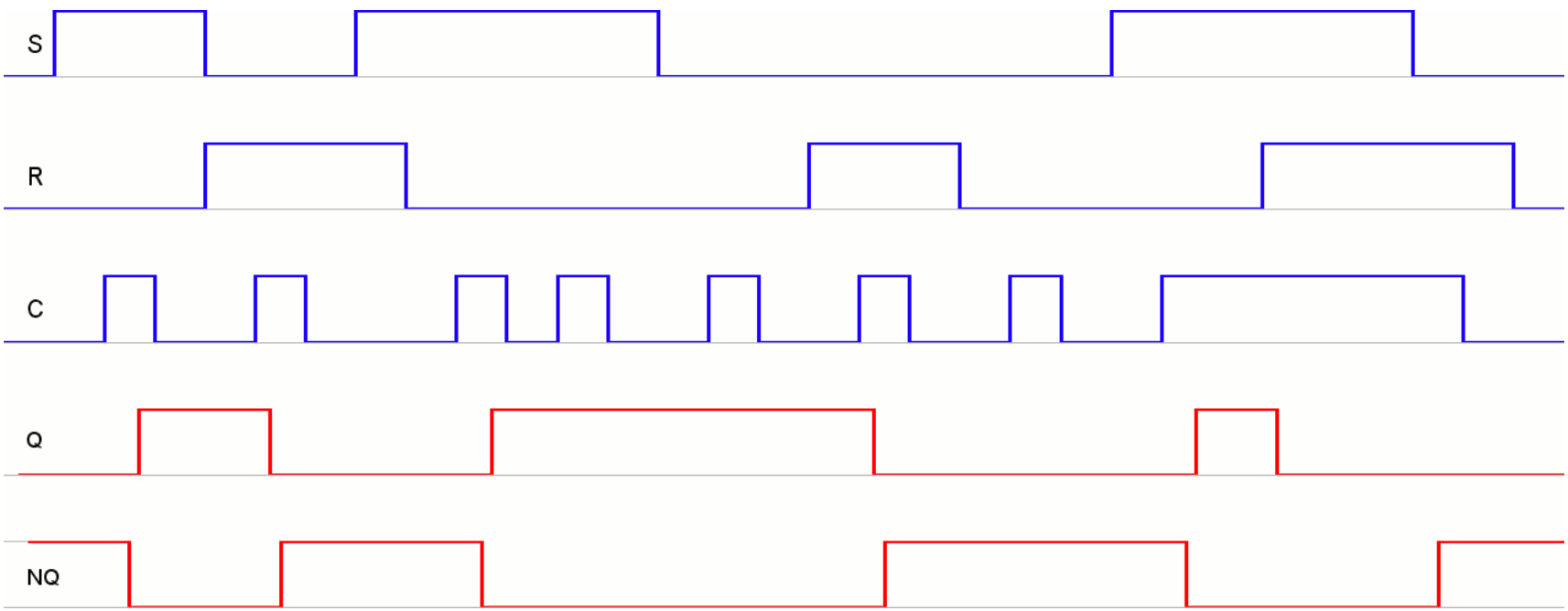
- RS-Basisflipflop mit zusätzlichem Takteingang $C$
- Änderungen nur wirksam, während $C$ aktiv ist




| 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) |

# RS-Flipflop mit Takt: Waveforms

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



# Pegelgesteuertes D-Flipflop (D-Latch)

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

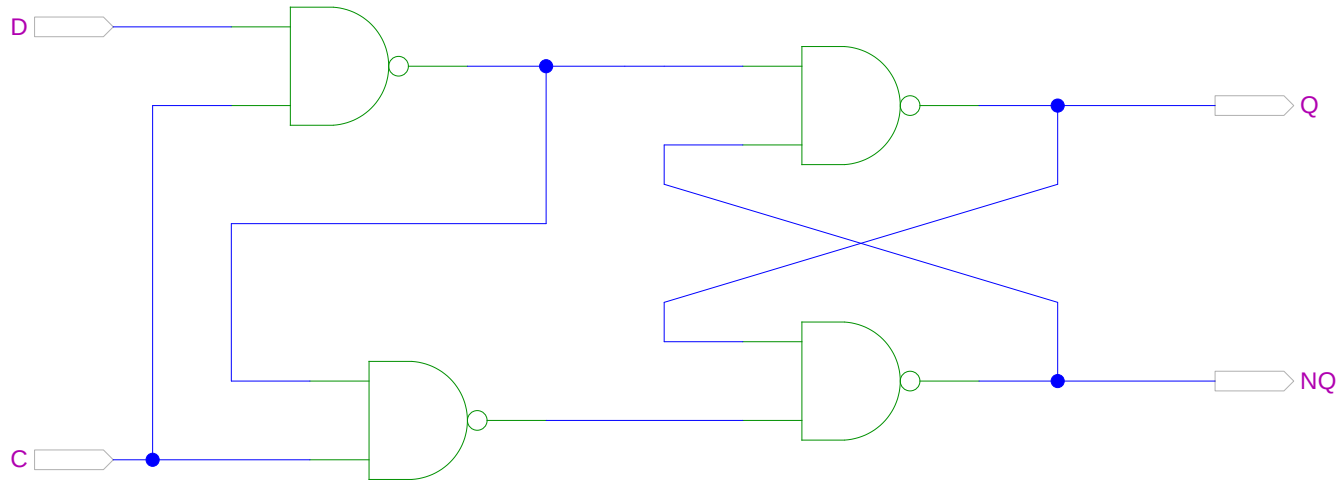
| $C$ | $D$ | $Q^+$ |
|---|---|---|
| 0 | 0 | $Q$ |
| 0 | 1 | $Q$ |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Takt $C = 0$: Wert wird gespeichert

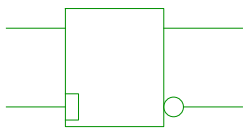
Takt $C = 1$: Eingangswert wird durchgeleitet („transparent“)

# D-Latch: minimierte NAND-Schaltung und Symbol

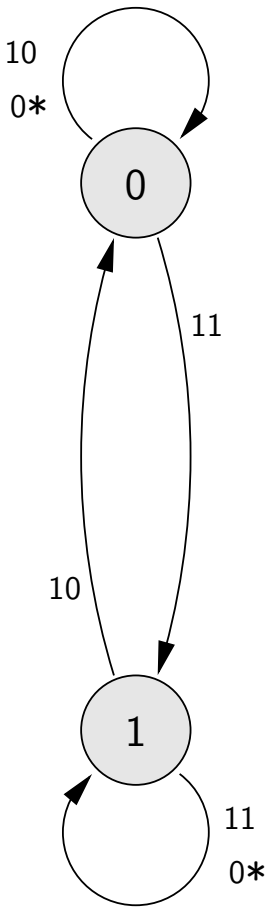
- Realisierung mit getaktetem RS-Flipflop und einem Inverter
  $S = D$, $R = \overline{D}$
- oder als minimierte NAND Schaltung:




- Schaltsymbol



# D-Latch: Zustandsdiagramm und Flusstafel




| | Eingabe $[C\ D]$ | | | |
|---|---|---|---|---|
| | 00 | 01 | 11 | 10 |
| Zustand $[Q]$ | Folgezustand $[Q^+]$ | | | |
| 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

stabiler Zustand

# Flankengesteuertes D-Flipflop

- Takteingang $C$
- Dateneingang $D$

| $C$ | $D$ | $Q^+$ |
|---|---|---|
| 0 | $*$ | $Q$ |
| 1 | $*$ | $Q$ |
| $\uparrow$ | 0 | 0 |
| $\uparrow$ | 1 | 1 |

- Wert am Dateneingang wird gespeichert, wenn das Taktsignal sich von 0 auf 1 ändert („Vorderflanke“)
- alternativ auch Rückflankensteuerung möglich
- Realisierung als Master-Slave Flipflop oder direkt

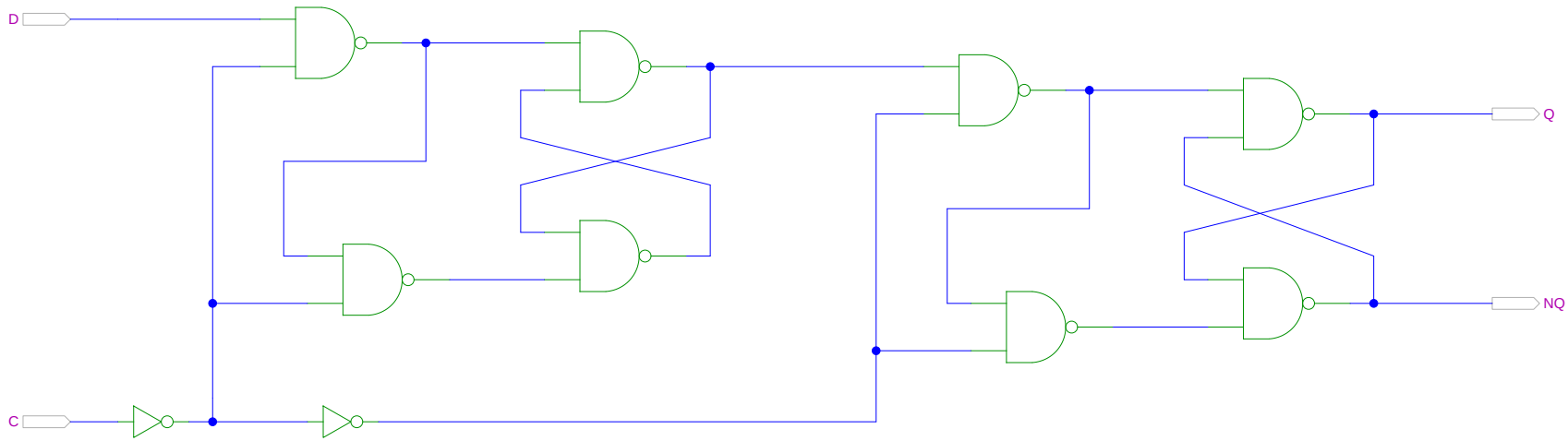
# Master-Slave D-Flipflop

- zwei kaskadierte D-Latches
- hinteres Latch mit gegenüber vorderem invertiertem Takt

- vorderes „Master“-Latch transparent (aktiv) während $C = 0$
- 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

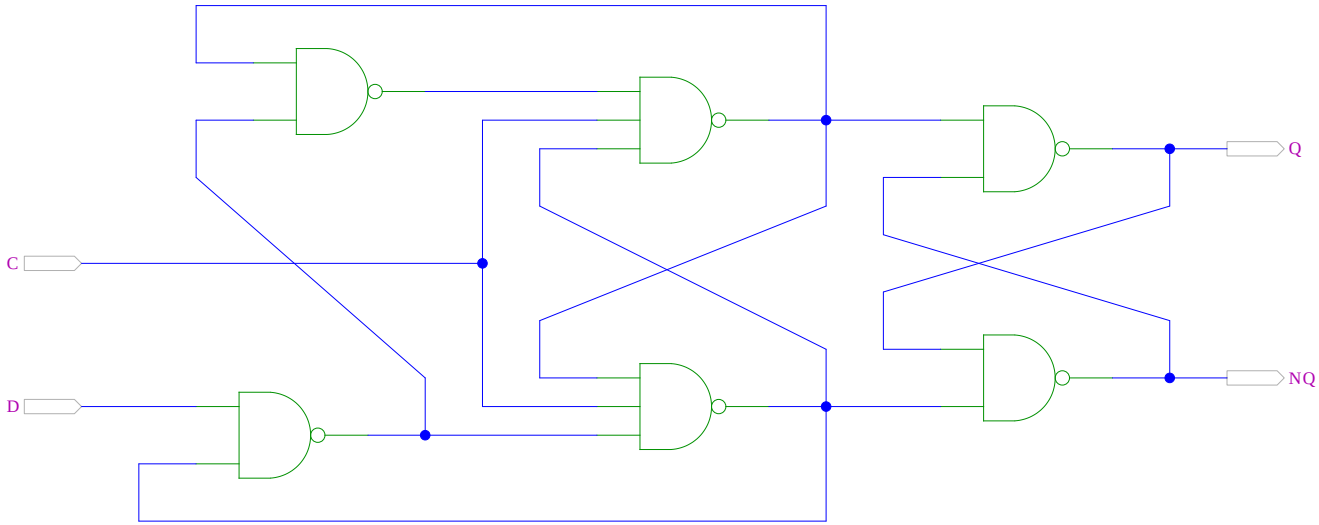
# Master-Slave D-Flipflop




- zwei kaskadierte pegel-gesteuerte D-Latches
- Master aktiv (transparent) während $C = 0$
- Slave übernimmt Master-Zustand bei Wechsel auf $C = 1$

(Hades Webdemos: 16-flipflops/20-dlatch/dff)

# Vorderflanken-gesteuertes D-Flipflop




- Dateneingang $D$ wird nur während der Takt-Vorderflanke ausgewertet
- Gatterlaufzeiten für Funktion essentiell
- Einhalten der Vorlauf- und Haltezeiten vor/nach der Taktflanke

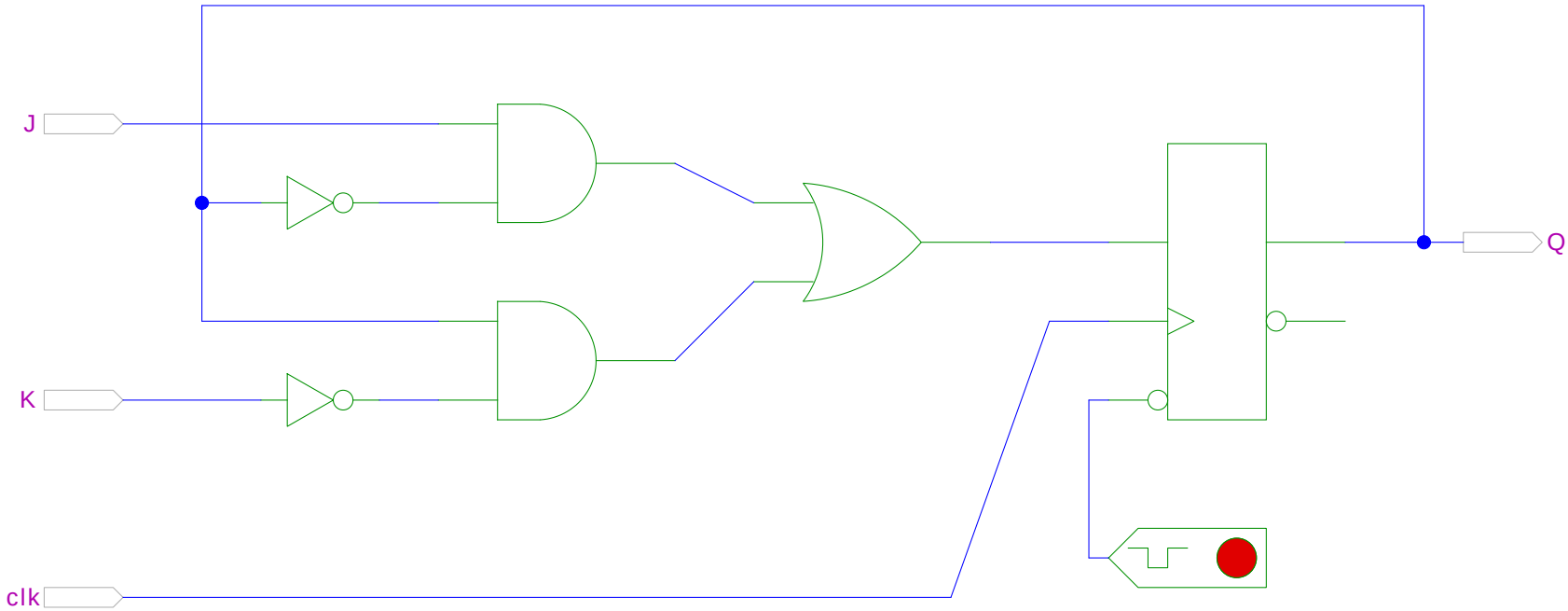
# JK-Flipflop

- Takt $C$
- Steuereingänge $J$ („jump“) und $K$ („kill“):

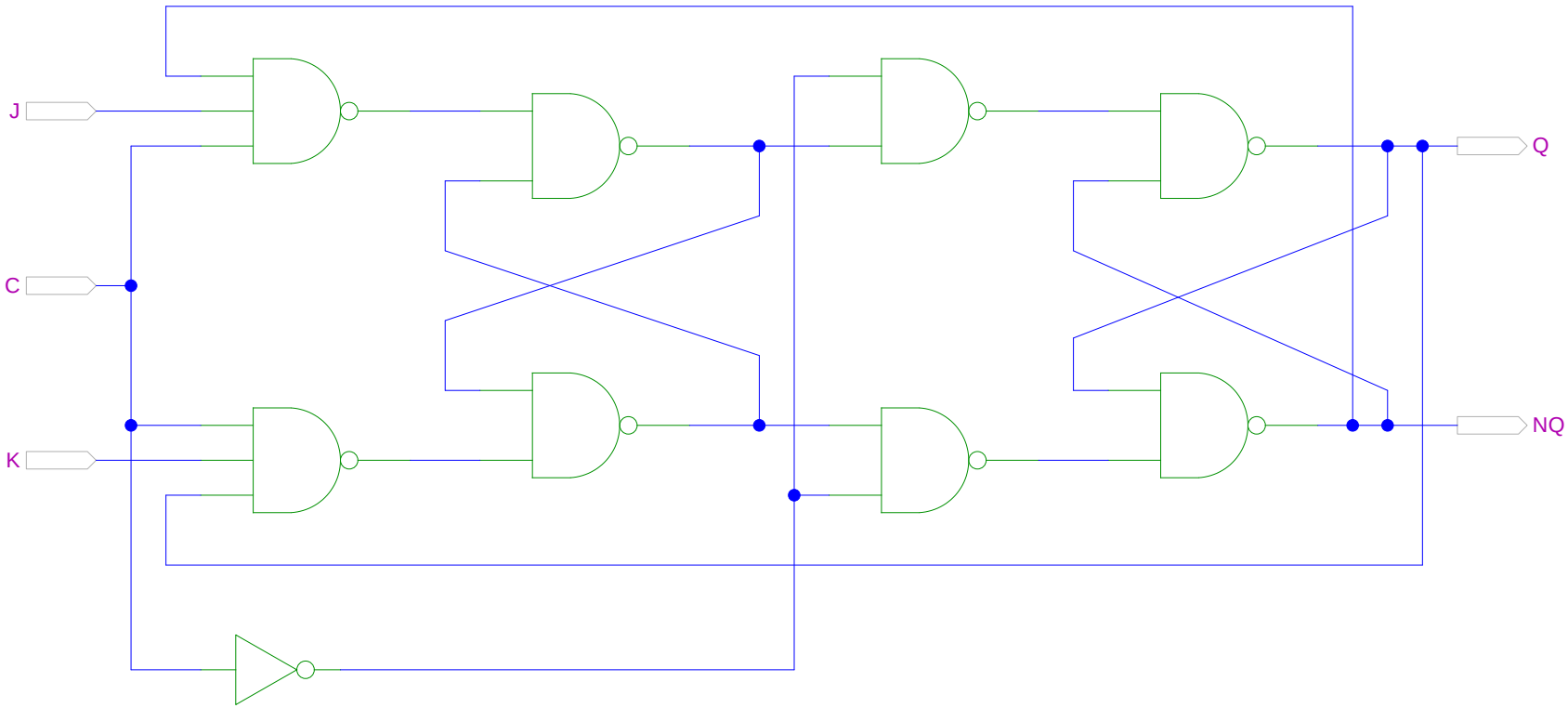
| C | J | K | $Q^+$ | Bemerkung |
|---|---|---|---|---|
| $*$ | $*$ | $*$ | $Q$ | Wert gespeichert |
| $\uparrow$ | 0 | 0 | $Q$ | Wert gespeichert |
| $\uparrow$ | 0 | 1 | 0 | Rücksetzen |
| $\uparrow$ | 1 | 0 | 1 | Setzen |
| $\uparrow$ | 1 | 1 | $\overline{Q}$ | Invertieren |

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

# JK-Flipflop: Realisierung mit D-Flipflop

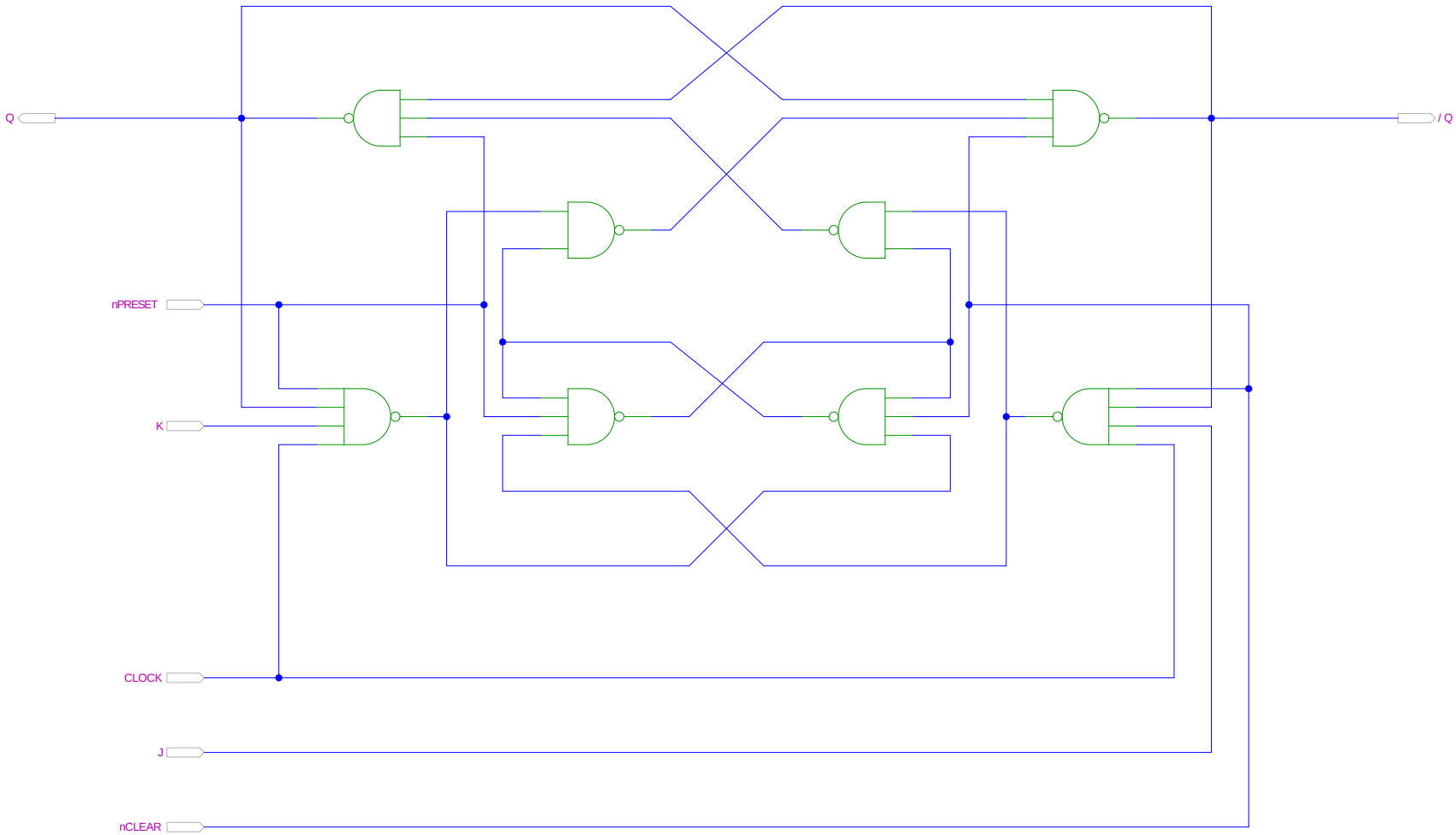


# JK-Flipflop: Master-Slave Realisierung



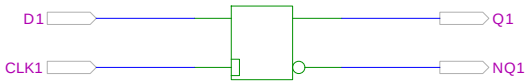

(Hades Webdemos: 16-flipflops/40-jkff/jkff  Achtung: Schaltung wegen Rückkopplungen schwer zu initialisieren)

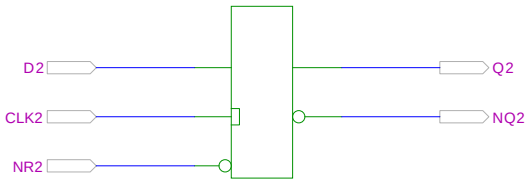
# JK-Flipflop: tatsächliche Schaltung im IC 7476



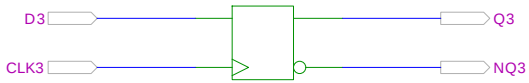
# Flipflop-Typen: Komponenten/Symbole in Hades

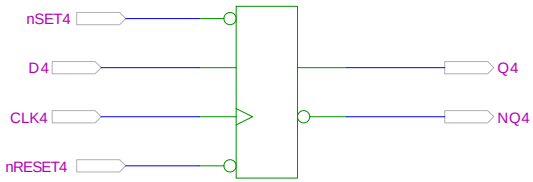
D-type latches



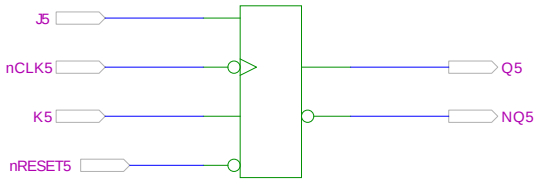




D-type flipflops

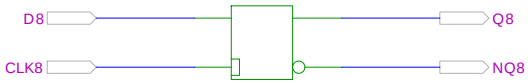


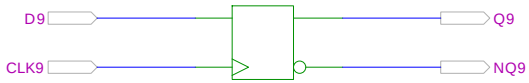

JK flipflop



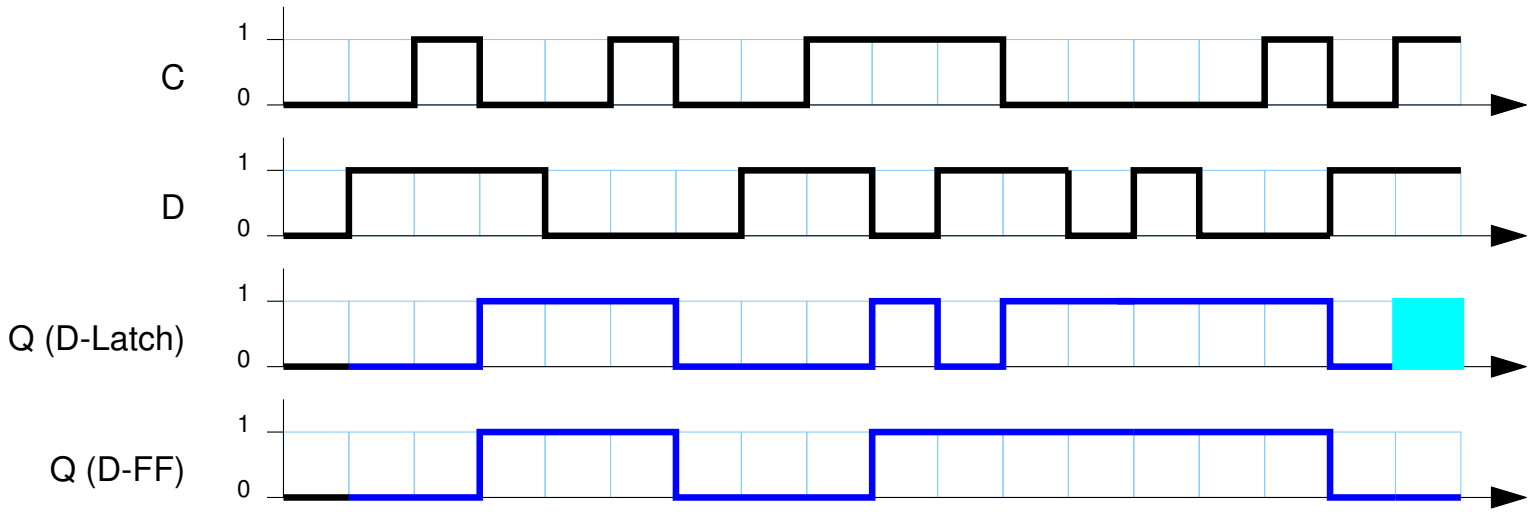

metastable D-Latch (don't use!)




metastable D-flipflop (don't use!)



# Flipflop-Typen: Impulsdiagramme




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

$\Rightarrow$ „Vorlauf- und Haltezeiten“ („setup-time“ „hold-time“)

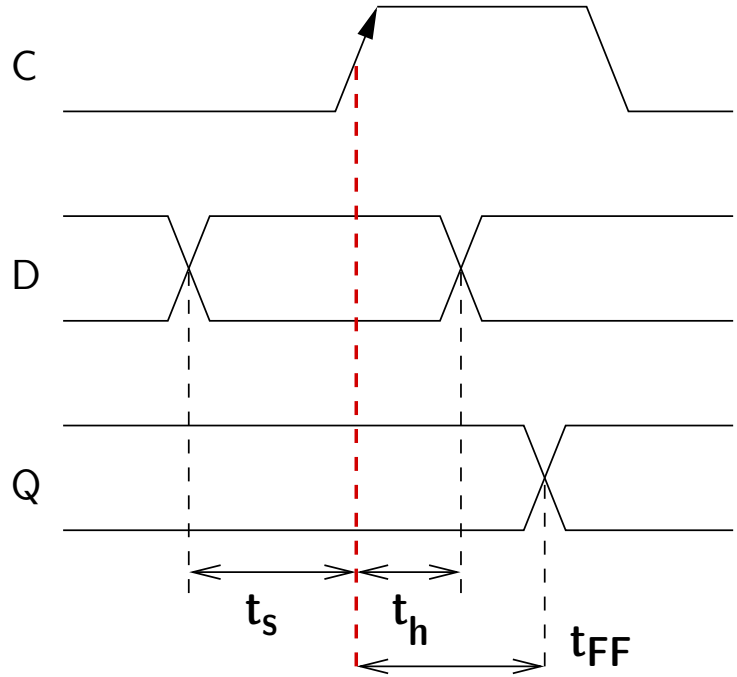
$\Rightarrow$ Daten- und Takteingänge dürfen sich nie gleichzeitig ändern

# Flipflops: Vorlauf- und Haltezeit

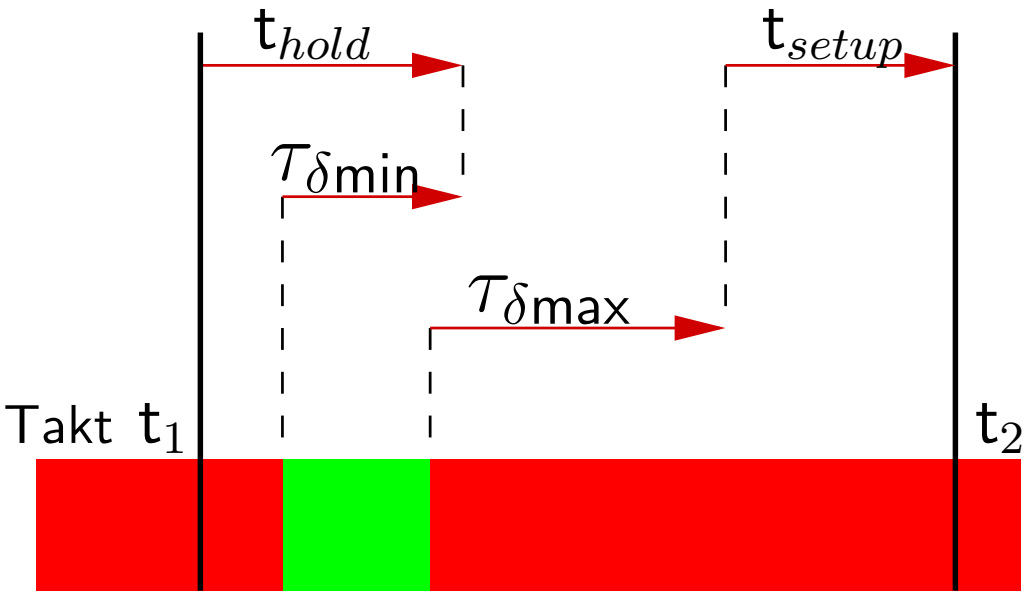
- $t_s$ Vorlaufzeit (engl. *setup-time*): Zeitintervall, innerhalb dessen das Datensignal *vor* dem nächsten Takt stabil anliegen muss
- $t_h$ Haltezeit (engl. *hold-time* ): Zeitintervall, innerhalb dessen das Datensignal *nach* einem Takt noch stabil anliegen muss

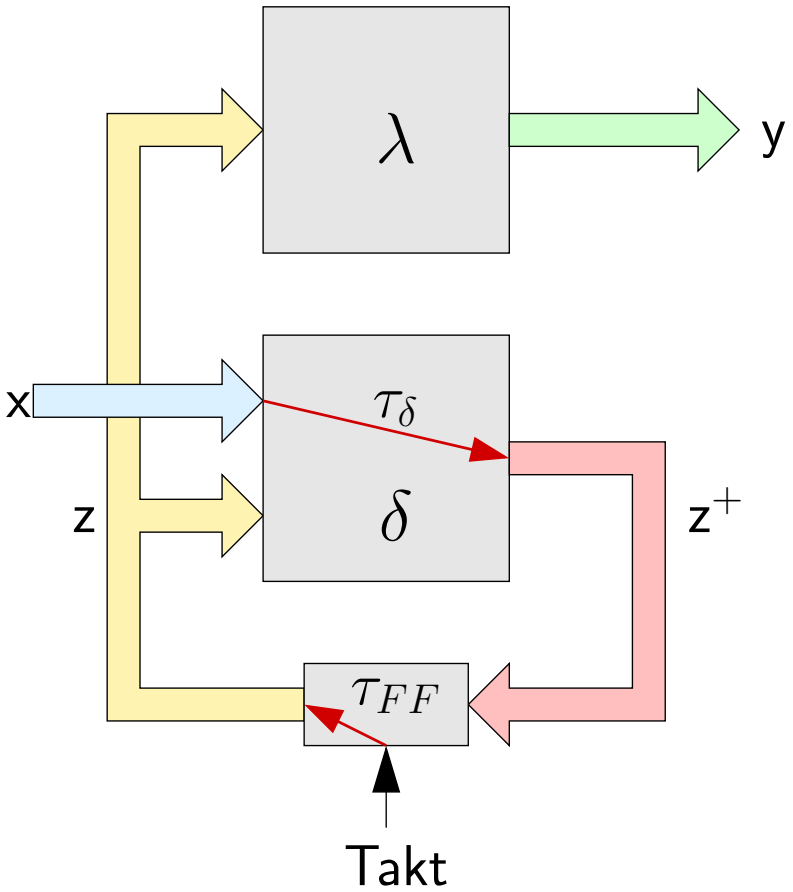
$\Rightarrow$ Verletzung der Zeitbedingungen „falscher“ Wert an Q

- $t_{FF}$ Ausgangsverzögerung



# Zeitbedingungen: Eingangsvektor



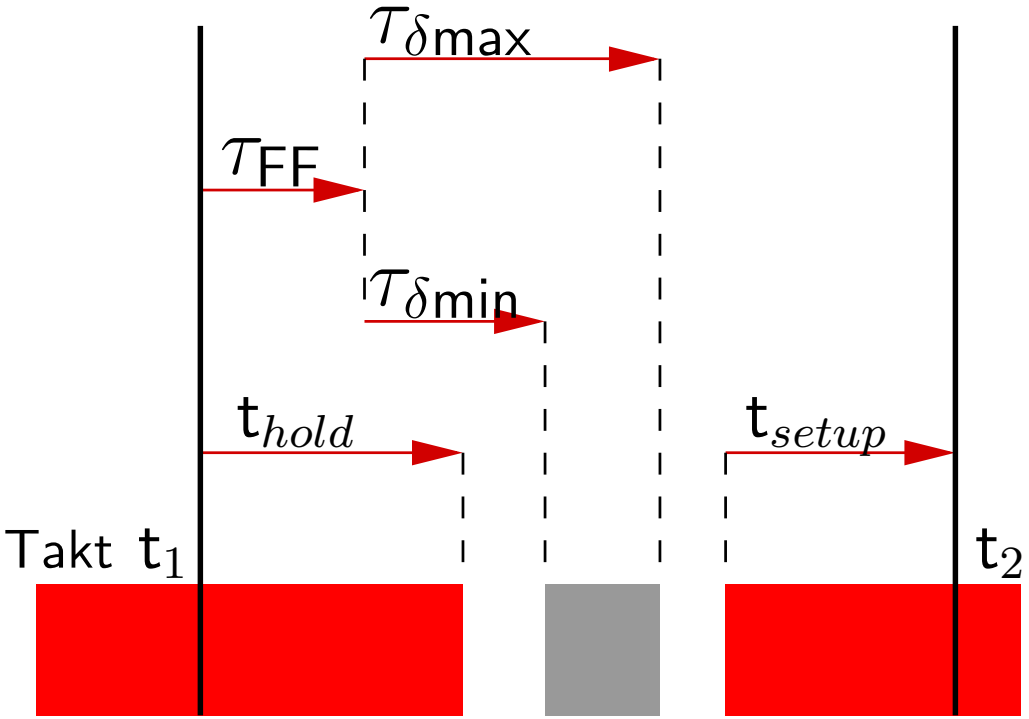



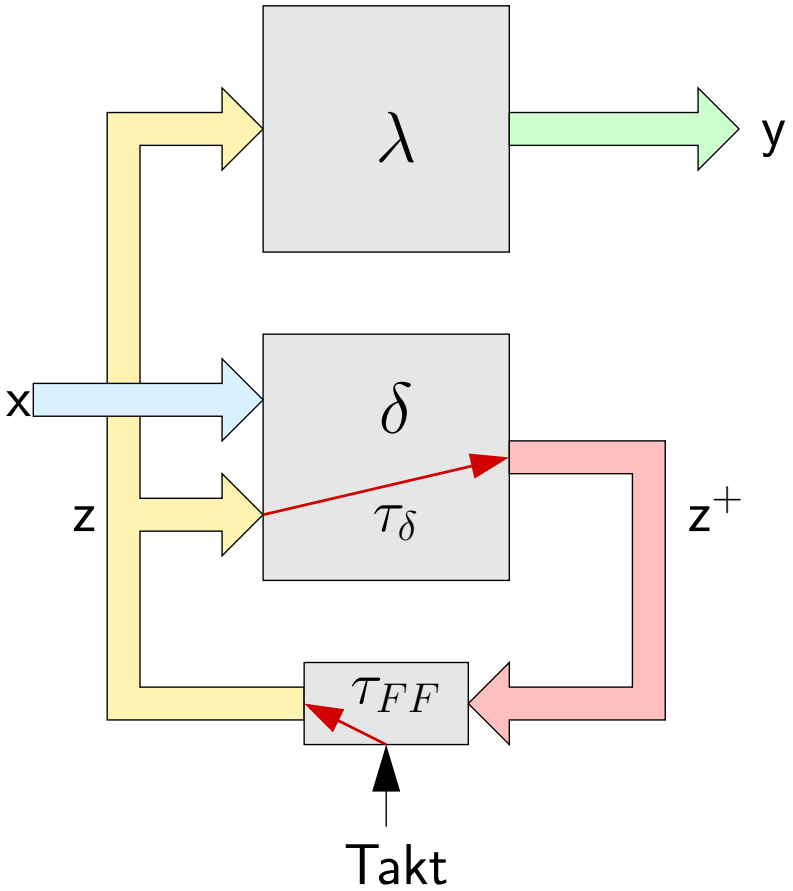
# Zeitbedingungen: Eingangsvektor (cont.)

- Änderungen der Eingangswerte $x$ werden beim Durchlaufen von $\delta$ mindestens um $\tau_{\delta_{\text{min}}}$, bzw. maximal um $\tau_{\delta_{\text{max}}}$ verzögert
- um die Haltezeit der Zeitglieder einzuhalten, darf $x$ sich nach einem Taktimpuls frühestens zum Zeitpunkt $(\text{t}_1 + \text{t}_{hold} - \tau_{\delta_{\text{min}}})$ wieder ändern
- um die Vorlaufzeit vor dem nächsten Takt einzuhalten, muss $x$ spätestens zum Zeitpunkt $(t_2 - \text{t}_{setup} - \tau_{\delta_{\text{max}}})$ wieder stabil sein

$\Rightarrow$ 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 $\tau_{\delta_{\text{min}}}$ und der langsamste ist $\tau_{\delta_{\text{max}}}$

$\Rightarrow$ innerhalb der Zeitintervalls $\tau_{FF} + \tau_{\delta_{\text{min}}}$ bis $\tau_{ff} + \tau_{\delta_{\text{max}}}$ 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)

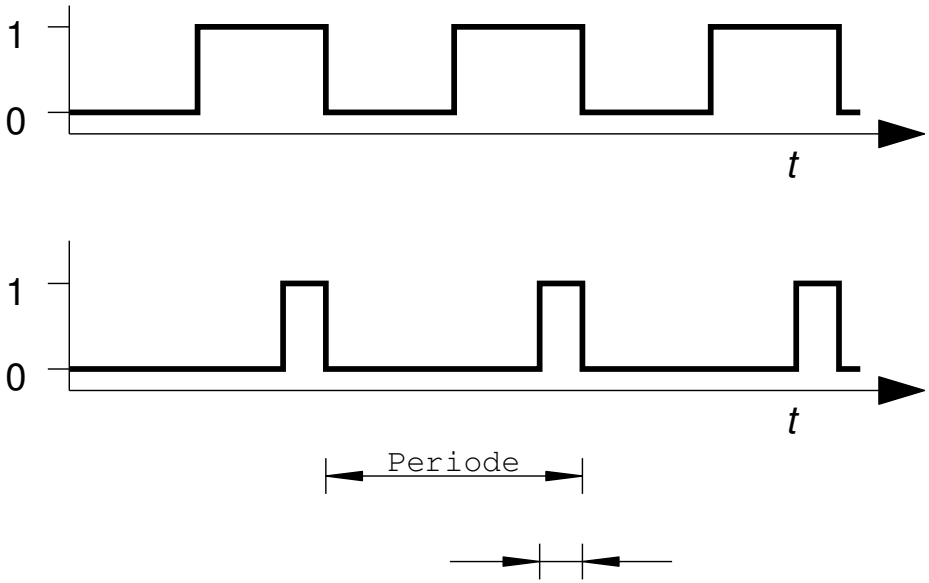
# Maximale Taktfrequenz einer Schaltung

- 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

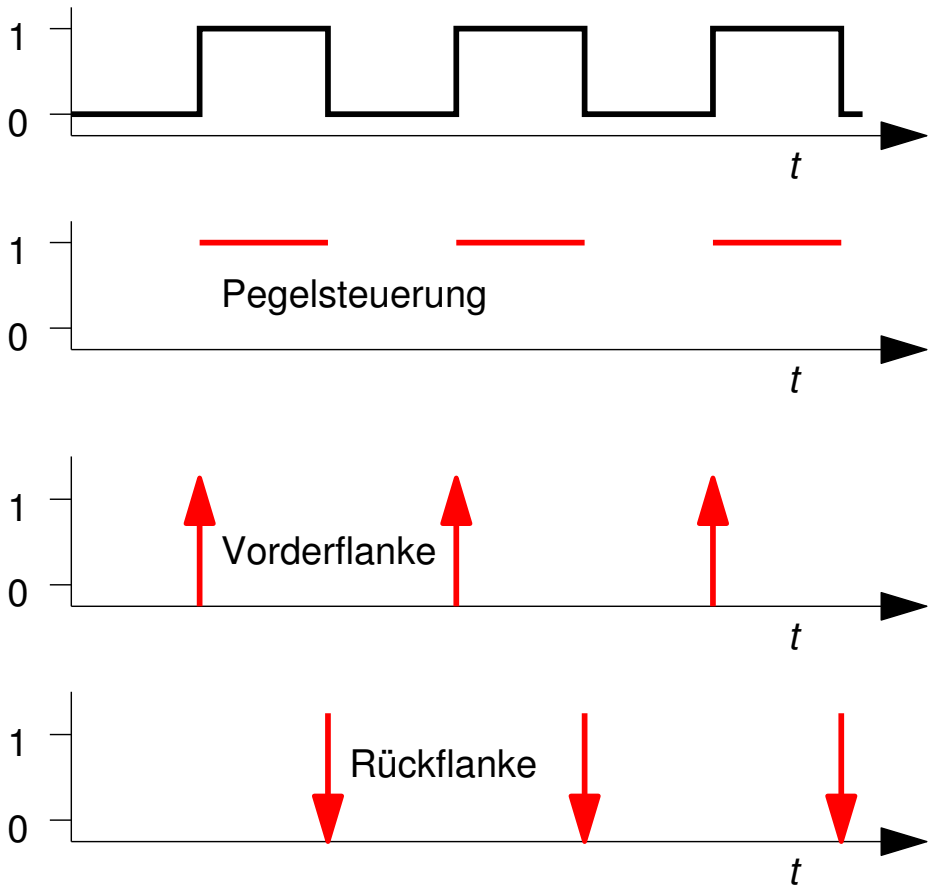



- periodisches digitales Signal, Frequenz $f$ bzw. Periode $\tau$
- oft symmetrisch
- asymmetrisch für Zweiphasentakt (s.u.)

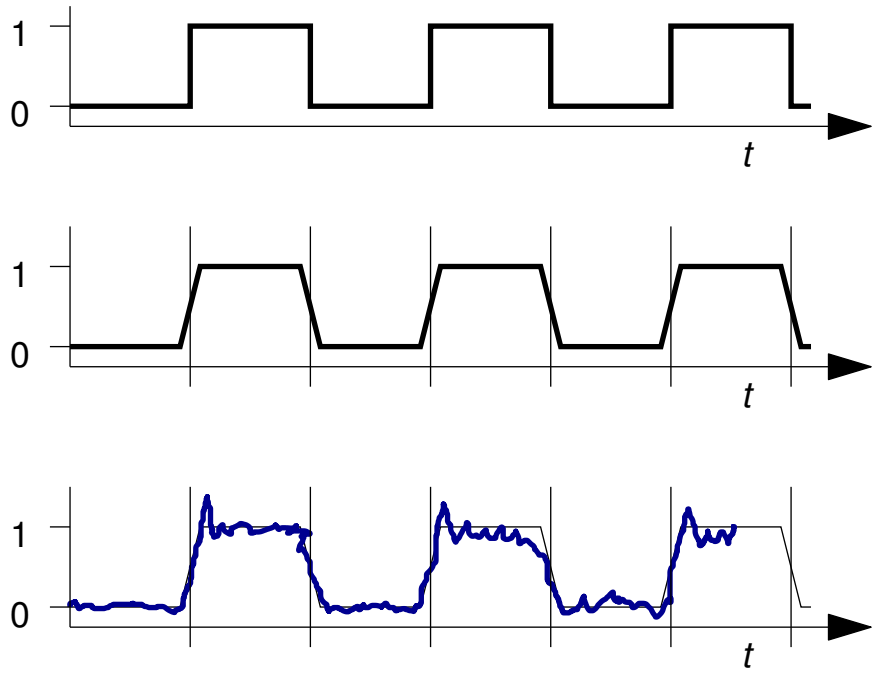
# Taktsignal: Varianten

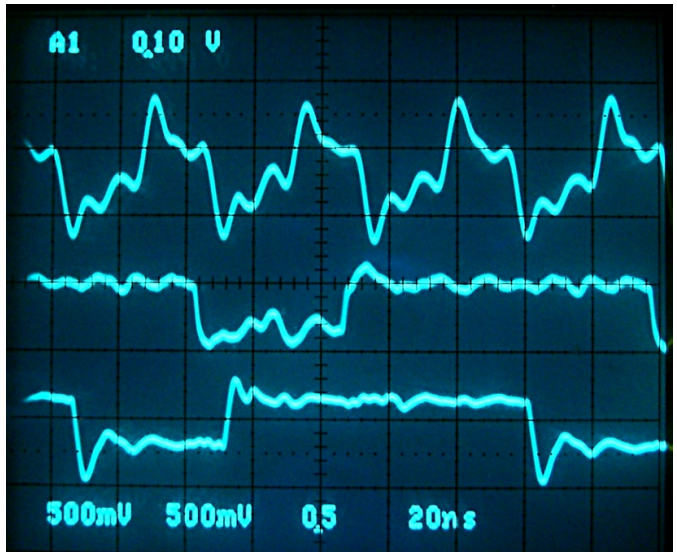
- **Pegelsteuerung**: Schaltung reagiert, während das Taktsignal den Wert 1 aufweist
- **Flankensteuerung**: Schaltung reagiert nur, während das Taktsignal seinen Wert wechselt
  - Vorderflankensteuerung: Wechsel von 0 nach 1
  - Rückflankensteuerung: Wechsel von 1 nach 0
- Zwei- und Mehrphasentakte

# Taktsignal, Vorder- und Rückflanken



# Taktsignal: Prinzip und Realität

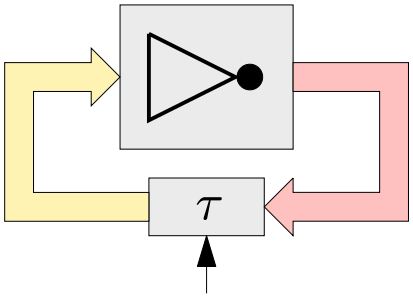



- Werteverläufe in realen Schaltungen stark gestört
- Überschwingen/Übersprechen benachbarter Signale
- Flankensteilheit nicht garantiert (bei starker Belastung)

# 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

$\Rightarrow$ entweder Verwendung von je zwei pegelgesteuerten Zeitgliedern und Einsatz von Zweiphasentakt

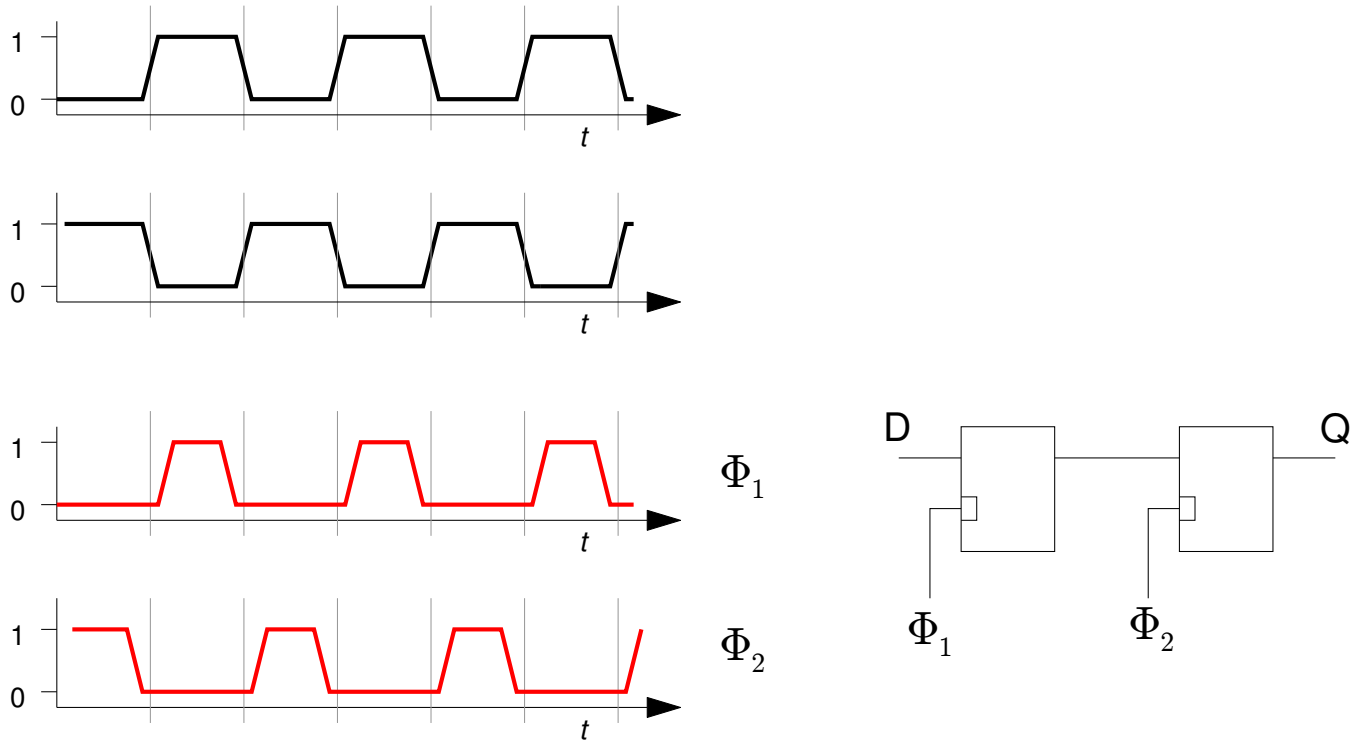
$\Rightarrow$ oder Verwendung flankengesteuerter D-Flipflops

# Zweiphasentakt

- pegelgesteuertes D-Latch ist bei aktivem Takt *transparent*
- rück-gekoppelte Werte werden sofort wieder durchgelassen
- Oszillation bei invertierten Rückkopplungen

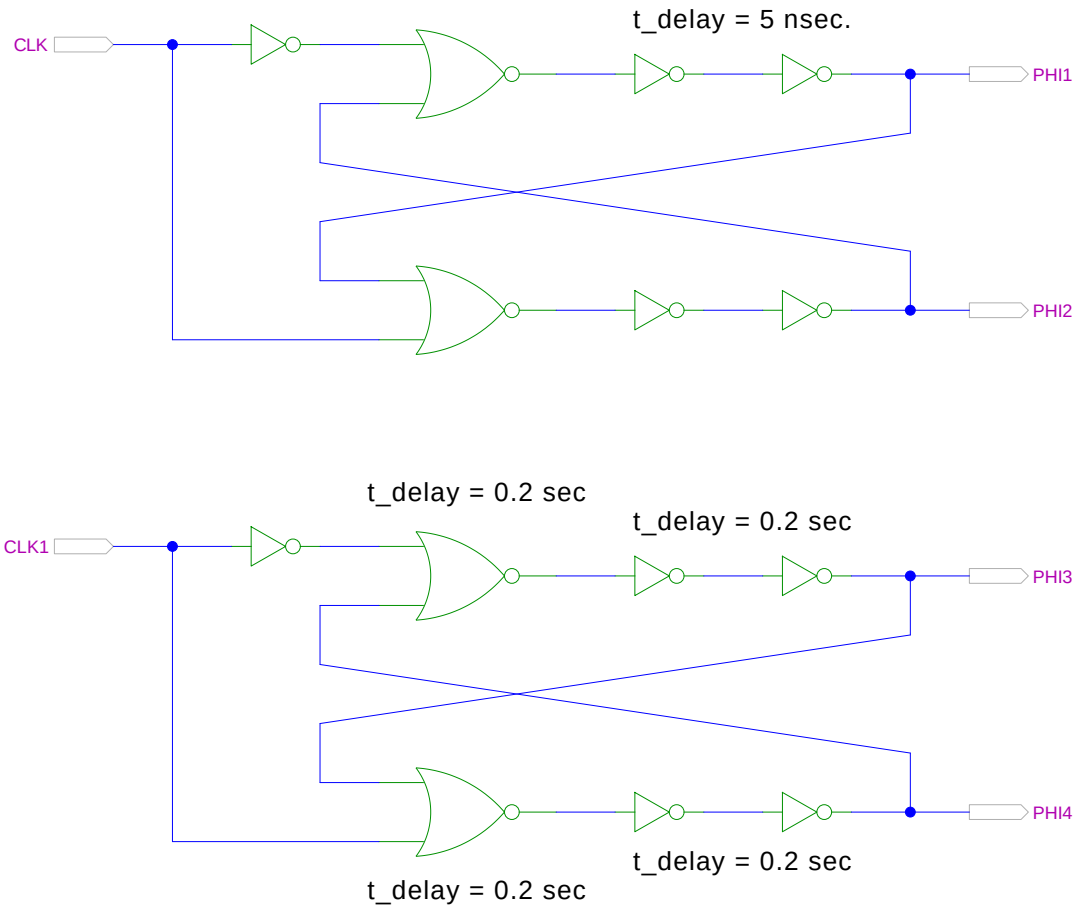
- Reihenschaltung aus jeweils zwei D-Latches
- zwei separate Takte $\Phi_1$ und $\Phi_2$
  - bei Takt $\Phi_1$ übernimmt vorderes Flipflop den Wert
  - erst bei Takt $\Phi_2$ übernimmt hinteres Flipflop

# Zweiphasentakt

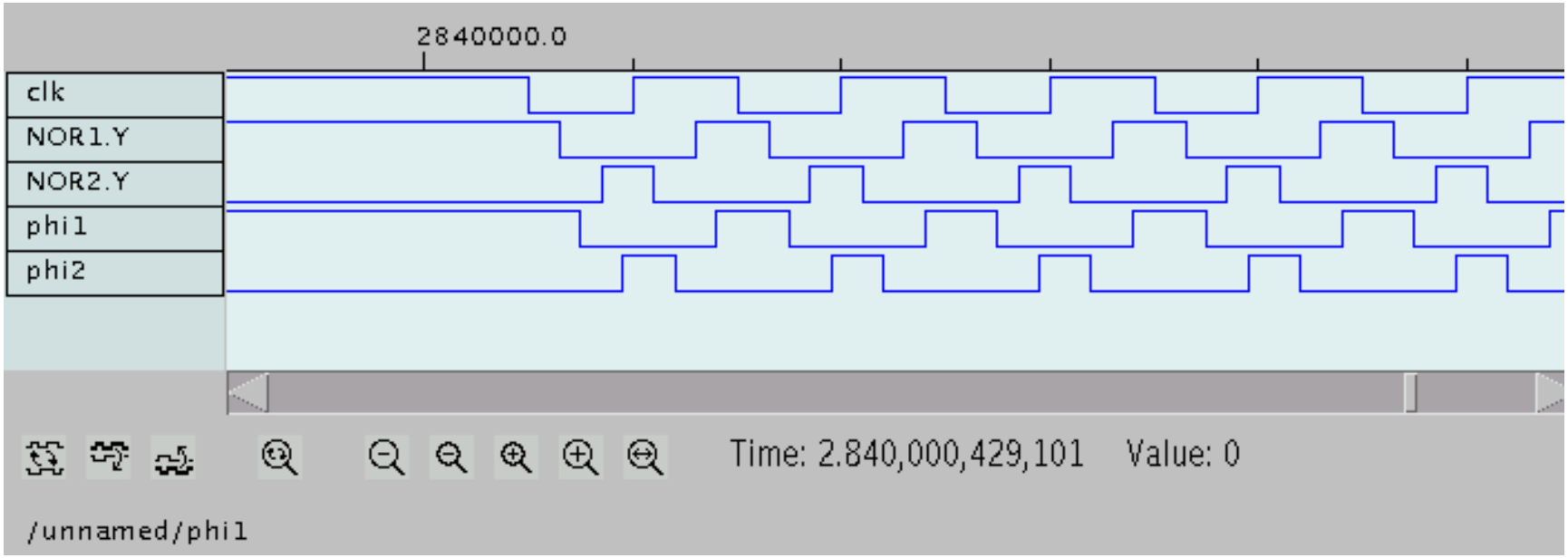



- nichtüberlappender Takt mit Phasen $\Phi_1$ und $\Phi_2$
- vorderes D-Latch übernimmt Eingangswert $D$ während $\Phi_1$
- bei $\Phi_2$ übernimmt das hintere D-Latch und liefert $Q$

# Zweiphasentakt: Erzeugung



# Zweiphasentakt: Waveforms




- Verzögerungen geeignet wählen
- Eins-Phasen der beiden Takte $c_1$ und $c_2$ sauber getrennt
- nicht-überlappende 2-Phasen-Taktimpulse
- Ansteuerung für 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 von aktuellem Zustand und den Eingabewerten
- beschreibt also das $\delta$-Schaltnetz

- **Ausgangstafel**: Tabelle für die Ausgabewerte als Funktion des aktuellen Zustands (und der Eingabewerte)
- beschreibt das $\lambda$-Schaltnetz

# Beispiel: Flusstafel für eine 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: Ausgangstafel für eine Ampel

- 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

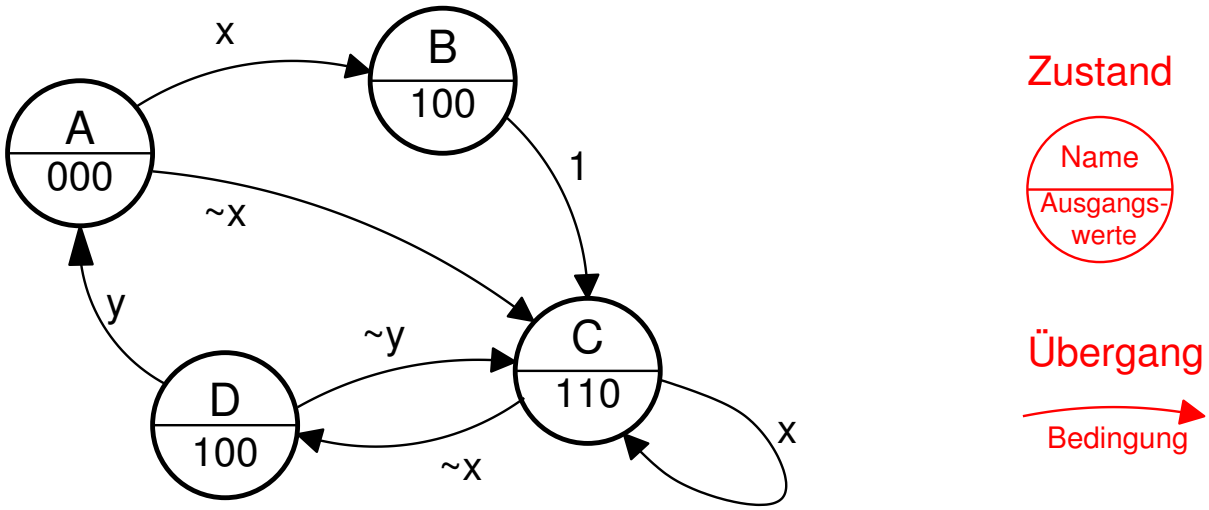
- **Zustandsdiagramm**: Graphische Darstellung eines Schaltwerks
- je ein Knoten für jeden Zustand
- je eine Kante für jeden möglichen Übergang

- Knoten werden passend benannt
- Kanten werden mit den Eingabemustern gekennzeichnet, bei denen der betreffene Übergang auftritt

- Moore-Schaltwerke: Ausgabe wird zusammen mit dem Namen im Knoten notiert
- Mealy-Schaltwerke: Ausgabe hängt vom Input ab, ggf. an den Kanten notieren

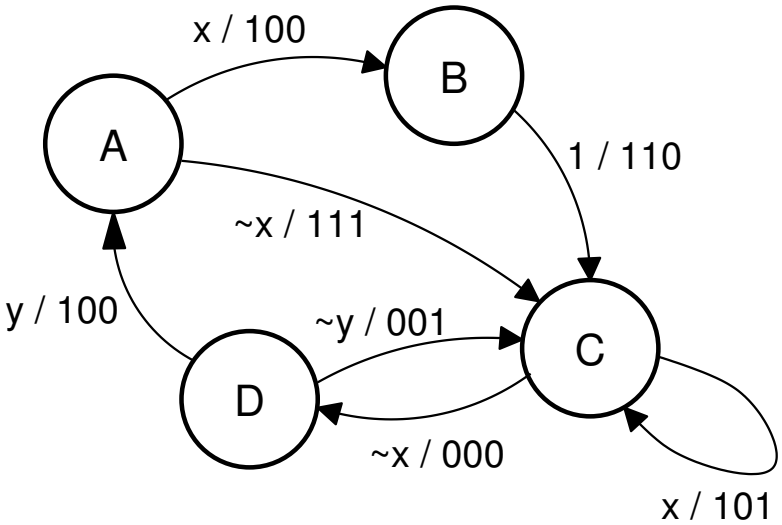
(siehe auch: http://en.wikipedia.org/wiki/State_diagram)

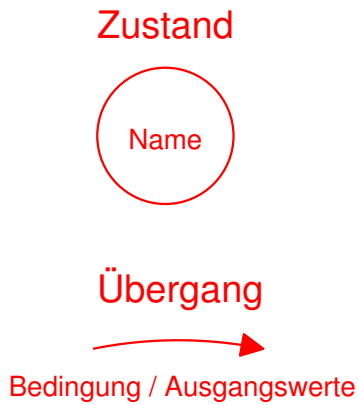
# Zustandsdiagramm: Moore-Automat




- 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. doppelter Kreis)

# Zustandsdiagramm: Mealy-Automat




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

# Hinweis: „State-Charts“

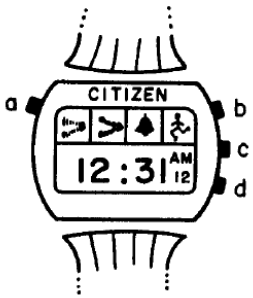
- hierarchische Version von Zustandsdiagrammen
- Knoten repräsentieren entweder einen Zustand
- oder einen eigenen (Unter-) Automaten

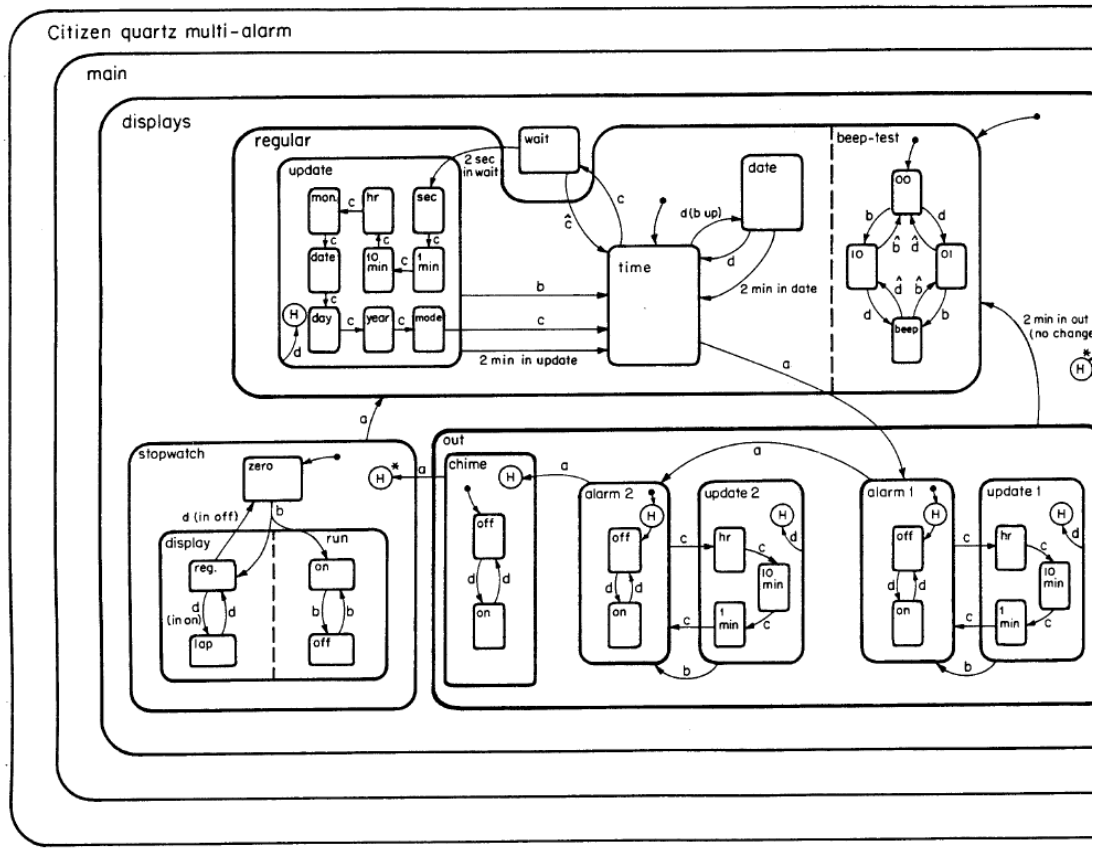
- beliebte Spezifikation für komplexe Automaten
- Embedded Systems, Kommunikationssysteme, etc.

- David Harel, *Statecharts, A visual approach to complex systems*, CS84-05, Department of Applied Mathematics, The Weizmann Institute of Science, 1984.

(www.wisdom.weizmann.ac.il/~dharel/SCANNED.PAPERS/Statecharts.pdf)

# State-Charts: Beispiel Digitaluhr




...

(www.wisdom.weizmann.ac.il/~dharel/SCANNED.PAPERS/Statecharts.pdf)

# 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 Circuits 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;
```

# Entwurf von Schaltwerken: sechs Schritte

1. Spezifikation (textuell oder graphisch, z.B. Zustandsdiagramm)
2. Aufstellen der 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: Zustandscodierung

Vielfalt möglicher Codierungen

- binäre Codierung: minimale Anzahl der Zustände
- 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: Widerspruchsfreiheit

- Entwurf ausgehend von Funktionstabellen problemlos
- alle Eingangsbelegungen und Zustände werden berücksichtigt
- don't-care Terme können berücksichtigt werden

- aber zwei typische Fehler beim 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?

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

# Überprüfung der Widerspruchsfreiheit

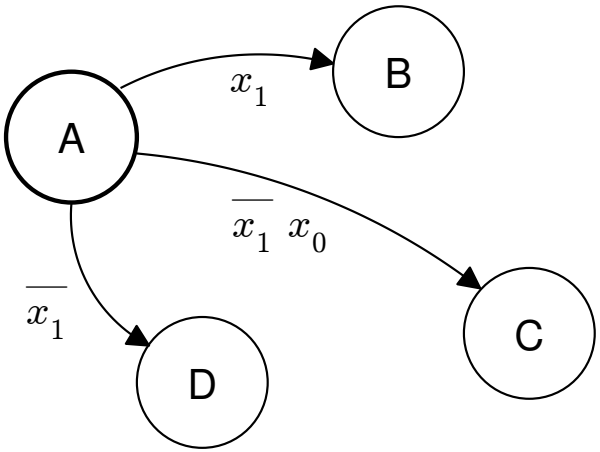
$p$ Zustände, Zustandsdiagramm mit Kanten $h_{ij}(x)$: Übergang von Zustand $i$ nach Zustand $j$ unter Belegung $x$

- für jeden Zustand überprüfen:

  kommen alle (spezifizierten) Eingangsbelegungen nur einmal vor?

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

# Vollständigkeit und Widerspruchsfreiheit: Beispiel




- Zustand A, Vollständigkeit: $x_1 \vee \overline{x}_1 x_0 \vee \overline{x}_0 = 1$ (vollständig)
- Zustand A, Widerspruchsfreiheit: alle Paare testen, also:

  $x_1 \wedge \overline{x}_1 x_0 = 0$ (ok)

  $x_1 \wedge \overline{x}_1 = 0$ (ok)

  $\overline{x}_1 x_0 \wedge \overline{x}_1 \neq 0$ (für $x_1 = 0$ und $x_0 = 1$ beide Übergänge aktiv)

# Entwurf von Schaltwerken: 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
- Digitaluhr: BCD-Zähler
- ...

# Entwurf von Schaltwerken: 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 $\delta$- und $\lambda$-Schaltnetz
- anschließend Minimierung der Schaltnetze
- Realisierung (je 1 D-Flipflop pro Zustandsbit) und Test

# Entwurf von Schaltwerken: Ampel (Variante 1)

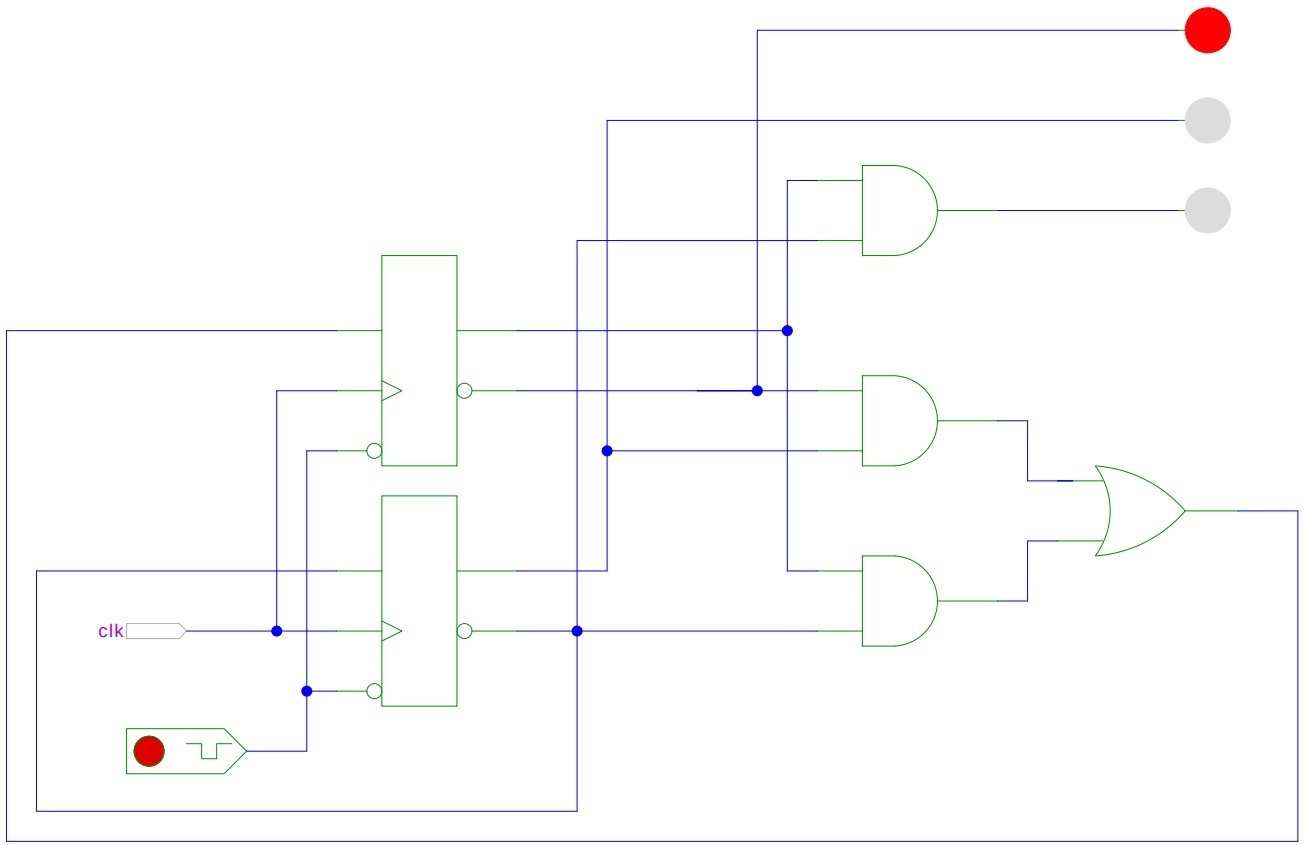
- vier Zustände, Codierung als 2-bit Vektor $(z_1, z_0)$
- Fluss- und Ausgangstafel für binäre Zustandscodierung

| 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

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

# Entwurf von Schaltwerken: Ampel (Variante 1)




(Hades Webdemos: 18-fsm/10-trafficlight/ampel_41)

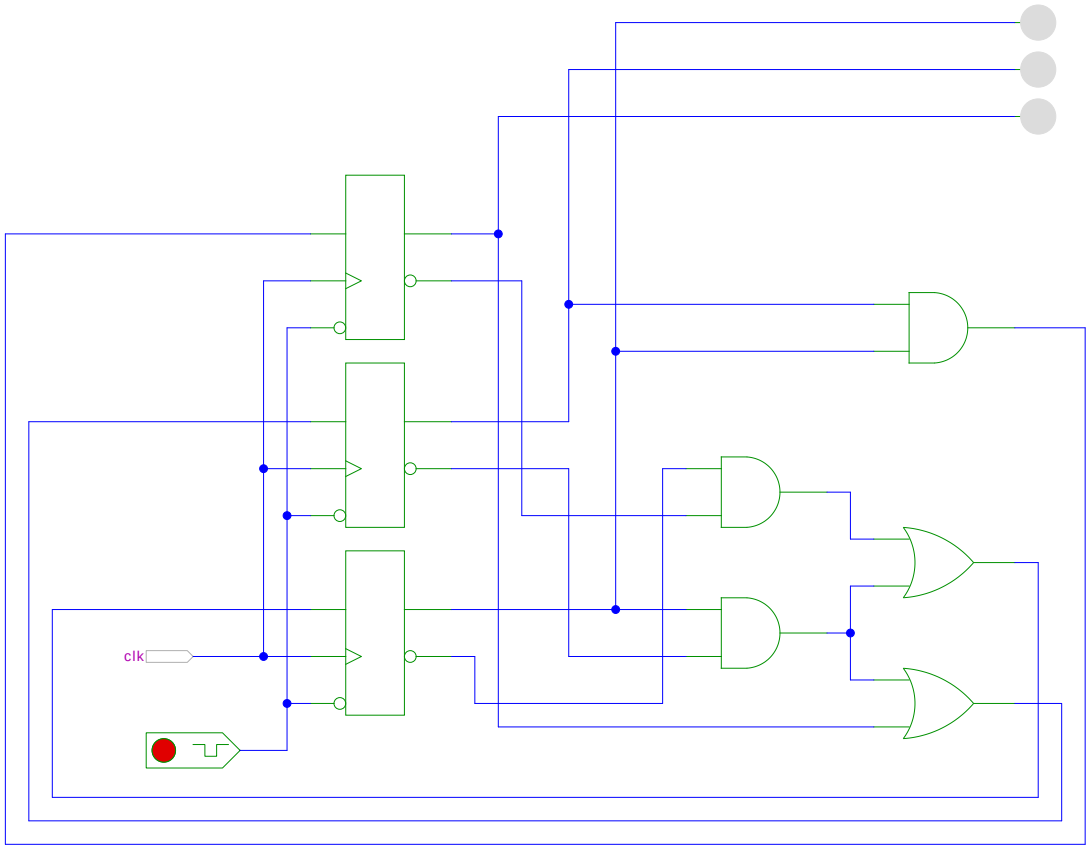
# Entwurf von Schaltwerken: Ampel (Variante 2)

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

| 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)

# Entwurf von Schaltwerken: Ampel (Variante 2)




(Hades Webdemos: 18-fsm/10-trafficlight/ampel_42)

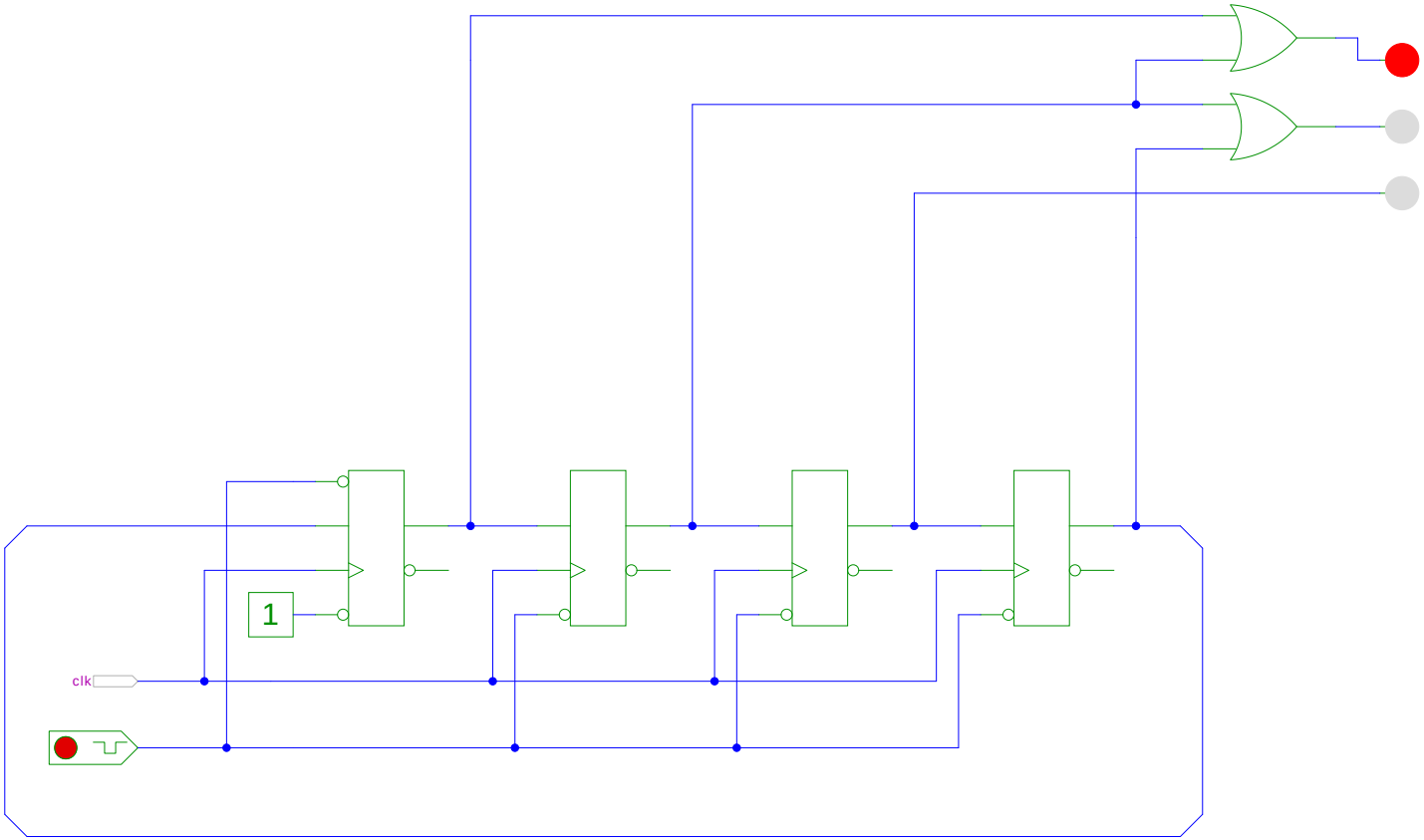
# Entwurf von Schaltwerken: Ampel (Variante 3)

- 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 $\delta$ minimal (Automat sehr schnell)
- Ausgangsfunktion $\lambda$ sehr einfach

# Entwurf von Schaltwerken: Ampel (Variante 3)




(Hades Webdemos: 18-fsm/10-trafficlight/ampel_44)

# Entwurf von Schaltwerken: Ampel (Zusammenfassung)

- viele Möglichkeiten der Zustandscodierung

- 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

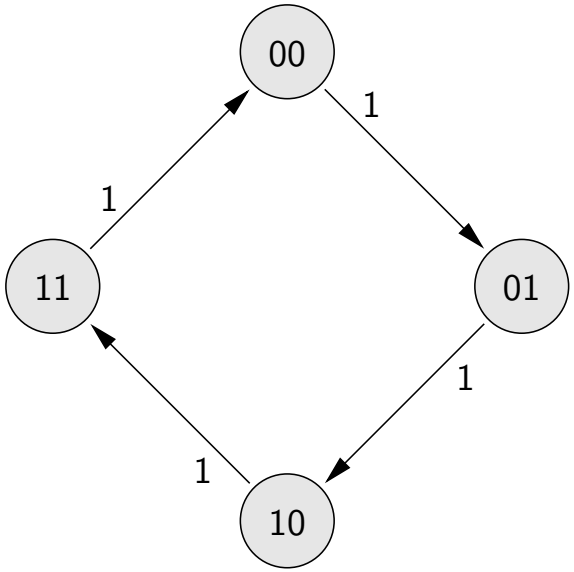
# Zählschaltungen

- diverse Beispiele für Zählschaltungen

- Zustandsdiagramme und Flusstafeln
- Schaltbilder

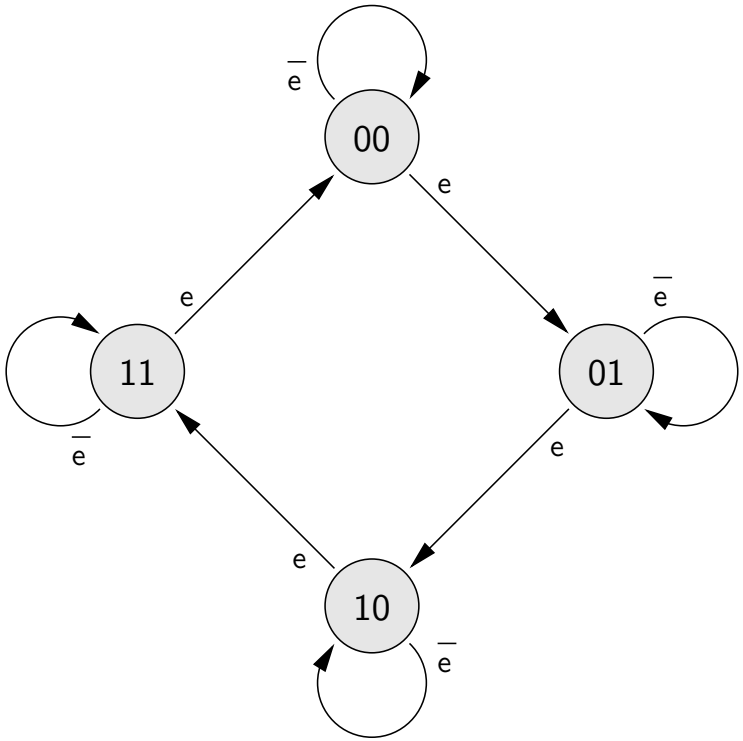
- *n*-bit Vorwärtszähler
- *n*-bit Zähler mit Stop und/oder Reset
- Vorwärts/Rückwärtszähler
- synchrone und asynchrone Zähler
- Beispiel: Digitaluhr (BCD-Zähler)

# 2-bit Zähler: Zustandsdiagramm



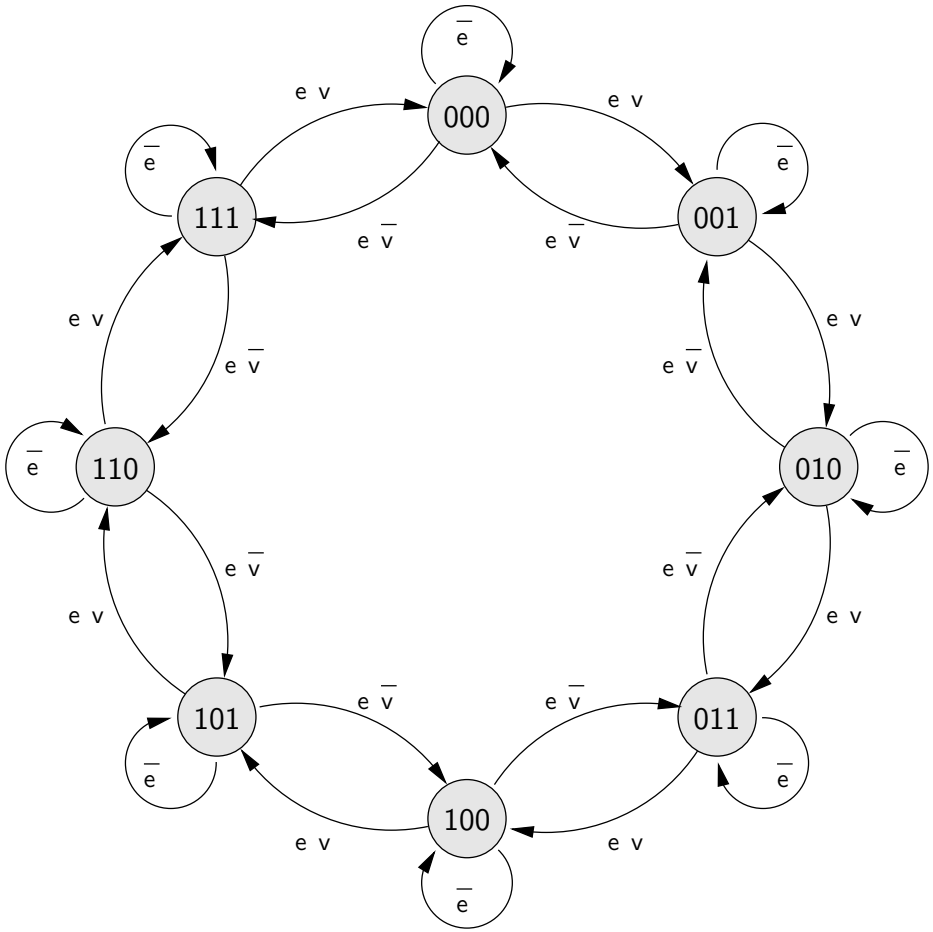

- Zähler als „trivialer“ endlicher Automat

# 2-bit Zähler mit Enable: Zustandsdiagramm und Flusstafel



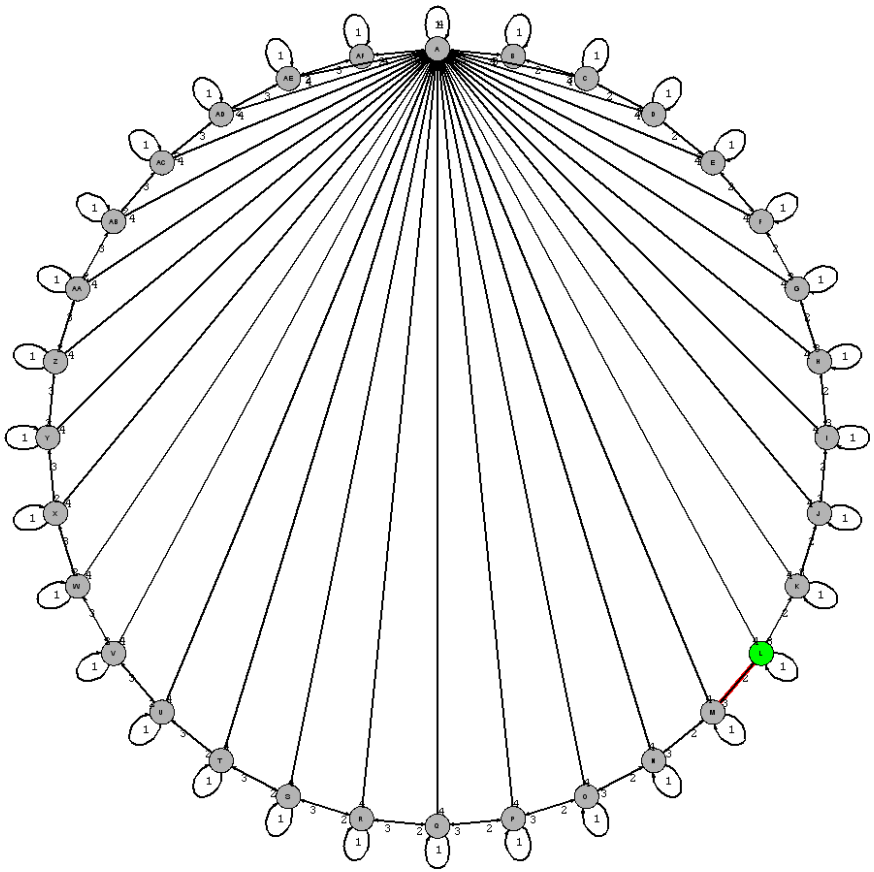

| | $e$ | $\overline{e}$ |
|---|---|---|
| Zustand | Folgezustand | |
| 00 | 01 | 00 |
| 01 | 10 | 01 |
| 10 | 11 | 10 |
| 11 | 00 | 11 |

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




| | $e\,v$ | $e\,\overline{v}$ | $\overline{e}\,*$ |
|---|---|---|---|
| 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 |

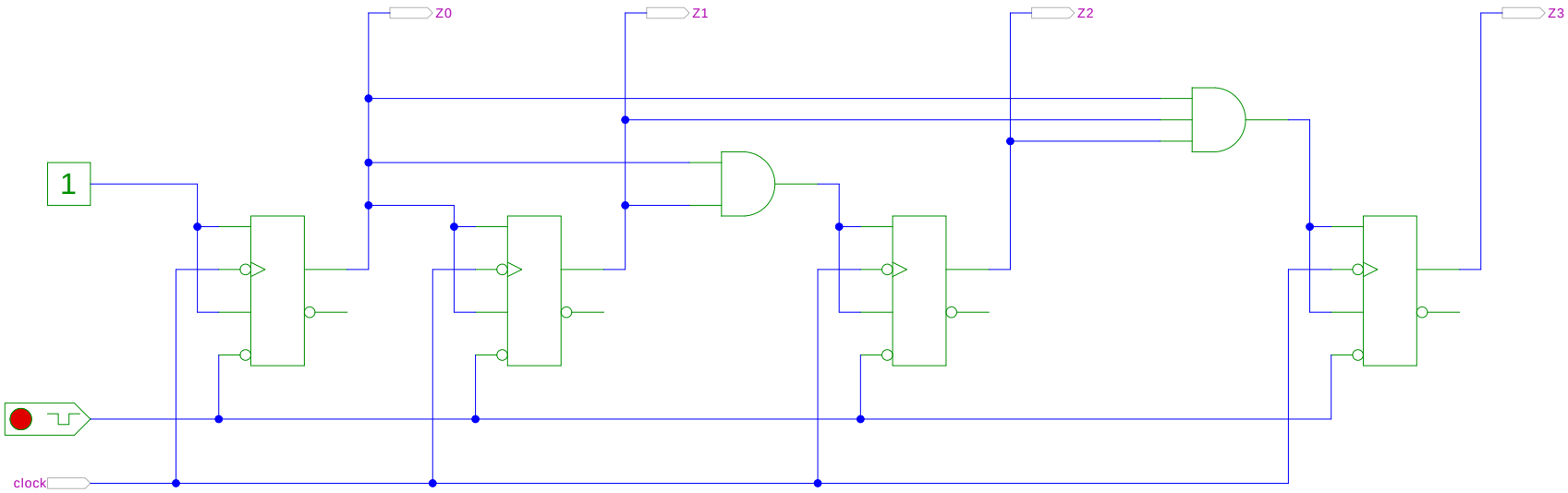
# 5-bit Zähler mit Reset: Zustandsdiagramm und Flusstafel



| 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 |

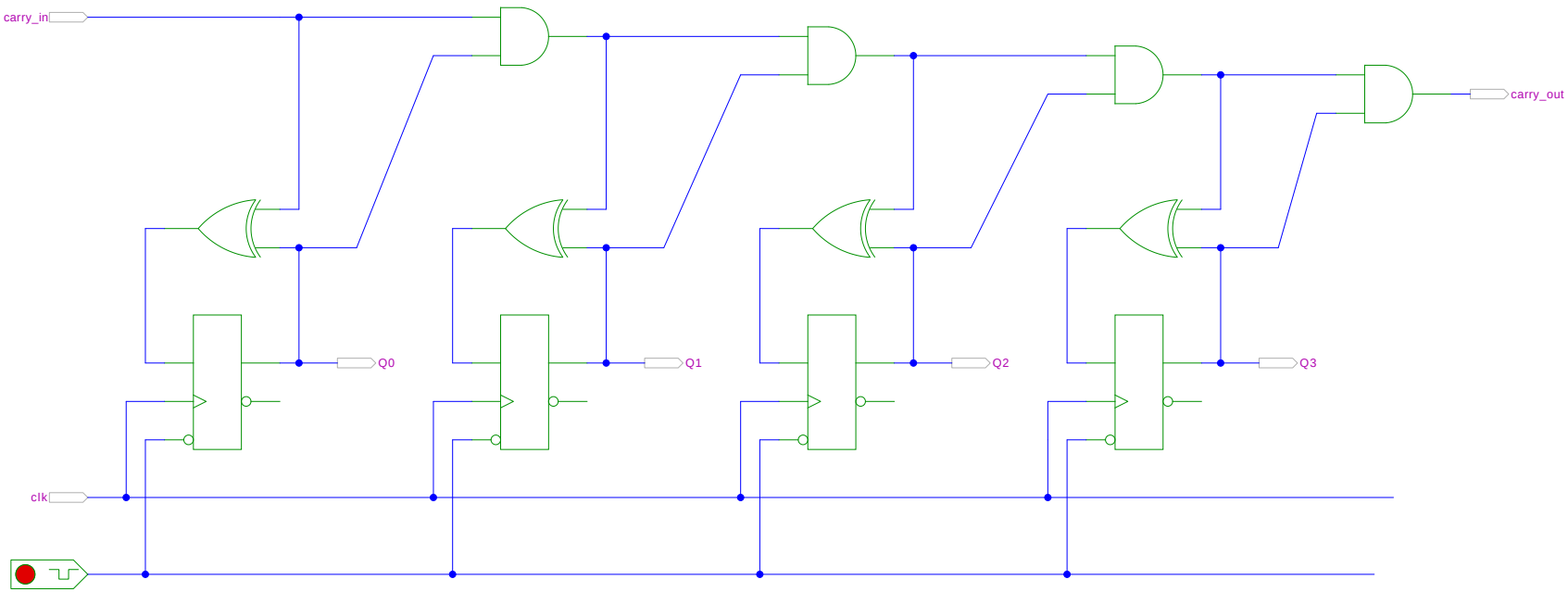
- Eingabe 1: stop, 2: zählen, 3: rückwärts zählen, 4: Reset nach A

# 4-bit Binärzähler mit JK-Flipflops



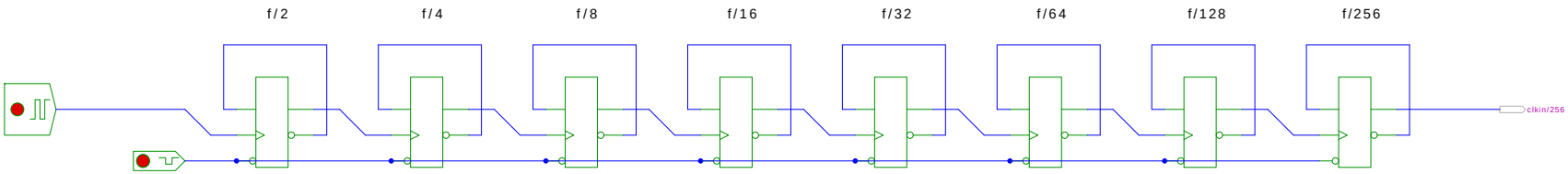

- $J_0 = K_0 = 1$: Ausgang $z_0$ wechselt bei jedem Takt
- $J_i = K_i = (z_0 z_1 \ldots z_{i-1})$:
  AusgangAusgang $z_i$ wechselt, wenn alle niederen Stufen 1 sind

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




- $D_0 = Q_0 \oplus c_{in}$: wechselt beim Takt, wenn $c_{in}$ aktiv ist
- $D_i = Q_i \oplus (c_{in} Q_0 Q_1 \ldots Q_{i-1})$
  wechselt, wenn alle niederen Stufen und Carry-In $c_{in}$ 1 sind

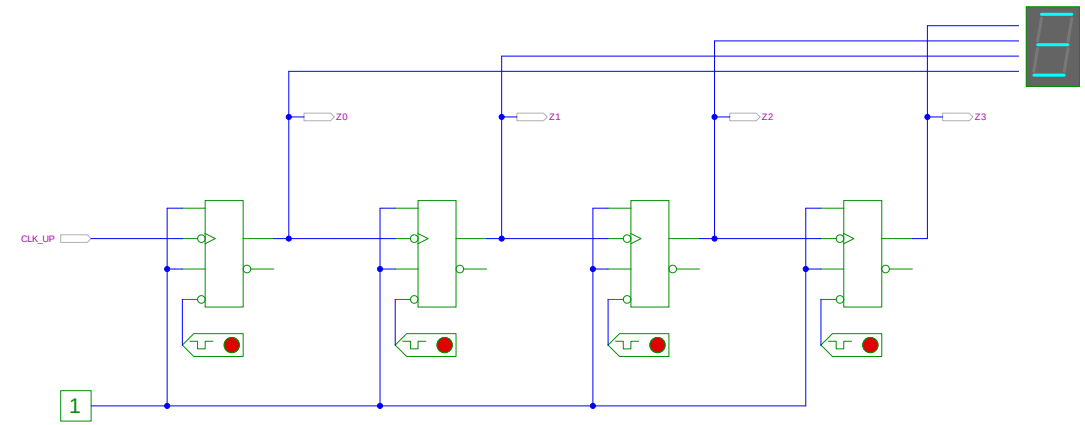
# Asynchroner $n$-bit Zähler/Teiler mit D-Flipflops

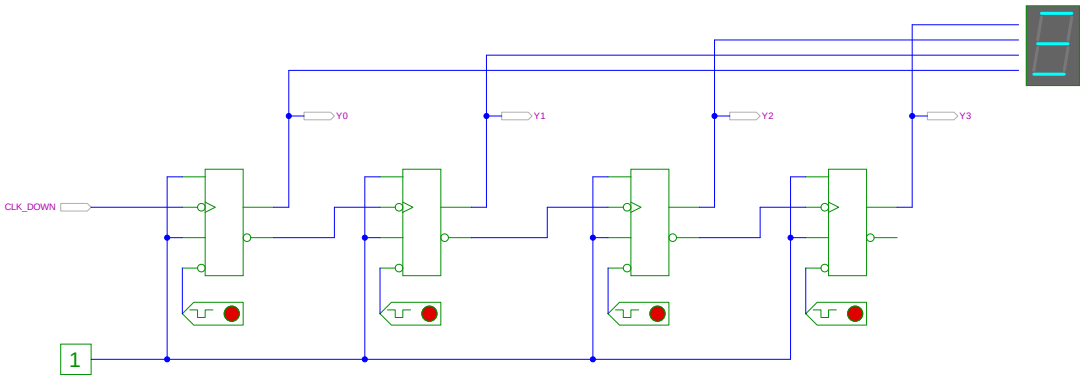



- $D_i = \overline{Q}_i$: jedes Flipflop wechselt bei seinem Taktimpuls
- Takteingang $C_0$ treibt nur das vorderste Flipflop
- $C_i = Q_{i-1}$: Ausgang der Vorgängerstufe als Takt von Stufe $i$

- erstes Flipflop wechselt bei jedem Takt (Zählrate $C_0/2$), zweites Flipflop bei jedem zweiten Takt (Zählrate $C_0/4$), $n$-tes Flipflop bei jedem $n$-ten Takt (Zählrate $c_0/2^n$)
- sehr hohe maximale Taktrate
- aber Flipflops schalten nacheinander, nicht gleichzeitig

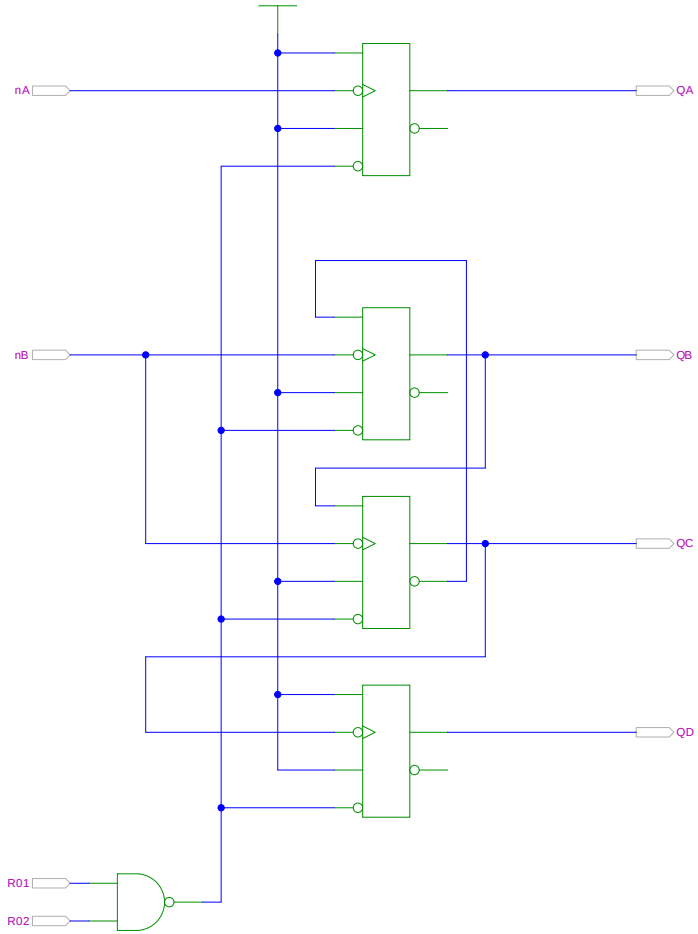
# Asynchrone 4-bit Vorwärts- und Rückwärtszähler



# 4-bit 1:12-Teiler mit JK-Flipflops: 7492

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



# 4-bit Vorwärts-Rückwärtszähler mit JK-Flipflops

- Inputs:

  `Up/nDown`

  `Enable`

  `nClk`

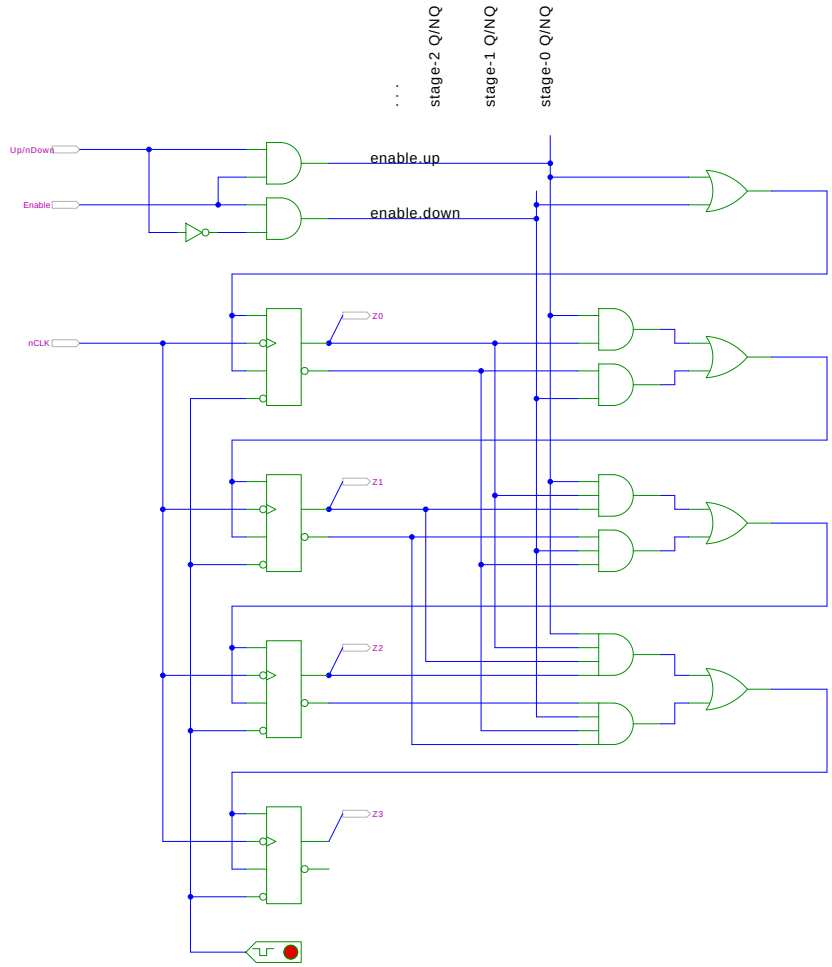
- Umschaltung der *Carry-Chain*:

  up:

  $J_i = K_i = (EQ_0Q_1 \dots Q_{i-1})$

  down:

  $J_i = K_i = (E\overline{Q}_0\overline{Q}_1 \dots \overline{Q}_{i-1})$

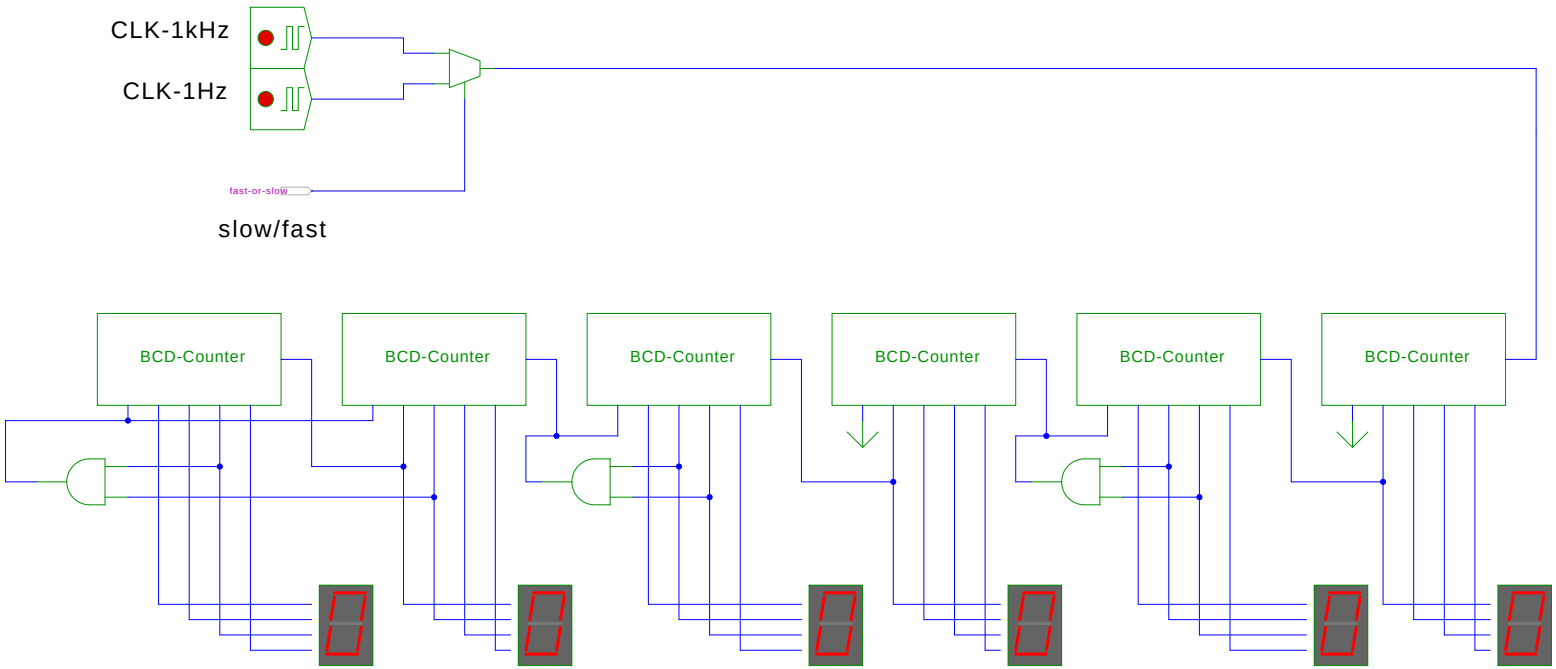


# Schaltwerke: weitere Beispiele

- Digital-Uhr
- DCF-77 Funk-Uhr
- Multiplex-Siebensegment-Anzeige

- Asynchrone Schaltungen: C-Gate und Micropipeline
- ...

# Digitaluhr mit BCD-Zählern




- Stunden Minuten Sekunden (hh:mm:ss)
- jeder BCD-Zähler mit Takt (rechts) und Reset (links unten)
- Übertrag Einer auf Zehner jeweils beim Übergang $9 \rightarrow 0$
- Übertrag und Reset der Zehner beim Auftreten des Wertes 6

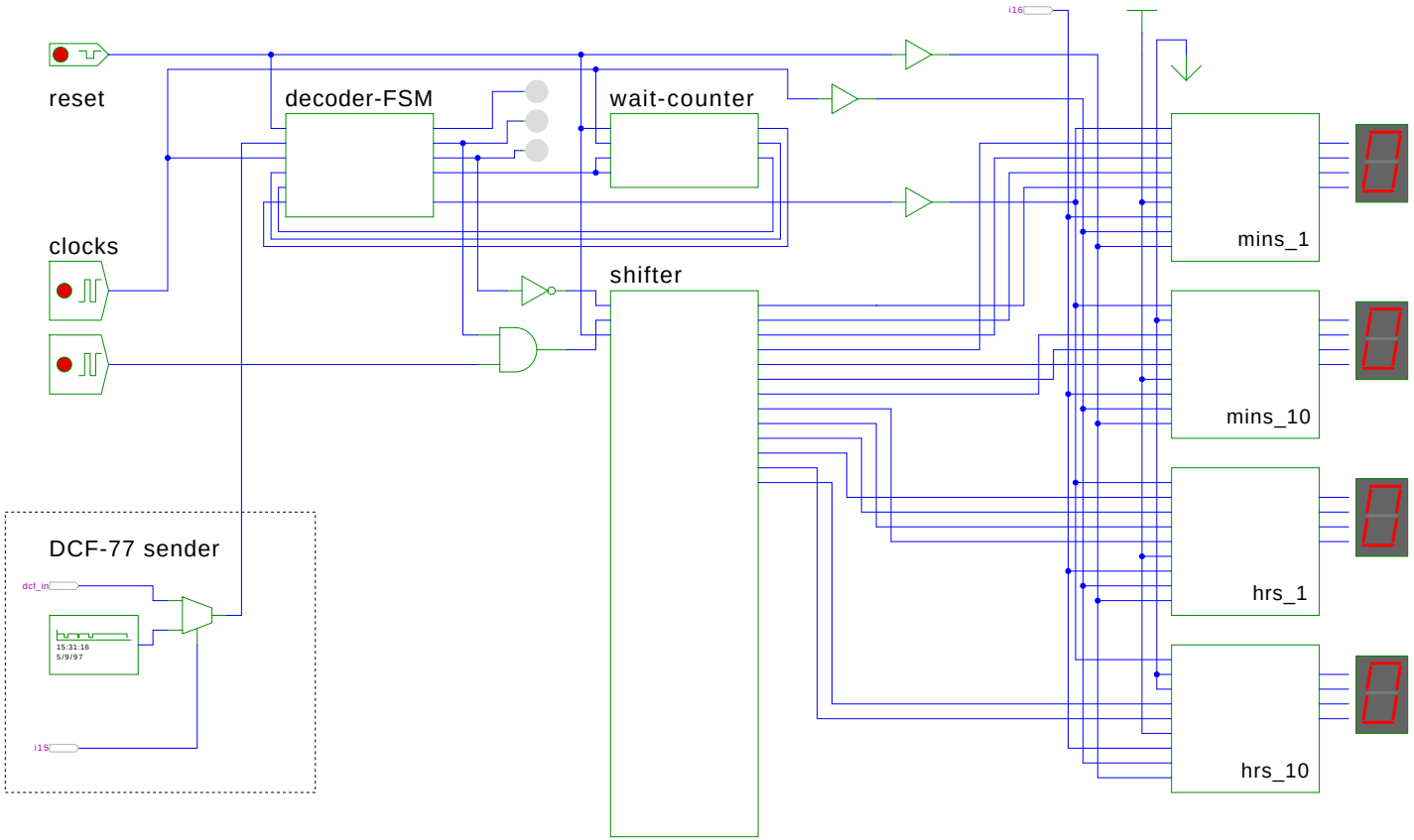
# Funkgesteuerte DCF-77 Uhr

- Beispiel für eine komplexe Schaltung aus mehreren einfachen Komponenten

- Decodierung des DCF-77 Zeitsignals, Sender nahe Frankfurt
- Langwelle 77 kHz, ganz Deutschland abgedeckt
- pro Sekunde wird ein Bit übertragen, als Puls mit abgesenktem Signalpegel („Amplitudenmodulation“)
- Pulslänge 100 ms entspricht Null, 200 ms entspricht Eins
- 59 Bits pro Sekunde (u.a. hh:mm:ss, Parität, Schaltjahr)
- fehlender 60 ter Puls markiert Ende einer Minute
- Decodierung der Bits mit entsprechend entworfenem Schaltwerk

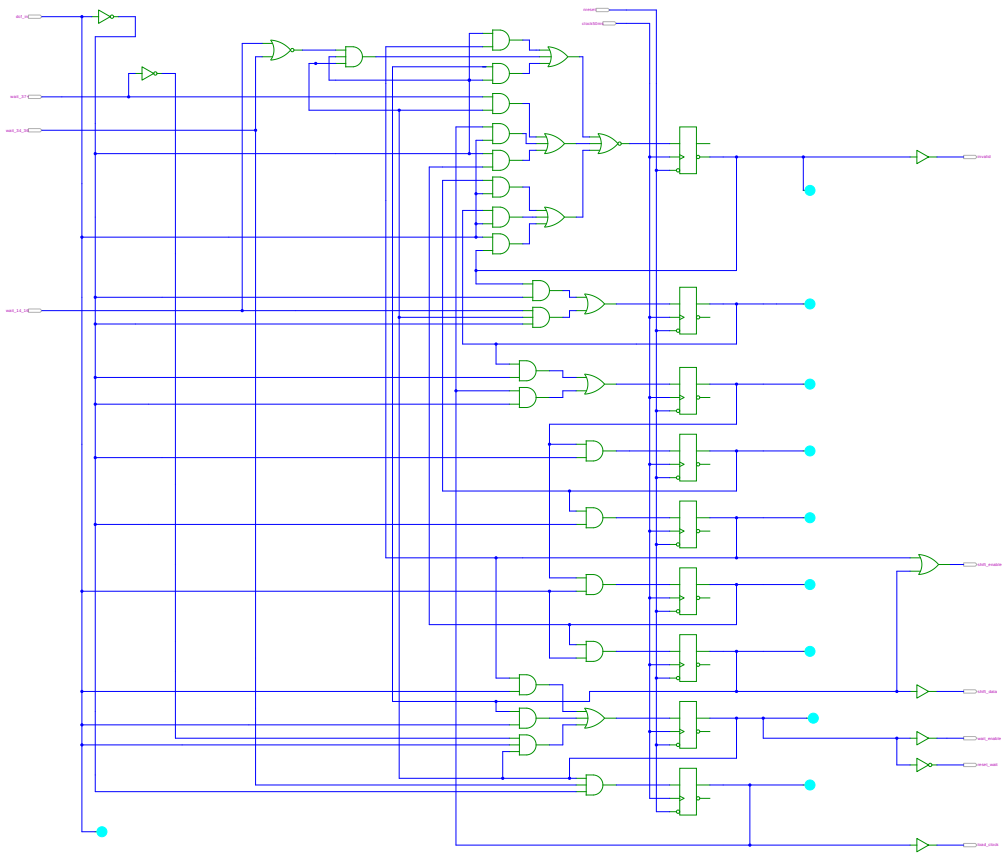
(u.a. http://de.wikipedia.org/wiki/DCF_77)

# Funkgesteuerte DCF-77 Uhr: Gesamtsystem




(Hades Webdemos: 45-misc/80-dcf77/dcf77)

# Funkgesteuerte DCF-77 Uhr: Decoder-Schaltwerk



(Hades Webdemos: 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

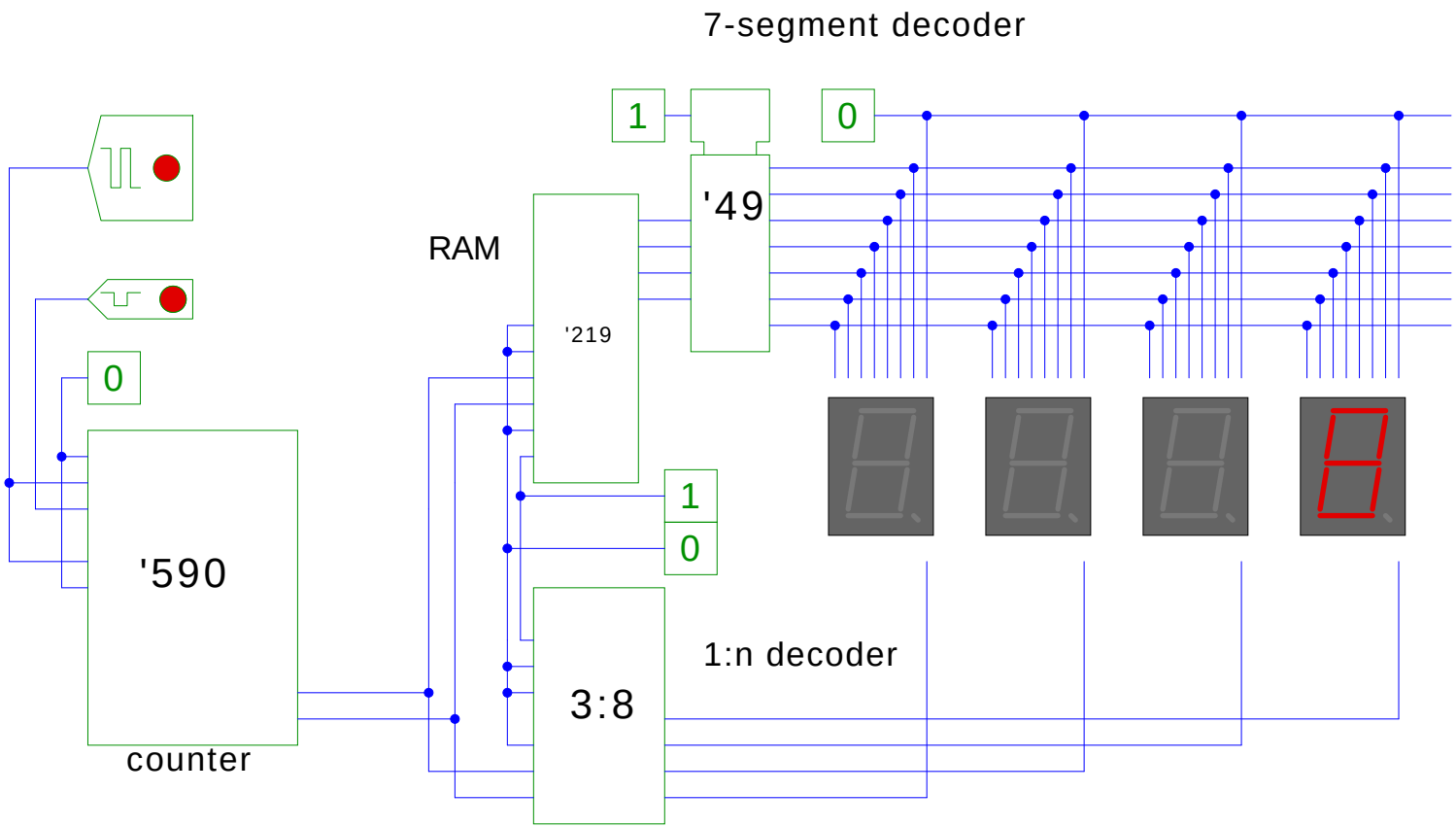
- die Anzeigen werden nacheinander nur ganz kurz eingeschaltet
- ein gemeinsamer Siebensegment-Decoder, Eingabe wird entsprechend der aktiven Ziffer umgeschaltet
- das Auge sieht die leuchtenden Segmente und „mittelt“
- ab ca. 100 Hz Frequenz erscheint die Anzeige ruhig

# Multiplex-Siebensegment-Anzeige

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




(Hades Webdemos: 45-misc/50-displays/multiplexed-display)

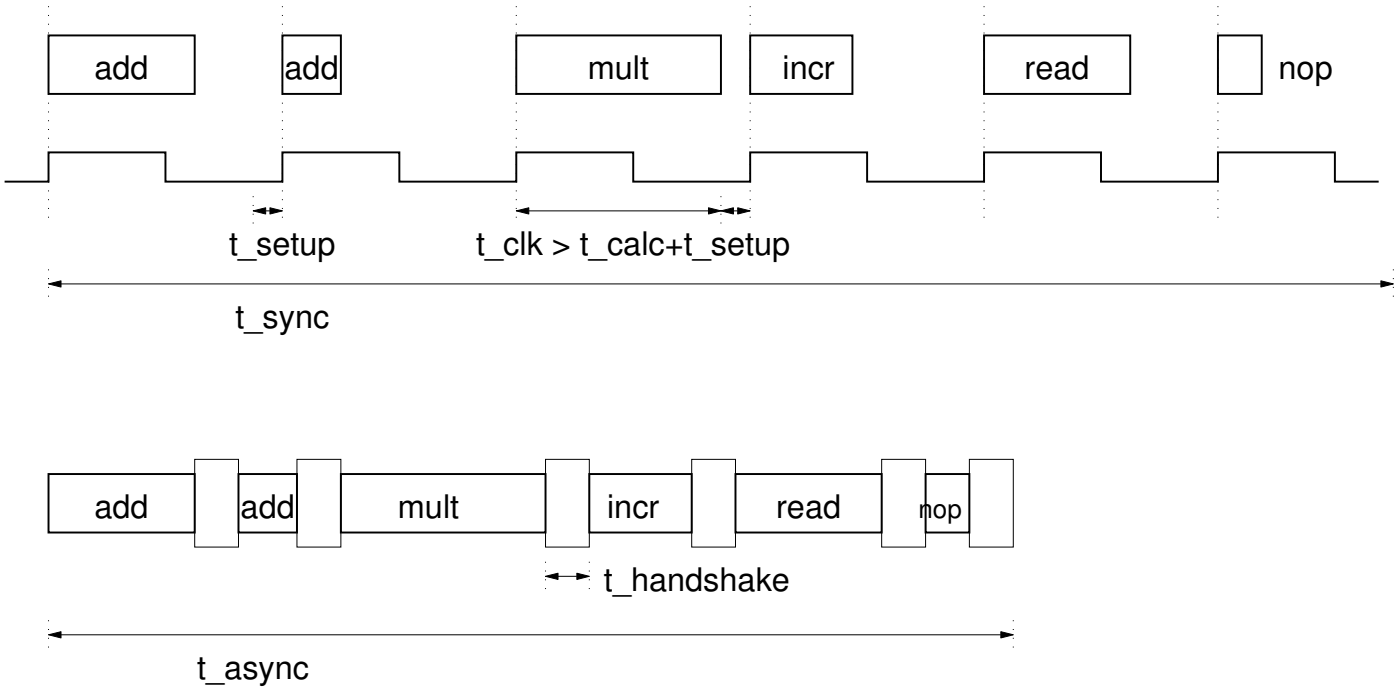
# Ausblick: Asynchrone Schaltungen

- 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, sobald Ergebnisse verfügbar
- kein zentraler Takt notwendig: so schnell wie möglich
- aber Probleme mit Deadlocks und Initialisierung

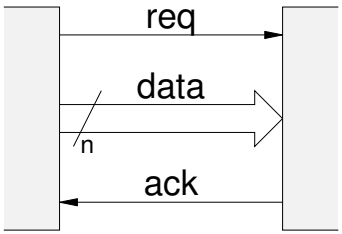
# Asynchrone Schaltungen: Performance



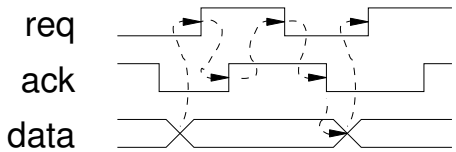

- synchron: Pipelining / Path-Balancing können Verschnitt verringern
- asynchron: Operationen langsamer wegen „completion detection“

# Zwei-Phasen und Vier-Phasen Handshake

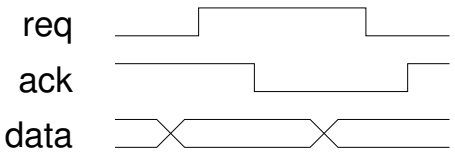
bundled data



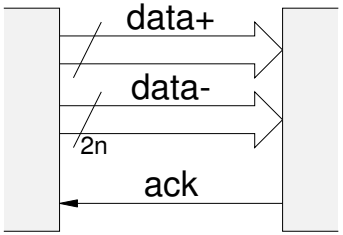

four-phase "level"



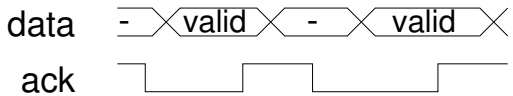

two-phase "edge"




dual rail




four-phase

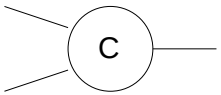



| | d+ | d- |
|---|---|---|
| empty | 0 | 0 |
| valid "0" | 0 | 1 |
| valid "1" | 1 | 0 |
| unused | 1 | 1 |

# Muller C-Gate: 2-Eingänge

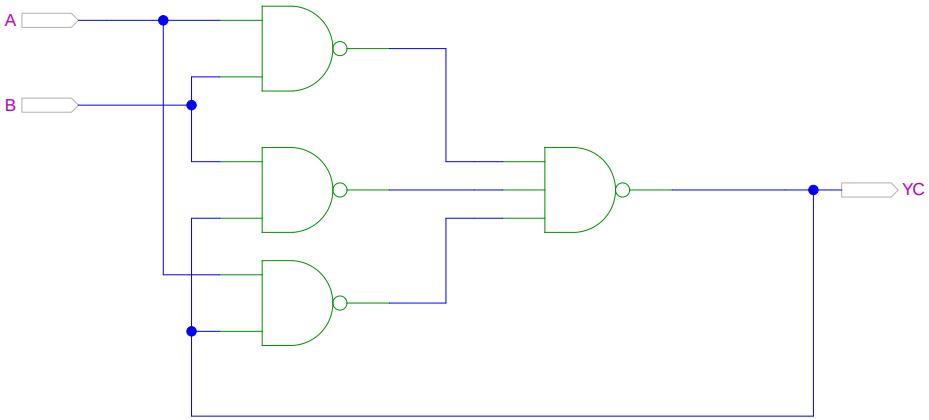
## Muller C-gate

- output changes to 0, if all inputs are 0
- output changes to 1, if all inputs are 1
- often used in asynchronous circuits

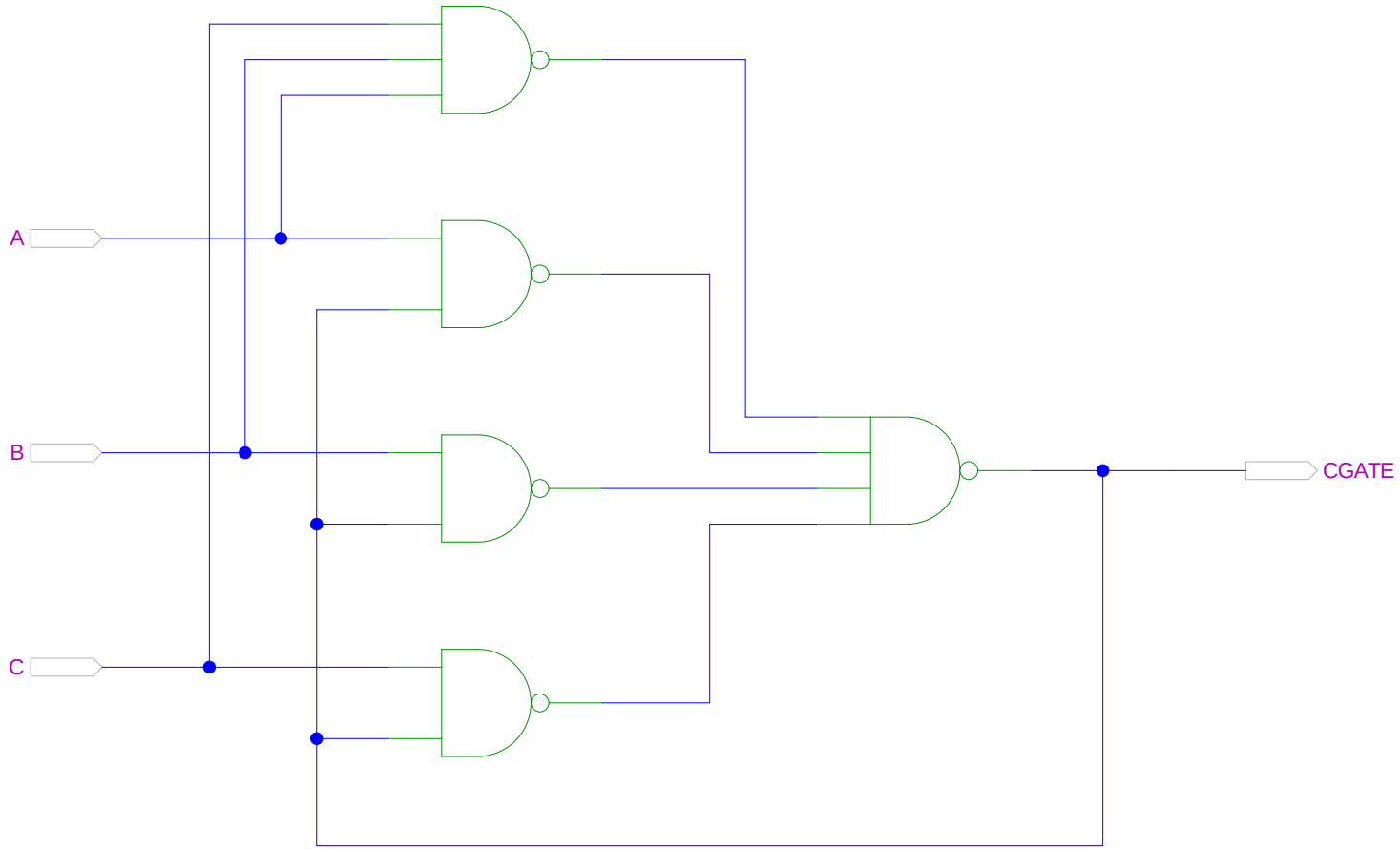



$c = ab + ac + bc$

| c \ ab | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |



# Muller C-Gate: 3-Eingänge



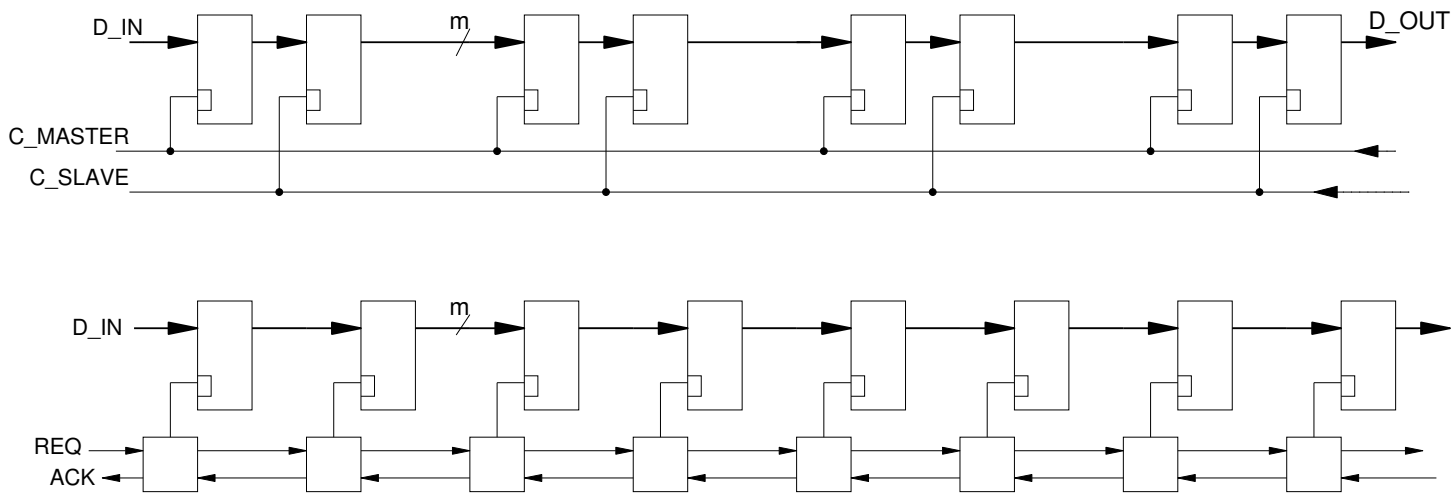
# Asynchrone Schaltungen: Micropipeline

- 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

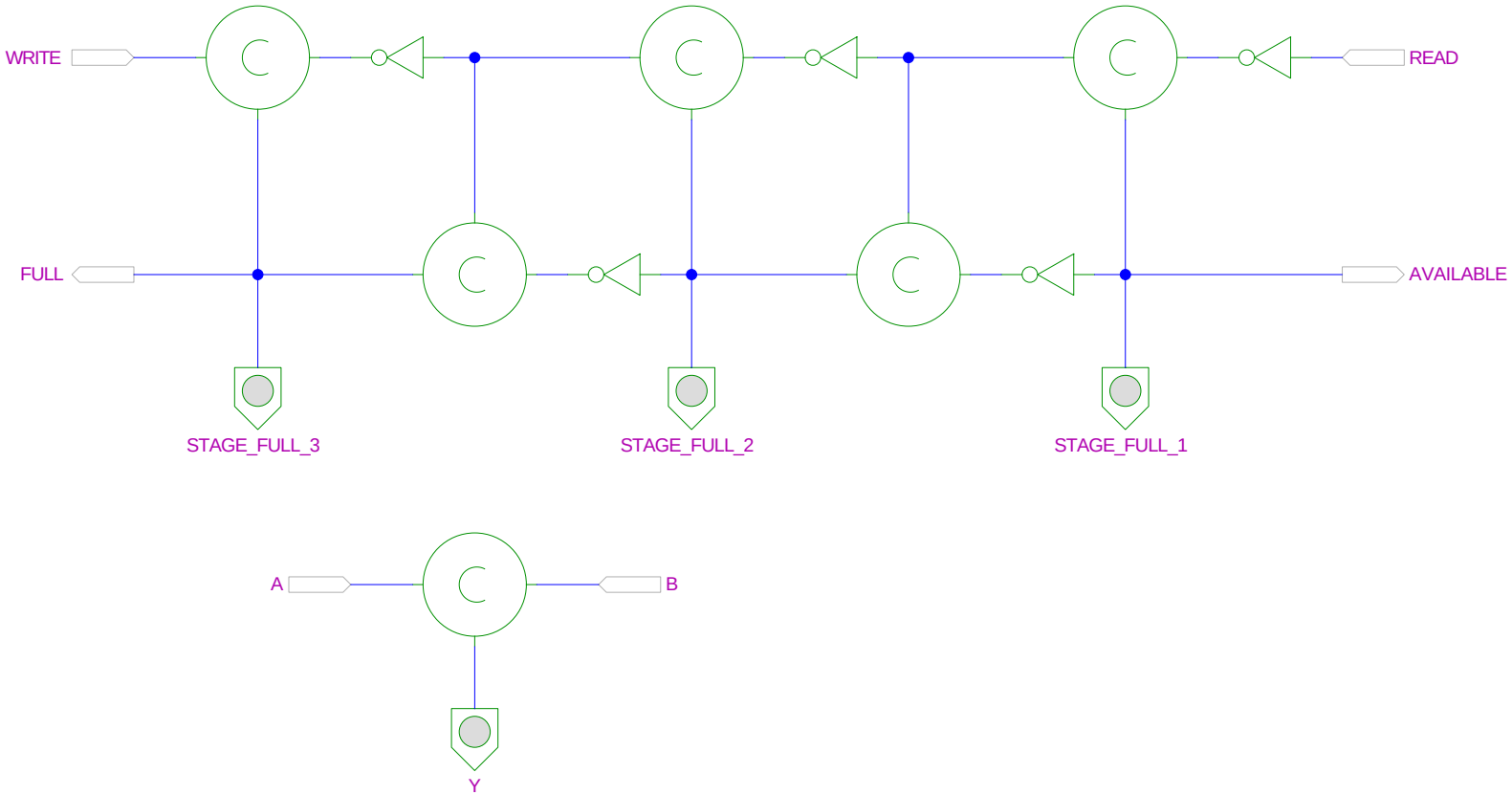
# Micropipeline: Konzept




*n*-stufige Micropipeline vs. getaktetes Schieberegister:

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

# Micropipeline: Demo mit C-Gates




(Hades Webdemos, 16-flipflops/80-micropipeline)

# Literatur: Vertiefung


- David Harel,
  *Statecharts, A visual formalism for complex systems*,
  CS84-05, Department of Applied Mathematics,
  The Weizmann Institute of Science, 1984.

- N.E.H. Weste & K. Eshragian,
  *Principles of CMOS VLSI Design — A Systems Perspective*,
  Addison-Wesley Publishing, 1993

# Literatur: Tools

- Klaus von der Heide, *Vorlesung Technische Informatik T1*, Universität Hamburg, FB Informatik, 2004
  tams-www.informatik.uni-hamburg.de/lectures/2004ws/vorlesungen/t1

- Norman Hendrich,
  *Hamburg Design System*,
  tams-www.informatik.uni-hamburg.de/applets/hades/