

## Rechnerstrukturen WS 2012/13

- ▶ Programmierbare Bausteine
  - ▶ Einleitung (Wiederholung)
  - ▶ Einsatz von PLAs
- ▶ Sequenzielle Schaltungen
  - ▶ Einleitung
  - ▶ Modellierung mit Automaten
- ▶ Synchrone Schaltwerke
  - ▶ Einleitung
  - ▶ Flip-Flops

Hinweis: Folien teilweise a. d. Basis von Materialien von Thomas Jansen

19. November 2012

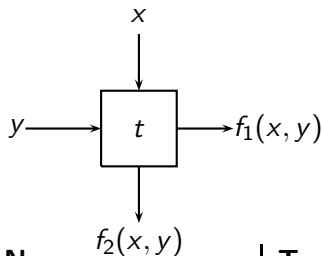
## Realisierungsmöglichkeit für Schaltnetze

- ▶ massenhaft produzierte
- ▶ darum **preisgünstige**
- ▶ nach der Fertigstellung in ihrer Funktion noch beeinflussbare
- ▶ **funktional vollständige**
- ▶ also **universelle** Standardbausteine

### Programmable Logic Array (PLA)

**Varianten** PAL, PROM, FPGA, ...  
(zum Teil eingeschränkte Funktionalität)

## PLA Grundbausteine



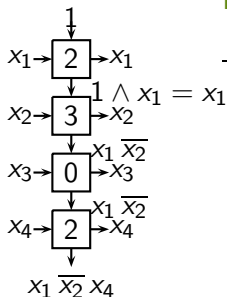
Name	Typ	$f_1(x, y)$	$f_2(x, y)$
Identer	0	$y$	$x$
Addierer	1	$x \vee y$	$x$
Multiplizierer	2	$y$	$x y$
Negat-Multiplizierer	3	$y$	$x \bar{y}$

klar funktional vollständig

## PLA: Monomrealisierung

Beispiel Monom  $x_1 \overline{x_2} x_4$

Erinnerung



Name	Typ	$f_r(o, l)$	$f_u(o, l)$
Identer	0	$/$	$o$
Addierer	1	$o \vee l$	$o$
Multiplizierer	2	$/$	$o /$
Negat-Multiplizierer	3	$/$	$o \bar{l}$

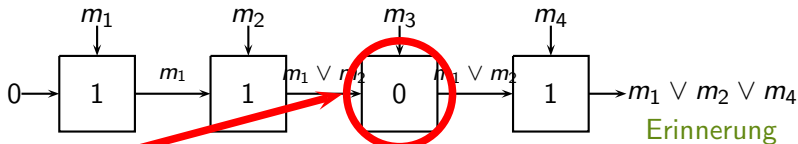
also jedes Monom leicht realisierbar

- ▶ falls Variable fehlt Typ 0
- ▶ falls  $x_i$  vorkommt Typ 2
- ▶ falls  $\overline{x_i}$  vorkommt Typ 3

## PLA: Polynomrealisierung

gesehen für  $f: \{0, 1\}^n \rightarrow \{0, 1\}$   
 $k$  verschiedene Monome  $m_1, m_2, \dots, m_k$   
in  $n$  Zeilen und  $k$  Spalten realisierbar

Wie können wir  $f$  realisieren, z. B.  $f = m_1 \vee m_2 \vee m_4$ ?



Ist das sinnvoll?

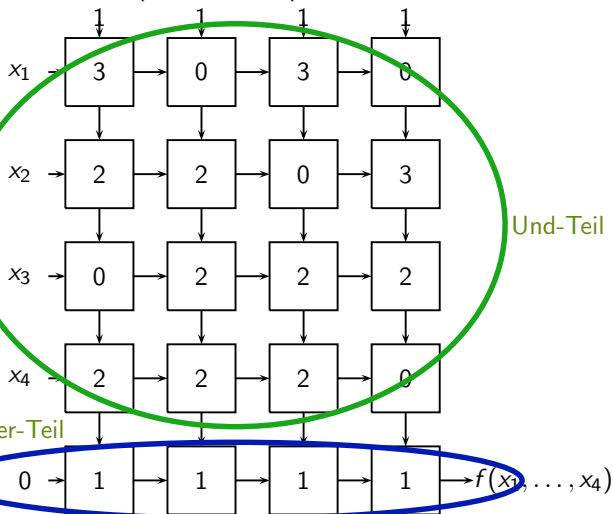
klar für  
 $f: \{0, 1\}^n \rightarrow \{0, 1\}$

nicht  
aber für  
 $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$   
schon

Name	Typ	$f_r(o, l)$	$f_u(o, l)$
Identer	0	$l$	$o$
Addierer	1	$o \vee l$	$o$
Multiplizierer	2	$l$	$o l$
Negat-Multiplizierer	3	$l$	$o \bar{l}$

## PLA: Ein konkretes Beispiel

Beispiel  $f(x_1, x_2, x_3, x_4) = \overline{x_1} x_2 x_4 \vee x_2 x_3 x_4 \vee \overline{x_1} x_3 x_4 \vee \overline{x_2} x_3$



## Fazit zur PLA-Nutzung

also Wir können jede Funktion  $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$ ,  
für deren Polynom insgesamt  $k$  Implikanten ausreichen,  
mit einem PLA mit  $n + m$  Zeilen und  $k$  Spalten realisieren.

klar Wir wünschen uns  $k$  klein.

dafür Minimalpolynome

Wie findet man Minimalpolynome für Funktionen  
 $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$ ?

## Minimalpolynome für $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$

**Notation** statt  $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$   
 $(f_1, f_2, \dots, f_m)$  mit  $f_i: \{0, 1\}^n \rightarrow \{0, 1\}$   
 für  $i \in \{1, 2, \dots, m\}$

**Definition** Ein **Minimalpolynom** für  $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$   
 ist  $(p_1, p_2, \dots, p_m)$  mit minimalen Kosten,  
 dabei ist  $p_i$  Polynom für  $f_i$ . Bei den Kosten  
 zählen mehrfach vorkommende Monome nur einmal.

Also suchen wir Minimalpolynome  $p_i$  für  $f_i$ ? **Nein!**

**Beispiel**  $p_1(x_1, x_2, x_3) = x_1 x_3 \vee \overline{x_2} x_3$  ist Minimalpolynom  
 $p_2(x_1, x_2, x_3) = \overline{x_1} x_2 \vee x_2 x_3$  ist Minimalpolynom

$p'_1(x_1, x_2, x_3) = x_1 x_2 x_3 \vee \overline{x_2} x_3$  stellt auch  $p_1$  dar  
 $p'_2(x_1, x_2, x_3) = \overline{x_1} x_2 \vee x_1 x_2 x_3$  stellt auch  $p_2$  dar

Gesamtkosten  $(p_1, p_2) = (n + m) \times k = (3 + 2) \times 4$

Gesamtkosten  $(p'_1, p'_2) = (n + m) \times k' = (3 + 2) \times 3$



## Monome für Minimalpolynome für $f: \{0, 1\}^n \rightarrow \{0, 1\}^k$

Welche Monome übernehmen die Rolle der Primimplikanten?

**Definition** Ein Monom  $m$  ist ein **multipler Primimplikant** von  $f = (f_1, f_2, \dots, f_k)$  mit  $f_i: \{0, 1\}^n \rightarrow \{0, 1\}$ , wenn  $m$  Primimplikant von  $\bigwedge_{i \in I} f_i$  ist für eine nicht-leere Menge  $I \subseteq \{1, 2, \dots, k\}$ .

**Theorem** Minimalpolynome für  $f: \{0, 1\}^n \rightarrow \{0, 1\}^k$  enthalten nur multiple Primimplikanten von  $f$ .

**Beweis** und **Algorithmus** zur Berechnung  $\rightarrow$  Skript

## Multiple Primimplikanten berechnen

Wie finden wir alle multiplen Primimplikanten?

**Theorem**  $m \in \text{PI} \left( \bigwedge_{i \in I} f_i \right) \Rightarrow \forall i \in I: \exists m_i \in \text{PI}(f_i): m = \bigwedge_{i \in I} m_i$

**Beweis**

klar  $m \in \text{PI} \left( \bigwedge_{i \in I} f_i \right)$  ist für alle  $i \in I$  Implikant von  $f_i$

Sei  $m_i \in \text{PI}(f_i)$  Verkürzung von  $m$  ( $i \in I$ )

klar  $\bigwedge_{i \in I} m_i$  ist Verkürzung von  $m$

weil  $m \in \text{PI} \left( \bigwedge_{i \in I} f_i \right)$ , kann  $\bigwedge_{i \in I} m_i$  keine echte Verkürzung von  $m$  sein

also  $\bigwedge_{i \in I} m_i = m$



## Berechnung aller multipler Primimplikanten

### Algorithmus

berechnet für  $I \subseteq \{1, 2, \dots, k\}$  die Menge  $M_I$  der multiplen Primimplikanten  $m = \bigwedge_{i \in I} m_i$

1. Für  $i \in \{1, 2, \dots, k\}$  berechne  $M_{\{i\}} := \text{PI}(f_i)$ .
2.  $M := \bigcup_{i=1}^k \{\{i\}\}$ ;  $M' := \bigcup_{i=1}^k M_{\{i\}}$ .
3. Wiederhole
4. Für  $I, J \in M$  mit  $I \cap J = \emptyset$
5.  $M_{I \cup J} := \{m \neq 0 \mid m = m' m'' \text{ mit } m' \in M_I, m'' \in M_J, \text{ keine echte Verkürzung von } m \text{ in } M_{I \cup J}\}$ .
6.  $M := M \cup \{I \cup J\}$ ;  $M' := M' \cup M_{I \cup J}$ .
7. So lange, bis  $M'$  eine Iteration unverändert bleibt.

## Minimalpolynomberechnung für $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$

### Algorithmus

1. Für alle  $f_i$  berechne alle Primimplikanten.
2. Berechne alle multiplen Primimplikanten.  
(ergeben sich potentiell als paarweise Verknüpfung "normaler" Primimplikanten)
3. Berechne eine möglichst günstige Überdeckung.

also günstigste Darstellung für PLA-Realisierungen  
mit uns bekannten Mitteln berechenbar  
allerdings **sehr aufwendig**

## PLA als ROM

**Aufgabe** Speichere  $2^n$  „Wörter“ der Länge  $m$ .

$$w_0 = w_{0,0} w_{0,1} w_{0,2} \cdots w_{0,m-1} \in \{0, 1\}^m$$

$$w_1 = w_{1,0} w_{1,1} w_{1,2} \cdots w_{1,m-1} \in \{0, 1\}^m$$

$$w_2 = w_{2,0} w_{2,1} w_{2,2} \cdots w_{2,m-1} \in \{0, 1\}^m$$

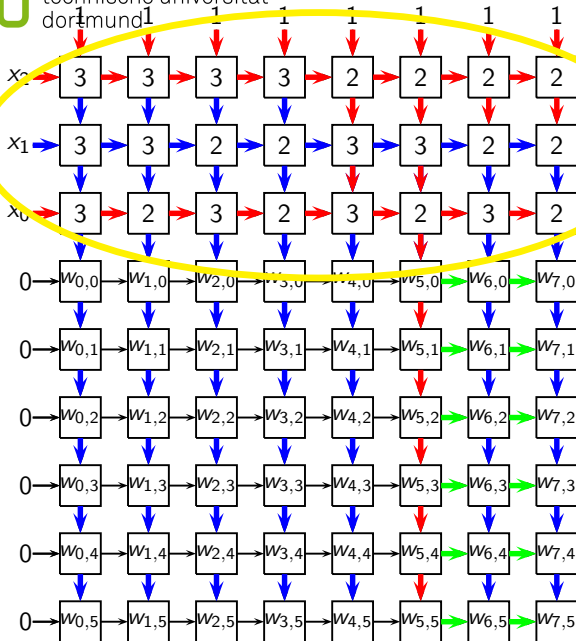
$\vdots$

$$w_{2^n-1} = w_{2^n-1,0} w_{2^n-1,1} w_{2^n-1,2} \cdots w_{2^n-1,m-1} \in \{0, 1\}^m$$

**Benutze** PLA mit  $n + m$  Zeilen,  $2^n$  Spalten

**Beobachtung**  $m \cdot 2^n$  Zellen für  $m \cdot 2^n$  zu speichernde Bits  
 mindestens erforderlich

**Adressierung** mit jeweils  $n$  Bits  
 in  $n$  zusätzlichen Zeilen



Beispiel

$n = 3, m = 6$

$2^3 = 8$  Wörter  
jeweils 6 Bits

Beobachtung

Und-Teil fest

nur von Größe  
abhängig

Anmerkung

PLAs mit festem  
Und-Teil werden  
als PROM  
verkauft.

## Zwischen-Fazit PLAs

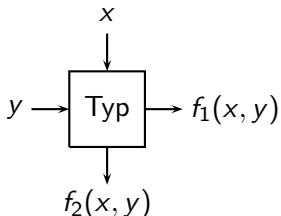
PLAs sind preiswerte, universelle Bausteine, die

- ▶ beliebige boolesche Funktionen leicht realisierbar machen,
- ▶ Minimalpolynomdarstellungen motivieren,
- ▶ Speicherung von  $2^n$  Wörtern der Länge  $m$  in einem  $(n + m) \times 2^n$ -PLA erlauben.

Nachteil nur einmal programmierbar

Wie kann man PLAs beliebig neu programmierbar machen?

## „Software“-PLAs



Typ	s	t	$f_1(x, y)$	$f_2(x, y)$
0	0	0	y	x
1	0	1	$x \vee y$	x
2	1	0	y	$x y$
3	1	1	y	$x \bar{y}$

**klar** vier verschiedene Typen, mit zwei Bits codierbar

**Idee** erweitere PLA-Baustein um zwei zusätzliche Eingaben, die den Baustein-Typ codieren

**jetzt**  $f_1$  und  $f_2$  als Funktionen von  $(s, t, x, y)$  darstellen

- ▶  $f_1(s, t, x, y) = y \vee \bar{s} t x$
- ▶  $f_2(s, t, x, y) = \bar{s} x \vee s x (t \oplus y)$



## Bemerkung zum Einsatz

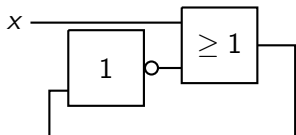
**klar** für ein  $n \times m$ -PLA werden zur Programmierung  $2nm$  Bits gebraucht.

**Beobachtung** Man kann diese  $2nm$  Bits gut in einem PROM speichern.

### Fazit

- ▶ einfache und günstige Realisierung von booleschen Funktionen
- ▶ besonders geeignet für kleine Stückzahlen
- ▶ besonders geeignet bei nur temporärem Gebrauch

## Sequenzielle Schaltungen

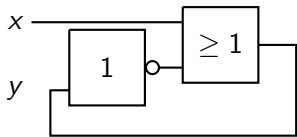


klar Das ist **kein** Schaltnetz.

allerdings Es ist eine „baubare“ Schaltung.

Was passiert in dieser Schaltung?

## Eine konkrete sequenzielle Schaltung



Was passiert in dieser Schaltung?

x	y	$x \vee \bar{y}$
0	0	1
0	1	0
1	0	1
1	1	1

klar und immer so weiter...

natürlich in der Realität viel schneller

darum heißt die Schaltung **Flimmerschaltung**

## Bewertung des Effekts

klar **Unkontrolliertes** Flimmern ist sehr **unschön**.

Also Kreise konsequent verbieten?

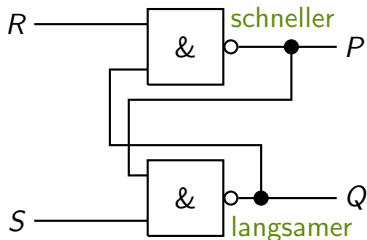
Wozu können Kreise gut sein?

**Beobachtung** Ausgänge werden zu Eingaben...

**etwas anders** Man kann schon Berechnetes noch einmal „sehen“.

**Einsicht** Das realisiert so etwas wie **Speicher**.

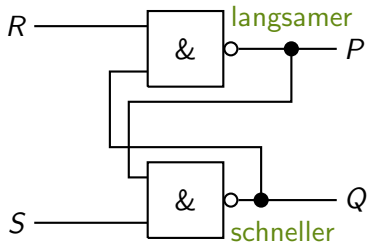
## Ein zweites Beispiel



$x$	$y$	$\overline{xy}$
0	0	1
0	1	1
1	0	1
1	1	0

$R_t$	$S_t$	$P_{t+\Delta}$	$Q_{t+\Delta}$	$P_{t+2\Delta}$	$Q_{t+2\Delta}$
0	0	1	1	1	1
0	1	1	0	1	0
1	0	$\overline{Q_t}$	1	0	1
1	1	$\overline{Q_t}$	$Q_t$	$\overline{Q_t}$	$Q_t$

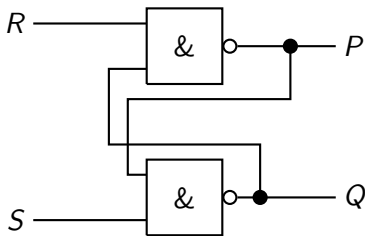
## Ein zweites Beispiel



x	y	$\overline{xy}$
0	0	1
0	1	1
1	0	1
1	1	0

$R_t$	$S_t$	$P_{t+\Delta}$	$Q_{t+\Delta}$	$P_{t+2\Delta}$	$Q_{t+2\Delta}$
0	0	1	1	1	1
0	1	1	$\overline{P_t}$	1	0
1	0	0	1	0	1
1	1	$P_t$	$\overline{P_t}$	$P_t$	$\overline{P_t}$

## Bi-stabile NAND-Kippstufe



Anmerkung  
heißt auch Latch

positiv kippt, flimmert nicht

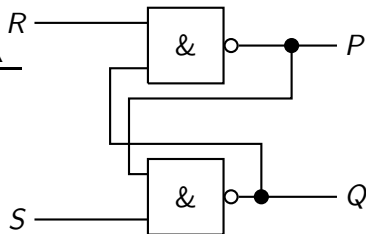
negativ Verhalten hängt von Schaltzeiten der beiden Gatter ab

genauer beobachtet Verhalten hängt manchmal von Schaltzeiten der beiden Gatter ab

## Analyse der bi-stabilen NAND-Kippstufe

1. Fall oberes NAND-Gatter schneller

$R_t$	$S_t$	$P_{t+\Delta}$	$Q_{t+\Delta}$	$P_{t+2\Delta}$	$Q_{t+2\Delta}$
0	0	1	1	1	1
0	1	1	0	1	0
1	0	$\overline{Q_t}$	1	0	1
1	1	$\overline{Q_t}$	$Q_t$	$\overline{Q_t}$	$Q_t$



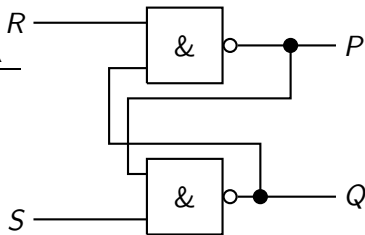
$$\begin{aligned}
 P_{t+2\Delta} &= \overline{R Q_{t+\Delta}} = \overline{R} \vee \overline{Q_{t+\Delta}} = \overline{R} \vee \overline{\overline{S P_{t+\Delta}}} \\
 &= \overline{R} \vee S P_{t+\Delta} = \overline{R} \vee S \overline{R Q_t} = \overline{R} \vee S (\overline{R} \vee \overline{Q_t}) \\
 &= \overline{R} \vee S \overline{R} \vee S \overline{Q_t} = \overline{R} \vee S \overline{Q_t}
 \end{aligned}$$



## Analyse der bi-stabilen NAND-Kippstufe

2. Fall unteres NAND-Gatter schneller

$R_t$	$S_t$	$P_{t+\Delta}$	$Q_{t+\Delta}$	$P_{t+2\Delta}$	$Q_{t+2\Delta}$
0	0	1	1	1	1
0	1	1	$\overline{P_t}$	1	0
1	0	0	1	0	1
1	1	$P_t$	$\overline{P_t}$	$P_t$	$\overline{P_t}$



$$\begin{aligned}
 P_{t+2\Delta} &= \overline{R Q_{t+2\Delta}} = \overline{R} \vee \overline{Q_{t+2\Delta}} = \overline{R} \vee \overline{\overline{S P_{t+\Delta}}} \\
 &= \overline{R} \vee S P_{t+\Delta} = \overline{R} \vee S \overline{R Q_{t+\Delta}} = \overline{R} \vee S (\overline{R} \vee \overline{Q_{t+\Delta}}) \\
 &= \overline{R} \vee S \overline{R} \vee S \overline{Q_{t+\Delta}} = \overline{R} \vee S \overline{Q_{t+\Delta}} = \overline{R} \vee S \overline{\overline{S P_t}} \\
 &= \overline{R} \vee S S P_t = \overline{R} \vee S P_t
 \end{aligned}$$

## Fazit zum Ausgang $P$ der bi-stabilen NAND-Kippstufe

1. Fall oberes NAND-Gatter schneller

$$P_{t+2\Delta} = \overline{R} \vee S \overline{Q}_t$$

2. Fall unteres NAND-Gatter schneller

$$P_{t+2\Delta} = \overline{R} \vee S P_t$$

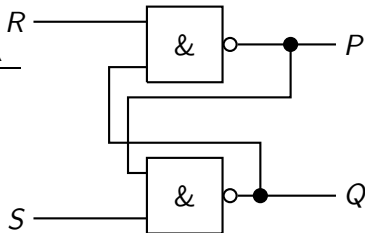
**Beobachtung** Wenn  $P_t = \overline{Q}_t$ , ist das Verhalten an  $P_t$   
**stabil**, also von den Schaltzeiten der Gatter unabhängig.

Was ist mit dem anderen Ausgang?

## Analyse der bi-stabilen NAND-Kippstufe

1. Fall oberes NAND-Gatter schneller

$R_t$	$S_t$	$P_{t+\Delta}$	$Q_{t+\Delta}$	$P_{t+2\Delta}$	$Q_{t+2\Delta}$
0	0	1	1	1	1
0	1	1	0	1	0
1	0	$\overline{Q_t}$	1	0	0
1	1	$\overline{Q_t}$	$Q_t$	$\overline{Q_t}$	$Q_t$

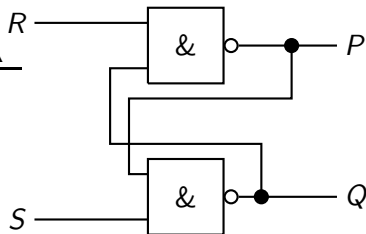


$$\begin{aligned}
 Q_{t+2\Delta} &= \overline{S P_{t+2\Delta}} = \overline{S} \vee \overline{P_{t+2\Delta}} = \overline{S} \vee \overline{\overline{\overline{R Q_{t+\Delta}}}} \\
 &= \overline{S} \vee R Q_{t+\Delta} = \overline{S} \vee R \overline{\overline{S P_{t+\Delta}}} = \overline{S} \vee R (\overline{S} \vee \overline{P_{t+\Delta}}) \\
 &= \overline{S} \vee R \overline{S} \vee R \overline{P_{t+\Delta}} = \overline{S} \vee R \overline{P_{t+\Delta}} = \overline{S} \vee R \overline{\overline{R Q_t}} \\
 &= \overline{S} \vee R R Q_t = \overline{S} \vee R Q_t
 \end{aligned}$$

## Analyse der bi-stabilen NAND-Kippstufe

2. Fall unteres NAND-Gatter schneller

$R_t$	$S_t$	$P_{t+\Delta}$	$Q_{t+\Delta}$	$P_{t+2\Delta}$	$Q_{t+2\Delta}$
0	0	1	1	1	1
0	1	1	$\overline{P_t}$	1	0
1	0	0	1	0	1
1	1	$P_t$	$\overline{P_t}$	$P_t$	$\overline{P_t}$



$$\begin{aligned}
 Q_{t+2\Delta} &= \overline{S P_{t+\Delta}} = \overline{S} \vee \overline{P_{t+\Delta}} = \overline{S} \vee \overline{\overline{R Q_{t+\Delta}}} \\
 &= \overline{S} \vee R Q_{t+\Delta} = \overline{S} \vee R \overline{S P_t} = \overline{S} \vee R (\overline{S} \vee \overline{P_t}) \\
 &= \overline{S} \vee R \overline{S} \vee R \overline{P_t} = \overline{S} \vee R \overline{P_t}
 \end{aligned}$$

## Fazit der Analyse der bi-stabilen NAND-Kippstufe

1. Fall oberes NAND-Gatter schneller

$$P_{t+2\Delta} = \overline{R} \vee S \overline{Q}_t$$

$$Q_{t+2\Delta} = S \vee R Q_t$$

2. Fall unteres NAND-Gatter schneller

$$P_{t+2\Delta} = \overline{R} \vee S P_t$$

$$Q_{t+2\Delta} = S \vee R \overline{P}_t$$

also Wenn  $Q_t = \overline{P}_t$ , so ist das Verhalten stabil, von den Schaltzeiten der Gatter unabhängig.

also **Forderung**  $P_t \neq Q_t$

## Wertetabelle bi-stabile NAND-Kippstufe

$R_t$	$S_t$	oberes Gatter schneller		unteres Gatter schneller	
		$P_{t+2\Delta}$	$Q_{t+2\Delta}$	$P_{t+2\Delta}$	$Q_{t+2\Delta}$
0	0	1	1	1	1
0	1	1	0	1	0
1	0	0	1	0	1
1	1	$\overline{Q_t}$	$Q_t$	$P_t$	$\overline{P_t}$

**Beobachtung** Wir müssen nur  $R = S = 0$  ausschließen.

$R_t$	$S_t$	$P_{t+2\Delta}$	$Q_{t+2\Delta}$
0	1	1	0
1	0	0	1
1	1	$P_t$	$\overline{P_t}$

- ▶  $(R, S) = (0, 1)$  setzt  $P = 1$
- ▶  $(R, S) = (1, 0)$  setzt  $P = 0$
- ▶  $(R, S) = (1, 1)$  lässt  $P$  unverändert

**Fazit** Bi-stabile NAND-Kippstufe realisiert 1-Bit-Speicher!

## Erstes Fazit zu sequenziellen Schaltungen

Bi-stabile NAND-Kippstufe realisiert 1-Bit-Speicher.

also

- ▶ Kreise in „Schaltnetzen“ manchmal sinnvoll
- ▶ neue Funktionalität
- ▶ Analyse schwierig

Wunsch    strukturierter Entwurf

## Automaten

Wunsch formales Modell eines Automaten

Was ist überhaupt ein Automat?

Beispiele

- ▶ Getränke-Automat
- ▶ einfache Ampelsteuerung
- ▶ Steuerung einer Waschmaschine

Gegenbeispiele

- ▶ Geldspielautomat  
wegen der Zufalls-Komponente
- ▶ Computer  
zu komplex
- ▶ Mensch  
für uns nicht formal beschreibbar



# Automatenmodell

## Grobbeschreibung

- ▶ verarbeitet eine Eingabe
- ▶ erzeugt eine Ausgabe
- ▶ ist in einem Zustand
- ▶ arbeitet in Takten
- ▶ arbeitet deterministisch (exakt vorhersagbar)

jetzt exakte, formale Beschreibung

## Definition Mealy-Automat

### Definiton 19

Ein **Mealy-Automat**  $M = (Q, q_0, \Sigma, \Delta, \delta, \lambda)$  ist definiert durch:

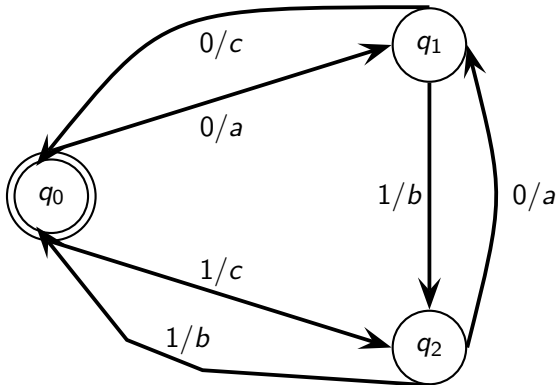
- ▶ endliche Zustandsmenge  $Q$
- ▶ Startzustand  $q_0 \in Q$
- ▶ endliches Eingabealphabet  $\Sigma$
- ▶ endliches Ausgabealphabet  $\Delta$
- ▶ Zustandsüberföhrungsfunktion  $\delta: Q \times \Sigma \rightarrow Q$
- ▶ Ausgabefunktion  $\lambda: Q \times \Sigma \rightarrow \Delta \cup \{\varepsilon\}$

In einem **Takt** mit aktuellem Zustand  $q$  und Eingabesymbol  $w$

- ▶ schreibt der Automat  $\lambda(q, w)$ ,
- ▶ wechselt der Automat in den Zustand  $\delta(q, w)$ .

## Beispiel Mealy-Automat

$$Q = \{q_0, q_1, q_2\}, \Sigma = \{0, 1\}, \Delta = \{a, b, c\}$$



Eingabe 0 1 0 0

Ausgabe a b a c

## Äquivalenz von Automaten

### Definition

Zwei Mealy-Automaten heißen **äquivalent**, wenn sie für jede Eingabe  $w \in \Sigma^*$  die gleiche Ausgabe  $a \in \Delta^*$  erzeugen.

**klar** Äquivalente Automaten können unterschiedlich groß sein.

**klar** Man wünscht sich möglichst kleine Automaten.

**Anmerkung** Komplexer Problemkreis,  
umfasst auch effiziente Minimierung von Automaten  
⇒ Näher i.d. Theoretischen Informatik (GTI/TIfAI)

**Hier:** noch ein anderes (ähnliches!) Automaten-Modell

## Definition Moore-Automat

### Definiton 20

Ein **Moore-Automat**  $M = (Q, q_0, \Sigma, \Delta, \delta, \lambda)$  ist definiert durch:

- ▶ endliche Zustandsmenge  $Q$
- ▶ Startzustand  $q_0 \in Q$
- ▶ endliches Eingabealphabet  $\Sigma$
- ▶ endliches Ausgabealphabet  $\Delta$
- ▶ Zustandsüberföhrungsfunktion  $\delta: Q \times \Sigma \rightarrow Q$
- ▶ Ausgabefunktion  $\lambda: Q \rightarrow \Delta \cup \{\varepsilon\}$

In einem **Takt** mit aktuellem Zustand  $q$  und Eingabesymbol  $w$

- ▶ schreibt der Automat  $\lambda(\delta(q, w))$ ,
- ▶ wechselt der Automat in den Zustand  $\delta(q, w)$ .

## Mealy- und Moore-Automaten

**Beobachtung** Zu jedem Moore-Automaten gibt es einen äquivalenten Mealy-Automaten.

**denn** zu Moore-Automat  $A = (Q, q_0, \Sigma, \Delta, \delta, \lambda)$   
 ist Mealy-Automat  $A' = (Q, q_0, \Sigma, \Delta, \delta, \lambda')$   
 mit  $\lambda'(q, w) := \lambda(\delta(q, w))$   
 offensichtlich äquivalent

**Beobachtung** Zu jedem Mealy-Automaten gibt es einen äquivalenten Moore-Automaten.

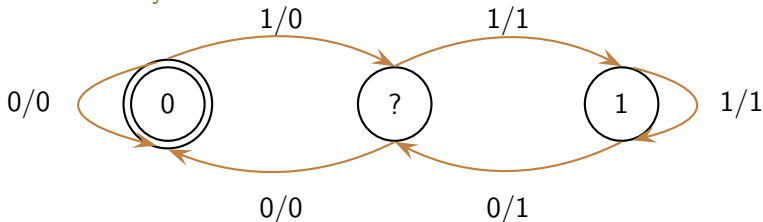
**denn** zu Mealy-Automaten  $A = (Q, q_0, \Sigma, \Delta, \delta, \lambda)$   
 ist Moore-Automat  $A' = (Q', q'_0, \Sigma, \Delta, \delta', \lambda')$   
 mit  $Q' := Q \times (\Delta \cup \{\varepsilon\})$ ,  $q'_0 := (q_0, \varepsilon)$ ,  
 $\delta'(q', w) = \delta'((q, v), w) := (\delta(q, w), \lambda(q, w))$ ,  
 $\lambda(\delta(q', w)) = \lambda'((q, v)) := v$

## Einfacher Beispiel-Automat

**Aufgabe** einfache „Datenglättung“  
 Filtere isolierte Bits aus Datenstrom aus.

**klar**  $\Sigma := \{0, 1\}$ ,  $\Delta := \{0, 1\}$

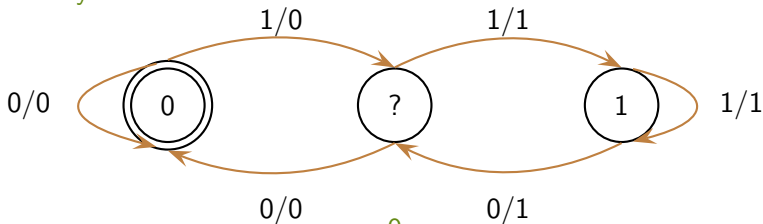
Mealy-Automat



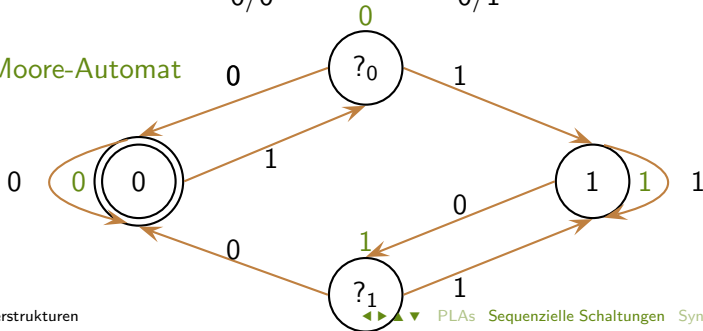
**übrigens**  $Q := \{0, 1, ?\}$ ,  $q_0 := 0$

## Äquivalente Automaten „Bit-Filter“

Mealy-Automat



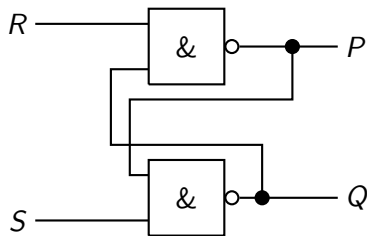
Moore-Automat



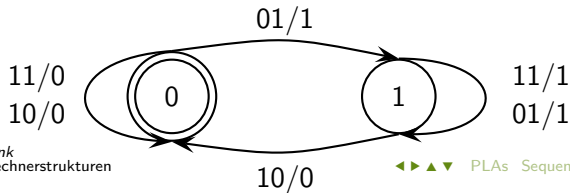


# Automaten und Schaltungen: Synchrone Schaltwerke

Erinnerung bi-stabile NAND-Kippstufe



$R_t$	$S_t$	$P_{t+2\Delta}$	$Q_{t+2\Delta}$
0	1	1	0
1	0	0	1
1	1	$P_t$	$\overline{P_t}$



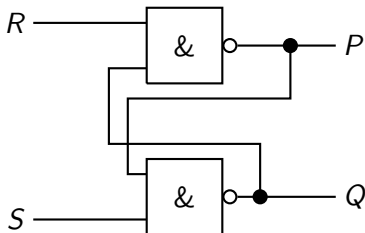
$$Q = \{0, 1\}, q_0 = 0$$

$$\Sigma = \{01, 10, 11\}$$

$$\Delta = \{0, 1\}$$

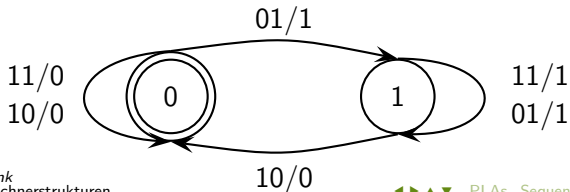
## Vergleich Automat und NAND-Kippstufe

nicht getaktet



$R_t$	$S_t$	$P_{t+2\Delta}$	$Q_{t+2\Delta}$
0	1	1	0
1	0	0	1
1	1	$P_t$	$\overline{P_t}$

getaktet



$$Q = \{0, 1\}, q_0 = 0$$

$$\Sigma = \{01, 10, 11\}$$

$$\Delta = \{0, 1\}$$

## Synchrone Schaltwerke

ab jetzt    getaktete Schaltwerke

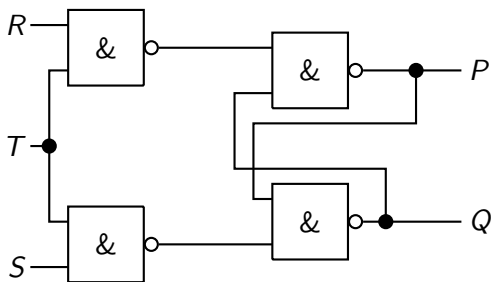
also    Führe Taktsignal  $T$  ein

verschiedene technische Möglichkeiten

- ▶ Pegelsteuerung: aktiv, wenn 1 anliegt
- ▶ positive Flankensteuerung: aktiv, wenn Wechsel von 0 nach 1
- ▶ negative Flankensteuerung: aktiv, wenn Wechsel von 1 nach 0

digital-logische Ebene    technisches Detail ignorieren

## RS-Flip-Flop



R	S	Q
0	0	Q
0	1	1
1	0	0
1	1	nicht erlaubt

### Zustandstabelle NAND-Kippstufe

$R_t$	$S_t$	$P_{t+2\Delta}$	$Q_{t+2\Delta}$
0	0	nicht erlaubt	
0	1	1	0
1	0	0	1
1	1	$\overline{Q_t}$	$Q_t$

### Wertetabelle NAND

x	y	$\overline{xy}$
0	0	1
0	1	1
1	0	1
1	1	0