Universität Bamberg

GRABS: Grundlagen der Rechnerarchitektur und Betriebssysteme

Vorlesung 3: Minimierung und EAs

Michael Engel (michael.engel@uni-bamberg.de)
Lehrstuhl für Praktische Informatik, insbes. Systemnahe Programmierung
https://www.uni-bamberg.de/sysnap

Logikfunktionen und Boolesche Algebra

Verknüpfungen binärer Variablen lassen sich strukturieren:

Seien $n, m \in \mathbb{N}$
Eine Funktion $f: B^n \rightarrow B^m$ heißt **boolesche Funktion**

- B^n = Menge aller n-stelligen Tupel über B = { 0, 1 }
- Beispiel: B^2 = {(0, 0), (0, 1), (1, 0), (1, 1)}

Boolesche Funktion $f: B^n \rightarrow B^m$ als Wertetabelle darstellbar mit $|B^n| = 2^n$ Zeilen und $|B^m| = 2^m$ Möglichkeiten je Zeile

Alle Boolesche Funktionen $f:B^2 \rightarrow B$

Es existieren 16 verschiedene Boolesche Funktionen $f_0 \ldots f_{15}$ $f:B^2 \rightarrow B$ mit 2 Eingabewerten und einem Ausgabewert:

x	0	0	1	1		
y	0	1	0	1		
f_0	0	0	0	0	Nullfunktion	0
f_1	0	0	0	1	UND	*
f_2	0	0	1	0		
f_3	0	0	1	1	Projektion	x
f_4	0	1	0	0		
f_5	0	1	0	1	Projektion	y
f_6	0	1	1	0	XOR	
f_7	0	1	1	1	ODER	+

x	0	0	1	1		
y	0	1	0	1		
f_8	1	0	0	0	NOR	
f_9	1	0	0	1	Äquivalenz	≡
f_{10}	1	0	1	0	Negation	/y
f_{11}	1	0	1	1		
f_{12}	1	1	0	0	Negation	/x
f_{13}	1	1	0	1	Implikation	⇒
f_{14}	1	1	1	0	NAND	
f_{15}	1	1	1	1	Einsfunktion	1

Index und Minterme

Darstellung boolescher Funktionen als **Wertetabelle**:

x	y	f_6
0	0	0
0	1	1
1	0	1
1	1	0

oder **Wertevektor** bei fester Index-Reihenfolge: $f_6 = (0,1,1,0)$

Definition: Die boolesche Funktion, für die nur der Index *i* einschlägig ist, heißt **Minterm** zum Index *i*

Index	x	y	f_6	
0	0	0	0	nicht einschlägig
1	0	1	**1**	**einschlägig**
2	1	0	**1**	**einschlägig**
3	1	1	0	nicht einschlägig

Ein Minterm ist nur mit Negationen (/)

und Konjunktionen (*) darstellbar:

$$x_j = \begin{cases} 0 \rightsquigarrow /x_j \\ 1 \rightsquigarrow x_j \end{cases}$$

und dann Konjunktion all dieser **Literale** (= [negierte] Variable)

Normalformen

Die Darstellung von f als **Disjunktion** (ODER, +) all ihrer Minterme zu einschlägigen Indizes heißt **disjunktive Normalform** (DNF)

Beispiel:

Index	x	y	f_6	Minterm
0	0	0	0	/x * /y
1	0	1	**1**	/x * y
2	1	0	**1**	x * /y
3	1	1	0	x * y

DNF von f_6: /x * y + x * /y

Vorrang- (Präzedenz-)regel:
"Punkt vor Strich"

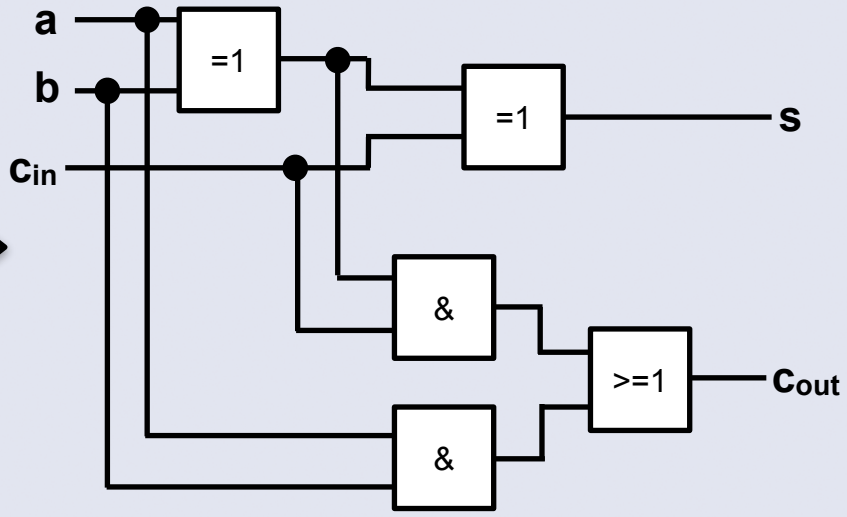
Jede boolesche Funktion $f: B^n \rightarrow B$ **ist nur mittels Konjunktion, Disjunktion und Negation darstellbar** (z. B. durch ihre DNF)

Dies wird auch als "funktional vollständig" bezeichnet

Wie erhält man eine Schaltung aus einer Wertetabelle?

Die Wertetabelle für den Volladdierer beschreibt **zwei** boolesche Funktionen $f: B^3 \rightarrow B$ – **s** und $\mathbf{c_{out}}$**:**

a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	**1**	0
0	1	0	**1**	0
0	1	1	0	**1**
1	0	0	**1**	0
1	0	1	0	**1**
1	1	0	0	**1**
1	1	1	**1**	**1**



Die Tabelle enthält zwei Funktionen – s und c_{out}
Für die Summe:

a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Wertetabelle des Volladdierers
aus Vorlesung 2

Wir betrachten *nur Zeilen mit einschlägigen Indizes* und stellen den Minterm für jede dieser Zeilen auf:

a	b	c_{in}	s
0	0	1	1
0	1	0	1
1	0	0	1
1	1	1	1

$s_1 = /a * /b * c_{in}$

$s_2 = /a * b * /c_{in}$

$s_3 = a * /b * /c_{in}$

$s_4 = a * b * c_{in}$

Schaltungssynthese

a	b	c_{in}	s
0	0	1	1
0	1	0	1
1	0	0	1
1	1	1	1

s_1 = /a * /b * c_{in}

s_2 = /a * b * /c_{in}

s_3 = a * /b * /c_{in}

s_4 = a * b * c_{in}

Vorgehensweise:

Eingangsvariablen mit UND verknüpfen erzeugt Minterm

- Wenn Eingangsvariable = 0: Variable invertieren
- Wenn Eingangsvariable = 1: Variable unverändert verwenden

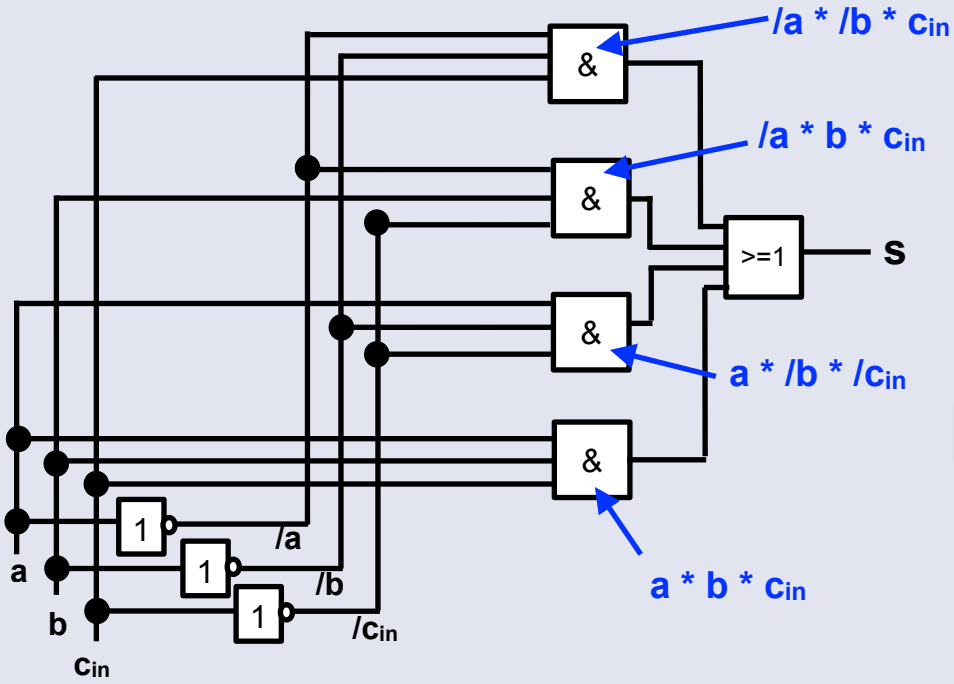
Die vier Zeilen sind die einzigen, die eine "1" an s ergeben
Damit können wir die Terme $s_1 \ldots s_4$ mit ODER zur DNF verbinden:

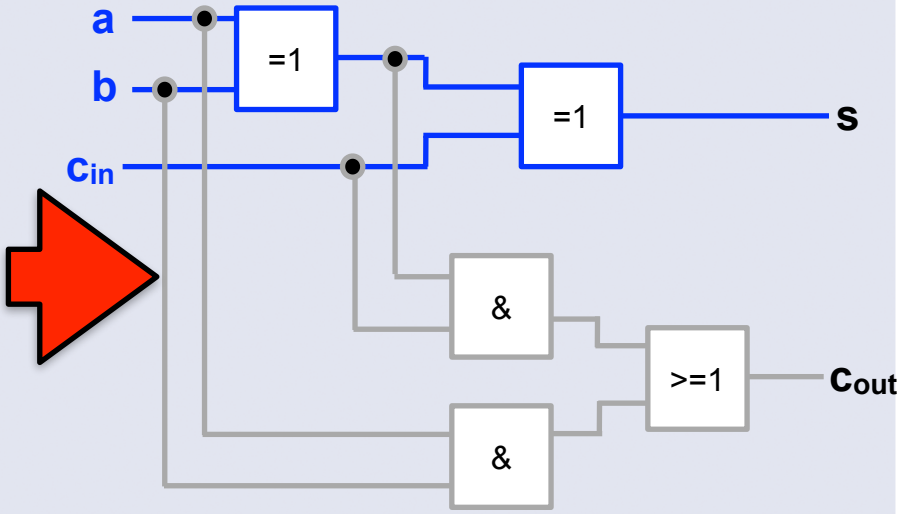
$$s = s_1 + s_2 + s_3 + s_4 = (/a * /b * c_{in}) + (/a * b * /c_{in}) + (a * /b * /c_{in}) + (a * b * c_{in})$$

Schaltungsminimierung

Universität Bamberg

Die erhaltenen Schaltungen sind korrekt, aber ***nicht optimal*** (d.h. sie verwenden mehr als eine minimal mögliche Anzahl von Gattern für die gegebene Funktion)





$$s = (/a * /b * c_{in}) + (/a * b * /c_{in}) + (a * /b * /c_{in}) + (a * b * c_{in})$$

$$s = (a \oplus b) \oplus c_{in}$$

Algebraische Vereinfachung von Schaltungen

Für Verknüpfungen boolescher Variablen x,y,z können wir **Rechenregeln** bestimmen (hier ohne Beweise):

Kommutativität: x*y = y*x, x+y = y+x
Assoziativität: (x*y)*z = x*(y*z), (x+y)+z = x+(y+z)
Distributivität: x*(y+z)=(x*y)+(x*z), x+(y*z) = (x+y)*(x+z)
Neutralelemente: x+0 = x, x*1 = x
Nullelemente: x+1 = 1, x*0 = 0
Idempotenz: x = x+x, x = x*x
Involution: x = //x
Absorption: (x+y)*x = x, (x*y)+x = x
Resolution: (x+y)*(/x+y) = y, (x*y)+(/x*y) = y
Komplementarität: x+(y*/y) = x, x*(y+/y) = x
deMorgan: /(x+y) = x*y, /(x*y) = x+y

Algebraische Vereinfachung von Schaltungen

Mit den Rechenregeln können z.B. redundante Ausdrücke eliminiert werden

Beispiel:

f: $\{0, 1\}^4 \rightarrow \{0, 1\}$ mit Wertevektor (1,0,1,0,1,0,1,0,1,1,1,1,0,0,0,0)

Minterm zu i ist Funktion m_i: $\{0, 1\}^4 \rightarrow \{0, 1\}$ mit
$m_i(x_1,x_2,x_3,x_4) = 1 \Leftrightarrow (x_1\ x_2\ x_3\ x_4)_2 = i$ (Index in Binärschreibweise = i)

Also ist für i=2: $m_2(x_1,x_2,x_3,x_4) = /x_1 * /x_2 * x_3 * /x_4$

da: $/x_1 * /x_2 * x_3 * /x_4 = 1$

$\Leftrightarrow (x_1=0) \wedge (x_2=0) \wedge (x_3=1) \wedge (x_4=0) \Leftrightarrow (x_1x_2x_3x_4)_2 = 0010_2 = 2_{10}$

Algebraische Vereinfachung von Schaltungen

f: $\{0, 1\}^4 \rightarrow \{0, 1\}$ mit Wertevektor (1,0,1,0,1,0,1,0,1,1,1,1,0,0,0,0)

Entsprechende DNF:

$f(x_1,x_2,x_3,x_4) = /x_1*/x_2*/x_3*/x_4 + /x_1*/x_2*x_3*/x_4 + /x_1*x_2*/x_3*/x_4 + /x_1*x_2*x_3*/x_4$
$+ x_1*/x_2*/x_3*/x_4 + x_1*/x_2*/x_3*x_4 + x_1*/x_2*x_3*/x_4 + x_1*/x_2*x_3*x_4$

$f(x_1,x_2,x_3,x_4) = /x_1*/x_2*/x_3*/x_4 + /x_1*/x_2*x_3*/x_4 + /x_1*x_2*/x_3*/x_4 + /x_1*x_2*x_3*/x_4$
$+ x_1*/x_2*/x_3*/x_4 + x_1*/x_2*/x_3*x_4 + x_1*/x_2*x_3*/x_4 + x_1*/x_2*x_3*x_4$

Vereinfachen:

$/x_1*/x_2*/x_3*/x_4 + /x_1*/x_2*x_3*/x_4 = /x_1*/x_2*/x_4$

$/x_1*x_2*/x_3*/x_4 + /x_1*x_2*x_3*/x_4 = /x_1*x_2*/x_4$

Resolution: (x+y)*(/x+y) = y, (x*y)+(/x*y) = y

…und so weiter!

Damit: $f(x_1,x_2,x_3,x_4) = /x_1*/x_2*/x_4 + /x_1*x_2*/x_4 + x_1*/x_2*/x_3 + x_1*/x_2*x_3$

Algebraische Vereinfachung von Schaltungen

$f(x_1,x_2,x_3,x_4) = /x_1*/x_2*/x_4 + /x_1*x_2*/x_4 + x_1*/x_2*/x_3 + x_1*/x_2*x_3$

Können wir weiter vereinfachen?

Resolution: (x+y)*(/x+y) = y, (x*y)+(/x*y) = y

$/x_1*/x_2*/x_4 + /x_1*x_2*/x_4 = /x_1*/x_4$

$x_1*/x_2*/x_3 + x_1*/x_2*x_3 = x_1*/x_2$

Damit ergibt sich die vereinfachte Funktion:

$$f(x_1,x_2,x_3,x_4) = /x_1*/x_4 + x_1*/x_2$$

Zum Vergleich: ursprüngliche DNF:

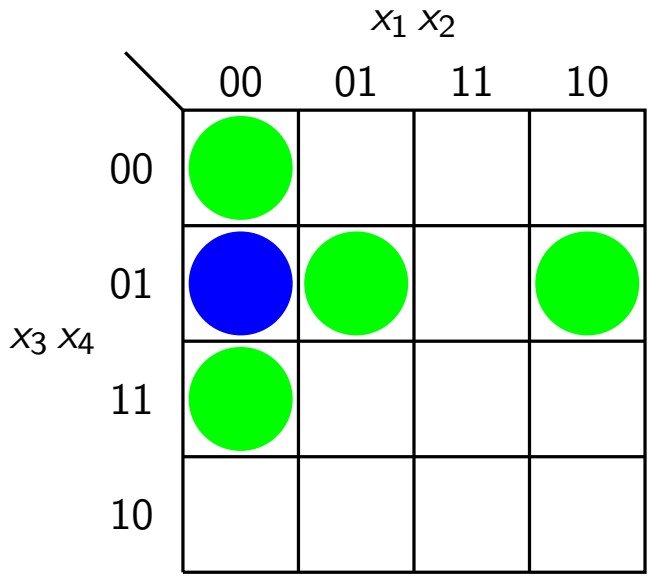
$f(x_1,x_2,x_3,x_4) = /x_1*/x_2*/x_3*/x_4 + /x_1*/x_2*x_3*/x_4 + /x_1*x_2*/x_3*/x_4 + /x_1*x_2*x_3*/x_4$
$+ x_1*/x_2*/x_3*/x_4 + x_1*/x_2*/x_3*x_4 + x_1*/x_2*x_3*/x_4 + x_1*/x_2*x_3*x_4$

…aber: Algebraische Vereinfachung ist extrem aufwendig!

Karnaugh-Veitch (KV)-Diagramme

Verwendet für Schaltungen mit kleiner Anzahl Eingangsvariable: systematischer, anschaulicher und viel übersichtlicherer Weg, um Funktionen $f: \{0,1\}^3 \rightarrow \{0, 1\}$ und $f: \{0,1\}^4 \rightarrow \{0,1\}$ zu vereinfachen

Idee: Nachbarschaften, d.h. Variablenbelegungen unterscheiden sich nur **in einer Stelle**



Hinweis: Nicht alle möglichen Nachbarschaften gezeigt!

Blau markiert:
$(x_1\ x_2\ x_3\ x_4) = 0001$, also $/x_1{}^*/x_2{}^*/x_3{}^*x_4$

Nachbarn in grün:

0000 = $/x_1{}^*/x_2{}^*/x_3{}^*/x_4$
0101 = $/x_1{}^*x_2{}^*/x_3{}^*x_4$
1001 = $x_1{}^*/x_2{}^*/x_3{}^*x_4$
0011 = $/x_1{}^*/x_2{}^*\ x_3{}^*x_4$

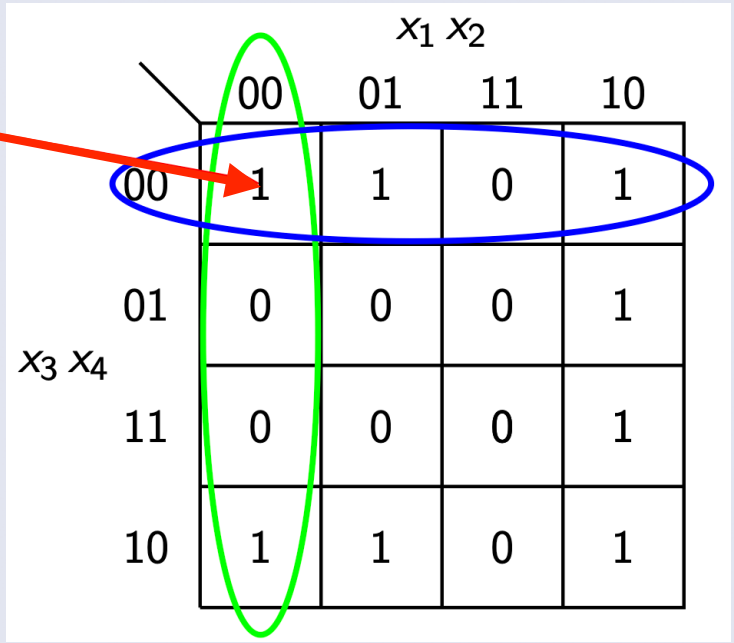
Bei Interesse mehr dazu: Slomka/Glaß [1] Kap. 3.3.4

Achtung: Fehler im Buch!
Beispiel 3.3.2 auf S. 168 muss lauten:
$s(x_1,x_2,x_3) = /x_1{}^*/x_2{}^*x_3 + /x_1{}^*x_2{}^*/x_3 + x_1{}^*/x_2{}^*/x_3 + x_1{}^*x_2{}^*x_3$

Karnaugh-Veitch (KV)-Diagramme

Beispiel: KV-Diagramm für
f: $\{0, 1\}^4 \rightarrow \{0, 1\}$ mit Wertevektor (1,0,1,0,1,0,1,0,1,1,1,1,0,0,0,0)
(Beispielfunktion wie bei arithm. Optimierung gesehen)

x_1	x_2	x_3	x_4	$f(x_1,x_2,x_3,x_4)$
0	0	0	0	1
		⋮		⋮
1	0	1	1	1
		⋮		⋮
1	1	1	1	0

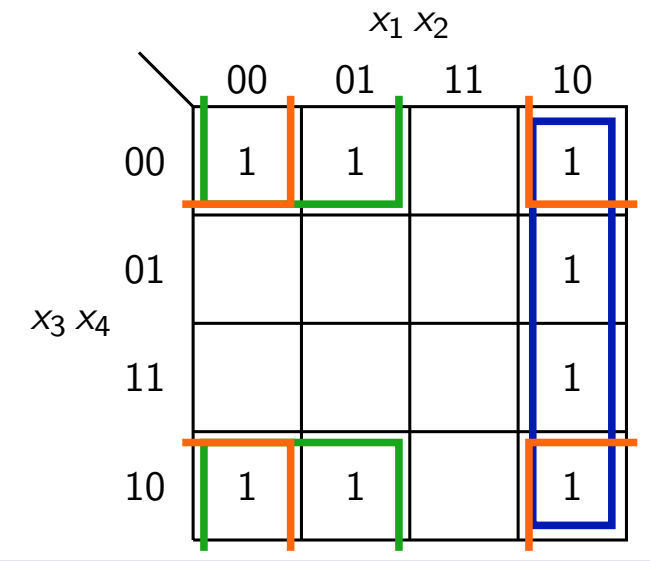


Nullen weglassen
für mehr Übersichtlichkeit…

Was fällt bei der Variablenbelegung auf? Warum ist das so?

Karnaugh-Veitch (KV)-Diagramme

Beispiel: KV-Diagramm für
$f: \{0, 1\}^4 \rightarrow \{0, 1\}$



$x_1 = 1, x_2 = 0 \Rightarrow x_1{}^*/x_2$

$x_1 = 0, x_4 = 0 \Rightarrow /x_1{}^*/x_4$

$x_2 = 0, x_4 = 0 \Rightarrow /x_2{}^*/x_4$

f(x1,x2,x3,x4) = /x1*/x4 + x1*/x2
(+ /x2*/x4)

1. **Suche** alle größten Rechtecke mit Zweierpotenzlänge
2. **Bilde** für jedes Rechteck passendes Monom
3. **Decke** alle Einsen durch **sparsame** Rechteckauswahl ab
4. **Bilde** *f* als Disjunktion der korrespondierenden Monome

*Der rot markierte Term **/x2*/x4** ist **redundant**, da bereits alle 1en von den ersten beiden Termen abgedeckt wurden!*

Andere Ansätze: Quine-McCluskey-Algorithmus

KV-Diagramme sind für Funktionen mit mehr als n=4 Variablen unübersichtlich!

Wie bestimmen wir Minimalpolynome für
$f: \{0,1\}^n \rightarrow \{0,1\}$ für größeres **n**? ⇒Wir suchen einen Algorithmus:

Algorithmus von Quine-McCluskey

Eingabe L_0: Liste aller Minterme zu einschlägigen Indizes von *f*
Ausgabe PI: Menge aller Primimplikanten zu *f*

Quine-McCluskey Teil 1:
Finden von Primimplikanten

```
1. i:=0; PI:=∅
2. Solange Lᵢ ≠ ∅ {
3.    Lᵢ₊₁ := {m | ∃x: {m*x,m*/x} ⊆ Lᵢ} (Resolution)
4.    PI := PI ∪ {m ∈ Lᵢ | m hat keine echte Verkürzung in Lᵢ+1}
5.    i  := i + 1
6. }
7. Ausgabe PI
```

Beispiel PI-Berechnung mit Quine-McCluskey


```
1. i:=0; PI:=∅
2. Solange Lᵢ ≠ ∅ {
3.    Lᵢ₊₁ := {m | ∃x: {m*x,m*/x} ⊆ Lᵢ} (Resolution)
4.    PI := PI ∪ {m ∈ Lᵢ | m hat keine echte Verkürzung in Lᵢ+1}
5.    i  := i + 1
6. }
7. Ausgabe PI
```

$L_0 = \{ /x_1*/x_2*x_3*/x_4,\ /x_1*/x_2*x_3*x_4,\ /x_1*x_2*/x_3*x_4,\ /x_1*x_2*x_3*x_4,\ x_1*/x_2*/x_3*x_4,\ x_1*/x_2*x_3*x_4,\ x_1*x_2*x_3*x_4,\ x_1*x_2*x_3*x_4 \}$

$L_1 = \{ /x_1*/x_2*x_3,\ /x_1*x_3*x_4,\ /x_2*/x_3*x_4,\ /x_1*x_2*x_4,\ x_2*x_3*/x_4,\ x_1*/x_2*x_4,\ x_1*x_3*x_4,\ x_1*x_2*x_3 \}$

$L_2 = \{ x_3*x_4 \}$

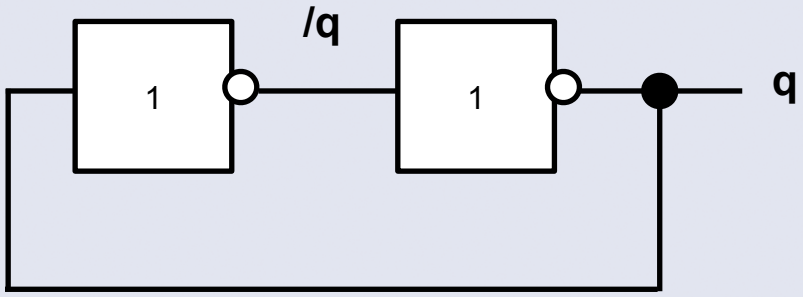
$L_3 = \emptyset$

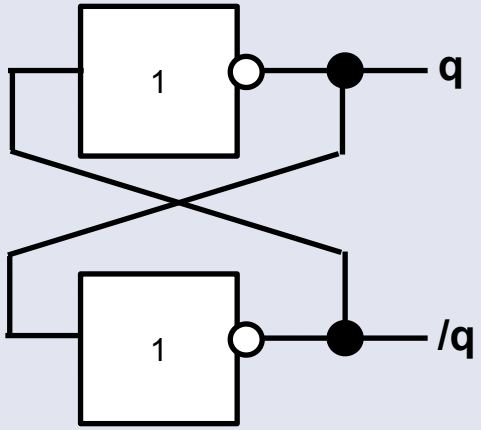
$PI = \{ /x_1*/x_2*x_3,\ /x_1*x_2*x_4,\ x_1*/x_2*x_4,\ x_1*x_2*x_3,\ x_3*x_4 \}$

Eine seltsame Schaltung…

Bisher haben wir Schaltungen entworfen, die Daten von "links" nach "rechts" weiterleiten: ***Schaltnetze***

- **Was macht diese Schaltung?**





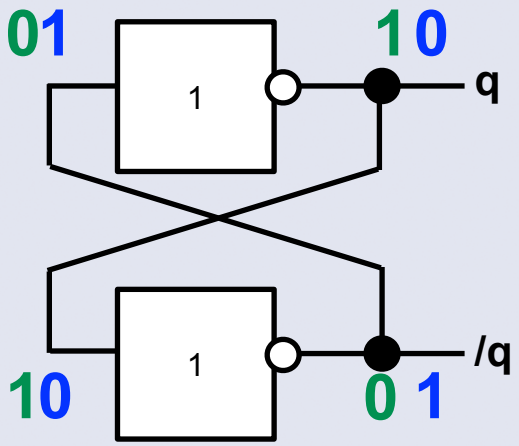
- Kein Eingang, aber ***Rückkopplung:*** ***bistabile*** Schaltung
- Allgemein: **Schaltwerk**

Analyse der bistabilen Grundschaltung

- Wir betrachten zwei Möglichkeiten

- **q** = 0: dann **/q** = 1 und **q** = 0
  - Konsistent und stabil

- **q** = 1: dann **/q** = 0 und **q** = 1
  - Konsistent und stabil



- Die bistabile Schaltung hat n=2 stabile Zustände:
  - Sie ***speichert ein Bit*** ($\log_2 n$) in **q** (oder /q) – 0 oder 1

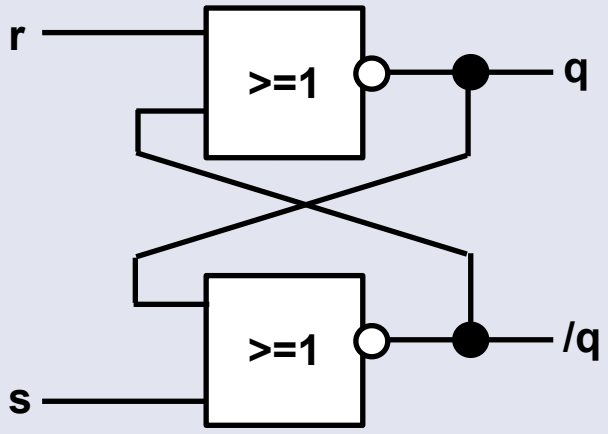
- Aber: **bisher kein Eingang**
  - Wie können wir den Zustand beeinflussen?

SR-Latch (Setzen/Rücksetzen)

- Eingänge zum
  - Setzen (**s**) und
  - Rücksetzen (**r**) des Wertes **q**

- Wir betrachten die vier Fälle:

  I. s = 0, r = 1
  II. s = 1, r = 0
  III. s = 1, r = 1
  IV. s = 0, r = 0

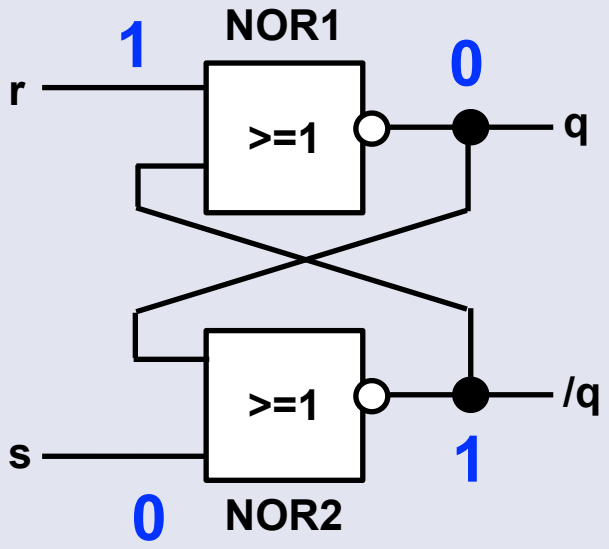


Analyse SR-Latch: s = 0, r = 1 (Fall I)

- **s = 0, r = 1**

- Mindestens ein Eingang von NOR1 (**r**) hat den Wert **1**
  - NOR1 erzeugt also auf jeden Fall eine **0** am Ausgang **q**

- Beide Eingänge (**s** und **q**) von NOR2 sind **0**
  - Damit wird eine **1** an Ausgang **/q** erzeugt

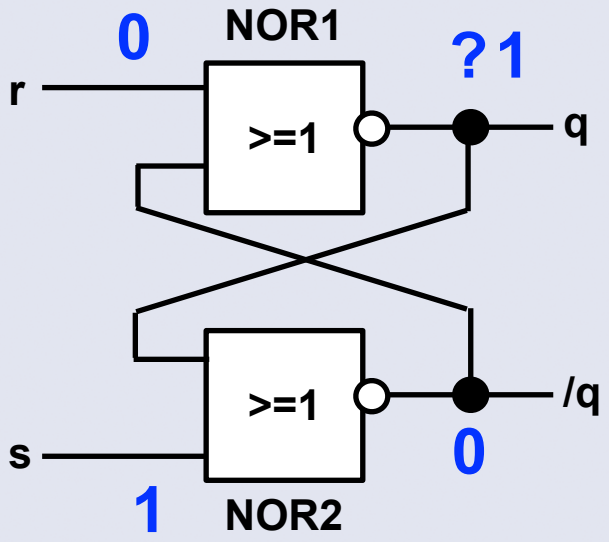


x	y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Analyse SR-Latch: s = 1, r = 0 (Fall II)

Universität Bamberg

- **s = 1, r = 0**
- Die Eingänge von NOR1 sind 0 (**r**) und **/q**:
  - Da wir den Wert von **/q** noch nicht kennen, können wir den Wert von Ausgang **q** noch nicht bestimmen

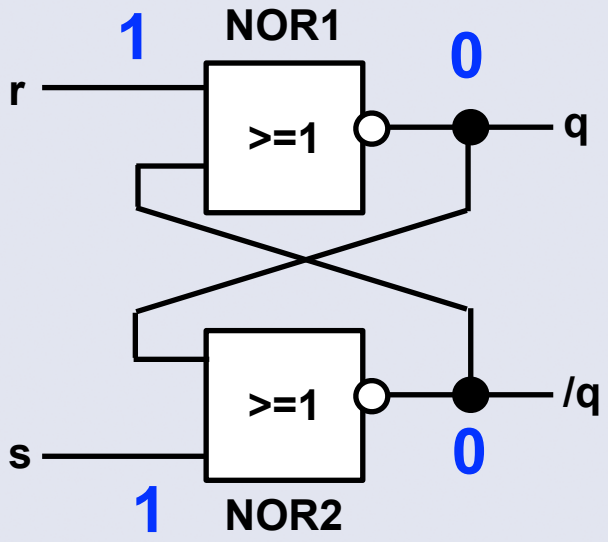


- An NOR2 liegt mindestens ein Eingang auf 1 (**s**) und erzeugt damit eine 0 an Ausgang **/q**:
  - Wenn wir nun NOR1 noch einmal betrachten, sehen wir, dass beide Eingänge den Wert 0 haben, also hat der Ausgang **q** den Wert 1

x	y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Analyse SR-Latch: s = 1, r = 1 (Fall III)

- **s = 1, r = 1**
- Beide NOR-Gatter NOR1 und NOR2 haben mindestens einen Eingang auf dem Wert 1 (**r** bzw. **s**)
  - Damit erzeugen **beide** NOR-Gatter den Ausgabewert **0** an **q** bzw. ***/q***
- *Das Verhalten ist definiert, aber wenig sinnvoll – gleichzeitig soll das Latch gesetzt und zurückgesetzt werden…*



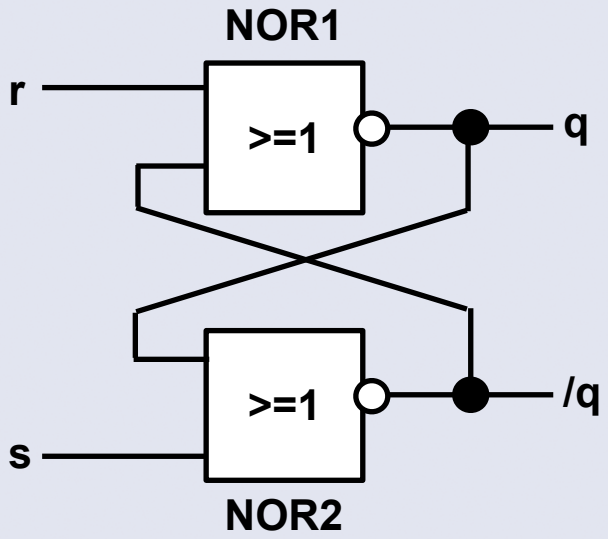
x	y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Analyse SR-Latch: s = 0, r = 0 (Fall IV)

- **s = 0, r = 0**

- NOR1 hat Eingänge **0** (**r**) und **/q**:
  - Da wir den Wert von **/q** noch nicht kennen, können wir den Wert an Ausgang **q** noch nicht bestimmen

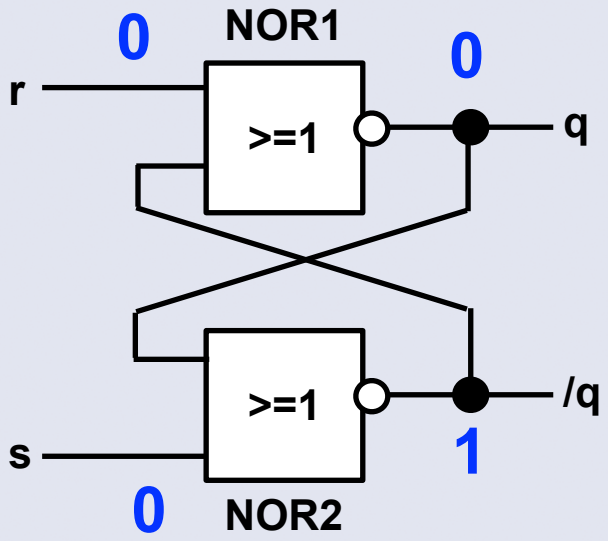
- NOR2 hat Eingangswerte **0** (**s**) und **q**
  - Da wir den Wert von **q** noch nicht kennen, können wir den Wert an Ausgang **/q** noch nicht bestimmen…



- **Problem:** ähnlich zu den über Kreuz verbundenen Invertern! Wir wissen aber, dass **q** entweder den Wert **0** oder **1** haben muss und können das Problem für beide Fälle analysieren!

Problemzustand: r=0, s=0, q=0 (Fall IVa)

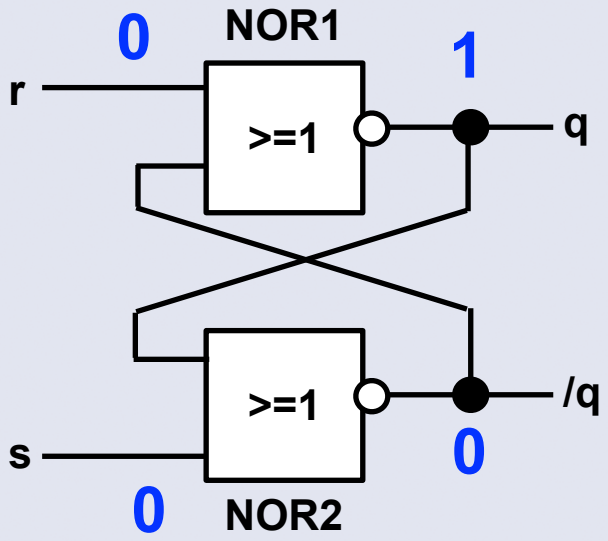
- r = 0, s = 0 und **q = 0**
- Sowohl **s** als auch **q** sind **0**
- Also erzeugt NOR2 den Wert **1** an Ausgang **/q**
- Damit erhält NOR1 eine **1** an Eingang **/q** – also hat der Ausgang **q** von NOR1 den Wert **0**, wie vorher angenommen! ✔



x	y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Problemzustand: r=0, s=0, q=1 (Fall IVb)

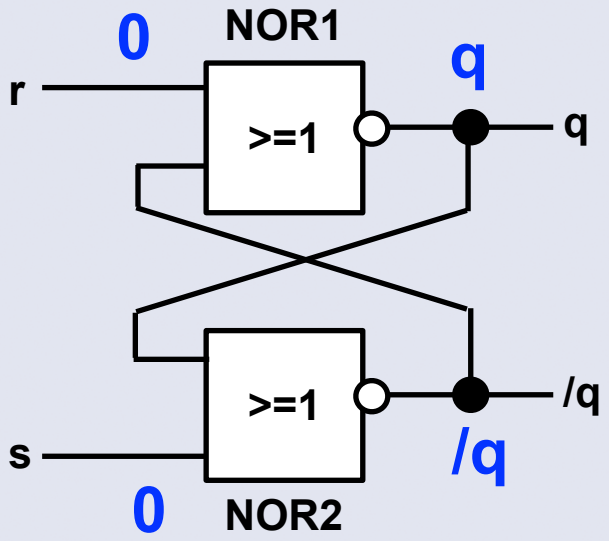
- r = 0, s = 0 und **q = 1**
- Da **q** den Wert **1** hat, erzeugt NOR2 eine **0** an Ausgang **/q**
- Beide Eingänge (r und **/q**) von NOR1 sind auf dem Wert **0**
- Damit erhält NOR1 jeweils eine **0** an Eingang **r** und **/q** – also hat der Ausgang **q** von NOR1 den Wert **1**, wie vorher angenommen!



x	y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Problemzustand… war keiner!

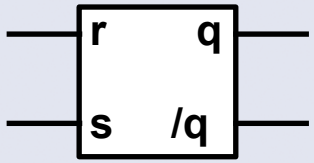
- **Zusammengefasst:**
  - Wir nehmen an, q hat vor unserer Betrachtung einen bekannten Wert $\mathbf{q_{prev}}$ (0 oder 1), der den Zustand des Systems beschreibt
  - Wenn nun **r** und **s** beide den Wert **0** haben, dann **speichert q** diesen vorherigen Wert $\mathbf{q_{prev}}$ und **/q** ist dessen Komplement
- **Diese Schaltung ist ein Speicher!**



Fall	s	r	q	/q
IV	0	0	q_{prev}	$/q_{prev}$
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0

Schaltplansymbol für SR-Latch

- SR steht für Setzen/Rücksetzen
  - Zwei unterscheidbare Zustände an q: 0 und 1
  - Speichert also **ein Bit** ($\log_2 2$) **Zustand** in **q**



- Festlegen des gespeicherten Wertes mit Eingängen s und r:
  - **setzen (set):** setze Ausgang auf 1 (s=1, r=0, q=1)
  - **rücksetzen (reset):** zurücksetzen des Ausgangs auf 0 (s=0, r=1, q=0)
- ***Illegalen Zustand*** vermeiden: es darf niemals s = r = 1 sein!

Ist das SR-Latch ein sinnvoller Speicher?

Das SR-Latch ist unschön zu verwenden, da es sich seltsam verhält, wenn **s** und **r gleichzeitig auf 1** gesetzt sind

Zudem vermischen die s- und r-Eingänge die Angaben…

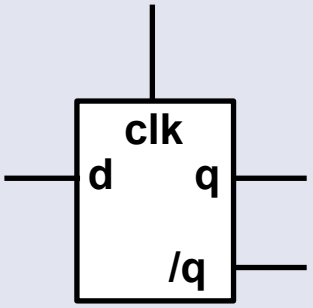
- **was** geändert werden soll (neuer Wert)
  und
- **wann** der Wert geändert werden soll (Zeitpunkt)

- Das Setzen eines der Eingänge auf 1 bestimmt also nicht nur, **welchen Wert** der neue Zustand haben soll, sondern auch, zu **welchem Zeitpunkt** er sich ändern soll
- Schaltkreise lassen sich einfacher entwerfen, wenn diese beiden Aspekte getrennt sind

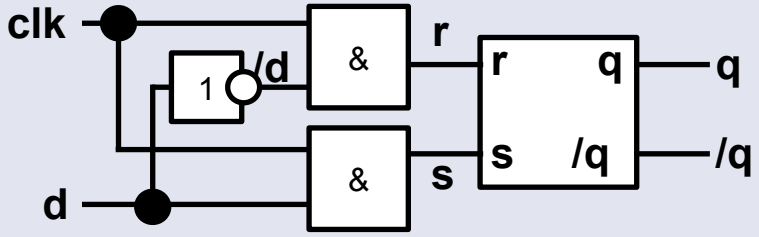
D-Latch

Lösung: **D-Latch** mit zwei Eingängen:

- Der **Daten**eingang **d** bestimmt den **Folgezustand**
- Der **Takt**eingang **clk** (*clock*) bestimmt, **wann** der Zustand geändert werden soll



Schaltsymbol D-Latch



Schaltung D-Latch

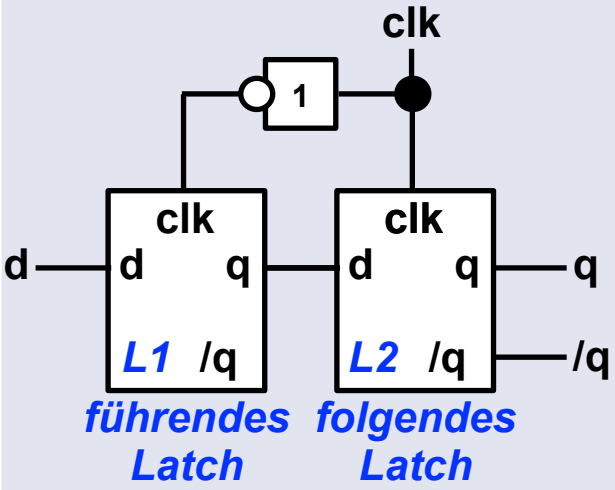
clk	d	/d	s	r	q	/q
0	X	/X	0	0	q_{prev}	$/q_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

X = "don't care": kann 0 oder 1 sein!

Wertetabelle D-Latch

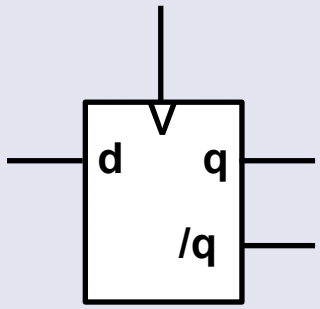
D-Flipflop

- Es ist oft nützlich, den Zustand nur **zu einem bestimmten Zeitpunkt** zu ändern. Dies kann das **D-Flipflop** (**D-FF**)
  - Wenn clk = 0, dann übernimmt das führende Latch L1 Wert d an q
  - Wenn clk = 1, dann übernimmt das folgende Latch L2 den Ausgang q von L1 an seinen Ausgang q
  - Damit wird ein **Wert an d von L1 erst bei einem Übergang 0→1 von clk an q von L2 übernommen:**

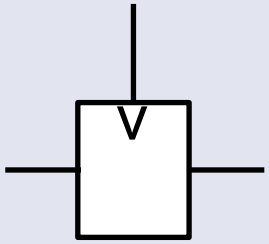


Schaltung D-Flipflop

"**flankengesteuert**" (*edge triggered*)



Schaltsymbol D-Flipflop

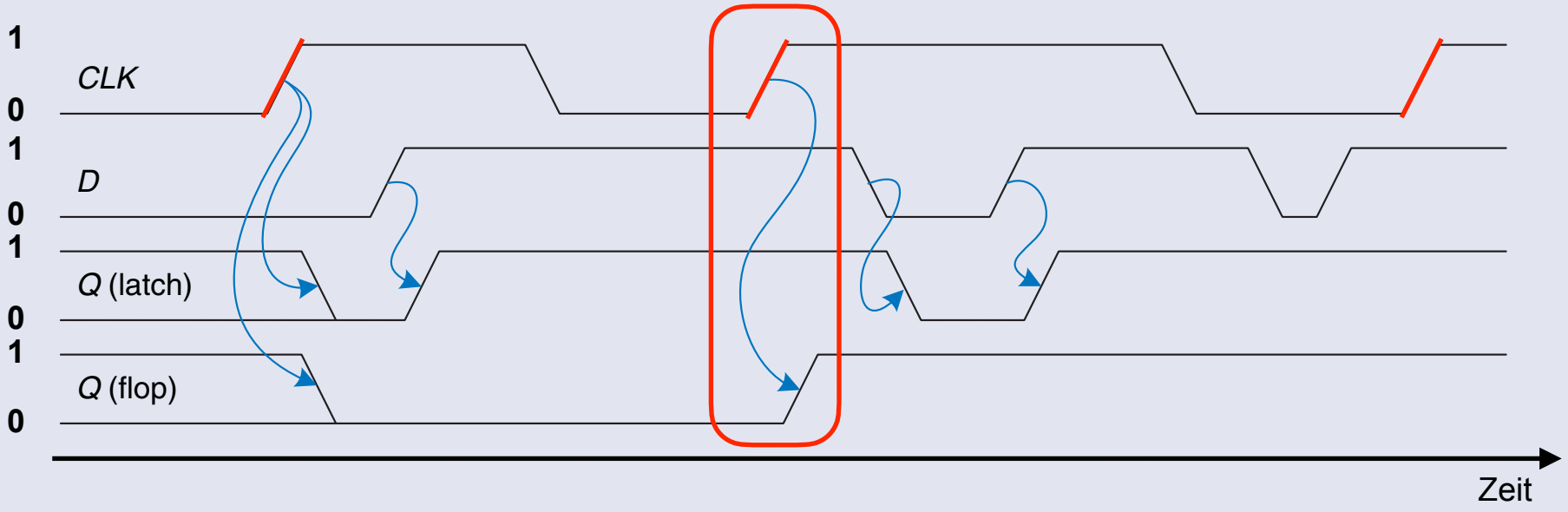


Schaltsymbol D-Flipflop (vereinfacht)

Unterschied im Zeitdiagramm

Wie unterscheiden sich D-Latch und D-Flipflop (D-FF)?

- Das D-FF übernimmt neue Daten an **q** von Eingang **d** nur an der **steigenden Flanke des clk-Signals**
- Der Ausgang **q** des D-Latch übernimmt dagegen neue Daten von **d**, **solange clk = 1** ist

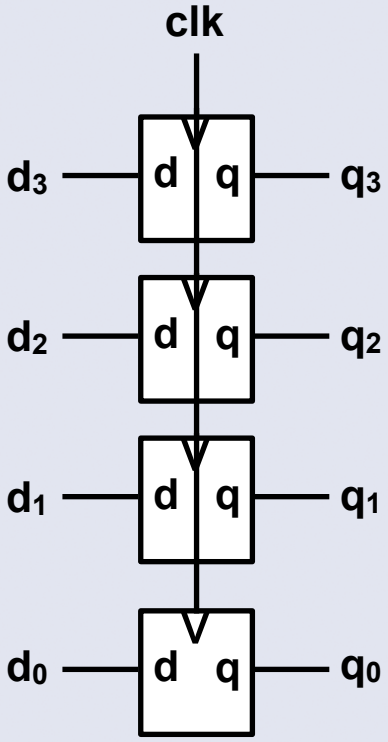


Register

Wie können wir **mehrere Bits** speichern?

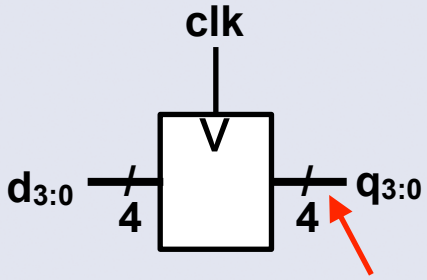
- Parallelschaltung von Flipflops: **Register**

…weil mehr als ein Bit "mitfährt" 😀



Schaltsymbol 4 Bit-Register

$d_{3:0}$ und **$q_{3:0}$** werden auch (4 Bit breiter) **Bus** genannt



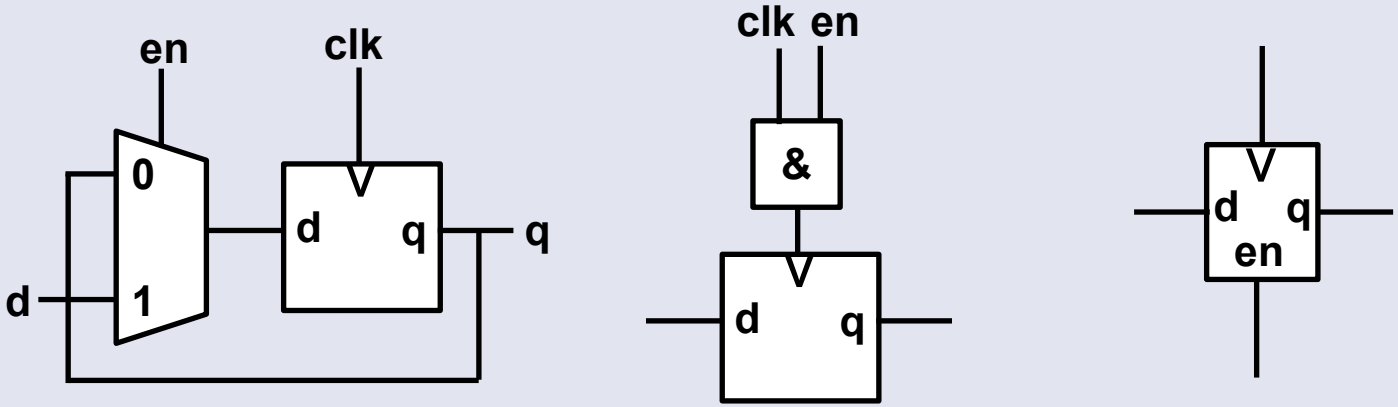
Busse werden in Schaltplänen oft mit **breiteren Linien** dargestellt

Schaltsymbol 4 Bit-Register (vereinfacht)

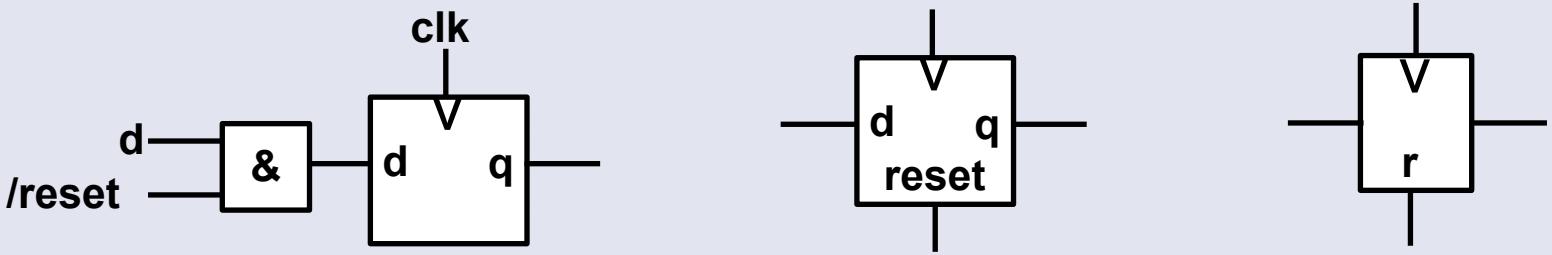
Weitere Flipflops

Es gibt viele verschiedene Arten von Flipflops, z.B. auch:

- **Aktivierbares** (*enabled*) Flipflop: **en** kontrolliert Übernahme

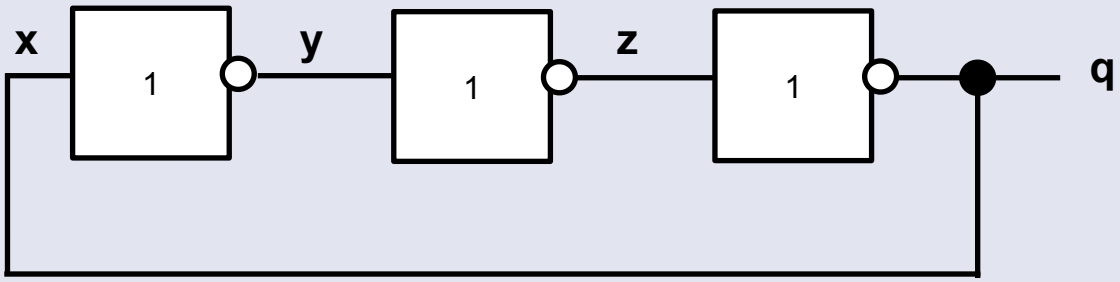


- **Zurücksetzbares** Flipflop: auf 0 (synchron) zurücksetzbar

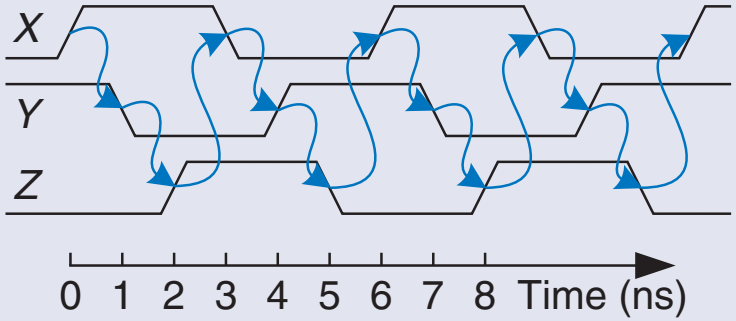


Spaß mit Rückkopplungen…

- Was macht diese Schaltung?



- Sei **x** anfänglich = 0. Dann ist **y** = 1, **z** = 0, und dann **x** = 1
- Dies ist **inkonsistent** zu unserer ursprünglichen Annahme!
- Schaltung ist **instabil (astabil):**
  es ex. kein stabiler Zustand
- Zeitverhalten von Propagationszeit der Inverter abhängig
- Jeder Knoten **schwingt** zwischen 0 und 1: **Ringoszillator**



Entwurf synchroner Schaltungen

Wir verwenden **synchrone Schaltwerke**

- **Probleme mit Rückkopplungen** (Ringosz.) **vermeiden!**
- Vorgabe eines **Takts** erleichtert Schaltungsentwurf
- **Taktdauer** (= 1/Taktfrequenz) muss größer als längste Verzögerung in kombinatorischen Teilschaltungen sein

Regeln für den Entwurf synchroner Schaltwerke:

- Jedes Schaltungselement ist entweder ein Register oder ein kombinatorisches Schaltnetz
- Mindestens ein Schaltungselement ist ein Register
- Alle Register werden mit dem selben Taktsignal versorgt
- Jeder rückgekoppelte Pfad enthält mindestens ein Register

Endliche Automaten

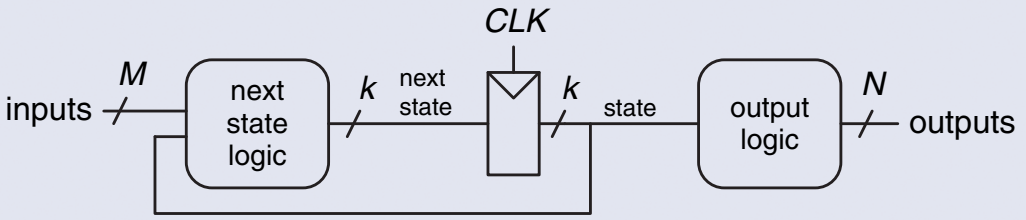
Synchroner Schaltwerke lassen sich durch **Endliche Automaten (EA)** (*finite state machines, FSM*) beschreiben

- Englische Bezeichnung FSM, weil eine Schaltung mit k Bit Registern sich in einem einer endlichen Anzahl (2^k) von unterscheidbaren Zuständen befindet
- Ein EA hat m Eingänge, n Ausgänge und k Bits Zustand
  - Er erhält auch einen Takt und optional ein Reset-Signal
- Ein EA besteht aus zwei Blöcken kombinatorischer Logik, der Folgezustandslogik (*next state*) und der Ausgabelogik (*output*), und einem Register, das den Zustand speichert
- Bei jeder (positiven) Taktflanke nimmt der EA den Folgezustand an, der basierend auf aktuellem Zustand und aktuellen Eingaben berechnet wurde

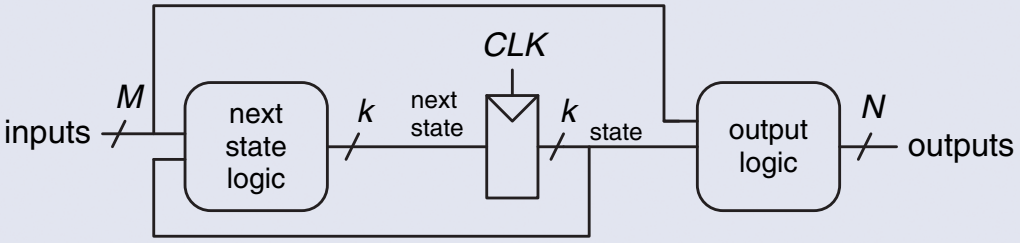
Endliche Automaten

Es existieren zwei grundsätzliche Strukturen für EAs:

- **Moore**-Automat:
  - Ausgabe des Automaten hängt nur vom aktuellen Zustand ab



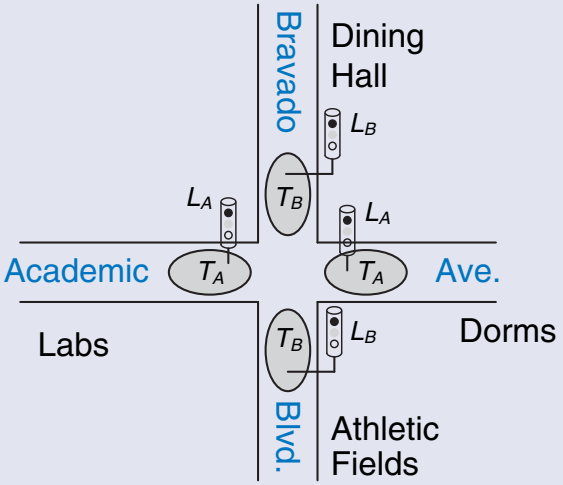
- **Mealy**-Automat:
  - Ausgabe des Automaten hängt vom aktuellen Zustand und der aktuellen Eingabe ab

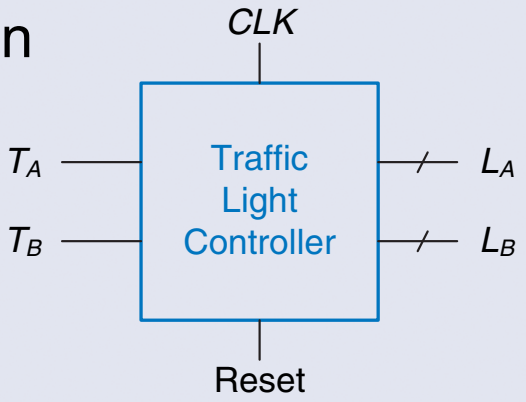


Beispiel: EA für Ampel

Das klassische Beispiel: **Ampelsteuerung**

- (Amerikanische*) Ampeln L_A, L_B für horizontale und vertikale Fahrtrichtung
- Zusätzlich Verkehrssensoren T_A, T_B für horizontale und vertikale Fahrtrichtung
- Der Takt hat eine **Periode** von 5 Sekunden
- Bei jeden "Tick" des Takts (steigende Flanke) können die Ampeln auf Basis der Sensordaten umschalten
  - **Sensoren** T_a, T_b liefern Werte 1 oder 0
  - **Ampeln** L_a, L_b haben Farbe rot/gelb/grün
  - **Zusätzlich**: Reset-Signal
- Aufgabe: EA für Ampelsteuerung bauen





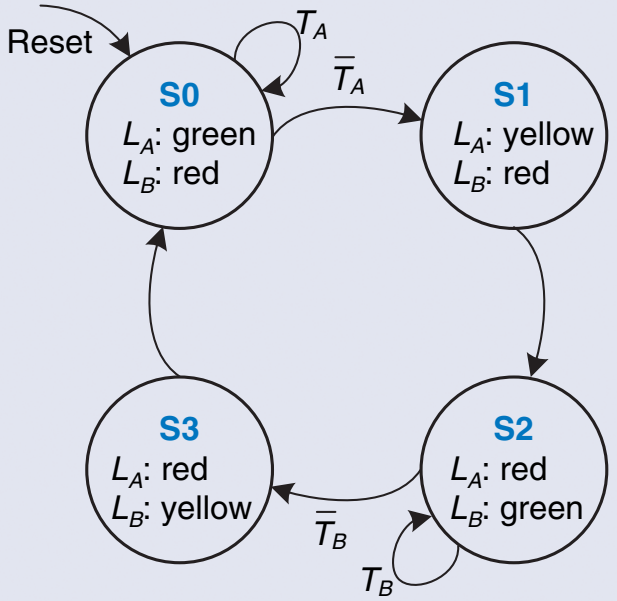
* keine Gelb-Rot-Phase – macht den Entwurf einfacher…

Beschreibung Endlicher Automaten

Beschreibung des EA-Verhaltens durch Zustandsübergangsgraph bzw. -tabelle

- **Moore**-Automaten: Ausgangswerte den Zuständen zuordnen:
  - **Zustände: Kreise** (Knoten)
  - **Übergänge: Pfeile** (Kanten)

Mealy-Automaten benötigen eine kombinierte Zustands-/Eingangs-/Ausgabetabelle (bzw. den entsprechenden Graphen)



Aktueller Zustand *S*	Eingang T_A	Eingang T_B	Folge-zustand *S'*
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

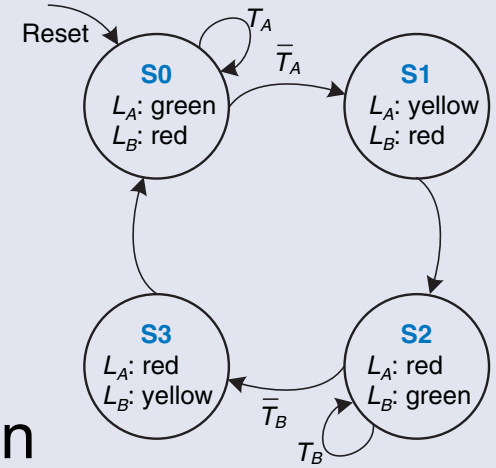
Zustandsübergangstabelle

Beschreibung Endlicher Automaten

Universität Bamberg

Zur Realisierung der Schaltung des EA müssen die **abstrakten Zustände** S0-S3 und die **Ausgabewerte** rot/gelb/grün noch Bitkombinationen zugeordnet werden

- **Zustandscodierung**
  - definiert eindeutige Binärdarstellung für jeden Zustand
- **Codierung der Ausgabewerte**
  - legt fest, welche Bitkombinationen am Ausgang anliegen sollen



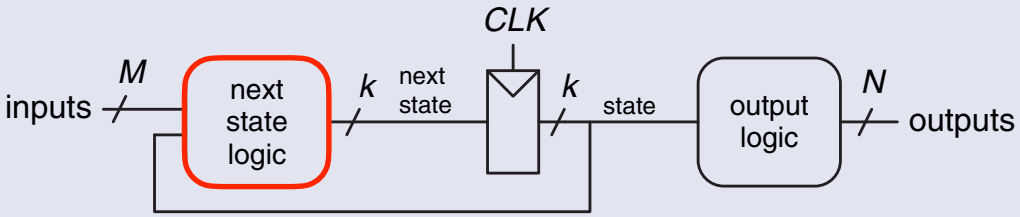
Zustand	Codierung $S_{1:0}$
S0	00
S1	01
S2	10
S3	11

Zustandscodierung

Ausgabe	Codierung $L_{1:0}$
grün	00
gelb	01
rot	10

Ausgabecodierung

Schaltungsumsetzung des EA



Für den Entwurf der realen Schaltung müssen die beiden (kombinatorischen) Schaltnetze für den **Folgezustand** und die **Ausgabe** entworfen werden

- Dazu schreiben wir die Zustandsübergangstabelle um, indem wir Zustände durch ihre binäre Codierung ersetzen

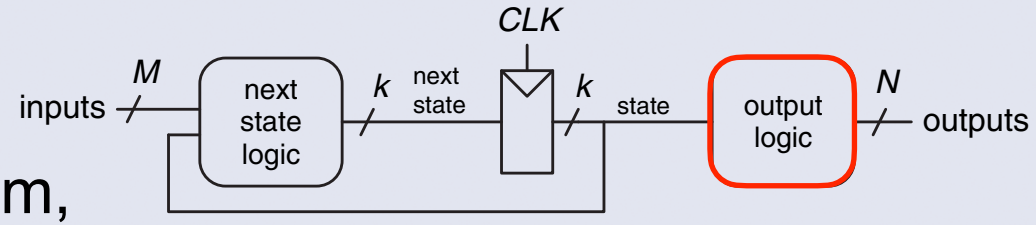
Aktueller Zustand		Eingang		Folge-zustand	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

Es ergibt sich damit:

- $S'_1 = /S_1S_0 + S_1/S_0/T_B + S_1/S_0T_B$
- $S'_0 = /S_1/S_0/T_A + S_1/S_0/T_B$

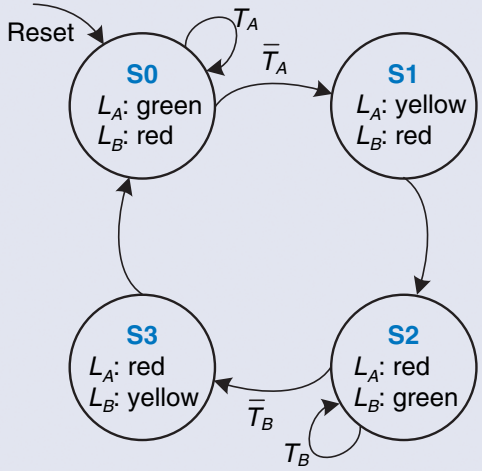
Schaltungsumsetzung des EA

- Nun schreiben wir noch die Ausgabecodierung um, indem wir Ausgabewerte durch ihre binäre Codierung ersetzen



Ausgabe	Codierung $L_{1:0}$
grün	00
gelb	01
rot	10

Ausgabecodierung

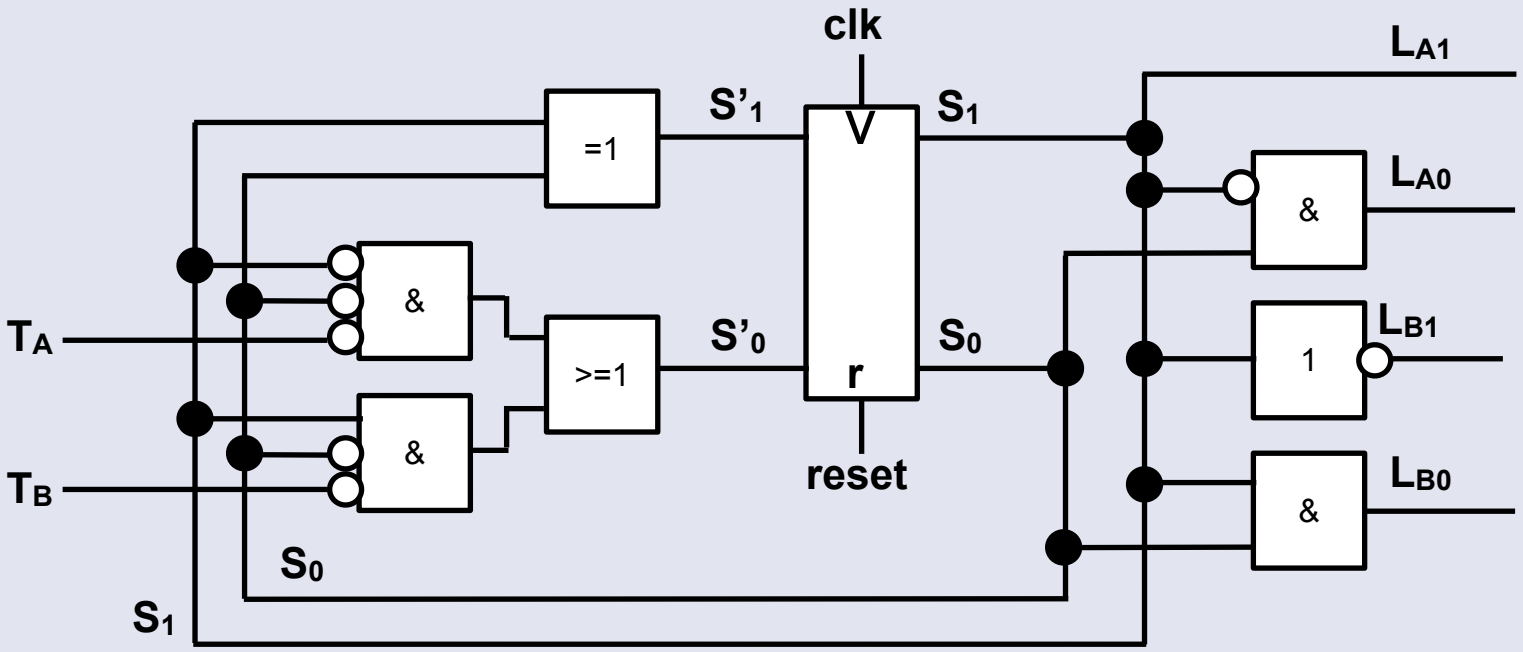


Es ergibt sich:

- $L_{A1} = S_1$
- $L_{A0} = /S_1S_0$
- $L_{B1} = /S_1$
- $L_{B0} = S_1S_0$

Aktueller Zustand S_1	Aktueller Zustand S_0	Ausgabe L_{A1}	Ausgabe L_{A0}	Ausgabe L_{B1}	Ausgabe L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Schaltung des Ampelsteuerungs-EA

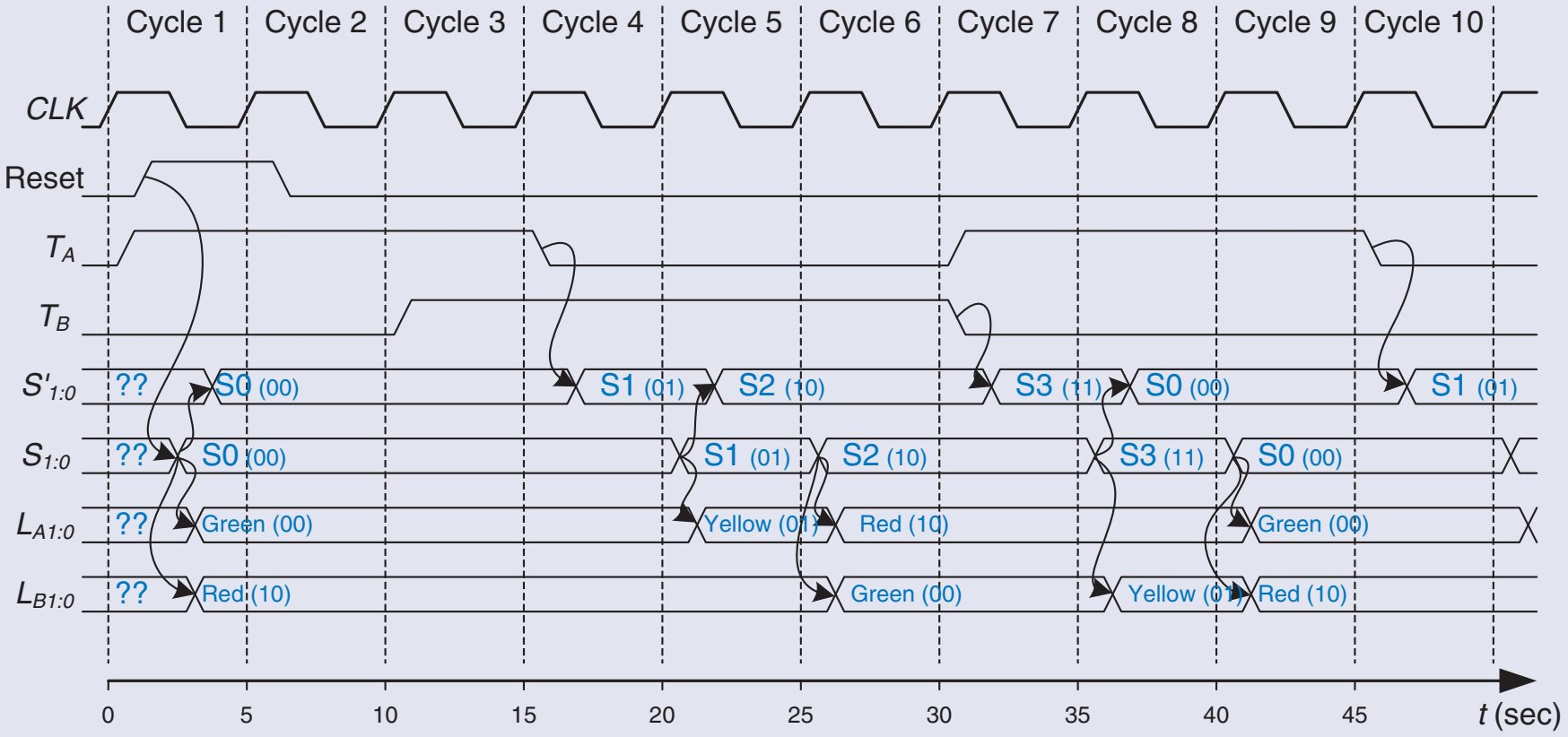


$$\mathbf{S'_1} = /S_1S_0 + S_1/S_0/\mathbf{T_B} + S_1/S_0\mathbf{T_B}$$
$$= /S_1S_0 + S_1/S_0 = \mathbf{S_1 \oplus S_0}$$
$$S'_0 = /S_1/S_0/T_A + S_1/S_0/T_B$$

$$L_{A1} = S_1$$
$$L_{A0} = /S_1S_0$$
$$L_{B1} = /S_1$$
$$L_{B0} = S_1S_0$$

Timing des Ampelsteuerungs-EA

Universität Bamberg



Zustandscodierung in EA

- Gesehen: **binär**
  - z.B. mit den Werten 00, 01, 10, 11 für vier Zustände
- Auch möglich: **1-aus-N-Code** ("*one-hot*")
  - Ein **Zustandsbit pro Zustand**
  - Zu jedem Zeitpunkt ist **genau ein** Zustandsbit gesetzt
  - z.B. als 0001, 0010, 0100, 1000 für vier Zustände
  - Benötigt mehr Flipflops…
    - aber Zustandsübergangs- und Ausgangslogiken sind oft **kleiner**
      - und **schneller**

Referenzen

[1] Frank Slomka, Michael Glaß
Grundlagen der Rechnerarchitektur
Von der Schaltung zum Prozessor

[2] David Harris, Sarah Harris
Digital Design and Computer Architecture, RISC-V Edition