

**IT Infrastrukturen – Rechnerstrukturen**

## **Thema 2: Entwurf digitaler Schaltungen**

**Prof. Dr.-Ing. Sebastian Schlesinger**  
**Professur für Wirtschaftsinformatik**  
**(Infrastruktur & Security)**

---

Dieser Foliensatz basiert auf den Folien  
zu „Rechnerorganisation“ von Prof. Dr.  
Ben Juurlink (TU Berlin) und Prof. Dr.  
Paula Herber (WWU Münster)

Nach dieser VL sollten Sie in der Lage sein:

- die **Wahrheitstabelle einer Booleschen Funktion** aufzustellen
- die **disjunktive oder konjunktive Normalform** aufzustellen
- zu zeigen, dass ein **Operatorensatz vollständig** ist
- Boolesche Funktionen mit **KV-Diagrammen** darzustellen und zu vereinfachen
- ein **Schaltnetz** aus **logischen Gattern** aufzubauen
- das Verhalten von **Speicherelementen** (Latch, Flipflop) zu erklären
- ein **Schaltwerk** aus logischen Gattern und Speicherelementen zu entwerfen
- die Implementierung von **Multiplexer, Decoder** und **Registersätzen** zu erklären
- eine **arithmetische-logische Einheit (ALU)** zu erklären und zu erweitern
- einen **Ripple-Carry-** und einen **Carry-Look-Ahead Addierer** zu erklären

## 1. Kombinatorische Logik (Schaltnetze)

- Logikgatter
- Boolesche Algebra
- Logiksynthese (Schaltnetzentwurf)
- Minimierung von Schaltungen

## 2. Sequentielle Logik (Schaltwerke)

- Speicherelemente (Flipflops, Latches)
- Schaltwerksentwurf

## 3. Wichtige Schaltungen

- Multiplexer und Decoder
- Registersatz
- Arithmetisch-Logische Einheit (ALU)
- Ripple-Carry und Carry-Look-Ahead Addierer

## 1. Kombinatorische Logik (Schaltnetze)

- Logikgatter
- Boolesche Algebra
- Logiksynthese (Schaltnetzentwurf)
- Minimierung von Schaltungen

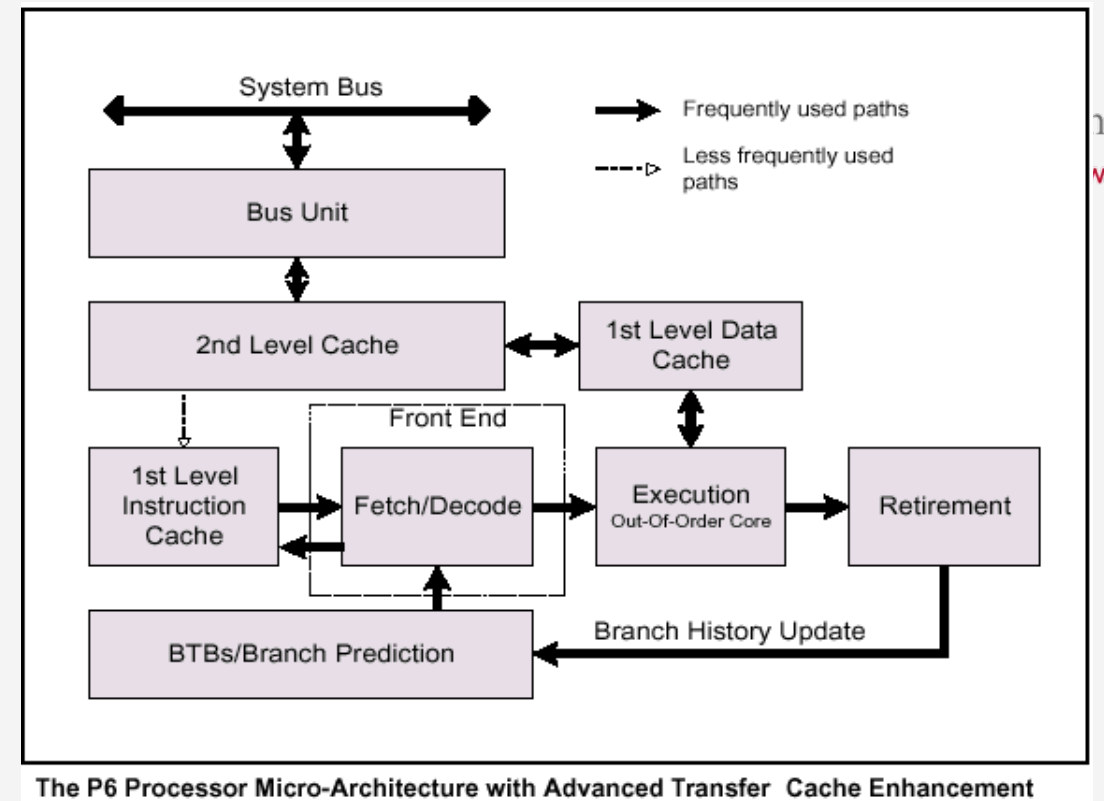
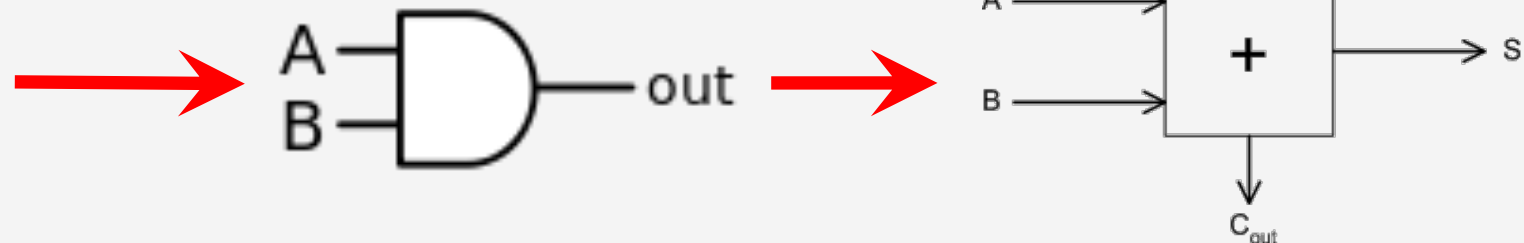
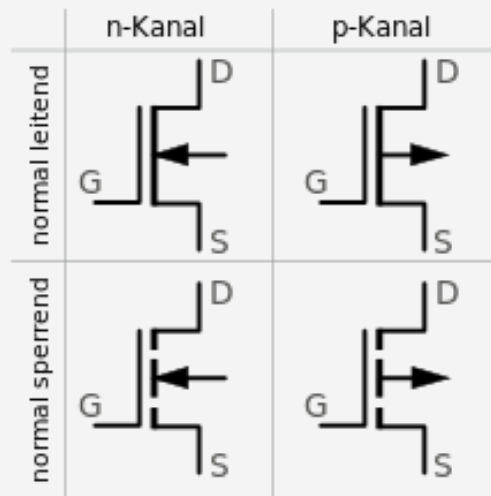
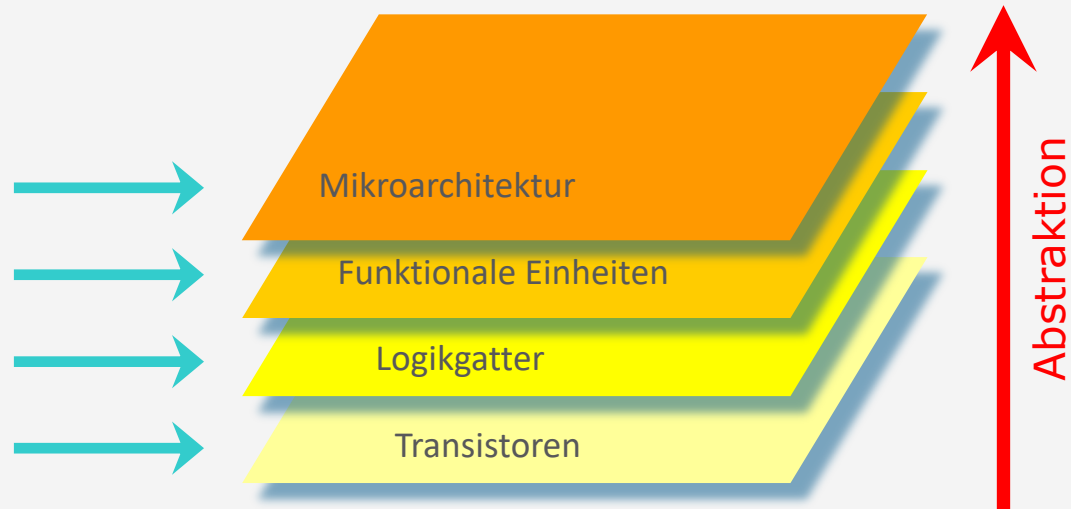
## 2. Sequentielle Logik (Schaltwerke)

- Speicherelemente (Flipflops, Latches)
- Schaltwerksentwurf



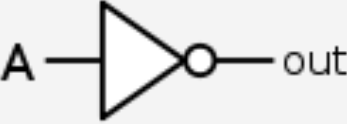
## 3. Wichtige Schaltungen

- Multiplexer und Decoder
- Registersatz
- Arithmetisch-Logische Einheit (ALU)
- Ripple-Carry und Carry-Look-Ahead Addierer

# Hierarchischer Entwurf





	UND	ODER	NICHT
Graphisch			
Aussagenlogik	$Q = A \wedge B$	$Q = A \vee B$	$out = \neg A$
C	$Q = A \& B$	$Q = A   B$	$out = !A$ $(\sim A)$
Schaltalgebra	$Q = A \cdot B$	$Q = A + B$	$out = \bar{A}$

# Wahrheitstabelle



	UND	ODER	NICHT
Graphisch			



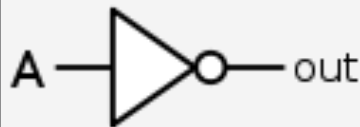
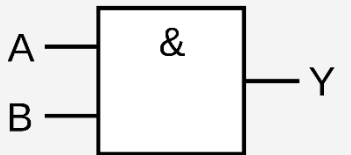
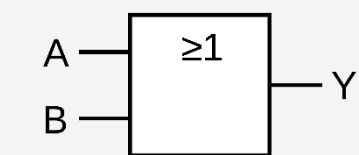
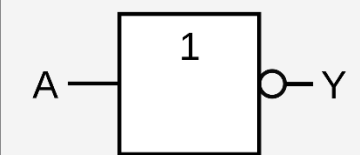
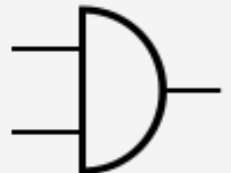
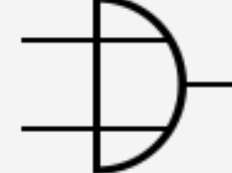
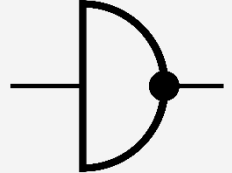
A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

A	$\bar{A}$
0	1
1	0

# Alternative Darstellungen






	UND	ODER	NICHT
MIL/ANSI			
IEC			
DIN			



# Weitere Gatter



	XOR	NAND	NOR
MIL/ANSI			

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0

# Alternative Darstellungen

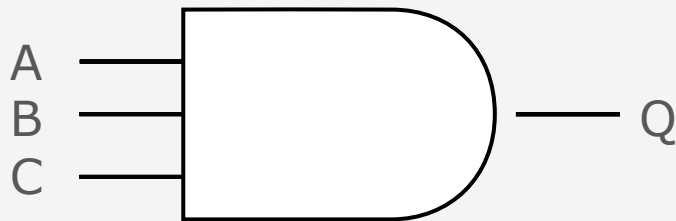


	XOR	NAND	NOR
MIL/ANSI			
IEC			
DIN			

# Größere Gatter



- Gatter können mehr als 2 Eingänge haben
- n-Input UND-Gatter hat als Ausgang 1, nur wenn **alle** Eingänge den Wert 1 haben
- n-Input ODER-Gatter hat als Ausgang 0, nur wenn **alle** Eingänge den Wert 0 haben
- Beispiel 3-Input UND-Gatter:



A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

## 1. Kombinatorische Logik (Schaltnetze)

- Logikgatter
- Boolesche Algebra
- Logiksynthese (Schaltnetzentwurf)
- Minimierung von Schaltungen

## 2. Sequentielle Logik (Schaltwerke)

- Speicherelemente (Flipflops, Latches)
- Schaltwerksentwurf

## 3. Wichtige Schaltungen

- Multiplexer und Decoder
- Registersatz
- Arithmetisch-Logische Einheit (ALU)
- Ripple-Carry und Carry-Look-Ahead Addierer

# Boolesche Algebra



Hochschule für  
Wirtschaft und Recht Berlin  
Berlin School of Economics and Law

- George Boole (1815-1864): Algebra der Logik mit zwei Elementen 0 und 1

## MENU DEALS



SAVE  
£1.20

### Menu 1 £6.90 / 8,50€

Buy any hot drink **or** soup, **and** any sandwich **or** hot snack or the Deli Delights snack box or Saviour snack box, **and** a Twix.

SAVE £1.20 / €1,50

### Menu 2 £6.20 / 8€

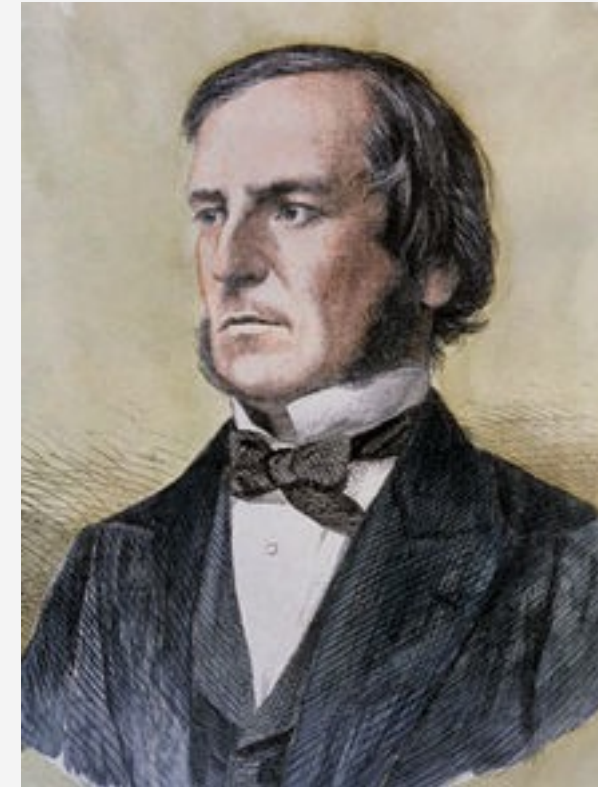
Buy a soft drink\* **and** any sandwich **or** hot snack **or** Deli Delights snack box or Saviour snack box **and** Boxerchips.

SAVE £1.80 / €2,50

\* Soft drinks included in this offer are: Princes Gate Water, Orangina, 7UP Free, Pepsi or Pepsi Max.

### Menu 3 £3.60 / 4€

Buy any hot drink **or** Orange **or** Apple Juice **and** a muffin **or** pastries trio.



Quelle: Wikipedia

# Recap: Boolesche Algebra



- Zwei Elemente: 0, 1
- Zwei binäre Verknüpfungen: UND ( $\cdot$ ), ODER (+)
- Eine unäre Verknüpfung: NICHT ( $\bar{a}$ )
- Axiome:
  - Neutrale Elemente:  $a \cdot 1 = a$   $a + 0 = a$
  - Komplementäre Elemente:  $a \cdot \bar{a} = 0$   $a + \bar{a} = 1$
  - Kommutativgesetze:  $a \cdot b = b \cdot a$   $a + b = b + a$
  - Distributivgesetze:  $(a + b) \cdot c = a \cdot c + b \cdot c$ ,  $a + (b \cdot c) = (a + b)(a + c)$
- UND hat höhere Priorität als ODER:  $a + b \cdot c = a + (b \cdot c)$

# Boolesche Algebra: Weitere Gesetze



- Idempotenzgesetze

- $a \cdot a = a + a = a$

- Assoziativgesetze

- $a \cdot (b \cdot c) = (a \cdot b) \cdot c$

- $a + (b + c) = (a + b) + c$

- De Morgansche Gesetze

- $\overline{a + b} = \bar{a} \cdot \bar{b}$

- $\overline{a \cdot b} = \bar{a} + \bar{b}$

- Und viele mehr...

- $a + (a \cdot b) = a, a \cdot (a + b) = a$

- ...

a	b	$\overline{a + b}$	$\bar{a} \cdot \bar{b}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

a	b	$\overline{a \cdot b}$	$\bar{a} + \bar{b}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

## 1. Kombinatorische Logik (Schaltnetze)

- Logikgatter
- Boolesche Algebra
- Logiksynthese (Schaltnetzentwurf)
- Minimierung von Schaltungen

## 2. Sequentielle Logik (Schaltwerke)

- Speicherelemente (Flipflops, Latches)
- Schaltwerksentwurf

## 3. Wichtige Schaltungen

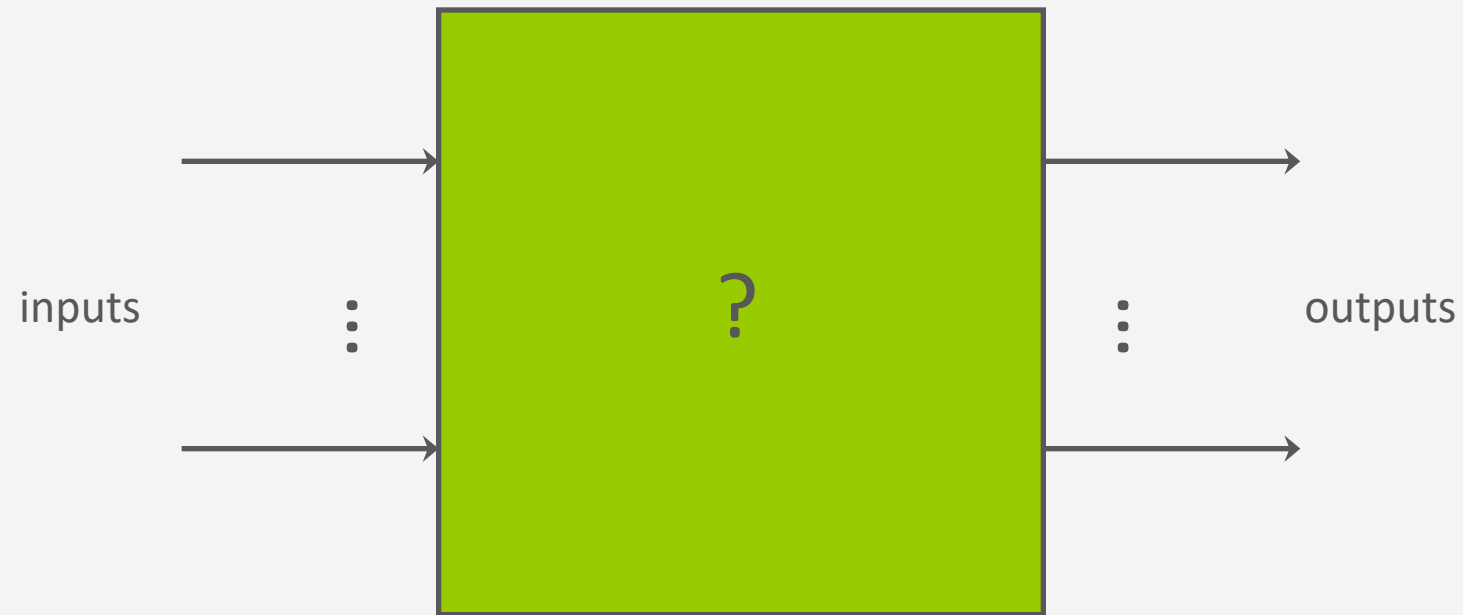
- Multiplexer und Decoder
- Registersatz
- Arithmetisch-Logische Einheit (ALU)
- Ripple-Carry und Carry-Look-Ahead Addierer



# Implementierung Boolescher Funktionen



Hochschule für  
Wirtschaft und Recht Berlin  
Berlin School of Economics and Law



# Spezifikation einer Funktion

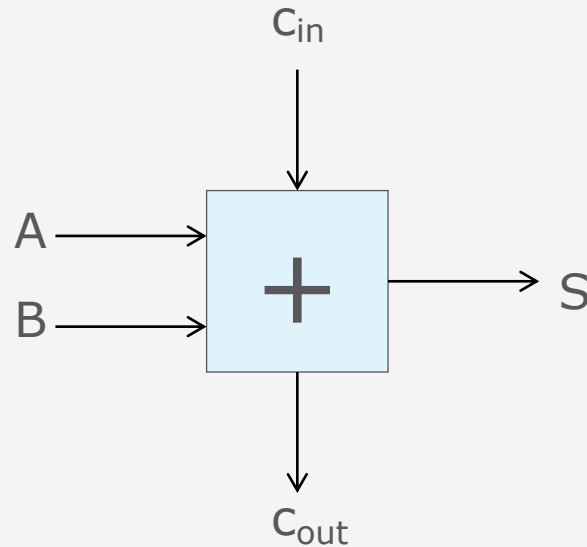


- $output_A = F_A(input_0, input_2, \dots, input_{n-1})$
- $output_B = F_B(input_0, input_2, \dots, input_{n-1})$
- $output_C = F_C(input_0, input_2, \dots, input_{n-1})$
- ...
  
- Methoden:
  - Wahrheitstabelle
  - Disjunktive Normalform (DNF)
  - Konjunktive Normalform (KNF)

# Beispiel: Wahrheitstabelle



Ein Volladdierer addiert drei  
einstellige Dualzahlen.

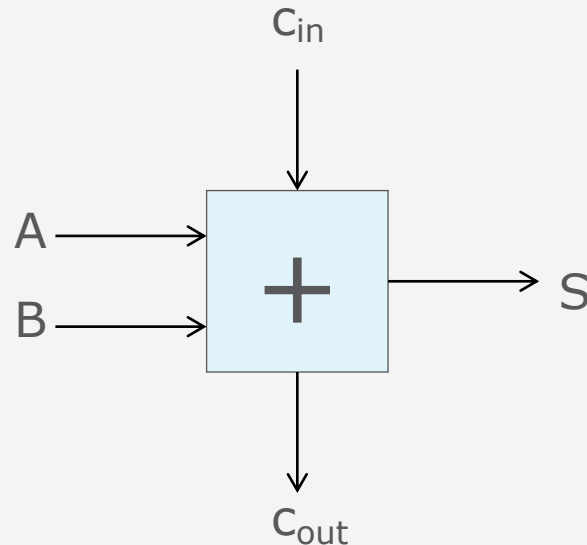


**Stellen Sie die  
Wahrheitstabelle auf!**

# Beispiel: Wahrheitstabelle



Ein Volladdierer addiert drei einstellige Dualzahlen.



**Stellen Sie die  
Wahrheitstabelle auf!**

Inputs			Outputs	
A	B	C <sub>in</sub>	C <sub>out</sub>	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



- Disjunktive Normalform (DNF)

- Disjunktion = ODER-Verknüpfung
- DNF  $\equiv$  Disjunktive Verknüpfung von Konjunktionstermen (*sum of products*)
- Beispiel:  $A \cdot \overline{B} + B \cdot \overline{C} + A \cdot B \cdot C + \overline{A}$

- Konjunktive Normalform

- Konjunktion = UND-Verknüpfung
- KNF  $\equiv$  Konjunktive Verknüpfung von Disjunktionstermen (*product of sums*)
- Beispiel:  $(A + \overline{B} + C) \cdot (A + \overline{C}) \cdot B$

# Ableitung der disjunktiven Normalform



Inputs			Outputs	
A	B	C <sub>in</sub>	C <sub>out</sub>	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Jede **Zeile** der Tabelle entspricht einer **Konjunktion**:
  - negierter Eingang bei **0**,  
unveränderter Eingang bei **1**
- **Ableitung DNF**:  
Verknüpfe Konjunktionen, bei denen der **Ausgang 1** ist, mit einer **Disjunktion**

$$C_{out} = (\bar{A} \cdot B \cdot C_{in}) + (A \cdot \bar{B} \cdot C_{in}) + (A \cdot B \cdot \overline{C_{in}}) + (A \cdot B \cdot C_{in})$$

$$S = (\bar{A} \cdot \bar{B} \cdot C_{in}) + (\bar{A} \cdot B \cdot \overline{C_{in}}) + (A \cdot \bar{B} \cdot \overline{C_{in}}) + (A \cdot B \cdot C_{in})$$

# Ableitung der konjunktiven Normalform



- jetzt lesen wir Terme ab, bei denen der Ausgang 0 wird und nutzen De Morgan
- De Morgan:  $\overline{a + b} = \overline{a} \cdot \overline{b}$  und  $\overline{a \cdot b} = \overline{a} + \overline{b}$

A	B	F
0	0	0
0	1	1
1	0	0
1	1	1

$$\overline{F} = (\overline{A} \cdot \overline{B}) + (A \cdot \overline{B})$$

$$F = \overline{(\overline{A} \cdot \overline{B}) + (A \cdot \overline{B})}$$

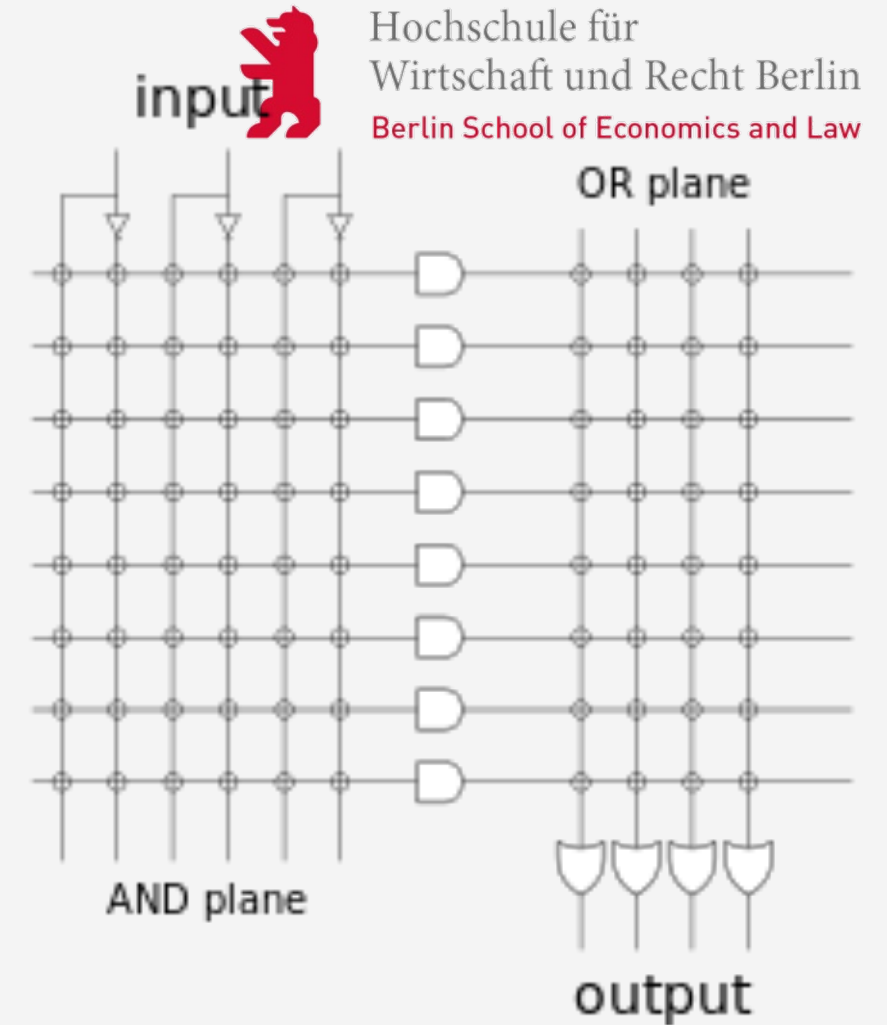
$$F = (A + B) \cdot (\overline{A} + B)$$

- Jede Zeile der Tabelle entspricht einer Disjunktion:
  - negierter Eingang bei 1, unveränderter Eingang bei 0
- Ableitung KNF:  
Verknüpfe Disjunktionen, bei denen der Ausgang 0 ist, mit einer Konjunktion

# Hauptsatz der Schaltalgebra

Jede einwertige Boolesche Funktion kann als Disjunktion von Konjunktionen dargestellt werden.

- Es gibt immer ein Schaltnetz, das nur aus NICHT, UND und ODER-Gattern besteht
- Maximal 3 Stufen: NICHT-Gatter, dann UND-Gatter, schließlich ein ODER-Gatter mit vielen Eingängen
- NICHT, UND und ODER bilden einen funktional vollständigen Operatorensatz



Programmierbare Logische Anordnung (PLA)  
*Programmable Logic Array*  
Quelle: Wikipedia



# Funktional vollständige Operatorensätze



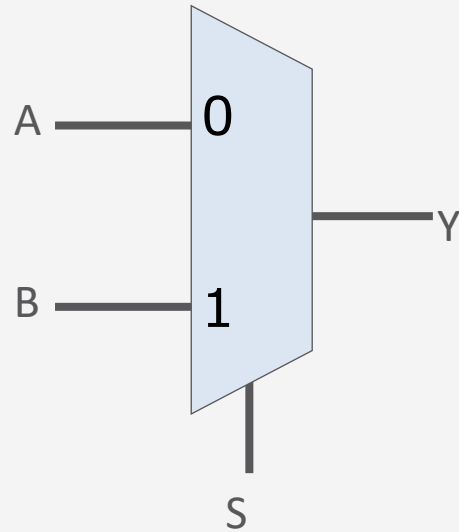
	NICHT	UND	ODER
Boolesche Basis	$\bar{a}$	$a \cdot b$	$a + b$
NICHT, UND	$\bar{a}$	$a \cdot b$	$\overline{\bar{a} \cdot \bar{b}}$
NICHT, ODER	Übung		
NAND	Übung		
NOR	Übung		

$a$	$b$	$\bar{a} \cdot \bar{b}$	$\overline{\bar{a} \cdot \bar{b}}$	$a + b$
0	0	1	0	0
0	1	0	1	1
1	0	0	1	1
1	1	0	1	1

# Multiplexer (MUX)



- Wählt je nach Steuersignal S einen der Inputs A oder B aus.



$$Y = (S) ? B : A;$$

S	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- Ableitung der DNF aus der Wahrheitstabelle

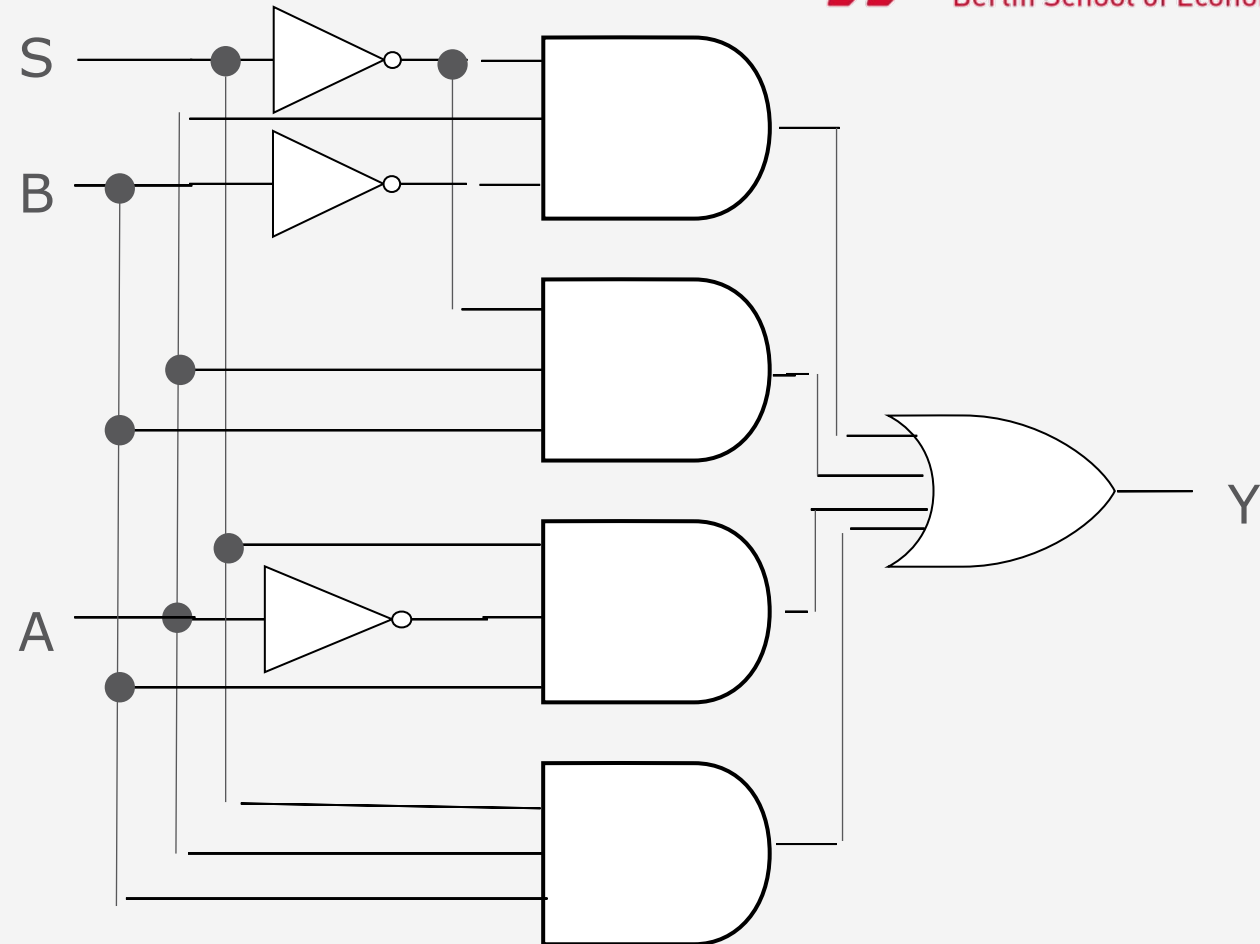
S	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$Y = \overline{S} \cdot A \cdot \overline{B} + \overline{S} \cdot A \cdot B + S \cdot \overline{A} \cdot B + S \cdot A \cdot B$$

# MUX-Schaltnetz



- Boolesche Gleichung:  
$$Y = \bar{S} \cdot A \cdot \bar{B} + \bar{S} \cdot A \cdot B + S \cdot \bar{A} \cdot B + S \cdot A \cdot B$$
- Jedes Gatter **erhöht den Platzbedarf**
- Jeder Gatter-Eingang **erhöht Platzbedarf und Gatterlaufzeit**



**Geht das auch besser?**

# MUX-Schaltnetz



- Boolesche Gleichung:

$$Y = \bar{S} \cdot A \cdot \bar{B} + \bar{S} \cdot A \cdot B \\ + S \cdot \bar{A} \cdot B + S \cdot A \cdot B$$

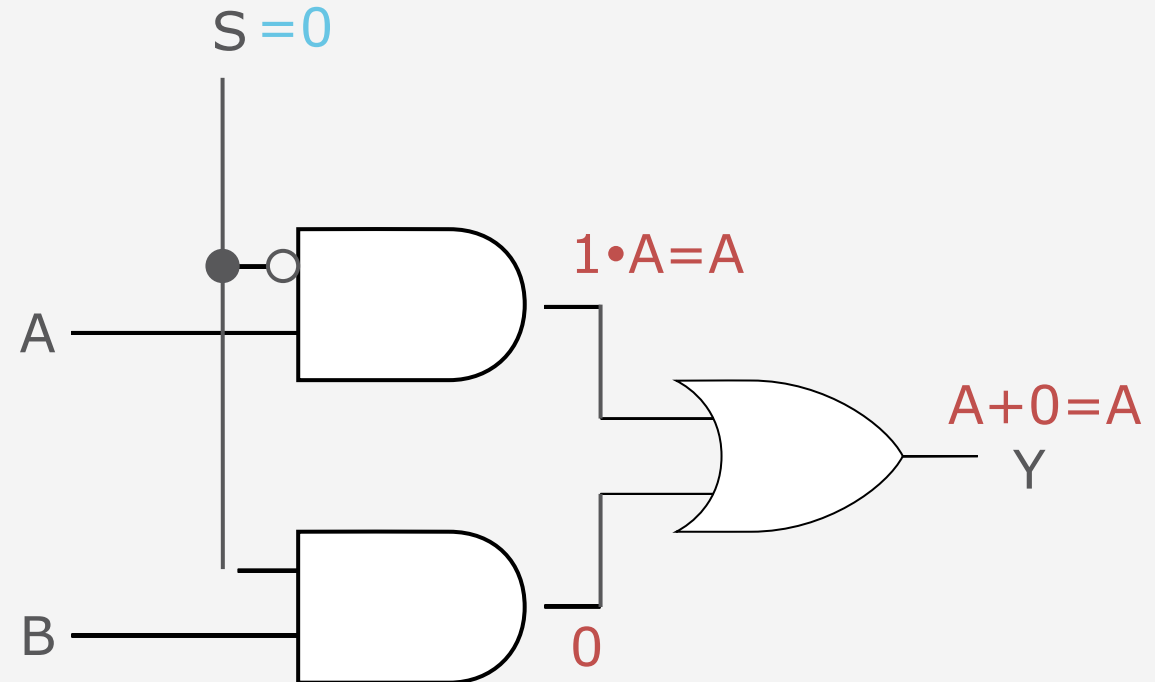
- Distributivgesetz:

$$(a + b) \cdot c = a \cdot c + b \cdot c$$

- Komplementäre Elemente:

$$a + \bar{a} = 1$$

- Vereinfacht:  $Y = \bar{S} \cdot A + S \cdot B$



# MUX-Schaltnetz



- Boolesche Gleichung:

$$Y = \bar{S} \cdot A \cdot \bar{B} + \bar{S} \cdot A \cdot B \\ + S \cdot \bar{A} \cdot B + S \cdot A \cdot B$$

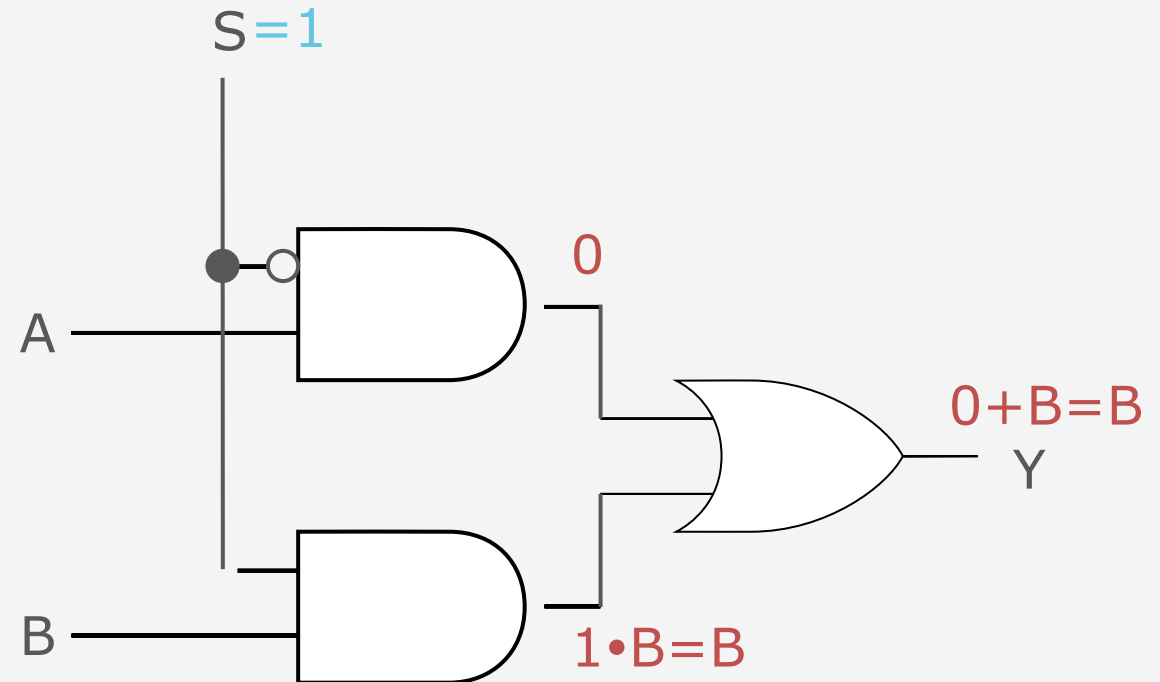
- Distributivgesetz:

$$(a + b) \cdot c = a \cdot c + b \cdot c$$

- Komplementäre Elemente:

$$a + \bar{a} = 1$$

➤ Vereinfacht:  $Y = \bar{S} \cdot A + S \cdot B$



# Don't-Care



- **Don't-Care** (\* oder -) bedeutet, dass dieser Wert keinen Einfluss auf die Logikschaltung hat  $\Rightarrow$  kompaktere Wahrheitstabelle möglich!

Vollständige  
Wahrheitstabelle

S	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Wahrheitstabelle mit Don't-Cares

S	A	B	Y
0	0	*	0
0	1	*	1
1	*	0	0
1	*	1	1



## 1. Kombinatorische Logik (Schaltnetze)

- Logikgatter
- Boolesche Algebra
- Logiksynthese (Schaltnetzentwurf)
- Minimierung von Schaltungen

## 2. Sequentielle Logik (Schaltwerke)

- Speicherelemente (Flipflops, Latches)
- Schaltwerksentwurf

## 3. Wichtige Schaltungen

- Multiplexer und Decoder
- Registersatz
- Arithmetisch-Logische Einheit (ALU)
- Ripple-Carry und Carry-Look-Ahead Addierer



- Ziel: Reduzierung der Gatter bei gleicher Funktionalität
  - früher wichtigster Kostenfaktor
  - heute werden Verbindungsleitungen und Energiebetrachtungen immer wichtiger
- Beispiel: Volladdierer
  - addiert A, B und Carry in ( $C_{in}$ ), liefert Carry out ( $C_{out}$ )\*
  - $C_{out} = \bar{A} \cdot B \cdot C_{in} + A \cdot \bar{B} \cdot C_{in} + A \cdot B \cdot \bar{C}_{in} + A \cdot B \cdot C_{in}$
  - $C_{out} = B \cdot C_{in} + A \cdot C_{in} + A \cdot B$

\* 2. Ausgang für die Summe S hier nicht betrachtet

A	B	$C_{in}$	$C_{out}$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



- **Manuelle Minimierung**
  - Umformen nach den Regeln der Booleschen Algebra
- **Algorithmisches Verfahren**
  - Verfahren nach Quine/McCluskey
  - Kann mit einem Programm automatisiert durchgeführt werden
  - Geeignet für Schaltfunktionen mit vielen Variablen
- **Graphische Verfahren**
  - Händlerscher Kreisgraph
  - **Karnaugh-Veitch Diagramme**
  - Geeignet für Schaltfunktionen mit wenig Variablen

# Karnaugh-Veitch-Diagramm



- Ziel: unnötige Terme und Variablen in einer DNF entfernen

⇒ Minimierung des Schaltnetzes, das die Funktion realisiert

## Idee

- graphische Darstellung aller möglichen Eingangskombinationen
- Anordnung der Felder so, dass benachbarte Felder zusammenfassbar sind

für zweistellige Schaltfunktionen  $f(x_1, x_2)$ :

⇒ jedes Feld entspricht einem Term  
der (kanonischen) DNF

	$\bar{x}_1$	$x_1$
$\bar{x}_2$	$\bar{x}_1 \cdot \bar{x}_2$	$x_1 \cdot \bar{x}_2$
$x_2$	$\bar{x}_1 \cdot x_2$	$x_1 \cdot x_2$

# Karnaugh-Veitch-Diagramm



- Jede Variable  $x_i$  halbiert das Diagramm in zwei zusammenhängende Teile

⇒ eine Hälfte für  $\bar{x}_i$ , die zweite Hälfte für  $x_i$

- Variable  $x_1$ :

	$\bar{x}_1$	$x_1$
$\bar{x}_2$	$\bar{x}_1 \cdot \bar{x}_2$	$x_1 \cdot \bar{x}_2$
$x_2$	$\bar{x}_1 \cdot x_2$	$x_1 \cdot x_2$

- Variable  $x_2$ :

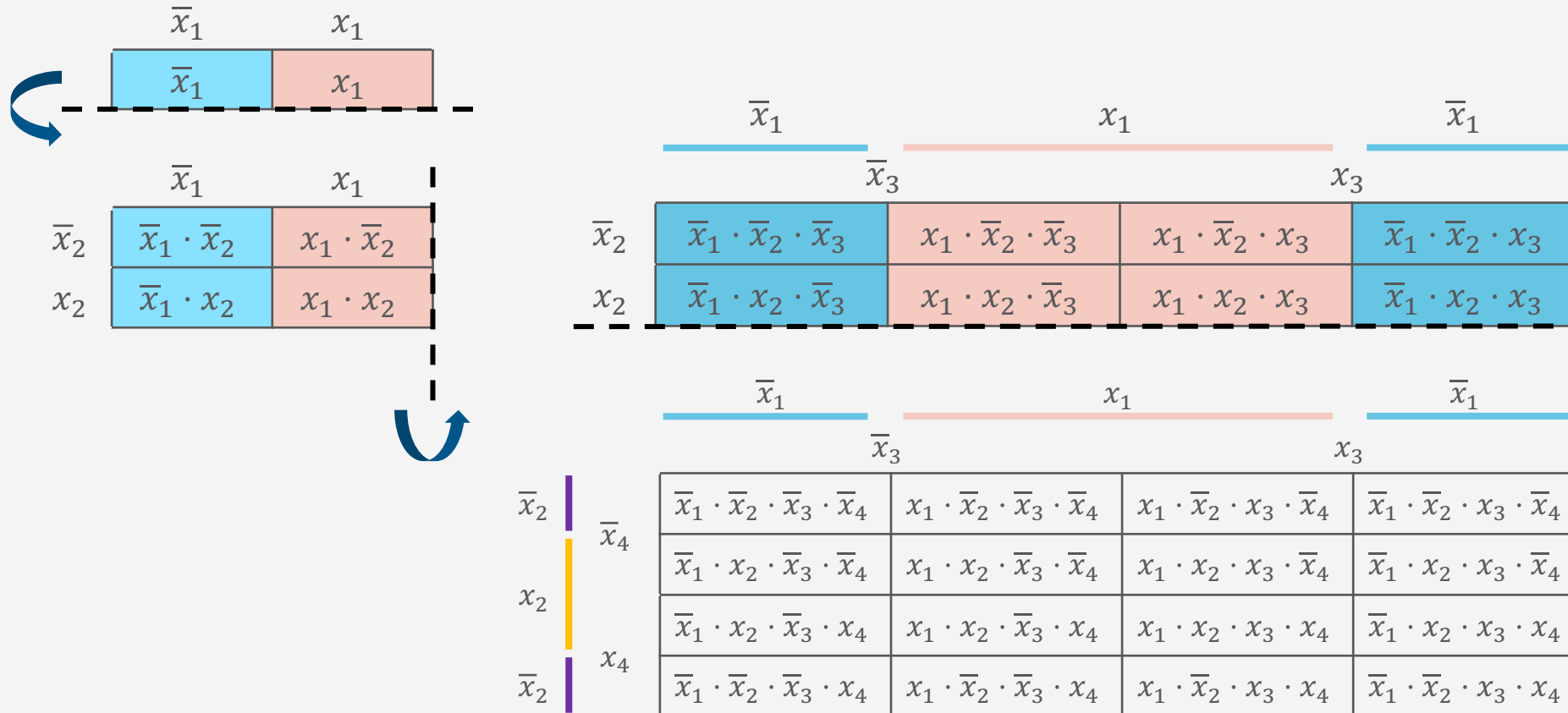
	$\bar{x}_1$	$x_1$
$\bar{x}_2$	$\bar{x}_1 \cdot \bar{x}_2$	$x_1 \cdot \bar{x}_2$
$x_2$	$\bar{x}_1 \cdot x_2$	$x_1 \cdot x_2$

- Benachbarte Felder unterscheiden sich nur in einer Variablen (in einem Bit)

# Konstruktion von Karnaugh-Veitch-Diagramm



- Diagramme für drei und vier Variablen erhält man durch „Spiegeln“ des Diagramms



- nach wie vor unterscheiden sich alle benachbarten Felder nur um ein Bit (auch über die Ränder hinaus)
- die Reihenfolge der Variablen spielt keine Rolle

# KV-Diagramme: Beispiel



## ■ Beispiel: Volladdierer

A	B	C <sub>in</sub>	C <sub>out</sub>
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

	$\bar{x}_2$	$x_2$	$\bar{x}_2$
	$\bar{x}_1$	$x_1$	
$\bar{x}_3$	$\bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3$	$\bar{x}_1 \cdot x_2 \cdot \bar{x}_3$	$x_1 \cdot x_2 \cdot \bar{x}_3$
$x_3$	$\bar{x}_1 \cdot \bar{x}_2 \cdot x_3$	$\bar{x}_1 \cdot x_2 \cdot x_3$	$x_1 \cdot \bar{x}_2 \cdot x_3$

	$\bar{A} \cdot \bar{B}$	$\bar{A} \cdot B$	$A \cdot B$	$A \cdot \bar{B}$
$\bar{C}_{in}$	0	0	1	0
$C_{in}$	0	1	1	1

Idee: Zusammenfassung benachbarter Felder, die eine 1 enthalten

⇒ Bildung von möglichst wenigen, möglichst großen Rechtecken mit  $2^k$  Feldern (also Zweierpotenzen, bei k Eingängen höchstens  $2^k$ ), in denen nur Einsen vorkommen

- Die Rechtecke können über den Rand hinauslaufen und sich gegenseitig überlappen.
- Alle Felder mit 1 müssen zu einem Rechteck gehören.
- Felder mit einem nicht definierten Funktionswert („Don't care“) können wahlweise als 1 oder 0 interpretiert werden.

Beispiel Volladdierer:

$$C_{out} = \bar{A} \cdot B \cdot C_{in} + A \cdot \bar{B} \cdot C_{in} + A \cdot B \cdot \bar{C}_{in} + A \cdot B \cdot C_{in}$$

	$\bar{A} \cdot \bar{B}$	$\bar{A} \cdot B$	$A \cdot B$	$A \cdot \bar{B}$
$\bar{C}_{in}$	0	0	1	0
$C_{in}$	0	1	1	1

# KV-Minimierung: Beispiel



Beispiel Volladdierer:

- Gruppe:  $A \cdot B$

- vollständig im Bereich von B  $\Rightarrow$  B gehört zum Term dieser Gruppe
- vollständig im Bereich von A  $\Rightarrow$  A gehört zum Term dieser Gruppe
- erstreckt sich über  $C_{in}$  und  $\bar{C}_{in}$   $\Rightarrow C_{in}$  gehört nicht zum Term dieser Gruppe

- analog für die anderen Gruppen

Minimierter Funktionsterm:  $C_{out} = A \cdot B + B \cdot C_{in} + A \cdot C_{in}$

	$\bar{A} \cdot \bar{B}$	$\bar{A} \cdot B$	$A \cdot B$	$A \cdot \bar{B}$
$\bar{C}_{in}$	0	0	1	0
$C_{in}$	0	1	1	1



# KV-Minimierung: fiktive Beispiele



	$\overline{A} \cdot \overline{B}$	$\overline{A} \cdot B$	$A \cdot B$	$A \cdot \overline{B}$
$\overline{C}$	1	0	0	1
$C$	0	0	1	1

		$\overline{x_2}$	$x_2$	$\overline{x_2}$
		$\overline{x_1}$		$x_1$
$\overline{x_4}$	$\overline{x_3}$	1	0	0
	$x_3$	1	0	1
$x_4$	$\overline{x_3}$	1	0	1
	$x_3$	1	1	1
$\overline{x_4}$	$\overline{x_3}$	1	0	0
$x_4$	$x_3$	1	0	1

Was ist jeweils die beste Markierung?

Geben Sie die minimierten Funktionsterme an!

Nicht vergessen:

- wie bei einer Weltkarte liegt ganz rechts neben ganz links
- das gleiche gilt für oben und unten bei größeren KV-Diagrammen

# KV-Minimierung: fiktives Beispiel 1/2



	$\overline{A} \cdot \overline{B}$	$\overline{A} \cdot B$	$A \cdot B$	$A \cdot \overline{B}$
$\overline{C}$	1	0	0	1
$C$	0	0	1	1

- orange Gruppe:  $A \cdot C$
- blaue Gruppe:  $\overline{B} \cdot \overline{C}$
- Funktionsterm:  $f = A \cdot C + \overline{B} \cdot \overline{C}$

# KV-Minimierung: fiktives Beispiel 2/2



- hellblaue Gruppe:  $x_3 \cdot x_4$
- orange Gruppe:  $\bar{x}_1 \cdot \bar{x}_2$
- rote Gruppe:  $x_1 \cdot x_2 \cdot x_4$
- blaue Gruppe:  $\bar{x}_2 \cdot \bar{x}_4$

	$\bar{x}_2$	$x_2$	$\bar{x}_2$	
	$\bar{x}_1$	$x_1$		
$\bar{x}_4$	1	0	0	1
$\bar{x}_3$	1	0	1	0
$x_4$	1	1	1	1
$x_3$	1	0	0	1
$\bar{x}_4$	1	0	0	1

- Funktionsterm:  $f = x_3 \cdot x_4 + \bar{x}_1 \cdot \bar{x}_2 + x_1 \cdot x_2 \cdot x_4 + \bar{x}_2 \cdot \bar{x}_4$

# KV-Minimierung: Mehrere Lösungen



- orange Gruppe:  $\bar{x}_1 \cdot \bar{x}_2$
- rote Gruppe:  $x_1 \cdot x_2$
- blaue Gruppe:  $\bar{x}_1 \cdot x_3 \cdot x_4$

ODER: hellblaue Gruppe:  $x_2 \cdot x_3 \cdot x_4$

- Funktionsterm:  $f = \bar{x}_1 \cdot \bar{x}_2 + x_1 \cdot x_2 + \bar{x}_1 \cdot x_3 \cdot x_4$

**ODER:**  $f = \bar{x}_1 \cdot \bar{x}_2 + x_1 \cdot x_2 + x_2 \cdot x_3 \cdot x_4$

	$\bar{x}_2$	$x_2$	$\bar{x}_2$	
	$\bar{x}_1$		$x_1$	
$\bar{x}_4$	1	0	1	0
$\bar{x}_3$	1	0	1	0
$x_4$	1	1	1	0
$x_3$	1	0	1	0
$\bar{x}_4$				

## 1. Kombinatorische Logik (Schaltnetze)

- Logikgatter
- Boolesche Algebra
- Logiksynthese (Schaltnetzentwurf)
- Minimierung von Schaltungen

## 2. Sequentielle Logik (Schaltwerke)

- Speicherelemente (Flipflops, Latches)
- Schaltwerksentwurf

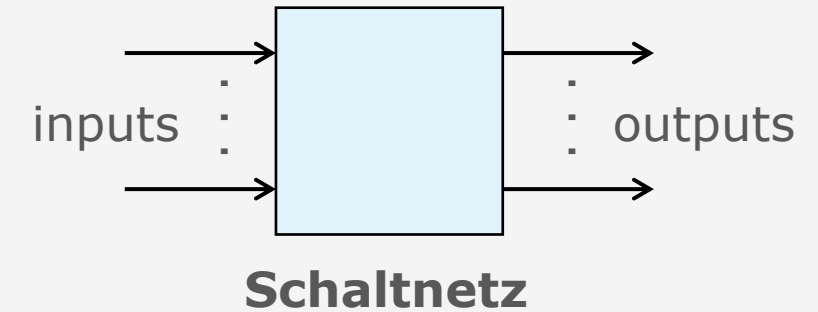
## 3. Wichtige Schaltungen

- Multiplexer und Decoder
- Registersatz
- Arithmetisch-Logische Einheit (ALU)
- Ripple-Carry und Carry-Look-Ahead Addierer

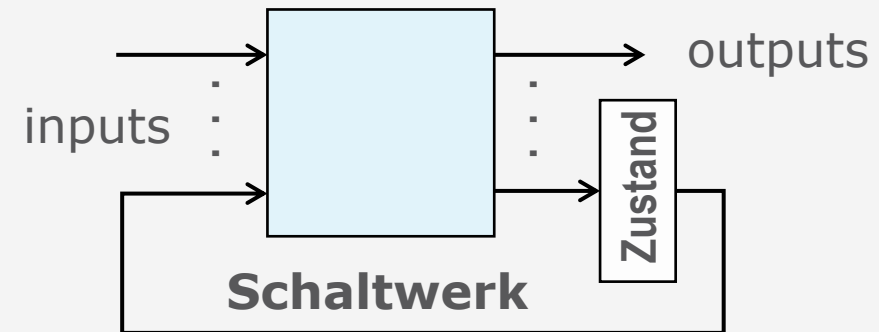
# Schaltnetz vs. Schaltwerk



- Bisher: **Schaltnetze** (*combinatorial circuits*):
  - ohne Rückkopplung
  - Ausgangssignale **hängen nur von Eingangssignalen ab**
  - eindeutig beschrieben durch (Boolesche) Schaltfunktion
  - **Speichern nicht möglich**
  - z.B. MUX, ALU, ...



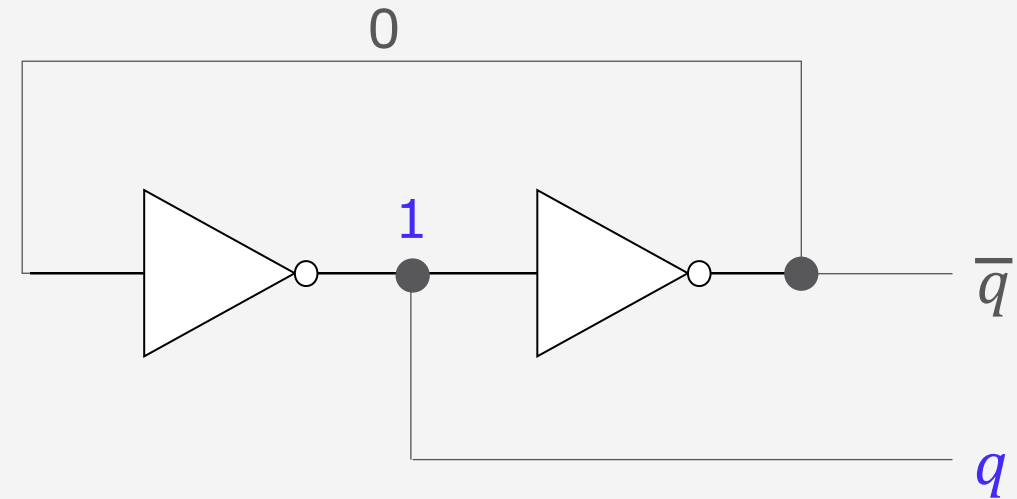
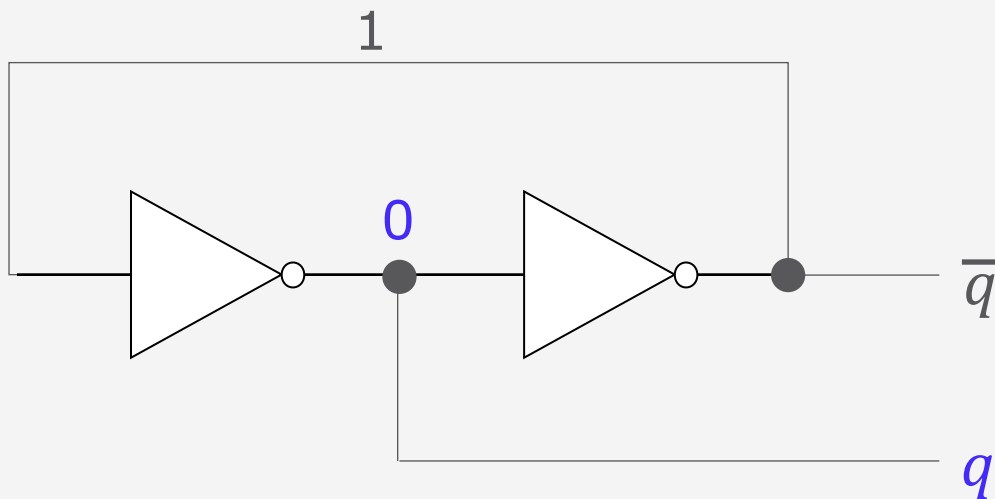
- **Schaltwerke** (*sequential circuits*):
  - mit Rückkopplung
  - **Speichern von Zuständen** möglich
  - Ausgänge hängen ab von Eingängen **und vom Inhalt des internen Speichers**
  - zustandsabhängige Schaltfunktion
  - z.B. Register, Speicher,...



# Speicher mit rückgekoppelten Gattern



- Durch Rückkopplung können Werte gespeichert werden
- Zwei Inverter bilden eine statische Speicherzelle
- Wert wird gespeichert, solange Versorgungsspannung anliegt



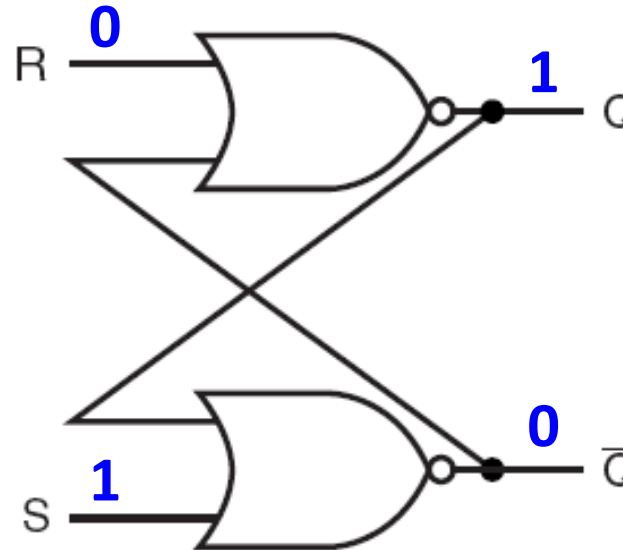
- **Problem:** der Wert ist immer gleich (es kann kein neuer Wert gesetzt werden)

# RS Flipflop



- Zwei rückgekoppelte NOR Gatter
- mit  $S = 1$  wird die Speicherzelle auf 1 gesetzt (Set),  
mit  $R = 1$  wird sie auf 0 zurückgesetzt (Reset)

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0



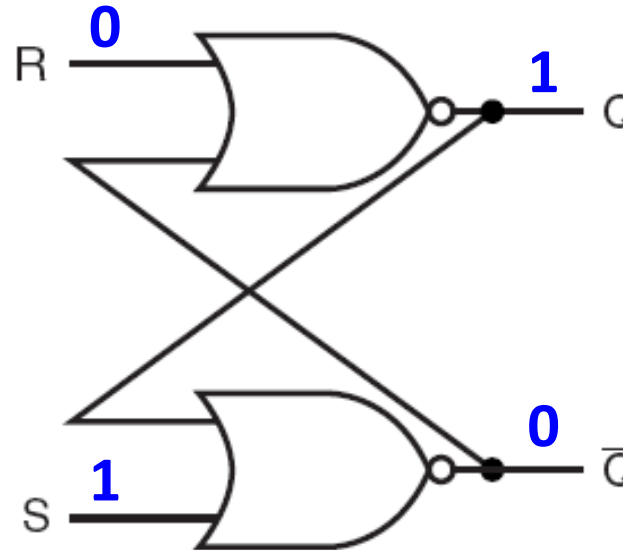


# RS Flipflop



- Zwei rückgekoppelte NOR Gatter
- mit  $S = 1$  wird die Speicherzelle auf 1 gesetzt (Set),  
mit  $R = 1$  wird sie auf 0 zurückgesetzt (Reset)

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0



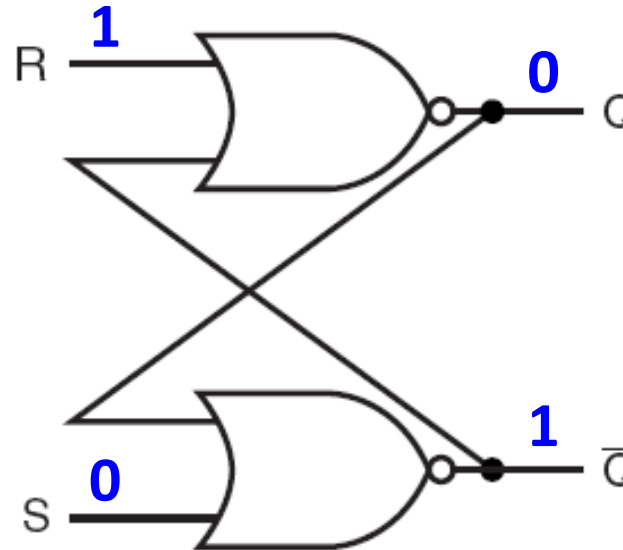
R	S	Q	$\bar{Q}$
0	0		
0	1	1	0
1	0		
1	1		

# RS Flipflop



- Zwei rückgekoppelte NOR Gatter
- mit  $S = 1$  wird die Speicherzelle auf 1 gesetzt (Set),  
mit  $R = 1$  wird sie auf 0 zurückgesetzt (Reset)

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0



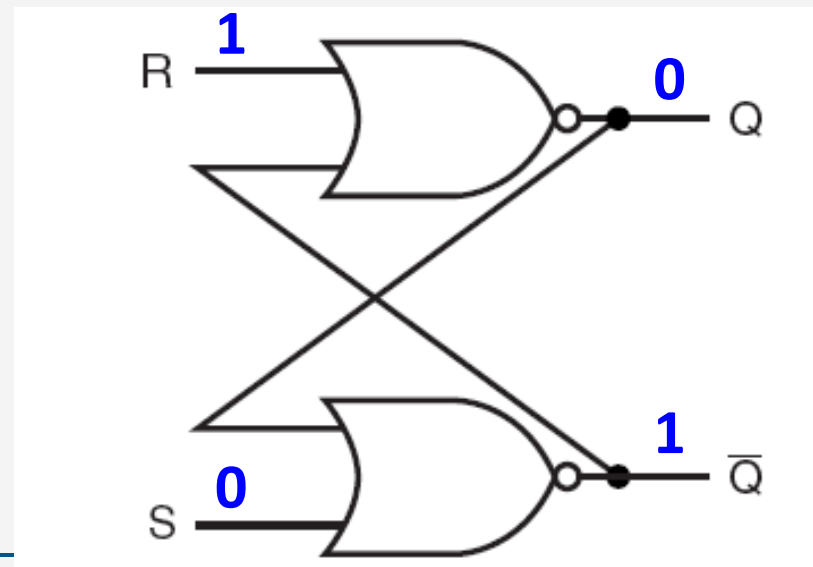
R	S	Q	$\bar{Q}$
0	0		
0	1	1	0
1	0		
1	1		

# RS Flipflop



- Zwei rückgekoppelte NOR Gatter
- mit  $S = 1$  wird die Speicherzelle auf 1 gesetzt (Set),  
mit  $R = 1$  wird sie auf 0 zurückgesetzt (Reset)

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0



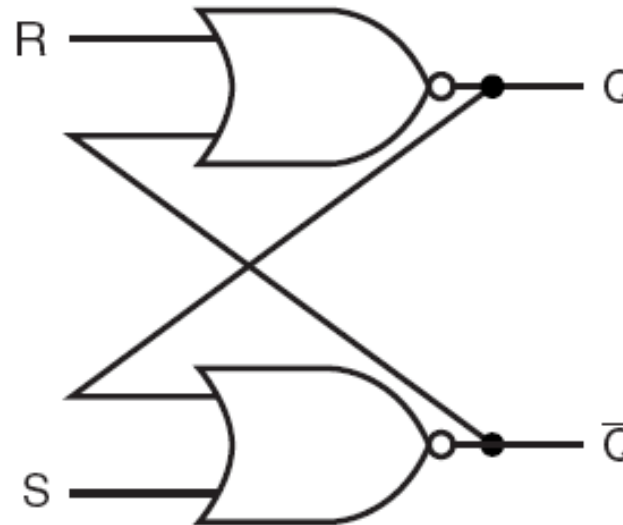
R	S	Q	$\bar{Q}$
0	0		
0	1	1	0
1	0	0	1
1	1		

# RS Flipflop



- Zwei rückgekoppelte NOR Gatter
- mit  $S = 1$  wird die Speicherzelle auf 1 gesetzt (Set),  
mit  $R = 1$  wird sie auf 0 zurückgesetzt (Reset)
- Was passiert für  $S = 0$  und  $R = 0$ ?
- Was passiert für  $S = 1$  und  $R = 1$ ? Ist das ein Problem?

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0



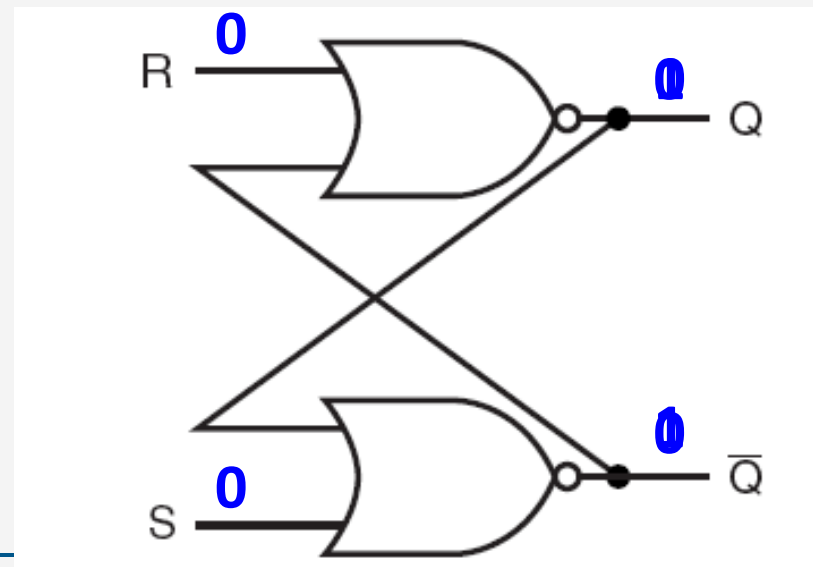
R	S	Q	$\bar{Q}$
0	0		
0	1	1	0
1	0	0	1
1	1		

# RS Flipflop



- Zwei rückgekoppelte NOR Gatter
- mit  $S = 1$  wird die Speicherzelle auf 1 gesetzt (Set),  
mit  $R = 1$  wird sie auf 0 zurückgesetzt (Reset)
- **Was passiert für  $S = 0$  und  $R = 0$ ?** Mit  $Q \equiv 0$  und  $\bar{Q} \equiv 0$

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0



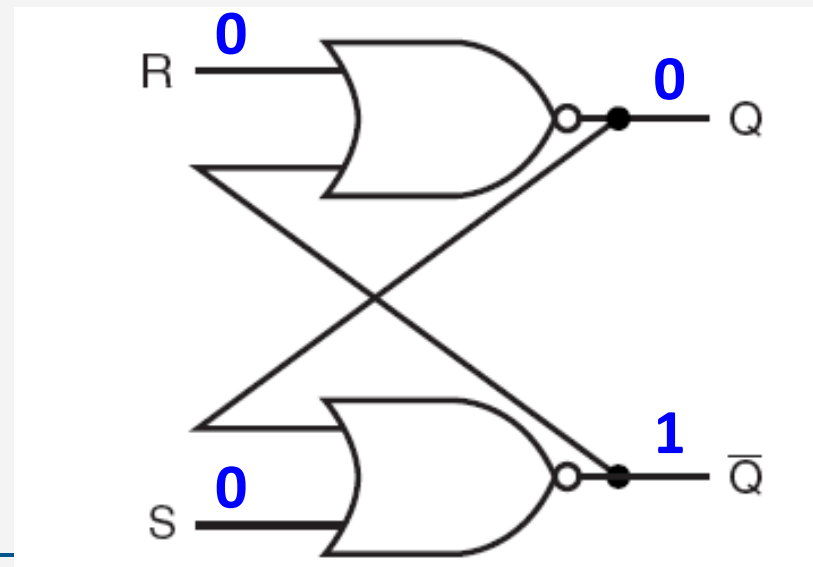
R	S	Q	$\bar{Q}$
0	0		
0	1	1	0
1	0	0	1
1	1		

# RS Flipflop



- Zwei rückgekoppelte NOR Gatter
- mit  $S = 1$  wird die Speicherzelle auf 1 gesetzt (Set),  
mit  $R = 1$  wird sie auf 0 zurückgesetzt (Reset)
- Was passiert für  $S = 0$  und  $R = 0$ ? Mit  $Q = 0$  und  $\bar{Q} = 1$

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0



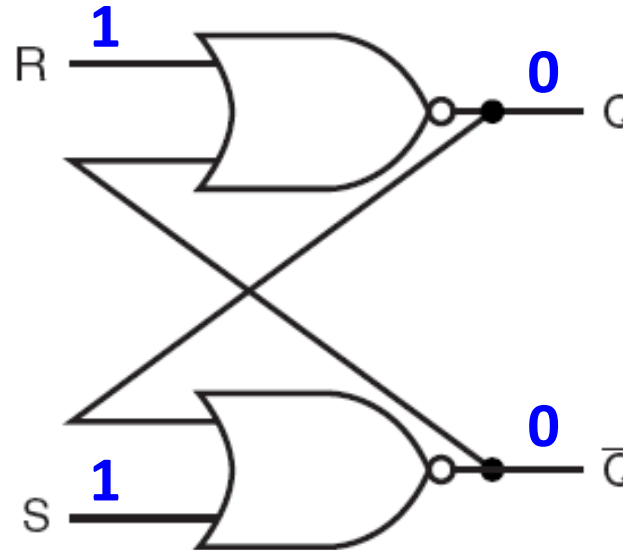
R	S	Q	$\bar{Q}$
0	0	$Q_{alt}$	$\bar{Q}_{alt}$
0	1	1	0
1	0	0	1
1	1		

# RS Flipflop



- Zwei rückgekoppelte NOR Gatter
- mit  $S = 1$  wird die Speicherzelle auf 1 gesetzt (Set),  
mit  $R = 1$  wird sie auf 0 zurückgesetzt (Reset)
- Was passiert für  $S = 0$  und  $R = 0$ ?
- **Was passiert für  $S = 1$  und  $R = 1$ ? Ist das ein Problem?**

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0



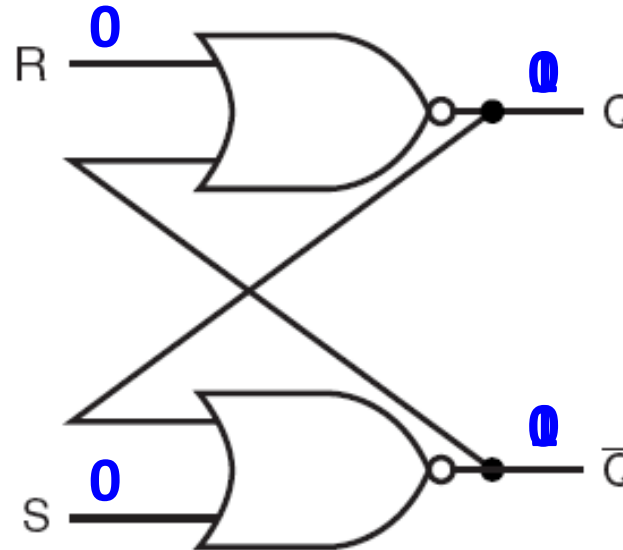
R	S	Q	$\bar{Q}$
0	0	$Q_{alt}$	$\bar{Q}_{alt}$
0	1	1	0
1	0	0	1
1	1		

# RS Flipflop



- Zwei rückgekoppelte NOR Gatter
- mit  $S = 1$  wird die Speicherzelle auf 1 gesetzt (Set),  
mit  $R = 1$  wird sie auf 0 zurückgesetzt (Reset)
- Was passiert für  $S = 0$  und  $R = 0$ ?
- **Was passiert für  $S = 1$  und  $R = 1$ ? Ist das ein Problem?**

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0



R	S	$Q$	$\bar{Q}$
0	0	$Q_{alt}$	$\bar{Q}_{alt}$
0	1	1	0
1	0	0	1
1	1		

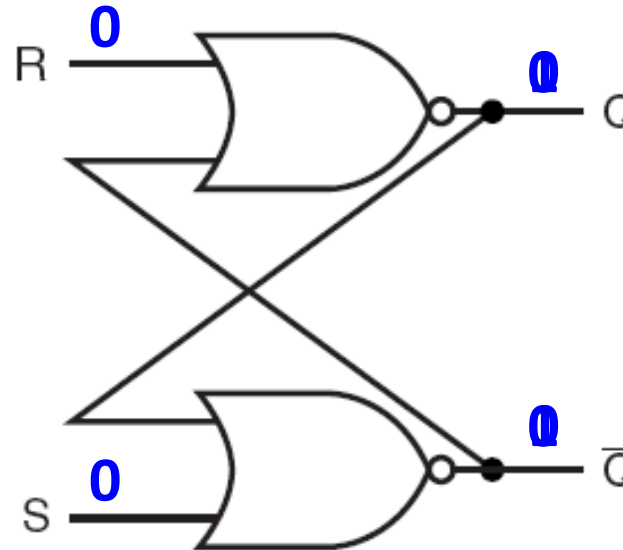


# RS Flipflop



- Zwei rückgekoppelte NOR Gatter
- mit  $S = 1$  wird die Speicherzelle auf 1 gesetzt (Set),  
mit  $R = 1$  wird sie auf 0 zurückgesetzt (Reset)
- Was passiert für  $S = 0$  und  $R = 0$ ?
- **Was passiert für  $S = 1$  und  $R = 1$ ? Ist das ein Problem?**

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0



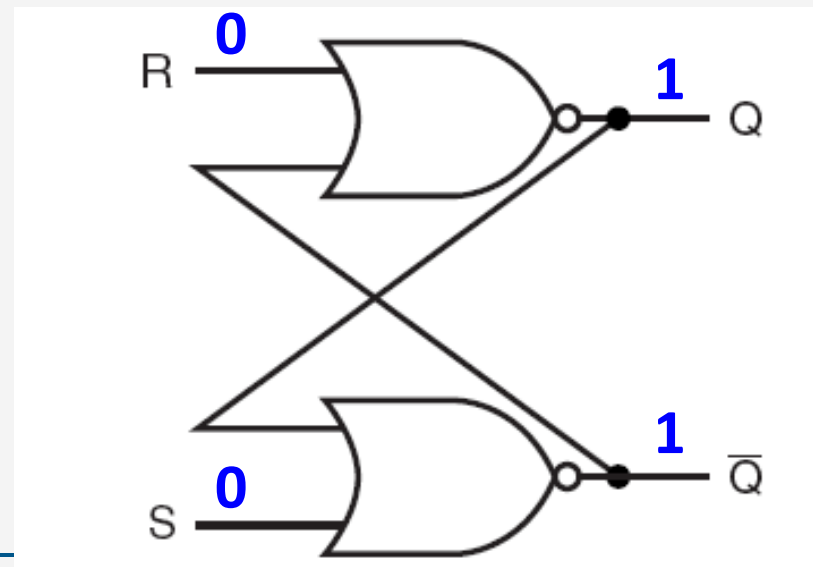
R	S	Q	$\bar{Q}$
0	0	$Q_{alt}$	$\bar{Q}_{alt}$
0	1	1	0
1	0	0	1
1	1		

# RS Flipflop



- Zwei rückgekoppelte NOR Gatter
- mit  $S = 1$  wird die Speicherzelle auf 1 gesetzt (Set),  
mit  $R = 1$  wird sie auf 0 zurückgesetzt (Reset)
- Was passiert für  $S = 0$  und  $R = 0$ ?
- **Was passiert für  $S = 1$  und  $R = 1$ ? Ist das ein Problem?**

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0

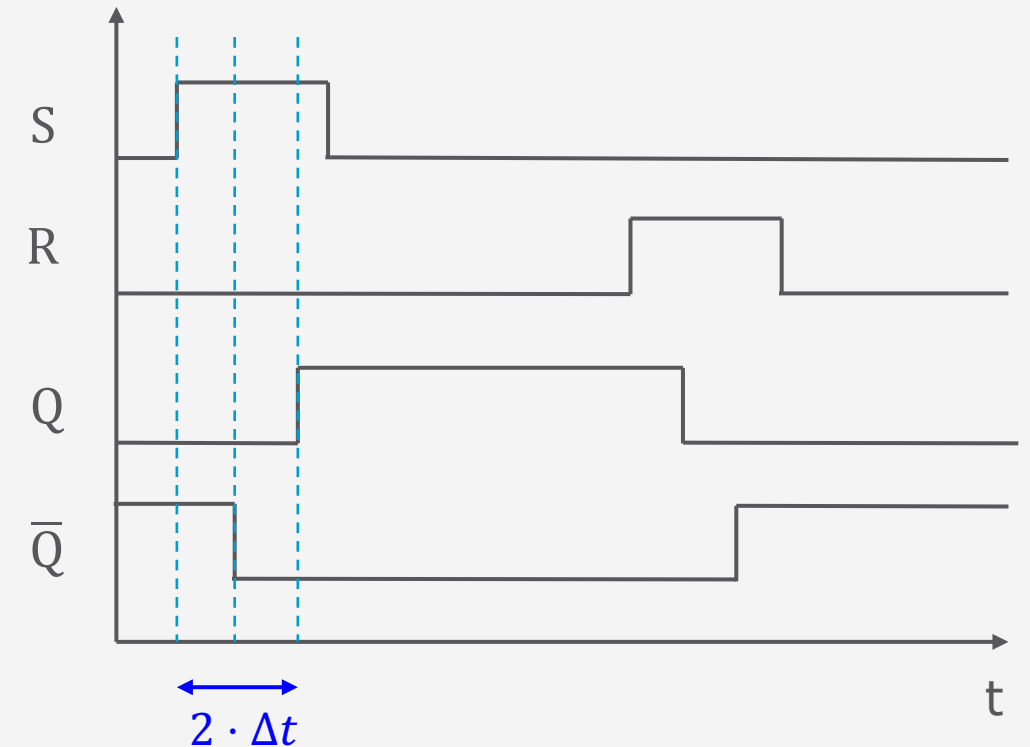
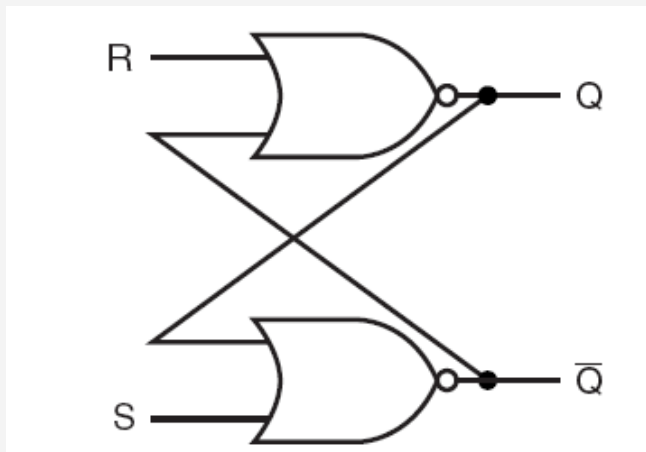


R	S	Q	$\bar{Q}$
0	0	$Q_{alt}$	$\bar{Q}_{alt}$
0	1	1	0
1	0	0	1
1	1	x	x

# RS Flipflop: Zeitverhalten



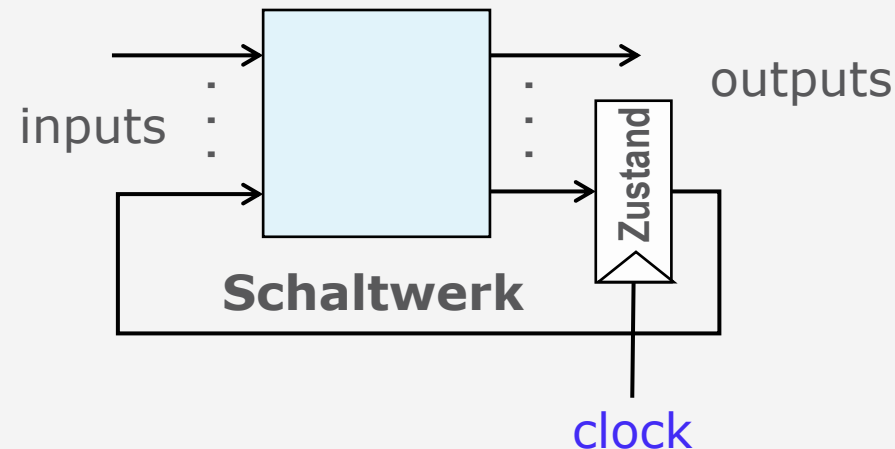
- Die bisher gezeigte Schaltung ist **asynchron**
- Entscheidend für Zeitverhalten ist die **Gatterlaufzeit  $\Delta t$**
- Ausgangssignal stabilisiert sich nach  $2\Delta t$
- **dies kann zu undefinierten Werten führen!**



# Synchrone vs. Asynchrone Schaltwerke



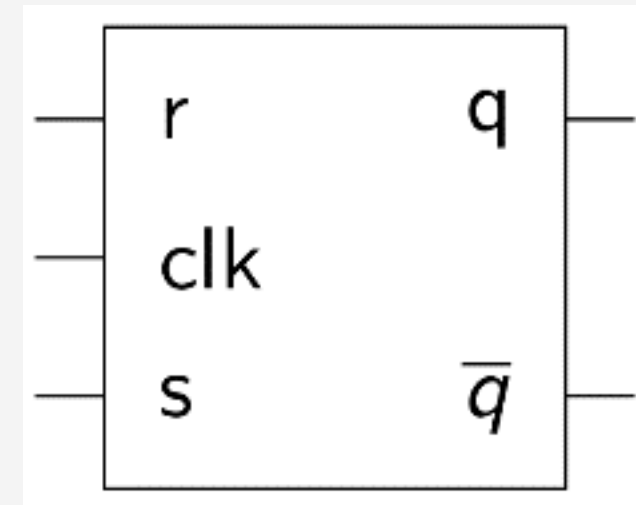
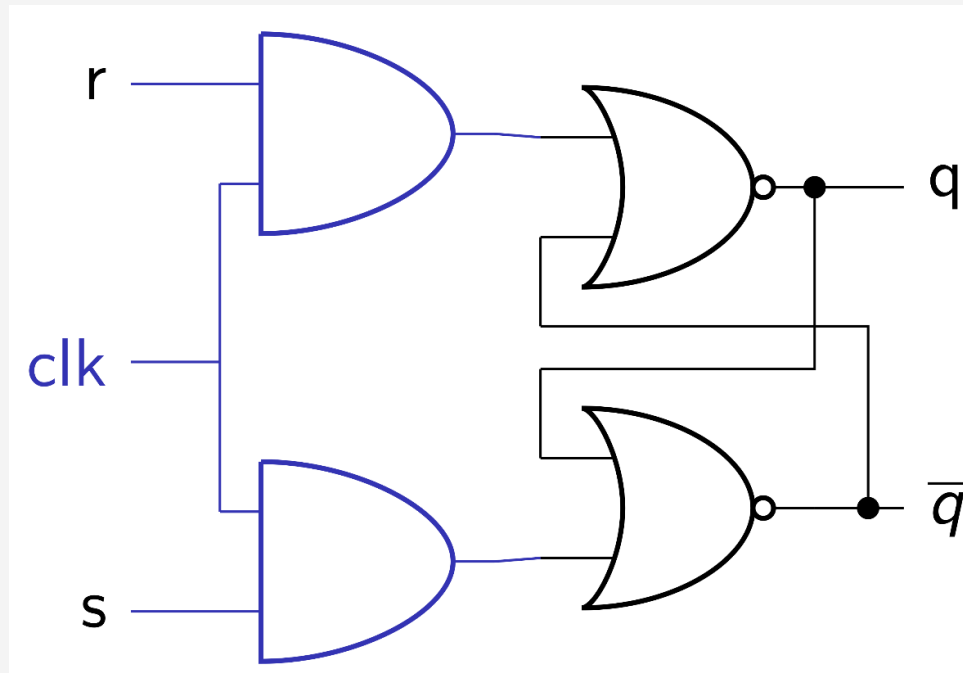
- Schaltwerke können **asynchron** oder **synchron** sein
- Synchronisierung erfolgt über **Taktsignale**
  - Taktsignal gibt an, **wann** der Zustand sich ändert
  - Zustandsänderungen nur zu **definierten Zeitpunkten**



# Getaktetes RS Flipflop



- Zustandswechsel (Wert) abhängig vom **Takt** (*Clock*)
- Änderungen am Eingang nur übernehmen, wenn  $\text{clk} = 1$

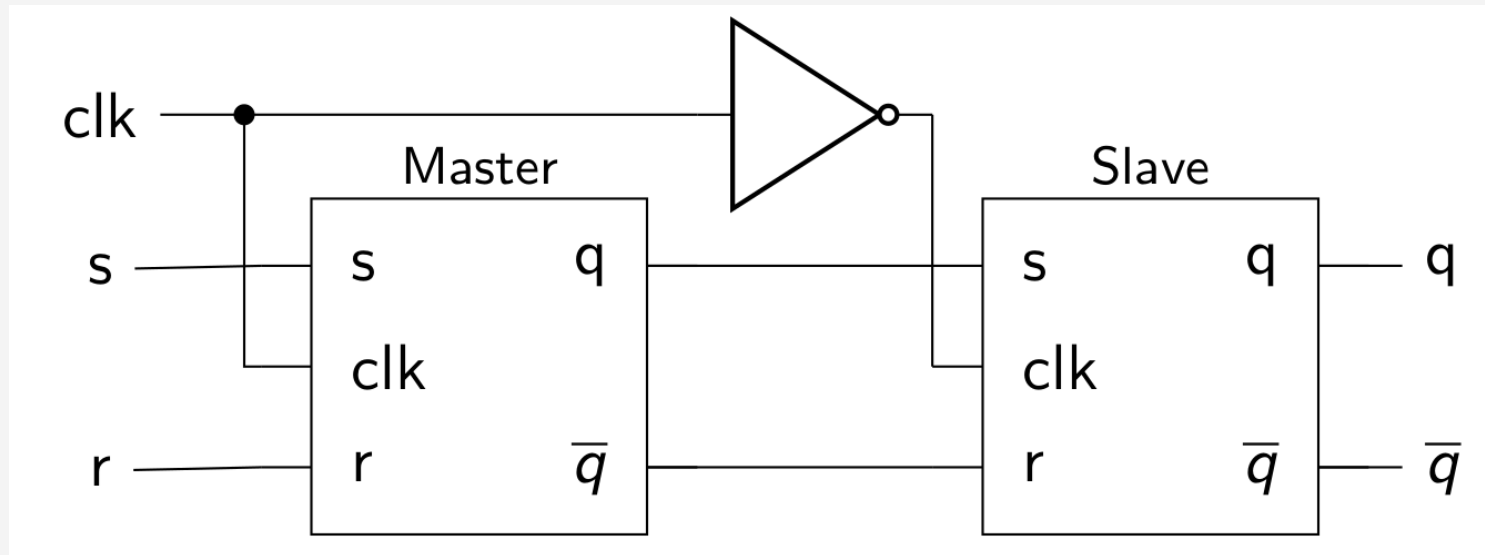


- **Problem:** mehrere Zustandsänderungen pro Takt möglich

# Master-Slave Flipflop



- besteht aus **zwei getakteten RS Flipflops** (Master und Slave)
- Slave erhält negiertes Taktsignal des Masters



- Master akzeptiert Änderungen bei  $clk = 1$ , Slave übernimmt den letzten Zustand wenn  $clk$  von 1 auf 0 wechselt (**flankengesteuertes Flipflop**)

## 1. Kombinatorische Logik (Schaltnetze)

- Logikgatter
- Boolesche Algebra
- Logiksynthese (Schaltnetzentwurf)
- Minimierung von Schaltungen

## 2. Sequentielle Logik (Schaltwerke)

- Speicherelemente (Flipflops, Latches)
- Schaltwerksentwurf

## 3. Wichtige Schaltungen

- Multiplexer und Decoder
- Registersatz
- Arithmetisch-Logische Einheit (ALU)
- Ripple-Carry und Carry-Look-Ahead Addierer



- Beschreibung des gewünschten Verhaltens z.B. **als endlicher Automat** gegeben
  - **Moore**-Automat: Ausgaben gekoppelt an Zustände
  - **Mealy**-Automat: Ausgaben als Funktion von Eingaben und Zustand

## Systematischer Schaltwerksentwurf (für Moore-Automaten)

1. **Zustandsdiagramm** mit binärer Zustandskodierung
2. **Wahrheitstabelle** der zustandsabhängigen Schaltfunktion aufstellen
3. ggf. Erweiterung der Wahrheitstabelle um **Ansteuerung der Flipflops**
4. **Minimierung** der Schaltfunktion
5. Aufbau der **Schaltung**

(für Mealy-Automaten analog, zusätzlich Definition und Minimierung der Ausgabefunktion)

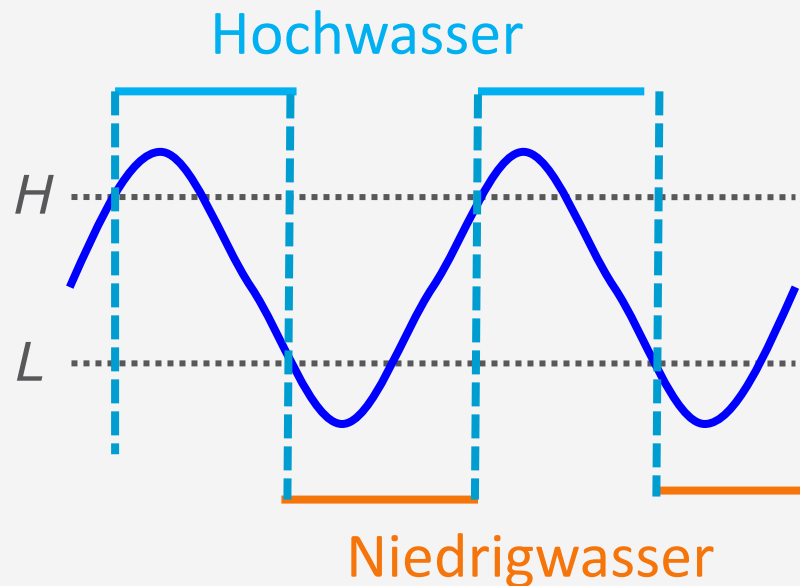


# Beispiel: Hochwassererkennung

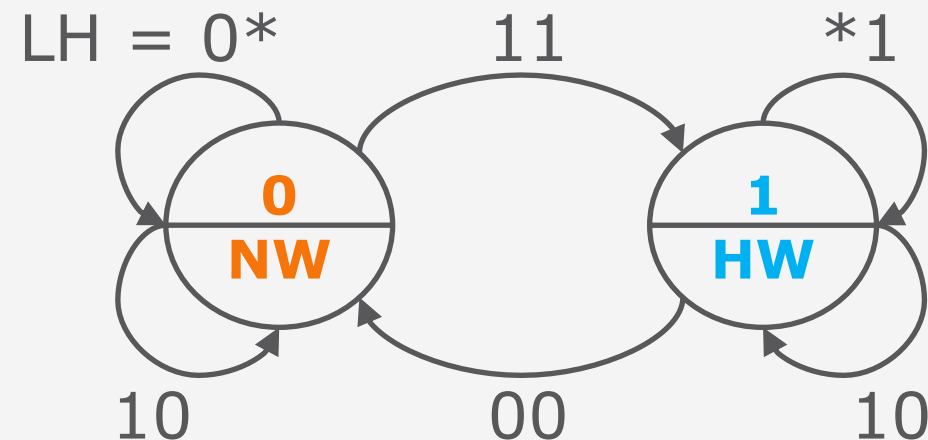


- **Ziel:** Wasserstandsanzeige (Hochwasser / Niedrigwasser)
- Gemessen wird mit zwei **Wasserstandssensoren H und L**
- Wasser oberhalb von H: Hochwasser, unterhalb von L: Niedrigwasser
- Dazwischen: alten Wert halten (stabile Anzeige)
- Verwendung eines RS Flipflops als Zustandsspeicher und Ausgabe

L=1 → Wasser über L,  
H=1 → Wasser über H



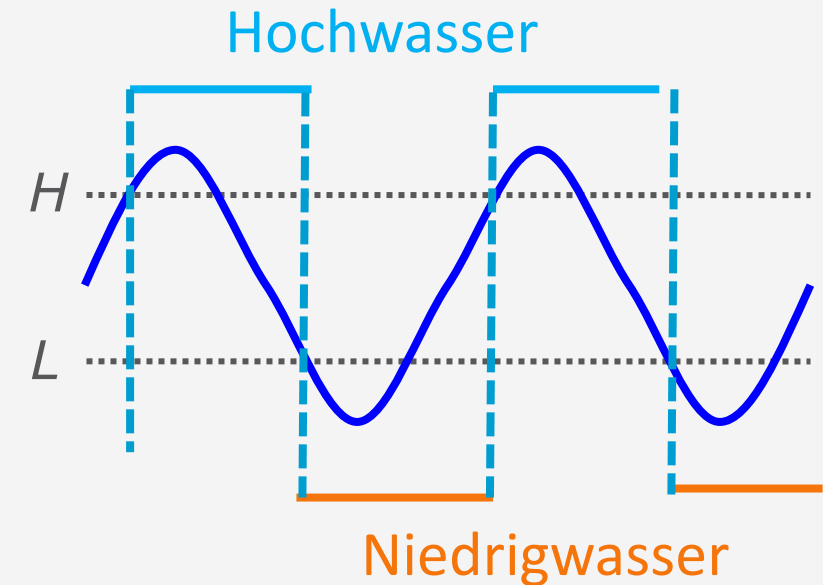
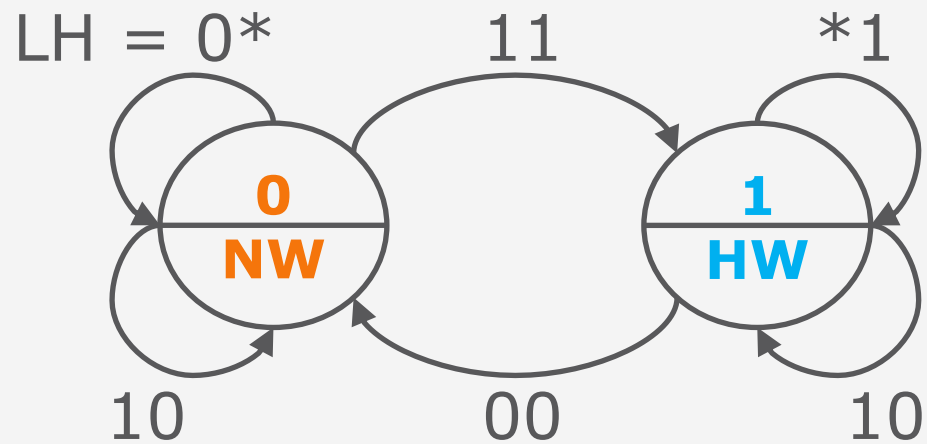
LH=01 eigentlich nicht möglich, aber  
wir entwickeln den Fall mit



# Beispiel: Wahrheitstabelle



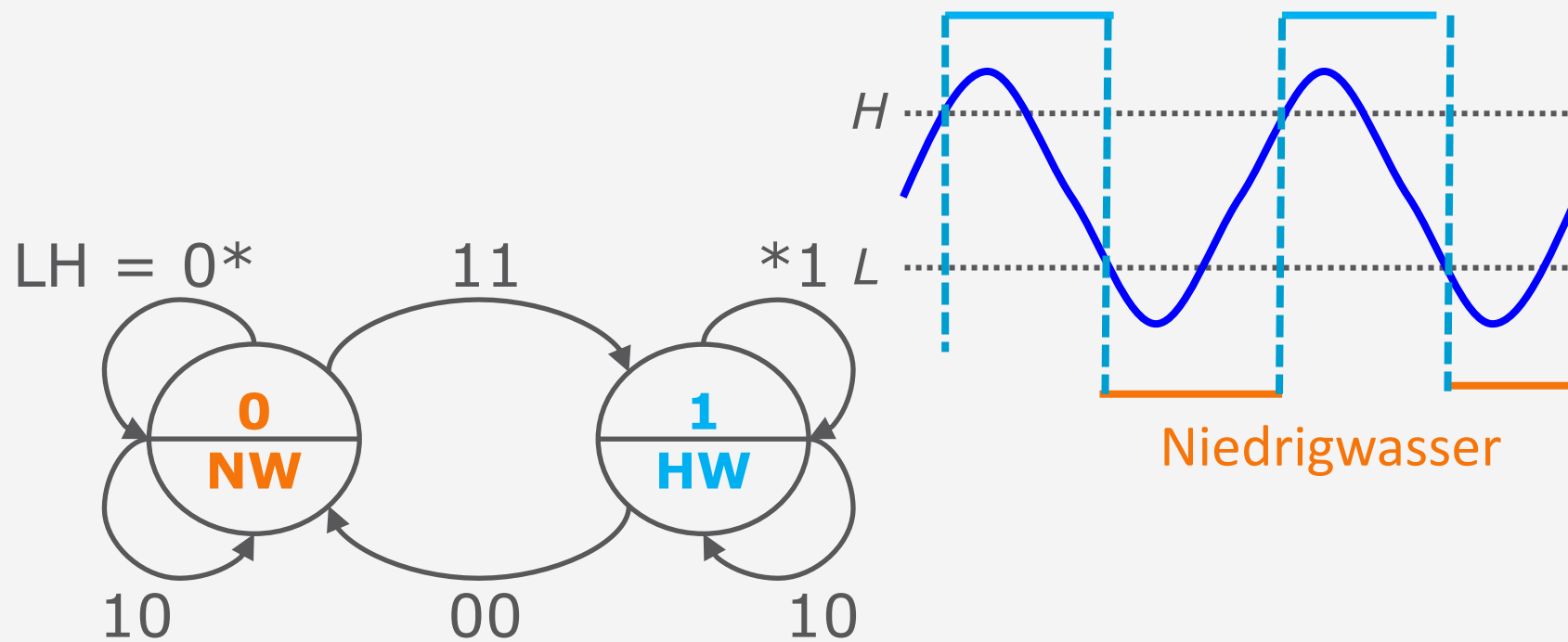
- aktueller Zustand (NW/HW):  $Q$ , Folgezustand:  $Q'$



# Beispiel: Wahrheitstabelle



- aktueller Zustand (NW/HW):  $Q$ , Folgezustand:  $Q'$



$Q$	$L$	$H$	$Q'$
0	0	*	0
0	1	0	0
0	1	1	1
1	*	1	1
1	1	0	1
1	0	0	0

# RS Flipflop: Zustandsübergänge



- Wie müssen R und S gesetzt werden, um die jeweiligen Zustandsübergänge  $Q \rightarrow Q'$  zu realisieren?

- Zur Erinnerung:

R	S	Q	$\bar{Q}$
0	0	$Q_{alt}$	$\bar{Q}_{alt}$
0	1	1	0
1	0	0	1
1	1	<b>x</b>	<b>x</b>

Zustandsübergangstabelle  
RS-Flipflop

$Q \rightarrow Q'$	R	S
0 → 0		
0 → 1		
1 → 0		
1 → 1		

# RS Flipflop: Zustandsübergänge



- Wie müssen R und S gesetzt werden, um die jeweiligen Zustandsübergänge  $Q \rightarrow Q'$  zu realisieren?

- Zur Erinnerung:

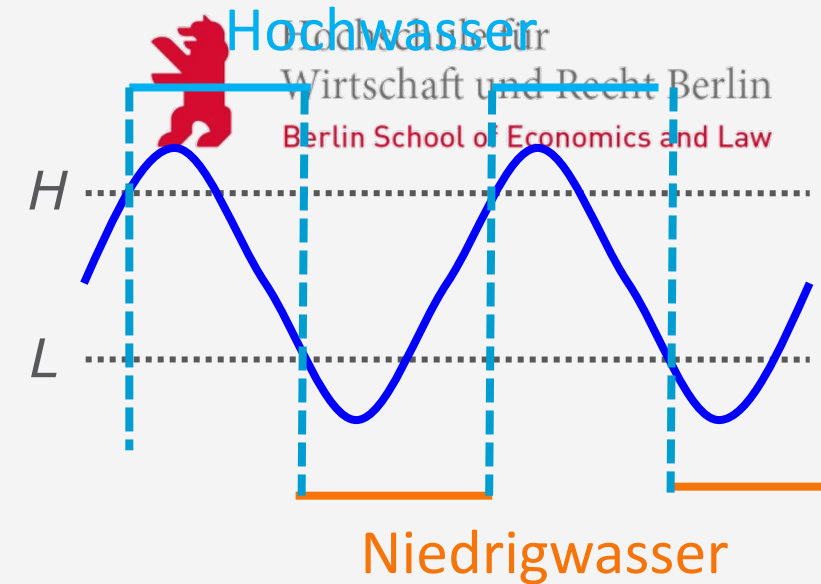
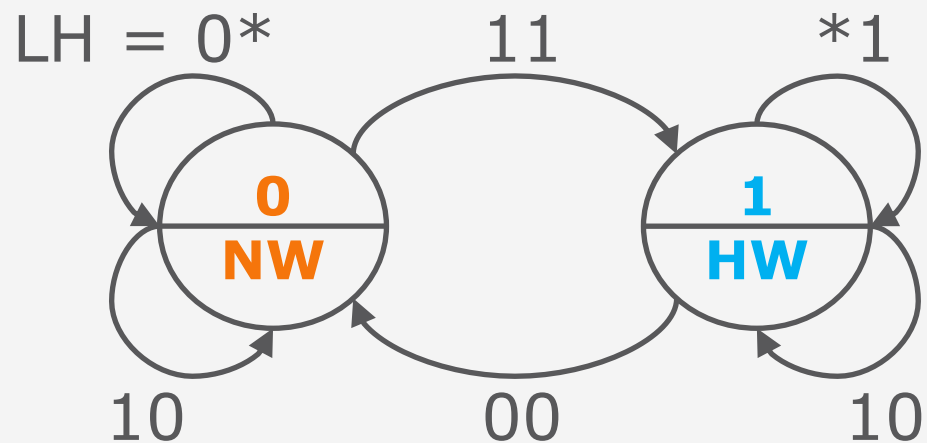
R	S	Q	$\bar{Q}$
0	0	$Q_{alt}$	$\bar{Q}_{alt}$
0	1	1	0
1	0	0	1
1	1	<b>x</b>	<b>x</b>

Zustandsübergangstabelle  
RS-Flipflop

$Q \rightarrow Q'$	R	S
$0 \rightarrow 0$	*	0
$0 \rightarrow 1$	0	1
$1 \rightarrow 0$	1	0
$1 \rightarrow 1$	0	*

# Beispiel: Flipflop-Ansteuerung

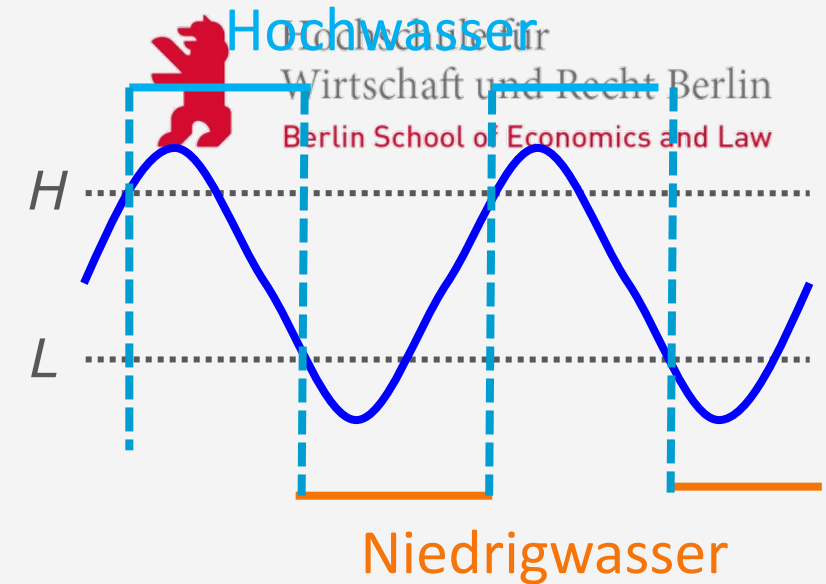
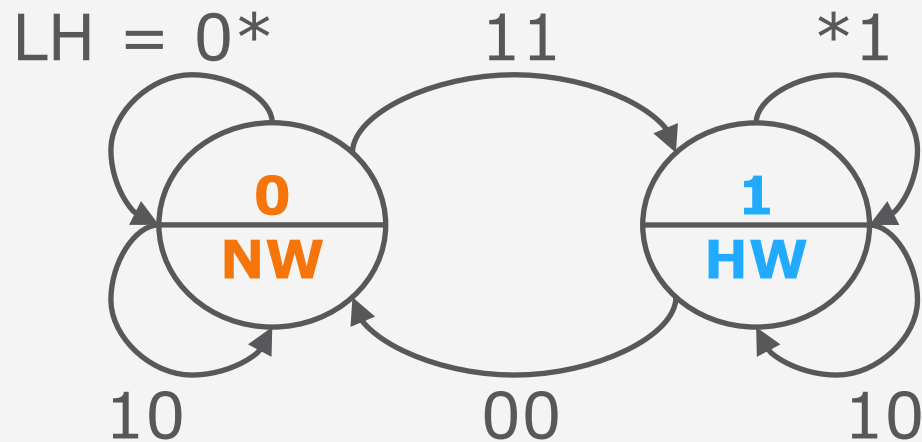
- aktueller Zustand (NW/HW):  $Q$ , Folgezustand:  $Q'$



$Q$	$L$	$H$	$Q'$
0	0	*	0
0	1	0	0
0	1	1	1
1	*	1	1
1	1	0	1
1	0	0	0

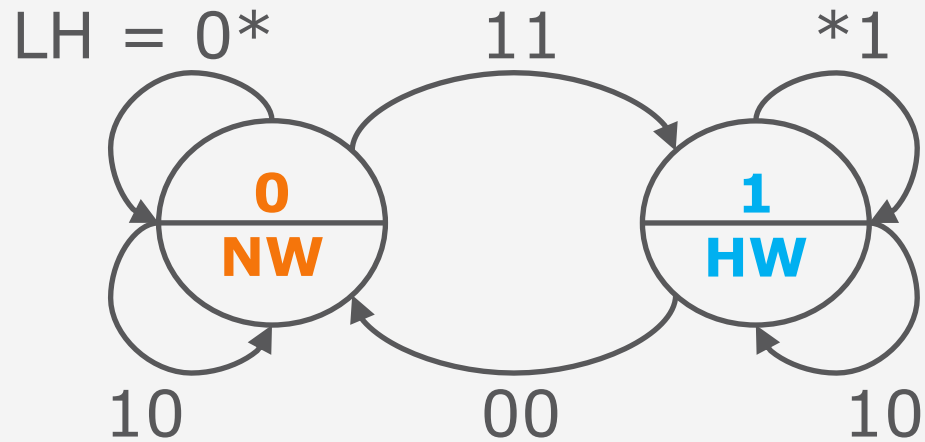
# Beispiel: Flipflop-Ansteuerung

- aktueller Zustand (NW/HW):  $Q$ , Folgezustand:  $Q'$



$Q$	$L$	$H$	$Q'$	$R$	$S$
0	0	*	0	*	0
0	1	0	0	*	0
0	1	1	1	0	1
1	*	1	1	0	*
1	1	0	1	0	*
1	0	0	0	1	0

# Beispiel: KV-Minimierung



$Q$	$L$	$H$	$Q'$	$R$	$S$
0	0	*	0	*	0
0	1	0	0	*	0
0	1	1	1	0	1
1	*	1	1	0	*
1	1	0	1	0	*
1	0	0	0	1	0

**R:**

	$\bar{L} \cdot \bar{H}$	$\bar{L} \cdot H$	$L \cdot H$	$L \cdot \bar{H}$
$\bar{Q}$	*	*	0	*
$Q$	1	0	0	0

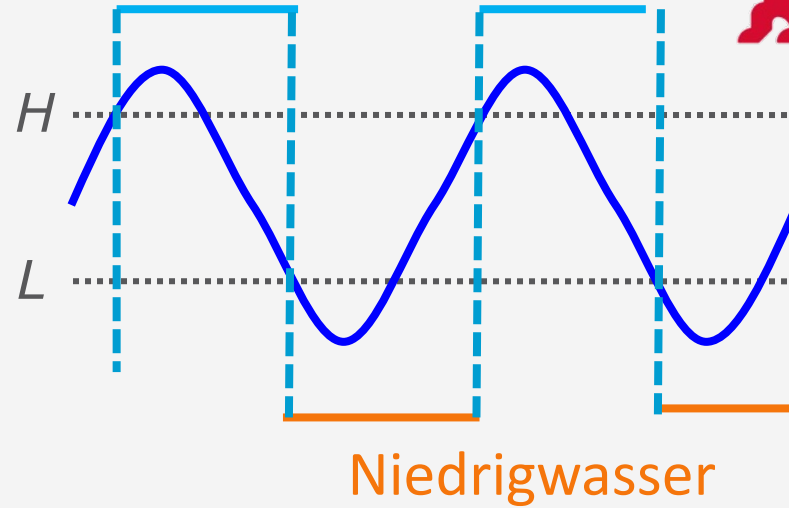
**S:**

	$\bar{L} \cdot \bar{H}$	$\bar{L} \cdot H$	$L \cdot H$	$L \cdot \bar{H}$
$\bar{Q}$	0	0	1	0
$Q$	0	*	*	*

$$R = \bar{L} \cdot \bar{H}$$

$$S = L \cdot H$$

Hochwasser

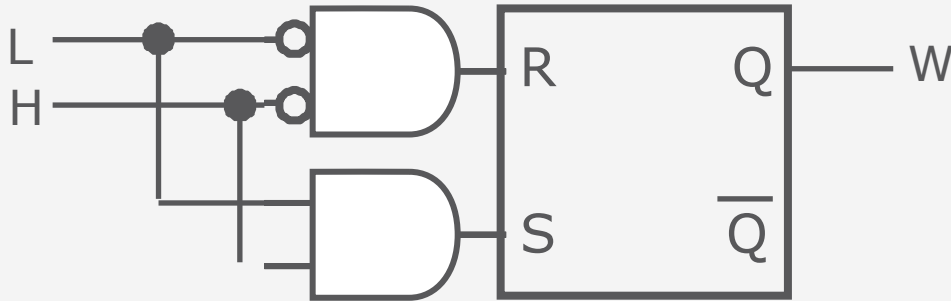




# Beispiel: Schaltung



- Mit  $R = \overline{L} \cdot \overline{H}$  und  $S = L \cdot H$  und Ausgang W als Zustand des RS Flipflops:



## 1. Kombinatorische Logik (Schaltnetze)

- Logikgatter
- Boolesche Algebra
- Logiksynthese (Schaltnetzentwurf)
- Minimierung von Schaltungen

## 2. Sequentielle Logik (Schaltwerke)

- Speicherelemente (Flipflops, Latches)
- Schaltwerksentwurf

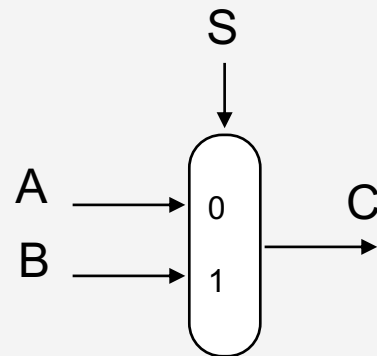
## 3. Wichtige Schaltungen

- Multiplexer und Decoder
- Registersatz
- Arithmetisch-Logische Einheit (ALU)
- Ripple-Carry und Carry-Look-Ahead Addierer

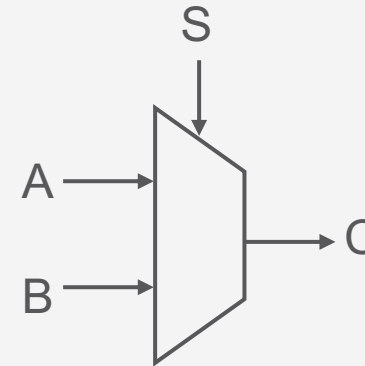
# MUX-Symbole



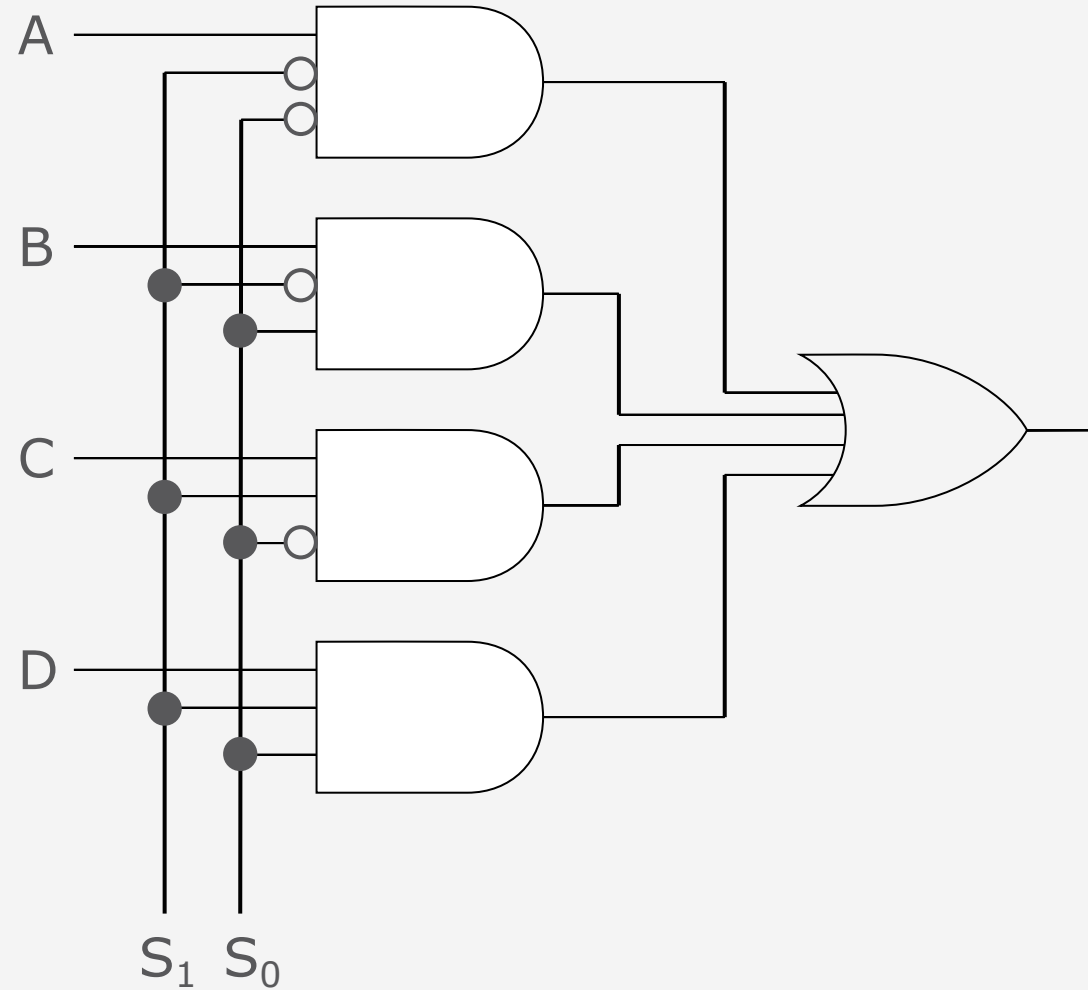
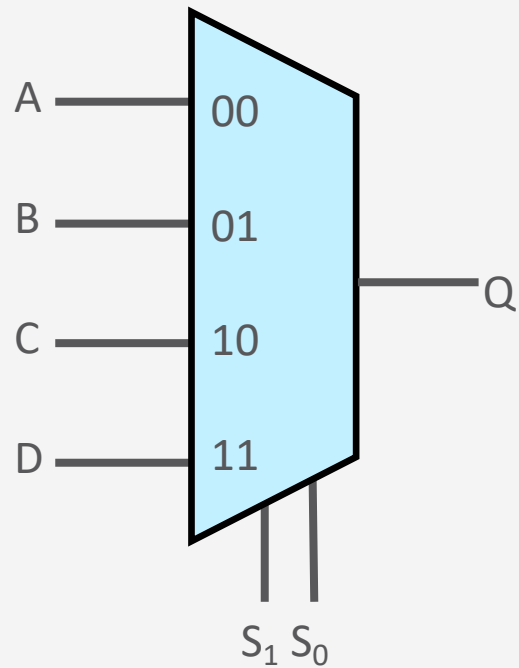
Symbol im Textbuch:



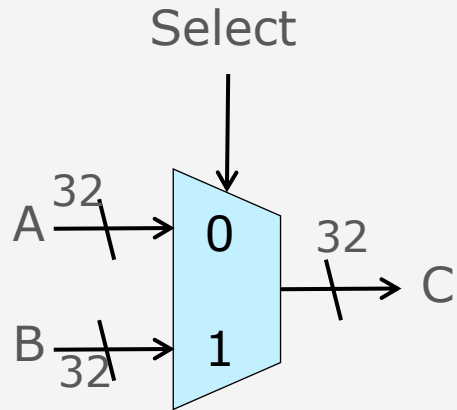
gebräuchlicher:



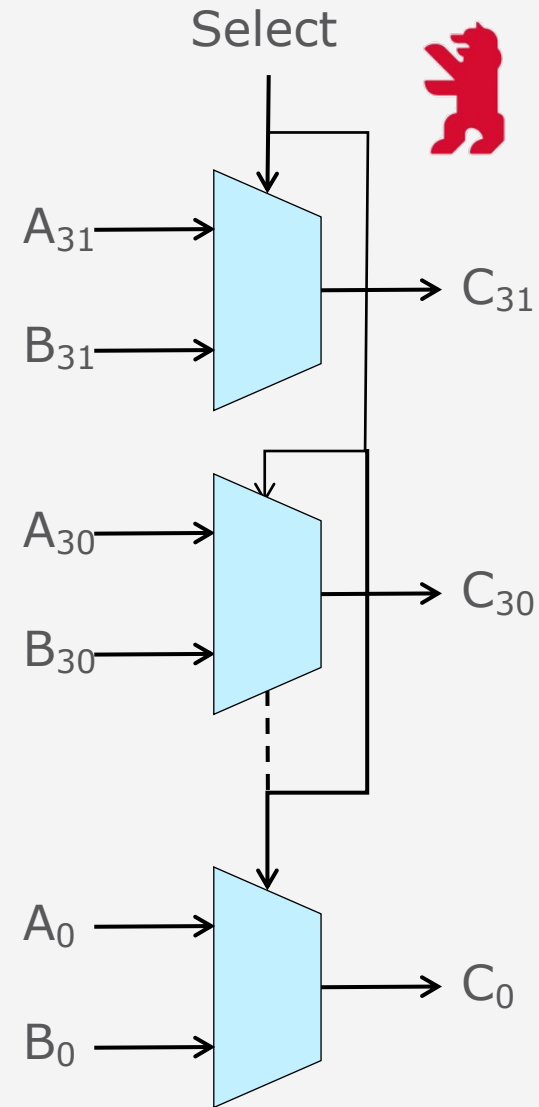
# 4-Input MUX



# 32 Bit Multiplexer



32 Bit breiter MUX



(Array von 32 1 Bit breiten MUX)



# 1-aus-n-Decoder



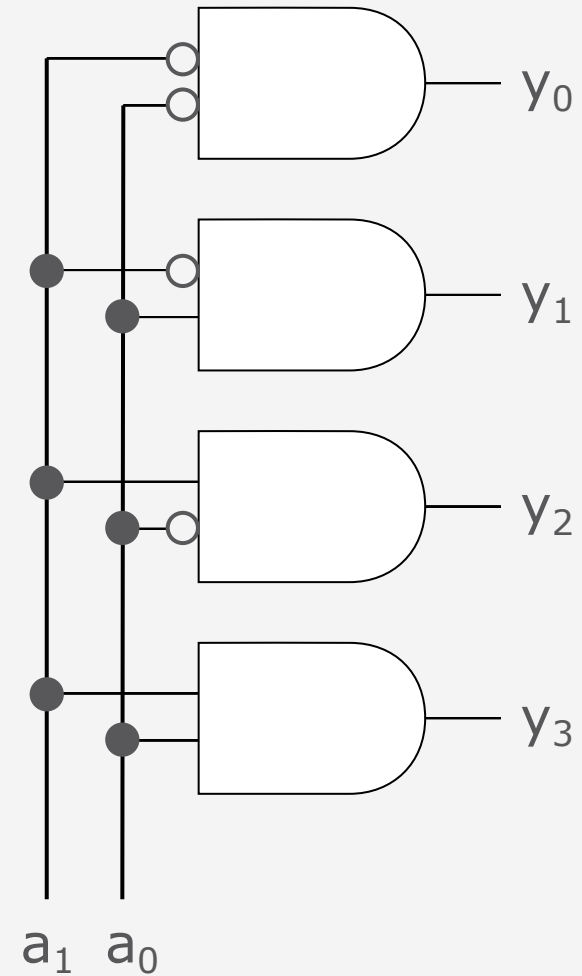
- Schaltung mit  $m$  Eingängen  $a_{m-1} \dots a_0$  und  $n = 2^m$  Ausgängen  $y_{n-1} \dots y_0$
- Ausgang  $y_i$  wird 1 (High) wenn die  $m$  Eingänge der Dualzahl  $i$  entsprechen
- Alle andere Ausgänge werden 0 (Low)

$a_1$	$a_0$	$y_3$	$y_2$	$y_1$	$y_0$
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

# Schaltnetz 1-aus-4-Decoder



$a_1$	$a_0$	$y_3$	$y_2$	$y_1$	$y_0$
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



## 1. Kombinatorische Logik (Schaltnetze)

- Logikgatter
- Boolesche Algebra
- Logiksynthese (Schaltnetzentwurf)
- Minimierung von Schaltungen

## 2. Sequentielle Logik (Schaltwerke)

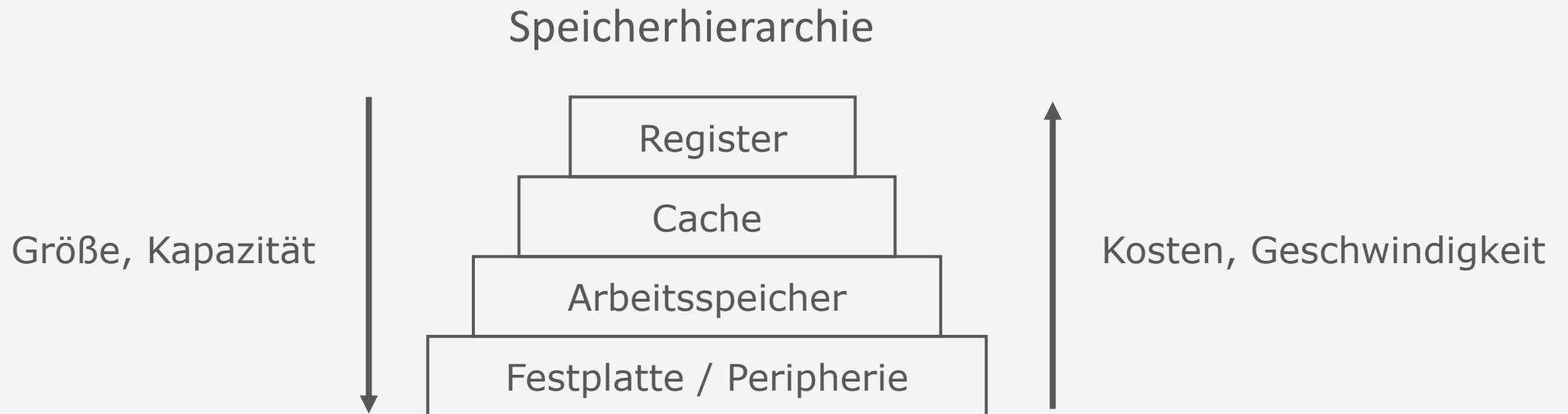
- Speicherelemente (Flipflops, Latches)
- Schaltwerksentwurf

## 3. Wichtige Schaltungen

- Multiplexer und Decoder
- Registersatz
- Arithmetisch-Logische Einheit (ALU)
- Ripple-Carry und Carry-Look-Ahead Addierer



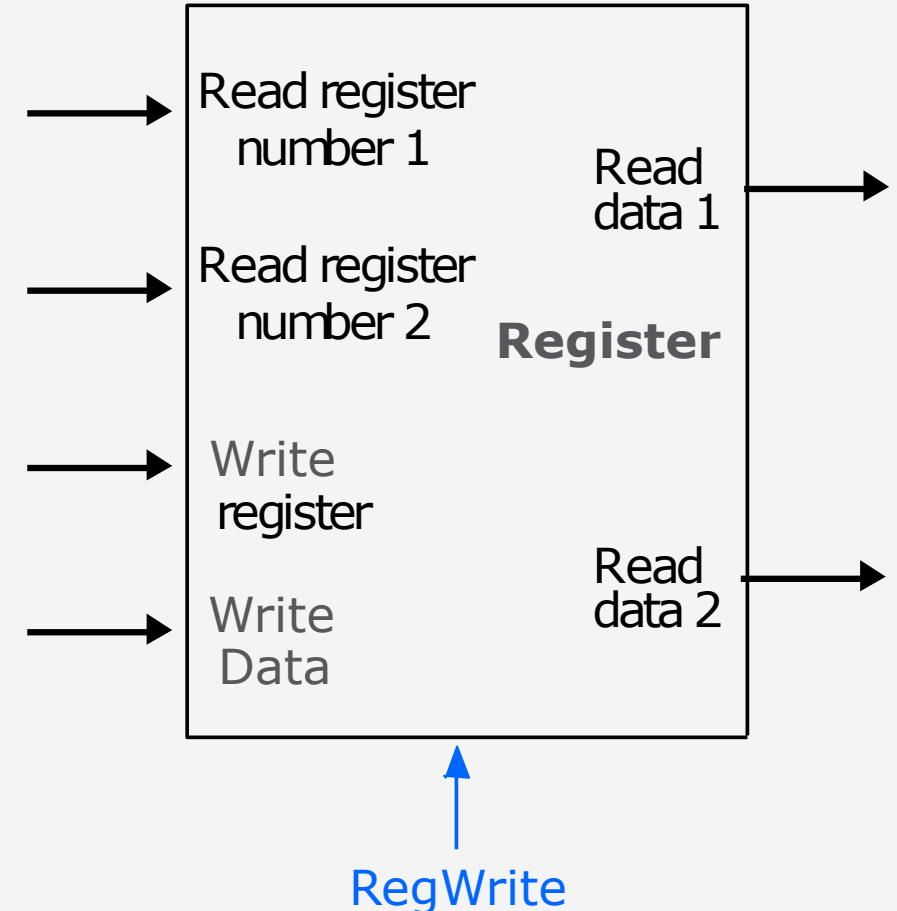
- jeder Prozessor verfügt über einen **Registersatz** / Registerdatei / Registerspeicher
- darin werden **Operanden und Ergebnisse aller Berechnungen** gespeichert
- Register sind i.d.R. genau so groß wie die Wortgröße des Prozessors (8, 16, 32 oder 64 Bit)



# Implementierung eines Registersatzes



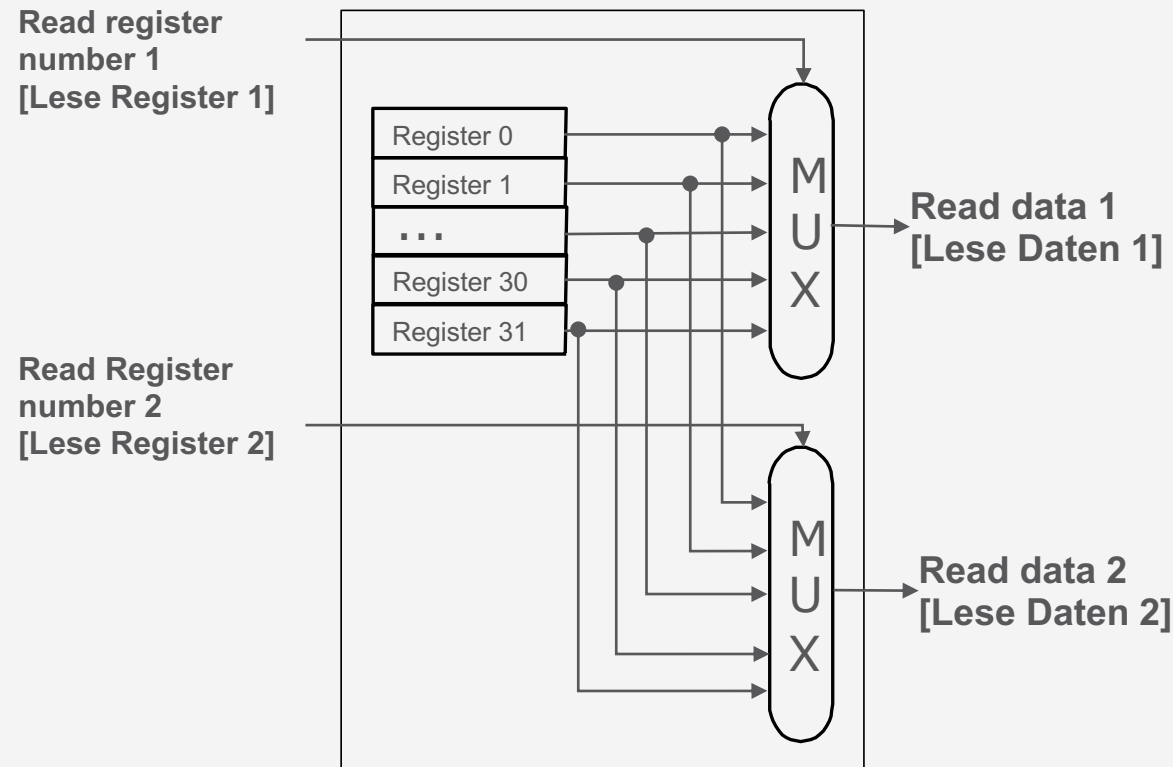
- 32-Bit Register können aus 32 D-Flip-Flops aufgebaut werden
- ein Registersatz (*register file*) besteht aus n Registern
- **Wie können Lese- und Schreibport implementiert werden?**  
Hinweis:  
Wenn eine "7" anliegt soll ein Leseport den Inhalt von Register 7 liefern, wenn eine "7" und eine "42" anliegen, soll der Schreibport 42 in Register 7 schreiben.



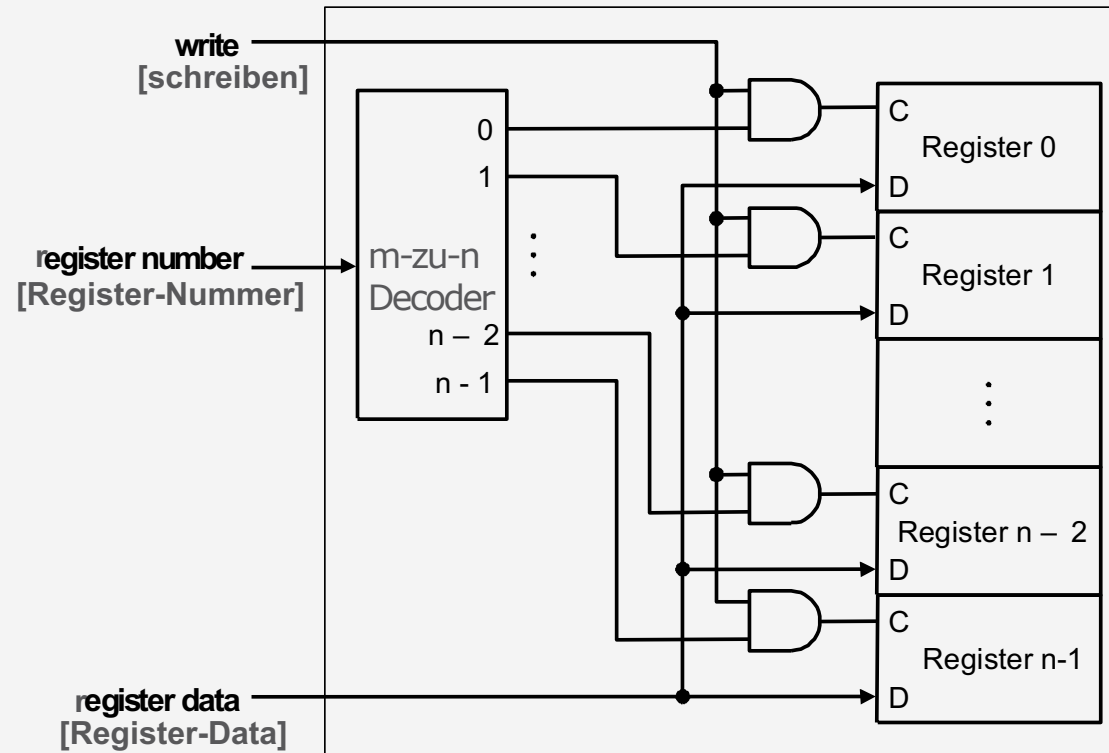
# Implementierung eines Registersatzes



## ■ Implementierung eines Leseports



# Implementierung eines Schreibports



- Takt (clock) hier nicht dargestellt, wird aber trotzdem benutzt, um zu bestimmen **wann** geschrieben wird



## 1. Kombinatorische Logik (Schaltnetze)

- Logikgatter
- Boolesche Algebra
- Logiksynthese (Schaltnetzentwurf)
- Minimierung von Schaltungen

## 2. Sequentielle Logik (Schaltwerke)

- Speicherelemente (Flipflops, Latches)
- Schaltwerksentwurf

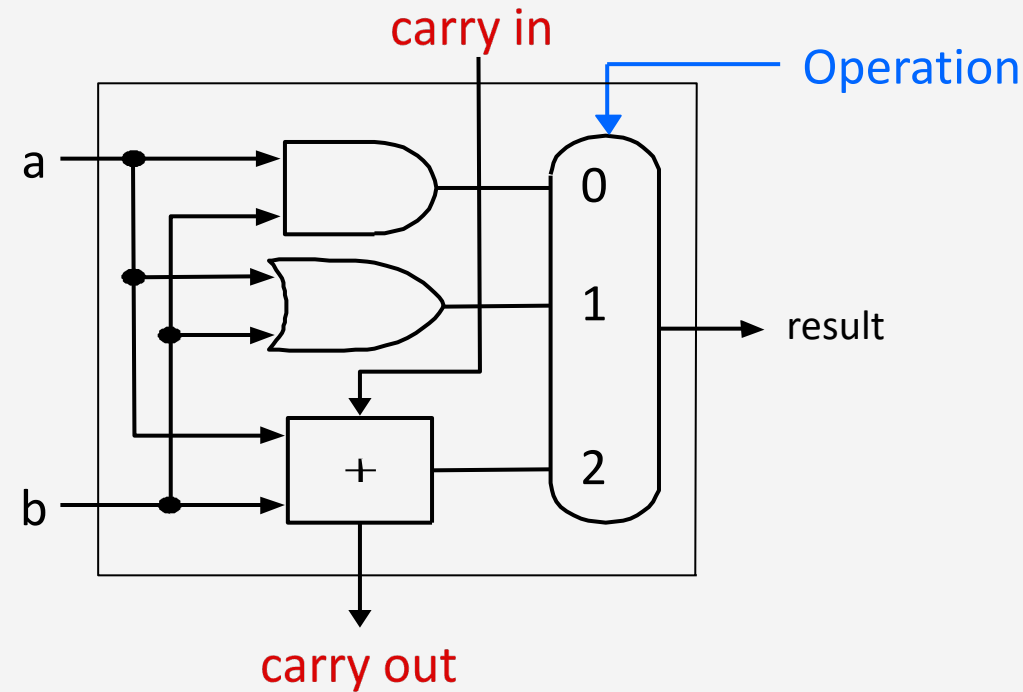
## 3. Wichtige Schaltungen

- Multiplexer und Decoder
- Registersatz
- Arithmetisch-Logische Einheit (ALU)
- Ripple-Carry und Carry-Look-Ahead Addierer

# Arithmetical Logical Unit (ALU)

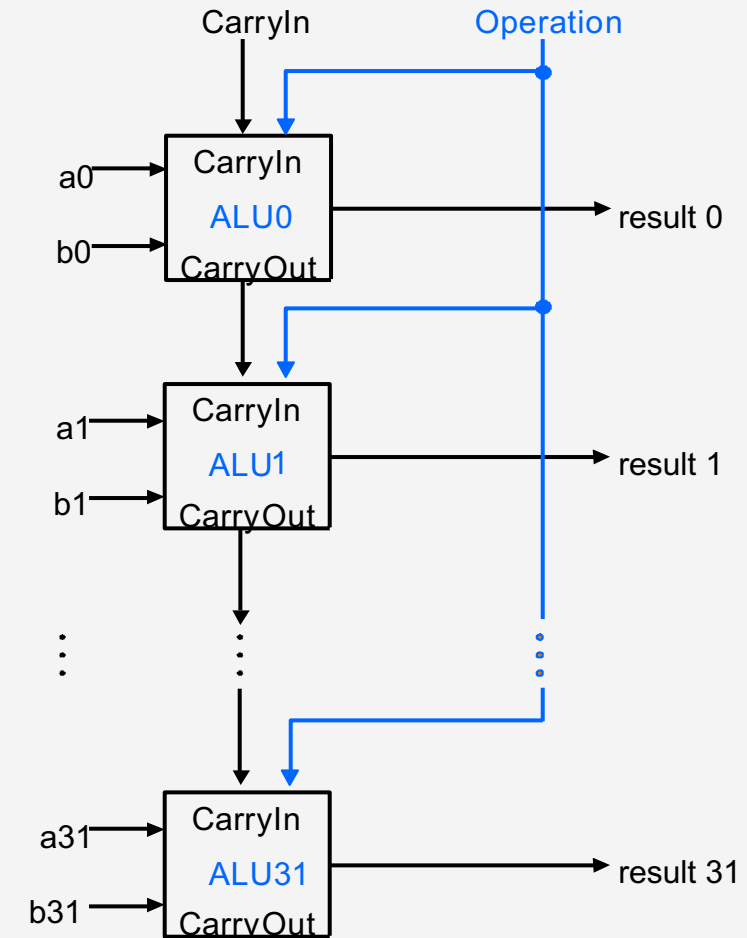


- MUX kann benutzt werden, um eine 1-Bit ALU zu konstruieren, die die Operationen AND, OR und + unterstützt.
- Einfach alles parallel ausführen und den richtigen Output auswählen:



# 32-Bit ALU

- 1-Bit ALU kann zur Realisierung einer 32-Bit ALU genutzt werden.
- Zum Beispiel mit “*ripple-carry adder*”



## 1. Kombinatorische Logik (Schaltnetze)

- Logikgatter
- Boolesche Algebra
- Logiksynthese (Schaltnetzentwurf)
- Minimierung von Schaltungen

## 2. Sequentielle Logik (Schaltwerke)

- Speicherelemente (Flipflops, Latches)
- Schaltwerksentwurf

## 3. Wichtige Schaltungen

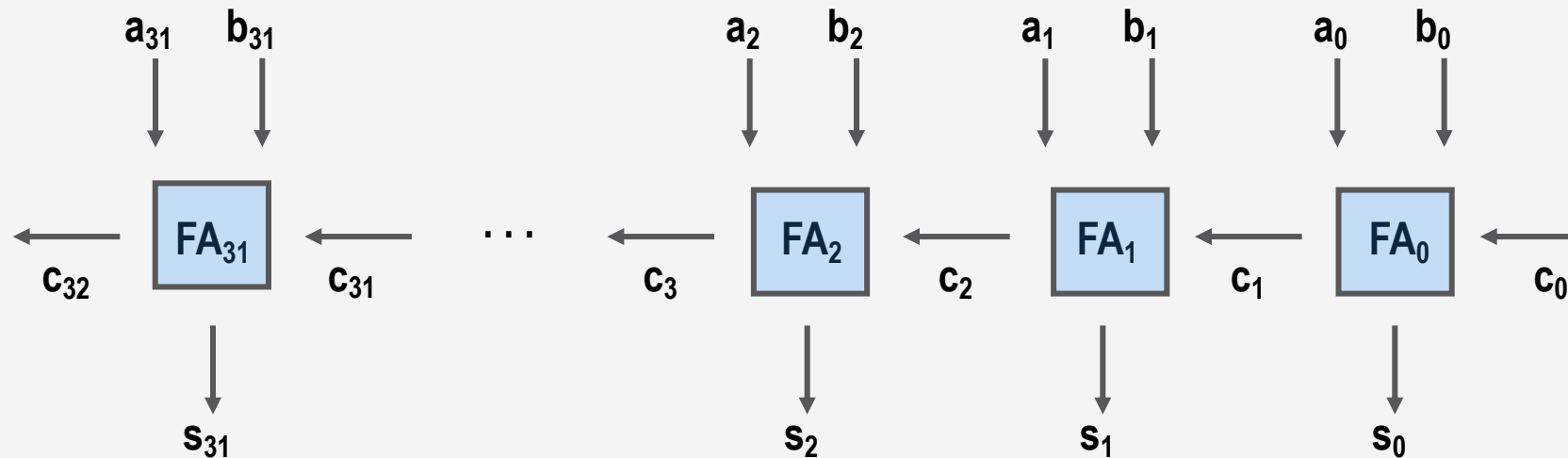
- Multiplexer und Decoder
- Registersatz
- Arithmetisch-Logische Einheit (ALU)
- Ripple-Carry und Carry-Look-Ahead Addierer



# 32-Bit Ripple-Carry Addierer



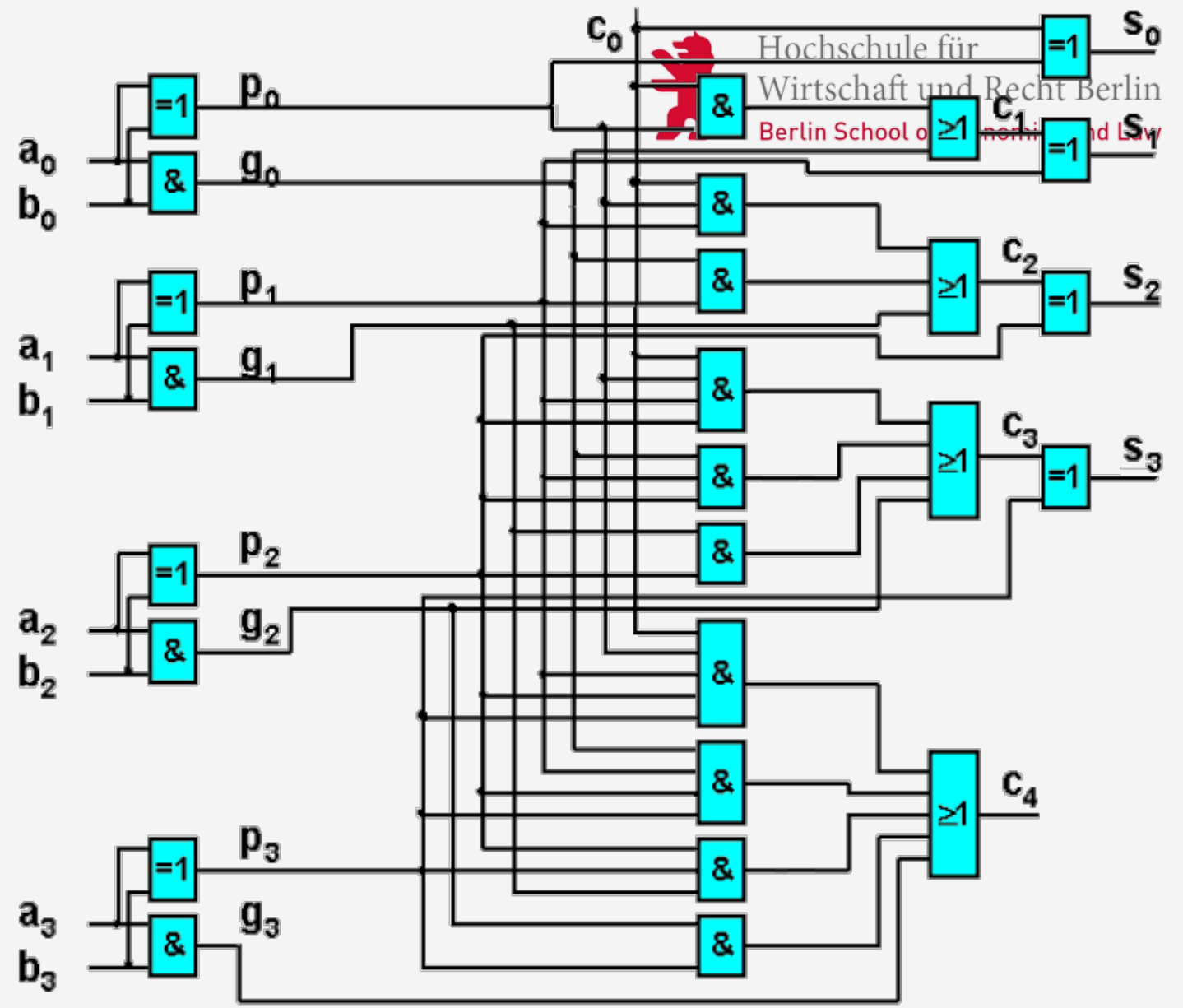
FA = Full Adder



- Problem: Lange Durchlaufzeit für den Übertrag bei großer Stellenzahl, z.B. bei 32-Bit- oder 64-Bit-Addierer.
- Lösung: Parallelisierung der Übertragsbildung  
⇒ Carry-Look-Ahead-Addierer (CLA-Addierer)

# 4-Bit CLA-Addierer

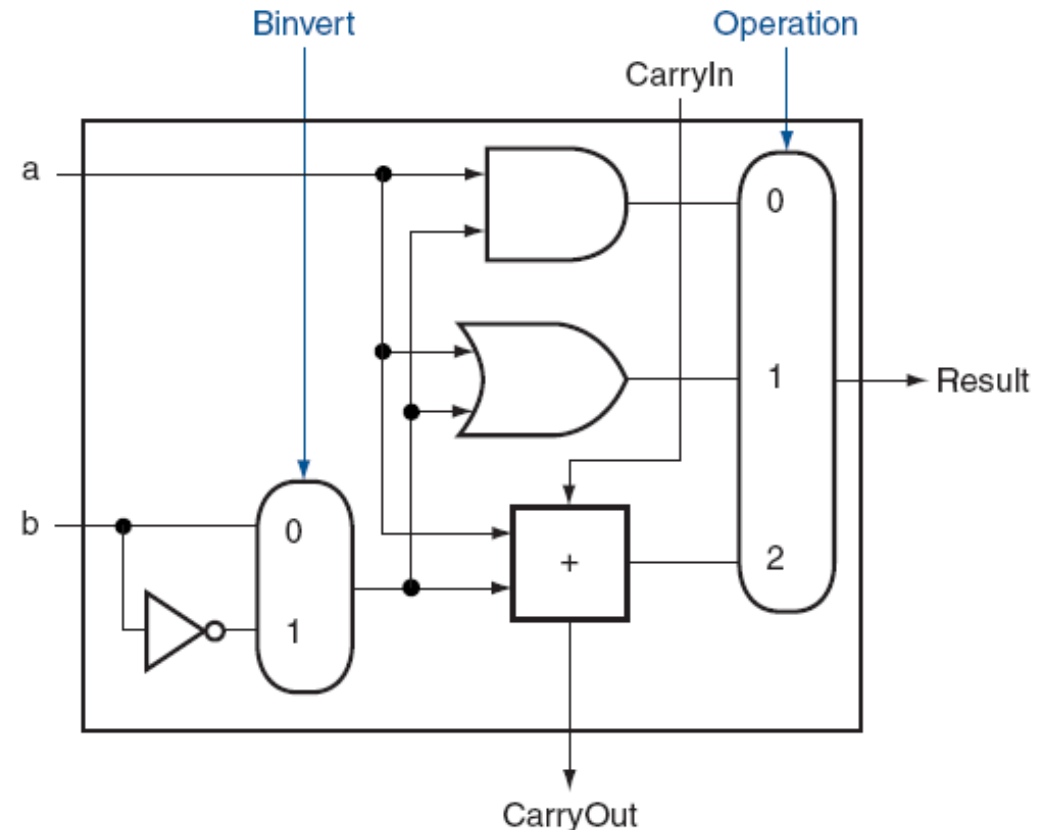
$$s_i = a_i \text{ XOR } b_i \text{ XOR } c_i$$
$$= p_i \text{ XOR } c_i$$



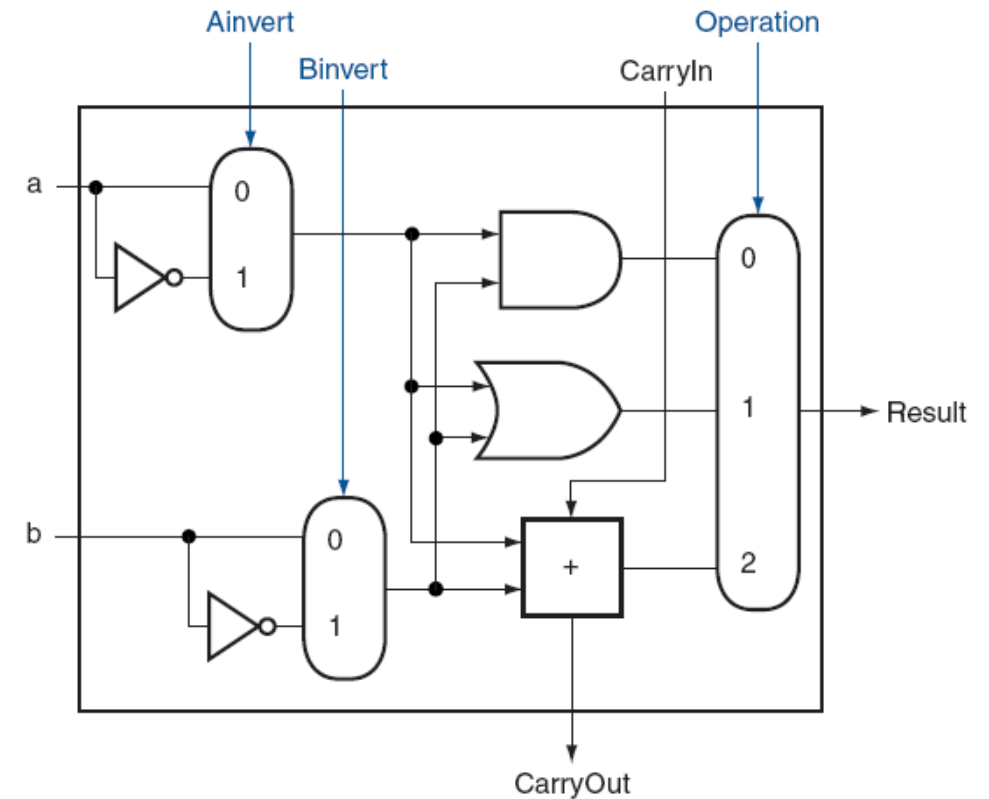
# Subtraktion



- Subtraktion verwendet 2er-Komplement:  
→ einfach b negieren und addieren
- Wie negiert man b?
- → alle Bits invertieren und 1 addieren
- Eine clevere Lösung:
  - Um zu subtrahieren, setze *CarryIn* von ALU0 auf 1, *Binvert* auf 1 und *Operation* auf 10

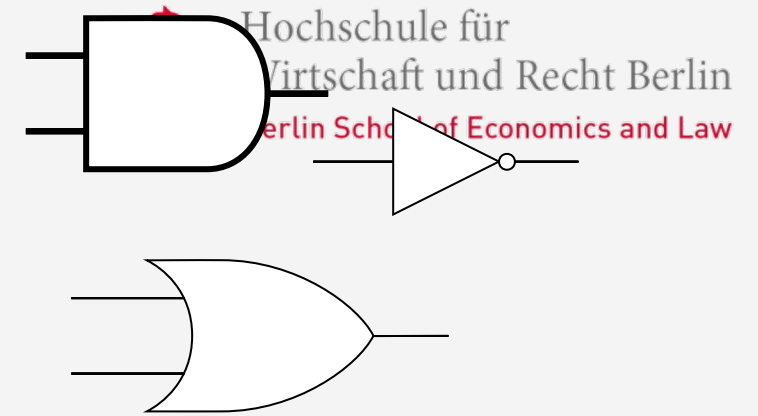


- Wahlweise kann auch a invertiert werden:
- Unsere ALU kann folgende arithmetisch-logische Operationen umsetzen:
  - and, or, nand, nor
  - Addition und Subtraktion



# Zusammenfassung

- Die logische Operatoren UND, ODER und NICHT bilden einen **funktional vollständigen Operatorensatz**



- Boolesche Funktionen können mit **Wahrheitstabelle** spezifiziert werden

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0

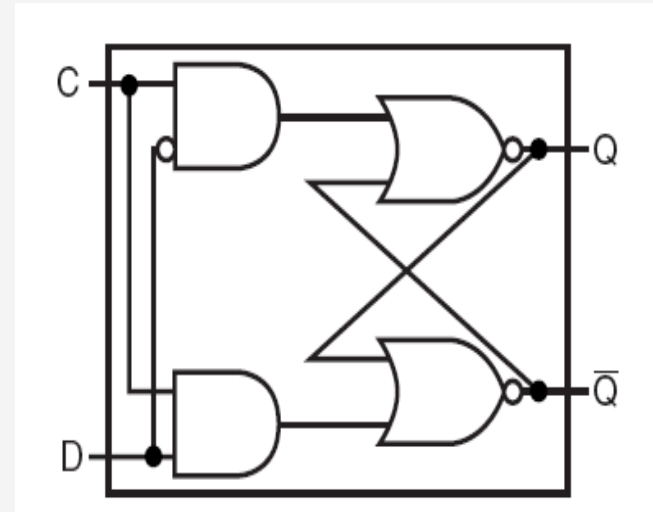
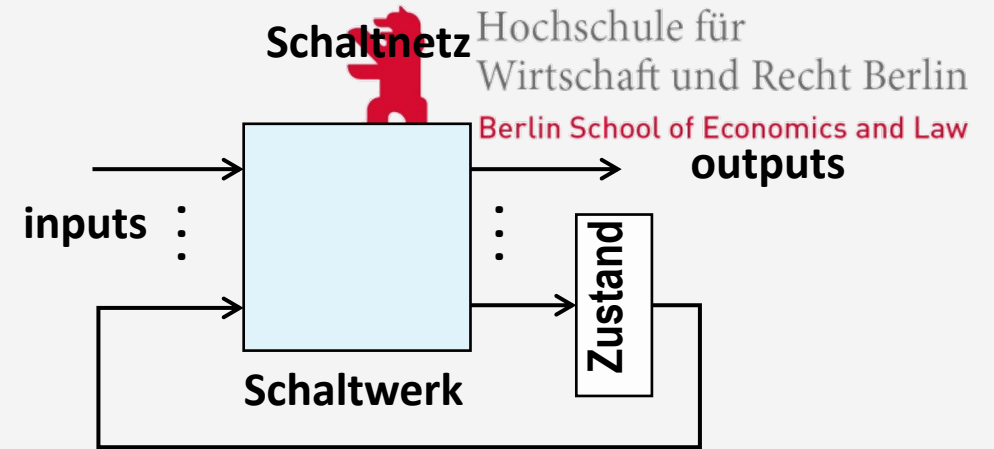
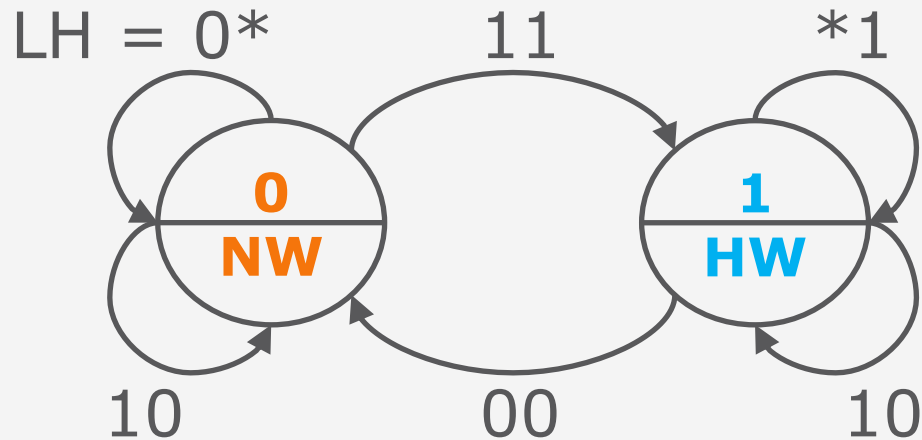
- Ableitung der Schaltfunktion: **KNF** und **DNF**

- Minimierung zum Beispiel mit **KV Diagrammen**

	$\bar{A} \cdot \bar{B}$	$\bar{A} \cdot B$	$A \cdot B$	$A \cdot \bar{B}$
$\bar{C}$	1	0	0	1
$C$	0	0	1	1

# Zusammenfassung

- **Schaltwerke** verfügen über internen Speicher
- Zur Speicherung können **Latches** oder **Flipflops** verwendet werden
- Schaltwerksentwurf aus **endlichen Automaten**



# Zusammenfassung

- Ein **Multiplexer** wählt je nach Steuersignal S einen der Inputs A oder B aus.
- aus Flipflops können wir **Register** bauen
- Wir können eine **ALU** konstruieren, die verschiedene Operationen unterstützt (AND, OR, ADD, SUB, NAND, NOR)
- Ein **Carry-Look-Ahead-Addierer** ist effizienter als ein Ripple-Carry-Addierer

