

# **Digitaltechnik**

Vorlesung  
*Prof. Dr.-Ing. I. Schoppa*

Der vorliegende Text ist ein unvollständiges Begleitmaterial,  
das zum persönlichen Gebrauch der Kursteilnehmer/innen  
in der Vorlesung und in der Übung bestimmt ist.

Eine Weitergabe der Inhalte insbesondere über Foren und Plattformen  
(bspw. Facebook, Web-Sites oder FileSharing) ist nicht gestattet.

Audio/Video-Aufnahmen sind während der Vorlesung  
und der Übung nicht gestattet.

# Organisation

- ◆ Prof. Dr.-Ing. Irenäus Schoppa
- ◆ Fakultät Informatik
- ◆ Gebäude F, Zimmer F124
- ◆ Telefon: 07531 / 206 – 644
- ◆ E-Mail: [irenaeus.schoppa@htwg-konstanz.de](mailto:irenaeus.schoppa@htwg-konstanz.de)
- ◆ Sprechstunden nach Vereinbarung
- ◆ Unterlagen:  
[moodle.htwg-konstanz.de/moodle/course/ view.php?id=4498](https://moodle.htwg-konstanz.de/moodle/course/view.php?id=4498)

# Organisation

Zeit	Mo	Di	Mi	Do	Fr
08:00 09:30		VL DIGI			
09:45 11:15				UE DIGI	
11:30 13:00				UE DIGI	
14:00 15:30		UE DIGI	VL DIGI		
15:45 17:15		UE DIGI			
17:30 19:00					

# Themenübersicht

- ◆ Einführung in Digitaltechnik
  - Zahlensysteme
  - Rechenarithmetik
  - Boolesche Algebra
- ◆ Schaltnetze
  - Minimierungsverfahren:
    - Graphische Minimierung nach Karnaugh-Veitch
    - Algorithmische Minimierung nach Quine-McCluskey
  - Synthese von Schaltnetzen:
    - Multiplexer, Schaltketten, prog. Logikbausteine
  - Logikfamilien und deren Kenndaten
  - dynamisches Verhalten von Schaltnetzen

# Themenübersicht

- ◆ Schaltwerke
  - asynchrone Speicherelemente
  - synchrone Speicherelemente
  - Register, Schieberegister
  - Zähler und Frequenzteiler
  - Zustandsautomaten
  - Registertransferoperationen
  - Realisierung von Steuerwerken
  - Synthese von Schaltwerken

# Literaturverzeichnis

- ◆ Basisliteratur:

1. Schoppa, I.: *Vorlesungs- und Übungsunterlagen*, HTWG Konstanz.
2. Beuth, K: *Elektronik 4. Digitaltechnik*, Vogel Fachbuchverlag, 2006.
3. Pernards, P.: *Digitaltechnik*, Hüthig Verlag, 1992.
4. Pernards, P.: *Digitaltechnik 2, Einführung in die Schaltwerke*, Hüthig Verlag, 1995.

- ◆ Ergänzende Literatur:

1. Liebig, H.: *Logischer Entwurf digitaler Systeme*, Springer Verlag, 4. Auflage, 2005.
2. Borucki, L.: *Digitaltechnik*, Teubner Verlag, 5. Auflage, 2000.
3. Seifart, M. und Beikirch, H.: *Digitale Schaltungen*, Verlag Technik, 5. Auflage, 1998
4. Biere, A.; Kroening, D.; Weissenbacher, G.; Wintersteiger, Ch.: *Digitaltechnik: Eine praxisnahe Einführung*, Springer Verlag, 1. Auflage, 2008.

# Motivation und Ziel

- ◆ Die Digitaltechnik ist ein faszinierendes Gebiet der modernen Elektronik.
- ◆ Durch den Einsatz neuer Technologien hat die Digitaltechnik in den letzten Jahren eine rasante Entwicklung erlebt.
- ◆ Die Komplexität der heutigen und vor allem zukünftigen Aufgaben erfordert interdisziplinäre Lösungen, die eine Kombination aus Software- und Hardware-Komponenten darstellen (sog. Hardware-Software-Codesign).
- ◆ Heutzutage sind Kenntnisse der Digitaltechnik für das Verständnis von vielen technischen Anwendungen unerlässlich (z.B. Mikrocomputertechnik und Telekommunikation).

# Motivation und Ziel

- ◆ Warum die Digitaltechnik im Informatikstudium?
  - Der Aufbau und die Funktionsweise von Computersystemen ist nicht nur eine wichtige Grundlage für die Informations-, Mikrosystem- und Automatisierungstechnik, sondern hat auch starke Auswirkungen auf Systemsoftware.
  - Im Bereich der eingebetteten Systeme beschäftigt man sich oft mit der Grenze zwischen Hardware und Software. Das ist ein wichtiger Arbeitsmarkt, speziell für Europa. Der Entwicklungsbedarf an eingebetteter Software wird in den kommenden Jahren drastisch steigen.
- ◆ Informatiker müssen verstehen, wie informationsverarbeitenden Systeme auf der Hardware-Ebene funktionieren.
- ◆ Informatiker sollen nicht nur in der Software kompetent sein, sondern auch in der Hardware.

# Motivation und Ziel

- ◆ Die Vorlesung *Digitaltechnik*
  - gibt einen breiten Überblick über das Fachgebiet,
  - definiert Grundbegriffe der Digitaltechnik,
  - vermittelt wesentliche theoretische und praktische Grundlagen,
  - umfaßt die Bau- und Funktionsweise digitaler Komponenten,
  - erklärt Methoden und Techniken zur Schaltungsanalyse und Schaltungssynthese,
  - zeigt Möglichkeiten zur systematischen Vorgehensweise bei der Schaltungsminimierung,
  - versucht für das Fachgebiet zu begeistern,
  - ist nicht einfacher aber auch nicht schwieriger als andere Vorlesungen.

# Motivation und Ziel

- ◆ Der Begriff *Digital* ist abgeleitet
  - vom lateinischen Wort *digitus* = Finger, was soviel bedeutet wie „mit Hilfe der Finger rechnen“,
  - aus dem englischen Wort *digit* = Ziffer oder Stelle, was man als „in Ziffernform“ interpretieren kann.
- ◆ Für die Darstellung von digitalen Größen verwendet man abzählbare Elemente.
- ◆ Die Digitaltechnik ist ein Teilgebiet der Elektronik und Informationstechnik, das sich mit der Erfassung, Verarbeitung, Darstellung und Übertragung digitaler Signale befasst.

# Motivation und Ziel

- ◆ Analyse digitaler Systeme

- Analyse bezeichnet die Zerlegung, Zergliederung eines digitalen Systems in seine Bestandteile mit anschließender systematischer Untersuchung des Verhaltens und zwar unter Berücksichtigung von Teilespekten (z.B. Signallaufzeit, Verlustleistung).

- ◆ Synthese (Entwurf) digitaler Systeme

- Synthese bezeichnet die Vereinigung, Zusammenschaltung von zwei oder mehreren digitalen Komponenten mit bekanntem Verhalten zu einer neuen Einheit, so daß daraus das gewünschte Verhalten resultiert, und zwar unter Einhaltung vorgegebener Randbedingungen (wirtschaftliche oder technologische Aspekte):
    - Herstellungskosten,
    - Verlustleistung,
    - Signallaufzeit (Timing Constraints),
    - Chip-Fläche (Area Constraints)).

- ◆ Top-Down-Entwurf (von oben nach unten)
  - iterative oder rekursive Zerlegung eines Ausgangsproblems in einfachere Teilprobleme, um letztendlich die Gesamtlösung aus einfachen Grundfunktionen zusammensetzen zu können.
- ◆ Bottom-Up-Entwurf (von unten nach oben)
  - sukzessive (schrittweise) Zusammensetzung von Teilsystemen aus elementaren Bausteinen zu komplexeren Komponenten, bis das gewünschte Systemverhalten erreicht ist.
- ◆ In der Praxis werden beide Entwurfsprinzipien zumeist kombiniert eingesetzt.
  - Man versucht beim Top-Down-Entwurf schon möglichst bald Teilprobleme für eine endgütige Realisierung zu erkennen und diese dann im Bottom-Up-Entwurf zu lösen.

# Anwendungen der Digitaltechnik

- ◆ Consumer-Bereich
  - Fahrkartenautomat, Waschmaschine, „intelligenter“ Kühlschrank, Fotokamera, CD-/DVD-/MP3-Player, Smart-Phone, ...
- ◆ Automobile-Bereich
  - ABS, ESP, Motorsteuerung, Automatikgetriebe, „intelligenter“ Allradantrieb (xDrive), Navigationssystem, ...
- ◆ Office-Bereich
  - FAX, Telefonanlagen, Drucker, Kopierer, Kassensysteme, Zeiterfassung, ...
- ◆ Medizin-Bereich
  - Herzschrittmacher, Infusionspumpe, Computertomographie, ...
- ◆ Wissenschaft/Technik
  - Messgeräte, Signalgeneratoren, Oszilloskope, ...

# Anwendungen der Digitaltechnik

- ◆ Luft-/Raumfahrt
  - Satelliten, Weltraumsonden, Autopilot, ...
- ◆ Automatisierungstechnik
  - Prozeßrechner, speicherprogrammierbare Steuerungen, CNC-Technik, Industrieroboter, ...
- ◆ Nachrichten-/Kommunikationstechnik
  - Kommunikationsnetze (ISDN, DSL, Internet), Mobilfunk (GPS, UMTS, LTE), Sprach- und Bildübertragung (DVB-T/-S/-C, IPTV), ...
- ◆ Computertechnik
  - Universalrechner, Minicomputer (Workstation), Personal Computer (Mikrocomputer), Notebook, ...
- ◆ Militärbereich
  - Sonar-/Radarsysteme, Feuerleitsysteme, unbemannte Flugkörper, ..

- ◆ Eine Zahl wird durch Symbole (i.d.R. Ziffern) dargestellt, die nach bestimmten Regeln aneinander gereiht werden.
- ◆ Ein *Zahlensystem* wird durch die Gesamtheit der Symbole und zugehörigen Regeln gebildet.
- ◆ In der Praxis und technischen Anwendungen sind *Stellenwert-* oder *Positionssysteme* relevant.
- ◆ Im Rahmen von *Digitaltechnik* lernen wir
  - die wichtigsten Zahlensysteme,
  - Grundrechenarten im Dualsystem und
  - Zahlenformate

- ◆ Stellenwert-/Positionssystem
  - heute allgemein verwendetes Zahlensystem
  - Dezimalsystem: zehn verschiedene Symbole (arabische Ziffern) {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Notationregel: Jeder Stelle (Position) innerhalb einer Zahl ist ein bestimmter Wert zugeordnet. Man bezeichnet ihn als Stellenwert.
- ◆ eine n-stellige nicht negative Zahl zur Basis b wird notiert:

$$(X)_b = (x_{n-1} x_{n-2} \dots x_i \dots x_1 x_0)_b$$

$$(X)_b = x_{n-1} \cdot b^{n-1} + x_{n-2} \cdot b^{n-2} + \dots + x_i \cdot b^i + \dots + x_1 \cdot b^1 + x_0 \cdot b^0$$

- b Basis des Stellenwertsystems,  $b \in \{2, 3, \dots\}$
- n Anzahl der Ziffernstellen
- $x_i$  Ziffer an der i-ten Position,  $x_i \in \{0, 1, 2, \dots, b-2, b-1\}$
- $x_{n-1}$  Ziffer an der höchstwertigen Stelle M S B
- $x_0$  Ziffer an der niedrigstwertigen Stelle L S B

- ◆ gebräuchliche Zahlensysteme in der Digitaltechnik
  - Duales Zahlensystem
    - Basis  $b=2$ , Ziffernmenge = {0, 1}
  - Oktales Zahlensystem
    - Basis  $b=8$ , Ziffernmenge = {0, 1, 2, 3, 4, 5, 6, 7}
  - Dezimales Zahlensystem
    - Basis  $b=10$ , Ziffernmenge = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Hexadezimales Zahlensystem
    - Basis  $b=16$ , Ziffernmenge = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
      - mit A := 10, B := 11, C := 12, D := 13, E := 14 und F := 15
- ◆ Beispiele

$$(11101)_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$b:2$  n:5

$$(527)_8 = 5 \cdot 8^2 + 2 \cdot 8^1 + 7 \cdot 8^0$$

$b:8$  n:3

$$(FE01)_{16} = F \cdot 16^3 + E \cdot 16^2 + 0 \cdot 16^1 + 1 \cdot 16^0$$

$b:16$  n:4

- ◆ Umwandlung einer nicht negativen Dualzahl in eine Hexadezimalzahl ( $b=16$ )
  - Besteht eine Dualzahl aus einer nicht durch 4 ohne Rest teilbaren Anzahl von Stellen, so wird sie linksseitig mit Nullen aufgefüllt.
  - Eine Dualzahl wird in 4er-Blöcke (sog. Tetrade) aufgeteilt, beginnend mit der niedrigstwertigen Stelle.
  - Jede Tetrade wird separat in ihre hexadezimale Darstellung laut der folgenden Tabelle transformiert:

Tetrade	0000	0001	0010	0011	0100	0101	0110	0111
HEX	0	1	2	3	4	5	6	7

Tetrade	1000	1001	1010	1011	1100	1101	1110	1111
HEX	8	9	A	B	C	D	E	F

- Die Hexadezimaldarstellung der Ausgangszahl ergibt sich aus der Konkatenation (Verkettung) dieser Transformationen unter Berücksichtigung der ursprünglichen Reihenfolge.

- ♦ Umwandlung einer nicht negativen Dualzahl in eine Oktalzahl ( $b=8$ )
  - Besteht eine Dualzahl aus einer nicht durch 3 ohne Rest teilbaren Anzahl von Stellen, so wird sie linksseitig mit Nullen aufgefüllt.
  - Eine Dualzahl wird in 3er-Blöcke aufgeteilt, beginnend mit der niedrigstwertigen Stelle.
  - Jeder 3er Block wird separat in seine oktale Darstellung laut der folgenden Tabelle transformiert:

3er Block	000	001	010	011	100	101	110	111
OCT	0	1	2	3	4	5	6	7

- Die Oktaldarstellung der Ausgangszahl ergibt sich aus der Konkatenation (Verkettung) dieser Transformationen unter Berücksichtigung der ursprünglichen Reihenfolge.

# Zahlensysteme

Beispiel:  $(X)_2 = (1101101001)_2$   $(X)_{16} = ?$

$$(X)_2 = \underbrace{0\ 0\ 1\ 1}_{(3)} \underbrace{0\ 1\ 1\ 0}_{(6)} \underbrace{1\ 0\ 0\ 1}_{(9)} \\ (X)_{16} = (3\ 6\ 9)_{16}$$

Beispiel:  $(X)_2 = (1101111011)_2$   $(X)_8 = ?$

$$(X)_2 = \underbrace{0\ 0\ 1}_{(1)} \underbrace{1\ 0\ 1}_{(5)} \underbrace{1\ 1\ 1\ 0\ 1}_{(7\ 3)} \\ (X)_8 = (1\ 5\ 7\ 3)_8$$

Beispiel:  $(X)_{16} = (1F5)_{16}$   $(X)_2 = ?$

$$(X)_2 = ( \underbrace{0\ 0\ 0\ 1}_{(1)} \quad 1\ 1\ 1\ 1 \quad 0\ 1\ 0\ 1 )_2 \\ (X)_2 = ( 1\ 1\ 1\ 1 \ 0\ 1\ 0\ 1 )_2$$

# Zahlensysteme

- Umwandlung einer n-stelligen nicht negativen Zahl X zur Basis b in eine Dezimalzahl Y mit der Direktmethode

$$(Y)_{10} = \sum (x_i)_b \cdot b^i \quad \text{für } 0 \leq i \leq n-1$$

Beispiel:  $(X)_2 = (100101)_2$ ,  $(Y)_{10} = ?$  n = 6

$$\begin{aligned} (Y)_{10} &= 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ (Y)_{10} &= 1 \cdot 2^5 + 1 \cdot 2^2 + 1 \cdot 2^0 \\ (Y)_{10} &= 32 + 4 + 1 = (37)_{10} \end{aligned}$$

Beispiel:  $(X)_5 = (3412)_5$ ,  $(Y)_{10} = ?$  n = 4

$$\begin{aligned} (Y)_{10} &= 3 \cdot 5^3 + 4 \cdot 5^2 + 1 \cdot 5^1 + 2 \cdot 5^0 \\ (Y)_{10} &= 3 \cdot 125 + 4 \cdot 25 + 1 \cdot 5 + 2 \cdot 1 \\ (Y)_{10} &= 375 + 100 + 5 \cdot 2 = (482)_{10} \end{aligned}$$

- Umwandlung einer n-stelligen nicht negativen Zahl X zur Basis b in eine Dezimalzahl Y nach dem Horner-Schema

$$(Y)_{10} = \sum (x_i)_b \cdot b^i \quad \text{für } 0 \leq i \leq n-1$$

$$\begin{aligned} \sum (x_i)_b \cdot b^i &= x_{n-1} \cdot b^{n-1} + x_{n-2} \cdot b^{n-2} + \dots + x_1 \cdot b^1 + x_0 \cdot b^0 \\ &= b \cdot (x_{n-1} \cdot b^{n-2} + x_{n-2} \cdot b^{n-3} + \dots + x_1) + x_0 \\ &= b \cdot (b \cdot (x_{n-1} \cdot b^{n-3} + x_{n-2} \cdot b^{n-4} + \dots) + x_1) + x_0 \\ &= \dots \\ &= b \cdot (b \cdot \dots b \cdot (x_{n-1} \cdot b + x_{n-2}) + \dots + x_1) + x_0 \end{aligned}$$

ergibt

$$(Y)_{10} := 0$$

$$(Y)_{10} := (Y)_{10} \cdot b + (x_i)_b \quad \text{für } i = \{n-1, n-2, \dots, 1, 0\}$$

# Zahlensysteme

Beispiel  $(X)_2 = (101011)_2$ ,  $(Y)_{10} = ?$ ,  $b := 2$ ,  $n := 6$

$$(Y)_{10} := 0$$

$$(Y)_{10} := (Y)_{10} \cdot 2 + (x_i)_2 \quad \text{für } i = \{5, 4, 3, 2, 1, 0\}$$

$$(Y)_{10} := (Y)_{10} \cdot 2 + 1 = 0 \cdot 2 + 1 = 1$$

$$(Y)_{10} := (Y)_{10} \cdot 2 + 0 = 1 \cdot 2 + 0 = 2 + 0 = 2$$

$$(Y)_{10} := (Y)_{10} \cdot 2 + 1 = 2 \cdot 2 + 1 = 4 + 1 = 5$$

$$(Y)_{10} := (Y)_{10} \cdot 2 + 0 = 5 \cdot 2 + 0 = 10 + 0 = 10$$

$$(Y)_{10} := (Y)_{10} \cdot 2 + 1 = 10 \cdot 2 + 1 = 20 + 1 = 21$$

$$(Y)_{10} := (Y)_{10} \cdot 2 + 1 = 21 \cdot 2 + 1 = 42 + 1 = 43$$

$$(Y)_{10} = 43$$

# Zahlensysteme

Beispiel  $(X)_8 = (1705)_8$ ,  $(Y)_{10} = ?$

$$(Y)_{10} := 0$$

$$(Y)_{10} := (Y)_{10} \cdot 8 + (x_i)_8 \quad \text{für } i = \{3, 2, 1, 0\}$$

$$(Y)_{10} := (Y)_{10} \cdot 8 + 1 = 0 \cdot 8 + 1 = 1$$

$$(Y)_{10} := (Y)_{10} \cdot 8 + 7 = 1 \cdot 8 + 7 = 15$$

$$(Y)_{10} := (Y)_{10} \cdot 8 + 0 = 15 \cdot 8 + 0 = 120$$

$$(Y)_{10} := (Y)_{10} \cdot 8 + 5 = 120 \cdot 8 + 5 = 965$$

$$(Y)_{10} = (965)_{10}$$

# Zahlensysteme

- ◆ Umwandlung einer nicht negativen Dezimalzahl X in eine n-stellige Zahl Y zur Basis b: *Division mit Rest -Methode*
  - gegeben  $(X)_{10} \geq 0$ , gesucht  $(Y)_b = (y_{n-1}y_{n-2}\dots y_1y_0)_b$  mit  $b \geq 2$
  - Lösung: iteratives Verfahren, in dem die Dezimalzahl X durch die Basis b dividiert wird, und der Rest aus der Division als die Ziffer an der i-ten Stelle notiert wird.  
solange  $(X)_{10} > 0$  berechne für  $i = \{0, 1, \dots n-1\}$  :
    1. Schritt:  $(X)_{10} / b = (Z)_{10} + (r)_{10} \Rightarrow (y_i)_b := (r)_{10}$
    2. Schritt:  $(X)_{10} := (Z)_{10}$
 mit
  - $n := \lceil \log_b(X)_{10} \rceil$
  - $/$  Divisionsoperator für ganzzahlige Division
  - $(Z)_{10}$  Ganzzahlquotient
  - $(r)_{10}$  Rest aus der ganzzahligen Division

# Zahlensysteme

Beispiel:  $(X)_{10} = (366)_{10}$ ,  $(Y)_2 = ?$

$$(X)_{10}/b = (Z)_{10} + \downarrow \text{Rest aus der Division}$$

$$366 / 2 = 183 + 0 \Rightarrow (y_0)_2 = 0$$

$$183 / 2 = 91 + 1 \Rightarrow (y_1)_2 = 1$$

$$91 / 2 = 45 + 1 \Rightarrow (y_2)_2 = 1$$

$$45 / 2 = 22 + 1 \Rightarrow (y_3)_2 = 1$$

$$22 / 2 = 11 + 0 \Rightarrow (y_4)_2 = 0$$

$$11 / 2 = 5 + 1 \Rightarrow (y_5)_2 = 1$$

$$5 / 2 = 2 + 1 \Rightarrow (y_6)_2 = 1$$

$$2 / 2 = 1 + 0 \Rightarrow (y_7)_2 = 0$$

$$1 / 2 = 0 + 1 \Rightarrow (y_8)_2 = 1$$

↑ Werte von unten  
nach oben auslesen

$$(Y)_2 = (y_8 \ y_7 \ y_6 \ y_5 \ y_4 \ y_3 \ y_2 \ y_1 \ y_0)_2 = (101101110)_2$$

# Zahlensysteme

Beispiel:  $(X)_{10} = (5949)_{10}$ ,  $(Y)_8 = ?$   $n = \lceil \log_8(5949) \rceil$

$$5949 / 8 = 743 + 5 \Rightarrow (y_0)_8 = 5$$

$$743 / 8 = 92 + 7 \quad (y_1)_8 = 7$$

$$92 / 8 = 11 + 4 \quad 4$$

$$11 / 8 = 1 + 3 \quad 3$$

$$1 / 8 = 0 + 1 \quad 1$$

$$(Y)_8 = (13475)_8$$

- ♦ eine  $(n+m)$ -stellige nicht negative Zahl zur Basis  $b$  wird notiert:

$$(X)_b = (x_{n-1} x_{n-2} \dots x_1 x_0, x_{-1} \dots x_{-m})_b$$

$$(X)_b = \underbrace{x_{n-1} \cdot b^{n-1} + \dots + x_0 \cdot b^0}_{\begin{array}{c} n\text{-stellige} \\ \text{Vorkommazahl } (V_x)_b \end{array}} + \underbrace{x_{-1} \cdot b^{-1} + \dots + x_{-m} \cdot b^{-m}}_{\begin{array}{c} m\text{-stellige} \\ \text{Nachkommazahl } (N_x)_b \end{array}}$$

$$(X)_b = (V_X, N_X)_b = (V_X)_b + (N_X)_b$$

- $b$  Basis des Stellenwertsystems,  $b \in \{2, 3, \dots\}$
  - $n$  Anzahl der Vorkommastellen
  - $m$  Anzahl der Nachkommastellen
  - $x_i$  Ziffer an der  $i$ -ten Position,  $x_i \in \{0, 1, 2, \dots, b-2, b-1\}$
  - $x_{n-1}$  Ziffer an der höchstwertigen Stelle
  - $x_{-m}$  Ziffer an der niedrigstwertigen Stelle

# Zahlensysteme

- Umwandlung einer  $(n+m)$ -stelligen nicht negativen Zahl X zur Basis b in eine Dezimalzahl Y mit der Direktmethode

$$(Y)_{10} = (V_X, N_X)_b = \sum (x_i)_b \cdot b^i \quad \text{für } -m \leq i \leq n-1$$

Beispiel:  $(X)_2 = (1001, \underline{11})_2$ ,  $(Y)_{10} = ?$   $b = 2, n = 4, m = 2$

$$(Y)_{10} = 1 \cdot 2^3 + \cancel{0 \cdot 2^2} + \cancel{0 \cdot 2^1} + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

$$(Y)_{10} = 1 \cdot 8 + 1 \cdot 1 + 1 \cdot 0,5 + 1 \cdot 0,25$$

$$(Y)_{10} = 9 + 0,75 = (9,75)_{10}$$

Beispiel:  $(X)_5 = (31, \underline{413})_5$ ,  $(Y)_{10} = ?$   $b = 5, n = 2, m = 3$

$$(Y)_{10} = 3 \cdot 5^1 + 1 \cdot 5^0 + 4 \cdot 5^{-1} + 1 \cdot 5^{-2} + 3 \cdot 5^{-3}$$

$$(Y)_{10} = 3 \cdot 5 + 1 \cdot 1 + 4 \cdot 0,2 + 1 \cdot 0,04 + 3 \cdot 0,008$$

$$(Y)_{10} =$$

- Umwandlung einer  $(n+m)$ -stelligen nicht negativen Zahl  $X$  zur Basis  $b$  in eine Dezimalzahl  $Y$  nach dem Horner-Schema

$$(Y)_{10} = (V_Y, N_Y)_{10} = (V_X, N_X)_b = \sum (x_i)_b \cdot b^i \quad \text{für } -m \leq i \leq n-1$$

$$= \sum (x_k)_b \cdot b^k + \sum (x_i)_b \cdot b^i \quad \text{für } 0 \leq k \leq n-1 \text{ und } -m \leq i \leq -1$$

$$(N_Y)_{10} = \sum (x_i)_b \cdot b^i = x_{-1} \cdot b^{-1} + \dots + x_{-m+1} \cdot b^{-m+1} + x_{-m} \cdot b^{-m}$$

$$= b^{-1} \cdot (x_{-1} + x_{-2} \cdot b^{-1} + \dots + x_{-m+1} \cdot b^{-m+2} + x_{-m} \cdot b^{-m+1})$$

$$= \dots$$

$$= b^{-1} \cdot (x_{-1} + b^{-1} \cdot (x_{-2} + \dots + b^{-1} \cdot (x_{-m+1} + x_{-m} \cdot b^{-1}) \dots ))$$

ergibt

$$(N_Y)_{10} := 0,0$$

$$(N_Y)_{10} := b^{-1} \cdot ((x_i)_b + (N_Y)_{10}) \quad \text{für } i = \{-m, -m+1, \dots, -2, -1\}$$

# Zahlensysteme

Beispiel:

$$(X)_2 = (1, \underline{1} \underline{0} \underline{1} \underline{1})_2, (Y)_{10} = (V_Y, N_Y)_{10} = ?$$

$$(V_Y)_{10} := 0$$

$$(V_Y)_{10} := (V_Y)_{10} \cdot 2 + 1 = 0 \cdot 2 + 1 = 1$$

$$(N_Y)_{10} := 0,0$$

$$(N_Y)_{10} := 2^{-1} \cdot (\underline{1} + (N_Y)_{10}) = 0,5 \cdot (\underline{1} + 0,00) = 0,5$$

$$(N_Y)_{10} := 2^{-1} \cdot (\underline{1} + (N_Y)_{10}) = 0,5 \cdot (\underline{1} + 0,50) = 0,75$$

$$(N_Y)_{10} := 2^{-1} \cdot (\underline{0} + (N_Y)_{10}) = 0,5 \cdot (\underline{0} + 0,75) = 0,375$$

$$(N_Y)_{10} := 2^{-1} \cdot (\underline{1} + (N_Y)_{10}) = 0,5 \cdot (\underline{1} + 0,375) = 0,6\underline{875}$$

$$(Y)_{10} = (V_Y, N_Y) = 1, \underline{6875}$$

# Zahlensysteme

Beispiel:

$$(X)_5 = (31,413)_5, (Y)_{10} = (V_Y, N_Y)_{10} = ?$$

$$(V_Y)_{10} := 0$$

$$(V_Y)_{10} := (V_Y)_{10} \cdot 5 + 3 = 0 \cdot 5 + 3 = 3$$

$$(V_Y)_{10} := (V_Y)_{10} \cdot 5 + 1 = 3 \cdot 5 + 1 = \underline{16}$$

$$(N_Y)_{10} := 0,0$$

$$(N_Y)_{10} := 5^{-1} \cdot (3 + (N_Y)_{10}) = 0,2 \cdot (3 + 0) = 0,6$$

$$(N_Y)_{10} := 5^{-1} \cdot (1 + (N_Y)_{10}) = 0,2 \cdot (1 + 0,6) = 0,32$$

$$(N_Y)_{10} := 5^{-1} \cdot (4 + (N_Y)_{10}) = 0,2 \cdot (4 + 0,32) = 0,864$$

$$(Y)_{10} = (16,864)_{10}$$

- ◆ Umwandlung einer nicht negativen Dezimalzahl X in eine (n+m)-steligen Zahl Y zur Basis b:
  - gegeben  $(X)_{10} = (V_x, N_x)_{10} \geq 0$ , und Anzahl der Nachkommastellen m
  - gesucht  $(Y)_b = (y_{n-1}y_{n-2}\dots y_1y_0, y_{-1}y_{-2}\dots y_{-m})_b$  mit  $b \geq 2$
  - Lösung: Aufteilung von  $(X)_{10}$  in  $(V_x, N_x)_{10}$ , Vorkommastellen  $(V_x)_{10}$  werden mit der Division-mit-Rest-Methode bestimmt, Nachkommastellen  $(N_x)_{10}$  werden mit **Multiplikation-mit-Abschneiden-Methode** berechnet:

für  $i := \{-1, -2, \dots, -m-1, -m\}$  berechne

→ 1. Schritt:  $(N_x)_{10} := (N_x)_{10} \cdot b$

2. Schritt:  $(y_i)_b := \underline{\text{trunc}}((N_x)_{10})$

3. Schritt:  $(N_x)_{10} := (N_x)_{10} - (y_i)_b$

mit einem Restfehler  $\varepsilon = (N_x)_{10} \cdot b^{-m}$

Funktion  $\text{trunc}(x)$  schneidet Nachkommastellen von x ab, (sie rundet also x in Richtung Null)

*trennt vor und nachkommastellen*

# Zahlensysteme

Beispiel:  $(X)_{10} = (251,79)_{10}$ ,  $m=8$ ,  $(Y)_2 = ?$

$$(X)_{10} = (V_X, N_X)_{10} \Rightarrow (V_X)_{10} = (251)_{10} \Rightarrow (V_Y)_2 = (\underline{\underline{11111011}})_2$$

$$(N_X)_{10} = (0,79)_{10}$$

1	$0,79 \cdot 2 = 1,58 = \underline{1} + 0,58 \Rightarrow (y_{-1})_2 = \underline{1}$	↓
2	$0,58 \cdot 2 = 1,16 = \underline{1} + 0,16 \Rightarrow (y_{-2})_2 = \underline{1}$	
3	$0,16 \cdot 2 = 0,32 = \underline{0} + 0,32 \Rightarrow (y_{-3})_2 = \underline{0}$	
4	$0,32 \cdot 2 = 0,64 = 0 + 0,64 \Rightarrow (y_{-4})_2 = 0$	
5	$0,64 \cdot 2 = 1,28 = 1 + 0,28 \Rightarrow (y_{-5})_2 = 1$	
6	$0,28 \cdot 2 = 0,56 = 0 + 0,56 \Rightarrow (y_{-6})_2 = 0$	
7	$0,56 \cdot 2 = 1,12 = 1 + 0,12 \Rightarrow (y_{-7})_2 = 1$	
<u>8</u>	$0,12 \cdot 2 = 0,24 = 0 + \underline{0,24} \Rightarrow (y_{-8})_2 = 0$	

$$(N_Y)_2 = (11001010)_2 \quad \text{mit Restfehler } \varepsilon = 0,24 \cdot 2^{-8} = 0,0009357$$

$$(Y)_2 = (V_Y, N_Y)_2 = (11111011, 11001010)_2 \quad \text{mit } \varepsilon = 0,24 \cdot 2^{-8}$$

Beispiel:

$$(X)_{10} = (251,79)_{10}, \quad m=3, \quad (Y)_8 = ?$$

$$(X)_{10} = (V_X, N_X)_{10} \Rightarrow (V_X)_{10} = (251)_{10} \Rightarrow (V_Y)_8 = (373)_8$$

$$(N_X)_{10} = (0,79)_{10}$$

- ♦ Im Stellenwertsystem erfolgen die Addition und die Subtraktion ( $n+m$ )-stelliger nicht negativer Zahlen positionswise, und zwar beginnend mit der niedrigstwertigen Stelle (von rechts nach links).
  - **Übertrag:** bei der Addition zweier Ziffer an der  $i$ -ten Stelle kann ein Übertrag (Carry) auf die höherwertige Stelle ( $i+1$ -te Stelle) auftreten.  
Der Übertrag von der höchstwertigen Stelle ( $n-1$ -ten Stelle) auf die  $n$ -te Stelle wird im Ergebnis als die neue höchstwertige Stelle mit der Ziffer 1 interpretiert.
  - **Borgen:** bei der Subtraktion zweier Ziffer an der  $i$ -ten Stelle kann ein Borgen (Borrow) von der höherwertigen Stelle ( $i+1$ -te Stelle) auftreten.

# Rechenarithmetik

- ◆ Addition und Subtraktion zweier  $(n+m)$ -stelligen nicht negativen Zahlen  $(X)_b$  und  $(Y)_b$  zur Basis  $b \geq 2$   
 für  $-m \leq i \leq n-1$        $i := \{-m, -m-1, \dots, -2, -1, 0, 1, 2, \dots, n-2, n-1\}$

	Bedingung	Übertrag	Ergebnis der Operation
Addition	wenn $x_i + y_i + c_i < b$	$c_{i+1} := 0$	$s_i := x_i + y_i + c_i$
	wenn $x_i + y_i + c_i \geq b$	$c_{i+1} := 1$	$s_i := x_i + y_i + c_i - b$
Subtraktion	wenn $x_i - y_i - c_i \geq 0$	$c_{i+1} := 0$	$s_i := x_i - y_i - c_i$
	wenn $x_i - y_i - c_i < 0$	$c_{i+1} := 1$	$s_i := x_i - y_i - c_i + b$

mit der Annahme, daß  $(Y)_b < (X)_b$

- $s_i$  Ergebnis der Addition/Subtraktion an der  $i$ -ten Stelle
- $c_{i+1}$  Übertrag von der  $i$ -ten auf die  $i+1$ -te Stelle bei der Addition  
 (Borgen von der  $i+1$ -ten Stelle bei der Subtraktion)
- $c_{-m} := 0$  (Initialisierungswert für Übertrag/Borgen)

Beispiel:

gegeben  $(X)_{10} = (6172,765)_{10}$  und  $(Y)_{10} = (1253,174)_{10}$

gesucht  $(Z)_{10} = (X)_{10} + (Y)_{10}$

Lösung:  $b = 10$ ,  $n = 4$ ,  $m = 3$ ,  $i := \{-3, -2, -1, 0, 1, 2, 3\}$ ,  $c_{-3} := 0$

i	$x_i$	$y_i$	$c_i$	$x_i + y_i + c_i$	Bed.	$c_{i+1}$	Kor.	$s_i$
-3	5	4	0	9	$< 10$	0	-	9
-2	6	7	0	13	$\geq 10$	1	$13 - 10$	3
-1	7	1	1	9	$< 10$	0	-	9
0	2	3	0	5	$< 10$	0	-	5
1	7	5	0	12	$\geq 10$	1	$72 - 70$	2
2	1	2	1	4	$< 10$	0	-	4
3	6	1	0	7	$< 10$	0	-	7



$$(Z)_{10} = (7425,939)_{10}$$

Beispiel:

gegeben  $(X)_{10} = (6172,736)_{10}$  und  $(Y)_{10} = (1255,174)_{10}$

gesucht  $(Z)_{10} = (X)_{10} - (Y)_{10}$

Lösung:  $b = 10$ ,  $n = 4$ ,  $m = 3$ ,  $i := \{-3, -2, -1, 0, 1, 2, 3\}$ ,  $c_{-3} := 0$

i	$x_i$	$y_i$	$c_i$	$x_i - y_i - c_i$	Bed.	$c_{i+1}$	Kor.	$s_i$
-3	6	4	0	2	$\geq 0$	0	-	2
-2	3	7	0	-4	$< 0$	1	-4+10	6
-1	7	1	1	5	$\geq 0$	0	-	5
0	2	5	0	-3	$< 0$	1	-3+10	7
1	7	5	1	1	$\geq 0$	0	-	1
2	1	2	1	-2	$< 0$	1	-2+10	8
3	6	1	1	4	$\geq 0$	0	-	4

$$(Z)_{10} = (4817,562)_{10}$$

Carry  $\cancel{0}011111\ 1010$

$$(X)_2 = \begin{array}{r} 100110,0101 \\ + 001101,1101 \\ \hline (110100,0010)_2 \end{array}$$

Carry  $\cancel{1}0\ 0011\ 0000$

$$(X)_2 = \begin{array}{r} 110110,0101 \\ - 101101,1101 \\ \hline (\underline{\underline{1001000,1000}})_2 \\ (1000,1)_2 \end{array}$$

Carry  $\cancel{0}0101\ 10$

$$(X)_8 = \begin{array}{r} 5321,41 \\ + 1071,67 \\ \hline (6413\ 30)_8 \end{array}$$

Carry  $\cancel{0}0111\ 10$

$$(X)_8 = \begin{array}{r} 5321,41 \\ - 1071,67 \\ \hline (4227,52)_8 \end{array}$$

- ◆ In der Darstellung positiver und negativer n-stelliger Zahlen zur Basis b wird die höchstwertige Ziffer  $x_{n-1}$  als das Vorzeichen interpretiert:
  - $x_{n-1} = 0$  für positive Zahlen
  - $x_{n-1} = b-1$  für negative Zahlen (z.B. F für HEX, 7 für OCT, 1 für BIN)
- ◆ für positive Zahl gilt immer:  $(X)_b = (0.x_{n-2}x_{n-3} \dots x_1x_0)_b$
- ◆ zur Darstellung negativer Zahlen gibt es drei Möglichkeiten:
  - Vorzeichen und Betrag
  - $(b-1)$ -Komplement
  - $b$ -Komplement
- ◆ in den Darstellungen „Vorzeichen und Betrag“ sowie  $(b-1)$ -Komplement hat die Zahl 0 jeweils zwei Darstellungen (sog. positive Null und negative Null).

# Rechenarithmetik

- ◆ für Dualzahlen (d.h. zur Basis  $b=2$ ) ergibt sich:
  - Bei Verwendung von **Vorzeichen und Betrag** wird zur Codierung einer negativen  $n$ -stelligen Zahl  $(X)_2$  das Vorzeichenbit  $x_{n-1}$  invertiert.
  - Beispiel:  $(45)_{10} = (0.101101)_2 \quad -(45)_{10} = (1.101101)_2$
  - Das  $(b-1)$ -Komplement wird **Einerkomplement** genannt; die Codierung von  $-(X)_2$  erhält man, indem man alle Bitstellen einer positiven  $n$ -stelligen Zahl  $(X)_2$  invertiert.
  - Beispiel:  $(45)_{10} = (0.101101)_2 \quad -(45)_{10} = (1.010010)_2$
  - Das  $b$ -Komplement wird **Zweierkomplement** genannt; die Codierung von  $-(X)_2$  erhält man, indem man zum Einerkomplement eine  $(1)_{10} = (000\dots01)_2$  hinzufügt.
  - Beispiel:  $(45)_{10} = (0.101101)_2 \quad -(45)_{10} = (1.010011)_2$

- ◆ **Einerkomplement:** der darstellbare Zahlenbereich für n-stellige Dualzahlen  $(X)_2$ :  $-2^{n-1} + 1 \leq X \leq 2^{n-1} - 1$
- ◆ Beispiel: für  $n = 8$ 
  - kleinste negative darstellbare Zahl:  $(1.0000000)_2 = -127_{10}$
  - größte negative darstellbare Zahl:  $(1.1111110)_2 = -1_{10}$
  - größte positive darstellbare Zahl:  $(0.1111111)_2 = 127_{10}$
  - alle Zahlen  $-127 \leq X \leq 127$  können dargestellt werden
- ◆ Vor- und Nachteile des Einerkomplements:
  - + der darstellbare Zahlenbereich ist symmetrisch zu 0
  - + sehr einfache Umwandlung von positiver zu negativer Zahl und umgekehrt durch Invertierung aller Bits ( $0 \Leftrightarrow 1$ )
  - zwei Darstellungen für die Null:  $(0.00\dots00)_2$  und  $(1.11\dots11)_2$
  - Addierwerke sind aufwendig, weil das Ergebnis beim Auftreten negativer Zahlen in manchen Fällen korrigiert werden muß.  
=> Einerkomplement wird daher i.a. nicht verwendet.

- ◆ **Zweierkomplement:** der darstellbarere Zahlenbereich für n-stellige Dualzahlen  $(X)_2$ :  $-2^{n-1} \leq X \leq 2^{n-1}-1$
- ◆ Beispiel: für  $n = 8$ 
  - kleinste negative darstellbare Zahl:  $(1.0000000)_2 = -128_{10}$
  - größte negative darstellbare Zahl:  $(1.1111111)_2 = -1_{10}$
  - größte positive darstellbare Zahl:  $(0.1111111)_2 = 127_{10}$
  - alle Zahlen  $-128 \leq X \leq 127$  können dargestellt werden
- ◆ Vor- und Nachteile des Zweierkomplements:
  - + eindeutige Darstellung der Null als  $(0.00\dots00)_2$
  - + einfache Realisierung der Addition auch beim Auftreten negativer Zahlen ohne zusätzlichen Aufwand
  - darstellbarer Zahlenbereich ist asymmetrisch
  - Umwandlung von positiver zu negativer Zahl und umgekehrt erfordert die Invertierung aller Bits sowie eine Addition von  $(0.00..01)_2$

Beispiel:

$$(X)_{10} = -(109)_{10} \Rightarrow (X)_2 = -(1101101)_2 \quad (\text{7-stellige Zahl})$$

Erweiterung der Zahl um das Vorzeichen zu einer 8-stelligen Zahl

$$(X)_2 = -(0.1101101)_2$$

$$\text{1-Komplement: } -(0.1101101)_2 \Rightarrow (1.0010010)_2$$

$$\text{2-Komplement: } 1\text{-Komplement} + (0.00..01)_2$$

$$\begin{aligned} -(0.1101101)_2 &\Rightarrow (1.0010010)_2 + (0.0000001)_2 \\ &= (1.0010011)_2 \end{aligned}$$

$$(X)_{10} = -(317,75)_{10} \Rightarrow (X)_2 = -(100111101,11)_2 \quad (\text{11-stellige Zahl})$$

Erweiterung der Zahl um das Vorzeichen zu einer 12-stelligen Zahl

$$(X)_2 = -(0.100111101,11)_2$$

$$\text{1-Komplement: } -(0.100111101,11)_2 \Rightarrow (1.011000010,00)_2$$

$$\begin{aligned} \text{2-Komplement: } -(0.100111101,11)_2 &\Rightarrow (1.011000011,01)_2 \\ &\quad +1 \end{aligned}$$

# Rechenarithmetik

- ◆ Addition und Subtraktion zweier  $(n+m)$ -stelligen Dualzahlen  $(X)_2$  und  $(Y)_2$  erfolgt positionsweise

für $-m \leq i \leq n-1$			Addition		Subtraktion	
$c_i$	$x_i$	$y_i$	$s_i$	$c_{i+1}$	$s_i$	$c_{i+1}$
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
1	0	0	1	0	1	1
1	0	1	0	1	0	1
1	1	0	0	1	0	0
1	1	1	1	1	1	1

- $s_i$  Ergebnis der Addition/Subtraktion an der  $i$ -ten Stelle
- $c_{i+1}$  Übertrag von der  $i$ -ten auf die  $i+1$ -te Stelle bei der Addition  
(Borgen von der  $i+1$ -ten Stelle bei der Subtraktion)

# Rechenarithmetik

Carry ~~0 0~~ 111100

$$\begin{array}{r}
 (X)_2 = 0.0110111 \quad (55)_{10} \\
 (Y)_2 = + 0.0101010 \quad (42)_{10} \\
 \hline
 \underline{0.\underline{1101001}} \quad (97)_{10}
 \end{array}$$

*g7*

Carry 11 1101100

$$\begin{array}{r}
 (X)_2 = 0.0110111 \quad (55)_{10} \\
 (Y)_2 = + 1.1010110 \quad (-42)_{10} \\
 \hline
 \underline{0.\underline{0001101}} \quad (13)_{10}
 \end{array}$$

*13*

Carry 11 0000000

$$\begin{array}{r}
 (X)_2 = 1.1001001 \quad (-55)_{10} \\
 (Y)_2 = + 1.1010110 \quad (-42)_{10} \\
 \hline
 \underline{1.0011111} \quad (-97)_{10}
 \end{array}$$

$\downarrow \downarrow$

$$\begin{array}{r}
 -(0.1100000)_2 \\
 + \underline{\underline{0.1100001}}_2 \\
 \hline
 -(0.1100001)_2
 \end{array}$$

(97)<sub>10</sub>

Carry 00 0010000

$$\begin{array}{r}
 (X)_2 = 0.0101010 \quad (42)_{10} \\
 (Y)_2 = + 1.1001001 \quad (-55)_{10} \\
 \hline
 \underline{1.1110011} \quad (-13)_{10}
 \end{array}$$

$\downarrow \downarrow$

$$\begin{array}{r}
 -(0.0001100)_2 \\
 + \underline{\underline{1}} \\
 - (0.0001101)_2 \\
 \hline
 \end{array}$$

(-13)<sub>10</sub>

# Rechenarithmetik

Carry  $\underline{\underline{00\ 001000}}0$

$$\begin{array}{r} (X)_2 = 0.0110111 \quad (55)_{10} \\ (Y)_2 = - 0.0101010 \quad (42)_{10} \\ \hline \underline{\underline{0.\ 0001101}} \quad (13)_{10} \end{array}$$

$\overbrace{\phantom{0.0001101}}^{13}$

Carry  $\underline{\underline{11\ 000000}}0$

$$\begin{array}{r} (X)_2 = 0.0110111 \quad (55)_{10} \\ (Y)_2 = - 1.1010110 \quad (-42)_{10} \\ \hline \underline{\underline{0.\ 1100001}} \quad (97)_{10} \end{array}$$

$\overbrace{\phantom{0.1100001}}^{97}$

Carry  $\underline{\underline{11\ 110111}}0$

$$\begin{array}{r} (X)_2 = 0.0101010 \quad (42)_{10} \\ (Y)_2 = - 0.0110111 \quad (55)_{10} \\ \hline \underline{\underline{1.1110011}} \quad (-13)_{10} \end{array}$$

↓  
||

$$\begin{array}{r} - (0.0001100)_2 \\ + \underline{\underline{(0.0001101)}_2}^1 \\ - \underline{\underline{(0.0001101)}_2} \\ - \quad (13)_{10} \end{array}$$

Carry ~~00~~ 0010010

$$\begin{array}{r}
 (X)_2 = 1.1010110 \quad (-42)_{10} \\
 (Y)_2 = -1.1001001 \quad (-55)_{10} \\
 \hline
 \underline{0.0001101} \quad (13)_{10} \\
 + (1101)_2
 \end{array}$$

Carry ~~11~~ 1101100

$$\begin{array}{r}
 (X)_2 = 1.1001001 \quad (-55)_{10} \\
 (Y)_2 = -1.1010110 \quad (-42)_{10} \\
 \hline
 1.110011 \quad (-13)_{10}
 \end{array}$$

$$\begin{array}{r}
 - (0.0001100)_2 \\
 + \underline{\hspace{1cm}} \\
 - (0.001101)_2 \\
 - (1101)_2
 \end{array}$$

Carry ~~00~~ 1111110

$$\begin{array}{r}
 (X)_2 = 1.1010110 \quad (-42)_{10} \\
 (Y)_2 = -0.0110111 \quad (55)_{10} \\
 \hline
 \underline{1.0011111} \quad (-97)_{10}
 \end{array}$$

$$\begin{array}{r}
 - (0.1100000)_2 \\
 + \underline{\hspace{1cm}} \\
 - (0.1100001)_2 \\
 - (97)_2
 \end{array}$$

- ♦ Subtraktion zweier  $(n+m)$ -stelligen Zahlen zur Basis  $b$  kann durch eine Addition mit einem  $b$ -Komplement ersetzt werden mit dem Ansatz:

$$(X)_b - (Y)_b = (X)_b + (-Y)_b = (X)_b + (b^n - (Y)_b)$$

- ♦ Praktische Realisierung der Subtraktion zweier  $(n+m)$ -stelligen Dualzahlen  $(X)_b - (Y)_b$ 
  1. Bildung des Einerkomplements von  $Y$  durch Invertierung aller Bitstellen
  2. Bildung des Zweierkomplements von  $Y$  durch Addition von  $(0..01)_2$  (in der Praxis wird dieser Schritt aus Effizienzgründen durch das Setzen der ersten Übertragsposition  $c_0$  bzw.  $c_{-m}$  auf 1 realisiert)
  3. Addition der beiden Dualzahlen
  4. Übertrag von  $c_{n-1}$  auf  $c_n$  wird ignoriert

$$55 - 42 \equiv 55 + (-42)$$

$$(X)_2 = 0.0110111 \quad (55)_{10}$$

$$(Y)_2 = -0.0101010 \quad (42)_{10}$$

---


$$(13)_{10}$$

Carry  $\cancel{1}11101111$

$$(X)_2 = 0.0110111 \quad (55)_{10}$$

$$(Y)_2 = +1.1010101 \quad (-42)_{10}$$

---


$$\underline{0.0001101} \quad (13)_{10}$$

$$(X)_2 = 1.1001001 \quad (-55)_{10}$$

$$(Y)_2 = -1.1010110 \quad (-42)_{10}$$

---


$$(-13)_{10}$$

Carry  $\cancel{0}00010001$

$$(X)_2 = 1.1001000 \quad (-55)_{10}$$

$$(Y)_2 = +0.0101010 \quad (42)_{10}$$

---


$$\underline{1.1110011} \quad (-13)_{10}$$

$$(-55) - (-42)$$

$$\equiv (-55) + (42)$$

$$\begin{array}{r}
 - (0.0001100)_2 \\
 + \hline
 - (0.0001101)_2 \\
 - (1101)_2
 \end{array}$$

## ♦ Boolesche Algebra

- ist eine algebraische Struktur, definiert durch
  - eine zweielementige Wertemenge  $B = \{0, 1\}$  (sog. boolesche Konstanten), (alternative Schreibweise: { Low, High }, { False, True }, { falsch, wahr }, { offen, geschlossen })
  - einen einstelligen Operator { ' } (NICHT-Operator)
  - zwei zweistellige Operatoren { . , + } (UND- und ODER-Operatoren)
  - mit folgender Rangfolge der Operatoren: NICHT vor UND vor ODER mit Klammern (...) kann die Rangfolge geändert werden
  - zehn Axiome { A1, ..., A10 }
  - neun Gesetze { G1, ..., G9 }
- oft ein Synonymbegriff für Schaltalgebra
- Die boolesche Algebra ist ein Formalismus, das uns ermöglicht, das Verhalten von Digitalschaltungen mit formalen Methoden zu beschreiben, zu analysieren und zu minimieren, ohne diese Schaltung technisch aufbauen zu müssen.

- ◆ Boolesche Variable

- ein symbolischer Bezeichner (Name), bestehend aus Buchstaben und ggf. Ziffern, der nur Werte aus der Wertemenge B annimmt.

$x, A, \text{Sum}, \text{carry}$

- ◆ Boolescher Term (Formel, Ausdruck)

- eine Zeichenkette, bestehend aus booleschen Konstanten, booleschen Variablen sowie booleschen Operatoren und Funktionen.

$x \cdot (y + 0) + b \cdot 1$

- ◆ n-stellige boolesche Funktion  $y = f(e_0, e_1, e_2, \dots, e_{n-1})$

- diskrete Zuordnung  $f$  zwischen einer (abhängigen) booleschen Variable  $y$  und (unabhängigen) booleschen Variablen  $e_0, e_1, \dots, e_{n-1}$  (Funktionsargumenten, Operanden, Eingangsvariablen).

$$S = f(a, b, c) = c \cdot a + a \cdot b + b \cdot c$$

- ◆ Funktionstabelle (Wahrheitstabelle)
  - geordnete, tabelarische Darstellung boolescher Funktionen
  - In einer Funktionstabelle sind die Ergebnisse der booleschen Funktionen für sämtliche Kombinationsmöglichkeiten der beteiligten Funktionsargumente aufgelistet.
  - Für  $n$  Funktionsargumente ergeben sich  $2^n$  Zeilen

Funktions-argumente			Funktions-werte	
u	x	y	s	v
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

Funktionstabelle für zwei Funktionen  
 $s = f(x, y, u)$   
 $v = g(x, y, u)$   
mit drei Argumenten x, y und u

# Boolesche Algebra

## ♦ NICHT-Operator

- Boolescher Komplement
- ist definiert durch
  - eine Eingangsvariable, die nur die Werte 0 oder 1 annehmen kann,
  - eine Ausgangsvariable, die genau den entgegengesetzten Wert der Eingangsvariable annimmt.
  - Wahrheitstabelle des NICHT-Operators:

E	E'
0	1
1	0

- Symbolik für die Notation: ' (  $\neg$ ,  $\neg$ ,  $\sim$ , !, /, not, nicht)  
Beispiel: X'

## ♦ UND-Operator

- Boolesches Produkt, Konjunktion
- ist definiert durch
  - zwei Eingangsvariablen, die nur die Werte 0 oder 1 annehmen können,
  - eine Ausgangsvariable, die den Wert 1 **nur dann** annimmt, wenn **beide** Eingangsvariablen gleichzeitig den Wert 1 haben.
  - Wahrheitstabelle des UND-Operators

E1	E2	$E1 \cdot E2$
0	0	0
0	1	0
1	0	0
1	1	1

- Symbolik für die Notation:  $\cdot$  ( $\&$ ,  $*$ ,  $\wedge$ , and, und)  
Beispiel:  $X \cdot Y$

# Boolesche Algebra

## ♦ ODER-Operator

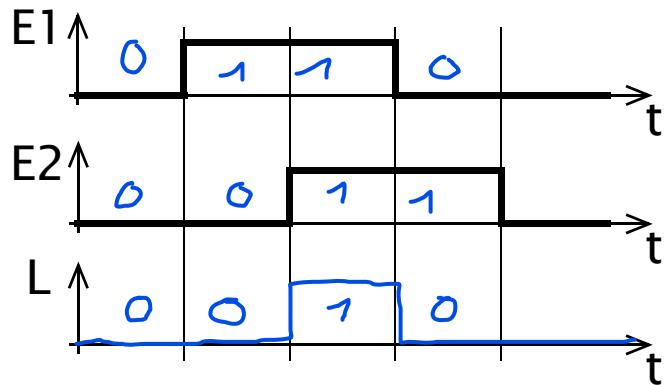
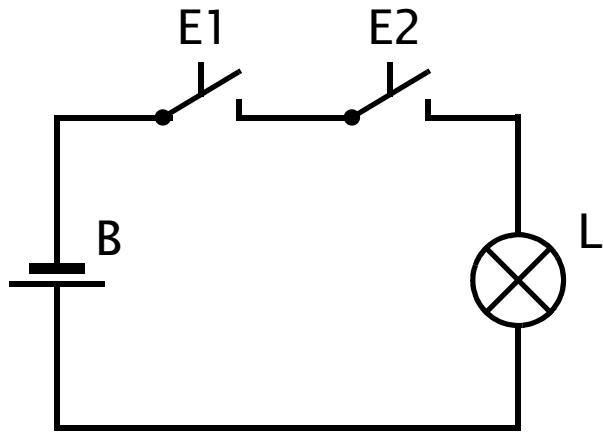
- Boolesche Summe, Disjunktion
- ist definiert durch
  - zwei Eingangsvariablen, die nur die Werte 0 oder 1 annehmen können,
  - eine Ausgangsvariable, die den Wert 1 dann annimmt, wenn **mindestens eine** Eingangsvariable den Wert 1 hat.
  - Wahrheitstabelle des ODER-Operators:

E1	E2	$E1+E2$
0	0	0
0	1	1
1	0	1
1	1	1

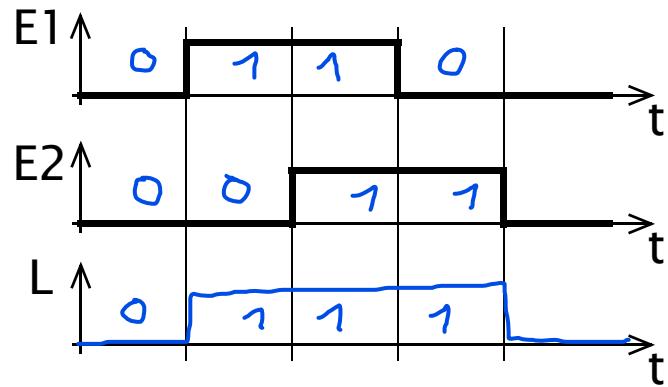
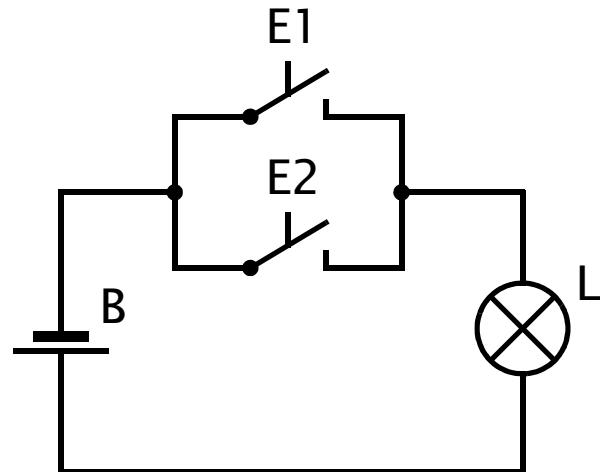
- Symbolik für die Notation: + ( |,  $\vee$ , or, oder)  
Beispiel:  $X+Y$

- Technische Realisierungen mit Schaltern

UND-Operator



ODER-Operator



- ♦ Rekonstruktion einer Funktionstabelle als einer booleschen Funktion

$$f(a, b, c) = a \cdot b + (b + c)'$$

$$f(0, 0, 0) = 0 \cdot 0 + (0 + 0)' = 0 + 0' = 0 + 1 = 1$$

$$f(0, 0, 1) = 0 \cdot 0 + (0 + 1)' = 0 + 1' = 0 + 0 = 0$$

$$f(0, 1, 0) = 0 \cdot 1 + (1+0)' = 0 + 1' = 0 + 0 = 0$$

$$f(0, 1, 1) = 0 \cdot 1 + (1+1)' = 0 + 1' = 0 + 0 = 0$$

$$f(1, 0, 0) = 1 \cdot 0 + (0+0)' = 0 + 0' = 0 + 1 = 1$$

$$f(1, 0, 1) = 1 \cdot 0 + (0+1)' = 0 + 1' = 0 + 0 = 0$$

$$f(1, 1, 0) = 1 \cdot 1 + (1+0)' = 1 + 1' = 1 + 0 = 1$$

$$f(1, 1, 1) = 1 \cdot 1 + (1+1)' = 1 + 1' = 1 + 0 = 1$$

a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- ♦ Substitutions-/Ersetzungsregeln:

- jede Variable in einem booleschen Ausdruck darf durch eine neue (d.h. in diesem Ausdruck noch nicht vorkommende) boolesche Variable ersetzt werden.
- jeder (Teil-)Term eines booleschen Ausdrucks darf durch eine neue boolesche Variable ersetzt werden, um so den gesamten Ausdruck zu vereinfachen und letztendlich auf einen bekannten oder einfacher handhabbaren Ausdruck zurückzuführen.
- jeder (Teil-)Term eines booleschen Ausdrucks darf durch einen anderen (äquivalenten) Ausdruck ersetzt werden, wenn er den gleichen Funktionswert wie der ersetzte Ausdruck hat, ohne den Funktionswert des gesamten booleschen Ausdruckes zu verändern.

- ◆ Kommutativitätsaxiom

- Vertauschung der Reihenfolge der Variablen

$$A1: a + b = b + a$$

$$A2: a \cdot b = b \cdot a$$

- ◆ Assoziativitätsaxiom

- Vertauschung der Reihenfolge gleichrangiger Operatoren

$$A3: a + b + c = (a + b) + c = a + (b + c)$$

$$A4: a \cdot b \cdot c = (a \cdot b) \cdot c = a \cdot (b \cdot c)$$

- ◆ Distributivitätsaxiom

- Auflösung von Klammerausdrücken unter Berücksichtigung der Rangfolge von Operatoren

$$A5: a \cdot (b + c) = a \cdot b + a \cdot c$$

$$A6: a + (b \cdot c) = (a + b) \cdot (a + c)$$

- ♦ Identitätsaxiom

- 0 ist ein neutrales Element für den ODER-Operator

$$A7: a + 0 = 0 + a = a$$

- 1 ist ein neutrales Element für den UND-Operator

$$A8: a \cdot 1 = 1 \cdot a = a$$

- ♦ Komplementär-Axiom

- ODER-Verknüpfung mit einem inversen Element ergibt 1

$$A9: a + a' = a' + a = 1$$

- UND-Verknüpfung mit einem inversen Element ergibt 0

$$A10: a \cdot a' = a' \cdot a = 0$$

- ♦ Axiome lassen sich mit Hilfe von Wahrheitstabellen beweisen

Beispiel: das Distributivität-Axiom A6 ist zu beweisen

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

a	b	c	$x = b \cdot c$	$a + x$	$y = a + b$	$z = a + c$	$y \cdot z$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

- Die unten stehenden Gleichung ist zu beweisen:

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

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

[Sub : e := c+d]  $\sim$

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

[A5]

$$= a \cdot e + b \cdot e$$

[Sub<sup>-1</sup> e]

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

[2x A5]

$$= a \cdot c + a \cdot d + b \cdot c + b \cdot d$$

# Boolesche Algebra



## Idempotenzgesetze

- mehrfach auftretende Variablen dürfen bis auf eine gestrichen werden. Die gleiche Variable darf beliebig oft in eine UND- oder ODER- Verknüpfung hinzugefügt werden.
- G1:  $a + a = a$
- G2:  $a \cdot a = a$

## Dominanzgesetze

- 1 dominiert eine ODER-Verknüpfung, und 0 eine UND-Verknüpfung
- G3:  $a + 1 = 1 + a = 1$
- G4:  $a \cdot 0 = 0 \cdot a = 0$

## Absorptionsgesetze

- G5:  $a \cdot (a + b) = a$
- G6:  $a + (a \cdot b) = a$

- ◆ Doppelte-Negation-Gesetz
  - wird eine boolesche Variable oder ein boolesche Ausdruck zweifach negiert, so heben sich die Negierungen auf.
  - G7:  $(a')' = a'' = a$
- ◆ erstes Gesetz von De Morgan (NOR-Operation)
  - negiert man eine ODER-Verknüpfung, so ist dies zu einer UND-Verknüpfung äquivalent, in der die einzelnen Variablen negiert sind.
  - G8:  $(a + b)' = a' \cdot b'$
  - Negation einer Disjunktion ist die Konjunktion von Negationen
- ◆ zweites Gesetz von De Morgan (NAND-Operation)
  - negiert man eine UND-Verknüpfung, so ist dies zu einer ODER-Verknüpfung äquivalent, in der die einzelnen Variablen negiert sind.
  - G9:  $(a \cdot b)' = a' + b'$
  - Negation einer Konjunktion ist die Disjunktion von Negationen

## ♦ Antivalenz-Operator

- XOR-Operator:  $a \oplus b = a' \cdot b + a \cdot b'$
- ist definiert durch
  - zwei Eingangsvariablen, die nur die Werte 0 oder 1 annehmen können,
  - eine Ausgangsvariable, die den Wert 1 dann annimmt, wenn **genau eine** der Eingangsvariablen den Wert 1 hat.
  - Wahrheitstabelle des XOR-Operators:

E1	E2	$E1 \oplus E2$
0	0	0
0	1	1
1	0	1
1	1	0

- Symbolik für die Notation:  $\oplus$  ( $\bar{\vee}$ ,  $\neq$ , xor)  
Beispiel:  $X \oplus Y$

- ◆ Äquivalenz-Operatoren

- XNOR-Operator:  $(a \equiv b) = (a \oplus b)' = a' \cdot b' + a \cdot b$
- ist definiert durch
  - zwei Eingangsvariablen, die nur die Werte 0 oder 1 annehmen können,
  - eine Ausgangsvariable, die den Wert 1 dann annimmt, wenn **beide** Eingangsvariablen den gleichen Wert haben, egal ob 0 oder 1.
  - Wahrheitstabelle des XNOR-Operators:

E1	E2	$E1 \equiv E2$
0	0	1
0	1	0
1	0	0
1	1	1

- Symbolik für die Notation:  $\equiv$  ( $\leftrightarrow$ ,  $=$ ,  $xnor$ )  
Beispiel:  $X \equiv Y$

- ◆ Beweisbeispiel:
  - das Absorptionsgesetz G5:  $a \cdot (a + b) = a$  ist zu beweisen

$a=R$	b	$x=a+b$	$a \cdot x=L$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

✓  
✓  
✓  
✓

$$G5: a \cdot (a + b) = a$$

$$L = a \cdot (a + b) \quad [A7]$$

$$= (a+0) \cdot (a+b) \quad [A6]$$

$$= a + (0 \cdot b) \quad [G4]$$

$$= a + 0 \quad [A7]$$

$$= a$$

## ♦ Vereinfachung boolescher Terme

- Für die algebraische Vereinfachung boolescher Terme gibt es kein allgemein gültiges Rezept, dafür aber eine Reihe heuristischer Regeln.

1. Regel: Ausdrücke ohne Klammern versucht man mit den Absorptionsgesetzen zu vereinfachen.

$$\begin{aligned} f(x, y, z) &= x \cdot y + x \cdot z + y \cdot z + y + z && [A1] \\ &= x \cdot y + y \cdot z + y + x \cdot z + z && [A8] \\ &= \underline{x \cdot y + y \cdot z} + \underline{\cancel{y} \cdot 1} + \underline{x \cdot z + z \cdot 1} && [2 \times A5] \\ &= \cancel{y} \cdot (x + z + 1) + z \cdot (x + 1) && [2 \times G3] \\ &= \cancel{y} \cdot 1 + z \cdot 1 && [2 \times AB] \\ &= \cancel{y} + z \end{aligned}$$

2. Regel: Beim Auftreten von komplexen Klammerausdrücken versucht man zunächst die Klammerinhalte zu vereinfachen.

$$f(x, y) = (x \cdot y + x' \cdot y + x' \cdot y')' = (g(x, y))'$$

$$g(x, y) = x \cdot y + x' \cdot y + x' \cdot y' \quad [A5]$$

$$= y \cdot (x + x') + x' \cdot y' \quad [A9]$$

$$= y \cdot 1 + x' \cdot y' \quad [A8]$$

$$= y + x' \cdot y' \quad [\text{Sub } a := y, b := x', c := y']$$

$$= a + b \cdot c \quad [A6] \quad [\text{Sub } a, b, c]$$

$$= (a+b) + (a+c) \quad [A9]$$

$$= (y+x') \cdot (y+y') \quad [A8]$$

$$= (y+x') \cdot 1 \quad [A8]$$

$$= y + x' \quad [A8]$$

$$f(x, y) = (y+x')' = [G8] = y \cdot x' = [G7] = \underline{\underline{y \cdot x}}$$

3. Regel: Geschachtelte Klammerausdrücke werden im allgemeinen von innen nach außen verarbeitet

$$\begin{aligned}
 f(a, b, c) &= [(a + b) \cdot (a + b') + a' \cdot b'] \quad [h \times A5] \\
 &= [(a \cdot a + a \cdot b' + b \cdot a + b \cdot b') + a' \cdot b'] \quad [A10] \\
 &= [(a \cdot a + a \cdot b' + b \cdot a + 0) + a' \cdot b'] \quad [A7, G2] \\
 &= [(a + a \cdot b' + b \cdot a) + a' \cdot b'] \quad [A8] \\
 &= [(a \cdot 1 + a \cdot b' + b \cdot a) + a' \cdot b'] \quad [A5] \\
 &= [a \cdot (1 + b' + b) + a' \cdot b'] \quad [G3, A8] \\
 &= a + a' \cdot b' \quad [A6] \\
 &= (a + a') \cdot (a + b') \quad [A9, A8] \\
 &= a + b
 \end{aligned}$$

4. Regel: Beim Auftreten von einfachen negierten Termen versucht man, diese zuerst mit den De Morgan-Gesetzen aufzulösen.

$$f(a, b, c) = (a + b') \cdot (\underline{a' \cdot b' \cdot c})'$$

[G9, 2xG7]

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

[n x A5]

$$= a \cdot a + a \cdot b + a \cdot c' + b' \cdot a + b' \cdot b + b' \cdot c'$$

[G2, A10, A7]

$$= a + a \cdot b + a \cdot c' + b' + a + \cancel{b} \cdot c'$$

[A8]

$$= a \cdot 1 + a \cdot b + a \cdot c' + b' + a + b \cdot c'$$

[A5]

$$= a \cdot (\underbrace{1 + b + c' + b'}_{\equiv 1}) + b \cdot c'$$

[G3, A8]

$$= a + b \cdot c'$$

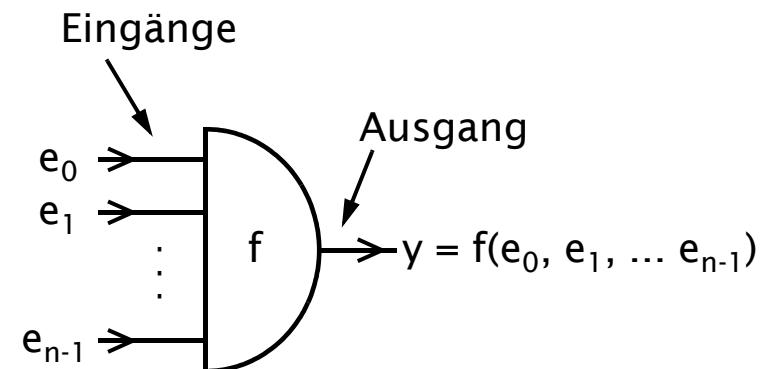
5. Regel: Durch den Einsatz komplexer Operatoren XOR und XNOR lassen sich Termen stark vereinfachen.

$$\begin{aligned}
 f(a, b, c) &= a' \cdot b' + a \cdot b' + a' \cdot b \cdot c' + a \cdot b \cdot c \quad [2 \times A5] \\
 &= b' \cdot (\underbrace{a' + a}_{\text{e}}) + b \cdot (\underbrace{a' \cdot c' + a \cdot c}_{\text{XNOR}}) \quad [A9, A8] \\
 &= b' + b \cdot (\underbrace{a' \cdot c' + a \cdot c}_{\text{XNOR}}) \quad [\text{XNOR}] \\
 &= b' + b \cdot \{a \oplus c\}' \quad [\text{Sub: } e := a \oplus c] \\
 &= b' + b \cdot e \quad [A6] \\
 &= (\underbrace{b' + b}_{\text{G9}}) \cdot (\underbrace{b' + e'}_{\text{Sub } e}) \quad [A9, A8] \\
 &= b' + e' \quad [\text{G9}] \\
 &\equiv (b \cdot e)' \quad [\text{Sub } e] \\
 &= (b \cdot (a \oplus c))'
 \end{aligned}$$

6. Regel: Bei einigen Termen führt erst eine geeignete „Expansion“ mit dem Faktor  $1=a+a'$  zur Anwendung des Absorptionsgesetzes und damit zur Vereinfachung

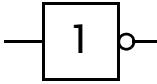
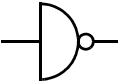
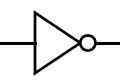
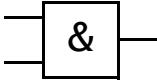
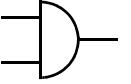
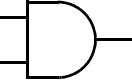
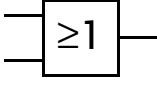
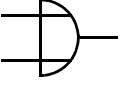
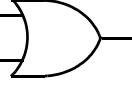
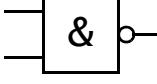
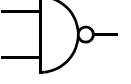
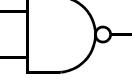
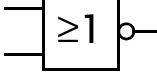
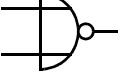
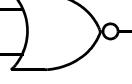
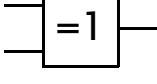
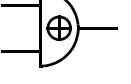
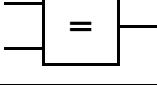
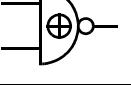
$$\begin{aligned}
 T(p, q, r) &= p' \cdot q + p \cdot r' + q \cdot r' \quad [A8] \\
 &= p' \cdot q + p \cdot r' + q \cdot r' \cdot 1 \quad [A9] \\
 &= p' \cdot q + p \cdot r' + q \cdot r' \cdot (p+p') \quad [A5] \\
 &= p' \cdot q + p \cdot r' + p \cdot q \cdot r' + p' \cdot q \cdot r' \\
 &= p' \cdot q + p' \cdot q \cdot r' + p \cdot r' + p \cdot q \cdot r' \quad [2 \cdot A8] \\
 &= p' \cdot q \cdot 1 + p' \cdot q \cdot r' + p \cdot r' \cdot 1 + p \cdot q \cdot r' \quad [2 \cdot A5] \\
 &= p' \cdot q \cdot (1+r') + p \cdot r' \cdot (1+q) \quad [2 \cdot G3] \\
 &= p' \cdot q \cdot 1 + p \cdot r' \cdot 1 \quad [2 \cdot A8] \\
 &= p' \cdot q + p \cdot r'
 \end{aligned}$$

- ♦ Technische Realisierung boolescher Funktionen
  - Eine boolesche Funktion kann in eine digitale Schaltung umgeformt werden, so daß die Schaltung den Wert der booleschen Funktion berechnet.
  - Das Grundelement einer digitalen Schaltung ist ein (Logik-)Gatter.
  - Ein Gatter ist
    - ein elektronisches, aus Transistoren aufgebautes und i.d.R. nicht weiter zerlegbares Bauelement,
    - ausgestattet mit einem, zwei oder mehreren Eingängen  $e_0, e_1, \dots, e_{n-1} \in \{0,1\}$  und genau einem Ausgang  $y \in \{0,1\}$ ,
    - zur Realisierung einer booleschen Funktion  $y = f(e_0, e_1, \dots, e_{n-1})$ .
    - Die Funktion  $f$  eines Gatters wird durch eine Funktionstabelle festgelegt.
  - Eine digitale Schaltung wird aus baumartig verketteten Gattern aufgebaut.

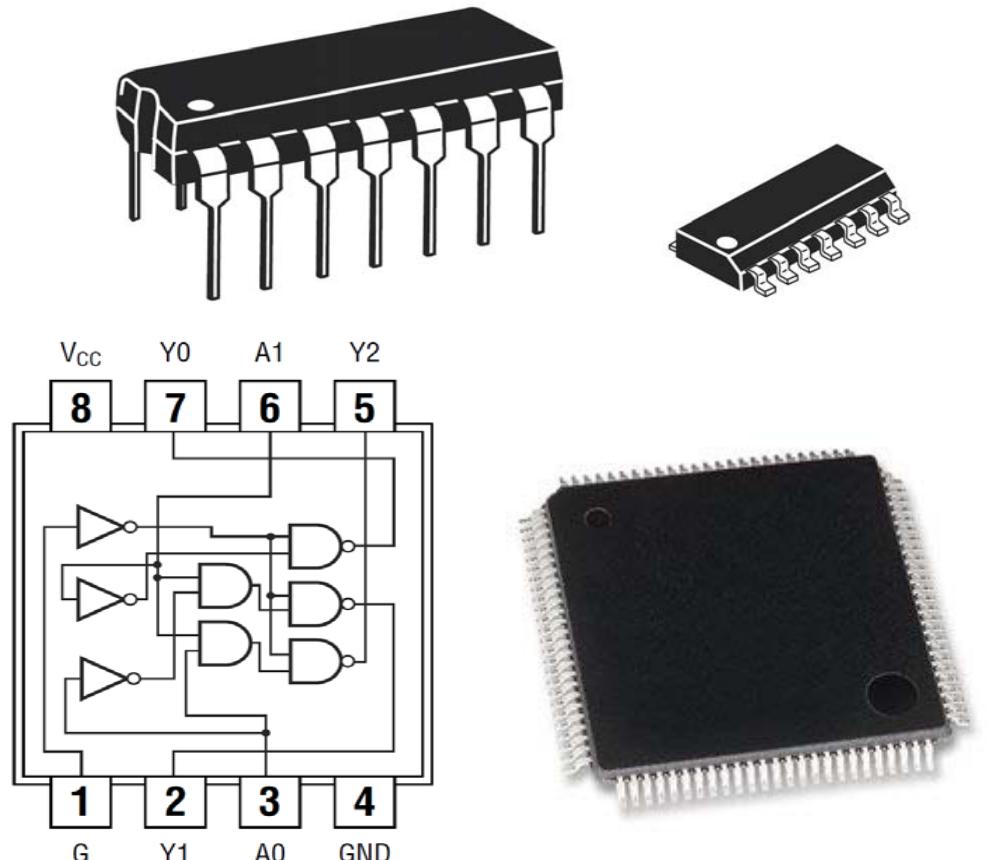
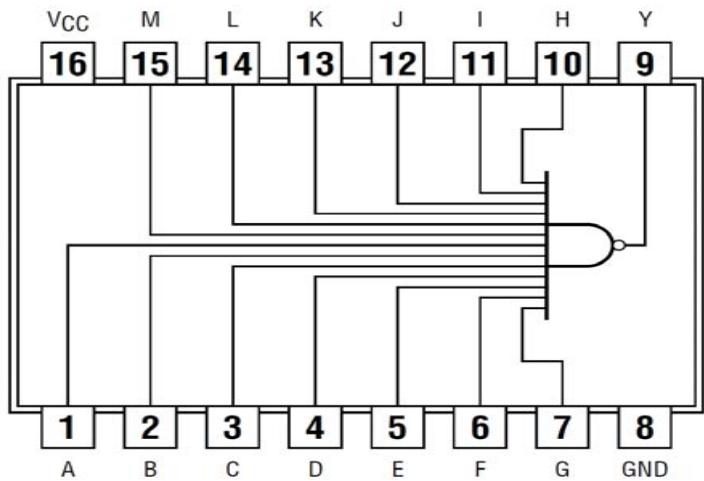
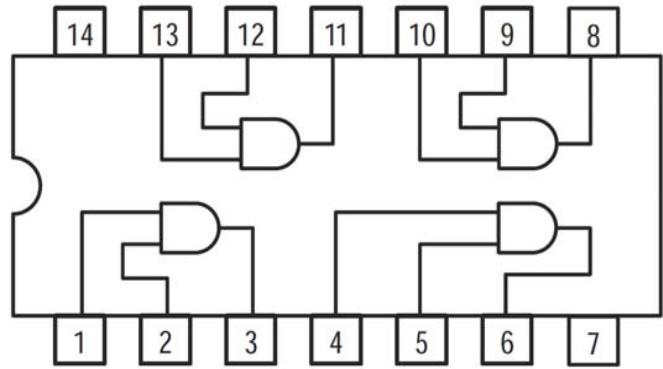


- ◆ **Schaltnetz (kombinatorische Schaltung)**
  - nach DIN IEC 748 Teil 2 ist ein Schaltnetz eine Digitalschaltung, in der es für jede mögliche Kombination von digitalen Signalen an den Eingängen eine – und nur eine – Kombination von digitalen Signalen an den Ausgängen gibt.
- ◆ **Schaltnetz (alternative Definitionen)**
  - ist eine schaltungstechnische Einheit zum Verarbeiten von Schaltvariablen, deren Wert an den Ausgängen zu einem Zeitpunkt nur von den Werten an den Eingängen zum gleichen Zeitpunkt abhängt.
  - eine Verknüpfung von logischen Grundelementen ohne interne Speicherung von Variablen

- ◆ Boolesche Verknüpfungen und Schaltsymbole

A	0	1	0	1	Verknüpfung	Funktion	Schaltsymbole/Gatter		
B	0	0	1	1			IEC 60617	DIN 40700	ANSI 91-1984
Y	1	0	-	-	$Y = A'$	NOT			
Y	0	0	0	1	$Y = A \cdot B$	AND			
Y	0	1	1	1	$Y = A + B$	OR			
Y	1	1	1	0	$Y = (A \cdot B)'$	NAND			
Y	1	0	0	0	$Y = (A + B)'$	NOR			
Y	0	1	1	0	$Y = A \oplus B$	XOR			
Y	1	0	0	1	$Y = (A \oplus B)'$	XNOR			

- ♦ Integrierte Bausteine

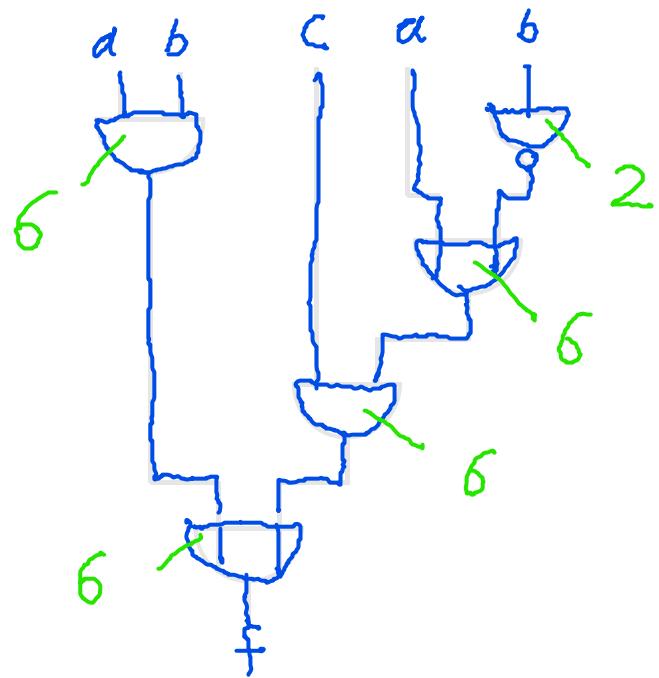


Quellen: Datenbücher von *ON Semiconductor* und *Texas Instruments Inc.*

- ♦ Darstellung boolescher Funktionen mit Schaltsymbolen

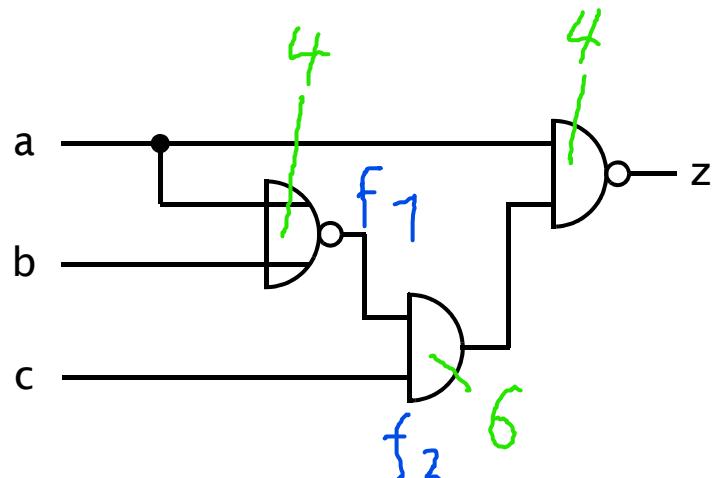
Boolesche Algebra	technische Realisierung
Gleichung	$\Leftrightarrow$ Schaltnetz
Variable	$\Leftrightarrow$ Signal(-leitung)
Operator	$\Leftrightarrow$ Gatter
Funktion	$\Leftrightarrow$ Funktionsblock

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



$$\begin{aligned} CU &\approx 2 + 4 \cdot 6 \\ &= 26 \end{aligned}$$

- Rekonstruktion boolescher Funktionen aus Schaltnetzen



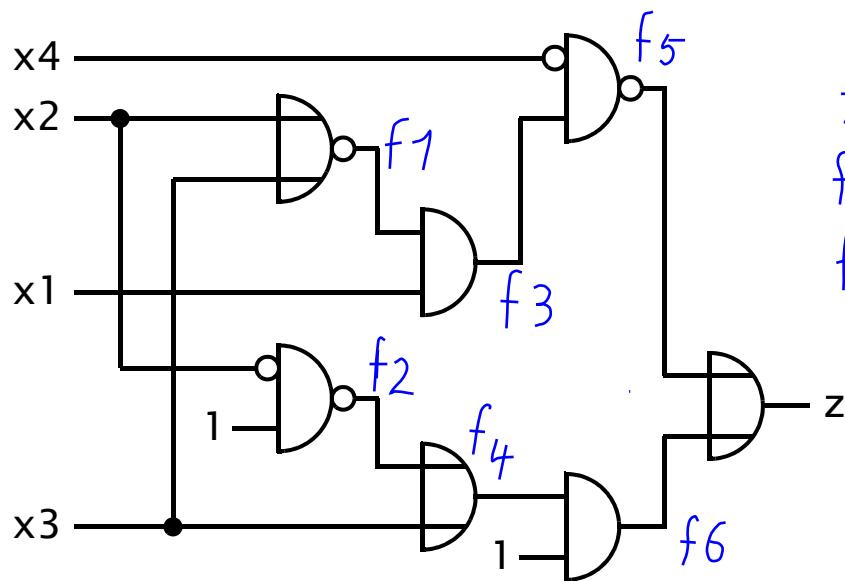
$$CO = 4 + 4 + 6 = 14$$

$$f_1 = (a + b)$$

$$f_2 = (f_1 \cdot c)$$

$$z = (a \cdot f_2) = (a \cdot (a + b) \cdot c)'$$

- Rekonstruktion und Minimierung boolescher Funktionen aus Schaltnetzen



$$\begin{aligned}
 f_1 &= (x_2 + x_3)' \\
 f_2 &= (x_2' \cdot 1)' = (x_2')' = x_2 \\
 f_3 &= (x_1 \cdot f_1) = x_1 \cdot (x_2 + x_3)' \\
 f_4 &= (x_3 + f_2) = x_3 \cdot x_2 \\
 f_5 &= (x_4' \cdot f_3)' = x_4' + f_3' \\
 &= x_4 + f_3' = x_4 + (x_1 \cdot (x_2 + x_3)')' \\
 &= x_4 + x_1' + x_2 + x_3 \\
 f_6 &= 1 \cdot f_4 = x_2 + x_3
 \end{aligned}$$

$$\begin{aligned}
 z &= f_5 + f_6 = x_4 + x_1' + x_2 + x_3 + \cancel{x_2} + \cancel{x_3} \\
 &= x_1' + x_2 + x_3 + x_4
 \end{aligned}$$

- ♦ Realisierung boolescher Funktionen mit NAND-Gattern
1. Umwandlung einer booleschen Funktion mit Hilfe des Axioms A5 in einen disjunktiven Ausdruck:  

$$f(a, b, c) = a \cdot (b + c') \stackrel{[A5]}{=} a \cdot b + a \cdot c'$$
  2. Eliminierung der ODER-Operatoren durch die Anwendung des Doppelte-Negation-Gesetzes G7 und anschließend des ersten Gesetzes von De Morgan G8:  

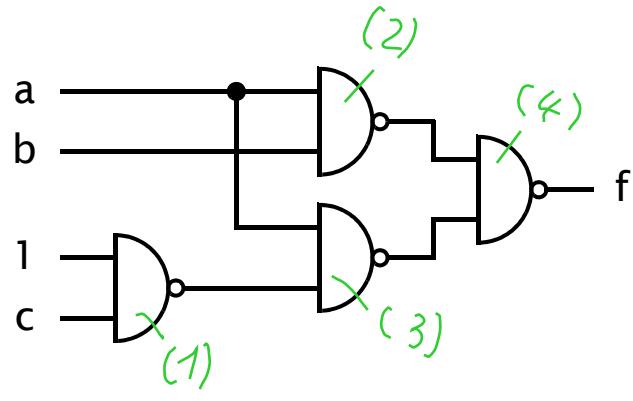
$$f(a, b, c) = a \cdot b + a \cdot c' \stackrel{[G7]}{=} ((a \cdot b) + a \cdot c')' \stackrel{[G8]}{=} ((a \cdot b)' \cdot (a \cdot c')')'$$
  3. Eliminierung negierter Variablen durch die Anwendung des Identitätsaxioms A8:  

$$f(a, b, c) = ((a \cdot b)' \cdot (a \cdot c')')' \stackrel{[A8]}{=} ((a \cdot b)' \cdot (a \cdot (c \cdot 1)'))' \quad (1)$$

$$= ((a \cdot b)' \cdot (a \cdot c))' \quad (2)$$

$$= (a \cdot b)' \cdot (a \cdot c) \quad (3)$$

$$= (a \cdot b)' \cdot c \quad (4)$$



- ◆ Realisierung boolescher Funktionen mit NOR-Gattern

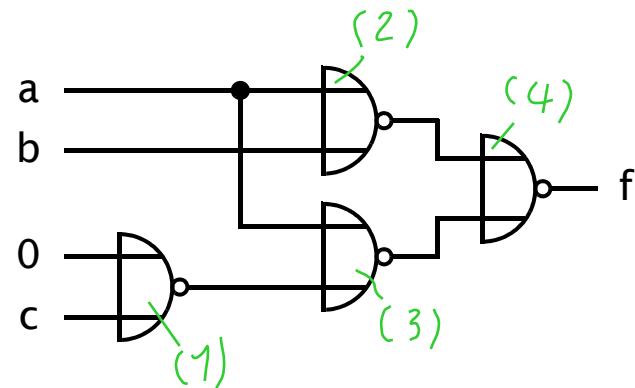
1. Umwandlung einer booleschen Funktion mit Hilfe des Axioms A6 in einen konjunktiven Ausdruck:

$$f(a, b, c) = a + b \cdot c' \stackrel{[A6]}{=} (a + b) + (a + c')$$

2. Eliminierung der UND-Operatoren durch die Anwendung des Doppelte-Negation-Gesetzes G7 und anschließend des zweiten Gesetzes von De Morgan G9:

$$f(a, b, c) = (a + b) \cdot (a + c') \stackrel{[G7]}{=} ((a + b) + (a + c'))' \stackrel{[G9]}{=} ((a + b) \cdot (a + c'))' = [G9]$$

### 3. Eliminierung negierter Variablen durch die Anwendung des Identitätsaxioms A7:



- ♦ Realisierung boolescher Funktionen mit NAND-Gattern
  - Beispiel: gesucht wird eine möglichst minimale Realisierung der booleschen Funktion  $f(a,b,c) = a \cdot (b' + c') + b \cdot c$  mit NAND-Gattern

- Lösung:

$$f(a,b,c) = a \cdot (b' + c') + b \cdot c \quad [G 8]$$

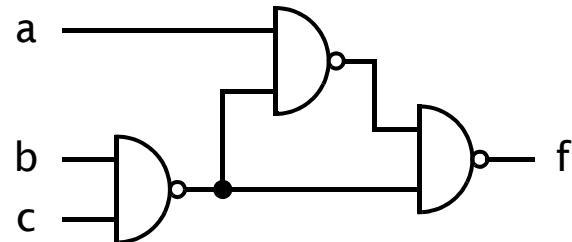
$$= a \cdot (b \cdot c)' + b \cdot c \quad [G 7]$$

$$= (a \cdot (b \cdot c)')' + b \cdot c'' \quad [G 8]$$

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

mit  $t(b,c) = (b \cdot c)'$

$$f(a,b,c) = ( (a \cdot t) \cdot t )'$$



- ◆ Äquivalente Darstellungen
  - verbale Beschreibungen
  - boolesche Funktionen
  - Funktions-/Wahrheitstabellen
  - Schaltpläne
  - Impulspläne
  - KV-Diagramme
- ◆ Normalformen boolescher Funktionen
  - Eine boolesche Funktion kann durch verschiedene aber äquivalente boolesche Ausdrücke beschrieben werden.
  - Solche Beschreibung ist also nicht eindeutig.
  - Eine Standarddarstellung boolescher Funktionen sind die konjunktive (KNF) und die disjunktive Normalform (DNF).

- ◆ Produktterm PT
  - Konjunktion von Variablen in negierter und nicht negierter Form  
 $a \cdot b' \cdot c'$  oder  $b \cdot c$
- ◆ Disjunktive Normalform (DNF)
  - Disjunktion (ODER-Verknüpfung) von Produktterms  
 $f(a, b, c) = \underbrace{a \cdot b' \cdot c'}_{\text{MT}} + \underbrace{a' \cdot b}_{\text{PT}} + \underbrace{b \cdot c}_{\text{PT}}$
- ◆ Minterm MT
  - Produktterm, in dem jede Variable einer booleschen Funktion genau einmal vorkommt (einfach oder negiert)  
 $a \cdot b' \cdot c'$  ist Minterm von  $f(a, b, c)$
- ◆ Kanonische Disjunktive Normalform (KDNF)
  - eindeutige Darstellung einer booleschen Funktion als Disjunktion von Mintermen
  - Jeder Minterm einer KDNF entspricht einer Zeile in der Funktionsstabelle, die den Funktionswert 1 liefert.

- ◆ Summenterm (Konjunktionsterm)

- Disjunktion von Variablen in negierter und nicht negierter Form

$$a + b' + c' \quad d' + b$$

- ◆ Konjunktive Normalform (KNF)

- Konjunktion (UND-Verknüpfung) von Summentermen

$$f(a, b, c) = (a + b' + c') \cdot (d' + b) \cdot (b + c')$$

- ◆ Maxterm

- Summenterm, in dem jede Variable einer booleschen Funktion genau einmal vorkommt (einfach oder negiert)

$$(a + b' + c') \text{ ist Maxterm von } f(a, b, c)$$

- ◆ Kanonische Konjunktive Normalform (KKNF)

- eindeutige Darstellung einer booleschen Funktion als Konjunktion von Maxterminen
  - Jeder Maxterm einer KKNF entspricht einer Zeile in der Funktionsstabelle, die den Funktionswert 0 liefert.

- ♦ Kanonische Disjunktive und Konjunktive Normalformen
  - sind eindeutige Darstellungen, d.h. jede boolesche Funktion hat genau eine KDNF und eine KKNF!

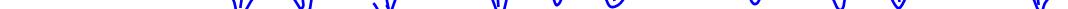
Nr	x	y	z	s	Minterme	Maxterme
0	0	0	0	0		$x + y + z$
1	0	0	1	1	$x' \cdot y' \cdot z$	
2	0	1	0	1	$x' \cdot y \cdot z'$	
3	0	1	1	0		$x + y' + z'$
4	1	0	0	1	$x \cdot y' \cdot z'$	
5	1	0	1	0		$x' + y + z'$
6	1	1	0	0		$x' + y' + z$
7	1	1	1	1	$x \cdot y \cdot z$	

$$\text{KDNF: } s(x, y, z) = x' \cdot y' \cdot z + x' \cdot y \cdot z' + x \cdot y' \cdot z' + x \cdot y \cdot z$$

$$\text{KKNF: } s(x, y, z) = (x + y + z) \cdot (x + y' + z') \cdot (x' + y + z') \cdot (x' + y' + z)$$

- ◆ Kanonische Disjunktive und Konjunktive Normalformen
    - kompakte Darstellung in der binären oder dezimalen Codierung  
(Hinweis: auf die Reihenfolge der Variablen in Termen achten)

$$f(a, b, c) = a \cdot b' \cdot c + a \cdot b \cdot c' + a' \cdot b \cdot c' + a' \cdot b' \cdot c$$



$$f(a, b, c) = \Sigma((101)_2, (110)_2, (010)_2, (001)_2)$$

$$f(a, b, c) = \sum(5, 6, 2, 1) \quad \text{hier gilt } f(a, b, c) = 1$$

$$g(a, b, c) = (a' + b' + c) \cdot (a + b' + c') \cdot (a + b + c')$$

$$(1 \quad 1 \quad 0) \quad (0 \quad 1 \quad 1) \quad (0 \quad 0 \quad 1)$$

$$g(a, b, c) = \Pi((\underline{1}0)_2, (\underline{0}1)_2, (001)_2)$$

$$g(a, b, c) = \Pi(6, 3, 1) \quad \text{hier gilt } g(a, b, c) = 0!$$

- Umwandlung von DNF in KDNF

gesucht wird die KDNF für die Funktion  $f(x, y, z) = x' \cdot (y + z)'$

- Schritt: negierte Ausdrücke beseitigen (G8, G9)  
alle Klammer beseitigen (A5)

$$f = x' \cdot (y + z)' = [G9] = x' \cdot (y' + z') = x' \cdot (y' + z) = [A5]$$

$$= x' \cdot y' + x' \cdot z$$

- Schritt: Produktterme auf Minterme expandieren (A8, A9, A5)

$$f = x' \cdot y' + x' \cdot z = [A8] = x' \cdot y' \cdot 1 + x' \cdot 1 \cdot z = [2 \cdot A9]$$

$$= x' \cdot y' \cdot (z + z') + x' \cdot (y' + y') \cdot z = [2 \cdot A5]$$

$$= \underline{x' \cdot y' \cdot z} + \underline{x' \cdot y' \cdot z'} + \underline{x' \cdot y \cdot z} + \cancel{x' \cdot y' \cdot z}$$

- Schritt: mehrfach vorkommende Minterme streichen (G1)

$$f = x' \cdot y' \cdot z + x' \cdot y' \cdot z' + x' \cdot y \cdot z + \cancel{x' \cdot y' \cdot z}$$

$$= (001) + (000) + (011) = \sum_{\text{0}}^{\text{3}} ((000)_2, (001)_2 (011)_2)$$

$$f(x, y, z) = \Sigma(0, 1, 3)$$

- Umwandlung von KNF in KKNF

gesucht wird die KKNF für die Funktion  $f(x, y, z) = (x + y)' + z$

- Schritt: negierte Ausdrücke beseitigen (G8, G9), Klammer einführen (A6)

$$f = (x + y)' + z = [G_8] = x' \cdot y' + z = [A_6] = (x' + z) \cdot (y' + z)$$

- Schritt: Summenterme auf Maxterme expandieren (A7, A10, A6)

$$\begin{aligned} f &= (x' + z) \cdot (y' + z) = [A_7] \approx (x' + 0 + z) \cdot (0 + y' + z) = [2 \times A_{10}] \\ &= (x' + \cancel{y \cdot y'} + z) \cdot (\cancel{x \cdot x'} + y' + z) = [2 \times A_6] \\ &= (x' + y + z) \cdot (\underline{x' + y'} + z) \cdot (\cancel{x + y'} + z) \cdot (\cancel{x' + y'} + z) \end{aligned}$$

- Schritt: mehrfach vorkommende Maxterme streichen (G2)

$$\begin{aligned} f &= (x' + y + z) \cdot (x' + y' + z) \cdot (x + y' + z) \cdot \cancel{(x' + y' + z)} \\ KKNF &= (x' + y + z) \cdot \cancel{(x' + y' + z)} \cdot (x + y' + z) \\ &\approx (100)_2 \cdot (110)_2 \cdot (010)_2 = \Pi(4, 6, 2) \end{aligned}$$

$$f(x, y, z) = \Pi(4, 6, 2)$$

Beispiel: gesucht wird die KKNF

$$f(a, b, c) = a \oplus (b + c)'$$

[Sub e :=  $(b + c)'$ ]

$$= a \oplus e = a \cdot e + a \cdot e'$$

[Sub x :=  $a \cdot e'$ ]

$$= a' \cdot e + x$$

[A6)

$$= (a' + x) \cdot (e + x)$$

[Sub  $x^{-1}$ ]

$$= (a' + a \cdot e') \cdot (e + a \cdot e')$$

[z x A6]

$$= (a' + a) \cdot (a' + e') \cdot (e + a) \cdot (e + e')$$

[z x A8, z x A9]

$$= (a' + e') \cdot (e + a)$$

[Sub  $e^{-1}$ ]

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

[G7, G8]

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

[A6]

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

[2 x A7, 2 x A10]

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

[z x A6]

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

~~(a + b' + c')~~ [G2]

$$= \pi(4, 2, 3, 1)$$

Beispiel: gesucht wird die KDNF  $\sum(0, 5, 6, 7)$

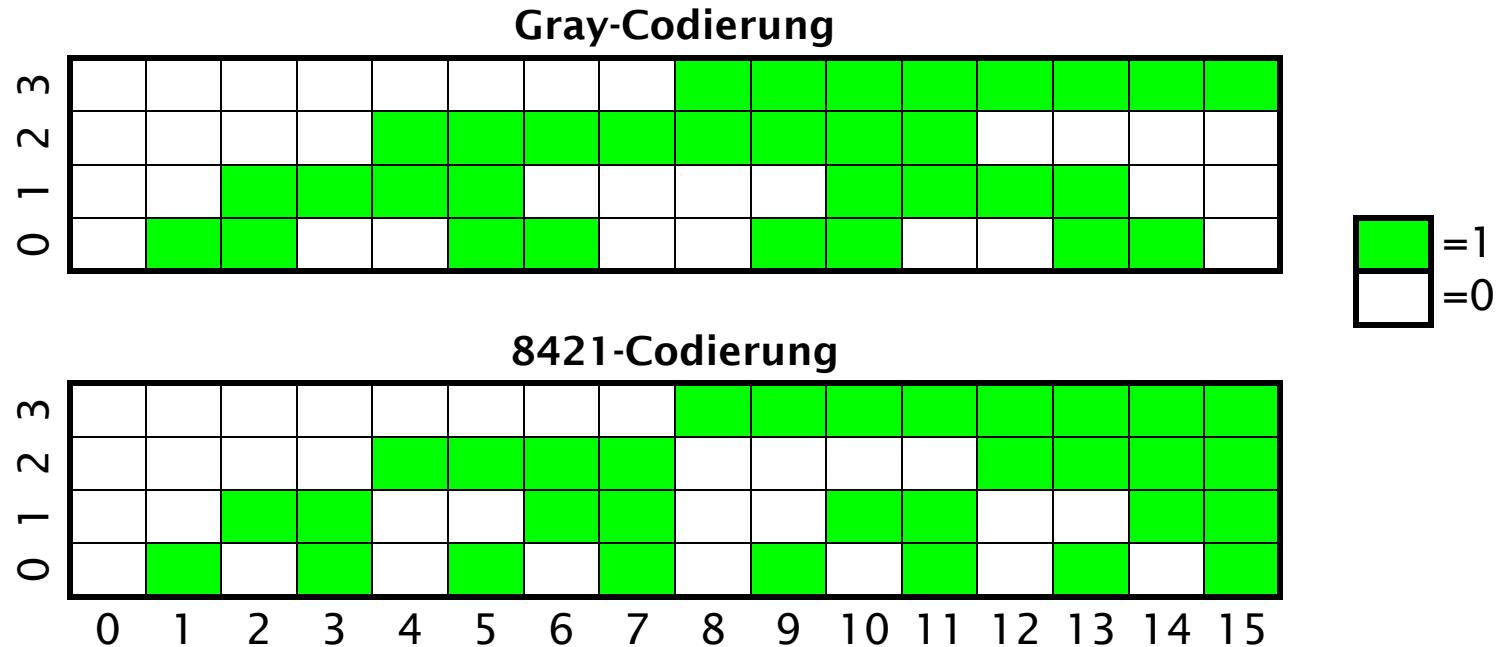
$$\begin{aligned}
 f(a, b, c) &= a \oplus (b + c)' && [\text{Sub } e := b + c] \\
 &= a \oplus e' = a' \cdot c' + a \cdot c'' && [G7, \text{Sub } e'] \\
 &= a' \cdot (b + c)' + a \cdot (b + c) && [G8] \\
 &\quad a \cdot (b + c) && [A5] \\
 &= a' \cdot b' \cdot c' + a \cdot b + a \cdot c && [2 \times A8, 2 \times A10] \\
 &= a' \cdot b' \cdot c' + a \cdot b \cdot (c + c') + a \cdot (b + b') + c && [2 \times A5] \\
 &= a' \cdot b' \cdot c' + \underline{a \cdot b \cdot c} + \underline{a \cdot b \cdot c'} + \cancel{a \cdot b \cdot c} + . && [G1] \\
 &= a' \cdot b' \cdot c' + a \cdot b \cdot c + a \cdot b \cdot c' + a \cdot b' \cdot c \\
 &= a' \cdot b' \cdot c' + a \cdot b \cdot c \\
 &= \sum(0, 7, 6, 5)
 \end{aligned}$$

- ◆ Jede boolesche Funktion kann in *kanonischer disjunktiver* und *konjunktiver Normalform* dargestellt werden.
- ◆ Beide Darstellungen sind äquivalent und ineinander überführbar. (KDNF  $\leftrightarrow$  KKNF)
- ◆ In der Praxis ist die (K)DNF übersichtlicher als die (K)KNF und hinsichtlich der Vereinfachung von booleschen Ausdrücken leichter zu handhaben.
- ◆ Anwendungen von KDNF und KKNF:
  - Schaltnetzsynthese und -analyse
  - Schaltungen auf Basis von LUT (Look-Up-Table)
  - Zweistufige Schaltnetze mit PLAs, PALs
  - Schaltnetze mit Multiplexern

- ◆ Anwendung der booleschen Algebra:
  - Expansion von Normalformen auf Kanonische Formen  
 $DNF/KNF \rightarrow KDNF/KKNF$
  - Umwandlung von (K)DNF in (K)KNF und umgekehrt
  - Umsetzung von DNF/KNF auf Realisierung mit NANDs/NORs
  - in der Schaltungsanalyse bei der Rekonstruktion einer Funktion aus einer Schaltung
  - algebraische Minimierung boolescher Funktionen
- ◆ Minimierung boolescher Funktionen:
  - algorithmisch schwer beschreibbar
  - heuristik-basierende Vereinfachung
  - manchmal suboptimale Ergebnisse
  - große Erfahrung erforderlich

- ◆ Die KV-Methode ist
  - eine nach Karnaugh und Veitch benannte Methode zur Minimierung boolescher Funktionen
  - systematisch, graphisch-orientiert, anschaulich und relativ einfach in der Anwendung
- ◆ Die KV-Methode basiert auf
  - einem (KV-)Diagramm: eine 2-dimensionale Matrix mit „geeigneter“ Indizierung der Felder zur Abbildung einer Funktionstabelle
  - einem Regelwerk zur Findung einer minimalen Lösung
- ◆ Bei einer „von-Hand“-Anwendung ist die KV-Methode für
  - einige wenige boolesche Variablen ( $\leq 4$ ) übersichtlich,
  - mehr als 6 boolesche Variablen kaum geeignet,
  - 5 oder 6 Variablen noch überschaubar und bedingt geeignet.

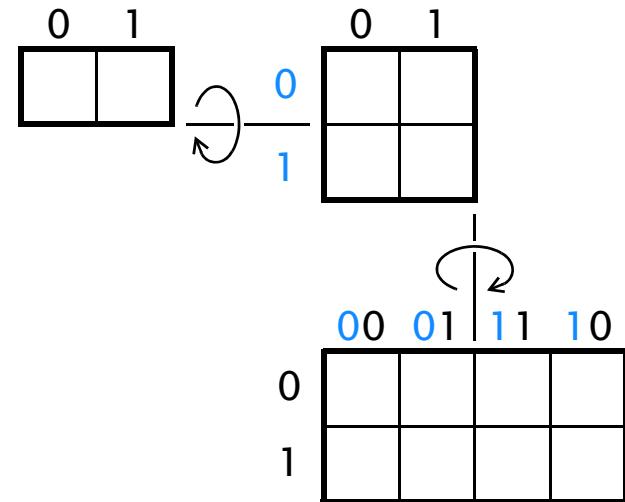
- ◆ Einschrittige vs. mehrschrittige Codierung
  - Benachbarte Elemente einer Menge werden durch Binärmuster repräsentiert, die sich in genau einer Stelle unterscheiden.
  - Die Eigenschaft „benachbart“ ist eine Relation, die auf den Elementen der Menge definiert ist.
    - räumliche Nachbarschaft in Abtastsystemen
    - zwei Knoten eines Zustandagraphen sind mit einer Kante verbunden



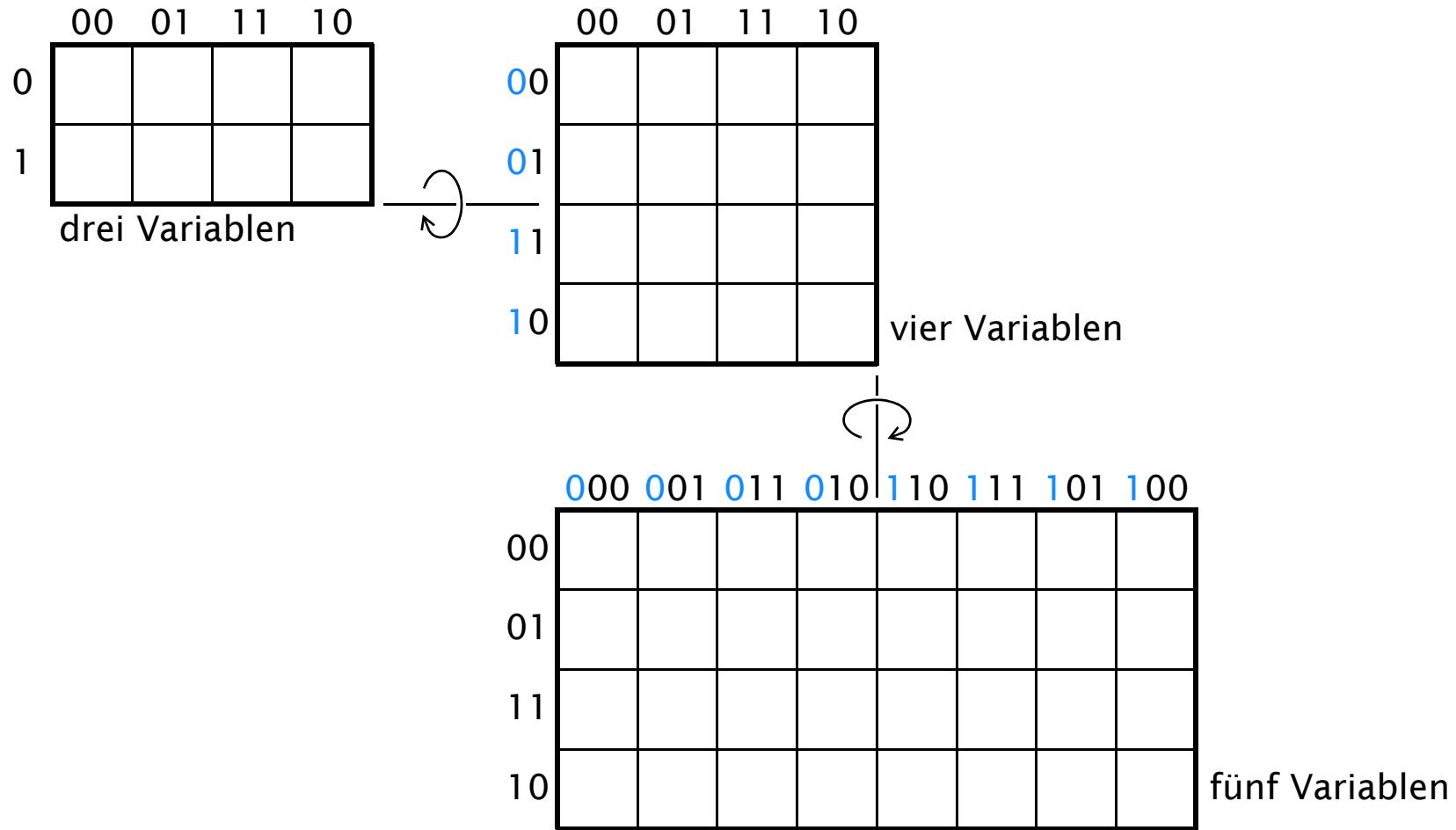
- ◆ Konstruktion eines KV-Diagramms

Indizierung der Felder (Spalten und Zeilen) mittels einer einschrittigen, zyklischen Codierung (z.B. Gray-Code)

1. Man beginnt mit einer Eingangsvariable. Das entsprechende KV-Diagramm besteht nun aus zwei Feldern, weil sie nur zwei Werte annehmen kann.
2. Durch Spiegelung des KV-Diagramms um die horizontale Randkante entsteht ein um eine zweite Eingangsvariable erweitertes KV-Diagramm.
3. Durch erneute Spiegelung des KV-Diagramms um die vertikale Randkante wird das ursprüngliche KV-Diagramm um eine dritte Eingangsvariable erweitert.  
Für jede zusätzliche Eingangsvariable verdoppelt sind die Anzahl der Felder im KV-Diagramm:  $2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 32$  usw..



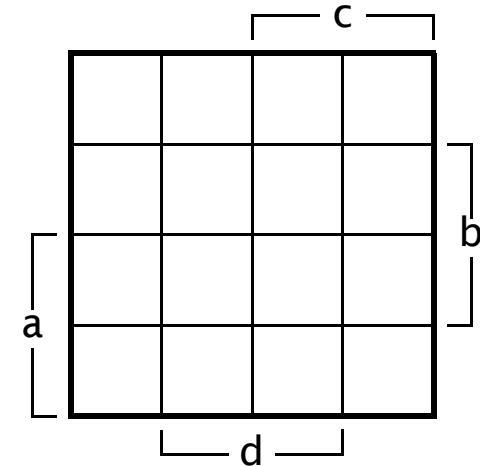
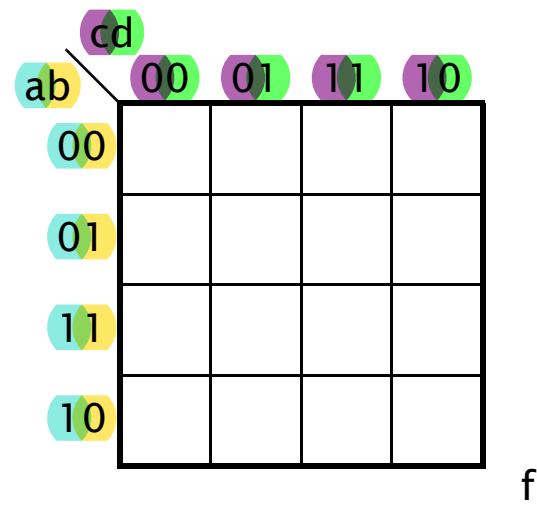
- ◆ Konstruktion eines KV-Diagramms für 3, 4 und 5 Eingangsvariablen



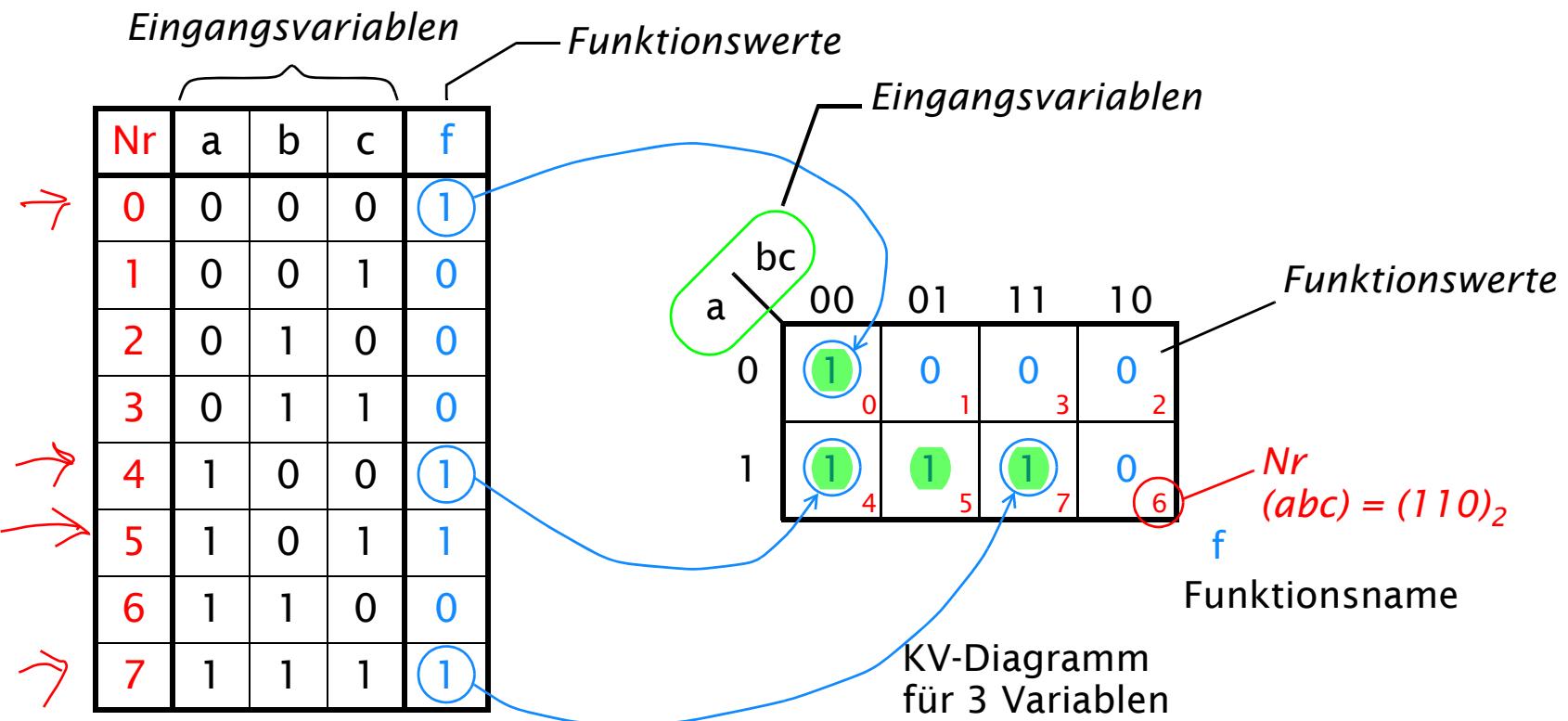
# KV-Methode

- ◆ äquivalente Darstellungen von KV-Diagrammen für 4 Variablen

	$f$	$c'd'$	$c'd$	$cd$	$cd'$
$a'b'$					
$a'b$					
$ab$					
$ab'$					



- ◆ Abbildung einer Funktionstabelle auf ein KV-Diagramm
  $f(a, b, c) = \Sigma(0, 4, 5, 7) = \Sigma(000_2, 100_2, 101_2, 111_2)$ 
  - in jedes Feld des KV-Diagramms wird der zu den Eingangsvariablen gehörige Funktionswert eingetragen.



- ♦ Regelwerk zur Findung einer minimalen DNF mit KV-Diagrammen
  1. Es dürfen nur „benachbarte“ Felder, die mit '1' besetzt sind, zu einer Gruppe zusammengefaßt werden.
  2. Es können nur rechteckige Gruppen aus 2, 4, 8, ... , allgemein aus  $2^N$  Feldern, zusammengefaßt werden.
  3. Es sollen immer die größtmöglichen Gruppen gebildet werden.
  4. Alle Felder müssen durch mindestens eine Gruppe erfaßt sein. Einzelne Felder dürfen in mehreren Gruppen enthalten sein. Es können Gruppen aus nur einem Feld vorkommen.
  5. Für jede Gruppe wird ein UND-verknüpfter Term aufgestellt, in dem nur diejenigen Variablen enthalten sind, die allen Feldern der Gruppe gemeinsam sind.
  6. Die Ausgangsfunktion wird durch die ODER-Verknüpfung aller Gruppen zu einer disjunktiven Normalform gebildet.

- ◆ Gruppieren von „benachbarten“ Feldern erfolgt auf der Grundlage der Axiome A5, A9 und A8

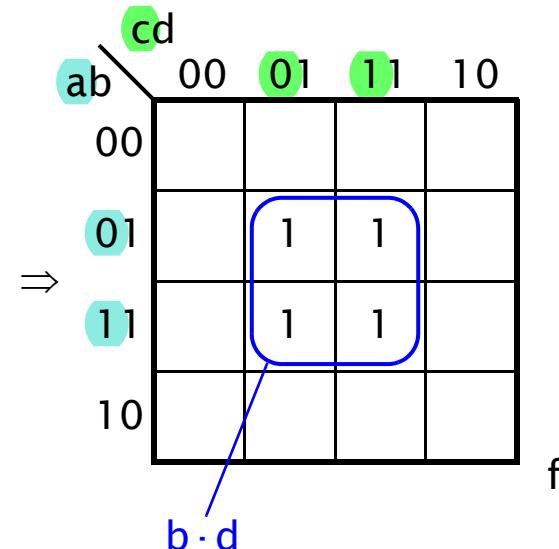
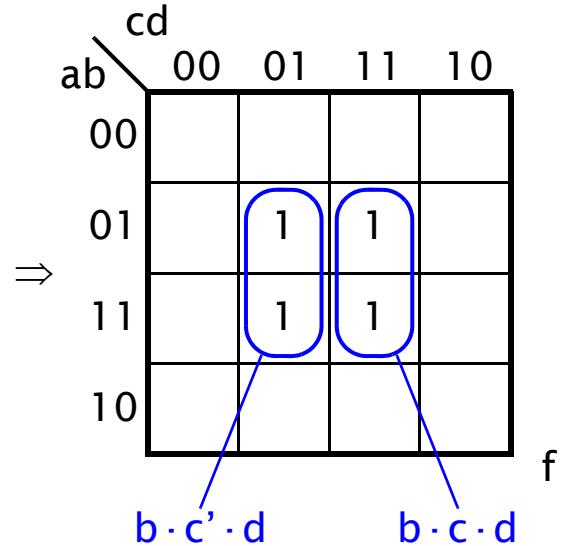
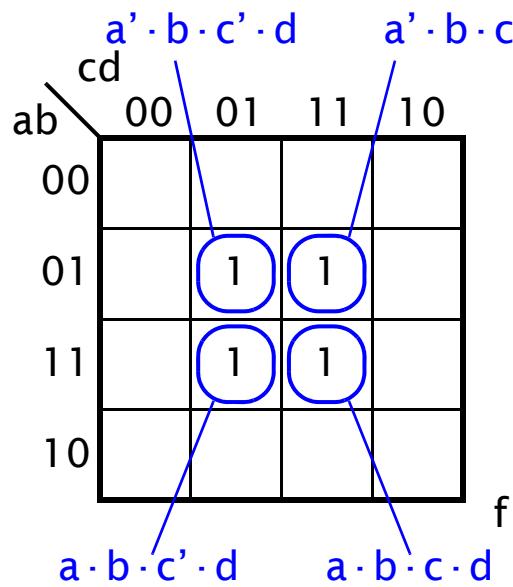
$$\begin{aligned}
 f(a, b, c) &= a \cdot b' \cdot c' + a \cdot b' \cdot c + a \cdot b \cdot c + a \cdot b \cdot c' \\
 &= (a \cdot b' \cdot c' + a \cdot b' \cdot c) + (a \cdot b \cdot c + a \cdot b \cdot c') \\
 &= (a \cdot b' \cdot (c' + c)) + (a \cdot b \cdot (c + c')) \\
 &= a \cdot b' \cdot 1 + a \cdot b \cdot 1 = a + b' \cdot a + b \\
 &= a \cdot (b' + b) = a \cdot 1 = a
 \end{aligned}$$

a	bc	00	01	11	10	f
0		0	0	0	0	
1		1	1	1	1	
	$a \cdot b' \cdot c'$	$a \cdot b' \cdot c$	$a \cdot b \cdot c$	$a \cdot b \cdot c'$		

a	bc	00	01	11	10	f
0		0	0	0	0	
1		1	1	1	1	
	$a \cdot b'$		$a \cdot b$			

a	bc	00	01	11	10	f
0		0	0	0	0	
1		1	1	1	1	
	$a$					

- ◆ Gruppieren von innenliegenden Feldern



$$f(a, b, c, d) = a' \cdot b \cdot c' \cdot d + a \cdot b \cdot c' \cdot d + a' \cdot b \cdot c \cdot d + a \cdot b \cdot c \cdot d \quad [A5]$$

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

$$= b \cdot c' \cdot d + b \cdot c \cdot d$$

$$= b \cdot d \cdot (c' + c)$$

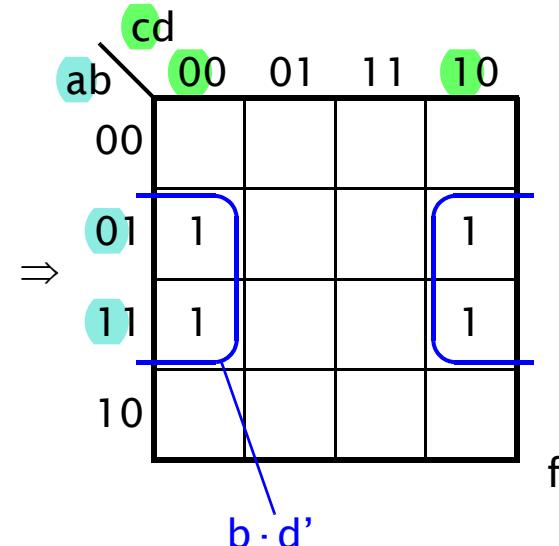
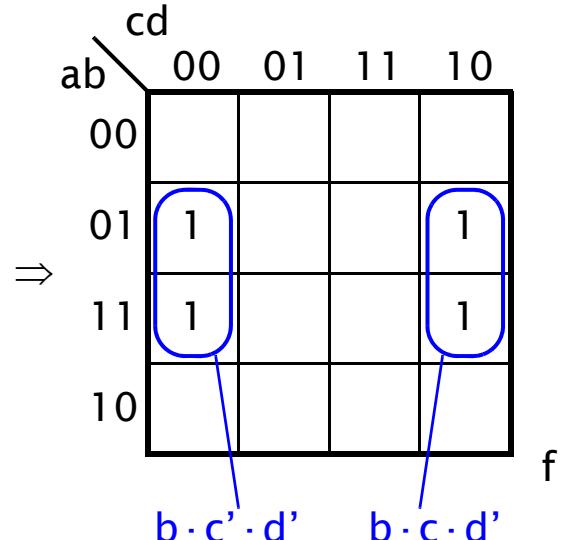
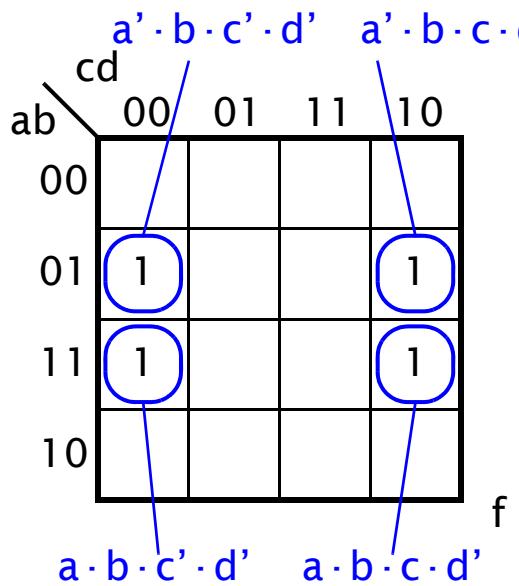
$$= b \cdot d$$

[A9, A8]

[A5]

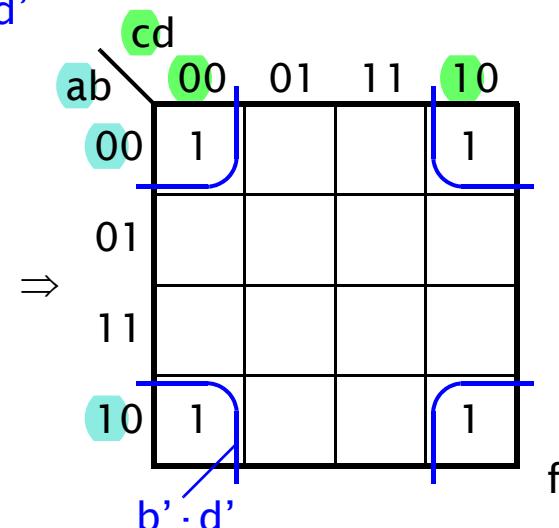
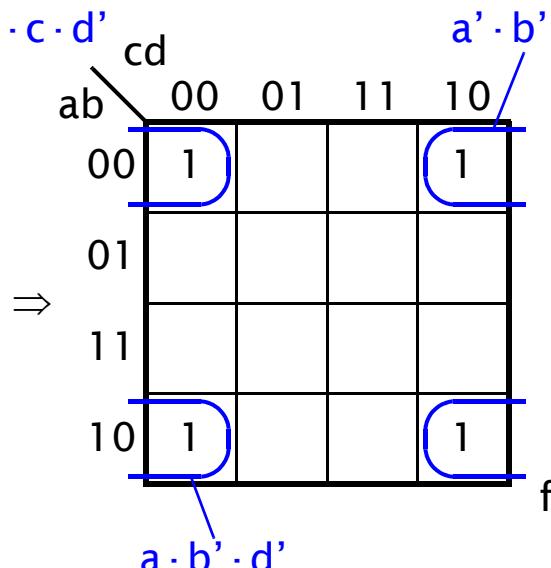
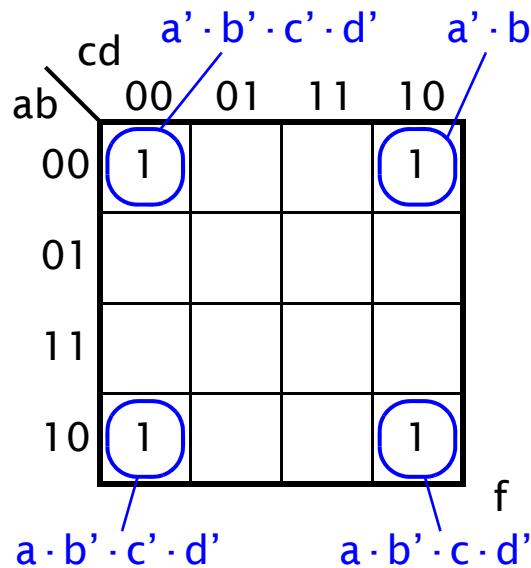
[A9, A8]

- ◆ Gruppieren von Feldern mit Randlage



$$\begin{aligned}
 f(a, b, c, d) &= (a \cdot b \cdot c' \cdot d' + a' \cdot b \cdot c' \cdot d') + (a \cdot b \cdot c \cdot d' + a' \cdot b \cdot c \cdot d') [A5] \\
 &= b \cdot c' \cdot d' \cdot (a' + a) + b \cdot c \cdot d' \cdot (a' + a) [A8, A9] \\
 &= b \cdot c' \cdot d' + b \cdot c \cdot d' [A5] \\
 &= b \cdot d' \cdot (c' + c) [A8, A9] \\
 &= b \cdot d'
 \end{aligned}$$

- ◆ Gruppieren von Feldern in Ecken



$$f(a, b, c, d) = (a \cdot b' \cdot c' \cdot d' + a \cdot b' \cdot c \cdot d') + (a' \cdot b' \cdot c' \cdot d' + a' \cdot b' \cdot c \cdot d') \quad [A5]$$

$$= a \cdot b' \cdot d' \cdot (c' + c) + a' \cdot b' \cdot d' \cdot (c' + c) \quad [A8, A9]$$

$$= a \cdot b' \cdot d' + d' \cdot b' \cdot d'$$

$$= b' \cdot d' \cdot (a + a')$$

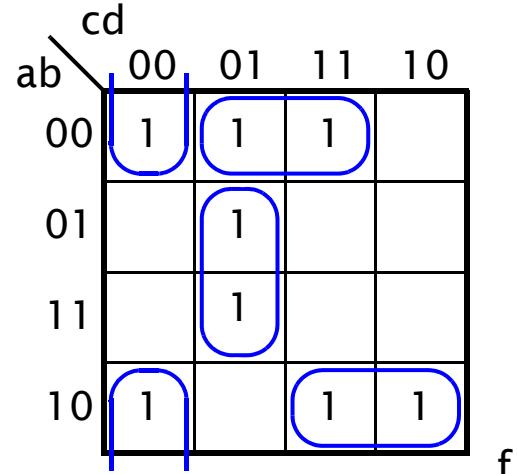
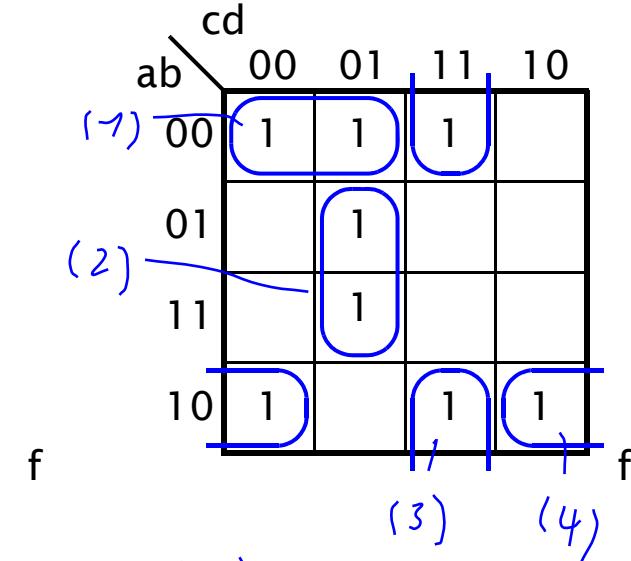
$$= b' \cdot d'$$

[A5]

[A8, A9]

- ◆ Gruppenbildung mit alternativen Lösungen

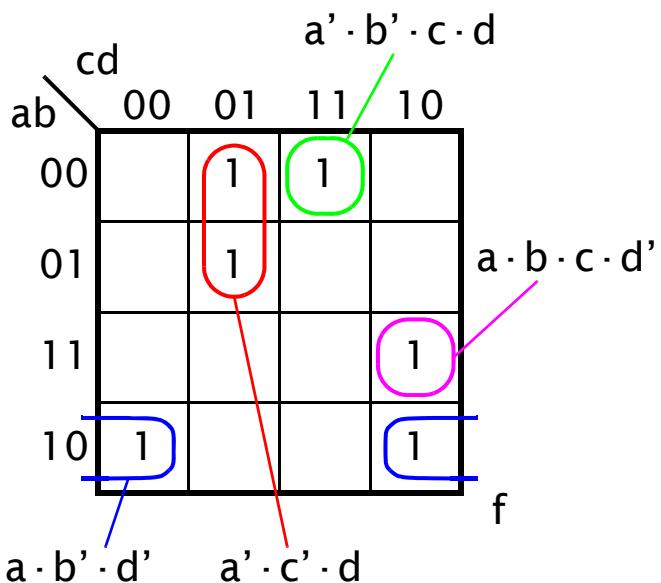
ab \ cd	00	01	11	10
00	1	1	1	0
01	0	1	0	0
11	0	1	0	0
10	1	0	1	1



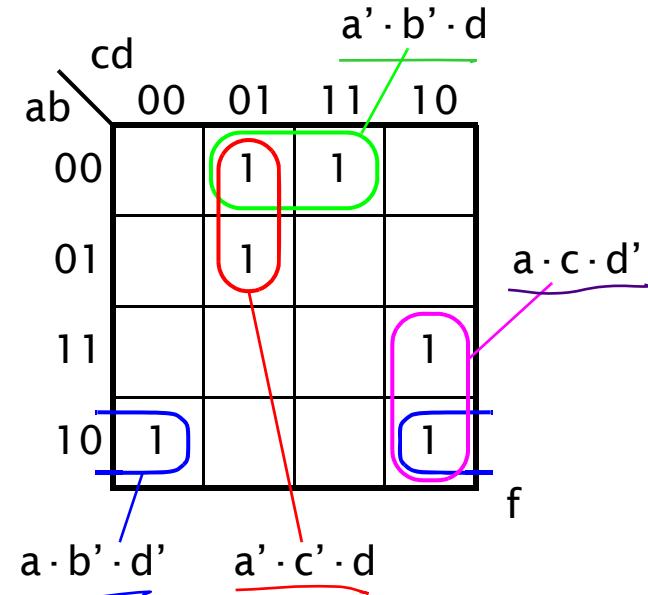
$$f(a, b, c, d) = b \cdot c' \cdot d + a' \cdot b' \cdot c' + b' \cdot c \cdot d + a \cdot b' \cdot d'$$

$$f(a, b, c, d) = b \cdot c' \cdot d + a' \cdot b' \cdot d + b' \cdot c' \cdot d' + a \cdot b' \cdot c$$

♦ Verschmelzung einzelner Felder mit größeren Gruppen

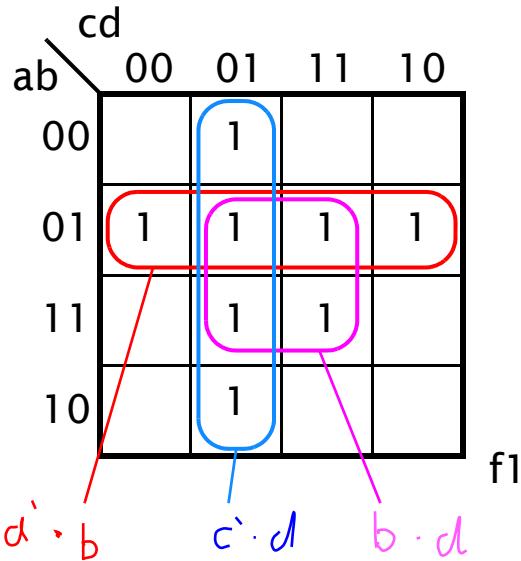


⇒



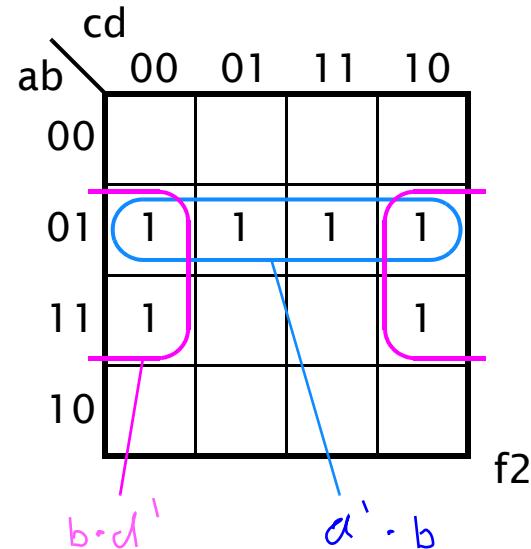
$$\begin{aligned}
 f(a, b, c, d) &= (a \cdot b' \cdot d' + a \cdot b \cdot c \cdot d') + (a' \cdot c' \cdot d + a' \cdot b' \cdot c \cdot d) \quad [A5] \\
 &= (a \cdot d' \cdot (b' + b \cdot c)) + (a' \cdot d \cdot (c' + b \cdot c)) \quad [A6] \\
 &= (a \cdot d' \cdot (b' + b) \cdot (b' + c)) + (a' \cdot d \cdot (c' + c) - (c' + b)) \quad [A8, A9] \\
 &= (d \cdot d' \cdot (b' + c)) + (a' \cdot d \cdot (c' + b')) \quad [A5] \\
 &= \underline{a \cdot b' \cdot d'} + \underline{a \cdot c \cdot d'} + \underline{a' \cdot c' \cdot d} + \underline{a' \cdot b' \cdot d}
 \end{aligned}$$

- ◆ Bildung überlappender Gruppen



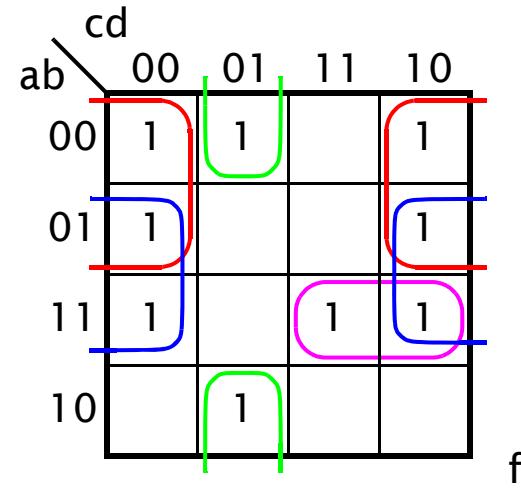
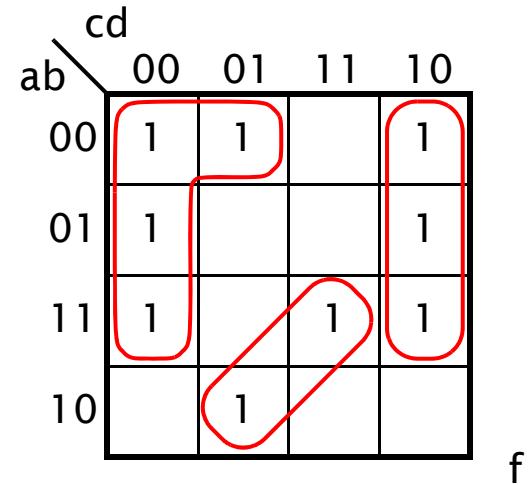
$$f1(a, b, c, d) = d' \cdot b + c' \cdot d + b \cdot d$$

$$f2(a, b, c, d) = d' \cdot b + b \cdot d' = b \cdot (d' + d)$$



# KV-Methode

- ◆ Beispiele mit KV-Diagrammen:
  - Beispiele mit **unzulässigen** Gruppen:
    - geknickte Gruppe
    - diagonale Gruppe
    - Gruppe mit ungerader Anzahl von Feldern
  - korrekte Zusammenfassung der einzelnen Felder:
    - zwei 2er-Gruppen  $b' \cdot c' \cdot d$ ,  $a \cdot b \cdot c$
    - zwei 4er-Gruppen  $a' \cdot d'$ ,  $b \cdot d'$



# KV-Methode

## ♦ Beispiel

die Funktion  $f(a, b, c) = a + b' \cdot c + a' \cdot b \cdot c$  ist mit der KV-Methode zu minimieren.

1. Schritt: die Funktion  $f$  wird mit den Axiomen und Gesetzen der booleschen Algebra in eine KDNF umgewandelt:

$$\begin{aligned}
 f(a, b, c) &= a + b' \cdot c + a' \cdot b \cdot c && [3 \times A8] \\
 &= a \cdot 1 \cdot 1 + 1 \cdot b' \cdot c + a' \cdot b \cdot c && [3 \times A9] \\
 &= a \cdot (b+b') \cdot (c+c') + (a+a') \cdot b' \cdot c + a' \cdot b \cdot c && [n \times A5] \\
 &= a \cdot b \cdot c + a \cdot b' \cdot c' + \underline{\underline{a \cdot b' \cdot c}} + \underline{\underline{a \cdot b' \cdot c'}} + \underline{\underline{a' \cdot b \cdot c}} + a' \cdot b \cdot c + a' \cdot b \cdot c
 \end{aligned}$$

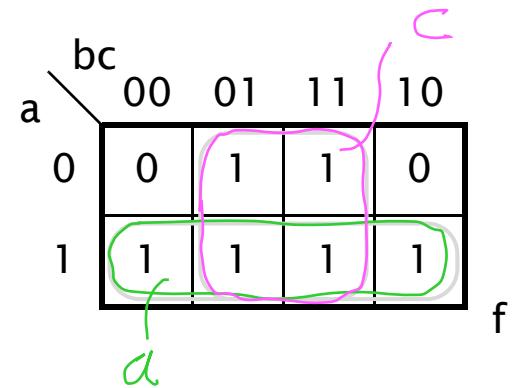
2. Schritt: die KDNF der Funktion  $f$  wird binär oder dezimal dargestellt:

$$\begin{aligned}
 \text{KDNF}(f) &= \sum \left( (111)_2, (110)_2, (101)_2, (100)_2, (011)_2, (010)_2 \right) \\
 \text{KDNF}(f) &= \sum \left( 7, 6, 5, 4, 1, 3 \right)
 \end{aligned}$$

3. Schritt: die KDNF der Funktion f wird in das KV-Diagramm eingetragen.

	bc	00	01	11	10	
a	0	0 <sub>0</sub>	1 <sub>1</sub>	1 <sub>3</sub>	0 <sub>2</sub>	f
	1	1 <sub>4</sub>	1 <sub>5</sub>	1 <sub>7</sub>	1 <sub>6</sub>	

4. Schritt: im KV-Diagramm werden benachbarte Felder mit '1' zu größtmöglichen Gruppen zusammengefaßt.



5. Schritt: aus dem KV-Diagramm werden Gruppen nacheinander ausgelesen und disjunktiv verknüpft.  
 $f(a, b, c) = a + c$

- ◆ KV-Methode mit unvollständig definierten Funktionen
  - In der Praxis kommen häufig Anwendungsfälle vor, in denen gewisse Kombinationen von Eingangssignalen nie auftreten, oder der Funktionswert bestimmter Minterme nicht relevant ist.
  - Solche Fälle werden als „don't care“-Terme (unbestimmte Terme) bezeichnet und können (müssen aber nicht) zur Minimierung boolescher Funktionen herangezogen werden, was i.d.R. zu einer einfacheren Formel führt.
  - Funktionswerte für die „don't care“-Terme werden in Funktionstabellen und in KV-Diagrammen mit 'x' oder '-' gekennzeichnet, und können nach Belieben entweder als 0 oder als 1 bei der Bildung von Gruppen interpretiert werden.

- Funktionstabelle für einen 8421-Gray-Code-Wandler

Nr	8421-Code				Gray-Code			
	a	b	c	d	s	t	u	v
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0

Nr	8421-Code				Gray-Code			
	a	b	c	d	s	t	u	v
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	-	-	-	-
11	1	0	1	1	-	-	-	-
12	1	1	0	0	-	-	-	-
13	1	1	0	1	-	-	-	-
14	1	1	1	0	-	-	-	-
15	1	1	1	1	-	-	-	-

$$s(a, b, c, d) = \Sigma(8, 9, (10, 11, 12, 13, 14, 15))$$

$$t(a, b, c, d) = \Sigma(4, 5, 6, 7, 8, 9, (10, 11, 12, 13, 14, 15))$$

$$u(a, b, c, d) = \Sigma(2, 3, 4, 5, (10, 11, 12, 13, 14, 15))$$

$$v(a, b, c, d) = \Sigma(1, 2, 5, 6, 9, (10, 11, 12, 13, 14, 15))$$

>> In der ersten Liste in Klammern stehen Minterme, in der zweiten „don't-care“-Terme

- ◆ KV-Diagramme für den 8421-Gray-Code-Wandler

	cd	00	01	11	10
ab	00	0	0	0	0
	01	0	1	3	2
	11	-	-	7 <sub>2</sub>	7 <sub>3</sub>
	10	7 <sub>8</sub>	1	-	7 <sub>10</sub>

s

	cd	00	01	11	10
ab	00	0	0	1	1
	01	1	1	0	0
	11	-	-	-	-
	10	0	0	-	-

u

	cd	00	01	11	10
ab	00	0	0	0	0
	01	1	1	1	1
	11	-	-	-	-
	10	1	1	-	-

t

	cd	00	01	11	10
ab	00	0	1	0	1
	01	0	1	0	1
	11	-	-	-	-
	10	0	1	-	-

v

# KV-Methode

- minimierte Gleichungen für den 8421-Gray-Code-Wandler

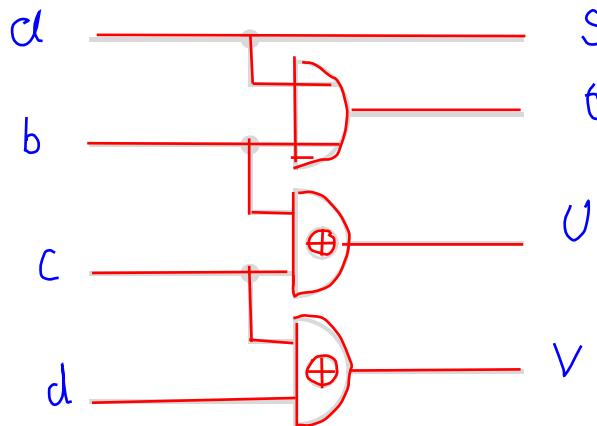
$$s(a, b, c, d) = \underline{d}$$

$$t(a, b, c, d) = \underline{d} + b$$

$$u(a, b, c, d) = \underline{b} \cdot \underline{c}' + \underline{b}' \cdot c = b \oplus c$$

$$v(a, b, c, d) = \underline{c} \cdot \underline{d}' + \underline{c}' \cdot d = c \oplus d$$

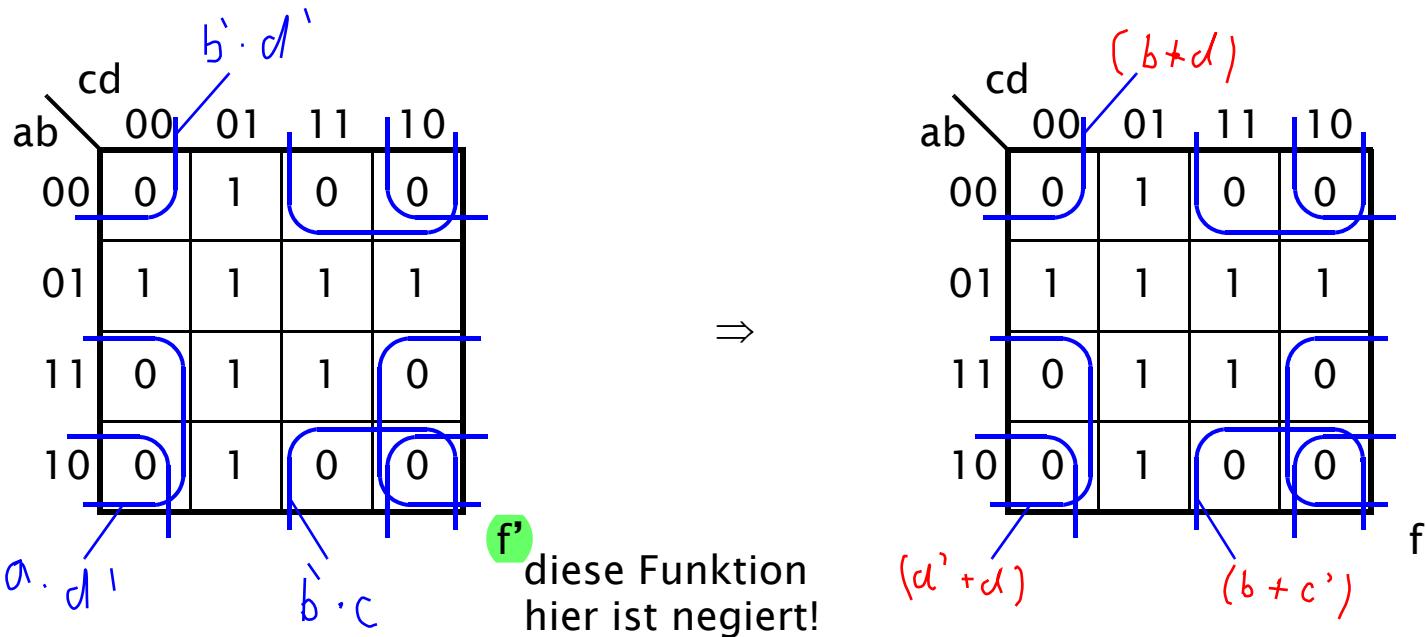
- Schaltplan des 8421-Gray-Code-Wandlers



- ◆ Regelwerk zur Findung einer minimalen KNF mit KV-Diagrammen
  1. Es dürfen nur „benachbarte“ Felder, die mit '0' besetzt sind, zu einer Gruppe zusammengefaßt werden.
  2. Es können nur rechteckige Gruppen aus 2, 4, 8, ... , allgemein aus  $2^N$  Feldern, zusammengefaßt werden.
  3. Es sollen immer die größtmöglichen Gruppen gebildet werden.
  4. Alle Felder müssen durch mindestens eine Gruppe erfaßt sein. Einzelne Felder dürfen in mehreren Gruppen enthalten sein. Es können Gruppen aus nur einem Feld vorkommen.
  5. Für jede Gruppe wird ein ODER-verknüpfter Term aufgestellt, in dem nur diejenigen Variablen enthalten sind, die allen Felder der Gruppe gemeinsam sind.
  6. Die Ausgangsfunktion wird durch die UND-Verknüpfung aller Gruppen zu einer konjunktiven Normalform gebildet.

# KV-Methode

- Findung einer minimalen KNF mit KV-Diagrammen



$$f(a, b, c, d)' = b' \cdot d' + a \cdot d' + b' \cdot c$$

beide Seiten negieren und mit den Gesetzen von de Morgan umwandeln

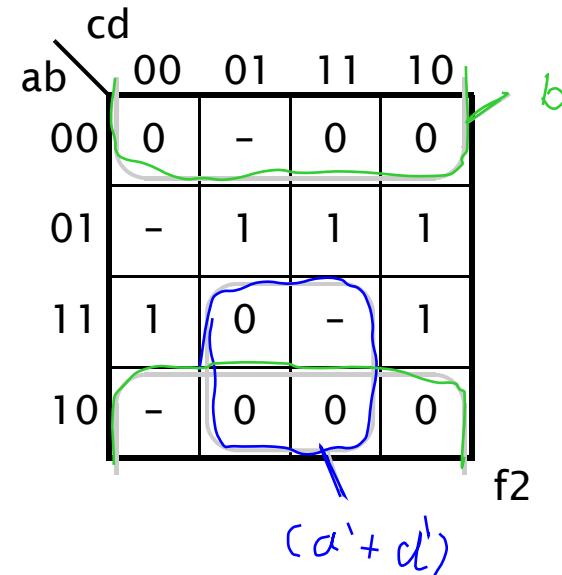
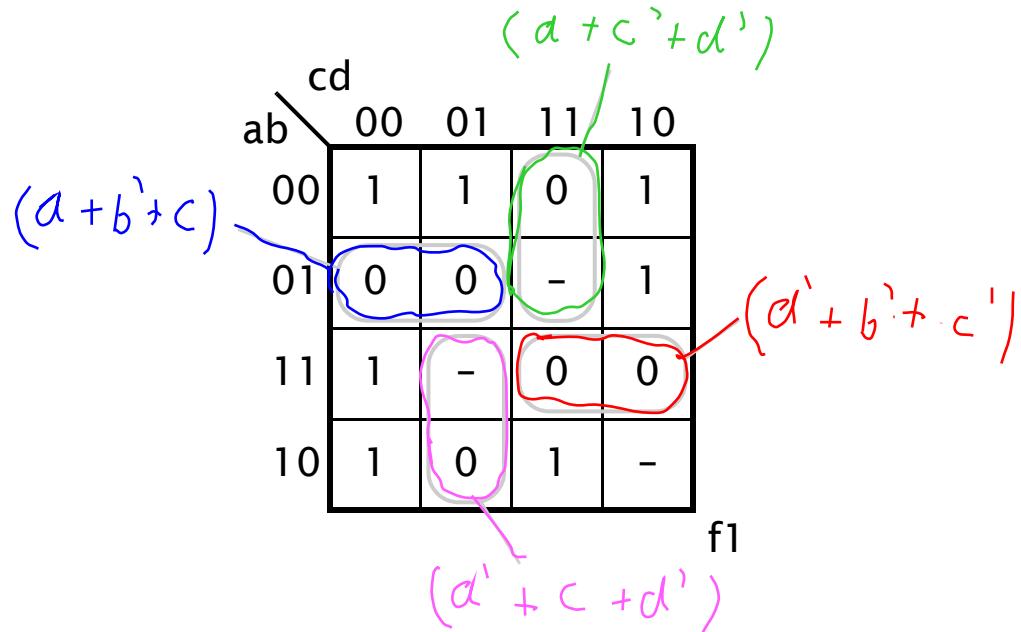
$$f(a, b, c, d)'' = (b' \cdot d' + a \cdot d' + b' \cdot c)' = (b' \cdot d')' \cdot (a \cdot d')' \cdot (b' \cdot c)'$$

$$f(a, b, c, d) = (b + d) \cdot (a' + d) \cdot (b + c')$$

- Findung einer minimalen KNF mit KV-Diagrammen

$$f_1(a, b, c, d) = \Pi(3, 4, 5, 9, 14, 15, (7, 10, 13))$$

$$f_2(a, b, c, d) = \Pi(0, 2, 3, 9, 10, 11, 13, (1, 4, 8, 15))$$

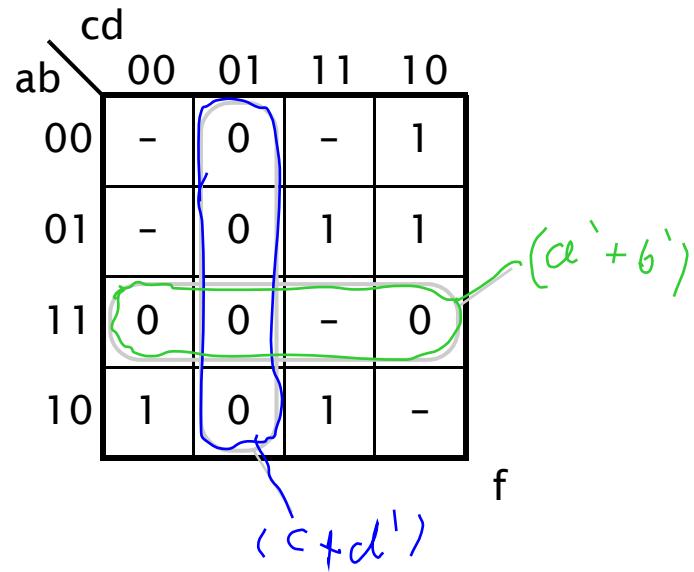
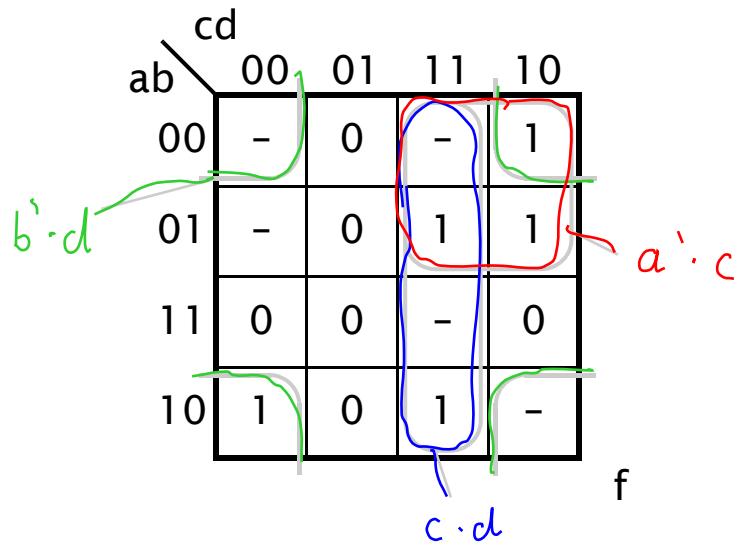


$$f_1(a, b, c, d) = (a + c' + d') \cdot (a' + b' + c') \cdot (a' + c + d') \cdot (a + b' + c)$$

$$f_2(a, b, c, d) = (a' + d') \cdot b$$

- Findung einer minimalen DNF/KNF mit KV-Diagrammen

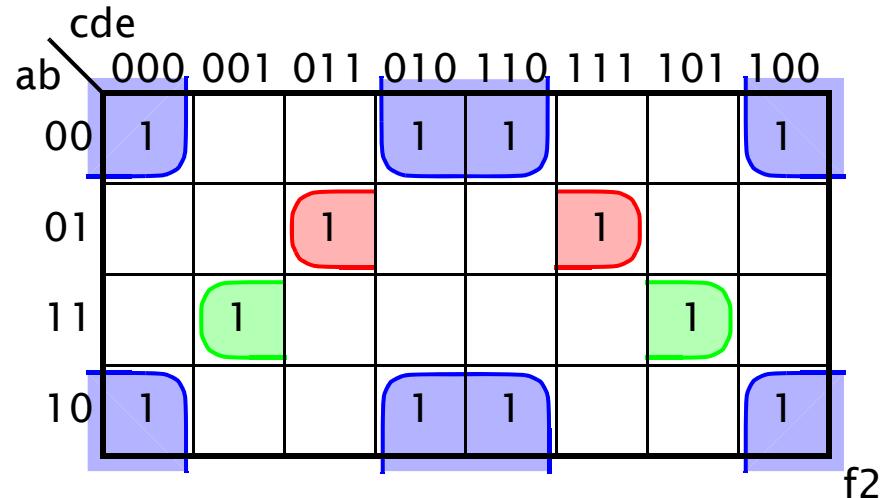
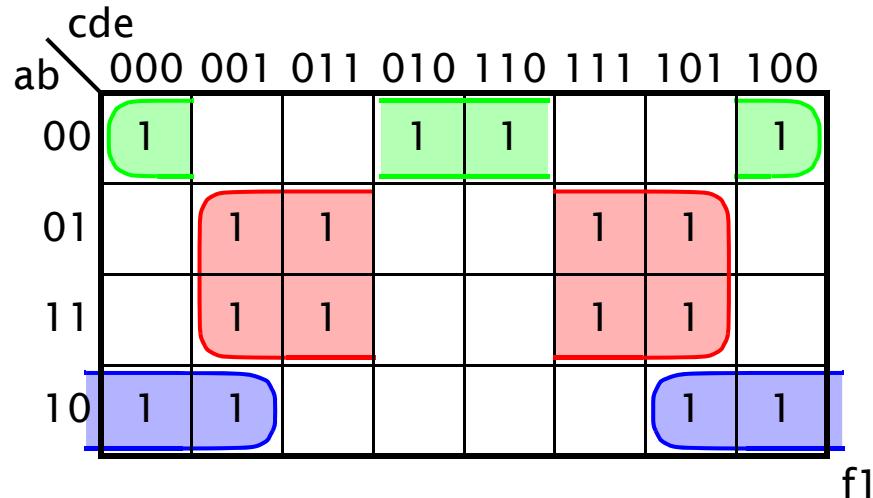
$$f(a, b, c, d) = \Sigma(2, 6, 7, 8, 11, (0, 3, 4, 10, 15))$$



als DNF:  $f(a, b, c, d) = b' \cdot d + a' \cdot c + c \cdot d$

als KNF:  $f(a, b, c, d) = (c + d') \cdot (a' + b')$

- Beispiele von KV-Diagrammen mit 5 Variablen



$$f_1(a, b, c, d, e) = a' \cdot b' \cdot e' + b \cdot e + a \cdot b' \cdot d'$$

$$f_2(a, b, c, d, e) = a' \cdot b \cdot d \cdot e + a \cdot b \cdot d' \cdot e + b' \cdot e'$$

- ◆ Die QM-Methode ist
  - eine nach Quine und McCluskey benannte Methode zur Minimierung boolescher Funktionen
  - systematisch, algorithmisch-orientiert, und relativ einfach in der Anwendung
  - theoretisch für beliebig viele Variablen anwendbar, weil sie keinen Einschränkungen bezüglich einer geometrischen Anordnung benachbarter Minterme bzw. Maxterme unterliegt (wie bei der KV-Methode), praktisch aber wegen NP-Komplexität begrenzt.
- ◆ Die QM-Methode hat als Ziel, üblicherweise aus einer gegebenen KDNF eine äquivalente DNF zu bestimmen, die
  1. eine minimale Anzahl an Implikanten aufweist und
  2. deren Implikanten so kurz wie möglich sind (d.h. so wenige Variable wie möglich enthalten).

- ♦ Ein *Implikant* ist ein Produktterm, für den gilt, daß wenn der Produktterm =1 ist, dann auch der Funktionswert =1 ist.

Beispiel:  $f(a, b, c) = \Sigma(0, 4, 5, 6, 7)$

alle Minterme von  $f$  sind automatisch Implikanten von  $f$ :

$a' \cdot b' \cdot c'$ ,  $a \cdot b' \cdot c'$ ,  $a \cdot b' \cdot c$ ,  
 $a \cdot b \cdot c'$ ,  $a \cdot b \cdot c$

		bc	00	01	11	10	
		a	0	1	0	0	0
0	1	0	1	1	1	1	f
		1	1	1	1	1	

aber auch folgende (Produkt-)Terme sind Implikanten von  $f$ :

$b' \cdot c'$ ,  $a \cdot b'$ ,  $a \cdot b$ ,  $a \cdot c'$ ,  $a \cdot c$ ,  $a$

- ◆ Ein *Primimplikant* P ist ein Implikant minimaler Länge (Anzahl der Variablen), der in keinem anderen Implikanten vollständig enthalten ist.  
Primimplikanten von f sind folgende Produktterme:  $b' \cdot c'$  und a
- ◆ Ein *Kernimplikant* K ist ein Primimplikant, der als einziger einen Minterm einer Funktion umfaßt. Ein Kernimplikant muß auf jeden Fall in die minimale Formel aufgenommen werden.  
Kernimplikanten von f sind folgende Primimplikanten:  $b' \cdot c'$  und a
- ◆ Ein *redundanter Primimplikant* ist ein Primimplikant, der bereits durch einen Kernimplikant abgedeckt ist.  
redundante Primimplikanten von f sind: keine

## ♦ Regelwerk

1. Ermittlung aller Minterme der zu minimierenden Funktion (KDNF).
2. Bildung von Minterm-Gruppen aus Mintermen mit gleicher Anzahl nicht negierter Variablen.
3. Bildung von Implikanten aus „benachbarten“ Minterm-Gruppen durch systematischen Vergleich unter Anwendung von Axiomen A5, A8 und A9, und Kennzeichnung zusammengefaßter Minterme.
4. Wiederholung von Schritt 3 mit den Implikanten, bis keine Vereinfachung mehr möglich ist. Alle nicht gekennzeichneten Minterme/Implikanten bilden Primimplikanten der Funktion.
5. Entfernung redundanter Primimplikanten.
6. Ermittlung von Kernimplikanten aus den gefundenen Primimplikanten mit Hilfe einer Überdeckungstabelle.
7. Disjunktive Verknüpfung von Kernimplikanten zur minimalen Formel.

- ◆ Ermittlung aller Minterme der zu minimierenden Funktion  
 $f(a, b, c, d) = \Sigma(0, 2, 5, 8, 10, 13, 14, 15)$
- ◆ Bildung von Minterm-Gruppen aus Mintermen mit gleicher Anzahl nicht negierter Variablen

\* Anzahl Einsen

Nr.	a	b	c	d
0	0	0	0	0
2	0	0	1	0
5	0	1	0	1
8	1	0	0	0
10	1	0	1	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

0					0
1					1
2					1
1					2
2					2
3					3
3					3
4					4

a	b	c	d	
0	0	0	0	0
0	0	1	0	1
1	0	0	0	1
0	1	0	1	2
1	0	1	0	2
1	1	0	1	3
1	1	1	0	3
1	1	1	1	4

mit fünf Minterm-Gruppen für die Funktion  $f(a, b, c, d)$   
 $G_0=(0), G_1=(2, 8), G_2=(5, 10), G_3=(13, 14)$  und  $G_4=(15)$

## ◆ Bildung von Implikanten

a) zwei Minterme werden zu einem Implikanten zusammengefaßt, wenn sie sich nur an einer einzigen Variablenposition unterscheiden. Diese Variablenposition wird dann mit einem „don't care“-Symbol markiert.

$$a \cdot b' \cdot c \cdot d' + a \cdot b' \cdot c \cdot d' \stackrel{A5}{=} a \cdot b' \cdot d' \cdot (c \cdot c) \stackrel{Ag}{=} a \cdot b' \cdot d'$$

a	b	c	d
1	0	0	0
1	0	1	0

 $\Rightarrow$ 

a	b	c	d
1	0	-	0

b) zwei Implikanten werden zu einem neuen Implikanten zusammengefaßt, wenn (1) sie „don't care“-Symbole an den gleichen Variablenpositionen haben, und (2) sich sonst nur an einer Variablenposition unterscheiden. Diese Variablenposition wird dann mit einem „don't care“-Symbol markiert.

$$a \cdot c' \cdot d' + a \cdot c' \cdot d \stackrel{A5}{=} a \cdot c' \cdot (d' + d) \stackrel{Ag}{=} a \cdot c' \cdot 1 \stackrel{A8}{=} a \cdot c'$$

a	b	c	d
1	-	0	0
1	-	0	1

 $\Rightarrow$ 

a	b	c	d
1	-	0	-

- ◆ Bildung von Implikanten aus Mintermen  
systematischer Vergleich von „benachbarten“ Minterm-Gruppen  
 $G_i$  mit  $G_{i+1}$  für  $i = 0..n-2$   
hier:  $G_0$  mit  $G_1$ ,  $G_1$  mit  $G_2$ ,  $G_2$  mit  $G_3$  und  $G_3$  mit  $G_4$

	a	b	c	d	
$G_0$	0	0	0	0	✓
$G_1$	2	0	0	1	0
	8	1	0	0	0
$G_2$	5	0	1	0	1
	10	1	0	1	0
$G_3$	13	1	1	0	1
	14	1	1	1	0
$G_4$	15	1	1	1	1

	a	b	c	d	
	(0,2)	0	0	—	0
	(0,8)	—	0	0	0
	(2,10)	—	0	1	0
	(8,10)	1	0	—	0
	(5,13)	—	1	0	1
	(10,14)	1	—	1	0
	(13,15)	1	1	—	1
	(14,15)	1	1	1	—

Jeder Minterm, der zu einem Implikanten zusammengefaßt ist, wird mit ✓ markiert

- ◆ Bildung von Implikanten aus Implikanten  
systematischer Vergleich von „benachbarten“ Implikanten-Gruppen, hier:  $G_0$  mit  $G_1$ ,  $G_1$  mit  $G_2$  und  $G_2$  mit  $G_3$

	a	b	c	d	$G_k$
0,2	0	0	-	0	0
0,8	-	0	0	0	0
2,10	-	0	1	0	1
8,10	1	0	-	0	1
5,13	-	1	0	1	2
10,14	1	-	1	0	2
13,15	1	1	-	1	3
14,15	1	1	1	-	3

(0,2,8,10) ~~(0,8,2,10)~~

a	b	c	d	$G_j$
-	0	-	0	
0	-	0	0	

$P_1 = (-101)_2$ ,  $P_2 = (1-10)_2$ ,  $P_3 = (11-1)_2$ ,  $P_4 = (111-)_2$ ,  $P_5 = (-0-0)_2$   
Alle nicht gekennzeichneten Implikanten/Minterme stellen Primimplikanten dar. Redundante Primimplikanten werden entfernt.

- ◆ Ermittlung von Kernimplikanten nach Bowman und McVey
    - schnelles, iteratives, matrixorientiertes Verfahren zur Ermittlung minimaler Abdeckung; findet in den meisten Fällen eine minimale Abdeckung, oder eine annährend minimale Abdeckung
  - ◆ Für diese Methode gelten:
    - $N$  Anzahl von Primimplikanten  $P_i$
    - $K$  Anzahl der Minterme  $m_i$
    - $R_j$  Anzahl der Primimplikanten, in denen der Minterm  $m_j$  enthalten ist, für  $j=1..K$
    - $S_j$  Stärke der Abdeckung eines Minterms  $m_j$
1. Initialisierung mit  $S_j = 1/R_j$  für  $j=1..K$
- Solange es nicht abgedeckte Minterme gibt, berechne:
2.  $W_i = \sum(S_j)$  - Gewicht eines Primimplikanten  $P_i$ ; für  $i=1..N$ ,  $j=1..K$
  3.  $W_{\max} = \max(W_i)$  für  $i=1..N$  bestimmt den Kernimplikanten  $K_{\max}$
  4.  $S_j := 0$  für alle Minterme  $m_j$ , die durch den  $K_{\max}$  abgedeckt sind

- ◆ Ermittlung von Kernimplikanten N=5, K=8

Minterme aus der Funktionstabelle

abcd		0000	0010	0101	1000	1010	1101	1110	1111	$W_i = \sum(S_j)$
		Nr	0	2	5	8	10	13	14	15
P1	-101	5,13			X			X		1,50
P2	1-10	10,14					X		X	1,00
P3	11-1	13,15						X		1,00
P4	111-	14,15							X X	1,00
P5	-0-0	0,2,8,10	X	X		X X				3,50
		$S_j = 1/R_j$	1	1	1	1/1	1/2	1/2	1/2	K1

$$W_{\max} = 3,50 \Rightarrow$$

Primimplikant P5 =  $(-0-0)_2 = b' \cdot d'$  ist Kernimplikant K1  
 (Minterme 0, 2, 8 und 10)

♦ Ermittlung von Kernimplikanten N=5, K=8

Minterme aus der Funktionstabelle

abcd		0000	0010	0101	1000	1010	1101	1110	1111	$W_i = \Sigma(S_j)$
Primimplikanten	Nr	0	2	5	8	10	13	14	15	
	-101	5,13			x		x			1,50
	1-10	10,14				x		x		0,50
	11-1	13,15					x		x	1,00
	111-	14,15						x	x	1,00
	-0-0	0,2,8,10	x	x		x	x			0,00
		$S_j = 1/R_j$	0	0	1	0	0	1/2	1/2	1/2

$$W_{\max} = 1,50 \Rightarrow$$

Primimplikant  $P_1 = (-101)_2 = b \cdot c' \cdot d$  ist Kernimplikant K2  
(Minterme 5 und 13).

$\partial + \frac{1}{2}$

♦ Ermittlung von Kernimplikanten N=5, K=8

Minterme aus der Funktionstabelle

abcd		0000	0010	0101	1000	1010	1101	1110	1111	$W_i = \sum(S_j)$
Primimplikanten	Nr	0	2	5	8	10	13	14	15	
	-101	5,13		x			x			0,00
	1-10	10,14				x		x		0,50
	11-1	13,15					x		x	0,50
	111-	14,15						x	x	1,00
	-0-0	0,2,8,10	x	x	x	x				0,00
$S_j = 1/R_j$		0	0	0	0	0	0	1/2	1/2	

$$W_{\max} = 1,00 \Rightarrow$$



Primimplikant  $P_4 = (111-)_2 = a \cdot b \cdot c$  ist Kernimplikant K3

(Minterme 14,15)

Disjunktive Verknüpfung von Kernimplikanten:

$$f(a, b, c, d) = K1 + K2 + K3 = b' \cdot d' + b \cdot c' \cdot d + a \cdot b \cdot c$$

- Beispiel mit einer unvollständig definierten Funktion  
 $f(a, b, c, d) = \Sigma(0, 1, 2, 4, 5, 10, (8, 12, 14))$

	a	b	c	d	
0	0	0	0	0	✓
1	0	0	0	1	✓
2	0	0	1	0	✓
4	0	1	0	0	✓
(8)	1	0	0	0	✓
5	0	1	0	1	✓
10	1	0	1	0	✓
(12)	1	1	0	0	✓
(14)	1	1	1	0	✓

	a	b	c	d	
(0,1)	0	0	0	-	
(0,2)	0	0	-	0	
(0,4)	0	-	0	0	
(0,8)	-	0	0	0	
(1,5)	0	-	0	1	
(2,10)	-	0	1	0	
(4,5)	0	1	0	-	
(4,12)	-	1	0	0	
(8,10)	1	0	-	0	
(8,12)	1	-	0	0	
(10,14)	1	-	1	0	
(12,14)	1	1	-	0	

- ◆ Bildung von Implikanten

	a	b	c	d	
0,1	0	0	0	-	✓
0,2	0	0	-	0	✓
0,4	0	-	0	0	✓
0,8	-	0	0	0	✓
1,5	0	-	0	1	✓
2,10	-	0	1	0	✓
4,5	0	1	0	-	✓
4,12	-	1	0	0	✓
8,10	1	0	-	0	✓
8,12	1	-	0	0	✓
10,14	1	-	1	0	✓
12,14	1	1	-	0	✓

a	b	c	d
(0,1,4,5)	0	-	0
(0,2,8,10)	-	0	-
(0,4,15)	0	-	0
(0,4,8,12)	-	-	0
(0,8,2,10)	-	0	-
(0,8,4,12)	-	-	0
(8,10,12,14)	1	-	-
(8,12,10,14)	1	-	-

- ◆ Entfernung redundanter Primimplikanten

	a	b	c	d	
0,1,4,5	0	-	0	-	P1
0,2,8,10	-	0	-	0	P2
0,4,8,12	-	-	0	0	P3
8,10,12,14	1	-	-	0	P4

Primimplikanten:

$$P1 = (0-0-)_2 = a' \cdot c'$$

$$P2 = (-0-0)_2 = b' \cdot d'$$

$$P3 = (--00)_2 = c' \cdot d'$$

$$P4 = (1--0)_2 = a \cdot d'$$

- ◆ Ermittlung von Kernimplikanten mit  $N = 4, K = 6$  (ohne „don't care“-Terme), hier also ohne 8, 12 und 14

	Nr	0	1	2	4	5	10	$W_i = \sum(S_j)$	
$P_1$	0,1,4,5	X	X		X	X		2,8̄3	0
$P_2$	0,2,8,10	X		X			X	1,8̄3	7,5
$P_3$	0,4,8,12	X			X			0,8̄3	6
$P_4$	8,10,12,14						X	6,5	0,5
	$S_j = 1/R_j$	1/3	1	1	1/2	1	1/2		
		0	0	1	0	0	1/2		
		0	0	0	0	0	0		

$0 \Rightarrow \text{ende}$

$$K_1 = P_1 (0,1,4,5)$$

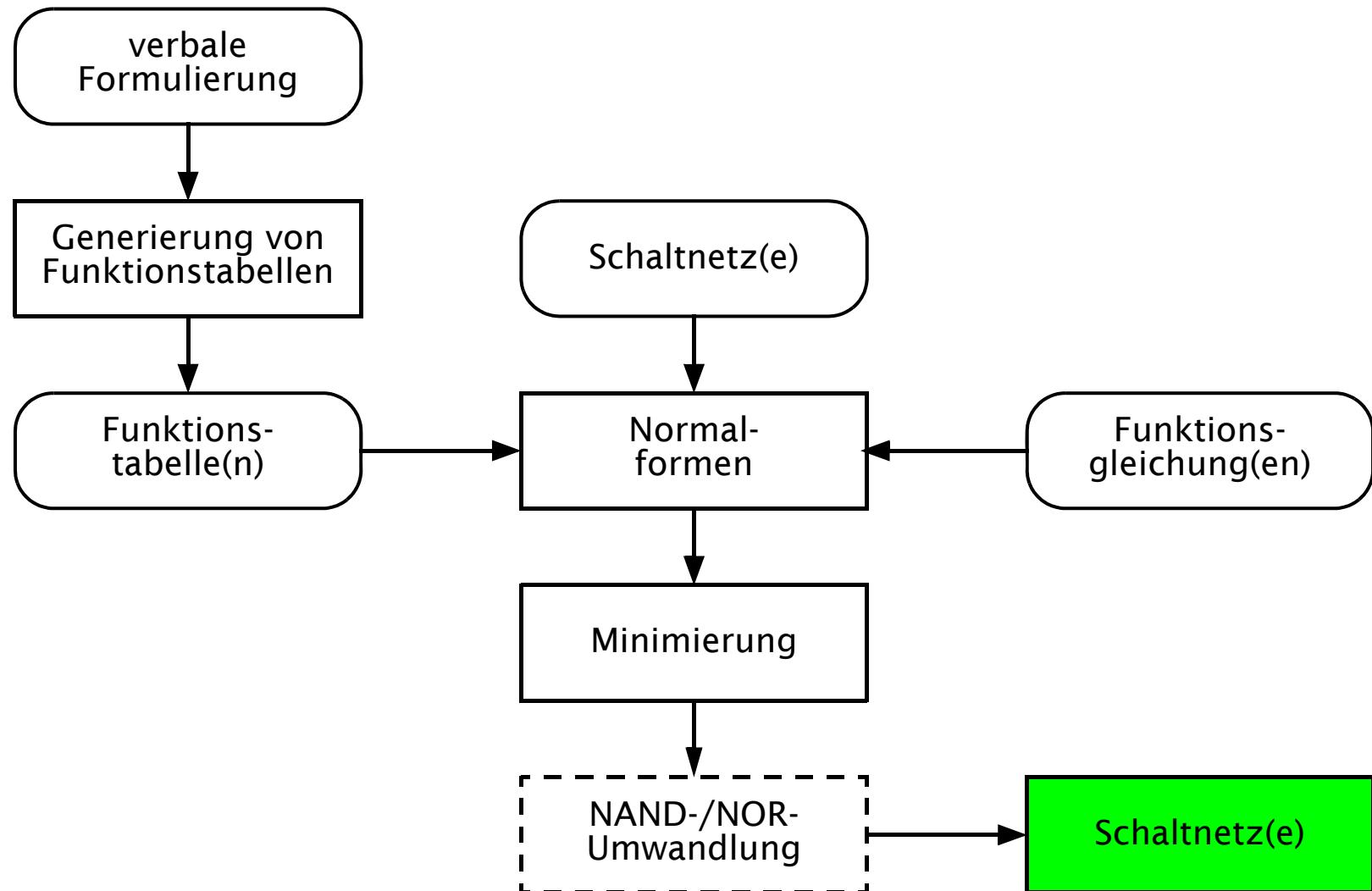
$$K_2 = P_2 (0,2,8,10)$$

$$f(a, b, c, d) = K_1 + K_2 = P_1 + P_2 = \underline{\dot{a} \cdot \dot{c}} + \underline{\dot{b} \cdot \dot{d}}$$

# Zusammenfassung

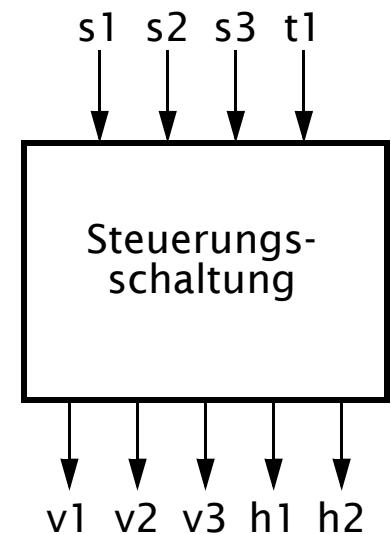
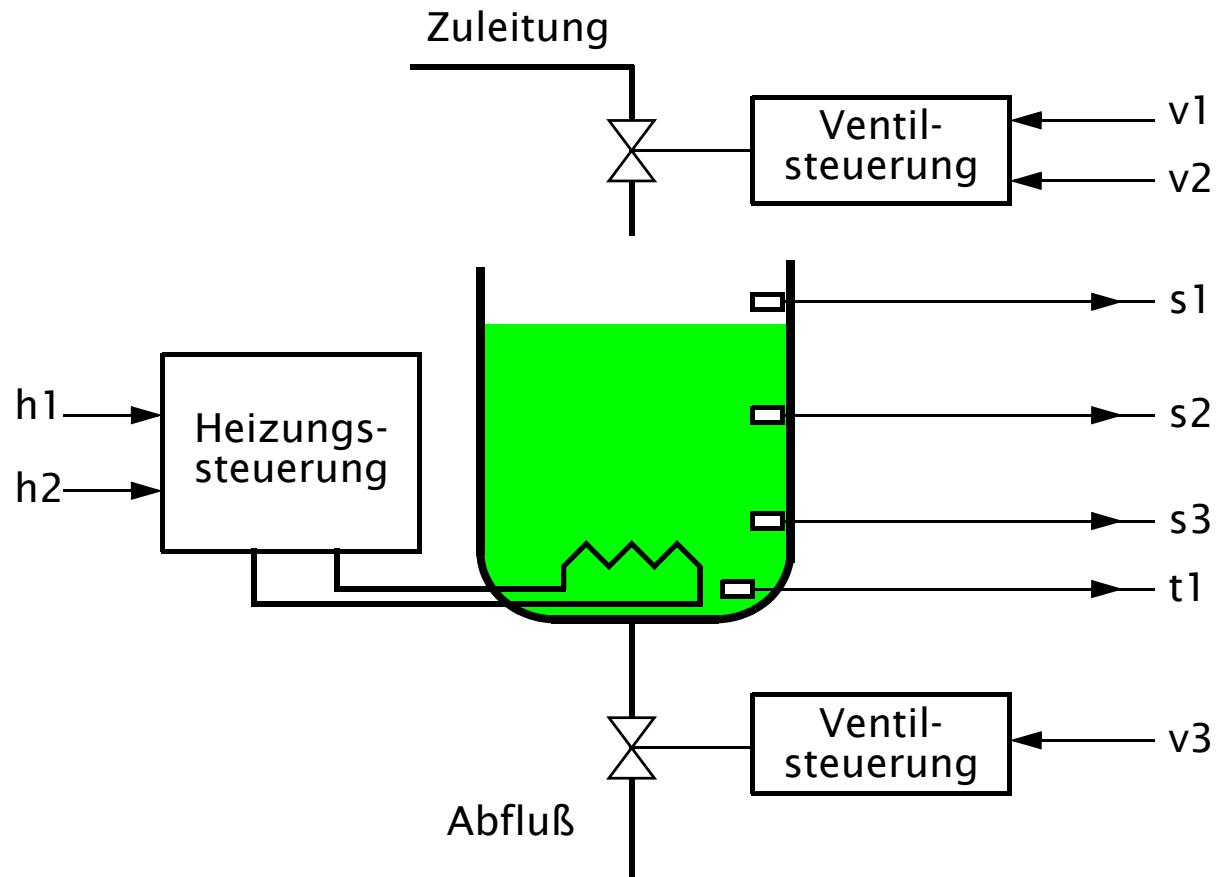
- ◆ Überblick über Minimierungsverfahren
  - Direkte Anwendung der booleschen Algebra
    - Vorteile: wenig Aufwand, für kleine Schaltnetze gut geeignet
    - Nachteile: Lösung stark von der Intuition und Erfahrung des Benutzers abhängig und stellt nicht immer das Optimum dar
  - KV-Methode
    - Vorteile: sehr anschaulich, Vereinfachung einer booleschen Funktion erfolgt in einem einzigen Schritt
    - Nachteile: für mehr als 6 Variablen unhandlich und unübersichtlich
  - QM-Methode
    - Vorteile: sehr gut geeignet auch bei vielen Eingangsvariablen und für Lösung mit Hilfe von Digitalrechnern geeignet
    - Nachteile: wenig anschaulich, erfordert viel Schreibaufwand, Vereinfachung erfolgt schrittweise

- ♦ Synthesemöglichkeiten



- ◆ Entwurf einer Steuerungsschaltung (als Schaltnetz) für eine verfahrenstechnische Anlage
- ◆ Die Anlage besteht aus
  - einem Kessel, in den über eine Zuleitung eine Flüssigkeit eingefüllt werden kann,
  - digitalen Sensoren zur Überwachung des Flüssigkeitspegels im Kessel und der Temperatur der Flüssigkeit,
  - digitalen Modulen zur Steuerung der Heizung sowie Zu- und Ablaufventile.
- ◆ Die Hauptaufgabe der Anlage ist es,
  - die Flüssigkeit im Kessel auf eine vorgegebene Mindesttemperatur zu erhitzen und an eine nachfolgende Anlage weiter zu leiten.

- ♦ schematischer Aufbau der Anlage



- Der Zufluß wird über eine Ventilstellung in drei Schritten mit zwei Signalen  $v1$  und  $v2$  gesteuert.

$v2$	$v1$	Ventilstellung
0	0	geschlossen
0	1	halb offen
1	1	ganz offen

- Der Flüssigkeitspegel wird über drei digitale Sensoren erfaßt und über die Signale  $s1$ ,  $s2$  und  $s3$  an die Steuerungsschaltung gemeldet.

$s3$	$s2$	$s1$	Pegelstand
1	1	1	Kessel voll
1	1	0	2/3 voll
1	0	0	1/3 voll
0	0	0	Kessel leer

- Das Erreichen der Solltemperatur wird über einen digitalen Thermo-schalter erfaßt und mit dem Signal  $t1$  an die Steuerungsschaltung gemeldet.
- Die Heizung kann über eine Heizungssteuerung in drei Leistungsstufen 0%, 50% und 100% der Leistung mit zwei Signalen  $h1$  und  $h2$  geschaltet werden.
- Solange der Behälter nicht leer ist und die Flüssigkeit warm genug ist, kann der Ablauf über eine Ventilsteuerung mit dem Signal  $v3$  geöffnet werden.

$t1$	Temperatur
0	Temp. < Soll
1	Temp. $\geq$ Soll

$h2$	$h1$	Leistung
0	0	0%
0	1	50%
1	1	100%

- ◆ Steuerungstabelle

beschreibt die Funktionsweise der Steuerung in der Abhängigkeit vom Flüssigkeitspegel und der aktuellen Temperatur

Flüssigkeitspegel H	Temp. $\geq$ Soll	Temp. $<$ Soll
$s_1 < H$	Heizung 0% Ablauf offen Zulauf zu <i>A</i>	Heizung 100% Ablauf zu <i>B</i> Zulauf zu
$s_2 < H < s_1$	Heizung 0% Ablauf offen Zulauf halb offen <i>C</i>	Heizung 50% Ablauf zu <i>D</i> Zulauf halb offen
$s_3 < H < s_2$	Heizung 0% Ablauf offen Zulauf ganz offen <i>E</i>	Heizung 50% <i>F</i> Ablauf zu Zulauf ganz offen
$H < s_3$	Heizung 0% Ablauf zu <i>G</i> Zulauf ganz offen	Heizung 0% Ablauf zu <i>H</i> Zulauf ganz offen

## ◆ Funktionstabelle

	Eingangssignale				Ausgangssignale				
	s3	s2	s1	t1	v1	v2	v3	h1	h2
15	A	1	1	1	1	0	0	1	0
14	B	1	1	1	0	0	0	1	1
13	C	1	1	0	1	1	0	1	0
12	D	1	1	0	0	1	0	0	1
9	E	1	0	0	1	1	1	0	0
8	F	1	0	0	0	1	1	0	1
1	G	0	0	0	1	1	0	0	0
0	H	0	0	0	0	1	1	0	0
2,3,4,5, 6,7,10,11		{ 0	-	1	-	-	-	-	-
		1	0	1	-	-	-	-	-
		0	1	0	-	-	-	-	-

♦ Kanonische Disjunktive Normalformeln:

$$v1(s_3, s_2, s_1, t_1) = \sum (0, 1, 8, 9, 12, 13, (2, 3, 4, 5, 6, 7, 10, 11))$$

$$v2(s_3, s_2, s_1, t_1) = \sum (0, 1, 8, 9, (2, 3, 4, 5, 6, 7, 10, 11))$$

$$v3(s_3, s_2, s_1, t_1) = \sum (9, 13, 15, (2, 3, 4, 5, 6, 7, 10, 11))$$

$$h1(s_3, s_2, s_1, t_1) = \sum (8, 12, 14, (2, 3, 4, 5, 6, 7, 10, 11))$$

$$h2(s_3, s_2, s_1, t_1) = \sum (14, (2, 3, 4, 5, 6, 7, 10, 11))$$

- ◆ Minimierung mit der KV-Methode

	s1 t1	00	01	11	10
s3 s2	00	1	1	-	-
00	-	-	-	-	-
01	-	-	-	-	-
11	1	1	0	0	-
10	1	1	-	-	-

	s1 t1	00	01	11	10
s3 s2	00	1	1	-	-
01	-	-	-	-	-
11	0	0	0	0	-
10	1	1	-	-	-

	s1 t1	00	01	11	10
s3 s2	00	0	0	-	-
01	-	-	-	-	-
11	0	1	1	0	-
10	0	1	-	-	-

$$V_1(s_3, s_2, s_1, t_1) = s_1$$

$$V_2(s_3, s_2, s_1, t_1) = s_2$$

$$V_3(s_3, s_2, s_1, t_1) = s_3 \cdot t_1$$

- ◆ Minimierung mit der KV-Methode

		s1	t1			
		00	01	11	10	
s3 s2		00	0	0	-	-
00	01	-	-	-	-	-
01	11	-	-	-	-	-
11	10	1	0	0	1	-
10	00	1	0	-	-	-

h1

$$h_1(s_3, s_2, s_1, t_1) = s_3 \cdot \bar{t}_1$$

		s1	t1			
		00	01	11	10	
s3,s2		00	0	0	-	-
00	01	-	-	-	-	-
01	11	0	0	0	1	-
11	10	0	0	-	-	-

h2

$$h_2(s_3, s_2, s_1, t_1) = s_1 \cdot \bar{t}_1$$

- ♦ Interpretation der Ergebnisse

- $v1 = s1'$ : Das Zulaufventil wird mindestens halb geöffnet, solange der Kessel noch nicht voll ist.
- $v2 = s2'$ : Das Zulaufventil wird ganz geöffnet, wenn der Kessel weniger als halb voll ist.
- $v3 = s3 \cdot t1'$ : Das Auslaßventil wird geöffnet, wenn die Soll-Temperatur erreicht ist ( $t1'$ ) und der Kessel noch nicht leer ist (d.h. der Flüssigkeitspegel über dem Sensor  $s3$  liegt).
- $h1 = s3 \cdot t1'$ : Die Heizung arbeitet mindestens mit halber Leistung, solange der Kessel nicht leer ist ( $s3$ ) und die Soll-Temperatur nicht erreicht ist ( $t1'$ ).
- $h2 = s1 \cdot t1'$ : Die Heizung arbeitet mit voller Leistung, wenn der Kessel voll ist ( $s1$ ) und die Soll-Temperatur nicht erreicht ist ( $t1'$ ).

## ♦ Erweiterte Analyse

- Im Beispiel konnten die Ausgangswerte für einige Kombinationen der Eingangssignale beliebig belegt werden, weil sie physikalisch nicht möglich sind.
- Diese Aussage gilt aber, nur solange man von einer korrekten Funktionsweise aller Sensoren ausgeht.
- Läßt man diese Annahme fallen, weil man in der Praxis durchaus mit einer wahrscheinlichen Möglichkeit eines Sensordefektes rechnet, so können die vorher unmöglichen Kombinationen auftreten.
- Man muß also entscheiden, was in den einzelnen Fällen zu tun ist.
- Eine Möglichkeit besteht darin, auf die technisch sichere Seite zu gehen, und im Falle eines Sensordefektes die Anlage anzuhalten.
- Für diese Variante wird man sich immer dann entscheiden, wenn der weitere Betrieb der Anlage bei einem defekten Sensor mit einem hohen Risiko verbunden ist (in sicherheitskritischen Systemen).

## ♦ Erweiterte Analyse

- Ist das Risiko vertretbar, so bieten sich andere, sinnvolle Möglichkeiten an.
- Man kann die Steuerung *fehlertolerant* entwerfen, so daß die Anlage auch mit einem defekten Sensor weiter arbeitet kann, sofern der Ausfall eindeutig einem Sensor zugeordnet werden kann.
- Man kann dann für diesen Sensor den richtigen Wert annehmen und mit der so korrigierten Kombination von Eingangssignalen weiterarbeiten.
- Die Steuerung der Anlage muß derart erweitert werden, daß sie in der Lage ist, diesen Fehlerzustand zu detektieren und ggf. zu signalisieren.
- Dazu wird die Funktionstabelle um einen zusätzlichen Ausgangssignal für eine Kontrolllampe  $x_1$  erweitert, die einem Wartungstechniker einen defekten Sensor meldet.

- ♦ erweiterte Funktionstabelle mit Fehleranzeige

	Eingangssignale					Ausgangssignale				
	s3	s2	s1	t1	v1	v2	v3	h1	h2	x1
15 -	1	1	1	1	0	0	1	0	0	0
14 -	1	1	1	0	0	0	0	1	1	0
13 -	1	1	0	1	1	0	1	0	0	0
12 -	1	1	0	0	1	0	0	1	0	0
9 -	1	0	0	1	1	1	1	0	0	0
8 -	1	0	0	0	1	1	0	1	0	0
1 -	0	0	0	1	1	1	0	0	0	0
0 -	0	0	0	0	1	1	0	0	0	0
2,3	0	0	1	-	1	1	0	0	0	1
7	0	1	1	1	0	0	1	0	0	1
6	0	0	1	0	0	0	0	1	1	1
10,11	1	0	1	-	0	0	0	0	0	1
4,5	0	1	1	-	0	0	0	0	0	1

$$v1 = \sum (13, 12, 9, 8, 1, 0, 2, 3)$$

- Minimierung mit der KV-Methode

s3	s2	s1	t1	v1
00	01	11	10	
01	00	00	00	
11	11	10	00	
10	11	00	00	

$$v_1(s_3, s_2, s_1, t_1) = s_3 \cdot s_1' + s_3' \cdot s_2'$$

s3	s2	s1	t1	v2
00	01	11	10	
01	00	00	00	
11	00	00	00	
10	00	00	00	

$$v_2(s_3, s_2, s_1, t_1) = s_3 \cdot s_2' + s_2 \cdot s_1$$

$$v_3(s_3, s_2, s_1, t_1) = s_2 \cdot s_1 \cdot t_1 + s_3 \cdot s_1 \cdot t_1$$

s3	s2	s1	t1	v3
00	01	11	10	
01	00	00	00	
11	00	00	00	
10	00	00	00	

- ◆ Minimierung mit der KV-Methode

	s1 t1	00	01	11	10
s3 s2	00				
00					
01					
11					
10					

$$h_1(s_3, s_2, s_1, t_1) = s_3 \cdot s_1' \cdot t_1' + s_2 \cdot s_1 \cdot t_1'$$

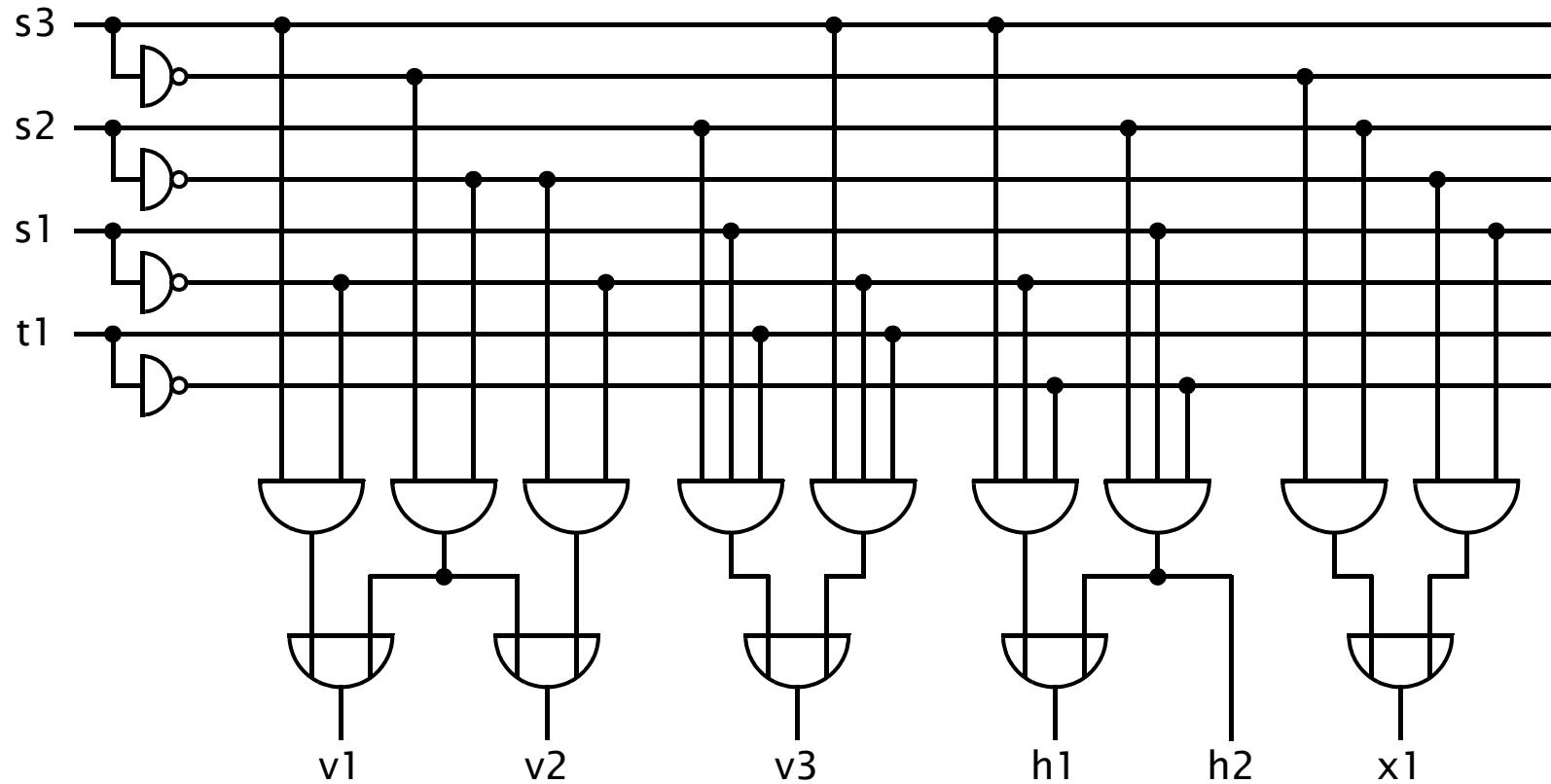
	s1 t1	00	01	11	10
s3,s2	00				
00					
01					
11					
10					

$$n_2(s_3, s_2, s_1, t_1) = s_2 \cdot s_1 \cdot t_1'$$

	s1 t1	00	01	11	10
s3,s2	00				
00					
01					
11					
10					

$$x_1(s_3, s_2, s_1, t_1) = s_3' \cdot s_2 + s_2' \cdot s_1$$

- ◆ Steuerungsschaltung



- ◆ 1-aus-n-Dekoder

- k (Steuer-/Select-/Adreß-)Eingänge:  $s_0, s_1, \dots, s_{k-1}$
- n Ausgänge:  $y_0, y_1, \dots, y_{n-1}$  mit  $n = 2^k$
- Funktionsweise:  
1-aus-n-Dekoder aktiviert in der Abhängigkeit von den Steuer-eingängen genau einen (den selektierten) Ausgang (=1), alle andere (nicht selektierte) Ausgängen bleiben deaktiviert (=0).

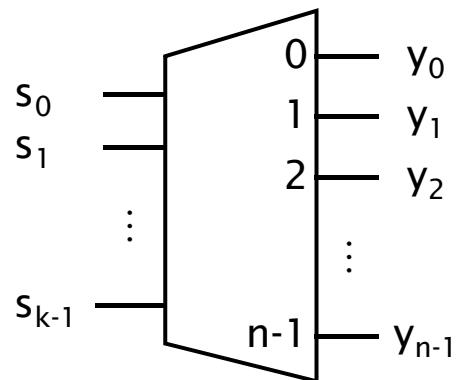
Funktionsgleichung:

für  $i = \{0, 1, \dots, n-1\}$  (mit  $n=2^k$ )

$$y_i := \begin{cases} 1 & \text{wenn } (s_{k-1}, \dots, s_1, s_0)_2 = (i)_{10} \\ 0 & \text{sonst} \end{cases}$$

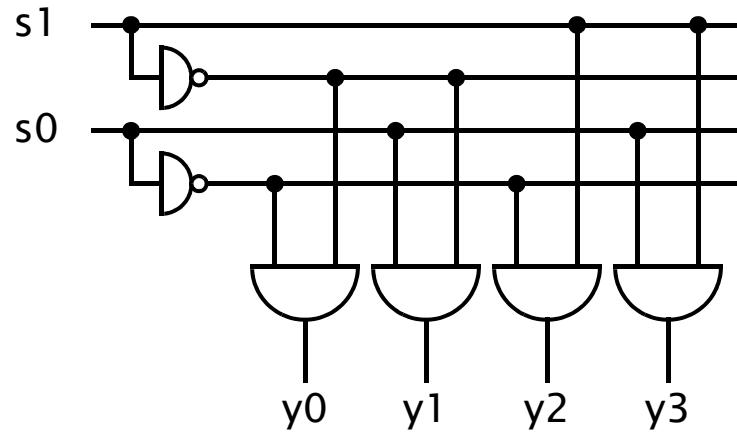
$(i)_{10}$  Nummer des Datenausgangs

$(s_{k-1}, \dots, s_1, s_0)_2$  dualcodierter Auswahlwert

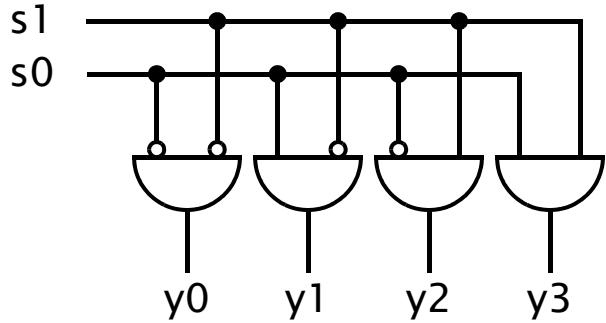


- ◆ 1-aus-4-Dekoder
  - Funktionstabelle und Schaltbilder

Eingänge		Ausgänge			
s1	s0	y3	y2	y1	y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

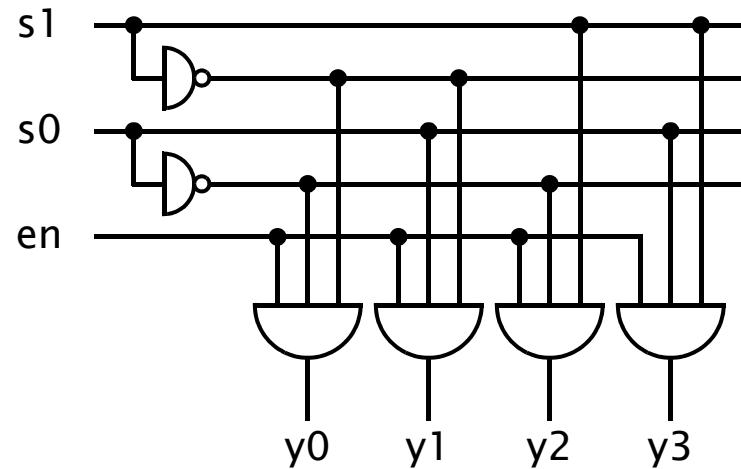


- Funktionsgleichungen



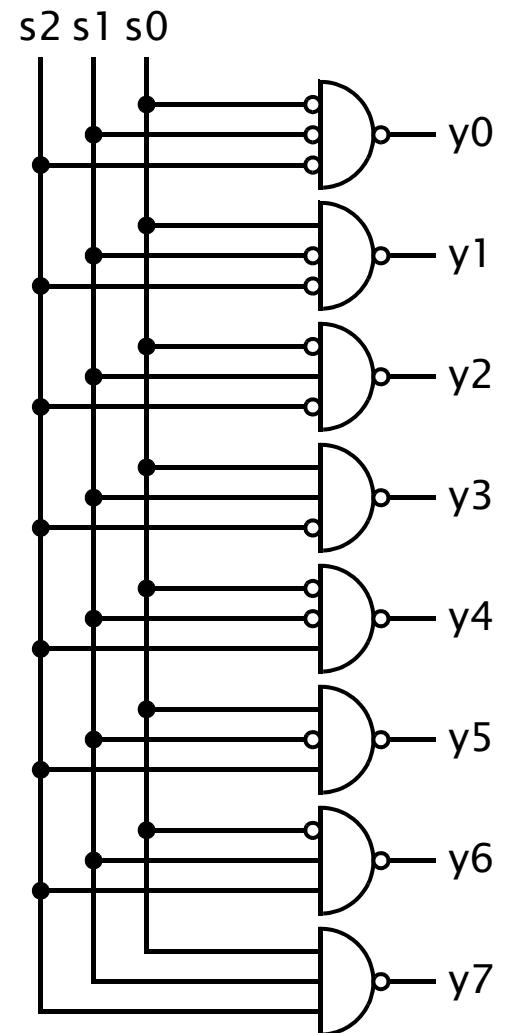
- ◆ 1-aus-4-Dekoder mit einem Enable-Eingang
  - Dekodierer-Schaltnetze sind in der Praxis oft mit einem zusätzlichen Steuereingang (sog. Enable-Eingang, en, EN) ausgestattet.
  - Dekodierer mit einem Enable-Eingang aktiviert den selektierten Ausgang nur dann, wenn der Enable-Eingang auch aktiv (=1) ist, sonst bleiben alle Ausgänge deaktiviert (=0).
  - Funktionstabelle und Schaltbild

Eingänge			Ausgänge			
en	s1	s0	y3	y2	y1	y0
0	-	-	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

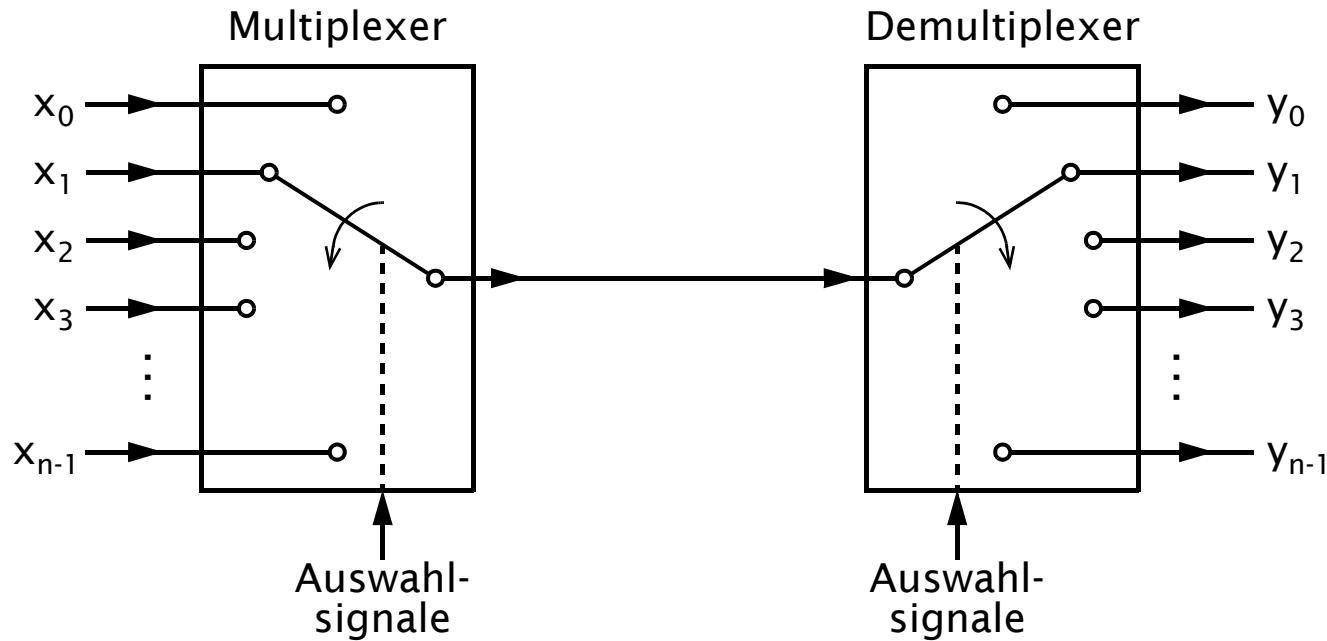


- ◆ 1-aus-8-Dekodierer mit invertierten Ausgängen
  - Funktionstabelle und Schaltbild

Eingänge			Ausgänge							
s2	s1	s0	y7	y6	y5	y4	y3	y2	y1	y0
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1



- ♦ Multiplexer (MUX) und Demultiplexer (DEMUX)
  - elektronisch gesteuerte Umschalter, (Auswahlschalter, Datenselektor)
  - Parallel-Seriell- und Seriel-Parallelwandlung von Daten
  - Erzeugung verschiedener Logikfunktionen



- ◆ 1-aus-n-Multiplexer

- k (Auswahl-/Steuer-/Select-/Adreß-)Eingänge:  $s_0, s_1, \dots, s_{k-1}$
- n (Daten-)Eingänge:  $x_0, x_1, \dots, x_{n-1}$  mit  $n = 2^k$
- 1 (Daten-)Ausgang:  $y$
- Funktionsweise:  
1-aus-n-Multiplexer schaltet in der Abhängigkeit von den Steuer-eingängen genau einen von den n Dateneingängen auf seinen Ausgang durch.

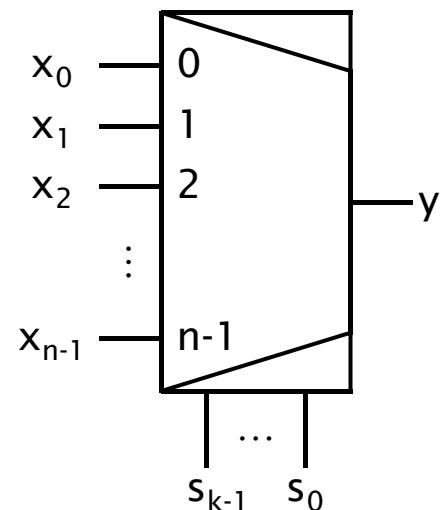
Funktionsgleichung:

für  $i = \{0, 1, \dots, n-1\}$  (mit  $n=2^k$ )

$y := x_i$  wenn  $(s_{k-1}, \dots, s_1, s_0)_2 = (i)_{10}$

$(i)_{10}$  Nummer des Dateineingangs

$(s_{k-1}, \dots, s_1, s_0)_2$  dualcodierter Aus-wahlwert

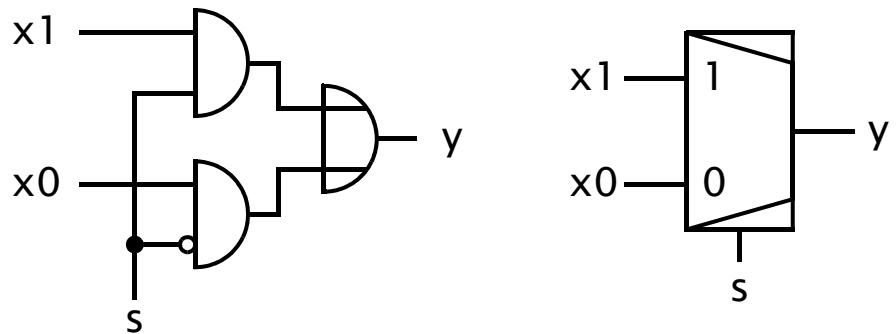
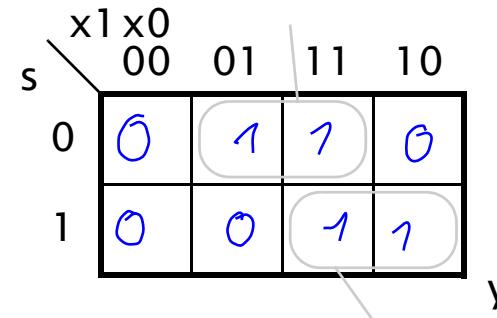


- ◆ 1-aus-2-Multiplexer
  - Funktionstabellen, KV-Diagramm und Schaltbilder

s	x1	x0	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

=>

s	y
0	x0
1	x1



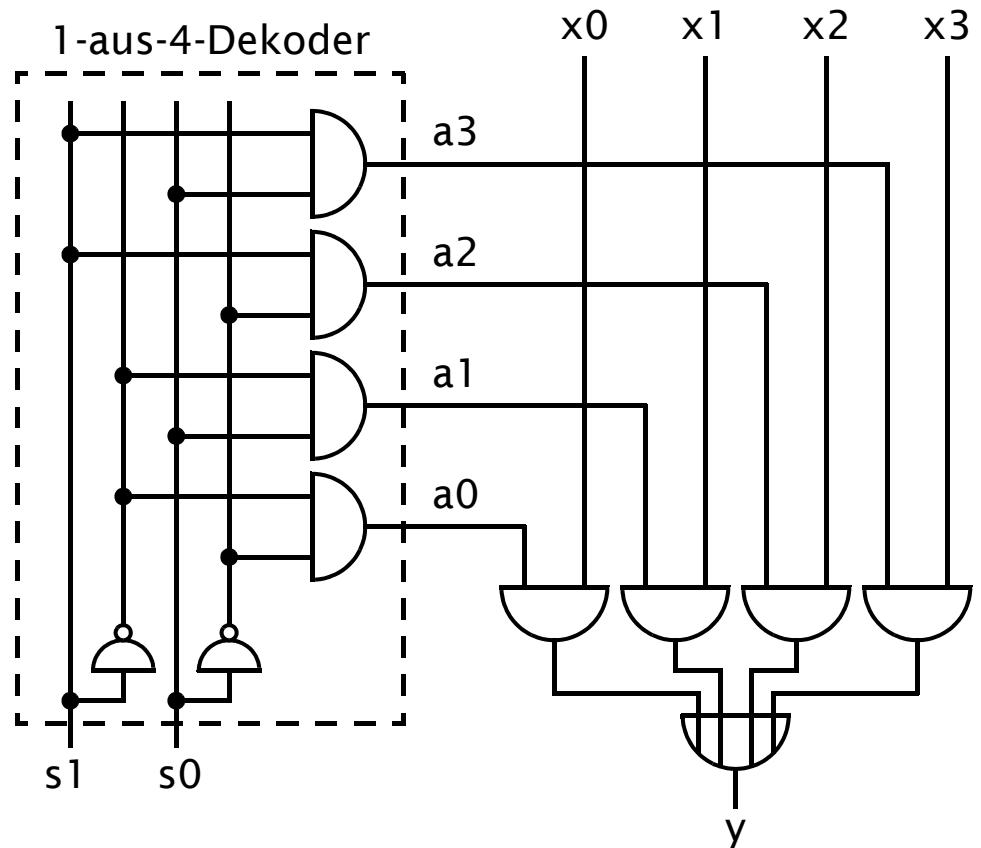
- Funktionsgleichung

- ◆ 1-aus-4-Multiplexer
  - Funktionstabelle und Schaltbild

s1	s0	y
0	0	x0
0	1	x1
1	0	x2
1	1	x3

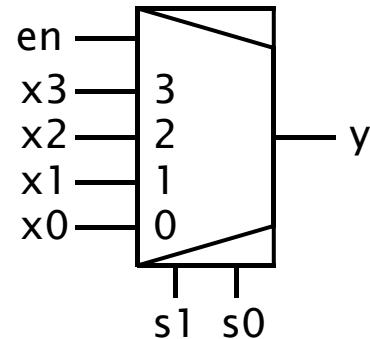
- Funktionsgleichung

$$\begin{aligned}
 y &= d_0 \cdot x_0 + d_1 \cdot x_1 \\
 &\quad + d_2 \cdot x_2 + d_3 \cdot x_3 \\
 &= (s_1 \cdot s_0) \cdot x_0 \\
 &\quad + (s_1 \cdot s_0) \cdot x_1 \\
 &\quad + (s_1 \cdot s_0) \cdot x_2 \\
 &\quad + (s_1 \cdot s_0) \cdot x_3
 \end{aligned}$$

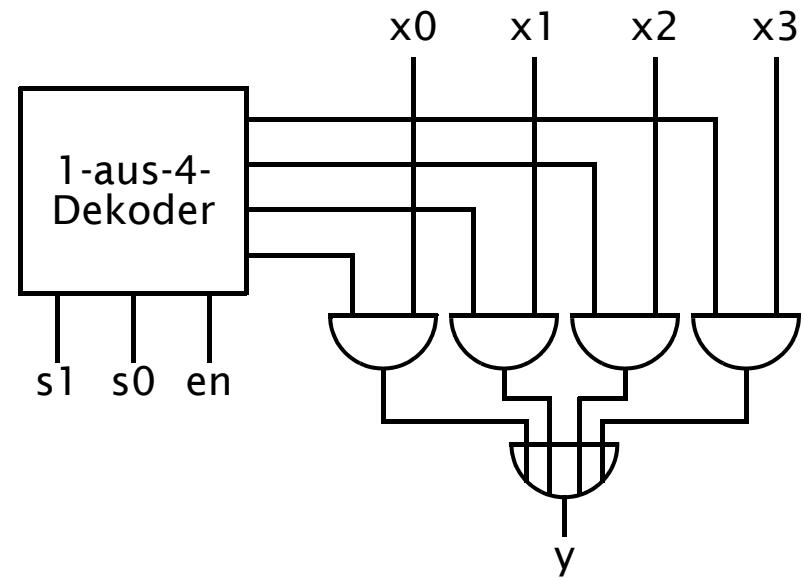


- ◆ 1-aus-4-Multiplexer mit einem Enable-Eingang
  - Funktionstabelle und Schaltbilder

en	s1	s0	y
0	-	-	0
1	0	0	x0
1	0	1	x1
1	1	0	x2
1	1	1	x3



- Funktionsgleichung



- ◆ 1-zu-n-Demultiplexer

- 1 (Daten-)Eingang:  $x$
- $k$  (Auswahl-/Steuer-/Select-/Adreß-)Eingänge:  $s_0, s_1, \dots, s_{k-1}$
- $n$  (Daten-)Ausgänge:  $y_0, y_1, \dots, y_{n-1}$  mit  $n = 2^k$
- Funktionsweise:  
1-zu-n-Demultiplexer schaltet in der Abhängigkeit von den Steuereingängen den Dateneingang auf einen von  $n$  Ausgängen durch.

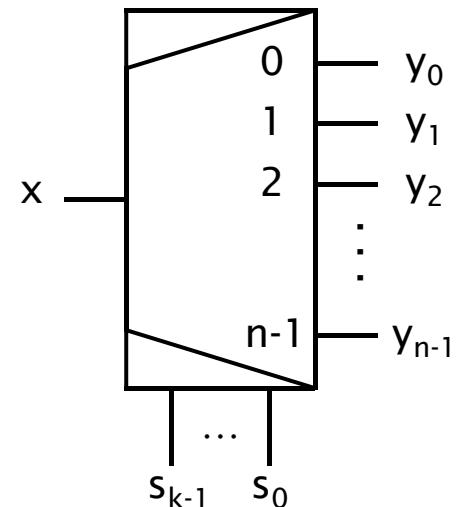
Funktionsgleichung:

für  $i = \{0, 1, \dots, n-1\}$  (mit  $n=2^k$ )

$$y_i := \begin{cases} x & \text{wenn } (s_{k-1}, \dots, s_1, s_0)_2 = (i)_{10} \\ 0 & \text{sonst} \end{cases}$$

$(i)_{10}$  Nummer des Datenausgangs

$(s_{k-1}, \dots, s_1, s_0)_2$  dualcodierter Auswahlwert



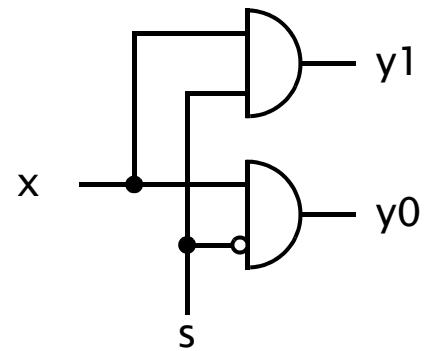
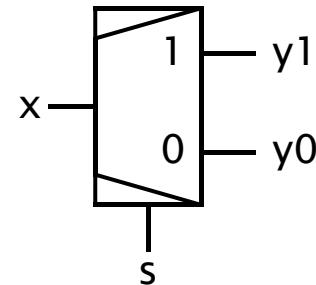
# Schaltnetze

- ◆ 1-zu-2-Demultiplexer
  - Funktionstabellen und Schaltbilder

s	x	y1	y0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

=>

s	y1	y0
0	0	x
1	x	0



- Funktionsgleichung

$$y_0 = x \cdot s'$$

$$y_1 = x \cdot s$$

- ◆ 1-zu-4-Demultiplexer
  - Funktionstabelle und Schaltbild

s1	s0	y3	y2	y1	y0
0	0	0	0	0	x
0	1	0	0	x	0
1	0	0	x	0	0
1	1	x	0	0	0

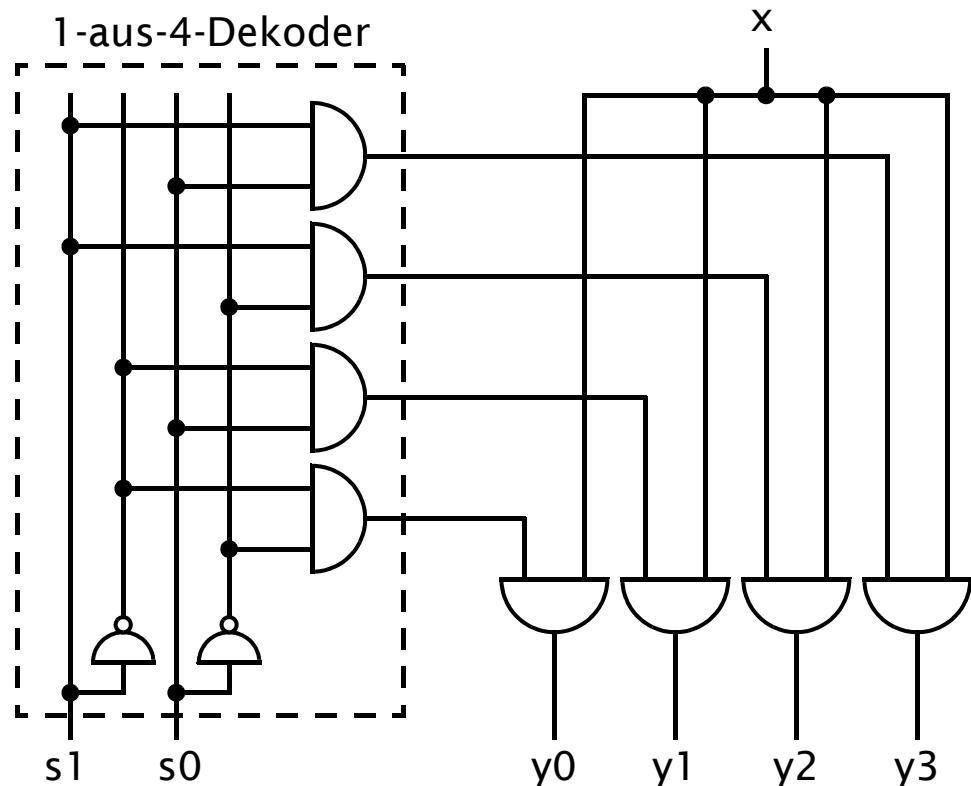
- Funktionsgleichungen

$$y_0 = x \cdot (\bar{s}_1 \cdot \bar{s}_0)$$

$$y_1 = x \cdot (\bar{s}_1 \cdot s_0)$$

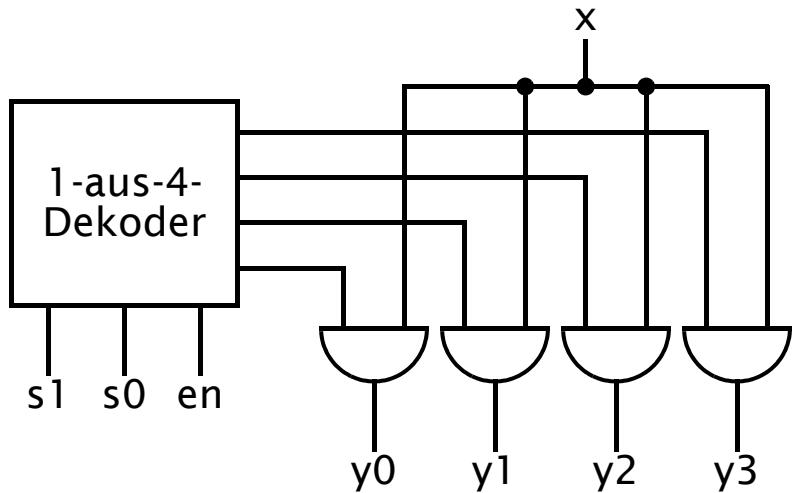
$$y_2 = x \cdot (s_1 \cdot \bar{s}_0)$$

$$y_3 = x \cdot (s_1 \cdot s_0)$$

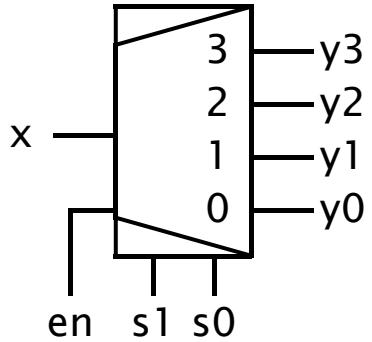


- ◆ 1-zu-4-Demultiplexer mit einem Enable-Eingang
  - Funktionstabelle und Schaltbild

en	s1	s0	y3	y2	y1	y0
0	-	-	0	0	0	0
1	0	0	0	0	0	x
1	0	1	0	0	x	0
1	1	0	0	x	0	0
1	1	1	x	0	0	0



- Funktionsgleichungen



- ◆ Entwicklungssatz von Shannon

- Dekomposition (Zerlegung) einer n-stelligen booleschen Funktion  $f$  hinsichtlich einer Variable  $e_i$ .
- Diese Dekomposition wird auch als IF-THEN-ELSE-Normalform bezeichnet.
- Realisierung boolescher Funktionen mit Multiplexern

$$f(e_0, e_1, \dots, e_i, \dots, e_{n-1}) =$$
$$\underbrace{e_i \cdot f(e_0, e_1, \dots, e_i=1, \dots, e_{n-1})}_{p_1} + \underbrace{e_i' \cdot f(e_0, e_1, \dots, e_i=0, \dots, e_{n-1})}_{p_0}$$
$$= e_i \cdot p_1 + e_i' \cdot p_0$$

mit zwei partiellen Funktionen  $p_1$  und  $p_0$  ohne die Variable  $e_i$

$$p_1(e_0, e_1, \dots, e_{i-1}, e_{i+1}, \dots, e_{n-1}) := f(e_0, e_1, \dots, e_i=1, \dots, e_{n-1})$$

$$p_0(e_0, e_1, \dots, e_{i-1}, e_{i+1}, \dots, e_{n-1}) := f(e_0, e_1, \dots, e_i=0, \dots, e_{n-1})$$

- ◆ Entwicklungssatz von Shannon

Für die boolesche Funktion  $f(a, b, c, d) = a \cdot b \cdot c' + b' \cdot (a' \cdot c + a \cdot d)$  ist eine Dekomposition hinsichtlich der Variable a zu bestimmen.

1. Definition der partiellen Funktionen  $p_0$  und  $p_1$  ohne Variable a:

$$p_0(b, c, d) := f(a=0, b, c, d)$$

$$p_1(b, c, d) := f(a=1, b, c, d)$$

2. Berechnung der partiellen Funktionen  $p_0$  und  $p_1$ :

$$p_0(b, c, d) := f(a=0, b, c, d) = 0 \cdot b \cdot c' + b' \cdot (0' \cdot c + 0 \cdot d) = b' \cdot c$$

$$p_1(b, c, d) := f(a=1, b, c, d) = 1 \cdot b \cdot c' + b' \cdot (1' \cdot c + 1 \cdot d) = b \cdot c' + b' \cdot d$$

3. Zusammenfassung der Resultate:

$$\begin{aligned} f(a, b, c, d) &= a' \cdot p_0(b, c, d) + a \cdot p_1(b, c, d) \\ &= a' \cdot (b' \cdot c) + a \cdot (b' \cdot d + b \cdot c') \end{aligned}$$

- ◆ Entwicklungssatz von Shannon

Für die Funktion  $f(a, b, c, d) = (a + b)' \cdot c + a \cdot (b + d)$  ist eine Dekomposition hinsichtlich der Variablen a und b zu bestimmen.

1. Def partieller Funktionen

$$p_0(c, d) := f(a=0, b=0, c, d)$$

$$p_1(c, d) := f(a=0, b=1, c, d)$$

$$p_2(c, d) := f(a=1, b=0, c, d)$$

$$p_3(c, d) := f(a=1, b=1, c, d)$$

2. Berechnung part. Funktionen

$$p_0(c, d) = c$$

$$p_1(c, d) = 0$$

$$p_2(c, d) = d$$

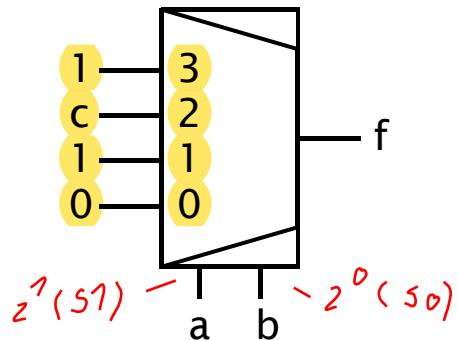
$$p_3(c, d) = 1$$

3. Zusammenfassung der Resultate

$$f(a, b, c, d) = \underline{\underline{a' \cdot b' \cdot c}} + \underline{\underline{a' \cdot b \cdot 0}} + \underline{\underline{a \cdot b' \cdot d}} + \underline{\underline{a \cdot b \cdot 1}}$$

- Analyse multiplexer-basierender Schaltnetze

Rekonstruktion/Minimierung boolescher Funktionen aus einstufigen Multiplexer-Schaltnetzen



Select-Eingänge:  $(s1, s0) := (a, b)$

Dateneingänge:

$$x0 := 0$$

$$x1 := 1$$

$$x2 := c$$

$$x3 := 1$$

- Funktionsgleichung des 1-aus-4-Multiplexers:

$$y = x0 \cdot (s1' \cdot s0') + x1 \cdot (s1' \cdot s0) + x2 \cdot (s1 \cdot s0') + x3 \cdot (s1 \cdot s0)$$

$$= 0 \cdot (a' \cdot b') + 1 \cdot (a' \cdot b) + c \cdot (a \cdot b') + 1 \cdot (a \cdot b)$$

$$= a' \cdot b + a \cdot b' \cdot c + a \cdot b \quad [A5, A9, A8]$$

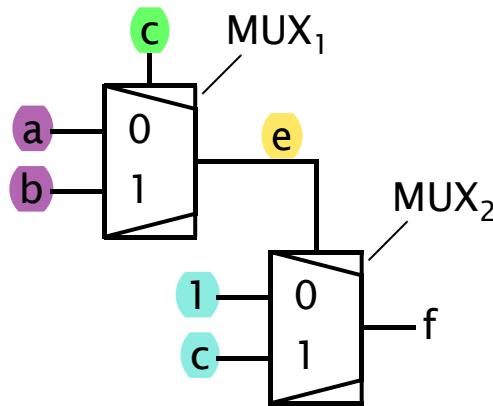
$$= b + a \cdot b' \cdot c \quad [A6]$$

$$= (b + b') \cdot (b + a \cdot c) \quad [A9, A8]$$

$$= b + a \cdot c$$

- Analyse multiplexer-basierender Schaltnetze

Rekonstruktion/Minimierung boolescher Funktionen aus mehrstufigen Multiplexer-Schaltnetzen



	Select	Daten	Daten	Ausgang
MUX <sub>1</sub>	(s) := c	x <sub>0</sub> := a	x <sub>1</sub> := b	e
MUX <sub>2</sub>	(s) := e	x <sub>0</sub> := 1	x <sub>1</sub> := c	f

- Funktionsgleichung des 1-aus-2-Multiplexers

$$y(s, x_0, x_1) := x_0 \cdot (s') + x_1 \cdot (s)$$

$$\text{MUX}_1 \quad e := y(a, b, c) = a \cdot c' + b \cdot c$$

$$\text{MUX}_2 \quad f := y(e, 1, c) = 1 \cdot e' + c \cdot e = (c' + e) \cdot (e' + c) = c'$$

- ◆ Analyse multiplexer-basierender Schaltnetze  
(Fortsetzung des Beispiels)

$$\begin{aligned}
 f &:= e' + c = (a \cdot c' + b \cdot c)' + c \quad [G8] \\
 &= (a \cdot c')' \cdot (b \cdot c)' + c \quad [Z+G9, G7] \\
 &= (a' + c) \cdot (b' + c') + c \quad [n \cdot A5] \\
 &= a' \cdot b' + a' \cdot c' + b' \cdot c + \underbrace{c \cdot c'}_0 + c \\
 &= a' \cdot b' + a' \cdot c' + c \\
 &= a' \cdot b' + a' + c \\
 &= \underline{a' + c}
 \end{aligned}$$

- ◆ Synthese multiplexer-basierender Schaltnetze basiert auf der Anwendung des Entwicklungssatzes von Shannon
- ◆ Realisierung boolescher Funktionen mit Multiplexern:
  1. Anzahl der Variablen in der booleschen Funktion = Anzahl der Select-Eingänge eines Multiplexers
  2. Anzahl der Variablen in der booleschen Funktion ist um 1 größer als die Anzahl der Select-Eingänge eines Multiplexers
  3. Anzahl der Variablen in der booleschen Funktion ist um mehr als 1 größer als die Anzahl der Select-Eingänge eines Multiplexers
    - Die Fälle 1) und 2) ergeben in der Realisierung einstufige Multiplexer-Schaltnetze.
    - Der Fall 3) ergibt in der Realisierung mehrstufige Multiplexer-Schaltnetze.

- ♦ Synthese multiplexer-basierender Schaltnetze, Fall 1)

1. Zuordnung boolescher Variablen zu Select-Eingängen eines 1-aus-n-Multiplexers

$$(s_{k-1}, s_{k-2}, \dots, s_1, s_0) := (e_{k-1}, e_{k-2}, \dots, e_1, e_0)$$

2. Bestimmung der Dateneingänge  $x_0, x_1, \dots, x_{n-1}$  mit  $n=2^k$

Beispiel: Realisierung  $g(a, b) = a + a' \cdot b$  mit einem 1-aus-4-Multiplexer:  $s_1=?$ ,  $s_0=?$ ,  $x_0=?$ ,  $x_1=?$ ,  $x_2=?$ ,  $x_3=?$

$$(s_1, s_0) := (a, b)$$

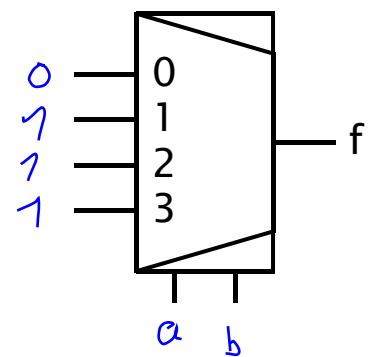


$$x_0 := g(a=0, b=0) = 0 + 0' \cdot 0 = 0$$

$$x_1 := g(a=0, b=1) = 0 + 0' \cdot 1 = 1$$

$$x_2 := g(a=1, b=0) = 1 + 1' \cdot 0 = 1$$

$$x_3 := g(a=1, b=1) = 1 + 1' \cdot 1 = 1$$



- ♦ Synthese multiplexer-basierender Schaltnetze, Fall 2)

Beispiel: Realisierung  $g(a, b, c) = a \cdot (b + c') + a' \cdot b \cdot c$  mit einem 1-aus-4-Multiplexer:  $s_1=?$ ,  $s_0=?$ ,  $x_0=?$ ,  $x_1=?$ ,  $x_2=?$ ,  $x_3=?$

1. Zuordnung boolescher Variablen zu zwei Select-Eingängen

2 aus 3 Variablen ergeben 3 Möglichkeiten:

$$(s_1, s_0) := (a, b) \text{ oder } (s_1, s_0) := (b, c) \text{ oder } (s_1, s_0) := (a, c)$$

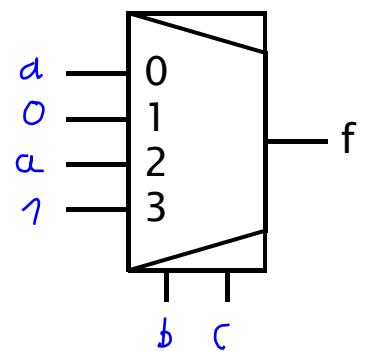
2. Bestimmung der Dateneingänge:

$$x_0: g(a, b=0, c=0) = a \cdot (0 + 0') + a' \cdot 0 \cdot 0 = a$$

$$x_1: g(a, b=0, c=1) = a \cdot (0 + 1') + a' \cdot 0 \cdot 1 = 0$$

$$x_2: g(a, b=1, c=0) = a \cdot (1 + 0') + a' \cdot 1 \cdot 0 = a$$

$$x_3: g(a, b=1, c=1) = a \cdot (1 + 1') + a' \cdot 1 \cdot 1 = 1$$



- ♦ Synthese multiplexer-basierender Schaltnetze, Fall 3)

Beispiel: Realisierung  $g(a, b, c) = a \cdot (b + c') + a' \cdot b \cdot c$  mit i.d.R. mehreren 1-aus-2-Multiplexern

1. Zuordnung boolescher Variablen zu dem Select-Eingang

ein 1-aus-2-Multiplexer hat nur einen Select-Eingang  $s_0$

die Auswahl einer aus 3 Variablen ergibt 3 Möglichkeiten:

(s) := (a) oder (s) := (b) oder (s) := (c)

2. Bestimmung der beiden Dateneingänge  $x_0$  und  $x_1$  des 1-aus-2-Multiplexers in der Abhängigkeit von der gewählten Zuordnung der booleschen Variable zum Select-Eingang.

$$x_0 := g(a=0, b, c) = 0 \cdot (b + c') + 0' \cdot b \cdot c = b \cdot c$$

$$x_1 := g(a=1, b, c) = 1 \cdot (b + c') + 1' \cdot b \cdot c = b + c'$$

- Wenn an den Dateneingängen keine einfachen Funktionen, die nur aus einer negierten oder nicht negierten Variable oder aus einer booleschen Konstante bestehen, sondern zusammengesetzte boolesche Ausdrücke entstehen, dann ist die Dekomposition für diese Ausdrücke zu wiederholen.

$$h(b, c) := g(a = 0, b, c) = b \cdot c$$

$$(S) := (b)$$

$$x_0 := h(b=0, c) = 0 \cdot c = 0$$

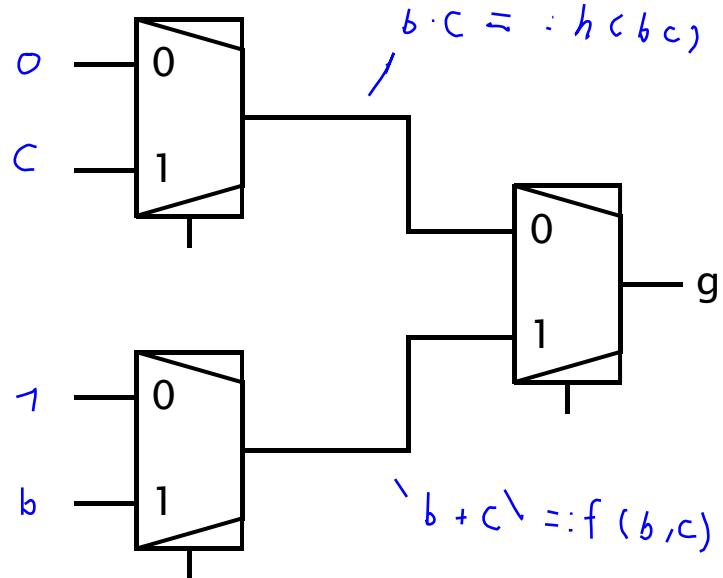
$$x_1 := h(b=1, c) = 1 \cdot c = c$$

$$f(b, c) := g(a = 1, b, c) = b + c$$

$$(S) := (c)$$

$$x_0 := f(c=0, b) = b + 0' = 1$$

$$x_1 := f(c=1, b) = b + 1' = b$$



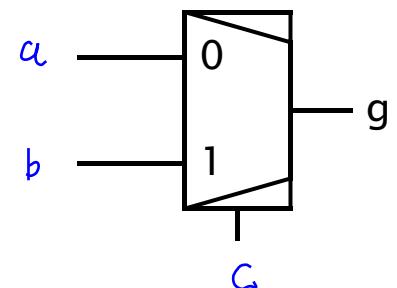
♦ Synthese multiplexer-basierender Schaltnetze, Fall 3)

Beispiel: Realisierung  $g(a, b, c) = a \cdot (b + c') + a' \cdot b \cdot c$  mit einem 1-aus-2-Multiplexer

1. Zuordnung boolescher Variable zum Select-Eingang  $(S)$ ;  $= (c)$
2. Bestimmung der beiden Dateneingänge  $x_0$  und  $x_1$  des 1-aus-2-Multiplexers in der Abhängigkeit von der gewählten Zuordnung der booleschen Variable zum Select-Eingang.

$$x_0 := g(a, b, c=0) = a \cdot (b + 0') + a' \cdot b \cdot 0 = a$$

$$\begin{aligned} x_1 &:= g(a, b, c=1) = a \cdot (b + 1') + a' \cdot b \cdot 1 \\ &= a \cdot b + a' \cdot b \quad [15, 19, 18] \\ &= b \end{aligned}$$



- ◆ Notwendigkeit
  - in bisherigen Betrachtungen kamen hauptsächlich digitale Systeme mit einer kleinen Anzahl (< 6) an Eingangsvariablen vor.
  - Somit war es möglich, diese Systeme tabellarisch, algebraisch oder graphisch zu beschreiben und anschließend zu synthetisieren.
  - Eine Funktionstabelle mit n Eingangsvariablen besteht aus  $2^n$  Einträgen, sofern die Funktion vollständig definiert ist.
  - Deshalb ist es kaum möglich, eine Funktionstabelle für z.B. 16 Eingangsvariablen aufzuschreiben ( $2^{16} = 65536$  Einträge)
- ◆ Schlußfolgerung
  - Man braucht eine andere Methode, mit der sich digitale Systeme auch mit einer großen Komplexität beschreiben und synthetisieren lassen => Schaltketten

- ◆ Schaltkette (kaskadierbares Schaltnetz, iterative Logik)
  - Eine Schaltkette ist eine Zusammenschaltung *gleichartiger* Schaltnetze (sog. Basiszelle, Grundschaltung) zu einer *kaskadenartigen*, in sich wiederholbaren Organisationsform.
  - Schaltketten spielen eine wichtige Rolle in der Digitaltechnik. Vor allem in der digitalen Signalverarbeitung dienen sie zur Realisierung von Rechenwerken mit arithmetisch-/logischen Operationen. Auf der Grundlage einer Basiszelle ist der Aufbau von Rechnewerken für mehrstellige Operationen problemlos möglich.
  - Schaltketten bilden die Basis für alle Operationen, die parallel auf digitale Wörter angewendet werden.
  - Im folgenden wird der Schwerpunkt vor allem auf arithmetische Operationen gelegt.

- ♦ Basiszelle einer Schaltkette

- unter einer Basiszelle versteht man ein Schaltnetz, das durch folgendes 6-Tupel beschrieben ist

$$B_z = (X, Y, U, V, f, g) \text{ mit}$$

X Eingabevektor

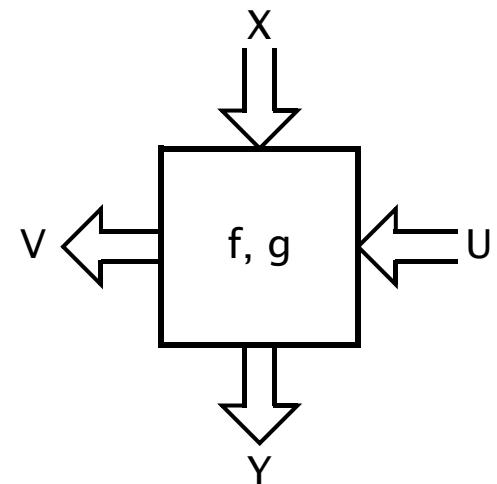
Y Ausgabevektor

U Übergabevektor

V Übergabevektor

f:  $X \times U \rightarrow Y$  Ausgabefunktion

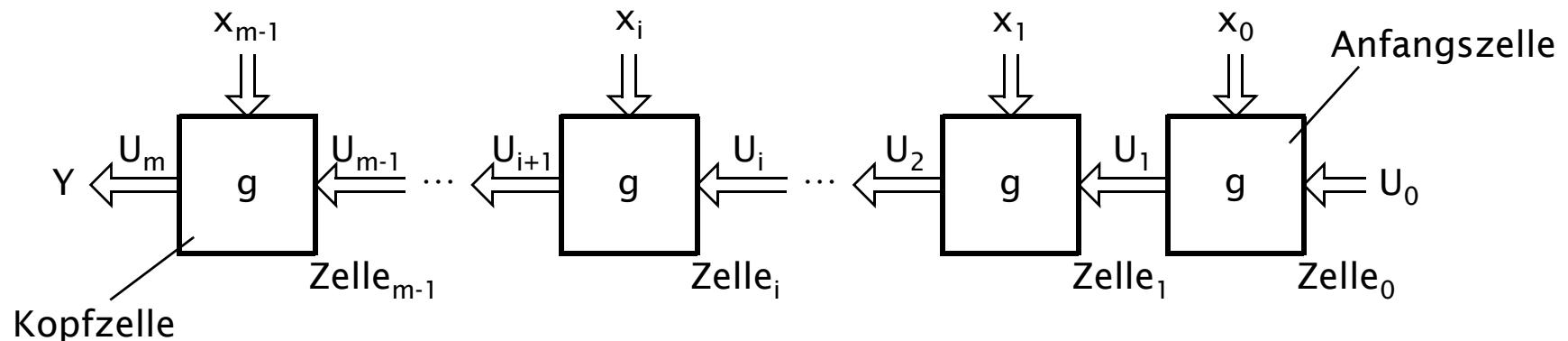
g:  $X \times U \rightarrow V$  Übergabefunktion



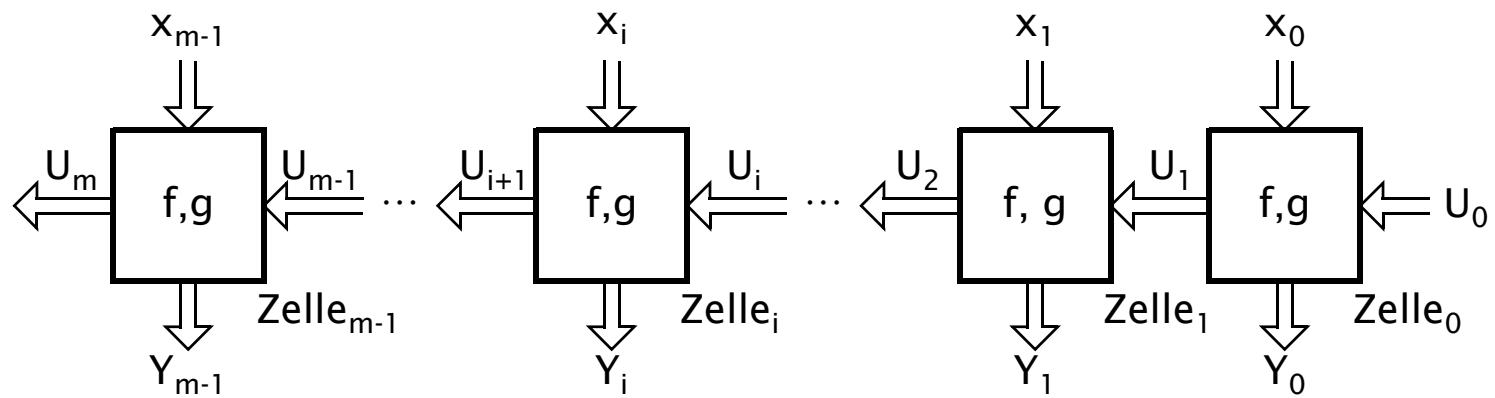
- Ein Schaltnetz mit n Eingangsvariablen, das auf n/k gleichartige Basiszellen mit Eingabevektoren der Länge k aufgeteilt ist, bezeichnet man als ein (*ortssequielles*) *k-iteratives* Schaltnetz.

# Schaltkette

- ◆ Schaltkette Typ 1

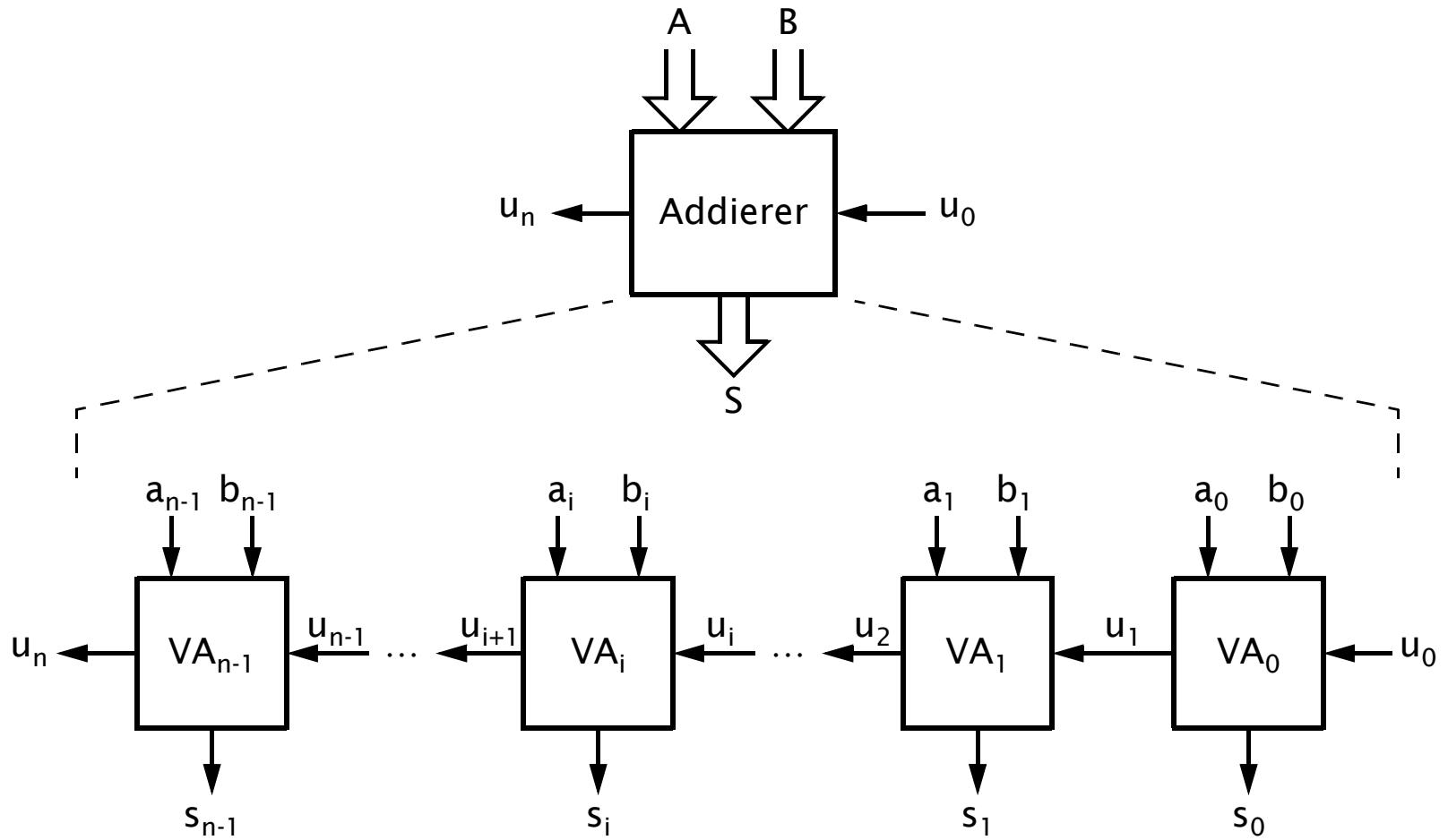


- ◆ Schaltkette Typ 2



- ◆ Systematischer Entwurf iterativer Schaltnetze
  1. Festlegung der Länge k des Eingabevektors X einer Basiszelle  
(Je kleiner der Wert k ist, desto höher die Gesamtverzögerung.)
  2. Bestimmung der Menge der Übergabevariablen durch funktionale Zerlegung des Eingabevektors mit dem gleichen Verhalten;  
=> Partitionierung der Schaltung in  $n/k$  gleichartige Basiszellen
  3. Ermittlung der Ausgangs- und Übergabefunktionen f und g für eine Basiszelle; Beschreibung auf einer abstrakten Ebene (z.B. mathematische Formulierung, Funktionstabelle)
  4. Minimierung der Ausgangs- und Übergabefunktionen mit bekannten Verfahren
  5. Initialisierung der Anfangszelle mit geeignet gewählten booleschen Konstanten (0, 1) und Vereinfachung der Anfangszelle
  6. Vereinfachung der Kopfzelle entsprechend der Ausgabe-/Übergabefunktionen

- ◆ n-Bit-Addierer für Dualzahlen



# Schaltkette

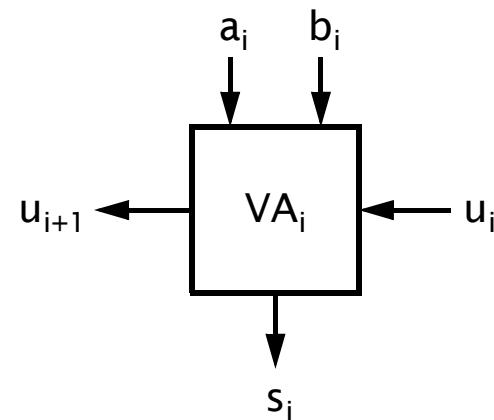
- ◆ Basiszelle: 1-Bit-Addierer (Volladdierer, VA)

- Schnittstelle

- Eingangsvariablen  $a_i$  und  $b_i$  für die Ziffern an der  $i$ -ten Position
- Übertragsvariable  $u_i$  aus der vorhergehenden  $(i-1)$ -ten Position
- Ausgangsvariable  $s_i$  für das Ergebnis der Addition
- Übertragsvariable  $u_{i+1}$  zu der nachfolgenden  $(i+1)$ -ten Position

- Funktionstabelle

$u_i$	$a_i$	$b_i$	$s_i$	$u_{i+1}$
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



- ◆ Basiszelle: 1-Bit-Addierer

Minimierung mit einem KV-Diagramm

$a_i b_i$	00	01	11	10
$u_i$	0			
	1			

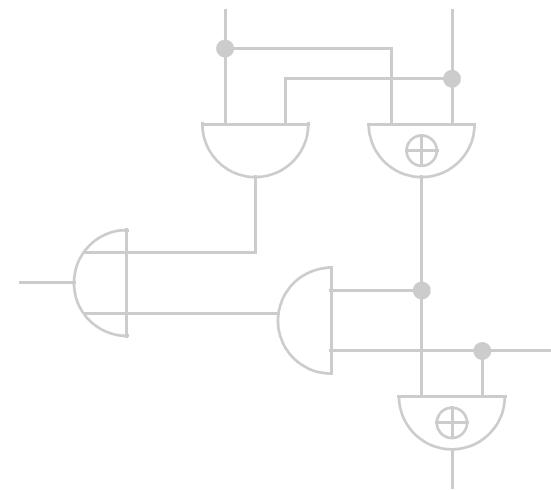
$a_i b_i$	00	01	11	10
$u_i$	0			
	1			

Ausgangsfunktion:

$$s_i := f(u_i, a_i, b_i) =$$

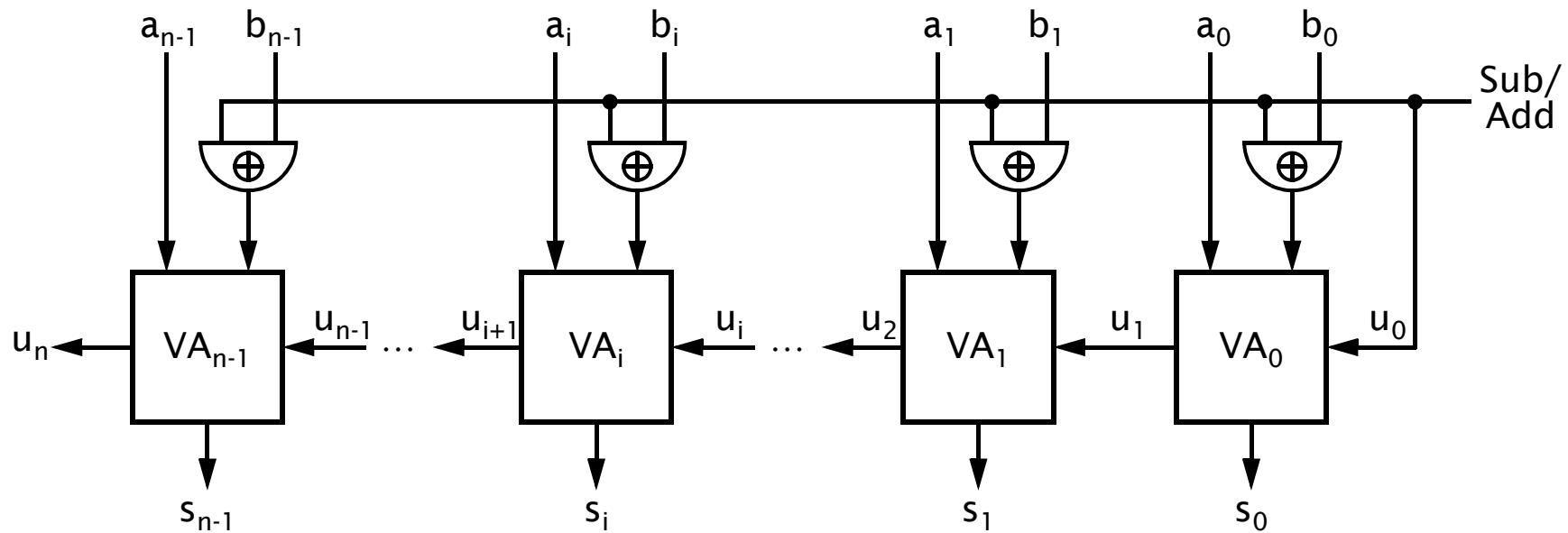
Übergabefunktion:

$$u_{i+1} := g(u_i, a_i, b_i) =$$



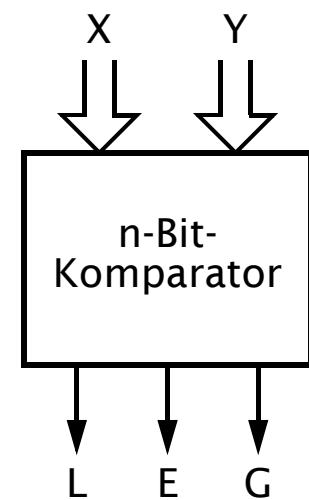
- ◆ n-Bit-Addierer/Subtrahierer

- Erweiterung des n-Bit-Addierers um Invertierlogik und um ein Auswahlsignal Sub/Add zur Bestimmung der Operation
  - Addition wenn Sub/Add = 0
  - Subtraktion wenn Sub/Add = 1

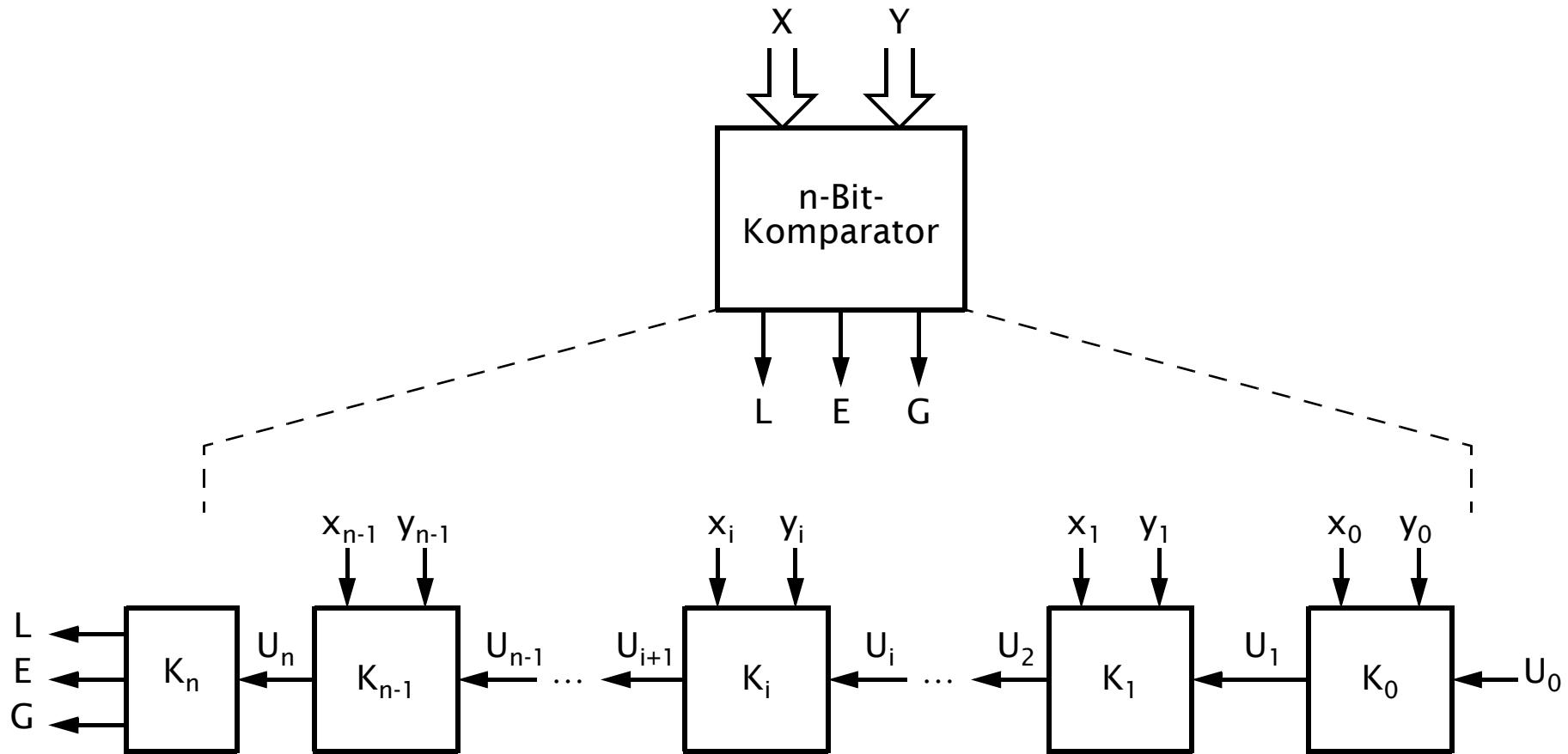


# Schaltkette

- ◆ n-Bit-Komparator für vorzeichenlose (nicht negative) Dualzahlen
  - Ein n-Bit-Komparator ermöglicht den Vergleich von zwei n-stelligen nicht negativen Dualzahlen X und Y.
  - Er trifft Entscheidung darüber, in welcher Relation zueinander die beiden an seinen Eingängen anliegenden Dualzahlen stehen.
  - Es ergeben sich drei Fälle:
    - gleich (Equal)  $E := (X = Y)$
    - kleiner als (Less than)  $L := (X < Y)$
    - größer als (Greater than)  $G := (X > Y)$



- ◆ n-Bit-Komparator für vorzeichenlose Dualzahlen



- ◆ Analyse sämtlicher relevanten Fälle für die Übergangs-funktion

$$X_p = (x_{i-1}, x_{i-2}, \dots, x_1, x_0) \text{ und } Y_p = (y_{i-1}, y_{i-2}, \dots, y_1, y_0)$$

Fall a) i-te Stelle mit  $X_p < Y_p$


Fall b) i-te Stelle mit  $X_p = Y_p$


Fall c) i-te Stelle mit  $X_p > Y_p$


- ◆ n-Bit-Komparator für vorzeichenlose Dualzahlen
  - symbolisch codierte Werte der Übertragsvariablen  $U_i$ , die die i-te Basiszelle von der vorherigen (i-1)-ten Basiszelle erhält:

$$X_p = (x_{i-1}, x_{i-2}, \dots, x_1, x_0) \quad Y_p = (y_{i-1}, y_{i-2}, \dots, y_1, y_0)$$

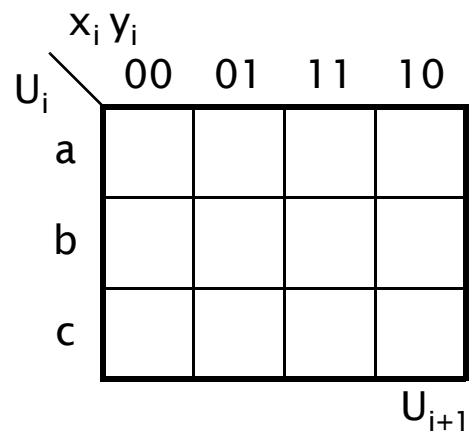
$U_i$	Code
$X_p < Y_p$	a
$X_p = Y_p$	b
$X_p > Y_p$	c

- symbolisch codierte Werte der Übertragsvariablen  $U_{i+1}$ , die die i-te Basiszelle an die darauffolgende (i+1)-Basiszelle weiter gibt:

$$X_{p+1} = (x_i, x_{i-1}, \dots, x_1, x_0) \quad Y_{p+1} = (y_i, y_{i-1}, \dots, y_1, y_0)$$

$U_{i+1}$	Code
$X_{p+1} < Y_{p+1}$	a
$X_{p+1} = Y_{p+1}$	b
$X_{p+1} > Y_{p+1}$	c

- ♦ Aufstellung der Funktionstabelle und des KV-Diagramms mit symbolisch codierten Fällen für die i-te Stelle:



# Schaltkette

- ♦ binäre Codierung der symbolischen Fälle und Auflösung des KV-Diagramms

$$(a) := (00)_2, (b) := (01)_2, c := (11)_2$$

$$U_i = (u_0 u_1)_i \quad U_{i+1} = (u_0 u_1)_{i+1}$$

	$x_i$	$y_i$	00	01	11	10
$(u_0 u_1)_i$	00	00				
	01	01				
	11	11				
	10	10				

$(u_0 u_1)_{i+1}$

	$x_i$	$y_i$	00	01	11	10
$(u_0 u_1)_i$	00	00				
	01	01				
	11	11				
	10	10				

$(u_0)_{i+1}$

	$x_i$	$y_i$	00	01	11	10
$(u_0 u_1)_i$	00	00				
	01	01				
	11	11				
	10	10				

$(u_1)_{i+1}$

- ♦ Auslesen der minimierten Übergangsgleichungen aus den KV-Diagrammen und ggf. weitere Vereinfachungen

$$(u_0)_{i+1} = x_i \cdot y'_i + (u_0)_i \cdot x_i + (u_0)_i \cdot y'_i = x_i \cdot y'_i + (u_0)_i \cdot (x_i + y'_i)$$

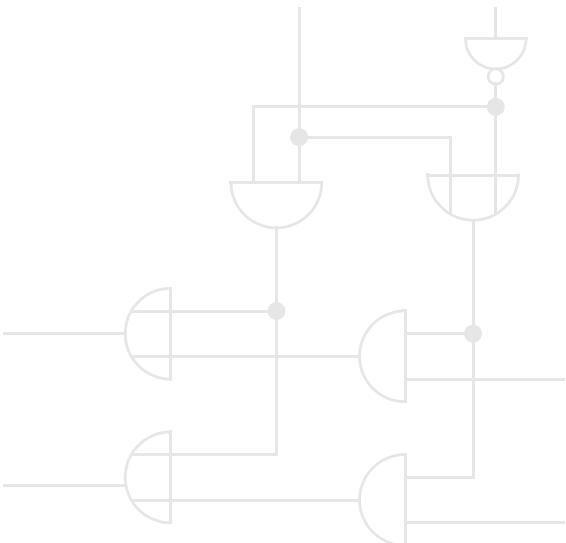
$$(u_1)_{i+1} = x_i \cdot y'_i + (u_1)_i \cdot x_i + (u_1)_i \cdot y'_i = x_i \cdot y'_i + (u_1)_i \cdot (x_i + y'_i)$$

mit

$$f_i = x_i \cdot y'_i \quad \text{und} \quad g_i = x_i + y'_i$$

$$(u_0)_{i+1} = f_i + (u_0)_i \cdot g_i$$

$$(u_1)_{i+1} = f_i + (u_1)_i \cdot g_i$$



- ◆ Bestimmung der Initialisierungswerte für die Übergangsvariablen in der Anfangszelle

$$(u_0)_0 =$$

$$(u_1)_0 =$$

- ◆ Bestimmung der Ausgangsfunktionen für die Kopfzelle

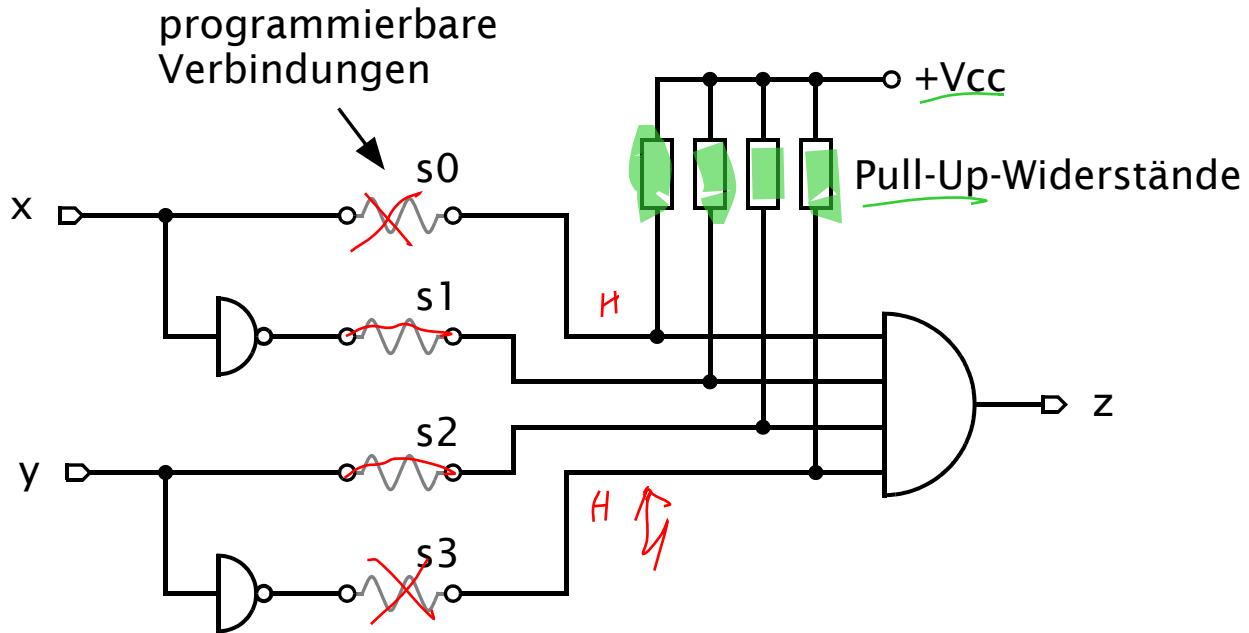
$$L =$$

$$E =$$

$$G =$$

- ◆ Realisierung digitaler Schaltungen
  - fest verdrahtete Lösung mit diskreten Elementen (einzelne Gatter)
  - fest verdrahtete Lösung mit Standard-Bausteinen (integrierte Gatter, Multiplexer)
  - fest verdrahtete Lösung mit kunden-/applikationsspezifischen integrierten Schaltungen (ASIC)
  - flexible Lösung mit programmierbaren Logikbausteinen (PLD: PLA, PAL, CPLD, FPGA)
- ◆ Programmierbare Logikbausteine
  - Realisierung boolescher Funktion als zweistufiges Schaltnetz in disjunktiver oder konjunktiver Normalform
  - integrierte Schaltungen mit festen, vordefinierten Grundstrukturen (UND-/ODER-Matrizen) und variablen, konfigurierbaren/programmierbaren Verbindungen
  - relativ leichte Änderbarkeit eines Entwurfes, oft im System (re)programmierbar

- ◆ Grundstruktur eines programmierbaren UND-Gatters

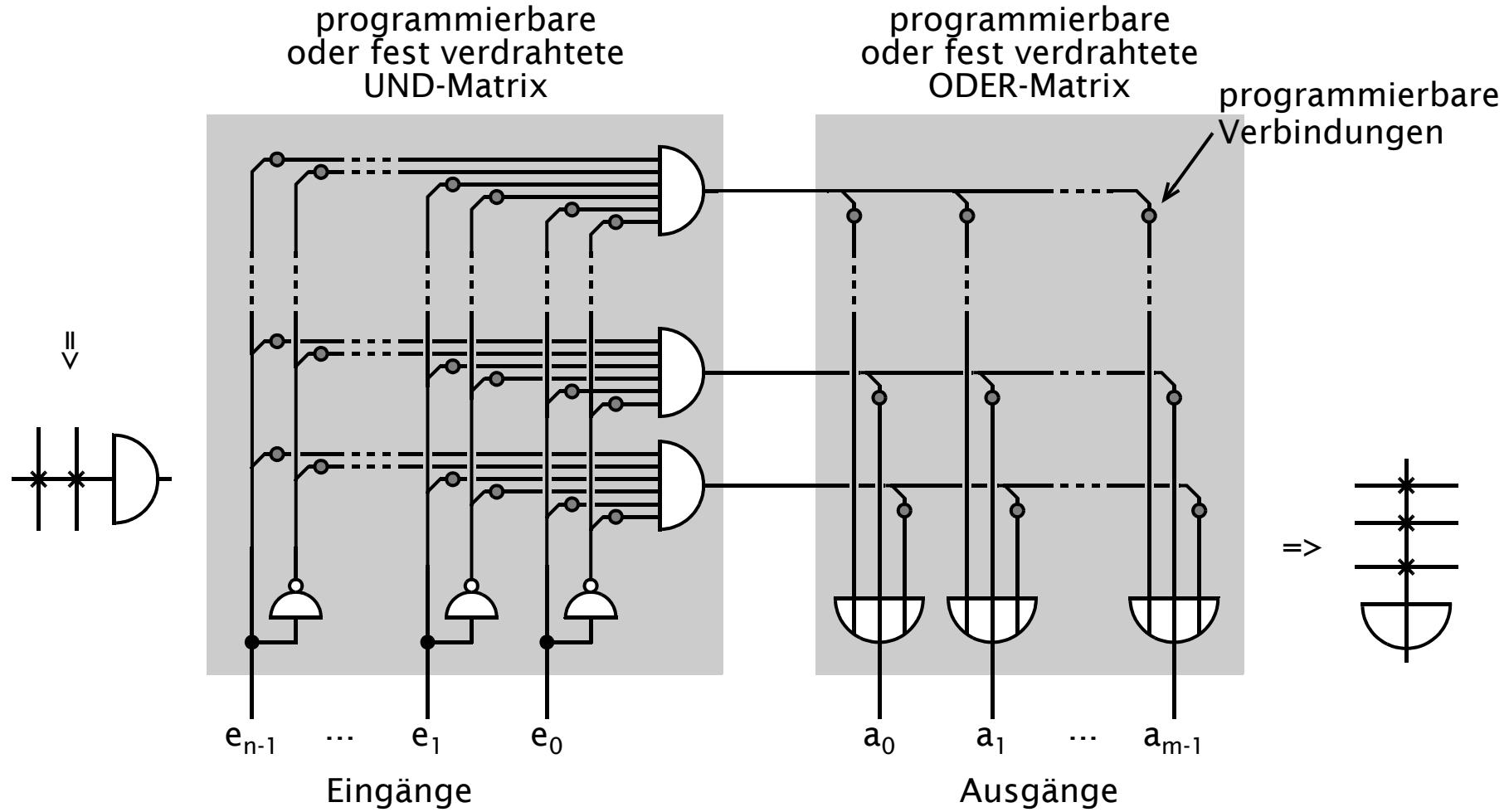


$$\text{Grundfunktion: } z(x, y) = x \cdot x' \cdot y \cdot y' = 0$$

$$f(x, y) = x' \cdot y \Rightarrow z = x \cdot x' \cdot y \cdot y' = s_0 \cdot x \cdot s_1 \cdot x' \cdot s_2 \cdot y \cdot s_3 \cdot y'$$

$\Rightarrow s_0$  wird geöffnet,  $s_1$  bleibt bestehen  
 $s_2$  bleibt bestehen,  $s_3$  wird geöffnet

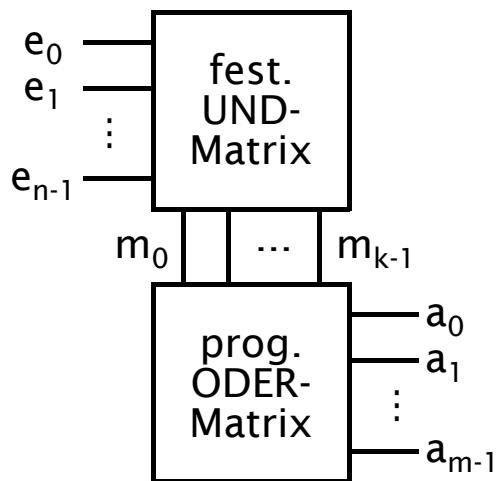
- ◆ Grundstrukturen programmierbarer Logikbausteine



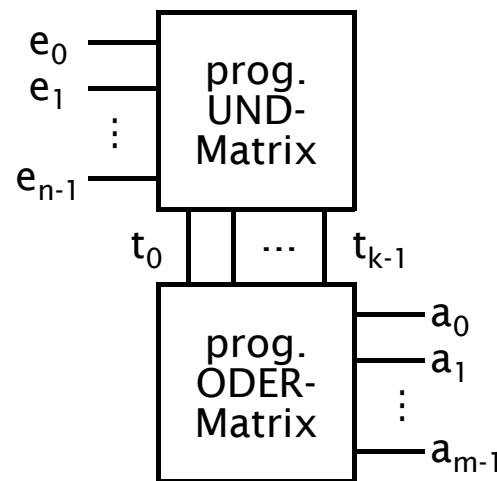
- ◆ Grundstrukturen programmierbarer Logikbausteine

## PLD (Programmable Logic Device)

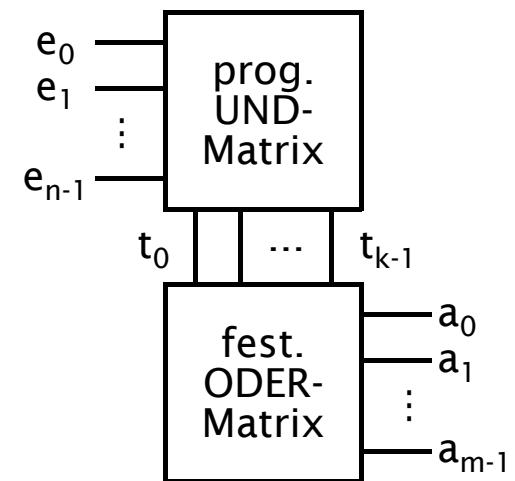
PROM  
(Programmable Read  
Only Memory)



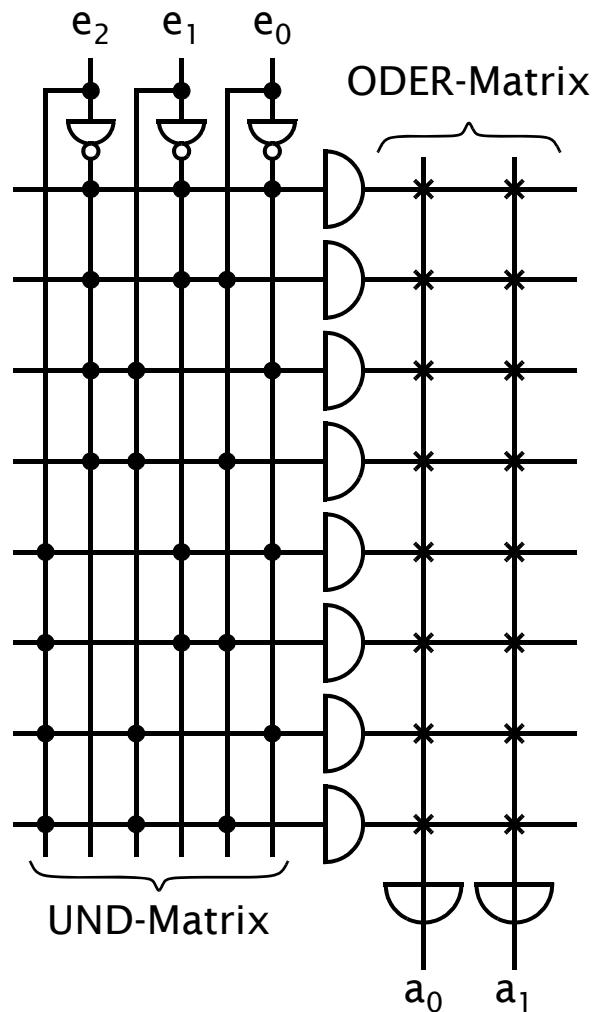
PLA  
(Programmable  
Logic Array)



PAL  
(Programmable  
Array Logic)



- ♦ Realisierung mit  $2^N \times M$ -Bit-PROM
  - fest verdrahtete UND-Matrix, ausgelegt als 1-aus-N-Dekoder
  - frei programmierbare ODER-Matrix für Summenterme (hier 2 Summenterme mit bis zu 8 Produkttermen pro Summenterm)
  - unmittelbare Umsetzung einer vollständig definierten Funktionstabelle, „don't care“-Einträge müssen aufgelöst werden
  - mit einem PROM-Baustein mit  $2^N$  Worten à M Bits lassen sich beliebige boolesche Funktion mit N Eingängen und M Ausgängen realisieren
  - Parallel-Addierer/-Multiplizierer



- Realisierung mit  $8 \times 2$ -Bit-PROM

$$f(a, b, c) = a \cdot (b + c') + b \cdot (a \oplus c)$$

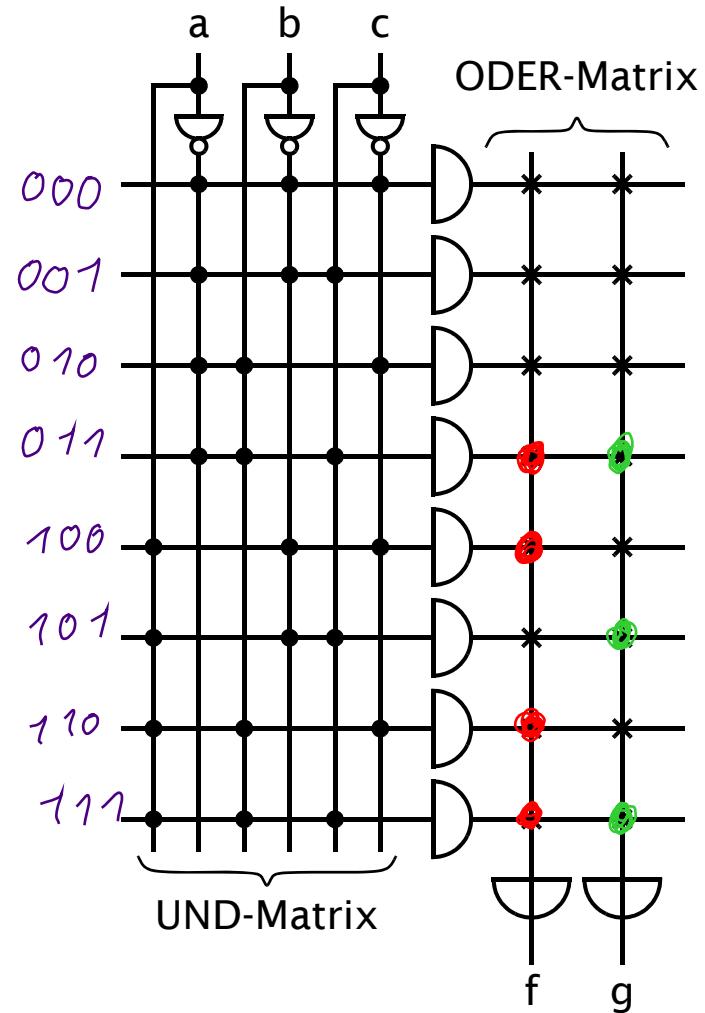
$$g(a, b, c) = c \cdot (a + b)$$

1. Umwandlung der Funktion(en) in kanonische disjunktive Normalform(en)

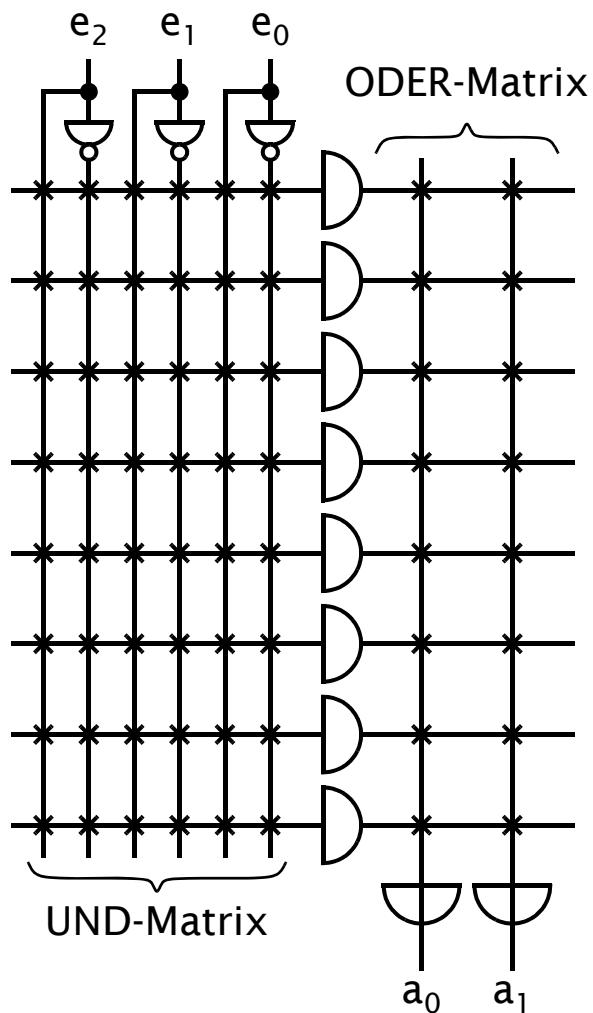
$$\underline{f(a,b,c)} = \sum (111, 110, 100, 011)$$

$$\underline{g(a,b,c)} = \sum (111, 011, 101)$$

2. Markierung programmierbarer Verbindungen in der ODER-Matrix



- ♦ Realisierung mit PLA (Programmable Logic Array)
  - frei programmierbare UND-Matrix für Produktterme (hier 8 Produktterme mit bis zu 3 (6) Variablen pro Produktterm)
  - frei programmierbare ODER-Matrix für Summenterme (hier 2 Summenterme mit bis zu 8 Produktterme pro Summenterm)
  - mit PLAs können beliebige boolesche Funktionen mit N Eingängen und M Ausgängen implementiert werden, sofern die Gesamtzahl der Produktterme im PLA ausreichend ist
  - Steuertabellen in programmierbaren Steuerwerken



- ♦ Realisierung mit PLA  
(Programmable Logic Array)

$$f(a, b, c) = a \cdot (b + c') + b \cdot (a \oplus c)$$

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

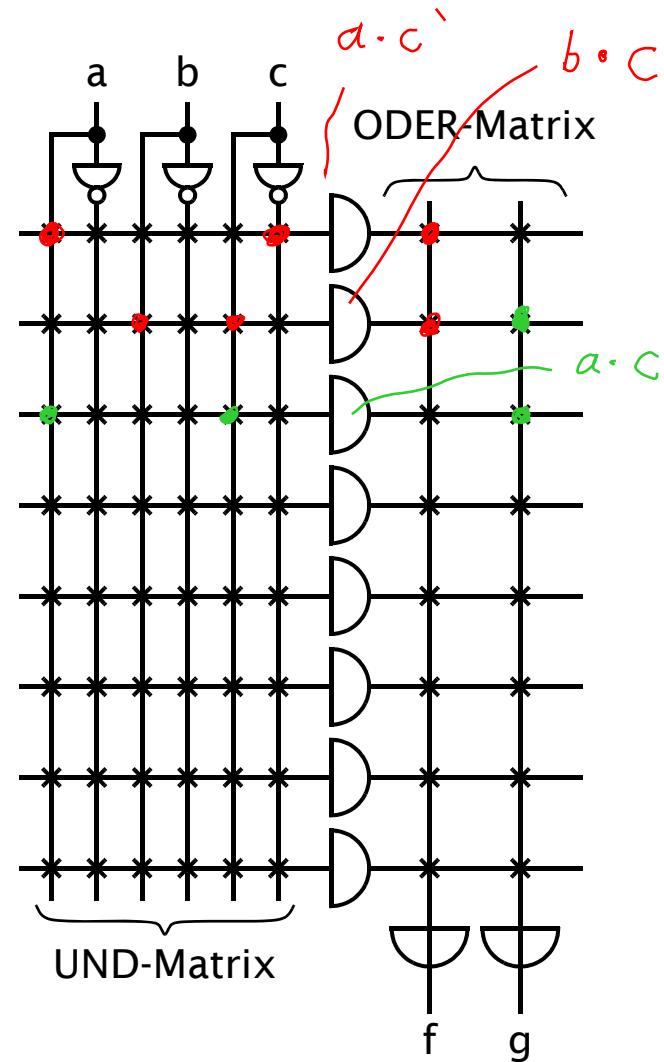
1. Minimierung der Funktion(en) und Umwandlung in disjunktive Normalform(en)

$$f(a, b, c) = a \cdot c' + b \cdot c$$

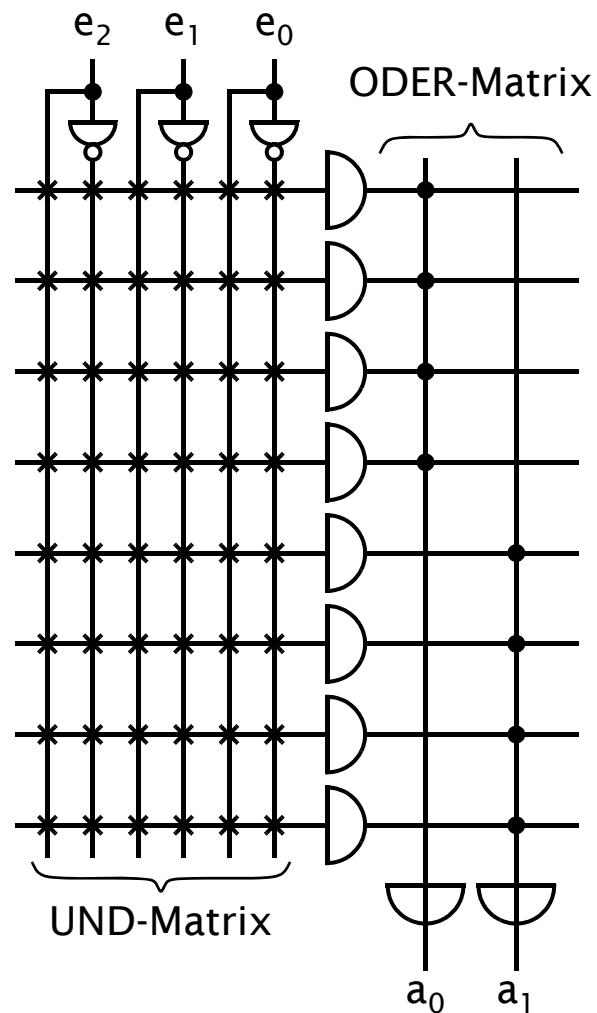
$$g(a, b, c) = a \cdot c + b \cdot c$$

2. Ausnutzung gemeinsamer (Teil-) Terme

3. Markierung programmierbarer Verbindungen in der UND/ODER-Matrix



- ♦ Realisierung mit PAL (Programmable Array Logic)
  - frei programmierbare UND-Matrix für Produktterme (hier 8 Produktterme mit bis zu 3 (6) Variablen pro Produktterm)
  - fest verdrahtete ODER-Matrix für Summenterme, Anzahl der Summenterme fest vorgegeben und zugeordnet, (hier 2 Summenterme mit bis zu 4 Produktterme pro Summenterm)
  - mit einem PAL kann jede (minimierte) Summe von Produkttermen realisiert werden, sofern die Anzahl der Produktterme pro Summenterm ausreicht
  - zahlreiche integrierte Bausteine



- Realisierung boolescher Funktionen mit PAL (Programmable Array Logic)

$$f(a, b, c) = a \cdot (b + c') + b \cdot (a \oplus c)$$

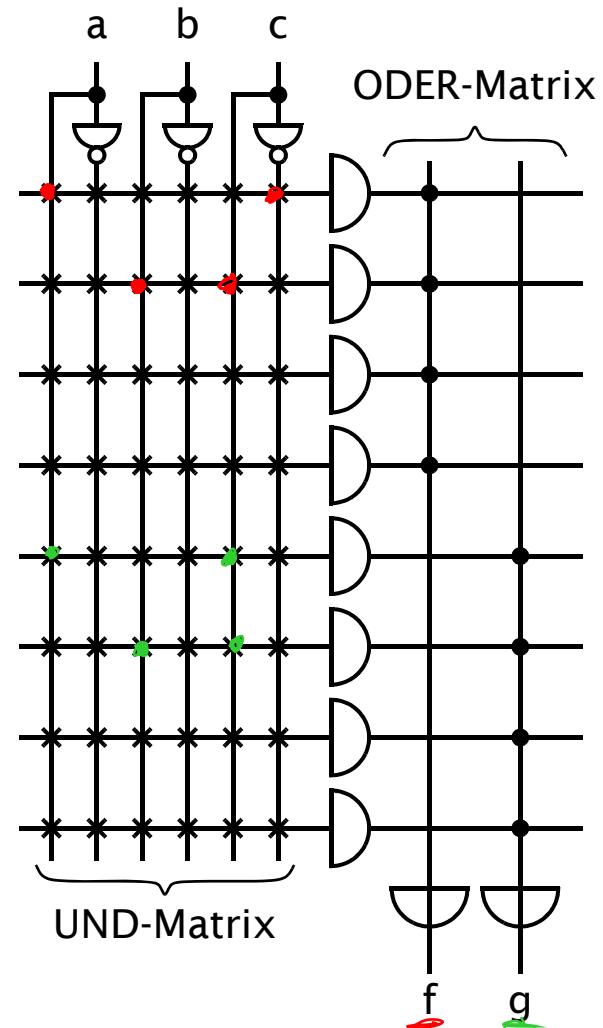
$$g(a, b, c) = c \cdot (a + b)$$

- Minimierung der Funktion(en) und Umwandlung in disjunktive Normalform(en)

$$f(a, b, c) = a \cdot c' + b \cdot c$$

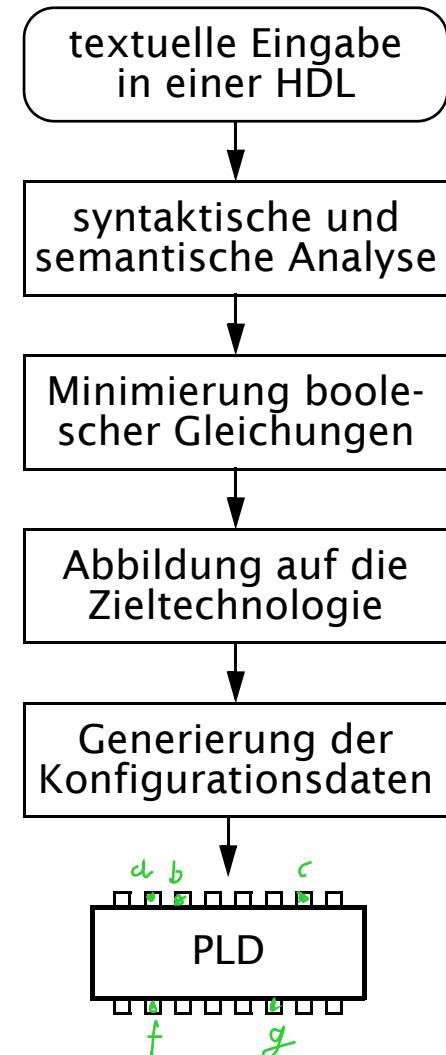
$$g(a, b, c) = a \cdot c + b \cdot c$$

- Markierung programmierbarer Verbindungen in der UND-Matrix



# Logikbausteine

- ◆ Computergestützter Entwurf
  - Ausgangspunkt ist eine funktionelle Beschreibung in einer Hardwarebeschreibungssprache (wie z.B. ABEL, AHDL, VHDL, Verilog, ...)
  - Minimierung boolescher Gleichungen mit optionaler Dekomposition und Faktorisierung
  - Abbildung auf die Zieltechnologie mit Berücksichtigung von PINs-Zuordnung
  - Generierung der Konfigurationsdaten in einem Format für ein Programmiergerät mit anschließender Programmierung



- ◆ 1-Bit-Volladdierer, notiert in VHDL

-- Schnittstelle

```
ENTITY volladdierer IS
```

```
  PORT(A: IN  bit;      -- Dateneingang
        B: IN  bit;      -- Dateneingang
        C: IN  bit;      -- Carry-Eingang
        S: OUT bit;     -- Summenausgang
        U: OUT bit);    -- Carry-Ausgang
```

```
END volladdierer;
```

-- Modellierung mit booleschen Gleichungen

```
ARCHITECTURE verhalten OF volladdierer IS
```

```
  SIGNAL T: bit;
```

```
BEGIN
```

```
  T <= A XOR B;
```

```
  S <= T XOR C;
```

```
  U <= (C AND T) OR (A AND B);
```

```
END verhalten;
```

## ♦ Logikfamilie

- Die mathematische Beschreibung der Ausgangsvariablen in Abhängigkeit von Eingangsvariablen erfolgt mit Hilfe boolescher Gleichungen.
- Zur Umsetzung dieser Gleichungen in eine funktionierende Schaltung stehen Gatter mit entsprechender Funktionalität (AND, OR, NOT, NAND, EXOR usw.) zur Verfügung.
- In der Digitaltechnik bezeichnet man diese Gatter als Logikfamilie.
- Eine Logikfamilie zeichnet sich durch Kenndaten, statische und dynamische Eigenschaften aus.
- Zu den gängigen Logikfamilien gehören:
  - CMOS Complementary Metal Oxide Semiconductor  $3.5 \text{ ns}$  |  $10 \text{ nW}$
  - TTL Transistor-Transistor-Logic  $10 \text{ ns}$  |  $10 \text{ mW}$
  - LSTTL Low-Power-Schottky-TTL  $8 \text{ ns}$  |  $2 \text{ mW}$
  - HC(T) High-Speed-CMOS  $8 \text{ ns}$  |  $25 \text{ nW}$
  - ECL Emitter Coupled-Logic  $1 \text{ ns}$  |  $25 \text{ mW}$

- ◆ Statische und dynamische Eigenschaften von Gattern
  - primär sehr stark von der Logikfamilie abhängig
  - innerhalb einer Logikfamilie stark von Betriebsbedingungen abhängig (z.B. Temperatur, Versorgungsspannung)
- ◆ Für die Schaltungsanalyse und -synthese sind folgende Kenndaten von Gattern relevant:
  - Betriebsparameter:
    - die Versorgungsspannung
    - der Betriebstemperaturbereich
    - die Leistungsaufnahme
  - zulässige Pegelbereiche und Störabstände,
  - die Belastbarkeit (Lastfaktoren Fan-In/Fan-Out),
  - das Zeitverhalten (Anstiegs- und Abfallzeiten, Verzögerung)

- ◆ Betriebsparameter (operating ranges)
  - LSTTL-Logikfamilie (Low-Power-Schottky TTL), Logikfamilie mit zur Zeit der größten Typenvielfalt, Industrie-Standard

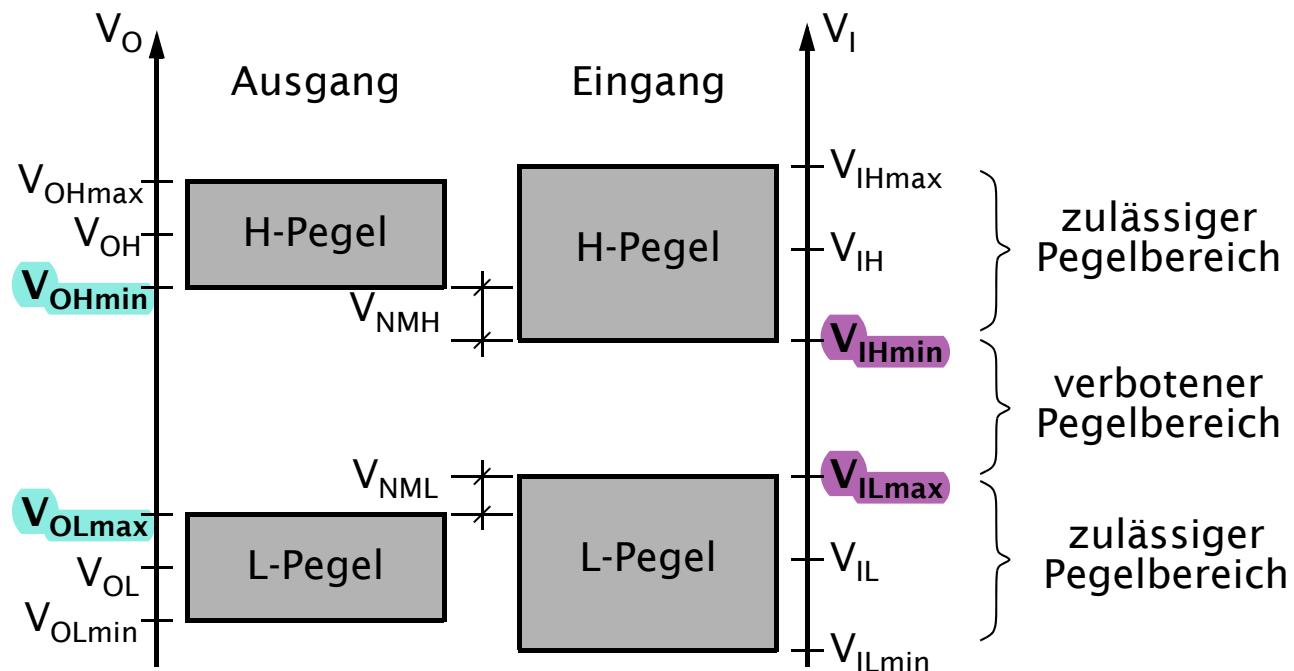
Symbol	Betriebsparameter	Einheit	Min.	Typ.	Max.
$V_{CC}$	Versorgungsspannung	V	4.75	5.00	5.25
$T_C$	Betriebstemperaturbereich	°C	0	+25	+70
$T_M$	Betriebstemperaturbereich	°C	-55	+25	+125
P	Leistungsaufnahme je Gatter	mW		2	

- ◆ Störabstand
  - Bei zwei in einer Reihe geschalteten Gattern muß sichergestellt werden, daß das Ausgangssignal des ersten Gatters vom zweiten Gatter richtig gedeutet wird.
  - Chip-Hersteller geben für ihre Logikfamilien den sog. garantierten statischen Störabstand an, der auch unter den ungünstigsten Betriebsbedingungen eingehalten wird.

- Der Störabstand ist eine Spannung, um die eine Ausgangsspannung variieren darf, ohne daß ein angeschlossener Eingang der selben Logikfamilie in den verbotenen Pegelbereich gelangt.
- Die H- und L-Pegel sind aus Gründen der Kompatibilität und Toleranzabdeckungen nicht als feste Werte sondern als Bereiche ausgelegt, und sind für Eingänge sowie Ausgänge unterschiedlich groß.
- Der zulässige Pegelbereich für Eingänge ist größer ausgelegt, als für Ausgänge, damit Herstellungstoleranzen, Parameterschwankungen und Laständerungen nicht zu Logikfehlern führen.
- Die Störfestigkeit (Robustheit) einer Logikfamilie ist von den Pegeldifferenzen zwischen dem H-Pegel und dem L-Pegel abhängig und in den verschiedenen Logikfamilien sehr unterschiedlich.

# Kenndaten

- ◆ Definition von Spannungen und Spannungsbereichen für die logischen H- und L-Pegeln



# Kenndaten

- ◆ **Eingangsspannungen**  
für die LSTTL-Baureihe mit  $V_{CC} = +5V$ ,  $\vartheta = 25^\circ C$

$V_{IHmax}$	maximale Eingangsspannung bei H-Pegel	$V_{CC} + 0.5 V$
$V_{IHmin}$	minimale Eingangsspannung bei H-Pegel	2.0 V
$V_{ILmax}$	maximale Eingangsspannung bei L-Pegel	0.8 V
$V_{ILmin}$	minimale Eingangsspannung bei L-Pegel	-0.5 V

- ◆ **Ausgangsspannungen**

$V_{OHmax}$	maximale Ausgangsspannung bei H-Pegel	$V_{CC}$
$V_{OHmin}$	minimale Ausgangsspannung bei H-Pegel	2.7 V
$V_{OLmax}$	maximale Ausgangsspannung bei L-Pegel	0.4 V
$V_{OLmin}$	minimale Ausgangsspannung bei L-Pegel	0.0 V

- ◆ **Störabstände**

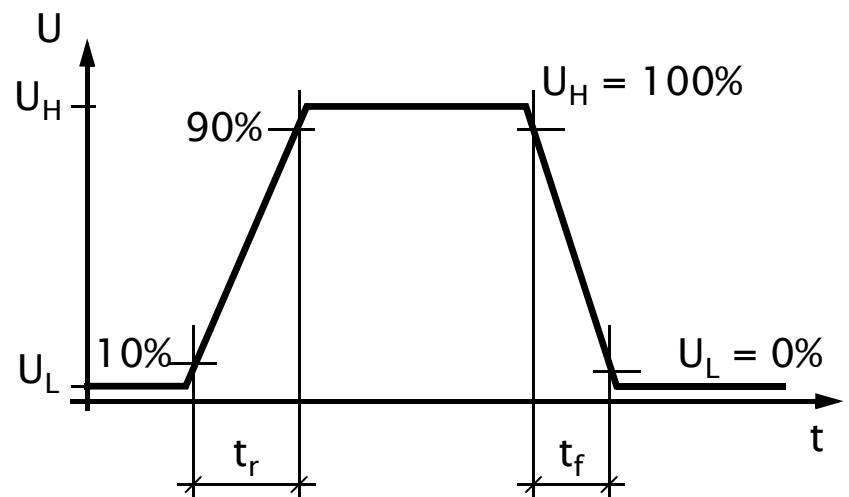
$V_{NMH}$	statischer Störabstand bei H-Pegel	$V_{OHmin} - V_{ILmin}$	0.7 V
$V_{NML}$	statischer Störabstand bei L-Pegel	$V_{ILmax} - V_{OLmax}$	0.4 V

# Kenndaten

- ◆ Anstiegs- und Abfallzeiten
  - Anstiegs- und Abfallzeiten werden zwischen 10% und 90% des Absolutwert-Pegels eines Signals gemessen und angegeben.

$t_r$  Anstiegszeit (rise time) einer Signalflanke ist die Zeitspanne, die ein Signal beim Übergang von L-Pegel auf H-Pegel benötigt.

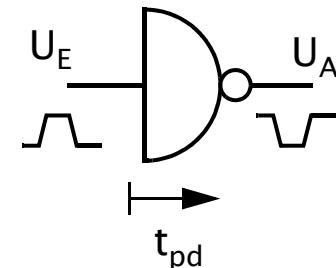
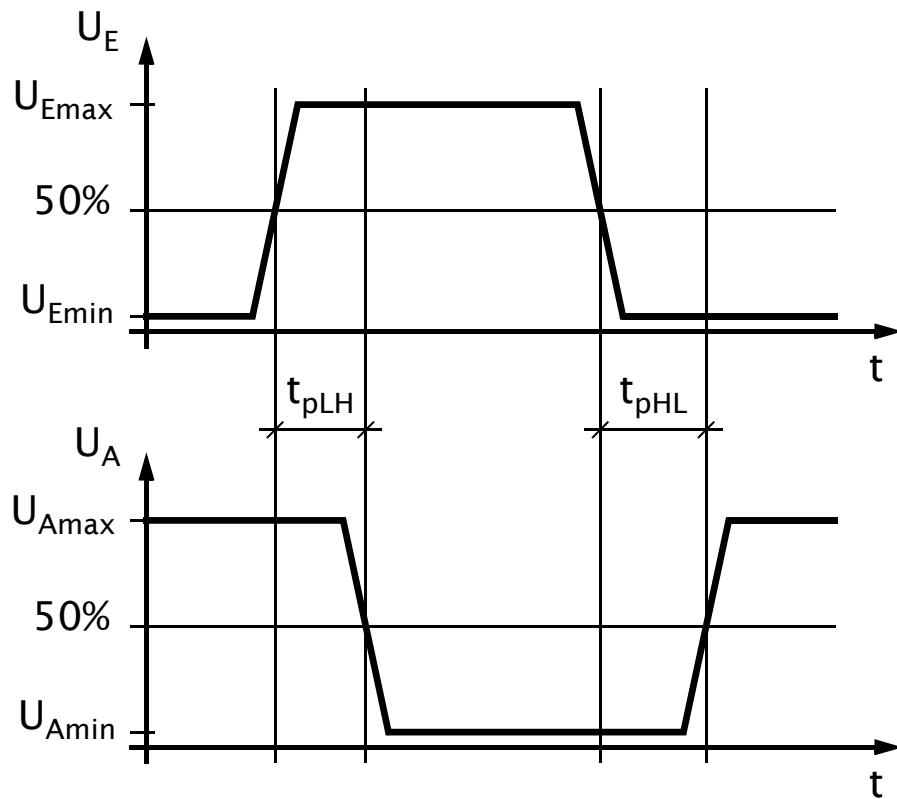
$t_f$  Abfallzeit (fall time) einer Signalflanke ist die Zeitspanne, die ein Signal beim Übergang von H-Pegel auf L-Pegel benötigt.



# Kenndaten

## ◆ Verzögerung

- Die Verzögerung ( $t_{pd}$ , propagation delay) beschreibt die Durchlaufzeit einer Signalflanke durch einen digitalen Baustein.



Verzögerungen sollte jeweils für die ansteigende Signalfalte ( $t_{pLH}$ ) und die abfallende Signalfalte ( $t_{pHL}$ ) gemessen und angegeben werden. SN 74 LS08

TYP:  $t_{PLH} = 8 \text{ ns}$ ,  $t_{pHL} = 16 \text{ ns}$   
 Max:  $t_{PLH} = 75 \text{ ns}$ ,  $t_{pHL} = 8 \text{ ns}$

# Kenndaten

- Man definiert die durchschnittliche Verzögerung  $t_{pd}$  als den arithmetischen Mittelwert aus Anstiegsverzögerung  $t_{pLH}$  und Abfallverzögerung  $t_{pHL}$

$$t_{pd} = (t_{pLH} + t_{pHL})/2$$

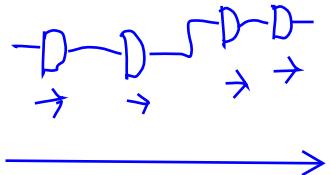
*SN74LS08: Typ:  $t_{pd} = 9 \text{ ns}$ , Max:  $t_{pd} = 17,5 \text{ ns}$*

- Die mittlere Verzögerung  $t_{pd}$  ist ebenfalls in ihrem Kehrwert als Maß für die maximale Frequenz  $f_{max}$  zu betrachten, bei der die digitale Schaltung noch korrekt funktioniert.

$$f_{max} = 1/t_{pd}$$

*SN74LS08: Typ:  $f_{max} = 57 \text{ MHz}$ , Max:  $f_{max} = 100 \text{ ns}$*

- Sind viele digitale Schaltstufen hintereinander geschaltet, so vergrößert sich die Gesamtverzögerung und damit muß die digitale Schaltung langsamer betrieben werden.

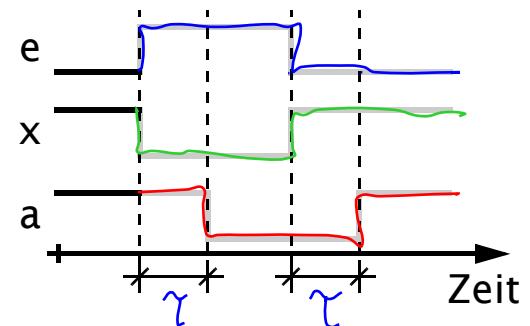
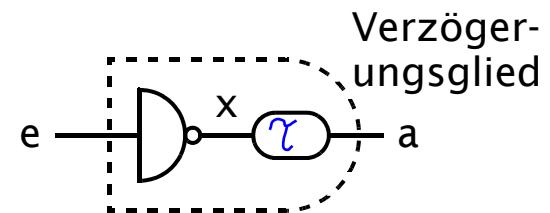
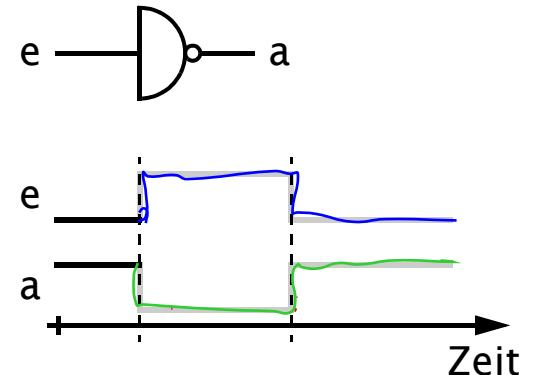


- ◆ Dynamisches Verhalten von Schaltnetzen und dessen Einfluß auf die Schaltnetzsynthese und -analyse
- ◆ Dynamisches Verhalten idealer Schaltnetzen
  - Signallaufzeiten einzelner Gatter und Signallaufzeiten auf Leitungen sind vernachlässigbar. 0-1- und 1-0-Übergänge sind verzögerungsfrei und unendlich schnell.
  - Zu einem Zeitpunkt ändert sich nur der Wert *einer* Variable, die Werte anderer Variablen bleiben konstant.
- ◆ Dynamisches Verhalten realer Schaltnetzen
  - Signallaufzeiten einzelner Gatter sowie 0-1- und 1-0-Übergänge sind nicht vernachlässigbar, und sie können unterschiedlich groß sein. Für die Laufzeitanalyse werden Signallaufzeiten auf Leitungen und für 0-1- und 1-0-Übergänge den jeweiligen Gattern zugerechnet.
  - Zu einem Zeitpunkt können sich Werte *vieler* Variablen ändern.

- ◆ Verzögerungsbehaftete Gatter

- Signale werden durch Gatter unterschiedlich verzögert. Es tritt aber keine Verformung der Signale ein.
- Ein verzögerungsfreies Gatter wird um ein Verzögerungsglied mit der Totzeit  $\tau$  erweitert.
- Das zeitliche Verhalten eines Gatters mit einem Verzögerungsglied mit der Totzeit  $\tau$  ist dasselbe wie vor dem Verzögerungsglied, aber um die Zeit  $\tau$  versetzt.

Gatter	Totzeit
NOT	$1 \cdot \tau$
AND	$2 \cdot \tau$
OR	$2 \cdot \tau$



# Hazards in Schaltnetzen

- ◆ Hazards
  - Als Antwort auf eine Wertänderung einer Eingangsvariable kann die Ausgangsvariable eines Schaltnetzes
    - sich nicht ändern, man spricht von einem statischen Übergang,
    - sich ändern, man spricht von einem dynamischen Übergang.
  - Durch Signalverzögerungen und unterschiedliche Signallaufzeiten können sich mehrfache Übergänge statt einzelner Übergänge ergeben.
  - Man bezeichnet solche Übergänge dementsprechend als
    - *statischer Hazard* und
    - *dynamischer Hazard*
- ◆ Hazards sind fehlerhafte Signalzustände, die infolge unterschiedlicher Signallaufzeiten im Schaltnetz entstehen.
- ◆ Hazards in Schaltnetzen führt zu vorübergehender Störung der Ausgangvariable.

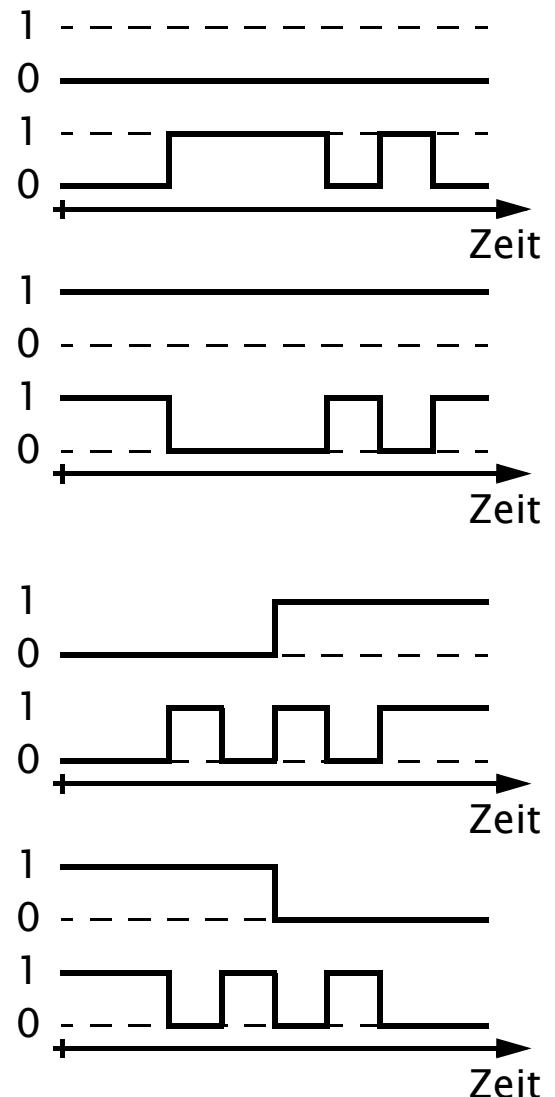
# Hazards in Schaltnetzen

## ♦ Statischer Hazard

- Als statischer Hazard wird ein kurzzeitiger (mehrfacher) Wechsel eines Signals bezeichnet, das eigentlich statisch 0 oder 1 hätte bleiben sollen.
  - Ein Hazard in einem statischen 0-Übergang heißt statischer 0-Hazard.
  - Ein Hazard in einem statischen 1-Übergang heißt statischer 1-Hazard.

## ♦ Dynamischer Hazard

- Als dynamischer Hazard wird ein kurzzeitiger (mehrfacher) Wechsel eines Signals bezeichnet, das sich eigentlich nur einmal hätte ändern sollen.
  - Ein Hazard in einem dynamischen 0-1-Übergang heißt dynamischer 01-Hazard.
  - Ein Hazard in einem dynamischen 1-0-Übergang heißt dynamischer 10-Hazard.

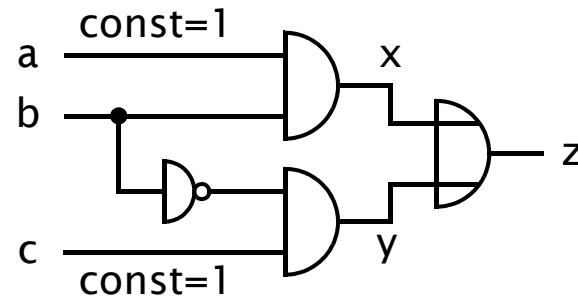


# Hazards in Schaltnetzen

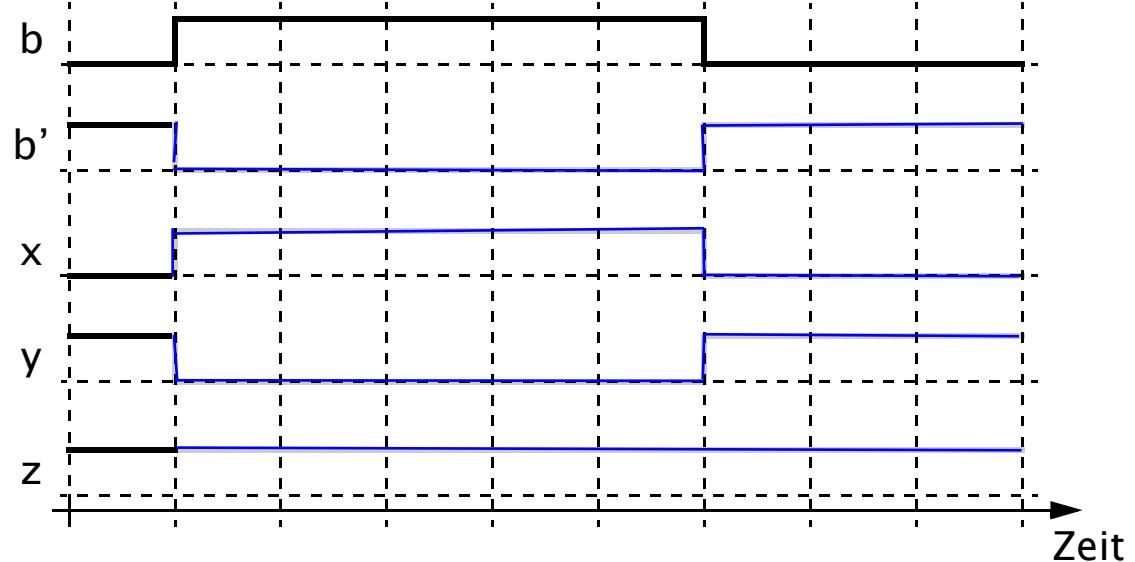
- Entstehung von Hazards

Beispiel:  $z(a, b, c) = \Sigma(1, 5, 6, 7) = a \cdot b + b' \cdot c$

Analyse des 0-1-Übergangs von  $z(1,0,1) = 1$  auf  $z(1,1,1) = 1$  mit den Variablen  $a=1$  und  $c=1$

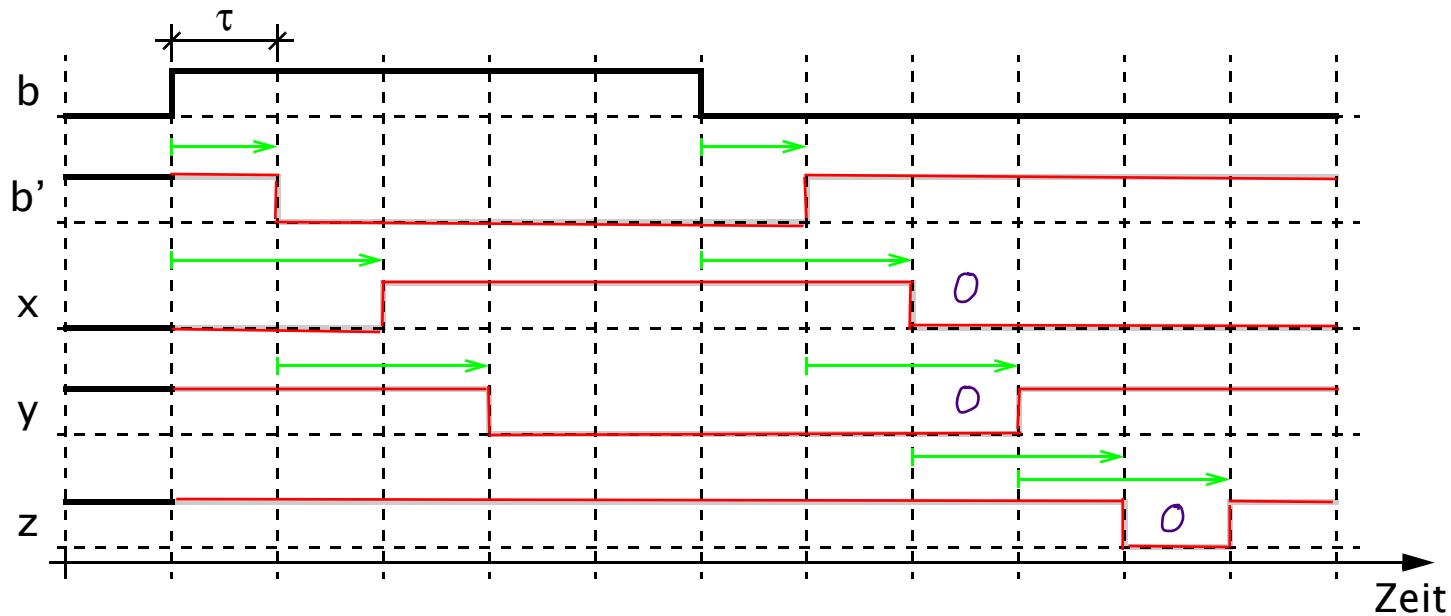
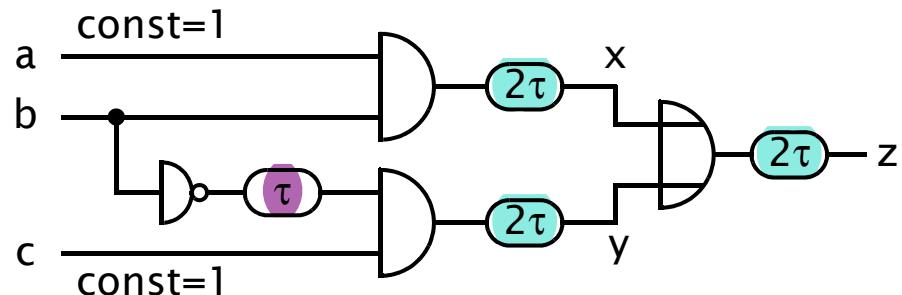


a	bc			
	00	01	11	10
0	0	1	0	0
1	0	1	1	1



# Hazards in Schaltnetzen

- Entstehung von Hazards



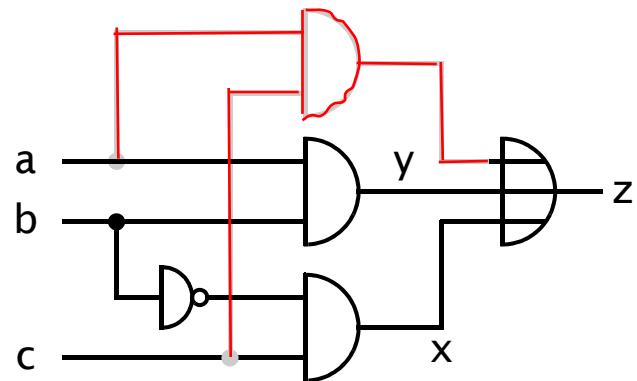
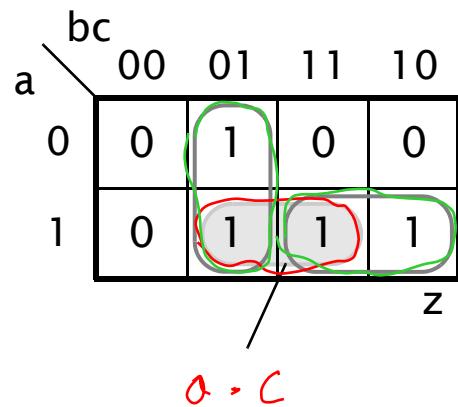
# Hazards in Schaltnetzen

- ◆ Ein *Strukturhazard* ist ein Hazard, dessen Ursache in der Struktur des realisierten Schaltnetzes liegt.
- ◆ Ein Strukturhazard läßt sich grundsätzlich durch die Änderung der Struktur des Schaltnetzes mit sog. Anti-Hazard-Termen unter Beibehaltung der Funktion beheben.
- ◆ Satz von Eichelberger:  
Ein Schaltnetz, das die Disjunktion *aller* Primimplikanten einer gegebenen Funktion realisiert, ist – unter der Voraussetzung, daß sich zu einem Zeitpunkt nur eine Eingangsvariable ändert – frei von
  - allen statischen Strukturhazards und
  - allen dynamischen Strukturhazards.

# Hazards in Schaltnetzen

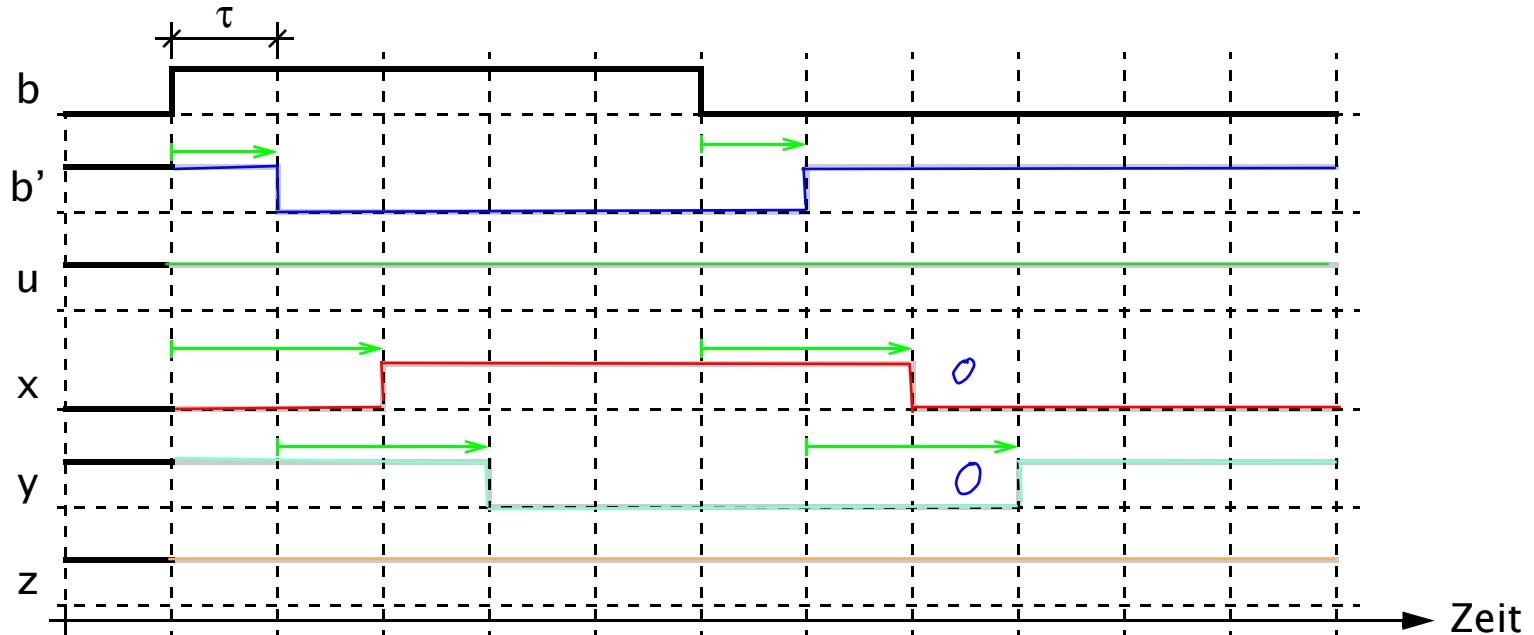
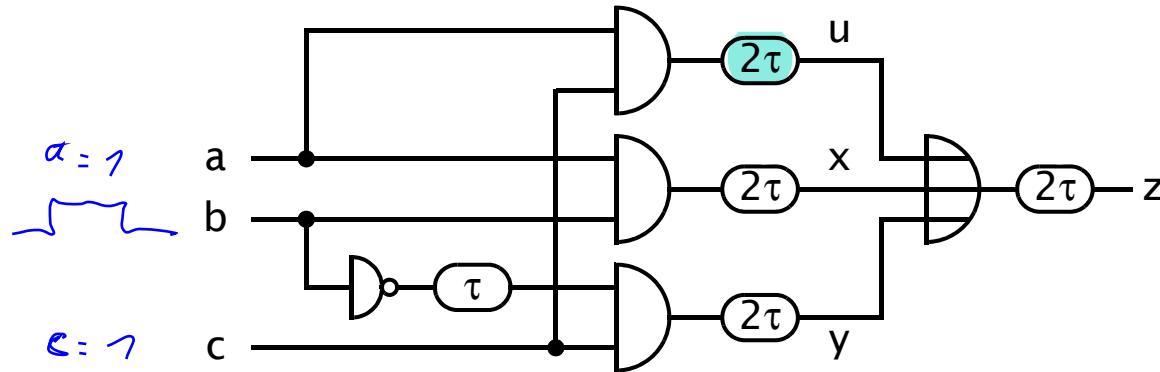
- ♦ Anti-Hazard-Gruppen

- Die Funktion  $z(a, b, c) = \Sigma(1, 5, 6, 7)$  hat drei Primimplikanten  $b' \cdot c$ ,  $a \cdot b$  und  $a \cdot c$
- Die minimierte Funktion  $z(a, b, c) = a \cdot b + b' \cdot c$  wird um den dritten fehlenden (und redundanten) Primimplikanten  $a \cdot c$  erweitert.
- Solche Primimplikanten werden oft als Anti-Hazard-Gruppen bezeichnet.
- Die hazardfreie Funktion  $z(a, b, c) = a \cdot b + b' \cdot c + a \cdot c$



# Hazards in Schaltnetzen

- ◆ Auswirkung von Anti-Hazard-Gruppen



- ◆ Ein *Funktionshazard* ist ein Hazard, dessen Ursache in der zu realisierenden Funktion selbst liegt, und deshalb in jedem möglichen Schaltnetz, das diese Funktion realisiert, auftreten muß.
- ◆ Der Funktionshazard selbst kann *nicht* behoben werden, aber für ein konkretes Schaltnetz kann eventuell der Fehler, der aus einem Funktionshazard resultiert, durch geeignete Wahl von Verzögerungsgliedern behoben werden.
- ◆ Der Funktionshazard läßt sich verhindern, wenn man erreicht, daß sich an jedem beliebig herausgegriffenen Zeitpunkt jeweils nur eine von denjenigen Eingangsvariablen, die den Hazard verursachen, ändert.

# Hazards in Schaltnetzen

- ◆ Statischer Funktionshazard

$$y(a, b, c) = \Sigma(3, 6, 7) = b \cdot (a + c)$$

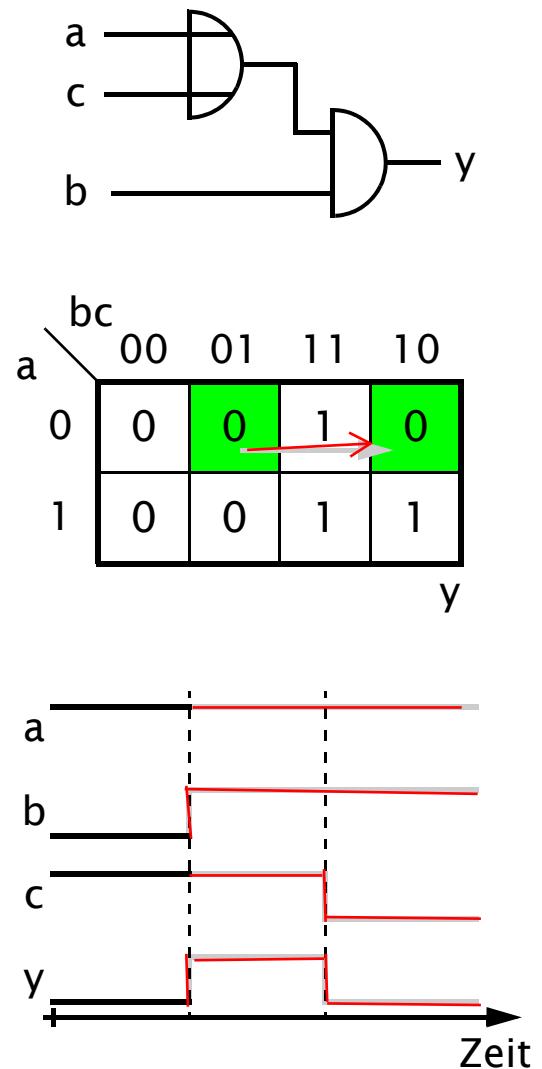
Übergang von  $y(0,0,1) = 0$  auf  $y(0,1,0) = 0$

- ideales Zeitverhalten:

Eingangsvariablen b und c ändern sich gleichzeitig  $\Rightarrow$  Ausgangsvariable y ändert sich nicht und bleibt  $y = 0$

- reales Zeitverhalten:

Eingangsvariablen b und c ändern sich *fast* gleichzeitig, so daß es zu kleinen Abweichungen im Zeitverhalten kommt. Dadurch können folgende Zwischenwerte und Zwischenübergänge  $(0,0,1)_{abc} \rightarrow (0,1,1)_{abc} \rightarrow (0,1,0)_{abc}$  entstehen, die zu einem statischen Funktionshazard für die Ausgangsvariable führen.



# Hazards in Schaltnetzen

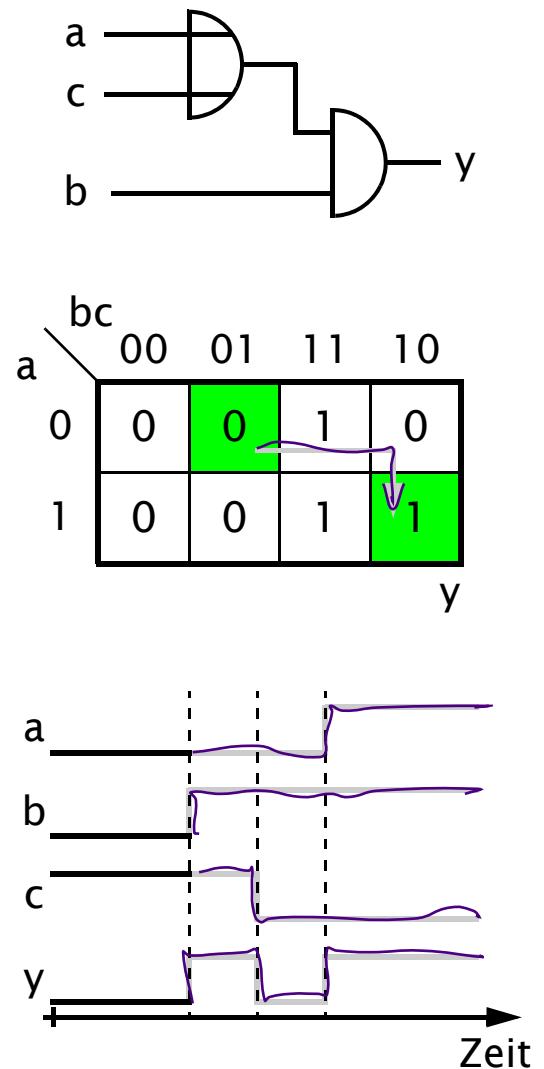
- ◆ Dynamischer Funktionshazard

$$y(a, b, c) = \Sigma(3, 6, 7) = b \cdot (a + c)$$

Übergang von  $y(0,0,1) = 0$  auf  $y(1,1,0) = 1$

- ideales Zeitverhalten: Eingangsvariablen  $a, b$ , und  $c$  ändern sich gleichzeitig  $\Rightarrow$  Ausgangsvariable  $y$  ändert sich ein einziges Mal von  $y = 0$  auf  $y = 1$
- reales Zeitverhalten: Eingangsvariablen  $a, b$ , und  $c$  ändern sich *fast* gleichzeitig, so daß es zu kleinen Abweichungen im Zeitverhalten kommt.

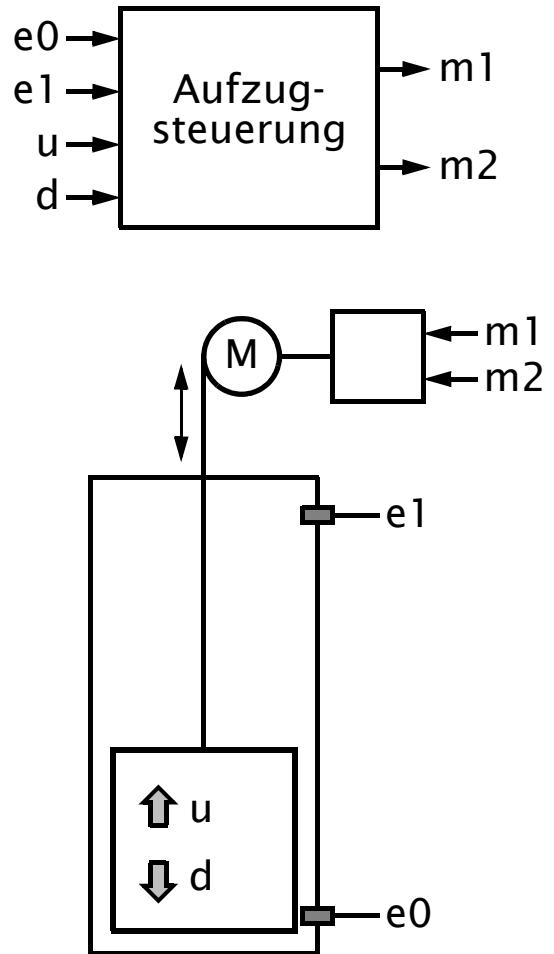
Dadurch können folgende Zwischenwerte und Zwischenübergänge  $(0,0,1)_{abc} \rightarrow (0,1,1)_{abc} \rightarrow (0,1,0)_{abc} \rightarrow (1,1,0)_{abc}$  entstehen, die zu einem dynamischen Funktionshazard für die Ausgangsvariable führen.



- ◆ Einführendes Beispiel – Aufzugsteuerung

Synthese einer Aufzugsteuerung für ein vereinfachtes Modell einer Aufzugsanlage in einem Gebäude mit zwei Etagen

- Aufzugsanlage besteht aus
  - einem Maschinenraum mit dem Antrieb und mit der Aufzugsteuerung
  - einem Aufzugsschacht mit Tragseilen, Gegengewichten und Positionssensoren
  - einer Fahrkabine mit einer einfachen Schalttafel



Im Aufzugsschacht sind zwei digitale (Positions-)Sensoren e0 und e1 zur Erfassung der Position der Fahrkabine montiert. Die Position der Fahrkabine wird laufend an die Aufzugsteuerung mitgeteilt. Mit  $e_i=1$  wird ein aktiver Sensor gekennzeichnet.

e1	e0	Position der Fahrkabine
0	0	zwischen den Etagen
1	0	in der oberen Etage
0	1	in der unteren Etage

In der Fahrkabine befindet sich eine einfache Schalttafel mit zwei Tasten u (up) und d (down), mit denen der Fahrgast die gewünschte Fahrtrichtung auswählen kann.

u	d	Fahrtrichtung
0	0	keine
1	0	aufwärts
0	1	abwärts

Der Antrieb besteht aus einem Elektromotor M, der über eine Motorsteuerung mit zwei Signalen m1 und m2 gesteuert wird. Der Elektromotor kann entweder stehen oder sich wahlweise entweder nach links oder rechts drehen. Dabei wird die Drehbewegung des Elektromotors über eine Umlenkrolle in die vertikale Bewegung der Fahrkabine umgesetzt.

m1	m2	Elektromotor
0	0	bleibt stehen
0	1	dreht sich nach rechts
1	0	dreht sich nach links

Die Aufzugsteuerung soll folgendes Verhalten aufweisen:

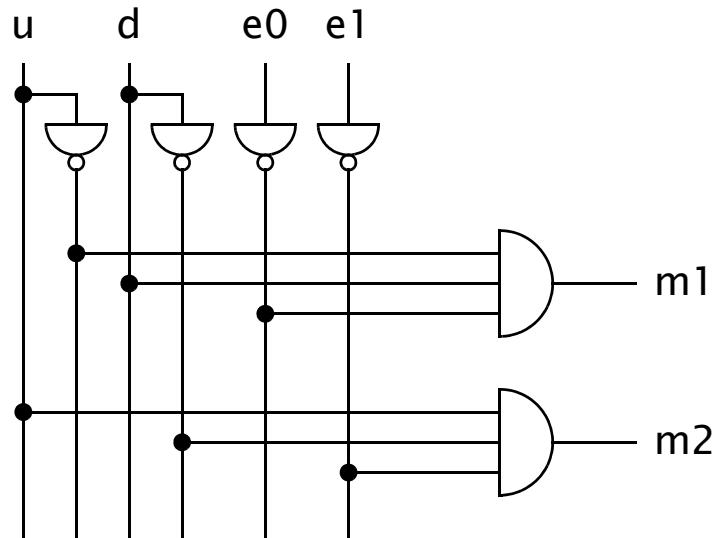
1. Die Fahrkabine wartet auf der aktuellen Etage so lange, bis ein Fahrgast über die Schalttafel die gewünschte Fahrtrichtung gewählt hat.
2. Die Fahrkabine soll sich in die gewünschte Fahrtrichtung bewegen (in die obere bzw. in die untere Etage), sofern sie sich nicht in der entsprechenden Etage befindet.

# Schaltwerke

- ♦ Aufzugsteuerung als Schaltnetz
  - Funktionstabelle

e1	e0	u	d	m1	m2
-	-	0	0	0	0
-	-	1	1	0	0
0	0	1	0	0	1
0	1	1	0	0	1
1	0	1	0	0	0
0	0	0	1	1	0
1	0	0	1	1	0
0	1	0	1	0	0

Schaltnetz



- Funktionsgleichungen
 
$$m1 = f1(u, d, e1, e0) = u' \cdot d \cdot e0'$$

$$m2 = f2(u, d, e1, e0) = u \cdot d' \cdot e1'$$

- ◆ Analyse der bisherigen Lösung

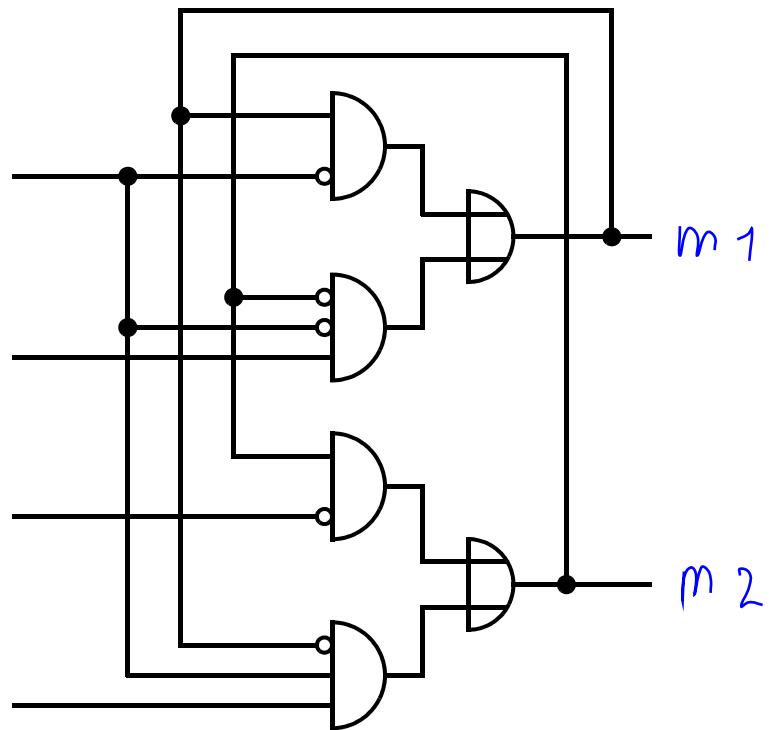
- Die Fahrkabine bewegt sich nur, wenn ein Fahrgast die passende Taste u oder d gedrückt hält. Läßt er die Taste los, dann bleibt die Fahrkabine stehen (auch zwischen den Etagen). Der Fahrgast kann die Fahrtrichtung zu jedem Zeitpunkt ändern.
- In der Realität funktioniert eine Aufzugsteuerung ein wenig anders.
- Man braucht nicht dauernd eine Taste für die Fahrtrichtung gedrückt zu halten, wenn man mit dem Aufzug fahren will.
- Man drückt nur einmal auf eine Taste und die Fahrkabine setzt sich in Bewegung.
- Die Aufzugsteuerung „merkt“ sich also irgendwie die gewählte Fahrtrichtung, so daß die Fahrkabine automatisch weiterfährt, auch nachdem der Fahrgast die Taste losgelassen hat.
- Wir erweitern deshalb die Aufzugsteuerung um eine neue Funktionalität, mit der dieses Verhalten realisiert wird.

- ◆ Analyse der bisherigen Lösung (Fortsetzung)
  - Dieses „Merken“ der gewählten Fahrtrichtung kann z.B. so erfolgen,
    - daß die Steuerung sich an die gedrückte Taste erinnert, oder
    - daß die Steuerung die Information aus der Drehrichtung des Elektromotors rekonstruiert.
  - Wir verfolgen den zweiten Ansatz:  
Dazu müssen in der Funktionstabelle eingangsseitig zwei neue Spalten eingefügt werden, die mit den Signalen m1 und m2 zur Steuerung des Elektromotors übereinstimmen.
  - Auf diese Weise entsteht eine Rückkopplung in der Funktion. Die Ausgangsfunktionen m1 und m2 sind nicht nur von der Eingangsvariablen/-signalen e1, e0, u und d abhängig, sondern auch von sich selbst (also von m1 und von m2).
  - Somit ergeben sich folgende, rückgekoppelte Funktionen:  
 $m1 = g1(m1, m2, e1, e0, u, d)$  und  $m2 = g2(m1, m2, e1, e0, u, d)$

- Erweiterte Aufzugsteuerung
  - Funktionstabelle

Eingänge						Ausgänge	
m1	m2	e1	e0	u	d	m1	m2
0	0	0	1	0	1	0	0
0	0	0	1	1	-	0	1
0	0	1	0	1	0	0	0
0	0	1	0	-	1	1	0
0	1	0	1	-	-	0	1
0	1	1	0	-	-	0	0
1	0	0	1	-	-	1	0
1	0	1	0	-	-	0	0

Schaltnetz mit Rückkopplung



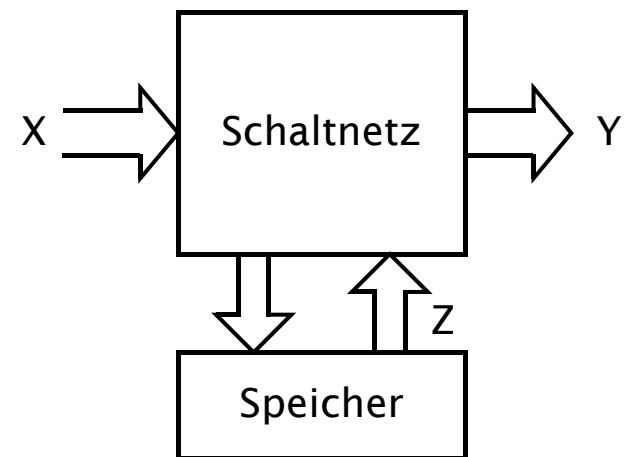
- Funktionsgleichungen

$$m_1 = g_1(m_1, m_2, u, d, e_1, e_0) = m_1 \cdot e_0' + m_2 \cdot e_0' \cdot d$$

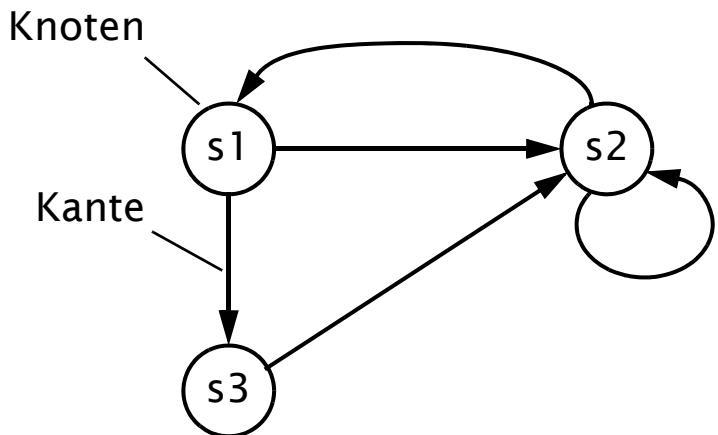
$$m_2 = g_2(m_1, m_2, u, d, e_1, e_0) = m_2 \cdot e_1' + m_1' \cdot e_1' \cdot u$$

- ◆ Funktionsweise der erweiterten Aufzugsteuerung
  - die Fahrkabine steht:
    - der Elektromotor dreht sich nicht ( $m_1=0, m_2=0$ )
    - sie steht entweder in der oberen Etage ( $e_1=1, e_0=0$ ) oder in der unteren Etage ( $e_1=0, e_0=1$ )
    - den Fall, daß sie zwischen den beiden Etagen stehen bleibt, schließen wir der Einfachheit halber aus.
    - der Fahrgast kann die Fahrtrichtung nicht zu jedem (beliebigen) Zeitpunkt auswählen, sondern nur wenn die Fahrkabine entweder in der unteren oder in der oberen Etage steht.
  - die Fahrkabine fährt aufwärts:
    - der Elektromotor dreht sich nach rechts ( $m_1=0, m_2=1$ ), und zwar nur so lange, bis die Fahrkabine die obere Etage erreicht hat ( $e_1=1$ ), dann stoppt die Aufzugsteuerung den Elektromotor.
  - die Fahrkabine fährt abwärts:
    - der Elektromotor dreht sich nach links ( $m_1=1, m_2=0$ ), und zwar nur so lange, bis die Fahrkabine die untere Etage erreicht hat ( $e_0=1$ ), dann stoppt die Aufzugsteuerung den Elektromotor.

- ◆ Ein Schaltnetz ist
  - eine zustandsfreie, kombinatorische Schaltung ohne „Gedächtnis“.
  - Ausgangswerte Y sind ausschließlich und unmittelbar (abgesehen vor einer kurzen Verzögerung) von den aktuellen Eingangswerten X abhängig.
- ◆ Ein Schaltwerk ist
  - ein Schaltnetz mit „Gedächtnis“, eine zustandsbehaftete, sequentielle Schaltung.
  - Ausgangswerte Y sind zu einem bestimmten Zeitpunkt sowohl vom vergangenen Verhalten der Schaltung Z als auch von den aktuellen Eingangswerten X abhängig.



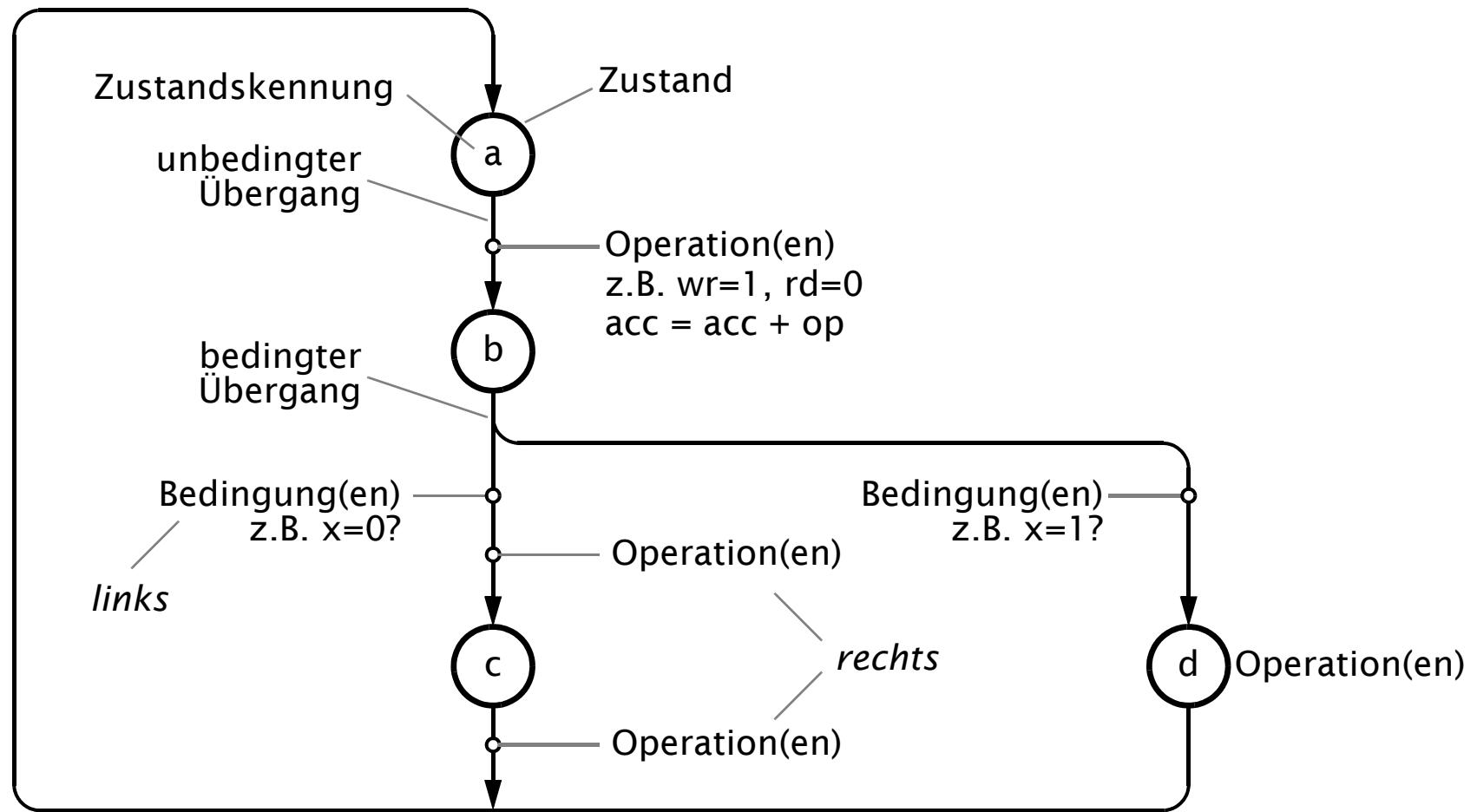
- ♦ Ein gerichteter Graph  $G = (V, E)$  besteht aus zwei endlichen Mengen:
  - einer nicht leeren Knotenmenge  $V$  und
  - einer (möglicherweise leeren) Kantenmenge  $E$ ,so daß jeder Kante  $e$  aus  $E$  ein geordnetes Knotenpaar  $(u, v)$  zugeordnet ist. Man sagt, daß die Kante  $e$  den Knoten  $u$  mit dem Knoten  $v$  verbindet. Dabei wird  $u$  als Anfangsknoten von  $e$  und  $v$  als Endknoten von  $e$  bezeichnet.



$$\begin{aligned}V &= \{s_1, s_2, s_3\} \\E &= \{(s_1, s_3), (s_1, s_2), \\&\quad (s_2, s_1), (s_2, s_2)\} \\&\quad (s_3, s_2)\}\end{aligned}$$

- ◆ Ein Zustandsgraph  $G_z = (G, V_0, V_N, B, A)$  ist ein gerichteter Graph G, dessen Knoten als *Zustände* und Kanten als *Zustandsübergänge* interpretiert werden, erweitert um folgende endliche Mengen:
  - eine nicht leere Knotenmenge  $V_0$  für Anfangszustände,
  - eine (möglicherweise leere) Knotenmenge  $V_N$  für Endzustände,
  - eine (möglicherweise leere) Menge von Bedingungen B, die den Zustandsübergängen zugeordnet sind,
  - eine (möglicherweise leere) Menge von Aktionen A (Operationen), die entweder den Zuständen oder den Zustandsübergängen zugeordnet sein können.
- ◆ Semantikregel: Wenn ein Zustand aktiv ist, und eine Bedingung wahr ist, dann finden der Zustandsübergang und die Ausführung der Aktion(en) statt.

- ♦ Zustandsgraph mit einer praktischen Erklärung



- ♦ Zustandsorientierte Modellierung der Aufzugsteuerung

- Zustände:  $V = \{s_1, s_2, s_3, s_4\}$ 
  - $s_1$ : Fahrkabine steht in der unteren Etage
  - $s_2$ : Fahrkabine steht in der oberen Etage
  - $s_3$ : Fahrkabine fährt aufwärts
  - $s_4$ : Fahrkabine fährt abwärts
- Zustandsübergänge:
  - $e_1$ : Fahrkabine setzt sich in Aufwärtsbewegung
  - $e_2$ : Fahrkabine erreicht die obere Etage
  - $e_3$ : Fahrkabine setzt sich in Abwärtsbewegung
  - $e_4$ : Fahrkabine erreicht die untere Etage
- Bedingungen:  $B = \{b_1, b_2, b_3, b_4\}$ 
  - $b_1$ : die Taste u (up) wird betätigt
  - $b_2$ : die Taste d (down) wird betätigt
  - $b_3$ : der Positionssensor in der unteren Etage wird aktiviert
  - $b_4$ : der Positionssensor in der oberen Etage wird aktiviert
- Aktionen:  $A = \{e_1, e_2, e_3\}$ 
  - $a_1$ : der E-Motor für die Abwärtsbewegung wird eingeschaltet
  - $a_2$ : der E-Motor für die Aufwärtsbewegung wird eingeschaltet
  - $a_3$ : der E-Motor bleibt stehen

$$e_5 = (s_1, s_1)$$

$$e_6 = (s_2, s_2)$$

$$e_7 = (s_3, s_3)$$

$$e_8 = (s_4, s_4)$$

$$e_9 = (s_1, s_3)$$

$$e_{10} = (s_3, s_2)$$

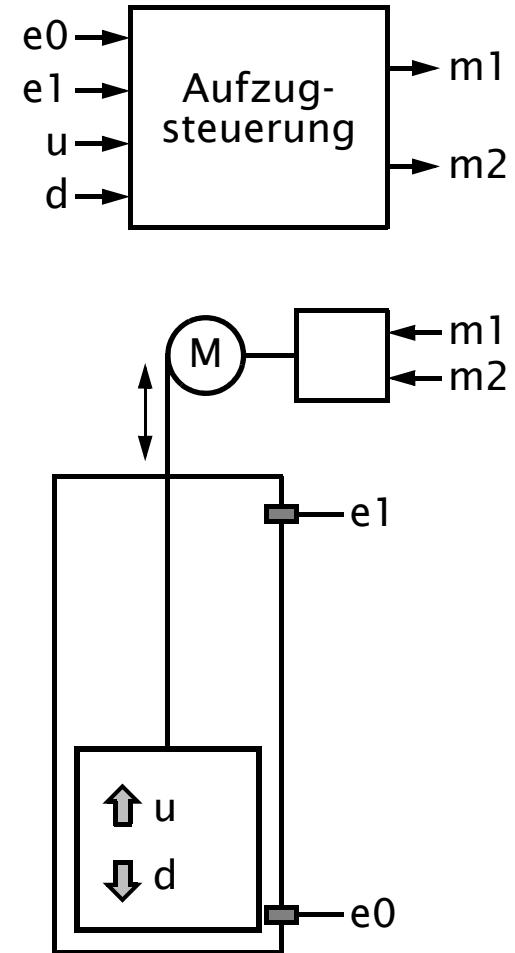
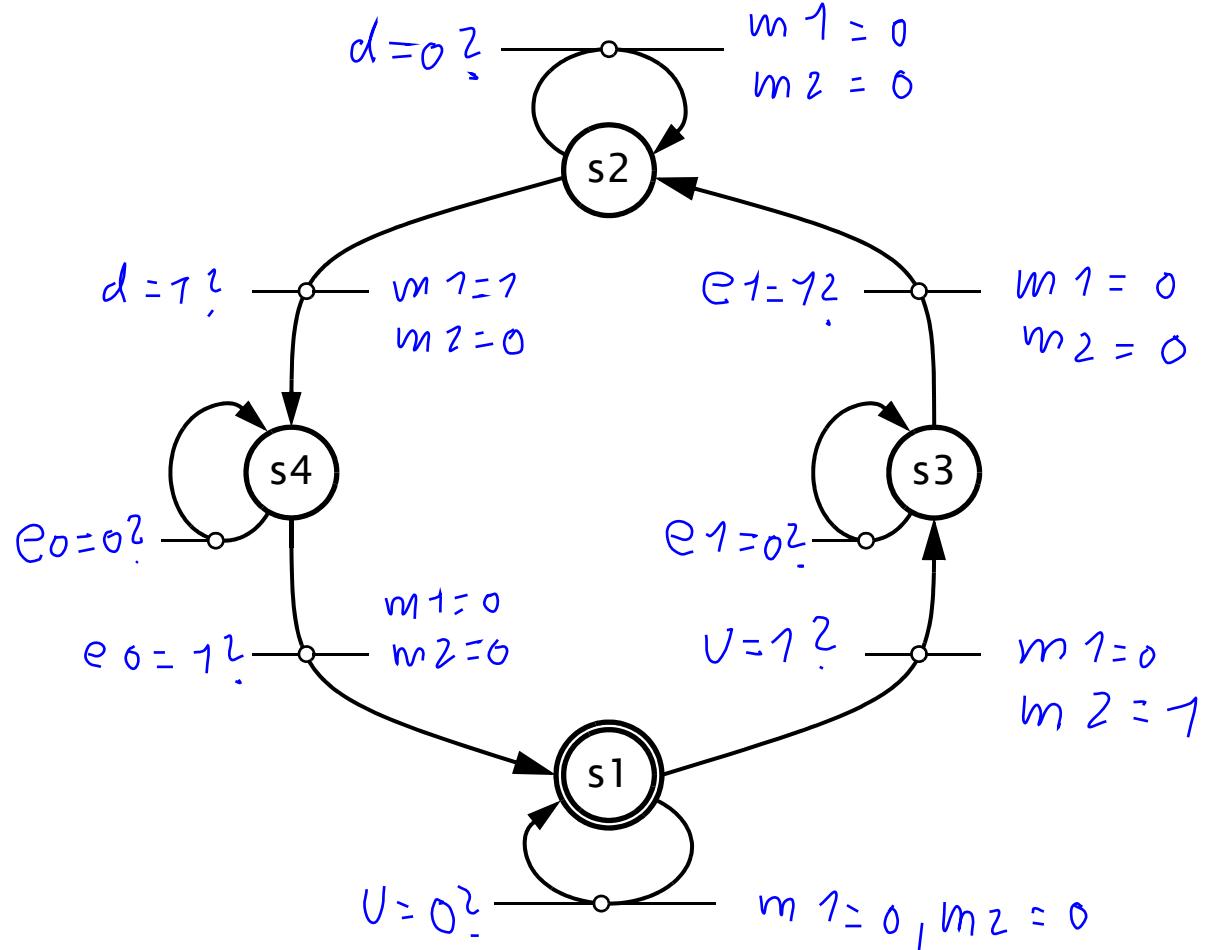
$$e_{11} = (s_2, s_4)$$

$$e_{12} = (s_4, s_1)$$

$$V_O = \{s_1\}$$

$$V_W = \{\emptyset\}$$

- Zustandsgraph für die Aufzugsteuerung

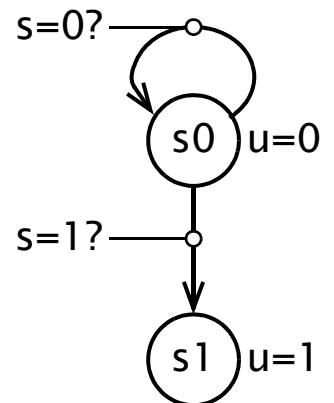
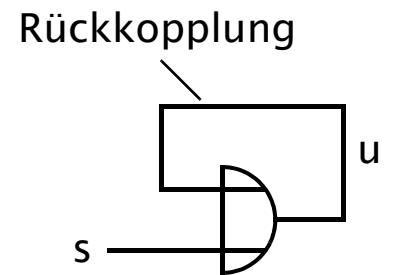


- ◆ Digitale Systeme lassen sich grundsätzlich in *Schaltnetze* und *Schaltwerke* aufteilen.
- ◆ Schaltwerke können als *synchron* oder *asynchron* arbeitende Schaltwerke realisiert sein.
- ◆ Synchrone Schaltwerke
  - Änderungen interner Zustände erfolgen nur zu vorgegebenen Zeitpunkten.
  - Zwischen den Änderungszeitpunkten bleiben alle Zustandswerte stabil.
  - Es gibt ausgewählte (externe) Signale (sog. Taktimpulse), die normalerweise auf alle Zustandsgrößen wirken, und Änderungen des Systems bewirken.
  - Da alle Zustandsgrößen sich gleichzeitig ändern, erhält man so ein synchrones Verhalten des gesamten Systems.

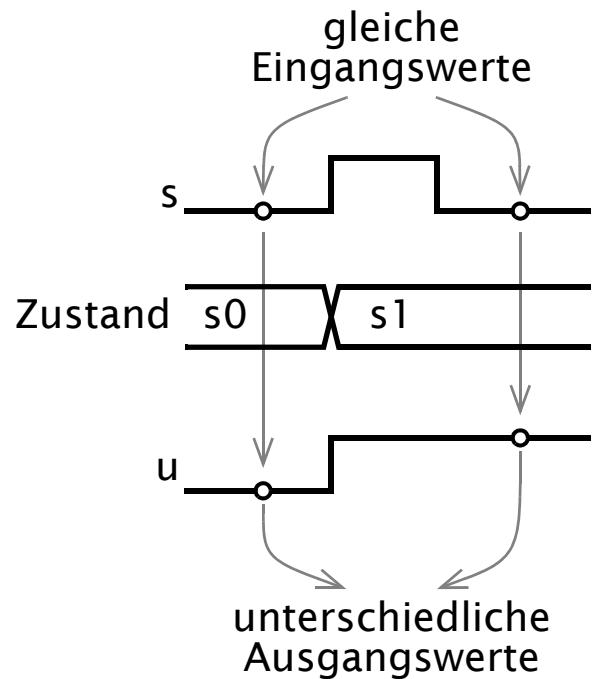
- ◆ **Asynchrone Schaltwerke**
  - Änderungszeitpunkte sind nicht extern vorgegeben (keine expliziten Taktimpulse).
  - Änderungszeitpunkte können jeder Zeit auftreten und sind normalerweise von der konkreten technischen Realisierung abhängig.
  - Aufgrund von Laufzeitunterscheiden bei den einzelnen Zustandsgrößen kann es zu Problemen wie Hazards oder Wettläufen (Races) kommen. Dadurch können sich Signale auf verschiedenen Pfaden überholen.
  - Sind mehrere Zustandsgrößen im System vorhanden, können sich diese asynchron (nicht gleichzeitig) ändern.
- ◆ **Synchrone Schaltwerke** zeigen diese Probleme nicht und sind daher bei der Realisierung zu bevorzugen.

- ◆ In Schaltnetzen mit Rückkopplung(en) können Informationen dauerhaft gespeichert werden (vgl. Aufzugsteuerung)  
⇒ Automaten, Schaltwerke
- ◆ Aufgrund der Komplexität heutiger Aufgabenstellungen werden Schaltwerke selten als Schaltnetze mit Rückkopplungen gebaut (aufwändige Timing-Analyse, Maßnahmen gegen Hazards).
- ◆ Komplexe Aufgabenstellungen werden
  - systematisch mit Hilfe der Automatentheorie modelliert, und
  - modular als Schaltwerke, bestehend aus Speicherelementen sowie aus Übergangs- und Ausgangsschaltnetzen realisiert.
- ◆ Die Grundlage solcher Speicherelemente bilden relativ einfache Schaltnetze mit Rückkopplungen.

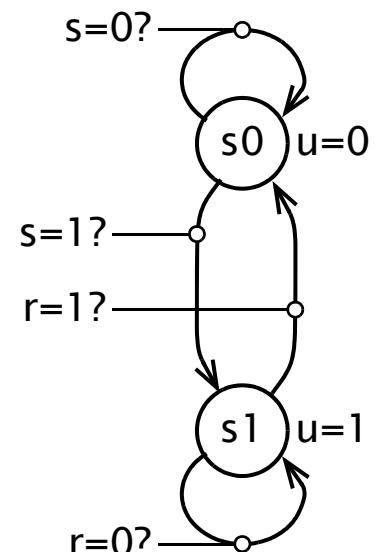
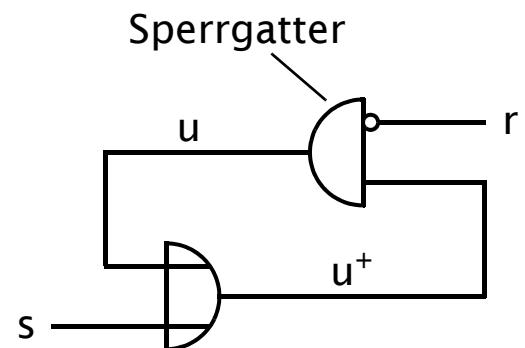
- ◆ Speicherelement zur Speicherung eines Impulses
  - Das einfachste Schaltnetz mit einer Rückkopplung besteht aus einem OR-Gatter mit zwei Eingängen und einem Ausgang, in dem der Ausgang mit einem der beiden Eingänge über eine Rückkopplung verbunden ist.
  - Das so aufgebaute minimale Schaltwerk kann an seinem Eingang einen Impuls  $s=0 \rightarrow 1$  „aufspüren“ und diese Änderung/Information durch die Rückkopplung dauerhaft als Zustand  $u=1$  „speichern“.
  - Funktionsweise:
    - Solange sich das Speicherelement im Zustand  $s_0$  befindet (mit  $u=0$ ), kann ein Impuls (also eine Signaländerung  $0 \rightarrow 1$ ) auf der Eingangsleitung  $s$  erfaßt werden.
    - Ist ein Impuls ( $s=1$ ) erfaßt, so geht das Speicherelement in den Zustand  $s_1$  (mit  $u=1$ ) über.
    - Alle nachfolgenden Impulse können den Zustand  $s_1$  des Speicherelements nicht mehr ändern.



- ♦ Speicherelement zur Speicherung eines Impulses
  - Am Impulsplan sieht man, wie sich die Änderung eines Eingangswertes auf den Zustand der Schaltung auswirkt.
  - **Der (innere) Zustand** ist derjenige Teil der Information über eine Schaltung, der neben den aktuellen Eingangswerten die Ausgangswerte mitbestimmt.
  - Das einfachste Schaltwerk zur Speicherung eines Impulses kann nicht in seinen Ausgangszustand  $s_0$  (mit  $u=0$ ) wieder versetzt werden, z.B. um erneut einen Impuls erfassen zu können.
  - Damit das Zurücksetzen (sog. Normieren) des Speicherelementes möglich ist, wird das Speicherelement um ein sog. Sperrgatter in der Rückkopplungsschleife erweitert.



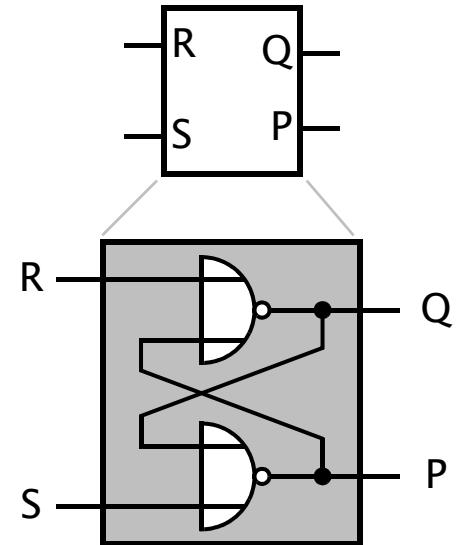
- ◆ Speicherelement zur Speicherung von Werten einer Variable
  - Das Sperrgatter ist ein 2er-AND-Gatter mit einem invertierten Eingang, an dem ein Rücksetzsignal  $r$  angelegt ist.
  - Bei  $r=0$  wird die Information durch die Rückkopplung aufrechterhalten.
  - Bei  $r=1$  wirkt das AND-Gatter wie eine Sperre und unterbricht den Informationsfluß in der Rückkopplung.
  - Durch Übergang auf NOR-Gatter und Herausführen beider Ausgänge entsteht eine elementare Schaltung, mit der die Werte einer booleschen Variable gespeichert werden können. Diese Schaltung wird als RS-Flipflop bezeichnet.



## ♦ RS-Flipflop

- zwei Eingänge: R (Reset) und S (Set)
- zwei Ausgänge: Q und P (mit  $P := Q'$ )
- Funktionstabelle

	S	R	$Q^+$	$P^+$	Funktion
(1)	1	0	1	0	setzen
(3)	0	1	0	1	löschen
(2)	0	0	Q	P	halten
(4)	1	1	x	x	irregulär



- zwei stabile Zustände ( $P=0, Q=1$ ) und ( $P=1, Q=0$ )
- einen transienten Zustand ( $P=Q=0$ )
- $S=R=1$  ist irregulär, weil:
  - beide Ausgänge dauerhaft  $P = Q = 0$
  - undefiniertes (metastabiles) Verhalten beim Wechsel von  $S=R=1$  nach  $S=R=0$

- ◆ Timing-Analyse eines RS-Flipflop
- Charakteristische Funktionsgleichung:

$$Q(t+1) = S(t) + R(t)' \cdot Q(t)$$

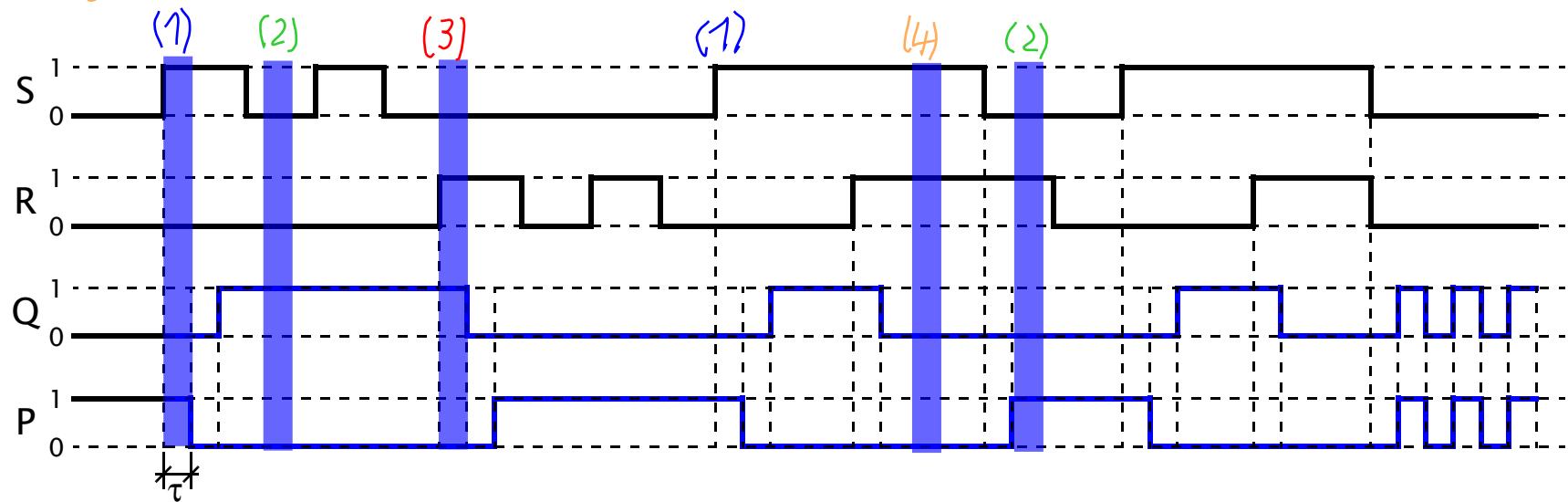
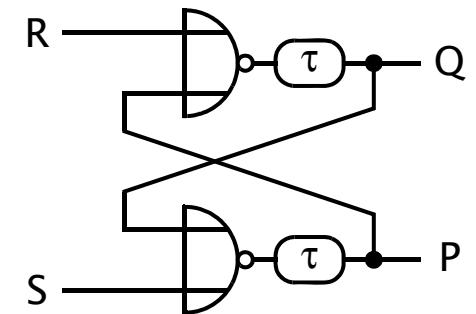
$$Q^+ = S + R' \cdot Q \quad \text{mit } P = Q'$$

(1) Setzen

(2) halten

(3) Löschen

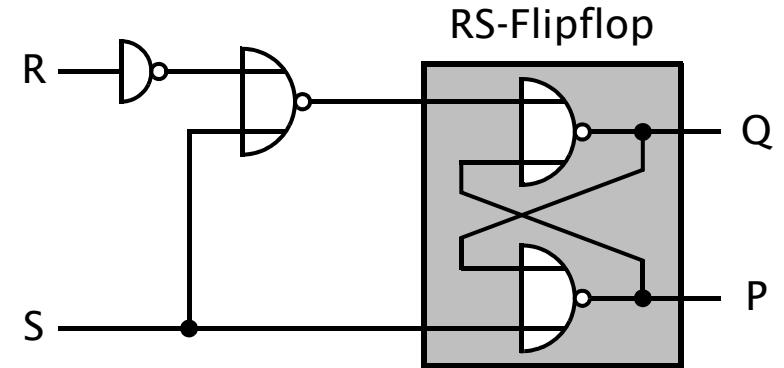
(4) irregulär



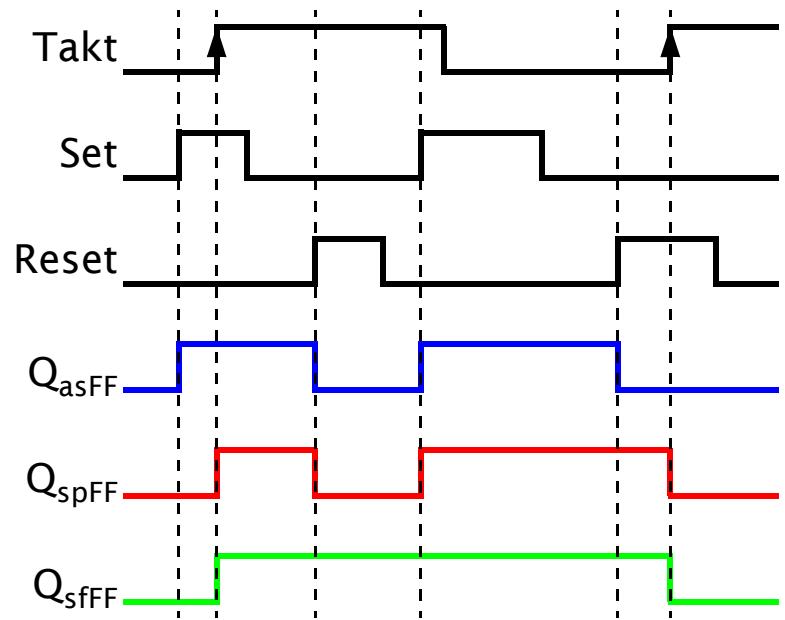
- ◆ Zusammenfassung der Timing-Analyse eines RS-Flipflop
  - Im Fall der irregulären Eingangsbedingung ( $S=R=1$ ) sind beide Ausgangssignale 0 ( $Q=P=0$ ).
  - Wenn unmittelbar nach der irregulären Eingangsbedingung die Funktion „halten“ ( $S=R=0$ ) folgt, dann zeigt das RS-Flipflop ein instabiles (oszillierendes) Verhalten.
  - Durch die Verzögerungen der beiden NOR-Gatter ist der übernommene Wert dann nicht eindeutig vorhersagbar.
  - Kann in einem Entwurf aber ausgeschlossen werden, daß unmittelbar nach der irregulären Eingangsbedingung die Funktion „halten“ folgt, dann läßt sich auch eine solche Eingangsbedingung in einer Schaltung sinnvoll einsetzen.
  - Folgen die Funktionen „setzen“ oder „löschten“ unmittelbar nach der irregulären Eingangsbedingung, so nimmt das RS-Flipflop einen definierten und stabilen Zustand an.

- ◆ RS-Flipflops mit besonderem Verhalten
  - Die irreguläre Eingangsbedingung  $S=R=1$  lässt sich mit einem zusätzlichen Schaltnetz vor den beiden Eingängen eines RS-Flipflops verhindern.
  - Es gibt drei Alternativen für die Behandlung der Eingangsbedingung  $S=R=1$ , die drei modifizierte RS-Flipflops ergeben:
    - RS-Flipflop mit Speichervorrang
    - RS-Flipflop mit Löschvorrang
    - RS-Flipflop mit Setzvorrang
  - RS-Flipflop mit Setzvorrang

S	R	$Q^+$	$P^+$	Funktion
1	0	1	0	setzen
0	1	0	1	löschen
0	0	Q	P	halten
1	1	1	0	setzen



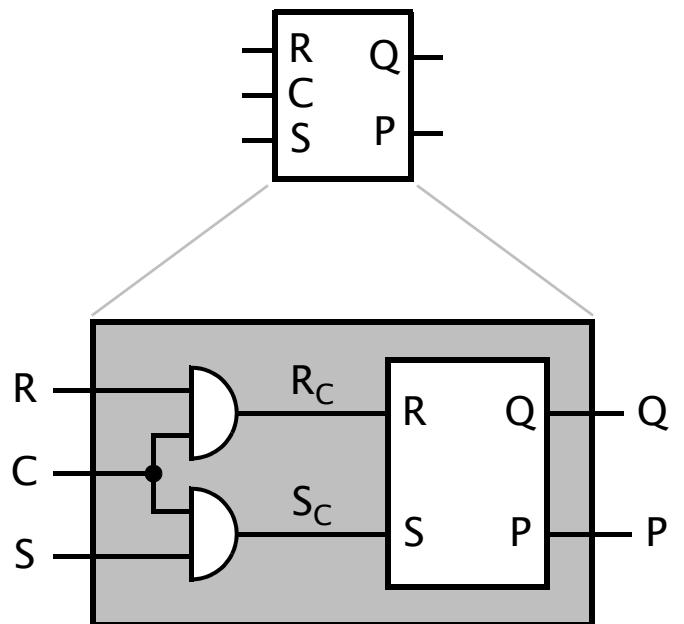
- ♦ Verschiedene Arten der Steuerung von Flipflops
  - **asynchrone** (ungetaktete) **Flipflops**: ihr Zustand wird nur durch die Setzen-/Rücksetzen-Eingänge (Set-/Reset) gesteuert.
  - **synchrone** (getaktete, getriggerte) **Flipflops**: der Zeitpunkt der Informationsübernahme wird durch ein Takt-/Steuersignal vorgegeben.
    - **pegelgesteuerte** (zustandsgesteuerte) **Flipflops**: die Übernahme der Information wird durch einen Pegel (0/1) des Steuersignals veranlaßt.
    - **einflankengesteuerte** **Flipflops**: die Übernahme der Information wird durch einen Zustandswechsel ( $0 \rightarrow 1$  oder  $1 \rightarrow 0$ ) des Steuersignals veranlaßt.
    - **zweiflankengesteuerte** **Flipflops** (Master-Slave-Flipflops)



- ♦ synchrones, pegelgesteuertes RS-Flipflop
  - Ein asynchrones RS-Flipflop wird um ein zusätzliches Schaltnetz, bestehend aus zwei AND-Gattern, vor den Eingängen R und S zu einem synchronen, pegelgesteuerten RS-Flipflop erweitert.
  - Eingangssignale an R und S werden nur dann übernommen, wenn das Taktsignal C aktiv ist, d.h. C=1.

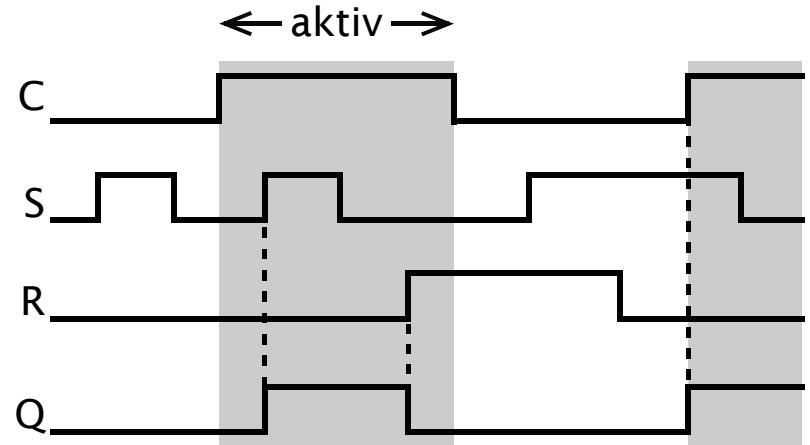
- Funktionstabelle

C	S	R	$Q^+$	$P^+$	Funktion
0	-	-	Q	P	halten
1	0	0	Q	P	halten
1	0	1	0	1	löschen
1	1	0	1	0	setzen
1	1	1	x	x	irregulär



## ♦ Timing-Analyse

- aktive Taktphase ( $C=1$ ): die R- und S-Eingänge sind „frei geschaltet“, mehrere Zustandsänderungen möglich.
- nicht aktiven Taktphase ( $C=0$ ): die R- und S-Eingänge bleiben „gesperrt“; Impulse auf diesen Eingangsleitungen haben keine Wirkung. Der Zustand des RS-Flipflops ist von den R- und S-Eingängen unabhängig und bleibt somit gespeichert.
- Bei einem Übergang des Taktes in die nicht aktive Phase ( $C:1 \rightarrow 0$ ) wird der letzte Zustand entsprechend den R- und S-Eingängen übernommen.
- Signale an den R- und S-Eingängen sollten stets länger anliegen, als die aktive Taktphase dauert.

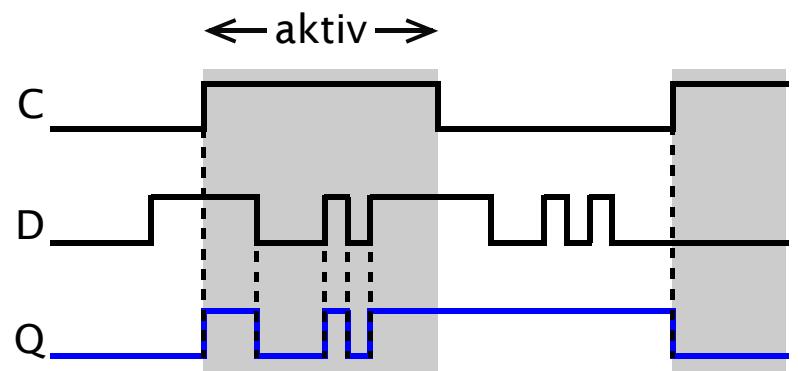
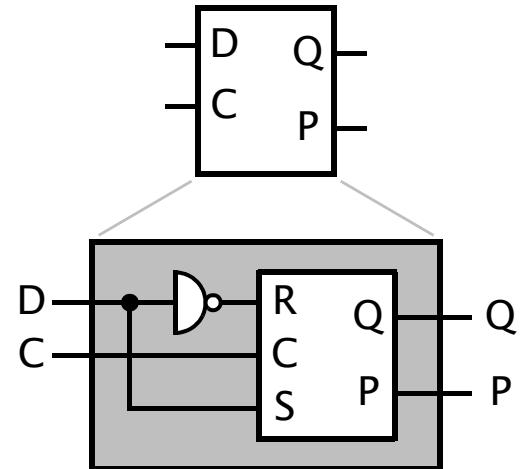


- ♦ synchrones, pegelgesteuertes D-Flipflop

- ein (Daten-)Eingang D
- zwei Ausgänge: Q und P (mit  $P := Q'$ )
- Funktionstabelle

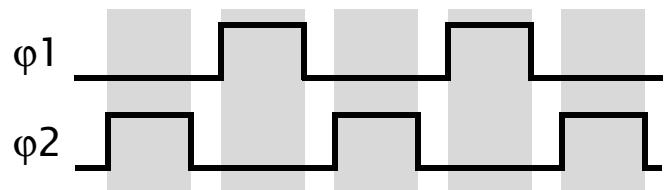
C	D	$Q^+$	$P^+$	Funktion
0	-	Q	P	halten
1	0	0	1	transparent
1	1	1	0	transparent

- Durch geeignete Beschaltung ( $S \neq R$ ) wird die ungültige Eingangskombination vermieden.
- Solange das Taktsignal aktiv ist ( $C=1$ ), ist das D-Flipflop für das Datensignal D *transparent*, d.h. der Ausgang folgt dem Eingang.



- ◆ Generelles Problem bei pegelgesteuerten Flipflops
  - In der aktiven Taktphase ( $C=1$ ) sind alle synchronen, pegelgesteuerten Speicherelemente transparent, d.h. eine Änderung am Eingang kann sich direkt auf den Ausgang auswirken.
  - Problematische Anwendung in Schaltwerken mit Rückkopplungen bei langen aktiven Taktphasen und kurzen Signallaufzeiten können asynchrone Signalschleifen entstehen, die zum nicht deterministischen Verhalten eines Schaltwerks führen können.
- ◆ Maßnahmen zur Beseitigung der Transparenz
  - Verkürzung der aktiven Taktphase  
die aktive Taktphase ( $C=1$ ) wird so kurz wie möglich gehalten (kürzer als die kürzeste Signallaufzeit),  $\Rightarrow$  asymmetrischer Takt mit kurzen aktiven Phasen und langen nicht aktiven Phasen.

- ◆ Maßnahmen zur Beseitigung der Transparenz
  - Zusicherung von Stabilität der Eingangssignale während der aktiven Taktphase
  - Einführung eines Zweiphasentaktes mit nicht überlappenden Taktphasen  $\varphi_1$  und  $\varphi_2$ 
    - möglich im Vollkunden-/ASIC-Design
    - schwierig in Systemen mit diskreten Bauteilen oder FPGA/CPLD-Bausteinen, weil solche Systeme i.d.R mit einen zentralen Taktgeber arbeiten
  - Entkopplung der Eingänge von den Ausgängen über
    - zweiflankengesteuerte Flipflops
    - einflankengesteuerte Flipflops

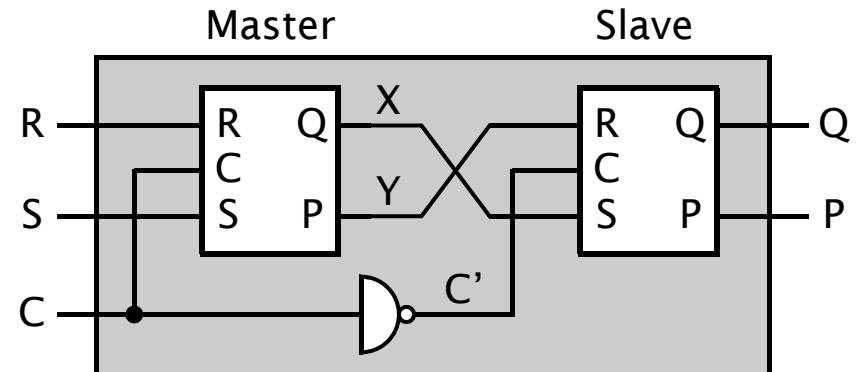


- ♦ zweiflankengesteuertes RS-Flipflop

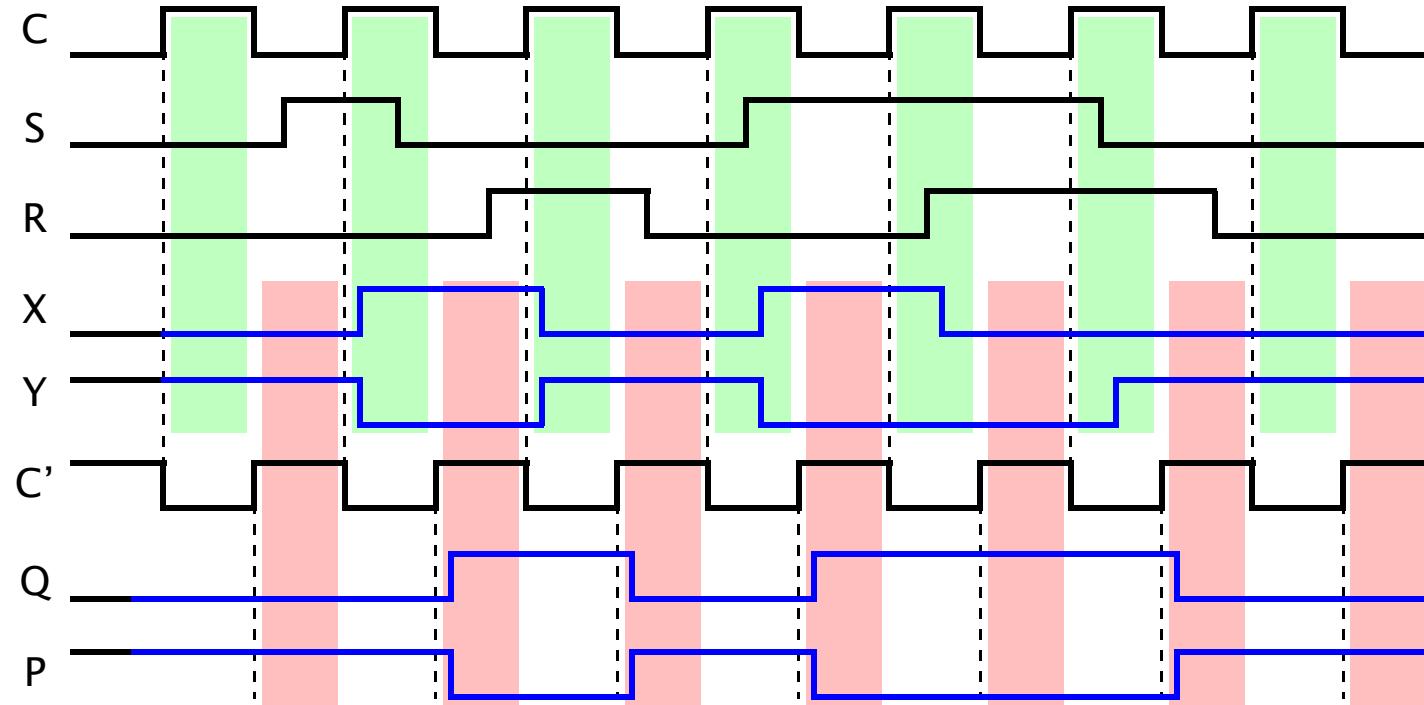
- Aufbau: zwei synchrone, pegelgesteuerte RS-Flipflops hintereinander geschaltet, angesteuert mit einem komplementären Taktsignal

- Funktionsweise:  
Während der aktiven Taktphase ( $C=1$ ) ist das erste Flipflop (Master) transparent, und das zweite Flipflop (Slave) gesperrt. In der nicht aktiven Taktphase ( $C=0$ ) ist das genau umgekehrt.

- Auf diese Weise sind beide Flipflops durch ein komplementäres Takt- signal wechselseitig verriegelt, und nie gleichzeitig durchgeschaltet.
  - Ein nach dem Master-Slave-Prinzip arbeitendes Flipflop kann aus synchronen, pegelgesteuerten RS-, D- und JK-Flipflops aufgebaut werden.



- ◆ Timing-Analyse des RS-Master-Slave-Flipflops

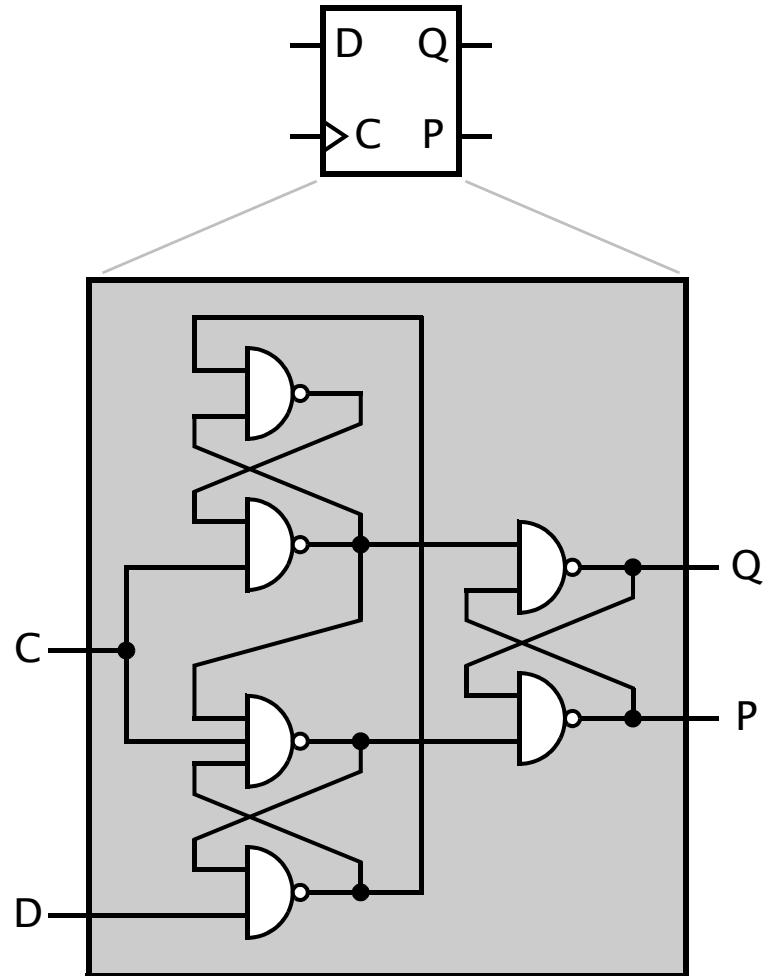
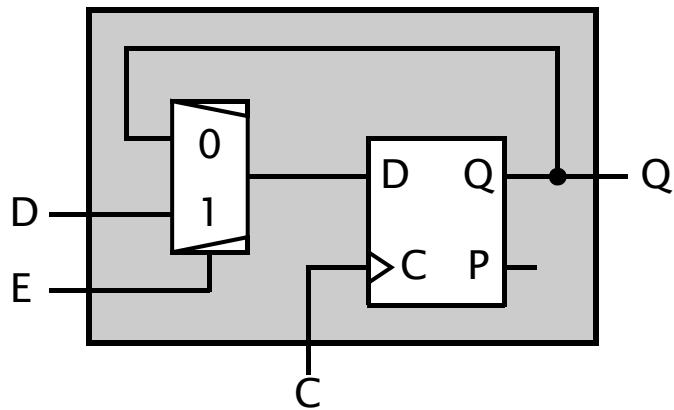


- ♦ einflankengesteuertes D-Flipflop

- Funktionstabelle und Aufbau

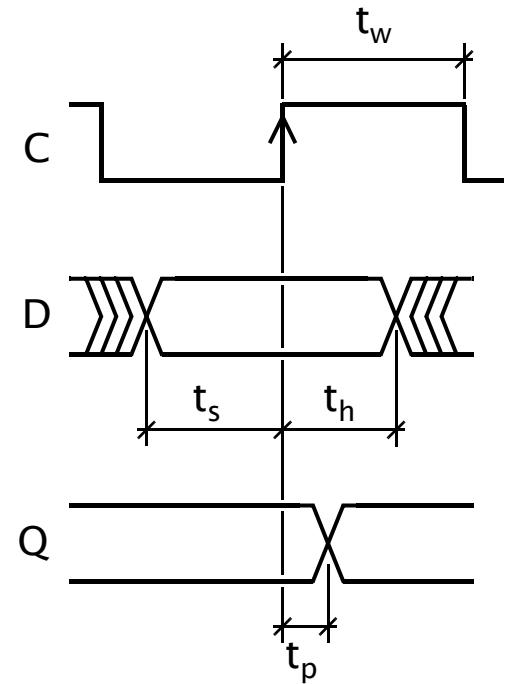
C	E	D	$Q^+$	$P^+$
0	-	-	Q	P
↑	0	-	Q	P
↑	1	0	0	1
↑	1	1	1	0

- D-Flipflop mit Enable-Signal



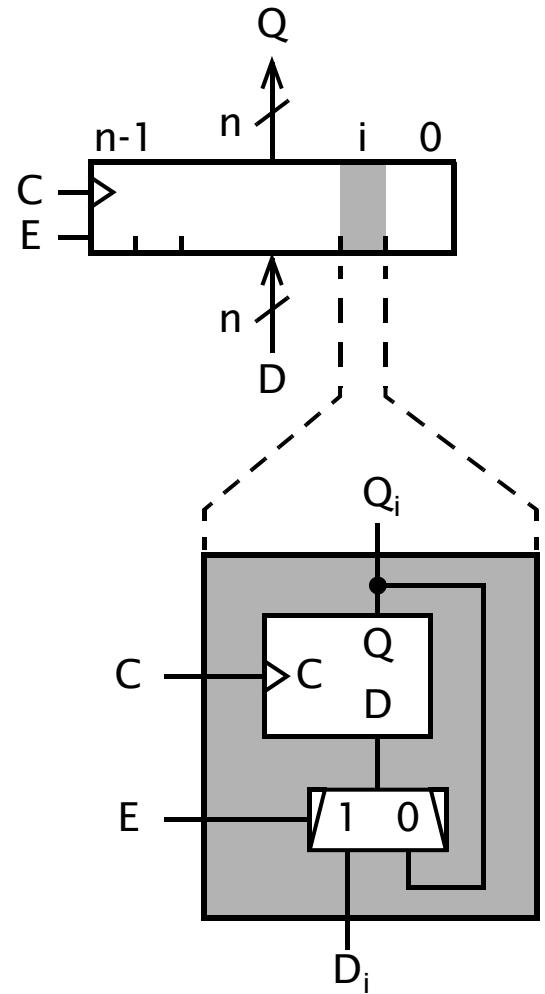
- ◆ Komplexe digitale Systeme werden vorwiegend als synchrone Systeme entwickelt.
- ◆ Typischerweise werden solche Systeme von einem globalen Taktsignal versorgt, d.h. es existiert nur ein Takt signal, das von allen Speicherelementen gleichermaßen benutzt wird.
- ◆ Flankengesteuerte Flipflops machen den Entwurf von Schaltwerken besonders übersichtlich und werden deshalb bevorzugt in programmierbaren Logikbausteinen (CPLDs, FPGAs) eingesetzt.
- ◆ Damit Eingangssignale in Speicherelemente sicher und synchron, d.h. gleichzeitig übernehmen werden, müssen diese Signale bestimmte zeitliche Kriterien erfüllen.

- ♦ Zeitliche Kenndaten von Flipflops
  - $t_w$  - pulse width (Impulsbreite): minimale Zeitspanne der aktiven Taktphase
  - $t_s$  - setup time (Vorbereitungszeit): minimale Zeitspanne, in der Daten am Eingang eines Flipflops vor dem Erscheinen der Taktflanke stabil bleiben müssen.
  - $t_h$  - hold time (Haltezeit): minimale Zeitspanne, in der Daten am Eingang eines Flipflops nach der Übernahme mit einer Taktflanke noch stabil bleiben müssen.
  - $t_p$  - propagation time clock to output (Verzögerung): maximale Zeitspanne von der Taktflanke, mit der Daten ins Flipflop übernommen wurden, bis zum Zeitpunkt, an dem die Daten am Ausgang des Flipflops erscheinen.



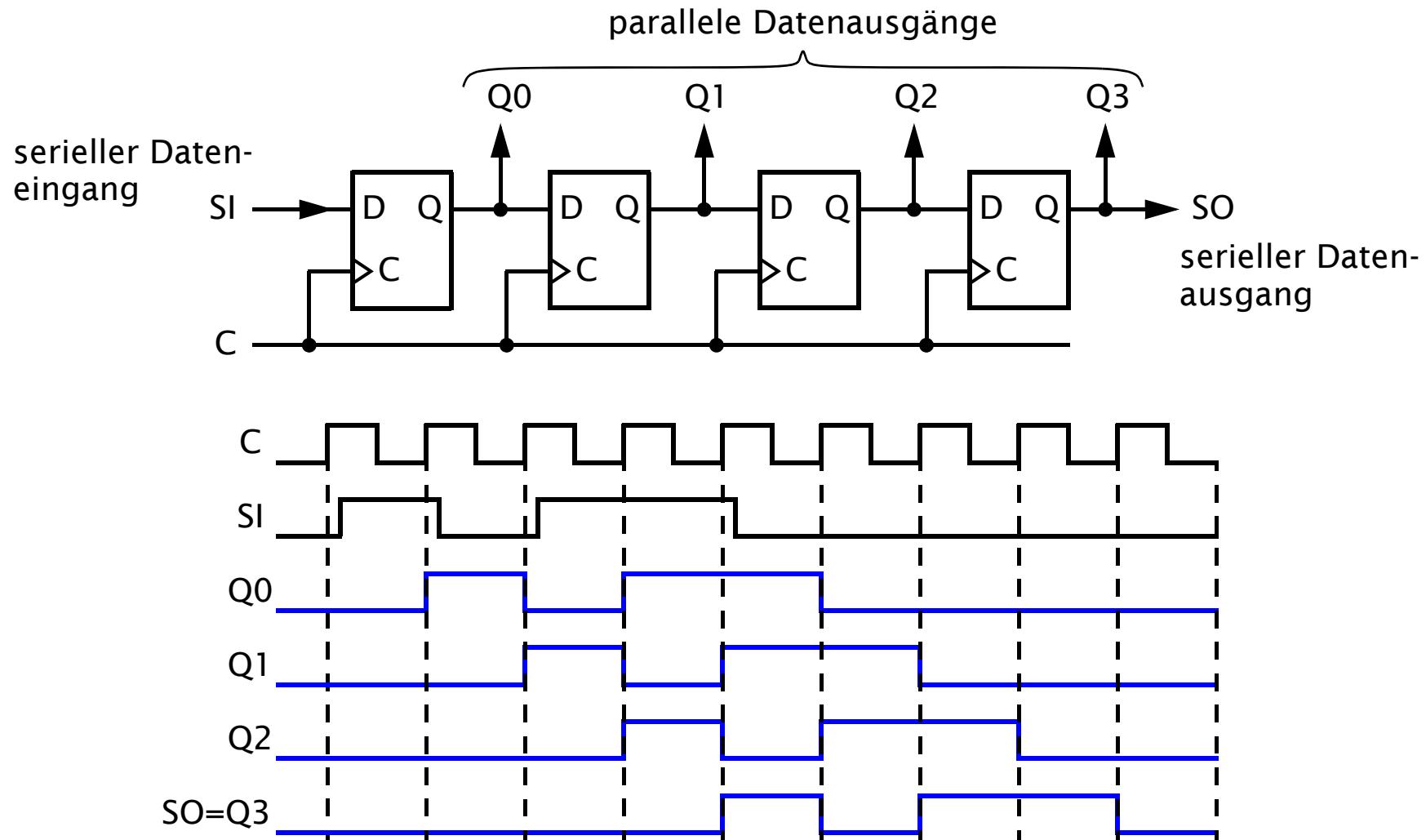
- ◆ n-Bit-Register

- Eine parallele Anordnung von Flipflops, i.d.R zur Speicherung von binären Werten, die für eine bestimmte Zeitspanne zur Weiterverarbeitung bereit gehalten werden.
- Im allgemeinen werden Register aus flankengesteuerten D-Flipflops mit einem Enable-Signal (E) aufgebaut.
- Einzelne Flipflops sind über ein gemeinsames Taktsignal (C) miteinander verbunden und arbeiten völlig synchron.
- In einem n-Bit-Register kann ein n-stelliger Binärwert gespeichert werden. Der Inhalt wird parallel ein- und ausgegeben.



- ◆ n-Bit-Schieberegister (Shift-Register)
  - Eine serielle Anordnung von Flipflops in der Form einer Ketten- schaltung, d.h. der Ausgang eines Flipflops ist mit dem Daten- eingang des folgenden Flipflops verbunden.
  - Der Dateneingang des ersten Flipflops in der Kettenschaltung ist der serielle Dateneingang SI des Schieberegisters, und der Aus- gang des letzten Flipflops ist der Datenausgang SO des Schiebe- registers.
  - Alle Flipflops sind über ein gemeinsames Taktsignal miteinander verbunden. In jedem Takt werden Binärwerte um eine Position nach rechts (links) geschoben.
  - Anwendungen:
    - Seriell-/Parallelwandlung
    - Erzeugung von Pseudozufallszahlen (LFSR – linear feedback shift register, linear rückgekoppeltes Schieberegister)
    - zyklische Redundanzprüfung (CRC – Cyclic Redundancy Check)

- ◆ 4-Bit-Seriell-/Parallel-Schieberegister



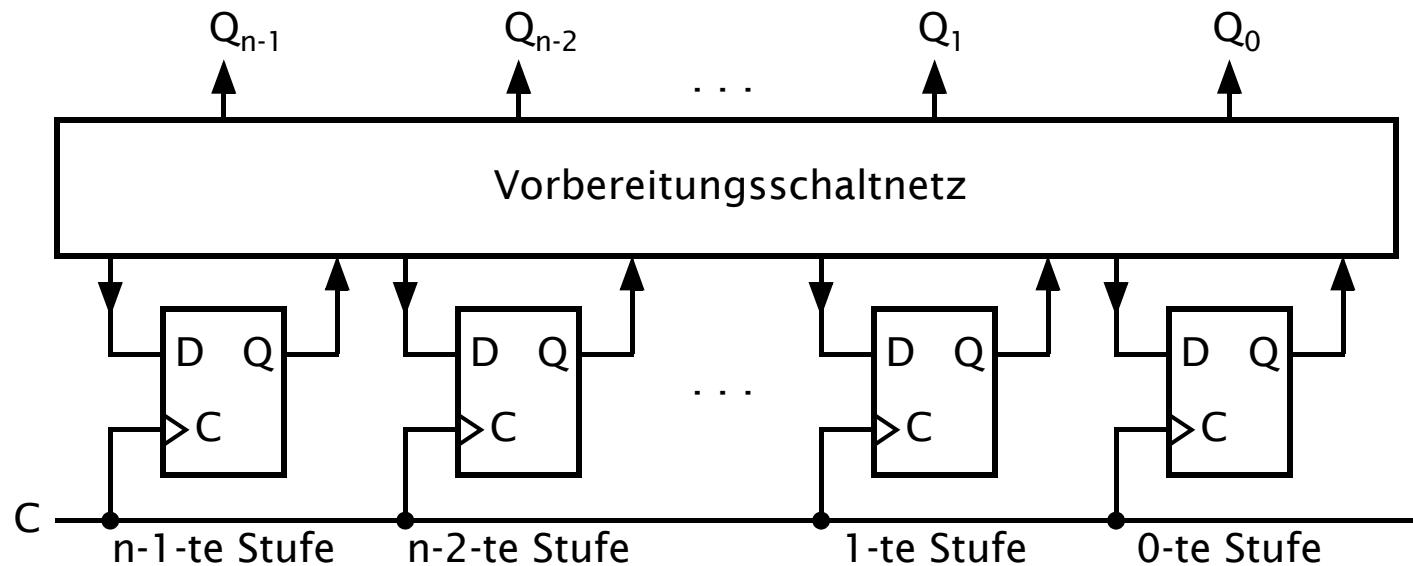
- ◆ **Zähler**
  - Schaltwerke, die Takt-/Steuerimpulse zählen können.
  - Schaltwerke, die eine eindeutige Zuordnung von Zählimpulsen am Eingang zu internen Flipflop-Zuständen ermöglichen.
  - Die internen Zustände müssen dabei nicht unbedingt einer gängigen, bekannten Zahlendarstellung entsprechen.
  - Zählfunktionen sind in vielen Steuerungsvorgänge als Teilaufgaben enthalten. Typische Aufgaben dieser Art sind:
    - die n-malige Wiederholung eines Vorganges
    - die Auslösung einer Reaktion nach n-maligem Auftreten eines bestimmten Ereignisses
    - präzise Zeit-/Positionsmessung
- ◆ **Klassifikation von Zählern**
  - Nach der Art der Darstellung des Zählstandes
    - Dualzähler: der Zählerstand wird als Dualzahl dargestellt,
    - BCD-Zähler: der Zählerstand wird pro Dezimalstelle separat dargestellt,
    - Darstellungen nach anwendungsspezifischen Codierungen.

## ♦ Klassifikation von Zählern

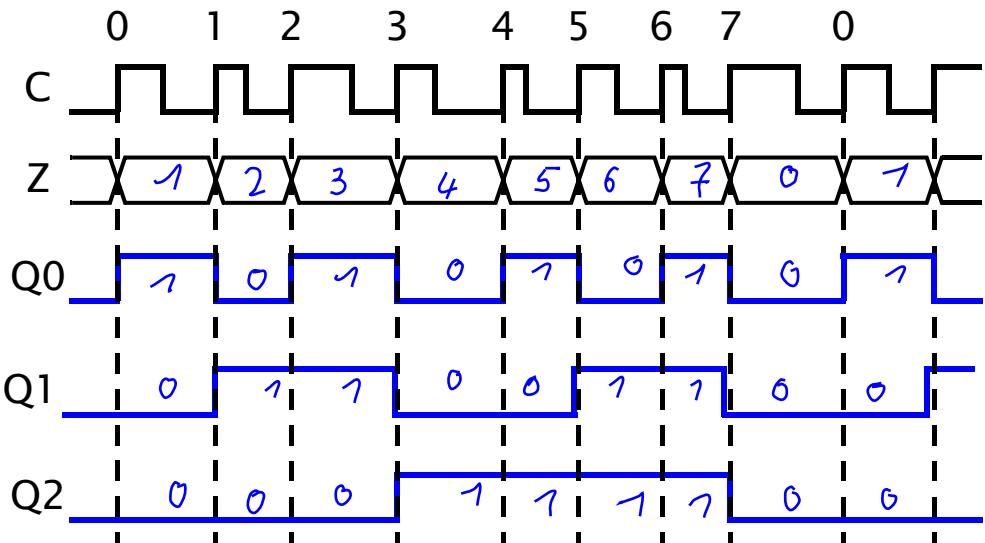
- Nach der Art der Ansteuerung
  - synchrone Zähler: alle FFs werden gleichzeitig mit Zählimpulsen versorgt,
  - asynchrone Zähler: mindestens ein FF enthält ein Taktsignal, das innerhalb des Schaltwerks (Zähler) generiert wurde,
  - semisynchrone Zähler: asynchrone Zähler, die abschnittsweise synchron arbeiten, z.B. bei der Kaskadierung von Zählern,
  - programmierbare Zähler: der Zählerstand lässt sich parallel laden und so von einem neu definierten Zählerstand weiter zählen.
- Nach der Zählrichtung
  - Vorwärtzähler mit  $Q := (Q + 1) \bmod m$
  - Rückwärtzähler mit  $Q := Q - 1$
  - umschaltbare Vor-/Rückwärtzähler.
- Nach der Art der Anordnung der Flipflops
  - Ringzähler: Schieberegister, in dem der Inhalt zyklisch verschoben wird, z.B. 4-Bit-Ringzähler: 1000, 0100, 0010, 0001, 1000, ...,
  - Johnson-Zähler: besondere Form des Ringzählers, z.B. 3-Bit-Johnson-Zähler: 000, 100, 110, 111, 011, 001, 000, ....

## ◆ Grundstruktur eines synchronen Zählers

- Im allgemeinen bestehen synchrone Zähler aus mehreren Stufen.
- Jede Stufe wird durch ein flankengesteuertes D-Flipflop realisiert.
- Ein zusätzliches (Vorbereitungs-)Schaltnetz übernimmt logische Verknüpfungen zur Ansteuerung der Flipflop-Eingänge und sofern notwendig der Ausgangsvariablen  $Q_{n-1}$ ,  $Q_{n-2}$ , ...,  $Q_1$ ,  $Q_0$ .



- ◆ Entwurf eines 3-Bit-Vorwärtszählers (als Dualzähler)
  - Ein Zähler soll steigende (positive) Flanken eines Eingangssignals C erfassen. An den Ausgängen Q2, Q1 und Q0 des Zählers Z soll der momentane Wert angezeigt werden.
  - Impulsplan und Funktionstabelle



Nr	Q2	Q1	Q0	Q2 <sup>+</sup>	Q1 <sup>+</sup>	Q0 <sup>+</sup>
0	0	0	0	0	0	1
1	0	0	1	0	1	0
2	0	1	0	0	1	1
3	0	1	1	1	0	0
4	1	0	0	1	0	1
5	1	0	1	1	1	0
6	1	1	0	1	1	1
7	1	1	1	0	0	0

- ◆ Entwurf eines 3-Bit-Vorwärtszählers

	Q1, Q0		00	01	11	10
Q2	0	0	0	1	1	0
	1	1	1	0	0	1
	Q2 <sup>+</sup>					

$$\begin{aligned}
 Q2^+ &= Q2 \cdot Q1' + Q2 \cdot Q0' \\
 &\quad + Q2' \cdot Q1 \cdot Q0 \\
 &= Q2 \cdot (Q1 \cdot Q0)' + Q2' \cdot Q1 \cdot Q0 \\
 &\xrightarrow{\text{ell}} Q2 \oplus (Q1 \cdot Q0)
 \end{aligned}$$

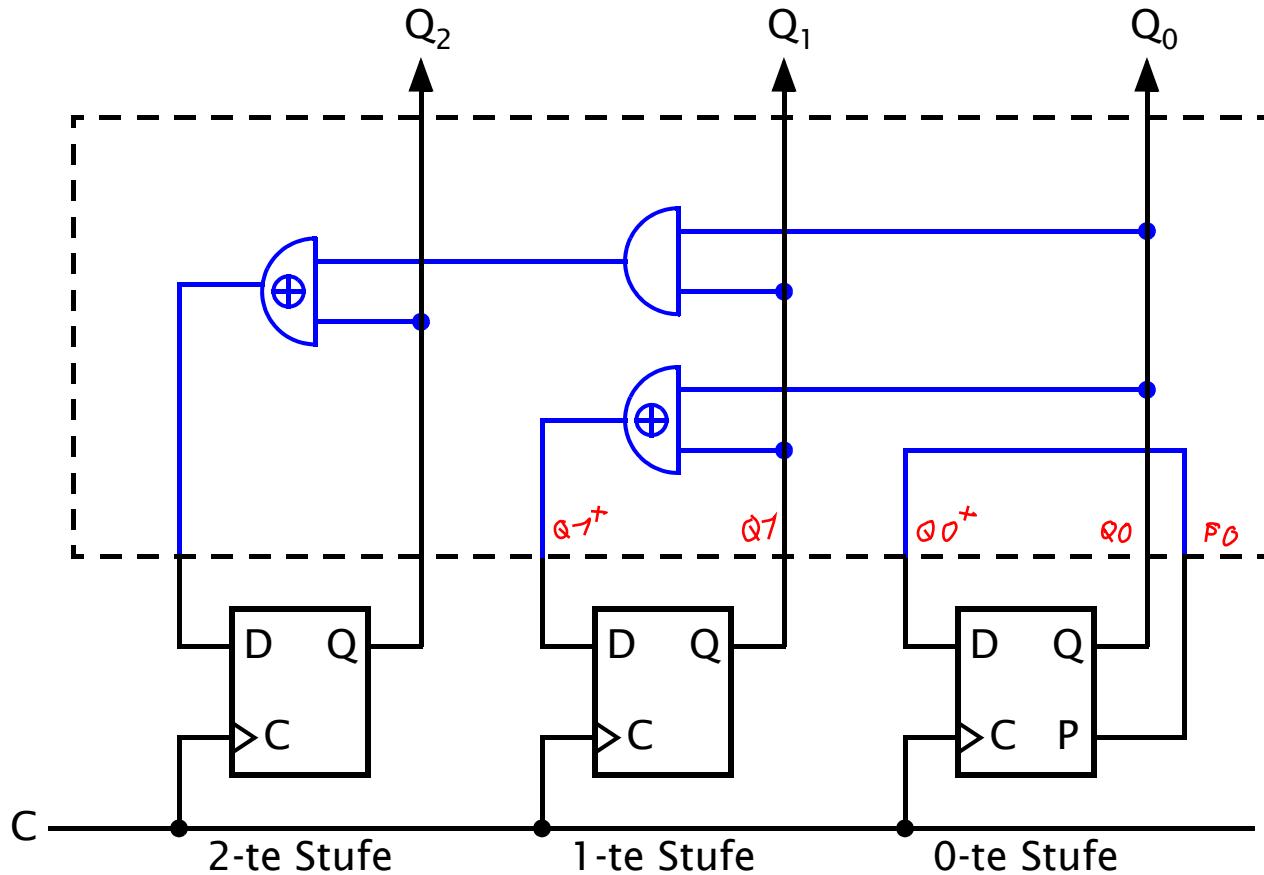
	Q1, Q0		00	01	11	10
Q2	0	0	0	1	1	1
	1	1	1	0	0	1
	Q1 <sup>+</sup>					

$$\begin{aligned}
 Q1^+ &= Q1 \cdot Q0' + Q1' \cdot Q0 \\
 &= Q1 \oplus Q0
 \end{aligned}$$

	Q1, Q0		00	01	11	10
Q2	0	0	0	0	1	1
	1	1	0	0	0	1
	Q0 <sup>+</sup>					

$$Q0^+ = Q0' = P_0$$

- ◆ Entwurf eines 3-Bit-Vorwärtszählers



## ♦ Modulo-m-Zähler

- Die aus der Mathematik bekannte Modulo-Funktion (mod) liefert den Rest aus der Division zweier ganzer Zahlen,
- Ein Modulo-m-Zähler ist ein Zähler, der i.d.R. zyklisch von 0 bis  $m-1$  vorwärts zählt, d.h. nachdem der Zählerstand den Wert  $m-1$  erreicht hat, wird der Zählvorgang wieder bei 0 fortgesetzt. Der Zählerstand bleibt immer kleiner als  $m$ .  
z.B. ein Modulo-5-Zähler liefert nur die Werte: 0, 1, 2, 3, 4
- Ein Modulo-m-Zähler lässt sich aus einem n-Bit-Dualzähler mit der Bedingung  $n < \log_2(m)$  aufbauen.  
z.B. aus einem 4-Bit-Dualzähler kann man folgende Modulo-m-Zähler aufbauen: mod 2, mod 3, mod 4, ..., mod 14 und mod 15.
- Die Bezeichnung „Modulo-m-Zähler“ wird manchmal auch in Verbindung mit Zählern verwendet, die nicht als Dualzähler arbeiten.
- Die Struktur eines Modulo-m-Zählers basiert auf der Grundstruktur eines synchronen Zählers.

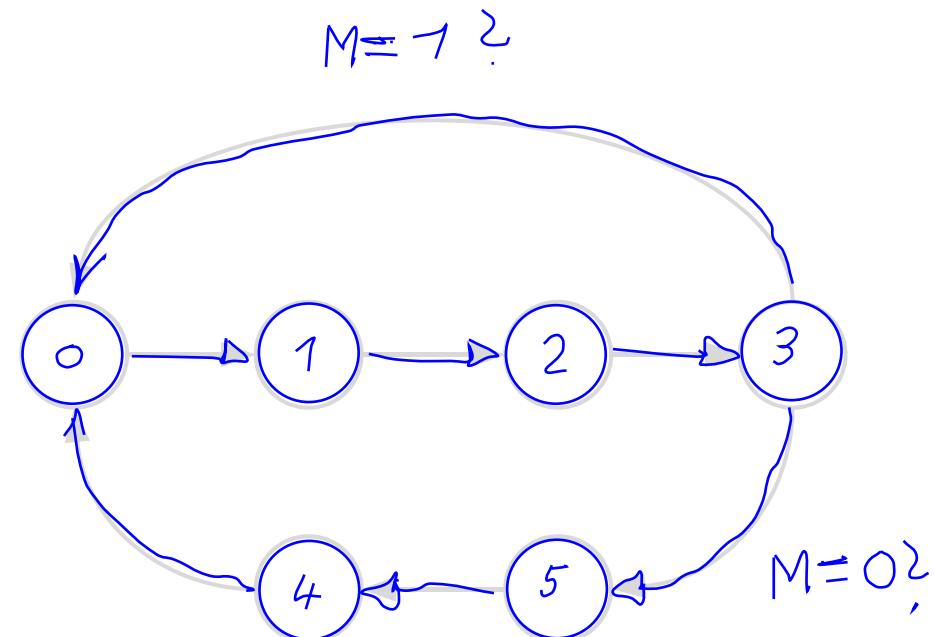
- Entwurf eines einfachen Modulo-m-Zählers

- Ein Zähler soll in der Abhängigkeit von einem Steuersignal M entweder modulo 6 ( $M=0$ ) oder modulo 4 ( $M=1$ ) zählen.
- Funktionstabelle und Zustandsgraph

Nr	M	Q2	Q1	Q0	$Q2^+$	$Q1^+$	$Q0^+$
0,8	0	— <sub>1</sub> <sup>0</sup>	0	0	0	0	1
1,9	1	— <sub>1</sub> <sup>0</sup>	0	0	1	0	1
2,10	2	— <sub>1</sub> <sup>0</sup>	0	1	0	1	1
3	3	1	0	1	1	0	0
3	3	0	0	1	1	0	0
4,12	4	— <sub>1</sub> <sup>0</sup>	1	0	0	1	0
5,13	5	— <sub>1</sub> <sup>0</sup>	1	0	1	0	0

mod 6

$\{0, 1, 2, 3, 4, 5\}$



mod 4  $\{0, 1, 2, 3\}$

- Entwurf eines einfachen Modulo-m-Zählers

M,Q2	Q1,Q0	00	01	11	10
00	0	0	1	0	0
01	1	0	—	—	6
11	1	0	—	—	14
10	0	0	0	0	10

Q2<sup>+</sup>

$$\begin{aligned}
 Q2^+ &= Q2 \cdot Q0' \\
 &\quad + M' \cdot Q1 \cdot Q0 \\
 &= Q2 \cdot P0 \\
 &\quad + M' \cdot Q1 \cdot Q0
 \end{aligned}$$

M,Q2	Q1,Q0	00	01	11	10
00	0	1	0	1	1
01	0	0	—	—	—
11	0	0	—	—	—
10	0	1	0	1	1

Q1<sup>+</sup>

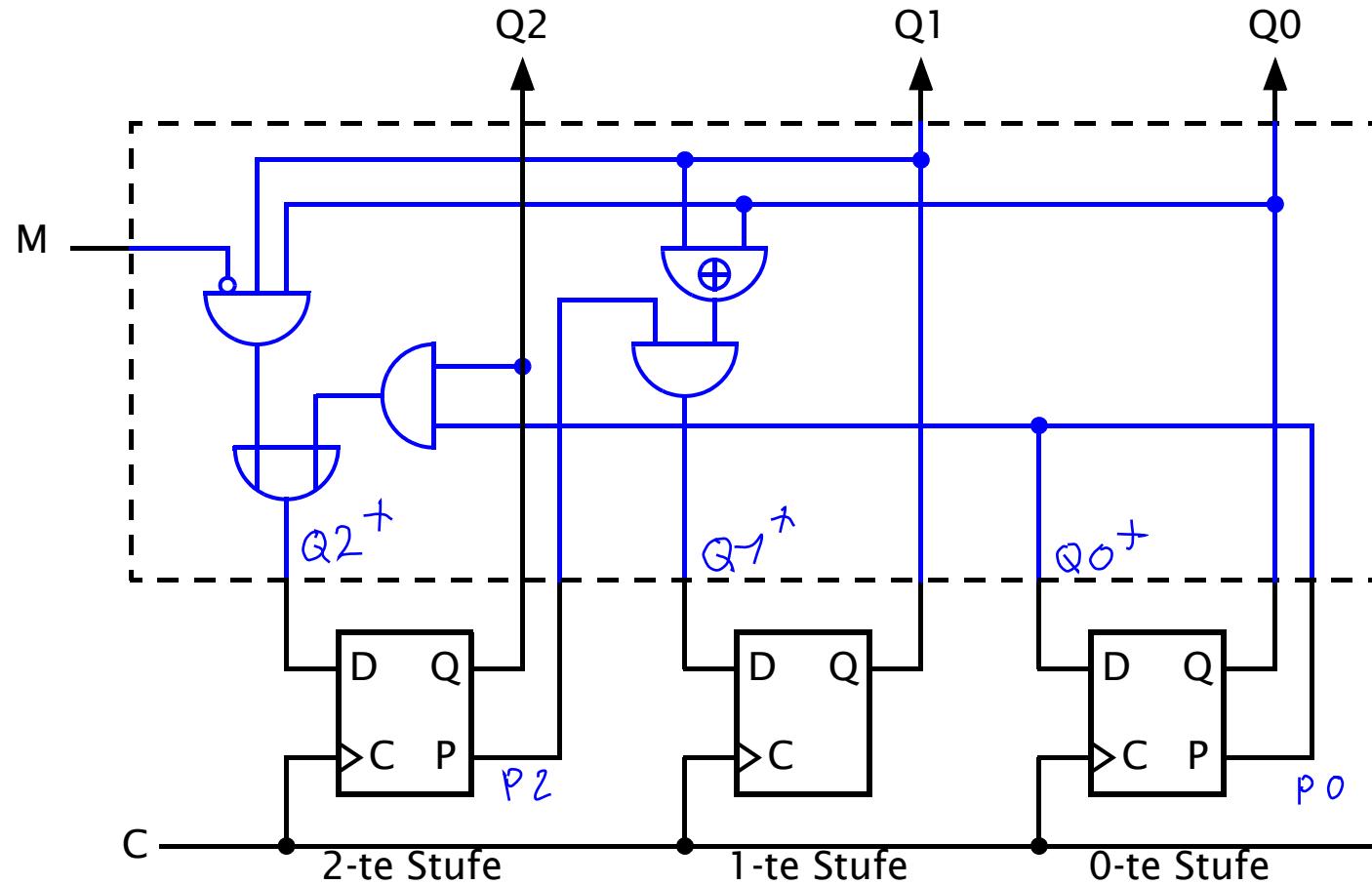
$$\begin{aligned}
 Q1^+ &= Q2' \cdot Q1' \cdot Q0 \\
 &\quad + Q2' \cdot Q1 \cdot Q0' \\
 &= P2 \cdot (Q1 \oplus Q0)
 \end{aligned}$$

M,Q2	Q1,Q0	00	01	11	10
00	1	0	0	1	1
01	1	0	—	—	—
11	1	0	—	—	—
10	1	0	0	1	1

Q0<sup>+</sup>

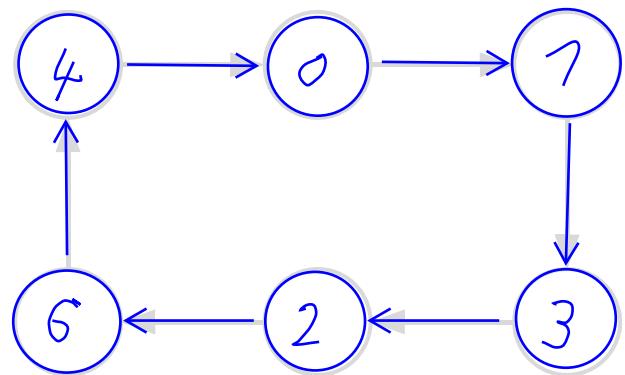
$$Q0^+ = Q0' = P0$$

- ◆ Entwurf eines einfachen Modulo-m-Zählers



- ♦ Entwurf eines selbstkorrigierenden Zählers
  - Ein zyklischer, selbstkorrigierender 3-Bit-Zähler mit einer einschrittig codierten Zählsequenz  $\{000_2, 001_2, 011_2, 010_2, 110_2, 100_2\}$  soll auf der Basis von D-Flipflops entworfen werden.
  - Funktionstabelle und Zustandsgraph

Nr	Q2	Q1	Q0	$Q2^+$	$Q1^+$	$Q0^+$
0	0	0	0	0	0	1
1	0	0	1	0	1	1
3	0	1	1	0	1	0
2	0	1	0	1	1	0
6	1	1	0	1	0	0
4	1	0	0	0	0	0
5	1	0	1	?	?	?
7	1	1	1	?	?	?



- ◆ Entwurf eines selbstkorrigierenden Zählers
  - Ein selbstkorrigierender Zähler zeichnet sich dadurch aus, daß er in der Lage ist, ungültige, fehlerhafte Zählzustände automatisch und selbstständig zu korrigieren.
  - Als fehlerhafte Zählzustände bezeichnet man solche Zählzustände, die der Zähler unter normalen Umständen nicht annehmen darf, also solche, die in der Sequenz der „gültigen“ Zählzustände nicht aufgelistet sind.
  - Fehlerhafte Zählzustände können sich aufgrund von elektrischen Störungen in digitalen Systemen oder nach einer Einschaltphase (Power-On) einstellen.
  - Beim Entwurf selbstkorrigierender Zähler ist zu beachten, daß fehlerhafte Zählzustände nach Möglichkeiten immer unter dem Aspekt des geringsten Realisierungsaufwands korrigiert werden sollen.

- ◆ Regelwerk zum Entwurf eines selbstkorrigierenden Zählers
  1. Für die „gültige“ Zählsequenz wird eine Funktionstabelle aufgestellt und diese dann in entsprechende KV-Diagramme überführt. Fehlerhafte Zählzustände bleiben vorläufig unberücksichtigt.
  2. Der geringste Realisierungsaufwand bei der Korrektur fehlerhafter Zählzustände lässt sich mit Hilfe von KV-Diagrammen ermitteln. Dort werden noch unbestimmte Felder so mit Nullen und Einsen belegt, daß dadurch bei der Minimierung möglichst die größten Gruppen gebildet werden können.
  3. Anschließend wird mit Hilfe des Zustandsgraphen geprüft, ob die dadurch erzeugte Belegung der fehlerhaften Zählzustände dazu führt, daß diese einen Übergang in die Sequenz der „gültigen“ Zählzustände haben.  
Wenn das nicht der Fall ist, ist die Belegung zu korrigieren und mit dem Schritt 2 fortzuführen.

- Entwurf eines selbstkorrigierenden Zählers

	Q1, Q0			
Q2	00	01	11	10
0	0 0	0 1	0 3	1 2
1	0 4	0 5	0 7	1 6

	Q1, Q0			
Q2	00	01	11	10
0	0 0	1 1	1 1	1 1
1	0 0	0 0	0 0	0 0

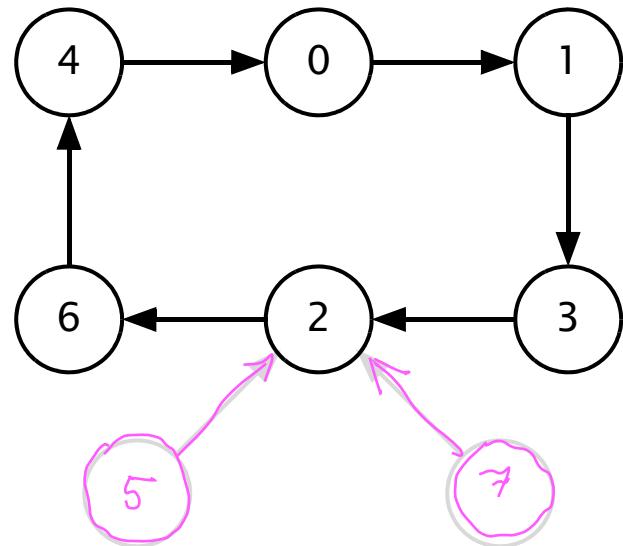
	Q1, Q0			
Q2	00	01	11	10
0	1 1	1 1	0 0	0 0
1	0 0	0 0	0 0	0 0

$$\begin{aligned} 5(101)_2 &\rightarrow (010)_2 = 2 \quad \checkmark \\ 7(111)_2 &\rightarrow (010)_2 = 2 \quad \checkmark \end{aligned}$$

$$Q0^+ = Q2' \cdot Q1' = p_2 \cdot p_1$$

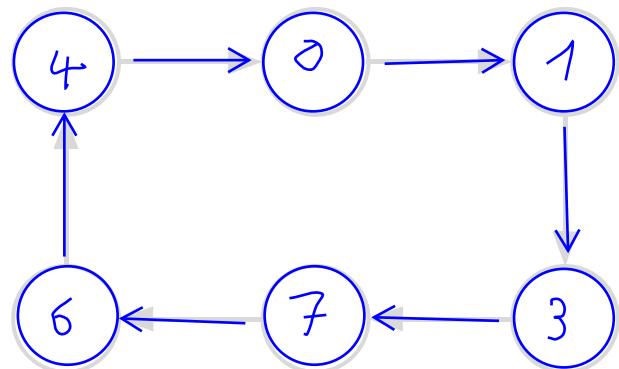
$$Q1^+ = Q0 + Q2' \cdot Q1 = Q0 + p_2 \cdot Q1$$

$$Q2^+ = Q1 \cdot Q0' = Q1 \cdot p_0$$



- ◆ Entwurf eines selbstkorrigierenden Zählers
  - Auf der Basis von D-Flipflops ist ein zyklischer, selbstkorrigierender 3-Bit-Johnson-Zähler mit folgender dezimalcodierten Zählsequenz {0, 1, 3, 7, 6, 4} zu entwerfen.
  - Funktionstabelle und Zustandsgraph

Nr	Q2	Q1	Q0	Q2 <sup>+</sup>	Q1 <sup>+</sup>	Q0 <sup>+</sup>
0	0	0	0	0	0	1
1	0	0	1	0	1	1
3	0	1	1	1	1	1
7	1	1	1	1	1	0
6	1	1	0	1	0	0
4	1	0	0	0	0	0
2	0	1	0	?	?	?
5	1	0	1	?	?	?



- ♦ Entwurf eines selbstkorrigierenden Zählers

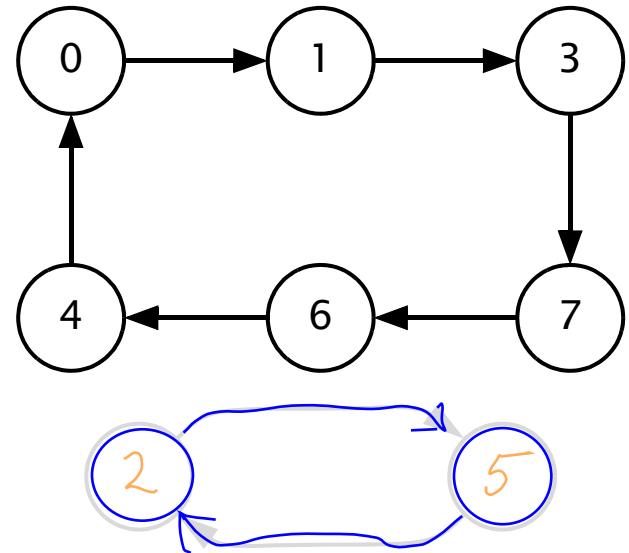
		Q1, Q0	
		00	01
Q2	0	0	0
	1	0	$\underline{(\bar{0})}_2$ 5
		1	1
			Q2 <sup>+</sup>

		Q1, Q0	
		00	01
Q2	0	0	1
	1	$\underline{(\bar{1})}_2$ 5	1
		1	0
			Q1 <sup>+</sup>

		Q1, Q0	
		00	01
Q2	0	1	1
	1	$\underline{(\bar{0})}_2$ 5	0
		1	0
			Q0 <sup>+</sup>

$$2 (010)_2 \rightsquigarrow (\bar{1}01)_2 = 5 \quad \times$$

$$5 (101)_2 \rightsquigarrow (010)_2 = 2 \quad \times$$



- Entwurf eines selbstkorrigierenden Zählers

		Q1, Q0			
		00	01	11	10
Q2	0	0	0	1	( $\bar{1}$ )
	1	0	( $\bar{1}$ )	1	1
		Q2 <sup>+</sup>			

		Q1, Q0			
		00	01	11	10
Q2	0	0	1	1	( $\bar{0}$ )
	1	0	( $\bar{1}$ )	1	0
		Q1 <sup>+</sup>			

		Q1, Q0			
		00	01	11	10
Q2	0	1	( $\bar{1}$ )	1	( $\bar{1}$ )
	1	0	( $\bar{0}$ )	0	0
		Q0 <sup>+</sup>			

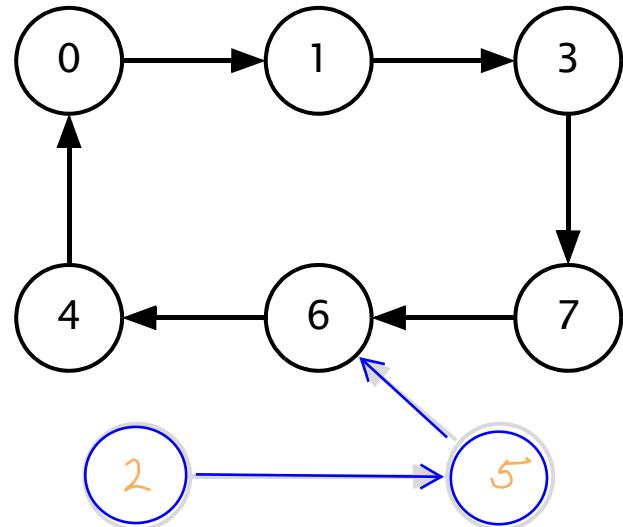
$$2(010) \rightsquigarrow (101)_2 = 5 \quad \checkmark$$

$$5(101) \rightsquigarrow (110)_2 = 6 \quad \checkmark$$

$$Q0^+ = Q2' = P2$$

$$Q1^+ = Q0$$

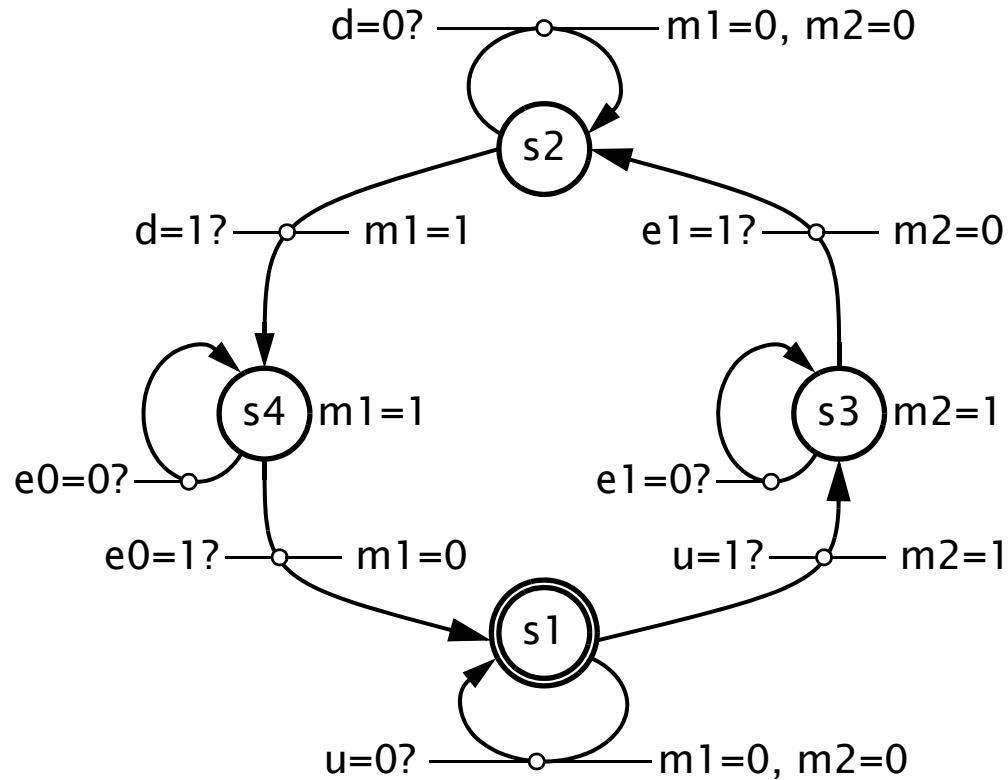
$$Q2^+ = Q1 + Q2 \cdot Q0$$



- ◆ Ein endlicher (boolescher) Automat (Finite State Machine, FSM) ist ein mathematisches Modell für eine Vorrichtung mit einer endlichen Anzahl interner Zustände, die eine Folge von booleschen Eingangswerten einliest, verarbeitet und eine Folge von booleschen Ausgangswerten erzeugt.
- ◆ Ein endlicher Automat ist durch 6-Tupel  
 $EA = (E, A, Z, Z_0, f, g)$  definiert, wo  
E endliche Eingabemenge (Eingangsalphabet, Menge aller möglichen Werte, die Eingangsvariablen annehmen können)  
A Ausgabemenge (Ausgangsalphabet, Menge aller möglichen Werte, die Ausgangsvariablen annehmen können)  
Z endliche Zustandsmenge  
 $Z_0$  Anfangszustand  
f (Zustands-)Übergangsfunktion für den Folgezustand  
g Ausgabefunktion (abhängig vom Automatentyp)

- ◆ Aufzugsteuerung modelliert als Automat
  - Eingabemenge E
    - vier Eingangsvariablen u, d, e0 und e1
    - jede Eingangsvariable kann entweder 0 oder 1 annehmen
    - zusammen ergeben sie also 16 mögliche Eingangskombinationen  
 $E(u, d, e0, e1) = \{(0000)_2, (0001)_2, \dots, (1110)_2, (1111)_2\}$
  - Ausgabemenge A
    - zwei Ausgangsvariablen m1 und m2
    - jede Ausgangsvariable kann entweder 0 oder 1 annehmen
    - daraus ergeben sich 4 mögliche Ausgangskombinationen  
 $A(m1, m2) = \{(00)_2, (01)_2, (10)_2, (11)_2\}$
  - Zustandsmenge
    - eine Zustandsvariable, die 4 symbolisch codierte Zustände annehmen kann  
 $Z = \{s1, s2, s3, s4\}$
  - Anfangszustand  
 $Z_0 = \{s1\}$

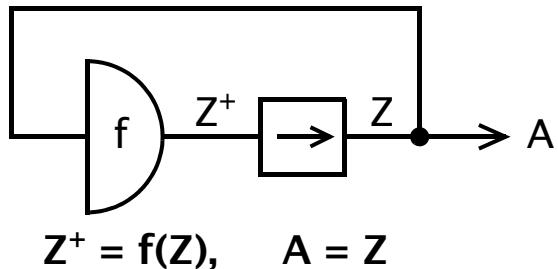
- ♦ Aufzugsteuerung modelliert als Automat
  - Tabellarische Notation der Übergangs- und Ausgabefunktionen
  - Übergangsfunktion  $Z^+ = f(Z, E)$
  - Ausgabefunktionen  $m1 = g1(Z, E)$  und  $m2 = g2(Z, E)$



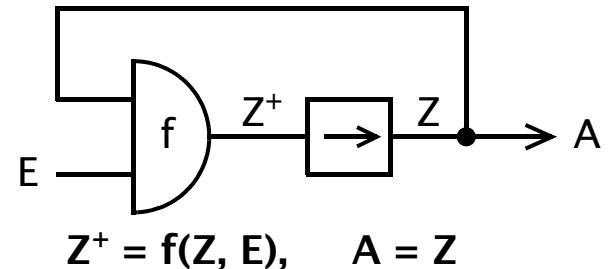
E(u, d, e0, e1)					$Z^+$	$m1$	$m2$
$Z$	$u$	$d$	$e0$	$e1$			
S1	0	—	—	—	S1	0	0
S1	1	—	—	—	S3	0	1
S3	—	—	—	0	S3	0	1
S3	—	—	—	1	S2	0	0
S2	—	0	—	—	S2	0	0
S2	—	1	—	—	S4	1	0
S4	—	—	0	—	S4	1	0
S4	—	—	1	—	S1	0	0

- ◆ Automatenmodelle

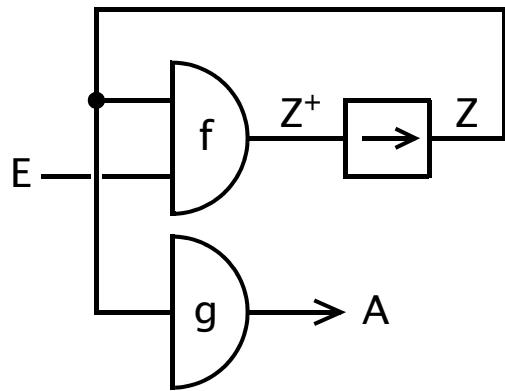
Autonomer Automat



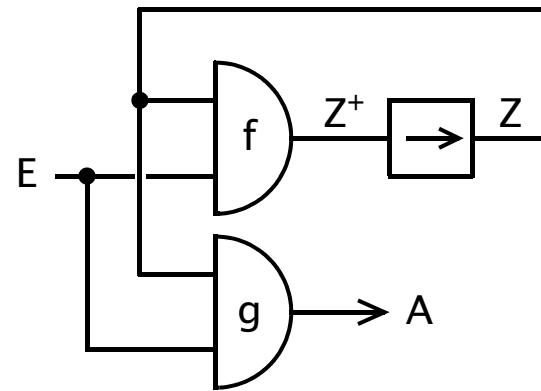
Medwedjew-Automat



Moore-Automat

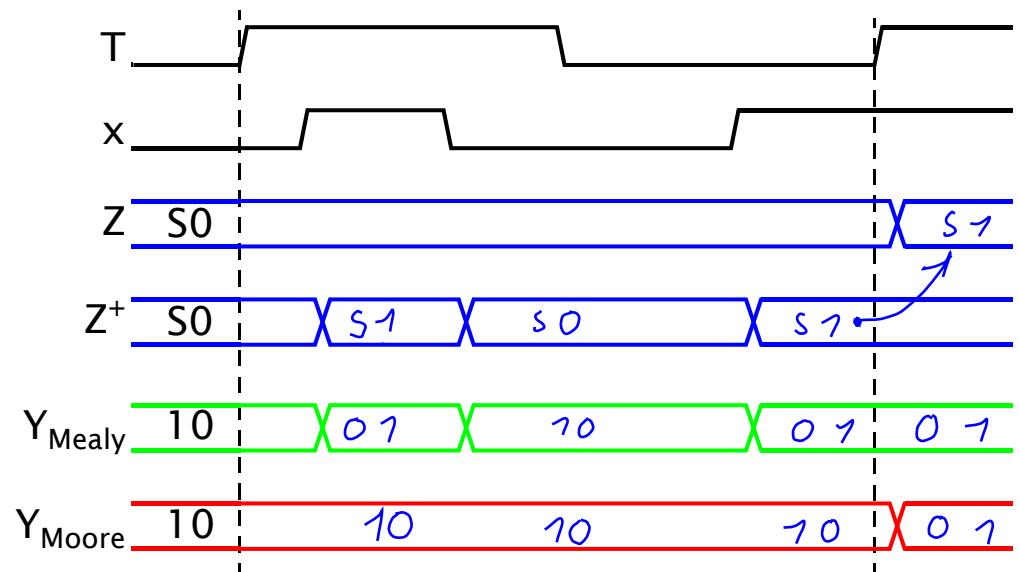
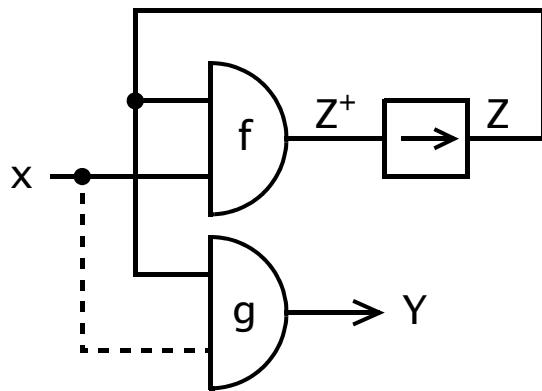
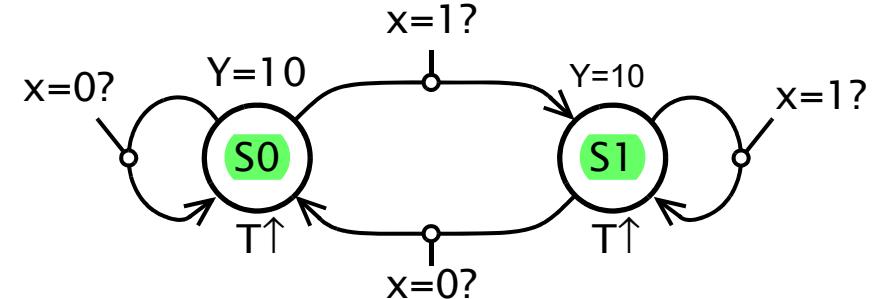
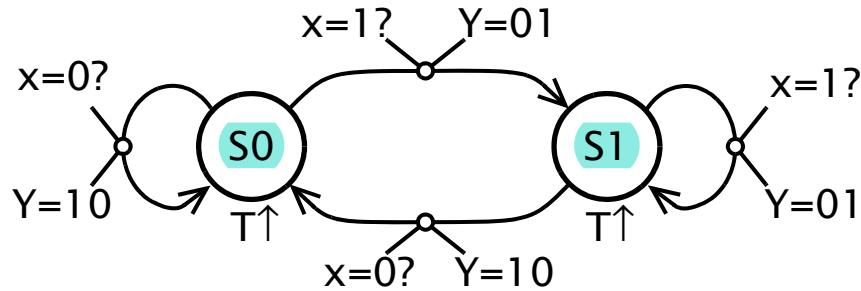


Mealy-Automat



- ◆ Das Verhalten eines Automaten wird im allgemein durch Übergangs- und Ausgabefunktionen beschrieben.
- ◆ Die Übergangsfunktion  $Z^+ = f(E, Z)$  gibt (abgesehen von autonomen Automaten) in Abhängigkeit von den momentanen Eingangswerten  $E$  und den aktuellen Werten der inneren Zustandsvariablen  $Z$  an, welche Werte die inneren Zustandsvariablen zum Folgezeitpunkt annehmen.
- ◆ Die Ausgabefunktion  $A = g(E, Z)$  bestimmt die Werte der Ausgangsvariablen in Abhängigkeit von den aktuellen Werten der inneren Zustandsvariablen  $Z$  und eventuell von den momentanen Eingangswerten  $E$ .

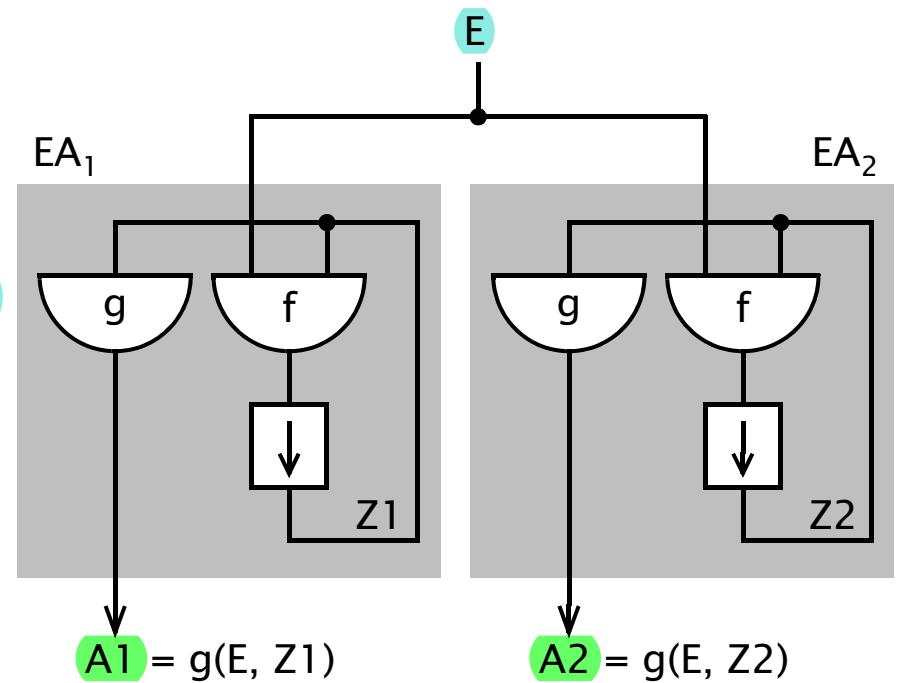
- Timing Analyse von Mealy- und Moore-Automaten



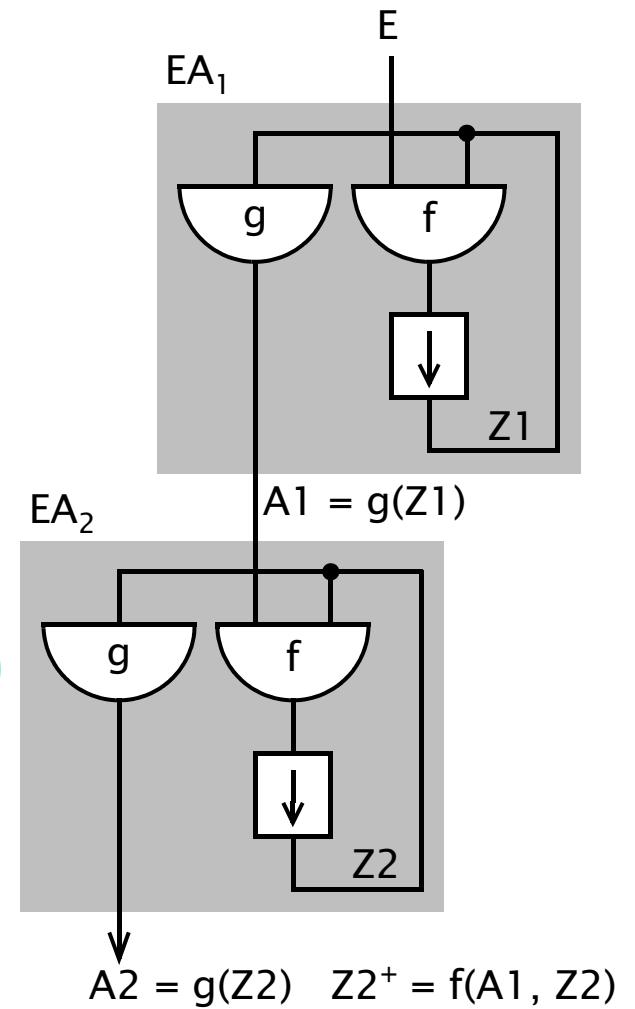
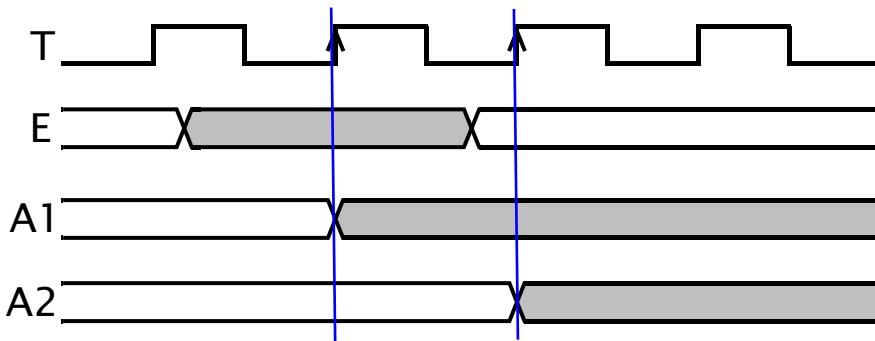
- ♦ Komposition von Automaten (Schaltwerken)
  - zwei oder mehrere endliche Automaten  $EA_i$  können auf verschiedene Arten miteinander kombiniert werden.

- ♦ Parallel Komposition:
  - zwei verschiedene Automaten  $EA_1$  und  $EA_2$  generieren dann verschiedene Ausgaben  $A1$  und  $A2$ , werden aber von den selben Eingaben  $E$  gespeist.

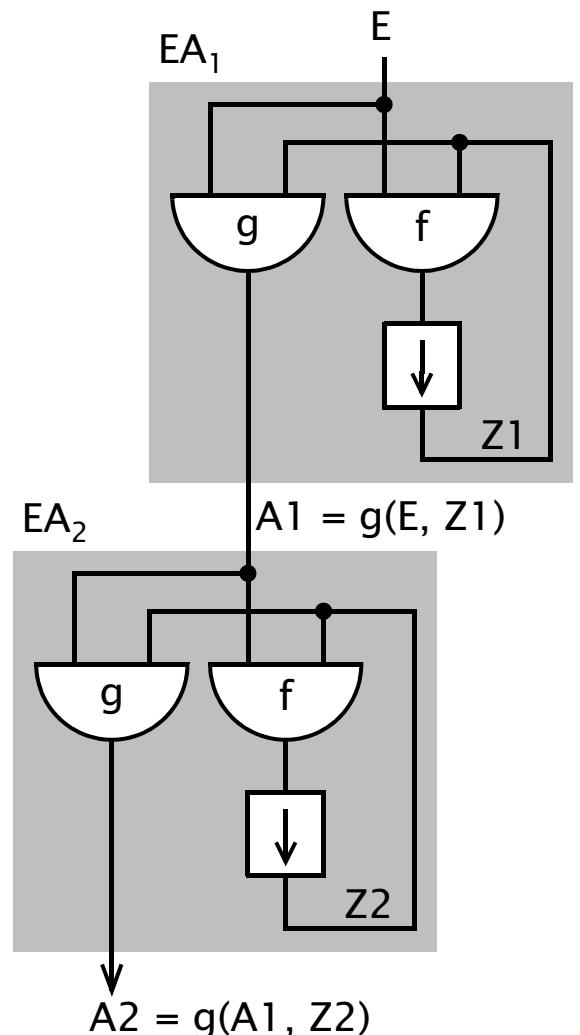
- ♦ Sequentielle Komposition:
  - Hintereinanderschaltung: die Ausgabe des ersten Automaten wird als Eingabe für den zweiten Automaten verwendet.



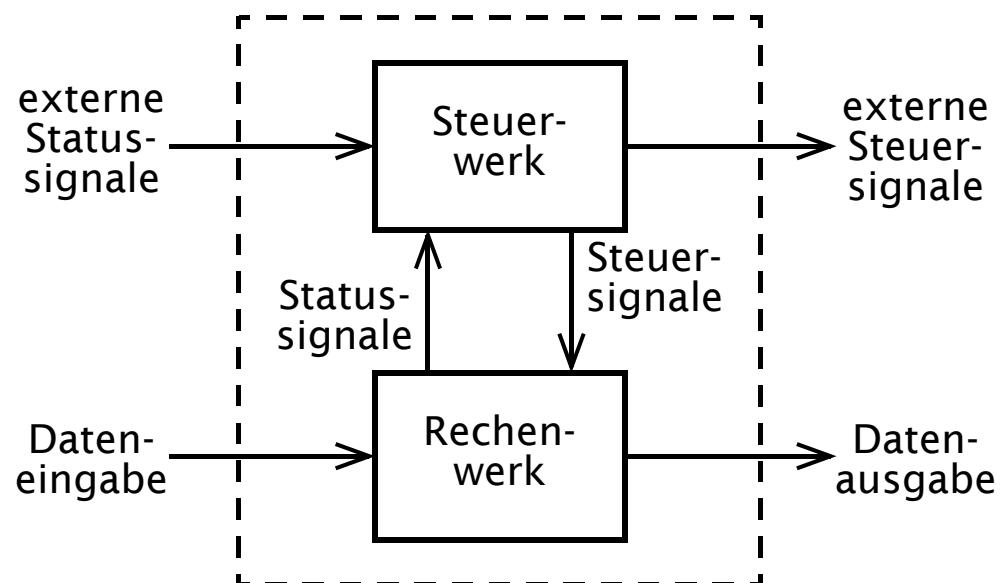
- ♦ sequentielle Komposition zweier Moore-Automaten
  - Die Ausgabe A1 aus dem ersten Moore-Automaten EA<sub>1</sub> ist erst nach einem Taktzyklus verfügbar.
  - Somit steht die Eingabe für den zweiten Moore-Automaten EA<sub>2</sub> erst für den zweiten Taktzyklus zur Verfügung.
  - Die Ausgabe A2 aus dem zweiten Moore-Automaten EA<sub>2</sub> ist gegenüber der Eingabe E um **zwei Taktzyklen verzögert**.



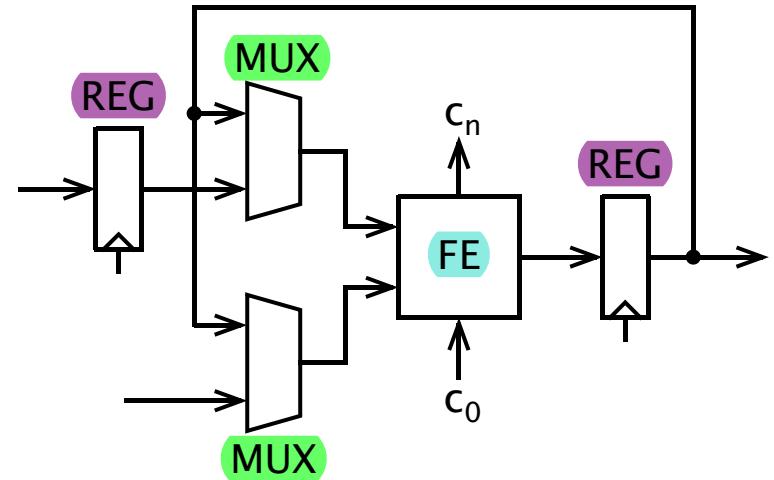
- ♦ sequentielle Komposition zweier Mealy-Automaten
  - Die Eingabe E des ersten Mealy-Automaten EA<sub>1</sub> wird durch das Ausgabeschaltnetz g geführt und dient als Eingabe (A1) für den zweiten Mealy-Automaten EA<sub>2</sub>.
  - Durch die unmittelbare Abhängigkeit der Ausgabe von den Eingabewerten kommt im allgemeinen zu einer Verlängerung der kombinatorischen Pfade ( $E \rightarrow A_1 \rightarrow A_2$ ).
  - Eine sequentielle Komposition mehrerer Mealy-Automaten muß wegen der Gatterlaufzeiten in den Ausgabeschaltnetzen mit einem um so langsameren Takt  $t_{cyc}$  betrieben werden, je mehr solche Automaten hintereinander geschaltet sind ( $t_{cyc} \geq \sum t_{pd}(g_i)$ ).



- ◆ Modularisierung bedeutet die logische Zerlegung komplexer Systeme in einfachere, leichter zu überschauende und somit auch leichter zu verstehende Subsysteme.
- ◆ Aufteilung eines digitalen Systems in
  - eine steuernde Einheit (Steuerwerk, Steuerpfad)
  - eine gesteuerte Einheit (Rechenwerk, Operationswerk, Datenpfad)
  - oft kein Steuerwerk in datenflußdominannten Anwendungen
  - oft kein Rechenwerk in Steuerungsaufgaben



- ♦ Das Rechenwerk besteht aus
  - Schaltnetzen/Funktionseinheiten (FE) zur Realisierung arithmetischer und logischer Operationen
  - Registern (REG) zur Aufnahme von Operanden und Ergebnissen
  - Multiplexern (MUX) als steuerbare Verbindungen zwischen Registern und Schaltnetzen.



- ♦ Das Steuerwerk hat die Aufgabe,
  - Statussignale (z.B. Übertragsbit aus einer Addition) aus dem Rechenwerk auszuwerten, und anhand eines Steuerprogramms daraus Steuersignale für das Rechenwerk zu generieren:
    - Enable-Signale für Register
    - Select-Signale für Multiplexer
    - Function-Signale für universelle Funktionseinheiten

## ♦ Register-Transfer-Operationen

- Ein digitales System verarbeitet Informationen, die in Registern gespeichert sind.
- Diese Informationen werden oft durch arithmetisch-logische Operationen (Mikrooperationen) verändert und wieder in Registern zurückgeschrieben.
- Daher kann man ein digitales System beschreiben durch:
  - Menge der Register und ihre Funktionen (laden, löschen, schieben),
  - Menge der möglichen Mikrooperationen (addieren, subtrahieren),
  - Steuerung, die die Ausführung der Mikrooperationen veranlaßt.
- Der Datentransfer von einem Register zu einem anderen Register wird symbolisch mit einem Zuweisungsoperator  $:=$  oder  $\leftarrow$  notiert.
- Auf der linken Seite des Zuweisungsoperators steht normalerweise ein n-stelliges Zielregister, rechts davon ein boolescher Ausdruck oder eine arithmetische Funktion.  
z.B. ACC  $:=$  A + B oder R1  $\leftarrow$  R2

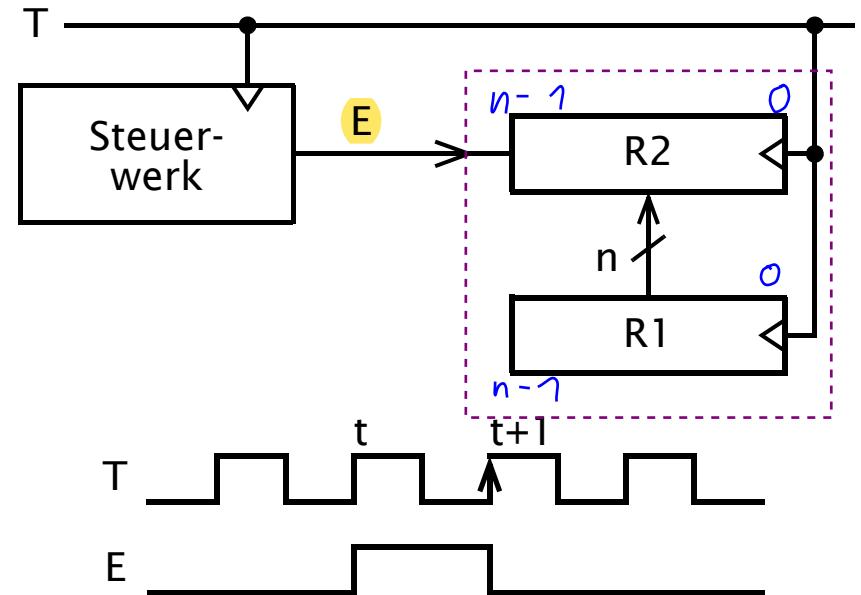
- ◆ Register-Transfer-Operationen

- Die Semantik der Notation  $R2 \leftarrow R1$ : der Inhalt von  $R1$  wird nach  $R2$  transportiert (kopiert), dabei wird  $R1$  nicht verändert.
- Dazu ist ein Datenpfad zwischen den Ausgängen von  $R1$  und den Eingängen von  $R2$  notwendig, und  $R2$  muß parallel geladen werden können.

- Der Transfer findet oft nicht mit jedem Takt statt, sondern unter gewissen Bedingungen:  
 $\text{if } (E=1) \text{ then } R2 \leftarrow R1 \text{ end}$   
 wo  $E$  ein Steuersignal

Register Transfer Language

- kompakte RTL-Notation:  
 $E: R2 \leftarrow R1$   
 $R2 := (E=1) ? R1 : R2$



- ◆ Register-Transfer-Operationen

- Alle Speicherelemente werden einheitlich mit dasselben Takt-signal gesteuert (=> Taktsignal erscheint nicht in der RTL-Notation),
- Register sind entweder aus flankengesteuerten Flipflops oder Master-Slave-Flipflops aufgebaut.  
z.B. n-Bit-Register  $IR[n-1:0]$  oder  $XYZ[1:n]$
- Mehrere Register-Transfer-Operationen können parallel ausgeführt werden, dann werden diese mit Komma voneinander getrennt.

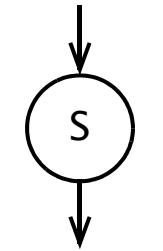
z.B. der Datentransfer zwischen zwei Registern unter der Kontrolle der Steuervariablen P lautet

P:  $R2 \leftarrow R5, PC \leftarrow PC+1, XYZ \leftarrow 0$

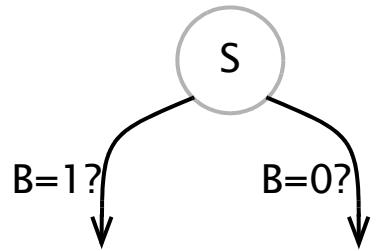
Interpretation: wenn die Bedingung erfüllt ist ( $P=1$ ), dann wird der Inhalt aus dem Register R5 ins Register R2 kopiert, und gleichzeitig (d.h. in demselben Takt), werden der Inhalt des Registers PC um 1 erhöht und der Inhalt des Registers XYZ auf Null gesetzt.

- ◆ Realisierung von Steuerwerken
  - festverdrahtete Steuerwerke
    - Schrittsteuerwerk (Einphasentakt)
    - Multiphasen-Generator
    - PLA und Zustandsregister
  - (mikro-)programmierbare Steuerwerke
- ◆ Schrittsteuerwerk mit One-Hot-Encoding der Zustände

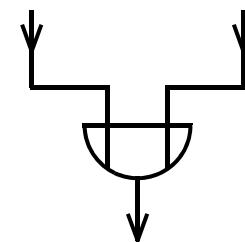
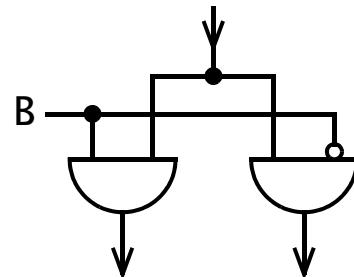
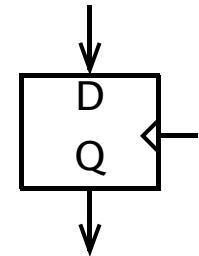
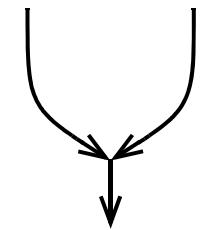
Zustand



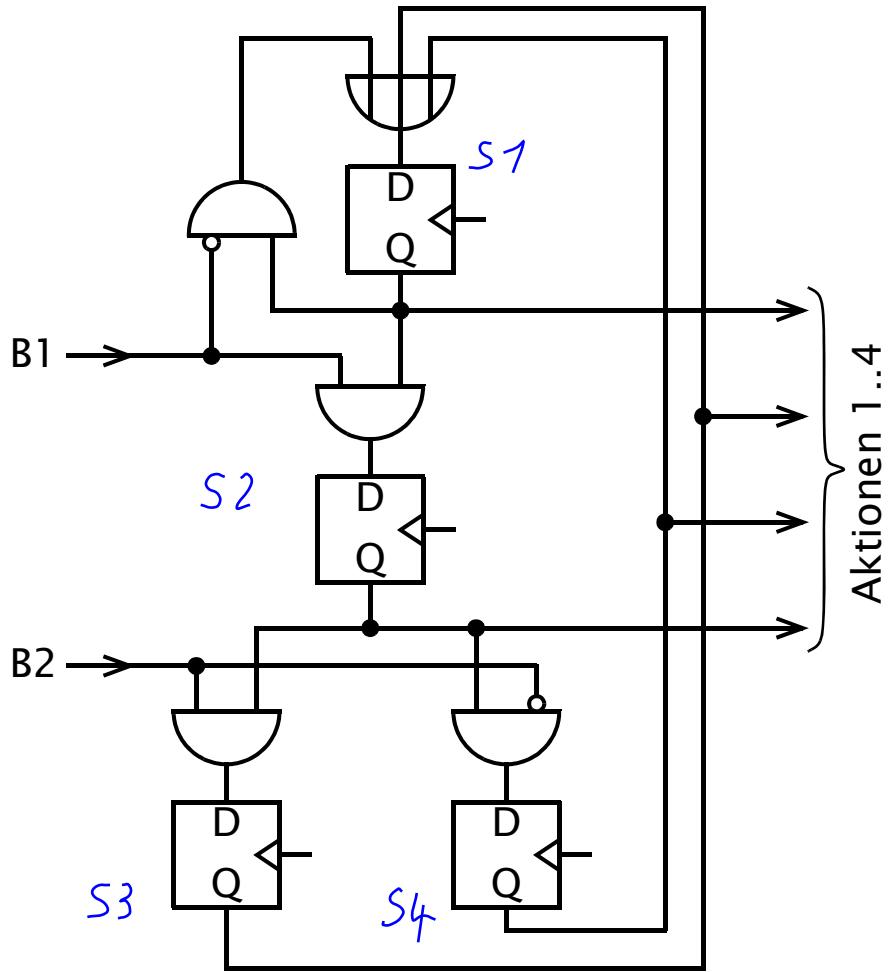
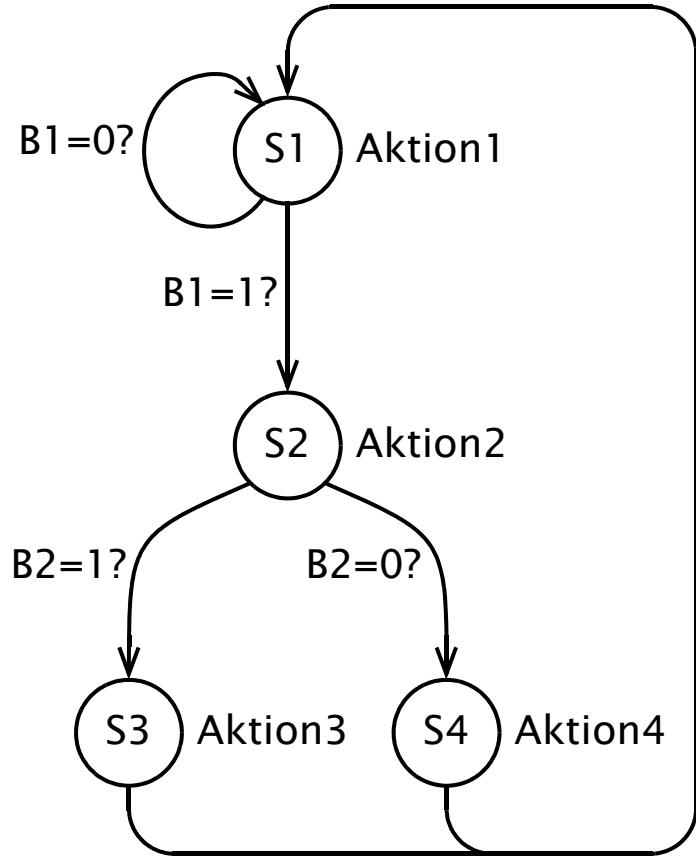
Verzweigung



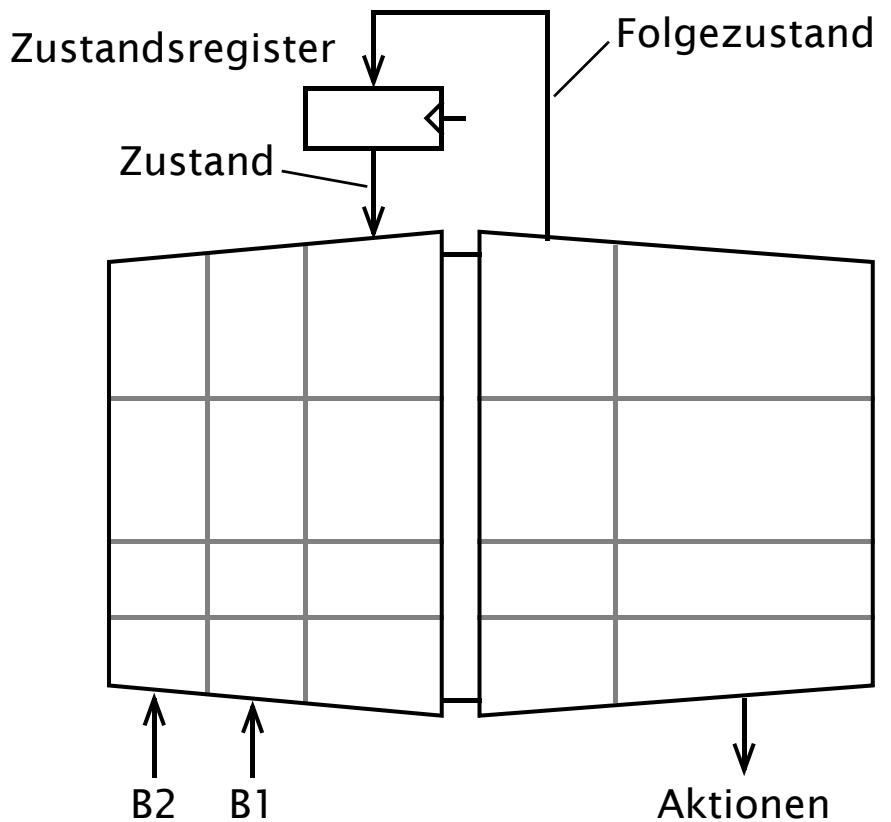
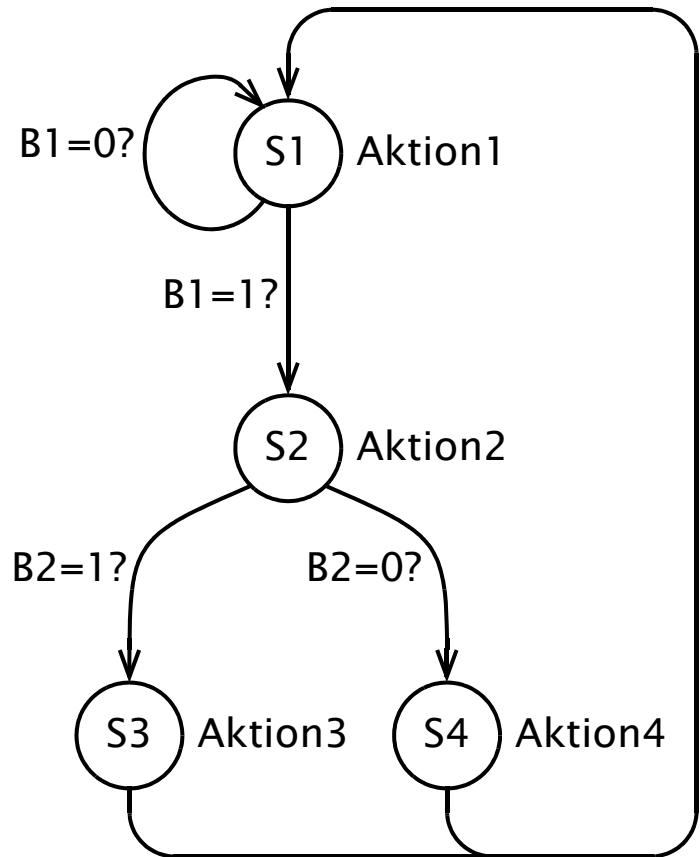
Zusammenführung



- ♦ Schrittsteuerwerk mit One-Hot-Encoding



- ◆ PLA-Steuerwerk mit binär codierten Zuständen
  - Codierung der Zustände:  $S1 := (00)_2$ ,  $S2 := (01)_2$ ,  $S3 := (10)_2$ ,  $S4 := (11)_2$



## ♦ Synthese synchroner Schaltwerke

- Das Ziel bei der Synthese von Schaltwerken ist wie bei Schaltnetzen eine Realisierung mit einer möglichst geringen Schaltungskomplexität und möglichst kürzeren Laufzeiten (=> größte Taktfrequenz)
- Zwei Größen haben erheblichen Einfluß auf die Komplexität von Übergangs- und Ausgangsschaltnetzen in Schaltwerken:
  - die Zahl der Zustandsvariablen
  - die Codierung der Zustände
- Diese Zusammenhänge sind so komplex, daß heute noch kein Verfahren zur Minimierung des Realisierungsaufwandes existiert, welches alle Entwurfsfreiheiten benutzt.

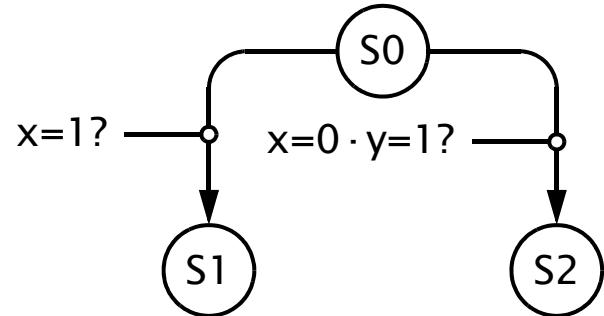
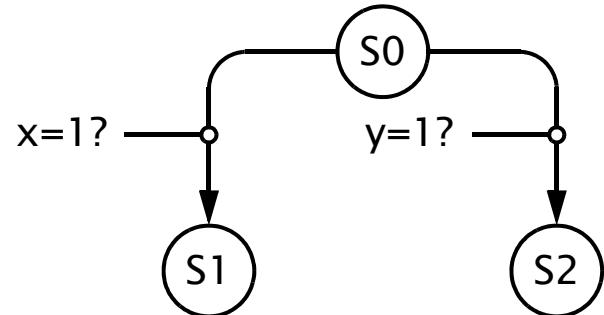
- ◆ Äquivalente Automaten

- Zwei Automaten  $EA_1$  und  $EA_2$  sind äquivalent, wenn sie aufgrund ihres äußereren Verhaltens nicht unterschieden werden können, d.h. wenn sie auf beliebige und beliebig lange Folgen von Eingangs-werten stets identische Folgen von Ausgangswerten erzeugen.

- ◆ Äquivalente Zustände

- Zwei Zustände  $Z_i$  und  $Z_j$  sind äquivalent, wenn der Automat auf eine beliebige Folge von Eingangswerten immer mit derselben Folge von Ausgangswerten reagiert, gleichgültig ob im Zustand  $Z_i$  oder im Zustand  $Z_j$  begonnen wird.
- Äquivalente Zustände können vereinigt werden, was im allgemeinen zur Vereinfachung des Automaten und dessen technischen Realisierung als Steuerwerk führt.
- Für eine Menge äquivalenter Zustände (sog. Äquivalenzklasse) wird dann stellvertretend nur ein Zustand (sog. Repräsentant) berücksichtigt.

- ♦ Deterministische Automaten
  - Zwei verschiedene Eingaben, die ausgehend von einem Zustand den Übergang zu zwei verschiedenen Zuständen auslösen, dürfen nicht gleichzeitig auftreten.
  - In diesem Fall wäre es sonst nicht entscheidbar, welcher Zustandsübergang erfolgen soll. Es entstünde ein nicht-deterministisches Verhalten, das mit klassischen Verfahren nicht mehr untersucht werden kann.
  - Die meisten Analyse-/Syntheseverfahren gehen deshalb davon aus, daß zwei Ereignisse/Eingaben nicht zeitgleich auftreten können.



- ♦ **Synthese synchroner Schaltwerke**

erfolgt meistens in sechs iterativ ablaufenden Phasen:

1. Umsetzung einer textuellen (verbalen, nicht formalen) Beschreibung in eine graphische (formale) Spezifikation (wie Zustandsgraph, Impulsplan) mit der Festlegung der Ein-/Ausgangsvariablen
  2. Aufstellung einer sog. primitiven Zustandstabelle
  3. Zustandsminimierung und Zustandsverschmelzung
  4. Wahl der Zustandscodierung
  5. Minimierung der Übergangs- und Ausgangsschaltnetze
  6. Überprüfung des realisierten Schaltwerks
- 
- Bei einer Unzufriedenheit mit dem Resultat der Synthese kann der Ablauf wiederholt werden, und zwar in einer inneren Schleife mit den Schritten 4, 5 und 6, oder in einer äußeren Schleife, welche die Schritte 3 bis 6 umfaßt.

- ◆ Umsetzung einer verbalen, nicht formalen Beschreibung in eine formale Spezifikation
  - verbale Beschreibungen (Pflichtenheft) sind nicht formale Spezifikationen, die oft unvollständig, mehrdeutig, manchmal widersprüchlich sind,
  - die Umsetzung einer verbalen Beschreibung in eine formale Spezifikation (UML-Diagramme, Zustandsgraph, Impulsplan, Hardwarebeschreibungssprache, Programmablaufplan, usw.) ist der schwierigste Schritt in der Schaltwerkssynthese:  
z.B. eine verbale Beschreibung kann durch viele unterschiedliche (semantisch äquivalente) Zustandsgraphen beschrieben werden,
  - leider keine universell anwendbare Methode verfügbar, die von einer verbalen Beschreibung zu einer formalen Spezifikation führt.

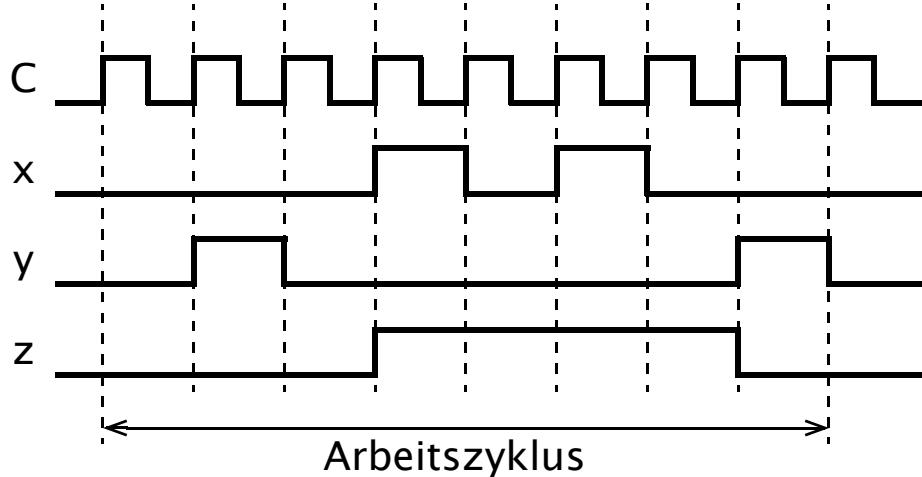
- ◆ Umsetzung einer verbalen, nicht formalen Beschreibung in eine formale Spezifikation
  - die Vorgehensweise und das Ergebnis der Umsetzung sind im hohen Maße von der Erfahrung und der Geschicklichkeit des Entwicklers abhängig,
  - Unübersichtlichkeit der Zustandsgraphen mit steigender Anzahl an Zuständen => Partitionierung in mehrere modulare, miteinander kooperierende Schaltwerke erforderlich => Handshaking- und Synchronisationssignale zwischen Schaltwerken notwendig
  - ist eine Aufgabe mit formalen Methoden spezifiziert, so sind restliche Schritte in der Schaltwerkssynthese mit systematischen Verfahren durchführbar.

- ◆ Zustandstabelle

- eine Tabelle mit Zeilen für sämtliche Zustände und mit Spalten für sämtliche Kombinationen der Eingangswerte und einer zusätzlichen Spalte für die Ausgangswerte.
- Zustände werden an den Zeilen notiert und Folgezustände in die Felder der Tabelle eingetragen.
- Sind für bestimmte Felder keine Folgezustände definiert, so bleiben diese leer.
- Die Werte der Ausgangsvariablen sind Zuständen zugeordneten und werden in der zusätzlichen Spalte zeilenweise eingetragen.

00	01	11	10	x	y	z

- ♦ Aufstellung einer primitiven Zustandstabelle aus einem Impulsplan
  - 1. Arbeitszyklus im Impulsplan identifizieren
  - 2. Arbeitszyklus in einzelne Taktphasen aufteilen, in den Taktphasen bleibt der Zustand des Schaltwerks stabil
  - 3. Taktphasen im Arbeitszyklus fortlaufend durchnumerieren
  - 4. Taktphasen in die primitive Zustandstabelle eintragen



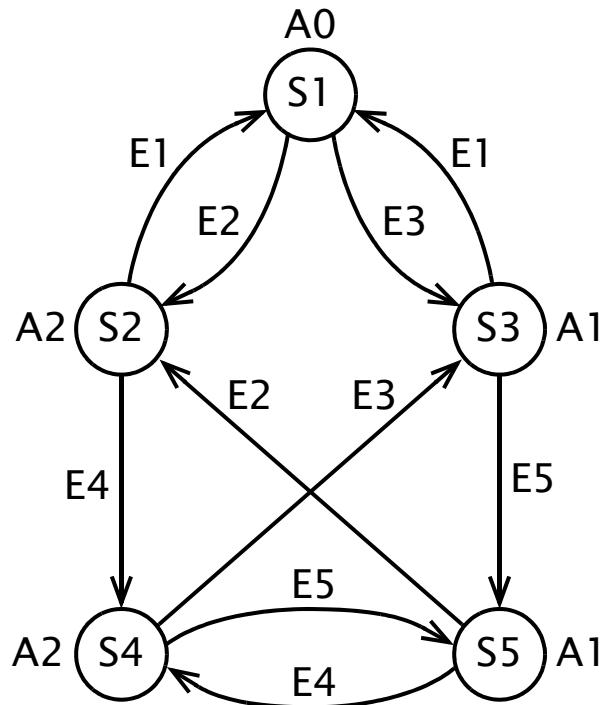
	xy	00	01	11	10	z
1						
2						
3						
4						
5						
6						
7						
8						

- ♦ Aufstellung einer primitiven Zustandstabelle aus einem Zustandsgraphen

Zustände: S1, S2, S3, S4, S5

Eingangskombinationen: E1, E2, E3, E4, E5

Ausgangskombinationen: A0, A1, A2



	E1	E2	E3	E4	E5
S1		S2	S3		
S2	S1			S4	
S3	S1				S5
S4			S3		S5
S5		S2		S4	

- ◆ Zustandsminimierung und Zustandsverschmelzung
  - Das Ziel der Zustandsminimierung und der anschließenden Zustandsverschmelzung ist es, die Anzahl der Zustände in einer primitiven Zustandstabelle zu reduzieren, ohne dadurch das Ein-/Ausgangsverhalten eines Systems zu ändern.
  - Die Reduktion der Anzahl der Zustände führt im allgemeinen zu Vereinfachung in Übergangs-/Ausgangsschaltnetzen.
  - Bei der Zustandsminimierung werden mit Hilfe einer Implikationsmatrix äquivalente und pseudo- äquivalente Zustände ermittelt und zur Reduzierung der Anzahl der Zustände benutzt.
  - Bei der Zustandsverschmelzung werden mit Hilfe eines Verschmelzungsgraphen sog. verschmelzbare Zustände mit dem gleichen Ausgangsverhalten ermittelt.

- ◆ Zustandsminimierung (1)

- zwei Zustände m und n sind äquivalent, und dürfen zu einem Zustand zusammengefaßt werden, wenn

	$E_1$	$E_2$	$\dots$	$E_{n-1}$	$E_n$	
m	p	m			s	0-01-
n		n		q	s	01-1-

=>

	$E_1$	$E_2$	$\dots$	$E_{n-1}$	$E_n$	
m	p	m		q	s	0101-

1. die Ausgangswerte der Zustände n und m gleich oder unbestimmt (don't care) sind, und
2. die den Zuständen n und m zugeordneten Folgezustände, also diejenigen, die durch dieselbe Eingangskombination  $E_i$  bestimmt sind (in derselben Spalte stehen), gleich oder unbestimmt sind.

- ♦ Zustandsminimierung (2)

- zwei Zustände m und n sind *pseudo*-äquivalent, und können *bedingt* zu einem Zustand zusammengefaßt werden, wenn

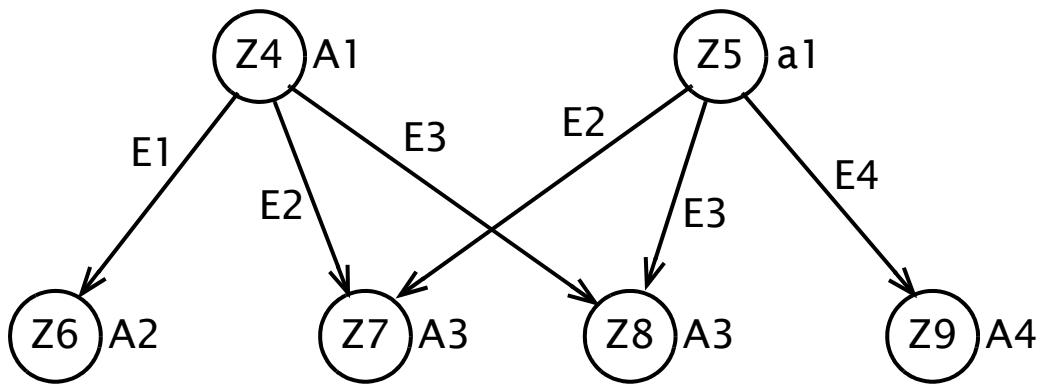
	$E_1$	$E_2$	$\dots$	$E_{n-1}$	$E_n$	
m	p	m			s	0-01-
n		n		q	r	01-1-

=>

	$E_1$	$E_2$	$\dots$	$E_{n-1}$	$E_n$	
m	p	m		q	s	0101-

1. die Ausgangswerte der Zustände n und m gleich oder unbestimmt sind, und
2. die den Zuständen n und m zugeordneten Folgezustände äquivalent oder *pseudo*-äquivalent sind.

- ◆ Zustandsminimierung



E1    E2    E3    E4

Z6	Z7	Z8	
----	----	----	--

=>

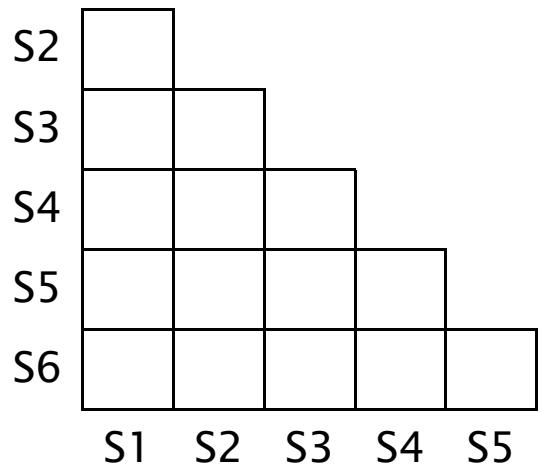
E1    E2    E3    E4

Z6	Z7	Z8	Z9
----	----	----	----

	Z7	Z8	Z9
--	----	----	----

## ♦ Implikationsmatrix

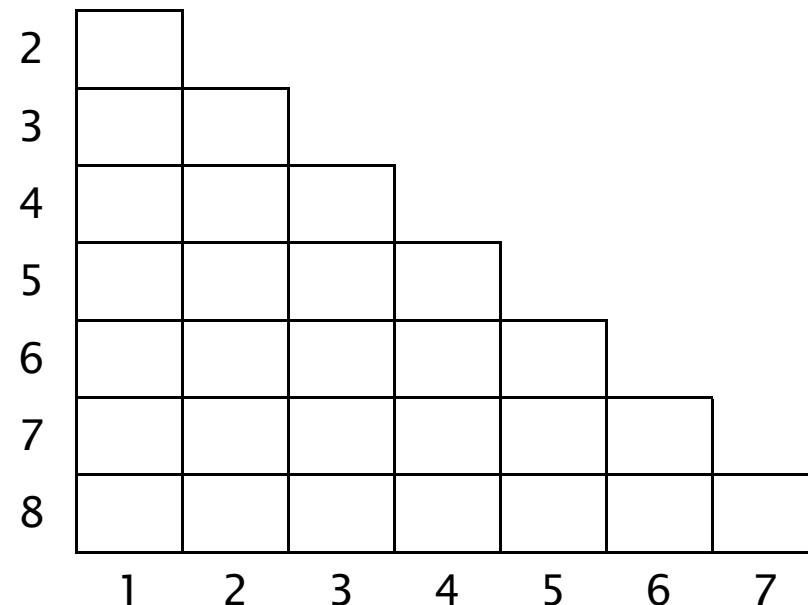
- dient zur Auffindung äquivalenter und pseudo-äquivalenter Zustände
- ist in Form einer Halbmatrix aufgebaut
- enthält alle Kombinationen der vorhandenen Zustände
- Beispiel: Automat mit 6 Zuständen:  
S1, S2, S3, S4, S5, S6
- in der Horizontalen (von links nach rechts) werden Zustände  $Z_1$  bis  $Z_{n-1}$  aufgetragen  
hier: S1, S2, S3, S4 und S5
- in der Vertikalen (von oben nach unten) werden Zustände  $Z_2$  bis  $Z_n$  notiert  
hier: S2, S3, S4, S5 und S6



- ◆ Einträge in der Implikationsmatrix:  
das (i, j)-Feld in der i-ten Zeile und j-ten Spalte kann einen der folgenden Einträge enthalten:
  - = Zustände i und j sind äquivalent und können unbedingt zusammengefaßt werden
  - ✗ Zustände i und j sind nicht äquivalent und können nicht zusammengefaßt werden
  - r,s Zustände i und j sind pseudo-äquivalent und können bedingt zusammengefaßt werden, und zwar in der Abhängigkeit davon, ob sich die Zustände r und s zusammenfassen lassen.
- ◆ Bei der Bestimmung der Einträge für die Implikationsmatrix werden sämtliche Zustände aus der primitiven Zustandstabelle paarweise miteinander verglichen.

- ♦ Aufstellung der Implikationsmatrix aus einer primitiven Zustandstabelle

	xy				z
	00	01	11	10	
1	1	2			0
2	3	2			0
3	3			4	0
4	5			4	1
5	5			6	1
6	7			6	1
7	7	8			1
8	1	8			0

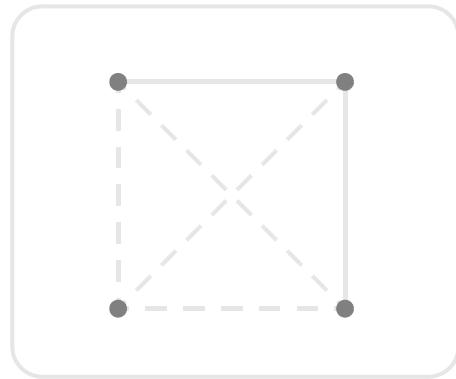
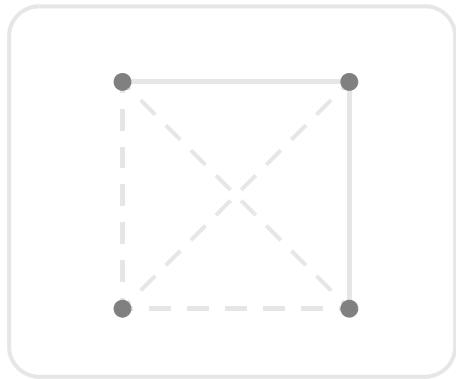


- ◆ Analyse der pseudo-äquivalenten Zustände aus der Implikationsmatrix  
für jedes Paar der pseudo-äquivalenten Zustände wird geprüft, ob es sich zusammenfassen lässt. Während der Analyse entsteht eine Implikationskette, die in einem Feld mit Zuständen terminiert, die:
  1. nicht äquivalent sind ( $\neq$ ): in diesem Fall werden sämtliche pseudo-äquivalenten Zustände in der entstandenen Implikationskette als nicht äquivalent markiert.
  2. äquivalent sind ( $=$ ): in diesem Fall werden sämtliche pseudo-äquivalenten Zustände in der entstandenen Implikationskette als äquivalent markiert.
  3. pseudo-äquivalent sind ( $(m,n)$ ), und bereits durch die Implikationskette analysiert worden sind. Somit entsteht eine endlose Rückkopplung in der Implikationskette: in diesem Fall können sämtliche pseudo-äquivalenten Zustände in der entstandenen Implikationskette als äquivalent markiert werden.

- ♦ Beispielsweise gilt für die Zusammenfassung der Zustände 1 und 8 folgende Implikationskette:
  - Die Zusammenfassung der Zustände 1 und 8 ist davon abhängig, ob die Zustände 2 und 8 zusammenfaßbar sind.
  - Die Zustände 2 und 8 können aber nur in der Abhängigkeit von der Zusammenfassung der Zustände 1 und 3 zusammengefaßt werden.
  - Die Zustände 1 und 3 sind äquivalent, somit können auch die Zustände 2 und 8 zusammengefaßt werden.
  - Und schließlich kann auch der Zustand 1 mit dem Zustand 8 zusammengefaßt werden.
  - Aufgrund der entstandenen Implikationskette bei der Analyse darf man auf keinen Fall nur die Zustände 1 und 8 zusammenfassen, ohne die Zusammenfassung der Zustände 2 und 8.

- ◆ Durch die Aufstellung eines Verschmelzungsgraphen wird die Analyse der pseudo-äquivalenten Zustände aus der Implikationsmatrix anschaulicher.
- ◆ Verschmelzungsgraph
  - ein ungerichteter Graph mit Knoten und Kanten
  - Knoten repräsentieren Zustände aus der Zustandstabelle
  - Kanten stellen die Möglichkeit der Zusammenfassung von äquivalenten und pseudo-äquivalenten Zuständen dar.
  - Eine Kante, die äquivalente Zustände miteinander verbindet, wird mit einer durchgezogenen Linie markiert.
  - Eine Kante, die pseudo-äquivalente Zustände verbindet, wird mit einer gestrichelten Linie markiert.

- ♦ Regelwerk für die Zustandsverschmelzung
  - zwei oder mehrere Zustände (Konten) dürfen zu einem Zustand verschmolzen werden, wenn sie alle unmittelbar mit Kanten untereinander verbunden sind (Gruppenbildung).
  - ein und derselbe Zustand darf nicht mehrfach an der Verschmelzung beteiligt sein.



- ♦ Aufstellung einer reduzierten Zustandstabelle
  - Einträge in der primitiven Zustandstabelle umbenennen:  
 $(1, 3, 2, 3) \rightarrow 1, (4, 5, 6, 7) \rightarrow 5$
  - mehrfach vorkommende Zeilen entfernen
  - Zeilen mit der gleichen Zustandsbezeichnung zusammenfassen

	xy				z
	00	01	11	10	
1	1	2			0
2	3	2			0
3	3			4	0
4	5			4	1
5	5			6	1
6	7			6	1
7	7	8			1
8	1	8			0

	xy				z
	00	01	11	10	

	xy				z
	00	01	11	10	

- ◆ Zustandscodierung

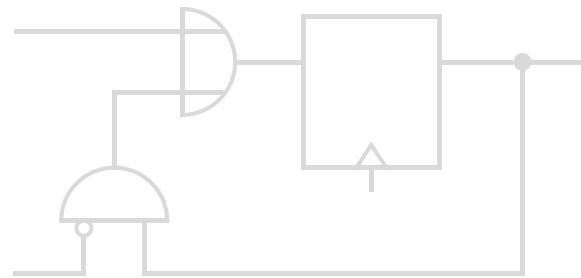
die Codierung der Zustände bei synchronen Schaltwerken kann theoretisch beliebig erfolgen, praktisch aber wird sie nach Gesichtspunkten der reinen Zweckmäßigkeit vorgenommen:

1. Zustandscodierung maximaler Länge („one hot encoding“):  
jede Zustandsvariable bekommt ein separates Flipflop zugeordnet,  
heute bevorzugt in schnellen Schaltwerken eingesetzt
2. Zustandscodierung mit minimalem Hardware-Aufwand:  
bspw. kann die Ausgangsfunktion (und somit auch das Ausgangsschaltnetz) bei Moore-Schaltwerken vollständig entfallen, wenn so eine Zustandscodierung gewählt wird, daß alle Werte der Ausgangsvariablen direkt mit der Zustandscodierung übereinstimmen.
3. Zustandscodierung mit Sicherung gegen externe Fehlerquellen:  
fehlerkorrigierende Codierung wie bei Zählern  
einschrittige Codierung „benachbarter“ Zustände

- ◆ Zustandscodierung und Realisierung als Schaltwerk mit festverdrahteter Logik
  - Zustandscodierung  $(1) \rightarrow (0)_2, (5) \rightarrow (1)_2$
  - Einträge in der reduzierten Zustandstabelle umbenennen
  - Minimierung der Übergangs- und Ausgangsfunktionen mit KV-Diagrammen

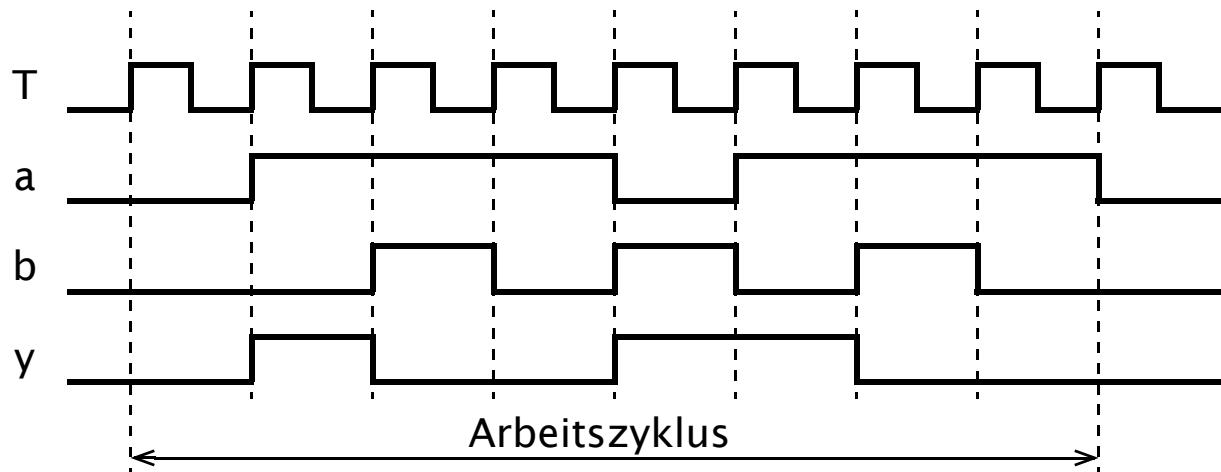
xy						z
		00	01	11	10	
1	1	1			5	0
	5	5	1		5	1

Q	xy						z
	00	01	11	10			



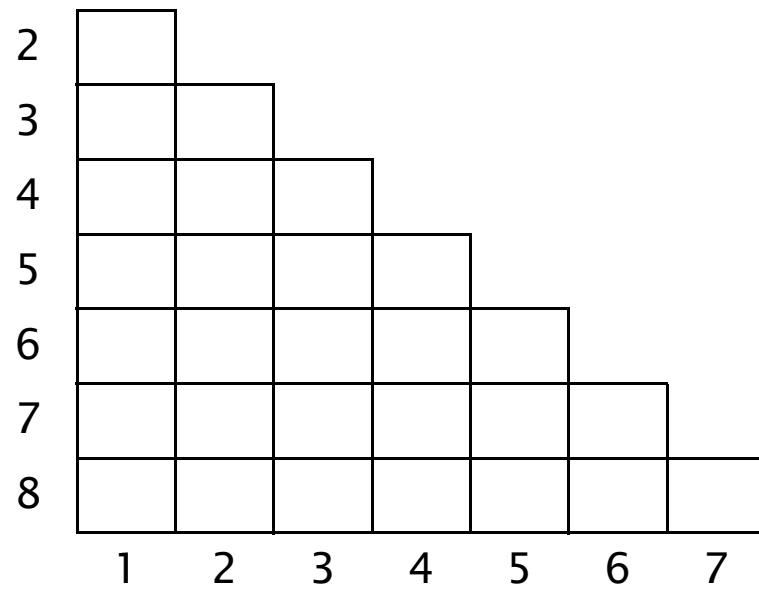
- ◆ Aufgabenstellung

- Es ist ein synchrones Schaltwerk mit zwei Eingängen  $a$  und  $b$  und einem Ausgang  $y$  mit möglichst minimaler Anzahl von Zuständen zu entwerfen. Das Schaltwerk arbeitet nach dem unten dargestellten Impulsplan.

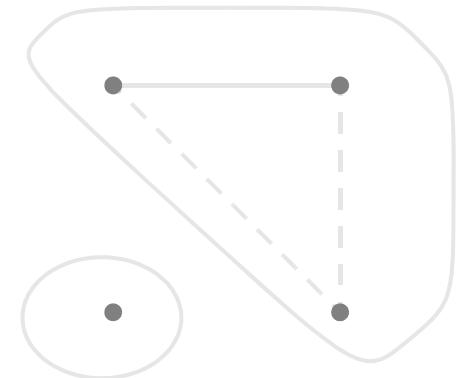
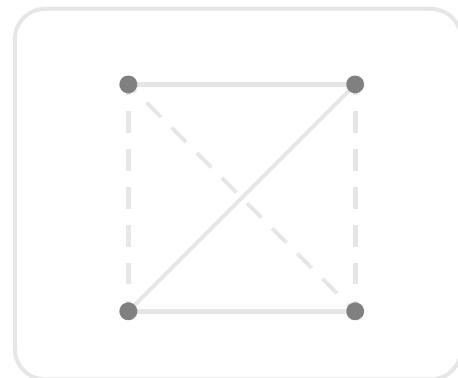


- ◆ primitive Zustandstabelle und Implikationsmatrix

ab	00	01	11	10	y



- ◆ Analyse der Einträge aus der Implikationsmatrix und Aufbau des Verschmelzungsgraphen
  - äquivalente Zustände:
  - pseudo-äquivalente Zustände:



# Schaltwerke

- reduzierte Zustandstabelle, Codierung der Zustände und Realisierung mit einem PLA-Steuerwerk

alter Zustand	neuer Zustand	bin. Code
1	A	00
2, 5, 6	B	01
3, 4, 7, 8	C	10

