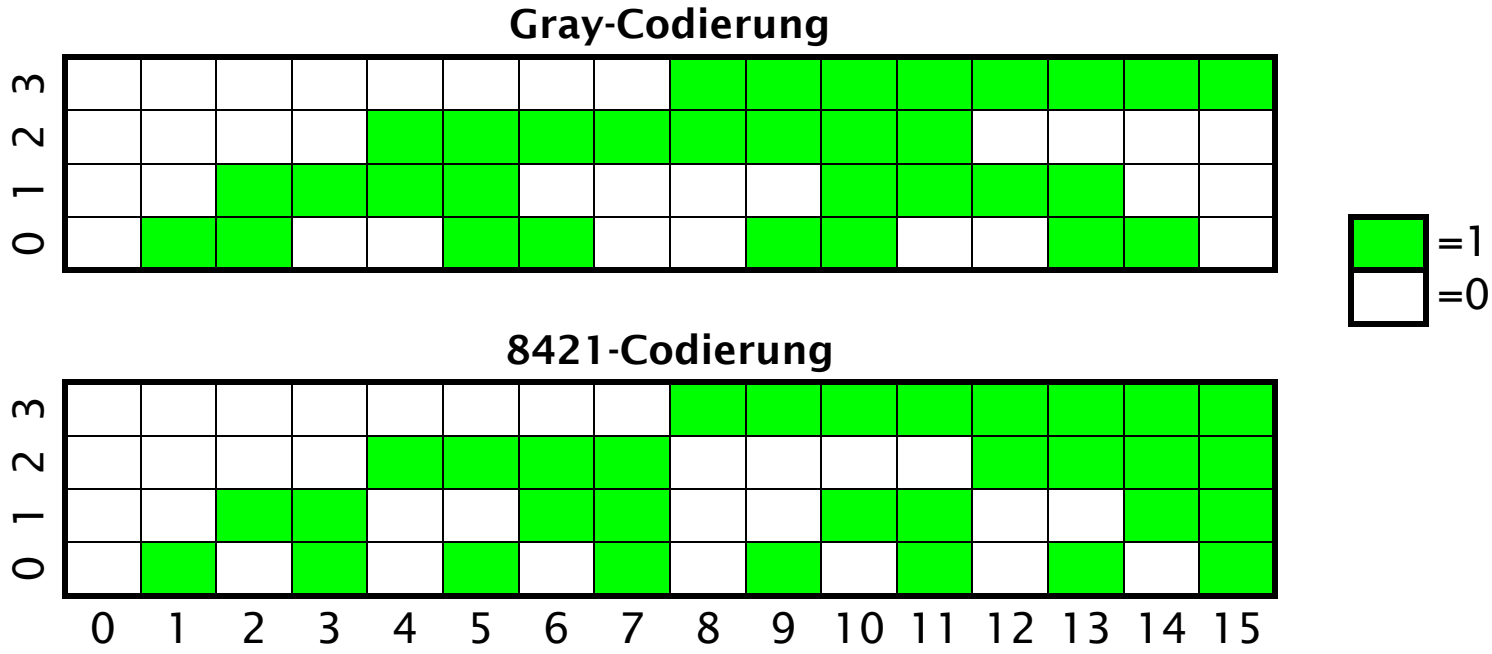
# KV-Methode

- 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.

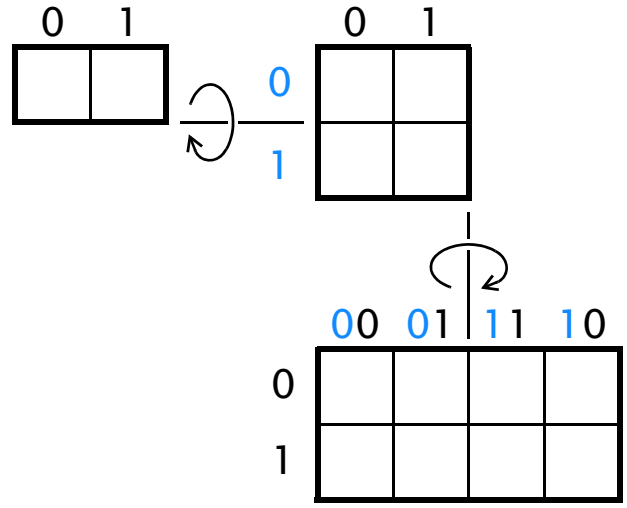
- Einschrittige vs. mehrschrittige Codierung
  - Benachbarte Elemente einer Menge werden durch Binärmuster repräsentiert, die sich in genau einer Stelle unterscheiden.
  - Die Eigenschaft „benachbart“ ist eine Relation, die auf den Elementen der Menge definiert ist.
    - räumliche Nachbarschaft in Abtastsystemen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



- Konstruktion eines KV-Diagramms

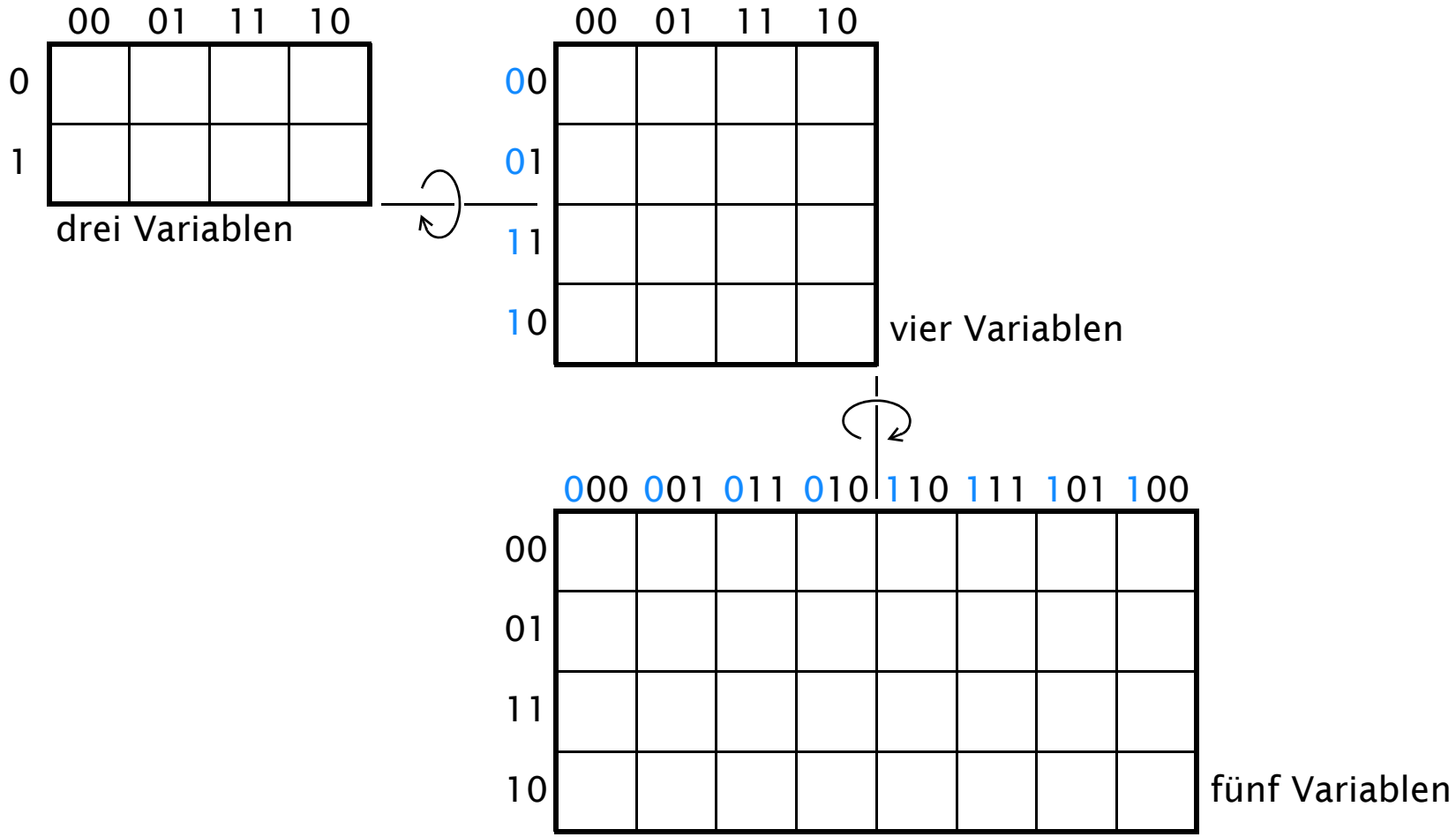
  Indizierung der Felder (Spalten und Zeilen) mittels einer einschrittigen, zyklischen Codierung (z.B. Gray-Code)

  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.

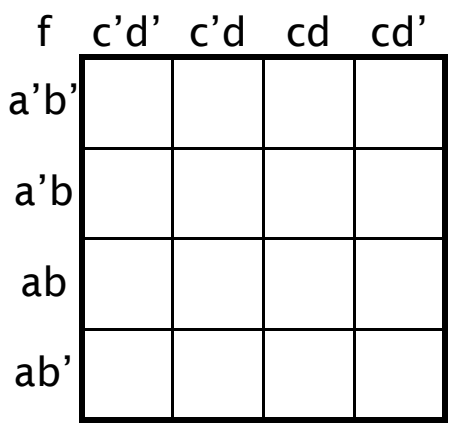


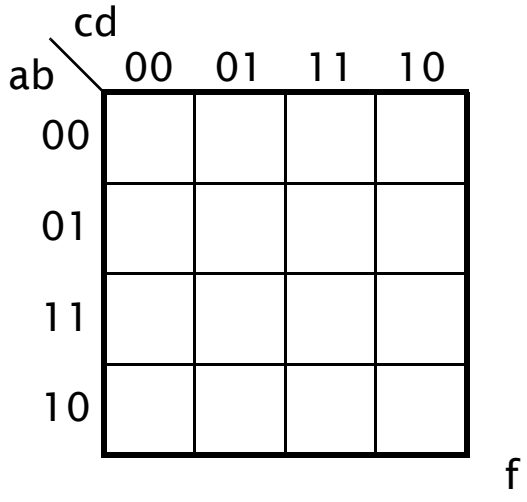

  3. Durch erneute Spiegelung des KV-Diagramms um die vertikale Randkante wird das ursprüngliche KV-Diagramm um eine dritte Eingangsvariable erweitert.
     Für jede zusätzliche Eingangsvariable verdoppelt sind die Anzahl der Felder im KV-Diagramm: $2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 32$ usw..

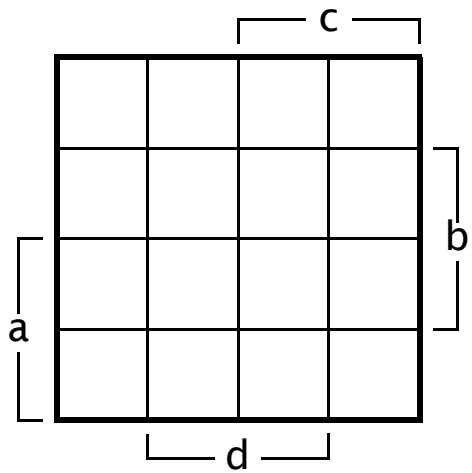
- Konstruktion eines KV-Diagramms für 3, 4 und 5 Eingangsvariablen



- äquivalente Darstellungen von KV-Diagrammen für 4 Variablen



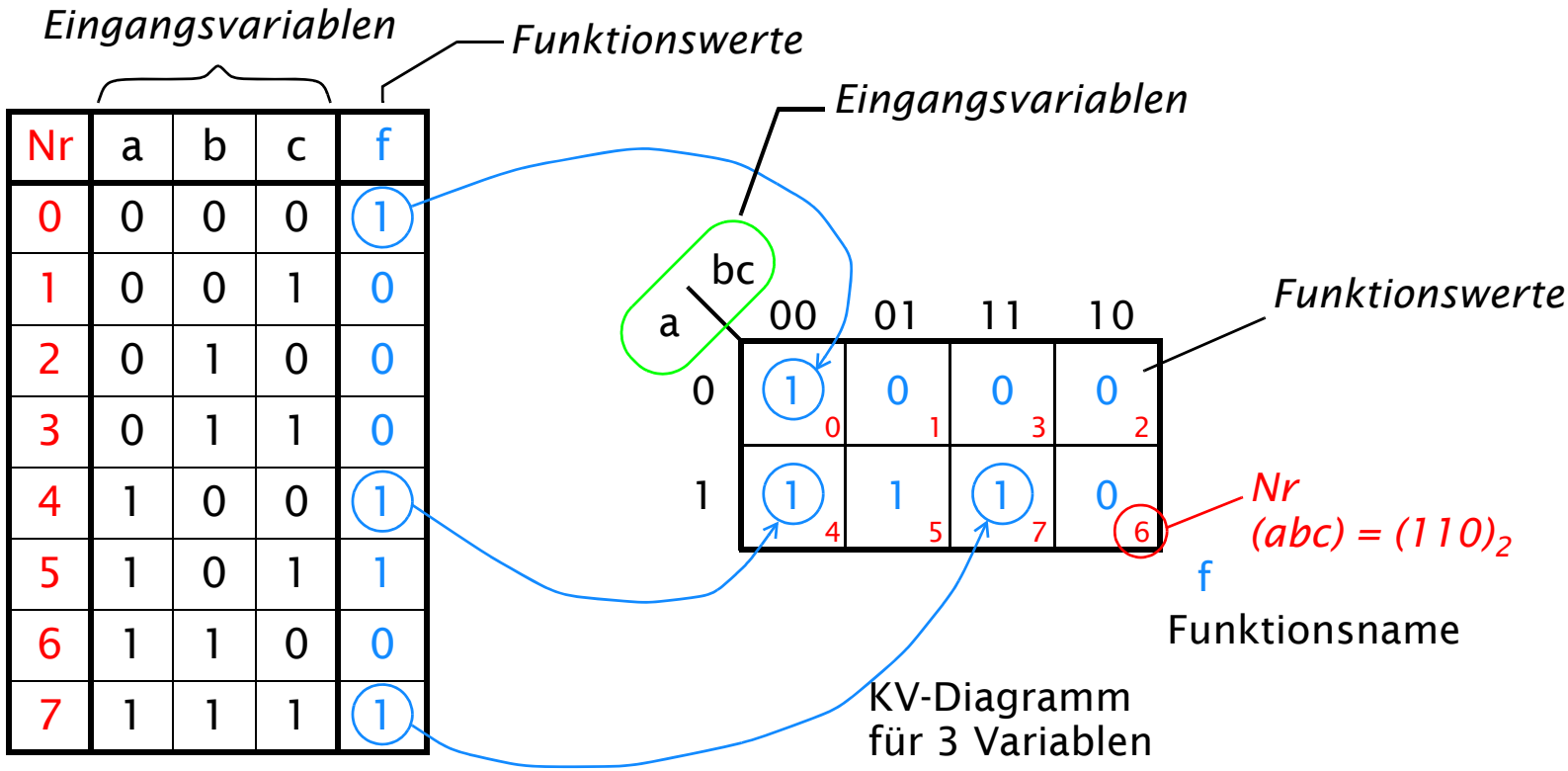



- Abbildung einer Funktionstabelle auf ein KV-Diagramm

  $f(a, b, c) = \Sigma(0, 4, 5, 7) = \Sigma(000_2, 100_2, 101_2, 111_2)$

  – 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 gebilet 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

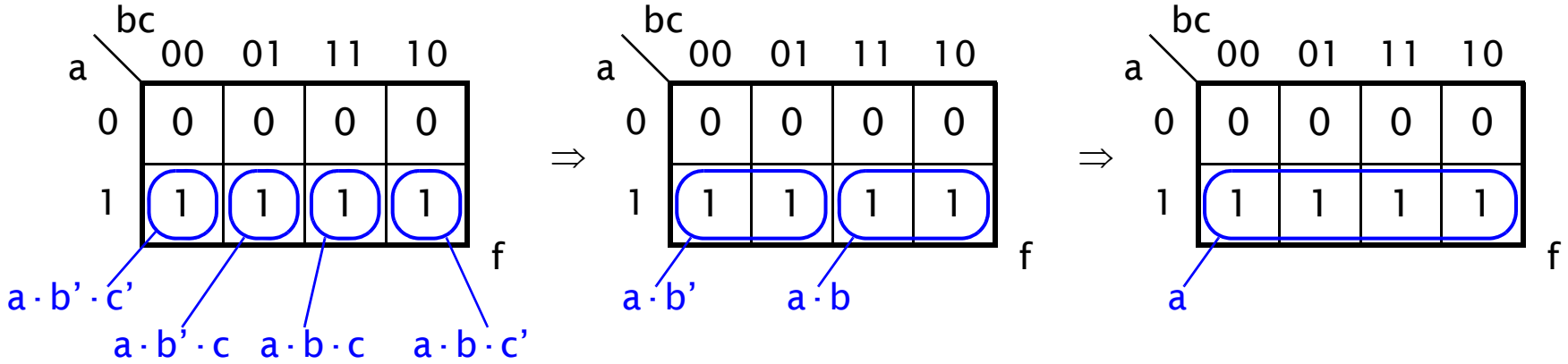
  $f(a, b, c) = a \cdot b' \cdot c' + a \cdot b' \cdot c + a \cdot b \cdot c + a \cdot b \cdot c'$

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

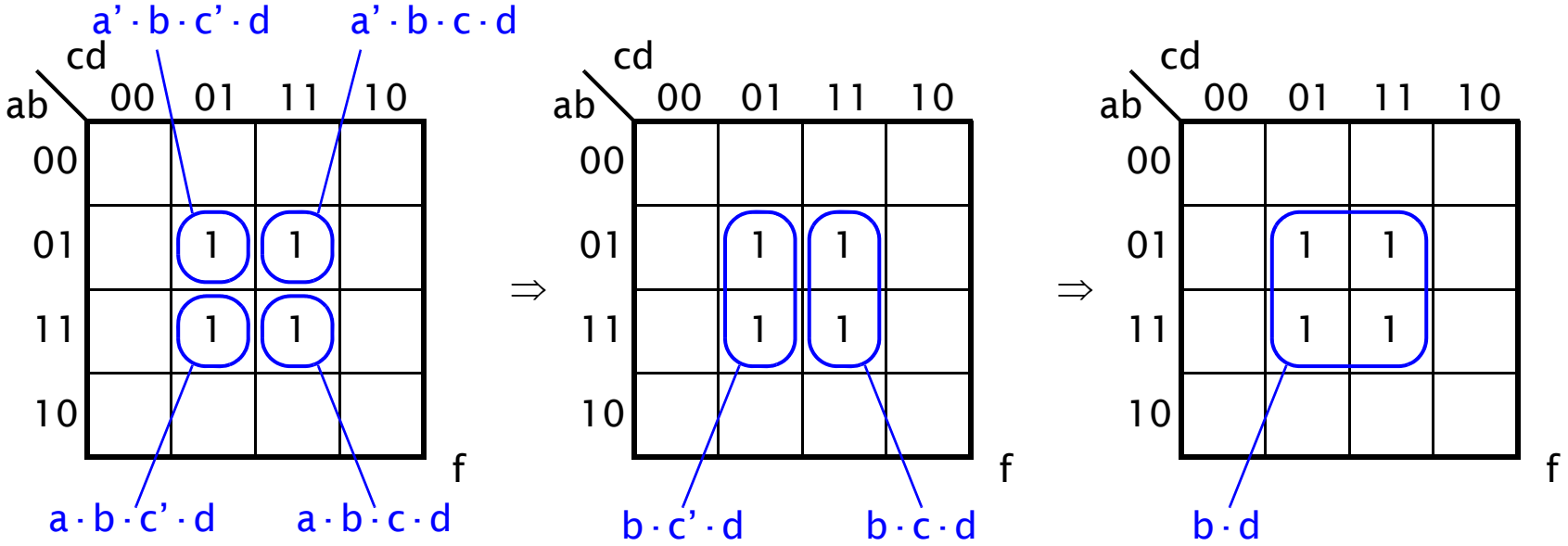
  $= ( a \cdot b' \cdot (c' + c) ) + ( a \cdot b \cdot (c + c') )$

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

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

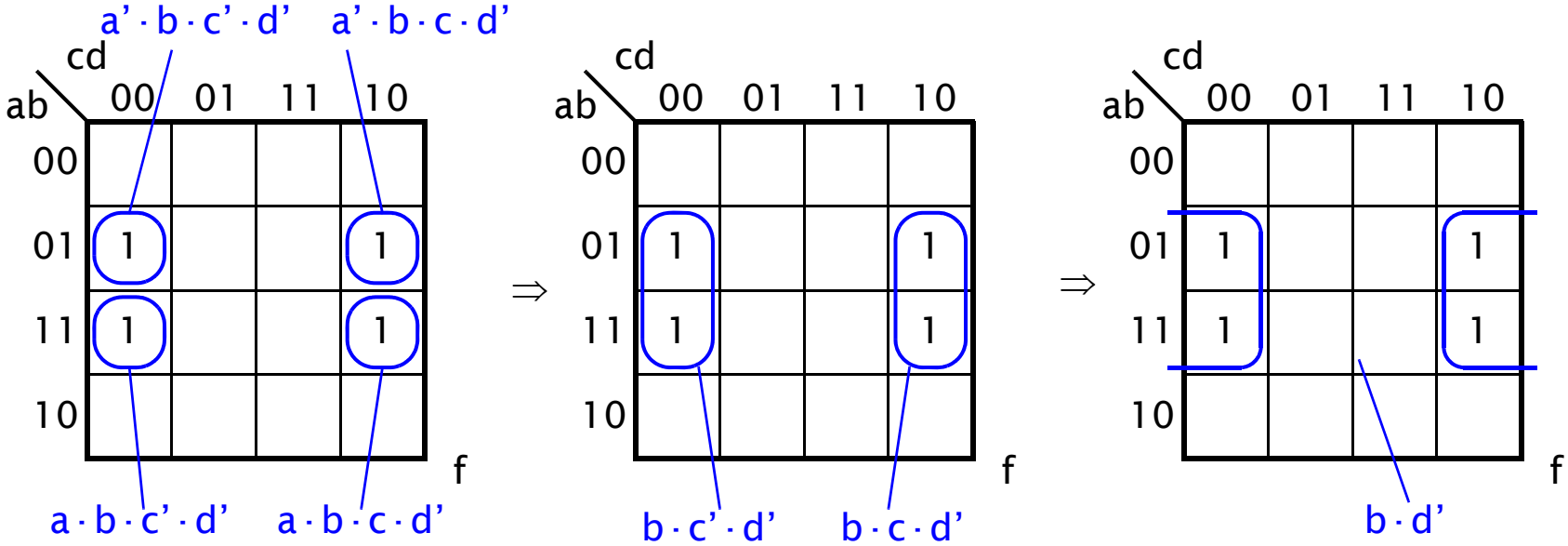


- Gruppieren von innenliegenden Feldern



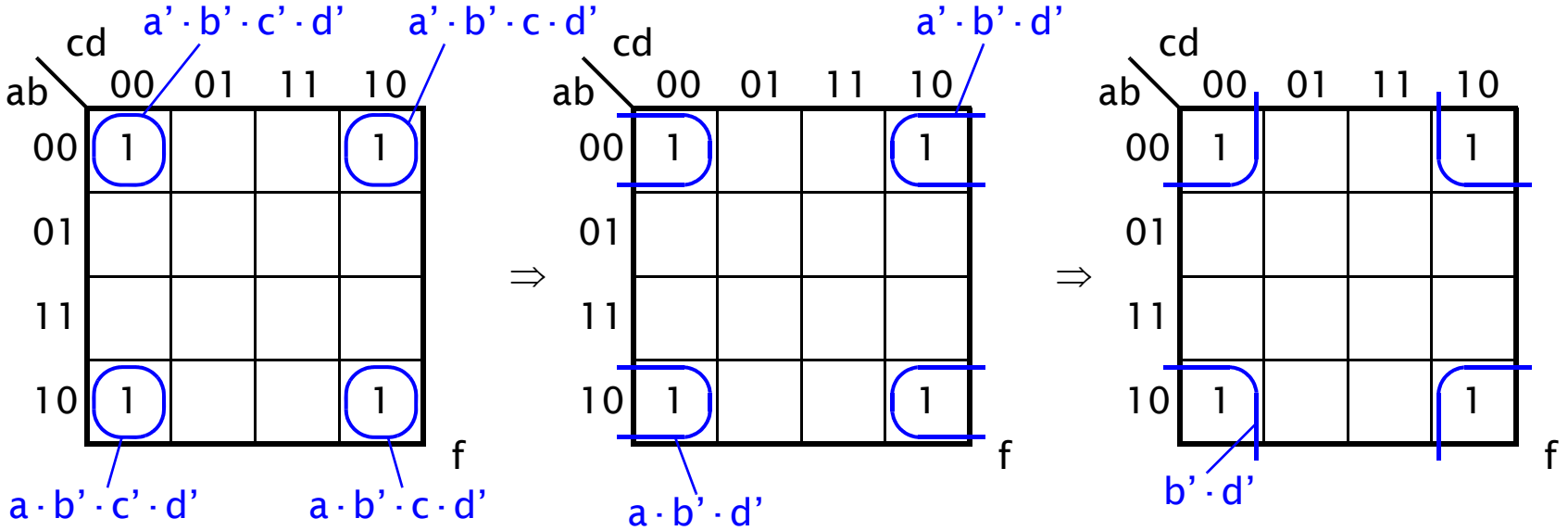

$$f(a, b, c, d) = a' \cdot b \cdot c' \cdot d + a \cdot b \cdot c' \cdot d + a' \cdot b \cdot c \cdot d + a \cdot b \cdot c \cdot d$$

- Gruppieren von Feldern mit Randlage



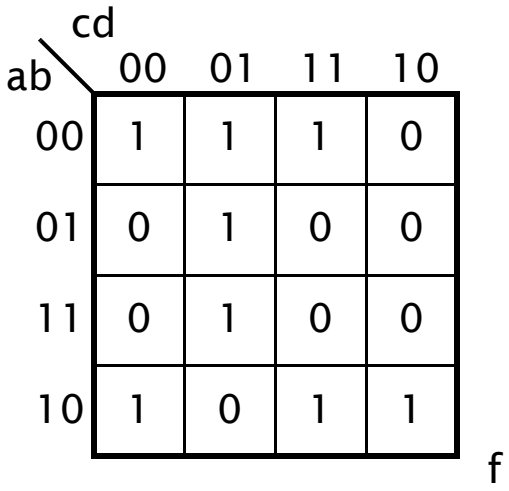

$$f(a, b, c, d) = a \cdot b \cdot c' \cdot d' + a' \cdot b \cdot c' \cdot d' + a \cdot b \cdot c \cdot d' + a' \cdot b \cdot c \cdot d'$$

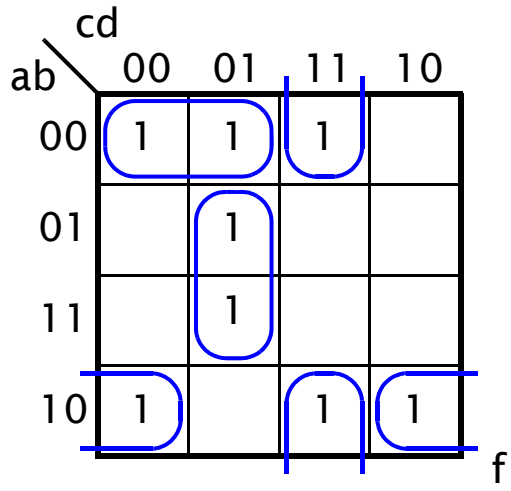
- Gruppieren von Feldern in Ecken

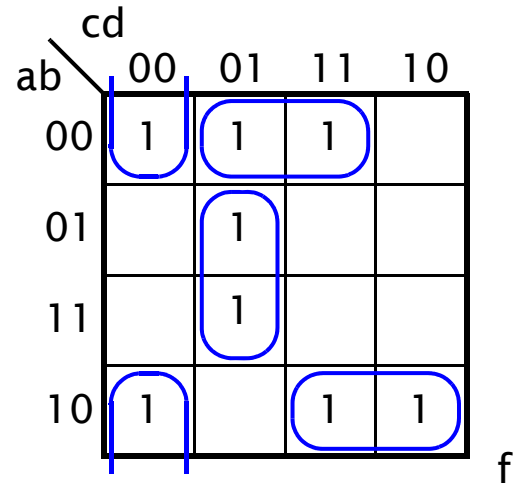



$$f(a, b, c, d) = a \cdot b' \cdot c' \cdot d' + a \cdot b' \cdot c \cdot d' + a' \cdot b' \cdot c' \cdot d' + a' \cdot b' \cdot c \cdot d'$$

- Gruppenbildung mit alternativen Lösungen



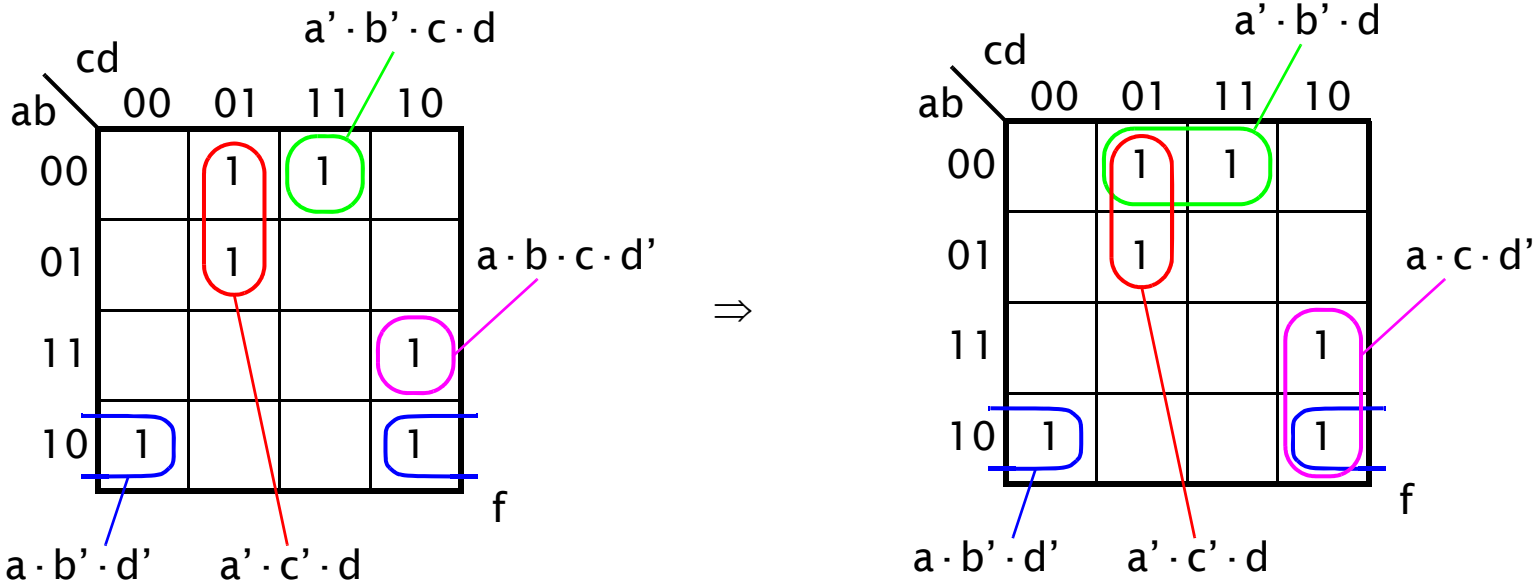




$f(a, b, c, d) = b \cdot c' \cdot d + a' \cdot b' \cdot c' + b' \cdot c \cdot d + a \cdot b' \cdot d'$

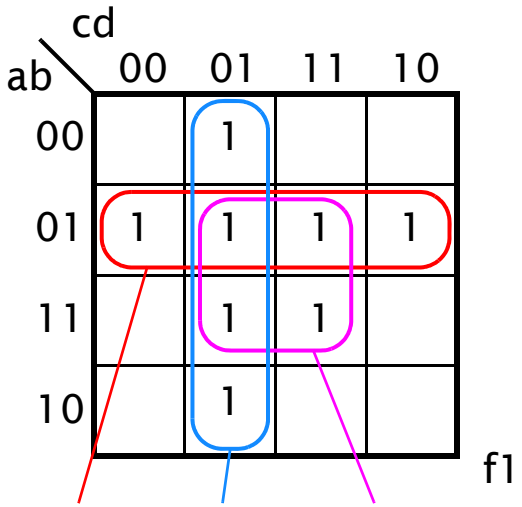
$f(a, b, c, d) = b \cdot c' \cdot d + a' \cdot b' \cdot d + b' \cdot c' \cdot d' + a \cdot b' \cdot c$

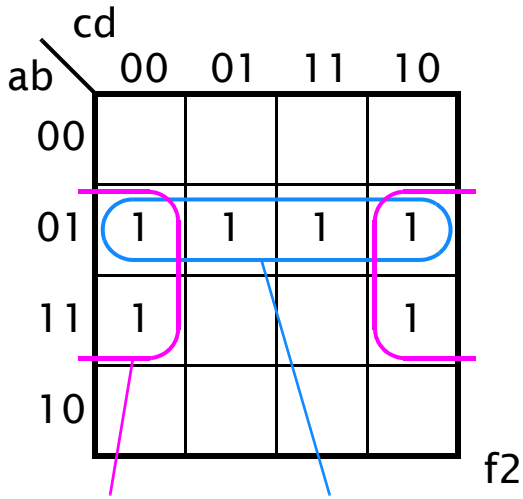
- Verschmelzung einzelner Felder mit größeren Gruppen




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

- Bildung überlappender Gruppen

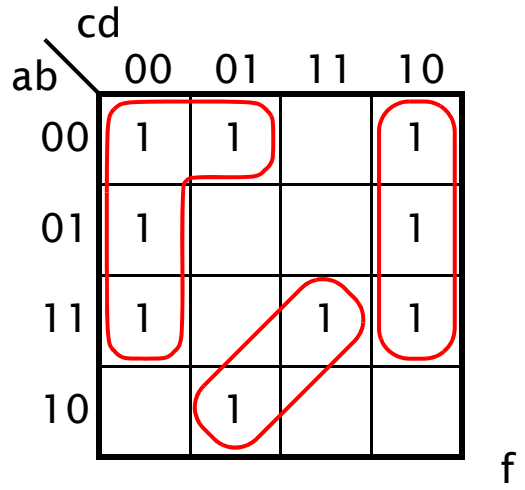


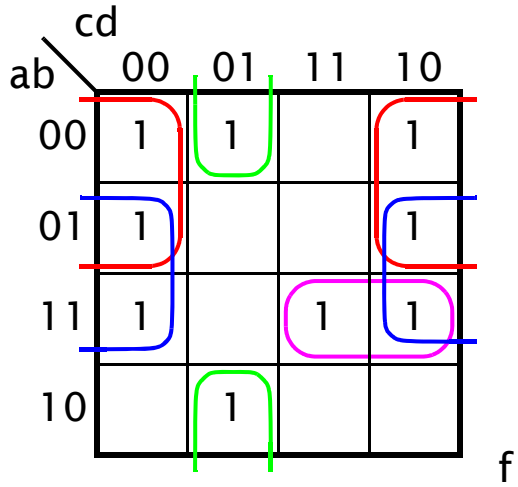

f1(a, b, c, d) =

f2(a, b, c, d) =

- 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 $a' \cdot d'$, $b \cdot d'$



- Beispiel

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

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

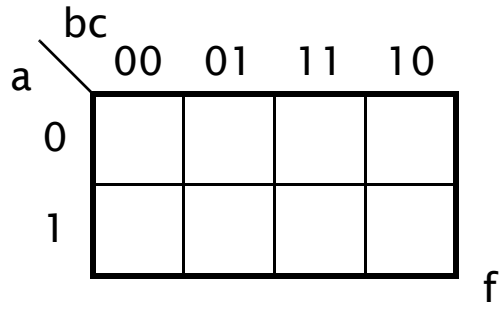
     f(a, b, c) = a + b' · c + a' · b · c

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

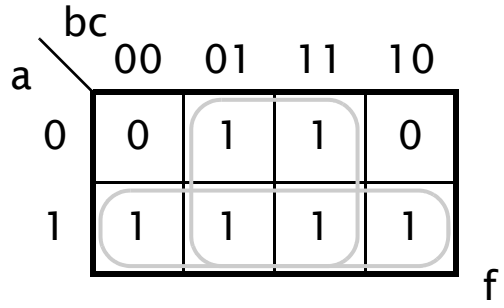
     KDNF(f) =

     KDNF(f) =

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




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




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 Mintereme 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

| | 8421-Code | | | | Gray-Code | | | |
|---|---|---|---|---|---|---|---|---|
| Nr | 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 |

| | 8421-Code | | | | Gray-Code | | | |
|---|---|---|---|---|---|---|---|---|
| Nr | 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 | – | – | – | – |

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

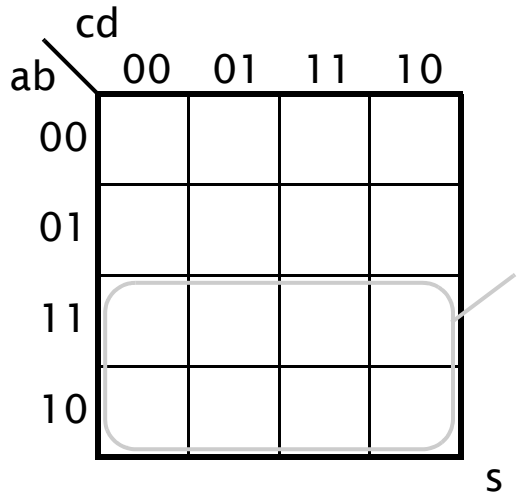
$t(a, b, c, d) = \Sigma(4, 5, 6, 7, 8, 9, (10, 11, 12, 13, 14, 15))$

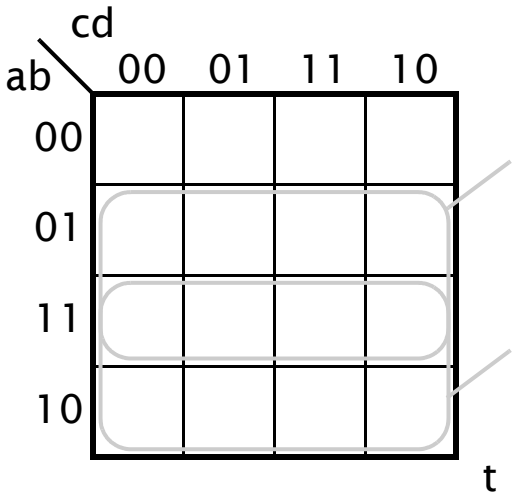
$u(a, b, c, d) = \Sigma(2, 3, 4, 5, (10, 11, 12, 13, 14, 15))$

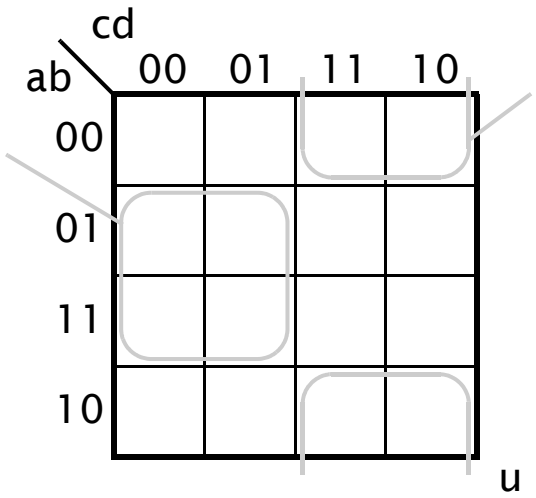
$v(a, b, c, d) = \Sigma(1, 2, 5, 6, 9, (10, 11, 12, 13, 14, 15))$

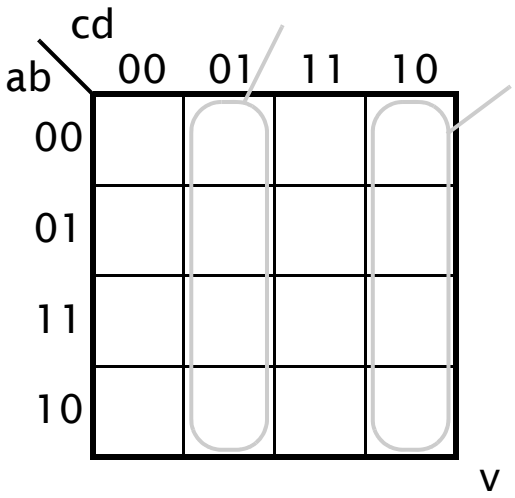
>> In der ersten Liste in Klammern stehen Minterme, in der zweiten „don‘t-care“-Terme

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



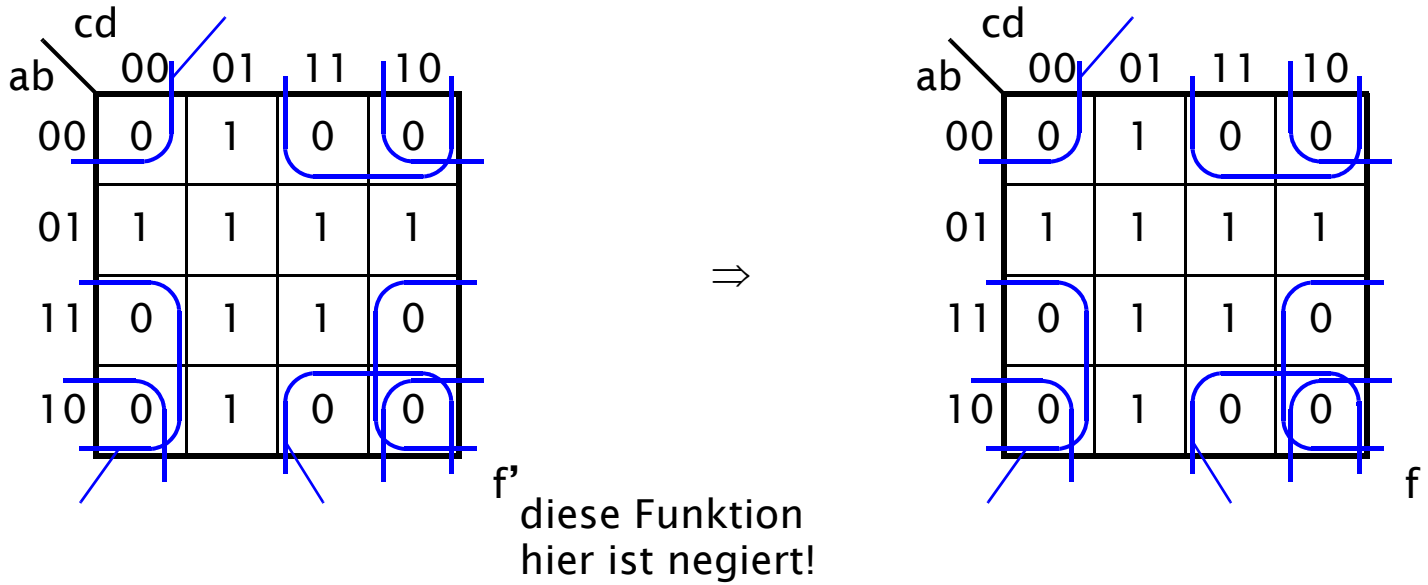
- minimierte Gleichungen für den 8421-Gray-Code-Wandler

  s(a, b, c, d) =
  t(a, b, c, d) =
  u(a, b, c, d) =
  v(a, b, c, 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 gebilet 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.

- 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

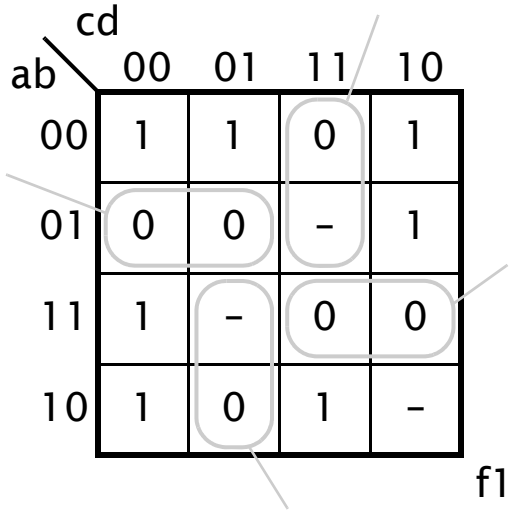
$f(a, b, c, d)'' = (b' \cdot d' + a \cdot d' + b' \cdot c)' = (b' \cdot d')' \cdot (a \cdot d')' \cdot (b' \cdot c)'$

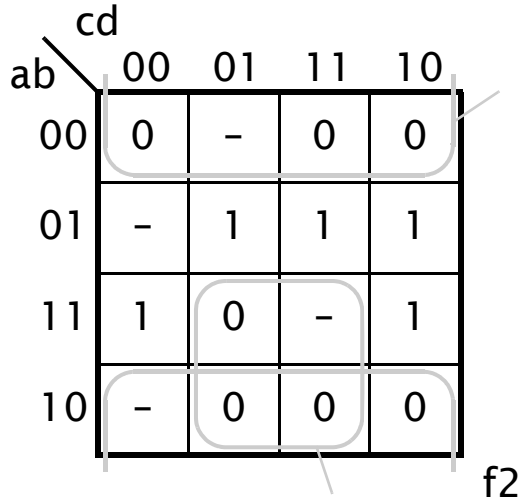
$f(a, b, c, d) = (b + d) \cdot (a' + d) \cdot (b + c')$

- Findung einer minimalen KNF mit KV-Diagrammen

  f1(a, b, c, d) = Π(3, 4, 5, 9, 14, 15, (7, 10, 13))

  f2(a, b, c, d) = Π(0, 2, 3, 9, 10, 11, 13, (1, 4, 8, 15))



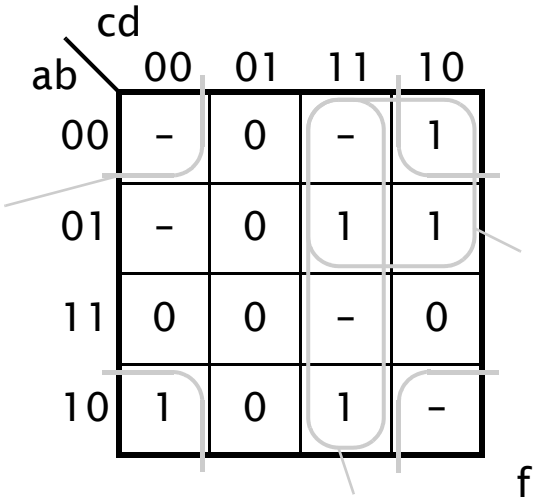



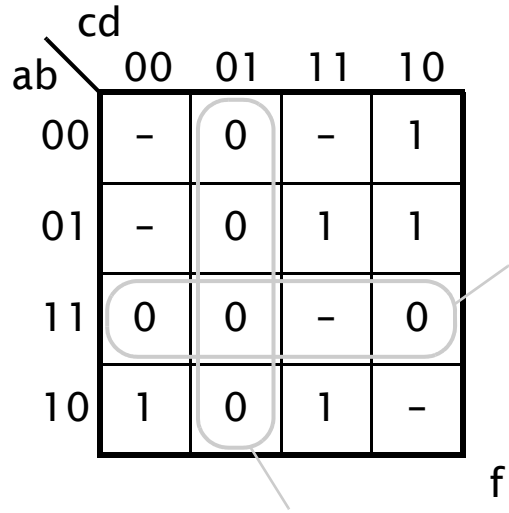

f1(a, b, c, d) =

f2(a, b, c, d) =

- Findung einer minimalen DNF/KNF mit KV-Diagrammen

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

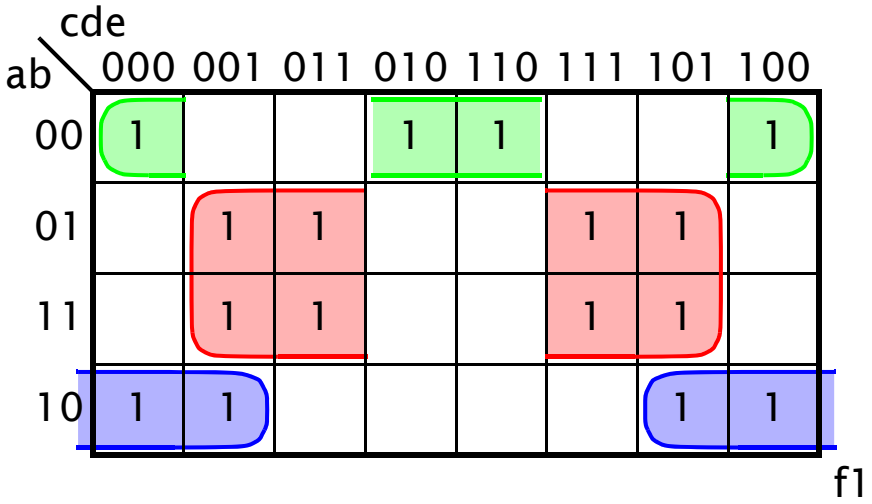


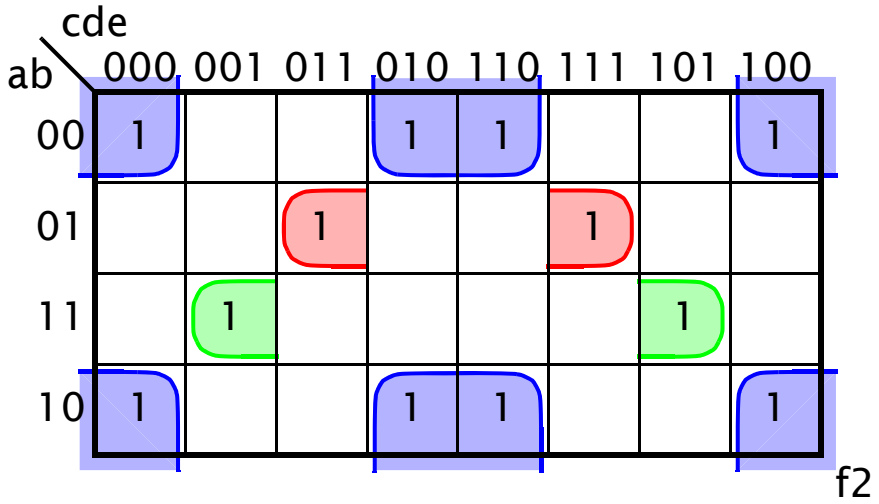




als KNF: f(a, b, c, d) =

als DNF: f(a, b, c, d) =

- Beispiele von KV-Diagrammen mit 5 Variablen




f1(a, b, c, d, e) = a' · b' · e' + b · e + a · b' · d'

f2(a, b, c, d, e) = a' · b · d · e + a · b · d' · e + b' · e'

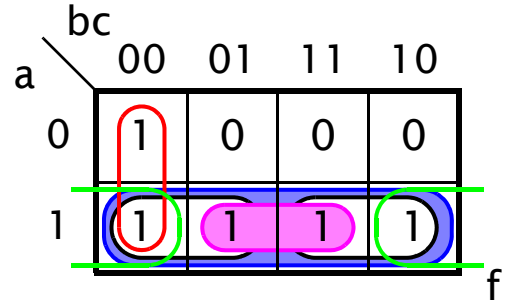
- Die QM-Methode ist
  - eine nach Quine und McCluskey benannte Methode zur Minimierung boolescher Funktionen
  - systematisch, algorithmisch-orientiert, und relativ einfach in der Anwendung
  - theoretisch für beliebig viele Variablen anwendbar, weil sie keinen Einschränkungen bezüglich einer geometrischen 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) = Σ(0, 4, 5, 6, 7)

alle Minterme von f sind automatisch Implikanten von f:
a' · b' · c', a · b' · c', a · b' · c,
a · b · c', a · b · c




aber auch folgende (Produkt-)Terme sind Implikanten von f:
b' · c', a · b', a · b, a · c', a · 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'·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'·c' und a

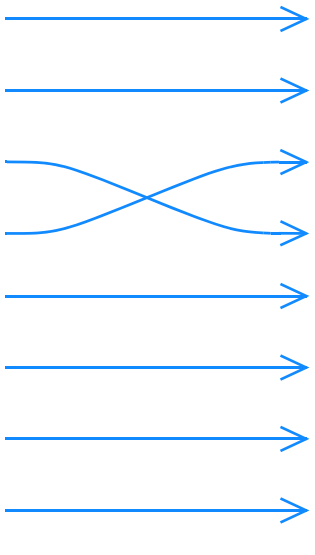
- Ein *redundanter Primimplikant* ist ein Primimplikant, der bereits durch einen Kernimplikant abgedeckt ist.

  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) = Σ(0, 2, 5, 8, 10, 13, 14, 15)
- Bildung von Minterm-Gruppen aus Mintermen mit gleicher Anzahl nicht negierter Variablen

| Nr. | a | b | c | d | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | *0* |
| 2 | 0 | 0 | 1 | 0 | *1* |
| 5 | 0 | 1 | 0 | 1 | *2* |
| 8 | 1 | 0 | 0 | 0 | *1* |
| 10 | 1 | 0 | 1 | 0 | *2* |
| 13 | 1 | 1 | 0 | 1 | *3* |
| 14 | 1 | 1 | 1 | 0 | *3* |
| 15 | 1 | 1 | 1 | 1 | *4* |



| a | b | c | d | |
|---|---|---|---|---|
| | | | | *0* |
| | | | | *1* |
| | | | | *1* |
| | | | | *2* |
| | | | | *2* |
| | | | | *3* |
| | | | | *3* |
| | | | | *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 | b | c | d |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |

⇒

| 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 | b | c | d |
|---|---|---|---|
| 1 | - | 0 | 0 |
| 1 | - | 0 | 1 |

⇒

| 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 | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | |
| 2 | 0 | 0 | 1 | 0 | |
| 8 | 1 | 0 | 0 | 0 | |
| 5 | 0 | 1 | 0 | 1 | |
| 10 | 1 | 0 | 1 | 0 | |
| 13 | 1 | 1 | 0 | 1 | |
| 14 | 1 | 1 | 1 | 0 | |
| 15 | 1 | 1 | 1 | 1 | |

| a | b | c | d | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

  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 |

| a | b | c | d | $G_j$ |
|---|---|---|---|---|
| | | | | |
| | | | | |

$P1 = (-101)_2$, $P2 = (1-10)_2$, $P3 = (11-1)_2$, $P4 = (111-)_2$, $P5 = (-0-0)_2$

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. $W_i = \Sigma(S_j)$ - 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

| Primimplikanten | abcd | 0000 | 0010 | 0101 | 1000 | 1010 | 1101 | 1110 | 1111 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Nr | 0 | 2 | 5 | 8 | 10 | 13 | 14 | 15 | $W_i=\Sigma(S_j)$ |
| -101 | 5,13 | | | x | | | x | | | 1,50 |
| 1-10 | 10,14 | | | | | x | | x | | 1,00 |
| 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 | | | | 3,50 |
| | $S_j=1/R_j$ | 1 | 1 | 1 | 1 | 1/2 | 1/2 | 1/2 | 1/2 | |

$W_{max}$ = 3,50 =>
Primimplikant P5 = $(-0-0)_2$ = b' · d' ist Kernimplikant K1
(Minterme 0, 2, 8 und 10)

- Ermittlung von Kernimplikanten N=5, K=8

Minterme aus der Funktionstabelle

| Primimplikanten | abcd | 0000 | 0010 | 0101 | 1000 | 1010 | 1101 | 1110 | 1111 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Nr | 0 | 2 | 5 | 8 | 10 | 13 | 14 | 15 | $W_i=\Sigma(S_j)$ |
| -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 P1 = $(-101)_2 = b \cdot c' \cdot d$ ist Kernimplikant K2 (Minterme 5 und 13).

- Ermittlung von Kernimplikanten N=5, K=8

Minterme aus der Funktionstabelle

| Primimplikanten | abcd | 0000 | 0010 | 0101 | 1000 | 1010 | 1101 | 1110 | 1111 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Nr | 0 | 2 | 5 | 8 | 10 | 13 | 14 | 15 | $W_i=\Sigma(S_j)$ |
| -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 P4 = $(111\text{-})_2 = a \cdot b \cdot c$ 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) = Σ(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 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

- 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 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

- 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 = $(0\text{–}0\text{–})_2$ = a' · c'

P2 = $(\text{–}0\text{–}0)_2$ = b' · d'

P3 = $(\text{––}00)_2$ = c' · d'

P4 = $(1\text{––}0)_2$ = a · d'

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

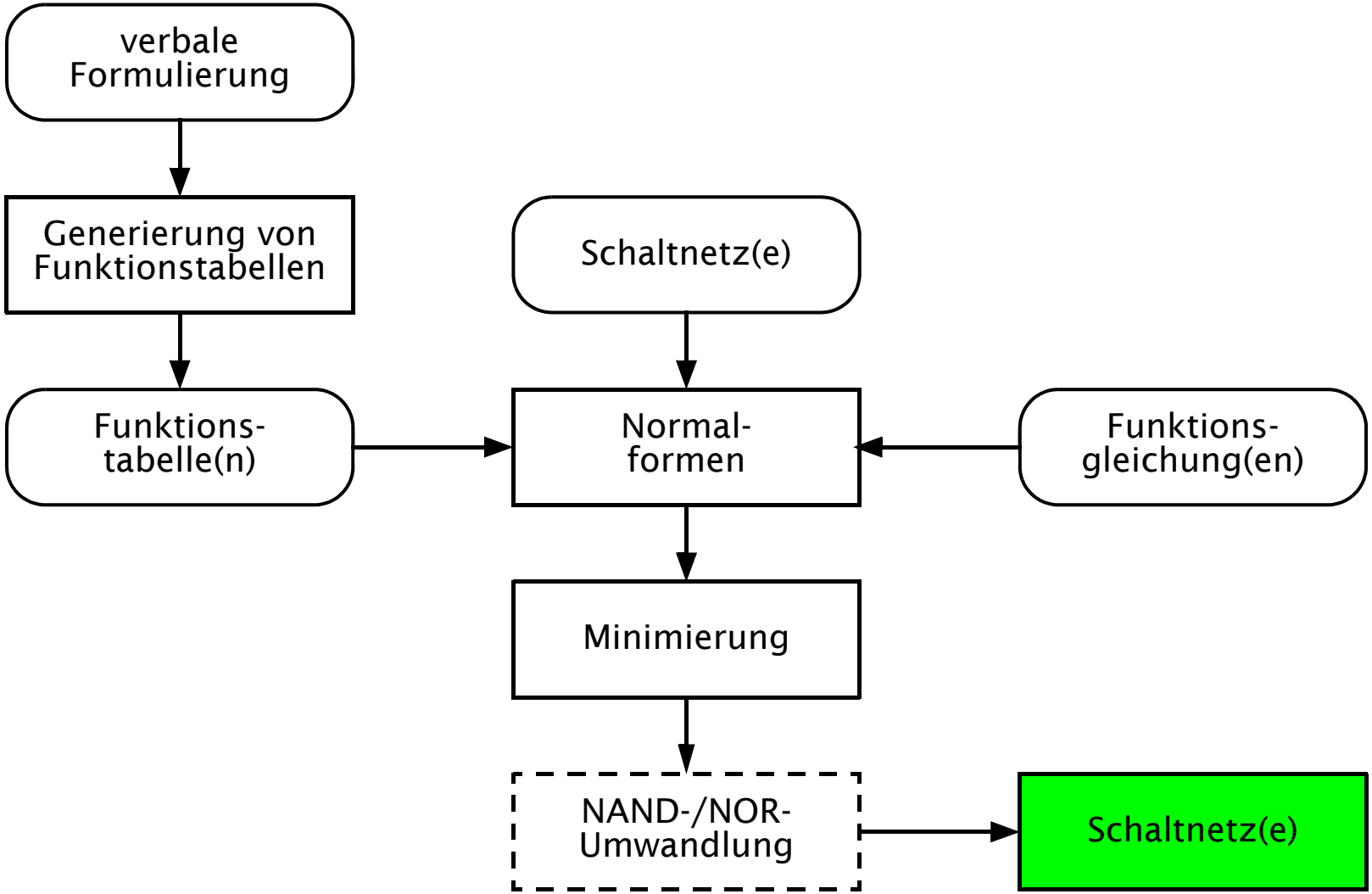
| Nr | 0 | 1 | 2 | 4 | 5 | 10 | $W_i=\Sigma(S_j)$ | |
|---|---|---|---|---|---|---|---|---|
| 0,1,4,5 | | | | | | | | |
| 0,2,8,10 | | | | | | | | |
| 0,4,8,12 | | | | | | | | |
| 8,10,12,14 | | | | | | | | |
| $S_j=1/R_j$ | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

f(a, b, c, 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

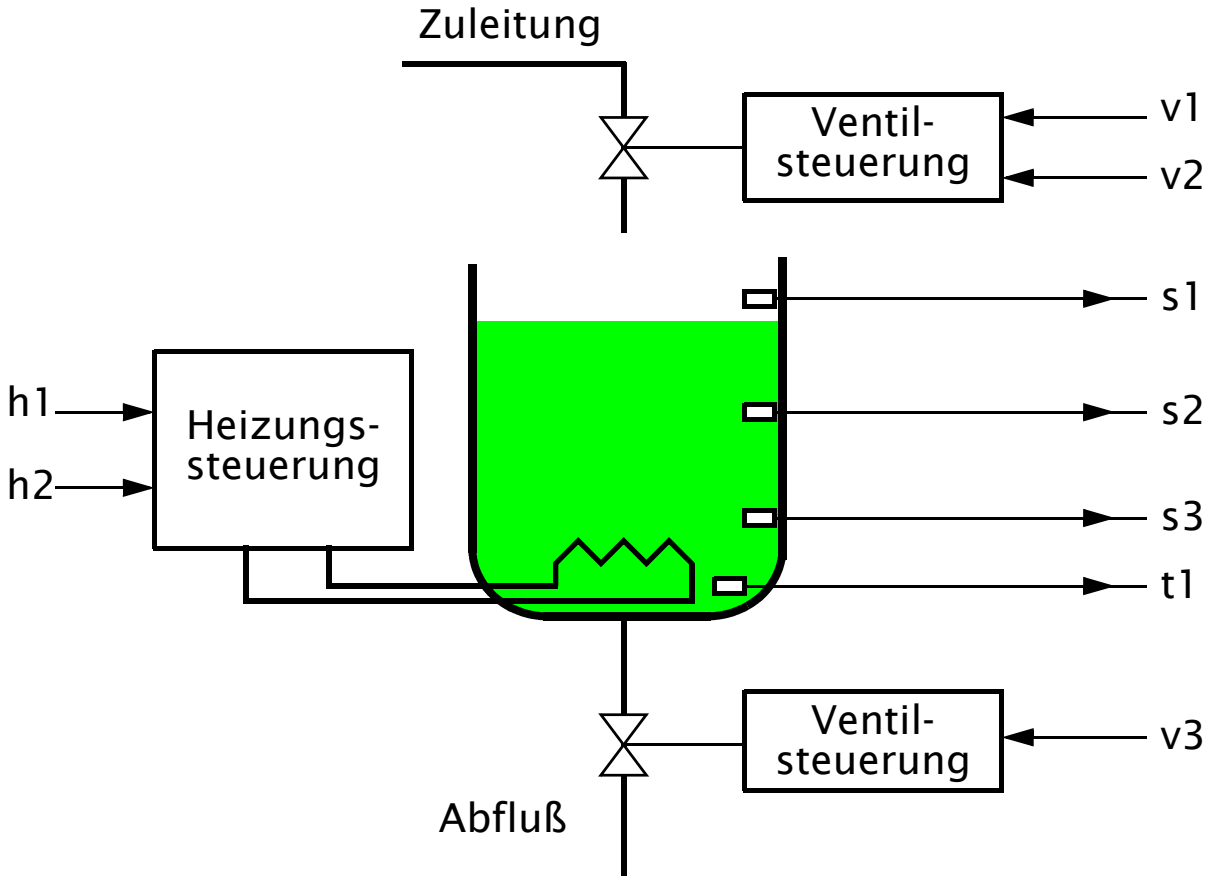


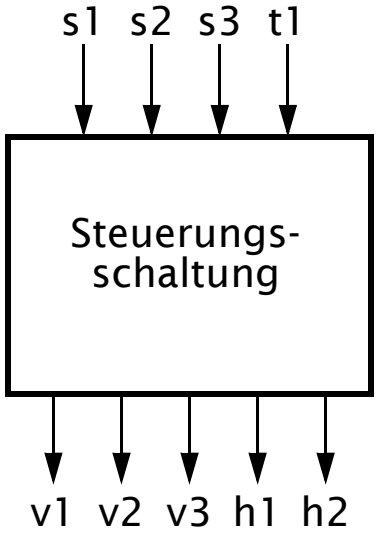
- Entwurf einer Steuerungsschaltung (als Schaltnetz) für eine verfahrenstechnische Anlage

- 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 Thermoschalter erfaßt und mit dem Signal *t1* an die Steuerungsschaltung gemeldet.

| t1 | Temperatur |
|---|---|
| 0 | Temp. < Soll |
| 1 | Temp. ≥ Soll |

- Die Heizung kann über eine Heizungssteuerung in drei Leistungsstufen 0%, 50% und 100% der Leistung mit zwei Signalen *h1* und *h2* geschaltet werden.

| h2 | h1 | Leistung |
|---|---|---|
| 0 | 0 | 0% |
| 0 | 1 | 50% |
| 1 | 1 | 100% |

- Solange der Behälter nicht leer ist und die Flüssigkeit warm genug ist, kann der Ablauf über eine Ventilsteuerung mit dem Signal *v3* geöffnet werden.

- Steuerungstabelle

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

| Flüssigkeitspegel H | Temp. ≥ Soll | Temp. < Soll |
|---|---|---|
| s1 < H | Heizung 0%<br>Ablauf offen<br>Zulauf zu | Heizung 100%<br>Ablauf zu<br>Zulauf zu |
| s2 < H < s1 | Heizung 0%<br>Ablauf offen<br>Zulauf halb offen | Heizung 50%<br>Ablauf zu<br>Zulauf halb offen |
| s3 < H < s2 | Heizung 0%<br>Ablauf offen<br>Zulauf ganz offen | Heizung 50%<br>Ablauf zu<br>Zulauf ganz offen |
| H < s3 | Heizung 0%<br>Ablauf zu<br>Zulauf ganz offen | Heizung 0%<br>Ablauf zu<br>Zulauf ganz offen |

- Funktionstabelle

| Eingangssignale | | | | Ausgangssignale | | | | |
|---|---|---|---|---|---|---|---|---|
| s3 | s2 | s1 | t1 | v1 | v2 | v3 | h1 | h2 |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

- Kanonische Disjunktive Normalformeln:

v1(s3, s2, s1, t1) =

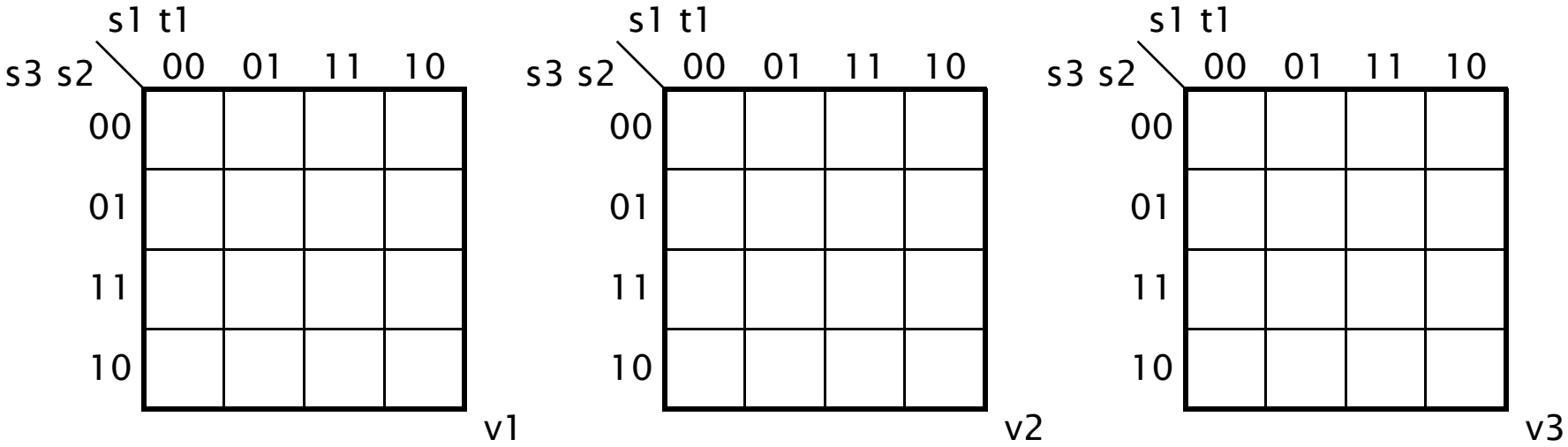
v2(s3, s2, s1, t1) =

v3(s3, s2, s1, t1) =

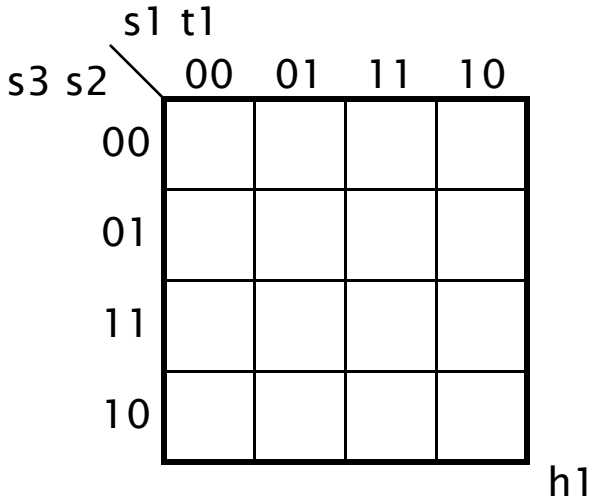
h1(s3, s2, s1, t1) =

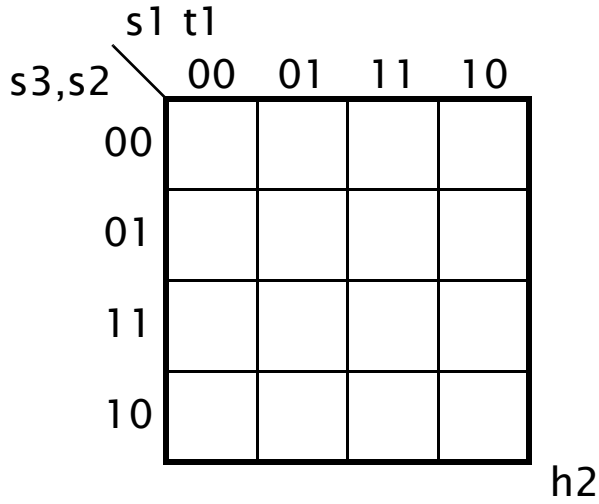
h2(s3, s2, s1, t1) =

- Minimierung mit der KV-Methode



- Minimierung mit der KV-Methode



- 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 · 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 · 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 · t1': Die Heizung arbeitet mit voller Leistung, wenn der Kessel voll ist (s1) und die Soll-Temperatur nicht erreicht ist (t1').

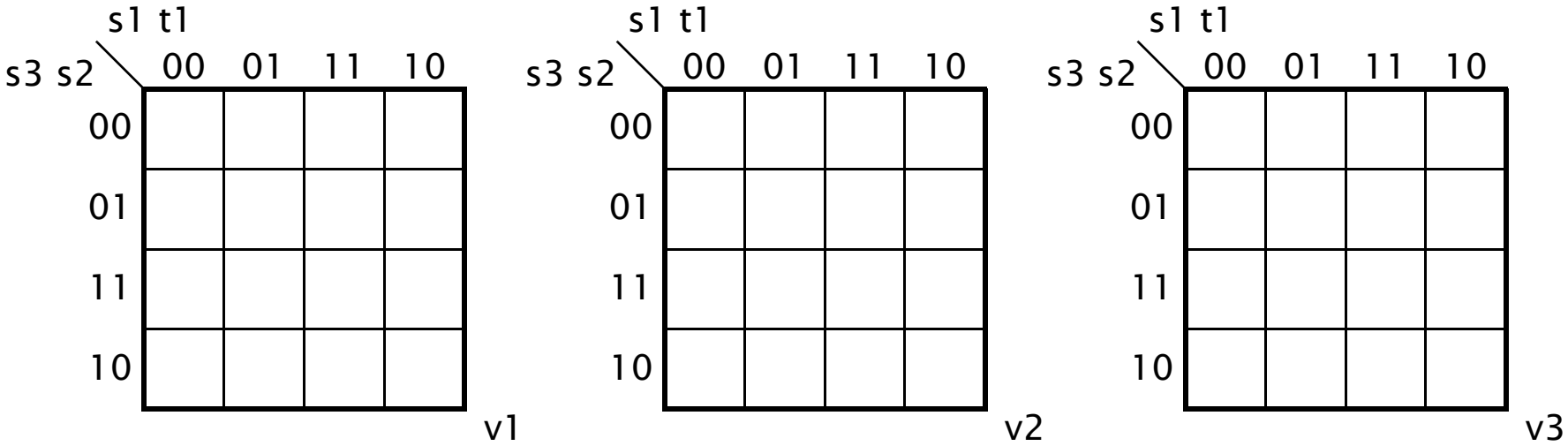
- Erweiterte Analyse
  - Im Beispiel konnten die Ausgangswerte für einige Kombinationen der Eingangssignale beliebig belegt werden, weil sie physikalisch nicht möglich sind.
  - Diese Aussage gilt aber, nur solange man von einer korrekten Funktionsweise aller Sensoren ausgeht.
  - Läßt man diese Annahme 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 Kontrollampe *x1* erweitert, die einem Wartungstechniker einen defekten Sensor meldet.

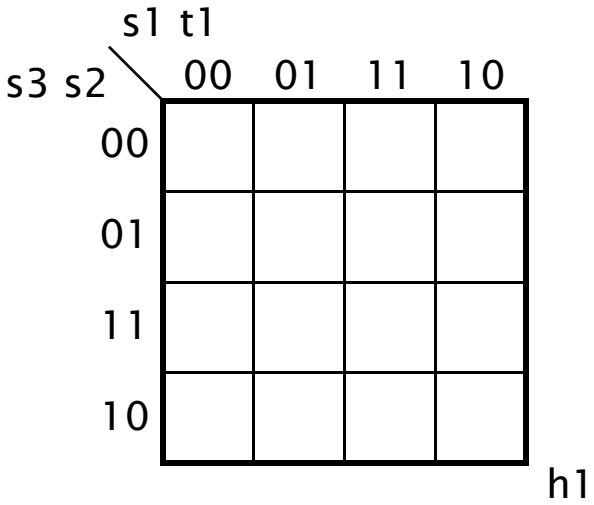
- erweiterte Funktionstabelle mit Fehleranzeige

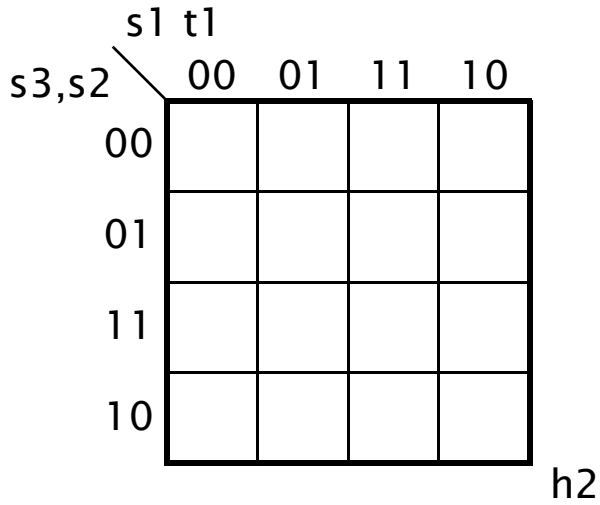
| Eingangssignale | | | | Ausgangssignale | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| s3 | s2 | s1 | t1 | v1 | v2 | v3 | h1 | h2 | x1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

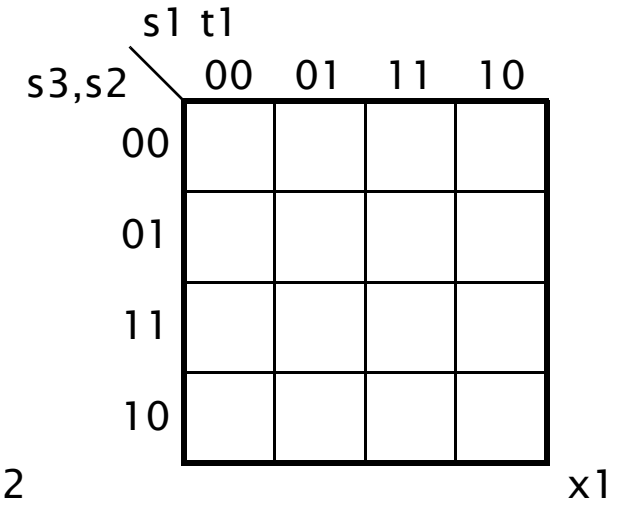
- Minimierung mit der KV-Methode



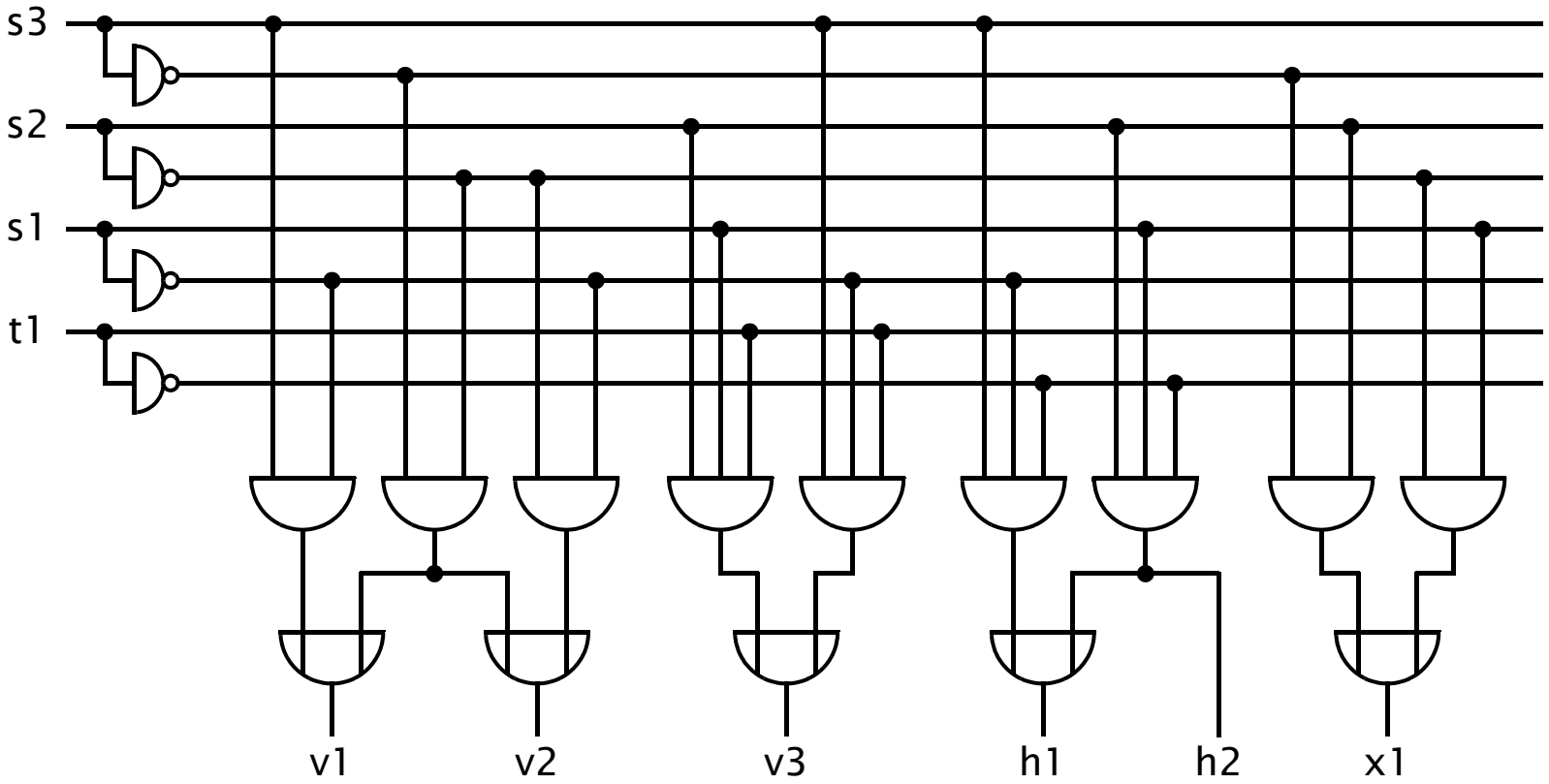
- Minimierung mit der KV-Methode



- Steuerungsschaltung



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

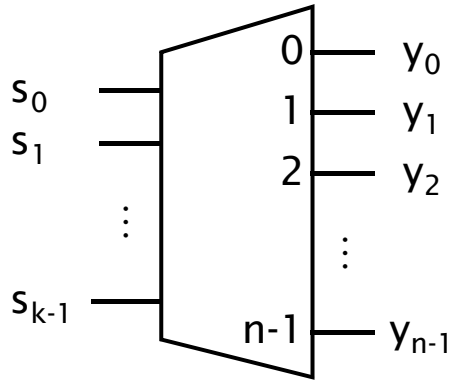
Funktionsgleichung:

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

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

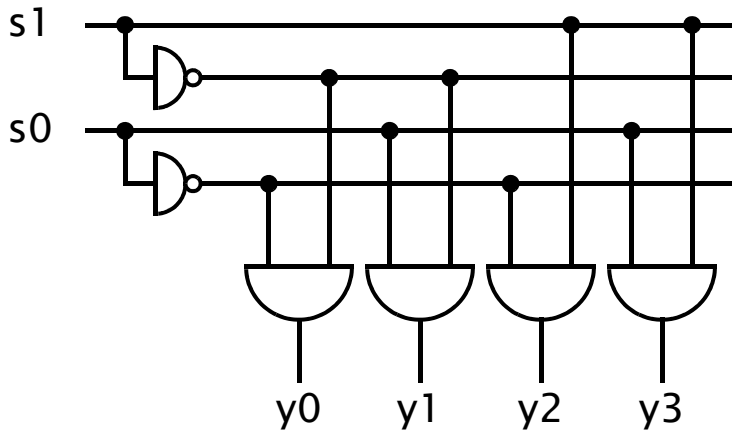
$(i)_{10}$ Nummer des Datenausgangs

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

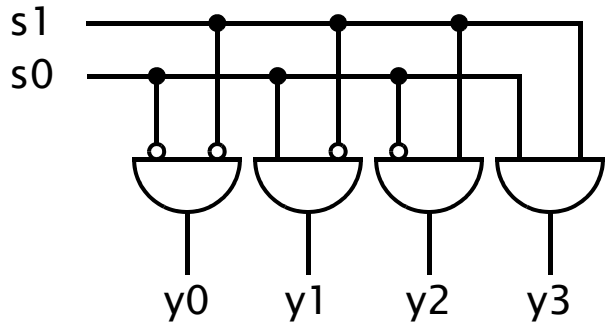


- 1-aus-4-Dekoder
  - Funktionstabelle und Schaltbilder

| 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 |

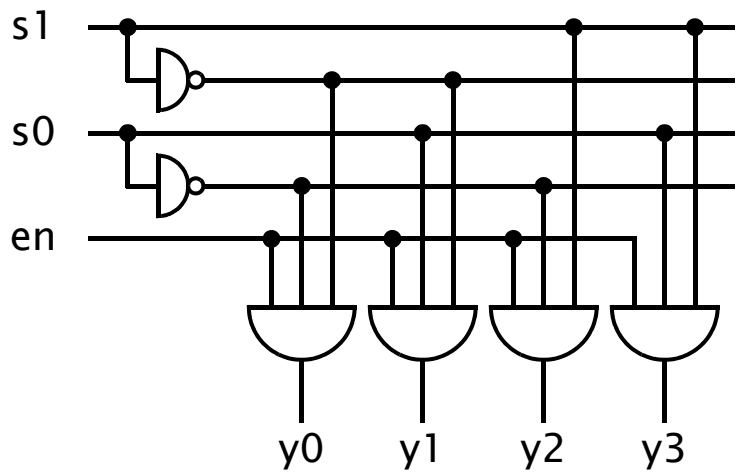



  - Funktionsgleichungen



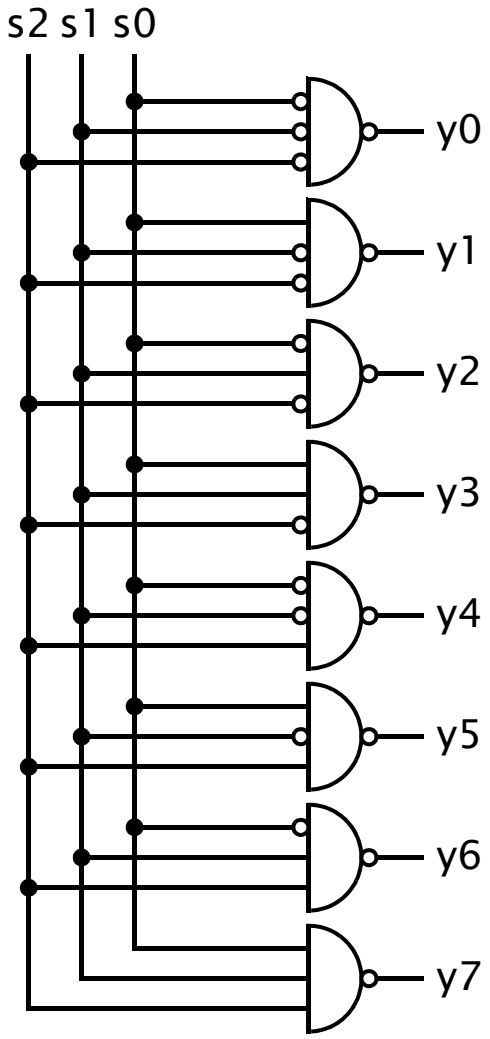
- 1-aus-4-Dekoder mit einem Enable-Eingang
  - Dekodierer-Schaltnetze sind in der Praxis oft mit einem zusätzlichen Steuereingang (sog. Enable-Eingang, en, EN) ausgestattet.
  - Dekodierer mit einem Enable-Eingang aktiviert den selektierten Ausgang nur dann, wenn der Enable-Eingang auch aktiv (=1) ist, 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 |

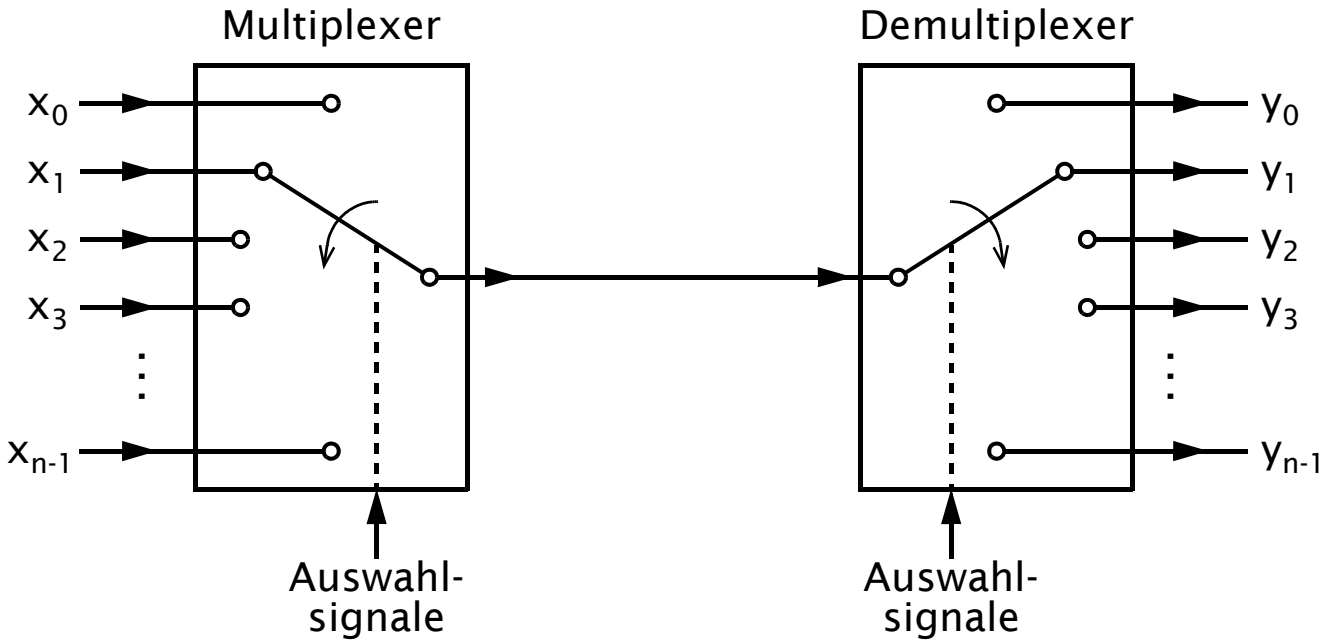


- 1-aus-8-Dekodierer mit inverierten Ausgängen
  - Funktionstabelle und Schaltbild

| 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-Seriel- und Seriel-Parallelwandlung von Daten
  - Erzeugung verschiedener Logikfunktionen



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

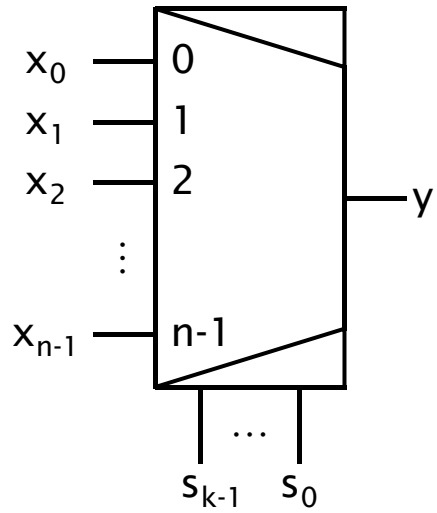
Funktionsgleichung:

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

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

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

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

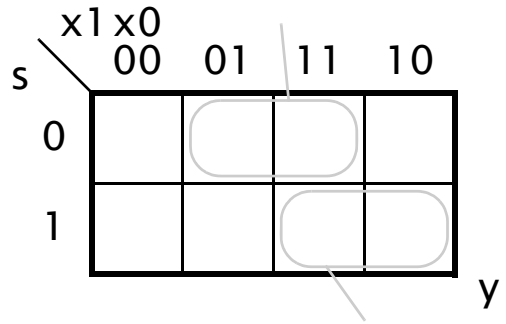


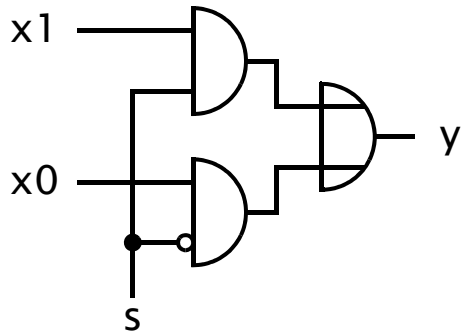
- 1-aus-2-Multiplexer
  - Funktionstabellen, KV-Diagramm und Schaltbilder

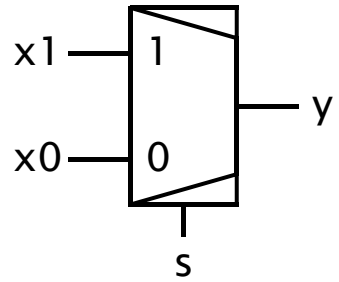
| s | x1 | x0 | y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

=>

| s | y |
|---|---|
| 0 | x0 |
| 1 | x1 |



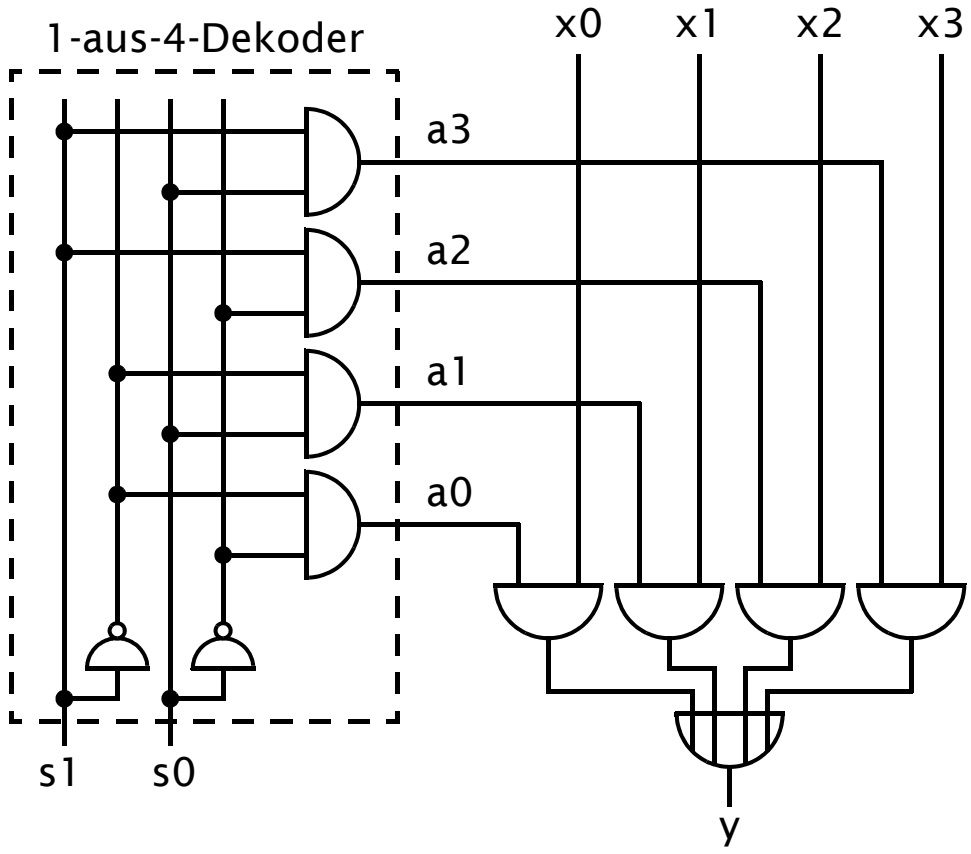




  - Funktionsgleichung

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

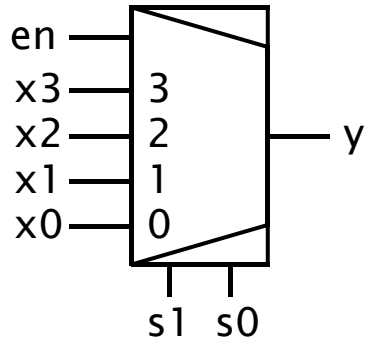
| s1 | s0 | y |
|---|---|---|
| 0 | 0 | x0 |
| 0 | 1 | x1 |
| 1 | 0 | x2 |
| 1 | 1 | x3 |

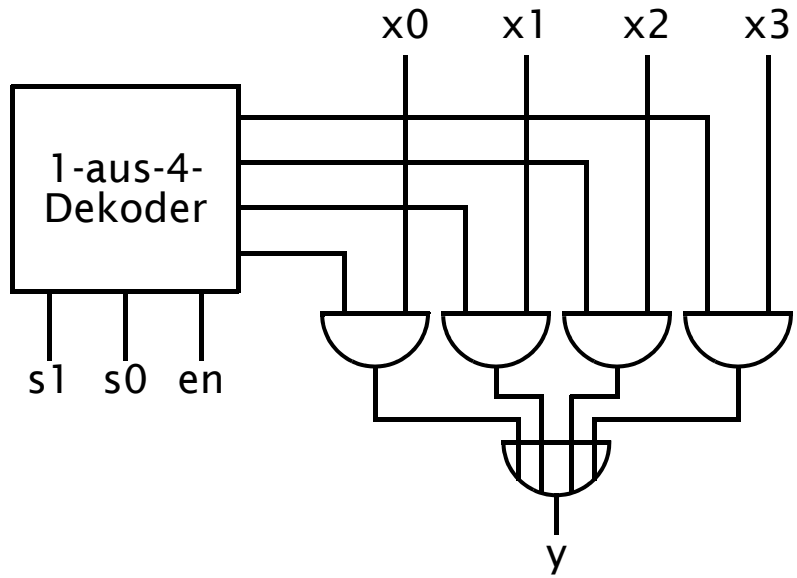
  - Funktionsgleichung



- 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: $s_0, s_1, ..., s_{k-1}$
  - n (Daten-)Ausgänge: $y_0, y_1, ..., y_{n-1}$ mit $n = 2^k$
  - Funktionsweise:
    1-zu-n-Demultiplexer schaltet in der Abhängigkeit von den Steuereingängen den Dateneingang auf einen von n Ausgängen durch.

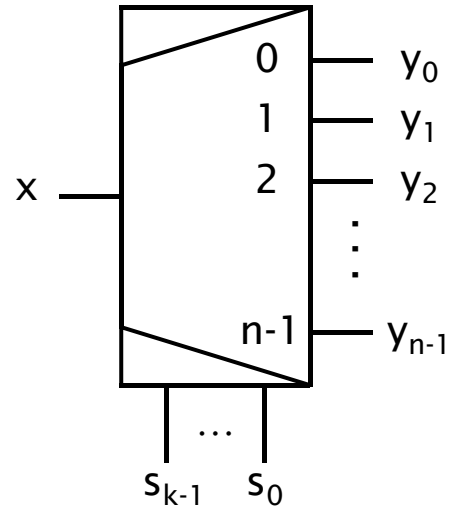
Funktionsgleichung:

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

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

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

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

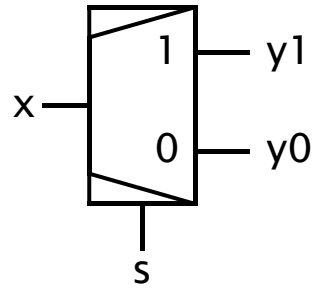


- 1-zu-2-Demultiplexer
  - Funktionstabellen und Schaltbilder

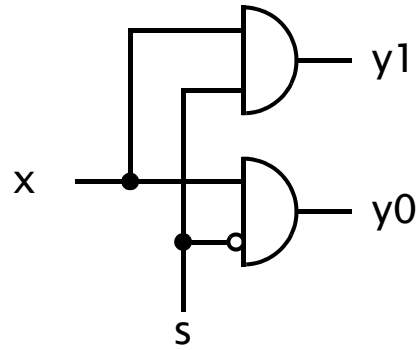
| s | x | y1 | y0 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 |

=>

| s | y1 | y0 |
|---|---|---|
| 0 | 0 | x |
| 1 | x | 0 |

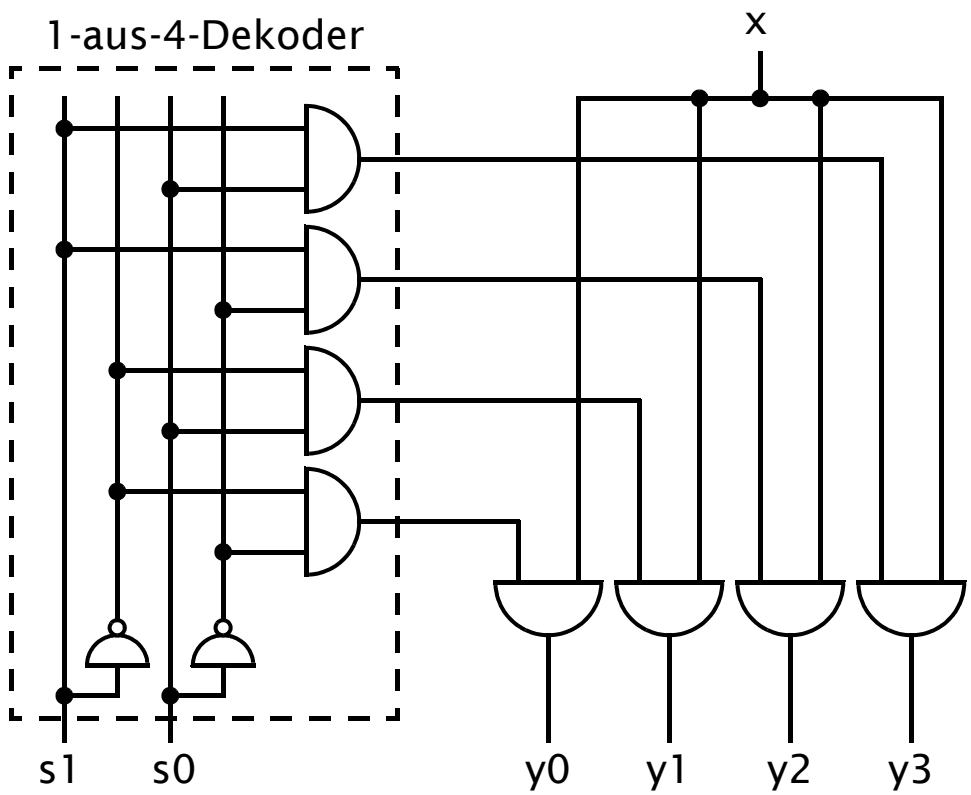



  - Funktionsgleichung



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

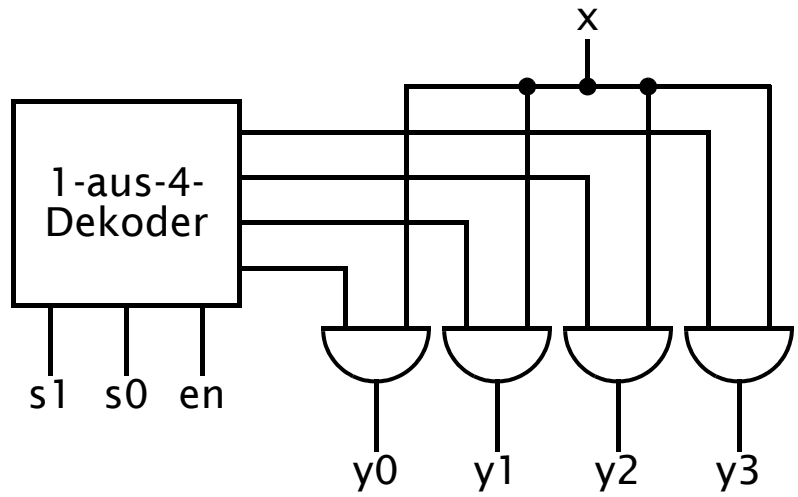
| s1 | s0 | y3 | y2 | y1 | y0 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | x |
| 0 | 1 | 0 | 0 | x | 0 |
| 1 | 0 | 0 | x | 0 | 0 |
| 1 | 1 | x | 0 | 0 | 0 |



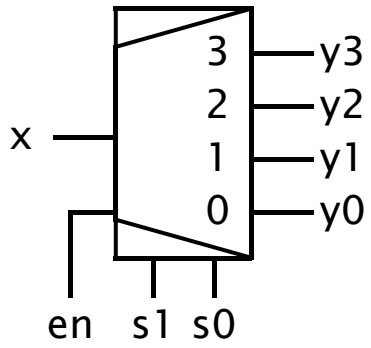

  - Funktionsgleichungen

- 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

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

$$= e_i \cdot p_1 + e_i' \cdot p_0$$

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

$$p_1(e_0, e_1, ..., e_{i-1}, e_{i+1}, ..., e_{n-1}) := f(e_0, e_1, ..., e_i=1, ..., e_{n-1})$$
$$p_0(e_0, e_1, ..., e_{i-1}, e_{i+1}, ..., e_{n-1}) := f(e_0, e_1, ..., e_i=0, ..., e_{n-1})$$

- 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 p0 und p1 ohne Variable a:

     $p0(b, c, d) := f(a=0, b, c, d)$

     $p1(b, c, d) := f(a=1, b, c, d)$

  2. Berechnung der partiellen Funktionen p0 und p1:

     $p0(b, c, d) := f(a=0, b, c, d) = 0 \cdot b \cdot c' + b' \cdot (0' \cdot c + 0 \cdot d) = b' \cdot c$

     $p1(b, c, d) := f(a=1, b, c, d) = 1 \cdot b \cdot c' + b' \cdot (1' \cdot c + 1 \cdot d) = b \cdot c' + b' \cdot d$

  3. Zusammenfassung der Resultate:

     $f(a, b, c, d) = a' \cdot p0(b, c, d) + a \cdot p1(b, c, d)$

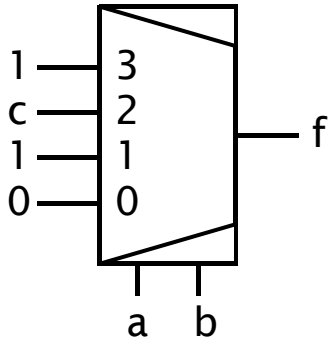
     $= a' \cdot (b' \cdot c) + a \cdot (b' \cdot d + b \cdot c')$

- Entwicklungssatz von Shannon

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

- 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

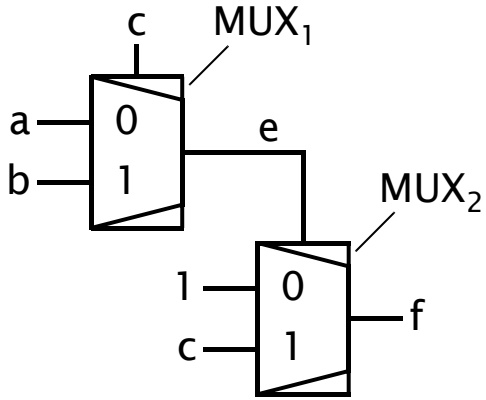
x3 := 1

  - Funktionsgleichung des 1-aus-4-Multiplexers:

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

- Analyse multiplexer-basierender Schaltnetze

  Rekonstruktion/Minimierung boolescher Funktionen aus mehr-stufigen Multiplexer-Schaltnetzen




| | Select | Daten | Daten | Ausgang |
|---|---|---|---|---|
| $MUX_1$ | (s) := c | x0 := a | x1 := b | e |
| $MUX_2$ | (s) := e | x0 := 1 | x1 := c | f |

  - Funktionsgleichung des 1-aus-2-Multiplexers

    $y(s, x0, x1) := x0 \cdot (s') + x1 \cdot (s)$

- Analyse multiplexer-basierender Schaltnetze
  (Fortsetzung des Beispiels)

- 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

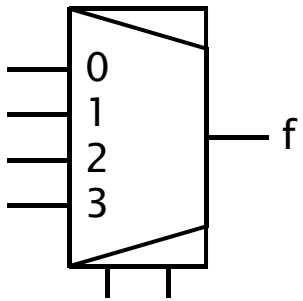
  - Die Fälle 1) und 2) ergeben in der Realisierung einstufige Multiplexer-Schaltnetze.
  - Der Fall 3) ergibt in der Realisierung mehrstufige Multiplexer-Schaltnetze.

- Synthese multiplexer-basierender Schaltnetze, Fall 1)
  1. Zuordnung boolescher Variablen zu Select-Eingängen eines 1-aus-n-Multiplexers

     $(s_{k-1}, s_{k-2}, ..., s_1, s_0) := (e_{k-1}, e_{k-2}, ..., e_1, e_0)$
  2. Bestimmung der Dateneingänge $x_0, x_1, ..., x_{n-1}$ mit $n=2^k$

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

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



- Synthese multiplexer-basierender Schaltnetze, Fall 2)

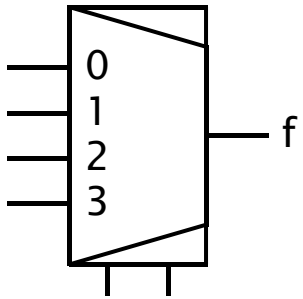
  Beispiel: Realisierung g(a, b, c) = a · (b + c') + a' · b · c mit einem 1-aus-4-Multiplexer: $s_1$=?, $s_0$=?, $x_0$=?, $x_1$=?, $x_2$=?, $x_3$=?

  1. Zuordnung boolescher Variablen zu zwei Select-Eingängen

     2 aus 3 Variablen ergeben 3 Möglichkeiten:

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

  2. Bestimmung der Dateneingänge:



- Synthese multiplexer-basierender Schaltnetze, Fall 3)

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

  1. Zuordnung boolescher Variablen zu dem Select-Eingang

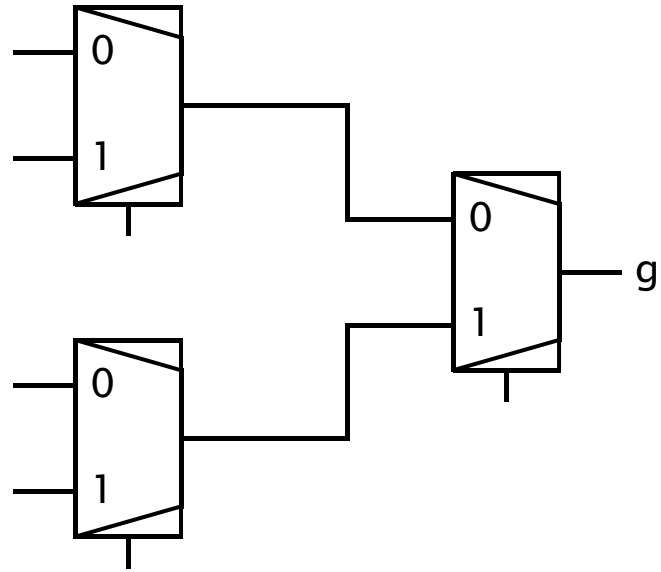
     ein 1-aus-2-Multiplexer hat nur einen Select-Eingang $s_0$

     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.

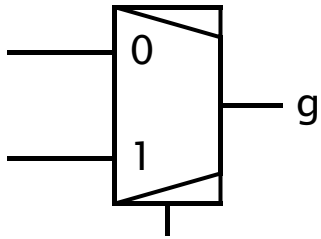
- Wenn an den Dateneingängen keine einfachen Funktionen, die nur aus einer negierten oder nicht negierten Variable oder aus einer booleschen Konstante bestehen, sondern zusammengesetzte boolesche Ausdrücke entstehen, dann ist die Dekomposition für diese Ausdrücke zu wiederholen.



- Synthese multiplexer-basierender Schaltnetze, Fall c)

  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
  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.



- 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.

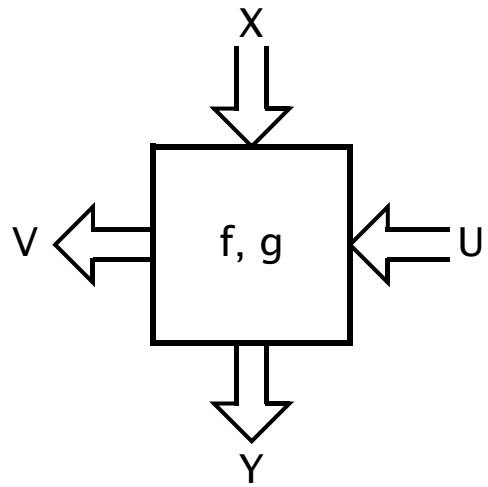
  - Schaltketten bilden die Basis für alle Operationen, die parallel auf digitale Wörter angewendet werden.

  - 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

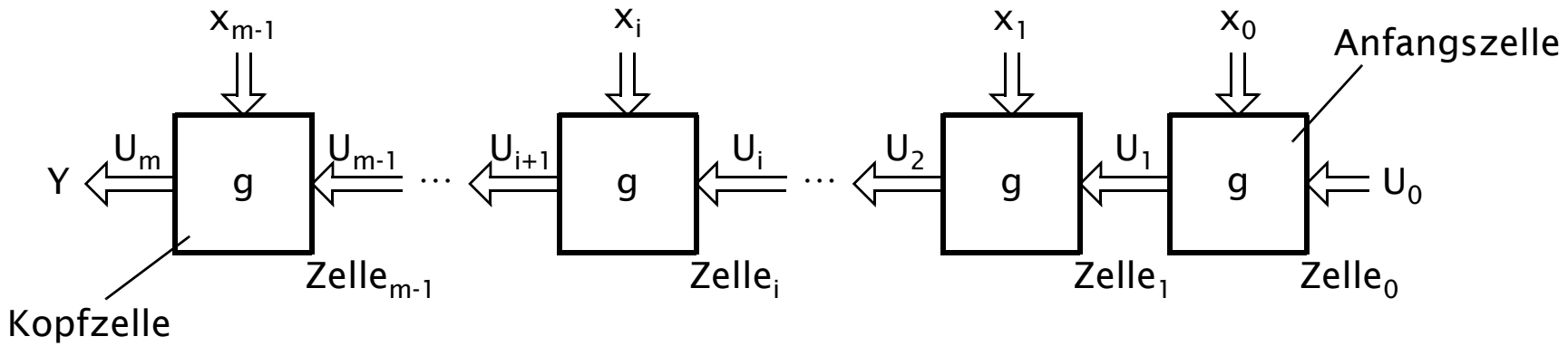
$B_z$ = (X, Y, U, V, f, g) mit

| | | |
|---|---|---|
| X | Eingabevektor | |
| Y | Ausgabevektor | |
| U | Übergabevektor | |
| V | Übergabevektor | |
| f: | $X \times U \rightarrow Y$ | Ausgabefunktion |
| g: | $X \times U \rightarrow V$ | Übergabefunktion |

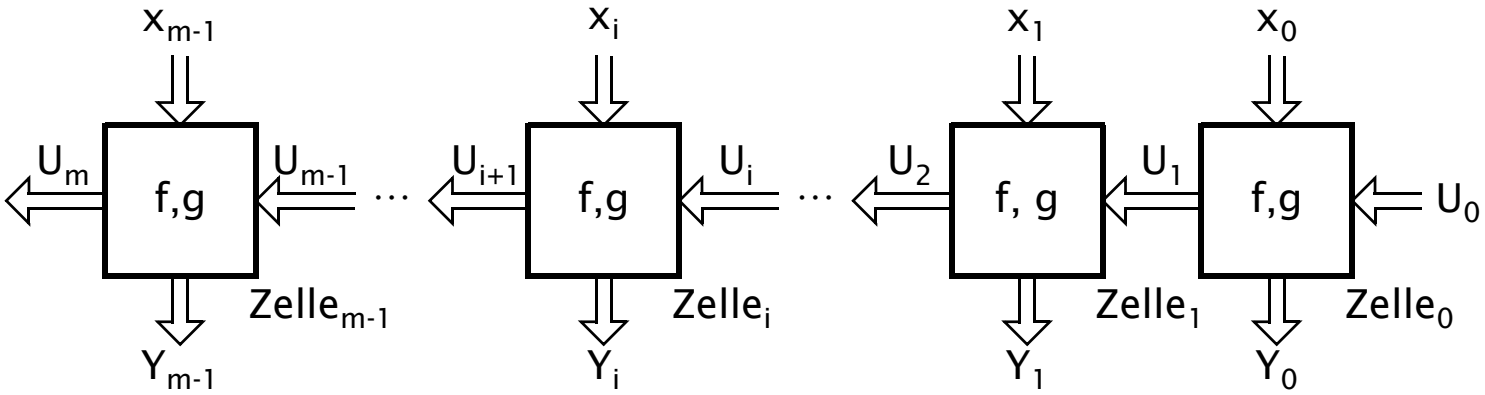



  - Ein Schaltnetz mit n Eingangsvariablen, das auf n/k gleichartige Basiszellen mit Eingabevektoren der Länge k aufgeteilt ist, bezeichnet man als ein (*ortssequntielles*) *k-iteratives* Schaltnetz.

- Schaltkette Typ 1

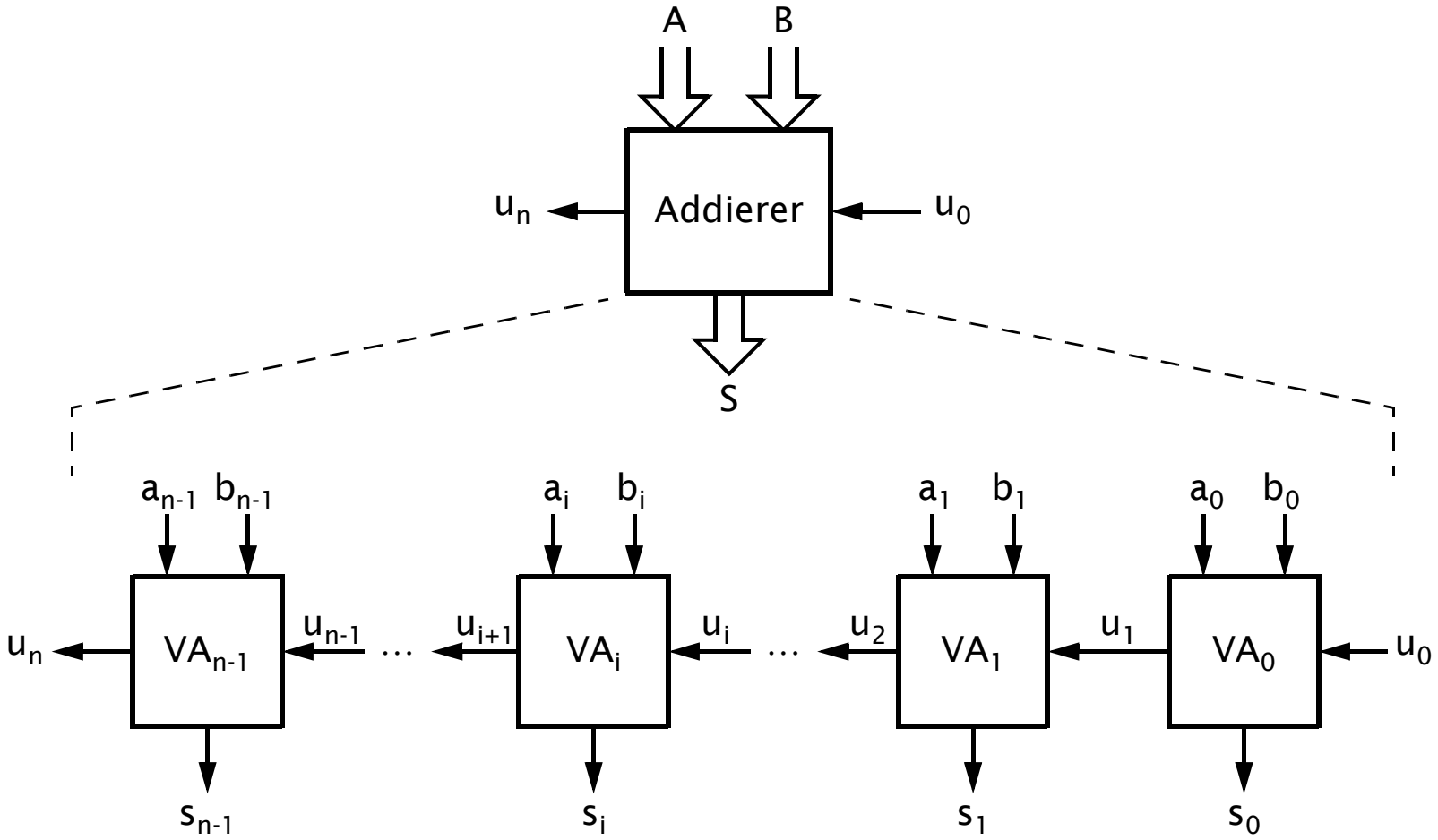



- Schaltkette Typ 2



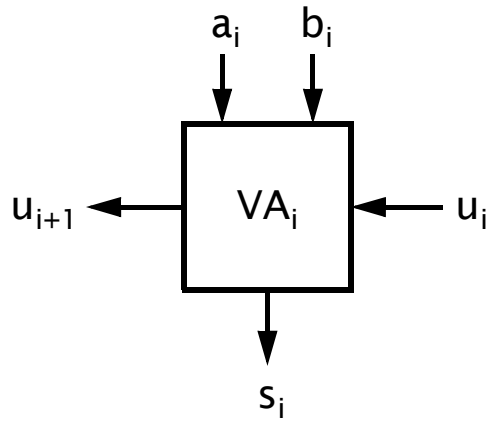
- Systematischer Entwurf iterativer Schaltnetze
  1. Festlegung der Länge k des Eingabevektors X einer Basiszelle (Je kleiner der Wert k ist, desto höher die Gesamtverzögerung.)
  2. Bestimmung der Menge der Übergabevariablen durch funktionale Zerlegung des Eingabevektors mit dem gleichen Verhalten; => 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



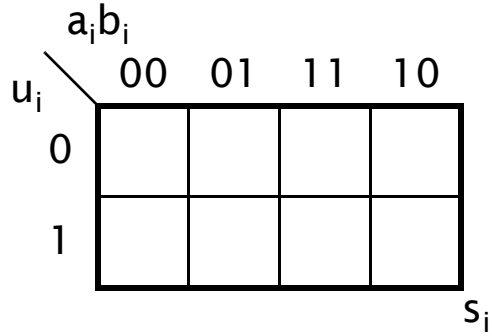
- Basiszelle: 1-Bit-Addierer (Volladdierer, VA)
  - Schnittstelle
    - Eingangsvariablen $a_i$ und $b_i$ für die Ziffern an der i-ten Position
    - Übertragsvariable $u_i$ aus der vorhergehenden (i-1)-ten Position
    - Ausgangsvariable $s_i$ für das Ergebnis der Addition
    - Übertragsvariable $u_{i+1}$ zu der nachfolgenden (i+1)-ten Position
  - Funktionstabelle

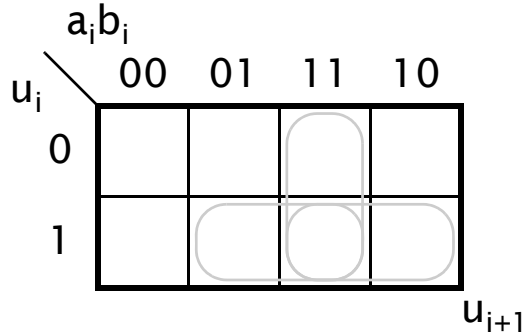
| $u_i$ | $a_i$ | $b_i$ | $s_i$ | $u_{i+1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |



- Basiszelle: 1-Bit-Addierer

  Minimierung mit einem KV-Diagramm



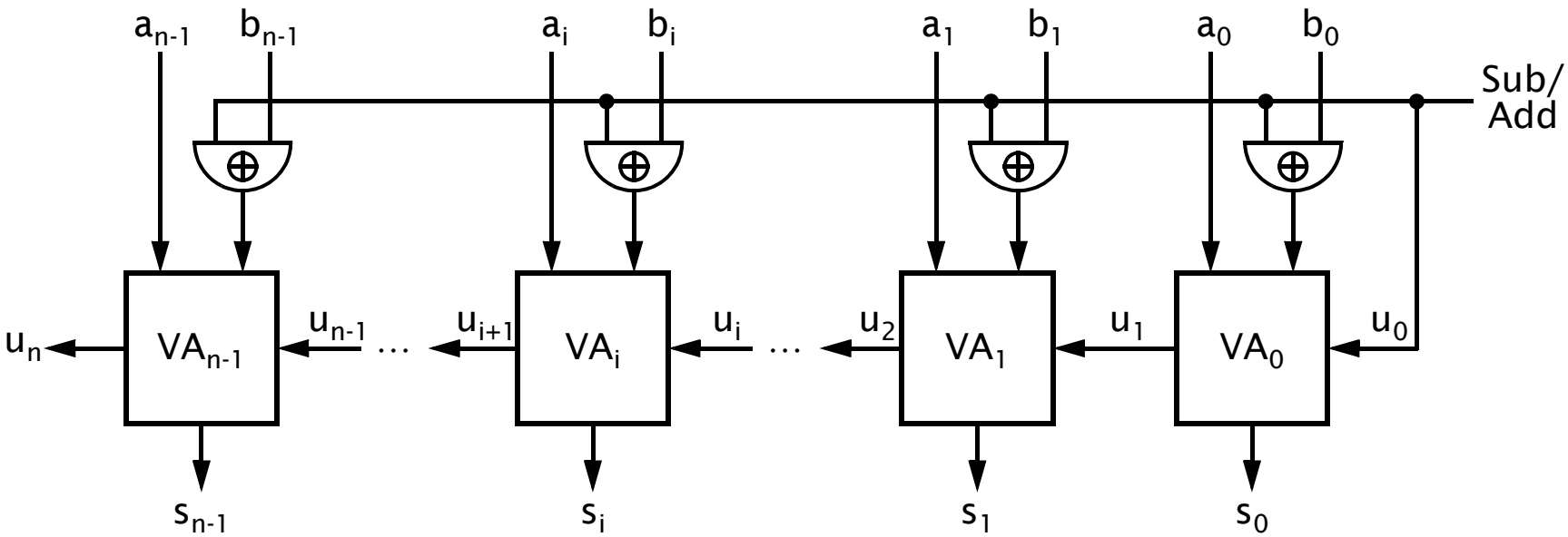

Ausgangsfunktion:

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

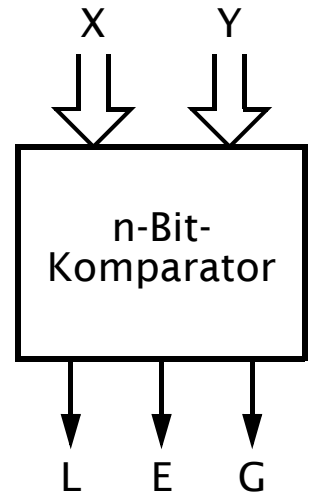
Übergabefunktion:

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

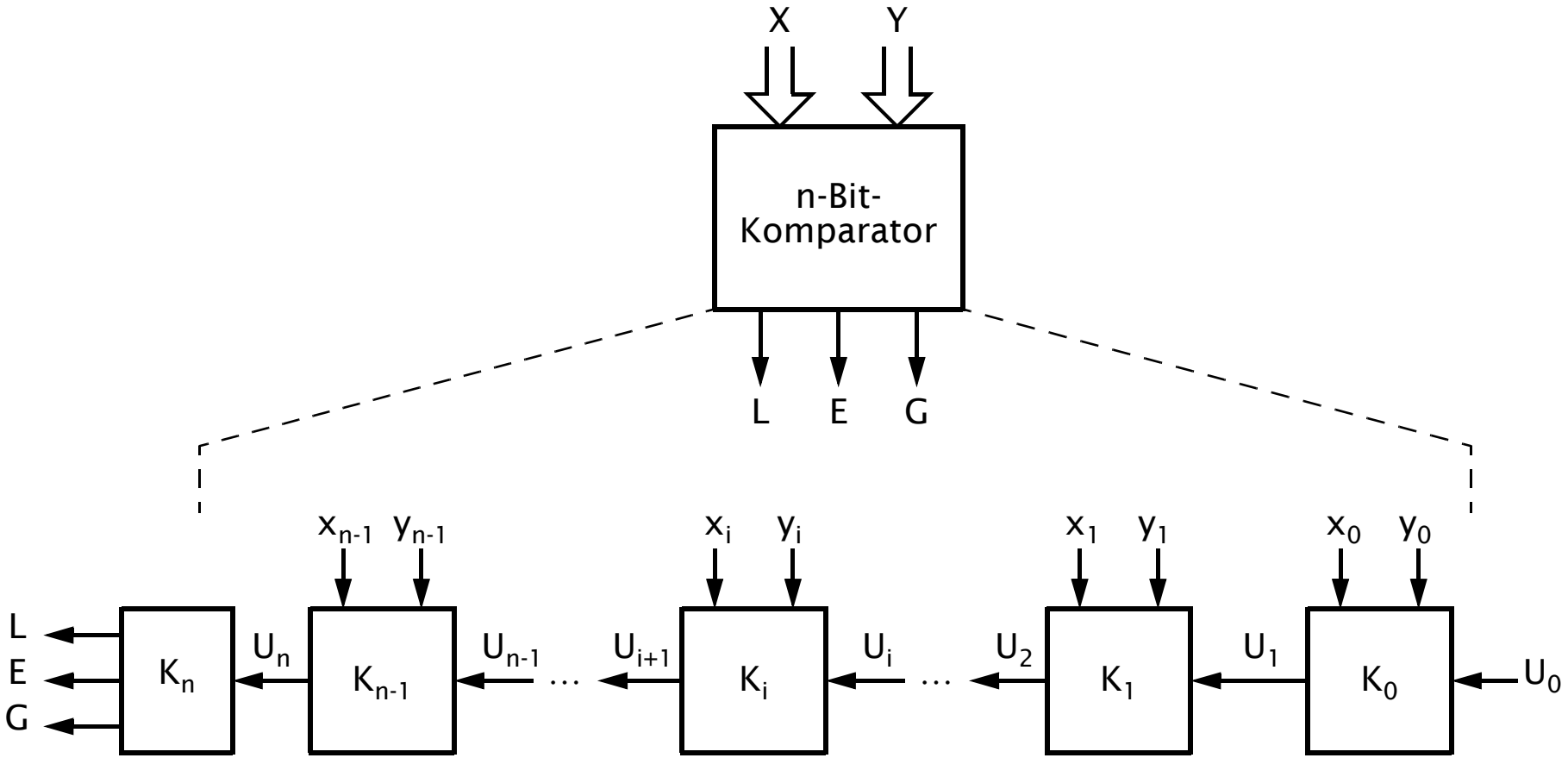
- n-Bit-Addierer/Subtrahierer
  - Erweiterung des n-Bit-Addierers um Invertierlogik und um ein Auswahlsignal Sub/Add zur Bestimmung der Operation
    - Addition wenn Sub/Add = 0
    - Subtraktion wenn Sub/Add = 1



- 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 Dual-zahlen 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

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

  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:

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

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

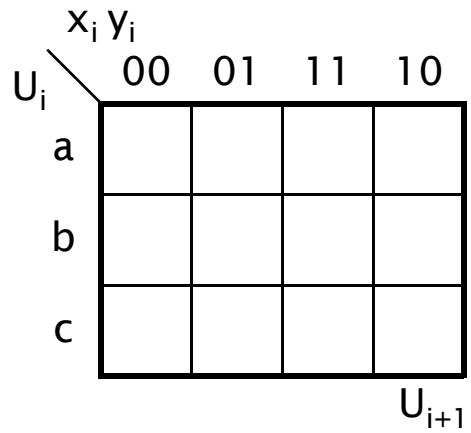
  - symbolisch codierte Werte der Übertragsvariablen $U_{i+1}$, die die i-te Basiszelle an die darauffolgende (i+1)-Basiszelle weiter gibt:

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

| $U_{i+1}$ | 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:

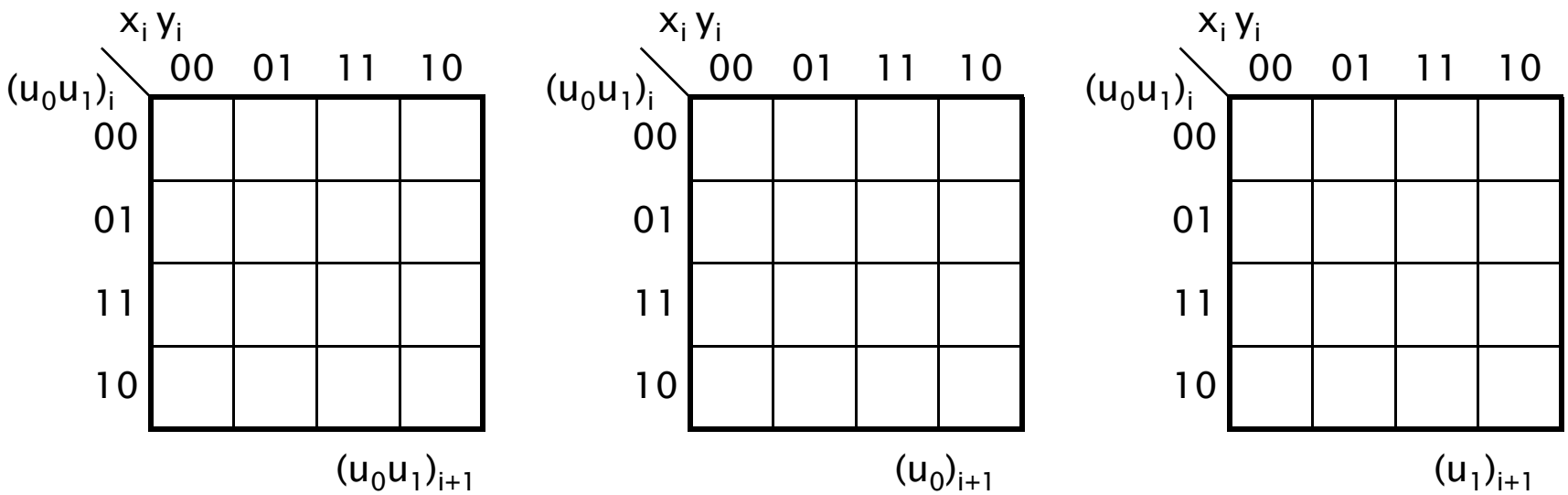
| | $U_i$ | $x_i$ | $y_i$ | $U_{i+1}$ |
|---|---|---|---|---|
| $X_p < Y_p$ | | | | |
| | | | | |
| | | | | |
| | | | | |
| $X_p = Y_p$ | | | | |
| | | | | |
| | | | | |
| | | | | |
| $X_p > Y_p$ | | | | |
| | | | | |
| | | | | |
| | | | | |



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

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

  $U_i = (u_0u_1)_i$  $U_{i+1} = (u_0u_1)_{i+1}$



- 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

$f_i = x_i \cdot y'_i$ und $g_i = x_i + y'_i$

$$(u_0)_{i+1} = f_i + (u_0)_i \cdot g_i$$
$$(u_1)_{i+1} = f_i + (u_1)_i \cdot g_i$$

# Schaltkette

- Bestimmung der Initialisierungswerte für die Übergangsvariablen in der Anfangszelle

  $(u_0)_0 =$

  $(u_1)_0 =$

- Bestimmung der Ausgangsfunktionen für die Kopfzelle

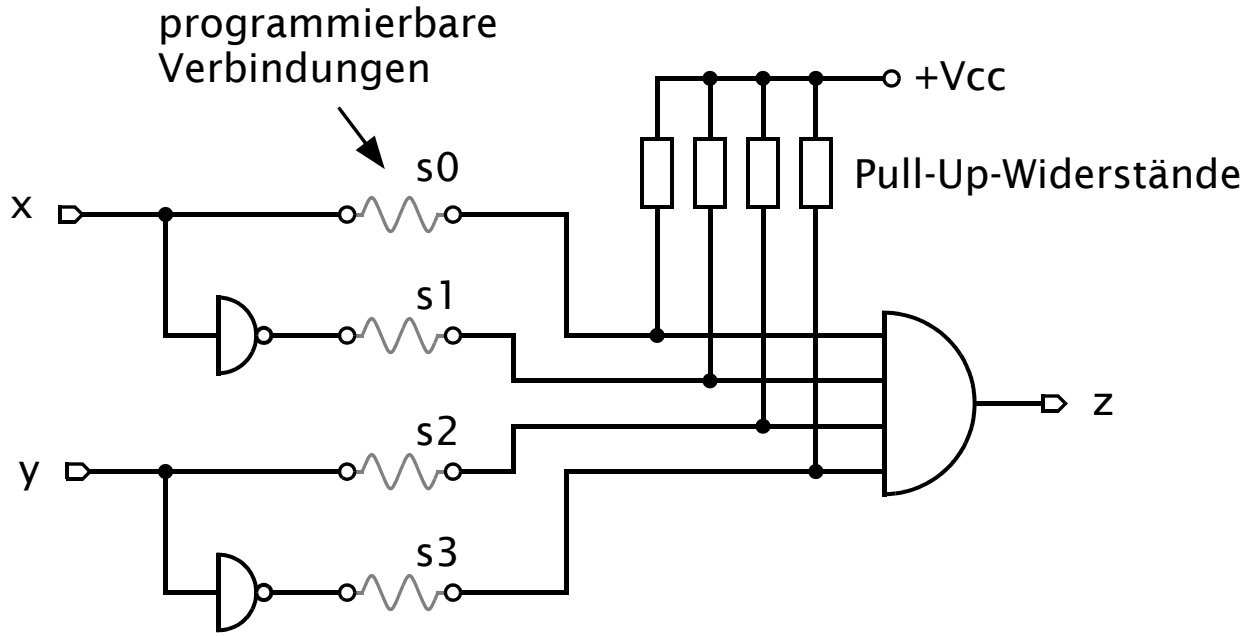
  L =

  E =

  G =

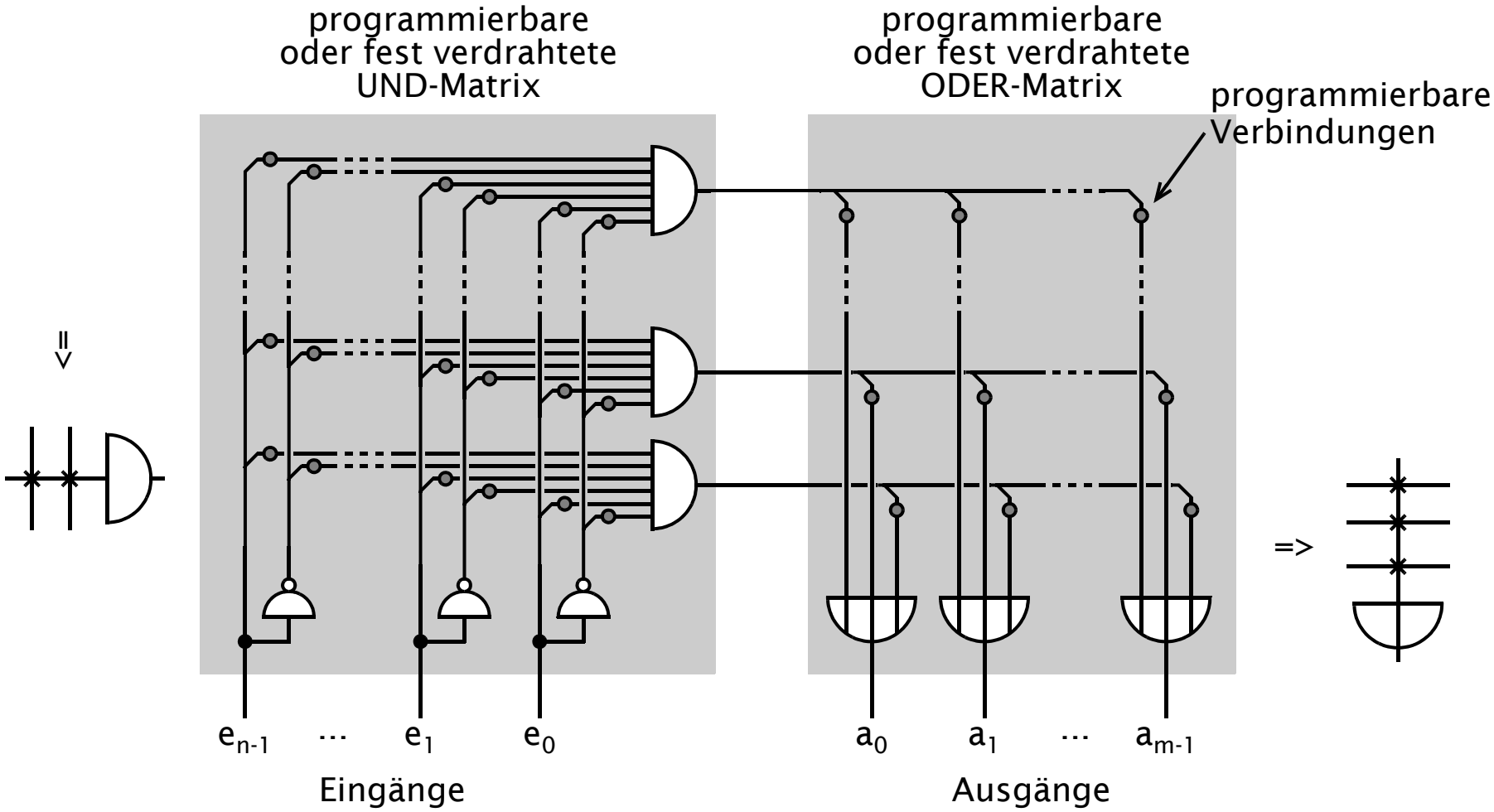
- Realisierung digitaler Schaltungen
  - fest verdrahtete Lösung mit diskreten Elementen (einzelne Gatter)
  - fest verdrahtete Lösung mit Standard-Bausteinen (integrierte Gatter, Multiplexer)
  - fest verdrahtete Lösung mit kunden-/applikationsspezifischen integrierten Schaltungen (ASIC)
  - flexible 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




Grundfunktion: $z(x, y) = x \cdot x' \cdot y \cdot y'$

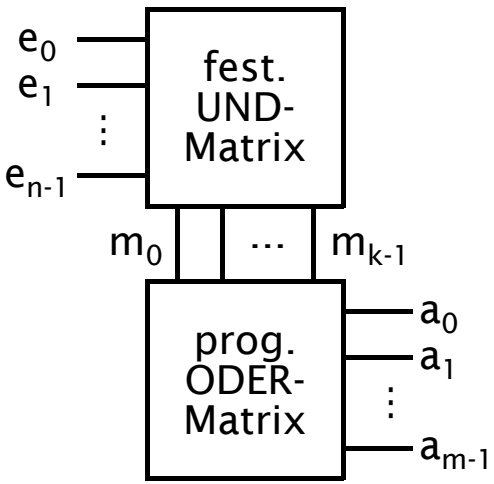
- Grundstrukturen programmierbarer Logikbausteine



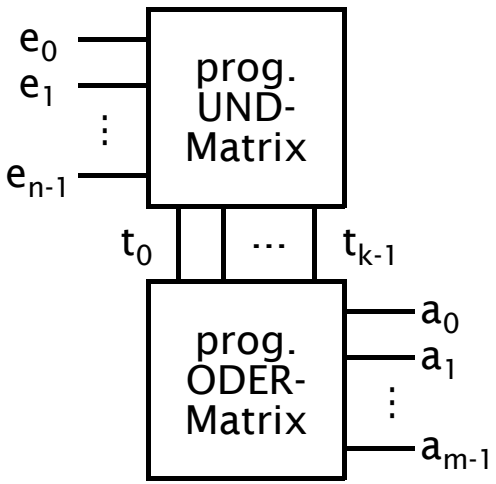
- Grundstrukturen programmierbarer Logikbausteine

PLD (Programmable Logic Device)

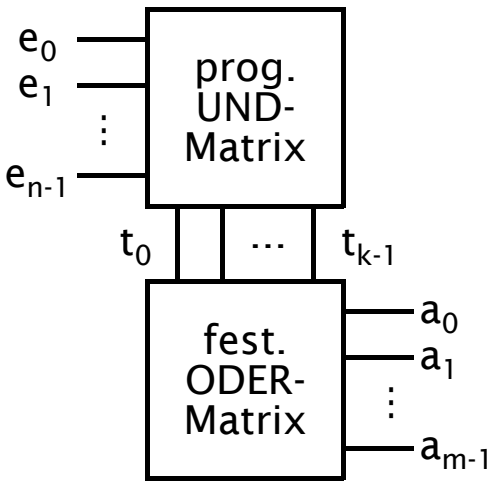
PROM
(Programmable Read
Only Memory)



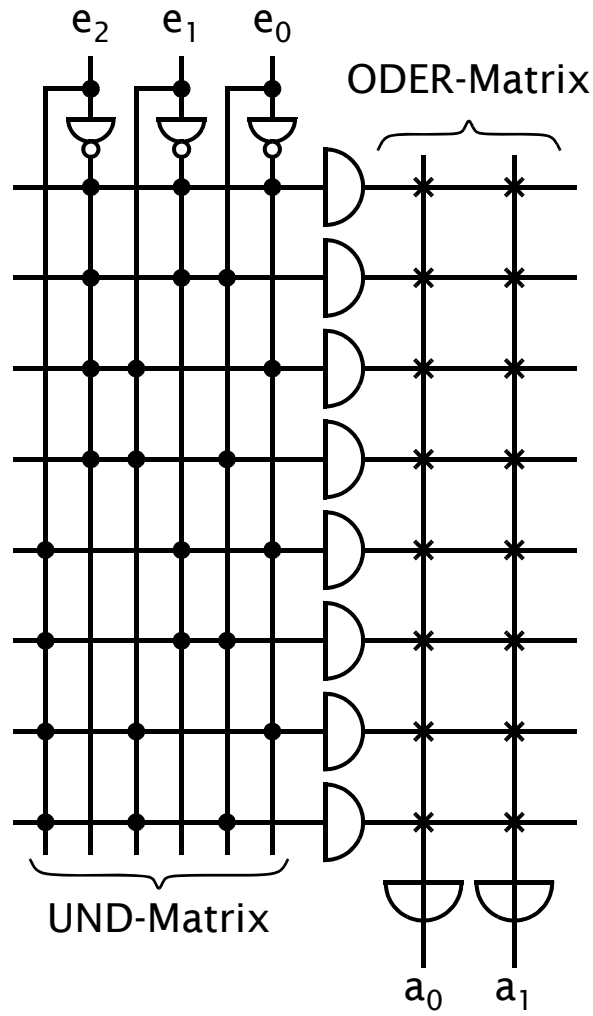

PLA
(Programmable
Logic Array)




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



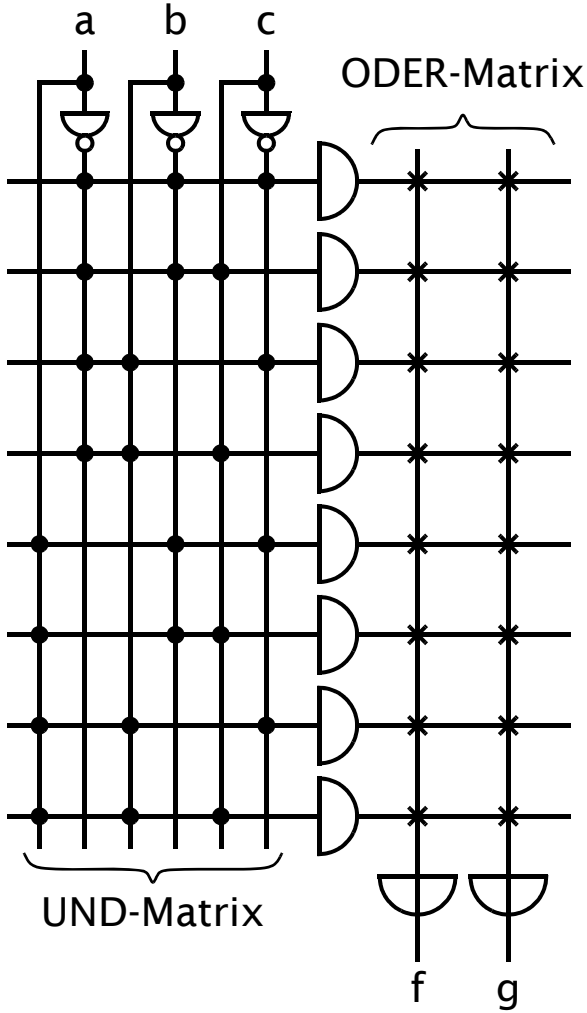
- Realisierung mit 8×2-Bit-PROM

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

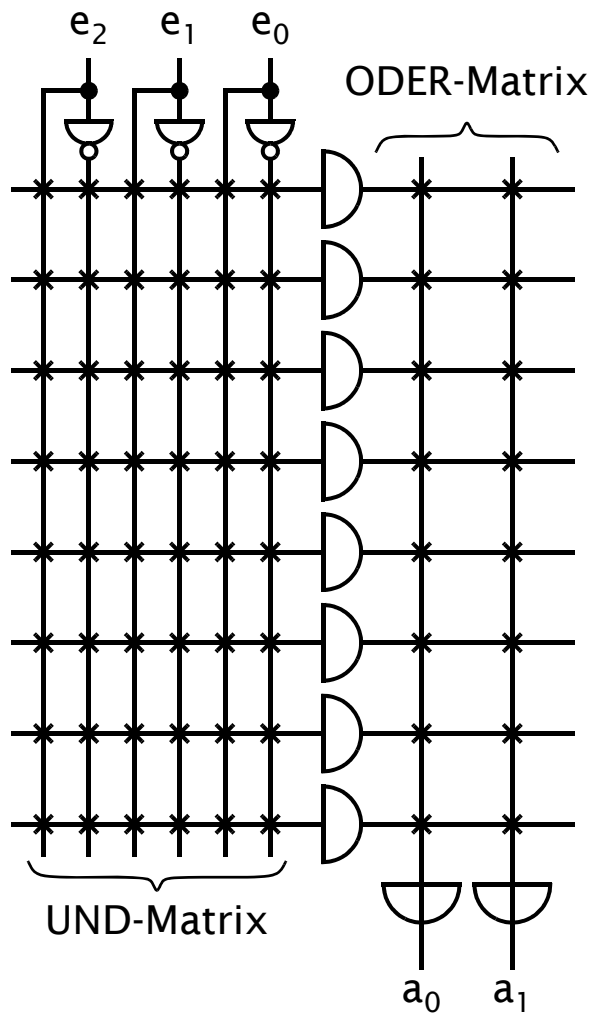
$g(a, b, c) = c \cdot (a + b)$

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

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

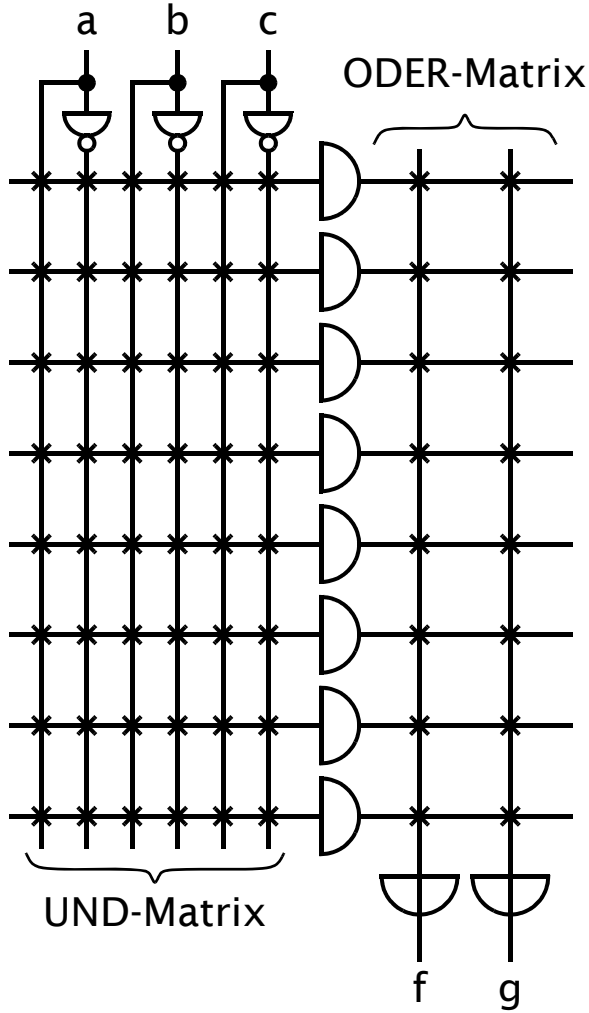


- Realisierung mit PLA (Programmable Logic Array)

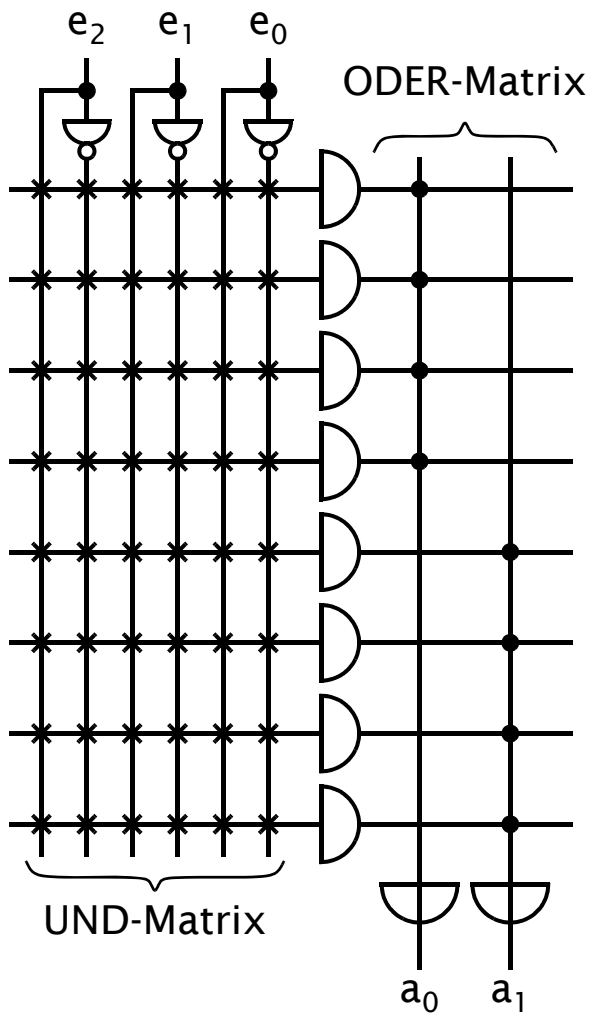
  $f(a, b, c) = a \cdot (b + c') + b \cdot (a \oplus c)$

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

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



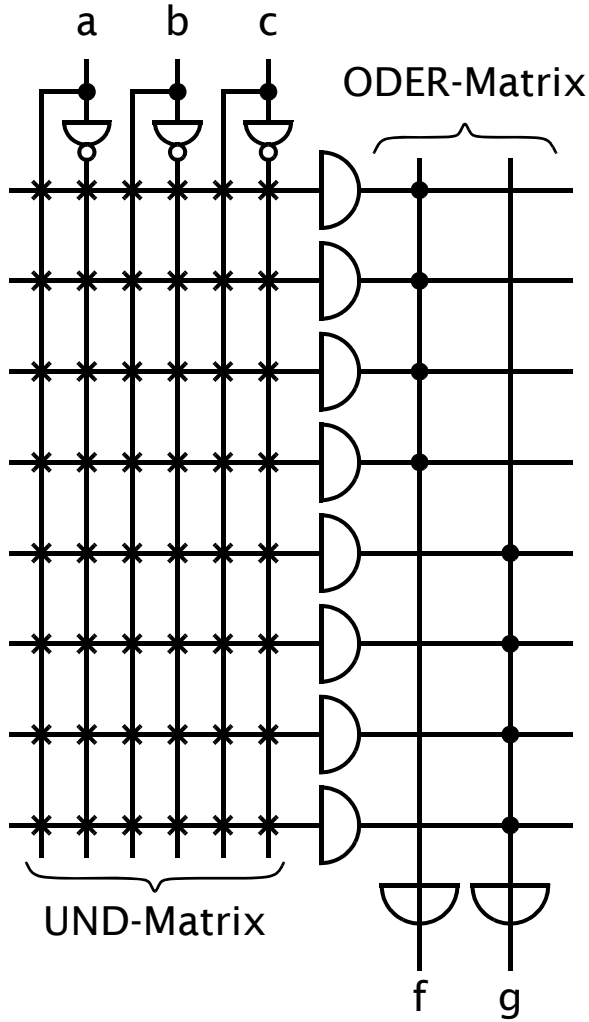
- Realisierung boolescher Funktionen mit PAL (Programmable Array Logic)

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

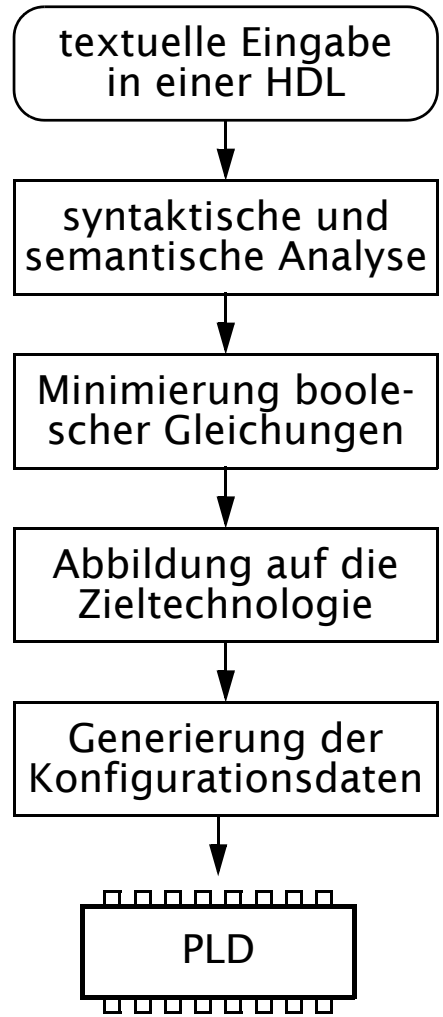
  $g(a, b, c) = c \cdot (a + b)$

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

2. Markierung programmierbarer Verbindungen in der UND-Matrix



- 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
    - TTL Transistor-Transistor-Logic
    - LSTTL Low-Power-Schottky-TTL
    - HC(T) High-Speed-CMOS
    - ECL Emitter Coupled-Logic

- 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.

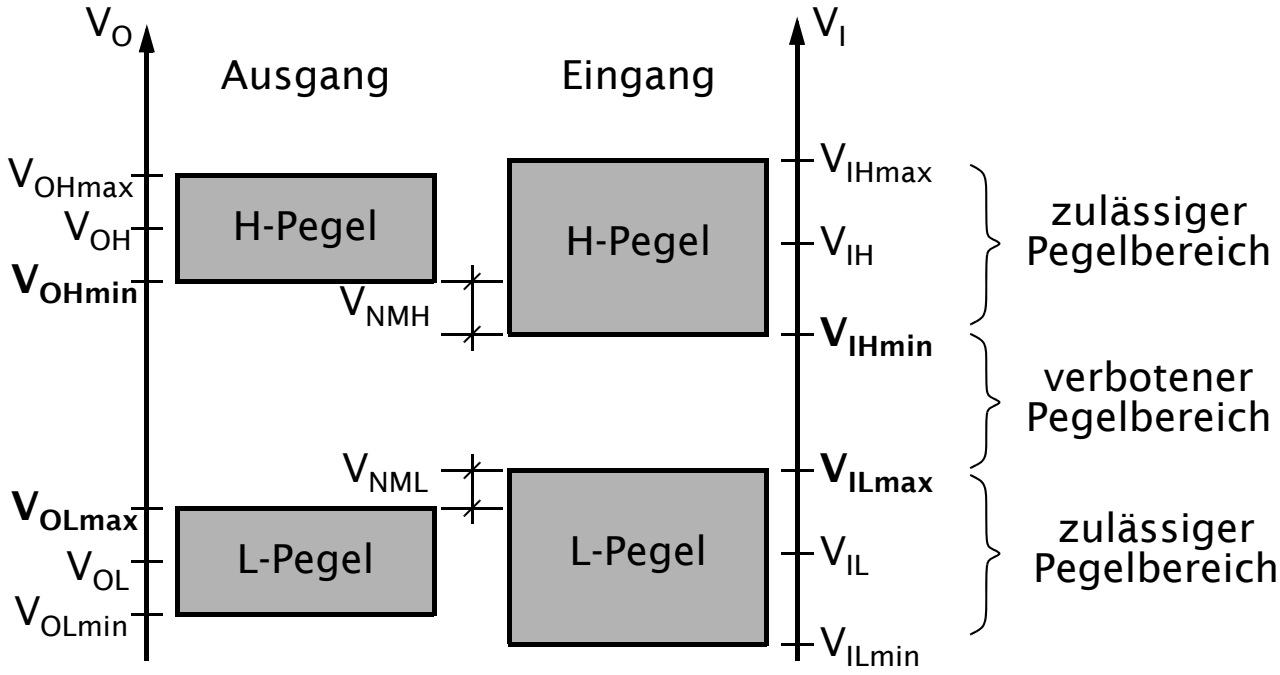
- Der Störabstand ist eine Spannung, um die eine Ausgangsspannung variieren darf, ohne daß ein angeschlossener Eingang derselben Logikfamilie in den verbotenen Pegelbereich gelangt.

- 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.

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



- Eingangsspannungen

  für die LSTTL-Baureihe mit $V_{CC}$ = +5V, $\vartheta$ = 25˚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 |

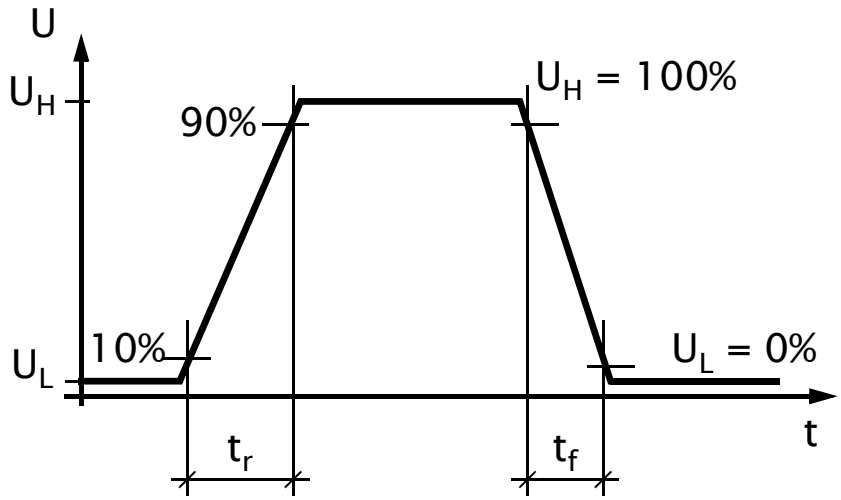
- Störabstände

| | | | |
|---|---|---|---|
| $V_{NMH}$ | 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 |

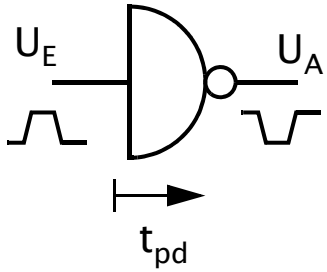
- Anstiegs- und Abfallzeiten
  - Anstiegs- und Abfallzeiten werden zwischen 10% und 90% des Absolutwert-Pegels eines Signals gemessen und angegeben.

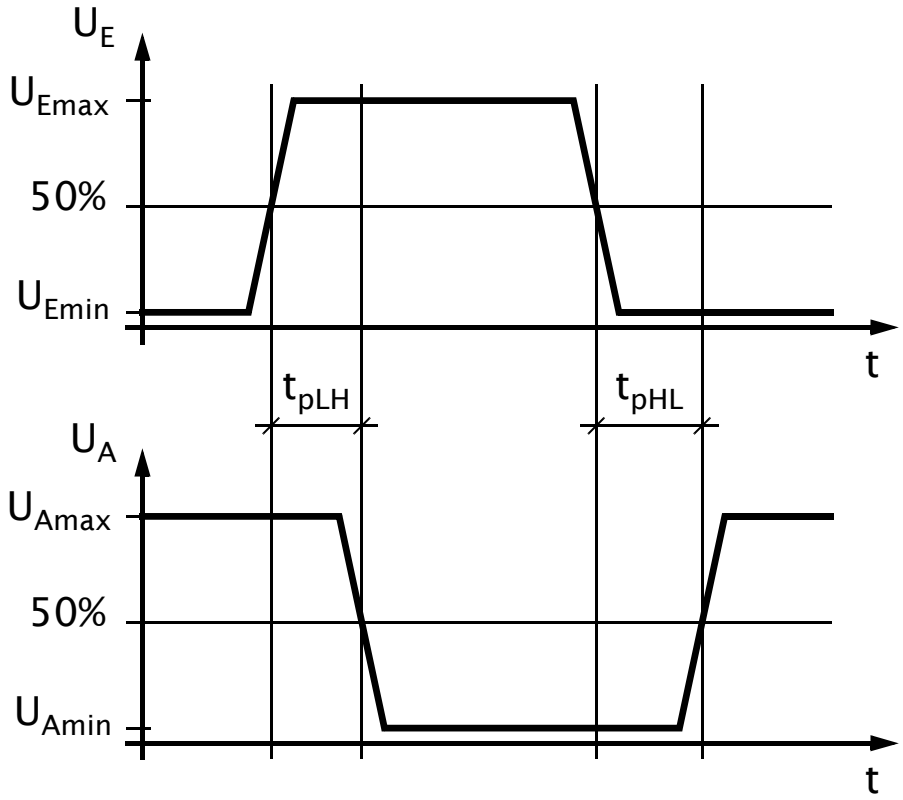
$t_r$ Anstiegszeit (rise time) einer Signalflanke ist die Zeitspanne, die ein Signal beim Übergang von L-Pegel auf H-Pegel benötigt.

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



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




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

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

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

- 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.

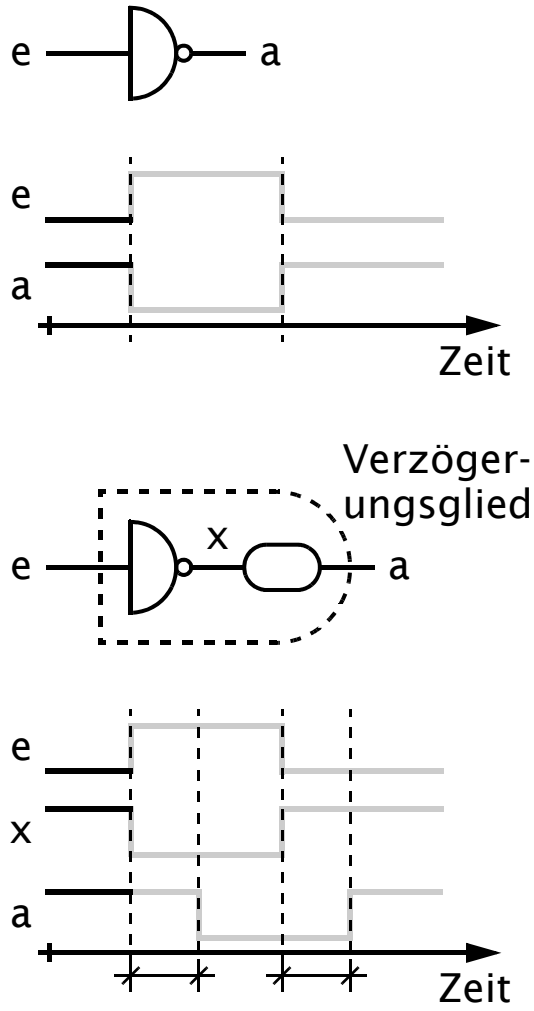
  $f_{max} = 1/t_{pd}$

- 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.

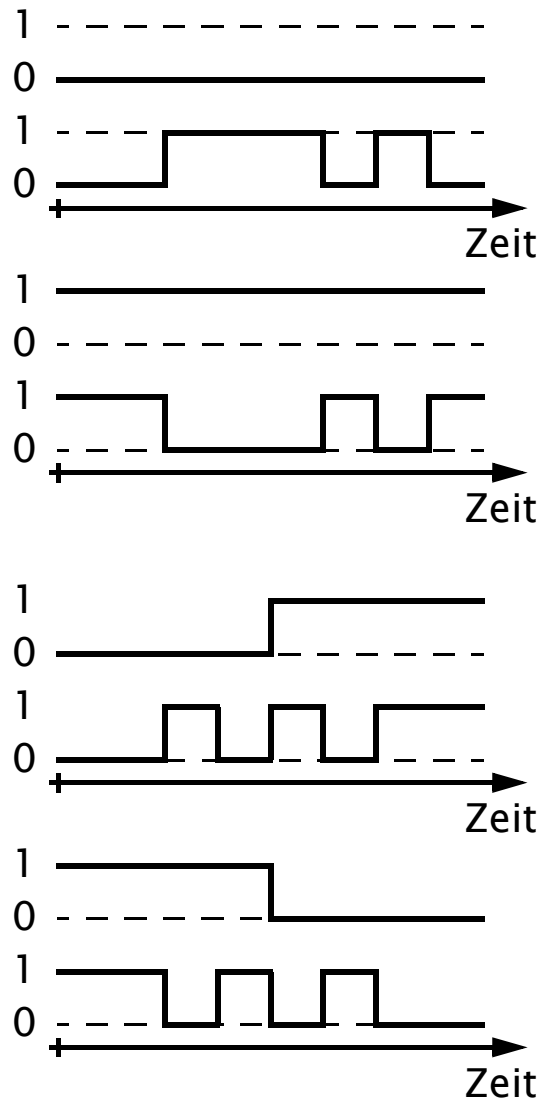
- Verzögerungsbehaftete Gatter
  - Signale werden durch Gatter unterschiedlich verzögert. Es tritt aber keine Verformung der Signale ein.
  - Ein verzögerungsfreies Gatter wird um ein Verzögerungsglied mit der Totzeit $\tau$ erweitert.
  - Das zeitliche Verhalten eines Gatters mit einem Verzögerungsglied 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
  - 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.

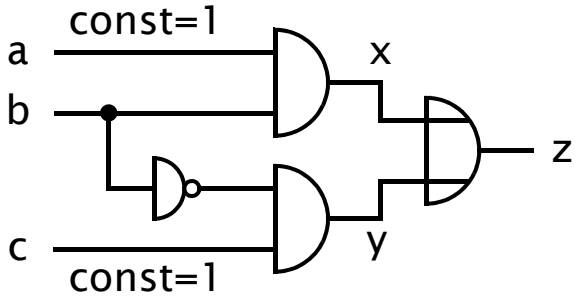
- Statischer Hazard
  - Als statischer Hazard wird ein kurzzeitiger (mehrfacher) Wechsel eines Signals bezeichnet, das eigentlich statisch 0 oder 1 hätte bleiben sollen.
    - Ein Hazard in einem statischen 0-Übergang heißt statischer 0-Hazard.
    - Ein Hazard in einem statischen 1-Übergang heißt statischer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

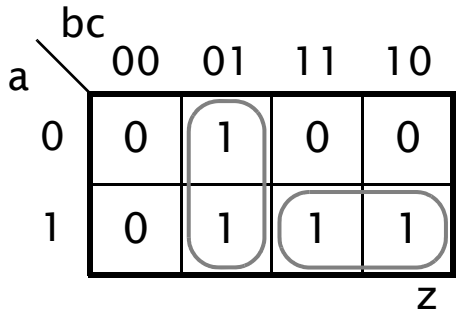


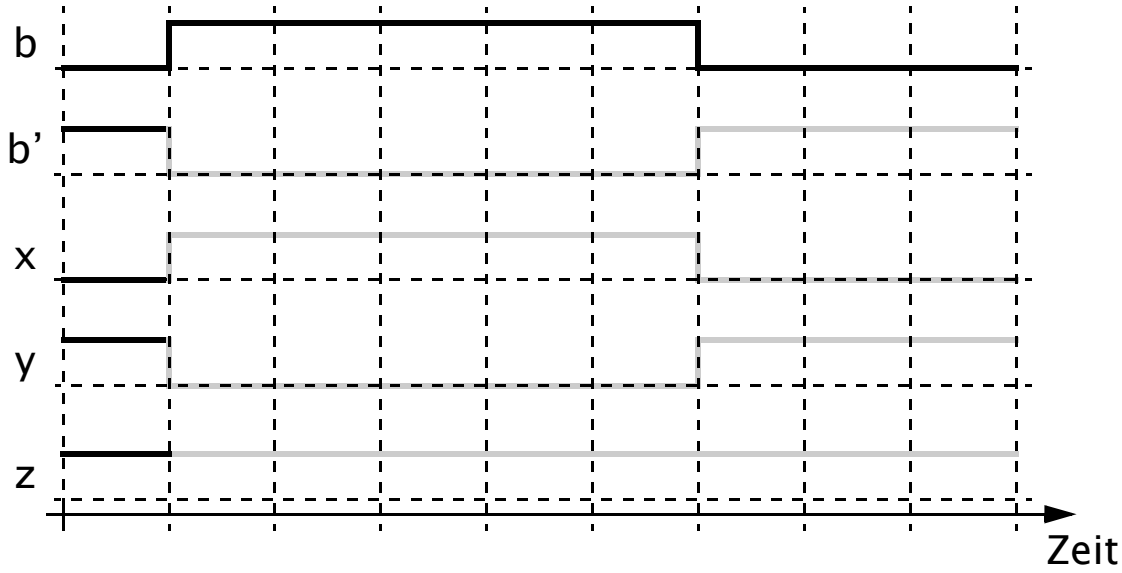
- Entstehung von Hazards

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

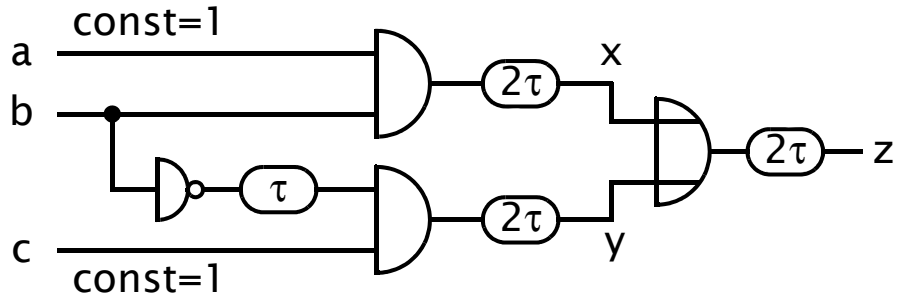
  Analyse des 0-1-Übergangs von $z(1,0,1) = 1$ auf $z(1,1,1) = 1$ mit den Variablen a=1 und c=1

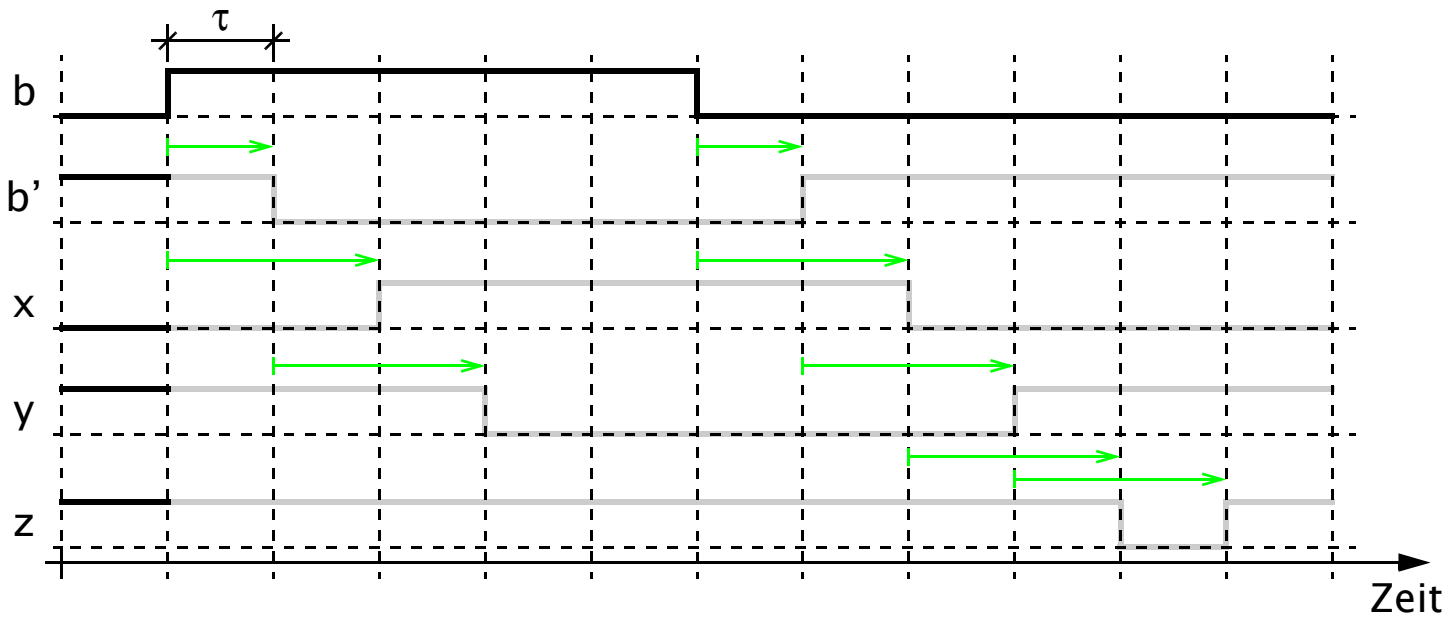


- Entstehung von Hazards

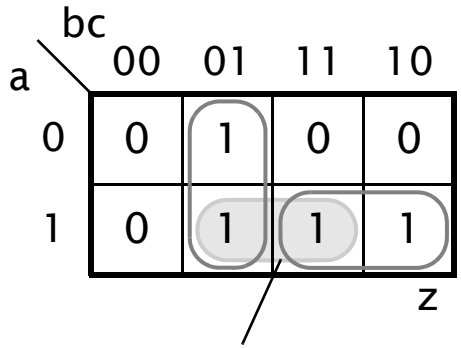


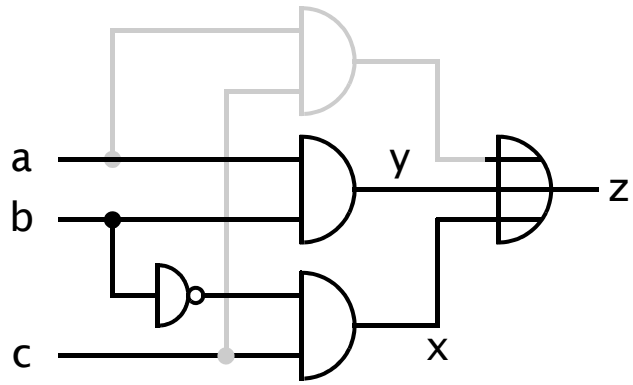



- Ein *Strukturhazard* ist ein Hazard, dessen Ursache in der Struktur des realisierten Schaltnetzes liegt.
- Ein Strukturhazard läßt sich grundsätzlich durch die Änderung der Struktur des Schaltnetzes mit sog. Anti-Hazard-Termen unter Beibehaltung der Funktion beheben.

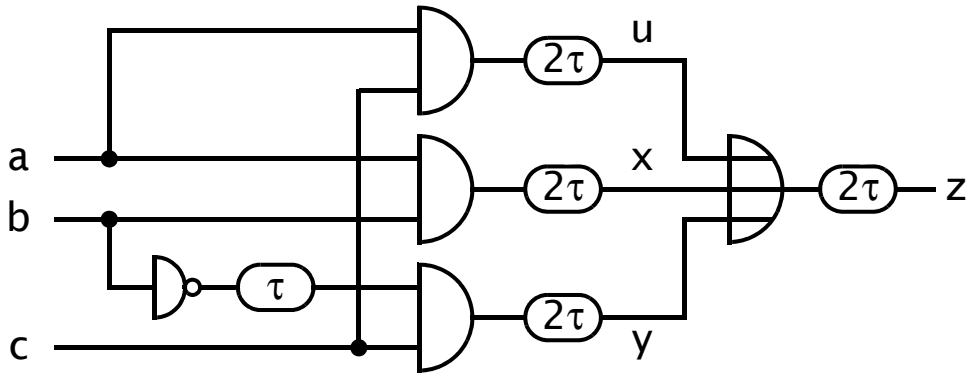
- Satz von Eichelberger:
  Ein Schaltnetz, das die Disjunktion *aller* Primimplikanten einer gegebenen Funktion realisiert, ist – unter der Voraussetzung, daß sich zu einem Zeitpunkt nur eine Eingangsvariable ändert – frei von
  - allen statischen Strukturhazards und
  - allen dynamischen Strukturhazards.

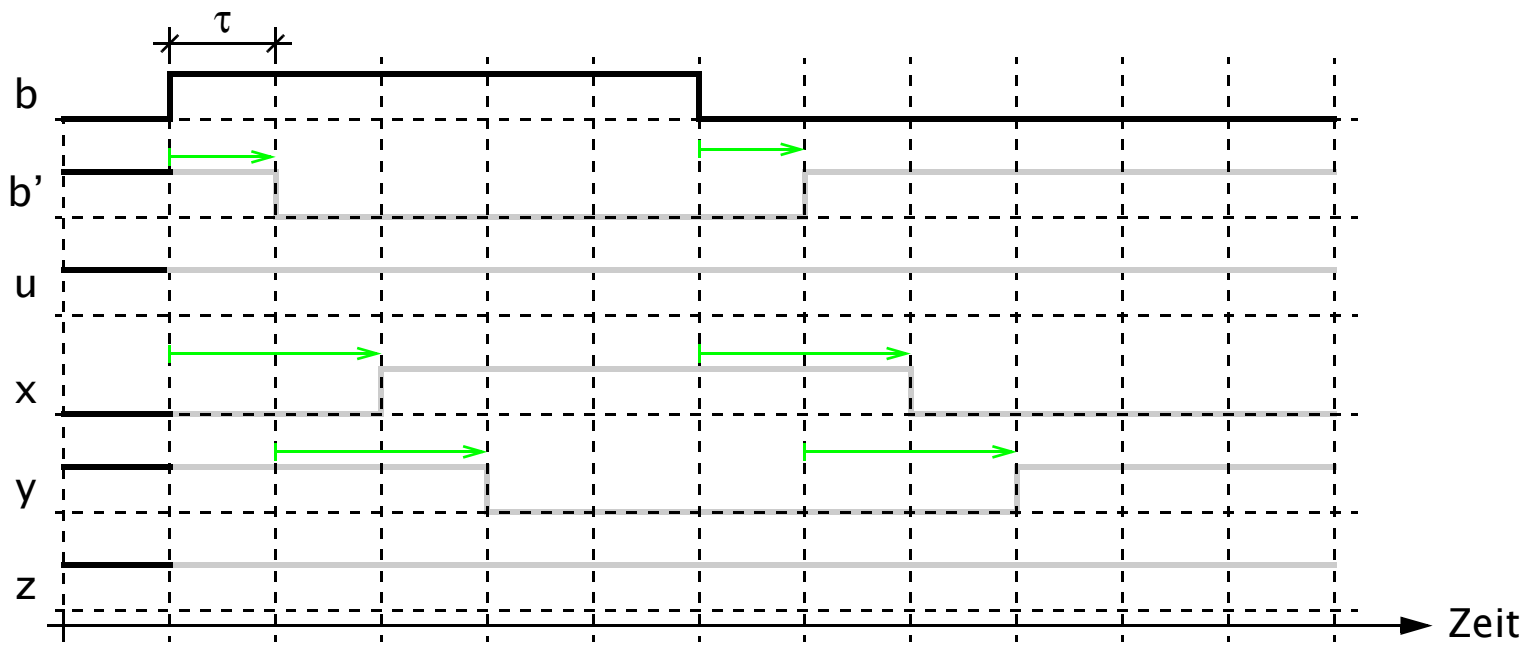
- Anti-Hazard-Gruppen
  - Die Funktion z(a, b, c) = Σ(1, 5, 6, 7) hat drei Primimplikanten b' · c, a · b und a · c
  - Die minimierte Funktion z(a, b, c) = a · b + b' · c wird um den dritten fehlenden (und redundanten) Primimplikanten a · c erweitert.
  - Solche Primimplikanten werden oft als Anti-Hazard-Gruppen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



- 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.

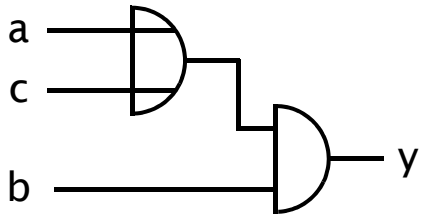
- Der Funktionshazard läßt sich verhindern, wenn man erreicht, daß sich an jedem beliebig herausgegriffenen Zeitpunkt jeweils nur eine von denjenigen Eingangsvariablen, die den Hazard verursachen, ändert.

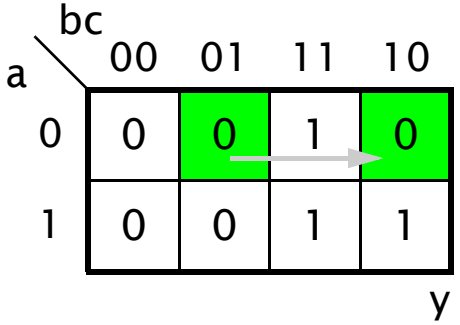
- Statischer Funktionshazard

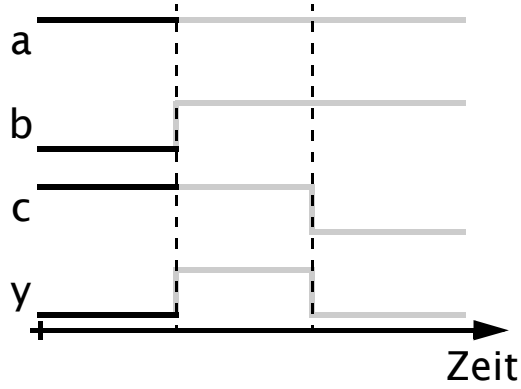
  $y(a, b, c) = \Sigma(3, 6, 7) = b \cdot (a + c)$

  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.



- 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 ⇒ 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.