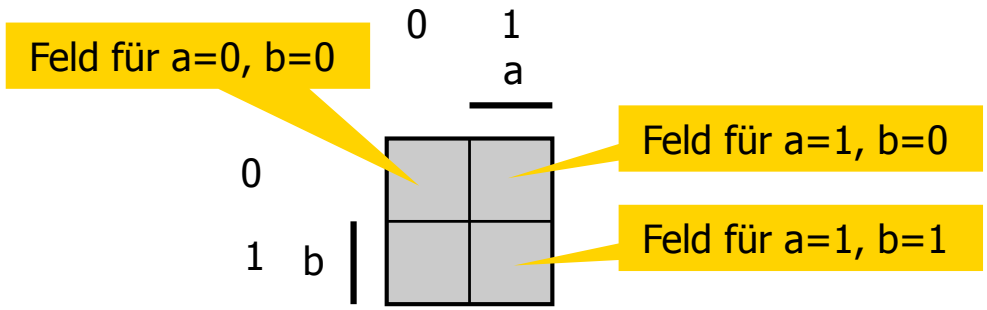
# Karnaugh - Diagramme

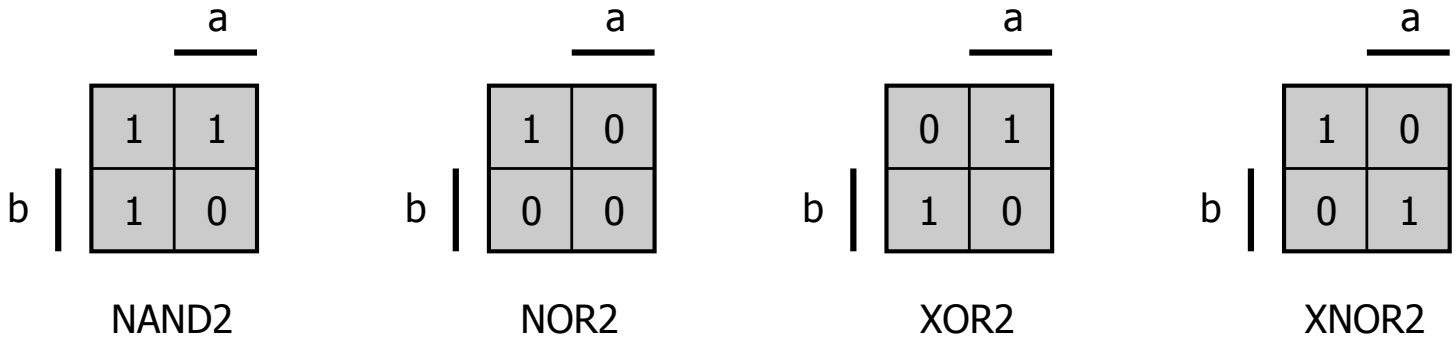
# Karnaugh - Diagramme

- Eine weitere Darstellungsform für Funktionen/Ausdrücke benutzt zweidimensionale **Tafeln**, die auch als **Karnaugh (-Veitch) - Diagramme** (**KMAPs**) bezeichnet werden
- Sie geben einen **grafischen Eindruck** der Funktion und können zur **Logikminimierung** benutzt werden

- Für eine Funktion von 2 Variablen (4 mögliche Eingangskombinationen) zeichnet man:




- Einfache Beispiele:

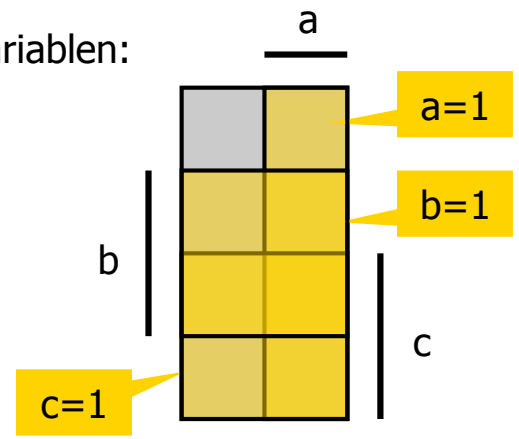


# Größere KMAPs

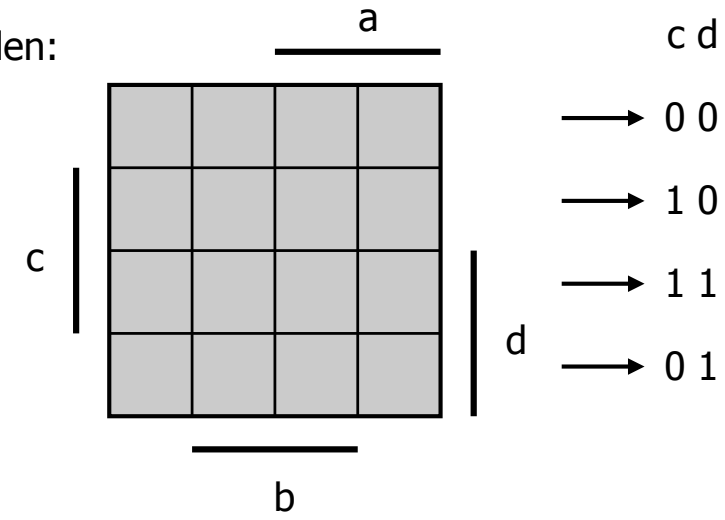
- Für Funktionen mit >2 Variablen werden die Tafeln erweitert
- Sie sind besonders nützlich, wenn für jede Variable die Einsen in den Zeilen/Spalten **einen Block** bilden.

- Dies kann nur für je 2 Variablen pro Dimension erfüllt werden. Sie sind dann Gray-codiert.
- Für >4 Variablen muß (müsste) man in die dritte Dimension gehen, ab 7 Variablen wird's schwierig....

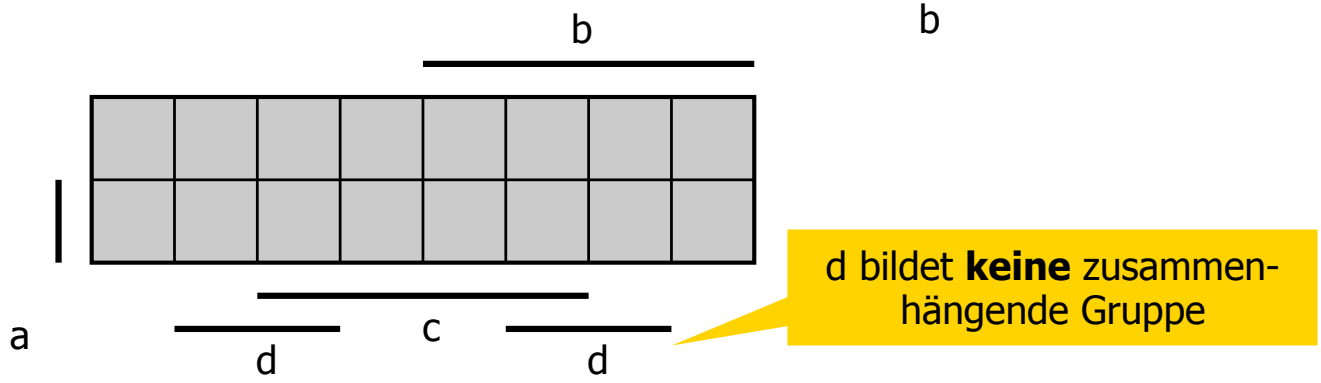
- 3 Variablen:




- 4 Variablen:



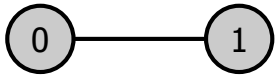

- 4 Variablen - Alternative:



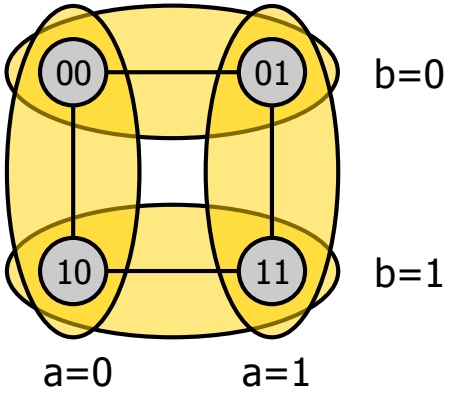
# Visualisierung der Nachbarschaft

- Die Nachbarschaft in N-stelligen Codes mit Hamming-Distanz H=1 wird in einer N-dimensionalen Darstellung besser verständlich:

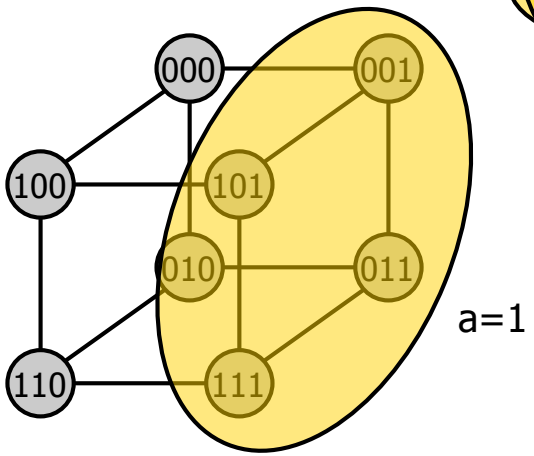
- N=1: Codewort a = 0 oder 1



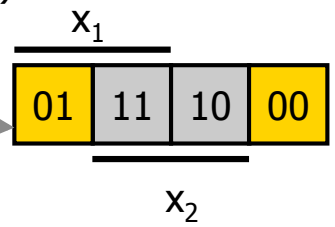

- N=2: Codewort ba = 00,01,10,11




- N=3: Codewort cba = 000,001,...

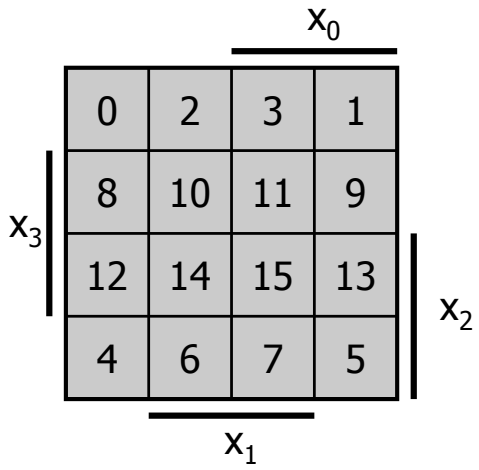



- Auf jeder Kante ändert sich genau *ein* Bit. Ein Code mit H=1 entsteht also beim 'Wandern auf Kanten'
- Auf jeder (N-1)-dimensionalen ‚Hyper'-Ebene (senkrecht zu einer der Achsen) ist ein Bit konstant
- Das 'Aufklappen' des N-dimensionalen Würfels ergibt die KMAP.
- **Gegenüberliegende Randfelder sind benachbart**!
- Die KMAP hat so viele Nachbarschaften, wie der Würfel Kanten hat.

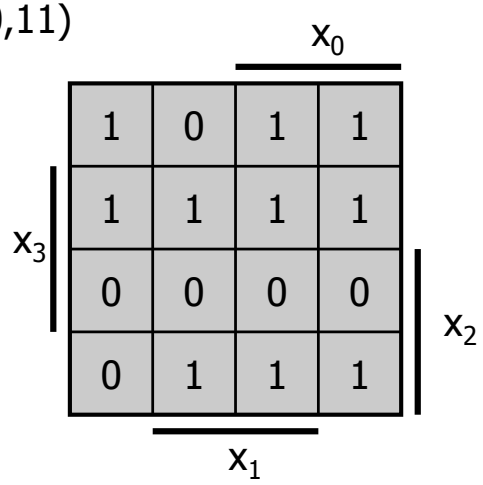


# Nummerierung

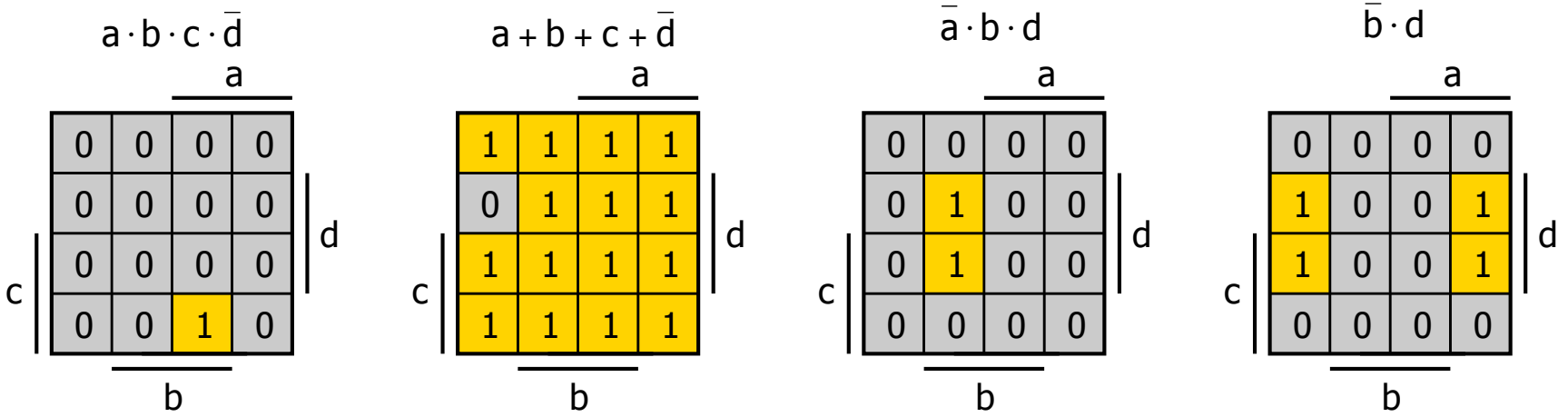
- Die Felder können nummeriert werden, wenn man die Eingangsvariable (hier $x_3x_2x_1x_0$) als Binärzahl auffaßt




- Eine beliebige Funktion der N Variablen wird durch das 0/1 Muster in der Tafel festgelegt. (Für N=4 gibt es $2^{2^4} = 2^{16} = 65536$ verschiedene Funktionen)
- In einer kompakten Schreibweise führt man die Eingangswerte auf, für die in der Tafel eine 1 steht und schreibt z.B. $F = \Sigma(0,1,3,5,6,7,8,9,10,11)$



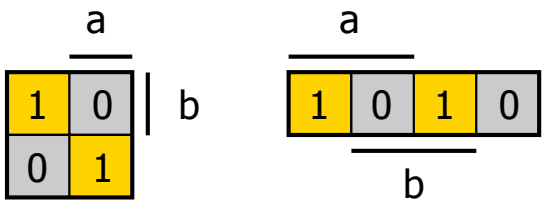
# UND und ODER in K-MAPS



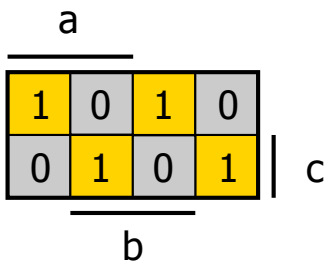

- Die UND-Verknüpfung aller **N Variablen** (direkt oder invertiert) selektiert **EIN Feld**. Man nennt diese 16 möglichen Ausdrücke **MINTERME**, weil sie in der Tafel die minimale Fläche einnehmen.
- Die ODER-Verknüpfung der N Variablen selektiert **ALLE AUSSER EINEM Feld**. Man nennt einen solchen Ausdruck **MAXTERM**, weil er in der Tafel die maximale Fläche einnimmt.
- Die UND-Verknüpfung von **N-1 Variablen** bilden einen **1x2 oder 2x1 Block**. (Dies gilt nur, wenn die Variablen Gray-codiert sind!). Im Beispiel oben ist f = !a·b·c·d + !a·b·!c·d = !a·b·d·(c+!c) = !a·b·d.
- Die UND-Verknüpfung von **N-2 Variablen** bilden einen **Block aus 4 Zellen** (1x4, 4x1 oder 2x2), etc...
- Blöcke können auch periodisch über den Rand hinaus gehen (s. rechtes Beispiel)!
- Entsprechendes gilt für das ODER von weniger als N Variablen

# EXOR in K-MAPS

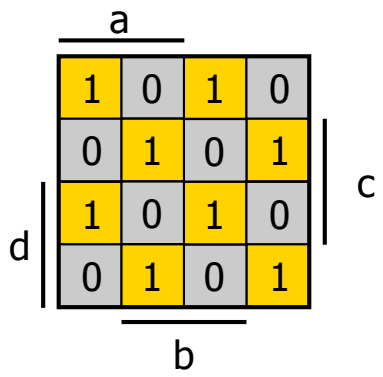
$a \oplus b = a \cdot !b + !a \cdot b$




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




$a \oplus b \oplus c \oplus d$




- EXOR und XNOR sind durch viele diagonale Einzelfelder gekennzeichnet.
- (NB: $a \oplus b \oplus c \oplus d$ ist 1 bei 1000, 0100, 0010 und 0001 aber **auch** bei 1011,…)
- Sie können daher nicht vereinfacht werden (s. später)

# Kanonische Formen

Die **Disjunktive Normalform** (DN) eines Ausdrucks hat die Form (Disjunktion = ODER!)

$$K_0 + K_1 + K_2 + \ldots$$

wobei die $K_i$ Konjunktionen (UND-Verknüpfungen) aus einfachen oder negierten Variablen ('Literalen') sind.

Man nennt die $K_i$ auch **Produktterme** oder Einsterme

Entsprechend hat die **Konjunktive Normalform** (KN) (das Duale Pendant) die Form

$$D_0 \cdot D_1 \cdot D_2 \cdot \ldots$$

Mit den Disjunktionen $D_i$, die man auch Nullterme nennt.

- Diese **Kanonischen Formen** sind z.B. wichtig, weil in PALs die entsprechende Struktur in Hardware implementiert wird.
- Die Formen werden **eindeutig**, wenn man verlangt, daß in jedem Eins/Nullterm **jedes Literal genau einmal** vorkommt (**Minterm/Maxterm**!).
  Man spricht dann von **ausgezeichneten Normalformen**.
  Beispiel: $y = a \cdot b \cdot !c + !a \cdot c = a \cdot b \cdot !c + !a \cdot c \cdot (b+!b) = a \cdot b \cdot !c + !a \cdot b \cdot c + !a \cdot !b \cdot c$

Die **ausgezeichnete Disjunktive Normalform** ist die einfachste **Summe von Mintermen**.

Die **ausgezeichnete Konjunktive Normalform** ist das einfachste **Produkt von Maxtermen**.

- Welcher Ausdruck (KN/DN) einfacher ist (weniger Terme hat), hängt von der Funktion ab

# Beispiel Konjunktive/Disjunktive Normalform

- Beispiel DN: $y = a \cdot b \cdot !c + !a \cdot c$
- Umwandlung zur KN durch Aus-ODER-n: $y = (a+!a) \cdot (b+!a) \cdot (!c+!a) \cdot (a+c) \cdot (b+c) \cdot (!c+c)$
  $= (b+!a) \cdot (!c+!a) \cdot (a+c) \cdot (b+c)$
- Test:

| a | b | c | ab!c | !ac | y | b+!a | !c+!a | a+c | b+c | y |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | **0** | 1 | 1 | 0 | 0 | **0** |
| 0 | 0 | 1 | 0 | 1 | **1** | 1 | 1 | 1 | 1 | **1** |
| 0 | 1 | 0 | 0 | 0 | **0** | 1 | 1 | 0 | 1 | **0** |
| 0 | 1 | 1 | 0 | 1 | **1** | 1 | 1 | 1 | 1 | **1** |
| 1 | 0 | 0 | 0 | 0 | **0** | 0 | 1 | 1 | 0 | **0** |
| 1 | 0 | 1 | 0 | 0 | **0** | 0 | 0 | 1 | 1 | **0** |
| 1 | 1 | 0 | 1 | 0 | **1** | 1 | 1 | 1 | 1 | **1** |
| 1 | 1 | 1 | 0 | 0 | **0** | 1 | 0 | 1 | 1 | **0** |

# Auffinden des Komplements einer Funktion

- Gegeben sei die Funktion $F = a \cdot c + !b \cdot !c.$
  Sie hat z.B. die KMAP

F:

| c \ a | 1 | 0 | 0 | 1 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | |

b

- Die KMAP der inversen Funktion !F findet man direkt durch Vertauschen von 0 und 1:

!F:

| c \ a | 0 | 1 | 1 | 0 |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | |

b

- Man sieht sofort, daß diese die Darstellung $!F = !a \cdot c + b \cdot !c$ hat.
- Mit den Regeln der Schaltalgebra ist das deutlich umständlicher:

$$\bar{F} = \overline{a \cdot c + \bar{b} \cdot \bar{c}} = \overline{a \cdot c} \cdot \overline{\bar{b} \cdot \bar{c}} = (\bar{a} + \bar{c}) \cdot (b + c)$$

$$= \bar{a}b + \bar{a}c + \bar{c}b + \bar{c}c = \bar{a}b + \bar{a}c + \bar{c}b$$

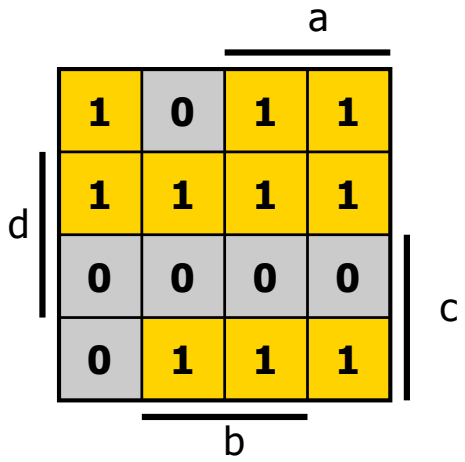
überflüssiger Term

$$= \bar{a}b(c + \bar{c}) + \bar{a}c + \bar{c}b = \bar{a}bc + \bar{a}b\bar{c} + \bar{a}c + \bar{c}b$$

$$= \bar{a}bc + \bar{a}c + \bar{a}b\bar{c} + b\bar{c} = \bar{a}(bc + c) + (\bar{a}b + b)\bar{c} = \bar{a}c + \bar{c}b$$

# Auffinden des einfachsten Ausdruck für eine Funktion




$$F = a \cdot \bar{c} + \bar{c} \cdot d + \bar{b} \cdot \bar{c} + a \cdot \bar{d} + b \cdot c \cdot \bar{d}$$

Term ist redundant

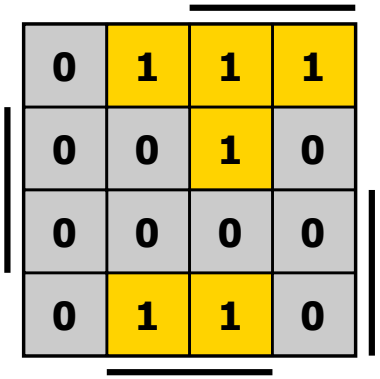
- Ist eine Funktion über ein Muster in der KMAP gegeben, so sucht man **eine vollständige Überdeckung** der Einsen mit **möglichst großen** rechteckigen Blöcken...
- Die Funktion ist dann die ODER-Verknüpfung all dieser sogenannten '**Primterme**' (Der Block eines Primterms kann nicht vergrößert werden)

- Es kann sein, daß eine Teilmenge der Primterme zur vollständigen Überdeckung ausreicht. Mit einer Tabelle kann man systematisch nach der kleinsten Teilmenge suchen.
- Im Beispiel sieht man, daß der erste Term **a·!c** bereits durch den zweiten (**!c·d**) und vierten (**a·!d**) abgedeckt wird. (Dies kann man beweisen, indem man **!c·d** + **a·!d** = **a·!c** + !a·!c·d + a·c·!d zeigt...)

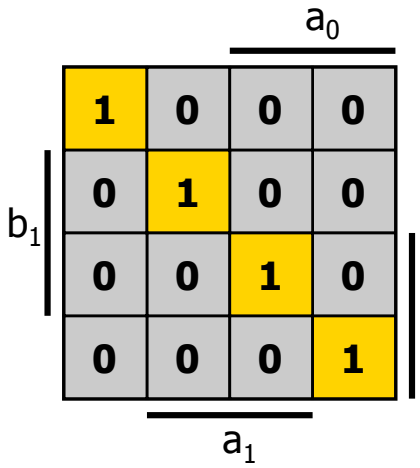
- Das Ergebnis kann man auch (aufwändiger) durch Umformung von
  F = Σ(0,1,3,5,6,7,8,9,10,11) = !a·!b·!c·!d + !a·!b·!c·d + !a·!b·c·d + ... finden.
- N.B.: Einen Ausdruck in UND-Form (a+b)·(!c+d)·... Findet man durch Abdecken der Nullen in der KMAP

# Beispiel: 2 + 2 Bit Vergleicher

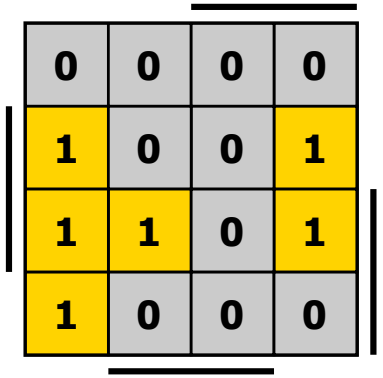
| $b_1$ | $b_0$ | $a_1$ | $a_0$ | a=b | a>b | a<b |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | | 0 | 1 | 0 | 1 | 0 |
| | | 1 | 0 | 0 | 1 | 0 |
| | | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| | | 0 | 1 | 1 | 0 | 0 |
| | | 1 | 0 | 0 | 1 | 0 |
| | | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | 0 | 1 | 0 | 0 | 1 |
| | | 1 | 0 | 1 | 0 | 0 |
| | | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| | | 0 | 1 | 0 | 0 | 1 |
| | | 1 | 0 | 0 | 0 | 1 |
| | | 1 | 1 | 1 | 0 | 0 |



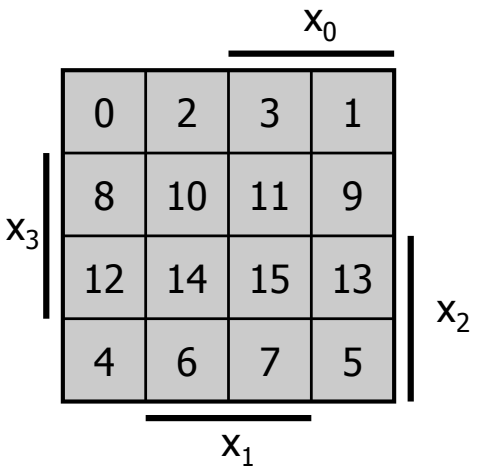

a>b




a=b




a<b

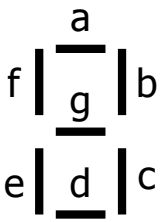


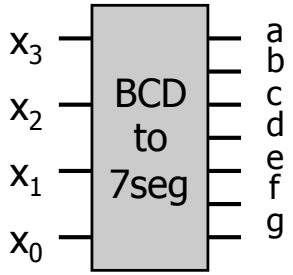

- Man erkennt sofort, dass a=b 'schwieriger' ist als die anderen beiden – man benötigt offenbar EXORs…

# KMAPs mit leeren Feldern (don't care)

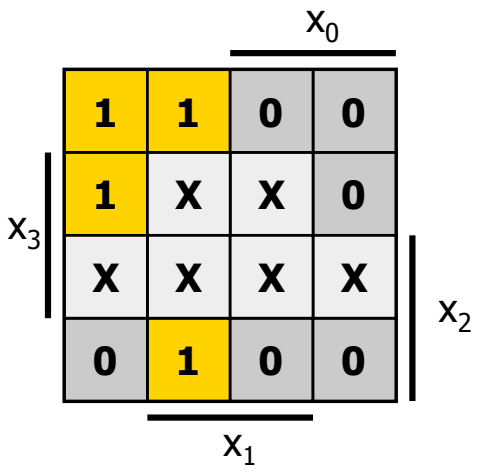
- Es kommt oft vor, daß nicht alle Eingangskombinationen vorkommen können. Dadurch bleiben Felder in der KMAP 'leer', sie werden meist mit 'x' markiert.
- Diese Felder können so mit 1 oder 0 belegt werden, daß die Funktion möglichst einfach wird, i.e. daß die Einsen mit möglichst wenigen, möglichst großen Blöcken überdeckt werden können.

- Beispiel: Ansteuerung einer Siebensegment-Anzeige mit eine BCD-Zahl:

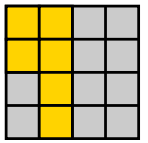



- Man schreibt z.B. für **Segment e**:
  **e=Σm(0,2,6,8) + Σd(10,11,12,13,14,15)**.
  Dabei steht 'm' für Minterm und 'd' für don't care




- Abdeckung nur der Einsen erfordert 2 Felder a 1x2, d.h zwei Primterme mit 3 Termen:
- $e = !x_0 \cdot !x_1 \cdot !x_2 + !x_0 \cdot x_1 \cdot !x_3$
- Ersetzt man zwei der don't care Felder durch Einsen, so kann man die Terme vereinfachen:
- $e = !x_0 \cdot !x_2 + !x_0 \cdot x_1$



- Je mehr don't cares es gibt, desto stärkere Vereinfachungen sind möglich. Man sollte die Funktion also nicht unnötig einschränken.

# Minimierung nach Quine-McCluskey

- Für mehr als 4 Eingangsvariable wird das grafische Verfahren der KMAPs unübersichtlich.
- Ein Algorithmus zum Auffinden der einfachsten Summendarstellung mit Primtermen ist das Quine-McCluskey Verfahren.
- Es unterteilt sich in 2 Schritte:
  - Auffinden aller Primterme
  - Bestimmung der minimalen Überdeckung (Verwerfen von redundanten Primtermen)

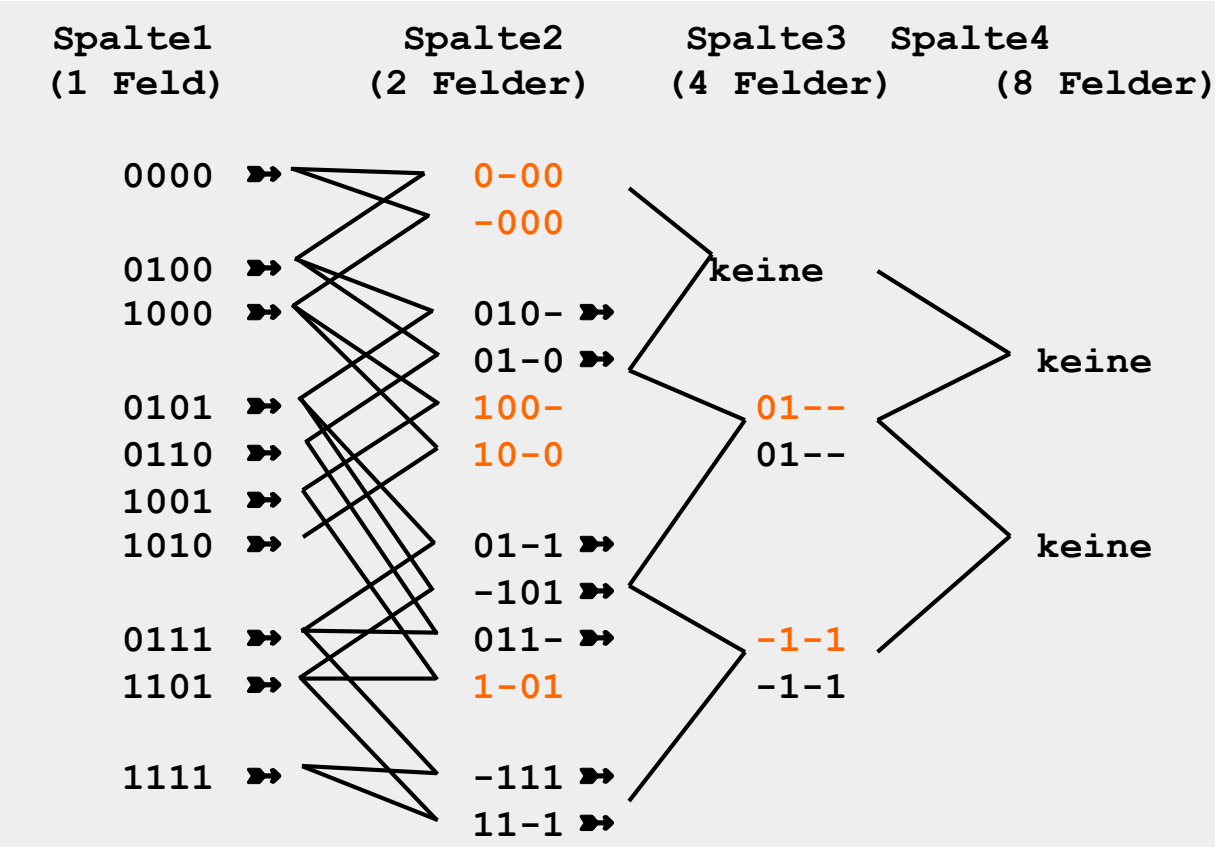
**Auffinden aller Primterme:**

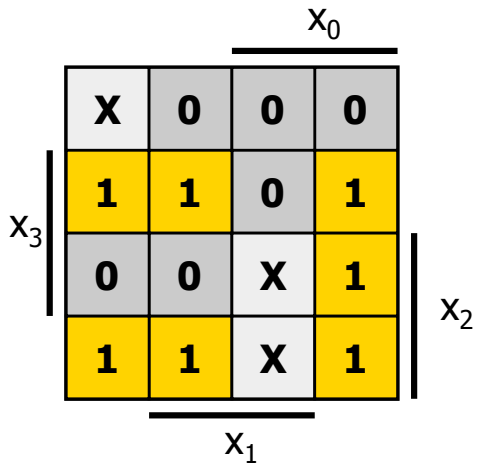
- 1. Schritt:
  - Schreibe die Binärdarstellung aller Einsen in die erste Spalte einer Tabelle.
    **Don't cares** werden zunächst **als Einsen** gewertet!
  - Dabei werden Gruppen von Termen mit keiner Eins, einer 1, zwei Einsen etc. gebildet.
    (Benachbarte Gruppen enthalten also benachbarte Felder in der KMAP)
- 2. Schritt:
  - Für jeweils aufeinanderfolgenden Gruppenpaare werden alle Elemente paarweise verglichen. ($\Rightarrow$ exp. Aufwand!)
  - Wenn ein Elementpaar sich nur in einer Stelle unterscheidet, existiert ein Term, der diese Stelle nicht enthält.
  - Alle solche Terme werden Gruppenweise in der Form 10-0 etc. in die nächste Spalte notiert.
  - Die Terme in der ersten Spalte, die eine Nachbarschaft aufweisen, werden mit einem Haken als nicht-prim markiert.

  Der 2. Schritt wird spaltenweise wiederholt, bis sich keine Paare mehr finden. Die '-' Zeichen müssen dabei beim Vergleich an der gleichen Stelle sitzen.
- 3. Schritt:
  - Die Terme ohne Haken sind die Primterme.
  - In ihrer Menge werden mit einer Tabelle die zur Überdeckung *notwendigen* Primterme gesucht.

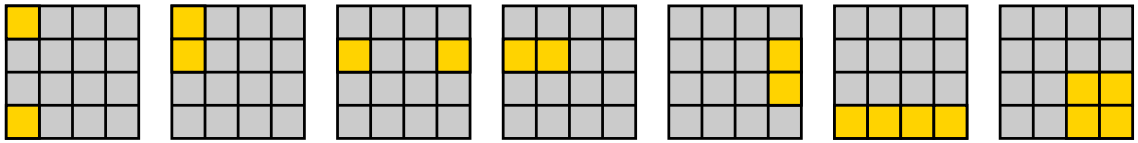
# Quine – McCluskey Tabelle

- Beispiel F = Σm(4,5,6,8,9,10,13)+Σd(0,7,15)



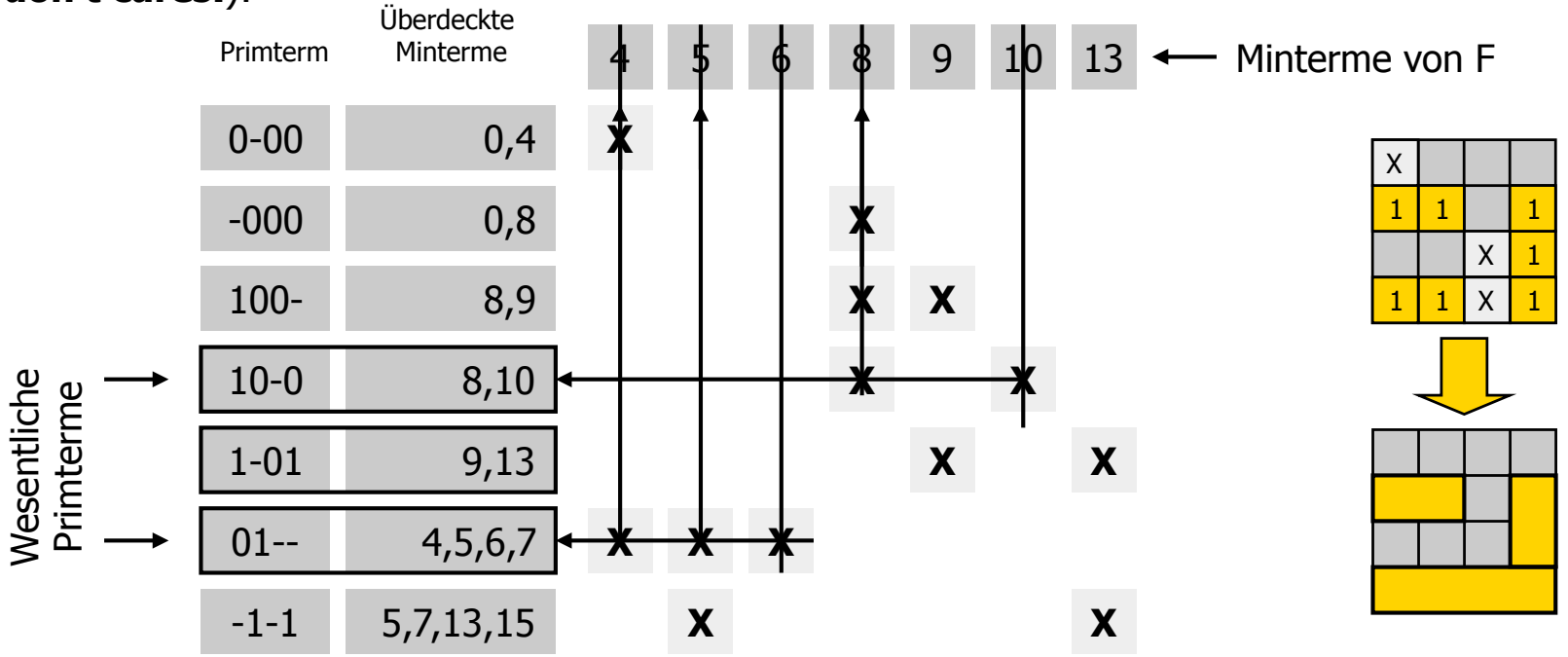




- Ergebnis: Die 7 Primterme sind 0-00, -000, 100-, 10-0, 1-01, 01--, -1-1



# Auffinden der Minimalen Überdeckung

- Für F = Σm(4,5,6,8,9,10,13)+Σd(0,7,15) sind die Primterme: 0-00, -000, 100-, 10-0, 1-01, 01--, -1-1
- Trage in einer Tabelle ein, welche Primterme welche Einsen der Funktion (d.h. Minterme) abdecken (**keine don't cares!**):




- Die **Wesentlichen Primterme** sind diejenigen, die unbedingt notwendig sind, weil ein Minterm nur von ihnen überdeckt wird.
  Sie werden aufgesucht (hier für 6 und 10) und alle Minterme ausgestrichen, die damit abgedeckt sind.
- Die restlichen Minterme werden mit ‚möglichst wenigen' **redundanten** Primtermen überdeckt.
  Im Beispiel reicht 1-01.
- Ergebnis: $F = x_3 \cdot !x_2 \cdot !x_0 + x_3 \cdot !x_1 \cdot x_0 + !x_3 \cdot x_2$
- In diesem Beispiel wurde der ‚don't-care' Term 7 auf Eins, die anderen auf Null gesetzt.

# Wie wichtig sind die kanonischen Formen / KMAPs?

Vorteile:

- Nützlich für eine Implementierung in PALs (s. später)
- Von theoretischen Interesse, da eindeutige Darstellung
- KMAPs geben sofort einen visuellen Eindruck der ‚Komplexität' eines Ausdrucks

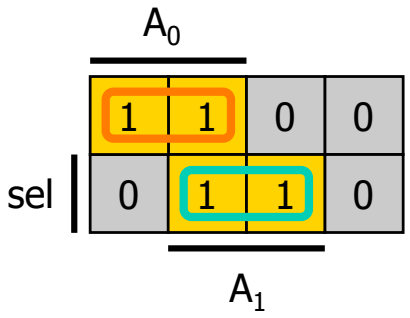
Nachteile:

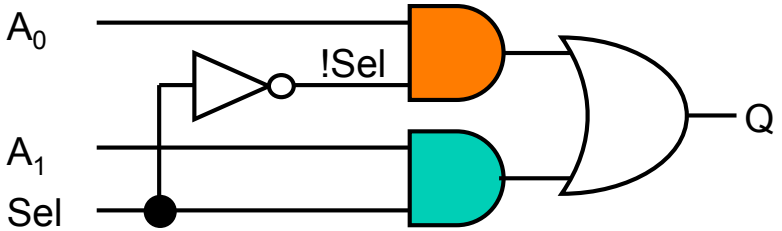
- KMAPs sind nur für max. 4 Variablen nützlich – in der Praxis oft zu wenig
- Für FPGAs sind kanonische Formen weniger geeignet, da die Hardware anders aussieht
- Minimale kanonische Formen ergeben manchmal Glitches (s. nächste Seite)
- EXORs werden nicht direkt erkannt
- Ebenso werden keine (oft sehr effiziente!) gemischten Gatter implementiert

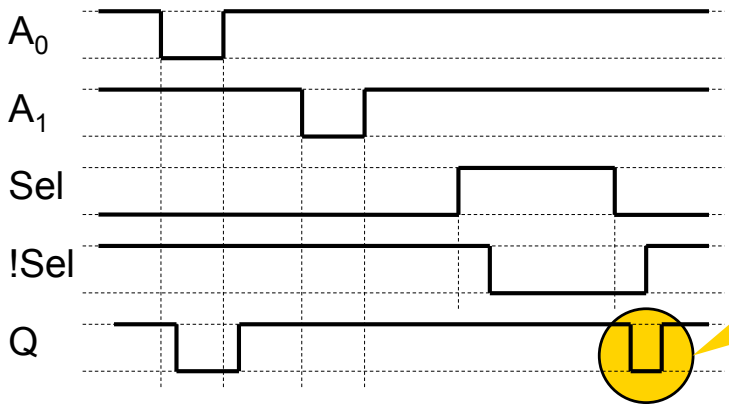
# Nachteil der minimalen Implementation: Glitches

- Klassisches Beispiel: Multiplexer

| sel | $A_1$ | $A_0$ | Q |
|---|---|---|---|
| 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 |



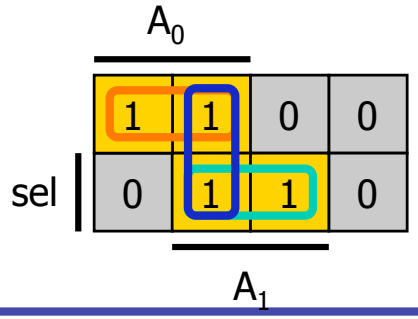



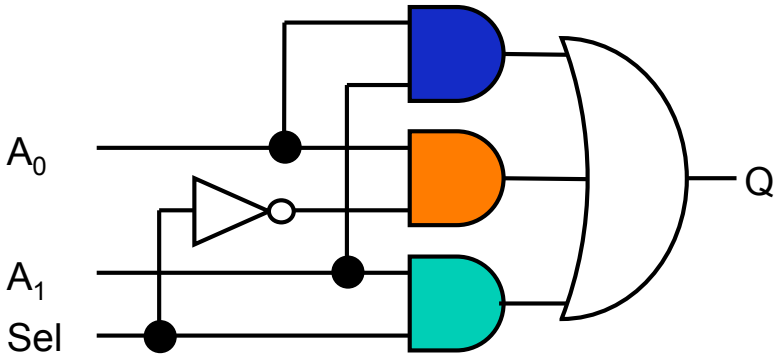




Realer Signalverlauf mit Verzögerungen

Kurzes Signal (Glitch, Race, Spike) beim Schalten von 1→1 (!). Es entsteht durch die Laufzeit im Inverter

Lösung: redundanten Term $A_0A_1$ hinzunehmen



# Schlußbemerkungen

- Die vorgestellten Verfahren führen auf Kanonische Darstellungen, die z.B. für PALs interessant sind.
- Aus verschiedenen Gründen (z.B. wegen Glitches) nimmt man manchmal nicht die minimale Form.
- Für FPGAs, Standardzellen und Full-Custom Designs versucht man, die angebotenen Ressourcen (z.B. auch gemischte Gatter) möglichst optimal zu nutzen.
  Man nennt diesen Vorgang **'Technology Mapping'**
- Es handelt sich um komplizierte **Optimierungsverfahren**, die die logische Funktion mit den zur Verfügung stehenden Ressourcen implementieren und dabei z.B. die Chipfläche oder die Verzögerung minimieren.
- Will man aus N Eingangsvariablen M Ausgangsvariable erzeugen, so kann man im Prinzip für jede der M Ausgänge das Verfahren wiederholen.
  Man kann jedoch viele Ressourcen sparen, wenn man **Zwischenterme** bildet und diese weiterbenutzt.
  Diese ‚Multiple Output Minimization' (**MOM**) ist in der Praxis sehr wichtig.
  Triviales Beispiel: N=6, M=2, y1 = abcde, y2 = abcdf: