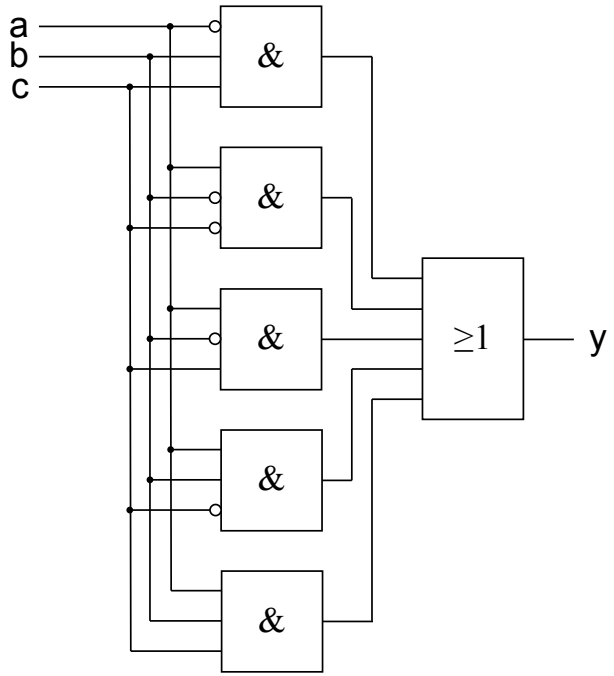
9. Optimierung von Schaltnetzen

Schaltnetz ohne Optimierung

Betrachtung der Umsetzung folgender Schaltfunktion:

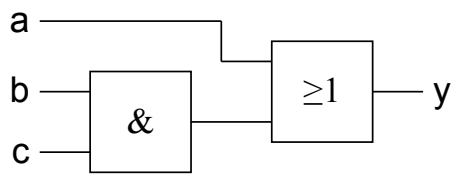
y = (a ‘ * b * c) + (a * b ‘ * c ‘) + (a * b ‘ * c) + (a * b * c ‘) + (a * b * c)



Schaltnetz mit Optimierung

Betrachtung der Umsetzung folgender Schaltfunktion:

$y = (a' * b * c) + (a * b' * c') + (a * b' * c) + (a * b * c') + (a * b * c)$



Gesucht: „einfachstes“ Schaltnetz zu Schaltfunktion! → Optimierung

Definition 9.1 (Optimales Schaltnetz)
Sei z(.) eine Funktion (sog. *Zielfunktion*), die die Güte eines Schaltnetzes bewertet.
Ein Schaltnetz SN^{opt}, das eine Schaltfunktion f implementiert, ist *optimal*, wenn es kein anderes Schaltnetz SN gibt, das f implementiert sodass $z(SN) < z(SN^{opt})$ gilt.

Optimierungsziele

Ziele der Optimierung (Zielfunktion z):

- Anzahl der benötigten Gatter
  Eine höhere Anzahl an Gattern verbraucht mehr Chipfläche

- Anzahl der Gatterebenen (logic level)
  Mehr Gatterebenen bedeutet eine höhere Verzögerungszeit → niedrigere Taktfrequenz der Schaltung

- Anzahl der Eingänge der Gatter
  Gatter mit mehr Eingängen benötigen größere Fläche
  Gatter mit einer hohen Anzahl an Eingängen sind technisch nicht einfach zu realisieren

Diese Ziele werden in dieser Vorlesung verfolgt

Optimierungsziele stehen häufig im Widerspruch zueinander und können nicht gleichzeitig erreicht werden. Z.B. sind mit mehr Gatterebenen auf Kosten höherer Verzögerungszeit oft Einsparungen von Gattern möglich.

Optimierungsziele

Weitere mögliche Optimierungsziele:

- Geringer Energieverbrauch, z.B. durch
  - Vermeidung von Glitches
  - Verwendung von Kodierungen, die zu verminderter Schaltaktivität (Schaltfrequenz) führen

- Hohe Zuverlässigkeit, z.B. durch
  - redundante Gatter, die Fehlfunktionen ausgleichen können
  - fehlerkorrigierende Kodierung

- Sicherheit (Security)
  - Äußerlich messbare Eigenschaften (Energieverbrauch, Berechnungsdauer) sollen sich für unterschiedliche Funktionen möglichst wenig unterscheiden, um z.B. Rückschlüsse auf den bei einer kryptographischen Operationen verwendeten Schlüssel zu erschweren.

Diese Ziele werden in dieser Vorlesung **nicht** vertieft.

Optimierungsansätze

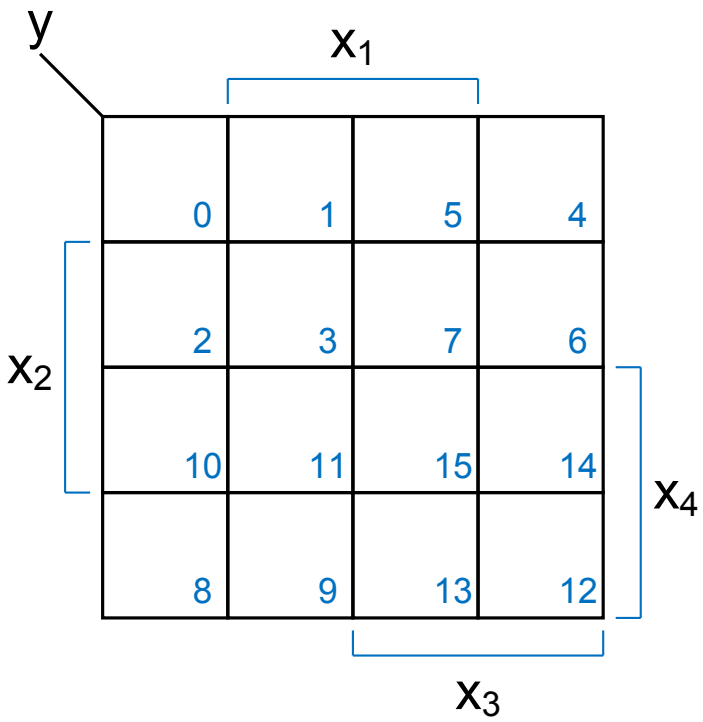
Klassifizierung von Optimierungsansätzen:

- Zweistufige Optimierung vs. mehrstufige Optimierung
- Exakte Optimierung vs. heuristische Optimierung
- Grafische, Tabellenbasierte und symbolische (formelbasierte) Verfahren
- Verfahren auf Basis von DNF/KNF vs. andere Operatorensysteme
- Technologieunabhängige Optimierung vs. technologieabhängige Optimierung

KV-Diagramme

Das Karnaugh-Veitch-Diagramm ist eine spezielle grafische Darstellungsform von Schaltfunktionen. Es wird zur grafischen Minimierung von Schaltfunktionen eingesetzt.

$y = f(x_1, x_2, x_3, x_4)$

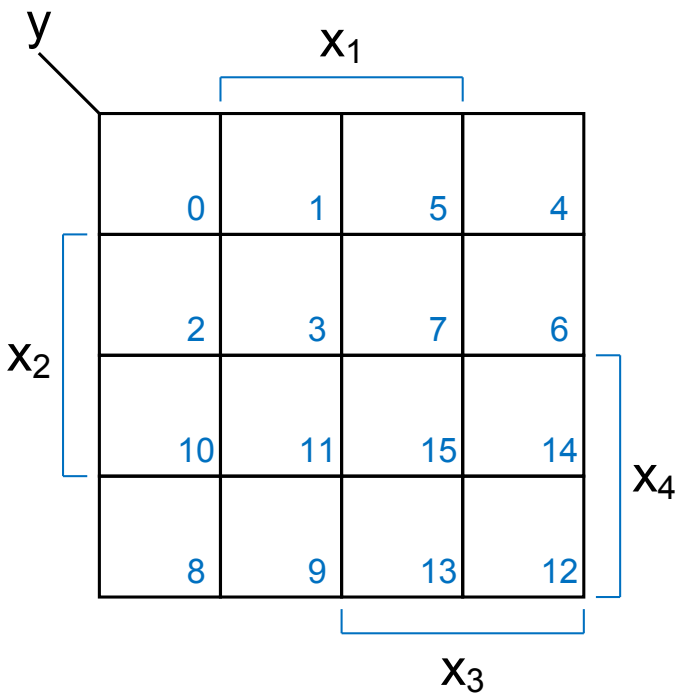


In die Felder des KV-Diagramms werden die Ausgangswerte y der Schaltfunktion je nach Wert der Eingangsvariablen eingetragen. Die blauen Zahlen geben die Zeile in einer Wahrheitstabelle an, aus dem der Funktionswert übernommen werden kann. Stimmt mit Index der Minterme überein, d.h. in Feld 0 kommt Wert von Minterm m_0.

Maurice Karnaugh
1924 –

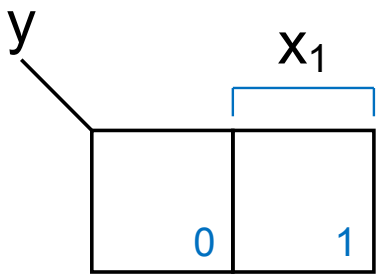
KV-Diagramm und Wahrheitstabelle

Zeile	x_4	x_3	x_2	x_1	$y = f(x_1, x_2, x_3, x_4)$
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0



Für die direkte Übertragung mit Hilfe der Zeilennummern ist die Anordnung der Variablen zu berücksichtigen!

KV-Diagramme – Konstruktion 1

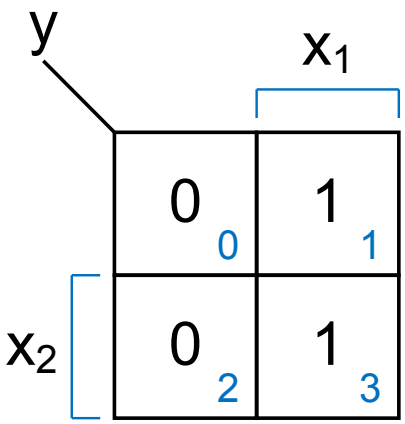


KV-Diagramm für eine unabhängige Variable

KV-Diagramm für zwei unabhängige Variablen:
Das KV-Diagramm für eine Schaltfunktion mit zwei Variablen wird aus dem KV-Diagramm mit einer Variablen durch Spiegelung an der horizontalen Achse konstruiert. Die neu hinzugekommene Variable x_2 wählt zwischen dem alten und neuen Teildiagramm aus.

KV-Diagramme - Minimierung 1

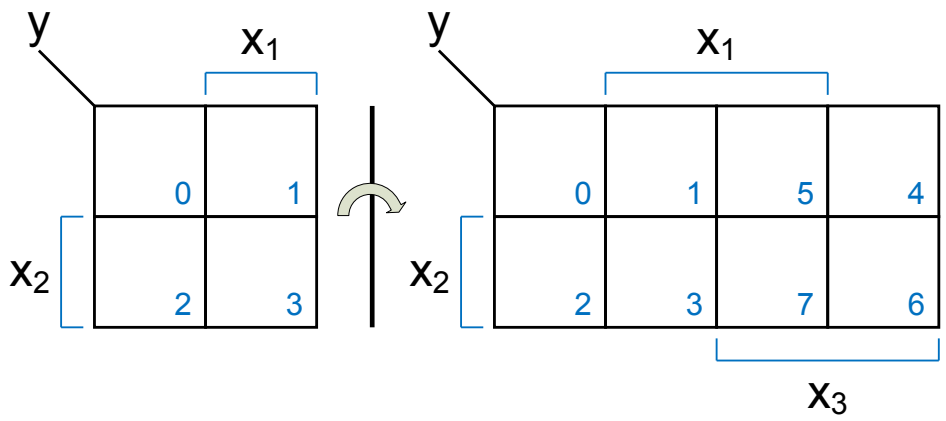
Beispiel: KV-Diagramm mit 2 Variablen



$$y = m_1 + m_3$$
$$= (x_1 * x_2\text{‘}) + (x_1 * x_2)$$
$$= x_1 * (x_2\text{‘} + x_2)$$
$$= x_1$$

KV-Diagramme können minimiert werden, indem benachbarte Felder mit dem Wert 1 zusammengefasst werden.

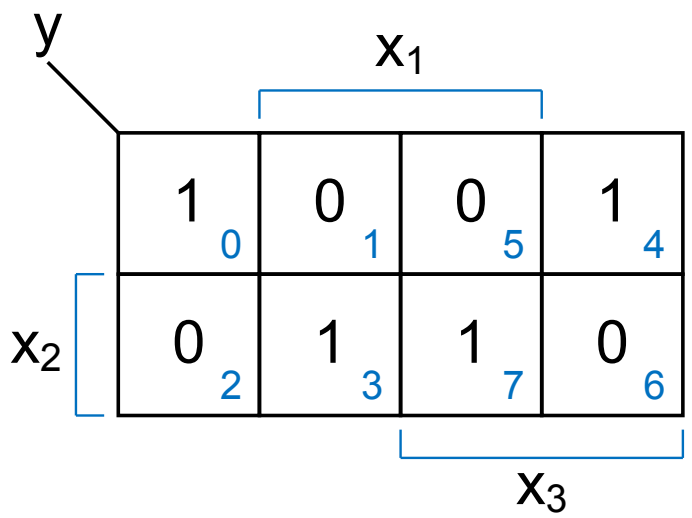
KV-Diagramme – Konstruktion 2



KV-Diagramm für drei unabhängige Variablen:
Das KV-Diagramm mit 2 Variablen wird dupliziert und an der vertikalen Achse gespiegelt. Die neu hinzugekommene Variable x_3 wählt zwischen dem alten und neuen Teildiagramm aus.

KV-Diagramme – Minimierung 2

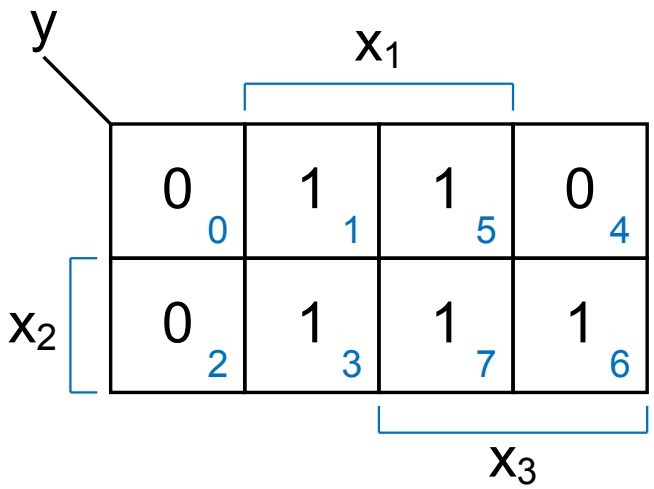
Beispiel: KV-Diagramm mit 3 Variablen



$$y = m_0 + m_4 + m_3 + m_7$$
$$= (x_1\text{‘} * x_2\text{‘} * x_3\text{‘}) + (x_1\text{‘} * x_2\text{‘} * x_3) + (x_1 * x_2 * x_3) + (x_1 * x_2 * x_3\text{‘})$$
$$= [(x_1\text{‘} * x_2\text{‘}) * (x_3\text{‘} + x_3)] + [(x_1 * x_2) * (x_3 + x_3\text{‘})]$$
$$= (x_1\text{‘} * x_2\text{‘}) + (x_1 * x_2)$$

KV-Diagramme – Minimierung 3

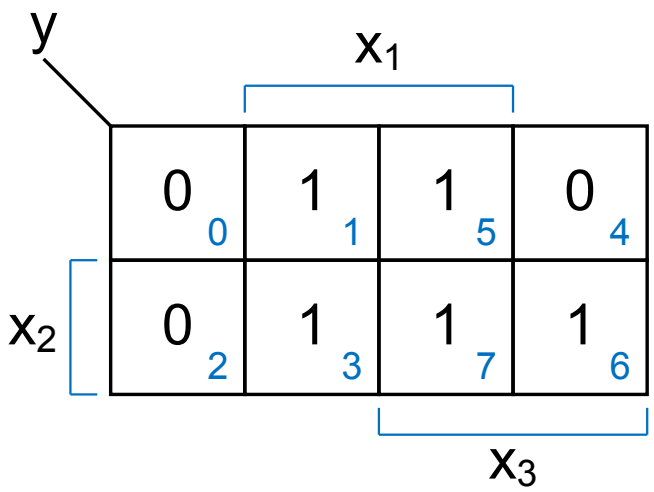
Zusammenfassung von größeren Gruppen



$$
\begin{aligned}
y &= m_1 + m_5 + m_3 + m_7 + m_6 \\
&= (x_1 * x_2\text{‘} * x_3\text{‘}) + (x_1 * x_2\text{‘} * x_3) + (x_1 * x_2 * x_3\text{‘}) + (x_1 * x_2 * x_3) + (x_1\text{‘} * x_2 * x_3) \\
&= [(x_1 * x_2\text{‘}) * (x_3\text{‘} + x_3)] + [(x_1 * x_2) * (x_3\text{‘} + x_3)] + (x_1\text{‘} * x_2 * x_3) \\
&= (x_1 * x_2\text{‘}) + (x_1 * x_2) + (x_1\text{‘} * x_2 * x_3) \\
&= [x_1 * (x_2\text{‘} + x_2)] + (x_1\text{‘} * x_2 * x_3) \\
&= x_1 + (x_1\text{‘} * x_2 * x_3)
\end{aligned}
$$

KV-Diagramme – Minimierung 4

Überdeckung



$$
\begin{aligned}
y &= m_1 + m_5 + m_3 + m_7 + m_6 \\
&= (x_1 * x_2\,' * x_3\,') + (x_1 * x_2\,' * x_3) + (x_1 * x_2 * x_3\,') + (x_1 * x_2 * x_3) + (x_1\,' * x_2 * x_3) \\
&= [(x_1 * x_2\,') * (x_3\,' + x_3)] + [(x_1 * x_2) * (x_3\,' + x_3)] + (x_1 * x_2 * x_3) + (x_1\,' * x_2 * x_3) \\
&= (x_1 * x_2\,') + (x_1 * x_2) + [(x_2 * x_3) * (x_1 + x_1\,')] \\
&= [x_1 * (x_2\,' + x_2)] + (x_2 * x_3) \\
&= x_1 + (x_2 * x_3)
\end{aligned}
$$

Implikanten

Erkenntnisse:

- Zusammenfassung von großen Minterm-Blöcken gibt Produktterme mit wenigen Literalen
- Je weniger Minterm-Gruppen gebildet werden, um alle Einser abzudecken, desto kleiner ist die Anzahl der Produktterme

Definition 9.2 (Implikant)
Ein Monom p, d.h. eine Konjunktion von Literalen, heißt *Implikant* für eine Funktion f, genau dann wenn gilt:
$(p = 1) \Rightarrow (f = 1)$

Man sagt, ein Monom p_1 überdeckt ein Monom p_2, wenn $(p_2 = 1) \Rightarrow (p_1 = 1)$

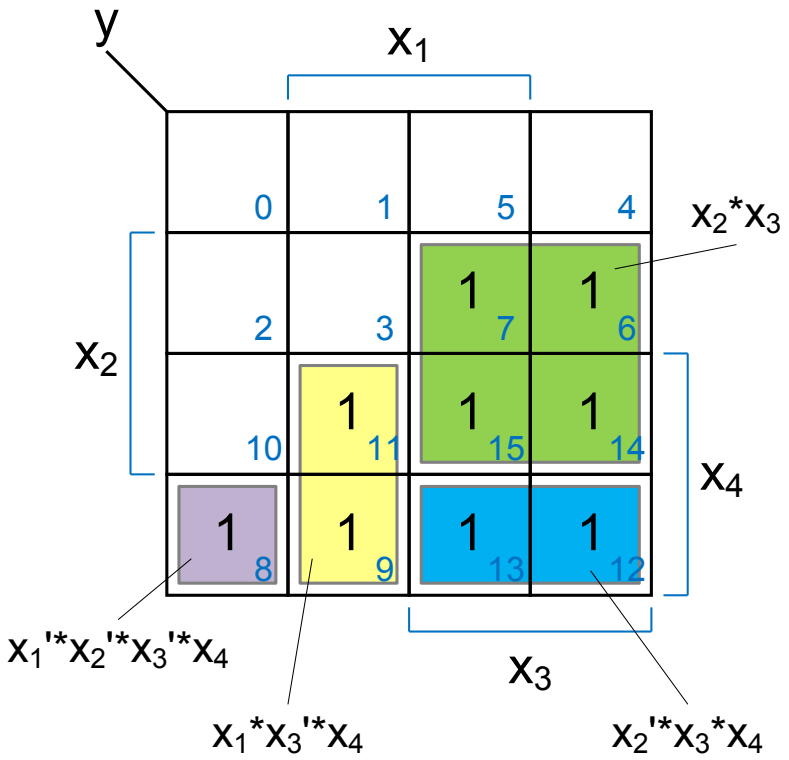
Beispiel: $f(x_1, x_2, x_3, x_4) = (x_1 * x_2 * x_3) + (x_1 ' * x_2 * x_3) + (x_1 * x_4) + (x_2 ' * x_4)$

Implikanten sind: $x_1 * x_2 * x_3$, $x_1 ' * x_2 * x_3$, $x_1 * x_4$, $x_2 ' * x_4$
Weiterer Implikant: $x_2 * x_3$
Der Implikant $x_2 * x_3$ überdeckt die Implikanten $x_1 * x_2 * x_3$ und $x_1 ' * x_2 * x_3$

Implikanten

Beispiel:

$f(x_1, x_2, x_3, x_4) = (x_1{*}x_2{*}x_3) + (x_1\text{'}{*}x_2{*}x_3) + (x_1{*}x_4) + (x_2\text{'}{*}x_4)$



Primimplikant

Definition 9.3 (Primimplikant)
Ein Implikant heißt *Primimplikant* (prime implicant) für eine Funktion f, wenn jeder Produktterm, der durch Wegstreichen eines Literals des Implikanten entsteht, kein Implikant für f ist.

Ein zu einem Primimplikanten gehörender Block im KV-Diagramm wird Primblock genannt.

Beispiel:

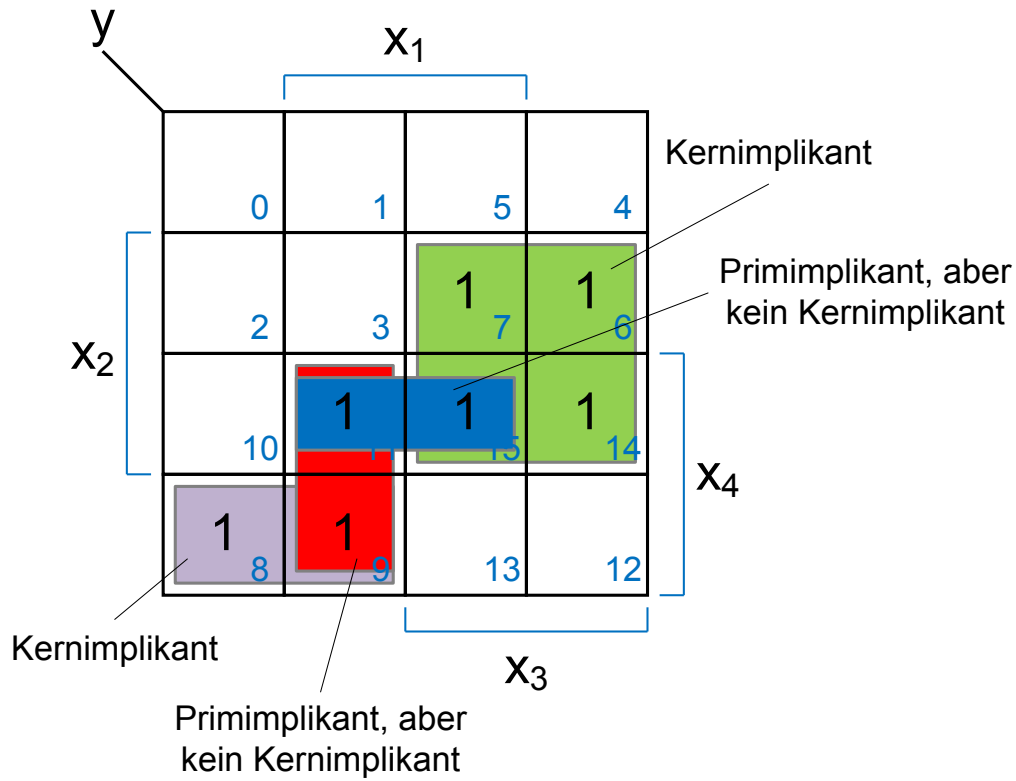
$f(x_1, x_2, x_3) = (x_1 * x_2) + (x_2 * x_3) + (x_1 * x_2 \text{ ‘} * x_3)$

Primimplikanten sind $x_1 * x_2$, $x_2 * x_3$, $x_1 * x_3$

Kernimplikant

Definition 9.4 (Kernimplikant)
Ein Primimplikant, der mindestens einen Minterm enthält, der in keinem anderen Primimplikanten enthalten ist, wird *Kernimplikant* genannt.

Beispiel:

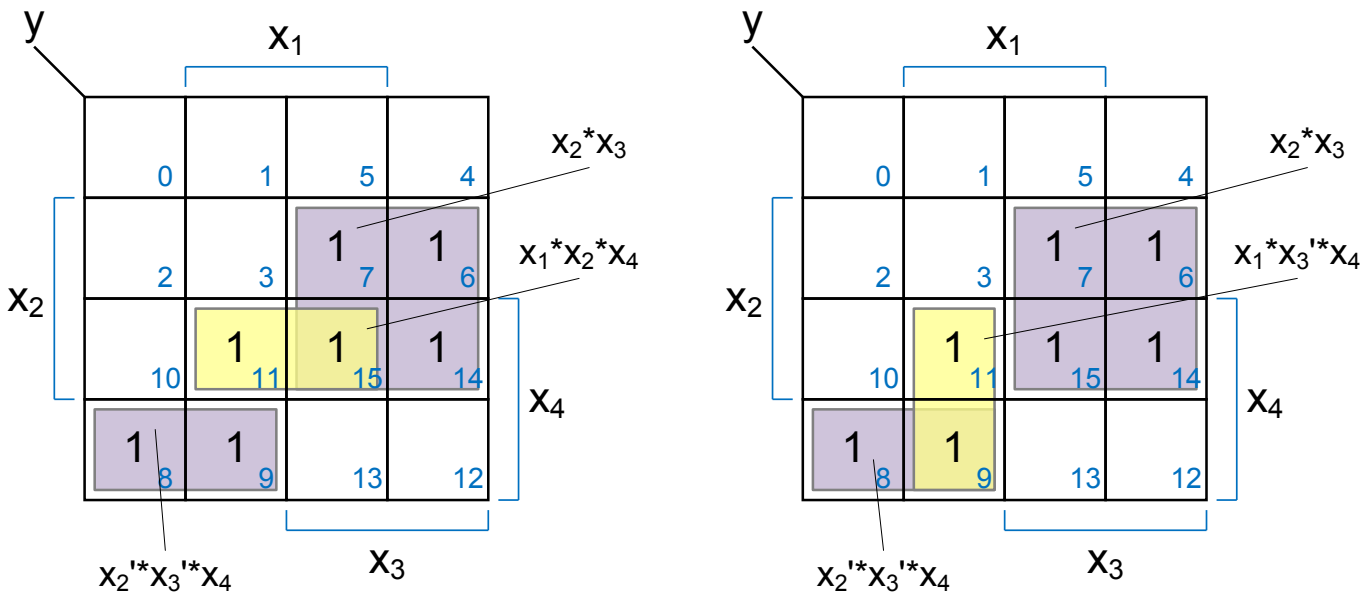


Minimale Summe

Definition 9.5 (Minimale Summe)
Eine *minimale Summe* einer Funktion f ist ein SOP-Ausdruck, so dass es keinen anderen SOP-Ausdruck gibt, der weniger Produktterme hat, und jeder SOP-Ausdruck, mit gleich vielen Produkttermen, mindestens genauso viele Literale besitzt.

Eine minimale Summe besteht nur aus Primimplikanten. Kernimplikanten müssen zur Summe gehören! Minimale Summe ist allerdings nicht immer eindeutig!

Beispiel:

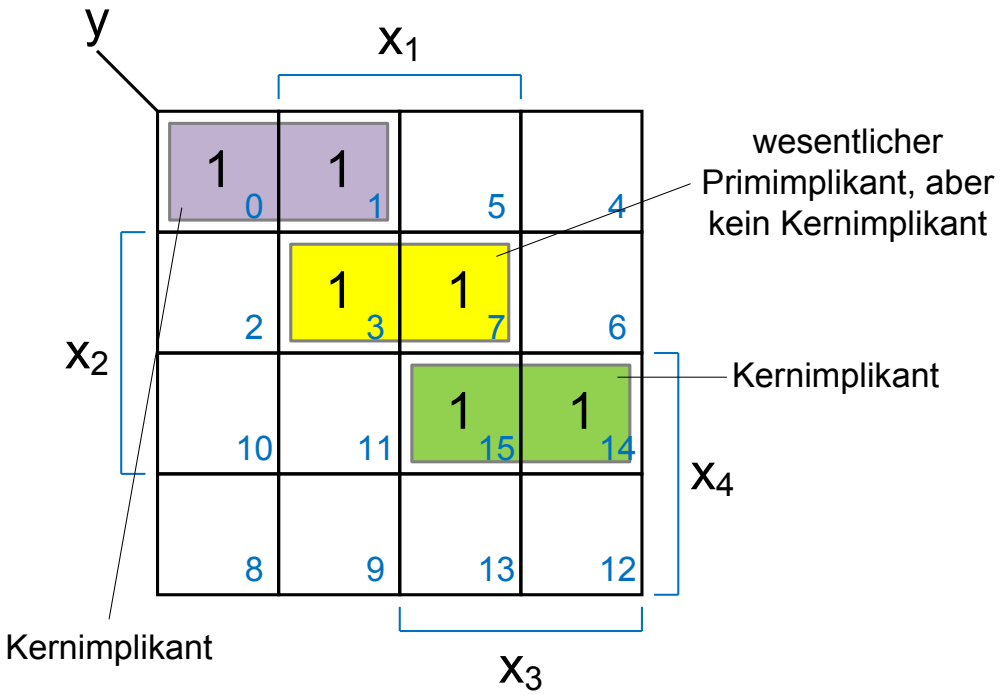


wesentlicher Primimplikant

Definition 9.6 (wesentlicher Primimplikant)
Ein Primimplikant, der zur Bildung der minimalen Summe zwingend verwendet werden muss, wird *wesentlicher Primimplikant* genannt.

Ein wesentlicher Primimplikant kann meist nicht direkt durch "anschauen" erkannt werden! (vergleiche 3.4)

Beispiel:



KV-Diagramme

Schritte zur Minimierung mit KV-Diagrammen

- Bestimmung aller Primblöcke
  Suche für jeden Einser im Diagramm den größtmöglichen Block benachbarter Einser => Primblock

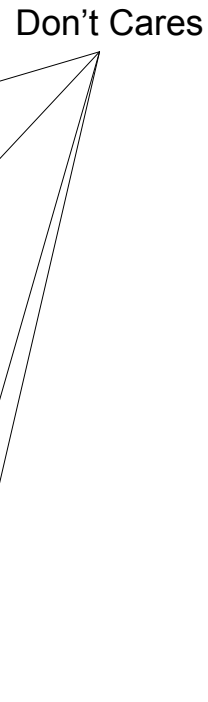
- Bestimmung der Kernimplikanten
  Wähle unter allen gefundenen Primblöcken diejenigen, welche eine Eins *alleine* überdecken (Kernblöcke) und markiere sie -> Kernimplikanten

- Bestimmung der vollständigen Überdeckung
  Wähle für mögliche Restmenge der Einser Primblöcke und markiere sie

- Extraktion der minimalen Summe
  Jeder gefundene Block der Größe 2^k wird durch einen Produktterm von n-k Variablen ausgedrückt (bei einer Funktion mit n Variablen). Minimale Summe ergibt sich aus der disjunktiven Verknüpfung dieser Terme

Unvollständig spezifizierte Funktionen

x_4	x_3	x_2	x_1	y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	-
0	1	1	0	0
0	1	1	1	-
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	-
1	1	1	0	0
1	1	1	1	-



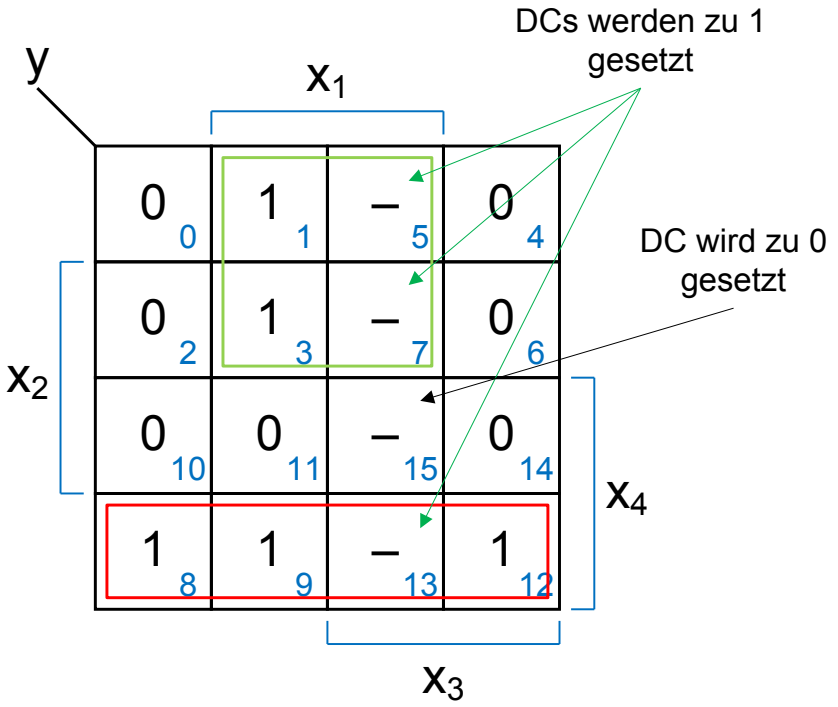
Es gibt Schaltfunktionen, die nur unvollständig spezifiziert sind. Man spricht auch von *partiell definierten Funktionen*. Für gewisse Eingangskombinationen ist der Ausgangswert nicht festgelegt. Er kann dann den Wert 0 oder 1 annehmen. Solche Ausgangswerte heißen Don ‘t-Care-Belegungen (DCs) und werden mit einem Bindestrich (-) dargestellt.
Für Don ‘t Cares gibt es wie für Minterme eine Kurzschreibweise:

$$f(x_1,x_2,x_3,x_4) = \sum m(1,3,8,9,12) + \sum d(5,7,13,15)$$

Minimierung mit DCs

DCs können bei der Minimierung ausgenutzt werden!

Beispiel:

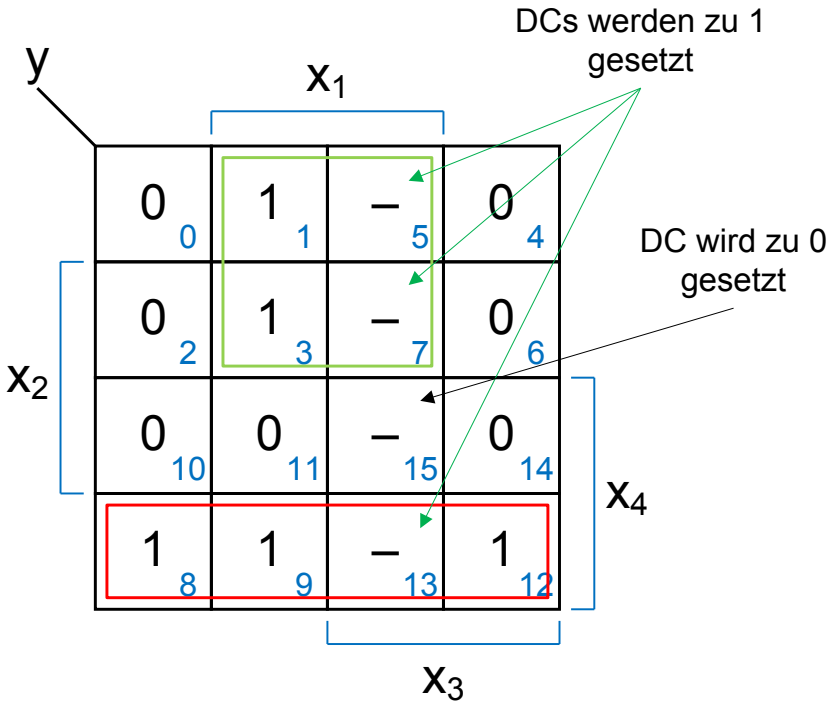


$y = (x_1 * x_4\,') + (x_2\,' * x_4)$

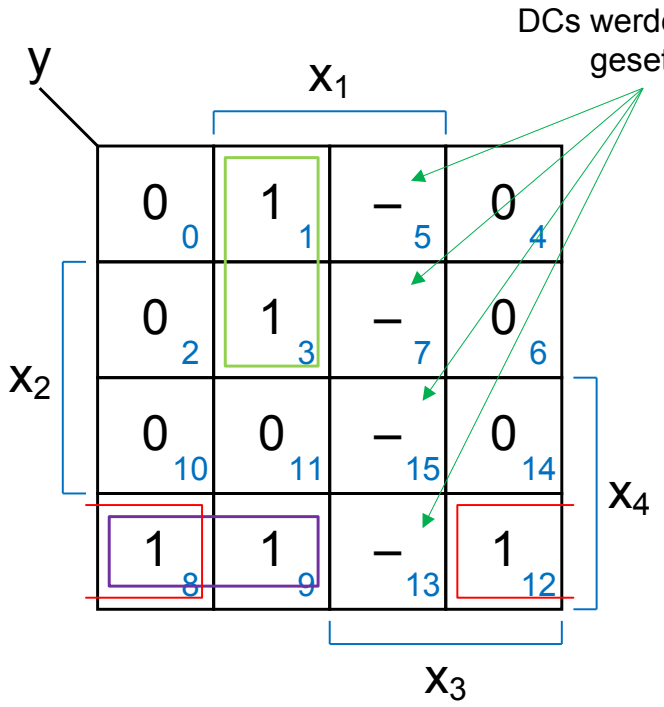
Minimierung mit DCs

DCs können bei der Minimierung ausgenutzt werden!

Beispiel:



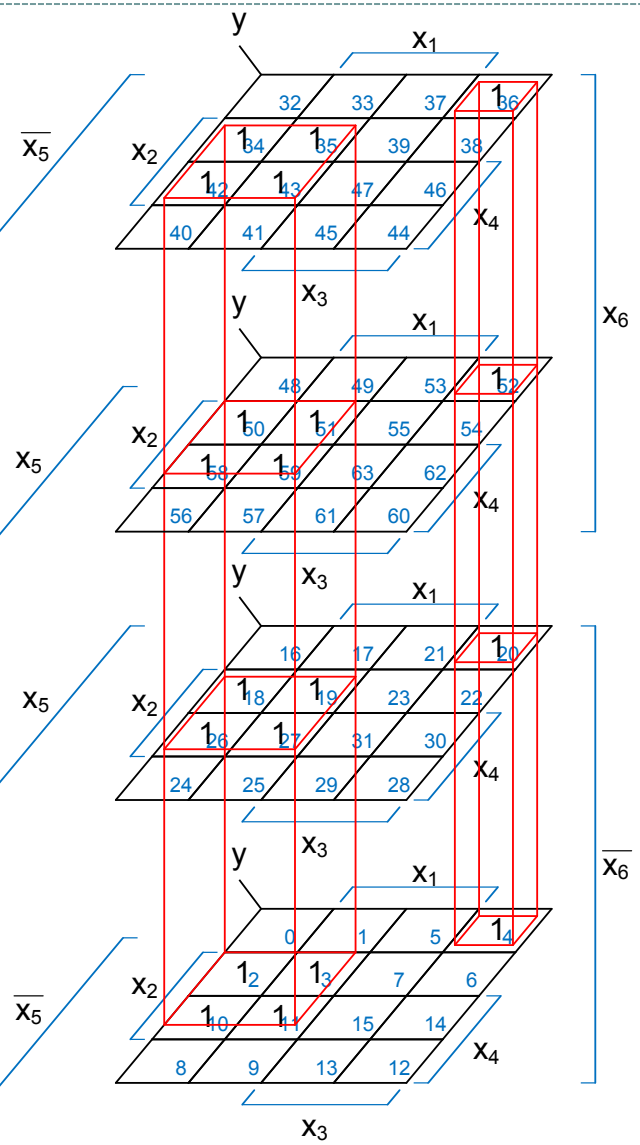
$y = (x_1 * x_4\,') + (x_2\,' * x_4)$

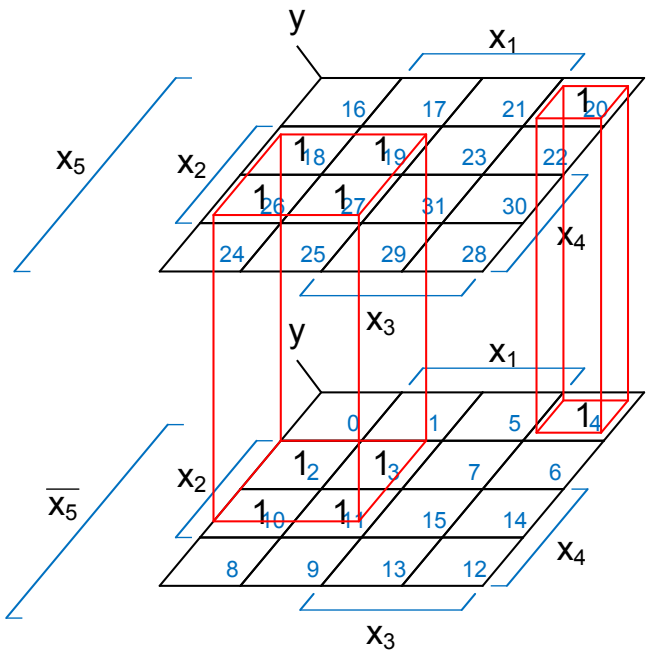


$y = (x_1 * x_3 * x_4\,') + (x_1\,' * x_2\,' * x_4) + (x_2\,' * x_3\,' * x_4)$

KV-Diagramme höherer Ordnung

KV-Diagramm
für 6 Variablen





KV-Diagramm für 5 Variablen

Quine-McCluskey Algorithmus

Tabellarisches Verfahren zur Minimierung

Algorithmus besteht aus 2 Schritten:

Schritt 1: Bestimmung der Primimplikanten mittels Primimplikantentabellen

Schritt 2: Konstruktion der minimalen Überdeckung mittels Überdeckungstabelle

Implikanten werden durch Kurzschreibweise dargestellt:

$x_4 * x_3 ' * x_2 * x_1 '$ entspricht: 1 0 1 0

Minimierung:

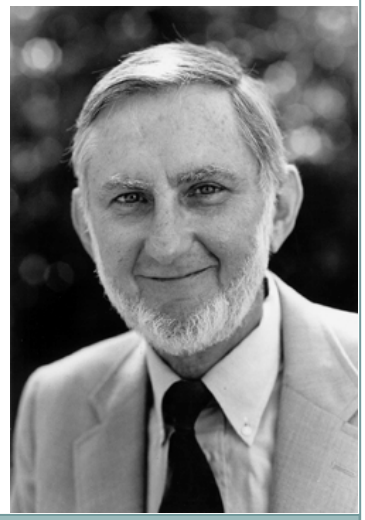
$(x_4 * x_3' * x_2 * x_1') + (x_4 * x_3' * x_2 * x_1) = x_4 * x_3' * x_2$

1 0 1 0

1 0 1 1

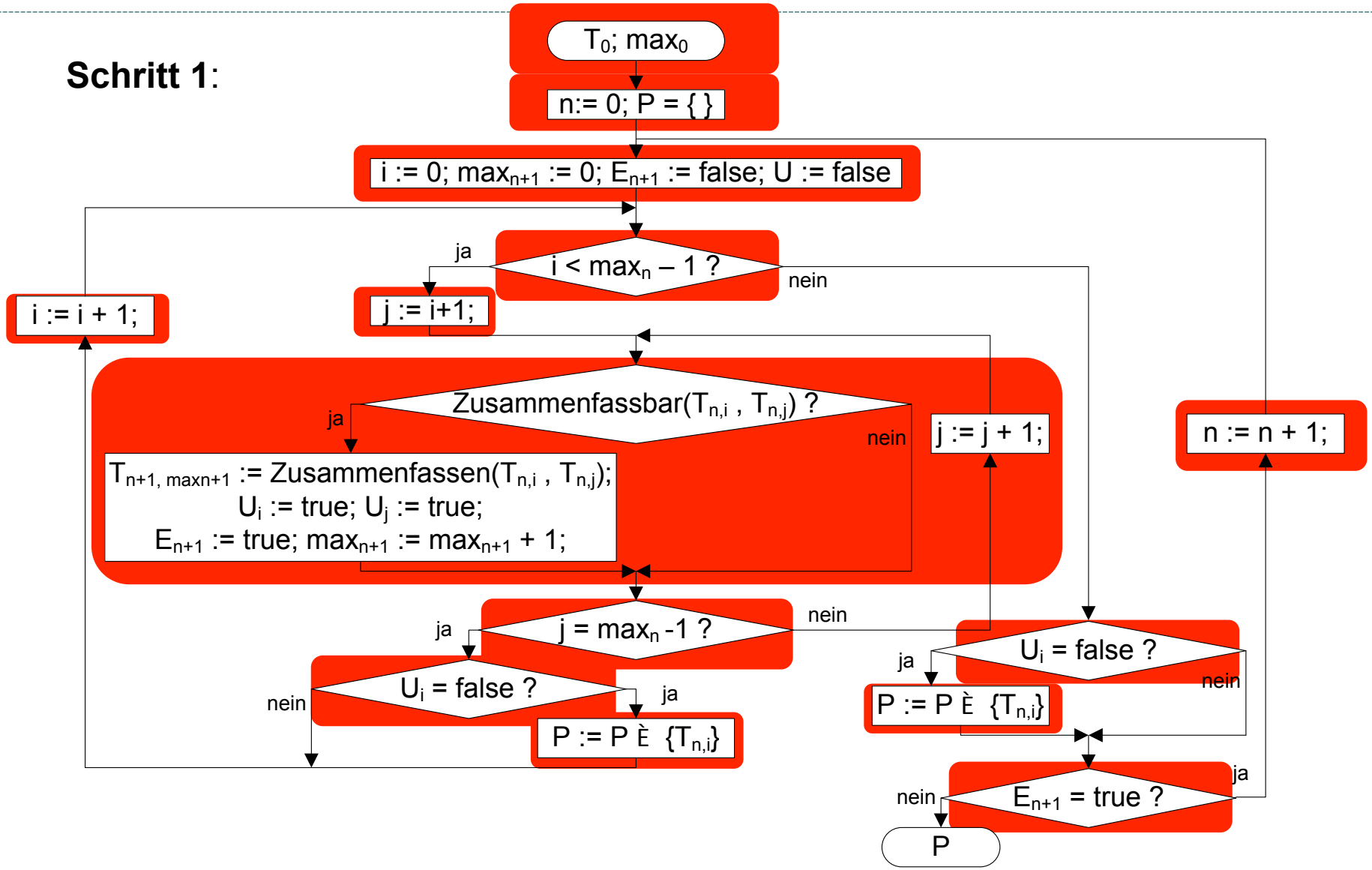
⇒ 1 0 1 –

Edward McCluskey 1929-2016



Quine-McCluskey Algorithmus

Schritt 1:



Quine-McCluskey Algorithmus

Input T_0, max_0 ; // Tabelle nullter Ordnung und Anzahl ihrer Zeilen
n := 0; P := { }; // aktuelle Tabellenordnung; gefundene Primimplikanten
do
    i := 0; max_{n+1} := 0; E_{n+1} := false; U_* := false;
    while(i < max_n – 1) do   // äußere Schleife für paarweisen Vergleich
        j := i + 1;
        while(j < max_n) do   // innere Schleife für paarweisen Vergleich
            if(zusammenfassbar($T_{n,i}$, $T_{n,j}$)) then
                $T_{n+1,maxn+1}$:= zusammenfassen($T_{n,i}$, $T_{n,j}$);
                U_i := true; U_j := true; E_{n+1} := true; max_{n+1} := max_{n+1} + 1;
            end if;
            j := j + 1;
        end while;
        if(U_i = false) then P := P + { $T_{n,i}$ }; end if;
        i := i + 1;
    end while; // i
    if(U_{maxn-1} = false) then P := P + { $T_{n,maxn-1}$ }; end if;
    n := n + 1;
while (E_n = true); // gehe zur nächst höheren Tabelle
Output P; // Menge der Primimplikanten

Bestimmung von Primimplikanten

Beispiel: $f(x_1, x_2, x_3, x_4) = \sum m(0,4,8,10,11,12,13,15)$

1. Schritt: Bestimmung von Primimplikanten

Implikanten nullter Ordnung:

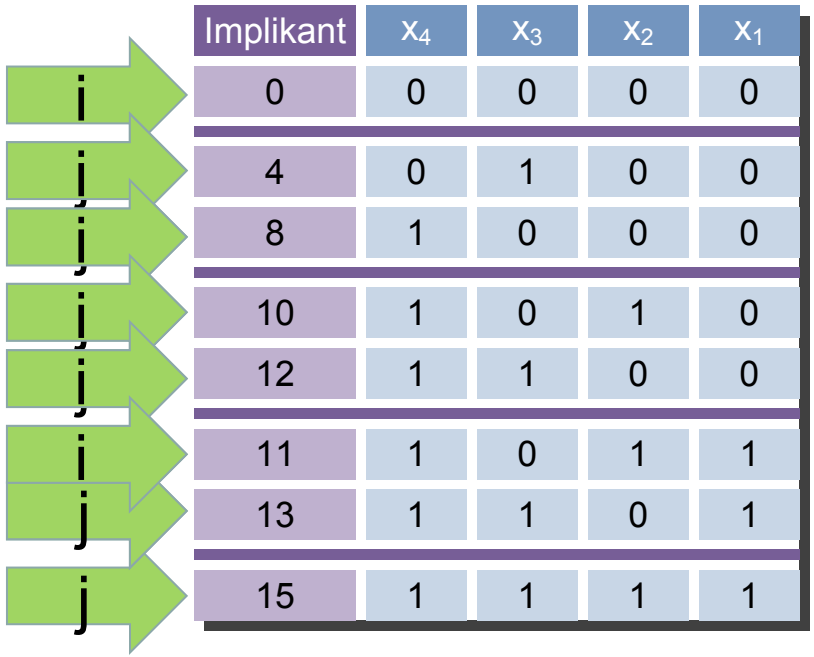
Implikant	x_4	x_3	x_2	x_1
0	0	0	0	0
4	0	1	0	0
8	1	0	0	0
10	1	0	1	0
12	1	1	0	0
11	1	0	1	1
13	1	1	0	1
15	1	1	1	1

Bestimmung von Primimplikanten

Beispiel: $f(x_1, x_2, x_3, x_4) = \sum m(0,4,8,10,11,12,13,15)$

1. Schritt: Bestimmung von Primimplikanten

Implikanten nullter Ordnung:



Implikant	x_4	x_3	x_2	x_1
0	0	0	0	0
4	0	1	0	0
8	1	0	0	0
10	1	0	1	0
12	1	1	0	0
11	1	0	1	1
13	1	1	0	1
15	1	1	1	1

Implikanten erster Ordnung:

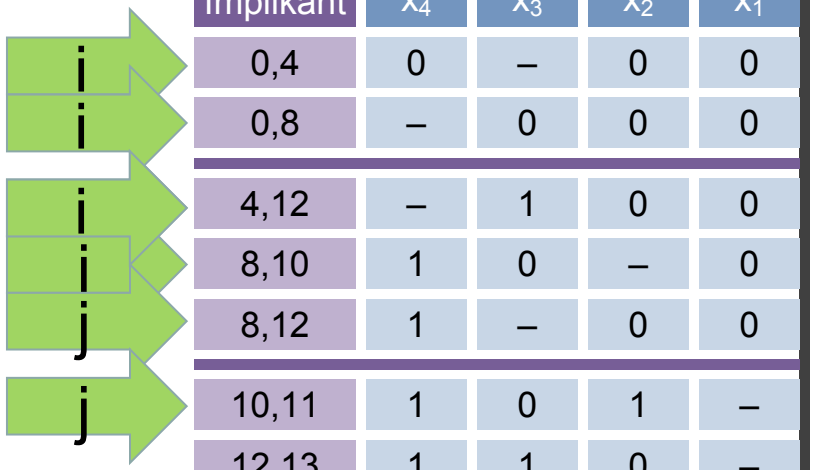
Implikant	x_4	x_3	x_2	x_1

Bestimmung von Primimplikanten

Beispiel: $f(x_1, x_2, x_3, x_4) = \sum m(0,4,8,10,11,12,13,15)$

1. Schritt: Bestimmung von Primimplikanten

Implikanten erster Ordnung:



Implikant	x_4	x_3	x_2	x_1
0,4	0	–	0	0
0,8	–	0	0	0
4,12	–	1	0	0
8,10	1	0	–	0
8,12	1	–	0	0
10,11	1	0	1	–
12,13	1	1	0	–
11,15	1	–	1	1
13,15	1	1	–	1

Implikanten zweiter Ordnung:

Implikant	x_4	x_3	x_2	x_1

Bestimmung von Primimplikanten

Beispiel: $f(x_1, x_2, x_3, x_4) = \sum m(0,4,8,10,11,12,13,15)$

1. Schritt: Bestimmung von Primimplikanten

Implikanten erster Ordnung:

Implikant	x_4	x_3	x_2	x_1	
0,4	0	–	0	0	✓
0,8	–	0	0	0	✓
4,12	–	1	0	0	✓
8,10	1	0	–	0	
8,12	1	–	0	0	✓
10,11	1	0	1	–	
12,13	1	1	0	–	
11,15	1	–	1	1	
13,15	1	1	–	1	

Implikanten zweiter Ordnung:

Implikant	x_4	x_3	x_2	x_1
0,4,8,12	–	–	0	0

gefundene Primimplikanten:

$(x_1'*x_3'*x_4)$
$(x_2*x_3'*x_4)$
$(x_2'*x_3*x_4)$
$(x_1*x_2*x_4)$
$(x_1*x_3*x_4)$
$(x_1'*x_2')$

Bestimmung von Primimplikanten

Schritt 1: Konstruktion der Primimplikantentafeln

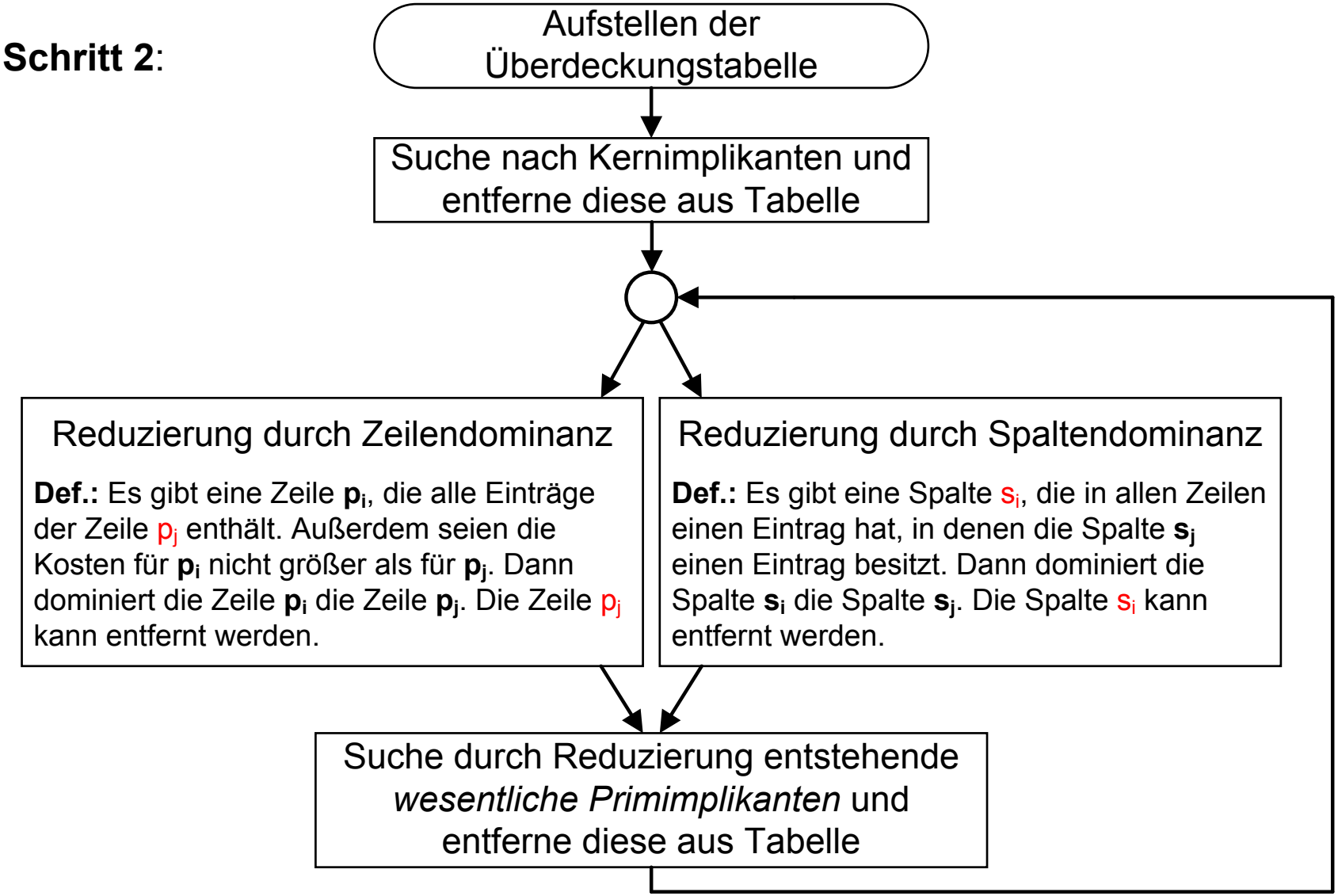
- Aufstellung einer Tabelle mit den Implikanten nullter Ordnung (Minterme) gruppiert nach der Anzahl der Einser.
  - Gruppen, die nicht benachbart sind, unterscheiden sich an mehr als einer Stelle und eignen sich damit nicht zum Zusammenfassen.

- alle Implikanten benachbarter Gruppen vergleichen, wenn nur in einer Stelle unterschiedlich, dann zusammenfassen
  - zusammengefasste Implikanten werden markiert
  - neuer Implikant wird in Tabelle erster Ordnung eingetragen

- nach gleichem Schema Tabellen höherer Ordnung aufstellen, bis keine Zusammenfassung mehr möglich
  - unmarkierte Einträge sind gefundene Primimplikanten

Quine-McCluskey Algorithmus

Schritt 2:



Quine-McCluskey Algorithmus

2. Schritt: Bestimmung der minimalen Überdeckung mit Überdeckungstabelle

Implikant	x_4	x_3	x_2	x_1	m_0	m_4	m_8	m_{10}	m_{11}	m_{12}	m_{13}	m_{15}
8,10	1	0	–	0			X	X				
10,11	1	0	1	–				X	X			
12,13	1	1	0	–						X	X	
11,15	1	–	1	1					X			X
13,15	1	1	–	1							X	X
0,4,8,12	–	–	0	0	X	X	X			X		

Quine-McCluskey Algorithmus

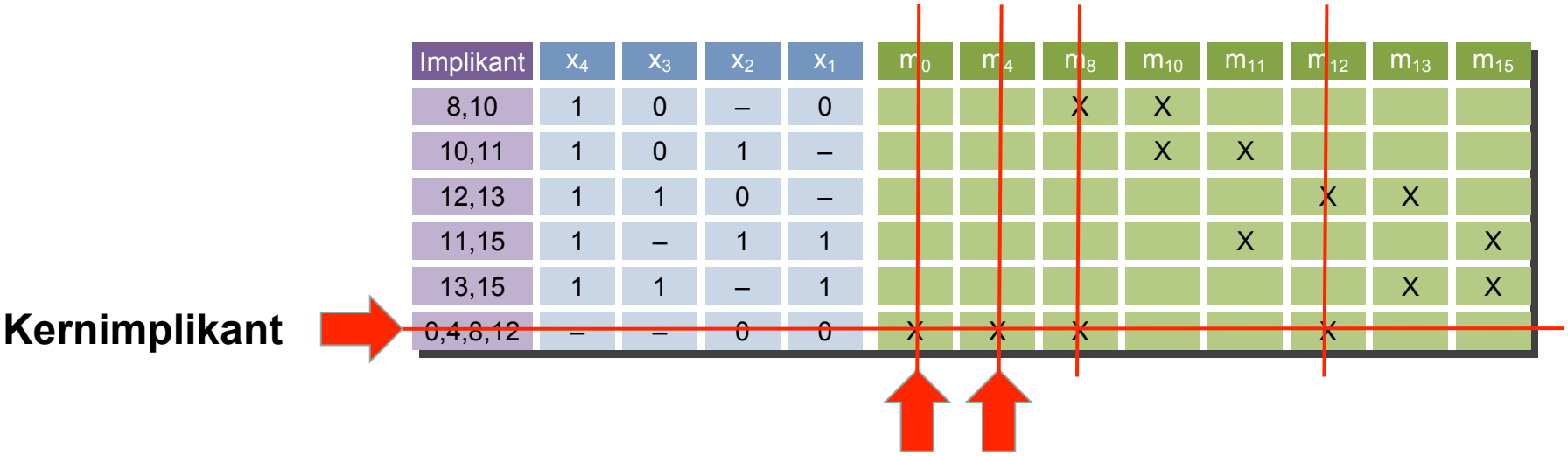
2. Schritt: Bestimmung der minimalen Überdeckung mit Überdeckungstabelle

Implikant	x_4	x_3	x_2	x_1	m_0	m_4	m_8	m_{10}	m_{11}	m_{12}	m_{13}	m_{15}
8,10	1	0	–	0			X	X				
10,11	1	0	1	–				X	X			
12,13	1	1	0	–						X	X	
11,15	1	–	1	1					X			X
13,15	1	1	–	1							X	X
0,4,8,12	–	–	0	0	X	X	X			X		

Kernimplikant

Quine-McCluskey Algorithmus

2. Schritt: Bestimmung der minimalen Überdeckung mit Überdeckungstabelle



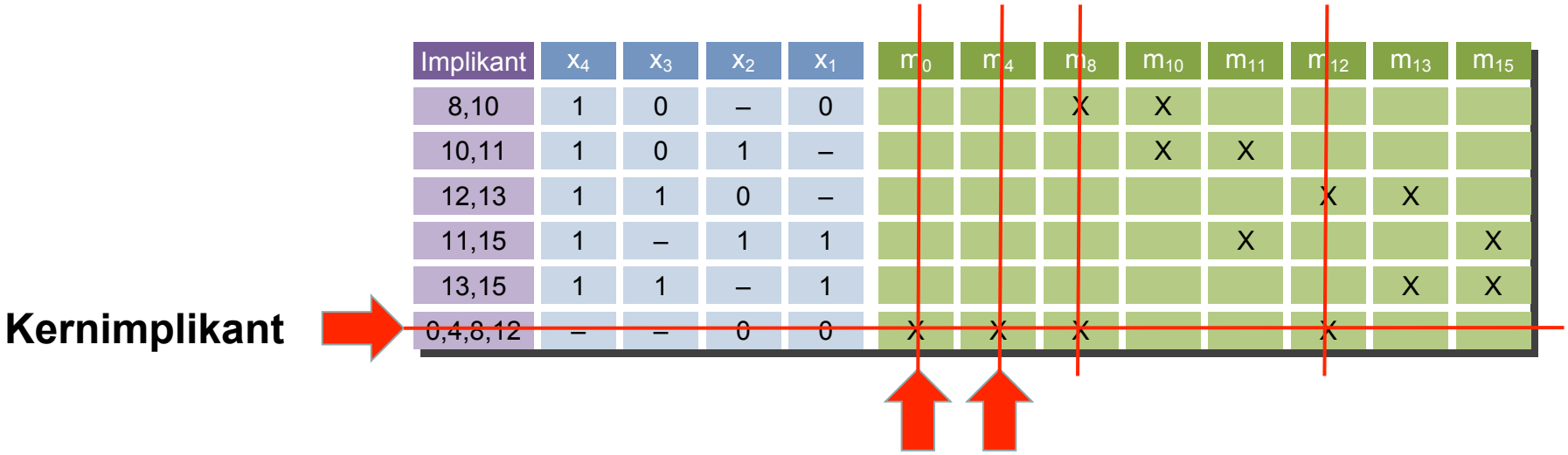
Implikant	x_4	x_3	x_2	x_1	m_0	m_4	m_8	m_{10}	m_{11}	m_{12}	m_{13}	m_{15}
8,10	1	0	–	0			X	X				
10,11	1	0	1	–				X	X			
12,13	1	1	0	–						X	X	
11,15	1	–	1	1					X			X
13,15	1	1	–	1							X	X
0,4,8,12	–	–	0	0	X	X	X			X		

Kernimplikant

Entfernen der Kernimplikanten und überdeckter Minterme:

Quine-McCluskey Algorithmus

2. Schritt: Bestimmung der minimalen Überdeckung mit Überdeckungstabelle



Implikant	x_4	x_3	x_2	x_1	m_0	m_4	m_8	m_{10}	m_{11}	m_{12}	m_{13}	m_{15}
8,10	1	0	–	0			X	X				
10,11	1	0	1	–				X	X			
12,13	1	1	0	–						X	X	
11,15	1	–	1	1					X			X
13,15	1	1	–	1							X	X
0,4,8,12	–	–	0	0	X	X	X			X		

Entfernen der Kernimplikanten und überdeckter Minterme:

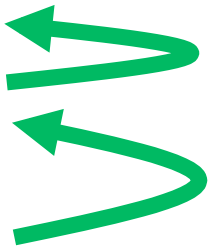
Implikant	x_4	x_3	x_2	x_1	m_{10}	m_{11}	m_{13}	m_{15}
8,10	1	0	–	0	X			
10,11	1	0	1	–	X	X		
12,13	1	1	0	–			X	
11,15	1	–	1	1		X		X
13,15	1	1	–	1			X	X

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_{10}	m_{11}	m_{13}	m_{15}
8,10	1	0	–	0	X			
10,11	1	0	1	–	X	X		
12,13	1	1	0	–			X	
11,15	1	–	1	1		X		X
13,15	1	1	–	1			X	X

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_{10}	m_{11}	m_{13}	m_{15}
8,10	1	0	–	0	X			
10,11	1	0	1	–	X	X		
12,13	1	1	0	–			X	
11,15	1	–	1	1		X		X
13,15	1	1	–	1			X	X

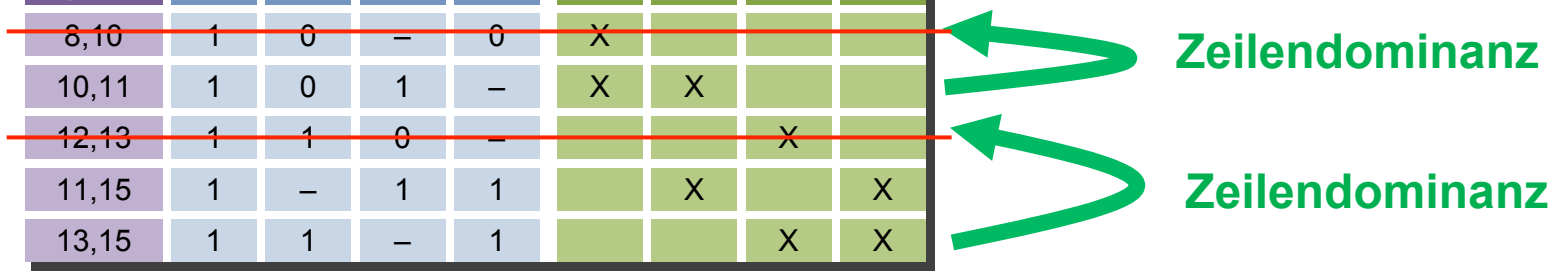


Zeilendominanz

Zeilendominanz

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_{10}	m_{11}	m_{13}	m_{15}
~~8,10~~	~~1~~	~~0~~	~~–~~	~~0~~	X			
10,11	1	0	1	–	X	X		
~~12,13~~	~~1~~	~~1~~	~~0~~	~~–~~			X	
11,15	1	–	1	1		X		X
13,15	1	1	–	1			X	X



Zeilendominanz: Entfernung von Implikanten, die von anderen Implikanten mit gleichen Kosten dominiert werden!

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_{10}	m_{11}	m_{13}	m_{15}
~~8,10~~	~~1~~	~~0~~	~~–~~	~~0~~	X			
10,11	1	0	1	–	X	X		
~~12,13~~	~~1~~	~~1~~	~~0~~	~~–~~			X	
11,15	1	–	1	1		X		X
13,15	1	1	–	1			X	X

Zeilendominanz

Zeilendominanz

Zeilendominanz: Entfernung von Implikanten, die von anderen Implikanten mit gleichen Kosten dominiert werden!

Implikant	x_4	x_3	x_2	x_1	m_{10}	m_{11}	m_{13}	m_{15}
10,11	1	0	1	–	X	X		
11,15	1	–	1	1		X		X
13,15	1	1	–	1			X	X

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_{10}	m_{11}	m_{13}	m_{15}
~~8,10~~	~~1~~	~~0~~	~~–~~	~~0~~	X			
10,11	1	0	1	–	X	X		
~~12,13~~	~~1~~	~~1~~	~~0~~	~~–~~			X	
11,15	1	–	1	1		X		X
13,15	1	1	–	1			X	X

Zeilendominanz

Zeilendominanz

Zeilendominanz: Entfernung von Implikanten, die von anderen Implikanten mit gleichen Kosten dominiert werden!

Implikant	x_4	x_3	x_2	x_1	m_{10}	m_{11}	m_{13}	m_{15}
10,11	1	0	1	–	X	X		
11,15	1	–	1	1		X		X
13,15	1	1	–	1			X	X

wesentlicher Primimplikant

wesentlicher Primimplikant

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_{10}	m_{11}	m_{13}	m_{15}
~~8,10~~	~~1~~	~~0~~	~~–~~	~~0~~	X			
10,11	1	0	1	–	X	X		
~~12,13~~	~~1~~	~~1~~	~~0~~	~~–~~			X	
11,15	1	–	1	1		X		X
13,15	1	1	–	1			X	X

Zeilendominanz

Zeilendominanz

Zeilendominanz: Entfernung von Implikanten, die von anderen Implikanten mit gleichen Kosten dominiert werden!

Implikant	x_4	x_3	x_2	x_1	m_{10}	m_{11}	m_{13}	m_{15}
10,11	1	0	1	–	X	X		
11,15	1	–	1	1		X		X
13,15	1	1	–	1			X	X

wesentlicher Primimplikant

wesentlicher Primimplikant

Resultat:
gefundene Implikanten für minimale Summe: $(x_1'*x_2')$; $(x_2*x_3'*x_4)$; $(x_1*x_3*x_4)$

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

Quine-McCluskey Algorithmus

Schritt 2: Konstruktion der Überdeckungstabelle

1.Aufstellung einer Tabelle mit den gefundenen Primimplikanten in jeder Zeile. Jede Spalte entspricht einem Minterm. Falls der Primimplikant den Minterm überdeckt, wird ein X in der Spalte eingetragen

2. Suche nach Primimplikanten, die zur Lösung gehören:
- Jede Spalte, die nur ein X enthält, wird gelöscht.
- Der zugehörige Primimplikant gehört zur Lösung
- Alle anderen Spalten, die von diesem Primimplikant abgedeckt werden, werden aus Tabelle entfernt

3. Reduzierung der Überdeckungstabelle durch Löschen von dominanten Spalten und dominierten Zeilen

4. Iterative Anwendung von Schritt 2 und 3, bis keine weitere Reduzierung möglich

Quine-McCluskey Algorithmus

Beispiel: $f(x_1, x_2, x_3, x_4) = \sum m(0,2,5,6,7,8,9,13) + \sum d(1,12,15)$

Don ‘t Care werden zur Ermittlung der Primimplikanten einbezogen

Implikanten nullter Ordnung:

Implikant	x_4	x_3	x_2	x_1	
0	0	0	0	0	✓
1	0	0	0	1	✓
2	0	0	1	0	✓
8	1	0	0	0	✓
5	0	1	0	1	✓
6	0	1	1	0	✓
9	1	0	0	1	✓
12	1	1	0	0	✓
7	0	1	1	1	✓
13	1	1	0	1	✓
15	1	1	1	1	✓

Implikanten erster Ordnung:

Implikant	x_4	x_3	x_2	x_1
0,1	0	0	0	–
0,2	0	0	–	0
0,8	–	0	0	0
1,5	0	–	0	1
1,9	–	0	0	1
2,6	0	–	1	0
8,9	1	0	0	–
8,12	1	–	0	0
5,7	0	1	–	1
5,13	–	1	0	1
6,7	0	1	1	–
9,13	1	–	0	1
12,13	1	1	0	–
7,15	–	1	1	1
13,15	1	1	–	1

Quine-McCluskey Algorithmus

Beispiel: $f(x_1, x_2, x_3, x_4) = \sum m(0,2,5,6,7,8,9,13) + \sum d(1,12,15)$

Don ‘t Care werden zur Ermittlung der Primimplikanten einbezogen

Implikanten erster Ordnung:

Implikant	x_4	x_3	x_2	x_1
0,1	0	0	0	–
0,2	0	0	–	0
0,8	–	0	0	0
1,5	0	–	0	1
1,9	–	0	0	1
2,6	0	–	1	0
8,9	1	0	0	–
8,12	1	–	0	0
5,7	0	1	–	1
5,13	–	1	0	1
6,7	0	1	1	–
9,13	1	–	0	1
12,13	1	1	0	–
7,15	–	1	1	1
13,15	1	1	–	1

Implikanten zweiter Ordnung:

Implikant	x_4	x_3	x_2	x_1

Quine-McCluskey Algorithmus

Beispiel: $f(x_1, x_2, x_3, x_4) = \sum m(0,2,5,6,7,8,9,13) + \sum d(1,12,15)$

Don ‘t Care werden zur Ermittlung der Primimplikanten einbezogen

Implikanten erster Ordnung:

Implikant	x_4	x_3	x_2	x_1	
0,1	0	0	0	–	✓
0,2	0	0	–	0	
0,8	–	0	0	0	✓
1,5	0	–	0	1	✓
1,9	–	0	0	1	✓
2,6	0	–	1	0	
8,9	1	0	0	–	✓
8,12	1	–	0	0	✓
5,7	0	1	–	1	✓
5,13	–	1	0	1	✓
6,7	0	1	1	–	
9,13	1	–	0	1	✓
12,13	1	1	0	–	✓
7,15	–	1	1	1	✓
13,15	1	1	–	1	✓

Implikanten zweiter Ordnung:

Implikant	x_4	x_3	x_2	x_1
0,1,8,9	–	0	0	–
1,5,9,13	–	–	0	1
8,9,12,13	1	–	0	–
5,7,13,15	–	1	–	1

gefundene Primimplikanten:

$(x_1'*x_3'*x_4')$
$(x_1'*x_2*x_4)$
$(x_2*x_3*x_4')$
$(x_2'*x_3')$
(x_1*x_2')
$(x_2'*x_4)$
(x_1*x_3)

Quine-McCluskey Algorithmus

Überdeckungstabelle: Don ‘t Cares (d_x) werden nicht berücksichtigt!

Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8	m_9	m_{13}
0,2	0	0	–	0	X	X						
2,6	0	–	1	0		X		X				
6,7	0	1	1	–				X	X			
0,1,8,9	–	0	0	–	X					X	X	
1,5,9,13	–	–	0	1			X				X	X
8,9,12,13	1	–	0	–						X	X	X
5,7,13,15	–	1	–	1			X		X			X

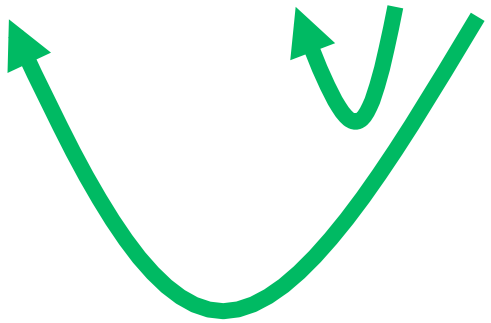
keine Kernimplikanten

Quine-McCluskey Algorithmus

Überdeckungstabelle: Don ‘t Cares (d_x) werden nicht berücksichtigt!

Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8	m_9	m_{13}
0,2	0	0	–	0	X	X						
2,6	0	–	1	0		X		X				
6,7	0	1	1	–				X	X			
0,1,8,9	–	0	0	–	X					X	X	
1,5,9,13	–	–	0	1			X				X	X
8,9,12,13	1	–	0	–						X	X	X
5,7,13,15	–	1	–	1			X		X			X

keine Kernimplikanten



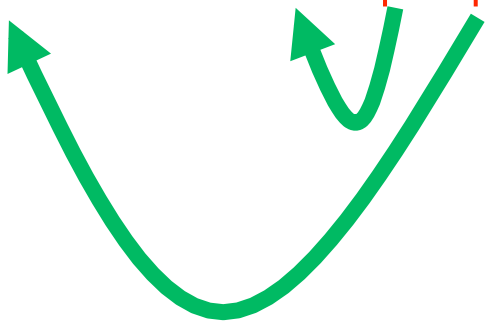
Spaltendominanz

Quine-McCluskey Algorithmus

Überdeckungstabelle: Don ‘t Cares (d_x) werden nicht berücksichtigt!

Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8	m_9	m_{13}
0,2	0	0	–	0	X	X						
2,6	0	–	1	0		X		X				
6,7	0	1	1	–				X	X			
0,1,8,9	–	0	0	–	X					X	X	
1,5,9,13	–	–	0	1			X				X	X
8,9,12,13	1	–	0	–						X	X	X
5,7,13,15	–	1	–	1			X		X			X

keine Kernimplikanten



Spaltendominanz

=> m_9 und m_{13} löschen

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8
0,2	0	0	–	0	X	X				
2,6	0	–	1	0		X		X		
6,7	0	1	1	–				X	X	
0,1,8,9	–	0	0	–	X					X
1,5,9,13	–	–	0	1			X			
8,9,12,13	1	–	0	–						X
5,7,13,15	–	1	–	1			X		X	

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8
0,2	0	0	–	0	X	X				
2,6	0	–	1	0		X		X		
6,7	0	1	1	–				X	X	
0,1,8,9	–	0	0	–	X					X
1,5,9,13	–	–	0	1			X			
8,9,12,13	1	–	0	–						X
5,7,13,15	–	1	–	1			X		X	

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8
0,2	0	0	–	0	X	X				
2,6	0	–	1	0		X		X		
6,7	0	1	1	–				X	X	
0,1,8,9	–	0	0	–	X					X
~~1,5,9,13~~			0	1			X			
~~8,9,12,13~~	1	–	0	–						X
5,7,13,15	–	1	–	1			X		X	

=> (1,5,9,13) und (8,9,12,13) löschen

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8
0,2	0	0	–	0	X	X				
2,6	0	–	1	0		X		X		
6,7	0	1	1	–				X	X	
0,1,8,9	–	0	0	–	X					X
~~1,5,9,13~~			0	1			X			
~~8,9,12,13~~	1	–	0	–						X
5,7,13,15	–	1	–	1			X		X	

=> (1,5,9,13) und (8,9,12,13) löschen

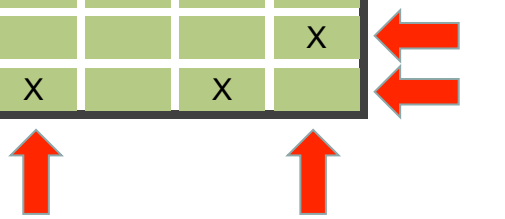
Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8
0,2	0	0	–	0	X	X				
2,6	0	–	1	0		X		X		
6,7	0	1	1	–				X	X	
0,1,8,9	–	0	0	–	X					X
5,7,13,15	–	1	–	1			X		X	

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8
0,2	0	0	–	0	X	X				
2,6	0	–	1	0		X		X		
6,7	0	1	1	–				X	X	
0,1,8,9	–	0	0	–	X					X
~~1,5,9,13~~			0	1			X			
~~8,9,12,13~~	1	–	0	–						X
5,7,13,15	–	1	–	1			X		X	

=> (1,5,9,13) und (8,9,12,13) löschen

Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8
0,2	0	0	–	0	X	X				
2,6	0	–	1	0		X		X		
6,7	0	1	1	–				X	X	
0,1,8,9	–	0	0	–	X					X
5,7,13,15	–	1	–	1			X		X	



wesentliche Primimplikanten

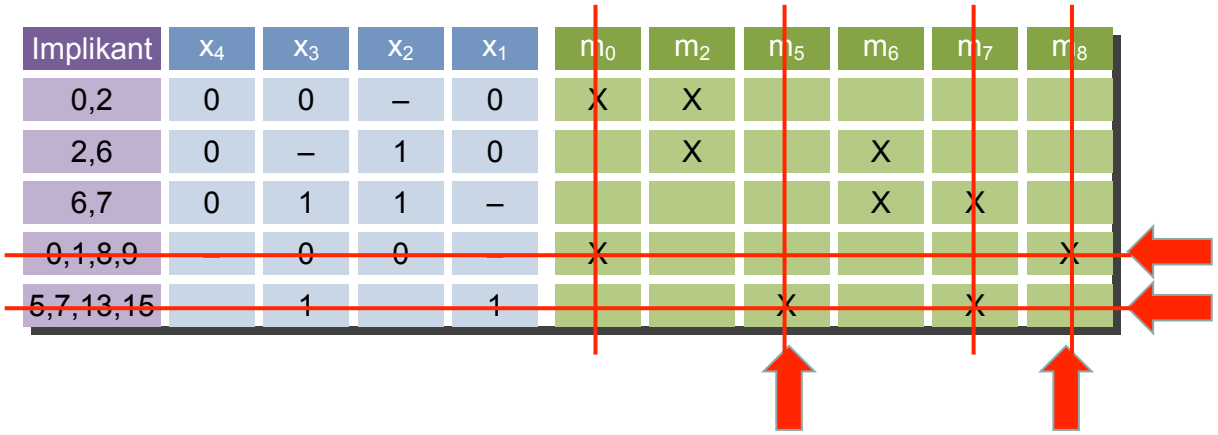
Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8
0,2	0	0	–	0	X	X				
2,6	0	–	1	0		X		X		
6,7	0	1	1	–				X	X	
0,1,8,9	–	0	0	–	X					X
~~1,5,9,13~~			0	1			X			
~~8,9,12,13~~	1	–	0	–						X
5,7,13,15	–	1	–	1			X		X	

Zeilendominanz

=> (1,5,9,13) und (8,9,12,13) löschen

Implikant	x_4	x_3	x_2	x_1	m_0	m_2	m_5	m_6	m_7	m_8
0,2	0	0	–	0	X	X				
2,6	0	–	1	0		X		X		
6,7	0	1	1	–				X	X	
~~0,1,8,9~~	–	0	0	–	X					X
~~5,7,13,15~~		1		1			X		X	



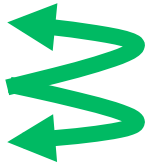
wesentliche Primimplikanten

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_2	m_6
0,2	0	0	–	0	X	
2,6	0	–	1	0	X	X
6,7	0	1	1	–		X

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_2	m_6
0,2	0	0	–	0	X	
2,6	0	–	1	0	X	X
6,7	0	1	1	–		X

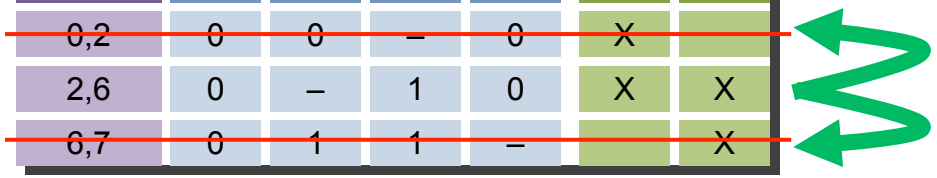


Zeilendominanz

=> (0,2) und (6,7) löschen

Quine-McCluskey Algorithmus

Implikant	x_4	x_3	x_2	x_1	m_2	m_6
~~0,2~~	~~0~~	~~0~~	~~–~~	~~0~~	X	
2,6	0	–	1	0	X	X
~~6,7~~	~~0~~	~~1~~	~~1~~	~~–~~		X



Zeilendominanz

=> (0,2) und (6,7) löschen

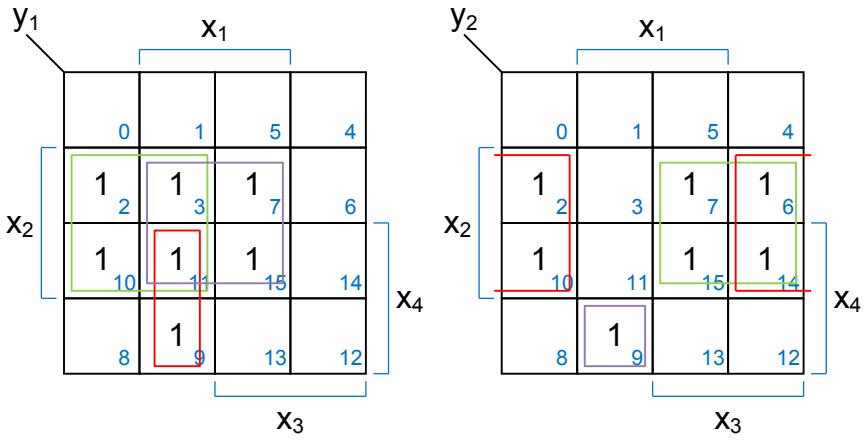
Resultat:

gefundene Implikanten für minimale Summe: $(x_2'*x_3')$; (x_1*x_3) ; $(x_1'*x_2*x_4')$

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

Bündelminimierung

- getrennte Minimierung

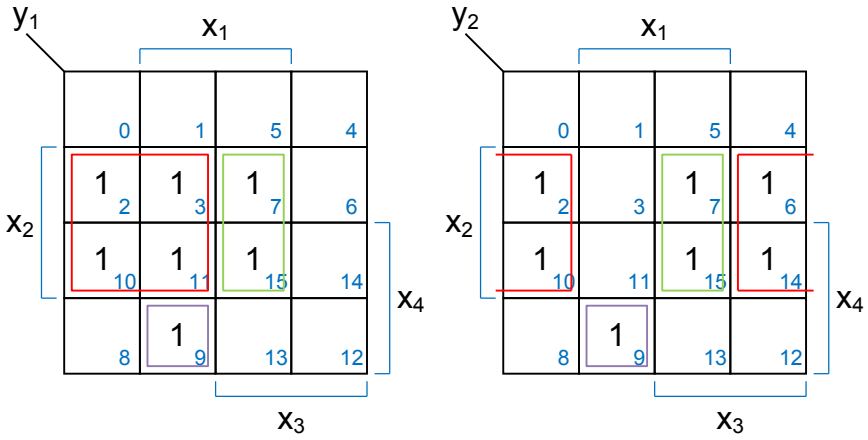


$y_1 = (x_1 * x_2) + (x_2 * x_3\,') + (x_1 * x_3\,' * x_4)$
Kosten: 2xUND2 1xUND3 1xODER3

$y_2 = (x_1\,' * x_2) + (x_2 * x_3) + (x_1 * x_2\,' * x_3\,' * x_4)$
Kosten: 2xUND2 1xUND4 1xODER3

Gesamt:
4xUND2 1xUND3 1xUND4 2xODER3

- Bündelminimierung

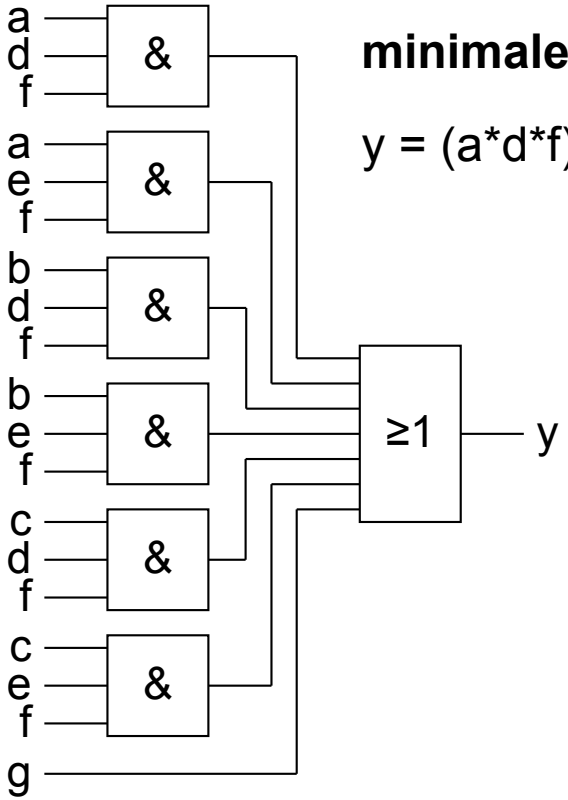


$y_1 = (x_2 * x_3\,') + (x_1 * x_2 * x_3) + (x_1 * x_2\,' * x_3\,' * x_4)$

$y_2 = (x_1\,' * x_2) + (x_1 * x_2 * x_3) + (x_1 * x_2\,' * x_3\,' * x_4)$

Gesamt:
2xUND2 1xUND3 1xUND4 2xODER3

mehrstufige Minimierung



minimaler SOP-Ausdruck für Funktion y = f(a, b, c, d, e, f, g):

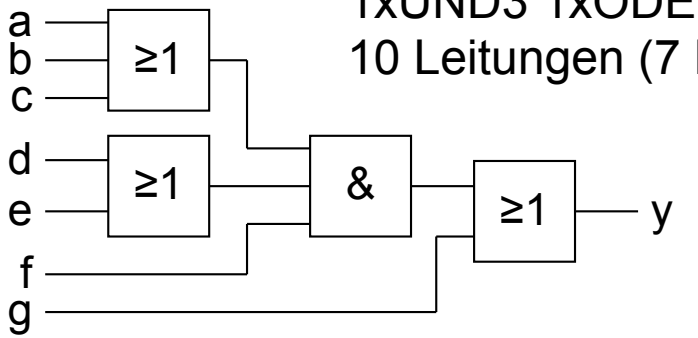
y = (a*d*f) + (a*e*f) + (b*d*f) + (b*e*f) + (c*d*f) + (c*e*f) + g

6xUND3 1xODER7
25 Leitungen (19 Literale und 6 interne)

faktorisierter Ausdruck:

y = [(a*d) + (a*e) + (b*d) + (b*e) + (c*d) + (c*e)] * f + g
= [((a + b + c) * d) + ((a + b + c) * e)] * f + g
= (a + b + c) * (d + e) * f + g

1xUND3 1xODER3 2xODER2
10 Leitungen (7 Literale und 3 interne)



Heuristiken

- Aufgrund des zu hohen Rechenaufwands lassen sich die Minimierungsprobleme in der Praxis nicht exakt lösen.
- Es werden daher Verfahren eingesetzt, die eine gute, aber nicht die optimale Lösung finden. Dabei werden verschiedene Strategien benutzt ("Faustregeln"), die sich in der Praxis bewährt haben -> Heuristiken.

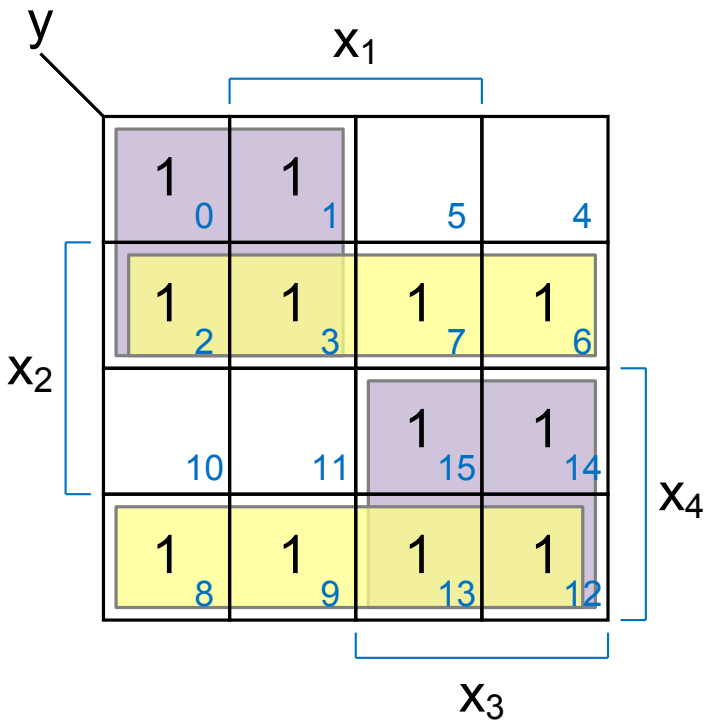
Bekanntestes Beispiel für zweistufige Minimierung mit Heuristiken ist Espresso:

- Generiert nicht zuerst alle Primimplikanten, geht stattdessen von einer Menge von Implikanten aus, welche alle Einsstellen überdecken
- Besteht aus mehreren Schritten, welche nacheinander wiederholt ausgeführt werden, solange die neue Lösung besser als die alte ist

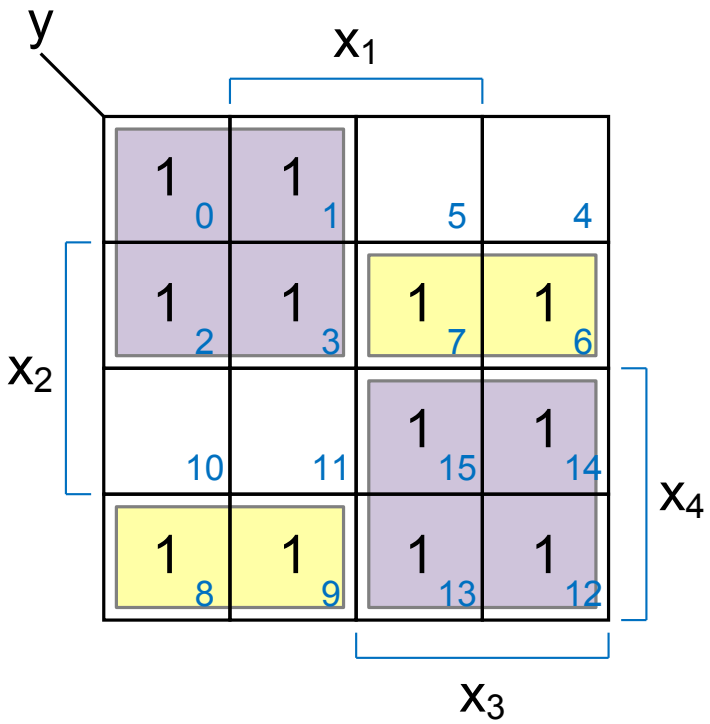
Heuristiken

Schritte bei Espresso:

- Im Schritt REDUCE werden die Primimplikanten so weit verkleinert, so dass sie die Einsstellen immer noch abdecken



Ausgangszustand

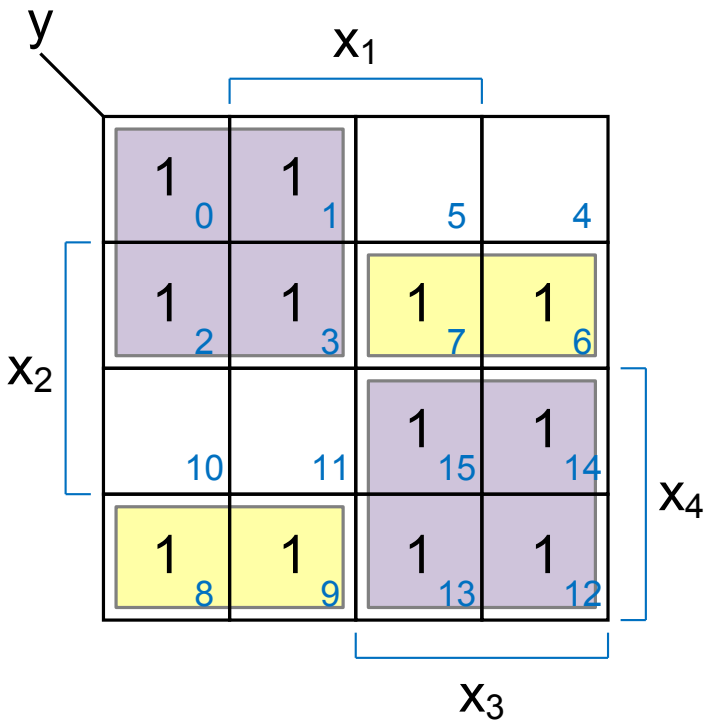


nach REDUCE-Schritt

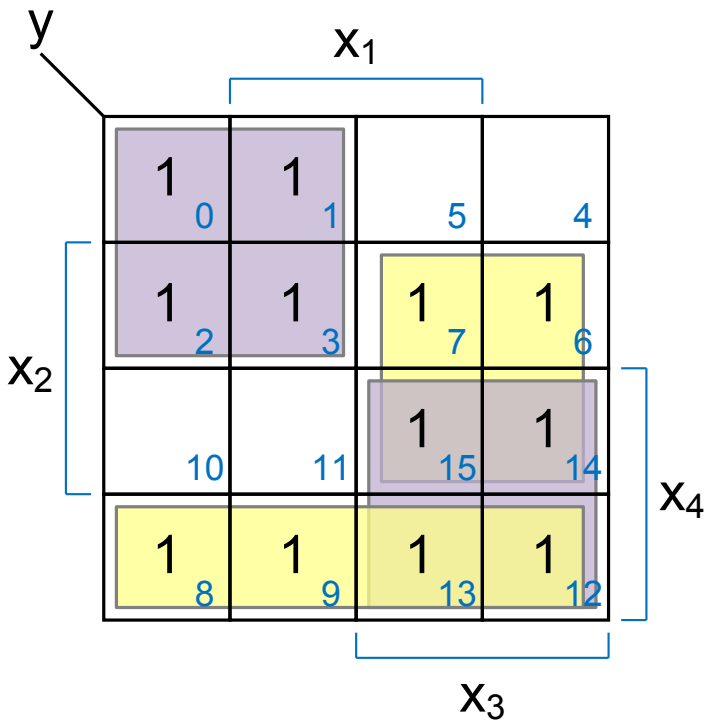
Heuristiken

Schritte bei Espresso:

- Im EXPAND-Schritt werden Implikanten zu ihrer maximalen Größe vergrößert und dadurch Primimplikanten gebildet



nach REDUCE-Schritt

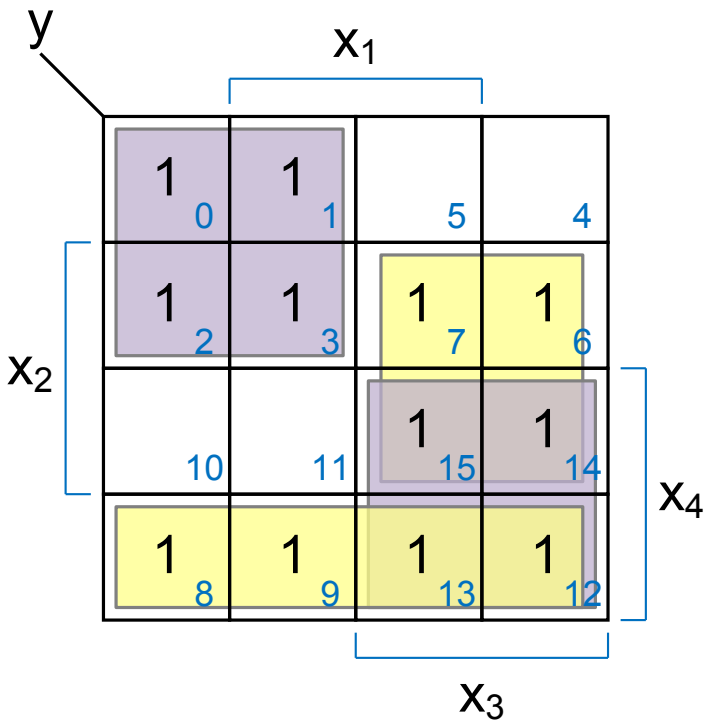


nach EXPAND-Schritt

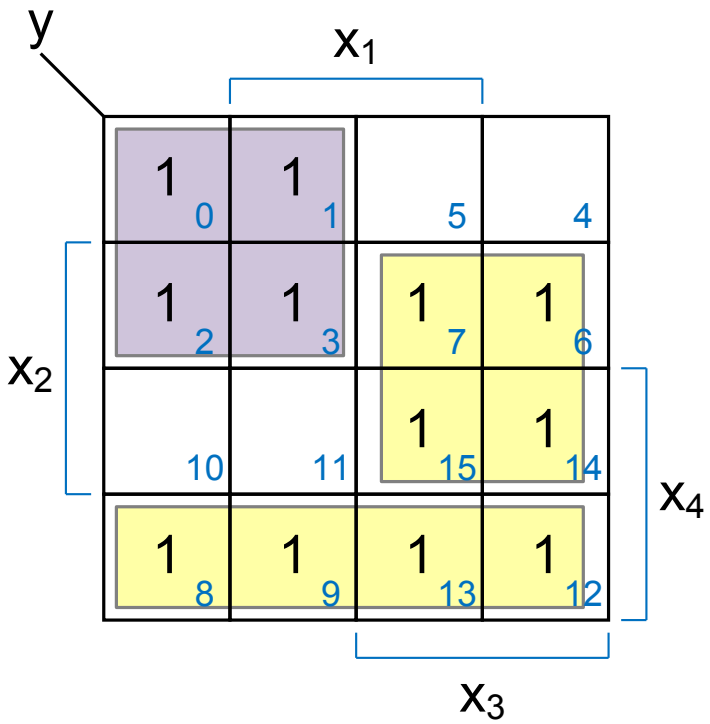
Heuristiken

Schritte bei Espresso:

- Im Schritt IRREDUNDANT COVER werden Implikanten entfernt, die nicht benötigt werden, um alle Einsstellen abzudecken

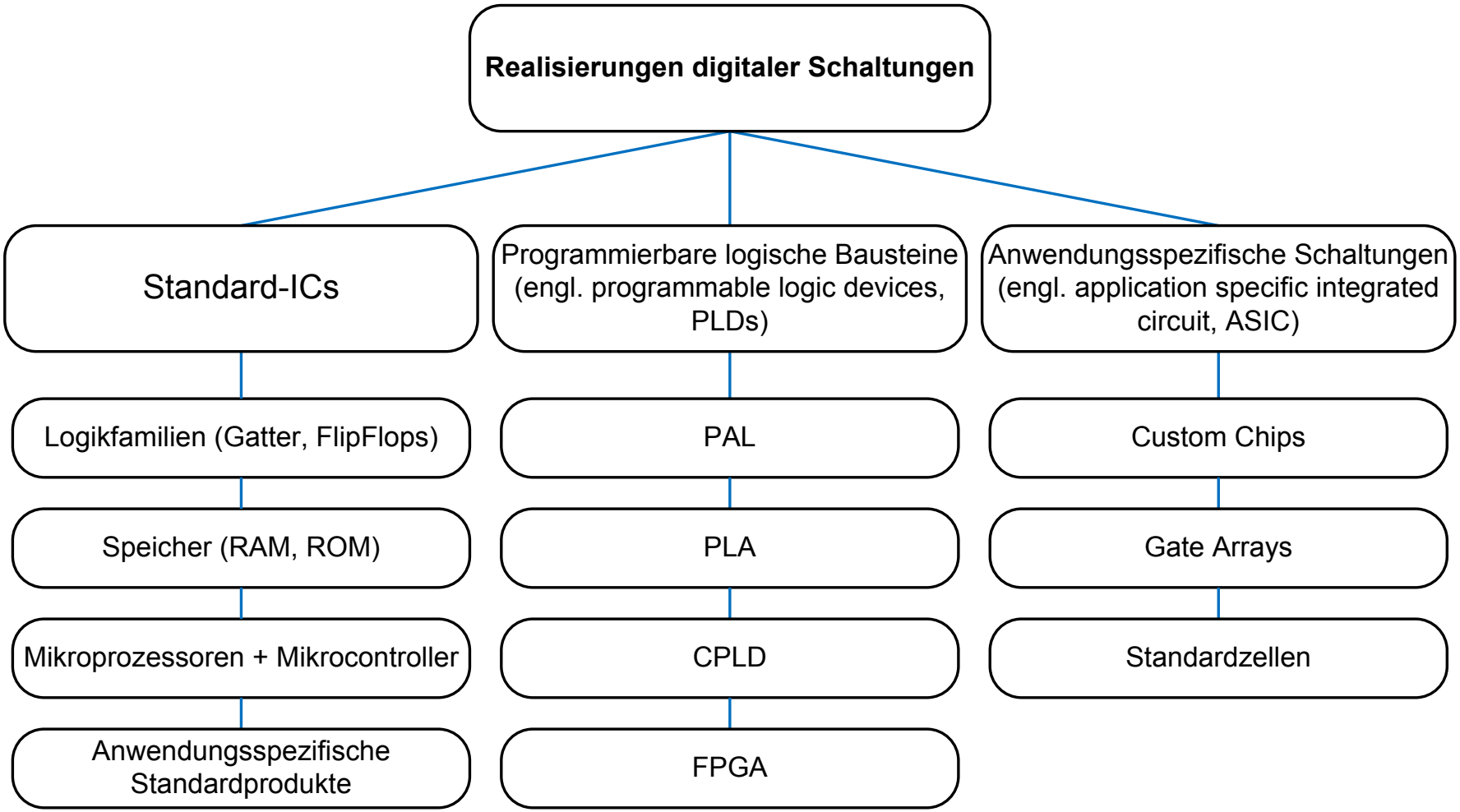


nach EXPAND-Schritt



nach IRREDUNDANT COVER-Schritt

Realisierungen



Vollständige Operatorensysteme

Definition 9.7 (Vollständiges Operatorensystem)
Sei M eine beliebe Menge von Operatoren. M ist ein *vollständiges Operatorensystem*, wenn sich jede Schaltfunktion durch einen Ausdruck beschreiben lässt, in dem neben den Variablen x_1, x_2, ..., x_n ausschließlich Operatoren aus M vorkommen.

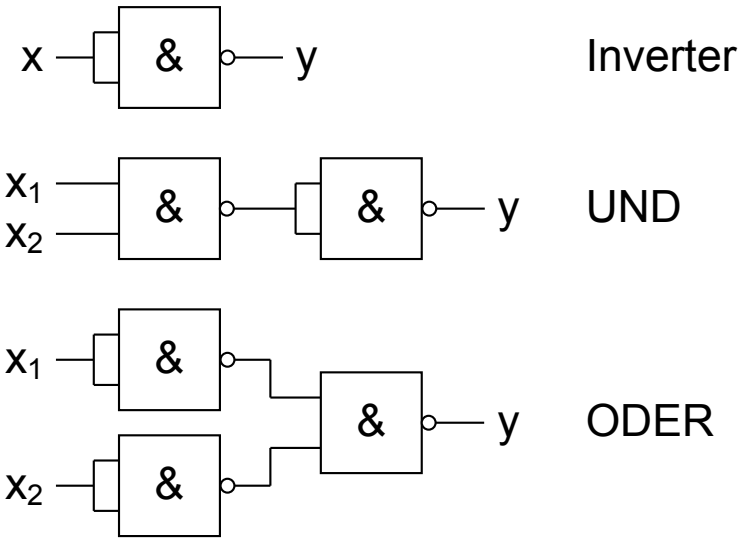
Die bisher benutzen Grundverknüpfungen UND, ODER, NICHT bilden ein vollständiges Operatorensystem (auch Basissystem der Schaltalgebra genannt).

Andere wichtige Basissysteme:
{NAND}, {NOR}, {UND, NICHT}, {ODER, NICHT}

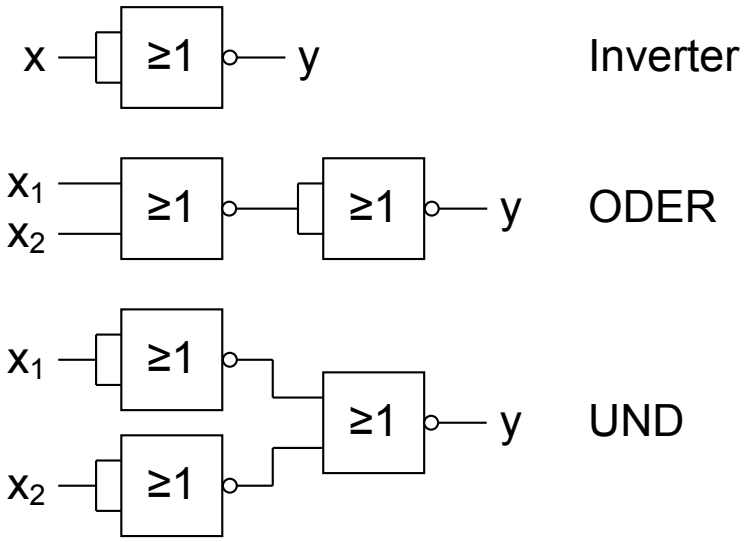
Jede Technologie, die zur Realisierung von Funktionen der Schaltalgebra eingesetzt werden soll, muss mindestens die Operatoren eines Basissystems realisieren!

NAND- und NOR-Gatter

NAND- und NOR-Gatter stellen beide ein Basissystem der Schaltalgebra dar.
Jede Schaltfunktion lässt sich nur aus diesen Gattertypen aufbauen.

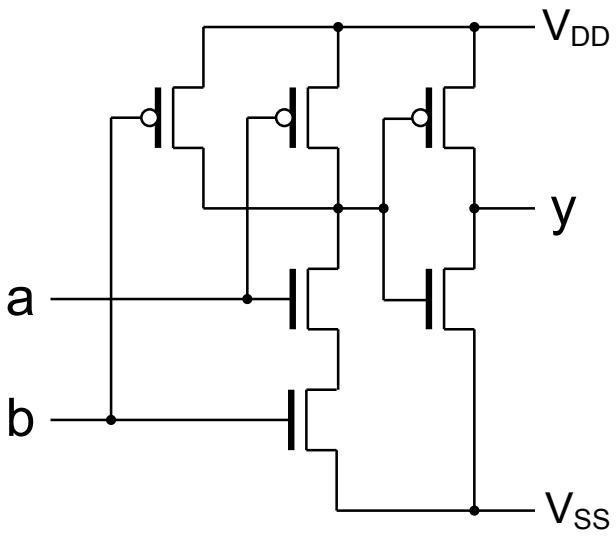


Basissystem NAND

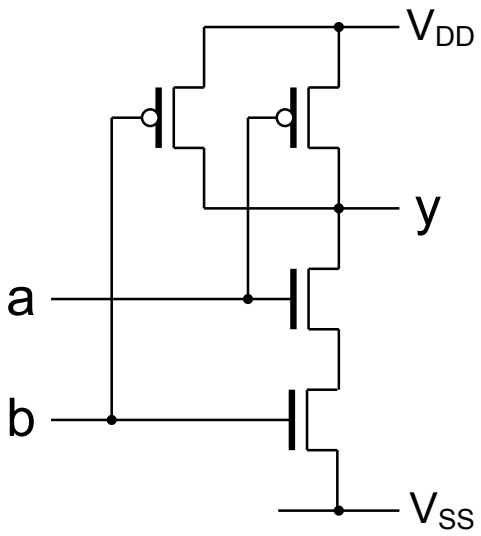


Basissystem NOR

CMOS-Technologie



UND



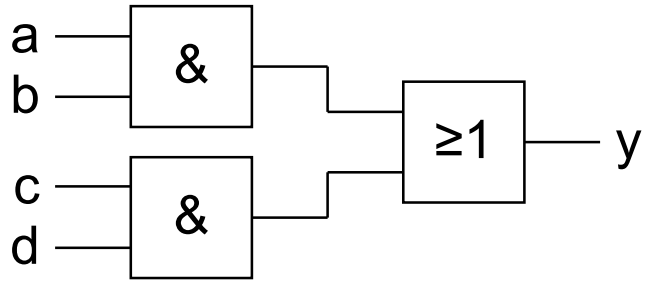
NAND

In der CMOS-Technologie benötigen NAND-Gatter weniger Transistoren als UND-Gatter. Sie verbrauchen daher weniger Chip-Fläche und sind schneller als UND-Gatter. Dasselbe gilt für NOR-Gatter im Vergleich zu ODER-Gattern.
Deshalb werden Schaltungen in CMOS-Technologie durch NAND/NOR-Schaltungen realisiert.

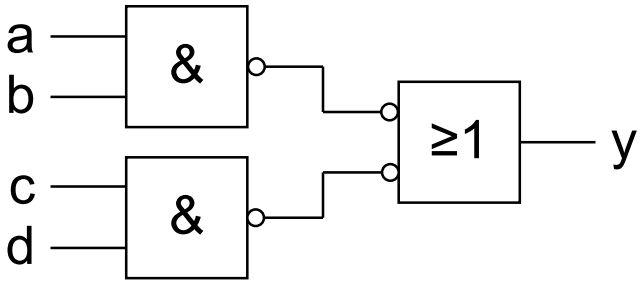
Abbildung auf NAND

Schaltnetze, welche in UND-ODER-Form vorliegen, können leicht in ein Schaltnetz aus NAND-Gattern abgebildet werden:

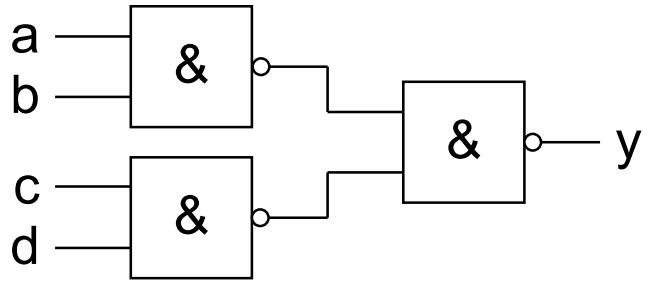
Beispiel: f(a, b, c, d) = (a * b) + (c * d)



1. Schritt: Einfügen von Invertern zwischen UND- und ODER-Gattern

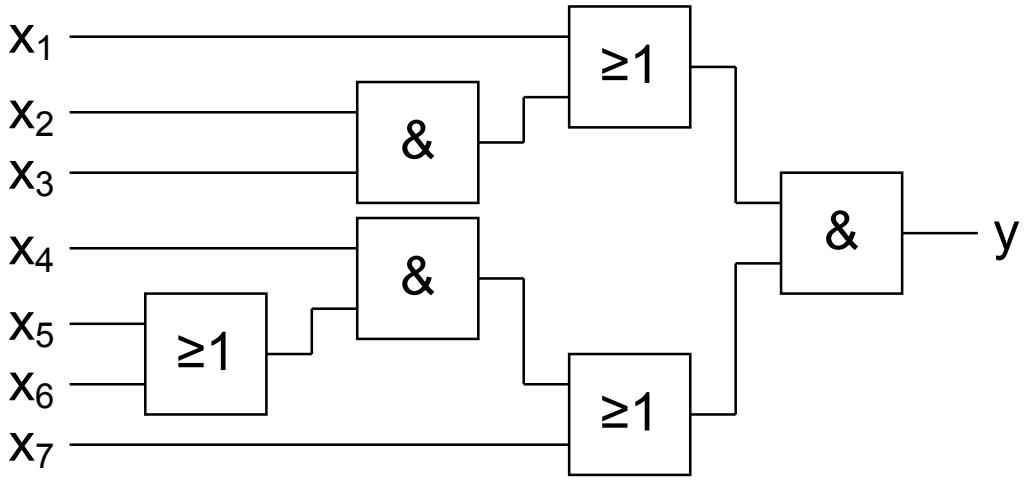


2. Schritt: Umwandlung des ODER-Gatters in NAND-Gatter

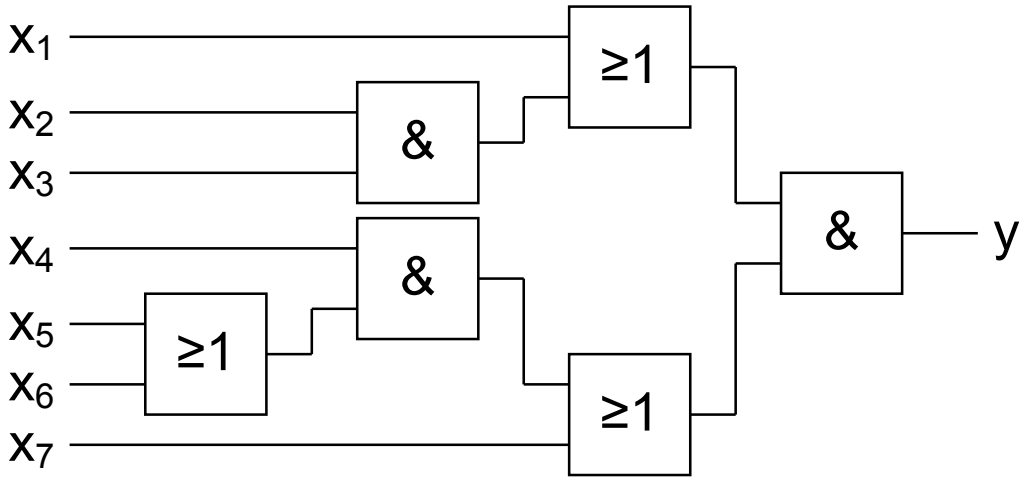


mehrstufige NAND Schaltnetze

Beispiel: $y = f(x_1, x_2, x_3, x_4, x_5, x_6) = [x_1 + (x_2 * x_3)] * [(x_4 * (x_5 + x_6)) + x_7]$

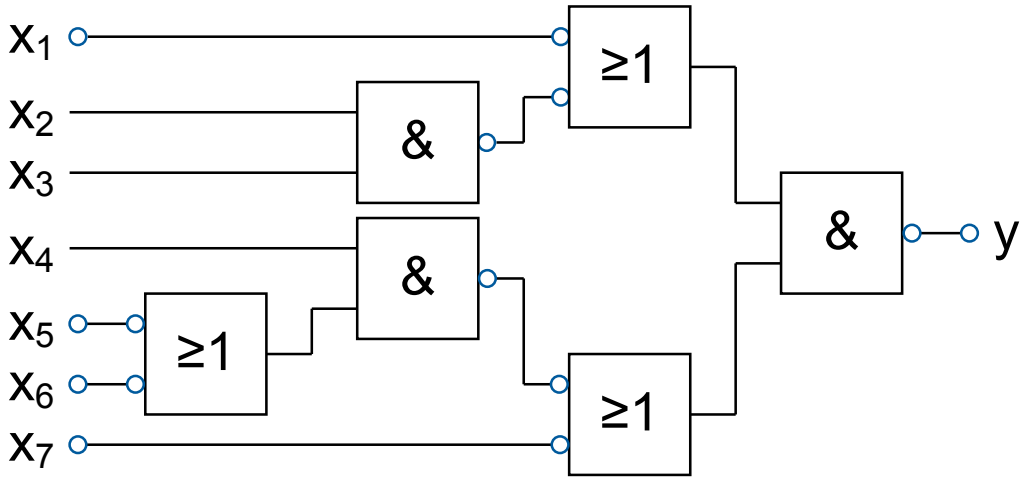


1. Schritt:
Einfügen von Invertern

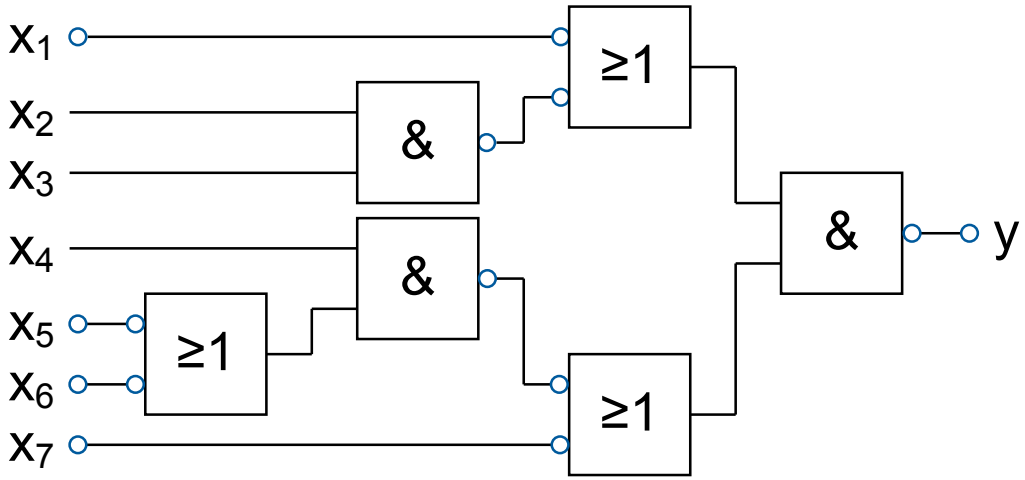


mehrstufige NAND Schaltnetze

Beispiel: $y = f(x_1, x_2, x_3, x_4, x_5, x_6) = [x_1 + (x_2 * x_3)] * [(x_4 * (x_5 + x_6)) + x_7]$

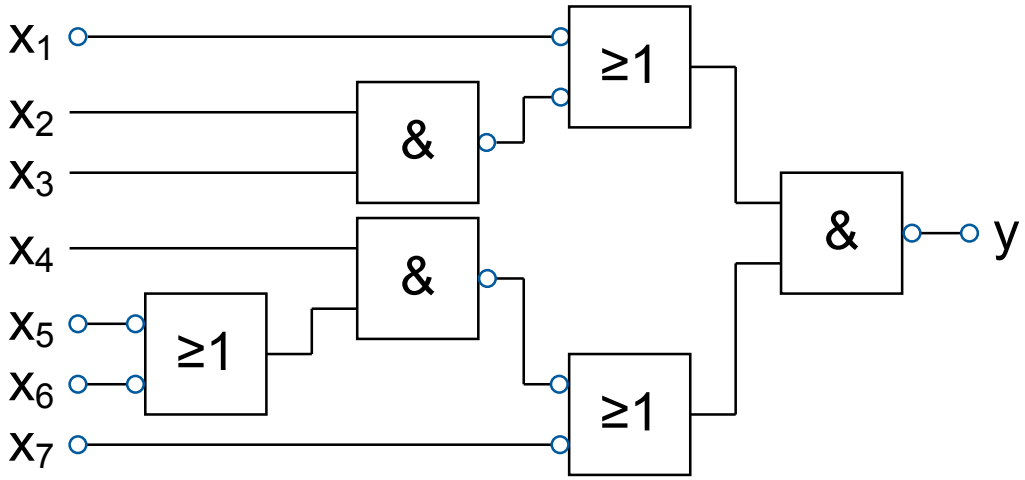


2. Schritt:
Umwandeln in NAND

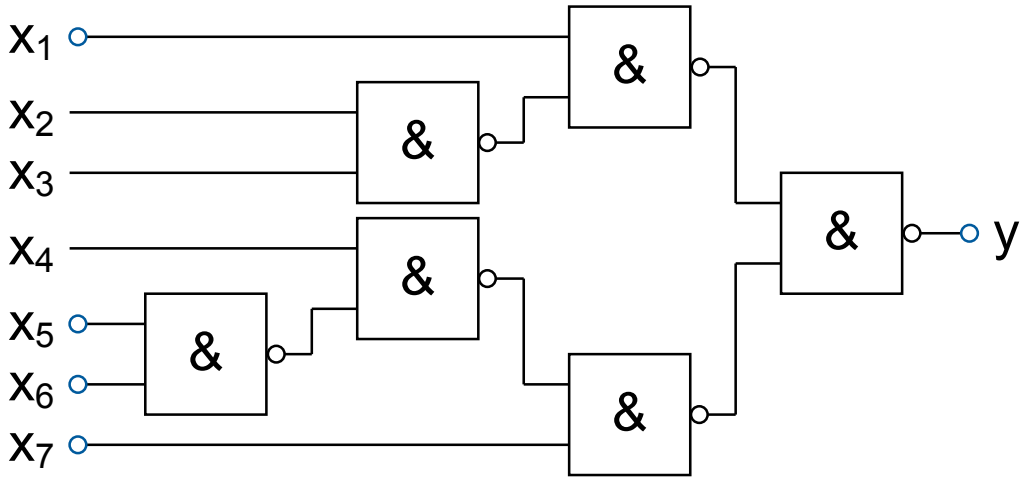


mehrstufige NAND Schaltnetze

Beispiel: $y = f(x_1, x_2, x_3, x_4, x_5, x_6) = [x_1 + (x_2 * x_3)] * [(x_4 * (x_5 + x_6)) + x_7]$



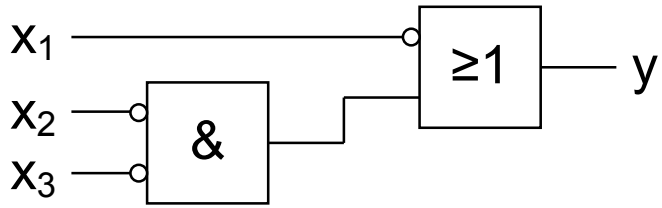
Endergebnis:



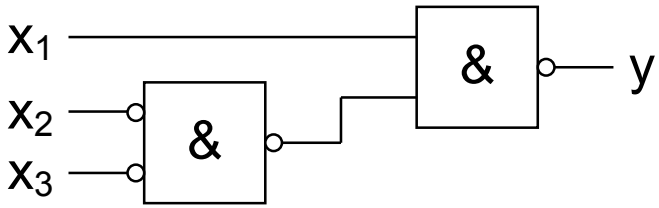
gemischte NAND/NOR-Schaltnetze

Beispiel:

$y = f(x_1, x_2, x_3) = x_1' + (x_2' * x_3')$

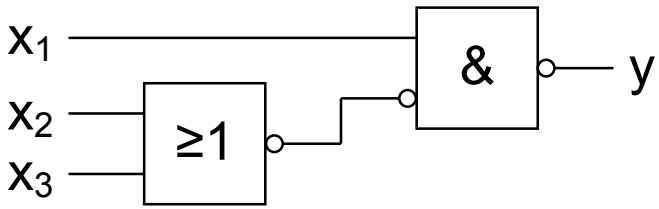


$y = (x_1' + (x_2' * x_3')) = (x_1' + (x_2' * x_3'))'' = (x_1'' * (x_2' * x_3')')' = (x_1 * (x_2' * x_3')')'$



2 NAND2 + 2 Inverter

gemischte NAND/NOR Realisierung:

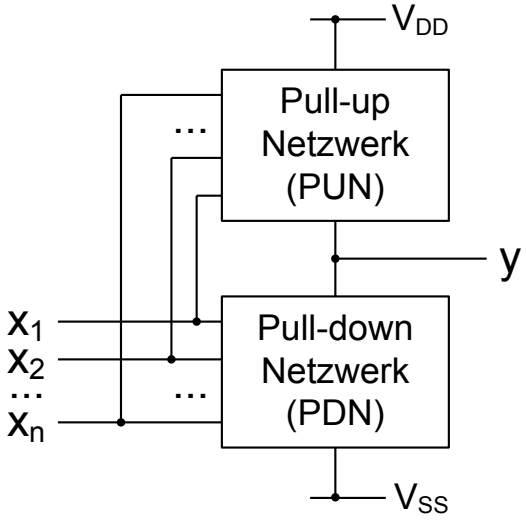


1 NAND2 + 1 NOR2 + 1Inverter

=> geringere Fläche

Komplexgatter

Erinnerung: CMOS-Gatter

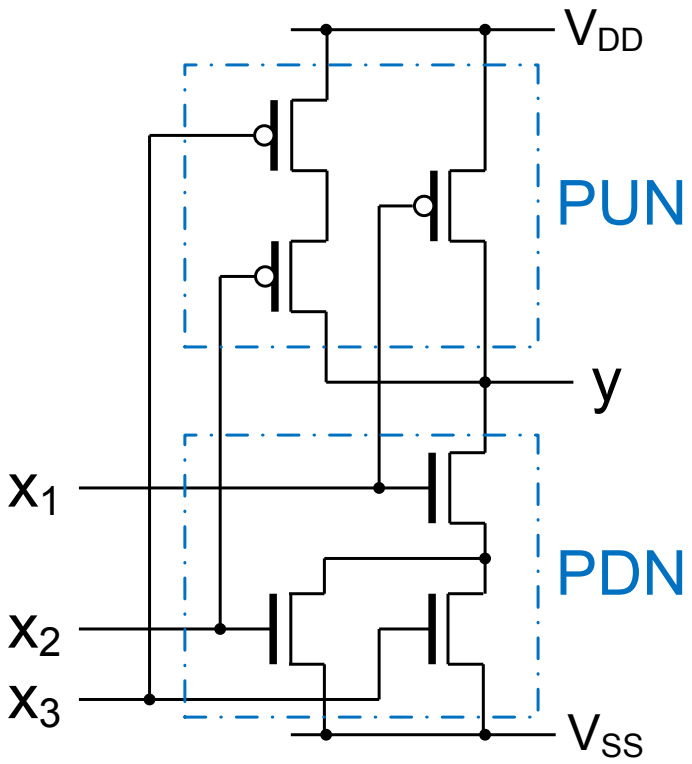


vorheriges Beispiel:

$y = f(x_1, x_2, x_3) = x_1' + (x_2' * x_3')$

PUN: $y = x_1' + (x_2' * x_3')$

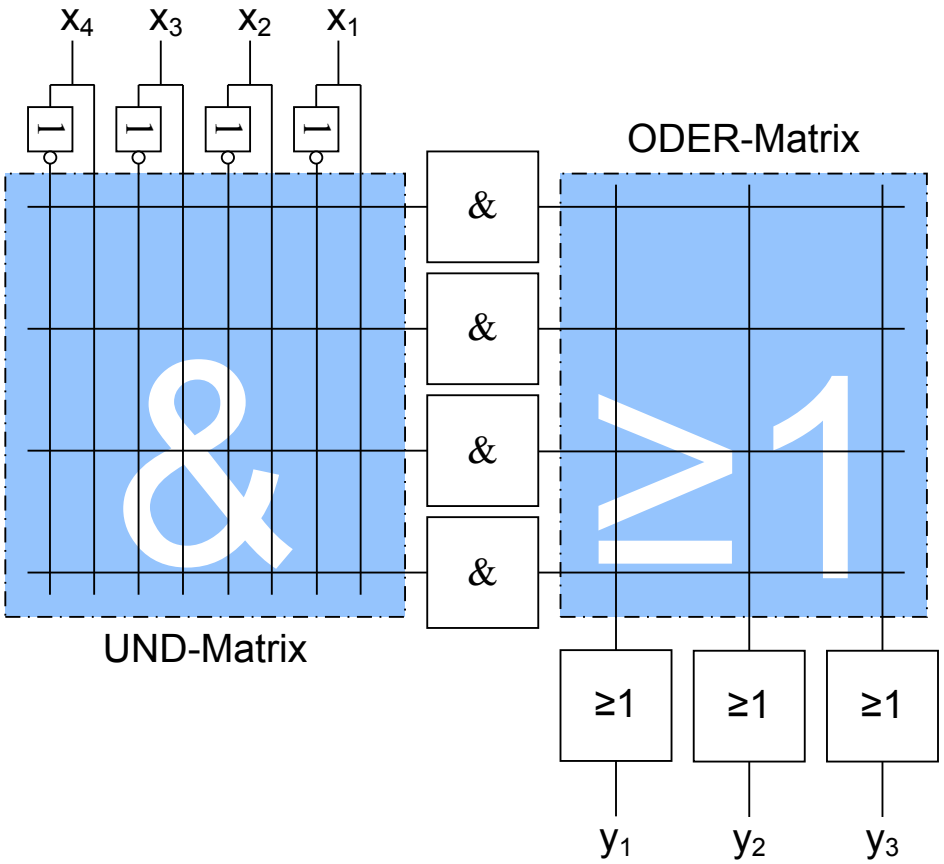
PDN: $y' = (x_1' + (x_2' * x_3'))'$
$= x_1'' * (x_2' * x_3')'$
$= x_1 * (x_2 + x_3)$



- geringere Fläche, da weniger Transistoren (6 statt 10)
- schneller, da weniger Transistoren und kürzere Verbindungsleitungen

Programmierbare Logikbausteine

Programmable Logic Array(PLA)

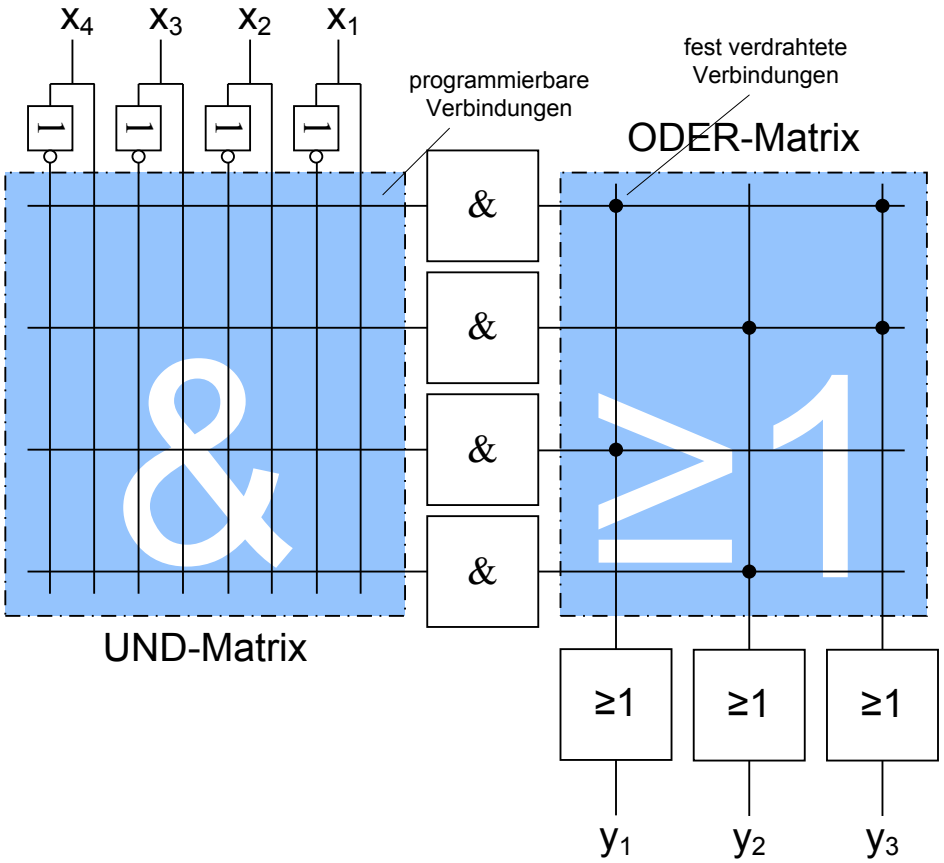


- durch Programmierung lassen sich an den Knoten Verbindungen herstellen (X) oder löschen
- Struktur zur direkten Umsetzung der disjunktiven Minimalform
- sind UND-Matrix und ODER-Matrix programmierbar spricht man von Programmable Logic Arrays (PLAs)

$y_1 = (x_2' * x_3' * x_4') + (x_1' * x_3' * x_4')$ $y_2 = (x_3' * x_4')$

Programmierbare Logikbausteine

Programmable Array Logic (PAL)

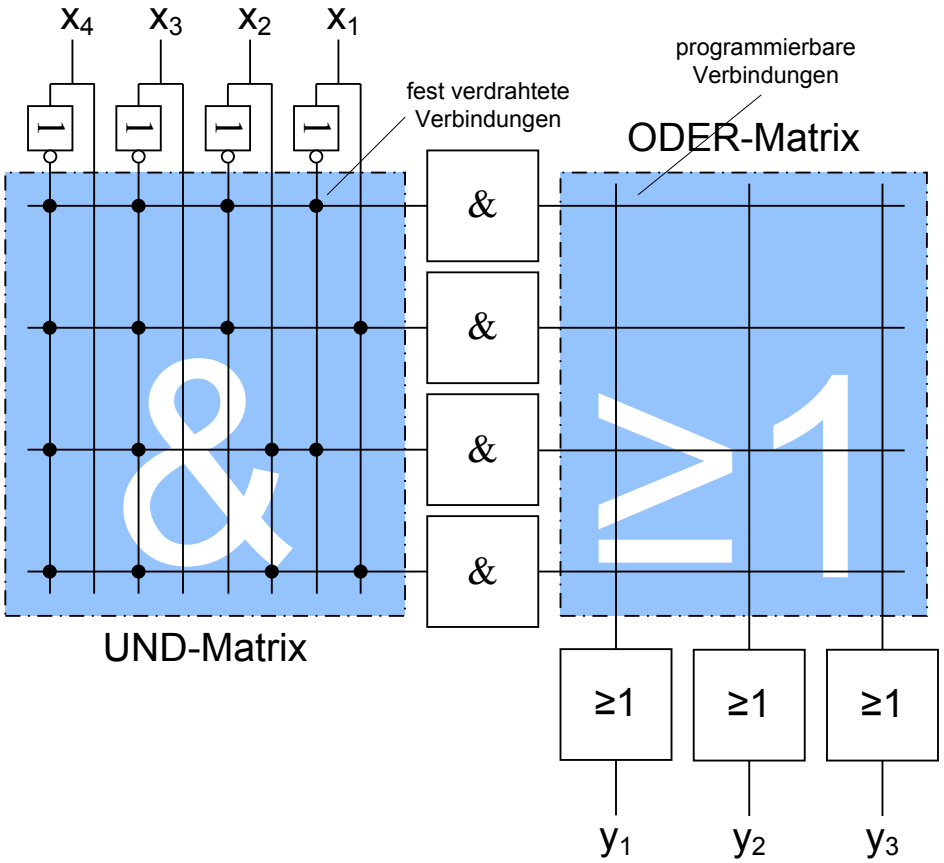


- Verbindungen von ODER-Matrix fest vorgegeben (●)
- Verbindungen von UND-Matrix sind programmierbar (X)

$y_1 = (x_2' * x_3' * x_4') + (x_1' * x_3' * x_4')$ $y_2 = (x_3' * x_4')$

PROM

Programmable Read Only Memory (PROM)



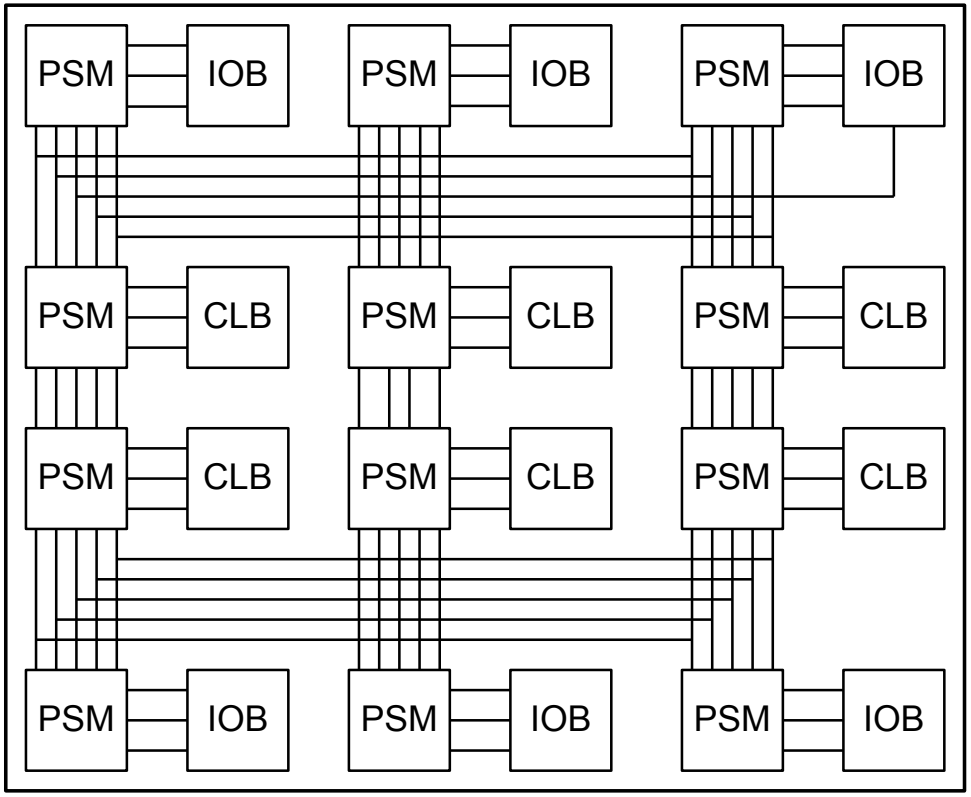
- Verbindungen von UND-Matrix fest vorgegeben (●)
- Verbindungen von ODER-Matrix sind programmierbar (X)

PROMs haben normalerweise anderen Verwendungszweck – können aber zur Realisierung von Schaltfunktionen benutzt werden!

$y_1 = (x_2' * x_3' * x_4') + (x_1' * x_3' * x_4')$ $y_2 = (x_3' * x_4')$

FPGA

Field-Programmable Gate Array (FPGA)



Prinzipaufbau eines FPGA:

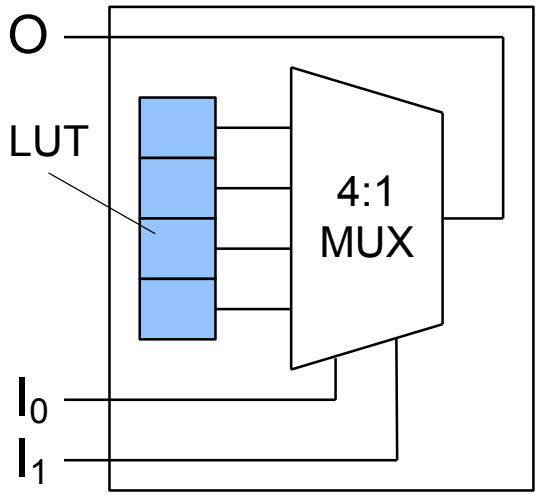
- CLB: Configurable Logic Block
  Bsp: Xilinx Virtex FPGAs
  XC2VP30 13.696 CLBs
  XC5VLX110T 17.280 CLBs
- PSM: Programmable Switch Matrix
- IOB: Input-/Output-Blocks

FPGA

Configurable Logic Block:

- enthält programmierbare Look-Up Table (LUT) zur Realisierung von Schaltfunktionen
- Speicherelemente (FlipFlops)

vereinfachter Aufbau:



Beispiel:

$y = f(x_1, x_2) = (x_1' * x_2') + (x_1 * x_2)$

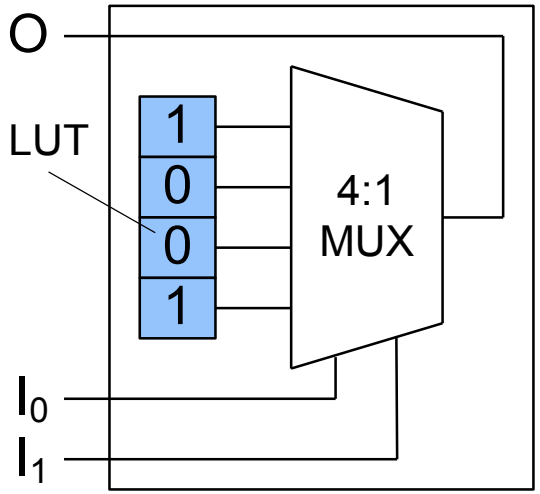
x_2	x_1	y
0	0	1
0	1	0
1	0	0
1	1	1

FPGA

Configurable Logic Block:

- enthält programmierbare Look-Up Table (LUT) zur Realisierung von Schaltfunktionen
- Speicherelemente (FlipFlops)

vereinfachter Aufbau:



Beispiel:

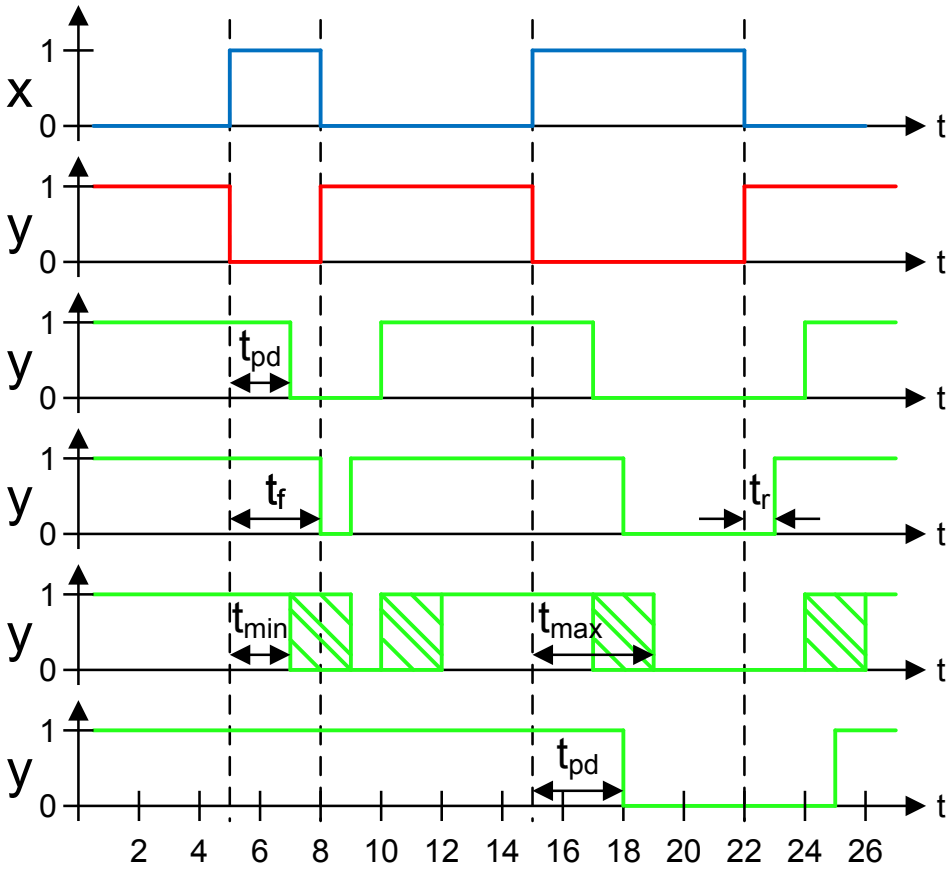
$y = f(x_1, x_2) = (x_1' * x_2') + (x_1 * x_2)$

x_2	x_1	y
0	0	1
0	1	0
1	0	0
1	1	1

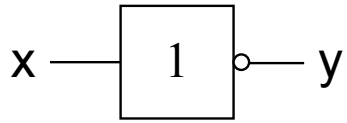
Mit LUT alle Funktionen von 2 Variablen darstellbar.
Üblich bei heutigen FPGAs: LUTs mit 4-6 Eingängen

Verzögerungsmodelle

Bisher nur ideale Gatter betrachtet - reale Gatter benötigen eine gewisse Zeit bis sich die Änderungen des Eingangs am Ausgang bemerkbar machen! Um diesen Effekt zu modellieren gibt es verschiedene sog. Verzögerungsmodelle.



Eingangssignal



Ausgangssignal unverzögert

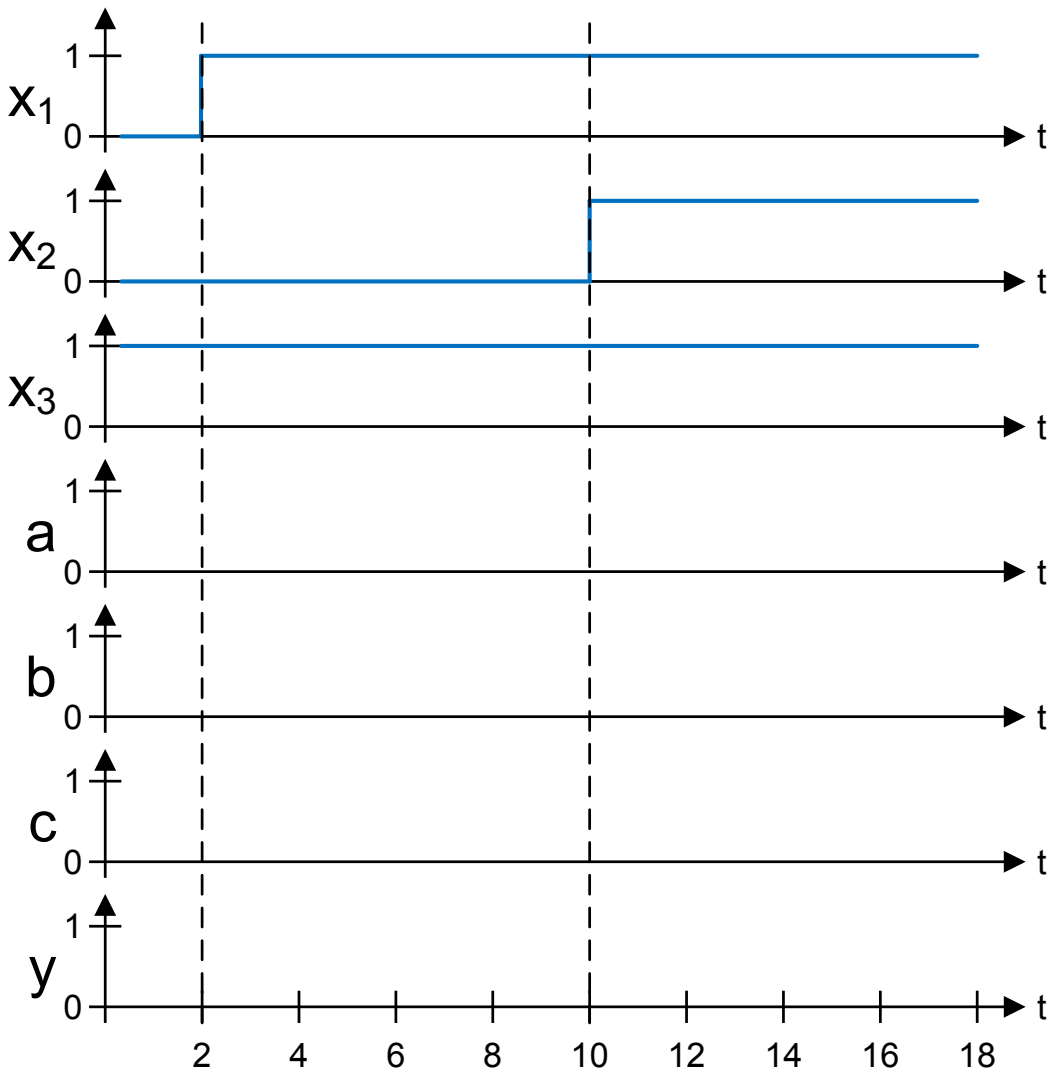
Einheitsverzögerung (unit delay) $t_{pd} = 2$

Anstiegszeit (rise delay) $t_r = 1$,
Abfallzeit (fall delay) $t_f = 3$

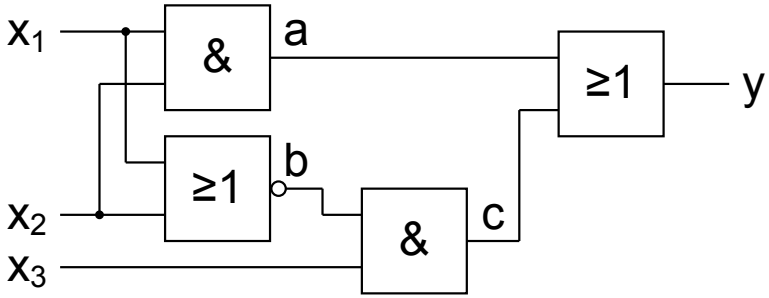
Intervall $t_{min} = 2$, $t_{max} = 4$ (ambiguity delay)

Einheitsverzögerung $t_{pd} = 3$ mit
Totzeit $t_T < 3$ (inertial delay)

verzögerungsbehaftete Simulation



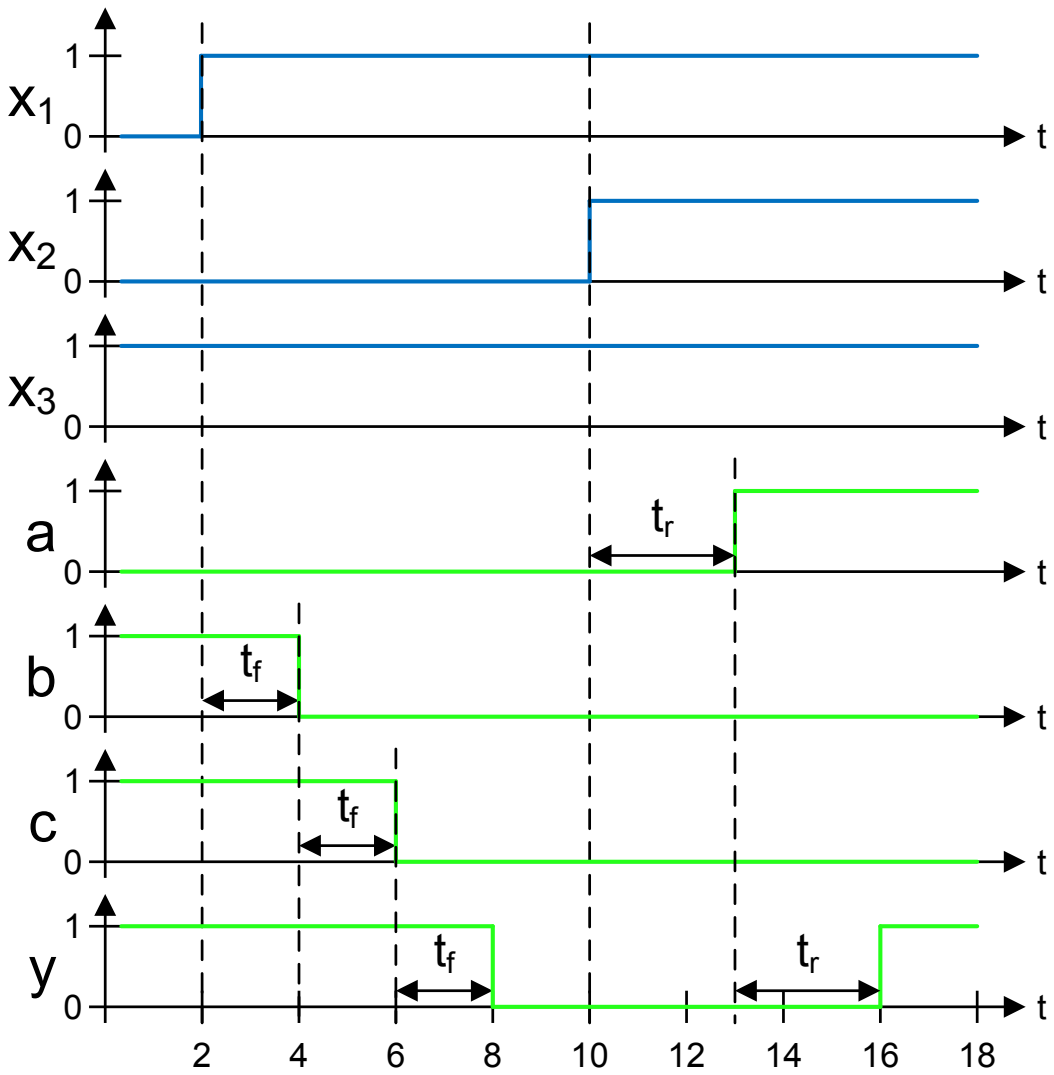
Beispiel:



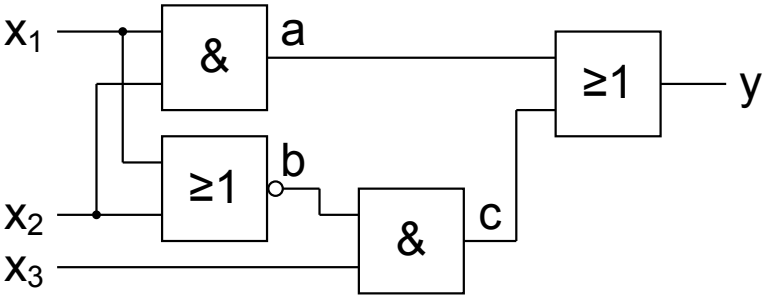
Verzögerungsmodell:
rise delay $t_r = 3$
fall delay $t_f = 2$

Neben der Bestimmung der Werte der Signale auch Bestimmung des Zeitpunktes der Signalwechsel entsprechend den Verzögerungsmodellen.

verzögerungsbehaftete Simulation



Beispiel:

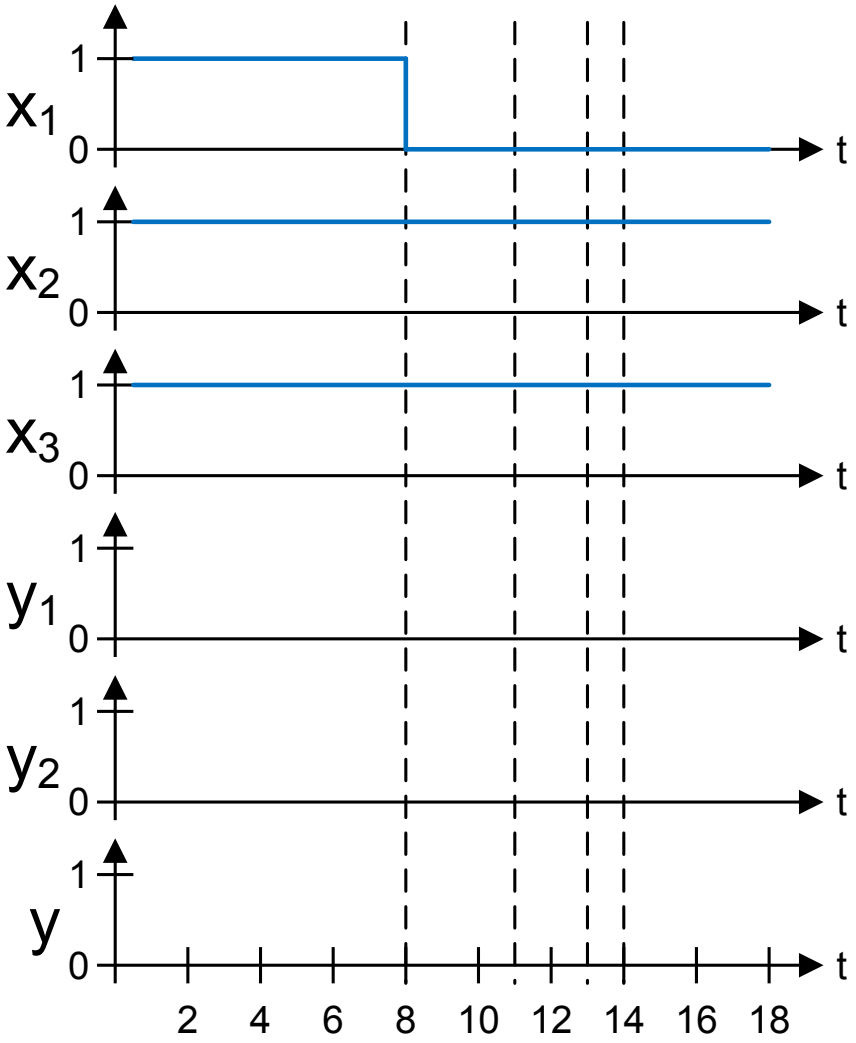


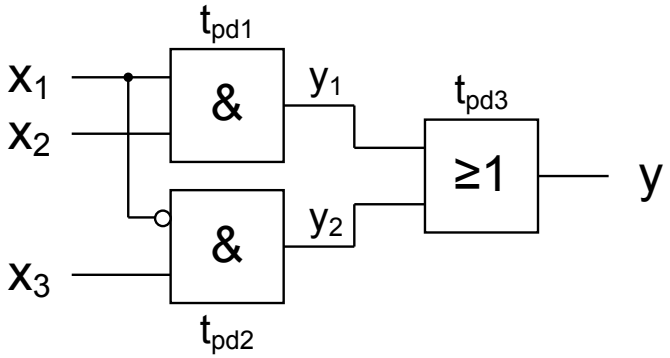
Verzögerungsmodell:
rise delay $t_r = 3$
fall delay $t_f = 2$

Neben der Bestimmung der Werte der Signale auch Bestimmung des Zeitpunktes der Signalwechsel entsprechend den Verzögerungsmodellen.

Störimpulse

Beispiel:

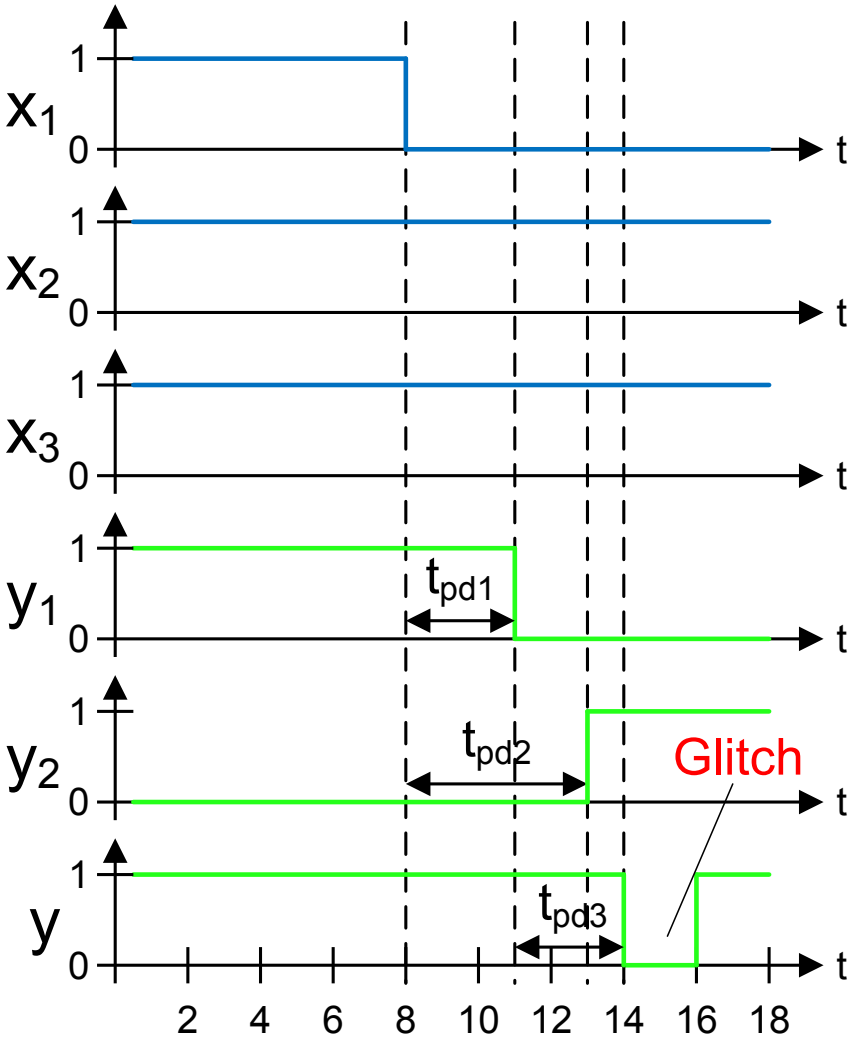


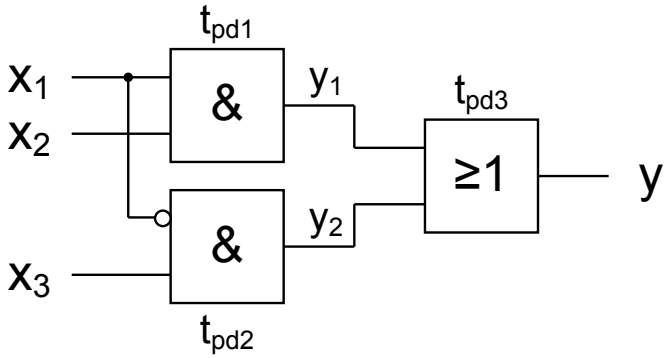


$t_{pd2} > t_{pd1}$

Störimpulse

Beispiel:





$t_{pd2} > t_{pd1}$

Auftreten eines Störimpulses (Glitch) wegen der unterschiedlichen Verzögerungen der Gatter!

Glitches und Hazards

Definition 9.8 (Störimpuls)
Ein kurzzeitig (temporär) auftretender Signalwechsel wird als *Störimpuls* (Glitch) bezeichnet.

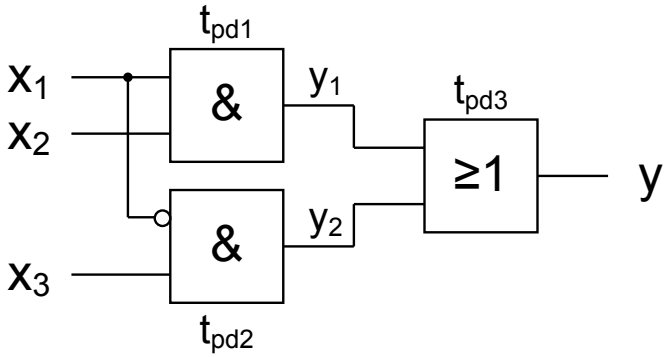
Definition 9.9 (statischer 1-Hazard)
Von einem *statischen 1-Hazard* spricht man, wenn es für ein Schaltnetz zwei Eingangskombinationen gibt, die sich nur in einer Eingangsvariablen unterscheiden und beide zu dem Ausgangswert 1 führen, und die Möglichkeit besteht, dass aufgrund eines Wechsels dieser Eingangsvariablen ein kurzzeitiger 0 Störimpuls am Ausgang entstehen kann.

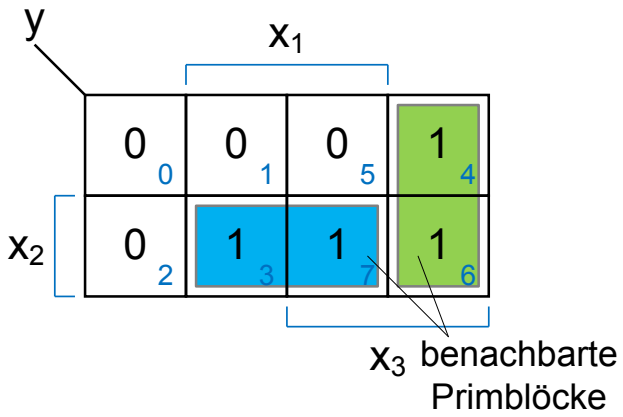
Definition 9.10 (statischer 0-Hazard)
Von einem *statischen 0-Hazard* spricht man, wenn es für ein Schaltnetz zwei Eingangskombinationen gibt, die sich nur in einer Eingangsvariablen unterscheiden und beide zu dem Ausgangswert 0 führen, und die Möglichkeit besteht, dass aufgrund eines Wechsels dieser Eingangsvariablen ein kurzzeitiger 1 Störimpuls am Ausgang entstehen kann.

Detektion von statischen Hazards

Die Detektion von statischen Hazards eines Schaltnetzes kann mit KV-Diagrammen geschehen. Die Möglichkeit einen Glitch zu produzieren besteht immer dann, wenn zwei Implikantenblöcke überlappungsfrei aneinandergrenzen.

Beispiel:



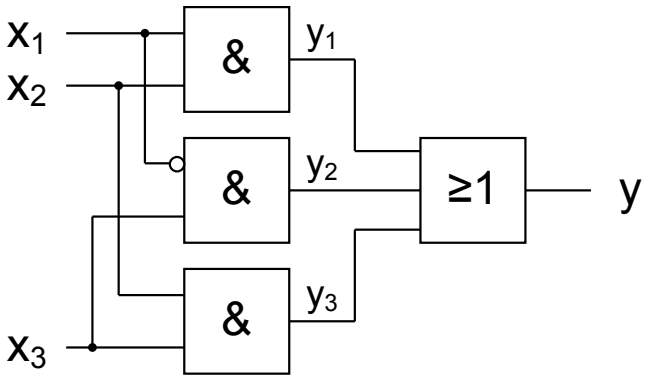


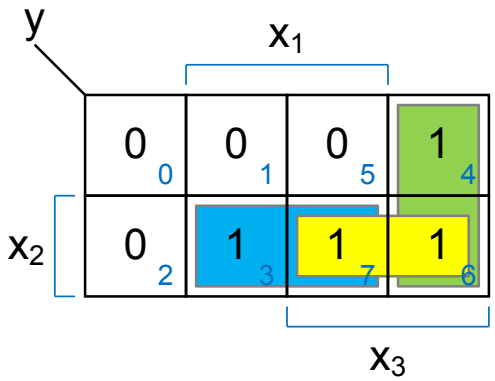
Der Ausgangswert ergibt sich aus der Disjunktion der beiden Implikanten. Bei einem Wechsel von x_1 wechseln auch die "aktiven" Implikanten und damit das Gatter, welches aktuell den Ausgangswert 1 liefert.

Behebung von statischen Hazards

Damit keine statischen Hazards vorliegen, müssen sich benachbarte Implikantenblöcke überlappen. Dies kann durch Einfügen eines weiteren Implikantenblocks erreicht werden.

Beispiel:



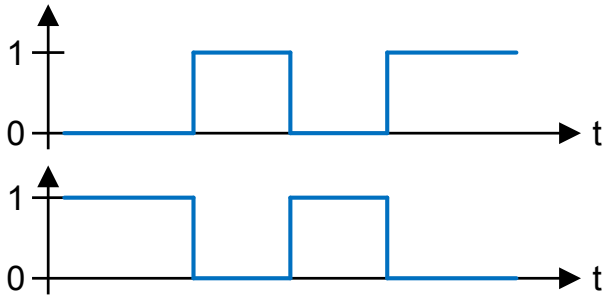


Unabhängig von x_1 liefert y_3 jetzt den Wert 1, dadurch kann kein Glitch mehr auftreten. Die Schaltung ist Hazard-frei.

Die Schaltung liegt nicht mehr in der minimalen Form vor!

dynamische Hazards

Definition 9.11 (dynamischer Hazard)
Ein *dynamischer Hazard* liegt vor, wenn bei einem Wechsel der Eingangssignale die Möglichkeit besteht, dass der Ausgangswert sich mehrfach ändert bis ein stabiler Wert anliegt. Treten die Fluktuation beim einem Wechsel vom Ausgangswert 0 auf 1 auf, so spricht man von einem dynamischen 1-Hazard, ansonsten von einem dynamischen 0-Hazard.



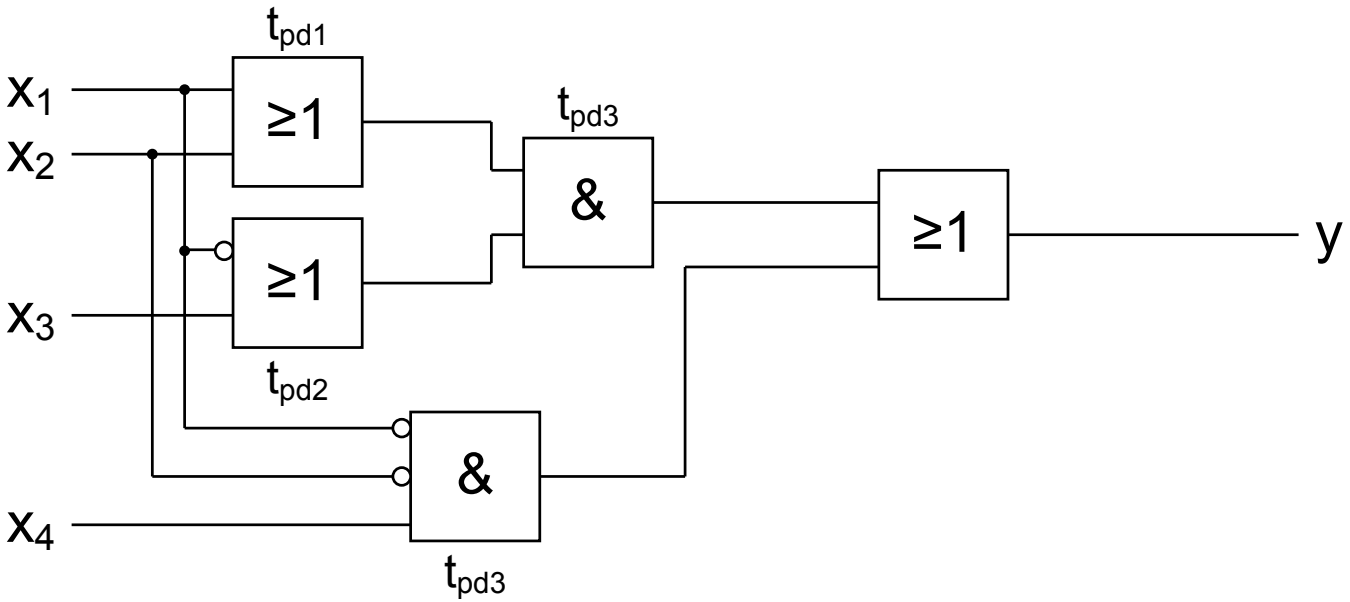
Glitch, produziert durch dynamischen 1-Hazard

Glitch, produziert durch dynamischen 0-Hazard

Dynamische Hazards treten nur in mehrstufigen Schaltnetzen auf. In zweistufigen Schaltznetzen können sie nicht auftreten!

dynamische Hazards

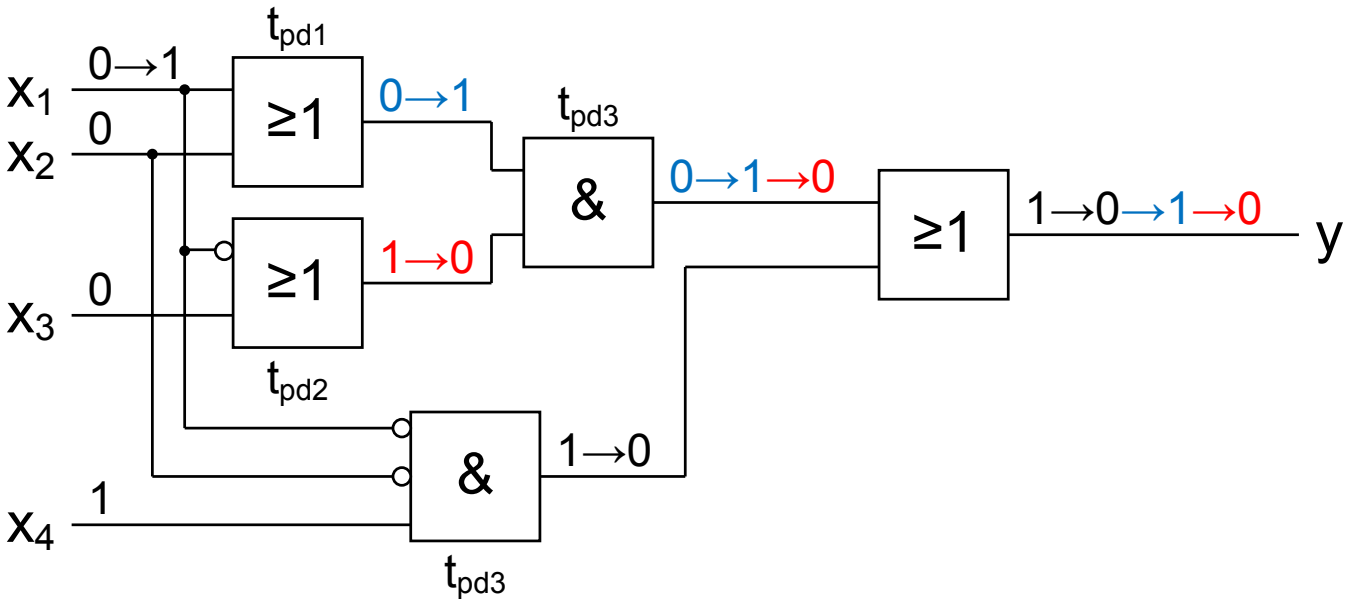
Beispiel:



$$t_{pd2} > t_{pd1} > t_{pd3}$$

dynamische Hazards

Beispiel:



$$t_{pd2} > t_{pd1} > t_{pd3}$$

Glitches

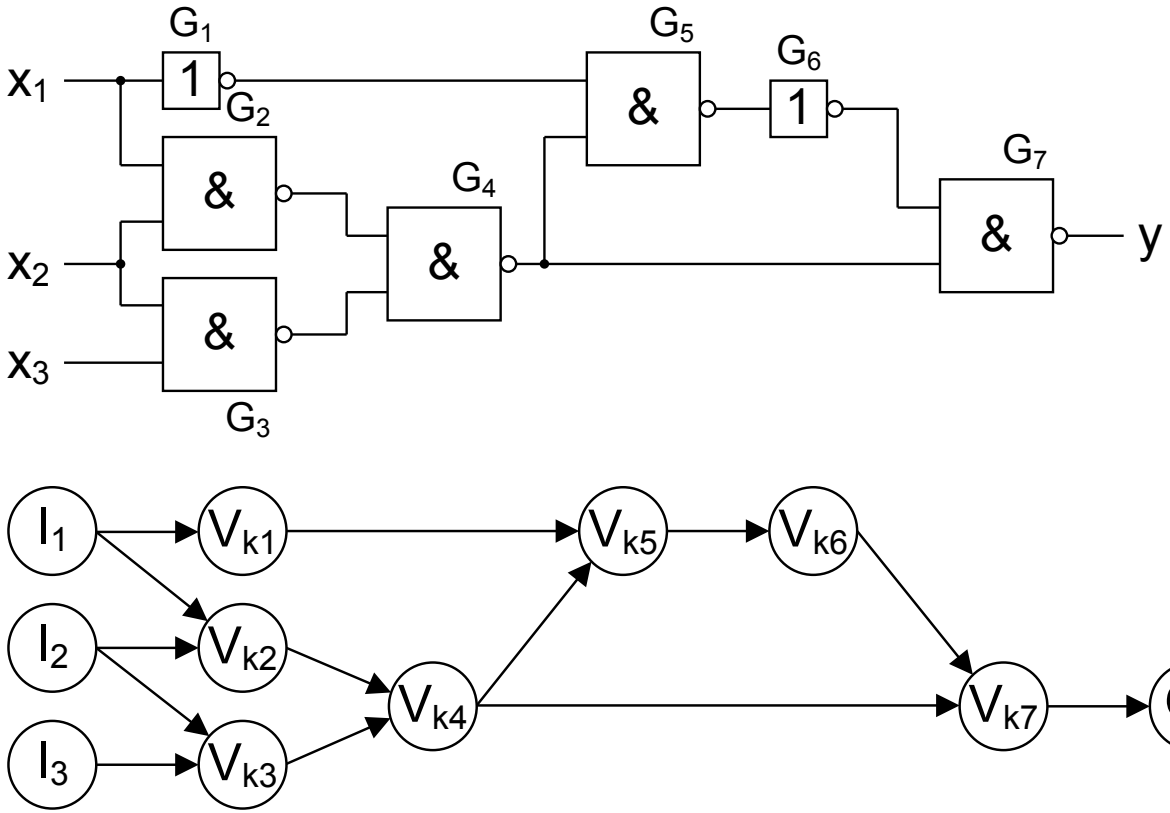
- Glitches erschweren die Entwicklung von asynchronen Schaltungen
- Glitches erhöhen die Leistungsaufnahme einer digitalen Schaltung (häufig ca. 30 – 50 %)
- Bei synchronen Schaltungen oft vernachlässigbar! (siehe Kapitel 5)

Für synchrone Schaltungen ist maximale Verzögerung wichtig => längster kombinatorischer Pfad

Schaltnetzgraph

Definition 9.12 (Schaltnetzgraph)
Ein *Schaltnetzgraph* G := (V, E) ist ein gerichteter Graph mit den Knoten V = $\{v_0, v_1, ..., v_n\}$ und den Kanten $E \subset V \times V$.
$V := V_k \cup I \cup O$ V_k : kombinatorische Knoten, I : Eingänge, O: Ausgänge

Beispiel:



Pfadanalyse

Definition 9.13 (Pfad)
Ein *Pfad* T ist eine Folge von Knoten v, die durch Kanten verbunden sind.

$T := \langle v_0, v_1, ..., v_i \rangle$ mit $\langle v_{j-1}, v_j \rangle \epsilon E$

Signalverzögerung einer Schaltung:

Im Allgemeinen gilt:
Zur Abschätzung der Signalverzögerung einer Schaltung untersucht man den längsten Pfad. Der längste Pfad ist derjenige Pfad, entlang dessen die Summe aller Verzögerungen (Gatter + Leitungen) maximal ist.

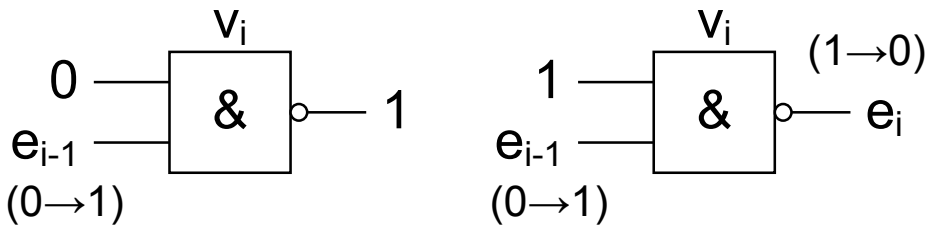
Allerdings ist nicht jeder logische Pfad an der Auswirkung von Eingang auf den Ausgang beteiligt. Dies muss bei der Untersuchung des längsten Pfads berücksichtigt werden → Sensibilisierbarkeit

Sensibilisierbarkeit

Definition 9.14 (Ereignis)
Ein *Ereignis* e_i ist ein Signalwechsel ($1 \rightarrow 0$, $0 \rightarrow 1$) am Knoten v_i.
Ein Ereignis e_{i-1} pflanzt sich fort, wenn e_i direkte Folge von e_{i-1} ist.

Definition 9.15 (Sensibilisierbarer Pfad)
Ein Pfad heißt *sensibilisiert*, wenn sich ein Ereignis am Eingang auf den Ausgang fortpflanzen kann.
Ein Pfad heißt *sensibilisierbar*, wenn es eine Eingabebelegung des Schaltnetzes gibt, so dass der Pfad sensibilisiert wird.

Beispiel: e_{i-1} blockiert: e_{i-1} sensibilisiert:



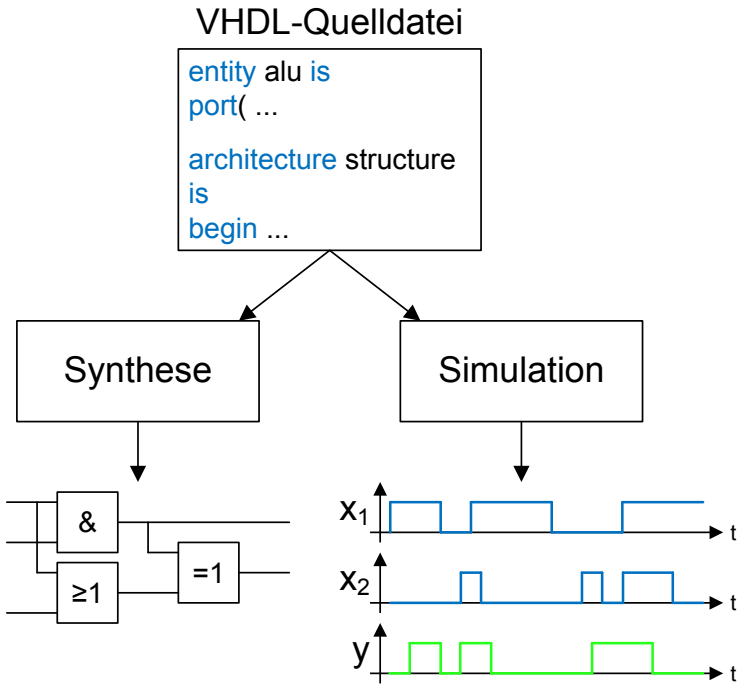
VHDL

VHDL steht für VHSIC Hardware Description Language. VHDL ist somit eine Sprache zur Beschreibung von digitalen Hardwareschaltungen.

VHDL erlaubt die Unterteilung des Systems in einzelne Untereinheiten (Komponenten), ähnlich den Modulen oder Klassen in Programmiersprachen, die getrennt modelliert werden können. Üblicherweise gibt es für jede Komponente eine VHDL-Quelldatei mit einer entity und architecture.

VHDL wird hauptsächlich für 2 Aufgaben eingesetzt:

- Synthese einer Schaltung zur anschließenden Implementierung
- Simulation einer Schaltung zur Verifikation der Funktionsweise



VHDL

ENTITY (Schnittstellenbeschreibung)

Beschreibung der Schnittstelle, d.h. Eingabe- und Ausgabeports der Komponente.

Beispiel: Schaltnetz mit 3 Eingängen und 1 Ausgang


```
entity schaltnetz is
port( a  : in  bit;
      b  : in  bit;
      c  : in  bit;
      y  : out bit);
end entity schaltnetz;
```

Kontrollfragen

- Führen Sie Unterschiede und Gemeinsamkeiten der Minimierung mit KV-Diagrammen im Vergleich zum QMC-Verfahren auf.
- Welche Eigenschaft horizontal oder vertikal direkt benachbarter Felder von KV-Diagrammen wird für die Minimierung / Zusammenfassung ausgenutzt?
- Welche Größe und Form müssen größere zusammengefasste Blöcke im KV-Diagramm aufweisen?
- Warum sind auch Zusammenfassungen über die Ränder des KV-Diagramms hinaus erlaubt?
- Definieren Sie die Begriffe Implikant, Primimplikant, Kernimplikant, wesentlicher Implikant. Was sind die Gemeinsamkeiten, was sind die Unterschiede?
- Was ist eine minimale Summe? Kann es mehrere minimale Summen derselben Schaltfunktion geben?
- Wie werden Funktionen mit „don‘t cares“ bei der Minimierung mit KV und QMC behandelt?
- Erläutern Sie die Vorgehensweise in den zwei Schritten des QMC-Verfahrens. Vergleichen Sie diese mit der Minimierung mit KV-Diagrammen.
- Wie identifiziert man im QMC-Verfahren die Primimplikanten?
- Wozu dienen die Streichungen im 2. Schritt (Überdeckungstabelle) von QMC?
- Endet das Verfahren immer mit einer leeren Tabelle? Was wenn nicht?

Kontrollfragen

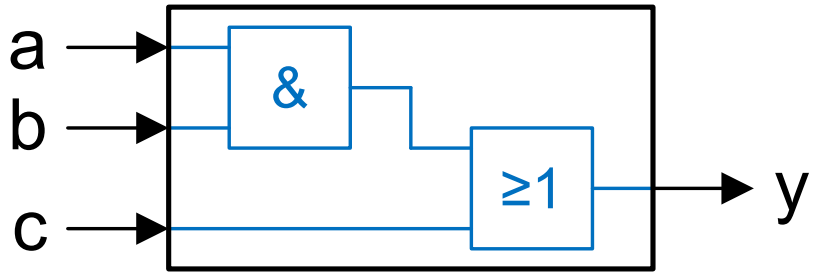
- Was bedeutet „Bündelminimierung“, „mehrstufige Minimierung“ und „heuristische Minimierung“?
- Identifizieren Sie ein vollständiges Operatorensystem, das nicht in der Vorlesung behandelt wurde. Z.B. eines, das ohne (N)AND und (N)OR auskommt.
- Warum ist die Technologieabbildung auf NAND und NOR besonders wichtig?
- Was ist der Unterschied eines Komplexgatters im Vergleich zu einem Grundgatter und im Vergleich zu einem Schaltnetz mit mehreren Grundgattern?
- Wie unterscheiden sich PLAs, PALs, ROMs und FPGAs? Wie könnte man die Technologieabbildung für diese Zieltechnologien geschickt durchführen?
- Welche Verzögerungsmodelle gibt es?
- Wodurch entstehen Störimpulse (Glitches)?
- Was ist der Unterschied zwischen Glitches und Hazards?
- Was ist der Unterschied zwischen statischen und dynamischen Hazards?
- Wie bestimmt man die Verzögerungszeit eines Schaltnetzes?
- Welche Rolle spielt dabei die Sensibilisierbarkeit von Pfaden?
- Recherchieren Sie gängige Algorithmen für die Bestimmung von längsten Pfaden.

ANHANG

ARCHITECTURE (Funktionsbeschreibung)

Beschreibung des Verhaltens oder des Aufbaus einer Komponente. Zu einer entity kann es mehrere architectures geben – entsprechen verschiedenen Realisierungsvarianten der Komponente.

Beispiel: Schaltnetz für y = (a * b) + c




```
architecture dataflow of schaltnetz is
begin
        y <= (a and b) or c;
end architecture dataflow;
```

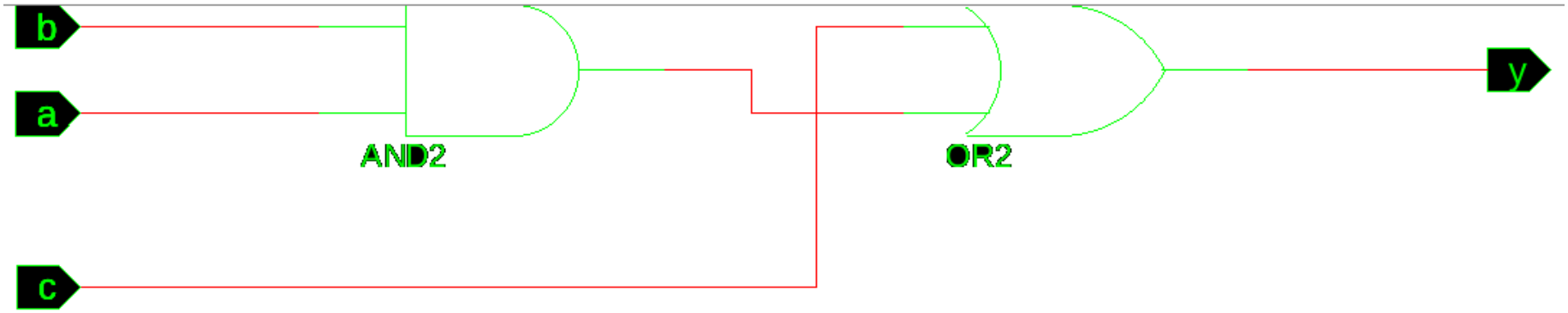
VHDL

Alternativ durch Strukturmodell:

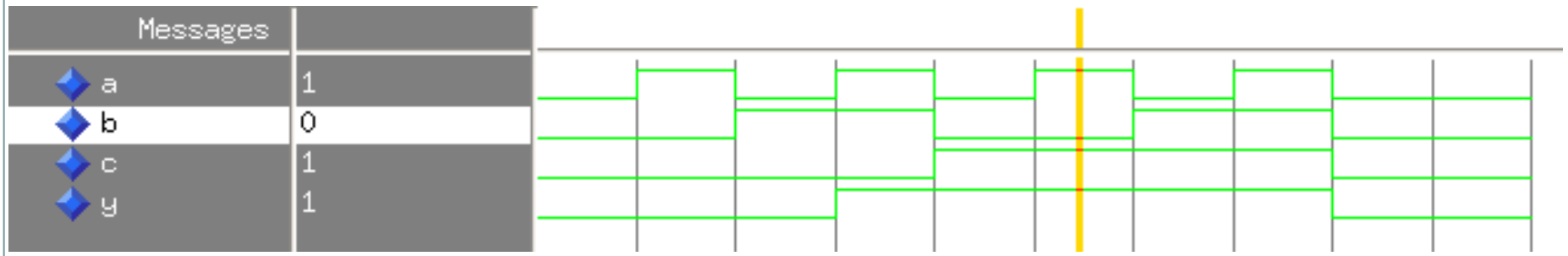
```
architecture structure of schaltnetz is
component and2 is
port( a, b  : in  bit;
      y     : out bit);
end component and2;
component or2 is
port( a, b  : in  bit;
      y     : out bit);
end component or2;
signal temp : bit;
begin
gate1:   component and2
         port map(a, b, temp);

gate2:   component or2
         port map(temp, c, y);
end architecture structure;
```

Syntheseergebnis XST:



Simulationsergebnis ModelSim:



VHDL

Beispiel: Schaltnetz mit mehreren Ausgängen und Modellierung von Einheits-Verzögerung mit Totzeit.

```
entity schaltnetz2 is
port( a,b,c : in  bit;
      x,y,z : out bit);
end entity schaltnetz2;

architecture dataflow of schaltnetz2 is
signal temp1, temp2, temp3 : bit;
begin
      temp1 <= a and b;
      temp2 <= temp1 xor c;
      temp3 <= c or d;
      x <= temp1 after 2 ns;
      y <= temp2 after 4 ns;
      z <= temp2 and temp3 after 6 ns;
end architecture dataflow;
```