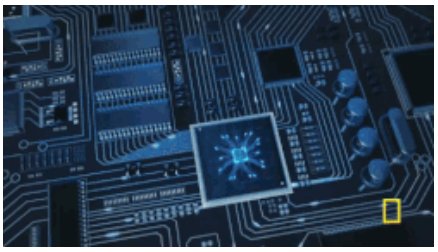
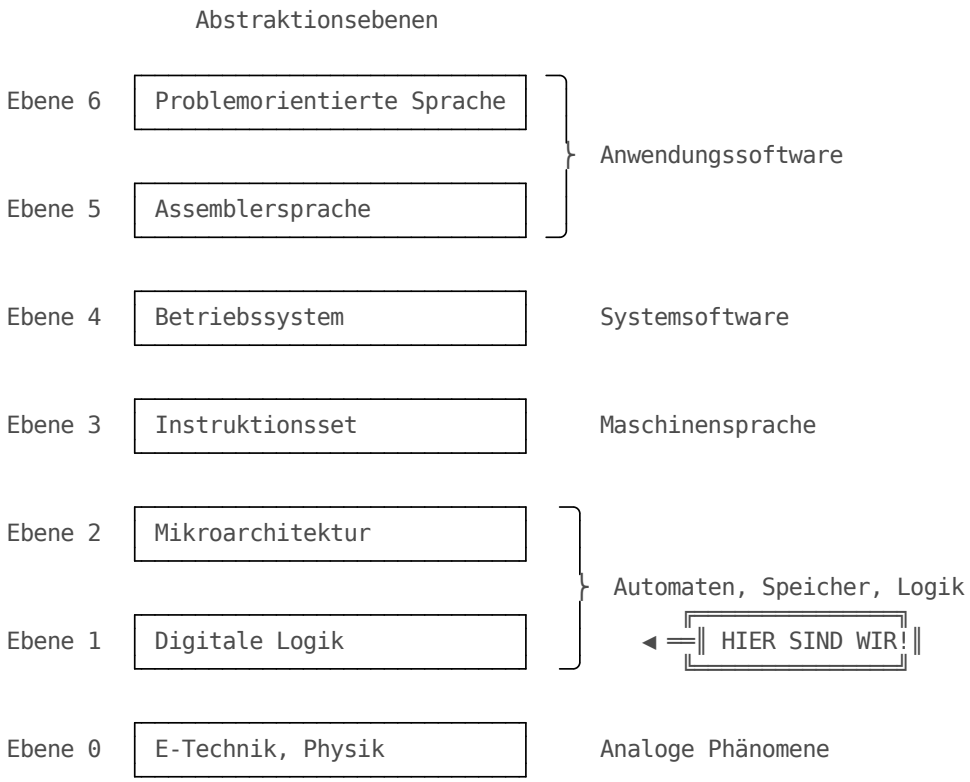
Minimierung von Boolesche Funktionen / Schaltnetze

Parameter	Kursinformationen
Veranstaltung:	`Digitale Systeme / Eingebettete Systeme`
Semester:	`Wintersemester 2025/26`
Hochschule:	`Technische Universität Freiberg`
Inhalte:	`Verfahren von Quine-McCluskey, Realisierung von Schaltnetzen`
Link auf GitHub:	https://github.com/TUBAF-IfI-LiaScript/VL_EingebetteteSysteme/blob/master/04_Schaltnetze.md
Autoren:	Sebastian Zug & André Dietrich & Fabian Bär & Copilot



Fragen an die Veranstaltung

- Erläutern Sie das Verfahren von Quine-McCluskey.
- Grenzen Sie die Begriffe Schaltnetz und Schaltfunktion voneinander ab.
- Erklären Sie die Idee von Multiplexern und Kodierern.
- Welche Besonderheit besteht bei der Ableitung der Schaltfunktionen für einen Dekodierer? Was ist ein SLPD und welche Ausprägungen kennen Sie davon?
- Welche Gatterkombinationen sind geeignet, um beliebige Schaltfunktionen damit umzusetzen?
- Was ist ein Glitch?
- Wie können Sie Hazards in kombinatorischen Schaltungen vermeiden?
- Welche Rolle spielt die Laufzeitverzögerung bei der Schaltnetzsynthese?



Feedbacks/Ergänzungen

Format	Informatik Studierende	Nicht-Informatik Studierende
Verbesserungsvorschlag	0	2
Fragen	2	1
generelle Hinweise	1	1

https://github.com/TUBAF-IfI-LiaScript/VL_EingebetteteSysteme/tree/master/exampleCode/03_Minimierung

Was gefällt Ihnen daran, wo sehen Sie ggf. Verbesserungsbedarf?

Können wir eine Übersicht bekommen, die die Formeln in der hier verwendeten und traditionellen Schreibweise gegenüberstellt?

ja, kommt auf Seite 12.

Beispielanwendung

Nehmen wir an, dass wir die Logik einer Kaffeemaschine umsetzen wollen. Diese verfügt über verschiedene Eingänge wie Sensoren an der Heizplatte, einem Füllstandsmesser, Drucksensorik usw. Wir gehen davon aus, dass diese lediglich digitale Ausgaben generiert.

Die Variable y gibt entsprechend einen Fehlerzustand wieder und ist zum Beispiel mit einer LED verknüpft. Ab Zustand 6 bis 14 soll diese Leuchten.

x_3	x_2	x_1	x_0	y	**Zustand**
0	0	0	0	0	0 - Initialisierung
0	0	0	1	0	1 - Heizplatte erwärmen
0	0	1	0	0	2 - Wasserkocher an
0	0	1	1	0	3 -
0	1	0	0	0	4 -
0	1	0	1	0	5 -
0	1	1	0	1	6 - Wasser fehlt
0	1	1	1	1	7 - Kaffeefach geöffnet
1	0	0	0	1	8 - Druckabfall
1	0	0	1	1	9 - ...
1	0	1	0	1	10 -
1	0	1	1	1	11 -
1	1	0	0	1	12 -
1	1	0	1	1	13 -
1	1	1	0	1	14 - Wartung fällig
1	1	1	1	0	15 - Kaffee fertig

Wie muss also die Schaltung für diese Aufgabe umgesetzt werden?

Schritt 1: Aufstellen der Gleichungen

Kanonische Disjunktive Normalform (KDNF)

- eindeutige Darstellung einer booleschen Funktion f als Disjunktion von Mintermen
- Beispiel: $(x \cdot y \cdot z) + (x \cdot y \cdot z) + (x \cdot y \cdot z)$ ist KDNF von $f(x, y, z)$

Kanonische Konjunktive Normalform (KKNF)

- eindeutige Darstellung einer booleschen Funktion f als Konjunktion von Maxtermen
- Beispiel: $(x + y) \cdot (x + y) \cdot (x + y)$ ist KKNF von $f(x, y)$

x_3	x_2	x_1	x_0	y	**Minterm**	**Maxterm**
0	0	0	0	0	???	???
0	0	0	1	0		
0	0	1	0	0		
0	0	1	1	0		
0	1	0	0	0		
0	1	0	1	0		
0	1	1	0	1		
0	1	1	1	1		
1	0	0	0	1		
1	0	0	1	1		
1	0	1	0	1		
1	0	1	1	1		
1	1	0	0	1		
1	1	0	1	1		
1	1	1	0	1		
1	1	1	1	0		

x_3	x_2	x_1	x_0	y	**Minterme**
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	1	$\overline{x_3} \cdot x_2 \cdot x_1 \cdot \overline{x_0}$
0	1	1	1	1	$\overline{x_3} \cdot x_2 \cdot x_1 \cdot x_0$
1	0	0	0	1	$x_3 \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0}$
1	0	0	1	1	$x_3 \cdot \overline{x_2} \cdot \overline{x_1} \cdot x_0$
1	0	1	0	1	$x_3 \cdot \overline{x_2} \cdot x_1 \cdot \overline{x_0}$
1	0	1	1	1	$x_3 \cdot \overline{x_2} \cdot x_1 \cdot x_0$
1	1	0	0	1	$x_3 \cdot x_2 \cdot \overline{x_1} \cdot \overline{x_0}$
1	1	0	1	1	$x_3 \cdot x_2 \cdot \overline{x_1} \cdot x_0$
1	1	1	0	1	$x_3 \cdot x_2 \cdot x_1 \cdot \overline{x_0}$
1	1	1	1	0	

Schritt 2: Vereinfachen der Minterme - Analytische Lösung

$$\begin{aligned} y = &\overline{x_3}x_2x_1\overline{x_0} + \overline{x_3}x_2x_1x_0 + \\ &x_3\overline{x_2}\,\overline{x_1}\,\overline{x_0} + x_3\overline{x_2}\,\overline{x_1}x_0 + \\ &x_3\overline{x_2}x_1\overline{x_0} + x_3\overline{x_2}x_1x_0 + \\ &x_3x_2\overline{x_1}\,\overline{x_0} + x_3x_2\overline{x_1}x_0 + \\ &x_3x_2x_1\overline{x_0} \end{aligned}$$

Wie gehen Sie vor? Wir suchen Paare von Mintermen, die sich lediglich in einer Variablen unterscheiden, und fassen diese entsprechd dem Distributivgesetz und Idempotenzgesetz zusammen.

$$\begin{aligned} \overline{x_3}x_2x_1\overline{x_0} + \overline{x_3}x_2x_1x_0 &= \overline{x_3}x_2x_1(\overline{x_0} + x_0) \\ &= \overline{x_3}x_2x_1(1) \\ &= \overline{x_3}x_2x_1 \end{aligned}$$

Erste Stufe der Vereinfachung

Zunächst erweitern wir unseren KDNF um einen der Minterme, so dass die möglichen Minimierungen pro Zeile offensichtlich sind.

$$\begin{aligned} y = &\overline{x_3}x_2x_1\overline{x_0} + \overline{x_3}x_2x_1x_0 + \\ &x_3\overline{x_2}\,\overline{x_1}\,\overline{x_0} + x_3\overline{x_2}\,\overline{x_1}x_0 + \\ &x_3\overline{x_2}x_1\overline{x_0} + x_3\overline{x_2}x_1x_0 + \\ &x_3x_2\overline{x_1}\,\overline{x_0} + x_3x_2\overline{x_1}x_0 + \\ &x_3\overline{x_2}x_1\overline{x_0} + x_3x_2x_1\overline{x_0} \end{aligned}$$

Damit ergibt sich die oben genannten Gleichung in der ersten Vereinfachungsstufe zu

$$y = \overline{x}_3x_2x_1 + x_3\overline{x}_2\,\overline{x}_1 + x_3\overline{x}_2x_1 + x_3x_2\overline{x}_1 + x_3x_1\overline{x}_0$$

Zweite Stufe der Vereinfachung

$$\begin{aligned} y = &\overline{x}_3x_2x_1 + \\ &x_3\overline{x}_2\,\overline{x}_1 + \\ &x_3\overline{x}_2x_1 + \\ &x_3x_2\overline{x}_1 + \\ &x_3x_1\overline{x}_0 \end{aligned}$$

$$y = \overline{x}_3x_2x_1 + x_3\,\overline{x}_2 + x_3\overline{x}_1 + x_3x_1\overline{x}_0$$

Gegenprobe I

Minimize.py

```
from sympy.logic import SOPform
from sympy import symbols
x3, x2, x1, x0 = symbols('x3 x2 x1 x0')

minterms = [[0, 1, 1, 0],
            [0, 1, 1, 1],
            [1, 0, 0, 0],
            [1, 0, 0, 1],
            [1, 0, 1, 0],
            [1, 0, 1, 1],
            [1, 1, 0, 0],
            [1, 1, 0, 1],
            [1, 1, 1, 0]]
result = SOPform([x3, x2, x1, x0], minterms)
print(result)
```

```
(x3 & ~x0) | (x3 & ~x1) | (x3 & ~x2) | (x1 & x2 & ~x3)
(x3 & ~x0) | (x3 & ~x1) | (x3 & ~x2) | (x1 & x2 & ~x3)
```

Überrascht? Offenbar gelingt es dem Minimierungsansatz von *sympy*, eine kompaktere Form zu finden.

Schauen wir uns die Funktion im Karnaugh-Diagramm an!

	$\overline{x}_1\,\overline{x}_0$	$\overline{x}_1 x_0$	$x_1 x_0$	$x_1\overline{x}_0$
$\overline{x}_3\,\overline{x}_2$	0	0	0	0
$\overline{x}_3 x_2$	0	0	1	1
$x_3 x_2$	1	1	0	1
$x_3\overline{x}_2$	1	1	1	1

$$y_{Karnaugh} = \overline{x}_3 x_2 x_1 + x_3\,\overline{x}_2 + x_3\overline{x}_1 + x_3\overline{x}_0$$
$$y_{Analytic} = \overline{x}_3 x_2 x_1 + x_3\,\overline{x}_2 + x_3\overline{x}_1 + x_3 x_1\overline{x}_0$$

Warum "übersieht" unsere analytische Lösung die mögliche Minimierung im letzten Term?

> Weil wir uns in der Vereinfachung der ersten Stufe mit einem Teilergebnis zufriedengegeben haben, das uns in der zweiten Stufe für eine weiteren Aggregation fehlt.

Welche Kombinationen sind zusätzlich möglich?

$$
\begin{aligned}
y = &\overline{x_3} x_2 x_1 \overline{x_0} + \overline{x_3} x_2 x_1 x_0 + \\
&x_3 \overline{x_2}\, \overline{x_1}\, \overline{x_0} + x_3 \overline{x_2}\, \overline{x_1} x_0 + \\
&x_3 \overline{x_2} x_1 \overline{x_0} + x_3 \overline{x_2} x_1 x_0 + \\
&x_3 x_2 \overline{x_1}\, \overline{x_0} + x_3 x_2 \overline{x_1} x_0 + \\
&x_3 \overline{x_2} x_1 \overline{x_0} + x_3 x_2 x_1 \overline{x_0}
\end{aligned}
$$

$$
y = \overline{x}_3 x_2 x_1 + x_3 \overline{x}_2\, \overline{x}_1 + x_3 \overline{x}_2 x_1 + x_3 x_2 \overline{x}_1 + x_3 x_1 \overline{x}_0 + x_3 \overline{x}_1\, \overline{x}_0
$$

Daraus ergibt sich dann auf der zweiten Stufe die "zusätzliche" Minimierungsoption, die die beiden letzten Minterme zusammenfasst.

$$
y = \overline{x}_3 x_2 x_1 + x_3 \overline{x}_2\, \overline{x}_1 + x_3 \overline{x}_2 x_1 + x_3 x_2 \overline{x}_1 + \underbrace{x_3 x_1 \overline{x}_0 + x_3 \overline{x}_1\, \overline{x}_0}
$$

$$
y = \overline{x}_3 x_2 x_1 + x_3\, \overline{x}_2 + x_3 \overline{x}_1 + x_3 \overline{x}_0
$$

Erkenntnisse:

- Das Karnaugh-Veitch-Diagramm zeigt mögliche Minimierungspotentiale auf, hinsichtlich der Bildung der Schleifen können unterschiedliche Strategien zum Tragen kommen.
- Offenkundig brauchen wir ein systematischeres Vorgehen bei der Vereinfachung, das alle Kombinationen möglicher Terme berücksichtigt.

Verfahren nach Quine-McCluskey

Das Verfahren bezieht sich zunächst nur auf Funktionsdarstellungen in kanonischer disjunktiver Normalform (KDNF) und zielt darauf ab eine systematische Minimierung der Minterme durchzuführen.

Ermittlung der Primterme

Die Minterme werden in tabellarischer Form entsprechend der Zahl der "1"en sortiert. Es gibt 1 Eintrag mit einer 1 (m_8) und jeweils 4 mit 2 oder 3 Einsen.

	x_3	x_2	x_1	x_0	y
	0	0	0	0	0
	0	0	0	1	0
	0	0	1	0	0
	0	0	1	1	0
	0	1	0	0	0
	0	1	0	1	0
m_6	0	1	1	0	1
m_7	0	1	1	1	1
m_8	1	0	0	0	1
m_9	1	0	0	1	1
m_{10}	1	0	1	0	1
m_{11}	1	0	1	1	1
m_{12}	1	1	0	0	1
m_{13}	1	1	0	1	1
m_{14}	1	1	1	0	1
	1	1	1	1	0

m_x	x_3	x_2	x_1	x_0	**OK**
m_8	1	0	0	0	.
m_6	0	1	1	0	.
m_9	1	0	0	1	.
m_{10}	1	0	1	0	.
m_{12}	1	1	0	0	.
m_7	0	1	1	1	.
m_{11}	1	0	1	1	.
m_{13}	1	1	0	1	.
m_{14}	1	1	1	0	.

In einem zweiten Schritt werden die sortierten Minterme evaluiert. Dabei können nur die Blöcke blau-weiß und weiß-grau potentiell zusammenfassbare Mintermpaare umfassen.

	x_3	x_2	x_1	x_0	OK
$m_8 \cdot m_9$	1	0	0	-	.
$m_8 \cdot m_{10}$	1	0	-	0	.
$m_8 \cdot m_{12}$	1	-	0	0	.
$m_6 \cdot m_7$	0	1	1	–	.
$m_6 \cdot m_{14}$	–	1	1	0	.
$m_9 \cdot m_{11}$	1	0	–	1	.
$m_9 \cdot m_{13}$	1	–	0	1	.
$m_{10} \cdot m_{11}$	1	0	1	–	.
$m_{10} \cdot m_{14}$	1	–	1	0	.
$m_{12} \cdot m_{13}$	1	1	0	–	.
$m_{12} \cdot m_{14}$	1	1	–	0	.

Kombinations.py

```python
import numpy as np
minterms = [[0, 1, 1, 0],
            [0, 1, 1, 1],
            [1, 0, 0, 0],
            [1, 0, 0, 1],
            [1, 0, 1, 0],
            [1, 0, 1, 1],
            [1, 1, 0, 0],
            [1, 1, 0, 1],
            [1, 1, 1, 0]]
distances = np.zeros([len(minterms),len(minterms)]).astype('float')
for i in range(0, len(minterms)):
    for k in range(0, i):
        diff = np.subtract(minterms[i], minterms[k])
        dist = np.sum(np.abs(diff))
        distances[i,k] = dist

print("Distanzen der Minterme")
for j in range(0, len(distances)):  # Requiered instead print(distances)
    print(distances[j])             # due to pyscript output constraints
print("Kombinationen mit Distanz 1: {}".format(np.count_nonzero(distances
  == 1)))
```

```
Distanzen der Minterme
Distanzen der Minterme
[0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0.]
[3. 4. 0. 0. 0. 0. 0. 0. 0.]
[4. 3. 1. 0. 0. 0. 0. 0. 0.]
[3. 4. 0. 0. 0. 0. 0. 0. 0.]
[2. 3. 1. 2. 0. 0. 0. 0. 0.]
[4. 3. 1. 0. 0. 0. 0. 0. 0.]
[2. 3. 1. 2. 0. 0. 0. 0. 0.]
[3. 2. 2. 1. 1. 0. 0. 0. 0.]
[3. 2. 2. 1. 1. 0. 0. 0. 0.]
[2. 3. 1. 2. 2. 3. 0. 0. 0.]
[2. 3. 1. 2. 2. 3. 0. 0. 0.]
[3. 2. 2. 1. 3. 2. 1. 0. 0.]
[3. 2. 2. 1. 3. 2. 1. 0. 0.]
[1. 2. 2. 3. 1. 2. 1. 2. 0.]
Kombinationen mit Distanz 1: 11
[1. 2. 2. 3. 1. 2. 1. 2. 0.]
Kombinationen mit Distanz 1: 11
```

Die zweite Stufe wiederholt die Vorgänge - Sortierung und Evaluation erneut.

	x_3	x_2	x_1	x_0	**OK**
$m_8 \cdot m_9$	1	0	0	-	$*$
$m_8 \cdot m_{10}$	1	0	-	0	$*$
$m_8 \cdot m_{12}$	1	-	0	0	$*$
$m_6 \cdot m_7$	0	1	1	–	P4
$m_6 \cdot m_{14}$	–	1	1	0	P5
$m_9 \cdot m_{11}$	1	0	–	1	$*$
$m_9 \cdot m_{13}$	1	–	0	1	$*$
$m_{10} \cdot m_{11}$	1	0	1	–	$*$
$m_{10} \cdot m_{14}$	1	–	1	0	$*$
$m_{12} \cdot m_{13}$	1	1	0	–	$*$
$m_{12} \cdot m_{14}$	1	1	–	0	$*$

Es dürfen nur Zeilen zusammengefasst werden, die die "-" an den gleichen Positionen haben (!). Außerdem tauchen ab der zweiten Zusammenfassung (also in der dritten Tabelle) Konjunktionen doppelt auf, welche aber nur einmal notiert werden.

	x_3	x_2	x_1	x_0	**OK**
$m_8 \cdot m_9 \cdot m_{10} \cdot m_{11}$	1	0	-	-	.
$m_8 \cdot m_9 \cdot m_{12} \cdot m_{13}$	1	-	0	-	.
$m_8 \cdot m_{10} \cdot m_9 \cdot m_{11}$	1	0	-	-	.
$m_8 \cdot m_{10} \cdot m_{12} \cdot m_{14}$	1	-	-	0	.
$m_8 \cdot m_{12} \cdot m_9 \cdot m_{13}$	1	-	0	-	.
$m_8 \cdot m_{12} \cdot m_{10} \cdot m_{14}$	1	-	-	0	.

Eine weitere Minimierung ist offenbar nicht möglich. Offenbar können 3 Minterme der ersten Stufe nicht weiter zusammengefasst werden.

	x_3	x_2	x_1	x_0	**OK**
$m_8 \cdot m_9 \cdot m_{10} \cdot m_{11}$	1	0	-	-	P1
$m_8 \cdot m_9 \cdot m_{12} \cdot m_{13}$	1	-	0	-	P2
$m_8 \cdot m_{10} \cdot m_9 \cdot m_{11}$	1	0	-	-	identisch P1
$m_8 \cdot m_{10} \cdot m_{12} \cdot m_{14}$	1	-	-	0	P3
$m_8 \cdot m_{12} \cdot m_9 \cdot m_{13}$	1	-	0	-	identisch P2
$m_8 \cdot m_{12} \cdot m_{10} \cdot m_{14}$	1	-	-	0	identisch P3

Mit den übrigen Ergebnissen der zweiten Stufe ergeben sich daraus insgesamt 5 Primimplikanten.

$$
\begin{aligned}
P_1 &= m_8 \bullet m_9 \bullet m_{10} \bullet m_{11} \\
P_2 &= m_8 \bullet m_9 \bullet m_{12} \bullet m_{13} \\
P_3 &= m_8 \bullet m_{10} \bullet m_{12} \bullet m_{14} \\
P_4 &= m_6 \bullet m_7 \\
P_5 &= m_6 \bullet m_{14}
\end{aligned}
$$

Darstellung der Minterme im Karnaugh-Diagramm

$$
\begin{aligned}
P_1 &= m_8 \bullet m_9 \bullet m_{10} \bullet m_{11} \\
P_2 &= m_8 \bullet m_9 \bullet m_{12} \bullet m_{13} \\
P_3 &= m_8 \bullet m_{10} \bullet m_{12} \bullet m_{14} \\
P_4 &= m_6 \bullet m_7 \\
P_5 &= m_6 \bullet m_{14}
\end{aligned}
$$

	$\overline{x}_1\,\overline{x}_0$	$\overline{x}_1 x_0$	$x_1 x_0$	$x_1\overline{x}_0$
$\overline{x}_3\,\overline{x}_2$	0	0	0	0
$\overline{x}_3 x_2$	0	0	1 (m_7)	1 (m_6)
$x_3 x_2$	1 (m_{12})	1 (m_{13})	0	1 (m_{14})
$x_3\overline{x}_2$	1 (m_8)	1 (m_9)	1 (m_{11})	1 (m_{10})

Visualisierung der generierten Primimplikanten

	$\overline{x}_1\,\overline{x}_0$	$\overline{x}_1 x_0$	$x_1 x_0$	$x_1\overline{x}_0$
$\overline{x}_3\,\overline{x}_2$	0	0	0	0
$\overline{x}_3 x_2$	0	0	1 (P_4)	1 (P_4, P_5)
$x_3 x_2$	1 (P_2, P_3)	1 (P_2)	0	1 (P_3, P_5)
$x_3\overline{x}_2$	1 (P_1, P_2, P_3)	1 (P_1, P_2)	1 (P_1)	1 (P_1, P_3)

Offenbar werden die Minterme bis auf zwei Ausnahmen mehrfach durch die Primimplikaten abgedeckt. Hier ist eine weitere Minimierung notwendig.

Die als zweite Quine'sche Tabelle bezeichnete Primimplikatentafel fasst die Primimplikanten und die zugehörigen Minterme zusammen.

	m_{14}	m_{13}	m_{12}	m_{11}	m_{10}	m_9	m_8	m_7	m_6
P_1				x	x	x	x		
P_2		x	x			x	x		
P_3	x		x		x		x		
P_4								x	x
P_5	x								x

Lösung der minimalen Überdeckung - intuitiv

Die Minterme m_{13}, m_{11} und m_7 kommen nun in jeweils einem Primimplikanten vor. Folglich sind diese sogenannte essentielle Primimplikanten (P_1, P_2, P_4).

	m_{14}	m_{13}	m_{12}	m_{11}	m_{10}	m_9	m_8	m_7	m_6
P_1				x	x	x	x		
P_2		x	x			x	x		
P_3	x		x		x		x		
P_4								x	x
P_5	x								x

	$\overline{x}_1\,\overline{x}_0$	$\overline{x}_1 x_0$	$x_1 x_0$	$x_1\overline{x}_0$
$\overline{x}_3\,\overline{x}_2$	0	0	0	0
$\overline{x}_3 x_2$	0	0	m_7 (P_4)	m_6 (P_4, P_5)
$x_3 x_2$	m_{12} (P_2, P_3)	m_{13} (P_2)	0	m_{14} (P_3, P_5)
$x_3\overline{x}_2$	m_8 (P_1, P_2, P_3)	m_9 (P_1, P_2)	m_{11} (P_1)	m_{10} (P_1, P_3)

Statement 1: m_{13} ist in P_2 enthalten und schließt damit m_{12}, m_9 und m_8 ein. Diese können gestrichen werden.

	m_{14}	m_{13}	m_{11}	m_{10}	m_7	m_6
P_1			x	x		
P_2		x				
P_3	x			x		
P_4					x	x
P_5	x					x

Statement 2: m_{11} ist nur in P_1 enthalten und schließt damit m_{10} ein.

Statement 3: P_4 deckt automatisch m_6 ab.

	m_{14}	m_{13}	m_{11}	m_7
P_1			x	
P_2		x		
P_3	x			
P_4				x
P_5	x			

P_3 und P_5 decken ein und den selben Minterm ab. Welcher sollte gewählt werden?

$$
\begin{aligned}
P_1 &= m_8 \bullet m_9 \bullet m_{10} \bullet m_{11} \\
P_2 &= m_8 \bullet m_9 \bullet m_{12} \bullet m_{13} \\
P_3 &= m_8 \bullet m_{10} \bullet m_{12} \bullet m_{14} \\
P_4 &= m_6 \bullet m_7 \\
P_5 &= m_6 \bullet m_{14}
\end{aligned}
$$

Ergebnis: Unsere relevanten Primimplikanten sind $y = P_1 + P_2 + P_3 + P_4 = \overline{x}_3 x_2 x_1 + x_3\,\overline{x}_2 + x_3 \overline{x}_1 + x_3 \overline{x}_0$.

Lösung der minimalen Überdeckung - algorithmisch

Spaltendominanzprüfung

Die Spalten werden paarweise darauf verglichen, ob nicht eine Spalte existiert, in der die markierten Primterme eine Teilmenge der markierten Primterme der anderen Spalte sind. Ist dies der Fall, so kann die Spalte mit der Obermenge gestrichen werden, denn es müssen alle Konjunktionen erfasst werden und daher ist die Konjunktion mit der Obermenge durch Auswahl der Konjunktion mit der Teilmenge ebenfalls erfasst.

Beispiel Spaltendominanz:

Bei der **Spaltendominanzprüfung** werden die Spalten der Primimplikantentafel paarweise miteinander verglichen. Jede Spalte repräsentiert dabei einen **Minterm** und enthält Markierungen („x“), die angeben, **welche Primimplikanten** diesen Minterm abdecken.

Eine Spalte (m_i) **dominiert** eine andere Spalte (m_j), wenn alle „x“ in (m_i) auch in (m_j) vorkommen, also:

$$m_i \subseteq m_j$$

In diesem Fall kann die **größere** (also die von (m_i) dominierte) Spalte (m_j) gestrichen werden, denn sobald (m_i) abgedeckt ist, ist (m_j) automatisch ebenfalls abgedeckt. Die dominierte Spalte liefert also keine zusätzliche Bedingung für die Auswahl der Primimplikanten.

	(m_1)	**(m_2)**	**(m_3)**
(P_A)	x	x	
(P_B)	x		x
Dominanzprüfung	streichen		

Wir lesen die Spalten als Mengen der abdeckenden Primimplikanten:

- ($m_1 = \{P_A, P_B\}$)
- ($m_2 = \{P_A\}$)
- ($m_3 = \{P_B\}$)

Da ($m_2 \subset m_1$) gilt, wird **(m_1)** gestrichen. Jede Abdeckung, die (m_2) erfüllt, deckt automatisch auch (m_1) ab.
Zeilendominanzprüfung

Bei der **Zeilendominanzprüfung** werden die **Zeilen** der Primimplikantentafel paarweise miteinander verglichen. Jede Zeile repräsentiert dabei einen **Primimplikanten** und enthält Markierungen („x“), die angeben, **welche Minterme** dieser Primimplikant abdeckt.

Eine Zeile (P_i) **dominiert** eine andere Zeile (P_j), wenn alle „x“ in (P_j) auch in (P_i) vorkommen, also:

$$P_j \subseteq P_i$$

In diesem Fall kann die **kleinere** (also die von (P_i) dominierte) Zeile (P_j) gestrichen werden, denn (P_i) deckt **alle** Minterme ab, die (P_j) abdeckt — und eventuell noch mehr. Damit ist (P_j) überflüssig.

	(m_1)	**(m_2)**	**(m_3)**	**Dominanzprüfung**
(P_A)	x	x		streichen
(P_B)	x	x	x	

Wir lesen die Zeilen als Mengen der abgedeckten Minterme:

- ($P_A = \{m_1, m_2\}$)
- ($P_B = \{m_1, m_2, m_3\}$)

Da ($P_A \subset P_B$) gilt, wird **(P_A)** gestrichen. (P_B) deckt alle Minterme ab, die auch (P_A) abdeckt, und zusätzlich noch (m_3).

Für das Verfahren von Quine-McCluskey exisitieren webbasierte Lösungen. Evaluieren Sie den Ablauf mit https://www.goldi-labs.de/SANE/view6

Schaltungssynthese

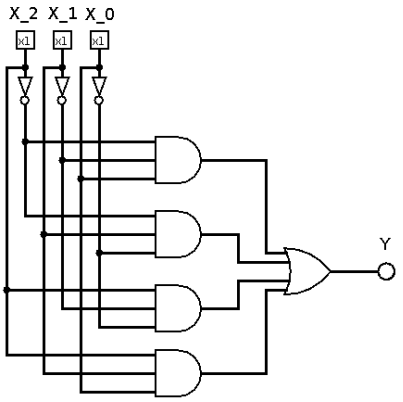
Schaltungssynthese beschreibt die Umsetzung einer booleschen Funktion in eine Hardware-Schaltung. Grundlage sind Logikgatter, die als spezifische Schaltnetze industriell gefertigt werden.

Beispiel

$y = f(x_0, x_1, x_2)$

Die Funktion evaluiert die Parität der Variablen. Für eine ungerade Zahl von einsen wird eine "1" ausgegeben, sonst eine "0".

x_2	x_1	x_0	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



Ein Schaltnetz ist eine schaltungstechnische Realisierung einer Schaltfunktion $f : \{0,1\}^n \to \{0,1\}^m$ mit $n, m \geq 1$. f ist zerlegbar in m Boolesche Funktionen mit den gleichen n Eingangsvariablen: $f_1(x_1, x_2, ..., x_n), f_2(x_1, x_2, ..., x_n), ..., f_m(x_1, x_2, ..., x_n)$

Jedes Schaltnetz ist als gerichteter, azyklischer Graph darstellbar:

- Gatter, Ein- und Ausgänge sind die Knoten
- Verbindungsleitungen entsprechen den gerichteten Kanten
- Zyklen (Rückkopplungen) sind nicht zulässig!

Aus der Darstellung als kanonische Normalform resultiert, dass jede Schaltfunktion durch ein zweistufiges Schaltnetz realisierbar ist, wenn

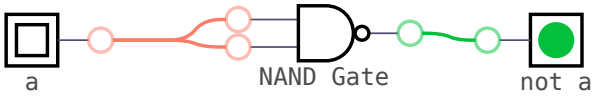
- alle Eingangssignale sowohl einfach als auch negiert vorliegen
- Gatter mit einer ausreichenden Größe (d.h. Anzahl an Eingängen)

zur Verfügung stehen

> **Merke:**
>
> - Jedes Schaltnetz kann nur mit UND-, ODER- und NICHT-Gattern aufgebaut werden.
> - Jedes Schaltnetz kann nur mit NAND Gattern aufgebaut werden.
> - Jedes Schaltnetz kann nur mit NOR Gattern aufgebaut werden.

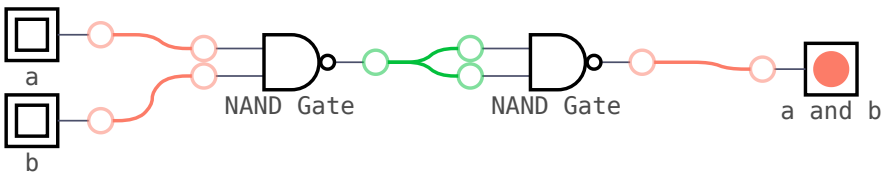
NOT

$\overline{a} = \overline{a \cdot a}$



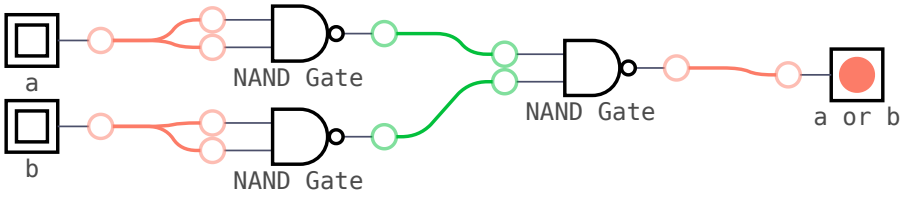
AND

$a \cdot b = \overline{\overline{a \cdot a}}$



OR

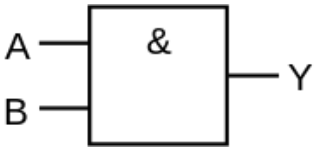
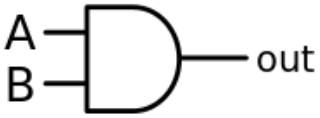
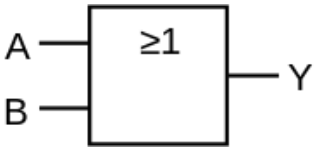
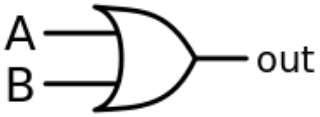
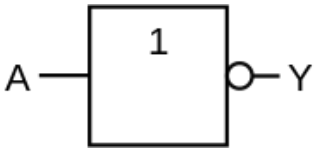
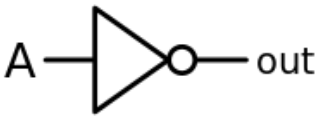
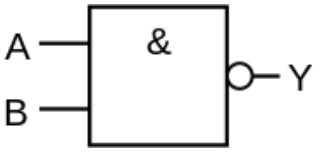
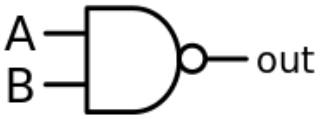
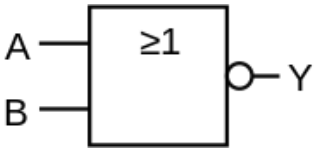
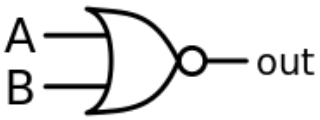
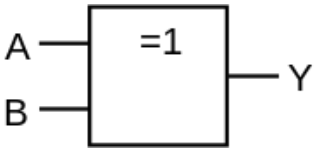
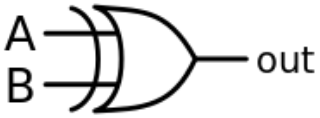
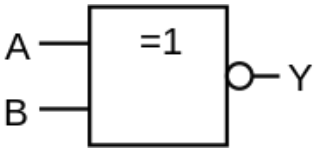
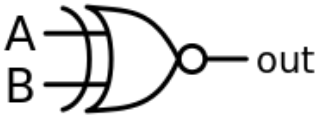
$a + b = \overline{\overline{a} \cdot \overline{b}}$



Logikgatter als Grundlage der Schaltnetze

Ein Logikgatter, auch nur Gatter ist die technische Realisierung einer booleschen Funktion, die binäre Eingangssignale zu einem binären Ausgangssignal verarbeitet. Die Eingangssignale werden durch Implementierung logischer Operatoren, wie der Konjunktion (Und-Gatter), der Disjunktion (Oder-Gatter), der Kontravalenz (Exklusiv-Oder-Gatter) oder der Negation (Nicht-Gatter) zu einem einzigen logischen Ergebnis umgewandelt und auf das Ausgangssignal abgebildet.

Gatterfunktionen können zudem einen negierten Ausgang abbilden: NAND-Gatter (Nicht-Und), NOR-Gatter (Nicht-Oder), XNOR-Gatter (Nicht-Exklusiv-Oder).

Funktion	IEC 60617-12 : 1997	ANSI/IEEE Std 91/91a-1991
UND $Y = A \cdot B$ $Y = A \wedge B$		
OR $Y = A + B$ $Y = A \vee B$		
NOT $Y = \overline{A}$ $Y = A$		
$NAND$ $Y = \overline{A \cdot B}$ $Y = A \overline{\wedge} B = \overline{A \wedge B}$		
NOR $Y = \overline{A + B}$ $Y = A \overline{\vee} B = \overline{A \vee B}$		
XOR $Y = A \oplus B$ $Y = A \underline{\vee} B$		
$XNOR$ $Y = A \odot B$ $Y = A \underline{\overline{\vee}} B = \overline{A \underline{\vee} B}$		

Im englischen Sprachraum waren und sind die amerikanischen Symbole (rechte Spalte) üblich. Die IEC-Symbole sind international auf beschränkte Akzeptanz gestoßen und werden in der amerikanischen Literatur (fast) durchgängig ignoriert.

Zweistufige Schaltungssynthese

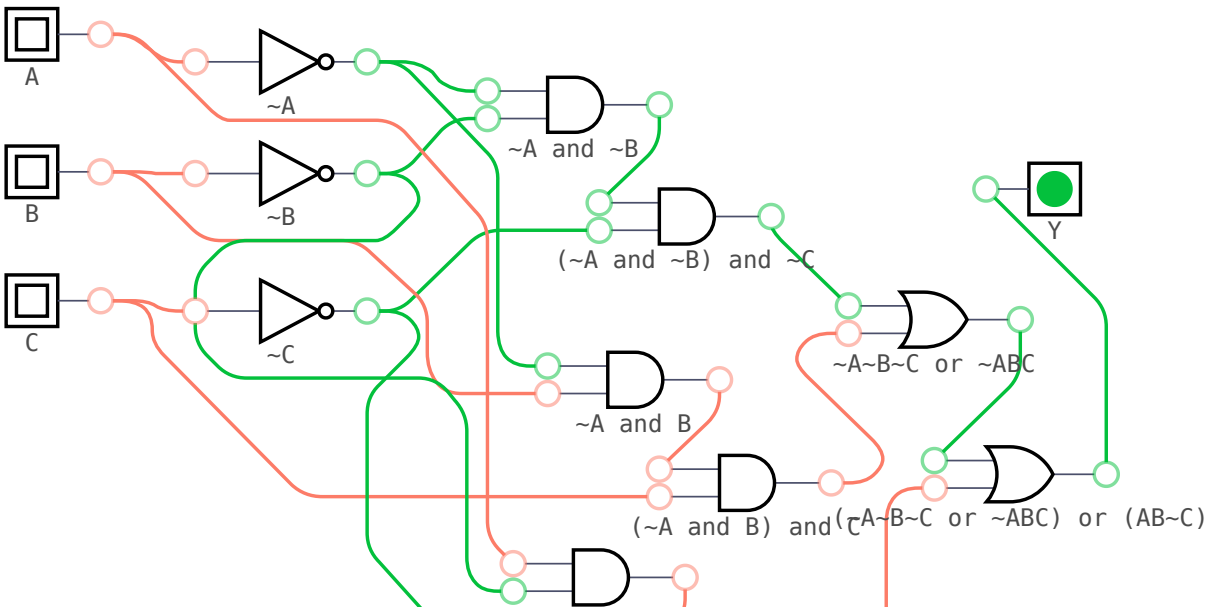
Dazu werden für eine KDNF je Minterm ein UND-Gatter und ein ODER-Gatter zur Disjunktion aller Minterme benötigt. Für die KKNF sind es entsprechend Maxterm ein ODER-Gatter ind ein UND-Gatter zur Konjunktion aller Maxterme.

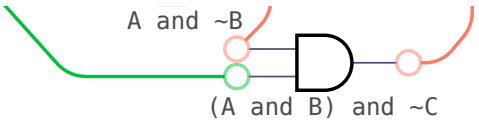
Die Anzahl der benötigten Gatter zur Realisierung der KDNF (bzw. KKNF) einer n-stelligen Schaltfunktion:

- je Boolescher Funktion max. 2^n UND-Gatter (bzw. ODER-Gatter) mit jeweils max. n Eingängen
- je Boolescher Funktion ein ODER-Gatter (bzw. UND-Gatter) mit max. 2^n Eingängen

Beispiel:

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

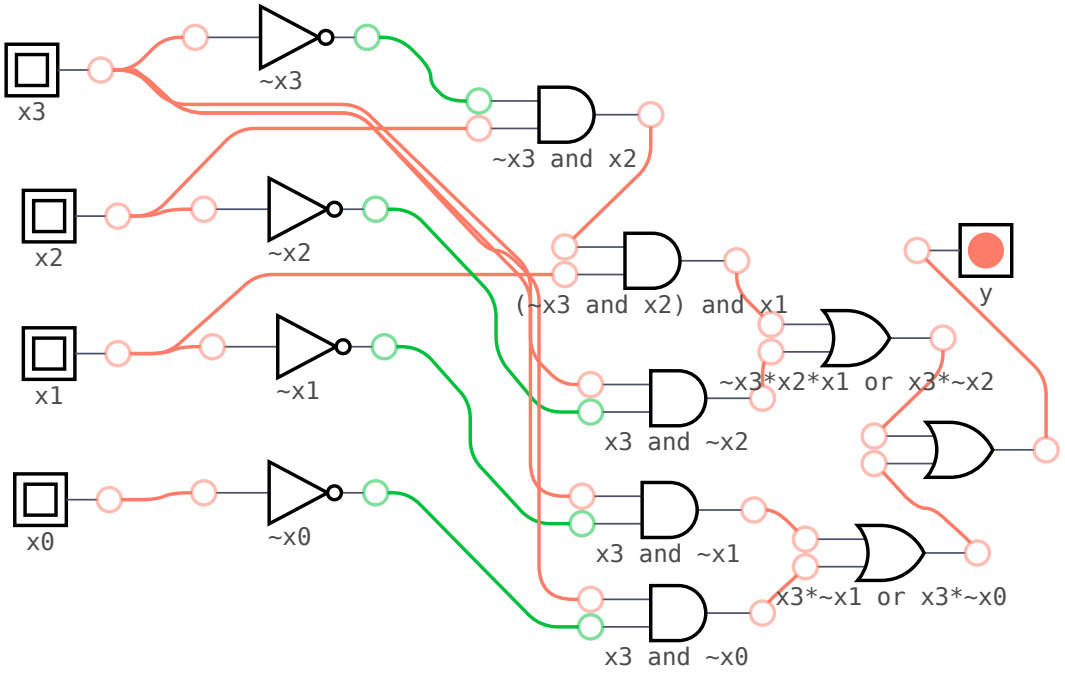




Zurück zum Beispiel - Schritt 3 Umsetzung

Für unser Kaffeemaschinenbeispiel kann eine Lösung also zum Beispiel folgendermaßen aussehen. Wir hatten ja folgende Funktion als minimale Darstellung identifiziert:

$$y = \overline{x}_3 x_2 x_1 + x_3 \overline{x}_2 + x_3 \overline{x}_1 + x_3 \overline{x}_0$$



Für die Zustände 6 bis 14 sollte ein Fehler ausgeben werden

x_3	x_2	x_1	x_0	y
0	0	0	0	0
...	...	...	...	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

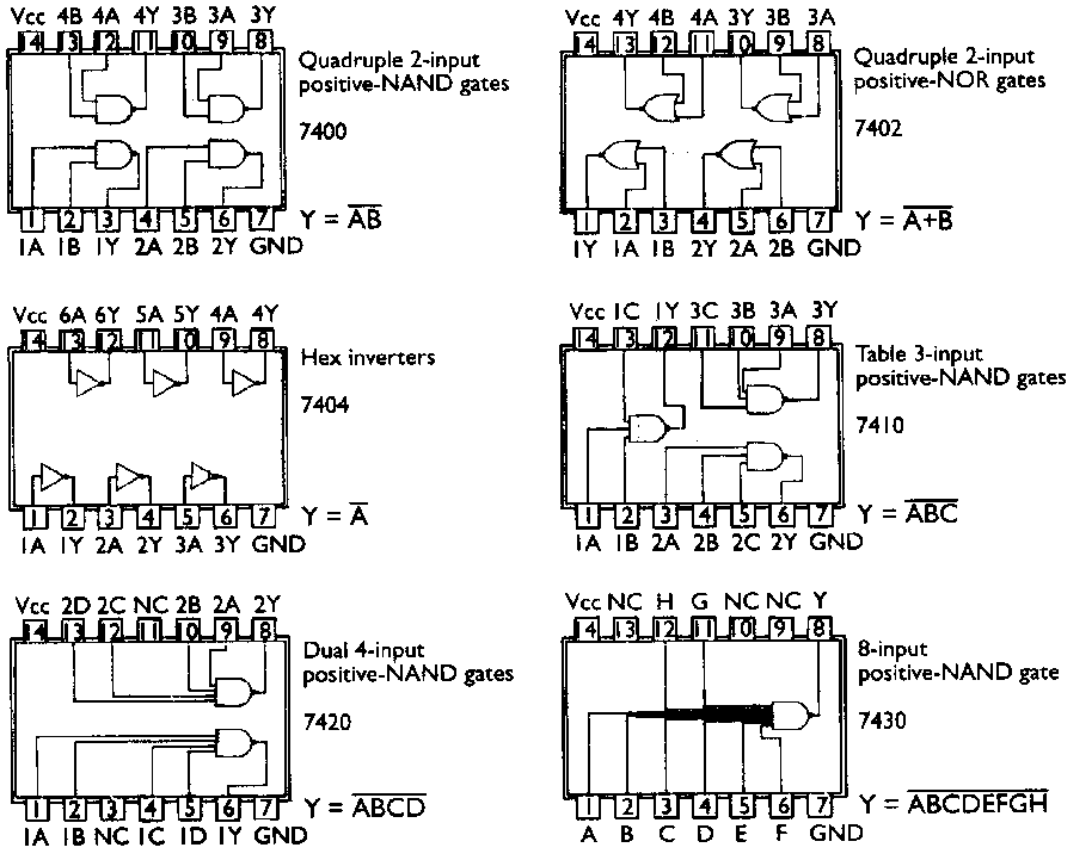
Worin unterscheidet sich unsere minimale Form von der technischen Umsetzung?

Spielen Sie ein wenig mit der Simulation, um sich von der Wirksamkeit unserer Lösung zu überzeugen.

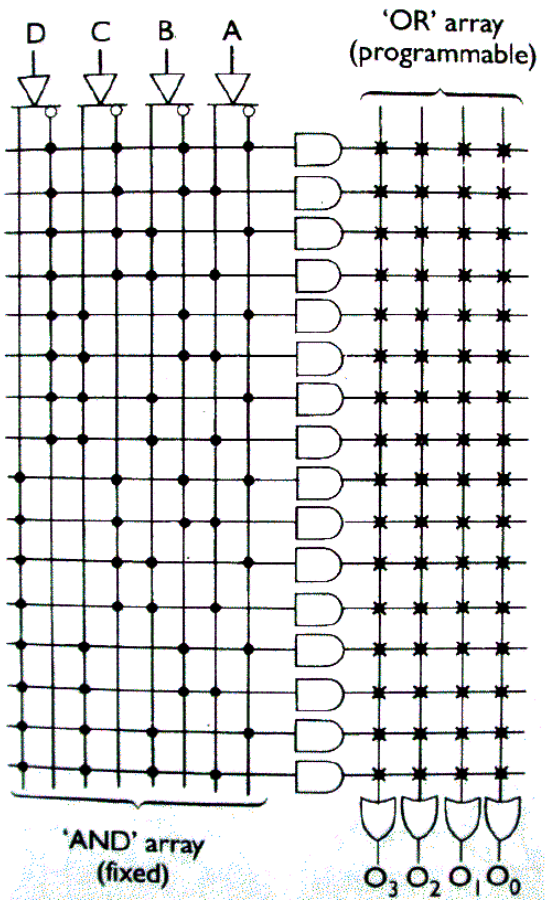
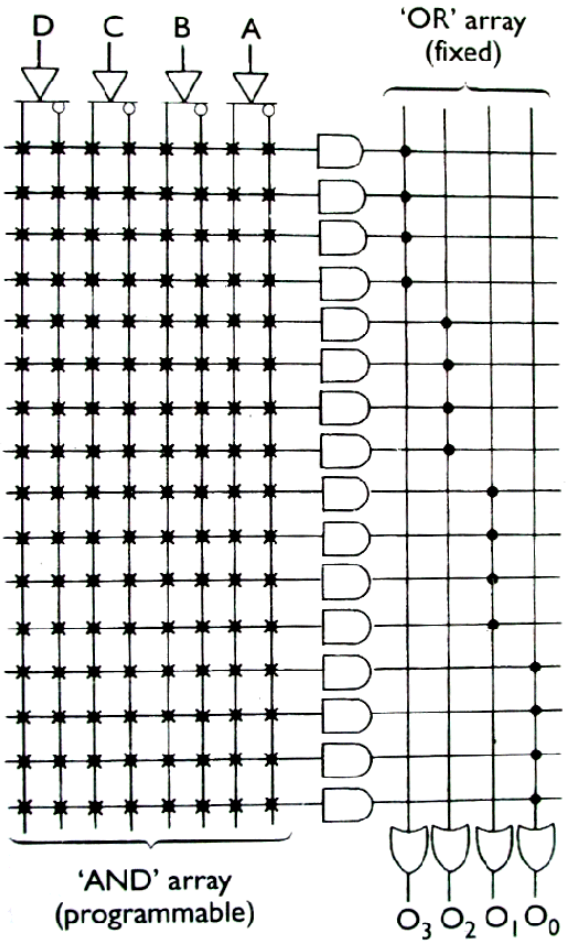
Umsetzung

Prinzipiell lassen sich alle logischen Verknüpfungen als Gatter realisieren. Die 74xx Reihe umfasst dabei die Basisbausteine und darüber hinaus integrierte Schaltnetze, die höhere Funktionen bereitstellen (Addierwerk, Multiplexer, Decoder)

https://de.wikipedia.org/wiki/Liste_von_integrierten_Schaltkreisen_der_74xx-Familie

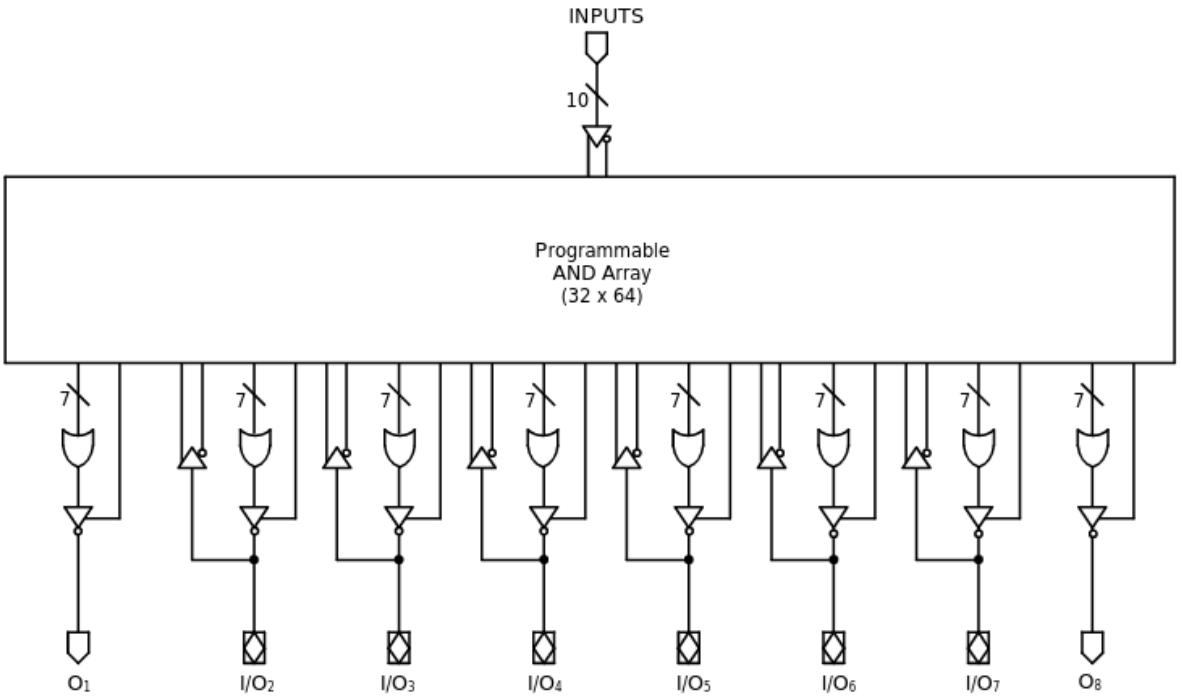


Aufwändigere Schaltnetze greifen auf ICs zurück, die zwei Ebenen AND-Array und OR-Array kombinieren.

PROM - „Programmable Read-Only Memory“	PAL - "Programmable Array Logic"
16 Worten zu 4 Bit	4 Ein- und Ausgängen und 4 Produkttermen je Ausgang
	
Adressdekoder entspricht einer festen UND-Matrix, Koppelelemente bilden eine programmierbare ODER-Matrix	programmierbare UND-Matrix, Produktterme werden mit fester ODER-Matrix verknüpft

PROM realisiert unmittelbar die Wahrheitstabelle in Hardware! Ein PROM mit 2^m n-Bit Worten kann jede beliebige Schaltfunktion $f : \{0,1\}^n \to \{0,1\}^m$ ohne Minimierung implementieren.

PAL / GAL („Programmable / Generic Array Logic“ kann jede (ggf. minimierte) DNF realisieren, wenn Zahl der Produktterme je ODER ausreicht.



PAL16R8 Family, Advanced Micro Devices [1]

PAL wird eingesetzt:

- wenn viele Variablen und relative wenige Terme vorkommen.
- viele Eingangsbelegungen werden auf dieselbe Ausgangsbelegung abgebildet.

PROM wird eingesetzt:

- wenn jede Eingangsbelegung auf eine individuelle Ausgangsbelegung abgebildet werden muss.

[1] Datenblatt PAL16R8 Family, Advanced Micro Devices, link, 1996

Herausforderungen

In der Elektronik bezeichnet man mit Glitch eine kurzzeitige Falschaussage in logischen Schaltungen und temporäre Verfälschung einer booleschen Funktion. Diese tritt auf, weil die Signallaufzeiten in den einzelnen Gattern niemals vollkommen gleich sind.

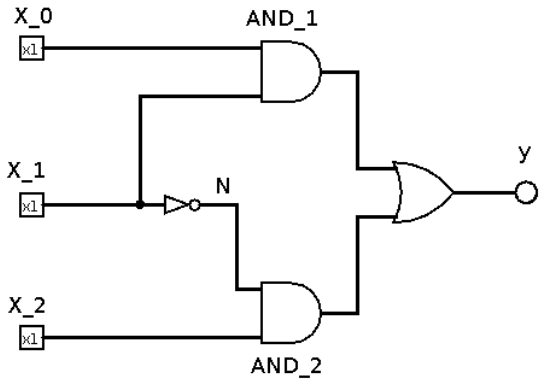
Ausgangspunkt

Zeitverhalten realer Logikbauteile

https://www.ti.com/lit/ds/symlink/sn74lvc2g08-ep.pdf

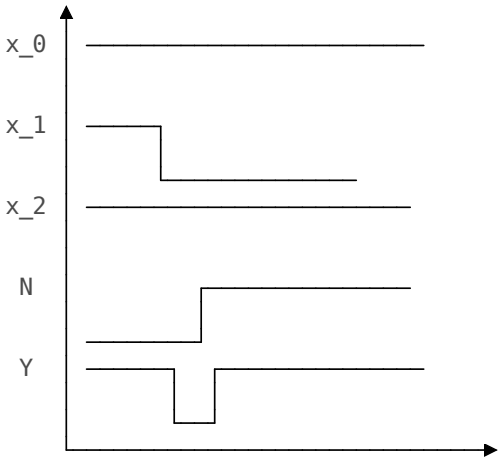
Konsequenz

Schaltwerk für die Umsetzung der Schaltfunktion $y = x_0 x_1 + \overline{x}_1 x_2$



Die Schaltung befindet sich zunächst in der Konfiguration $x_0 = x_1 = x_2 = 1$. Entsprechend liegt für y ebenfalls eine 1 vor.

Nun wechselt x_1 auf "0". Der Inverter benötigt allerdings eine gewisse Zeit, um den Ausgang von "1" in eine „0“ umzusetzen. Für kurze Zeit ist sowohl $x_1 = 0$, als auch der Wert hinter dem Inverter "0". Entsprechend schaltet "AND_2" nicht durch, sondern gibt eine "0" weiter an den Oder-Baustein.



.

Analyse im Karnaugh-Veitch-Diagramm

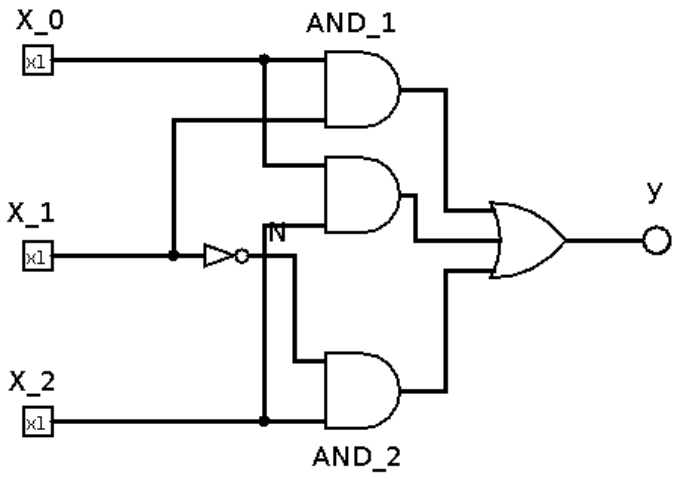
Das potentielle Auftreten eines Glitches wird im Karnaugh-Veitch Diagramm anhand nebeneinander liegender, nicht durch Überlappungen verbundener Schleifen deutlich.

	$\overline{x}_1\,\overline{x}_0$	$\overline{x}_1 x_0$	$x_1 x_0$	$x_1 \overline{x}_0$
$\overline{x}_2$	0	0	1	0
x_2	1	1	1	0

Lösungsansätze

1. Integration Glitch-Freier Pfade in dem wir die Überlappung im Karnaugh-Veitch Diagramm über den Minimierungsgedanken stellen.

$y = x_0 x_1 + \overline{x}_1 x_2 + x_0 x_2$



2. Synchronisation gültiger Zustände