Physikalisches Praktikum für Vorgerückte

an der ETH Zürich

vorgelegt von

Mattia Rigotti

mrigotti@student.ethz.ch

14.02.2003

Digitale Elektronik

Versuchsprotokoll

Inhaltverzeichnis

1. Zusammenfassung

Die vorliegende Arbeit befasst sich mit der Vereinfachung (Minimierung) von Logikfunktionen durch *Karnaugh-Veitch-Diagramme* („KV-Diagramme“) und durch das gleichbedeutende *Quine-McCluskey-Verfahren*. Dank dieser Methoden wird es möglich, elektronische Schaltkreise aufzubauen, die eine wesentlich kleinere Anzahl von Schaltelemente brauchen als die entsprechenden Darstellungen in *Kanonische Normalform*. Diese Verfahren erweisen also tatsächliche praktische Anwendung in der Digitalelektronik, die später in mit konkreten Beispiele erläutert werden.

2. Boolesche Algebren

Im Gegenteil zur Analogelektronik beruht die Digitalelektronik auf einer relativ einfachen aber sehr mächtigen Theorie, die Theorie der Boolschen Algebra, auch Schaltalgebra genannt. In diesem Kapitel erläutern wir diese theoretischen Grundlagen der Digitalelektronik.

2.1. Logikvariablen und Logikfunktionen

In der Digitaltechnik hat man mit besonderen Variablen zu tun, sogenannte *Logikvariablen*, die nur die Werte 0 oder 1 (falsch oder wahr, offen oder geschlossen...) annehmen können.
Analog können Logikfunktionen (auch Binärfunktionen oder Schaltfunktionen genannt) nur die Werte 0 oder 1 annehmen, in Abhängigkeit von den Eingangsvariablen:

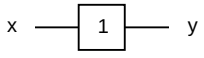
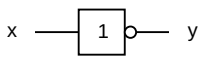
$$y = f(x_1, x_2, \ldots, x_n) \qquad \text{wobei } x_i, y \in \{0, 1\}$$

ist eine *n-stellige Logikfunktion*.
Diese Funktionen können durch Tabellen dargestellt werden, sogennante *Wahrheitstabellen*, in denen die 2^n Kombinationen der n Eingangsvariablen aufgelistet sind.
In der folgenden Tabelle 2-1 sind alle *Einstelligen Binärfunktionen* eingetragen, d.h. alle Logikfunktionen des Typs:

$$y = f(x) \qquad \text{wobei } x, y \in \{0, 1\},$$

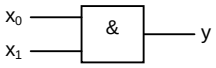
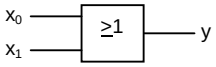
(insgesamt $2^2 = 4$ Funktionen) zusammen mit dem entsprechenden Schaltzeichen, das in der Digitaltechnik angewandt wird.

Tabelle 2-1: Einstellige Binärfunktionen

Wahrheitstabelle	Funktion	Schaltzeichen	Name
x | y 0 | 0 1 | 0	y = 0		
x | y 0 | 0 1 | 0	y = x		
x | y 0 | 0 1 | 0	$y = \overline{x}$		NOT, Komplement, Negation
x | y 0 | 1 1 | 1	y = 1		

(Gelegentlich werden wir die Negation von x gemäss der alten Notation auch mit $\neg x$ bezeichnen: $\neg x := \bar{x}$).
Von grosser Bedeutung sind die *zweistelligen Binärfunktionen*, weil jede Binärfunktion mit mehreren Eingangsvariablen auf eine Komposition von diesen zurückgeführt werden kann.
Technisch wichtig sind die Grundverknüpfungen UND (AND) und ODER (OR), die in der Tabelle 2-2 definiert sind.

Tabelle 2-2: Grundverknüpfungen UND und ODER

Wahrheitstabelle	Funktion	Schaltzeichen	Name
x_1 x_0 y 0 0 0 0 1 0 1 0 0 1 1 1	$y = x_0 \cdot x_1$		UND, AND, Konjunktion
x_1 x_0 y 0 0 0 0 1 0 1 0 0 1 1 1	$y = x_0 + x_1$		ODER, OR, Disjunktion

Es ist einfach zu sehen, dass es $2^4 = 16$ verschiedene 2-stellige Binärefuntionen gibt:

Tabelle 2-3: Wahrheitstabelle für zweistellige Binärfunktionen

x_1	x_0	y
0	0	y(0,0)
0	1	y(0,1)
1	0	y(1,0)
1	1	y(1,1)

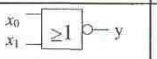
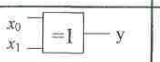
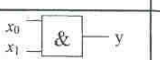
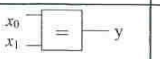
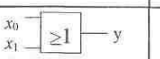
Jedem der vier Bildpunkte y(0,0),..., y(1,1) kann man nämlich einer der beiden Werte 0 oder 1 zuordnen. In der Tabelle 2-4 sind alle 2-stelligen Binärfunktionen eingetragen. Diese Tabelle kann auch als Beweis der Tatsache betrachtet werden, dass alle 2-stelligen Funktionen durch Kombination von den Grundverknüpfungen AND, OR, NOT dargestellt werden können. Dieses Triplett ist also *logisch vollständig*, denn sie dient als Basis der Darstellung jeder möglichen Logikfunktion (mehrstellige Logikfunktionen können nämlich durch Verknüpfen von 2-stelligen Funktionen dargestellt werden).
Auch das Duo NAND, NOR (Wahrheitstafel in der Tabelle 2-4) ist logisch vollständig, weil es durch Kombination dieser beiden Funktionen möglich ist, die drei Funktionen AND, OR, NOT darzustellen:

- $\neg x = \neg (x \cdot x) = x \text{ NAND } x$ (oder auch: $\neg x = \neg (x + x) = x \text{ NOR } x$)
- $x_0 \cdot x_1 = \neg(\neg x_0 + \neg x_1) = \neg x_0 \text{ NOR } \neg x_1 = (x_0 \text{ NAND } x_0) \text{ NOR } (x_1 \text{ NAND } x_1)$
- $x_0 + x_1 = \neg(\neg x_0 \cdot \neg x_1) = \neg x_0 \text{ NAND } \neg x_1 = (x_0 \text{ NAND } x_0) \text{ NAND } (x_1 \text{ NAND } x_1)$.

Mehr als dies: es ergibt sich sogar, dass allein die NAND Operation und allein die NOR Operation logisch vollständig sind, weil NOR als Kombination von NAND dargestellt werden kann, und umgekehrt:

- x_0 NOR $x_1 = \neg(x_0 + x_1) = (\neg x_0 \cdot \neg x_1) = \neg(\neg x_0$ NAND $\neg x_1) = \neg((x_0$ NAND $x_0)$ NAND $(x_1$ NAND $x_1))$ $= ((x_0$ NAND $x_0)$ NAND $(x_1$ NAND $x_1))$ NAND $((x_0$ NAND $x_0)$ NAND $(x_1$ NAND $x_1))$
- x_0 NAND $x_1 = \neg(x_0 \cdot x_1) = (\neg x_0 + \neg x_1) = \neg(\neg x_0$ NOR $\neg x_1) = \neg((x_0$ NOR $x_0)$ NOR $(x_1$ NOR $x_1)) = ((x_0$ NOR $x_0)$ NOR $(x_1$ NOR $x_1))$ NOR $((x_0$ NOR $x_0)$ NOR $(x_1$ NOR $x_1))$

Tabelle 2-4: 2-stellige Binärfunktionen: Wahrheitstabelle, Darstellung durch (AND, NOT, OR), Schaltsymbol und Funktionsname

Wahrheitstabelle					Funktion	Schaltsymbol	Name
x_0	1	0	1	0			
x_1	1	1	0	0			
y	0	0	0	0	$y = 0$		Null
y	0	0	0	1	$y = \overline{x_0 + x_1}$ $y = (x_0 \overline{\vee} x_1)$		NOR
y	0	0	1	0	$y = x_0 \cdot \overline{x_1}$		Inhibition
y	0	0	1	1	$y = \overline{x_1}$		Komplement
y	0	1	0	0	$y = \overline{x_0} \cdot x_1$		Inhibition
y	0	1	0	1	$y = \overline{x_0}$		Komplement
y	0	1	1	0	$y = (\overline{x_0} \cdot x_1) + (x_0 \cdot \overline{x_1})$ $y = x_0 \leftrightarrow x_1$		EXOR
y	0	1	1	1	$y = \overline{(x_0 \cdot x_1)}$ $y = (x_0 \overline{\wedge} x_1)$		NAND
y	1	0	0	0	$y = x_0 \cdot x_1$		UND, AND
y	1	0	0	1	$y = (x_0 \cdot x_1) + (\overline{x_0} \cdot \overline{x_1})$ $y = x_0 \leftrightarrow x_1$		Äquivalenz
y	1	0	1	0	$y = x_0$		Identität
y	1	0	1	1	$y = x_0 + \overline{x_1}$		Implikation
y	1	1	0	0	$y = x_1$		Identität
y	1	1	0	1	$y = x_1 + \overline{x_0}$		Implikation
y	1	1	1	0	$y = x_0 + x_1$		ODER, OR
y	1	1	1	1	$y = 1$		Eins

Das Resultat dieser Überlegung ist, dass die einzelne NAND Operation allein jede mögliche Logikfunktion darstellen kann (dasselbe gilt für die NOR Operation). Dies ist in der Digitaltechnik sehr wichtig, denn es besagt, dass jedes beliebige logische System aus einem einzelnen Schaltelement entstehen kann.

2.2. Rechenregeln

Wir wollen jetzt die Rechenregeln betrachten, die wir beim Vereinfachen von logischen Funktionen gebrauchen werden. Diese Regeln können einfach aus den Wahrheitstabellen abgeleitet werden[1].

1. Kommutativgesetz:
   $x_0 \cdot x_1 = x_1 \cdot x_0$, $x_0 + x_1 = x_1 + x_0$.
2. Assoziativgesetz:
   $(x_0 \cdot x_1) \cdot x_2 = x_0 \cdot (x_1 \cdot x_2)$, $(x_0 + x_1) + x_2 = x_0 + (x_1 + x_2)$.
3. Distributivgesetz:
   $x_0 \cdot (x_1 + x_2) = (x_0 \cdot x_1) + (x_0 \cdot x_2)$, $x_0 + (x_1 \cdot x_2) = (x_0 + x_1) \cdot (x_0 + x_2)$.
4. Absorptionsgesetz:
   $x_0 \cdot (x_0 + x_1) = x_0$, $x_0 + (x_0 \cdot x_1) = x_0$.
5. Existenz der neutralen Elemente:
   $x_0 \cdot 1 = x_0$, $x_0 + 0 = x_0$.
6. Existenz der komplementären Elemente:
   $x_0 \cdot \overline{x}_0 = 0$, $x_0 + \overline{x}_0 = 1$.
7. De Morgansche Regeln:
   $x_0 \cdot x_1 = \overline{(\overline{x_0} + \overline{x_1})}$, $x_0 + x_1 = \overline{(\overline{x_0} \cdot \overline{x_1})}$.

In diesen Regeln erkennt man sofort eine gewisse Symmetrie: gilt nämlich ein Gesetz, so gilt auch das Gesetz, das man erhält, indem man AND mit OR und die Konstanten mit 1 vertauscht. Das so entstandene Gesetz bezeichnet man al das *duale Gesetz*. Analog bezeichnet man eine Funktion F', die aus F durch Vertauschen von AND mit OR und von 1 mit 0 entsteht, als die zu F duale Funktion.

[1] Diese Rechenregeln können auch als Axiome betrachtet werden, die eine sogenannte Boolesche Algebra definieren. Daraus kann man dann die Wahrheitstafeln der Verknüpfungen $\wedge$ und $\vee$ ableiten. Es stellt sich nämlich heraus, dass nur zwei Elemente zu dieser Algebra gehören können (eben 0 und 1) und zwar das neutrale Element bzgl. $\vee$ bzw. $\wedge$.

2.3. Kanonische disjunktive Normalform (KDNF)

In diesem Abschnitt wollen wir Methoden betrachten, die uns ermöglichen werden, aus einer Wahrheitstabelle die entsprechende Logikfunktion als Kombination von OR und AND Operatoren auszudrücken.
Eine mögliche Methode ist die sogennante *Kanonische disjunktive Normalform (KDNF)*. Einen logischen Ausdruck in KDNF sieht als Disjunktion von sogenannte *Minterme* m_i, die ihrerseits aus Konjunktionen der Einträge bestehen.
Wir machen ein Beispiel, um zu zeigen wie dieses Verfahren funktioniert. Betrachten wir die 3-stellige Funktion, deren Wahrheitstafel in der Tabelle 2-5 dargestellt ist.

Tabelle 2-5: Wahrheitstabelle für Beispiel zur KDNF

n	x_2	x_1	x_0	y
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

Wir betrachten zunächst die Zeilen (sogennante Eingangsvektoren X_n), für die die Funktion $y = f(X)$ den Wert 1 annimmt, dh. die Eingangsvektore X_n, für die $f(X_n) = 1$ gilt. In unserem Beispiel sind das X_1, X_3, X_4, X_7. Nun bilden wir die Minterme, indem wir eine Konjunktion der Elemente x_i, die genau für diesen Eingangsvektor den Wert 1 annimmt. Machen wir das Beispiel für X_1:

$$m_1 = \neg x_2 \cdot \neg x_1 \cdot x_0.$$

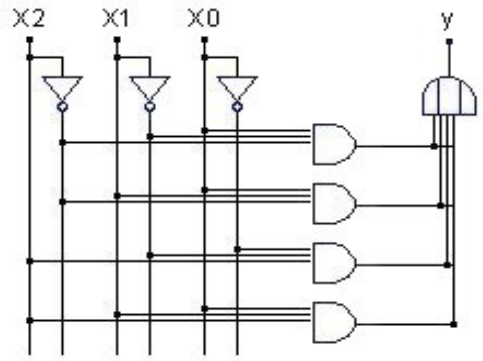
Die Minterme werden auch *Vollkonjunktionen* genannt, wiel sie immer alle Variablen enthalten. In einem Minterm kommen die Eingangsvariablen invertiert oder nichtinvertiert vor, je nachdem, ob die entsprechende Eingangsvariable 1 oder 0 ist. Für unseres Beispiel sind die anderen Minterme:

$$m_3 = \neg x_2 \cdot x_1 \cdot x_0$$
$$m_4 = x_2 \cdot \neg x_1 \cdot \neg x_0$$
$$m_7 = x_2 \cdot x_1 \cdot x_0.$$

Ein Minterm hat also für genau einen bestimmten Fall der Eingangsvariablen den Wert 1.
Die Funktion y erhalten wir schliesslich indem wir sie durch die Disjunktion der Minterme darstellen. Dies funktioniert, denn die Funktion bekommt den Wert 1, wenn mindestens einer der Minterme gleich 1 wird.
In unserem Beispiel sieht also die Funktion in KDNF so aus:

$$y = (\neg x_2 \cdot \neg x_1 \cdot x_0) + (\neg x_2 \cdot x_1 \cdot x_0) + (x_2 \cdot \neg x_1 \cdot \neg x_0) + (x_2 \cdot x_1 \cdot x_0).$$

Bild 2-1: Schaltnetz für die Realisierung der KDNF der Funktion in der Tabelle 2-5



2.4. Kanonische konjunktive Normalform (KKNF)

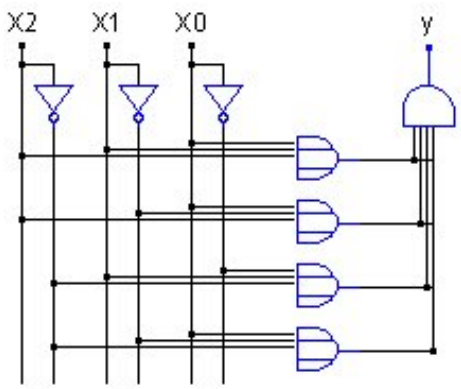
Eine Alternative ist die *Kanonische konjunktive Normalform (KKNF)*. Sie wird bestimmt, indem wir diesmal die Eingangsvektoren betrachten, für die $f(X_n) = 0$ gilt. Bei der Funktion in der Tabelle 2-5 wären das X_0, X_2, X_5, X_6.
Es werden die sogenannten *Maxterme* M_i, als Disjunktion der Einträge so gebildet, dass sie genau dann gleich 0 sind, wenn der entsprechende Eingangsvektor X_i anliegt. Für unseres Beispiel:

$$M_0 = x_2 + x_1 + x_0$$
$$M_2 = x_2 + \neg x_1 + x_0$$
$$M_5 = \neg x_2 + x_1 + \neg x_0$$
$$M_6 = \neg x_2 + \neg x_1 + x_0.$$

Die gesamte Funktion erhalten wir durch Konjunktion der Maxterme. Dies funktioniert, denn der Funktionswert darf genau dann 0 sein, wenn mindestens einer der Maxterme gleich 0 ist. Die KKNF unseres Beispiels wäre dann:

$$y = (x_2 + x_1 + x_0) \cdot (x_2 + \neg x_1 + x_0) \cdot (\neg x_2 + x_1 + \neg x_0) \cdot (\neg x_2 + \neg x_1 + x_0).$$

Bild 2-2: Schaltnetz für die Realisierung der KDNF der Funktion in der Tabelle 2-5



2.5. Shannonsche Regel: Zusammenhang zwischen KDNF und KKNF

Die Shannonsche Regel ist eine einfache aber wichtige Regel, die beim Arbeiten mit den Normalformen sehr nützlich sein kann. Sie ist eine Art Verallgemeinerung der de Morgansche Gesetze. Sie lautet:

Zu einer beliebigen Booleschen Funktion $y = f(x_0, x_1, \ldots, x_n, \cdot, +, 1, 0)$ ist die invertierte Funktion $\neg y = f(\neg x_0, \neg x_1, \ldots, \neg x_n, +, \cdot, 0, 1)$.

D.h.: um eine Funktion zu invertieren, müssen wir alle Variablen invertieren und alle Operatoren durch ihre dualen ersetzen.
Die Funktion

$$y = (\neg x_2 \cdot \neg x_1 \cdot x_0) + (\neg x_2 \cdot x_1 \cdot x_0)$$

wird also z.B. so invertiert:

$$\neg y = (x_2 + x_1 + \neg x_0) \cdot (x_2 + \neg x_1 + \neg x_0).$$

Wir können also die KKNF einer Funktion einfach bestimmen, indem wir die Shannonsche Regel auf der KDNF der Inverse dieser Funktion anwenden.

3. Minimierung von Schaltnetzen

Die konjunktive und die disjunktive Normal-Formen ermöglichen uns eine Schaltfunktion aus seiner Wahrheitstabelle darzustellen. Der erhaltene Ausdruck ist aber im allg. nicht optimiert, d.h. es kann sein, dass es einen einfacheren logischen Ausdruck gibt, der derselben Wahrheitstabelle, also derselben Logikfunktion entspricht.
Als Beispiel betrachten wir die Logikfunktion

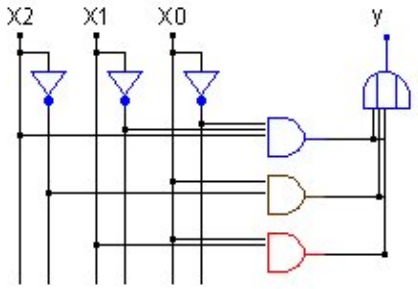
$$f\,(x_2, x_1, x_0) = (\neg x_2 \cdot \neg x_1 \cdot x_0) + (\neg x_2 \cdot x_1 \cdot x_0) + (x_2 \cdot \neg x_1 \cdot \neg x_0) + (x_2 \cdot x_1 \cdot x_0)$$

aus Abschnitt 2.3. Mit einer Wahrheitstabelle ist es einfach zu sehen, dass diese Funktion äquivalent ist zu:

$$f'(x_2, x_1, x_0) = (x_2 \cdot \neg x_1 \cdot \neg x_0) + (\neg x_2 \cdot x_0) + (\, x_1 \cdot x_0), \qquad (3.1)$$

dessen Schaltnetz im Bild 3-1 zu sehen ist.

Bild 3-1: Minimiertes Schaltnetz für die Realisierung Funktion in der Tabelle 2-5 (vgl. mit Bild 2-1).



f' stellt dieselbe Funktion wie f dar, ist aber ein einfacherer logischen Ausdruck und dies widerspiegelt sich in der Vereinfachung des Schaltnetzes (vgl. Bild 2-1 mit Bild 3-1).

3.1. Minimierung mit Karnaugh-Veitch-Diagramme (KV-Diagramme)

Eine sehr verbreitete Minimierungsmethode ist das sogenannte KV-Diagramm, das sich für den Entwurf von Hand sehr gut eignet. Dieses Verfahren beruht auf der Regel

$$(x_0 \cdot x_1) + (x_0 \cdot \neg x_1) = x_0 \cdot (x_1 \cdot \neg x_1) = x_0 \cdot 1 = x_0,$$

also:

$$(x_0 \cdot x_1) + (x_0 \cdot \neg x_1) = x_0 \qquad (3.2)$$

Betrachten wir z.B. wieder die Logikfunktion aus Abschnitt 2.3:

$$f(x_2, x_1, x_0) = (\neg x_2 \cdot \neg x_1 \cdot x_0) + (\neg x_2 \cdot x_1 \cdot x_0) + (x_2 \cdot \neg x_1 \cdot \neg x_0) + (x_2 \cdot x_1 \cdot x_0)$$

und wenden wir darauf nach einer kleinen Umformung die Gl. (3.2):

$$\begin{aligned} f(x_2, x_1, x_0) &= (x_2 \cdot \neg x_1 \cdot \neg x_0) + (\neg x_2 \cdot \neg x_1 \cdot x_0) + (\neg x_2 \cdot x_1 \cdot x_0) + (x_2 \cdot x_1 \cdot x_0) \\ &= (x_2 \cdot \neg x_1 \cdot \neg x_0) + (\neg x_1 \cdot (\neg x_2 \cdot x_0)) + (x_1 \cdot (\neg x_2 \cdot x_0)) + (\neg x_2 \cdot (x_1 \cdot x_0)) + \\ &\quad + (x_2 \cdot (x_1 \cdot x_0)) = (x_2 \cdot \neg x_1 \cdot \neg x_0) + (\neg x_2 \cdot x_0) + (x_1 \cdot x_0) \end{aligned}$$

Damit erhalten wir genau die minimierte Funktion f' aus (3.1). Wir stellen fest, dass dank Gl. (3.2) diejenigen Minterme „zusammengeschmolzen" werden können, die sich in genau einer Stelle unterscheiden. Dann kann man z.B. das folgende schreiben:

$$(\neg x_2 \cdot x_1 \cdot x_0) + (x_2 \cdot x_1 \cdot x_0) = (x_1 \cdot x_0).$$

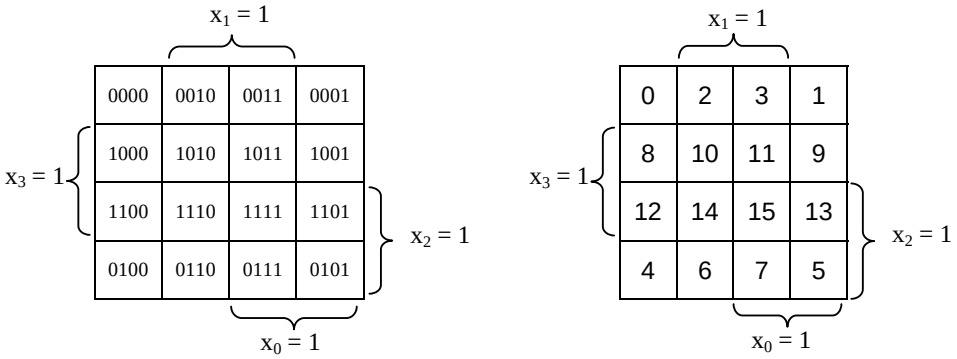
Das KV-Diagramm ermöglicht eine gewisse „Automatisierung" dieser Art von Umformung. Wir wollen dieses Verfahren durch ein konkretes Beispiel erläutern.
Nehmen wir die 4-stellige Logikfunktion, deren Wahrheitstafel in der Tabelle 3-1 aufgelistet ist. Diese Funktion ist nur bis den Dezimalwert 9 definiert. Wir seien nämlich an den Werten der Funktion nicht interessiert, die sie zwischen den Dezimalzahlen 10 und 15 annimmt („don't care" Terme). Wir sprechen dann von *unvollständig spezifizierte Funktionen*.

Tabelle 3-1: Wahrheitstafel für Beispiel zum KV-Diagramm (die Werte von 10 bis 15 können beliebig sein).

N	x_3	x_2	x_1	x_0	y
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
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	0

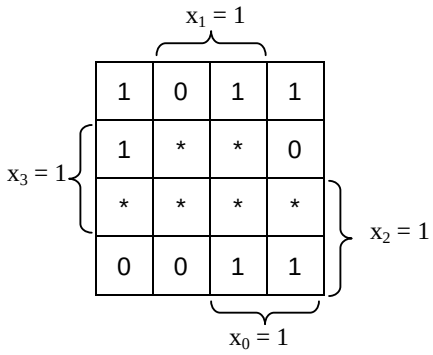
Jetzt stellen wir Diagramme auf, wo diejenigen Minterme benachbart sind, die sich in genau einer Stelle unterscheiden, damit wir Gl. (3.2) benutzen können. Diese Diagramme sind die KV-Diagramme.

Bild 3-2: KV-Diagramme für 4-Eingangsvariablen a) mit binärer Bezeichnung der Felder, b) mit dezimaler Bezeichnung der Felder



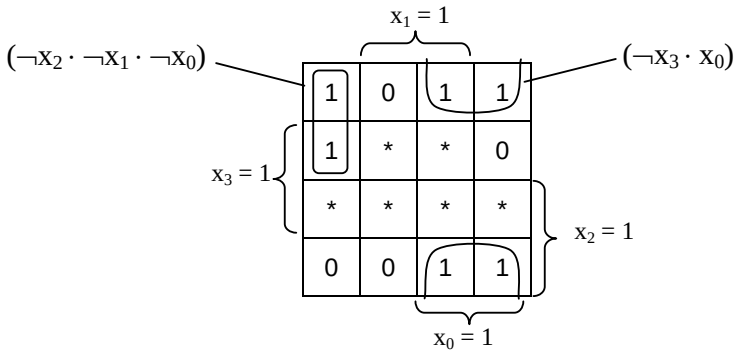
In diesem Diagramm werden die Minterme unserer Funktion eingetragen, wobei die „don't care" Terme mit einem Stern (*) bezeichnet werden (s. Bild 3-3).

Bild 3-3: KV-Diagramme der Minterme der Funktion aus Tabelle 3-1.



Jetzt können wir die benachbarten Felder nach Gl. (3.2) zusammenfassen, weil sie sich immer nur in einer Variablen unterscheiden. Wir müssen möglichst grosse Gebiete mit einer 1 bilden, die zusammenhängend, konvex und mit 1, 2, 4, 8, usw. Feldern sein müssen. Man darf sich auch eine gedankliche Verbindung zwischen rechter und linker, und oberer und unterer Seite vorstellen, weil die binären Darstellungen von z.B. 8 und 10 (obere und untere Seite) sich auch in genau einer Stelle unterscheiden. Die *-Felder können beim Bilden eines Gebietes mitgenommen werden. Eines der so erhaltenen Gebiete stellt einen logischen Ausdruck dar, der durch eine Konjunktion der Eingangsvariablen beschrieben wird und den wir *Implikant* nennen wollen. Je grösserer das Gebiet ist, desto weniger Variablen hat der Implikant, d.h. desto einfacher wird der minimierte Ausdruck.

Bild 3-4: KV-Diagramme mit den Mintermen der Funktion aus Tabelle 3-1.



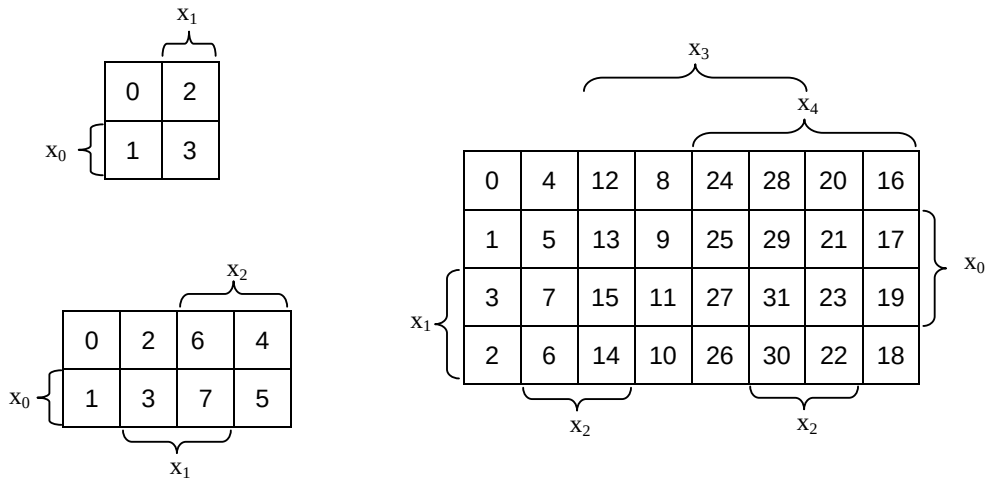
Die Logikfunktion wird dann als Disjunktion von Implikanten dargestellt, die alle ‚1' überdecken. In unseren Beispiel im Bild 3-4 sieht man also, dass die Funktion in der Tabelle 3-1 mit dem minimierten Ausdruck

$$y = (\neg x_2 \cdot \neg x_1 \cdot \neg x_0) + (\neg x_3 \cdot x_0)$$

gegeben ist, wobei die KDNF folgenderweise ausgesehen hätte:

$$y = (\neg x_3 \cdot \neg x_2 \cdot \neg x_1 \cdot \neg x_0) + (\neg x_3 \cdot \neg x_2 \cdot \neg x_1 \cdot x_0) + (\neg x_3 \cdot \neg x_2 \cdot x_1 \cdot x_0) + (\neg x_3 \cdot x_2 \cdot \neg x_1 \cdot x_0) + (\neg x_3 \cdot x_2 \cdot x_1 \cdot x_0) + (x_3 \cdot \neg x_2 \cdot \neg x_1 \cdot \neg x_0).$$

Bild 3-5: KV-Diagramme für 2,3 und 5 Eingansvariablen (mit mehr als 5 Variablen werden KV-Diagramme zu unübersichtlich, um verwendet zu werden)



3.2. Minimierung mit dem Quine-McCluskey-Verfahren

Das Quine-McCluskey-Verfahren ist sehr ähnlich zur Methode der KV-Diagramme. Es beruht auch nämlich auf der Gl. (3.2) zur Minimierung von logischen Ausdrucke:

$$(x_0 \cdot x_1) + (x_0 \cdot \neg x_1) = x_0. \qquad (3.2)$$

Es hat die Vorteile, dass es sich für beliebig viele Eingangsvariablen ziemlich einfach in einem Computerprogramm implementieren lässt, auch weil es nicht auf einem „graphischen Vorgehen“ wie die KV-Diagramme beruht.
Es lohnt sich die folgenden Abkürzungen einzuführen: das Zeichen ‚·’ werden wir weglassen („$(x_0 \cdot x_1) + (x_0 \cdot \neg x_1)$“ wird „$x_0x_1 + x_0\neg x_1$“). Ausserdem stellen wir die Minterme einer Funktion in binäräquivalenter Form dar: für eine im Minterm vorkommende Variable wird eine 1 gesetzt, für ihre Negation eine 0 und für eine nicht vorkommende ein Strich (-). Ein Beispiel: $x_3x_2\neg x_0$ wird: 11-0.
Wir wollen das QMC-Verfahren mittels der Minimierung von der Funktion in der Tabelle 3-1 erklären. Zuerst, genau wie beim KDNF, betrachten wir nur die Minterme, deren Bilder eine 1 sind, und wir ordnen sie in Gruppen, die ihrer Anzahl von Eins-Elementen des Binäräquivalent entsprechen (s. Tabelle 3-2).

Tabelle 3-2: Ordnung der Minterme nach Gruppen mit gleich vielen 1-Elementen

N	x_3	x_2	x_1	x_0	Gruppe	
0	0	0	0	0	0	✓
1	0	0	0	1	1	✓
5	0	1	0	1	1	✓
8	1	0	0	0	1	✓
3	0	0	1	1	2	✓
7	0	1	1	1	3	✓

Jetzt kann unsere Optimierung beginnen. In der nächsten Tabelle fassen wir die Minterme als aufeinanderfolgende Gruppen auf, die sich nur in einer Stelle unterscheiden. Die Stelle, in der sich die Elemente unterscheiden, wird durch einen Strich (-) markiert.
Wir können z.B. die Minterme 0 und 1 zusammenfassen, weil sie sich nur an der Stelle x_0 unterschieden.

Alle Terme, die sich zusammenfassen lassen, werden mit einem ✓ markiert. Nicht markierte Minterme sind Implikante, die in der minimierten Funktion erscheinen werden (sogenannte Primimplikante).

Dieses Verfahren wird dann solange wiederholt, bis es nicht mehr möglich ist. Sind die Binäräquivalente mehrerer Terme gleich, so werden die überflüssigen Terme gestrichen.

Tabelle 3-3: Zusammenfassung der Minterme nach Gruppen mit gleicher Anzahl von 1- Elementen (1. Schritt)

N	x_3	x_2	x_1	x_0	Gruppe	
0,1	0	0	0	-	0	
0,8	-	0	0	0	0	
1,5	0	-	0	1	1	✓
3,7	0	-	1	1	2	✓

Tabelle 3-4: Zusammenfassung der Minterme nach Gruppen mit gleicher Anzahl von 1- Elementen (2. Schritt)

N	x_3	x_2	x_1	x_0	Gruppe
1,5,3,7	0	-	-	1	1

Wenn dieses Verfahren am Ende ist, müssen wir die Primimplikante (d.h. die Minterme, die nicht als gehackt wurden) klassifizieren. Sie werden in *Kern-Primimplikanten*, *absolut eliminierbar* und *relativ eliminierbaren Primimplikanten* aufgeteilt.
Wir stellen eine weitere Tabelle auf, auf deren Ordinate die Minterme und auf deren Abszisse die Primimplikanten aufgetragen werden. Die Minterme, die in einem Primimplikant erscheinen, werden mit einem × markert (s. Tabelle 3-5).

Tabelle 3-5: Primimplikantentfel für unseres Beispiel

	0	**1**	**3**	**5**	**7**	**8**
0,1	×	×				
0,8	×					×
1,5,3,7		×	×	×	×	

Befindet sich in einer Spalte nur ein ×, so ist der dazugehörige Primimplikant ein Kern-Primimplikant, d.h. er wird in der minimierten Funktion erscheinen. Die Minterme, die durch ihn abgedeckt werden, werden mit einem Kreis ⊗ markiert, und zwar auch in den anderen Zeilen. Die Primimplikanten, die nicht Kern-Primimplikant sind, aber deren allen Minterme am Ende dieses Prozesses markiert sind, sind absolut eliminierbar und können „weggeworfen werden“. Die nicht markierten Kreuze sind diejenigen, die in einem relativ eliminierbaren Primimplikanten auftreten, und wir müssen sie so wählen, dass mit den Kern-Primimplikanten alle Minterme überdeckt werden. In unserem Beispiel in Tabelle 3-6 erscheinen nur Kern-Primimplikanten (1,5,3,7; 0,8) und absolut eliminierbare Primimplikante (0,1).

Tabelle 3-6: Primimplikantentfel mit markierten Termen (⊗)

	0	**1**	**3**	**5**	**7**	**8**
0,1	⊗	⊗				
0,8	⊗					⊗
1,5,3,7		⊗	⊗	⊗	⊗	

Tabelle 3-7: Zuordnung der Implikanten

N	x_3	x_2	x_1	x_0	Implikant
0,1	0	0	0	-	$\neg x_3 \neg x_2 \neg x_1$
0,8	-	0	0	0	$\neg x_2 \neg x_1 \neg x_0$
1,5,3,7	0	-	-	1	$\neg x_3 x_0$

Unsere minimierte Logifunktion stellen wir also mit Hilfe der Implikanten 1,5,3,7 und 0,8 dar und wir erhalten:

$$y = (\neg x_2 \cdot \neg x_1 \cdot \neg x_0) + (\neg x_3 \cdot x_0).$$

Wir stellen fest, dass dies dasselbe Resultat ist, wie bei der KV-Diagramm-Methode. Es konnte aber auch nicht anders sein, da sie auf denselben theoretischen Überlegungen stützen.

4. Anwendung: ein Synchronzähler

In diesem kurzen Kapitel wollen wir als praktische Anwendung eine Logikfunktion entwerfen, die wir in der Konstruktion eines Synchronzählers benutzen werden.
Wir wollen einen Zähler konstruieren, der die Signale eines Taktes C sequentiell von 0 bis 9 zählt, und wenn er der 9. gezählt hat, soll er von 0 wieder anfangen. Unser Takt wird eine folge von Signale 0,1,0,1,… aussenden, und zwischen jeder Veränderung des Taktes wird eine gewisse Zeitspanne verlaufen.
Zuerst stellen wir fest, dass wir mindestens int($\log_2(9)$) = 4 Bits brauchen, denn wir müssen natürlich bis 9 zählen können. Diese 4 Bits werden 4 Eingängen Q_3,…, Q_0 einer Logikfunktion entsprechen, deren Konfiguration ihrerseits die Zahl des gegenwärtigen Taktes darstellt (z.B. Q_3=0, Q_2=0, Q_1=1, Q_0=1 entspricht $0011_2 = 3$).
Die Taktik beruht auf zwei Etappen:

1. Wir stellen eine 4-stellige Funktion JK(Q_3,Q_2,Q_1,Q_0) auf, die, gegeben eine Konfiguration der Eingänge (d.h. die Zahl des gegenwärtigen Taktes), 4 Ausgänge JK_3, JK_2, JK_1, JK_0 ergibt. Diese sagen uns, welche der Eingänge Q_i im nächsten Takt ihren Wert wechseln müssen (JK_i=1) und welche ihren Wert behalten müssen (JK_i=0). Das Ganze wird nach der Tabelle 4-1 funktionieren.

Tabelle 4-1: Warheitstabelle der Funktion JK($Q_{3,...,}Q_0$). JK_i =1 in n bedeutet $Q_i^{n+1}=\neg Q_i^n$, JK_i=0 bedeutet $Q_i^{n+1}=Q_i^n$.

n	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

J_3	J_2	J_1	J_0
0	0	0	1
0	0	1	1
0	0	0	1
0	1	1	1
0	0	0	1
0	0	1	1
0	0	0	1
1	1	1	1
0	0	0	1
1	0	0	1

2. Wir stellen eine 3-stellige rückgekoppelte Funktion Q_i^{m+1} = FF(C,JK_i,Q_i^m) auf, abhängig vom Takt C und von einem Eingang Q_i^m. Dieser Eingang Q_i^m ist der Ausgang der Funktion im vorherigen Takt m. Analog wird der Ausgang Q_i^{m+1} im nächsten Takt als Eingang der Funktion benutzt. Die Funktion FF modifiziert einfach den Wert von Q_i^m. Diese soll bei C=1 und JK_i=1 Q_i^m umklappen: $Q_i^{m+1} = \neg Q_i^m$, und bei C=1 und JK_i=0 Q_i^m unverändert lassen: $Q_i^{m+1} = Q_i^m$ (s. Tabelle 4-2).
   Diese Arbeit wird normalerweise mit einem JK-Master-Slave Flip-Flop ganz einfach ausgeführt. Da aber wir nicht alle Funktionen dieses Schaltelementes benötigen, wollen wir mit der einfacheren selbst vorbereitete Funktion FF vorgehen, damit wir auch die Behandlung des Flip-Flops vermeiden können.

Das Zusammensetzen dieser beiden Elementen ergibt offenbar unseren 9-stelligen Zähler.

Tabelle 4-2: Die Funktion FF muss bei C=1 das folgende machen: Q_i umklappen falls JK_i=1, oder Q_i unverändert lassen falls JK_i=0 (wie auch bei C=0).

C	JK_i	Q_i^{m+1}
0	*	Q_i^m
1	0	Q_i^m
1	1	$\neg Q_i^m$

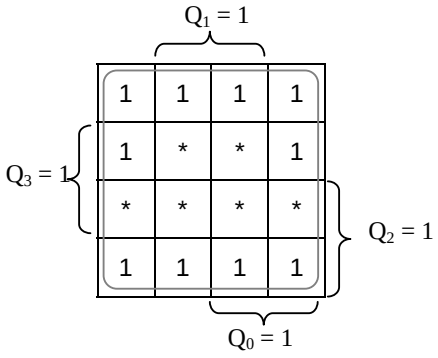
Tabelle 4-3: Die vollständige Wahrheitstafel der Funktion Q_i^{m+1}=FF(C,JK_i,Q_i^m) (äquivalent zur Tabelle 4-2)

C	JK_i	Q_i^m	Q_i^{m+1}
0	*	0	0
0	*	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Nachdem wir die Wahrheitstafeln der Funktion JK (Tabelle 4-1) und der rückgekoppelte Funktion FF (Tabelle 4-3) aufgestellt haben, können wir sie endlich dank unserer KV- oder Quine-McCluskey-Diagramme vereinfachen.
Die Optimierung mittels KV-Diagramme der Funktion JK ergibt folgende Resultate:

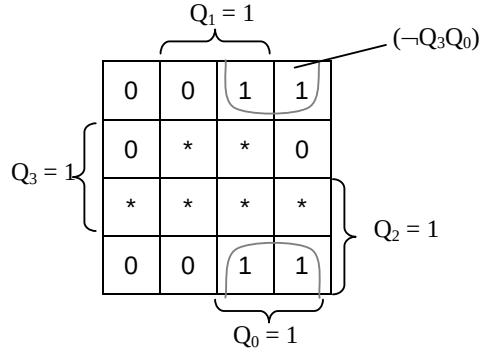
KV-Diagramme für die Funktion JK:

JK_0:

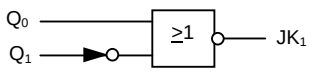


JK_0=1

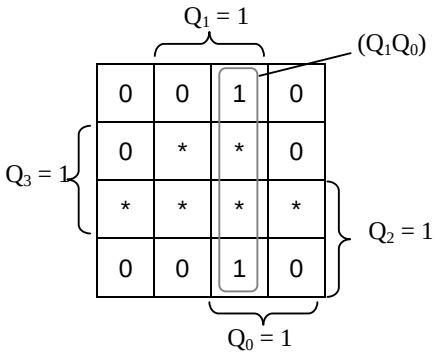
JK_1:



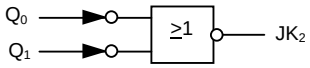
JK_1=¬Q_3Q_0=Q_3 NOR ¬Q_0



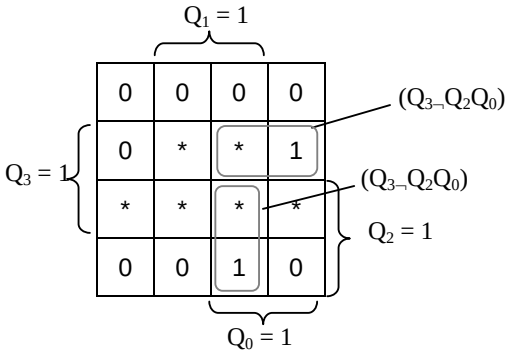
JK_2:



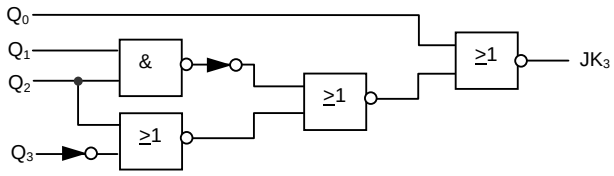
$JK_2=Q_1Q_0=\neg Q_1$ NOR $\neg Q_0$



JK_3:



$JK_3=Q_2Q_1Q_0+Q_3\neg Q_2Q_0 = (Q_2Q_1 +\neg Q_2Q_3)Q_0= (\neg(Q_2$ NAND Q1) NOR $(Q_2$ NOR $\neg Q_3)$) NOR $\neg Q_0$



5. Literatur

- Klaus Fricke: Digitaltechnik,
  Braunschweig/Wiesbaden: Vieweg, 2001
- K.H. Rohe/D.Kamke: Digitalelektronik,
  Stuttgart: B. G. Teubner, 1985
- F. Lüsch Lucchinetti: Digital Elektronik,
  Zürich: 2001
- Uni Hamburg, FB Informatik
  http://tech-www.informatik.uni-hamburg.de/applets/kvd/kvd.html
- http://www.bucephalus.org/bucware/toc.html
- http://www.digitalsimulator.de/