Dr. Jürg M. Stettbacher
Neugutstrasse 54
CH-8600 Dübendorf

Telefon: +41 43 299 57 23
E-Mail: dsp@stettbacher.ch

Digitaltechnik

Gatter und kombinatorische Schaltungen

Version 2.13
2017-09-18

Zusammenfassung: Es werden die wichtigsten Grundlagen der Digitaltechnik eingeführt. Dazu zählen verschiedene Gatter, Boolesche Algebra und das Aufstellen, sowie die Vereinfachung von kombinatorischen Schaltungen.

Inhaltsverzeichnis

nicht prüfungsrelevant, aber interessant

1 Zweck

In diesem Dokument geht es zuerst um Kombinatorik. Das heisst, dass die Ausgänge der betrachteten digitalen Ausdrücke und Schaltungen nur und unmittelbar von den Eingangssignalen abhängig sind, nicht aber von gespeicherten, inneren Zuständen. Flip-Flops[1] und sequenzielle Schaltungen, zum Beispiel Zustandsmaschinen, werden nur am Rand behandelt. Das Dokument verfolgt diese Ziele:

- Einführen der Booleschen Operatoren und der Booleschen Algebra, sowie von Gattern als Hilfsmittel für die grafische Darstellung von Booleschen Ausdrücken.
- Aufzeigen, wie man Boolesche Ausdrücke mit Hilfe der Regeln der Booleschen Algebra und mit Hilfe von Karnaugh Tafeln vereinfachen kann.
- Darstellen von einigen wichtigen Grundschaltungen, wie Addierer, Subtrahierer, Komparator und Decoder für BCD zu 7-Segment.
- Der Leser sollte nach der Lektüre in der Lage sein, die erwähnten Konzepte in praktischen Beispielen anwenden zu können. Die behandelten Beispielschaltungen sollte er kennen.

2 Einleitung

Die elementare Digitaltechnik basiert auf den sog. Gattern (englisch: Gates). Als Gatter bezeichnet man die Implementationen oder die grafischen Darstellungen von binären Operatoren, welche die Grundlage für die Boolesche[2] Algebra bilden. Man nennt die Gatter daher auch Boolesche Operatoren. Die Boolesche Algebra wird in Anlehnung an Gatterschaltungen auch als Schaltalgebra bezeichnet.

[1] Flip-Flops sind elementare Speicherelemente für je ein Bit.

[2] Benannt nach George Boole (1815 bis 1864), der sich mit mathematischer Logik befasste.

3 Boolesche Operatoren

- Grundlage für den Entwurf von digitalen Schaltungen.
- Variablen können nur die Werte 0 und 1 (falsch und wahr) annehmen.

NOT

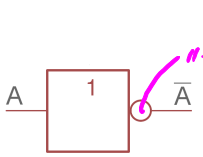
- Alternative Bezeichnungen: Negation, Nicht.
- Der NOT-Operator hat immer genau einen Eingang.
- Funktion:

$$\text{NOT}: \quad A \longrightarrow \text{NOT}(A) = \overline{A}$$

- Wahrheitstabelle[3]:

A	$\overline{A}$
0	1
1	0

- Der Ausgang des NOT-Operators ist das Inverse des Eingangs.
- Alternative Notation: $\text{NOT}(A) = \#A = !A = \neg A = \sim A = A'$
- Schaltsymbol[4] für NOT-Gatter:



[3] Die Wahrheitstabelle ist eine Möglichkeit, um Boolesche Operatoren oder Ausdrücke zu beschreiben. Die Tabelle weist zeilenweise allen möglichen Eingangskombinationen den entsprechenden Ausgangswert oder die Ausgangswerte zu.

[4] Es gibt verschiedene Normen für die Darstellung binärer Operatoren. Besonders verbreitet sind die amerikanischen ANSI-Symbole, sowie die eher in Europa benutzten IEC-Symbole. An Bedeutung verloren haben dagegen die alten DIN-Symbole. Wir verwenden hier die IEC-Symbole mit einem kleinen Kreis für die Inversion. Oft sieht man die Symbole auch mit einem kleinen Dreieck statt dem Kreis.

AND

- Alternative Bezeichnung: Konjunktion, Und.
- Der AND-Operator kann zwei oder mehr Eingänge haben.
- Funktion für zwei Eingänge:

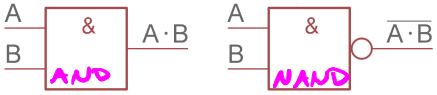
$$\text{AND}: \quad A, B \longrightarrow \text{AND}(A, B) = A \cdot B$$

$$\text{NAND}: \quad A, B \longrightarrow \text{NAND}(A, B) = \overline{A \cdot B}$$

- Wahrheitstabelle für zwei Eingänge:

A	B	$A \cdot B$	$\overline{A \cdot B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

- Der Ausgang des AND-Operators ist 1, wenn alle Eingänge 1 sind.
- Alternative Notation: $\text{AND}(A, B) = A \star B = A \,\&\, B = A \cap B = A \wedge B = AB$
- Schaltsymbole für AND- und NAND-Gatter:



OR

- Alternative Bezeichnung: Disjunktion, Oder.
- Der OR-Operator kann zwei oder mehr Eingänge haben.
- Funktion:

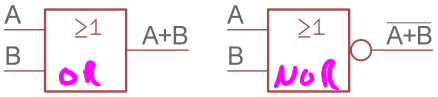
$$\text{OR}: \quad A, B \longrightarrow \text{OR}(A, B) = A + B$$

$$\text{NOR}: \quad A, B \longrightarrow \text{NOR}(A, B) = \overline{A + B}$$

- Wahrheitstabelle:

A	B	$A+B$	$\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

- Der Ausgang des OR-Operators ist 1, wenn mindestens ein Eingang 1 ist.
- Alternative Notation: $\mathrm{OR}(A,B) = A \cup B = A \vee B$
- Schaltsymbole für OR- und NOR-Gatter:



XOR

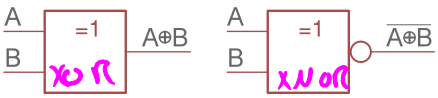
- Alternative Bezeichnung: Kontrajunktion, Exklusiv-Oder, Exklusiv-Or, EXOR.
- Der XOR-Operator hat immer genau zwei Eingänge.
- Funktion:

$$\begin{aligned} \mathrm{XOR}: \quad & A, B \longrightarrow \mathrm{XOR}(A,B) = A \oplus B \\ \mathrm{XNOR}: \quad & A, B \longrightarrow \mathrm{XNOR}(A,B) = \overline{A \oplus B} \end{aligned}$$

- Wahrheitstabelle:

A	B	$A \oplus B$	$\overline{A \oplus B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

- Der Ausgang des XOR-Operators ist 1, wenn genau ein Eingang 1 ist.
- Alternative Notation: $\mathrm{XOR}(A,B) = A \bar{\vee} B = A \underline{\vee} B = A \sqcup B = A \lesseqgtr B$
- Schaltsymbole für XOR- und XNOR-Gatter:



4 Boolesche Algebra

Es seien die binären Variablen $A, B, C \in \{0, 1\}$ gegeben. Die folgenden Sätze beziehen sich auf NOT, AND und OR. Beachte, dass einige Sätze nicht für XOR gelten, z. B. das Distributivgesetz.

Kommutativgesetz Vertauschen von Argumenten eines Operators:

$$\begin{aligned} A \cdot B &= B \cdot A \\ A + B &= B + A \end{aligned}$$

Assoziativgesetz Reihenfolge der Ausführung bei mehrfachen, identischen Operatoren:

$$\begin{aligned} (A \cdot B) \cdot C &= A \cdot (B \cdot C) \\ (A + B) + C &= A + (B + C) \end{aligned}$$

Distributivgesetz Auflösen von Klammern:

$$\begin{aligned} A \cdot (B + C) &= (A \cdot B) + (A \cdot C) = A \cdot B + A \cdot C \\ A + (B \cdot C) &= (A + B) \cdot (A + C) \end{aligned}$$

→ kann nicht weiter vereinfacht werden

Beachte, dass die zweite Form des Distributivgesetzes in unserer klassischen Algebra nicht gilt, wohl aber in der Booleschen Algebra. In Anlehnung an die klassische Algebra geben wir dem AND-Operator Priorität über das OR, so dass Klammern in der Regel nur um OR-Ausdrücke gesetzt werden müssen.

Neutrale Elemente Operanden ohne Einfluss:

$$\begin{aligned} 1 \cdot A &= A \\ 0 + A &= A \end{aligned}$$

Konstante Elemente Verknüpfungen mit konstantem Resultat:

$$\begin{aligned} A \cdot \overline{A} &= 0 \\ A + \overline{A} &= 1 \end{aligned}$$

Reduktionen Ausdrücke, die sich vereinfachen lassen:

$$\begin{aligned} A \cdot A &= A \\ A + A &= A \end{aligned}$$

$$\overline{\overline{A}} = A$$

$$\begin{aligned} A \cdot (A + B) &= A \\ A + (A \cdot B) &= A \end{aligned}$$

$$\begin{aligned} A \cdot \left(\overline{A} + B\right) &= A \cdot B \\ A + \left(\overline{A} \cdot B\right) &= A + B \end{aligned}$$

Gesetz von de Morgan[5] Umwandlung negierter Ausdrücke:

$$\begin{aligned} \overline{A \cdot B} &= \overline{A} + \overline{B} \\ \overline{A + B} &= \overline{A} \cdot \overline{B} \end{aligned}$$

Beweise Die Beweise der oben aufgeführten Sätze können über die Wahrheitstabelle geführt werden. Als Beispiel betrachten wir das Gesetz von de Morgan:

A	B	$A \cdot B$	$\overline{A \cdot B}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Es zeigt sich, dass die Spalten 4 und 7 identisch sind. Diese Spalten enthalten alle möglichen Kombination von A und B. Daher ist der Beweis vollständig.

Hier ist der Beweis für das etwas überraschende Distributivgesetz in der Form $A + (B \cdot C) = (A + B) \cdot (A + C)$, das in der klassischen Algebra nicht gültig ist:

[5] Benannt nach Augustus de Morgan (1806 bis 1871), einem englischen Mathematiker.

A	B	C	$B \cdot C$	$A + B \cdot C$	$A + B$	$A + C$	$(A + B) \cdot (A + C)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Auch hier sind alle $2^3 = 8$ Kombinationen der drei Variablen A, B, C berücksichtigt. Die Spalten 5 und 8 sind identisch. Der Beweis ist also erbracht.

Beispiel Mit Hilfe des Satzes von de Morgan können wir zeigen, dass man AND auch durch OR und NOT ausdrücken kann und umgekehrt:

$$\begin{aligned} \overline{A \cdot B} &= \overline{A} + \overline{B} \\ \overline{\overline{A \cdot B}} &= \overline{\overline{A} + \overline{B}} \\ A \cdot B &= \overline{\overline{A} + \overline{B}} \end{aligned}$$

$$\begin{aligned} \overline{A + B} &= \overline{A} \cdot \overline{B} \\ \overline{\overline{A + B}} &= \overline{\overline{A} \cdot \overline{B}} \\ A + B &= \overline{\overline{A} \cdot \overline{B}} \end{aligned}$$

Beispiel In vielen praktischen Anwendungen ist eine Aufgabe durch eine Wahrheitstabelle gegeben. Betrachte den folgenden Fall:

- Aussage A heisst: “Ein Auto fährt vorbei.”
- Aussage B heisst: “Die Ampel steht auf grün.”
- Aussage Y_1 lautet: “Ich kann sicher und korrekt über die Strasse gehen.”
- Der Wahrheitsgehalt der Aussage Y_1 soll nun aus den Aussagen A und B abgeleitet werden.
- Ist eine Aussage wahr, so setzen wir dafür 1 ein, ist sie falsch, so setzen wir dafür 0 ein. Damit können wir die folgende Wahrheitstabelle aufstellen:

A	B	Y_1
0	0	0
0	1	1
1	0	0
1	1	0

Ich kann also nur sicher und korrekt über die Strasse gehen, wenn die Ampel auf grün steht $(B = 1)$ und keine Auto vorbei fährt $(A = 0)$. Man kann nun daraus für Y_1 einen Booleschen Ausdruck bilden, der 1 ergibt. Dazu macht man eine AND-Verknüpfung der Aussagen A und B. Die wahren Aussagen (hier B) führt man direkt in die Verknüpfung, die falschen Aussagen (hier A) werden invertiert in der Verknüpfung verwendet, so dass sie ebenfalls wahr werden. Damit ergibt die AND-Verknüpfung eine 1, ist also wahr.

$$Y_1 = \overline{A} \cdot B$$

Zur Verifikation schreiben wir die Wahrheitstabelle dieses Ausdrucks nochmals auf:

A	$\overline{A}$	B	$Y_1 = \overline{A} \cdot B$
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0

Nun könnte man argumentieren, dass auch die folgende Funktion ein sicheres - wenn auch nicht korrektes - Überqueren der Strasse ermöglicht:

A	B	Y_2
0	0	1
0	1	1
1	0	0
1	1	0

Damit erhalten wir die neue Boolesche Funktion:

$$Y_2 = \overline{A} \cdot \overline{B} + \overline{A} \cdot B$$

In der Wahrheitstabelle gibt es zwei Zeilen, die je eine 1 ergeben. Diese Zeilen fassen wir mit OR zusammen. Das heisst: Entweder trifft eine Zeile zu oder die andere, oder beide. Man nennt das Zusammenfassen mehrerer Fälle durch OR die *disjunktive Normalform*. Es gäbe auch eine *konjunktive*

Normalform, auf die wir hier aber nicht eingehen. Den Booleschen Ausdruck für Y_2 können wir vereinfachen, indem wir die weiter oben aufgelisteten Regeln anwenden:

$$\begin{aligned} Y_2 &= \overline{A} \cdot \left(\overline{B} + B\right) \\ &= \overline{A} \cdot 1 \\ &= \overline{A} \end{aligned}$$

Das Ergebnis kann man leicht in der Wahrheitstabelle nachvollziehen. Y_2 ist das Inverse der Aussage A. Oder in Klartext: Ich kann über die Strasse gehen, wenn kein Auto vorbei fährt. Die Aussage B über die Ampel ist in diesem Beispiel belanglos.

5 Karnaugh Tafeln

Das Verfahren von Karnaugh[6] erleichtert in vielen (einfachen) Fällen das Reduzieren vom Booleschen Ausdrücken mit bis zu vier Variablen. Bei mehr als vier Variablen wird das Verfahren unübersichtlich.

Beispiel Wir bemühen nochmals den Fall von oben:

$$Y_2 = \overline{A} \cdot \overline{B} + \overline{A} \cdot B$$

Eine Karnaugh Tafel für die zwei Aussagen (resp. Variablen) A und B sieht folgendermassen aus. Es ist nichts Anderes als eine alternative Darstellung der Wahrheitstabelle. Dabei liegt jedes Feld im Schnittpunkt der Eingangsvariablen, die je wahr oder falsch sein können. Im vorliegenden Fall haben wir $2\,x\,2$ Felder. Das Feld oben links entspricht der Eingangskombinat $A = 1$ und $B = 1$. Das Feld rechts daneben entspricht der Kombination $\overline{A} = 1$, also $A = 0$ und $B = 1$.

	A	$\overline{A}$
B		
$\overline{B}$		

Nun werden die Werte der dazu gehörenden Wahrheitstabelle in die Tafel übernommen, wobei es reicht, die Werte $Y_2 = 1$ einzutragen. Ins Feld oben rechts $(\overline{A}, B)$ setzen wir jenen Wert für Y_2 aus der Wahrheitstabelle ein, der bei $A = 0$ und $B = 1$ gilt.

Y_2	A	$\overline{A}$
B		**1**
$\overline{B}$		**1**

[6] Maurice Karnaugh (gebohren 1924), Physiker und Pionier der Informatik.

In diesem Fall erkennt man sofort, dass die eingetragenen Einsen einen Block bilden, der genau $\overline{A}$ entspricht, aber unabhängig vom Zustand von B ist. Also ist $Y_2 = \overline{A}$. Bei der Anwendung von Karnaugh Tafeln geht es also immer darum, möglichst grosse Blöcke zu finden, welche durch möglichst wenige Variablen definiert sind. Dies muss wiederholt werden, bis alle Einsen in der Tafel mindestens einmal von einem Block erfasst wurden. Das Resultat ist dann eine disjunktive Normalform.

Beispiel Betrachte die folgende Wahrheitstabelle:

A	B	C	Y_3
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Die Karnaugh Tafel dazu sieht so aus:

Y_3	A	A	$\overline{A}$	$\overline{A}$
B	1	1	1	1
$\overline{B}$		1		
	C	$\overline{C}$	$\overline{C}$	C

Beachte, dass die Karnaugh Tafel so aufgebaut ist, dass alle Kombinationen der Wahrheitstabelle vorkommen. In der Tafel erkennen wir nun, dass es eine horizontale Linie von Einsen gibt, die $Y_3 = B$ entspricht. Zudem gibt es eine vertikale Linie, bestehend aus zwei Einsen bei $Y_3 = A\overline{C}$. Die obere dieser beiden Einsen ist beiden Linien gemeinsam. Das ist aber kein Problem. Nun haben wir zwei Ausdrücke, die zusammen alle Einsen in der Tafel umfassen. Damit können wir den entsprechenden disjunktiven, Booleschen Ausdruck notieren:

$$Y_3 = B + A\overline{C}$$

Alternativ zur Methode mit der Karnaugh Tafel hätte man den Booleschen Ausdruck auch in disjunktiver Normalform aus der Wahrheitstabelle lesen und anschliessend vereinfachen können:

$$Y_3 = \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\overline{C} + AB\overline{C} + ABC$$

Es folgt:

$$\begin{aligned} Y_3 &= \overline{A}B\left(\overline{C}+C\right)+A\overline{B}\overline{C}+AB(\overline{C}+C) \\ &= \overline{A}B+A\overline{B}\overline{C}+AB \\ &= B(\overline{A}+A)+A\overline{B}\overline{C} \\ &= B+A\overline{B}\overline{C} \\ &= B+A\overline{C} \end{aligned}$$

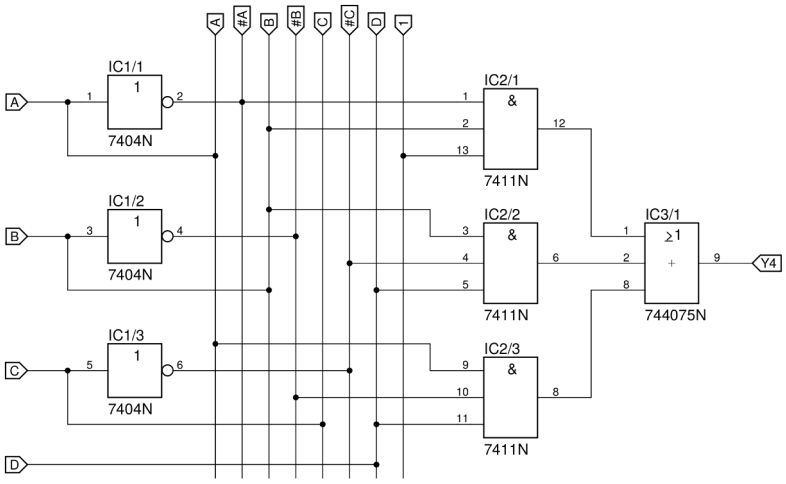
Beispiel Nun wollen wir uns überlegen, wie eine Karnaugh Tafel für vier binäre Variablen aussieht. Hier ist sie, wobei als Beispiel schon einige Einsen eingesetzt sind:

Y_4	A	A	$\overline{A}$	$\overline{A}$	
B		**1**	**1**	**1**	D
B			**1**	**1**	$\overline{D}$
$\overline{B}$					$\overline{D}$
$\overline{B}$	**1**	**1**			D
	C	$\overline{C}$	$\overline{C}$	C	

Wir führen die Vereinfachung durch: Zuerst gibt es einen Block an vier Einsen, der gegeben ist durch $Y_{4a} = \overline{A}B$. Direkt links davon steht eine einzelne Eins. Sie kann beschrieben werden als $Y_{4b} = AB\overline{C}D$. Besser ist es jedoch, wenn wir zusammen mit dem Nachbar rechts davon eine Zweiergruppe bilden mit $Y_{4c} = B\overline{C}D$. Dieser Ausdruck ist einfacher. Schliesslich haben wir noch eine Zweiergruppe links unten mit $Y_{4d} = A\overline{B}D$. Gesamthaft resultiert:

$$Y_4 = \overline{A}B + B\overline{C}D + A\overline{B}D$$

Dieser Ausdruck lässt sich nicht weiter reduzieren. Man könnte ihn höchstens noch etwas umformen, beispielsweise indem man noch einzelne Variablen ausklammert. Das wollen wir hier aber nicht tun. Statt dessen kann man Y_4 nun direkt in der angegebenen Form aus Logikgattern implementieren.



Die Schaltung ist von links nach rechts aufgebaut. Zuerst werden mit NOT-Gattern die benötigten inversen Signale zu A, B, C erzeugt. Es stehen dann alle Signale auf einem vertikalen Bus zur Verfügung. Diese Signale werden nun in die AND-Ebene geführt. Am Ende werden die Ausgänge der AND-Gatter in einem OR-Gatter zusammengefasst. Die Schaltung wurde in einem CAD-System für digitale Schaltungen gezeichnet. Die Abbildung enthält daher einige zusätzliche Angaben für die Implementation, nämlich den jeweiligen Typ der integrierten Schaltung[7], welche das Gatter enthält, die Pin-Nummern und weitere. Beachte, dass hier AND- und OR-Gatter mit je drei Eingängen verwendet wurden.

Wir betrachten nochmals die Eins links neben dem 4er-Block in der Karnaugh Tafel. Statt diese Eins mit dem Nachbarn rechts davon zu verbinden, hätte man sie auch mit der Eins in der selben Spalte ganz unten verbinden können, sozusagen hinten herum. Das hätte zum Term $Y_{4e} = A\overline{C}D$ geführt und das Resultat hätte so ausgeschaut:

$$Y_4 = \overline{A}B + A\overline{C}D + A\overline{B}D$$

Die Komplexität dieser Lösung ist aber die gleiche wie in der zuerst beschriebenen Lösung. Es gibt aber durchaus Fälle, wo die Verbindung über den Tabellenrand hinweg Vorteile hat, zum Beispiel hier:

Y_5	A	A	$\overline{A}$	$\overline{A}$
B	1			1
$\overline{B}$	1			1
	C	$\overline{C}$	$\overline{C}$	C

[7] Integrierte Schaltungen (IC, Integrated Circuit) kauft man in der Form von kleinen schwarzen Käfern mit Metallfüssen (Pins), welche mit den internen Gattern verbunden sind. Zum Beispiel hat eine integrierte Schaltung vom Typ 7404 genau 14 Füsse und beinhaltet sechs NOT-Gatter.

Bei diesem Beispiel kann man 2er-Blöcke bilden: $Y_{5a} = AC$ und $Y_{5b} = \overline{A}C$. Damit folgt:

$$Y_5 = AC + \overline{A}C$$

Dieser Ausdruck lässt sich aber leicht vereinfachen:

$$Y_5 = AC + \overline{A}C = C\left(A + \overline{A}\right) = C$$

Dies zeigt uns, dass die 2er-Blöcke nicht ideal gewählt waren. Bei genauerem Hinsehen, und wenn wir uns die Karnaugh Tafel vertikal aufgerollt vorstellen, erkennen wir, dass wir auch einen 4er-Block mit $Y_5 = C$ hätten bilden können. Das wäre die bessere Lösung gewesen.

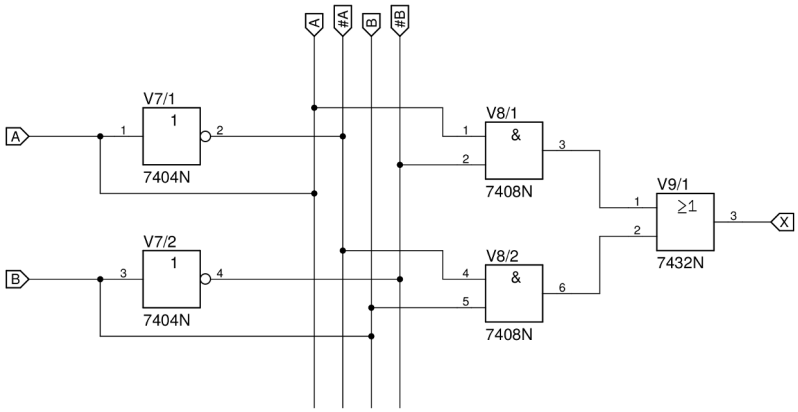
Beispiel Wir wollen noch zeigen, dass sich ein XOR auch durch AND, OR und NOT synthetisieren lässt. Wir beginnen gleich mit der Karnaugh Tafel für XOR:

X	A	$\overline{A}$
B		**1**
$\overline{B}$	**1**	

Es folgt der Ausdruck:

$$X = A\overline{B} + \overline{A}B$$

Er lässt sich nicht weiter vereinfachen. Die Gatter-Schaltungen davon sieht so aus:



6 Addierer

Als weiteres Beispiel wollen wir untersuchen, wie man einen binären Addierer für N Bit implementiert. Der Addierer soll nach dem bekannten Schema funktionieren:

A			a_N	...	a_2	a_1	a_0	$_b$
B	+		b_N	...	b_2	b_1	b_0	$_b$
C		c_N	c_{N-1}	...	c_1	c_0		
S			s_N	...	s_2	s_1	s_0	$_b$

Dabei sind A und B die Summanden, S ist die Summe und C bezeichnet den Übertrag (englisch: Carry) aus der jeweils voran gegangenen Spalte.

Halbaddierer Als Halbaddierer bezeichnet man einen Addierer für nur ein Bit, also $0+0=0$, $0+1=1$, $1+0=1$ und $1+1=10$. Der letzte Fall ist speziell, denn das Resultat ist nicht mehr nur ein Bit. Daher sagen wir, das Resultat-Bit sei null, es gibt aber zusätzlich einen Übertrag.

Den Halbaddierer verwenden wir für das Bit Nummer 0 im N-Bit Addierer. Damit folgt die Wahrheitstabelle des Halbaddierers mit den Eingängen a_0 und b_0:

a_0	b_0	s_0	c_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Aus der Wahrheitstabelle können wir die Bildungsregel für s_0 und c_0 sofort angeben:

$$\begin{aligned} s_0 &= a_0 \oplus b_0 \\ c_0 &= a_0 \cdot b_0 \end{aligned}$$

Für die Implementation des Halbaddierers benötigt man also nur ein XOR- und ein AND-Gatter.

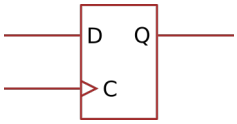
Volladdierer Der Volladdierer ist wiederum ein Addierer für ein Bit, er kann aber den Übertrag aus der vorher gehenden Stufe mit addieren. Wir verwenden demnach Volladdierer für alle Bits $0 < k \leq N$ im N-Bit Addierer. Es folgt die Wahrheitstabelle für das Bit Nummer k:

a_k	b_k	c_{k-1}	s_k	c_k
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Mit den oben präsentierten Werkzeugen liesse sich daraus nun eine Gatterschaltung für einen kompletten Addierer mit mehreren Bits ableiten.

7 Flip-Flops

Ein Flip-Flop ist ein Speicher, der ein Bit speichern, resp. festhalten kann. Es gibt verschiedene Arten von derartigen Flip-Flops. Wir behandeln hier nur das besonders wichtige flankengesteuerte D-Flip-Flop[8]. Sein Schaltsymbol sieht so aus:



Der Anschluss D ist der Dateneingang, Q der Datenausgang und C ist der Clock-Eingang. Bei jeder ansteigenden Flanke am Clock-Eingang C übernimmt das Flip-Flop den Zustand vom Eingang D und hält ihn am Ausgang Q fest, bis der Clock eine erneute ansteigende Flanke erhält. Das Verhalten lässt sich in folgender Weise in einer Wahrheitstabelle beschreiben.

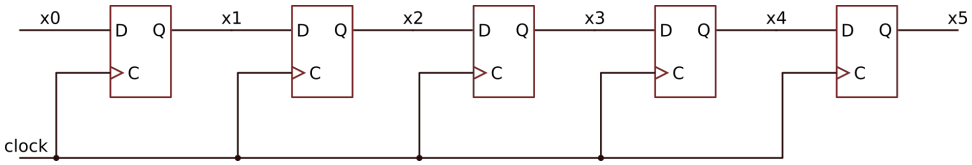
D	C	Q_{n+1}
X	0	Q_n
X	1	Q_n
X	⎺_	Q_n
0	_/⎺	0
1	_/⎺	1

In dieser Wahrheitstabelle bedeutet D = X, dass der Eingang D keine Rolle spielt. Wenn $Q_{n+1} = Q_n$ angegeben ist, so meint das, dass sich der Ausgang nicht ändert. Es bezeichnet n den Zeitpunkt vor und $n+1$ nach der Beobachtung, resp. nach der Clock-Flanke. Der Ausgang bleibt unverändert, wenn C eine konstante 0 oder 1 ist und ebenso bei einer abfallenden Flanke. Nur während der ansteigenden Flanken wird der Zustand von D auf den Ausgang Q_{n+1} übernommen. Dieser Vorgang dauert wenige Nanosekunden.

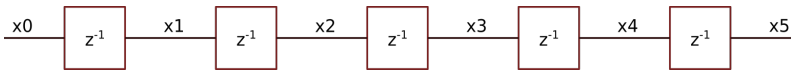
Mit Hilfe von Flip-Flops lassen sich sequenzielle Schaltungen bauen, also solche, die einen gewissen Ablauf erzeugen. Flip-Flops sind damit unter anderem die Grundbausteine für Mikroprozessoren und Zustandsmaschinen[9]. Als Beispiel betrachten wir ein Schieberegister.

[8] Das D in der Bezeichnung des D-Flip-Flops steht für *Delay*.

[9] Eine Zustandsmaschine ist eine digitale Schaltung, die einen gewissen zeitlichen Ablauf an ihren Ausgängen erzeugt. Ein typisches Beispiel davon wäre etwa ein Binärzähler oder eine einfache Ampelsteuerung.



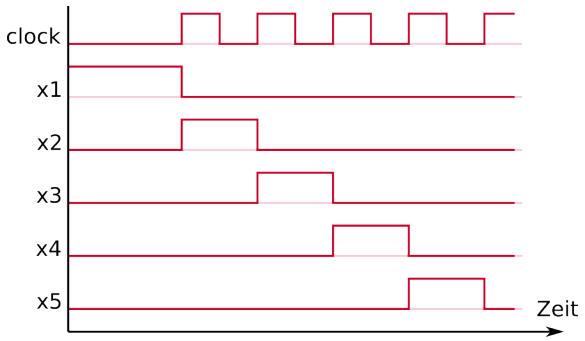
In vielen Fällen wird das Schieberegister jedoch vereinfacht gezeichnet. Die Flip-Flops werden reduziert zu Verzögerungselementen, die mit z^{-1} oder einfach mit T beschriftet sind[10]. Der gemeinsame Clock wird nicht mehr dargestellt.



Ist nun das Schieberegister beispielsweise mit x1 = 1 und x2 = x3 = x4 = x5 = 0 initialisiert und sei ausserdem x0 fix auf 0 verbunden, so kann der Verlauf wiederum in einer Wahrheitstabelle dargestellt werden.

clock	x0	x1	x2	x3	x4	x5
0	0	1	0	0	0	0
⎍	0	0	1	0	0	0
⎍	0	0	0	1	0	0
⎍	0	0	0	0	1	0
⎍	0	0	0	0	0	1
⎍	0	0	0	0	0	0

Man erkennt leicht, dass die 1 von x1 nach jeder Clock-Flanke um eine Stelle weiter geschoben wird und dass die Nullen von x0 ständig nachrücken, bis das gesamte Schiebregister nur noch Nullen enthält. Wäre nun der Eingang x0 mit dem Ausgang x5 verbunden (statt mit 0), so würde die initiale 1 fortwährend im Kreis herum rotiert. Zum Schluss wollen wir noch eine alternative Darstellung des Verlaufs, das sogenannte Zeitverlaufsdiagramm zeigen.



[10] Bei dieser Notation ist z^{-1} der Verzögerungsoperator in einem zeitdiskreten System mit fixem Abtastinterval T.

8 Übungsaufgaben

Die folgenden Aufgaben dienen der Lernkontrolle.

8.1 Boolsche Algebra

Vereinfachen sie die folgenden Booleschen Ausdrücke:

(a) $Q_1 = R + (R+S) \cdot (R+T)$

(b) $Q_2 = R + \overline{R+S}$

8.2 Wahrheitstabelle

Ein Lift hat Etagen-Tasten in der Kabine (interne Tasten) und je eine Ruf-Taste auf jeder Etage (externe Tasten). Zudem ist im Boden der Kabine ein Personensensor angebracht. Eine vorverarbeitende Logik stellt die folgenden logischen Signal zur Verfügung:

P	Mindestens eine Person ist in der Kabine.
T_{int}	Mindestens eine interne Taste wurde gedrückt.
T_{ext}	Mindestens eine externe Taste wurde gedrückt.

Daraus sollen die zwei logischen Signale F_1 und F_2 erzeugt werden, die angeben, ob auf Grund eines Tastendrucks in der Kabine gefahren werden soll, oder auf Grund einer Ruf-Taste auf einer Etage.

F_1	Es wird auf Grund einer internen Taste gefahren. Das Signal ist wahr, wenn sich eine Person in der Kabine befindet und wenn eine interne Taste gedrückt wurde.
F_2	Es wird auf Grund einer externen Taste gefahren. Das Signal ist wahr, wenn sich keine Person in der Kabine befindet und wenn eine externe Taste gedrückt wurde.

Stellen Sie eine Wahrheitstabelle für die logischen Signale F_1 und F_2 auf.

8.3 Disjunktive Normalform

Stellen Sie ausgehend von der folgenden Wahrheitstabelle die disjunktive Normalform für Z als Funktion der Eingänge A, B und C auf. Vereinfachen Sie anschliessend den Ausdruck mit Hilfe der Booleschen Algebra.

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

8.4 Karnaugh Tafel

Zeichnen Sie für die Wahrheitstabelle aus Aufgabe 8.3 die Karnaugh Tafel. Lesen Sie daraus den vereinfachten Ausdruck für Z ab.

8.5 Gatter-Schaltung

Skizzieren Sie den vereinfachten Ausdruck für Z aus den Aufgaben 8.3 und 8.4 als Gatter-Schaltung.

8.6 Subtrahierer

Analog zum Addierer (siehe Kapitel 6) lässt sich ein Subtrahierer für N Bit durch einen Halb- und $N-1$ Vollsubtrahierer realisieren. Entwickeln Sie die Wahrheitstabellen, sowie die Gatterschaltungen für Halb- und Vollsubtrahierer.

8.7 Komparator

Ein Komparator vergleicht zwei binäre Zahlen, die aus einem oder aus mehreren Bits bestehen können. Für die Zahlen A und B mit nur je einem Bit sieht die Wahrheitstabelle so aus:

A	B	$A = B$	$A < B$	$A > B$
0	0	1		
0	1	0		
1	0	0		
1	1	1		

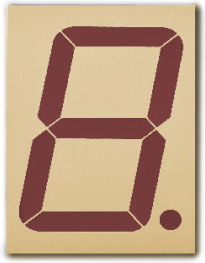
(a) Vervollständigen Sie die Wahrheitstabelle.

(b) Bilden Sie vereinfachte Boolesche Ausdrücke für $A = B$, $A < B$ und $A > B$.

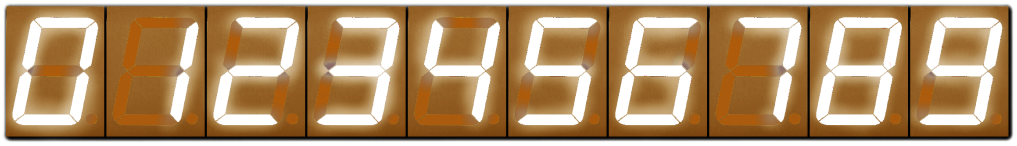
(c) Skizzieren Sie die Gatter-Schaltungen für $A = B$, $A < B$ und $A > B$.

8.8 BCD zu 7-Segment Decoder

Oft verwendet man für die Anzeige von Ziffern sog. 7-Segment Anzeigen. Jedes der sieben Segmente[11] ist eine Leuchtdiode (LED), die idividuell ein- oder ausgeschaltet werden kann.



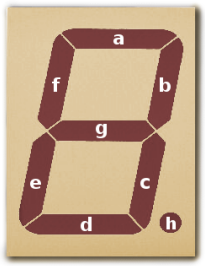
Die Ziffern von 0 bis 9 lassen sich darstellen, indem jeweils eine bestimmte Auswahl aus den sieben möglichen Segmenten eingeschaltet wird.



Ein BCD zu 7-Segment Decoder ist ein Logikblock, der dafür sorgt, dass für jede BCD-Ziffer auf der 7-Segment Anzeige jeweils die richtigen LEDs eingeschaltet sind, so dass die entsprchende Ziffer dargestellt wird. Anders ausgedrückt: Der Decoder hat für jedes Segment eine Logik, die entscheidet bei welcher BCD-Zahl das Segment eingeschaltet sein muss.

Die Aufgabe besteht nun darin, für das Segment *a* diese Logik zu entwerfen.

[11] Wenn man den Dezimalpunkt mit einrechnet, so sind es acht Segmente.



Gehen Sie so vor:

- Stellen Sie für das Segment *a* die Wahrheitstabelle auf.
- Stellen Sie die Karnaugh Tafel auf.
- Extrahieren Sie den Booleschen Ausdruck für das Segment *a*.
- Entwerfen Sie eine entsprechende Gatter-Schaltung.